9.3

IBM MQ 用のアプリケーションの開発

注記

本書および本書で紹介する製品をご使用になる前に、1303 ページの『特記事項』に記載されている情報をお読みください。

本書は、 IBM® MQ バージョン 9 リリース 3、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様が IBM に情報を送信する場合、お客様は IBM に対し、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で情報を使用または配布する非独占的な権利を付与します。

目次

IBM MQ 用アプリケーションの開発

メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

IBM MQ のアプリケーション開発を初めて使用する場合

IBM MQ のアプリケーション開発について詳しくは、IBM Developer をご確認ください。

- IBM MQ Developer Essentials (基本を学習し、デモを実行し、アプリをコーディングし、より高度なチュートリアルを受講します)
- IBM MQ Downloads for Developers (無料の開発者用エディションと試用版を含む)

以下のセクションで説明している概念をよく理解しておくと、アプリケーションを開発しやすくなる可能性があります。

- 7 ページの『アプリケーション開発の概念』
- 49 ページの『IBM MQ アプリケーションの設計上の考慮事項』

オブジェクト指向の言語およびフレームワークのサポート

IBM MQ は、以下の言語およびフレームワークで開発されたアプリケーションのコア・サポートを提供します。

- JMS
- Java
- C++
- .NET

15 ページの『オブジェクト指向のアプリケーション』も参照してください。

.NET は、多くの言語で開発されたアプリケーションをサポートします。 IBM MQ classes for .NET を使用して IBM MQ キューにアクセスする方法を説明するために、MQ 製品資料には以下の言語の情報を記載しています。

- C# のサンプル・コードとサンプル・アプリケーション
- C++ のサンプル・アプリケーション
- Visual Basic サンプル・アプリケーション

578 ページの『IBM MQ .NET プログラムの作成およびデプロイ』 を参照してください。

IBM MQ は、IBM MQ 9.1.1 以降の Windows 環境のアプリケーション、および IBM MQ 9.1.2 以降の Linux® 環境のアプリケーションに対して .NET Core をサポートします。 詳しくは、554 ページの『IBM MQ classes for .NET のインストール』を参照してください。

ALW IBM MQ は、OASIS AMQP 1.0 プロトコルを実装する AMQP クライアントもサポートします。

MQ Light、 Apache Qpid クライアント (Apache Qpid Proton や Apache Qpid JMS API など) は、このプロトコルに基づいています。

MQ Light API は、 IBM MQ Light で使用できます。

Apache Qpid クライアントは、 QPid プロトンで入手できます。

以下の言語バインディングがそのまま提供されます。

- Go バインディング
- Node.js アプリケーションと連携する JavaScript API 実装

プログラマチック REST API のサポート

IBM MQ は、メッセージを送受信するために次のプログラマチック REST API のサポートを提供します。

- IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM DataPower® ゲートウェイ

703 ページの『IBM MQ での REST アプリケーションの開発』を参照し、IBM Developer の IBM MQ エリアにあるチュートリアル Get started with the IBM MQ messaging REST API を参照してください。 このチュートリアルには、IBM MQ messaging REST API でそのまま使用できる、以下の言語のサンプルが含まれています。

- MQ メッセージング REST API を使用する Go のサンプル
- HTTPS モジュールを使用する Node.js のサンプル
- Promise モジュールを使用する Node.js のサンプル

プロシージャー型プログラミング言語のサポート

IBM MQ は、以下のプロシージャー型プログラミング言語で開発されたアプリケーションをサポートします。

- C
- Windows Visual Basic (Windows システムのみ)
- COBOL
- z/OS アセンブラー (IBM MQ for z/OS のみ)
- z/OS PL/I (IBM MQ for z/OS のみ)
- IBM i RPG (IBM MQ for IBM i のみ)

これらの言語は、メッセージ・キュー・インターフェース (MQI) を使用して、メッセージ・キューイング・サービスにアクセスします。 717 ページの『IBM MQ での MQI アプリケーションの開発』を参照してください。 オブジェクト指向の言語およびフレームワークで使用される IBM MQ オブジェクト・モデルには、MQI を使用するプロシージャー型言語では利用できない追加の機能が用意されていることに注意してください。

アプリケーション名の指定

ALW

IBM MQ 9.1.2 の前に、Java または JMS クライアント・アプリケーションでアプリケーション名を指定することができます。 IBM MQ 9.1.2 から、追加のプログラミング言語でアプリケーション名を指定することもできます。 詳しくは、52 ページの『サポートされるプログラミング言語でのアプリケーション名の指定』を参照してください。

関連タスク

1247 ページの『MQ Telemetry 用アプリケーションの開発』

1272 ページの『IBM MQ を使用した Microsoft Windows Communication Foundation アプリケーションの開発』

IBM MQ 用の Microsoft Windows Communication Foundation (WCF) カスタム・チャネルは、WCF クライアントとサービスの間でメッセージを送受信します。

関連資料

1225 ページの『Managed File Transfer 用アプリケーションの開発』

Managed File Transfer で実行するプログラムを指定し、 Managed File Transfer で Apache Ant を使用し、ユーザー出口で Managed File Transfer をカスタマイズし、エージェント・コマンド・キューにメッセージを書き込むことによって Managed File Transfer を制御します。

アプリケーション開発の概念

選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

IBM MQ 用に作成できるアプリケーションの種類については、5 ページの『IBM MQ 用アプリケーションの開発』および 7 ページの『アプリケーションが実行可能なアクション』を参照してください。

関連概念

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

アプリケーションが実行可能なアクション

ビジネス・プロセスをサポートするために必要なメッセージを送受信するアプリケーションを開発できます。 キュー・マネージャーや関連リソースを管理するためのアプリケーションを開発することもできます。

IBM MQ for Multiplatforms でアプリケーションが実行可能なアクション

Multi

マルチプラットフォーム では、以下のアクションを実行するアプリケーションを作成できます。

- 同じオペレーティング・システム下で実行中の他のアプリケーションにメッセージを送信する。 アプリケーションは、同じシステム上と別のシステム上のどちらにあっても構いません。
- 他の IBM MQ プラットフォーム上で実行されるアプリケーションにメッセージを送信する。
- 次のシステムで CICS® からのメッセージ・キューイングを使用する。
  - AIX TXSeries®の AIX®
  - IBM i IBM i
  - Windows Windows
- 次のシステムで Encina からのメッセージ・キューイングを使用する。
  - AIX AIX
  - Windows Windows
- 次のシステムで Tuxedo からのメッセージ・キューイングを使用する。
  - AIX AIX
  - AT&T
  - Windows Windows
- IBM MQ をトランザクション・マネージャーとして使用し、IBM MQ の作業単位で外部リソース・マネージャーによって行われた更新を調整する。 以下の外部リソース管理プログラムがサポートされており、X/Open XA インターフェースに準拠しています。
  - Db2®
  - Informix®
  - Oracle
  - Sybase

- いくつかのメッセージを、コミットまたはバックアウトすることのできる 1 つの作業単位として、まとめて処理する。
- 全機能を使用できる IBM MQ 環境、または IBM MQ クライアント環境から実行する。

IBM MQ for z/OS でアプリケーションが実行可能なアクション

z/OS

z/OS では、以下のアクションを実行するアプリケーションを作成できます。

- CICS または IMS 内からメッセージ・キューイングを使用する。
- 各機能に最適の環境を選択して、バッチ、CICS、および IMS の各アプリケーション間でメッセージを送信する。
- 他の IBM MQ プラットフォーム上で実行されるアプリケーションにメッセージを送信する。
- いくつかのメッセージを、コミットまたはバックアウトすることのできる 1 つの作業単位として、まとめて処理する。
- IMS ブリッジによって IMS アプリケーションにメッセージを送信し、対話する。
- RRS によって調整された作業単位に関係する。

z/OS 内の各環境には、それぞれ固有の特性、長所、および短所があります。 IBM MQ for z/OS の長所は、アプリケーションを 1 つの環境に固定せずに、分散して各環境の利点を活用できることです。 例えば、TSO または CICS を使用してエンド・ユーザー・インターフェースを開発したり、z/OS バッチで処理集中型のモジュールを実行したり、IMS または CICS でデータベース・アプリケーションを実行したりすることができます。 どの場合も、アプリケーションの種々の部分がメッセージおよびキューを用いて通信することができます。

IBM MQ アプリケーションの設計担当者は、これらの環境による違いや制約を意識する必要があります。以下に例を示します。

- IBM MQ はキュー・マネージャー間の相互通信機能を提供する (これは分散キューイング と呼ばれる)。
- 変更に対するコミットやバックアウトの方法はバッチ環境と CICS 環境で異なる。
- IBM MQ for z/OS は、IMS 環境で、オンラインのメッセージ処理プログラム (MPP)、対話式ファスト・パス・プログラム (IFP)、およびバッチ・メッセージ処理プログラム (BMP) をサポートする。 バッチ DL/I プログラムを作成する場合は、1029 ページの『z/OS バッチ・アプリケーションの構築』や 732 ページの『z/OS バッチの考慮事項』などのトピックで、z/OS バッチ・プログラムに関する指針を確認してください。
- IBM MQ for z/OS の複数のインスタンスが単一の z/OS システム上に存在できますが、CICS 領域は一度に 1 つのキュー・マネージャーにしか接続できません。 ただし、複数の CICS 領域を同一のキュー・マネージャーに接続することはできます。 IMS および z/OS バッチ環境では、プログラムは複数のキュー・マネージャーに接続することができます。
- IBM MQ for z/OS を使用すると、キュー・マネージャーのグループがローカル・キューを共用できるようになり、スループットおよび可用性が改善されます。 このようなキューを共用キュー と言います。そして、キュー・マネージャーはキュー共用グループ を形成し、同じ共用キュー上のメッセージを処理することができます。 バッチ・アプリケーションは、特定のキュー・マネージャー名ではなく、キュー共用グループ名を指定することにより、キュー共用グループに含まれるキュー・マネージャーのうちの 1 つに接続できます。 これをグループ・バッチ接続、またはもっと簡単にグループ接続 と言います。 共用キューおよびキュー共用グループを参照してください。

z/OS サポートされているそれぞれの環境の違いや制約の詳細については、893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』を参照してください。

関連概念

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』

プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

721 ページの『プロシージャー型キューイング・アプリケーションの作成』
この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

918 ページの『プロシージャー型クライアント・アプリケーションの作成』
IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

347 ページの『IBM MQ classes for Java の使用』
Java 環境で IBM MQ を使用します。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

553 ページの『.NET アプリケーションの開発』
IBM MQ classes for .NET を使用すると、.NET アプリケーションは、IBM MQ MQI client として IBM MQ に接続することも、 IBM MQ サーバーに直接接続することもできます。

525 ページの『C++ アプリケーションの開発』
IBM MQ では、IBM MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。

1005 ページの『プロシージャー型アプリケーションの構築』
IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

関連タスク

1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』
これらのサンプル・プログラムは、プロシージャー型言語で作成されており、Message Queue Interface (MQI) の標準的な使用法を示しています。 異なるプラットフォーム上の IBM MQ プログラム。

1272 ページの『IBM MQ を使用した Microsoft Windows Communication Foundation アプリケーションの開発』
IBM MQ 用の Microsoft Windows Communication Foundation (WCF) カスタム・チャネルは、WCF クライアントとサービスの間でメッセージを送受信します。

セキュリティー

Multi アプリケーション、アプリケーション名、およびアプリケーション・インスタンス

アプリケーションの設計と記述を開始する前に、アプリケーションの基本概念、アプリケーション名、およびアプリケーション・インスタンスについて理解しておいてください。

アプリケーション

Multi

キュー・マネージャーへの複数の接続は、それらが同じアプリケーション名 である場合、同じアプリケーション からのものと見なされます。 アプリケーション名は、DISPLAY CONN(*) TYPE CONN コマンドの APPLTAG 属性として表示されます。

注 :

1. IBM MQ 9.1.2 より前のバージョンの IBM MQ client を使用するアプリケーションの場合、アプリケーション名は IBM MQ client によって自動的に設定されます。 その値は、アプリケーションのプログラミン

グ言語とアプリケーションを実行しているプラットフォームによって決まります。 詳しくは、PutApplName を参照してください。

2. IBM MQ 9.1.2 以降の IBM MQ client を使用する IBM MQ client アプリケーションの場合、アプリケーション名を特定の値に設定することができます。 ほとんどの場合、アプリケーション・コードの変更やアプリケーションの再コンパイルは不要です。 詳細については、54 ページの『サポートされるプログラミング言語でのアプリケーション名の使用方法』 を参照してください。

アプリケーション・インスタンス

接続はさらにアプリケーション・インスタンス に分割されます。 アプリケーションのインスタンスは、そのアプリケーションに対して 1 つの「実行単位」を提供する、密接に関連した接続のセットです。 通常これは、単一のオペレーティング・システム・プロセスで、複数のスレッドおよび関連する IBM MQ 接続を持つことができます。

IBM MQ for Multiplatforms では、アプリケーション・インスタンスは特定の接続タグに関連付けられます。キュー・マネージャーは、関連付けられていることを確認できると、新規接続を既存のアプリケーション・インスタンスと自動的に関連付けます。

注 :

- クライアント接続を使用している場合、これらのプロセスは 1 つ以上の実行中のチャネルを介してキュー・マネージャーに接続する可能性があります。
- JMS アプリケーションでは、アプリケーション・インスタンスは特定の JMS 接続および関連するすべての JMS セッションにマップされます。

均一クラスターの 自動アプリケーション・バランシングを使用する場合、 IBM MQ for Multiplatforms ではアプリケーション・インスタンスが特に重要です。 IBM MQ for Multiplatforms プラットフォームでは、DISPLAY APSTATUS コマンドを使用して、現在接続されているアプリケーション・インスタンスを表示できます。

キュー・マネージャーは、以下の場合は特に、アプリケーション・インスタンスの関連付けへの接続を正しく実行できない可能性があります。

- 複数のアプリケーション名を使用して、同じプロセスからの共有会話で複数の接続を確立する場合。
- 古いレベルのクライアント・ライブラリーが使用されている場合。 例えば、 IBM MQ JMS クライアントは IBM MQ 9.1.2 以前でインストールされます。

このような状況では、アプリケーション自体が再接続可能として定義されていない場合に、これは許可されますが、一部のアプリケーション・インスタンスのグループ化が正しくなくなる可能性があります。 いずれかの接続が MQCNO_RECONNECT として宣言されている場合、これはアプリケーション・バランシングに大きく影響します。したがって、MQCONN 呼び出しは MQCNO_RECONNECT_INCOMPATIBLE で拒否されます。

関連概念

52 ページの『サポートされるプログラミング言語でのアプリケーション名の指定』
IBM MQ 9.2.0 より前では、 Java または JMS クライアント・アプリケーションで既にアプリケーション名を指定している場合があります。 IBM MQ 9.2.0 以降、この機能は IBM MQ for Multiplatforms 上の他のプログラミング言語にまで拡張されています。

MQI を使用するアプリケーション・プログラム

IBM MQ のアプリケーション・プログラムを正常に実行するためには、一定のオブジェクトが必要です。

11 ページの図 1 は、キューからメッセージを取り出し、そのメッセージを処理し、その後同じキュー・マネージャー上の別のキューにいくつかの結果を送信するアプリケーションを示しています。

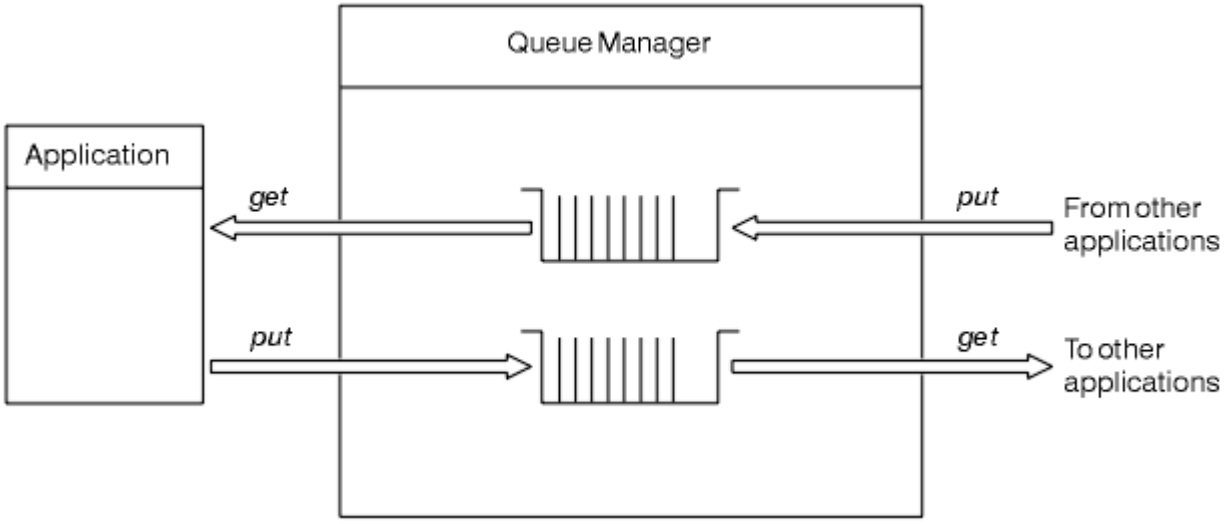


図 1. キュー、メッセージ、およびアプリケーション

アプリケーションは、ローカル・キューまたはリモート・キューにメッセージを (MQPUT を使用して) 書き込むことができますが、ローカル・キューからしかメッセージを (MQGET を使用して) 直接読み取れません。

このアプリケーションが実行されるには、次の条件が満たされている必要があります。

- キュー・マネージャーが存在しており、実行されている。
- 最初のアプリケーション・キュー (ここからメッセージが取り出される) が定義されている。
- 2 番目のキュー (アプリケーションはここにメッセージを書き込む) も定義されている。
- アプリケーションが、キュー・マネージャーに接続可能である。 このためには、アプリケーションは IBM MQ にリンクされている必要があります。 1005 ページの『プロシージャー型アプリケーションの構築』を参照してください。
- 最初のキューにメッセージを書き込むアプリケーションも、キュー・マネージャーに接続する。 また、それらのアプリケーションがリモートである場合も、伝送キューとチャネルは共にセットアップされている。 11 ページの図 1 には、システムのこの部分は示されていません。

クライアント接続を使用した複数の IBM MQ キュー・マネージャーへの接続

複数のキュー・マネージャーに接続するように、クライアント接続アプリケーションを構成することができます (ロード・バランシングまたはサービス可用性の理由から)。

IBM MQ クライアントでこれを実現するための基本メカニズムは、クライアント・チャネル定義テーブルの使用です。 クライアント・チャネル定義テーブルの構成、または接続リストを参照してください。

外部ロード・バランシング製品を使用するか、ホスト名または IP アドレスをリダイレクトできる「スタブ」で IBM MQ 接続コードをラップすることで、同様の動作を実現することもできます。

これらの各手法にはいくつかの制限があり、特定のアプリケーション要件に対して多かれ少なかれ適している場合があります。 以下のセクションでは、すべてを網羅しているわけではありませんが、考慮すべき特定の側面、およびこれらのさまざまなアプローチがこれらの側面に与える影響について説明します。

IBM MQ 均一クラスターについては、 均一クラスターについてを参照してください。複数の宛先を提供するために、CCDT の基本メカニズムに基づいて、複数のキュー・マネージャーにまたがるアプリケーションの水平スケーリングを実現する強力なメカニズムが用意されています。 均一クラスターは、基礎となる IBM MQ プロトコルを認識しない外部ロード・バランサーを使用して、可能な範囲を超える機能を提供することができます。そのため、以下で説明する問題の一部は回避できます。そのため、該当する場合は、他の技法よりも均一クラスターを優先して使用することを検討してください。

重要 : ロード・バランシング・テクノロジーを使用してキュー・マネージャーに接続する IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging(いずれかの IBM MQ リソース・アダプターを使用するものを含む) を使用するアプリケーションは、注意して使用する必要があります。問題が発生した場合は、ロード・バランシングを使用せずにそれらの問題を再現してください。

複数の問題が関係しています。これらの問題はすべて、そのような接続には最も問題があり、最悪の場合はまったく信頼性がないことを意味します。

- 任意の形式のロード・バランシングを使用してキュー・マネージャーに複数の接続を行うアプリケーションを接続する場合は、特に注意が必要です。 これには、IBM MQ Classes for JMS/Jakarta Messaging を使用するすべてのアプリケーションが含まれます。これらのアプリケーションは、一般的に複数の IBM MQ 接続を作成するためです。 外部ロード・バランサーまたはカスタム・コード・スタブを使用する場合は、常に同じアプリケーション・インスタンスから同じキュー・マネージャーに接続を経路指定する必要があります。
- XA トランザクション管理または JTA (Java Transaction API) の使用は、同じキュー・マネージャーに一貫して接続できるかどうかに依存しています。実際には、これはどのような形式のロード・バランシングでも現実的ではありません。
- -均一クラスター管理は、干渉を受けずに特定のキュー・マネージャーに再接続するようにクライアントに指示できることに依存しています。 外部ロード・バランシングと均一クラスターの使用を組み合わせることはお勧めできません。

外部ロード・バランシング・テクノロジーではなく、複数のキュー・マネージャーにわたるアプリケーションの水平スケーリングを実現するには、 IBM MQ 均一クラスター機能を使用する必要があります。 均一クラスターの作成方法や使用方法など、均一クラスターについては、均一クラスターの構成および以下のトピックを参照してください。

本書で使用される用語

CCDT 複数 QMGR
グループ (つまりキュー・マネージャー名クライアント接続 (QMNAME CLNTCONN) 属性) が同じであるクライアント接続 (CLNTCONN) チャネルが複数設定された CCDT ファイルのことです。異なる CLNTCONN 項目はそれぞれ別のキュー・マネージャーに解決されます。

これは、同じ複数インスタンス・キュー・マネージャーに属する複数の異なる IP アドレスまたはホスト名を表す複数の CLNTCONN 項目を設定しただけの CCDT ファイル (この方法は、コード・スタブと組み合わせて使用することもできます) とは異なります。

CCDT 複数キュー・マネージャーのアプローチを選択する場合、項目の優先順位付けを行うか、ランダム化されたワークロード管理 (WLM) を行うかを選択する必要があります。

優先順位付け
CLNTWGHT(1) 属性および AFFINITY(PREFERRED) 属性を設定したアルファベット順の複数の項目を使用して、最後の正常な接続を記憶します。

ランダム化
CLNTWGHT(1) 属性および AFFINITY(NONE) 属性を使用します。 CLNTWGHT を調整することにより、さまざまなスケールの IBM MQ サーバー間で WLM の加重を調整することができます。

注 : チャネル間で CLNTWGHT が大きく異なることがないようにしてください。

ロード・バランサー
複数の IBM MQ キュー・マネージャーの TCP/IP リスナーのポート・モニタリング機能がある、仮想 IP アドレス (VIP) が構成されたネットワーク・アプライアンスを意味します。 ネットワーク・アプライアンスでの VIP の構成方法は、使用しているネットワーク・アプライアンスによって異なります。

以下の選択項目は、メッセージを送信するアプリケーションと、同期要求/応答メッセージングを開始するアプリケーションにのみ関連します。 これらのメッセージおよび要求を処理するアプリケーションに関する考慮事項 (例えば、リスナーの完全な分離など) については、「メッセージ・リスナーをキューに接続する」で詳しく説明します。

単一のキュー・マネージャーに接続する既存のアプリケーションに必要なコード変更の規模

CONNAME リスト、CCDT 複数 QMGR、およびロード・バランサー
MQCONN("QMNAME") を MQCONN("*QMNAME") に変更

キュー・マネージャー名は、Java Platform, Enterprise Edition (Java EE) アプリケーションの Java Naming および Directory Interface (JNDI) 構成内にある場合があります。 そうでない場合は、1 文字のコード変更が必要になります。

コード・スタブ
既存の JMS または MQI の接続ロジックをコード・スタブで置き換えます。

さまざまな WLM 戦略のサポート

CONNAME リスト
優先順位付けのみ。

これは、コードに悪い影響を及ぼす可能性があります。

CCDT 複数 QMGR
優先順位付けまたはランダム。

これがコードに影響を及ぼすことはないと考えられます。

ロード・バランサー
すべて (すべてのメッセージの各接続を含む)。

これは、コードに良い影響を及ぼす可能性があります。

コード・スタブ
すべて (すべてのメッセージの各メッセージを含む)。

これは、コードに良い影響を及ぼす可能性があります。

1 次キュー・マネージャーを使用できない場合のパフォーマンス・オーバーヘッド

CONNAME リスト
リスト内の最初のものが常に試行されます。

これは、コードに悪い影響を及ぼす可能性があります。

CCDT 複数 QMGR
最後の正常な接続を記憶します。

これは、コードに良い影響を及ぼす可能性があります。

ロード・バランサー
ポート・モニタリングにより、適切でないキュー・マネージャーが回避されます。

これは、コードに良い影響を及ぼす可能性があります。

コード・スタブ
最後の正常な接続を記憶して、インテリジェントに再試行できます。

これは、コードに良い影響を及ぼす可能性があります。

XA トランザクションのサポート

CONNAME リスト、CCDT 複数 QMGR、およびロード・バランサー
トランザクション・マネージャーは、同じキュー・マネージャー・リソースに再接続するリカバリー情報を保管する必要があります。

複数の異なるキュー・マネージャーに解決される MQCONN 呼び出しでは、通常これは無効化されます。 例えば、Java EE では、XA を使用する場合、単一の接続ファクトリーは単一のキュー・マネージャーに解決される必要があります。

これは、コードに悪い影響を及ぼす可能性があります。

コード・スタブ
コード・スタブにより、トランザクション・マネージャーの XA 要件 (例えば、複数の接続ファクトリーなど) を満たすことができます。

これは、コードに良い影響を及ぼす可能性があります。

インフラストラクチャーの変更をアプリケーションから隠すための管理の柔軟性

CONNAME リスト
DNS のみ。

これは、コードに悪い影響を及ぼす可能性があります。

CCDT 複数 QMGR
DNS および共有ファイル・システム、共有ファイル・システム、または CCDT ファイル・プッシュ。

これがコードに影響を及ぼすことはないと考えられます。

ロード・バランサー
動的仮想 IP アドレス (VIP)。

これは、コードに良い影響を及ぼす可能性があります。

コード・スタブ
DNS または単一キュー・マネージャーの CCDT 項目。

これがコードに影響を及ぼすことはないと考えられます。

計画保守に伴う中断の回避

キュー・マネージャーの計画保守中のアプリケーションの中断 (エンド・ユーザーに影響のあるエラーやタイムアウトなど) を回避する方法について、検討と計画が必要な状況が他にもあります。 中断を回避する最善の方法は、キュー・マネージャーを停止する前にそこからすべての作業を除去することです。

要求と応答のシナリオを検討してください。 処理中のすべての要求を完了させ、アプリケーションにより応答が処理されるようにする一方で、追加の作業がシステムに送信されないようにする必要があります。しかし、キュー・マネージャーを静止させるだけでは、この要件を満たすことはできません。適切にコーディングされたアプリケーションでは、処理中の要求に対する応答メッセージを受信する前に、戻りコード RC2161 MQRC_Q_MGR_QUIESCING の例外を受け取るためです。

この場合、作業の実行依頼に使用する要求キューには PUT(DISABLED) を設定し、応答キューは PUT(ENABLED) と GET(ENABLED) のままにしておくことができます。 この方法により、要求キュー、伝送キュー、および応答キューの深さをモニターできます。 処理中の要求が完了するかタイムアウトになってすべてが安定化したら、キュー・マネージャーを停止できます。

ただし、要求側のアプリケーションで PUT(DISABLED) 要求キューを処理するための適切なコーディングを行う必要があります。この要求キューでは、メッセージの送信を試行したときに戻りコード RC2051 MQRC_PUT_INHIBITED のエラーが発生します。

IBM MQ への接続の作成時、または要求キューのオープン時には、この例外は発生しないことに注意してください。 この例外が発生するのは、MQPUT 呼び出しを使用して、メッセージを実際に送信しようとした場合のみです。

要求と応答のシナリオで、このエラー処理ロジックを含むコード・スタブを作成し、アプリケーション・チームにこのようなコード・スタブを今後使用するように依頼することで、動作が一貫したアプリケーションを開発することができます。

V 9.3.0 柔軟でスケーラブルなクライアント・アプリケーションの開発

フォールト・トレランスおよびスケーラビリティーの場合、均一クラスターへの接続オプションサポートするクライアント・アプリケーションをデプロイすることで、アプリケーションのインスタンスをキュー・マネージャー間で再バランスすることができます

均一クラスターの概要については、「均一クラスターについて」を参照してください。

この再バランシングはアプリケーションからは見えないのが理想ですが、この種類の種類のデプロイメントには特定の種類のアプリケーションのみが適しており、アプリケーション設計で何らかの考慮事項が必要となる場合があります。

これらの考慮事項は、2 つの主なカテゴリーに当てはまります。

- 再接続できるアプリケーションのために既に存在する可能性があるものの、均一クラスターにデプロイされると、より発生しやすくなる、珍しいエラー・パス。 例えば、再接続の後に、処理中の作業単位がバ

ックアウトされ、ブラウズ・カーソルがリセットされます。 これらは、現在の環境内の再接続可能アプリケーションにとってまれなイベントであり、したがって、アプリケーション・コードによってできるだけクリーンに処理されない場合があります。 アプリケーション・ロジックを見直し、そのような場合に適切な処理が行われるようにすることで、予期せぬ問題の発生を回避できます。

- 特定のキュー・マネージャーに対する類似性。 アプリケーションが常に同じキュー・マネージャーまたは特定のキュー・マネージャーに接続する必要があることが分かっている場合は、そのキュー・マネージャーに再接続するようにアプリケーションを構成するか、そのキュー・マネージャーへの接続が有効になっていないようにする必要があります。 ただし、これらの類似性は、応答メッセージの待機など、一時的なものである可能性があります。 以下のセクションでは、アプリケーション・コードからのこれらの類似性を考慮したバランシング・アルゴリズムへの影響について説明します。 これらのオプションの詳細、およびアプリケーション・コードではなく構成を使用して同様の方法を実行する方法については、「均一クラスターでのアプリケーション再リバランシングの影響」を参照してください。

MQI での再接続オプションの影響

MQCNO_RECONNECT の詳細については、「再接続オプション」を参照してください。

アプリケーションが常に同じキュー・マネージャーまたは特定のキュー・マネージャーに接続する必要があることが分かっている場合は、そのアプリケーションを MQCNO_RECONNECT_Q_MGR または MQCNO_RECONNECT_DISABLED として構成する必要があります。

MQI におけるバランシング・アルゴリズムの影響

ただし、特定の種類のアプリケーションのニーズに合わせて再バランシング動作を制御したり、影響を与えたりすることをおすすめします。例えば、処理中のトランザクションの中断の最小化や、リクエスター・アプリケーションが移動前に応答を受信するようにすることが例として挙げられます。

「均一クラスターでのアプリケーションの再バランシングの影響」では、特定のデフォルトの望ましい動作が想定され、説明されています。 また、構成時またはデプロイメント時における特定のアプリケーションの動作の場合、そのトピックで説明されているように、client.ini ファイルを使用して影響を与えることもできます。

それ以外の状況では、アプリケーション・ロジックのバランシング動作と要件部分を作成するほうが理にかなっているかもしれません。 これらの場合、MQBNO と呼ばれる構造 (バランシング・オプション) で、MQCONNX 呼び出し上のキュー・マネジャーへの接続時に、アプリケーションの同一の関連する特性を IBM MQ に提供することができます。

MQBNO 構造を指定する場合、IBM MQ が必要とするすべての情報を提供して、アプリケーションが異なるキュー・マネジャーへ再接続するよう要求される方法とタイミングを決める必要があります。

次のものを指定する必要があります。

- アプリケーションの **Type**
- 状態に関係なく、インスタンスがリバランスされる **Timeout**
- 任意の特殊な **BalanceOptions**

この例外として、必要に応じてタイムアウトを MQBNO_TIMEOUT_DEFAULT としてそのままにしておくことができます。 この場合、タイムアウトは、client.ini ファイル、アプリケーション、またはグローバル・スタンザ内で値が提供されいれば、その値に決まり、提供されていない場合は、基本のデフォルトである 10 秒に決まります。

この構造のフォーマットについて詳しくは、「MQBNO」を参照してください。

.NET アプリケーションについては、「.NET でのアプリケーションのバランシングの影響」を参照してください。

オブジェクト指向のアプリケーション

IBM MQ は、JMS、Java、C++、および .NET のサポートを提供します。 これらの言語およびフレームワークは、IBM MQ オブジェクト・モデルを使用します。これは、IBM MQ の呼び出しおよび構造体と同じ機能を持つクラスを提供します。

IBM MQ オブジェクト・モデルを使用する言語およびフレームワークの中には、メッセージ・キュー・インターフェース (MQI) でプロシージャー型言語を使用する場合には利用できない追加の機能を提供しているものがあります。

このモデルによって提供されるクラス、メソッド、およびプロパティーの詳細については、16 ページの『IBM MQ オブジェクト・モデル』を参照してください。

JMS

IBM MQ は、 Jakarta Messaging 3.0 および Java Message Service 2.0 仕様を実装するクラスを提供します。 IBM MQ classes for JMS について詳しくは、 IBM MQ classes for JMS の使用を参照してください。 IBM MQ classes for Java と IBM MQ classes for JMS の違いについては、どちらを使用するかを決定する際に参考にするために、 81 ページの『JMS/Jakarta Messaging および Java アプリケーションの開発』を参照してください。

IBM MQ Message Service Client (XMS) for C/C++ および IBM MQ Message Service Client (XMS) for .NET は、XMS と呼ばれるアプリケーション・プログラミング・インターフェース (API) を提供します。 この API は、Java Message Service (JMS) API と同じ一連のインターフェースを備えています。 詳しくは、614 ページの『XMS .NET アプリケーションの開発』を参照してください。

Java

Java で IBM MQ オブジェクト・モデルを使用するプログラムのコーディングについては、 IBM MQ classes for Java の使用 を参照してください。

Stabilized IBM では、IBM MQ classes for Java に対してこれ以上拡張機能を提供することはありません。また、IBM MQ 8.0 で出荷されたレベルで機能的に固定化されています。 IBM MQ classes for Java と IBM MQ classes for JMS の違いを知り、どちらを使うか決めるためには、81 ページの『JMS/Jakarta Messaging および Java アプリケーションの開発』を参照してください。

C++

IBM MQ では、IBM MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。 C++ における IBM MQ オブジェクト・モデルを使用したプログラムのコーディングについては、C++ の使用を参照してください。C/C++ のメッセージ・サービス・クライアントおよび .NET は、Java Message Service (JMS) と同じインターフェースのセットを持つ XMS と呼ばれるアプリケーション・プログラミング・インターフェース (API) を提供します。

.NET

IBM MQ .NET クラスを使用する .NET プログラムのコーディングについては、 .NET アプリケーションの開発 を参照してください。 C/C++ および .NET 用の Message Service Client は、Java Message Service (JMS) API と同じインターフェース・セットを持つ、XMS という名前のアプリケーション・プログラミング・インターフェース (API) を提供します。

関連概念

717 ページの『IBM MQ での MQI アプリケーションの開発』
IBM MQ は、C、Visual Basic、COBOL、アセンブラー、RPG、pTAL、および PL/I のサポートを提供します。 これらのプロシージャー型言語は、Message Queue Interface (MQI) を使用してメッセージ・キューイング・サービスにアクセスします。

技術概要
7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

関連資料

アプリケーションの開発に関する参照情報

IBM MQ オブジェクト・モデル

IBM MQ オブジェクト・モデルは、クラス、メソッド、およびプロパティーで構成されます。

IBM MQ オブジェクト・モデルは、以下から構成されます。

- クラス。これは、キュー・マネージャー、キュー、メッセージなどの、よく使用する IBM MQ の概念を表します。
- メソッド。これはクラスごとに存在し、MQI 呼び出しに対応します。
- プロパティー。これはクラスごとに存在し、IBM MQ オブジェクトの属性に対応します。

IBM MQ オブジェクト・モデルを使って IBM MQ アプリケーションを作成するときは、まず、アプリケーション内にクラスのインスタンスを作成します。 オブジェクト指向プログラミングでは、クラスのインスタンスをオブジェクト と呼びます。 オブジェクトの作成が終わったら、オブジェクトのプロパティーの値の検査や設定 (MQINQ や MQSET 呼び出しの発行と同等)、およびオブジェクトに対するメソッド呼び出しの作成 (その他の MQI 呼び出しの発行と同等) によって、オブジェクトと対話します。

クラス

IBM MQ オブジェクト・モデルには、クラスの基本セットとして次のものが用意されています。

実際のモデルは、サポートされるオブジェクト指向環境によって若干異なります。

MQQueueManager
MQQueueManager クラスのオブジェクトは、キュー・マネージャーへの接続を表します。 メソッドは、Connect()、Disconnect()、Commit()、および Backout() です (MQCONN または MQCONNX、MQDISC、MQCMIT、および MQBACK と等価)。 プロパティーは、キュー・マネージャーの属性に対応しています。 キュー・マネージャーの属性プロパティーにアクセスすると、まだキュー・マネージャーに接続していない場合は暗黙的にキュー・マネージャーに接続されます。 MQQueueManager オブジェクトを破棄すると、キュー・マネージャーへの接続が暗黙的に切断されます。

MQQueue
MQQueue クラスのオブジェクトは、キューを表します。 メソッドは、キューへのメッセージの Put() とキューからの Get() です (MQPUT および MQGET と同じ)。 プロパティーは、キューの属性に対応しています。 キューの属性プロパティーにアクセスしたり、Put() または Get() メソッド呼び出しを発行したりすると、暗黙的にキューがオープンされます (MQOPEN と同じ)。 MQQueue オブジェクトを破棄すると、キューが暗黙的にクローズされます (MQCLOSE と同じ)。

MQTopic
MQTopic クラスのオブジェクトは、トピックを表します。 メソッドは、トピックへのメッセージの Put() (パブリッシュ) とトピック からの Get() (受信およびサブスクライブ) です (MQPUT および MQGET に相当)。 プロパティーは、トピックの 属性に対応しています。 1 つの MQTopic オブジェクトには、パブリッシュまたはサブスクリプションの ためにのみアクセスでき、両方同時にはできません。メッセージの受信に 使用される場合、MQTopic オブジェクトを非管理または管理サブスクリプションと共に、永続的または非永続的サブスクライバーとして 作成できます。これらの異なるシナリオ用に複数の多重定義のコンストラクターが 用意されています。

MQMessage
MQMessage クラスのオブジェクトは、キューに書き込まれるメッセージまたはキューから読み取られるメッセージを表します。 このクラスのオブジェクトにはバッファーが含まれ、アプリケーション・データと MQMD の両方をカプセル化します。 プロパティーは、MQMD のフィールドに対応しています。また、さまざまなタイプ (ストリング、長整数、短整数、単一バイトなど) のユーザー・データのバッファーへの書き込みと読み取りを行うメソッドがあります。

MQPutMessageOptions
MQPutMessageOptions クラスのオブジェクトは、MQPMO 構造体を表します。 プロパティーは、MQPMO のフィールドに対応しています。

MQGetMessageOptions
MQGetMessageOptions クラスのオブジェクトは、MQGMO 構造体を表します。 プロパティーは、MQGMO のフィールドに対応しています。

MQProcess
MQProcess クラスのオブジェクトは、プロセス定義 (トリガー処理に使用する) を表します。 特性は、プロセス定義の属性を表します。

Multi **MQDistributionList**
MQDistributionList クラスのオブジェクトは、配布リスト (1 回の MQPUT で複数のメッセージを送信するために使用する) を表します。 このクラスのオブジェクトには、MQDistributionListItem オブジェクトのリストが含まれます。

Multi **MQDistributionListItem**
MQDistributionListItem クラスのオブジェクトは、配布リストの宛先の 1 つを表します。 このクラスのオブジェクトは MQOR、MQRR、および MQPMR 構造体をカプセル化します。プロパティーは、これらの構造体のフィールドに対応しています。

オブジェクト参照子

MQI を使用する IBM MQ プログラムでは、IBM MQ は接続ハンドルとオブジェクト・ハンドルをプログラムに戻します。

これらのハンドルは、その後の IBM MQ 呼び出しでパラメーターとして渡さなければなりません。 IBM MQ オブジェクト・モデルでは、これらのハンドルはアプリケーション・プログラムからは見えません。 その代わり、クラスからオブジェクトを作成すると、アプリケーション・プログラムにオブジェクト参照子が戻されます。 オブジェクトに対するメソッド呼び出しやプロパティー・アクセスを行うときには、このオブジェクト参照子が使用されます。

戻りコード

メソッド呼び出しの発行やプロパティー値の設定を行うと、戻りコードが設定されます。

設定される戻りコードは完了コードと理由コードであり、それ自体がオブジェクトのプロパティーです。 完了コードと理由コードの値は MQI で定義される値と同じですが、オブジェクト指向環境に固有の値が付加されます。

IBM MQ メッセージ

IBM MQ メッセージは、メッセージ・プロパティーとアプリケーション・データから構成されます。 送信側アプリケーションと受信側アプリケーションの間でメッセージが伝送されるときには、メッセージ・キューイング・メッセージ記述子 (MQMD) に、アプリケーション・データに付随する制御情報が含まれます。

メッセージの各部分

IBM MQ のメッセージは次の 2 つの部分で構成されます。

- メッセージ・プロパティー
- アプリケーション・データ

18 ページの図 2 はメッセージを表し、それがメッセージ・プロパティーとアプリケーション・データに論理的に分割される方法を示します。

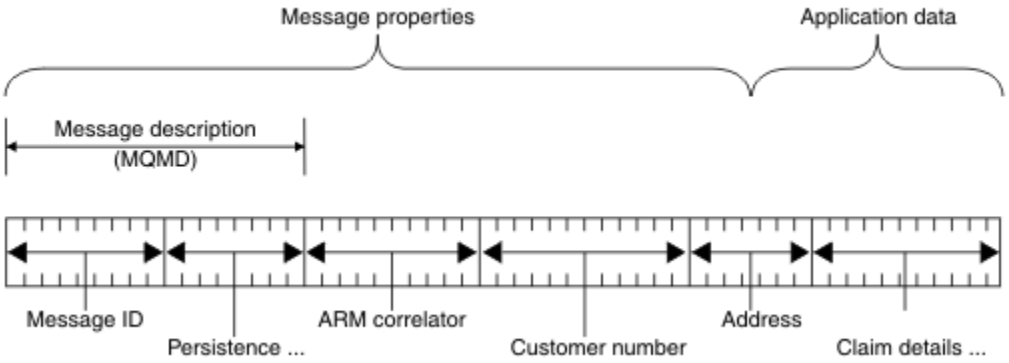


図 2. メッセージの表現

IBM MQ メッセージで 運ばれるアプリケーション・データは、データ変換がそれに対して行われない限り、キュー・マネージャーによって変更されることはありません。 また、IBM MQ は、このデータの内容に何の制約も付けません。 各メッセージ内のデータの長さは、 キューとキュー・マネージャーのいずれの **MaxMsgLength** 属性の値を超えることもできません。

ALW AIX, Linux, and Windows では、キュー・マネージャーおよびキューの *MaxMsgLength* 属性はデフォルトで 4 MB (4 194 304 バイト) に設定されます。これは、必要に応じて最大 100 MB (104 857 600 バイト) まで変更できます。

IBM i IBM i では、キュー・マネージャーおよびキューの *MaxMsgLength* 属性はデフォルトで 4 MB (4 194 304 バイト) に設定されます。これは、必要に応じて最大 100 MB (104 857 600 バイト) まで変更できます。 15 MB より大きい IBM MQ メッセージを IBM i で使用する場合は、1011 ページの『IBM i でのプロシージャー型アプリケーションの構築』を参照してください。

z/OS z/OS では、キュー・マネージャーの **MaxMsgLength** 属性は 100 MB に固定されます。キューの **MaxMsgLength** 属性はデフォルトで 4 MB (4 194 304 バイト) になり、必要に応じて最大 100 MB まで変更できます。

一部の環境では、メッセージを **MaxMsgLength** 属性の値より少し短くします。 詳しくは、758 ページの『メッセージ内のデータ』を参照してください。

MQPUT または MQPUT1 MQI 呼び出しを使用する際は、メッセージを作成します。 ユーザーがこれらの呼び出しへの入力として制御情報 (メッセージの優先順位や応答キューの名前など) とデータを提供した後、呼び出しによってキューにメッセージが書き込まれます。 これらの呼び出しの詳細については、MQPUT および MQPUT1 を参照してください。

メッセージ記述子

メッセージ制御情報にアクセスするには、メッセージ記述子 を定義する MQMD 構造体を使用します。

MQMD 構造体の詳細な説明については、MQMD - メッセージ記述子を参照してください。

メッセージの発信元に関する情報が含まれる MQMD 内のフィールドを使用する方法については、47 ページの『メッセージ・コンテキスト』を参照してください。

メッセージ記述子にはさまざまなバージョンがあります。 メッセージのグループ化とセグメント化に関する追加情報 (44 ページの『メッセージ・グループ』を参照) は、 メッセージ記述子 (または MQMDE) のバージョン 2 で提供されます。 これは、追加のフィールドがあることを除いてバージョン 1 のメッセージ記述子と同じです。 これらのフィールドについては、MQMDE - 拡張メッセージ記述子を参照してください。

メッセージのタイプ

IBM MQ によって定義されるメッセージには、4 つのタイプがあります。

これら 4 つのメッセージは次のとおりです。

- データグラム
- 要求メッセージ
- 応答メッセージ
- 報告メッセージ
  - 報告メッセージのタイプ
  - 報告メッセージのオプション

アプリケーションは、アプリケーション間で情報を渡すために、最初の 3 つのタイプのメッセージを使用できます。 4 番目のタイプである、報告タイプは、アプリケーションおよびキュー・マネージャーがエラーの発生のようなイベントについての情報を報告するために用いるものです。

メッセージの各タイプは、MQMT_* の値によって識別されます。 独自のメッセージ・タイプを定義することもできます。 使用できる値の範囲については、MsgType を参照してください。

データグラム

メッセージを受信した (つまり、キューからメッセージを読み取った) アプリケーションからの応答が不要のときは、データグラム を使用します。

データグラムを使用できるアプリケーションの例としては、空港の待合室でフライト情報を表示するアプリケーションがあります。 メッセージには、1 画面分のフライト情報のデータを入れることができます。 このようなアプリケーションでは、メッセージが伝達されないことがほとんど問題にならないので、メッセージについての肯定応答を要求することはあまりありません。 アプリケーションは、時間をおかずに更新メッセージを送信します。

要求メッセージ

メッセージを受信したアプリケーションからの応答を要求するときは、要求メッセージ を使用します。

要求メッセージを使用できるアプリケーションの例としては、当座預金口座の残高を表示するアプリケーションがあります。 要求メッセージには口座番号が入り、応答メッセージには口座の残高が入ります。

応答メッセージと要求メッセージをリンクする場合、次の 2 つのオプションがあります。

- 要求メッセージに関連付けられた応答メッセージに情報を確実に書き込む役目を、要求メッセージを処理するアプリケーションに与える。
- 要求メッセージのメッセージ記述子の報告フィールドを使用して、応答メッセージの *MsgId* フィールドおよび *CorrelId* フィールドの内容を指定する。
  - 元のメッセージの *MsgId* または *CorrelId* のどちらかを応答メッセージの *CorrelId* フィールドにコピーするように要求できます (デフォルトのアクションでは *MsgId* がコピーされます)。
  - 応答メッセージに対して新規の *MsgId* を生成するか、または元のメッセージの *MsgId* を応答メッセージの *MsgId* フィールドにコピーするかのどちらかを要求できます (デフォルトのアクションでは新規のメッセージ ID が生成されます)。

応答メッセージ

他のメッセージに応答するときは、応答メッセージ を使用します。

応答メッセージを作成する場合は、応答を送る先のメッセージのメッセージ記述子に設定されたすべてのオプションを考慮に入れてください。 報告オプションは、メッセージ ID (*MsgId*) および相関 ID (*CorrelId*) フィールドの内容を指定します。 これらのフィールドによって、応答を受け取るアプリケーションは、応答と元の要求の相関をとることができます。

レポート・メッセージ

報告メッセージ は、メッセージ処理中のエラー発生などのイベントをアプリケーションに通知します。

報告メッセージは、以下により生成できます。

- キュー・マネージャー
- メッセージ・チャネル・エージェント (例えば、メッセージを送達できないとき)、または
- アプリケーション (例えば、メッセージ内のデータが使用不能のとき)

報告メッセージはいつでも生成され、ご使用のアプリケーションが予期していないときでもキューに到着する可能性があります。

報告メッセージのタイプ

キューにメッセージを書き込む場合、次の受信を選択できます。

- 例外報告メッセージ。 このメッセージは、例外フラグ・セットがあるメッセージへの応答として送信されます。 これは、メッセージ・チャネル・エージェント (MCA) またはアプリケーションによって生成されます。

- 満了報告メッセージ。 このメッセージは、アプリケーションが満了しきい値に達したメッセージの取り出しを試みたことを示します。このメッセージには、破棄するためのマークが付けられます。 このタイプの報告は、キュー・マネージャーによって生成されます。
- 到着確認 *(COA)* 報告メッセージ。 このメッセージは、メッセージが宛先キューに到達したことを示します。 これは、キュー・マネージャーによって生成されます。
- 送達確認 *(COD)* 報告メッセージ。 このメッセージは、受信側のアプリケーションによってメッセージが取り出されたことを示します。 これは、キュー・マネージャーによって生成されます。
- 肯定アクション通知 *(PAN)* 報告メッセージ。 このメッセージは、要求が正常にサービスされた (つまり、メッセージ内で要求されたアクションが正常に実行された) ことを示します。 このタイプの報告は、アプリケーションによって生成されます。
- 否定アクション通知 *(NAN)* 報告メッセージ。 このメッセージは、要求が正常にサービスされなかった (つまり、メッセージ内で要求されたアクションが正常に実行されなかった) ことを示します。 このタイプの報告は、アプリケーションによって生成されます。

注 : 各タイプのレポート・メッセージには、次のいずれかの情報が含まれています。

- 元のメッセージ全体
- 元のメッセージの最初の 100 バイトのデータ
- 元のメッセージのデータは含まない

キューにメッセージを書き込む場合、複数のタイプの報告メッセージを要求することができます。 送達確認レポート・メッセージと例外レポート・メッセージのオプションを選択すると、メッセージの送達が失敗した場合に、例外レポート・メッセージが受信されます。 ただし、送達確認報告メッセージのオプションだけを選択し、そのメッセージの送達が失敗した場合には、例外報告メッセージは受信されません。

特定のメッセージを生成するための基準に合致すると、要求した報告メッセージだけが、受信する報告メッセージとなります。

報告メッセージのオプション

例外が発生した後にメッセージを破棄 できます。 例外報告メッセージを要求したあとに廃棄オプションを選択すると、その報告メッセージは *ReplyToQ* と *ReplyToQMgr* に送られ、元のメッセージは廃棄されます。

注 : これの利点は、送達不能キューに入れられるメッセージの数を減らすことができることです。 しかし、このことは、データグラム・メッセージのみの送信でない限り、ユーザーのアプリケーションは戻されたメッセージを処理しなければならないことを意味します。 例外報告メッセージが生成される場合、その例外報告メッセージは元のメッセージの持続性を継承します。

報告メッセージが送達できない場合 (例えば、キューが満ぱいの場合)、その報告メッセージは送達不能キューに入れられます。

報告メッセージを受信したい場合には、*ReplyToQ* フィールドに応答先キューの名前を指定します。指定しないと、元のメッセージの MQPUT または MQPUT1 は失敗して MQRC_MISSING_REPLY_TO_Q を出します。

メッセージに関して作成される報告メッセージの *MsgId* および *CorrelId* フィールドの内容を指定するために、メッセージのメッセージ記述子 (MQMD) に以下の報告オプションを使用することもできます。

- 元のメッセージの *MsgId* または *CorrelId* のいずれかを報告メッセージの *CorrelId* フィールドにコピーするように要求できます。 デフォルトのアクションでは、メッセージ ID がコピーされます。 MQRO_COPY_MSG_ID_TO_CORRELID を使用すると、メッセージの送信側は応答または報告メッセージを元のメッセージと相関できます。 応答メッセージまたは報告メッセージの相関 ID は、元のメッセージのメッセージ ID と同じになります。
- 報告メッセージに対して新規の *MsgId* を生成するか、または元のメッセージの *MsgId* を報告メッセージの *MsgId* フィールドにコピーするかのどちらかを要求できます。 デフォルトのアクションでは新規のメッセージ ID が生成されます。 MQRO_NEW_MSG_ID を使用すると、システムにある各メッ

セージに異なるメッセージ ID が与えられ、それぞれのメッセージをシステム内の他のすべてのメッセージと明確に区別できるようになります。

- 特殊なアプリケーションは MQRO_PASS_MSG_ID または MQRO_PASS_CORREL_ID を使用する必要が生じるかもしれません。 しかし、例えば、同じメッセージ ID のある複数のメッセージがキューに入っている場合などに、キューからメッセージを読み取るアプリケーションが正しく機能するように設計する必要があります。

  サーバー・アプリケーションは、要求メッセージにあるこれらのフラグの設定を検査して、*MsgId* および *CorrelId* フィールドを適切な応答またはレポート・メッセージに設定する必要があります。

  要求側アプリケーションとサーバー・アプリケーション間の仲介役として機能するアプリケーションでは、これらのフラグの設定を検査する必要はありません。 これは、それらのアプリケーションが、通常は *MsgId*、*CorrelId*、および *Report* フィールドが変更されていないサーバー・アプリケーションにメッセージを転送するからです。 これにより、サーバー・アプリケーションは応答メッセージの *CorrelId* フィールドにある元のメッセージから *MsgId* をコピーできます。

メッセージについての報告を生成するとき、サーバー・アプリケーションはこれらのオプションのいずれかが設定されているかどうかをテストする必要があります。

レポート・メッセージの使用方法の詳細については、Report を参照してください。

報告の性質を示すために、キュー・マネージャーはフィードバック・コードの範囲を用います。 キュー・マネージャーは、これらのコードを報告メッセージのメッセージ記述子の *Feedback* フィールドに入れます。 また、キュー・マネージャーは、MQI 理由コードも *Feedback* フィールドに戻すことができます。 IBM MQ は、アプリケーションが使用するフィードバック・コードの範囲を定義します。

フィードバック・コードおよび理由コードの詳細については、Feedback を参照してください。

フィードバック・コードを用いるプログラムの例としては、キューを扱うその他のプログラムのワークロードをモニターするプログラムがあります。 このようなプログラムは、例えば、キューを扱うプログラムのインスタンスが複数あって、キューに到着するメッセージの数がもう限界にきている場合に、処理プログラムの 1 つに報告メッセージ (フィードバック・コード MQFB_QUIT をもつ) を送って、その活動を終了するように指示できます。 (モニター・プログラムは 1 つのキューを扱うプログラムがいくつあるかを知るために MQINQ 呼び出しを用いる可能性があります。)

Multi 報告とセグメント化されたメッセージ

IBM MQ for z/OS ではサポートされない。

メッセージがセグメント化された場合、報告を生成するように求めると、メッセージがセグメント化されていないときよりも多くの報告を受け取ることがあります。

セグメント化されたメッセージの説明については、792 ページの『メッセージのセグメント化』を参照してください。

IBM MQ によって生成される報告の場合

メッセージをセグメント化したり、キュー・マネージャーにそれを求めたりした場合は、メッセージ全体に対して 1 つの報告しか受け取らないケースは 1 つしかありません。 これは、COD 報告だけを要求し、読み取りアプリケーションで MQGMO_COMPLETE_MSG を指定したときです。

これ以外の場合は、複数の報告 (通常はセグメントごとに 1 つの報告) を処理できるようにアプリケーションを準備しておく必要があります。

注 : メッセージをセグメント化したときに、元のメッセージ・データの最初の 100 バイトだけが戻されるようにしたい場合は、100 以上のオフセットがあるセグメントのデータがない報告を求めるため、報告オプションの設定を変更してください。 設定を変更しないで、各セグメントが 100 バイトずつデータを要求するように設定したままにして、MQGMO_COMPLETE_MSG を指定して単一の MQGET で報告メッセージを取り出すと、報告は、適切なオフセットごとに 100 バイトずつ読み取ったデータが入った 1 つの大規模メッセージにアセンブルされます。 この場合は、大容量のバッファーが必要です。ない場合には、MQGMO_ACCEPT_TRUNCATED_MSG を指定する必要があります。

アプリケーションによって生成される報告の場合

アプリケーションによって報告が生成される場合は、元のメッセージ・データの先頭に存在する IBM MQ ヘッダーを、報告メッセージ・データに必ずコピーします。

それから、報告メッセージ・データに何も追加しないか、あるいは元のメッセージ・データの 100 バイトまたはすべて (あるいは通常含めている量) を追加します。

コピーしなければならない IBM MQ ヘッダーは、連続する形式名を確認することによって認識できます。この形式名は、MQMD で始まり、そのあとにすべての存在するヘッダーが続いています。 次の `Format` 名は、これらの IBM MQ のヘッダーを示します。

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* は、文字 MQH で始まるあらゆる名前を表します。

`Format` 名は、MQDLH と MQXQH では特定の位置にありますが、他の IBM MQ ヘッダーでは同じ位置にあります。 ヘッダーの長さは、MQMDE、MQIMS、およびすべての MQH* ヘッダーで同じ位置に存在するフィールドに入っています。

バージョン 1 の MQMD を使用し、セグメント、グループ内のメッセージ、セグメント化が認められているメッセージのいずれかについて報告する場合、その報告データは MQMDE で始まる必要があります。 *OriginalLength* フィールドを元のメッセージ・データの長さに設定します。ただし、存在する IBM MQ ヘッダーの長さは除きます。

報告の取得

COA 報告または COD 報告を要求する場合は、MQGMO_COMPLETE_MSG を使ってそれらの報告を再アセンブルするように求めることができます。

MQGMO_COMPLETE_MSG が指定された MQGET は、十分な報告メッセージ (1 つのタイプのメッセージ、例えば COA で、*GroupId* が同じである) がキューにあって、1 つの完全な元のメッセージを表しているとき、正常に実行されます。 これは、報告メッセージ自体に元のデータが完全には含まれていないときにも言えることです。各報告メッセージ内の *OriginalLength* フィールドは、その報告メッセージが表す元のデータの長さを示します (これは、データ自体が存在しなくても同様です)。

キューに複数の異なる報告タイプがある (例えば COA と COD が両方ある) ときでも、この技法を使用できます。MQGMO_COMPLETE_MSG が指定された MQGET では、*Feedback* コードが同じ場合にだけ、報告メッセージを再アセンブルするためです。 ただし、一般的に例外報告の *Feedback* コードはそれぞれ異なるため、通常、例外報告にこの技法は使用できません。

この技法を使用して、メッセージ全体が到達したという肯定的な結果を読み取ることができます。 ただし、多くの環境では、いくつかのセグメントによって例外 (または満了、ただし満了を認めている場合) が生成される一方で、ほかのいくつかのセグメントが到達するという可能性に対応できるようにしておく必要があります。 このような場合には MQGMO_COMPLETE_MSG は使用できません。一般的に別々のセグメントの *Feedback* コードは異なり、セグメントの報告が複数になることがあるためです。 ただし、MQGMO_ALL_SEGMENTS_AVAILABLE は使用できます。

これを使用するには、報告が到達したときにその報告を検索し、元のメッセージに何が起きたかを表すピクチャーをアプリケーションの中に作成しなければならないことがあります。 報告メッセージ内の *GroupId* フィールドを使用して、報告と元のメッセージの *GroupId* の相関をとることができます。また、*Feedback* フィールドを使用して、各報告メッセージのタイプを識別できます。 これらの方法は、使用しているアプリケーションの要件によって異なります。

次に 1 つの方法を示します。

- COD 報告と例外報告を要求する。

- 一定の時間が過ぎたら、MQGMO_COMPLETE_MSG を使って COD 報告の完全なセットが受け取られているかどうかを調べる。 受け取られていた場合は、メッセージ全体が処理されたことをアプリケーションが認識します。
- 受け取られていなかった場合で、このメッセージに関連付けられている例外報告がある場合は、セグメント化されていないメッセージに関して問題を処理しますが、このときある時点で孤立セグメントのクリーンアップも確実にする必要があります。
- どの種類の報告もないセグメントがある場合は、元のセグメント (または報告) がチャネルの再接続を待機しているか、あるいはネットワークがある時点で過負荷になることがあります。 例外報告をまったく受け取らない場合 (あるいは、存在する例外報告が一時的なものだけであると考えられる場合) は、アプリケーションの待ち時間を少し長くしても構いません。

  この考慮事項は、前述のセグメント化されていないメッセージを処理するときの考慮事項と同様です。ただしここでは、孤立セグメントのクリーンアップの可能性も考慮する必要があります。

元のメッセージが重要でない場合 (例えば、あとから繰り返すことのできる照会やメッセージである場合) は、その孤立セグメントが削除されるように有効期限時刻を設定します。

バックレベルのキュー・マネージャー

セグメント化をサポートするキュー・マネージャーが報告を生成したが、セグメント化をサポートしないキュー・マネージャーがその報告を受け取った場合は、メッセージの中に元のデータの 0、100 バイト、またはすべてが入っているのに加えて、MQMDE 構造体 (報告が表す *Offset* と *OriginalLength* を識別する) が必ず報告データの中にあります。

しかし、メッセージのセグメントが、セグメント化をサポートしないキュー・マネージャーに処理されるときには、報告がそこで生成された場合に、元のメッセージ内の MQMDE 構造体はデータとして扱われるだけです。 したがって、元のデータのゼロ・バイトが要求されたときには、メッセージのセグメントは報告データに含まれません。 MQMDE がないと、報告メッセージは利用価値がなくなることがあります。

メッセージがバックレベルのキュー・マネージャーによって処理される可能性のある場合には、報告内のデータを 100 バイト以上要求してください。

メッセージ制御情報およびメッセージ・データの形式

キュー・マネージャーは、メッセージ内の制御情報の形式だけに関係しますが、メッセージを処理するアプリケーションは、制御情報とデータの両方の形式に関係します。

メッセージ制御情報の形式

メッセージ記述子の文字ストリング・フィールド内の制御情報はキュー・マネージャーが使用する文字セットを用いなければなりません。

キュー・マネージャー・オブジェクトの **CodedCharSetId** 属性がこの文字セットを定義します。 アプリケーションがメッセージを 1 つのキュー・マネージャーから別のキュー・マネージャーに渡すとき、このメッセージを伝送するメッセージ・チャネル・エージェントは、どういうデータ変換を実行しなければならないかを決定するためにこの属性の値を使用するので、制御情報はこの文字セットを使う必要があります。

メッセージ・データの形式

次のいずれかを指定できます。

- アプリケーション・データの形式
- 文字データの文字セット
- 数値データの形式

上記を指定するには、以下のフィールドを使用します。

Format
これは、メッセージの受信側に対して、メッセージ内のアプリケーション・データの形式を示します。

キュー・マネージャーがメッセージを作成するときは、状況によってはこの *Format* フィールドを使用して、メッセージの形式を識別します。 例えば、キュー・マネージャーがメッセージを送達できないときは、そのメッセージを送達不能 (未配布メッセージ) キューに書き込みます。 キュー・マネージャーは、メッセージにヘッダー (さらに制御情報を含んでいる) を追加し、このことを示すために *Format* フィールドを変更します。

キュー・マネージャーは MQ で始まる名前 (例えば MQFMT_STRING) を持つ組み込み形式 を数多く持っています。 これらの形式が要求を満たさない場合、独自の形式 (ユーザー定義形式) を定義することができますが、その際には MQ で始まる名前は使用しないでください。

独自の形式を作成して使用する際には、MQGMO_CONVERT を使ってメッセージを読み取るプログラムをサポートするためのデータ変換出口を作成する必要があります。

CodedCharSetId
これには、メッセージ内の文字データの文字セットを定義します。 この文字セットをキュー・マネージャーの文字セットに設定したい場合は、このフィールドを定数 MQCCSI_Q_MGR または MQCCSI_INHERIT に設定できます。

キューからメッセージを読み取るときは、*CodedCharSetId* フィールドの値とアプリケーションが期待している値とを比較してください。 2 つの値が異なる場合、メッセージの文字データを変換したり、どちらかが使用できる場合にはデータ変換メッセージ出口を使用することが必要になる場合があります。

Encoding
これには、2 進整数、パック 10 進整数、浮動小数点数が入っている数値メッセージ・データの形式を記述します。 このフィールドは、通常キュー・マネージャーを実行中の特別なマシンに応じてエンコードさせます。

メッセージをキューに書き込むときは、通常、*Encoding* フィールドに定数 MQENC_NATIVE を指定します。 これは、メッセージ・データのエンコードが、アプリケーションが稼働しているマシンでのエンコードと同じであることを意味します。

メッセージをキューから読み取るときは、メッセージ記述子内の *Encoding* フィールドの値と、マシン上の定数 MQENC_NATIVE の値を比較してください。 2 つの値が異なる場合、メッセージの数値データを変換したり、どちらかが使用できる場合にはデータ変換メッセージ出口を使用することが必要になる場合があります。

アプリケーション・データの変換

アプリケーション・データは、異なるプラットフォームが関係する別のアプリケーションによって必要とされる文字セットやエンコード形式に変換しなければならないことがあります。

変換は送信側のキュー・マネージャーによって行われる場合と、受信側のキュー・マネージャーによって行われる場合があります。 組み込み形式のライブラリーが要求を満たさない場合には、独自のものを定義することができます。 変換のタイプは、メッセージ記述子 MQMD の形式フィールドで指定されるメッセージ形式によって異なります。

注 : MQFMT_NONE が指定されたメッセージは変換されません。

送信側のキュー・マネージャーでの変換

アプリケーション・データを変換するために送信側のメッセージ・チャネル・エージェント (MCA) が必要な場合は、CONVERT チャネル属性を「YES」に設定してください。

特定の組み込み形式や、適切なユーザー出口が提供されているときのユーザー定義の形式については、送信側のキュー・マネージャーによって変換が実行されます。

組み込み形式

以下が含まれます。

- すべての文字からなるメッセージ (形式名 MQFMT_STRING を使用)
- IBM MQ 定義のメッセージ (プログラム式コマンド形式など)

IBM MQ では、管理メッセージとイベントについて (この場合、使用される形式名は MQFMT_ADMIN) プログラム式コマンド形式メッセージを使用します。 独自のメッセージについて同じ形式 (形式名 MQFMT_PCF) を使用ができ、組み込みデータ変換を活用できます。

キュー・マネージャーの組み込み形式はすべて MQFMT から始まる名前を持っています。 これらについては、Format にリストと説明が記載されています。

アプリケーション定義の形式

ユーザー定義の形式の場合、アプリケーションのデータ変換は、データ変換出口プログラムで行う必要があります (詳細は、988 ページの『データ変換出口の作成』を参照してください)。 クライアント / サーバー環境では、出口プログラムはサーバーでロードされ、そこで変換が行われます。

受信側のキュー・マネージャーでの変換

アプリケーション・メッセージ・データは、受信側のキュー・マネージャーによって、組み込み形式とユーザー定義形式の両方に変換できます。

MQGMO_CONVERT オプションを指定すると、MQGET 呼び出しの処理中に変換が行われます。 詳細については、Options を参照してください。

コード化文字セット

IBM MQ 製品では、使用中のオペレーティング・システムで提供しているコード化文字セットをサポートします。

キュー・マネージャーを作成する際には、使用されるそのキュー・マネージャーのコード化文字セット ID (CCSID) は使用する環境のコード化文字セット ID に基づきます。 これが混合コード・ページの場合には、IBM MQ は混合コード・ページの SBCS の部分をキュー・マネージャーの CCSID として使用します。

一般的なデータ変換では、使用中のオペレーティング・システムが DBCS コード・ページをサポートしている場合、IBM MQ はそれを使用できます。

オペレーティング・システムがサポートしているコード化文字セットの詳細については、ご使用のオペレーティング・システムの資料を参照してください。

複数のプラットフォームにまたがるアプリケーションを作成するときは、アプリケーション・データの変換、形式名、およびユーザー出口を考慮する必要があります。 データ変換出口の呼び出し方法と書き込み方法については、988 ページの『データ変換出口の作成』を参照してください。

メッセージ優先順位

メッセージの優先順位として数値を設定することも、キューのデフォルトの優先順位をメッセージに取得させることも可能です。

メッセージをキューに書き込むとき、メッセージの優先順位を (MQMD 構造体の *Priority* フィールドに) 設定します。 優先順位の数値を設定することも、メッセージにキューを優先順位としてデフォルト値を取らせることも可能です。

キューの **MsgDeliverySequence** 属性によって、キュー上のメッセージが FIFO (先入れ先出し) または優先順位内の FIFO のどちらの順序で格納されるかが決まります。 この属性が MQMDS_PRIORITY に設定されている場合は、メッセージ記述子の *Priority* フィールドに指定されている優先順位でメッセージがキューに入れられます。しかし、MQMDS_FIFO が設定されている場合は、キューのデフォルト優先順位でメッセージがキューに入れられます。 同じ優先順位のメッセージは到着順にキューに格納されます。

キューの **DefPriority** 属性は、そのキューに入れられるメッセージのデフォルト優先順位の値を設定します。 この値は、キューの作成時に設定されますが、後で変更可能です。 別名キューおよびリモート・キューのローカル定義は、それらが解決される基本キューとは異なるデフォルト優先順位を持つ可能性があります。 解決経路 (745 ページの『名前の解決』を参照) に複数のキュー定義がある場合、デフォルト優先順位には、オープン・コマンドで指定されるキューの **DefPriority** 属性の (書き込み操作時の) 値が使用されます。

キュー・マネージャーの **MaxPriority** 属性の値は、そのキュー・マネージャーによって処理されるメッセージに割り当て可能な最高優先順位です。 ユーザーはこの属性の値を変更できません。 IBM MQ では、この属性の値は 9 です。したがって、0 (最低) から 9 (最高) までの優先順位をもつメッセージを作成できます。

メッセージ・プロパティー

メッセージ・プロパティーを使用すると、アプリケーションは、処理するメッセージを選択したり、MQMD または MQRFH2 ヘッダーにアクセスせずにメッセージに関する情報を取得する ことができます。 また、メッセージ・プロパティーは、IBM MQ と JMS アプリケーションの間の通信を行いやすくします。

メッセージ・プロパティーは、メッセージに関連付けられたデータであり、 名前テキストと、特定のタイプの値からなります。 メッセージ・セレクターはメッセージ・プロパティーを 使用して、トピックへのパブリケーションをフィルタリングしたり、キューから選択的にメッセージを 読み取ります。 メッセージ・プロパティーを、 ビジネス・データまた状況情報を (アプリケーション・データ内への格納の必要なしで) 保管するのに使用できます。 アプリケーション は、MQ メッセージ記述子 (MQMD) または MQRFH2 ヘッダー内の データにアクセスする必要はありません。なぜなら、これらのデータ構造体内のフィールドには、メッセージ・キュー・インターフェース (MQI) 機能呼び出しを使用してメッセージ・プロパティーとしてアクセスできるからです。

IBM MQ でのメッセージ・プロパティーの使用は、JMS でのプロパティーの使用によく似ています。 これは、JMS アプリケーションでプロパティーを設定して、それらのプロパティーをプロシージャー型 IBM MQ アプリケーションで取り出すことができる (逆も可) ことを意味します。 あるプロパティーを JMS アプリケーションで使用可能にするには、そのプロパティーに 接頭部「usr」を割り当てます。そうすると、そのプロパティーは (接頭部なしで) JMS メッセージ・ユーザー・プロパティーとして使用可能になります。 例えば、 IBM MQ プロパティー *usr.myproperty* (文字ストリング) は、 JMS 呼び出し `message.getStringProperty('myproperty')` を使用して JMS アプリケーションからアクセスできます。 接頭部「usr」の付いたプロパティーに 2 つ以上の U+002E (".") 文字が含まれている場合、JMS アプリケーションでそのプロパティーにアクセスできないことに注意してください。 接頭部がなく U+002E (".") 文字もないプロパティーは、 接頭部「usr」が付いているかのように扱われます。 逆に、 JMS アプリケーションで設定されたユーザー・プロパティーには、 IBM MQ アプリケーションで「usr」を追加することによってアクセスできます。 MQINQMP 呼び出しで照会されたプロパティー名の接頭部。

メッセージ・プロパティーとメッセージ長

キュー・マネージャー属性 *MaxPropertiesLength* を使用すると、IBM MQ キュー・マネージャーで任意のメッセージに付加して流すことができるプロパティーのサイズを制御できます。

通常、MQSETMP を使ってプロパティーを設定する場合、プロパティーのサイズは、MQSETMP 呼び出しに渡される際のプロパティー名の長さ (バイト) とプロパティー値の長さ (バイト) の合計になります。 プロパティー名およびプロパティー値の文字セットは、Unicode への変換が原因で、宛先へのメッセージ伝送中に変更される可能性があります。 その場合、プロパティーのサイズが変更されることがあります。

MQPUT または MQPUT1 呼び出し時、キューおよびキュー・マネージャーにおいて、メッセージのプロパティーはメッセージの長さにカウントされません。 しかし、キュー・マネージャーによって認識される際にプロパティーの長さにカウントされます (メッセージ・プロパティー MQI 呼び出しを使って設定されたかどうかにかかわらず)。

プロパティーのサイズが最大プロパティー長を超える場合、メッセージは MQRC_PROPERTIES_TOO_BIG として拒否されます。 プロパティーのサイズは表記に依存しているため、全体レベルで最大プロパティー長を設定する必要があります。

プロパティーが含まれるバッファーの場合、 アプリケーションは *MaxMsgLength* の値より大きいバッファーを使用してメッセージを正常に書き込むことができます。 なぜなら、MQRFH2 エレメントとして表されている場合であっても、メッセージ・プロパティーはメッセージの長さにカウントされないからです。 MQRFH2 ヘッダー・フィールド がプロパティーの長さに加えられるのは、1 つ以上のフォルダーが含まれていて、ヘッダー内のすべてのフォルダーが プロパティーを含んでいる場合のみです。 MQRFH2 ヘッダーに 1 つ以上のフォルダーが含まれていて、 どのフォルダーもプロパティーを含んでいない場合、代わりに MQRFH2 ヘッダー・フィールドが メッセージ長に加算されます。

MQGET 呼び出し時、メッセージのプロパティーは、キューおよびキュー・マネージャーに関する限りメッセージの長さにカウントされません。 しかし、プロパティーは別にカウントされるため、MQGET 呼び出しにより戻されるバッファーを *MaxMsgLength* 属性の値より大きくすることができます。

MQGET の呼び出し前に、アプリケーションが *MaxMsgLength* の値を照会し、このサイズのバッファーを割り振ることがないようにしてください。その代わりに、十分な大きさのバッファーを割り振ります。 MQGET が失敗する場合、*DataLength* パラメーターのサイズで指定されるバッファーを割り振ります。

MQGMO 構造体でメッセージ・ハンドルが指定されていない場合、MQGET 呼び出しの *DataLength* パラメーターは、アプリケーション・データおよび提供されたバッファーに戻された全プロパティーの長さ (バイト) を戻します。

MQPUT 呼び出しの *Buffer* パラメーターには、送信されるアプリケーション・メッセージ・データ、およびメッセージ・データ内に示される全プロパティーが含まれます。

各メッセージのメッセージ記述子またはエクステンションを除き、メッセージ・プロパティーには 100 MB の長さ制限があります。

内部表記におけるプロパティーのサイズは、名前の長さ、値のサイズ、およびプロパティーの制御データの和です。 また、プロパティー 1 つがメッセージに追加されると、プロパティーのセット用の制御データも含まれます。

プロパティー名

プロパティー名は、文字ストリングです。 長さと使用できる文字セットには、一定の制限が 適用されます。

プロパティー名は、大/小文字の区別がある文字ストリングで、コンテキストによって特に制限されない限り +4095 文字に制限されています。 この制限は MQ_MAX_PROPERTY_NAME_LENGTH 定数に含まれています。

メッセージ・プロパティー MQI 呼び出しを使用する際にこの最大長を超えた場合、呼び出しは失敗して理由コード MQRC_PROPERTY_NAME_LENGTH_ERR が出されます。

JMS にはプロパティー名の最大長はないため、JMS アプリケーションが設定する有効な JMS プロパティー名が、MQRFH2 構造体に格納されるときには有効な IBM MQ プロパティー名でなくなる可能性があります。

この場合、解析時にプロパティー名の最初の 4095 文字のみが使用され、残りの文字は切り捨てられます。この処理の結果、セレクターを使用するアプリケーションは、選択ストリングとの一致、または予期しない場合のストリングとの一致に失敗するおそれがあります。これは切り捨てにより、複数のプロパティーが同じ名前になる場合があるためです。 プロパティー名が切り捨てられる際、WebSphere® MQ はエラー・ログ・メッセージを発行します。

プロパティー名はすべて、Java 言語仕様で定義された Java ID に関する規則に準拠していなければなりません。ただし Unicode 文字 U+002E (.) は例外で、名前の一部として使用できますが、先頭には使用できません。 Java ID の規則は、JMS 仕様に含まれるプロパティー名に関する規則と同じです。

空白文字および比較演算子は禁止されています。 プロパティー名にヌルの埋め込みは可能ですが、推奨されていません。 ヌルを埋め込むと、可変長ストリングを指定するために MQCHARV 構造体と共に使用する際、MQVS_NULL_TERMINATED 定数を使用できなくなります。

プロパティー名は単純なものにしてください。なぜなら、アプリケーションはプロパティー名に基づいてメッセージを選択できますが、名前およびセレクターの文字セット間の変換により、予期せず選択に失敗するおそれがあるためです。

IBM MQ のプロパティー名では、文字 U+002E (.) を使用してプロパティーを論理的に分類します。 こうして、プロパティーのために名前空間が配分されます。 以下の接頭部が付いたプロパティーは、製品で使用するため、すべての大文字小文字の組み合わせで予約されています。

- `mcd`
- `jms`
- `usr`
- `mq`
- `sib`

- wmq
- Root
- Body
- Properties

名前の衝突を効果的に回避するには、すべてのアプリケーションにおいてインターネットのドメイン・ネームを接頭部としてメッセージ・プロパティーに付けます。 例えば ourcompany.com というドメイン・ネームを使用するアプリケーションを開発している場合、接頭部 com.ourcompany を使用してプロパティーすべてに名前を付けます。 また、この命名規則によりプロパティーの選択が容易になります。例えば、アプリケーションは、com.ourcompany.% で始まるすべてのメッセージ・プロパティーに対して照会を実行できます。

プロパティー名の使用に関する詳細については、『プロパティー名の制約事項』を参照してください。

プロパティー名の制約事項

プロパティーに名前を付けるときには、一定の規則を守らなければなりません。

以下の制約事項が、プロパティー名に適用されます。

1. プロパティーを以下のストリングで始めてはなりません。
   - "JMS" - IBM MQ classes for JMS で使用するために予約済みです。
   - "usr.JMS" - 無効です。

   例外は、JMS プロパティーの同義語として機能する以下のプロパティーのみです。

Property	**(の) 同義語**
JMSCorrelationID	Root.MQMD.CorrelId または jms.Cid
JMSDeliveryMode	Root.MQMD.Persistence または jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root.MQMD.Expiry または jms.Exp
JMSMessageID	Root .MQMD.MsgId
JMSPriority	Root.MQMD.Priority または jms.Pri
JMSRedelivered	Root .MQMD.BackoutCount
JMSReplyTo (URI としてエンコードされたストリング)	Root.MQMD.ReplyToQ または Root.MQMD.ReplyToQMgr または jms.Rto
JMSTimestamp	Root.MQMD.PutDate または Root.MQMD.PutTime または jms.Tms
JMSType	mcd.Type または mcd.Set または mcd.Fmt
JMSXAppID	Root .MQMD.PutApplName
JMSXDeliveryCount	Root .MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId または jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber または jms.Seq
JMSXUserID	Root .MQMD.UserIdentifier

これらの同義語により、MQI アプリケーションは IBM MQ classes for JMS クライアント・アプリケーションと同様の方法で JMS プロパティーにアクセスできます。 これらのプロパティーのうち、JMSCorrelationID、JMSReplyTo、JMSType、JMSXGroupID、 および JMSXGroupSeq だけが、MQI を使用して設定可能です。

なお、IBM MQ classes for JMS 内から利用できる JMS_IBM_* プロパティーは、MQI を使用して利用可能ではありません。 JMS_IBM_* プロパティーが参照するフィールドには、MQI アプリケーションにより他の方法でアクセスできます。

2. 大文字小文字の組み合わせ方を問わず、プロパティーを次の名前で呼んではなりません。「NULL」、「TRUE」、「FALSE」、「NOT」、「AND」、「OR」、「BETWEEN」、「LIKE」、「IN」、「IS」、および「ESCAPE」。これらは、選択ストリングで使用される SQL キーワードの名前です。
3. " で始まるプロパティー名 MQ " "mq_usr" の先頭以外で小文字または大文字が混在している場合は、1 つの "." のみを含めることができます。 文字 (U+002E)。 複数の「.」 文字は、それらの接頭部を持つプロパティーでは許可されません。
4. 2 "." 文字の間に他の文字が含まれている必要があります。階層内で空のポイントを使用することはできません。 同様に、プロパティー名の末尾を「.」にすることはできません。 文字。
5. アプリケーションによりプロパティー「a.b」次いでプロパティー「a.b.c」が設定されると、階層内で「b」に値あるいは他の論理グループがあるのか不明確になります。 こうした階層は「混合コンテンツ」であり、サポートされていません。 混合コンテンツを引き起こすプロパティーの設定は許可されていません。

これらの制約事項は、以下の妥当性検査方式により適用されます。

- メッセージ・ハンドルが作成された場合、妥当性検査が要求されると、MQSETMP-メッセージ・プロパティーの設定 呼び出しを使用してプロパティーを設定する際にプロパティー名が検証されます。 プロパティーの妥当性検査が試行され、プロパティー名の指定の誤りが原因で不合格となる場合、完了コードは MQCC_FAILED で、以下の理由を伴います。
  - 理由 1 から 4 の場合、MQRC_PROPERTY_NAME_ERROR
  - 理由 5 の場合、MQRC_MIXED_CONTENT_NOT_ALLOWED
- MQRFH2 エレメントとして直接指定されたプロパティーの名前については、MQPUT 呼び出しにより妥当性検査が行われるとは限りません。

プロパティーとしてのメッセージ記述子フィールド
ほとんどのメッセージ記述子フィールドは、プロパティーとして扱うことができます。 プロパティー 名は、メッセージ記述子フィールドの名前に接頭部を追加することで構成されます。

MQI アプリケーションにおいて、メッセージ記述子フィールドに含まれるメッセージ・プロパティーを識別したい場合 (例えば、セレクター・ストリングで、またはメッセージ・プロパティー API の使用において) は、以下の構文を使用してください。

プロパティー名	メッセージ記述子フィールド
Root.MQMD.フィールド	フィールド

Field を C 言語宣言の MQMD 構造体フィールドのケースと同様に指定します。 例えば、プロパティー名 `Root.MQMD.AccountingToken` の場合、メッセージ記述子の `AccountingToken` フィールドにアクセスします。

メッセージ記述子の `StrucId` フィールドおよび `Version` フィールドは、上記の構文を使用してアクセスすることはできません。

他のプロパティーについては、メッセージ記述子フィールドが MQRFH2 ヘッダー内に表されることはありません。

キュー・マネージャーに受け入れられる MQMDE でメッセージ・データが始まる場合、`Root.MQMD.`*`Field`* 表記を使って MQMDE フィールドにアクセスできます。 この場合 MQMDE フィールドは、プロパティーの観点で論理上 MQMD の一部として扱われます。 MQMDE の概要を参照してください。

プロパティーのデータ型と値

プロパティーは、ブール、バイト・ストリング、文字ストリング、 浮動小数点、または整数です。 コンテキストによる制限が特にない限り、プロパティーにはデータ型の範囲内で任意の有効な値を格納できます。

プロパティー値のデータ型は、以下のいずれかの値でなければなりません。

- MQBOOL
- MQBYTE[ ]
- MQCHAR[ ]
- MQFLOAT32
- MQFLOAT64
- MQINT8
- MQINT16
- MQINT32
- MQINT64

定義された値を持たないプロパティーが存在できます。それはヌル・プロパティーです。 ヌル・プロパティーは、定義済みでも空の値、つまり、長さ 0 の値を持つという点で、バイト・プロパティー (MQBYTE[ ]) や文字ストリング・プロパティー (MQCHAR[ ]) とは異なります。

バイト・ストリングは、JMS および XMS において有効なプロパティーのデータ型ではありません。 *usr* フォルダーでバイト・ストリングのプロパティーを使用しないようにしてください。

キューからのメッセージの選択

MQGET 呼び出しで MsgId フィールドと CorrelId フィールド を使用するか、MQOPEN または MQSUB 呼び出しで SelectionString を使用して、キューからメッセージを選択することができます。

Selectors

メッセージ・セレクターは、アプリケーションがインタレストを登録するために使用する可変長のストリングです。この選択ストリングが表している構造化照会言語 (SQL) 照会に合致するプロパティーをもつメッセージだけに絞り込まれます。

MQSUB および MQOPEN 関数呼び出しを使用する選択

SelectionString (MQCHARV タイプの構造体) を使用して、MQSUB および MQOPEN 呼び出しによる選択を行います。

SelectionString 構造体は、可変長の選択ストリングをキュー・マネージャーに渡すために使用されます。

セレクター・ストリングに関連付けられた CCSID が、MQCHARV 構造体の VSCCSID フィールドを介して設定されます。 使用される値は、セレクター・ストリングでサポートされる CCSID でなければなりません。サポートされるコード・ページのリストは、コード・ページ変換を参照してください。

CCSID の指定時に、その CCSID のために IBM MQ 対応の Unicode 変換が提供されていないと、MQRC_SOURCE_CCSID_ERROR のエラーとなります。 このエラーは、セレクターがキュー・マネージャーに渡されるとき、すなわち MQSUB、MQOPEN、または MQPUT1 呼び出し時に戻されます。

VSCCSID フィールドのデフォルト値は、MQCCSI_APPL です。それは、選択ストリングの CCSID が、キュー・マネージャー CCSID またはクライアント CCSID (クライアント経由で接続する場合) と等しいことを示します。 しかし MQCCSI_APPL 定数は、コンパイル前に再定義を行うアプリケーションによりオーバーライド可能です。

MQCHARV セレクターがヌル・ストリングを表す場合、そのメッセージ・コンシューマーのために選択は実行されず、セレクターが使用されなかったかのようにメッセージが送信されます。

選択ストリングの最大長は、MQCHARV フィールドの *VSLength* が表す内容によってのみ制限されます。

バッファーが用意されていて、VSBufSize に正のバッファー長が入っている場合、 MQSO_RESUME サブスクライブ・オプションを使用する MQSUB 呼び出しからの出力に SelectionString が戻されます。 バッファーが提供されていない場合、選択ストリングの長さのみが MQCHARV の VSLength フィールドに戻されます。 フィールドを返すのに必要なスペースよりも提供されたバッファーが小さい場合、VSBufSize バイトのみがそのバッファーに戻されます。

アプリケーションは、まずキュー (MQOPEN の場合) またはサブスクリプション (MQSUB の場合) に対するハンドルをクローズしてからでなければ、選択ストリングを変更できません。 その後であれば、新しい選択ストリングを後続の MQOPEN または MQSUB 呼び出しに指定できます。

MQOPEN
MQCLOSE を使用し、オープンされたハンドルをクローズします。その後、次の MQOPEN 呼び出し時に選択ストリングを新規に指定します。

MQSUB
MQCLOSE を使用し、戻されたサブスクリプション・ハンドル (hSub) をクローズします。その後、次の MQSUB 呼び出し時に選択ストリングを新規に指定します。

32 ページの図 3 には、MQSUB 呼び出しを使用する選択のプロセスが示されています。

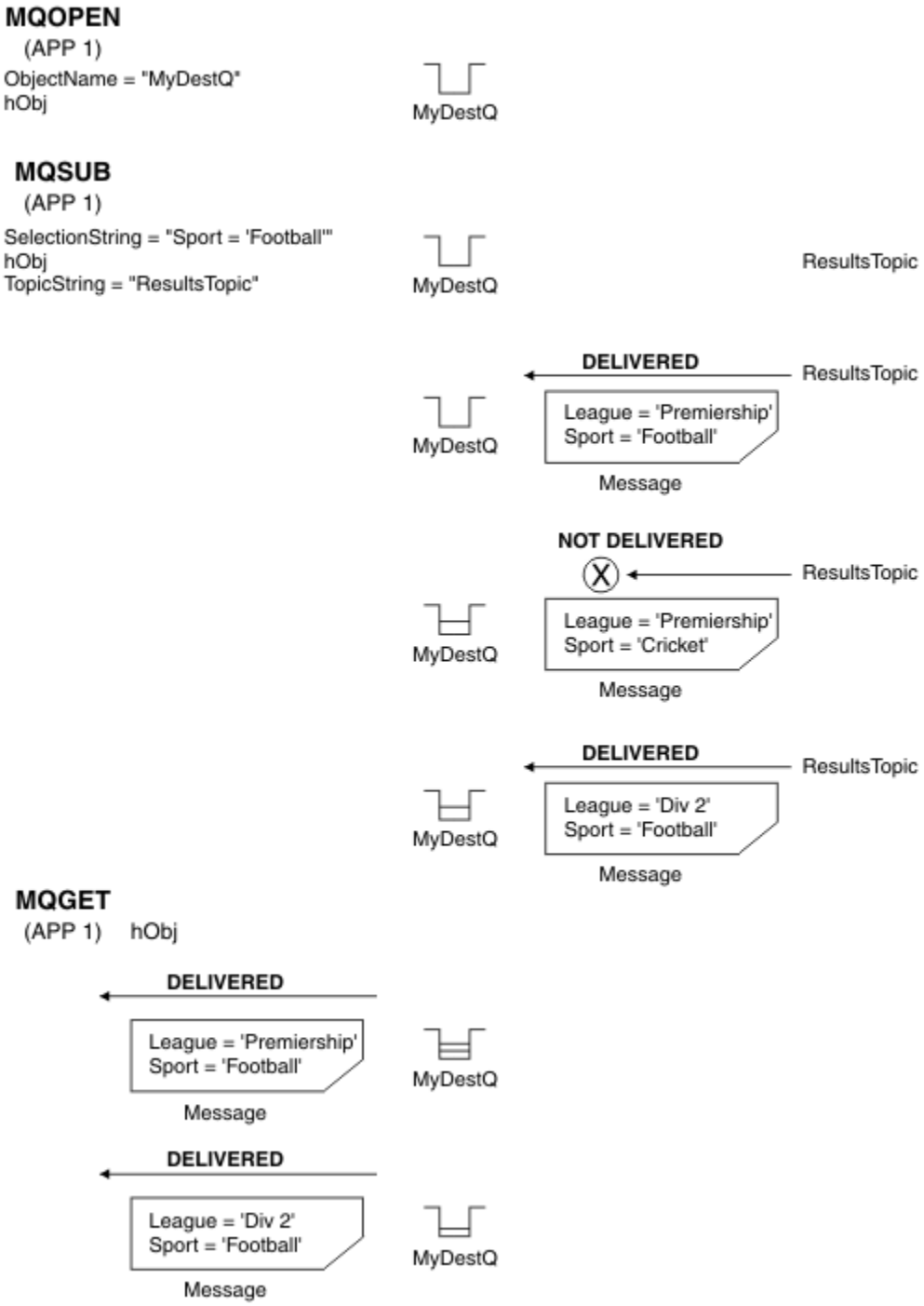


図 *3. MQSUB* 呼び出しを使用する選択

MQSD 構造体の *SelectionString* フィールドを使用することにより、呼び出し時にセレクターが MQSUB に渡されます。 MQSUB にセレクターを渡した結果、サブスクライブされているトピックにパブリッシュされたメッセージのうち、提供された選択ストリングと一致するメッセージのみが、宛先キューで使用可能となります。

33 ページの図 4 には、MQOPEN 呼び出しを使用する選択のプロセスが示されています。

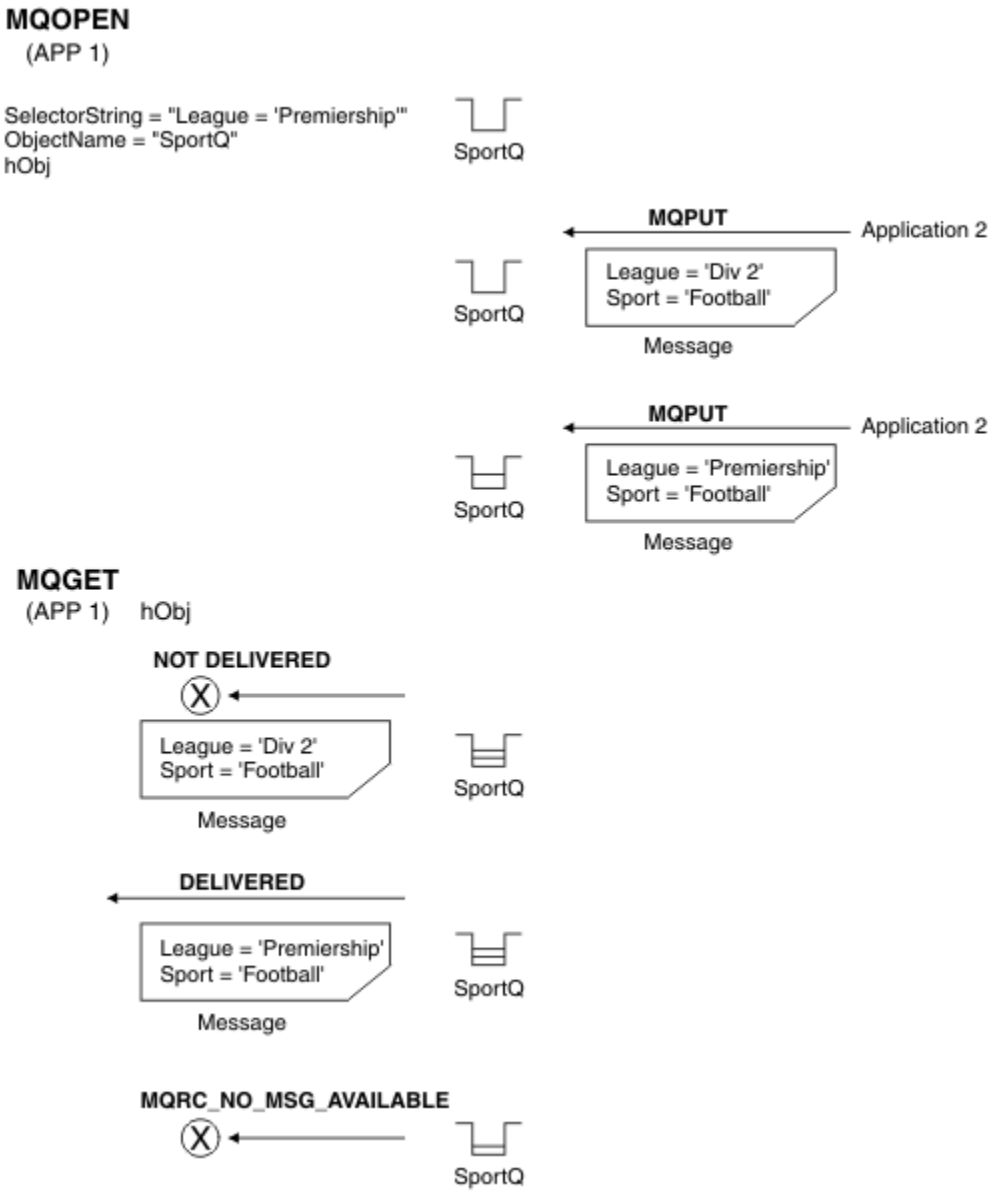


図 *4. MQOPEN* 呼び出しを使用する選択

MQOD 構造体の *SelectionString* フィールドを使用することにより、呼び出し時にセレクターが MQOPEN に渡されます。 MQOPEN 呼び出し時にセレクターを渡した結果、オープンされたキューにあるメッセージのうち、セレクターと一致するメッセージのみがメッセージ・コンシューマーに送信されます。

MQOPEN 呼び出し時にセレクターを使用するのは、主に point-to-point の場合です。この場合アプリケーションは、キューに入っているメッセージのうち、セレクターと一致するメッセージのみ選んで受信できます。 上記の例は、MQOPEN によって開かれたキュー上に 2 つのメッセージが書き込まれた場合に、セレクターに合致するメッセージ 1 つだけをアプリケーションが受け取り、それがセレクターに合致する唯一のものであるという、単純なシナリオを示しています。

所定のセレクターと一致するメッセージはこれ以上キューに存在しないため、後続の MQGET 呼び出しの結果は MQRC_NO_MSG_AVAILABLE となります。

関連概念

40 ページの『選択ストリングの規則と制約事項』

選択ストリングがどのように解釈されるのかに関する規則および文字に関する制約事項を よく理解することで、セレクターを使用する際に発生しうる問題を回避できます。

選択動作

IBM MQ の選択動作に関する概要

MQMD が以下の条件に当てはまる場合、MQMDE 構造体のフィールドは、対応するメッセージ記述子プロパティーのためのメッセージ・プロパティーとみなされます。

- MQFMT_MD_EXTENSION フォーマットがある。
- 直後に有効な MQMDE 構造体が続く。
- バージョン 1 である。または、デフォルトのバージョン 2 フィールドのみを含む。

メッセージ・プロパティーに対する突き合わせが行われる前に、選択ストリングを TRUE または FALSE に解決することが可能です。 例えば、選択文字列が "TRUE <> FALSE"に設定されている場合などです。 こうした早期の評価の実行が保証されるのは、選択ストリング内にメッセージ・プロパティー参照がない場合のみです。

メッセージ・プロパティーが調べられる前に選択ストリングが TRUE に解決される場合、コンシューマーによりサブスクライブされたトピックにパブリッシュされた全メッセージが配信されます。 メッセージ・プロパティーが調べられる前に選択ストリングが FALSE に解決される場合、セレクターを提供した関数の呼び出し時に、理由コードの MQRC_SELECTOR_ALWAYS_FALSE と完了コードの MQCC_FAILED が戻されます。

メッセージにメッセージ・プロパティー (ヘッダー・プロパティーを除く) が含まれていない場合でも、それは選択の対象となり得ます。 存在しないメッセージ・プロパティーを選択ストリングが参照している場合、このプロパティーはヌルまたは「不明」な値を持つものと見なされます。

例えば、メッセージが 'Color IS NULL'のような選択ストリングを満たす場合でも、 'Color' はメッセージ内のメッセージ・プロパティーとして存在しません。

選択は、メッセージに関連付けられたプロパティーに対してのみ実行可能で、メッセージそのものに対しては実行できません。ただし、拡張メッセージ選択プロバイダーが使用可能である場合は除きます。 拡張メッセージ選択プロバイダーが使用可能である場合にのみ、 メッセージ・ペイロードで選択を実行できます。

各メッセージ・プロパティーには、1 つのタイプが関連付けられています。 選択の実行時に、メッセージ・プロパティーを検査するための式で使用される値が、適切なタイプであることを確認する必要があります。タイプの不一致が起こると、問題の式は FALSE に解決します。

ユーザーの責任において、選択ストリングとメッセージ・プロパティーで互換タイプが使用されるようにしてください。

選択基準は、非アクティブな永続サブスクライバーのために引き続き適用されます。こうして、最初に提供された選択ストリングと一致するメッセージのみが保持されます。

永続サブスクリプションが変更 (MQSO_ALTER) により再開される際、選択ストリングは変更できません。永続サブスクライバーのアクティビティー再開時に別の選択ストリングが提供されると、MQRC_SELECTOR_NOT_ALTERABLE がアプリケーションに戻されます。

選択基準を満たすメッセージがキューにない場合、アプリケーションは MQRC_NO_MSG_AVAILABLE の戻りコードを受け取ります。

プロパティー値を含む選択ストリングがアプリケーションにより指定されている場合、一致するプロパティーを含むメッセージのみが選択のために適格です。 例えば、サブスクライバーが "a = 3" という選択ストリングを指定していて、プロパティーを含まないメッセージがパブリッシュされるか、' a' が存在しないか、または 3 と等しくないプロパティーが公開されているとします。 サブスクライバーは、そのメッセージを宛先キューに受信しません。

メッセージング・パフォーマンス

キューからメッセージを選択する場合は、IBM MQ がキューにある各メッセージを順番に検査する必要があります。 メッセージの検査は、選択基準に一致するメッセージが検出されるか、検査対象のメッセージがなくなるまで続けられます。 そのため、深いキューでメッセージの選択が使用された場合は、メッセージのパフォーマンスが低下します。

メッセージの選択が JMSCorrelationID または JMSMessageID に基づく場合に、深いキューでのメッセージの選択を最適化するには、以下の形式の選択ストリングを使用します。

- JMSCorrelationID ='ID:*correlation_id*'
- JMSMessageID='ID:*message_id*'

ここで、

- *correlation_id* は、標準の IBM MQ 相関 ID を含むストリングです。
- *message_id* は、標準の IBM MQ メッセージ ID を含むストリングです。

注 : セレクターはプロパティーの 1 つのみを参照する必要があります。 上記のいずれかの形式のセレクターを使用すると、JMSCorrelationID に基づいて選択を行う際のパフォーマンスが大幅に改善されます。また、JMSMessageID の場合も、パフォーマンスがわずかに向上します。 詳細については、146 ページの『JMS のメッセージ・セレクター』を参照してください。

複雑なセレクターの使用

セレクターに多数のコンポーネントが含まれる場合があります。例:

a および b または c および d または e および f または g および h または i および j... または y および z

このような複雑なセレクターを使用すると、パフォーマンスへの影響が深刻になったり、リソース要件が過大になったりする場合があります。 したがって、IBM MQ は、あまりにも複雑でシステム・リソース不足を招き得るセレクターを処理しないことで、システムを保護します。 保護が起こり得るのは、選択ストリングに含まれるテストが 100 を超える場合か、オペレーティング・システムのスタックのサイズ制限に達しようとしていることを IBM MQ が検出した場合です。 多数のコンポーネントが含まれる選択ストリングを使用する場合は、適切なプラットフォームで十分に試行とテストを行い、保護の制限に達することがないようにしてください。

括弧を追加してコンポーネントをまとめることによりセレクターを単純化すると、その複雑さが軽減して、パフォーマンスが改善される可能性があります。 以下に例を示します。

(a と b または c と d) または (e と f または g と h) または (i と j) ...

関連概念

40 ページの『選択ストリングの規則と制約事項』
選択ストリングがどのように解釈されるのかに関する規則および文字に関する制約事項を よく理解することで、セレクターを使用する際に発生しうる問題を回避できます。

メッセージ・セレクター構文

IBM MQ メッセージ・セレクターはストリングであり、その構文は SQL92 条件式構文のサブセットに基づいています。

メッセージ・セレクターが評価される順序は、優先順位内で左から右です。 括弧を使用してこの順序を変更できます。 定義済みのセレクター・リテラルおよび演算子名は、大文字でここに書き込まれます。ただし、これらには大/小文字の区別がありません。

セレクターが API を介して指定される場合、IBM MQ は、メッセージ・セレクターが提示された時点で、その構文が正しいかどうかを検証します。 選択文字列の構文が正しくないか、プロパティー名が無効であり、拡張メッセージ選択プロバイダーが使用できない場合は、 MQRC_SELECTION_NOT_AVAILABLE がアプリケーションに戻されます。 選択ストリングの構文が正しくないか、 プロパティー名が有効でない場合、サブスクリプションが再開されると、 アプリケーションに MQRC_SELECTOR_SYNTAX_ERROR が戻されます。 プロパティーが設定されたときに (MQCMHO_VALIDATE の代わりに MQCMHO_NONE を設定することによって) プロパティー名の妥当性検査が 使用不可に設定されていた場合、その後にアプリケーション

が、無効なプロパティー名を使うメッセージを書き込むと、 そのメッセージは決して選択されることはありません。

DISPLAY SUB パラメーター **SELTYPE** の値 EXTENDED で示されるように、管理上定義されたサブスクリプション・セレクターが拡張メッセージ構文を使用していることを IBM MQ が判別した場合、セレクターの提示時にエラーは返されません。 この場合、選択文字列の構文検査は、公開時間まで据え置かれます (MQRC_SELECTION_NOT_AVAILABLE を参照) 。

セレクターには、以下のものを含めることができます。

- リテラル:
  - ストリング・リテラルは、単一引用符で囲まれます。 連続した 2 つの単一引用符は、単一引用符を表します。 例は、'literal' および 'literal''s' です。 Java ストリング・リテラルと同様に、これらは Unicode 文字エンコードを使用します。 二重引用符を使用してストリング・リテラルを囲むことはできません。 単一引用符に囲まれた中では、任意のバイトのシーケンスを使用できます。
  - バイト・ストリングは、二重引用符で囲まれた 1 対以上の 16 進数文字で、接頭部が 0x です。 例えば、"0x2F1C" や "0XD43A" などです。 バイト・ストリングの長さは、少なくとも 1 バイト必要です。 セレクターのバイト・ストリングをタイプ MQTYPE_BYTE_STRING のメッセージ・プロパティーに突き合わせるとき、先行ゼロまたは後続ゼロについては特別な処置は行われません。 そのバイトは別の文字として扱われます。 エンディアン種別も 考慮されません。 セレクターのバイト・ストリングとプロパティーのバイト・ストリングの両方の長さは等しくなければならず、バイトのシーケンスは同じでなければなりません。

    一致するバイト・ストリング選択の例を以下に示します (*myBytes* = 0AFC23 と想定)。

    - "myBytes = "0x0AFC23"" = TRUE

    以下のストリング選択は一致しません。

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (バイト数が 2 の倍数ではないため)
    - "myBytes = "0x0AFC2300"" = FALSE (比較において後続ゼロが有意であるため)
    - "myBytes = "0x000AFC23"" = FALSE (比較において先行ゼロが有意であるため)
    - "myBytes = "0x23FC0A"" = FALSE (エンディアンネスは考慮されないため)
  - 16 進数はゼロで始まり、大文字または小文字の x が続きます。 リテラルの残りの部分は、1 つ以上の有効な 16 進数文字となります。 例は、0xA、0xAF、0X2020 です。
  - 先行ゼロの後に 0-7 の範囲の 1 つ以上の数字が続いているものは、 常に 8 進数の開始と解釈されます。 接頭部がゼロの 10 進数で表すことはできません。例えば 09 の場合、9 は有効な 8 進数字ではないため構文エラーが戻されます。 8 進数字の例は、0177、0713 です。
  - 厳密な数値リテラルは、小数点のない数値 (57、 -957、 +62 など) です。 正確な数字リテラルには、末尾に大文字または小文字の L を付けることができます。これが数値の格納または解釈の方法に影響を与えることはありません。 IBM MQ は、-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までの範囲の正確な数字をサポートします。
  - 近似数値リテラルは、7E3 または -57.9E2 などの浮動小数における数値、または 7.、-95.7、または +6.2 などの小数部を持つ数値です。 IBM MQ は、-1.797693134862315E+308 から 1.797693134862315E+308 までの範囲の数値をサポートします。

    仮数部はオプションの符号文字 (+ または -) の後に 続きます。 仮数部は、整数または小数のいずれかです。 仮数部の小数部分には、先行桁は必要ありません。

    大文字または小文字の E は、オプションの指数の開始を示します。 指数の基数は 10 進であり、 指数の数字部分にはオプションで符号文字を接頭部に付ける ことができます。

    近似数値リテラルは、文字 F または D (大/小文字の区別はない) で終了できます。 この構文は、短精度または倍精度の数値をタグ付けするための、多言語に対応する方式を サポートするために存在しています。 これらの文字はオプションであり、 近似数値リテラルがどのように格納または処理されるのかには影響 しません。 これらの数値は常に倍精度を使用して格納および処理されます。

– ブール・リテラルの TRUE および FALSE。

注: 非有限 IEEE-754 表記 (NaN、+Infinity、-Infinity など) は、 選択ストリング中ではサポートされません。 従って、これらの値を式の中でオペランドとして 使用することはできません。 負のゼロは、数学演算では正のゼロと して扱われます。

- ID:

  ID は可変長文字シーケンスで、必ず有効な ID 開始文字で始まり、ゼロまたはそれ以上の有効な ID 部分文字が続きます。 ID 名の規則は、メッセージ・プロパティー名の規則と同じです。詳細については、28 ページの『プロパティー名』 および 29 ページの『プロパティー名の制約事項』 を参照してください。

  注: 拡張メッセージ選択プロバイダーが使用可能である場合にのみ、メッセージ・ペイロードで選択を実行できます。

  ID は、ヘッダー・フィールド参照またはプロパティー参照のいずれかです。 メッセージ・セレクター内のプロパティー値のタイプは、プロパティーの設定に使用されるタイプに対応していなければなりません。ただし、可能な場合に数値プロモーションが実行されます。 タイプの不一致が起きると、式の結果は FALSE となります。 メッセージ内に存在しないプロパティーが参照されると、その値は NULL となります。

  プロパティーのゲット・メソッドに適用する型変換は、メッセージ・セレクター式でプロパティーが使用されている場合には適用されません。 例えば、ストリング値としてプロパティーを設定し、それを数値として照会するためにセレクターを使用すると、式は FALSE を戻します。

  プロパティー名または MQMD フィールド名にマップされる JMS フィールド名およびプロパティー名も、選択ストリング内の有効な ID です。 IBM MQ は、認識された JMS フィールドおよびプロパティー名 をメッセージ・プロパティー値にマップします。 詳しくは、146 ページの『JMS のメッセージ・セレクター』を参照してください。 例えば、選択ストリング "JMSPriority >=" は、現行メッセージの jms フォルダーにある 「1 次」 プロパティーを選択します。

- オーバーフロー/アンダーフロー:

  10 進数および近似数値の両方で、 次の条件は定義されていません。

  – 定義された範囲に入っていない数の指定
  – オーバーフローまたはアンダーフローを引き起こす可能性のある算術式の指定

  これらの条件は検査されません。

- 空白文字:

  スペース、用紙送り、改行、復帰、水平タブ、または垂直タブとして定義されます。 以下の Unicode 文字は空白文字として認識されます。

  – \u0009 to \u000D
  – \u0020
  – \u001C
  – \u001D
  – \u001E
  – \u001F
  – \u1680
  – \u180E
  – \u2000 から \u200A へ
  – \u2028
  – \u2029
  – \u202F
  – \u205F
  – \u3000

- 式:
  - セレクターは、条件式です。 true に評価されるセレクターは一致し、false または unknown に評価されるセレクターは一致しません。
  - 演算式は、それ自体と、算術演算、ID (ID 値は数値リテラルとして扱われる)、および数値リテラルから構成されています。
  - 条件式は、それ自体と、比較演算、および論理演算から構成されています。
- 標準の括弧 () (式が評価される順序を設定する) がサポートされています。
- 論理演算子 (優先順位どおりに列挙): NOT、AND、OR。
- 比較演算子: =、>、>=、<、<=、<>(等しくない)。
  - 2 つのバイト・ストリングが等しいのは、両ストリングが同じ長さで、バイトのシーケンスが等しい場合のみ です。
  - 同じタイプの値のみを比較できます。 唯一の例外は、正確な数値と近似数値の比較が有効であることです (必要な型変換は Java 数値上位変換の規則によって定義されます)。 異なるタイプを比較する試みがある場合、セレクターは常に false です。
  - ストリングとブールの比較は、= および <> に制限されます。 2 つのストリングは、それらのストリングに含まれている文字シーケンスがまったく同じ場合にのみ等しくなります。
- 算術演算子 (優先順位どおりに列挙):
  - 単項 +、 -。
  - 乗算 *、および除算 /。
  - 加算 +、および減算 -。
  - ヌル値での算術演算はサポートされていません。 ヌル値での算術演算が試行される場合、完全セレクターは常に false です。
  - 算術演算は、Java 数値プロモーションを使用しなければなりません。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比較演算子:
  - `Age BETWEEN 15 and 19` は `age>= 15 AND age <= 19` と同等です。
  - `Age NOT BETWEEN 15 and 19` は `age < 15 OR age > 19` と同等です。
  - BETWEEN 演算の式のいずれかがヌルである場合、演算の値は false です。 NOT BETWEEN 演算の式のいずれかが NULL である場合、演算の値は true です。
- ID がストリング値または NULL 値を持つ場合の ID `[NOT] IN (string-literal1, string-literal2,...)` 比較演算子。
  - `Country IN ('UK', 'US', 'France')` は `'UK'` の場合には true であり、`'Peru'` の場合には false です。 これは、式 `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')` と同等です。
  - `Country NOT IN ('UK', 'US', 'France')` は `'UK'` の場合には false であり、`'Peru'` の場合には true です。 これは、式 `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))` と同等です。
  - IN または NOT IN 演算の ID がヌルである場合、演算の値は不明です。
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` 比較演算子。 identifier にはストリング値があります。 *pattern-value* はストリング・リテラルです。ここで、_ は単一文字を表しており、% は文字シーケンス (空のシーケンスを含む) を表しています。 その他のすべての文字はそれ自体を表しています。 オプションの *escape-character* は単一の文字ストリング・リテラルです。この文字は、*pattern-value* 内の _ および % の特殊な意味をエスケープするために使用されます。 LIKE 演算子は、2 つのストリング値の比較のみに使用する必要があります。
  - `phone LIKE '12%3'` は、123 および 12993 の場合には true で、1234 の場合には false です。
  - `word LIKE 'l_se'` は、lose の場合には true で、loose の場合には false です。

- underscored LIKE '_%' ESCAPE '\' は _foo の場合には true であり、bar の場合には false です。
- phone NOT LIKE '12%3' は、123 および 12993 の場合には false で、1234 の場合には true です。
- LIKE 操作または NOT LIKE 操作の ID が NULL である場合、操作の値は不明です。

注: LIKE 演算子は、2 つのストリング値の比較のために使用する必要があります。 Root.MQMD.CorrelId の値は、文字ストリングではなく、24 バイトのバイト配列です。 セレクター・ストリング Root.MQMD.CorrelId LIKE 'ABC%' は、構文的に有効なものとしてパーサーは受け付けますが、false に評価されます。 このため、バイト配列を文字ストリングと比較するときには、LIKE を使用することはできません。

- identifier IS NULL 比較演算子は、 NULL ヘッダー・フィールド値、または欠落しているプロパティー値をテストします。
- identifier IS NOT NULL 比較演算子は、ヌル以外のヘッダー・フィールド値またはプロパティー値の存在をテストします。
- ヌル値

  NULL 値を含むセレクター式の評価は、以下のように SQL 92 NULL セマンティクスによって定義されます。

  - SQL は、 NULL 値を不明として扱います。
  - 不明値を持つ比較または算術は、常に、不明値を生じさせます。
  - IS NULL および IS NOT NULL 演算子は、不明値を TRUE および FALSE 値に変換します。

  ブール演算子は、3 つの値を持つロジックを使用します (T=TRUE、F=FALSE、U=UNKNOWN)。

表 *1.* ロジックが *A AND B* の場合のブール演算子の結果の値

演算子 A	**演算子 B**	**結果 (A AND B)**
T	F	F
T	U	U
T	T	T
F	T	F
F	U	F
F	F	F
U	T	U
U	U	U
U	F	F

表 *2.* ロジックが *A OR B* の場合のブール演算子の結果の値

演算子 A	**演算子 B**	**結果 (A OR B)**
T	F	T
T	U	T
T	T	T
F	T	T
F	U	U
F	F	F

表 2. ロジックが *A OR B* の場合のブール演算子の結果の値 (続き)		
演算子 A	**演算子 B**	**結果 (A OR B)**
U	T	T
U	U	U
U	F	U

表 3. ロジックが *NOT A* の場合のブール演算子の結果の値	
演算子 A	**結果 (NOT A)**
T	F
F	T
U	U

以下のメッセージ・セレクターは、メッセージ・タイプが car、色が blue、重量が 2500 lbs より大きいというメッセージを選択します。

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

SQL は固定小数点の比較および算術をサポートしますが、メッセージ・セレクターはサポートしません。これは、正確な数値リテラルが小数部を持たない数値リテラルに制限されるためです。 また同じ理由で、近似数値の代替表記として小数部を持つ数値があります。

SQL コメントは、サポートされていません。

関連概念

27 ページの『メッセージ・プロパティー』
メッセージ・プロパティーを使用すると、アプリケーションは、処理するメッセージを選択したり、MQMD または MQRFH2 ヘッダーにアクセスせずにメッセージに関する情報を取得する ことができます。 また、メッセージ・プロパティーは、IBM MQ と JMS アプリケーションの間の通信を行いやすくします。

関連資料

MsgHandle

MQBUFMH - バッファーからメッセージ・ハンドルへの変換

選択ストリングの規則と制約事項

選択ストリングがどのように解釈されるのかに関する規則および文字に関する制約事項を よく理解することで、セレクターを使用する際に発生しうる問題を回避できます。

- パブリッシュ/サブスクライブ・メッセージングのメッセージ選択は、パブリッシャーによって送信されるときにメッセージ上で行われます。 「文字列の選択」を参照してください。
- 単一の等号文字を使用して、等価であるかどうかがテストされます。例えば、a = b が正しく、a == b は正しくありません。
- 多くのプログラミング言語で使用される、「等しくない」ことを表す演算子は != です。 この表記は、<> の有効な同義語ではありません。例えば、a <> b は有効ですが、a != b は無効です。
- 単一引用符が認識されるのは、' (U+0027) 文字が使用されている場合のみです。 同様に、二重引用符はバイト・ストリングを囲むために使用される場合にのみ有効であり、" (U+0022) 文字を使用する必要があります。
- &、&&、|、および || といった記号は、論理積/論理和の同義語ではありません。たとえば、a && b を a AND b として指定する必要があります。
- ワイルドカード文字 * および ? は、% および _ の同義語ではありません。

- `20 < b < 30` などの複合式を含むセレクターは無効です。 演算子の優先順位が同じ場合、パーサーは左から右の順に評価します。 したがって、この例は `(20 < b) < 30` になりますが、これには意味がありません。 その代わり、式は `(b > 20) AND (b < 30)` として作成される必要があります。
- バイト・ストリングは二重引用符で囲む必要があります。単一引用符が使用された場合、バイト・ストリングはストリング・リテラルであると見なされます。 `0x` に続く文字の数 (文字が表す数ではない) は、2 の倍数でなければなりません。
- キーワード `IS` は等号の同義語ではありません。 したがって、選択ストリング `a IS 3` および `b IS 'red'` は無効です。 `IS` キーワードは、`IS NULL` および `IS NOT NULL` のケースをサポートするためにのみ存在します。

関連概念

34 ページの『選択動作』
IBM MQ の選択動作に関する概要

関連資料

選択ストリング

メッセージ・セレクターを使用するときの *UTF-8* および *Unicode* の考慮事項

選択ストリングの予約済みキーワードを形成する、単一引用符で囲まれていない文字は、Basic Latin Unicode (文字 U+0000 から U+0007F までの範囲) で入力される必要があります。 英数字の他のコード・ポイント表記の使用は 無効です。 例えば、数字 1 は Unicode で U+0031 と表す必要があり、全角数字の等価 U+FF11 またはアラビア語の等価 U+0661 の使用は無効です。

メッセージ・プロパティー名は、任意の有効な Unicode 文字のシーケンスを 使用して指定できます。 UTF-8 でエンコードされた選択ストリング内に含まれるメッセージ・プロパティー名 は、マルチバイト文字を含んでいる場合でも妥当性検査されます。 マルチバイト UTF-8 の妥当性検査は厳密であり、有効な UTF-8 シーケンス がメッセージ・プロパティー名に使用されるようにする必要があります。 メッセージ・プロパティー名において、Unicode 基本多言語面より上の文字 (U+FFFF より上)、つまり UTF-16 の場合にサロゲート・コード・ポイント (X'D800' から X'DFFF' まで) で表わされる文字、または UTF-8 の場合 4 バイトで表わされる文字はサポートされていません。

等しいことを確認するための比較時には、プロパティー名または値に対する余分な処理は 実行されません。 これは、例えば、事前組み立て/分解が行われないこと、および合字に特別な意味が与えられないことを意味します。 例えば、事前に組み立てられたウムラウト 文字 U+00FC は U+0075 + U+0308 と等価とは見なされず、 文字シーケンス ff は Unicode U+FB00 (LATIN SMALL LIGATURE FF) と等価とは見なされません。

単一引用符で囲まれたプロパティー・データは、バイトの任意のシーケンスによって表すことができ、妥当性検査されません。

メッセージの内容の選択

メッセージ・ペイロードの内容の選択 (内容のフィルター処理とも呼ばれる) に基づいてサブスクライブすることができますが、そのようなサブスクリプションにどのメッセージを送信するかについての決定を IBM MQ で直接行うことはできません。メッセージを処理するには、代わりに、IBM Integration Bus などの拡張メッセージ選択プロバイダーが必要です。

トピック・ストリングでアプリケーションがパブリッシュするときに、1 つ以上のサブスクライバーがメッセージの内容を選択する選択ストリングを持つ場合、IBM MQ は、拡張メッセージ選択プロバイダーがパブリケーションを構文解析し、内容のフィルターを使用して各サブスクライバーによって指定された選択基準とそのパブリケーションが一致するかどうかを IBM MQ に通知するように要求します。

パブリケーションがサブスクライバーの選択ストリングと一致すると、拡張メッセージ選択プロバイダーが判断した場合、 メッセージは引き続きサブスクライバーに送信されます。

パブリケーションが一致しないと拡張メッセージ選択プロバイダーが判断した場合、 メッセージはサブスクライバーに送信されません。 これによって、MQPUT または MQPUT1 呼び出しが理由コード MQRC_PUBLICATION_FAILURE で失敗する場合があります。 拡張メッセージ選択プロバイダーが パブリケーションを構文解析できなければ、理由コード MQRC_CONTENT_ERROR が戻され、 MQPUT または MQPUT1 呼び出しが失敗します。

拡張メッセージ選択プロバイダーが、使用不可であるか、 そのサブスクライバーでパブリケーションを受け取るべきかどうかを判断できなければ、 理由コード MQRC_SELECTION_NOT_AVAILABLE が戻され、 MQPUT または MQPUT1 呼び出しが失敗します。

内容のフィルターを使用してサブスクリプションを作成中に、 拡張メッセージ選択プロバイダーが使用不可である場合、MQSUB 呼び出しは理由コード MQRC_SELECTION_NOT_AVAILABLE で失敗します。 内容のフィルターを使用したサブスクリプションを再開中に、 拡張メッセージ選択プロバイダーが使用不可である場合、MQSUB 呼び出しが MQRC_SELECTION_NOT_AVAILABLE の警告を戻しますが、サブスクリプションは再開できます。

関連資料
選択ストリング

IBM MQ メッセージの非同期コンシューム

非同期コンシュームでは、メッセージ・キュー・インターフェース (MQI) 拡張のセット、 MQI 呼び出し MQCB および MQCTL が使用されます。これによって、 一連のキューからのメッセージをコンシュームする MQI アプリケーションの作成が可能になります。 メッセージ、またはメッセージを表すトークンのいずれかを渡すアプリケーションによって識別される 「コードの単位」を起動することにより、メッセージがアプリケーションに送られます。

最も単純なアプリケーション環境では、コードの単位は関数ポインターによって定義されますが、 他の環境では、プログラムまたはモジュールの名前によってコードの単位を定義できます。

メッセージの非同期コンシュームでは、以下の用語が使用されます。

メッセージ・コンシューマー
このプログラミング構造を使用すると、アプリケーションの要件に一致するメッセージが入手された場合にメッセージとともに起動されるプログラムまたは関数を定義することができます。

イベント・ハンドラー
このプログラミング構造を使用すると、非同期イベント (例えばキュー・マネージャーの静止) が発生した場合に起動するプログラムまたは関数を定義できます。

コールバック
メッセージ・コンシューマーまたはイベント・ハンドラー・ルーチンを指す一般的な用語。

非同期コンシュームを使用すると、新規アプリケーション (特に、複数の入力キューまたはサブスクリプションを処理するアプリケーション) の設計とインプリメンテーション が簡単になります。 しかし、複数の入力キューを 使用する場合で、優先順位の順序でメッセージを処理する場合、 優先順位の順序は各キュー内で個々に監視されます。1 つのキューからの低優先順位メッセージを 別のキューの高優先順位メッセージより先に受け取る可能性 があります。 複数のキューにまたがるメッセージ順序は 保証されません。
API 出口を使用する場合は、MQCB および MQCTL 呼び出しを組み込むように出口の変更が必要なことがある点にも注意してください。

以下の図は、この機能の使用例を示しています。

43 ページの図 5 は、2 つのキューからのメッセージを消費するマルチスレッド・アプリケーションを示しています。 この例では、すべてのメッセージが 1 つの関数に送られています。

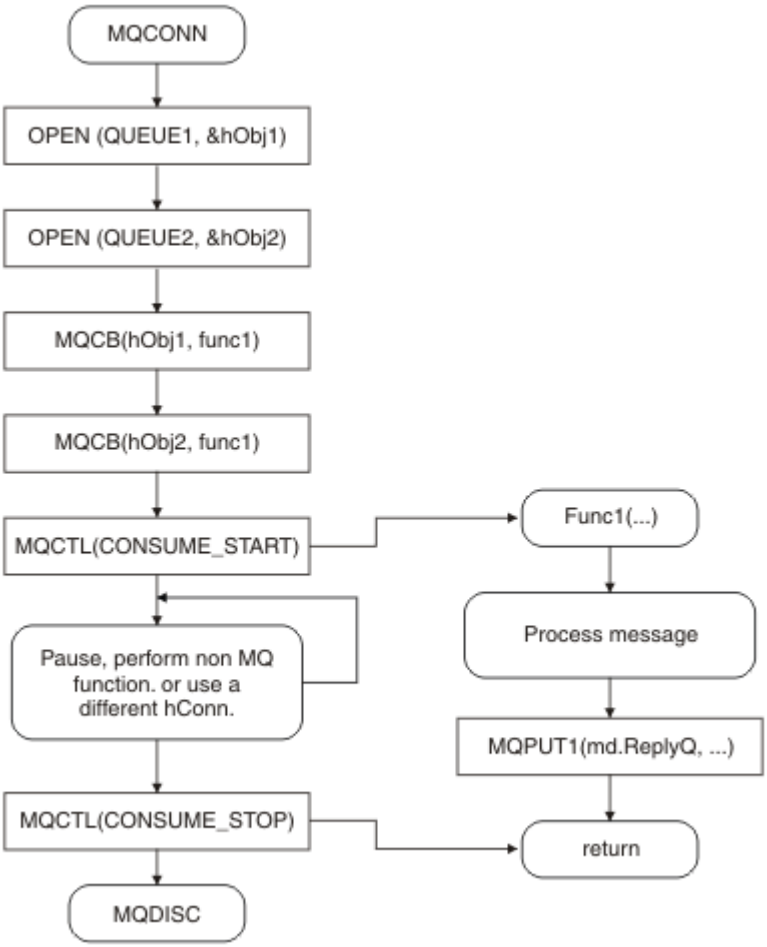


図 *5. 2* つのキューから消費する標準的なメッセージ・ドリブン・アプリケーション

z/OS z/OS では、主制御スレッドが、終了前に MQDISC 呼び出しを発行する必要があります。 これによって、すべてのコールバック・スレッドを 終了してシステム・リソースを解放することができます。

44 ページの図 6 のサンプル・フローは、2 つのキューからのメッセージを消費する単一スレッド・アプリケーションを示しています。 この例では、すべてのメッセージが 1 つの関数に送られています。

非同期の場合との違いは、すべてのコンシューマーが非アクティブ化するまで (つまり 1 つのコンシューマーが MQCTL STOP 要求を出すか、 キュー・マネージャーが静止するまで)、MQCTL の 発行者に制御が戻らないことです。

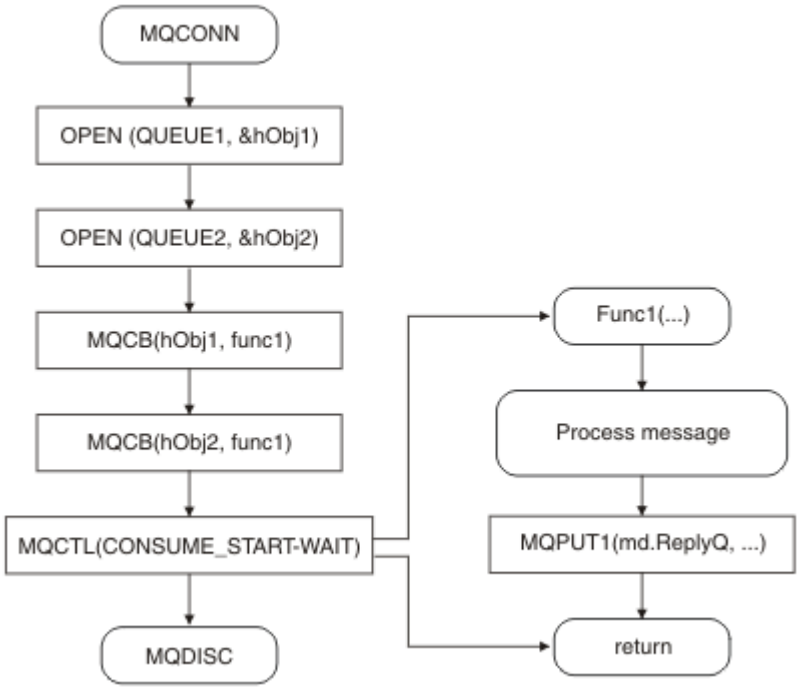


図 *6. 2* つのキューから消費する単一スレッドのメッセージ・ドリブン・アプリケーション

メッセージ・グループ

メッセージは、メッセージを順序付けするためにグループ化できます。

メッセージ・グループを使用すると、複数のメッセージを相互に関連したものとしてマークすることができます。また、グループに論理順序を適用することができます (774 ページの『論理的な順序付けと物理的な順序付け』 を参照)。 マルチプラットフォーム では、メッセージ・セグメンテーションによって大きなメッセージを小さなセグメントに分割できます。 グループ化またはセグメント化したメッセージは、トピックに書き込む際に使用することはできません。

グループ内の階層は、次のようになります。

グループ
これは、階層内の最高レベルで、*GroupId* で識別されます。 グループは、*GroupId* がすべて同じである 1 つ以上のメッセージで構成されます。 これらのメッセージは、キューの任意の位置に格納できます。

注 : メッセージ という用語は、ここでは、MQGMO_COMPLETE_MSG を指定しない 1 つの MQGET によって戻される、キューの 1 つの項目を示すために使用します。

次の 44 ページの図 7 は論理メッセージのグループを示しています。

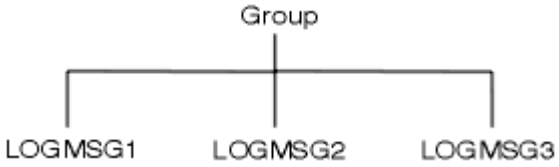


図 *7.* 論理メッセージのグループ

キューを開くときに MQOO_BIND_ON_GROUP を指定すると、このキューに送信されるメッセージのうち 1 つのグループに含まれるすべてのメッセージが、このキューの同一のインスタンスに送信されます。 BIND_ON_GROUP オプションの詳細については、メッセージの類縁性の処理を参照してください。

論理メッセージ

グループ内論理メッセージは、*GroupId* フィールドと *MsgSeqNumber* フィールドで識別されます。 *MsgSeqNumber* メッセージは 1 で始まり、これがグループ内の最初のメッセージに割り当てられますが、メッセージがグループ内に存在しない場合も、フィールドの値は 1 になります。

グループ内の論理メッセージを次の目的で使用します。

- 順序付けを保証する (メッセージが伝送される環境で順序付けが保証されていない場合)
- アプリケーションで類似のメッセージをグループ化できます (例えば、同一のサーバー・インスタンスで処理する必要のあるメッセージをすべてグループ化するなど)。

グループ内の各メッセージは、複数のセグメントに分割されない限り、1 つの物理メッセージから成ります。 各メッセージは論理的には別々のメッセージであり、グループ内の他のメッセージとの関係を保持する必要があるのは、MQMD 内の *GroupId* フィールドと *MsgSeqNumber* フィールドのみです。 MQMD 内の他のフィールドは独立しており、いくつかのフィールドはグループ内のすべてのメッセージについて同じで、他のいくつかのフィールドはそれぞれ異なります。 例えば、1 つのグループに属するメッセージであっても、フォーマット名、CCSID、エンコード方式が異なっていて構いません。

セグメント

セグメントは、書き込み用または読み取り用のアプリケーション、あるいはキュー・マネージャー (メッセージが処理されるときに介入するキュー・マネージャーなど) にとって大きすぎるメッセージを処理するために使用されます。 詳しくは、792 ページの『メッセージのセグメント化』 を参照してください。

個々のメッセージは、セグメント と呼ばれる、より小さいメッセージに分割されます。 メッセージのセグメントは、*GroupId* フィールド、*MsgSeqNumber* フィールド、および *Offset* フィールドで識別されます。 *Offset* フィールドは、ゼロで始まり、それがメッセージ内の最初のセグメントに割り当てられます。

各セグメントは、1 つの物理メッセージから成り、このメッセージがグループに属している場合もあります (45 ページの図 8 にグループ内のメッセージの例を示します)。 セグメントは、論理的には 1 つのメッセージの一部分であり、同じメッセージの個別のセグメント間では MQMD 内の *MsgId*、*Offset*、*MsgFlags* の各フィールドが異なるだけです。 セグメントが到着に失敗した場合は、理由コード MQRC_INCOMPLETE_GROUP または MQRC_INCOMPLETE_MSG が必要に応じて戻されます。

45 ページの図 8 に一部がセグメント化された論理メッセージのグループを示します。

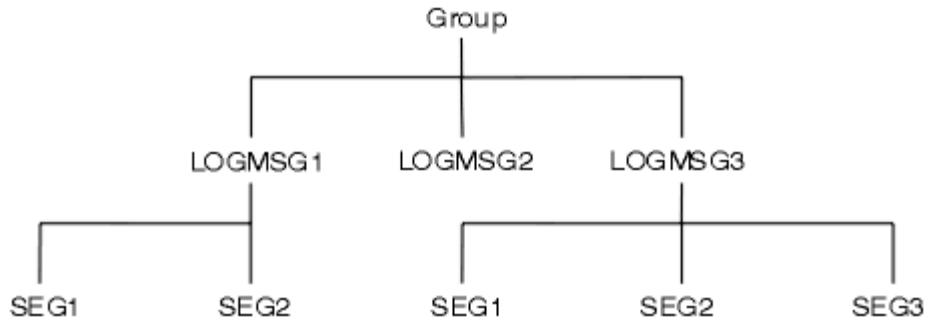


図 *8.* セグメント化されたメッセージ

z/OS セグメンテーションは、IBM MQ for z/OS ではサポートされません。

パブリッシュ/サブスクライブでは、セグメント化またはグループ化されたメッセージを使用できません。

関連概念

792 ページの『メッセージのセグメント化』
この情報を使用して、メッセージのセグメント化について学習します。 この機能は、IBM MQ for z/OS ではサポートされていません。また IBM MQ classes for JMS を使用するアプリケーションによってもサポートされていません。

関連資料

774 ページの『論理的な順序付けと物理的な順序付け』

キューにあるメッセージの順序には、(優先順位レベルごとの) 物理 順序と論理 順序があります。

MQMD - メッセージ記述子

メッセージの持続性

持続メッセージはログとキュー・データ・ファイルに書き込まれます。 キュー・マネージャーが障害のあとに再始動される場合は、 キュー・マネージャーがログ・データから必要に応じてこれらの持続メッセージを回復します。 キュー・マネージャーが停止すると、 その停止がオペレーター・コマンドの結果であれ、システムのある部分の障害であれ、非持続メッセージは破棄されます。

z/OS z/OS のカップリング・ファシリティー (CF) に格納されている非持続メッセージはこの点の例外となります。 これらは、CF が使用可能な限り持続します。

メッセージを作成するときにデフォルト値を使用してメッセージ記述子 (MQMD) を初期設定する場合、メッセージの持続性には MQOPEN コマンドで指定されるキューの **DefPersistence** 属性が使用されます。 または、MQMD 構造体の *Persistence* フィールドを使用してメッセージの持続値を設定し、持続または非持続としてメッセージを定義できます。

持続メッセージを使用する場合、ご使用のアプリケーションのパフォーマンスに影響があります。影響の範囲は、マシンの入出力サブシステムのパフォーマンスの特性と、各プラットフォームでの同期点オプションの使用法によって次のように異なります。

- 現在の作業単位外の持続メッセージは、書き込みまたは読み取り操作ごとにディスクに書き込まれます。857 ページの『作業単位のコミットとバックアウト』を参照。
- z/OS ALW IBM i 以外のすべてのプラットフォームでは、現在の作業単位内の持続メッセージは、その作業単位がコミットされ、その作業単位に多くのキュー操作を格納できる場合にのみログに記録されます。

非持続メッセージは、高速メッセージングに使用できます。 高速メッセージの詳細については、メッセージの安全性を参照してください。

注 : 作業単位内への持続メッセージ書き込みと作業単位外への持続メッセージ書き込みを組み合わせると、アプリケーションに深刻なパフォーマンス上の問題が発生する場合があります。 両方の操作に同じ宛先キューを使用する場合、特に発生しやすくなります。

送達できないメッセージ

キュー・マネージャーがメッセージをキューに入れられない場合、さまざまなオプションがあります。

以下のことが可能です。

- 再び、キューにメッセージを書き込むよう試みる。
- メッセージを送信側に戻すように要求する。
- メッセージを送達不能キューに書き込む。

詳しくは、1042 ページの『プロシージャー型プログラム・エラーの処理』を参照してください。

バックアウトされるメッセージ

作業単位の制御の下でキューからのメッセージを処理しているとき、作業単位が 1 つ以上のメッセージから成ることがあります。 バックアウトが起こると、キューから取り出されたメッセージはキューに復元され、別の作業単位で再び処理できます。 特定のメッセージの処理が問題を引き起こしている場合は、その作業単位が再度バックアウトされます。 これにより処理ループが起こる可能性があります。 キューに書き込まれたメッセージは、キューから削除されます。

アプリケーションは、そのようなループ中に出たメッセージを MQMD の *BackoutCount* フィールドをテストすることによって検知できます。 アプリケーションは状況を修正するか、あるいは警告メッセージをオペレーターに出すことができます。

Multi バックアウト・カウントは、必ず、キュー・マネージャーの再始動後も存続します。 **HardenGetBackout** 属性に対する変更は、すべて無視されます。

z/OS 共有キューの場合、バックアウト・カウントは、必ず、キュー・マネージャーの再始動後も存在します。 z/OS 上の他のすべての構成では、専用キューのバックアウト・カウントをキュー・マネージャーの再始動時に確実に保持するために、*HardenGetBackout* 属性を MQQA_BACKOUT_HARDENED に設定してください。このようにしないと、キュー・マネージャーを再始動する必要がある場合に、各メッセージの正確なバックアウト・カウントが維持されません。 この属性設定を行うと、余分な処理によるコストが発生します。

メッセージのコミットおよびバックアウトの詳細については、857 ページの『作業単位のコミットとバックアウト』を参照してください。

応答先キューおよびキュー・マネージャー

次のような場合、送信したメッセージに対する応答としてメッセージを受け取ることがあります。

- 要求メッセージに対する応答メッセージ
- 予期しないイベントまたは満了に関する報告メッセージ
- COA (到着確認) または COD (送達確認) のイベントに関する報告メッセージ
- PAN (肯定アクション通知) または NAN (否定アクション通知) のイベントに関する報告メッセージ

MQMD 構造体を使用して、応答および報告メッセージを送信したいキューの名前を *ReplyToQ* フィールドに指定してください。 *ReplyToQMgr* フィールドに応答先キューを所有するキュー・マネージャーの名前を指定してください。

ReplyToQMgr フィールドをブランクにしておくと、キュー・マネージャーはそのキューにメッセージ記述子内の次のフィールドの内容を設定します。

ReplyToQ
ReplyToQ がリモート・キューのローカル定義である場合、*ReplyToQ* フィールドにはそのリモート・キューの名前が設定されます。そうでない場合、このフィールドは変更されません。

ReplyToQMgr
ReplyToQ がリモート・キューのローカル定義である場合、*ReplyToQMgr* フィールドにはそのリモート・キューを所有するキュー・マネージャーの名前が設定されます。そうでない場合、*ReplyToQMgr* フィールドにはアプリケーションが接続したキュー・マネージャーの名前が設定されます。

注 : キュー・マネージャーにメッセージの送達を複数回試行させるようにし、送達できなかった場合にそのメッセージを破棄するよう要求することができます。 送達できなかったメッセージを破棄できなかった場合、リモート・キュー・マネージャーはそのメッセージを送達不能 (未配布メッセージ) キューに入れます (1046 ページの『送達不能 (未配布メッセージ) キューの使用』を参照)。

メッセージ・コンテキスト

メッセージ・コンテキスト 情報により、メッセージを受信するアプリケーションは、そのメッセージの発信元についての情報を得ることができます。

取り出しを行うアプリケーションは次のような処理を行うことがあります。

- 送信側アプリケーションの許可レベルが適正であるかをチェックする。
- 取り出しアプリケーションが実行しなければならない作業の対価を送信側アプリケーションに請求できるようにするため、課金機能を実行する。
- 処理したすべてのメッセージの監査記録を保持する。

メッセージをキューに書き込むために MQPUT または MQPUT1 呼び出しを使用するときは、キュー・マネージャーがメッセージ記述子に何らかのデフォルト・コンテキスト情報を追加するように指定できます。適切な許可レベルを持つアプリケーションは、余分のコンテキスト情報を追加できます。 コンテキスト情報の指定方法の詳細については、759 ページの『メッセージ・コンテキスト情報の制御』を参照してください。

ユーザー・コンテキストは、次のタイプのレポート・メッセージを生成するときにキュー・マネージャーによって使用されます。

- 送達時の確認
- Expiry

これらのレポート・メッセージが生成されるときには、ユーザー・コンテキストにレポート宛先に対する +put 権限と +passid 権限があるかどうかが検査されます。 ユーザー・コンテキストの権限が不足している場合は、レポート・メッセージは送達不能キューに置かれます (そのキューが定義されている場合)。 送達不能キューがない場合、そのレポート・メッセージは破棄されます。

すべてのコンテキスト情報は、メッセージ記述子のコンテキスト・フィールド内に保管されます。 情報のタイプは、識別コンテキスト情報、起点コンテキスト情報、およびユーザー・コンテキスト情報に分けられます。

identity コンテキスト

識別コンテキスト 情報により、そのメッセージをキューに最初に書き込んだアプリケーションのユーザーを次のように識別します。 適切に権限を与えられたアプリケーションは、次のフィールドを設定できます。

- キュー・マネージャーは、*UserIdentifier* フィールドにユーザーを識別する名前を入れる。その方法は、アプリケーションが稼働している環境によって異なる。
- キュー・マネージャーは、*AccountingToken* フィールドに、そのメッセージを書き込んだアプリケーションから判別したトークンまたは番号を入れる。
- アプリケーションは、ユーザーに関して追加したい特別の情報 (例えば、暗号化されたパスワード) を *ApplIdentityData* フィールドに入れることができる。

Windows システム・セキュリティー ID (SID) は、メッセージが IBM MQ for Windows の下に作成されるときに *AccountingToken* フィールドに保管されます。 SID の使用目的は、*UserIdentifier* フィールドの補足とユーザーの資格情報の確立です。

キュー・マネージャーが *UserIdentifier* および *AccountingToken* の各フィールドに情報をどのように書き込むかについては、UserIdentifier および AccountingToken にあるこれらのフィールドの説明を参照してください。

1 つのキュー・マネージャーから別のキュー・マネージャーへメッセージを渡すアプリケーションは、他のアプリケーションがメッセージの発信元の ID を知ることができるように、識別コンテキスト情報も渡さなければなりません。

origin コンテキスト

起点コンテキスト 情報は、メッセージが現在入れられているキューにそのメッセージを書き込んだアプリケーションを記述します。 メッセージ記述子には、起点コンテキスト情報のための以下のフィールドがあります。

- *PutApplType* は、メッセージを書き込んだアプリケーションのタイプ (例えば、CICS トランザクション) を定義します。
- *PutApplName* は、メッセージを書き込んだアプリケーションの名前 (例えば、ジョブ名またはトランザクション名) を定義します。
- *PutDate* は、メッセージがキューに書き込まれた日付を定義します。
- *PutTime* は、メッセージがキューに書き込まれた時刻を定義します。
- *ApplOriginData* は、メッセージの起点に関してアプリケーションが組み込むよう求める特別な情報を定義します。 例えば、適切な許可を持つアプリケーションがこのフィールドを設定し、識別データが正しいかどうかを示すことができる。

起点コンテキスト情報は通常キュー・マネージャーにより提供されます。 起点コンテキスト情報は通常キュー・マネージャーにより提供されます。*PutDate* フィールドおよび *PutTime* フィールドには、グリニッジ標準時 (GMT) が使用されます。 PutDate および PutTime にあるこれらのフィールドの説明を参照してください。

適格な許可のあるアプリケーションは、独自のコンテキストを提供できます。 これにより、1 人のユーザーが、発行したメッセージを処理する各システムごとに異なるユーザー ID を持つ場合でも会計情報を保存できます。

IBM MQ オブジェクト

この情報は、IBM MQ オブジェクトの詳細を提供します。これには、キュー・マネージャー、キュー共有グループ、キュー、管理トピック・オブジェクト、名前リスト、プロセス定義、認証情報オブジェクト、チャネル、ストレージ・クラス、リスナー、およびサービスがあります。

キュー・マネージャーでは、これらのオブジェクトの特性 (つまり、属性) を定義します。 これらの属性の値は、IBM MQ がこれらのオブジェクトを処理する方法に影響します。 ユーザーのアプリケーションからこれらのオブジェクトを制御するには、メッセージ・キュー・インターフェース (MQI) を使用します。 オブジェクトは、プログラムからアドレッシングされるときは、オブジェクト記述子 (MQOD) によって識別されます。

IBM MQ コマンドを使用してオブジェクトの定義、変更、または削除を行う場合、例えば、キュー・マネージャーが必要なレベルの権限でこれらの操作を実行しているかどうかを検査します。 同様に、アプリケーションが MQOPEN 呼び出しを使用してオブジェクトをオープンするとき、キュー・マネージャーは、そのオブジェクトへのアクセスを許可する前に、アプリケーションが必要なレベルの権限を持っているかどうかを検査します。 検査は、オープンされているオブジェクトの名前に対して行われます。

関連概念

759 ページの『メッセージ・コンテキスト情報の制御』
メッセージをキューに書き込むために MQPUT または MQPUT1 呼び出しを使用するときは、キュー・マネージャーがメッセージ記述子に何らかのデフォルト・コンテキスト情報を追加するように指定できます。適切な許可レベルを持つアプリケーションは、余分のコンテキスト情報を追加できます。 MQPMO 構造体のオプション・フィールドを使用して、コンテキスト情報を制御できます。

関連資料

750 ページの『メッセージ・コンテキストに関連する MQOPEN オプション』
メッセージをキューに書き込むときに、コンテキスト情報とメッセージの関連付けができるようにしたい場合は、キューをオープンするときにメッセージ・コンテキスト・オプションの 1 つを使用しなければなりません。

Windows Microsoft Transaction Server アプリケーションの準備と実行

MTS アプリケーションを、IBM MQ MQI client・アプリケーションとして実行するように準備するには、環境に応じて以下の指示に従ってください。

IBM MQ リソースにアクセスする Microsoft Transaction Server (MTS) アプリケーションの開発方法に関する一般情報については、 IBM MQ ヘルプ・センターの MTS に関するセクションを参照してください。

IBM MQ MQI client・アプリケーションとして動作するように MTS アプリケーションを準備するには、アプリケーションのコンポーネントごとに、次のいずれかを実行してください。

- コンポーネントが MQI に C 言語バインディングを使用する場合は、1022 ページの『Windows での C プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー mqic.lib ではなく、ライブラリー mqicxa.lib でリンクします。
- コンポーネントが IBM MQ C++ クラスを使用する場合は、548 ページの『Windows における C++ プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー imqc23vn.lib ではなく、ライブラリー imqx23vn.lib でリンクします。
- コンポーネントが MQI に Visual Basic 言語バインディングを使用する場合は、1025 ページの『Windows での Visual Basic プログラムの準備』 の指示に従います。ただし、Visual Basic プロジェクトを定義する際、「**条件付きコンパイル引数**」 フィールドに MqType=3 と入力します。

IBM MQ アプリケーションの設計上の考慮事項

プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

IBM MQ アプリケーションを設計する際には、以下の質問およびオプションについて検討してください。

アプリケーションのタイプ
アプリケーションの用途は何ですか。 以下のリンクを参考にして、開発可能なアプリケーションのさまざまなタイプについての詳細を確認してください。

- サーバー
- クライアント
- パブリッシュ/サブスクライブ
- Web サービス
- ユーザー出口、API 出口、およびインストール可能サービス

さらに、独自のアプリケーションを作成して、IBM MQ の管理を自動化できます。 詳細については、IBM MQ 管理インターフェース (MQAI) と管理タスクの自動化を参照してください。

プログラミング言語
IBM MQ では、さまざまなプログラミング言語でアプリケーションを作成できます。 詳細については、5 ページの『IBM MQ 用アプリケーションの開発』を参照してください。

複数のプラットフォームに対応するアプリケーション

使用するアプリケーションは、複数のプラットフォームで稼働するのでしょうか。 現在使用中のアプリケーションから、異なるプラットフォームに移動するための戦略がありますか。 これらの質問のどちらかに対する答えが「はい」の場合、プラットフォームの独立性に対応するようにプログラムをコーディングしているかどうかを確認してください。

例えば C を使用している場合は、ANSI 標準 C でコードを使用します。 プラットフォーム固有の関数がより迅速であったりより効率的であったりする場合でも、同等のプラットフォーム固有の関数ではなく、標準の C ライブラリー関数を使用してください。 例外となるのは、コーディングにおける効率を優先する場合と、#ifdef を使用して上記の両方の状況を想定したコーディングを行う場合です。 以下に例を示します。

```
#ifdef _AIX
    AIX specific code
#else
    generic code
#endif
```

キューのタイプ
必要に応じてキューを作成するのか、または既に設定されているキューを使用するのか。 キューを使用した後、そのキューを削除するのか、あるいは再び使用するのか。 アプリケーションの独立性のために別名キューを使用するのかなどを決める必要があります。 サポートされているキューのタイプを確認するには、「キュー」を参照してください。

z/OS 共用キュー、キュー共用グループ、およびキュー共用グループ・クラスターの使用 (IBM MQ for z/OS のみ)
キュー共用グループで共用キューを使用するときに可能となる、増強された可用性、スケーラビリティー、および作業負荷の平衡化を利用したい場合があります。 詳しくは、「共用キューおよびキュー共用グループ」を参照してください。

また、メッセージ・フローの平均およびピークを見積もって、キュー共用グループ・クラスター の使用による作業負荷の分散を検討する必要がある場合も考えられます。 詳しくは、「共用キューおよびキュー共用グループ」を参照してください。

キュー・マネージャー・クラスターの使用
クラスターを使用すると、システム管理が単純化され、さらに、可用性、 スケーラビリティー、およびワークロード・バランシングが向上する可能性が あります。

メッセージのタイプ
単純なメッセージ用にデータグラムを用いる場合もあれば、別の状況では要求メッセージ (応答を期待するメッセージ) を用いる場合もあります。 あるメッセージには異なった優先順位を割り当てたい場合もあります。 メッセージの設計について詳しくは、58 ページの『メッセージの設計手法』を参照してください。

パブリッシュ/サブスクライブ・メッセージングまたは Point-to-Point メッセージングの使用

パブリッシュ/サブスクライブ・メッセージングを使用する場合、送信側アプリケーションは、共用する情報を IBM MQ メッセージに入れて、IBM MQ パブリッシュ/サブスクライブが管理する標準宛先へ送信し、その情報の配布を IBM MQ に処理させます。 ターゲット・アプリケーションは、受け取る情報の送信元に ついて何も知る必要はなく、単に 1 つ以上のトピックに関する興味を登録し、 その情報があれば受け取ります。 パブリッシュ/サブスクライブ・メッセージングについて詳しくは、「パブリッシュ/サブスクライブ・メッセージング」を参照してください。

point-to-point メッセージングを使用する場合、送信側アプリケーションは、 受信側アプリケーションがそこから取り出すことが分かっている特定のキューにメッセージを送信 します。 受信側アプリケーションは、特定のキューからメッセージを 読み取り、それらのメッセージの内容に応じて処理を行います。 1 つのアプリケーションが送信側と受信側の 両方として機能することもよくあり、別のアプリケーションにクエリーを送信し、 応答を受け取ります。

IBM MQ プログラムの制御

あるプログラムを自動的に開始したり、特定のメッセージがキューに入るまでプログラムを待機させたい場合があります (IBM MQ トリガー操作 機能を使用します。この機能については、869 ページの『トリガーによる IBM MQ アプリケーションの開始』を参照)。 あるいは、キュー上のメッセージが十分な速さで処理されない場合に、アプリケーションの別のインスタンスを開始することもできます (インスツルメンテーション・イベント で説明されているように、 IBM MQ インスツルメンテーション・イベント 機能を使用します)。

IBM MQ クライアント上でのアプリケーションの稼働

MQI の全機能は、クライアント環境でサポートされています。プロシージャー型言語で作成された大部分の IBM MQ アプリケーションは、IBM MQ MQI client 上で稼働するように再リンクできます。 IBM MQ MQI client 上のアプリケーションは、MQI ライブラリーではなく、MQIC ライブラリーにリンクしてください。 z/OS z/OS 上の読み取り (信号) は、サポートされない。

注 : IBM MQ クライアント上で稼働しているアプリケーションは、複数のキュー・マネージャーに同時に接続したり、アスタリスク (*) 付きのキュー・マネージャー名を MQCONN 呼び出しまたは MQCONNX 呼び出しに使用したりすることができます。 クライアントのライブラリーではなくキュー・マネージャーのライブラリーにリンクしたい場合は、アプリケーションを変更してください。なぜなら、この機能は使用できないからです。

詳しくは、 925 ページの『IBM MQ MQI client 環境でのアプリケーションの実行』 を参照してください。

アプリケーションのパフォーマンス

設計上の決定事項がアプリケーションのパフォーマンスに影響を与える場合があります。 IBM MQ アプリケーションのパフォーマンスを向上させるための推奨事項については、60 ページの『アプリケーションの設計およびパフォーマンスに関する考慮事項』 IBM i および 63 ページの『IBM i アプリケーションの設計およびパフォーマンスに関する考慮事項』 を参照してください。

IBM MQ の高度な手法

より高機能なアプリケーションでは、応答の対応付けや IBM MQ コンテキスト情報の生成と送信など、IBM MQ の高度な手法を使用することをお勧めします。 詳しくは、62 ページの『高機能アプリケーションの設計手法』 を参照してください。

データの保護およびデータの整合性の維持

メッセージと共に渡されるコンテキスト情報を使用して、そのメッセージが許容ソースから送信されたものかどうかを調べることができます。 また、IBM MQ やご使用のオペレーティング・システムで提供される同期点処理機能を使用して、データが確実に他のリソースとの整合性を持ち続けるようにすることができます (詳細は、857 ページの『作業単位のコミットとバックアウト』 を参照してください)。 IBM MQ メッセージの 持続性 機能を使用して、重要なメッセージを確実に配信することができます。

IBM MQ アプリケーションのテスト

IBM MQ プログラムのアプリケーション開発環境は他のアプリケーション用のものと異なる点はないので、IBM MQ トレース機能と同様に、同じ開発ツールを使用できます。

z/OS IBM MQ for z/OS を使用して CICS アプリケーションをテストする場合、 CICS 実行診断機能 (CEDF) を使用できます。 CEDF は、すべての MQI 呼び出しおよびすべての CICS サービスへの呼び出しの入り口と出口にトラップを付けます。 また、CICS 環境で、各 MQI 呼び出しの前と後に診断情

報を提供するための API 交差出口プログラムを作成できます。 これを行う方法については、893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』を参照してください。

IBM i IBM i アプリケーションをテストするときは、標準のデバッガーを使用できます。 これを開始するには、STRDBG コマンドを使用してください。

例外およびエラーの処理

送達不能なメッセージの処理方法や、キュー・マネージャーによって報告されるエラー状態の解決方法について考慮する必要があります。 いくつかの報告については、MQPUT で報告オプションを設定する必要があります。

関連概念

IBM MQ の技術概要

65 ページの『z/OS アプリケーションの設計およびパフォーマンスに関する考慮事項』
アプリケーション設計は、パフォーマンスに影響する最も重要な要因の 1 つです。 このトピックを使用して、パフォーマンスに関連する設計要因の一部を理解します。

5 ページの『IBM MQ 用アプリケーションの開発』
メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

721 ページの『プロシージャー型キューイング・アプリケーションの作成』
この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

918 ページの『プロシージャー型クライアント・アプリケーションの作成』
IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

553 ページの『.NET アプリケーションの開発』
IBM MQ classes for .NET を使用すると、.NET アプリケーションは、IBM MQ MQI client として IBM MQ に接続することも、 IBM MQ サーバーに直接接続することもできます。

525 ページの『C++ アプリケーションの開発』
IBM MQ では、IBM MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

347 ページの『IBM MQ classes for Java の使用』
Java 環境で IBM MQ を使用します。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

サポートされるプログラミング言語でのアプリケーション名の指定

IBM MQ 9.2.0 より前では、 Java または JMS クライアント・アプリケーションで既にアプリケーション名を指定している場合があります。 IBM MQ 9.2.0 以降、この機能は IBM MQ for Multiplatforms 上の他のプログラミング言語にまで拡張されています。

アプリケーション名が使用される方法

アプリケーション名は以下から出力されます。

- runmqsc DISPLAY CONN APPLTAG
- runmqsc DISPLAY QSTATUS TYPE(HANDLE) APPLTAG
- runmqsc DISPLAY CHSTATUS RAPPLTAG
- MQMD.PutApplName
- アプリケーション・アクティビティー・トレース

アプリケーション名は、アプリケーションのアクティビティー・トレースを構成する際にも使用されます。Java 以外のアプリケーションのデフォルト・アプリケーション名は、 Windows および IBM i の場合を除き、実行可能ファイルの切り捨てられた名前です。

Windows Windows では、デフォルト名は、完全修飾の実行可能ファイル名の左側を切り捨てて 28 文字にした名前になります。

IBM i IBM i の場合、デフォルト名はジョブ名です。

Java アプリケーションの場合、デフォルト名は、クラス名の前にパッケージ名を付け、その左側を切り捨てて 28 文字にした名前になります。

詳しくは、PutApplName を参照してください。

IBM MQ 9.2.0 以降では、管理的手法か各種のプログラミング・メソッドを使用することにより、 IBM MQ for Multiplatforms 上のアプリケーションでそのアプリケーション名を設定することができます。 これにより、アプリケーションの ACTIVITY トレースを構成する際、または各種 **runmqsc** コマンドで出力する際に、アプリケーションで、プラットフォームに依存しない、より有意性のある名前を渡すことができます。

IBM MQ 9.2.0 以降、ユニフォーム・クラスター全体でアプリケーションをリバランスできます。 その実行のために、有意性のあるアプリケーション名が使用されます。

サポートされる文字

アプリケーション名の指定方法について詳しくは、54 ページの『アプリケーション名に推奨される文字』を参照してください。

プログラム言語

C およびその他のプログラミング言語で IBM MQ ライブラリーに解決されるアプリケーションがアプリケーション名を提供する方法について詳しくは、 56 ページの『プログラミング言語の接続』 を参照してください。

管理対象 .NET アプリケーション

管理対象 .NET アプリケーションがアプリケーション名を提供する方法については、 56 ページの『管理対象 .NET アプリケーション』 を参照してください。

XMS アプリケーション

XMS アプリケーションでアプリケーション名を渡す方法については、57 ページの『XMS アプリケーション』を参照してください。

Java および JMS のバインディング・アプリケーション

ALW

Java および JMS アプリケーションがアプリケーション名を提供する方法については、 58 ページの『Java および JMS のバインディング・アプリケーション』 を参照してください。

関連概念
アプリケーション・アクティビティー・トレース
均等クラスターについて
関連資料
MQCNO
IBM i 上での MQCNO

サポートされるプログラミング言語でのアプリケーション名の使用方法

ここでは、IBM MQ でサポートされるさまざまな言語で、どのようにアプリケーション名が選択されるかについて説明します。

アプリケーション名に推奨される文字

アプリケーション名は、キュー・マネージャー・フィールドの **CodedCharSetId** 属性で指定された文字セットでなければなりません。 この属性について詳しくは、 キュー・マネージャーの属性を参照してください。

しかし、アプリケーションが IBM MQ MQI client として実行している場合、アプリケーション名はクライアントの文字セット内およびエンコード内になければなりません。

キュー・マネージャー間でアプリケーション名をスムーズに遷移させ、リソース・モニター・トピックを介してアプリケーション・リソースをモニターできるようにするには、アプリケーション名に 1 バイトの印刷可能文字のみを含める必要があります。

注 :

- また、アプリケーション名にはスラッシュ文字とアンパーサンド文字を使用しないでください。
- V 9.3.0 アプリケーション名には、アンパーサンド文字の使用を避ける必要があります。 アンパーサンドを含むアプリケーション名に関するシステム・トピック STATAPP メトリックは作成されません。

したがって、名前に使用できるものは、以下のみに限定されます

- 英数字: A-Z、 a-z、および 0-9

  注 : EBCDIC カタカナを使用するシステムでは、アプリケーション名に小文字 a-z を使用してはいけません。
- スペース文字
- EBCDIC で不変の印刷可能文字: + < = > % * ' ( ) , _ - . : ; ?
- V 9.3.0 / 文字。 名前にスラッシュが含まれているアプリケーションのアクティビティー・トレースまたは STATAPP システム・トピック・メトリックをサブスクライブする場合は、スラッシュ文字をアンパーサンド文字に置き換える必要があります。 例えば、「DEPT1/APPS/STOCKQUOTE」というアプリケーションの STATAPP メトリックを受け取るには、トピック・ストリング「$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE」をサブスクライブする必要があります。 amqsact および amqsrua サンプル・アプリケーションでは、サブスクリプションの作成時にスラッシュ文字が自動的にアンパーサンドに変換されます。

文字の設定方法

以下の表に、IBM MQ でサポートされるさまざまな言語で、どのようにアプリケーション名が選択されるかについてまとめます。 名前の選択方法を優先度が高いほうから順に記載しています。

	C バインディングおよびクライアント	Java バインディングおよびクライアント	JMS バインディングおよびクライアント	管理対象 .NET クライアント	非管理対象 .NET バインディングおよびクライアント	管理対象 XMS クライアント	非管理対象 .XMS バインディングおよびクライアント
接続プロパティーのオーバーライド		Java 接続プロパティーのオーバーライド		.NET 接続プロパティーのオーバーライド	.NET 接続プロパティーのオーバーライド		
オーバーライドされるプロパティー		Java のオーバーライドされるプロパティー		.NET のオーバーライドされるプロパティー	.NET のオーバーライドされるプロパティー		
MQEnvironment		Java MQ 環境		.NET MQEnvironment	.NET MQEnvironment		
接続ファクトリー・プロパティー			接続ファクトリー・プロパティー			接続ファクトリー・プロパティー	接続ファクトリー・プロパティー
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	接続オプション						
環境変数	環境変数				環境変数		環境変数
mqclient.ini (クライアント接続の場合にのみ該当)	クライアント接続				クライアント接続		クライアント接続
Java クラス名		Java クラス名	Java クラス名				
デフォルト名	デフォルト名			.NET のデフォルト名	.NET のデフォルト名	.NET のデフォルト名	.NET のデフォルト名

注: 「C バインディングおよびクライアント」列は、以下のプログラミング言語にも適用されます。

- COBOL
- アセンブラー

- Visual Basic
- RPG

プログラミング言語の接続

C およびその他のプログラミング言語の IBM MQ ライブラリーとして解決されるアプリケーションは、以下の方法でアプリケーション名を渡すことができます。

接続方法を優先度の高いほうから順に記載しています。

Multi **接続オプション**

- ALW MQCNO

  注 : z/OS IBM MQ for z/OS キュー・マネージャーに接続する場合、アプリケーション名を設定するには、クライアント・モード接続を使用するか、 IBM MQ classes for JMS または IBM MQ classes for Java アプリケーションを使用する必要があります。

- IBM i IBM i 上での MQCNO

ALW **環境変数**

アプリケーション名をまだ選択していない場合は、 **MQAPPLNAME** 環境変数を使用して、キュー・マネージャーへの接続を識別できます。 以下に例を示します。

```
export MQAPPLNAME=ExampleApplName
```

最初の 28 文字のみが使用されることに注意してください。これらの文字をすべてブランクまたはすべて NULL にしてはいけません。

注 : この属性は、サポートされるプログラミング言語、非管理対象 .NET、および非管理対象 XMS 接続にのみ適用されます。

ALW **クライアント構成ファイル**

アプリケーション名をまだ選択しておらず、接続がクライアント接続である場合は、クライアント構成ファイル (例えば、 `mqclient.ini`) に以下の情報を指定して、キュー・マネージャーへの接続を識別することができます。

```
Connection:
    ApplName=ExampleApplName
```

注 :

1. 最初の 28 文字のみが使用されます。これらの文字をすべてブランクまたはすべて NULL にしてはいけません。
2. この属性は、サポートされるプログラミング言語、非管理対象 .NET、および非管理対象 XMS 接続のクライアント接続にのみ適用されます。

詳しくは、 IBM MQ MQI client 構成ファイル `mqclient.ini` を参照してください。

デフォルト名

まだアプリケーション名を選択していなかった場合は、デフォルト名がそのまま使用されます。このデフォルト名にはオペレーティング・システムに表示されるものと同じパス名 (可能な長さまで)および実行可能ファイル名が入っています。 詳しくは、PutApplName を参照してください。

管理対象 .NET アプリケーション

管理対象 .NET アプリケーションは次の方法でアプリケーション名を渡すことができます。

接続方法を優先度の高いほうから順に記載しています。

接続プロパティーのオーバーライド

次の方法で、接続詳細のオーバーライド・ファイルをアプリケーションに渡すことができます。

```
<appSettings>
     <add key="overrideConnectionDetails" value="true" />
     <add key="overrideConnectionDetailsFile" value="<location>" />
</appSettings>
```

overrideConnectionDetailsFile で指定したファイルには、接頭語 mqj が付いたプロパティーのリストが含まれています。 アプリケーションで **mqj.APPNAME** プロパティーを定義し、キュー・マネージャーへの接続を識別するための名前を **mqj.APPNAME** プロパティーの値に指定する必要があります。

名前の最初の 28 文字のみが使用されます。 以下に例を示します。

```
mqj.APPNAME=ExampleApplName
```

オーバーライドされるプロパティー

定数 **MQC.APPNAME_PROPERTY** に値 *APPNAME* が定義されました。 これで、名前の最初の 28 文字のみを使用して、このプロパティーを **MQQueueManager** コンストラクターに渡せるようになりました。 以下に例を示します。

```
Hashtable properties = new Hashtable();
properties.Add( MQC.APPNAME_PROPERTY, "ExampleApplName" );
MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
```

詳細については、652 ページの『.NET における管理操作および非管理操作』を参照してください。

MQEnvironment

AppName プロパティーが **MQEnvironment** クラスに追加され、最初の 28 文字のみが使用されます。 以下に例を示します。

```
MQEnvironment.AppName = "ExampleApplName";
```

デフォルト名

上記のいずれかの方法でアプリケーション名を指定していない場合、アプリケーション名は自動的に実行可能ファイル名 (および適合するパスの量) として設定されます。

XMS アプリケーション

接続方法を優先度の高いほうから順に記載しています。

接続ファクトリー・プロパティー

XMS アプリケーションは、**XMSC.WMQ_APPLICATIONNAME** プロパティー (「*XMSC_WMQ_APPNAME*」) を使用して、接続ファクトリーでアプリケーション名を提供できます。 JMS と同様の方法で。 最大 28 文字の名前を指定できます。

詳しくは、661 ページの『XMS .NET 管理対象オブジェクトの作成』および 668 ページの『XMS メッセージのプロパティー』を参照してください。

JMSAdmin

管理ツールでは、このプロパティーは「**APPLICATIONNAME**」または「**APPNAME**」と呼ばれます。

Java および JMS のバインディング・アプリケーション

接続方法を優先度の高いほうから順に記載しています。

ALW Java および JMS のクライアント・アプリケーションでは既にアプリケーション名を指定できますが、IBM MQ for Multiplatforms では MQCNO の **ApplName** フィールドを使用することでこれがバインディング・アプリケーションにまで拡張されました。

接続プロパティーのオーバーライド

オーバーライドできる接続プロパティーのリストに、 **Application name** プロパティーが追加されました。 詳しくは、 IBM MQ 接続プロパティー・オーバーライドの使用を参照してください。

重要 : 接続プロパティーと接続プロパティー・オーバーライド・ファイルの使用方法は、IBM MQ classes for Java と .NET の両方で同じです。

オーバーライドされるプロパティー

定数 **MQC.APPNAME_PROPERTY** に値 *APPNAME* が定義されました。 これで、名前の最初の 28 文字のみを使用して、このプロパティーを **MQQueueManager** コンストラクターに渡せるようになりました。 詳しくは、 IBM MQ classes for Java での接続プロパティーのオーバーライドの使用を参照してください。

MQEnvironment

AppName プロパティーが **MQEnvironment** クラスに追加され、最初の 28 文字のみが使用されます。

詳細については、375 ページの『IBM MQ classes for Java 用の IBM MQ 環境のセットアップ』を参照してください。

Java クラス名

前述の方法でアプリケーション名を指定しなかった場合、アプリケーション名はメイン・クラス名から取得されます。

詳細については、375 ページの『IBM MQ classes for Java 用の IBM MQ 環境のセットアップ』を参照してください。

重要 : IBM i IBM i ではメイン・クラス名を照会できないので、代わりに IBM MQ client for Java が使用されます。

関連概念

375 ページの『IBM MQ classes for Java 用の IBM MQ 環境のセットアップ』
アプリケーションがクライアント・モードでキュー・マネージャーに接続するには、チャネル名、ホスト名、およびポート番号を指定する必要があります。

関連資料

MQCNO
IBM i 上での MQCNO

メッセージの設計手法

セレクターやメッセージ・プロパティーに関する考慮事項など、メッセージの設計に役立つ考慮事項。

設計段階での考慮事項

メッセージの作成は、MQI 呼び出しを使用してこのメッセージをキューに書き込むときに行います。 呼び出しへの入力として、いくつかの制御情報をメッセージ記述子 (MQMD) に与え、さらに他のプログラムに送信したいデータを提供します。 ただし設計段階で、メッセージの作成方法に影響する、以下の事柄について検討する必要があります。

使用するメッセージのタイプ

メッセージを送信するだけで、その後のアクションが不要な単純なアプリケーションを設計する予定ですか。 それとも、質問に対する応答を求めますか。 質問を出している場合は、応答を受信したいキューの名前をメッセージ記述子に入れることができます。

要求メッセージと応答メッセージを同期させたいですか。 この場合は、要求に応答するためのタイムアウト期間を設定し、その期間内に応答を受信しなかったときには、エラーとして処理されます。

あるいは、非同期で作業するようにしますか。非同期にすると、プロセスは特定のイベントのオカレンス (共通のタイミング信号機能など) に依存しなくて済みます。

もう 1 つの考慮事項は、すべてのメッセージを作業単位の中に入れるかどうかです。

メッセージごとに異なる優先順位を割り当てる

各メッセージに優先順位の値を割り当て、キューがそのメッセージを優先順位の順に保持するようにキューを定義できます。 このようにすると、別のプログラムがキューからメッセージを取り出すときは、常に優先順位が最も高いメッセージを読み取ることになります。 キューがメッセージを優先順位の順に保持しない場合は、キューからメッセージを取り出すプログラムは、キューに入れられた順序でメッセージを取り出します。

また、プログラムは、メッセージがキューに書き込まれたときに キュー・マネージャーが割り当てた ID を使用して、そのメッセージを選択できます。 代わりに、各メッセージに対して独自の ID を割り当てることも可能です。

キュー・マネージャーの再始動がメッセージに与える影響

キュー・マネージャーを再始動すると、そのキュー・マネージャーはすべての持続メッセージを保持し、必要に応じて IBM MQ ログ・ファイルからそれらのメッセージを回復します。 非持続メッセージや一時動的キューは、保存されません。 廃棄したくないメッセージは、作成時に持続として定義する必要があります。 IBM MQ for Windows または IBM MQ (AIX and Linux システム用) のアプリケーションを作成するとき、アプリケーションがログ・ファイルの限界まで実行されるような設計となる危険性を減らすため、ログ・ファイルの割り振りに関してシステムがどのようにセットアップされているかを確認してください。

z/OS 共用キュー (IBM MQ for z/OS でのみ使用可能) 上のメッセージは Coupling Facility (CF) に保持されるため、非持続メッセージは、キュー・マネージャーが再始動されても、CF が使用可能である限り保存されます。 CF に障害が起こると、非持続メッセージは失われます。

メッセージの受信者に自分の情報を提供する

通常は、キュー・マネージャーがユーザー ID を設定しますが、適切な許可が与えられたアプリケーションがこのフィールドを設定することもあります。この場合は、課金またはセキュリティーのために受信側のプログラムが使用できる独自のユーザー ID やその他の情報をそのフィールドに入れることができます。

受け取るキューの量

Multi 1 つのメッセージを複数のキューに入れる必要がある場合は、トピックまたは配布リストにパブリッシュします。

z/OS 1 つのメッセージを複数のキューに入れる必要がある場合は、トピックにパブリッシュします。

セレクターとメッセージのプロパティー

メッセージには、そのメッセージのメインのペイロードに加えて、メッセージに関するメタデータを含めることができます。 このようなメッセージ・プロパティーは、追加のデータを提供するために役立つことがあります。

この追加データには、認識しておくべき重要な 2 つの側面があります。

- これらのプロパティーは、Advanced Message Security (AMS) 保護の対象ではありません。 AMS を使用してデータを保護するには、そのデータをメッセージ・プロパティーではなくペイロードに入れてください。
- プロパティーを使用して、メッセージの選択を実行できます。

セレクターを使用すると、先入れ先出しという標準のメッセージ規則が守られないことに注意する必要があります。 キュー・マネージャーはこのワークロード用に最適化されているので、複雑なセレクターを提供することは、パフォーマンス上の理由から推奨されません。 キュー・マネージャーはメッセージ・プロパティーの索引を保管しないので、メッセージの検索は必ずリニア・サーチになります。 キューが深くなるほど、セレクターは複雑になり、セレクターがメッセージと一致する可能性が低くなるので、パフォーマンスに悪影響を与えます。

複雑な選択が必要な場合は、IBM Integration Bus などのアプリケーションや処理エンジンを使用してメッセージをフィルタリングし、さまざまな宛先に振り分けることをお勧めします。 また、トピック階層を使用することが役立つ場合もあります。

注 : IBM MQ classes for Java はセレクターの使用をサポートしていません。 セレクターを使用する場合には、JMS API を介して行う必要があります。

アプリケーションの設計およびパフォーマンスに関する考慮事項

プログラム設計の悪さは、いろいろな面でパフォーマンスに影響します。 どのような影響があるのかを見つけるのは難しい場合があります。 プログラム自体は正しく実行されているように見えても、他のタスクのパフォーマンスに影響を与えていることがあるためです。 このトピックでは、IBM MQ 呼び出しを行うプログラムに特有のいくつかの問題を取り上げます。

効率的なアプリケーションを設計するための方法を以下に示します。

- 処理がユーザーの考慮時間と並行して進むようにアプリケーションを設計する。
  - パネルを表示させ、アプリケーションが初期化されている間でもユーザーが入力を開始できるようにする。
  - 異なるサーバーから並行して必要なデータを得る。
- 接続やキューを再利用する場合には、オープンやクローズ、接続や切断を何回も行うのではなく、接続とキューをオープンのままにしておく。
- ただし、メッセージを 1 つだけ書き込むサーバー・アプリケーションの場合には、MQPUT1 を使用する必要があります。
- キュー・マネージャーは、サイズが 4 KB から 100 KB までのメッセージ向けに 最適化されています。 あまりに大きいメッセージは非効率的で、1 MB ずつの 100 個のメッセージを送信するほうが、100 MB のメッセージを 1 つ送信 するよりもおそらく効率的です。 小さすぎるメッセージも 効率的ではありません。 キュー・マネージャーは、1 バイトのメッセージを処理するのにも、4 KB のメッセージの 場合と同じ作業量を必要とします。
- メッセージを 1 つの作業単位内で終わらせる。これにより、メッセージのコミットやバックアウトを同時に行うことができる。
- リカバリー可能にする必要のないメッセージに非持続性オプションを使用する。
- 多数のターゲット・キューに同じメッセージを送信する必要がある 場合、配布リストの使用を検討する。

メッセージ長の影響

メッセージ内のデータ量が、そのメッセージを処理するアプリケーションのパフォーマンスに影響することがあります。 アプリケーションのパフォーマンスを最大限に引き出すには、不可欠のデータだけをメッセージに入れて送信してください。 例えば、銀行預金口座の借方への記入要求では、クライアントからサーバー・アプリケーションに渡す必要のある情報は、口座番号と借方の金額だけです。

メッセージ持続の影響

通常、持続メッセージはログに記録されます。 メッセージをログに記録すると、アプリケーションのパフォーマンスが低下します。 したがって、重要なデータの場合のみ持続メッセージを使用してください。 メッセージ中のデータが、キュー・マネージャーが停止もしくは誤動作したときに廃棄してもかまわないものであれば、 非持続メッセージを使用してください。

z/OS 持続メッセージの MQPUT 操作と MQGET 操作は、リカバリー・ログ・スペースが不足していて操作を記録できない場合にはブロックされます。 こうした状態は、キュー・マネージャーのジョブ・ロ

グの CSQJ110E および CSQJ111A メッセージからわかります。 こうした状態の対処と防止のために、モニター・プロセスを適用してください。

特定メッセージの検索

MQGET 呼び出しでは、通常、キューから最初のメッセージを検索します。 メッセージ記述子で、メッセージ ID (*MsgId*) と相関 ID (*CorrelId*) を用いて特定のメッセージを指定すると、 キュー・マネージャーは、指定されたメッセージが見つかるまでキュー内を検索しなければなりません。 このような方法で MQGET 呼び出しを使用すると、アプリケーションのパフォーマンスに影響します。

可変長メッセージを含んでいるキュー

アプリケーションが固定長のメッセージを使用できない場合は、一般的なメッセージ・サイズに合うように、バッファーのサイズを動的に調整してください。 アプリケーションが MQGET 呼び出しを発行し、バッファーが小さすぎるためにこれが失敗する場合、メッセージ・データのサイズが戻されます。 これに合わせてバッファーをサイズ変更し、MQGET 呼び出しを再発行するコードをアプリケーションに追加してください。

注 : MaxMsgLength 属性を明示的に設定しない場合、デフォルトの 4 MB に設定されます。アプリケーションのバッファー・サイズがこの値によって制御された場合、非常に効率が悪くなることがあります。

同期点の頻度

1 つの同期点内で多数の MQPUT 呼び出しまたは MQGET 呼び出しをコミットなしで発行するプログラムは、パフォーマンス上の問題を引き起こす可能性があります。 影響を受けるキューは、現在アクセス不能なメッセージで満杯になり、他のタスクはそれらのメッセージを取得するために待機することがあります。これは、ストレージに影響があるうえに、メッセージを取得しようとする処理以外の処理を実行できなくなるというスレッドへの影響もあります。

MQPUT1 呼び出しの使用

MQPUT1 呼び出しの使用は、キューに書き込むメッセージが 1 つしかないときに限ってください。 複数のメッセージを書き込みたい場合は、まず MQOPEN を呼び出し、 続いて一連の MQPUT を呼び出して、最後に 1 回の MQCLOSE 呼び出しを行います。

使用するスレッドの数

Windows IBM MQ for Windows では、アプリケーションが多数のスレッドを必要とする場合があります。 各キュー・マネージャー・プロセスには、最大許容数のアプリケーション・スレッドが割り振られます。

アプリケーションが使用するスレッドが多すぎる可能性があります。 アプリケーションがこの可能性を考慮に入れているかどうか、またアプリケーションがこうしたタイプの出来事の発生を防ぐかまたは報告する処置をとることを考慮してください。

同期点での持続メッセージの書き込み

持続メッセージの書き込みと取得は、同期点で行う必要があります。 同期点の外で持続メッセージを取得する場合、取得に失敗したときに、メッセージがキューから取得されたのかどうかも、取得された後に失われたのかどうかもアプリケーションで認識することができないからです。 同期点の中で持続メッセージを取得する場合は、何かに失敗するとトランザクションがロールバックされます。持続メッセージはキューに残っているので失われることはありません。

同様に、持続メッセージを書き込むときにも、同期点の中で書き込んでください。 持続メッセージの書き込みと取得を同期点の中で行うもう 1 つの理由は、 IBM MQ の持続メッセージのコードが同期点用に高度に最適化されているためです。 そのために、同期点の中で行われる持続メッセージの書き込みや取得は、同期点の外で行われる持続メッセージの書き込みや取得よりも速くなります。

アプリケーションが同期点の外で持続メッセージを書き込む場合、キュー・マネージャーがアプリケーションの代わりに暗黙の同期点を作成できるかどうかを検査します。 作成できる場合、キュー・マネージャーはその同期点内に書き込みを行い、自動的にコミットします。 詳細については、865 ページの『Multiplatforms での暗黙の同期点』を参照してください。

一方、IBM MQ の非持続メッセージは同期点の外用に最適化されているので、非持続メッセージの書き込みと取得は同期点の外で行う方が速くなります。 持続メッセージはディスクに永続化されるので、持続メッセージの書き込みと取得はディスクの速度で実行されます。 一方、非持続メッセージの書き込みと取得は、同期点を使用する場合であってもディスクへの書き込みを伴わないので、CPU の速度で実行されます。

アプリケーションがメッセージを取得するときに、それが持続メッセージかどうかを事前に判別できない場合には、GMO オプションの MQGMO_SYNCPOINT_IF_PERSISTENT を使用できます。

高機能アプリケーションの設計手法

高機能のアプリケーションを設計するときには、メッセージの待機、応答の対応付け、コンテキスト情報の設定と使用、 アプリケーションの自動開始、レポートの生成、クラスター化を使用する場合のメッセージ親和性の除去など、検討したほうが良い手法がいくつかあります。

単純な IBM MQ アプリケーションの場合、使用しているアプリケーションにどの IBM MQ オブジェクトを使用するか、また、どのタイプのメッセージを使用するかについて決める必要があります。 より高度なアプリケーションでは、以下の節で紹介する技法のいくつかを使用する場合があります。

メッセージの待機

キューを扱うプログラムは、次のような方法でメッセージを待機できます。

- メッセージが到着するか、または指定の時間間隔が経過するまで待機する (797 ページの『メッセージの待機』を参照)。
- z/OS IBM MQ for z/OS でのみ、メッセージが到着したときにプログラムに知らせる信号を設定する。 詳しくは、798 ページの『信号機能』を参照してください。
- メッセージの到着時に実行されるコールバック出口を確立します。 詳細は、42 ページの『IBM MQ メッセージの非同期コンシューム』を参照してください。
- キューに対して、メッセージが到着しているかどうかを検査するための呼び出しを周期的に出す (ポーリング)。 これはパフォーマンスに影響する可能性があるため、通常は推奨されません。

応答の対応付け

IBM MQ アプリケーションでは、何らかの作業の実行を要求するメッセージをプログラムが受信した場合、プログラムは通常、1 つ以上の応答メッセージを要求側に送信します。

要求側がこれらの応答と元の要求との対応付けをしやすいように、アプリケーションは各メッセージの記述子内に相関 *ID* フィールドを設定できます。 それから、プログラムは要求メッセージのメッセージ ID を、それらの応答メッセージの相関 ID フィールド内にコピーします。

コンテキスト情報の設定と使用

コンテキスト情報 は、メッセージを、生成したユーザーに関連付けるためと、そのメッセージを生成したアプリケーションを識別するために用いられます。 このような情報は、セキュリティー、アカウンティング、監査、および問題判別のために役立ちます。

メッセージを生成するときに、キュー・マネージャーがデフォルトのコンテキスト情報をメッセージに関連付けるように要求するオプションを指定できます。

コンテキスト情報の設定と使用の詳細については、47 ページの『メッセージ・コンテキスト』を参照してください。

IBM MQ プログラムの自動的な開始

IBM MQ のトリガー操作 を使用すると、メッセージがキューに入ったときにプログラムが自動的に開始するようにできます。

キューに対して以下のいずれかのトリガー条件を設定し、その条件が満たされたときにプログラムがそのキューの処理を開始するようにできます。

- メッセージがキューに到着するたび。
- 最初のメッセージがキューに到着したとき。
- キュー上のメッセージの数が事前定義の数に達したとき。

トリガー操作の詳細については、869 ページの『トリガーによる IBM MQ アプリケーションの開始』を参照してください。 トリガー操作は、プログラムを自動的に開始する方法の一つにすぎません。 例えば、IBM MQ 以外の機能を使ってタイマーで自動的にプログラムを開始することもできます。

Multi マルチプラットフォーム では、IBM MQ はサービス・オブジェクトを使用して、キュー・マネージャーの開始時に IBM MQ プログラムを開始するように定義できます。サービス・オブジェクトを参照してください。

IBM MQ の報告の生成

アプリケーション内では、次の報告を要求できます。

- 例外報告
- 満了報告
- 到着確認 (COA) 報告
- 送達確認 (COD) 報告
- 肯定アクション通知 (PAN) 報告
- 否定アクション通知 (NAN) 報告

これらについては、20 ページの『レポート・メッセージ』を参照してください。

クラスターおよびメッセージ類似性

1 つのキューに対して複数の定義がある場合にクラスターを使用するには、その前に、関連メッセージの交換を必要としているアプリケーションがあるかを調べてください。

1 つのクラスター内では、該当するキューのインスタンスをホスト管理しているどのキュー・マネージャーにもメッセージが転送される可能性があります。 そのため、メッセージ類似性を含むアプリケーションの論理は無効になる場合があります。

例えば、2 つのアプリケーションがあり、両方ともその間を流れる質問形式と応答形式の一連のメッセージを信頼しているものとします。 重要な点として、質問はすべて一方のキュー・マネージャーに送られ、応答はすべてもう一方のキュー・マネージャーに送り返されます。 この場合、該当するキューのインスタンスを管理しようとしているキュー・マネージャーに対して、ワークロード管理ルーチンからメッセージは送信されないので注意してください。

可能ならば、類似性を除去してください。 メッセージ類似性を除去することによって、アプリケーションの可用性と拡張容易性が向上します。

詳細については、メッセージの類縁性の処理を参照してください。

IBM i アプリケーションの設計およびパフォーマンスに関する考慮事項

この情報は、アプリケーション設計、スレッド、およびストレージがパフォーマンスに及ぼす影響について理解するために使用します。

この情報は 2 つのセクションに分かれています。

- 64 ページの『アプリケーション設計に関する考慮事項』

- 65 ページの『特定のパフォーマンス問題』

アプリケーション設計に関する考慮事項

プログラム設計の悪さは、いろいろな面でパフォーマンスに影響します。 他のタスクのパフォーマンスに影響を及ぼしていても、プログラムのパフォーマンスは良好に見えることがあるため、これらの問題は検出が困難であることがあります。 以下の各セクションでは、IBM MQ for IBM i 呼び出しを行うプログラムに特有のいくつかの問題を取り上げます。

アプリケーション設計について詳しくは、49 ページの『IBM MQ アプリケーションの設計上の考慮事項』を参照してください。

メッセージ長の影響
IBM MQ for IBM i では、メッセージが最大 100 MB のデータを保持することが可能ですが、メッセージ内のデータ量は、そのメッセージを処理するアプリケーションのパフォーマンスに影響します。 アプリケーションのパフォーマンスを最大にするためには、必要なデータだけをメッセージとして送ってください。 例えば、銀行口座の借方への記入要求があった場合、 クライアント・アプリケーションからサーバー・アプリケーションへ引き渡す情報は、口座番号と記入金額だけで十分です。

メッセージ持続の影響
持続メッセージをジャーナル処理します。 メッセージをジャーナル処理するとアプリケーションのパフォーマンスが低下するため、持続メッセージは重要なデータのみに使用します。 メッセージ中のデータが、キュー・マネージャーが停止もしくは誤動作したときに廃棄してもかまわないものであれば、非持続メッセージを使用してください。

特定メッセージの検索
MQGET 呼び出しでは、通常、キューから最初のメッセージを検索します。 メッセージ記述子でメッセージと相関 ID (*MsgId* および *CorrelId*) を使用して特定のメッセージを指定する場合、キュー・マネージャーは、そのメッセージを検出するまでキューを検索する必要があります。 このような方法で MQGET 呼び出しを使用すると、アプリケーションのパフォーマンスに影響します。

可変長メッセージを含んでいるキュー
キュー上のメッセージの長さが異なる場合、アプリケーションは、メッセージのサイズを判別するために、 *BufferLength* フィールドをゼロに設定した MQGET 呼び出しを使用して、呼び出しが失敗してもメッセージ・データのサイズを戻すことができます。 その後、アプリケーションは最初の呼び出しで把握したメッセージの ID と正しいサイズのバッファーを指定して、呼び出しをもう一度行うことができます。 ただし、同じキューをサービスしている他のアプリケーションがある場合には、ユーザーのアプリケーションのパフォーマンスは低下します。なぜなら、2 番目に出す MQGET 呼び出しは、最初の呼び出しと 2 番目の呼び出しの間に別のアプリケーションによって取り出されたメッセージを検索するために時間を費やすからです。

固定長メッセージを使用できないアプリケーションでこの問題を解決するには、例えば MQINQ 呼び出しを使用して、そのキューが受け付ける最大メッセージ・サイズを見つけ、 その値を MQGET 呼び出しに使用することが考えられます。 キューのメッセージの最大サイズは、キューの **MaxMsgLen** 属性に保管されます。 しかし、このキュー属性の値は IBM MQ for IBM i が許可する最大値になる場合があり、2 GB を超えることもあるため、この方法では大量のストレージを使用する可能性があります。

同期点の頻度
1 つの同期点内で多数の MQPUT 呼び出しをコミットなしで出すプログラムは、パフォーマンス効率の問題を起こす可能性があります。 対象となるキューが、当面使用できないメッセージでいっぱいになり、 他のタスクがそのメッセージを取得するために待ち状態を続けるという事態になりかねません。 この問題は、使用されるストレージの点と、メッセージを取得しようとするタスクでスレッドが拘束されるという点で影響があります。

MQPUT1 呼び出しの使用
MQPUT1 呼び出しの使用は、キューに書き込むメッセージが 1 つしかないときに限ってください。 複数のメッセージを書き込みたい場合は、まず MQOPEN を呼び出し、 続いて一連の MQPUT を呼び出して、最後に 1 回の MQCLOSE 呼び出しを行います。

使用するスレッドの数
アプリケーションは多くのスレッドを必要とすることがあります。 各キュー・マネージャー・プロセスには、最大許容数のスレッドが割り振られます。 アプリケーションに問題がある場合、使用するスレッドが多すぎるというアプリケーションの設計が原因であることがあります。 アプリケーションが

この可能性を考慮に入れているかどうか、またアプリケーションがこうしたタイプの出来事の発生を防ぐかまたは報告する処置をとることを考慮してください。 IBM i で許可されるスレッドの最大数は 4,095 です。 ただし、デフォルトは 64 です。 IBM MQ では 63 個までのスレッドをプロセスに使用できます。

特定のパフォーマンス問題

このセクションではストレージおよびパフォーマンスの低下の問題について説明します。

ストレージに関する問題

システム・メッセージ CPF0907. Serious storage condition may exist を受け取った場合、IBM MQ for IBM i キュー・マネージャーに関連するスペースがいっぱいになっていることが考えられます。

アプリケーションまたは IBM MQ for IBM i の動作が遅いですか

アプリケーションの動作が遅いことの原因としては、ループが起こっていること、 あるいは使用できない何らかのリソースを待っていることが考えられます。 また、動作が遅いのはパフォーマンス上の問題が原因の可能性もあります。 おそらく、システムが能力の限界近くで操作されているためと考えられます。 この種の問題は、おそらくシステム負荷がピークに達する時間 (通常は、午前の中ごろと午後の中ごろ) に最悪になります (ただし、 使用しているネットワークが異なる時差の地域にまたがっている場合は、 システム負荷のピーク時が、見かけ上、異なることがあります)。

パフォーマンスの低下がシステム負荷に依存しておらず、システム負荷が軽いときでも起こることがあると分かった場合には、アプリケーション・プログラムの設計の不備がおそらく原因になっています。この問題は、特定のキューへのアクセスが起こったときにのみ、問題として現れることがあります。

この場合は、システム値 QTOTJOB および QADLTOTJ を調べてみるとよいでしょう。

次のような症状が現れたなら、IBM MQ for IBM i の動作が遅くなっていると考えられます。

- MQSC コマンドへのシステムの応答が遅い場合
- キューのサイズが繰り返し表示されることにより、大量のキュー活動が予想されるアプリケーションでキューの処理が遅いことを示している場合
- IBM MQ トレースが実行されているか。

Linux Linux on Power Systems - Little Endian アプリケーションの設計上の考慮事項

Linux on Power® Systems - Little Endian は 64 ビット・アプリケーションのみをサポートするため、32 ビット・アプリケーションのサポートは IBM MQ では提供されません。

関連概念

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

z/OS z/OS アプリケーションの設計およびパフォーマンスに関する考慮事項

アプリケーション設計は、パフォーマンスに影響する最も重要な要因の 1 つです。 このトピックを使用して、パフォーマンスに関連する設計要因の一部を理解します。

プログラム設計の悪さは、いろいろな面でパフォーマンスに影響します。 他のタスクのパフォーマンスに影響を及ぼしていても、プログラムのパフォーマンスは良好に見えることがあるため、これらの問題は検出が困難であることがあります。 以下のセクションでは、MQI 呼び出しを行うプログラムに固有のいくつかの問題を取り上げます。

アプリケーション設計について詳しくは、49 ページの『IBM MQ アプリケーションの設計上の考慮事項』を参照してください。

メッセージ長の影響

IBM MQ for z/OS では、メッセージが最大 100 MB のデータを保持することが可能ですが、メッセージ内のデータ量は、そのメッセージを処理するアプリケーションのパフォーマンスに影響します。 アプリケーションのパフォーマンスを最大限に引き出すには、不可欠のデータだけをメッセージに入れて送信してください。 例えば、銀行預金口座に記入する要求では、クライアントからサーバー・アプリケーションに渡す必要のある情報は、口座番号と借方の金額だけです。

メッセージ持続の影響

持続メッセージはログに記録されます。 メッセージをログに記録すると、アプリケーションのパフォーマンスが低下します。 したがって、重要なデータの場合のみ持続メッセージを使用してください。 メッセージ中のデータが、キュー・マネージャーが停止もしくは誤動作したときに廃棄してもかまわないものであれば、 非持続メッセージを使用してください。

持続メッセージのデータは、ログ・バッファーに書き込まれます。 それらのバッファーは、次のような場合にログ・データ・セットに書き込まれます。

- コミットが発生する
- メッセージが取得されるか同期点外に置かれる
- WRTHRSH バッファーが満杯になる

多くのメッセージを 1 つの作業単位で処理すると、メッセージが作業単位ごとに 1 つ処理される場合や、同期点外で処理される場合に比べて、入出力が少なくなります。

特定メッセージの検索

MQGET 呼び出しは、通常、キューから最初のメッセージを取り出します。 メッセージおよび相関 ID を使用する場合 (**MsgId** および **CorrelId**) 特定のメッセージを指定するためにメッセージ記述子で指定すると、キュー・マネージャーはそのメッセージが見つかるまでキューを検索します。 MQGET をこのように使用すると、アプリケーションのパフォーマンスに影響が出ます。 それは、IBM MQ が特定のメッセージを検索するためにキュー全体を走査しなければならない場合があるからです。

IndexType キュー属性を使用して、キュー上の MQGET 操作の速度を上げるために使用できる索引をキュー・マネージャーが保守するように指定できます。 ただし、索引の保守のためにわずかながらパフォーマンスが低下するので、索引は使用する必要がある場合にのみ生成するようにしてください。 メッセージ ID または相関 ID の索引を作成することも、メッセージが順番に取り出されるキューについては索引を作成しないこともできます。 同じキー値で多くの項目を作るのではなく、異なるキー値を数多く使用してください。 例えば、 Balance1、Balance2、 Balance3 などのように分ける方が、 3 つとも Balance とするよりもよいでしょう。 共用キューの場合は、正しい **IndexType** が必要です。 **IndexType** キュー属性について詳しくは、 索引タイプを参照してください。

索引付きキューを使って、キュー・マネージャーの再始動時への影響を避けるには、 CSQ6SYSP マクロで QINDXBLD(NOWAIT) パラメーターを使用します。 これによって、キュー・マネージャーは、キュー索引の構築の完了を待機することなく再始動し、完了します。

IndexType 属性およびその他のオブジェクト属性について詳しくは、オブジェクトの属性を参照してください。

可変長メッセージを含んでいるキュー

予期されるメッセージのサイズに合ったバッファー・サイズを使用して、メッセージを取得してください。 メッセージが長すぎることを示す戻りコードを受け取る場合、より大きなバッファーを取得してください。 取得がこの方法で失敗する場合、戻されるデータ長は、変換されなかったメッセージ・データのサイズです。 MQGET 呼び出しで MQGMO_CONVERT を指定し、データが変換中に拡張する場合も、データがバッファーに収まらないことがあり、その場合はさらにバッファーのサイズを大きくする必要があります。

バッファー長ゼロの MQGET を送出すれば、メッセージのサイズが戻されるので、アプリケーションは、このサイズのバッファーを取得してから、取得を再送出できます。 キューを処理するアプリケーションが複数ある場合、オリジナルのアプリケーションが取得を再送出するときには、 別のアプリケーションがすでにメッセージを処理している場合があります。 大きなメッセージを持つことがある場合は、それらのメッセージのためだけに大きなバッファーを確保し、そのメッセージが処理されたらそのバッファーを解放する必要がある場合があります。 すべてのアプリケーションが大きなバッファーを確保すれば、 仮想ストレージに関する問題が減少することになります。

固定長メッセージを使用できないアプリケーションでこの問題を解決するには、例えば MQINQ 呼び出しを使用して、そのキューが受け付ける最大メッセージ・サイズを見つけ、 その値を MQGET 呼び出しに使用することが考えられます。 キューのメッセージの最大サイズは、キューの **MaxMsgL** 属性に保管されます。 ただし、 **MaxMsgL** の値は 100 MB (IBM MQ for z/OS で許可される最大値) に達する可能性があるため、この方法では大量のストレージを使用する可能性があります。

注 : 大きなメッセージがキューに書き込まれた後に、 **MaxMsgL** パラメーターを小さくすることができます。 例えば、100 MB のメッセージを書き込んでから、**MaxMsgL** を 50 バイトに設定することができます。 つまり、アプリケーションが予期する以上の大きさのメッセージを取得できる、 ということです。

同期点の頻度

同期点の中で、 コミットせずに多くの MQPUT 呼び出しを行うプログラムは、 パフォーマンス上の問題を引き起こすことがあります。 対象となるキューが、当面使用できないメッセージでいっぱいになり、 他のタスクがそのメッセージを取得するために待ち状態を続けるという事態になりかねません。 これは、使用されるストレージの面からも、 メッセージを取得しようとするタスクでスレッドが拘束されるという面からも、 好ましいことではありません。

一般に、複数のアプリケーションが 1 つのキューを処理している場合、最高のパフォーマンスが得られるのは、通常、同期点ごとに以下のいずれかがある場合です。

- 100 の短メッセージ (1 KB 未満)
- より大きなメッセージ (100 KB) が 1 つ

:NONE. キューを処理するアプリケーションが 1 つしかない場合は、作業単位あたりのメッセージ数がもっと多くなければなりません。

MAXUMSGS キュー・マネージャー属性を使用して、単一のリカバリー単位内でタスクが読み取りまたは書き込みできるメッセージの数を制限することができます。 この属性については、MQSC コマンドの **ALTER QMGR** コマンドを参照してください。

MQPUT1 呼び出しの利点

MQPUT1 呼び出しの使用は、キューに書き込むメッセージが 1 つしかないときに限ってください。 複数のメッセージを書き込みたい場合は、まず MQOPEN を呼び出し、 続いて一連の MQPUT を呼び出して、最後に 1 回の MQCLOSE 呼び出しを行います。

キュー・マネージャーが保持できるメッセージの数

ローカル・キュー

キュー・マネージャーが保持できるローカル・メッセージの数は、基本的にはページ・セットのサイズです。 最大で 100 ページ・セットを持つことがでます (ただし、ページ・セット 0 とページ・セット 1 は、システム関連のオブジェクトとキュー用にすることをお勧めします)。 ページ・セットは拡張フォーマットで使用でき、ページ・セットの容量は増やすことができます。

共用キュー

共用キューの容量は、カップリング・ファシリティー (CF) のサイズに依存します。 基礎的な記憶単位がエントリーおよびエレメントである場合、IBM MQ は CF リスト構造体を使用します。 各メッセージは 1 つのエントリーおよび複数のエレメントとして保管され、関連する MQMD とその他のメッセージ・データがそこに含まれます。 1 つのメッセージでコンシュームされるエレメントの数は、メッセージのサイズに依存し、さらに CFLEVEL(5) の場合は MQPUT 時に有効なオフロード規則にも依存します。 メッセージ・データが Db2 または SMDS にオフロードされる場合、必要なエレメントの数はより少なくなります。 メッセージが既にオフロードされた場合、メッセージ・データ・アクセスがより遅くなります。 メッセージ・オフロードに関連するパフォーマンスおよび CPU オーバーヘッドの詳しい比較については、Performance Supportpac MP1H を参照してください。

何がパフォーマンスに影響を与えるか

パフォーマンスは、どれだけ高速にメッセージを処理できるか、という意味の場合もありますが、メッセージ当たりに必要な CPU の量を意味することもあります。

何がメッセージの処理速度に影響するか

持続メッセージの場合、最も大きな影響は、ログ・データ・セットの速度です。 ログ・データ・セットの速度は、それを格納している DASD に依存します。 したがって、競合を減らすためにログ・データ・セットを使用頻度の少ないボリュームに配置する場合は注意が必要です。 MQ ログをストライピングすると、入出力ごとに複数のページが書き込まれたとき、ログのパフォーマンスが向上します。 また、Z High Performance Fibre 接続 (zHPF) には、入出力サブシステムが使用中の場合、入出力応答時間に対して優れたパフォーマンスを発揮します。

メッセージの取得および書き込み要求がある場合は、キューの整合性を保持するために、その要求の間はキューへのアクセスはロックされます。 計画を立案する場合は、要求全体に対してキューをロックすることを検討してください。 つまり、書き込み時間が 100 マイクロ秒で、毎秒 10,000 を超える要求がある場合は、1 秒の遅延時間が発生します。 実際にはこれよりも良くなりますが、原則としてはこれで十分です。 異なる複数のキューを使用してパフォーマンスを改善することができます。

これに対して考えられる理由は以下のとおりです。

- すべての CICS トランザクションが使用する共通の応答キューを使用している。
- 各 CICS トランザクションにはキューに対する固有の応答が与えられている。
- CICS 領域用のキューに対する応答、および CICS 領域内のすべてのトランザクションが、このキューを使用している。

答えは、毎秒ごとの要求の数と、要求の応答時間によって異なります。

メッセージをページ・セットから読み取る必要がある場合は、メッセージがバッファー・プールにある場合と比較して遅くなります。 バッファー・プールに収容できないほど多くのメッセージがある場合、メッセージはディスクにあふれ出します。 そのため、必ずバッファー・プールは、寿命の短いメッセージに十分な大きさにする必要があります。 何時間も後に処理するメッセージがあると、それらのメッセージがディスクにあふれ出す可能性が高いので、これらのメッセージの取得には、メッセージがバッファー・プールにある場合よりも時間がかかることを予期しておく必要があります。

共用キューの場合、メッセージの速度はカップリング・ファシリティーの速度に依存します。 物理プロセッサー内の CF は、外部の CF よりも高速になる可能性があります。 物理プロセッサー内の CF の応答時間は、その CF の負荷に依存します。 例えば、Hursley システムの場合、CF の負荷が 17% のときの応答時間は 14 マイクロ秒でした。 この CF の負荷が 95% のときは、応答時間は 45 マイクロ秒でした。

MQ 要求で多くの CPU が使用されると、メッセージを処理する速度に影響する可能性があります。 論理区画 (LPAR) が CPU の制約を受ける場合、CPU を待つためにアプリケーションは遅延します。

メッセージ当たりの CPU 使用量

一般に、メッセージが大きくなると使用される CPU も多くなるので、可能であれば大きな (x MB) メッセージは避けてください。

キューから特定のメッセージを取得する場合は、そのキューに索引を付ける必要があります。 これにより、キュー・マネージャーは直接そのメッセージにアクセスできます (したがって、キューの全体ス

キャンが回避される可能性があります)。 キューに索引が付けられていない場合、そのキューは、メッセージを探しながら先頭からスキャンされます。 そのキューに 1000 個のメッセージがある場合、1000 個のメッセージをすべてスキャンする必要がある可能性があります。 この結果、必要以上に CPU が使用されることになります。

TLS を使用しているチャネルの場合は、メッセージの暗号化のために、さらにコストがかかります。

MQ V7 では、 **CORRELID** または **MSGID** に加えて、セレクター・ストリングによってメッセージを選択できます。 すべてのメッセージを調べる必要があるため、キューに数多くのメッセージがある場合、これは高くつきます。

アプリケーションが PUT1 PUT1 よりも OPEN PUT PUT CLOSE を実行する方がより効率的です。

CICS でのトリガー操作

トリガーされたキュー宛のメッセージのメッセージ到着速度が低い場合、最初にトリガーを使用するのが効率的です。 メッセージ到着速度が毎秒 10 個のメッセージを超える場合は、最初のトランザクションをトリガーし、次にそのトランザクションで 1 つのメッセージを処理し、そして次のメッセージを取得する、というようにする方がより効率的です。 メッセージが短期間経過しても到着しなかった場合 (例えば、0.1 から 1 秒の間)、トランザクションは終了します。 スループットが高い場合、実行中の複数のトランザクションでメッセージを処理し、メッセージの蓄積を防止する必要があります。 この場合、作成されたすべてのトリガー・メッセージごとにトリガー・メッセージの書き込みと取得が必要になるので、メッセージのコストは実際には 2 倍になります。

サポートされる接続数または同時ユーザー数

各接続は、キュー・マネージャー内の仮想ストレージを使用するので、同時ユーザーの数が増えると、使用されるストレージも多くなります。 非常に大きなバッファー・プールと、多数のユーザーが必要な場合は、仮想ストレージの制約を受ける可能性があり、バッファー・プールのサイズを小さくする必要が生じる場合があります。

セキュリティーが使用されている場合、キュー・マネージャーは情報を長期間、キュー・マネージャー内にキャッシュします。 キュー・マネージャー内で使用される仮想ストレージの量に影響が出ます。

CHINIT は、最大で約 10,000 個の接続をサポートできます。 これは、仮想ストレージにより制限を受けます。 接続で使用されるストレージが多い場合 (TLS を使用する場合など)、接続ごとにストレージが増加するため、 **CHINIT** がサポートできる接続の数が少なくなります。 大きなメッセージを処理する場合は、**CHINIT** がサポートするメッセージが少なくなるように、**CHINIT** のバッファー用に必要なストレージが多くなります。

リモート・キュー・マネージャーとの接続の方がクライアント接続よりも効率的です。 例えば、すべての MQ クライアント要求には、2 つのネットワーク・フローが必要です (1 つは要求用、もう 1 つは応答用)。 リモート・キュー・マネージャーへのチャネルの場合、応答が戻ってくるまでにネットワークで 50 回送信する可能性もあります。 大規模なクライアント・ネットワークを検討している場合は、分散ボックスでコンセントレーター・キュー・マネージャーを使用し、1 つのチャネルがコンセントレーターに入出力する方が効率が良くなる可能性があります。

パフォーマンスに影響を与えるその他の事項

ログ・データ・セットには、サイズとして少なくとも 1000 シリンダーが必要です。 ログがこれよりも小さいと、チェックポイント・アクティビティーの頻度が多くなりすぎることがあります。 負荷がかかっているシステムでは、一般にチェックポイントは 15 分ごと、あるいはそれよりも長くなりますが、スループットが非常に高い場合には、これよりも短くなる可能性があります。 チェックポイントが発生するとバッファー・プールがスキャンされ、「古い」メッセージおよび変更されたページがディスクに書き込まれます。 チェックポイントの発生頻度が多すぎる場合は、それによってパフォーマンスに影響が出る可能性があります。 LOGLOAD の値がチェックポイントの頻度に影響することもあります。 キュー・マネージャーが異常終了すると、キュー・マネージャーは再始動時に 3 チェックポイント戻って読み取りを行う必要があることもあります。 最適なチェックポイント間隔は、チェックポイントが取られたときのアクティビティーと、キュー・マネージャーの再始動時に読み取る必要がある可能性のあるログ・データの量の間のバランスです。

チャネルの始動時には、かなり大きなオーバーヘッドがかかります。 通常は、チャネルを頻繁に始動および停止するのではなく、1 つのチャネルを始動して、そのチャネルを接続したままにしておくことをお勧めします。

関連情報

MP1K: IBM MQ for z/OS 9.0 パフォーマンス・レポート

z/OS IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション

この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

- IMS アプリケーションで同期点と MQI 呼び出しを使用する場合は、70 ページの『IBM MQ を使用した IMS アプリケーションの作成』を参照してください。
- IBM MQ - IMS ブリッジを使用するアプリケーションを作成する場合は、74 ページの『IMS ブリッジ・アプリケーションの作成』を参照してください。

IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーションについて詳しくは、以下のリンクを参照してください。

- 70 ページの『IBM MQ を使用した IMS アプリケーションの作成』
- 74 ページの『IMS ブリッジ・アプリケーションの作成』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

z/OS IBM MQ を使用した IMS アプリケーションの作成

IMS アプリケーションで IBM MQ を使用する際には、さらに考慮事項があります。これには、どの MQ API 呼び出しを使用できるか、および同期点に使用されるメカニズムが含まれます。

IBM MQ for z/OS での IMS アプリケーションの作成について詳しくは、以下のリンクを参照してください。

- 71 ページの『IMS アプリケーションにおける同期点』
- 71 ページの『IMS アプリケーションにおける MQI 呼び出し』

制約事項

IMS アダプターを使用するアプリケーションが使用できる IBM MQ API 呼び出しには、いくつかの制約事項があります。

以下の IBM MQ API 呼び出しは、IMS アダプターを使用するアプリケーションではサポートされていません。

- MQCB
- MQCB_FUNCTION
- MQCTL

関連概念

74 ページの『IMS ブリッジ・アプリケーションの作成』
このトピックでは、IBM MQ - IMS ブリッジを使用するアプリケーションの作成について説明します。

z/OS *IMS アプリケーションにおける同期点*

IMS アプリケーションでは、IOPCB および CHKP (checkpoint) に対する GU (get unique) などの IMS 呼び出しを使用して同期点を確立します。

直前のチェックポイント以降のすべての変更をバックアウトするには、IMS ROLB (rollback) 呼び出しを使用できます。 詳しくは、 IMS 資料の ROLB 呼び出し を参照してください。

このキュー・マネージャーは、2 フェーズ・コミット・プロトコルの参加プログラムです。 IMS 同期点管理プログラムはそのためのコーディネーターです。

すべてのオープン状態のハンドルは、IMS アダプターによって同期点でクローズされます (バッチまたは非メッセージ・ドリブン BMP 環境の場合を除く)。 これは、別のユーザーが次の作業単位を開始することもあり、また、IBM MQ のセキュリティー検査が、(MQPUT または MQGET 呼び出し時でなく) MQCONN、MQCONNX および MQOPEN 呼び出し時に実行されるためです。

しかし、入力待ち (WFI (Wait-for-Input)) または疑似入力待ち (PWFI (pseudo Wait-for-Input)) 環境では、IMS は、次のメッセージが到着するか、QC 状況コードがアプリケーションに戻されるまでは、IBM MQ にハンドルをクローズするよう通知しません。 アプリケーションが IMS 領域内で待機していて、これらのハンドルのいずれかが、トリガーされたキューに属している場合、それらのキューはオープンしているのでトリガーは発生しません。 この理由のため、 WFI または PWFI 環境で実行するアプリケーションは、次のメッセージ用に IOPCB に対する GU を行う前に、明示的な MQCLOSE を キュー・ハンドルに対して実行する必要があります。

IMS アプリケーション (BMP または MPP) が MQDISC 呼び出しを発行した場合は、オープン・キューはクローズされますが、暗黙の同期点はとられません。 アプリケーションが正常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のコミットが行われます。 アプリケーションが異常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のバックアウトが行われます。

z/OS *IMS アプリケーションにおける MQI 呼び出し*

この情報を使用して、サーバー・アプリケーションおよび照会アプリケーションにおける MQI 呼び出しの使用方法について学習します。

この節では、以下のタイプの IMS アプリケーションで MQI 呼び出しを使用する方法について説明します。

- 71 ページの『サーバー・アプリケーション』
- 74 ページの『照会アプリケーション』

サーバー・アプリケーション

MQI サーバー・アプリケーション・モデルの概要を次に示します。

```
Initialize/Connect
```

```
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

サンプル・プログラム CSQ4ICB3 では、C/370 での、このモデルを使用した BMP の実装方式を示します。プログラムは、まず IMS、次に IBM MQ の順で通信を確立します。

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

IMS 初期設定では、プログラムがメッセージ・ドリブン、バッチ指向のいずれの BMP として呼び出されているかが判定され、それぞれの場合に応じて、IBM MQ キュー・マネージャーの接続とキュー・ハンドルが制御されます。

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if
```

```
Return to calling function
```

IBM MQ 初期設定では、キュー・マネージャーに接続して、キューをオープンします。 メッセージ・ドリブンの BMP では、それぞれの IMS 同期点が取られてから呼び出されます。バッチ指向の BMP では、プログラムの始動時にのみ呼び出されます。

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

MPP 内のサーバー・モデルの実装方式は、MPP が呼び出しごとに 1 つの作業単位を処理するという事実に影響されます。 これは、同期点 (GU) がとられるとき、接続とキュー・ハンドルがクローズされ、次の IMS メッセージが送達されるからです。 この制限は、以下のいずれかの方法によって部分的に克服することができます。

- **1 つの作業単位内で多数のメッセージを処理する方法**

  これには、以下の処理が含まれます。

  – メッセージの読み取り
  – 必要な更新の処理
  – 応答の書き込み

  同期点がとられる時点で、1 つのループのなかで、すべてのメッセージが処理されるか、または最大数のメッセージ群が処理されるまで、上記の処理を行います。

  特定のタイプのアプリケーション (例えば、単純なデータベースの更新または照会) についてのみ、この方法が適用できます。 MQI 応答メッセージは、処理中の MQI メッセージの発信元の許可によって書き込むことができますが、IMS リソースの更新がセキュリティーに及ぼす影響には注意する必要があります。

- **MPP を呼び出すごとに 1 つのメッセージを処理し、使用可能なすべてのメッセージを処理するための MPP のスケジュールを複数作成できるようにする方法**

  IBM MQ IMS トリガー・モニター・プログラム (CSQQTRMN) は、 IBM MQ キューにメッセージがあって、それにサービスを提供するアプリケーションがない場合に、MPP トランザクションをスケジュールするために使用します。

  トリガー・モニターが MPP を開始すると、キュー・マネージャー名とキュー名は、以下の COBOL コードの抜粋で示すように、プログラムに渡されます。

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
```

```
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME        ='
MQTMC-QNAME  OF MQTMC '='.
```

タスクの実行が長時間になることが予想されるサーバー・モデルは、バッチ処理領域内のサポートをお勧めします。ただし、BMP は CSQQTRMN で起動することができません。

照会アプリケーション

照会または更新作業を開始する一般的な IBM MQ アプリケーションは以下の処理を行います。

- ユーザーからのデータの収集
- 1 つ以上の IBM MQ メッセージの書き込み
- 応答メッセージの読み取り (待機しなければならない場合もある)
- ユーザーへの応答

IBM MQ キューに書き込まれたメッセージは、コミットされるまでは他の IBM MQ アプリケーションで使用できないため、それらのメッセージを同期点から外すか、あるいは IMS アプリケーションを 2 つのトランザクションに分割する必要があります。

照会で 1 つのメッセージの書き込みを処理する場合、*no syncpoint* オプションを使用できます。しかし、照会がより複雑な場合、またはリソースの更新が絡む場合には、同期点制御が稼働していないときに障害が発生すると、整合性上の問題が起こる可能性があります。

これを克服するために、プログラム間メッセージ通信を使用する MQI 呼び出しを使用して IMS MPP トランザクションを分割することができます。これについては、「*IMS システム間連絡 (ISC)*」を参照してください。 これによって、照会プログラムが MPP に実装できるようになります。

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS IMS ブリッジ・アプリケーションの作成

このトピックでは、IBM MQ - IMS ブリッジを使用するアプリケーションの作成について説明します。

IBM MQ-IMS ブリッジに関する情報については、「IMS ブリッジ」を参照してください。

IBM MQ for z/OS 上での IMS ブリッジ・アプリケーションの作成について詳しくは、以下のリンクを参照してください。

- 75 ページの『IMS ブリッジがメッセージを処理する方法』

- 916 ページの『IBM MQ を使用した IMS トランザクション・プログラムの作成』

関連概念

70 ページの『IBM MQ を使用した IMS アプリケーションの作成』
IMS アプリケーションで IBM MQ を使用する際には、さらに考慮事項があります。これには、どの MQ API 呼び出しを使用できるか、および同期点に使用されるメカニズムが含まれます。

z/OS *IMS* ブリッジがメッセージを処理する方法

IBM MQ-IMS ブリッジを使用して IMS アプリケーションにメッセージを送信する場合、メッセージを特殊なフォーマットで構成する必要があります。

また、ターゲット IMS システムの XCF グループとメンバー名を指定するストレージ・クラスで定義されている IBM MQ キューにもメッセージを書き込む必要があります。 これらは、MQ-IMS ブリッジ・キュー、または簡単に**ブリッジ**・キューとして知られています。

IBM MQ-IMS ブリッジを QSGDISP(QMGR) を指定して定義した場合、または QSGDISP(SHARED) と NOSHARE オプションを指定して定義した場合は、ブリッジ・キューに対する排他的な入力権限 (MQOO_INPUT_EXCLUSIVE) がブリッジになければなりません。

IMS アプリケーションにメッセージを送信する前に、ユーザーは IMS にサインオンする必要はありません。セキュリティー・チェックには、MQMD 構造体の *UserIdentifier* フィールドのユーザー ID が使われます。 チェックのレベルは、IBM MQ が IMS に接続されるときに決定されます。詳しくは、IMS ブリッジに関するアプリケーション・アクセス制御を参照してください。 これにより、疑似サインオンをインプリメントすることが可能になります。

IBM MQ - IMS ブリッジは、次のタイプのメッセージを受け付けます。

- メッセージには IMS トランザクション・データと MQIIH 構造体 (MQIIH を参照) が含まれています。

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

注:

1. 角括弧 [ ] は、任意指定の複数セグメントを表します。
2. MQIIH 構造体を使用するには、MQMD 構造体の *Format* フィールドを MQFMT_IMS に設定してください。

- IMS トランザクション・データが入っているが、MQIIH 構造体は入っていないメッセージ。

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ は、メッセージ・データをチェックして、LL バイト数と MQIIH (存在する場合) の長さの合計が、メッセージの長さに等しいかどうかを確認します。

IBM MQ - IMS ブリッジは、ブリッジ・キューからメッセージを読み取ると、それらのメッセージを次のように処理します。

- メッセージに MQIIH 構造体が含まれている場合、ブリッジは MQIIH を検証し (MQIIH を参照)、OTMA ヘッダーを作成してから IMS にメッセージを送信します。 トランザクション・コードは、入力メッセージに指定されます。 これが LTERM の場合、IMS は応答として DFS1288E メッセージを出します。 トランザクション・コードがコマンドを表す場合、IMS は、そのコマンドを実行するか、もしくはメッセージを IMS 内のトランザクション用のキューに入れます。
- そのメッセージに IMS トランザクション・データは入っているが、MQIIH 構造体が入っていない場合は、IMS ブリッジは次のように仮定します。
  - トランザクション・コードはユーザー・データのバイト 5 から 12 までである。
  - トランザクションは非会話型である。
  - トランザクションはコミット・モード 0 (コミット後送信) である。
  - MQMD 内の *Format* は *MFSMapName* (入力上) として使用される。

– セキュリティー・モードは MQISS_CHECK である。

応答メッセージも MQIIH 構造体なしで作成され、MQMD の *Format* は IMS 出力の *MFSMapName* から取得されます。

IBM MQ - IMS ブリッジでは、各 IBM MQ キューにつき 1 つまたは 2 つの Tpipe を使用します。

- 同期化 Tpipe は、コミット・モード 0 (COMMIT_THEN_SEND) を使用するすべてのメッセージに使用されます (これらは、IMS /DIS TMEMBER クライアント TPIPE xxxx コマンドの状況フィールドに SYN と表示されます)。
- 非同期 Tpipe は、コミット・モード 1 (SEND_THEN_COMMIT) のメッセージに使用します。

Tpipe は、初めて使用するときに IBM MQ によって作成されます。 非同期 Tpipe は、IMS を再始動するまで削除されません。 同期 Tpipe は、IMS をコールド・スタートするまで削除されません。 ユーザーがこれらの Tpipe を削除することはできません。

IBM MQ - IMS ブリッジのメッセージの扱い方について詳しくは、以下のトピックを参照してください。

- 76 ページの『IBM MQ メッセージと IMS トランザクション・タイプの対応関係』
- 77 ページの『メッセージを IMS キューに書き込めない場合』
- 77 ページの『IMS ブリッジのフィードバック・コード』
- 77 ページの『IMS ブリッジからのメッセージ内の MQMD フィールド』
- 79 ページの『IMS ブリッジからのメッセージ内の MQIIH フィールド』
- 79 ページの『IMS からの応答メッセージ』
- 80 ページの『IMS トランザクションにおける代替応答 PCB の使用』
- 80 ページの『IMS からの非送信請求メッセージの送信』
- 80 ページの『メッセージのセグメント化』
- 80 ページの『IMS ブリッジとの間のメッセージのデータ変換』

関連概念

916 ページの『IBM MQ を使用した IMS トランザクション・プログラムの作成』
IBM MQ を介して IMS トランザクションを処理するために必要なコーディングは、 IMS トランザクションが必要とするメッセージ形式と、トランザクションが返すことができる応答の範囲によって異なります。しかし、アプリケーションが IMS 画面形式制御情報を扱うときには、考慮すべき点がいくつかあります。

z/OS *IBM MQ* メッセージと *IMS* トランザクション・タイプの対応関係

表は IBM MQ メッセージと IMS トランザクション・タイプの対応関係を示しています。

表 *4. IBM MQ* メッセージと *IMS* トランザクション・タイプの対応関係

IBM MQ メッセージ・タイプ	**コミット後送信 (モード 0) - 同期 IMS Tpipe を使用**	**送信後コミット (モード 1) - 非同期 IMS Tpipe を使用**
持続 IBM MQ メッセージ	• リカバリー可能な全機能トランザクション • リカバリー不能なトランザクションの場合は IMS で拒否	• ファースト・パス・トランザクション • 会話型トランザクション • 全機能トランザクション
非持続 IBM MQ メッセージ	• リカバリー不能な全機能トランザクション • リカバリー可能なトランザクションは IMS V8 および APAR PQ61404、および以降すべてのバージョンの IMS で許可	• ファースト・パス・トランザクション • 会話型トランザクション • 全機能トランザクション

注 : IMS コマンドは、コミット・モードが 0 の持続 IBM MQ メッセージを使用できません。 詳しくは、 コミット・モード (commitMode) を参照してください。

z/OS メッセージを *IMS* キューに書き込めない場合

メッセージを IMS キューに書き込めない場合に取られる処置について説明します。

メッセージを IMS キューに書き込めない場合は、IBM MQ によって次の処理が行われます。

- メッセージが無効なために IMS に書き込めない場合は、そのメッセージは送達不能キューに書き込まれ、メッセージがシステム・コンソールに送られます。
- メッセージが有効であるのに、IMS によって拒否された場合、IBM MQ はシステム・コンソールにエラー・メッセージを送信します。このメッセージは IMS センス・コードを含んでおり、IBM MQ メッセージは送達不能キューに書き込まれます。 IMS センス・コードが 001A の場合、IMS はその障害の理由を含む IBM MQ メッセージを応答先キューに送信します。

  注 : 上のリストのような環境では、IBM MQ が何らかの理由で送達不能キューにメッセージを書き込めないとき、メッセージは発信元の IBM MQ キューに返されます。 エラー・メッセージがシステム・コンソールに送られ、そのキューから、他のメッセージは送られません。

  メッセージを再送するには、次の **いずれか** を行ってください。

  - そのキューに対応する IMS 内の Tpipes を停止し、再始動する。
  - そのキューを GET(DISABLED) に変更し、再び GET(ENABLED) に変更する。
  - IMS または OTMA を停止、再始動する。
  - IBM MQ サブシステムを停止し、再始動する。
- メッセージが、メッセージ・エラー以外の理由で IMS に拒否された場合、IBM MQ メッセージは元のキューに返され、IBM MQ はそのキューの処理を停止します。そして、システム・コンソールにはエラー・メッセージが送られます。

  例外報告メッセージが必要な場合は、そのブリッジは発信元の許可によりそのメッセージを応答先キューに書き込みます。 例外報告メッセージを応答先キューに書き込めない場合は、その報告メッセージはブリッジの許可により送達不能キューに書き込まれます。 メッセージ DLQ に書き込めない場合、そのメッセージは廃棄されます。

z/OS *IMS* ブリッジのフィードバック・コード

IMS センス・コードは通常、CSQ2001I のような IBM MQ コンソール・メッセージでは 16 進形式の出力となります (例えば、センス・コード 0x001F)。 IBM MQ フィードバック・コード (送達不能キューに書き込まれるメッセージの送達不能ヘッダーの中に見られる) は、10 進数です。

IMS ブリッジのフィードバック・コードの範囲は 301 から 399 まで、または、NACK センス・コード 0x001A の場合は 600 から 855 までです。 このフィードバック・コードは、IMS-OTMA センス・コードから次のようにマッピングされます。

1. IMS-OTMA センス・コードが 16 進数から 10 進数に変換されます。
2. 1 の計算結果に 300 が加算され、 IBM MQ *Feedback* コードが示されます。
3. IMS-OTMA センス・コード 0x001A、10 進数 26 は特殊なケースです。 範囲 600 から 855 までのフィードバック・コードが生成されます。
   a. IMS-OTMA 理由コードが 16 進数から 10 進数に変換されます。
   b. a の計算結果に 600 を足して、IBM MQ フィードバック・コードを作成します。

IMS-OTMA センス・コードについては、「NAK メッセージ用の OTMA センス・コード」を参照してください。

z/OS *IMS* ブリッジからのメッセージ内の *MQMD* フィールド

IMS ブリッジからのメッセージ内の MQMD フィールドについて説明します。

元のメッセージの MQMD は、OTMA ヘッダーの User Data セクションで、IMS によって渡されます。 メッセージが IMS から発信している場合、これは IMS 宛先解決出口によって構築されます。 IMS から受信されるメッセージの MQMD は次のように構築されます。

StrucID
"MD "

バージョン
MQMD_VERSION_1

レポート
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
MQIIH の Flags フィールドで MQIIH_PASS_EXPIRATION が設定されている場合は、このフィールドには残りの満了時間が含まれ、設定されていない場合は、このフィールドは MQEI_UNLIMITED に設定されます。

Feedback
MQFB_NONE

Encoding
MQENC.Native (z/OS システムのエンコード)

CodedCharSetId
MQCCSI_Q_MGR (z/OS システムの CodedCharSetID)。

Format
入力メッセージの MQMD.Format が MQFMT_IMS の場合は MQFMT_IMS。その他の場合は、IOPCB.MODNAME。

優先順位
入力メッセージの MQMD.Priority。

Persistence
コミット・モードによって異なります。CM-1 の場合は、入力メッセージの MQMD.Persistence。CM-0 の場合は、IMS メッセージの回復可能性と一致します。

MsgId
MQRO_PASS_MSG_ID の場合、MQMD.MsgId。その他の場合は、New MsgId (デフォルト)。

CorrelId
MQRO_PASS_CORREL_ID の場合は、入力メッセージの MQMD.CorrelId。その他の場合は、入力メッセージの MQMD.MsgId (デフォルト)。

BackoutCount
0

ReplyToQ
ブランク

ReplyToQMgr
ブランク (MQPUT の間はキュー・マネージャーによってローカル・キュー・マネージャー名に設定されます)。

UserIdentifier
入力メッセージの MQMD.UserIdentifier。

AccountingToken
入力メッセージの MQMD.AccountingToken。

ApplIdentityData
入力メッセージの MQMD.ApplIdentityData。

PutApplType
エラーがない場合、MQAT_XCF。その他の場合は、MQAT_BRIDGE。

PutApplName
エラーがない場合、<XCFgroupName><XCFmemberName>。その他の場合は、QMGR 名。

PutDate
メッセージを書き込んだ日付

PutTime
メッセージを書き込んだ時刻

ApplOriginData
ブランク

z/OS *IMS* ブリッジからのメッセージ内の *MQIIH* フィールド
IMS ブリッジからのメッセージ内の MQIIH フィールドについて説明します。

IMS から受信されるメッセージの MQIIH は次のように構築されます。

StrucId
"IIH "

バージョン
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat がブランクでない場合は、入力メッセージの MQIIH.ReplyToFormat。ブランクの場合は、IOPCB.MODNAME。

フラグ
0

LTermOverride
OTMA ヘッダーの LTERM 名 (Tpipe)。

MFSMapName
OTMA ヘッダーのマップ名。

ReplyToFormat
ブランク

Authenticator
応答メッセージが MQ-IMS ブリッジ・キューに書き込まれる場合は入力メッセージの MQIIH.Authenticator。書き込まれない場合はブランク。

TranInstanceId
会話中の場合は、OTMA ヘッダーの会話 ID またはサーバー・トークン。 バージョンが V14 より前の IMS では、会話中でない場合、このフィールドは常にヌルになります。 IMS V14 以降では、会話中でない場合でも、このフィールドは IMS によって設定されることがあります。

TranState
会話中の場合は、"C"。それ以外の場合はブランク。

CommitMode
OTMA ヘッダーのコミット・モード ("0" または "1")。

SecurityScope
ブランク

Reserved
ブランク

z/OS *IMS* からの応答メッセージ
IMS トランザクション ISRT がその IOPCB へ送られると、メッセージは発信元の LTERM または TPIPE に戻されます。

これらは IBM MQ では応答メッセージとして表示されます。 IMS からの応答メッセージは、元のメッセージで指定された応答先キューに書き込まれます。 応答メッセージを応答先キューに書き込めない場合は、そのメッセージはブリッジの許可により送達不能キューに書き込まれます。 応答メッセージを送達不能キューに書き込めない場合は、メッセージが受信できないことを示す否定応答が IMS に送られます。 この時点で、そのメッセージに対する責任は IMS に戻ります。 コミット・モード 0 を使用している場合、Tpipe

からのメッセージはブリッジには送られず、IMS キュー上に留まります。つまり、再始動するまではメッセージは送られません。 コミット・モード 1 を使用している場合、他の作業を続けることができます。

応答が MQIIH 構造体を持っている場合、その形式は MQFMT_IMS となります。この構造体を持っていない場合は、その形式はメッセージを挿入するときに使用される IMS MOD 名により指定されます。

z/OS *IMS* トランザクションにおける代替応答 *PCB* の使用

IMS トランザクションが代替応答 PCB (ALTPCB に対して ISRT を使用する場合、または変更可能な PCB に CHNG 呼び出しを発行する場合) を使用し、そのメッセージを転送する必要があるかどうかを判別するために、プリルーティング出口 (DFSYPRX0) が呼び出されます。

メッセージの再経路指定が必要な場合は、宛先解決出口 (DFSYDRU0) が呼び出され、宛先の確認とヘッダー情報の作成が行われます。これらの出口プログラムについては、IMS における OTMA 出口の使用および事前経路指定出口 DFSYPRX0 を参照してください。

出口でアクションが取られない限り、 IBM MQ キュー・マネージャーから開始された IMS トランザクションからのすべての出力は、IOPCB または ALTPCB に関係なく、同じキュー・マネージャーに戻されます。

z/OS *IMS* からの非送信請求メッセージの送信

IMS から IBM MQ キューにメッセージを送信するには、ISRT が ALTPCB に対して実行する IMS トランザクションを呼び出す必要があります。

事前経路指定および宛先解決出口を作成して、IMS からの非送信請求メッセージの経路を指定し、OTMA ユーザー・データを作成することによって、メッセージの MQMD が正確に構築されるようにする必要があります。 これらの出口プログラムについては、事前経路指定出口 DFSYPRX0 および宛先解決ユーザー出口を参照してください。

注 : IBM MQ - IMS ブリッジは、受信したメッセージが応答メッセージと非送信請求メッセージのどちらなのかを判断することができません。 いずれの場合であっても、メッセージも同じようにして扱い、そのメッセージとともに受信した OTMA UserData に基づいて、応答の MQMD および MQIIH を構築します。

非送信請求メッセージの場合は、Tpipe を新規に作成できます。 例えば、既存の IMS トランザクションを新規の LTERM (例: PRINT01) に変更したときに、実装システムでは出力を OTMA によって転送する必要がある場合、新規の Tpipe (この場合は PRINT01) が作成されます。 デフォルトの場合、作成されるのは非同期 Tpipe となります。 実装システムでメッセージをリカバリー可能にする必要がある場合は、宛先解決出口の出力フラグを設定してください。 詳細については、 *IMS* カスタマイズの手引き を参照してください。

z/OS メッセージのセグメント化

単一のセグメントまたは複数のセグメントの入力が予期されるとき、IMS トランザクションを定義できます。

発信元の IBM MQ アプリケーションは、MQIIH 構造体に続くユーザー入力を 1 つまたは複数の LLZZ データ・セグメントとして作成する必要があります。 IMS メッセージのすべてのセグメントは、1 回の MQPUT 呼び出しで送られる 1 つの IBM MQ メッセージの中に入っていなければなりません。

1 つの LLZZ データ・セグメントの最大長は、IMS/OTMA によって定義されます (32767 バイト)。 IBM MQ メッセージの全体の長さは、LL バイトの合計に、MQIIH 構造体の長さを加えた値となります。

応答のすべてのセグメントは、1 つの IBM MQ メッセージが入っています。

このほか、MQFMT_IMS_VAR_STRING 形式のメッセージについては、32 KB の制限があります。 ASCII 混合 CCSID メッセージ内のデータが EBCDIC 混合 CCSID メッセージに変換されるとき、SBCS 文字と DBCS 文字との間の遷移が発生するたびに、シフトイン・バイトまたはシフトアウト・バイトが追加されます。 32 KB の制限は、メッセージの最大サイズに適用されます。 つまり、メッセージの LL フィールドが 32 KB を超えることができないため、メッセージはシフトイン文字およびシフトアウト文字を含めて 32 KB を超えることができません。 メッセージを作成するアプリケーションでは、これを考慮に入れる必要があります。

z/OS *IMS* ブリッジとの間のメッセージのデータ変換

データ変換は、メッセージをそのストレージ・クラス用に定義された XCF 情報を持つ宛先キューに書き込むときに、分散キューイング機能 (必要なすべての出口を呼び出すことができる) またはグループ内キュー

イング・エージェント (出口の使用をサポートしない) によって実行されます。 データ変換は、パブリッシュ/サブスクライブによってメッセージがキューに送信された場合には行われません。

必要な出口はすべて、CSQXLIB DD ステートメントで参照されるデータ・セット内の分散キューイング機能から利用できなければなりません。 つまり、任意の IBM MQ プラットフォームから IBM MQ - IMS ブリッジを使用して IMS アプリケーションにメッセージを送信できます。

変換エラーがある場合、そのメッセージは未変換キューに書き込まれます。これは、IBM MQ - IMS ブリッジがそのヘッダー形式を認識できないため、最終的にそのブリッジによってエラーとして処理されます。変換エラーが発生すると、エラー・メッセージが z/OS コンソールに送られます。

一般的なデータ変換の詳細については、988 ページの『データ変換出口の作成』を参照してください。

IBM MQ - IMS ブリッジへのメッセージの送信

変換が正しく行われるようにするため、キュー・マネージャーにメッセージの形式を指示する必要があります。

メッセージが MQIIH 構造体を持っている場合は、MQMD 内の *Format* は組み込み形式 MQFMT_IMS に設定し、MQIIH 内の *Format* はメッセージ・データを記述する形式の名前に設定する必要があります。MQIIH がない場合は、MQMD 内の *Format* を所定の形式名に設定します。

データ (LLZZ 以外) がすべて文字データ (MQCHAR) の場合、形式名 (MQIIH または MQMD 内) として組み込み形式 MQFMT_IMS_VAR_STRING を使用します。 文字データでない場合は、独自の形式を使用します。また、その場合は、その形式にデータ変換出口も必要となります。 この出口では、データ自体に加えて、メッセージ内の LLZZ の変換も処理しなければなりません (ただし、メッセージの開始時に MQIIH を処理する必要はありません)。

アプリケーションで *MFSMapName* を使用する場合、MQFMT_IMS でメッセージを代わりに使用して、MQIIH の MFSMapName フィールドで IMS トランザクションに渡すマップ名を定義することができます。

IBM MQ - IMS ブリッジからのメッセージの受信

IMS に送信している元のメッセージに MQIIH 構造体がある場合は、応答メッセージにも MQIIH 構造体があります。

応答が正しく変換されるようにするには、次のステップに従ってください。

- 元のメッセージに MQIIH 構造体がある場合は、応答メッセージに指定したい形式を元のメッセージの MQIIH *ReplytoFormat* フィールドに指定します。 この値は応答メッセージの MQIIH *Format* フィールドに入れられます。 これは、出力データがすべて LLZZ< 文字データ> の形式の場合、特に役に立ちます。
- 元のメッセージに MQIIH 構造体がない場合は、応答メッセージに指定したい形式を IMS アプリケーションの ISRT 内の MFS MOD 名として IOPCB に指定します。

JMS/Jakarta Messaging および Java アプリケーションの開発

IBM MQ には、3 つの Java 言語インターフェース (IBM MQ classes for Jakarta Messaging、 IBM MQ classes for JMS、および IBM MQ classes for Java) が用意されています。

このタスクについて

IBM MQ classes for Jakarta Messaging

IBM MQ classes for Jakarta Messaging は、 IBM MQ 用の Jakarta Messaging インターフェースをメッセージング・システムとして実装する Jakarta Messaging プロバイダーです。 Jakarta Connectors Architecture は、 Jakarta EE 環境で実行されるアプリケーションを IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。

詳しくは、83 ページの『IBM MQ classes for Jakarta Messaging を使用する理由』および 86 ページの『Java からの IBM MQ へのアクセス-API の選択』を参照してください。

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースをメッセージング・システムとして実装する JMS プロバイダーです。 Java Platform, Enterprise Edition Connector Architecture (JCA) は、Java EE 環境内で実行されているアプリケーションを、IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。

詳しくは、85 ページの『IBM MQ classes for JMS を使用する理由』および86 ページの『Java からの IBM MQ へのアクセス-API の選択』を参照してください。

IBM MQ classes for Java

IBM MQ classes for Java を使用すると、 Java 環境で IBM MQ を使用できます。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

IBM MQ classes for Java は、メッセージ・キュー・インターフェース (MQI)、ネイティブ IBM MQ API をカプセル化し、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用します。一方、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 Oracle および Java Community Process の Java メッセージング・インターフェースをそれぞれ実装します。

詳しくは、348 ページの『IBM MQ classes for Java を使用する理由』および86 ページの『Java からの IBM MQ へのアクセス-API の選択』を参照してください。

注 :

Stabilized IBM では、IBM MQ classes for Java に対してこれ以上拡張機能を提供することはありません。また、IBM MQ 8.0 で出荷されたレベルで機能的に固定化されています。 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、新機能は追加されず、機能拡張の要求も拒否されます。 完全なサポートとは、欠陥が見つかった場合、IBM MQ システム要件の変更によって必要とされる変更と一緒に修正されることを意味します。

IBM MQ classes for Java は IMS ではサポートされません。

IBM MQ classes for Java は WebSphere Liberty ではサポートされません。 IBM MQ Liberty メッセージング・フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、これらを使用してはいけません。 詳しくは、 Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。

IBM MQ classes for JMS/Jakarta Messaging の使用

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

V 9.3.0 JM 3.0 V 9.3.0 IBM MQ 9.3.0 以降、新規アプリケーションの開発で Jakarta Messaging 3.0 がサポートされるようになりました。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで JMS 2.0 API と Jakarta Messaging 3.0 API の両方を使用することはサポートされていません。

注 : Jakarta Messaging 3.0 の場合、 JMS 仕様の制御は Oracle から Java Community Process に移動します。 ただし、 Oracle は「javax」名の制御を保持します。この名前は、 Java Community Process に移動されていない他の Java テクノロジーで使用されます。 そのため、 Jakarta Messaging 3.0 は JMS 2.0 と機能的に同等ですが、命名にはいくつかの違いがあります。

- バージョン 3.0 の正式な名前は、 Java Message Service ではなく Jakarta Messaging です。
- パッケージ名および定数名には、`javax` ではなく `jakarta` が接頭部として付けられます。 例えば、JMS 2.0 では、メッセージング・プロバイダーへの初期接続は `javax.jms.Connection` オブジェクトであり、 Jakarta Messaging 3.0 では `jakarta.jms.Connection` オブジェクトです。

JMS 2.0 javax.jms パッケージは JMS インターフェースを定義し、 JMS プロバイダーはこれらのインターフェースを特定のメッセージング製品用に実装します。 IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースを実装する JMS プロバイダーです。

JM 3.0 jakarta.jms パッケージは Jakarta Messaging インターフェースを定義し、Jakarta Messaging プロバイダーはこれらのインターフェースを特定のメッセージング製品用に実装します。 IBM MQ classes for Jakarta Messaging は、 IBM MQ 用の Jakarta Messaging インターフェースを実装する Jakarta Messaging プロバイダーです。

JMS 仕様および Jakarta Messaging 仕様では、 ConnectionFactory オブジェクトと Destination オブジェクトが管理対象オブジェクトであることが想定されています。 管理者は、中央リポジトリーに管理対象オブジェクトを作成して保守します。 JMS または Jakarta Messaging アプリケーションは、 Java Naming Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。

JMS 2.0 JMS 2.0 の場合、管理者は IBM MQ JMS 管理ツール **JMSAdmin** または IBM MQ Explorer を使用して、中央リポジトリーで管理対象オブジェクトを作成および保守することができます。

JM 3.0 Jakarta Messaging 3.0 の場合、 IBM MQ Explorer を使用して JNDI を管理することはできません。 JNDI 管理は、 **JMSAdmin** の Jakarta Messaging 3.0 バリアント (**JMS30Admin**) によってサポートされます。

JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピック内の JMS に対するこれ以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

IBM MQ classes for JMS は JMS API に対する 2 セットの拡張機能も提供します。 それらの拡張機能は主に、接続ファクトリーおよび宛先を実行時に動的に作成および構成することに重点が置かれていますが、メッセージングと直接的には関係のない機能 (例えば問題判別のための機能) も提供します。

IBM MQ JMS 拡張機能

IBM MQ classes for JMS には、 MQConnectionFactory、MQQueue、および MQTopic オブジェクトなどのオブジェクトで実装された拡張機能が含まれています。 これらのオブジェクトには IBM MQ に固有のプロパティーやメソッドがあります。 オブジェクトは管理対象オブジェクトにすることができます。あるいは、アプリケーションはオブジェクトを実行時に動的に作成することができます。 これらの拡張機能は、 IBM MQ JMS 拡張機能と呼ばれます。

IBM JMS 拡張機能

IBM MQ classes for JMS には、 JMS API に対する拡張機能のより汎用的なセットも用意されています。これらの拡張機能は、メッセージング・システムとしての IBM MQ や、使用されるプログラミング言語としての Java に固有ではありません。 これらの拡張機能は、 IBM JMS 拡張機能と呼ばれ、以下のような幅広い目的があります。

- IBM JMS プロバイダー間でより高いレベルの整合性を提供するため。
- 2 つの IBM メッセージング・システム間のブリッジ・アプリケーションの作成を容易にする。
- ある IBM JMS プロバイダーから別のプロバイダーへのアプリケーションの移植を容易にするため。

拡張機能は、IBM MQ Message Service Client (XMS) for C/C++ および IBM MQ Message Service Client (XMS) for .NET で提供される機能と類似した機能を備えています。

関連概念

IBM MQ Java 言語インターフェース

関連タスク

141 ページの『IBM MQ classes for JMS/Jakarta Messaging アプリケーションの作成』
JMS モデルを簡単に紹介した後、このセクションでは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging アプリケーションの作成方法に関する詳細なガイダンスを提供します。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging を使用する理由

あらゆる既存の Jakarta Messaging スキルを組織内でしようできるようになることや、Jakarta Messaging プロバイダーから、より独立しているアプリケーションと基礎となる IBM MQ 構成など、IBM MQ classes for Jakarta Messaging の使用には多くの利点があります。

IBM MQ classes for Jakarta Messaging を使用する利点の要約

IBM MQ classes for Jakarta Messaging を使用すると、既存の Jakarta Messaging スキルを再使用すること、およびアプリケーションの独立性を提供することができます。

- Jakarta Messaging のスキルを再利用できます。

  IBM MQ classes for Jakarta Messaging は、 IBM MQ 用の Jakarta Messaging インターフェースをメッセージング・システムとして実装する Jakarta Messaging プロバイダーです。 組織が IBM MQ を初めて使用するが、Jakarta Messaging (または JMS) アプリケーション開発スキルを既に持っている場合は、IBM MQ で提供されている他の API の 1 つではなく、使い慣れた Jakarta Messaging API を使用して IBM MQ リソースにアクセスする方が簡単です。

- Jakarta Messaging は、 Jakarta EE の整数部分です。

  Jakarta Messaging は、Jakarta EE プラットフォーム上でのメッセージングに使用するのに適した API です。 Jakarta EE に準拠するすべてのアプリケーション・サーバーには、Jakarta Messaging プロバイダーが組み込まれていなければなりません。 Jakarta Messaging は、アプリケーション・クライアント、サーブレット、JavaServer Page (JSP)、エンタープライズ Java Bean (EJB)、およびメッセージ・ドリブン Bean (MDB) で使用できます。 特に、Jakarta EE アプリケーションは MDB を使用してメッセージを非同期に処理し、すべてのメッセージは Jakarta Messaging メッセージとして MDB に配信されることに注意してください。

- 接続ファクトリーと宛先は、アプリケーションにハードコーディングするのではなく、Jakarta Messaging 管理対象オブジェクトとして中央リポジトリーに保管できます。

  管理者は、中央リポジトリーで Jakarta Messaging 管理対象オブジェクトを作成および保守することができ、IBM MQ classes for Jakarta Messaging アプリケーションは、Java Naming Directory Interface (JNDI) を使用してこれらのオブジェクトを取得できます。Jakarta Messaging 接続ファクトリーおよび宛先は、キュー・マネージャー名、チャネル名、接続オプション、キュー名、およびトピック名など、IBM MQ 固有の情報をカプセル化します。 接続ファクトリーおよび宛先が管理対象オブジェクトとして保管される場合、この情報はアプリケーションにハードコーディングされません。 このため、この調整により、アプリケーションは基盤となる IBM MQ 構成からある程度の独立性を保持することができます。

- Jakarta Messaging は、アプリケーションの移植性を実現する業界標準の API です。

  Jakarta Messaging アプリケーションは、 JNDI を使用して、管理対象オブジェクトとして保管されている接続ファクトリーおよび宛先を取得し、`jakarta.jms` (Jakarta Messaging 3.0) パッケージで定義されているインターフェースのみを使用してメッセージング操作を実行できます。 そうするとアプリケーションは、いずれの Jakarta Messaging プロバイダー (IBM MQ classes for Jakarta Messaging など) からも完全に独立し、アプリケーションを変更しなくても、異なる Jakarta Messaging プロバイダー間で移植することができます。

  特定のアプリケーション環境で JNDI を使用できない場合、 IBM MQ classes for Jakarta Messaging アプリケーションは Jakarta Messaging API の拡張機能を使用して、実行時に接続ファクトリーと宛先を動的に作成して構成することができます。 そうすると、アプリケーションは完全に自己完結型となりますが、IBM MQ classes for Jakarta Messaging に Jakarta Messaging プロバイダーとして結合されます。

- ブリッジ・アプリケーションは、Jakarta Messaging を使用して作成する方が簡単です。

  ブリッジ・アプリケーションとは、メッセージをあるメッセージング・システムから受信し、それを別のメッセージング・システムに送信するアプリケーションです。 プロダクト固有の API およびメッセージ形式を使用してブリッジ・アプリケーションを作成することは、複雑になる可能性があります。 その代わりに、2 つの Jakarta Messaging プロバイダー (各メッセージング・システムに 1 つずつ) を使用して、ブリッジ・アプリケーションを作成できます。 そうすると、アプリケーションは 1 つの API、つまり Jakarta Messaging API のみを使用し、Jakarta Messaging メッセージのみを処理します。

実装可能な環境

Jakarta EE アプリケーション・サーバーと統合するために、Jakarta EE 規格ではリソース・アダプターを提供するメッセージング・プロバイダーが必要です。 Jakarta Connectors Architecture 仕様に従って、IBM MQ は、Jakarta Messaging を使用して、認定された Jakarta EE 環境内でメッセージング機能を提供するリソース・アダプターを提供します。 詳しくは、442 ページの『Liberty と IBM MQ リソース・アダプター』を参照してください。

注 : WebSphere Application Server traditional は現在 Jakarta EE をサポートしていません。

Jakarta EE 環境の外部では、OSGi および JAR ファイルが提供され、IBM MQ classes for Jakarta Messaging のみを簡単に取得できます。 これらの JAR ファイルは、スタンドアロンでも、Maven などのソフトウェア管理フレームワーク内でも、より簡単にデプロイできます。 詳しくは、128 ページの『IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging を個別に入手する』を参照してください。

関連概念

IBM MQ classes for Jakarta Messaging: 概要

86 ページの『Java からの IBM MQ へのアクセス-API の選択』
IBM MQ は、3 つの Java 言語インターフェースを提供します。

JMS 2.0 IBM MQ classes for JMS を使用する理由

あらゆる既存の JMS スキルを組織内でしようできるようになることや、JMS プロバイダーから、より独立しているアプリケーションと基礎となる IBM MQ 構成など、IBM MQ classes for JMS の使用には多くの利点があります。

IBM MQ classes for JMS を使用する利点の要約

IBM MQ classes for JMS を使用すると、既存の JMS スキルを再使用すること、およびアプリケーションの独立性を提供することができます。

V 9.3.0 V 9.3.0

注 : JMS 2.0 は Jakarta Messaging に置き換えられました。 IBM MQ classes for JMS は引き続き JMS 2.0 標準をサポートしますが、 Java メッセージングに対する将来の機能拡張は Jakarta Messaging でのみ行われるため、IBM MQ classes for Jakarta Messaging で行われます。 IBM MQ classes for JMS は、既存の JMS 2.0 アプリケーションを保守および拡張する場合にのみ推奨されます。 IBM MQ classes for Jakarta Messaging は、新しい開発に推奨されるテクノロジーでなければなりません。

- JMS のスキルを再利用できます。

  IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースをメッセージング・システムとして実装する JMS プロバイダーです。 お客様の組織が IBM MQ を初めて使う組織であっても、既に JMS アプリケーション開発スキルを持っている場合は、IBM MQ で提供されているその他の API のいずれかを使うよりも、使い慣れた JMS API を使って、IBM MQ リソースにアクセスするほうがより容易に行えます。

- JMS は、Java Platform, Enterprise Edition (Java EE) のうちの不可欠な部分です。

  JMS は、Java EE プラットフォーム上でのメッセージングに使用するのに適した API です。 Java EE に準拠するすべてのアプリケーション・サーバーには、JMS プロバイダーが組み込まれていなければなりません。 JMS は、アプリケーション・クライアント、サーブレット、JavaServer Page (JSP)、エンタープライズ Java Bean (EJB)、およびメッセージ・ドリブン Bean (MDB) で使用できます。 特に、Java EE アプリケーションは MDB を使用してメッセージを非同期に処理し、すべてのメッセージは JMS メッセージとして MDB に配信されることに注意してください。

- 接続ファクトリーと宛先は、アプリケーションにハードコーディングするのではなく、JMS 管理対象オブジェクトとして中央リポジトリーに保管できます。

  管理者は、中央リポジトリーで JMS 管理対象オブジェクトを作成および保守することができ、IBM MQ classes for JMS アプリケーションは、Java Naming Directory Interface (JNDI) を使用してこれらのオブジェクトを取得できます。JMS 接続ファクトリーおよび宛先は、キュー・マネージャー名、チャネル名、接続オプション、キュー名、およびトピック名など、IBM MQ 固有の情報をカプセル化します。 接続ファクトリーおよび宛先が管理対象オブジェクトとして保管される場合、この情報はアプリケーションにハードコーディングされません。 このため、この調整により、アプリケーションは基盤となる IBM MQ 構成からある程度の独立性を保持することができます。

- JMS は、アプリケーションの移植性を実現する業界標準の API です。

  JMS アプリケーションは、JNDI を使用して、管理対象オブジェクトとして保管されている接続ファクトリーおよび宛先を取得し、`javax.jms` パッケージで定義されているインターフェースのみを使用してメッセージング操作を実行できます。 そうするとアプリケーションは、いずれの JMS プロバイダー (IBM

MQ classes for JMS など) からも完全に独立し、アプリケーションを変更しなくても、異なる JMS プロバイダー間で移植することができます。

JNDI が特定のアプリケーション環境で使用できない場合、IBM MQ classes for JMS アプリケーションは JMS API への拡張機能を使用して、実行時に接続ファクトリーおよび宛先を動的に作成および構成できます。 そうすると、アプリケーションは完全に自己完結型となりますが、IBM MQ classes for JMS に JMS プロバイダーとして結合されます。

- ブリッジ・アプリケーションは、JMS を使用して作成する方が簡単です。

  ブリッジ・アプリケーションとは、メッセージをあるメッセージング・システムから受信し、それを別のメッセージング・システムに送信するアプリケーションです。 プロダクト固有の API およびメッセージ形式を使用してブリッジ・アプリケーションを作成することは、複雑になる可能性があります。 その代わりに、2 つの JMS プロバイダー (各メッセージング・システムに 1 つずつ) を使用して、ブリッジ・アプリケーションを作成できます。 そうすると、アプリケーションは 1 つの API、つまり JMS API のみを使用し、JMS メッセージのみを処理します。

実装可能な環境

Java EE アプリケーション・サーバーと統合するために、Java EE 規格ではリソース・アダプターを提供するメッセージング・プロバイダーが必要です。 Java EE Connector Architecture (JCA) 仕様に従い、IBM MQ は JMS を使用して認証された任意の Java EE 環境にメッセージング機能を提供するリソース・アダプターを提供します。

IBM MQ classes for Java を Java EE の内部で利用することは可能でしたが、この API はこの目的のために作成または最適化されていません。 Java EE 内での IBM MQ classes for Java の考慮事項について詳しくは、349 ページの『Java EE 内での IBM MQ classes for Java アプリケーションの実行』を参照してください。

Java EE 環境の外部では、OSGi および JAR ファイルが提供され、IBM MQ classes for JMS のみを簡単に取得できます。 これらの JAR ファイルは、スタンドアロンでも、Maven などのソフトウェア管理フレームワーク内でも、より簡単にデプロイできます。 詳しくは、128 ページの『IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging を個別に入手する』を参照してください。

関連概念

V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging: 概要

83 ページの『IBM MQ classes for Jakarta Messaging を使用する理由』
あらゆる既存の Jakarta Messaging スキルを組織内でしようできるようになることや、Jakarta Messaging プロバイダーから、より独立しているアプリケーションと基礎となる IBM MQ 構成など、IBM MQ classes for Jakarta Messaging の使用には多くの利点があります。

86 ページの『Java からの IBM MQ へのアクセス-API の選択』
IBM MQ は、3 つの Java 言語インターフェースを提供します。

Java からの IBM MQ へのアクセス-API の選択

IBM MQ は、3 つの Java 言語インターフェースを提供します。

- V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging を使用すると、 Jakarta Messaging 3.0 API を使用して作成されたアプリケーションで IBM MQ をメッセージング・プロバイダーとして使用することができます。

Jakarta Messaging は、 Java アプリケーションにおけるメッセージングの戦略的方向性です。

Jakarta Messaging 3.0 は JMS 2.0 と機能的に同等です。詳しくは、82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』を参照してください。

IBM MQ classes for JMS

IBM MQ classes for JMS を使用すると、 JMS 2.0 API を使用して作成されたアプリケーションで IBM MQ をメッセージング・プロバイダーとして使用することができます。

Jakarta Messaging supersedes JMS として、既存のアプリケーション、または Jakarta Messaging をサポートしない環境 (WebSphere Application Server など) で IBM MQ classes for JMS を使用することをお勧めします。

同じアプリケーションで IBM MQ classes for Jakarta Messaging と IBM MQ classes for JMS の両方を使用することはサポートされていません。

詳しくは、82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』を参照してください。

IBM MQ classes for Java

Stabilized Java アプリケーションが IBM MQ リソースにアクセスするために使用できるもう 1 つの API は IBM MQ classes for Java です。これは、プログラムが IBM MQ をメッセージング・プロバイダーとして使用するための IBM MQ 指向の API を提供します。 ただし、 IBM MQ classes for Java は、 IBM MQ 8.0 で出荷されたレベルで機能的に安定化されています。 詳しくは、348 ページの『IBM MQ classes for Java を使用する理由』 を参照してください。 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、新しいアプリケーションでは IBM MQ classes for Jakarta Messaging を使用する必要があります。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の共通機能

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、IBM MQ の Point-to-Point メッセージング機能とパブリッシュ/サブスクライブ・メッセージング機能の両方へのアクセスを提供します。アプリケーションは JMS の標準のメッセージング・モデルをサポートする JMS メッセージを送信するだけでなく、追加ヘッダーなしでメッセージを送受信できるため、C MQI アプリケーションなど、他の IBM MQ アプリケーションと相互運用することができます。 MQMD および MQ メッセージ・ペイロードの完全な制御が可能です。

メッセージ・ストリーミング、非同期書き込み、レポートのメッセージなど、ほかの IBM MQ 機能も利用できます。

提供された PCF ヘルパー・クラスを使用すると、IBM MQ PCF 管理メッセージを JMS API を介して送受信して、キュー・マネージャーを管理するために使用することができます。

最近 IBM MQ に追加された機能 (非同期コンシュームや自動再接続など) は、IBM MQ classes for Java では使用できませんが、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging で使用できます。

機能拡張の要求

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では使用できない IBM MQ 機能にアクセスする必要がある場合は、アイデアを提案できます。

IBM は、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging の実装で実装が可能かどうか、またはフォローできるベスト・プラクティスがあるかどうかをアドバイスできます。

IBM はオープン・スタンダードの貢献者であるので、追加のメッセージング機能を JCP プロセスの一部として提出することができます。 これらは Jakarta Messaging にのみ適用されます。

関連情報

IBM Ideas Portal へようこそ

JMS Java 仕様のレビュー・プロセス

Using JMS to send PCF messages

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging の前提条件

このトピックでは、IBM MQ classes for Jakarta Messaging を使用するために事前に必要な知識について説明します。 IBM MQ classes for Jakarta Messaging アプリケーションを開発し、実行するには、前提条件としていくつかのソフトウェア・コンポーネントが必要です。

IBM MQ classes for Jakarta Messaging の前提条件については、IBM MQ のシステム要件を参照してください。

IBM MQ classes for Jakarta Messaging アプリケーションを開発するには、Java SE Software Development Kit (SDK) が必要です。 ご使用のオペレーティング・システムでサポートされる JDK について詳しくは、IBM MQ のシステム要件を参照してください。

IBM MQ classes for Jakarta Messaging アプリケーションを実行するには、以下のソフトウェア・コンポーネントが必要です。

- IBM MQ キュー・マネージャー。
- アプリケーションを実行する各システムに対して、Java runtime environment (JRE)。
- IBM i IBM i の場合は、Qshell (オペレーティング・システムのオプション 30)。
- z/OS z/OS の場合は、z/OS UNIX System Services (z/OS UNIX)。

IBM JSSE プロバイダーは、FIPS 認定の暗号プロバイダーを含むため、すぐに使用可能な FIPS 140-2 準拠用にプログラムで構成することができます。 したがって、FIPS 140-2 準拠は、IBM MQ classes for Jakarta Messaging から直接サポートすることができます。

Oracle の JSSE プロバイダーは、FIPS 認定の暗号プロバイダーを含むように構成することができますが、すぐに使用できるようになっておらず、プログラムで構成することもできません。 したがって、この場合、IBM MQ classes for Jakarta Messaging は FIPS 140-2 準拠を直接有効にすることができません。 そうした準拠を手動で有効にできる可能性はありますが、IBM は現時点でこれに関するガイダンスを提供しておりません。

ご使用の Java 仮想マシン (JVM) およびオペレーティング・システム上の TCP/IP 実装で IPv6 アドレスがサポートされている場合は、IBM MQ classes for Jakarta Messaging アプリケーションで Internet Protocol・バージョン 6 (IPv6) アドレスを使用できます。 IBM MQ Jakarta Messaging 管理ツールの **JMS30Admin** は、 IPv6 アドレスも受け入れます。 このツールについて詳しくは、 管理ツールを使用した JMS および Jakarta Messaging オブジェクトの構成を参照してください。

IBM MQ JMS 管理ツールおよび IBM MQ Explorer は、Java Naming Directory Interface (JNDI) を使用してディレクトリー・サービスにアクセスします。このディレクトリー・サービスは管理対象オブジェクトを保管します。 IBM MQ classes for Jakarta Messaging アプリケーションも、JNDI を使用してディレクトリー・サービスから管理対象オブジェクトを取り出すことができます。

注 : Jakarta Messaging 3.0 の場合、 IBM MQ Explorer を使用して JNDI を管理することはできません。 JNDI 管理は、 **JMSAdmin** の Jakarta Messaging 3.0 バリアント (**JMS30Admin**) によってサポートされます。

サービス・プロバイダーは、JNDI 呼び出しをディレクトリー・サービスにマップすることによりディレクトリー・サービスへのアクセスを提供するコードです。 `fscontext.jar` ファイルと `providerutil.jar` ファイルのファイル・システム・サービス・プロバイダーは IBM MQ classes for Jakarta Messaging で提供されます。 ファイル・システム・サービス・プロバイダーは、ローカル・ファイル・システムに基づいてディレクトリー・サービスへのアクセスを提供します。

LDAP サーバーに基づくディレクトリー・サービスを使用する場合は、LDAP サーバーをインストールおよび構成するか、または既存の LDAP サーバーへのアクセスを取得する必要があります。 具体的には、Java オブジェクトを保管するように LDAP サーバーを構成する必要があります。 LDAP サーバーをインストールおよび構成する方法について詳しくは、サーバーに付属する文書を参照してください。

JMS 2.0 IBM MQ classes for JMS の前提条件

このトピックでは、IBM MQ classes for JMS を使用するために事前に必要な知識について説明します。 IBM MQ classes for JMS アプリケーションを開発し、実行するには、前提条件としていくつかのソフトウェア・コンポーネントが必要です。

IBM MQ classes for JMS の前提条件については、 IBM MQ のシステム要件を参照してください。

IBM MQ classes for JMS アプリケーションを開発するには、Java SE Software Development Kit (SDK) が必要です。 ご使用のオペレーティング・システムでサポートされる JDK について詳しくは、IBM MQ のシステム要件を参照してください。

IBM MQ classes for JMS アプリケーションを実行するには、以下のソフトウェア・コンポーネントが必要です。

- IBM MQ キュー・マネージャー。
- アプリケーションを実行する各システムに対して、Java runtime environment (JRE)。
- IBM i IBM i の場合は、Qshell (オペレーティング・システムのオプション 30)。
- z/OS z/OS の場合は、z/OS UNIX System Services (z/OS UNIX)。

IBM JSSE プロバイダーは、FIPS 認定の暗号プロバイダーを含むため、すぐに使用可能な FIPS 140-2 準拠用にプログラムで構成することができます。 したがって、IBM MQ classes for Java と IBM MQ classes for JMS から、FIPS 140-2 準拠を直接サポートできます。

Oracle の JSSE プロバイダーは、FIPS 認定の暗号プロバイダーを含むように構成することができますが、すぐに使用できるようになっておらず、プログラムで構成することもできません。 したがってこの場合、IBM MQ classes for Java と IBM MQ classes for JMS は FIPS 140-2 準拠を直接有効にすることはできません。 そうした準拠を手動で有効にできる可能性はありますが、IBM は現時点でこれに関するガイダンスを提供しておりません。

ご使用の Java 仮想マシン (JVM) およびオペレーティング・システム上の TCP/IP 実装で IPv6 アドレスがサポートされている場合は、 IBM MQ classes for JMS アプリケーションで Internet Protocol ・バージョン 6 (IPv6) アドレスを使用できます。 IBM MQ JMS 管理ツール (管理ツールを使用した JMS オブジェクトの構成を参照) でも IPv6 アドレスを使用できます。

IBM MQ JMS 管理ツールおよび IBM MQ Explorer は、Java Naming Directory Interface (JNDI) を使用してディレクトリー・サービスにアクセスします。このディレクトリー・サービスは管理対象オブジェクトを保管します。 IBM MQ classes for JMS アプリケーションも、JNDI を使用してディレクトリー・サービスから管理対象オブジェクトを取り出すことができます。 サービス・プロバイダーは、JNDI 呼び出しをディレクトリー・サービスにマップすることによりディレクトリー・サービスへのアクセスを提供するコードです。 ファイル `fscontext.jar` および `providerutil.jar` 内のファイル・システム・サービス・プロバイダーは、IBM MQ classes for JMS と一緒に提供されます。 ファイル・システム・サービス・プロバイダーは、ローカル・ファイル・システムに基づいてディレクトリー・サービスへのアクセスを提供します。

LDAP サーバーに基づくディレクトリー・サービスを使用する場合は、LDAP サーバーをインストールおよび構成するか、または既存の LDAP サーバーへのアクセスを取得する必要があります。 具体的には、Java オブジェクトを保管するように LDAP サーバーを構成する必要があります。 LDAP サーバーをインストールおよび構成する方法について詳しくは、サーバーに付属する文書を参照してください。

IBM MQ classes for JMS/Jakarta Messaging のインストールと構成

このセクションでは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging のインストール時に作成されるディレクトリーおよびファイルについて説明し、インストール後に IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging を構成する方法について説明します。

関連概念

436 ページの『IBM MQ リソース・アダプターの使用』

リソース・アダプターによって、アプリケーション・サーバーで実行されているアプリケーションは IBM MQ リソースにアクセスできます。 インバウンド通信とアウトバウンド通信をサポートします。

IBM MQ classes for JMS でインストールされる内容

IBM MQ classes for JMS のインストール時にはいくつかのファイルとディレクトリーが作成されます。 Windows では、インストールの際に、自動的に環境変数を設定することによっていくつかの構成が実行されます。 他のプラットフォームやある特定の Windows 環境では、IBM MQ classes for JMS アプリケーションを実行するために、まず環境変数を設定する必要があります。

ほとんどのオペレーティング・システムでは、IBM MQ classes for JMS は IBM MQ のインストール時にオプションのコンポーネントとしてインストールされます。

IBM MQ のインストールの詳細については、以下を参照してください。

Multi IBM MQ のインストール

z/OS IBM MQ for z/OS のインストール

重要 : 92 ページの『IBM MQ classes for JMS/Jakarta Messaging 再配置可能 JAR ファイル』で説明する再配置可能 JAR ファイルを除いて、IBM MQ classes for JMS JAR ファイルまたはネイティブ・ライブラリーを他のマシンにコピーすることや、IBM MQ classes for JMS がインストールされているマシン上の異なるロケーションにコピーすることは、サポートされていません。

インストール・ディレクトリー

90 ページの表 5 に、各プラットフォームでの IBM MQ classes for JMS ファイルのインストール先を示します。

表 *5. IBM MQ classes for JMS* のインストール・ディレクトリー

プラットフォーム	ディレクトリー
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/java

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

インストール・ディレクトリーの内容は、以下のとおりです。

- *MQ_INSTALLATION_PATH*\java\lib ディレクトリーにある、再配置可能な JAR ファイルを含む IBM MQ classes for JMS JAR ファイル。
- IBM MQ ネイティブ・ライブラリー。これは Java Native Interface を使用するアプリケーションによって使用されます。

  32 ビットのネイティブ・ライブラリーは、*MQ_INSTALLATION_PATH*\java\lib ディレクトリーにインストールされ、64 ビットのネイティブ・ライブラリーは *MQ_INSTALLATION_PATH*\java\lib64 ディレクトリーで見つかります。

  IBM MQ ネイティブ・ライブラリーの詳細については、97 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照してください。
- 追加スクリプト (124 ページの『IBM MQ classes for JMS/Jakarta Messaging で提供されるスクリプト』を参照)。 これらのスクリプトは、*MQ_INSTALLATION_PATH*\java\bin ディレクトリーにあります。
- IBM MQ classes for JMS API の仕様。 API の仕様を含む HTML ページの生成には Javadoc ツールが使用されています。

  HTML ページは、*MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses ディレクトリーにあります。

  – ALW AIX, Linux, and Windows では、このサブディレクトリーに個別の HTML ページが含まれます。

  - IBM i では、HTML ページは `wmqjms_javadoc.jar` というファイル内にあります。
  - z/OS では、HTML ページは `wmqjms_javadoc.jar` というファイル内にあります。
- OGSi のサポート。 OSGi バンドルは、`java\lib\OSGi` ディレクトリーにインストールされ、125 ページの『IBM MQ classes for JMS での OSGi のサポート』 で説明されています。
- IBM MQ リソース・アダプター。これは、任意の Java Platform, Enterprise Edition 7 (Java EE 7) または Jakarta EE 準拠のアプリケーション・サーバーにデプロイできます。

  IBM MQ リソース・アダプターは *MQ_INSTALLATION_PATH*`\java\lib\jca` ディレクトリーにあります。詳しくは、436 ページの『IBM MQ リソース・アダプターの使用』を参照してください。
- Windows では、デバッグに使用できるシンボルが *MQ_INSTALLATION_PATH*`\java\lib\symbols` ディレクトリーにインストールされています。

インストール・ディレクトリーには、 他の IBM MQ コンポーネントに属するファイルも含まれています。

サンプル・アプリケーション

IBM MQ classes for JMS にはいくつかのサンプル・アプリケーションが提供されています。91 ページの表 6 に、各プラットフォームでのサンプル・アプリケーションのインストール先を示します。

IBM MQ classes for Jakarta Messaging の場合、新しいサンプルを準備しています。

表 *6. IBM MQ classes for JMS* のサンプル・ディレクトリー

プラットフォーム	ディレクトリー
AIX and Linux	*MQ_INSTALLATION_PATH*`/samp/jms`
Windows	*MQ_INSTALLATION_PATH*`\tools\jms`
IBM i	`/QIBM/ProdData/mqm/java/samples/jms`
z/OS	*MQ_INSTALLATION_PATH*`/java/samples/jms`

この表では、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

インストールの後、アプリケーションをコンパイルおよび実行するためには、いくつかの構成タスクを実行しなければならない場合があります。

94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』 では、サンプル IBM MQ classes for JMS アプリケーションを実行するために必要なクラスパスについて説明します。 また、このトピックでは、特殊な状況において参照する必要がある追加の JAR ファイルや、IBM MQ classes for JMS で提供されているスクリプトを実行するために設定する必要のある環境変数についても説明しています。

アプリケーションのトレースやロギングなどのプロパティーを制御するためには、構成プロパティー・ファイルを用意する必要があります。 IBM MQ classes for JMS 構成プロパティー・ファイルについては、99 ページの『IBM MQ classes for JMS/Jakarta Messaging 構成ファイル』で説明しています。

関連概念

リソース・アダプターをデプロイする際の問題

関連タスク

121 ページの『IBM MQ classes for JMS サンプル・アプリケーションの使用』

IBM MQ classes for JMS サンプル・アプリケーションは、JMS API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

IBM MQ classes for JMS/Jakarta Messaging 再配置可能 *JAR* ファイル
再配置可能 JAR ファイルは、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を実行する必要があるシステムに移動できます。

重要 :

- 再配置可能 JAR ファイルに記述されている再配置可能 JAR ファイルを除き、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging がインストールされているマシン上の他のマシンまたは別の場所に IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging JAR ファイルまたはネイティブ・ライブラリーをコピーすることはサポートされていません。
- WebSphere Application Server や WebSphere Liberty などの Java EE アプリケーション・サーバーにデプロイされたアプリケーション内には、再配置可能 JAR ファイルを含めないでください。 これらの環境では、代わりに IBM MQ リソース・アダプターをデプロイして使用する必要があります。 WebSphere Application Server には、IBM MQ リソース・アダプターが組み込まれているため、この環境に手動でデプロイする必要はありません。
- クラス・ローダーの競合を回避するため、同じ Java ランタイム内の複数のアプリケーション内で再配置可能 JAR ファイルをバンドルすることは推奨されません。 このシナリオでは、 IBM MQ 再配置可能 JAR ファイルを Java ランタイムのクラスパスで使用できるようにします。
- 再配置可能 JAR ファイルをアプリケーション内にバンドルする場合は、再配置可能 JAR ファイルに説明されているとおりに、すべての前提条件の JAR ファイルを必ず組み込むようにしてください。 また、バンドルされた JAR ファイルをアプリケーション保守の一部として更新するための適切な手順があることを確認して、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging がまだ最新であり、既知の問題が修正されていることを確認する必要があります。

再配置可能 JAR ファイル

企業内では、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を実行する必要があるシステムに以下のファイルを移動できます。

- V 9.3.0 V 9.3.0 bcpkix-jdk15to18.jar 93 ページの『4』
- V 9.3.5 bcpkix-jdk18on.jar 93 ページの『3』
- V 9.3.0 V 9.3.0 bcprov-jdk15to18.jar 93 ページの『4』
- V 9.3.5 bcprov-jdk18on.jar 93 ページの『3』
- V 9.3.0 V 9.3.0 bcutil-jdk15to18.jar 93 ページの『4』
- V 9.3.5 bcutil-jdk18on.jar 93 ページの『3』
- JMS 2.0 com.ibm.mq.allclient.jar 93 ページの『1』
- JM 3.0 V 9.3.0 V 9.3.0 com.ibm.mq.jakarta.client.jar 93 ページの『2』
- V 9.3.3 Removed com.ibm.mq.traceControl.jar
- fscontext.jar
- V 9.3.3 jackson-annotations.jar
- V 9.3.3 jackson-core.jar
- V 9.3.3 jackson-databind.jar
- jakarta.jms-api.jar
- jms.jar
- org.json.jar
- providerutil.jar

注 :

1. JMS 2.0 と JMS 1.1
2. Jakarta Messaging 3.0
3. IBM MQ 9.3.5 からの Continuous Delivery
4. Long Term Support および Continuous Delivery (IBM MQ 9.3.5 より前)

JMS JAR ファイル

V 9.3.0 V 9.3.0 `jms.jar` には、JMS 1.1 インターフェースと JMS 2.0 インターフェースが含まれています。これらのインターフェースの名前は `javax.jms.*`です。

JM 3.0 V 9.3.0 V 9.3.0 `jakarta.jms-api.jar` には、 Jakarta Messaging 3.0 インターフェースが含まれています。これらのインターフェースの名前は `jakarta.jms.*`です。

fscontext.jar および providerutil.jar

`fscontext.jar` ファイルおよび `providerutil.jar` ファイルは、アプリケーションがファイル・システム・コンテキストを使用して JNDI 検索を実行する場合に必要です。

Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイル

Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイルは必須です。 詳しくは、「AMS での非 IBM JREs のサポート」を参照してください。

V 9.3.5 IBM MQ 9.3.5 からの Continuous Delivery の場合、以下の JAR ファイルが必要です。

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS IBM MQ 9.3.5 より前の Long Term Support および Continuous Delivery の場合、以下の JAR ファイルが必要です。

- `bcpkix-jdk15to18.jar`
- `bcprov-jdk15to18.jar`
- `bcutil-jdk15to18.jar`

org.json.jar

IBM MQ classes for JMS アプリケーションが JSON 形式で CCDT を使用する場合、`org.json.jar` ファイルが必要です。

com.ibm.mq.allclient.jar および com.ibm.mq.jakarta.client.jar

ファイル `com.ibm.mq.allclient.jar` および `com.ibm.mq.jakarta.client.jar` には、 IBM MQ classes for JMS、IBM MQ classes for Jakarta Messaging、IBM MQ classes for Java、および PCF と Headers クラスが含まれています。 これらの JAR ファイルを新しい場所に移動する場合は、新しい IBM MQ フィックスパックでこの新しい場所を維持するための手順を必ず実行してください。 また、暫定修正を入手する場合は、ファイルの使用が IBM サポートに認識されていることを確認してください。

ファイル `com.ibm.mq.allclient.jar` および `com.ibm.mq.jakarta.client.jar` のバージョンを判別するには、次のコマンドを使用します。

JM 3.0 V 9.3.0 V 9.3.0

```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
java -jar com.ibm.mq.allclient.jar
```

次の例は、このコマンドの出力例を示しています。

```
C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar、 jackson-core.jar 、および jackson-databind.jar

V 9.3.3

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションがキュー・マネージャーへのセキュアな TLS 接続を作成する場合は、3 つの Jackson JAR ファイルが必要です。

IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定
IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションをコンパイルして実行するには、 **CLASSPATH** 環境変数の設定に IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging Java アーカイブ (JAR) ファイルが含まれている必要があります。 要件によっては、他にも JAR ファイルをクラス・パスに追加しなければならない場合もあります。 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging で提供されるスクリプトを実行するには、他の環境変数を設定する必要があります。

始める前に

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

重要 : Java オプション *-Xbootclasspath* を IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を組み込むように設定することはサポートされていません。

このタスクについて

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションをコンパイルして実行するには、以下の表に示すように、ご使用のプラットフォームおよび Java メッセージング・バージョンの **CLASSPATH** 設定を使用します。 別の方法として、環境変数を使用する代わりに、**java** コマンドでクラス・パスを指定することもできます。

JMS 2.0 IBM MQ classes for JMS の場合、この設定にはサンプル・ディレクトリーが含まれているため、 IBM MQ classes for JMS サンプル・アプリケーションをコンパイルして実行できます。

JM 3.0 IBM MQ classes for Jakarta Messaging の場合、新しいサンプルを準備しています。

JM 3.0

表 7. *IBM MQ classes for Jakarta Messaging* アプリケーションをコンパイルして実行するための *Jakarta Messaging 3.0* の ***CLASSPATH*** 設定

プラットフォーム	**CLASSPATH 設定**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

表 8. *IBM MQ classes for JMS* アプリケーション (サンプル・アプリケーションを含む) をコンパイルして実行するための *JMS 2.0* の ***CLASSPATH*** 設定

プラットフォーム	**CLASSPATH 設定**
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*¥tools¥jms¥samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

これらの表で、 *MQ_INSTALLATION_PATH* は、 IBM MQ がインストールされている上位ディレクトリーを表します。

JAR ファイル `com.ibm.mq.jakarta.client.jar` または `com.ibm.mq.allclient.jar` のマニフェストには、 IBM MQ classes for JMS アプリケーションが必要とする他のほとんどの JAR ファイルへの参照が含まれているため、これらの JAR ファイルをクラスパスに追加する必要はありません。 これらの JAR ファイルには、Java Naming Directory Interface (JNDI) を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すアプリケーション、および Java Transaction API (JTA) を使用するアプリケーションが必要とする JAR ファイルが含まれています。

ただし、以下のような場合には、追加の JAR ファイルをクラス・パスに組み込む必要があります。

- `com.ibm.mq.exits` パッケージで定義されているチャネル出口インターフェースではなく、`com.ibm.mq` パッケージで定義されたチャネル出口インターフェースを実装するチャネル出口クラスを使用している場合は、IBM MQ classes for Java JAR ファイル (`com.ibm.mq.jar`) をクラスパスに追加する必要があります。
- アプリケーションが JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出す場合は、以下の JAR ファイルもクラス・パスに追加する必要があります。
  - `fscontext.jar`
  - `providerutil.jar`
- ご使用のアプリケーションが JTA を使用する場合は、`jta.jar` をクラスパスに追加する必要もあります。

注 : これらの追加 JAR ファイルはアプリケーションのコンパイルでのみ必要になり、実行には必要ありません。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging で提供されるスクリプトは、以下の環境変数を使用します。

MQ_JAVA_DATA_PATH
この環境変数は、ログおよびトレース出力のディレクトリーを指定します。

MQ_JAVA_INSTALL_PATH
この環境変数は、IBM MQ classes for JMS のインストール先ディレクトリーを指定します。

MQ_JAVA_LIB_PATH
この環境変数は、前の表に示すように、 IBM MQ classes for JMS ライブラリーが保管されるディレクトリーを指定します。

手順

- Windows

  Windows では、 IBM MQ をインストールした後、コマンド **setmqenv** を実行します。

  このコマンドを最初に実行しないと、**dspmqver** コマンドの発行時に以下のエラー・メッセージが表示される場合があります。

  ```
  AMQ8351: IBM MQ Java environment has not been configured
  correctly, or the IBM MQ JRE feature has not been installed.
  ```

  注 : このメッセージは、 IBM MQ Java runtime environment (JRE) をインストールしなかった場合に予期されるものです (追加の Windows 機能の前提条件の検査を参照してください)。

- Linux AIX

  AIX and Linux システムでは、環境変数を自分で設定します。

  JMS 2.0 JMS 2.0 の場合は、以下のいずれかのスクリプトを使用して環境変数を設定します。

  - 32 ビット JVM を使用している場合は、スクリプト `setjmsenv` を使用します。
  - AIX または Linux システムで 64 ビット JVM を使用している場合は、スクリプト `setjmsenv64` を使用します。

  JM 3.0 Jakarta Messaging 3.0 の場合は、以下のいずれかのスクリプトを使用して環境変数を設定します。

  - 32 ビット JVM を使用している場合は、スクリプト `setjms30env` を使用します。
  - 64 ビット JVM を使用している場合は、スクリプト `setjms30env64` を使用します。

  これらのスクリプトは、*MQ_INSTALLATION_PATH*`/java/bin` ディレクトリーにあります。ここで *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

これらのスクリプトはさまざまな方法で使用できます。 このスクリプトは、表に示されているように必要な環境変数を設定するための基礎として使用することも、テキスト・エディターを使用して .profile に追加することもできます。 一般的ではない設定を使用している場合は、必要に応じてスクリプトの内容を編集してください。 または、JMS 始動スクリプトが実行されるセッションごとにこのスクリプトを実行することもできます。 このオプションを選択した場合は、JMS 検査プロセス中に、開始するすべてのシェル・ウィンドウでスクリプトを実行する必要があります。

- JMS 2.0 JMS 2.0 の場合は、 . ./setjmsenv または ../setjmsenv64 と入力します。
- JM 3.0 Jakarta Messaging 3.0 の場合は、 . ./setjms30env または . ./setjms30env64 と入力します。

IBM i IBM i では、環境変数 **QIBM_MULTI_THREADED** を Y に設定する必要があります。 それにより、単一スレッド・アプリケーションを実行する場合と同じようにマルチスレッド・アプリケーションを実行できます。 詳しくは、 Setting up IBM MQ with Java and JMS を参照してください。

関連タスク

121 ページの『IBM MQ classes for JMS サンプル・アプリケーションの使用』
IBM MQ classes for JMS サンプル・アプリケーションは、JMS API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

関連資料

124 ページの『IBM MQ classes for JMS/Jakarta Messaging で提供されるスクリプト』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の使用時に実行する必要がある一般的なタスクを支援するために、いくつかのスクリプトが用意されています。

Java Native Interface (JNI) ライブラリーの構成
バインディング・トランスポートまたはクライアント・トランスポートのいずれかを使用してキュー・マネージャーに接続する IBM MQ classes for JMS アプリケーションが、Java 以外の言語で作成されたチャネル出口プログラムを使用する場合、Java Native Interface (JNI) ライブラリーにアクセスできる環境で実行する必要があります。

始める前に

WebSphere Application Server 環境の使用についての詳細は、「ネイティブ・ライブラリー情報を使用した IBM MQ メッセージング・プロバイダーの構成」を参照してください。

このタスクについて

この環境をセットアップするには、IBM MQ classes for JMS アプリケーションを開始する前に Java Virtual Machine (JVM) が mqjbnd ライブラリーをロードできるように、環境のライブラリー・パスを構成する必要があります。

IBM MQ は、以下の 2 つの Java Native Interface (JNI) ライブラリーを提供しています。

mqjbnd
このライブラリーは、バインディング・トランスポートを使用してキュー・マネージャーに接続するアプリケーションが使用します。 このライブラリーは、IBM MQ classes for JMS とキュー・マネージャーの間のインターフェースを提供します。 IBM MQ 9.3 と一緒にインストールされる mqjbnd ライブラリーは、IBM MQ 9.3 以前の任意のキュー・マネージャーに接続するために使用できます。

mqjexitstub02
mqjexitstub02 ライブラリーは、アプリケーションがクライアント・トランスポートを使用してキュー・マネージャーに接続し、Java 以外の言語で作成されたチャネル出口プログラムを使用するときに、IBM MQ classes for JMS によってロードされます。

特定のプラットフォームでは、IBM MQ は、これらの JNI ライブラリーの 32 ビット・バージョンと 64 ビット・バージョンをインストールします。 各プラットフォームでのライブラリーの場所を、表 1 に記載します。

表 9. 各プラットフォームでの *IBM MQ classes for JMS* ライブラリーの場所	
プラットフォーム	**IBM MQ classes for JMS ライブラリーが含まれるディレクトリー**
AIX Linux (POWER、x86-64 および zSeries s390x プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Windows	*MQ_INSTALLATION_PATH*¥java¥lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*¥java¥lib64 (64 ビット・ライブラリー)
z/OS	*MQ_INSTALLATION_PATH*/java/lib (31 ビットおよび 64 ビット・ライブラリー)

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

注 : z/OS では、31 ビットまたは 64 ビットのいずれかの Java Virtual Machine (JVM) を使用できます。 使用する JNI ライブラリーを指定する必要はありません。どの JNI ライブラリーをロードするかは、IBM MQ classes for JMS によって自動的に判別されます。

手順

1. JVM の **java.library.path** プロパティーを次の 2 つの方法のうちいずれかを実行して構成します。
   - 次の例に示されているように JVM 引数を指定します。

     ```
     -Djava.library.path=path_to_library_directory
     ```

     Linux 例えば、64 ビット JVM (Linux) のデフォルト場所におけるインストールの場合、次のように指定します。

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - シェルの環境を構成することにより、JVM は独自の `java.library.path` を設定します。 このパスはプラットフォーム、および IBM MQ のインストール場所によって異なります。 例えば、64 ビットの JVM で、デフォルトの IBM MQ インストール場所の場合、次の設定を使用できます。

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   環境が適切に構成されていない場合に表示される例外スタックの例を次に示します。

   ```
   Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
   AMQ8598: Failed to load the WebSphere MQ native JNI library: 'mqjbnd'.
       at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
       at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
       at java.security.AccessController.doPrivileged(AccessController.java:400)
       at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
       at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
   ```

```
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 more
Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 more
```

2. 32 ビットまたは 64 ビットの環境のセットアップ後、次のコマンドを使用して IBM MQ classes for JMS アプリケーションを開始します。

   ```
   java application-name
   ```

   ここで、 *application-name* は実行する IBM MQ classes for JMS アプリケーション名です。

   以下の場合に、IBM MQ によって IBM MQ classes for JMS 理由コード 2495 (MQRC_MODULE_NOT_FOUND) を含む例外がスローされます。

   - IBM MQ classes for JMS アプリケーションが 32 ビットの Java runtime environment で実行されており、IBM MQ classes for JMS 用に 64 ビット環境がセットアップされている場合。32 ビットの Java runtime environment では 64 ビットの Java ネイティブ・ライブラリーをロードできません。
   - IBM MQ classes for JMS アプリケーションが 64 ビットの Java runtime environment で実行されており、IBM MQ classes for JMS 用に 32 ビット環境がセットアップされている場合。64 ビットの Java runtime environment では 32 ビットの Java ネイティブ・ライブラリーをロードできません。

IBM MQ classes for JMS/Jakarta Messaging 構成ファイル

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging 構成ファイルは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の構成に使用されるプロパティーを指定します。

注 : 構成ファイル内に定義するプロパティーは、JVM システム・プロパティーとしても設定できます。 プロパティーが構成ファイルとシステム・プロパティーの両方で設定されている場合は、システム・プロパティーのほうが優先されます。 したがって、必要な場合は、**java** コマンドでシステム・プロパティーとして指定することで、構成ファイル内のプロパティーを指定変更できます。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルのフォーマットは、標準の Java プロパティー・ファイルのフォーマットです。 `jms.config` という名前のサンプル構成ファイルが、 IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーに用意されています。 このファイル文書には、サポートされているすべてのプロパティーとそれらのデフォルト値が記録されています。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルの名前と場所を選択できます。 アプリケーションを開始するときに、以下のフォーマットで **java** コマンドを使用してください。

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

コマンドで、*config_file_url* は、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルの名前と場所を指定する Uniform Resource Locator (URL) です。 次のタイプの URL (http、file、ftp、および jar) がサポートされています。

次に、**java** コマンドの例を示します。

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

このコマンドは、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルを、ローカル Windows システム上のファイル `D:\mydir\mjms.config` として識別します。

アプリケーションが開始されると、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging は構成ファイルの内容を読み取り、指定されたプロパティーを内部プロパティー・ストアに保管します。 **java** コマンドが構成ファイルを識別しない場合、または構成ファイルが見つからない場合、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging はすべてのプロパティーのデフォルト値を使用します。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルは、アプリケーションとキュー・マネージャーまたはブローカーとの間のサポートされる任意のトランスポートで使用できます。

IBM MQ MQI client 構成ファイルで指定されたプロパティーの指定変更

IBM MQ MQI client 構成ファイルは、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging の構成に使用されるプロパティーを指定することもできます。 ただし、IBM MQ MQI client 構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

必要に応じて、 IBM MQ MQI client 構成ファイル内の属性を IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイル内のプロパティーとして指定することにより、その属性をオーバーライドできます。 IBM MQ MQI client 構成ファイル内の属性をオーバーライドするには、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイル内で以下の形式のエントリーを使用します。

```
com.ibm.mq.cfg. stanza. propName = propValue
```

エントリー中の変数には、以下の意味があります。

stanza
属性を含んでいる IBM MQ MQI client 構成ファイル中のスタンザの名前

propName
IBM MQ MQI client 構成ファイルで指定されている属性の名前

propValue
IBM MQ MQI client 構成ファイルで指定されている属性の値を指定変更するプロパティーの値

あるいは、 **java** コマンドでシステム・プロパティーとしてプロパティーを指定することにより、 IBM MQ MQI client 構成ファイル内の属性をオーバーライドすることができます。 プロパティーをシステム・プロパティーとして指定するには、前述のフォーマットを使用してください。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging に関連するのは、 IBM MQ MQI client 構成ファイル内の以下の属性のみです。 他の属性を指定または指定変更しても、効果はありません。特に、 クライアント構成ファイルの CHANNELS スタンザ の `ChannelDefinitionFile` および `ChannelDefinitionDirectory` は使用されないことに注意してください。 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging で CCDT を使用する方法について詳しくは、 285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』 を参照してください。

表 *10.* クライアント構成ファイルのどのスタンザにどの属性が含まれているか

スタンザ	属性
クライアント構成ファイルの CHANNELS スタンザ	Put1DefaultAlwaysSync
クライアント構成ファイルの CHANNELS スタンザ	DefRecon
クライアント構成ファイルの CHANNELS スタンザ	ReconDelay

表 10. クライアント構成ファイルのどのスタンザにどの属性が含まれているか (続き)	
スタンザ	**属性**
クライアント構成ファイルの CHANNELS スタンザ	PasswordProtection
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの JMQI スタンザ	useMQCSPauthentication
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

IBM MQ MQI client 構成について詳しくは、IBM MQ MQI client 構成ファイル (`mqclient.ini`) を参照してください。

Java 標準環境トレースを使用した *JMS* トレースの構成
Java 標準環境トレース設定スタンザを使用して、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging トレース機能を構成します。

`com.ibm.msg.client.commonservices.trace.outputName = `*traceOutputName*
`traceOutputName` はトレース出力の送信先のディレクトリーおよびファイル名です。

デフォルトでは、トレース情報はアプリケーションの現行作業ディレクトリー内のトレース・ファイルに書き込まれます。 トレース・ファイルの名前は、アプリケーションが実行されている環境によって異なります。

- JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、アプリケーションが再配置可能 JAR ファイル `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) から IBM MQ classes for Jakarta Messaging をロードした場合、または再配置可能 JAR ファイル `com.ibm.mq.allclient.jar` (JMS 2.0) から IBM MQ classes for JMS をロードした場合、トレースは `mqjavaclient_%PID%.cl%u.trc` というファイルに書き込まれます。
- IBM MQ 9.1.5 および IBM MQ 9.1.0 Fix Pack 5 から:
  - アプリケーションが IBM MQ classes for JMS を再配置可能 JAR ファイル `com.ibm.mq.allclient.jar` からロードした場合、トレースは `mqjavaclient_%PID%.cl%u.trc` というファイルに書き込まれます。
  - アプリケーションが IBM MQ classes for JMS を JAR ファイル `com.ibm.mqjms.jar` からロードした場合、トレースは `mqjava_%PID%.cl%u.trc` というファイルに書き込まれます。
- IBM MQ 9.0.0 Fix Pack 2 以降:

  - アプリケーションが IBM MQ classes for JMS を再配置可能 JAR ファイル com.ibm.mq.allclient.jar からロードした場合、トレースは mqjavaclient_*%PID%*.trc というファイルに書き込まれます。
  - アプリケーションが IBM MQ classes for JMS を JAR ファイル com.ibm.mqjms.jar からロードした場合、トレースは mqjava_*%PID%*.trc というファイルに書き込まれます。
- IBM MQ classes for JMS for IBM MQ 9.0.0 Fix Pack 1 以前では、トレースは mqjms_*%PID%*.trc という名前のファイルに書き込まれます。

ここで、*%PID%* はトレースされるアプリケーションのプロセス ID です。*%u* は、異なる Java クラス・ローダーの下でトレースを実行するスレッドの間でファイルを区別するための固有の番号です。

プロセス ID が使用できない場合は、乱数が生成され、接頭部に f の文字が付けられます。 指定するファイル名にプロセス ID を組み込むには、ストリング %PID% を使用します。

代替ディレクトリーを指定する場合、そのディレクトリーが存在していなければならず、また、そのディレクトリーへの書き込み権限が必要です。 書き込み権限がないと、トレース出力は System.err に書き込まれます。

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList は、トレースされるパッケージおよびクラスのリスト、または特殊値 ALL または NONE です。

パッケージまたはクラス名はセミコロン ; で区切ります。 *includeList* はデフォルトで ALL に設定され、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 内のすべてのパッケージおよびクラスをトレースします。

注 : パッケージを含めた後、そのパッケージのサブパッケージを除外することができます。 例えば、パッケージ a.b は含め、パッケージ a.b.x は除外する場合、a.b.y および a.b.z 内のものはすべてトレースに含まれますが、a.b.x 内のものと a.b.x.1 内のものは、どちらも除外されます。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList は、トレースされないパッケージおよびクラスのリスト、または特殊値 ALL または NONE です。

パッケージまたはクラス名はセミコロン ; で区切ります。 *excludeList* のデフォルトは NONE であるため、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging のパッケージおよびクラスはトレース対象から除外されません。

注 : パッケージを除外した後、そのパッケージのサブパッケージを含めることができます。 例えば、パッケージ a.b は除外し、パッケージ a.b.x は含める場合、a.b.x および a.b.x.1 内のものはすべてトレースに含まれますが、a.b.y 内のものと a.b.z 内のものは、どちらも除外されます。

両方 (包含と除外) を指定した場合、同じレベルのパッケージまたはクラスはすべて含められます。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes は、任意のバイト配列からトレースされる最大バイト数です。

maxArrayBytes が正整数に設定されると、トレース・ファイルに書き出されるバイト配列内のバイト数が制限されます。 *maxArrayBytes* の書き出し後に、バイト配列が切り捨てられます。 *maxArrayBytes* を設定すると、結果として生成されるトレース・ファイルのサイズが削減され、トレースがアプリケーションのパフォーマンスに与える影響が低減されます。

このプロパティーの値 0 は、どのバイト配列の内容もトレース・ファイルに送信されないことを意味します。

デフォルト値は -1 です。これは、トレース・ファイルに送信されるバイト配列内のバイト数の制限を除去します。

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes は、トレース出力ファイルに書き込まれる最大バイト数です。

maxTraceBytes は *traceCycles* と連携して機能します。 書き込まれたトレースのバイト数が制限に近い場合、ファイルが閉じられ、新しいトレース出力ファイルが開始されます。

値 0 は、トレース出力ファイルの長さがゼロであることを意味します。 デフォルト値は -1 です。これは、トレース出力ファイルに書き込まれるデータの量が無制限であることを意味します。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles は、循環するトレース出力ファイルの数です。

現行のトレース出力ファイルが *maxTraceBytes* で指定された制限に達すると、ファイルが閉じます。以降のトレース出力は、順序で次にあるトレース出力ファイルに書き込まれます。 各トレース出力ファイルは、ファイル名に付加される数値の接尾部によって区別されます。 現行つまり最新のトレース出力ファイルは mqjms.trc.0 で、次に新しいトレース出力ファイルは mqjms.trc.1 です。 それより古いトレース・ファイルは、制限に達するまで、同じ番号付けパターンに従います。

traceCycles のデフォルト値は 1 です。 *traceCycles* が 1 であれば、現在のトレース出力ファイルが最大サイズに達すると、そのファイルは閉じられて、削除されます。 同じ名前の新しいトレース出力ファイルが開始されます。 したがって、トレース出力ファイルは一度に 1 つだけ存在することになります。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters は、メソッド・パラメーターおよび戻り値をトレースに含めるかどうかを制御します。

traceParameters はデフォルトで TRUE に設定されます。 *traceParameters* が FALSE に設定されると、メソッド・シグニチャーのみがトレースされます。

com.ibm.msg.client.commonservices.trace.startup = *startup*
リソースが割り振られる IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の初期化フェーズがあります。 主なトレース機能の初期化が、このリソース割り振りフェーズで行われます。

startup が TRUE に設定されると、開始トレースが使用されます。 トレース情報が即時に生成され、すべてのコンポーネント (トレース機能自体を含む) の設定が含まれます。 開始トレース情報は、構成の問題を診断するために使用できます。 開始トレース情報は常に System.err に書き込まれます。

startup はデフォルトで FALSE に設定されます。

startup は、初期化が完了する前にチェックされます。 このため、プロパティーを指定する際には必ず、コマンド・ラインで Java システム・プロパティーとしてのみ指定してください。 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging 構成ファイルでは指定しないでください。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
compressedTrace を TRUE に設定して、トレース出力を圧縮します。

compressedTrace のデフォルト値は FALSE です。

compressedTrace が TRUE に設定されると、トレース出力が圧縮されます。 デフォルトのトレース出力ファイル名には、.trz という拡張子が付いています。 圧縮をデフォルト値の FALSE に設定すると、ファイルの拡張子は .trc となります。これは、非圧縮であることを示します。 ただし、トレース出力のファイル名が *traceOutputName* で指定された場合には、その名前が代わりに使用されます。ファイルに接尾部は適用されません。

圧縮されたトレース出力は、圧縮されていないものよりサイズが小さくなります。 入出力が少なくなるため、圧縮されていないトレースよりも、圧縮されたトレースのほうが書き込み速度が速くなります。 圧縮されたトレースは、圧縮されていないトレースよりも、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging のパフォーマンスに与える影響が少なくなります。

maxTraceBytes および *traceCycles* が設定されると、複数のフラット・ファイルの代わりに、複数の圧縮されたトレース・ファイルが作成されます。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging が制御されない方法で終了した場合、圧縮されたトレース・ファイルが有効でない可能性があります。 このため、トレース圧縮は、IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging が制御された方法で終了する場合にのみ使用する必要があります。 調査中の問題が原因で JVM 自体が予期せず停止することがない場合にのみ、トレース圧縮を使用してください。 System.Halt() シャットダウンまたは異常終了、

無制御の JVM 終了につながる可能性のある問題を診断しているときは、トレースの圧縮を使用しないでください。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel は、トレースのフィルター・レベルを指定します。 以下は、定義済みのトレース・レベルです。

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3
- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

各トレース・レベルはすべての下位レベルを含みます。 例えば、トレース・レベルを TRACE_INFO に設定した場合、定義済みレベルの TRACE_EXCEPTION、TRACE_WARNING、または TRACE_INFO を持つトレース・ポイントがすべてトレースに書き込まれます。 それ以外のトレース・ポイントはすべて除外されます。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace は、 IBM MQ JMS クライアント・トレース・サービスを WebSphere Application Server 環境で使用するかどうかを制御します。

standaloneTrace を TRUE に設定した場合、IBM MQ JMS クライアント・トレース・プロパティーを使用してトレース構成が決定されます。

standaloneTrace を FALSE に設定して、かつ IBM MQ JMS クライアントが WebSphere Application Server コンテナー内で実行されている場合には、WebSphere Application Server のトレース・サービスが使用されます。 生成されるトレース情報は、アプリケーション・サーバーのトレース設定によって変わります。

standaloneTrace のデフォルト値は FALSE です。

ロギング・スタンザ
IBM MQ classes for JMS のログ機能を構成する場合は、ロギング・スタンザを使用します。

ロギング・スタンザには以下のプロパティーを含めることができます。

com.ibm.msg.client.commonservices.log.outputName = path
IBM MQ classes for JMS ログ機能で使用されるログ・ファイルの名前。 デフォルト値は mqjms.log です。この値は、IBM MQ classes for JMS が実行されている Java ランタイム環境の現在の作業ディレクトリーに書き込まれます。

プロパティーには、以下のいずれかの値を使用できます。

- 単一のパス名
- パス名のコンマ区切りリスト (すべてのデータがすべてのファイルにログ記録されます)。

各パス名は絶対パス名または相対パス名にすることも、以下のように指定することもできます。

「**stderr**」または「**System.err**」
これは標準エラー・ストリームを表します。

「**stdout**」または「**System.out**」
これは標準出力ストリームを表します。

com.ibm.msg.client.commonservices.log.maxBytes
ログ・メッセージ・データの任意の呼び出し時にログに記録される最大バイト数。

正の整数
データは、ログ呼び出し当たりのそのバイト値まで書き込まれます。

0
データは書き込まれません。

-1
データは無制限に書き込まれます (デフォルト)。

com.ibm.msg.client.commonservices.log.limit
任意の 1 つのログ・ファイルに書き込まれる最大バイト数 (デフォルトは 262144 です)。

正の整数
データは、ログ・ファイル当たりのそのバイト値まで書き込まれます。

0
データは書き込まれません。

-1
データは無制限に書き込まれます。

com.ibm.msg.client.commonservices.log.count
循環するログ・ファイルの数。 各ファイルが `com.ibm.msg.client.commonservices.trace.limit` トレースに到達すると、次のファイルで開始されます。デフォルトは 3 です。

正の整数
循環するファイルの数。

0
単一ファイル。

Java SE 特性スタンザ
Java SE Specifics スタンザを使用して、 IBM MQ classes for JMS が Java Standard Edition 環境で使用されている場合に使用されるプロパティーを構成します。

com.ibm.msg.client.commonservices.j2se.produceJavaCore=TRUE|FALSE
IBM MQ classes for JMS が FDC ファイルを生成した直後に Java コア・ファイルを書き込むかどうかを決定します。 TRUE に設定すると、 IBM MQ classes for JMS が実行されている Java ランタイム環境の作業ディレクトリーに Java コア・ファイルが作成されます。

TRUE
Java ランタイム環境の機能に応じて、JavaCore を生成します。

FALSE
JavaCore を生成しません。これはデフォルト値です。

IBM MQ プロパティー・スタンザ
IBM MQ プロパティー・スタンザを使用して、 IBM MQ classes for JMS が IBM MQ と対話する方法に影響するプロパティーを設定します。

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
IBM MQ classes for JMS を使用するアプリケーションが IBM MQ メッセージング・プロバイダーのマイグレーション・モードを使用して IBM MQ キュー・マネージャーに接続している場合、IBM MQ classes for JMS はメッセージを受信する際にデフォルトのバッファー・サイズである 4 KB を使用します。 アプリケーションが取得しようとしているメッセージが 4 KB より大きい場合、IBM MQ classes for JMS はバッファーのサイズを変更して、メッセージを収容するのに十分な大きさにします。 これで、後続メッセージの受信時に、より大きいバッファー・サイズが使用されます。

このプロパティーで、バッファー・サイズを 4 KB に戻すタイミングを制御します。 デフォルトでは、この大きいバッファー・サイズより小さい 10 件のメッセージが連続して受信された場合に、バッファー・サイズが 4 KB に戻されます。 メッセージが受信されるたびにバッファー・サイズを 4 KB にリセットするには、プロパティーを 0 の値に設定します。

0

バッファーは常にデフォルト・サイズにリセットします。

10

これがデフォルト値です。 バッファーのサイズは 10 件のメッセージが受信されるたびに変更されます。

com.ibm.msg.client.wmq.receiveConversionCCSID

IBM MQ classes for JMS を使用するアプリケーションが IBM MQ メッセージング・プロバイダー通常モードを使用して IBM MQ キュー・マネージャーに接続している場合、 receiveConversionCCSID プロパティーを設定して、キュー・マネージャーからメッセージを受信するために使用される MQMD 構造体のデフォルト CCSID 値をオーバーライドすることができます。 デフォルトでは、MQMD に含まれる CCSID フィールドは 1208 に設定されますが、キュー・マネージャーがこのコード・ページにメッセージを変換できないような場合には変更が可能です。

有効値は、任意の有効な CCSID 番号または以下のいずれかの値です。

-1

プラットフォームのデフォルトを使用します。

1208

これがデフォルト値です。

クライアント・モード特性スタンザ

CLIENT トランスポートを使用しているキュー・マネージャーに IBM MQ classes for JMS が接続する際に使用されるプロパティーを指定する場合は、クライアント・モード特性のスタンザを使用します。

com.ibm.mq.polling.RemoteRequestEntry

IBM MQ classes for JMS が、キュー・マネージャーからの応答の待機時に接続の切断を確認するために使用するポーリング間隔を指定します。

正の整数

確認までの待機時間 (ミリ秒数)。 デフォルト値は 10000 (10 秒) です。 最小値は 3000 です。これより小さい値は、この最小値と同様に扱われます。

JMS クライアント動作の構成に使用されるプロパティー

これらのプロパティーを使用して、JMS クライアントの動作を構成します。

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE

JMS 2.0 仕様では、特定の動作方法が変更されました。 IBM MQ 8.0 には com.ibm.mq.jms.SupportMQExtensions というプロパティーが含まれており、これを *TRUE* に設定することで、変更された動作を以前の実装に戻すことができます。 変更された動作を元に戻す操作は、JMS 2.0 アプリケーションを使用する場合や、 JMS 1.1 API を使用するアプリケーションを IBM MQ 8.0 IBM MQ classes for JMS に対して実行する場合などに必要になります。

TRUE

SupportMQExtensions を TRUE に設定すると、以下の 3 つの機能の領域が元に戻ります。

メッセージ優先順位

メッセージに *0* から *9* までの優先度を割り当てることができます。 JMS 2.0 より前の場合、メッセージに値 *-1* (キューのデフォルトの優先度が使用されることを示す) を使用することもできます。 JMS 2.0 では、*-1* のメッセージ優先順位を設定することはできません。 SupportMQExtensions をオンにすれば、*-1* の値を使用できます。

クライアント ID

JMS 2.0 仕様では、NULL 以外のクライアント ID の固有性を接続時に確認するよう要求されます。 SupportMQExtensions をオンにすることは、この要求が無視され、クライアント ID を再使用できることを意味します。

NoLocal

JMS 2.0 仕様では、この定数がオンの場合は、コンシューマーが同じクライアント ID でパブリッシュされたメッセージを受信できないよう要求されます。 JMS 2.0 より前では、この属性はサブスクライバーに設定され、独自の接続でパブリッシュされたメッセージが受信されないようになっていました。 SupportMQExtensions をオンにすれば、この動作は以前の実装に戻ります。

FALSE
動作の変更が保持されます。

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
IBM MQ 8.0.0 Fix Pack 2 以降、アプリケーションがバイトまたはストリーム・メッセージを送信した後で、IBM MQ classes for JMS が、直前に送信されたメッセージの状態を、読み取り専用または書き込み専用に設定できるようになりました。

TRUE
送信された後、オブジェクトは読み取り専用に設定されます。 この値を設定すると、JMS 2.0 仕様との互換性が維持されます。

FALSE
送信された後、オブジェクトは書き込み専用に設定されます。 これがデフォルト値です。

関連概念
328 ページの『SupportMQExtensions プロパティー』
JMS 2.0 仕様では、特定の動作の動作方法が変更されました。 IBM MQ 8.0 以降には、プロパティー **com.ibm.mq.jms.SupportMQExtensions** が組み込まれています。これを TRUE に設定すると、変更された動作を以前の実装に戻すことができます。

z/OS 上の *IBM MQ classes for JMS* の *STEPLIB* 構成
z/OS の場合、実行時に使用される STEPLIB に IBM MQ SCSQAUTH および SCSQANLE ライブラリーが含まれていなければなりません。 これらのライブラリーは、始動 JCL または `.profile` ファイルを使用して指定します。

z/OS UNIX System Services からこれらを追加するには、以下のコード・スニペットに示すように `.profile` の行を使用することにより、`thlqual` を、IBM MQ のインストール時に選択した上位レベルのデータ・セット修飾子で置き換えます。

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

他の環境では一般的に、SCSQAUTH と SCSQANLE が STEPLIB 連結に含まれるように始動 JCL を編集する必要があります。

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
        DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for JMS とソフトウェア管理ツール
Apache Maven などのソフトウェア管理ツールは、 IBM MQ classes for JMS 、および IBM MQ classes for Jakarta Messaging で使用できます。

多くの大規模開発会社がこれらのツールを使用して、サード・パーティー・ライブラリーのリポジトリーを集中管理しています。

IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging は、いくつかの JAR ファイルで構成されています。 この API を使用して Java 言語アプリケーションを開発する場合は、IBM MQ サーバー、IBM MQ クライアント、または IBM MQ クライアント SupportPac のいずれかを、アプリケーションを開発するマシンにインストールする必要があります。

このようなツールを使用し、IBM MQ classes for JMS を構成する JAR ファイルを集中管理リポジトリーに追加する場合は、以下の点を守る必要があります。

- リポジトリーまたはコンテナーは、社内の開発者だけが使用できるようにしなければなりません。 社外に分散させることは許可されません。
- リポジトリーには、単一の IBM MQ リリースまたはフィックスパックからの整合した完全な JAR ファイル・セットを入れる必要があります。
- IBM サポートが提供するメンテナンスでリポジトリーを更新する必要があります。

以下の JAR ファイルをリポジトリーにインストールする必要があります。

- JMS 2.0 IBM MQ classes for JMS を使用する場合は、 `com.ibm.mq.allclient.jar` および `jms.jar` が必要です。
- JM 3.0 IBM MQ classes for Jakarta Messaging を使用する場合は、 `com.ibm.mq.jakarta.client.jar` および `jakarta.jms-api.jar` が必要です。
- IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用していて、ファイル・システム JNDI コンテキストに保管されている JMS 管理対象オブジェクトにアクセスする場合は、 `fscontext.jar` が必要です。
- IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用していて、ファイル・システム JNDI コンテキストに保管されている JMS 管理対象オブジェクトにアクセスする場合は、 `providerutil.jar` が必要です。
- IBM 以外の JRE のサポートには、Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイルが必要です。 詳しくは、IBM 以外の JRE のサポートを参照してください。

Java security manager での *IBM MQ classes for JMS* アプリケーションの実行

IBM MQ classes for JMS は、 Java security manager を有効にして実行できます。 Java security manager を有効にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java Virtual Machine (JVM) を構成する必要があります。

適切なポリシー定義ファイルを作成する最も簡単な方法は、Java runtime environment (JRE) に付属するポリシー構成ファイルを変更する方法です。 ほとんどのシステムでは、このファイルは、JRE ディレクトリーに関連するディレクトリー `lib/security/java.policy` にあります。 ポリシー構成ファイルは、好みのエディターを使用して編集することも、JRE に付属するポリシー・ツール・プログラムを使用して編集することもできます。

ポリシー構成ファイルの例

以下は、デフォルトのセキュリティー・マネージャーでの IBM MQ classes for JMS の正常な実行を可能にするポリシー構成ファイルの一例です。 このファイルをカスタマイズして、特定のファイルおよびディレクトリーの場所を指定する必要があります。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。 *MQ_DATA_DIRECTORY* は MQ データ・ディレクトリーの場所を表します。そして *QM_NAME* は、アクセス権限を構成するキュー・マネージャーの名前を表します。

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

  //For the client transport type.
```

```
   permission java.net.SocketPermission "*","connect,resolve";

   //For the bindings transport type.
   permission java.lang.RuntimePermission "loadLibrary.*";

   //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

   //For applications that use User Exits
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
   permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
   permission java.lang.RuntimePermission "createClassLoader";

   //Required for the z/OS platform
   permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

   // Used by the internal ConnectionFactory implementation
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

   // Used for controlled class loading
   permission java.lang.RuntimePermission "setContextClassLoader";

   // Used to default the Application name in Client mode connections
   permission java.util.PropertyPermission "sun.java.command","read";

   // Used by the IBM JSSE classes
   permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

   //Required to determine if an IBM Java Runtime is running in FIPS mode,
   //and to modify the property values status as required.
   permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
   permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
   //Required if an IBM FIPS provider is to be used for SSL communication.
   permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

   // Required for non-IBM Java Runtimes that establish secure client
   // transport mode connections using mutual TLS authentication
   permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
   permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

この例の grant ステートメントには、IBM MQ classes for JMS で必要なアクセス権が含まれています。 ポリシー構成ファイルでこれらの grant ステートメントを使用するために、IBM MQ classes for JMS をインストールした場所とアプリケーションが保管されている場所に応じてパス名を変更する必要が生じる場合があります。

IBM MQ classes for JMS に付属するサンプル・アプリケーションとそれを実行するスクリプトは、セキュリティー・マネージャーを使用可能にしません。

重要 :

IBM MQ classes for JMS トレース機能は、他にもシステム・プロパティーの照会やファイル・システム操作を行うので、さらに追加の権限が必要になります。

トレースを有効にしたセキュリティー・マネージャーの下で実行するための適切なテンプレート・セキュリティー・ポリシー・ファイルは、IBM MQ インストールの samples/wmqjava ディレクトリーで example.security.policy として提供されます。

IBM MQ classes for JMS アプリケーションのインストール後のセットアップ

このトピックでは、IBM MQ classes for JMS アプリケーションがキュー・マネージャーのリソースにアクセスするために必要な権限について説明します。 また、接続モードの概要を示し、アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成する方法を説明します。

IBM MQ の README ファイルを必ず確認してください。 このトピックの情報に優先する情報が含まれていることがあります。

特権を持たないユーザーには権限が必要な、*JMS* が使用するオブジェクト

JMS で使用されるキューにアクセスする場合、特権を持たないユーザーは、権限を付与してもらう必要があります。 どの JMS アプリケーションでも、処理対象のキュー・マネージャーに対する権限が必要になります。

IBM MQ におけるアクセス制御の詳細については、セキュリティーのセットアップを参照してください。

IBM MQ classes for JMS アプリケーションには、キュー・マネージャーに対する connect および inq 権限が必要です。 **setmqaut** 制御コマンドを使用して適切な権限を設定することができます。例えば、以下のようになります。

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Point-to-Point ドメインについては、以下の権限が必要です。

- MessageProducer オブジェクトが使用するキューには、書き込み権限が必要です。
- MessageConsumer オブジェクトと QueueBrowser オブジェクトが使用するキューには、読み取り、照会、およびブラウズ権限が必要です。
- QueueSession.createTemporaryQueue() メソッドには、QueueConnectionFactory オブジェクトの TEMPMODEL プロパティーで指定されたモデル・キューへのアクセス許可が必要です。 デフォルトで、このモデル・キューは SYSTEM.TEMP.MODEL.QUEUE です。

これらのキューのいずれかが別名キューである場合、そのターゲット・キューには照会権限が必要です。ターゲット・キューがクラスター・キューの場合、ブラウズ権限も必要です。

パブリッシュ/サブスクライブ・ドメインでは、IBM MQ classes for JMS が IBM MQ メッセージング・プロバイダー移行モードで IBM MQ キュー・マネージャーに接続している場合、以下のキューが使用されます。

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

IBM MQ メッセージング・プロバイダー移行モードについて詳しくは、 JMS **PROVIDERVERSION** プロパティーの構成を参照してください。

また、IBM MQ classes for JMS がこのモードでキュー・マネージャーに接続している場合、メッセージをパブリッシュするすべてのアプリケーションには、TopicConnectionFactory またはトピック・オブジェクトによって指定されたストリーム・キューへのアクセス権限が必要です。 デフォルトで、キューは SYSTEM.BROKER.DEFAULT.STREAM です。

ConnectionConsumer、IBM MQ リソース・アダプター、または WebSphere Application Server IBM MQ メッセージング・プロバイダーを使用する場合は、さらに追加の権限が必要になることがあります。

ConnectionConsumer によって読み取られるキューには、get、inq、および browse 権限が必要です。 システム送達不能キュー、および ConnectionConsumer によって使用されるバックアウト・リキュー・キューまたはレポート・キューには、put および passall 権限が必要です。

パブリッシュ/サブスクライブ・メッセージングを実行するために IBM MQ メッセージング・プロバイダーの通常モードを使用するアプリケーションは、キュー・マネージャーによって提供される統合されたパブリッシュ/サブスクライブ機能を使用します。 使用されるトピックおよびキューの機密保護については、パブリッシュ/サブスクライブのセキュリティーを参照してください。

IBM MQ classes for JMS の接続モード

IBM MQ classes for JMS アプリケーションは、クライアント・モードかバインディング・モードのいずれかでキュー・マネージャーに接続できます。 クライアント・モードでは、IBM MQ classes for JMS は TCP/IP を介してキュー・マネージャーに接続します。 バインディング・モードでは、IBM MQ classes for JMS は Java Native Interface (JNI) を使用して直接キュー・マネージャーに接続します。

z/OS 上の WebSphere Application Server で実行されているアプリケーションは、バインディング・モードまたはクライアント・モードのいずれかでキュー・マネージャーに接続できますが、 z/OS 上の他の環境で

実行されているアプリケーションは、バインディング・モードでのみキュー・マネージャーに接続できます。 それ以外のプラットフォームで実行されるアプリケーションはすべて、バインディング・モードかクライアント・モードのいずれかでキュー・マネージャーに接続できます。

現行のキュー・マネージャーで、IBM MQ classes for JMS のサポートされている現行バージョンまたは旧バージョンを使用できます。また、IBM MQ classes for JMS の現行バージョンで、キュー・マネージャーのサポートされている現行バージョンまたは旧バージョンを使用できます。 異なるバージョンを混用する場合、機能は旧バージョンのレベルに制限されます。

以下のセクションでは、それぞれの接続モードについて詳しく説明します。

クライアント・モード

クライアント・モードでキュー・マネージャーに接続する際には、キュー・マネージャーが実行されているのと同じシステムか、または別のシステムで IBM MQ classes for JMS アプリケーションを実行できます。いずれの場合も、IBM MQ classes for JMS は、TCP/IP を介してキュー・マネージャーに接続します。

バインディング・モード

バインディング・モードでキュー・マネージャーを接続するには、キュー・マネージャーが実行されているのと同じシステムで IBM MQ classes for JMS アプリケーションを実行する必要があります。

IBM MQ classes for JMS は Java Native Interface (JNI) を使用して直接キュー・マネージャーに接続します。 バインディング・トランスポートを使用するには、IBM MQ classes for JMS を、IBM MQ Java Native Interface ライブラリーにアクセスできる環境で実行する必要があります。詳細については、97 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照してください。

IBM MQ classes for JMS は、*ConnectOption* について以下の値をサポートしています。

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

IBM MQ classes for JMS で使用される接続オプションを変更するには、接続ファクトリーのプロパティー CONNOPT を変更します。

接続オプションの詳細については、738 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』を参照してください。

バインディング・トランスポートを使用するには、使用中の Java ランタイム環境で、IBM MQ classes for JMS が接続されているキュー・マネージャーのコード化文字セット ID (CCSID) がサポートされている必要があります。

Java ランタイム環境でサポートされる CCSID を判別する方法について詳しくは、IBM MQ FDC with Probe ID 21 generated when using the IBM MQ V7 classes for Java or IBM MQ V7 classes for JMS を参照してください。

IBM MQ classes for JMS アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成する
IBM MQ classes for JMS アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成するには、サーバー接続チャネル定義を作成し、リスナーを開始する必要があります。

サーバー接続チャネル定義の作成

サーバー接続チャネル定義は、MQSC コマンド DEFINE CHANNEL を使用してすべてのプラットフォームで作成できます。 次のような例があります。

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i IBM i では、次の例のように、CL コマンド CRTMQMCHL を代わりに使用することができます。

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

このコマンドの *QMGRNAME* は、キュー・マネージャーの名前です。

Windows Linux Linux および Windows では、IBM MQ Explorer を使用してサーバー接続チャネル定義を作成することもできます。

z/OS z/OS では、オペレーションおよび制御パネルを使用して、サーバー接続チャネル定義を作成できます。

チャネルの名前 (上記の例では JAVA.CHANNEL) は、アプリケーションがキュー・マネージャーとの接続で使用する接続ファクトリーの CHANNEL プロパティーで指定されたチャネル名と同じでなければなりません。 CHANNEL プロパティーのデフォルト値は SYSTEM.DEF.SVRCONN です。

リスナーの開始

キュー・マネージャーのリスナーがまだ開始されていない場合は、開始する必要があります。

Multi マルチプラットフォームでは、以下の例に示すように、最初に MQSC コマンド DEFINE LISTENER を使用してリスナー・オブジェクトを作成した後で、MQSC コマンド START LISTENER を使用してリスナーを開始できます。

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS z/OS では、次の例のように、START LISTENER コマンドだけを使用しますが、リスナーを開始する前に、チャネル・イニシエーターのアドレス・スペースを開始する必要があります。

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i IBM i では、次の例のように、CL コマンド STRMQMLSR を使用してリスナーを開始することもできます。

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

このコマンドの *QMGRNAME* は、キュー・マネージャーの名前です。

ALW AIX, Linux, and Windows では、次の例のように、制御コマンド **runmqlsr** を使用してリスナーを開始することもできます。

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

このコマンドの *QMgrName* は、キュー・マネージャーの名前です。

Windows Linux Linux と Windows では、IBM MQ Explorer を使用してリスナーを開始することもできます。

z/OS z/OS では、オペレーションおよび制御パネルを使用してリスナーを開始することもできます。

リスナーが listen しているポートの番号は、アプリケーションがキュー・マネージャーとの接続で使用する接続ファクトリーの PORT プロパティーで指定されたポート番号と同じでなければなりません。 PORT プロパティーのデフォルト値は 1414 です。

IBM MQ classes for JMS の Point-to-Point IVT

Point-to-Point インストール検査テスト (IVT) プログラムは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、メッセージを SYSTEM.DEFAULT.LOCAL.QUEUE というキューに送信した後、メッセージをキューから受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

まず、インストール検査テストを JNDI を使わずに実行します。 このテストは自己完結型であり、ディレクトリー・サービスの使用を必要としないからです。 管理対象オブジェクトについては、管理ツールを使用した JMS オブジェクトの構成を参照してください。

JNDI を使用しない Point-to-Point インストール検査テスト

このテストで IVT プログラムは、必要なオブジェクトをすべて実行時に動的に作成および構成し、JNDI は使用しません。

Multi マルチプラットフォームでは、IVT プログラムを実行するためのスクリプトが用意されています。 このスクリプトは、 AIX and Linux システムでは **IVTRun** 、 Windows では **IVTRun.bat** と呼ばれます。 スクリプトは、IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。 クラスパスに `com.ibm.mqjms.jar` が含まれている必要があります。

テストをバインディング・モードで実行するには、以下のコマンドを入力します。

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

クライアント・モードでテストを実行するには、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』の説明に従って、まずキュー・マネージャーをセットアップします。 使用されるチャネルのデフォルトは **SYSTEM.DEF.SVRCONN** であり、使用されるキューは **SYSTEM.DEFAULT.LOCAL.QUEUE** であることに注意してから、次のコマンドを入力します。

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-v providerVersion ] [-ccsid ccsid ] [-t]
```

z/OS z/OS システムでは、同等のスクリプトは提供されません。 代わりに、 Java クラスを直接呼び出すことによって、バインディング・モードで IVT を実行します。 z/OS では、IVT プログラムの 2 つの機能的に同一のインスタンスから選択します。

- `com.ibm.mq.jms.MQJMSIVT`。 IBM MQ classes for JMS (JMS 2.0) で使用できます。 このプログラムを使用するには、クラスパスに `com.ibm.mqjms.jar` または `com.ibm.mq.allclient.jar` が含まれている必要があります。
- `com.ibm.mq.jakarta.jms.MQJMSIVT`。 IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) で使用できます。 このプログラムを使用するには、クラスパスに `com.ibm.mq.jakarta.client.jar` が含まれている必要があります。

z/OS でバインディング・モードでテストを実行するには、以下のコマンドを入力します。

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-m *qmgr*
IVT プログラムの接続先のキュー・マネージャーの名前。 テストをバインディング・モードで実行するときにこのパラメーターを省略すると、IVT プログラムはデフォルトのキュー・マネージャーに接続します。

-host *hostname*
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

-port *port*
キュー・マネージャーのリスナーが listen しているポートの番号。 デフォルト値は 1414 です。

-channel *channel*
IVT プログラムがキュー・マネージャーへの接続で使用する MQI チャネルの名前。 デフォルト値は SYSTEM.DEF.SVRCONN です。

-v *providerVersion*
IVT プログラムの接続先として想定されるキュー・マネージャーのリリース・レベル。

このパラメーターは、MQQueueConnectionFactory オブジェクトの PROVIDERVERSION プロパティーを設定するために使用され、PROVIDERVERSION プロパティーと同じ有効値を持ちます。 それで、このパラメーターとその有効値について詳しくは、JMS: PROVIDERVERSION プロパティーの変更点と、IBM MQ classes for JMS オブジェクトのプロパティーに記載されている PROVIDERVERSION プロパティーの説明を参照してください。

デフォルト値は unspecified です。

-ccsid *ccsid*
接続で使用されるコード化文字セットまたはコード・ページの ID (CCSID)。 デフォルト値は 819 です。

-t
トレースを有効にします。 デフォルトでは、トレースは無効です。

テストが正常に行われると、次の出力例のような出力が生成されます。

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
```

```
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

JNDI を使用した Point-to-Point インストール検査テスト

このテストでは、IVT プログラムは JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出します。

テストを実行する前に、Lightweight Directory Access Protocol (LDAP) サーバーまたはローカル・ファイル・システムに基づくディレクトリー・サービスを構成しておく必要があります。 また、ディレクトリー・サービスを使用して管理対象オブジェクトを保管できるように IBM MQ JMS 管理ツールを構成しておく必要もあります。 これらの前提条件について詳しくは、89 ページの『IBM MQ classes for JMS の前提条件』を参照してください。 IBM MQ JMS 管理ツールを構成する方法については、「JMS 管理ツールの構成」を参照してください。

IVT プログラムで JNDI を使用して MQQueueConnectionFactory オブジェクトおよび MQQueue オブジェクトをディレクトリー・サービスから取り出せるようにしておく必要があります。 これらの管理オブジェクトを作成するためのスクリプトが用意されています。 このスクリプトは、 AIX and Linux システムでは IVTSetup、 Windows では IVTSetup.bat と呼ばれており、 IBM MQ classes for JMS インストール・ディレクトリーの bin サブディレクトリーにあります。 スクリプトを実行するには、以下のコマンドを入力します。

```
IVTSetup
```

このスクリプトは IBM MQ JMS 管理ツールを呼び出し、管理対象オブジェクトを作成します。

MQQueueConnectionFactory オブジェクトは ivtQCF という名前でバインドされ、すべてそのプロパティーのデフォルト値で作成されます。これは、IVT プログラムがバインディング・モードで実行され、デフォルトのキュー・マネージャーに接続することを意味します。 IVT プログラムをクライアント・モードで実行する場合、またはデフォルト以外のキュー・マネージャーに接続する場合は、IBM MQ JMS 管理ツールまたは IBM MQ Explorer を使用して、MQQueueConnectionFactory オブジェクトの該当するプロパティーを変更する必要があります。 IBM MQ Explorer JMS 管理ツールの使用方法については、管理ツールを使用した JMS オブジェクトの構成を参照してください。 IBM MQ Explorer の使用方法について詳しくは、IBM MQ Explorer の概要、または IBM MQ Explorer に付属するヘルプを参照してください。

MQQueue オブジェクトは ivtQ という名前でバインドされ、QUEUE プロパティー (SYSTEM.DEFAULT.LOCAL.QUEUE という値を持つ) を除き、すべてそのプロパティーのデフォルト値で作成されます。

管理対象オブジェクトを作成した後、IVT プログラムを実行できます。 JNDI を使用してテストを実行するには、以下のコマンドを入力します。

```
IVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-url "*providerURL*"
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://hostname/contextName` (LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/directoryPath` (ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

URL は引用符 (") で囲む必要があります。

-icf *initCtxFact*
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値でなければなりません。

- `com.sun.jndi.ldap.LdapCtxFactory`(LDAP サーバーに基づくディレクトリー・サービスの場合)。 これがデフォルト値です。
- `com.sun.jndi.fscontext.RefFSContextFactory`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)。

-t
トレースを有効にします。 デフォルトでは、トレースは無効です。

テストが正常に行われると、JNDI を使用しない場合にテストが正常に行われたときに生成される出力と同様の出力が生成されます。 主な違いは、テストで JNDI を使用して MQQueueConnectionFactory オブジェクトおよび MQQueue オブジェクトが取り出されたことが出力で示される点です。

必須ではありませんが、IVTSetup スクリプトによって作成された管理対象オブジェクトを削除してテスト後にタイディアップすることをお勧めします。 これを行うためのスクリプトが用意されています。 このスクリプトは、 AIX and Linux システムでは IVTTidy、 Windows では IVTTidy.bat と呼ばれており、 IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

Point-to-Point インストール検査テストの問題判別

Multi

インストール検査テストは、次のような理由で失敗することがあります。

- クラスを検出できないことを示すメッセージが IVT プログラムによって書き込まれる場合は、94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているとおりにクラス・パスが正しく設定されているか確認してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
and host name ' hostname '
```

関連する理由コードは 2059 です。 メッセージの変数の意味は次のとおりです。

qmgr
IVT プログラムの接続試行先のキュー・マネージャーの名前。 IVT プログラムがバインディング・モードでデフォルトのキュー・マネージャーに接続を試行している場合、このメッセージ挿入はブランクになります。

connMode
接続モード。 `Bindings` または `Client` のいずれかです。

hostname
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

このメッセージは、IVT プログラムの接続試行先のキュー・マネージャーが使用不可であることを意味します。 キュー・マネージャーが稼働していること、および IVT プログラムがデフォルトのキュー・マネージャーに接続を試行している場合は、そのキュー・マネージャーがシステムのデフォルトのキュー・マネージャーとして定義されていることを確認してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

このメッセージは、IVT プログラムが接続されているキュー・マネージャーにキュー SYSTEM.DEFAULT.LOCAL.QUEUE が存在しないことを意味します。 あるいは、キューが存在していてもメッセージの書き込みおよび取得が有効になっていないために IVT プログラムはそのキューを開くことができません。 キューが存在すること、およびメッセージの書き込みと取得が有効になっていることを確認してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
Unable to bind to object
```

  このメッセージは、LDAP サーバーとの接続は存在するものの、LDAP サーバーが正しく構成されていないことを意味します。 LDAP サーバーが Java オブジェクトを保管するように構成されていないか、オブジェクトに対するアクセス権または接尾部が正しくないかのどちらかです。 この状態でのヘルプ情報については、ご使用の LDAP サーバーの資料を参照してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

  このメッセージは、システムからのクライアント接続を受け入れるようにキュー・マネージャーが正しくセットアップされていないことを意味します。 詳細は、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』を参照してください。

IBM MQ classes for JMS のパブリッシュ/サブスクライブ *IVT*

パブリッシュ/サブスクライブ・インストール検査テスト (IVT) プログラムは、IBM MQ classes for JMS に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、トピックにサブスクライブし、メッセージをトピックにパブリッシュした後、そのメッセージを受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

まず、インストール検査テストを JNDI を使わずに実行します。 このテストは自己完結型であり、ディレクトリー・サービスの使用を必要としないからです。 管理対象オブジェクトについては、管理ツールを使用した JMS オブジェクトの構成を参照してください。

JNDI を使用しないパブリッシュ/サブスクライブ・インストール検査テスト

このテストで IVT プログラムは、必要なオブジェクトをすべて実行時に動的に作成および構成し、JNDI は使用しません。

IVT プログラムを実行するためのスクリプトが用意されています。 このスクリプトは、AIX and Linux システムでは PSIVTRun、 Windows では PSIVTRun.bat と呼ばれており、 IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

テストをバインディング・モードで実行するには、以下のコマンドを入力します。

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

クライアント・モードでテストを実行する場合は、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』の説明に従ってまずキュー・マネージャーをセットアップします。その際には、使用されるチャネルがデフォルトでは SYSTEM.DEF.SVRCONN であることに注意してください。次いで、以下のコマンドを入力します。

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port ] [-channel channel ]
[-bqm brokerQmgr ] [-v providerVersion ] [-ccsid ccsid ] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-m *qmgr*
: IVT プログラムの接続先のキュー・マネージャーの名前。 テストをバインディング・モードで実行するときにこのパラメーターを省略すると、IVT プログラムはデフォルトのキュー・マネージャーに接続します。

-host *hostname*
: キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

-port *port*
キュー・マネージャーのリスナーが listen しているポートの番号。 デフォルト値は 1414 です。

-channel *channel*
IVT プログラムがキュー・マネージャーへの接続で使用する MQI チャネルの名前。 デフォルト値は SYSTEM.DEF.SVRCONN です。

-bqm *brokerQmgr*
ブローカーが稼働しているキュー・マネージャーの名前。 デフォルト値は、IVT プログラムの接続先のキュー・マネージャーの名前です。

このパラメーターは、キュー・マネージャーのバージョン番号 v が 7 以上の場合は関係ありません。

-v *providerVersion*
IVT プログラムの接続先として想定されるキュー・マネージャーのリリース・レベル。

このパラメーターは、MQTopicConnectionFactory オブジェクトの PROVIDERVERSION プロパティーを設定するために使用され、PROVIDERVERSION プロパティーと同じ有効値を持ちます。 それで、このパラメーターの詳細 (その有効値を含む) については、IBM MQ classes for JMS オブジェクトのプロパティーにある PROVIDERVERSION プロパティーの説明を参照してください。

デフォルト値は unspecified です。

-ccsid *ccsid*
接続で使用されるコード化文字セットまたはコード・ページの ID (CCSID)。 デフォルト値は 819 です。

-t
トレースを有効にします。 デフォルトでは、トレースは無効です。

テストが正常に行われると、次の出力例のような出力が生成されます。

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
```

```
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

JNDI を使用したパブリッシュ/サブスクライブ・インストール検査テスト

このテストでは、IVT プログラムは JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出します。

テストを実行する前に、Lightweight Directory Access Protocol (LDAP) サーバーまたはローカル・ファイル・システムに基づくディレクトリー・サービスを構成しておく必要があります。 また、ディレクトリー・サービスを使用して管理対象オブジェクトを保管できるように IBM MQ JMS 管理ツールを構成しておく必要もあります。 これらの前提条件について詳しくは、89 ページの『IBM MQ classes for JMS の前提条件』を参照してください。 IBM MQ JMS 管理ツールを構成する方法については、「JMS 管理ツールの構成」を参照してください。

IVT プログラムで JNDI を使用して MQTopicConnectionFactory オブジェクトおよび MQTopic オブジェクトをディレクトリー・サービスから取り出せるようにしておく必要があります。 これらの管理オブジェクトを作成するためのスクリプトが用意されています。 このスクリプトは、 AIX and Linux システムでは IVTSetup、 Windows では IVTSetup.bat と呼ばれており、 IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。 スクリプトを実行するには、以下のコマンドを入力します。

```
IVTSetup
```

このスクリプトは IBM MQ JMS 管理ツールを呼び出し、管理対象オブジェクトを作成します。

MQTopicConnectionFactory オブジェクトは ivtTCF という名前でバインドされ、すべてそのプロパティーのデフォルト値で作成されます。これは、IVT プログラムがバインディング・モードで実行され、デフォルトのキュー・マネージャーに接続し、組み込みパブリッシュ/サブスクライブ機能を使用することを意味します。 IVT プログラムをクライアント・モードで実行する場合、またはデフォルト以外のキュー・マネージャーに接続する場合、あるいは組み込みパブリッシュ/サブスクライブ機能の代わりに IBM Integration Bus を使用する場合には、IBM MQ JMS 管理ツールまたは IBM MQ エクスプローラーを使用して、MQTopicConnectionFactory オブジェクトの該当するプロパティーを変更する必要があります。 IBM MQ JMS 管理ツールの使用方法については、管理ツールを使用した JMS オブジェクトの構成を参照してください。 IBM MQ エクスプローラーの使い方については、IBM MQ エクスプローラーに付属するヘルプを参照してください。

MQTopic オブジェクトは ivtT という名前でバインドされ、TOPIC プロパティー (MQJMS/PSIVT/Information という値を持つ) を除き、すべてそのプロパティーのデフォルト値で作成されます。

管理対象オブジェクトを作成した後、IVT プログラムを実行できます。 JNDI を使用してテストを実行するには、以下のコマンドを入力します。

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-url "*providerURL*"
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://`*`hostname`*`/`*`contextName`* (LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/`*`directoryPath`* (ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

URL は引用符 (") で囲む必要があります。

-icf *initCtxFact*
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値でなければなりません。

- `com.sun.jndi.ldap.LdapCtxFactory`(LDAP サーバーに基づくディレクトリー・サービスの場合)。 これがデフォルト値です。
- `com.sun.jndi.fscontext.RefFSContextFactory`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)。

-t
トレースを有効にします。 デフォルトでは、トレースは無効です。

テストが正常に行われると、JNDI を使用しない場合にテストが正常に行われたときに生成される出力と同様の出力が生成されます。 主な違いは、テストで JNDI を使用して MQTopicConnectionFactory オブジェクトおよび MQTopic オブジェクトが取り出されたことが出力で示される点です。

必須ではありませんが、IVTSetup スクリプトによって作成された管理対象オブジェクトを削除してテスト後にタイディアップすることをお勧めします。 これを行うためのスクリプトが用意されています。 このスクリプトは、 AIX and Linux システムでは IVTTidy、 Windows では IVTTidy.bat と呼ばれており、 IBM MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

パブリッシュ/サブスクライブ・インストール検査テストの問題判別

インストール検査テストは、次のような理由で失敗することがあります。

- クラスを検出できないことを示すメッセージが IVT プログラムによって書き込まれる場合は、94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているとおりにクラス・パスが正しく設定されているか確認してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
Failed to connect to queue manager ' qmgr ' with
connection mode ' connMode ' and host name ' hostname '
```

関連する理由コードは 2059 です。 メッセージの変数の意味は次のとおりです。

qmgr
IVT プログラムの接続試行先のキュー・マネージャーの名前。 IVT プログラムがバインディング・モードでデフォルトのキュー・マネージャーに接続を試行している場合、このメッセージ挿入はブランクになります。

connMode
接続モード。 `Bindings` または `Client` のいずれかです。

hostname
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

このメッセージは、IVT プログラムの接続試行先のキュー・マネージャーが使用不可であることを意味します。 キュー・マネージャーが稼働していること、および IVT プログラムがデフォルトのキュー・マネージャーに接続を試行している場合は、そのキュー・マネージャーがシステムのデフォルトのキュー・マネージャーとして定義されていることを確認してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
Unable to bind to object
```

このメッセージは、LDAP サーバーとの接続は存在するものの、LDAP サーバーが正しく構成されていないことを意味します。 LDAP サーバーが Java オブジェクトを保管するように構成されていないか、オブジェクトに対するアクセス権または接尾部が正しくないかのどちらかです。 この状態でのヘルプ情報については、ご使用の LDAP サーバーの資料を参照してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

このメッセージは、システムからのクライアント接続を受け入れるようにキュー・マネージャーが正しくセットアップされていないことを意味します。 詳細については、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』を参照してください。

JMS 2.0 IBM MQ classes for JMS サンプル・アプリケーションの使用

IBM MQ classes for JMS サンプル・アプリケーションは、JMS API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

このタスクについて

独自のアプリケーションを作成するために手助けが必要な場合は、サンプル・アプリケーションを開始点として使用できます。 各アプリケーションのソース・バージョンとコンパイル・バージョンの両方が提供されています。 サンプルのソース・コードを検討し、アプリケーションに必要な各オブジェクト (ConnectionFactory、Connection、Session、Destination、さらに Producer と Consumer のいずれかまたは両方) を作成するための主な手順や、アプリケーションの動作を指定するときに必要な特定のプロパティーを設定するための主な手順を見極めます。 詳細については、141 ページの『IBM MQ classes for JMS/Jakarta Messaging アプリケーションの作成』を参照してください。 これらのサンプルは、IBM MQ の将来のリリースで変更される可能性があります。

JMS 2.0 の場合、121 ページの表 11 は、各プラットフォームで IBM MQ classes for JMS サンプル・アプリケーションがインストールされる場所を示しています。

注 :

JM 3.0 IBM MQ classes for Jakarta Messaging の場合、新しいサンプルを準備しています。

表 *11. IBM MQ classes for JMS* サンプル・アプリケーションのインストール・ディレクトリー

プラットフォーム	ディレクトリー
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*/samp/jms/samples
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms\samples
IBM i IBM i	/qibm/proddata/mqm/java/samples/jms/samples
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

このディレクトリー内には、121 ページの表 12 に示すように、1 つ以上のサンプル・アプリケーションを収めたサブディレクトリーがあります。

表 *12. IBM MQ classes for JMS* サンプル・アプリケーション

サンプル名	説明
JmsBrowser.java	指定されたキュー上にある使用可能なすべてのメッセージを、コンシューマー・アプリケーションが受け取る順序で、削除せずに閲覧する JMS キュー・ブラウザー・アプリケーション。
JmsConsumer.java	接続ファクトリー・インスタンスおよび宛先インスタンスを初期コンテキストで検索することによって、指定されたキュー上にある使用可能なすべてのメッセージを、コンシューマー・アプリケーションが受け取る順序で、削除せずに閲覧する JMS キュー・ブラウザー・アプリケーション (このサンプルは、ファイル・システム・コンテキストのみをサポートします)。

表 *12. IBM MQ classes for JMS* サンプル・アプリケーション *(続き)*	
サンプル名	**説明**
JmsJndiConsumer.java	接続ファクトリー・インスタンスおよび宛先インスタンスを初期コンテキストで検索することによって、指定された宛先 (キューまたはトピック) からメッセージを受け取る JMS コンシューマー (受信側またはサブスクライバー) アプリケーション (このサンプルは、ファイル・システム・コンテキストのみをサポートします)。
JmsJndiProducer.java	接続ファクトリー・インスタンスおよび宛先インスタンスを初期コンテキストで検索することによって、指定された宛先 (キューまたはトピック) に単純メッセージを送信する JMS プロデューサー (送信側またはパブリッシャー) アプリケーション (このサンプルは、ファイル・システム・コンテキストのみをサポートします)。
JmsProducer.java	指定された宛先 (キューまたはトピック) に単純メッセージを送信する JMS プロデューサー (送信側またはパブリッシャー) アプリケーション。
/interactive/	
SampleConsumerJava.java	トピック/キューからメッセージを受信します。
SampleProducerJava.java	トピック/キューにメッセージを送信します。
/interactive/helper/	
BaseOptions.java	ユーザー・オプション機能を提供するために拡張可能な抽象クラス。
IsValidType.java	妥当性検査プログラム・クラスのための抽象クラス。
JmsApp.java	コンシューマー/プロデューサー機能を提供するために拡張可能な抽象クラス。
Keys.java	サンプル・アプリケーションのオプションを定義するキーのセット。
Literals.java	定数リテラルのセット。
MyContext.java	オプションが表示されるコンテキスト。
Options.java	ユーザー・オプションの機能を提供します。
OptionsPresenter.java	現在のオプションが表示されるコンテキスト。
/simple/	
SimpleAsyncPutPTP.java	Point-to-Point メッセージング用の単純なアプリケーション。メッセージは非同期で送信されます (応答不要送信メッセージングとも呼ばれます)。 受信するメッセージはありません。
SimpleDurableSub.java	永続サブスクリプション機能の実例を示す単純なアプリケーション。
SimpleJNDILookup.java	初期コンテキストを使用した JMS オブジェクトの検索の実例を示す単純で最小のアプリケーション。 キュー・マネージャーへの接続は行われず、メッセージの送受信は行われません。

表 12. *IBM MQ classes for JMS* サンプル・アプリケーション (続き)	
サンプル名	**説明**
SimpleMQMDRead.java	JMS アプリケーションが MQ Message Descriptor (MQMD) フィールドを JMS メッセージ・プロパティーとして利用する方法の実例を示す単純なアプリケーション。 メッセージは送信されません。使用するキューにいくつかのメッセージが入っているという前提で機能します。
SimpleMQMDWrite.java	JMS アプリケーションが MQ Message Descriptor (MQMD) フィールドに書き込む方法の実例を示す単純なアプリケーション。 受信するメッセージはありません。
SimplePTP.java	Point-to-Point メッセージング用の単純な最小のアプリケーション。
SimplePubSub.java	パブリッシュ・サブスクライブ・メッセージング用の単純な最小のアプリケーション。
SimpleReadAheadPTP.java	Point-to-Point メッセージング用の単純なアプリケーション。メッセージはキュー・マネージャーからストリームされます (先読み機能とも呼ばれます)。 メッセージは送信されません。使用するキューにいくつかのメッセージが入っているという前提で機能します。
SimpleRequestor.java	リクエスターを使用して要求メッセージを送信し、その応答を待機して受信する単純なアプリケーション。 注: これは、要求メッセージを処理し応答メッセージを送信する他のアプリケーションがあるという前提で機能します。
SimpleResponder.java	メッセージの宛先で listen し、メッセージの replyTo 宛先に応答を送信する単純なアプリケーション。 このアプリケーションは、SimpleRequestor サンプルと組み合わせて作動するように作成されています。
SimpleRetainedPub.java	保存パブリケーションの実例を示す単純なアプリケーション。 受信するメッセージはありません。
SimpleWMQJMSPTP.java	Point-to-Point メッセージング用の単純な最小のアプリケーション。
SimpleWMQJMSPubSub.java	パブリッシュ/サブスクライブ・メッセージング用の単純な最小のアプリケーション。

IBM MQ classes for JMS は、サンプル・アプリケーションの実行に使用できる `runjms` と呼ばれるスクリプトを提供します。 このスクリプトは、IBM MQ 環境をセットアップして、IBM MQ classes for JMS サンプル・アプリケーションを実行できるようにします。

123 ページの表 13 は、プラットフォームごとのスクリプトの場所を示しています。

表 13. *runjms* スクリプトのロケーション	
プラットフォーム	**ディレクトリー**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	または
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

runjms スクリプトを使用してサンプル・アプリケーションを起動するには、以下のステップを実行します。

手順

1. コマンド・プロンプトを表示し、実行するサンプル・アプリケーションが格納されているディレクトリーにナビゲートします。
2. 以下のコマンドを入力します。

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   サンプル・アプリケーションは、必要とするパラメーターのリストを表示します。
3. 次のコマンドを入力して、これらのパラメーターを指定してサンプルを実行します。

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

例

Linux 例えば、 Linux で JmsBrowser サンプルを実行するには、以下のコマンドを入力します。

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

関連概念

90 ページの『IBM MQ classes for JMS でインストールされる内容』
IBM MQ classes for JMS のインストール時にはいくつかのファイルとディレクトリーが作成されます。 Windows では、インストールの際に、自動的に環境変数を設定することによっていくつかの構成が実行されます。 他のプラットフォームやある特定の Windows 環境では、IBM MQ classes for JMS アプリケーションを実行するために、まず環境変数を設定する必要があります。

IBM MQ classes for JMS/Jakarta Messaging で提供されるスクリプト

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の使用時に実行する必要がある一般的なタスクを支援するために、いくつかのスクリプトが用意されています。

124 ページの表 14 では、すべてのスクリプトとその用法をリストしています。 スクリプトは、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging インストール・ディレクトリーの bin サブディレクトリーにあります。

表 *14. IBM MQ classes for JMS* および *IBM MQ classes for Jakarta Messaging* で提供されるスクリプト

ユーティリティー	以下を使用してください。
Cleanup [1]	このスクリプトは前のリリースとの互換性のために維持されていますが、実行する機能はありません。 サブスクリプション情報の手動クリーンアップは不要になりました。
DefaultConfiguration	Windows 以外のプラットフォーム上でデフォルト構成アプリケーションを実行します。
formatLog [1]	このスクリプトは前のリリースとの互換性のために維持されていますが、実行する機能はありません。 ログ出力は、読み取り可能なテキストで生成されるようになりました。
IVTRun [1] IVTSetup [1] IVTTidy [1]	point-to-point インストール検査テストで使用されます (113 ページの『IBM MQ classes for JMS の Point-to-Point IVT』 を参照)。
JM 3.0 JMS30Admin [1]	IBM MQ Jakarta Messaging 管理ツールを実行します (管理ツールの開始を参照)。

表 14. *IBM MQ classes for JMS* および *IBM MQ classes for Jakarta Messaging* で提供されるスクリプト (続き)

ユーティリティー	以下を使用してください。
JM 3.0 JMS30Admin.config	IBM MQ Jakarta Messaging 管理ツールの構成ファイル (JMS 管理ツールの構成を参照)。
JMS 2.0 JMSAdmin [1]	IBM MQ JMS 管理ツールを実行します (管理ツールの開始を参照)。
JMS 2.0 JMSAdmin.config	IBM MQ JMS 管理ツールの構成ファイル (JMS 管理ツールの構成を参照)。
PSIVTRun [1]	パブリッシュ/サブスクライブのインストール検査テスト・プログラムを実行します (117 ページの『IBM MQ classes for JMS のパブリッシュ/サブスクライブ IVT』を参照)。
PSReportDump.class	このクラスは以前のリリースとの互換性のために維持されていますが、実行する機能はありません。
JM 3.0 V 9.3.0 V 9.3.0 setjms30env 125 ページの『2』	Linux AIX Jakarta Messaging 3.0 の場合、 94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているように、 AIX and Linux システム上の 32 ビット Java 仮想マシン (JVM) で IBM MQ classes for JMS アプリケーションを実行するための環境変数を設定します。
JMS 2.0 setjmsenv 125 ページの『2』	Linux AIX JMS 2.0 の場合、 94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているように、AIX and Linux システム上の 32 ビット Java 仮想マシン (JVM) で IBM MQ classes for JMS アプリケーションを実行するための環境変数を設定します。
JM 3.0 V 9.3.0 V 9.3.0 setjms30env64 125 ページの『2』	Linux AIX Jakarta Messaging 3.0 の場合、 94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているように、 AIX and Linux システム上の 64 ビット JVM で IBM MQ classes for JMS アプリケーションを実行するための環境変数を設定します。
JMS 2.0 setjmsenv64 125 ページの『2』	Linux AIX JMS 2.0 の場合、 94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』で説明されているように、 AIX and Linux システム上の 64 ビット JVM で IBM MQ classes for JMS アプリケーションを実行するための環境変数を設定します。

注:

1. Windows では、ファイル名の拡張子は `.bat` です。
2. これらのスクリプトは、AIX and Linux でのみ使用可能です。 Windows では、IBM MQ をインストールした後、コマンド **setmqenv** を実行します。 詳しくは、94 ページの『IBM MQ classes for JMS/Jakarta Messaging の環境変数の設定』を参照してください。

IBM MQ classes for JMS での OSGi のサポート

OSGi は、バンドルの形によるアプリケーションのデプロイメントをサポートするフレームワークを提供します。 OSGi バンドルは、 IBM MQ classes for JMS の一部として提供されます。

IBM MQ classes for JMS には、以下の OSGi バンドルが組み込まれています。

com.ibm.msg.client.osgi.jms*version_number*.jar
IBM MQ classes for JMS 内のコードの共通レイヤー。 JMS の IBM MQ クラスの階層化アーキテクチャーについては、「JMS アーキテクチャーの IBM MQ クラス」を参照してください。

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
共通レイヤーの前提条件 Java アーカイブ (JAR) ファイル。

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Java Platform, Standard Edition (Java SE) アプリケーションの共通サービス。

com.ibm.msg.client.osgi.nls_*version_number*.jar
共通レイヤーのメッセージ。

com.ibm.msg.client.osgi.wmq_*version_number*.jar
IBM MQ classes for JMS 内の IBM MQ メッセージング・プロバイダー。 IBM MQ classes for JMS の階層化アーキテクチャーについては、「JMS アーキテクチャーの IBM MQ クラス」を参照してください。

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
IBM MQ メッセージング・プロバイダーの前提条件 JAR ファイル。

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
IBM MQ メッセージング・プロバイダーのメッセージ。

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、この JAR ファイルにより、アプリケーションは IBM MQ classes for JMS と IBM MQ classes for Java の両方を使用できます。また、PCF メッセージを処理するためのコードも含まれています。

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、この JAR ファイルは com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar の前提条件を提供します。

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 JMS 2.0 の場合、この JAR ファイルにより、アプリケーションは IBM MQ classes for JMS と IBM MQ classes for Java の両方を使用できます。また、PCF メッセージを処理するコードも含まれています。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 JMS 2.0 の場合、この JAR ファイルは com.ibm.mq.osgi.allclient_*version_number*.jar の前提条件を提供します。

ここで、 *version_number* は、インストールされている IBM MQ のバージョン番号です。

これらのバンドルは、IBM MQ インストール済み環境の java/lib/OSGi サブディレクトリーにインストールされるか、Windows 上の java\lib\OSGi フォルダーにインストールされます。

IBM MQ 8.0 以降では、あらゆる新しいアプリケーションのためにバンドル com.ibm.mq.osgi.allclient_8.0.0.0.jar、および com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar を使用します。 このバンドルを使用することで、同一 OSGi フレームワーク内で IBM MQ classes for JMS と IBM MQ classes for Java の両方を実行できないという制限がなくなります。 ただし、他の制限事項はすべて、従来どおり適用されます。

IBM MQ インストール済み環境の java/lib/OSGi サブディレクトリー、または Windows 上の java\lib\OSGi フォルダーにもインストールされるバンドル com.ibm.mq.osgi.java*version_number*.jar は、IBM MQ classes for Java の一部です。 このバンドルは、既に IBM MQ classes for JMS がロードされている OSGi ランタイム環境にロードしてはなりません。

IBM MQ classes for JMS 用の OSGi バンドルは、OSGi リリース 4 仕様に合わせて記述されています。 OSGi リリース 3 環境では機能しません。

OSGi ランタイム環境が必要な DLL ファイルまたは共用ライブラリーを検出できるように、システム・パスまたはライブラリー・パスを正しく設定する必要があります。

IBM MQ classes for JMS 用の OSGi バンドルを使用する場合、一時トピックは機能しません。 さらに、複数のクラス・ローダー環境 (OSGi など) でクラスをロードする際に固有の問題があるため、Java で作成されたチャネル出口クラスはサポートされません。 ユーザー・バンドルは IBM MQ classes for JMS バンドルを認識できますが、IBM MQ classes for JMS バンドルはユーザー・バンドルを認識できません。 結果として、IBM MQ classes for JMS バンドル内で使用されるクラス・ローダーは、ユーザー・バンドル内のチャネル出口クラスをロードできません。

OSGi の詳細については、OSGi Alliance Web サイトをご覧ください。

z/OS MQ Adv. VUE z/OS 上で実行されているバッチ・アプリケーションへの JMS/Jakarta Messaging クライアント接続

特定の条件下では、 z/OS 上の IBM MQ classes for JMS/Jakarta Messaging アプリケーションは、クライアント接続を使用して z/OS 上のキュー・マネージャーに接続できます。 クライアント接続を使用すると、IBM MQ トポロジーを単純化できます。

クライアント接続を使用することにより、 IBM MQ classes for JMS/Jakarta Messaging アプリケーションがバッチ環境で実行されていて、以下のいずれかの条件が該当する場合、アプリケーションはリモート z/OS キュー・マネージャーに接続できます。

- V 9.3.4 LTS IBM MQ classes for JMS/Jakarta Messaging コードは、 IBM MQ 9.3.4 以降、または APAR PH56722 が適用された Long Term Support にあります。 キュー・マネージャーは、サポートされている任意のバージョンにすることができます。
- 接続先のキュー・マネージャーは IBM MQ Advanced for z/OS Value Unit Edition ライセンスを使用して実行されているため、 **ADVCAP** パラメーターは ENABLED に設定されています。 キュー・マネージャーは、サポートされている任意のバージョンにすることができます。

IBM MQ Advanced for z/OS Value Unit Edition について詳しくは、 IBM MQ 製品 ID およびエクスポート情報を参照してください。

ADVCAP について詳しくは、DISPLAY QMGR を参照してください。 **QMGRPROD** について詳しくは、START QMGR を参照してください。

サポートされる環境はバッチのみであることに注意してください。 JMS/Jakarta Messaging (CICS の場合) または JMS/Jakarta Messaging (IMS の場合) はサポートされません。

z/OS 上の IBM MQ classes for JMS/Jakarta Messaging アプリケーションは、 z/OS に設定されていないキュー・マネージャーに接続するためにクライアント・モード接続を使用することはできません。

z/OS 上の IBM MQ classes for JMS/Jakarta Messaging アプリケーションがクライアント・モードを使用して接続しようとしたが、それが許可されていない場合、例外メッセージ JMSFMQ0005 が発行されます。

Advanced Message Security (AMS) サポート

IBM MQ classes for JMS/Jakarta Messaging クライアント・アプリケーションは、このトピックで前述した条件に従って、リモート z/OS キュー・マネージャーに接続するときに AMS を使用できます。

この方法で AMS を使用するには、クライアント・アプリケーションは `keystore.conf` で鍵ストア・タイプ `jceracfks` を使用する必要があります。ここで、

- プロパティー名の接頭部は `jceracfks` で、この名前接頭部では大/小文字は区別されません。
- 鍵ストアは RACF 鍵リングです。
- パスワードは必須ではなく、無視されます。 RACF 鍵リングではパスワードが使用されないためです。
- プロバイダーを指定する場合、そのプロバイダーは `IBMJCE` でなければなりません。

AMS で `jceracfks` を使用する場合、鍵ストアは `safkeyring://`*`user`*`/`*`keyring`* の形式でなければなりません。ここで、

- `safkeyring` はリテラルで、この名前では大/小文字は区別されません。
- *`user`* は、鍵リングを所有する RACF ユーザー ID です。
- *`keyring`* は RACF 鍵リングの名前で、鍵リングの名前には大/小文字の区別があります。

以下の例では、ユーザー `JOHNDOE` の標準 AMS 鍵リングを使用しています。

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

関連概念

372 ページの『z/OS で実行されているバッチ・アプリケーションへの Java クライアント接続』
特定の条件下では、 z/OS 上の IBM MQ classes for Java アプリケーションは、クライアント接続を使用して z/OS 上のキュー・マネージャーに接続できます。 クライアント接続を使用すると、 IBM MQ トポロジーを単純化できます。

IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging を個別に入手する

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用してアプリケーションを実行するために必要なライブラリーおよびユーティリティーは、 Fix Central からダウンロードした自己解凍型 JAR ファイルで入手できます。 これは、例えばソフトウェア管理ツールへのデプロイメントやスタンドアロン・クライアント・アプリケーションで使用するために、これらのファイルのみを取得する場合に行います。

始める前に

この作業を開始する前に、Java runtime environment (JRE) がマシンにインストールされていて、JRE がシステム・パスに追加されていることを確認してください。

このインストール処理で使用する Java インストーラーは、root または他の特定のユーザーとして実行する必要はありません。 唯一の要件は、それを実行するユーザーに、ファイルを入れるディレクトリーへの書き込み権限があることです。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

このタスクについて

自己解凍型 JAR ファイルは、ダウンロードのサイズと、抽出の実行にかかる時間を最小化するために使用されます。 この JAR ファイルの正確な内容、およびこの JAR ファイルがファイルを抽出するサブディレクトリーは、 IBM MQ のバージョンによって異なります。

自己解凍型 JAR ファイルを実行すると、 IBM MQ のご使用条件が表示されます。これに同意する必要があります。 また、抽出の親ディレクトリーを変更することもできます。

IBM MQ 9.3 の場合、自己解凍型 JAR ファイルは以下のディレクトリー構造にファイルを抽出します。

wmq/JavaEE
IBM MQ リソース・アダプター EAR ファイルおよび RAR ファイル。

JMS 2.0 以下のファイルは、 JMS 2.0 オブジェクトおよび JMS 1.1 オブジェクトで使用するためのものです。

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Jakarta Messaging 3.0 オブジェクトで使用するための同等のセットもあります。

- `wmq.jakarta.jmsra.ivt.ear` インストール検査テスト・ファイルが入っています。
- `wmq.jakarta.jmsra.rar` リソース・アダプター・ファイルが入っています。

wmq/JavaSE

wmq/JavaSE/bin
JMSAdmin および **JMS30Admin** ツール。 JMS オブジェクトまたは Jakarta Messaging オブジェクトを表す JNDI エンティティーを定義するために使用されます。

JMS 2.0 以下のファイルは、JMS 2.0 オブジェクトおよび JMS 1.1 オブジェクトで使用するためのものです。

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Jakarta Messaging 3.0 オブジェクトで使用するための同等のセットもあります。

- `JMS30Admin.bat` Windows でツールを開始するために使用されるファイル。
- `JMS30Admin` Linux および UNIX プラットフォームでツールを開始するために使用されるスクリプト。
- `JMS30Admin.config` このツールのサンプル構成ファイル。

注:

- **JMSAdmin** ツールが自己解凍型 JAR ファイルに追加される前は、このディレクトリー内のファイルは親ディレクトリー `wmq/JavaSE` にありました。
- 自己解凍型 JAR ファイルを使用してインストールされたクライアントは、 **JMSAdmin** または **JMS30Admin** ツールを使用して、ファイル・システム・コンテキスト (`.bindings` ファイル) 内に Java メッセージング管理対象オブジェクトを作成できます。 クライアントは、これらの管理オブジェクトを検索および使用することもできます。
- JMS 2.0 JMS 2.0 オブジェクトおよび JMS 1.1 オブジェクトで使用する **JMSAdmin** ツールが、 IBM MQ 9.2.0 Fix Pack 2 および IBM MQ 9.2.2 の自己解凍型 JAR ファイルに追加されました。
- JM 3.0 Jakarta Messaging 3.0 オブジェクトで使用するための **JMS30Admin** ツールが、自己解凍型 JAR ファイル (IBM MQ 9.3.0) に追加されました。

wmq/JavaSE/lib

Advanced Message Security は、以下のオープン・ソース Bouncy Castle パッケージを使用して、暗号メッセージ構文 (CMS) をサポートします。 AMS を使用した非 IBM JRE のサポートを参照してください。

V 9.3.5 IBM MQ 9.3.5 からの Continuous Delivery の場合:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

LTS IBM MQ 9.3.5 より前の Long Term Support および Continuous Delivery の場合:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- `bcutil-jdk15on.jar`

以下のファイルにはそれぞれ、特定の JMS または Jakarta Messaging レベルのクラスが含まれています。

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 および JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

その他の前提条件 JAR ファイル:

- V 9.3.3 Removed `com.ibm.mq.traceControl.jar` IBM MQ classes for JMS アプリケーションのトレースを動的に制御するために使用されます。

- `fscontext.jar` アプリケーションがファイル・システム・コンテキストを使用して JNDI 検索を実行する場合は必須です。
- V 9.3.3 `jackson-annotations.jar`、 `jackson-core.jar`、 `jackson-databind.jar`: キュー・マネージャーへのセキュア TLS 接続を作成するときに CipherSuite と CipherSpec のマッピングを実行するために使用されるクラスが含まれています。
- JM 3.0 `jakarta.jms-api.jar` Jakarta Messaging 3.0 インターフェースおよび例外定義が含まれています。
- JMS 2.0 `jms.jar` JMS 2.0 インターフェースおよび例外定義が含まれています。
- `org.json.jar` IBM MQ classes for JMS が JSON 形式の CCDT ファイルを解釈できるようにするクラスが含まれます。
- `providerutil.jar` アプリケーションがファイル・システム・コンテキストを使用して JNDI 検索を実行する場合は必須です。

注: Stabilized `com.ibm.mq.allclient.jar` および `com.ibm.mq.jakarta.client.jar` には、両方とも IBM MQ classes for Java のコピーが含まれています。 ただし、 IBM MQ 9.0 では、これらのクラスは、 IBM MQ 8.0 で出荷されたレベルで機能的に安定化されたものとして宣言されます。 IBM MQ 9.0 での非推奨、固定化、および削除を参照してください。

wmq/OSGi

IBM MQ OSGi クライアント・バンドル:

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

ここで、*V.R.M.F* は、バージョン、リリース、変更、およびフィックスパック番号です。

手順

1. Fix Central から IBM MQ Java / JMS クライアント JAR ファイルをダウンロードします。
   a) こちらのリンク IBM MQ Java / JMS クライアントをクリックします。
   b) 表示された選択可能なフィックスのリストから、ご使用のバージョンの IBM MQ 用のクライアントを見つけます。
   以下に例を示します。

   ```
   release level: 9.3.0.0-IBM-MQ-Install-Java-All
   Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
   ```

   次に、クライアントのファイル名をクリックし、ダウンロード・プロセスに従います。
2. ファイルをダウンロードしたディレクトリーから抽出を開始します。
   抽出を開始するには、以下の形式でコマンドを入力します。

   ```
   java -jar V.R.M.F-IBM-MQ-Install-Java-All.jar
   ```

   ここで、 *V.R.M.F* は製品のバージョン番号 (例: `9.3.0.0`) であり、 *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` は Fix Central からダウンロードしたファイルの名前です。
   例えば、 IBM MQ 9.3.0 リリースの JMS クライアントを抽出するには、以下のコマンドを使用します。

   ```
   java -jar 9.3.0.0-IBM-MQ-Install-Java-All.jar
   ```

注: このインストールを実行するには、JRE がマシンにインストールされ、システム・パスに追加されている必要があります。

コマンドを入力すると、以下の情報が表示されます。

```
IBM MQ V9.3 を使用、抽出、またはインストールするには、事前に以下を受け入れる必要があります。
条件 1。 IBM 評価のためのご使用条件
プログラム 2。 IBM プログラムのご使用条件および追加情報
ライセンス情報。 以下の使用条件をよくお読みください。

ご使用条件は、以下を使用して個別に表示できます。
--viewLicense 契約オプション。

ここでライセンス条項を表示する場合は Enter を、スキップする場合は「x」を押してください。
```

3. ライセンス条項を確認して受け入れます。
   a) ライセンスを表示するには、Enter を押します。
   あるいは、x を押してライセンスの表示をスキップします。
   ライセンスが表示された後、または x を押すとすぐに、以下のメッセージが表示されます。

   ```
   追加のライセンス情報は、以下を使用して個別に表示できます。
   --viewLicense 情報オプション。

   ここで追加のライセンス情報を表示する場合は Enter を、スキップする場合は「x」を押してください。
   ```

   b) 追加のライセンス条項を表示するには、Enter を押します。
   あるいは、x を押して追加のライセンス条項の表示をスキップします。
   追加のライセンス条項が表示された後、または即時に x を押すと、以下のメッセージが表示されます。

   ```
   以下の「同意する」オプションを選択すると、以下の条件に同意したことになります。
   ご使用条件および IBM 以外の条項 (該当する場合)。 そうでない場合は、
   同意する場合は、「同意しない」を選択します。

   [1] 同意する、または [2] 同意しないを選択
   ```

   c) 使用条件を受け入れて、インストール・ディレクトリーの選択に進むには、1 を選択します。
   あるいは、2 を選択して、インストールを即時に終了します。
   1 を選択すると、次のようなメッセージが表示されます。

   ```
   製品ファイルのディレクトリーを入力するか、ブランクのままにしてデフォルト値を受け入れます。
   デフォルトのターゲット・ディレクトリーは H: ¥ downloads です。

   製品ファイルのターゲット・ディレクトリー?
   ```

4. 抽出の親ディレクトリーを指定します。

   デフォルトのロケーションは現行ディレクトリーです。

   - 製品ファイルをデフォルトの場所に解凍する場合は、値を指定せずに Enter キーを押します。
   - プロダクト・ファイルを別の場所に抽出する場合は、ファイルを抽出するディレクトリーの名前を指定してから、Enter キーを押して抽出を開始します。

   指定するディレクトリー名が既に存在していてはなりません。存在していないと、抽出の開始時にエラーが報告され、ファイルはインストールされません。
   まだ存在していない場合は、指定されたディレクトリーが作成され、プログラム・ファイルがこのディレクトリーに抽出されます。 インストール時に、指定した親ディレクトリー内に wmq という名前の新規ディレクトリーが作成されます。

   3 つのサブディレクトリー JavaEE、JavaSE、および OSGi は、以下の内容を含む wmq ディレクトリーに作成されます。

   JavaEE

   JM 3.0 wmq.jakarta.jmsra.ivt.ear

   JM 3.0 wmq.jakarta.jmsra.rar

   JMS 2.0 wmq.jmsra.ivt.ear

JMS 2.0 wmq.jmsra.rar

JavaSE

このディレクトリーには、以下のサブディレクトリーとファイルが含まれています。

JavaSE/lib

V 9.3.5 bcpkix-jdk18on.jar

LTS bcpkix-jdk15on.jar

V 9.3.5 bcprov-jdk18on.jar

LTS bcprov-jdk15on.jar

V 9.3.5 bcutil-jdk18on.jar

LTS bcutil-jdk15on.jar

JMS 2.0 com.ibm.mq.allclient.jar

JM 3.0 com.ibm.mq.jakarta.client.jar

V 9.3.3 Removed com.ibm.mq.traceControl.jar

fscontext.jar

V 9.3.3 jackson-annotations.jar

V 9.3.3 jackson-core.jar

V 9.3.3 jackson-databind.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

抽出が完了すると、以下の例に示すような確認メッセージが表示されます。

```
ファイルを H: ¥ downloads¥ wmq に抽出しています
すべての製品ファイルを正常に抽出しました。
```

IBM MQ classes for JMS/Jakarta Messaging での許可リスティング

Java オブジェクト・シリアライゼーションおよびデシリアライゼーション・メカニズムは、潜在的なセキュリティー・リスクとして識別されています。 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging での許可リスティングにより、一部のシリアライゼーション・リスクに対する保護が提供されます。

このタスクについて

デシリアライゼーションは任意の Java オブジェクトをインスタンス化するので、送信された悪意のあるデータによってさまざまな問題が発生する可能性があります。 このため、Java オブジェクトのシリアライゼーションおよびデシリアライゼーションのメカニズムは、潜在的なセキュリティー・リスクとして見なされています。 シリアライゼーションの注目すべき適用例の 1 つとして、 Jakarta Messaging 3.0 および Java Message Service 2.0 ObjectMessages では、シリアライゼーションを使用して任意のオブジェクトをカプセル化および転送します。

シリアライゼーションの許可リスティングを使用すると、シリアライゼーションがもたらすリスクの一部を軽減できる可能性があります。 許可リスティングは、ObjectMessages でカプセル化と抽出を実行できるクラスを明示的に指定することで、シリアライゼーションによるリスクから、ある程度保護することができます。

関連概念

108 ページの『Java security manager での IBM MQ classes for JMS アプリケーションの実行』
IBM MQ classes for JMS は、 Java security manager を有効にして実行できます。 Java security manager を有効にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java Virtual Machine (JVM) を構成する必要があります。

許可リスティングの概念

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、 JMS ObjectMessage インターフェースの実装におけるクラスの許可リスト登録がサポートされています。 これにより、Java オブジェクトのシリアライゼーションおよびデシリアライゼーション・メカニズムに関連する可能性があるいくつかのセキュリティー・リスクを軽減することができます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging での許可リスティング

重要 :

可能な限り、ホワイトリスト という用語は、許可リスト という用語に置き換えられました。 IBM MQ 9.0 以降のリリースでは、これには、このトピックで言及されているいくつかの Java システム・プロパティー名が含まれます。 既存の構成を変更する必要はありません。 以前のシステム・プロパティー名も引き続き有効です。

IBM MQ classes for JMS (JMS 2.0) V 9.3.0 V 9.3.0 および IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) は、 JMS ObjectMessage インターフェースの実装におけるクラスの許可リスト登録をサポートします。

- JMS 2.0 IBM MQ classes for JMS の場合、関連するプロパティー名は **com.ibm.mq.jms.allowlist.*** です。
- JM 3.0 IBM MQ classes for Jakarta Messaging の場合、関連するプロパティー名は **com.ibm.mq.jakarta.jms.allowlist.*** です。

許可リストには、ObjectMessage.setObject() でシリアライズできる Java クラスと、ObjectMessage.getObject() でデシリアライズできる Java クラスを定義します。

- JMS 2.0 許可リストに含まれていないクラスのインスタンスを ObjectMessage でシリアライズ/デシリアライズしようとすると、 java.io.InvalidClassException が原因の javax.jms.MessageFormatException がスローされます。
- JM 3.0 ObjectMessage を使用して、許可リストに含まれていないクラスのインスタンスをシリアライズまたはデシリアライズしようとすると、 jakarta.jms.MessageFormatException がスローされ、その原因は java.io.InvalidClassException になります。

許可リストの作成

重要 : IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、許可リストと一緒に配布することはできません。 ObjectMessages を使用して転送するクラスの選択は、アプリケーション設計上の選択であるため、IBM MQ が事前に設定できることではありません。

そのため、許可リスティングのメカニズムには以下の 2 つの動作モードが用意されています。

DISCOVERY
このモードでは、メカニズムは完全修飾クラス名のリストを生成します。ObjectMessages でのシリアライズ/デシリアライズが検出されたすべてのクラスが報告されます。

ENFORCEMENT
このモードでは、メカニズムは許可リスティングを適用します。許可リストに含まれていないクラスのシリアライズ/デシリアライズの試行は拒否されます。

このメカニズムを使用する場合には、最初に DISCOVERY モードで実行して現在シリアライズ/デシリアライズされているクラスのリストを収集し、 そのリストを確認してから、独自のホワイトリストを作成するための基礎として使用する必要があります。 そのリストを変更せずに使用して良い場合もありますが、そのように決定する前に、まずはリストを確認する必要があります。

許可リスティング・メカニズムの制御

許可リスティング・メカニズムを制御するために、以下の 3 つのシステム・プロパティーが用意されています。

com.ibm.mq.jms.allowlist (JMS 2.0) および com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
このプロパティーは、次のいずれかの方法で指定できます。

- ファイル URI フォーマット (つまり `file:`で始まるもの) 内の許可リストを含むファイルのパス名。DISCOVERY モードの場合は、このファイルは許可リスティング・メカニズムによって書き込まれます。 このファイルが存在していてはなりません。 ファイルが存在していると、メカニズムはそのファイルを上書きするのではなく、例外をスローします。 ENFORCEMENT モードの場合は、このファイルは許可リスティング・メカニズムによって読み取られます。
- 許可リストを構成する、完全修飾クラス名をコンマで区切ったもの。

このプロパティーが設定されていない場合、許可リスト・メカニズムは非アクティブになります。

Java security manager を使用している場合は、IBM MQ classes for JMS JAR ファイルがこのファイルに対する読み取りおよび書き込みを行うアクセス権を持っていることを確認する必要があります。

com.ibm.mq.jms.allowlist.discover (JMS 2.0) および com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- このプロパティーが設定されていないか false に設定されている場合、許可リスト・メカニズムは ENFORCEMENT モードで実行されます。
- このプロパティーが true に設定されていて、許可リストがファイル URI として指定されている場合、許可リスト・メカニズムは DISCOVERY モードで実行されます。
- このプロパティーが true に設定されていて、許可リストがクラス名のリストとして指定されている場合、許可リスト・メカニズムは該当する例外をスローします。
- このプロパティーが true に設定されていて、許可リストが com.ibm.mq.jms.allowlist または com.ibm.mq.jakarta.jms.allowlist プロパティーを使用して指定されていない場合、許可リスト・メカニズムは非アクティブになります。
- このプロパティーが true に設定されていて、許可リスト・ファイルが既に存在する場合、許可リスト・メカニズムは java.io.InvalidClassException をスローし、エントリーはファイルに追加されません。

com.ibm.mq.jms.allowlist.mode (JMS 2.0) および com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)
このストリング・プロパティーは、以下の 3 つの方法のいずれでも指定できます。

- このプロパティーが SERIALIZE に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.setObject() メソッドでのみ許可リストの妥当性検査が行われます。
- このプロパティーが DESERIALIZE に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.getObject() メソッドでのみ許可リストの妥当性検査が行われます。
- このプロパティーが設定されていない場合、または他の値に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.getObject() メソッドと ObjectMessage.setObject() メソッドの両方で許可リストの妥当性検査が行われます。

許可リスト・ファイルの形式

許可リスト・ファイルの形式の主な特徴を次に示します。

- 許可リストファイルは、プラットフォームに適切な行の終わりを含むデフォルトのプラットフォーム・ファイル・エンコーディングにあります。

  注: 許可リスト・ファイルを使用する場合、ファイルの読み書きには、必ず JVM のデフォルトのファイル・エンコードが使用されます。

  これは、許可リスト・ファイルが次のようにして生成される場合には問題ありません。

  - z/OS で実行されるスタンドアロン・アプリケーションによって生成され、同じく z/OS で実行される他のスタンドアロン・アプリケーションによって使用される場合。
  - WebSphere Application Server の内部で実行される (プラットフォームは不問) アプリケーションによって生成され、別の WebSphere Application Server インスタンスによって使用される場合。
  - Multi IBM MQ for Multiplatforms で実行されるスタンドアロン・アプリケーションによって生成され、IBM MQ for Multiplatforms で実行される他のスタンドアロン・アプリケーションか任意のプラットフォームの WebSphere Application Server の内部で実行されるアプリケーションによって使用される場合。

  ただし、 WebSphere Application Server は ASCII を使用し、スタンドアロンの JVM は EBCDIC を使用するため、許可リスト・ファイルが次のようにして生成される場合には、ファイル・エンコードに関する問題が発生します。

  - z/OS で生成され、z/OS 以外のプラットフォームで実行されるスタンドアロン・アプリケーションか WebSphere Application Server によって使用される場合。
  - WebSphere Application Server、または z/OS 以外のプラットフォームで実行されるスタンドアロン・アプリケーションによって生成され、z/OS のスタンドアロン・アプリケーションによって使用される場合。
- 空でない各行には 1 つの完全修飾クラス名が含まれます。 空の行は無視されます。
- コメントを含めることができます。'#' 文字からその行末までの内容はすべて無視されます。
- 以下のような、ごく基本的なワイルドカード・メカニズムを使用できます。
  - '*' はクラス名の**最後の**要素として使用できます。
  - '*' はクラス名の**単一の**要素と一致します。つまりクラスとは一致しますがパッケージの部分とは一致しません。

  そのため、`com.ibm.mq.*` は `com.ibm.mq.MQMessage` と一致しますが、`com.ibm.mq.jmqi.remote.api.RemoteFAP` とは一致しません。

  ワイルドカードは、デフォルト・パッケージ内のクラス、つまり明示的なパッケージ名のないクラスに対しては機能しないので、"*" のクラス名は拒否されます。
- 形式が正しくない許可リスト・ファイル (例えば、ワイルドカードが最後の要素でない `com.ibm.mq.*.Message` などの項目を含むファイル) を使用すると、java.lang.IllegalArgumentException がスローされます。
- 空の許可リスト・ファイルを使用すると、ObjectMessage の使用が完全に無効になります。

コンマ区切りリストとしての許可リストの形式

コンマ区切りリストとしての許可リストでも、同じワイルドカード・メカニズムが使用可能です。

- '*' をコマンド行、シェル・スクリプト、またはバッチ・ファイルに指定すると、オペレーティング・システムによって展開される場合があるので、特別な処理が必要になります。
- '#' コメント文字を使用できるのは、ファイルを指定する場合のみです。 許可リストがクラス名のコンマ区切りリストとして指定されている場合、オペレーティング・システムまたはシェルがそれを処理しないと想定します。これは、多くの AIX and Linux シェルではデフォルトのコメント文字であるため、通常の文字として扱われます。

許可リスティングが行われるタイミング

許可リスティングは、アプリケーションが初めて ObjectMessage の setMessage() メソッドまたは getMessage() メソッドを実行したときに開始されます。

メカニズムが初期化されると、システム・プロパティーが評価され、許可リスト・ファイルが開かれ、ENFORCEMENT モードの場合は、許可リストに含まれているクラスのリストが読み込まれます。 この時点で、エントリーがアプリケーションの IBM MQ JMS ログ・ファイルに書き込まれます。

メカニズムが初期化されたら、そのパラメーターを変更することはできません。 初期化のタイミングはアプリケーションの動作に依存するため、簡単には予測できません。 したがって、システム・プロパティーの設定値と許可リスト・ファイルの内容は、アプリケーションの開始時から固定であると見なす必要があります。 結果が保証されないため、アプリケーションの実行中にプロパティーや許可リスト・ファイルの内容を変更しないでください。

考慮事項

Java シリアライゼーション・メカニズムに固有のリスクを軽減するための最良のアプローチは、ObjectMessage の代わりに JSON を使用するなど、別のデータ転送方式を検討することです。 Advanced Message Security (AMS) メカニズムを使用すると、メッセージの送信元が信頼できるソースであることが保証されるので、セキュリティーを高めることができます。

アプリケーションに Java security manager のメカニズムを使用する場合は、以下の権限を付与する必要があります。

- 使用する許可リスト・ファイルに対する FilePermission (ENFORCEMENT モードの場合は読み取り権限、DISCOVER モードの場合は書き込み権限を指定)
- JMS 2.0 **com.ibm.mq.jms.allowlist**、**com.ibm.mq.jms.allowlist.discover**、**com.ibm.mq.jms.allowlist.mode** の各プロパティーに対する PropertyPermission (読み取り)。
- JM 3.0 **com.ibm.mq.jakarta.jms.allowlist**、**com.ibm.mq.jakarta.jms.allowlist.discover**、**com.ibm.mq.jakarta.jms.allowlist.mode** の各プロパティーに対する PropertyPermission (読み取り)。

詳細情報

許可リストについて詳しくは、137 ページの『JMS または Jakarta Messaging 許可リストのセットアップと使用』および 139 ページの『WebSphere Application Server での許可リスティング』を参照してください。

関連概念

108 ページの『Java security manager での IBM MQ classes for JMS アプリケーションの実行』
IBM MQ classes for JMS は、 Java security manager を有効にして実行できます。 Java security manager を有効にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java Virtual Machine (JVM) を構成する必要があります。

JMS または *Jakarta Messaging* 許可リストのセットアップと使用

ここでは、許可リストがどのように機能するか、およびアプリケーションが処理できる ObjectMessages のタイプのリストを含む許可リスト・ファイルを生成するために IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging に含まれている機能を使用して許可リストをセットアップする方法について説明します。

始める前に

重要 :

可能な限り、ホワイトリスト という用語は、許可リスト という用語に置き換えられました。 IBM MQ 9.0 以降のリリースでは、これには、このトピックで言及されているいくつかの Java システム・プロパティー名が含まれます。 既存の構成を変更する必要はありません。 以前のシステム・プロパティー名も引き続き有効です。

このタスクを開始する前に、133 ページの『許可リスティングの概念』 を読み、その内容を理解しておいてください。

このタスクについて

JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピック内の JMS に対するこれ以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

許可リスティング機能を有効にすると、IBM MQ classes for JMS は次の方法でその機能を使用します。

- アプリケーションは、ObjectMessage を送信するとき、次のいずれかを呼び出してそれを作成することができます。
  - Session.createObjectMessage(Serializable) メソッド。その際、メッセージの中に入れるオブジェクトを渡します。
  - Session.createObjectMessage() メソッド。これにより、空の ObjectMessage が作成されます。その後 ObjectMessage.setObject(Serializable) を呼び出して、送信するオブジェクトを ObjectMessage の中に格納します。

  Session.createObjectMessage(Serializable) メソッドまたは ObjectMessage.setObject(Serializable) メソッドのいずれかが呼び出されると、JMS のクラスは、渡されたオブジェクトが許可リストに含まれているタイプであるかどうかを検査します。

  そこに含まれているタイプであれば、オブジェクトは直列化され、ObjectMessage 内に格納されます。ただし、そのオブジェクトが、許可リストにないタイプの場合、IBM MQ classes for JMS はメッセージを含む JMSException をスローします。

  ```
  JMSCC0052: オブジェクトのシリアライズ中に例外が発生しました:
  'java.io.InvalidClassException: < object class>; クラスをシリアライズできません
  許可リスト「< allowlist>」に含まれていないため、非直列化されました。
  ```

  アプリケーションに戻ります。

  重要 : Session.createObjectMessage(Serializable) メソッドから例外がスローされた場合、ObjectMessage は作成されません。 同様に、ObjectMessage.setObject(Serializable) メソッドから JMSException がスローされた場合、オブジェクトは ObjectMessage に追加されません。
- アプリケーションは、ObjectMessage を受信すると、メソッド ObjectMessage.getObject() を呼び出して、その中に入っているオブジェクトを取得します。 このメソッドが呼び出されると、IBM MQ classes for JMS は、ObjectMessage の中に含まれるオブジェクトのタイプを検査して、そのオブジェクトのタイプが許可リストに指定されたタイプであるかどうかを確認します。

  そうである場合、オブジェクトは非直列化され、アプリケーションに返されます。 ただし、そのオブジェクトが、許可リストにないタイプの場合、IBM MQ classes for JMS はメッセージを含む JMSException をスローします。

  ```
  JMSCC0053: メッセージのデシリアライズ中に例外が発生しました:
  'java.io.InvalidClassException: < object class>; クラスは以下のいずれかではない可能性があります。
  に含まれていないため、直列化または非直列化
  許可リスト '< allowlist>'. '
  ```

アプリケーションに戻ります。

例えば、java.net.URI というタイプのオブジェクトを入れた ObjectMessage を送信するための次のコードがアプリケーションに含まれているとします。

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

許可リスティングが有効ではないため、アプリケーションはメッセージを必要な宛先に正常に渡すことができます。

`java.net.URL` という単一のエントリーを含む `C:\allowlist.txt` というファイルを作成し、 Java システム・プロパティーを設定してアプリケーションを再度開始する場合は、以下のようにします。

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

許可リスト機能は有効になります。 java.net.URI のタイプは許可リストで指定されているため、アプリケーションは引き続き、そのタイプのオブジェクトを入れた ObjectMessage を作成して送信することができます。

ただし、 `allowlist.txt` ファイルを変更して、ファイルに単一エントリー `java.util.Calendar` が含まれるようにした場合は、アプリケーションが以下を呼び出すときに、許可リスト機能が引き続き有効になっているため、以下のようになります。

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS は、許可リストを検査し、java.net.URI の項目が含まれていないことを検出します。

その結果、JMSCC0052 メッセージが記述された JMSException がスローされます。

同様に、次のコードを使用して ObjectMessages を受け取る別のアプリケーションがあるとします。

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

許可リスティングが有効になっていない場合、アプリケーションは、任意のタイプのオブジェクトを入れた ObjectMessages を受け取ることができます。 アプリケーションはその後、オブジェクトのタイプが java.net.URL であるかどうかを検査してから適切な処理を実行します。

今度は、次のように Java システム・プロパティーを設定してアプリケーションを開始するとします。

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

これを設定すると、許可リスティング機能は有効になります。 アプリケーションが次の呼び出しを行うとします。

```
Object messageBody = objectMessage.getObject();
```

この場合、ObjectMessage.getObject() メソッドはタイプが java.net.URL のオブジェクトのみを返します。

ObjectMessage の中に入っているオブジェクトがこのタイプではない場合、ObjectMessage.getObject() メソッドは JMSCC0053 メッセージが記述された JMSException をスローします。 その場合、アプリケーションはそのメッセージをどのように処理するかを決定する必要があります。例えば、メッセージをそのキュー・マネージャーの送達不能キューに移動することもできます。

アプリケーションは、ObjectMessage の中に入っているオブジェクトのタイプが java.net.URL である場合にのみ、正常にそのオブジェクトを返します。

手順

1. 以下の Java システム・プロパティーを指定して、ObjectMessages を処理するアプリケーションを実行します。

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   アプリケーションが実行されると、IBM MQ classes for JMS はアプリケーションが処理したタイプのオブジェクトを設定したファイルを作成します。
2. アプリケーションが一定期間にわたって ObjectMessage の代表的なサンプルを処理したら、アプリケーションを停止します。

   この時点で、許可リスト・ファイルには、アプリケーションが実行中に処理した ObjectMessage の中に入っているすべてのタイプのオブジェクトのリストが設定されています。

   アプリケーションを十分な時間実行した場合、ObjectMessage の中に入っている、アプリケーションが今後処理すると考えられるすべてのタイプのオブジェクトがこのリストに設定されます。
3. アプリケーションを、次のシステム・プロパティーを設定して再始動します。

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   これで許可リスティングが有効になり、許可リストにないタイプの ObjectMessage を IBM MQ classes for JMS が検出すると、JMSCC0052 メッセージまたは JMSCC0053 メッセージが記述された JMSException がスローされます。

WebSphere Application Server での許可リスティング

WebSphere Application Server で IBM MQ classes for JMS 許可リスティングを使用する方法。

重要 :

可能な限り、ホワイトリスト という用語は、許可リスト という用語に置き換えられました。 IBM MQ 9.0 以降のリリースでは、これには、このトピックで言及されているいくつかの Java システム・プロパティー名が含まれます。 既存の構成を変更する必要はありません。 以前のシステム・プロパティー名も引き続き有効です。

WebSphere Application Server のインストール済み環境に、許可リスティングをサポートするバージョンの IBM MQ リソース・アダプターが組み込まれていることを確認してください。

この 2 つの製品の使用について詳しくは、498 ページの『IBM MQ と WebSphere Application Server の併用』を参照してください。

IBM MQ 9.0.0 Fix Pack 1 以降には適切な機能が含まれています。

アプリケーション・サーバーが更新されると、次の Java システム・プロパティーを使用できます。

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

137 ページの『JMS または Jakarta Messaging 許可リストのセットアップと使用』を参照してください。

注 : 変更を有効にするには、 Java システム・プロパティーを、アプリケーション・サーバーの実行に使用される Java Virtual Machine 上の汎用 JVM 引数として設定し、アプリケーション・サーバーを再始動する必要があります。

詳細については、 「Java 仮想マシン設定」の「汎用 *JVM* 引数」を参照してください。

プロパティーを設定するには、 「プロセス定義」 の Java Virtual Machine ウィンドウに移動し、適切な引数を入力します。

次の設定を考えます。

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

アプリケーション・サーバーが許可リスト *youruserId_MyObject* を使用するようにします。 このタイプのオブジェクトのみがアプリケーション・サーバーによって処理されます。

次の設定を考えます。

```
-Dcom.ibm.mq.jms.allowlist.discover=true
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

ディスカバー・モードを使用するようにアプリケーション・サーバーを構成し、アプリケーション・サーバーが処理する JMS ObjectMessages の詳細をファイル `C:\allowlist.txt` に記録します。

次の設定を考えます。

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

アプリケーション・サーバーがファイル `C:/allowlist.txt` をロードするようにして、そのファイル内の情報を使用して許可リストを判別します。

関連概念

108 ページの『Java security manager での IBM MQ classes for JMS アプリケーションの実行』
IBM MQ classes for JMS は、 Java security manager を有効にして実行できます。 Java security manager を有効にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java Virtual Machine (JVM) を構成する必要があります。

IBM MQ classes for JMS の文字ストリング変換

IBM MQ classes for JMS では、文字ストリング変換に CharsetEncoders と CharsetDecoders を直接使用します。 文字ストリング変換のデフォルト動作は、2 つのシステム・プロパティーを使用して構成できます。マップできない文字を含むメッセージの処理は、UnmappableCharacterAction と置換バイトの設定用メッセージ・プロパティーを使用して構成できます。

IBM MQ 8.0 の前は、IBM MQ classes for JMS のストリング変換は、
`java.nio.charset.Charset.decode(ByteBuffer)` メソッドおよび
`Charset.encode(CharBuffer)` メソッドを呼び出すことによって行われていました。

このいずれかのメソッドを使用すると、誤った形式のデータまたは変換不能なデータに対してデフォルトの置換が行われます (REPLACE)。 この動作により、アプリケーションでのエラーが覆い隠され、予期しない文字 (? など) が変換データに現れることがあります。

IBM MQ 8.0 から、以前よりも効率的にそのような問題を検出するために、IBM MQ classes for JMS は CharsetEncoders と CharsetDecoders を直接使用して、誤った形式のデータおよび変換不能なデータの処理を明示的に構成します。 デフォルト動作は変更され、適切な `MQException` をスローして、その種の問題を `REPORT` するようになりました。

構成

UTF-16 (Java で使用される文字表現) から固有文字セット (UTF-8 など) への変換を *encoding* と呼ぶのに対し、反対方向への変換を *decoding* と呼びます。

デコードでは `CharsetDecoders` のデフォルト動作に従い、例外をスローしてエラーを報告します。

今回、エンコードとデコードの両方で 1 つの設定を使用して `java.nio.charset.CodingErrorAction` を指定し、エラー処理を制御します。 エンコード時に置換バイトを制御するためにもう 1 つ別の設定が使用されます。 デコード操作では、デフォルトの Java 置き換えストリングが使用されます。

IBM MQ classes for JMS における UnmappableCharacterAction と置換バイトの設定

IBM MQ 8.0 以降、UnmappableCharacterAction と置換バイトの設定に次の 2 つのプロパティーを使用できます。 適切な定数定義は、`com.ibm.msg.client.wmq.WMQConstants` にあります。

JMS_IBM_UNMAPPABLE_ACTION
エンコード操作またはデコード操作で文字をマップできないときに適用する `CodingErrorAction` を設定または取得します。

これは、以下のように CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString() として設定する必要があります。

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
エンコード操作で文字をマップできないときに適用する置換バイトを設定または取得します。

デフォルトの Java 置き換えストリングがデコード操作で使用されます。

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

JMS_IBM_UNMAPPABLE_ACTION プロパティーと JMS_IBM_UNMAPPABLE_REPLACEMENT プロパティーを宛先またはメッセージに設定できます。 メッセージに設定される値は、メッセージ送信の宛先に設定される値をオーバーライドします。

JMS_IBM_UNMAPPABLE_REPLACEMENT を 1 バイトとして設定する必要があります。

システム・デフォルトの設定用システム・プロパティー

IBM MQ 8.0 以降、文字ストリングの変換についてのデフォルト動作を構成するために使用可能な 2 つの Java システム・プロパティーがあります。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
エンコードおよびデコードの際に変換不能データに対して取られるアクションを指定します。 値は REPORT、REPLACE、または IGNORE です。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
エンコード操作で文字をマップできないときに適用する置換バイトを設定または取得します。デコード操作では、デフォルトの Java 置き換えストリングが使用されます。

Java 文字と固有バイト表現の間の混乱を避けるため、com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement は固有文字セットの置換バイトを表す 10 進数として指定する必要があります。

例えば、固有バイトとしての ? の 10 進値は、固有文字セットが ASCII ベース (ISO-8859-1 など) の場合は 63 であるのに対し、固有文字セットが EBCDIC の場合は 111 になります。

注 : MQMD オブジェクトまたは MQMessage オブジェクトに **unmappableAction** フィールドまたは **unMappableReplacement** フィールドが設定されている場合、それらのフィールドの値が Java システム・プロパティーよりも優先されることに留意してください。 したがって、必要に応じてメッセージごとに、Java システム・プロパティーで指定された値をオーバーライドすることが可能です。

関連概念
351 ページの『IBM MQ classes for Java の文字ストリング変換』
IBM MQ classes for Java では、文字ストリング変換に CharsetEncoders と CharsetDecoders を直接使用します。 文字ストリング変換のデフォルト動作は、2 つのシステム・プロパティーを使用して構成できます。マップできない文字を含むメッセージの処理は、com.ibm.mq.MQMD を使用して構成できます。

IBM MQ classes for JMS/Jakarta Messaging アプリケーションの作成

JMS モデルを簡単に紹介した後、このセクションでは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging アプリケーションの作成方法に関する詳細なガイダンスを提供します。

このタスクについて

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

関連概念
IBM MQ classes for Jakarta Messaging: 概要

JMS および Jakarta Messaging モデル

JMS および Jakarta Messaging モデルは、 Java アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットを定義します。 IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging は両方ともメッセージング・プロバイダーです。 これらは、 JMS オブジェクトと Jakarta Messaging オブジェクトが IBM MQ の概念にどのように関連するかを定義します。 JMS および Jakarta Messaging の指定は、特定の JMS および Jakarta Messaging オブジェクトが管理対象オブジェクトであることを想定しています。

JMS 2.0 IBM MQ 8.0 では、 JMS 標準の JMS 2.0 バージョンのサポートが追加されました。これにより、 JMS 1.1 から簡素化された API が導入され、従来の API も保持されます。

V 9.3.0 JM 3.0 V 9.3.0 IBM MQ 9.3.0 以降、新規アプリケーションの開発で Jakarta Messaging 3.0 がサポートされるようになりました。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで JMS 2.0 API と Jakarta Messaging 3.0 API の両方を使用することはサポートされていません。

注 : Jakarta Messaging 3.0 の場合、 JMS 仕様の制御は Oracle から Java Community Process に移動します。 ただし、 Oracle は「javax」名の制御を保持します。この名前は、 Java Community Process に移動されていない他の Java テクノロジーで使用されます。 そのため、 Jakarta Messaging 3.0 は JMS 2.0 と機能的に同等ですが、命名にはいくつかの違いがあります。

- バージョン 3.0 の正式な名前は、 Java Message Service ではなく Jakarta Messaging です。
- パッケージ名および定数名には、`javax` ではなく `jakarta` が接頭部として付けられます。 例えば、JMS 2.0 では、メッセージング・プロバイダーへの初期接続は `javax.jms.Connection` オブジェクトであり、 Jakarta Messaging 3.0 では `jakarta.jms.Connection` オブジェクトです。

JMS 2.0 javax.jms パッケージは JMS インターフェースを定義し、 JMS プロバイダーはこれらのインターフェースを特定のメッセージング製品用に実装します。 IBM MQ classes for JMS は、 IBM MQ 用の JMS インターフェースを実装する JMS プロバイダーです。

JM 3.0 jakarta.jms パッケージは Jakarta Messaging インターフェースを定義し、Jakarta Messaging プロバイダーはこれらのインターフェースを特定のメッセージング製品用に実装します。 IBM MQ classes for Jakarta Messaging は、 IBM MQ 用の Jakarta Messaging インターフェースを実装する Jakarta Messaging プロバイダーです。

JMS と Jakarta Messaging は多くの共通点を共有しているため、このトピック内の JMS に対するこれ以降の参照は、両方の参照と見なすことができます。 必要に応じて、差異が強調表示されます。

簡易 API

JMS 2.0 では、単純化された API が導入されました。また、 JMS 1.1 からのドメイン固有およびドメイン独立のインターフェースも保持されます。 簡易 API では、メッセージの送受信に必要なオブジェクトの数が減り、次のインターフェースで構成されます。

ConnectionFactory
ConnectionFactory は、接続を作成するために JMS クライアントが使用する管理オブジェクトです。 このインターフェースはクラシック API でも使用されます。

JMSContext
このオブジェクトは、クラシック API の接続オブジェクトとセッション・オブジェクトを結合します。 JMSContext オブジェクトは、他の JMSContext オブジェクトから作成でき、基礎接続が複製されます。

JMS プロデューサー
JMSProducer は、JMSContext によって作成され、キューまたはトピックにメッセージを送信するために使用されます。 JMSProducer オブジェクトによって、メッセージの送信に必要とされるオブジェクトの作成が発生します。

JMS コンシューマー
JMSConsumer は、JMSContext によって作成され、トピックまたはキューからのメッセージの受信に使用されます。

簡易 API には、様々な効果があります。

- JMSContext オブジェクトは、常に自動的に基礎接続を開始します。
- JMSProducer と JMSConsumer は、メッセージ・オブジェクト全体を取得しなくても、メッセージの getBody メソッドを使用してメッセージ本文を直接処理できるようになりました。
- メッセージのプロパティーは、「本文」、つまりメッセージのコンテンツを送信する前に、メソッド・チェーニングを使用して JMSProducer オブジェクトに設定できます。 JMSProducer は、メッセージの送信に必要なすべてのオブジェクトの作成を処理します。 JMS 2.0 を使用して、次のようにプロパティーを設定して、メッセージを送信できます。

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 では、メッセージを複数のコンシューマー間で共有できる共有サブスクリプションも導入されました。 すべての JMS 1.1 のサブスクリプションは、非共有サブスクリプションとして処理されます。

クラシック API

次のリストは、クラシック API の主な JMS インターフェースを要約しています。

Destination
Destination は、アプリケーションがメッセージを送信する場所、またはアプリケーションが受信するメッセージの送信元、あるいはその両方です。

ConnectionFactory
ConnectionFactory オブジェクトは、接続の構成プロパティーのセットをカプセル化します。 アプリケーションは、接続ファクトリーを使用して接続を作成します。

接続
Connection オブジェクトは、メッセージング・サーバーに対するアプリケーションのアクティブな接続をカプセル化します。 アプリケーションは、接続を使用してセッションを作成します。

Session
Session は、メッセージを送受信する単一スレッド化されたコンテキストです。 アプリケーションは、Session を使用してメッセージ、メッセージ・プロデューサー、およびメッセージ・コンシューマーを作成します。 セッションは、トランザクション化しても、トランザクション化しなくてもかまいません。

メッセージ
Message オブジェクトは、アプリケーションが送信または受信するメッセージをカプセル化します。

MessageProducer
アプリケーションがメッセージ・プロデューサーを使用して宛先にメッセージを送信します。

MessageConsumer
アプリケーションがメッセージ・コンシューマーを使用して宛先に送信されたメッセージを受信します。

144 ページの図 9 は、これらのオブジェクトとその関係を示します。

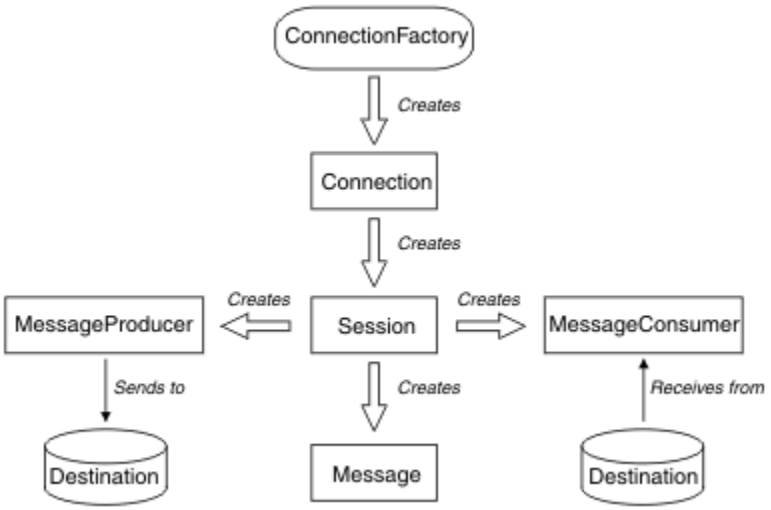


図 *9. JMS* オブジェクトとその関係

Destination、ConnectionFactory、または Connection オブジェクトは、マルチスレッド・アプリケーションの異なるスレッドによって並行して使用できますが、Session、MessageProducer、または MessageConsumer オブジェクトは、異なるスレッドによって並行して使用することはできません。 Session、MessageProducer、または MessageConsumer オブジェクトが並行して使用されないようにする最も簡単な方法は、スレッドごとに別の Session オブジェクトを作成することです。

メッセージ・ドメイン

JMS は、次の 2 つのスタイルのメッセージングをサポートします。

- Point-to-Point メッセージング
- パブリッシュ/サブスクライブ・メッセージング

メッセージングのこれらのスタイルは、メッセージ・ドメイン とも呼ばれ、1 つのアプリケーションで両方のスタイルのメッセージングを結合することができます。 Point-to-Point ドメインの場合、宛先はキューであり、パブリッシュ/サブスクライブ・ドメインの場合、宛先はトピックです。

JMS 1.1 の以前のバージョンの JMS では、Point-to-Point ドメインのプログラミングはインターフェースとメソッドの 1 つのセット使用され、パブリッシュ/サブスクライブ・ドメインのプログラミングは別のセットを使用します。 2 つのセットは似ていますが、別のものです。 JMS 1.1 以降、両方のメッセージ・ドメインをサポートするインターフェースとメソッドの共通のセットを使用できます。 共通のインターフェースは、メッセージ・ドメインごとにドメイン非依存のビューを提供します。 144 ページの表 15 に、JMS のドメイン非依存インターフェースと対応するドメイン固有インターフェースをリストします。

表 *15. JMS* のドメイン非依存インターフェースとドメイン固有インターフェース

ドメイン非依存インターフェース	Point-to-Point ドメインのドメイン固有インターフェース	パブリッシュ/サブスクライブ・ドメインのドメイン固有インターフェース
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
接続	QueueConnection	TopicConnection
Destination	キュー	トピック
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 は、以前の JMS 1.1 ドメイン固有インターフェースと、JMS 2.0 の単純化された API の両方をサポートします。 そのため、 IBM MQ classes for JMS 2.0 を使用して、既存のアプリケーションを保守することができます。これには、既存のアプリケーションでの新機能の開発も含まれます。

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 は、同じインターフェースの Jakarta Messaging バージョンをサポートしており、新しいアプリケーション開発に推奨されます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、 JMS オブジェクトは、以下の方法で IBM MQ の概念に関連しています。

- Connection オブジェクトには、接続の作成に使用された接続ファクトリーのプロパティーから派生したプロパティーがあります。 これらのプロパティーは、アプリケーションがキュー・マネージャーに接続する方法を制御します。 これらのプロパティーの例はキュー・マネージャーの名前であり、クライアント・モードのキュー・マネージャーに接続するアプリケーションでは、キュー・マネージャーが動作しているシステムのホスト名または IP アドレスです。
- Session オブジェクトは、IBM MQ 接続ハンドルをカプセル化するため、セッションのトランザクションの範囲を定義します。
- MessageProducer オブジェクトと MessageConsumer オブジェクトは、それぞれ IBM MQ オブジェクト・ハンドルをカプセル化します。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用する場合は、IBM MQ の通常の規則がすべて適用されます。 特に、アプリケーションはリモート・キューにメッセージを送信できますが、アプリケーションが接続しているキュー・マネージャーによって所有されるキューからしかメッセージを受信できないことに注意してください。

JMS 仕様は、ConnectionFactory オブジェクトと Destination オブジェクトが管理オブジェクトであると想定します。 管理者は管理オブジェクトを中央リポジトリーで作成して保守し、JMS アプリケーションは Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、Destination インターフェースの実装は Queue および Topic の抽象スーパークラスであるため、Destination のインスタンスは Queue オブジェクトまたは Topic オブジェクトのいずれかになります。 ドメイン非依存インターフェースは、キューまたはトピックを宛先として処理します。 MessageProducer オブジェクトまたは MessageConsumer オブジェクトのメッセージ・ドメインは、宛先がキューまたはトピックのいずれであるかによって決定されます。

したがって、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging では、以下のタイプのオブジェクトを管理対象オブジェクトにすることができます。

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- キュー
- トピック
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

関連概念

IBM MQ Java 言語インターフェース

207 ページの『接続ファクトリーおよび宛先の作成および構成』

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションは、管理対象オブジェクトとして Java Naming and Directory Interface (JNDI) 名前空間から、 IBM JMS 拡張機能を使用して、または IBM MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先を作成できます。 また、アプリケーションは IBM JMS 拡張機能または IBM MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先のプロパティーを設定できます。

JMS メッセージ

JMS メッセージは、ヘッダー、プロパティー、および本体で構成されます。 JMS では、5 つのタイプのメッセージ本体を定義します。

JMS メッセージは、以下の部分から構成されています。

ヘッダー
すべてのメッセージは、同じヘッダー・フィールドのセットをサポートします。 ヘッダー・フィールドには、メッセージを識別し、経路指定するために、クライアントとプロバイダーの両方によって使用される値が含まれています。

プロパティー
各メッセージには、アプリケーション定義のプロパティー値をサポートする組み込み機能が含まれています。 プロパティーは、アプリケーション定義のメッセージをフィルターに掛けるための効果的なメカニズムを提供します。

Body
JMS では、現在使用されているメッセージング・スタイルのほとんどを網羅する次の 5 つのタイプのメッセージ本体が定義されています。

ストリーム
Java プリミティブ値のストリーム。 ストリームが満たされると、順次に読み取られます。

マップ
名前と値のペア (名前はストリングで、値は Java プリミティブ・タイプ) のセットです。 エントリーには、順次アクセスか、または名前によるランダムでのアクセスを実行できます。 エントリーの順序は定義されていません。

テキスト
java.lang.String を含むメッセージ。

オブジェクト
シリアライズ可能 Java オブジェクトを含むメッセージ。

バイト
非解釈バイトのストリーム。 このメッセージ・タイプは、既存のメッセージ形式と一致するように本体を事実上エンコードするためのものです。

JMSCorrelationID ヘッダー・フィールドは、1 つのメッセージを別のメッセージとリンクするために使用されます。 JMSCorrelationID ヘッダー・フィールドは、通常、応答メッセージを要求メッセージとリンクします。 JMSCorrelationID は、プロバイダー特定のメッセージ ID、アプリケーション固有のストリング、またはプロバイダー・ネイティブの byte[] 値を保持できます。

JMS のメッセージ・セレクター

メッセージには、アプリケーション定義のプロパティー値を含めることができます。 アプリケーションは、メッセージ・セレクターを使用して JMS プロバイダーでメッセージをフィルター操作できます。

メッセージには、アプリケーション定義のプロパティー値をサポートするための組み込み機能が含まれています。 これは事実上、メッセージにアプリケーション固有のヘッダー・フィールドを追加するためのメカニズムを提供します。 プロパティーを利用すると、アプリケーションがメッセージ・セレクターを使用することで、アプリケーションの代わりに JMS プロバイダーが、アプリケーション固有の基準でメッセージを選択またはフィルター操作することができるようになります。 アプリケーション定義のプロパティーは、以下の規則に従わなければなりません。

- プロパティー名は、メッセージ・セレクター ID に関する規則に従わなければなりません。
- プロパティー値には、boolean、byte、short、int、long、float、double、および String を指定できます。
- JMSX および JMS_ name プレフィックスは予約されています。

プロパティー値は、メッセージを送信する前に設定されます。 クライアントがメッセージを受信すると、メッセージ・プロパティーは、読み取り専用になります。 この時点で、クライアントがプロパティーを設定しようとする場合、MessageNotWriteableException がスローされます。 clearProperties が呼び出される場合、プロパティーは読み取りにも書き込みにもなることができます。

プロパティー値は、メッセージの本体に値を複製する場合もあります。 JMS では、プロパティーに何が作成されるかについてポリシーを定義しません。 ただし、アプリケーション開発者は、JMS プロバイダーが

通常、メッセージのプロパティー内のデータよりも効率的にメッセージ本体内のデータを処理できる点に留意してください。 最適なパフォーマンスを得るために、アプリケーションは、メッセージのヘッダーをカスタマイズする必要が生じた場合にのみ、メッセージ・プロパティーを使用してください。 これを行う主な理由は、カスタマイズされたメッセージ選択をサポートするためです。

JMS メッセージ・セレクターを利用すると、クライアントはメッセージ・ヘッダーを使用して、対象となるメッセージを指定できます。 ヘッダーがセレクターと一致するメッセージのみが送達されます。

メッセージ・セレクターは、メッセージ本体の値を参照することはできません。

メッセージのヘッダー・フィールドおよびプロパティー値がセレクター内の対応する ID に置換される際に、セレクターが true に評価された場合、メッセージ・セレクターはメッセージと一致します。

メッセージ・セレクターはストリングであり、その構文は SQL92 条件式構文のサブセットに基づいています。 メッセージ・セレクターが評価される順序は、優先順位内で左から右です。 括弧を使用してこの順序を変更できます。 定義済みのセレクター・リテラルおよび演算子名は、大文字でここに書き込まれます。 ただし、これらには大/小文字の区別がありません。

メッセージ・セレクターの内容

メッセージ・セレクターには以下を指定できます。

- リテラル
  - ストリング・リテラルは、引用符で囲まれます。 二重引用符は引用符を表します。 例は、'literal' および 'literal''s' です。 Java ストリング・リテラルと同様に、これらは Unicode 文字エンコードを使用します。
  - 正確な数字リテラルは、57、-957、+62 など、小数点なしの数値です。 Java long の範囲内の数値がサポートされています。
  - 近似数値リテラルは、7E3 または -57.9E2 などの浮動小数における数値、または 7.、-95.7、または +6.2 などの小数部を持つ数値です。 Java double の範囲内の数値がサポートされています。
  - ブール・リテラルの TRUE および FALSE。
- ID:
  - ID は、Java 英字および Java 数字の無制限の長さシーケンスであり、そのうちの最初のものは、Java 英字である必要があります。 英字は、メソッド Character.isJavaLetter が true を戻す任意の文字です。これには _ と $ が含まれます。英字や数字は、メソッド Character isJavaLetterOrDigit が true を戻す任意の文字です。
  - ID は、名前 NULL、TRUE、または FALSE となることはできません。
  - ID は、NOT、AND、OR、BETWEEN、LIKE、IN、および IS となることはできません。
  - ID は、ヘッダー・フィールド参照またはプロパティー参照のいずれかです。
  - ID には、大/小文字の区別があります。
  - メッセージ・ヘッダー・フィールド参照は、以下のものに制限されます。
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID、および JMSType 値は、ヌルの場合があります。その場合は、ヌル値として扱われます。
  - JMSX で始まる名前は、JMS 定義のプロパティー名です。
  - JMS_で始まる名前は、プロバイダー固有のプロパティー名です。

  - JMS で始まらない名前は、アプリケーション固有のプロパティー名です。 メッセージ内に存在しないプロパティーへの参照がある場合、その値はヌルです。 存在する場合には、その値は、対応するプロパティー値です。
- 空白は、Java に定義されているのと同じで、スペース、水平タブ、改ページ、および行終了文字です。
- 式:
  - セレクターは、条件式です。 true に評価されるセレクターは一致し、false または unknown に評価されるセレクターは一致しません。
  - 演算式は、それ自体と、算術演算、ID (その値は数値リテラルとして扱われる)、および数値リテラルから構成されています。
  - 条件式は、それ自体と、比較演算、および論理演算から構成されています。
- 標準の括弧 () (式が評価される順序を設定する) がサポートされています。
- 論理演算子 (優先順位どおりに列挙): NOT、AND、OR。
- 比較演算子: =、>、>=、<、<=、<> (等しくない)。
  - 同じタイプの値のみを比較できます。 1 つの例外は、正確な数値と近似数値の比較が有効であることです。 (必要な型変換は、Java 数値プロモーションの規則によって定義されます。) 異なるタイプを比較する試みがある場合、セレクターは常に false です。
  - ストリングとブールの比較は、= および <> に制限されます。2 つのストリングは、同じ文字シーケンスを含んでいる場合にのみ等しくなります。
- 算術演算子 (優先順位どおりに列挙):
  - 単項 +、-。
  - *、/ (乗算および除算)。
  - +、- (加算および減算)。
  - ヌル値での算術演算はサポートされていません。 ヌル値での算術演算が試行される場合、完全セレクターは常に false です。
  - 算術演算は、Java 数値プロモーションを使用しなければなりません。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比較演算子:
  - Age BETWEEN 15 and 19 は、age >= 15 AND age <= 19 と同じです。
  - Age NOT BETWEEN 15 and 19 は、age < 15 OR age > 19 と同じです。
  - BETWEEN 演算の式のいずれかがヌルである場合、演算の値は false です。 NOT BETWEEN 演算の式のいずれかがヌルである場合、演算の値は true です。
- identifier [NOT] IN (string-literal1, string-literal2, ...) ID がストリング値または NULL 値を持つ比較演算子。
  - Country IN (' UK', 'US', 'France') は 'UK' の場合には true であり、'Peru' の場合には false です。 これは、式 (Country = ' UK') OR (Country = ' US') OR (Country = ' France') と同じです。
  - Country NOT IN (' UK', 'US', 'France') は 'UK' の場合には false であり、'Peru' の場合には true です。 これは、式 NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France')) と同じです。
  - IN または NOT IN 演算の ID がヌルである場合、演算の値は不明です。
- ID がストリング値を持つ identifier [NOT] LIKE pattern-value [ESCAPE escape-character] 比較演算子。 pattern-value はストリング・リテラルです。ここで、_ は単一文字を表しており、% は文字シーケンス (空のシーケンスを含む) を表しています。 その他のすべての文字はそれ自体を表しています。 任意の escape-character は単一の文字ストリング・リテラルです。この文字は、pattern-value 内の _ および % の特殊な意味をエスケープするために使用されます。
  - phone LIKE '12%3' は、123 および 12993 の場合には true で、1234 の場合には false です。
  - word LIKE 'l_se' は、lose の場合には true で、loose の場合には false です。
  - underscored LIKE '¥_%' ESCAPE '¥' は _foo の場合には true で、bar の場合には false です。
  - phone NOT LIKE '12%3' は、123 および 12993 の場合には false で、1234 の場合には true です。

  - LIKE または NOT LIKE 演算の ID がヌルである場合には、演算の値は不明です。
- identifier IS NULL 比較演算子は、ヌルのヘッダー・フィールド値または欠落したプロパティー値をテストします。
  - prop_name IS NULL
- identifier IS NOT NULL 比較演算子は、ヌル以外のヘッダー・フィールド値またはプロパティー値の存在をテストします。
  - prop_name IS NOT NULL

メッセージ・セレクターの例

以下のメッセージ・セレクターは、メッセージ・タイプが car、色が blue、重量が 2500 lbs より大きいというメッセージを選択します。

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

NULL プロパティー値

上記でも指摘したとおり、プロパティー値はヌルである場合があります。 ヌル値を含むセレクター式の評価は、SQL 92 NULL セマンティクスによって定義されます。 以下は、これらのセマンティクスの要旨です。

- SQL はヌル値を不明として扱います。
- 不明値を持つ比較または算術は、常に、不明値を生じさせます。
- IS NULL 演算子は、不明値を TRUE 値に変換します。
- IS NOT NULL 演算子は、不明値を FALSE 値に変換します。

JMSMessageID および JMSCorrelationID の特殊な振る舞い

JMSMessageID または JMSCorrelationID に基づいてキューからメッセージを選択する場合、IBM MQ classes for JMS には最適化が含まれています。

アプリケーションで次の形式のセレクターを指定するとします。

JMSMessageID='ID:*message_id*'

ここで、*message_id* は標準の IBM MQ メッセージ ID を含むストリングです。この場合、IBM MQ classes for JMS は **MatchOption** MQMO_MATCH_MSG_ID を使用して、指定されたメッセージ ID を持つメッセージを取得します。

例えば、キューからメッセージ ID 414D51207061756C745639314C545320C57C1A5F25ECE602 を持つメッセージを取得するには、アプリケーションで以下のメッセージ・セレクターを使用する必要があります。

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

同様に、アプリケーションで次の形式のセレクターを指定するとします。

JMSCorrelationID ='ID:*correlation_id*'

ここで、*correlation_id* は標準の IBM MQ 相関 ID を含むストリングです。この場合、IBM MQ classes for JMS は **MatchOption** MQMO_MATCH_CORREL_ID を使用して、指定された相関 ID を持つメッセージをキューから取得します。

以下の例では、414D51207061756C745639314C545320846E5B5F25B1CC02 の相関 ID を持つメッセージを取得するために、メッセージ・セレクターが使用されます。

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

メッセージ・セレクターで、*message_id* または *correlation_id* のいずれかについて、すべてがゼロである値が含まれている場合は、キューのすべてのメッセージと一致します。 例えば、アプリケーションで次のセレクターを使用しているとします。

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

この場合、キューのすべてのメッセージが一致と見なされ、アプリケーションに返されます。

MQMO_MATCH_MSG_ID and MQMO_MATCH_CORREL_ID **MatchOptions** の詳細については、MatchOptions (MQLONG)を参照してください。

制約事項

SQL では固定小数点の比較および算術をサポートしますが、JMS メッセージ・セレクターではサポートしません。 これは、正確な数値リテラルが小数部を持たない数値リテラルに制限されるためです。 また同じ理由で、近似数値の代替表記として小数部を持つ数値があります。

SQL コメントは、サポートされていません。

JMS メッセージの *IBM MQ* メッセージへのマッピング
IBM MQ メッセージは、メッセージ記述子、オプションの MQRFH2 ヘッダー、および本体で構成されます。 JMS メッセージの内容は、IBM MQ メッセージに、その一部がマップされ、一部がコピーされます。

このトピックでは、このセクションの最初の部分で説明されている JMS メッセージ構造体が IBM MQ メッセージにマップされる方法を説明します。 このセクションは、JMS アプリケーションと従来の IBM MQ アプリケーションとの間でメッセージを伝送する必要があるプログラマーを対象としています。 また、2 つの JMS アプリケーション間で伝送されるメッセージを操作する必要がある場合も対象となります (IBM Integration Bus の実装など)。

このセクションは、アプリケーションがブローカーとのリアルタイム接続を使用する場合には適用されません。 アプリケーションがリアルタイム接続を使用する場合は、通信はすべて TCP/IP を使用して直接行われます。IBM MQ キューやメッセージは使用されません。

IBM MQ メッセージは、以下の 3 つのコンポーネントから構成されています。

- IBM MQ メッセージ記述子 (MQMD)
- IBM MQ MQRFH2 ヘッダー
- メッセージ本体

MQRFH2 はオプションで、出力メッセージへの組み込みは JMS Destination クラスの TARGCLIENT フラグによって制御されます。 IBM MQ JMS 管理ツールを使用してこのフラグを設定できます。 MQRFH2 は JMS 固有の情報を伝送するため、受信宛先が JMS アプリケーションであることを送信側が認識している場合には、これを必ずメッセージに含めます。 通常、メッセージを非 JMS アプリケーションに直接送信するときには、MQRFH2 を省略してください。 これは、そのようなアプリケーションが IBM MQ メッセージ内に MQRFH2 を予期していないためです。

着信メッセージに MQRFH2 ヘッダーが含まれていない場合、そのメッセージの JMSReplyTo ヘッダー・フィールドから取り出された Queue オブジェクトまたは Topic オブジェクトでは、キューまたはトピックに送信される応答メッセージにも MQRFH2 ヘッダーが含まれないようにするため、デフォルトでこのフラグが設定されます。 元のメッセージに MQRFH2 ヘッダーが含まれている場合に限り、応答メッセージに MQRFH2 ヘッダーを含める動作をオフに切り替えることができます。これを行うには、接続ファクトリーの TARGCLIENTMATCHING プロパティーを NO に設定します。

151 ページの図 10 に、JMS メッセージの構造体が IBM MQ メッセージに変換されてから元に戻る様子を示します。

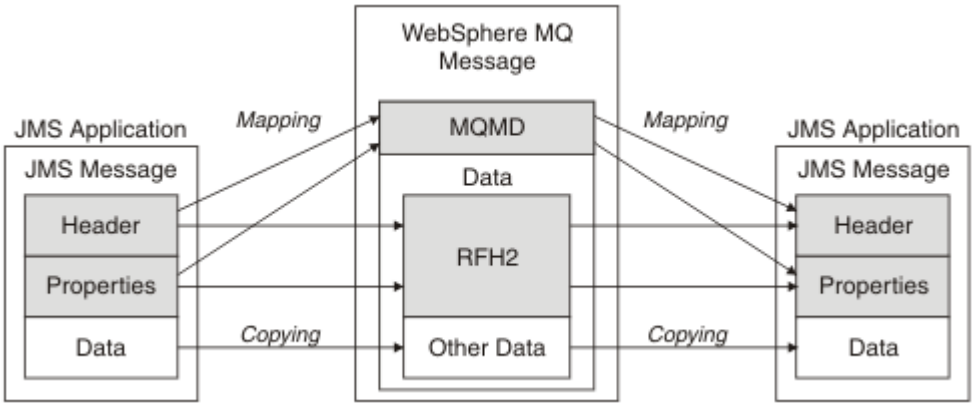


図 *10. MQRFH2* ヘッダーを使用して、メッセージが *JMS* と *IBM MQ* の間で変換される様子

これらの構造体は、以下の 2 つの方法で変換されます。

マッピング
MQMD が JMS フィールドと同等のフィールドを含んでいる場合、JMS フィールドは MQMD フィールドにマップされます。 追加の MQMD フィールドは、JMS プロパティーとして公開されます。これは、JMS アプリケーションが、非 JMS アプリケーションと通信する際にこれらのフィールドを取得または設定することが必要になる場合があるためです。

コピー
MQMD に同等のものがない場合、JMS ヘッダー・フィールドまたはプロパティーが、MQRFH2 内のフィールドとして渡され、場合によっては変換されます。

MQRFH2 ヘッダーと *JMS*
この一連のトピックでは、MQRFH バージョン 2 ヘッダーについて説明します。これは、メッセージ内容と関連のある JMS 固有のデータを伝送します。 MQRFH バージョン 2 ヘッダーは拡張可能であり、直接 JMS と関連のない追加情報も伝送できます。 ただし、このセクションでは、JMS による使用のみを扱います。すべての説明は、MQRFH2 – 規則および書式ヘッダー 2 を参照してください。

ヘッダーには 2 つの部分として、固定部分と変数部分があります。

固定部分
固定部分は、標準の IBM MQ ヘッダー・パターンにモデル化され、以下のフィールドから成り立っています。

StrucId (MQCHAR4)
構造体 ID

Must be MQRFH_STRUC_ID (値: "RFH ") (初期値)。

MQRFH_STRUC_ID_ARRAY (値: "R"、"F"、"H"、"") も定義されます。

Version (MQLONG)
構造体のバージョン番号。

MQRFH_VERSION_2 (値: 2) (初期値) でなければなりません。

StrucLength (MQLONG)
NameValueData フィールドを含む、MQRFH2 の全長。

StrucLength に設定される値は、4 の倍数でなければなりません (NameValueData フィールド内のデータは、これをアーカイブするためにスペース文字で埋められる場合があります)。

Encoding (MQLONG)
データ・エンコード。

MQRFH2 の後のメッセージの部分にある数値データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) のエンコード。

CodedCharSetId (MQLONG)
コード化文字セット ID。

MQRFH2 の後のメッセージ部分にある文字データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) の表記。

Format (MQCHAR8)
フォーマット名。

MQRFH2 の後のメッセージの部分のフォーマット名。

フラグ (MQLONG)
フラグ。

MQRFH_NO_FLAGS =0。 フラグが何も設定されていません。

NameValueCCSID (MQLONG)
このヘッダーに含まれている NameValueData 文字ストリング用のコード化文字セット ID (CCSID)。 NameValueData は、ヘッダー (StrucID および Format) 内に含まれているその他の文字ストリングとは異なる文字セットでコード化される場合があります。

NameValueCCSID が 2 バイトの Unicode CCSID (1200、13488、または 17584) である場合、Unicode のバイト順は、MQRFH2 内の数値フィールドのバイト順と同じです。 (例えば、Version、StrucLength、NameValueCCSID 自体。)

表 *16. NameValueCCSID* フィールドに指定できる値

CCSID	**意味**
1200	UTF-16 (サポートされる最新バージョンの Unicode)
13488	UTF-16 (Unicode バージョン 2.0 サブセット)
17584	UTF-16 (Unicode バージョン 3.0 サブセット) (ユーロ記号を含む)
1208	UTF-8 (サポートされる最新バージョンの Unicode)

変数部分
変数部分は、固定部分の後に続きます。 変数部分には、可変数の MQRFH2 フォルダーが含まれます。 それぞれのフォルダーには、可変数のエレメントまたはプロパティーが含まれます。 フォルダーは、関連のあるプロパティーをグループにまとめます。 JMS によって作成される MQRFH2 ヘッダーには、以下のいずれのフォルダーも含めることができます。

mcd フォルダー

mcd には、メッセージの形式を記述するプロパティーが入ります。 例えば、メッセージ・サービス・ドメインの Msd プロパティーは、JMS メッセージを JMSTextMessage、JMSBytesMessage、JMSStreamMessage、JMSMapMessage、JMSObjectMessage、またはヌルとして識別します。

mcd フォルダーは常に、MQRFH2 が入っている JMS メッセージ内に存在します。

これは常に、IBM Integration Bus から送信された MQRFH2 が入っている含むメッセージ内に存在します。 そして、メッセージのドメイン、形式、タイプ、およびメッセージ・セットを記述します。

表 *17. mcd* のプロパティー名、同義語、データ型、およびフォルダー

プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>

表 *17. mcd* のプロパティー名、同義語、データ型、およびフォルダー *(続き)*			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

独自のプロパティーを mcd フォルダーに追加しないでください。

jms フォルダー

jms には、JMS ヘッダー・フィールドと、MQMD で完全には表現できない JMSX プロパティーが含まれています。 jms フォルダーは常に、JMS MQRFH2 に存在します。

usr フォルダー

usr には、メッセージに関連付けられているアプリケーション定義の JMS プロパティーが入ります。 usr フォルダーは、アプリケーションがアプリケーション定義プロパティーを設定した場合のみ存在します。

mqext フォルダー

mqext には、以下のタイプのプロパティーが入ります。

- WebSphere Application Server 専用のプロパティー。
- メッセージの遅延送達に関連するプロパティー。

このフォルダーが存在するのは、アプリケーションで IBM 定義のプロパティーを 1 つ以上設定したか、遅延送達を使用した場合です。

表 *18. mqext* のプロパティー名、同義語、データ型、およびフォルダー			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

独自のプロパティーを mqext フォルダーに追加しないでください。

mqps フォルダー

mqps には、IBM MQ パブリッシュ/サブスクライブによってのみ使用されるプロパティーが入ります。 このフォルダーは、統合されたパブリッシュ/サブスクライブ・プロパティーを少なくとも 1 つアプリケーションが設定した場合にのみ存在します。

表 19. *mqps* のプロパティー名、同義語、データ型、およびフォルダー			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData...*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

独自のプロパティーを mqps フォルダーに追加しないでください。

154 ページの表 20 には、プロパティー名の完全なリストを示します。

表 20. *JMS* によって使用される *MQRFH2* フォルダーおよびプロパティー				
JMS フィールド名	**Java タイプ**	**MQRFH2 フォルダー名**	**プロパティー名**	**タイプ/値**
JMSDestination	Destination	jms	Dst	ストリング
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	ストリング	jms	Cid	ストリング
JMSReplyTo	Destination	jms	Rto	ストリング
JMSTimestamp	long	jms	Tms	i8
JMSType	ストリング	mcd	Type、Set、Fmt	ストリング
JMSXGroupID	ストリング	jms	Gid	ストリング
JMSXGroupSeq	int	jms	Seq	i4
xxx (ユーザー定義)	任意	usr	xxx	any

表 20. JMS によって使用される MQRFH2 フォルダーおよびプロパティー (続き)				
JMS フィールド名	**Java タイプ**	**MQRFH2 フォルダー名**	**プロパティー名**	**タイプ/値**
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
この長さフィールドの直後にある NameValueData ストリングの長さ (バイト単位) (それ自体の長さは含まれません)。

NameValueData (MQCHARn)
単一文字ストリング。長さ (バイト単位) は、前の NameValueLength フィールドに示されています。 この文字ストリングには、プロパティーのシーケンスを保持しているフォルダーが含まれています。 それぞれのプロパティーは、名前/タイプ/値のセットで、名前がフォルダー名である XML エレメント内に含まれており、以下のとおりです。

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

終了 </foldername> タグの後には、埋め込み文字としてスペースを続けることができま。 それぞれのセットは、XML と同様の構文を使用してエンコードされます。

```
<name dt='datatype'>value</name>
```

dt='datatype' エレメントはオプションであり、データ・タイプが事前定義されているため、多くのプロパティーでは省略されています。 これを含める場合は、 dt= タグの前に 1 つ以上のスペース文字を含める必要があります。

name
プロパティーの名前。154 ページの表 20 を参照してください。

datatype
省略後、155 ページの表 21 に示すデータ・タイプの 1 つと一致しなければなりません。

value
155 ページの表 21 の定義を使用して伝えられる値のストリング表記です。

ヌル値は、以下の構文を使用してエンコードされます。

```
<name dt='datatype' xsi:nil='true'></name>
```

xsi:nil='false'は使用しないでください。

表 21. プロパティーのデータ・タイプ	
データ・タイプ	**定義**
ストリング	< および & を除く任意の文字シーケンス
boolean	文字 0 または 1 (0 = false, 1 = true)
bin.hex	オクテットを表す 16 進数字。

表 21. プロパティーのデータ・タイプ (続き)	
データ・タイプ	**定義**
i1	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -128 から 127 (両端を含む) の範囲内になければなりません。
i2	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -32768 から 32767 (両端を含む) の範囲内になければなりません。
i4	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -2147483648 から 2147483647 (両端を含む) の範囲内になければなりません。
i8	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -9223372036854775808 から 9223372036854775807 (両端を含む) の範囲内になければなりません。
int	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 i8 と同じ範囲内になければなりません。 送信側が特定の精度をプロパティーと関連付けない場合に、'i*' タイプのいずれかの代わりにこれを使用できます。
r4	浮動小数点数、絶対値 < = 3.40282347E+38、> = 1.175E-37 数字を使用して表される 0..9、オプションの符号、オプションの小数桁、オプションの指数
r8	浮動小数点数、絶対値 < = 1.7976931348623E+308、> = 2.225E-307 数字を使用して表される 0..9、オプションの符号、オプションの小数桁、オプションの指数

ストリング値には、スペースを含めることができます。 ストリング値では以下のエスケープ・シーケンスを使用しなければなりません。

- & 文字の場合は &
- < 文字の場合は <

以下のエスケープ・シーケンスを使用できますが、必須ではありません。

- > 文字の場合は >
- ' 文字の場合は '
- " 文字の場合は "

対応する *MQMD* フィールドを持つ *JMS* フィールドおよびプロパティー

以下の表に、JMS ヘッダー・フィールド、JMS プロパティー、および JMS プロバイダー固有のプロパティーに対応する MQMD フィールドを示します。

156 ページの表 22 には、JMS ヘッダー・フィールドをリストし、157 ページの表 23 には、MQMD フィールドに直接マップされる JMS プロパティーをリストします。 157 ページの表 24 は、プロバイダー固有プロパティーと、それらがマップされる MQMD フィールドをリストしています。

表 22. *MQMD* フィールドへの *JMS* ヘッダー・フィールドのマッピング			
JMS ヘッダー・フィールド	**Java タイプ**	**MQMD フィールド**	**C タイプ**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	優先順位	MQLONG
JMSMessageID	ストリング	MsgID	MQBYTE24

表 22. *MQMD* フィールドへの *JMS* ヘッダー・フィールドのマッピング (続き)			
JMS ヘッダー・フィールド	**Java タイプ**	**MQMD フィールド**	**C タイプ**
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	ストリング	CorrelId	MQBYTE24

表 23. *MQMD* フィールドへの *JMS* プロパティーのマッピング			
JMS プロパティー	**Java タイプ**	**MQMD フィールド**	**C タイプ**
JMSXUserID	ストリング	UserIdentifier	MQCHAR12
JMSXAppID	ストリング	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	ストリング	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

表 24. *MQMD* フィールドへの *JMS* プロバイダー固有プロパティーのマッピング			
JMS プロバイダー固有プロパティー	**Java タイプ**	**MQMD フィールド**	**C タイプ**
JMS_IBM_Report_Exception	int	レポート	MQLONG
JMS_IBM_Report_Expiration	int	レポート	MQLONG
JMS_IBM_Report_COA	int	レポート	MQLONG
JMS_IBM_Report_COD	int	レポート	MQLONG
JMS_IBM_Report_PAN	int	レポート	MQLONG
JMS_IBM_Report_NAN	int	レポート	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	レポート	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	レポート	MQLONG
JMS_IBM_Report_Discard_Msg	int	レポート	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	ストリング	Format 158 ページの『1』	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	ストリング	CodedCharacterSetId 158 ページの『2』	MQLONG

表 24. *MQMD* フィールドへの *JMS* プロバイダー固有プロパティーのマッピング *(続き)*			
JMS プロバイダー固有プロパティー	**Java タイプ**	**MQMD フィールド**	**C タイプ**
JMS_IBM_PutDate	ストリング	PutDate	MQCHAR8
JMS_IBM_PutTime	ストリング	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

注 :

1. JMS_IBM_Format は、メッセージ本体のフォーマットを表します。 これは、メッセージの JMS_IBM_Format プロパティーを設定するアプリケーションによって定義できます (8 文字の長さ制限があることに注意してください)。または、JMS メッセージ・タイプに適したメッセージ本文の IBM MQ 形式にデフォルト設定することもできます。 メッセージに RFH あるいは RFH2 セクションが含まれていない場合、JMS_IBM_Format は MQMD Format フィールドにのみマップします。 標準的なメッセージでは、メッセージ本体の直前に RFH2 の Format フィールドにマップします。
2. JMS_IBM_Character_Set プロパティーの値は、数値の CodedCharacterSetId 値と同等の Java 文字セットを含むストリング値です。 MQMD フィールド CodedCharacterSetId は、JMS_IBM_Character_Set プロパティーで指定された Java 文字セットのストリングと同等のものを含む数値です。

JMS フィールドの *IBM MQ* フィールドへのマッピング *(*出力メッセージ*)*
これらの表は、JMS ヘッダーおよびプロパティー・フィールドが、send() または publish() 時に MQMD および MQRFH2 フィールドにどのようにマップされるかを示しています。

158 ページの表 25 に、send() または publish() 時に JMS ヘッダー・フィールドが MQMD/RFH2 フィールドにマップされる方法を示します。 159 ページの表 26 に、send() または publish() 時に JMS プロパティーが MQMD/RFH2 フィールドにマップされる方法を示します。 160 ページの表 27 に、send() または publish() 時に JMS プロバイダー固有プロパティーが MQMD フィールドにマップされる方法を示します。

設定にメッセージ・オブジェクトが示されているフィールドの場合、send() または publish() の実行の直前に、JMS メッセージ内に保持されている値が送信されます。 JMS メッセージ内の値は、この操作によって変更されることはありません。

設定に Send メソッドと示されているフィールドの場合、send() または publish() の実行時に値が割り当てられます (JMS メッセージ内に保持されている値は無視されます)。 JMS メッセージ内の値は、使用されている値を示すように更新されます。

受信のみというマークが付いているフィールドは、伝送されず、send() または publish() によってメッセージ内で変更されることはありません。

表 25. 出力メッセージ・フィールドのマッピング			
JMS ヘッダー・フィールド名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSDestination		MQRFH2	Send メソッド
JMSDeliveryMode	Persistence	MQRFH2	Send メソッド
JMSExpiration	Expiry	MQRFH2	Send メソッド
JMSPriority	優先順位	MQRFH2	Send メソッド
JMSMessageID	MsgID		Send メソッド
JMSTimestamp	PutDate/PutTime		Send メソッド

表 25. 出力メッセージ・フィールドのマッピング (続き)			
JMS ヘッダー・フィールド名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSCorrelationID	CorrelId	MQRFH2	メッセージ・オブジェクト
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	メッセージ・オブジェクト
JMSType		MQRFH2	メッセージ・オブジェクト
JMSRedelivered			受信のみ

注:

1. MQMD フィールド CodedCharacterSetId は、JMS_IBM_Character_Set プロパティーで指定された Java 文字セットのストリングと同等のものを含む数値です。

表 26. 出力メッセージ *JMS* プロパティーのマッピング			
JMS プロパティー名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSXUserID	UserIdentifier		Send メソッド
JMSXAppID	PutApplName		Send メソッド
JMSXDeliveryCount			受信のみ
JMSXGroupID	GroupId	MQRFH2	メッセージ・オブジェクト
JMSXGroupSeq	MsgSeqNumber	MQRFH2	メッセージ・オブジェクト

注:

これらのプロパティーは、JMS 仕様に基づいて読み取り専用として定義され、(一部のケースではオプションで) JMS プロバイダーによって設定されます。

IBM MQ classes for JMS では、これらのプロパティーの 2 つをアプリケーションによってオーバーライドできます。 これを行うには、以下のプロパティーを設定して、宛先が適切に構成されていることを確認してください。

1. プロパティー `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` を `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT` に設定します。
2. プロパティー `WMQConstants.WMQ_MQMD_WRITE_ENABLED` を `true` に設定します。

次のプロパティーはアプリケーションによってオーバーライドできます。

JMSXAppID
このプロパティーは、メッセージのプロパティー `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` を設定することによってオーバーライドできます。値は Java ストリングでなければなりません。

JMSXGroupID
このプロパティーは、メッセージにプロパティー `WMQConstants.JMS_IBM_MQMD_GROUPID` を設定することでオーバーライドできます (値はバイト配列であることが必要です)。

表 27. 出力メッセージの *JMS* プロバイダー固有プロパティーのマッピング

JMS プロバイダー固有プロパティー名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMS_IBM_Report_Exception	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Expiration	レポート		メッセージ・オブジェクト
JMS_IBM_Report_COA/COD	レポート		メッセージ・オブジェクト
JMS_IBM_Report_NAN/PAN	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Pass_Msg_ID	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Pass_Correl_ID	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Discard_Msg	レポート		メッセージ・オブジェクト
JMS_IBM_MsgType	MsgType		メッセージ・オブジェクト
JMS_IBM_Feedback	Feedback		メッセージ・オブジェクト
JMS_IBM_Format	Format		メッセージ・オブジェクト
JMS_IBM_PutApplType	PutApplType		Send メソッド
JMS_IBM_Encoding	Encoding		メッセージ・オブジェクト
JMS_IBM_Character_Set	CodedCharacterSetId		メッセージ・オブジェクト
JMS_IBM_PutDate	PutDate		Send メソッド
JMS_IBM_PutTime	PutTime		Send メソッド
JMS_IBM_Last_Msg_In_Group	MsgFlags		メッセージ・オブジェクト

send() または *publish()* の際の *JMS* ヘッダー・フィールドのマッピング
以下の情報は、send() または publish() の際の JMS フィールドのマッピングに関連した注記です。

JMSDestination → MQRFH2
これは、受信側の JMS が同等の宛先オブジェクトを再構成できるように宛先オブジェクトの顕著な特性をシリアライズするストリングとして保管されます。 MQRFH2 フィールドは URI としてエンコードされます (URI 表記の詳細については、225 ページの『Uniform Resource Identifier (URI)』を参照してください)。

JMSReplyTo → MQMD.ReplyToQ、ReplyToQMgr、MQRFH2
キュー名は MQMD.ReplyToQ フィールドにコピーされ、キュー・マネージャー名は ReplyToQMgr フィールドにコピーされます。 宛先拡張情報 (宛先オブジェクト内に保管されているその他の役立つ情報) は、MQRFH2 フィールドにコピーされます。 MQRFH2 フィールドは URI としてエンコードされます (URI 表記の詳細については、225 ページの『Uniform Resource Identifier (URI)』を参照してください)。

JMSDeliveryMode → MQMD.Persistence
Destination Object が指定変更しない限り、JMSDeliveryMode 値は send() または publish() メソッド、または MessageProducer によって設定されます。 JMSDeliveryMode 値は、以下のように MQMD.Persistence フィールドにマップされます。

- JMS 値 PERSISTENT は MQPER_PERSISTENT と等価です。
- JMS 値 NON_PERSISTENT は MQPER_NOT_PERSISTENT と等価です。

MQQueue 持続プロパティーが WMQConstants.WMQ_PER_QDEF に設定されないと、送達モード値も MQRFH2 でエンコードされます。

JMSExpiration ←/→ MQMD.Expiry、MQRFH2
JMSExpiration は、有効期限が切れる時刻 (現行時間と存続時間の合計) を保管しますが、MQMD は存続時間を保管します。 また、JMSExpiration はミリ秒単位ですが、MQMD.Expiry は 1/10 秒単位です。

- send() メソッドが無制限の存続時間を設定する場合、MQMD.Expiry は MQEI_UNLIMITED に設定され、JMSExpiration は MQRFH2 にエンコードされません。
- send() メソッドが 214748364.7 秒 (約 7 年) より短い存続時間を設定する場合、存続時間は MQMD.Expiry 内に保管され、有効期限が切れる時刻 (ミリ秒) は MQRFH2 内で i8 値としてエンコードされます。
- send() メソッドが 214748364.7 秒より長い存続時間を設定する場合、MQMD.Expiry は MQEI_UNLIMITED に設定されます。 正確な有効期限が切れる時刻 (ミリ秒単位) は、MQRFH2 内で i8 値としてエンコードされます。

JMSPriority → MQMD.Priority
JMSPriority 値 (0 から 9) を MQMD 優先順位値 (0 から 9) に直接マップします。 JMSPriority が非デフォルト値に設定される場合、優先順位も MQRFH2 内でエンコードされます。

JMSMessageID ← MQMD.MessageID
JMS から送信されるすべてのメッセージは、IBM MQ によって割り当てられる固有のメッセージ ID を持っています。 割り当てられた値は、MQPUT 呼び出し後に MQMD.MessageId フィールド内に戻され、JMSMessageID フィールド内のアプリケーションに渡されます。 IBM MQ messageId は 24 バイトのバイナリー値ですが、JMSMessageID はストリングです。 JMSMessageID は、文字 ID: という接頭部を持つ、48 個の 16 進文字のシーケンスに変換されたバイナリーの messageId 値から構成されています。 JMS は、メッセージ ID の生成を使用不可に設定できるヒントを提供します。 このヒントは無視され、すべての場合に固有 ID が割り当てられます。 send() の前に JMSMessageID フィールドに設定される値は上書きされます。

MQMD.MessageID。これは、 247 ページの『IBM MQ classes for JMS アプリケーションからのメッセージ記述子の読み取りと書き込み』 で説明されているいずれかの IBM MQ JMS 拡張機能を使用して行うことができます。

JMSTimestamp → MQRFH2
送信の際に、JMSTimestamp フィールドは JVM クロックに従って設定されます。 この値は、MQRFH2 に設定されます。 send() の前に JMSTimestamp フィールドに設定される値は上書きされます。 JMS_IBM_PutDate および JMS_IBM_PutTime プロパティーも参照してください。

JMSType → MQRFH2
このストリングは MQRFH2 mcd.Type フィールドに設定されます。 URI フォーマットの場合、mcd.Set および mcd.Fmt フィールドにも影響する可能性があります。

JMSCorrelationID → MQMD.CorrelId、MQRFH2
JMSCorrelationID は、以下のうちの 1 つを保持することができます。

プロバイダー固有のメッセージ ID
これは、前に送信または受信されたメッセージのメッセージ ID であり、ID: という接頭部が付いた 48 桁の小文字の 16 進数のストリングである必要があります。接頭部は削除され、残りの文字はバイナリーに変換され、その後 MQMD.CorrelId フィールドに設定されます。

プロバイダー・ネイティブの byte[] 値
値は、MQMD.CorrelId フィールドにコピーされ、必要であれば、24 バイトまでヌルで埋め込まれるか、または切り捨てられます。 correlid 値は、MQRFH2 でエンコードされます。

アプリケーション固有のストリング
値は、MQRFH2 にコピーされます。 ストリングの最初の 24 バイト (UTF8 形式) が、MQMD.CorrelID に書き込まれます。

JMS プロパティー・フィールドのマッピング
以下の注記は、JMS プロパティー・フィールドの IBM MQ メッセージへのマッピングを示しています。

JMSXUserID ← MQMD UserIdentifier
JMSXUserID は、送信呼び出しからの戻り時に設定されます。

JMSXAppID ← MQMD PutApplName
JSMXAppID は、送信呼び出しの戻り時に設定されます。

JMSXGroupID → MQRFH2 (Point-to-Point)
Point-to-Point メッセージの場合、JMSXGroupID は、MQMD GroupID フィールドにコピーされます。 JMSXGroupID が接頭部 ID: で始まる場合には、バイナリーに変換されます。 そうでない場合、UTF8 ストリングとしてエンコードされます。 必要であれば、値は 24 バイトの長さまで埋め込まれるか、または切り捨てられます。 MQMF_MSG_IN_GROUP フラグが立てられます。

JMSXGroupID → MQRFH2 (パブリッシュ/サブスクライブ)
パブリッシュ/サブスクライブ・メッセージの場合、JMSXGroupID はストリングとして MQRFH2 にコピーされます。

JMSXGroupSeq MQMD MsgSeqNumber (point-to-point)
Point-to-Point メッセージの場合、JMSXGroupSeq は、MQMD MsgSeqNumber フィールドにコピーされます。 MQMF_MSG_IN_GROUP フラグが立てられます。

JMSXGroupSeq MQMD MsgSeqNumber (パブリッシュ/サブスクライブ)
パブリッシュ/サブスクライブ・メッセージの場合、JMSXGroupSeq は i4 として MQRFH2 にコピーされます。

JMS プロバイダー固有のフィールドのマッピング
以下の注記は、JMS プロバイダー固有フィールドの IBM MQ メッセージへのマッピングを示しています。

JMS_IBM_Report_*XXX* → MQMD Report
JMS アプリケーションは、以下の JMS_IBM_Report_*XXX* プロパティーを使用して、MQMD Report オプションを設定できます。 単一の MQMD は、いくつかの JMS_IBM_Report_*XXX* プロパティーにマップされます。

JMS_IBM_Report_*XXX* 定数は、 `com.ibm.msg.client.jakarta.wmq.WMQConstants` または `com.ibm.msg.client.wmq.WMQConstants` にあります。

JMS_IBM_Report_Exception

MQRO_EXCEPTION または
MQRO_EXCEPTION_WITH_DATA または
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION または
MQRO_EXPIRATION_WITH_DATA または
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA または
MQRO_COA_WITH_DATA または
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD または
MQRO_COD_WITH_DATA または
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

MQRO の値については、 `com.ibm.mq.constants.CMQC` を参照してください。

JMS_IBM_MsgType → MQMD MsgType
値は、直接 MQMD MsgType にマップします。 アプリケーションが JMS_IBM_MsgType の明示的な値を設定していない場合には、デフォルト値が使用されます。 このデフォルト値は、以下のように決定されます。

- JMSReplyTo が IBM MQ キュー宛先に設定される場合、MSGType は値 MQMT_REQUEST に設定されます。
- JMSReplyTo が設定されていないか、または IBM MQ キュー宛先以外の宛先に設定されている場合には、MsgType は値 MQMT_DATAGRAM に設定されます。

JMS_IBM_Feedback → MQMD Feedback
値は、直接 MQMD Feedback にマップします。

JMS_IBM_Format → MQMD Format
値は、直接 MQMD Format にマップします。

JMS_IBM_Encoding → MQMD Encoding
設定される場合、このプロパティーは Destination Queue または Topic の数値エンコードを指定変更します。

JMS_IBM_Character_Set → MQMD CodedCharacterSetId
設定される場合、このプロパティーは、Destination Queue または Topic のコード化文字セット・プロパティーを指定変更します。

JMS_IBM_PutDate ← MQMD PutDate
このプロパティーの値は、送信の際に、MQMD 内の PutDate フィールドから直接設定されます。 send の前に JMS_IBM_PutDate プロパティーに設定される値は上書きされます。 このフィールドは 8 文字のストリングで、IBM MQ 日付形式が YYYYMMDD です。 このプロパティーは、JMS_IBM_PutTime プロパティーと使用して、キュー・マネージャーに従ってメッセージが書き込まれた時間を判別することができます。

JMS_IBM_PutTime ← MQMD PutTime
このプロパティーの値は、送信の際に、MQMD 内の PutTime フィールドから直接設定されます。 send の前に JMS_IBM_PutTime プロパティーに設定される値は上書きされます。 このフィールドは 8 文字のストリングで、IBM MQ 時刻形式が HHMMSSTH です。 このプロパティーは、JMS_IBM_PutDate プロパティーと使用して、キュー・マネージャーに従ってメッセージが書き込まれた時間を判別することができます。

JMS_IBM_Last_Msg_In_Group → MQMD MsgFlags
Point-to-Point メッセージングの場合、このブール値は MQMD MsgFlags フィールドの MQMF_LAST_MSG_IN_GROUP フラグにマップします。 通常、JMSXGroupID および JMSXGroupSeq プロパティーと共に使用され、レガシー IBM MQ アプリケーションに対してこのメッセージがグループ内で最後であることを示します。 パブリッシュ/サブスクライブ・メッセージングの場合には、このプロパティーは無視されます。

IBM MQ フィールドの *JMS* フィールドへのマッピング *(*着信メッセージ*)*
これらの表は、JMS ヘッダーおよびプロパティー・フィールドが get() または receive() 時に MQMD および MQRFH2 フィールドへどのようにマップされるかを示しています。

164 ページの表 28 に、get() または receive() 時に JMS ヘッダー・フィールドが MQMD/MQRFH2 フィールドにマップされる方法を示します。 164 ページの表 29 に、get() または receive() 時に JMS プロパティー・フィールドが MQMD/MQRFH2 フィールドにマップされる方法を示します。 165 ページの表 30 に、JMS プロバイダー固有プロパティーがマップされる方法を示します。

表 *28.* 着信メッセージ *JMS* ヘッダー・フィールドのマッピング

JMS ヘッダー・フィールド名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMSDestination		jms.Dst または mqps.Top 164 ページの『1』
JMSDeliveryMode	Persistence 164 ページの『2』	jms.Dlv 164 ページの『2』
JMSExpiration		jms.Exp
JMSPriority	優先順位	
JMSMessageID	MsgID	
JMSTimestamp	PutDate 164 ページの『2』 PutTime 164 ページの『2』	jms.Tms 164 ページの『2』
JMSCorrelationID	CorrelId 164 ページの『2』	jms.Cid 164 ページの『2』
JMSReplyTo	ReplyToQ 164 ページの『2』 ReplyToQMgr 164 ページの『2』	jms.Rto 164 ページの『2』
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

注 :

1. jms.Dst と mqps.Top の両方が設定されている場合、jms.Dst の値が使用されます。
2. MQRFH2 または MQMD から値を取得することができるプロパティーに関して、両方が使用可能な場合には MQRFH2 の設定が使用されます。
3. JMS_IBM_Character_Set プロパティーの値は、数値の CodedCharacterSetId 値と同等の Java 文字セットを含むストリング値です。

表 *29.* 着信メッセージ・プロパティーのマッピング

JMS プロパティー名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId 164 ページの『1』	jms.Gid 164 ページの『1』
JMSXGroupSeq	MsgSeqNumber 164 ページの『1』	jms.Seq 164 ページの『1』

注 :

1. MQRFH2 または MQMD から値を取得することができるプロパティーに関して、両方が使用可能な場合には MQRFH2 の設定が使用されます。 MQMF_MSG_IN_GROUP または MQMF_LAST_MSG_IN_GROUP のメッセージ・フラグが設定されている場合、このプロパティーは MQMD 値からのみ設定されます。

表 30. 着信メッセージのプロバイダー固有 *JMS* プロパティーのマッピング		
JMS プロパティー名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMS_IBM_Report_Exception	レポート	
JMS_IBM_Report_Expiration	レポート	
JMS_IBM_Report_COA	レポート	
JMS_IBM_Report_COD	レポート	
JMS_IBM_Report_PAN	レポート	
JMS_IBM_Report_NAN	レポート	
JMS_IBM_Report_ Pass_Msg_ID	レポート	
JMS_IBM_Report_Pass_Correl_ID	レポート	
JMS_IBM_Report_Discard_Msg	レポート	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding [165 ページの『1』](#)	Encoding	
JMS_IBM_Character_Set [165 ページの『1』](#)	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. 着信メッセージが Bytes Message である場合にのみ設定されます。

JMS アプリケーションと従来の *IBM MQ* アプリケーションとの間でのメッセージの交換
このトピックでは、JMS アプリケーションが、MQRFH2 ヘッダーを処理できない従来の IBM MQ アプリケーションとメッセージを交換するときになされる処理について説明します。

[166 ページの図 11](#) に、そのマッピングを示します。

管理者は、宛先の TARGCLIENT プロパティーを *MQ* に設定することにより、 JMS アプリケーションが従来の IBM MQ アプリケーションと通信していることを示します。 これは、MQRFH2 ヘッダーが生成されないことを示しています。 これが行われない場合、受信側アプリケーションが MQRFH2 ヘッダーを扱えなければなりません。

従来の IBM MQ アプリケーションをターゲットとする JMS から MQMD へのマッピングは、 JMS アプリケーションをターゲットとする JMS から MQMD へのマッピングと同じです。 IBM MQ classes for JMS が受け取った IBM MQ メッセージの MQMD *Format* フィールドが、MQFMT_RFH2 以外に設定されている場合、受信されるデータは非 JMS アプリケーションからのものです。 フォーマットが MQFMT_STRING である場合、メッセージは JMS テキスト・メッセージとして受信されます。 それ以外の場合、メッセージは JMS バイト・メッセージとして受信されます。 MQRFH2 が存在しないため、MQMD で伝送されるそれらの JMS プロパティーのみを復元できます。

IBM MQ classes for JMS が、MQRFH2 ヘッダーを含まないメッセージを受信した場合、メッセージの JMSReplyTo ヘッダー・フィールドから取り出された Queue オブジェクトまたは Topic オブジェクトの TARGCLIENT プロパティーは、デフォルトで MQ に設定されます。 つまり、キューまたはトピックに送信される応答メッセージにも、MQRFH2 ヘッダーは含まれません。 元のメッセージに MQRFH2 ヘッダーが

含まれている場合に限り、応答メッセージに MQRFH2 ヘッダーを含める動作をオフに切り替えることができます。これを行うには、接続ファクトリーの TARGCLIENTMATCHING プロパティーを NO に設定します。

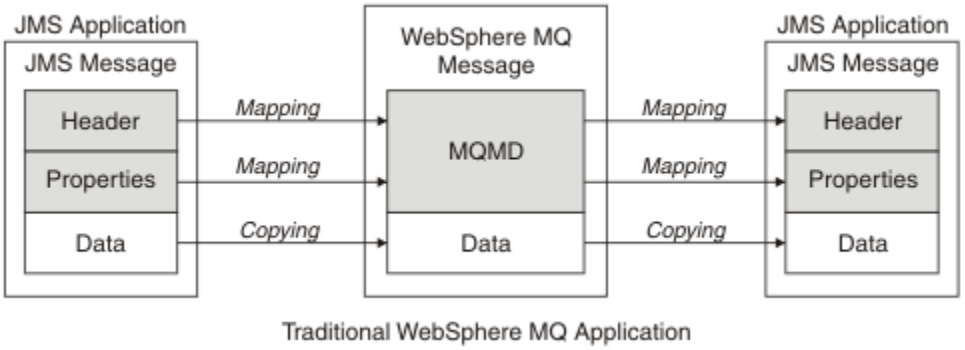


図 *11. JMS* メッセージが *IBM MQ* メッセージに変換される様子 *(MQRFH2* ヘッダーなし*)*

JMS メッセージ本体
このトピックには、メッセージ本体自体のエンコードに関する情報が記載されています。 エンコードは、JMS メッセージのタイプによって異なります。

ObjectMessage
ObjectMessage は、通常の方法で、Java Runtime によってシリアライズされるオブジェクトです。

TextMessage
TextMessage は、エンコードされたストリングです。 出力メッセージの場合、ストリングは、宛先オブジェクトによって提供されている文字セットにエンコードされます。 これは、デフォルトでは UTF8 エンコードになります (UTF8 エンコードは、メッセージの最初の文字で始まります。先頭に長さフィールドはありません)。 ただし、IBM MQ classes for JMS によってサポートされているその他の文字セットを指定できます。 こうした文字セットは、主に、非 JMS アプリケーションにメッセージを送信する際に使用されます。

文字セットが 2 バイト・セット (UTF16 を含む) である場合、宛先オブジェクトの整数エンコード仕様により、バイト順が決定されます。

着信メッセージは、メッセージ自体に指定されている文字セットおよびエンコードを使用して解釈されます。 これらの指定は、末尾の IBM MQ ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、末尾のヘッダーは、通常、MQRFH2 です。

BytesMessage
デフォルトで BytesMessage は、JMS 1.0.2 仕様および関連のある Java 資料で定義されている、バイトのシーケンスです。

アプリケーション自体によってアセンブルされた出力メッセージの場合、宛先オブジェクトのエンコード・プロパティーは、メッセージ内に含まれている整数および浮動小数点フィールドのエンコードを指定変更するために使用されることがあります。 例えば、浮動小数点値は IEEE 形式ではなく S/390 形式で保管されるように要求することができます。

着信メッセージは、メッセージ自体に指定されている数値エンコードを使用して解釈されます。 この指定は、最後の IBM MQ ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、末尾のヘッダーは、通常、MQRFH2 です。

BytesMessage が受信され、変更されることなく再送信される場合、メッセージ本体は、受信されたとおりに、バイトごとに送信されます。 宛先オブジェクトのエンコード・プロパティーはメッセージ本体に影響を与えません。 BytesMessage 内で明示的に送信できる唯一のストリング系エンティティーは、UTF8 ストリングです。 これは、Java UTF8 形式でエンコードされ、2 バイトの長さフィールドで始まります。 宛先オブジェクトの文字セット・プロパティーは、出力 BytesMessage のエンコードに影響を与えません。 着信 IBM MQ メッセージ内の文字セット値は、そのメッセージの JMS BytesMessage としての解釈に影響を与えません。

Java 以外のアプリケーションでは、通常、Java UTF8 エンコードを認識しません。 したがって、テキスト・データを含む BytesMessage を送信する JMS アプリケーションの場合、アプリケーション自体

は、そのストリングをバイト配列に変換し、これらのバイト配列を BytesMessage に書き込まなければなりません。

MapMessage
MapMessage は、XML の名前/タイプ/値のトリプレットを含むストリングです。次のようにエンコードされます。

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

ここで、datatype は、155 ページの表 21 に示したデータ・タイプの 1 つになります。 デフォルトのデータ・タイプは string であるため、ストリング・エレメントの属性 dt="string" は省略されています。

マップ・メッセージの本体を形成する XML ストリングのエンコードまたは解釈に使用される文字セットは、テキスト・メッセージに適用される規則に従って決定されます。

5.3 より前のバージョンの IBM MQ classes for JMS では、次のようなフォーマットを使用して、マップ・メッセージの本体をエンコードしていました。

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 以降のバージョンはどちらのフォーマットでも解釈できますが、IBM MQ classes for JMS の 5.3 より前のバージョンは現行フォーマットを解釈できません。

アプリケーションが、5.3 より前の IBM MQ classes for JMS を使用する別のアプリケーションにマップ・メッセージを送信する必要がある場合、送信側アプリケーションは接続ファクトリー・メソッド setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE) を呼び出して、マップ・メッセージが以前のフォーマットで送信されることを指定する必要があります。 デフォルトでは、すべてのマップ・メッセージは現行フォーマットで送信されます。

StreamMessage
StreamMessage は、マップ・メッセージと同様のものですが、エレメント名は持ちません。

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

ここで、datatype は、155 ページの表 21 に示したデータ・タイプの 1 つになります。 デフォルトのデータ・タイプは string であるため、ストリング・エレメントの属性 dt="string" は省略されています。

StreamMessage 本体を構成する XML ストリングのエンコードまたは解釈に使用される文字セットは、TextMessage に適用される規則に従って決定されます。

MQRFH2.format フィールドは、以下のように設定されます。

MQFMT_NONE
ObjectMessage、BytesMessage、または本体がないメッセージの場合。

MQFMT_STRING
TextMessage、StreamMessage、または MapMessage の場合。

JMS メッセージ変換
JMS のメッセージ・データ変換は、メッセージの送受信時に実行されます。 IBM MQ では、ほとんどのデータ変換が自動的に実行されます。 テキストおよび数値データは、JMS アプリケーション間でメッセージ

を転送するときに変換されます。 JMS アプリケーションと IBM MQ アプリケーションの間で JMSTextMessage を交換すると、テキストが変換されます。

複雑なメッセージ交換を実行する場合は、以下の各トピックが参考になります。 複雑なメッセージ交換には次のようなものがあります。

- IBM MQ アプリケーションと JMS アプリケーションの間での非テキスト・メッセージの転送
- バイト形式のテキスト・データの交換
- アプリケーションでのテキストの変換

JMS メッセージ・データ

データ変換は、アプリケーション間でテキストおよび数値データを交換するために必要です。2 つの JMS アプリケーション間で交換する場合も同様です。 テキストと数値の内部表記は、メッセージで転送できるようにエンコードする必要があります。 エンコードにより、数値とテキストの表記方法が強制的に決定されます。 IBM MQ は、 JMSObjectMessage を除き、 JMS メッセージ内のテキストと数値のエンコードを管理します。 174 ページの『JMSObjectMessage』 を参照してください。 3 つのメッセージ属性を使用します。 3 つの属性は、CodedCharacterSetId、Encoding、および Format です。

通常、これらの 3 つのメッセージ属性は、 JMS メッセージの JMS ヘッダー、 MQRFH2 フィールドに保管されます。 メッセージ・タイプが JMS タイプのメッセージではなく MQ の場合、属性はメッセージ記述子 MQMD に保管されます。 これらの属性は JMS メッセージ・データの変換に使用されます。 JMS メッセージ・データは、IBM MQ メッセージのメッセージ・データ部分で転送されます。

JMS メッセージ・プロパティー

JMS_IBM_CHARACTER_SET などの JMS メッセージ・プロパティーは、 MQRFH2 なしでメッセージが送信された場合を除き、JMS メッセージの MQRFH2 ヘッダー部分で交換されます。 MQRFH2 なしで送信できるのは、JMSTextMessage と JMSBytesMessage のみです。 JMS プロパティーが IBM MQ メッセージ・プロパティーとしてメッセージ記述子 MQMD に格納されている場合は、MQMD 変換の一部として変換されます。 JMS プロパティーが MQRFH2 に格納されている場合は、MQRFH2.NameValueCCSID で指定された文字セットで格納されています。 メッセージを送受信する際に、メッセージ・プロパティーは JVM で内部表記との間で変換されます。 変換は、メッセージ記述子の文字セットまたは MQRFH2.NameValueCCSID との間で行われます。 数値データはテキストに変換されます。

JMS メッセージ変換

以下のトピックには、変換を必要とする複雑なメッセージを交換する場合に役立つ例とタスクが記載されています。

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、IBM MQ によって自動的に実行されます。

メッセージ変換のアプローチ方法に関するいくつかの質問を紹介します。

メッセージ変換について考慮する必要はありますか。
場合によっては、JMS から JMS へのメッセージ転送、および IBM MQ プログラムとのテキスト・メッセージの交換などの場合、IBM MQ は自動的に必要な変換を実行します。 しかし、パフォーマンス上の理由でデータ変換を自分で制御したい場合や、定義済みフォーマットを持つ複雑なメッセージを交換する場合もあります。 このような場合は、メッセージ変換を理解し、次のトピックを読む必要があります。

変換にはどのような種類がありますか。
変換には、主に 4 つの種類があります。それぞれについて次のセクションで説明します。

1. 169 ページの『JMS クライアント・データ変換』

2. 169 ページの『アプリケーション・データの変換』
3. 170 ページの『キュー・マネージャー・データ変換』
4. 171 ページの『メッセージ・チャネル・データ変換』

変換はどこで実行する必要がありますか。

171 ページの『メッセージ変換へのアプローチの選択: 「受信側で実行」』のセクションでは、"「受信側で実行」"の通常のアプローチについて説明します。 "「受信側で実行」"は JMS データ変換にも適用されます。

JMS クライアント・データ変換

JMS クライアント [1] データ変換では、Java プリミティブとオブジェクトを宛先に送信するときに JMS メッセージのバイトに変換し、受信時に変換して元に戻します。 JMS クライアント・データ変換は、JMSMessage クラスのメソッドを使用します。 172 ページの表 31 に、それらのメソッドを JMSMessage クラス・タイプ別に示します。

数値とテキストの内部 JVM 表記との変換は、read、get、set、および write メソッドで実行されます。 変換が実行されるのは、メッセージの送信時、および受信したメッセージに対して read または get メソッドが呼び出されたときです。

メッセージの内容の書き込み (write) または設定 (set) で使用されるコード・ページと数値エンコードは、宛先の属性として定義されます。 宛先のコード・ページと数値エンコードは、管理上変更できます。 アプリケーションで、メッセージ内容の書き込みまたは設定を制御するメッセージ・プロパティーを設定して、宛先のコード・ページとエンコードを指定変更することもできます。

Native エンコードが定義されていない宛先への JMSBytesMessage メッセージの送信時に、数値エンコードを変換する場合は、メッセージの送信前にメッセージ・プロパティー JMS_IBM_ENCODING を設定する必要があります。 "receiver" パターンに従っている場合、または JMS アプリケーション間でメッセージを交換する場合、アプリケーションは JMS_IBM_ENCODING を設定する必要はありません。 ほとんどの場合、Encoding プロパティーは Native のままで構いません。

JMSStreamMessage、JMSMapMessage、および JMSTextMessage の各メッセージでは、宛先の文字セット ID のプロパティーが使用されます。 数値がテキスト形式で書き出されると、エンコードは送信時に無視されます。 宛先の文字セットのプロパティーが適用される場合、メッセージの送信前に JMS クライアント・アプリケーション・プログラムで JMS_IBM_CHARACTER_SET を設定する必要はありません。

メッセージのデータを取得するために、アプリケーションは JMS メッセージの read メソッドまたは get メソッドを呼び出します。 これらのメソッドは、直前のメッセージ・ヘッダーで定義されているコード・ページとエンコードを参照して、Java プリミティブとオブジェクトを正しく作成します。

JMS クライアント・データ変換は、JMS クライアント間でメッセージを交換するほとんどの JMS アプリケーションのニーズを満たすものです。 明示的なデータ変換をコーディングする必要はありません。 テキストをファイルに書き込むときに一般的に使用される java.nio.charset.Charset クラスは使用しないでください。 writeString メソッドと setString メソッドによって変換が実行されます。

JMS クライアント・データ変換の詳細については、181 ページの『JMS クライアント・メッセージ変換とエンコード』を参照してください。

アプリケーション・データの変換

JMS クライアント・アプリケーションは、java.nio.charset.Charset クラスを使用して明示的な文字データ変換を実行することができます。174 ページの図 14 および 174 ページの図 15 内の例を参照してください。 ストリング・データは、getBytes メソッドを使用してバイトに変換され、バイトとして送信されます。 このバイトは、バイト配列と Charset を取得する String コンストラクターを使用して、再びテキストに変換されます。 文字データは、encode メソッドと decode Charset メソッドを使用して変換されます。 通常、メッセージは JMSBytesMessage として送信または受信されます。これは、JMSBytesMessage のメッセージ部分に、アプリケーションによって書き込まれたデータ以外のものが含

[1] "JMS クライアント"は、JMS インターフェースを実装する IBM MQ classes for JMS を参照します。これにより、クライアント・モードまたはバインディング・モードを実行します。

まれていないためです。[2]JMSStreamMessage、JMSMapMessage、または JMSObjectMessage を使用してバイトを送信および受信することもできます。

異なる複数のエンコード形式で表現された数値データが含まれるバイトをエンコードおよびデコードするための Java メソッドはありません。 数値データは、数値の JMSMessage の read メソッドおよび write メソッドを使用して自動的にエンコードおよびデコードされます。 read メソッドおよび write メソッドでは、メッセージ・データの JMS_IBM_ENCODING 属性の値を使用します。

アプリケーション・データ変換が一般的に使用されるのは、JMS クライアントが JMS 以外のアプリケーションからフォーマット済みのメッセージを送受信する場合です。 フォーマット済みメッセージには、データ・フィールドの長さで編成されたテキスト、数値、およびバイトのデータが含まれています。 JMS 以外のアプリケーションで "MQSTR "フォーマットが指定されていない限り、メッセージは JMSBytesMessage として構成されます。 JMSBytesMessage でフォーマット済みメッセージ・データを受信するには、一連のメソッドを呼び出す必要があります。 メソッドは、メッセージに書き込まれたフィールドと同じ順序で呼び出す必要があります。 フィールドが数値である場合、数値データのエンコードと長さを把握する必要があります。 いずれかのフィールドにバイト・データまたはテキスト・データが含まれている場合は、メッセージ内のバイト・データの長さを把握する必要があります。 フォーマット済みのメッセージを利便性がある Java オブジェクトに変換する方法は 2 つあります。

1. レコードに対応する Java クラスを構成し、メッセージの読み取りと書き込みをカプセル化します。 レコード内のデータへのアクセスには、クラスの get メソッドと set メソッドを使用します。
2. com.ibm.mq.headers クラスを拡張することにより、レコードに対応する Java クラスを構成します。 クラス内のデータへのアクセスは、 getStringValue(*fieldname*); という形式のタイプ固有のアクセサーを使用して行われます。

189 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照。

キュー・マネージャー・データ変換

コード・ページ変換は、 JMS クライアント・プログラムがメッセージを取得するときに、キュー・マネージャーによって実行できます。 変換は C プログラムに対して実行される変換と同じです。 C プログラムは、MQGMO_CONVERT を MQGET GetMsgOpts パラメーター・オプションとして設定します。173 ページの図 13 を参照してください。 キュー・マネージャーは、メッセージを受信する JMS クライアント・プログラムの変換を実行します。 WMQ_RECEIVE_CONVERSION 宛先プロパティーが WMQ_RECEIVE_CONVERSION_QMGR に設定されている場合、JMS クライアント・プログラムは宛先プロパティーを設定することもできます。 170 ページの図 12 を参照してください。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

または

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

図 *12.* キュー・マネージャー・データ変換の有効化

キュー・マネージャー変換の主な利点は、JMS 以外のアプリケーションとメッセージを交換するときに得られます。 メッセージの Format フィールドが定義され、宛先の文字セットまたはエンコードがメッセージと異なる場合は、ターゲット・アプリケーションからデータ変換が要求されると、キュー・マネージャーによって変換が実行されます。 キュー・マネージャーは、 CICS bridge ヘッダー (MQCIH) などの事前定義 IBM MQ メッセージ・タイプの 1 つに従ってフォーマット設定されたメッセージ・データを変換します。

[2] 1 つの例外: writeUTF を使用して書き込まれたデータは、2 バイトの長さフィールドから始まります

Format フィールドがユーザー定義の場合、キュー・マネージャーは Format フィールドに指定された名前のデータ変換出口を探します。

キュー・マネージャーのデータ変換は、"「受信側で実行」"の設計パターンによる効果を最適化するために使用されます。 送信側の JMS クライアントで変換を実行する必要はありません。 JMS 以外の受信プログラムは、メッセージが必要なコード・ページとエンコードで確実に送信されるように、変換出口に依存します。 送信側の JMS クライアント、および非 JMS 受信側では、この例は IBM MQ に適用されます。

データ変換出口ユーティリティー **crtmqcvx** を使用してデータ変換出口を作成し、キュー・マネージャーが独自のレコード・フォーマット済みデータを変換できるようにすることも可能です。 独自のレコード・フォーマットを作成し、com.ibm.mq.headers を使用してそれを Java クラスとしてアクセスし、独自の変換出口を使用してそれを変換することができます。 ユーティリティーの名前は、z/OS では **CSQUCVX**、IBM i では **CVTMQMDTA** です。 189 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照してください。

メッセージ・チャネル・データ変換

IBM MQ の送信側、サーバー、クラスター受信側、およびクラスター送信側のチャネルには、メッセージ変換オプション CONVERT があります。 メッセージの内容は、メッセージの送信時にオプションで変換できます。 変換は、チャネルの送信側で行われます。 クラスター受信側の定義を使用して、対応するクラスター送信側チャネルが自動定義されます。

メッセージ・チャネルによるデータ変換は、通常、他の変換形式を使用できない場合に使用されます。

メッセージ変換へのアプローチの選択: "「受信側で実行」"

コード変換のための IBM MQ アプリケーション設計における通常の方法は、"「受信側で実行」"です。 "「受信側で実行」"は、メッセージ変換の数を減らします。 また、メッセージ転送時に一部の中間キュー・マネージャーでメッセージ変換が失敗した場合に、予期しないチャネル・エラーの問題が発生することを回避することもできます。 "「受信側で実行」"ルールは、受信側が実行できない理由がある場合にのみ中断されます。 例えば、受信側のプラットフォームに適切な文字セットがない場合があります。

"「受信側で実行」" は、JMS クライアント・アプリケーションの一般的なガイダンスにもなります。 ただし、特定のケースでは、送信元で正しい文字セットに変換したほうが効率的な場合があります。 JVM 内部表記からの変換は、テキスト・タイプや数値タイプを含んだメッセージの送信時に行う必要があります。 受信側が JMS クライアントでない場合、受信側に必要な文字セットに変換しておくことにより、JMS 以外の受信側で変換を実行しなくてもよくなる場合があります。 受信側が JMS クライアントである場合、再度変換を実行してメッセージ・データをデコードし、Java プリミティブとオブジェクトを作成します。

JMS クライアント・アプリケーションと C などの言語で作成されたアプリケーションの違いは、Java でデータ変換を実行する必要があるという点です。 Java アプリケーションで数値とテキストを内部表記からメッセージで使用されているエンコードされたフォーマットに変換する必要があります。

宛先またはメッセージ・プロパティーを設定することにより、IBM MQ で使用される文字セットとエンコードを設定して、メッセージ内の数値とテキストをエンコードできます。 通常、文字セットは 1208、エンコードは Native のままにします。

IBM MQ では、バイト配列は変換されません。 ストリングおよび文字配列をバイト配列にエンコードするには、java.nio.charset パッケージを使用します。 Charset では、ストリングまたは文字配列をバイト配列に変換するために使用される文字セットを指定します。 Charset を使用してバイト配列をストリングまたは文字配列にデコードすることもできます。 ストリングおよび文字配列をエンコードするときには、java.nio.charset.Charset.defaultCodePage に依存するのはよくありません。 デフォルトの Charset は、通常、 Windows では windows-1252 、 AIX and Linux では UTF-8 です。 windows-1252 は 1 バイト文字セット、UTF-8 はマルチバイト文字セットです。

通常、宛先の文字セットおよびエンコード・プロパティーは、他の JMS アプリケーションとメッセージを交換するときに、デフォルト値の UTF-8 および Native のままにします。 数値またはテキストが含まれるメッセージを JMS アプリケーションと交換する場合は、メッセージ・タイプを JMSTextMessage、JMSStreamMessage、JMSMapMessage、または JMSObjectMessage の中から目的に合わせて選択します。 その他に実行する変換タスクはありません。

レコード・フォーマットを使用する JMS 以外のアプリケーションとメッセージを交換する場合は、さらに複雑になります。 レコード全体にテキストが含まれ、JMSTextMessage として転送できる場合を除き、アプリケーションでテキストをエンコードおよびデコードする必要があります。 宛先のメッセージ・タイプを MQ に設定し、JMSBytesMessage を使用して、IBM MQ classes for JMS によって追加のヘッダーとタグ付け情報がメッセージ・データに追加されるのを防ぎます。 数値とバイトの書き込みには JMSBytesMessage メソッドを使用し、テキストをバイト配列に明示的に変換するには Charset クラスを使用します。 多くの要因が文字セットの選択に影響します。

- パフォーマンス: 最も多くのサーバーで使用されている文字セットにテキストを変換することによって変換の回数を減らすことができますか。
- 均一性: すべてのメッセージを同じ文字セットで転送します。
- 豊富な種類: アプリケーションで使用する必要があるすべてのコード・ポイントを含む文字セットはどれですか。
- 単純さ: 可変長およびマルチバイトの文字セットより、1 バイト文字セットのほうが簡単に使用できます。

189 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照。 非 JMS アプリケーションと交換されるメッセージの変換の例を示します。

例

メッセージ・タイプと変換タイプの表

表 *31.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**テキスト**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

表 31. メッセージ・タイプと変換タイプ (続き)

	変換タイプ			
メッセージ・タイプ	テキスト	数字	その他	なし
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

C プログラムからのデータ変換の呼び出し

```
gmo.Options = MQGMO_WAIT         /* wait for new messages            */
            | MQGMO_NO_SYNCPOINT /* no transaction                   */
            | MQGMO_CONVERT;     /* convert if necessary             */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle                */
           Hobj,                /* object handle                    */
           &md,                 /* message descriptor               */
           &gmo,                /* get message options              */
           buflen,              /* buffer length                    */
           buffer,              /* message buffer                   */
           &messlen,            /* message length                   */
           &CompCode,           /* completion code                  */
           &Reason);            /* reason code                      */
```

図 13. *amqsget0.c* からのコード・スニペット

JMSBytesMessage でのテキストの送受信

174 ページの図 14 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 複数のタイプが混在したバイト・メッセージでテキスト・ストリングを受信するには、バイト単位のストリング長を把握する必要がありま

す。これは、174 ページの図 15 にある *TEXT_LENGTH* という名前の変数に格納されます。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *14. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *15. JMSBytesMessage* からの *String* の受信

関連概念

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 メッセージを受信する JMS クライアントも、オプションのキュー・マネージャー・データ変換を使用します。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できるデータ変換出口、および JMS クライアント・アプリケーションを設計して作成するには、このタスクで推奨されるステップに従ってください。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMSObjectMessage

JMSObjectMessage には、1 つのオブジェクトおよびそのオブジェクトが参照するすべてのオブジェクトが含まれ、これらのオブジェクトは JVM によってバイト・ストリームにシリアライズされています。 テキストは UTF-8 にシリアライズされ、65534 バイト以下のストリングまたは文字配列に制限されます。
JMSObjectMessage の利点は、アプリケーションがオブジェクトのメソッドと属性のみを使用している限り、データ変換の実行に関与しないという点です。 JMSObjectMessage には複雑なオブジェクト用のデータ変換が用意されているため、アプリケーション・プログラマーがメッセージのオブジェクトのエンコ

ード方法を考慮する必要はありません。 JMSObjectMessage を使用した場合の欠点は、メッセージを交換できる相手が他の JMS アプリケーションのみであるという点です。 他の JMS メッセージ・タイプを選択することによって、JMS メッセージを JMS 以外のアプリケーションと交換できます。

177 ページの『JMSObjectMessage の送受信』に、メッセージで交換される String オブジェクトを示します。

JMS クライアント・アプリケーションは、 JMS スタイルの本体を持つメッセージでのみ JMSObjectMessage を受信できます。 宛先で JMS スタイルの本体を指定する必要があります。

JMSTextMessage

JMSTextMessage には、1 つのテキスト・ストリングが格納されます。 テキスト・メッセージを送信すると、テキスト Format が "MQSTR　　" の WMQConstants.MQFMT_STRING に設定されます。 テキストの CodedCharacterSetId は、宛先に定義されたコード化文字セット ID に設定されます。 テキストは、IBM MQ によって CodedCharacterSetId にエンコードされます。 CodedCharacterSetId および Format フィールドは、メッセージ記述子、 MQMD、または MQRFH2 の JMS フィールドのいずれかに設定されます。 メッセージの本体スタイルが WMQ_MESSAGE_BODY_MQ として定義されているか本体スタイルが指定されておらず、ターゲット宛先が WMQ_TARGET_DEST_MQ である場合、メッセージ記述子フィールドが設定されます。 それ以外の場合、メッセージには JMS RFH2 が含まれ、フィールドは MQRFH2 の固定部分に設定されます。

アプリケーションは、宛先に定義されているコード化文字セット ID を指定変更できます。 メッセージ・プロパティー JMS_IBM_CHARACTER_SET をコード化文字セット ID に設定する必要があります。177 ページの『JMSTextmessage の送受信』 の例を参照してください。

JMS クライアントが consumer.receive メソッドを呼び出す場合、キュー・マネージャー変換はオプションです。 キュー・マネージャー変換は、宛先プロパティー WMQ_RECEIVE_CONVERSION を WMQ_RECEIVE_CONVERSION_QMGR に設定すると有効になります。 キュー・マネージャーは、メッセージを JMS クライアントに転送する前に、メッセージに対して指定された JMS_IBM_CHARACTER_SET からテキスト・メッセージを変換します。 宛先に別の WMQ_RECEIVE_CCSID が設定されているのでない限り、変換後のメッセージの文字セットは 1208、UTF-8 です。 JMSTextMessage を参照するメッセージ内の CodedCharacterSetId は、ターゲットの文字セット ID に更新されます。 テキストは、getText メソッドによってターゲットの文字セットから Unicode にデコードされます。177 ページの『JMSTextmessage の送受信』 の例を参照してください。

JMSTextMessage は、MQ スタイルのメッセージ本体で JMS MQRFH2 ヘッダーなしで送信できます。 メッセージ本体のスタイルは、アプリケーションによって指定変更されない限り、宛先属性 WMQ_MESSAGE_BODY と WMQ_TARGET_DEST の値によって決まります。 アプリケーションは、destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) または destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) を呼び出すことにより、宛先に設定された値を指定変更できます。

MQ スタイルの本体で JMSTextMessage を宛先に送信するときに、WMQ_MESSAGE_BODY を WMQ_MESSAGE_BODY_MQ に設定した場合、メッセージを同じ宛先から JMSTextMessage として受信することはできません。 WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定された宛先から受信したすべてのメッセージは、JMSBytesMessage として受信されます。 メッセージを JMSTextMessage として受信しようとすると、例外 ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage が発生します。

注 : JMSBytesMessage 内のテキストは JMS クライアントによって変換されません。 クライアントは、メッセージ内のテキストをバイト配列としてのみ受信できます。 キュー・マネージャー変換が有効になっている場合、テキストはキュー・マネージャーによって変換されますが、JMS クライアントはそのテキストを JMSBytesMessage でバイト配列として受信する必要があります。

通常は、WMQ_TARGET_DEST プロパティーを使用して、JMSTextMessage の本体スタイルを MQ と JMS のどちらで送信するかを制御したほうがよいでしょう。 その後、WMQ_TARGET_DEST が WMQ_TARGET_DEST_MQ または WMQ_TARGET_DEST_JMS に設定された宛先からメッセージを受信できます。 WMQ_TARGET_DEST は受信側に影響を及ぼしません。

JMSMapMessage と JMSStreamMessage

これらの 2 つの JMS メッセージ・タイプは似ています。 DataInputStream および DataOutputStream インターフェースに基づくメソッドを使用して、プリミティブ型をメッセージから読み取ったりメッセージに書き込んだりできます。180 ページの『メッセージ・タイプと変換タイプの表』を参照してください。 詳細については、181 ページの『JMS クライアント・メッセージ変換とエンコード』で説明します。 各プリミティブには、タグが付いています。166 ページの『JMS メッセージ本体』を参照してください。

数値データは、XML テキストとしてエンコードされたメッセージに対して読み書きされます。 宛先プロパティー JMS_IBM_ENCODING に対する参照は行われません。 テキスト・データは、JMSTextMessage 内のテキストと同じように扱われます。 178 ページの図 20 の例で作成されたメッセージ内容を調べると、すべてのメッセージ・データは、文字セット値 37 で送信されたために EBCDIC になっています。

JMSMapMessage または JMSStreamMessage では複数の項目を送信できます。

JMSMapMessage で指定された名前または JMSStreamMessage で指定された位置によってデータの個々の項目を取得できます。 各項目は、メッセージに格納されている CodedCharacterSetId 値を使用して get メソッドまたは read メソッドを呼び出すと、デコードされます。 項目の取得に使用されたメソッドによって、送信されたタイプとは異なるタイプが返された場合、タイプは変換されます。 タイプを変換できない場合、例外がスローされます。 詳細については Class JMSStreamMessage を参照してください。 178 ページの『JMSStreamMessage および JMSMapMessage でのデータの送信』 の例は、型変換と、シーケンスからの JMSMapMessage の内容の取得を示しています。

JMSMapMessage および JMSStreamMessage の MQRFH2.format フィールドは "MQSTR "に設定されます。 宛先プロパティー WMQ_RECEIVE_CONVERSION が WMQ_RECEIVE_CONVERSION_QMGR に設定されている場合、メッセージ・データは JMS クライアントに送信される前にキュー・マネージャーによって変換されます。 メッセージの MQRFH2.CodedCharacterSetId は宛先の WMQ_RECEIVE_CCSID です。 MQRFH2.Encoding は Native です。 WMQ_RECEIVE_CONVERSION が WMQ_RECEIVE_CONVERSION_CLIENT_MSG である場合、MQRFH2 の CodedCharacterSetId と Encoding は送信側によって設定された値です。

JMS クライアント・アプリケーションは、 JMS スタイルの本体を持つメッセージ内、および MQ スタイルの本体を指定しない宛先からのみ、 JMSMapMessage または JMSStreamMessage を受け取ることができます。

JMSBytesMessage

JMSBytesMessage には複数のプリミティブ型を含めることができます。 DataInputStream および DataOutputStream インターフェースに基づくメソッドを使用して、プリミティブ型をメッセージから読み取ったりメッセージに書き込んだりできます。180 ページの『メッセージ・タイプと変換タイプの表』を参照してください。 詳細については、174 ページの『JMS メッセージ・タイプと変換』で説明します。

メッセージ内の数値データのエンコードは、数値データを JMSBytesMessage に書き込む前に設定された JMS_IBM_ENCODING の値によって制御されます。 アプリケーションは、メッセージ・プロパティー JMS_IBM_ENCODING を設定して JMSBytesMessage に対して定義されたデフォルトの Native エンコードをオーバーライドできます。

テキスト・データは、readUTF および writeUTF を使用して UTF-8 で読み書きするか、readChar メソッドおよび writeChar メソッドを使用して Unicode ストリングで読み書きすることができます。 CodedCharacterSetId を使用するメソッドはありません。 または、JMS クライアントは、Charset クラスを使用してテキストをバイトにエンコードおよびデコードできます。 IBM MQ classes for JMS で変換を実行せずに JVM とメッセージの間でバイトを転送します。178 ページの『JMSBytesMessage でのテキストの送受信』を参照してください。

MQ アプリケーションに送信された JMSBytesMessage は、通常、MQ スタイルのメッセージ本体で JMS MQRFH2 ヘッダーなしで送信されます。 JMS アプリケーションに送信された場合、メッセージ本体のスタイルは通常 JMS です。 メッセージ本体のスタイルは、アプリケーションによって指定変更されない限り、宛先属性 WMQ_MESSAGE_BODY と WMQ_TARGET_DEST の値によって決まります。 アプリケーションは、destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) または

destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) を呼び出すことにより、宛先に設定された値を指定変更できます。

MQ スタイルの本体で JMSBytesMessage を送信した場合、MQ または JMS メッセージの本体スタイルを定義する宛先からメッセージを受信できます。 JMS スタイルの本体で JMSBytesMessage を送信した場合、JMS メッセージの本体スタイルを定義する宛先からメッセージを受信する必要があります。 そうしなければ、MQRFH2 はユーザー・メッセージ・データの一部として処理され、予期しているものとは異なる場合があります。

メッセージの本体スタイルが MQ と JMS のいずれであっても、その受信方法は WMQ_TARGET_DEST の設定の影響を受けません。

メッセージ・データに Format が指定されていて、キュー・マネージャーのデータ変換が有効である場合、メッセージは後でキュー・マネージャーによって変換されることがあります。 メッセージ・データのフォーマットを指定する以外の目的でフォーマット・フィールドを使用しないでください。または空白 (MQConstants.MQFMT_NONE) のままにしておくことは可能です。

JMSBytesMessage で複数の項目を送信できます。 メッセージに定義されているエンコードを使用してメッセージが送信されるときに、各数値項目が変換されます。

JMSBytesMessage からデータの個々の項目を取得できます。 メッセージを作成するために write メソッドが呼び出されたのと同じ順序で read メソッドを呼び出します。 各数値項目は、メッセージに格納されている Encoding 値を使用してメッセージが呼び出されるときに変換されます。

JMSMapMessage や JMSStreamMessage とは異なり、JMSBytesMessage には、アプリケーションによって書き込まれたデータのみが含まれます。 メッセージ・データには、JMSMapMessage および JMSStreamMessage の項目の定義に使用される XML タグなどの追加データは格納されません。 そのため、他のアプリケーション用にフォーマットされたメッセージを転送するには、JMSBytesMessage を使用します。

JMSBytesMessage、DataInputStream、および DataOutputStream の間の変換は、一部のアプリケーションで役立ちます。 JMS で com.ibm.mq.header パッケージを使用するには、 179 ページの『DataInputStream および DataOutputStream を使用したメッセージの読み書き』の例に基づくコードが必要です。

例

JMSObjectMessage の送受信

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

図 *16. JMSObjectMessage* の送受信

JMSTextmessage の送受信

テキスト・メッセージに、異なる文字セットのテキストを含めることはできません。 例では、2 つの異なるメッセージに送信された、異なる文字セットのテキストを示します。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

図 *17.* 宛先で定義された文字セットのテキスト・メッセージの送信

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

図 *18. ccsid 37* のテキスト・メッセージの送信

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

図 *19.* テキスト・メッセージの受信

JMSStreamMessage および JMSMapMessage でのデータの送信

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                         " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

図 *20. JMSStreamMessage* および *JMSMapMessage* でのデータの送信

JMSBytesMessage でのテキストの送受信

179 ページの図 21 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 複数のタイプが混在したバイト・メッセージでテキスト・ストリングを受信するには、バイト単位のストリング長を把握する必要があります。これは、179 ページの図 22 にある *TEXT_LENGTH* という名前の変数に格納されます。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                        .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *21. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *22. JMSBytesMessage* からの *String* の受信

DataInputStream および DataOutputStream を使用したメッセージの読み書き

179 ページの図 23 のコードは、DataOutputStream を使用して JMSBytesMessage を作成します。

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

図 *23. DataOutputStream* を使用した *JMSBytesMessage* の送信

JMS_IBM_ENCODING プロパティーを設定するステートメントがコメント化されています。
JMSBytesMessage に直接書き込む場合、ステートメントは有効ですが、DataOutputStream に書き込む場合は有効ではありません。 DataOutputStream に書き込まれる数値は、Native エンコードでエンコードされます。 JMS_IBM_ENCODING を設定しても効果はありません。

180 ページの図 24 のコードは、DataInputStream を使用して JMSBytesMessage を受信します。

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

図 *24. DataInputStream* を使用した *JMSBytesMessage* の受信

コード・ページは、入力メッセージ・データのコード・ページ・プロパティー JMS_IBM_CHARACTER_SET を使用して印刷されます。 入力では、JMS_IBM_CHARACTER_SET は Java コード・ページであり、コード化数字セット ID ではありません。

メッセージ・タイプと変換タイプの表

表 *32.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**テキスト**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

表 32. メッセージ・タイプと変換タイプ (続き)				
	変換タイプ			
メッセージ・タイプ	**テキスト**	**数字**	**その他**	**なし**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

関連概念

JMS メッセージ変換のアプローチ

JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、IBM MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード

JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換

キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 メッセージを受信する JMS クライアントも、オプションのキュー・マネージャー・データ変換を使用します。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換

JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できるデータ変換出口、および JMS クライアント・アプリケーションを設計して作成するには、このタスクで推奨されるステップに従ってください。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

JMS クライアント・メッセージ変換とエンコード

JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

変換とエンコードは、Java プリミティブまたはオブジェクトを JMS メッセージに対して読み書きする場合に行われます。 この変換は JMS クライアント・データ変換と呼ばれ、キュー・マネージャー・データ変換やアプリケーション・データ変換とは区別されます。 変換は、JMS メッセージに対してデータが読み書きされる場合にのみ行われます。 テキストは、内部の 16 ビット Unicode 表記との間で変換されます。[3] メッセージ内のテキストに使用されている文字セットに変換されます。 数値データは、Java プリミティブ数値型に変換され、メッセージに定義されているエンコードに変換されます。 変換を実行するかどうか、および実行する変換のタイプは、JMS メッセージ・タイプと読み取り操作や書き込み操作によって異なります。

182 ページの表 33 は、さまざまな JMS メッセージ・タイプの read メソッドと write メソッドを、実行される変換のタイプに分類したものです。 変換タイプについては、表の後で説明します。

表 *33.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**テキスト**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar

[3] 一部の Unicode 表記では、16 ビット以上が必要です。 Java SE のリファレンスを参照してください。

表 33. メッセージ・タイプと変換タイプ (続き)				
	変換タイプ			
メッセージ・タイプ	テキスト	数字	その他	なし
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

テキスト

宛先のデフォルトの CodedCharacterSetId は 1208、UTF-8 です。 デフォルトでは、テキストは Unicode から変換され、UTF-8 テキスト・ストリングとして送信されます。 受信時に、テキストは、クライアントが受信したメッセージのコード化文字セットから Unicode に変換されます。

setText および writeString メソッドによって、テキストは Unicode から宛先に定義されている文字セットに変換されます。 アプリケーションは、メッセージ・プロパティー JMS_IBM_CHARACTER_SET を設定して、宛先の文字セットをオーバーライドできます。 メッセージを送信するとき、JMS_IBM_CHARACTER_SET は、数字で構成されるコード化文字セット ID である必要があります [4].

185 ページの『JMSTextmessage の送受信』 のコード・スニペットは、2 つのメッセージを送信します。 1 つは宛先に定義されている文字セットで送信され、もう 1 つはアプリケーションによって定義された文字セット 37 で送信されます。

getText および readString メソッドは、メッセージのテキストをメッセージに定義されている文字セットから Unicode に変換します。 これらのメソッドは、メッセージ・プロパティー JMS_IBM_CHARACTER_SET に定義されているコード・ページを使用します。 コード・ページは、MQRFH2.CodedCharacterSetId からマップされますが、メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合を除きます。 メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合、コード・ページは MQMD.CodedCharacterSetId からマップされます。

186 ページの図 29 のコード・スニペットは、宛先に送信されたメッセージを受信します。 メッセージ内のテキストはコード・ページ IBM037 から Unicode に再び変換されます。

注 : テキストがコード化文字セット 37 に変換されたことを簡単に確認するには、IBM MQ エクスプローラーを使用します。 キューを参照し、受信前にメッセージのプロパティーを表示します。

186 ページの図 28 のコード・スニペットを 184 ページの図 25 の正しくないコード・スニペットと比較してください。 正しくないスニペットでは、テキスト・ストリングは 2 回変換されています。1 回目はアプリケーションによって、2 回目は IBM MQ によって変換されています。

[4] メッセージを受信すると、 JMS_IBM_CHARACTER_SET は Java Charset コード・ページ名になります。

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

図 *25.* 正しくないコード・ページ変換

writeUTF メソッドは、テキストを Unicode から 1208、UTF-8 に変換します。 テキスト・ストリングの前には、2 バイトの長さが付加されます。 テキスト・ストリングの最大長は 65534 バイトです。 readUTF メソッドは、writeUTF メソッドによって書き込まれたメッセージ内の項目を読み取ります。 それは writeUTF メソッドによって書き込まれたバイトの数を正確に読み取ります。

数字

宛先のデフォルトの数値エンコードは Native です。 Java の Native エンコード定数の値は 273 (x'00000111') で、これはすべてのプラットフォームで同じです。 受信時に、メッセージ内の数値は数値の Java プリミティブに正しく変換されます。 変換では、メッセージに定義されているエンコードと read メソッドによって返されたタイプが使用されます。

send メソッドは、set と write によってメッセージに追加された数値を、宛先に定義されている数値エンコードに変換します。 メッセージの宛先のエンコードは、アプリケーション設定のメッセージ・プロパティー JMS_IBM_ENCODING によってオーバーライドできます。 例えば、次のとおりです。

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

数値メソッドの get と read は、メッセージ内の数値を、メッセージに定義されている数値エンコードから変換します。 数値を、read または get メソッドによって指定されているタイプに変換します。 ENCODING プロパティーを参照してください。 メソッドでは、JMS_IBM_ENCODING で定義されているエンコードが使用されます。 エンコードは、MQRFH2.Encoding からマップされますが、メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合を除きます。 メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合、メソッドでは、MQMD.Encoding で定義されているエンコードが使用されます。

186 ページの図 30 の例は、宛先の形式で数値をエンコードし、それを JMSStreamMessage で送信するアプリケーションを示しています。 186 ページの図 30 の例を 186 ページの図 31 の例と比較してください。 違いは、JMS_IBM_ENCODING を JMSBytesMessage で設定する必要があるという点です。

注 : 数値が正しくエンコードされたことを簡単に確認するには、IBM MQ エクスプローラーを使用します。 キューを参照し、コンシューム前にメッセージのプロパティーを表示します。

その他

boolean メソッドは、JMSByteMessage、JMSStreamMessage、および JMSMapMessage で true および false を x'01' および x'00' としてエンコードします。

UTF メソッドは Unicode を UTF-8 テキスト・ストリングにエンコードおよびデコードします。 ストリングは 65536 文字未満に制限され、2 バイトの長さフィールドがストリングの前に付加されます。

Object メソッドは、プリミティブ型をオブジェクトとしてラップします。 数値型とテキスト型は、プリミティブ型が数値メソッドとテキスト・メソッドを使用して読み書きされる場合と同じようにエンコードまたは変換されます。

なし

readByte、readBytes、readUnsignedByte、writeByte、および writeBytes メソッドは、1 バイトまたはバイト配列をアプリケーションとメッセージ間で変換なしに取得または書き込みます。 readChar および writeChar メソッドは、2 バイトの Unicode 文字をアプリケーションとメッセージ間で変換なしに取得および書き込みます。

readBytes および writeBytes メソッドを使用すると、アプリケーションは、187 ページの『JMSBytesMessage でのテキストの送受信』と同様に独自のコード・ポイント変換を実行できます。

IBM MQ は、メッセージが JMSBytesMessage で、readBytes および writeBytes メソッドが使用されるため、クライアントでコード・ページ変換を実行しません。 ただし、バイトがテキストを表す場合は、アプリケーションによって使用されるコード・ページが宛先のコード化文字セットに一致することを確認してください。 メッセージは、キュー・マネージャーの変換出口によって再び変換される場合があります。 別の可能性として、受信側の JMS クライアント・プログラムは、メッセージ内の JMS_IBM_CHARACTER_SET プロパティーを使用して、メッセージ内のテキストを表すバイト配列をストリングまたは文字に変換する規則に従うことがあります。

この例では、クライアントは宛先のコード化文字セットを変換に使用します。

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

あるいは、クライアントは、コード・ページを選択してから、対応するコード化文字セットをメッセージの JMS_IBM_CHARACTER_SET プロパティーに設定した可能性があります。 IBM MQ classes for Java は、 JMS_IBM_CHARACTER_SET を使用して、 MQRFH2 またはメッセージ記述子 MQMD の JMS プロパティーに CodedCharacterSetId フィールドを設定します。

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

バイト配列が JMSStringMessage または JMSMapMessage に書き込まれると、IBM MQ classes for JMS はバイトが、JMSStringMessage および JMSMapMessage 内のテキストではなく 16 進データとして入力されるため、データ変換を実行しません。

アプリケーションでバイトが文字を表す場合は、メッセージに対して読み書きするコード・ポイントを考慮する必要があります。 185 ページの図 26 のコードは、宛先のコード化文字セットの使用規則に従っています。 JVM のデフォルトの文字セットを使用してストリングを作成する場合、バイト内容はプラットフォームによって異なります。 Windows 上の JVM では、通常、デフォルトの Charset は windows-1252 で、AIX and Linux では UTF-8 です。 Windows と AIX and Linux 間の交換では、テキストをバイトとして交換するために明示的なコード・ページを選択する必要があります。

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

図 *26.* 宛先の文字セットを使用した、*JMSStreamMessage* へのストリングを表すバイトの書き込み

例

JMSTextmessage の送受信

テキスト・メッセージに、異なる文字セットのテキストを含めることはできません。 例では、2 つの異なるメッセージに送信された、異なる文字セットのテキストを示します。

[5] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

図 *27.* 宛先で定義された文字セットのテキスト・メッセージの送信

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

図 *28. ccsid 37* のテキスト・メッセージの送信

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

図 *29.* テキスト・メッセージの受信

エンコードの例

以下の例では、宛先に定義されたエンコードでの数値の送信を示しています。 JMSBytesMessage の JMS_IBM_ENCODING プロパティーを、宛先に指定された値に設定する必要があります。

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

図 *30. JMSStreamMessage* での宛先のエンコードを使用した数値の送信

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

図 *31. JMSBytesMessage* での宛先のエンコードを使用した数値の送信

JMSBytesMessage でのテキストの送受信

187 ページの図 32 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 複数のタイプが混在したバイト・メッセージでテキスト・ストリングを受信するには、バイト単位のストリング長を把握する必要があります。これは、187 ページの図 33 にある *TEXT_LENGTH* という名前の変数に格納されます。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                      .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *32. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *33. JMSBytesMessage* からの *String* の受信

関連概念

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、IBM MQ によって自動的に実行されます。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 メッセージを受信する JMS クライアントも、オプションのキュー・マネージャー・データ変換を使用します。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できるデータ変換出口、および JMS クライアント・アプリケーションを設計して作成するには、このタスクで推奨されるステップに従ってください。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 メッセージを受信する JMS クライアントも、オプションのキュー・マネージャー・データ変換を使用します。

キュー・マネージャーは、メッセージ・データに設定されている CodedCharacterSetId、Encoding、および Format の値を使用して、メッセージ・データ内の文字および数値データを変換できます。 JMS 以外のアプリケーションでは、GetMessageOption と GMO_CONVERT を設定することによって変換機能を常に使用できます。

キュー・マネージャーは、JMS クライアントに送信されるメッセージを変換できます。 キュー・マネージャーの変換は、宛先プロパティー WMQ_RECEIVE_CONVERSION を WMQ_RECEIVE_CONVERSION_QMGR または WMQ_RECEIVE_CONVERSION_CLIENT_MSG に設定することによって制御されます。 アプリケーションで宛先の設定を変更できます。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

または

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

図 *34.* キュー・マネージャー・データ変換の有効化

JMS クライアントのキュー・マネージャー・データ変換は、クライアントが consumer.receive メソッドを呼び出すときに行われます。 テキスト・データは、デフォルトでは UTF-8 (1208) に変換されます。 後続の read メソッドと get メソッドは、内部の Unicode エンコードで Java テキスト・プリミティブを作成して、UTF-8 からの受信データ内のテキストをデコードします。 UTF-8 は、キュー・マネージャー・データ変換からの唯一のターゲット文字セットではありません。 WMQ_RECEIVE_CCSID 宛先プロパティーを設定して、別の CCSID を選択できます。

アプリケーションでは、宛先設定を例えば 437 の DOS-US に設定して変更することもできます。

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

または

```
((MQDestination)destination).setReceiveCCSID(437);
```

図 *35.* キュー・マネージャー変換のターゲットコード化文字セットの設定

WMQ_RECEIVE_CCSID を変更することには特殊な理由があります。 選択された CCSID によって、JVM で作成されたテキスト・オブジェクトに違いが生じることはありません。 ただし、一部の JVM では、プラットフォームによってメッセージ内のテキストの CCSID から Unicode への変換を処理できない場合があります。 オプションにより、メッセージでクライアントに送信されるテキストの CCSID を選択できます。 一部の JMS クライアント・プラットフォームには、UTF-8 で送信されるメッセージ・テキストに関する問題がありました。

JMS コードは、189 ページの図 36 の C コードの太字テキストと同じです。

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

図 *36. amqsget0.c* からのコード・スニペット

注 :

キュー・マネージャー変換は、既知の IBM MQ 形式のメッセージ・データに対してのみ実行されます。 MQSTR、 または MQCIH は、事前定義されている既知の形式の例です。 データ変換出口を指定した場合に限り、既知のフォーマットにユーザー定義のフォーマットを使用することもできます。

JMSTextMessage、JMSMapMessage、および JMSStreamMessage として構成されたメッセージは、MQSTR フォーマットで、キュー・マネージャーによって変換できます。

関連概念

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、IBM MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

989 ページの『データ変換出口の起動』
データ変換出口とは、MQGET 呼び出しの処理中に制御を受け取るユーザー作成出口です。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できるデータ変換出口、および JMS クライアント・アプリケーションを設計して作成するには、このタスクで推奨されるステップに従ってください。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できるデータ変換出口、および JMS クライアント・アプリケーションを設計して作成するには、このタスクで推奨されるステップ

に従ってください。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

始める前に

JMSTextMessage を使用して、JMS 以外のアプリケーションとメッセージを交換するためのより簡単なソリューションを設計できる場合があります。 このタスクの手順に従う前に、その可能性を排除します。

このタスクについて

JMS クライアントは、他の JMS クライアントと交換される JMS メッセージのフォーマットの詳細に関与しない場合は、簡単に作成できます。 メッセージ・タイプが JMSTextMessage、JMSMapMessage、JMSStreamMessage、または JMSObjectMessage である限り、IBM MQ はメッセージのフォーマットの詳細に注意を払います。 IBM MQ は、各種プラットフォームにおけるコード・ページと数値エンコードの相違を処理します。

これらのメッセージ・タイプを使用して、メッセージを JMS 以外のアプリケーションと交換できます。 そのためには、これらのメッセージが IBM MQ classes for JMS によってどのように構成されているかを理解する必要があります。 メッセージを解釈するように JMS 以外のアプリケーションを変更することもできます。 150 ページの『JMS メッセージの IBM MQ メッセージへのマッピング』を参照してください。

これらのいずれかのメッセージ・タイプを使用する利点は、JMS クライアント・プログラミングがメッセージを交換するアプリケーションのタイプに依存しないという点です。 欠点は、別のプログラムの変更が必要になる場合がありますが、他のプログラムを変更できないという点です。

その代わりに、既存のメッセージ・フォーマットを処理できる JMS クライアント・アプリケーションを作成する方法があります。 多くの場合、既存のメッセージは固定フォーマットであり、不定形式データ、テキスト、および数値が混在しています。 JMS 以外のアプリケーションとフォーマット済みレコードを交換できる JMS クライアントを作成するための開始点として、このタスクのステップ、および 193 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』の JMS クライアントの例を使用してください。

手順

1. レコード・レイアウトを定義するか、事前定義された IBM MQ ヘッダー・クラスのいずれかを使用します。

   事前定義された IBM MQ ヘッダーを処理するには、「IBM MQ メッセージ・ヘッダーの処理」を参照してください。

   191 ページの図 37 は、データ変換ユーティリティーで処理できるユーザー定義の固定長レコード・レイアウトの例です。
2. データ変換出口を作成します。

   「データ変換出口プログラムの作成」の指示に従ってデータ変換出口を作成します。

   193 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』の例を試すには、データ変換出口に MYRECORD という名前を付けます。
3. Java クラスを作成して、レコード・レイアウトとレコードの送受信をカプセル化します。 次の 2 つの方法があります。

   - レコードが含まれる JMSBytesMessage を読み書きするクラスを作成します。 193 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』を参照してください。
   - レコードのデータ構造を定義するために com.ibm.mq.header.Header を拡張するクラスを書き込みます。「新規ヘッダー・タイプのクラスの作成」を参照してください。
4. 交換するメッセージのコード化文字セットを決定します。

   メッセージ変換のアプローチの選択: 「受信側で実行」を参照してください。
5. JMS MQRFH2 ヘッダーなしで MQ タイプのメッセージを交換する宛先を構成します。

送信側の宛先と受信側の宛先の両方を、MQ タイプのメッセージを交換するように構成する必要があります。 送信側と受信側の両方に同じ宛先を使用できます。

アプリケーションで宛先のメッセージ本体プロパティーをオーバーライドできます。

```
((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

193 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』の例では、宛先のメッセージ本体プロパティーをオーバーライドして、MQ スタイルのメッセージが送信されるようにしています。

6. JMS と JMS 以外のアプリケーションでソリューションをテストします。

   データ変換出口のテストに便利なツールは、次のとおりです。

   - amqsgetc0.c サンプル・プログラムは、JMS クライアントによって送信されたメッセージの受信をテストするのに役立ちます。 推奨される変更内容を参照して、192 ページの図 38 の例のヘッダー RECORD.h を使用します。 この変更により、amqsgetc0.c は、JMS クライアントの例 TryMyRecord.java によって送信されたメッセージを受け取ります。193 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』を参照してください。
   - サンプル IBM MQ ブラウズ・プログラム amqsbcg0.c は、メッセージ・ヘッダーの内容、JMS ヘッダー、MQRFH2、およびメッセージ内容を調べるのに役立ちます。
   - **rfhutil** プログラム (以前は SupportPac IH03 で使用可能) を使用すると、テスト・メッセージをキャプチャーしてファイルに保管し、メッセージ・フローを駆動するために使用することができます。 出力メッセージをさまざまな形式で読み取ったり表示したりすることもできます。 この形式には、2 つのタイプの XML と、COBOL コピーブックに対する突き合わせも含まれます。 データは EBCDIC または ASCII のいずれかを使用します。 メッセージを送信する前に、RFH2 ヘッダーをメッセージに追加することができます。

   変更された amqsgetc0.c サンプル・プログラムを使用してメッセージを受信しようとして理由コード 2080 のエラーが発生した場合は、メッセージに MQRFH2 があるかどうかを確認します。 変更では、MQRFH2 が指定されていないメッセージが宛先に送信されたことを想定しています。

例

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

図 *37. RECORD.h*

- RECORD.h データ構造の宣言

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- RECORD, を使用するように MQGET 呼び出しを変更します。

  1. 変更前:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. 変更後:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- print ステートメントを変更します。

  1. 変更前:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. 変換後:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

図 *38. amqsget0.c* の変更

関連概念

JMS メッセージ変換のアプローチ

JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、IBM MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 メッセージを受信する JMS クライアントも、オプションのキュー・マネージャー・データ変換を使用します。

変換出口コードを作成するためのユーティリティー

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成
このタスクの目的は、JMSBytesMessage でデータ変換と固定レコード・レイアウトを結合する方法を、例を示して検討することです。 タスクでは、いくつかの Java クラスを作成して、JMSBytesMessage でサンプル・レコード構造体を交換します。 例を変更して、他のレコード構造を交換するクラスを作成できます。

JMSBytesMessage は、JMS 以外のプログラムと混合データ型レコードを交換するための最適な JMS メッセージ・タイプです。 JMS プロバイダーによってメッセージ本体に挿入される追加データはありません。 したがって、JMS クライアント・プログラムが既存の IBM MQ プログラムと相互運用する場合は、これが最適なメッセージ・タイプです。 JMSBytesMessage を使用する場合の主な課題は、他のプログラムに必要なエンコードと文字セットのマッチングです。 解決策は、レコードをカプセル化するクラスの作成です。 特定のレコード・タイプについて、JMSBytesMessage の読み書きをカプセル化するクラスを使用すると、固定形式のレコードを JMS プログラムで送受信しやすくなります。 抽象クラスにインターフェースの汎用的な側面を取り込むことによって、解決策の多くをさまざまなレコード・フォーマットに再利用できます。 抽象汎用クラスを拡張するクラスにさまざまなレコード・フォーマットを実装できます。

代替手段として、com.ibm.mq.headers.Header クラスを拡張します。 Header クラスには、より宣言的な方法でレコード・フォーマットを作成する addMQLONG などのメソッドがあります。 Header クラスを使用した場合の欠点は、属性の取得と設定で複雑な解釈インターフェースが使用される点です。 どちらの方法でも、アプリケーション・コードの量は同じくらいになります。

JMSBytesMessage は、MQRFH2 に加えて 1 つのフォーマットだけを、1 つのメッセージにカプセル化します。 ただし、各レコードで同じフォーマット、コード化文字セット、およびエンコードが使用される場合を除きます。 JMSBytesMessage のフォーマット、エンコード、および文字セットは、MQRFH2 に続く、すべてのメッセージのプロパティーです。 例は、JMSBytesMessage にユーザー・レコードが 1 つのみ含まれていることを想定して作成されています。

始める前に

1. スキル・レベル: Java プログラミングと JMS を理解している必要があります。 Java 開発環境の設定に関する指示はありません。 JMSTextMessage、JMSStreamMessage、または JMSMapMessage を交換するようにプログラムを作成しておくと良いでしょう。 そうすれば、JMSBytesMessage を使用したメッセージ交換の違いを確認できます。
2. 例では IBM WebSphere MQ 7.0 が必要です。
3. 例は、Eclipse ワークベンチの Java パースペクティブを使用して作成されました。 これには JRE 6.0 以上が必要です。 IBM MQ エクスプローラーの Java パースペクティブを使用して、Java クラスを開発および実行できます。 または、独自の Java 開発環境を使用することもできます。
4. IBM MQ エクスプローラーを使用すると、テスト環境の設定とデバッグをコマンド行ユーティリティーを使用する場合より簡単に実行できます。

このタスクについて

2 つのクラス、RECORD と MyRecord の作成方法についてガイドします。 これらの 2 つのクラスで固定形式のレコードがカプセル化されます。 これらのクラスには、属性を取得および設定するメソッドがあります。 get メソッドは JMSBytesMessage からレコードを読み取り、put メソッドはレコードを JMSBytesMessage に書き込みます。

このタスクの目的は、再利用できる実動品質のクラスを作成することではありません。 タスクの例を使用して、独自のクラスの作成に取り掛かることもできます。 このタスクの目的は、JMSBytesMessage を使用する際の、主に文字セット、フォーマット、およびエンコードの使用方法に関するガイダンスを示すことです。 クラス作成の各ステップについて説明し、見逃されることがある JMSBytesMessage の使用の側面について説明します。

RECORD クラスは抽象クラスであり、ユーザー・レコードの共通フィールドをいくつか定義します。 共通フィールドは、目印、バージョン、および長さフィールドを持つ標準の IBM MQ ヘッダー・レイアウトでモデル化されます。 多くの IBM MQ ヘッダーにあるエンコード、文字セット、およびフォーマット・フィールドは省略されます。 別のヘッダーは、ユーザー定義のフォーマットに従うことはできません。
RECORD クラスを拡張する MyRecord クラスは、追加のユーザー・フィールドでレコードを拡張してユーザー定義のフォーマットに従います。 クラスによって作成された JMSBytesMessage は、キュー・マネージャーのデータ変換出口によって処理できます。

200 ページの『例の実行に使用されるクラス』には、RECORD と MyRecord の完全なリストがあります。 RECORD と MyRecord をテストするための、追加の"足場"となるクラスのリストもあります。 追加のクラスは次のとおりです。

TryMyRecord
RECORD と MyRecord をテストするメインプログラムです。

EndPoint
JMS 接続、宛先、およびセッションを 1 つのクラスにカプセル化する抽象クラスです。 そのインターフェースは、RECORD と MyRecord クラスのテストのニーズを満たします。 それは JMS アプリケーションを作成するための確立された設計パターンではありません。

注: Endpoint クラスには、宛先の作成後に次のコード行が含まれます。

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

V7.0 では、V7.0.1.5 から、キュー・マネージャー変換を有効にする必要があります。 デフォルトでは無効になっています。 V7.0 の V7.0.1.4 までは、キュー・マネージャー変換はデフォルトで有効になっており、このコード行はエラーになります。

MyProducer と MyConsumer
EndPoint を拡張し、接続された、要求の受け入れ準備ができている MessageConsumer と MessageProducer を作成するクラスです。

すべてのクラスによって、JMSBytesMessage でのデータ変換の使用方法を理解するために構築および試すことのできる完全なアプリケーションが構成されます。

手順

1. デフォルトのコンストラクターで標準フィールドを IBM MQ ヘッダーにカプセル化する抽象クラスを作成します。 後で、このクラスを拡張して、要件に合わせてヘッダーを調整できます。

```
public abstract class RECORD implements Serializable {
   private static final long serialVersionUID = -1616617232750561712L;
   protected final static int UTF8 = 1208;
   protected final static int MQLONG_LENGTH = 4;
   protected final static int RECORD_STRUCT_ID_LENGTH = 4;
   protected final static int RECORD_VERSION_1 = 1;
   protected final String RECORD_STRUCT_ID = "BLNK";
   protected final String RECORD_TYPE = "BLANK   ";
   private String structID = RECORD_STRUCT_ID;
```

```
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
```

注:

a. 属性 structID から nextFormat までは、標準の IBM MQ メッセージ・ヘッダーに配置されている順序でリストされています。

b. 属性 format、messageEncoding、および messageCharset は、ヘッダー自体を記述し、ヘッダーの一部ではありません。

c. レコードのコード化文字セット ID を格納するか、文字セットを格納するかを決定する必要があります。 Java では文字セットを使用し、IBM MQ メッセージではコード化文字セット ID を使用します。 例のコードでは文字セットが使用されています。

d. int は、 IBM MQ によって MQLONG にシリアライズされます。 MQLONG は 4 バイトです。

2. 専用属性の getter と setter を作成します。

a) getter を作成または生成します。

```
     public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }
```

b) setter を作成または生成します。

```
     public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

3. JMSBytesMessage から RECORD インスタンスを作成するコンストラクターを作成します。

```
  public RECORD(BytesMessage message) throws JMSException, IOException,
         MQDataException {
     super();
     setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
     setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
     byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
     message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
     setStructID(new String(structID, getMessageCharset()));
     setVersion(message.readInt());
     setStructLength(message.readInt());
 }
```

注:

a. messageCharset と messageEncoding は、宛先に設定された値をオーバーライドするため、メッセージ・プロパティーから取り込まれます。 format は更新されません。 この例では、エラー・チェックは実行されません。 Record(BytesMessage) コンストラクターが呼び出される場合、JMSBytesMessage は RECORD タイプのメッセージと見なされます。 ライン "setStructID(new String(structID, getMessageCharset()))" は目印を設定します。

b. メソッドを完了するコード行は、メッセージ内のフィールドを順番にデシリアライズし、RECORD インスタンスに設定されたデフォルト値を更新します。

4. JMSBytesMessage にヘッダー・フィールドを書き込む put メソッドを作成します。

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
      JMSException, UnsupportedEncodingException {
   setHeaderEncoding(myProducer.getEncoding());
   setHeaderCharset(myProducer.getCharset());
   myProducer.setMQClient(true);
   BytesMessage bytes = myProducer.session.createBytesMessage();
   bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
   bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
   bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                        myProducer.getCCSID());
   bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                    + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                    .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
   bytes.writeInt(getVersion());
   bytes.writeInt(getStructLength());
   return bytes;
}
```

注:

a. MyProducer は、JMS Connection、Destination、Session、および MessageProducer を 1 つのクラスにカプセル化します。 後で使用される MyConsumer は、JMS Connection、Destination、Session、および MessageConsumer を 1 つのクラスにカプセル化します。

b. JMSBytesMessage では、エンコードが Native 以外の場合、メッセージ内にエンコードを設定する必要があります。 宛先のエンコードがメッセージ・エンコード属性 JMS_IBM_CHARACTER_SET にコピーされ、RECORD クラスの属性として保存されます。

i) "setMessageEncoding(myProducer.getEncoding());" は "(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));" を呼び出し、宛先エンコードを取得します。

ii) "Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());" はメッセージ・エンコードを設定します。

c. テキストをバイトに変換するために使用される文字セットは、宛先から取得され、RECORD クラスの属性として保存されます。 これは、 JMSBytesMessage の作成時に IBM MQ classes for JMS によって使用されないため、メッセージには設定されません。

"messageCharset = myProducer.getCharset();" 呼び出し

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

コード化文字セット ID から Java 文字セットを取得します。

" CCSID.getCodepage(ccsid)" は、パッケージ com.ibm.mq.headers にあります。 ccsid は、宛先を照会する MyProducer で別のメソッドから取得されます。

```
public int getCCSID() throws JMSException {
    return (((MQDestination) destination)
```

```
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. "myProducer.setMQClient(true);" は、クライアント。タイプの宛先設定をオーバーライドし、それを IBM MQ MQI client に強制します。 管理構成エラーが覆い隠されるため、このコード行を省略することをお勧めします。

"myProducer.setMQClient(true);" は以下を呼び出します。

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

コードには、JMS の設定をオーバーライドする必要がある場合に、IBM MQ 本体のスタイルを未指定に設定するという副次作用があります。

注:

IBM MQ classes for JMS は、メッセージのフォーマット、エンコード、および文字セット ID をメッセージ記述子 MQMD または JMS ヘッダー MQRFH2 に書き込みます。 これは、メッセージに IBM MQ スタイルの本体が含まれているかどうかによって異なります。 MQMD フィールドを手動で設定しないでください。

メッセージ記述子のプロパティーを手動で設定するメソッドが存在します。 このメソッドは JMS_IBM_MQMD_* プロパティーを使用します。 宛先プロパティー WMQ_MQMD_WRITE_ENABLED を設定して JMS_IBM_MQMD_* プロパティーを設定する必要があります。

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

宛先プロパティー WMQ_MQMD_READ_ENABLED を設定してプロパティーを読み取る必要があります。

メッセージ・ペイロード全体を完全に制御する場合にのみ JMS_IBM_MQMD_* を使用します。 JMS_IBM_* プロパティーとは異なり、JMS_IBM_MQMD_* プロパティーは IBM MQ classes for JMS が JMS メッセージを構成する方法は制御しません。 JMS メッセージのプロパティーと競合するメッセージ記述子プロパティーを作成する可能性があります。

e. メソッドを完了するコード行は、クラス内の属性をメッセージ内のフィールドとしてシリアライズします。

ストリング属性にはブランクが埋め込まれます。 ストリングは、レコードに定義されている文字セットを使用してバイトに変換され、メッセージ・フィールドの長さに切り捨てられます。

5. インポートを追加してクラスを完成させます。

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. RECORD クラスを拡張して追加フィールドを含めるクラスを作成します。 デフォルトのコンストラクターを含めます。

```
public class MyRecord extends RECORD {
   private static final long serialVersionUID = -370551723162299429L;
   private final static int FLAGS = 1;
   private final static String STRUCT_ID = "MYRD";
   private final static int DATA_LENGTH = 32;
   private final static String FORMAT = "MYRECORD";
   private int flags = FLAGS;
   private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";
```

```
public MyRecord() {
    super();
    super.setStructID(STRUCT_ID);
    super.setHeaderFormat(FORMAT);
    super.setStructLength(super.getStructLength() + MQLONG_LENGTH
            + DATA_LENGTH);
}
```

注:

a. RECORD サブクラスの MyRecord は、ヘッダーの目印、フォーマット、および長さをカスタマイズします。

7. getter および setter を作成または生成します。

a) getter を作成します。

```
 public int getFlags() { return flags; }
public String getRecordData() { return recordData; } .
```

b) setter を作成します。

```
 public void setFlags(int flags) {
    this.flags = flags; }
public void setRecordData(String recordData) {
    this.recordData = recordData; }
}
```

8. JMSBytesMessage から MyRecord インスタンスを作成するコンストラクターを作成します。

```
public MyRecord(BytesMessage message) throws JMSException, IOException,
       MQDataException {
   super(message);
   setFlags(message.readInt());
   byte[] recordData = new byte[DATA_LENGTH];
   message.readBytes(recordData, DATA_LENGTH);
   setRecordData(new String(recordData, super.getMessageCharset()));
}
```

注:

a. 標準のメッセージ・テンプレートを構成するフィールドは、まず RECORD クラスによって読み取られます。

b. recordData テキストは、メッセージの文字セット・プロパティーを使用して String に変換されます。

9. コンシューマーからメッセージを取得し、新しい MyRecord インスタンスを作成する静的メソッドを作成します。

```
public static MyRecord get(MyConsumer myConsumer) throws JMSException,
       MQDataException, IOException {
   BytesMessage message = (BytesMessage) myConsumer.receive();
   return new MyRecord(message);
}
```

注:

a. 例では、簡潔にするために、MyRecord(BytesMessage) コンストラクターを静的 get メソッドから呼び出しています。 通常は、メッセージの受信を新しい MyRecord インスタンスの作成と分離できます。

10. カスタマー・フィールドをメッセージ・ヘッダーが含まれる JMSBytesMessage に追加する put メソッドを作成します。

```
 public BytesMessage put(MyProducer myProducer) throws JMSException,
        IOException {
    BytesMessage bytes = super.put(myProducer);
    bytes.writeInt(getFlags());
    bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                    + DATA_LENGTH + "s",getRecordData())
                    .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
    myProducer.send(bytes);
    return bytes;
}
```

注:

a. コード内のメソッド呼び出しは、MyRecord クラス内の属性をメッセージ内のフィールドとしてシリアライズします。

- recordData String 属性にはブランクが埋め込まれ、レコードに定義されている文字セットを使用してバイトに変換されて、RecordData フィールドの長さに切り捨てられます。

11. include ステートメントを追加してクラスを完成させます。

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

タスクの結果

- TryMyRecord クラスの実行結果を次に示します。
  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用します。

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用しません。

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- メッセージを受信しないように TryMyRecord クラスを変更し、代わりに変更した amqsget0.c サンプルを使用してメッセージを受信した結果は、次のとおりです。 変更されたサンプルは、定様式レコードを受け入れます。 189 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』の 192 ページの図 38 を参照してください。
  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用します。

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用しません。

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

例を別のコード・ページとデータ変換出口で試すには、次のようにします。 Java クラスを作成し、IBM MQ を構成して、メインプログラム TryMyRecord を実行します。 200 ページの『#unique_196/unique_196_Connect_42_Try』を参照してください。

1. 例を実行するように IBM MQ と JMS を構成します。 例を Windows で実行するための手順を示します。
   a. キュー・マネージャーの作成

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. キューを作成します。

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. JNDI ディレクトリーを作成します。

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. JMS bin ディレクトリーに切り替えます。

   JMS 管理プログラムをここから実行する必要があります。 パスは *MQ_INSTALLATION_PATH*\java\bin です。

   e. JMSQM1Q1.txt というファイル内に以下の JMS 定義を作成します。

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. JMSAdmin プログラムを実行して JMS リソースを作成します。

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. IBM MQ エクスプローラーを使用して作成した定義を、作成、変更、および参照できます。
3. TryMyRecord を実行します。

例の実行に使用されるクラス

以下のコード・ブロックにリストされているクラスは、圧縮ファイルでも使用できます。 jm25529_.zip または jm25529_.tar.gz をダウンロードします。

TryMyRecord

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
```

```
    }
}
```

RECORD

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
```

```
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
```

```
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
```

```
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
```

```
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

JM 3.0 V 9.3.0 V 9.3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

接続ファクトリーおよび宛先の作成および構成

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションは、管理対象オブジェクトとして Java Naming and Directory Interface (JNDI) 名前空間から、 IBM JMS 拡張機能を使用して、または IBM MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先を作成できます。 また、アプリケーションは IBM JMS 拡張機能または IBM MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先のプロパティーを設定できます。

接続ファクトリーおよび宛先は、 JMS または Jakarta Messaging アプリケーションのロジックのフローの開始点です。 アプリケーションは ConnectionFactory オブジェクトを使用してメッセージング・サーバー

への接続を作成し、Queue または Topic オブジェクトをターゲットとして使用してメッセージを送信したり、ソースとして使用してそこからメッセージを受信します。 そのため、アプリケーションは少なくとも 1 つの接続ファクトリーおよび 1 つ以上の宛先を作成する必要があります。 接続ファクトリーまたは宛先を作成したら、次にアプリケーションは 1 つ以上のプロパティーを設定してオブジェクトを構成する必要があるかもしれません。

要約すれば、アプリケーションは接続ファクトリーおよび宛先を以下の方法で作成し、構成できます。

JNDI を使用した管理対象オブジェクトの取得
管理者は、管理ツールを使用した JMS および Jakarta Messaging オブジェクトの構成、または IBM MQ Explorer IBM MQ Explorer を使用した JMS 2.0 オブジェクトの構成で説明されているように IBM MQ JMS 管理ツールを使用して、接続ファクトリーおよび宛先を JNDI 名前空間内の管理対象オブジェクトとして作成および構成することができます。 そうすると、アプリケーションは JNDI 名前空間から管理対象オブジェクトを取得できます。 管理対象オブジェクトを取得したら、アプリケーションは、必要な場合にはそのオブジェクトの 1 つ以上のプロパティーを設定または変更できます。それには、IBM JMS 拡張機能または IBM MQ JMS 拡張機能のいずれかを使用します。

注 : JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、 IBM MQ Explorer を使用して JNDI を管理することはできません。 JNDI 管理は、 **JMSAdmin** の Jakarta Messaging 3.0 バリアント (**JMS30Admin**) によってサポートされます。

IBM JMS 拡張機能の使用
アプリケーションは IBM JMS 拡張機能を使用して、接続ファクトリーおよび宛先を実行時に動的に作成できます。 アプリケーションは最初に JmsFactoryFactory オブジェクトを作成し、次にこのオブジェクトのメソッドを使用して、接続ファクトリーおよび宛先を作成します。 接続ファクトリーまたは宛先を作成したら、アプリケーションは JmsPropertyContext インターフェースから継承したメソッドを使用してそのプロパティーを設定できます。 あるいは、アプリケーションは Uniform Resource Identifier (URI) を使用して、宛先の作成時にその宛先の 1 つ以上のプロパティーを指定できます。

IBM MQ JMS 拡張機能の使用
またアプリケーションは IBM MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先を実行時に動的に作成できます。 アプリケーションは提供されたコンストラクターを使用して、接続ファクトリーおよび宛先を作成します。 接続ファクトリーまたは宛先を作成したら、アプリケーションはオブジェクトのメソッドを使用してそのプロパティーを設定できます。 あるいは、アプリケーションは URI を使用して、宛先の作成時にその宛先の 1 つ以上のプロパティーを指定できます。

関連タスク
JMS および Jakarta Messaging リソースの構成

JNDI を使用して *JMS* または *Jakarta Messaging* アプリケーションで管理対象オブジェクトを取得する
Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトを取得するには、 JMS または Jakarta Messaging アプリケーションで初期コンテキストを作成してから、lookup () メソッドを使用してオブジェクトを取得する必要があります。

アプリケーションで管理対象オブジェクトを JNDI ネーム・スペースから取り出す前に、管理者はまず管理対象オブジェクトを作成する必要があります。

JMS 2.0 JMS 2.0 の場合、管理者は IBM MQ JMS 管理ツール、 **JMSAmin**、または IBM MQ Explorer を使用して、JNDI 名前空間で管理対象オブジェクトを作成および保守できます。 詳しくは、JNDI ネーム・スペースでの接続ファクトリーおよび宛先の構成を参照してください。

JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、 IBM MQ Explorer を使用して JNDI を管理することはできません。 JNDI 管理は、 **JMSAdmin** の Jakarta Messaging 3.0 バリアント (**JMS30Admin**) によってサポートされます。

アプリケーション・サーバーには通常、管理対象オブジェクトのための独自のリポジトリーと、オブジェクトの作成と保守のための独自のツールが備えられています。

管理対象オブジェクトを JNDI ネーム・スペースから取り出すには、以下のコードに示されているように、アプリケーションでまず初期コンテキストを作成する必要があります。

JM 3.0 V 9.3.0 V 9.3.0

```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

このコードのストリング変数 `url` および `icf` の意味はそれぞれ次のとおりです。

url
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://`*`hostname`*`/`*`contextName`* (LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/`*`directoryPath`* (ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

icf
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値になります。

- `com.sun.jndi.ldap.LdapCtxFactory`(LDAP サーバーに基づくディレクトリー・サービスの場合)
- `com.sun.jndi.fscontext.RefFSContextFactory`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

JNDI パッケージと Lightweight Directory Access Protocol (LDAP) サービス・プロバイダーの組み合わせによっては、LDAP エラー 84 が発生する場合があります。 この問題を解決するには、InitialDirContext への呼び出しを行う前に、以下のコード行を挿入してください。

```
environment.put(Context.REFERRAL, "throw");
```

初期コンテキストが取得された後、アプリケーションは、以下の例で示されているように lookup() メソッドを使用して JNDI ネーム・スペースから管理対象オブジェクトを取り出すことができます。

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
```

```
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

このコードは LDAP に基づくネーム・スペースから以下のオブジェクトを取り出します。

- myCF という名前でバインドされる ConnectionFactory オブジェクト
- myQ という名前でバインドされる Queue オブジェクト
- myT という名前でバインドされる Topic オブジェクト

JNDI の使い方について詳しくは、Oracle Corporation が提供する JNDI の資料を参照してください。

関連タスク

IBM MQ エクスプローラーを使用した JMS 2.0 オブジェクトの構成

管理ツールを使用した JMS および Jakarta Messaging オブジェクトの構成

WebSphere Application Server での JMS 2.0 リソースの構成

IBM JMS 拡張機能の使用

IBM MQ classes for JMS (JMS 2.0) および IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) にはそれぞれ、 IBM JMS 拡張と呼ばれる JMS API に対する機能的に同一の拡張セットが含まれています。 アプリケーションは、これらの拡張機能を使用して、接続ファクトリーおよび宛先を実行時に動的に作成したり、 IBM MQ classes for JMS オブジェクトまたは IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定したりすることができます。 この拡張機能は、任意のメッセージング・プロバイダーと共に使用できます。

IBM JMS 拡張機能は、以下のパッケージ内の一連のインターフェースおよびクラスです。

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Jakarta Messaging 3.0 の場合、これらのパッケージは `com.ibm.jakarta.client.jar` にあります。

JMS 2.0 JMS 2.0 の場合、これらのパッケージは `com.ibm.mqjms.jar` または `com.ibm.mq.allclient.jar` にあります。

これらの拡張機能は、以下の機能を提供します。

- Java Naming and Directory Interface (JNDI) ネーム・スペースから管理対象オブジェクトとして取り出す代わりに、接続ファクトリーおよび宛先を実行時に動的に作成するための、ファクトリー・ベース・メカニズム
- IBM MQ classes for JMS オブジェクトまたは IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定するための一連のメソッド
- 問題に関する詳細情報を入手するためのメソッドがある一連の例外クラス
- トレースを制御するための一連のメソッド
- IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging に関するバージョン情報を取得するための一連のメソッド

実行時に接続ファクトリーと宛先を動的に作成し、それらのプロパティーを設定および取得するために、IBM JMS 拡張は、 IBM MQ JMS 拡張に対するインターフェースの代替セットを提供します。 ただし、IBM MQ JMS 拡張機能は IBM MQ メッセージング・プロバイダーに固有のものですが、IBM JMS 拡張機能は IBM MQ に固有のものではなく、「JMS アーキテクチャーの IBM MQ クラス」で書かれている階層的アーキテクチャー内のどのメッセージング・プロバイダーでも使用できます。

インターフェース com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) または com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) には、 IBM JMS 拡張を使用して IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定する際にアプリケーションが使用できる定数の定義が含まれています。 このインターフェースには、IBM MQ メッセージング・プロバイダーの定数と、どのメッセージング・プロバイダーにも依存していない JMS 定数が含まれています。

以下のコード例では、以下のインポート・ステートメントが Java クラスに含まれていることを前提としています。

JM 3.0 V 9.3.0 V 9.3.0

```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0

```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

接続ファクトリーと宛先の作成

アプリケーションが IBM JMS 拡張機能を使用して接続ファクトリーおよび宛先を作成する前に、まず JmsFactoryFactory オブジェクトを作成する必要があります。 以下の例が示すように、JmsFactoryFactory オブジェクトを作成するために、アプリケーションは JmsFactoryFactory クラスの getInstance() メソッドを呼び出します。

JM 3.0 V 9.3.0 V 9.3.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

getInstance() 呼び出しのパラメーターは、IBM MQ メッセージング・プロバイダーを、選択したメッセージング・プロバイダーとして識別する定数です。 次いでアプリケーションは JmsFactoryFactory オブジェクトを使用して、接続ファクトリーと宛先を作成できます。

以下の例が示すように、接続ファクトリーを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createConnectionFactory() メソッドを呼び出します。

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsConnectionFactory オブジェクトを作成します。これはアプリケーションがバインディング・モードでデフォルトのキュー・マネージャーに接続することを意味します。 アプリケーションをクライアント・モードで接続したいか、またはデフォルトのキュー・マネージャー以外のキュー・マネージャーに接続したい場合、接続を作成する前に、アプリケーションは JmsConnectionFactory オブジェクトの適切なプロパティーを設定する必要があります。 これを行う方法については、212 ページの『IBM MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

JmsFactoryFactory クラスには、以下のタイプの接続ファクトリーを作成するためのメソッドも含まれています。

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

以下の例が示すように、Queue オブジェクトを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createQueue() メソッドを呼び出します。

```
JmsQueue q1 = ff.createQueue("Q1");
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsQueue オブジェクトを作成します。 オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる IBM MQ キューを表します。 このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

createQueue() メソッドは、キュー Uniform Resource Identifier (URI) をパラメーターとして受け入れることもできます。 キュー URI は、IBM MQ キューの名前と、オプションで、キューを所有するキュー・マネージャーの名前、および JmsQueue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

このステートメントによって作成される JmsQueue オブジェクトは、キュー・マネージャー QM2 が所有する Q2 と呼ばれる IBM MQ キューを表し、この宛先に送信されるすべてのメッセージは永続的なもので、優先順位 5 を持ちます。 キューの URI について詳しくは、225 ページの『Uniform Resource Identifier (URI)』を参照してください。 JmsQueue オブジェクトのプロパティーの別の設定方法については、212 ページの『IBM MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

以下の例が示すように、Topic オブジェクトを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createTopic() メソッドを使用できます。

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsTopic オブジェクトを作成します。 オブジェクトは Sport/Football/Results という名前のトピックを表します。

さらに、createTopic() メソッドは、トピック URI をパラメーターとして受け入れることもできます。 トピック URI は、トピックの名前と、オプションで JmsTopic オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントには、トピック URI の例が含まれています。

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

これらのステートメントによって作成される JmsTopic オブジェクトは、Sport/Tennis/Results というトピックを表し、この宛先に送信されるすべてのメッセージは非永続的なもので、優先順位 0 を持ちます。 トピックの URI について詳しくは、225 ページの『Uniform Resource Identifier (URI)』を参照してください。 JmsTopic オブジェクトのプロパティーの別の設定方法については、212 ページの『IBM MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

アプリケーションが接続ファクトリーまたは宛先を作成した後は、そのオブジェクトは選択されたメッセージング・プロバイダーだけでしか使用できません。

IBM MQ classes for JMS オブジェクトのプロパティーの設定

IBM MQ classes for JMS オブジェクトのプロパティーを IBM JMS 拡張機能を使用して設定するには、アプリケーションは com.ibm.msg.client.JmsPropertyContext インターフェースのメソッドを使用します。 同様に、 IBM JMS 拡張を使用して IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定するために、アプリケーションは com.ibm.msg.jakarta.client.JmsPropertyContext インターフェースのメソッドを使用します。

各 Java データ型に対して、JmsPropertyContext インターフェースには、そのデータ型でプロパティーの値を設定するメソッドと、そのデータ型でプロパティーの値を取得するメソッドが含まれています。 例えば、アプリケーションは setIntProperty() メソッドを呼び出して、整数値でプロパティーを設定し、getIntProperty() メソッドを呼び出して、整数値でプロパティーを取得します。

com.ibm.mq.jms および com.ibm.mq.jakarta.jms パッケージ内のクラスのインスタンスは、対応する JmsProperty コンテキスト・インターフェースのメソッドを継承します。 したがってアプリケーションは、これらのメソッドを使用して、MQConnectionFactory、MQQueue、および MQTopic の各オブジェクトのプロパティーを設定できます。

アプリケーションが IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトを作成すると、デフォルト値を持つすべてのプロパティーが自動的に設定されます。 アプリケーションがプロパティーを設定するときは、新規の値がプロパティーの以前の値を置き換えます。 プロパティーが設定された後には、それを削除することはできませんが、その値を変更することはできます。

アプリケーションがプロパティーをそのプロパティーの有効な値ではない値に設定しようとすると、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging は JMSException 例外をスローします。 アプリケーションが、設定されていないプロパティーを取得しようとした場合、その動作は JMS 仕様に記述されているとおりになります。 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、プリミティブ・データ型の場合は NumberFormat 例外をスローし、参照されるデータ型の場合は NULL を返します。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトの事前定義プロパティーに加えて、アプリケーションは独自のプロパティーを設定できます。 これらのアプリケーション定義プロパティーは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging によって無視されます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーについて詳しくは、 IBM MQ classes for JMS オブジェクトのプロパティーを参照してください。

以下のコードは、IBM JMS 拡張機能を使用してプロパティーを設定する方法の例です。 このコードは、接続ファクトリーの 5 つのプロパティーを設定します。

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

それらのプロパティーを設定すると、アプリケーションは QM1.SVR という名前の MQI チャネルを使用して、クライアント・モードでキュー・マネージャー QM1 に接続します。 キュー・マネージャーは HOST1 というホスト名を持つシステムで稼働し、キュー・マネージャーのリスナーはポート番号 1415 で listen します。 この接続およびその下にあるセッションに関連付けられた他のキュー・マネージャー接続には、アプリケーション名「My Application」が関連付けられます。

注 : z/OS プラットフォーム上で実行するキュー・マネージャーでは、アプリケーション名の設定をサポートしていないため、この設定は無視されます。

JmsPropertyContext インターフェースには setObjectProperty() メソッドも含まれており、アプリケーションはそれをプロパティーの設定に使用できます。 メソッドの 2 番目のパラメーターは、プロパティーの値をカプセル化するオブジェクトです。 例えば、以下のコードは整数 1415 をカプセル化する整数オブジェクトを作成し、次いで setObjectProperty() を呼び出して、接続ファクトリーの PORT プロパティーを値 1415 に設定します。

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

したがってこのコードは、以下のステートメントと同等です。

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

それとは反対に、getObjectProperty() メソッドは、プロパティーの値をカプセル化するオブジェクトを戻します。

プロパティー値のデータ型の暗黙的な変換

アプリケーションが JmsPropertyContext インターフェースのメソッドを使用して、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定または取得する場合、プロパティーの値は、あるデータ・タイプから別のデータ・タイプに暗黙的に変換できます。

例えば、以下のステートメントは、JmsQueue オブジェクト q1 の PRIORITY プロパティーを設定します。

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

PRIORITY プロパティーには整数値が指定されているため、setStringProperty() 呼び出しは、ストリング "5" (ソース値) を整数 5 (ターゲット値) に暗黙的に変換し、それが PRIORITY プロパティーの値になります。

これとは逆に、以下のステートメントは、JmsQueue オブジェクト q1 の PRIORITY プロパティーを取得します。

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

整数 5 (ソース値) は、PRIORITY プロパティーの値です。これは、getStringProperty() 呼び出しによって暗黙的にストリング "5" (ターゲット値) に変換されます。

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging でサポートされる変換を 214 ページの表 34 に示します。

表 *34.* あるデータ型から別のデータ型への、サポートされる変換

ソースのデータ型	**サポートされているターゲット・データ・タイプ**
boolean	ストリング
byte	int、long、short、String
char	ストリング
double	ストリング
float	double、String
int	long、String
long	ストリング
short	int、long、String
ストリング	boolean、byte、double、float、int、long、short

サポートされる変換を制御する一般規則は、以下のとおりです。

- 変換中に失われるデータがない場合には、数値をあるデータ型から別のデータ型に変換できます。 例えば、データ型 `int` の値は、データ型 `long` の値に変換できますが、データ型 `short` の値には変換できません。
- 任意のデータ型の値を、ストリングに変換できます。
- 変換に適正な形式であれば、ストリングは任意の他のデータ型 (`char` を除く) の値に変換できます。 アプリケーションが正しい形式でないストリングを変換しようとすると、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は NumberFormat 例外をスローします。
- サポートされていない変換をアプリケーションが試行すると、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は MessageFormat 例外をスローします。

あるデータ型から別のデータ型に値を変換するための特定の規則は、以下のとおりです。

- ブール値をストリングに変換する場合、値 `true` はストリング "true" に変換され、値 `false` はストリング "false" に変換されます。
- ストリングをブール値に変換する場合、ストリング "true" (大/小文字の区別をしない) は `true` に変換され、ストリング "false" (大/小文字の区別をしない) は `false` に変換されます。 それ以外のストリングは `false` に変換されます。
- ストリングをデータ型 `byte`、`int`、`long`、または `short` の値に変換する場合、ストリングは以下の形式でなければなりません。

[*blanks*][*sign*]*digits*

ストリングの構成要素の意味は以下のとおりです。

blanks
オプションの先行ブランク文字。

sign
オプションの正符号 (+) または負符号 (-)。

digits
数字 (0 から 9) の連続シーケンス。 少なくとも 1 つの数字がなければなりません。

数字のシーケンスの後に、ストリングには数字ではない他の文字を含めることができますが、これらの文字の先頭に達するとすぐに変換は停止します。 ストリングは 10 進整数を表すと想定されます。

ストリングの形式が正しくない場合、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は NumberFormat 例外をスローします。

- ストリングをデータ型 `double` または `float` の値に変換する場合、ストリングは以下の形式でなければなりません。

  [*blanks*][*sign*]*digits*[*e_char*[*e_sign*]*e_digits*]

  ストリングの構成要素の意味は以下のとおりです。

  blanks
  オプションの先行ブランク文字。

  sign
  オプションの正符号 (+) または負符号 (-)。

  digits
  数字 (0 から 9) の連続シーケンス。 少なくとも 1 つの数字がなければなりません。

  e_char
  指数文字。*E* または *e* のいずれかです。

  e_sign
  指数用のオプションの正符号 (+) または負符号 (-)。

  e_digits
  指数用の数字 (0 から 9) の連続シーケンス。 ストリングに指数文字が含まれている場合は、少なくとも 1 つの数字がなければなりません。

  数字のシーケンス、または指数を表すオプション文字の後に、ストリングには数字ではない他の文字を含めることができますが、これらの文字の先頭に達するとすぐに変換は停止します。 ストリングは、10 の累乗の指数を持つ 10 進浮動小数点数を表すと想定されます。

  ストリングの形式が正しくない場合、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は NumberFormat 例外をスローします。

- 数値 (データ型が `byte` の値を含む) をストリングに変換する場合、値は 10 進数としての値のストリング表現に変換され、その値の ASCII 文字を含むストリングに変換されるのではありません。 例えば、整数 65 はストリング "65" に変換され、ストリング "A" に変換されるのではありません。

単一呼び出しでの複数のプロパティーの設定

JmsPropertyContext インターフェースには setBatchProperties() メソッドも含まれており、アプリケーションはそれを単一呼び出しでの複数のプロパティーの設定に使用できます。 メソッドのパラメーターは、一連のプロパティーの名前と値のペアをカプセル化する Map オブジェクトです。

例えば、以下のコードは setBatchProperties() メソッドを使用して、212 ページの『IBM MQ classes for JMS オブジェクトのプロパティーの設定』に示す接続ファクトリーの同じ 5 つのプロパティーを設定します。 このコードは、Map インターフェースを実装する HashMap クラスのインスタンスを作成します。

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
```

```
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Map.put() メソッドの 2 番目のパラメーターはオブジェクトでなければならないことに注意してください。 したがって、例で示すとおり、プリミティブ・データ型のプロパティー値は、オブジェクト内でカプセル化するか、またはストリングによって表現されなければなりません。

setBatchProperties() メソッドは、各プロパティーを妥当性検査します。 setBatchProperties() メソッドが、例えばその値が無効であるなどの理由でプロパティーを設定できない場合、指定されたどのプロパティーも設定されません。

プロパティーの名前と値

アプリケーションが適切な JmsPropertyContext インターフェースのメソッドを使用して、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーを設定および取得する場合、アプリケーションは、以下のいずれかの方法でプロパティーの名前と値を指定できます。 記載されている各例は、JmsQueue オブジェクト q1 の PRIORITY プロパティーを設定し、キューに送信されるメッセージが、send() 呼び出しに指定された優先順位を持つようにする方法を示しています。

com.ibm.msg.client.wmq.WMQConstants インターフェースに定数として定義されたプロパティーの名前と値の使用

以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

キューおよびトピック Uniform Resource Identifier (URI) で使用できるプロパティーの名前と値の使用

以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setIntProperty("priority", -2);
```

宛先のプロパティーの名前と値だけがこの方法で指定できます。

IBM MQ JMS 管理ツールにより認識されるプロパティーの名前と値の使用

以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setStringProperty("PRIORITY", "APP");
```

以下のステートメントで示すとおり、簡易形式のプロパティー名も受け入れることができます。

```
q1.setStringProperty("PRI", "APP");
```

アプリケーションがプロパティーを取得するときに戻される値は、アプリケーションがプロパティーの名前を指定する方法に応じて異なります。 例えば、アプリケーションが定数 WMQConstants.WMQ_PRIORITY をプロパティー名として指定する場合、戻される値は整数 -2 です。

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

アプリケーションがストリング "priority" をプロパティー名として指定すると、同じ値が戻されます。

```
int n2 = getIntProperty("priority");
```

ただし、アプリケーションがストリング "PRIORITY" または "PRI" をプロパティー名として指定する場合、戻される値はストリング "APP" です。

```
String s1 = getStringProperty("PRI");
```

内部では、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、一致する WMQConstants インターフェースで定義されたリテラル値としてプロパティー名と値を保管します。 これはプロパティーの名前と値の定義された正規形式です。 一般に、アプリケーションがプロパティーの名前と値を指定する他の 2 つの方法のいずれかを使用してプロパティーを設定する場合、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、指定された入力形式から標準形式に名前と値を変換する必要があります。 同様に、アプリケーションがプロパティーの名前と値を指定する他の 2 つの方法のいずれかを使用してプロパティーを取得する場合、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、指定された入力形式の名前を正規形式に変換し、正規形式の値を必要な出力形式に変換する必要があります。 これらの変換を実行する必要があることには、パフォーマンス上の含意があります。

トレース・ファイル、または IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging ログで例外によって返されるプロパティーの名前と値は、常に正規形式です。

Map インターフェースの使用

JmsPropertyContext インターフェースは java.util.Map インターフェースを拡張します。 したがって、アプリケーションは、Map インターフェースのメソッドを使用して、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging オブジェクトのプロパティーにアクセスできます。

例えば、以下のコードは、接続ファクトリーのすべてのプロパティーの名前と値を印刷します。 このコードは、Map インターフェースのメソッドだけを使用して、プロパティーの名前と値を取得します。

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Map インターフェースのメソッドを使用しても、プロパティーの妥当性検査または変換はバイパスされません。

IBM MQ JMS 拡張機能の使用
IBM MQ classes for JMS には、 IBM MQ JMS 拡張機能と呼ばれる、 JMS API に対する拡張機能のセットが含まれています。 アプリケーションはこれらの拡張機能を使用して接続ファクトリーや宛先を実行時に動的に作成したり、接続ファクトリーや宛先のプロパティーを設定したりすることができます。

IBM MQ classes for JMS のパッケージ com.ibm.jms および com.ibm.mq.jms にはクラスのセットが入っています。 これらのクラスは JMS インターフェースを実装し、IBM MQ JMS 拡張機能を含みます。 以下のコード例は、次のステートメントによってそれらのパッケージがインポート済みであることを前提としています。

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

アプリケーションは IBM MQ JMS 拡張機能を使用して、以下の機能を実行することができます。

- 接続ファクトリーや宛先を管理対象オブジェクトとして Java Naming and Directory Interface (JNDI) ネーム・スペースから取り出す代わりに、それらを実行時に動的に作成する
- 接続ファクトリーおよび宛先のプロパティーを設定する

接続ファクトリーの作成

接続ファクトリーを作成するために、アプリケーションは以下の例で示されているように MQConnectionFactory コンストラクターを使用できます。

```
MQConnectionFactory factory = new MQConnectionFactory();
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQConnectionFactory オブジェクトを作成します。これは、アプリケーションがバインディング・モードでデフォルトのキュー・マネージャーに接続することを意味します。 アプリケーションをクライアント・モードで接続する場合、またはデフォルト以外のキュー・マネージャーに接続する場合、アプリケーションは接続を作成する前に MQConnectionFactory オブジェクトの適切なプロパティーを設定する必要があります。 これを行う方法については、218 ページの『接続ファクトリーのプロパティーの設定』を参照してください。

アプリケーションは、同様の方法で以下のタイプの接続ファクトリーを作成することができます。

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

接続ファクトリーのプロパティーの設定

アプリケーションは、接続ファクトリーの適切なメソッドを呼び出すことにより、接続ファクトリーのプロパティーを設定することができます。 接続ファクトリーは、管理対象オブジェクトまたは実行時に動的に作成されたオブジェクトのいずれかです。

例えば、次のようなコードがあるとします。

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

このコードは MQConnectionFactory オブジェクトを作成した後、オブジェクトの 5 つのプロパティーを設定します。 それらのプロパティーを設定すると、アプリケーションは QM1.SVR という名前の MQI チャネルを使用してクライアント・モードでキュー・マネージャー QM1 に接続します。 キュー・マネージャーは HOST1 というホスト名を持つシステムで稼働し、キュー・マネージャーのリスナーはポート番号 1415 で listen します。

ブローカーとのリアルタイム接続を使用するアプリケーションは、パブリッシュ/サブスクライブ・スタイルのメッセージングのみ使用できます。 Point-to-Point スタイルのメッセージングは使用できません。

特定の接続ファクトリーのプロパティーの組み合わせのみ有効です。 有効な組み合わせについて詳しくは、IBM MQ classes for JMS オブジェクトのプロパティー間の依存関係を参照してください。

接続ファクトリーのプロパティー、およびそのプロパティーの設定に使用するメソッドについて詳しくは、IBM MQ classes for JMS オブジェクトのプロパティーを参照してください。

宛先の作成

Queue オブジェクトを作成するために、アプリケーションは以下の例で示されているように MQQueue コンストラクターを使用できます。

```
MQQueue q1 = new MQQueue("Q1");
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQQueue オブジェクトを作成します。オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる IBM MQ キューを表します。このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

MQQueue コンストラクターの代替形式は、以下の例に示されているように、2 つのパラメーターを持ちます。

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

このステートメントによって作成される MQQueue オブジェクトは、キュー・マネージャー QM2 が所有する Q2 という名前の IBM MQ キューを表します。 このようにして識別されるキュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合は、アプリケーションがこの宛先に送信したメッセージを、WebSphere MQ がローカル・キュー・マネージャーからリモート・キュー・マネージャーに転送できるように、IBM MQ を構成する必要があります。

MQQueue コンストラクターは単一パラメーターとしてキュー Uniform Resource Identifier (URI) を受け入れることもできます。 キュー URI は、IBM MQ キューの名前と、オプションで、キューを所有するキュー・マネージャーの名前、および MQQueue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

このステートメントによって作成される MQQueue オブジェクトは、キュー・マネージャー QM3 が所有する Q3 と呼ばれる IBM MQ キューを表し、この宛先に送信されるすべてのメッセージは持続し、優先順位 5 を持ちます。 キューの URI について詳しくは、225 ページの『Uniform Resource Identifier (URI)』を参照してください。 MQQueue オブジェクトのプロパティーを設定する別の方法については、219 ページの『宛先のプロパティーの設定』を参照してください。

アプリケーションは、以下の例で示されているように MQTopic コンストラクターを使用して Topic オブジェクトを作成することができます。

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQTopic オブジェクトを作成します。 オブジェクトは Sport/Football/Results という名前のトピックを表します。

MQTopic コンストラクターはパラメーターとしてトピック URI を受け入れることもできます。 トピック URI は、トピックの名前と、オプションで MQTopic オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはトピック URI の例が含まれています。

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

このステートメントによって作成される MQTopic オブジェクトは、Sport/Tennis/Results というトピックを表し、この宛先に送信されるすべてのメッセージは非永続で、優先順位 0 を持ちます。 トピックの URI について詳しくは、225 ページの『Uniform Resource Identifier (URI)』を参照してください。 MQTopic オブジェクトのプロパティーを設定する別の方法については、219 ページの『宛先のプロパティーの設定』を参照してください。

宛先のプロパティーの設定

アプリケーションは、宛先の適切なメソッドを呼び出すことにより、宛先のプロパティーを設定することができます。 宛先は、管理対象オブジェクトまたは実行時に動的に作成されたオブジェクトのいずれかです。

例えば、次のようなコードがあるとします。

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

このコードは MQQueue オブジェクトを作成した後、オブジェクトの 2 つのプロパティーを設定します。 それらのプロパティーを設定すると、その宛先に送信されるメッセージはすべて永続メッセージとなり、優先順位は 5 になります。

アプリケーションは、以下の例で示されているとおり、同様の方法で MQTopic オブジェクトのプロパティーを設定することができます。

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

このコードは MQTopic オブジェクトを作成した後、オブジェクトの 2 つのプロパティーを設定します。 それらのプロパティーを設定すると、その宛先に送信されるメッセージはすべて非永続メッセージとなり、優先順位は 0 になります。

宛先のプロパティー、およびそのプロパティーの設定に使用するメソッドについて詳しくは、IBM MQ classes for JMS オブジェクトのプロパティーを参照してください。

Linux AIX *JMS アプリケーションから IBM MQ への接続*

接続を構築するために、 JMS アプリケーションは **ConnectionFactory** オブジェクトを使用して **Connection** オブジェクトを作成してから、接続を開始します。

JMS 2.0 以降の場合、アプリケーションは通常、 **ConnectionFactory** オブジェクトおよび `createContext()` メソッドを使用してメッセージング・プロバイダーに接続します。

以前のバージョンの JMS では、最初に `createConnection` を使用して **Connection** オブジェクトを作成してから、接続呼び出し `getSession()` を開始して、メッセージング操作を実行できる **Session** オブジェクトを作成する必要がありました。

JMSContext オブジェクトは、 **Connection** オブジェクトと **Session** オブジェクトの両方を効果的にカプセル化します。 従来のアプローチを使用し、接続オブジェクトとセッション・オブジェクトを直接作成する場合は、 220 ページの『JMS アプリケーションでの接続の構築』 および 221 ページの『JMS アプリケーションでのセッションの作成』を参照してください。

JMSContext オブジェクトを作成するために、アプリケーションは、以下の例に示すように、**ConnectionFactory** オブジェクトの `createContext()` メソッドを使用します。

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createContext();
```

JMS 接続が作成されると、 IBM MQ classes for JMS は接続ハンドル (Hconn) を作成し、キュー・マネージャーとの会話を開始します。

注 : アプリケーション・プロセス ID は、キュー・マネージャーに渡されるデフォルト・ユーザー ID として使用されることに注意してください。 アプリケーションがクライアント・トランスポート・モードで実行される場合は、このプロセス ID がサーバー上に存在し、関係する許可がプロセス ID に与えられている必要があります。 別の ID を使用する場合は、 `createConnection(username,password)` メソッドを使用します。

V 9.3.5 このメカニズムを使用して認証トークンを提供することもできます。 選択したトークン発行者からの認証トークンの取得を参照してください。

JMS 1.0 *JMS アプリケーションでの接続の構築*

JMS 1.0 で接続を作成するために、 JMS アプリケーションは ConnectionFactory オブジェクトを使用して Connection オブジェクトを作成してから、接続を開始します。

アプリケーションは、以下の例で示されているように ConnectionFactory オブジェクトの createConnection() メソッドを使用して Connection オブジェクトを作成します。

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

JMS 接続が作成されると、IBM MQ classes for JMS は接続ハンドル (Hconn) を作成して、キュー・マネージャーとの会話を開始します。

QueueConnectionFactory インターフェースと TopicConnectionFactory インターフェースはそれぞれ、ConnectionFactory インターフェースから createConnection() メソッドを継承しています。 このため、以下の例で示されているように createConnection() メソッドを使用してドメイン特定のオブジェクトを作成することができます。

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

このコード断片は、QueueConnection オブジェクトを作成します。 アプリケーションは、このオブジェクト上でドメイン独立の操作、または Point-to-Point ドメインにのみ適用される操作を実行できます。 ただし、アプリケーションがパブリッシュ/サブスクライブ・ドメインに対してのみ適用される操作を実行しようとすると、IllegalStateException 例外が以下のメッセージと一緒にスローされます。

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

その理由は、接続がドメイン特定の接続ファクトリーにより作成されているためです。

注 : アプリケーション・プロセス ID は、キュー・マネージャーに渡されるデフォルト・ユーザー ID として使用されることに注意してください。 アプリケーションがクライアント・トランスポート・モードで実行される場合は、このプロセス ID がサーバー上に存在し、関係する許可がプロセス ID に与えられている必要があります。 別の ID を使用する必要がある場合は、createConnection(username, password) メソッドを使用してください。

JMS 仕様は、接続が stopped 状態で作成されると記述しています。 接続が開始するまで、接続に関連付けられたメッセージ・コンシューマーはメッセージを受信できません。 アプリケーションは、以下の例で示されているように Connection オブジェクトの start() メソッドを使用して接続を開始します。

```
connection.start();
```

V 9.3.5 このメカニズムを使用して認証トークンを提供することもできます。 選択したトークン発行者からの認証トークンの取得を参照してください。

JMS 1.0 *JMS アプリケーションでのセッションの作成*

JMS 1.0 でセッションを作成するために、 JMS アプリケーションは Connection オブジェクトの createSession() メソッドを使用します。

createSession() メソッドは 2 つのパラメーターを持ちます。

1. セッションがトランザクション化されているかどうかを指定するパラメーター
2. セッションの確認応答モードを指定するパラメーター

例えば、以下のコードは、トランザクション化されていないセッションを作成し、確認応答モードは AUTO_ACKNOWLEDGE です。

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

JMS セッションが作成されると、IBM MQ classes for JMS は接続ハンドル (Hconn) を作成して、キュー・マネージャーとの会話を開始します。

Session オブジェクト、およびそこから作成された MessageProducer または MessageConsumer オブジェクトはすべて、マルチスレッド・アプリケーションの別のスレッドで並行して使用することができません。それらのオブジェクトが並行して使用されないようにするための最も簡単な方法は、各スレッドごとに別個の Session オブジェクトを作成することです。

V 9.3.5 このメカニズムを使用して認証トークンを提供することもできます。 選択したトークン発行者からの認証トークンの取得を参照してください。

JMS アプリケーションでのトランザクション化されたセッション
JMS アプリケーションは、最初にトランザクション化されたセッションを作成することによって、ローカル・トランザクションを実行できます。 アプリケーションでは、トランザクションのコミットとロールバックが可能です。

JMS アプリケーションは、ローカル・トランザクションを実行できます。 ローカル・トランザクションとは、アプリケーションが接続されているキュー・マネージャーのリソースにのみに対して行われた変更が含まれるトランザクションのことです。 ローカル・トランザクションを実行するには、まずアプリケーションは、Connection オブジェクトの createSession() メソッドを呼び出すことでトランザクション化セッションを作成し、セッションがトランザクション化されるパラメーターとして指定する必要があります。 その後、そのセッション内で送受信されたすべてのメッセージは、トランザクションの順序でグループ化されます。 トランザクションは、トランザクションが開始されてから送受信したメッセージがアプリケーションでコミットまたはロールバックされると終了します。

トランザクションをコミットするには、アプリケーションで Session オブジェクトの commit() メソッドを呼び出します。 トランザクションがコミットされると、そのトランザクション内に送信されたすべてのメッセージは、他のアプリケーションに配信できるようになります。また、そのトランザクション内に受信したすべてのメッセージが認知されるので、メッセージング・サーバーはそれらのメッセージをアプリケーションへ再配信しなくなります。 また、Point-to-Point ドメインでは、受信したメッセージがメッセージング・サーバーのキューからも除去されます。

トランザクションをロールバックするには、アプリケーションで Session オブジェクトの rollback() メソッドを呼び出します。 トランザクションがロールバックされると、そのトランザクション内に送信されたすべてのメッセージはメッセージング・サーバーによって破棄されます。また、そのトランザクション内に受信したすべてのメッセージは再配信できるようになります。 Point-to-Point ドメインでは、受信されたメッセージはキューに書き戻され、再び他のアプリケーションから見えるようになります。

アプリケーションがトランザクション化されたセッションを作成するか、commit() または rollback() メソッドを呼び出すと、自動的に新しいトランザクションが開始されます。 したがって、トランザクション化されたセッションには常にアクティブなトランザクションが含まれます。

アプリケーションがトランザクション化されたセッションを閉じると、暗黙的なロールバックが行われます。 アプリケーションが接続を閉じると、その接続のトランザクション化されたセッションすべてで暗黙的なロールバックが行われます。

アプリケーションが接続を閉じずに終了した場合も、その接続のトランザクション化されたセッションすべてで暗黙のロールバックが行われます。

トランザクションは、トランザクション化されたセッションに完全に包含されています。 トランザクションがセッションをまたぐことはできません。 つまり、アプリケーションは、トランザクション化された複数のセッションの中でメッセージを送受信したり、これらのすべてのアクションを単一のトランザクションとしてコミットまたはロールバックしたりすることはできません。

JMS セッションの確認応答モード
トランザクション化されないすべてのセッションには、アプリケーションによって受信されたメッセージをどのように確認するかを決定する、確認応答モードが存在します。 使用可能な確認応答モードは 3 つあり、どの確認応答モードを選択するかはアプリケーションの設計に影響を与えます。

セッションがトランザクション化されない場合、アプリケーションが受信するメッセージを確認する方法は、セッションの確認応答モードによって決定されます。 以下の部分では、3 つの確認応答モードについて説明します。

AUTO_ACKNOWLEDGE

セッションは、アプリケーションが受信した各メッセージを自動的に確認します。

メッセージがアプリケーションに同期化して配信されると、セッションは、Receive 呼び出しが正常に完了するたびにメッセージの受信を確認します。 メッセージが非同期的に送達された場合は、セッションは、メッセージ・リスナーの onMessage() メソッドの呼び出しが正常に完了するたびにメッセージの受信を確認します。

アプリケーションがメッセージを正常に受信しても、障害によって確認が行えない場合は、そのメッセージは再び送達可能になります。 このため、アプリケーションは、再送達されたメッセージを扱うことができなければなりません。

DUPS_OK_ACKNOWLEDGE

セッションは、メッセージ選択時にアプリケーションが受信したメッセージを確認します。

この確認応答モードを使用すると、セッションで行わなければならない作業の量を減らすことができますが、障害によってメッセージの確認ができなかったときは、複数のメッセージが再び送達可能になる可能性があります。 このため、アプリケーションは、再送達されたメッセージを扱うことができなければなりません。

制約事項 : AUTO_ACKNOWLEDGE および DUPS_OK_ACKNOWLEDGE モードでは、JMS は、メッセージ・リスナーで処理できない例外のアプリケーションによるスローをサポートしません。 これは、メッセージ・リスナーの処理が正常に行われたかどうかに関係なく (ただし、障害がいずれも致命的なものではなく、アプリケーションの続行を妨げるものでない限り)、メッセージ・リスナーから処理が戻ると、必ずメッセージが確認されることを意味します。 メッセージの確認をより細かく制御する必要がある場合は、CLIENT_ACKNOWLEDGE またはトランザクション化モードを使用します。これらのモードを使用すれば、確認の機能をアプリケーションから完全に制御できます。

CLIENT_ACKNOWLEDGE

Message クラスの Acknowledge メソッドを呼び出すことにより、受信したメッセージをアプリケーションが確認します。

アプリケーションは各メッセージの受信を個々に確認するか、または複数のメッセージを一括して受信し、受信した最後のメッセージに対してのみ Acknowledge メソッドを呼び出すことができます。 Acknowledge メソッドが呼び出されると、このメソッドの前回の呼び出し以降に受信したすべてのメッセージが確認されます。

これらの確認応答モードのいずれかと組み合わせることにより、アプリケーションは Session クラスの Recover メソッドを呼び出してセッションでメッセージの送達を停止したり、再開させたりすることができます。 受信されたが前回は未確認だったメッセージについては、再送達されます。 ただし、前回送達されたときと同じシーケンスで送達されるとは限りません。 これらのメッセージが再送達されるまでの間に、より優先順位の高いメッセージが届いている可能性もありますし、オリジナルのメッセージの一部が有効期限切れになっている場合もあります。 Point-to-Point ドメインの場合は、オリジナルのメッセージの一部が別のアプリケーションによって消費されている可能性もあります。

アプリケーションでは、メッセージの JMSRedelivered ヘッダー・フィールドの内容を調べることによって、メッセージが再送達中かどうかを確認できます。 アプリケーションでこれを行うには、Message クラスの getJMSRedelivered() メソッドを呼び出します。

JMS アプリケーションでの宛先の作成

Java Naming and Directory Interface (JNDI) ネーム・スペースから宛先を管理対象オブジェクトとして取り出す代わりに、JMS アプリケーションは実行時に動的に宛先を作成するセッションを使用できます。 アプ

リケーションは URI (Uniform Resource Identifier) を使用して IBM MQ キューまたはトピックを識別し、オプションで、Queue または Topic オブジェクトの 1 つ以上のプロパティーを指定することができます。

セッションを使用した Queue オブジェクトの作成

以下の例が示すように、Queue オブジェクトを作成するために、アプリケーションは Session オブジェクトの createQueue() メソッドを使用できます。

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

このコードは、すべてのプロパティーにデフォルト値が指定された Queue オブジェクトを作成します。 オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる IBM MQ キューを表します。 このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

createQueue() メソッドは、キュー URI もパラメーターとして受け入れます。 キュー URI は、IBM MQ キューの名前を指定し、オプションで、キューを所有するキュー・マネージャーの名前、および Queue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

このステートメントによって作成されるキュー・オブジェクトは、QM2 というキュー・マネージャーが所有する Q2 キューと呼ばれる IBM MQ キューを表し、この宛先に送信されるすべてのメッセージは持続し、優先順位は 5 になります。 このようにして識別されるキュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合、IBM MQ は、アプリケーションがメッセージをこの宛先に送信するときに WebSphere MQ がメッセージをローカル・キュー・マネージャーからキュー・マネージャー QM2 に経路指定できるように構成する必要があります。 URI の詳細については、225 ページの『Uniform Resource Identifier (URI)』を参照してください。

createQueue() メソッドのパラメーターには、プロバイダー固有の情報が含まれていることに注意してください。 したがって、Queue オブジェクトを管理対象オブジェクトとして JNDI ネーム・スペースから取り出す代わりに、createQueue() メソッドを使用して Queue オブジェクトを作成すると、アプリケーションの移植性は低くなる場合があります。

以下の例が示すように、アプリケーションは Session オブジェクトの createTemporaryQueue() メソッドを使用して TemporaryQueue オブジェクトを作成できます。

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

セッションは一時キューを作成するために使用されますが、一時キューの有効範囲は、セッションを作成するために使用された接続です。 接続のどのセッションでも、一時キューのためのメッセージ・プロデューサーおよびメッセージ・コンシューマーを作成できます。 一時キューは、接続が終了するまで、またはアプリケーションが TemporaryQueue.delete() メソッドを使用して一時キューを明示的に削除するまで、そのどちらかが起きるまで存在します。

アプリケーションが一時キューを作成すると、IBM MQ classes for JMS は、アプリケーションの接続先になるキュー・マネージャーに動的キューを作成します。 接続ファクトリーの TEMPMODEL プロパティーは、動的キューを作成するために使用するモデル・キューの名前を指定し、接続ファクトリーの TEMPQPREFIX プロパティーは、動的キューの名前を形成するために使用する接頭部を指定します。

セッションを使用した Topic オブジェクトの作成

以下の例が示すように、Topic オブジェクトを作成するために、アプリケーションは Session オブジェクトの createTopic() メソッドを使用できます。

```
Session session;
```

```
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

このコードは、すべてのプロパティーにデフォルト値が指定された Topic オブジェクトを作成します。 オブジェクトは Sport/Football/Results という名前のトピックを表します。

さらに、createTopic() メソッドはトピック URI をパラメーターとして受け入れます。 トピック URI は、トピックの名前と、オプションで 1 つ以上の Topic オブジェクトのプロパティーを指定するストリングです。 以下のコードには、トピック URI の例が含まれています。

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

このコードによって作成されるトピック・オブジェクトは、「Sport/Tennis/Results」というトピックを表し、この宛先に送信されるすべてのメッセージは非永続で、優先順位 0 を持ちます。 トピックの URI について詳しくは、225 ページの『Uniform Resource Identifier (URI)』を参照してください。

createTopic() メソッドのパラメーターには、プロバイダー固有の情報が含まれていることに注意してください。 したがって、Topic オブジェクトを管理対象オブジェクトとして JNDI ネーム・スペースから取り出す代わりに、createTopic() メソッドを使用して Topic オブジェクトを作成すると、アプリケーションの移植性は低くなる場合があります。

以下の例が示すように、アプリケーションは Session オブジェクトの createTemporaryTopic() メソッドを使用して TemporaryTopic オブジェクトを作成できます。

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

セッションは一時トピックを作成するために使用されますが、一時トピックの有効範囲は、セッションを作成するために使用された接続です。 接続のどのセッションでも、一時トピックのためのメッセージ・プロデューサーおよびメッセージ・コンシューマーを作成できます。 一時トピックは、接続が終了するまで、またはアプリケーションが TemporaryTopic.delete() メソッドを使用して一時トピックを明示的に削除するまで、そのどちらかが起きるまで存在します。

アプリケーションが一時トピックを作成すると、IBM MQ classes for JMS は TEMP/*tempTopicPrefix* という文字で始まる名前のトピックを作成します。ここで、 *tempTopicPrefix* は、接続ファクトリーの TEMPTOPICPREFIX プロパティーの値です。

Uniform Resource Identifier (URI)

キュー URI は、IBM MQ キューの名前を指定し、オプションで、キューを所有するキュー・マネージャーの名前、およびアプリケーションにより作成される Queue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 トピック URI は、トピックの名前と、オプションでアプリケーションにより作成される 1 つ以上の Topic オブジェクトのプロパティーを指定するストリングです。

キュー URI の形式は以下のとおりです。

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

トピック URI の形式は以下のとおりです。

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

これらの形式の変数には、以下の意味があります。

qMgrName
URI により識別されるキューを所有するキュー・マネージャーの名前。

キュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合、IBM MQ は、アプリケーションがメッセージをキューに送信するときに WebSphere MQ がメッセージをローカル・キュー・マネージャーからリモート・キュー・マネージャーに経路指定できるように構成する必要があります。

名前が指定されない場合、ローカル・キュー・マネージャーが想定されます。

qName
IBM MQ キューの名前。

キューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

キュー名を作成する規則については、「IBM MQ オブジェクトの命名規則」を参照してください。

topicName
トピックの名前。

トピック名を作成する規則については、「IBM MQ オブジェクトの命名規則」を参照してください。 ワイルドカード文字 +、#、*、および? は使用しないでください。 トピック名で使用できます。 これらの文字を含むトピック名は、サブスクライブするときに予期しない結果になる場合があります。 トピック・ストリングの結合を参照してください。

***propertyName1*, *propertyName2*, ...**
アプリケーションにより作成される Queue または Topic オブジェクトのプロパティー名。 226 ページの表 35 は、URI で使用できる有効なプロパティー名をリストしています。

プロパティーを指定しない場合、Queue または Topic オブジェクトは、そのすべてのプロパティーでデフォルト値を取ります。

***propertyValue1*, *propertyValue2*, ...**
アプリケーションにより作成される Queue または Topic オブジェクトのプロパティーの値。 226 ページの表 35 は、URI で使用できる有効なプロパティーの値をリストしています。

大括弧 ([]) はオプション・コンポーネントを示し、省略符号 (...) は、プロパティーの名前と値の対のリストがあれば、それに 1 つ以上の名前と値の対を含められることを意味します。

226 ページの表 35 は、キューおよびトピック URI で使用できる有効なプロパティー名および有効値をリストしています。 IBM MQ JMS 管理ツールがプロパティーの値にシンボリック定数を使用するとしても、URI にはシンボリック定数を含めることはできません。

表 *35.* キューおよびトピック *URI* で使用するプロパティー名と有効値

プロパティー名	説明	有効値
CCSID	IBM MQ classes for JMS がメッセージを宛先に転送するときに、メッセージの本体の文字データが表記される方法	• IBM MQ によりサポートされる任意のコード化文字セット ID。
encoding	IBM MQ classes for JMS がメッセージを宛先に転送するときに、メッセージの本体の数値データが表記される方法	• IBM MQ メッセージ記述子内の *Encoding* フィールドの任意の有効な値。
expiry	宛先に送信されるメッセージの存続時間	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの存続時間。 • 0 - 宛先に送信されるメッセージは有効期限が切れることはありません。 • ミリ秒で存続時間を指定する正整数。

表 35. キューおよびトピック *URI* で使用するプロパティー名と有効値 (続き)

プロパティー名	説明	有効値
multicast	ブローカーへのリアルタイム接続を使用するときのトピックに対するマルチキャストの設定	以下のリストには有効値が含まれます。各値と関連付けられるのは、IBM MQ JMS 管理ツールで使用される、MULTICAST プロパティーの対応する値です。MULTICAST プロパティーおよびその有効値の説明については、Properties of IBM MQ classes for JMS オブジェクトのプロパティーを参照してください。 • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED
persistence	宛先に送信されるメッセージの持続性	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの持続性。 • -1- IBM MQ キューまたはトピックの *DefPersistence* 属性によって指定されたとおり。 • 1 - 非持続。 • 2 - 持続。 • 3 - IBM MQ JMS 管理ツールで使用される PERSISTENCE プロパティーの値 HIGH と等価。 この値の説明については、256 ページの『JMS 持続メッセージ』を参照してください。
priority	宛先に送信されるメッセージの優先順位	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの優先順位。 • -1- IBM MQ キューまたはトピックの *DefPriority* 属性によって指定されたとおり。 • 宛先に送信されるメッセージの優先順位を指定する、0 から 9 の範囲内の整数。
targetClient	宛先に送信されるメッセージに MQRFH2 ヘッダーが含まれるかどうか。	• 0 - メッセージには MQRFH2 ヘッダーが含まれます。 • 1 - メッセージには MQRFH2 ヘッダーは含まれません。

例えば、以下の URI は、ローカル・キュー・マネージャーにより所有される、Q1 という IBM MQ キューを識別します。 この URI を使用して作成された Queue オブジェクトには、そのすべてのプロパティーのデフォルト値があります。

```
queue:///Q1
```

以下の URI は、QM2 というキュー・マネージャーによって所有される、Q2 という IBM MQ キューを識別します。 この宛先に送信されるすべてのメッセージの優先順位は 6 です。 この URI を使用して作成されたキュー・オブジェクトの残りのプロパティーには、デフォルト値が含まれます。

```
queue://QM2/Q2?priority=6
```

以下の URI は、Sport/Athletics/Results というトピックを識別します。 この宛先に送信されるすべてのメッセージは非永続であり、優先順位は 0 です。 この URI を使用して作成されたトピック・オブジェクトの残りのプロパティーには、デフォルト値が含まれます。

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

JMS アプリケーションでのメッセージの送信

JMS アプリケーションがメッセージを宛先に送信する前に、まず宛先用の MessageProducer オブジェクトを作成する必要があります。 メッセージを宛先に送信するために、アプリケーションは Message オブジェクトを作成し、MessageProducer オブジェクトの send() メソッドを呼び出します。

アプリケーションは MessageProducer オブジェクトを使用してメッセージを送信します。 アプリケーションは通常、MessageProducer オブジェクトを特定の宛先用 (キューまたはトピックが可能) に作成するため、同じメッセージ・プロデューサーを使用して送信されたメッセージは、すべて同じ宛先に送られます。 したがって、アプリケーションが MessageProducer オブジェクトを作成する前に、まず Queue または Topic オブジェクトを作成する必要があります。 Queue または Topic オブジェクトの作成方法について詳しくは、以下のトピックを参照してください。

- 208 ページの『JNDI を使用して JMS または Jakarta Messaging アプリケーションで管理対象オブジェクトを取得する』
- 210 ページの『IBM JMS 拡張機能の使用』
- 217 ページの『IBM MQ JMS 拡張機能の使用』
- 223 ページの『JMS アプリケーションでの宛先の作成』

以下の例が示すように、MessageProducer オブジェクトを作成するために、アプリケーションは Session オブジェクトの createProducer() メソッドを使用できます。

```
MessageProducer producer = session.createProducer(destination);
```

パラメーター `destination` は、アプリケーションによって前もって作成された Queue または Topic オブジェクトです。

アプリケーションがメッセージを送信する前に、Message オブジェクトを作成する必要があります。 メッセージ本体にはアプリケーション・データが含まれており、JMS は以下の 5 タイプのメッセージ本体を定義しています。

- バイト
- マップ
- オブジェクト
- ストリーム
- テキスト

メッセージ本体の各タイプにはそれぞれ固有の JMS インターフェースがあります。これは Message インターフェースのサブインターフェースであり、その本体のタイプでメッセージを作成するための Session インターフェースのメソッドです。 以下のステートメントが示すように、例えばテキスト・メッセージの

インターフェースは TextMessage であり、アプリケーションが Session オブジェクトの createTextMessage() メソッドを使用してテキスト・メッセージを作成します。

```
TextMessage outMessage = session.createTextMessage(outString);
```

メッセージおよびメッセージ本体の詳細については、146 ページの『JMS メッセージ』を参照してください。

以下の例が示すように、メッセージを送信するために、アプリケーションは MessageProducer オブジェクトの send() メソッドを使用します。

```
producer.send(outMessage);
```

アプリケーションは send() メソッドを使用して、どちらのメッセージング・ドメインでもメッセージを送信できます。 宛先の種類によって、どのメッセージング・ドメインが使用されるかが決定されます。 ただし、パブリッシュ/サブスクライブ・ドメイン特定である MessageProducer のサブインターフェースである TopicPublisher には、send() メソッドの代わりに使用できる publish() メソッドもあります。 これら 2 つのメソッドは、機能的には同じです。

アプリケーションは、宛先を指定しない MessageProducer オブジェクトを作成できます。 この場合、アプリケーションは、send() メソッドを呼び出すときに宛先を指定する必要があります。

アプリケーションがトランザクション内でメッセージを送信する場合、トランザクションがコミットされるまで、そのメッセージは宛先に送信されません。 これは、アプリケーションがメッセージを送信できず、同じトランザクション内でメッセージに対する応答を受信できないことを意味します。

アプリケーションがメッセージを送信したときに IBM MQ classes for JMS がメッセージを転送し、キュー・マネージャーがメッセージを支障なく受信したかどうかを判別せずに制御をアプリケーションに戻すように、宛先を構成することが可能です。 これは、非同期書き込み と呼ばれることもあります。 詳しくは、320 ページの『IBM MQ classes for JMS でのメッセージの非同期書き込み』 を参照してください。

JMS アプリケーションでのメッセージの受信

アプリケーションはメッセージ・コンシューマーを使用してメッセージを受信します。 永続トピック・サブスクライバーは、コンシューマーが非アクティブの間に送信されたメッセージも含め、宛先に送信されたすべてのメッセージを受信するメッセージ・コンシューマーです。 アプリケーションは、メッセージ・セレクターを使用することによって、受信するメッセージを選択することができ、メッセージ・リスナーを使用することによって、メッセージを非同期で受信することができます。

アプリケーションは MessageConsumer オブジェクトを使用してメッセージを受信します。 アプリケーションは、特定の宛先 (キューまたはトピック) の MessageConsumer オブジェクトを作成し、メッセージ・コンシューマーを使って受信したすべてのメッセージが同じ宛先から受信されるようにします。 そのため、アプリケーションは MessageConsumer オブジェクトを作成する前に、Queue または Topic オブジェクトを最初に作成しておく必要があります。 Queue または Topic オブジェクトの作成方法について詳しくは、以下のトピックを参照してください。

- 208 ページの『JNDI を使用して JMS または Jakarta Messaging アプリケーションで管理対象オブジェクトを取得する』
- 210 ページの『IBM JMS 拡張機能の使用』
- 217 ページの『IBM MQ JMS 拡張機能の使用』
- 223 ページの『JMS アプリケーションでの宛先の作成』

アプリケーションは、以下の例で示されているように Session オブジェクトの createConsumer() メソッドを使用して MessageConsumer オブジェクトを作成します。

```
MessageConsumer consumer = session.createConsumer(destination);
```

パラメーター `destination` は、アプリケーションによって前もって作成された Queue または Topic オブジェクトです。

その後、アプリケーションは、以下の例で示されているように MessageConsumer オブジェクトの receive() メソッドを使用して宛先からメッセージを受信します。

```
Message inMessage = consumer.receive(1000);
```

receive() 呼び出しのパラメーターは、メッセージを即時に入手できない場合に適切なメッセージが到着するまでメソッドが待機する時間 (ミリ秒単位) を指定します。 このパラメーターを省略すると、呼び出しは適切なメッセージが到着するまで無期限にブロックされます。 アプリケーションがメッセージを待機しないようにするには、代わりに receiveNoWait() メソッドを使用します。

receive() メソッドは、特定のタイプのメッセージを戻します。 例えば、アプリケーションがテキスト・メッセージを受信した場合に、receive() 呼び出しで戻されるオブジェクトは TextMessage オブジェクトです。

しかし、receive() 呼び出しで戻される、宣言されたタイプのオブジェクトは、Message オブジェクトです。そのため、受信したばかりのメッセージの本文からデータを抽出するには、アプリケーションは Message クラスから、より特定的なサブクラス (TextMessage など) にキャストする必要があります。 メッセージのタイプが分からない場合、アプリケーションは `instanceof` 演算子を使用してタイプを判別することができます。 エラーをスムーズに処理できるように、常に、キャストする前にアプリケーションでメッセージのタイプを判別しておくことをお勧めします。

以下のコードは、`instanceof` 演算子を使用し、テキスト・メッセージの本文からデータを抽出する方法を示します。

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

アプリケーションがトランザクション内でメッセージを送信する場合、トランザクションがコミットされるまで、そのメッセージは宛先に送信されません。 これは、アプリケーションがメッセージを送信できず、同じトランザクション内でメッセージに対する応答を受信できないことを意味します。

先読み用に構成されている宛先からメッセージ・コンシューマーがメッセージを受信する場合、アプリケーションが終了したときに先読みバッファーにある非永続メッセージはすべて廃棄されます。

パブリッシュ/サブスクライブ・ドメインで JMS は、非永続トピック・サブスクライバーと永続トピック・サブスクライバーの 2 種類のメッセージ・コンシューマーを識別します。これらについては、以下の 2 つのセクションで説明されます。

非永続トピック・サブスクライバー

非永続トピック・サブスクライバーは、サブスクライバーがアクティブになっている間にパブリッシュされたメッセージのみを受信します。 非永続サブスクリプションは、アプリケーションが非永続トピック・サブスクライバーを作成したときに開始し、アプリケーションがサブスクライバーを閉じるかまたはサブスクライバーが有効範囲から外れたときに終了します。 非永続トピック・サブスクライバーは、IBM MQ classes for JMS の拡張として、保存パブリケーションも受信します。

非永続トピック・サブスクライバーを作成するために、アプリケーションは、宛先として Topic オブジェクトを指定して、ドメイン独立 createConsumer() メソッドを使用できます。 あるいは、以下の例で示されているように、ドメイン特定 createSubscriber() メソッドをアプリケーションで使用することもできます。

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

パラメーター `topic` は、アプリケーションによって前もって作成された Topic オブジェクトです。

永続トピック・サブスクライバー

制約事項 : アプリケーションは、ブローカーとのリアルタイム接続を使用しているときには永続トピック・サブスクライバーを作成できません。

永続トピック・サブスクライバーは、永続サブスクリプションの存続中にパブリッシュされるすべてのメッセージを受信します。 これには、サブスクライバーがアクティブになっていない間にパブリッシュされるすべてのメッセージも含まれます。 永続トピック・サブスクライバーは、IBM MQ classes for JMS の拡張として、保存パブリケーションも受信します。

アプリケーションは、以下の例で示されているように Session オブジェクトの createDurableSubscriber() メソッドを使用して永続トピック・オブジェクトを作成します。

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

createDurableSubscriber() 呼び出しの最初のパラメーターは、アプリケーションによって前もって作成された Topic オブジェクトで、2 番目のパラメーターは、永続サブスクリプションを識別するために使用される名前です。

永続トピック・サブスクライバーを作成するためのセッションには、それに関連付けられたクライアント ID が必要です。 セッションに関連付けられているクライアント ID は、そのセッションの作成時に使用された接続のクライアント ID と同じです。 クライアント ID は、 ConnectionFactory オブジェクトの CLIENTID プロパティーを設定することによって指定できます。 あるいは、アプリケーションから、Connection オブジェクトの setClientID() メソッドを呼び出すことによって、クライアント ID を指定できます。

永続サブスクリプションの識別で使用される名前は、クライアント ID 内でのみ固有であることが必要です。したがって、クライアント ID は、永続サブスクリプションの完全な固有 ID の一部を形成します。 既に作成済みの永続サブスクリプションを使用し続けるには、アプリケーションがその永続サブスクリプションに関連付けられているものと同じクライアント ID を持つセッション、および同じサブスクリプション名を使用して、永続トピック・サブスクライバーを作成する必要があります。

永続サブスクリプションは、永続サブスクリプションが現時点で存在していないクライアント ID とサブスクリプション名を使用してアプリケーションが永続トピック・サブスクライバーを作成するときに開始します。 ただし、永続サブスクリプションは、アプリケーションが永続トピック・サブスクライバーを閉じるときには終了しません。 永続サブスクリプションを終了するには、アプリケーションがその永続サブスクリプションに関連付けられているものと同じクライアント ID を持つ Session オブジェクトの unsubscribe() メソッドを呼び出す必要があります。 以下の例で示されているように、unsubscribe() 呼び出しのパラメーターはサブスクリプション名です。

```
session.unsubscribe("D_SUB_000001");
```

永続サブスクリプションの有効範囲はキュー・マネージャーです。 永続サブスクリプションが 1 つのキュー・マネージャーに存在し、別のキュー・マネージャーに接続されているアプリケーションが同じクライアント ID とサブスクリプション名を持つ永続サブスクリプションを作成する場合、2 つの永続サブスクリプションは完全に独立したものとなります。

メッセージ・セレクター

後続の receive() 呼び出しで特定の条件を満たすメッセージのみを戻すように、アプリケーションで指定することができます。 MessageConsumer オブジェクトを作成する際、どのメッセージを取り出すかを決定する構造化照会言語 (SQL) 式をアプリケーションで指定することができます。 この SQL 式は、メッセージ・セレクター と呼ばれます。 メッセージ・セレクターには JMS メッセージ・ヘッダー・フィールドとメッセージ・プロパティーの名前を含めることができます。 メッセージ・セレクターを構成する方法について詳しくは、146 ページの『JMS のメッセージ・セレクター』を参照してください。

以下の例は、myProp という名前のユーザー定義プロパティーに基づいてアプリケーションでメッセージを選択する方法を示しています。

```
MessageConsumer consumer;
```

```
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

JMS の仕様では、アプリケーションでメッセージ・コンシューマーのメッセージ・セレクターを変更することができません。 アプリケーションがメッセージ・セレクターとともにメッセージ・コンシューマーを作成した後、そのコンシューマーが存続する間、メッセージ・セレクターも存続します。 アプリケーションで複数のメッセージ・セレクターが必要な場合、アプリケーションは各メッセージ・セレクターごとにメッセージ・コンシューマーを作成する必要があります。

アプリケーションがバージョン 7 のキュー・マネージャーと接続されている場合、接続ファクトリーの MSGSELECTION プロパティーには効果がありません。 パフォーマンスを最適化するために、キュー・マネージャーがすべてのメッセージ選択を行います。

ローカル・パブリケーションの抑制

アプリケーションは、コンシューマー独自の接続でパブリッシュされたパブリケーションを無視するメッセージ・コンシューマーを作成することができます。 アプリケーションは、以下の例で示されているように、createConsumer() 呼び出しの 3 番目のパラメーターを `true` に設定することによってこれを行います。

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

createDurableSubscriber() 呼び出しでは、アプリケーションは、以下の例で示されているように 4 番目のパラメーターを `true` に設定することによってこれを行います。

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

メッセージの非同期送達

アプリケーションは、メッセージ・リスナーをメッセージ・コンシューマーに登録することによって、メッセージを非同期で受信することができます。 メッセージ・リスナーは onMessage という名前のメソッドを持ちます。このメソッドは、適切なメッセージが入手可能になると非同期で呼び出されます。メッセージを処理することがこのメソッドの目的です。 以下のコードはそのメカニズムを示しています。

JM 3.0 V 9.3.0 V 9.3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

アプリケーションは、receive() 呼び出しを使用してメッセージを同期的に受信するか、またはメッセージ・リスナーを使用してメッセージを非同期で受信するために、セッションを使用することができます。セッションがその両方を兼ねることはできません。 アプリケーションが同期および非同期でメッセージを受信する必要がある場合、別個にセッションを作成する必要があります。

セッションが非同期でメッセージを受信するようにセットアップされると、以下のメソッドをそのセッションまたはそのセッションから作成されたオブジェクトで呼び出すことはできません。

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()
- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- MessageProducer.send(Destination, Message, CompletionListener)
- MessageProducer.send(Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send(Message, CompletionListener)
- MessageProducer.send(Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()

- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

これらのメソッドのいずれかを呼び出すと、以下のメッセージを含む JMSException がスローされます。

`JMSCC0033: セッションが非同期的に使用されている場合、同期メソッド呼び出しは許可されません。'method name'`

はスローされます。

有害メッセージの受信

アプリケーションは、処理できないメッセージを受信することがあります。 メッセージを処理できない理由はさまざまです。例えば、メッセージの形式が誤っている場合などが考えられます。 このようなメッセージを有害メッセージと呼び、そのメッセージが繰り返し処理されるのを防ぐには、特別な処理が必要になります。

有害メッセージの処理方法の詳細については、236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

受信するメッセージに合わせてバッファー・サイズを調整する

JMS 以外のアプリケーションが IBM MQ からメッセージを受信した場合、メッセージの書き込み先のアプリケーションがメッセージ・バッファーを提供する必要があります。 JMS アプリケーションは、手動でバッファーを作成する必要はありません。 IBM MQ classes for JMS は、受信するメッセージのサイズに合わせて、メッセージ・バッファーを自動的に作成およびサイズ変更します。 ほとんどのアプリケーションでは、自動的に管理されるバッファーが、アプリケーション開発者にとって適切なパフォーマンスと利便性のバランスを提供します。 状況によっては、メッセージ・バッファーの初期サイズを手動で指定すると便利な場合があります。 IBM MQ JMS 受信バッファーのデフォルトの初期サイズは 4 KB です。 アプリケーションが常に 256 KB のサイズのメッセージを受信する場合は、初期バッファー・サイズを 256 KB に構成することをお勧めします。 これにより、 IBM MQ classes for JMS がメッセージを 256 KB にサイズ変更して正常に受信する前に、4 KB バッファーに受信しようとして失敗することを回避できます。 クライアント接続アプリケーションの場合、これにより、 IBM MQ classes for JMS が使用する正しいバッファー・サイズを判別する際に、潜在的に無駄なネットワーク往復が不要になります。

初期バッファー・サイズは、 com.ibm.mq.jmqi.defaultMaxMsgSize Java プロパティーを任意の値 (バイト単位) に設定することによって構成できます。 このプロパティーは、Java Virtual Machine 内で実行されているすべての IBM MQ JMS アプリケーションに影響するため、異なるサイズのメッセージを受信する他のメッセージ・コンシューマーに悪影響を及ぼさないように注意してください。

IBM MQ classes for JMS は、構成されたサイズより小さい複数のメッセージを受信した場合でも、バッファーのサイズを自動的に削減しようとします。 デフォルトでは、これは、すべてバッファー・サイズより小さい 10 個のメッセージを受信した場合に発生します。 例えば、サイズが 128 KB の行で 10 個のメッセージを受信した場合、バッファーは 256 KB から 128 KB に削減されます。 その後、より大きなメッセージ

を受信すると、再度増加します。 バッファーのサイズが削減される前に受信する必要があるメッセージの数を構成することができます。 例えば、アプリケーションが 5 つの大きなメッセージを受信し、その後に 10 個の小さなメッセージを受信し、さらに 5 個の大きなメッセージを受信することが分かっている場合に、これが役立つことがあります。 デフォルト設定では、10 個の小さいメッセージを受信した後にバッファーが削減され、大きいメッセージの場合は再度増やす必要があります。 Java システム・プロパティー com.ibm.mq.jmqi.smallMsgBufferReductionThreshold は、バッファーのサイズが削減される前に受信する必要があるメッセージの数に設定できます。 この例では、10 個の小さいメッセージによってバッファー・サイズが削減されないようにするために、20 に設定することができます。

プロパティーは、相互に独立して設定できます。 例えば、初期バッファー・サイズをデフォルト値の 4 KB のままにする一方で、 com.ibm.mq.jmqi.smallMsgBufferReductionThreshold の値を増やして、バッファーのサイズが増やされると、そのサイズが長くなるようにすることができます。

MQI 統計レコードで多数の MQRC_TRUNCATED_MSG_FAILED (2080) 戻りコードが JMS アプリケーションに表示される場合、これらのアプリケーションに対してより高い初期バッファー・サイズを構成したり、バッファー・サイズを削減する頻度を減らしたりすることで利点が得られる可能性があります。 ただし、長時間実行アプリケーションの場合は、ごく少数の MQRC_TRUNCATED_MSG_FAILED 戻りコードしか表示されない可能性があることに注意してください。 これは、通常、最初の大きなメッセージが受信された直後にバッファーのサイズが正しいサイズに増やされ、小さなメッセージが受信されない限り、サイズが縮小されないためです。 したがって、多数の MQRC_TRUNCATED_MSG_FAILED が、切断前に 1 つまたは 2 つのメッセージのみを受信するために IBM MQ に接続するなど、その他の不適切なアプリケーション・プラクティスを示している可能性があります。

サブスクリプション・ユーザー・データの取得

IBM MQ classes for JMS アプリケーションがキューから取り込むメッセージが、管理上定義された永続サブスクリプションによって書き込まれたものである場合、アプリケーションはそのサブスクリプションに関連付けられたユーザー・データ情報にアクセスする必要があります。 その情報はプロパティーとしてメッセージに追加されています。

MQPS フォルダーを持つ RFH2 ヘッダーを含むキューからメッセージがコンシュームされる場合、Sud キーに関連付けられた値 (存在する場合) が、IBM MQ classes for JMS アプリケーションに返される JMS メッセージ・オブジェクトにストリング・プロパティーとして追加されます。 メッセージからこのプロパティーを取得できるようにするには、 JmsConstants インターフェース内の定数 JMS_IBM_SUBSCRIPTION_USER_DATA を以下のメソッドで使用して、サブスクリプション・ユーザー・データを取得できます。

- JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

以下の例では、MQSC コマンド **DEFINE SUB** を使用して、管理永続サブスクリプションを定義します。

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

トピック・ストリング PUBLIC にパブリッシュされたメッセージのコピーが、キュー MY.SUBSCRIPTION.Q に書き込まれます。 そして、そのメッセージには、この永続サブスクリプションに関連付けたユーザー・データがプロパティーとして追加されます。プロパティーは、キー Sud を使用して RFH2 ヘッダーの MQPS フォルダーに保管されます。

IBM MQ classes for JMS アプリケーションで以下の呼び出しを実行できます。

JM 3.0 V 9.3.0 V 9.3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0
```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

すると、次のストリングが返されます。

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

関連概念
151 ページの『MQRFH2 ヘッダーと JMS』
関連タスク
管理サブスクリプションの定義
関連資料
DEFINE SUB
インターフェース JmsConstants

IBM MQ classes for JMS アプリケーションのクローズ

IBM MQ classes for JMS アプリケーションが停止する前に、特定の JMS オブジェクトを明示的に閉じることは重要です。 ファイナライザーは呼び出せない場合があるので、リソースを解放するためにファイナライザーをあてにしないでください。 圧縮されたトレースをアクティブにしてアプリケーションを終了させないようにしてください。

ガーベッジ・コレクションのみでは、すべての IBM MQ classes for JMS および IBM MQ リソースをタイムリーに解放することはできません。特に、アプリケーションがセッション・レベル以下に多数の一時 JMS オブジェクトを作成する必要がある場合にそう言えます。 このため、必要でなくなった時に、アプリケーションが Connection、Session、MessageConsumer、または MessageProducer オブジェクトを閉じることが重要です。

アプリケーションが接続を閉じないで終了すると、その接続のトランザクション化されたセッションすべてで暗黙的なロールバックが行われます。 アプリケーションによる変更が確実にコミットされるようにするには、アプリケーションを閉じる前に接続を明示的に閉じてください。

JMS オブジェクトを閉じるためにアプリケーション内でファイナライザーを使用しないでください。 ファイナライザーは呼び出せない場合があるので、リソースを解放できないことがあります。 Connection を閉じると、それから作成されたすべての Session も閉じられます。 同様に、Session から作成された MessageConsumer および MessageProducer も、Session を閉じる時に閉じられます。 ただし、リソースがタイムリーな方法で解放されるようにするために、Session、MessageConsumer、および MessageProducer は明示的に閉じることを考慮してください。

トレース圧縮がアクティブ化されている場合、System.Halt() のシャットダウンと、異常で制御されていない JVM の終了は、トレース・ファイルの破損という結果になる可能性があります。 可能であれば、必要とするトレース情報を収集した後、トレース機能をオフにしてください。 アプリケーションを異常終了までトレースする場合は、圧縮されていないトレース出力を使用してください。

注 : キュー・マネージャーから切断するため、JMS アプリケーションは接続オブジェクトに対して close() メソッドを呼び出します。

IBM MQ classes for JMS でのポイズン・メッセージの処理

ポイズン・メッセージは、受信側のアプリケーションでは処理できないメッセージです。 有害メッセージがアプリケーションに配信され、ロールバックされるということが、指定回数発生した場合、 IBM MQ classes for JMS はそのメッセージをバックアウト・キューに移動できます。

有害メッセージは、受信側アプリケーションで処理できないメッセージです。 メッセージに予期しないタイプが含まれているか、アプリケーションのロジックでは処理できない情報が含まれている可能性があります。 有害メッセージがアプリケーションに配信された場合、アプリケーションはそれを処理できず、メッセージを送信元のキューにロールバックします。 デフォルトでは、IBM MQ classes for JMS はメッセージをアプリケーションに繰り返し再配信します。 これにより、アプリケーションは、有害メッセージの処理を試みてロールバックするということを繰り返すループ状態に陥る可能性があります。

IBM MQ classes for JMS は、これを防ぐために、有害メッセージを検出して別の宛先に移動することができます。 IBM MQ classes for JMS は、そのために、以下のプロパティーを使用します。

- 検出されたメッセージの MQMD 内の BackoutCount フィールドの値。
- メッセージが入っている入力キューの IBM MQ キュー属性 **BOTHRESH** (バックアウトしきい値) および **BOQNAME** (バックアウト・リキュー・キュー)。

アプリケーションによってメッセージがロールバックされるたびに、キュー・マネージャーはそのメッセージの BackoutCount フィールドの値を自動的に増やしていきます。

IBM MQ classes for JMS は、BackoutCount がゼロより大きいメッセージを検出すると、BackoutCount の値を **BOTHRESH** 属性の値と比較します。

- BackoutCount が **BOTHRESH** 属性の値より小さい場合、IBM MQ classes for JMS はそのメッセージを処理のためにアプリケーションに送信します。
- しかし、BackoutCount が **BOTHRESH** 以上である場合は、そのメッセージは有害メッセージと見なされます。 この場合、IBM MQ classes for JMS はそのメッセージを **BOQNAME** 属性で指定されたキューに移動します。 メッセージをバックアウト・キューに書き込めない場合は、メッセージのレポート・オプションに応じて、メッセージがキュー・マネージャーの送達不能キューに移動されるか、破棄されます。

注 :

- **BOTHRESH** 属性がデフォルト値の `0` のままである場合は、有害メッセージ処理が無効になります。 この場合、有害メッセージは入力キューに戻されることになります。
- もう 1 つ注意しなければならないのは、IBM MQ classes for JMS は、BackoutCount が 0 より大きいメッセージを最初に検出したときに、キューの **BOTHRESH** 属性と **BOQNAME** 属性を照会することです。 これらの属性の値はキャッシュに入れられ、BackoutCount がゼロより大きいメッセージを IBM MQ classes for JMS が検出するたびに使用されます。

有害メッセージ処理を実行するためのシステムの構成

IBM MQ classes for JMS が **BOTHRESH** 属性および **BOQNAME** 属性を照会するときに使用するキューは、実行されているメッセージングのスタイルによって異なります。

- Point-to-Point メッセージングの場合、これは基礎ローカル・キューです。 JMS アプリケーションが別名キューまたはクラスター・キューのメッセージをコンシュームしている場合、これは重要です。
- パブリッシュ/サブスクライブ・メッセージングの場合は、アプリケーションのメッセージを保持するための管理対象キューが作成されます。 IBM MQ classes for JMS は管理対象キューを照会して、**BOTHRESH** 属性と **BOQNAME** 属性の値を判別します。

  管理対象キューは、アプリケーションのサブスクライブ先の Topic オブジェクトに関連付けられているモデル・キューから作成され、モデル・キューから **BOTHRESH** 属性と **BOQNAME** 属性の値を継承します。 使用されるモデル・キューは、受信側アプリケーションが永続サブスクリプションと非永続サブスクリプションのどちらを取得したかによって異なります。

  – 永続サブスクリプションに使用されるモデル・キューは、Topic の **MDURMDL** 属性によって指定されます。 この属性のデフォルト値は `SYSTEM.DURABLE.MODEL.QUEUE` です。

  – 非永続サブスクリプションの場合、使用されるモデル・キューは **MNDURMDL** 属性によって指定されます。 **MNDURMDL** 属性のデフォルト値は `SYSTEM.NDURABLE.MODEL.QUEUE` です。

BOTHRESH 属性と **BOQNAME** 属性を照会するとき、IBM MQ classes for JMS は以下の処理を行います。

- ローカル・キュー、または別名キューのターゲット・キューを開きます。
- **BOTHRESH** 属性と **BOQNAME** 属性を照会します。
- ローカル・キュー、または別名キューのターゲット・キューを閉じます。

ローカル・キュー、または別名キューのターゲット・キューを開くときに使用されるオープン・オプションは、使用されている IBM MQ classes for JMS のバージョンによって異なります。

- IBM MQ classes for JMS の IBM MQ 9.1.0 Fix Pack 1 以前、または IBM MQ 9.1.1 では、ローカル・キュー (または別名キューのターゲット・キュー) がクラスター・キューの場合、IBM MQ classes for JMS は `MQOO_INPUT_AS_Q_DEF`、`MQOO_INQUIRE` および `MQOO_FAIL_IF_QUIESCING` のオプションを使用してキューを開きます。 これは、受信側アプリケーションを実行しているユーザーが、クラスター・キューのローカル・インスタンスに対する「照会および取得」アクセス権を持っていなければならないことを意味します。

  IBM MQ classes for JMS は、他のすべてのタイプのローカル・キューを、オープン・オプション `MQOO_INQUIRE` および `MQOO_FAIL_IF_QUIESCING` を使用して開きます。 IBM MQ classes for JMS が属性の値を照会するためには、受信側アプリケーションを実行しているユーザーがローカル・キューに対する照会アクセス権を持っていなければなりません。

- IBM MQ classes for JMS の IBM MQ 9.1.0 Fix Pack 2 以降、または IBM MQ 9.1.2 以降を使用している場合は、キューのタイプに関係なく、受信側アプリケーションを実行しているユーザーがローカル・キューに対する照会アクセス権を持っていなければなりません。

有害メッセージをバックアウト・リキュー・キューまたはキュー・マネージャーの送達不能キューに移動するには、アプリケーションを実行するユーザーに put および passall 権限を付与する必要があります。

同期アプリケーションの有害メッセージの処理

以下のいずれかのメソッドを呼び出すことによって、アプリケーションがメッセージを同期的に受信すると、IBM MQ classes for JMS は、アプリケーションがメッセージを取得しようとしたときにアクティブだった作業単位内で有害メッセージをリキューします。

- JMSConsumer.receive()
- JMSConsumer.receive(long timeout)
- JMSConsumer.receiveBody(Class<T> c)
- JMSConsumer.receiveBody(Class<T> c, long timeout)
- JMSConsumer.receiveBodyNoWait Class<T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive()
- MessageConsumer.receive(long timeout)
- MessageConsumer.receiveNoWait()
- QueueReceiver.receive()
- QueueReceiver.receive(long timeout)
- QueueReceiver.receiveNoWait()
- TopicSubscriber.receive()
- TopicSubscriber.receive(long timeout)
- TopicSubscriber.receiveNoWait()

つまりアプリケーションが、トランザクション化されている JMS コンテキストまたはセッションを使用している場合、バックアウト・キューへのメッセージの移動処理は、トランザクションがコミットされるまでコミットされません。

BOTHRESH 属性がゼロ以外の値に設定されている場合は、**BOQNAME** 属性も設定する必要があります。**BOTHRESH** がゼロより大きい値に設定されていて、**BOQNAME** が設定されていない場合の動作は、メッセージのレポート・オプションによって決まります。

- メッセージにレポート・オプション MQRO_DISCARD_MSG が設定されている場合、そのメッセージは破棄されます。
- メッセージにレポート・オプション MQRO_DEAD_LETTER_Q が指定されている場合、IBM MQ classes for JMS はメッセージをキュー・マネージャーの送達不能キューに移動しようとします。
- メッセージに MQRO_DISCARD_MSG も MQRO_DEAD_LETTER_Q も設定されていない場合、IBM MQ classes for JMS はメッセージをキュー・マネージャーの送達不能キューに入れようとします。

送達不能キューへのメッセージの書き込みが何らかの理由で失敗した場合にメッセージがどのように処理されるかは、受信側アプリケーションが使用している JMS コンテキストまたはセッションがトランザクション化されているかどうかによって決まります。

- 受信側アプリケーションが、トランザクション化された JMS コンテキストまたはセッションを使用していて、そのトランザクションがコミットされている場合、メッセージは破棄されます。
- 受信側アプリケーションがトランザクション化された JMS コンテキストまたはセッションを使用していて、そのトランザクションをロールバックした場合、メッセージは入力キューに返されます。
- 受信側アプリケーションが非トランザクションの JMS コンテキストまたはセッションを作成した場合、メッセージは破棄されます。

非同期アプリケーションの有害メッセージの処理

アプリケーションが MessageListener を介して非同期にメッセージを受信している場合、IBM MQ classes for JMS はメッセージ配信に影響を与えることなく有害メッセージをリキューします。 このリキュー処理は、アプリケーションへの実際のメッセージ送達に関連付けられたすべての作業単位の外側で実行されます。

BOTHRESH がゼロより大きい値に設定されていて、**BOQNAME** が設定されていない場合の動作は、メッセージのレポート・オプションによって決まります。

- メッセージにレポート・オプション MQRO_DISCARD_MSG が設定されている場合、そのメッセージは破棄されます。
- メッセージにレポート・オプション MQRO_DEAD_LETTER_Q が指定されている場合、IBM MQ classes for JMS はメッセージをキュー・マネージャーの送達不能キューに移動しようとします。
- メッセージに MQRO_DISCARD_MSG も MQRO_DEAD_LETTER_Q も設定されていない場合、IBM MQ classes for JMS はメッセージをキュー・マネージャーの送達不能キューに入れようとします。

何らかの理由で送達不能キューへのメッセージの書き込みが失敗した場合、IBM MQ classes for JMS はメッセージを入力キューに返します。

活動化仕様および ConnectionConsumer が有害メッセージを処理する方法については、ASF でのキューからのメッセージの除去を参照してください。

バックアウト・キューに移動されたメッセージの処理

有害メッセージがバックアウト・リキュー・キューにリキューされると、IBM MQ classes for JMS はそのメッセージに RFH2 ヘッダーを追加し (まだない場合)、メッセージ記述子 (MQMD) 内のいくつかのフィールドを更新します。

有害メッセージに (例えば、そのメッセージが JMS メッセージであるために) RFH2 ヘッダーが含まれている場合、メッセージをバックアウト・リキュー・キューに移動する際に、IBM MQ classes for JMS は MQMD 内の以下のフィールドを変更します。

- 「バックアウト数」フィールドがゼロにリセットされます。
- メッセージの「有効期限」フィールドが、JMS アプリケーションが有害メッセージを受け取った時点での残りの有効期限を反映するように更新されます。

有害メッセージに RFH2 ヘッダーが含まれていない場合は、バックアウト処理の一環として、IBM MQ classes for JMS がヘッダーを 1 つ追加し、MQMD 内の以下のフィールドを更新します。

- 「バックアウト数」フィールドがゼロにリセットされます。
- メッセージの「有効期限」フィールドが、JMS アプリケーションが有害メッセージを受け取った時点での残りの有効期限を反映するように更新されます。
- メッセージの「形式」フィールドが MQHRF2 に変更されます。
- CCSID フィールドが 1208 に変更されます。
- 「エンコード」フィールドが 273 に変更されます。

これに加えて、有害メッセージの「CCSID」フィールドと「エンコード」フィールドが RFH2 ヘッダーの「CCSID」フィールドと「エンコード」フィールドにコピーされ、バックアウト・リキュー・キュー上のメッセージのヘッダー・チェーンが正しいことが保証されます。

関連概念

337 ページの『ASF での有害メッセージの処理』
Application Server Facilities (ASF) では、有害メッセージの処理の方法が IBM MQ classes for JMS 内の他の場所とは少し異なっています。

IBM MQ classes for JMS での例外

IBM MQ classes for JMS アプリケーションは、JMS API 呼び出しによってスローされた例外 または例外ハンドラーに渡された例外を 処理できなければなりません。

IBM MQ classes for JMS は、例外をスローすることで実行時の問題をレポートします。 スローされる例外のタイプ、およびこれらの例外の処理方法は、アプリケーションによって使用される JMS 仕様のバージョンによって異なります。

- JMS 1.1 以前で定義されたインターフェースのメソッドでは、チェック例外がスローされます。 これらの例外の基底クラスは JMSException です。 チェック例外の処理方法について詳しくは、240 ページの『チェック例外の処理』を参照してください。
- JMS 2.0 で追加されたインターフェースのメソッドは、チェックされていない例外をスローします。 これらの例外の基底クラスは JMSRuntimeException です。 チェックなし例外の処理方法について詳しくは、243 ページの『チェックなし例外の処理』を参照してください。

JMS の Connection または JMSContext に ExceptionListener を登録することもできます。 こうすると、キュー・マネージャーへの接続で問題が検出された場合、またはメッセージを非同期で配信しようとしているときに問題が発生した場合に、MQ classes for JMS が ExceptionListener に通知します。 詳しくは、246 ページの『ExceptionListeners』を参照してください。

関連概念

IBM MQ classes for JMS

関連資料

ASYNCEXCEPTION

チェック例外の処理

JMS 1.1 以前で定義されたインターフェースのメソッドでは、チェック例外がスローされます。 これらの例外の基本クラスは JMSException です。 したがって、 JMSExceptions をキャッチすると、これらのタイプの例外を処理する一般的な方法が提供されます。

すべての JMSException は、以下の情報をカプセル化します。

- アプリケーションが Throwable.getMessage() メソッドを呼び出すことによって取得できる、プロバイダー固有の例外メッセージ。
- アプリケーションが JMSException.getErrorCode() メソッドを呼び出すことによって取得できる、プロバイダー固有のエラー・コード。
- リンクの例外。 JMS 1.1 API 呼び出しによりスローされる例外の多くは下位レベルの問題の結果であり、この例外にリンクされた別の例外により報告されます。 アプリケーションは、JMSException.getLinkedException() メソッドまたは Throwable.getCause() メソッドのいずれかを呼び出すことによって、リンクされた例外を取得できます。

JMS 1.1 API を使用する場合、 IBM MQ classes for JMS によってスローされるほとんどの例外は、JMSException のサブクラスのインスタンスです。 これらのサブクラスは、以下の追加情報を提供する com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装します。

- 例外メッセージの説明。 アプリケーションは、JmsExceptionDetail.getExplanation() メソッドを呼び出すことによってこのメッセージを取得できます。
- 例外に対する推奨されるユーザー応答。 アプリケーションは、JmsExceptionDetail.getUserAction() メソッドを呼び出すことによってこのメッセージを取得できます。
- 例外メッセージ内のメッセージ挿入のためのキー。 アプリケーションは、JmsExceptionDetail.getKeys() メソッドを呼び出すことによって、すべてのキーのイテレーターを取得できます。
- 例外メッセージ内のメッセージ挿入。 例えば、メッセージ挿入項目が例外の原因となったキューの名前である場合は、アプリケーションでその名前にアクセスすると役に立つ場合があります。 アプリケーションは、JmsExceptionDetail.getValue() メソッドを呼び出すことによって、指定されたキーに対応するメッセージ挿入を取得できます。

使用可能な詳細がない場合、 JmsExceptionDetail インターフェースのすべてのメソッドはヌルを返します。

例えば、存在しない IBM MQ キュー用のメッセージ・プロデューサーをアプリケーションが 作成しようとすると、以下の情報を持つ例外がスローされます。

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

スローされる例外 com.ibm.msg.client.jms.DetailedInvalidDestinationException は、以下のクラスのサブクラスであり、 com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装します。

- JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms.InvalidDestinationException
- JMS 2.0 javax.jms.InvalidDestinationException

リンクされた例外

リンクされた例外は、実行時の問題に関する詳細な情報を提供します。 したがって、スローされる JMSException ごとに、アプリケーションはリンクされた例外を検査する必要があります。

リンク付きの例外自体には、別のリンク付きの例外がある場合があるため、リンク付きの例外は元の根本の問題に戻るチェーンを形成します。 リンクされた例外は、 java.lang.Throwable クラスのチェーニングされた例外メカニズムを使用して実装されます。アプリケーションは、 Throwable.getCause() メソッドを呼び出すことによって、リンクされた例外を取得できます。 JMSException の場合、 getLinkedException() メソッドは Throwable.getCause() メソッドに委任します。

例えば、キュー・マネージャーへの接続時にアプリケーションが誤ったポート番号を指定する場合、例外は以下のチェーンを形成します。

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

一般に、チェーン内の各例外は、コード内の異なるレイヤーからスローされます。 例えば、先行するチェーン内の例外は、以下のレイヤーによりスローされます。

- の例外 (JMSException のサブクラスのインスタンス) は、 IBM MQ classes for JMS の共通レイヤーによってスローされます。
- 次の例外である com.ibm.mq.MQException のインスタンスは、 IBM MQ メッセージング・プロバイダーによってスローされます。
- 次の 2 つの例外 (両方とも com.ibm.mq.jmqi.JmqiException のインスタンス) は、 Java Message Queueing Interface (JMQI) によってスローされます。 JMQI は、キュー・マネージャーと通信するために IBM MQ classes for JMS によって使用されるコンポーネントです。
- 最後の例外である java.net.ConnectionException のインスタンスは、 Java クラス・ライブラリーによってスローされます。

IBM MQ classes for JMS の階層化アーキテクチャーについて詳しくは、「JMS アーキテクチャーの IBM MQ クラス」を参照してください。

次の例に示すように、このチェーンを反復して、すべての該当する情報を抽出するようにアプリケーションをコーディングすることができます。

JM 3.0 V 9.3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
```

```
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

例外のタイプは異なる場合があり、さまざまなタイプの例外がさまざまな情報をカプセル化するため、アプリケーションはチェーン内の各例外のタイプを常時確認する必要があることに注意してください。

問題に関する IBM MQ 固有情報の入手

com.ibm.mq.MQException および com.ibm.mq.jmqi.JmqiException のインスタンスは、問題に関する IBM MQ 固有の情報をカプセル化します

MQException は、以下の情報をカプセル化します。

- アプリケーションが getCompCode() メソッドを呼び出すことによって取得できる完了コード。
- アプリケーションが getReason() メソッドを呼び出すことによって取得できる理由コード。:

これらのメソッドの使用方法の例については、リンクされた例外のサンプル・コードを参照してください。

JmqiException は、完了コードと理由コードもカプセル化します。 これに加えて、 JmqiException には、AMQ *Nnnn* または CSQ *Nnnn* メッセージの情報が含まれています (例外に関連付けられている場合)。 アプリケーションは、以下のメソッドを呼び出すことによって、このメッセージのさまざまなコンポーネントを取得できます。

- getWmqMsgExplanation() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの説明を返します。
- getWmqMsgSeverity() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの重大度を返します。
- getWmqMsgSummary() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの要約を返します。
- getWmqMsgUserResponse() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージに関連付けられたユーザー応答を返します。

チェックなし例外の処理
JMS 2.0 で定義されたインターフェースのメソッドでは、チェックなし例外がスローされます。 これらの例外の基本クラスは JMSRuntimeException です。 したがって、 JMSRuntimeExceptions をキャッチすると、これらのタイプの例外を処理する一般的な方法が提供されます。

すべての JMSRuntimeException は、以下の情報をカプセル化します。

- アプリケーションが JMSRuntimeException.getMessage() メソッドを呼び出すことによって取得できる、プロバイダー固有の例外メッセージ。
- アプリケーションが JMSRuntimeException.getErrorCode() メソッドを呼び出すことによって取得できる、プロバイダー固有のエラー・コード。
- リンクの例外。 JMS 2.0 API 呼び出しによりスローされる例外の多くは下位レベルの問題の結果であり、この例外にリンクされた別の例外により報告されます。 アプリケーションは、 JMSRuntimeException.getCause() メソッドを呼び出すことによって、リンクされた例外を取得できます。

JMS 2.0 API によって提供されるインターフェースでメソッドを呼び出す場合、 IBM MQ classes for JMS によってスローされるほとんどの例外は、 JMSRuntimeException のサブクラスのインスタンスです。 これらのサブクラスは com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装しますが、これは以下の追加情報を提供します。

- 例外メッセージの説明。 アプリケーションは、 JmsExceptionDetail.getExplanation() メソッドを呼び出すことによってこのメッセージを取得できます。

- 例外に対する推奨されるユーザー応答。 アプリケーションは、JmsExceptionDetail.getUserAction() メソッドを呼び出すことによってこのメッセージを取得できます。
- 例外メッセージ内のメッセージ挿入のためのキー。 アプリケーションは、JmsExceptionDetail.getKeys() メソッドを呼び出すことによって、すべてのキーのイテレーターを取得できます。
- 例外メッセージ内のメッセージ挿入。 例えば、メッセージ挿入項目が例外の原因となったキューの名前である場合は、アプリケーションでその名前にアクセスすると役に立つ場合があります。 アプリケーションは、JmsExceptionDetail.getValue() メソッドを呼び出すことによって、指定されたキーに対応するメッセージ挿入を取得できます。

使用可能な詳細がない場合、 JmsExceptionDetail インターフェースのすべてのメソッドはヌルを返します。

例えば、存在しない IBM MQ キューに対してアプリケーションが JMSProducer を作成しようとすると、以下の情報を含む例外がスローされます。

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

スローされる例外 com.ibm.msg.client.jms.DetailedInvalidDestinationException は、以下のクラスのサブクラスであり、 com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装します。

- JM 3.0 V 9.3.0 V 9.3.0 jakarta.jms.InvalidDestinationException
- JMS 2.0 javax.jms.InvalidDestinationException

チェーニングされた例外

通常、例外は他の例外によって発生します。 したがって、スローされる JMSRuntimeException ごとに、アプリケーションはリンクされた例外を検査する必要があります。

JMSRuntimeException の原因として、別の例外が考えられます。 これらの例外は元の根本の問題に戻るチェーンを形成します。 例外の原因は、 java.lang.Throwable クラスのチェーニングされた例外メカニズムを使用して実装されます。アプリケーションは、 Throwable.getCause() メソッドを呼び出すことによって、リンクされた例外を取得できます。

例えば、キュー・マネージャーへの接続時にアプリケーションが誤ったポート番号を指定する場合、例外は以下のチェーンを形成します。

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

一般に、チェーン内の各例外は、コード内の異なるレイヤーからスローされます。 例えば、先行するチェーン内の例外は、以下のレイヤーによりスローされます。

- の例外 (JMSRuntimeException のサブクラスのインスタンス) は、 IBM MQ classes for JMS の共通レイヤーによってスローされます。
- 次の例外である com.ibm.mq.MQException のインスタンスは、IBM MQ メッセージング・プロバイダーによってスローされます。
- 次の 2 つの例外 (両方とも com.ibm.mq.jmqi.JmqiException のインスタンス) は、 Java Message Queueing Interface (JMQI) によってスローされます。 JMQI は、キュー・マネージャーと通信するために IBM MQ classes for JMS によって使用されるコンポーネントです。
- 最後の例外である java.net.ConnectionException のインスタンスは、Java クラス・ライブラリーによってスローされます。

IBM MQ classes for JMS の階層化アーキテクチャーについて詳しくは、「JMS アーキテクチャーの IBM MQ クラス」を参照してください。

次の例に示すように、このチェーンを反復して、すべての該当する情報を抽出するようにアプリケーションをコーディングすることができます。

JM 3.0 V 9.3.0 V 9.3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
```

```
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

例外のタイプは異なる場合があり、さまざまなタイプの例外がさまざまな情報をカプセル化するため、アプリケーションはチェーン内の各例外のタイプを常時確認する必要があることに注意してください。

問題に関する IBM MQ 固有情報の入手

com.ibm.mq.MQException および com.ibm.mq.jmqi.JmqiException のインスタンスは、問題に関する IBM MQ 固有の情報をカプセル化します

MQException は、以下の情報をカプセル化します。

- アプリケーションが getCompCode() メソッドを呼び出すことによって取得できる完了コード。
- アプリケーションが getReason() メソッドを呼び出すことによって取得できる理由コード。:

これらのメソッドの使用方法の例については、チェーニングされた例外のサンプル・コードを参照してください。

JmqiException は、完了コードと理由コードもカプセル化します。 これに加えて、 JmqiException には、AMQ *Nnnn* または CSQ *Nnnn* メッセージの情報が含まれています (例外に関連付けられている場合)。 アプリケーションは、以下のメソッドを呼び出すことによって、このメッセージのさまざまなコンポーネントを取得できます。

- getWmqMsgExplanation() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの説明を返します。
- getWmqMsgSeverity() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの重大度を返します。
- getWmqMsgSummary() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージの要約を返します。
- getWmqMsgUserResponse() メソッドは、AMQ *Nnnn* または CSQ *Nnnn* メッセージに関連付けられたユーザー応答を返します。

ExceptionListeners
JMSConnection および JMSContext オブジェクトには、キュー・マネージャーへの接続が関連付けられています。 アプリケーションは、 ExceptionListener を JMS Connection または JMSContext に登録できます。 Connection または JMSContext に関連付けられた接続を使用不可にする問題が発生した場合、 IBM MQ classes for JMS は ExceptionListener メソッドを呼び出すことによって onException() に例外を送信します。 次いで、アプリケーションには、接続を再確立する機会があります。

IBM MQ classes for JMS は、メッセージを非同期に送達しようとしているときに問題が発生した場合は、例外を例外リスナーにも送達できます。

例外リスナー

IBM MQ 8.0.0 Fix Pack 2,以降、JMS `MessageListener` および JMS `ExceptionListener` を構成する現行 JMS アプリケーションの動作を維持し、IBM MQ classes for JMS が JMS 仕様と整合するようにするために、 `ConnectionFactory` プロパティー 非同期例外 のデフォルト値が ASYNC_EXCEPTIONS_CONNECTIONBROKEN に変更されました。 結果として、切断された接続エラー・コードに対応する例外のみが、アプリケーションの `ExceptionListener` に送信されます。

APAR IT14820(IBM MQ 9.0.0 Fix Pack 1 に含まれています) は、 IBM MQ classes for JMS を更新して以下を可能にします。

- アプリケーションによって登録された `ExceptionListener` は、アプリケーションが同期メッセージ・コンシューマーを使用しているか非同期メッセージ・コンシューマーを使用しているかに関係なく、すべての接続切断例外に対して呼び出されます。
- アプリケーションが非同期メッセージ・コンシューマーを使用しており、アプリケーションが使用する `ExceptionListener` ConnectionFactory の ASYNC_EXCEPTION プロパティーの値が ASYNC_EXCEPTIONS_ALL に設定されている場合、メッセージ配信中に発生する非接続切断例外 (MQRC_GET_INHIBITED など) は、アプリケーションの JMS に配信されます。

注 : `ExceptionListener` は、2 つの TCP/IP 接続 (1 つは JMS 接続によって使用され、もう 1 つは JMS セッションによって使用される) が切断された場合でも、接続切断例外のために 1 回だけ呼び出されます。

それ以外の種類の問題が発生した場合は、その時点の JMS API 呼び出しによって例外がスローされます。スローされる例外のタイプは、アプリケーションが使用している JMS API のバージョンによって異なります。

- アプリケーションが JMS 1.1 仕様で提供されているインターフェースを使用している場合、例外は `JMSException` です。 これらの例外の処理方法について詳しくは、240 ページの『チェック例外の処理』を参照してください。
- アプリケーションが JMS 2.0 インターフェースを使用している場合、例外は `JMSRuntimeException` です。 これらの例外の処理方法について詳しくは、243 ページの『チェックなし例外の処理』を参照してください。

アプリケーションが例外リスナーを `Connection` または `JMSContext` に登録しない場合、例外リスナーに配信されるすべての例外が IBM MQ classes for JMS ログに書き込まれます。

IBM MQ classes for JMS アプリケーションから *IBM MQ* 機能へのアクセス

IBM MQ classes for JMS は、IBM MQ のいくつかの機能を活用する機能を提供します。

重要 : これらの機能は JMS 仕様の枠外であるか、またはある場合においては JMS 仕様に違反するものです。 それらを使用すると、ご使用のアプリケーションが他の JMS プロバイダーとの互換性を失う恐れがあります。 それら JMS 仕様に準拠しない機能には「重要」通知が付されます。

IBM MQ classes for JMS アプリケーションからのメッセージ記述子の読み取りと書き込み

宛先やメッセージのプロパティーを設定することによって、メッセージ記述子 (MQMD) へのアクセス権限を制御します。

IBM MQ アプリケーションの中には、送られてくるメッセージの MQMD に特定の値が設定されていることが必要なものもあります。 IBM MQ classes for JMS は、JMS アプリケーションが MQMD フィールドを設定することにより、JMS アプリケーションが IBM MQ アプリケーションを「駆動」できるようにするためのメッセージ属性を提供します。

MQMD プロパティーの設定値が有効になるように、宛先オブジェクト・プロパティー WMQ_MQMD_WRITE_ENABLED を true に設定する必要があります。 そうすれば、MQMD フィールドに値を割り当てるために、メッセージのプロパティー設定方式 (例えば、setStringProperty) を使用することができます。 StrucId と Version を除くすべての MQMD フィールドが公開されます。BackoutCount は読み取り可能ですが、書き込むことはできません。

この例では、MQMD.UserIdentifier が「JoeBloggs」に設定されたキューまたはトピックにメッセージが書き込まれます。

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

JMS_IBM_MQMD_UserIdentifier を設定する前に WMQ_MQMD_MESSAGE_CONTEXT を設定する必要があります。 WMQ_MQMD_MESSAGE_CONTEXT の使用に関する詳細については、250 ページの『JMS メッセージ・オブジェクトのプロパティー』を参照してください。

同様に、メッセージを受信する前に WMQ_MQMD_READ_ENABLED を true に設定することによって MQMD フィールドの内容を抽出し、次いで getStringProperty のようなメッセージのゲット・メソッドを使用できます。 受信するプロパティーはすべて読み取り専用です。

次の例では、メッセージの MQMD.ApplIdentityData フィールドの値を保持している *value* フィールドが、キューまたはトピックから取得される結果となります。

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

JMS 宛先オブジェクトのプロパティー
宛先オブジェクトの 2 つのプロパティーは、JMS からの MQMD へのアクセスを制御し、3 番目のものはメッセージのコンテキストを制御します。

表 *36.* プロパティーの名前および説明

Property	**短縮形**	**説明**
WMQ_MQMD_WRITE_ENABLED	MDW	JMS アプリケーションが MQMD フィールドの値を設定できるかどうか
WMQ_MQMD_READ_ENABLED	MDR	JMS アプリケーションが MQMD フィールドの値を抽出できるかどうか
WMQ_MQMD_MESSAGE_ CONTEXT	MDCTX	JMS アプリケーションによって、メッセージ・コンテキストのどのレベルが設定されるか。このプロパティーが有効となるためには、アプリケーションが適切なコンテキスト権限を持って実行されていなければなりません。

表 37. プロパティーの名前、値、および設定方式

Property	管理ツール内での有効値 (太字はデフォルト)	プログラム内での有効値	Set メソッド
WMQ_MQMD_WRITE_ENABLED	• **NO** JMS_IBM_MQMD* プロパティーはすべて無視され、その値は基礎となる MQMD 構造にコピーされません。 • YES JMS_IBM_MQMD* プロパティーは処理されます。 それらの値は基礎となる MQMD 構造にコピーされます。	• **False** • True	setMQMDWriteEnabled
WMQ_MQMD_READ_ENABLED	• **NO** メッセージの送信時に、送信されたメッセージの JMS_IBM_MQMD* プロパティーは、MQMD での更新済みのフィールド値を反映するように更新されません。 メッセージを受信するときに、送信側が JMS_IBM_MQMD* プロパティーの一部またはすべてを設定した場合でも、受信されたメッセージでそのプロパティーを使用できません。 • YES メッセージ送信の際に、送られるメッセージ上のすべての JMS_IBM_MQMD* プロパティーは、送信者が明示的に設定しないかったものも含め、MQMD 内の更新されたフィールド値を反映して更新されます。 メッセージを受信するときに、受信されたメッセージですべての JMS_IBM_MQMD* プロパティー (送信側が明示的に設定しなかったものを含む) を使用できます。	• **False** • True	setMQMDReadEnabled

表 37. プロパティーの名前、値、および設定方式 (続き)			
Property	**管理ツール内での有効値 (太字はデフォルト)**	**プログラム内での有効値**	**Set メソッド**
WMQ_MQMD _MESSAGE_CONTEXT	• **DEFAULT** MQOPEN API 呼び出しおよび MQPMO 構造体は、メッセージ・コンテキスト・オプションを明示的に指定しません。 • SET_IDENTITY_CONTEXT MQOPEN API 呼び出しは メッセージ・コンテキスト・ オプション MQOO_SET_ IDENTITY_CONTEXT を指定し、MQPMO 構造体は MQPMO_SET_IDENTITY_ CONTEXT を指定します。 • SET_ALL_CONTEXT MQOPEN API 呼び出しは メッセージ・コンテキスト・ オプション MQOO_SET_ALL_ CONTEXT を指定し、MQPMO 構造体は MQPMO_SET_ALL_ CONTEXT を指定します。	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

JMS メッセージ・オブジェクトのプロパティー
JMS_IBM_MQMD という接頭部を持つメッセージ・オブジェクトのプロパティーは、対応する MQMD フィールドを設定したり読み取ったりすることを可能にします。

メッセージの送信

StrucId と Version を除くすべての MQMD フィールドが表されます。 これらのプロパティーは MQMD フィールドのみを参照しています。このフィールドでは、MQMD ヘッダーと MQRFH2 ヘッダーの両方でプロパティーが発生し、MQRFH2 のバージョンは設定も抽出もされません。

JMS_IBM_MQMD_BackoutCount を除き、これらのすべてのプロパティーを設定できます。 JMS_IBM_MQMD_BackoutCount に設定された値はすべて無視されます。

プロパティーが最大長を持っていて、長過ぎる値が提供された場合、その値は切り捨てられます。

特定のプロパティーについては、宛先オブジェクト上の WMQ_MQMD_MESSAGE_CONTEXT プロパティーも設定する必要があります。 このプロパティーが有効となるためには、アプリケーションが適切なコンテキスト権限を持って実行されていなければなりません。 WMQ_MQMD_MESSAGE_CONTEXT に適切な値を設定しない場合、プロパティーの値は無視されます。 WMQ_MQMD_MESSAGE_CONTEXT を適切な値に設定したものの、キュー・マネージャーに対する十分なコンテキスト権限がない場合、JMSException が発行されます。 WMQ_MQMD_MESSAGE_CONTEXT の特定の値を必要とするプロパティーは、以下のものです。

次のプロパティーでは、WMQ_MQMD_MESSAGE_CONTEXT が WMQ_MDCTX_SET_IDENTITY_CONTEXT または WMQ_MDCTX_SET_ALL_CONTEXT に設定される必要があります。

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

次のプロパティーでは、WMQ_MQMD_MESSAGE_CONTEXT が WMQ_MDCTX_SET_ALL_CONTEXT に設定される必要があります。

- JMS_IBM_MQMD_PutApplType

- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

メッセージの受信

受信メッセージ上のこれらすべてのプロパティーは、WMQ_MQMD_READ_ENABLED プロパティーが true に設定されていれば、メッセージを作成したアプリケーションで設定した実際のプロパティーと関係なく利用できます。 JMS 仕様によれば、すべてのプロパティーをまずクリアしない限り、アプリケーションが受信メッセージのプロパティーを変更することはできません。 プロパティーを変更せずに、受信メッセージを転送することができます。

重要 : ご使用のアプリケーションが、WMQ_MQMD_READ_ENABLED プロパティーが true に設定されている宛先からメッセージを受信し、そのメッセージを WMQ_MQMD_WRITE_ENABLED が true に設定されている宛先に転送すると、受信されたメッセージのすべての MQMD フィールド値が転送メッセージにコピーされるという結果になります。

プロパティーの表

この表は、MQMD フィールドを表すメッセージ・オブジェクトのプロパティーのリストを示します。 フィールドと、許容されている値についての完全な説明については、リンク先を参照してください。

表 *38.* プロパティーの名前、説明、およびタイプ

Property	**説明**	**Java タイプ**	**完全な説明のリンク先**
JMS_IBM_MQMD_Report	レポート・メッセージのオプション	整数	レポート
JMS_IBM_MQMD_MsgType	メッセージ・タイプ	整数	MsgType
JMS_IBM_MQMD_Expiry	メッセージの存続時間。	整数	Expiry
JMS_IBM_MQMD_Feedback	フィードバックまたは理由コード	整数	Feedback
JMS_IBM_MQMD_Encoding	メッセージ・データの数値エンコード	整数	Encoding
JMS_IBM_MQMD_CodedCharSetId	メッセージ・データの文字セット ID	整数	CodedCharSetId
JMS_IBM_MQMD_Format	メッセージ・データの形式名。	ストリング	形式
JMS_IBM_MQMD_Priority [1]	メッセージ優先順位	整数	priority
JMS_IBM_MQMD_Persistence	メッセージの持続性	整数	永続性
JMS_IBM_MQMD_MsgId [2]	メッセージ ID	Object (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	相関 ID	Object (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	バックアウトのカウンター	整数	BackoutCount
JMS_IBM_MQMD_ReplyToQ	応答キューの名前	ストリング	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	応答キュー・マネージャーの名前	ストリング	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	ユーザー ID	ストリング	UserIdentifier

表 38. プロパティーの名前、説明、およびタイプ (続き)

Property	説明	Java タイプ	完全な説明のリンク先
JMS_IBM_MQMD_AccountingToken	アカウンティング・トークン	Object (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	ID に関連するアプリケーション・データ	ストリング	ApplIdentityData
JMS_IBM_MQMD_PutApplType	メッセージを書き込んだアプリケーションのタイプ	整数	PutApplType
JMS_IBM_MQMD_PutApplName	メッセージを書き込んだアプリケーションの名前	ストリング	PutApplName
JMS_IBM_MQMD_PutDate	メッセージを書き込んだ日付	ストリング	PutDate
JMS_IBM_MQMD_PutTime	メッセージを書き込んだ時刻	ストリング	PutTime
JMS_IBM_MQMD_ApplOriginData	発生元に関係するアプリケーション・データ	ストリング	ApplOriginData
JMS_IBM_MQMD_GroupId	グループ ID	Object (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	グループ中の論理メッセージの順序番号	整数	MsgSeqNumber
JMS_IBM_MQMD_Offset	論理メッセージの先頭を起点とする、物理メッセージ中のデータのオフセット	整数	オフセット
JMS_IBM_MQMD_MsgFlags	メッセージ・フラグ	整数	MsgFlags
JMS_IBM_MQMD_OriginalLength	元のメッセージの長さ	整数	OriginalLength

1. **重要 :** JMS_IBM_MQMD_Priority に 0 から 9 までの範囲に含まれない値を割り当てると、JMS 仕様違反になります。
2. **重要 :** JMS 仕様では、メッセージ ID は JMS プロバイダーによって設定されなければならず、それは固有のものまたはヌルでなければならないと規定されています。 JMS_IBM_MQMD_MsgId に値を割り当てると、この値は JMSMessageID にコピーされます。 このようにして、メッセージ ID が JMS プロバイダーによって設定されず、固有のものとならない場合もあります。これは JMS 仕様に違反します。
3. **重要 :** 'ID:' というストリングから始まる値を JMS_IBM_MQMD_CorrelId に割り当てると、これは JMS 仕様に違反します。
4. **重要 :** メッセージでバイト配列プロパティーを使用すると、JMS 仕様に違反します。

IBM MQ classes for JMS を使用したアプリケーションからの *IBM MQ* メッセージ・データへのアクセス

IBM MQ classes for JMS を使用して、アプリケーション内の完全な IBM MQ メッセージ・データにアクセスできます。 すべてのデータにアクセスするには、メッセージが JMSBytesMessage である必要があります。 JMSBytesMessage の本体には、MQRFH2 ヘッダー、その他の IBM MQ ヘッダー、および以下のメッセージ・データが含まれています。

宛先の WMQ_MESSAGE_BODY プロパティーを WMQ_MESSAGE_BODY_MQ に設定して、JMSBytesMessage 内のすべてのメッセージ本体データを受信します。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS または WMQ_MESSAGE_BODY_UNSPECIFIED に設定されていると、メッセージ本体は JMS MQRFH2 ヘッダーなしで返され、JMSBytesMessage のプロパティーには RFH2 で設定されているプロパティーが反映されます。

一部のアプリケーションは、このトピックで説明している機能を使用できません。 アプリケーションが IBM MQ V6 キュー・マネージャーに接続されている場合、または PROVIDERVERSION が 6 に設定されている場合、これらの機能を使用できません。

メッセージの送信

メッセージの送信の際、宛先プロパティー WMQ_MESSAGE_BODY は WMQ_TARGET_CLIENT よりも優先されます。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS に設定されている場合、 IBM MQ classes for JMS は、JMSMessage プロパティーおよびヘッダー・フィールドの設定に基づいて MQRFH2 ヘッダーを自動的に生成します。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定されている場合、メッセージ本体に追加のヘッダーが加えられることはありません。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_UNSPECIFIED に設定されている場合、WMQ_TARGET_CLIENT が WMQ_TARGET_DEST_MQ に設定されていない限り、 IBM MQ classes for JMS は MQRFH2 ヘッダーを送信します。 受信の際、WMQ_TARGET_CLIENT が WMQ_TARGET_DEST_MQ に設定されていると、あらゆる MQRFH2 がメッセージ本体から削除されます。

注 : JMSBytesMessage および JMSTextMessage では MQRFH2 は不要です。一方、JMSStreamMessage、JMSMapMessage、および JMSObjectMessage では必要となります。

WMQ_MESSAGE_BODY_UNSPECIFIED は WMQ_MESSAGE_BODY のデフォルト設定であり、WMQ_TARGET_DEST_JMS は WMQ_TARGET_CLIENT のデフォルト設定です。

JMSBytesMessage を送信する場合、IBM MQ メッセージの構成時に JMS メッセージ本体のデフォルト設定をオーバーライドできます。 以下のプロパティーを使用します。

- JMS_IBM_Format または JMS_IBM_MQMD_Format: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合、IBM MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのフォーマットを指定します。
- JMS_IBM_Character_Set または JMS_IBM_MQMD_CodedCharSetId: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合に JMS メッセージ本体を開始する IBM MQ ヘッダーまたはアプリケーション・ペイロードの CCSID を指定します。
- JMS_IBM_Encoding または JMS_IBM_MQMD_Encoding: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合、IBM MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのエンコード方式を指定します。

両方のタイプのプロパティーを指定した場合、宛先プロパティー WMQ_MQMD_WRITE_ENABLED が true に設定されている限り、JMS_IBM_MQMD_* プロパティーによって、対応する JMS_IBM_* プロパティーが指定変更されます。

JMS_IBM_MQMD_* と JMS_IBM_* を使用したメッセージ・プロパティーの設定には、実際には大きな相違があります。

1. JMS_IBM_MQMD_* プロパティーは、IBM MQ JMS プロバイダーに固有です。
2. JMS_IBM_MQMD_* プロパティーは、MQMD でのみ設定されます。 JMS_IBM_* プロパティーが MQMD に設定されるのは、メッセージに MQRFH2 JMS ヘッダーがない場合のみです。 それ以外の場合は、JMS RFH2 ヘッダーで設定されます。
3. JMS_IBM_MQMD_* プロパティーは JMSMessage に書き込まれたテキストおよび数値のエンコード方式には影響しません。

   受信側のアプリケーションは、MQMD.Encoding と MQMD.CodedCharSetId の値が、メッセージ本体内の数値およびテキストのエンコード方式と文字セットに対応していると見なす場合があります。

JMS_IBM_MQMD_* プロパティーを使用する場合、送信側アプリケーションでそのように設定する必要があります。 メッセージ本体内の数値およびテキストのエンコード方式と文字セットは、JMS_IBM_* プロパティーで設定します。

254 ページの図 39 の不適切にコード化されたスニペットは、文字セット 1208 でエンコードされ、MQMD.CodedCharSetId が 37 に設定されたメッセージを送信します。

a. 誤ってエンコードされたメッセージの送信

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

b. MQMD.CodedCharSetId の値によって設定された JMS_IBM_CHARACTER_SET の値に依存した、メッセージの受信

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

c. 結果としての出力:

```
Message is "éÈÊ'>...??>?"
```

図 *39.* 一貫性なくコード化された *MQMD* とメッセージ・データ

254 ページの図 40 のコードのいずれかのスニペットは、自動的に生成される MQRFH2 ヘッダーが追加されることなく、メッセージ本体にアプリケーション・ペイロードが含まれている状態で、メッセージがキューまたはトピックに置かれる結果となります。

1. WMQ_MESSAGE_BODY_MQ の設定:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. WMQ_TARGET_DEST_MQ の設定:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

図 *40. MQ* メッセージ本体が含まれたメッセージの送信

メッセージの受信

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS に設定されている場合、インバウンド JMS メッセージ・タイプおよび本体は、受信される WebSphere MQ メッセージの内容によって判別されます。 このメッセージ・タイプおよび本体は、MQRFH2 ヘッダー内のフィールドによって決定されます。または、MQRFH2 がない場合は MQMD 内のフィールドによって決定されます。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定されている場合、インバウンド JMS メッセージ・タイプは JMSBytesMessage です。 JMS メッセージ本体は、基礎となる MQGET API 呼び出しによって戻

されるメッセージ・データです。 メッセージ本体の長さは、MQGET 呼び出しによって戻された長さです。 メッセージ本体のデータの文字セットおよびエンコード方式は、MQMD の CodedCharSetId および Encoding フィールドによって判別されます。 メッセージ本体内のデータのフォーマットは、MQMD の Format フィールドによって判別されます。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_UNSPECIFIED に設定されている場合 (デフォルト値)、IBM MQ classes for JMS がそれを WMQ_MESSAGE_BODY_JMS に設定します。

JMSBytesMessage を受信したならば、以下のプロパティーを参照してデコードすることができます。

- JMS_IBM_Format または JMS_IBM_MQMD_Format: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合、IBM MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのフォーマットを指定します。
- JMS_IBM_Character_Set または JMS_IBM_MQMD_CodedCharSetId: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合に JMS メッセージ本体を開始する IBM MQ ヘッダーまたはアプリケーション・ペイロードの CCSID を指定します。
- JMS_IBM_Encoding または JMS_IBM_MQMD_Encoding: このプロパティーは、先行する WebSphere MQ ヘッダーがない場合、IBM MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのエンコード方式を指定します。

以下のコード・スニペットの結果は、受信されたメッセージ (JMSBytesMessage) となります。 受信されたメッセージ、および受信された MQMD のフォーマット・フィールドの内容に関係なく、メッセージは JMSBytesMessage となります。

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

宛先プロパティー *WMQ_MESSAGE_BODY*
WMQ_MESSAGE_BODY は、JMS アプリケーションが IBM MQ メッセージの MQRFH2 を、メッセージ・ペイロードの一部として (すなわち、JMS メッセージ本体の一部として) 処理するかどうかを決定します。

表 *39.* プロパティーの名前および説明

Property	**短縮形**	**説明**
WMQ_MESSAGE_BODY	MBODY	JMS アプリケーションが IBM MQ メッセージの MQRFH2 を、メッセージ・ペイロードの一部として (すなわち、JMS メッセージ本体の一部として) 処理するかどうか。

表 *40.* プロパティーの名前、値、および設定方式

Property	**管理ツール内での有効値 (太字はデフォルト)**	**プログラム内での有効値**	**Set メソッド**
WMQ_MESSAGE_BODY	• **UNSPECIFIED** 送信の際に、IBM MQ classes for JMS が MQRFH2 ヘッダーを生成して組み込むかどうかは、WMQ_TARGET_CLIENT の値によって異なります。 受信の際には、JMS の値に従って作動します。 • JMS 送信の際に、IBM MQ classes for JMS は自動的に MQRFH2 ヘッダーを生成して IBM MQ メッセージに組み込みます。 受信の際に、IBM MQ classes for JMS は、MQRFH2 (存在する場合) の値に従って JMS メッセージ・プロパティーを設定します。 MQRFH2 を JMS メッセージ本体の一部として表示しません。 • MQ 送信の際に、IBM MQ classes for JMS は MQRFH2 を 生成しません。 受信の際に、IBM MQ classes for JMS は MQRFH2 を JMS メッセージ本体の一部として表示します。	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 持続メッセージ

IBM MQ classes for JMS アプリケーションでは、**NonPersistentMessageClass** キュー属性を使用して JMS 持続メッセージのパフォーマンスを向上させることができます。 ただし、信頼性は多少低下します。

IBM MQ キューには、**NonPersistentMessageClass** という名前の属性があります。 キュー・マネージャーが再始動するときにキュー上の非持続メッセージが破棄されるかどうかは、この属性の値によって決まります。

ローカル・キューに対してこの属性を設定するには、IBM MQ スクリプト・コマンド (MQSC) である DEFINE QLOCAL を、以下のいずれかのパラメーターと共に使用します。

NPMCLASS(NORMAL)
キュー・マネージャーが再始動するとき、キュー上の非持続メッセージは破棄されます。 これがデフォルト値です。

NPMCLASS(HIGH)
キュー・マネージャーが静止シャットダウンまたは即時シャットダウンの後で再始動するとき、キュー上の非持続メッセージは破棄されません。 ただし、優先シャットダウンや障害の後には、非持続メッセージが破棄される可能性があります。

このトピックでは、IBM MQ classes for JMS アプリケーションでこのキュー属性を使用することによって JMS 持続メッセージのパフォーマンスを向上させる方法について説明します。

Queue オブジェクトまたは Topic オブジェクトの PERSISTENCE プロパティーには、HIGH という値を使用できます。 この値は IBM MQ JMS 管理ツールを使用して設定できます。また、アプリケーションで Destination.setPersistence() メソッドを呼び出して、値 WMQConstants.WMQ_PER_NPHIGH をパラメーターとして渡すこともできます。

PERSISTENCE プロパティーの値が HIGH の宛先にアプリケーションから JMS 持続メッセージまたは JMS 非持続メッセージが送信されたときに、基礎 IBM MQ キューが NPMCLASS(HIGH) に設定されていると、送信されたメッセージは、IBM MQ 非持続メッセージとしてそのキューに入れられます。 宛先の PERSISTENCE プロパティーの値が HIGH ではない場合、または基礎キューが NPMCLASS(NORMAL) に設定されている場合には、JMS 持続メッセージは IBM MQ 持続メッセージとしてキューに入れられ、JMS 非持続メッセージは IBM MQ 非持続メッセージとしてキューに入れられます。

JMS 持続メッセージが IBM MQ 非持続メッセージとしてキューに入れられる場合にキュー・マネージャーの静止シャットダウンや即時シャットダウンの後にそのようなメッセージが破棄されないようにするには、メッセージが経路指定されるすべてのキューを NPMCLASS(HIGH) に設定する必要があります。 パブリッシュ/サブスクライブ・ドメインでは、これらのキューにサブスクライバー・キューが含まれます。 この構成を強制するために、PERSISTENCE プロパティーの値が HIGH であり、基礎となる IBM MQ キューが NPMCLASS (NORMAL) に設定されている宛先のメッセージ・コンシューマーをアプリケーションが作成しようとすると、 IBM MQ classes for JMS は InvalidDestination 例外をスローします。

宛先の PERSISTENCE プロパティーを HIGH に設定しても、その宛先でメッセージが受信される方法には影響しません。 JMS 持続メッセージとして送信されたメッセージは JMS 持続メッセージとして受信され、JMS 非持続メッセージとして送信されたメッセージは JMS 非持続メッセージとして受信されます。

アプリケーションで、PERSISTENCE プロパティーの値が HIGH の宛先に最初のメッセージが送信される、または PERSISTENCE プロパティーの値が HIGH の宛先に対して最初のメッセージ・コンシューマーが作成されると、IBM MQ classes for JMS は MQINQ 呼び出しを発行して、基礎となる IBM MQ キュー上で NPMCLASS(HIGH) が設定されているかどうかを判断します。 したがって、アプリケーションは、キューに対して照会を行う権限を持っていなければなりません。 また、IBM MQ classes for JMS は、該当の宛先が削除されるまで MQINQ 呼び出しの結果を保持します。追加の MQINQ 呼び出しを実行することはありません。 したがって、アプリケーションでその宛先がまだ使用されているにもかかわらず基礎キューの NPMCLASS 設定を変更した場合、IBM MQ classes for JMS ではその新しい設定を認識できません。

JMS 持続メッセージを IBM MQ 非持続メッセージとして IBM MQ キューに入れることができるようにすると、パフォーマンスが向上します。ただし、信頼性は多少低下します。 JMS 持続メッセージの信頼性を最大にする必要がある場合は、PERSISTENCE プロパティーの値が HIGH である宛先にはメッセージを送信しないでください。

JMS レイヤーは SYSTEM.JMS.TEMPQ.MODEL(SYSTEM.DEFAULT.MODEL.QUEUE。SYSTEM.DEFAULT.MODEL.QUEUE では持続メッセージを受け入れることができないため、SYSTEM.JMS.TEMPQ.MODEL は持続メッセージを受け入れる永続動的キューを作成します。 一時キューを使用して持続メッセージを受け入れるには、SYSTEM.JMS.TEMPQ.MODEL を使用するか、自分で選択した別のキューにモデル・キューを変更する必要があります。

IBM MQ classes for JMS での *TLS* の使用

IBM MQ classes for JMS アプリケーションで、Transport Layer Security (TLS) 暗号化を使用できます。 これを実行するには、JSSE プロバイダーが必要です。

TRANSPORT(CLIENT) を使用した IBM MQ classes for JMS 接続では、TLS 暗号化がサポートされます。TLS は、通信の暗号化、認証、およびメッセージの整合性を提供します。 これは一般的に、インターネット上やイントラネット内で、任意の 2 つのピアの間の通信を保護するために使用されます。

IBM MQ classes for JMS は Java Secure Socket Extension (JSSE) を使用して TLS 暗号化を処理するため、JSSE プロバイダーが必要になります。 JSE v1.4 JVM には、JSSE プロバイダーが組み込まれています。 証明書を管理して保管する方法は、プロバイダーごとに違います。 詳細については、各 JSSE プロバイダーの資料を参照してください。

この節では、JSSE プロバイダーが正常にインストールおよび構成されていることと、適切な証明書がユーザーの JSSE プロバイダーにインストールされて使用可能であることを前提としています。 つまり、JMSAdmin を使用して いくつかの管理プロパティーを設定する準備が完了しています。

IBM MQ classes for JMS アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合は、285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』を参照してください。

SSLCIPHERSUITE オブジェクト・プロパティー
SSLCIPHERSUITE を設定して、ConnectionFactory オブジェクトに対する TLS 暗号化を使用可能にします。

ConnectionFactory オブジェクトで TLS 暗号化を使用可能にするには、JMSAdmin を使用して、SSLCIPHERSUITE プロパティーを JSSE プロバイダーによってサポートされている CipherSuite に設定します。 これは、宛先チャネルで設定された CipherSpec と一致していなければなりません。 ただし、CipherSuites は CipherSpec とは異なるため、異なる名前が付けられています。 261 ページの『IBM MQ classes for JMS での TLS CipherSpec と CipherSuite』 には、IBM MQ にサポートされている CipherSpec を、JSSE に知られている同等な CipherSpec にマップする表を示します。 IBM MQ での CipherSpec および CipherSpec の詳細については、IBM MQ の保護を参照してください。

例えば、TLS_RSA_WITH_AES_128_CBC_SHA の CipherSpec を使用して TLS 対応の MQI チャネルに接続するよう ConnectionFactory オブジェクトを設定するには、JMSAdmin に対して次のコマンドを発行します。

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

これは、MQConnectionFactory オブジェクトで setSSLCipherSuite() メソッドを使用することにより、プログラムから設定することも可能です。

CipherSpec が SSLCIPHERSUITE プロパティーで指定されている場合、JMSAdmin は便宜上 CipherSpec を適切な CipherSuite に割り当てようとして警告を発行します。 アプリケーションがプロパティーを指定している場合は、このようなマッピングは試みられません。

または、クライアント・チャネル定義テーブル (CCDT) を使用します。 詳細については、285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』を参照してください。

SSLFIPSREQUIRED オブジェクト・プロパティー
IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を使用する接続が必要な場合は、接続ファクトリーの SSLFIPSREQUIRED プロパティーを YES に設定します。

注 : AIX, Linux, and Windows では、 IBM MQ は IBM Crypto for C (ICC) 暗号モジュールを介して FIPS 140-2 準拠を提供します。 このモジュールの証明書は「履歴」ステータスに移動されました。 お客様は、IBM Crypto for C (ICC) 証明書 を表示し、NIST から提供されたアドバイスに注意する必要があります。 交換用の FIPS 140-3 モジュールが現在進行中であり、その状況を表示するには、 「NIST CMVP modules in process list」でそのモジュールを検索します。

このプロパティーのデフォルト値は NO です。つまり、接続で IBM MQ によってサポートされる任意の CipherSuite を使用することができます。

アプリケーションが複数の接続を使用する場合、アプリケーションが最初の接続を作成するときに使用される SSLFIPSREQUIRED の値によって、アプリケーションが以降のすべての接続を作成するときに使用される値が決まります。 これは、以降の接続の作成時に使用される接続ファクトリーの SSLFIPSREQUIRED プロパティーの値が無視されることを意味します。 別の SSLFIPSREQUIRED 値を使用する場合は、アプリケーションを再始動する必要があります。

アプリケーションでは、ConnectionFactory オブジェクトの setSSLFipsRequired() メソッドを呼び出すことで、このプロパティーを設定できます。 CipherSuite が設定されていない場合、このプロパティーは無視されます。

関連タスク
MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する
関連資料
AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

SSLPEERNAME オブジェクト・プロパティー
SSLPEERNAME を使用して識別名パターンを指定し、JMS アプリケーションが確実に正しいキュー・マネージャーに接続するようにします。

JMS アプリケーションでは、識別名 (DN) パターンを指定することにより、正しいキュー・マネージャーに接続していることが確認できます。 キュー・マネージャーがパターンと一致する DN を提示する場合にのみ、接続は成功します。 このパターンの形式の詳細については、関連トピックを参照してください。

DN は、ConnectionFactory オブジェクトの SSLPEERNAME プロパティーを使用して設定されます。 例えば、次の JMSAdmin コマンドは、ConnectionFactory オブジェクトが QMGR. で始まる共通名、および少なくとも 2 つの組織単位名、最初は IBM、次は WEBSPHERE を使用して、キュー・マネージャーを識別するように設定します。

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

検査は大文字と小文字を区別しないで行われます。またコンマの代わりにセミコロンを使用できます。 SSLPEERNAME は、MQConnectionFactory オブジェクトで setSSLPeerName() メソッドを使用することにより、アプリケーションから設定することも可能です。 このプロパティーが設定されない場合、キュー・マネージャーで提供される識別名 (DN) に対して検査は実行されません。 CipherSuite が設定されなければ、このプロパティーは無視されます。

SSLCERTSTORES オブジェクト・プロパティー
SSLCERTSTORES を使用して、証明書取り消しリスト (CRL) の検査に使用する LDAP サーバーのリストを指定します。

非トラステッドになった証明書を識別するには、証明書取り消しリスト (CRL) が一般的に使用されます。 多くの場合 CRL は、LDAP サーバーにホストされています。 JMS を使用すると、Java 2 v1.4 以降での CRL 検査に、LDAP サーバーを指定できるようになります。 次の JMSAdmin の例では、crl1.ibm.com という LDAP サーバーにホストされている CRL を使用するよう JMS に指示しています。

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

注 : CertStore を LDAP サーバーでホストされている CRL と共に正常に使用するには、ご使用の Java Software Development Kit (SDK) が CRL に適合することを確認してください。 SDK によっては、CRL が LDAP v2 用のスキーマを定義する RFC 2587 に準拠している必要があります。 ほとんどの LDAP v3 サーバーは、代わりに RFC 2256 を使用しています。

使用している LDAP サーバーがデフォルト・ポートの 389 で稼働していない場合、ホスト名にコロン (:) とポート番号を追加して、そのポートを指定できます。 キュー・マネージャーによって提示される証明書が、crl1.ibm.com にホストされている CRL に提示される場合、接続は完了しません。 Single Point of Failure を避けるため、JMS では、スペース文字で区切った LDAP サーバーのリストを提供することにより、複数の LDAP サーバーを指定できます。 以下が例となります。

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

複数の LDAP サーバーを指定すると、JMS は、キュー・マネージャーの証明書を正常に検査できるサーバーを見つけるまで、各サーバーを順番に試行します。 各サーバーには、同一の情報が含まれている必要があります。

この形式の文字列は、MQConnectionFactory.setSSLCertStores() メソッドで、アプリケーションにより提供することができます。 別の方法として、アプリケーションで 1 つ以上の java.security.cert.CertStore オブジェクトを作成し、適切な Collection オブジェクトに配置し、この Collection オブジェクトを setSSLCertStores() メソッドに提供することができます。 この方法で、アプリケーションは CRL 検査をカスタマイズすることができます。 CertStore オブジェクトの組み立て方法と使用方法の詳細については、ご使用の JSSE の資料を参照してください。

接続を設定するときにキュー・マネージャーによって示される証明書は、以下のようにして検査されます。

1. sslCertStores によって識別される Collection にある最初の CertStore オブジェクトを使用し、CRL サーバーを識別します。
2. CRL サーバーへ接続してみます。
3. 接続が成功すれば、一致する証明書をサーバー内で検索します。

a. 証明書が失効したことが分かった場合、検索プロセスは終了して接続要求は失敗します。理由コードは MQRC_SSL_CERTIFICATE_REVOKED です。

b. 証明書が見つからない場合、検索プロセスは終了し、接続を先に進めることができます。

4. サーバーへの接続が失敗する場合、次の CertStore オブジェクトを使用して、CRL サーバーを識別します。プロセスはステップ 2 から繰り返されます。

   これが Collection の最後の CertStore であるか、Collection に CertStore オブジェクトが含まれない場合、検索プロセスは失敗し、接続要求は失敗します。理由コードは MQRC_SSL_CERT_STORE_ERROR です。

Collection オブジェクトは、CertStores を使用する順序を決定します。

アプリケーションが setSSLCertStores() を使用して CertStore オブジェクトの Collection を設定する場合、MQConnectionFactory を JNDI 名前空間にバインドできなくなります。 バインドしようとすると、例外が発生します。 sslCertStores プロパティーが設定されない場合、キュー・マネージャーで提供される証明書に対する失効の検査は実行されません。 CipherSuite が設定されなければ、このプロパティーは無視されます。

SSLRESETCOUNT オブジェクト・プロパティー
このプロパティーは、暗号化に使用された秘密鍵が再ネゴシエーションされる前に、接続で送信および受信したバイトの総数を表します。

送信バイト数は暗号化前の数であり、受信バイト数は暗号化解除された後の数です。 バイト数には、IBM MQ classes for JMS によって送受信される制御情報も含まれています。

例えば、TLS 対応の MQI チャネル (このチャネルの秘密鍵は、4 MB のデータが流れた後再ネゴシエーションされる) を介した接続の作成に使用できる ConnectionFactory オブジェクトを構成するには、JMSAdmin に対して次のコマンドを発行します。

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

アプリケーションでは、ConnectionFactory オブジェクトの setSSLResetCount() メソッドを呼び出すことで、このプロパティーを設定できます。

このプロパティーの値がゼロ (デフォルト値) の場合、秘密鍵の再ネゴシエーションは行われません。 CipherSuite が設定されていない場合、このプロパティーは無視されます。

SSLSocketFactory オブジェクト・プロパティー
アプリケーション用に TLS 接続の他の面をカスタマイズするには、SSLSocketFactory を作成し、JMS がそれを使用するように構成します。

特定アプリケーションのために、TLS 接続の他の面をカスタマイズできます。 例えば、暗号ハードウェアを初期設定したり、使用中の鍵ストアおよびトラストストアを変更したりしたい場合があります。 そのためには、アプリケーションで、それに従ってカスタマイズされた javax.net.ssl.SSLSocketFactory オブジェクトをまず作成する必要があります。 カスタマイズできる機能はプロバイダーによって異なるため、インスタンスを作成する方法についてはご使用の JSSE の資料を参照してください。 適切な SSLSocketFactory オブジェクトを入手したら、MQConnectionFactory.setSSLSocketFactory() メソッドを使用して、カスタマイズした SSLSocketFactory オブジェクトを使用するよう JMS を構成します。

アプリケーションが setSSLSocketFactory() メソッドを使用してカスタマイズした SSLSocketFactory オブジェクトを設定する場合、MQConnectionFactory オブジェクトは JNDI ネーム・スペースにバインドできなくなります。 バインドしようとすると、例外が発生します。 このプロパティーが設定されていなければ、デフォルトの SSLSocketFactory オブジェクトが使用されます。 デフォルトの SSLSocketFactory オブジェクトの振る舞いの詳細については、ご使用の JSSE の資料を参照してください。 CipherSuite が設定されなければ、このプロパティーは無視されます。

重要 : それ自体が保護されていない JNDI ネーム・スペースから ConnectionFactory オブジェクトを取り出す場合、SSL プロパティーを使用してもセキュリティーが保証されるとは考えないでください。 特に、JNDI を標準的に LDAP 実装しても保護にはなりません。 アタッカーは LDAP サーバーをだませるため、JMS アプリケーションはだまされたと気付かずに間違ったサーバーに接続する可能性があります。 適切なセキュリティー配置がなされていれば、JNDI の他の実装 (fscontext 実装など) は保護されます。

JSSE 鍵ストアまたは *JSSE* トラストストアの変更

鍵ストアまたはトラストストアに変更を加える場合、選出する変更に対して特定のアクションを取る必要があります。

JSSE 鍵ストアや JSSE トラストストアの内容を変更したり、鍵ストア・ファイルやトラストストア・ファイルの位置を変更したりした場合、その時点で実行中の IBM MQ classes for JMS アプリケーションはその変更を自動的には認識しません。 変更を有効にするには、以下のアクションを行う必要があります。

- アプリケーションは、すべての接続をクローズし、接続プール内の未使用の接続を破棄する必要がある。
- JSSE プロバイダーが鍵ストアおよびトラストストアからの情報をキャッシュしている場合は、この情報をリフレッシュする必要がある。

これらのアクションが完了すると、アプリケーションは接続を再作成できます。

アプリケーションの設計方法や、JSSE プロバイダーが提供する機能によっては、アプリケーションを停止および再始動しなくても上記のアクションを実行できる場合があります。 ただし、アプリケーションを停止して再始動するのが最も簡単な解決方法です。

IBM MQ classes for JMS での *TLS CipherSpec* と *CipherSuite*

IBM MQ classes for JMS アプリケーションがキュー・マネージャーへの接続を確立できるかどうかは、MQI チャネルのサーバー側で指定された CipherSpec およびクライアント側で指定された CipherSuite によって決まります。

以下の表に、IBM MQ によってサポートされる CipherSpec と、それらに相当する CipherSuite のリストを示します。

Deprecated トピック『推奨されない CipherSpec』を参照して、次の表にリストしている CipherSpec が IBM MQ で非推奨になっていないか、また、非推奨になっている場合はどの更新プログラムで CipherSpec が非推奨になったかを確認してください。

重要 : リストされている CipherSuites は、IBM MQ で提供される IBM Java ランタイム環境 (JRE) によってサポートされるものです。 リストされている CipherSuites には、 Oracle Java JRE によってサポートされている CipherSuites が含まれています。 Oracle Java JRE を使用するようにアプリケーションを構成する方法について詳しくは、「IBM Java または Oracle Java CipherSuite マッピングを使用するためのアプリケーションの構成」を参照してください。

この表に、通信に使用されるプロトコルと、CipherSuite が FIPS 140-2 標準に準拠するかどうかも示します。

注 : AIX, Linux, and Windows では、 IBM MQ は IBM Crypto for C (ICC) 暗号モジュールを介して FIPS 140-2 準拠を提供します。 このモジュールの証明書は「履歴」ステータスに移動されました。 お客様は、IBM Crypto for C (ICC) 証明書 を表示し、NIST から提供されたアドバイスに注意する必要があります。 交換用の FIPS 140-3 モジュールが現在進行中であり、その状況を表示するには、 「NIST CMVP modules in process list」でそのモジュールを検索します。

FIPS 140-2 準拠を強制するようにアプリケーションが構成されていない場合に FIPS 140-2 準拠として示されている CipherSuite を使用することはできますが、FIPS 140-2 準拠となるようにアプリケーションが構成されている場合 (下記の構成注記を参照)、FIPS 140-2 互換のマークが付いている CipherSuite しか構成できません。その他の CipherSuite を使用しようとするとエラーになります。

注 : 各 JRE は複数の暗号化セキュリティー・プロバイダーを持つことができ、それぞれが同じ CipherSuite の実装を提供することができます。 ただし、すべてのセキュリティー・プロバイダーが FIPS 140-2 認定されている訳ではありません。 アプリケーションで FIPS 140-2 準拠が強制されていない場合には、非認定の CipherSuite の実装環境を使用できる場合があります。 非認定の実装環境では、FIPS 140-2 で求められる最小セキュリティー・レベルを CipherSuite が理論的には満たしている場合であっても、その標準に準拠した方法で作動しない場合があります。IBM MQ JMS アプリケーションにおいて FIPS 140-2 を強制する場合の構成について詳しくは、以下の注記を参照してください。

CipherSpec と CipherSuite の FIPS 140-2 準拠と Suite-B 準拠について詳しくは、CipherSpec の指定を参照してください。 また、米国の連邦情報処理標準に関わる情報に留意しなければならない場合もあります。

CipherSuite の全セットを使用し、FIPS 140-2 準拠または Suite-B 準拠 (あるいはその両方に準拠) として認定されたものを使用してそれを作動させるには、適切な JRE が必要です。IBM Java 7 Service Refresh 4 フ

ィックスパック 2 以上のレベルの IBM JRE は、 262 ページの表 41 にリストされている TLS 1.2 CipherSuites の適切なサポートを提供します。

TLS 1.3 Ciphers を使用できるようにするには、アプリケーションを実行する JRE が TLS 1.3 をサポートしている必要があります。

注 : 一部の CipherSuite を使用する場合に、「無制限」ポリシー・ファイルを JRE で構成する必要があります。 SDK または JRE でポリシー・ファイルをセットアップする方法について詳しくは、ご使用のバージョンの「 *Security Reference for IBM SDK, Java Technology Edition* 」の「 *IBM SDK* ポリシー・ファイル」トピックを参照してください。

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	いいえ

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	いいえ
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	いいえ

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	いいえ
TLS_RSA_WITH_3DES_EDE_CBC_SHA 280 ページの『2』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	いいえ 280 ページの『4』

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	いいえ 280 ページの『4』
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	いいえ 280 ページの『4』

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	いいえ 280 ページの『4』
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	いいえ 280 ページの『4』

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	いいえ 280 ページの『4』
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	いいえ 280 ページの『4』

表 *41. IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite (*続き*)*

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	いいえ
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	いいえ
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	いいえ

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes
TLS_AES_128_GCM_SHA256 280 ページの『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	いいえ
TLS_AES_256_GCM_SHA384 280 ページの『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	いいえ

表 41. IBM MQ によってサポートされる CipherSpec およびそれらと同等の CipherSuite (続き)

CipherSpec 280 ページの『1』	同等の CipherSuite (IBM JRE)	同等の CipherSuite (Oracle JRE)	プロトコル	FIPS 140-2 互換
TLS_CHACHA20_POLY1305_SHA256 280 ページの『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	いいえ
TLS_AES_128_CCM_SHA256 280 ページの『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	いいえ

表 41. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 280 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_AES_128_CCM_8_SHA256 280 ページの『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	いいえ
任意 280 ページの『3』	*ANY	*ANY	複数	いいえ
ANY_TLS13 280 ページの『3』	*TLS13	*TLS13	TLS V13	いいえ
ANY_TLS12_OR_HIGHER 280 ページの『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 以上	いいえ
ANY_TLS13_OR_HIGHER 280 ページの『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 以上	いいえ

注:

1. これは、CCDT (バイナリーまたは JSON) を含め、 IBM MQ のチャネルで構成された値です。
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。
3. TLS v1.3 暗号を使用できるようにするには、アプリケーションを実行する Java runtime environment (JRE) が TLS v1.3 をサポートしている必要があります。
4. V 9.3.0.17 V 9.3.5.1 IBM MQ 9.3.5 CSU 1 および IBM MQ 9.3.0 CSU 17 以降、 IBM Java 8 JRE は、FIPS モードでの操作時に RSA 鍵交換のサポートを除去します。

IBM MQ classes for JMS アプリケーションでの CipherSuite と FIPS 準拠の構成

- IBM MQ classes for JMS を使用するアプリケーションは、接続に CipherSuite を設定するときに次の 2 つの方式のいずれかを使用できます。
  - ConnectionFactory オブジェクトの setSSLCipherSuite メソッドを呼び出す。
  - IBM MQ JMS 管理ツールを使用して、ConnectionFactory オブジェクトの SSLCIPHERSUITE プロパティーを設定する。
- IBM MQ classes for JMS を使用するアプリケーションは、FIPS 140-2 準拠を強制するために、次の 2 つの方式のいずれかを使用できます。
  - ConnectionFactory オブジェクトの setSSLFipsRequired メソッドを呼び出す。
  - IBM MQ JMS 管理ツールを使用して、ConnectionFactory オブジェクトの SSLFIPSREQUIRED プロパティーを設定する。

IBM Java または Oracle Java CipherSuite マッピングを使用するためのアプリケーションの構成

注 : V 9.3.3 IBM MQ 9.3.3 以降の Continuous Delivery では、どのマッピングを使用するかを制御する Java「システム・プロパティー」 `com.ibm.mq.cfg.useIBMCipherMappings` が製品から削除されています。 IBM MQ 9.3.3 以降、Cipher は CipherSpec または CipherSuite のいずれかの名前として定義でき、IBM MQ によって正しく処理されます。 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging アプリケーションがキュー・マネージャーへのセキュア TLS 接続を作成する場合は、以下の 3 つの Jackson JAR ファイルが必要です。

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

重要 : `com.ibm.mq.cfg.useIBMCipherMappings` に関する以下の情報は、 IBM MQ 9.3.3 より前の Long Term Support および Continuous Delivery にのみ適用されます。

アプリケーションがデフォルトの IBM Java CipherSuite を IBM MQ CipherSpec マッピングに使用するか、Oracle CipherSuite を IBM MQ CipherSpec マッピングに使用するかを構成できます。 そのため、IBM JRE と Oracle JRE のどちらをアプリケーションで使用するかに関係なく、TLS CipherSuites を使用できます。 Java「システム・プロパティー」 `com.ibm.mq.cfg.useIBMCipherMappings` は、どのマッピングを使用するかを制御します。 プロパティーは、次の値のうちのいずれかです。

true
IBM Java CipherSuite を IBM MQ CipherSpec マッピングに使用します。

この値がデフォルト値です。

false
Oracle CipherSuite を IBM MQ CipherSpec マッピングに使用します。

IBM MQ Java および TLS 暗号の使用について詳しくは、MQdev ブログ投稿 MQ Java, TLS Ciphers, Non-IBM JRE & APARs IT06775, IV66840, IT09423, IT10837 を参照してください。

インターオペラビリティーの制約

いくつかの CipherSuite は、使用するプロトコルによっては、複数の IBM MQ CipherSpec と適合します。 ただし、サポートされるのは表 1 で指定された TLS バージョンを使用する CipherSuite/CipherSpec の組み合わせのみです。 サポートされない CipherSuite と CipherSpec の組み合わせを使用しようとすると失敗し、該当する例外が発生します。 これらの CipherSuite/CipherSpec のいずれかの組み合わせを使用するインストールをサポートされる組み合わせに移行する必要があります。

以下の表に、この制約が適用される CipherSuite を示します。

表 *42. CipherSuite* と、それでサポートされる *CipherSpec* とサポートされない *CipherSpec*		
CipherSuite	**サポートされる TLS CipherSpec**	**サポートされない SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 281 ページの『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

注 :

1. Deprecated この CipherSpec の TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

Java for IBM MQ classes for JMS でのチャネル出口の作成
指定されたインターフェースを実装する Java クラスを定義することにより、チャネル出口を作成します。

セキュリティー出口の概要については、「チャネル・セキュリティー出口プログラム」トピックを参照してください。

com.ibm.mq.exits パッケージには、次の 3 つのインターフェースが定義されています。

- WMQSendExit (送信出口用)
- WMQReceiveExit (受信出口用)
- WMQSecurityExit (セキュリティー出口用)

以下のサンプル・コードは、3 つのインターフェースすべてを実装するクラスを定義しています。

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
                                    MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
```

```
  {
    // Complete the body of the security exit here
  }
}
```

各出口は、MQCXP オブジェクトおよび MQCD オブジェクトをパラメーターとして受け取ります。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

送信出口が呼び出される場合、agentBuffer パラメーターには、サーバー・キュー・マネージャーに送信される直前のデータが入ります。 データの長さは式 agentBuffer.limit() で指定されるため、長さパラメーターは不要です。 送信出口は、サーバー・キュー・マネージャーに送信するデータを値として返します。 しかし、この送信出口が一連の送信出口の最後の送信出口でない場合は、返されたデータは一連の送信出口の次のものに渡されます。 送信出口は、agentBuffer パラメーターに受け取るデータを変更して返すこともできますし、変更せずに返すこともできます。 それで、考えられる最も単純な出口の本体は次のとおりです。

```
{ return agentBuffer; }
```

受信出口が呼び出される場合、agentBuffer パラメーターには、サーバー・キュー・マネージャーから受け取ったデータが入ります。 受信出口は、IBM MQ classes for JMS がアプリケーションに渡すデータを値として返します。 しかし、この受信出口が一連の受信出口の最後の受信出口でない場合は、返されたデータは一連の受信出口の次のものに渡されます。

セキュリティー出口が呼び出される場合、agentBuffer パラメーターには、接続のサーバー側のセキュリティー出口からのセキュリティー・フローで受け取ったデータが入ります。 セキュリティー出口は、サーバーのセキュリティー出口へのセキュリティー・フローで送信するデータを値として返します。

チャネル出口は、バッキング配列を持つバッファーとともに呼び出されます。 最良のパフォーマンスを得るためには、出口はバッキング配列を持つバッファーを戻す必要があります。

チャネル出口を呼び出すときには、32 文字までのユーザー・データを渡すことができます。 出口はこのユーザー・データに MQCXP オブジェクトの getExitData() メソッドを呼び出すことによってアクセスします。 出口は setExitData() メソッドを呼び出してユーザー・データを変更できますが、ユーザー・データは出口が呼び出されるごとにリフレッシュされます。 したがって、ユーザー・データに加えられた変更はすべて失われます。 しかし、出口は MQCXP オブジェクトの出口ユーザー域を使用して、ある呼び出しから次の呼び出しにデータを渡すことができます。 出口は getExitUserArea() メソッドを呼び出して、出口ユーザー域に参照でアクセスします。

すべての出口クラスには、コンストラクターがなければなりません。 コンストラクターは、前の例に示したようにデフォルト・コンストラクターにすることも、ストリング・パラメーターのあるコンストラクターにすることもできます。 コンストラクターは呼び出されて、クラス内に定義されている出口ごとに出口クラスのインスタンスを作成します。 したがって、前出の例の場合、送信出口のために MyMQExits クラスの 1 つのインスタンスが作成され、受信出口のためにもう 1 つのインスタンスが作成され、セキュリティー出口のために 3 番目のインスタンスが作成されます。 ストリング・パラメーターのあるコンストラクターが呼び出される場合、インスタンスが作成されるチャネル出口に渡されるのと同じユーザー・データがそのパラメーターに入ります。 出口クラスにデフォルト・コンストラクターと単一パラメーターを持つコンストラクターの両方がある場合、単一パラメーターを持つコンストラクターが優先されます。

接続はチャネル出口内から閉じないでください。

接続のサーバー側にデータが送信される際には、TLS 暗号化はチャネル出口が呼び出された後で 実行されます。 同様に、接続のサーバー側からデータを受信する際には、TLS 暗号化解除はチャネル出口が呼び出される前に 実行されます。

IBM MQ classes for JMS の IBM WebSphere MQ 7.0 より前のバージョンでは、チャネル出口はインターフェース MQSendExit、MQReceiveExit、および MQSecurityExit を使用して実装していました。 これらのインターフェースは今でも使用できますが、機能とパフォーマンスが向上しているので、新しいインターフェースの方が望ましいです。

チャネル出口を使用するように *IBM MQ classes for JMS* を構成する

IBM MQ classes for JMS のアプリケーションは、キュー・マネージャーに接続したときに開始する MQI チャネル上のチャネル・セキュリティー出口、送信出口、受信出口を使用することができます。 アプリケー

ションは、Java、C、または C++ で作成された出口を使用することができます。また、アプリケーションは、連続して実行される一連の送信出口または受信出口を使用することもできます。

以下のプロパティーを使用して、JMS 接続で使用する 1 つの送信出口か一連の送信出口を指定します。

- MQConnectionFactory オブジェクトの **SENDEXIT** プロパティー。
- IBM MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **sendexit** プロパティー。
- 出力通信用に IBM MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **sendexit** プロパティー。

このプロパティーの値は、コンマで区切った 1 つ以上の項目からなるストリングです。 各項目には、以下の方法のいずれかで 1 つの送信出口を指定します。

- Java で作成された送信出口の `WMQSendExit` インターフェースを実装するクラスの名前。
- C または C++ で作成された送信出口用の *libraryName*(*entryPointName*) という形式のストリング。

同様に、以下のプロパティーでは、接続で使用する 1 つの受信出口か一連の受信出口を指定します。

- MQConnectionFactory オブジェクトの **RECEXIT** プロパティー。
- IBM MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **receiveexit** プロパティー。
- 出力通信用に IBM MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **receiveexit** プロパティー。

以下のプロパティーでは、接続で使用するセキュリティー出口を指定します。

- MQConnectionFactory オブジェクトの **SECEXIT** プロパティー。
- IBM MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **securityexit** プロパティー。
- 出力通信用に IBM MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **securityexit** プロパティー。

MQConnectionFactories の場合、**SENDEXIT**、**RECEXIT**、および **SECEXIT** プロパティーは、IBM MQ JMS 管理ツールまたは IBM MQ Explorer を使うことによって設定できます。 あるいは、アプリケーションから、`setSendExit()` メソッド、`setReceiveExit()` メソッド、および `setSecurityExit()` メソッドを呼び出すことによって、これらのプロパティーを設定できます。

チャネル出口はその独自のクラス・ローダーによってロードされます。 チャネル出口を見つけるために、クラス・ローダーは指定された順序で下記の場所を検索します。

1. プロパティー **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** によって指定されたクラスパス、または IBM MQ クライアント構成ファイルの Channels スタンザの **JavaExitsClassPath** 属性によって指定されたクラスパス。
2. Deprecated Java システム・プロパティー **com.ibm.mq.exitClasspath** で指定されたクラス・パス。 このプロパティーは現在、推奨されていません。
3. IBM MQ 出口ディレクトリー (283 ページの表 43 を参照)。 最初に、クラス・ローダーは、Java アーカイブ (JAR) ファイルにパッケージされていないクラス・ファイルを求めてディレクトリーを検索します。チャネル出口が見つからなかった場合、クラス・ローダーは次にディレクトリー内の JAR ファイルを検索します。

表 *43. IBM MQ* 出口ディレクトリー

プラットフォーム	ディレクトリー
Linux AIX AIX and Linux	`/var/mqm/exits` (32 ビット・チャネル出口) `/var/mqm/exits64` (64 ビット・チャネル出口)

表 43. IBM MQ 出口ディレクトリー (続き)	
プラットフォーム	**ディレクトリー**
Windows	*install_data_dir*¥exits *install_data_dir* は、インストール中に IBM MQ データ・ファイル用に選択したディレクトリーです。 デフォルト・ディレクトリーは C: ¥ProgramData¥IBM¥MQ です。

注 : チャネル出口が複数の場所に存在する場合、IBM MQ classes for JMS は最初に検出したインスタンスをロードします。

クラス・ローダーの親は、IBM MQ classes for JMS のロードで使用されるクラス・ローダーです。 そのため、親クラス・ローダーは、チャネル出口が上記のいずれの場所でも見つからなかった場合にそれをロードすることができます。 ただし、 JEE アプリケーション・サーバーなどの環境で IBM MQ classes for JMS を使用する場合は、親クラス・ローダーの選択に影響を与える可能性が低いため、アプリケーション・サーバーで Java システム・プロパティー **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** を設定してクラス・ローダーを構成する必要があります。

Java security manager を有効化してアプリケーションを実行している場合は、アプリケーションが実行されている Java ランタイム環境によって使用されるポリシー構成ファイルに、チャネル出口クラスをロードするための許可が必要です。 これを行う方法については、Java Security Manager での IBM MQ classes for JMS アプリケーションの実行を参照してください。

IBM WebSphere MQ 7.0 より前のバージョンで提供されている `MQSendExit`、 `MQReceiveExit`、および `MQSecurityExit` インターフェースは引き続きサポートされます。 これらのインターフェースを実装するチャネル出口を使用する場合は、`com.ibm.mq.jar` がクラスパスに存在している必要があります。

C でチャネル出口を作成する方法については、967 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。 C または C++ で書かれたチャネル出口プログラムは、283 ページの表 43 に示されるディレクトリーに保管しなければなりません。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合は、285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』を参照してください。

IBM MQ classes for JMS を使用する際のチャネル出口に渡すユーザー・データの指定

チャネル出口を呼び出すときには、32 文字までのユーザー・データを渡すことができます。

MQConnectionFactory オブジェクトの SENDEXITINIT プロパティーでは、各送信出口が呼び出されたときにその出口に渡されるユーザー・データを指定します。 このプロパティーの値は、コンマで区切ったユーザー・データの 1 つ以上の項目からなるストリングです。 ストリング内のユーザー・データの各項目の位置は、そのユーザー・データが一連の送信出口の内のどの送信出口に渡されるかを決定します。 例えば、ストリング内のユーザー・データの最初の項目は、一連の送信出口の内の最初の送信出口に渡されます。

IBM MQ JMS 管理ツールまたは IBM MQ Explorer を使用して、SENDEXITINIT プロパティーを設定することができます。 あるいは、アプリケーションから、setSendExitInit() メソッドを呼び出すことによって、プロパティーを設定できます。

同様に、ConnectionFactory オブジェクトの RECEXITINIT プロパティーでは各受信出口に渡されるユーザー・データを指定し、SECEXITINIT プロパティーではセキュリティー出口に渡されるユーザー・データを指定します。 これらのプロパティーは、IBM MQ JMS 管理ツールまたは IBM MQ Explorer を使用して設定できます。 あるいは、アプリケーションから、setReceiveExitInit() メソッドと setSecurityExitInit() メソッドを呼び出すことによって、これらのプロパティーを設定できます。

チャネル出口に渡されるユーザー・データを指定する際には、以下の規則に注意してください。

- ストリング内のユーザー・データの項目数が一連の出口の数より多い場合、ユーザー・データの余分の項目は無視されます。
- ストリング内のユーザー・データの項目数が一連の出口の数より少ない場合、指定されていないユーザー・データの項目はそれぞれ空ストリングに設定されます。 ストリング内の 2 つの連続したコンマやストリングの始まりのコンマも、指定されていないユーザー・データの項目を表します。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーと接続する場合、チャネル出口が呼び出されるときに、クライアント接続チャネル定義で指定されたユーザー・データがチャネル出口に渡されます。 クライアント・チャネル定義テーブルの使用については、285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』を参照してください。

IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用

IBM MQ classes for JMS アプリケーションでは、クライアント・チャネル定義テーブル (CCDT) に格納されたクライアント接続チャネル定義を使用できます。 CCDT を使用するには、ConnectionFactory オブジェクトを構成します。 このコマンドの使用には、いくつかの制限があります。

IBM MQ classes for JMS アプリケーションでは、ConnectionFactory オブジェクトの特定のプロパティーを設定してクライアント接続チャネル定義を作成する代わりに、クライアント・チャネル定義テーブルに保管されているクライアント接続チャネル定義を使用することができます。 このような定義の作成には、IBM MQ スクリプト・コマンド (MQSC) または IBM MQ プログラマブル・コマンド・フォーマット (PCF) のコマンドを使用します。 アプリケーションで Connection オブジェクトが作成されると、IBM MQ classes for JMS は、クライアント・チャネル定義テーブルを検索して適切なクライアント接続チャネル定義を見つけ、そのチャネル定義を使用して MQI チャネルを開始します。 クライアント・チャネル定義テーブルの詳細とその構成方法については、クライアント・チャネル定義テーブルを参照してください。

クライアント・チャネル定義テーブルを使用するには、ConnectionFactory オブジェクトの CCDTURL プロパティーに URL オブジェクトを設定する必要があります。 IBM MQ classes for JMS では、IBM MQ MQI client 構成ファイルから CCDT に関する情報は読み取りませんが、そのファイルにある他の一部の値は使用されます (これに該当する値については、99 ページの『IBM MQ classes for JMS/Jakarta Messaging 構成ファイル』を参照してください)。 URL オブジェクトは、クライアント・チャネル定義テーブルを格納するファイルの名前と場所を識別する URL (Uniform Resource Locator) をカプセル化し、そのファイルへのアクセス方法を指定します。 CCDTURL プロパティーは、IBM MQ JMS 管理ツールを使用して設定できます。また、アプリケーションで、URL オブジェクトを作成してから ConnectionFactory オブジェクトの setCCDTURL() メソッドを呼び出すことによってこのプロパティーを設定できます。

例えば、ファイル ccdt1.tab にクライアント・チャネル定義テーブルが含まれており、アプリケーションが実行されるシステムと同じシステムにこのファイルが保管されている場合、アプリケーションで CCDTURL プロパティーを設定するには、以下のようにします。

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

もう 1 つの例として、ファイル ccdt2.tab にクライアント・チャネル定義テーブルが入っており、 アプリケーションの実行システムとは異なるシステムにこのファイルが格納されている場合を想定します。 FTP プロトコルを使用してこのファイルにアクセスできる場合、アプリケーションでは以下の方法で CCDTURL プロパティーを設定できます。

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

ConnectionFactory オブジェクトの CCDTURL プロパティーを設定する必要があるほかに、同じオブジェクトの QMANAGER プロパティーに以下のいずれかの値を設定する必要もあります。

- キュー・マネージャーの名前。
- アスタリスク (*) とそれに続くキュー・マネージャー・グループ名。

これらの値は、メッセージ・キュー・インターフェース (MQI) を使用しているクライアント・アプリケーションによって発行された MQCONN 呼び出しで、**QMgrName** パラメーターに使用できる値と同じです。 したがって、これらの値の意味の詳細については、MQCONN を参照してください。 QMANAGER プロパティーは、IBM MQ JMS 管理ツールまたは IBM MQ エクスプローラーを使うことによって設定できます。 あるいは、アプリケーションから、ConnectionFactory オブジェクトの setQueueManager() メソッドを呼び出すことによって、プロパティーを設定できます。

その後、アプリケーションで ConnectionFactory オブジェクトから Connection オブジェクトが作成されると、IBM MQ classes for JMS は、CCDTURL プロパティーが示すクライアント・チャネル定義テーブルにアクセスします。次に、QMANAGER プロパティーを使用してこのテーブルを検索し、適切なクライアント接

続チャネル定義を見つけ、そのチャネル定義を使用してキュー・マネージャーへの MQI チャネルを開始します。

アプリケーションが createConnection() メソッドを呼び出すときに、ConnectionFactory オブジェクトの CCDTURL プロパティーと CHANNEL プロパティーの両方を設定することはできないことに注意してください。 両方のプロパティーが設定されると、このメソッドは例外をスローします。 CCDTURL プロパティーまたは CHANNEL プロパティーは、その値がヌル、空ストリング、ブランク文字のみのストリングのいずれでもない場合に設定されていると見なされます。

IBM MQ classes for JMS は、クライアント・チャネル定義テーブル内に適切なクライアント接続チャネル定義を見つけると、そのテーブルから取り出した情報のみを使用して MQI チャネルを開始します。 ConnectionFactory オブジェクトのチャネル関連プロパティーはすべて無視されます。

特に、TLS を使用する場合は、以下の点に注意してください。

- MQI チャネルで TLS が使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義に、IBM MQ classes for JMS でサポートされる CipherSpec の名前が指定されている場合に限られます。
- クライアント・チャネル定義テーブルには、証明書取り消しリスト (CRL) を保持する LDAP (Lightweight Directory Access Protocol) サーバーの場所に関する情報も含まれます。 IBM MQ classes for JMS は、この情報のみを使用して、CRL を保持する LDAP サーバーにアクセスします。
- クライアント・チャネル定義テーブルには、OCSP 応答側の場所を含めることもできます。 IBM MQ classes for JMS では、クライアント・チャネル定義テーブル・ファイルの OCSP 情報を使用できません。 ただし、OCSP を構成することはできます (Java および JMS クライアント・アプリケーションでの Online Certificate Status Protocol (OCSP) セクションを参照)。

クライアント・チャネル定義テーブルでの TLS の使用の詳細については、TLS チャネルを持つ拡張トランザクション・クライアントの使用を参照してください。

チャネル出口を使用する場合は、以下の点にも注意してください。

- MQI チャネルで使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義によって指定されているチャネル出口と関連ユーザー・データに限られます。
- クライアント・チャネル定義テーブルから抽出されたチャネル定義には、Java で作成されたチャネル出口を指定できます。 つまり、例えば クライアント接続チャネル定義を作成する DEFINE CHANNEL 上の SCYEXIT パラメーターは、WMQSecurityExit インターフェースを実装するクラスの名前を指定できます。 同様に、SENDEXIT パラメーターには、WMQSendExit インターフェースを実装するクラスの名前を指定できます。また、RCVEXIT パラメーターには、WMQReceiveExit インターフェースを実装するクラスの名前を指定できます。 Java でのチャネル出口の作成方法の詳細については、281 ページの『Java for IBM MQ classes for JMS でのチャネル出口の作成』を参照してください。

  Java 以外の言語で作成されたチャネル出口の使用もサポートされています。 別の言語で作成されたチャネル出口を DEFINE CHANNEL コマンドの SCYEXIT、SENDEXIT、および RCVEXIT パラメーターに指定する方法については、DEFINE CHANNEL を参照してください。

JMS クライアントの自動再接続

ネットワーク、キュー・マネージャー、またはサーバーで障害が起こった後に自動的に再接続が行われるよう、JMS クライアントを構成します。

通常、スタンドアロンの IBM MQ classes for JMS アプリケーションをクライアント・トランスポートを使用してキュー・マネージャーに接続し、キュー・マネージャーが何らかの理由 (ネットワーク障害、キュー・マネージャー障害、キュー・マネージャーの停止など) で使用できなくなった場合は、アプリケーションが次回キュー・マネージャーと通信しようとするときに、IBM MQ classes for JMS は JMSException をスローします。 アプリケーションは JMSException をキャッチして、キュー・マネージャーへの再接続を試行する必要があります。 アプリケーションの設計は、クライアントの自動再接続を有効にすることで単純化できます。 キュー・マネージャーが使用不可になると、IBM MQ classes for JMS は、アプリケーションの代わりに自動的にキュー・マネージャーへの再接続を試行します。 これは、アプリケーションが再接続ロジックを組み込む必要がないことを意味します。

自動クライアント再接続のこの実装の使用は、Java Platform、Enterprise Edition アプリケーション・サーバー内ではサポートされていません。 代替の実装については、292 ページの『Java EE 環境でのクライアントの自動再接続の使用』を参照してください。

JMS クライアントの自動再接続の使用
スタンドアロンの IBM MQ classes for JMS アプリケーションで CONNECTIONNAMELIST または CCDTURL プロパティーが設定されている接続ファクトリーを使用している場合、そのアプリケーションではクライアントの自動再接続を使用できます。

自動クライアント再接続は、キュー・マネージャー (高可用性 (HA) 構成の一部であるものを含む) に再接続するために使用できます。 HA 構成には、複数インスタンス・キュー・マネージャー、RDQM キュー・マネージャー、または IBM MQ アプライアンス上の HA キュー・マネージャーが含まれます。

IBM MQ classes for JMS で提供されるクライアントの自動再接続機能の動作は、以下のプロパティーによって異なります。

JMS 接続ファクトリーのプロパティー TRANSPORT (短縮名は TRAN)

TRANSPORT は、接続ファクトリーを使用するアプリケーションからキュー・マネージャーへの接続方法を指定します。 クライアントの自動再接続を使用する場合は、このプロパティーを CLIENT の値に設定する必要があります。 TRANSPORT プロパティーが BIND、DIRECT、または DIRECTHTTP に設定されている接続ファクトリーを使用するキュー・マネージャーに接続されているアプリケーションでは、クライアントの自動再接続を使用することはできません。

JMS 接続ファクトリーのプロパティー QMANAGER (短縮名は QMGR)

QMANAGER プロパティーは、接続ファクトリーの接続先であるキュー・マネージャーの名前を指定します。

JMS 接続ファクトリーのプロパティー CONNECTIONNAMELIST (短縮名は CRHOSTS)

CONNECTIONNAMELIST プロパティーはコンマ区切りのリストです。各エントリーには、CLIENT トランスポートの使用時に QMANAGER プロパティーで指定されたキュー・マネージャーに接続するために使用されるホスト名とポートの情報が含まれます。 このリストの形式は host name(port), host name(port) です。

JMS 接続ファクトリーのプロパティー CCDTURL (短縮名は CCDT)

CCDTURL プロパティーは、IBM MQ classes for JMS が CCDT を使用してキュー・マネージャーに接続するときに使用するクライアント・チャネル定義テーブルを指します。

JMS 接続ファクトリーのプロパティー CLIENTRECONNECTOPTIONS (短縮名は CROPT)

CLIENTRECONNECTOPTIONS は、キュー・マネージャーが使用可能になった場合に、アプリケーションの代わりに IBM MQ classes for JMS が自動的にキュー・マネージャーへの接続を試行するかどうかを制御します。

クライアント構成ファイルの Channels スタンザの DefRecon 属性

DefRecon 属性は、すべてのアプリケーションで自動再接続を有効にするか、自動的に再接続するように作成されたアプリケーションで自動再接続を無効にするための管理オプションを提供します。

クライアントの自動再接続を使用できるのは、アプリケーションが正常にキュー・マネージャーに接続された場合のみです。

アプリケーションが CLIENT トランスポートを使用するキュー・マネージャーに接続した場合、IBM MQ classes for JMS は接続ファクトリーのプロパティー CLIENTRECONNECTOPTIONS の値を使用して、アプリケーションの接続先であるキュー・マネージャーが使用不可になった場合にクライアントの自動再接続を使用するかどうかを判別します。 表 1 に、CLIENTRECONNECTOPTIONS プロパティーに指定可能な値と、これらの値ごとの IBM MQ classes for JMS の動作を示します。

表 44. 指定可能な *CLIENTRECCECTOPTIONS* プロパティー値。

CLIENTRECONNECTOPTIONS	**IBM MQ classes for JMS の動作**
ANY	CONNECTIONNAMELIST が設定されている場合は、CONNECTIONNAMELIST プロパティーの値を使用して、ホスト名とポートの組み合わせへの接続を開き、任意のキュー・マネージャーに接続します。 このクライアントの自動再接続オプションを使用するためには、QMANAGER プロパティーをデフォルト値または「*」に設定する必要があります。 CCDTURL が設定されている場合は、CCDTURL プロパティーで指定されたクライアント・チャネル定義テーブルを開いてテーブルのエントリーを選出し、そのエントリーを使用してキュー・マネージャーへのクライアント接続チャネルを開始します。 このクライアントの自動再接続オプションを使用するには、QMANAGER プロパティーを次のいずれかに設定する必要があります。 • アスタリスク (*)。 • アスタリスク (*) とそれに続くキュー・マネージャー・グループ名。 • 空ストリング、またはブランク文字のみを含むストリング。
ASDEF	DefRecon の値を使用して、クライアントの自動再接続が使用可能かどうかを判別します。
DISABLED	クライアントの自動再接続を実行せずに、JMSException をアプリケーションに返します。
QMGR	クライアントが同じキュー・マネージャーに再接続する必要があることを指定します。 同じキュー・マネージャーの別のインスタンスへの再接続が必要となる高可用性ソリューションでは、このオプションを使用する必要があります。 CONNECTIONNAMELIST が設定されている場合、CONNECTIONNAMELIST プロパティーの値を使用して、ホスト名とポートの組み合わせへの接続を開き、QMANAGER プロパティーで指定された任意のキュー・マネージャーに接続します。 CCDTURL が設定されている場合、CCDTURL プロパティーで指定されたクライアント・チャネル定義テーブルを開き、QMANAGER プロパティーで指定されたキュー・マネージャーの名前と一致するエントリーをテーブルで見つけてから、それらのエントリーを使用してそのキュー・マネージャーへのクライアント接続チャネルを開始します。

CONNECTIONNAMELIST が設定されている場合、クライアント自動再接続の実行時に、IBM MQ classes for JMS は接続ファクトリーのプロパティー CONNNECTIONNAMELIST の情報を使用して、再接続するシステムを判別します。

IBM MQ classes for JMS は最初に、CONNECTIONNAMELIST の最初のエントリーで指定されたホスト名とポートを使用して再接続を試行します。 接続された場合、IBM MQ classes for JMS は次に、QMANAGER プロパティーで指定された名前を持つキュー・マネージャーへの接続を試行します。 キュー・マネージャ

ーへの接続が確立できる場合、IBM MQ classes for JMS は、クライアントの自動再接続の前にアプリケーションが開いた IBM MQ オブジェクトをすべて再オープンして、従来どおり実行を続けます。

CONNECTIONNAMELIST の最初のエントリーを使用して、必要なキュー・マネージャーへの接続が確立できない場合、IBM MQ classes for JMS は CONNECTIONNAMELIST の 2 番目のエントリーを試し、以下同様に続きます。

IBM MQ classes for JMS は、CONNECTIONNAMELIST 内のエントリーをすべて試したら、再度再接続を試行する前に一定の時間待機します。 新たに再接続を試行するために、IBM MQ classes for JMS は CONNECTIONNAMELIST 内の最初のエントリーから始めます。 その後は、再接続が行われるか、CONNECTIONNAMELIST の最後に達する (この時点で、IBM MQ classes for JMS は再試行前に一定の時間待機します) まで、CONNECTIONNAMELIST 内のエントリーをそれぞれ順に試します。

CCDTURL が設定されている場合、クライアント自動再接続の実行時に、IBM MQ classes for JMS は CCDTURL プロパティーで指定されたクライアント・チャネル定義テーブルを使用して、再接続するシステムを判別します。

IBM MQ classes for JMS は最初にクライアント・チャネル定義テーブルを解析して、QMANAGER プロパティーの値と一致する適切なエントリーを見つけます。 エントリーが見つかった場合、IBM MQ classes for JMS はそのエントリーを使用して、必要なキュー・マネージャーへの再接続を試行します。 キュー・マネージャーへの接続が確立できる場合、IBM MQ classes for JMS は、クライアントの自動再接続の前にアプリケーションが開いた IBM MQ オブジェクトをすべて再オープンして、従来どおり実行を続けます。

必要なキュー・マネージャーへの接続を確立できない場合、IBM MQ classes for JMS はクライアント・チャネル定義テーブルで別の適切なエントリーを探し、そのエントリーの使用を試行し、以下同様に続きます。

IBM MQ classes for JMS は、クライアント・チャネル定義テーブル内の適切なエントリーをすべて試したら、再度再接続を試行する前に一定の時間待機します。 新たに再接続を試行するために、IBM MQ classes for JMS はクライアント・チャネル定義テーブルを再度解析して、最初の適切なエントリーを試します。 その後は、再接続が行われるか、クライアント・チャネル定義テーブル内の最後の適切なエントリーが試される (この時点で、IBM MQ classes for JMS は再試行前に一定の時間待機します) まで、クライアント・チャネル定義テーブル内の適切なエントリーをそれぞれ順に試します。

CONNECTIONNAMELIST と CCDTURL のどちらを使用している場合も、クライアントの自動再接続のプロセスは、IBM MQ classes for JMS が QMANAGER プロパティーで指定されたキュー・マネージャーに正常に再接続するまで続きます。

デフォルトでは、再接続は以下の間隔で試行されます。

- 最初の試行は、1 秒に 250 ミリ秒までのランダム要素を加算した初期遅延の後、実行されます。
- 2 回目の試行は、最初の試行が失敗した後、2 秒に 500 ミリ秒までのランダム間隔を加算した遅延を空けて、実行されます。
- 3 回目の試行は、2 回目の試行が失敗した後、4 秒に 1 秒までのランダム間隔を加算した遅延を空けて、実行されます。
- 4 回目の試行は、3 回目の試行が失敗した後、8 秒に 2 秒までのランダム間隔を加算した遅延を空けて、実行されます。
- 5 回目の試行は、4 回目の試行が失敗した後、16 秒に 4 秒までのランダム間隔を加算した遅延を空けて、実行されます。
- 6 回目の試行、およびそれ以後のすべての試行は、前回の試行が失敗した後、25 秒に 6.25 秒までのランダム間隔を加算した遅延を空けて、実行されます。

再接続の試行は、一部は固定で一部はランダムの間隔で遅延されます。 これは、使用できなくなったキュー・マネージャーに接続されたすべての IBM MQ classes for JMS アプリケーションが同時に再接続できないようにするために行われます。

キュー・マネージャーのリカバリーや、スタンバイ・キュー・マネージャーがアクティブになるまでに必要な時間をより正確に反映させるために、デフォルト値を増やす必要がある場合は、クライアント構成ファイルの Channel スタンザの ReconDelay 属性を変更します。詳しくは、クライアント構成ファイルの CHANNELS スタンザを参照してください。

再接続が自動的に行われた後に、IBM MQ classes for JMS アプリケーションが引き続き正常に稼働できるかどうかは、再接続の設計にかかっています。 関連トピックを参照し、自動再接続機能を使用できるアプリケーションの設計方法を理解してください。

キュー・マネージャーを使用することができなくなったことを示す理由コード

IBM MQ classes for JMS の自動再接続の試行時に、キュー・マネージャーが使用できなくなったか、キュー・マネージャーに到達できないことを示す理由コードを紹介します。

286 ページの『JMS クライアントの自動再接続』では、JMSExceptions の概要と、アプリケーションを自動的に再始動する方法を簡単に説明しています。また、287 ページの『JMS クライアントの自動再接続の使用』では、クライアントの自動再接続に必要な要件の詳細を説明しています。

アプリケーションで検査する必要のある IBM MQ 理由コードを以下のリストで説明します。

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

エンタープライズ・アプリケーションに対してスローされるほとんどの JMSExceptions には、理由コードを保持するリンク付き MQException が含まれます。 前述のリストにある理由コードの再試行ロジックを実装するには、次の例のようなコードを使用して、エンタープライズ・アプリケーションでこのリンク付き例外を検査する必要があります。

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

関連概念
IBM MQ classes for JMS

Java SE および *Java EE* 環境での自動クライアント再接続の使用

IBM MQ 自動クライアント再接続を使用して、 Java SE および Java EE 環境内のさまざまな高可用性 (HA) ソリューションおよび災害復旧 (DR) ソリューションを容易にすることができます。

以下のような各種のプラットフォームのさまざまな HA ソリューションおよび DR ソリューションを使用できます。

- Multi 複数インスタンス・キュー・マネージャーは、異なる複数のサーバー上で構成されている同じキュー・マネージャーのインスタンスです (複数インスタンス・キュー・マネージャーを参照)。 キュー・マネージャーの 1 つのインスタンスはアクティブ・インスタンスとして定義され、もう 1 つのインスタンスはスタンバイ・インスタンスとして定義されます。 アクティブ・インスタンスで障害が発生すると、複数インスタンス・キュー・マネージャーは、スタンバイ・サーバーで自動的に再始動します。

  アクティブ・キュー・マネージャーとスタンバイ・キュー・マネージャーの両方に、同じキュー・マネージャー ID (QMID) があります。 複数インスタンス・キュー・マネージャーに接続する IBM MQ クライアント・アプリケーションは、自動クライアント再接続を使用して、キュー・マネージャーのスタンバイ・インスタンスに自動的に再接続するように構成できます。

- Linux RDQM (複製データ・キュー・マネージャー) は、Linux プラットフォーム上で使用できる高可用性ソリューションです (RDQM 高可用性を参照)。 RDQM 構成は、高可用性 (HA) グループとして構成されている 3 つのサーバーから成り、その個々にキュー・マネージャーのインスタンスがあります。 インスタンスの 1 つは実行中のキュー・マネージャーで、データを他の 2 つのインスタンスに同期的に複製します。 このキュー・マネージャーを実行しているサーバーに障害が発生すると、別のキュー・マネージャーのインスタンスが開始され、このインスタンスには操作に使用する現行データがあります。 キュー・マネージャーのインスタンスは 3 つとも浮動 IP アドレスを共有しているので、クライアントの構成時に使用する必要がある IP アドレスは 1 つだけです。 RDQM キュー・マネージャーに接続するクライアント・アプリケーションは、自動クライアント再接続を使用して、キュー・マネージャーのスタンバイ・インスタンスに自動的に再接続するように構成できます。

- MQ Appliance HA ソリューションは、 IBM MQ アプライアンスのペアによって提供することもできます (IBM MQ Appliance 資料の 高可用性 および 災害復旧 を参照してください)。 一方のアプライアンスで HA キュー・マネージャーを実行し、もう一方のアプライアンス上のこのキュー・マネージャーのスタンバイ・インスタンスに同期的にデータを複製します。 1 次アプライアンスで障害が発生した場合は、もう一方のアプライアンスでキュー・マネージャーが自動的に起動して実行されます。 キュー・マネージャーのこれら 2 つのインスタンスは、浮動 IP アドレスを共有するように構成し、そうすることにより、クライアントを単一の IP アドレスのみを使用して構成できるようにすることができます。 IBM MQ Appliance 上の HA キュー・マネージャーに接続するクライアント・アプリケーションは、自動クライアント再接続を使用して、キュー・マネージャーのスタンバイ・インスタンスに自動的に再接続するように構成できます。

注 : WebSphere Application Server などの Java EE 環境では、IBM MQ classes for JMS によって提供される機能を使用するアクティブ化仕様を含む自動クライアント再接続はサポートされていません。 IBM MQ リソース・アダプターは、アクティベーション・スペックの接続先キュー・マネージャーが使用不可になった場合に、そのアクティベーション・スペックを再接続するための独自のメカニズムを提供します。 詳しくは、293 ページの『Java EE 環境での自動クライアント再接続のサポート』を参照してください。

関連概念
複数インスタンス・キュー・マネージャー
クライアントの自動再接続
関連資料
RDQM 高可用性

Java SE 環境でのクライアントの自動再接続の使用
Java SE 環境で実行される IBM MQ classes for JMS を使用するアプリケーションは、接続ファクトリー・プロパティー **CLIENTRECONNECTOPTIONS** を介して自動クライアント再接続機能を使用できます。

接続ファクトリー・プロパティー **CLIENTRECONNECTOPTIONS** は、2 つの追加接続ファクトリー・プロパティー **CONNECTIONNAMELIST** および **CCDTURL** を使用して、キュー・マネージャーが実行しているサーバーに接続する方法を決定します。

CONNECTIONNAMELIST プロパティー

CONNECTIONNAMELIST プロパティーは、クライアント・モードでキュー・マネージャーに接続するために使用されるホスト名とポート情報が入った、コンマ区切りのリストです。 このプロパティーは、**QMANAGER** 値および **CHANNEL** 値と共に使用されます。 アプリケーションが **CONNECTIONNAMELIST** プロパティーを使用してクライアント接続を作成する際 IBM MQ classes for JMS リストの順序で各ホストへの

接続を試みます。 最初のキュー・マネージャー・ホストが使用できない場合、IBM MQ classes for JMS はリストの次のホストへの接続を試みます。 接続を作成できずに接続名リストの終わりに到達した場合、IBM MQ classes for JMS は MQRC_QMGR_NOT_AVAILABLE IBM MQ 理由コードをスローします。

アプリケーションの接続先のキュー・マネージャーで障害が発生すると、そのキュー・マネージャーへの接続に **CONNECTIONNAMELIST** を使用したすべてのアプリケーションは、そのキュー・マネージャーが使用できないことを示す例外を受け取ります。 アプリケーションはその例外をキャッチし、使用していたすべてのリソースをクリアする必要があります。 接続を作成するため、アプリケーションは接続ファクトリーを使用する必要があります。 接続ファクトリーは再びリストの順序で各ホストへの接続を試みますが、障害を発生したキュー・マネージャーは現在使用不可になっています。 接続ファクトリーは、リスト内の別のホストへの接続を試みます。

CCDTURL プロパティー

CCDTURL プロパティーにはクライアント・チャネル定義テーブル (CCDT) を指す Uniform Resource Locator (URL) が含まれており、このプロパティーは **QMANAGER** プロパティーと共に使用されます。 CCDT には、IBM MQ システムで定義されたキュー・マネージャーへの接続に使用されるクライアント・チャネルのリストが含まれます。 IBM MQ classes for JMS が CCDT を使用する方法については、285 ページの『IBM MQ classes for JMS を含むクライアント・チャネル定義テーブルの使用』を参照してください。

CLIENTRECONNECTOPTIONS プロパティーを使用した IBM MQ classes for JMS 内部での自動クライアント再接続の有効化

CLIENTRECONNECTOPTIONS プロパティーは、IBM MQ classes for JMS 内部で自動クライアント再接続を有効にするために使用します。 このプロパティーに指定可能な値は、以下のとおりです。

ASDEF
自動クライアント再接続の動作は、IBM MQ クライアント構成ファイル (mqclient.ini) のチャネル・スタンザに指定されているデフォルト値によって定義されます。

disabled
自動クライアント再接続は無効です。

QMGR
IBM MQ classes for JMS は、以下のいずれかのオプションを使用して、以前接続したキュー・マネージャーと同じキュー・マネージャー ID を使用して、キュー・マネージャーへの接続を試みます。

- **CONNECTIONNAMELIST** プロパティーおよび **CHANNEL** プロパティーで定義されたチャネル。
- **CCDTURL** プロパティーで定義された CCDT。

ANY
IBM MQ classes for JMS は、**CONNECTIONNAMELIST** プロパティーまたは **CCDTURL** のいずれかを使用して、同じ名前のキュー・マネージャーへの再接続を試みます。

関連情報
クライアント構成ファイルの CHANNELS スタンザ

Java EE 環境でのクライアントの自動再接続の使用
IBM MQ リソース・アダプターは、Java EE (Java Platform, Enterprise Edition) 環境にデプロイすることができ、WebSphere Application Server IBM MQ メッセージング・プロバイダーは IBM MQ classes for JMS を使用して IBM MQ キュー・マネージャーと通信します。 IBM MQ リソース・アダプターおよび WebSphere Application Server IBM MQ メッセージング・プロバイダーは、アクティベーション・スペックを許可するいくつかのメカニズム、WebSphere Application Server リスナー・ポートおよびアプリケーションを提供します。それらはキュー・マネージャーに自動的に再接続するために、クライアント・コンテナー内で実行されます。 Enterprise JavaBeans (EJB) および Web ベース・アプリケーションは、独自の再接続ロジックを実装する必要があります。

注 : IBM MQ classes for JMS が提供する機能を使用したアクティベーション・スペックを含む自動クライアント再接続はサポートされていません (286 ページの『JMS クライアントの自動再接続』を参照)。 IBM MQ リソース・アダプターは、アクティベーション・スペックの接続先キュー・マネージャーが使用不可になった場合に、そのアクティベーション・スペックを再接続するための独自のメカニズムを提供します。

リソース・アダプターが提供するメカニズムは、以下のもので制御されます。

- IBM MQ リソース・アダプターのプロパティー **reconnectionRetryCount**。
- IBM MQ リソース・アダプターのプロパティー **reconnectionRetryInterval**。
- アクティベーション・スペックのプロパティー **connectionNameList**。

これらのプロパティーの詳細については、450 ページの『ResourceAdapter オブジェクト・プロパティーの構成』を参照してください。

メッセージ駆動型 Bean アプリケーションの onMessage() メソッド内や、Java Platform, Enterprise Edition 環境内で実行されているその他のアプリケーション内でのクライアント自動再接続の使用はサポートされていません。 接続先のキュー・マネージャーが使用不可になった場合、アプリケーションは独自の再接続ロジックを実装する必要があります。 詳しくは、300 ページの『Java EE アプリケーションへの再接続ロジックの実装』を参照してください。

Java EE 環境での自動クライアント再接続のサポート

WebSphere Application Server、IBM MQ リソース・アダプター、および WebSphere Application Server IBM MQ メッセージング・プロバイダーなどの Java EE 環境では、アプリケーションがキュー・マネージャーに自動的に再接続するためのメカニズムがいくつか提供されています。 ただし、場合によっては、サポートに制限が適用されます。

Java EE 環境および WebSphere Application Server IBM MQ メッセージング・プロバイダーにデプロイできる IBM MQ リソース・アダプターは、IBM MQ classes for JMS を使用して、IBM MQ キュー・マネージャーと通信します。

次の表は、IBM MQ リソース・アダプターと WebSphere Application Server IBM MQ メッセージング・プロバイダーによる自動クライアント再接続のサポートを要約しています。

表 *45. Java EE* 環境での自動クライアント再接続オプションのサポートの要約

自動再接続のオプション	**CONNECTIONNAMELIST プロパティー**	**CCDTURL プロパティー**	**CLIENTRECONNECTOPTIONS プロパティー**	**自動クライアント再接続に対する代替アプローチ**
アクティベーション・スペック	制限付きでサポート	制限付きでサポート	サポート対象外	Java EE 環境とアクティベーション・スペックはそれぞれ独自の再接続メカニズムを提供します
WebSphere Application Server リスナー・ポート	制限付きでサポート	制限付きでサポート	サポート対象外	WebSphere Application Server は独自の再接続メカニズムを提供します
Enterprise JavaBeans および Web ベース・アプリケーション	制限付きでサポート	制限付きでサポート	サポート対象外	アプリケーションは独自の再接続ロジックを実装する必要があります
クライアント・コンテナー内部で実行するアプリケーション	サポート対象	サポート対象	サポート対象	適用外

IBM MQ classes for JMS などの Java EE 環境にインストールされたメッセージ駆動型 Bean アプリケーションは、アクティベーション・スペックを使用して IBM MQ システム上のメッセージを処理できます。 アクティベーション・スペックを使用して、IBM MQ システムに着信するメッセージが検出され、それらのメッセージが処理のためにメッセージ駆動型 Bean に送信されます。 メッセージ駆動型 Bean は、

onMessage() メソッド内から IBM MQ システムへの接続を増やすこともできます。 これらの接続がどのようにして自動クライアント再接続を使用できるかについて詳しくは、Enterprise JavaBeans および Web ベース・アプリケーションを参照してください。

アクティベーション・スペック
アクティベーション・スペックでは、**CONNECTIONNAMELIST** プロパティーおよび **CCDTURL** プロパティーは制約付きでサポートされていますが、**CLIENTRECONNECTOPTIONS** プロパティーはサポートされていません。

WebSphere Application Server のような Java EE 環境にインストールされているメッセージ駆動型 Bean (MDB) アプリケーションは、IBM MQ システム上のメッセージを処理するためにアクティベーション・スペックを使用することができます。

アクティベーション・スペックを使用して、IBM MQ システムに着信するメッセージが検出され、それらのメッセージが処理のために MDB に送信されます。 このセクションでは、アクティベーション・スペックが IBM MQ システムをモニターする方法について取り上げます。

MDB は、 onMessage() メソッドの内部から IBM MQ システムへの追加接続を行うこともできます。

これらの接続が自動クライアント再接続を使用する方法について詳しくは、297 ページの『Enterprise JavaBeans および Web ベース・アプリケーション』 を参照してください。

CONNECTIONNAMELIST プロパティー

始動時に、アクティベーション・スペックは以下を使用してキュー・マネージャーへの接続を試みます。

- **QMANAGER** プロパティーで指定されたキュー・マネージャー
- **CHANNEL** プロパティーで指定されたチャネル
- **CONNECTIONNAMELIST** の最初の項目で指定されたホスト名とポート情報

アクティベーション・スペックがリスト内の最初の項目を使用してキュー・マネージャーに接続できない場合、アクティベーション・スペックは 2 番目の項目に移動し、キュー・マネージャーへの接続が確立するか、リストの終わりに到達するまで、以降同様に移動します。

アクティベーション・スペックは、**CONNECTIONNAMELIST** 内のどの項目を使用しても指定されたキュー・マネージャーに接続できない場合は停止します。この場合は、アクティベーション・スペックを再始動する必要があります。

アクティベーション・スペックはいったん実行されると、IBM MQ システムからメッセージを受け取り、そのメッセージを処理のために MDB に送信します。

メッセージの処理中にキュー・マネージャーで障害が発生すると、Java EE 環境がその障害を検出し、アクティベーション・スペックの再接続を試みます。

アクティベーション・スペックは、再接続を試行するとき、以前と同じく **CONNECTIONNAMELIST** プロパティーの情報を使用します。

アクティベーション・スペックは、**CONNECTIONNAMELIST** 内のすべての項目を試してもキュー・マネージャーに接続できない場合、IBM MQ リソース・アダプター・プロパティー **reconnectionRetryInterval** に指定された期間待機してから、再試行します。

IBM MQ リソース・アダプター・プロパティー **reconnectionRetryCount** は、アクティベーション・スペックが停止され、手動による再始動が必要になるまでに試行される再接続の連続回数を定義します。

アクティベーション・スペックがいったん IBM MQ システムに再接続すると、Java EE 環境は必要なすべてのトランザクション・クリーンアップを実行し、処理のための MDB へのメッセージの送信を再開します。

トランザクション・クリーンアップを正常に動作させるためには、障害を発生したキュー・マネージャーのログに Java EE 環境がアクセス可能でなければなりません。

アクティベーション・スペックが XA トランザクションに参加するトランザクション MDB と共に使用されており、複数インスタンス・キュー・マネージャーに接続している場合、**CONNECTIONNAMELIST** にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

トランザクション MDB がスタンドアロン・キュー・マネージャーで使用されている場合、アクティベーション・スペックが障害後に常に同じシステムで実行されている同じキュー・マネージャーに再接続するようにするには、 **CONNECTIONNAMELIST** プロパティーに単一の項目が含まれている必要があります。

CCDTURL プロパティー

開始時に、アクティベーション・スペックは、クライアント・チャネル定義テーブル (CCDT) の最初の項目を使用して、 **QMANAGER** プロパティーで指定されたキュー・マネージャーへの接続を試行します。

アクティベーション・スペックがテーブル内の最初の項目を使用してキュー・マネージャーに接続できない場合、アクティベーション・スペックは 2 番目の項目に移動し、キュー・マネージャーへの接続が確立するか、テーブルの終わりに到達するまで、以降同様に移動します。

アクティベーション・スペックは、CCDT 内のどの項目を使用しても指定されたキュー・マネージャーに接続できない場合は停止します。この場合は、アクティベーション・スペックを再始動する必要があります。

アクティベーション・スペックはいったん実行されると、IBM MQ システムからメッセージを受け取り、そのメッセージを処理のために MDB に送信します。

メッセージの処理中にキュー・マネージャーで障害が発生すると、Java EE 環境がその障害を検出し、アクティベーション・スペックの再接続を試みます。

アクティベーション・スペックは、再接続を試行するとき、以前と同じく CCDT プロパティーの情報を使用します。

アクティベーション・スペックは、CCDT 内のすべての項目を試してもキュー・マネージャーに接続できない場合、IBM MQ リソース・アダプター・プロパティー **reconnectionRetryInterval** に指定された期間待機してから、再試行します。

IBM MQ リソース・アダプター・プロパティー **reconnectionRetryCount** は、アクティベーション・スペックが停止され、手動による再始動が必要になるまでに試行される再接続の連続回数を定義します。

アクティベーション・スペックがいったん IBM MQ システムに再接続すると、Java EE 環境は必要なすべてのトランザクション・クリーンアップを実行し、処理のための MDB へのメッセージの送信を再開します。

トランザクション・クリーンアップを正常に動作させるためには、障害を発生したキュー・マネージャーのログに Java EE 環境がアクセス可能でなければなりません。

アクティベーション・スペックが XA トランザクションに参加するトランザクション MDB と共に使用されており、複数インスタンス・キュー・マネージャーに接続している場合、CCDT にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

トランザクション MDB がスタンドアロン・キュー・マネージャーと共に使用されている場合、CCDT には単一の項目のみが含まれている必要があります。これは、アクティベーション・スペックが障害後に常に同じシステムで実行している同じキュー・マネージャーに確実に再接続するようにするためです。

同じアクティブ・キュー・マネージャーに接続が行われるようにするため、アクティベーション・スペックと共に使用する CCDT の **AFFINITY** プロパティーの *PREFERRED* のデフォルト値を設定したことを確認します。

CLIENTRECONNECTOPTIONS プロパティー

アクティベーション・スペックは、独自の再接続機能を提供します。 提供される機能により、仕様は接続先のキュー・マネージャーに障害が発生しても、自動的に IBM MQ システムに再接続できます。

このため、IBM MQ classes for JMS の提供する自動クライアント再接続機能はサポートされていません。

Java EE で使用されるすべてのアクティベーション・スペックについて、 **CLIENTRECONNECTOPTIONS** プロパティーを *DISABLED* に設定する必要があります。

WebSphere Application Server リスナー・ポート
WebSphere Application Server にインストールされているメッセージ駆動型 Bean (MDB) アプリケーションは、リスナー・ポートを使用して IBM MQ システム上のメッセージを処理することもできます。

リスナー・ポートを使用して、IBM MQ システムに着信するメッセージが検出され、それらのメッセージが処理のために MDB に送信されます。 このトピックでは、リスナー・ポートが IBM MQ システムをどのようにモニターするかについて説明します。

MDB は、 onMessage() メソッドの内部から IBM MQ システムへの追加接続を行うこともできます。

これらの接続が自動クライアント再接続を使用する方法について詳しくは、297 ページの『Enterprise JavaBeans および Web ベース・アプリケーション』を参照してください。

WebSphere Application Server リスナー・ポートの場合:

- **CONNECTIONNAMELIST** および **CCDTURL** は制限付きでサポートされます
- **CLIENTRECONNECTOPTIONS** はサポートされません

CONNECTIONNAMELIST プロパティー

リスナー・ポートは、 IBM MQ への接続時に JMS 接続プールを使用するため、接続プールの使用による影響を受けます。 詳しくは、294 ページの『アクティベーション・スペック』を参照してください。

空き接続がなく、この接続ファクトリーから作成された接続の数がまだ接続の最大数に達していない場合、**CONNECTIONNAMELIST** プロパティーを使用して IBM MQ への新規接続の作成が試みられます。

CONNECTIONNAMELIST 内のすべての IBM MQ システムにアクセスできない場合、リスナー・ポートは停止します。

その後、リスナー・ポートはメッセージ・リスナー・サービスのカスタム・プロパティー **RECOVERY.RETRY.INTERVAL** で指定された期間待機してから、再接続を再試行します。

この再接続の試みでは、接続プール内に空き接続がないか検査されます。接続の試行の間に、空き接続が戻されていることがあるためです。 使用可能な接続がなければ、リスナー・ポートは以前と同じく **CONNECTIONNAMELIST** を使用します。

リスナー・ポートがいったん IBM MQ システムに再接続すると、Java EE 環境は必要なすべてのトランザクション・クリーンアップを実行してから、処理のための MDB へのメッセージの送信を再開します。

トランザクション・クリーンアップを正常に動作させるためには、障害を発生したキュー・マネージャーのログに Java EE 環境がアクセス可能でなければなりません。

リスナー・ポートが XA トランザクションに参加するトランザクション MDB と共に使用されており、**複数インスタンス・キュー・マネージャー**に接続している場合、**CONNECTIONNAMELIST** にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

トランザクション MDB がスタンドアロン・キュー・マネージャーで使用されている場合、アクティベーション・スペックが障害後に常に同じシステムで実行されている同じキュー・マネージャーに再接続するようにするには、 **CONNECTIONNAMELIST** プロパティーに単一の項目が含まれている必要があります。

CCDTURL プロパティー

リスナー・ポートは、開始時に、CCDT の最初の項目を使用して、 **QMANAGER** プロパティーで指定されたキュー・マネージャーへの接続を試行します。

リスナー・ポートがテーブル内の最初の項目を使用してキュー・マネージャーに接続できない場合、リスナー・ポートは 2 番目の項目に移動し、キュー・マネージャーへの接続が確立するか、テーブルの終わりに到達するまで、以降同様に移動します。

リスナー・ポートは、CCDT 内のどの項目を使用しても指定されたキュー・マネージャーに接続できない場合は停止します。

その後、リスナー・ポートはメッセージ・リスナー・サービスのカスタム・プロパティー **RECOVERY.RETRY.INTERVAL** で指定された期間待機してから、再接続を再試行します。

この再接続の試みは、以前と同じく、CCDT 内のすべての項目に対して実行されます。

リスナー・ポートはいったん実行されると、IBM MQ システムからメッセージを受け取り、それらを処理のために MDB に送信します。

メッセージの処理中にキュー・マネージャーで障害が発生すると、Java EE 環境がその障害を検出し、リスナー・ポートの再接続を試みます。 リスナー・ポートは、再接続を試行するとき、CCDT 内の情報を使用します。

リスナー・ポートは、CCDT 内のすべての項目を試してもキュー・マネージャーに接続できない場合、**RECOVERY.RETRY.INTERVAL** プロパティーに指定された期間待機してから、再試行します。

メッセージ・リスナー・サービスのプロパティー **MAX.RECOVERY.RETRIES** は、リスナー・ポートが停止され、手動による再始動が必要になるまでに試行される再接続の連続回数を定義します。

リスナー・ポートがいったん IBM MQ システムに再接続すると、Java EE 環境は必要なすべてのトランザクション・クリーンアップを実行してから、処理のための MDB へのメッセージの送信を再開します。

トランザクション・クリーンアップを正常に動作させるためには、障害を発生したキュー・マネージャーのログに Java EE 環境がアクセス可能でなければなりません。

リスナー・ポートが XA トランザクションに参加するトランザクション MDB と共に使用されており、複数インスタンス・キュー・マネージャーに接続している場合、CCDT にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

トランザクション MDB がスタンドアロン・キュー・マネージャーと共に使用されている場合、CCDT には単一の項目のみが含まれている必要があります。これは、リスナー・ポートが障害後に常に同じシステムで実行している同じキュー・マネージャーに確実に再接続するようにするためです。

同じアクティブ・キュー・マネージャーに接続が行われるようにするため、リスナー・ポートと共に使用する CCDT の **AFFINITY** プロパティーの *PREFERRED* のデフォルト値を設定したことを確認します。

CLIENTRECONNECTOPTIONS プロパティー

リスナー・ポートは、独自の再接続機能を提供します。 提供される機能により、リスナー・ポートは接続先のキュー・マネージャーに障害が発生しても、自動的に IBM MQ システムに再接続できます。

このため、IBM MQ classes for JMS の提供する自動クライアント再接続機能はサポートされていません。

Java EE で使用されるすべてのリスナー・ポートに対して、 **CLIENTRECONNECTOPTIONS** プロパティーを *DISABLED* に設定する必要があります。

Enterprise JavaBeans および *Web* ベース・アプリケーション
Enterprise JavaBean (EJB) アプリケーションおよびサーブレットなどの Web コンテナー内で実行するアプリケーションは、JMS 接続ファクトリーを使用して IBM MQ キュー・マネージャーへの接続を作成します。

EJB および Web ベース・アプリケーションには以下の制限事項が適用されます。

- **CONNECTIONNAMELIST** および **CCDTURL** は制限付きでサポートされます
- **CLIENTRECONNECTOPTIONS** はサポートされません

CONNECTIONNAMELIST プロパティー

Java EE 環境で JMS 接続用の接続プールが提供されている場合、これが **CONNECTIONNAMELIST** プロパティーの動作にどのように影響するかについては、299 ページの『接続プールでの CONNECTIONNAMELIST または CCDT の使用』 を参照してください。

Java EE 環境が JMS 接続プールを提供しない場合。 アプリケーションは、 Java SE アプリケーションと同じ方法で **CONNECTIONNAMELIST** プロパティーを使用します。

アプリケーションが XA トランザクションに参加するトランザクション MDB と共に使用されており、複数インスタンス・キュー・マネージャーに接続している場合、**CONNECTIONNAMELIST** にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

アプリケーションがスタンドアロン・キュー・マネージャーで使用されている場合、**CONNECTIONNAMELIST** プロパティーには単一の項目が含まれている必要があります。これは、アプリケーションが障害の後に常に同じシステムで実行されている同じキュー・マネージャーに再接続するようにするためです。

CCDTURL プロパティー

Java EE 環境で JMS 接続の接続プールが提供されている場合、これが **CCDTURL** プロパティーの動作にどのように影響するかについては、299 ページの『接続プールでの CONNECTIONNAMELIST または CCDT の使用』 を参照してください。

Java EE 環境が JMS 接続プールを提供しない場合。 アプリケーションは、 Java SE アプリケーションと同じ方法で **CCDTURL** プロパティーを使用します。

アプリケーションが XA トランザクションに参加するトランザクション MDB と共に使用されており、複数インスタンス・キュー・マネージャーに接続している場合、CCDT にはアクティブとスタンバイ両方のキュー・マネージャー・インスタンスの項目が含まれている必要があります。

これはつまり、Java EE 環境が、トランザクション・リカバリーを実行しなければならない場合に、キュー・マネージャーのログにアクセスできることを意味しています。これは、障害後、どのキュー・マネージャーに環境が再接続するかに関係ありません。

アプリケーションがスタンドアロン・キュー・マネージャーと共に使用されている場合、CCDT には単一の項目のみが含まれている必要があります。これは、アクティベーション・スペックが障害後に常に同じシステムで実行している同じキュー・マネージャーに確実に再接続するようにするためです。

CLIENTRECONNECTOPTIONS プロパティー

Web コンテナーで実行される EJB またはアプリケーションによって使用されるすべての JMS 接続ファクトリーに対して、 **CLIENTRECONNECTOPTIONS** プロパティーを *DISABLED* に設定する必要があります。

使用しているキュー・マネージャーで障害が発生した場合に、新規キュー・マネージャーに自動的に再接続する必要がするアプリケーションは、独自の再接続ロジックを実装する必要があります。 詳しくは、300 ページの『Java EE アプリケーションへの再接続ロジックの実装』 を参照してください。

シナリオ: IBM MQ を使用した WebSphere Application Server
シナリオ: IBM MQ を使用する WebSphere Application Server Liberty プロファイル

クライアント・コンテナー内部で実行するアプリケーション
WebSphere Application Server などの一部の Java EE 環境には、Java SE アプリケーションを実行するために使用できるクライアント・コンテナーが用意されています。

これらの環境内部で実行するアプリケーションは、JMS 接続ファクトリーを使用して IBM MQ キュー・マネージャーに接続します。

クライアント・コンテナー内部で実行するアプリケーションの場合

- **CONNECTIONNAMELIST** および **CCDTURL** が完全にサポートされます。
- **CLIENTRECONNECTOPTIONS** が完全にサポートされます。

CONNECTIONNAMELIST プロパティー

Java EE 環境で JMS 接続用の接続プールが提供されている場合、これが **CONNECTIONNAMELIST** プロパティーの動作にどのように影響するかについては、299 ページの『接続プールでの CONNECTIONNAMELIST または CCDT の使用』 を参照してください。

Java EE 環境が JMS 接続プールを提供しない場合。 アプリケーションは、 Java SE アプリケーションと同じ方法で **CONNECTIONNAMELIST** プロパティーを使用します。

CCDTURL プロパティー

Java EE 環境で JMS 接続の接続プールが提供されている場合、これが **CCDTURL** プロパティーの動作にどのように影響するかについては、299 ページの『接続プールでの CONNECTIONNAMELIST または CCDT の使用』 を参照してください。

Java EE 環境が JMS 接続プールを提供しない場合。 アプリケーションは、 Java SE アプリケーションと同じ方法で **CCDTURL** プロパティーを使用します。

接続プールでの *CONNECTIONNAMELIST* または *CCDT* の使用
一部の Java EE 環境 (WebSphere Application Server など) では、 JMS 接続プールが提供されます。 Java SE アプリケーションの実行に使用できるコンテナー。

Java EE 環境で定義されている接続ファクトリーを使用して接続を作成するアプリケーションは、この接続ファクトリーの接続プールから既存の空き接続を獲得するか、接続プール内に適切な接続がない場合は新しい接続を獲得します。

これは、**CONNECTIONNAMELIST** または **CCDTURL** プロパティーが定義されて接続ファクトリーが構成されている場合、影響を及ぼす可能性があります。

初めて接続ファクトリーを使用して接続を作成するときに、 Java EE 環境は **CONNECTIONNAMELIST** のいずれかを使用します。 または **CCDTURL** を使用して、 IBM MQ システムへの新規接続を作成します。 この接続は、不要になったときに、接続プールに戻され再使用可能になります。

何らかのものが接続ファクトリーから接続を作成する場合、Java EE 環境は **CONNECTIONNAMELIST** または **CCDTURL** プロパティーを使用して新規接続を作成する代わりに、接続プールから接続を戻します。

キュー・マネージャー・インスタンスで障害が発生したときに接続が使用中になっていると、その接続は廃棄されます。 ただし、接続プール内コンテンツは廃棄されない可能性があります。これはつまり、プール内に、既に実行されていないキュー・マネージャーへのる接続が残される可能性があることを意味します。

このような場合、次回接続ファクトリーから接続を作成する要求が出されると、障害が発生したキュー・マネージャーに対する接続が戻されます。 キュー・マネージャーがもう実行していないため、この接続を使用する試みはすべて失敗し、接続が廃棄される原因となります。

Java EE 環境は、接続プールが空である場合にのみ、**CONNECTIONNAMELIST** または **CCDTURL** プロパティーを使用して IBM MQ への新規接続を作成します。

CONNECTIONNAMELIST および CCDT を使用して JMS 接続を作成する方法により、異なる IBM MQ システムへの接続を含む接続プールを使用することもできます。

例えば、次の値に設定された **CONNECTIONNAMELIST** プロパティーで接続ファクトリーが構成されていると想定します。

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

アプリケーションが初めてこの接続ファクトリーからスタンドアロン・キュー・マネージャーへの接続の作成を試みたときに、システム `hostname1(port1)` で実行しているキュー・マネージャーにアクセスで

きなかったと想定します。 その結果、アプリケーションは hostname2(port2) で実行しているキュー・マネージャーに対する接続を使用することになります。

この時点で、別のアプリケーションが起動し、同じ接続ファクトリーから JMS 接続を作成します。 hostname1(port1) のキュー・マネージャーが使用可能になっているので、新しい JMS 接続がこの IBM MQ システムに対して作成され、このアプリケーションに戻されます。

両方のアプリケーションが終了すると、これらのアプリケーションはそれぞれの JMS 接続を閉じ、これらの接続は接続プールに戻されます。

結果として、この時点で接続ファクトリーの接続プールには次の 2 つの JMS 接続が入っていることになります。

- hostname1(port1) で実行しているキュー・マネージャーに対する 1 つの接続
- hostname2(port2) で実行しているキュー・マネージャーに対する 1 つの接続

このことが、トランザクション・リカバリーに関連する問題につながる可能性があります。 Java EE システムがトランザクションをロールバックする必要がある場合、トランザクション・ログに対するアクセス権を持つキュー・マネージャーに接続できなければなりません。

Java EE アプリケーションへの再接続ロジックの実装
キュー・マネージャーが失敗して、自身の再接続論理を実装する必要がある場合に自動的に再接続する、Enterprise JavaBeans および Web ベース・アプリケーション。

以下のオプションで、これを達成する方法に関する詳細を説明します。

アプリケーションの障害対応可能

このアプローチでは、アプリケーションに変更を加える必要はありませんが、接続ファクトリー定義の管理再構成に **CONNECTIONNAMELIST** プロパティーを組み込む必要があります。 ただし、このアプローチでは、呼び出し側が障害を適切に処理できなければなりません。 このことは、接続障害に関連しない MQRC_Q_FULL のような障害に対しても同様に必要である点に注意してください。

このプロセスのコーディング例は、次のとおり。

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
       throws ServletException, IOException {
          try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

上記のコードは、このサーブレットが使用している接続ファクトリーに **CONNECTIONNAMELIST** プロパティーが定義されていることを想定しています。

サーブレットが最初に処理するときに、同じキュー・マネージャーに接続している他のアプリケーションから使用可能なプール接続がないことを前提として、 **CONNECTIONNAMELIST** プロパティーを使用して新規接続が作成されます。

close() 呼び出しの後で接続が解放されると、この接続はプールに戻され、次回サーブレットが実行するときに、**CONNECTIONNAMELIST** を参照することなく、接続障害が起きるまで再使用されます。障害が発生すると、 CONNECTION_ERROR_OCCURRED イベントが生成されます。 このイベントは、プールに対して、障害を発生した接続の破棄を求めるプロンプトを出します。

アプリケーションの次回の実行時には、プール内に使用可能な接続はなく、**CONNECTIONNAMELIST** が使用されて最初の使用可能なキュー・マネージャーに接続します。 キュー・マネージャーのフェイルオーバーが発生した (例えば、障害が一時的なネットワーク障害ではなかった) 場合、 サーブレットはバックアップ・インスタンスが使用可能になったときに、そのインスタンスに接続します。

データベースなどの他のリソースがアプリケーションに関係している場合、アプリケーション・サーバーがトランザクションをロールバックするように指示することが適切な場合があります。

アプリケーション内での再接続の処理

呼び出し側がサーブレットから障害を処理できない場合は、アプリケーション内部で再接続を処理する必要があります。 次の例に示すように、アプリケーション内部で再接続を処理するには、アプリケーションが JNDI から検索した接続ファクトリーをキャッシュに入れて、JMSException (例えば、「JMSCMQ0001: WebSphere MQ 呼び出しは完了コード '2' ('MQCC_FAILED') 理由 '2009' ('MQRC_CONNECTION_BROKEN') で失敗しました」など) を処理できるように、アプリケーションで新規接続を要求する必要があります。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
        // drive reconnection
        setupResources();
    }
  }
```

次の例では、setupResources() は JMS オブジェクトを作成し、即時的でない再接続を処理するためにスリープと再試行のループが組み込まれています。 実際、このメソッドは再接続の試みが多くなるのを防ぎます。 分わかりやすくするため、例では出口条件が省略されている点に注意してください。

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

アプリケーションで再接続を管理する場合は、IBM MQ キュー・マネージャーかバックエンド・サービス (データベースなど) かにかかわらず、他のリソースに、保持している接続を解放することが重要です。 これらの接続は、新しい IBM MQ キュー・マネージャー・インスタンスへの再接続が完了したら、再確立する必要があります。 接続を再確立しないと、再接続が試行される間、アプリケーション・サーバーのリソースが不必要に保持されるため、再利用される前にタイムアウトになる可能性があります。

WorkManager の使用

処理時間が数十秒を上回る長期実行アプリケーション (例えば、バッチ処理など) の場合、WebSphere Application Server WorkManager を使用できます。 WebSphere Application Server のコード・フラグメント例は、以下のとおりです。

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                                 throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

ここで web.xml には、以下のものが含まれます:

```
<resource-ref>
      <description>WorkManager</description>
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

この時点で、バッチが次の処理インターフェースを通じて実装されます。

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
                ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
```

```
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

例えば、大きなメッセージ、低速のネットワーク、大規模なデータベース・アクセス (特に、低速のフェイルオーバーと結び付いた場合) など、バッチ処理の実行に長い時間がかかる場合、サーバーは、以下の例に示すような、ハング・スレッド警告の出力を開始します。

WSVR0605W: スレッド "WorkManager.DefaultWorkManager : 0" (00000035) が 694061 ミリ秒間アクティブで、ハングしている可能性があります。 サーバー内には、ハングしている可能性のあるスレッドが合計 1 本あります。

これらの警告は、バッチのサイズを小さくするか、ハング・スレッドのタイムアウトを長めに設定することで最小限に抑えることができます。 ただし、通常はこの処理を EJB (バッチ送信のため) またはメッセージ駆動型 Bean (コンシュームまたはコンシュームおよび応答のため) 処理に実装することが推奨されます。

アプリケーション管理再接続では実行時エラーを処理する汎用ソリューションが提供されませんが、それでもアプリケーションが接続障害に関連していないエラーを処理しなければならないことに注意してください。

例えば、満杯のキューにメッセージの挿入を試みている (2053 MQRC_Q_FULL)、または無効なセキュリティー資格情報を使用してキュー・マネージャーへの接続を試みている (2035 MQRC_NOT_AUTHORIZED) など。

アプリケーションは、フェイルオーバーの進行中にすぐに使用可能なインスタンスがない場合、2059 MQRC_Q_MGR_NOT_AVAILABLE エラーにも対処しなければなりません。 これは、サイレントに再接続を試みる代わりに、JMS 例外の発生時に報告するアプリケーションによって行います。

IBM MQ classes for JMS オブジェクト・プーリング

Java EE の外部で何らかの接続プーリングを使用すると、全体的な負荷の軽減に役立ちます。負荷が発生する原因の例としては、フレームワークを使用したりクラウド環境にデプロイされたりしたスタンドアロン・アプリケーション、QueueManagers へのクライアント接続の数が増えたことによるアプリケーションとキュー・マネージャーのサーバー統合の増加などが挙げられます。

Java EE プログラミング・モデル内には、使用されるさまざまなオブジェクトのライフサイクルが適切に定義されています。 メッセージ駆動型 Bean (MDB) では最も制約が多く、Servlet ではより多くの自由があります。 そのため、Java EE サーバーで使用可能なプーリング・オプションは、使用されるさまざまなプログラミング・モデルに適したものとなります。

Java SE (または Spring などの別のフレームワーク) を使用すると、プログラミング・モデルは非常に柔軟なものとなります。 そのため、単一のプーリング戦略ですべての状況に対応することはできません。 例えば Spring など、何らかの形式のプーリングが可能なフレームワークを使用できるようになるかどうかを検討する必要があります。

使用するプーリング戦略は、アプリケーションが実行する環境によって異なります。

Java EE 環境でのオブジェクト・プーリング

Java EE アプリケーション・サーバーは、メッセージ駆動型 Bean アプリケーション、Enterprise Java Beans およびサーブレットで使用できる接続プーリング機能を提供します。

WebSphere Application Server は、パフォーマンス向上のために、JMS プロバイダーへの接続のプールを保守します。 アプリケーションが JMS 接続を作成するときに、アプリケーション・サーバーは空き接続プール内に既存の接続がないかどうかを判別します。 既存の接続があれば、その接続がアプリケーションに戻されます。それ以外の場合は、新規接続が作成されます。

304 ページの図 41 に、アクティベーション・スペックとリスナー・ポートの両方について、どのように JMS 接続を確立し、その接続を使用して通常モードでメッセージの宛先をモニターするかを示します。

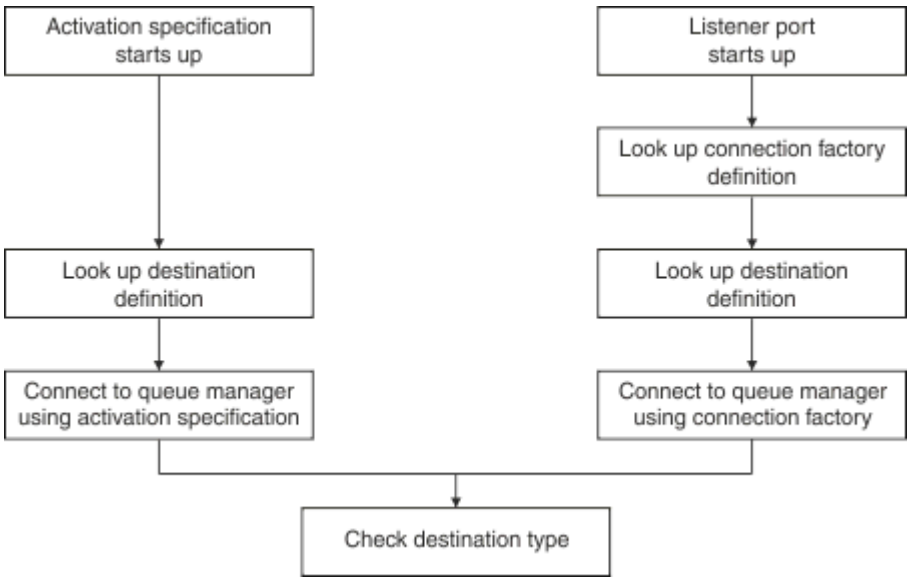


図 *41.* 通常モード

IBM MQ メッセージング・プロバイダーを使用する場合、アウトバウンド・メッセージングを実行するアプリケーション (Enterprise Java Beans およびサーブレットなど)、およびメッセージ駆動型 Bean リスナー・ポート・コンポーネントは、これらの接続プールを使用できます。

IBM MQ メッセージング・プロバイダーのアクティベーション・スペックは、IBM MQ リソース・アダプターが提供する接続プール機能を使用します。 詳しくは、WebSphere MQ リソース・アダプターのプロパティーの構成を参照してください。

308 ページの『接続プールの使用例』では、アウトバウンド・メッセージングを実行するアプリケーション、およびリスナー・ポートが、JMS 接続を作成するときに空きプールを使用する方法について説明します。

310 ページの『空き接続プール保守スレッド』では、アプリケーションまたはリスナー・ポートが接続を終了したときに、これらの接続に何が起きるかを説明します。

312 ページの『プール保守スレッドの例』では、JMS 接続の不整合を防ぐために空き接続プールをクリーンアップする方法について説明します。

WebSphere Application Server では、接続ファクトリーの *maximum connections* プロパティーによって指定される、ファクトリーから作成可能な接続の数に制限があります。 このプロパティーのデフォルト値は 10 であり、これは、ファクトリーから一度に作成できる接続の数が最大 10 個であることを意味します。

各ファクトリーには、1 つの空き接続プールが関連付けられています。 アプリケーション・サーバーの始動時に、空き接続プールは空になっています。 ファクトリーの空きプールに存在可能な接続の最大数も、maximum connections プロパティーで指定されます。

ヒント : JMS 2.0 では、接続ファクトリーは、接続とコンテキストの両方の作成に使用できます。 そのため、接続とコンテキストの両方が混在する接続ファクトリーを接続プールに関連付けることが可能です。 1 つの接続ファクトリーでは、接続だけを作成するか、あるいはコンテキストだけを作成することをお勧めします。 このようにすると、その接続ファクトリーの接続プールには 1 種類のオブジェクトのみが含まれるようになるので、プールの効率が向上します。

WebSphere Application Server での接続プーリングの仕組みについては、JMS 接続用の接続プールの構成を参照してください。 その他のアプリケーション・サーバーについては、該当するアプリケーション・サーバーの資料を参照してください。

接続プールの使用方法

すべての JMS 接続ファクトリーには、1 つの接続プールが関連付けられており、接続プールには JMS 接続が入っていない場合も、入っている場合もあります。 すべての JMS 接続には 1 つの JMS セッション・プ

ールが関連付けられており、すべての JMS セッション・プールには JMS セッションが入っていない場合も、入っている場合もあります。

305 ページの図 42 に、これらのオブジェクトの間の関係を示します。

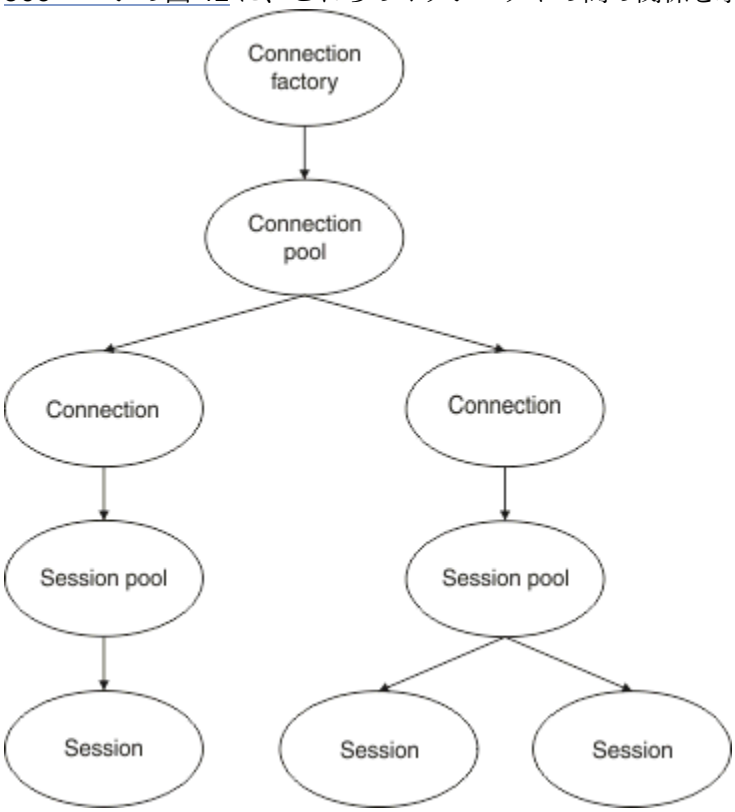


図 *42.* 接続プールとセッション・プール

リスナー・ポートが始動する、またはアウトバウンド・メッセージを実行するアプリケーションがファクトリーを使用して接続を作成する場合、ポートまたはアプリケーションは以下のメソッドのいずれか 1 つを呼び出します。

- **connectionFactory.createConnection()**
- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

WebSphere Application Server 接続マネージャーは、このファクトリーの空きプールから接続の取得を試み、その接続をアプリケーションに返します。

プール内に空き接続がなく、このファクトリーから作成される接続の数がファクトリーの *maximum connections* プロパティーに指定された制限にまだ達していない場合、接続マネージャーはアプリケーションが使用する新規接続を作成します。

ただし、アプリケーションが接続の作成を試み、このファクトリーから作成される接続の数がファクトリーの *maximum connections* プロパティーと既に等しくなっている場合、アプリケーションは接続が使用可能になる (空きプールに戻される) まで待機します。

アプリケーションが待機する時間は、接続プールの *connection timeout* プロパティーで指定され、デフォルト値は 180 秒になります。 180 秒の間に接続が空きプールに戻されると、接続マネージャーはそれをすぐに再度プールから取り出してアプリケーションに渡します。 ただし、タイムアウト期間を超過すると、*ConnectionWaitTimeoutException* がスローされます。

アプリケーションが接続を終了し、以下を呼び出してその接続を閉じた場合:

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

接続は実際に開いたままになり、空きプールに戻されて、別のアプリケーションが再使用できるようになります。 したがって、アプリケーション・サーバー上で JMS アプリケーションが実行されていない場合でも、 WebSphere Application Server と JMS プロバイダーの間の接続を開くことができます。

接続プールの拡張プロパティー
JMS 接続プールの動作を制御するために使用できる拡張プロパティーがいくつかあります。

サージ保護

309 ページの『アウトバウンド・メッセージングを実行するアプリケーションによる接続プールの使用方法』で、connectionFactory.createConnection() を取り込む sendMessage() メソッドの使用に関して説明しています。

ejbCreate() メソッドの一部としてすべて同じ接続ファクトリーから JMS 接続を作成している 50 個の EJB があるケースについて検討します。

これらの Bean すべてが同時に作成され、ファクトリーの空き接続プールに接続が 1 つもない場合、アプリケーション・サーバーは同じ JMS プロバイダーに対して 50 個の JMS 接続を同時に作成しようとします。その結果、WebSphere Application Server と JMS プロバイダーの両方に著しい負荷がかかります。

サージ保護プロパティーを使用して、1 つの接続ファクトリーから一度に作成できる JMS 接続の数を制限し、追加接続の作成をずらすことで、このような状況を回避できます。

一度に作成できる JMS 接続の数の制限は、次の 2 つのプロパティーを使用して行います。

- surge threshold (サージしきい値)
- surge creation interval (サージ作成間隔)

EJB アプリケーションが接続ファクトリーから JMS 接続の作成を試みると、接続マネージャーは作成されている接続の数を調べます。 その数が surge threshold プロパティーの値以下の場合、接続マネージャーは新規接続のオープンを続行します。

ただし、作成される接続の数が surge threshold プロパティーを超えると、接続マネージャーは、surge creation interval プロパティーで指定された期間待機してから、新規接続を作成して開きます。

スタック接続

JMS アプリケーションがその接続を使用して JMS プロバイダーに要求を送信し、プロバイダーが一定の時間内に応答しない場合、 JMS 接続は stuck と見なされます。

WebSphere Application Server は、 stuck JMS 接続を検出する方法を提供します。この機能を使用するには、以下の 3 つのプロパティーを設定する必要があります。

- Stuck Time Timer (スタック時間タイマー)
- Stuck Time (スタック時間)
- Stuck Threshold (スタックしきい値)

312 ページの『プール保守スレッドの例』では、プール保守スレッドが定期的に実行されて接続ファクトリーの空きプールのコンテンツが検査される方法について説明しています。これは、一定期間使用されていない、または存続期間が長すぎる接続を探すことによって行われます。

スタック接続を検出するために、アプリケーション・サーバーは接続ファクトリーから作成されたすべてのアクティブな接続の状態を検査するスタック接続スレッドも管理して、JMS プロバイダーからの応答を待機している接続がないかどうかを調べます。

スタック接続スレッドが実行するタイミングは、Stuck time timer プロパティーで決定されます。 このプロパティーのデフォルト値は 0 であり、これはスタック接続検出が実行されないことを意味します。

スレッドは、応答を待機している接続を検出すると、その待機時間を判別し、その時間と Stuck time プロパティーの値を比較します。

JMS プロバイダーの応答に要する時間が Stuck time プロパティーに指定された値を上回る場合、アプリケーション・サーバーはその JMS 接続にスタックとしてマークを付けます。

例えば、接続ファクトリー jms/CF1 の Stuck time timer プロパティーが 10 に設定され、 Stuck time プロパティーが 15 に設定されているとします。

スタック接続スレッドは 10 秒おきにアクティブになり、jms/CF1 から作成された接続のうち、IBM MQ からの応答を 15 秒を超えて待機しているものがないかどうかを検査します。

EJB が jms/CF1 を使用して IBM MQ への JMS 接続を作成し、 Connection.createSession() を呼び出してその接続を使用して JMS セッションの作成を試行するとします。

ただし、何らかの原因で JMS プロバイダーの要求への応答が阻害されているとします。 マシンがフリーズしている、または JMS プロバイダーで実行しているプロセスでデッドロックが発生しており、新しい作業の処理を阻害している可能性があります。

EJB が Connection.createSession() を呼び出した 10 秒後に、スタック接続タイマーがアクティブになり、jms/CF1 から作成されたアクティブな接続を調べます。

アクティブな接続は、例えば c1 という名前の 1 つのみであるとします。 最初の EJB は、c1 に送信した要求に対する応答を 10 秒間待機しています。これは、Stuck time の値より短いので、スタック接続タイマーはこの接続を無視し、非アクティブになります。

10 秒後、スタック接続スレッドが再度アクティブになり、jms/CF1 のアクティブな接続を調べます。 前回同様、ここには 1 つの接続、c1 のみがあるとします。

最初の EJB が createSession() を呼び出してから、この時点で 20 秒になりますが、EJB はまだ応答を待機しています。 20 秒は Stuck time プロパティーで指定された時間より長いため、滞留接続スレッドは c1 に滞留のマークを付けます。

5 秒後に、ついに IBM MQ から応答があり、最初の EJB に JMS セッションの作成を許可した場合、接続は使用中に戻ります。

アプリケーション・サーバーは、スタック状態にある、接続ファクトリーから作成された JMS 接続の数をカウントします。 アプリケーションがその接続ファクトリーを使用して新しい JMS 接続を作成し、ファクトリーの空きプールに空き接続が存在しない場合、接続マネージャーはスタック接続の数と Stuck threshold プロパティーの値を比較します。

スタック接続の数が Stuck threshold プロパティーに設定された値より少ない場合、接続マネージャーは新しい接続を作成し、それをアプリケーションに渡します。

ただし、滞留接続の数が Stuck threshold プロパティーの値と等しい場合、アプリケーションはリソース例外を受け取ります。

プール・パーティション

WebSphere Application Server は、接続ファクトリーの空き接続プールをパーティション化できる次の 2 つのプロパティーを提供しています。

- Number of free pool partitions は、空き接続プールをいくつのパーティションに分割するかをアプリケーション・サーバーに通知します。
- Free pool distribution table size は、パーティションに索引付けする方法を判別します。

これらのプロパティーは、IBM サポート・センターから変更する旨依頼された場合を除いて、デフォルト値である 0 のままにします。

WebSphere Application Server には Number of shared partitions という名前の追加の接続プール拡張プロパティーが 1 つある点に注意してください。 このプロパティーは、共有接続を保管するために使

用されるパーティションの数を指定します。 ただし、JMS 接続は常に非共有であるため、このプロパティーは適用されません。

接続プールの使用例
メッセージ駆動型 Bean リスナー・ポート・コンポーネント、およびアウトバウンド・メッセージングを実行するアプリケーションでは、JMS 接続プールを使用します。

308 ページの図 43 に、WebSphere Application Server V7.5 および V8.0 での接続プールの動作方法を示します。

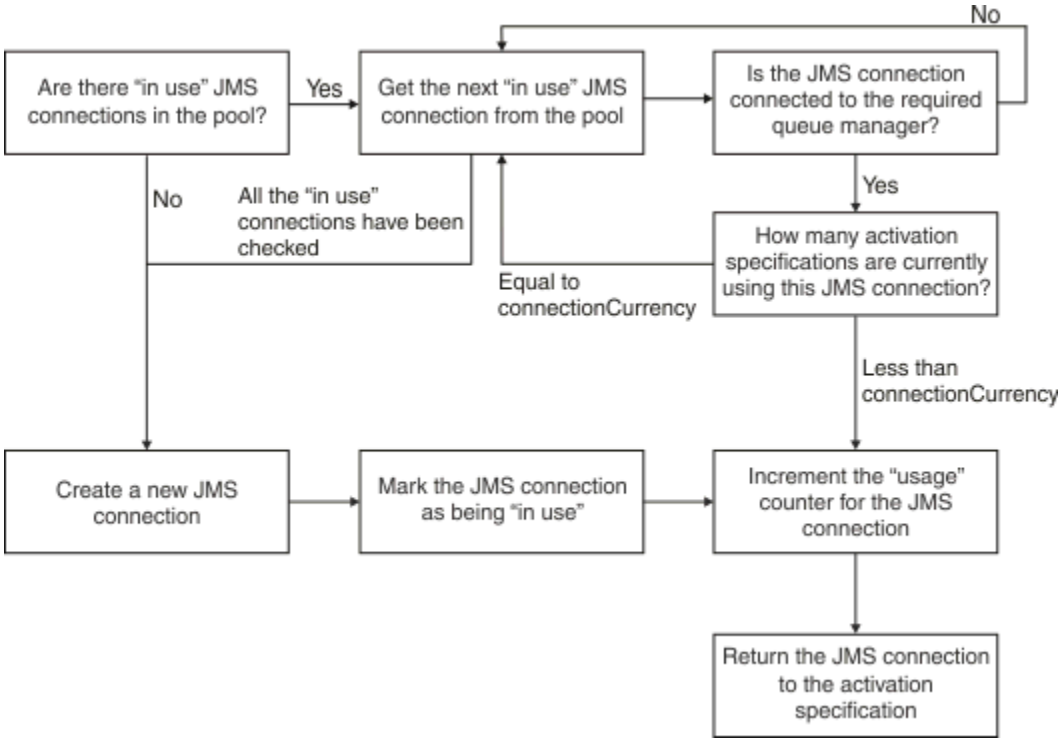


図 *43. WebSphere Application Server V7.5* および *V8.0* - 接続プールの動作方法

308 ページの図 44 に、WebSphere Application Server V8.5 での接続プールの動作方法を示します。

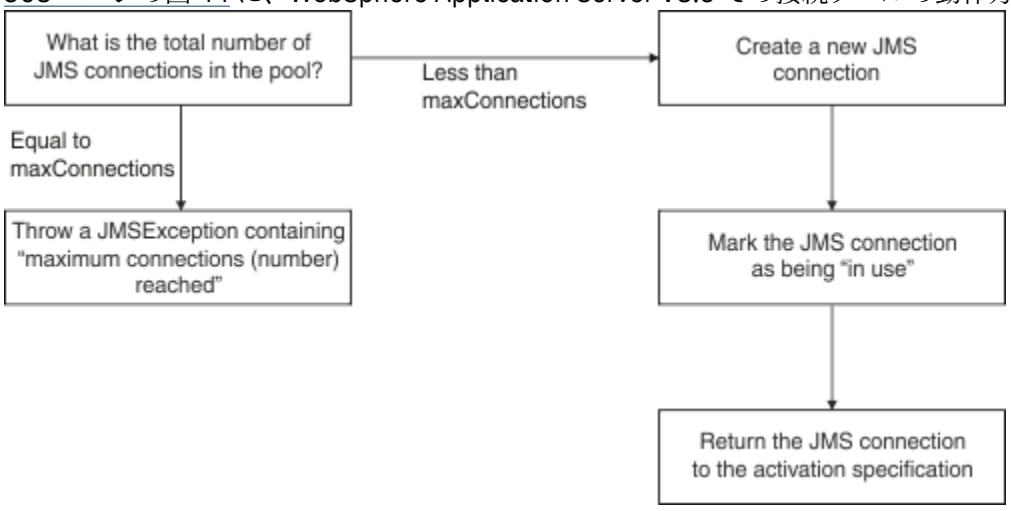


図 *44. WebSphere Application Server V8.5* - 接続プールの動作方法

MDB リスナー・ポートによる接続プールの使用方法

IBM MQ を JMS プロバイダーとして使用する WebSphere Application Server Network Deployment システムに MDB がデプロイされているとします。 MDB は、例えば `jms/CF1` という名前で、*maximum connections* プロパティーが 2 に設定された接続ファクトリー (これはつまり、このファクトリーから一度

に作成できる接続は 2 つだけであることを意味します) を使用しているリスナー・ポートに対してデプロイされています。

リスナー・ポートが始動すると、ポートは jms/CF1 接続ファクトリーを使用して IBM MQ への接続を作成しようとします。

そのために、ポートは接続マネージャーから接続を要求します。 これが、jms/CF1 接続ファクトリーの初回の使用となるので、jms/CF1 空き接続プールに接続はありません。そこで、接続マネージャーが、例えば c1 という名前の新しい接続を作成します。 この接続が、このリスナー・ポートの存続期間中を通じて存在することになる点に注意してください。

ここで WebSphere Application Server 管理コンソールを使用してリスナー・ポートを停止する状況について検討します。 この場合、接続マネージャーが接続を取得し、空きプールに戻します。 しかし、IBM MQ への接続は開いたままになります。

リスナー・ポートを再始動すると、ポートから再び接続マネージャーに、キュー・マネージャーへの接続が依頼されます。 この時点で、空きプールには接続 (c1) が既存なので、接続マネージャーがこの接続をプールから取得し、リスナー・ポートが使用できるようにします。

ここで、別のリスナー・ポートを使用している 2 番目の MDB がアプリケーション・サーバーにデプロイされているとします。

次に、同じく jms/CF1 接続ファクトリーを使用するように構成されている 3 番目のリスナー・ポートを始動するとします。 3 番目のリスナー・ポートは、接続マネージャーからの接続を要求し、接続マネージャーは jms/CF1 の空きプールを探し、それが空であることを検出します。 次に、jms/CF1 ファクトリーで既に作成された接続の数を検査します。

jms/CF1 の最大接続プロパティーは 2 に設定されており、このファクトリーからは既に 2 つの接続が作成されているので、接続マネージャーは接続が使用可能になるまで 180 秒 (接続タイムアウト・プロパティーのデフォルト値) 待ちます。

ただし、最初のリスナー・ポートを停止した場合、その接続 c1 は jms/CF1 の空きプールに入れられます。接続マネージャーはこの接続を取得し、3 番目のリスナーに付与します。

ここで、最初のリスナーの再始動を試みた場合、他のリスナー・ポートのいずれか 1 つが停止するまで待たなければ、再始動できません。 実行中のリスナー・ポートが 180 秒以内に停止しない場合、最初のリスナーは ConnectionWaitTimeoutException エラーを受け取り、停止します。

アウトバウンド・メッセージングを実行するアプリケーションによる接続プールの使用方法

このオプションでは、例えば EJB1 という名前の単一の EJB がアプリケーション・サーバーにインストールされているとします。 この Bean は、sendMessage() という名前のメソッドを以下のように実装しています。

- connectionFactory.createConnection()を使用して、ファクトリー jms/CF1 から IBM MQ への JMS 接続を作成します。
- 接続から JMS セッションを作成します。
- セッションからメッセージ・プロデューサーを作成します。
- メッセージを送信します。
- プロデューサーを閉じます。
- セッションを閉じます。
- connection.close() を呼び出して接続を閉じます。

ファクトリー jms/CF1 の空きプールが空であると想定します。 EJB の初回呼び出し時に、Bean は IBM MQ への接続をファクトリー jms/CF1 から作成しようと試みます。 ファクトリーの空きプールは空なので、接続マネージャーは新しい接続を作成して EJB1 に与えます。

このメソッドは、終了直前に connection.close() を呼び出します。 接続マネージャーは、c1 を閉じる代わりに、接続を取得し jms/CF1 の空きプールに入れます。

次に sendMessage() が呼び出されると、 connectionFactory.createConnection() メソッドは c1 をアプリケーションに返します。

EJB の 2 番目のインスタンスを、最初のインスタンスと同時に実行しているとします。 両方のインスタンスが sendMessage() を呼び出している場合、2 つの接続が jms/CF1 接続ファクトリーから作成されます。

ここで、Bean の 3 番目のインスタンスが作成されるとします。 3 番目の Bean が sendMessage() を呼び出すと、このメソッドは connectionFactory.createConnection() を呼び出して jms/CF1 から接続を作成します。

ただし、現在 jms/CF1 からは、このファクトリーの最大接続の値と同じ 2 つの接続が作成されています。したがって、createConnection() メソッドは、接続が使用可能になるのを 180 秒 (接続タイムアウト・プロパティーのデフォルト値) 待ちます。

ただし、最初の EJB の sendMessage() メソッドが connection.close() を呼び出して終了した場合、それが使用していた接続 c1 は空き接続プールに戻されます。 接続マネージャーはその接続を空きプールから取得して、3 番目の EJB に付与します。 その Bean から connectionFactory.createConnection() が呼び出され、戻ってきたら、sendMessage() メソッドが完了できます。

同じ接続プールを使用している MDB リスナー・ポートおよび EJB

前の 2 つの例は、リスナー・ポートおよび EJB がそれぞれ独立して接続プールを使用する方法を示しています。 しかし、リスナー・ポートと EJB の両方を同じアプリケーション・サーバー内で実行させて、同じ接続ファクトリーを使用して JMS 接続を作成することが可能です。

この状況の影響について検討する必要があります。

覚えておく必要がある重要な点は、接続ファクトリーがリスナー・ポートと EJB の間で共有されるということです。

例えば、リスナーと EJB が同時に実行しているとします。 いずれも jms/CF1 接続ファクトリーを使用しており、これはファクトリーの最大接続プロパティーで指定された接続制限に達していることを意味します。

別のリスナー・ポート、または EJB の別のインスタンスのいずれかを始動しようとすると、接続が jms/CF1 の空き接続プールに戻されるまで、いずれも待機する必要があります。

空き接続プール保守スレッド
各空き接続プールには、空きプールをモニターし、その中の接続が有効な状態を保っていることを確認するプール保守スレッドが関連付けられています。

プール保守スレッドが、空きプール内の接続を破棄する必要があると判断した場合、スレッドは IBM MQ への JMS 接続を物理的に閉じます。

プール保守スレッドの動作方法

プール保守スレッドの動作は、接続プールの以下の 4 つのプロパティーの値によって決まります。

Aged timeout
開いたままの状態の接続時間。

Minimum connections
接続マネージャーが接続ファクトリーの空きプール内に保持する接続の最小数。

Reap time
プール保守スレッドの実行頻度。

Unused timeout
接続が閉じられるまでに空きプール内に留まる時間。

デフォルトでは、プール保守スレッドは 180 秒間隔で実行されますが、この値は、接続プールの **Reap time** プロパティーを設定することで変更できます。

保守スレッドは、プール内の各接続について、その接続がどのくらいの時間プール内に留まっているか、およびその接続が作成され、最後に使用されてからどのくらいの時間が経過したかを検査します。

接続が接続プールの **Unused timeout** プロパティーの値より長い期間使用されていない場合、保守スレッドは、現在フリー・プール内にある接続の数を検査します。 この数に応じて、以下のようになります。

- 接続数が **Minimum connections** の値よりも多い場合、接続マネージャーは接続を閉じます。
- 接続数が **Minimum connections** の値と等しい場合、接続は閉じられず、空きプール内に残されます。

Minimum connections プロパティーのデフォルト値は *1* であり、これはつまり、パフォーマンス上の理由から、接続マネージャーが空きプール内に少なくとも 1 つの接続を常に保持しようと試みることを意味しています。

Unused timeout プロパティーのデフォルト値は 1800 秒です。 デフォルトでは、接続が空きプールに戻され 1800 秒以上使用されない状態が続くと、その接続は閉じられます。その接続を閉じたとしても、空きプール内には少なくとも 1 つの接続が残されます。

この手順により、未使用の接続が不整合になることを防ぎます。 この機能をオフにするには、**Unused timeout** プロパティーを 0 に設定します。

接続が空きプール内にあり、作成されてから経過した時間が接続プールの **Aged timeout** プロパティーの値よりも長い場合、その接続は最後に使用してからの経過時間に関係なく閉じられます。

デフォルトでは、**Aged timeout** プロパティーは 0 に設定されます。これは、保守スレッドがこの検査を実行しないことを意味します。 **Aged timeout** プロパティーよりも長い時間存続している接続は、空きプール内に残される接続の数に関係なく廃棄されます。 この状況に、**Minimum connections** プロパティーが影響を与えることはない点に注意してください。

プール保守スレッドの無効化

上記の説明から、プール保守スレッドがアクティブになるとかなりの量の作業を行うことがわかります。これは特に、接続ファクトリーの空きプール内に多数の接続がある場合に顕著です。

例えば、3 つの JMS 接続ファクトリーがあり、各ファクトリーの **Maximum connections** プロパティーが 10 に設定されているとします。 180 秒おきに、3 つのプール保守スレッドがアクティブになり、各接続ファクトリーの空きプールを個別にスキャンします。 空きプールに多数の接続がある場合、保守スレッドは多くの作業を行うことになり、そのことがパフォーマンスに重大な影響を与えます。

個々の空き接続プールの **Reap time** プロパティーを 0 に設定することで、それらのプール保守スレッドを無効にすることができます。

保守スレッドを無効にするということは、接続が **Unused timeout** を超過しても閉じられないことを意味します。 ただし、**Aged timeout** を超えた場合には、接続が閉じられます。

アプリケーションが接続を終了すると、接続マネージャーは接続の存続期間を確認します。その期間が **Aged timeout** プロパティーの値よりも長い場合、接続マネージャーは接続を空きプールに戻すのではなく、接続を閉じます。

Aged timeout のトランザクションへの影響

前のセクションで説明したように、**Aged timeout** プロパティーは、JMS プロバイダーへの接続が、接続マネージャーによって閉じられるまで、開いたままの状態を維持する時間を指定します。

Aged timeout プロパティーのデフォルト値は 0 であり、これは、接続が古くなりすぎても閉じられることがないことを意味します。 **Aged timeout** を使用可能にすると、EJB 内で JMS を使用する場合にトランザクションに影響を与える可能性があるため、**Aged timeout** プロパティーはこの値のままにしておく必要があります。

JMS では、トランザクションの単位は JMS セッション であり、これは JMS 接続 より作成されます。 トランザクションに enlist されるのはこの JMS セッション であり、JMS 接続 ではありません。

アプリケーション・サーバーの設計により、**Aged timeout** が経過しているため、その接続から作成された JMS セッションがトランザクションに関与している場合でも、 JMS 接続を閉じることができます。

JMS 接続を閉じると、JMS 仕様で説明しているように、JMS セッションの未処理のトランザクション作業がロールバックされます。 ただし、アプリケーション・サーバーは、接続から作成された JMS セッションが有効ではなくなったことを検知しません。 サーバーがそのセッションを使用してトランザクションのコミットまたはロールバックを試みると、IllegalStateException が発生します。

重要 : EJB 内からの JMS 接続で **Aged timeout** を使用する場合は、 JMS 操作を実行する EJB メソッドが終了する前に、 JMS セッションで JMS 作業が明示的にコミットされていることを確認してください。

プール保守スレッドの例
Enterprise JavaBean (EJB) の例を使用して、プール保守スレッドがどのように機能するかを理解します。必要なのは空きプールで接続を取得する方法のみなので、Message Driven Beans (MDB) とリスナー・ポートも使用できる点に注意してください。

sendMessage() メソッドの詳細については、309 ページの『アウトバウンド・メッセージングを実行するアプリケーションによる接続プールの使用方法』を参照してください。

以下の値で接続ファクトリーを構成しています。

- デフォルト値 180 秒の **Reap time**
- デフォルト値 0 秒の **Aged timeout**
- **Unused timeout** は 300 秒に設定されます。

アプリケーション・サーバーの始動後、sendMessage() メソッドが呼び出されます。

このメソッドは、ファクトリー jms/CF1 を使用して、例えば c1 という名前の接続を作成し、そのファクトリーを使用してメッセージを送信し、connection.close() を呼び出します。これにより、c1 が空きプールに入れられます。

180 秒後、プール保守スレッドが始動し、jms/CF1 空き接続プールを調べます。 プール内には空き接続 c1 があるので、保守スレッドは接続がプールに戻された時刻を調べ、それを現在時刻と比較します。

接続が空きプールに入れられてから 180 秒が経過しています。これは、jms/CF1 の **Unused timeout** プロパティーの値よりも短い時間です。 したがって、保守スレッドは接続をそのままにします。

180 秒後、プール保守スレッドが再実行されます。 保守スレッドは接続 c1 を検出し、その接続がプール内に 360 秒入っていることを判別します。 これは、設定された **Unused timeout** 値よりも長いので、接続マネージャーはこの接続を閉じます。

sendMessage() メソッドを再実行すると、アプリケーションが connectionFactory.createConnection() を呼び出すときに、接続マネージャーが IBM MQ への新しい接続を作成します。これは、接続ファクトリーの空き接続プールが空であるためです。

上記の例は、**Aged timeout** プロパティーがゼロに設定されている場合に、失効した接続を防止するために保守スレッドが **Reap time** プロパティーと **Unused timeout** プロパティーをどのように使用するかを示しています。

Aged timeout プロパティーはどのように動作するのでしょうか。

以下の例では、次のように設定されているとしています。

- 300 秒に設定された **Aged timeout** プロパティー
- 0 秒に設定された **Unused timeout** プロパティー

sendMessage() メソッドを呼び出すと、このメソッドは jms/CF1 接続ファクトリーからの接続の作成を試みます。

このファクトリーの空きプールは空なので、接続マネージャーは新しい接続 c1 を作成して、それをアプリケーションに戻します。 sendMessage() が connection.close() を呼び出すと、c1 は空き接続プールに戻されます。

180 秒後、プール保守スレッドが実行されます。 このスレッドは、空き接続プール内で c1 を見つけ、それがどのくらい前に作成されたかを調べます。 この接続は 180 秒存続しており、**Aged timeout** よりも短い時間です。そのため、プール保守スレッドはその接続をそのままにしてスリープ状態に戻ります。

60 秒後、sendMessage() が再び呼び出されます。 今回は、このメソッドが connectionFactory.createConnection() を呼び出すと、接続マネージャーが jms/CF1 の空きプール内に使用可能な接続 c1 があることを検出します。 接続マネージャーは接続 c1 を空きプールから取得して、アプリケーションに付与します。

この接続は、sendMessage() が終了すると空きプールに戻されます。 120 秒後、プール保守スレッドは再度起動し、jms/CF1 の空きプールのコンテンツをスキャンし c1 を検出します。

接続は 120 秒前に使用されているだけですが、存続期間は合計で 360 秒になり、**Aged timeout** プロパティーで設定した値 300 秒よりも長くなるので、プール保守スレッドはこの接続を閉じます。

最小接続プロパティーがプール保守スレッドに与える影響

308 ページの『MDB リスナー・ポートによる接続プールの使用方法』 の例を再度使用し、アプリケーション・サーバーに、それぞれ別のリスナー・ポートを使用している 2 つの MDB がデプロイされているとします。

リスナー・ポートは両方とも、以下の値で構成した jms/CF1 接続ファクトリーを使用するように構成されています。

- 120 秒に設定された **Unused timeout** プロパティー
- 180 秒に設定された **Reap time** プロパティー
- 1 に設定された **Minimum connections** プロパティー

最初のリスナーが停止され、その接続 c1 が空きプールに入れられているとします。 180 秒後、プール保守スレッドが起動し、jms/CF1 の空きプールのコンテンツをスキャンし、c1 が接続ファクトリーの **Unused timeout** プロパティーの値よりも長い時間空きプールに入っていることを検出します。

しかし、c1 を閉じる前に、プール保守スレッドは、この接続が廃棄された場合にプールに残される接続の数を調べます。 c1 は空き接続プール内の唯一の接続であるため、これを閉じた場合、空きプール内に残される接続の数が **Minimum connections** に設定した値よりも少なくなるので、接続マネージャーはこの接続を閉じません。

次に、2 番目のリスナーが停止しているとします。 空き接続プール内には、現在 2 つの空き接続 c1 と c2 が入っています。

180 秒後、プール保守スレッドが再実行されます。 この時点までに、c1 は 360 秒、c2 は 180 秒空き接続プールに入っています。

プール保守スレッドは c1 を検査してこれが **Unused timeout** プロパティーの値よりも長い時間プールに入っていることを検出します。

スレッドは次に、空きプール内の接続の数を検査し、その数と **Minimum connections** プロパティーの値を比較します。 プールには 2 つの接続が入っており、**Minimum connections** は 1 に設定されているので、接続マネージャーは c1 を閉じます。

保守スレッドは次に c2 を調べます。 これは、 **Unused timeout** プロパティーの値よりも長い間、空き接続プールにも存在しています。 ただし、 c2 を閉じると、空き接続プールは設定された最小接続数未満のままになるため、接続マネージャーは c2 のみのままになります。

JMS 接続および *IBM MQ*
IBM MQ を JMS プロバイダーとして使用することに関する情報。

バインディング・トランスポートの使用

接続ファクトリーがバインディング・トランスポートを使用するように構成されている場合、すべての JMS 接続は、 IBM MQ との会話 (**hconn** とも呼ばれる) を確立します。 会話では、プロセス間通信 (または共有メモリー) を使用してキュー・マネージャーと通信します。

クライアント・トランスポートの使用

IBM MQ メッセージング・プロバイダー接続ファクトリーがクライアント・トランスポートを使用するように構成されている場合、そのファクトリーから作成されるすべての接続は、 IBM MQ に対して新しい会話 (**hconn** とも呼ばれる) を確立します。

IBM MQ メッセージング・プロバイダー通常モードを使用してキュー・マネージャーに接続する接続ファクトリーの場合、接続ファクトリーから作成される複数の JMS 接続が IBM MQ への TCP/IP 接続を共有することが可能です。 詳しくは、316 ページの『IBM MQ classes for JMS における TCP/IP 接続の共用』を参照してください。

JMS 接続が任意の時点で使用するクライアント・チャネルの最大数を決定するため、同じキュー・マネージャーをポイントするすべての接続ファクトリーの *Maximum connections* プロパティーの値を加算します。

例えば、 `jms/CF1` と `jms/CF2` という 2 つの接続ファクトリーがあり、同じ IBM MQ チャネルを使用して同じ IBM MQ キュー・マネージャーに接続するように構成されているとします。

これらのファクトリーは、デフォルトの接続プール・プロパティーを使用しており、これはつまり *Maximum connections* が 10 に設定されているということを意味します。 すべての接続が jms/CF1 および jms/CF2 の両方から同時に使用されている場合、アプリケーション・サーバーと IBM MQ の間には 20 の会話があることになります。

接続ファクトリーが IBM MQ メッセージング・プロバイダーの通常モードを使用してキュー・マネージャーに接続している場合、これらの接続ファクトリーのアプリケーション・サーバーとキュー・マネージャーの間で存在可能な TCP/IP 接続の最大数は、次のようになります。

```
20/the value of SHARECNV for the IBM MQ channel
```

接続ファクトリーが IBM MQ メッセージング・プロバイダーの移行モードを使用して接続するように構成されている場合、これらの接続ファクトリーのアプリケーション・サーバーと IBM MQ の間の TCP/IP 接続の最大数は 20 (2 つのファクトリーの接続プール内の JMS 接続ごとに 1 つずつ) となります。

関連概念

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

Java SE 環境でのオブジェクト・プーリング
Java SE (または Spring などの別のフレームワーク) を使用すると、プログラミング・モデルは非常に柔軟なものとなります。 そのため、単一のプーリング戦略ですべての状況に対応することはできません。 例えば Spring など、何らかの形式のプーリングが可能なフレームワークを使用できるかどうかを検討する必要があります。

使用できない場合、アプリケーション・ロジックで対処できる場合があります。 アプリケーション自体がどれほど複雑であるかを検討してください。 接続性からメッセージング・システムまで、アプリケーションとその要件を理解することをお勧めします。 また、基本的な JMS API に対する独自のラッパー・コード内でアプリケーションが記述されることもよくあります。

これはとても賢明なアプローチで、複雑さを隠蔽する働きを期待できますが、問題が生じる可能性があることにも注意してください。 例えば、頻繁に呼び出される汎用の `getMessage()` メソッドは、コンシューマーを開いて閉じるだけのものではありません。

以下の点を考慮する必要があります。

- アプリケーションが IBM MQ にアクセスする必要のある時間の長さは? 常時ですか、またはときどきですか。
- メッセージを送信する頻度は? 頻度が小さいほど、IBM MQ への単一の接続を共有できる数が多くなります。

- 接続の切断の例外は、通常はプールされた接続を再作成する必要を示すサインです。 以下についても考慮してください。
  - セキュリティー例外またはホストの使用不可
  - キュー・フル例外
- 接続の切断の例外が発生した場合、プール内にあるフリーな他の接続はどうなりますか? それらを閉じてから、再作成する必要はありますか?
- 例えば、TLS が使用されている場合、単一の接続をいつまで開いたままにしておきますか?
- プールされた接続を識別できるようにして、キュー・マネージャー管理者がその接続を見つけてトラック・バックできるようにする方法。

すべての JMS オブジェクトをプーリングのために検討して、いつでも可能になったときに、そのオブジェクトをプールする必要があります。 以下のオブジェクトがあります。

- JMS の接続
- Session
- コンテキスト
- 異なるタイプすべてのプロデューサーとコンシューマー

クライアント・トランスポートを使用するとき、JMS の接続、セッション、およびコンテキストは、 IBM MQ キュー・マネージャーと通信する際にソケットを使用します。 これらのオブジェクトをプールすると、キュー・マネージャーに着信する IBM MQ 接続 (hConns) の数が節約され、 チャネル・インスタンスの数が縮小します。

キュー・マネージャーへのバインディング・トランスポートを使用すると、ネットワーキング層が完全に除去されます。 ただし、多くのアプリケーションではクライアント・トランスポートを使用して、使用可能性が高くワークロード・バランシングに優れた構成を提供しています。

JMS のプロデューサーとコンシューマーは、キュー・マネージャーで宛先を開きます。 開かれるキューやトピックの数が少ない場合、アプリケーションの複数のパーツでこれらのオブジェクトが使用されていれば、それらをプールすることは役立ちます。

IBM MQ の視点からは、このプロセスによって一連の MQOPEN と MQCLOSE の操作が節約されます。

接続、セッション、およびコンテキスト

これらのオブジェクトはすべて、キュー・マネージャーへの IBM MQ 接続ハンドルをカプセル化します。 これらは `ConnectionFactory` から生成されます。 アプリケーションにロジックを追加して、単一の接続ファクトリーから作成された接続や他のオブジェクトの数を特定の数に制限できます。

アプリケーションで簡単なデータ構造体を使用して、作成された接続が含まれるようにすることができます。 これらのいずれかのデータ構造体を使用する必要のあるアプリケーション・コードは、使用するオブジェクトをチェックアウト することができます。

以下の点に注意してください。

- 接続をいつプールから除去しますか? 一般的には、接続に例外リスナーを作成します。 例外を処理するためにそのリスナーが呼び出されると、接続およびその接続から作成されたすべてのセッションを再作成することが必要になります。
- ワークロード・バランシングのために CCDT が使用される場合、接続は異なるキュー・マネージャーに向かうことがあります。 これはプーリングの要件に適用できる場合があります。

JMS 仕様によれば、 複数のスレッドが 1 つのセッションまたはコンテキストに同時にアクセスすると、プログラミング・エラーになることに注意してください。 確かに、IBM MQ JMS コードでは、スレッドが厳格に取り扱われます。 ただし、アプリケーションにロジックを追加して、1 つのスレッドによって一度に 1 つのセッションまたはコンテキスト・オブジェクトだけが使用されるようにする必要があります。

プロデューサーとコンシューマー

作成された各プロデューサーとコンシューマーは、キュー・マネージャーで宛先を開きます。 さまざまなタスクで同じ宛先が使用される場合、コンシューマーまたはプロデューサーのオブジェクトを開いたままにしておくのは妥当なことです。 すべての作業が完了したときにのみオブジェクトを閉じます。

宛先を開くことや閉じることは短時間の操作ですが、頻繁に行われる場合には所要時間が累積して長くなることがあります。

これらのオブジェクトは、適用範囲がそれらの作成されたセッションまたはコンテキスト内なので、その適用範囲内に保持する必要があります。 一般には、これを簡単に行うことができるようにアプリケーションが記述されています。

モニター

アプリケーションはそのオブジェクト・プールをどのようにモニターしますか? その答えは、実装されたプーリングのソリューションの複雑さに大きく依存します。

JavaEE プーリング実装を検討する場合、以下のような多数のオプションがあります。

- プールの現行サイズ
- そこでのオブジェクトの経過時間
- プールのクリーニング
- 接続の更新

再使用された単一のセッションがキュー・マネージャーに表示される方法も検討する必要があります。 アプリケーションを識別する接続ファクトリー・プロパティー (appName など) があり、役立つことがあります。

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

IBM MQ classes for JMS における *TCP/IP* 接続の共用
MQI チャネルの複数インスタンスが、単一の TCP/IP 接続を共用するようにできます。

同じ Java ランタイム環境内で実行され、IBM MQ classes for JMS または IBM MQ リソース・アダプターを使用して CLIENT トランスポートを使用してキュー・マネージャーに接続するアプリケーションを作成して、チャネル・インスタンスを共有することができます。

1 より大きい値に設定された **SHARECNV** パラメーターを使用してチャネルが定義されている場合、その数の会話が 1 つのチャネル・インスタンスを共有することができます。 接続ファクトリーまたはアクティベーション・スペックでこの機能を使用できるようにするには、 **SHARECONVALLOWED** プロパティーを はい に設定します。

JMS アプリケーションで作成されたすべての JMS 接続および JMS セッションは、キュー・マネージャーとの独自の会話を作成します。

アクティベーション・スペックが開始されると、IBM MQ リソース・アダプターは、アクティベーション・スペックで使用するキュー・マネージャーとの会話を開始します。 アクティベーション・スペックに関連付けられているサーバー・セッション・プール内の各サーバー・セッションでも、キュー・マネージャーとの会話が開始されます。

SHARECNV 属性は、接続を共有するためのベスト・エフォート・アプローチです。 したがって、0 より大きい **SHARECNV** 値が IBM MQ classes for JMS で使用される場合、新しい接続要求が既に確立されている接続を常に共有することは保証されません。

TCP/IP 接続の共有方法

TCP/IP 接続の共用には、以下の 2 つの方法があります。

GLOBAL 戦略

この方針は、TCP/IP 接続を共用するためのデフォルトの方針です。 JMS 接続またはセッションは、適切な TCP/IP 接続で会話を使用できます。 適合性は、ホスト・アドレス、ポート番号、ユーザー ID とパスワード、TLS/SSL パラメーターなどの要因によって決定されます。

TCP/IP 接続を共用するこの方法により、使用中のチャネル・インスタンスの数は最小限に抑えられますが、TCP/IP 接続のグローバル・プールへのアクセスの競合が発生することになります。

CONNECTION 戦略

この方法では、チャネル・インスタンスは、関連する JMS オブジェクト間でのみ共有されます。 具体的には、JMS 接続が作成されると、その接続用にチャネル・インスタンスが作成され、そのチャネル・インスタンスでの追加の会話は、その JMS 接続によって作成された JMS セッションでのみ使用可能になります。

SHARECNV 属性で指定した数より多くの会話が作成されると、新しいチャネル・インスタンスが作成されます。このチャネル・インスタンスは、元の JMS 接続によって作成された JMS セッションでのみ使用できます。

チャネル・インスタンスを共有するこの方法により、会話の競合を減らすことができますが、代わりにチャネル・インスタンスを大幅に増やす必要が生じる可能性があります。

チャネル・インスタンス共有戦略の明示的な指定

V 9.3.2

デフォルトでは、アプリケーションが再接続可能でない場合は GLOBAL 戦略が使用されます。 再接続可能なアプリケーションは、常に CONNECTION 戦略を使用します。

IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用するアプリケーションの場合、アプリケーション全体で再接続不可能なアプリケーションに対して CONNECTION 戦略を有効にすることができます。 CONNECTION 戦略を有効にするには、システム・プロパティー `com.ibm.mq.jms.channel.sharing` を値 `CONNECTION` に設定します。 この値には大/小文字の区別がなく、 `CONNECTION` 以外の値は無視されます。

システム・プロパティー `com.ibm.mq.jms.channel.sharing` は、以下のいずれかの方法で設定できます。

- 「-D」コマンド行オプションを使用して、JVM 初期化の一部としてプロパティーを設定します。

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- `System.setProperty()` を使用して、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用する前にプロパティーを設定します。

GLOBAL 共有戦略のチャネル・インスタンス数の計算

アプリケーションで作成されるチャネル・インスタンスの最大数を決定するには、以下の公式を使用します。

アクティベーション・スペック

チャネル・インスタンスの数 = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

ここで、 最大 *PoolDepth_value* は **maxPoolDepth** プロパティーの値、 *SHARECNV_*値 はアクティベーション・スペックによって使用されるチャネルの **SHARECNV** プロパティーの値です。

その他の JMS アプリケーション

チャネル・インスタンスの数 = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

ここで、*jms_connections* はアプリケーションによって作成された接続の数、*jms_sessions* はアプリケーションによって作成された JMS セッションの数、 *SHARECNV_value* はアクティベーション・スペックによって使用されるチャネルの **SHARECNV** プロパティーの値です。

CONNECTION 共有戦略のチャネル・インスタンス数の計算

チャネル・インスタンスの数は、アプリケーション内の JMS 接続間での JMS セッションの分散によって異なります。

JMS 接続に対して 1 つの会話を許可し、その JMS 接続の下の JMS セッションごとに 1 つの会話を許可してから、 **SHARECNV** 値で除算して端数を切り上げます。 この計算により、その JMS 接続に必要なチャネル・インスタンスが得られます。

アクティベーション・スペックにも同じ原則を適用できます。 アクティベーション・スペックを JMS 接続と見なし、 **maxPoolDepth** プロパティーを JMS セッションの数と見なします。

例

以下の例は、アプリケーションが IBM MQ classes for JMS または IBM MQ リソース・アダプターを使用して、キュー・マネージャーに作成するチャネル・インスタンスの数を計算するために公式を使用する方法を示しています。

JMS アプリケーション実例
JMS アプリケーション接続では、CLIENT トランスポートを使用してキュー・マネージャーに接続し、1 つの JMS 接続と 3 つの JMS セッションを作成します。 アプリケーションがキュー・マネージャーに接続するために使用するチャネルの **SHARECNV** プロパティーは、値 10 に設定されています。 アプリケーションの実行時には、アプリケーションとキュー・マネージャーの間に存在する会話は 4 つあり、チャネル・インスタンスは 1 つあります。 4 つの会話はすべてチャネル・インスタンスを共用します。

アクティベーション・スペックの例
アクティベーション・スペックでは、CLIENT トランスポートを使用してキュー・マネージャーに接続します。 アクティベーション・スペックは、 **maxPoolDepth** プロパティーを 10 に設定して構成されます。 アクティベーション・スペックが使用するように構成されているチャネルでは、 **SHARECNV** プロパティーが 10 に設定されています。 アクティベーション・スペックが実行され、同時に 10 件のメッセージが処理されている場合、アクティベーション・スペックとキュー・マネージャーの間の会話の数は 11 (10 個の会話はサーバー・セッション用で、1 個はアクティベーション・スペック用です) となります。 アクティベーション・スペックで使用されるチャネル・インスタンスの数は 2 です。

アクティベーション・スペックの例
アクティベーション・スペックでは、CLIENT トランスポートを使用してキュー・マネージャーに接続します。 アクティベーション・スペックは、 **maxPoolDepth** プロパティーを 5 に設定して構成されます。 アクティベーション・スペックが使用するように構成されているチャネルの **SHARECNV** プロパティーが 0 に設定されている。 アクティベーション・スペックが実行されていて、5 つのメッセージを同時に処理している場合、アクティベーション・スペックとキュー・マネージャーの間の会話の数は 6 (サーバー・セッションの場合は 5、アクティベーション・スペックの場合は 1) になります。 アクティベーション・スペックによって使用されるチャネル・インスタンスの数は 6 です。チャネルの **SHARECNV** プロパティーが 0 に設定されているため、会話ごとに独自のチャネル・インスタンスが使用されます。

関連タスク
501 ページの『WebSphere Application Server から IBM MQ に対して作成される TCP/IP 接続の数の決定』
共用会話機能を使用して、複数の会話で MQI チャネル・インスタンス (TCP/IP 接続としても知られる) を共用できるようになります。

IBM MQ classes for JMS でのクライアント接続を受け入れるためのポート範囲の指定
LOCALADDRESS プロパティーを使用して、アプリケーションがバインドできるポート範囲を指定します。

IBM MQ classes for JMS アプリケーションがクライアント・モードで IBM MQ キュー・マネージャーに接続しようとした場合、ファイアウォールは指定されたポートまたはポートの範囲から発生する接続のみを許可します。 この場合、ConnectionFactory、QueueConnectionFactory または TopicConnectionFactory オブジェクトの LOCALADDRESS プロパティーを使用して、アプリケーションがバインドできるポート、またはポートの範囲を指定することができます。

LOCALADDRESS プロパティーは、IBM MQ JMS 管理ツールを使うか、または JMS アプリケーションで setLocalAddress() メソッドを呼び出すことによって設定できます。 アプリケーション内からプロパティーを設定する例を以下に示します。

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

アプリケーションがキュー・マネージャーに接続すると、アプリケーションはローカル IP アドレスおよび 192.0.2.0(2000) から 192.0.2.0(3000) の範囲にあるポート番号にバインドされます。

複数のネットワーク・インターフェースが存在するシステムでは、LOCALADDRESS プロパティーを使用して、1 つの接続に対してどのネットワーク・インターフェースを使用すべきかを指定することもできます。

ブローカーにリアルタイムで接続する場合、マルチキャストが使用される場合にのみ、LOCALADDRESS プロパティーが使用されます。 この場合、このプロパティーを使用して、1 つの接続に対してどのローカル・ネットワーク・インターフェースを使用すべきかを指定できますが、プロパティーの値にポート番号、またはポート番号の範囲を指定することはできません。

ポートの範囲を制限すると、接続エラーが起きる可能性があります。 エラーが起きた場合、IBM MQ 理由コード MQRC_Q_MGR_NOT_AVAILABLE および以下のメッセージを含む、MQException が組み込まれた JMSException がスローされます。

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

指定された範囲内のすべてのポートが使用中である場合、または指定された IP アドレス、ホスト名、ポート番号が正しくない場合 (負のポート番号など) は、エラーが発生する可能性があります。

IBM MQ classes for JMS はアプリケーションが必要とする接続以外の接続を作成する可能性があるため、常にポートの範囲を指定するようにしてください。 一般的に、アプリケーションが作成するセッションごとに 1 つのポートが必要で、IBM MQ classes for JMS はさらに 3 つまたは 4 つの追加ポートが必要になる場合があります。 接続エラーが起きた場合は、ポートの範囲を増やしてください。

IBM MQ classes for JMS でデフォルトで使用される接続プーリングは、ポートの再使用が可能になる速度に影響を与える場合があります。 このため、ポートの解放中に接続エラーが起きる可能性があります。

IBM MQ classes for JMS でのチャネル圧縮

IBM MQ classes for JMS アプリケーションは、IBM MQ 機能を使用してメッセージ・ヘッダーまたはデータを圧縮できます。

IBM MQ チャネルを流れるデータを圧縮すると、チャネルのパフォーマンスを改善し、ネットワーク・トラフィックを削減することができます。 IBM MQ で提供される機能を使用して、メッセージ・チャネルと MQI チャネルを流れるデータを圧縮できます。 また、どちらのタイプのチャネルでも、ヘッダー・データとメッセージ・データを個別に圧縮できます。 デフォルトでは、チャネル上のデータは圧縮されません。

IBM MQ classes for JMS アプリケーションは、接続のヘッダー・データまたはメッセージ・データの圧縮に使用できる手法を指定するために、java.util.Collection オブジェクトを作成します。 各圧縮手法は、このコレクション内の Integer オブジェクトです。アプリケーションが圧縮手法をコレクションに追加する順序は、アプリケーションの接続作成時に圧縮手法がキュー・マネージャーにネゴシエーションされた順序になります。 その後、アプリケーションで、このコレクションを ConnectionFactory オブジェクトに渡します。このとき、対象データがヘッダー・データであれば setHdrCompList() メソッドを呼び出し、メッセージ・データであれば setMsgCompList() メソッドを呼び出します。 アプリケーション側の準備が完了すれば、接続を作成できます。

以下のコード・フラグメントは、ここで説明した方法の例です。 最初のコード・フラグメントは、ヘッダー・データ圧縮の実装方法を示します。

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
```

```
.
connection = cf.createConnection();
```

2 番目のコード・フラグメントは、メッセージ・データ圧縮の実装方法を示します。

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

2 番目の例では、圧縮技法は、接続の作成時に RLE、次に ZLIBHIGH の順にネゴシエーションされます。選択された圧縮手法は、Connection オブジェクトが存続している間は変更できません。 接続上で圧縮を使用するには、Connection オブジェクトを作成する前に、setHdrCompList() メソッドと setMsgCompList() メソッドを呼び出す必要があります。

IBM MQ classes for JMS でのメッセージの非同期書き込み
通常、アプリケーションが宛先にメッセージを送信すると、アプリケーションはキュー・マネージャーが要求を処理したことを確認するまで待機しなければなりません。 ある環境では、そうする代わりにメッセージを非同期的に書き込ませることで、メッセージングのパフォーマンスを向上させることができます。アプリケーションにメッセージを非同期的に書き込ませる場合、キュー・マネージャーは、呼び出しのたびに成功や失敗を返しません。その代わりに、周期的にエラーの確認を行うことができます。

キュー・マネージャーがメッセージを支障なく受信したかどうかを判別することなく、宛先がアプリケーションに制御を 戻すかどうかは、以下のプロパティーによって異なります。

JMS 宛先プロパティー PUTASYNCALLOWED (短縮名 - PAALD)。
PUTASYNCALLOWED は、JMS の宛先が表している基礎となるキューまたはトピックによってこのオプションが許可されている場合に、JMS アプリケーションがメッセージを非同期に書き込むことができるかどうかを制御します。

IBM MQ キュー またはトピック・プロパティー DEFPRESP (デフォルトの書き込み応答タイプ)。
DEFPRESP は、メッセージをキューに書き込んだりトピックにパブリッシュしたりするアプリケーションで、非同期書き込み機能を使用できるかどうかを指定します。

次の表は、PUTASYNCALLOWED プロパティーと DEFPRESP プロパティーに指定できる値、および非同期書き込み機能を有効にするために使用される値の組み合わせを示しています。

表 *46.* メッセージが宛先に非同期に書き込まれるかどうかを決定する、*PUTASYNCALLOWED* プロパティーと *DEFPRESP* プロパティーの組み合わせ

IBM MQ キュー・プロパティー	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**非同期書き込み機能が有効**
DEFPRESP=SYNC	非同期書き込み機能が無効	非同期書き込み機能が有効	PUTASYNCALLOWED = AS_DEST または AS_Q_DEF または AS_T_DEF
DEFPRESP=ASYNC	非同期書き込み機能が無効	非同期書き込み機能が有効	PUTASYNCALLOWED = AS_DEST または AS_Q_DEF または AS_T_DEF

この動作を変更するには、以下の表に示すように IBM MQ-JMS 宛先プロパティーを「NO」または「YES」と指定しますが、Java 仮想マシン全体に対して JVM **SystemProperty** と値を使用してオーバーライドすることもできます。

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

トランザクション化されたセッションで送信されるメッセージの場合、アプリケーションは最終的に、commit() を呼び出すときにキュー・マネージャーがメッセージを支障なく受け取ったかどうかを判別します。

アプリケーションがトランザクション化されたセッションの中で持続メッセージを送信し、1 つ以上のメッセージの受信に支障があった場合、トランザクションはコミットに失敗し、例外が生成されます。 しかし、アプリケーションがトランザクション化されたセッションの中で非持続メッセージを送信し、1 つ以上のメッセージの受信に支障があった場合は、トランザクションは正常にコミットされます。 このアプリケーションは、非持続メッセージが無事に到着しなかったというフィードバックを受信しません。

トランザクション化されていないセッションで送信された非持続メッセージの場合、*ConnectionFactory* オブジェクトの SENDCHECKCOUNT プロパティーで、キュー・マネージャーがメッセージを支障なく受け取ったことを IBM MQ classes for JMS が検査するまでにいくつのメッセージを送信するかを指定します。

検査により 1 つ以上のメッセージの受信に支障があったことが分かり、アプリケーションが例外リスナーをその接続で登録している場合、IBM MQ classes for JMS は例外リスナーの onException() メソッドを呼び出し、JMS 例外をアプリケーションに渡します。

JMS 例外はエラー・コード JMSWMQ0028 を持ち、このコードにより以下のメッセージが表示されます。

```
At least one asynchronous put message failed or gave a warning.
```

さらに JMS 例外には、詳細を提供するリンクされた例外があります。 SENDCHECKCOUNT プロパティーのデフォルト値はゼロで、これはそのような検査が行われないことを意味します。

この最適化は、クライアント・モードでキュー・マネージャーに接続するアプリケーションに最も利点があり、一連のメッセージを連続して迅速に送信する必要がありますが、送信された各メッセージについてキュー・マネージャーから即時のフィードバックは必要としません。 ただし、バインディング・モードでキュー・マネージャーに接続する場合でも、アプリケーションは引き続きこの最適化を使用することができますが、予期されるパフォーマンスの利点はそれほど大きくありません。

注 : 未確認の **MessageProducer** を使用してトランザクションの下でメッセージを送信する場合、デフォルトでは、メッセージは非同期書き込みメカニズムを使用してキューに書き込まれます。

これは、 JMS API では、以下の構文を使用して、宛先を指定せずに **MessageProducer** を作成できるために発生する可能性があります。

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

このシナリオでは、**MessageProducer** が構成される前ではなく、メッセージが送信されるときに JMS 宛先が提供されます。 IBM MQ API の場合、結果として MQPUT1 が発行され、メッセージがキューに書き込まれます。

これを IBM MQ 同期点の下で行うと (JMS 用語では)、トランザクション化された JMS セッションを使用するか、または XASession を使用して、 IBM MQ classes for JMS API が非同期書き込みの使用に切り替わります。

IBM MQ classes for JMS での先読みの使用

IBM MQ で提供される先読み機能を使用することで、トランザクション外で受信された非永続メッセージを、アプリケーションから要求される前に IBM MQ classes for JMS に送信することができます。 IBM MQ classes for JMS は内部バッファーにメッセージを保管し、アプリケーションから要求された場合はそのメッセージをアプリケーションに渡します。

MessageConsumers または MessageListeners を使用してトランザクション外の宛先からメッセージを受信する IBM MQ classes for JMS アプリケーションは、先読み機能を使用できます。 先読みを使用することで、オブジェクトを使用するアプリケーションは、メッセージ受信時のパフォーマンスを改善できます。

MessageConsumers または MessageListeners を使用するアプリケーションが先読みを使用できるかどうかは、以下のプロパティーによって決まります。

JMS 宛先プロパティー READAHEADALLOWED (短縮名 - RAALD)。
READAHEADALLOWED は、JMS 宛先が表している基礎となるキューまたはトピックでこのオプションが許可されている場合に、JMS アプリケーションがトランザクション外の非永続メッセージの取得時または参照時に先読みを使用できるかどうかを制御します。

IBM MQ キューまたはトピック・プロパティー DEFREADA (デフォルトの先読み)。
DEFREADA は、トランザクション外の非永続メッセージを受信または参照しているアプリケーションが先読みを使用できるかどうかを指定します。

次の表は、READAHEADALLOWED プロパティーと DEFREADA プロパティーに指定できる値、および先読み機能を有効にするために使用される値の組み合わせを示しています。

表 *47.* トランザクション外の非永続メッセージを受信または参照する際に先読みを使用するかどうかを決定する、*READAHEADALLOWED* プロパティーと *DEFREADA* プロパティーの組み合わせ

IBM MQ キュー・プロパティー	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST または AS_Q_DEF または AS_T_DEF**
DEFREADA = NO	先読み機能が有効	先読み機能が無効	先読み機能が無効
DEFREADA = YES	先読み機能が有効	先読み機能が無効	先読み機能が有効
DEFREADA = DISABLED	先読み機能が無効	先読み機能が無効	先読み機能が無効

先読み機能が有効な場合に、アプリケーションで `MessageConsumer` または `MessageListener` が作成されると、IBM MQ classes for JMS は、`MessageConsumer` または `MessageListener` がモニターする宛先の内部バッファーを作成します。 `MessageConsumer` または `MessageListener` にはそれぞれ 1 つの内部バッファーがあります。 アプリケーションが以下のいずれかのメソッドを呼び出すと、キュー・マネージャーは IBM MQ classes for JMS への非永続メッセージの送信を開始します。

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

IBM MQ classes for JMS は、アプリケーションによるメソッド呼び出しで、最初のメッセージをアプリケーションに自動的に返します。 その他の非永続メッセージは、IBM MQ classes for JMS によって、宛先用に作成された内部バッファーに保管されます。 アプリケーションが次のメッセージの処理を要求すると、IBM MQ classes for JMS は内部バッファーの次のメッセージを返します。

内部バッファーが空の場合、IBM MQ classes for JMS はキュー・マネージャーからの非永続メッセージをさらに要求します。

IBM MQ classes for JMS によって使用される内部バッファーは、アプリケーションが `MessageConsumer` を閉じるとき、または `MessageListener` が関連付けられている JMS セッションを閉じるときに削除されます。

`MessageConsumers` の場合、内部バッファー内の未処理メッセージはすべて失われます。

`MessageListeners` を使用している場合、内部バッファー内のメッセージに何が起こるかは、 JMS 宛先プロパティー READAHEADCLOSEPOLICY (短縮名-RACP) によって異なります。 このプロパティーのデフォルト値は DELIVER_ALL です。これは、 `MessageListener` の作成に使用された JMS セッションが、内部バッファー内のすべてのメッセージがアプリケーションに配信されるまでクローズされないことを意味します。 プロパティーを DELIVER_CURRENT に設定すると、現行メッセージがアプリケーションで処理され、内部バッファー内の残りのメッセージがすべて破棄されてから、JMS セッションがクローズされます。

IBM MQ classes for JMS での保存パブリケーション

IBM MQ classes for JMS クライアントは、保存パブリケーションを使用するように構成することができます。

パブリッシャーは、将来このトピックに関心を持つサブスクライバーが現れた際にパブリケーションを送信できるよう、パブリケーションのコピーを保存しておくことを指定できます。 これは、整数プロパティー JMS_IBM_RETAIN を値 1 に設定することによって、IBM MQ classes for JMS で行われます。 これらの値に対して、com.ibm.msg.client.jms.JmsConstants インターフェースで定数が定義されています。 例えば、*msg* というメッセージを作成した際に、これを保存パブリケーションとして設定するには、以下のコードを使用します。

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

これで、このメッセージを通常のものとして送信できるようになります。 JMS_IBM_RETAIN を受信メッセージ内で照会することも可能です。 つまり、受信メッセージが保存パブリケーションであるかどうかを照会できるわけです。

IBM MQ classes for JMS での XA のサポート

JMS は、JEE コンテナー内のサポートしているトランザクション・マネージャーを使用して、バインディング・モードとクライアント・モードの XA 準拠トランザクションをサポートします。

アプリケーション・サーバー環境で XA 機能が必要な場合は、アプリケーションを適切に構成する必要があります。 分散トランザクションを使用するようにアプリケーションを構成する方法については、アプリケーション・サーバー独自の資料を参照してください。

IBM MQ キュー・マネージャーは、JMS のトランザクション・マネージャーの機能を果たすことはできません。

JMS メッセージの送達遅延

JMS 2.0 以降の場合、メッセージ送信時の送達遅延を指定できます。 キュー・マネージャーは、指定された送達遅延が経過するまでメッセージを送達しません。

アプリケーションは、 `MessageProducer.setDeliveryDelay(long deliveryDelay)` または `JMSProducer.setDeliveryDelay(long deliveryDelay)`のいずれかを使用して、メッセージの送信時の送達遅延をミリ秒単位で指定できます。 この値はメッセージの送信時間に追加され、他のアプリケーションがそのメッセージを取得できる最も早い時刻を指定します。

送達遅延は単一の内部ステージング・キューを使用して実装されます。 送達遅延がゼロ以外のメッセージは、送達遅延とターゲット・キューの情報を示すヘッダーを持つこのキューに入れられます。 ステージング・キューのメッセージは、送達遅延プロセッサーというキュー・マネージャーのコンポーネントによってモニターされます。 メッセージの送達遅延が完了すると、メッセージはステージング・キューから取り出され、ターゲット・キューに入れられます。

メッセージング・クライアント

IBM MQ の送達遅延の実装は、JMS クライアントを使用している場合にのみ使用可能になります。 IBM MQ で送達遅延を使用する場合は以下の制約事項が適用されます。 これらの制限は `MessageProducers` と `JMSProducers` にも同様に適用されますが、`JMSProducers` の場合は `JMSRuntimeExceptions` がスローされます。

- IBM MQ 8.0 より前のキュー・マネージャーに接続されているときに `MessageProducer.setDeliveryDelay` をゼロ以外の値で呼び出そうとすると、 `JMSException` に MQRC_FUNCTION_NOT_SUPPORTED メッセージが表示されます。
- 送達遅延は、MQBND_BIND_NOT_FIXED 以外の **DEFBIND** 値を持つクラスター宛先ではサポートされません。 `MessageProducer` でゼロ以外の送達遅延が設定されている状態で、この要件を満たしていない宛先に送信しようとすると、呼び出し時に `JMSException` が発生し、MQRC_OPTIONS_ERROR メッセージが表示されます。
- 以前に指定したゼロ以外の送達遅延より短い存続時間値を設定しようとすると (またはその逆の場合)、MQRC_EXPIRY_ERROR メッセージを伴う `JMSException` が発生します。 この確認は、選択された正確な操作セットに応じて、`setTimeToLive` または `setDeliveryDelay` あるいは `send` メソッドの呼び出し時に行われます。

- 保存パブリケーションと送達遅延の使用はサポートされていません。 送達遅延を使用したメッセージをパブリッシュしようとして、そのメッセージに msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION) を使用して保持のマークが付けられていると、MQRC_OPTIONS_ERROR メッセージを伴う JMSException が発生します。
- 送達遅延およびメッセージのグループ化はサポートされていません。この組み合わせを使用しようとすると、 JMSException に MQRC_OPTIONS_ERROR メッセージが出されます。

送達遅延を指定したメッセージの送信に失敗した場合、クライアントは JMSException をスローし、「キューがいっぱいです」などの適切なエラー・メッセージが表示されます。 状態によっては、エラー・メッセージがターゲット宛先またはステージング・キュー、あるいはその両方に適用される場合があります。

注 : IBM MQ を使用することで、アプリケーションはメッセージを作業単位で書き込み、その作業単位がコミットされなくても、同じメッセージを再度取得することができます。 この手法は、送達遅延では機能しません。メッセージは作業単位がコミットされるまでステージング・キューに入れられないため、ターゲット宛先には送信されないからです。

認証

IBM MQ は、送達遅延がゼロ以外のメッセージをアプリケーションが送信する際に、元のターゲット宛先に対して許可検査を行います。 アプリケーションが許可されていない場合、送信は失敗します。 キュー・マネージャーは、メッセージの送達遅延が完了したことを検出した時点でターゲット・キューを開きます。この時点では許可検査は行われません。

SYSTEM.DDELAY.LOCAL.QUEUE

システム・キュー SYSTEM.DDELAY.LOCAL.QUEUE は送達遅延を実装するために使用されます。

- Multi マルチプラットフォームの場合、SYSTEM.DDELAY.LOCAL.QUEUE はデフォルトで存在します。 システム・キューを変更して、その MAXMSGL および MAXDEPTH 属性が予想されるロードに十分な属性になるようにします。
- z/OS IBM MQ for z/OS の場合、 SYSTEM.DDELAY.LOCAL.QUEUE は、ローカル・キューと共用キューの両方に送達遅延を指定して送信されるメッセージのステージング・キューとして使用されます。 z/OS では、このキューを作成および定義して、その MAXMSGL および MAXDEPTH 属性が予想されるロードに十分な属性になるようにします。

このキューを作成したら、キューへのアクセス権限を持つユーザーをできるだけ少なくするために保護する必要があります。 このキューには保守とモニター目的でのみアクセスする必要があります。

送達遅延がゼロ以外のメッセージが JMS アプリケーションによって送信されると、新しいメッセージ ID を持つメッセージがこのキューに入れられます。 元のメッセージ ID はメッセージの相関 ID に入れられます。 この相関 ID を使用すれば、アプリケーションは必要な場合 (誤って長い送達遅延が使用された場合など) にステージング・キューからメッセージを検索することができます。

z/OS の考慮事項

z/OS

z/OS でシステムが実行されている場合、送達遅延を使用するには、注意すべき追加の考慮事項があります。

送達遅延を使用する場合は、システム・キュー SYSTEM.DDELAY.LOCAL.QUEUE を定義する必要があります。 このキューは、予想されるロードに十分な記憶域クラスを使用し、INDXTYPE(NONE) および MSGDLVSQ(FIFO) を指定して定義する必要があります。 システム・キューのサンプル定義は、コメント化された状態で CSQ4INSG JCL で提供されます。

共有キュー

送達遅延は共用キューにメッセージを送信する場合にサポートされます。 ただし、ターゲット・キューが共用されるかどうかに関係なく、使用されるのは単一の専用ステージング・キューのみです。 遅延完了後

に遅延メッセージをターゲット共有キューに送信するには、その専用キューを所有するキュー・マネージャーを実行する必要があります。

注 : 送達遅延を指定した非永続メッセージが共有キューに入れられ、ステージング・キューを所有するキュー・マネージャーがシャットダウンした場合は、元のメッセージが失われます。 結果として、送達遅延を指定して共用キューに送信される非永続メッセージは、送達遅延を指定せずに共用キューに送信される非永続メッセージより失われる可能性が高くなります。

ターゲットの宛先解決

メッセージをキューに送信した場合、解決は 2 回行われます。1 回は JMS アプリケーションによるもので、もう 1 回はキュー・マネージャーによるものです。これは、ステージング・キューからメッセージが取り出され、ターゲット・キューに送信されるときに行われます。

パブリケーションのためのターゲット・サブスクリプションは、JMS アプリケーションが send メソッドを呼び出す際にマッチングされます。

キュー定義に従って持続性または優先順位を持つメッセージが送信された場合、最初の解決に対して値が設定されますが、2 回目の解決には設定されません。

有効期限間隔

送達遅延は、有効期限プロパティー **MQMD.Expiry** の動作を保持します。 例えば、JMS アプリケーションから有効期限間隔が 20,000 ミリ秒で送達遅延が 5,000 ミリ秒のメッセージが書き込まれ、10,000 ミリ秒経過後に取得された場合、MQMD.expiry フィールドの値は約 50 (1/10 秒単位) になる可能性があります。この値は、メッセージが書き込まれてから取得されるまでに 15 秒が経過したことを意味します。

ステージング・キューに入れられている間にメッセージの有効期限が切れ、MQRO_EXPIRATION_* オプションのいずれかが設定されている場合、アプリケーションによって送信された元のメッセージに関するレポートが生成され、送達遅延情報を含めるために使用されたヘッダーは削除されます。

送達遅延プロセッサーの停止と開始

z/OS z/OS では、送達遅延プロセッサーはキュー・マネージャーの MSTR アドレス・スペースに統合されています。 キュー・マネージャーが始動すると、送達遅延プロセッサーも始動します。 ステージング・キューが使用可能な場合は、キューが開き、処理するメッセージの到着を待機します。 ステージング・キューが定義されていない場合、または get を使用できないか、あるいは別のエラーが発生した場合、送達遅延プロセッサーはシャットダウンします。 ステージング・キューが後で定義されたか、get を使用できるように変更された場合、送達遅延プロセッサーが再始動します。 送達遅延プロセッサーがその他の理由でシャットダウンした場合は、ステージング・キューの **PUT** 属性を ENABLED から DISABLED に変更し、再度 ENABLED に戻すことによって再始動できます。 何らかの理由で送達遅延プロセッサーを停止しなければならない場合は、ステージング・キューの PUT 属性を DISABLED に設定します。

Multi マルチプラットフォームでは、遅延プロセッサーはキュー・マネージャーと一緒に始動するため、リカバリー可能な障害が発生すると、自動的に再始動されます。

ターゲット・キューへの書き込み失敗

遅延完了後に遅延メッセージをターゲット・キューに書き込めない場合、メッセージはレポート・オプションに示されたとおりに処理されます (廃棄されるか、送達不能キューに送信されます)。 このアクションが失敗した場合、メッセージを後で書き込むよう試行されます。 アクションが成功すると、例外レポートが生成され、指定されたキューに送信されます (レポートが要求された場合)。 レポート・メッセージを送信できなかった場合、レポート・メッセージは送達不能キューに送信されます。 送達不能キューへのメッセージの送信が失敗して、メッセージが持続する場合、すべての変更は破棄され、元のメッセージはロールバックされて、後で再送達されます。 メッセージが持続しない場合、レポート・メッセージは廃棄されますが、他の変更はコミットされます。 サブスクライバーがアンサブスクライブされたために遅延パブリケーションを送達できない場合や、サブスクライバーが切断されたために非永続である場合、メッセージは暗黙的に廃棄されます。 レポート・メッセージは前述のとおり、引き続き生成されます。

遅延パブリケーションをサブスクライバーに送達できないため、送達不能キューに書き込もうとして失敗した場合、メッセージは廃棄されます。

送達遅延完了後のターゲット・キューへの書き込み失敗の可能性を低くするために、キュー・マネージャーは、JMS クライアントが送達遅延がゼロ以外のメッセージを送信する際に基本検査をいくつか実行します。 これらの検査には、キューで put が使用不可かどうかや、メッセージがメッセージ最大許容長を超えているかどうか、またキューがいっぱいかどうかの確認が含まれます。

パブリッシュ/サブスクライブ

送達遅延がゼロ以外のメッセージを JMS アプリケーションが送信する際に、パブリケーションと使用可能なサブスクリプションとのマッチングが行われます。 一致する各サブスクライバーのメッセージは SYSTEM.DDELAY.LOCAL.QUEUE キューに書き込まれ、送達遅延が完了するまで保持されます。 これらのサブスクライバーのいずれかが別のキュー・マネージャーのプロキシー・サブスクリプションである場合は、送達遅延完了後にそのキュー・マネージャーでファンアウトが発生します。 これにより、他のキュー・マネージャーのサブスクライバーが、サブスクライブの前に最初にパブリッシュされたパブリケーションを受信する場合があります。 これは、 JMS 2.0 以降の仕様からの逸脱です。

パブリッシュ/サブスクライブでの送達遅延がサポートされるのは、ターゲット・トピックが (N)PMSGDLV = ALLAVAIL で構成されている場合のみです。 他の値を使用しようとすると、MQRC_PUBLICATION_FAILURE エラーが発生します。 送達遅延プロセッサーがターゲット・キューへのメッセージの書き込み中に失敗すると、『ターゲット・キューへの書き込み失敗』セクションに記載されているとおりの結果となります。

レポート・メッセージ

以下の無視されるオプションを除くすべてのレポート・オプションが送達プロセッサーによってサポートされ、処理されます。 ただし、ターゲット・キューに送信されるメッセージでは、パススルーされます。

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY

複製サブスクリプションおよび共用サブスクリプション

IBM MQ 8.0 以降では、同じサブスクリプションへのアクセス権限を複数のコンシューマーに付与する方法が 2 つあります。 それらは、クローン・サブスクリプションを使用する方法と共用サブスクリプションを使用する方法です。

クローン・サブスクリプション

クローン・サブスクリプションは IBM MQ 拡張です。 クローン・サブスクリプションでは、異なる Java 仮想マシン (JVM) の複数のコンシューマーが、サブスクリプションに同時にアクセスできます。 この動作は、connectionFactory オブジェクトの **CLONESUPP** プロパティーを「使用可能」に設定することによって使用できます。 デフォルトの **CLONESUPP** は「使用不可」です。 クローン・サブスクリプションは、永続サブスクリプションでのみ使用可能にできます。 **CLONESUPP** を使用可能にすると、この connectionFactory を使用して行われる後続の各接続は複製されます。

そのサブスクリプションからメッセージを受け取るコンシューマーが 1 つ以上作成された場合 (つまり、それらのコンシューマーが同じサブスクリプション名を指定して作成された場合)、永続サブスクリプションはクローンと見なすことができます。 これが可能なのは、コンシューマーが作成された接続の **CLONESUPP** が MQConnectionFactory で「使用可能」に設定されている場合のみです。 サブスクリプションのトピックに関してメッセージがパブリッシュされると、そのメッセージのコピーがサブスクリプションに送られます。 どのコンシューマーもメッセージを使用可能ですが、受け取ることができるのは 1 つのコンシューマーだけです。

注 : クローン・サブスクリプションの使用可能にすると、JMS 仕様が拡張されます。

共用サブスクリプション

JMS 2.0 仕様で導入された共有サブスクリプションを使用すると、トピック・サブスクリプションからのメッセージを複数のコンシューマーで共有することができます。 サブスクリプションからの各メッセージは、そのサブスクリプションのコンシューマーの 1 つにのみ配信されます。 共有サブスクリプションは、JMS 2.0 以降の API への関連する呼び出しによって有効になります。

API は、以下のいずれかの方法で呼び出すことができます。

- Java SE アプリケーション (または Java EE Client Container) から。
- サーブレットまたは MDB の実装から。

JMS 2.0 以降の仕様では、共用サブスクリプションから MDB を駆動する標準的な方法は定義されていないため、IBM MQ 8.0 以降では、この目的のために **sharedSubscription** アクティベーション・スペック・プロパティーを提供しています。 このプロパティーの詳細については、453 ページの『インバウンド通信のリソース・アダプターの構成』および 469 ページの『sharedSubscription プロパティーの定義方法を示す例』を参照してください。

共用サブスクリプションを使用可能にすると、非共用にすることはできません。

共用サブスクリプションは、永続サブスクリプションまたは非永続サブスクリプションとして作成できます。 通常の JMS 構成以外では、キュー・マネージャー側でオブジェクトを個別に作成する必要はありません。 必要なオブジェクトはすべて動的に作成されます。

共用サブスクリプションかクローン・サブスクリプションかの判断

共有サブスクリプションとクローン・サブスクリプションのどちらを使用するかを決定する際には、両方の利点を考慮してください。 可能な場合は、 IBM MQ 固有の拡張機能ではなく、仕様定義の動作である共有サブスクリプションを使用してください。

共用サブスクリプションとクローン・サブスクリプションのどちらにするかを判断する際に考慮すべきいくつかのポイントを、次の表に記載しています。

表 *48.* 共用サブスクリプションとクローン・サブスクリプションの考慮事項の比較

共用サブスクリプション	**クローン・サブスクリプション**
共用サブスクリプションは、JMS 2.0 以降の仕様の標準部分です。	クローン・サブスクリプションは、IBM MQ 固有の拡張です。
共用サブスクリプションは、明示的な API メソッド呼び出しを使用することによって作成されます。	クローン・サブスクリプションは、ConnectionFactory レベルで管理的に制御されます。
共用サブスクリプションは、永続または非永続が可能です。	クローン・サブスクリプションは、永続のみ可能です。
共用サブスクリプションは、個々のサブスクリプションで明示的に作成されます。	クローン・サブスクリプションは、この機能が使用可能な接続でのあらゆる永続サブスクリプションに使用されます。
共用として作成されたサブスクリプションを後で非共用に変更することはできません。逆もまた同様です。	所有接続の **CLONESUPP** プロパティーが変更された場合、サブスクリプションの再オープンのたびに、そのサブスクリプションをクローンから非クローンに変更することができます。

関連概念

サブスクライバーとサブスクリプション

サブスクリプション永続性

関連タスク

JMS 2.0 共用サブスクリプションの使用

関連資料

469 ページの『sharedSubscription プロパティーの定義方法を示す例』

WebSphere Liberty `server.xml` ファイル内でアクティベーション・スペックの sharedSubscription プロパティーを定義することができます。 あるいは、アノテーションを使用してメッセージ駆動型 Bean (MDB) 内でプロパティーを定義できます。

CLONESUPP

SupportMQExtensions プロパティー

JMS 2.0 仕様では、特定の動作の動作方法が変更されました。 IBM MQ 8.0 以降には、プロパティー **com.ibm.mq.jms.SupportMQExtensions** が組み込まれています。これを TRUE に設定すると、変更された動作を以前の実装に戻すことができます。

JM 3.0 V 9.3.0 V 9.3.0 **com.ibm.mq.jakarta.jms.SupportMQExtensions** プロパティー (Jakarta Messaging 3.0) は、 `com.ibm.mq.jakarta.client.jar` で使用可能な IBM MQ classes for Jakarta Messaging によってサポートされます。

JMS 2.0 **com.ibm.mq.jms.SupportMQExtensions** (JMS 2.0) プロパティーは、 `com.ibm.mq.allclient.jar` または `com.ibm.mqjms.jar` で使用可能な IBM MQ classes for JMS によってサポートされます。

以下の 3 つの機能は、**SupportMQExtensions** を `True` に設定して戻します。

メッセージ優先順位
メッセージには 0 から 9 の優先順位を割り当てることができます。 JMS 2.0 より前では、メッセージは値 -1 を使用することもできました。これは、キューのデフォルト優先順位が使用されることを示します。 JMS 2.0 以降では、-1 のメッセージ優先順位を設定することはできません。 **SupportMQExtensions** をオンにすると、値 -1 を使用できます。

クライアント ID
JMS 2.0 以降の指定では、NULL 以外のクライアント ID が接続時に固有かどうかを検査する必要があります。 **SupportMQExtensions** をオンにすると、この要件は無視され、クライアント ID を再利用できます。

NoLocal
JMS 2.0 以降の仕様では、この定数をオンにすると、コンシューマーは同じクライアント ID によってパブリッシュされたメッセージを受信できなくなります。 JMS 2.0 より前では、この属性はサブスクライバーに設定され、独自の接続でパブリッシュされたメッセージが受信されないようになっていました。 **SupportMQExtensions** をオンにすれば、この動作は以前の実装に戻ります。

このプロパティーは、以下のように設定できます。

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

このプロパティーは、標準の JVM システム・プロパティーとして **java** コマンドで設定するか、または IBM MQ classes for JMS 構成ファイルに含めることができます。

関連概念
99 ページの『IBM MQ classes for JMS/Jakarta Messaging 構成ファイル』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging 構成ファイルは、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の構成に使用されるプロパティーを指定します。

関連資料
106 ページの『JMS クライアント動作の構成に使用されるプロパティー』
これらのプロパティーを使用して、JMS クライアントの動作を構成します。

JMS アプリケーションでの共用サブスクリプションの使用

共有サブスクリプションを使用すると、単一のサブスクリプションが複数のコンシューマー間で共有され、どの時点においても 1 つのコンシューマーのみがパブリケーションを受け取ります。

共有サブスクリプションは、 JMS 2.0 以降で使用できます。 したがって、 IBM MQ 8.0 以降用の JMS アプリケーションを開発する場合は、この機能がキュー・マネージャーに与える影響を考慮する必要があります。

共有サブスクリプションの背後にある考え方は、複数のコンシューマー間で負荷を共有することです。 永続サブスクリプションも、複数のコンシューマーで共有することができます。

例えば、次のような状況があるとします。

- サブスクリプション SUB は、サッカーの対戦結果の最新情報を受け取るためのトピック FIFA2014/UPDATES にサブスクライブしていて、C1、C2、および C3 という 3 つのコンシューマーが共有しています。
- プロデューサー P1 は、FIFA2014/UPDATES トピックにパブリッシュしています。

FIFA2014/UPDATES に対してパブリケーションが行われると、そのパブリケーションを受け取るのは、全員ではなく 3 つのコンシューマー (C1 、C2、または C3) の中のいずれか 1 つのみです。

以下のサンプルは、共有サブスクリプションの使用法を示しています。また、メッセージ本文のみを取得するための JMS 2.0 の追加 API `Message.receiveBody()`の使用法も示しています。

このサンプルでは、FIFA2014/UPDATES トピックへの共有サブスクリプションを作成する 3 つのサブスクライバー・スレッドと、1 つのパブリッシャー・スレッドを作成します。

JM 3.0 V 9.3.0 V 9.3.0

```
package mqv91Samples;

import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;
```

```
                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
```

```
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
```

```
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

関連概念

IBM MQ Java 言語インターフェース

V 9.3.2 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用するためのモジュラー・アプリケーションの構成

V 9.3.2 アプリケーション内で適切なモジュールを要求し、適切なディレクトリーを module-path に組み込むことにより、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging をモジュラー方式で使用することができます。

モジュラー・パッケージ

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging の統合 JAR ファイルは、自動モジュール名を提供します。これにより、JAR ファイル名から派生したデフォルト名が置き換えられます。

- IBM MQ classes for JMS (com.ibm.mq.allclient.jar) には、 com.ibm.mq.javax というモジュール名が用意されています。
- IBM MQ classes for Jakarta Messaging (com.ibm.mq.jakarta.client.jar) には、com.ibm.mq.jakarta というモジュール名が用意されています。

デフォルトの MQ_HOME/java/lib ディレクトリーは、モジュールが同じパッケージを含むことができず、デフォルト・ディレクトリーに複数の JAR 内の同じパッケージが含まれているため、モジュラー使用には適していません。 そのため、JAR 間でパッケージを重複させずに、必要な JAR ファイルのみを含む新規ディレクトリーを使用できます。 これらのディレクトリーは、module-path に組み込むのに適しています。

注 : デフォルトのモジュール名に依存して、使用可能な JAR ファイルをモジュラー・コンテキストで使用するアプリケーションがある場合は、新しいモジュール名を必要とするようにアプリケーションを更新する必要があります。 デフォルトのモジュール名は、JAR ファイル名から派生します。

IBM MQ classes for JMS を使用するためのモジュラー・アプリケーションの構成

以下のステップを実行して、IBM MQ classes for JMS (com.ibm.mq.allclient.jar) を使用するようにモジュラー・アプリケーションを構成できます。

- com.ibm.mq.javax モジュールを必要とするようにアプリケーションを構成します。
- MQ_HOME/java/lib/modules/javax ディレクトリーを module-path に組み込むようにアプリケーションを構成します。

IBM MQ classes for Jakarta Messaging を使用するためのモジュラー・アプリケーションの構成

以下のステップを実行して、 IBM MQ classes for Jakarta Messaging (com.ibm.mq.jakarta.client.jar) を使用するようにモジュラー・アプリケーションを構成できます。

- com.ibm.mq.jakarta モジュールを必要とするようにアプリケーションを構成します。
- MQ_HOME/java/lib/modules/jakarta ディレクトリーを module-path に組み込むようにアプリケーションを構成します。

IBM MQ classes for Java を使用するためのモジュラー・アプリケーションの構成

IBM MQ classes for Java をモジュラー・アプリケーションから使用するには、両方のクライアント JAR ファイルが IBM MQ classes for Java をサポートするため、 IBM MQ classes for JMS の構成または IBM MQ classes for Jakarta Messaging の構成のいずれかを使用できます。 ただし、アプリケーションは、これらの構成の両方ではなく、いずれか 1 つのみを使用する必要があります。

IBM MQ classes for JMS アプリケーション・サーバー機構

このトピックでは、IBM MQ classes for JMS が Session クラス内の ConnectionConsumer クラスおよび拡張機能を実装する方法を説明します。 さらに、サーバー・セッション・プールの機能も要約しています。

重要 : この情報は、あくまでも参考用です。 このインターフェースを使用するアプリケーションを作成してはいけません。このインターフェースは、IBM MQ リソース・アダプター内で Java EE サーバーに接続するために使用されます。 接続に関する実際的な情報については、436 ページの『IBM MQ リソース・アダプターの使用』を参照してください。

IBM MQ classes for JMS は、 *Java Message Service* 仕様 に指定されている Application Server Facilities (ASF) をサポートしています (Oracle Technology Network for Java Developers を参照してください)。 この仕様では、このプログラミング・モデル内で以下の 3 つの役割を定めています。

- **JMS プロバイダー**は、ConnectionConsumer および拡張セッション機能を提供します。
- **アプリケーション・サーバー**は、ServerSessionPool および ServerSession 機能を提供します。
- **クライアント・アプリケーション**は、JMS プロバイダーおよびアプリケーション・サーバーが提供する機能を使用します。

このトピックの情報は、アプリケーションがブローカーとのリアルタイム接続を使用する場合には適用されません。

JMS ConnectionConsumer

ConnectionConsumer インターフェースは、スレッドのプールに同時にメッセージを送達するための高性能のメソッドを提供します。

JMS 仕様により、アプリケーション・サーバーは、 `ConnectionConsumer` インターフェースを使用して JMS 実装と緊密に統合できます。 この機能は、メッセージの並行処理を提供します。 通常、アプリケーション・サーバーがスレッドのプールを作成し、JMS 実装がこれらのスレッドに使用可能なメッセージを作成します。 JMS 対応のアプリケーション・サーバー (WebSphere Application Server など) は、この機能を使用して、メッセージ駆動型 Bean など高水準のメッセージング機能を提供することができます。

通常のアプリケーションは ConnectionConsumer を使用しませんが、エキスパート JMS クライアントは ConnectionConsumer を使用することがあります。 そのようなクライアントに対して、ConnectionConsumer は、スレッドのプールに同時にメッセージを送達するための高性能のメソッドを提供します。 メッセージがキューまたはトピックに到達した際、JMS はプールからスレッドを選択し、そのスレッドにメッセージのバッチを送達します。 これを行うために、JMS は、関連する MessageListener の `onMessage()` メソッドを実行します。

複数の Session および MessageConsumer オブジェクト (それぞれ登録された MessageListener を持つ) を構成することによって、同じ効果が得られます。 ただし、ConnectionConsumer の方が、パフォーマンスが良く、リソースの使用量が少なく、より大きな柔軟性があります。 特に、必要となる Session オブジェクトが少数で済みます。

ASF によるアプリケーションの計画

このセクションでは、以下のようなアプリケーション計画の方法について説明します。

- 334 ページの『ASF を使用した Point-to-Point メッセージングの一般原則』
- 335 ページの『ASF を使用したパブリッシュ/サブスクライブ・メッセージングの一般原則』
- 336 ページの『ASF でのキューからのメッセージの除去』
- ASF での有害メッセージの処理。 236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照。

ASF を使用した *Point-to-Point* メッセージングの一般原則

このトピックでは、ASF を使用した Point-to-Point メッセージングの一般情報について説明します。

アプリケーションは、QueueConnection オブジェクトから ConnectionConsumer を作成する際、JMS キュー・オブジェクトおよびセレクター・ストリングを指定します。 その後、ConnectionConsumer は、関連した ServerSessionPool 内のセッションにメッセージを供給し始めます。 メッセージがキューに到着し、セレクターと一致する場合、メッセージは、関連付けられている ServerSessionPool 内のセッションに送達されます。

IBM MQ 用語では、キュー・オブジェクトは、ローカル・キュー・マネージャー上にある QLOCAL または QALIAS のいずれかを表します。 QALIAS である場合には、その QALIAS は QLOCAL を参照しなければなりません。 完全に解決された IBM MQ QLOCAL を、基礎 *QLOCAL* と言います。 ConnectionConsumer がクローズされておらず、その親 QueueConnection が開始されている場合には、ConnectionConsumer はアクティブ であると言います。

複数の ConnectionConsumer (それぞれ異なるセレクターを持つ) を、同じ基礎 QLOCAL に対して実行することが可能です。 パフォーマンスを保つために、不必要なメッセージをキュー上に累積させないようにしてください。 不必要なメッセージとは、アクティブな ConnectionConsumer が一致するセレクターを持たないメッセージを指します。 これらの不必要なメッセージがキューから除去されるように QueueConnectionFactory を設定できます (詳細については、336 ページの『ASF でのキューからのメッセージの除去』を参照してください)。 以下の 2 つの方法のいずれかで、この動作を設定できます。

- JMS 管理ツールを使用して、QueueConnectionFactory を MRET(NO) に設定する。
- プログラムで、以下を使用する。

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

この設定を変更しない場合、デフォルトでは、キュー上にこうした不必要なメッセージを保存します。

IBM MQ キュー・マネージャーをセットアップする際は、以下の点を考慮してください。

- 基礎 QLOCAL は、共用入力のために使用可能でなければなりません。 これを行うには、以下の MQSC コマンドを使用します。

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- キュー・マネージャーは、使用可能な送達不能キューを所有していなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れる際に問題が発生する場合、基礎 QLOCAL からのメッセージ送達は停止します。 送達不能キューを定義するには、以下のものを使用します。

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- ConnectionConsumer を実行するユーザーは、MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を所有していなければなりません。 詳細については、ご使用のプラットフォーム用の IBM MQ 資料を参照してください。
- 不必要なメッセージがキュー上に残されている場合、システム・パフォーマンスが低下します。 したがって、メッセージ・セレクター間で、ConnectionConsumer がキューからすべてのメッセージを除去するように、メッセージ・セレクターを計画してください。

MQSC コマンドの詳細については、MQSC コマンドを参照してください。

ASF を使用したパブリッシュ/サブスクライブ・メッセージングの一般原則
ConnectionConsumers は、指定した Topic のメッセージを受信します。 ConnectionConsumer は、永続にも非永続にもできます。 ConnectionConsumer が使用するキュー (複数可) を指定する必要があります。

アプリケーションが TopicConnection オブジェクトから ConnectionConsumer を作成する際、アプリケーションは、Topic オブジェクトおよびセレクター・ストリングを指定します。 その後、ConnectionConsumer は、サブスクライブ対象のトピックに関する保存されているパブリケーションを含め、Topic オブジェクト上のセレクターと一致するメッセージを受信し始めます。

また、アプリケーションは、特定の名前に関連付けられた永続 ConnectionConsumer の作成もできます。 この ConnectionConsumer は、永続 ConnectionConsumer が最後にアクティブであったとき以降に、Topic についてパブリッシュされているメッセージを受信します。 この ConnectionConsumer は、Topic 上のセレクターと一致するすべてのメッセージを受信します。 しかし、ConnectionConsumer が先読みを使用する場合、クローズする際に、クライアント・バッファーにある非持続メッセージが失われることがあります。

IBM MQ classes for JMS が IBM MQ メッセージング・プロバイダーのマイグレーション・モードである場合、別個のキューが非永続 ConnectionConsumer サブスクリプションに使用されます。
TopicConnectionFactory 上の CCSUB 構成可能オプションは、使用するキューを指定します。 通常、CCSUB は、同じ TopicConnectionFactory を使用するすべての ConnectionConsumer が使用するための単一キューを指定する必要があります。 ただし、キュー名の接頭部とその後にアスタリスク (*) を指定することによって、各 ConnectionConsumer に一時キューを生成させることができます。

IBM MQ classes for JMS が IBM MQ メッセージング・プロバイダーのマイグレーション・モードである場合、Topic の CCDSUB プロパティーは、永続サブスクリプションに使用するキューを指定します。 ここでもまた、既に存在するキュー、またはキュー名の接頭部の後にアスタリスク (*) が付いたものになり得ます。既に存在するキューを指定すると、トピックをサブスクライブするすべての永続 ConnectionConsumer がこのキューを使用します。 キュー名の接頭部とその後にアスタリスク (*) を指定した場合、永続 ConnectionConsumer が指定の名前で初めて作成されるときにキューが生成されます。 このキューは、後に永続 ConnectionConsumer が同じ名前で作成される際に再使用されます。

IBM MQ キュー・マネージャーをセットアップする際は、以下の点を考慮してください。

- キュー・マネージャーは、使用可能な送達不能キューを所有していなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れる際に問題が発生する場合、基礎 QLOCAL からのメッセージ送達は停止します。 送達不能キューを定義するには、以下のものを使用します。

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- ConnectionConsumer を実行するユーザーは、MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を所有していなければなりません。 詳細については、ご使用のプラットフォーム用の IBM MQ 資料を参照してください。
- 個々の ConnectionConsumer 用に、別個の専用キューを作成することによって、パフォーマンスを最適化することができます。 この場合、リソースが余分に使用されます。

ASF でのキューからのメッセージの除去
アプリケーションが ConnectionConsumer を使用する際、いくつかの状況では、JMS は、キューからメッセージを除去する必要があります。

これは、次のような状況です。

不適切なフォーマットのメッセージ
JMS が解析不能なメッセージが到着することがあります。

有害メッセージ
メッセージはバックアウトしきい値に達しますが、ConnectionConsumer は、バックアウト・キュー上へのメッセージのリキューに失敗します。

関係のない ConnectionConsumer
Point-to-Point メッセージングの場合、QueueConnectionFactory が不必要なメッセージを保存しないように設定されている場合、どの ConnectionConsumer にも不必要なメッセージが到着します。

これらの状態では、ConnectionConsumer は、キューからメッセージを除去しようとします。 メッセージの MQMD のレポート・フィールド内の後処理オプションは、正確な振る舞いを設定します。 そうしたオプションは、以下のとおりです。

MQRO_DEAD_LETTER_Q
メッセージは、キュー・マネージャーの送達不能キューにリキューされます。 これはデフォルトです。

MQRO_DISCARD_MSG
メッセージは破棄されます。

また、ConnectionConsumer はレポート・メッセージも生成しますが、これもメッセージの MQMD のレポート・フィールドに依存します。 このメッセージは、ReplyToQmgr 上のメッセージの ReplyToQ に送信されます。 レポート・メッセージが送信されている間にエラーがある場合、メッセージは、代わりに、送達不能キューに送信されます。 メッセージの MQMD のレポート・フィールド内の例外レポート・オプションは、レポート・メッセージの詳細を設定します。 そうしたオプションは、以下のとおりです。

MQRO_EXCEPTION
オリジナル・メッセージの MQMD を含むレポート・メッセージが生成されます。 このメッセージには、いかなるメッセージ本体データも含まれていません。

MQRO_EXCEPTION_WITH_DATA
MQMD、任意の MQ ヘッダー、および 100 バイトの本体データを含むレポート・メッセージが生成されます。

MQRO_EXCEPTION_WITH_FULL_DATA
オリジナル・メッセージからのすべてのデータを含むレポート・メッセージが生成されます。

default
レポート・メッセージは生成されません。

レポート・メッセージが生成される際、以下のオプションが有効です。

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID

- MQRO_PASS_CORREL_ID

ポイズン・メッセージがリキューできない場合、たいていは送達不能キューがいっぱいであるか、許可が正しく指定されていないことが原因ですが、どのような動作になるかはメッセージが持続メッセージであるかどうかによって異なります。 メッセージが非持続メッセージである場合、メッセージは破棄され、レポート・メッセージは生成されません。 メッセージが持続メッセージである場合は、その宛先で listen しているすべての接続コンシューマーへのメッセージの送達が停止します。 これらの接続コンシューマーを閉じ、問題を解決する必要があります。その後、接続コンシューマーを再作成し、メッセージの送達を再開することができます。

送達不能キューを定義し、定期的にチェックして、問題が発生していないかを確認することは重要です。特に、送達不能キューがその最大の深さに達していないこと、および、その最大メッセージ・サイズがすべてのメッセージに対して十分な大きさであることを確認してください。

メッセージが送達不能キューにリキューされるとき、メッセージの前に IBM MQ 送達不能ヘッダー (MQDLH) が付けられます。 MQDLH のフォーマットの詳細については、MQDLH - 送達不能ヘッダーを参照してください。 以下のフィールドによって、ConnectionConsumer が送達不能キューに入れたメッセージを識別したり、ConnectionConsumer が生成したメッセージを報告したりできます。

- PutApplType は、MQAT_JAVA (0x1C) です。
- PutApplName は、"MQ JMS ConnectionConsumer" です。

これらのフィールドは、送達キュー上のメッセージの MQDLH 内およびレポート・メッセージの MQMD 内にあります。 MQMD のフィードバック・フィールド、および MQDLH の Reason フィールドには、エラーを説明しているコードが含まれています。 これらのコードの詳細については、338 ページの『ASF での理由コードおよびフィードバック・コード』を参照してください。 他のフィールドについては、MQDLH - 送達不能ヘッダーを参照してください。

ASF での有害メッセージの処理

Application Server Facilities (ASF) では、有害メッセージの処理の方法が IBM MQ classes for JMS 内の他の場所とは少し異なっています。

IBM MQ classes for JMS での有害メッセージの処理については、236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

Application Server Facilities (ASF) を使用する場合は、MessageConsumer の代わりに ConnectionConsumer で有害メッセージを処理します。 ConnectionConsumer は、キューの BackoutThreshold および BackoutRequeueQName プロパティーに従ってメッセージをリキューします。

アプリケーションが ConnectionConsumers を使用している場合、メッセージがバックアウトされる状況は、アプリケーション・サーバーが提供するセッションによって異なります。

- セッションが非トランザクション化セッションで、AUTO_ACKNOWLEDGE または DUPS_OK_ACKNOWLEDGE が指定されている場合、メッセージがバックアウトされるのは、システム・エラーの後、またはアプリケーションが予期せずに終了した場合のみです。
- セッションが非トランザクションで CLIENT_ACKNOWLEDGE を伴う場合、認められていないメッセージは、Session.recover() を呼び出すアプリケーション・サーバーによってバックアウトできます。

  通常、MessageListener またはアプリケーション・サーバーのクライアント実装は、Message.acknowledge() を呼び出します。Message.acknowledge() は、これまでにセッションに送達されたすべてのメッセージを認識します。
- セッションがトランザクション化セッションの場合、無応答メッセージは、アプリケーション・サーバーが Session.rollback() を呼び出すことによってバックアウトすることができます。
- アプリケーション・サーバーが XASession を提供する場合、メッセージは分散トランザクションに応じてコミットまたはバックアウトされます。 アプリケーション・サーバーは、トランザクションを完了させる責任があります。

関連概念

236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』
ポイズン・メッセージは、受信側のアプリケーションでは処理できないメッセージです。 有害メッセージがアプリケーションに配信され、ロールバックされるということが、指定回数発生した場合、 IBM MQ classes for JMS はそのメッセージをバックアウト・キューに移動できます。

エラーの処理

このセクションでは、338 ページの『ASF でのエラー状態からの回復』や338 ページの『ASF での理由コードおよびフィードバック・コード』などを含めて、エラー処理のさまざまな側面を取り上げます。

ASF でのエラー状態からの回復
ConnectionConsumer が重大エラーに遭遇した場合、同じ QLOCAL 内にインタレストを持つすべての ConnectionConsumer へのメッセージ送達は停止します。 このエラーの発生時には、影響を受けた Connection とともに登録されている ExceptionListener が通知されます。 アプリケーションがこれらのエラー状態から回復できる 2 つの方法があります。

通常、ConnectionConsumer が送達不能キューにメッセージをリキューできない場合にこのような重大なエラーが発生するか、または QLOCAL からメッセージを読み取るときにエラーに遭遇します。

影響を受けた Connection とともに登録されている ExceptionListener が通知されるため、これらを使用して問題の原因を特定できます。 問題解決のためにシステム管理者が介入しなければならない場合があります。

これらのエラー状態から回復するには、以下のいずれかの手法を使用してください。

- 影響を受けたすべての ConnectionConsumer で `close()` を呼び出します。 アプリケーションは、影響を受けたすべての ConnectionConsumer がクローズされて、システム問題が解決された後にのみ、新しい ConnectionConsumer を作成できます。
- 影響を受けたすべての Connection で `stop()` を呼び出します。 すべての Connection が停止され、システム問題が解決されると、アプリケーションは、その Connection を正常に `start()` できます。

ASF での理由コードおよびフィードバック・コード
理由コードおよびフィードバック・コードを使用して、エラーの原因を判別します。 ConnectionConsumer によって生成される一般的な理由コードをここで示します。

エラーの原因を判別するには、以下の情報を使用します。

- レポート・メッセージ内のフィードバック・コード
- 送達不能キュー内のメッセージの MQDLH 内にある理由コード

ConnectionConsumer は、以下の理由コードを生成します。

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

原因
メッセージは、QLOCAL 上に定義されている Backout Threshold に達したが、Backout Queue が定義されていません。

Backout Queue を定義できないプラットフォームでは、メッセージは、JMS 定義のバックアウトしきい値 20 に達します。

アクション
これが必要なければ、関係のある QLOCAL に Backout Queue を定義してください。 また、複数のバックアウトの原因も調べてください。

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

原因
Point-to-Point メッセージングで、キューをモニターしている ConnectionConsumer 用のセレクターのいずれかと一致しないメッセージが存在します。 パフォーマンスを保つために、メッセージが送達不能キューにリキューされます。

アクション
この状態を回避するには、キューを使用している ConnectionConsumer がすべてのメッセージを処理する一連のセレクターを提供するか、または QueueConnectionFactory を設定してメッセージを保存してください。

または、メッセージの送信元を調べてください。

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

原因

JMS が、キュー上のメッセージを解釈できません。

アクション

メッセージの発信元を調べてください。 通常、JMS は、BytesMessage または TextMessage として予期しないフォーマットのメッセージを送達します。 時折、メッセージが極めて不適切なフォーマットである場合には、このアクションは失敗します。

他のコードがこれらのフィールドに表示されるのは、Backout Queue へのメッセージのリキューが失敗したことが原因です。 この状態では、コードは、リキューが失敗した理由を説明しています。 これらのエラーの原因を診断するには、 API 完了コードと理由コードを参照してください。

レポート・メッセージを ReplyToQ に入れることができない場合、レポート・メッセージは送達不能キューに入れられます。 この状態では、MQMD のフィードバック・フィールドは、このトピックで説明されているとおりに埋められます。 MQDLH 内の理由フィールドは、レポート・メッセージを ReplyToQ に入れることができなかった理由を説明します。

AFS でのサーバー・セッション・プールの機能

このトピックでは、サーバー・セッション・プールの機能を要約しています。

340 ページの図 45 に、ServerSessionPool および ServerSession 機能の基本を要約します。

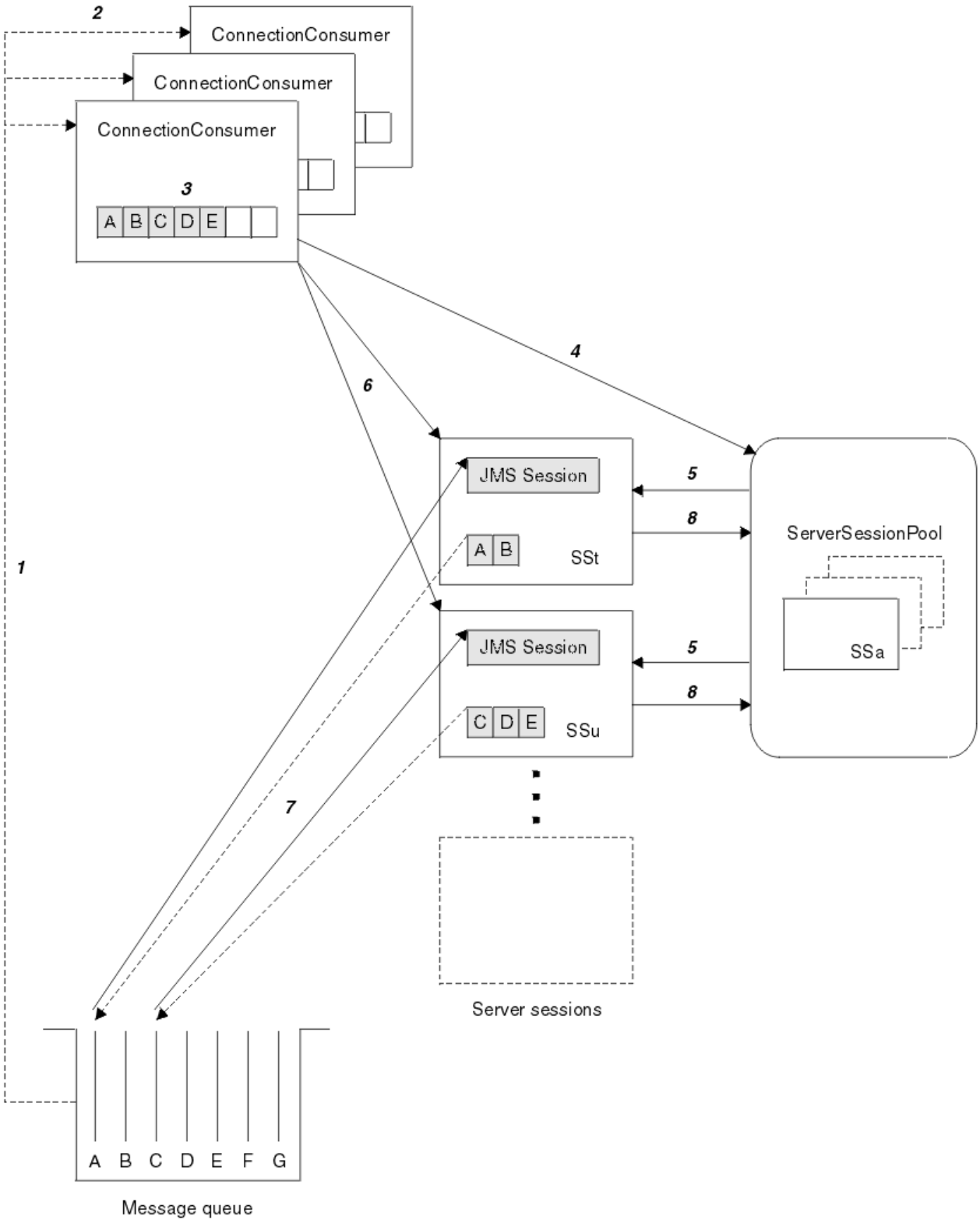


図 *45. ServerSessionPool* および *ServerSession* 機能

1. ConnectionConsumer は、キューからメッセージ参照を取得します。
2. それぞれの ConnectionConsumer は特定のメッセージ参照を選択します。
3. ConnectionConsumer バッファーは、選択されたメッセージ参照を保持します。
4. ConnectionConsumer は、ServerSessionPool から 1 つ以上の ServerSession を要求します。
5. ServerSession は ServerSessionPool から割り振られます。

6. ConnectionConsumer は、ServerSession にメッセージ参照を割り当て、ServerSession スレッドの実行を開始します。
7. 各 ServerSession は、キューからその参照されたメッセージを取得し、JMS セッションに関連付けられている MessageListener から onMessage メソッドにそれらを渡します。
8. その処理を完了した後、ServerSession はプールに戻されます。

アプリケーション・サーバーは、ServerSessionPool および ServerSession 機能を提供します。

z/OS CICS Liberty JVM サーバーでの IBM MQ classes for JMS の使用

IBM MQ 9.1.0 以降、CICS Liberty JVM サーバーで実行中の Java プログラムは、IBM MQ classes for JMS を使用して IBM MQ にアクセスすることができます。

IBM MQ 9.1.0 以降のバージョンの IBM MQ リソース・アダプターを使用している必要があります。 このリソース・アダプターは Fix Central (446 ページの『Liberty へのリソース・アダプターのインストール』を参照) から入手できます。

CICS 5.3 以降で使用可能な Liberty プロファイル JVM には、2 つのフレーバーがあります。 IBM MQ に対して可能な接続のタイプは、以下のように制限されます。

CICS Liberty 標準

- IBM MQ リソース・アダプターは、IBM MQ の現在サポートされているすべてのバージョンに CLIENT モードで接続できます。
- IBM MQ リソース・アダプターは、同じ CICS 領域から同じキュー・マネージャーへの CICS 接続 (アクティブな CICS MQCONN リソース定義) がない場合に、BINDINGS モードで任意のサービス中バージョンの IBM MQ for z/OS に接続できます。

CICS Liberty 統合

- IBM MQ リソース・アダプターは、IBM MQ の現在サポートされているすべてのバージョンに CLIENT モードで接続できます。
- BINDINGS モード接続はサポートされていません。

システムのセットアップと構成について詳しくは、CICS 資料の Liberty JVM サーバーでの IBM MQ classes for JMS の使用 を参照してください。

IMS での IBM MQ classes for JMS/ Jakarta Messaging の使用

IMS 環境内での標準ベースのメッセージング・サポートは、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用して提供されます。

各企業で使用している IMS システムのシステム要件を確認してください。 詳しくは、 IMS 15.2 を参照してください。

この一連のトピックでは、 IMS 環境で IBM MQ classes for JMS をセットアップする方法、およびクラシック (JMS 1.1) インターフェースと簡易 (JMS 2.0) インターフェースを使用する場合に適用される API の制約事項について説明します。 API 固有の情報のリストについては、346 ページの『JMS API の制約事項』を参照してください。

注 : 同様の制約が既存の (JMS 1.0.2) ドメイン固有のインターフェースにも当てはまります。しかし、それらについてはここでは特に説明しません。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

サポートされる IMS に依存する地域

以下の依存する領域タイプがサポートされています。

- MPR
- BMP
- IFP
- JMP 31 および 64 ビットの Java 仮想マシン (JVM)
- JBP 31 および 64 ビットの JVM

以下のトピックで特に言及されていない限り、 IBM MQ classes for JMS と IBM MQ classes for Jakarta Messaging はすべての領域タイプで同じように動作します。

サポートされる Java Virtual Machines

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging には、 IBM Runtime Environment、Java Technology Edition 8 が必要です。 IBM Semeru Runtime Certified Edition for z/OS バージョン 11 はサポートされていません。

その他の制約事項

IMS 環境で IBM MQ classes for JMS を使用する場合は、以下の制約事項が適用されます。

- クライアント・モード接続はサポートされていません。
- IBM MQ メッセージング・プロバイダー `Normal`・モードを使用した IBM MQ 8.0 キュー・マネージャーへの接続のみがサポートされます。

  接続ファクトリーの **PROVIDERVERSION** 属性は、指定されていないか、7 以上の値でなければなりません。
- XA 接続ファクトリー (`com.ibm.mq.jms.MQXAConnectionFactory` など) の使用はサポートされていません。

関連タスク

IBM MQ の定義 IMS

IBM MQ classes for JMS/Jakarta Messaging で使用するための *IMS* アダプターのセットアップ

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、他のプログラミング言語で使用されているものと同じ IBM MQ-IMS アダプターを使用します。 このアダプターは、IMS 外部サブシステム接続機能 (ESAF) を使用します。

始める前に

以下の手順を実行する前に、 IMS アダプターのセットアップの説明に従って、関連するキュー・マネージャー、および IMS 制御領域と従属領域用に IMS アダプターを構成する必要があります。

重要 : 動的スタブを他の目的で必要とする場合を除いて、動的スタブの作成方法を示す手順を実行する必要はありません。

IMS アダプターを構成した後、以下の手順を実行します。

手順

1. 従属領域の JCL (DFSJVMEV など) に含まれる ENVIRON パラメーターで参照される IMS PROCLIB のメンバー内の LIBPATH 変数を更新して、 それに IBM MQ classes for JMS ネイティブ・ライブラリーが含まれるようにします。

   これは、`libmqjims.so` を含む zFS ディレクトリーです。 例えば、DFSJVMEV は以下のようになります。最後の行は、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging ネイティブ・ライブラリーを含むディレクトリーです。

```
LIBPATH=>
/java/latest/bin/j9vm:>
```

```
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. java.class.path オプションを更新して、 IMS 従属領域によって使用される JVM のクラスパスに IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を追加します。
   これを実行するには、IMS PROCLIB データ・セットの DFSJVMMS メンバーの指示に従います。

   例えば、以下を使用することができます。太字の行は更新を示しています。

JM 3.0 V 9.3.0 V 9.3.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0

```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   注 : IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を含むディレクトリーには多数の異なる jar ファイルがありますが、必要なのは com.ibm.mq.allclient.jar (JMS 2.0) または com.ibm.mq.jakarta.client.jar (Jakarta Messaging 3.0)のみです。

3. IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging を使用するすべての IMS 従属領域を停止して再始動します。

次のタスク

接続ファクトリーと宛先を作成して構成します。

接続ファクトリーと宛先の IBM MQ 実装のインスタンス化には 3 つのアプローチがあります。 詳細は 207 ページの『接続ファクトリーおよび宛先の作成および構成』 を参照してください。

IMS 環境では、これら 3 つのアプローチがすべて有効です。

関連概念

IMS アダプターのセットアップ

IBM MQ の定義 IMS

トランザクション動作

IMS 環境で IBM MQ classes for JMS によって送受信されるメッセージは、常に、現行タスクでアクティブな IMS 作業単位 (UOW) に関連付けられます。

その UOW は、com.ibm.ims.dli.tm.Transaction オブジェクトのインスタンスで、コミット・メソッドまたはロールバック・メソッドを呼び出すことによってのみ完了できます。 また、IMS タスクが正常に終了する場合にも完了しますが、その場合 UOW は暗黙的にコミットされます。 IMS タスクが異常終了した場合、UOW はロールバックされます。

この結果として、Connection.createSession メソッドまたは ConnectionFactory.createContext メソッドを呼び出す際に **transacted** 引数と **acknowledgeMode** 引数の値は無視されます。 また、以下のメソッドはサポートされていません。 以下のメソッドを呼び出すと、セッション・ケースで IllegalStateException になります。

- javax.jms.Session.commit()
- javax.jms.Session.recover()
- javax.jms.Session.rollback()

また、 JMS コンテキスト・ケースの IllegalStateRuntimeSession は、以下のようになります。

- javax.jms.JMSContext.commit()
- javax.jms.JMSContext.recover()

- javax.jms.JMSContext.rollback()

この動作の例外が 1 つあります。 セッションまたは JMS コンテキストが以下の手段のいずれかを使って作成される場合:

- Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)
- Connection.createSession(Session.AUTO_ACKNOWLEDGE)
- ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)

そのセッションの動作、または JMS コンテキストは、次のようになります。

- 送信されるメッセージはすべて、IMS UOW の外で送信されます。 つまり、ターゲット宛先で即時に、または指定された送達遅延間隔が完了したときに使用可能になります。
- 非持続メッセージは、セッションまたは JMS コンテキストを作成した接続ファクトリーで SYNCPOINTALLGETS プロパティーが指定されていない限り、 IMS UOW の外部で受信されます。
- 永続メッセージは常に IMS UOW 内で受信されます。

これは例えば、UOW がロールバックするときでも監査メッセージをキューに書き込む場合などに役立ちます。

IMS 同期点の影響

IBM MQ classes for JMS は、ESAF を使用する既存の IBM MQ アダプター・サポートに基づいて構築されます。 つまり、同期点の発生時にすべての開いたハンドルが IMS アダプターによって閉じられるなどの、文書化されている動作が適用されます。

詳しくは、 71 ページの『IMS アプリケーションにおける同期点』 を参照してください。

この点を例示するために、JMP 環境で実行している次のコードを検討します。 mp.send() を 2 回目に呼び出すと、 messageQueue.getUnique(inputMessage) コードによってすべてのオープン IBM MQ 接続とオブジェクト・ハンドルがクローズされるため、 JMSException になります。

類似の動作は、getUnique() 呼び出しが Transaction.commit() で置き換えられたときにも生じますが、 Transaction.rollback() が使用されたときには生じません。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

このシナリオで使用する正しいコードは以下のとおりです。 この場合、 getUnique()を呼び出す前に IBM MQ への接続が閉じられます。 その後、別のメッセージを送信するために、接続とセッションが再作成されます。

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");
```

```
Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

IMS アダプターを使用する際の考慮事項

以下の制約事項を周知している必要があります。 各キュー・マネージャーに対して接続ハンドルは 1 つしか使用できません。 JMS とネイティブ・コードの両方を使用する場合、 IBM MQ との対話には影響があります。 接続認証と許可への制限事項もあります。

各キュー・マネージャーの 1 つの接続ハンドル

IMS 従属領域では、特定のキュー・マネージャーへの接続ハンドルは一度に 1 つしか使用できません。 同じキュー・マネージャーに接続する後続のすべての試行では、既存のハンドルを再使用します。

この動作によって、 IBM MQ classes for JMS のみを使用するアプリケーションで問題が発生することはありませんが、COBOL や C などの言語で作成されたネイティブ・コードで IBM MQ classes for JMS と MQI の両方を使用すると、 IBM MQ と対話するアプリケーションで問題が発生する可能性があります。

JMS とネイティブ・コードの両方を使用する場合の IBM MQ との対話の影響

IBM MQ 機能を使用する Java コードとネイティブ・コードをインターリービングするとき、およびネイティブ・コードまたは Java コードのいずれかを終了する前に IBM MQ への接続が閉じられないときに、問題が発生する可能性があります。

例えば、以下の疑似コードでは、キュー・マネージャーへの接続ハンドルは、最初に IBM MQ classes for JMS を使用して Java コードで確立されます。 接続ハンドルが COBOL コードで再使用され、MQDISC への呼び出しによって無効化されます。

次に IBM MQ classes for JMS が接続ハンドルを使用すると、MQRC_HCONN_ERROR の理由コードによる JMSException が発生します。

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

その他にも MQRC_HCONN_ERROR が発生する可能性のある類似した使用パターンが存在します。

ネイティブ・コードと Java コードの間で IBM MQ 接続ハンドルを共有することはできますが (例えば、前の例は、MQDISC 呼び出しがなかった場合に機能します)、ベスト・プラクティスは、 Java からネイティブ・コードに変更する前、またはその逆に変更する前に接続ハンドルを閉じることです。

接続の認証と許可

JMS 仕様では、接続または JMS コンテキスト・オブジェクトを作成する際の認証と許可でユーザー名とパスワードを指定することが許されています。

これは、IMS 環境ではサポートされません。 ユーザー名とパスワードを指定して接続を作成しようとすると、JMS Exception がスローされます。 ユーザー名とパスワードを指定して JMS コンテキストを作成しようとすると、JMSRuntimeException がスローされます。

代わりに、 IMS 環境から IBM MQ に接続する際に、認証と許可のための既存のメカニズムを使用する必要があります。

詳しくは、z/OS でのセキュリティーのセットアップを参照してください。 特に、使用できるユーザー ID について説明したセキュリティー検査のためのユーザー ID を参照してください。

関連タスク
z/OS でのセキュリティーのセットアップ

JMS API の制約事項

JMS 仕様の観点からは、IBM MQ classes for JMS は IMS を Java EE または Jakarta EE 準拠のアプリケーション・サーバーとして扱います。このアプリケーション・サーバーでは、常に JTA トランザクションが進行中です。

例えば、IMS では javax.jms.Session.commit() を呼び出すことができません。JMS 仕様では、JTA トランザクションが進行中の間は、JEE EJB または Web コンテナーでこれを呼び出せないことになっているからです。

これにより、 343 ページの『トランザクション動作』 で説明されている制約事項に加えて、 JMS API に以下の制約事項が生じます。

クラシック API の制約事項

- javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int) は常に JMSException をスローします。
- javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) は常に JMSException をスローします。
- 既に接続の既存のセッションがアクティブの場合、javax.jms.Connection.createSession の 3 つのバリアントすべては常に、JMSException をスローします。
- javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) は常に JMSException をスローします。
- javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int) は常に JMSException をスローします。
- javax.jms.Connection.setClientID() は常に JMSException をスローします。
- javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener) は常に JMSException をスローします。
- javax.jms.Connection.stop() は常に JMSException をスローします。
- javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener) は常に JMSException をスローします。
- javax.jms.MessageConsumer.getMessageListener() は常に JMSException をスローします。

- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` は常に `JMSException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` は常に `JMSException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` は常に `JMSException` をスローします。
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener)` は常に `JMSException` をスローします。
- `javax.jms.Session.run()` は常に `JMSRuntimeException` をスローします。
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` は常に `JMSException` をスローします。
- `javax.jms.Session.getMessageListener()` は常に `JMSException` をスローします。

単純化 API の制約事項

- `javax.jms.JMSContext.createContext(int)` は常に `JMSRuntimeException` をスローします。
- `javax.jms.JMSContext.setClientID(String)` は常に `JMSRuntimeException` をスローします。
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` は常に `JMSRuntimeException` をスローします。
- `javax.jms.JMSContext.stop()` は常に `JMSRuntimeException` をスローします。
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` は常に `JMSRuntimeException` をスローします。

IBM MQ classes for Java の使用

Java 環境で IBM MQ を使用します。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

注:

Stabilized IBM では、IBM MQ classes for Java に対してこれ以上拡張機能を提供することはありません。また、IBM MQ 8.0 で出荷されたレベルで機能的に固定化されています。 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、新機能は追加されず、機能拡張の要求も拒否されます。 完全なサポートとは、欠陥が見つかった場合、IBM MQ システム要件の変更によって必要とされる変更と一緒に修正されることを意味します。

IBM MQ classes for Java は IMS ではサポートされません。

IBM MQ classes for Java は WebSphere Liberty ではサポートされません。 IBM MQ Liberty メッセージング・フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、これらを使用してはいけません。 詳細については、Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。

IBM MQ classes for Java は、Java アプリケーションが IBM MQ リソースへのアクセスに使用できる 3 つの代替 API の 1 つです。 その他の API は以下のとおりです。

- JM 3.0 V 9.3.0 V 9.3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

詳細については、86 ページの『Java からの IBM MQ へのアクセス-API の選択』を参照してください。

IBM MQ 9.3 以降、IBM MQ classes for Java は Java 8 を使用して構築されます。 Java 8 ランタイム環境は、以前のバージョンのクラス・ファイルの実行をサポートします。

IBM MQ classes for Java は、メッセージ・キュー・インターフェース (MQI)、ネイティブ IBM MQ API をカプセル化し、 IBM MQ への C++ および .NET インターフェースと同様のオブジェクト・モデルを使用します。

プログラマブル・オプションにより、IBM MQ classes for Java は、次のいずれかの方法で IBM MQ に接続できます。

- クライアント・モードで、伝送制御プロトコル/インターネット・プロトコル (TCP/IP) を使用する IBM MQ MQI client として接続。
- バインディング・モードで、 Java Native Interface (JNI) を使用して IBM MQ に直接接続

注 : IBM MQ classes for Java は自動クライアント再接続をサポートしていません。

クライアント・モード接続

IBM MQ classes for Java アプリケーションは、サポートされているいずれのキュー・マネージャーにもクライアント・モードを使用して接続できます。

クライアント・モードでキュー・マネージャーに接続する際には、キュー・マネージャーが実行されているのと同じシステムか、または別のシステムで IBM MQ classes for Java アプリケーションを実行できます。 いずれの場合も、IBM MQ classes for Java は、TCP/IP を介してキュー・マネージャーに接続します。

クライアント・モード接続を使用するアプリケーションの作成方法について詳しくは、373 ページの『IBM MQ classes for Java の接続モード』を参照してください。

バインディング・モード接続

バインディング・モードで使用すると、IBM MQ classes for Java は、ネットワーク経由で通信するのではなく、Java Native Interface (JNI) を使用して、既存のキュー・マネージャー API を直接呼び出します。 大部分の環境において、クライアント・モードで接続するよりもバインディング・モードで接続した方が TCP/IP 通信の負荷を回避でき、IBM MQ classes for Java アプリケーションのパフォーマンスが向上します。

IBM MQ classes for Java を使用してバインディング・モードで接続するアプリケーションは、接続先のキュー・マネージャーと同じシステムで実行されなければなりません。

IBM MQ classes for Java アプリケーションの実行に使用される Java ランタイム環境は、 IBM MQ classes for Java ライブラリーをロードするように構成する必要があります。詳しくは、 358 ページの『IBM MQ classes for Java ライブラリー』 を参照してください。

バインディング・モード接続を使用するアプリケーションの作成方法について詳しくは、373 ページの『IBM MQ classes for Java の接続モード』を参照してください。

関連概念

IBM MQ Java 言語インターフェース

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』

IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

関連タスク

IBM MQ classes for Java アプリケーションのトレース

Java および JMS の問題のトラブルシューティング

IBM MQ classes for Java を使用する理由

Java アプリケーションは、IBM MQ classes for Java または IBM MQ classes for JMS のいずれかを使用して、IBM MQ リソースにアクセスできます。

注 : V 9.3.0 V 9.3.0 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、新しいアプリケーションでは IBM MQ classes for Jakarta Messaging を使用する必要があります。 最近 IBM MQ に追加された機能 (非同期コンシュームや自動再接続など) は、 IBM MQ classes for Java では使用できませんが、 IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging で使用できます。 詳しくは、85 ページの『IBM MQ classes for JMS を使用する理由』および83 ページの『IBM MQ classes for Jakarta Messaging を使用する理由』を参照してください。

注 :

Stabilized IBM では、IBM MQ classes for Java に対してこれ以上拡張機能を提供することはありません。また、IBM MQ 8.0 で出荷されたレベルで機能的に固定化されています。 IBM MQ classes for Java を使用する既存のアプリケーションは引き続き完全にサポートされますが、新機能は追加されず、機能拡張の要求も拒否されます。 完全なサポートとは、欠陥が見つかった場合、IBM MQ システム要件の変更によって必要とされる変更と一緒に修正されることを意味します。

IBM MQ classes for Java は IMS ではサポートされません。

IBM MQ classes for Java は WebSphere Liberty ではサポートされません。 IBM MQ Liberty メッセージング・フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、これらを使用してはいけません。 詳細については、Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。

関連概念

86 ページの『Java からの IBM MQ へのアクセス-API の選択』
IBM MQ は、3 つの Java 言語インターフェースを提供します。

IBM MQ classes for Java の前提条件

IBM MQ classes for Java を使用するためには、特定の他のソフトウェア製品を必要とします。

IBM MQ classes for Java の前提条件については、IBM MQ のシステム要件の Web ページを参照してください。

IBM MQ classes for Java アプリケーションを開発するには、Java Development Kit (JDK) が必要です。 ご使用のオペレーティング・システムでサポートされている JDK の詳細については、IBM MQ のシステム要件という資料に記載されています。

IBM MQ classes for Java アプリケーションを実行するには、以下のソフトウェア・コンポーネントが必要です。

- キュー・マネージャーに接続するアプリケーションの場合は、IBM MQ キュー・マネージャー
- アプリケーションを実行する各システムに対して、Java Runtime Environment (JRE)。 適合する JRE が IBM MQ とともに提供されます。
- IBM i IBM i の場合は、QShell (オペレーティング・システムのオプション 30)
- z/OS z/OS の場合は、z/OS UNIX System Services (z/OS UNIX)

TLS 接続で FIPS 140-2 認証の暗号モジュールを使用するよう要求する場合は、IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) が必要です。 IBM JDK および JRE バージョン 1.4.2 以降にはすべて、IBMJSSEFIPS が含まれています。

ご使用のオペレーティング・システム上の Java 仮想マシン (JVM) および TCP/IP 実装によってサポートされる IBM MQ classes for Java アプリケーション IPv6 の場合 で、 Internet Protocol バージョン 6 (IPv6) アドレスを使用できます。

Java EE 内での IBM MQ classes for Java アプリケーションの実行

Java EE で IBM MQ classes for Java を使用する前に考慮しなければならない制約事項と設計上の考慮事項がいくつかあります。

IBM MQ classes for Java には、Java Platform, Enterprise Edition (Java EE) 環境内で使用する場合、制限があります。 Java EE 環境内で実行される IBM MQ classes for Java アプリケーションを設計、実装、および

管理する際に考慮する必要がある追加の考慮事項もあります。 これらの制限および考慮事項の概要は以下のセクションで説明します。

JTA トランザクションの制限

IBM MQ classes for Java を使用するアプリケーションでサポートされるトランザクション・マネージャーは、IBM MQ 自体のみです。 JTA 制御下のアプリケーションは IBM MQ classes for Java を利用できますが、これらのクラスを通して実行された作業は JTA 作業単位によっては制御されません。 代わりに、それらの作業は、JTA インターフェースを通してアプリケーション・サーバーにより管理される作業単位とは別のローカル作業単位を形成します。 特に、JTA トランザクションのロールバックによって、送受信されたメッセージのロールバックが生じることはありません。 この制限は、アプリケーションまたは Bean によって管理されるトランザクション、コンテナーによって管理されるトランザクション、およびすべての Java EE コンテナーに適用されます。 メッセージング処理を直接 IBM MQ で、アプリケーション・サーバーによって調整されるトランザクション内で実行する場合は、代わりに IBM MQ classes for JMS を使用しなければなりません。

スレッドの作成

IBM MQ classes for Java は、さまざまな操作のためにスレッドを内部で作成します。 例えば、BINDINGS モードで実行してローカル・キュー・マネージャーに直接呼び出しを行う場合、呼び出しは IBM MQ classes for Java によって内部で作成される「ワーカー」スレッドで実行されます。 例えば、接続プールから使用されていない接続を消去したり、終了したパブリッシュ/サブスクライブ・アプリケーションのサブスクリプションを除去したりする他のスレッドも内部で作成されることがあります。

一部の Java EE アプリケーション (例えば EJB および Web コンテナーで実行するアプリケーション) では、新規スレッドを作成してはなりません。 その場合、すべての作業はアプリケーション・サーバーによって管理されるメイン・アプリケーション・スレッドで実行されなければなりません。 アプリケーションが IBM MQ classes for Java を使用する場合、アプリケーション・サーバーはアプリケーション・コードと IBM MQ classes for Java コードを区別できない場合があり、上記で説明したスレッドによって、アプリケーションがコンテナー仕様に準拠しなくなります。 IBM MQ classes for JMS はこれらの Java EE 仕様に違反しないため、こちらを使用する必要があります。

セキュリティー制限

アプリケーション・サーバーによって実装されているセキュリティー・ポリシーによっては、新しい制御スレッドの作成や操作など、IBM MQ classes for Java API によって実行される特定の操作が実行できなくなる場合があります (前出のセクションの説明を参照)。

例えば、アプリケーション・サーバーは通常、デフォルトで Java Security が無効にされた状態で実行され、アプリケーション・サーバー固有の構成で Java Security を有効にすることができます (アプリケーション・サーバーによっては、Java Security で使用されるポリシーをより詳細に構成することができるものもあります)。 Java セキュリティーが有効にされた場合、IBM MQ classes for Java は、アプリケーション・サーバー用に定義された Java セキュリティー・ポリシーのスレッド化規則に違反する可能性があり、正常に機能するために必要なすべてのスレッドを API が作成できなくなる可能性があります。 スレッド管理の問題を回避するために、 Java セキュリティーが有効になっている環境では IBM MQ classes for Java の使用はサポートされていません。

アプリケーション独立の考慮事項

アプリケーションを Java EE 環境内で実行する目的は、アプリケーション独立という利点を得ることにあります。 IBM MQ classes for Java の設計と実装は、 Java EE 環境の前に行われます。 IBM MQ classes for Java をアプリケーション独立の概念をサポートしない方法で使用することも可能です。 この領域での考慮事項の具体的な例には次のようなものがあります。

- 以下のような、MQEnvironment クラス内での静的 (JVM プロセス全体) 設定の使用
  - 接続 ID および認証に使用されるユーザー ID およびパスワード
  - クライアント接続に使用されるホスト名、ポート、およびチャネル
  - 保護されたクライアント接続用の TLS 構成

1 つのアプリケーションの利点のために MQEnvironment プロパティーを変更した場合、同じプロパティーを利用する他のアプリケーションも影響されます。 Java EE などのマルチアプリケーション環境で実行する場合、各アプリケーションは、プロセス全体の MQEnvironment クラスで構成されたプロパティーをデフォルトとして使用するのではなく、プロパティーの特定のセットで MQQueueManager オブジェクトを作成することにより、個々の独特な構成を使用する必要があります。

- MQEnvironment クラスは、同じ JVM プロセス内で IBM MQ classes for Java を使用するアプリケーションすべてにグローバルに作用するいくつかの静的メソッドを導入します。特定のアプリケーションに対してこの動作をオーバーライドする方法はありません。 この例には、以下のものが含まれます。
  - 鍵ストアの場所などの、TLS プロパティーの構成
  - クライアント・チャネル出口の構成
  - 診断トレースの使用可能化または使用不可化
  - キュー・マネージャーへの接続の使用を最適化するために使用されるデフォルト接続プールの管理

  このようなメソッドを呼び出すと、同じ Java EE 環境で実行されるアプリケーションすべてが影響を受けます。
- 接続プーリングは、同じキュー・マネージャーへの複数の接続を行うプロセスを最適化するために有効にされています。 デフォルト接続プール・マネージャーはプロセス規模であり、複数のアプリケーションによって共用されます。 したがって、接続プール構成への変更 (例えば MQEnvironment.setDefaultConnectionManager() メソッドを使用して、1 つのアプリケーションのデフォルト接続マネージャーを置換するなど) は、同じ Java EE アプリケーション・サーバー内で実行されるその他のアプリケーションに影響します。
- TLS は、IBM MQ classes for Java を使用するアプリケーション用に、MQEnvironment クラスおよび MQQueueManager オブジェクト・プロパティーを使用して構成されます。 アプリケーション・サーバー自体の管理セキュリティー構成には統合されません。 IBM MQ classes for Java を適切に構成することによって必要なレベルのセキュリティーが提供されるようにし、アプリケーション・サーバーの構成を使用しないようにしてください。

バインディング・モードの制約事項

IBM MQ と WebSphere Application Server を同じマシンにインストールして、キュー・マネージャーのメジャー・バージョンと WebSphere Application Server に付属の IBM MQ リソース・アダプター (RA) のメジャー・バージョンが異なるようにすることができます。

キュー・マネージャーとリソース・アダプターの主要なバージョンが異なる場合、バインディング接続は使用できません。 リソース・アダプターを使用した WebSphere Application Server からキュー・マネージャーへの接続はすべて、クライアント・タイプ接続を使用する必要があります。 バージョンが同じ場合は、バインディング接続を使用できます。

IBM MQ classes for Java の文字ストリング変換

IBM MQ classes for Java では、文字ストリング変換に CharsetEncoders と CharsetDecoders を直接使用します。 文字ストリング変換のデフォルト動作は、2 つのシステム・プロパティーを使用して構成できます。マップできない文字を含むメッセージの処理は、`com.ibm.mq.MQMD` を使用して構成できます。

IBM MQ 8.0 の前は、IBM MQ classes for Java のストリング変換は、`java.nio.charset.Charset.decode(ByteBuffer)` メソッドおよび `Charset.encode(CharBuffer)` メソッドを呼び出すことによって行われていました。

このいずれかのメソッドを使用すると、誤った形式のデータまたは変換不能なデータに対してデフォルトの置換が行われます (REPLACE)。 この動作により、アプリケーションでのエラーが覆い隠され、予期しない文字 (? など) が変換データに現れることがあります。

IBM MQ 8.0 から、以前よりも効率的にそのような問題を検出するために、IBM MQ classes for Java は CharsetEncoders と CharsetDecoders を直接使用して、誤った形式のデータおよび変換不能なデータの処理を明示的に構成します。 デフォルト動作は変更され、適切な `MQException` をスローして、その種の問題を REPORT するようになりました。

構成

UTF-16 (Java で使用される文字表現) から固有文字セット (UTF-8 など) への変換を *encoding* と呼ぶのに対し、反対方向への変換を *decoding* と呼びます。

デコードでは CharsetDecoders のデフォルト動作に従い、例外をスローしてエラーを報告します。

今回、エンコードとデコードの両方で 1 つの設定を使用して java.nio.charset.CodingErrorAction を指定し、エラー処理を制御します。 エンコード時に置換バイトを制御するためにもう 1 つ別の設定が使用されます。 デコード操作では、デフォルトの Java 置き換えストリングが使用されます。

IBM MQ classes for Java における変換不能データの処理の構成

IBM MQ 8.0 以降、com.ibm.mq.MQMD には次の 2 つのフィールドが含まれます。

byte[] unMappableReplacement
入力文字を変換できず、REPLACE が指定されている場合に、エンコードされたストリングに書き込まれるバイト・シーケンス。

デフォルト: "?".getBytes()
デフォルトの Java 置き換えストリングがデコード操作で使用されます。

java.nio.charset.CodingErrorAction unmappableAction
エンコードおよびデコードの際に変換不能データに対して取られるアクションを指定します。

デフォルト: CodingErrorAction.REPORT;

システム・デフォルトの設定用システム・プロパティー

IBM MQ 8.0 以降、文字ストリングの変換についてのデフォルト動作を構成するために使用可能な 2 つの Java システム・プロパティーがあります。

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
エンコードおよびデコードの際に変換不能データに対して取られるアクションを指定します。 値は REPORT、REPLACE、または IGNORE です。

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
エンコード操作で文字をマップできないときに適用する置換バイトを設定または取得します。デコード操作では、デフォルトの Java 置き換えストリングが使用されます。

Java 文字と固有バイト表現の間の混乱を避けるため、com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement は固有文字セットの置換バイトを表す 10 進数として指定する必要があります。

例えば、固有バイトとしての ? の 10 進値は、固有文字セットが ASCII ベース (ISO-8859-1 など) の場合は 63 であるのに対し、固有文字セットが EBCDIC の場合は 111 になります。

注 : MQMD オブジェクトまたは MQMessage オブジェクトに **unmappableAction** フィールドまたは **unMappableReplacement** フィールドが設定されている場合、それらのフィールドの値が Java システム・プロパティーよりも優先されることに留意してください。 したがって、必要に応じてメッセージごとに、Java システム・プロパティーで指定された値をオーバーライドすることが可能です。

関連概念
140 ページの『IBM MQ classes for JMS の文字ストリング変換』
IBM MQ classes for JMS では、文字ストリング変換に CharsetEncoders と CharsetDecoders を直接使用します。 文字ストリング変換のデフォルト動作は、2 つのシステム・プロパティーを使用して構成できます。マップできない文字を含むメッセージの処理は、UnmappableCharacterAction と置換バイトの設定用メッセージ・プロパティーを使用して構成できます。

IBM MQ classes for Java のインストールと構成

このセクションでは、IBM MQ classes for Java のインストール時に作成されるディレクトリーとファイルについて説明し、インストール後に IBM MQ classes for Java を構成する方法を示します。

IBM MQ classes for Java でインストールされる内容

最新バージョンの IBM MQ classes for Java が IBM MQ と共にインストールされます。 そうするには、デフォルトのインストール・オプションをオーバーライドする必要があります。

IBM MQ のインストールについて詳しくは、以下のトピックを参照してください。

- Multi IBM MQ のインストール
- z/OS IBM MQ for z/OS 製品のインストール

IBM MQ classes for Java は、Java アーカイブ (JAR) ファイル、`com.ibm.mq.jar`、および `com.ibm.mq.jmqi.jar` に含まれています。

プログラマブル・コマンド・フォーマット (PCF) などの標準メッセージ・ヘッダーのサポートは、JAR ファイル `com.ibm.mq.headers.jar` に含まれています。

プログラマブル・コマンド・フォーマット (PCF) のサポートは、JAR ファイル `com.ibm.mq.pcf.jar` に含まれています。

注 : アプリケーション・サーバー内では IBM MQ classes for Java を使用することは勧められていません。 この環境で稼働する場合に適用される制約事項については、349 ページの『Java EE 内での IBM MQ classes for Java アプリケーションの実行』を参照してください。 詳しくは、Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。

重要 : 353 ページの『IBM MQ classes for Java 再配置可能 JAR ファイル』で説明する再配置可能 JAR ファイルを除いて、IBM MQ classes for Java JAR ファイルまたはネイティブ・ライブラリーを他のマシンにコピーすることや、IBM MQ classes for Java がインストールされているマシン上の異なるロケーションにコピーすることは、サポートされていません。

IBM MQ classes for Java 再配置可能 *JAR* ファイル

再配置可能 JAR ファイルは、IBM MQ classes for Java を実行する必要のあるシステムに移動することができます。

重要 :

- 再配置可能 JAR ファイルで説明する再配置可能 JAR ファイルを除いて、IBM MQ classes for Java JAR ファイルまたはネイティブ・ライブラリーを他のマシンにコピーすることや、IBM MQ classes for Java がインストールされているマシン上の異なるロケーションにコピーすることは、サポートされていません。
- クラス・ローダーの競合を回避するため、同じ Java ランタイム内の複数のアプリケーション内で再配置可能 JAR ファイルをバンドルすることは推奨されません。 このシナリオでは、IBM MQ 再配置可能 JAR ファイルを Java ランタイムのクラスパスで使用できるようにすることを検討してください。
- WebSphere Application Server などの Java EE アプリケーション・サーバーにデプロイされたアプリケーション内には、再配置可能 JAR ファイルを組み込まないでください。 こうした環境では、代わりに IBM MQ リソース・アダプターをデプロイして使用する必要があります。これには IBM MQ classes for Java が含まれるためです。 WebSphere Application Server には、IBM MQ リソース・アダプターが組み込まれているため、この環境に手動でデプロイする必要はありません。 また、IBM MQ classes for Java は WebSphere Liberty ではサポートされません。 詳細については、442 ページの『Liberty と IBM MQ リソース・アダプター』を参照してください。
- 再配置可能 JAR ファイルをアプリケーション内にバンドルする場合は、再配置可能 JAR ファイルに説明されているとおりに、すべての前提条件の JAR ファイルを必ず組み込むようにしてください。 また、IBM MQ classes for Java を最新の状態に維持し、既知の問題が修復されるように、バンドルされた JAR ファイルをアプリケーションの保守の一部として更新するための適切な手順も確保する必要があります。

再配置可能 JAR ファイル

IBM MQ classes for Java アプリケーションを実行するために必要な社内システムに、以下のファイルを移動できます。

- JMS 2.0 `com.ibm.mq.allclient.jar` 354 ページの『1』
- JM 3.0 V 9.3.0 V 9.3.0 `com.ibm.mq.jakarta.client.jar` 354 ページの『2』

- V 9.3.3 Removed com.ibm.mq.traceControl.jar
- V 9.3.5 bcpkix-jdk18on.jar 354 ページの『3』
- V 9.3.0 V 9.3.0 bcpkix-jdk15to18.jar 354 ページの『4』
- V 9.3.5 bcprov-jdk18on.jar 354 ページの『3』
- V 9.3.0 V 9.3.0 bcprov-jdk15to18.jar 354 ページの『4』
- V 9.3.5 bcutil-jdk18on.jar 354 ページの『3』
- V 9.3.0 V 9.3.0 bcutil-jdk15to18.jar 354 ページの『4』
- V 9.3.3 jackson-annotations.jar
- V 9.3.3 jackson-core.jar
- V 9.3.3 jackson-databind.jar
- org.json.jar

注:

1. JMS 2.0 と JMS 1.1
2. Jakarta Messaging 3.0
3. IBM MQ 9.3.5 からの Continuous Delivery
4. Long Term Support および Continuous Delivery (IBM MQ 9.3.5 より前)

Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイル

Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイルは必須です。 詳しくは、「AMS での非 IBM JREs のサポート」を参照してください。

V 9.3.5 IBM MQ 9.3.5 からの Continuous Delivery の場合、以下の JAR ファイルが必要です。

- bcpkix-jdk18on.jar
- bcprov-jdk18on.jar
- bcutil-jdk18on.jar

LTS IBM MQ 9.3.5 より前の Long Term Support および Continuous Delivery の場合、以下の JAR ファイルが必要です。

- bcpkix-jdk15to18.jar
- bcprov-jdk15to18.jar
- bcutil-jdk15to18.jar

org.json.jar

IBM MQ classes for Java アプリケーションが JSON 形式で CCDT を使用する場合、org.json.jar ファイルが必要です。

com.ibm.mq.allclient.jar および com.ibm.mq.jakarta.client.jar

ファイル com.ibm.mq.allclient.jar および com.ibm.mq.jakarta.client.jar には、 IBM MQ classes for JMS、 IBM MQ classes for Java、および PCF クラスとヘッダー・クラスが含まれています。 これらのファイルを新しい場所に移動する場合は、新しい IBM MQ フィックスパックでこの新しい場所を維持するための手順を必ず実行してください。 また、暫定修正を入手する場合は、ファイルの使用が IBM サポートに認識されていることを確認してください。

com.ibm.mq.allclient.jar ファイルまたは com.ibm.mq.jakarta.client.jar ファイルのバージョンを判別するには、以下のコマンドを使用します。

JM 3.0 V 9.3.0 V 9.3.0
```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
java -jar com.ibm.mq.allclient.jar
```

次の例は、このコマンドの出力例を示しています。

```
C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

jackson-annotations.jar、 jackson-core.jar 、および jackson-databind.jar

V 9.3.3

IBM MQ classes for Java アプリケーションがキュー・マネージャーへのセキュア TLS 接続を作成する場合は、3 つの Jackson JAR ファイルが必要です。

IBM MQ classes for Java のインストール・ディレクトリー
IBM MQ classes for Java のファイルとサンプルは、プラットフォームによって異なる場所にインストールされます。 IBM MQ とともにインストールされる Java ランタイム環境 (JRE) の場所も、プラットフォームによって異なります。

IBM MQ classes for Java ファイルのインストール・ディレクトリー

355 ページの表 49 に、IBM MQ classes for Java ファイルのインストール先を示します。

表 *49. IBM MQ classes for Java* のインストール・ディレクトリー

プラットフォーム	ディレクトリー
AIX AIX	*MQ_INSTALLATION_PATH*/java/ライブラリー
	/QIBM/ProdData/mqm/java/lib
Linux Linux	*MQ_INSTALLATION_PATH*/java/ライブラリー
Windows Windows	*MQ_INSTALLATION_PATH*¥java¥lib
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

サンプルのインストール・ディレクトリー

インストール検査プログラム (IVP) など、一部のサンプル・アプリケーションは IBM MQ に付属しています。 356 ページの表 50 に、サンプル・アプリケーションのインストール先を示します。 IBM MQ classes for Java サンプルは、`wmqjava` というサブディレクトリーにあります。 PCF サンプルは、`pcf` というサブディレクトリー内にあります。

表 *50.* サンプル・ディレクトリー

プラットフォーム	ディレクトリー
AIX AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i IBM i	/QIBM/ProdData/mqm/java/samples
Linux Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows Windows	*MQ_INSTALLATION_PATH*¥tools¥wmqjava¥
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

JRE のインストール・ディレクトリー

IBM MQ classes for JMS は Java 7 (以上) Java ランタイム環境 (JRE) を必要とします。 適合する JRE が IBM MQ と共にインストールされます。 356 ページの表 51 は、この JRE がインストールされる場所を示しています。 提供されたサンプルなど、この JRE を使用する Java プログラムを実行するには、明示的に *JRE_LOCATION*`/bin/java` を呼び出か、*JRE_LOCATION*`/bin` をお使いのプラットフォームの PATH 環境 (または同等の物) へ追加します。ここで、*JRE_LOCATION* は、356 ページの表 51 で与えられるディレクトリーです。

表 *51. JRE* のディレクトリー

プラットフォーム	ディレクトリー
AIX AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i IBM i	/QIBM/ProdData/mqm/java/jre
Linux Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows Windows	*MQ_INSTALLATION_PATH*¥java¥jre
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

IBM MQ classes for Java に関連する環境変数

IBM MQ classes for Java アプリケーションを実行する場合は、そのクラスパスに IBM MQ classes for Java ディレクトリーとサンプル・ディレクトリーが含まれている必要があります。

IBM MQ classes for Java アプリケーションを実行するには、クラス・パスに適切な IBM MQ classes for Java ディレクトリーが含まれている必要があります。 サンプル・アプリケーションを実行する場合は、クラス・パスに適切な samples ディレクトリーも組み込まなければなりません。 この情報は、 Java 呼び出しコマンドまたは **CLASSPATH** 環境変数で指定できます。

重要 : IBM MQ classes for Java を含めるための Java オプション `-Xbootclasspath` の設定は、サポートされていません。

357 ページの表 52 は、 IBM MQ classes for Java アプリケーション (サンプル・アプリケーションを含む) を実行するために各プラットフォームで使用する適切な **CLASSPATH** 設定を示しています。

表 *52. IBM MQ classes for Java* アプリケーションを実行するための ***CLASSPATH*** 設定 *(IBM MQ classes for Java* サンプル・アプリケーションを含む*)*

プラットフォーム	**CLASSPATH** 設定
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*¥Java¥lib¥com.ibm.mq.jar; *MQ_INSTALLATION_PATH*¥tools¥wmqjava¥samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R3M0/java/samples/pcf

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

`-Xlint` オプションを使用してコンパイルすると、`com.ibm.mq.ese.jar` が存在しないことを示す警告メッセージが表示される場合があります。 この警告は無視して構いません。 このファイルは、Advanced Message Security をインストールしている場合にのみ存在します。

IBM MQ classes for JMS に付属しているスクリプトは、以下の環境変数を使用します。

MQ_JAVA_DATA_PATH
この環境変数は、ログおよびトレース出力のディレクトリーを指定します。

MQ_JAVA_INSTALL_PATH
この環境変数は、 IBM MQ classes for Java インストール・ディレクトリーに示すように、 IBM MQ classes for Java がインストールされているディレクトリーを指定します。

MQ_JAVA_LIB_PATH
この環境変数は、 各プラットフォームの IBM MQ classes for Java ライブラリーの場所に示すように、 IBM MQ classes for Java ライブラリーが保管されるディレクトリーを指定します。 IBM MQ classes for Java に付属しているスクリプトの中には、IVTRun など、この環境変数を使用するものがあります。

Windows では、すべての環境変数がインストール時に自動的に設定されます。

AIX and Linux では、スクリプト `setjmsenv` (32 ビット JVM を使用している場合) または `setjmsenv64` (64 ビット JVM を使用している場合) を使用して、環境変数を設定することができます。 これらのスクリプトは *MQ_INSTALLATION_PATH*/java/bin ディレクトリーにあります。

IBM i では、環境変数 **QIBM_MULTI_THREADED** を Y に設定する必要があります。 それにより、単一スレッド・アプリケーションを実行する場合と同じようにマルチスレッド・アプリケーションを実行できます。 詳しくは、 Java および JMS を使用した IBM MQ のセットアップを参照してください。

IBM MQ classes for Java には Java 7 Java ランタイム環境 (JRE) が必要です。 IBM MQ とともにインストールされる適切な JRE の場所については、355 ページの『IBM MQ classes for Java のインストール・ディレクトリー』 を参照してください。

IBM MQ classes for Java ライブラリー
IBM MQ classes for Java ライブラリーの場所は、プラットフォームによって異なります。 アプリケーションの開始時にこの場所を指定します。

Java Native Interface (JNI) ライブラリーの位置を指定するには、**java** コマンドを以下の形式で使用して、アプリケーションを始動します。

```
java -Djava.library.path= library_path application_name
```

ここで、*library_path* は、JNI ライブラリーが含まれる IBM MQ classes for Java へのパスです。 358 ページの表 53 は、各プラットフォームでの IBM MQ classes for Java ライブラリーの場所を示しています。 この表では、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

表 *53.* 各プラットフォームでの *IBM MQ classes for Java* ライブラリーの場所

プラットフォーム	**IBM MQ classes for Java ライブラリーが含まれるディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Linux (x86 プラットフォーム)	*MQ_INSTALLATION_PATH*/java/ライブラリー
Linux (POWER、x86-64 および zSeries s390x プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Windows	*MQ_INSTALLATION_PATH*¥java¥lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*¥java¥lib64 (64 ビット・ライブラリー)
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (32 ビットおよび 64 ビット・ライブラリー)

注 :

1. Linux AIX AIX または Linux (Power プラットフォーム) では、32 ビット・ライブラリーまたは 64 ビット・ライブラリーを使用します。 64 ビット・ライブラリーは、64 ビット・プラットフォーム上の 64 ビット Java 仮想マシン (JVM) でアプリケーションを実行する場合にのみ使用してください。 それ以外の場合は 32 ビット・ライブラリーを使用してください。
2. Windows Windows では、 **java** コマンドで IBM MQ classes for Java ライブラリーの場所を指定する代わりに、PATH 環境変数を使用してそれらのライブラリーの場所を指定できます。
3. IBM i IBM i で IBM MQ classes for Java をバインディング・モードで使用するには、ライブラリー QMQMJAVA がライブラリー・リストにあることを確認してください。

4. z/OS では、32 ビットまたは 64 ビットのいずれかの Java 仮想マシン (JVM) を使用できます。 使用するライブラリーを指定する必要はありません。どの JNI ライブラリーをロードするかは、IBM MQ classes for Java によって自動的に判別されます。

関連概念

IBM MQ classes for Java の使用
IBM MQ classes for Java をインストールした後に、独自のアプリケーションを実行するためにインストール環境を構成できます。

IBM MQ classes for Java での OSGi のサポート
OSGi は、バンドルの形によるアプリケーションのデプロイメントをサポートするフレームワークを提供します。 3 つの OSGi バンドルが IBM MQ classes for Java の一部として提供されます。

OSGi は、汎用で、安全で、管理された Java フレームワークを提供します。これは、バンドルの形態で付属するアプリケーションのデプロイメントをサポートします。 OSGi 対応デバイスはバンドルをダウンロードしてインストールし、それらが不要になったら削除することができます。 このフレームワークでは、バンドルのインストールと更新を動的で拡張が容易な方法で管理します。

IBM MQ classes for Java には、以下の OSGi バンドルが組み込まれています。

com.ibm.mq.osgi.java_*version_number*.jar
アプリケーションが IBM MQ classes for Java を使用できるようにする JAR ファイル。

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、この JAR ファイルにより、アプリケーションは IBM MQ classes for JMS と IBM MQ classes for Java の両方を使用できます。また、PCF メッセージを処理するためのコードも含まれています。

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 JMS 2.0 の場合、この JAR ファイルにより、アプリケーションは IBM MQ classes for JMS と IBM MQ classes for Java の両方を使用できます。また、PCF メッセージを処理するコードも含まれています。

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、この JAR ファイルは `com.ibm.mq.jakarta.osgi.allclient_`*`version_number`*`.jar` の前提条件を提供します。

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 JMS 2.0 の場合、この JAR ファイルは `com.ibm.mq.osgi.allclient_`*`version_number`*`.jar` の前提条件を提供します。

ここで、 *`version_number`* は、インストールされている IBM MQ のバージョン番号です。

これらのバンドルは、IBM MQ インストール済み環境の `java/lib/OSGi` サブディレクトリーにインストールされるか、Windows 上の `java\lib\OSGi` フォルダーにインストールされます。

IBM MQ 8.0 以降では、あらゆる新しいアプリケーションのためにバンドル `com.ibm.mq.osgi.allclient_8.0.0.0.jar`、および `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` を使用します。 これらのバンドルを使用することで、同一 OSGi フレームワーク内で IBM MQ classes for JMS と IBM MQ classes for Java の両方を実行できないという制限がなくなります。 ただし、他の制限事項はすべて、従来どおり適用されます。 IBM MQ 8.0 より前のバージョンの IBM MQ の場合、IBM MQ classes for JMS または IBM MQ classes for Java のいずれかを使用するという制限が適用されます。

9 つの他のバンドルが IBM MQ インストール済み環境の `java/lib/OSGi` サブディレクトリーか、Windows の `java\lib\OSGi` フォルダーにインストールされます。 これらのバンドルは IBM MQ classes for JMS の一部であり、IBM MQ classes for Java バンドルがロードされている OSGi ランタイム環境にロードしてはなりません。 IBM MQ classes for Java OSGi バンドルを、既に IBM MQ classes for JMS バンドルがロードされている OSGi ランタイム環境にロードすると、IBM MQ classes for Java バンドルまたは IBM

MQ classes for JMS バンドルを使用するアプリケーションが実行される時に、以下の例に示すエラーが発生します。

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

IBM MQ classes for Java 用の OSGi バンドルは、OSGi リリース 4 仕様に合わせて記述されているため、OSGi リリース 3 環境では動作しません。

OSGi ランタイム環境が必要な DLL ファイルまたは共用ライブラリーを検出できるように、システム・パスまたはライブラリー・パスを正しく設定する必要があります。

IBM MQ classes for Java 用の OSGi バンドルを使用する場合、Java で記述されたチャネル出口クラスはサポートされません。OSGi などの複数クラス・ローダー環境にクラスをロードする際に固有の問題があるためです。 ユーザー・バンドルは IBM MQ classes for Java バンドルを認識できますが、IBM MQ classes for Java バンドルはユーザー・バンドルを認識できません。 結果として、IBM MQ classes for Java バンドル内で使用されるクラス・ローダーは、ユーザー・バンドル内のチャネル出口クラスをロードできません。

OSGi の詳細については、OSGi Alliance Web サイトをご覧ください。

z/OS *z/OS への IBM MQ classes for Java のインストール*

z/OS の場合、実行時に使用される STEPLIB に IBM MQ SCSQAUTH および SCSQANLE ライブラリーが含まれていなければなりません。

z/OS UNIX System Services からこれらのライブラリーを追加するには、`.profile` で以下の例に示すような行を使用して、`thlqual` を IBM MQ のインストール時に選択した高位データ・セット修飾子に置き換えます。

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

他の環境では一般的に、SCSQAUTH が STEPLIB 連結に含まれるように始動 JCL を編集する必要があります。

```
 STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
         DD DSN=thlqual.SCSQANLE,DISP=SHR
```

IBM MQ classes for Java 構成ファイル

IBM MQ classes for Java 構成ファイルは、IBM MQ classes for Java を構成するために使用されるプロパティーを指定します。

IBM MQ classes for Java 構成ファイルのフォーマットは、標準的な Java プロパティー・ファイルのフォーマットです。

IBM MQ classes for Java インストール・ディレクトリーの `bin` サブディレクトリーに、サンプル構成ファイル `mqjava.config` が提供されます。 このファイルは、サポートされているすべてのプロパティーとそのデフォルト値を文書化します。

注 : IBM MQ インストール環境を将来のフィックスパックにアップグレードすると、このサンプル構成ファイルは上書きされます。 そのため、アプリケーションで使用できるようにサンプル構成ファイルをコピーしておくことをお勧めします。

IBM MQ classes for Java 構成ファイルの名前と場所を選ぶことができます。 アプリケーションを開始するときに、以下のフォーマットで **java** コマンドを使用してください。

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

コマンド中、*config_file_url* は IBM MQ classes for Java 構成ファイルの名前と場所を指定する Uniform Resource Locator (URL) です。 以下のタイプの URL がサポートされます。`http`、`file`、`ftp`、および `jar`。

以下の例は、**java** コマンドを示しています。

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

このコマンドは、IBM MQ classes for Java 構成ファイルを、ローカル Windows システム上のファイル `D:\mydir\mqjava.config` として識別します。

アプリケーションが開始すると、IBM MQ classes for Java は構成ファイルの内容を読み取り、指定されたプロパティーを内部のプロパティー・ストアに保管します。 **java** コマンドが構成ファイルを識別しない場合、または構成ファイルが見つからない場合、IBM MQ classes for Java はすべてのプロパティーについてデフォルト値を使用します。 **java** コマンドでシステム・プロパティーとして指定することにより、必要に応じて構成ファイル中のプロパティーを指定変更できます。

IBM MQ classes for Java 構成ファイルは、アプリケーションとキュー・マネージャーまたはブローカーとの間のサポートされているいずれのトランスポートとも、共に使用することができます。

IBM MQ MQI client 構成ファイルで指定されたプロパティーの指定変更

IBM MQ MQI client 構成ファイルは、IBM MQ classes for Java を構成するために使用されるプロパティーを指定することもできます。 ただし、IBM MQ MQI client 構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

必要であれば、IBM MQ classes for Java 構成ファイルのプロパティとして指定することで、IBM MQ MQI client 構成ファイル内のあらゆる属性をオーバーライドすることができます。 IBM MQ MQI client 構成ファイル中の属性を指定変更するには、IBM MQ classes for Java 構成ファイルで次のフォーマットのエントリーを使用します。

```
com.ibm.mq.cfg.stanza.propName=propValue
```

エントリー中の変数には、以下の意味があります。

stanza
属性を含んでいる IBM MQ MQI client 構成ファイル中のスタンザの名前。

propName
IBM MQ MQI client 構成ファイルに指定されている属性の名前。

propValue
IBM MQ MQI client 構成ファイルに指定されている属性の値をオーバーライドするプロパティーの値。

あるいは、 **java** コマンドでシステム・プロパティーとしてプロパティーを指定することにより、IBM MQ MQI client 構成ファイル内の属性をオーバーライドすることができます。 プロパティーをシステム・プロパティーとして指定するには、前述のフォーマットを使用してください。

IBM MQ MQI client 構成ファイル中の属性のうち、IBM MQ classes for Java に関係するのは以下のものだけです。 他の属性を指定または指定変更しても、効果はありません。 特に、クライアント構成ファイルの CHANNELS スタンザ の `ChannelDefinitionFile` および `ChannelDefinitionDirectory` は使用されないことに注意してください。 IBM MQ classes for Java で CCDT を使用する方法について詳しくは、377 ページの『IBM MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』 を参照してください。

表 *54.* クライアント構成ファイルのどのスタンザにどの属性が含まれているか

スタンザ	**属性**
クライアント構成ファイルの CHANNELS スタンザ	Put1DefaultAlwaysSync
クライアント構成ファイルの CHANNELS スタンザ	PasswordProtection
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath

表 54. クライアント構成ファイルのどのスタンザにどの属性が含まれているか (続き)	
スタンザ	**属性**
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの JMQI スタンザ	useMQCSPauthentication
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBuffSize
クライアント構成ファイルの TCP スタンザ	ClntSndBuffSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

IBM MQ MQI client 構成について詳しくは、 IBM MQ MQI client 構成ファイル (`mqclient.ini`) を参照してください。

関連タスク

IBM MQ classes for Java アプリケーションのトレース

Java 標準環境トレースを使用した *Java* トレースの構成

Java 標準環境トレース設定スタンザを使用して、 IBM MQ classes for Java トレース機能を構成します。

`com.ibm.msg.client.commonservices.trace.outputName` = *`traceOutputName`*

`traceOutputName` はトレース出力の送信先のディレクトリーおよびファイル名です。

デフォルトでは、トレース情報はアプリケーションの現行作業ディレクトリー内のトレース・ファイルに書き込まれます。 トレース・ファイルの名前は、アプリケーションが実行されている環境によって異なります。

- IBM MQ 9.1.5 および IBM MQ 9.1.0 Fix Pack 5 から:
  - アプリケーションが IBM MQ classes for Java を再配置可能 JAR ファイル `com.ibm.mq.allclient.jar` からロードした場合、トレースは `mqjavaclient_%PID%.cl%u.trc` というファイルに書き込まれます。
  - アプリケーションが IBM MQ classes for Java を JAR ファイル `com.ibm.mq.jar` からロードした場合、トレースは `mqjava_%PID%.cl%u.trc` というファイルに書き込まれます。
- IBM MQ 9.0.0 Fix Pack 2 以降:
  - アプリケーションが IBM MQ classes for Java を再配置可能 JAR ファイル `com.ibm.mq.allclient.jar` からロードした場合、トレースは `mqjavaclient_%PID%.trc` というファイルに書き込まれます。
  - アプリケーションが IBM MQ classes for Java を JAR ファイル `com.ibm.mq.jar` からロードした場合、トレースは `mqjava_%PID%.trc` というファイルに書き込まれます。
- IBM MQ classes for Java for IBM MQ 9.0.0 Fix Pack 1 以前では、トレースは `mqjms_%PID%.trc` という名前のファイルに書き込まれます。

ここで、*%PID%* はトレースされるアプリケーションのプロセス ID です。*%u* は、異なる Java クラス・ローダーの下でトレースを実行するスレッドの間でファイルを区別するための固有の番号です。

プロセス ID が使用できない場合は、乱数が生成され、接頭部に `f` の文字が付けられます。 指定するファイル名にプロセス ID を組み込むには、ストリング `%PID%` を使用します。

代替ディレクトリーを指定する場合、そのディレクトリーが存在していなければならず、また、そのディレクトリーへの書き込み権限が必要です。 書き込み権限がないと、トレース出力は `System.err` に書き込まれます。

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList は、トレースされるパッケージおよびクラスのリスト、または特殊値 `ALL` または `NONE` です。

パッケージまたはクラス名はセミコロン `;` で区切ります。 *includeList* はデフォルトで `ALL` に設定され、IBM MQ classes for Java のすべてのパッケージとクラスをトレースします。

注: パッケージを含めた後、そのパッケージのサブパッケージを除外することができます。 例えば、パッケージ `a.b` は含め、パッケージ `a.b.x` は除外する場合、`a.b.y` および `a.b.z` 内のものはすべてトレースに含まれますが、`a.b.x` 内のものと `a.b.x.1` 内のものは、どちらも除外されます。

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList は、トレースされないパッケージおよびクラスのリスト、または特殊値 `ALL` または `NONE` です。

パッケージまたはクラス名はセミコロン `;` で区切ります。 *excludeList* はデフォルトで `NONE` に設定されるため、IBM MQ classes for JMS のパッケージおよびクラスはいずれもトレースから除外されません。

注: パッケージを除外した後、そのパッケージのサブパッケージを含めることができます。 例えば、パッケージ `a.b` は除外し、パッケージ `a.b.x` は含める場合、`a.b.x` および `a.b.x.1` 内のものはすべてトレースに含まれますが、`a.b.y` 内のものと `a.b.z` 内のものは、どちらも除外されます。

両方 (包含と除外) を指定した場合、同じレベルのパッケージまたはクラスはすべて含められます。

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBytes*
maxArrayBytes は、任意のバイト配列からトレースされる最大バイト数です。

maxArrayBytes が正整数に設定されると、トレース・ファイルに書き出されるバイト配列内のバイト数が制限されます。 *maxArrayBytes* の書き出し後に、バイト配列が切り捨てられます。 *maxArrayBytes* を設定すると、結果として生成されるトレース・ファイルのサイズが削減され、トレースがアプリケーションのパフォーマンスに与える影響が低減されます。

このプロパティーの値 `0` は、どのバイト配列の内容もトレース・ファイルに送信されないことを意味します。

デフォルト値は `-1` です。これは、トレース・ファイルに送信されるバイト配列内のバイト数の制限を除去します。

com.ibm.msg.client.commonservices.trace.limit = *maxTraceBytes*
maxTraceBytes は、トレース出力ファイルに書き込まれる最大バイト数です。

maxTraceBytes は *traceCycles* と連携して機能します。 書き込まれたトレースのバイト数が制限に近い場合、ファイルが閉じられ、新しいトレース出力ファイルが開始されます。

値 `0` は、トレース出力ファイルの長さがゼロであることを意味します。 デフォルト値は `-1` です。これは、トレース出力ファイルに書き込まれるデータの量が無制限であることを意味します。

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles は、循環するトレース出力ファイルの数です。

現行のトレース出力ファイルが *maxTraceBytes* で指定された制限に達すると、ファイルが閉じます。以降のトレース出力は、順序で次にあるトレース出力ファイルに書き込まれます。 各トレース出力ファイルは、ファイル名に付加される数値の接尾部によって区別されます。 現行つまり最新のトレース

出力ファイルは mqjms.trc.0 で、次に新しいトレース出力ファイルは mqjms.trc.1 です。 それより古いトレース・ファイルは、制限に達するまで、同じ番号付けパターンに従います。

traceCycles のデフォルト値は 1 です。 *traceCycles* が 1 であれば、現在のトレース出力ファイルが最大サイズに達すると、そのファイルは閉じられて、削除されます。 同じ名前の新しいトレース出力ファイルが開始されます。 したがって、トレース出力ファイルは一度に 1 つだけ存在することになります。

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters は、メソッド・パラメーターおよび戻り値をトレースに含めるかどうかを制御します。

traceParameters はデフォルトで TRUE に設定されます。 *traceParameters* が FALSE に設定されると、メソッド・シグニチャーのみがトレースされます。

com.ibm.msg.client.commonservices.trace.startup = *startup*
IBM MQ classes for Java には、リソースが割り振られるフェーズである、初期化フェーズがあります。主なトレース機能の初期化が、このリソース割り振りフェーズで行われます。

startup が TRUE に設定されると、開始トレースが使用されます。 トレース情報が即時に生成され、すべてのコンポーネント (トレース機能自体を含む) の設定が含まれます。 開始トレース情報は、構成の問題を診断するために使用できます。 開始トレース情報は常に System.err に書き込まれます。

startup はデフォルトで FALSE に設定されます。

startup は、初期化が完了する前にチェックされます。 このため、プロパティーを指定する際には必ず、コマンド・ラインで Java システム・プロパティーとしてのみ指定してください。 IBM MQ classes for Java 構成ファイルでは指定しないでください。

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
compressedTrace を TRUE に設定して、トレース出力を圧縮します。

compressedTrace のデフォルト値は FALSE です。

compressedTrace が TRUE に設定されると、トレース出力が圧縮されます。 デフォルトのトレース出力ファイル名には、 .trz という拡張子が付いています。 圧縮をデフォルト値の FALSE に設定すると、ファイルの拡張子は .trc となります。これは、非圧縮であることを示します。 ただし、トレース出力のファイル名が *traceOutputName* で指定された場合には、その名前が代わりに使用されます。ファイルに接尾部は適用されません。

圧縮されたトレース出力は、圧縮されていないものよりサイズが小さくなります。 入出力が少なくなるため、圧縮されていないトレースよりも、圧縮されたトレースのほうが書き込み速度が速くなります。 圧縮されたトレースでは、圧縮されていないトレースよりも、IBM MQ classes for Java のパフォーマンスに与える影響が少なくなります。

maxTraceBytes および *traceCycles* が設定されると、複数のフラット・ファイルの代わりに、複数の圧縮されたトレース・ファイルが作成されます。

制御されない形で IBM MQ classes for Java が終了した場合、圧縮されたトレース・ファイルが無効である可能性があります。 このため、トレースの圧縮は、必ず IBM MQ classes for Java が制御された形で終了する場合にのみ使用してください。 調査中の問題が原因で JVM 自体が予期せず停止することがない場合にのみ、トレース圧縮を使用してください。 System.Halt() シャットダウンまたは異常終了、無制御の JVM 終了につながる可能性のある問題を診断しているときは、トレースの圧縮を使用しないでください。

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel は、トレースのフィルター・レベルを指定します。 以下は、定義済みのトレース・レベルです。

- TRACE_NONE: 0
- TRACE_EXCEPTION: 1
- TRACE_WARNING: 3

- TRACE_INFO: 6
- TRACE_ENTRYEXIT: 8
- TRACE_DATA: 9
- TRACE_ALL: Integer.MAX_VALUE

各トレース・レベルはすべての下位レベルを含みます。 例えば、トレース・レベルを TRACE_INFO に設定した場合、定義済みレベルの TRACE_EXCEPTION、TRACE_WARNING、または TRACE_INFO を持つトレース・ポイントがすべてトレースに書き込まれます。 それ以外のトレース・ポイントはすべて除外されます。

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace は、 IBM MQ classes for Java クライアント・トレース・サービスを WebSphere Application Server 環境で使用するかどうかを制御します。

standaloneTrace を TRUE に設定した場合、IBM MQ classes for Java クライアント・トレース・プロパティーを使用してトレース構成が決定されます。

standaloneTrace を FALSE に設定して、かつ IBM MQ classes for Java クライアントが WebSphere Application Server コンテナー内で実行されている場合には、WebSphere Application Server のトレース・サービスが使用されます。 生成されるトレース情報は、アプリケーション・サーバーのトレース設定によって変わります。

standaloneTrace のデフォルト値は FALSE です。

IBM MQ classes for Java とソフトウェア管理ツール
Apache Maven などのソフトウェア管理ツールを IBM MQ classes for Java で使用できます。

多くの大規模開発会社がこれらのツールを使用して、サード・パーティー・ライブラリーのリポジトリーを集中管理しています。

IBM MQ classes for Java は、いくつかの JAR ファイルで構成されています。 この API を使用して Java 言語アプリケーションを開発する場合は、IBM MQ サーバー、IBM MQ クライアント、または IBM MQ クライアント SupportPac のいずれかを、アプリケーションを開発するマシンにインストールする必要があります。

ソフトウェア管理ツールを使用し、IBM MQ classes for Java を構成する JAR ファイルを集中管理リポジトリーに追加する場合は、以下の点を守る必要があります。

- リポジトリーまたはコンテナーは、社内の開発者だけが使用できるようにしなければなりません。 社外に分散させることは許可されません。
- リポジトリーには、単一の IBM MQ リリースまたはフィックスパックからの整合した完全な JAR ファイル・セットを入れる必要があります。
- IBM サポートが提供するメンテナンスでリポジトリーを更新する必要があります。

IBM MQ 8.0 以降、com.ibm.mq.allclient.jar JAR ファイルをリポジトリーにインストールする必要があります。

IBM MQ 9.0 以降、Bouncy Castle セキュリティー・プロバイダーおよび CMS サポート JAR ファイルは必須です。 詳しくは、353 ページの『IBM MQ classes for Java 再配置可能 JAR ファイル』および IBM 以外の JRE のサポートを参照してください。

IBM MQ classes for Java アプリケーションのインストール後のセットアップ

IBM MQ classes for Java をインストールした後に、独自のアプリケーションを実行するためにインストール環境を構成できます。

IBM MQ 製品 README ファイルで、最新情報やご使用の環境に関するより具体的な情報を必ず確認するようにしてください。 最新バージョンの製品 README ファイルは、IBM MQ、 WebSphere MQ、および MQSeries® 製品の README Web ページにあります。

IBM MQ classes for Java アプリケーションをバインディング・モードで実行する前に、「構成」の説明に従って IBM MQ を構成したことを確認してください。

IBM MQ classes for Java からのクライアント接続を受け入れるためのキュー・マネージャーの構成
クライアントからの着信接続要求を受け入れるようにキュー・マネージャーを構成するには、サーバー接続チャネルの使用を定義して許可し、リスナー・プログラムを開始します。

詳細は 1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』 を参照してください。

Java security manager での *IBM MQ classes for Java* アプリケーションの実行
IBM MQ classes for Java は、 Java security manager を有効にして実行できます。 Java security manager が有効になっているアプリケーションを正常に実行するには、適切なポリシー定義ファイルを使用して Java Virtual Machine (JVM) を構成する必要があります。

適切なポリシー定義ファイルを作成する最も簡単な方法は、Java runtime environment (JRE) に付属するポリシー・ファイルを変更する方法です。 ほとんどのシステムでは、このファイルは、JRE ディレクトリーに対して相対的な `path lib/security/java.policy` に保管されます。 ポリシー・ファイルは、好みのエディターを使用して編集することも、JRE に付属する policytool プログラムを使用して編集することもできます。

`com.ibm.mq.jmqi.jar` ファイルに権限を付与して、以下のことを実行できるようにする必要があります。

- (クライアント・モードで) ソケットを作成する。
- (バインディング・モードで) 固有のライブラリーをロードする。
- 環境から種々のプロパティーを読み取る。

Java security manager の下で実行する場合は、システム・プロパティー **os.name** が IBM MQ classes for Java で使用可能でなければなりません。

Java アプリケーションが Java security manager を使用する場合、アプリケーションが使用する `java.security.policy` ファイルに以下の許可を追加する必要があります。そうでない場合は、例外がアプリケーションにスローされます。

```
permission java.lang.RuntimePermission "modifyThread";
```

この RuntimePermission は、キュー・マネージャーに対する TCP/IP 接続を介した多重会話の割り当てとクローズの管理の一環として、クライアントで必要になります。

ポリシー・ファイル・エントリーの例

以下は、デフォルトのセキュリティー・マネージャーの下で IBM MQ classes for Java を正常に実行できるようにするポリシー・ファイル・エントリーの一例です。 この例に含まれるストリング *MQ_INSTALLATION_PATH* は、ご使用のシステムで IBM MQ classes for Java がインストールされている場所に置き換えてください。

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
```

```
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";

// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

このポリシー・ファイルの例により、IBM MQ classes for Java はセキュリティー・マネージャーの下で正常に作動するようになりますが、特定のアプリケーションを作動させるには、独自のコードを正常に実行できるよう、事前の作業がさらに必要な場合もあります。

IBM MQ classes for Java に付属のサンプル・コードは特にセキュリティー・マネージャーと併用可能になっているわけではありませんが、IVT テストは上記のポリシー・ファイルとデフォルトのセキュリティー・マネージャーを適切に使用して実行されます。

重要:

IBM MQ classes for Java トレース機能は、他にもシステム・プロパティーの照会やファイル・システム操作を行うので、さらに追加の権限が必要になります。

トレースを有効にしたセキュリティー・マネージャーの下で実行するための適切なテンプレート・セキュリティー・ポリシー・ファイルは、IBM MQ インストールの `samples/wmqjava` ディレクトリーで `example.security.policy` として提供されます。

デフォルト・インストールの場合、`example.security.policy` ファイルは以下の場所に配置されます。

Windows
C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy 内

Linux
/opt/mqm/samp/wmqjava/samples/example.security.policy 内

Solaris
/opt/mqm/samp/wmqjava/samples/example.security.policy 内

AIX
/usr/mqm/samp/wmqjava/samples/example.security.policy 内

CICS Transaction Server での *IBM MQ classes for Java* アプリケーションの実行
IBM MQ classes for Java アプリケーションを CICS Transaction Server の下でトランザクションとして実行することができます。

IBM MQ classes for Java アプリケーションを CICS Transaction Server for z/OS の下でトランザクションとして実行するには、以下の手順を実行します。

1. 提供されている CEDA トランザクションを使用して、アプリケーションとトランザクションを CICS に定義する。
2. IBM MQ CICS アダプターが CICS システムにインストールされていることを確認します。
   z/OS (詳しくは、 CICS での IBM MQ の使用 を参照してください。)
3. CICS で指定した JVM 環境に、適切な CLASSPATH および LIBPATH エントリーが組み込まれていることを確認する。
4. 通常のプロセスを使用してトランザクションを開始する。

CICS Java トランザクションの実行に関する詳細は、お手持ちの CICS システムの資料を参照してください。

IBM MQ classes for Java インストール済み環境の検証
インストール検査プログラム MQIVP が IBM MQ classes for Java に付属しています。 このプログラムを使用して、IBM MQ classes for Java のすべての接続モードをテストできます。

このプログラムでは、いくつかの選択項目とその他のデータについてプロンプトが表示され、検査対象の接続モードを判別します。 インストールを検査するには、以下の手順に従ってください。

1. クライアント・モードでプログラムを実行する場合は、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』の説明に従ってキュー・マネージャーを構成します。 使用するキューは SYSTEM.DEFAULT.LOCAL.QUEUE
2. プログラムをクライアント・モードで実行する場合は、347 ページの『IBM MQ classes for Java の使用』も参照してください。

   プログラムを実行するシステムで、この手順の残りのステップを実行します。
3. 356 ページの『IBM MQ classes for Java に関連する環境変数』の指示に従って CLASSPATH 環境変数を更新したことを確認します。
4. ディレクトリーを *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples に変更します。ここで *MQ_INSTALLATION_PATH* は IBM MQ インストール済み環境へのパスです。 その後、コマンド・プロンプトに次を入力します。

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   ここで、*library_path* は IBM MQ classes for Java ライブラリーへのパスです (358 ページの『IBM MQ classes for Java ライブラリー』を参照)。

   (1) のマークが付いたプロンプトでは、次のようにします。

   - TCP/IP 接続を使用する場合は、IBM MQ サーバーのホスト名を入力します。
   - ネイティブ接続 (バインディング・モード) を使用する場合は、このフィールドをブランクのままにします (名前は入力しないでください)。

このプログラムは、以下の処理を実行します。

- 1. キュー・マネージャーに接続します。
- 2. キュー SYSTEM.DEFAULT.LOCAL.QUEUE を開き、キューにメッセージを書き込み、キューからメッセージを取得し、キューを閉じます。
- 3. キュー・マネージャーへの接続を切断します。
- 4. 操作が正常に実行された場合にメッセージを返します。

以下は、表示されるプロンプトと応答の例です。 実際のプロンプトと応答は、使用している IBM MQ ネットワークによって異なります。

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

注 :

1. z/OS z/OS の場合、(1) のマークが付いたプロンプトのフィールドは空にしておきます。
2. サーバー接続を選択した場合は、(2) のマークが付いたプロンプトは表示されません。
3. IBM i IBM i の場合、QShell から `java MQIVP` コマンドのみを発行できます。 別の方法として、CL コマンド `RUNJVA CLASS(MQIVP)` を使用してアプリケーションを実行することもできます。

IBM MQ classes for Java サンプル・アプリケーションの使用

IBM MQ classes for Java サンプル・アプリケーションは、IBM MQ classes for Java API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

このタスクについて

独自のアプリケーションを作成するために手助けが必要な場合は、サンプル・アプリケーションを開始点として使用できます。 各アプリケーションのソース・バージョンとコンパイル・バージョンの両方が提供されています。 サンプルのソース・コードを検討し、アプリケーションに必要な各オブジェクト (MQQueueManager、MQConstants、MQMessage、MQPutMessageOptions および MQDestination) を作成するための主な手順や、アプリケーションの動作を指定するときに必要な特定のプロパティーを設定するための主な手順を見極めます。 詳細については、373 ページの『IBM MQ classes for Java アプリケーションの作成』を参照してください。 これらのサンプルは、IBM MQ Java の将来のリリースで変更される可能性があります。

369 ページの表 55 に、各プラットフォームで IBM MQ classes for Java サンプル・アプリケーションがどこにインストールされるかを示します。

表 *55. IBM MQ classes for Java* サンプル・アプリケーションのインストール・ディレクトリー

プラットフォーム	ディレクトリー
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`

表 55. *IBM MQ classes for Java* サンプル・アプリケーションのインストール・ディレクトリー (続き)	
プラットフォーム	**ディレクトリー**
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\samples
IBM i	/qibm/proddata/mqm/java/samples/wmqjava/samples
z/OS	*MQ_INSTALLATION_PATH*/java/samples/wmqjava

370 ページの表 56 は、 IBM MQ classes for Java で提供されるサンプル・アプリケーションのセットを示しています。

表 56. *IBM MQ classes for Java* サンプル・アプリケーション	
サンプル名	**説明**
IMSBridgeSample.java	IBM MQ classes for Java で IMS ブリッジを使用する方法を簡単に説明するプログラム。
MQIVP.java	IBM MQ Java のインストール検査プログラム。
MQMessagePropertiesSample.java	メッセージ・プロパティー API の使用例を示します。
MQPubSubApiSample.java	パブリッシュ/サブスクライブ API の使用例を示します。
MQSample.java	キューへのメッセージの書き込みと取得を行う実例を示す単純なプログラム。
MQSampleMessageManager.java	IBM MQ の基本 Java サンプルでのメッセージ処理用のユーティリティー・クラス。
mqjcivp.properties	このリソース・バンドルには、IBM MQ classes for Java インストール検査プログラム (MQIVP.java) によって使用されるメッセージが含まれます。

IBM MQ classes for Java は、サンプル・アプリケーションの実行に使用できる runjms と呼ばれるスクリプトを提供します。 このスクリプトは、IBM MQ 環境をセットアップして、IBM MQ classes for Java サンプル・アプリケーションを実行できるようにします。

370 ページの表 57 は、プラットフォームごとのスクリプトの場所を示しています。

表 57. *runjms* スクリプトのロケーション	
プラットフォーム	**ディレクトリー**
AIX Linux	*MQ_INSTALLATION_PATH*/java/bin/runjms
Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat
IBM i	/qibm/proddata/mqm/java/bin/runjms または /qibm/proddata/mqm/java/bin/runjms64
z/OS	*MQ_INSTALLATION_PATH*java/bin/runjms

runjms スクリプトを使用してサンプル・アプリケーションを起動するには、以下のステップを実行します。

手順

1. コマンド・プロンプトを表示し、実行するサンプル・アプリケーションが格納されているディレクトリーにナビゲートします。
2. 以下のコマンドを入力します。

```
Path to the runjms script/runjms sample_application_name
```

   サンプル・アプリケーションは、必要とするパラメーターのリストを表示します。
3. 次のコマンドを入力して、これらのパラメーターを指定してサンプルを実行します。

```
Path to the runjms script/runjms sample_application_name parameters
```

例

Linux 例えば、Linux で MQIVP サンプルを 実行するには、次のコマンドを入力します。

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

関連概念

90 ページの『IBM MQ classes for JMS でインストールされる内容』
IBM MQ classes for JMS のインストール時にはいくつかのファイルとディレクトリーが作成されます。 Windows では、インストールの際に、自動的に環境変数を設定することによっていくつかの構成が実行されます。 他のプラットフォームやある特定の Windows 環境では、IBM MQ classes for JMS アプリケーションを実行するために、まず環境変数を設定する必要があります。

IBM MQ classes for Java の問題の解決

最初に、インストール検査プログラムを実行します。 トレース機能も使用する必要がある場合があります。

アプリケーションが正常に完了しない場合は、インストール検査プログラムを実行し、診断メッセージに示されるアドバイスに従ってください。 インストール検査プログラムについては、368 ページの『IBM MQ classes for Java インストール済み環境の検証』で説明されています。

問題が解決せず、IBM サービスに連絡する必要があるときは、トレース機能をオンにするようお願いする場合があります。 以下の例のようにこれを行ってください。

MQIVP プログラムをトレースするには、以下のようにします。

- com.ibm.mq.commonservices プロパティー・ファイルを作成します (「com.ibm.mq.commonservices の使用」を参照)。
- 以下のコマンドを入力します。

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

ここで、

- *commonservices_properties_file* は、com.ibm.mq.commonservices プロパティー・ファイルへのパス (ファイル名を含む) です。
- *library_path* は IBM MQ classes for Java ライブラリーへのパスです (358 ページの『IBM MQ classes for Java ライブラリー』を参照)。

トレースの使用法について詳しくは、IBM MQ classes for Java アプリケーションのトレースを参照してください。

z/OS MQ Adv. VUE z/OS で実行されているバッチ・アプリケーションへの Java クライアント接続

特定の条件下では、 z/OS 上の IBM MQ classes for Java アプリケーションは、クライアント接続を使用して z/OS 上のキュー・マネージャーに接続できます。 クライアント接続を使用すると、 IBM MQ トポロジーを単純化できます。

クライアント接続を使用して、 IBM MQ classes for Java アプリケーションがバッチ環境で実行されていて、以下のいずれかの条件が該当する場合、アプリケーションはリモート z/OS キュー・マネージャーに接続できます。

- V 9.3.4 LTS IBM MQ classes for Java コードは、 IBM MQ 9.3.4 以降、または APAR PH56722 が適用された Long Term Support にあります。 キュー・マネージャーは、サポートされている任意のバージョンにすることができます。
- 接続先のキュー・マネージャーは IBM MQ Advanced for z/OS Value Unit Edition ライセンスを使用して実行されているため、 **ADVCAP** パラメーターは ENABLED に設定されています。 キュー・マネージャーは、サポートされている任意のバージョンにすることができます。

IBM MQ Advanced for z/OS Value Unit Edition について詳しくは、 IBM MQ 製品 ID およびエクスポート情報を参照してください。

ADVCAP について詳しくは、DISPLAY QMGR を参照してください。 **QMGRPROD** について詳しくは、START QMGR を参照してください。

z/OS 上の IBM MQ classes for Java アプリケーションは、クライアント・モード接続を使用して、 z/OS

z/OS 上の IBM MQ classes for Java アプリケーションがクライアント・モードを使用して接続しようとしたが、それが許可されていない場合、 MQRC_ENVIRONMENT_ERROR が戻されます。

Advanced Message Security (AMS) サポート

IBM MQ classes for Java クライアント・アプリケーションは、このトピックで前述した条件に従って、リモート z/OS キュー・マネージャーに接続するときに AMS を使用できます。

この方法で AMS を使用するには、クライアント・アプリケーションは `keystore.conf` で鍵ストア・タイプ `jceracfks` を使用する必要があります。ここで、

- プロパティー名の接頭部は `jceracfks` で、この名前接頭部では大/小文字は区別されません。
- 鍵ストアは RACF 鍵リングです。
- パスワードは必須ではなく、無視されます。 RACF 鍵リングではパスワードが使用されないためです。
- プロバイダーを指定する場合、そのプロバイダーは `IBMJCE` でなければなりません。

AMS で `jceracfks` を使用する場合、鍵ストアは `safkeyring://`*`user`*`/`*`keyring`* の形式でなければなりません。ここで、

- `safkeyring` はリテラルで、この名前では大/小文字は区別されません。
- *`user`* は、鍵リングを所有する RACF ユーザー ID です。
- *`keyring`* は RACF 鍵リングの名前で、鍵リングの名前には大/小文字の区別があります。

以下の例では、ユーザー JOHNDOE の標準 AMS 鍵リングを使用しています。

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

関連概念

127 ページの『z/OS 上で実行されているバッチ・アプリケーションへの JMS/Jakarta Messaging クライアント接続』
特定の条件下では、 z/OS 上の IBM MQ classes for JMS/Jakarta Messaging アプリケーションは、クライアント接続を使用して z/OS 上のキュー・マネージャーに接続できます。 クライアント接続を使用すると、 IBM MQ トポロジーを単純化できます。

IBM MQ classes for Java アプリケーションの作成

この一連のトピックでは、IBM MQ システムと対話する Java アプリケーションの作成に役立つ情報を提供します。

IBM MQ classes for Java を使用して IBM MQ キューにアクセスするには、 IBM MQ キューにメッセージを書き込んだり、キューからメッセージを取得したりする呼び出しを含む Java アプリケーションを作成します。 個々のクラスの詳細については、IBM MQ classes for Java を参照してください。

注 : IBM MQ classes for Java は自動クライアント再接続をサポートしていません。

IBM MQ classes for Java インターフェース

プロシージャー型 IBM MQ アプリケーション・プログラミング・インターフェースは、オブジェクトに対してアクションを実行する動詞を使用します。 Java プログラミング・インターフェースでは、メソッドを呼び出すことによって操作するオブジェクトを使用します。

プロシージャー型 IBM MQ アプリケーション・プログラミング・インターフェースは、以下のような動詞によって構築されています。

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

これらの動詞はすべて、操作対象の IBM MQ オブジェクトのハンドルをパラメーターとして取ります。 ユーザーのプログラムは、一連の IBM MQ オブジェクトで構成されています。これらのオブジェクトは、メソッドを呼び出すことによって操作します。

手続き型のインターフェースを使用する場合は、MQDISC (Hconn, CompCode, Reason) の呼び出しを用いてキュー・マネージャーから切断します。*Hconn* はキュー・マネージャーに対するハンドルの 1 つです。

Java インターフェースでは、キュー・マネージャーはクラス MQQueueManager のオブジェクトで表されます。 キュー・マネージャーからの切断は、そのクラスで disconnect() メソッドを呼び出すことによって行われます。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

IBM MQ classes for Java の接続モード

IBM MQ classes for Java のプログラミング方法は、使用する接続モードによって多少異なります。

クライアント接続を使用する場合は、IBM MQ MQI client との相違がいくつかありますが、概念的には似ています。 バインディング・モードを使用する場合は、ファスト・パス・バインディングの使用、および MQBEGIN コマンドの発行が可能です。 MQEnvironment クラスで変数を設定することで、使用するモードを指定します。

IBM MQ classes for Java のクライアント接続

IBM MQ classes for Java がクライアントとして使用される場合は、IBM MQ MQI client と類似していますが、いくつかの相違点があります。

IBM MQ classes for Java をクライアントとして使用するためにプログラミングする場合は、次の違いに注意してください。

- TCP/IP のみをサポートします。
- 始動時にどの IBM MQ 環境変数も読み取りません。
- チャネル定義および環境変数に格納される情報は、Environment と呼ばれるクラスに格納できます。 または、この情報を、接続の際にパラメーターとして渡すことができます。

- エラー条件と例外条件を MQException クラスに指定されたログに書き込みます。 デフォルトのエラー宛先は Java コンソールです。
- IBM MQ クライアント構成ファイル中の属性のうち、IBM MQ classes for Java に関係するのは以下のものだけです。 他の属性を指定しても、効果はありません。

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBuffSize
クライアント構成ファイルの TCP スタンザ	ClntSndBuffSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

- 文字データの変換が必要なキュー・マネージャーに接続したとき、キュー・マネージャーで変換を行えない場合には、V7 Java クライアントでその変換を実行できます。 クライアント JVM は、クライアントの CCSID とキュー・マネージャーの CCSID の間の変換をサポートしている必要があります。
- IBM MQ classes for Java は自動クライアント再接続をサポートしていません。

クライアント・モードで使用する場合、*IBM MQ classes for Java* は MQBEGIN の呼び出しをサポートしません。

IBM MQ classes for Java のバインディング・モード

IBM MQ classes for Java のバインディング・モードは、主に次の 3 つの点でクライアント・モードとは異なります。

バインディング・モードで使用すると、IBM MQ classes for Java は、ネットワーク経由で通信するのではなく、Java Native Interface (JNI) を使用して、既存のキュー・マネージャー API を直接呼び出します。

デフォルトでは、IBM MQ classes for Java をバインディング・モードで使用するアプリケーションは、*ConnectOption* の MQCNO_STANDARD_BINDINGS を使用してキュー・マネージャーに接続します。

IBM MQ classes for Java は、以下の *ConnectOptions* をサポートしています。

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

ConnectOptions の詳細については、738 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』 を参照してください。

バインディング・モードで、キュー・マネージャーによって調整されるグローバル作業単位を開始する MQBEGIN 呼び出しがサポートされるのは、IBM MQ for IBM i と IBM MQ for z/OS を除く、すべてのプラットフォームです。

MQEnvironment クラスによって提供されるパラメーターの大部分はバインディング・モードとは無関係で、無視されます。

使用する *IBM MQ classes for Java* 接続の定義

使用する接続のタイプは、MQEnvironment クラスでの変数の設定によって決定されます。

以下の 2 つの変数が使用されます。

MQEnvironment.properties
接続タイプは、キー名 CMQC.TRANSPORT_PROPERTY に関連付けられた値によって決定されます。 次の値を指定できます。

CMQC.TRANSPORT_MQSERIES_BINDINGS
バインディング・モードで接続します

CMQC.TRANSPORT_MQSERIES_CLIENT
クライアント・モードで接続します

CMQC.TRANSPORT_MQSERIES
接続モードは *hostname* プロパティーの値によって決定されます

MQEnvironment.hostname
この変数の値を次のように設定します。

- クライアント接続の場合、接続先の IBM MQ サーバーのホスト名にこの変数の値を設定します。
- バインディング・モードの場合、この変数を設定しないでおくか、または NULL に設定します。

キュー・マネージャーに対する操作

以下の一連のトピックでは、IBM MQ classes for Java を使用してキュー・マネージャーに接続する方法と、そのキュー・マネージャーから接続を切断する方法について説明します。

IBM MQ classes for Java 用の *IBM MQ* 環境のセットアップ

アプリケーションがクライアント・モードでキュー・マネージャーに接続するには、チャネル名、ホスト名、およびポート番号を指定する必要があります。

注 : このトピックの情報は、アプリケーションがクライアント・モードでキュー・マネージャーに接続する場合にのみ該当します。 バインディング・モードで接続する場合は、該当しません。 110 ページの『IBM MQ classes for JMS の接続モード』 を参照して

チャネル名、ホスト名、およびポート番号を指定するには、2 つの方法 (MQEnvironment クラスのフィールドと MQQueueManager オブジェクトのプロパティー) のいずれかを使用することができます。

MQEnvironment クラスのフィールドを設定する場合、それらのフィールドはアプリケーション全体に適用されます (プロパティーのハッシュ・テーブルによりオーバーライドされる場所を除く)。 MQEnvironment でチャネル名およびホスト名を指定するには、次のコードを使用します。

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

これは、**MQSERVER** 環境変数を次のように設定することと同じです。

```
"java.client.channel/TCP/host.domain.com".
```

デフォルトでは、Java クライアントはポート 1414 で IBM MQ リスナーへの接続を試行します。 別のポートを指定するには、次のコードを使用します。

```
MQEnvironment.port = nnnn;
```

nnnn は、必要なポート番号です。

キュー・マネージャー・オブジェクトの作成時にプロパティーを渡すと、その設定はそのキュー・マネージャーだけに適用されます。 **hostname**、**channel**、およびオプションで **port** というキーと適切な値を持ったエントリーを Hashtable オブジェクト内に作成します。 デフォルトのポート 1414 を使用する場合は、**port** エントリーを省略できます。 プロパティーのハッシュ・テーブルを受け付けるコンストラクターを使用して MQQueueManager オブジェクトを作成します。

アプリケーション名の設定によるキュー・マネージャーへの接続の識別

アプリケーションに名前を設定して、アプリケーションからキュー・マネージャーへの接続を識別することができます。 このアプリケーション名は、 **DISPLAY CONN MQSC/PCF** コマンドによって表示されます (このフィールドの名前は **APPLTAG** です)。 または IBM MQ 「エクスプローラー」 「**アプリケーション接続**」 画面 (このフィールドの名前は **App name**) で表示されます。

アプリケーション名は 28 文字に制限されているため、それより長い名前は切り捨てられます。 アプリケーション名が指定されていない場合は、デフォルトが提供されます。 デフォルト名は起動 (メイン) クラスを基にしますが、この情報が利用できない場合は、テキスト `IBM MQ Client for Java` が使用されます。

起動クラスの名前が使用される場合は、必要に応じて、制限内に収まるように、それより前の位置にあるパッケージ名を削除して調整されます。 例えば、起動クラスが `com.example.MainApp` である場合は完全な名前が使用されますが、起動クラスが `com.example.dictionaryAndThesaurus.multilingual.mainApp` である場合は `multilingual.mainApp` が使用されます。これが、有効な長さに収まるクラス名と右端のパッケージ名との最長の組み合わせだからです。

クラス名自体が 28 文字より長い場合は、収まるように切り捨てられます。 例えば、`com.example.mainApplicationForSecondTestCase` は `mainApplicationForSecondTest` となります。

MQEnvironment クラスにアプリケーション名を設定するには、次のコードを使用して、**MQConstants.APPNAME_PROPERTY** キーを指定し、名前を MQEnvironment.properties ハッシュ・テーブルに追加します。

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

MQQueueManager コンストラクターに渡されるプロパティー・ハッシュ・テーブルにアプリケーション名を設定するには、**MQConstants.APPNAME_PROPERTY** キーを指定し、 プロパティー・ハッシュ・テーブルに名前を追加します。

IBM MQ クライアント構成ファイルで指定されたプロパティーの指定変更

IBM MQ クライアント構成ファイルは、IBM MQ classes for Java を構成するために使用されるプロパティーを指定することもできます。 ただし、IBM MQ MQI client 構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

必要に応じて、次のいずれかの方法で IBM MQ 構成ファイルの任意の属性を指定変更することができます。オプションを優先順位の高いものから示します。

- 構成プロパティーの Java システム・プロパティーを設定する。
- MQEnvironment.properties マップのプロパティーを設定する。
- Java5 以降のリリースで、システム環境変数を設定する。

IBM MQ クライアント構成ファイル中の属性のうち、IBM MQ classes for Java に関係するのは以下のものだけです。 他の属性を指定または指定変更しても、効果はありません。

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

IBM MQ classes for Java でのキュー・マネージャーへの接続
MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続します。 disconnect() メソッドを呼び出して、キュー・マネージャーから切断します。

ここまでで、次のように MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続できる状態になっています。

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

キュー・マネージャーから切断するには、次のようにキュー・マネージャーで disconnect() メソッドを呼び出します。

```
queueManager.disconnect();
```

disconnect メソッドを呼び出すと、そのキュー・マネージャーからアクセスしたオープン・キューとプロセスはすべてクローズされます。 しかし、使用を終えた時点でこれらのリソースを明示的にクローズするプログラミングの習慣を付けておくことをお勧めします。 これは、関連するオブジェクトに close() メソッドを使って行います。

キュー・マネージャーの commit() メソッドと backout() メソッドは、プロシージャー型インターフェースで使用される MQCMIT 呼び出しと MQBACK 呼び出しに相当するものです。

IBM MQ classes for Java を含むクライアント・チャネル定義テーブルの使用
IBM MQ classes for Java クライアント・アプリケーションでは、クライアント・チャネル定義テーブル (CCDT) に格納されたクライアント接続チャネル定義を使用できます。

MQEnvironment クラスの特定フィールドや環境プロパティーを設定したり、それらをプロパティー・ハッシュ・テーブルに入れて MQQueueManager に渡したりすることによりクライアント接続チャネル定義を作成する代わりに、IBM MQ classes for Java クライアント・アプリケーションでは、クライアント・チャネル定義テーブルに保管されたクライアント接続チャネル定義を使用できます。 このような定義の作成には、IBM MQ スクリプト・コマンド (MQSC) または IBM MQ プログラマブル・コマンド・フォーマット (PCF) のコマンド、あるいは IBM MQ Explorer を使用します。

アプリケーションが MQQueueManager オブジェクトを作成すると、IBM MQ classes for Java クライアントはクライアント・チャネル定義テーブルを検索して、適切なクライアント接続チャネル定義を探し、そ

のチャネル定義を使用して MQI チャネルを開始します。 クライアント・チャネル定義テーブルの詳細とその構成方法については、クライアント・チャネル定義テーブルを参照してください。

クライアント・チャネル定義テーブルを使用する場合、アプリケーションはまず URL オブジェクトを作成する必要があります。 URL オブジェクトは、クライアント・チャネル定義テーブルを格納するファイルの名前と場所を識別する URL (Uniform Resource Locator) をカプセル化し、そのファイルへのアクセス方法を指定します。

例えば、ファイル `ccdt1.tab` にクライアント・チャネル定義テーブルが含まれており、アプリケーションが実行されているのと同じシステムに保管されている場合、アプリケーションは以下の方法で URL オブジェクトを作成できます。

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

もう 1 つの例として、ファイル `ccdt2.tab` にクライアント・チャネル定義テーブルが含まれており、アプリケーションが実行されているシステムとは異なるシステムに保管されているとします。 FTP プロトコルを使用してこのファイルにアクセスできる場合、アプリケーションは以下の方法で URL オブジェクトを作成できます。

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

URL オブジェクトを作成したら、アプリケーションは、URL オブジェクトをパラメーターにとるコンストラクターの 1 つを使用して `MQQueueManager` オブジェクトを作成できます。 以下が例となります。

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

このステートメントが実行されると、IBM MQ classes for Java クライアントは、URL オブジェクト chanTab2 によって識別されるクライアント・チャネル定義テーブルにアクセスし、テーブルを検索して、適切なクライアント接続チャネル定義を探します。そして、そのチャネル定義を使用して、MARS というキュー・マネージャーへの MQI チャネルを開始します。

アプリケーションがクライアント・チャネル定義テーブルを使用する場合は、以下の事項が適用されるので、注意してください。

- アプリケーションが、URL オブジェクトをパラメーターにとるコンストラクターを使用して `MQQueueManager` オブジェクトを作成する場合、`MQEnvironment` クラスにはフィールドとしても環境プロパティーとしてもチャネル名を設定しないでください。 チャネル名が設定されていると、IBM MQ classes for Java クライアントは `MQException` をスローします。 チャネル名を指定するフィールドまたは環境プロパティーは、その値がヌル、空ストリング、ブランク文字のみのストリングのいずれでもない場合に設定されていると見なされます。
- `MQQueueManager` コンストラクター上の **queueManagerName** パラメーターは、次のいずれかの値をとることができます。
  - キュー・マネージャーの名前。
  - アスタリスク (*) とそれに続くキュー・マネージャー・グループ名。
  - アスタリスク (*)。
  - ヌル、空ストリング、またはブランク文字のみを含むストリング

  これらの値は、メッセージ・キュー・インターフェース (MQI) を使用しているクライアント・アプリケーションによって発行された MQCONN 呼び出しで、**QMgrName** パラメーターに使用できる値と同じです。 これらの値の意味について詳しくは、722 ページの『Message Queue Interface の概要』を参照してください。

  アプリケーションが接続プーリングを使用する場合は、398 ページの『IBM MQ classes for Java でのデフォルト接続プールの制御』を参照してください。
- IBM MQ classes for Java クライアントは、クライアント・チャネル定義テーブル内で適切なクライアント接続チャネル定義を見つけたら、このチャネル定義から抽出された情報のみを使用して、MQI チャネ

ルを開始します。 アプリケーションが MQEnvironment クラスに設定したチャネル関連のフィールドまたは環境プロパティーは無視されます。

特に、Transport Layer Security (TLS) を使用する場合は、以下の点に注意してください。

- MQI チャネルで TLS が使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義に、IBM MQ classes for Java クライアントでサポートされる CipherSpec の名前が指定されている場合に限られます。
- クライアント・チャネル定義テーブルには、証明書取り消しリスト (CRL) を保持する LDAP (Lightweight Directory Access Protocol) サーバーの場所に関する情報も含まれます。 IBM MQ classes for Java クライアントは、この情報のみを使用して、CRL を保持する LDAP サーバーにアクセスします。
- クライアント・チャネル定義テーブルには、OCSP 応答側の場所を含めることもできます。 IBM MQ classes for Java では、クライアント・チャネル定義テーブル・ファイルの OCSP 情報を使用できません。 ただし、OCSP を構成することはできます (Online Certificate Protocol の使用セクションを参照)。

クライアント・チャネル定義テーブルでの TLS の使用の詳細については、MQI チャネルで TLS を使用するよう指定するを参照してください。

チャネル出口を使用する場合は、以下の点にも注意してください。

- MQI チャネルで使用されるのは、他のメソッドを使用して指定されたチャネル出口とデータに優先して、クライアント・チャネル定義テーブルから抽出されたチャネル定義によって指定されているチャネル出口とそれに関連するユーザー・データです。
- クライアント・チャネル定義テーブルから抽出されたチャネル定義は、Java、C、または C++ で書かれたチャネル出口を指定できます。Java でのチャネル出口の作成方法について詳しくは 392 ページの『IBM MQ classes for Java でのチャネル出口の作成』を参照してください。 チャネル出口を他の言語で作成する方法について詳しくは、395 ページの『Java で作成されていないチャネル出口を IBM MQ classes for Java で使用する』を参照してください。

IBM MQ classes for Java クライアント接続を受け入れるためのポート範囲の指定
2 つの方法のいずれかでアプリケーションがバインドできるポート、またはポートの範囲を指定することができます。

IBM MQ classes for Java アプリケーションがクライアント・モードで IBM MQ キュー・マネージャーに接続しようとした場合、ファイアウォールは指定されたポートまたはポートの範囲から発生する接続のみを許可する可能性があります。 この場合、アプリケーションがバインドできるポート、またはポートの範囲を指定することができます。 以下の方法でポートを指定できます。

- MQEnvironment クラスの localAddressSetting フィールドを設定できます。 以下が例となります。

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- 環境プロパティー CMQC.LOCAL_ADDRESS_PROPERTY を設定できます。 以下が例となります。

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- MQQueueManager オブジェクトを構成する際には、値が "192.0.2.0(2000,3000)" である LOCAL_ADDRESS_PROPERTY を含むプロパティーのハッシュ・テーブルをパスすることができます。

これらの例では、アプリケーションがキュー・マネージャーに後で接続すると、アプリケーションはローカル IP アドレスおよび 192.0.2.0(2000) から 192.0.2.0(3000) の範囲にあるポート番号にバインドされます。

複数のネットワーク・インターフェースを持つシステムでは、 localAddressSetting フィールドまたは環境プロパティー CMQC.LOCAL_ADDRESS_PROPERTY を 使用して、どのネットワーク・インターフェースを接続に使用するかを指定します。

ポートの範囲を制限すると、接続エラーが起きる可能性があります。 エラーが発生すると、IBM MQ 理由コード MQRC_Q_MGR_NOT_AVAILABLE と次のメッセージを含む MQException がスローされます。

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

指定された範囲内のすべてのポートが使用中である場合、または指定された IP アドレス、ホスト名、ポート番号が正しくない場合 (負のポート番号など) は、エラーが発生する可能性があります。

IBM MQ classes for Java でのキュー、トピック、およびプロセスへのアクセス

キュー、トピック、およびプロセスへアクセスするには、MQQueueManager クラスのメソッドを使用します。 MQOD (オブジェクト記述子構造体) は、これらのメソッドのパラメーターに縮小されます。

キュー

キューをオープンするには、MQQueueManager クラスの accessQueue メソッドを使用できます。 例えば、queueManager というキュー・マネージャーの場合、以下のコードを使用します。

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

accessQueue メソッドは、クラス MQQueue の新規オブジェクトを戻します。

キューの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
queue.close();
```

MQQueue コンストラクターを使用してキューを作成することもできます。 各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、accessQueue メソッドの場合とまったく同じものです。 以下に例を示します。

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

キューの作成時には、いくつかのオプションを指定できます。 これらの詳細については、Class.com.ibm.mq.MQQueue を参照してください。 この方法でキュー・オブジェクトを構成すると、MQQueue の独自のサブクラスを作成できます。

トピック

同じように、MQQueueManager クラスの accessTopic メソッドを使用してトピックをオープンすることができます。 例えば、queueManager というキュー・マネージャーの場合は、以下のコードを使用してサブスクライバーおよびパブリッシャーを作成します。

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

トピックの使用が完了したら、close() メソッドを使用してそのトピックをクローズしてください。

MQTopic コンストラクターを使用してトピックを作成することもできます。 キュー・マネージャー・パラメーターが追加されていることを除き、各パラメーターは accessTopic メソッドのものとまったく同じです。 以下に例を示します。

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

トピックの作成時には、いくつかのオプションを指定できます。 これらの詳細については、クラス com.ibm.mq.MQTopic を参照してください。 この方法でトピック・オブジェクトを構成すると、MQTopic のユーザー固有のサブクラスを作成することもできます。

トピックはパブリケーションかサブスクリプションのいずれかに対してオープンしなければなりません。 MQQueueManager クラスには 8 個の accessTopic メソッドがあり、 Topic クラスには 8 個のコンストラクターがあります。 それぞれ、4 個には **destination** パラメーターがあり、4 個には **subscriptionName** パラメーターがあります (内 2 個ずつには両方のパラメーターがあります)。 これらはトピックをサブスクリプションに対してオープンするためだけに使用できます。 残りの 2 個のメソッドには **openAs** パラメーターがあり、**openAs** パラメーターの値に応じてパブリケーションとサブスクリプションのいずれかに対してトピックをオープンできます。

永続サブスクライバーとしてトピックを作成するには、サブスクリプション名を受け付ける MQQueueManager クラスの accessTopic メソッドか MQTopic コンストラクターを使用し、どちらの場合も CMQC.MQSO_DURABLE オプションを設定します。

Processes

プロセスにアクセスするには、MQQueueManager の accessProcess メソッドを使用します。 例えば、queueManager というキュー・マネージャーの場合は、以下のコードを使用して MQProcess オブジェクトを作成します。

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

プロセスにアクセスするには、MQQueueManager の accessProcess メソッドを使用します。

accessProcess メソッドは、クラス MQProcess の新規オブジェクトを戻します。

プロセス・オブジェクトの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
process.close();
```

MQProcess コンストラクターを使用してプロセスを作成することもできます。 各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、accessProcess メソッドの場合とまったく同じものです。 以下に例を示します。

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

この方法でプロセス・オブジェクトを構成すると、MQProcess のユーザー固有のサブクラスを作成することもできます。

IBM MQ classes for Java でのメッセージの処理

メッセージは MQMessage クラスで表されます。 メッセージの書き込みと取得は、MQQueue および MQTopic のサブクラスを持つ MQDestination クラスのメソッドを使用して行います。

MQDestination クラスの put() メソッドを使用して、メッセージをキューまたはトピックに書き込みます。 MQDestination クラスの get() メソッドを使用してキューまたはトピックからメッセージを取得します。

MQPUT および MQGET でバイトの配列の書き込みと取得を行うプロシージャー型インターフェースとは異なり、Java プログラム言語では MQMessage クラスのインスタンスの書き込みと取得が行われます。 MQMessage クラスは、実際のメッセージ・データが入っているデータ・バッファーを、そのメッセージを記述しているすべての MQMD (メッセージ記述子) パラメーターおよびメッセージ・プロパティーと共にカプセル化します。

新規メッセージを作成するには、MQMessage クラスの新規インスタンスを作成し、writeXXX メソッドを使用してデータをメッセージ・バッファーに書き込みます。

新しいメッセージ・インスタンスが作成されると、すべての MQMD パラメーターがデフォルト値に自動的に設定されます。 詳細については、MQMD の初期値および言語ごとの宣言を参照してください。 また、MQDestination の put() メソッドは、パラメーターとして MQPutMessageOptions クラスのインスタンスを取ります。 このクラスは MQPMO 構造体を表します。 次の例では、メッセージを作成して、キューに書き込んでいます。

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.put(myMessage,pmo);
```

MQDestination の get() メソッドは、MQMessage の新しいインスタンスを戻します。 これはキューから取られた直後のメッセージを表します。 また、このメソッドは MQGetMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQGMO 構造体を表します。

get() メソッドは自動的にその内部バッファーのサイズを着信メッセージが収まるように調整するので、最大メッセージ・サイズの指定は不要です。 戻されたメッセージ中のデータにアクセスするには、MQMessage クラスの readXXX メソッドを使用します。

次の例は、メッセージをキューから読み取る方法を示しています。

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo); // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

encoding メンバー変数を設定することにより、読み取りメソッドと書き込みメソッドで使用する数字形式を変更できます。

characterSet メンバー変数を設定することにより、読み取りと書き込みのストリングで使用する文字セットを変更できます。

詳細については、MQMessage クラスを参照してください。

注 : MQMessage の writeUTF() メソッドでは、含まれている Unicode バイト数に加えてストリングの長さも自動的にエンコードされます。 メッセージが別の Java プログラムによって (readUTF() を使用して) 読み取られる場合、これがストリング情報を送信する最も簡単な方法です。

IBM MQ classes for Java における非永続メッセージのパフォーマンス向上

メッセージをブラウズしたり、クライアント・アプリケーションから非永続メッセージをコンシュームしたりする際に、パフォーマンスの改善のために先読み を使用することができます。 MQGET または非同期

コンシュームを使用するクライアント・アプリケーションは、メッセージをブラウズする際に、または非永続メッセージをコンシュームする際に、パフォーマンスの改善による益を得ます。

先読み機能の一般情報については、関連トピックを参照してください。

IBM MQ classes for Java において、MQQueue または MQTopic オブジェクトの CMQC.MQSO_READ_AHEAD プロパティーおよび CMQC.MQSO_NO_READ_AHEAD プロパティーを使用して、メッセージ・コンシューマーおよびキュー・ブラウザーがそのオブジェクトに対して先読みができるかどうかを決定します。

IBM MQ classes for Java を使用したメッセージの非同期書き込み
メッセージを非同期に書き込むには、MQPMO_ASYNC_RESPONSE を設定します。

MQDestination クラスの put() メソッドを使用して、メッセージをキューまたはトピックに書き込みます。メッセージを非同期に書き込む、つまりキュー・マネージャーからの応答を待たずに操作が完了できるようにするには、MQPutMessageOptions の options フィールドで MQPMO_ASYNC_RESPONSE を設定します。 非同期書き込みが成功したか失敗したかを判断するには、MQQueueManager.getAsyncStatus 呼び出しを使用します。

IBM MQ classes for Java でのパブリッシュ/サブスクライブ

IBM MQ classes for Java では、トピックは MQTopic クラスによって表され、それに対するパブリッシュは MQTopic.put() メソッドを使用して行われます。

IBM MQ パブリッシュ/サブスクライブに関する一般情報については、「パブリッシュ/サブスクライブ・メッセージング」を参照してください。

IBM MQ classes for Java を使用した *IBM MQ* メッセージ・ヘッダーの処理

さまざまなタイプのメッセージ・ヘッダーを表す Java クラスが提供されています。 2 つのヘルパー・クラスも提供されています。

MQHeader インターフェース

ヘッダー・オブジェクトは MQHeader インターフェースにより記述されます。これはヘッダー・フィールドへのアクセスおよびメッセージ内容の読み書きのための汎用メソッドを提供します。 各ヘッダー・タイプは MQHeader インターフェースを実装する固有のクラスを持ち、それぞれのフィールドの getter メソッドおよび setter メソッドを追加します。 例えば、MQRFH2 ヘッダー・タイプは MQRFH2 クラスにより表され、MQDLH ヘッダー・タイプは MQDLH クラスにより表される、という具合になります。 ヘッダー・クラスは必要なデータ変換を自動的に実行し、指定された任意の数値エンコードまたは文字セット (CCSID) でデータを読み書きできます。

重要 : MQRFH2 ヘッダー・クラスは、メッセージをランダム・アクセス・ファイルとして扱います。そのため、カーソルはメッセージの先頭に置かれている必要があります。 MQRFH、 MQRFH2、MQCIH、MQDEAD、MQIIH、または MQXMIT などの内部メッセージ・ヘッダー・クラスを使用する前に、メッセージをクラスに渡す前に、メッセージのカーソル位置を正しい位置に更新してください。

ヘルパー・クラス

MQHeaderIterator および MQHeaderList の 2 つのヘルパー・クラスは、メッセージ内のヘッダー内容の読み取りとデコード (構文解析) を支援します。

- MQHeaderIterator クラスは、java.util.Iterator のように機能します。 メッセージ内にさらに多くのヘッダーがあれば、next() メソッドは true を戻し、nextHeader() または next() メソッドは次のヘッダー・オブジェクトを戻します。
- MQHeaderList は、java.util.List のように機能します。 MQHeaderIterator のように、これはヘッダー内容を構文解析しますが、特定のヘッダーの検索、新規ヘッダーの追加、既存のヘッダーの除去、ヘッダー・フィールドの更新、およびヘッダー内容のメッセージへの書き戻しもできます。 別の方法として、空の MQHeaderList を作成して、それにヘッダー・インスタンスを取り込み、それをメッセージに一度または繰り返し書き込むことができます。

MQHeaderIterator および MQHeaderList クラスは、MQHeaderRegistry 内の情報を使用して、どの IBM MQ ヘッダー・クラスが特定のメッセージ・タイプおよびメッセージ形式と関連しているかを把握します。 MQHeaderRegistry は、現行のすべての IBM MQ フォーマットとヘッダー・タイプ、およびその実装クラスについての知識に従って構成され、ユーザー固有のヘッダー・タイプを登録することもできます。

一般的に使用される以下の IBM MQ ヘッダーもサポートされています。

- MQRFH - 規則およびフォーマット・ヘッダー
- MQRFH2 – MQRFH のように、IBM Integration Bus に属するメッセージ・ブローカーとのメッセージのやり取りに使用されます。 メッセージ・プロパティーを含めるためにも使用されます。
- MQCIH - CICS ブリッジ
- MQDLH - 送達不能ヘッダー
- MQIIH - IMS 情報ヘッダー
- MQRMH - 参照メッセージ・ヘッダー
- MQSAPH - SAP ヘッダー
- MQWIH - 作業情報ヘッダー
- MQXQH - 伝送キュー・ヘッダー
- MQDH - 配布ヘッダー
- MQEPH - カプセル化 PCF ヘッダー

ユーザー固有のヘッダーを表すクラスを定義することもできます。

MQHeaderIterator を使用して RFH2 ヘッダーを取得するには、GetMessageOptions の MQGMO_PROPERTIES_FORCE_MQRFH2 を設定するか、またはキュー・プロパティー PROPCTL を FORCE に設定します。

IBM MQ classes for Java を使用した、メッセージ内のすべてのヘッダーの出力
この例では、MQHeaderIterator のインスタンスは、キューから受け取った MQMessage 内のヘッダーを構文解析します。 nextHeader() メソッドから戻された MQHeader オブジェクトは、toString メソッドが呼び出されると、その構造と内容を表示します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

IBM MQ classes for Java を使用した、メッセージ内のヘッダーのスキップオーバー
この例では、MQHeaderIterator の skipHeaders() メソッドは、メッセージ読み取りカーソルを最後のヘッダーの直後に配置します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

IBM MQ classes for Java を使用した、送達不能メッセージ内の理由コードの検索
この例では、read メソッドは、メッセージからの読み取りを行って、MQDLH オブジェクトへのデータの取り込みを行います。 読み取り操作の後に、メッセージ読み取りカーソルは MQDLH ヘッダー内容の直後に配置されます。

キュー・マネージャーの送達不能キューにあるメッセージには、送達不能ヘッダー (MQDLH) が接頭部に付きます。 これらのメッセージを処理する方法を決定するために (例えばそれらを再試行するか破棄するかを決定する)、送達不能処理アプリケーションは、MQDLH に含まれている理由コードを参照する必要があります。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

すべてのヘッダー・クラスは、単一ステップでメッセージから直接に自分自身を初期化するための、便利なコンストラクターを備えています。 そのため、この例のコードは以下のように単純化することができます。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

IBM MQ classes for Java を使用した、送達不能メッセージからのヘッダーの読み取りおよび除去
この例では、送達不能メッセージからのヘッダーの除去に MQDLH が使用されています。

送達不能処理アプリケーションは、理由コードが一時エラーを示している場合は、通常、拒否されたメッセージを再処理依頼します。 メッセージを再処理依頼する前に、MQDLH ヘッダーを除去する必要があります。

この例は、以下のステップを実行します (サンプル・コード内のコメントを参照してください)。

1. MQHeaderList はメッセージ全体を読み取り、メッセージ内で検出された各ヘッダーはリスト内の項目になります。
2. 送達不能メッセージには最初のヘッダーとして MQDLH が含まれるため、これはヘッダー・リスト内の最初の項目で検出できます。 MQDLH は、MQHeaderList の作成時にメッセージから既に取り込まれているので、その読み取りメソッドを呼び出す必要はありません。
3. MQDLH クラスにより提供される getReason() メソッドを使用して、理由コードが抽出されます。
4. 理由コードが検査され、メッセージの再処理依頼が適切であることが示されます。 MQHeaderList remove() メソッドを使用して、MQDLH が除去されます。
5. MQHeaderList は、その残りの内容を新規メッセージ・オブジェクトに書き込みます。 新規メッセージには、MQDLH を除くオリジナル・メッセージのすべてが含まれ、キューに書き込めるようになります。コンストラクターおよび write メソッドへの **true** 引数は、メッセージ本体が MQHeaderList 内に保持され、再度書き出されることを示します。
6. 新規メッセージのメッセージ記述子の format フィールドには、以前には MQDLH フォーマット・フィールドにあった値が含まれることになります。 メッセージ・データは、メッセージ記述子内の数値エンコードおよび CCSID セットと一致します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
```

```
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

IBM MQ classes for Java を使用した、メッセージの内容の出力
この例では、ヘッダーも含むメッセージの内容を印刷するために MQHeaderList を使用します。

出力には、メッセージの本体に加え、すべてのヘッダー内容のビューも含まれています。 MQHeaderList クラスはすべてのヘッダーを一度の実行でデコードしますが、MQHeaderIterator はそれらを、アプリケーション制御下で一度に 1 つずつステップスルーします。 この技法は、Websphere MQ アプリケーションの作成時の単純なデバッグ・ツールを備えるために使用できます。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

この例は、MQMD クラスを使用して、メッセージ記述子フィールドも印刷します。
com.ibm.mq.headers.MQMD クラスの copyFrom() メソッドは、メッセージ本体の読み取りによってではなく、MQMessage のメッセージ記述子フィールドから、ヘッダー・オブジェクトに取り込みを行います。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

IBM MQ classes for Java を使用した、メッセージ内の特定タイプのヘッダーの検索
この例では、MQHeaderList の indexOf(String) メソッドを使用して、メッセージ内の MQRFH2 ヘッダーを (もしあれば) 検索します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
        ...
}
```

IBM MQ classes for Java を使用した、*MQRFH2* ヘッダーの分析
この例は、MQRFH2 クラスを使用して、指定されたフォルダー内の既知のフィールド値にアクセスする方法を示しています。

MQRFH2 クラスは、構造体の固定部分のフィールドだけでなく、NameValueData フィールドで伝送される XML エンコード・フォルダーの内容にもアクセスする多数の方法を提供します。 この例は、指定されたフォルダー内の既知のフィールド値にアクセスする方法を示しています。この例では、jms フォルダーの Rto フィールドにアクセスします。これは、MQ JMS メッセージの応答キュー名を表しています。

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

(特定のフィールドを直接要求する場合とは異なり) MQRFH2 の内容を調べるには、getFolders メソッドを使用して MQRFH2.Element のリストを戻すことができます。これはフォルダーの構造を表すものであり、これにはフィールドや他のフォルダーが含まれていることがあります。 フィールドまたはフォルダーをヌルに設定すると、それが MQRFH2 から除去されます。 この方法で NameValueData フォルダーの内容を操作すると、それに応じて StrucLength フィールドが自動的に更新されます。

IBM MQ classes for Java を使用した、*MQMessage* オブジェクト以外のバイト・ストリームの読み取りおよび書き込み

以下の例では、データ・ソースが MQMessage オブジェクトではない場合に、ヘッダー・クラスを使用して、IBM MQ ヘッダー内容を構文解析および操作します。

データ・ソースが MQMessage オブジェクト以外のものである場合でも、IBM MQ ヘッダー内容を構文解析および操作するために、ヘッダー・クラスを使用できます。 すべてのヘッダー・クラスにより実装される MQHeader インターフェースは、メソッド `int read (java.io.DataInput message, int encoding, int characterSet)` および `int write (java.io.DataOutput message, int encoding, int characterSet)` を提供します。 com.ibm.mq.MQMessage クラスは、java.io.DataInput および java.io.DataOutput インターフェースを実装します。 これはつまり、2 つの MQHeader メソッドを使用して、MQMessage の内容を読み書きし、メッセージ記述子で指定されたエンコードと CCSID をオーバーライドできることを意味します。 これは、エンコードの異なる一連のヘッダーを含むメッセージの場合に役立ちます。

DataInput および DataOutput オブジェクトは、ファイルやソケットのストリーム、または JMS メッセージで伝送されるバイト配列などの、他のデータ・ストリームから取得することもできます。 java.io.DataInputStream クラスは DataInput を実装し、java.io.DataOutputStream クラスは DataOutput を実装します。 この例では、IBM MQ ヘッダーの内容をバイト配列から読み取ります。

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

先頭が MQHeaderIterator の行は、以下で置き換えることができます。

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

この例は、DataOutputStream を使用してバイト配列に書き込みます。

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

この方法でストリームを処理する場合は、encoding および characterSet 引数に正しい値を使用するように注意してください。 ヘッダーの読み取り時には、最初にバイト内容を書き込んだときに使用したエンコードおよび CCSID を指定します。 ヘッダーの書き込み時には、生成したいエンコードおよび CCSID を指定します。 データ変換は、ヘッダー・クラスにより自動的に実行されます。

IBM MQ classes for Java を使用した、新規ヘッダー・タイプのクラスの作成

IBM MQ classes for Java で提供されていないヘッダー・タイプの Java クラスを作成できます。

IBM MQ classes for Java で提供されるヘッダー・クラスと同じ方法で使用できる新しいヘッダー・タイプを表す Java クラスを追加するには、MQHeader インターフェースを実装するクラスを作成します。 最も簡

単な方法は、com.ibm.mq.headers.impl.Header クラスを拡張することです。 この例では、MQTM ヘッダー構造体を表す完全機能クラスを作成します。 それぞれのフィールドに個別の getter メソッドおよび setter メソッドを追加する必要はありませんが、これはヘッダー・クラスのユーザーには便利なものです。 フィールド名のストリングを取る汎用 getValue メソッドおよび汎用 setValue メソッドは、ヘッダー・タイプで定義されるすべてのフィールドで機能します。 継承された read メソッド、write メソッド、および size メソッドにより、新規ヘッダー・タイプのインスタンスの読み取りおよび書き込みが可能になり、そのフィールド定義に基づいてヘッダー・サイズが正確に計算されます。 タイプ定義は一度だけ作成されますが、このヘッダー・クラスのインスタンスは多数作成されます。 MQHeaderIterator クラスまたは MQHeaderList クラスを使用して新規ヘッダー定義をデコードできるようにするには、MQHeaderRegistry を使用してこれを登録します。 ただし、MQTM ヘッダー・クラスは実際にはこのパッケージで既に提供され、デフォルトのレジストリーで登録されていることに注意してください。

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);

    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

IBM MQ classes for Java での PCF メッセージの処理

Java クラスは、PCF 構造化メッセージを作成および構文解析し、PCF 要求の送信と PCF 応答の収集に役立てるために提供されています。

クラス PCFMessage および MQCFGR は、PCF パラメーター構造体の配列を表します。 これらは PCF パラメーターを追加および取得するための便利なメソッドを提供します。

PCF パラメーター構造体は、クラス MQCFH、MQCFIN、MQCFIN64、MQCFST、MQCFBS、MQCFIL、MQCFIL64、MQCFSL、および MQCFGR で表されます。 これらは以下の基本操作インターフェースを共有します。

- メッセージ内容の読み取りおよび書き込みのためのメソッド: read ()、write ()、および size ()
- パラメーターの操作のためのメソッド: getValue ()、setValue ()、getParameter ()、その他
- MQMessage 内の PCF の内容を構文解析する、列挙子メソッド .nextParameter ()

PCF フィルター・パラメーターは、フィルター機能を提供する inquire コマンドで使用されます。 これは以下のクラスでカプセル化されます。

- MQCFIF - 整数フィルター
- MQCFSF - ストリング・フィルター

- MQCFBF - バイト・フィルター

PCFAgent と PCFMessageAgent の 2 つのエージェント・クラスが、キュー・マネージャー、コマンド・サーバー・キュー、および関連応答キューへの接続を管理するために提供されています。 PCFMessageAgent は PCFAgent の拡張版であるため、通常はこちらを優先して使用すべきです。 PCFMessageAgent クラスは受け取った MQMessages を変換し、それらを PCFMessage 配列として呼び出し元に戻します。 PCFAgent は MQMessages の配列を戻しますが、これは使用する前に構文解析する必要があります。

IBM MQ classes for Java でのメッセージ・プロパティーの処理

IBM MQ classes for Java には、メッセージ・ハンドルを処理する関数呼び出しに相当するものはありません。 メッセージ・ハンドルのプロパティーを設定する、戻す、または削除するには、MQMessage クラスのメソッドを使用します。

メッセージ・プロパティーに関する一般情報については、28 ページの『プロパティー名』を参照してください。

IBM MQ classes for Java では、メッセージに対するアクセスは MQMessage クラスを通して行います。 したがって、Java 環境においてメッセージ・ハンドルは用意されておらず、IBM MQ 関数呼び出し MQCRTMH、MQDLTMH、MQMHBUF、および MQBUFMH に対応するものはありません。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーを設定するには MQSETMP 呼び出しを使用します。 IBM MQ classes for Java では MQMessage クラスの該当するメソッドを使用します。

- setBooleanProperty
- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty
- setStringProperty
- setObjectProperty

これらは、*set*property* メソッドと総称される場合があります。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーの値を戻すには MQINQMP 呼び出しを使用します。 IBM MQ classes for Java では MQMessage クラスの該当するメソッドを使用します。

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty

- getDoubleProperty
- getStringProperty
- getObjectProperty

これらは、*get*property* メソッドと総称される場合があります。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーの値を削除するには MQDLTMP 呼び出しを使用します。 IBM MQ classes for Java では、MQMessage クラスの deleteProperty メソッドを使用します。

IBM MQ classes for Java でのエラーの処理

IBM MQ classes for Java で発生したエラーを、Java `try` および `catch` ブロックを使用して処理します。

Java インターフェースのメソッドは完了コードと理由コードを戻しません。 その代わりに、IBM MQ の呼び出しによる完了コードと理由コードが両方ともゼロでない場合は必ず例外をスローします。 これによってプログラム・ロジックが簡単になり、IBM MQ への呼び出しごとに戻りコードを検査する必要がなくなります。 プログラムのどの部分で障害に対処するかを決めることができます。 これらのポイントでは、次の例のように、コードを `try` と `catch` のブロックで囲むことができます。

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                       "CC = " + ex.completionCode +
                       "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

z/OS の Java 例外で報告される IBM MQ 呼び出し理由コードは、 API 完了コードと理由コードに記載されています。

IBM MQ classes for Java アプリケーションの実行中にスローされる例外も、ログに書き込まれます。 ただし、アプリケーションは MQException.logExclude() メソッドを呼び出して、指定の理由コードに関連した例外はログに記録されないようにすることができます。 この処理は、指定の理由コードに関連した例外が多数スローされると予想されるため、ログがこれらの例外で一杯にならないようにしたい、という場合に使用できます。 例えば、アプリケーションがループを反復するたびにキューからメッセージを取得しようとするときに、そのほとんどの試行で、適切なメッセージがキュー上にないと予想される場合は、理由コード MQRC_NO_MSG_AVAILABLE に関連した例外をログに記録しないようにできます。 アプリケーションは以前に指定の理由コードに関連した例外をログに記録しないようにした場合は、メソッド MQException.logInclude() を呼び出せば、それらの例外を再度ログに記録できます。

理由コードが、エラーに関連する詳細を必ずしも全部は伝えてはいない場合があります。 アプリケーションは、スローされる例外ごとにリンク付きの例外も検査する必要があります。 リンク付きの例外自体には、別のリンク付きの例外がある場合があるため、リンク付きの例外は元の根本の問題に戻るチェーンを形成します。 リンク付きの例外は、java.lang.Throwable クラスのチェーニングされた例外メカニズムを使用して実装され、アプリケーションは Throwable.getCause() メソッドを呼び出してリンク付き例外を取得します。 com.ibm.mq.jmqi.JmqiException の基礎となるインスタンスは MQException のインスタンスである例外から MQException.getCause() で取得され、このエラーを発生させた原因となる java.lang.Exception は、この例外から getCause によって取得されます。

IBM MQ classes for Java 内の属性値の取得および設定

getXXX() および setXXX() メソッドが多数の共通属性に対して提供されています。 その他の属性は、汎用 inquire() および set() メソッドを使用してアクセスできます。

多くの共通属性では、MQManagedObject、MQDestination、MQQueue、MQTopic、MQProcess、および MQQueueManager クラスに getXXX() メソッドと setXXX() メソッドが含まれており、これらのメソッドによって、その属性値を取得および設定できます。 ただし、MQDestination、MQQueue、および MQTopic の

場合、これらのメソッドは、オブジェクトのオープン時に適切な inquire フラグおよび set フラグを指定した場合にのみ機能します。

あまり使用されない属性については、MQQueueManager、MQDestination、MQQueue、MQTopic、および MQProcess クラスはすべて、MQManagedObject と呼ばれるクラスから継承します。 このクラスは inquire() および set() のインターフェースを定義します。

new 演算子を使用して新規キュー・マネージャー・オブジェクトを作成すると、そのオブジェクトは自動的に inquire 用にオープンされます。 また、accessProcess() メソッドを使用してプロセス・オブジェクトにアクセスすると、そのオブジェクトは自動的に inquire 用にオープンされます。 しかし、accessQueue() メソッドを使用してキュー・オブジェクトにアクセスした場合は、そのオブジェクトは自動的には inquire または set 操作用にはオープンされません。 これは、これらのオプションを自動的に追加すると、一部のタイプのリモート・キューで問題を生じさせる可能性があるためです。 キューに対して inquire、set、getXXX、および setXXX メソッドを使用するには、適切な「inquire」フラグと「set」フラグを accessQueue() メソッドの openOptions パラメーターに指定しなければなりません。 同じことが、宛先およびトピック・オブジェクトの場合にも言えます。

inquire メソッドと set メソッドは、次の 3 つのパラメーターを取ります。

- selectors 配列
- intAttrs 配列
- charAttrs 配列

Java では、配列の長さが常に認識されているので、MQINQ にある SelectorCount、IntAttrCount、および CharAttrLength パラメーターは不要です。 次の例は、キューの照会を行う方法を示しています。

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Java におけるマルチスレッド・プログラム

Java ランタイム環境は本質的にマルチスレッド環境です。 IBM MQ classes for Java では、キュー・マネージャー・オブジェクトを複数のスレッド間で共用できますが、ターゲット・キュー・マネージャーへの全アクセスが確実に同期するようになります。

Java では、マルチスレッド・プログラムは避け難いものです。 始動時にキュー・マネージャーに接続して、キューをオープンする単純なプログラムを考えてみましょう。 このプログラムは、画面上に 1 つのボタンを表示します。 ユーザーがこのボタンをクリックすると、プログラムはキューからメッセージを取り出します。

Java ランタイム環境は本質的にマルチスレッド環境です。 したがって、ユーザー・アプリケーションの初期化はある 1 つのスレッドで実行され、ボタンが押された応答で実行されるコードは別のスレッド (ユーザー・インターフェース・スレッド) で実行されます。

C ベースの IBM MQ MQI client では、複数のスレッド間でのハンドルの共用に制限があるため、この処理は問題となります。 IBM MQ classes for Java では、この制約が緩和されているため、キュー・マネージャー・オブジェクト (さらに、それと関連したキュー・オブジェクト、トピック・オブジェクト、およびプロセス・オブジェクト) を複数のスレッド間で共用できます。

IBM MQ classes for Java を実装すると、特定の接続 (MQQueueManager オブジェクト・インスタンス) に対しては、ターゲットの IBM MQ キュー・マネージャーへのすべてのアクセスは、必ず同期化されます。 キュー・マネージャーに呼び出しを発行するスレッドは、その接続で進行中の他の呼び出しがすべて完了

するまでブロックされます。 プログラム内の複数のスレッドから同じキュー・マネージャーに同時にアクセスする必要がある場合は、同時アクセスが必要なスレッドごとに新しい MQQueueManager オブジェクトを作成します。 (これは、スレッドごとに別の MQCONN 呼び出しを発行する ことと同じです。)

注 : クラス `com.ibm.mq.MQGetMessageOptions` のインスタンスは、メッセージを同時に要求する複数のスレッド間で共用することはできません。 このクラスのインスタンスは、対応する MQGET 要求の間にデータが更新されます。そのため、このオブジェクトの同じインスタンスを複数のスレッドが同時に処理しようとすると、予期しない結果になることがあります。

IBM MQ classes for Java でのチャネル出口の使用

IBM MQ classes for Java を使用するアプリケーションでのチャネル出口の使用方法の概要。

以下のトピックでは、Java でチャネル出口を作成する方法、その割り当て方法、およびデータをそのチャネル出口に渡す方法を説明します。 その後、C で作成されたチャネル出口の使用方法および一連のチャネル出口の使用方法を説明します。

アプリケーションに、チャネル出口クラスをロードするための適切なセキュリティー権限が必要です。

IBM MQ classes for Java でのチャネル出口の作成

適切なインターフェースを実装する Java クラスを定義することによって、固有のチャネル出口を提供することができます。

出口を実装するには、適切なインターフェースを実装する新規 Java クラスを定義します。
com.ibm.mq.exits パッケージには、次の 3 つの出口インターフェースが定義されています。

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

注 : チャネル出口は、クライアント接続でのみサポートされます。バインディング接続ではサポートされません。 IBM MQ classes for Java の外部で Java チャネル出口を使用することはできません。例えば、C で作成されたクライアント・アプリケーションを使用する場合などです。

接続に定義されている TLS 暗号化はすべて、送信出口およびセキュリティー出口が呼び出された後で 実行されます。 同様に暗号化解除は、受信およびセキュリティー出口が呼び出される 前に 実行されます。

以下の例は、3 つのインターフェースすべてを実装するクラスを定義しています。

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                  MQCD channelDefinition,
                                  ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                     MQCD channelDefinition,
                                     ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                      MQCD channelDefinition,
                                      ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

出口ごとに、MQCXP オブジェクトおよび MQCD オブジェクトが渡されます。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

作成する出口クラスには、コンストラクターがなければなりません。 これはデフォルト・コンストラクターまたはストリング引数を持つコンストラクターのいずれかです。 それがストリングを取る場合、ユーザー・データは作成時に出口クラスに渡されます。 出口クラスにデフォルトのコンストラクターと単一引数コンストラクターの両方が含まれる場合、単一引数コンストラクターが優先されます。

送信出口およびセキュリティー出口の場合、出口コードは、サーバーに送信されるデータを戻す必要があります。 受信出口の場合、出口コードは、IBM MQ に解釈させる変更済みデータを戻す必要があります。

次は、考えられる最も単純な出口の本体です。

```
{ return agentBuffer; }
```

キュー・マネージャーはチャネル出口内からはクローズしないでください。

既存のチャネル出口クラスの使用

IBM MQ の 7.0 より前のバージョンでは、これらの出口を以下の例のようにインターフェース MQSendExit、MQReceiveExit、および MQSecurityExit を使用して実装します。 この方法は有効なままですが、機能性とパフォーマンスを向上させたい場合には、新しい方法をお勧めします。

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

IBM MQ classes for Java でのチャネル出口の割り当て
IBM MQ classes for Java を使用してチャネル出口を割り当てることができます。

IBM MQ classes for Java の IBM MQ チャネルに直接相当するものはありません。 チャネル出口は MQQueueManager に割り当てられます。 例えば、WMQSecurityExit インターフェースを実装するクラスが定義されている場合、アプリケーションは次の 4 とおりの方法でセキュリティー出口を使用できます。

- MQQueueManager オブジェクトを作成する前に、クラスのインスタンスを MQEnvironment.channelSecurityExit フィールドに割り当てる
- MQQueueManager オブジェクトを作成する前に、MQEnvironment.channelSecurityExit フィールドをセキュリティー出口クラスを表すストリングに設定する
- MQQueueManager へ CMQC.SECURITY_EXIT_PROPERTY のキーで渡されるプロパティーのハッシュ・テーブル内にキー/値のペアを作成する
- クライアント・チャネル定義テーブル (CCDT) の使用

MQEnvironment.channelSecurityExit フィールドをストリングに設定することにより、プロパティーのハッシュ・テーブルにキー/値のペアを作成することにより、または CCDT を使用することによって割り当てられる出口は、デフォルト・コンストラクターで作成されていなければなりません。 アプリケーションによっては、クラスのインスタンスとして割り当てられた出口には、デフォルト・コンストラクターは必要ありません。

アプリケーションは、同様の方法で送信出口または受信出口を使用できます。 例えば、以下のコード・フラグメントは、クラス MyMQExits (MQEnvironment を使用して以前に定義済み) に実装されるセキュリティー出口、送信出口、および受信出口の使用方法を示しています。

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

複数のメソッドを使用することによってチャネル出口が割り当てられた場合、優先順位は次のとおりです。

1. CCDT の URL が MQQueueManager に渡された場合、CCDT の内容によって使用されるチャネル出口が決定され、MQEnvironment またはプロパティーのハッシュ・テーブル内の出口定義は無視されます。
2. CCDT URL が渡されていない場合、MQEnvironment およびハッシュ・テーブルからの出口定義がマージされます。
   - 同じ出口タイプが MQEnvironment とハッシュ・テーブルの両方で定義されている場合、ハッシュ・テーブル内の定義が使用されます。
   - 旧タイプと新タイプで等価の出口が指定されている場合 (例えば、IBM WebSphere MQ 7.0 より前のバージョンで使用されるタイプの出口にのみ使用できる sendExit フィールドと、どの送信出口にも使用できる channelSendExit フィールド)、旧出口ではなく新規の出口 (channelSendExit) が使用されます。

チャネル出口をストリングとして宣言した場合、IBM MQ がチャネル出口プログラムの場所を探索できるようにする必要があります。 これは、アプリケーションが稼働している環境およびチャネル出口プログラムのパッケージ方法に応じて、さまざまな方法で実行できます。

- アプリケーション・サーバーで実行するアプリケーションの場合は、ファイルを 395 ページの表 58 で示されたディレクトリーに格納するか、または **exitClasspath** によって参照される JAR ファイルにパッケージする必要があります。
- アプリケーション・サーバーで実行しないアプリケーションの場合は、以下の規則が適用されます。
  - チャネル出口クラスが別の JAR ファイルにパッケージされている場合、これらの JAR ファイルは **exitClasspath** に含まれていなければなりません。
  - チャネル出口クラスが JAR ファイルにパッケージされていない場合、クラス・ファイルは 395 ページの表 58 で示されたディレクトリーまたは JVM システム・クラスパスまたは **exitClasspath** 内の任意のディレクトリーに保管できます。

exitClasspath プロパティーは 4 とおりの方法で指定できます。 これらの方法を優先度の順に以下に示します。

1. システム・プロパティー com.ibm.mq.exitClasspath (コマンド行で `-D` オプションを使用して定義される)
2. `mqclient.ini` ファイルの exitPath スタンザ
3. キー CMQC.EXIT_CLASSPATH_PROPERTY を使用したハッシュ・テーブル項目
4. MQEnvironment 変数 **exitClasspath**

java.io.File.pathSeparator 文字を使用した個別の複数のパス。

表 58. チャネル出口プログラムのディレクトリー	
プラットフォーム	**ディレクトリー**
AIX Linux	/var/mqm/exits (32 ビット・チャネル出口プログラム) /var/mqm/exits64 (64 ビット・チャネル出口プログラム)
Windows	*install_data_dir*¥exits

注 : *install_data_dir* は、インストール中に IBM MQ データ・ファイル用に選択したディレクトリーです。 デフォルト・ディレクトリーは C: ¥ProgramData¥IBM¥MQ です。

IBM MQ classes for Java でのチャネル出口へのデータの引き渡し
チャネル出口へデータを渡したり、チャネル出口からアプリケーションへデータを戻したりすることができます。

agentBuffer パラメーター

送信出口の場合、*agentBuffer* パラメーターには、送信される直前のデータが入ります。 受信出口やセキュリティー出口の場合は、受信された直後のデータが *agentBuffer* パラメーターに入れられます。 配列の長さは式 agentBuffer.limit() で指示されるため、長さパラメーターは必要ありません。

送信出口およびセキュリティー出口の場合、出口コードは、サーバーに送信されるデータを戻す必要があります。 受信出口の場合、出口コードは、IBM MQ に解釈させる変更済みデータを戻す必要があります。

次は、考えられる最も単純な出口の本体です。

```
{ return agentBuffer; }
```

チャネル出口は、バッキング配列を持つバッファーとともに呼び出されます。 最良のパフォーマンスを得るためには、出口はバッキング配列を持つバッファーを戻す必要があります。

ユーザー・データ

channelSecurityExit、channelSendExit、または channelReceiveExit を設定することによってアプリケーションがキュー・マネージャーに接続する場合、channelSecurityExitUserData、channelSendExitUserData、または channelReceiveExitUserData フィールドを使用して、該当するチャネル出口クラスの呼び出し時に 32 バイトのユーザー・データをチャネル出口クラスに渡すことができます。 このユーザー・データはチャネル出口クラスに使用できますが、出口が呼び出されるたびにリフレッシュされます。 したがって、チャネル出口内でユーザー・データに加えられた変更はすべて失われます。 チャネル出口内のデータに永続的な変更を加える場合は、MQCXP exitUserArea を使用してください。 このフィールドのデータは、出口が呼び出される間も維持されます。

アプリケーションが securityExit、sendExit、または receiveExit を設定した場合、これらのチャネル出口クラスへはユーザー・データを渡すことはできません。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーと接続する場合、チャネル出口クラスが呼び出されるときに、クライアント接続チャネル定義で指定されたユーザー・データがチャネル出口クラスに渡されます。 クライアント・チャネル定義テーブルの使用については、377 ページの『IBM MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』を参照してください。

Java で作成されていないチャネル出口を *IBM MQ classes for Java* で使用する
C で作成されたチャネル出口プログラムを Java アプリケーションから使用する方法。

IBM MQ では、C 言語で作成されたチャネル出口プログラムの名前を、MQEnvironment オブジェクトまたはプロパティーのハッシュ・テーブル内の channelSecurityExit、channelSendExit、または channelReceiveExit フィールドへ渡すストリングとして指定することができます。 ただし、Java で記述されたチャネル出口プログラムを別の言語で記述されたアプリケーションで使用することはできません。

出口プログラム名は library(function) の形式で指定し、出口プログラムの場所が出口へのパスに示されているとおりに指定されていることを確認してください。

C でチャネル出口を作成する方法については、967 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

IBM MQ classes for Java での一連のチャネル送信出口または受信出口の使用
IBM MQ classes for Java アプリケーションでは、連続して実行される、一連のチャネル送信出口または受信出口を使用できます。

一連の送信出口を使用するために、アプリケーションは、送信出口を含むリストまたはストリングを作成できます。 リストを使用する場合、リストの各エレメントは以下のいずれかにします。

- WMQSendExit インターフェースを実装するユーザー定義クラスのインスタンス
- MQSendExit インターフェースを実装するユーザー定義クラスのインスタンス (Java で作成された送信出口の場合)
- MQExternalSendExit クラスのインスタンス (Java で作成されていない送信出口の場合)
- MQSendExitChain クラスのインスタンス
- ストリング・クラスのインスタンス

リストには別のリストを含めることはできません。

アプリケーションは、同様の方法で、一連の受信出口を使用できます。

ストリングを使用する場合は、1 つ以上のコンマ区切りの出口定義で構成する必要があります。各出口定義は、 Java クラスの名前、または library(function) という形式の C プログラムにすることができます。

これによりアプリケーションは、MQQueueManager オブジェクトを作成する前に、リストまたはストリング・オブジェクトを MQEnvironment.channelSendExit フィールドに割り当てます。

出口に受け渡される情報のコンテキストは、出口のドメイン内だけです。 例えば、Java 出口および C 出口がチェーニングされている場合、Java 出口の存在は C 出口に全く影響をもたらしません。

出口チェーン・クラスの使用

IBM WebSphere MQ 7.0 より前のバージョンでは、出口のシーケンスを許可するために以下の 2 つのクラスが提供されていました。

- MQSendExitChain。MQSendExit インターフェースを実装します。
- MQReceiveExitChain。MQReceiveExit インターフェースを実装します。

これらのクラスの使用は依然として有効ですが、新規のメソッドが推奨されます。 Java インターフェース用に IBM MQ クラスを使用することで、com.ibm.mq.jar 上で依存性がない場所で com.ibm.mq.exits パッケージのインターでーすの新しいセットが使用された場合に、アプリケーションはこれまでと変わらず com.ibm.mq.jar で依存性を保持します。

一連の送信出口を使用するために、アプリケーションは、オブジェクトのリストを作成しました。リスト内の各オブジェクトは、次のいずれかでした。

- MQSendExit インターフェースを実装するユーザー定義クラスのインスタンス (Java で作成された送信出口の場合)
- MQExternalSendExit クラスのインスタンス (Java で作成されていない送信出口の場合)
- MQSendExitChain クラスのインスタンス

アプリケーションは、このオブジェクト・リストをコンストラクターのパラメーターとして渡すことによって、MQSendExitChain オブジェクトを作成しました。 これによりアプリケーションは、MQQueueManager オブジェクトを作成する前に、MQSendExitChain オブジェクトを MQEnvironment.sendExit フィールドに割り当てることができました。

IBM MQ classes for Java でのチャネル圧縮

チャネルを流れるデータを圧縮すると、チャネルのパフォーマンスを改善し、ネットワーク・トラフィックを削減することができます。 IBM MQ classes for Java は、IBM MQ に構築されている圧縮機能を使用します。

IBM MQ で提供される機能を使用して、メッセージ・チャネルと MQI チャネルを流れるデータを圧縮できます。また、どちらのタイプのチャネルでも、ヘッダー・データとメッセージ・データを個別に圧縮できます。 デフォルトでは、チャネル上のデータは圧縮されません。 IBM MQ への実装方法など、チャネル圧縮の 詳細については、データ圧縮 (COMPMSG) およびヘッダー圧縮 (COMPHDR) を参照してください。

IBM MQ classes for Java アプリケーションは、クライアント接続のヘッダー・データまたはメッセージ・データの圧縮に使用できる手法を指定するために、java.util.Collection オブジェクトを作成します。 各圧縮手法は、このコレクション内の Integer オブジェクトであり、アプリケーションが圧縮手法をコレクションに追加する順序は、クライアント接続の開始時にキュー・マネージャーとの間で圧縮手法がネゴシエーションされる順序になります。 これによりアプリケーションは、MQEnvironment クラスの hdrCompList フィールド (ヘッダー・データの場合) または msgCompList フィールド (メッセージ・データの場合) にコレクションを割り当てることができます。 アプリケーションの準備ができたら、MQQueueManager オブジェクトを作成すればクライアント接続を開始できます。

以下のコード・フラグメントは、ここで説明した方法の例です。 最初のコード・フラグメントは、ヘッダー・データ圧縮の実装方法を示します。

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

2 番目のコード・フラグメントは、メッセージ・データ圧縮の実装方法を示します。

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));

:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

2 番目の例では、クライアント接続の開始時に、圧縮手法は RLE、ZLIBHIGH の順でネゴシエーションされます。 選択された圧縮手法は、MQQueueManager オブジェクトの存続期間中は変更できません。

クライアント接続のクライアントとキュー・マネージャーの両方でサポートされるヘッダー・データおよびメッセージ・データの圧縮手法は、MQChannelDefinition オブジェクトの hdrCompList フィールドおよび msgCompList フィールドのコレクションとして、チャネル出口に渡されます。 クライアント接続でヘッダー・データおよびメッセージ・データの圧縮に現在使用されている実際の手法は、MQChannelExit オブジェクトの CurHdrCompression フィールドおよび CurMsgCompression フィールドのチャネル出口に渡されます。

圧縮がクライアント接続で使用される場合、データは、チャネル送信出口が処理される前に圧縮され、チャネル受信出口が処理された後に抽出されます。 このため、送信出口および受信出口に渡されるデータは、圧縮された状態になります。

圧縮手法の指定および使用可能な圧縮手法の詳細については、クラス com.ibm.mq.MQEnvironment およびインターフェース com.ibm.mq.MQC を参照してください。

IBM MQ classes for Java における *TCP/IP* 接続の共用

MQI チャネルの複数インスタンスが、単一の TCP/IP 接続を共用するようにできます。

IBM MQ classes for Java では、単一の TCP/IP 接続を共用できる会話の数を、MQEnvironment.sharingConversations 変数を使用して制御します。

SHARECNV 属性は、接続共用へのベスト・エフォート・アプローチです。 したがって、IBM MQ classes for Java で 0 より大きい SHARECNV 値を使用した場合は、新しい接続要求は既に確立されている接続を常に共用するという保証はありません。

IBM MQ classes for Java での接続プール

IBM MQ classes for Java では、再利用のために予備の接続をプールすることができます。

IBM MQ classes for Java では、IBM MQ キュー・マネージャーへの複数接続を扱うアプリケーションのサポートが追加されました。 そのため、ある接続が必要でなくなった際に、その接続を破棄しないでプールに入れておき、後で再利用することが可能になります。 これは、任意のキュー・マネージャーに連続して接続するアプリケーションやミドルウェアのパフォーマンスを大幅に向上させます。

IBM MQ には、デフォルトの接続プールがあります。 アプリケーションは、MQEnvironment クラスでトークンを登録したり登録から外したりすることで、この接続プールをアクティブにしたり非アクティブにしたりできます。 プールがアクティブになっていると、IBM MQ classes for Java が MQQueueManager オブジェクトを作成する際にこのデフォルトのプールが検索され、適切な接続があればそれが再利用されます。 そして、MQQueueManager.disconnect() が呼び出されると、その下にある接続はプールに戻されます。

別の方法として、特定の用途のためにアプリケーションで MQSimpleConnectionManager 接続プールを構成することもできます。 こうして作成されたプールは、MQQueueManager オブジェクトの構成の際に指定することもできるほか、デフォルト接続プールとして使用するために MQEnvironment に渡すこともできます。

接続があまりに多くのリソースを使わないようにするために、MQSimpleConnectionManager オブジェクトが処理できる接続の総数を制限したり、接続プールの大きさを制限することができます。 JVM 内での接続に重複した需要がある場合、限度を設定することは便利です。

getMaxConnections() メソッドは、デフォルトで値ゼロを戻します。 これは、MQSimpleConnectionManager オブジェクトが処理できる接続数に制限がないことを意味します。 setMaxConnections() メソッドを使うことにより、制限を設定することができます。 制限を設定し、その限界に達した場合、それ以降の接続要求は MQException をスローすることになり、理由コードは MQRC_MAX_CONNS_LIMIT_REACHED になります。

IBM MQ classes for Java でのデフォルト接続プールの制御

この例では、デフォルト接続プールの使用方法を示します。

次のサンプル・アプリケーション MQApp1 について検討します。

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 は、コマンド行からローカル・キュー・マネージャーのリストを入手し、リスト内の各キュー・マネージャーに順番に接続して、何らかの操作を実行します。 しかし、コマンド行に同じキュー・マネージャーが何回もリストされているような場合は、接続の確立を 1 回のみとして、その接続を何回も再利用した方がより効率的です。

IBM MQ classes for Java ではデフォルト接続プールが提供されており、これを使用して接続を再利用できます。 このプールを使用可能にする場合は、いずれかの MQEnvironment.addConnectionPoolToken() メソッドを使用します。 プールを使用不可にする場合は、MQEnvironment.removeConnectionPoolToken() を使用します。

次のサンプル・アプリケーション MQApp2 は、機能的には MQApp1 と同じですが、それぞれのキュー・マネージャーに一度ずつしか接続しません。

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

このアプリケーションでは、1 つ目の太字になっている行で MQPoolToken オブジェクトを MQEnvironment に登録することにより、デフォルト接続プールが使用可能にされています。

MQQueueManager コンストラクターは、このプールに適当な接続がないかどうかを調べ、該当するキュー・マネージャーへの接続が存在していない場合にのみ接続を作成します。 使用された接続は、再利用された後、qmgr.disconnect() 呼び出しでプールに戻されます。 これらの API 呼び出しは、サンプル・アプリケーション MQApp1 と同じです。

2 つ目の強調表示されている行では、デフォルト接続プールが非アクティブにされています。これにより、そのプールに保管されているキュー・マネージャー接続はすべて破棄されます。 このように接続を破棄しないと、プール内のいくつかのキュー・マネージャー接続が使用されたままの状態でアプリケーションが終了されてしまうため、この処理は重要です。 接続が破棄されないままアプリケーションを終了すると、キュー・マネージャー・ログに記録されるようなエラーを引き起こす恐れがあります。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合、MQQueueManager コンストラクターは、まずテーブルを検索して、適切なクライアント接続チャネル定義を探します。 該当する定義が見つかった場合、コンストラクターは、チャネルに使用できる接続のデフォルト接続プールを検索します。 プール内で適切な接続が見つからなかった場合、コンストラクターは、クライアント・チャネル定義テーブルを検索して、次に適切なクライアント接続チャネル定義を探し、前述のとおりに処理します。 クライアント・チャネル定義テーブルの検索は完了したが、プール内で適切な接続が見つからなかった場合、コンストラクターはそのテーブルについて 2 回目の検索を開始します。 この検索では、コンストラクターは、適切な各クライアント接続チャネル定義について順番に新規接続の作成を試行して、最初に作成できた接続を使用します。

デフォルト接続プールでは、使用されていない接続を最大で 10 まで保管でき、使用されていない接続を最大で 5 分、アクティブに保つことができます。 この制限は、アプリケーションで変更できます (詳細は 400 ページの『IBM MQ classes for Java での異なる接続プールの提供』を参照してください)。

MQEnvironment を使用して MQPoolToken を用意する代わりに、次のように、アプリケーションで独自のものを構成することもできます。

```
   MQPoolToken token=new MQPoolToken();
   MQEnvironment.addConnectionPoolToken(token);
```

一部のアプリケーション・ベンダーやミドルウェアのベンダーでは、カスタム接続プールに情報を渡すために MQPoolToken のサブクラスを用意しています。 この方法で構成して addConnectionPoolToken() に渡すと、その接続プールに追加情報も渡すことができます。

IBM MQ classes for Java でのデフォルト接続プールと複数のコンポーネント
この例では、登録された MQPoolToken オブジェクトの静的な集合の MQPoolToken を追加または削除する方法を示します。

MQEnvironment は、登録された MQPoolToken オブジェクトの静的な集合を保持します。 この集合での MQPoolTokens の追加と除去には、次のメソッドを使用します。

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

アプリケーションは、独立して存在し、キュー・マネージャーを使用して作業を行ういくつものコンポーネントによって構成されている場合があります。 そのようなアプリケーションでは、それぞれのコンポーネントが、その存続期間の間、MQEnvironment の集合に MQPoolToken を追加する必要があります。

例えば、サンプル・アプリケーション MQApp3 では、10 のスレッドを作成し、それぞれを開始します。 各スレッドはそれぞれの MQPoolToken に登録し、ある程度の時間待機して、それからキュー・マネージャーに接続します。 接続が切断されると、スレッドはそれぞれの MQPoolToken を除去します。

デフォルト接続プールは、MQPoolTokens の集合に 1 つでもトークンが残っている限り、つまりこのアプリケーションの存続期間の間はアクティブのままとなっています。 アプリケーションは、これらのスレッド全体の制御において、マスター・オブジェクトを保持する必要はありません。

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

IBM MQ classes for Java での異なる接続プールの提供
この例は、クラス **com.ibm.mq.MQSimpleConnectionManager** を使用して別の接続プールを用意する方法を示しています。

このクラスには、接続プーリングのための基本的な機能が備わっており、アプリケーションはこのクラスを使用してプールの振る舞いをカスタマイズできます。

MQSimpleConnectionManager は、インスタンス化されると、MQQueueManager コンストラクターで指定できるようになります。 MQSimpleConnectionManager は、構成された MQQueueManager の下にある接続を管理するようになります。 MQSimpleConnectionManager にプールされた適切な接続が含まれる場合、その接続は再利用され、MQQueueManager.disconnect() 呼び出しが実行された後に MQSimpleConnectionManager に戻されます。

次のコード・フラグメントは、その動作を実例で示しています。

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

最初の MQQueueManager コンストラクターで作られた接続は、qmgr.disconnect() 呼び出しの後 myConnMan に保管されます。 この接続は、次に MQQueueManager コンストラクターが呼び出されたときに再利用されます。

2 番目の行では MQSimpleConnectionManager が使用可能にされます。 そして最後の行では MQSimpleConnectionManager が使用不可にされ、そのプールに保持されていた接続がすべて破棄されます。 なお、MQSimpleConnectionManager はデフォルトで MODE_AUTO になっていますが、これについてはこのセクションの後の方で説明します。

MQSimpleConnectionManager は、一番最近に使用された接続から順に割り振り、使用されてから最も時間が経過している接続から順に破棄していきます。 デフォルトでは、その接続が 5 分間使用されなかった場合と、プール内の使用されていない接続の数が 10 を超えた場合に、接続が破棄されます。 これらの値を変更するには、MQSimpleConnectionManager.setTimeout() を呼び出します。

加えて、MQQueueManager コンストラクターに Connection Manager が指定されなかった場合に使用するため、デフォルト接続プールとして使用する MQSimpleConnectionManager をセットアップすることもできます。

次のアプリケーションは、これを例示したものです。

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

太線の行は、MQSimpleConnectionManager オブジェクトを作成および構成します。 構成は以下を行います。

- 1 時間に渡って使用されなかった接続を破棄します。
- myConnMan によって管理される接続の数を 75 までに制限します。
- プール内の使用されていない接続を 50 までに制限します。
- MODE_AUTO を設定します。これはデフォルトです。 これを使用すると、これがデフォルト接続マネージャーであり、かつ MQEnvironment によって保持される MQPoolTokens の集合に少なくとも 1 つ以上のトークンが含まれている場合にのみ、このプールがアクティブになります。

新しい MQSimpleConnectionManager は、デフォルト接続マネージャーとして設定されます。

最後の行では、アプリケーションは MQApp3.main() を呼び出します。これにより、複数のスレッドが実行されます。各スレッドでは IBM MQ が独立して使用されます。 これらのスレッドは、接続を作成する際に myConnMan を使用します。

IBM MQ classes for Java を使用した *JTA/JDBC* の調整

IBM MQ classes for Java は、MQQueueManager.begin() メソッドをサポートしています。これにより、IBM MQ は、JDBC タイプ 2 または JDBC タイプ 4 に準拠したドライバーを提供するデータベースのコーディネーターとして機能することができます。

このサポートは、一部のプラットフォームでは使用できません。 JDBC の調整をサポートしているプラットフォームを確認するには、IBM MQ のシステム要件 を参照してください。

XA-JTA サポートを使用するには、特殊な JTA 切り替えライブラリーを使用する必要があります。 このライブラリーを使用するためのメソッドは、Windows を使用しているか、またはその他のプラットフォームのいずれかを使用しているかによって異なります。

Windows での *JTA/JDBC* 調整の構成

XA ライブラリーは、`jdbcxxx.dll` という形式の名前の DLL として提供されています。

提供されている `jdbcora12.dll` は、 IBM MQ for Windows サーバーのインストール用に、 Oracle 12C との互換性を提供します。

Windows システムでは、新しい XA ライブラリーは完全 DLL として提供されています。 この DLL の名前は `jdbcxxx.dll` です。`xxx` は、切り替えライブラリーがコンパイルされたデータベースを示します。 このライブラリーは、IBM MQ classes for Java インストール済み環境の `java\lib\jdbc` ディレクトリーまたは `java\lib64\jdbc` ディレクトリーにあります。 スイッチ・ロード・ファイルとしても説明されている XA ライブラリーをキュー・マネージャーに対して宣言する必要があります。 IBM MQ Explorer を使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。 ライブラリーの名前のみを指定する必要があります。 以下に例を示します。

Db2 データベース・セットの場合は、SwitchFile フィールドを dbcdb2 に設定します。

Oracle データベース・セットの場合は、SwitchFile フィールドを `jdbcora` に設定します。

注 :

1. Oracle 12C は、 IBM MQ for Windows でのみ、 IBM MQ classes for Java によってサポートされます。
2. Oracle 12C のサポート対象バージョンは 12.1.0.1.0 Enterprise Edition および将来のフィックスパックです。
3. 64 ビット Windows 上の Oracle 64 ビット・データベースには、32 ビット Oracle クライアントが必要です。
4. IBM MQ classes for Java を使用すると、IBM MQ はトランザクション・コーディネーターのように機能することができます。 ただし、JTA スタイルのトランザクションには参加できません。

Windows 以外のプラットフォームでの *JTA/JDBC* 調整の構成

オブジェクト・ファイルは付属しています。 付属の makefile を使用して、適切なオブジェクト・ファイルをリンクし、構成ファイルを使用してそのファイルをキュー・マネージャーに宣言します。

データベース管理システムごとに、IBM MQ は 2 つのオブジェクト・ファイルを提供します。 一方のオブジェクト・ファイルは 32 ビット切り替えライブラリーを作成する場合にリンクし、もう一方のオブジェクト・ファイルは 64 ビット切り替えライブラリーを作成する場合にリンクする必要があります。 Db2 の場合、各オブジェクト・ファイルの名前は `jdbcdb2.o` で、Oracle の場合、各オブジェクト・ファイルの名前は `jdbcora.o` です。

各オブジェクト・ファイルにリンクするには、IBM MQ で提供される適切な makefile を使用する必要があります。 切り替えライブラリーが必要とするその他のライブラリーの中には、異なるシステムの異なる場所に保管されるものもあります。 ただし、切り替えライブラリーは、ライブラリー・パス環境変数を使用してこれらのライブラリーを位置指定することはできません。切り替えライブラリーはキュー・マネージャーによってロードされますが、キュー・マネージャーは `setuid` 環境で稼働するからです。 したがって、提供された Make ファイルにより、スイッチ・ライブラリーにこれらのライブラリーの完全修飾パス名が含まれるようになります。

切り替えライブラリーを作成するには、 **make** コマンドを次の形式で入力します。 32 ビットのスイッチ・ライブラリーを作成するには、IBM MQ インストール済み環境の `/java/lib/jdbc` ディレクトリーにコマ

ンドを入力します。 64 ビットのスイッチ・ライブラリーを作成するには、/java/lib64/jdbc ディレクトリーにコマンドを入力します。

```
make DBMS
```

DBMS は、切り替えライブラリーの作成対象となるデータベース管理システムです。 有効な値は、 Db2 の場合は db2 、 Oracle の場合は oracle です。

注 :

- 32 ビット・アプリケーションを実行するには、 使用するデータベース管理システムごとに 32 ビット切り替えライブラリーと 64 ビット切り替えライブラリーの両方を作成する必要があります。 64 ビット・アプリケーションを実行する場合、 作成する必要があるのは 64 ビット切り替えライブラリーだけです。 各切り替えライブラリーの名前は、Db2 の場合は jdbcdb2、Oracle の場合は jdbcora です。 makefile を使用すると、32 ビット切り替えライブラリーと 64 ビット切り替えライブラリーが確実に異なる IBM MQ ディレクトリーに保管されます。 32 ビットのスイッチ・ライブラリーは /java/lib/jdbc ディレクトリーに保管され、64 ビットのスイッチ・ライブラリーは /java/lib64/jdbc ディレクトリーに保管されます。
- Oracle はシステム上のどこにでもインストールできるため、make ファイルでは **ORACLE_HOME** 環境変数を使用して、Oracle のインストール場所を位置指定します。
- IBM MQ がデフォルト以外の場所にインストールされている場合、make ファイルの **MQ_INSTALLATION_PATH** の値を変更します。

Db2、Oracle、または両方の切り替えライブラリーを作成したら、それらをキュー・マネージャーに宣言する必要があります。 キュー・マネージャー構成ファイル (qm.ini) に Db2 または Oracle データベースの XAResourceManager スタンザが既に含まれている場合は、各スタンザの SwitchFile エントリーを以下のいずれかに置き換える必要があります。

Db2 データベースの場合

```
SwitchFile=jdbcdb2
```

Oracle データベースの場合

```
SwitchFile=jdbcora
```

32 ビットまたは 64 ビットの切り替えライブラリーの完全修飾パス名は指定しないでください。 ライブラリーの名前のみを指定してください。

キュー・マネージャー構成ファイルに Db2 または Oracle データベース用の XAResourceManager スタンザがまだ含まれていない場合、または XAResourceManager スタンザを追加する場合は、XAResourceManager スタンザの構成方法について IBM MQ の管理 を参照してください。 ただし、新規 XAResourceManager スタンザの各 SwitchFile 項目は、Db2 データベースまたは Oracle データベースの場合について前述したとおりでなければなりません。 また、ThreadOfControl=PROCESS の項目も含めなければなりません。

キュー・マネージャー構成ファイルの更新が完了し、すべての適切なデータベース環境変数が設定されたことを確認したら、キュー・マネージャーを再始動できます。

JTA/JDBC 調整の使用
提供されている例のように API 呼び出しをコーディングします。

以下に、ユーザー・アプリケーション用の一連の基本 API 呼び出しを示します。

```
  qMgr = new MQQueueManager("QM1")
  Connection con = qMgr.getJDBCConnection( xads );
  qMgr.begin()

 < Perform MQ and DB operations to be grouped in a unit of work >
```

```
qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

getJDBCConnection 呼び出しの xads は、接続先のデータベースの詳細を定義する、データベース固有の XADataSource インターフェースの実装です。 getJDBCConnection に渡す適切な XADataSource オブジェクトの作成方法を判別するには、ご使用のデータベースに関する資料を参照してください。

JDBC の機能を実行するために、該当するデータベース固有の jar ファイルでクラスパスを更新する必要もあります。

複数のデータベースに接続する必要がある場合は、getJDBCConnection を複数回呼び出して、異なるいくつかの接続に対してトランザクションを実行する必要があります。

XADataSource.getXAConnection の 2 種類の形式を反映して、getJDBCConnection には 2 種類の形式があります。

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

これらのメソッドは、throws 文節で Exception を宣言します。それは、JTA 機能を使用しないお客様が JVM ベリファイヤーに関する問題を回避できるようにするためです。 実際には javax.transaction.xa.XAException という例外がスローされ、以前は必要なかったプログラムのクラスパスに jta.jar ファイルを追加しなければならなくなります。

JTA/JDBC サポートを使用するには、アプリケーションに以下のステートメントを組み込む必要があります。

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

JTA/JDBC の調整に関する既知の問題と制限
JTA/JDBC サポートに関する問題と制限のいくつかは、使用するデータベース管理システムによって異なります。 例えば、アプリケーションの実行中にデータベースがシャットダウンすると、テスト済みの JDBC ドライバーは異なる動作になります。 アプリケーションで使用しているデータベースへの接続が切断された場合は、キュー・マネージャーとデータベースに対する新しい接続を再確立し、 それらの新しい接続を使用して必要なトランザクション処理を実行できるようにするために、アプリケーションが実行できる手順があります。

JTA/JDBC サポートは JDBC ドライバーを呼び出すため、この JDBC ドライバーの実装は、システムの動作に重大な影響を与えることがあります。 特に、アプリケーションの稼働中にデータベースをシャットダウンしてしまうと、テスト済みの JDBC ドライバーは所定の動作をしません。

重要 : データベースへの接続をオープンしているアプリケーションが存在する間は、必ず、データベースが突然シャットダウンすることがないようにしてください。

注 : IBM MQ classes for Java アプリケーションは、 IBM MQ をデータベース・コーディネーターとして機能させるために、バインディング・モードを使用して接続する必要があります。

複数の XAResourceManager スタンザ
キュー・マネージャー構成ファイル qm.ini 内の複数の XAResourceManager スタンザの使用はサポートされていません。 最初のもの以外の XAResourceManager スタンザは無視されます。

Db2

Db2 が SQL0805N エラーを戻すことがあります。 この問題は、以下の CLP コマンドを使用して解決できます。

```
DB2 bind @db2cli.lst blocking all grant public
```

詳しくは、Db2 の資料を参照してください。

XAResourceManager スタンザは、ThreadOfControl=PROCESS を使用するように構成しなければなりません。 Db2 8.1 以降では、これは Db2 の制御設定のデフォルトのスレッドと一致しないので、toc=p を XA Open String で指定する必要があります。 JTA/JDBC と調整した Db2 用の XAResourceManager のスタンザの例は、以下の通りです。

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

これは JTA/JDBC 調整を使用した Java アプリケーションがマルチスレッド化するのを妨げるものではありません。

Oracle
MQQueueManager.disconnect() の後に JDBC の Connection.close() メソッドを呼び出すと、SQLException が生成されます。 MQQueueManager.disconnect() の前に Connection.close() を呼び出すか、または Connection.close() への呼び出しを省略してください。

データベース接続に関する問題への対処

IBM MQ classes for Java アプリケーションは、 IBM MQ によって提供される JTA/JDBC サポートを使用する場合、通常、以下のステップを実行します。

1. トランザクション・マネージャーとして動作するキュー・マネージャーへの接続を表す新しい MQQueueManager オブジェクトを作成します。
2. トランザクションに参加するデータベースへの接続方法の詳細を含む XADataSource オブジェクトを作成します。
3. 作成した XADataSource を渡して、メソッド MQQueueManager.getJDBCConnection(XADataSource) を呼び出します。 これにより、IBM MQ classes for Java はデータベースへの接続を確立します。
4. メソッド MQQueueManager.begin() を呼び出して XA トランザクションを開始します。
5. メッセージングとデータベースの処理を実行します。
6. 必要な作業すべてが完了したら、メソッド MQQueueManager.commit() を呼び出します。これで XA トランザクションが完了します。
7. この時点で新しい XA トランザクションが必要な場合、アプリケーションは手順 4、5、および 6 を繰り返すことができます。
8. アプリケーションの終了時には、手順 3 で作成したデータベース接続をクローズし、メソッド MQQueueManager.disconnect() を呼び出してキュー・マネージャーから切断する必要があります。

IBM MQ classes for Java は、アプリケーションが MQQueueManager.getJDBCConnection(XADataSource) を呼び出したときに作成されたすべてのデータベース接続の内部リストを保持します。 XA トランザクションの処理中にキュー・マネージャーがデータベースと通信する必要がある場合、以下の処理が行われます。

1. キュー・マネージャーが IBM MQ classes for Java を呼び出し、データベースに渡す必要のある XA 呼び出しの詳細を渡します。
2. すると、IBM MQ classes for Java がリストから適切な接続を検索し、その接続を使用して XA 呼び出しをデータベースに送ります。

この処理のいずれかの時点でデータベースへの接続が失われた場合、アプリケーションは以下を行う必要があります。

1. メソッド MQQueueManager.backout() を呼び出して、このトランザクションで実行された既存の処理をバックアウトします。
2. データベース接続をクローズします。 これにより、IBM MQ classes for Java は、切断されたデータベース接続の詳細をその内部リストから削除します。
3. メソッド MQQueueManager.disconnect() を呼び出して、キュー・マネージャーから切断します。

4. 新しい MQQueueManager オブジェクトを構築して、キュー・マネージャーへの新しい接続を確立します。
5. メソッド MQQueueManager.getJDBCConnection(XADataSource) を呼び出して、新しいデータベース接続を作成します。
6. トランザクション処理を再実行します。

これにより、アプリケーションは、キュー・マネージャーとデータベースに対する新しい接続を再確立し、それらの新しい接続を使用して必要なトランザクション処理を実行できるようになります。

IBM MQ classes for Java での Transport Layer Security (TLS) サポート

IBM MQ classes for Java クライアント・アプリケーションは、TLS 暗号化をサポートしています。 TLS 暗号化を使用するには、JSSE プロバイダーが必要です。

TRANSPORT(CLIENT) を使用した IBM MQ classes for Java クライアント・アプリケーションでは TLS 暗号化がサポートされます。 TLS は、通信の暗号化、認証、およびメッセージの整合性を提供します。 これは一般的に、インターネット上やイントラネット内で、任意の 2 つのピアの間の通信を保護するために使用されます。

IBM MQ classes for Java は Java Secure Socket Extension (JSSE) を使用して TLS 暗号化を処理するため、JSSE プロバイダーが必要になります。 JSE v1.4 JVM には、JSSE プロバイダーが組み込まれています。 証明書を管理して保管する方法は、プロバイダーごとに違います。 詳細については、各 JSSE プロバイダーの資料を参照してください。

この節では、JSSE プロバイダーが正常にインストールおよび構成されていることと、適切な証明書がユーザーの JSSE プロバイダーにインストールされて使用可能であることを前提としています。

IBM MQ classes for Java クライアント・アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーと接続する場合は、377 ページの『IBM MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』を参照してください。

IBM MQ classes for Java での TLS の使用可能化

TLS を有効にするには、CipherSuite を指定します。 CipherSuite を指定する方法は 2 つあります。

TLS がサポートされているのは、クライアント接続の場合のみです。 TLS を使用可能にするには、キュー・マネージャーとの通信時に使用する CipherSuite を指定する必要があります。また、この CipherSuite は、宛先チャネルに設定されている CipherSpec と一致していなければなりません。 さらに、指定した CipherSuite は、使用している JSSE プロバイダーでサポートされていなければなりません。 ただし、CipherSuites は CipherSpec とは異なるため、異なる名前が付けられています。 410 ページの『IBM MQ classes for Java での TLS CipherSpec と CipherSuite』には、IBM MQ にサポートされている CipherSpec を、JSSE に知られている同等な CipherSpec にマップする表を示します。

TLS を有効にするには、MQEnvironment の sslCipherSuite 静的メンバー変数を使用して、CipherSuite を指定します。 次の例は、TLS_RSA_WITH_AES_128_CBC_SHA256 の CipherSpec を使った TLS が必要であるため、セットアップされている SECURE.SVRCONN.CHANNEL という名前の SVRCONN チャネルに付加するものです。

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

このチャネルの CipherSpec は TLS_RSA_WITH_AES_128_CBC_SHA256 ですが、Java アプリケーションが指定する必要のある CipherSuite は SSL_RSA_WITH_AES_128_CBC_SHA256 です。 CipherSpec と CipherSuite の間のマッピングのリストは、410 ページの『IBM MQ classes for Java での TLS CipherSpec と CipherSuite』を参照してください。

アプリケーションは、環境プロパティー CMQC.SSL_CIPHER_SUITE_PROPERTY を設定して、CipherSuite を指定することもできます。

または、クライアント・チャネル定義テーブル (CCDT) を使用します。 詳しくは、377 ページの『IBM MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』 を参照してください。

クライアント接続で IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を使用する必要がある場合、 アプリケーションは MQEnvironment クラスの sslFipsRequired フィールドを `true` に設定することができます。 あるいは、アプリケーションは環境プロパティー CMQC.SSL_FIPS_REQUIRED_PROPERTY を設定することもできます。 デフォルト値は `false` であり、IBM MQ のサポートする任意の CipherSuite を クライアント接続に使用できます。

アプリケーションが複数のクライアント接続を使用する場合は、 アプリケーションが最初のクライアント接続を作成するときに使用される sslFipsRequired フィールドの値によって、後続のクライアント接続の作成時に使用される値が決まります。 このため、 アプリケーションが後続のクライアント接続を確立する時、 sslFipsRequired フィールドの値は無視されます。 sslFipsRequired フィールドに別の値を使用する場合は、アプリケーションを再始動する必要があります。

TLS を使用した接続を正常に行うには、キュー・マネージャーによって示されている証明書の認証元である、認証局のルート証明書を使用して、JSSE truststore をセットアップしなければなりません。 同様に、SVRCONN チャネルの SSLClientAuth が MQSSL_CLIENT_AUTH_REQUIRED に設定されている場合、キュー・マネージャーによって認められている識別証明書が、JSSE keystore に含まれていなければなりません。

関連資料

AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

IBM MQ classes for Java でのキュー・マネージャーの識別名の使用

キュー・マネージャーはそれ自体を TLS 証明書を使用して識別します。この TLS 証明書には識別名 (DN) が含まれています。 IBM MQ classes for Java クライアント・アプリケーションでこの DN を使用すると、確実に正しいキュー・マネージャーと通信することができます。

DN パターンを指定するときは、MQEnvironment の sslPeerName 変数を使用します。 例えば、以下のように設定すると、

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

キュー・マネージャーが QMGR で始まる共通名を持つ証明書を提示した場合にのみ、接続を成功させます。少なくとも 2 つの組織単位名があり、最初の組織単位名は IBM で、2 番目の組織単位名は WebSphere でなければなりません。

sslPeerName が設定されている場合に接続が正常に行われるのは、有効なパターンが設定されていて、キュー・マネージャーが一致する証明書を示している場合のみです。

アプリケーションは、 環境プロパティー CMQC.SSL_PEER_NAME_PROPERTY を設定することによって、キュー・マネージャーの識別名を指定することもできます。 識別名の詳細については、識別名を参照してください。

IBM MQ classes for Java での証明書取り消しリストの使用

java.security.cert.CertStore クラスを通して使用する証明書取り消しリスト (CRL) を指定します。 こうすると、IBM MQ classes for Java は、指定された CRL に照らして証明書を検査します。

証明書取り消しリスト (CRL) は、発行元の認証局かローカル組織のいずれかによって取り消された証明書のセットです。 多くの場合 CRL は、LDAP サーバーにホストされています。 Java 2 v1.4 では、CRL サーバーを接続時に指定でき、接続が許可される前に、キュー・マネージャーによって示されている証明書が、CRL と比較して検査されます。 証明書取り消しリストおよび IBM MQ について詳しくは、証明書取り消しリストおよび権限取り消しリストの取り扱いと IBM MQ classes for Java および IBM MQ classes for JMS を使用した CRL および ARL へのアクセスを参照してください。

注 : CertStore を LDAP サーバーでホストされている CRL と共に正常に使用するには、ご使用の Java Software Development Kit (SDK) が CRL に適合することを確認してください。 SDK によっては、CRL が LDAP v2 用のスキーマを定義する RFC 2587 に準拠している必要があります。 ほとんどの LDAP v3 サーバーは、代わりに RFC 2256 を使用しています。

使用する CRL は、java.security.cert.CertStore クラスによって指定されます。 CertStore のインスタンスの取得方法については、このクラスの資料を参照してください。 LDAP サーバーに基づいて CertStore を作成するには、まず LDAPCertStoreParameters インスタンス (使用するサーバーおよびポート設定によって初期設定される) を作成します。 以下に例を示します。

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

CertStoreParameters インスタンスを作成してから、CertStore で静的コンストラクターを使用し、タイプ LDAP の CertStore を作成します。

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

他の CertStore タイプ (Collection など) もサポートされています。 通常は、同じ CRL 情報で複数の CRL サーバーをセットアップすることにより、冗長性が提供されます。 CRL サーバーのそれぞれに CertStore オブジェクトを用意したら、適切な Collection 内にそれらを配置してください。 次の例では、ArrayList に置かれている CertStore オブジェクトが示されています。

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

この Collection は、接続して CRL 検査を使用可能にする前に、以下のように MQEnvironment 静的変数 sslCertStores に設定できます。

```
MQEnvironment.sslCertStores = crls;
```

接続を設定するときにキュー・マネージャーによって示される証明書は、以下のようにして検査されます。

1. sslCertStores によって識別される Collection にある最初の CertStore オブジェクトを使用し、CRL サーバーを識別します。
2. CRL サーバーへ接続してみます。
3. 接続が成功すれば、一致する証明書をサーバー内で検索します。
   a. 証明書が失効したことが分かった場合、検索プロセスは終了して接続要求は失敗します。理由コードは MQRC_SSL_CERTIFICATE_REVOKED です。
   b. 証明書が見つからない場合、検索プロセスは終了し、接続を先に進めることができます。
4. サーバーへの接続が失敗する場合、次の CertStore オブジェクトを使用して、CRL サーバーを識別します。プロセスはステップ 2 から繰り返されます。

   コレクションの最後の CertStore であった場合、 またはコレクションに CertStore オブジェクトが 1 つも入っていない場合、検索プロセスは失敗し、接続要求は理由コード MQRC_SSL_CERT_STORE_ERROR で失敗します。

Collection オブジェクトは、CertStores を使用する順序を決定します。

CMQC.SSL_CERT_STORE_PROPERTY を使用して、CertStore の Collection を設定することもできます。 便宜上、このプロパティーに単一の CertStore を (コレクションのメンバーにせずに) 指定することも可能です。

sslCertStores をヌルに設定した場合、CRL 検査は実行されません。 sslCipherSuite を設定しなければ、このプロパティーは無視されます。

IBM MQ classes for Java での秘密鍵の再ネゴシエーション

IBM MQ classes for Java クライアント・アプリケーションは、クライアント接続で暗号化に使用される秘密鍵を再ネゴシエーションするタイミングを、送受信される合計バイト数単位で制御できます。

アプリケーションは、これを以下のいずれかの方法で実行できます。アプリケーションでこれらの方法を複数使用する場合、通常の優先順位ルールが適用されます。

- MQEnvironment クラスの sslResetCount フィールドを設定する方法。
- Hashtable オブジェクトの環境プロパティー MQC.SSL_RESET_COUNT_PROPERTY を設定する。 この方法では、アプリケーションによって、MQEnvironment クラスの `properties` フィールドにハッシュ・テ

ーブルが割り当てられるか、またはそのコンストラクターで MQQueueManager オブジェクトにハッシュ・テーブルが受け渡されます。

sslResetCount フィールドまたは環境プロパティー MQC.SSL_RESET_COUNT_PROPERTY の値は、秘密鍵が再ネゴシエーションされる前に IBM MQ classes for Java クライアント・コードが送受信するバイトの総数を表します。 送信バイト数は暗号化前の数であり、受信バイト数は暗号化解除された後の数です。 バイト数には、IBM MQ classes for Java クライアントによって送受信される制御情報も含まれています。

リセット・カウントがゼロ (デフォルト値) の場合、秘密鍵は再ネゴシエーションされません。 CipherSuite が指定されていない場合、リセット・カウントは無視されます。

IBM MQ classes for Java でのカスタマイズした *SSLSocketFactory* の提供
カスタマイズした JSSE ソケット・ファクトリーを使用する場合、MQEnvironment.sslSocketFactory を、そのカスタマイズしたファクトリー・オブジェクトに設定します。 詳細は、JSSE 実装によって異なります。

別々の JSSE 実装に、それぞれ別個のフィーチャーを提供できます。 例えば、特殊化した JSSE 実装で、暗号化ハードウェアの特定のモデルの構成が可能です。 さらに、いくつかの JSSE プロバイダーで、プログラムによって keystores および truststores をカスタマイズしたり、keystore からの識別証明書の選択を変更したりすることもできます。 JSSE では、それらのすべてのカスタマイズが、ファクトリー・クラス javax.net.ssl.SSLSocketFactory に抽象化されます。

カスタマイズした SSLSocketFactory 実装を作成する方法については、JSSE の資料を参照してください。 それらの詳細はプロバイダーごとに異なりますが、典型的な一連の手順を以下に示します。

1. SSLContext で静的メソッドを使用して、SSLContext オブジェクトを作成します。
2. 適切な KeyManager および TrustManager 実装 (それら自身のファクトリー・クラスから作成される) を使用して、この SSLContext を初期設定します。
3. SSLContext から SSLSocketFactory を作成します。

SSLSocketFactory オブジェクトがある場合、MQEnvironment.sslSocketFactory を、そのカスタマイズしたファクトリー・オブジェクトに設定します。 以下に例を示します。

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java はこの SSLSocketFactory を使用して、IBM MQ キュー・マネージャーに接続します。 このプロパティーは、CMQC.SSL_SOCKET_FACTORY_PROPERTY を使用して設定することもできます。 sslSocketFactory がヌルに設定されている場合、JVM のデフォルト SSLSocketFactory が使用されます。 sslCipherSuite を設定しなければ、このプロパティーは無視されます。

カスタムの SSLSocketFactories を使用する場合には、TCP/IP 接続を共有することによる影響を考慮してください。 接続の共用が可能である場合は、生成されたソケットが後続の接続要求のコンテキストにおいて何らかの形で異なっている場合でも、用意された SSLSocketFactory に新規ソケットは要求されません。 例えば、後続の接続で別のクライアント証明書が提示される場合は、接続の共用を許可するべきではありません。

IBM MQ classes for Java での *JSSE* 鍵ストアまたはトラストストアの変更
JSSE 鍵ストアまたはトラストストアを変更する場合、変更を有効にするためにいくつかのアクションを実行する必要があります。

JSSE 鍵ストアや JSSE トラストストアの内容を変更したり、鍵ストア・ファイルやトラストストア・ファイルの位置を変更したりした場合、その時点で実行中の IBM MQ classes for Java アプリケーションはその変更を自動的には認識しません。 変更を有効にするには、以下のアクションを行う必要があります。

- アプリケーションは、すべての接続をクローズし、接続プール内の未使用の接続を破棄する必要がある。
- JSSE プロバイダーが鍵ストアおよびトラストストアからの情報をキャッシュしている場合は、この情報をリフレッシュする必要がある。

これらのアクションが完了すると、アプリケーションは接続を再作成できます。

アプリケーションの設計方法や、JSSE プロバイダーが提供する機能によっては、アプリケーションを停止および再始動しなくても上記のアクションを実行できる場合があります。 ただし、アプリケーションを停止して再始動するのが最も簡単な解決方法です。

IBM MQ classes for Java で TLS を使用する場合のエラー処理
TLS を使用してキュー・マネージャーに接続する場合、いくつかの理由コードが IBM MQ classes for Java によって発行される場合があります。

これらは、以下のリストで説明します。

MQRC_SSL_NOT_ALLOWED
sslCipherSuite プロパティーが設定されましたが、バインディング接続が使用されました。 TLS をサポートしているのは、クライアント接続のみです。

MQRC_JSSE_ERROR
IBM MQ が処理できないエラーを、JSSE プロバイダーが報告しました。 この原因として、JSSE での構成の問題か、またはキュー・マネージャーによって示された証明書が、妥当性検査できなかったことが考えられます。 JSSE によって作成された例外は、MQException で getCause() メソッドを使用して取り出すことができます。

MQRC_SSL_INITIALIZATION_ERROR
TLS 構成オプションが指定された MQCONN または MQCONNX 呼び出しが発行されましたが、TLS 環境の初期化中にエラーが発生しました。

MQRC_SSL_PEER_NAME_MISMATCH
sslPeerName プロパティーで指定した DN パターンが、キュー・マネージャーによって示された DN と一致しませんでした。

MQRC_SSL_PEER_NAME_ERROR
sslPeerName プロパティーで指定した DN パターンが無効でした。

MQRC_UNSUPPORTED_CIPHER_SUITE
sslCipherSuite で指定された CipherSuite が、JSSE プロバイダーによって認識されませんでした。 JSSE プロバイダーがサポートしている CipherSuites の完全なリストは、SSLSocketFactory.getSupportedCipherSuites() メソッドを使用して、プログラムにより取得できます。 IBM MQ との通信に使用できる CipherSuite のリストについては、410 ページの『IBM MQ classes for Java での TLS CipherSpec と CipherSuite』を参照してください。

MQRC_SSL_CERTIFICATE_REVOKED
キュー・マネージャーによって示された証明書が、sslCertStores プロパティーで指定された CRL にありました。 信頼されている証明書を使用するために、キュー・マネージャーを更新してください。

MQRC_SSL_CERT_STORE_ERROR
キュー・マネージャーで示された証明書を検索したものの、指定された CertStore がありませんでした。 MQException.getCause() メソッドが、試行された最初の CertStore の検索の間に発生したエラーを戻しました。 原因である例外が NoSuchElementException、ClassCastException、または NullPointerException である場合、sslCertStores プロパティーで指定されている Collection に、有効な CertStore オブジェクトが 1 つ以上含まれているか検査してください。

IBM MQ classes for Java での TLS CipherSpec と CipherSuite
IBM MQ classes for Java アプリケーションがキュー・マネージャーへの接続を確立できるかどうかは、MQI チャネルのサーバー側で指定された CipherSpec およびクライアント側で指定された CipherSuite によって決まります。

以下の表に、IBM MQ によってサポートされる CipherSpec と、それらに相当する CipherSuite のリストを示します。

Deprecated トピック『推奨されない CipherSpec』を参照して、次の表にリストしている CipherSpec が IBM MQ で非推奨になっていないか、また、非推奨になっている場合はどの更新プログラムで CipherSpec が非推奨になったかを確認してください。

重要 : リストされている CipherSuites は、IBM MQ で提供される IBM Java ランタイム環境 (JRE) によってサポートされるものです。 リストされている CipherSuites には、 Oracle Java JRE によってサポートされている CipherSuites が含まれています。 Oracle Java JRE を使用するためのアプリケーションの構成方法

について詳しくは、430 ページの『IBM Java または Oracle Java CipherSuite マッピングを使用するためのアプリケーションの構成』を参照してください。

この表に、通信に使用されるプロトコルと、CipherSuite が FIPS 140-2 標準に準拠するかどうかも示します。

注 : AIX, Linux, and Windows では、 IBM MQ は IBM Crypto for C (ICC) 暗号モジュールを介して FIPS 140-2 準拠を提供します。 このモジュールの証明書は「履歴」ステータスに移動されました。 お客様は、IBM Crypto for C (ICC) 証明書 を表示し、NIST から提供されたアドバイスに注意する必要があります。 交換用の FIPS 140-3 モジュールが現在進行中であり、その状況を表示するには、 「NIST CMVP modules in process list」でそのモジュールを検索します。

FIPS 140-2 準拠を強制するようにアプリケーションが構成されていない場合に FIPS 140-2 準拠として示されている CipherSuite を使用することはできますが、FIPS 140-2 準拠となるようにアプリケーションが構成されている場合 (下記の構成注記を参照)、FIPS 140-2 互換のマークが付いている CipherSuite しか構成できません。その他の CipherSuite を使用しようとするとエラーになります。

注 : 各 JRE は複数の暗号化セキュリティー・プロバイダーを持つことができ、それぞれが同じ CipherSuite の実装を提供することができます。 ただし、すべてのセキュリティー・プロバイダーが FIPS 140-2 認定されている訳ではありません。 アプリケーションで FIPS 140-2 準拠が強制されていない場合には、非認定の CipherSuite の実装環境を使用できる場合があります。 非認定の実装環境では、FIPS 140-2 で求められる最小セキュリティー・レベルを CipherSuite が理論的には満たしている場合であっても、その標準に準拠した方法で作動しない場合があります。 IBM MQ Java アプリケーションにおいて FIPS 140-2 を強制する場合の構成について詳しくは、以下の注記を参照してください。

CipherSpec と CipherSuite の FIPS 140-2 準拠と Suite-B 準拠について詳しくは、CipherSpec の指定を参照してください。 また、米国の連邦情報処理標準に関わる情報に留意しなければならない場合もあります。

CipherSuite の全セットを使用し、FIPS 140-2 準拠または Suite-B 準拠 (あるいはその両方に準拠) として認定されたものを使用してそれを作動させるには、適切な JRE が必要です。IBM Java 7 Service Refresh 4 フィックスパック 2 以上のレベルの IBM JRE は、 412 ページの表 59 にリストされている TLS 1.2 CipherSuites の適切なサポートを提供します。

TLS 1.3 Ciphers を使用できるようにするには、アプリケーションを実行する JRE が TLS 1.3 をサポートしている必要があります。

注 : 一部の CipherSuite を使用する場合に、「無制限」ポリシー・ファイルを JRE で構成する必要があります。 SDK または JRE でポリシー・ファイルをセットアップする方法について詳しくは、ご使用のバージョンの「 *Security Reference for IBM SDK, Java Technology Edition* 」の「 *IBM SDK* ポリシー・ファイル」トピックを参照してください。

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite*

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	いいえ

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	いいえ
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	yes

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	yes
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	いいえ

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	いいえ
TLS_RSA_WITH_3DES_EDE_CBC_SHA 430 ページの『2』	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	いいえ 430 ページの『4』

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	いいえ 430 ページの『4』
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	いいえ 430 ページの『4』

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	いいえ 430 ページの『4』
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	いいえ 430 ページの『4』

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	いいえ 430 ページの『4』
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	いいえ 430 ページの『4』

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	いいえ
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	いいえ
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	いいえ

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	yes
TLS_AES_128_GCM_SHA256 430 ページの『3』	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS V1.3	いいえ
TLS_AES_256_GCM_SHA384 430 ページの『3』	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS V1.3	いいえ

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_CHACHA20_POLY1305_SHA256 430 ページの『3』	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS V1.3	いいえ
TLS_AES_128_CCM_SHA256 430 ページの『3』	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS V1.3	いいえ

表 59. *IBM MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec 430 ページの『1』	**同等の CipherSuite (IBM JRE)**	**同等の CipherSuite (Oracle JRE)**	**プロトコル**	**FIPS 140-2 互換**
TLS_AES_128_CCM_8_SHA256 430 ページの『3』	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS V1.3	いいえ
任意 430 ページの『3』	*ANY	*ANY	複数	いいえ
ANY_TLS13 430 ページの『3』	*TLS13	*TLS13	TLS V13	いいえ
ANY_TLS12_OR_HIGHER 430 ページの『3』	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 以上	いいえ
ANY_TLS13_OR_HIGHER 430 ページの『3』	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 以上	いいえ

注:

1. これは、CCDT (バイナリーまたは JSON) を含め、 IBM MQ のチャネルで構成された値です。
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。
3. TLS v1.3 暗号を使用できるようにするには、アプリケーションを実行する Java runtime environment (JRE) が TLS v1.3 をサポートしている必要があります。
4. V 9.3.0.17 V 9.3.5.1 IBM MQ 9.3.5 CSU 1 および IBM MQ 9.3.0 CSU 17 以降、 IBM Java 8 JRE は、FIPS モードでの操作時に RSA 鍵交換のサポートを除去します。

IBM MQ classes for Java アプリケーションでの CipherSuite と FIPS 準拠の構成

- IBM MQ classes for Java を使用するアプリケーションは、接続に CipherSuite を設定するときに次の 2 つの方式のいずれかを使用できます。
  - MQEnvironment クラスの sslCipherSuite フィールドに CipherSuite 名を設定する。
  - MQQueueManager コンストラクターに渡されるプロパティー・ハッシュ・テーブルのプロパティー CMQC.SSL_CIPHER_SUITE_PROPERTY に CipherSuite 名を設定する。
- IBM MQ classes for Java を使用するアプリケーションは、FIPS 140-2 準拠を強制するために、次の 2 つの方式のいずれかを使用できます。
  - MQEnvironment クラスの sslFipsRequired フィールドに true を設定する。
  - MQQueueManager コンストラクターに渡されるプロパティー・ハッシュ・テーブルのプロパティー CMQC.SSL_FIPS_REQUIRED_PROPERTY に true を設定する。

IBM Java または Oracle Java CipherSuite マッピングを使用するためのアプリケーションの構成

注 :

V 9.3.3 IBM MQ 9.3.3 以降の Continuous Delivery では、どのマッピングを使用するかを制御する Java「システム・プロパティー」 `com.ibm.mq.cfg.useIBMCipherMappings` が製品から削除されています。 IBM MQ 9.3.3 以降、Cipher は CipherSpec または CipherSuite のいずれかの名前として定義でき、IBM MQ によって正しく処理されます。 IBM MQ classes for Java アプリケーションがキュー・マネージャーへのセキュア TLS 接続を作成する場合は、以下の 3 つの Jackson JAR ファイルが必要です。

- `jackson-annotations.jar`
- `jackson-core.jar`
- `jackson-databind.jar`

重要 : `com.ibm.mq.cfg.useIBMCipherMappings` に関する以下の情報は、 IBM MQ 9.3.3 より前の Long Term Support および Continuous Delivery にのみ適用されます。

アプリケーションがデフォルトの IBM Java CipherSuite を IBM MQ CipherSpec マッピングに使用するか、Oracle CipherSuite を IBM MQ CipherSpec マッピングに使用するかを構成できます。 そのため、IBM JRE と Oracle JRE のどちらをアプリケーションで使用するかに関係なく、TLS CipherSuites を使用できます。 Java「システム・プロパティー」 `com.ibm.mq.cfg.useIBMCipherMappings` は、どのマッピングを使用するかを制御します。 プロパティーは、次の値のうちのいずれかです。

true
IBM Java CipherSuite を IBM MQ CipherSpec マッピングに使用します。

この値がデフォルト値です。

false
Oracle CipherSuite を IBM MQ CipherSpec マッピングに使用します。

IBM MQ Java および TLS 暗号の使用について詳しくは、MQdev ブログ投稿「MQ Java, TLS Ciphers, Non-IBM JREs & APAR IT06775, IV66840, IT09423, IT10837」を参照してください。

インターオペラビリティーの制約

いくつかの CipherSuite は、使用するプロトコルによっては、複数の IBM MQ CipherSpec と適合します。 ただし、サポートされるのは表 1 で指定された TLS バージョンを使用する CipherSuite/CipherSpec の組み合わせのみです。 サポートされない CipherSuite と CipherSpec の組み合わせを使用しようとすると失敗し、該当する例外が発生します。 これらの CipherSuite/CipherSpec のいずれかの組み合わせを使用するインストールをサポートされる組み合わせに移行する必要があります。

以下の表に、この制約が適用される CipherSuite を示します。

表 *60. CipherSuite* と、それでサポートされる *CipherSpec* とサポートされない *CipherSpec*		
CipherSuite	**サポートされる TLS CipherSpec**	**サポートされない SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA 431 ページの『1』	TRIPLE_DES_SHA_US
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

注 :

1. Deprecated この CipherSpec の TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

IBM MQ classes for Java アプリケーションの実行

クライアント・モードまたはバインディング・モードのいずれかを使用してアプリケーション (main() メソッドが含まれているクラス) を作成する場合は、Java インタープリターでプログラムを実行してください。

次のコマンドを使用します。

```
java -Djava.library.path= library_path MyClass
```

ここで、*library_path* は、IBM MQ classes for Java ライブラリーへのパスです。 詳しくは、358 ページの『IBM MQ classes for Java ライブラリー』 を参照してください。

関連タスク

IBM MQ classes for Java アプリケーションのトレース

IBM MQ リソース・アダプターのトレース

環境による IBM MQ classes for Java の振る舞いの違い

IBM MQ classes for Java により、異なるバージョンの IBM MQ に対して実行できるアプリケーションを作成できます。 このトピックのコレクションでは、バージョンが異なるときに違ってくる Java クラスの振る舞いについて説明します。

IBM MQ classes for Java は、すべての環境で一貫した機能と振る舞いを提供する、クラスのコアを提供します。 このコアの外部の機能は、アプリケーションの接続先のキュー・マネージャーの機能によって異なります。

ここに記載するもの以外は、各キュー・マネージャー用の MQI アプリケーション・リファレンスで説明されているとおりの振る舞いになります。

IBM MQ classes for Java 内のコア・クラス

IBM MQ classes for Java には、クラスのコア・セットが含まれます。これは、すべての環境で使用できます。

以下に示すクラスのコア・セットはコア・クラスと見なされ、433 ページの『IBM MQ classes for Java のコア・クラスの制限とバリエーション』にリストされているわずかなバリエーションだけで、すべての環境で使用できます。

- MQEnvironment
- MQException
- MQGetMessageOptions

  以下は除外します。

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  以下は除外します。

  - inquire()
  - set()
- MQMessage

  以下は除外します。

  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  以下は除外します。

  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  以下は除外します。

  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

注 :

1. 一部の定数はコアには含まれていないため (詳細については、433 ページの『IBM MQ classes for Java のコア・クラスの制限とバリエーション』を参照)、完全に移植可能なプログラム内ではそれらの定数を使用しないでください。
2. プラットフォームによっては、一部の接続モードがサポートされていない場合があります。 このようなプラットフォームでは、サポートされているモードに関連するコア・クラスとオプションのみが使用できます。

IBM MQ classes for Java のコア・クラスの制限とバリエーション

通常、同等の MQI 呼び出しに環境の違いがある場合でも、コア・クラスはすべての環境で一貫した振る舞いをします。 その振る舞いは、以下の小さな制限とバリエーションを例外として、AIX、Linux、または Windows キュー・マネージャーが使用される場合と同様になります。

IBM MQ classes for Java での *MQGMO_** 値の制限

特定の MQGMO_* 値は、すべてのキュー・マネージャーでサポートされているわけではありません。

以下の MQGMO_* 値を使用すると、MQQueue.get() から MQException がスローされる可能性があります。

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

さらに、MQGMO_SET_SIGNAL は、Java から使用する場合にはサポートされません。

IBM MQ classes for Java での *MQPMRF_** 値の制限

これらは、メッセージを配布リストに書き込むときにのみ使用されるため、配布リストをサポートするキュー・マネージャーのみにサポートされます。 例えば、z/OS キュー・マネージャーは配布リストをサポートしません。

IBM MQ classes for Java での *MQPMO_** 値の制限

一部の MQPMO_* 値は、すべてのキュー・マネージャーでサポートされているわけではありません。

以下の MQPMO_* 値を使用すると、MQQueue.put() または MQQueueManager.put() から MQException がスローされる可能性があります。

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

IBM MQ classes for Java での *MQCNO_** 値の制限とバリエーション

特定の MQCNO_* 値はサポートされていません。

- 自動クライアント再接続は、 IBM MQ classes for Java ではサポートされていません。 MQCNO_RECONNECT_* をどのような値に設定しても、接続時の動作は MQCNO_RECONNECT_DISABLED と設定された場合と同じになります。
- MQCNO_FASTPATH は、MQCNO_FASTPATH をサポートしていないキュー・マネージャーでは無視されます。 クライアント接続でも、同様に無視されます。

IBM MQ classes for Java での *MQRO_** 値の制限
以下のレポート・オプションを設定できます。

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

詳しくは、レポートを参照してください。

z/OS 上の *IBM MQ classes for Java* とその他のプラットフォームとのその他の相違点
IBM MQ for z/OS は、いくつかの面で、他のプラットフォーム上の IBM MQ とは異なる動作をします。

BackoutCount
z/OS キュー・マネージャーでは、メッセージが 255 回を超えてバックアウトされていても、最大 BackoutCount 255 が戻されます。

デフォルト動的キュー接頭部
バインディング接続を使用して z/OS キュー・マネージャーに接続した場合、デフォルト動的キュー接頭部は CSQ.* です。 そうではない場合、デフォルト動的キュー接頭部は AMQ.* です。

MQQueueManager コンストラクター
クライアント接続は、z/OS ではサポートされていません。 クライアント・オプションを指定して接続しようとすると、MQException が出されて、MQCC_FAILED および MQRC_ENVIRONMENT_ERROR になります。

MQQueueManager コンストラクターが失敗した場合は、MQRC_CHAR_CONVERSION_ERROR (IBM-1047 コード・ページと ISO8859-1 コード・ページ間の変換の初期設定に失敗したとき)、または MQRC_UCS2_CONVERSION_ERROR (キュー・マネージャーのコード・ページとユニコード間の変換の初期設定に失敗したとき) が出される可能性もあります。 使用しているアプリケーションが、これらの理由コードのいずれかで失敗する場合、Language Environment の National Language Resources コンポーネントがインストールされていることと、正確な変換テーブルが使用可能であることを確認してください。

ユニコードの変換テーブルは、z/OS C/C++ オプショナル機能の一部としてインストールされます。 UCS-2 変換の使用可能化についての詳細は、「*z/OS C/C++* プログラミング・ガイド」(SC88-8849) を参照してください。

IBM MQ classes for Java のコア・クラス外の機能

IBM MQ classes for Java には、すべてのキュー・マネージャーでサポートされていない API 拡張機能を使用するために特別に設計された機能が含まれています。 このトピックのコレクションでは、それらの機能をサポートしないキュー・マネージャーを使用するときの、それらの機能の動作方法について説明します。

MQQueueManager コンストラクター・オプションでのバリエーション
MQQueueManager コンストラクターの中には、オプションの整数引数が含まれているものもあります。 この引数の値の中には、すべてのプラットフォームでは受け入れられないものがあります。

MQQueueManager コンストラクターにオプションの整数引数が含まれる場合、それは MQI の MQCNO オプション・フィールドにマップされ、標準接続とファスト・パス接続を切り替えるために使用されます。 コンストラクターのこの拡張形式は、使用されたオプションが MQCNO_STANDARD_BINDING または MQCNO_FASTPATH_BINDING の場合のみ、すべての環境で受け入れられます。 その他のオプションはすべて、コンストラクターが MQRC_OPTIONS_ERROR によって失敗する原因になります。 ファースト・パス・オプション CMQC.MQCNO_FASTPATH_BINDING は、このオプションをサポートするキュー・マネージャーへのバインディング接続で使用する場合にのみ受け付けられます。 他の環境では、無視されます。

MQQueueManager.begin() メソッドでの制限
このメソッドは、バインディング・モードの AIX, Linux, and Windows システム上の IBM MQ キュー・マネージャーに対してのみ使用できます。 それ以外の場合は、失敗して MQRC_ENVIRONMENT_ERROR が出されます。

詳しくは、402 ページの『IBM MQ classes for Java を使用した JTA/JDBC の調整』を参照してください。

MQGetMessageOptions フィールドでのバリエーション
キュー・マネージャーによっては、バージョン 2 MQGMO 構造体をサポートしないものがあるため、いくつかのフィールドはデフォルト値に設定しなければなりません。

バージョン 2 MQGMO 構造体をサポートしないキュー・マネージャーを使用する場合、以下のフィールドはデフォルト値に設定したままにしておかなければなりません。

GroupStatus
SegmentStatus
Segmentation

また、MatchOptions フィールドは、MQMO_MATCH_MSG_ID および MQMO_MATCH_CORREL_ID のみをサポートします。 サポートされない値をこれらのフィールドに指定すると、それ以降の MQDestination.get() は失敗し、MQRC_GMO_ERROR が出されます。 キュー・マネージャーがバージョン 2 MQGMO 構造をサポートしない場合、これらのフィールドは正常な MQDestination.get() の後に更新されません。

IBM MQ classes for Java での配布リストにおける制限
すべてのキュー・マネージャーで MQDistributionList をオープンできるわけではありません。

以下のクラスは、配布リストを作成するために使用されます。

MQDistributionList
MQDistributionListItem
MQMessageTracker

どのような環境でも MQDistributionLists および MQDistributionListItems を作成して取り込めますが、すべてのキュー・マネージャーで MQDistributionList をオープンできるわけではありません。 特に、z/OS キュー・マネージャーは配布リストをサポートしません。 そのようなキュー・マネージャーを使用して MQDistributionList をオープンしようとすると、MQRC_OD_ERROR が出されます。

MQPutMessageOptions フィールドでのバリエーション
キュー・マネージャーが配布リストをサポートしない場合、特定の MQPMO フィールドの扱いが異なります。

MQPMO の 4 つのフィールドは、MQPutMessageOptions クラスの次の 4 つのメンバー変数として提供されます。

knownDestCount
unknownDestCount
invalidDestCount
recordFields

これらのフィールドは、主に配布リストでの使用を目的にしたものです。 しかし、配布リストをサポートするキュー・マネージャーは、単一キューへの MQPUT 後に DestCount フィールドも指定します。 例えば、キューがローカル・キューに解決されると、knownDestCount は 1 に設定され、その他の 2 つのカウント・フィールドは 0 に設定されます。

キュー・マネージャーが配布リストをサポートしない場合、これらの値は以下のようにシミュレートされます。

- put() が成功した場合には、unknownDestCount が 1 に設定され、その他は 0 に設定されます。
- put() が失敗した場合には、invalidDestCount が 1 に設定され、その他は 0 に設定されます。

recordFields 変数は配布リストで使用されます。 値は、環境とは関係なく、いつでも recordFields に書き込むことができます。 MQPutMessageOptions オブジェクトが、MQDistributionList.put() ではなく、後続の MQDestination.put() または MQQueueManager.put() で使用される場合、その値は無視されます。

IBM MQ classes for Java での *MQMD* フィールドにおける制限
セグメンテーションをサポートしないキュー・マネージャーを使用する場合には、メッセージのセグメンテーションに関連した特定の MQMD フィールドはデフォルト値のままにしておく必要があります。

以下の MQMD フィールドは、メッセージのセグメント化と大きく関係しています。

GroupId
MsgSeqNumber
オフセット
MsgFlags
OriginalLength

アプリケーションでこれらの MQMD フィールドのいずれかをデフォルト以外の値に設定した後、これらの値をサポートしないキュー・マネージャーに put() または get() を出す場合、put() または get() によって MQException が出され、MQRC_MD_ERROR になります。 そのようなキュー・マネージャーでの正常な put() または get() では、常に、MQMD フィールドがそのデフォルト値に設定されたままとなります。 グループ化されたメッセージまたはセグメント化されたメッセージは、メッセージのグループ化およびセグメント化をサポートしないキュー・マネージャーに対して実行中の Java アプリケーションには送信しないでください。

Java アプリケーションが、これらのフィールドをサポートしないキュー・マネージャーからメッセージを get() しようとする場合で、取得する物理メッセージがセグメント化されたメッセージのグループの一部である場合 (つまり、MQMD フィールドにデフォルト値以外がある場合)、そのメッセージはエラーなしで取得されます。 しかし、MQMessage の MQMD フィールドは更新されず、MQMessage の format プロパティーは MQFMT_MD_EXTENSION に設定され、実際のメッセージ・データには、新規フィールドの値が入っている MQMDE 構造体によって接頭部が付けられます。

CICS Transaction Server での IBM MQ classes for Java の制約事項

CICS Transaction Server for z/OS 環境では、メイン (最初) のスレッドしか CICS または IBM MQ 呼び出しを発行できません。

IBM MQ JMS クラスを CICS Java アプリケーション内で使用することはサポートされていないことに注意してください。

これは、この環境では、スレッド間で MQQueueManager オブジェクトや MQQueue オブジェクトを共用したり、子スレッドに新しい MQQueueManager を作成したりできないからです。

z/OS 434 ページの『z/OS 上の IBM MQ classes for Java とその他のプラットフォームとのその他の相違点』 は、 z/OS キュー・マネージャーに対して実行する場合に IBM MQ classes for Java に適用されるいくつかの制約事項とバリエーションを示しています。 さらに、CICS のもとで実行する際には、MQQueueManager に対するトランザクション制御メソッドはサポートされません。 アプリケーションは、MQQueueManager.commit() または MQQueueManager.backout() を発行する代わりに、J CICS タスク同期メソッド、task.commit()、および Task.rollback() を使用します。タスク・クラスは、com.ibm.cics.server パッケージ内の J CICS によって提供されます。

IBM MQ リソース・アダプターの使用

リソース・アダプターによって、アプリケーション・サーバーで実行されているアプリケーションは IBM MQ リソースにアクセスできます。 インバウンド通信とアウトバウンド通信をサポートします。

リソース・アダプターの内容

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、新規アプリケーションの開発で Jakarta Messaging 3.0 がサポートされるようになりました。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 Java EE および JMS 2.0 をサポートするリソース・アダプターに加えて、 IBM MQ 9.3.0 は Jakarta Messaging をサポートするリソース・アダプターを提供します。

JM 3.0 **Jakarta Messaging 用の IBM MQ リソース・アダプター**

Jakarta Connectors Architecture は、Jakarta EE 環境で実行されているアプリケーションを IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続するための標準的な方法を提供します。 Jakarta Messaging 用の IBM MQ リソース・アダプターは、 Jakarta Connectors 2.0.0 インターフェースを実装し、 IBM MQ classes for Jakarta Messaging を含んでいます。 これにより、アプリケーション・サーバーで実行されている Jakarta Messaging アプリケーションおよびメッセージ駆動型 Bean (MDB) が IBM MQ キュー・マネージャーのリソースにアクセスできるようになります。 リソース・ア

ダプターは Point-to-Point ドメインとパブリッシュ/サブスクライブ・ドメインの両方をサポートします。

JMS 2.0 **JMS 2.0 用の IBM MQ リソース・アダプター**
Java Platform, Enterprise Edition Connector Architecture (JCA) は、Java EE 環境内で実行されているアプリケーションを、IBM MQ や Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。 JMS 2.0 用の IBM MQ リソース・アダプターは、 JCA 1.7 インターフェースを実装し、 IBM MQ classes for JMS を含んでいます。 このリソース・アダプターは、アプリケーション・サーバーで実行される JMS アプリケーションと Message Driven Beans (MDB) が、IBM MQ キュー・マネージャーのリソースにアクセスできるようにするためのものです。 リソース・アダプターは Point-to-Point ドメインとパブリッシュ/サブスクライブ・ドメインの両方をサポートします。

IBM MQ リソース・アダプターは、アプリケーションとキュー・マネージャーの間の 2 つのタイプの通信をサポートします。

アウトバウンド通信
アプリケーションはキュー・マネージャーへの接続を開始し、同期的に JMS メッセージを JMS 宛先に送信し、JMS 宛先から JMS メッセージを受信します。

インバウンド通信
JMS 宛先に届く JMS メッセージは MDB に送信され、MDB はメッセージを非同期的に処理します。

リソース・アダプターには、IBM MQ classes for Java も含まれています。 クラスは、リソース・アダプターがデプロイされているアプリケーション・サーバーで実行されているアプリケーションで自動的に使用可能になり、そのアプリケーション・サーバーで実行されているアプリケーションが IBM MQ キュー・マネージャーのリソースにアクセスするときに IBM MQ classes for Java API を使用できるようになります。

Java EE 環境内での IBM MQ classes for Java の使用は、制限付きでサポートされます。 これらの制限については、349 ページの『Java EE 内での IBM MQ classes for Java アプリケーションの実行』を参照してください。

使用するリソース・アダプターのバージョン

V 9.3.0 V 9.3.0 使用するリソース・アダプターのバージョンは、 Jakarta EE または Java EE をサポートするアプリケーション・サーバーにデプロイするかどうかによって異なります。

JM 3.0 V 9.3.0 V 9.3.0 **Jakarta EE**
IBM MQ 9.3.0 以降では、 Jakarta Messaging 3.0 がサポートされます。 Jakarta Messaging 用の IBM MQ リソース・アダプターは、 Jakarta EE をサポートするアプリケーション・サーバー内にデプロイする必要があります。

Java EE 7
IBM MQ 8.0 およびそれ以降のリソース・アダプターは JCA v1.7 をサポートし、JMS 2.0 サポートを提供します。 このリソース・アダプターは、Java EE 7 およびそれ以降のアプリケーション・サーバー内にデプロイされる必要があります (438 ページの『IBM MQ リソース・アダプターのサポートに関するステートメント』を参照)。

IBM MQ 8.0 以降のリソース・アダプターは、Java Platform, Enterprise Edition 7 仕様に準拠するものとして認証されているすべてのアプリケーション・サーバーにインストールできます。 IBM MQ 8.0 以降のリソース・アダプターを使用すると、アプリケーションは BINDINGS または CLIENT トランスポートのいずれかを使用してキュー・マネージャーに接続できます。

重要 : IBM MQ 8.0 以降のリソース・アダプターは、JMS 2.0 をサポートするアプリケーション・サーバーにのみデプロイできます。

WebSphere Application Server traditional とのリソース・アダプターの使用

IBM MQ 9.0 以降、IBM MQ リソース・アダプターは WebSphere Application Server traditional 9.0 以降でプリインストールされます。 そのため、新規リソース・アダプターのインストールは要求されません。

JM 3.0 WebSphere Application Server traditional は現在 Jakarta EE をサポートしていません。 IBM MQ リソース・アダプターのサポート・ステートメントを参照してください。

注 : IBM MQ 9.0 以降のリソース・アダプターは、CLIENT または BINDINGS トランスポート・モードで、サービス IBM MQ キュー・マネージャー内のいずれにも接続できます。

WebSphere Liberty とのリソース・アダプターの使用

WebSphere Liberty から IBM MQ に接続するには、 IBM MQ リソース・アダプターを使用する必要があります。 Liberty には IBM MQ リソース・アダプターが含まれないため、別途 Fix Central から取得する必要があります。

V 9.3.0 V 9.3.0 使用するリソース・アダプターのバージョンは、 Jakarta EE または Java EE をサポートする Liberty のバージョンにデプロイするかどうかによって異なります。

リソース・アダプターのダウンロードとインストールの方法について詳しくは、446 ページの『Liberty へのリソース・アダプターのインストール』を参照してください。

関連概念

453 ページの『インバウンド通信のリソース・アダプターの構成』
インバウンド通信を構成するには、1 つ以上の ActivationSpec オブジェクトのプロパティーを定義します。

470 ページの『アウトバウンド通信のリソース・アダプターの構成』
アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』
IBM MQ classes for JMS および IBM MQ classes for Jakarta Messaging は、 IBM MQ で提供される Java メッセージング・プロバイダーです。 JMS および Jakarta Messaging 仕様で定義されたインターフェースの実装に加えて、これらのメッセージング・プロバイダーは、2 セットの拡張機能を Java メッセージング API に追加します。

347 ページの『IBM MQ classes for Java の使用』
Java 環境で IBM MQ を使用します。 IBM MQ classes for Java では、Java アプリケーションは IBM MQ に IBM MQ クライアントとして接続するか、または IBM MQ キュー・マネージャーに直接接続することができます。

関連資料

最新のリソース・アダプター保守レベルを使用するためのアプリケーション・サーバーの構成
IBM MQ リソース・アダプターの問題判別

WebSphere Application Server のトピック

IBM MQ リソース・アダプターの保守
IBM MQ メッセージング・プロバイダーを使用するための JMS アプリケーションの Liberty へのデプロイ

IBM MQ リソース・アダプターのサポートに関するステートメント

アプリケーションとキュー・マネージャーの間の通信に使用する必要がある IBM MQ リソース・アダプターは、 Jakarta Messaging 3.0 API を使用するか、 JMS 2.0 API を使用するかによって異なります。

JMS 2.0 IBM MQ 8.0 以降には、 JMS 2.0 仕様を実装するリソース・アダプターが付属しています。 Java Platform, Enterprise Edition 7 (Java EE 7) に準拠し、そのために JMS 2.0 をサポートしているアプリケーション・サーバーにのみデプロイできます。 Java Platform, Enterprise Edition の認定アプリケーション・サーバーのリストは、 Oracle の Web サイトで保守されています。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、新規アプリケーションの開発で Jakarta Messaging 3.0 がサポートされるようになりました。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 Java EE および JMS 2.0 をサポートするリソース・アダプターに加えて、IBM MQ 9.3.0 は Jakarta Messaging をサポートするリソース・アダプターを提供します。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、IBM MQ classes for JMS の使用を参照してください。

WebSphere Liberty へのデプロイメント

WebSphere Liberty 8.5.5 Fix Pack 6 以降および WebSphere Application Server Liberty 9.0 以降は、Java EE 7 認証アプリケーションサーバーで、IBM MQ 9.0 リソース・アダプターをその中にデプロイできます。

V 9.3.0 V 9.3.0 Jakarta Messaging 用の IBM MQ リソース・アダプターを Liberty で使用するには、 Jakarta EE をサポートするバージョンの Liberty を使用する必要があります。

WebSphere Liberty には、リソース・アダプターの操作に使用できる以下の機能があります。

- JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 リソース・アダプターでの作業を可能にする messaging-3.0 フィーチャー。
- JMS 2.0 リソース・アダプターの使用を可能にする wmqJmsClient-2.0 機能。
- JMS 1.1 リソース・アダプターの使用を可能にする wmqJmsClient-1.1 機能。

重要 :

- JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 用の IBM MQ リソース・アダプターは、Jakarta EE をサポートするバージョンの Liberty にデプロイする必要があります。 このリソース・アダプターは、 Jakarta EE 以外の古い Java EE 仕様をサポートするバージョンの Liberty では使用できません。
- JMS 2.0 JMS 2.0 をサポートする IBM MQ 8.0 以降のリソース・アダプターは、 wmqJmsClient-2.0 機能とともにデプロイする必要があります。

WebSphere Application Server traditional へのデプロイメント

WebSphere Application Server traditional 9.0 は、IBM MQ 9.0 リソース・アダプターが既にインストールされた状態で提供されます。 そのため、新規リソース・アダプターのインストールは要求されません。 インストールされているリソース・アダプターは、サポートされているバージョンの IBM MQ で実行されている任意のキュー・マネージャーに CLIENT または BINDINGS トランスポート・モードで接続できます。 詳しくは、 440 ページの『IBM MQ 8.0 以降のキュー・マネージャーへの接続』を参照してください。

重要 :

- IBM MQ 9.0 リソース・アダプターは、IBM MQ 9.0 より前のバージョンの WebSphere Application Server traditional にはデプロイできません。 これらのバージョンは Java EE 7 認定ではないためです。
- JM 3.0 WebSphere Application Server traditional は現在 Jakarta EE をサポートしていません。

WebSphere Application Server に付属のリソース・アダプターのバージョンについて詳しくは、技術情報「Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server?」を参照してください。

他のアプリケーション・サーバーでのリソース・アダプターの使用

他のすべての Java EE 7 または Jakarta EE 準拠アプリケーション・サーバーの場合、 IBM MQ リソース・アダプターの インストール検査テスト (IVT) が正常に完了した後に発生した問題を IBM に報告して、 IBM MQ 製品トレースおよびその他の IBM MQ 診断情報を調査することができます。 IBM MQ リソース・アダプター IVT を正常に実行できない場合、検出された問題は、アプリケーション・サーバー固有の誤ったデプロイメントまたは誤ったリソース定義が原因である可能性があります。その問題は、アプリケーション・サーバーの資料およびそのアプリケーション・サーバーのサポート組織を使用して調査する必要があります。

Java ランタイム

アプリケーション・サーバーの実行に使用される Java Runtime (JRE) は、IBM MQ 9.0 以降の Client でサポートされるものでなければなりません。 詳しくは、IBM MQ のシステム要件を参照してください。 (レポートを表示するバージョンおよびオペレーティング・システムまたはコンポーネントを選択し、「**サポートされるソフトウェア (Supported Software)**」タブの下にリストされている **Java** リンクをたどってください)。

IBM MQ 8.0 以降のキュー・マネージャーへの接続

Java EE 7 認定アプリケーション・サーバーにデプロイされたリソース・アダプターを使用して IBM MQ 8.0 以降のキュー・マネージャーに接続する場合、全範囲の JMS 2.0 機能を使用できます。 JMS 2.0 機能を使用するには、IBM MQ メッセージング・プロバイダーの通常モードを使用して、リソース・アダプターがキュー・マネージャーに接続する必要があります。 詳しくは、JMS **PROVIDERVERSION** プロパティーの構成を参照してください。

Jakarta EE 認定アプリケーション・サーバーにデプロイされているリソース・アダプターを使用して IBM MQ 9.3 キュー・マネージャーに接続すると、全範囲の Jakarta Messaging 3.0 機能を使用できます。

MQ の機能拡張

JMS 2.0 の仕様によって、いくつかの動作方法が変更されました。 IBM MQ 8.0 以降ではこの仕様を実装しているため、IBM MQ 8.0 以降と旧バージョンの製品とでは動作に違いがあります。 IBM MQ 8.0 以降では、IBM MQ classes for JMS に Java システム・プロパティー `com.ibm.mq.jms.SupportMQExtensions` のサポートが含まれています。このプロパティーを TRUE に設定すると、これらのバージョンの IBM MQ は、IBM WebSphere MQ 7.5 以前のバージョンの動作に戻ります。 プロパティーのデフォルト値は FALSE です。

IBM MQ 9.0 以降のリソース・アダプターには、`com.ibm.mq.jms.SupportMQExtensions` Java システム・プロパティーと同じ効果とデフォルト値を持つ `supportMQExtensions` というリソース・アダプター・プロパティーも含まれています。 このリソース・アダプター・プロパティーは、`ra.xml` ではデフォルトで false に設定されています。

リソース・アダプター・プロパティーと Java システム・プロパティーの両方が設定されている場合は、システム・プロパティーが優先されます。

WebSphere Application Server traditional 9.0 に既にデプロイされたリソース・アダプターでは、移行を支援するために、このプロパティーが自動的に TRUE に設定されていることに注意してください。

詳細については、328 ページの『SupportMQExtensions プロパティー』を参照してください。

一般的な問題

セッション・インターリービングはサポートされない

一部のアプリケーション・サーバーは、同じ JMS セッションを複数のトランザクションで使用できる (ただし、一度に 1 つのトランザクションでのみ使用可能) セッション・インターリービングという機能を備えています。 IBM MQ リソース・アダプターはこの機能をサポートしないので、以下の問題が生じることがあります。

メッセージを MQ キューに書き込もうとすると、理由コード 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE) で失敗します。

xa_close() の呼び出しは理由コード -3 (XAER_PROTO) で失敗し、 アプリケーション・サーバーからアクセスされている IBM MQ キュー・マネージャーにプローブ ID AT040010 の FDC が生成されます。 この機能を無効にする方法について詳しくは、アプリケーション・サーバーの資料を参照してください。

XA トランザクション・リカバリーのための XA リソースのリカバリー方法についての Java Transaction API (JTA) 仕様

JTA 仕様のセクション 3.4.8 には、XA トランザクション・リカバリーを実行するために XA リソースを再作成する具体的なメカニズムが規定されていません。 そのため、XA トランザクションに関係する XA リソースのリカバリー方法は、個々のトランザクション・マネージャーに (つまり、アプリケーション・サーバーに) 委ねられています。 アプリケーション・サーバーによっては、XA トランザクション・リカバリーを実行するために使用するアプリケーション・サーバー固有の具体的なメカニズムを、IBM MQ 9.0 リソース・アダプターに実装しないことも可能です。

ManagedConnectionFactory での接続のマッチング

アプリケーション・サーバーは、IBM MQ リソース・アダプターによって提供される ManagedConnectionFactory インスタンスに対して matchManagedConnections メソッドを呼び出すことができます。 ManagedConnection が返されるのは、アプリケーション・サーバーからそのメソッドに渡された **javax.security.auth.Subject** 引数と

javax.resource.spi.ConnectionRequestInfo 引数の両方に一致するものをメソッドが検出した場合のみです。

IBM MQ リソース・アダプターの制限

IBM MQ リソース・アダプターは、すべての IBM MQ プラットフォームでサポートされます。 ただし、IBM MQ リソース・アダプターを使用する場合には、IBM MQ の一部の機能が利用できない、または制限されます。

IBM MQ リソース・アダプターには以下の制限があります。

- IBM MQ 8.0 以降、リソース・アダプターは、 JMS 2.0 機能を提供する Java Platform, Enterprise Edition 7 (Java EE 7) リソース・アダプターです。 したがって、IBM MQ 8.0 以降のリソース・アダプターを Java EE 7 以降の認定アプリケーション・サーバーにインストールする必要があります。 これは CLIENT または BINDINGS トランスポート・モードでサービス・キュー・マネージャー内のいずれにも接続できます。
- WebSphere Liberty アプリケーション・サーバー内で実行する場合、固定化された IBM MQ classes for Java はサポートされません。 他のアプリケーション・サーバー内では、IBM MQ classes for Java の使用をお勧めしません。 Java EE 内の IBM MQ classes for Java の考慮事項について詳しくは、IBM 技術情報「Using WebSphere MQ Java Interfaces in J2EE/JEE Environments」を参照してください。
- z/OS 上の WebSphere Liberty アプリケーション・サーバー内で実行する場合は、 wmqJmsClient-2.0 機能を使用する必要があります。 汎用 JCA サポートは、z/OS では利用できません。
- IBM MQ リソース・アダプターは、Java 以外の言語で作成されたチャネル出口プログラムをサポートしません。
- アプリケーション・サーバーが実行している間、sslFipsRequired プロパティーの値は、すべての JCA リソースについて true であるか、すべての JCA リソースについて false である必要があります。 JCA リソースが同時に使用されない場合でも、これが要件です。 sslFipsRequired プロパティーの値が JCA リソースごとに異なる場合、TLS 接続が使用されていなくても、IBM MQ は理由コード MQRC_UNSUPPORTED_CIPHER_SUITE を出します。
- アプリケーション・サーバーに複数の鍵ストアを指定することはできません。 複数のキュー・マネージャーへの接続がある場合、すべての接続は同じ鍵ストアを使用する必要があります。 この制限は、WebSphere Application Server には適用されません。
- 適合するクライアント接続チャネル定義を複数指定してクライアント・チャネル定義テーブル (CCDT) を使用すると、リソース・アダプターが別のチャネル定義を選択し、それゆえに CCDT から異なるキュー・マネージャーを選択するという失敗をした場合に、トランザクション・リカバリーの問題を引き起こす可能性があります。 リソース・アダプターは、そのような構成が使用されないように防ぐアクションを実行しません。 トランザクション・リカバリーの問題を引き起こす可能性がある構成を避けるのは、ユーザーの責任となります。
- 接続再試行機能は、 Java EE コンテナー (EJB/Servlet) で実行されている場合のアウトバウンド接続ではサポートされません。 アダプターが JEE コンテナー・コンテキストで使用される場合、トランザクション構成に関係なく、またはトランザクション化されていない使用のために、アウトバウンド JMS の接続再試行はまったくサポートされません。
- Java EE Connector Architecture のバージョン 1.7 仕様のセクション 9.1.9 で定義されている、JMS 接続の再認証はサポートされていません。 IBM MQ リソース・アダプター内の `ra.xml` ファイルでは、**reauthentication-support** というプロパティーが値 `false` に設定されている必要があります。 アプリケーション・サーバーで JMS 接続を再認証しようとすると、IBM MQ リソース・アダプターで javax.resource.spi.SecurityException がスローされ、MQJCA1028 メッセージ・コードが出されます。

関連タスク

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

関連資料

AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

WebSphere Application Server と IBM MQ リソース・アダプター

IBM MQ リソース・アダプターは、 WebSphere Application Server で IBM MQ メッセージング・プロバイダーとの JMS メッセージングを実行するアプリケーションによって使用されます。

重要 : IBM MQ リソース・アダプターは、WebSphere Application Server 6.0 または WebSphere Application Server 6.1 と一緒に使用しないでください。

WebSphere Application Server traditional 9.0 には、IBM MQ 9.0 リソース・アダプターのバージョンが含まれます。 IBM MQ 9.0 以降のリソース・アダプターは、以前のバージョンの WebSphere Application Server にはデプロイできません。これらのバージョンは Java EE 7 認定ではないためです。

JM 3.0 V 9.3.0 V 9.3.0 WebSphere Application Server traditional は現在 Jakarta EE をサポートしていません。 IBM MQ リソース・アダプターのサポート・ステートメントを参照してください。

JMS アプリケーションを使用して WebSphere Application Server 内から IBM MQ キュー・マネージャーのリソースにアクセスする場合は、 WebSphere Application Server の IBM MQ メッセージング・プロバイダーを使用します。 IBM MQ メッセージング・プロバイダーには、IBM MQ classes for JMS の 1 つのバージョンが含まれています。 詳しくは、技術情報『Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?』を参照してください。

重要 : IBM MQ classes for JMS または IBM MQ classes for Java JAR ファイルのいずれもアプリケーションに含めないでください。 含めると、ClassCastExceptions となる可能性があり、保守が困難になるおそれがあります。

Liberty と IBM MQ リソース・アダプター

IBM MQ リソース・アダプターは、 Liberty フィーチャーを使用して WebSphere Liberty にインストールできます。 使用する機能は、インストールするリソース・アダプターのバージョンによって異なります。 また制約はありますが、一般的な Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) サポートを使用してリソース・アダプターをインストールすることもできます。

リソース・アダプターを Liberty にインストールする場合の一般的な制約事項

wmqJmsClient-1.1 フィーチャーまたは wmqJmsClient-2.0 フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、リソース・アダプターには以下の制約事項が適用されます。

- IBM MQ classes for Java は Liberty ではサポートされません。 IBM MQ Liberty メッセージング・フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、これらを使用してはいけません。 詳しくは、 Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。
- IBM MQ リソース・アダプターのトランスポート・タイプは BINDINGS_THEN_CLIENT です。 このトランスポート・タイプは IBM MQ Liberty メッセージング・フィーチャーではサポートされていません。
- IBM MQ 9.0 より前は、Advanced Message Security (AMS) フィーチャーが IBM MQ Liberty メッセージング・フィーチャーに含まれていませんでした。 しかし、AMS は IBM MQ 9.0 以降のリソース・アダプターでサポートされます。

注 : IBM MQ のバージョンが IBM MQ 9.0.0.6 および IBM MQ 9.1.0.1 より大きい場合は、 ssl-1.0 フィーチャーの代わりに transportSecurity-1.0 フィーチャーを使用する必要があります。

詳細については、以下を参照してください。

Liberty での SSL 通信の使用可能化
Liberty での SSL のデフォルト
Transport Security 1.0

Liberty フィーチャーを使用する場合の制約事項

WebSphere Liberty 8.5.5 Fix Pack 2 から WebSphere Liberty 8.5.5 Fix Pack 5 まで (両方を含む) は、wmqJmsClient-1.1 フィーチャーしか提供されていなかったため、使用できるのは JMS 1.1 のみでした。 WebSphere Liberty 8.5.5 Fix Pack 6 で、wmqJmsClient-2.0 フィーチャーが追加され、JMS 2.0 が使用可能になりました。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、Jakarta Messaging 3.0 がサポートされます。 Jakarta Messaging 用の IBM MQ リソース・アダプターを Liberty で使用するには、 Jakarta EE をサポートするバージョンの Liberty を使用する必要があります。 Jakarta Messaging 用のリソース・アダプターは、Liberty 汎用 messaging-3.0 フィーチャーと共に使用する必要があります。

使用する必要がある機能は、使用しているリソース・アダプターのバージョンによって異なります。

- IBM MQ 8.0.0 Fix Pack 3 以降の IBM MQ 8.0 リソース・アダプターは、wmqJmsClient-2.0 フィーチャーでのみ使用できます。
- IBM MQ 9.0 リソース・アダプターは、wmqJmsClient-2.0 フィーチャーでのみ使用できます。
- JM 3.0 V 9.3.0 V 9.3.0 messaging-3.0 フィーチャーにより、Jakarta Messaging 3.0 リソース・アダプターを操作できます。

一般的な JCA サポートを使用する場合の制約事項

汎用 JCA サポートを使用する場合は、以下の制約事項が適用されます。

- 汎用 JCA サポートを使用する場合は、 JMS のレベルを指定する必要があります。 JMS 2.0 および JCA 1.7 は、IBM MQ 8.0.0 Fix Pack 3 以降の IBM MQ 8.0 リソース・アダプターでのみ使用する必要があります。
- 汎用 JCA サポートを使用して z/OS 上で IBM MQ リソース・アダプターを実行することはできません。 z/OS で IBM MQ リソース・アダプターを実行するには、 wmqJmsClient-1.1 または wmqJmsClient-2.0 フィーチャーを指定して実行する必要があります。
- リソース・アダプターの場所は、以下の xml 要素を使用して指定します。

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

重要 : ID タグの値は、wmqJms 以外の任意の値にすることができます。 wmqJms を ID として使用すると、Liberty がリソース・アダプターを正しくロードできなくなります。 wmqJms は、IBM MQ の特定のフィーチャーを表すために内部で使用される ID だからです。 誤って使用すると NullPointerException が生成されます。

以下の例は、 JMS 2.0 を実行する際の `server.xml` ファイルからのいくつかのスニペットを示しています。

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

ヒント : jca-1.7 および jms-2.0 のフィーチャーが使用されていて、wmqJmsClient-2.0 フィーチャーは使用されていないことに注意してください。

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

ヒント : ID に mqJms が使用されていることに注意してください。これが望ましい値です。 wmqJms は使用しないでください。

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

ヒント : classloaderProvider は、ID mqJms を介してリソース・アダプターに戻されることに注意してください。これは、 IBM MQ 固有のクラスのロードを許可するためです。

汎用 JCA サポートを使用してトレースする場合の制約事項

Liberty トレース・システム内にトレースおよびロギングは組み込まれていません。 代わりに、 IBM MQ classes for JMS アプリケーションのトレースで説明されているように、Java システム・プロパティーまたは IBM MQ classes for JMS 構成ファイルを使用して、 IBM MQ リソース・アダプター・トレースを使用可能にする必要があります。 Liberty で Java システム・プロパティーを設定する方法について詳しくは、 WebSphere Liberty の資料を参照してください。

例えば、 Liberty 19.0.0.9 で IBM MQ リソース・アダプターのトレースを有効にするには、 Liberty ファイル `jvm.options` にエントリーを追加します。

1. `jvm.options` という名前のテキスト・ファイルを作成します。
2. 以下の JVM オプションを挿入して、トレースを 1 行に 1 つずつこのファイルに挿入します。

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. これらの設定を単一サーバーに適用するには、 `jvm.options` を以下の場所に保存します。

   ```
   ${server.config.dir}/jvm.options
   ```

   これらの変更をすべての Liberty に適用するには、 `jvm.options` を以下の場所に保存します。

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   これは、ローカルに定義された `jvm.options` ファイルを持たないすべての JVM に対して有効になります。
4. 変更を有効にするには、サーバーを再始動してください。

これにより、ディレクトリー `<path_to_trace_to>` にある `MQRA-WLP_<process identifier>.trc` と呼ばれるトレース・ファイルにトレースが書き込まれます。

クライアント・チャネル定義テーブルでの Liberty XA の完全サポート

IBM MQ 9.2.0 以降で WebSphere Liberty 18.0.0.2 を使用する場合は、XA トランザクションとともにクライアント・チャネル定義テーブル (CCDT) 内のキュー・マネージャー・グループを使用できます。 つまり、トランザクションの整合性を維持しながら、キュー・マネージャー・グループによって提供されるワークロード分散と可用性を利用できるようになりました。

キュー・マネージャーへの接続エラーが発生した場合、トランザクションを解決するためには、キュー・マネージャーが再び使用可能にならなければなりません。 トランザクションのリカバリーは Liberty によって管理されるため、キュー・マネージャーが再び使用可能になるための適切な期間が与えられるように、トランザクション・マネージャーを構成しなければならない場合があります。 詳しくは、 WebSphere Liberty 製品資料の「 トランザクション・マネージャー (トランザクション) 」を参照してください。

これはクライアント・サイド・フィーチャーです。つまり、 IBM MQ 9.2.0 以降のキュー・マネージャーではなく、 IBM MQ 9.2.0 以降のリソース・アダプターが必要です。

IBM MQ リソース・アダプターのインストール

IBM MQ リソース・アダプターは、リソース・アーカイブ (RAR) ファイルとして提供されます。 RAR ファイルをアプリケーション・サーバーにインストールします。 ディレクトリーをシステム・パスに追加する必要がある場合があります。

このタスクについて

IBM MQ リソース・アダプターは、リソース・アーカイブ (RAR) ファイルとして提供されます。

- JM 3.0 V 9.3.0 V 9.3.0 Jakarta Messaging 3.0 の場合、このファイルの名前は `wmq.jakarta.jmsra.rar` です。 RAR ファイルには、 IBM MQ classes for Jakarta Messaging および Jakarta Connectors Architecture (JCA) インターフェースの IBM MQ 実装が含まれています。
- JMS 2.0 JMS 2.0 の場合、このファイルの名前は `wmq.jmsra.rar` です。 RAR ファイルには、 IBM MQ classes for JMS と、 Java EE Connector Architecture (JCA) インターフェースの IBM MQ 実装が含まれています。

IBM MQ 製品インストールの一部としてリソース・アダプターをインストールすると、RAR ファイルは、445 ページの表 61 に示すディレクトリーに IBM MQ classes for JMS と共にインストールされます。

表 *61.* 各プラットフォームの *RAR* ファイルを含む *IBM MQ* ディレクトリー

プラットフォーム	**ディレクトリー**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*¥java¥lib¥jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

IBM MQ リソース・アダプターを使用して、アプリケーション・サーバーから IBM MQ に接続する必要があります。 使用しているアプリケーション・サーバーに応じて、リソース・アダプターはプリインストールされていることもあれば、ユーザーがインストールしなければならないこともあります。

表 *62.* アプリケーション・サーバーへのリソース・アダプターのインストール

アプリケーション・サーバー	**プリインストールされるか、インストールする必要があるか**
WebSphere Application Server traditional 9.0	IBM MQ 9.0 リソース・アダプターは WebSphere Application Server traditional 9.0 内にプリインストールされています。 そのため、 WebSphere Application Server traditional 9.0 に新しいリソース・アダプターをインストールする必要はありません。
WebSphere Liberty	WebSphere Liberty には IBM MQ リソース・アダプターが含まれていないため、別途 Fix Central から取得する必要があります。
その他の Java EE または Jakarta EE アプリケーション・サーバー	WebSphere Liberty の場合のように、Fix Central とは別にリソース・アダプターを取得してください。

手順

- WebSphere Liberty、または別の Java EE または Jakarta EE アプリケーション・サーバーから IBM MQ に接続する場合は、 446 ページの『Liberty へのリソース・アダプターのインストール』 の説明に従って、 IBM MQ リソース・アダプターをダウンロードしてインストールします。
- Linux AIX
  AIX and Linux システムで接続をバインディングするためには、Java Native Interface (JNI) ライブラリーを含むディレクトリーが、システム・パスにあることを確認してください。
  このディレクトリー (IBM MQ classes for JMS ライブラリーも含む) の場所については、97 ページの『Java Native Interface (JNI) ライブラリーの構成』 を参照してください。

  Windows Windows では、このディレクトリーは、IBM MQ classes for JMS のインストール時にシステム・パスに自動的に追加されます。

ヒント : システム・パスの設定に代わる方法として、IBM MQ リソース・アダプターにある nativeLibraryPath というプロパティーを使用して、JNI ライブラリーの場所を指定することができます。 例えば、WebSphere Liberty では、以下の例に示すように構成されます。

```
<wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
```

トランザクションは、クライアント・モードとバインディング・モードの両方でサポートされます。

Liberty へのリソース・アダプターのインストール

WebSphere Liberty、または他の Java EE または Jakarta EE アプリケーション・サーバーから IBM MQ に接続するには、 IBM MQ リソース・アダプターを使用する必要があります。 Liberty には IBM MQ リソース・アダプターが含まれないため、別途 Fix Central から取得する必要があります。

始める前に

注 : このトピックの情報は、WebSphere Application Server traditional 9.0 には適用されません。 IBM MQ 9.0 リソース・アダプターは WebSphere Application Server traditional 9.0 内にプリインストールされています。 そのため、 このケースでは、新規リソース・アダプターのインストールは必要ありません。

この作業を開始する前に、Java runtime environment (JRE) がマシンにインストールされていて、JRE がシステム・パスに追加されていることを確認してください。

このインストール処理で使用する Java インストーラーは、root または他の特定のユーザーとして実行する必要はありません。 唯一の要件は、実行するユーザーに、ファイルを作成するディレクトリーに対する書き込み権限があることです。

Liberty バーションから WebSphere Liberty 8.5.5 Fix Pack 1 までの場合、EJB が `ejb-jar.xml` 内の構成のみを使用してデプロイされている場合は、Liberty プロファイルが使用している WebSphere Application Server バージョンには APAR PM89890 が適用されている必要があります。 リソース・アダプターのインストール検査プログラム (IVT) にはこの構成方式が使用されているので、IVT を実行するためには、この APAR が必要です。

JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、Jakarta Messaging 3.0 がサポートされます。 Jakarta Messaging 用の IBM MQ リソース・アダプターを Liberty で使用するには、 Jakarta EE をサポートするバージョンの Liberty を使用する必要があります。 例えば、 Liberty 汎用 messaging-3.0 フィーチャーを使用できます。

このタスクについて

Fix Central からダウンロードできるリソース・アダプターの JAR ファイルは実行可能です。 この実行可能ファイルを実行すると、IBM MQ の使用条件が表示されるので、これを受け入れる必要があります。 IBM MQ リソース・アダプターのインストール先ディレクトリーを尋ねられます。 その後、そのディレクトリーに、リソース・アダプター RAR ファイルとインストール検査テスト (IVT) プログラムがインストールされます。 デフォルトを受け入れることも、別のディレクトリーを指定することもできます。 その場合は、アプリケーション・サーバーのリソース・アダプター・ディレクトリーまたはシステム上の他の任意のディレクトリーを指定できます。 ディレクトリーが存在しない場合は、インストールの一環として作成されます。

IBM MQ 9.0 の前に、ダウンロードされるファイルの名前が *V.R.M.F*`-WS-MQ-Java-InstallRA.jar` の形式で存在していました。例: `8.0.0.6-WS-MQ-Java-InstallRA.jar` IBM MQ 9.0 では、ファイル名の形式は *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar` です。例: `9.0.0.0-IBM-MQ-Java-InstallRA.jar`

リソース・アダプターをダウンロードしてインストールしたら、WebSphere Liberty で構成することができます。

手順

1. IBM MQ リソース・アダプターを Fix Central からダウンロードします。

a) こちらのリンク IBM MQ リソース・アダプターをクリックします。

b) 表示された選択可能なフィックスのリストから、ご使用のバージョンの IBM MQ 用のリソース・アダプターを見つけます。

以下に例を示します。

```
release level: 9.1.4.0-IBM-MQ-Java-InstallRA
Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
Servers
```

次に、リソース・アダプターのファイル名をクリックし、ダウンロード・プロセスに従います。

2. ファイルをダウンロードしたディレクトリーから以下のコマンドを入力して、インストールを開始します。

IBM MQ 9.0 から、コマンドの形式は以下のようになっています。

```
java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
```

ここで、*V.R.M.F* はバージョン、リリース、変更、および修正パッケージ番号で、*V.R.M.F*-IBM-MQ-Java-InstallRA.jar は Fix Central からダウンロードされたファイルの名前です。

例えば、 IBM MQ 9.1.4 リリース用の IBM MQ リソース・アダプターをインストールするには、以下のコマンドを使用します。

```
java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
```

注: このインストールを実行するには、JRE がマシンにインストールされ、システム・パスに追加されている必要があります。

コマンドを入力すると、以下の情報が表示されます。

```
IBM MQ 9.1 を使用、抽出、またはインストールする前に、以下を受け入れる必要があります。
条件 1。 IBM 評価のためのご使用条件
プログラム 2。 IBM プログラムのご使用条件および追加情報
ライセンス情報。 以下の使用条件をよくお読みください。

ご使用条件は、以下を使用して個別に表示できます。
--viewLicense 契約オプション。
ここでライセンス条項を表示する場合は Enter を、スキップする場合は「x」を押してください。
```

3. ライセンス条項を確認して受け入れます。

a) ライセンスを表示するには、Enter を押します。

または x を押してライセンスの表示をスキップすることもできます。

ライセンスを表示した後、または x を選択した直後に、追加のライセンス条項を表示できることを示す次のメッセージが表示されます。

```
追加のライセンス情報は、以下を使用して個別に表示できます。
--viewLicense 情報オプション。
ここで追加のライセンス情報を表示する場合は Enter を、スキップする場合は「x」を押してください。
```

b) 追加のライセンス条項を表示するには、Enter を押します。

または x を押して追加のライセンス条項の表示をスキップすることもできます。

追加のライセンス条項を表示した後、または x を選択した直後に、使用条件の受け入れを求める次のメッセージが表示されます。

```
以下の「同意する」オプションを選択すると、以下の条件に同意したことになります。
ご使用条件および IBM 以外の条項（該当する場合)。 そうでない場合は、
同意する場合は、「同意しない」を選択します。

[1] 同意する、または [2] 同意しないを選択
```

c) 使用条件を受け入れて、インストール・ディレクトリーの選択に進むには、1 を選択します。

代わりに 2 を選択した場合、インストールは即時に終了します。

1 を選択した場合、ターゲットのインストール・ディレクトリーの選択を求める次のメッセージが表示されます。

```
製品ファイルのディレクトリーを入力するか、ブランクのままにしてデフォルト値を受け入れます。
デフォルトのターゲット・ディレクトリーは H:&#xa5;Liberty&#xa5;WMQ です。
製品ファイルのターゲット・ディレクトリー?
```

4. リソース・アダプターのインストール・ディレクトリーを指定します。
   - リソース・アダプターをデフォルトの場所にインストールする場合、値を指定せずに Enter を押します。
   - リソース・アダプターをデフォルトとは異なる場所にインストールする場合、 リソース・アダプターのインストール先ディレクトリーの名前を指定してから Enter を押します。

   選択した場所にファイルがインストールされた後に、以下の例に示すような確認メッセージが表示されます。

   ```
   ファイルを H:&#xa5;Liberty&#xa5;WMQ&#xa5;wmq に抽出しています。
   すべての製品ファイルを正常に抽出しました。
   ```

   インストール中に、wmq という名前の新規ディレクトリーが、選択されたインストール・ディレクトリー内に作成され、wmq ディレクトリーに以下のファイルがインストールされます。

   - インストール検査テスト・プログラム、 `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) または `wmq.jmsra.ivt` (JMS 2.0)。
   - IBM MQ RAR ファイル、`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 または `wmq.jmsra.rar` (JMS 2.0)。

5. JMS 2.0

   オプション: WebSphere Liberty Profile で Java EE 7 (JMS 2.0) リソース・アダプターを構成します。

   Liberty でリソース・アダプターを構成するために必要な手順は、以下のとおりです。 詳しくは、WebSphere Application Server 製品の資料を参照してください。

   a) wmqJmsClient-2.0 機能を `server.xml` ファイルへ追加し、IBM MQ リソース・アダプターとの作業を許可します。

   詳細については、437 ページの『使用するリソース・アダプターのバージョン』を参照してください。

   b) インストールした `wmq.jmsra.rar` (JMS 2.0) ファイルへの参照を追加します。

   JNDI を使用したサーブレットと MDB をサポートする構成の例は、以下のようなものになります。

   ```
   <featureManager>
      <feature>wmqJmsClient-2.0</feature>
      <feature>servlet-3.0</feature>
      <feature>jmsMdb-3.1</feature>
      <feature>jndi-1.0</feature>
      </featureManager>

   <variable name="wmqJmsClient.rar.location"
         value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
   ```

6. JM 3.0

   オプション: WebSphere Liberty Profile で Jakarta EE 9 (Jakarta Messaging 3.0) リソース・アダプターを構成します。

   Liberty でリソース・アダプターを構成するために必要な手順は、以下のとおりです。 詳しくは、WebSphere Application Server 製品の資料を参照してください。

   a) wmqJmsClient-3.0 機能を `server.xml` ファイルに追加して、 IBM MQ リソース・アダプターを操作できるようにします。

   詳細については、437 ページの『使用するリソース・アダプターのバージョン』を参照してください。

   b) インストールした `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) ファイルへの参照を追加します。

   JNDI を使用したサーブレットと MDB をサポートする構成の例は、以下のようなものになります。

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

注 : WebSphere Liberty Profile ではなく Open Liberty を使用している場合は、「wmqJmsClient-3.0」の代わりに汎用リソース・アダプター・サポート機能「messagingClient-3.0」を使用する必要があり、構成のその他の側面は異なります。 詳しくは、 Open Liberty の資料を参照してください。

IBM MQ リソース・アダプターの構成

IBM MQ リソース・アダプターを構成するには、さまざまな Java Platform, Enterprise Edition Connector Architecture (JCA) リソースを定義し、オプションでシステム・プロパティーを定義します。 インストール検証テスト (IVT) プログラムを実行するため、リソース・アダプターも構成する必要があります。 IBM サービスでは、IBM 以外のアプリケーション・サーバーが正しく構成されていることを示すためにこのプログラムの実行を必要とする場合があるため、これは重要です。

始める前に

このタスクは、ユーザーが既に JMS および IBM MQ classes for JMS に精通していることを前提としています。 IBM MQ リソース・アダプターを構成するために使用されるプロパティーの多くは、IBM MQ classes for JMS オブジェクトのプロパティーと同等であり、同じ機能を持っています。

このタスクについて

すべてのアプリケーション・サーバーは、独自の管理インターフェースのセットを備えています。 JCA リソースを定義するためのグラフィカル・ユーザー・インターフェースを備えたアプリケーション・サーバーもあれば、XML デプロイメント計画を作成するためのアドミニストレーターを必要とするアプリケーション・サーバーもあります。 したがって、各アプリケーション・サーバー用に IBM MQ リソース・アダプターを構成する方法については、この資料では扱われていません。

そのため、以下の手順では、構成する必要がある事柄についてのみ焦点を当てています。 JCA リソース・アダプターの構成方法については、使用しているアプリケーション・サーバーで提供されている資料を参照してください。

手順

以下のカテゴリーの JCA リソースを定義してください。

- ResourceAdapter オブジェクトのプロパティーを定義します。

  リソース・アダプターのグローバル・プロパティーを表すこれらのプロパティー (診断トレースのレベルなど) については、450 ページの『ResourceAdapter オブジェクト・プロパティーの構成』で説明しています。
- ActivationSpec オブジェクトのプロパティーを定義します。

  これらのプロパティーは、インバウンド通信用に MDB がアクティブ化される方法を決定します。 詳細については、453 ページの『インバウンド通信のリソース・アダプターの構成』を参照してください。
- ConnectionFactory オブジェクトのプロパティーを定義します。

  アプリケーション・サーバーは、これらのプロパティーを使用して、アウトバウンド通信用の JMS ConnectionFactory オブジェクトを作成します。 詳しくは、470 ページの『アウトバウンド通信のリソース・アダプターの構成』を参照してください。
- 管理対象の宛先オブジェクトのプロパティーを定義します。

  アプリケーション・サーバーは、これらのプロパティーを使用して、アウトバウンド通信用の JMS Queue オブジェクトまたは JMS Topic オブジェクトを作成します。 詳しくは、470 ページの『アウトバウンド通信のリソース・アダプターの構成』を参照してください。

- オプション: リソース・アダプターのデプロイメント計画を定義します。

  IBM MQ リソース・アダプター RAR ファイルには、リソース・アダプター用のデプロイメント記述子が含まれている `META-INF/ra.xml` というファイルが含まれています。 このデプロイメント記述子は、https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd の XML スキーマによって定義され、リソース・アダプターおよびそれが提供するサービスに関する情報が含まれています。 また、アプリケーション・サーバーもリソース・アダプターのデプロイメント計画を必要とすることがあります。 このデプロイメント計画はアプリケーション・サーバーに固有です。

必要に応じて JVM システム・プロパティーを指定します。

- Transport Layer Security (TLS) を使用している場合は、以下の例のように、鍵ストア・ファイルおよびトラストストア・ファイルの場所を JVM システム・プロパティーとして指定してください。

  ```
  java ... -Djavax.net.ssl.keyStore=
  key_store_location
           -Djavax.net.ssl.trustStore=trust_store_location
           -Djavax.net.ssl.keyStorePassword=key_store_password
  ```

  これらのプロパティーは、ActivationSpec オブジェクトと ConnectionFactory オブジェクトではプロパティーとして使用することができません。また、1 つのアプリケーション・サーバーに複数の鍵ストアを指定することもできません。 プロパティーは JVM 全体に適用されるため、アプリケーション・サーバーで実行している他のアプリケーションが TLS 接続を使用する場合には、そのアプリケーション・サーバーに影響を与えることがあります。 また、アプリケーション・サーバーはこれらのプロパティーを別の値にリセットすることもあります。 IBM MQ classes for JMS で TLS を使用する方法について詳しくは、257 ページの『IBM MQ classes for JMS での TLS の使用』を参照してください。

- オプション: 必要に応じて、警告メッセージをアプリケーション・サーバーの標準出力ログに記録するようにリソース・アダプターを構成します。

  リソース・アダプターのログ、警告、およびエラー・メッセージは、IBM MQ classes for JMS と同じメカニズムを使用します。 詳しくは、IBM MQ classes for JMS のエラーのロギングを参照してください。つまり、デフォルトでは、`mqjms.log` というファイルにメッセージが入ることになります。 リソース・アダプターを構成して、警告メッセージをアプリケーション・サーバーの標準出力ログに追加で記録するには、アプリケーション・サーバーの以下の JVM システム・プロパティーを設定します。

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  これは、IBM MQ classes for JMS のトレースを制御するために使用するプロパティーと同じプロパティーです。 IBM MQ classes for JMS の場合と同様に、`jms.config` ファイルを指すシステム・プロパティーを使用することもできます (99 ページの『IBM MQ classes for JMS/Jakarta Messaging 構成ファイル』を参照)。 JVM システム・プロパティーを設定する方法については、アプリケーション・サーバーの資料を参照してください。

インストール検証テストを実行するためにリソース・アダプターを構成します。

- IBM MQ リソース・アダプターに付属のインストール検査テスト (IVT) プログラムを実行するためにリソース・アダプターを構成します。

  IVT プログラムを実行するために構成すべきものに関する詳細は、490 ページの『リソース・アダプターのインストールの検証』を参照してください。

  IBM サービスでは、IBM 以外のアプリケーション・サーバーが正しく構成されていることを示すためにこのプログラムの実行を必要とする場合があるため、これは重要です。

  重要 : このプログラムを実行するには、その前にリソース・アダプターを構成しておく必要があります。

ResourceAdapter オブジェクト・プロパティーの構成

ResourceAdapter オブジェクトは、診断トレースのレベルなど、IBM MQ リソース・アダプターのグローバル・プロパティーをカプセル化します。 これらのプロパティーを定義するには、アプリケーション・サーバーの付属資料の説明に従ってリソース・アダプターの機能を使用してください。

ResourceAdapter オブジェクトには次の 2 つのプロパティー・セットがあります。

- 診断トレースに関連したプロパティー

- リソース・アダプターによって管理される接続プールに関連したプロパティー

これらのプロパティーの定義方法は、アプリケーション・サーバーに用意されている管理インターフェースによって異なります。 WebSphere Application Server traditional を使用している場合は、452 ページの『WebSphere Application Server traditional 構成』を参照してください。また、WebSphere Liberty を使用している場合は、452 ページの『WebSphere Liberty 構成』を参照してください。 他のアプリケーション・サーバーの場合は、そのアプリケーション・サーバー用の製品資料を参照してください。

診断トレースに関連したプロパティーの定義についての詳細は、IBM MQ リソース・アダプターのトレースを参照してください。

リソース・アダプターは、MDB へのメッセージの送達に使用される、JMS 接続の内部接続プールを管理します。 451 ページの表 63 は、接続プールに関連する ResourceAdapter オブジェクトのプロパティーを示します。

表 *63.* 接続プールに関連する *ResourceAdapter* オブジェクトのプロパティー

プロパティー名	タイプ	デフォルト値	説明
maxConnections	ストリング	50	IBM MQ キュー・マネージャーへの接続の最大数およびデプロイ済みの MDB の最大数
connectionConcurrency	ストリング	1	JMS 接続を共有する MDB の最大数。 接続を共有することはできません。このプロパティーの値は常に 1 です。
reconnectionRetryCount	ストリング	5	接続が失敗した場合に、リソース・アダプターが IBM MQ キュー・マネージャーに再接続しようとする最大数
reconnectionRetryInterval	ストリング	300 000	リソース・アダプターが IBM MQ キュー・マネージャーに再接続しようとする前に待機する時間 (ミリ秒)。
startupRetryCount	ストリング	0	開始時に MDB への接続を試行するデフォルトの回数 (アプリケーション・サーバーの始動時にキュー・マネージャーが実行されていない場合)。
startupRetryInterval	ストリング	30 000	次の接続開始を試行するまでのデフォルトのスリープ時間 (ミリ秒)。
supportMQExtensions	ストリング	false	IBM MQ JMS の動作を、JMS 2.0 の動作前の動作に戻します。 詳しくは、328 ページの『SupportMQExtensions プロパティー』を参照してください。
nativeLibraryPath	ストリング	<空>	バインディング・モード接続を許可するために IBM MQ JNI ライブラリーをロードする際に使用されるパス。 Windows Windows では、システム・パスには、一致する IBM MQ インストールのロケーションも含める必要があります。

MDB がアプリケーション・サーバーにデプロイされている場合、maxConnection プロパティーで指定された最大接続数を超えていなければ、新規 JMS 接続が作成され、キュー・マネージャーとの会話が開始されます。 このため、MDB の最大数は最大接続数と等しくなります。 デプロイされた MDB の数がこの最大数に達すると、別の MDB をデプロイしようとしても失敗します。 MDB が停止される場合、その接続は別の MDB が使用できます。

一般に、多数の MDB がデプロイされる場合、maxConnections プロパティーの値を大きくする必要があります。

IBM MQ キュー・マネージャーへの接続が失敗すると (例えば、ネットワーク障害のため)、reconnectionRetryCount および reconnectionRetryInterval プロパティーがリソース・アダプターの振る舞いを制御します。 接続が失敗すると、リソース・アダプターはその接続によって提供されるすべての MDB に対するメッセージの送達を、reconnectionRetryInterval プロパティーで指定されたインターバルの間、中断します。 その後、リソース・アダプターはキュー・マネージャーに再接続しようとします。 試行が失敗する場合、リソース・アダプターは、reconnectionRetryCount プロパティーによって課せられた制限に達するまで、reconnectionRetryInterval プロパティーで指定された間隔でさらに再接続しようとします。 すべての試行が失敗する場合、MDB が手動で再始動されるまで、送達は永久に停止されます。

一般に、ResourceAdapter オブジェクトは管理を必要としません。 ただし、例えば AIX and Linux システムで診断トレースを使用可能にするには、以下のプロパティーを設定できます。

```
traceEnabled:    true
traceLevel:      10
```

これらのプロパティーは、リソース・アダプターが開始されていない場合は効果がありません。 例えば、IBM MQ リソースを使用するアプリケーションがクライアント・コンテナーでのみ実行している場合が、これに該当します。 この状態では、診断トレースのプロパティーを Java 仮想マシン (JVM) システム・プロパティーとして設定できます。 以下の例のように、**java** コマンドで -D フラグを使用することによって、プロパティーを設定することができます。

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

ResourceAdapter オブジェクトのすべてのプロパティーを定義する必要はありません。 未指定のままになっているプロパティーはデフォルト値をとります。 管理された環境では、プロパティーの指定に 2 つの方法を混用しないほうが良いでしょう。 混用する場合、JVM システム・プロパティーが ResourceAdapter オブジェクトのプロパティーに優先して指定されます。

WebSphere Application Server traditional 構成

WebSphere Application Server traditional のリソース・アダプターにも同じプロパティーを使用できますが、これらのプロパティーは、リソース・アダプターのプロパティー・パネル内で設定する必要があります (WebSphere Application Server traditional 製品資料の「JMS プロバイダー設定」を参照してください)。 トレースは、WebSphere Application Server traditional 構成の診断セクションによって制御されます。 詳しくは、 WebSphere Application Server traditional 製品資料の「診断プロバイダーの操作」を参照してください。

WebSphere Liberty 構成

リソース・アダプターは、以下の例に示されているように、`server.xml` ファイル内の XML エレメントを使用して構成されます。

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
```

```
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

トレースは、次の XML エレメントを追加して有効にします。

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

インバウンド通信のリソース・アダプターの構成

インバウンド通信を構成するには、1 つ以上の ActivationSpec オブジェクトのプロパティーを定義します。

ActivationSpec オブジェクトのプロパティーは、メッセージ駆動型 Bean (MDB) が、IBM MQ キューから JMS メッセージを受信する方法を決定します。 MDB のトランザクションの振る舞いはデプロイメント記述子で定義されています。

ActivationSpec オブジェクトには次の 2 つのプロパティー・セットがあります。

- IBM MQ キュー・マネージャーへの JMS 接続を作成するために使用されるプロパティー
- 指定されたキューにメッセージが到着すると、それらを非同期で送達する JMS 接続コンシューマーを作成するために使用するプロパティー。

ActivationSpec オブジェクトのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。

IBM MQ キュー・マネージャーへの JMS 接続を作成するために使用されるプロパティー

453 ページの表 64 のプロパティーはすべてオプションです。

表 *64. JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
applicationName	ストリング	• 起動クラス名が使用可能である場合は、28 文字以内に調整されます。 使用可能でない場合は、ストリング `WebSphere MQ Client for Java` が使用されます。	アプリケーションをキュー・マネージャーに登録する際に使用した名前。 このアプリケーション名は、**DISPLAY CONN MQSC/PCF** コマンドによって表示されます (このフィールドの名前は **APPLTAG** です)。 または IBM MQ Explorer「**アプリケーション接続**」画面 (このフィールドの名前は **App name**) で表示されます。
brokerCCDurSubQueue [1]	ストリング	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが永続サブスクリプション・メッセージを受信するキューの名前
brokerCCSubQueue [1]	ストリング	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが非永続サブスクリプション・メッセージを受信するキューの名前
brokerControlQueue [1]	ストリング	• **SYSTEM.BROKER.CONTROL.QUEUE** • キュー名	ブローカー制御キューの名前

表 64. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
brokerQueueManager [1]	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	ブローカーが稼働しているキュー・マネージャーの名前
brokerSubQueue [1]	ストリング	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • キュー名	非永続メッセージ・コンシューマーがメッセージを受信するキューの名前
brokerVersion [1]	ストリング	• **指定なし** - ブローカーを V6 から V7 に移行した後、このプロパティーを設定し、RFH2 ヘッダーが使用されないようにします。 移行した後、このプロパティーは関係なくなります。 • **V1** - IBM MQ パブリッシュ/サブスクライブ・ブローカーを使用する場合。この値は、TRANSPORT が BIND または CLIENT に設定されている場合のデフォルト値です。 • **V2** - IBM Integration Bus のブローカーをネイティブ・モードで使用する場合。 この値は、TRANSPORT が DIRECT または DIRECTHTTP に設定されている場合のデフォルト値です。	使用されているブローカーのバージョン
ccdtURL	ストリング	• **null** • Uniform Resource Locator (URL)	クライアント・チャネル定義テーブル (CCDT) が格納されているファイルの名前と場所を識別し、このファイルのアクセス方法を指定する URL。
CCSID	ストリング	• **819** • Java 仮想マシン (JVM) でサポートされるコード化文字セット ID	接続用のコード化文字セット ID
channel	ストリング	• **SYSTEM.DEF.SVRCONN** • MQI チャネルの名前	使用する MQI チャネルの名前
cleanupInterval [1]	int	• **3 600 000** • 正整数	パブリッシュ/サブスクライブ・クリーンアップ・ユーティリティーがバックグラウンドで実行する間隔 (ミリ秒)。
cleanupLevel [1]	ストリング	• **SAFE** • NONE • strong • FORCE • NONDUR	ブローカー・ベースのサブスクリプション・ストアのクリーンアップ・レベル

表 64. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
clientID	ストリング	• **null** • クライアント ID	接続のクライアント ID
cloneSupport	ストリング	• **DISABLED** - 一度に稼働できる永続トピック・サブスクライバーのインスタンスは 1 つだけです。 • ENABLED - 1 つの永続トピック・サブスクライバーの 2 つ以上のインスタンスを同時に稼働できますが、各インスタンスを別々の Java 仮想マシン (JVM) で稼働させる必要があります。	1 つの永続トピック・サブスクライバーの複数のインスタンスを同時に稼働できるかどうか
connectionFactoryLookup	ストリング	• **null** • ConnectionFactory オブジェクトの JNDI 名	このプロパティーが設定されている場合、ActivationSpec は、アプリケーション・サーバーの JNDI 名前空間で指定された JNDI 名を持つ JMS ConnectionFactory オブジェクトを検索し、そのオブジェクトのプロパティーを使用して IBM MQ キュー・マネージャーへの JMS 接続を作成します。ただし、1 つの例外があります。 JMS 接続を作成するときに ActivationSpec のプロパティーから clientID のみが使用されます。 詳しくは、466 ページの『ActivationSpec の connectionFactoryLookup プロパティーと destinationLookup プロパティー』を参照してください。

表 64. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
connectionNameList	ストリング	• **localhost(1414)** • コンマで区切られている項目で構成されるストリングです。各項目は以下の形式になります。 `HOSTNAME(PORT)` ここで、*HOSTNAME* は DNS 名または IP アドレスです。	インバウンド通信に使用される TCP/IP 接続名のリスト。 指定すると、**hostname** プロパティーおよび **port** プロパティーは、**connectionNameList** に置き換えられます。 このプロパティーを使用して、複数インスタンス・キュー・マネージャーに再接続します。 **connectionNameList** は **localAddress** の形式に類似していますが、混同しないでください。**localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
V 9.3.0 V 9.3.0 dynamicallyBalanced [4]	Boolean	• **false** • true	均一クラスター内のアプリケーション・バランシングの一部として、異なるキュー・マネージャーからのメッセージを受け取るようにこの MDB で要求できるかどうか。
failIfQuiesce	Boolean	• **true** • false	キュー・マネージャーが静止状態の場合、特定のメソッドの呼び出しが失敗するかどうか
headerCompression	ストリング	• **NONE** • SYSTEM - RLE メッセージ・ヘッダーの圧縮が実行される	接続のヘッダー・データを圧縮するために使用できる技法のリスト
hostName	ストリング	• **localhost** • ホスト名 • IP アドレス	キュー・マネージャーが存在しているシステムのホスト名または IP アドレス。 **connectionNameList** プロパティーが指定されている場合、**hostname** プロパティーと **port** プロパティーはこのプロパティーに置き換えられます。

表 64. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
localAddress	ストリング	• **null** • 以下の形式のストリング `[ host_name ][(low_port [, high_port ])]` ここで、*host_name* はホスト名または IP アドレス、*low_port* および *high_port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	このプロパティーは、キュー・マネージャーへの接続に対して、以下のいずれかまたは両方を指定します。 • 使用されるローカル・ネットワーク・インターフェース • 使用されるローカル・ポート、またはローカル・ポートの範囲 **localAddress** は **connectionNameList** の形式に類似していますが、混同しないでください。 **localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
messageCompression	ストリング	• **NONE** • ブランク文字で区切られた以下の値の 1 つ以上のリスト。 RLE ZLIBFAST ZLIBHIGH	接続のメッセージ・データを圧縮するために使用できる技法のリスト
messageRetention [1]	Boolean	• **true** - 不要なメッセージを入力キューに残す • false - 不要なメッセージは指定されている後処理オプションに従って扱う	入力キューにある不要なメッセージを接続のコンシューマーに保持させるかどうか
messageSelection [1]	ストリング	• **CLIENT** • BROKER	メッセージ選択を IBM MQ classes for JMS またはブローカーのどちらが行うかを決定します。 ブローカーによるメッセージ選択は、brokerVersion の値が 1 の場合、サポートされません。
パスワード	ストリング	• **null** • パスワード	キュー・マネージャーへの接続を作成する際に使用するデフォルト・パスワード

表 64. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
pollingInterval [1]	int	• **5000** • 任意の正整数	この値は、各セッション内のメッセージ・リスナーのキューに適切なメッセージがない場合に、各メッセージ・リスナーがキューからメッセージの取得を再度試みるまでの最大の時間間隔 (ミリ秒) です。 セッション内のいずれのメッセージ・リスナーでも適切なメッセージがない状態が頻繁に発生する場合は、このプロパティーの値を大きくすることを考えてください。 このプロパティーは、TRANSPORT の値が BIND または CLIENT の場合にのみ使用されます。
port	int	• **1414** • TCP ポート番号	キュー・マネージャーが listen を行うポート。 **connectionNameList** プロパティーが指定されている場合、**hostname** プロパティーと **port** プロパティーはこのプロパティーに置き換えられます。
providerVersion	ストリング	• **指定なし** • 以下のいずれかの形式のストリング – V.R.M.F – V.R.M – V.R – V ここで、V、R、M、および F は、ゼロ以上の整数値です。	MDB が接続することになっているキュー・マネージャーのバージョン、リリース、修正レベルおよびフィックスパック。
queueManager	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	接続するキュー・マネージャーの名前
receiveExit [3]	ストリング	• **null** • コンマで区切られた 1 つ以上の項目から成るストリング。各項目は、IBM MQ classes for Java インターフェース、MQReceiveExit を実装するクラスの完全修飾名です。	チャネル受信出口プログラム、または連続して実行される一連の受信出口プログラムを識別します。
receiveExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル受信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ

表 64. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
rescanInterval [1]	int	• **5000** • 任意の正整数	Point-to-Point ドメインのメッセージ・コンシューマーが、受信するメッセージを選択するためにメッセージ・セレクターを使用する場合、IBM MQ classes for JMS は IBM MQ キューを検索して、そのキューの **MsgDeliverySequence** 属性によって決定される順序で適切なメッセージを探します。 IBM MQ classes for JMS が適切なメッセージを見つけてコンシューマーに配信すると、 IBM MQ classes for JMS は、キュー内の現在位置から次の適切なメッセージの検索を再開します。 IBM MQ classes for JMS は、キューの終わりに到達するまで、またはこのプロパティーの値で決定される時間間隔 (ミリ秒単位) が期限に達するまで、このようにしてキューの検索を続行します。 いずれの場合も、IBM MQ classes for JMS はキューの先頭に戻って検索を続行し、これにより新しい時間間隔が開始します。
securityExit [3]	ストリング	• **null** • IBM MQ classes for Java インターフェース、MQSecurityExit を実装するクラスの完全修飾名	チャネル・セキュリティー出口プログラムを識別します。
securityExitInit	ストリング	• **null** • ユーザー・データのストリング	チャネル・セキュリティー出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ
sendExit [3]	ストリング	• **null** • 1 つ以上の項目をコンマで区切ったストリング。 各項目は、IBM MQ classes for Java インターフェース、MQSendExit を実装するクラスの完全修飾名です。	チャネル送信出口プログラム、または連続して実行される一連の送信出口プログラムを識別します。
sendExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル送信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ

表 64. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
shareConvAllowed	Boolean	• NO-クライアント接続はそのソケットを共用できません。 • **YES** -クライアント接続はそのソケットを共用できます。	チャネル定義が一致する場合に、クライアント接続が、同じプロセスから同じキュー・マネージャーへの他のトップレベル JMS 接続と、ソケットを共有できるかどうか。
sparseSubscriptions [1]	Boolean	• **false** - サブスクリプションが頻繁なマッチング・メッセージを受信する。 • true - サブスクリプションが頻繁でないマッチング・メッセージを受信する。 この値は、サブスクリプション・キューがブラウズ用にオープンできることを必要とします。	TopicSubscriber オブジェクトのメッセージ検索ポリシーを制御する
sslCertStores	ストリング	• **null** • ブランクで区切られた 1 つ以上の LDAP URL のストリング。 各 LDAP URL の形式は次のとおりです。 `ldap://host_name [: port ]` ここで、*host_name* はホスト名または IP アドレス、*port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	TLS 接続で使用する証明書取り消しリスト (CRL) を保持する Lightweight Directory Access Protocol (LDAP) サーバー
sslCipherSuite	ストリング	• **null** • CipherSuite の名前	TLS 接続で使用する CipherSuite
sslFipsRequired [2]	Boolean	• **false** • true	IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を TLS 接続で使用する必要があるかどうか
sslPeerName	ストリング	• **null** • 識別名のテンプレート	TLS 接続で、キュー・マネージャーで提供されるデジタル証明書の識別名を確認するために使用するテンプレート
sslResetCount	int	• **0** • 0 から 999 999 999 の範囲の整数	TLS によって使用された秘密鍵の再ネゴシエーションの前に、TLS 接続で送信および受信したバイトの総数

表 64. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
sslSocketFactory	ストリング	javax.net.ssl.SSLSocketFactory インターフェースのインプリメンテーションを提供するクラスの、完全修飾クラス名を表すストリング。コンストラクター・メソッドに渡される引数を、括弧内に含めることもできます。	管理されたオブジェクトの有効範囲内で設定されたすべての接続は、SSLSocketFactory インターフェースのインプリメンテーションから得られたソケットを使用する。
statusRefreshInterval [1]	int	• **60000** • 任意の正整数	サブスクライバーがキュー・マネージャーとの接続を失ったときに検出する長期実行トランザクションのリフレッシュの間隔 (ミリ秒)。 このプロパティーは、**subscriptionStore** の値が QUEUE の場合にのみ使用されます。
subscriptionStore [1]	ストリング	• **ブローカー** • MIGRATE • QUEUE	IBM MQ classes for JMS がアクティブ・サブスクリプションに関する永続データを保管する場所を決定します。

表 64. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
transportType	ストリング	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	キュー・マネージャーへの接続で、クライアント・モードまたはバインディング・モードのどちらを使用するか。BINDINGS_THEN_CLIENT という値が指定されると、リソース・アダプターはまずバインディング・モードで接続を試みます。 この接続試行が失敗すると、リソース・アダプターはクライアント・モード接続を確立しようとします。 z/OS WebSphere Application Server for z/OS システム上で実行されているアクティベーション・スペックが BINDINGS_THEN_CLIENT トランスポート・モードを使用するように構成されていて、以前に確立した接続が切断されると、アクティベーション・スペックによって試行される再接続では最初に BINDINGS トランスポート・モードが使用されます。BINDINGS トランスポート・モード接続が成功しないと、その後、アクティベーション・スペックは CLIENT トランスポート・モード接続を試行します。
ユーザー名	ストリング	• **null** • ユーザー名	キュー・マネージャーへの接続を作成する際に使用するデフォルト・ユーザー名
wildcardFormat	ストリング	• CHAR- ブローカー・バージョン 1 で使用された文字ワイルドカードのみ認識します。 • **TOPIC** - ブローカー・バージョン 2 で使用されたトピック・レベル・ワイルドカードのみ認識します。	使用されるワイルドカード構文のバージョン

注:

1. このプロパティーは、IBM MQ classes for JMS のバージョン 70 で使用できます。
2. sslFipsRequired プロパティーの使用の重要な詳細については、441 ページの『IBM MQ リソース・アダプターの制限』を参照してください。
3. 出口を見つけることができるようにリソース・アダプターを構成する方法については、282 ページの『チャネル出口を使用するように IBM MQ classes for JMS を構成する』を参照してください。

4. V 9.3.0 dynamicallyBalanced プロパティーは、XA トランザクション・サポートと併用できません。 dynamicallyBalanced が「true」の場合、MDB では XA トランザクションが無効になるように構成する必要があります。

JMS 接続コンシューマーを作成するために使用されるプロパティー

注 : destination および **destinationType** は、明示的に定義する必要があります。 463 ページの表 65 の他のプロパティーはすべてオプションです。

表 *65. JMS* 接続コンシューマーを作成するために使用される *ActivationSpec* オブジェクトのプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
destination	ストリング	宛先名	メッセージを受信する宛先。 **useJNDI** プロパティーは、このプロパティーの値がどのように解釈されるのかを判別します。
destinationLookup	ストリング	• **null** • Destination オブジェクトの JNDI 名	このプロパティーが設定されると、ActivationSpec は、アプリケーション・サーバーの JNDI 名前空間で指定された JNDI 名の JMS Destination オブジェクトを検索し、ActivationSpec で指定されているプロパティーに優先して、対象オブジェクトのプロパティーを使用して JMS 接続コンシューマーを作成します。 詳しくは、466 ページの『ActivationSpec の connectionFactoryLookup プロパティーと destinationLookup プロパティー』 を参照してください。
destinationType	ストリング	• V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) • V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	宛先のタイプ。キュー、またはトピック。
maxMessages	int	• **1** • 正整数	サーバー・セッションに一度に割り当てることができるメッセージの最大数。アクティベーション・スペックが XA トランザクションで MDB にメッセージを送達する場合、このプロパティーの設定に関係なく、値 1 が使用されます。
maxPoolDepth	int	• **10** • 正整数	接続コンシューマーによって使用されるサーバー・セッション・プール内のサーバー・セッションの最大数
messageSelector	ストリング	• **null** • SQL92 メッセージ・セレクター式	送達されるメッセージを指定するメッセージ・セレクター式

表 65. JMS 接続コンシューマーを作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
nonASFTimeout	int	• **0** • 正整数	正の値は、非 ASF 送達が使用されることを示します。 この値は、読み取り要求で、まだ到着していない可能性のあるメッセージを待機する時間 (ミリ秒単位) です (待機を伴う読み取りの呼び出し)。デフォルト値 0 は ASF 送達が使用されることを示します。 このパラメーターは、以下の場合に有効です。 • アプリケーションが WebSphere Application Server 7.0 以降で実行されている。 • アプリケーションは、適切なレベルの wmqJms クライアント機能を使用して WebSphere Liberty で実行されています。 詳しくは、442 ページの『Liberty と IBM MQ リソース・アダプター』を参照してください。
nonASFRollbackEnabled	Boolean	• **false** - MDB で障害が起きても、メッセージはコンシュームされます。 • true - MDB 内で障害が起きたときは、メッセージがキューにロールバックされます。	MDB が非トランザクションである場合に、メッセージ送達が IBM MQ 同期点内で行われるかどうかを指定します。 MDB がトランザクションであるか、または **nonASFTimeout** が 0 に設定されている場合は、無視されます。
poolTimeout	int	• **300000** • 正整数	未使用のサーバー・セッションが、非アクティブ状態が原因でクローズされる前に、サーバー・セッション・プールでオープンしたままになっている時間 (ミリ秒)
readAheadAllowed	int	• **DESTINATION** - キュー定義またはトピック定義を参照することによって先読みが許可されるかどうかを判別します。 • DISABLED - 先読みは許可されない • ENABLED - 先読みは許可される • QUEUE - キュー定義を参照することによって先読みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって先読みが許可されるかどうかを判別します。	破壊的なコンシュームが原因でサーバー・セッションにハンドオフされる前に、アクティベーション・スペック・ブラウズ・スレッドが先読みを使用して、この宛先から内部バッファーへの複数のメッセージをブラウズすることが許可されるかどうか。 **注 :** 先読みを有効にすると、MDB の処理速度が宛先からのメッセージのブラウズ速度に追いつかない場合に、JMSCC0108 メッセージの増加、またはパフォーマンスの低下、あるいはその両方につながります。

表 65. *JMS* 接続コンシューマーを作成するために使用される *ActivationSpec* オブジェクトのプロパティー *(続き)*

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
readAheadClosePolicy	int	• **ALL** - 内部先読みバッファー内のすべてのメッセージは、停止する前に MDB に送信されます。 • CURRENT - 現行の MDB 呼び出しのみが完了します。内部先読みバッファー内にメッセージが残される可能性がありますが、それらは破棄されます。	管理者が MDB を停止すると、内部先読みバッファー内のメッセージに何が生じるか。
receiveCCSID	int	• **0** - JVM `Charset.defaultCharset` を使用します。 • 1208 - UTF-8 • サポートされるコード化文字セット ID	キュー・マネージャー・メッセージ変換のターゲット CCSID を 設定する宛先プロパティー。 **receiveConversion** が QMGR に設定されていない場合、値は無視されます。
receiveConversion	ストリング	• **CLIENT_MSG** • QMGR	キュー・マネージャーによりデータ変換を実行するかどうかを決定する宛先プロパティー。
sharedSubscription	Boolean	• **False** - MDB はサブスクリプションを共有サブスクリプションとして開いてはなりません。 • True-MDB はサブスクリプションを共有サブスクリプションとして開く必要があります (JMS 2.0 が暗黙指定する規則を使用します。 Java.net の JMS 2.0 仕様を参照してください)。	MDB が共有サブスクリプションから駆動される方法を制御します。 このプロパティーの使用方法について詳しくは、469 ページの『sharedSubscription プロパティーの定義方法を示す例』を参照してください。
startTimeout	int	• **10 000** • 正整数	MDB へのメッセージの送達処理がスケジュールされた後でその送達を開始しなければならない時間 (ミリ秒)。 この時間が経過すると、メッセージはキュー上にロールバックされます。
subscriptionDurability	ストリング	• **NonDurable** - 非永続サブスクリプションが、トピックにサブスクライブする MDB にメッセージを送達するために使用されます。 • Durable - 永続サブスクリプションが、トピックにサブスクライブする MDB にメッセージを送達するために使用されます。	トピックにサブスクライブする MDB にメッセージを送達するために、永続サブスクリプションが使用されるか、それとも非永続サブスクライブが使用されるか
subscriptionName	ストリング	• **"" (空ストリング)** • サブスクリプション名	永続サブスクリプションの名前

表 65. JMS 接続コンシューマーを作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
useJNDI	Boolean	• **false** - destination というプロパティーが、IBM MQ キューまたはトピックの名前として解釈されます。 • true-destination というプロパティーは、アプリケーション・サーバーの JNDI 名前空間にある以下のいずれかのオブジェクトの名前として解釈されます。 – V 9.3.0 V 9.3.0 jakarta.jms.Queue (Jakarta Messaging 3.0) – V 9.3.0 V 9.3.0 jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated destination というプロパティーの値がどのように解釈されるのかを決定します。 **注 :** このプロパティーは、IBM MQ 9.0 では非推奨です。 代わりに、destinationLookup プロパティーを使用してください。

プロパティーの競合と依存関係

ActivationSpec オブジェクトは、競合するプロパティーを持つことができます。 例えば、バインディング・モードの接続に TLS プロパティーを指定できます。 この場合、振る舞いは移送タイプおよびメッセージング・ドメインによって決定されます。これは、**destinationType** プロパティーで判別された Point-to-Point かパブリッシュ/サブスクライブのいずれかです。 指定された移送タイプまたはメッセージング・ドメインに該当しないプロパティーは無視されます。

あるプロパティーを定義する際にその他のプロパティーを定義する必要があるが、それらの他のプロパティーを定義していない場合、ActivationSpec オブジェクトは、MDB のデプロイメント時にその validate() メソッドが呼び出される際に InvalidPropertyException 例外をスローします。 例外は、アプリケーション・サーバーに応じた仕方で、アプリケーション・サーバーの管理者に報告されます。 例えば、subscriptionDurability プロパティーを Durable (永続サブスクリプションを使用することを示す) に設定する場合、**subscriptionName** プロパティーも定義する必要があります。

ccdtURL および **channel** というプロパティーが両方とも定義されている場合、InvalidPropertyException 例外がスローされます。 ただし、**ccdtURL** プロパティーのみを定義し、**channel** というプロパティーはデフォルト値の SYSTEM.DEF.SVRCONN、例外はスローされず、**ccdtURL** プロパティーによって識別されるクライアント・チャネル定義テーブルが JMS 接続の開始に使用されます。

ActivationSpec の connectionFactoryLookup プロパティーと destinationLookup プロパティー

JMS 2.0 仕様では、2 つの新しい ActivationSpec プロパティーが導入されました。connectionFactoryLookup および destinationLookup プロパティーを管理対象オブジェクトの JNDI 名を指定して設定し、その他の ActivationSpec プロパティーより優先して使用させることができます。

例えば、接続ファクトリーが JNDI で定義されており、アクティベーション・スペックの connectionFactory ルックアップ・プロパティーでそのオブジェクトの JNDI 名が指定されている場合、 JNDI で定義されている接続ファクトリーのすべてのプロパティーが、 453 ページの表 64 のプロパティーよりも優先して使用されます。

宛先が JNDI で定義されており、 JNDI 名が ActivationSpec の destinationLookup プロパティーで設定されている場合、その値が 463 ページの表 65 の値よりも優先して使用されます。 これらの 2 つのプロパティーを使用する方法について詳しくは、466 ページの『ActivationSpec の connectionFactoryLookup プロパティーと destinationLookup プロパティー』を参照してください。

これら 2 つのプロパティーを使用して、 453 ページの表 64 および 463 ページの表 65 で定義されている ActivationSpec のプロパティーよりも優先して使用される ConnectionFactory および Destination オブジェクトの JNDI 名を指定することができます。

これらのプロパティーの機能について詳述した、以下の点に留意することは重要です。

connectionFactoryLookup
JNDI から検索される ConnectionFactory は、453 ページの表 64 にリストされたプロパティーのソースとして使用されます。 ConnectionFactory オブジェクトは JMS 接続を実際に作成するためには使用されず、オブジェクトのプロパティーのみが照会されます。 ConnectionFactory のこれらのプロパティーは、ActivationSpec で定義されているプロパティーをオーバーライドします。 これには 1 つの例外があります。 ActivationSpec に **ClientID** プロパティーが設定されている場合、このプロパティーの値は ConnectionFactory で指定されている値をオーバーライドします。 これは、単一の ConnectionFactory を複数の ActivationSpec で使用するというのが一般的なシナリオであるためです。 これにより、管理が簡単になります。 ただし、JMS 2.0 仕様では、ConnectionFactory に基づいて作成されるすべての JMS 接続の **ClientID** は固有でなければならないと定められています。 このため、ActivationSpec では ConnectionFactory で設定されているどの値もオーバーライドできなければなりません。 **ClientID** が ActivationSpec に設定されていない場合は、接続ファクトリーにある値が使用されます。

destinationLookup
Destination プロパティーと **UseJndi** プロパティーは、ActivationSpec で定義されます。 **UseJndi** フラグが `true` に設定されている場合、宛先プロパティーで指定されたテキストが JNDI 名であると見なされ、その JNDI 名の宛先オブジェクトが JNDI で検索されます。

destinationLookup プロパティーは、まったく同じように機能します。 設定すると、このプロパティーで指定された JNDI 名の宛先オブジェクトが JNDI で検索されます。 このプロパティーは **useJNDI** プロパティーより優先されます。

Deprecated useJNDI プロパティーは、 IBM MQ 9.0 では推奨されません。これは、**destinationLookup** プロパティーが JMS 2.0 仕様であるか、またはそれ以降で同じ機能を実行することに相当するためです。

IBM MQ classes for JMS で同等の機能を持たない ActivationSpec プロパティー

ActivationSpec オブジェクトのプロパティーのほとんどは、 IBM MQ classes for JMS オブジェクトまたは IBM MQ classes for Jakarta Messaging オブジェクトのプロパティー、あるいは IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging メソッドのパラメーターに相当します。 ただし、 IBM MQ classes for JMS または IBM MQ classes for Jakarta Messaging には、3 つのチューニング・プロパティーと 1 つのユーザビリティー・プロパティーに相当するものがありません。

startTimeout
リソース・アダプターがメッセージを MDB に送達するように Work オブジェクトをスケジュールした後で、アプリケーション・サーバーの作業マネージャーがリソースが使用可能になるのを待機する時間 (ミリ秒)。 メッセージの送達が開始する前にこの時間が経過する場合、Work オブジェクトはタイムアウトし、メッセージはキュー上にロールバックされ、リソース・アダプターはメッセージの送達をもう一度試行できます。 診断トレースが使用可能になっている場合、警告は診断トレースに書き込まれますが、使用可能になっていない場合、メッセージの送達処理に影響を与えません。 アプリケーション・サーバーの負荷が非常に高くなっているときに限り、こうした条件が発生することを予期できます。 この条件が定期的に発生する場合、このプロパティーの値を大きくして、作業マネージャーでメッセージ送達のスケジュール時間を長くすることを考慮してください。

maxPoolDepth
接続コンシューマーによって使用されるサーバー・セッション・プール内のサーバー・セッションの最大数。 サーバー・セッションが作成されると、キュー・マネージャーとの会話が開始されます。 接続コンシューマーはサーバー・セッションを使用して、メッセージを MDB に送達します。 プールの深さ

を深くすると、高容量の状態で同時に送達されるメッセージの数も増えますが、使用するアプリケーション・サーバーのリソースも増えることになります。 多数の MDB がデプロイされる場合、アプリケーション・サーバー上で管理可能なレベルで負荷を維持するには、プールの深さを浅くすることを考慮してください。 各接続コンシューマーはその独自のサーバー・セッション・プールを使用するため、このプロパティーはすべての接続コンシューマーに使用可能なサーバー・セッションの総数を定義するわけではないことに注意してください。

poolTimeout
未使用のサーバー・セッションが、非アクティブ状態が原因でクローズされる前に、サーバー・セッション・プールでオープンしたままになっている時間 (ミリ秒)。 メッセージ・ワークロードの一時的な増加により、負荷を分散させるために追加のサーバー・セッションが作成されます。しかし、メッセージ・ワークロードが標準に戻った後も、追加のサーバー・セッションはプール内に残り、使用されません。

サーバー・セッションが使用されるたびに、タイム・スタンプでマークが付けられます。 スカベンジャー・スレッドは定期的に、各サーバー・セッションがこのプロパティーで指定された期間内に使用されているか確認します。 サーバー・セッションが使用されていない場合、それはクローズされ、サーバー・セッション・プールから除去されます。 指定された期間が経過した直後にサーバー・セッションがクローズされないことがあります。 このプロパティーは、除去される前に非アクティブ状態になる最小期間を表すからです。

useJNDI
このプロパティーの説明については、463 ページの表 65 を参照してください。

MDB のデプロイ

MDB をデプロイするには、まず、MDB が必要とするプロパティーを指定して ActivationSpec オブジェクトのプロパティーを定義します。 以下の例は、明示的に定義される代表的なプロパティー・セットです。

JM 3.0 V 9.3.0 V 9.3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

アプリケーション・サーバーはプロパティーを使用して、ActivationSpec オブジェクトを作成し、それは MDB に関連付けられます。 ActivationSpec オブジェクトのプロパティーにより、メッセージが MDB に送達される方法が決定されます。 MDB が分散トランザクションを必要とするが、リソース・アダプターが分散トランザクションをサポートしていない場合、MDB のデプロイメントは失敗します。 分散トランザクションがサポートされるようにリソース・アダプターをインストールする方法については、444 ページの『IBM MQ リソース・アダプターのインストール』 を参照してください。

複数の MDB が同じ宛先からメッセージを受信する場合、Point-to-Point ドメインで送信されたメッセージは、他の MDB がメッセージを受信する資格があるとしても 1 つの MDB にのみ受信されます。 特に、2 つの MDB が異なるメッセージ・セレクターを使用しており、着信メッセージが両方のメッセージ・セレクターに一致する場合、1 つの MDB のみメッセージを受信します。 メッセージを受信するように選ばれた MDB は未定義で、特定の MDB がメッセージを受信するようには指定できません。 パブリッシュ/サブスクライブ・ドメインで送信されたメッセージは資格があるすべての MDB によって受信されます。

ある環境では、MDB に送達されたメッセージが IBM MQ キューにロールバックされることがあります。 例えば、メッセージが後でロールバックされる作業単位内で送達される場合は、ロールバックが起こります。 ロールバックされるメッセージは再度送達されますが、不良フォーマットのメッセージは、MDB が繰り返し失敗する原因になるため、送達できません。 こうしたメッセージは有害メッセージと呼ばれます。 IBM MQ classes for JMS が今後の調査のために有害メッセージを別のキューに自動的に転送したり、メッセージを破棄したりするように、 IBM MQ を構成することができます。

有害メッセージの処理方法の詳細については、236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

関連概念

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

関連タスク

WebSphere Application Server での JMS リソースの構成

sharedSubscription プロパティーの定義方法を示す例

WebSphere Liberty `server.xml` ファイル内でアクティベーション・スペックの sharedSubscription プロパティーを定義することができます。 あるいは、アノテーションを使用してメッセージ駆動型 Bean (MDB) 内でプロパティーを定義できます。

例: Liberty `server.xml` ファイル内での定義

WebSphere Liberty `server.xml` ファイル内で、以下の例に示すようにアクティベーション・スペックを定義します。 この例では、localhost/ポート 1490 のキュー・マネージャーに永続的な共有サブスクリプションを作成します。

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

例: MDB 内での定義

次の例に示すように、アノテーションを使用して MDB 内で sharedSubscription プロパティーを定義することもできます。

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

以下の例は、アノテーション方式を使用する MDB コードの一部を示します。

JM 3.0 V 9.3.0 V 9.3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
```

```
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

関連概念

サブスクライバーとサブスクリプション

サブスクリプション永続性

326 ページの『複製サブスクリプションおよび共用サブスクリプション』

IBM MQ 8.0 以降では、同じサブスクリプションへのアクセス権限を複数のコンシューマーに付与する方法が 2 つあります。 それらは、クローン・サブスクリプションを使用する方法と共用サブスクリプションを使用する方法です。

アウトバウンド通信のリソース・アダプターの構成

アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

アウトバウンド通信の使用例

アウトバウンド通信を使用する場合、アプリケーション・サーバーで実行するアプリケーションはキュー・マネージャーへの接続を開始し、それから同期的な方法でメッセージをそのキューに送信し、そのキューからメッセージを受信します。 例えば、以下のサーブレット・メソッド、doGet() はアウトバウンド通信を使用します。

JM 3.0 V 9.3.0 V 9.3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer
```

```
     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

サーブレットが HTTP GET 要求を受信すると、ConnectionFactory オブジェクトおよび Queue オブジェクトを JNDI ネーム・スペースから検索し、それらのオブジェクトを使用してメッセージを IBM MQ キューに送信します。 次いで、サーブレットは送信したメッセージを受信します。

アウトバウンド通信に必要なリソース

アウトバウンド通信を構成するには、以下のカテゴリーの Java EE Connector Architecture (JCA) リソースを定義してください。

- ConnectionFactory オブジェクトのプロパティー。アプリケーション・サーバーが JMS ConnectionFactory オブジェクトを作成するために使用します。
- 管理対象宛先オブジェクトのプロパティー。アプリケーション・サーバーが JMS Queue オブジェクトまたは JMS Topic オブジェクトを作成するために使用します。

これらのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。 アプリケーション・サーバーによって作成された ConnectionFactory、Queue、お

よび Topic オブジェクトは、JNDI ネーム・スペースにバインドされ、アプリケーションはそのネーム・スペースからオブジェクトを検索できます。

通常、アプリケーションが接続する必要があるキュー・マネージャーごとに 1 つの ConnectionFactory オブジェクトを定義します。 アプリケーションが Point-to-Point ドメインでアクセスする必要があるキューごとに 1 つの Queue オブジェクトを定義します。 そして、アプリケーションがパブリッシュまたはサブスクライブするトピックごとに 1 つの Topic オブジェクトを定義します。 ConnectionFactory オブジェクトはドメインに依存しないようにすることができます。 あるいは、ドメイン固有にすることもできます。 Point-to-Point ドメインの場合は QueueConnectionFactory オブジェクト、パブリッシュ/サブスクライブ・ドメインの場合は TopicConnectionFactory オブジェクトです。

ヒント : JMS 2.0 では、接続ファクトリーは、接続とコンテキストの両方の作成に使用できます。 そのため、接続とコンテキストの両方が混在する接続ファクトリーを接続プールに関連付けることが可能です。 1 つの接続ファクトリーでは、接続だけを作成するか、あるいはコンテキストだけを作成することをお勧めします。 このようにすると、その接続ファクトリーの接続プールには 1 種類のオブジェクトのみが含まれるようになるので、プールの効率が向上します。

ConnectionFactory オブジェクトのプロパティー

472 ページの表 66 は、ConnectionFactory オブジェクトのプロパティーをリストしています。 アプリケーション・サーバーは、これらのプロパティーを使用して、JMS ConnectionFactory オブジェクトを作成します。

表 *66. ConnectionFactory* オブジェクトのプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
applicationName	ストリング	• 起動クラス名が使用可能である場合は、28 文字以内に調整されます。 使用可能でない場合は、ストリング `WebSphere MQ Client for Java` が使用されます。	アプリケーションをキュー・マネージャーに登録する際に使用した名前。 このアプリケーション名は、**DISPLAY CONN MQSC/PCF** コマンド (フィールド名は **APPLTAG**) または IBM MQ Explorer の 「**アプリケーション接続**」画面 (フィールド名は **App name**) に表示されます。
brokerCCSub キュー	ストリング	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが非永続サブスクリプション・メッセージを受信するキューの名前。
brokerControl キュー	ストリング	• **SYSTEM.BROKER.CONTROL.QUEUE** • キュー名	ブローカー制御キューの名前。
brokerPub キュー	ストリング	• **SYSTEM.BROKER.DEFAULT.STREAM** • キュー名	パブリッシュ済みメッセージが送信されたキュー (ストリーム・キュー) の名前。
brokerQueue マネージャー	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	ブローカーが稼働しているキュー・マネージャーの名前。
brokerSub キュー	ストリング	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • キュー名	非永続メッセージ・コンシューマーがメッセージを受信するキューの名前。 詳細については、BROKERSUBQ プロパティーを参照してください。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
brokerVersion	ストリング	• **指定なし** - ブローカーを V6 から V7 に移行した後、このプロパティーを設定し、RFH2 ヘッダーが使用されないようにします。 移行した後、このプロパティーは関係なくなります。 • **V1** - IBM MQ パブリッシュ/サブスクライブ・ブローカーを使用する場合。 この値は、TRANSPORT が BIND または CLIENT に設定されている場合のデフォルト値です。 • **V2** - IBM Integration Bus のブローカーをネイティブ・モードで使用する場合。この値は、TRANSPORT が DIRECT または DIRECTHTTP に設定されている場合のデフォルト値です。	使用されているブローカーのバージョン。
ccdtURL	ストリング	• **null** • Uniform Resource Locator (URL)	クライアント・チャネル定義テーブル (CCDT) が格納されているファイルの名前と場所を識別し、このファイルのアクセス方法を指定する URL。
CCSID	ストリング	• **819** • Java 仮想マシン (JVM) でサポートされるコード化文字セット ID	接続用のコード化文字セット ID。
channel	ストリング	• **SYSTEM.DEF.SVRCONN** • MQI チャネルの名前	使用する MQI チャネルの名前。
cleanupInterval	int	• **3 600 000** • 正整数	パブリッシュ/サブスクライブ・クリーンアップ・ユーティリティーがバックグラウンドで実行する間隔 (ミリ秒)。
cleanupLevel	ストリング	• **SAFE** • NONE • strong • FORCE • NONDUR	ブローカー・ベースのサブスクリプション・ストアのクリーンアップ・レベル。
clientID	ストリング	• **null** • クライアント ID	接続のクライアント ID。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
cloneSupport	ストリング	• **DISABLED** - 一度に稼働できる永続トピック・サブスクライバーのインスタンスは 1 つだけです。 • ENABLED - 1 つの永続トピック・サブスクライバーの 2 つ以上のインスタンスを同時に稼働できますが、各インスタンスを別々の Java 仮想マシン (JVM) で稼働させる必要があります。	1 つの永続トピック・サブスクライバーの複数のインスタンスを同時に稼働できるかどうか。
connectionNameList	ストリング	• **localhost(1414)** • コンマで区切られている項目で構成されるストリングです。各項目は以下の形式になります。 `HOSTNAME(PORT)` ここで、*HOSTNAME* は DNS 名または IP アドレスです。	アウトバウンド通信に使用される TCP/IP 接続名のリスト。 **hostname** プロパティーおよび **port** プロパティーは、**connectionNameList** に置き換えられます。 このプロパティーを使用して、複数インスタンス・キュー・マネージャーに再接続します。 **connectionNameList** は **localAddress** の形式に類似していますが、混同しないでください。**localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
failIfQuiesce	Boolean	• **true** • false	キュー・マネージャーが静止状態の場合、特定のメソッドの呼び出しが失敗するかどうか。
headerCompression	ストリング	• **NONE** • SYSTEM - RLE メッセージ・ヘッダーの圧縮が実行される	接続のヘッダー・データを圧縮するために使用できる技法のリスト。
hostName	ストリング	• **localhost** • ホスト名 • IP アドレス	キュー・マネージャーが存在しているシステムのホスト名または IP アドレス。 **connectionNameList** プロパティーが指定されている場合、**hostname** プロパティーと **port** プロパティーはこのプロパティーに置き換えられます。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
localAddress	ストリング	• **null** • 以下の形式のストリング `[ host_name ][(low_port [, high_port ])]` ここで、*host_name* はホスト名または IP アドレス、*low_port* および *high_port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	キュー・マネージャーとの接続場合、このプロパティーは以下のいずれか、または両方を指定します。 • 使用されるローカル・ネットワーク・インターフェース • 使用されるローカル・ポート、またはローカル・ポートの範囲 **localAddress** は **connectionNameList** の形式に類似していますが、混同しないでください。 **localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
messageCompression	ストリング	• **NONE** • ブランク文字で区切られた以下の値の 1 つ以上のリスト。 RLE ZLIBFAST ZLIBHIGH	接続のメッセージ・データを圧縮するために使用できる技法のリスト。
messageSelection	ストリング	• **CLIENT** • BROKER	メッセージ選択を IBM MQ classes for JMS またはブローカーのどちらが行うかを決定します。 ブローカーによるメッセージ選択は、**brokerVersion** の値が 1 の場合、サポートされません。
パスワード	ストリング	• **null** • パスワード	キュー・マネージャーへの接続を作成する際に使用するデフォルト・パスワード。
pollingInterval	int	• **5000** • 任意の正整数	この値は、各セッション内のメッセージ・リスナーのキューに適切なメッセージがない場合に、各メッセージ・リスナーがキューからメッセージの取得を再度試みるまでの最大の時間間隔 (ミリ秒) です。 セッション内のいずれのメッセージ・リスナーでも適切なメッセージがない状態が頻繁に発生する場合は、このプロパティーの値を大きくすることを考えてください。 このプロパティーは、**TRANSPORT** の値が BIND または CLIENT の場合にのみ使用されます。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
port	int	• **1414** • TCP ポート番号	キュー・マネージャーが listen を行うポート。 **connectionNameList** プロパティーが指定されている場合、**hostname** プロパティーと **port** プロパティーはこのプロパティーに置き換えられます。
providerVersion	ストリング	• **指定なし** • 以下のいずれかの形式のストリング – V.R.M.F – V.R.M – V.R – V ここで、V、R、M、および F は、ゼロ以上の整数値です。	アプリケーションが接続することになっているキュー・マネージャーのバージョン、リリース、修正レベル、およびフィックスパック。
pubAck 間隔	int	• **25** • 正整数	IBM MQ classes for JMS がブローカーからの確認通知を要求するまでに、パブリッシャーによって公開されるメッセージの数。
queueManager	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	接続するキュー・マネージャーの名前。
receiveExit [3]	ストリング	• **null** • コンマで区切られた 1 つ以上の項目から成るストリング。各項目は、IBM MQ classes for Java インターフェース、MQReceiveExit を実装するクラスの完全修飾名です。	チャネル受信出口プログラム、または連続して実行される一連の受信出口プログラムを識別します。
receiveExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル受信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
rescanInterval	int	• **5000** • 任意の正整数	Point-to-Point ドメインのメッセージ・コンシューマーがメッセージ・セレクターを使用して、受信するメッセージを選択する場合、IBM MQ classes for JMS は IBM MQ キューを検索して、このキューの **MsgDeliverySequence** 属性によって決定される順序で適切なメッセージを探します。 IBM MQ classes for JMS が適切なメッセージを見つけてコンシューマーに配信すると、IBM MQ classes for JMS は、キュー内の現在位置から次の適切なメッセージの検索を再開します。 IBM MQ classes for JMS は、キューの終わりに到達するまで、またはこのプロパティーの値で決定される時間間隔 (ミリ秒単位) が期限に達するまで、このようにしてキューの検索を続行します。 いずれの場合も、IBM MQ classes for JMS はキューの先頭に戻って検索を続行し、これにより新しい時間間隔が開始します。
securityExit [3]	ストリング	• **null** • IBM MQ classes for Java インターフェース、MQSecurityExit を実装するクラスの完全修飾名	チャネル・セキュリティー出口プログラムを識別します。
securityExitInit	ストリング	• **null** • ユーザー・データのストリング	チャネル・セキュリティー出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。
sendCheckCount	int	• **0** • 任意の正整数	単一の未処理 JMS セッション内で、非同期書き込みエラーの検査が行われてから次の検査が行われるまでに許可される送信呼び出しの数。
sendExit [3]	ストリング	• **null** • 1 つ以上の項目をコンマで区切ったストリング。 各項目は、IBM MQ classes for Java インターフェース、MQSendExit を実装するクラスの完全修飾名です。	チャネル送信出口プログラム、または連続して実行される一連の送信出口プログラムを識別します。
sendExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル送信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
shareConvAllowed	Boolean	• NO-クライアント接続はそのソケットを共用できません。 • **YES** -クライアント接続はそのソケットを共用できます。	チャネル定義が一致する場合に、クライアント接続が、同じプロセスから同じキュー・マネージャーへの他のトップレベル JMS 接続と、ソケットを共有できるかどうか。
sparseSubscriptions	Boolean	• **false** - サブスクリプションが頻繁なマッチング・メッセージを受信する。 • true - サブスクリプションが頻繁でないマッチング・メッセージを受信する。この値は、サブスクリプション・キューがブラウズ用にオープンできることを必要とします。	TopicSubscriber オブジェクトのメッセージ検索ポリシーを制御する。
sslCertStores	ストリング	• **null** • ブランクで区切られた 1 つ以上の LDAP URL のストリング。 各 LDAP URL の形式は次のとおりです。 `ldap://host_name [: port ]` ここで、*host_name* はホスト名または IP アドレス、*port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	TLS 接続で使用する証明書取り消しリスト (CRL) を保持する Lightweight Directory Access Protocol (LDAP) サーバー。
sslCipherSuite	ストリング	• **null** • CipherSuite の名前	TLS 接続で使用する CipherSuite。
sslFipsRequired [2]	Boolean	• **false** • true	IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を TLS 接続で使用する必要があるかどうか。
sslPeerName	ストリング	• **null** • 識別名のテンプレート	TLS 接続で、キュー・マネージャーで提供されるデジタル証明書の識別名を確認するために使用するテンプレート。
sslResetCount	int	• **0** • 0 から 999 999 999 の範囲の整数	TLS によって使用された秘密鍵の再ネゴシエーションの前に、TLS 接続で送信および受信したバイトの総数。
sslSocketFactory	ストリング	javax.net.ssl.SSLSocketFactory インターフェースのインプリメンテーションを提供するクラスの、完全修飾クラス名を表すストリング。 コンストラクター・メソッドに渡される引数を、括弧内に含めることもできます。	管理された宛先オブジェクトの有効範囲内で設定されたすべての接続は、SSLSocketFactory インターフェースのインプリメンテーションから得られたソケットを使用します。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
statusRefresh 間隔	int	• **60000** • 任意の正整数	サブスクライバーがキュー・マネージャーとの接続を失ったときに検出する長期実行トランザクションのリフレッシュの間隔 (ミリ秒)。このプロパティーは、**SUBSTORE** の値が QUEUE の場合にのみ使用されます。
subscriptionStore	ストリング	• **ブローカー** • MIGRATE • QUEUE	IBM MQ classes for JMS がアクティブ・サブスクリプションに関する永続データを保管する場所を決定します。
targetClientMatching	Boolean	• **true** • false	着信メッセージの JMSReplyTo ヘッダー・フィールドで識別されるキューに送信された応答メッセージに MQRFH2 ヘッダーがあるかどうか (着信メッセージに MQRFH2 ヘッダーがある場合のみ)。 このプロパティーは、アクティベーション・スペックのために構成することもできます。 詳細については、488 ページの『アクティベーション・スペックの targetClientMatching プロパティーの構成』を参照してください。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
temporaryModel	ストリング	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • 任意のストリング	JMS 一時キューの作成に使用するモデル・キューの名前。 以下の両方のステートメントが true の場合は、SYSTEM.DEFAULT.MODEL.QUEUE を使用します。 • アプリケーションが非永続メッセージを受け入れる一時キューを使用する。 • ConnectionFactory が指すキュー・マネージャーの一時キューは、一度に 1 つのアプリケーションによってのみ作成される。SYSTEM.DEFAULT.MODEL.QUEUE を開くことができるのは、一度に 1 つのアプリケーションによってのみであることに注意してください。 以下の状態では SYSTEM.JMS.TEMPQ.MODEL を使用します。 • アプリケーションが永続メッセージを受け入れる一時キューを使用する場合。 • 複数のアプリケーションが ConnectionFactory が指すキュー・マネージャーに接続でき、それらのアプリケーションで同時に一時キューを作成する必要がある場合。 以下の状態では、**DEFPSIST** 属性を YES に設定し、**DEFSOPT** 属性を SHARED に設定して新規モデル・キューを定義します。 • アプリケーションが非永続メッセージを受け入れる一時キューを使用し、複数のアプリケーションが ConnectionFactory が指すキュー・マネージャーに接続し、それらのアプリケーションで同時に一時キューを作成する必要がある場合。 新規モデル・キューが作成されたら、**temporaryModel** プロパティーを新規モデル・キューの名前に設定します。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
tempQPrefix	ストリング	• **"" (空ストリング)** • IBM MQ 動的キューの名前を形成するために使用できる接頭句。 接頭部を形成するための規則は、IBM MQ オブジェクト記述子、構造体 MQOD 内の **DynamicQName** フィールドの内容を形成するための規則と同じですが、最後の非ブランク文字はアスタリスク (*) でなければなりません。 プロパティーの値が空ストリングの場合、 IBM MQ classes for JMS は値 AMQ.* を使用します。 動的キューの作成時。	IBM MQ 動的キューの名前を形成するために使用された接頭部。
tempTopicPrefix	ストリング	IBM MQ トピック・ストリングに有効な文字だけで構成される非ヌル・ストリング	一時トピックを作成すると、 JMS は「TEMP/*TEMPTOPICPREFIX*/*unique_id* 」という形式のトピック・ストリングを生成するか、このプロパティーがデフォルト値のままである場合は「TEMP/*unique_id*」のみを生成します。 空でない **TEMPTOPICPREFIX** を指定すると、この接続の下で作成された一時トピックに対して、 サブスクライバー用の管理されたキューを作成するための特定のモデル・キューを定義できます。

表 66. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
transportType	ストリング	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	キュー・マネージャーへの接続で、クライアント・モードまたはバインディング・モードのどちらを使用するか。 BINDINGS_THEN_CLIENT という値が指定されると、リソース・アダプターはまずバインディング・モードで接続を試みます。 この接続が失敗すると、リソース・アダプターはクライアント・モード接続の確立を試行します。 z/OS WebSphere Application Server for z/OS システム上で実行されているアクティベーション・スペックが BINDINGS_THEN_CLIENT トランスポート・モードを使用するように構成されていて、以前に確立した接続が切断されると、アクティベーション・スペックによって試行される再接続では最初に BINDINGS トランスポート・モードが使用されます。 BINDINGS トランスポート・モード接続が成功しないと、その後、アクティベーション・スペックは CLIENT トランスポート・モード接続を試行します。
ユーザー名	ストリング	• **null** • ユーザー名	キュー・マネージャーへの接続を作成する際に使用するデフォルト・ユーザー名。
wildcardFormat	int	• CHAR- ブローカー・バージョン 1 で使用された文字ワイルドカードのみ認識します。 • TOPIC - ブローカー・バージョン 2 で使用されたトピック・レベル・ワイルドカードのみ認識します。	使用されるワイルドカード構文のバージョン。

注 :

1. sslFipsRequired プロパティーの使用の重要な詳細については、441 ページの『IBM MQ リソース・アダプターの制限』を参照してください。
2. 出口を見つけることができるようにリソース・アダプターを構成する方法については、282 ページの『チャネル出口を使用するように IBM MQ classes for JMS を構成する』を参照してください。

以下の例は、ConnectionFactory オブジェクトの代表的なプロパティー・セットを示します。

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

管理対象の宛先オブジェクトのプロパティー

アプリケーション・サーバーは、管理対象の宛先オブジェクトのプロパティーを使用して、 JMS Queue オブジェクトまたは JMS Topic オブジェクトを作成します。

483 ページの表 67 は、Queue オブジェクトおよび Topic オブジェクトに共通するプロパティーを示します。

表 *67. Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
CCSID	ストリング	• **1208** • Java 仮想マシン (JVM) でサポートされるコード化文字セット ID	宛先用のコード化文字セット ID。
encoding	ストリング	• **NATIVE** • 以下の 3 文字のストリング – 先頭文字は 2 進整数の表記を示します。 - *N* は normal (標準) のエンコードを示します。 - *R* は reverse (逆) のエンコードを示します。 – 2 番目の文字はパック 10 進整数の表記を示します。 - *N* は normal (標準) のエンコードを示します。 - *R* は reverse (逆) のエンコードを示します。 – 3 番目の文字は浮動小数点数の表記を示します。 - *N* は標準の IEEE エンコードを示します。 - *R* は逆の IEEE エンコードを示します。 - *3* は zSeries エンコードを示します。 NATIVE はストリング NNN に相当します。	宛先の 2 進整数、パック 10 進整数、および浮動小数点数の表記
expiry	ストリング	• **APP** - メッセージの有効期限時刻はメッセージ・プロデューサーによって決定される • UNLIM - メッセージに有効期限を設けない • 0 - メッセージに有効期限を設けない • メッセージの有効期限時刻を表す正整数 (ミリ秒)	宛先に送信されるメッセージの有効期限時刻。
failIfQuiesce	ストリング	• **true** • false	キュー・マネージャーが静止状態の場合、宛先へのアクセスが失敗するかどうか。

表 67. *Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
messageBodyStyle	ストリング	• **UNSPECIFIED** • JMS • MQ	**messageBodyStyle** プロパティーを JMS キューと以下のトピックで設定できます。 UNSPECIFIED (デフォルト) • 送信の際に、IBM MQ classes for JMS は、WMQ_TARGET_CLIENT の値に応じて、MQRFH2 ヘッダーを生成して組み込みます。 • 受信の際に、IBM MQ classes for JMS は、MQRFH2 が存在する場合はその値に従って、JMS メッセージ・プロパティーを設定します。 MQRFH2 は JMS メッセージ本文の一部として示されません。 JMS • 送信の際に、IBM MQ classes for JMS は自動的に MQRFH2 ヘッダーを生成し、そのヘッダーを IBM MQ メッセージに組み込みます。 • 受信の際に、IBM MQ classes for JMS は、MQRFH2 が存在する場合はその値に従って、JMS メッセージ・プロパティーを設定します。 MQRFH2 は JMS メッセージ本文の一部として示されません。 MQ • 送信の際に、IBM MQ classes for JMS は MQRFH2 を生成しません。 • 受信の際に、IBM MQ classes for JMS は MQRFH2 を JMS メッセージ本文の一部として示します。

表 67. *Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
persistence	ストリング	• **APP** - メッセージの持続性はメッセージ・プロデューサーによって決定される • QDEF-メッセージの持続性は、IBM MQ キューの **DefPersistence** 属性によって決定されます。 • PERS - メッセージに持続性を与える • NON - メッセージに持続性を与えない • HIGH-メッセージの持続性は、256 ページの『JMS 持続メッセージ』の説明に従って、IBM MQ キューの **NonPersistentMessageClass** 属性によって決定されます。	宛先に送信されるメッセージの持続性。
priority	ストリング	• **APP** - メッセージの優先順位はメッセージ・プロデューサーによって決定される • QDEF-メッセージの優先順位は、IBM MQ キューの **DefPriority** 属性によって決定される。 • 0 (最低優先順位) から 9 (最高優先順位) の範囲の整数	宛先に送信されるメッセージの優先順位。
putAsyncAllowed	ストリング	• QUEUE - キュー定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • DESTINATION - キュー定義またはトピック定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • DISABLED - 非同期書き込みは許可されない • ENABLED - 非同期書き込みは許可される	メッセージ・プロデューサーが非同期書き込みを使用してこの宛先にメッセージを送信することを許可されるかどうか。
readAheadAllowed	int	• **DESTINATION** - キュー定義またはトピック定義を参照することによって先読みが許可されるかどうかを判別します。 • DISABLED - 先読みは許可されない • ENABLED - 先読みは許可される • QUEUE - キュー定義を参照することによって先読みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって先読みが許可されるかどうかを判別します。	メッセージ・コンシューマーおよびキュー・ブラウザーが先読みを使用してこの宛先から内部バッファーへの非永続メッセージを受信する前に、それらを取得することを許可されるかどうか。

表 67. *Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー (続き)			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
receiveCCSID	int	• **0** - JVM `Charset.defaultCharset` を使用します。 • 1208 - UTF-8 • サポートされるコード化文字セット ID	キュー・マネージャー・メッセージ変換のターゲット CCSID を 設定する宛先プロパティー。 **receiveConversion** が QMGR に設定されていない場合、値は無視されます。
receiveConversion	ストリング	• **CLIENT_MSG** • QMGR	キュー・マネージャーによりデータ変換を実行するかどうかを決定する宛先プロパティー。
targetClient	ストリング	• **JMS** - JMS アプリケーションをメッセージのターゲットにする • MQ - JMS 以外の IBM MQ アプリケーションをメッセージのターゲットにする	宛先に送信されるメッセージのターゲットが JMS アプリケーションかどうか。 ターゲットが JMS アプリケーションであるメッセージには MQRFH2 ヘッダーが含まれています。

486 ページの表 68 は、Queue オブジェクトに固有のプロパティーを示します。

表 *68. Queue* オブジェクトに固有のプロパティー			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
baseQueueManagerName	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	基礎となる IBM MQ キューを所有するキュー・マネージャーの名前。
baseQueueName	ストリング	• **"" (空ストリング)** • キュー名	基礎となる IBM MQ キューの名前。

486 ページの表 69 は、Topic オブジェクトに固有のプロパティーを示します。

表 *69. Topic* オブジェクトに固有のプロパティー			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
baseTopicName	ストリング	• **"" (空ストリング)** • トピック名	基礎となるトピックの名前。
brokerCCDurSubQueue >	ストリング	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが永続サブスクリプション・メッセージを受信するキューの名前。
brokerDurSubQueue	ストリング	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • キュー名	永続トピック・サブスクライバーがメッセージを受信するキューの名前。 詳しくは、 IBM MQ Explorer 資料の **BROKEDURRSUBQ** プロパティーを参照してください。

表 69. *Topic* オブジェクトに固有のプロパティー *(続き)*			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
brokerPub キュー	ストリング	• **設定されない** • キュー名	パブリッシュ済みメッセージが送信されたキュー (ストリーム・キュー) の名前。 このプロパティーの値は、ConnectionFactory オブジェクトの **brokerPubQueue** プロパティーの値をオーバーライドします。 ただし、このプロパティーの値を設定しない場合、ConnectionFactory オブジェクトの **brokerPubQueue** プロパティーの値が代わりに使用されます。
brokerPubQueueManager	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	トピックに関して公開されたメッセージが送信されるキューを所有するキュー・マネージャーの名前。
brokerVersion	ストリング	• **設定されない** • 1 • 2	使用されているブローカーのバージョン。 このプロパティーの値は、ConnectionFactory オブジェクトの **brokerVersion** プロパティーの値をオーバーライドします。 ただし、このプロパティーの値を設定しない場合、ConnectionFactory オブジェクトの **brokerVersion** プロパティーの値が代わりに使用されます。

以下の例は、Queue オブジェクトのプロパティー・セットを示します。

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

以下の例は、Topic オブジェクトのプロパティー・セットを示します。

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

関連タスク

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

WebSphere Application Server での JMS リソースの構成

関連資料

AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

アクティベーション・スペックの *targetClientMatching* プロパティーの構成

アクティベーション・スペックの **targetClientMatching** プロパティーを構成することで、要求メッセージに MQRFH2 ヘッダーが含まれないときに応答メッセージに MQRFH2 ヘッダーを含めることができます。 これは、応答メッセージでアプリケーションが定義するメッセージ・プロパティーがメッセージの送信時に組み込まれることを意味します。

このタスクについて

例えば、メッセージ駆動型 Bean (MDB) アプリケーションが、MQRFH2 ヘッダーを含まないメッセージを IBM MQ JCA リソース・アダプターのアクティベーション・スペックを介してコンシュームし、その後、要求メッセージの JMSReplyTo フィールドから作成された JMS 宛先に応答メッセージを送信する場合は、要求メッセージに MQRFH2 ヘッダーが含まれていなくてもそれを応答メッセージに組み込む必要があります。そうしないと、アプリケーションが応答メッセージで定義したメッセージ・プロパティーが失われてしまいます。

targetClientMatching プロパティーでは、着信メッセージに MQRFH2 ヘッダーがある場合にのみ応答メッセージ (着信メッセージの JMSReplyTo ヘッダー・フィールドで識別されるキューに送信される) に MQRFH2 ヘッダーを組み込むようにするかを定義します。 アクティベーション・スペックのこのプロパティーは、WebSphere Application Server traditional でも WebSphere Liberty でも構成できます。

targetClientMatching プロパティーの値を `false` に設定すると、着信要求メッセージに MQRFH2 が含まれない場合に、その応答メッセージ (着信要求メッセージの JMSReplyTo ヘッダーから作成された JMS 宛先に送信される) に MQRFH2 ヘッダーが組み込まれます。 これは、JMS 宛先の **targetClient** プロパティーが値 `0` (メッセージに MQRFH2 ヘッダーが含まれることを意味する) に設定されるからです。 アウトバウンド・メッセージに MQRFH2 ヘッダーが存在すれば、IBM MQ キューへの送信時にメッセージにユーザー定義のメッセージ・プロパティーを格納できます。

targetClientMatching プロパティーを `true` に設定した場合、要求メッセージに MQRFH2 ヘッダーが組み込まれていなければ、応答メッセージにも MQRFH2 ヘッダーは組み込まれません。

手順

- WebSphere Application Server traditional で、管理コンソールを使用して、**targetClientMatching** プロパティーを IBM MQ アクティベーション・スペックのカスタム・プロパティーとして定義します。
  a) ナビゲーション・ペインで、「**リソース**」->「**JMS**」->「**アクティベーション・スペック**」をクリックします。
  b) 表示または変更するアクティベーション・スペックの名前を選択します。
  c)「**カスタム・プロパティー**」->「**新規**」をクリックした後、新規カスタム・プロパティーの詳細を入力します。
  プロパティーの名前を `targetClientMatching` に設定し、タイプを `java.lang.Boolean` に設定し、値を `false` に設定します。
- WebSphere Liberty で、 `server.xml` 内のアクティベーション・スペックの定義で **targetClientMatching** プロパティーを指定します。
  以下に例を示します。

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

関連概念

223 ページの『JMS アプリケーションでの宛先の作成』

Java Naming and Directory Interface (JNDI) ネーム・スペースから宛先を管理対象オブジェクトとして取り出す代わりに、JMS アプリケーションは実行時に動的に宛先を作成するセッションを使用できます。 アプリケーションは URI (Uniform Resource Identifier) を使用して IBM MQ キューまたはトピックを識別し、オプションで、Queue または Topic オブジェクトの 1 つ以上のプロパティーを指定することができます。

470 ページの『アウトバウンド通信のリソース・アダプターの構成』
アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

WebSphere Liberty での IBM MQ メッセージ駆動型 Bean の一時停止

アクティベーション・スペックの **maxSequentialDeliveryFailures** プロパティーにより、メッセージ駆動型 Bean (MDB) を一時停止する前にリソース・アダプターで許容される MDB インスタンスに対する連続するメッセージ・デリバリーの失敗の最大数が定義されます。

始める前に

WebSphere Liberty で MDB を休止させる原因になる一連のイベントについて理解しておく必要があります。 リソース・アダプターは、以下の状況をメッセージ送達に失敗したものと見なします。

- MDB の **onMessage** メソッドからチェックなし例外がスローされた。
- メッセージを MDB に送達する前に、リソース・アダプターの処理で JMSException が発生した。
- メッセージを MDB に送達した後に、リソース・アダプターの処理で JMSException が発生した。
- メッセージの取り込みに使用された XA トランザクションまたはローカル・トランザクションがロールバックされた。
- メッセージを MDB に送達するために使用可能なスレッドがアプリケーション・サーバーに存在しない。

このタスクについて

maxSequentialDeliveryFailures プロパティーのデフォルト値は *-1* です。これは、MDB が休止されないことを意味します。 その他の負の値はすべて *-1* と同じ扱いになります。 値は以下のいずれかです。

- *0* は、最初のエラーで MDB が休止することを意味します
- *1* は、エラーが 2 回連続すると MDB が休止することを意味します
- *2* は、エラーが 3 回連続すると MDB が休止することを意味します。

アクティベーション・スペックのこのプロパティーは、 WebSphere Liberty でのみ (Liberty のレベルが 18.0.0.4 以上) 構成できます。

重要 : Liberty 以外のアプリケーション・サーバー環境でこの属性をデフォルト値以外の値に設定すると、その値は無視され、警告メッセージがログに書き込まれます。

また、IBM MQ リソース・アダプターを汎用リソース・アダプターとして WebSphere Liberty にインストールすることが可能です。 これを行うと、IBM MQ と WebSphere Application Server の統合機能がすべて無効になるので、リソース・アダプターが Liberty で実行されていることを検出できなくなります。 したがって、 **maxSequentialDeliveryFailures** を *0* 以上の値に設定することはサポートされません。設定すると、警告メッセージがログに書き込まれます。

手順

- WebSphere Liberty で、 server.xml 内のアクティベーション・スペックの定義で **maxSequentialDeliveryFailures** プロパティーを指定します。

  以下に例を示します。

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

関連概念

470 ページの『アウトバウンド通信のリソース・アダプターの構成』

アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

リソース・アダプターのインストールの検証

IBM MQ リソース・アダプター用のインストール検査テスト (IVT) プログラムは、EAR ファイルとして提供されます。 このプログラムを使用するには、このプログラムをデプロイし、JCA リソースとして一部のオブジェクトを定義する必要があります。

このタスクについて

インストール検査テスト (IVT) プログラムは、 `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) または `wmq.jmsra.ivt.ear` (JMS 2.0) と呼ばれるエンタープライズ・アーカイブ (EAR) ファイルとして提供されます。 このファイルは、 IBM MQ classes for JMS と共に、 IBM MQ リソース・アダプター RAR ファイル `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) または `wmq.jmsra.rar` (JMS 2.0) と同じディレクトリーにインストールされます。 これらのファイルがインストールされる場所については、 444 ページの『IBM MQ リソース・アダプターのインストール』 を参照してください。

IVT プログラムはアプリケーション・サーバー上にデプロイする必要があります。 IVT プログラムには、サーブレットと、IBM MQ キューとのメッセージの送受信が可能かどうかをテストする MDB が含まれています。 IVT プログラムを使用して、IBM MQ リソース・アダプターが分散トランザクションをサポートするように正しく構成されていることを検証できます。 IBM MQ リソース・アダプターを IBM 以外のアプリケーション・サーバーにデプロイする場合、IBM サービスは、アプリケーション・サーバーが正しく構成されていることを検証するために IVT が作動していることを実証することを要求する場合があります。

IVT プログラムを実行するには、その前に ConnectionFactory オブジェクト、Queue オブジェクト、および (場合によっては) Activation Specification オブジェクトを JCA リソースとして定義する必要があります。また、ご使用のアプリケーション・サーバーが、これらの定義から JMS オブジェクトを作成して、それらを JNDI 名前空間にバインドするようにする必要があります。 独自の QueueManager のホストおよびポート設定と一致するようにオブジェクトのプロパティーを選択することができますが、以下にプロパティー・セットの単純な例を示します。

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

ConnectionFactory、Queue、および Activation Specification オブジェクトの定義に使用されるメカニズムは、アプリケーション・サーバーによって異なります。 例えば、WebSphere Liberty 内でこれらのプロパティーを設定するには、以下のエントリーをアプリケーション・サーバーの `server.xml` ファイルに追加します。

```
JM 3.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
```

```
maxPoolDepth="1"/>
</jmsActivationSpec>
```

```
JMS 2.0 <!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

デフォルトでは、IVT プログラムは、JNDI 名前空間内で jms/ivt/IVTCF という名前の ConnectionFactory オブジェクト、および jms/ivt/IVTQueue という名前の Queue オブジェクトがバインドされることを予期します。 別の名前を使用することもできますが、その場合はオブジェクトの名前を IVT プログラムの最初のページに入力し、EAR ファイルを適切に変更する必要があります。

IVT プログラムをデプロイし、アプリケーション・サーバーで JMS オブジェクトを作成して、 それらを JNDI 名前空間にバインドした後、以下の手順を実行して IVT プログラムを開始できます。

手順

1. Web ブラウザーに以下のフォーマットで URL を入力して、IVT プログラムを開始します。

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   ここで、*app_server_host* は、アプリケーション・サーバーが稼働しているシステムの IP アドレスまたはホスト名で、*port* は、アプリケーション・サーバーが listen している TCP ポートの番号です。 以下が例となります。

   ```
   http://localhost:9080/WMQ_IVT/
   ```

   以下に、IVT プログラムによって表示される初期ページの例を示します。

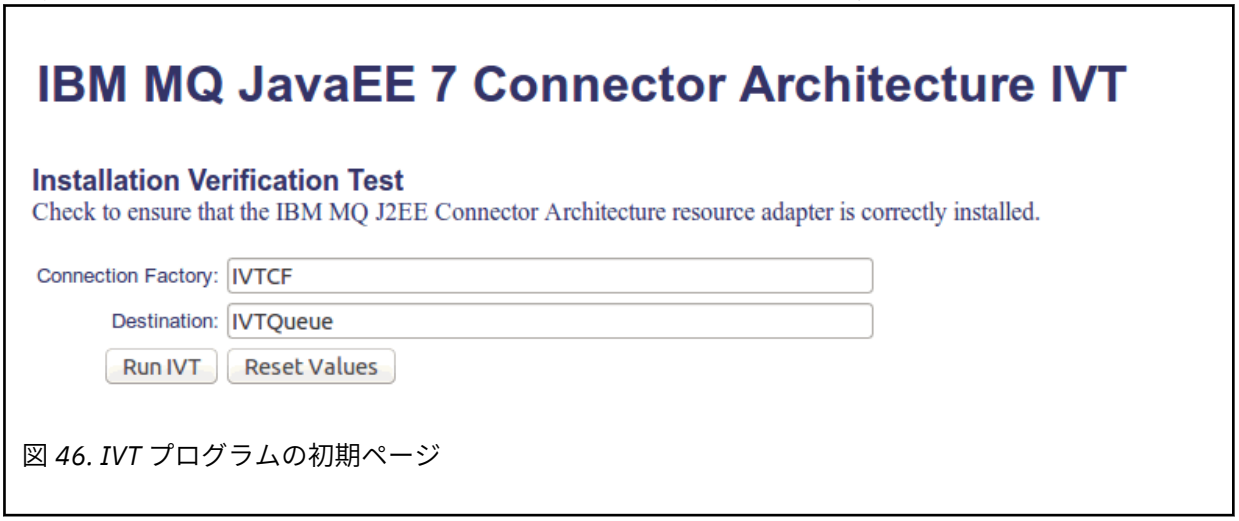
   

   図 *46. IVT* プログラムの初期ページ

2. テストを実行するには、「**IVT の実行**」をクリックします。

以下に、IVT が成功した場合に表示されるページの例を示します。

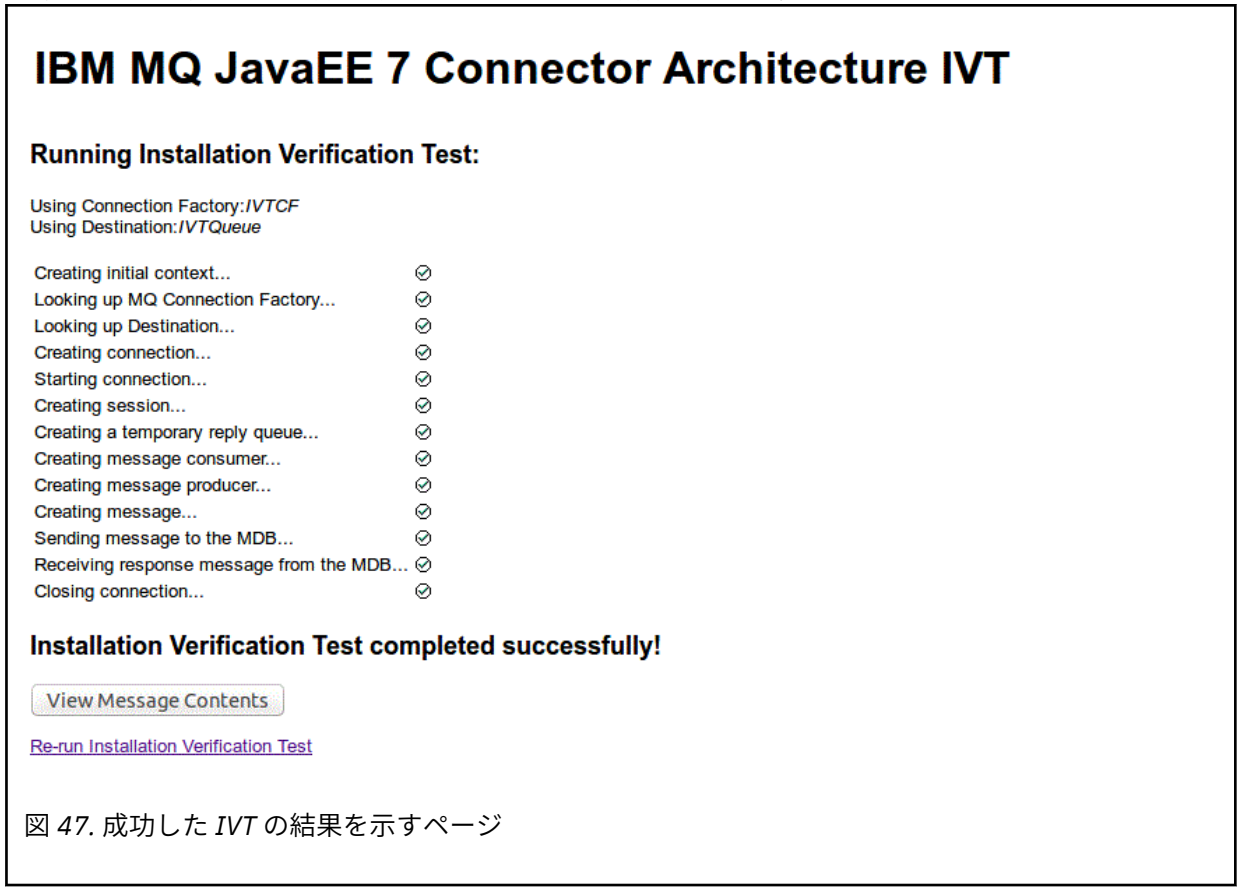


図 *47.* 成功した *IVT* の結果を示すページ

以下に、IVT が失敗した場合に表示されるページの例を示します。 失敗の原因についてさらに詳しい情報を入手するには、「**スタック・トレースの表示**」をクリックします。

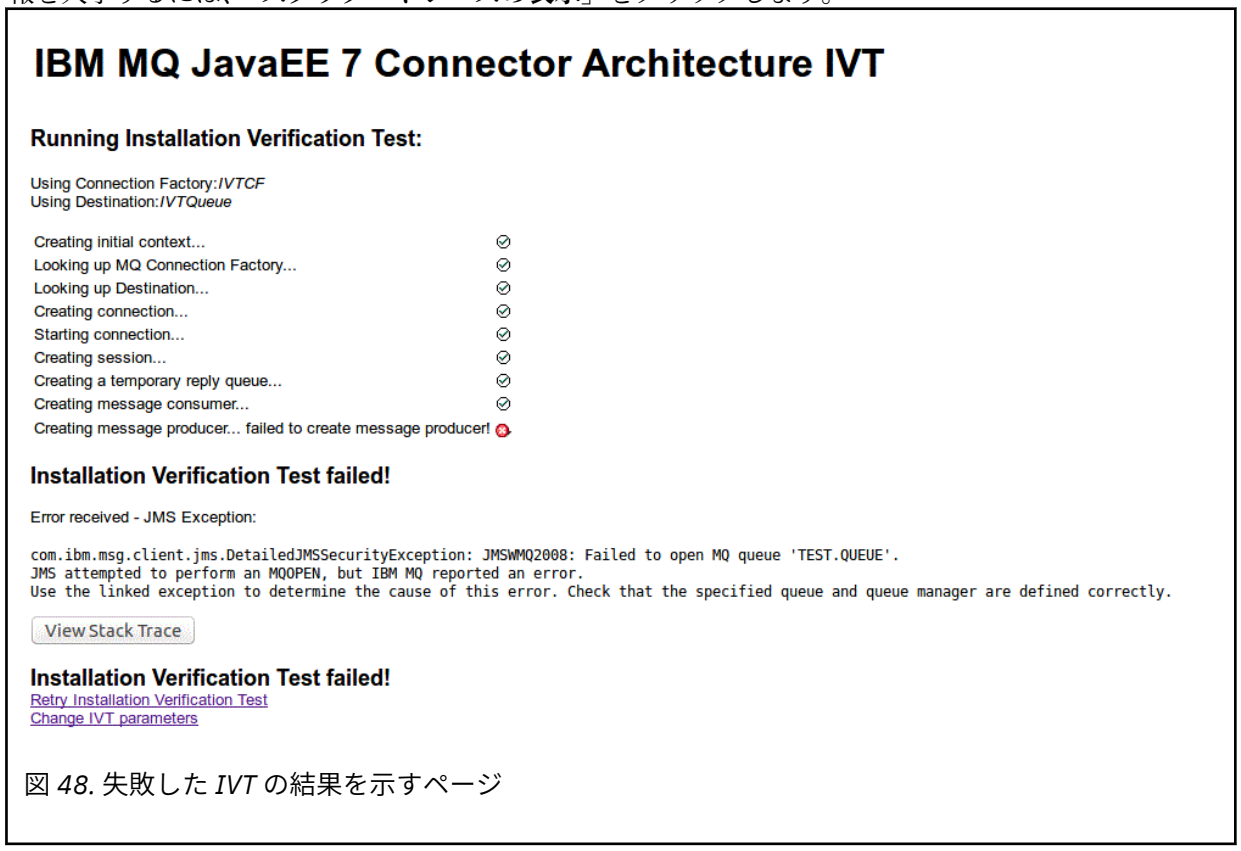


図 *48.* 失敗した *IVT* の結果を示すページ

Windows GlassFish Server でのリソース・アダプターのインストールおよびテスト

Windows オペレーティング・システム上の GlassFish Server に IBM MQ リソース・アダプターをインストールするには、まずドメインを作成して開始する必要があります。 その後、リソース・アダプターを展開して構成し、インストール検査テスト (IVT) アプリケーションを実行できます。

始める前に

- この説明は、GlassFish Server バージョン 4 に当てはまります。
- このバージョンの GlassFish Server は Jakarta EE をサポートしていません。

このタスクについて

このタスクは、GlassFish Server アプリケーション・サーバーが実行されていることと、ユーザーがその標準的な管理タスクに精通していることを前提としています。 さらに、このタスクは IBM MQ がローカル・システムにインストールされていることと、ユーザーが標準的な管理タスクに精通していることも前提としています。

注 : 次のタスクの手順を実行するには、IBM MQ が正常にインストールされ、次のオブジェクトが構成されている必要があります。

- QM というキュー・マネージャー。ポート 1414 で始動し、チャネル SYSTEM.DEF.SVRCONN を使用し、クライアント・トランスポートを使用して接続するようにします。
- Q1 というキュー。

手順

1. GlassFish Server の **asadmin** シェル・プログラムを開始します。
   a) Windows コマンド行を開き、*GlassFish*/bin ディレクトリーにナビゲートします。ここで、*GlassFish* は GlassFish Server バージョン 4 がインストールされているディレクトリーです。
   b) コマンド・ラインでコマンド **asadmin** を入力します。

      asadmin コマンドにより、新規ドメインの作成を可能にするシェル・プログラムがコマンド・ラインで開かれます。

   GlassFish Server バージョン 4 がシステムで開始されます。
2. ドメインを作成し、開始します。
   a) ポートとドメイン・ネームを指定して **create-domain** コマンドを使用して、新規ドメインを作成します。 コマンド行で、次のコマンドを入力します。

      ```
      create-domain --adminport port domain_name
      ```

      port はポート番号で、*domain_name* はドメインに使用する名前です。

      注 : create-domain コマンドには、多くのオプション・パラメーターが関連付けられています。 ただし、このタスクの場合に必要となるのは `--adminport` パラメーターのみです。 詳しくは、GlassFish Server バージョン 4 の製品資料を参照してください。

      指定したポートが使用中の場合、次のメッセージが表示されます。

      domain_name port のポートは使用中です

      指定したドメイン・ネームが使用中の場合、指定した名前は既に使用中であることを知らせるメッセージを受け取ります。また、現在使用できないすべてのドメイン・ネームのリストも受け取ります。
   b) ユーザー名とパスワードの入力を求めるプロンプトが出されたら、Web ブラウザーを介してアプリケーション・サーバーへのログオンに使用する資格情報を入力します。

コマンドが正常に完了すると、ドメイン作成の要約を示すメッセージがコマンド行に表示されます。このメッセージには、`Command create-domain executed successfully.` というメッセージが含まれます。

正常にドメインが作成されました。

c) コマンド・ラインに次のコマンドを入力してドメインを開始します。

```
start-domain domain_name
```

domain_name は、前に指定したドメイン・ネームです。

3. Web ブラウザーを使用して、GlassFish アプリケーション・サーバーにアクセスします。
   a) Web ブラウザーのアドレス・バーに、次のコマンドを入力します。

```
localhost:port
```

port は、ドメインを作成したときに指定したポートです。

GlassFish Console が表示されます。

   b) GlassFish Console がロードされると、ユーザー名とパスワードを求めるプロンプトが出されます。ステップ 2b で指定した資格情報を入力してください。
4. リソース・アダプターを GlassFish Server 4 にアップロードします。
   a) ツールバー「**共通タスク (Common Tasks)**」で、「**アプリケーション**」メニュー項目を選択し、「**アプリケーション**」ページを表示します。
   b)「**デプロイ (Deploy)**」ボタンをクリックし、「**アプリケーションまたはモジュールのデプロイ (Deploy Applications or Modules)**」ページを開きます。
   c)「**参照**」 ボタンをクリックして、 `wmq.jmsra.rar` ファイルの場所にナビゲートします。 ファイルを選択し、 「**OK**」をクリックします。
5. 接続プールを作成します。
   a) ツールバーの「**リソース**」の下の「**コネクター (Connectors)**」メニュー項目を選択します。
   b)「**コネクター接続プール (Connector Connection Pools)**」メニュー項目を選択し、「**コネクター接続プール (Connector Connection Pools)**」ページを開きます。
   c)「**新規**」をクリックして、「**新規コネクター接続プール (ステップ 1/2) (New Connector Connection Pool (Step 1 of 2))**」ページを開きます。
   d)「**新規コネクター接続プール (ステップ 1/2) (New Connector Connection Pool (Step 1 of 2))**」ページで、プール名 jms/ivt/IVTCF-Connection-Pool を「**プール名 (Pool Name)**」フィールドに入力します。
   e)「**リソース・アダプター (Resource Adapter)**」フィールドで、`wmq.jmsra` を選択します。
   f)「**接続定義**」 フィールドに、 `javax.jms.ConnectionFactory` と入力します。
   g)「**次へ**」を選択し、「**完了**」を選択します。
6. コネクター・リソースを作成します。
   a) ツールバーの「**コネクター (Connectors)**」メニューで、「**コネクター・リソース (Connector Resource)**」オプションを選択し、「**コネクター・リソース (Connector Resources)**」ページを開きます。
   b)「**新規**」を選択し、「**新規コネクター・リソース (New Connector Resource)**」ページを開きます。
   c)「**JNDI 名 (JNDI Name)**」フィールドに、`IVTCF` を入力します。
   d)「**プール名 (Pool Name)**」フィールドに、`jms/ivt/IVTCF-Connection-Pool` と入力します。
   e) その他のすべてのフィールドは空のままにします。
   f) 以下の各プロパティー/値のペアに対して、「**プロパティーの追加 (Add Property)**」をクリックし、以下の例に示されているプロパティー名と値を入力します。

- 名前: host; 値: localhost
- 名前: port; 値: 1414
- 名前: channel; 値: SYSTEM.DEF.SVRCONN
- 名前: queueManager; 値: QM
- 名前: transportType; 値: CLIENT

注 : 独自の構成設定に対して正しい値を使用していることを確認してください。それは、この例で示されているものと異なる場合があります。

g) ツールバーの「**コネクター (Connectors)**」で、「**管理オブジェクト・リソース (Admin Object Resources)**」メニュー項目を選択し、「**管理オブジェクト・リソース (Admin Object Resources)**」ページを開きます。

h)「**管理オブジェクト・リソース (Admin Object Resources)**」ページで、「**新規**」をクリックして、「**新規管理オブジェクト・リソース (New Admin Object Resource)**」ページを開きます。

i)「**JNDI 名 (JNDI Name)**」フィールドに、IVTQueue を入力します。

j)「**リソース・アダプター (Resource Adapter)**」フィールドに、wmq.jmsra と入力します。

k)「**リソース・タイプ**」フィールドに、javax.jms.Queue と入力します。

l)「**クラス名**」フィールドはそのままにしておきます。

m) 以下の各プロパティー/値のペアに対して、「**プロパティーの追加 (Add Property)**」をクリックし、以下の例に示されているプロパティー名と値を入力します。

- 名前: name; 値: IVTQueue
- 名前: baseQueueManagerName; 値: QM
- 名前: baseQueueName; 値: Q1

注 : 独自の構成設定に対して正しい値を使用していることを確認してください。それは、この例で示されているものと異なる場合があります。

n)「**OK**」をクリックします。

o)「**使用可能**」チェック・ボックスを選択し、「**使用可能**」をクリックします。

7. EAR ファイル wmq.jmsra.ivt.ear を GlassFish Server にデプロイします。

a) ツールバーの「**アプリケーション**」オプションをクリックし、「**アプリケーション**」ページを表示します。

b)「**デプロイ (Deploy)**」をクリックし、IVT アプリケーションを追加します。

c)「**ロケーション**」フィールドで、wmq.jmsra.ivt.ear へナビゲートして選択します。

d)「**仮想サーバー (Virtual Servers)**」フィールドで、「**サーバー**」を選択し、「**OK**」をクリックします。

8. IVT プログラムを起動します。

a) ツールバーの「**アプリケーション**」オプションをクリックし、「**アプリケーション**」ページを表示します。

b)「デプロイ済みアプリケーション」テーブルの wmq.jmsra.ivt をクリックします。

c)「モジュールとコンポーネント (Modules and Components)」テーブルの「**起動 (Launch)**」ボタンをクリックします。

d) http: リンクを選択します。

e)「**IVT の実行 (Run IVT)**」をクリックします。

IVT プログラムが起動します。正常に起動すると、次の出力が表示されます。

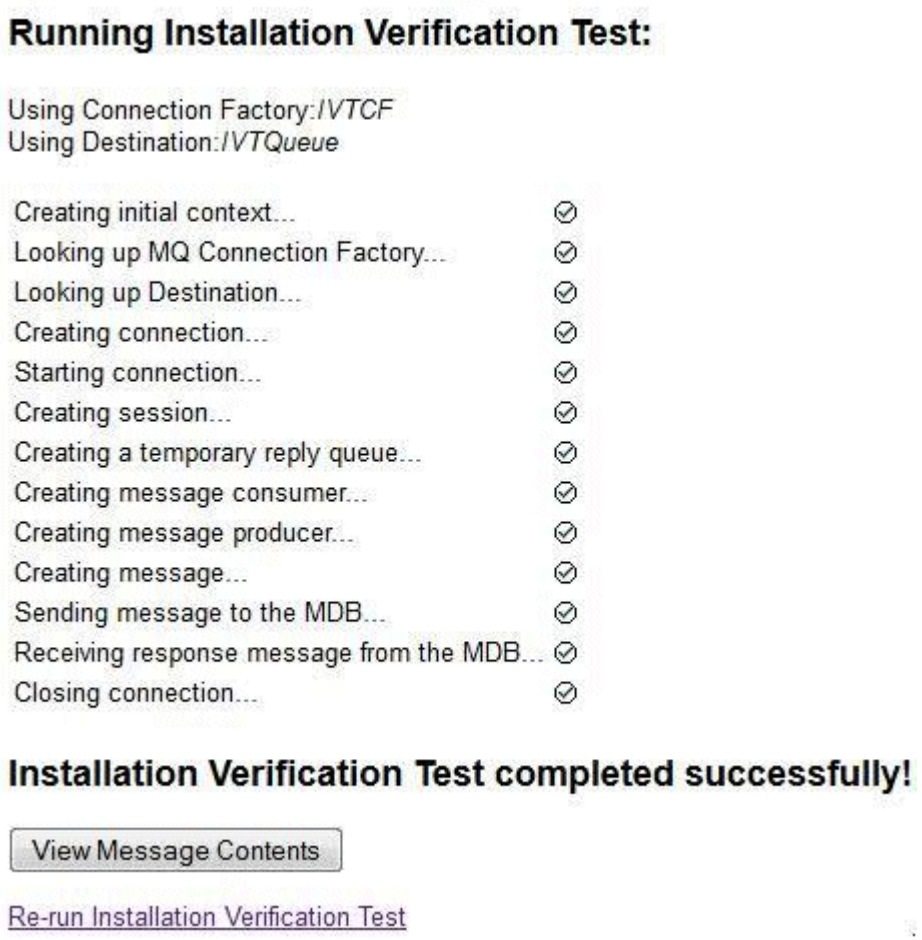


図 *49. IVT* 正常な起動を示す出力

Wildfly でのリソース・アダプターのインストールおよびテスト

IBM MQ リソース・アダプターを Wildfly V10 にインストールする場合、最初に構成ファイルにいくつかの変更を加え、IBM MQ リソース・アダプター用のサブシステム定義を追加する必要があります。 その後、リソース・アダプターをデプロイし、インストール検査テスト (IVT) アプリケーションをインストールして実行することによってテストできます。

始める前に

- この説明は、Wildfly V10 に当てはまります。
- このバージョンの WildFly は、 Jakarta EE をサポートしていません。

このタスクについて

このタスクは、WildFly アプリケーション・サーバーが実行されていることと、ユーザーがその標準的な管理タスクに精通していることを前提としています。 さらに、このタスクは IBM MQ がインストールされていることと、ユーザーが標準的な管理タスクに精通していることも前提としています。

手順

1. IBM MQ キュー・マネージャー ExampleQM を作成し、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』 で説明されているようにセットアップします。

   キュー・マネージャーのセットアップ時に、次の点に注意してください。

   - リスナーはポート 1414 で開始する必要があります。

- 使用するチャネルは SYSTEM.DEF.SVRCONN です。
- IVT アプリケーションが使用するキューの名前は TEST.QUEUE です。

また、モデル・キュー SYSTEM.DEFAULT.MODEL.QUEUE も DSP 権限と PUT 権限を付与され、このアプリケーションが一時的な応答キューを作成できるようにする必要があります。

2. 構成ファイル *WildFly_Home*/standalone/configuration/standalone-full.xml を編集して、以下のサブシステムを追加します。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. wmq.jmsra.rar ファイルをディレクトリー *WildFly_Home*/standalone/deployments にコピーすることで、リソース・アダプターをサーバーにデプロイします。
4. wmq.jmsra.ivt.ear ファイルをディレクトリー *WildFly_Home*/standalone/deployments にコピーすることで、IVT アプリケーションをデプロイします。

5. コマンド・プロンプトを表示し、ディレクトリー *WildFly_Home*/bin にナビゲートし、以下のコマンドを実行することで、アプリケーション・サーバーを起動します。

```
standalone.bat -c standalone-full.xml
```

6. IVT アプリケーションを実行します。
   詳細については、490 ページの『リソース・アダプターのインストールの検証』を参照してください。
   Wildfly の場合、デフォルト URL は `http://localhost:8080/WMQ_IVT/` です。

IBM MQ と WebSphere Application Server の併用

WebSphere Application Server の IBM MQ メッセージング・プロバイダーを介して、Java Message Service (JMS) メッセージング・アプリケーションは、 JMS メッセージング・リソースの外部プロバイダーとして IBM MQ システムを使用できます。

このタスクについて

Java で作成され、 WebSphere Application Server で実行されているアプリケーションは、 Java Message Service (JMS) 仕様を使用してメッセージングを実行できます。 この環境でのメッセージングは、IBM MQ キュー・マネージャーが提供することができます。

IBM MQ キュー・マネージャーを使用する利点は、接続する JMS アプリケーションが IBM MQ ネットワークの機能を十分に利用できるという点です。 これにより、アプリケーションは、多数のプラットフォームで実行されているキュー・マネージャーとメッセージを交換することができます。

アプリケーションは、キュー接続ファクトリー・オブジェクト用にクライアント・トランスポート または バインディング・トランスポート を 使用できます。 バインディング・トランスポートの場合、キュー・マネージャーは接続を要求するアプリケーションのローカルに存在している必要があります。

デフォルトでは、 IBM MQ キューに保持される JMS メッセージは、 MQRFH2 ヘッダーを使用して、 JMS メッセージ・ヘッダー情報の一部を保持します。 多数のレガシー IBM MQ アプリケーションは、これらのヘッダー付きメッセージを処理できず、独自の特性ヘッダー (例えば、CICS ブリッジ用 MQCIH、または IBM MQ Workflow アプリケーション用 MQWIH) が必要です。 これらの特殊な考慮事項について詳しくは、JMS メッセージの IBM MQ メッセージへのマッピングを参照してください。

関連タスク

WebSphere Application Server での JMS リソースの構成
最新のリソース・アダプター保守レベルを使用するためのアプリケーション・サーバーの構成

WebSphere Application Server と IBM MQ の併用

IBM MQ および IBM MQ for z/OS は、WebSphere Application Server に組み込まれているデフォルトのメッセージング・プロバイダーと併用したり、プロバイダーの代わりとして使用したりすることができます。

IBM MQ メッセージング・プロバイダーは、WebSphere Application Server の一部としてインストールされます。 これには、IBM MQ 版のリソース・アダプターと IBM MQ 拡張トランザクション・クライアントの機能が含まれ、これによりキュー・マネージャーがアプリケーション・サーバーによって管理される XA トランザクションに参加できます。 リソース・アダプターを使用して、メッセージ駆動型 Bean は活動化仕様またはリスナー・ポートのいずれかを使用するよう構成できます。

IBM MQ リソース・アダプターのインストール検査テスト・プログラムをアプリケーション・サーバーにデプロイして正常に実行できれば、そのアプリケーション・サーバーはサポートされます。 IBM MQ リソース・アダプターのインストール検査テスト・プログラムが正常に実行されたら、IBM MQ リソース・アダプターを、サポートされている任意の IBM MQ キュー・マネージャーに接続できます。

WebSphere Application Server から IBM MQ への JMS 接続

WebSphere Application Server で使用できる IBM MQ のレベルを考慮する前に、アプリケーション・サーバー内部で実行される Java Message Service (JMS) アプリケーションが IBM MQ キュー・マネージャーに接続する方法を理解することが重要です。

IBM MQ キュー・マネージャーのリソースにアクセスする必要がある JMS アプリケーションは、以下のいずれかのトランスポート・タイプを使用してアクセスできます。

BINDINGS
このトランスポートは、アプリケーション・サーバーとキュー・マネージャーが同じマシンおよびオペレーティング・システム・イメージにインストールされている場合に使用できます。 BINDINGS モードを使用する場合、2 つの製品の間のすべての通信はプロセス間通信 (IPC) を使用して行われます。

IBM MQ メッセージング・プロバイダーには、BINDINGS モードの IBM MQ キュー・マネージャーへの接続に必要なネイティブ・ライブラリーが含まれていません。 BINDINGS モードの接続を使用するには、IBM MQ をアプリケーション・サーバーと同じマシンにインストールして、リソース・アダプターのネイティブ・ライブラリーのパスが、これらのライブラリーが配置されている IBM MQ ディレクトリーを指すよう構成する必要があります。 詳しくは、以下の WebSphere Application Server 製品の資料を参照してください。

- WebSphere Application Server traditional については、『ネイティブ・ライブラリー情報による IBM MQ メッセージ・プロバイダーの構成』を参照してください。
- WebSphere Liberty については、「IBM MQ メッセージング・プロバイダーを使用するための Liberty への JMS アプリケーションのデプロイ」を参照してください。

z/OS z/OS では、WebSphere Application Server 接続ファクトリーをバインディング・モードで IBM MQ キュー・マネージャーに接続する場合、WebSphere Application Server STEPLIB 連結に正しい IBM MQ ライブラリーを指定する必要があります。 詳細については、WebSphere Application Server 製品資料の「IBM MQ のライブラリーおよび WebSphere Application Server For z/OS STEPLIB」を参照してください。

CLIENT
クライアント・トランスポートは、WebSphere Application Server と IBM MQ の間の通信に TCP/IP を使用します。 CLIENT モードはアプリケーション・サーバーとキュー・マネージャーが別のマシンに配置されている場合に使用されるだけでなく、2 つの製品が同じマシンおよびオペレーティング・システム・イメージにインストールされている場合にも使用できます。

JMS アプリケーションは、BINDINGS_THEN_CLIENT のトランスポート・タイプも指定できます。 このトランスポート・タイプが使用される場合、アプリケーションは最初に BINDINGS モードを使用してキュー・マネージャーに接続しようとします。 接続できない場合は、CLIENT トランスポートを試行します。

WebSphere Application Server 内にインストールされている IBM MQ リソース・アダプターのバージョンを確認する方法

WebSphere Application Server 内にインストールされている IBM MQ リソース・アダプターのバージョンについては、技術情報「Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server?」を参照してください。

次の Jython コマンドと JACL コマンドを使用して、WebSphere Application Server で現在使用されているリソース・アダプターのレベルを調べることができます。

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

注 : このコマンドを実行するには、入力した後で **Return** を 2 回クリックする必要があります。

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

リソース・アダプターの更新

アプリケーション・サーバーとともにインストールされる IBM MQ リソース・アダプターに対する更新は、WebSphere Application Server フィックスパックに組み込まれています。 「**リソース・アダプターの更新 ...**」 を使用して IBM MQ リソース・アダプターを更新しています。 WebSphere Application Server Administrative Console の機能は推奨されません。 WebSphere Application Server フィックスパックで提供されている更新は無効になるためです。

MQ_INSTALL_ROOT 変数

WebSphere Application Server 7.0 以降、MQ_INSTALL_ROOT はネイティブ・ライブラリーを配置するためにのみ使用され、リソース・アダプターで構成されるネイティブ・ライブラリー・パスによってオーバーライドされます。

WebSphere Application Server から IBM MQ への接続

重要 :

1. サポートされるすべてのバージョンの WebSphere Application Server は、バンドルされている IBM MQ リソース・アダプターを使用して、サポートされるすべてのバージョンの IBM MQ に接続できます。
2. バインディング・モードが使用される場合、WebSphere Application Server 内の特定のライブラリーのバージョンは接続先のキュー・マネージャーのバージョンと一致している必要があります。
   - WebSphere Application Server は、IBM MQ 9.3 と共に提供されるネイティブ・ライブラリーをロードするよう構成する必要があります。 詳しくは、97 ページの『Java Native Interface (JNI) ライブラリーの構成』 を参照してください。
   - z/OS z/OS では、WebSphere Application Server STEPLIB 連結に正しい IBM MQ ライブラリーを指定する必要があります。

     必要な IBM MQ ライブラリーの詳細については、「IBM MQ ライブラリおよび WebSphere Application Server for z/OS STEPLIB」を参照してください。

     LINKLIST (LINKLST) 内に 1 つのバージョンの IBM MQ 用のライブラリーがある場合、STEPLIB 内でライブラリーをオーバーライドすることにより、異なるバージョンの IBM MQ に接続できます。
3. IBM MQ Resource Adapter バージョンは、キュー・マネージャーのインストールで提供されるネイティブ (共有) ライブラリーから独立しています。

   例えば、WebSphere Application Server 8.5 の場合、IBM MQ 8.0 リソース・アダプターによって、IBM MQ 9.0 キュー・マネージャーに対するバインディング接続を IBM MQ 9.0 ネイティブ・ライブラリーを使用して引き続き管理できます。

詳細については、438 ページの『IBM MQ リソース・アダプターのサポートに関するステートメント』を参照してください。

BINDINGS および CLIENT トランスポート・タイプは、任意のバージョンの WebSphere Application Server から IBM MQ に接続するために使用できます。 BINDINGS トランスポート・タイプの場合は、以下の制約事項が適用されます。

- IBM MQ は、アプリケーション・サーバーと同じマシン上にインストールする必要があります。
- WebSphere Application Server は、IBM MQ と共に提供されるネイティブ・ライブラリーをロードするよう構成する必要があります。
- z/OS z/OS では、 WebSphere Application Server 接続ファクトリーを IBM MQ キュー・マネージャーにバインディング・モードで接続する場合、 WebSphere Application Server STEPLIB 連結で正しい IBM MQ ライブラリーを指定する必要があります。

次の表は、 IBM MQ リソース・アダプターの各バージョンの実行がサポートされている WebSphere Application Server のバージョンを示しています。

表 *70. WebSphere Application Server* のバージョンを *IBM MQ* リソース・アダプターのバージョンにマップする。

IBM MQ リソース・アダプターのバージョン	**このバージョンのリソース・アダプターを実行できる WebSphere Application Server のバージョン**
IBM MQ 9.0 以降	リソース・アダプターは以下で実行できます。 • WebSphere Liberty の任意の Java EE 7 対応バージョン。 • WebSphere Application Server traditional 9.0
IBM MQ 8.0	リソース・アダプターは、Java EE 7 に準拠した任意のバージョンの WebSphere Liberty で実行できます。 IBM MQ 8.0 リソース・アダプターは WebSphere Application Server traditional では実行できません。 WebSphere Application Server traditional に既にインストールされているリソース・アダプターを使用して、 IBM MQ 8.0 キュー・マネージャーに接続する必要があります。

関連概念

438 ページの『IBM MQ リソース・アダプターのサポートに関するステートメント』
アプリケーションとキュー・マネージャーの間の通信に使用する必要がある IBM MQ リソース・アダプターは、 Jakarta Messaging 3.0 API を使用するか、 JMS 2.0 API を使用するかによって異なります。

関連情報

IBM MQ のシステム要件

WebSphere Application Server から IBM MQ に対して作成される TCP/IP 接続の数の決定

共用会話機能を使用して、複数の会話で MQI チャネル・インスタンス (TCP/IP 接続としても知られる) を共用できるようになります。

このタスクについて

IBM MQ メッセージング・プロバイダー通常モードを使用する WebSphere Application Server 7 および WebSphere Application Server 8 の内部で実行されるアプリケーションは、この機能を自動的に使用します。 これは、同じ IBM MQ キュー・マネージャーに接続する同一のアプリケーション・サーバー・インスタンス内で実行されている複数のアプリケーションが、同じチャネル・インスタンスを共用できることを意味します。

単一のチャネル・インスタンス内で共用できる会話数は、IBM MQ チャネル・プロパティー **SHARECNV** によって決定されます。 サーバー接続チャネルのこのプロパティーのデフォルト値は 10 です。

WebSphere Application Server 7 および WebSphere Application Server 8 によって作成される会話数を監視することで、作成されるチャネル・インスタンスの数を決定することが可能になります。

IBM MQ メッセージング・プロバイダー・モードについて詳しくは、PROVIDERVERSION 通常モードを参照してください。

関連概念

共用会話の使用
共用会話が使用できる環境では、会話は MQI チャネル・インスタンスを共用できます。

316 ページの『IBM MQ classes for JMS における TCP/IP 接続の共用』
MQI チャネルの複数インスタンスが、単一の TCP/IP 接続を共用するようにできます。

JMS 接続ファクトリー

WebSphere Application Server の内部で実行され、IBM MQ メッセージング・プロバイダーの接続ファクトリーを使用して接続とセッションを作成するアプリケーションは、接続ファクトリーから作成されたすべての JMS 接続、および JMS 接続から作成されたすべての JMS セッションに対してアクティブな会話を持ちます。

接続ファクトリーから作成されたすべての JMS 接続に対して 1 つの会話

各 JMS 接続ファクトリーには、関連付けられた接続プールがあり、それは空きプールとアクティブ・プールの 2 つのセクションに分けられます。 プールは両方とも最初は空です。

アプリケーションが接続ファクトリーから JMS 接続を作成するとき、WebSphere Application Server は、空きプール内に JMS 接続があるかどうかを調べます。 ある場合、それはアクティブ・プールに移動され、アプリケーションに渡されます。 ない場合は、新しい JMS 接続が作成され、アクティブ・プールに入れられて、アプリケーションに戻されます。 接続ファクトリーから作成できる接続の最大数は、接続ファクトリー接続プール・プロパティー **Maximum connections** によって指定されます。 このプロパティーのデフォルト値は 10 です。

アプリケーションが JMS 接続を使用し終えてそれを閉じると、その接続はアクティブ・プールから空きプールに移動され、そこで再利用のために使用可能になります。 接続プール・プロパティー **Unused timeout** は、 JMS 接続が切断される前に空きプールに留まることができる期間を定義します。 このプロパティーのデフォルト値は 1800 秒 (30 分) です。

JMS 接続が最初に作成されたときに、WebSphere Application Server と IBM MQ の間の会話が開始されます。 会話は、空きプールの **Unused timeout** プロパティーの値を超えたときに接続が閉じられるまでアクティブのままです。

JMS 接続から作成されたすべての JMS セッションに対して 1 つの会話

IBM MQ メッセージング・プロバイダー接続ファクトリーから作成されるすべての JMS 接続には、関連付けられた JMS セッション・プールがあります。 そのセッション・プールは、接続プールと同様に機能します。 単一の JMS 接続から作成できる JMS セッションの最大数は、接続ファクトリー・セッション・プール・プロパティー **Maximum connections** によって決定されます。 このプロパティーのデフォルト値は 10 です。

会話は、JMS セッションが最初に作成されたときに開始されます。会話は、セッション・プールの **Unused timeout** プロパティーの値よりも長くフリー・プールに残っているため、JMS セッションが閉じられるまでアクティブのままです。

SHARECNV プロパティーの値の計算

単一の接続ファクトリーから IBM MQ への会話の最大数は、以下の数式を使用して計算できます。

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

この数の会話が発生できるように作成されるチャネル・インスタンスの数は、以下の計算で算出できます。

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

この計算に剰余があるなら、切り上げることができます。

接続プール **Maximum connections** およびセッション・プール **Maximum connections** ・プロパティーのデフォルト値を使用する単純接続ファクトリーの場合、この接続ファクトリーの WebSphere Application Server と IBM MQ の間に存在できる会話の最大数は以下のとおりです。

```
Maximum number of conversations =
```

```
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

以下に例を示します。

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

この接続ファクトリーが、 **SHARECNV** プロパティーが 10 に設定されているチャネルを使用して IBM MQ に接続する場合、この接続ファクトリー用に作成されるチャネル・インスタンスの最大数は以下のようになります。

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

以下に例を示します。

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

アクティベーション・スペック

アクティベーション・スペックを使用するように構成されたメッセージ駆動型 Bean アプリケーションは、アクティベーション・スペックが JMS の転送先をモニターし、メッセージ駆動型 Bean インスタンスの実行に使用されるすべてのサーバー・セッションがメッセージを処理するように、会話をアクティブにします。

アクティベーション・スペックを使用するよう構成されたメッセージ駆動型 Bean アプリケーションで、以下の会話がアクティブです。

- 適切なメッセージについてアクティベーション・スペックが JMS の転送先をモニターする 1 つの会話。この会話は、アクティベーション・スペックが開始するとすぐに開始され、アクティベーション・スペックが停止するまでアクティブのままになります。
- メッセージ駆動型 Bean インスタンスを実行してメッセージを処理するために使用されたサーバー・セッションごとに 1 つの会話。

アクティベーション・スペック拡張プロパティー **Maximum server sessions** は、特定のアクティベーション・スペックに対して一度にアクティブにできるサーバー・セッションの最大数を指定します。 このプロパティーのデフォルト値は 10 です。 サーバー・セッションは、必要に応じて作成され、アクティベーション・スペック拡張プロパティー **Server session pool timeout** で指定された期間アイドル状態であった場合はクローズされます。 このプロパティーのデフォルト値は 300000 ミリ秒 (5 分) です。

会話は、サーバー・セッションが作成された時点で開始され、アクティベーション・スペックが停止された時点、またはサーバー・セッションがタイムアウトになった時点のいずれかに停止します。

つまり、単一のアクティベーション・スペックから IBM MQ への会話の最大数は、以下の公式を使用して計算できます。

```
Maximum number of conversations = Maximum server sessions + 1
```

この数の会話が発生できるように作成されるチャネル・インスタンスの数は、以下の計算によって検出できます。

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

この計算に剰余があるなら、切り上げることができます。

Maximum server sessions プロパティーのデフォルト値を使用する単純なアクティベーション・スペックの場合、このアクティベーション・スペックの WebSphere Application Server と IBM MQ の間に存在できる会話の最大数は、以下のように計算されます。

```
Maximum number of conversations = Maximum server sessions + 1
```

以下に例を示します。

```
    = 10 + 1
    = 11
```

このアクティベーション・スペックが、 **SHARECNV** プロパティーが 10 に設定されたチャネルを使用して IBM MQ に接続する場合、作成されるチャネル・インスタンスの数は以下のように計算されます。

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

以下に例を示します。

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Application Server Facilities (ASF) モードで実行されるリスナー・ポート

メッセージ駆動型 Bean アプリケーションによって使用される ASF モードで動作するリスナー・ポートは、サーバー・セッションごとに会話を作成します。 1 つは適切なメッセージの宛先をモニターし、もう 1 つはメッセージ駆動型 Bean インスタンスを実行してメッセージを処理します。 各リスナー・ポートの会話の数は、セッションの最大数から計算できます。

デフォルトで、リスナー・ポートは 1.1 仕様の一部として ASF モードで実行されます。この仕様は、アプリケーション・サーバーがメッセージを検出し、処理のためにメッセージ駆動型 Bean に配信するために使用するメカニズムを定義します。 このデフォルト・モードの操作でリスナー・ポートを使用するようセットアップされたメッセージ駆動型 Bean アプリケーションは、以下のように会話を作成します。

リスナー・ポートが適切なメッセージの宛先をモニターするための 1 つの会話
: リスナー・ポートは、JMS 接続ファクトリーを使用するよう構成されます。 リスナー・ポートが開始されると、接続ファクトリーの空きプールから JMS の接続に対して要求が行われます。 リスナー・ポートが停止すると、接続は空きプールに戻されます。 接続プールの使用方法およびこれが IBM MQ との会話の数に与える影響について詳しくは、502 ページの『JMS 接続ファクトリー』を参照してください。

メッセージ駆動型 Bean インスタンスを実行してメッセージを処理するために使用されたサーバー・セッションごとに 1 つの会話
: リスナー・ポート・プロパティー **Maximum sessions** は、特定のリスナー・ポートに対して一度にアクティブにできるサーバー・セッションの最大数を指定します。 このプロパティーのデフォルト値は 10 です。 サーバー・セッションは必要に応じて作成され、リスナー・ポートが使用している JMS 接続に関連付けられたセッション・プールから取得された JMS セッションを利用します。

メッセージ・リスナー・サービスのカスタム・プロパティー **SERVER.SESSION.POOL.UNUSED.TIMEOUT** で指定される期間サーバー・セッションでアイドル状態が続くと、セッションは閉じられ、使用された JMS セッションはセッション・プールの空きプールに戻されます。 JMS セッションは、必要になるまでセッション・プールの空きプールに残ります。または、セッション・プールの **Unused timeout** プロパティーの値より長く空きプールでアイドル状態であったために閉じられます。

セッション・プールの使用方法、および WebSphere Application Server と IBM MQ の間の会話の管理方法について詳しくは、502 ページの『JMS 接続ファクトリー』を参照してください。

メッセージ・リスナー・サービスのカスタム・プロパティー **SERVER.SESSION.POOL.UNUSED.TIMEOUT**。WebSphere Application Server 製品資料の「 リスナー・ポートのサーバー・セッション・プールのモニター 」を参照してください。

単一のリスナー・ポートから IBM MQ への会話の最大数の計算

単一のリスナー・ポートから IBM MQ への会話の最大数は、以下の公式を使用して計算できます。

```
Maximum number of conversations = Maximum sessions + 1
```

この数の会話が発生できるように作成されるチャネル・インスタンスの数は、以下の計算で算出できます。

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

この計算に剰余があるなら、切り上げることができます。

Maximum sessions プロパティーのデフォルト値を使用する単純なリスナー・ポートの場合、このリスナー・ポートの WebSphere Application Server と IBM MQ の間に存在できる会話の最大数は、以下のように計算されます。

```
Maximum number of conversations = Maximum sessions + 1
```

以下に例を示します。

```
    = 10 + 1
    = 11
```

このリスナー・ポートが、 **SHARECNV** プロパティーが 10 に設定されているチャネルを使用して IBM MQ に接続している場合、作成されるチャネル・インスタンスの数は以下のように計算されます。

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

以下に例を示します。

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Application Server Facilities (ASF) 以外のモードで実行されているリスナー・ポート

ASF 以外のモードで実行されるリスナー・ポートを、サーバー・セッションを使用してキュー宛先とトピック宛先をモニターするよう構成できます。 サーバー・セッションは、複数の会話を持つことができ、最大数はケースごとに計算できます。

リスナー・ポートは ASF 以外のモードで実行されるように構成でき、そうするとリスナー・ポートが JMS 宛先をモニターする方法が変わります。 非 ASF 操作モードのリスナー・ポートを使用するメッセージ駆動型 Bean アプリケーションは、メッセージ駆動型 Bean インスタンスの実行に使用されるすべてのサーバー・セッションに会話を作成して、メッセージを処理します。 リスナー・ポート・プロパティー「**最大セッション数**」は、特定のリスナー・ポートに対して同時にアクティブにできるサーバー・セッションの最大数を指定します。 このプロパティーのデフォルト値は 10 です。

非 ASF モードで実行されている場合、キュー宛先をモニターしているリスナー・ポートは、リスナー・ポート・プロパティー「**最大セッション**」で指定された数のサーバー・セッションを自動的に作成します。 これらのすべてのサーバー・セッションは、リスナー・ポートが使用している JMS 接続に関連付けられているセッション・プールから取得された JMS セッションを利用し、該当するメッセージがないかどうか継続的に JMS 宛先をモニターします。

リスナー・ポートが、トピック宛先をモニターするように構成されている場合、「**最大セッション**」の値は無視され、単一のサーバー・セッションが使用されます。

非 ASF モードで実行されているリスナー・ポートが使用するサーバー・セッションは、リスナー・ポートが停止するまでアクティブのままで、そのとき、使用されていた JMS セッションは、リスナー・ポートが使用していた JMS 接続用のセッション・プールの空きプールに戻されます。

セッション・プールの使用方法、および WebSphere Application Server と IBM MQ の間の会話の管理方法について詳しくは、502 ページの『JMS 接続ファクトリー』を参照してください。

WebSphere Application Server での ASF モードおよび非 ASF モードの操作について、および非 ASF モードを使用するようにリスナー・ポートを構成する方法について詳しくは、 ASF モードおよび非 ASF モードでのメッセージ処理を参照してください。

キュー宛先をモニターしている場合の会話の最大数の計算

非 ASF モードで実行されていてキュー宛先をモニターしている単一のリスナー・ポートから IBM MQ への会話の最大数は、以下の数式を使用して計算できます。

```
Maximum number of conversations = Maximum sessions
```

この数の会話が発生できるように作成されるチャネル・インスタンスの数は、以下の計算で算出できます。

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

この計算に剰余があるなら、切り上げることができます。

最大セッション数 プロパティーのデフォルト値を使用し、キュー宛先をモニターしている非 ASF モードで実行されている単純なリスナー・ポートの場合、このリスナー・ポートの WebSphere Application Server と IBM MQ の間に存在できる会話の最大数は以下のとおりです。

```
Maximum number of conversations = Maximum sessions
```

以下に例を示します。

```
= 10
```

このリスナー・ポートが、 **SHARECNV** プロパティーが 10 に設定されているチャネルを使用して IBM MQ に接続している場合、作成されるチャネル・インスタンスの数は以下のように計算されます。

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

以下に例を示します。

```
= 10 / 10
= 1
```

トピック宛先をモニターしている場合の会話の最大数の計算

非 ASF モードで実行されていて、トピック宛先をモニターするように構成されているリスナー・ポートの場合、リスナー・ポートから IBM MQ への会話の数は、次のようになります。

```
Maximum number of conversations = 1
```

この数の会話が発生できるように作成されるチャネル・インスタンスの数は、以下の計算で算出できます。

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

この計算に剰余があるなら、切り上げることができます。

最大セッション数 プロパティーのデフォルト値を使用し、トピック宛先をモニターしている非 ASF モードで実行されている単純なリスナー・ポートの場合、このリスナー・ポートの WebSphere Application Server と IBM MQ の間に存在できる会話の最大数は以下のとおりです。

```
Maximum number of conversations = Maximum sessions
```

以下に例を示します。

```
     = 10
```

このリスナー・ポートが、 **SHARECNV** プロパティーが 10 に設定されているチャネルを使用して IBM MQ に接続している場合、作成されるチャネル・インスタンスの数は以下のように計算されます。

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

以下に例を示します。

```
     = 10 / 10
     = 1
```

IBM MQ への WebSphere Application Server 接続を保護するための認証別名の構成

認証別名は、 IBM MQ への WebSphere Application Server 接続を保護するために使用できるユーザー名とパスワードの組み合わせにマップされます。 接続ファクトリーを認証別名で構成できます。

エンタープライズ・アプリケーションでの認証別名の使用

WebSphere Application Server の内部で実行されているエンタープライズ・アプリケーションが IBM MQ への JMS 接続を作成しようとすると、アプリケーションは、アプリケーション・サーバーの Java Naming Directory Interface (JNDI) リポジトリーから IBM MQ メッセージング・プロバイダー接続ファクトリー定義を検索します。

IBM MQ メッセージング・プロバイダー接続ファクトリーの定義がアプリケーション・サーバーの JNDI リポジトリー内で特定されると、以下のいずれかのメソッドが呼び出されます。

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

定義されている J2C 認証別名を使って接続ファクトリーが構成されている場合、その接続ファクトリーを使用して接続を作成したときに、その認証別名のユーザー名とパスワードを IBM MQ に受け渡せます。

接続ファクトリーおよび認証別名

IBM MQ メッセージング・プロバイダー接続ファクトリーには、IBM MQ キュー・マネージャーへの接続方法に関する情報が含まれています。 WebSphere Application Server の内部で実行されるエンタープライズ・アプリケーションは、接続ファクトリーを使用して IBM MQ への JMS 接続を作成できます。

WebSphere Application Server は JNDI を使用してアクセス可能なリポジトリーに接続ファクトリーの定義を保管します。 接続ファクトリーが作成されると JNDI 名が付与され、その接続ファクトリーが定義されているアプリケーション・サーバーの有効範囲 (セル、ノード、またはサーバーの有効範囲のいずれか) で一意に識別されます。

例えば、WebSphere Application Server セル有効範囲で定義された IBM MQ メッセージング・プロバイダー接続ファクトリーには、BINDINGS トランスポートを使用してキュー・マネージャー (`myQM`) に接続する方法に関する情報が含まれています。 この接続ファクトリーには、一意的に識別するための JNDI 名 `jms/myCF` が付けられます。

接続ファクトリーは、認証別名を使用するように構成することもできます。 認証別名はユーザー名とパスワードの組み合わせにマップされます。 接続ファクトリーの使用方法に応じて、認証別名のユーザー名とパスワードは、 JMS 接続の作成時に IBM MQ に受け渡される場合とされない場合があります。

重要 : IBM MQ 8.0 以前は、接続の作成時に、ユーザー名が IBM MQ に渡されたことだけを確認するために許可検査を実行した、デフォルトの IBM MQ オブジェクト権限マネージャー (OAM) が、キュー・マネージャーへのアクセス権限を保持していました。

指定されたパスワードを検証する検査は行われませんでした。 認証検査を実行し、ユーザー ID とパスワードが一致することを検証するために、IBM MQ チャネル・セキュリティー出口を作成する必要がありました。 これを行う方法について詳しくは、 975 ページの『チャネル・セキュリティー出口プログラム』を参照してください。

IBM MQ 8.0 以降、キュー・マネージャーがユーザー名に加えてパスワードも検査します。

接続ファクトリーの使用

以下のトピックには、直接および間接検索を使う接続ファクトリーの使用に関する情報が示されています。

- 511 ページの『直接検索を介した接続ファクトリーの使用』
- 511 ページの『間接検索を介した接続ファクトリーの使用』

CLIENT トランスポートの使用

CLIENT トランスポートを使用するように構成された接続ファクトリーは、キュー・マネージャーに接続するために使用する IBM MQ サーバー接続チャネル (SVRCONN) を指定する必要があります。

接続ファクトリーで使用するように構成されているチャネルの IBM MQ チャネル・エージェント・ユーザー ID (MCAUSER) プロパティーがブランクのままになっている場合、接続ファクトリーは直接検索または間接検索のどちらとも一緒に使用できます。

MCAUSER プロパティーがユーザー ID に設定されている場合、IBM MQ への接続を作成するために接続ファクトリーが使用されるときは、エンタープライズ・アプリケーションが直接検索または間接検索のどちらを使用しているかに関係なく、このユーザー ID が IBM MQ に受け渡されます。

要約表

以下の表に、BINDINGS トランスポートおよび CLIENT トランスポートがそれぞれ使用されるときに、どのユーザー ID が IBM MQ に受け渡されるかを要約しています。

表 *71. BINDINGS* モード

構成	**アプリケーションが `ConnectionFactory.createConnection()` を呼び出す**	**アプリケーションが `ConnectionFactory.createConnection(String username, String password)` を呼び出す**
アプリケーションのデプロイメント記述子に、接続ファクトリーへのリソース参照が含まれない	アプリケーション・サーバー・プロセスのユーザー ID が IBM MQ に受け渡されます。	`ConnectionFactory.createConnection(String username, String password)` メソッドに渡されたユーザー ID およびパスワードが、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には接続ファクトリーへのリソース参照が含まれ、**res-auth** プロパティーが「Application」に設定されている	アプリケーション・サーバー・プロセスのユーザー ID が IBM MQ に受け渡されます。	`ConnectionFactory.createConnection(String username, String password)` メソッドに渡されたユーザー ID およびパスワードが、IBM MQ に受け渡されます。

表 *71. BINDINGS* モード *(続き)*

構成	アプリケーションが **ConnectionFactory.createConnection()** を呼び出す	アプリケーションが **ConnectionFactory.createConnection(String username, String password)** を呼び出す
アプリケーションのデプロイメント記述子には接続ファクトリーへのリソース参照が含まれ、**res-auth** プロパティーが「Container」に設定されている	接続ファクトリーの認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。	接続ファクトリーの認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが「Container」に設定された接続ファクトリーへのリソース参照が含まれ、アプリケーションが認証別名によって構成されている	アプリケーションで使用するように構成された認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。	アプリケーションで使用するように構成された認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。

表 *72. CLIENT* モード

構成	アプリケーションが **ConnectionFactory.createConnection()** を呼び出す	アプリケーションが **ConnectionFactory.createConnection(String username, String password)** を呼び出す
アプリケーションのデプロイメント記述子には接続ファクトリーへのリソース参照が含まれず、接続ファクトリーは MCAUSER プロパティーが設定されていない IBM MQ チャネルを使用するように構成されている	アプリケーション・サーバー・プロセスのユーザー ID が IBM MQ に受け渡されます。	`ConnectionFactory.createConnection(String username, String password)` メソッドに渡されたユーザー ID およびパスワードが、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には接続ファクトリーへのリソース参照が含まれず、接続ファクトリーは MCAUSER プロパティーがユーザー ID に設定されている IBM MQ チャネルを使用するように構成されている	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Application* に設定された接続ファクトリーへのリソース参照が含まれ、接続ファクトリーは MCAUSER プロパティーが設定されていない IBM MQ チャネルを使用するように構成されている	アプリケーション・サーバー・プロセスのユーザー ID が IBM MQ に受け渡されます。	`ConnectionFactory.createConnection(String username, String password)` メソッドに渡されたユーザー ID およびパスワードが、IBM MQ に受け渡されます。

表 72. *CLIENT* モード (続き)

構成	アプリケーションが **ConnectionFactory.createConnection()** を呼び出す	アプリケーションが **ConnectionFactory.createConnection(String username, String password)** を呼び出す
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Application* に設定された接続ファクトリーへのリソース参照が含まれ、接続ファクトリーは MCAUSER プロパティーがユーザー ID に設定された IBM MQ チャネルを使用するように構成されている	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Container* に設定された接続ファクトリーへのリソース参照が含まれ、接続ファクトリーは MCAUSER プロパティーが設定されていない IBM MQ チャネルを使用するように構成されている	接続ファクトリーの認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。	接続ファクトリーの認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Container* に設定された接続ファクトリーへのリソース参照が含まれ、接続ファクトリーは MCAUSER プロパティーがユーザー ID に設定された IBM MQ チャネルを使用するように構成されている	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Container* に設定された接続ファクトリーへのリソース参照が含まれ、アプリケーションは認証別名で構成され、接続ファクトリーは MCAUSER プロパティーが設定されていない IBM MQ チャネルを使用するように構成されている	アプリケーションで使用するように構成された認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。	アプリケーションで使用するように構成された認証別名に指定されたユーザー ID とパスワードが、IBM MQ に受け渡されます。
アプリケーションのデプロイメント記述子には、**res-auth** プロパティーが *Container* に設定された接続ファクトリーへのリソース参照が含まれ、アプリケーションは認証別名で構成され、接続ファクトリーは MCAUSER プロパティーがユーザー ID に設定された IBM MQ チャネルを使用するように構成されている	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。	接続ファクトリーで使用するように構成されている IBM MQ チャネルの MCAUSER プロパティーによって指定されたユーザー ID が、IBM MQ に受け渡されます。

直接検索を介した接続ファクトリーの使用

IBM MQ メッセージング・プロバイダー接続ファクトリーが定義されると、エンタープライズ・アプリケーションは接続ファクトリー定義を検索し、それを使用して IBM MQ キュー・マネージャーへの JMS 接続を作成することができます。 これは、直接検索を介して行えます。

直接検索を使用するため、エンタープライズ・アプリケーションは以下のメソッド呼び出しを行って、アプリケーション・サーバーの JNDI リポジトリーに接続します。

```
InitialContext ctx = new InitialContext();
```

いったん JNDI リポジトリーに接続すると、エンタープライズ・アプリケーションは以下のように、接続ファクトリーの JNDI 名を使用して接続ファクトリー定義を識別します。

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

注 :

- アプリケーション開発者は、エンタープライズ・アプリケーションを開発するときに、必要な接続ファクトリーの JNDI 名を把握しておく必要があります。 JNDI 名はアプリケーション内部にハードコーディングされているので、JNDI 名が変更された場合、アプリケーションを書き直し、再デプロイする必要があります。
- このような方法で接続ファクトリー定義が使用される場合、認証別名 (接続ファクトリーが使用するように構成されたもの) に指定されたユーザー名とパスワードは、IBM MQ に受け渡されません。 これは、不正なアプリケーションが接続ファクトリーを識別し、それを使用してセキュア IBM MQ システムに接続することを防ぐためです。

IBM MQ に受け渡されるユーザー名とパスワードは、接続ファクトリーから JMS 接続を作成するために使用されたメソッドに応じて決まります。

アプリケーションが次のメソッドを使用して JMS 接続を作成するとします。

```
ConnectionFactory.createConnection()
```

すると、デフォルトのユーザー ID が IBM MQ に受け渡されます。 これは、エンタープライズ・アプリケーションを実行しているアプリケーション・サーバーの始動時に使用したユーザー名とパスワードです。

あるいは、アプリケーションは次のメソッドを呼び出して JMS 接続を作成することもできます。

```
ConnectionFactory.createConnection(String username, String password)
```

アプリケーションが接続ファクトリーの直接検索を実行してからこのメソッドを呼び出した場合、`createConnection()` メソッドに渡されたユーザー名とパスワードが IBM MQ に受け渡されます。

重要 : IBM MQ 8.0 以前は、受け渡されたユーザー名を確認することのみを目的として、許可検査を処理した IBM MQ が、キュー・マネージャーへのアクセス権限を保持していました。

パスワードに関する検査は行われていませんでした。 認証検査を実行し、ユーザー名とパスワードが有効であることを検証するために、IBM MQ チャネル・セキュリティー出口を作成する必要がありました。 これを行う方法について詳しくは、975 ページの『チャネル・セキュリティー出口プログラム』を参照してください。

IBM MQ 8.0 以降、キュー・マネージャーがユーザー名に加えてパスワードも検査します。

間接検索を介した接続ファクトリーの使用

エンタープライズ・アプリケーションを作成しているときに、接続ファクトリーの JNDI 名がわからない場合、またはアプリケーションが別の JNDI 名 (インストール先のアプリケーション・サーバーに応じて決まる) を持つ別の接続ファクトリーを使用して別のアプリケーション・サーバーにインストールされる場合、リソース参照を使用して接続ファクトリーを検索できます。 これは、間接検索を介して行えます。

例

jms/myCF を使用して接続ファクトリーを直接検索するのではなく、エンタープライズ・アプリケーションにはローカル JNDI 名 jms/myResourceReferenceCF を持つリソース参照が含まれています。

この JNDI 名を使用するため、アプリケーションは、次のように、直接検索を実行する場合と同じ方法でアプリケーション・サーバーの JNDI リポジトリーに接続します。

```
InitialContext ctx = new InitialContext();
```

この場合、アプリケーションは jms/myCF を直接識別する代わりに、リソース参照の JNDI 名を識別します。

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

エンタープライズ・アプリケーションが間接検索を実行していることをアプリケーション・サーバーに通知するには、ローカル JNDI 名に java:comp/env 接頭部を付ける必要があります。

アプリケーションがデプロイされると、ユーザーは JNDI リソース参照の名前 jms/myResourceReferenceCF を、アプリケーションが既に作成した接続ファクトリーの JNDI 名 jms/myCF にマップします。

アプリケーションが実行されると、アプリケーション・サーバーが jms/myCF にマップするローカル JNDI 名を使用して、 JMS 接続ファクトリーを検索します。 次に、アプリケーションはこの接続ファクトリーを使用して IBM MQ への接続を作成します。

認証別名と間接検索

リソース参照では、指定された接続ファクトリーの動作を変更する追加のプロパティーを定義することもできます。 リソース参照のプロパティーの 1 つは、**res-auth** です。 このプロパティーの値は、エンタープライズ・アプリケーションが、IBM MQ への接続を作成するときにリソース参照がマップする接続ファクトリーの認証別名 (認証別名が定義されている場合) を使用するか、アプリケーションが独自のユーザー名とパスワードを指定しているかを指定します。

このプロパティーのデフォルト値は *Application* です。 これは、JMS 接続の作成時に、キュー・マネージャーに受け渡されるユーザー名とパスワードがアプリケーション自体によって決定されることを意味します。 リソース参照がマップする接続ファクトリーの認証別名は使用されません。

アプリケーションは、次のいずれかのメソッドを使用して JMS 接続を作成できます。

- ConnectionFactory.createConnection()
- ConnectionFactory.createConnection(String username, String password)

アプリケーションが ConnectionFactory.createConnection() を使用し、**res-auth** が *Application* に設定されている場合、デフォルトのユーザー ID が IBM MQ に受け渡されます。 これは、エンタープライズ・アプリケーションを実行しているアプリケーション・サーバーの始動時に使用したユーザー名とパスワードです。

アプリケーションが ConnectionFactory.createConnection(String username, String password)を使用し、 **res-auth** が *Application* に設定されている場合、メソッドに渡されるユーザー名とパスワードは IBM MQ に送信されます。

接続作成時にリソース参照がマップする接続ファクトリーに定義された認証別名を使用するには、**res-auth** プロパティーに値 *Container* を設定する必要があります。 アプリケーションが JMS 接続を作成すると、createConnection 呼び出しにユーザー名とパスワードが指定されていたとしても、認証別名の詳細が使用されます。

間接検索使用時の認証別名のオーバーライド

res-auth プロパティーが *Container* に設定されたリソース参照をアプリケーションが使用する場合、JMS 接続の作成時に使用される認証別名をオーバーライドできます。

認証別名をオーバーライドするには、リソース参照に **authDataAlias** という名前の追加のプロパティーを指定する必要があります。このプロパティーは、アプリケーションのデプロイ先となるアプリケーション・サーバー環境内に既に作成されている既存の認証別名にマップされます。 このプロパティーは、 IBM によって提供される Rational® ツールを使用して作成されるすべてのリソース参照に対して指定できます。

このメソッドを使用して、間接的に検索されている JMS 接続ファクトリーの使用時に、別の認証別名を使用できます。 指定された認証別名が存在しない場合、エンタープライズ・アプリケーションのインストール後に新しい認証別名を指定できます。 詳しくは、WebSphere Application Server 製品資料の リソース参照 を参照してください。

WebSphere Application Server 8.5.5 の関連情報
リソース参照
WebSphere Application Server 8.0 の関連情報
リソース参照
WebSphere Application Server 7.0 の関連情報
リソース参照

WebSphere Application Server のクラスターを使用する場合のメッセージ駆動型 Bean のワークロード・バランシング

WebSphere Application Server 7.0 と WebSphere Application Server 8.0 のクラスターにデプロイされ、IBM MQ メッセージング・プロバイダーの通常モードで実行されるように構成されたメッセージ駆動型 Bean アプリケーションを使用する場合、クラスター・メンバーのいずれかがメッセージの大半を処理します。 複数のクラスター・メンバーにメッセージの処理を分散するために、クラスター・メンバーのワークロードのバランスを取ることができます。

IBM MQ には、 **Asynchronous consume** という機能が組み込まれています。これにより、アプリケーションは、 **MQCB** および **MQCTL** という API を使用して、キューからメッセージを非同期にコンシュームできます。

WebSphere Application Server 7.0 および WebSphere Application Server 8.0 の内部で実行され、IBM MQ メッセージング・プロバイダーの通常モードを使用するメッセージ駆動型 Bean アプリケーションは、自動的にこの機能を使用します。 アプリケーションが始動すると、 **MQCB** を呼び出すことによってモニターするように構成されている JMS 宛先に非同期コンシューマーがセットアップされます。 次に **MQCTL** API が呼び出され、アプリケーションが JMS 宛先からのメッセージを受信する準備ができたことを示します。

メッセージ駆動型 Bean アプリケーションが WebSphere Application Server クラスターに実装されると、各クラスター・メンバーは、メッセージ駆動型 Bean がメッセージをモニターする JMS 宛先に対して非同期コンシューマーをセットアップします。 JMS 宛先をホストする IBM MQ キュー・マネージャーは、 JMS 宛先に処理に適したメッセージがある場合に、クラスター・メンバーに通知する役割を担います。

WebSphere Application Server が IBM MQ キュー・マネージャーに接続する場合、 JMS 宛先に到着するメッセージは、その JMS 宛先に登録されているすべての非同期コンシューマーにより均等に分散されます。 WebSphere Application Server 7.0 および WebSphere Application Server 8.0 のクラスターの内部にデプロイされたメッセージ駆動型 Bean アプリケーションの場合、これは、メッセージがクラスター・メンバー間でより均等に分散されることを意味します。

関連タスク
JMS **PROVIDERVERSION** プロパティーの構成

IBM MQ Headers パッケージの使用

IBM MQ Headers パッケージは、メッセージの IBM MQ ヘッダーを操作するために使用できる、ヘルパー・インターフェースとクラスのセットを提供します。 通常、IBM MQ Headers パッケージを使用するのは、

(プログラマブル・コマンド・フォーマット (PCF) メッセージを使用して) コマンド・サーバーで管理サービスを実行するためです。

このタスクについて

IBM MQ ヘッダー・パッケージは、`com.ibm.mq.headers` パッケージと `com.ibm.mq.headers.pcf` パッケージにあります。 この機能は、 Java アプリケーションで使用するために IBM MQ が提供する 2 つの代替 API の両方に使用できます。

- IBM MQ classes for Java (別名 IBM MQ Base Java)。
- IBM MQ classes for Java Message Service (別名 IBM MQ classes for JMS、IBM MQ JMS)。

IBM MQ Base Java アプリケーションは通常は MQMessage オブジェクトを操作します。 Headers サポート・クラスは、ネイティブで IBM MQ Base Java インターフェースを理解できるので、これらのオブジェクトと直接対話できます。

IBM MQ JMS では、メッセージのペイロードは、通常は、DataInput および DataOutput ストリームを使用して操作できるストリングまたはバイト配列オブジェクトです。 IBM MQ Headers パッケージは、これらのデータ・ストリームと対話するために使用でき、 IBM MQ JMS アプリケーションで送受信する MQ メッセージを操作するのに適しています。

そのため、IBM MQ Headers パッケージは IBM MQ Base Java パッケージへの参照を含んでいますが、 同時に IBM MQ JMS アプリケーション内での使用も意図されているので、 Java Platform, Enterprise Edition (Java EE) 環境内での使用に適しています。

IBM MQ Headers パッケージの一般的な使用方法は、例えば以下のような理由でプログラマブル・コマンド・フォーマット (PCF) の管理メッセージを操作することです。

- IBM MQ リソースに関する詳細にアクセスするため。
- キューの深さをモニターするため。
- キューへのアクセスを禁止するため。

IBM MQ JMS API で PCF メッセージを使用することにより、 IBM MQ Base Java API を使用することなく、この種のアプリケーション中心リソースの管理を Java EE アプリケーション内から実行できます。

手順

- IBM MQ Headers パッケージを使用して IBM MQ classes for Java のメッセージ・ヘッダーを操作するには、514 ページの『IBM MQ classes for Java の使用』を参照してください。
- IBM MQ Headers パッケージを使用して IBM MQ classes for JMS のメッセージ・ヘッダーを操作するには、515 ページの『IBM MQ classes for JMS の使用』を参照してください。

IBM MQ classes for Java の使用

IBM MQ classes for Java アプリケーションは通常は MQMessage オブジェクトを操作します。 Headers サポート・クラスは、ネイティブで IBM MQ classes for Java インターフェースを理解できるので、これらのオブジェクトと直接対話できます。

このタスクについて

IBM MQ には、IBM MQ Headers パッケージを IBM MQ Base Java API (IBM MQ classes for Java) で使用する方法を示すいくつかのサンプル・アプリケーションが用意されています。

サンプルは、以下の 2 つの事柄を示しています。

- PCF メッセージを作成して管理アクションを実行し、応答メッセージを解析する方法。
- IBM MQ classes for Java を使用してその PCF メッセージを送信する方法。

使用しているプラットフォームによって、これらのサンプルは、`samples` 内の `pcf` ディレクトリーまたは IBM MQ インストールの `tools` ディレクトリーの下でインストールされます (355 ページの『IBM MQ classes for Java のインストール・ディレクトリー』 を参照)。

手順

1. 管理アクションを実行し、応答メッセージを解析するために、PCF メッセージを作成します。
2. IBM MQ classes for Java を使用してその PCF メッセージを送信します。

関連概念

383 ページの『IBM MQ classes for Java を使用した IBM MQ メッセージ・ヘッダーの処理』
さまざまなタイプのメッセージ・ヘッダーを表す Java クラスが提供されています。 2 つのヘルパー・クラスも提供されています。

388 ページの『IBM MQ classes for Java での PCF メッセージの処理』
Java クラスは、PCF 構造化メッセージを作成および構文解析し、PCF 要求の送信と PCF 応答の収集に役立てるために提供されています。

IBM MQ classes for JMS の使用

IBM MQ Headers を IBM MQ classes for JMS で使用するには、IBM MQ classes for Java の場合と同じ重要な手順を実行します。 IBM MQ Headers パッケージおよび IBM MQ classes for Java の場合と同じサンプル・コードを使用して、PCF メッセージを作成し、応答をまったく同じ方法で解析することができます。

このタスクについて

IBM MQ API を使用して PCF メッセージを送信するには、 メッセージ・ペイロードを JMS Bytes Message に書き込み、標準の JMS API を使用して送信する必要があります。 唯一の考慮事項として、メッセージには JMS RFH2 ヘッダーや MQMD に特定の値を指定した他のヘッダーを含めてはいけません。

PCF メッセージを送信するには、以下のステップを実行します。 PCF メッセージの作成方法、および応答メッセージからの情報の抽出方法は、IBM MQ classes for Java の場合と同じです (514 ページの『IBM MQ classes for Java の使用』 を参照)。

手順

1. SYSTEM.ADMIN.COMMAND.QUEUE を表す JMS キュー宛先を作成します。

   IBM MQ JMS アプリケーションは、PCF メッセージを SYSTEM.ADMIN.COMMAND.QUEUE。このキューを表す JMS 宛先オブジェクトにアクセスする必要があります。 宛先には以下のプロパティーを設定する必要があります。

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   WebSphere Application Server を使用する場合は、これらのプロパティーを宛先のカスタム・プロパティーとして定義する必要があります。

   アプリケーション内から宛先をプログラムで作成するには、以下のコードを使用します。

   ```
   Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
   ((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
       WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ((MQQueue) q1).setMQMDWriteEnabled(true);
   ```

2. PCF メッセージを、適切な MQMD 値を含む JMS Bytes メッセージに変換します。

   JMS Bytes メッセージを作成し、それに PCF メッセージを書き込む必要があります。 応答キューを作成する必要がありますが、固有の設定は必要ありません。

   以下のサンプル・コード・スニペットは、JMS Bytes メッセージを作成して、それに com.ibm.mq.headers,pcf.PCFMessage オブジェクトを書き込む方法を示しています。 PCFMessage オブジェクト (pcfCmd) は、IBM MQ Headers パッケージを使用して、事前に作成されています (PCFMessage をロードするパッケージが com.ibm.mq.headers.pcf.PCFMessage であることに注目してください)。

   ```
    // create the JMS Bytes Message
   final BytesMessage msg = session.createBytesMessage();
   ```

```
// Create the wrapping streams to put the bytes into the message payload
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutput dataOutput = new DataOutputStream(baos);

// Set the JMSReplyTo so the answer comes back
msg.setJMSReplyTo(new MQQueue("adminResp"));

// write the pcf into the stream
pcfCmd.write(dataOutput);
baos.flush();
msg.writeBytes(baos.toByteArray());

// we have taken control of the MD, so need to set all
// flags in the MD that we require - main one is the format
msg.setJMSPriority(4);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
  CMQC.MQPER_NOT_PERSISTENT);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
    CMQC.MQRO_PASS_CORREL_ID);
msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

// and send the message
sender.send(msg);
```

3. メッセージを送信し、標準の JMS API を使用して応答を受信します。
4. 応答メッセージを処理するために PCF メッセージに変換します。
   応答メッセージを取り出して、PCF メッセージとして処理するには、以下のコードを使用します。

```
 // Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

関連概念

146 ページの『JMS メッセージ』
JMS メッセージは、ヘッダー、プロパティー、および本体で構成されます。 JMS では、5 つのタイプのメッセージ本体を定義します。

IBM i Java および JMS を使用した IBM i での IBM MQ のセットアップ

このトピックの集まりにより、CL コマンドまたは Qshell 環境を使って、IBM i 上の Java と JMS を含む IBM MQ を設定およびテストする方法の概要が提供されます。

注 :

- IBM MQ 8.0 以降、`ldap.jar`、`jndi.jar`、および `jta.jar` は JDK の一部となっています。
- JM 3.0 V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 Jakarta Messaging 3.0 は新規アプリケーションの開発用にサポートされています。 IBM MQ 9.3.0 は、既存のアプリケーションに対して JMS 2.0 を引き続きサポートします。 同じアプリケーションで Jakarta Messaging 3.0 API と JMS 2.0 API の両方を使用することはサポートされていません。 詳しくは、 Using IBM MQ classes for JMS/Jakarta Messaging を参照してください。

CL コマンドの使用

設定する CLASSPATH は、 MQ ベース Java、JNDI を使用する JMS、および JNDI を使用しない JMS でテストするためのものです。

/home/Userprofile ディレクトリーの下で .profile ファイルを使用しない場合は、以下のシステム・レベル環境変数を設定する必要があります。 **WRKENVVAR** コマンドを使用して、それらが設定されていることを確認できます。

1. システム全体の環境変数を表示するには、コマンド **WRKENVVAR LEVEL(*SYS)** を発行します。
2. ジョブに特定の環境変数を表示するには、コマンド **WRKENVVAR LEVEL(*JOB)** を発行します。
3. CLASSPATH が設定されていない場合は、以下のコマンドを発行します。

JM 3.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0

```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. QIBM_MULTI_THREADED が設定されていない場合は、以下のコマンドを発行します。

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. QIBM_USE_DESCRIPTOR_STDIO が設定されていない場合は、以下のコマンドを発行します。

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. QSH_REDIRECTION_TEXTDATA が設定されていない場合は、以下のコマンドを発行します。

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

qshell 環境の使用

QSHELL 環境を使用する場合は、 /home/Userprofile ディレクトリーに .profile をセットアップすることができます。 詳しくは、Qshell Interpreter (qsh) の資料を参照してください。

.profile で以下を指定します。 CLASSPATH ステートメントは、単一行にするか、示されているように \ 文字を使用して別々の行に分離する必要があることに注意してください。

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

コマンド **DSPLIBL** を発行して、QMQMJAVA ライブラリーがライブラリー・リストに含まれていることを確認します。

QMQMJAVA ライブラリーがリストに含まれていない場合は、コマンド **ADDLIBLE LIB(QMQMJAVA)** を使用してそれを追加してください。

IBM i Java を使って IBM i 上で IBM MQ をテスト

MQIVP サンプル・プログラムを使用して Java で IBM MQ をテストする方法。

IBM MQ ベースの Java のテスト

以下の手順を実行します。

1. 以下のコマンドを発行して、キュー・マネージャーが開始されていること、およびキュー・マネージャーの状態が ACTIVE であることを確認します。

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 以下のコマンドを発行して、JAVA.CHANNEL サーバー接続チャネルが作成されていることを確認します。

   ```
   WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

   a. JAVA.CHANNEL が存在しない場合は、以下のコマンドを発行します。

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
   ```

3. **WRKMQMLSR** コマンドを発行して、ポート 1414 または使用しているいずれかのポートで、キュー・マネージャー・リスナーが実行していることを確認します。

   a. キュー・マネージャーに対して開始されたリスナーがない場合は、以下のコマンドを発行します。

   ```
   STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
   ```

MQIVP サンプル・テスト・プログラムの実行

1. コマンド STRQSH を発行して、コマンド・ラインから qshell を開始します。
2. **export** コマンドを発行してから以下のように **cd** コマンドを発行して、正しい CLASSPATH が設定されていることを確認します。

   ```
   cd /qibm/proddata/mqm/java/samples/wmqjava/samples
   ```

3. 以下のコマンドを発行して、**java** プログラムを実行します。

```
java MQIVP
```

以下を指定するためのプロンプトが出されたら、ENTER キーを押すことができます。

- 接続のタイプ
- IP アドレス
- キュー・マネージャー名

それにより、デフォルト値が使用されます。 これは、QMQMJAVA ライブラリー内にある、製品のバインディングを検証します。

以下に示す例のような出力を受け取ります。 著作権文は、使用する製品のバージョンによって異なることに注意してください。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM MQ Java クライアント接続のテスト

以下を指定する必要があります。

- 接続タイプ
- IP アドレス
- ポート
- サーバー接続チャネル
- キュー・マネージャー

以下に示す例のような出力を受け取ります。 著作権文は、使用する製品のバージョンによって異なることに注意してください。

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
```

```
Press Enter to continue ...>
$
```

IBM i JMS を使って IBM i 上で IBM MQ をテスト

JNDI を使用する場合と使用しない場合の JMS での IBM MQ のテスト方法

IVTRun サンプルを使用した、JNDI なしの JMS のテスト

以下の手順を実行します。

1. 以下のコマンドを発行して、キュー・マネージャーが開始されていること、およびキュー・マネージャーの状態が ACTIVE であることを確認します。

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. **STRQSH** コマンドを発行して、コマンド・ラインから qshell を開始します。
3. 以下のように **cd** コマンドを使用して、ディレクトリーを変更します。

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. スクリプト・ファイルを次のように実行します。

   ```
   IVTRun -nojndi [-m qmgrname]
   ```

以下に示す例のような出力を受け取ります。 著作権文は、使用する製品のバージョンによって異なることに注意してください。

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
```

```
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

JNDI なしの IBM MQ JMS クライアント・モードのテスト

以下の手順を実行します。

1. 以下のコマンドを発行して、キュー・マネージャーが開始されていること、およびキュー・マネージャーの状態が ACTIVE であることを確認します。

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. 以下のコマンドを発行して、サーバー接続チャネルが作成されていることを確認します。

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. **WRKMQMLSR** コマンドを発行して、リスナーが正しいポートに対して開始したことを確認します。
4. **STRQSH** コマンドを発行して、コマンド・ラインから qshell を開始します。
5. **export** コマンドを発行して、CLASSPATH が正しいことを確認します。
6. 以下のように **cd** コマンドを使用して、ディレクトリーを変更します。

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

7. スクリプト・ファイルを次のように実行します。

   ```
   IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
   ```

以下に示す例のような出力を受け取ります。 著作権文は、使用する製品のバージョンによって異なることに注意してください。

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
```

```
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

JNDI 使用の IBM MQ JMS のテスト

以下のコマンドを発行して、キュー・マネージャーが開始されていること、およびキュー・マネージャーの状態が ACTIVE であることを確認します。

```
WRKMQM MQMNAME(QMGRNAME)
```

IVTRun サンプル・テスト・スクリプトの使用

以下の手順を実行します。

1. JMSAdmin.config ファイルに適切な変更を加えます。 このファイルを編集するには、IBM i コマンド行から **EDTF** (ファイルの編集) コマンドを使用します。

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. LDAP for Weblogic を使用するには、以下からコメントを削除します。

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
   ```

   b. LDAP for WebSphere Application Server を使用するには、以下からコメントを削除します。

   ```
   INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
   ```

   c. ファイル・システムをテストするには、以下からコメントを削除します。

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   ```

   d. 適切な行からコメントを削除して、正しい PROVIDER_URL が選択されていることを確認します。

   e. # 記号を使用して、他のすべての行をコメント化します。

   f. すべての変更を行った後に、**F2=Save** および **F3=Exit** を押します。

2. **STRQSH** コマンドを発行して、コマンド・ラインから qshell を開始します。
3. **export** コマンドを発行して、CLASSPATH が正しいことを確認します。
4. 以下のように **cd** コマンドを使用して、ディレクトリーを変更します。

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

5. **IVTSetup** コマンドを発行して **IVTSetup** スクリプトを開始し、管理対象オブジェクト (*MQQueueConnectionFactory* と *MQQueue*) を作成します。

6. 以下のコマンドを発行して、IVTRun スクリプトを実行します。

```
IVTRun -url providerURL [-icf initCtxFact]
```

以下に示す例のような出力を受け取ります。 著作権文は、使用する製品のバージョンによって異なることに注意してください。

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Java Maven リポジトリーを使用したアプリケーション開発

Maven リポジトリーを使用して依存関係を自動的にインストールすることにより、IBM MQ 用の Java アプリケーションを開発する場合、IBM MQ インターフェースを使用する前に、明示的に何かをインストールする必要はありません。

Maven 中央リポジトリー

Maven はアプリケーション作成用のツールであり、アプリケーションがアクセスできる成果物を保持するためのリポジトリーも提供します。

Maven リポジトリー (または中央リポジトリー) には、JAR ファイルなどのファイルが複数の異なるバージョンを持つことを可能にする構造があります。それらのファイルは、既知の命名メカニズムを使用して簡単に検出できるようになります。 その後、ビルド・ツールはそれらの名前を使用して、アプリケーションのための依存関係を動的にプルします。 ビルド・ツールとして Maven を使用する場合、POM ファイルと呼ばれるアプリケーションの定義では、依存関係に名前を付け、ビルド・プロセスはそこから何を行うかを認識します。

IBM MQ クライアント・ファイル

IBM MQ Java クライアント・インターフェースのコピーは、`com.ibm.mq GroupId` の下の中央リポジトリーで使用できます。 `com.ibm.mq.jakarta.client.jar` ファイル (Jakarta Messaging 3.0) および `com.ibm.mq.allclient.jar` ファイル (JMS 2.0) を見つけることができます。 これらのファイルは通常、スタンドアロン・プログラムに使用されます。 `wmq.jakarta.jmsra.rar` ファイル (Jakarta Messaging 3.0) および `wmq.jmsra.rar` ファイル (JMS 2.0) もあります。これらは、 Java EE アプリケーション・サーバーで使用するためのものです。 `jakarta.client.jar` と `allclient.jar` の両方に、IBM MQ classes for JMS と IBM MQ classes for Java が含まれています。

重要 : Apache Maven アセンブリー・プラグインの *jar-with-dependencies* 形式を使用して、IBM MQ 再配置可能 JAR ファイルを含むアプリケーションを構築することはサポートされていません。

maven コマンドによって処理される `pom.xml` ファイルでは、以下の例に示すように、これらの JAR ファイルの依存関係を追加します。

- JM 3.0 V 9.3.0 V 9.3.0 アプリケーション・コードと `com.ibm.mq.jakarta.client.jar` の関係を表示するには、以下のようにします。

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 アプリケーション・コードと `com.ibm.mq.allclient.jar` の関係を表示するには、以下のようにします。

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- JM 3.0 V 9.3.0 V 9.3.0 Jakarta EE リソース・アダプターを使用するためには、以下のようにします。

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 JMS 2.0 Java EE リソース・アダプターを使用する場合:

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Eclipse で JMS プロジェクトを実行するための単純なプロジェクトの例については、 IBM Developer の記事「Developing Java applications for MQ just got 容易に with Maven」を参照してください。

C++ アプリケーションの開発

IBM MQ では、IBM MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。

IBM WebSphere MQ 7.0 における IBM MQ プログラミング・インターフェースの機能拡張は、C++ クラスには適用されません。

IBM MQ C++ は、以下の機能を提供します。

- IBM MQ データ構造体の自動初期設定。
- ジャストインタイムのキュー・マネージャーの接続およびキューのオープン。
- 暗黙のキューの閉止およびキュー・マネージャーの切断。
- 送達不能ヘッダーの伝送と受信。
- IMS ブリッジ・ヘッダーの伝送と受信。
- 参照メッセージ・ヘッダーの伝送と受信。
- トリガー・メッセージの受信。
- CICS bridge・ヘッダーの伝送と受信。
- 作業ヘッダーの伝送と受信。
- クライアント・チャネル定義。

次に示す Booch のクラス・ダイアグラムを見てください。これらの図では、どのクラスも、ハンドルかデータ構造体のどちらかをもつ手続き型 MQI (例えば C の使用) の IBM MQ エンティティーとほぼ対応しています。 すべてのクラスは、ImqError (ImqError C++ クラスを参照) クラスの性質を継承しているため、エラー状態とオブジェクトを個別に関連付けることができます。

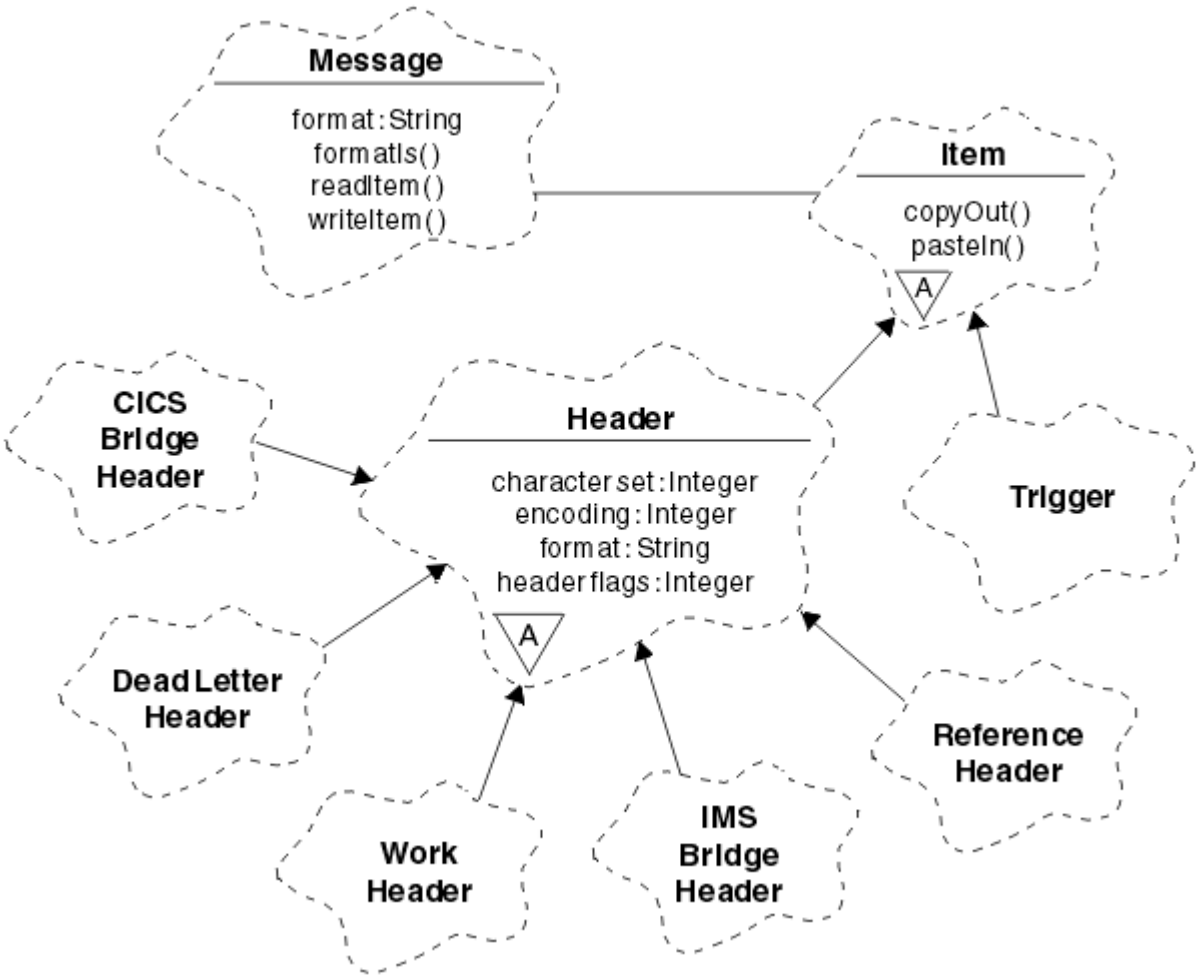


図 *50. IBM MQ C++* クラス (項目ハンドル)

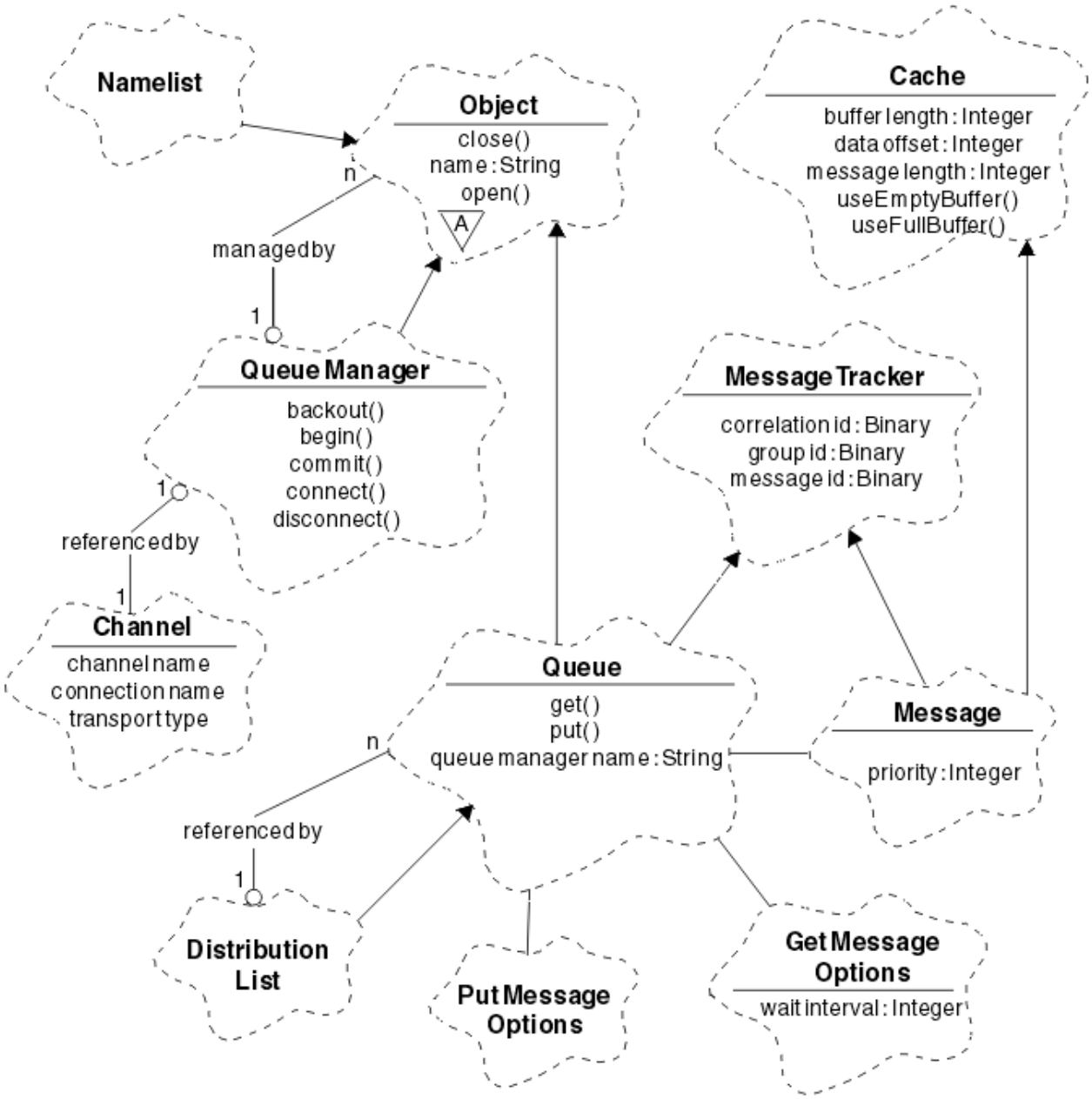


図 *51. IBM MQ C++* クラス *(*キュー管理*)*

Booch のクラス・ダイアグラムを正しく解釈するために、以下の表記規則に留意してください。

- クラス 名の下にはメソッドと重要な属性を示します。
- 抽象クラス は、雲型の囲みの中の小さな三角形で示します。
- 継承関係 は、親クラスの方向を指す矢印で示します。
- 2 つのクラスの間に協調関係 がある場合は、雲型の囲みの間を数字や文字の付いていない線でつないであります。
- 2 つのクラスの間に参照関係 がある場合は、雲型の囲みの間を数字付きの線でつないであります。 この数字は、特定の関係に同時に関与できるオブジェクトの数を表します。

以下のクラスおよびデータ・タイプは、 キュー管理クラス (527 ページの図 51 を参照) および 項目ハンドル・クラス (526 ページの図 50 を参照) の C++ メソッドシグニチャーで使用されます。

- MQBYTE24 などのバイト・アレイをカプセル化する ImqBinary クラス (ImqBinary C++ クラスを参照)。
- **typedef unsigned char ImqBoolean** として定義されている ImqBoolean データ・タイプ。

- MQCHAR64 などの文字アレイをカプセル化する ImqString クラス (ImqString C++ クラスを参照)。

データ構造体を持つエンティティーは、適切なオブジェクト・クラス内に包含されます。 個々のデータ構造体フィールド (C++ と MQI の相互参照を参照) は、メソッドでアクセスされます。

ハンドルを持つエンティティーは、ImqObject クラス階層 (ImqObject C++ クラスを参照) の下にあり、MQI へのカプセル化されたインターフェースを提供します。 これらのクラスのオブジェクトは、手続き型 MQI に関連して必要なメソッド呼び出し回数を減らすことのできる知的機能を持っています。 例えば、必要に応じてキュー・マネージャー接続を確立したり、廃棄したりでき、適切なオプションを使用してキューをオープンしてからクローズすることも可能です。

ImqMessage クラス (ImqMessage C++ クラスを参照) は、MQMD データ構造体をカプセル化します。また、キャッシュ式バッファー機能を提供することによって、ユーザー・データおよび項目 (537 ページの『C++ によるメッセージの読み取り』を参照) 用の保存場所としても利用できます。 ユーザー・データ用に固定長バッファーを提供し、そのバッファーを何回も使用することができます。 バッファー内に存在するデータの量は、使用するたびに変わる可能性があります。 別の方法として、システムが可変長のバッファーを提供して、管理することができます。 この場合には、バッファーのサイズ (メッセージの受信に使用できる量) と実際に使用される量 (伝送のためのバイト数または実際に受信されるバイト数のいずれか) の両方を慎重に考慮する必要があります。

関連概念

技術概要

528 ページの『C++ サンプル・プログラム』
メッセージの取得と書き込みのデモ用に 4 つのサンプル・プログラムが提供されています。

532 ページの『C++ 言語に関する考慮事項』
このトピック・コレクションでは、Message Queue Interface (MQI) を使用するアプリケーション・プログラムを作成する際に考慮しなければならない C++ 言語の使用法および規則について詳述します。

536 ページの『C++ によるメッセージ・データの作成』
メッセージ・データは、システムまたはアプリケーションが提供できるバッファー内に作成されます。 いずれの方法にも利点がいくつかあります。 バッファーの使用例がいくつか提供されています。

5 ページの『IBM MQ 用アプリケーションの開発』
メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

関連資料

543 ページの『IBM MQ C++ プログラムの作成』
サポートされるコンパイラーの URL が、IBM MQ プラットフォーム上で C++ プログラムおよびサンプルをコンパイル、リンク、および実行するために使用するコマンドとともにリストされます。

C++ と MQI の相互参照

IBM MQ C++ クラス

C++ サンプル・プログラム

メッセージの取得と書き込みのデモ用に 4 つのサンプル・プログラムが提供されています。

サンプル・プログラムには次のものがあります。

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

サンプル・プログラムは、529 ページの表 73 に示されているディレクトリーの中にあります。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

表 73. サンプル・プログラムの位置		
Environment	**ソースが入ったディレクトリー**	**ビルドを含むディレクトリー プログラム**
AIX AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
AIX V 9.3.5 AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (注 529 ページの『1』 を参照)
IBM i IBM i	/QIBM/ProdData/mqm/samp/	(注 529 ページの『2』 を参照)
Linux Linux	*MQ_INSTALLATION_PATH*/samp	なし
Windows Windows	*MQ_INSTALLATION_PATH*¥tools& #xa5;cplus¥samples	*MQ_INSTALLATION_PATH*¥tools& #xa5;cplus¥ samples¥bin¥vn (注 529 ページの『3』 を参照)
z/OS z/OS	thlqual.SCSQCPPS	

注 :

1. AIX V 9.3.5 XLC 17 コンパイラーを使用してビルドされたプログラムは「ca」フォルダーにあり、XLC 16 コンパイラーを使用してビルドされたプログラムは「ia」フォルダーにあります。
2. IBM i ILE C++ コンパイラー (IBM i 用) を 使用して作成されたプログラムは、ライブラリー QMQM に入っています。 ソース・ファイルは /QIBM/ProdData/mqm/samp にあります。
3. Windows Microsoft Visual Studio Visual Studio を使用して構築されたプログラムは、*MQ_INSTALLATION_PATH*\tools\cplus\samples\bin\vn にあります。 これらのコンパイラーの詳細については、548 ページの『Windows における C++ プログラムの作成』を参照してください。

サンプル・プログラム HELLO WORLD (imqwrld.cpp)

この C++ サンプル・プログラムは、ImqMessage クラスを使用して通常のデータグラム (C 構造体) を読み書きする方法を示します。

このプログラムは、ImqMessage クラスを使用して通常のデータグラム (C 構造体) を読み書きする方法を指示します。 このサンプルでは、メソッド呼び出しをほとんど採用していませんが、**open**、**close**、**disconnect** といった暗黙のメソッド呼び出しを利用します。

z/OS 以外のすべてのプラットフォームの場合

IBM MQ へのサーバー接続を使用している場合には、以下のいずれかの手順を実行します。

- 既存のデフォルト・キュー SYSTEM.DEFAULT.LOCAL.QUEUE を使用する場合は、パラメーターを引き渡さずに、プログラム **imqwrlds** を実行します。
- 動的に割り当てられた一時的なキューを使用する場合は、デフォルトのモデル・キューの名前 SYSTEM.DEFAULT.MODEL.QUEUE を引き渡して **imqwrlds** を実行します。

IBM MQ へのクライアント接続を使用している場合には、以下のいずれかの手順を実行します。

- MQSERVER 環境変数をセットアップし (詳しくは、 *MQSERVER* を参照)、 **imqwrldc** を実行します。

- **queue-name**、**queue-manager-name**、および **channel-definition** をパラメーターとして引き渡して、**imqwrldc** を実行します。 ここで、**channel-definition** は通常、SYSTEM.DEF.SVRCONN/TCP/*hostname*(1414) です。

z/OS 上

z/OS

サンプル JCL **imqwrldr** により、バッチ・ジョブを作成して実行します。

詳しくは、 z/OS バッチ、RRS バッチ、および CICS を参照してください。

サンプル・コード

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );
```

```
      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

サンプル・プログラム SPUT (imqsput.cpp) および SGET (imqsget.cpp)

これらの C++ プログラムは、指定されたキューにメッセージを書き込み、指定されたキューからメッセージを取り出します。

これらのサンプルは、以下のクラスの使用を示したものです。

- ImqError (ImqError C++ クラスを参照)
- ImqMessage (ImqMessage C++ クラス) を参照
- ImqObject (ImqObject C++ クラス) を参照

- ImqQueue (ImqQueue C++ クラス) を参照
- ImqQueueManager (ImqQueueManager C++ クラスを参照)

プログラムを実行するには、該当する指示に従います。

z/OS 以外のすべてのプラットフォームの場合

1. **imqsputs** *queue-name* を実行します。
2. コンソールでテキスト行を入力します。 これらの行は、指定されたキューにメッセージとして書き込まれます。
3. 入力データを終了するためにヌル行を 1 行入力します。
4. すべての行を取り出してコンソールで表示するために、**imqsgets** *queue-name* を実行します。

z/OS 詳しくは、551 ページの『z/OS Batch、RRS Batch、および CICS における C++ プログラムの作成』を参照してください。

z/OS 上

z/OS

1. サンプル JCL **imqsputr** により、バッチ・ジョブを作成して実行します。 SYSIN データ・セットからメッセージが読み込まれます。
2. サンプル JCL **imqsgetr** により、バッチ・ジョブを作成して実行します。 キューからメッセージが取り出されて、SYSPRINT データ・セットに送信されます。

サンプル・プログラム DPUT (imqdput.cpp)

この C++ のサンプル・プログラムは、2 つのキューで構成された配布リストにメッセージを書き込みます。

DPUT は、ImqDistributionList クラス (ImqDistributionList C++ クラスを参照) の使用を示しています。 このサンプルは、z/OS ではサポートされていません。

1. 名前を指定された 2 つのキューにメッセージを配置するために、**imqdputs** *queue-name-1 queue-name-2* を実行します。
2. それらのキューからメッセージを取り出すために、**imqsgets** *queue-name-1* および **imqsgets** *queue-name-2* を実行します。

C++ 言語に関する考慮事項

このトピック・コレクションでは、Message Queue Interface (MQI) を使用するアプリケーション・プログラムを作成する際に考慮しなければならない C++ 言語の使用法および規則について詳述します。

C++ ヘッダー・ファイル

C++ 言語での IBM MQ アプリケーション・プログラムの作成を支援するために、MQI の定義の一部としてヘッダー・ファイルが用意されています。

これらのヘッダー・ファイルを、以下の表に要約します。

表 *74. C/C++* ヘッダー・ファイル

ファイル名	内容
IMQI.HPP	C++ MQI クラス (CMQC.H および IMQTYPE.H を含む)
IMQTYPE.H	**ImqBoolean** データ・タイプを定義します
CMQC.H	MQI データ構造体およびマニフェスト定数

アプリケーションの移植性を向上させるために、次のように、ヘッダー・ファイルの名前を **#include** プリプロセッサー指示に小文字でコーディングしてください。

```
#include <imqi.hpp> // C++ classes
```

C++ のメソッドと属性

メソッド名は大/小文字混合です。 パラメーターと戻り値には、さまざまな考慮事項が適用されます。 属性には、必要に応じてゲット・メソッドおよびセット・メソッドを使用してアクセスします。

const となっているメソッドのパラメーターは入力専用です。 シグニチャーにポインター (*) または参照 (&) が含まれているパラメーターは、参照により渡されます。 ポインターや参照を含んでいない戻り値は、値により渡されます。返されたオブジェクトの場合、新規エンティティーであるこれらのオブジェクトは呼び出し側の責任となります。

一部のメソッド・シグニチャーには、指定がない場合にデフォルトをとる項目があります。 そのような項目は、必ずシグニチャーの終わりにあり、等号 (=) で示されています。この等号の後の値は、その項目が省略された場合に適用されるデフォルト値を示します。

これらのクラス内のメソッド名はすべて、小文字で始まり、残りの文字は大文字と小文字が混在しています。 メソッド名の最初のワードを除き、各ワードは、大文字で始まります。 意味が一般的に理解されない限り、省略語は使用されません。 使用される省略語には、*id* (ID の意) および *sync* (「同期」 の意) が含まれます。

オブジェクト属性には、「set」メソッドおよび「get」メソッドを使用してアクセスします。 set メソッドは ワード *set* で始まりますが、get メソッドには接頭部はありません。 属性が読み取り専用 であれば、「set」メソッドはありません。

属性はオブジェクトの作成中に有効な状態に初期設定されるため、オブジェクトの状態は常に一貫しています。

C++ のデータ・タイプ

データ・タイプはすべて、C **typedef** ステートメントによって定義されます。

タイプ **ImqBoolean** は IMQTYPE.H で **unsigned char** として定義されており、値 TRUE および FALSE を取ることができます。 **MQBYTE** 配列の代わりに **ImqBinary** クラス・オブジェクトを 使用したり、**char *** の代わりに **ImqString** クラス・オブジェクトを使用したりできます。 多くのメソッドで、ストレージ管理を容易にするために、**char** ポインターや **MQBYTE** ポインターの代わりにオブジェクトが返されます。 すべての戻り値は呼び出し元の責任となり、戻り値がオブジェクトの場合は、delete を使用してその記憶域を破棄できます。

C++ における 2 進ストリングの操作

2 進データのストリングは、**ImqBinary** クラスのオブジェクトとして宣言されます。 このクラスのオブジェクトは、一般的な C 演算子を使用して、コピー、比較、および設定することができます。 コード例が提供されています。

以下のコード例は、2 進ストリングに対する操作を示しています。

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
...
```

C++ における文字ストリングの操作

文字データは、変換演算子を使用して **char *** にキャストできる **ImqString** クラスのオブジェクトで返されることがよくあります。 ImqString クラスには、文字ストリングの処理を支援するメソッドが含まれています。

MQI C++ メソッドを使用して文字データが受け入れられたり返されたりするとき、その文字データは、必ずヌルで終わるため 任意の長さにできます。 ただし、IBM MQ には特定の制限があるため、情報が切り捨てられることがあります。 ストレージ管理を容易にするために、ほとんどの文字データが **ImqString** クラス・オブジェクトに入れて返されます。 このようなオブジェクトは、所定の変換演算子を使用して **char *** にキャストでき、**char *** を必要とする多くの状況で読み取り専用 で使用できます。

注 : ImqString クラス・オブジェクトからの **char *** 変換結果は、 ヌルになることがあります。

C 関数は **char *** 上で使用できますが、**ImqString** クラスには、より望ましい特殊なメソッドがあります。 **operator length** ( ) は、**strlen** および **storage** ( ) に相当するものです。文字データに割り振られたメモリーを示します。

C++ のオブジェクトの初期状態

すべてのオブジェクトには、それぞれの属性により表された一貫した初期状態があります。 初期値は、クラス記述に定義されます。

C++ からの C の使用

C++ プログラムからの C 関数を使用する場合は、適切なヘッダーをインクルードします。

次の例は、C++ プログラムにインクルードされた `string.h` を示しています。

```
extern "C" {
#include <string.h>
}
```

C++ の表記規則

この例は、メソッドの呼び出し方法とパラメーターの宣言方法を示しています。

このコード・サンプルでは、メソッドとパラメーター **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*) を使用します

パラメーターを以下のように宣言し、使用します。

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

...
}
```

C++ による暗黙の操作

メソッドが正常に実行されるためには前提条件を満たすことが必要ですが、そのための動作が適時に かつ暗黙的に実行されることがあります。 これらの暗黙命令とは、接続、オープン、再オープン、クローズ、および切断です。 接続とオープンの暗黙的な動作は、クラス属性によって制御できます。

接続

ImqQueueManager オブジェクトは、結果的に MQI に対するなんらかの呼び出し (C++ と MQI の相互参照を参照) が行われるメソッドの場合に自動的に接続されます。

オープン

ImqObject オブジェクトは、結果的に MQGET、MQINQ、MQPUT、または MQSET 呼び出しが発生するメソッドの場合に自動的にオープンされます。 **openFor** メソッドは、1 つまたは複数の関連する **open option** 値を指定するために使用します。

再オープン

ImqObject は、結果的に MQGET、MQINQ、MQPUT、または MQSET 呼び出しが発生するメソッドの場合に自動的に再オープンされます。これらの呼び出しでは、オブジェクトは既にオープンされていますが、既存の **open options** には、MQI 呼び出しが正常に行われるほど十分な機能はありません。 オブジェクトは、MQCO_NONE という一時的な **close options** 値を使用して一時的にクローズされます。 **openFor** メソッドを使用して、関連するものを追加**オープン・オプション**。

条件によっては再オープンで問題が発生する場合もあります。

- 一時動的キューは、クローズされたときに破棄され、再オープンすることはできません。
- 排他的入力用としてオープンされた (明示的に、あるいはデフォルトによる) キューは、クローズ時点と再オープン時点でウィンドウから他のプロセスによりアクセスされることがあります。
- ブラウズ・カーソル位置についての情報は、キューがクローズされた時点で失われます。 このような状態になっても、クローズや再オープンができなくなることはありませんが、この後、再度 MQGMO_BROWSE_FIRST が使用されるまでカーソルは使用できなくなります。
- 最後に取り出されたメッセージのコンテキストは、キューはクローズされた時点で失われます。

上記の状態のいずれかが発生したり、予測できる場合は、オブジェクトが (明示的または暗黙的に) オープンされる前に、適切な **open options** を明示的に設定して、再オープンを避けてください。

複雑なキューの取扱状態について明示的に **open options** を設定すると、結果的にパフォーマンスが向上し、再オープンの使用にかかわる問題が回避されます。

クローズ

ImqObject は、オブジェクト状態が実行可能でなくなった時点で (例えば、ImqObject の connection reference が切断された場合や、ImqObject オブジェクトが破棄された場合)、自動的にクローズされます。

切断

ImqQueueManager は、接続が実行可能でなくなった時点で (例えば、ImqObject の connection reference が切断された場合や、ImqQueueManager オブジェクトが破棄された場合)、自動的に切断されます。

C++ における 2 進ストリングと文字ストリング

ImqString クラスは、従来の *char * データ・フォーマットをカプセル化します。 ImqBinary クラスは、2 進のバイト配列をカプセル化します。 文字データを設定するメソッドには、データを切り捨てるものがあります。

文字 (**char ***) データを設定するメソッドでは、必ずデータのコピーが 作成されますが、IBM MQ には 特定の制限があるため、メソッドによってはそのコピーの上限を超えた部分は切り捨てられる場合があります。

ImqString クラス (ImqString C++ クラスを参照) は、従来の **char *** をカプセル化し、次の項目をサポートします。

- 比較
- 連結
- コピー
- 整数とテキスト間の変換

- トークン (ワード) 抽出
- 大文字変換

ImqBinary クラス (ImqBinary C++ クラスを参照) は任意のサイズの 2 進バイト・アレイをカプセル化します。 このクラスは、特に以下の属性を保持するために使用されます。

- **accounting token** (MQBYTE32)
- **connection tag** (MQBYTE128)
- **correlation id** (MQBYTE24)
- **facility token** (MQBYTE8)
- **group id** (MQBYTE24)
- **instance id** (MQBYTE24)
- **message id** (MQBYTE24)
- **message token** (MQBYTE16)
- **transaction instance id** (MQBYTE16)

これらの属性は次のクラスのオブジェクトに属しています。

- ImqCICSBridgeHeader (ImqCICSBridgeHeader C++ クラスを参照)
- ImqGetMessageOptions (ImqGetMessageOptions C++ クラスを参照)
- ImqIMSBridgeHeader (ImqIMSBridgeHeader C++ クラスを参照)
- ImqMessageTracker (ImqMessageTracker C++ クラスを参照)
- ImqQueueManager (ImqQueueManager C++ クラスを参照)
- ImqReferenceHeader (ImqReferenceHeader C++ クラスを参照)
- ImqWorkHeader (ImqWorkHeader C++ クラスを参照)

ImqBinary クラスは、比較とコピーもサポートします。

C++ ではサポートされない機能

IBM MQ C++ のクラスとメソッドは、IBM MQ プラットフォームに依存しないように設計されています。 したがって、備えている機能の一部が特定のプラットフォーム上でサポートされていない場合もあります。

ある機能を、その機能がサポートされていないプラットフォーム上で使用しようとした場合、その機能は IBM MQ によって検出されますが、C++ 言語バインディングには検出されません。 IBM MQ は、他の MQI エラーと同様に、プログラムにエラーを報告します。

C++ でのメッセージング

このトピック集では、C++ でメッセージングを準備して読み書きする方法について説明します。

C++ によるメッセージ・データの作成

メッセージ・データは、システムまたはアプリケーションが提供できるバッファー内に作成されます。 いずれの方法にも利点がいくつかあります。 バッファーの使用例がいくつか提供されています。

メッセージを送信する際に、まず最初に、ImqCache オブジェクト (ImqCache C++ クラスを参照) によって管理されるバッファーでメッセージ・データが作成されます。 バッファーは、(継承により) 各 ImqMessage オブジェクト (ImqMessage C++ クラスを参照) と関連付けられています。そのため、バッファーは (**useEmptyBuffer** または **useFullBuffer** メソッドを使用して)、アプリケーションにより提供されますが、システムにより自動的に提供することもできます。 メッセージ・バッファーを提供するアプリケーションの利点は、そのアプリケーションが作成されたデータ域を直接使用できるため、ほとんどの場合データのコピーが不要であることです。 欠点は、提供されたバッファーが固定長であることです。

バッファーは再利用できます。また、伝送バイト数は、各伝送時に必要に応じて変更できます。この変更には、送信前に **setMessageLength** メソッドを使用します。

システムにより自動的に提供される場合、使用可能なバイト数はシステムによって管理されるため、例えば、ImqCache の **write** メソッドまたは ImqMessage の **writeItem** メソッドを使用してデータをメッセージ・バッファーにコピーすることができます。 メッセージ・バッファーは、必要に応じて大きくなります。バッファーが大きくなる際に、以前に書き込まれたデータが失われることはありません。 大きなメッセージや複数の部分から成るメッセージは分割して、各部分を続けて書き込むことができます。

次の例は、単純化されたメッセージ送信を示しています。

1. ユーザーの提供するバッファーにある、準備済みデータを使用します。

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. ユーザーの提供するバッファーにある、準備済みデータを使用します。バッファー・サイズがデータ・サイズより大きくなっています。

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. ユーザー提供のバッファーにデータをコピーします。

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. システム提供のバッファーにデータをコピーします。

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. オブジェクトを使用して、システム提供のバッファーにデータをコピーします。(オブジェクトは内容だけでなくメッセージ・フォーマットを設定します。)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

C++ によるメッセージの読み取り

バッファーは、アプリケーションまたはシステムが提供できます。 データは、バッファーから直接アクセスするか、順次に読み取ることができます。 各メッセージ・タイプには、それと等価なクラスがあります。サンプル・コードが提供されています。

データを受信する際に、アプリケーションまたはシステムは、適切なメッセージ・バッファーを提供します。 特定の ImqMessage オブジェクトの複数の伝送および複数の受信の両方に、同一のバッファーを使用できます。 メッセージ・バッファーは、自動的に提供された場合、どのような長さのデータでも受信できるように大きくなります。 ただし、アプリケーションによって提供されたメッセージ・バッファーが、受信したデータを保持するのに十分な大きさではない場合があります。 その場合には、メッセージ受信に使用されるオプションに応じて、切り捨てが発生するか、受信が失敗します。

着信データは、メッセージ・バッファーから直接アクセスできますが、その場合、データ長は着信データの合計量を示します。 これとは別に、着信データをメッセージ・バッファーから順番に読み取ることができます。 この場合、データ・ポインターは着信データの次のバイトをアドレッシングし、データ・ポインターおよびデータ長はデータが読み取られるたびに更新されます。

項目 とはメッセージの各部分のことで、すべてがメッセージ・バッファーのユーザー域に入っています。 項目は、順番に別個に処理する必要があります。 通常のユーザー・データと異なり、項目は送達不能ヘッダーまたはトリガー・メッセージであっても構いません。 項目は、必ずメッセージ形式と関連付けられています。ただし、メッセージ形式は必ずしも項目と**関連付けられてはいません**。

各項目ごとに、認識可能な IBM MQ メッセージ形式に対応するオブジェクトのクラスがあります。 各送達不能ヘッダーおよび各トリガー・メッセージごとに 1 つずつあります。 ユーザー・データについてのオブジェクト・クラスはありません。 つまり、認識可能な形式が使い尽くされてしまうと、残りの部分の処理はアプリケーション・プログラムに任されます。 ユーザー・データのクラスは、ImqItem クラスを限定することによって作成できます。

次の例で示すメッセージ受信は、ユーザー・データに先行するいくつかの項目を想定し、そのような項目の処理を考慮しています。 項目に属さないユーザー・データは、項目が特定されたあとに受信されるデータとして定義されます。 任意の大きさのメッセージ・データを格納するには自動バッファー (デフォルト) を使用します。

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
```

```
      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                               */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */

      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

この例の中の FMT_USERCLASS は、UserClass クラスのオブジェクトに対応する形式の名前 (8 文字) を表す定数です。この定数は、アプリケーションによって定義されます。

UserClass は、ImqItem クラス (ImqItem C++ クラスを参照) の派生クラスです。UserClass では ImqItem クラスの仮想メソッド **copyOut** および **pasteIn** を使用します。

次に ImqDeadLetterHeader クラス (ImqDeadLetterHeader C++ クラスを参照) のコード例を 2 つ示します。 1 つ目の例は、カプセル化されたメッセージ書き込み用 カスタム・コードを示します。

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

2 つ目の例は、カプセル化されたメッセージ読み取り用 カスタム・コードを示します。

```
// Read a dead-letter header.
// Return TRUE if successful.
```

```
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

自動バッファーでは、バッファー記憶が揮発性 です。 つまり、バッファー・データは、**get** メソッド呼び出しのたびに、物理位置が変わる可能性があります。 したがって、バッファー・データが参照されるたびに、**bufferPointer** メソッドまたは **dataPointer** メソッドを使用してメッセージ・データにアクセスしてください。

メッセージ・データの受信用に固定記憶域を確保しておくプログラムも書くことができます。 この場合、**get** メソッドを使用する前に **useEmptyBuffer** メソッドを呼び出すことができます。

固定の非自動領域を使用すると、メッセージは最大限のサイズに制限されるため、ImqGetMessageOptions オブジェクトの MQGMO_ACCEPT_TRUNCATED_MSG オプションを考慮に入れることが重要です。 このオプションを指定しない場合 (デフォルトです) は、MQRC_TRUNCATED_MSG_FAILED 理由コードが戻ると予想できます。 このオプションを指定した場合は、アプリケーションの設計により、MQRC_TRUNCATED_MSG_ACCEPTED という理由コードが戻る場合もあります。

次のコード例は、固定記憶域を使用してメッセージをどのように受信することができるかを示しています。

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

このコード例では、バッファーは、**bufferPointer** メソッドを使用した場合とは逆に、 常に *pszBuffer* メソッドで直接アドレッシングすることができます。 ただし、汎用アクセスの場合は、**dataPointer** メソッドを使用したほうがよいでしょう。 アプリケーション (ImqCache クラス・オブジェクトではない) では、ユーザー定義の (非自動) バッファーを廃棄する必要があります。

注意: useEmptyBuffer を使用してヌル・ポインターと、ゼロの長さを指定しても、 当然のことながら、ゼロの長さの固定長バッファーが指定されることはありません。 この組み合わせは、あらゆる直前のユーザ

ー定義バッファーを無視し、代わりに元どおりに自動バッファーを使用するという要求として解釈されます。

C++ による送達不能キューへのメッセージの書き込み

送達不能キューにメッセージを書き込むためのプログラム・コードの例。

複数の部分から成るメッセージの代表的な例として、送達不能ヘッダーを持つメッセージがあります。 処理できないメッセージからのデータは、送達不能ヘッダーの最後に追加されます。

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

C++ による IMS ブリッジへのメッセージの書き込み

IMS ブリッジにメッセージを書き込むためのプログラム・コードの例。

IBM MQ - IMS ブリッジに送信されるメッセージでは、特別なヘッダーが使用されることがあります。 IMS ブリッジ・ヘッダーが、通常のメッセージ・データの前に付けられます。

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
```

```
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++ による CICS bridge へのメッセージの書き込み

CICS bridge にメッセージを書き込むためのプログラム・コードの例。

CICS bridge を使用して IBM MQ for z/OS に送信されるメッセージには、特別なヘッダーが必要です。 CICS bridge・ヘッダーが、通常のメッセージ・データの前に付けられます。

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++ による作業ヘッダーでのメッセージの書き込み

z/OS ワークロード・マネージャーによって管理されるキュー宛のメッセージを書き込むためのプログラム・コードの例。

IBM MQ for z/OS に送信されるメッセージの場合、z/OS Workload Manager によって管理されているキューが宛先となります。このようなメッセージには、特別なヘッダーが必要です。 作業ヘッダーは、通常のメッセージ・データの前に付けられます。

```
ImqQueueManager mgr ;             // The queue manager.
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );
```

```
// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

IBM MQ C++ プログラムの作成

サポートされるコンパイラーの URL が、IBM MQ プラットフォーム上で C++ プログラムおよびサンプルをコンパイル、リンク、および実行するために使用するコマンドとともにリストされます。

サポート対象プラットフォーム別および IBM MQ バージョン別のコンパイラーのリストについては、IBM MQ のシステム要件を参照してください。

IBM MQ C++ プログラムをコンパイルおよびリンクするために必要なコマンドは、ご使用のインストール済み環境および要件により異なります。 以下の例では、複数のプラットフォーム上における、IBM MQ のデフォルトのインストール済み環境を使用するコンパイラー用の標準的なコンパイル・コマンドおよびリンク・コマンドが示されています。

AIX AIX における C++ プログラムの作成

XL C Enterprise Edition コンパイラーを使用して、 AIX 上で IBM MQ C++ プログラムをビルドします。

V 9.3.5 XLC 16 コンパイラーと XLC 17 コンパイラーの間のコンパイラー・オプションの異なるマッピングについて詳しくは、 オプションのマッピングを参照してください。

Deprecated V 9.3.5 AIX での XL C/C++ for AIX 16 コンパイラーのサポートは、 IBM MQ 9.3.5 から非推奨になりました。

クライアント

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

32 ビット非スレッド・アプリケーション

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32 ビット・スレッド・アプリケーション

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64 ビット非スレッド・アプリケーション

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64 ビット・スレッド・アプリケーション

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.3.5 **32 ビットのスレッド化されていないアプリケーション (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **32 ビット・スレッド・アプリケーション (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.3.5 **64 ビットのスレッド化されていないアプリケーション (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.3.5 **64 ビット・スレッド・アプリケーション (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

サーバー

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

32 ビット非スレッド・アプリケーション

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32 ビット・スレッド・アプリケーション

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64 ビット非スレッド・アプリケーション

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64 ビット・スレッド・アプリケーション

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.3.5 **32 ビットのスレッド化されていないアプリケーション (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **32 ビット・スレッド・アプリケーション (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.3.5 **64 ビットのスレッド化されていないアプリケーション (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.3.5 **64 ビット・スレッド・アプリケーション (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i IBM i における C++ プログラムの作成

ILE C++ コンパイラーを使用して、 IBM i で IBM MQ C++ プログラムを作成します。

IBM ILE C++ for IBM i は C++ プログラム用のネイティブ・コンパイラーです。 以下の手順では、このコンパイラーを使用して、 *Hello World!* を使用する IBM MQ C++ アプリケーションを作成する方法について説明します。 例として IBM MQ サンプル・プログラムがあります。

1. 「*Read Me first!* 」の指示に従って、ILE C++ for IBM i コンパイラーをインストールします。 製品に付属するマニュアルを参照してください。
2. QCXXN ライブラリーがライブラリー・リストの中にあることを確認します。
3. HELLO WORLD サンプル・プログラムを作成します。
   a. モジュールを作成します。

```
CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
TERASPACE(*YES)
```

   C++ サンプル・プログラムのソースは /QIBM/ProdData/mqm/samp にあり、組み込みファイルは /QIBM/ProdData/mqm/inc にあります。

   または、ソースがライブラリー SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD) に入っていることもあります。

   b. これを IBM MQ に付属のサービス・プログラムとバインドし、プログラム・オブジェクトを生成します。

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

   スレッド・アプリケーションを作成するには、次の再入可能サービス・プログラムを使用します。

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

   c. SYSTEM.DEFAULT.LOCAL.QUEUE を使用して HELLO WORLD サンプル・プログラムを実行します。

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux Linux における C++ プログラムの作成

GNU g + + コンパイラーを使用して、 Linux 上で IBM MQ C++ プログラムをビルドします。

System p

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

クライアント: System p

32 ビット非スレッド・アプリケーション

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

サーバー: System p

32 ビット非スレッド・アプリケーション

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

クライアント: IBM Z

32 ビット非スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

サーバー: IBM Z

32 ビット非スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 ビット)

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

クライアント: x86-64 (32 ビット)

32 ビット非スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

サーバー: x86-64 (32 ビット)

32 ビット非スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows における C++ プログラムの作成

Microsoft Visual Studio C++ コンパイラーを使用して、Windows 上で IBM MQ C++ プログラムを作成します。

重要 : IBM MQ に付属するライブラリーは動的ライブラリーであり、静的ライブラリーではありません。 IBM MQ には、コンパイル時にのみ使用できる「`import libraries`」というものが用意されています。 実行時には、動的ライブラリーを使用する必要があります。

IBM MQ 8.0.0 Fix Pack 4 以降では、IBM MQ には、IBM MQ アプリケーションの実行に必要なライブラリーを含む再配布可能なクライアントが付属しています。 これらのライブラリーは、クライアント・アプリケーションと共にパッケージ化して再配布することができます。 詳しくは、再配布可能クライアント (Windows) を参照してください。

32 ビット・アプリケーションで使用するライブラリー (.lib) ファイルおよび dll ファイルは、*MQ_INSTALLATION_PATH*/Tools/Lib にインストールされます。 64 ビット・アプリケーションで使用されるファイルは *MQ_INSTALLATION_PATH*/Tools/Lib64 にインストールされます。
MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

クライアント

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

サーバー

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

ユニバーサル C ランタイムのインストール

Windows 8.1 または Windows Server 2012 R2 を使用している場合は、Microsoft から汎用 C ランタイム更新 (ユニバーサル CRT) をインストールする必要があります。 このランタイムは、Windows 10、および Windows Server 2016 の一部として組み込まれています。

Universal CRT 更新プログラムは、Microsoft 更新プログラム KB3118401 です。 C:\Windows\System32 ディレクトリー内で ucrtbase.dll という名前のファイルを検索することで、この更新があるのか確認できます。 存在しない場合は、Microsoft ページの https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401 から更新プログラムをダウンロードできます。

このランタイムがインストールされていない状態で、IBM MQ プログラムを実行しようとしたり、Microsoft Visual Studio 2017 を使用してユーザー自身がコンパイルしたプログラムを実行しようとしたりすると、次のようなエラーになります。

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Microsoft Visual Studio 2012 プログラム用ランタイムの準備

Microsoft Visual Studio 2012 を使用して IBM MQ プログラムをコンパイルした場合は、IBM MQ インストーラーが Microsoft Visual Studio 2012 C/C++ ランタイムをインストールしないことに注意してください。 以前のバージョンの IBM MQ が同じコンピューターにインストールされていた場合は、そのインストール済み環境から Microsoft Visual Studio 2012 ランタイムが使用可能になります。

一方、Microsoft Visual Studio 2012 を使用してビルドされたプログラムを使用していて、前のバージョンの IBM MQ がインストールされていなかった場合は、以下のいずれかを実行する必要があります。

- Microsoft から **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** をダウンロードしてインストールします。
- Microsoft Visual Studio 2017、またはランタイムがインストールされる別のレベルの Microsoft Visual Studio でプログラムを再コンパイルする。

Microsoft Visual Studio 2015 コンパイラーを使用して作成された C++ クライアント・ライブラリー

IBM MQ は、Microsoft Visual Studio 2015 C++ コンパイラーを使用して作成された C++ クライアント・ライブラリー、および Microsoft Visual Studio 2017 C++ コンパイラーを提供します。

IBM MQ C++ ライブラリーの 32 ビット・バージョンと 64 ビット・バージョンの両方が用意されています。 32 ビット・ライブラリーは bin\vs2015 フォルダーの下にインストールされ、64 ビット・ライブラリーは bin64\vs2015 フォルダーの下にインストールされます。

デフォルトでは、IBM MQ は Microsoft Visual Studio 2017 ライブラリーを使用するように構成されています。 Microsoft Visual Studio 2015 ライブラリーを使用するには、 IBM MQ をインストールする前、または **setmqenv** または **setmqinst** コマンドを使用する前に、 MQ_PREFIX_VS_LIBRARIES 環境変数を MQ_PREFIX_VS_LIBRARIES=vs2015 に設定する必要があります。

異なる名前の IBM MQ C++ ライブラリーの使用

IBM MQ は異なる名前の付けられた追加の C++ クライアント・ライブラリーをいくつか提供します。 これらのライブラリーは、Microsoft Visual Studio 2015 および Microsoft Visual Studio 2017 C++ コンパイラーでビルドされています。 これらのライブラリーは、同じく Microsoft Visual Studio 2017 C++ コンパイラーでビルドされた既存の C++ ライブラリーに加えて提供されています。 これらの追加の IBM MQ C++ ライブラリーの名前は異なるため、 IBM MQ C++ を使用してビルドされ、 Microsoft Visual Studio 2017 およびそれ以前のバージョンの製品でコンパイルされた IBM MQ C++ アプリケーションを、同じコンピューター上で実行することができます。

追加の Microsoft Visual Studio 2017 ライブラリーの名前は、以下のとおりです。

- imqb23vnvs2017.dll
- imqc23vnvs2017.dll
- imqs23vnvs2017.dll
- imqx23vnvs2017.dll

追加の Microsoft Visual Studio 2015 ライブラリーの名前は、以下のとおりです。

- imqb23vnvs2015.dll
- imqc23vnvs2015.dll
- imqs23vnvs2015.dll
- imqx23vnvs2015.dll

これらのライブラリーは、32 ビット・バージョンと 64 ビット・バージョンの両方が提供されます。 32 ビット・ライブラリーは bin フォルダーの下にインストールされ、64 ビット・ライブラリーは bin64 フォルダーの下にインストールされます。 対応するインポート・ライブラリーは Tools\lib ディレクトリーおよび Tools\lib64 ディレクトリーの下にインストールされます。

アプリケーションが imq*vs2015.lib ファイルを使用する場合は、Microsoft Visual Studio 2015 コンパイラーを使用してコンパイルする必要があります。 Microsoft Visual Studio 2015 でコンパイルされた IBM MQ C++ アプリケーション、または旧バージョンの製品でコンパイルされたアプリケーションを同じコンピューター上で実行するには、以下の例に示すように PATH 環境変数に接頭部を付ける必要があります。

- 32 ビット・アプリケーションの場合:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- 64 ビット・アプリケーションの場合:

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

関連概念
Windows: IBM MQ 8.0 からの変更点

z/OS z/OS Batch、RRS Batch、および CICS における C++ プログラムの作成

z/OS 上でバッチ環境、RRS バッチ環境、または CICS 環境用の IBM MQ C++ プログラムを作成し、サンプル・プログラムを実行します。

IBM MQ for z/OS がサポートする以下の 3 つの環境で実行できる C++ プログラムを作成できます。

- バッチ
- RRS バッチ
- CICS

コンパイル、プリリンク、およびリンク

C++ ソース・コードのコンパイル、プリリンク、およびリンク・エディットにより、z/OS アプリケーションを作成します。

IBM MQ C++ for z/OS は、 IBM C++ for z/OS 言語用の z/OS DLL として実装されています。 DLL を使用して、プリリンク時に、付属の定義 SIDEDECK とコンパイラーの出力を連結します。 これにより、リンカーは IBM MQ C++ のメンバー関数の呼び出しを検査できます。

注 : 定義体は、3 つの環境のそれぞれについて 3 セットずつあります。

IBM MQ for z/OS C++ アプリケーションを作成するには、JCL を作成して実行します。 次の手順を実行してください。

1. CICS 環境下でアプリケーションを実行する場合は、CICS 提供のプロシージャーを使用してプログラム内の CICS コマンドを変換してください。

   さらに、CICS アプリケーションには、以下を行う必要があります。

   a. SCSQLOAD ライブラリーを DFHRPL 連結に追加する。

   b. SCSQPROC ライブラリー内のメンバー IMQ4B100 を使用して CSQCAT1 CEDA グループを定義する。

   c. CSQCAT1 をインストールする。
2. オブジェクト・コードを生成するプログラムをコンパイルする。 コンパイルを実行する JCL に、コンパイラーが製品データ定義ファイルを使用できるようにするステートメントを必ず組み込んでください。 データ定義は、以下の IBM MQ for z/OS ライブラリーで提供されます。

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   また、コンパイラー・オプション /cxx を必ず指定してください。

   注 : thlqual という名前は、 z/OS 上の IBM MQ インストール・ライブラリーの高位修飾子です。
3. 手順 551 ページの『2』 で作成したオブジェクト・コードを、以下に示す定義体も含めてプリリンクする。これらの定義体は **thlqual**.SCSQDEFS の中にあります。

   a. バッチの場合は imqs23dm および imqb23dm

   b. RRS バッチの場合は imqs23dr および imqb23dr

   c. CICS の場合は imqs23dc および imqb23dc

   これらは対応 DLL です。

   a. バッチの場合は imqs23im および imqb23im

   b. RRS バッチの場合は imqs23ir および imqb23ir

   c. CICS の場合は imqs23ic および imqb23ic
4. 手順 551 ページの『3』 で作成したオブジェクト・コードをリンク・エディットして、ロード・モジュールを生成し、アプリケーション・ロード・ライブラリーに保存する。

バッチ・プログラムまたは RRS バッチ・プログラムを実行する場合には、ライブラリー **thlqual**.SCSQAUTH および **thlqual**.SCSQLOAD を STEPLIB データ・セット連結または JOBLIB データ・セット連結に組み込んでください。

CICS プログラムを実行するには、まずシステム管理者に依頼して、そのプログラムを IBM MQ プログラムおよびトランザクションとして CICS に定義します。 プログラムを CICS に定義した後は、通常の方法で実行することができます。

サンプル・プログラムの実行

各プログラムについては、528 ページの『C++ サンプル・プログラム』を参照してください。

付属のサンプル・アプリケーションはソース形式のみです。 以下に、各プログラム・ファイルを示します。

表 *75. z/OS* サンプル・プログラム・ファイル

サンプル	ソース・プログラム (格納先ライブラリー: thlqual.SCSQCPPS)	JCL (格納先ライブラリー: thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

サンプル・プログラムを実行するには、C++ プログラムの場合と同じ方法でサンプル・プログラムをコンパイルして連係編集します (551 ページの『z/OS Batch、RRS Batch、および CICS における C++ プログラムの作成』を参照)。 付属の JCL により、バッチ・ジョブを作成して実行してください。 それにはまず、JCL の中のコメント部分の指示に従って、JCL をカスタマイズする必要があります。

z/OS z/OS UNIX System Services における C++ プログラムの作成

z/OS UNIX System Services (z/OS UNIX) で IBM MQ C++ プログラムをビルドします。

z/OS UNIX シェルのもとでアプリケーションを構築するには、IBM MQ インクルード・ファイル (`thlqual.SCSQC370` および `hlqual.SCSQHPPS` にあるもの) にコンパイラー・アクセス権を与え、DLL サイド・デックの 2 つ (`thlqual.SCSQDEFS` にあるもの) にリンクする必要があります。 実行時には、アプリケーションは、`thlqual.SCSQANLE` のような IBM MQ データ・セット `thlqual.SCSQLOAD`、`thlqual.SCSQAUTH`、および言語固有のデータ・セットの 1 つにアクセスする必要があります。[6].

コンパイル

1. TSO **oput** コマンドまたはファイル・システムを使用して、サンプルを FTP にコピーします。 この例の残りの部分では、サンプルを `/u/fred/sample` というディレクトリーにコピーし、それを `imqwrld.cpp` という名前にしたものと想定しています。
2. z/OS UNIX シェルにログインし、サンプルを入れたディレクトリーに移動します。
3. DLL 定義体と .cpp ファイルを入力として受け入れることができるよう、C++ コンパイラーをセットアップします。

   ```
   /u/fred/sample:> export _CXX_EXTRA_ARGS=1
   /u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
   ```

4. サンプル・プログラムをコンパイルおよびリンクします。 次のコマンドは、プログラムをバッチ定義体にリンクします。代わりに RRS バッチ定義体を使用することもできます。 \ 文字は、コマンドを複数行

[6] 3 つの環境のいずれかで z/OS UNIX を実行するために、「オブジェクトコードのプリリンク」にリストされているいずれかのサイド・デックとリンクすることができます。551 ページの『z/OS Batch、RRS Batch、および CICS における C++ プログラムの作成』

に分割するために使用されます。 この文字を入力しないで、コマンドを 1 つの行として入力してください。

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

TSO **oput** コマンドについて詳しくは、 「z/OS UNIX コマンド解説書」を参照してください。

MAKE ユーティリティーを使用して、C++ プログラムの構築を単純化することも可能です。 以下に、HELLO WORLD C++ サンプル・プログラムを構築するための MAKE ファイルのサンプルを示します。 これは、コンパイル・ステージとリンク・ステージを分離するものです。 MAKE を実行する前に、ステップ 552 ページの『3』 にあるように環境をセットアップしてください。

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

make の使用法について詳しくは、「z/OS UNIX System Services プログラミング・ツール」を参照してください。

実行中

1. z/OS UNIX シェルにログインし、サンプルを構築したディレクトリーに移動します。
2. STEPLIB 環境変数をセットアップして、IBM MQ データ・セットを組み込みます。

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. サンプルを実行します。

```
/u/fred/sample:> ./imqwrld
```

.NET アプリケーションの開発

IBM MQ classes for .NET を使用すると、.NET アプリケーションは、IBM MQ MQI client として IBM MQ に接続することも、 IBM MQ サーバーに直接接続することもできます。

Microsoft .NET Framework を使用し、IBM MQ の機能を活用するアプリケーションがある場合は、IBM MQ classes for .NET を使用する必要があります。 詳しくは、561 ページの『IBM MQ classes for .NET Framework のインストール』を参照してください。

IBM MQ 9.1.1 以降、IBM MQ は Windows 環境のアプリケーションに対して .NET Core をサポートします。詳しくは、554 ページの『IBM MQ classes for .NET のインストール』を参照してください。

IBM MQ 9.1.2 以降、 IBM MQ は Linux 環境のアプリケーションに対して .NET Core をサポートします。

IBM MQ 9.1.4 以降、IBM MQ .NET 管理アプリケーションは複数のクラスター・キュー・マネージャー間で、自動的に接続のバランスを取るようになりました。 .NET Framework ライブラリーと .NET Standard ライブラリーの両方がサポートされています。 詳しくは、均等クラスターについておよびアプリケーションの自動バランシングを参照してください。

オブジェクト指向型の IBM MQ .NET インターフェースは、MQI 動詞を使用するのではなく、オブジェクトのメソッドを使用するという点で、MQI インターフェースとは異なります。

プロシージャー型 IBM MQ アプリケーション・プログラミング・インターフェースは、以下のリストにあるような動詞の周囲に作成されます。

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

これらの動詞はすべて、操作対象の IBM MQ オブジェクトのハンドルをパラメーターとして取ります。 .NET はオブジェクト指向なので、.NET プログラミング・インターフェースもオブジェクト指向になっています。 ユーザーのプログラムは、一連の IBM MQ オブジェクトで構成されています。これらのオブジェクトは、メソッドを呼び出すことによって操作します。 .NET でサポートされる任意の言語でプログラムを作成できます。

手続き型インターフェースを使用する場合は、呼び出し MQDISC(*Hconn*, CompCode, Reason) を使用して、キュー・マネージャーから切断します。 *Hconn* はキュー・マネージャーのハンドルです。

.NET インターフェースでは、キュー・マネージャーはクラス MQQueueManager のオブジェクトで表されます。 このクラスの Disconnect() メソッドの呼び出しにより、 キュー・マネージャーから切断します。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

IBM MQ classes for .NET は、.NET アプリケーションによる IBM MQ との対話を可能にする一連のクラスです。 アプリケーションで使用する、キュー・マネージャー、キュー、チャネル、メッセージといった IBM MQ の様々なコンポーネントを表します。 これらのクラスの詳細は、IBM MQ .NET のクラスとインターフェースを参照してください。

作成したアプリケーションをコンパイルするためには、.NET Framework がインストールされている必要があります。 IBM MQ classes for .NET および .NET Framework のインストール手順については、561 ページの『IBM MQ classes for .NET Framework のインストール』を参照してください。

関連概念

技術概要

5 ページの『IBM MQ 用アプリケーションの開発』
メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

関連タスク

IBM サポートへのお問い合わせ

IBM MQ .NET の問題のトラブルシューティング

1272 ページの『IBM MQ を使用した Microsoft Windows Communication Foundation アプリケーションの開発』
IBM MQ 用の Microsoft Windows Communication Foundation (WCF) カスタム・チャネルは、WCF クライアントとサービスの間でメッセージを送受信します。

614 ページの『XMS .NET アプリケーションの開発』
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) は、XMS と呼ばれるアプリケーション・プログラミング・インターフェース (API) を提供します。 この API は、Java Message Service (JMS) API と同じ一連のインターフェースを備えています。 IBM MQ Message Service Client (XMS) for .NET には、 XMS の完全に管理された実装が含まれています。これは、どの .NET 準拠言語でも使用できます。

Windows Linux IBM MQ classes for .NET のインストール

IBM MQ classes for .NET(サンプルを含む) は、 IBM MQ とともに Windows および Linux にインストールされます。

前提条件とインストール

IBM MQ 9.2.0 以降、 .NET Standard を使用して作成された IBM MQ .NET クライアント・ライブラリーは、Windows および Linux で使用できます。 IBM MQ classes for .NET Standard を実行するには、Microsoft .NET Core をインストールする必要があります。 Microsoft.NET Core 3.1 は、 IBM MQ classes for .NET Standard を実行するために最低限必要なバージョンです。

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 IBM MQ は IBM MQ classes for .NET Standard を使用する .NET 6 アプリケーションをサポートします。 .NET Core 3.1 アプリケーションを使用している場合は、再コンパイルを必要とせずに、 `csproj` ファイルで小さな編集を行い、 `targetframeworkversion` を `"net6.0"`に設定してこのアプリケーションを実行することができます。

V 9.3.1 IBM MQ 9.3.1 は、 .NET 6 に対してビルドされた IBM MQ .NET クライアント・ライブラリーをターゲット・フレームワークとして提供します。 IBM MQ 9.3.1 以降、 .NET 6 をターゲット・フレームワークとして使用してビルドされた IBM MQ ライブラリーを使用してアプリケーションを実行するために必要な最小バージョンは Microsoft .NET 6.0 です。

V 9.3.1 IBM MQ 9.3.1 以降、 .NET Standard を使用して作成された IBM MQ .NET クライアント・ライブラリーは、新規フォルダー `netstandard2.0` の下で使用できます。また、ターゲット・フレームワークとして .NET 6 を使用して作成された IBM MQ .NET クライアント・ライブラリーは、 Windows 上の *MQ_INSTALLATION_PATH*`/bin` および Linux 上の *MQ_INSTALLATION_PATH*`/lib64` の下で使用できます。

IBM MQ classes for .NET の最新バージョンは、標準の IBM MQ インストールの一部として、デフォルトで *Java* メッセージング、*.NET* メッセージング、および *Web* サービス・フィーチャーにインストールされています。

Windows Windows での前提条件とインストールについて詳しくは、以下を参照してください。

- IBM MQ classes for .NET を実行するための前提ソフトウェアについては、IBM MQ classes for .NET の要件を参照してください。
- インストールの指示については、「Windows での IBM MQ サーバーのインストール」または「Windows システムでの IBM MQ クライアントのインストール」を参照してください。

Linux Linux での前提条件とインストールについて詳しくは、以下を参照してください。

- IBM MQ classes for .NET を実行するための前提ソフトウェアについては、IBM MQ classes for .NET の要件を参照してください。
- rpm のインストール手順については、Linux システムへの IBM MQ クライアントのインストールを参照してください。
- Debian パッケージを使用する Linux Ubuntu の場合は、「Linux システムでの IBM MQ クライアントのインストール」を参照してください。

IBM MQ classes for .NET Standard ライブラリー (`amqmdnetstd.dll`) は、NuGet リポジトリーからダウンロードすることができます。 詳しくは、 560 ページの『NuGet リポジトリーからの IBM MQ classes for .NET のダウンロード』を参照してください。

amqmdnetstd.dll ライブラリー

V 9.3.1 IBM MQ 9.3.1 以降、 `amqmdnetstd.dll` ライブラリーは以下の場所から入手できます。

ターゲット・フレームワークとして .NET Standard 2.0 を使用して作成されたライブラリー

- Windows Windows の場合: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`。

Deprecated これらのライブラリーは非推奨になっており、 IBM は将来のリリースで削除する予定です。

ターゲット・フレームワークとして .NET 6 を使用して作成されたライブラリー

- Windows Windows の場合: *MQ_INSTALLATION_PATH*\bin。 サンプル・アプリケーションは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にインストールされています。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*\lib64。 .NET サンプルは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にあります。

LTS IBM MQ 9.3.0 Long Term Support の場合、 amqmdnetstd.dll ライブラリーは以下の場所で使用可能です。

- Windows Windows の場合: *MQ_INSTALLATION_PATH*\bin。 サンプル・アプリケーションは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にインストールされています。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*/lib64 path。 .NET サンプルは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にあります。

重要 : Deprecated V 9.3.1 IBM MQ 9.3.1 以降、ターゲット・フレームワークとして .NET Standard 2.0 を使用して作成された IBM MQ .NET クライアント・ライブラリーは非推奨になり、これらのライブラリーを参照するアプリケーションはコンパイル時に警告 CS0618 をスローします。

LTS Stabilized .NET Framework の amqmdnet.dll ライブラリーはこれまでと変わらず提供されていますが、このライブラリーは安定化されています。つまり、新しい機能は何も導入されません。 最新の機能を使用する場合は、amqmdnetstd.dll ライブラリーに移行する必要があります。 ただし、 IBM MQ 9.1 以降の Long Term Support または Continuous Delivery リリースでは、引き続き amqmdnet.dll ライブラリーを使用できます。

V 9.3.1 .NET Framework アプリケーションが IBM MQ 9.3.1 より前のバージョンの amqmdnetstd.dll または amqmxmsstd.dll を使用してコンパイルされ、同じアプリケーションが .NET 6 ベースの IBM MQ クライアント・ライブラリーを使用して実行される場合、以下の FileLoadException タイプの例外が .NET によってスローされます。

```
例外をキャッチしました: System.IO.FileLoadException: ファイルまたはアセンブリーをロードできませんでした
'amqmdnetstd, Version =x.x.x.x, Culture=ニュートラル, PublicKeyToken=23d6cb914eeaac0e' または
依存関係の 1 つです。 検出されたアセンブリーのマニフェスト定義が、
アセンブリー参照。 (HRESULT からの例外: 0x80131040)

ファイル名: 'amqmdnetstd, Version =x.x.x.x, Culture=ニュートラル,
PublicKeyToken=23d6cb914eeaac0e'
```

このエラーを解決するには、 *MQ_INSTALLATION_PATH*/bin/netstandard2.0 にあるライブラリーを、 .NET Framework アプリケーションが実行されているディレクトリーにコピーする必要があります。

dspmqver コマンド

dspmqver コマンドを使用して、.NET Core コンポーネントのバージョン情報およびビルド情報を表示できます。

IBM MQ classes for .NET Framework と IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) の機能比較

以下の表に、IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) の機能と比較した IBM MQ classes for .NET Framework の機能をリストします。

表 76. *IBM MQ classes for .NET Framework* と *IBM MQ classes for .NET (.NET Standard* ライブラリーと *.NET 6* ライブラリー*)* の相違点

フィーチャー	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー)
クラス名 (API)	すべてのクラスは、各ネットワークで同じ状態を維持します。	すべてのクラスは、各ネットワークで同じ状態を維持します。
オペレーティング・システム	Windows	Windows Docker 化したコンテナー Linux macOS
`app.config` ファイル (再配布可能なクライアントでトレースを有効にするための構成ファイル)	`app.config` ファイルは、再配布可能パッケージおよびスタンドアロン IBM MQ .NET クライアントのトレースを有効にするために使用されます。 トレースに使用する変数 (**MQTRACEPATH** や **MQTRACELEVEL** など) について詳しくは、 アプリケーション構成ファイルを使用した IBM MQ classes for .NET Framework クライアントのトレース を参照してください。	`app.config` サポートされていません。 環境変数を使用します。

表 76. *IBM MQ classes for .NET Framework* と *IBM MQ classes for .NET (.NET Standard* ライブラリーと *.NET 6* ライブラリー*)* の相違点 (続き)

フィーチャー	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー)
トレース	IBM MQ のフルクライアント・インストールの場合、**strmqtrc** コマンドを使用して、 IBM MQ classes for .NET Framework のトレースを有効にすることができます。 再配布可能クライアントの場合、トレースを有効にするために `app.config` ファイルも使用されます。 詳しくは、IBM MQ .NET アプリケーションのトレースを参照してください。 V 9.3.3 IBM MQ 9.3.3 以降、`mqclient.ini` ファイルを使用し、Trace スタンザの適切なプロパティーを設定することにより、トレースを有効または無効にすることができます。`mqclient.ini` ファイルを使用して、トレースを動的に使用可能または使用不可にすることもできます。 詳しくは、 mqclient.ini を使用した IBM MQ .NET アプリケーションのトレースを参照してください。	再配布可能クライアントのトレースを有効にするには、環境変数 **MQDOTNET_TRACE_ON** を使用します。 0 以下の値は、トレースを使用可能にしません。 値 1 は、デフォルト・レベルのトレースを使用可能にします。 1 より大きい値を指定すると、詳細トレースが使用可能になります。 この環境変数をストリングなどの他の値に設定しても、トレースは有効になりません。 環境変数を使用した IBM MQ .NET アプリケーションのトレースを参照してください。 **MQDOTNET_TRACE_ON** 環境変数は、IBM MQ トレース・ディレクトリーが使用可能かどうかを検査します。 トレース・ディレクトリーが使用可能な場合、トレース・ファイルはトレース・ディレクトリーに生成されます。 ただし、 IBM MQ がインストールされていない場合、トレース・ファイルは現行作業ディレクトリーにコピーされます。 IBM MQ classes for .NET Framework に使用される他の環境変数 (**MQERRORPATH**、 **MQLOGLEVEL**、**MQSERVER** など) も同じように使用でき、機能します。 V 9.3.3 IBM MQ 9.3.3 以降、`mqclient.ini` ファイルを使用し、Trace スタンザの適切なプロパティーを設定することにより、トレースを有効または無効にすることができます。`mqclient.ini` ファイルを使用して、トレースを動的に使用可能または使用不可にすることもできます。 詳しくは、 mqclient.ini を使用した IBM MQ .NET アプリケーションのトレースを参照してください。
トランスポート・モード	管理、非管理、およびバインディング	管理対象

表 76. *IBM MQ classes for .NET Framework* と *IBM MQ classes for .NET (.NET Standard* ライブラリーと *.NET 6* ライブラリー*)* の相違点 *(*続き*)*

フィーチャー	IBM MQ classes for .NET Framework	IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー)
TLS	Windows 鍵ストアを使用して証明書を保管します。	Windows Windows では、証明書の保管に鍵ストアを使用する必要があります。 許可値は *USER と *SYSTEM です。 入力に基づいて、IBM MQ .NET クライアントによって、現在のユーザーまたはシステム全体の Windows 鍵ストアが確認されます。 Linux Linux では、X509Store クラスを使用して証明書をインストールし、.NET Core によって場所 ".dotnet/corefx/cryptography/x509stores" に証明書をインストールすることをお勧めします。
CCDT	サポート対象	サポート対象。CCDT パスの設定は .NET Framework クラスと同じです。
クライアント自動再接続	サポート対象	サポート対象
分散トランザクション	サポート対象	サポート対象外
グローバル・アセンブリー・キャッシュ (GAC) へのダイナミック・リンク・ライブラリー (dll) のインストール	Dll は IBM MQ インストールの一部として GAC にインストールされます。	Dll は IBM MQ インストールの一部として GAC にインストールされません。

注 : Windows Windows セキュリティー識別子 (SID):

ドメイン・レベルの認証は、 IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) ではサポートされません。 認証にはログイン・ユーザーの ID が使用されます。

macOS での IBM MQ .NET Core アプリケーションの開発

macOS

IBM MQ .NET Core アプリケーションは、 macOS で開発できます。

IBM MQ .NET ライブラリーは macOS ツールキットにパッケージ化されていないため、それらを Windows または Linux IBM MQ クライアントから macOS にコピーする必要があります。 その後、これらのライブラリーを使用して、 macOS 上で IBM MQ .NET Core アプリケーションを開発できます。

開発したアプリケーションは、Windows または Linux 環境でサポートされ、実行できます。

関連概念

561 ページの『IBM MQ classes for .NET Framework のインストール』
IBM MQ classes for .NET Framework(サンプルを含む) は、 IBM MQ と共にインストールされます。 Windows 上の Microsoft.NET Framework には前提条件があります。

619 ページの『IBM MQ classes for XMS .NET のインストール』
IBM MQ classes for XMS .NET(サンプルを含む) は、IBM MQ on Windows および Linux と共にインストールされます。

Windows Linux NuGet リポジトリーからの IBM MQ classes for .NET のダウンロード

IBM MQ classes for .NET を NuGet リポジトリーからダウンロードできます。これにより、 .NET 開発者は簡単に利用できます。

このタスクについて

NuGet は、 .NET を含む Microsoft 開発プラットフォーム用のパッケージ・マネージャーです。 NuGet クライアント・ツールは、パッケージを作成して取り込むための機能を提供します。 NuGet パッケージは、コンパイル済みコード (DLL) を含む .nupkg 拡張子、そのコードに関連するその他のファイル、およびパッケージのバージョン番号などの情報を含む記述マニフェストを持つ単一の圧縮ファイルです。

amqmdnetstd.dll ライブラリー含む IBMMQDotnetClient NuGet パッケージを NuGet Gallery (すべてのパッケージ作成者と利用者が使用する中央パッケージ・リポジトリー) からダウンロードできます。

注 : V 9.3.1 IBM MQ 9.3.1 以降、 NuGet パッケージには、ターゲット・フレームワークとして .NET Standard 2.0 および .NET 6 を使用して作成されたライブラリーが含まれています。 .NET Standard 2.0 のライブラリーは netstandard2.0 フォルダーの下にあり、 .NET 6 のライブラリーは net6.0 フォルダーの下にあります。

IBMMQDotnetClient パッケージのダウンロードするには、以下の 3 つの方法があります。

- Microsoft Visual Studio を使用します。 NuGet は Microsoft Visual Studio の拡張として配布されます。 Microsoft Visual Studio 2012 以降では、 NuGet はデフォルトでプリインストールされています。
- NuGet Package Manager または .NET CLI を使用してコマンド・ラインから。
- Web ブラウザーを使用して。

再配布可能パッケージの場合は、環境変数 **MQDOTNET_TRACE_ON** を使用してトレースを有効にします。

手順

- Microsoft Visual Studio 内の Package Manager UI を使用して IBMMQDotnetClient パッケージをダウンロードするには、以下の手順を実行します。

  a) .NET プロジェクトを右クリックしてから、「**Nuget パッケージの管理 (Manage Nuget Packages)**」をクリックします。

  b)「**参照**」 タブをクリックして、 "IBMMQDotnetClient"を検索します。

  c) パッケージを選択して、「**インストール**」をクリックします。

  インストール中に、Package Manager はコンソール文の形式で進行情報を示します。
- コマンド行から IBMMQDotnetClient パッケージをダウンロードするには、以下のいずれかのオプションを選択します。
  - NuGet Package Manager を使用して、以下のコマンドを入力します。

    ```
    Install-Package IBMMQDotnetClient -Version 9.1.4.0
    ```

    インストール中に、Package Manager はコンソール文の形式で進行情報を示します。 出力をログ・ファイルにリダイレクトするができます。
  - .NET CLI を使用して、以下のコマンドを入力します。

    ```
    dotnet add package IBMMQDotnetClient --version 9.1.4
    ```
- Web ブラウザーを使用して、https://www.nuget.org/packages/IBMMQDotnetClient から IBMMQDotnetClient パッケージをダウンロードします。

関連概念

IBM MQ Client for .NET のライセンス情報

関連タスク

622 ページの『NuGet リポジトリーからの IBM MQ classes for XMS .NET のダウンロード』
IBM MQ classes for XMS .NET は、.NET 開発者が容易にコンシュームできるようにするため、NuGet レポジトリーからダウンロードして使用することができます。

Windows IBM MQ classes for .NET Framework のインストール

IBM MQ classes for .NET Framework(サンプルを含む) は、 IBM MQ と共にインストールされます。 Windows 上の Microsoft.NET Framework には前提条件があります。

最新バージョンの IBM MQ classes for .NET Framework は、デフォルトでは、IBM MQ 標準インストールの一部として、*Java and .NET Messaging and Web Services* フィーチャーにインストールされます。 インストール手順については、Windows への IBM MQ サーバーのインストール または Windows システムへの IBM MQ クライアントのインストールを参照してください。

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 IBM MQ classes for .NET Framework を実行するには、Microsoft.NET Framework V4.7.2 以降をインストールする必要があります。 これは、最低限必要なバージョンが V4.6.2 であった IBM MQ 9.2 からの変更です。

V 9.3.0 V 9.3.0 アプリケーションの `app.config` ファイルにある以下のタグを追加することで、再コンパイルすることなく、Microsoft.NET Framework V3.5 でコンパイルされた既存のアプリケーションを実行できます。

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

注 : IBM MQ をインストールする前に Microsoft .NET Framework V4.7.2 以降がインストールされていない場合、 IBM MQ 製品のインストールはエラーなしで続行されますが、 IBM MQ classes for .NET は使用できません。 IBM MQ をインストールした後に .NET Framework がインストールされた場合、 *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` スクリプトを実行することで、IBM MQ.NET アセンブリーを登録する必要があります。ここで *WMQInstallDir* は、IBM MQ がインストールされているディレクトリーです。 このスクリプトにより、必要なアセンブリーがグローバル・アセンブリー・キャッシュ (GAC) にインストールされます。 実行するアクションを記録する一連の `amqi*.log` ファイルは、%TEMP% ディレクトリーで作成されます。 .NET が以前のバージョン (例えば、.NET V3.5) から V4.7.2 以上にアップグレードされている場合は、 `amqiRegisterdotNet.cmd` スクリプトを再実行する必要はありません。

複数インストール環境で、IBM MQ classes for .NET をサポート・パックとして事前にインストールしてある場合は、そのサポート・パックをまずアンインストールしなければ IBM MQ をインストールできません。 IBM MQ classes for .NET と一緒にインストールされる IBM MQ フィーチャーに、サポート・パックと同じ機能が含まれています。

ソース・ファイルを含む、サンプル・アプリケーションも用意されています。562 ページの『.NET 用のサンプル・アプリケーション』を参照してください。

.NET を含む Microsoft WCF のために、IBM MQ カスタム・チャネルを使用することに関する詳細は、1272 ページの『IBM MQ を使用した Microsoft Windows Communication Foundation アプリケーションの開発』を参照してください

関連概念

554 ページの『IBM MQ classes for .NET のインストール』
IBM MQ classes for .NET(サンプルを含む) は、 IBM MQ とともに Windows および Linux にインストールされます。

関連タスク

IBM MQ .NET アプリケーションのトレース

IBM MQ classes for .NET のキュー・マネージャーへの接続のオプション

IBM MQ classes for .NET のキュー・マネージャーへの接続には、3 つのモードが存在します。 どのタイプの接続が最も必要に適しているかを考慮してください。

クライアント・バインディング接続

IBM MQ classes for .NET を IBM MQ MQI client として使用するには、IBM MQ MQI client を使用して、IBM MQ サーバー・マシンまたは別のマシンにインストールすることができます。 クライアント・バインディング接続では、XA または非 XA トランザクションを使用できます。

サーバー・バインディング接続

IBM MQ classes for .NET は、サーバー・バインディング・モードで使用されている場合、ネットワーク経由で通信を行うのではなく、キュー・マネージャー API を使用します。 これにより、ネットワーク接続を使用した場合と比べて IBM MQ アプリケーションの パフォーマンスが向上します。

バインディング接続を使用するには、IBM MQ サーバーに IBM MQ classes for .NET をインストールする必要があります。

管理対象クライアント接続

このモードでの接続では、ローカル・マシンまたはリモート・マシンで実行されている IBM MQ サーバーに IBM MQ クライアントとして接続します。

このモードで接続する IBM MQ classes for .NET は、.NET 管理対象コードに残り、 ネイティブ・サービスに対する呼び出しを行いません。 管理対象コードについて詳しくは、Microsoft の資料を参照してください。

管理対象クライアントの使用には、いくつかの制限があります。 これらについて詳しくは、578 ページの『管理対象クライアント接続』を参照してください。

.NET 用のサンプル・アプリケーション

ユーザー独自の .NET アプリケーションを実行するには、サンプル・アプリケーションの代わりにユーザーのアプリケーションの名前を使用して、検証プログラムの指示に従います。

以下のサンプル・アプリケーションが用意されています。

- メッセージ書き込みアプリケーション
- メッセージ読み取りアプリケーション
- 「Hello World」アプリケーション
- パブリッシュ/サブスクライブ・アプリケーション
- メッセージ・プロパティーを使用するアプリケーション

これらのサンプル・アプリケーションはすべて C# 言語で提供されていますが、一部は C++ および Visual Basic でも提供されています。 .NET でサポートされる任意の言語でアプリケーションを作成できます。

「メッセージ書き込み」プログラム SPUT (nmqsput.cs、mmqsput.cpp、vmqsput.vb)
このプログラムは、メッセージを指定のキューに書き込む方法を示します。 プログラムには 3 つのパラメーターがあります。

- キューの名前 (必須) (例: SYSTEM.DEFAULT.LOCAL.QUEUE)
- キュー・マネージャーの名前 (オプション)
- チャネルの定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。 チャネルが定義されている場合、そのフォーマットは MQSERVER 環境変数と同じです。

「メッセージ読み取り」プログラム SGET (nmqsget.cs、mmqsget.cpp、vmqsget.vb)
このプログラムは、メッセージを指定のキューから読み取る方法を示します。 プログラムには 3 つのパラメーターがあります。

- キューの名前 (必須) (例: SYSTEM.DEFAULT.LOCAL.QUEUE)
- キュー・マネージャーの名前 (オプション)
- チャネルの定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。 チャネルが定義されている場合、そのフォーマットは MQSERVER 環境変数と同じです。

「Hello World」プログラム (nmqwrld.cs、mmqwrld.cpp、vmqwrld.vb)
このプログラムは、メッセージを書き込んだり読み取ったりする方法を示します。 プログラムには 3 つのパラメーターがあります。

- キューの名前 (オプション) (例: SYSTEM.DEFAULT.LOCAL.QUEUE、SYSTEM.DEFAULT.MODEL.QUEUE など)
- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー名が指定されなかった場合、キュー名はデフォルトで SYSTEM.DEFAULT.LOCAL.QUEUE になります。 キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。

「パブリッシュ/サブスクライブ」プログラム (MQPubSubSample.cs)
このプログラムは、IBM MQ のパブリッシュ/サブスクライブを使用する方法を示します。 このプログラムは C# でのみ提供されています。 プログラムには 2 つのパラメーターがあります。

- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション)

「メッセージ・プロパティー」プログラム (MQMessagePropertiesSample.cs)
このプログラムは、メッセージ・プロパティーを使用する方法を示します。 このプログラムは C# でのみ提供されています。 プログラムには 2 つのパラメーターがあります。

- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション)

インストールを検証する場合は、これらのアプリケーションをコンパイルして実行します。

インストール場所

サンプル・アプリケーションは、作成に使用された言語に応じて、以下の場所にインストールされます。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

サンプル・アプリケーションのビルド

サンプル・アプリケーションを作成するために、バッチ・ファイルが言語ごとに用意されています。

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

bldcssamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

bldmcpsamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

これらのアプリケーションを Microsoft Visual Studio 2003/.NET SDKv1.1 でコンパイルする場合は、コンパイル・コマンド

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

を次のコマンドに置き換えてください:

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

bldvbsamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

IBM MQ を Microsoft .NET Core とともに使用するサンプル

IBM MQ 9.2.0 以降、IBM MQ は、Windows 環境で .NET Core for IBM MQ .NET アプリケーションをサポートします。 IBM MQ classes for .NET Standard (サンプルを含む) は、IBM MQ の標準インストールの一環でデフォルトでインストールされます。

IBM MQ .NET のサンプル・アプリケーションは、&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base にインストールされています。 サンプルをコンパイルするために使用できるスクリプトも提供されています。

提供された build.bat ファイルを使用してサンプルを作成することができます。 Windows 上の以下の場所で、各サンプルにつき 1 つの build.bat があります。

- MQ\tools\dotnet\samples\cs\core\base\SimpleGet
- MQ\tools\dotnet\samples\cs\core\base\SimplePut

Linux IBM MQ 9.2.0 以降では、IBM MQ は、Linux 環境のアプリケーションに関しても Core もサポートします。

IBM MQ を Microsoft .NET Core とともに使用する場合の詳細については、554 ページの『IBM MQ classes for .NET のインストール』を参照してください。

キュー・マネージャーが TCP/IP クライアント接続を受け入れるように構成する

クライアントからの着信接続要求を受け入れるようにキュー・マネージャーを構成します。

このタスクについて

このタスクでは、TCP/IP クライアント接続を受け入れるようにキュー・マネージャーを構成するための基本ステップを説明します。 実動システムの場合は、キュー・マネージャーを構成する際にセキュリティー関係も考慮する必要があります。

手順

1. サーバー接続チャネルを定義します。
   a. キュー・マネージャーを始動します。
   b. NET.CHANNEL というサンプル・チャネルを次のように定義します。

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   重要 : このサンプルは、サンドボックス環境専用です。 セキュリティー関係の考慮は含まれていません。 実動システムの場合は、TLS やセキュリティー出口の使用を考慮してください。 詳しくは、IBM MQ の保護 を参照してください。
2. リスナーを開始します。

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

注 : 大括弧は、オプション・パラメーターを表します。 *qmname* は、デフォルトのキュー・マネージャーの場合は不要です。 ポート番号 *portnum* は、デフォルトの (1414) が使用される場合は不要です。

.NET での分散トランザクション

分散トランザクションやグローバル・トランザクションを使用すると、 クライアント・アプリケーションは複数のネットワーク・システムにある複数の異なるデータ・ソースを 1 つのトランザクションに 含めることができます。

分散トランザクションでは、トランザクション・マネージャーが複数のリソース・マネージャーの間で トランザクションの調整と管理を行います。

トランザクションは、単一フェーズ・コミット・プロセスまたは 2 フェーズ・コミット・プロセスにすることができます。 単一フェーズ・コミットは、1 つだけのリソース・マネージャーがトランザクションに参加するプロセスであり、 2 フェーズ・コミット・プロセスでは、複数のリソース・マネージャーがトランザクションに参加します。 2 フェーズ・コミット・プロセスでは、 トランザクション・マネージャーは

準備呼び出しを送信して、 すべてのリソース・マネージャーがコミットする準備ができているかどうかをチェックします。 すべてのリソース・マネージャーから確認応答が受信されたら、 コミット呼び出しが発行されます。 そうでない場合、トランザクション全体のロールバックが実行されます。 詳しくは、「トランザクション管理およびサポート」を参照してください。 リソース・マネージャーは、自身がトランザクションに参加することをトランザクション・マネージャーに知らせる必要があります。 自身の参加をトランザクション・マネージャーに知らせた後、 そのリソース・マネージャーは、そのトランザクションがコミットまたはロールバックされるときに トランザクション・マネージャーからコールバックを受信します。

IBM MQ .NET のクラスは、既に非管理対象モードおよびサーバー・バインディング・モードの接続での分散トランザクションをサポートしています。 これらのモードでは、IBM MQ .NET クラスはすべての呼び出しを、.NET の代わりにトランザクション処理を管理する C 拡張トランザクション・クライアントに委任します。

現在、IBM MQ .NET のクラスは、管理対象モードでの分散トランザクションをサポートしています。 管理対象モードでは、IBM MQ .NET のクラスは System.Transactions 名前空間を使用して分散トランザクションをサポートします。 System.Transactions インフラストラクチャーでは、IBM MQ を含むすべてのリソース・マネージャーで開始されたトランザクションがサポートされ、トランザクション・プログラミングを単純かつ効率的に行えます。 IBM MQ .NET アプリケーションは、.NET の暗黙的トランザクション・プログラミング・モデルまたは明示的トランザクション・プログラミング・モデルを使用して、メッセージの書き込みと読み取りを行うことができます。 暗黙的トランザクションでは、 トランザクションをコミット、ロールバック (明示的トランザクションの場合)、または完了するタイミングを決定するアプリケーション・プログラム によってトランザクション境界が作成されます。 明示的トランザクションでは、トランザクションのコミット、ロールバック、および完了を行うかどうかを明示的に指定する必要があります。

IBM MQ.NET は、Microsoft Distributed Transaction Coordinator (MS DTC) をトランザクション・マネージャーとして使用して、複数のリソース・マネージャーの間でトランザクションの調整と管理を行います。 IBM MQ はリソース・マネージャーとして使用されます。 なお、XA トランザクションでは TLS を使用できません。 CCDT を使用する必要があります。 詳しくは、TLS チャネルを持つ拡張トランザクション・クライアントの使用を参照してください。

IBM MQ.NET は、X/Open 分散トランザクション処理 (DTP) モデルに従います。 X/Open Distributed Transaction Processing モデルは、ベンダー・コンソーシアムの Open Group によって提案された分散トランザクション処理モデルです。 このモデルはトランザクション処理とデータベースの領域を扱うほとんどの商用ベンダーの間で標準となっています。 ほとんどの商用のトランザクション管理製品は、X/DTP モデルをサポートしています。

トランザクションのモード

- 567 ページの『.NET 管理対象モードの分散トランザクション』
- 非管理対象モード用の分散トランザクション

さまざまなシナリオでのトランザクションの調整

- 1 つの接続が複数のトランザクションに参加している場合がありますが、どの時点においてもアクティブなトランザクションは 1 つだけです。
- トランザクション中は、MQQueueManager.Disconnect 呼び出しが優先されます。 この呼び出しの場合、トランザクションはロールバックするよう要求されます。
- トランザクション中は、MQQueue.Close または MQTopic.Close 呼び出しが優先されます。 この呼び出しの場合、トランザクションはロールバックするよう要求されます。
- トランザクション境界は、 トランザクションをコミット、ロールバック (明示的トランザクションの場合)、 または完了する (暗黙的トランザクションの場合) タイミングを決定するアプリケーション・プログラムによって作成されます。
- トランザクション中に、キュー呼び出しまたはトピック呼び出しに対して Put または Get 呼び出しを発行する前に、予期しないエラーのためにクライアント・アプリケーションが中断した場合、このトランザクションはロールバックされ、MQException がスローされます。
- キュー呼び出しまたはトピック呼び出しに対して Put または Get 呼び出し中に MQCC_FAILED 理由コードが返された場合は、理由コードとともに MQException がスローされ、トランザクションはロールバックされます。 準備呼び出しがトランザクション・マネージャーによって既に発行済みの場合、IBM

MQ .NET は、トランザクションを強制的にロールバックすることによって準備要求を返します。 次に、トランザクション・マネージャー DTC が発端になり、現在の周辺トランザクションに関わっているすべてのリソース・マネージャーとの現行作業がロールバックされます。

- 複数のリソース・マネージャーが関係するトランザクション中に、何らかの環境上の理由で Put または Get 呼び出しがいつまでもハングする場合、トランザクション・マネージャーは規定された期間待機します。 その期間が経過した後、すべてのリソース・マネージャーが現在の周辺トランザクションに入っているすべての作業がロールバックされます。 この無期限の待機が準備フェーズで発生する場合、トランザクション・マネージャーはタイムアウトするか、リソースに対して未確定呼び出しを発行します。 この場合、トランザクションはロールバックされます。
- トランザクションを使用しているアプリケーションは、SYNC_POINT の下でメッセージを Put または Get する必要があります。 SYNC_POINT の下にないトランザクション・コンテキストでメッセージの Put 呼び出しまたは Get 呼び出しが発行された場合、その呼び出しは MQRC_UNIT_OF_WORK_NOT_STARTED 理由コードで失敗します。

Microsoft.NET System.Transactions 名前空間を使用した管理対象クライアント・トランザクション・サポートと非管理対象クライアント・トランザクション・サポートの間の動作上の違い

ネストされたトランザクションでは、TransactionScope が別の TransactionScope の中にあります。

- IBM MQ .NET が完全に管理するクライアントは、ネストされた TransactionScope をサポートします。
- IBM MQ .NET が管理しないクライアントは、ネストされた TransactionScope をサポートしません。

System.Transactions からの従属トランザクション

- IBM MQ .NET が完全に管理するクライアントは、System.Transactions が提供する従属トランザクション機能をサポートします。
- IBM MQ .NET が管理しないクライアントは、System.Transactions が提供する従属トランザクション機能をサポートしません。

製品サンプル

製品サンプル `SimpleXAPut`、および `SimpleXAGet` は、`WebSphere MQ\tools\dotnet\samples\cs\base` のもとで使用できます。 これらのサンプルは C# アプリケーションです。これは、SystemTransactions 名前空間を使用する分散トランザクションでの MQPUT および MQGET の使用を実演するものです。 これらのサンプルについて詳しくは、570 ページの『TransactionScope 内での単純なメッセージの書き込みおよび読み取りの作成』を参照してください。

.NET 管理対象モードの分散トランザクション

IBM MQ .NET クラスは、管理対象モードでの分散トランザクションのサポートに System.Transactions 名前空間を使用します。 管理対象モードでは、MS DTC は、トランザクションに参加しているすべてのサーバーにわたる分散トランザクションを調整および管理します。

IBM MQ .NET クラスは、System.Transactions.Transaction クラスに基づく明示的プログラミング・モデルと、トランザクションがインフラストラクチャーによって自動的に管理される、System.Transactions.TransactionScope クラスを使用する暗黙的プログラミング・モデルを提供します。

暗黙的トランザクション

次のコードの一部で、IBM MQ .NET アプリケーションが .NET の暗黙的トランザクション・プログラミングを使用してメッセージを書き込む方法を説明します。

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

暗黙的トランザクションのコード・フローの説明

コードは、*TransactionScope* を作成し、メッセージを有効範囲内に書き込みます。 その後、*Complete* を呼び出して、トランザクション・コーディネーターにトランザクションの完了を通知します。 トランザクション・コーディネーターは、*prepare* および *commit* を発行して、トランザクションを完了します。 発行が検出されると、*rollback* が呼び出されます。

明示的トランザクション

次のコードで、IBM MQ .NET アプリケーションが .NET の明示的トランザクション・プログラミング・モデルを使用してメッセージを書き込む方法を説明します。

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

明示的トランザクションのコード・フローの説明

コードの一部では *CommitableTransaction* クラスを使用してトランザクションを作成します。 これは、その有効範囲内にメッセージを書き込んでから、*commit* を明示的に呼び出してトランザクションを完了します。 問題が発生した場合には、*rollback* が呼び出されます。

.NET 非管理モードの分散トランザクション

IBM MQ.NET クラスは、暗黙的または明示的なトランザクション・プログラミング・モデルを使用して、拡張トランザクション・クライアントおよび COM+/MTS をトランザクション・コーディネーターとして使用する非管理対象接続 (クライアント) をサポートします。 非管理対象モードでは、IBM MQ .NET クラスは、そのすべての呼び出しを、.NET に代わってトランザクション処理を管理する C 拡張トランザクション・クライアントに委任します。

トランザクション処理は外部のトランザクション・マネージャーによって制御され、そのトランザクション・マネージャーの API の制御下でグローバルな作業単位が調整されます。 MQBEGIN、MQCMIT、および MQBACK の各 verb は使用できません。IBM MQ .NET クラスは、非管理対象トランスポート・モード (C クライアント) を介してこのサポートを公開します。 XA 準拠トランザクション・マネージャーの構成を参照してください。

MTS は、CICS、Tuxedo、およびその他のプラットフォームで使用可能な機能と同じ機能を Windows NT で提供するために、トランザクション処理 (TP) システムとして進化しました。 MTS がインストールされているとき、Microsoft Distributed Transaction Coordinator (MSDTC) を呼び出した Windows NT へ別のサービスが追加されます。MSDTC は、別のデータ・ストアまたはリソースをスパンするトランザクションを調整します。 これが機能するには、各データ・ストアに専有のリソース・マネージャーが実装されている必要があります。

IBM MQ は、DTC XA 呼び出しを IBM MQ(X/Open) 呼び出しにマップするために管理するインターフェース (プロプラエタリー・リソース・マネージャー・インターフェース) を実装することにより、 MSDTC と互換性を持ちます。 IBM MQ は、リソース・マネージャーの役割を果たします。

COM+ などのコンポーネントが IBM MQ へのアクセスを要求すると、COM は通常、適切な MTS コンテキスト・オブジェクトでトランザクションが必要かどうかを確認します。 トランザクションが必要な場合、COM は DTC に通知し、この操作に不可欠な IBM MQ トランザクションを自動的に開始します。 次に COM は、MQMTS ソフトウェアを使用してデータを操作し、必要に応じてメッセージの書き込みおよび読み取りを行います。 COM から取得されたオブジェクト・インスタンスは、データに対するすべてのアクションが完了

した後で、SetComplete メソッドまたは SetAbort メソッドを呼び出します。 アプリケーションが SetComplete を呼び出すと、アプリケーションがトランザクションを完了したことが DTC に通知され、DTC は 2 フェーズ・コミット・プロセスに進むことができます。 次に DTC が MQMTS を呼び出し、今度は MQMTS が IBM MQ を呼び出して、トランザクションをコミットまたはロールバックします。

非管理対象クライアントを使用した IBM MQ .NET アプリケーションの作成

COM+ のコンテキスト内で実行するには、.NET クラスが System.EnterpriseServices.ServicedComponent を継承していなければなりません。 サービスを受けるコンポーネントを使用するアセンブリーを作成するための規則と推奨事項は、以下のとおりです。

注 : 以下のステップは、System.EnterpriseServices モードを使用する場合だけ該当します。

- COM+ で開始されるクラスとメソッドは、どちらも public でなければなりません (internal クラスでも、protected メソッドでも、static メソッドでもありません)。
- クラスとメソッドの属性: TransactionOption 属性は、クラスのトランザクション・レベル、すなわちトランザクションが使用不可か、サポートされているか、必須かどうかを指示します。 ExecuteUOW() メソッドの AutoComplete 属性は、COM+ に対して、スローされる未処理例外がない場合にトランザクションをコミットするよう指示します。
- アセンブリーの厳密な命名: アセンブリーには厳密な名前を付け、グローバル・アセンブリー・キャッシュ (GAC) に登録する必要があります。 アセンブリーは COM+ に明示的に登録されるか、GAC に登録された後に遅延登録によって登録されます。
- COM+ へのアセンブリーの登録: COM クライアントに公開されるアセンブリーを準備します。 次に、アセンブリー登録ツール regasm.exe を使用して、タイプ・ライブラリーを作成します。

```
regasm UnmanagedToManagedXa.dll
```

- アセンブリーを GAC `gacutil /i UnmanagedToManagedXa.dll` に登録します。
- .NET サービス・インストーラー・ツール regsvcs.exe を使用して、アセンブリーを COM+ に登録します。regasm.exe によって作成されたタイプ・ライブラリーを、次のようにして確認します。

```
Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
```

- アセンブリーが GAC にデプロイされ、遅延登録により後で COM+ に登録されます。 .NET Framework は、このコードが初めて実行された後に登録を行います。

以下のセクションでは、System.EnterpriseServices モデルと、COM+ での System.Transactions を使用したコード・フローの例について説明します。

System.EnterpriseServices モデルを使用したコード・フローの例

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
```

```
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
                                     MQC.MQOO_OUTPUT +
                                     MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

COM+ との対話のための System.Transactions を使用したコード・フローの例

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                 opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                          MQC.MQOO_INPUT_SHARED +
                                          MQC.MQOO_OUTPUT +
                                          MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

TransactionScope 内での単純なメッセージの書き込みおよび読み取りの作成

製品のサンプル C# アプリケーションを IBM MQ 内で使用できます。 これらの単純なアプリケーションは、TransactionScope 内でのメッセージの書き込みおよび読み取りを実演するものです。 このタスクの終わりには、メッセージをキューまたはトピックに書き込んだり、キューまたはトピックから読み取ったりできるようになります。

始める前に

MSDTC サービスが実行されていて XA トランザクションに対して有効になっている必要があります。

このタスクについて

この例は、単純なアプリケーション SimpleXAPut と SimpleXAGet です。 プログラム SimpleXAPut および SimpleXAGet は、IBM MQ 内で使用できる C# アプリケーションです。 SimpleXAPut は、SystemTransactions 名前空間を使用する分散トランザクションでの MQPUT の使用を実演するものです。 SimpleXAGet は、SystemTransactions 名前空間を使用する分散トランザクションでの MQGET の使用を実演するものです。

SimpleXAPut は MQ\tools\dotnet\samples\cs\base にあります。

手順

アプリケーションは、tools\dotnet\samples\cs\base\bin のコマンド行パラメーターを使用して実行できます。

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

パラメーターは、以下のとおりです。

-destinationURI

これはキューまたはトピックになります。 キューの場合は queue://queueName のように指定し、トピックの場合は topic://topicName のように指定します。

-host

これは localhost などのホスト名か IP アドレスになります。

-port

キュー・マネージャーが実行されているポートです。

-channel

使用する接続チャネルです。 デフォルトは SYSTEM.DEF.SVRCONN です。

-transaction

トランザクションの結果 (例えば commit または rollback など) です。

-mode

トランスポート・モード (例えば managed または unmanaged など) です。

-numberOfMsgs

メッセージの数です。 デフォルトは 1 です。

例

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

IBM MQ .NET でのトランザクションの回復

このセクションでは、管理対象モードを使用している IBM MQ .NET XA におけるトランザクションの回復のプロセスについて説明します。

このタスクについて

分散トランザクション処理では、トランザクションを正常に完了させることができますが、様々な理由でトランザクションが失敗する可能性のあるシナリオが存在する可能性があります。 これらの理由には、システム障害、ハードウェア障害、ネットワーク・エラー、誤ったデータまたは無効なデータ、アプリケーション・エラー、または自然災害や人災などがあります。 トランザクション障害を阻止することはできません。 分散トランザクション・システムは、これらの障害を処理できなければなりません。 エラーが発生したときに、それらのエラーを検出し、訂正できなければなりません。 このプロセスは、トランザクションの回復と呼ばれています。

分散トランザクション処理の重要な側面は、未完了または未確定のトランザクションを回復することです。ある特定のトランザクションの作業単位部分は、それが回復されるまでロックされたままになるので、回復を実行することが重要になります。 Microsoft .NET の System.Transactions クラス・ライブラリーは、未完了/未確定のトランザクションを回復するためのオプションを提供しています。 このリカバリー・サポートでは、リソース・マネージャーがトランザクション・ログを保守し、必要な場合には回復を実行することが期待されています。

Microsoft .NET のトランザクション回復モデルであるトランザクション・マネージャー (System.Transactions または Microsoft Distributed Transaction コーディネーター (MS DTC)、あるいはその両方) は、トランザクションの回復を開始、調整、および管理します。 OLE Tx プロトコル (Microsoft の XA プロトコル) を基礎とするリソース・マネージャーは、それらに代わって回復を実施、調整、および制御するよう DTC を構成するためのオプションを提供しています。 これを行うには、リソース・マネージャーがネイティブ・インターフェースを使用して XA_Switch を MS DTC に登録する必要があります。

XA_Switch は、リソース・マネージャーにおける xa_start、xa_end、および xa_recover などの XA 関数のエントリー・ポイントを分散トランザクション・コーディネーターに提供しています。

Microsoft 分散トランザクション・コーディネーター (DTC) を使用した回復:

Microsoft 分散トランザクション・コーディネーターは、2 種類の回復プロセスを提供しています。

コールド・リカバリー

コールド・リカバリーは、XA リソース・マネージャーとの接続が開いている間にトランザクション・マネージャー・プロセスが失敗すると実行されます。 トランザクション・マネージャーは、再始動すると、トランザクション・マネージャーのログを読み取り、XA リソース・マネージャーとの接続を再確立し、回復を開始します。

ホット・リカバリー

ホット・リカバリーは、XA リソース・マネージャーまたはネットワークが失敗したためにトランザクション・マネージャーと XA リソース・マネージャーとの間の接続が失敗しているのに、トランザクション・マネージャーが起動したままになっている場合に実行されます。 トランザクション・マネージャーは、失敗後に、定期的に XA リソース・マネージャーとの再接続を試みます。 接続が再確立すると、トランザクション・マネージャーは XA の回復を開始します。

System.Transactions 名前空間は、トランザクション・マネージャーとしての MS DTC に基づく分散トランザクションの管理された実装を提供します。 この実装は、MS DTC のネイティブ・インターフェースの機能に類似した機能を提供しますが、完全に管理された環境内にあります。 唯一の違いは、トランザクションの回復に関するものです。 System.Transactions は、リソース・マネージャーそれ自体が回復を実施し、トランザクション・マネージャー (MS DTC) と調整を行うことを期待しています。 リソース・マネージャーは、特定の未完了トランザクションの回復を要求しなければならず、トランザクション・マネージャーはそれを受け入れて、その特定のトランザクションの実際の結果に基づいて調整を行います。

IBM MQ .NET のトランザクション・リカバリー・プロセス

このセクションでは、分散トランザクションを IBM MQ .NET クラスでリカバリーする方法について説明します。

概要

未完了のトランザクションをリカバリーするには、リカバリー情報が必要です。 トランザクション回復情報は、リソース・マネージャーがストレージにログとして書き込まなければなりません。IBM MQ .NET クラスは、同様のパスに従います。 トランザクション回復情報は、SYSTEM.DOTNET.XARECOVERY.QUEUE と呼ばれるシステム・キューにログとして書き込まれます。

IBM MQ .NET でのトランザクション・リカバリーは、次の 2 つの段階のプロセスがあります。

1. SYSTEM.DOTNET.XARECOVERY.QUEUE 内のトランザクション・リカバリー情報のロギング。
2. XA Monitor アプリケーション WmqDotnetXAMonitor を使用したトランザクションのリカバリー。

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEM.DOTNET.XARECOVERY.QUEUE は、不完全なトランザクションのトランザクション・リカバリー情報を保持するシステム・キューです。 このキューは、キュー・マネージャーの作成時に作成されます。

トランザクションごとに、準備フェーズで、回復情報を含む持続メッセージが SYSTEM.DOTNET.XARECOVERY.QUEUE に追加されます。 このメッセージは、コミット呼び出しが成功すると削除されます。

注 : SYSTEM.DOTNET.XARECOVERY.QUEUE キューを削除することはできません。

WMQDotnetXAMonitor アプリケーション

IBM MQ .NET XA Monitor アプリケーション WmqDotnetXAMonitor は、キュー・マネージャーをモニターし、SYSTEM.DOTNET.XARECOVERY. キュー内のメッセージを処理し、不完全なトランザクションをリカバリーする .NET 管理対象アプリケーションです。

メッセージ・チャネル・エージェント (MCA) がメッセージを宛先キューに書き込むことができない場合は、元のメッセージを含む例外レポートが生成され、元のメッセージで指定されている応答先キューに送信される伝送キューに書き込まれます。 (応答先キューが MCA と同じキュー・マネージャー上にある場合は、メッセージは伝送キューには送られず、その応答先キューに直接書き込まれます。)

以下のものは、未完了のトランザクションと見なされ、リカバリーされます。

- トランザクションは作成されたけれども、タイムアウト期間中にコミットが完了しなかった場合。
- トランザクションは作成されたものの、IBM MQ キュー・マネージャーが停止してしまった場合。
- トランザクションは作成されたけれども、トランザクション・マネージャーが停止してしまった場合。

  XA Monitor アプリケーションは、IBM MQ .NET クライアント・アプリケーションが実行されているのと同じシステムから実行する必要があります。 複数のシステムで実行中のアプリケーションがあり、同じキュー・マネージャーに接続するアプリケーションがある場合は、すべてのシステムから WmqDotnetXAMonitor アプリケーションを実行する必要があります。 各クライアント・マシンには、アプリケーションをリカバリーするために実行される XA Monitor アプリケーションのインスタンスがありますが、各 XA Monitor インスタンスは、現在の XA Monitor のローカル MS DTC が調整しているトランザクションに対応するメッセージを識別できるようにする必要があります。これにより、そのトランザクションを再登録して完了することができます。

関連概念

573 ページの『IBM MQ .NET のトランザクション・リカバリーのユース・ケース』
トランザクションのリカバリーの必要が生じる可能性のある、いくつかの異なるユース・ケースがあります。

関連タスク

575 ページの『WMQDotnetXAMonitor アプリケーションの使用』
IBM MQ .NET クライアントは、不完全な分散トランザクションをリカバリーするために使用できる XA モニター・アプリケーション WmqDotnetXAMonitor を提供します。 WmqDotnetXAMonitor アプリケーションは、トランザクションが未確定であるキュー・マネージャーへの接続を確立し、設定したパラメーターに基づいてトランザクションを解決します。

IBM MQ .NET のトランザクション・リカバリーのユース・ケース

トランザクションのリカバリーの必要が生じる可能性のある、いくつかの異なるユース・ケースがあります。

- **単一の DTC および単一のキュー・マネージャー・インスタンスを使用する IBM MQ アプリケーション:** このユース・ケースでは、トランザクションの下でキュー・マネージャーと作業単位 (UoW) に接続するとき。トランザクションが失敗して不完全状態になると、XA モニター・アプリケーションはトランザクションをリカバリーして完了します。

  このユース・ケースでは、単一のキュー・マネージャーがトランザクションに関連付けられているため、XA モニター・アプリケーションの単一のインスタンスが実行されます。

- **単一の DTC および単一のキュー・マネージャー・インスタンスを使用する複数の IBM MQ アプリケーション:** このユース・ケースでは、単一の DTC の下に複数の IBM MQ アプリケーションがあり、すべてが同じキュー・マネージャーに接続され、トランザクションの下で UoW が実行されます。

  トランザクションが失敗して未完了になった場合、XA Monitor アプリケーションはそれらのトランザクションをリカバリーし、すべてのアプリケーションに関連するトランザクションを完了します。

  このユース・ケースでは、トランザクションで 1 つのキュー・マネージャーが使用されるため、XA モニター・アプリケーションの単一インスタンスが実行されます。
- **複数の IBM MQ アプリケーション、複数の DTC、異なるキュー・マネージャー・インスタンス:** このユース・ケースでは、異なる DTC の下に複数の IBM MQ アプリケーション (つまり、各アプリケーションが異なるマシン上で実行されています)、および異なるキュー・マネージャーへの接続があります。

  障害が発生してトランザクションが未完了になると、モニター・アプリケーションがメッセージ内の TransactionManagerWhereabouts を検査して、DTC アドレスを判別します。 TransactionManagerWhereabouts の値が、モニターが実行されている DTC アドレスに一致すると、モニター・アプリケーションは回復を完了し、一致しなければ、DTC に対応するメッセージが見つかるまで検索を続けます。

  このユース・ケースでは、各クライアントがトランザクションで使用する独自のキュー・マネージャーを持つため、クライアント (ユーザーまたはコンピューター) ごとに実行される XA モニター・アプリケーションのインスタンスは 1 つのみになります。
- **複数の IBM MQ アプリケーション、複数の DTC、複数の同じキュー・マネージャー・インスタンス:** このユース・ケースでは、異なる DTC の下に複数の IBM MQ アプリケーションが存在し (各アプリケーションは異なるマシン上で実行されている)、すべてが同じキュー・マネージャーに接続されています。

  障害が発生してトランザクションが未完了になると、モニター・アプリケーションがメッセージ内の TransactionManagerWhereabouts を検査して、DTC アドレスと値が、モニターが実行されている DTC に一致するかどうかをチェックします。 両方の値が一致すると、モニター・アプリケーションは回復を完了し、一致しなければその DTC に対応するメッセージが見つかるまで検索を続けます。

  このユース・ケースでは、各クライアントがトランザクションで使用する独自のキュー・マネージャーを持つため、クライアント (ユーザーまたはコンピューター) ごとに実行される XA モニター・アプリケーションのインスタンスは 1 つのみになります。
- **複数の IBM MQ アプリケーション、単一の DTC、異なるキュー・マネージャー・インスタンス:** このユース・ケースでは、単一の DTC の下に複数の IBM MQ アプリケーション (つまり、コンピューター上で複数の IBM MQ アプリケーションが実行されています)、および異なるキュー・マネージャーへの接続が存在します。

  トランザクションが失敗して未完了になると、モニター・アプリケーションがそのトランザクションを回復します。

  このユース・ケースでは、実行されているモニター・アプリケーションのインスタンスの数と、それらのインスタンスに接続しているキュー・マネージャーの数は同じになります。トランザクションでは、各アプリケーションそれ自体のキュー・マネージャーが使用されており、各トランザクションを回復する必要があるからです。

  注 : XA モニター・アプリケーションがバックグラウンドで実行されていない場合は、そのアプリケーションを開始できます。

関連概念

572 ページの『IBM MQ .NET のトランザクション・リカバリー・プロセス』
このセクションでは、分散トランザクションを IBM MQ .NET クラスでリカバリーする方法について説明します。

関連タスク

575 ページの『WMQDotnetXAMonitor アプリケーションの使用』
IBM MQ .NET クライアントは、不完全な分散トランザクションをリカバリーするために使用できる XA モニター・アプリケーション WmqDotnetXAMonitor を提供します。 WmqDotnetXAMonitor アプリケーションは、トランザクションが未確定であるキュー・マネージャーへの接続を確立し、設定したパラメーターに基づいてトランザクションを解決します。

WMQDotnetXAMonitor アプリケーションの使用

IBM MQ .NET クライアントは、不完全な分散トランザクションをリカバリーするために使用できる XA モニター・アプリケーション WmqDotnetXAMonitor を提供します。 WmqDotnetXAMonitor アプリケーションは、トランザクションが未確定であるキュー・マネージャーへの接続を確立し、設定したパラメーターに基づいてトランザクションを解決します。

このタスクについて

WMQDotnetXAMonitor アプリケーションは手動で実行する必要があります。 このアプリケーションは、いつでも開始できます。 SYSTEM.DOTNET.XARECOVERY.QUEUE でメッセージを確認したらアプリケーションを開始できます。または、IBM MQ .NET クラスを使って書かれたアプリケーションを使ってトランザクション処理をする前にバックグラウンドで実行し続けさせることもできます。

WMQDotnetXAMonitor のパラメーター値は、コマンド行を介するか、アプリケーション構成ファイルを使用するかして設定することができます。 アプリケーション構成ファイルを介して提供される値は、コマンド行を介して設定された値よりも優先されます。

IBM MQ 9.3.0 より前では、 WMQDotnetXAMonitor が確立する接続は非セキュア接続です。

V 9.3.0 IBM MQ 9.3.0 以降には、WMQDotnetXAMonitor に追加のパラメーターを設定することによって、キュー・マネージャーへのセキュア接続を確立するオプションがあります。

手順

- アプリケーション構成ファイルを使用して WmqDotNETXAMonitor に入力を提供するには、577 ページの『WmqDotNETXAMonitor アプリケーション構成ファイルの設定』を参照してください。
- コマンド行から WMQDotnetXAMonitor アプリケーションを開始するには、以下のコマンドを、必要なパラメーターを指定して使用します。

  IBM MQ 9.3.0 の前:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
  ```

  V 9.3.0 IBM MQ 9.3.0 以降:

  ```
  WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
  Repository -s Cipher Spec
  ```

  指定できるパラメーターは以下のとおりです。

  – **-m *QueueManagerName***
  キュー・マネージャーの名前。

  オプション

  -n *ConnectionName*
  host(port) 形式の接続名。 *ConnectionName* には複数の接続名を指定できます。 複数の接続名をコンマ区切りリスト (例えば、`localhost (1414), localhost (1415), localhost (1416)`) に指定する必要があります。 WMQDotnetXAMonitor アプリケーションは、コンマ区切りリストで指定されたそれぞれの接続名に対してリカバリーを実行します。

  -c *ChannelName*
  チャネル名。

  -i
  ヒューリスティック・ブランチ完了。

  オプション

  V 9.3.0 **-k *SSL Key Repository***
  SSL キー・リポジトリーの名前。 サポートされている値は、以下のとおりです。

  - *SYSTEM (これがデフォルト値です)

- * ユーザー

オプション

V 9.3.0 **-s *Cipher Spec***
設定する CipherSpec は、サポートされるバージョンの CipherSpec のいずれかでなければなりません。また、Windows グループ・ポリシーで指定されたものと同じものにすることが理想です。詳しくは、597 ページの『管理対象 .NET クライアントの CipherSpec サポート』を参照してください。

キュー・マネージャーへのセキュア接続を確立するための必須事項。

V 9.3.0 **-dn *SSLPeer Name***
ピア・キュー・マネージャーからの証明書の識別名 (DN) を検査するために使用される SSL ピア名。

オプション

V 9.3.0 **-cl *Certificate Label***
証明書を識別するラベル名。

オプション

V 9.3.0 **-sn *OutboundSNI***
TLS 接続の開始時に Server Name Indication (SNI) をリモート・システムへのターゲット IBM MQ チャネル名に設定するか、ホスト名に設定するか。 このオプションでサポートされる値は次のとおりです。

- CHANNEL (これがデフォルト値です)
- HOSTNAME
- *

値が設定されていない場合は、デフォルト値 (CHANNEL) が使用されます。

オプション

V 9.3.0 **-cr *Certificate Revocation Check***
証明書失効検査を実行するかどうか。 このオプションでサポートされる値は次のとおりです。

- true
- false (これがデフォルト値です)

オプション

V 9.3.0 **-kr *KeyResetCount***
暗号化に使用される秘密鍵が再折衝される前にチャネルで送受信非暗号化バイトの総数。

デフォルト値の 0 は、秘密鍵が再折衝されないことを示します。

オプション

WMQDotnetXAMonitor アプリケーションは、以下のアクションを実行します。

1. 100 秒間隔で SYSTEM.DOTNET.XARECOVERY.QUEUE のキューの深さを確認します。
2. キューの深さが 0 より大きい場合は、メッセージのキューを参照し、メッセージが不完全なトランザクション基準を満たしているかどうかを確認します。
3. メッセージが不完全なトランザクション基準を満たしている場合は、それを引き出し、トランザクション・リカバリー情報を取得します。
4. リカバリー情報がローカル Microsoft 分散トランザクション・コーディネーター (MS DTC) に関連しているかどうかを判別します。 この場合、WMQDotnetXAMonitor がトランザクションのリカバリーを続行します。そうでない場合は、次のメッセージをブラウズするために戻ります。
5. 不完全なトランザクションをリカバリーするために、キュー・マネージャーに呼び出しを行います。

WmqDotNETXAMonitor アプリケーション構成ファイルの設定
アプリケーション構成ファイルを使用して、IBM MQ .NET XA モニター・アプリケーション (WmqDotNETXAMonitor) へ入力を提供することができます。 アプリケーション構成ファイルのサンプルは、IBM MQ .NET に付属しています。 要件に応じてこのサンプル・ファイルを変更することができます。

アプリケーション構成ファイルによって提供される入力値は、最も高い優先順位をとります 575 ページの『WMQDotnetXAMonitor アプリケーションの使用』 およびアプリケーション構成ファイルで説明されているように、コマンド・ラインの両方に入力値を指定すると、アプリケーション構成ファイルの値が優先されます。

IBM MQ 9.3.0 より前のバージョンのサンプル・アプリケーション構成ファイル。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

V 9.3.0 IBM MQ 9.3.0 からのサンプル・アプリケーション構成ファイル。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor アプリケーション・ログ
モニター・アプリケーションは、モニターの進行状況とトランザクションの回復状況をログに記録するためのログ・ファイルをアプリケーション・ディレクトリーに作成します。 ロギングは接続名と、回復が実行されている現在のキュー・マネージャーを表示するためのチャネル詳細で始まります。

回復が開始すると、トランザクション回復メッセージの MessageId、未完了トランザクションの TransactionId、およびトランザクションの実際の結果が、トランザクション・マネージャーの調整に従ってログに書き込まれます。

ログ・ファイルのサンプル:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

IBM MQ .NET プログラムの作成およびデプロイ

IBM MQ classes for .NET を使用して IBM MQ キューにアクセスするには、IBM MQ キュー上へメッセージを置いたり、そこからメッセージを入手したりする呼び出しを含む .NET によってサポートされた任意の言語でプログラムを書きます。

IBM MQ の資料には、C#、C++、および Visual Basic の言語に関する情報のみが含まれます。

この一連のトピックでは、IBM MQ システムと対話するアプリケーションの作成に役立つ情報を紹介します。 個別クラスの詳細は、IBM MQ .NET のクラスとインターフェースを参照してください。

接続の違い

IBM MQ.NET のプログラミング方法は、使用する接続モードによって多少異なります。

IBM MQ classes for .NET が管理対象クライアントとして使用される場合、管理対象クライアントでは一部の機能が使用できないため、標準の IBM MQ MQI client とは異なる点がいくつかあります。

IBM MQ.NET は、使用する接続タイプの判別を、接続名、チャネル名、カスタマイズ値 NMQ_MQ_LIB、およびプロパティー MQC.TRANSPORT_PROPERTY に対して指定された設定に基づいて行います。

管理対象クライアント接続

IBM MQ classes for .NET が管理対象クライアントとして使用されている場合は、標準の IBM MQ MQI client とは異なる点が数多くあります。

管理対象クライアントからは、以下の機能を使用できません。

- チャネル圧縮
- チャネル出口チェーニング

管理対象 クライアントでこれらの機能を使用しようとすると、MQException が返されます。 接続のクライアント側でエラーが検出されると、理由コード MQRC_ENVIRONMENT_ERROR が使用されます。 サーバー側でエラーが検出されると、サーバーから戻された理由コードが使用されます。

非管理対象クライアント用に作成されたチャネル出口は、機能しません。 管理対象クライアント専用の新しい出口を作成する必要があります。 ご使用のクライアント・チャネル定義テーブル (CCDT) に無効なチャネル出口が指定されていないか確認してください。

管理対象チャネル出口の名前の長さは最大 999 文字です。 ただし、CCDT を使用してチャネル出口名を指定する場合は、128 文字までに制限されます。

通信は、TCP/IP を使用した場合にのみサポートされます。

endmqm コマンドを使用してキュー・マネージャーを停止する場合、.NET 管理対象クライアントへのサーバー接続チャネルは、その他のクライアントへのサーバー接続チャネルに比べて、閉じるまでに時間がかかることがあります。

管理対象 IBM MQ 問題診断を使用するために *NMQ_MQ_LIB* を managed に設定した場合、**strmqtrc** コマンドのパラメーター -i、-p、-s、-b、および -c はいずれもサポートされません。

XA トランザクションを使用する管理対象 .NET アプリケーションは、z/OS キュー・マネージャーと連動しません。 管理対象 .NET クライアントは、MQOPEN 呼び出しで z/OS キュー・マネージャーに接続しようと

しますが、エラー MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) で失敗します。 ただし、XA トランザクションを使用する管理対象 .NET アプリケーションは、分散キュー・マネージャーと連動します。

使用する接続タイプの定義

接続タイプは、接続名、チャネル名、カスタマイズ値 NMQ_MQ_LIB、およびプロパティー MQC.TRANSPORT_PROPERTY の設定によって指定されます。

接続名は次のように指定できます。

- MQQueueManager コンストラクターで明示的に指定。

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーの、プロパティー MQC.HOST_NAME_PROPERTY とオプションの MQC.PORT_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 明示的な MQEnvironment 値として設定。

```
MQEnvironment.Hostname
```

  MQEnvironment.Port(オプション)。

- MQEnvironment.properties ハッシュ・テーブルの、プロパティー MQC.HOST_NAME_PROPERTY とオプションの MQC.PORT_PROPERTY で設定。

チャネル名は次のように指定できます。

- MQQueueManager コンストラクターで明示的に指定。

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーのプロパティー MQC.CHANNEL_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 明示的な MQEnvironment 値として設定。

```
MQEnvironment.Channel
```

- MQEnvironment.properties ハッシュ・テーブルのプロパティー MQC.CHANNEL_PROPERTY で設定。

TRANSPORT プロパティーは次のように指定できます。

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーのプロパティー MQC.TRANSPORT_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- MQEnvironment.properties ハッシュ・テーブルのプロパティー MQC.TRANSPORT_PROPERTY で設定。

以下のいずれかの値を使用して、必要な接続タイプを選択してください。

MQC.TRANSPORT_MQSERIES_BINDINGS - サーバーとして接続
MQC.TRANSPORT_MQSERIES_CLIENT - 非 XA クライアントとして接続
MQC.TRANSPORT_MQSERIES_XACLIENT - XA クライアントとして接続
MQC.TRANSPORT_MQSERIES_MANAGED - 非 XA 管理対象クライアントとして接続

カスタマイズ値 NMQ_MQ_LIB を設定することで、次の表に示す接続タイプを明示的に選択できます。

NMQ_MQ_LIB 値	**接続タイプ**
mqic.dll	非 XA クライアントとして接続
mqicxa.dll	XA クライアントとして接続
mqm.dll	サーバーまたは非 XA クライアントとして接続
managed	非 XA 管理対象クライアントとして接続

注 : 旧リリースとの互換性を保つため、値 mqic32.dll および mqic32xa.dll は mqic.dll および mqicxa.dll の同義として受け入れられます。 ただし、mqm.dll および mqm.pdb は、IBM WebSphere MQ 7.1 以降のクライアント・パッケージの一部でしかありません。

お客様の環境で使用できない接続タイプを選択すると (例えば mqic32xa.dll を指定したが XA サポートがない場合)、IBM MQ .NET によって例外がスローされます。

NMQ_MQ_LIB を「managed」に設定すると、クライアントは管理対象の IBM MQ 問題診断テストおよび .NET データ変換などの管理対象の低レベル IBM MQ 機能を使用するようになります。

NMQ_MQ_LIB にそれ以外のすべての値を設定した場合、.NET プロセスは非管理対象の IBM MQ 問題診断テストおよびデータ変換などの非管理対象の低レベル IBM MQ 機能を使用するようになります (IBM MQ MQI client またはサーバーがシステムにインストールされていることが前提)。

IBM MQ .NET は次のように接続タイプを選択します。

1. MQC.TRANSPORT_PROPERTY が指定されている場合は、MQC.TRANSPORT_PROPERTY の値に従って接続します。

   ただし、MQC.TRANSPORT_PROPERTY を MQC.TRANSPORT_MQSERIES_MANAGED に設定しても、クライアント・プロセスが管理下で実行されることが保証されるわけではないので注意してください。 この設定を行っていても、以下のケースではクライアントは管理対象になりません。

   - プロセス内の他のスレッドが、MQC.TRANSPORT_MQSERIES_MANAGED 以外の値に設定された MQC.TRANSPORT_PROPERTY に接続している場合。
   - NMQ_MQ_LIB が「managed」に設定されていない場合、問題診断テスト、データ変換、および他の低レベル関数は、完全には管理対象になりません (IBM MQ MQI client またはサーバーがシステムにインストールされていることを想定しています)。
2. 接続名が指定され、チャネル名が指定されていない場合、またはチャネル名が指定され、接続名が指定されていない場合は、エラーがスローされます。
3. 接続名とチャネル名の両方が指定されている場合は、次のようになります。
   - NMQ_MQ_LIB が mqic32xa.dll に設定されている場合は、XA クライアントとして接続します。
   - NMQ_MQ_LIB が managed に設定されている場合は、管理対象クライアントとして接続します。
   - それ以外の場合は、非 XA クライアントとして接続します。
4. NMQ_MQ_LIB が指定されている場合は、NMQ_MQ_LIB の値に従って接続します。
5. IBM MQ サーバーがインストールされている場合は、サーバーとして接続します。
6. IBM MQ MQI client がインストールされている場合は、非 XA クライアントとして接続します。
7. それ以外の場合は、管理対象クライアントとして接続します。

Windows IBM MQ .NET プロジェクト・テンプレートの使用

IBM MQ .NET クライアントには、プロジェクト・テンプレートを使用して .NET Core アプリケーションの開発を支援する機能があります。

始める前に

ご使用のシステムに Microsoft Visual Studio 2017 以降、および .NET Core 2.1 がインストールされている必要があります。

.NET テンプレートを以下からコピーする必要があります。

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

へのディレクトリー

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

ディレクトリー。ここで、

- *&MQ_INSTALL_ROOT* はインストール済み環境のルート・ディレクトリーです。
- *&USER_HOME_DIRECTORY* はユーザーのホーム・ディレクトリーです。

テンプレートを選択するには、Microsoft Visual Studio を停止してから再始動する必要があります。

このタスクについて

.NET プロジェクト・テンプレートには、アプリケーションの開発を支援するために使用できる一般的なコードが含まれています。 組み込みコードを使用すると、組み込みコード内のプロパティーを変更するだけで IBM MQ キュー・マネージャーに接続し、書き込みや取得の操作を実行できます。

手順

1. Microsoft Visual Studio を開きます。
2. 「**ファイル**」をクリックしてから「**新規作成**」をクリックし、次に「**プロジェクト**」をクリックします。
3. 「新規プロジェクトの作成」ウィンドウで、 IBM MQ .NET Client App (.NET Core) を選択し、「**次へ**」をクリックします。
4. 「新しいプロジェクトの構成」ウィンドウで、必要に応じてプロジェクトの「プロジェクト名」を変更し、「**作成**」をクリックして .NET プロジェクトを作成します。

   MQDotnetApp.cs は、プロジェクト・ファイルとともに作成されるファイルです。 このファイルには、キュー・マネージャーに接続し、書き込みや取得の操作を実行するコードが含まれています。

   接続プロパティーは、次のデフォルト値に設定されます。

   - MQC.CONNECTION_NAME_PROPERTY は *localhost(1414)* に設定されます
   - MQC.CHANNEL_PROPERTY は *DOTNET.SVRCONN* に設定されます

   キューは *Q1* に設定され、これらのプロパティーは適宜変更できます。
5. アプリケーションをコンパイルおよび実行します。

関連概念

IBM MQ のコンポーネントと機能

.NET アプリケーション・ランタイム - Windows のみ

IBM MQ classes for .NET の構成ファイル

.NET クライアント・アプリケーションは、IBM MQ MQI client 構成ファイルと、管理対象接続タイプを使用している場合は .NET アプリケーション構成ファイルを使用することができます。 アプリケーション構成ファイルの設定が優先されます。

クライアント構成ファイル

IBM MQ classes for .NET クライアント・アプリケーションでは、他のすべての IBM MQ MQI client と同じ方法でクライアント構成ファイルを使用できます。 通常、このファイルは `mqclient.ini` と呼ばれますが、別のファイル名を指定できます。 クライアント構成ファイルについて詳しくは、 IBM MQ MQI client 構成ファイル `mqclient.ini` を参照してください。

IBM MQ MQI client 構成ファイル中の属性のうち、IBM MQ classes for .NET に関係するのは以下のものだけです。 その他の属性を指定しても、効果はありません。

表 77. *IBM MQ classes for .NET* に関連するクライアント構成ファイルの属性

スタンザ	属性
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ReconDelay
CHANNELS	DefRecon
CHANNELS	MQReconnectTimeout
CHANNELS	ServerConnectionParms
CHANNELS	Put1DefaultAlwaysSync
CHANNELS	PasswordProtection
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
V 9.3.0 V 9.3.0 セキュリティー	MQIInitialKey ファイル
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository
V 9.3.0 V 9.3.0 SSL	SSLKeyRepository パスワード
tcp	ClntRcvBufSize
tcp	ClntSndBufSize
tcp	IPAddressVersion
tcp	KeepAlive

これらの属性はすべて、該当する環境変数を使用して指定変更することができます。

アプリケーション構成ファイル

管理対象接続タイプを使用して実行している場合は、 .NET アプリケーション構成ファイルを使用して、IBM MQ クライアント構成ファイルおよび同等の環境変数をオーバーライドすることもできます。

.NET アプリケーション構成ファイルの設定は、管理対象接続タイプで実行されている場合にのみ有効で、その他の接続タイプでは無視されます。

.NET アプリケーション構成ファイルとそのフォーマットは、 .NET フレームワーク内で一般的に使用するために Microsoft によって定義されていますが、この資料で言及されている特定のセクション名、キー、および値は IBM MQ に固有のものです。

.NET アプリケーション構成ファイルのフォーマットは、数多くのセクション で構成されます。 各セクションには、1 つ以上のキー が含まれ、各キーには、関連付けられた値 があります。 次の例に、TCP/IP KeepAlive プロパティーを制御するために .NET アプリケーション構成ファイルで使用されている、セクション、キー、および値を示します。

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

.NET アプリケーション構成ファイルのセクション名およびキーで使用されたキーワードは、クライアント構成ファイルで定義されたスタンザおよび属性のキーワードと完全に一致します。

セクション <configSections> は、<configuration> 要素の最初の子エレメントである必要があります。

詳しくは、Microsoft の資料を参照してください。

.NET で使用する C# コード・フラグメントの例

アプリケーションがキュー・マネージャーに接続して、キューにメッセージを書き込み、応答を受信する実例を示す C# コード・フラグメント。

以下の C# コードの例は、次の 3 つのアクションを実行するアプリケーションを示しています。

1. キュー・マネージャーに接続します。
2. メッセージを SYSTEM.DEFAULT.LOCAL.QUEUE に書き込みます。
3. メッセージを返します。

また、接続タイプを変更する方法も示します。

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
```

```
/// </summary>
/// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
static Hashtable init(String connectionType)
{
  Hashtable connectionProperties = new Hashtable();

  // Add the connection type
  connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

  // Set up the rest of the connection properties, based on the
  // connection type requested
  switch(connectionType)
  {
    case MQC.TRANSPORT_MQSERIES_BINDINGS:
      break;
    case MQC.TRANSPORT_MQSERIES_CLIENT:
    case MQC.TRANSPORT_MQSERIES_XACLIENT:
    case MQC.TRANSPORT_MQSERIES_MANAGED:
      connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
      connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
      break;
  }

  return connectionProperties;
}
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static int Main(string[] args)
{
  try
  {
    Hashtable connectionProperties = init(connectionType);

    // Create a connection to the queue manager using the connection
    // properties just defined
    MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

    // Set up the options on the queue we want to open
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

    // Now specify the queue that we want to open,and the open options
    MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

    // Define an IBM MQ message, writing some text in UTF format
    MQMessage hello_world = new MQMessage();
    hello_world.WriteUTF("Hello World!");

    // Specify the message options
    MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                         // same as MQPMO_DEFAULT

    // Put the message on the queue
    system_default_local_queue.Put(hello_world, pmo);



    // Get the message back again

    // First define an IBM MQ message buffer to receive the message
    MQMessage retrievedMessage =new MQMessage();
    retrievedMessage.MessageId =hello_world.MessageId;

    // Set the get message options
    MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                        //same as MQGMO_DEFAULT

    // Get the message off the queue
    system_default_local_queue.Get(retrievedMessage,gmo);

    // Prove we have the message by displaying the UTF message text
    String msgText = retrievedMessage.ReadUTF();
    Console.WriteLine("The message is: {0}", msgText);

    // Close the queue
    system_default_local_queue.Close();

    // Disconnect from the queue manager
    qMgr.Disconnect();
  }
```

```
    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

IBM MQ 環境のセットアップ

クライアント接続を使用してキュー・マネージャーに接続するには、まず、IBM MQ 環境をセットアップする必要があります。

注 : サーバー・バインディング・モードで IBM MQ classes for .NET を使用する場合、この手順は不要です。

.NET プログラミング・インターフェースにより、NMQ_MQ_LIB カスタマイズ値を使用することができ、クラス MQEnvironment も組み込まれます。 このクラスでは、接続を試行する際に使用される、以下にリストしたような詳細情報を指定できます。

- チャネル名
- ホスト名
- ポート番号
- チャネル出口
- SSL パラメーター
- ユーザー ID およびパスワード

MQEnvironment クラスについて詳しくは、 MQEnvironment.NET クラス を参照してください。

チャネル名とホスト名を指定するには、次のコードを使用します。

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

デフォルトでは、クライアントはポート 1414 で IBM MQ リスナーに接続しようとします。 別のポートを指定するには、次のコードを使用します。

```
MQEnvironment.Port = nnnn;
```

キュー・マネージャーへの接続とキュー・マネージャーからの切断

IBM MQ 環境を構成したら、キュー・マネージャーに接続する準備ができました。

キュー・マネージャーに接続するには、次のように MQQueueManager クラスの新規インスタンスを作成します。

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

キュー・マネージャーから切断するには、キュー・マネージャーで Disconnect メソッドを呼び出します。

```
queueManager.Disconnect();
```

キュー・マネージャーに接続するには、キュー・マネージャーに対する照会 (inq) 権限が必要です。 照会権限がない場合、接続試行は失敗します。

Disconnect メソッドを呼び出すと、そのキュー・マネージャーからアクセスした、オープンしているキューとプロセスがすべてクローズされます。 しかし、使用を終えた時点でこれらのリソースを明示的にクローズするプログラミングの習慣を付けておくことをお勧めします。 リソースをクローズするには、各リソースに関連付けられているオブジェクトで Close メソッドを使用します。

キュー・マネージャーの Commit メソッドと Backout メソッドは、手続き型インターフェースで使用される MQCMIT 呼び出しと MQBACK 呼び出しを置き換えるものです。

キューおよびトピックへのアクセス

MQQueueManager のメソッドまたは適切なコンストラクターを使用して、キューとトピックにアクセスできます。

キューにアクセスするには、MQQueueManager クラスのメソッドを使用します。 MQOD (オブジェクト記述子構造体) は、これらのメソッドのパラメーターに縮小されます。 例えば、MQQueueManager オブジェクトで表される queueManager というキュー・マネージャーのキューをオープンするには、次のコードを使用します。

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

options パラメーターは、MQOPEN 呼び出しの Options パラメーターと同じです。

AccessQueue メソッドは、クラス MQQueue の新しいオブジェクトを戻します。

キューの使用が終了したら、次の例に示すように、Close() メソッドを使用してそのキューをクローズします。

```
queue.Close();
```

IBM MQ .NET では、MQQueue コンストラクターを使用してキューを作成することもできます。 パラメーターは、accessQueue メソッドと同じパラメーター、および使用するインスタンス化済みの MQQueueManager オブジェクトを指定するキュー・マネージャー・パラメーターです。 以下に例を示します。

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

この方法でキュー・オブジェクトを構成すると、MQQueue の独自のサブクラスを作成できます。

同様に、MQQueueManager クラスのメソッドを使用してトピックにもアクセスすることができます。 トピックを開くには、AccessTopic() メソッドを使用します。 このメソッドは、クラス MQTopic の新規オブジェクトを返します。 トピックの使用が完了したなら、MQTopic の Close() メソッドを使用してトピックを閉じます。

MQTopic コンストラクターを使用してトピックを作成することもできます。 トピック用のコンストラクターはいくつもあります。詳しくは、MQTopic .NET クラスを参照してください。

メッセージの処理

メッセージは、キューまたはトピック・クラスのメソッドを使用して処理されます。 新しいメッセージを作成するには、新しい MQMessageobject を作成します。

キューまたはトピックへのメッセージの書き込みには、MQQueue または MQTopic クラスの Put() メソッドを使用します。 キューまたはトピックからのメッセージの読み取りには、MQQueue または MQTopic クラスの Get() メソッドを使用します。 MQPUT および MQGET でバイトの配列の書き込みと取得を行うプロシージャー型インターフェースとは異なり、IBM MQ classes for .NET では MQMessage クラスのインスタンスの書き込みと取得が行われます。 MQMessage クラスは、実際のメッセージ・データが含まれるデータ・バッファーと、そのメッセージについて記述するすべての MQMD (メッセージ記述子) パラメーターをまとめてカプセル化します。

新しいメッセージを作成するには、MQMessage クラスの新しいインスタンスを作成し、WriteXXX メソッドを使用してメッセージ・バッファーにデータを書き込みます。

新しいメッセージ・インスタンスが作成されると、すべての MQMD パラメーターがデフォルト値に自動的に設定されます。 詳細については、MQMD の初期値および言語ごとの宣言を参照してください。
MQQueue の Put() メソッドも、MQPutMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQPMO 構造体を表します。 次の例では、メッセージを作成して、キューに書き込んでいます。

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message
!queue.Put(myMessage,pmo);
```

MQQueue の Get() メソッドは、キューから読み取ったメッセージを表す MQMessage の新しいインスタンスを返します。 また、このメソッドは MQGetMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQGMO 構造体を表します。

Get() メソッドは着信メッセージに合わせて内部バッファーのサイズを自動調整するため、最大メッセージ・サイズを指定する必要はありません。 返されたメッセージのデータにアクセスするには、MQMessage クラスの ReadXXX メソッドを使用します。

次の例は、メッセージをキューから読み取る方法を示しています。

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

encoding メンバー変数を設定することにより、読み取りメソッドと書き込みメソッドで使用する数字形式を変更できます。

characterSet メンバー変数を設定することにより、読み取りと書き込みのストリングで使用する文字セットを変更できます。

詳しくは、MQMessage .NET クラスを参照してください。

注 : MQMessage の WriteUTF() メソッドは、ストリングに含まれる Unicode バイトだけでなく、ストリングの長さを自動的にエンコードします。 メッセージが別の .NET プログラムによって (readUTF() を使用して) 読み取られる場合、これがストリング情報を送信する最も簡単な方法です。

メッセージ・プロパティーの処理

メッセージ・プロパティーでは、メッセージを選択したり、メッセージのヘッダーにアクセスすることなくメッセージについての情報を取得したりすることができます。 MQMessage クラスには、プロパティーの取得および設定を行うメソッドが含まれています。

メッセージ・プロパティーを使用して、アプリケーションに処理するメッセージを選択させたり、MQMD または MQRFH2 ヘッダーにアクセスすることなくメッセージについての情報を取得させたりすることができます。 また、メッセージ・プロパティーは、IBM MQ と JMS アプリケーションの間の通信を行いやすくします。 IBM MQ のメッセージ・プロパティーについて詳しくは、メッセージ・プロパティーを参照してください。

MQMessage クラスには、プロパティーのデータ型に従ってプロパティーを取得および設定するいくつものメソッドが用意されています。 取得のメソッドには Get*Property という形式の名前が付いており、設定のメソッドには Set*Property という形式の名前が付いています。アスタリスク (*) の部分には、以下のいずれかのストリングが入ります。

- Boolean
- Byte
- バイト
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- オブジェクト
- 短
- ストリング

例えば、IBM MQ プロパティー myproperty (文字ストリング) を取得するには、呼び出し `message.GetStringProperty('myproperty')` を使用します。 オプションで、プロパティー記述子を渡すことができます。プロパティー記述子を追加する処理は IBM MQ によって行われます。

エラーの処理

IBM MQ classes for .NET で発生したエラーを、`try` および `catch` ブロックを使用して処理します。

.NET インターフェースのメソッドは完了コードと理由コードを戻しません。 その代わりに、IBM MQ の呼び出しによる完了コードと理由コードが両方ともゼロでない場合は必ず例外をスローします。 これによってプログラム・ロジックが簡単になり、IBM MQ への呼び出しごとに戻りコードを検査する必要がなくなります。 プログラムのどの部分で障害に対処するかを決めることができます。 これらのポイントでは、次の例のように、コードを `try` と `catch` のブロックで囲むことができます。

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                        "CC = " + ex.CompletionCode +
                        "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

属性値の取得と設定

クラス MQManagedObject、MQQueue、および MQQueueManager には、属性値を取得または設定できるメソッドがあります。 MQQueue では、キューのオープン時に適切な照会フラグおよび設定フラグを指定している場合にのみメソッドが機能することに注意してください。

共通属性に関しては、MQQueueManager クラスと MQQueue クラスは MQManagedObject というクラスから継承します。 このクラスは、Inquire() インターフェースと Set() インターフェースを定義します。

new 演算子を使用して新規キュー・マネージャー・オブジェクトを作成すると、そのオブジェクトは自動的に inquire 用にオープンされます。 AccessQueue() メソッドを使用してキュー・オブジェクトにアクセスした場合、オブジェクトは照会操作用または設定操作用に自動的にはオープンされません。これにより、一部のタイプのリモート・キューでは問題が発生することがあります。 Inquire メソッドと Set メソッドを使用してキューのプロパティーを設定するには、AccessQueue() メソッドの openOptions パラメーターで適切な照会フラグと設定フラグを指定する必要があります。

inquire メソッドと set メソッドは、次の 3 つのパラメーターを取ります。

- selectors 配列
- intAttrs 配列
- charAttrs 配列

配列の長さは常に判明しているため、MQINQ で使用する SelectorCount、IntAttrCount、および CharAttrLength の各パラメーターは必要ありません。 次の例は、キューの照会を行う方法を示しています。

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

これらのオブジェクトの属性はすべて照会可能です。 属性のサブセットはオブジェクトのプロパティーとして公開されます。 オブジェクトの属性のリストについては、オブジェクトの属性を参照してください。 オブジェクトのプロパティーについては、該当するクラスの説明を参照してください。

マルチスレッド・プログラム

.NET ランタイム環境は本質的にマルチスレッド環境です。 IBM MQ classes for .NET では、キュー・マネージャー・オブジェクトを複数のスレッドで共用できますが、ターゲット・キュー・マネージャーへのすべてのアクセスが確実に同期するようになります。

始動時にキュー・マネージャーに接続して、キューをオープンする単純なプログラムを考えてみましょう。このプログラムは、画面上に 1 つのボタンを表示します。 ユーザーがこのボタンをクリックすると、プログラムはキューからメッセージを取り出します。 この場合、アプリケーションの初期化は 1 つのスレッドで行われ、ボタンが押されたことに対する応答としてのコードの実行は別のスレッド (ユーザー・インターフェース・スレッド) で行われます。

IBM MQ .NET を実装すると、特定の接続 (MQQueueManager オブジェクト・インスタンス) に対しては、ターゲットの IBM MQ キュー・マネージャーへのすべてのアクセスは、必ず同期化されます。 デフォルトの動作では、キュー・マネージャーに呼び出しを行ったスレッドは、その接続に対して行われている他のすべての呼び出しが完了するまでブロックされます。 プログラム内の複数のスレッドから同じキュー・マネージャーに同時にアクセスする必要がある場合は、同時アクセスが必要なスレッドごとに新しい MQQueueManager オブジェクトを作成します。 (これは、スレッドごとに別の MQCONN 呼び出しを発行する ことと同じです。)

デフォルトの接続オプションが MQC.MQCNO_HANDLE_SHARE_NONE または MQC.MQCNO_SHARE_NO_BLOCK によって指定変更された場合、キュー・マネージャーは同期しなくなります。

.NET を含むクライアント・チャネル定義テーブルの使用

IBM MQ classes for .NET では、クライアント・チャネル定義テーブル (CCDT) を使用できます。 管理対象接続を使用しているか、非管理対象接続を使用しているかに応じて、CCDT の場所を指定する方法は異なります。

非 XA または XA 非管理対象クライアント接続タイプ

非管理対象接続タイプを使用して、次の 2 つの方法で CCDT の場所を指定できます。

- 環境変数 MQCHLLIB を使用して、テーブルが置かれているディレクトリーを指定し、MQCHLTAB を使用して、テーブルのファイル名を指定します。
- クライアント構成ファイルを使用します。 CHANNELS スタンザで、属性 **ChannelDefinitionDirectory** を使用してテーブルが置かれているディレクトリーを指定し、**ChannelDefinitionFile** を使用してファイル名を指定します。

クライアント構成ファイルと環境変数の両方で場所が指定されている場合は、環境変数が優先します。 この機能を使用して、標準の位置をクライアント構成ファイルで指定し、必要な場合に環境変数を使用して標準の位置をオーバーライドすることができます。

管理対象クライアント接続タイプ

管理対象接続タイプを使用して、次の 3 つの方法で CCDT の場所を指定できます。

- .NET アプリケーション構成ファイルを使用します。 CHANNELS セクションで、キー **ChannelDefinitionDirectory** を使用してテーブルが置かれているディレクトリーを指定し、**ChannelDefinitionFile** を使用してファイル名を指定します。
- 環境変数 MQCHLLIB を使用して、テーブルが置かれているディレクトリーを指定し、MQCHLTAB を使用して、テーブルのファイル名を指定します。
- クライアント構成ファイルを使用します。 CHANNELS スタンザで、属性 **ChannelDefinitionDirectory** を使用してテーブルが置かれているディレクトリーを指定し、**ChannelDefinitionFile** を使用してファイル名を指定します。

複数の方法で場所が指定されている場合は、環境変数がクライアント構成ファイルに優先し、.NET アプリケーション構成ファイルがその他の 2 つの方法に優先します。 このフィーチャーを使用することで、クライアント構成ファイルで標準の場所を指定し、必要に応じて、環境変数またはアプリケーション構成ファイルを使用して標準の場所を指定変更できます。

IBM MQ 9.3.0 より前では、キュー・マネージャーのグループ化で CCDT を使用する場合、管理対象 .NET クライアントと IBM MQ Java および C クライアントの間で動作に違いがあります。 CCDT ファイルに、同じ 3 つのキュー・マネージャーに対する 3 つのキュー・マネージャー・グループと 3 つの明示的な CLNTCONN が含まれており、アプリケーションが "*" をキュー・マネージャーとして提供する場合、C および Java クライアントは MQRC_Q_MGR_NAME_ERROR を返します。 ただし、管理対象 .NET クライアントは、使用可能な最初の CLNTCONN を使用し、使用可能な CLNTCONN がない場合は、キュー・マネージャーのグループ化された CLNTCONN を使用します。

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、.NET クライアントは C および Java クライアントと同じように動作し、キュー・マネージャーのグループ化で CCDT を使用すると MQRC_Q_MGR_NAME_ERROR を返します。

どのチャネル定義を使用するかを .NET アプリケーションが判別する方法

IBM MQ .NET クライアント環境では、使用するチャネル定義をさまざまな方法で指定できます。 チャネル定義の指定が複数存在することもあります。 アプリケーションは、1 つ以上のソースからチャネル定義を取り出します。

複数のチャネル定義が存在する場合は、次の優先順位で、使用する 1 つのチャネル定義が選択されます。

1. MQQueueManager コンストラクターで、明示的に、またはハッシュ・テーブルに *MQC.CHANNEL_PROPERTY* を組み込むことによって、指定されるプロパティー。

2. MQEnvironment.properties ハッシュ・テーブル内のプロパティー *MQC.CHANNEL_PROPERTY*。
3. MQEnvironment 内のプロパティー *Channel*。
4. .NET アプリケーション構成ファイル、セクション名 CHANNELS、キー ServerConnectionParms (管理対象接続にのみ適用)
5. *MQSERVER* 環境変数。
6. クライアント構成ファイル、スタンザ CHANNELS、属性 ServerConnectionParms
7. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、.NET アプリケーション構成ファイルで指定されます (管理対象接続にのみ適用)。
8. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、環境変数 *MQCHLIB* および *MQCHLTAB* を使用して指定されます。
9. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、クライアント構成ファイルを使用して指定されます。

項目 1 から 3 まででは、チャネル定義は、アプリケーションによって提供された値から、フィールド単位で作成されます。 これらの値は、さまざまなインターフェースを使用して提供することができ、インターフェースごとに複数の値が存在することもあります。 フィールド値は、次の優先順位に従って、チャネル定義に追加されます。

1. MQQueueManager コンストラクター上の *connName* の値。
2. MQQueueManager.properties ハッシュ・テーブルからのプロパティーの値。
3. MQEnvironment.properties ハッシュ・テーブルからのプロパティーの値。
4. MQEnvironment フィールドとして設定された値 (例えば、MQEnvironment.Hostname、MQEnvironment.Port)

項目 4 から 6 まででは、チャネル定義全体が値として提供されます。 チャネル定義で指定されていないフィールドは、システム・デフォルトを取ります。 チャネルとそのフィールドを定義するその他の方法からの値が、これらの指定にマージされることはありません。

項目 7 から 9 まででは、チャネル定義全体が CCDT から取得されます。 チャネルが定義されたときに明示的に指定されなかったフィールドは、システム・デフォルトを取ります。 チャネルとそのフィールドを定義するその他の方法からの値が、これらの指定にマージされることはありません。

IBM MQ .NET でのチャネル出口の使用

クライアント・バインディングを使用する場合は、他のクライアント接続の場合のようにチャネル出口を使用できます。 管理対象バインディングを使用する場合は、適切なインターフェースを実装する出口プログラムを作成する必要があります。

クライアント・バインディング

クライアント・バインディングを使用すると、チャネル出口で説明されている方法でチャネル出口を使用できます。 管理対象バインディング用に作成したチャネル出口は使用できません。

管理対象バインディング

管理対象接続を使用している場合は、出口を実装するために、適切なインターフェースを実装する新しい .NET クラスを定義します。 IBM MQ パッケージには次の 3 つの出口インターフェースが定義されています。

- MQSendExit
- MQReceiveExit
- MQSecurityExit

注 : このインターフェースを使用して作成されたユーザー出口は、非管理対象環境ではチャネル出口としてサポートされません。

次の例では、3 つすべてを実装するクラスを定義しています。

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

各出口は、MQChannelExit および MQChannelDefinition オブジェクトのインスタンスが渡されます。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

送信出口によって送信されるデータと、セキュリティー出口または受信出口によって受信されるデータは、出口のパラメーターを使用して指定されます。

オフセット *dataOffset* で長さ *dataLength* のバイト配列 *dataBuffer* のデータは、チャネル出口が送信出口であれば、送信出口で送信されようとしているデータです。チャネル出口がセキュリティー出口または受信出口であれば、セキュリティー出口または受信出口で受信されたデータです。 パラメーター *dataMaxLength* は、*dataBuffer* の出口で使用可能な (*dataOffset* からの) 最大長を示します。 注: セキュリティー出口では、 出口が初めて呼び出された場合、またはパートナー側がデータを送信しないことを初めて選択した場合は、dataBuffer がヌルになることがあります。

戻りでは、*dataOffset* と *dataLength* の値は、返されたバイト配列内のオフセットと 長さを示すように設定される必要があります。返されたバイト配列は、次に .NET クラスが使用します。 送信出口では、これは送信するデータを示します。 セキュリティー出口または受信出口では、解釈されるデータです。 通常、出口はバイト配列を返します (例外として、データを送信しないことを選択できるセキュリティー出口、および INIT または TERM という理由で呼び出されるすべての出口があります)。 したがって、作成できる最も単純な出口は、dataBuffer だけを戻す出口です。

次は、考えられる最も単純な出口の本体です。

```
{
  return dataBuffer;
}
```

MQChannelDefinition クラス

管理対象 .NET クライアント・アプリケーションで指定されたユーザー ID とパスワードは、クライアント・セキュリティー出口に渡される IBM MQ .NET MQChannelDefinition クラスで設定されます。 セキュリティー出口はユーザー ID とパスワードを MQCD.RemoteUserIdentifier フィールドと MQCD.RemotePassword フィールドにコピーします (979 ページの『セキュリティー出口の作成』を参照してください)。

チャネル出口の指定 (管理対象クライアント)

MQQueueManager オブジェクトを (MQEnvironment または MQQueueManager コンストラクターで) 作成する際にチャネル名と接続名を指定する場合、チャネル出口を 2 つの方法で指定できます。

この 2 つの方法を優先順に示します。

1. MQQueueManager コンストラクターでのハッシュ・テーブル・プロパティー MQC.SECURITY_EXIT_PROPERTY、MQC.SEND_EXIT_PROPERTY、 または MQC.RECEIVE_EXIT_PROPERTY の引き渡し
2. MQEnvironment の SecurityExit、SendExit、または ReceiveExit プロパティーの設定

チャネル名と接続名を指定しない場合、チャネル出口は、クライアント・チャネル定義テーブル (CCDT) から選択されたチャネル定義から使用されます。 チャネル定義に保管されている値を指定変更することはできません。 チャネル定義テーブルについて詳しくは、クライアント・チャネル定義テーブルおよび 590 ページの『.NET を含むクライアント・チャネル定義テーブルの使用』を参照してください。

どちらの場合も、次の形式のストリングで指定します。

```
Assembly_name(Class_name)
```

Class_name は、IBM.WMQ.MQSecurityExit、IBM.WMQ.MQSendExit、または IBM.WMQ.MQReceiveExit インターフェースを (必要に応じて) 実装する .NET クラスの完全修飾名 (名前空間の指定を含む) です。 *Assembly_name* は、このクラスを含むアセンブリーの、ファイル拡張子を含めて完全修飾された場所です。 MQEnvironment または MQQueueManager のプロパティーを使用する場合、ストリングの長さは最大 999 文字です。 ただし、チャネル出口名が CCDT に指定されている場合、128 文字までに制限されます。 必要に応じて、.NET クライアント・コードは 指定されたストリングを解析し、指定されたクラスのインスタンスをロードして作成します。

チャネル出口ユーザー・データの指定 (管理対象クライアント)

チャネル出口は、関連するユーザー・データを持つことができます。 MQQueueManager オブジェクトを (MQEnvironment または MQQueueManager コンストラクターで) 作成する際にチャネル名と接続名を指定する場合、ユーザー・データを 2 つの方法で指定できます。

この 2 つの方法を優先順に示します。

1. MQQueueManager コンストラクターでのハッシュ・テーブル・プロパティー MQC.SECURITY_USERDATA_PROPERTY、MQC.SEND_USERDATA_PROPERTY、 または MQC.RECEIVE_USERDATA_PROPERTY の引き渡し
2. MQEnvironment の SecurityUserData、SendUserData、または ReceiveUserData プロパティーの設定

チャネル名と接続名を指定しなかった場合は、クライアント・チャネル定義テーブル (CCDT) から選択されたチャネル定義からの出口ユーザー・データ値が使用されます。 チャネル定義に保管されている値を指定変更することはできません。 チャネル定義テーブルについて詳しくは、クライアント・チャネル定義テーブルおよび 590 ページの『.NET を含むクライアント・チャネル定義テーブルの使用』を参照してください。

どちらの場合も、最大 32 文字のストリングで指定します。

.NET でのクライアントの自動再接続

予測しない接続中断時にクライアントが自動的にキュー・マネージャーに再接続するように設定できます。

クライアントは、キュー・マネージャーが停止したりネットワークやサーバーに障害が発生したりした場合に、 予期せずにキュー・マネージャーから切断されることがあります。

クライアント自動再接続を使用しない場合、接続に障害が発生したときにエラーが生成されます。 エラー・コードは、接続を再確立するのに役立ちます。

クライアント自動再接続機能を使用するクライアントは、再接続可能なクライアントと呼ばれます。 再接続可能なクライアントを作成するには、 キュー・マネージャーへの接続時に、再接続オプションと呼ばれる特定のオプションを指定します。

クライアント・アプリケーションが IBM MQ .NET クライアントである場合、クライアント自動再接続をオンにするには、MQQueueManager クラスを使用してキュー・マネージャーを作成する際に CONNECT_OPTIONS_PROPERTY に適切な値を指定します。 CONNECT_OPTIONS_PROPERTY の値の詳細については、再接続オプションを参照してください。

クライアント・アプリケーションが常に接続と再接続を行う相手として、 同じ名前のキュー・マネージャー、同じキュー・マネージャー、または クライアント接続テーブルで同じ QMNAME によって定義された 任意のセットのキュー・マネージャー (詳細については、 CCDT 内のキュー・マネージャー・グループを参照) から選択できます。

.NET での Transport Layer Security (TLS) サポート

IBM MQ classes for .NET クライアント・アプリケーションは、Transport Layer Security (TLS) 暗号化をサポートします。 TLS プロトコルは、インターネットでの通信セキュリティーを提供し、クライアント/サーバー・アプリケーションが、機密性の保たれた、信頼できる方法で通信できるようにします。

関連概念

IBM MQ.NET 管理対象クライアントの TLS サポート

暗号セキュリティー・プロトコル: TLS

非管理対象 *.NET* クライアントのための *TLS* サポート

非管理対象 .NET クライアントの TLS サポートは、C MQI および IBM Global Security Kit (GSKit)に基づいています。 C MQI は TLS 操作を処理し、 GSKit は TLS セキュア・ソケット・プロトコルを実装します。

非管理対象 *.NET* クライアントの *TLS* の使用可能化

TLS がサポートされているのは、クライアント接続の場合のみです。 TLS を使用可能にするには、キュー・マネージャーとの通信時に使用する CipherSpec を指定する必要があります。これは、ターゲット・チャネル上の CipherSpec セットと一致する必要があります。

TLS を有効にするには、MQEnvironment の SSLCipherSpec 静的メンバー変数を使用して CipherSpec を指定します。 次の例では、SECURE.SVRCONN.CHANNEL という名前の SVRCONN チャネルに接続しています。このチャネルは、TLS_RSA_WITH_AES_128_CBC_SHA の CipherSpec で TLS を要求するように設定されています。

V 9.3.0 V 9.3.0

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

CipherSpec のリストについては、CipherSpec の指定を参照してください。

SSLCipherSpec プロパティーは、接続プロパティーのハッシュ・テーブルの MQC.SSL_CIPHER_SPEC_PROPERTY を使用して設定することもできます。

TLS を使用して正常に接続するには、キュー・マネージャーによって提示される証明書を認証できる認証局のルート証明書チェーンを使用して、クライアント鍵ストアをセットアップしておく必要があります。 同様に、SVRCONN チャネルの SSLClientAuth を MQSSL_CLIENT_AUTH_REQUIRED に設定してある場合は、キュー・マネージャーによって信頼されている識別個人証明書がクライアントの鍵ストアに含まれていなければなりません。

キュー・マネージャーの識別名の使用

キュー・マネージャーは、識別名 (DN) が含まれている TLS 証明書を使用して自身を識別します。

IBM MQ .NET クライアント・アプリケーションでこの DN を使用すると、確実に正しいキュー・マネージャーと通信することができます。 DN パターンを指定するときは、MQEnvironment の sslPeerName 変数を使用します。 例えば、以下のように設定すると、

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

キュー・マネージャーが QMGR で始まる共通名を持つ証明書を提示した場合にのみ、接続を成功させます。少なくとも 2 つの組織単位名。最初の組織単位名は IBM で、2 番目の組織単位名は WEBSPHERE でなければなりません。

SSLPeerName プロパティーは、接続プロパティーのハッシュ・テーブルの MQC.SSL_PEER_NAME_PROPERTY を使用して設定することもできます。 識別名、およびピア名の設定規則の詳細については、IBM MQ の保護を参照してください。

SSLPeerName が設定されている場合、それが有効なパターンに設定されており、キュー・マネージャーが一致する証明書を提示した場合のみ接続が成功します。

TLS の使用時のエラー処理

TLS を使用してキュー・マネージャーに接続した場合に、IBM MQ classes for .NET から発行される理由コードは次のとおりです。

MQRC_SSL_NOT_ALLOWED
SSLCipherSpec プロパティーが設定されましたが、バインディング接続が使用されました。 TLS をサポートしているのは、クライアント接続のみです。

MQRC_SSL_PEER_NAME_MISMATCH
SSLPeerName プロパティーで指定された DN パターンが、キュー・マネージャーによって提示された DN と一致しませんでした。

MQRC_SSL_PEER_NAME_ERROR
SSLPeerName プロパティーで指定された DN パターンが有効ではありませんでした。

V 9.3.0 V 9.3.0 **MQRC_KEY_REPOSITORY_ERROR**
鍵リポジトリーのロケーションが指定されていないか、無効であるか、またはアクセスできません。

管理対象 *.NET* クライアントのための *TLS* サポート

管理対象 .NET クライアントは、 Microsoft .NET Framework ライブラリーを使用して TLS セキュア・ソケット・プロトコルを実装します。 Microsoft System.Net.SecuritySslStream クラスは、接続された TCP ソケットを介したストリームとして機能し、そのソケット接続を介してデータを送受信します。

最低限必要な .NET Framework レベルは .NET Framework v3.5 です。 暗号アルゴリズム・サポートのレベルは、アプリケーションが使用している .NET Framework レベルに基づいています。

- .NET Framework レベル 3.5 および 4.0 に基づくアプリケーションの場合、使用可能なセキュア・ソケット・プロトコルは SSL 3.0 および TSL 1.0 です。
- .NET Framework レベル 4.5 に基づくアプリケーションの場合、使用可能なセキュア・ソケット・プロトコルは、SSL 3.0、TLS 1.1 、および TLS 1.2 です。

.NET Framework で Microsoft セキュリティー・サポートに定義されているように、より高い TLS プロトコル・サポートを必要とするアプリケーションを、より新しいバージョンのフレームワークに移動することが必要になる場合があります。

管理対象 .NET クライアントのための TLS サポートの主なフィーチャーは、以下のとおりです。

TLS プロトコルのサポート
.NET 管理対象クライアントのための TLS サポートは .NET SSLStream クラスを通して定義され、アプリケーションが使用する .NET Framework に依存します。 詳しくは、597 ページの『管理対象 .NET クライアントのための TLS プロトコル・サポート』を参照してください。

CipherSpec サポート
.NET 管理対象クライアントのための TLS 設定は、Microsoft .NET TLS ストリームの場合と同様です。詳しくは、597 ページの『管理対象 .NET クライアントの CipherSpec サポート』および 599 ページの『管理対象 .NET クライアントの CipherSpec マッピング』を参照してください。

鍵リポジトリー
クライアント・サイドの鍵リポジトリーは、Windows 鍵ストアです。 サーバー・サイドのリポジトリーは、CMS (Cryptographic Message Syntax) タイプのリポジトリーです。 詳しくは、601 ページの『管理対象 .NET クライアント用の鍵リポジトリー』を参照してください。

証明書
TLS 自己署名証明書を使用して、クライアントとキュー・マネージャーの間の相互認証を実装できます。 詳しくは、601 ページの『管理対象 .NET クライアント用の証明書の使用』を参照してください。

SSLPEERNAME
.NET では、アプリケーションはオプションの SSLPEERNAME 属性を使用して、識別名 (DN) パターンを指定できます。 詳しくは、602 ページの『SSLPEERNAME』を参照してください。

FIPS 準拠
FIPS をプログラマチックに有効にすることは、Microsoft.NET Security ライブラリーでサポートされていません。 FIPS の有効化は、Windows のグループ・ポリシー設定で制御されます。

NSA Suite B 準拠
IBM MQ は、RFC 6460 を実装しています。 NSA suite B に対応する Microsoft.NET 実装は 5430 です。これは .NET Framework 3.5 以降でサポートされています。

秘密鍵のリセットまたは再ネゴシエーション
SSLStream クラスは秘密鍵のリセットまたは再ネゴシエーションをサポートしませんが、他の IBM MQ クライアントとの整合性のために、 .NET 管理対象クライアントはアプリケーションが SSLKeyResetCount を設定することを許可します。 詳しくは、602 ページの『管理対象 .NET クライアントの秘密鍵のリセットまたは再ネゴシエーション』を参照してください。

失効チェック
SSLStream クラスは証明書の失効チェックをサポートします。 このチェックは、証明書チェーン・エンジンによって自動的に行われます。 詳しくは、603 ページの『失効チェック』を参照してください。

IBM MQ セキュリティー出口のサポート
SSLStream クラスは、 IBM MQ セキュリティー出口の限定的なサポートを提供します。 SSLPeerNamePtr(サブジェクト DN) および SSLRemCertIssNamePtr (発行者 DN) を取得するためにローカル証明書およびリモート証明書を照会することができます。これは、これが Microsoft.NET でサポートされているためです。 ただし、DNQ、UNSTRUCTUREDNAME、UNSTRUCTUREDADDRESS のような属性を取得することはサポートされていないため、出口を使用してそれらの値を取得することはできません。

暗号化ハードウェアのサポート
管理対象 .NET クライアントでの暗号化ハードウェアはサポートされていません。

管理対象 IBM MQ .NET および XMS .NET クライアントでの TLS1.3 のサポート

V 9.3.2

IBM MQ 9.3.2 以降、オペレーティング・システムが TLS1.3 をサポートする場合、 IBM MQ .NET および XMS .NET クライアントは TLS1.3 をサポートします。

管理対象 .NET クライアントは、 Microsoft .NET Framework ライブラリーを使用して TLS セキュア・ソケット・プロトコルを実装します。 Microsoft System.Net.SecuritySslStream クラスは、接続された TCP ソケットを介してストリームとして作動し、そのソケット接続を介してデータを送受信します。

Windows では、 .NET は SCHCHANNEL を使用し、 Linux .NET では SSL 通信に OpenSSL を使用します。

Windows Windows 上で実行される IBM MQ .NET クライアント・アプリケーションの場合

Microsoft は、 Windows 11 および Windows Server 2022 がデフォルトで TSL1.3 暗号をサポートすることを発表しました。

TLS_AES_128_GCM_SHA256 および TLS_AES_256_GCM_SHA384 暗号スイートは、両方のバージョンの Windows でデフォルトで有効になっています。

重要 :

- TLS_CHACHA20_POLY1305_SHA256 暗号スイートはデフォルトでは有効になっていませんが、サポートされています。
- TLS1.3 が有効になっている IBM MQ .NET クライアントの場合、キュー・マネージャーに正常に接続するには、 IBM Global Security Kit (GSKit) バージョン 8.0.55.29 がキュー・マネージャー・サイドで必要な最小バージョンです。

Linux Linux 上で実行される IBM MQ .NET クライアント・アプリケーションの場合

.NET は SSL 通信用に Linux 上の OpenSSL を使用するため、 TLS1.3 を使用するには、 OpenSSL v1.1.1 が最小要件です。

さらに、 .NET は Linux で OpenSSL を使用するため、 OpenSSL によってサポートされるすべての暗号は .NET でも機能する必要があります。

OpenSSL は、 TLS1.3: に対して以下の CipherSpecs をサポートします。

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

関連概念

599 ページの『管理対象 .NET クライアントの CipherSpec マッピング』
IBM MQ .NET インターフェースは、IBM MQ から Microsoft .NET へのマッピング・テーブルを維持します。このテーブルは、管理対象クライアントがキュー・マネージャーとのセキュア接続を確立するのに必要な TLS プロトコルのバージョンを決定するために使用されます。

管理対象 *.NET* クライアントのための *TLS* プロトコル・サポート
IBM MQ.NET TLS サポートは、.NET SSLStream クラスに基づいています。

注 : 管理対象 .NET クライアントのための TLS プロトコル・サポートは、アプリケーションが使用する .NET Framework レベルに依存します。 詳しくは、595 ページの『管理対象 .NET クライアントのための TLS サポート』 を参照してください。

Microsoft.NET SSLStream クラスが TLS を初期化してキュー・マネージャーとのハンドシェークを行うために設定する必要のある必須パラメーターの 1 つが **SSLProtocol** です。このパラメーターで TLS のバージョン番号を指定する必要があります。このバージョンは、以下のいずれかの値でなければなりません。

- SSL3.0
- TLS1.0
- TLS1.2

このパラメーターの値は、優先される CipherSpec が属するプロトコル・ファミリーと密接に結び付けられます。 SSLStream はサーバー (キュー・マネージャー) との TLS ハンドシェークを開始するときに、**SSLProtocol** で指定された TLS のバージョンを使用して、ネゴシエーションに使用する CipherSpec のリストを識別します。

IBM MQ.NET では、アプリケーションがこの値を設定するために使用できるプロパティーはありません。代わりに、IBM MQ はマッピング・テーブルを使用して、設定された CipherSpec をプロトコル・ファミリーに内部的にマップし、使用する SSLProtocol のバージョンを識別します。 このテーブルは、サポートされる CipherSpec それぞれの Microsoft.NET と IBM MQ の間のマッピングと、それらが属するプロトコル・バージョンを示します。 詳細については、599 ページの『管理対象 .NET クライアントの CipherSpec マッピング』 を参照してください。

管理対象 *.NET* クライアントの *CipherSpec* サポート
アプリケーションの CipherSpec 設定は、サーバーとのハンドシェーク中に使用されます。

IBM MQ クライアントにより、キュー・マネージャーとのハンドシェーク中に使用される CipherSpec 値を設定することができます。 IBM MQ クライアントは、セキュア接続を確立するために、有効な CipherSpec (できれば Windows グループ・ポリシーで指定された CipherSpec) を設定する必要があります。 このフィールドをブランクのままにすると、ソケットでセキュリティーなしの非暗号化テキスト・チャネルを使用することを示します。

IBM MQ.NET 管理対象クライアントでは、TLS 設定は Microsoft.NET SSLStream クラス用です。 SSLStream では、CipherSpec (または CipherSpec の優先リスト) は Windows のグループ・ポリシーでのみ設定でき、これはコンピューター全体のための設定です。 そして、SSLStream はサーバーとのハンドシェーク中に、指定された CipherSpec または優先リストを使用します。 他の IBM MQ クライアントの場合、CipherSpec プロパティーはアプリケーションの IBM MQ チャネル定義で設定でき、同じ設定が TLS ネゴシエーションで使用されます。 この制限の結果として、IBM MQ チャネル構成で指定されたものに関係なく、TLS ハンドシェークはサポートされる CipherSpec をネゴシエーションすることがあります。 このため、キュー・マネージャーでエラー AMQ9631 が発生してしまうことがよくあります。 このエラーを防ぐには、Windows グループ・ポリシーの TLS 構成としてアプリケーションで設定したのと同じ CipherSpec を設定してください。

新しい IBM MQ.NET TLS クライアント・コードは、正しいプロトコル・バージョンでネゴシエーションが行われていることだけをチェックします。 この TLS プロトコル・バージョンは、アプリケーションが設定し、サーバー (キュー・マネージャー) との TLS ハンドシェークで使用される CipherSpec から派生します。 このため、IBM MQ.NET 管理対象クライアント・アプリケーションで CipherSpec を設定することが設計上必要です。 IBM MQ クライアントによって設定された CipherSpec が SSL 3.0、TLS 1.0、および TLS 1.2 プロトコル以外のものである場合、IBM MQ 管理対象 .NET クライアントは、デフォルトで SSL 3.0 または TLS 1.0 のいずれかのプロトコルの暗号でネゴシエーションし、エラーを報告しません。

注 : アプリケーションから提供された CipherSpec 値が、IBM MQ が認識する CipherSpec ではない場合、IBM MQ 管理対象の .NET クライアントは、これを無視し、Windows システムのグループ・ポリシーに基づいて接続をネゴシエーションします。

CipherSpec の設定

CipherSpec を設定するには、以下の 3 つの方法があります。

MQEnvironment .NET クラス
次の例は MQEnvironment クラスを使って CipherSpec を設定する方法を示します。

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

TLS CipherSpec プロパティー
次の例は、hashtable パラメーターを MQQueueManager コンストラクターに追加することにより CipherSpec を設定する方法を示します。

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows グループ・ポリシー
Windows グループ・ポリシー管理コンソールを使用して暗号スイート・リストが構成されている場合、SVRCONN チャネル定義で、一致する CipherSpec を指定する必要があります。 一致する CipherSpec は、「ANY_TLS12_OR_HIGHER」などの総称値、または番号付きリストからネゴシエーションされる最高の暗号スイートにマップされる特定の値のいずれかになります。 CipherSpec の総称値を使用する

と、クライアント・リストの順序が変更された場合に SVRCONN CipherSpec 構成を変更する必要がないため、.NET クライアントではこの値を使用することをお勧めします。

CCDT の使用法

IBM MQ.NET は、ローカル・コンピューター上にある CCDT (クライアント・チャネル定義テーブル) (.TAB ファイル) だけをサポートします。 CipherSpec 値が設定されている既存の CCDT ファイルは、IBM MQ.NET 接続で使用できます。 ただし、クライアント接続チャネルで設定されている CipherSpec 値が TLS プロトコル・バージョンを決定し、この値は Windows グループ・ポリシーで設定された CipherSpec とも一致しなければなりません。

関連概念

585 ページの『IBM MQ 環境のセットアップ』
クライアント接続を使用してキュー・マネージャーに接続するには、まず、IBM MQ 環境をセットアップする必要があります。

595 ページの『管理対象 .NET クライアントのための TLS サポート』
管理対象 .NET クライアントは、 Microsoft .NET Framework ライブラリーを使用して TLS セキュア・ソケット・プロトコルを実装します。 Microsoft System.Net.SecuritySslStream クラスは、接続された TCP ソケットを介したストリームとして機能し、そのソケット接続を介してデータを送受信します。

関連タスク

CipherSpec の指定

関連資料

MQEnvironment .NET クラス

管理対象 *.NET* クライアントの *CipherSpec* マッピング
IBM MQ .NET インターフェースは、IBM MQ から Microsoft .NET へのマッピング・テーブルを維持します。 このテーブルは、管理対象クライアントがキュー・マネージャーとのセキュア接続を確立するのに必要な TLS プロトコルのバージョンを決定するために使用されます。

CipherSpec が SVRCONN チャネルで指定される場合、TLS ハンドシェークの完了後に、キュー・マネージャーはその CipherSpec を、クライアント・アプリケーションが使用するネゴシエーション済みの CipherSpec と一致させようとします。 キュー・マネージャーが一致する CipherSpec を見つけることができないと、通信はエラー AMQ9631 で失敗します。

IBM MQ .NET インターフェースは、IBM MQ を Microsoft.NET CipherSpec マッピング・テーブルへ維持します。 このテーブルは、キュー・マネージャーとの保護されたソケット接続を確立するためにクライアントが使用する TLS プロトコル・バージョンを決定するために使用されます。 SSLCipherSpec の値に基づき、どのバージョンの Microsoft.NET Framework を使用するかに応じて、SSLProtocol バージョンは TLS 1.0、TLS 1.2 のいずれかになります。

SSLCipherSpec に誤った値を指定すると SSL 3.0 または TLS 1.0 プロトコルが使用される可能性があるため、正しい値を指定するようにしてください。

表 *78. IBM MQ* と *Microsoft.NET* のマッピング・テーブル

IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**TLS バージョン**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2

表 78. *IBM MQ と Microsoft.NET のマッピング・テーブル (続き)*

IBM MQ CipherSpec	Microsoft.NET CipherSpec	TLS バージョン
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.3.2 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3
V 9.3.2 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.3.2 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

注 :

1. Deprecated この CipherSpec の TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

関連概念

595 ページの『管理対象 .NET クライアントのための TLS サポート』
管理対象 .NET クライアントは、 Microsoft .NET Framework ライブラリーを使用して TLS セキュア・ソケット・プロトコルを実装します。 Microsoft System.Net.SecuritySslStream クラスは、接続された TCP ソケットを介したストリームとして機能し、そのソケット接続を介してデータを送受信します。

管理対象 *.NET* クライアント用の鍵リポジトリー
管理対象 .NET クライアントによって使用される鍵リポジトリーは、Windows 鍵ストアです。 クライアント・アプリケーションが、TLS ハンドシェーク中に ID および信頼を手に入れるために使用できるように、証明書と秘密鍵は、ユーザー鍵ストアまたはシステム鍵ストアのいずれかで使用できるようになっている必要があります。

クライアント・サイド

アプリケーション内で、以下の値のいずれかを鍵リポジトリー用に設定することができます。

- "*USER": IBM MQ .NET は、現在のユーザーの証明書ストアにアクセスし、クライアント証明を取得します。
- "*SYSTEM": IBM MQ .NET はローカル・コンピューター・アカウントにアクセスし、証明書を取得します。

クライアントの証明書は、ユーザーまたはコンピューター・アカウントのマイ証明書ストアに保管されている必要があります。 すべてのサーバー (CA) 証明書は、証明書ストアのルート・ディレクトリーに保管しなければなりません。

注 : 1 つのファイルに複数の証明書を保管するには、以下の形式を使用します。

- 個人情報の交換 - PKCS #12 (.PFX, .P12)
- 暗号メッセージ構文標準 - PKCS #7 証明書 (.P7B)
- Microsoft シリアライズされた証明書ストア (.SST)

管理対象 *.NET* クライアント用の証明書の使用
クライアント証明書の場合、IBM MQ 管理対象 .NET クライアントは、Windows 鍵ストアにアクセスし、証明書ラベルによって一致するか、ストリングによって一致したクライアントの証明書をすべてロードします。

使用する証明書を選ぶときに、IBM MQ 管理対象 .NET クライアントは常に SSLStream TLS ハンドシェーク用の最初に一致する証明書を使用します。

証明書ラベルによる証明書のマッチング

証明書ラベルを設定すると、IBM MQ 管理対象 .NET クライアントは、指定されたラベル名を使って Windows 証明書ストアを検索し、クライアント証明書を識別します。 一致する証明書がすべてロードされ、リストの最初の証明書が使用されます。 証明書ラベルを設定するための 2 つのオプションがあります。

- 証明書ラベルは、MQEnvironment.CertificateLabel にアクセスして MQEnvironment クラスに設定できます。
- 証明書ラベルは、ハッシュ・テーブル・プロパティーでも設定できます。 これは次の例に示すように、MQQueueManager コンストラクターの入力パラメーターとして指定できます。

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

名前 ("CertificateLabel") とその値には大/小文字の区別があります。

ストリングによる証明書のマッチング

証明書ラベルが設定されていない場合、ストリング "ibmwebspheremq" と現在のログオン・ユーザー (小文字) に一致する証明書が検索されて使用されます。

関連タスク

キュー・マネージャーへのクライアントのセキュア接続

関連資料

MQEnvironment .NET クラス

SSLPEERNAME

SSLPEERNAME 属性は、ピア・キュー・マネージャーからの証明書の識別名 (DN) を確認するために使用します。

IBM MQ.NET では、次の例に示すようにアプリケーションは SSLPEERNAME を使用して識別名パターンを指定することもできます。

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

他の IBM MQ クライアントの場合、SSLPEERNAME はオプション・パラメーターです。

SSLPEERNAME 値が設定されないと、IBM MQ.NET 管理対象クライアントはリモート (サーバー) 証明書の妥当性検査をまったく行わず、管理対象クライアントはリモート (/サーバー) 証明書をそのまま受け入れます。

SSLPEERNAME を設定する方法は、どの IBM MQ スタック・オファリングを使用しているかによって変わります。

IBM MQ classes for .NET

以下の 3 つのオプションがあります。

1. MQEnvironment クラスの MQEnvironment.SSLPeerName を設定します。
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. キュー・マネージャー・コンストラクター `MQQueueManager (String queueManagerName, Hashtable properties)`を使用します。 オプション 2 については、SSLPEERNAME を `Hashtable properties` で指定します。

XMS .NET

接続ファクトリーで SSL ピア名を設定します。

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

SslPeerName をセミコロン区切りフィールドとして URI に組み込みます。

関連資料

MQEnvironment .NET クラス

管理対象 *.NET* クライアントの秘密鍵のリセットまたは再ネゴシエーション

SSLStream クラスは、秘密鍵のリセットや再ネゴシエーションをサポートしません。 ただし、他の IBM MQ クライアントとの整合性を保つために、 IBM MQ 管理対象 .NET クライアントでは、アプリケーションで **SSLKeyResetCount** を設定することができます。

限度に達すると、IBM MQ.NET はキュー・マネージャーから切断し、アプリケーションには理由コード MQRC_CONNECTION_BROKEN 付きの例外でそのことが通知されます。 アプリケーションは例外を処理して接続を再確立するか、IBM MQ.NET の MQCNO_RECONNECT オプションを有効にして自動的にキュー・マネージャーに再接続するかを選択できます。

クライアントの自動再接続機構を有効にするということは、鍵リセット数が限度に達した場合、既存の接続はすべて停止され、IBM MQ.NET クライアントがすべての接続を新たに作成し直すということを意味します。 クライアントの自動再接続について詳しくは、クライアントの自動再接続を参照してください。

関連概念
SSL および TLS 秘密鍵のリセット

失効チェック
SSLStream クラスは証明書の失効チェックをサポートします。

失効チェックは、証明書チェーン・エンジンによって自動的に実行されます。 これは、OCSP (Online Certificate Status Protocol) と CRL (Certificate Revocation List) の両方に当てはまります。 SSLStream クラスは、証明書に指定されたサーバーのみを使用する証明書の失効を使用します。 つまり、サーバーは証明書そのものによって決まります。 HTTP CDP 拡張機能や OCSP HTTP が HTTP プロキシー・サーバー経由でプロキシーに要求することが可能です。

失効チェックを設定する方法は、どの IBM MQ スタック・オファリングを使用しているかによって異なります。

IBM MQ.NET
失効チェックは、MQEnvironment.cs クラス・ファイルの **MQEnvironment.SSLCertRevocationCheck** プロパティーにアクセスすることにより設定できます。

XMS .NET
失効チェックは、以下の例に示すように接続ファクトリー・プロパティーのコンテキストで設定できます。

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
失効チェックは、以下の命名規則を使用して URI で設定できます。

```
"SslCertRevocationCheck=true"
```

管理対象 *IBM MQ .NET* 用の *TLS* の構成
管理対象 IBM MQ .NET 用の TLS の構成には、署名者証明書を作成してから、サーバー・サイド、クライアント・サイド、アプリケーション・プログラムを構成することが含まれます。

このタスクについて

TLS を構成するには、まず適切な署名者証明書を作成しなければなりません。 署名者証明書は、自己署名したものでも、認証局によって提供されたものでも構いません。 自己署名証明書は開発システム、テスト・システム、実動前システムで使用できますが、実動システムでは使用しないでください。 実動システムの場合は、信頼できる外部の認証局 (CA) から入手した証明書を使用してください。

手順

1. 署名者証明書を作成します。
   a) 自己署名証明書を作成するには、IBM MQ に同梱の以下のツールを使用します。
      strmqikm GUI、または **runmqckm** あるいは **runmqakm** をコマンド行から使用します。 これらのツールの使用について詳しくは、**runmqckm**、**runmqakm**、**strmqikm** を使用したデジタル証明書の管理を参照してください。
   b) キュー・マネージャーとクライアントのための証明書を認証局 (CA) から入手するには、認証局からの個人用証明書の取得にある指示に従ってください。
2. サーバー・サイドを構成します。

a) キュー・マネージャーへのクライアントのセキュア接続の説明に従って、 IBM Global Security Kit (GSKit)を使用してキュー・マネージャーで TLS を構成します。

b) SVRCONN チャネルの TLS 属性を設定します。

- **SSLCAUTH** を "REQUIRED/OPTIONAL" に設定します。
- **SSLCIPH** を適切な CipherSpec に設定します。

詳しくは、594 ページの『非管理対象 .NET クライアントの TLS の使用可能化』を参照してください。

3. クライアント・サイドを構成します。

a) Windows 証明書ストア (ユーザー/コンピューター・アカウントの下) にクライアント証明書をインポートします。

IBM MQ .NET は、Windows 証明書ストアからクライアント証明書にアクセスするため、IBM MQ へのセキュア・ソケット接続を確立するには、証明書を Windows 証明書ストアにインポートする必要があります。 Windows 鍵ストアにアクセスしてクライアント・サイド証明書をインポートする方法について詳しくは、証明書と秘密キーをインポートまたはエクスポートするを参照してください。

b) キュー・マネージャーへのクライアントのセキュア接続で説明されているように、CertificateLabel を指定します。

c) 必要なら、Windows グループ・ポリシーを編集して CipherSpec を設定してから、Windows グループ・ポリシーの更新が有効になるようにコンピューターを再始動します。

4. アプリケーション・プログラムを構成します。

a) MQEnvironment または SSLCipherSpec 値を、接続が保護された接続であることを示すように設定します。

指定した値で、使用されているプロトコル (TLS) を識別できます。 設定される CipherSpec は、サポートされる SSLProtocol バージョンのいずれかの CipherSpec でなければなりません。できれば、Windows グループ・ポリシーで指定された CipherSpec と同じにすることをお勧めします。 (サポートされる SSLProtocol バージョンは、使用される .NET Framework によって異なります。 どのバージョンの Microsoft .NET Framework を使用するかに応じて、SSLProtocol バージョンは TLS 1.0、TLS 1.2 のいずれかになります。)

注 : アプリケーションから提供された CipherSpec 値が、IBM MQ が認識する CipherSpec ではない場合、IBM MQ 管理対象の .NET クライアントは、これを無視し、Windows システムのグループ・ポリシーに基づいて接続をネゴシエーションします。

b) SSLKeyRepository プロパティーを "*SYSTEM" または "*USER"に設定します。

c) オプション: SSLPEERNAME をサーバー証明書の識別名 (DN) に設定します。

d) キュー・マネージャーへのクライアントのセキュア接続で説明されているように、CertificateLabel を指定します。

e) KeyResetCount や CertificationRevocationCheck などさらに必要なオプション・パラメーターを設定し、FIPS を有効にします。

TLS プロトコルと TLS 鍵リポジトリーの設定方法の例

ベース .NET の場合、次の例に示すように、TLS プロトコルと TLS 鍵リポジトリーは MQEnvironment クラスを使って設定できます。

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

あるいは、次の例に示すように、hashtable を MQQueueManager コンストラクターの一部として指定することにより、TLS プロトコルと TLS 鍵リポジトリーを設定できます。

```
Hashtable properties = new Hashtable();
```

```
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

次のタスク

IBM MQ .NET 管理対象 TLS アプリケーションの開発を始めることについて詳しくは、605 ページの『簡単なアプリケーションの作成』を参照してください。

関連資料

MQEnvironment .NET クラス

KeyResetCount (MQLONG)

AIX, Linux, and Windows での連邦情報処理標準 (FIPS)

簡単なアプリケーションの作成
接続ファクトリーのための SSL プロパティーの設定、キュー・マネージャー・インスタンス、接続、セッション、および宛先の作成、テスト・メッセージの送信の例を含め、簡単な IBM MQ 管理対象 .NET TLS アプリケーションを作成するためのヒントです。

始める前に

最初に、603 ページの『管理対象 IBM MQ .NET 用の TLS の構成』で説明されているように、管理対象 IBM MQ.NET 用に TLS を構成する必要があります。

ベース .NET のアプリケーション・プログラム構成の場合、SSL プロパティーは、MQEnvironment クラスを使用するか、MQQueueManager コンストラクターの一部に hashtable を指定することにより設定します。

XMS .NET のアプリケーション・プログラム構成の場合、SSL プロパティーは接続ファクトリーのプロパティー・コンテキストで設定します。

手順

1. 接続ファクトリーの SSL プロパティーは、次の例で示すように設定します。

   IBM MQ.NET の例

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   properties.Add("CertificateLabel", "ibmwebspheremq");
   MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
   ```

   XMS .NET の例

   ```
   cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
   cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
   cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
   cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
   cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
   ```

2. キュー・マネージャー・インスタンス、接続、セッション、宛先を次の例で示すように作成します。

   MQ .NET の例

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   ```

```
queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
MQC.MQOO_FAIL_IF_QUIESCING);
Console.WriteLine("done");
```

XMS .NET の例

```
connectionWMQ = cf.CreateConnection();
// Create session
sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

// Create destination
destination = sessionWMQ.CreateQueue(destinationName);

// Create producer
producer = sessionWMQ.CreateProducer(destination);
```

3. 次の例で示すように、メッセージを送信します。

MQ .NET の例

```
// creating a message object
message = new MQMessage();
message.WriteString(messageString);

// putting messages continuously
for (int i = 1; i <= numberOfMsgs; i++)
{
Console.Write("Message " + i + " <" + messageString + ">.. ");
queue.Put(message);
Console.WriteLine("put");
}
```

XMS .NET の例

```
textMessage = sessionWMQ.CreateTextMessage();
textMessage.Text = simpleMessage;
producer.Send(textMessage);
```

4. TLS 接続を検査します。

   チャネル状況をチェックして、TLS 接続が確立されて正しく機能していることを検査します。

SSLStream のトレースの構成

SSLStream クラスに関連するトレース・イベントとメッセージをキャプチャーするには、アプリケーションのアプリケーション構成ファイルにシステム診断のための構成セクションを追加しなければなりません。

このタスクについて

注 :

このタスクは、 IBM MQ classes for .NET Framework にのみ適用されます。 アプリケーション構成ファイルは、 IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) ではサポートされていません。

アプリケーション構成ファイルにシステム診断のための構成セクションを追加しないと、IBM MQ 管理対象 .NET クライアントは、TLS および SSLStream クラスに関連したイベント、トレース、デバッグ・ポイントをキャプチャーしません。

注 : strmqtrc を使用して IBM MQ トレースを開始しても、必要なすべての TLS トレースがキャプチャーされるわけではありません。

手順

1. アプリケーション・プロジェクト用のアプリケーション構成 (App.Config) ファイルを作成します。
2. 次の例に示すように、system.diagnostics 構成セクションを追加します。

```
<system.diagnostics>
          <sources>
                <source name="System.Net" tracemode="includehex">
                      <listeners>
                            <add name="ExternalSourceTrace"/>
                      </listeners>
                </source>
                <source name="System.Net.Sockets">
                      <listeners>
                            <add name="ExternalSourceTrace"/>
                      </listeners>
                </source>
                <source name="System.Net.Cache">
                      <listeners>
                            <add name="ExternalSourceTrace"/>
                      </listeners>
                </source>
                <source name="System.Net.Security">
                      <listeners>
                            <add name="ExternalSourceTrace"/>
                      </listeners>
                </source>
                <source name="System.Security">
                      <listeners>
                            <add name="ExternalSourceTrace"/>
                      </listeners>
                </source>
          </sources>
          <switches>
                <add name="System.Net" value="Verbose"/>
                <add name="System.Net.Sockets" value="Verbose"/>
                <add name="System.Net.Cache" value="Verbose"/>
                <add name="System.Security" value="Verbose"/>
                <add name="System.Net.Security" value="Verbose"/>
          </switches>

          <sharedListeners>
                <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
          </sharedListeners>
        <trace autoflush="true"/>
    </system.diagnostics>
```

重要 : add name エントリーの Version フィールドは、使用している .net amqmdnet.dll ファイルのどのバージョンでも指定する必要があります。

関連タスク

アプリケーション構成ファイルを使用した IBM MQ classes for .NET Framework クライアントのトレース

管理対象 *.NET* で *TLS* を実装するためのサンプル・アプリケーション

WCF 用の IBM MQ classes for .NET、 XMS .NET 、および IBM MQ カスタム・チャネルで管理対象 .NET の TLS の実装を示すために、サンプル・アプリケーションが提供されています。

次の表にサンプル・アプリケーションの場所を示します。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

表 *79.* 管理対象 *.NET* で *TLS* を実装するためのサンプル・アプリケーションの場所

IBM MQ.NET スタック・オファリング	**サンプルの場所**
基本 .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs

表 79. 管理対象 *.NET* で *TLS* を実装するためのサンプル・アプリケーションの場所 (続き)	
IBM MQ.NET スタック・オファリング	**サンプルの場所**
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
WCF 用の IBM MQ カスタム・チャネル	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows .NET モニターの使用

.NET モニターは、IBM MQ トリガー・モニターに類似したアプリケーションです。

重要 : 重要な情報については、 Windows 上のプライマリー・インストールでのみ使用できる機能 を参照してください。

モニター対象のキューでメッセージが受信されるとインスタンス化され、その後そのメッセージを処理する、.NET コンポーネントを作成することができます。 .NET モニターは、**runmqdnm** コマンドで開始され、**endmqdnm** コマンドで停止されます。 これらのコマンドについて詳しくは、runmqdnm および endmqdnm を参照してください。

.NET モニターを使用するには、amqmdnm.dll で定義される IMQObjectTrigger インターフェースを実装するコンポーネントを作成します。

コンポーネントはトランザクションであってもトランザクションでなくてもかまいません。 トランザクション・コンポーネントは、System.EnterpriseServices.ServicedComponent から継承され、RequiresTransaction または SupportsTransaction として登録される必要があります。 .NET モニターがトランザクションを既に開始しているため、RequiresNew として登録することはできません。

コンポーネントは、**runmqdnm** から MQQueueManager、MQQueue、および MQMessage オブジェクトを受け取ります。 またこのコンポーネントは、runmqdnm の開始時に *-u* コマンド行オプションを使用してユーザー・パラメーター・ストリングが指定された場合、そのストリングも受け取ります。 コンポーネントは、モニター対象のキューに到着したメッセージの内容を MQMessage オブジェクトで受け取ることに注意してください。 コンポーネントは、キュー・マネージャーに接続して、 キューをオープンしたり、メッセージそのものを取得する必要はありません。 コンポーネントは必要に応じてメッセージを処理し、.NET モニターに制御を返す必要があります。

コンポーネントがトランザクション・コンポーネントとして作成されている場合、そのコンポーネントは、System.EnterpriseServices.ServicedComponent で提供されている機能を使用してトランザクションをコミットするのかロールバックするのかを登録します。

コンポーネントがメッセージだけでなく MQQueueManager オブジェクトと MQQueue オブジェクトを受け取ると、そのメッセージに対する完全なコンテキスト情報が得られるため、例えば IBM MQ に個別に接続することなく、同じキュー・マネージャー上の別のキューをオープンすることができます。

Windows コード例

このトピックには、.NET モニターからメッセージを取得して出力するコンポーネントのコード例が 2 つあります。 1 つはトランザクション処理を使用し、もう 1 つは非トランザクション処理を使用します。 3 番目の例は、上の 2 つの例に適用できる共通ユーティリティー・ルーチンを示しています。 すべてのコード例は C# で作成されています。

例 1: トランザクション処理

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
```

```
/* (C) Copyright IBM Corp. 2005, 2024.                              */
/********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

例 2: 非トランザクション処理

```
/********************************************************************/
/* Licensed materials, property of IBM                              */
/* 63H9336                                                          */
/* (C) Copyright IBM Corp. 2005, 2024.                              */
/********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m QMNAME -q QNAME -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
```

```
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

例 3: 共通ルーチン

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ---------------------------------------------------------------- */
    /* Default prefix string of the namespace.                           */
    /* ---------------------------------------------------------------- */
    private string prefixText = "dnmsamp";

    /* ---------------------------------------------------------------- */
    /* Constructor that takes the replacement prefix string to use.     */
    /* ---------------------------------------------------------------- */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ---------------------------------------------------------------- */
    /* Display an arbitrary string to the console.                      */
    /* ---------------------------------------------------------------- */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ---------------------------------------------------------------- */
    /* Display the content of the message passed to the console.        */
    /* ---------------------------------------------------------------- */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }
```

```
    /* ---------------------------------------------------------------- */
    /* Convert the byte array into a hex string.                        */
    /* ---------------------------------------------------------------- */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);

        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }
  }
}
```

IBM MQ .NET プログラムのコンパイル

さまざまな言語で作成した .NET アプリケーションをコンパイルするためのコマンドの実例を示します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

IBM MQ classes for .NET を使用して C# アプリケーションをビルドするには、次のコマンドを 使用します。

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

IBM MQ classes for .NET を使用して Visual Basic アプリケーションをビルドするには、次のコマンドを 使用します。

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

IBM MQ classes for .NET を使用して Managed C++ アプリケーションをビルドするには、次のコマンドを使用します。

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

他の言語については、その言語のベンダーによって提供されている資料を参照してください。

スタンドアロン IBM MQ .NET クライアントの使用

IBM MQ .NET クライアントを使用すると、アプリケーションを実行するために実動システムで IBM MQ クライアントのフルインストール済み環境を使用する必要なく、 IBM MQ .NET アセンブリーをパッケージ化してデプロイすることができます。

始める前に

V 9.3.1 IBM MQ 9.3.1 以降、デフォルトの場所にインストールされる `amqmdnetstd.dll` クライアント・ライブラリーは、.NET 6 に基づいています。 .NET Standard ベースの `amqmdnetstd.dll` クライアント・ライブラリーは、 IBM MQ クライアント・インストール・パッケージの新しい場所に移動され、以下の場所で使用できるようになりました。

- Windows で Windows: *MQ_INSTALLATION_PATH*\bin\netstandard2.0
- Linux で Linux: *MQ_INSTALLATION_PATH*\lib64\netstandard2.0

LTS IBM MQ 9.3.0 Long Term Support の場合、 amqmdnetstd.dll ライブラリーは以下の場所で使用可能です。

- Windows Windows の場合: *MQ_INSTALLATION_PATH*\bin。 サンプル・アプリケーションは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にインストールされています。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*/lib64 path。 .NET サンプルは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にあります。

LTS Stabilized amqmdnet.dll ライブラリーは引き続き提供されていますが、このライブラリーは安定化しています。 つまり、新機能は導入されません。 最新の機能を使用する場合は、amqmdnetstd.dll ライブラリーに移行する必要があります。 ただし、 IBM MQ 9.1 Long Term Support または Continuous Delivery リリースでは、引き続き amqmdnet.dll ライブラリーを使用できます。

このタスクについて

フル IBM MQ クライアントがインストールされているマシン上で IBM MQ .NET アプリケーションをビルドし、後で IBM MQ .NET アセンブリー、つまり amqmdnetstd.dll をアプリケーションとともにパッケージ化し、実動システムにデプロイすることができます。

ビルドおよびデプロイするアプリケーションは、従来の .NET アプリケーション、サービス、または Microsoft Azure Web/Worker アプリケーションにすることができます。

そのようなデプロイメントの場合、IBM MQ .NET クライアントは、キュー・マネージャーに対する管理対象モードの接続のみをサポートします。 サーバー・バインディング接続および非管理対象クライアント・モード接続は使用できません。 これら 2 つのモードでは、フル IBM MQ クライアント・インストールが必要だからです。 これらの他の 2 つのモードの使用を試みると、アプリケーション例外になります。

手順

アプリケーションでの IBM MQ .NET クライアント・アセンブリーの参照

- アプリケーション内の amqmdnetstd.dll アセンブリーを、以前のリリースのときと同じ方法で参照します。

  amqmdnetstd.dll アセンブリーの **CopyLocal** プロパティーを True に設定して、amqmdnetstd.dll アセンブリーがアプリケーションの bin ディレクトリーにコピーされるようにします。 このプロパティーを設定することで、アプリケーション・パッケージ化ツールが、実動システムおよび Microsoft Azure PaaS クラウド環境でのデプロイメントに必要なバイナリー・ファイルをパッケージすることもできます。

グローバル・トランザクション・サポートの追加

- アプリケーションがモニター・アプリケーション WMQDotnetXAMonitor を、アプリケーション自体とともにマシン上にデプロイしていることを確認してください。

  アプリケーションが IBM MQ .NET 管理対象グローバル・トランザクション機能を使用する場合にも、アプリケーション自体とともにマシン上に WMQDotnetXAMonitor をデプロイする必要があります。 このユーティリティーは、未確定トランザクションのリカバリーに必要です。

トレースの開始と停止

- IBM MQ classes for .NET Framework の場合のみ、アプリケーション構成ファイルと IBM MQ 固有のトレース構成ファイルを使用してトレースを開始および停止するには、アプリケーション構成ファイルを使用した IBM MQ classes for .NET Framework クライアントのトレースを参照してください。

  アプリケーション構成ファイルと IBM MQ 固有のトレース構成ファイルを使用する必要があります。 フル IBM MQ クライアント・インストールがないため、トレースの開始と停止に使用される標準ツール **strmqtrc** と **endmqtrc** を使用できないからです。

  注:

  – このトレース生成方法は、.NET 再配布可能管理対象クライアントおよびスタンドアロン .NET クライアントに適用されます。 .NET アプリケーション・ランタイム- Windows のみを参照してください。

- アプリケーション構成ファイルは、 IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) ではサポートされません。 IBM MQ classes for .NET (.NET Standard および .NET 6 ライブラリー) のトレースを有効にするには、 **MQDOTNET_TRACE_ON** 環境変数を使用します。 環境変数を使用した IBM MQ .NET アプリケーションのトレースを参照してください。

- V 9.3.3

  mqclient.ini ファイルを使用し、Trace スタンザの適切なプロパティーを設定して、トレースを開始および停止します。

  mqclient.ini を使用した IBM MQ .NET アプリケーションのトレースを参照してください。

  IBM MQ 9.3.3 以降、 mqclient.ini ファイルを使用し、Trace スタンザの適切なプロパティーを設定することにより、トレースを構成できます。 mqclient.ini ファイルを使用して、トレースを動的に使用可能または使用不可にすることもできます。

アプリケーション構成ファイルでのバインディング・リダイレクトの有効化

- IBM MQ .NET アセンブリーのコンパイル時バインディング参照を、後のバージョンのアセンブリーに対して有効にするには、<dependentAssembly> プロパティーをアプリケーション構成ファイルに追加します。

  以下の app.config ファイルのサンプルのスニペットでは、IBM MQ 8.0.0 Fix Pack 2 (8.0.0.2) バージョンの IBM MQ .NET アセンブリーを使用してコンパイルされ、後にフィックスパック IBM MQ 8.0.0 Fix Pack 3 が適用された (それによって IBM MQ.NET アセンブリーが更新された) アプリケーションを、8.0.0.3 にリダイレクトしています。

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

関連概念

554 ページの『IBM MQ classes for .NET のインストール』
IBM MQ classes for .NET(サンプルを含む) は、 IBM MQ とともに Windows および Linux にインストールされます。

再配布可能クライアント

.NET アプリケーション・ランタイム - Windows のみ

関連タスク

575 ページの『WMQDotnetXAMonitor アプリケーションの使用』
IBM MQ .NET クライアントは、不完全な分散トランザクションをリカバリーするために使用できる XA モニター・アプリケーション WmqDotnetXAMonitor を提供します。 WmqDotnetXAMonitor アプリケーションは、トランザクションが未確定であるキュー・マネージャーへの接続を確立し、設定したパラメーターに基づいてトランザクションを解決します。

IBM MQ .NET アプリケーションのトレース

V 9.3.0 OutboundSNI プロパティー

アプリケーションで **OutboundSNI** プロパティーを設定するには、プロパティーまたは環境変数のいずれかを使用します。

IBM MQ 9.3.0 から、MQQueueManager クラスを使用してキュー・マネージャーに接続するときにハッシュ・テーブルを使用して、アプリケーション内で MQC.OUTBOUND_SNI_PROPERTY を設定することができます。

MQC.OUTBOUND_SNI_PROPERTY には、以下の値が使用されます。

- MQC.OUTBOUND_SNI_CHANNEL。これは、「CHANNEL」にマップされます。
- MQC.OUTBOUND_SNI_HOSTNAME。これは「HOSTNAME」にマップされます。
- MQC.OUTBOUND_SNI_ASTERISK。これは「*」にマップされます。

さらに、MQOUTBOUND_SNI 環境変数を使用して、**OutboundSNI** プロパティーを設定することもできます。これは以下の値を使用します。

- CHANNEL
- HOSTNAME
- *

また、他の mqclient.ini プロパティーと同様に、 App.config ファイルに **OutboundSNI** 値を設定します。

注: 特定の値が設定されていない場合、プロパティーはデフォルトで MQC.OUTBOUND_SNI_CHANNEL に設定されます。

管理対象ノードで **OutboundSNI** プロパティーを設定する際の優先順位は、以下のとおりです。

1. アプリケーション・レベル・プロパティー
2. 環境変数

非管理対象ノードの **OutboundSNI** プロパティーでは、 mqclient.ini のみがサポートされます。

App.config ファイルに設定されたプロパティーは、 .NET Framework アプリケーションにのみ適用されます。

アプリケーション・レベルまたは App.config ファイル内で無効な値を指定すると、戻りコード MQRC_OUTBOUND_SNI_NOT_VALID が発行されます。

無効な環境変数を設定した場合、または mqclient.ini ファイル内で無効な値を指定した場合は、デフォルト値の CHANNEL が使用されます。

OutboundSNI と複数の証明書

IBM MQ は、SNI ヘッダーを使用して複数の証明書機能を提供します。 アプリケーションが、CERTLABL フィールドを介して別の証明書を使用するように構成されている IBM MQ チャネルに接続する場合、アプリケーションは **OutboundSNI** を CHANNEL に設定して接続する必要があります。

OutboundSNI に CHANNEL 以外の設定を持つアプリケーションが、証明書ラベルが構成されたチャネルに接続すると、そのアプリケーションは MQRC_SSL_INITIALIZATION_ERROR で拒否され、キュー・マネージャーのエラー・ログに AMQ9673 メッセージが出力されます。

IBM MQ が複数の証明書機能を提供する方法について詳しくは、 IBM MQ が複数の証明書機能を提供する方法 を参照してください。

XMS .NET アプリケーションの開発

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) は、XMS と呼ばれるアプリケーション・プログラミング・インターフェース (API) を提供します。 この API は、Java Message Service (JMS) API と同じ一連のインターフェースを備えています。 IBM MQ Message Service Client (XMS) for .NET には、 XMS の完全に管理された実装が含まれています。これは、どの .NET 準拠言語でも使用できます。

このタスクについて

XMS では以下がサポートされます。

- Point-to-Point メッセージング
- パブリッシュ/サブスクライブ・メッセージング
- 同期メッセージ配信
- 非同期メッセージ配信

XMS アプリケーションは、以下のタイプのアプリケーションとメッセージを交換できます。

- XMS アプリケーション
- IBM MQ classes for JMS アプリケーション
- ネイティブ IBM MQ アプリケーション
- IBM MQ デフォルト・メッセージング・プロバイダーを使用する JMS アプリケーション

XMS アプリケーションは、以下のメッセージング・サーバーに接続し、これらのサーバーのリソースを使用できます。

IBM MQ キュー・マネージャー
アプリケーションはバインディング・モードまたはクライアント・モードのいずれかで接続できます。

WebSphere Application Server service integration bus
アプリケーションは直接 TCP/IP 接続または HTTP over TCP/IP を使用できます。

IBM Integration Bus
アプリケーションとブローカーの間では、WebSphere MQ Real-Time Transport を使用してメッセージを移送します。 WebSphere MQ Multicast Transport を使用してメッセージをアプリケーションに送信することも可能です。

IBM MQ キュー・マネージャーに接続することで、XMS アプリケーションは WebSphere MQ Enterprise Transport を使用して IBM Integration Bus と通信できます。 あるいは、XMS アプリケーションは IBM MQ に接続することでパブリッシュ/サブスクライブを実行することも可能です。

IBM MQ 9.1.1 以降、IBM MQ は Windows 環境のアプリケーションに対して .NET Core をサポートします。 詳しくは、619 ページの『IBM MQ classes for XMS .NET のインストール』を参照してください。

IBM MQ 9.1.2 以降、 IBM MQ は Linux 環境のアプリケーションに対して .NET Core をサポートします。

IBM MQ 9.1.4 以降、XMS .NET 管理アプリケーションは複数のクラスター・キュー・マネージャー間で、自動的に接続のバランスを取るようになりました。 .NET Framework ライブラリーと .NET Standard ライブラリーの両方がサポートされています。 詳しくは、均等クラスターについておよびアプリケーションの自動バランシングを参照してください。

関連タスク
IBM サポートへのお問い合わせ
XMS .NETproblems の問題のトラブルシューティング

XMS でサポートされるメッセージングのスタイル

XMS は、Point-to-Point スタイルおよびパブリッシュ/サブスクライブ・スタイルのメッセージングをサポートします。

メッセージングのスタイルは、メッセージング・ドメインとも呼ばれます。

Point-to-Point メッセージング

一般的な形式の Point-to-Point メッセージングでは、キューイングを使用します。 最も単純な事例では、暗黙的または明示的に宛先キューを指定することにより、アプリケーションが他のアプリケーションにメッセージを送信します。 下位層のメッセージング・システムおよびキューイング・システムは、送信側アプリケーションからメッセージを受信して、そのメッセージをシステムの宛先キューに転送します。 受信側アプリケーションは、受信後、キューからメッセージを取り出すことができます。

下位層のメッセージング・システムとキューイング・システムに IBM Integration Bus が含まれている場合、IBM Integration Bus は、メッセージを複製してメッセージのコピーを別々のキューに転送できます。 その結果、複数のアプリケーションがメッセージを受信できるようになります。 IBM Integration Bus は、メッセージを変換してデータを追加することもできます。

point-to-point メッセージングの重要な特性は、アプリケーションがメッセージの送信時にメッセージをローカル・キューに配置することです。 メッセージの送信先になる宛先キューは、下位層のメッセージング・システムとキューイング・システムによって決まります。 受信側のアプリケーションは、その宛先キューからメッセージを取り出します。

パブリッシュ/サブスクライブ・メッセージング

パブリッシュ/サブスクライブ・メッセージングには、パブリッシャーとサブスクライバーの 2 つのタイプのアプリケーションがあります。

パブリッシャー は、パブリケーション・メッセージの形式で情報を提供します。 パブリッシャーは、メッセージを公開するときにトピックを指定します。トピックは、メッセージ内部の情報の主題を識別します。

サブスクライバー は、公開されている情報のコンシューマーです。 サブスクライバーは、サブスクリプションを作成することによって、関心のあるトピックを指定します。

パブリッシュ/サブスクライブ・システムは、パブリッシャーからパブリケーションを受け取り、サブスクライバーからサブスクリプションを受け取ります。 その後、パブリケーションをサブスクライバーに送付します。 サブスクライバーは、サブスクライブしたトピックについてのみパブリケーションを受け取ります。

パブリッシュ/サブスクライブ・メッセージングの重要な特性は、パブリッシャーがメッセージのパブリッシュ時にトピックを指定することです。 サブスクライバーを指定するわけではありません。 サブスクライバーのないトピックにメッセージを公開した場合、このメッセージを受信するアプリケーションは存在しません。

1 つのアプリケーションがパブリッシャーとサブスクライバーの両方を兼ねることもあります。

XMS オブジェクト・モデル

XMS API はオブジェクト指向インターフェースです。 XMS オブジェクト・モデルは、JMS 1.1 オブジェクト・モデルに基づいています。

主な XMS クラス

主な XMS クラスまたはオブジェクトのタイプは以下のとおりです。

ConnectionFactory
ConnectionFactory オブジェクトは、接続に使用する一連のパラメーターをカプセル化します。 アプリケーションは接続経路の作成に ConnectionFactory を使用します。 アプリケーションは、実行時にパラメーターを提供して ConnectionFactory オブジェクトを作成できます。 あるいは、管理対象オブジェクトのリポジトリーに接続パラメーターを格納しておくことも可能です。 アプリケーションは、そのリポジトリーからオブジェクトを取り出し、そのオブジェクトから ConnectionFactory オブジェクトを作成できます。

接続
Connection オブジェクトは、アプリケーションからメッセージング・サーバーへのアクティブな接続をカプセル化したオブジェクトです。 アプリケーションは、接続を使用してセッションを作成します。

Destination
アプリケーションは、Destination オブジェクトを使用してメッセージを送受信します。 パブリッシュ/サブスクライブ・ドメインでは、Destination オブジェクトはトピックをカプセル化し、point-to-point ドメインでは、Destination オブジェクトはキューをカプセル化します。 アプリケーションは、実行時にパラメーターを提供して Destination オブジェクトを作成できます。 あるいは、管理対象オブジェクトのリポジトリーに格納されているオブジェクト定義から Destination オブジェクトを作成することも可能です。

Session
Session オブジェクトは、メッセージを送受信するための単一スレッド・コンテキストです。 アプリケーションは、Session オブジェクトを使用して、Message、MessageProducer、MessageConsumer の各オブジェクトを作成します。

メッセージ
Message オブジェクトは、アプリケーションが MessageProducer オブジェクトを使用して送信し、MessageConsumer オブジェクトを使用して受信する Message オブジェクトをカプセル化したオブジェクトです。

MessageProducer
MessageProducer オブジェクトは、アプリケーションが宛先にメッセージを送信するために使用するオブジェクトです。

MessageConsumer
MessageConsumer オブジェクトは、アプリケーションが宛先に送信されたメッセージを受信するために使用するオブジェクトです。

XMS オブジェクトとその関係

617 ページの図 52 は、XMS オブジェクトの主な型である ConnectionFactory、Connection、Session、MessageProducer、MessageConsumer、Message、および Destination を示します。 アプリケーションは、接続ファクトリーを使用して接続を作成し、接続を使用してセッションを作成します。 アプリケーションは、次にセッションを使用してメッセージ、メッセージ・プロデューサー、およびメッセージ・コンシューマーを作成します。 アプリケーションはメッセージ・プロデューサーを使用してメッセージを宛先に送信し、メッセージ・コンシューマーを使用して宛先に送信されたメッセージを受信します。

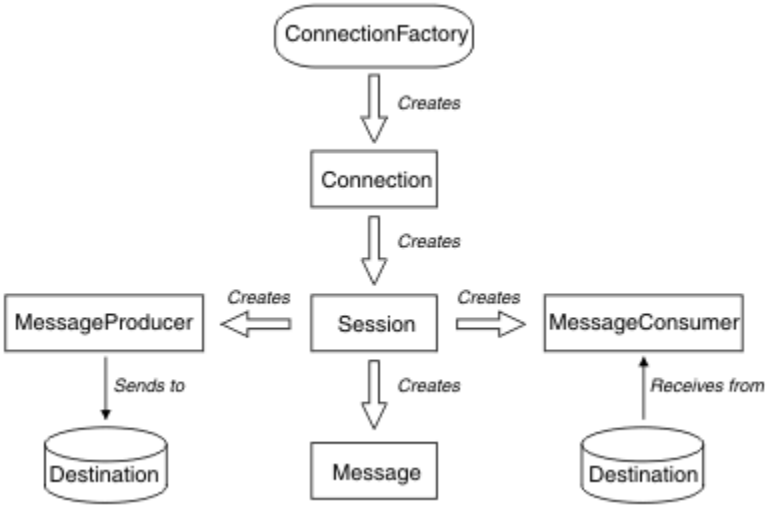


図 *52. XMS* オブジェクトとその関係

XMS .NET では、XMS クラスは一組の .NET インターフェースとして定義されます。 XMS .NET アプリケーションをコーディングするときに必要なのは、宣言済みのインターフェースだけです。

XMS オブジェクト・モデルは、Java Message Service 仕様バージョン 1.1 に記述されている、ドメインに依存しないインターフェースに基づいています。 ドメイン固有のクラス (Topic、TopicPublisher、TopicSubscriber など) は提供されません。

XMS オブジェクトの属性とプロパティー

XMS オブジェクトには、オブジェクトの特性である属性とプロパティーを設定できます。これらは以下のようなさまざまな方法で実装されます。

属性
オブジェクトは、属性に値がない場合でも、常時存在してストレージを占有します。 この点で、属性は固定長データ構造のフィールドと似ています。 異なる特徴としては、属性にはそれぞれ、その値を設定および取得するための独自のメソッドがあることが挙げられます。

プロパティー
オブジェクトのプロパティーが存在してストレージを占有するのは、その値を設定した後だけです。値の設定後にプロパティーを削除したり、そのストレージをリカバリーしたりすることはできません。

値を変更することは可能です。 XMS には、プロパティー値を設定および取得するための、一連の汎用メソッドが備わっています。

管理対象オブジェクト

管理対象オブジェクトを使用すると、クライアント・アプリケーションが使用する接続設定を中央のリポジトリーから管理できます。 アプリケーションは、中央のリポジトリーからオブジェクト定義を取り出して使用することにより、ConnectionFactory オブジェクトや Destination オブジェクトを 作成できます。 管理対象オブジェクトを使用すれば、アプリケーションと、アプリケーションが実行時に使用するリソースとを切り離すことができます。

例えば、XMS アプリケーションの記述やテストは、テスト環境で一組の接続経路と宛先を参照する管理対象オブジェクトを使用して行うことができます。 アプリケーションをデプロイするときには、管理対象オブジェクトを変更して、アプリケーションが実稼働環境の接続と宛先を参照するように構成できます。

XMS は、次の 2 種類の管理対象オブジェクトをサポートしています。

- ConnectionFactory オブジェクト。このオブジェクトをアプリケーションが使用する目的は、サーバーへの初期接続経路の作成です。
- Destination オブジェクト。このオブジェクトをアプリケーションが使用する目的は、送信対象メッセージの宛先と受信対象メッセージの送信元の指定です。 宛先は、アプリケーションの接続先となるサーバー上のトピックまたはキューです。

管理ツール **JMSAdmin** は、IBM MQ に付属しています。 これは、管理対象オブジェクト を管理対象オブジェクトの中央リポジトリーに作成および管理するために使用されます。

リポジトリー内の管理対象オブジェクトは、IBM MQ classes for JMS アプリケーションと XMS アプリケーションで使用できます。 XMS アプリケーションは、 ConnectionFactory オブジェクトおよび Destination オブジェクトを使用して、 IBM MQ キュー・マネージャーに接続できます。 管理者は、リポジトリー内に保持されているオブジェクト定義を、アプリケーション・コードに影響を与えずに変更できます。

次の図は、XMS アプリケーションによる管理対象オブジェクトの通常の使用方法を示しています。 図の左側は、管理コンソールを使用して管理される ConnectionFactory オブジェクト定義と Destination オブジェクト定義を格納しているリポジトリーを示しています。 図の右側は、リポジトリー内部のオブジェクト定義を検索し、そのオブジェクト定義をメッセージング・サーバーとの接続時に使用する XMS アプリケーションを示しています。

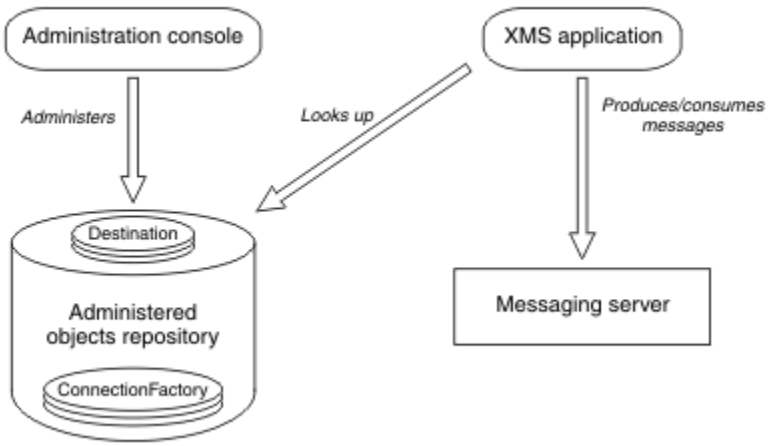


図 *53. XMS* アプリケーションによる管理対象オブジェクトの標準的な使用方法

XMS メッセージ・モデル

XMS メッセージ・モデルは、IBM MQ classes for JMS メッセージ・モデルと同じものです。

特に、XMS は、次に示すように IBM MQ classes for JMS が実装するのと同じメッセージ・ヘッダー・フィールドおよびメッセージ・プロパティーを実装します。

- JMS ヘッダー・フィールド。 これらのフィールドには、JMS というプレフィックスで始まる名前が付いています。
- JMS 定義のプロパティー。 これらのフィールドには、名前が JMSX というプレフィックスで始まるプロパティーがあります。
- IBM 定義のプロパティー。 これらのフィールドには、名前が JMS_IBM_ というプレフィックスで始まるプロパティーがあります。

このため、XMS アプリケーションは、IBM MQ classes for JMS アプリケーションとメッセージを交換できます。 それぞれのメッセージには、アプリケーションによって設定されるヘッダー・フィールドとプロパティーもあれば、XMS または IBM MQ classes for JMS によって設定されるヘッダー・フィールドとプロパティーもあります。 XMS または IBM MQ classes for JMS によって設定されるフィールドの中には、メッセージの送信時に設定されるフィールドもあれば、メッセージの受信時に設定されるフィールドもあります。適切な状況では、ヘッダー・フィールドとプロパティーがメッセージング・サーバー経由でメッセージと一緒に伝搬します。 そのようにして、メッセージを受信するすべてのアプリケーションで使用できるようになります。

関連概念

IBM MQ classes for JMS

Windows Linux IBM MQ classes for XMS .NET のインストール

IBM MQ classes for XMS .NET(サンプルを含む) は、IBM MQ on Windows および Linux と共にインストールされます。

IBM MQ 9.2.0 以降、Microsoft.NET Core 3.1 が、IBM MQ classes for XMS .NET Standard を実行するために必要な最小限のバージョンです。

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、IBM MQ は IBM MQ classes for XMS .NET Standard を使用する .NET 6 アプリケーションをサポートします。 .NET Core 3.1 アプリケーションを使用している場合は、再コンパイルを必要とせずに、 `csproj` ファイルで小さな編集を行い、 `targetframeworkversion` を `"net6.0"`に設定してこのアプリケーションを実行することができます。

V 9.3.1 IBM MQ 9.3.1 は、.NET 6 に対してビルドされた XMS .NET クライアント・ライブラリーをターゲット・フレームワークとして提供します。 IBM MQ 9.3.1 以降、 .NET 6 をターゲット・フレームワークとして使用してビルドされた IBM MQ ライブラリーを使用してアプリケーションを実行するために必要な最小バージョンは Microsoft .NET 6.0 です。

V 9.3.1 IBM MQ 9.3.1 以降、 .NET Standard を使用して作成された XMS .NET クライアント・ライブラリーは、新規フォルダー `netstandard2.0` の下で使用できます。また、ターゲット・フレームワークとして .NET 6 を使用して作成された XMS .NET クライアント・ライブラリーは、 Windows 上の *MQ_INSTALLATION_PATH*`/bin` および Linux 上の *MQ_INSTALLATION_PATH*`/lib64` の下で使用できます。

amqmxmsstd.dll ライブラリー

V 9.3.1 IBM MQ 9.3.1 以降、 `amqmxmsstd.dll` ライブラリーは以下の場所から入手できます。

ターゲット・フレームワークとして .NET Standard 2.0 を使用して作成されたライブラリー

- Windows Windows の場合: *MQ_INSTALLATION_PATH*`\bin\netstandard2.0`。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*`\lib64\netstandard2.0`。

Deprecated これらのライブラリーは非推奨になっており、 IBM は将来のリリースで削除する予定です。

ターゲット・フレームワークとして .NET 6 を使用して作成されたライブラリー

- Windows Windows の場合: *MQ_INSTALLATION_PATH*\bin。 サンプル・アプリケーションは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にインストールされています。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*\lib64。 .NET サンプルは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/base にあります。

LTS IBM MQ 9.3.0 Long Term Support の場合、 IBM MQ classes for XMS .NET Standard ライブラリー amqmxmsstd.dll は、以下の場所で使用可能です。

- Windows Windows の場合: *MQ_INSTALLATION_PATH*\bin。 サンプル・アプリケーションは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/xms にインストールされています。
- Linux Linux の場合: *MQ_INSTALLATION_PATH*/lib64 path。 .NET サンプルは、*MQ_INSTALLATION_PATH*/samp/dotnet/samples/cs/core/xms にあります。

詳細については、554 ページの『IBM MQ classes for .NET のインストール』を参照してください。

重要 : Deprecated V 9.3.1 IBM MQ 9.3.1 以降、ターゲット・フレームワークとして .NET Standard 2.0 を使用して作成された IBM MQ .NET クライアント・ライブラリーは非推奨になり、これらのライブラリーを参照するアプリケーションはコンパイル時に警告 CS0618 をスローします。

Stabilized すべての IBM.XMS.* ライブラリーが以前と同様に提供されていますが、これらのライブラリーは安定化されています。つまり、新しい機能は組み込まれません。 最新機能を利用するには、amqmxmsstd.dll ライブラリーに移行する必要があります。 ただし、 IBM MQ 9.1 Long Term Support リリースまたは Continuous Delivery リリースでは引き続き既存のライブラリーを使用できます。

V 9.3.1 .NET Framework アプリケーションが IBM MQ 9.3.1 より前のバージョンの amqmdnetstd.dll または amqmxmsstd.dll を使用してコンパイルされ、同じアプリケーションが .NET 6 ベースの IBM MQ クライアント・ライブラリーを使用して実行される場合、以下の FileLoadException タイプの例外が .NET によってスローされます。

```
例外をキャッチしました: System.IO.FileLoadException: ファイルまたはアセンブリーをロードできませんでした
'amqmdnetstd, Version =x.x.x.x, Culture=ニュートラル, PublicKeyToken=23d6cb914eeaac0e' または
依存関係の 1 つです。 検出されたアセンブリーのマニフェスト定義が、
アセンブリー参照。 (HRESULT からの例外: 0x80131040)

ファイル名: 'amqmdnetstd, Version =x.x.x.x, Culture=ニュートラル,
PublicKeyToken=23d6cb914eeaac0e'
```

このエラーを解決するには、 *MQ_INSTALLATION_PATH*/bin/netstandard2.0 にあるライブラリーを、 .NET Framework アプリケーションが実行されているディレクトリーにコピーする必要があります。

IBM MQ 9.2.0 以降、NuGet レポジトリーからのダウンロードに IBM MQ classes for XMS .NET Standard が使用できるようになりました。 NuGet パッケージには、amqmxmsstd.dll ライブラリーと amqmdnetstd.dll ライブラリーの両方が含まれています。 amqmxmsstd.dll は amqmdnetstd.dll に依存しており、 XMS .NET Core アプリケーションをパッケージ化する際には、 amqmxmsstd.dll と amqmdnetstd.dll の両方を XMS .NET Core アプリケーションとともにパッケージ化する必要があります。 詳細については、622 ページの『NuGet リポジトリーからの IBM MQ classes for XMS .NET のダウンロード』を参照してください。

dspmqver コマンド

dspmqver コマンドを使用して、.NET Core コンポーネントのバージョン情報およびビルド情報を表示できます。

IBM MQ classes for XMS .NET Framework と IBM MQ classes for XMS .NET (.NET Standard および .NET 6 ライブラリー) の比較

以下の表に、IBM MQ classes for XMS .NET (.NET Standard および .NET 6 ライブラリー) のフィーチャーと比較した IBM MQ classes for XMS .NET Framework のフィーチャーをリストします。

表 *80. IBM MQ classes for XMS .NET Framework* と *IBM MQ classes for XMS .NET (.NET Standard* および *.NET 6* ライブラリー*)* の相違点

フィーチャー	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET (.NET Standard および .NET 6 ライブラリー)
クラス名 (API)	すべてのクラスは、各ネットワークで同じ状態を維持します。	すべてのクラスは、各ネットワークで同じ状態を維持します。
オペレーティング・システム	Windows	Windows Docker 化したコンテナー Linux macOS
`app.config` ファイル (再配布可能なクライアントでトレースを有効にするための構成ファイル)	`app.config` ファイルは、再配布可能パッケージのトレースを有効にするために使用されます。	`app.config` サポートされていません。 環境変数を使用します。
トレース	XMS .NET クライアントをトレースするには、トレースを有効にするために使用される環境変数 **XMS_TRACE_ON** など、既存の環境変数を使用できます。詳しくは、 XMS 環境変数を使用した XMS .NET トレースの構成を参照してください。 再配布可能クライアントの場合、`app.config` ファイルを使用してトレースを有効にすることができます。	XMS .NET クライアントをトレースするには、トレースを有効にするために使用される環境変数 **XMS_TRACE_ON** など、既存の環境変数を使用できます。詳しくは、 XMS 環境変数を使用した XMS .NET トレースの構成を参照してください。
トランスポート・モード	管理、非管理、およびバインディング	管理対象
TLS	Windows 鍵ストアを使用して証明書を保管します。	Windows では、証明書の保管に鍵ストアを使用する必要があります。 許可値は *USER と *SYSTEM です。 入力に基づいて、IBM MQ .NET クライアントによって、現在のユーザーまたはシステム全体の Windows 鍵ストアが確認されます。 Linux では、X509Store クラスを使用して証明書をインストールし、.NET Core によって場所 "`.dotnet/corefx/cryptography/x509stores`" に証明書をインストールすることをお勧めします。
CCDT	サポート対象	サポート対象。CCDT パスの設定は .NET Framework クラスと同じです。

表 80. *IBM MQ classes for XMS .NET Framework* と *IBM MQ classes for XMS .NET (.NET Standard および .NET 6 ライブラリー)* の相違点 (続き)

フィーチャー	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET (.NET Standard および .NET 6 ライブラリー)**
クライアント自動再接続	サポート対象	サポート対象
分散トランザクション	サポート対象	サポート対象外
グローバル・アセンブリー・キャッシュ (GAC) へのダイナミック・リンク・ライブラリー (dll) のインストール	Dll は IBM MQ インストールの一部として GAC にインストールされます。	Dll は IBM MQ インストールの一部として GAC にインストールされません。
WMQ、WPM、および RTT 接続タイプのサポート	WMQ、WPM、および RTT 接続タイプをサポート	WMQ のみをサポート
JNDI 管理対象オブジェクト	LDAP とファイル・システムをサポート	ファイル・システムのみをサポート

V 9.3.0 V 9.3.0 IBM MQ 9.3.0 以降、 IBM MQ classes for XMS .NET Framework を実行するには、 Microsoft.NET Framework V4.7.2 以降をインストールする必要があります。

関連タスク

629 ページの『XMS サンプル・アプリケーションの使用』
XMS .NET サンプル・アプリケーションは、各 API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

Windows Linux NuGet リポジトリーからの IBM MQ classes for XMS .NET のダウンロード

IBM MQ classes for XMS .NET は、.NET 開発者が容易にコンシュームできるようにするため、NuGet レポジトリーからダウンロードして使用することができます。

このタスクについて

NuGet は、.NET を含む Microsoft 開発プラットフォーム用のパッケージ・マネージャーです。 NuGet クライアント・ツールは、パッケージを作成して取り込むための機能を提供します。 NuGet パッケージは、コンパイル済みコード (DLL) を含む .nupkg 拡張子、そのコードに関連するその他のファイル、およびパッケージのバージョン番号などの情報を含む記述マニフェストを持つ単一の圧縮ファイルです。

`amqmdnetstd.dll` ライブラリーと `amqmxmsstd.dll` ライブラリーの両方を含む `IBMXMSDotnetClient` NuGet パッケージを NuGet Gallery (すべてのパッケージ作成者と利用者が使用する中央パッケージ・リポジトリー) からダウンロードできます。

注 : V 9.3.1 IBM MQ 9.3.1 以降、 NuGet パッケージには、ターゲット・フレームワークとして .NET Standard 2.0 および .NET 6 を使用して作成されたライブラリーが含まれています。 .NET Standard 2.0 のライブラリーは `netstandard2.0` フォルダーの下にあり、 .NET 6 のライブラリーは `net6.0` フォルダーの下にあります。

`IBMXMSDotnetClient` パッケージのダウンロードするには、以下の 3 つの方法があります。

- Microsoft Visual Studio を使用します。 NuGet は Microsoft Visual Studio の拡張として配布されます。 Microsoft Visual Studio 2012 以降では、NuGet はデフォルトでプリインストールされています。
- NuGet Package Manager または .NET CLI を使用してコマンド・ラインから。
- Web ブラウザーを使用して。

再配布可能パッケージの場合、環境変数 **XMS_TRACE_ON** を使用してトレースを有効にします。

手順

- Microsoft Visual Studio 内の Package Manager UI を使用して IBMXMSDotnetClient パッケージをダウンロードするには、以下の手順を実行します。
  a) .NET プロジェクトを右クリックしてから、「**Nuget パッケージの管理 (Manage Nuget Packages)**」をクリックします。
  b)「**参照**」 タブをクリックして、 "IBMXMSDotnetClient"を検索します。
  c) パッケージを選択して、「**インストール**」をクリックします。
  インストール中に、Package Manager はコンソール文の形式で進行情報を示します。
- コマンド行から IBMXMSDotnetClient パッケージをダウンロードするには、以下のいずれかのオプションを選択します。
  - NuGet Package Manager を使用して、以下のコマンドを入力します。

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    インストール中に、Package Manager はコンソール文の形式で進行情報を示します。 出力をログ・ファイルにリダイレクトするができます。
  - .NET CLI を使用して、以下のコマンドを入力します。

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Web ブラウザーを使用して、https://www.nuget.org/packages/IBMXMSDotnetClient から IBMXMSDotnetClient パッケージをダウンロードします。

関連概念

554 ページの『IBM MQ classes for .NET のインストール』
IBM MQ classes for .NET(サンプルを含む) は、 IBM MQ とともに Windows および Linux にインストールされます。

IBM MQ Client for .NET のライセンス情報

関連タスク

560 ページの『NuGet リポジトリーからの IBM MQ classes for .NET のダウンロード』
IBM MQ classes for .NET を NuGet リポジトリーからダウンロードできます。これにより、 .NET 開発者は簡単に利用できます。

メッセージング・サーバー環境のセットアップ

このセクションのトピックでは、XMS アプリケーションがサーバーに接続できるように、メッセージング・サーバー環境をセットアップする方法を説明します。

このタスクについて

IBM MQ キュー・マネージャーに接続するアプリケーションでは、IBM MQ クライアント (バインディング・モードの場合はキュー・マネージャー) が必要です。

ブローカーとのリアルタイム接続を使用するアプリケーションには、現在のところ前提条件はありません。

XMS アプリケーション (XMS に付属のサンプル・アプリケーションを含む) を実行するには、その前にメッセージング・サーバー環境をセットアップする必要があります。

このセクションでは、以下のトピックについて説明します。

- 626 ページの『IBM MQ キュー・マネージャーに接続するアプリケーション用のキュー・マネージャーおよびブローカーの構成』
- 619 ページの『IBM MQ classes for XMS .NET のインストール』
- 627 ページの『ブローカーへのリアルタイム接続を使用するアプリケーション用のブローカーの構成』

- 628 ページの『WebSphere Application Server に接続するアプリケーション用のサービス統合バスの構成』

XMS .NET でのメッセージ・リスナー

メッセージ・リスナーは、メッセージを非同期に受信するために使用されます。 MessageConsumer.receive() 呼び出しとは異なり、メッセージ・リスナーは呼び出しスレッドをブロックしません。代わりに、アプリケーション指定のコールバック・メソッド (通常は onMessage メソッド) にメッセージを配信します。

Connection.Start() メソッドが呼び出されると、メッセージ配信が開始されます。 メッセージ配信は、Connection.Stop() メソッドおよび Connection.Start() メソッドを使用して、いつでも停止および再開できます。

メッセージ・リスナーをセッション内の少なくとも 1 つのコンシューマーに設定した後に Connection.Start() メソッドが呼び出されると、そのセッションは非同期セッションになります。 セッションが非同期になると、 XMS .NET 同期メソッドを呼び出すことはできません。 例えば、 MessageProducer.Send() などです。 これを行うと、 IBM MQ 理由コード MQRC_HCONN_ASYNC_ACTIVE (2500) の例外が発生します。

非同期セッションでの同期呼び出し

Session.Close は、非同期セッションで許可される唯一の同期呼び出しです。 アプリケーションは、メッセージ・リスナー・コールバック・メソッド、つまり onMessage メソッドを使用して、同期呼び出し (Session.Close を除く) を行うこともできます。

これらの 2 つのオプションを除き、アプリケーションが同期呼び出しを行うには、 Connection.Stop() メソッドを使用して接続を停止する必要があります。 呼び出しを行った後、 Connection.Start() メソッドを使用して接続を再開する必要があります。 メッセージ配信を再開します。

セッションに含めることができる非同期メッセージ・コンシューマーの数

セッションは、複数の非同期メッセージ・コンシューマーを持つことができます。 ただし、メッセージは常に 1 つのコンシューマーにのみ配信されます。 実際には、 XMS .NET が最初のメッセージを配信するためにコンシューマーの onMessage() メソッドを呼び出したときに 2 番目のメッセージが到着した場合、onMessage() メソッドが戻るまで、2 番目のメッセージはセッションのコンシューマーに配信されません。

2 番目のメッセージは、 onMessage() メソッドが戻った後にのみ、セッション内のコンシューマーに配信されます。 これは、セッションが 1 つのスレッドのみを使用してコンシューマーへのメッセージ配信を管理するためです。 つまり、一度に配信できるメッセージは 1 つのみであり、コンシューマーは任意のメッセージにすることができます。

アプリケーションが並行メッセージ配信を必要とする場合、つまり、すべてのコンシューマーが同時にメッセージを受信する必要がある場合、アプリケーションは複数のセッションを作成し、それぞれが 1 つの非同期メッセージ・コンシューマーを持つ必要があります。

以下の例は、この機能をより明確に示しています。

最初の例では、1 つのセッションに複数の非同期メッセージ・コンシューマーがあります。 セッション S には 3 つの非同期メッセージ・コンシューマーがあります。 AMC1、 AMC2 、および AMC3 は、3 つの異なる宛先 Q1、 Q2 、および Q3 からメッセージを受信します。

セッション S は 1 つしかないため、コンシューマー AMC1、 AMC2、および AMC3 にメッセージを配信するためのメッセージ配信スレッドのみが存在します。 セッションが AMC1 にメッセージを配信する場合、Q2 および Q3 に配信準備ができているメッセージがあっても、他の 2 つのコンシューマー AMC2 および AMC3 は待機します。

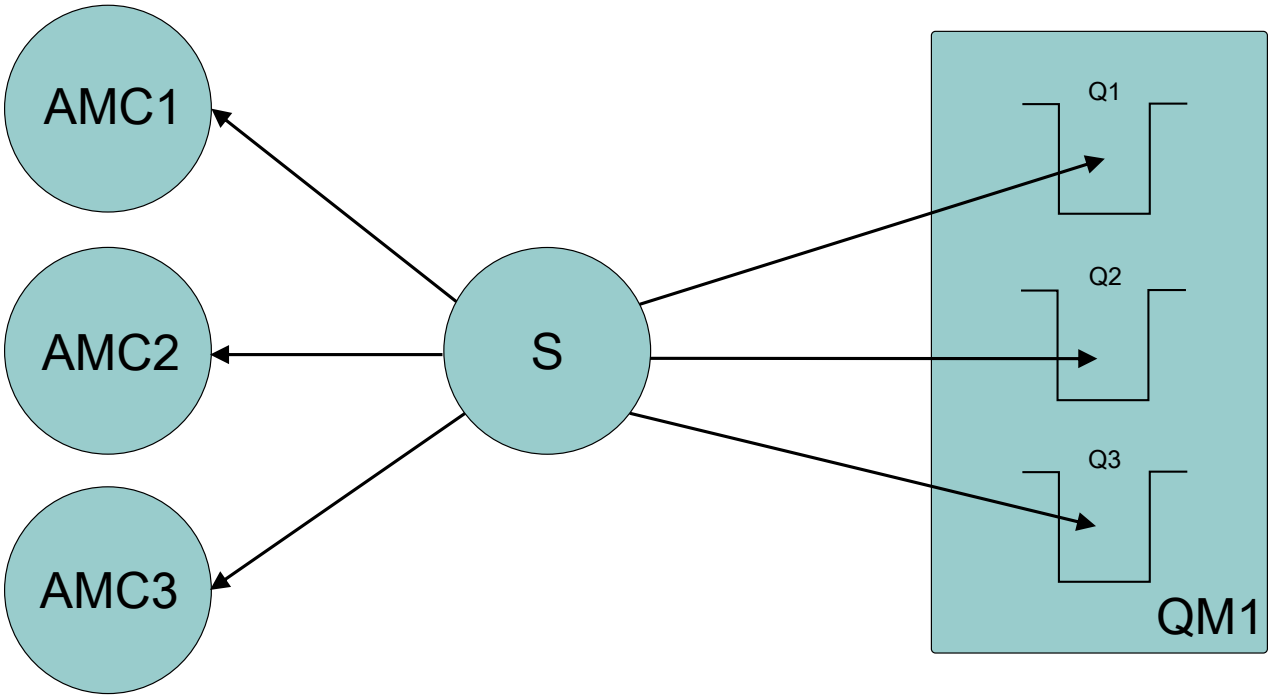


図 *54. 3* つの非同期メッセージ・コンシューマーとの *1* つのセッション

2 番目のケースでは、複数のセッション S1、 S2、および S3 があり、それぞれに 1 つの非同期メッセージ・コンシューマー AMC1、 AMC2、および AMC3 があります。 セッションごとに 1 つのコンシューマーがあるため、メッセージは同時にコンシューマーに配信されます。

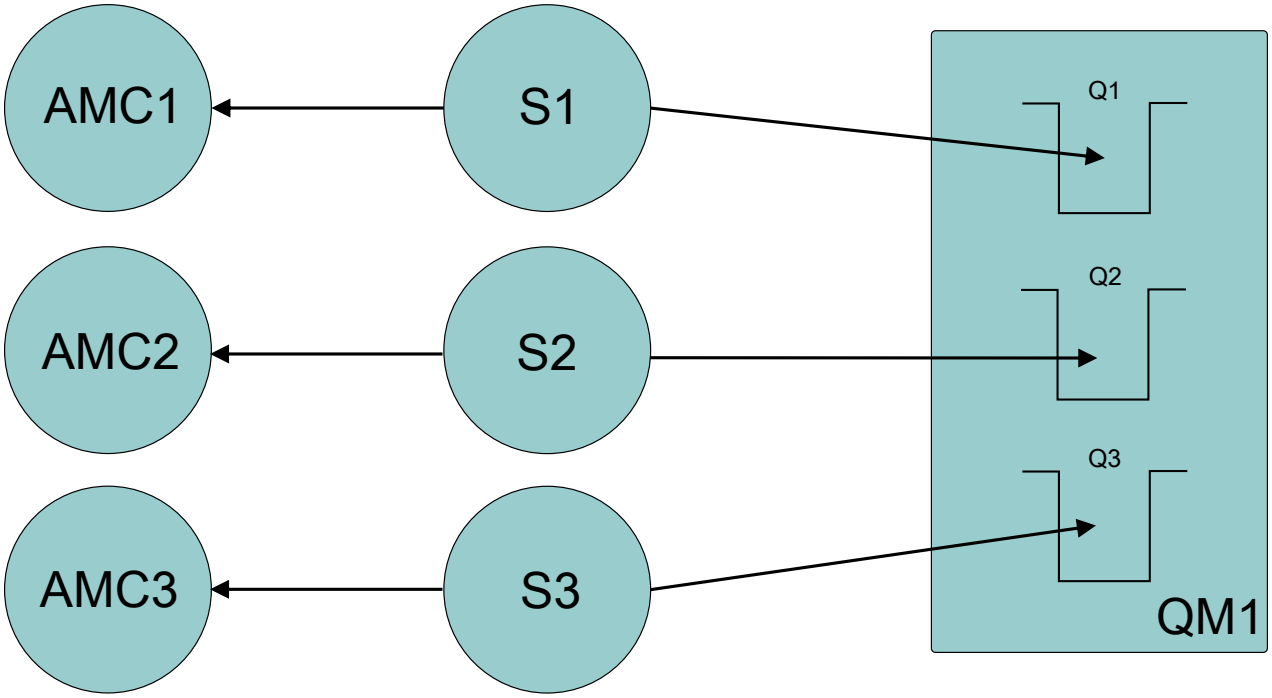


図 *55.* それぞれが *1* つの非同期メッセージ・コンシューマーを持つ複数セッション

これは、並行メッセージ配信が必要な場合に、複数のセッションが必要であることを示しています。

IBM MQ キュー・マネージャーに接続するアプリケーション用のキュー・マネージャーおよびブローカーの構成

このセクションでは、IBM WebSphere MQ 7.0.1 以降を使用していることを前提としています。 IBM MQ キュー・マネージャーに接続するアプリケーションを実行するには、その前にキュー・マネージャーを構成する必要があります。 パブリッシュ/サブスクライブ・アプリケーションでは、キュー・パブリッシュ/サブスクライブ・インターフェースを使用している場合に追加構成が必要です。

始める前に

XMS は、IBM Integration Bus または WebSphere Message Broker 6.1 以降で動作します。

このタスクを開始する前に、以下の手順を実行します。

- アプリケーションが、稼働中のキュー・マネージャーにアクセスできることを確認します。
- アプリケーションがパブリッシュ/サブスクライブ・アプリケーションであり、キュー・パブリッシュ/サブスクライブ・インターフェースを使用する場合は、キュー・マネージャーの **PSMODE** 属性が ENABLED に設定されていることを確認してください。
- アプリケーションが使用する接続ファクトリーでキュー・マネージャーへ接続するようにプロパティーが適切に設定されていることを確認します。 アプリケーションがパブリッシュ/サブスクライブ・アプリケーションである場合は、適切な接続ファクトリー・プロパティーがブローカーを使用するように設定されていることを確認します。 接続ファクトリーのプロパティーについて詳しくは、 ConnectionFactory のプロパティーを参照してください。

このタスクについて

IBM MQ JMS アプリケーションを実行するようにキュー・マネージャーおよびキュー・パブリッシュ/サブスクライブ・インターフェースを構成するのと同じ方法で、 XMS アプリケーションを実行するようにキュー・マネージャーおよびブローカーを構成します。 以下のステップで、実行する必要のある操作を要約しています。

手順

1. キュー・マネージャーで、アプリケーションが必要とするキューを作成します。

   キューの作成方法の概要については、「キューの定義」を参照してください。

   アプリケーションがパブリッシュ/サブスクライブ・アプリケーションであり、IBM MQ classes for JMS システム・キューへのアクセスを必要とするキュー型パブリッシュ/サブスクライブ・インターフェースを使用する場合は、ステップ 4a でキューを作成します。
2. アプリケーションに関連付けられたユーザー ID に、キュー・マネージャーに接続する権限と、キューにアクセスするための適切な権限を付与します。

   許可についての概要は、「保護」を参照してください。 アプリケーションがクライアント・モードでキュー・マネージャーに接続する場合は、「クライアントおよびサーバー」も参照してください。
3. アプリケーションがクライアント・モードでキュー・マネージャーに接続する場合は、キュー・マネージャーでサーバー接続チャネルが定義されていること、およびリスナーが開始されていることを確認してください。

   キュー・マネージャーに接続するアプリケーションごとにこのステップを実行する必要はありません。 1 つのサーバー接続チャネル定義および 1 つのリスナーが、クライアント・モードで接続するすべてのアプリケーションをサポートします。
4. アプリケーションがパブリッシュ/サブスクライブ・アプリケーションで、キュー型パブリッシュ/サブスクライブ・インターフェースを使用している場合は、以下のステップを実行します。

   a) キュー・マネージャーで、IBM MQ で提供されている MQSC コマンドのスクリプトを実行して、IBM MQ classes for JMS システム・キューを作成します。 IBM Integration Bus または WebSphere Message Broker に関連付けられたユーザー ID に、キューにアクセスする権限があることを確認してください。

スクリプトの保管場所および実行方法については、「IBM MQ classes for Java の使用」を参照してください。

キュー・マネージャーに対して、このステップは 1 回だけ実行します。 同一の IBM MQ classes for JMS システム・キューのセットが、キュー・マネージャーに接続しているすべての XMS アプリケーションおよび IBM MQ classes for JMS アプリケーションをサポートできます。

b) アプリケーションに関連するユーザー ID に、IBM MQ classes for JMS システム・キューにアクセスする権限を与えます。

ユーザー ID が必要とする権限については、「IBM MQ classes for JMS の使用」を参照してください。

c) IBM Integration Bus または WebSphere Message Broker のブローカーの場合は、アプリケーションが公開するメッセージを送信するキューを保守するためのメッセージ・フローを、作成およびデプロイします。

基本的なメッセージ・フローは、公開されたメッセージを読み取る MQInput メッセージ処理ノードと、メッセージを公開する Publication メッセージ処理ノードで構成されています。

メッセージ・フローを作成してデプロイする方法については、 IBM Integration Bus 製品資料ライブラリーの Web ページから入手できる IBM Integration Bus または WebSphere Message Broker の製品資料を参照してください。

適切なメッセージ・フローが既にブローカーでデプロイされている場合は、このステップを実行する必要はありません。

タスクの結果
これで、アプリケーションを開始することができます。

ブローカーへのリアルタイム接続を使用するアプリケーション用のブローカーの構成

ブローカーへのリアルタイム接続を使用するアプリケーションを実行するには、まずそのブローカーを構成する必要があります。

始める前に

この作業を開始する前に、以下の手順を実行してください。

- アプリケーションが、稼働中のブローカーにアクセスできることを確認します。
- アプリケーションが使用する接続ファクトリーで、プロパティーがブローカーへのリアルタイム接続用に適切に設定されていることを確認します。 接続ファクトリーのプロパティーについて詳しくは、ConnectionFactory のプロパティーを参照してください。

このタスクについて
XMS アプリケーションを実行するようにブローカーを構成する場合と同様に、IBM MQ classes for JMS アプリケーションを実行するようにブローカーを構成します。 以下のステップで、実行する必要のある操作を要約しています。

手順

1. ブローカーがメッセージを listen および公開する TCP/IP ポートからメッセージを読み取るためのメッセージ・フローを作成およびデプロイします。

   以下のいずれかの方法で、これを実行することができます。

   - **Real-timeOptimizedFlow** メッセージ処理ノードを含むメッセージ・フローを作成します。
   - **Real-timeInput** メッセージ処理ノードおよび Publication メッセージ処理ノードを含むメッセージ・フローを作成します。

リアルタイム接続に使用されるポートを listen するには、**Real-timeOptimizedFlow** または **Real-timeInput** ノードを構成する必要があります。 XMS では、リアルタイム接続のデフォルト・ポート番号は 1506 です。

適切なメッセージ・フローが既にブローカーでデプロイされている場合は、このステップを実行する必要はありません。

2. IBM MQ classes for JMS を使用してアプリケーションにメッセージが配信されるようにする必要がある場合は、マルチキャストを使用可能にするようにブローカーを構成します。 信頼性の高いマルチキャストが必要なトピックに、信頼性の高いサービス品質を指定して、マルチキャストを有効にする必要があるトピックを構成します。
3. アプリケーションがブローカーへの接続時にユーザー ID およびパスワードを提供し、ブローカーがこの情報を使用してアプリケーションを認証するようにしたい場合は、単純な telnet に類似したパスワード認証を行うようにユーザー・ネーム・サーバーおよびブローカーを構成します。

タスクの結果
これで、アプリケーションを開始することができます。

WebSphere Application Server に接続するアプリケーション用のサービス統合バスの構成

WebSphere Application Server service integration technologies サービス統合バスに接続するアプリケーションを実行するには、その前に、デフォルトのメッセージング・プロバイダーを使用する JMS アプリケーションを実行するようにサービス統合バスを構成する方法と同じ方法で、サービス統合バスを構成する必要があります。

始める前に
このタスクを開始する前に、以下のステップを実行する必要があります。

- メッセージング・バスが作成されたこと、およびサーバーがバス・メンバーとしてバスに追加されたことを確認します。
- アプリケーションが、稼働中のメッセージング・エンジンを少なくとも 1 つ持つサービス統合バスにアクセスできることを確認します。
- HTTP オペレーションが必要である場合、HTTP メッセージング・エンジンのインバウンド・トランスポート・チャネルを定義する必要があります。 SSL と TCP のチャネルは、デフォルトで、サーバーのインストール時に定義されます。
- アプリケーションが使用する接続ファクトリーで、ブートストラップ・サーバーを使用してサービス統合バスへ接続するようにプロパティーが適切に設定されていることを確認します。 必要最小限の情報は、以下のとおりです。
  - プロバイダー・エンドポイント。これは、メッセージング・サーバーへの (すなわちブートストラップ・サーバーを通じた) 接続をネゴシエーションする際に使用するロケーションおよびプロトコルを記述します。 デフォルト設定でインストールしたサーバーに関する最も単純な形式では、プロバイダー・エンドポイントをサーバーのホスト名に設定できます。
  - メッセージの送信時に通過するバスの名前。

  接続ファクトリーのプロパティーについて詳しくは、ConnectionFactory のプロパティーを参照してください。

このタスクについて

キューまたはトピック・スペースが必要であれば、それを定義する必要があります。 デフォルトでは、Default.Topic.Space というトピック・スペースがサーバーのインストール時に定義されますが、さらにトピック・スペースが必要な場合は、自分でそれらのトピック・スペースを作成する必要があります。 トピック・スペース内の個々のトピックを事前に定義する必要はありません。サーバーが必要に応じて動的に個々のトピックをインスタンス化します。

以下のステップで、実行する必要のある操作を要約しています。

手順

1. Point-to-Point メッセージングのためにアプリケーションが必要とするキューを作成します。
2. パブリッシュ/サブスクライブ・メッセージングのためにアプリケーションが必要とする追加トピック・スペースを作成します。

タスクの結果

これで、アプリケーションを開始することができます。

XMS サンプル・アプリケーションの使用

XMS .NET サンプル・アプリケーションは、各 API の一般的な機能の概要を示します。 インストール環境やメッセージング・サーバーのセットアップを検証したり、ユーザー独自のアプリケーションを作成したりするときに活用できます。

このタスクについて

独自のアプリケーションを作成するために手助けが必要な場合は、サンプル・アプリケーションを開始点として使用できます。 各アプリケーションのソース・バージョンとコンパイル・バージョンの両方が提供されています。 サンプルのソース・コードを検討し、アプリケーションに必要な各オブジェクト (ConnectionFactory、Connection、Session、Destination、さらに Producer と Consumer のいずれかまたは両方) を作成するための主な手順や、アプリケーションの動作を指定するときに必要な特定のプロパティーを設定するための主な手順を見極めます。 詳細については、632 ページの『XMS アプリケーションの作成』を参照してください。 これらのサンプルは、XMS の将来のリリースで変更される可能性があります。

XMS に用意されている一連のサンプル・アプリケーション (API ごとに 1 セットずつ) を以下の表にまとめます。

表 *81. XMS .NET* 用のサンプル・アプリケーション

サンプル名	説明
SampleConsumerCS	キューからメッセージを取り込んだり、トピックにサブスクライブしたりするメッセージ・コンシューマー・アプリケーション。
SampleProducerCS	キューまたはトピックへのメッセージを作成するメッセージ・プロデューサー・アプリケーション。
SampleConfigCS	ファイル・ベースの管理対象オブジェクト・リポジトリーを作成するために使用できる構成アプリケーション。 このアプリケーションには、特定の接続設定の接続ファクトリーと宛先が格納されています。 この管理対象オブジェクト・リポジトリーは、各サンプル・コンシューマー・アプリケーションおよびサンプル・プロデューサー・アプリケーションで使用できます。

さまざまな API の同一の機能をサポートするサンプルには、構文上の違いがあります。

- サンプルのメッセージ・コンシューマー・アプリケーションとメッセージ・プロデューサー・アプリケーションはどちらも、以下の機能をサポートしています。
  - IBM MQ、IBM Integration Bus への接続 (ブローカーへのリアルタイム接続を使用)、および WebSphere Application Server service integration bus への接続
  - 初期コンテキスト・インターフェースによる管理対象オブジェクト・リポジトリーの検索
  - キューへの接続 (IBM MQ および WebSphere Application Server service integration bus) およびトピックへの接続 (IBM MQ、 ブローカーへのリアルタイム接続、および WebSphere Application Server service integration bus)
  - ベース、バイト、マップ、オブジェクト、ストリーム、およびテキストの各メッセージ

- サンプル・メッセージ・コンシューマー・アプリケーションは、同期受信モード、非同期受信モード、および SQL セレクター・ステートメントをサポートしています。
- サンプル・メッセージ・プロデューサー・アプリケーションでは、永続送達モードと非永続送達モードがサポートされています。

サンプルは以下のいずれかのモードで作動します。

シンプル・モード
最小限のユーザー入力でサンプルを実行できます。

拡張モード
サンプルの動作を詳細にカスタマイズできます。

すべてのサンプルは相互に互換性があるため、異なる言語間で操作できます。

Windows IBM MQ 9.1.1 以降、IBM MQ は、Windows 環境で .NET Core for XMS .NET をサポートします。 IBM MQ classes for .NET Standard (サンプルを含む) は、IBM MQ の標準インストールの一環でデフォルトでインストールされます。

Linux IBM MQ 9.1.2 以降、IBM MQ は、Linux 環境内のアプリケーション用の .NET Core をサポートします。

XMS .NET のサンプル・アプリケーションは、`&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms` にインストールされています。

詳細については、619 ページの『IBM MQ classes for XMS .NET のインストール』を参照してください。

.NET サンプル・アプリケーションの実行

.NET サンプル・アプリケーションは、シンプル・モードまたは拡張モードで対話式に実行できます。自動生成の応答ファイルまたはカスタマイズした応答ファイルを使用して非対話式に実行することも可能です。

始める前に

付属のサンプル・アプリケーションを実行する前に、最初にメッセージング・サーバー環境をセットアップし、 アプリケーションがサーバーに接続できるようにする必要があります。 623 ページの『メッセージング・サーバー環境のセットアップ』を参照。

手順

.NET サンプル・アプリケーションを実行するための手順は、以下のとおりです。

ヒント : サンプル・アプリケーションを実行している場合、次のように入力します。 いつでも、次に何をすべきかについての支援を得ることができます。

1. サンプル・アプリケーションを実行するときのモードを選択します。
   `Advanced` または `Simple` と入力します。
2. 質問に回答します。
   質問の最後に大括弧で表示されるデフォルト値を選択するには、Enter を押します。 別の値を選択するには、適切な値を入力してから Enter を押します。

   質問の例を示します。

   ```
   Enter connection type [wpm]:
   ```

   この場合、デフォルト値は `wpm` です (WebSphere Application Server service integration bus への接続)。

タスクの結果

サンプル・アプリケーションを実行すると、応答ファイルが現在の作業ディレクトリーに自動生成されます。 応答ファイル名は、*connection_type-sample_type*.rsp の形式になっています。例えば、wpm-producer.rsp のようになります。 必要に応じて、生成した応答ファイルを使用して、同じオプションでサンプル・アプリケーションを再実行することも可能です。そのときに、オプションを再び入力する必要はありません。

関連タスク

.NET サンプル・アプリケーションの作成
サンプルの .NET アプリケーションを作成する場合は、選択したサンプルの実行可能バージョンが作成されます。

ユーザー独自のアプリケーションの作成
ユーザー独自のアプリケーションをビルドする方法は、サンプル・アプリケーションをビルドする場合と同様です。

.NET サンプル・アプリケーションの作成

サンプルの .NET アプリケーションを作成する場合は、選択したサンプルの実行可能バージョンが作成されます。

始める前に

該当するコンパイラーをインストールします。 このタスクは、Microsoft Visual Studio 2012 がインストールされており、それを使用することに慣れていることを前提としています。

手順

.NET サンプル・アプリケーションをビルドするための手順は、以下のとおりです。

1. .NET サンプルで提供されている Samples.sln ソリューション・ファイルをクリックします。
2. 「ソリューション エクスプローラ」ウィンドウで Samples というソリューションを右クリックし、「**ソリューションのビルド**」を選択します。

タスクの結果

実行可能プログラムは、選択した構成に応じて、サンプルの該当するサブフォルダー (bin/Debug または bin/Release のいずれか) で作成されます。 このプログラムは、フォルダーと同じ名前を持ち、接尾部が CS になっています。 例えば、メッセージ・プロデューサー・サンプル・アプリケーションの C# バージョンを作成する場合、SampleProducerCS.exe は SampleProducer フォルダー内に作成されます。

関連タスク

.NET サンプル・アプリケーションの実行
.NET サンプル・アプリケーションは、シンプル・モードまたは拡張モードで対話式に実行できます。自動生成の応答ファイルまたはカスタマイズした応答ファイルを使用して非対話式に実行することも可能です。

ユーザー独自のアプリケーションの作成
ユーザー独自のアプリケーションをビルドする方法は、サンプル・アプリケーションをビルドする場合と同様です。

631 ページの『ユーザー独自のアプリケーションの作成』
ユーザー独自のアプリケーションをビルドする方法は、サンプル・アプリケーションをビルドする場合と同様です。

ユーザー独自のアプリケーションの作成

ユーザー独自のアプリケーションをビルドする方法は、サンプル・アプリケーションをビルドする場合と同様です。

始める前に

該当するコンパイラーをインストールします。 このタスクは、Microsoft Visual Studio 2012 がインストールされており、それを使用することに慣れていることを前提としています。

手順

- .NET アプリケーションをビルドします (631 ページの『.NET サンプル・アプリケーションの作成』を参照)。

  ユーザー独自のアプリケーションの作成方法についての補足指示については、各サンプル・アプリケーションの Make ファイルを活用してください。

  ヒント: 障害発生時の問題診断を支援するため、シンボルを組み込んだアプリケーションをコンパイルすると便利な場合があります。

関連タスク

.NET サンプル・アプリケーションの実行

.NET サンプル・アプリケーションは、シンプル・モードまたは拡張モードで対話式に実行できます。自動生成の応答ファイルまたはカスタマイズした応答ファイルを使用して非対話式に実行することも可能です。

.NET サンプル・アプリケーションの作成

サンプルの .NET アプリケーションを作成する場合は、選択したサンプルの実行可能バージョンが作成されます。

XMS アプリケーションの作成

このセクションのトピックでは、XMS アプリケーションを作成する際に役立つ情報を提供します。

このタスクについて

このセクションでは、XMS アプリケーションを作成する場合の一般的な概念について説明します。
XMS .NET アプリケーションの作成に固有の情報については、 651 ページの『XMS .NET アプリケーションの作成』 も参照してください。

IBM MQ 9.2.0 以降では、XMS.NET ダイナミック・リンク・ライブラリーの数が大幅に削減され、合計 5 つになりました。 この 5 つのダイナミック・リンク・ライブラリーは、以下のとおりです。

- `IBM.XMS.dll` - すべての各国語のメッセージを組み込みます。
- `IBM.XMS.Comms.RMM.dll`
- 次の 3 つのポリシー・ダイナミック・リンク・ライブラリー
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

V 9.3.0 Removed V 9.3.0 XMS .NET マルチキャスト・メッセージング (RMM を使用) は、IBM MQ 9.2 以降で非推奨になり、 IBM MQ 9.3 で削除されした。

このセクションでは、以下のトピックについて説明します。

- 634 ページの『スレッド化モデル』
- 634 ページの『ConnectionFactories オブジェクトと Connection オブジェクト』
- 636 ページの『セッション数』
- 639 ページの『宛先』
- 641 ページの『メッセージ・プロデューサー』
- 642 ページの『メッセージ・コンシューマー』
- 645 ページの『キュー・ブラウザー』

- 646 ページの『リクエスター』
- 646 ページの『オブジェクトの削除』
- 647 ページの『XMS プリミティブ型』
- 648 ページの『プロパティー値のデータ型の暗黙的な変換』
- 650 ページの『イテレーター』
- 650 ページの『コード化文字セット ID』
- 650 ページの『XMS エラーおよび例外コード』
- 631 ページの『ユーザー独自のアプリケーションの作成』

Windows IBM MQ XMS .NET プロジェクト・テンプレートの使用

IBM MQ XMS .NET クライアントには、プロジェクト・テンプレートを使用して XMS .NET Core アプリケーションの開発を支援する機能があります。

始める前に

ご使用のシステムに Microsoft Visual Studio 2017 以降、および .NET Core 2.1 がインストールされている必要があります。

XMS .NET テンプレートを以下からコピーする必要があります。

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip

へのディレクトリー

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

ディレクトリー。ここで、

- *&MQ_INSTALL_ROOT* はインストール済み環境のルート・ディレクトリーです。
- *&USER_HOME_DIRECTORY* はユーザーのホーム・ディレクトリーです。

テンプレートを選択するには、Microsoft Visual Studio を停止してから再始動する必要があります。

このタスクについて

XMS .NET プロジェクト・テンプレートには、アプリケーションの開発を支援するために使用できる一般的なコードが含まれています。 組み込みコードを使用すると、組み込みコード内のプロパティーを変更するだけで IBM MQ キュー・マネージャーに接続し、書き込みや取得の操作を実行できます。

手順

1. Microsoft Visual Studio を開きます。
2. 「**ファイル**」をクリックしてから「**新規作成**」をクリックし、次に「**プロジェクト**」をクリックします。
3. 「新規プロジェクトの作成」ウィンドウで、 IBM XMS .NET Client App (.NET Core) を選択し、「**次へ**」をクリックします。
4. 「新しいプロジェクトの構成」ウィンドウで、必要に応じてプロジェクトの「プロジェクト名」を変更し、「**作成**」をクリックして XMS .NET プロジェクトを作成します。

   XMSDotnetApp.cs は、プロジェクト・ファイルとともに作成されるファイルです。 このファイルには、キュー・マネージャーに接続し、送信操作および受信操作を実行するコードが含まれています。

   接続プロパティーは、次のデフォルト値に設定されます。

   - WMQ_CONNECTION_NAME_LIST は *localhost(1414)* に設定されます
   - XMSC.WMQ_CHANNEL は *DOTNET.SVRCONN* に設定されます

キューは *Q1* に設定され、これらのプロパティーは適宜変更できます。

5. アプリケーションをコンパイルおよび実行します。

関連概念

IBM MQ のコンポーネントと機能

.NET アプリケーション・ランタイム - Windows のみ

スレッド化モデル

マルチスレッド・アプリケーションがどのように XMS オブジェクトを使用できるかは、一般的な規則によって決まります。

- 複数のスレッドで並行して使用できるオブジェクトは以下のタイプのオブジェクトのみです。
  - ConnectionFactory
  - 接続
  - ConnectionMetaData
  - Destination
- Session オブジェクトは、一度に 1 つのスレッドでのみ使用可能です。

これらの規則の例外は、「IBM Message Service Client for .NET reference」のメソッドのインターフェース定義にある "「スレッド・コンテキスト」" というラベルの項目によって示されます。

ConnectionFactories オブジェクトと Connection オブジェクト

ConnectionFactory オブジェクトには、アプリケーションが Connection オブジェクトを作成するときに使用するテンプレートがあります。 アプリケーションは、Connection オブジェクトを使用して Session オブジェクトを作成します。

.NET の場合、XMS アプリケーションは、最初に XMSFactoryFactory オブジェクトを使用して、必要なタイプのプロトコルに適した ConnectionFactory オブジェクトへの参照を取得します。 この結果、この ConnectionFactory オブジェクトは、対象のプロトコル・タイプに対してのみ接続経路を作成できます。

XMS アプリケーションは、複数の接続経路を作成できます。また、マルチスレッド化アプリケーションは、複数のスレッドで 1 つの Connection オブジェクトを並行して使用できます。 Connection オブジェクトは、アプリケーションとメッセージング・サーバー間の通信接続経路をカプセル化します。

接続経路は、以下のように複数の役割を果たします。

- アプリケーションが接続経路を作成した場合は、このアプリケーションを認証できます。
- アプリケーションは、固有のクライアント ID を接続経路に関連付けることができます。 クライアント ID は、パブリッシュ/サブスクライブ・ドメインでの永続サブスクリプションをサポートするときに使用します。 クライアント ID は次の 2 とおりの方法で設定できます。

  接続のクライアント ID を割り当てる方法として望ましいのは、プロパティーを使用してクライアント固有の ConnectionFactory オブジェクトでそれを 構成し、作成する接続にそれを透過的に割り当てるという方法です。

  クライアント ID を割り当てる別の方法は、Connection オブジェクトに対して設定されているプロバイダー固有の値を使用する方法です。 その値によって、管理者により構成されている ID がオーバーライドされることはありません。 それは、管理者によって指定された ID が存在しない場合のために提供されています。 管理者の指定する ID が存在する場合、プロバイダー固有の値によってそれをオーバーライドしようとすると、例外がスローされます。 アプリケーションが明示的に ID を設定する場合は、接続を作成した直後、かつその接続で他のアクションを実行する前に、ID を設定する必要があります。そうでないと、例外がスローされます。

XMS アプリケーションは、通常、1 つの接続経路、1 つ以上のセッション、およびいくつかのメッセージ・プロデューサーおよびメッセージ・コンシューマーを作成します。

接続経路の作成には通信接続経路の確立が必要であり、アプリケーションの認証が必要になる場合もあるため、システム・リソースの点で比較的コストが高くなります。

接続の開始モードと停止モード

接続は、開始モードまたは停止モードで機能します。

アプリケーションが接続経路を作成したとき、この接続経路は停止モードになっています。 接続経路が停止モードになっていると、アプリケーションは、セッションを初期設定してメッセージを送信することはできますが、同期か非同期かにかかわらず、 受信することはできません。

アプリケーションは、`Start Connection` メソッドを呼び出すことにより、接続を開始できます。 接続経路が開始モードになっている場合、アプリケーションはメッセージの送信および受信を実行できます。 その後、アプリケーションは Stop Connection メソッドおよび `Start Connection` メソッドを呼び出すことにより、接続の停止や再開を行うことができます。

接続の終了

アプリケーションは、Close Connection メソッドを呼び出すことにより、接続を終了します。 アプリケーションが接続を終了すると、XMS は以下のアクションを実行します。

- 接続に関連するすべてのセッションを終了して、これらのセッションに関連する特定のオブジェクトを削除します。 どのオブジェクトが削除されるかの詳細については、646 ページの『オブジェクトの削除』を参照してください。 同時に、XMS は、セッション内で現在進行中のすべてのトランザクションをロールバックします。
- メッセージング・サーバーとの通信接続を終了します。
- 接続で使用されていたメモリーやその他の内部リソースを解放します。

XMS は、接続を終了する前に、セッション中に確認できなかったメッセージの受信確認は行いません。 メッセージ受信の確認について詳しくは、637 ページの『メッセージの確認応答』を参照してください。

例外処理

XMS .NET 例外は、すべて System.Exception から派生します。 詳しくは、655 ページの『.NET でのエラー処理』を参照してください。

サービス統合バスへの接続

XMS アプリケーションは、TCP/IP 直接接続を使用するか、HTTP over TCP/IP を使用することにより、WebSphere Application Server サービス統合バスに接続できます。

HTTP プロトコルは、TCP/IP 接続を直接行うことができない状態で使用できます。 一般的な状態の 1 つは、2 つの企業がメッセージを交換する場合など、ファイアウォールを介して通信する場合です。 ファイアウォールを介して通信するために HTTP を使用することを、多くの場合、*HTTP* トンネリングと呼びます。 ただし、HTTP トンネリングは、TCP/IP 接続を直接使用する場合よりも本質的に低速です。これは、HTTP ヘッダーがあると、転送されるデータの量が大幅に増加することや、HTTP プロトコルでは TCP/IP よりも多くの通信フローが必要になることが原因です。

TCP/IP 接続経路を作成するため、アプリケーションは、XMSC_WPM_TARGET_TRANSPORT_CHAIN プロパティーが XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC に設定されている接続ファクトリーを使用できます。 これはプロパティーのデフォルト値です。 接続経路が正常に作成されると、この接続経路の XMSC_WPM_CONNECTION_PROTOCOL プロパティーは XMSC_WPM_CP_TCP に設定されます。

HTTP を使用する接続経路を作成するには、アプリケーションが使用する接続ファクトリーの XMSC_WPM_TARGET_TRANSPORT_CHAIN プロパティーを、HTTP トランスポート・チャネルを使用するよう構成されたインバウンド・トランスポート・チェーンの名前に設定する必要があります。 接続経路が正常に作成されると、この接続経路の XMSC_WPM_CONNECTION_PROTOCOL プロパティーは、XMSC_WPM_CP_HTTP に設定されます。 トランスポート・チェーンの構成方法については、 WebSphere Application Server 製品資料の「 トランスポート・チェーンの構成 」を参照してください。

アプリケーションには、ブートストラップ・サーバーに接続する場合、通信プロトコルについて同様の選択項目があります。 接続ファクトリーの XMSC_WPM_PROVIDER_ENDPOINTS プロパティーは、ブートストラップ・サーバーの 1 つ以上のエンドポイント・アドレスの列です。 各エンドポイント・アドレスのブートストラップ・トランスポート・チェーン・コンポーネントは、ブートストラップ・サーバーへの

TCP/IP 接続の場合は XMSC_WPM_BOOTSTRAP_TCP、 HTTP を使用する接続の場合は XMSC_WPM_BOOTSTRAP_HTTP になります。

セッション数

Session は、メッセージを送受信する単一スレッド化されたコンテキストです。

アプリケーションはセッションを使用して、メッセージ、メッセージ・プロデューサー、メッセージ・コンシューマー、キュー・ブラウザー、および一時宛先を作成できます。 また、アプリケーションはセッションを使用してローカル・トランザクションを実行することもできます。

アプリケーションは複数のセッションを作成できます。この場合、各セッションは、他のセッションとは関係なくメッセージを作成およびコンシュームします。 個別のセッションまたは同一セッション内の 2 つのメッセージ・コンシューマーが同一トピックをサブスクライブする場合、これらの メッセージ・コンシューマーはそれぞれ、そのトピックについて公開されたメッセージのコピーを受信します。

Connection オブジェクトとは異なり、Session オブジェクトを複数のスレッドで並行して使用することはできません。 Session オブジェクトの Close Session メソッドだけを、Session オブジェクトがその時点で使用しているスレッド以外のスレッドから呼び出すことができます。 Close Session メソッドはセッションを終了し、そのセッションに割り振られていたシステム・リソースを解放します。

アプリケーションで複数のスレッドのメッセージを並行処理する必要がある場合、アプリケーションは、スレッドごとにセッションを作成し、そのセッションを、そのスレッド内の送信操作または受信操作に使用する必要があります。

トランザクション化されたセッション

XMS アプリケーションは、ローカル・トランザクションを実行できます。 ローカル・トランザクション とは、 リソースに対する変更を伴うトランザクションのことです。 対象となるリソースは、アプリケーションの接続先となるキュー・マネージャーまたはサービス統合バスのリソースです。

このトピックの情報は、アプリケーションが IBM MQ キュー・マネージャーまたは WebSphere Application Server サービス統合バスに接続する場合にのみ該当します。 この情報は、ブローカーへのリアルタイム接続の場合は該当しません。

ローカル・トランザクションを実行するには、まず、セッションが処理されたことをパラメーターとして指定して Connection オブジェクトの Create Session メソッドを呼び出すことにより、アプリケーションでトランザクション化セッションを作成する必要があります。 その後、そのセッション内で送受信されたすべてのメッセージは、トランザクションの順序でグループ化されます。 トランザクションは、トランザクションが開始されてから送受信したメッセージがアプリケーションでコミットまたはロールバックされると終了します。

トランザクションをコミットするため、アプリケーションは Session オブジェクトの Commit メソッドを呼び出します。 トランザクションがコミットされると、そのトランザクション内に送信されたすべてのメッセージは、他のアプリケーションに配信できるようになります。また、そのトランザクション内に受信したすべてのメッセージが認知されるので、メッセージング・サーバーはそれらのメッセージをアプリケーションへ再配信しなくなります。 また、Point-to-Point ドメインでは、受信したメッセージがメッセージング・サーバーのキューからも除去されます。

トランザクションをロールバックするため、アプリケーションは Session オブジェクトの Rollback メソッドを呼び出します。 トランザクションがロールバックされると、そのトランザクション内に送信されたすべてのメッセージはメッセージング・サーバーによって破棄されます。また、そのトランザクション内に受信したすべてのメッセージは再配信できるようになります。 Point-to-Point ドメインでは、受信されたメッセージはキューに書き戻され、再び他のアプリケーションから見えるようになります。

新規トランザクションは、アプリケーションがトランザクション化セッションを作成するか、Commit または Rollback メソッドを呼び出すと、自動的に開始します。 したがって、トランザクション化されたセッションには常にアクティブなトランザクションが含まれます。

アプリケーションがトランザクション化されたセッションを閉じると、暗黙的なロールバックが行われます。 アプリケーションが接続を閉じると、その接続のトランザクション化されたセッションすべてで暗黙的なロールバックが行われます。

トランザクションは、トランザクション化されたセッションに完全に包含されています。 トランザクションがセッションをまたぐことはできません。 つまり、アプリケーションは、トランザクション化された複数のセッションの中でメッセージを送受信したり、これらのすべてのアクションを単一のトランザクションとしてコミットまたはロールバックしたりすることはできません。

関連概念

メッセージの確認応答
トランザクション化されないすべてのセッションには、アプリケーションによって受信されたメッセージをどのように確認するかを決定する、確認応答モードが存在します。 使用可能な確認応答モードは 3 つあり、どの確認応答モードを選択するかはアプリケーションの設計に影響を与えます。

メッセージの送達
XMS は、永続モードと非永続モードのメッセージ送達、および非同期と同期のメッセージ送達をサポートします。

メッセージの確認応答

トランザクション化されないすべてのセッションには、アプリケーションによって受信されたメッセージをどのように確認するかを決定する、確認応答モードが存在します。 使用可能な確認応答モードは 3 つあり、どの確認応答モードを選択するかはアプリケーションの設計に影響を与えます。

このトピックの情報は、アプリケーションが IBM MQ キュー・マネージャーまたは WebSphere Application Server サービス統合バスに接続する場合にのみ該当します。 この情報は、ブローカーへのリアルタイム接続の場合は該当しません。

XMS が使用しているメッセージ受信を確認する仕組みは、JMS が使用している仕組みと同じです。

セッションがトランザクション化されない場合、アプリケーションが受信するメッセージを確認する方法は、セッションの確認応答モードによって決定されます。 以下の部分では、3 つの確認応答モードについて説明します。

XMSC_AUTO_ACKNOWLEDGE
セッションは、アプリケーションが受信した各メッセージを自動的に確認します。

メッセージがアプリケーションに同期化して配信されると、セッションは、Receive 呼び出しが正常に完了するたびにメッセージの受信を確認します。

アプリケーションがメッセージを正常に受信しても、障害によって確認が行えない場合は、そのメッセージは再び送達可能になります。 このため、アプリケーションは再配信されるメッセージを処理できる必要があります。

XMSC_DUPS_OK_ACKNOWLEDGE
セッションは、メッセージ選択時にアプリケーションが受信したメッセージを確認します。

この確認応答モードを使用すると、セッションで行わなければならない作業の量を減らすことができますが、障害によってメッセージの確認ができなかったときは、複数のメッセージが再び送達可能になる可能性があります。 このため、アプリケーションは再配信される複数のメッセージを処理できる必要があります。

XMSC_CLIENT_ACKNOWLEDGE
Message クラスの Acknowledge メソッドを呼び出すことにより、受信したメッセージをアプリケーションが確認します。

アプリケーションは各メッセージの受信を個々に確認するか、または複数のメッセージを一括して受信し、受信した最後のメッセージに対してのみ Acknowledge メソッドを呼び出すことができます。 Acknowledge メソッドが呼び出されると、このメソッドの前回の呼び出し以降に受信したすべてのメッセージが確認されます。

これらの確認応答モードのいずれかと組み合わせることにより、アプリケーションは Session クラスの Recover メソッドを呼び出してセッションでメッセージの送達を停止したり、再開させたりすることができます。 以前に受信が確認されなかったメッセージは、再配信されます。 ただし、前回送達されたときと同じシーケンスで送達されるとは限りません。 これらのメッセージが再送達されるまでの間に、より優先順位の高いメッセージが届いている可能性もありますし、オリジナルのメッセージの一部が有効期限切れになっている場合もあります。 Point-to-Point ドメインの場合は、オリジナルのメッセージの一部が別のアプリケーションによって消費されている可能性もあります。

アプリケーションでは、メッセージの JMSRedelivered ヘッダー・フィールドの内容を調べることによって、メッセージが再送達中かどうかを確認できます。 アプリケーションでこの確認を行うには、Message クラスの Get JMSRedelivered メソッドを呼び出します。

関連概念

トランザクション化されたセッション

XMS アプリケーションは、ローカル・トランザクションを実行できます。 ローカル・トランザクション とは、 リソースに対する変更を伴うトランザクションのことです。 対象となるリソースは、アプリケーションの接続先となるキュー・マネージャーまたはサービス統合バスのリソースです。

メッセージの送達

XMS は、永続モードと非永続モードのメッセージ送達、および非同期と同期のメッセージ送達をサポートします。

メッセージの送達

XMS は、永続モードと非永続モードのメッセージ送達、および非同期と同期のメッセージ送達をサポートします。

メッセージ送達モード

XMS は、次の 2 種類のメッセージ送達モードをサポートしています。

永続
永続メッセージは、1 回送信されます。 メッセージング・サーバーは、メッセージのロギングなどの特殊な予防措置を取り、障害が発生した場合にも転送中に永続メッセージを失わないようにしています。

非持続
非永続メッセージが送信されるのは 1 回以内です。 非永続メッセージは、障害が発生した場合、転送中に失われる可能性があるため、永続メッセージより信頼性は低くなります。

送達モードの選択は、信頼性とパフォーマンスとのトレードオフになります。 非永続メッセージは、通常、永続メッセージより転送速度が高速になります。

非同期メッセージ配信

XMS は、1 つのスレッドを使用して、あるセッションのすべての非同期メッセージ配信を処理します。 このことは、一度に実行できるのは 1 つのメッセージ・リスナー関数または 1 つの onMessage() メソッドのみであるという意味です。

あるセッションで複数のメッセージ・コンシューマーが複数のメッセージを非同期で受信しており、メッセージ・リスナー関数または onMessage() メソッドが 1 つのメッセージを 1 つのメッセージ・コンシューマーに配信している場合、同じメッセージを待っている他のメッセージ・コンシューマーは引き続き待機する必要があります。 セッションへの配信を待機中のその他のメッセージも、引き続き待機する必要があります。

アプリケーションがメッセージを並行して配信する必要がある場合、XMS が複数のスレッドを使用して非同期メッセージ配信を処理するように、複数のセッションを作成します。 このようにして、複数のメッセージ・リスナー関数または onMessage() メソッドを並行して実行できます。

コンシューマーにメッセージ・リスナーを割り当てても、セッションは非同期にはなりません。 セッションが非同期になるのは、Connection.Start メソッドが呼び出されたときだけです。
Connection.Start メソッドが呼び出されるまでは、すべての同期呼び出しが可能です。
Connection.Start が呼び出されると、コンシューマーへのメッセージ配信が開始されます。

コンシューマーやプロデューサーの作成などの同期呼び出しを非同期セッションで行う必要がある場合は、Connection.Stop を呼び出す必要があります。 Connection.Start メソッドを呼び出してメッセージの配信を開始することにより、セッションを再開できます。 この唯一の例外が、メッセージをコールバック関数に配信するセッション・メッセージ配信スレッドです。 このスレッドは、セッションのメッセージ・コールバック関数でどのような呼び出しでも (クローズ呼び出しを除く) 行えます。

注 : 非管理モードでは、IBM MQ .NET クライアントがコールバック関数内の MQDISC 呼び出しを使用することはできません。 したがって、クライアント・アプリケーションが非同期受信モードの MessageListener

コールバックでセッションを作成したり閉じたりすることはできません。 MessageListener メソッドの外部でセッションを作成して処理します。

同期メッセージ配信

アプリケーションが MessageConsumer オブジェクトの Receive メソッドを使用している場合は、メッセージをアプリケーションに同期した状態で配信します。

Receive メソッドを使用すると、アプリケーションはメッセージを指定の期間または無期限に待機できます。 あるいは、アプリケーションにメッセージを待機させない場合は、Receive with No Wait メソッドを使用できます。

関連概念

トランザクション化されたセッション

XMS アプリケーションは、ローカル・トランザクションを実行できます。 ローカル・トランザクション とは、 リソースに対する変更を伴うトランザクションのことです。 対象となるリソースは、アプリケーションの接続先となるキュー・マネージャーまたはサービス統合バスのリソースです。

メッセージの確認応答

トランザクション化されないすべてのセッションには、アプリケーションによって受信されたメッセージをどのように確認するかを決定する、確認応答モードが存在します。 使用可能な確認応答モードは 3 つあり、どの確認応答モードを選択するかはアプリケーションの設計に影響を与えます。

宛先

XMS アプリケーションは、送信対象メッセージの宛先と受信対象メッセージの送信元を指定するときに Destination オブジェクトを使用します。

XMS アプリケーションは、Destination オブジェクトを実行時に作成することも、管理対象オブジェクトのリポジトリーから事前定義の宛先を取得することもできます。

ConnectionFactory の場合と同様に、XMS アプリケーションで宛先を指定するための最も柔軟な方法は、宛先を管理対象オブジェクトとして定義する方法です。 この方法を使用すると、C、C++、および .NET の各言語で作成したアプリケーションと Java で作成したアプリケーションが、宛先の定義を共用できるようになります。 管理対象の Destination オブジェクトのプロパティーは、コードを変更せずに変更できます。

.NET アプリケーションの場合は、CreateTopic メソッドまたは CreateQueue メソッドを使用して宛先を作成します。 これら 2 つのメソッドは、.NET API の ISession オブジェクトと XMSFactoryFactory オブジェクトの両方で使用できます。 詳細については、653 ページの『.NET での宛先』および../refdev/sapidest.dita#sapidest を参照してください。

トピック *URI*

トピック URI はトピック名を指定します。 また、オプションでトピックのプロパティー (複数可) を指定することもできます。

トピックの URI は topic:// というシーケンスで始まり、この後にトピック名が指定されます。また、他のトピック・プロパティーを設定する名前と値のペアのリストを指定できます。 トピック名を空にすることはできません。

この例を以下の .NET コードのフラグメントに示します。

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

URI に使用できる名前と有効値など、トピックのプロパティーについて詳しくは、宛先のプロパティーを参照してください。

サブスクリプションで使用するトピック URI を指定するときに、ワイルドカードを使用できます。 これらのワイルドカードの構文は、接続タイプとブローカー・バージョンによって異なります。以下のオプションを使用できます。

- WebSphere Application Server サービス統合バス

WebSphere Application Server サービス統合バス

WebSphere Application Server サービス統合バスでは、以下のワイルドカード文字が使用されます。

* (階層内の単一レベルの任意の文字に一致)
// (0 以上のレベルに一致)
//。0 以上のレベル (トピック式の末尾) に一致させるため

640 ページの表 82 に、このワイルドカード方式の使用法の例を示します。

表 *82. WebSphere Application Server* サービス統合バスのワイルドカード方式を使用した *URI* の例

URI	一致するトピック	例
"topic://Sport/*ball/Results"	Sport と Results の間に、「ball」で終わる単一階層レベル名があるすべてのトピック	「topic://Sport/Football/Results」や「topic://Sport/Netball/Results」
"topic://Sport//Results"	「Sport/」で始まり「/Results」で終わるすべてのトピック	「topic://Sport/Football/Results」や「topic://Sport/Hockey/National/Div3/Results」
"topic://Sport/Football//."	「Sport/Football/」で始まるすべてのトピック	「topic://Sport/Football/Results」や「topic://Sport/Football/TeamNews/Signings/Managerial」
"topic://Sport/*ball//Results//."	トピック	「topic://Sport/Football/Results」や「topic://Sport/Netball/National/Div3/Results/2002/November」

関連概念

キュー URI
キューの URI は、キューの名前を指定します。 また、オプションでキューのプロパティー (複数可) を指定することもできます。

一時宛先
XMS アプリケーションは一時宛先を作成および使用できます。

キュー *URI*

キューの URI は、キューの名前を指定します。 また、オプションでキューのプロパティー (複数可) を指定することもできます。

キューの URI は、シーケンス queue://で始まり、その後にキューの名前が続きます。また、残りのキュー・プロパティーを設定する名前と値のペアのリストも含まれる場合があります。

IBM MQ キュー (WebSphere Application Server のデフォルト・メッセージング・プロバイダー・キューを除く) の場合、キューがあるキュー・マネージャーをキューの前に指定できます。指定する場合は、キュー・マネージャー名とキュー名を / で区切ります。

キュー・マネージャーを指定する場合、このマネージャーはこのキューを使用する接続のために XMS が直接接続されているキュー・マネージャーであるか、またはこのキューからアクセスできるキュー・マネージャーでなければなりません。 リモート・キュー・マネージャーは、キューからのメッセージの取り出し操作でのみサポートされており、キューへのメッセージの書き込み操作ではサポートされていません。 詳しくは、IBM MQ キュー・マネージャーの資料を参照してください。

キュー・マネージャーを指定しない場合は、追加の / 分離文字はオプションです。/ 文字の有無によってキューの定義が異なることはありません。

以下のキュー定義はすべて、 XMS が直接接続されている QM_A というキュー・マネージャー上の QB という IBM MQ キューに相当します。

```
queue://QB
queue:///QB
queue://QM_A/QB
```

関連概念

トピック URI
トピック URI はトピック名を指定します。 また、オプションでトピックのプロパティー (複数可) を指定することもできます。

一時宛先
XMS アプリケーションは一時宛先を作成および使用できます。

一時宛先

XMS アプリケーションは一時宛先を作成および使用できます。

一般にアプリケーションは、一時宛先を使用して要求メッセージに対する応答を受信します。 要求メッセージに対する応答の送信先となる宛先を指定するため、アプリケーションは要求メッセージを表す Message オブジェクトの Set JMSReplyTo メソッドを呼び出します。 呼び出しに宛先として一時宛先を指定できます。

セッションを使用して一時宛先が作成されますが、一時宛先の実際の有効範囲は、セッションの作成に使用された接続になります。 接続のどのセッションでも、一時宛先のメッセージ・プロデューサーとメッセージ・コンシューマーを作成できます。 一時宛先は、明示的に削除されるまで、あるいは接続が終了するまで存続します。

アプリケーションが一時キューを作成する場合、キューはアプリケーションの接続先メッセージング・サーバーに作成されます。 アプリケーションがキュー・マネージャーに接続されている場合、XMSC_WMQ_TEMPORARY_MODEL プロパティーによって名前が指定されているモデル・キューから動的キューが作成され、動的キューの名前を形成するために使用される接頭部が XMSC_WMQ_TEMP_Q_PREFIX プロパティーによって指定されます。 アプリケーションがサービス統合バスに接続されている場合、一時キューがバス内に作成され、一時キューの名前を形成するために使用される接頭部が XMSC_WPM_TEMP_Q_PREFIX プロパティーによって指定されます。

サービス統合バスに接続されているアプリケーションが一時トピックを作成する場合、一時トピックの名前を形成するために使用される接頭部は、 XMSC_WPM_TEMP_TOPIC_PREFIX プロパティーによって指定されます。

関連概念

トピック URI
トピック URI はトピック名を指定します。 また、オプションでトピックのプロパティー (複数可) を指定することもできます。

キュー URI
キューの URI は、キューの名前を指定します。 また、オプションでキューのプロパティー (複数可) を指定することもできます。

メッセージ・プロデューサー

XMS では、有効な宛先が指定された状態、または関連した宛先のない状態のいずれかで、メッセージ・プロデューサーを作成することができます。 宛先のない状態でメッセージ・プロデューサーを作成する場合は、メッセージの送信時に有効な宛先を指定する必要があります。

関連した宛先のあるメッセージ・プロデューサー

このシナリオでは、メッセージ・プロデューサーを、有効な宛先を使用して作成します。 送信操作の際に宛先を指定する必要はありません。

関連した宛先のないメッセージ・プロデューサー

XMS .NET では、宛先のない状態でメッセージ・プロデューサーを作成できます。

.NET API の使用時に宛先が関連付けられていないメッセージ・プロデューサーを作成するには、ISession オブジェクトの `CreateProducer()` メソッド (`session.CreateProducer(null)`など) にパラメーターとして NULL を渡す必要があります。 ただし、メッセージの送信時には有効な宛先を指定する必要があります。

メッセージ・コンシューマー

メッセージ・コンシューマーは、永続サブスクライバーと非永続サブスクライバー、および同期メッセージ・コンシューマーと非同期メッセージ・コンシューマーに分類することができます。

永続サブスクライバー

永続サブスクライバーは、特定のトピックに関してパブリッシュされたすべてのメッセージ (サブスクライバーが非アクティブだったときにパブリッシュされたメッセージも含む) を受信するメッセージ・コンシューマーです。

このトピックの情報は、アプリケーションが IBM MQ キュー・マネージャーまたは WebSphere Application Server サービス統合バスに接続する場合にのみ該当します。 この情報は、ブローカーへのリアルタイム接続の場合は該当しません。

トピックの永続サブスクライバーを作成するため、アプリケーションは永続サブスクリプションを示す名前とトピックを表す Destination オブジェクトをパラメーターとして指定して、Session オブジェクトの Create Durable Subscriber メソッドを呼び出します。 アプリケーションはメッセージ・セレクターを備えた永続サブスクライバーまたはメッセージ・セレクターがない永続サブスクライバーを作成できます。また、永続サブスクライバーがサブスクライバー自体の接続により公開されたメッセージを受信するかどうかを指定できます。

永続サブスクライバーの作成に使用されるセッションには、クライアント ID が関連付けられている必要があります。 このクライアント ID は、セッション作成時に使用された接続に関連付けられていたクライアント ID と同一です。それは、634 ページの『ConnectionFactories オブジェクトと Connection オブジェクト』の記述に従って指定されます。

永続サブスクリプションを示す名前はクライアント ID において固有でなければなりません。したがって、クライアント ID は永続サブスクリプションの完全な固有 ID の一部を構成します。 メッセージング・サーバーは永続サブスクリプションのレコードを維持しており、トピックについて公開されたすべてのメッセージが、永続サブスクライバーにより確認されるか、または有効期限が切れるまで保存されるようにします。

永続サブスクライバーが閉じた後も、メッセージング・サーバーは引き続き永続サブスクリプションのレコードを維持します。 以前に作成された永続サブスクリプションを再利用するには、アプリケーションが同じサブスクリプション名を指定し、永続サブスクリプションに関連付けられていたものと同じクライアント ID のセッションを使用して、永続サブスクライバーを作成する必要があります。 一度に 1 つのセッションだけが、特定の永続サブスクリプションの永続サブスクライバーを維持できます。

永続サブスクリプションの有効範囲は、サブスクリプションのレコードを維持するメッセージング・サーバーです。 2 つのアプリケーションがそれぞれ異なるメッセージング・サーバーに接続しており、各アプリケーションが同じサブスクリプション名とクライアント ID を使用して永続サブスクライバーを作成すると、完全に独立した 2 つの永続サブスクリプションが作成されます。

永続サブスクリプションを削除する場合、アプリケーションは永続サブスクリプションを示す名前をパラメーターとして指定し、Session オブジェクトの Unsubscribe メソッドを呼び出します。 セッションに関連付けられているクライアント ID は、永続サブスクリプションに関連付けられているクライアント ID と同一でなければなりません。 メッセージング・サーバーは 保守している永続サブスクリプションのレコードを削除し、永続サブスクライバーにこれ以降メッセージを送信 しなくなります。

既存のサブスクリプションを変更する場合、アプリケーションは同一サブスクリプション名とクライアント ID を使用し、異なるトピックまたはメッセージ・セレクター (あるいはこの両方) を指定して永続サブスクライバーを作成できます。 永続サブスクリプションの変更は、サブスクリプションを削除してから新規サブスクリプションを作成する操作と同等です。

IBM MQ キュー・マネージャーに接続するアプリケーションの場合、 XMS がサブスクライバー・キューを管理します。 そのため、アプリケーションでは、サブスクライバー・キューを指定する必要はありません。指定すると、XMS は、サブスクライバー・キューを無視します。

永続サブスクリプションのサブスクライバー・キューは変更できない点に注意してください。 サブスクライバー・キューを変更する必要がある場合は、サブスクリプションを削除してから新規作成する方法でのみ変更できます。

サービス統合バスに接続するアプリケーションの場合、各永続サブスクライバーには永続サブスクリプション・ホームが指定されている必要があります。 同じ接続を使用するすべての永続サブスクライバーの永続サブスクリプション・ホームを指定するには、その接続の作成に使用される ConnectionFactory オブジェクトの XMSC_WPM_DUR_SUB_HOME プロパティーを設定します。 個別のトピックの永続サブスクリプション・ホームを指定するには、 トピックを表す Destination オブジェクトの XMSC_WPM_DUR_SUB_HOME プロパティーを設定します。 接続を使用する永続サブスクライバーをアプリケーションが作成する前に、その接続の永続サブスクリプション・ホームを指定する必要があります。 宛先に対して指定されている値は、接続に対して指定されている値に優先します。

同期および非同期メッセージ・コンシューマー

同期メッセージ・コンシューマーはキューから同期でメッセージを受信し、非同期メッセージ・コンシューマーは、キューから非同期でメッセージを受信します。

同期メッセージ・コンシューマー

同期メッセージ・コンシューマーは、メッセージを 1 つずつ受信します。 `Receive(wait interval)` メソッドが使用される場合、呼び出しは、指定された期間 (ミリ秒単位) だけメッセージを待機するか、メッセージ・コンシューマーがクローズされるまで待機します。

ReceiveNoWait() メソッドが使用された場合、同期メッセージ・コンシューマーは、遅延なしでメッセージを受信します。次のメッセージが使用可能になると、直ちにそのメッセージを受信します。それ以外の場合は、ヌルの Message オブジェクトへのポインターが返されます。

非同期メッセージ・コンシューマー

キューで新しいメッセージが使用可能になるたびに、アプリケーションによって登録されたメッセージ・リスナーが呼び出されます。

XMS の有害メッセージ

有害メッセージは、受信側 MDB アプリケーションで処理できないメッセージのことです。 有害メッセージが見つかると、XMS MessageConsumer オブジェクトは、BOQUEUE および BOTHRESH の 2 つのキュー・プロパティーに従って、そのメッセージを再キューイングできます。

ある環境では、MDB に送達されたメッセージが IBM MQ キューにロールバックされることがあります。 これは、例えば、メッセージが 1 つの作業単位の中で配信され、その後、その作業単位がロールバックされた場合に発生する可能性があります。 通常、ロールバックされたメッセージは再配信されますが、メッセージのフォーマットが正しくないと、MDB が繰り返し失敗し、したがってメッセージを配信できなくなります。 こうしたメッセージは有害メッセージと呼ばれます。 こうした有害メッセージを後で調査するために別のキューに自動的に転送したり、廃棄したりするように、IBM MQ を構成することができます。 このように IBM MQ を構成する方法について詳しくは、ASF での有害メッセージの処理を参照してください。

時折、不適切なフォーマットのメッセージがキューに到着する場合があります。 ここで言う「不適切なフォーマット」とは、受信側のアプリケーションがメッセージを正しく処理できないことを意味します。 このようなメッセージがあると、受信側のアプリケーションに障害が発生したり、この不適切なフォーマットのメッセージがアプリケーションによってバックアウトされたりすることがあります。 そして、メッセージは繰り返し入力キューに送達され、アプリケーションによって繰り返しバックアウトされる可能性があります。 これらのメッセージを、有害メッセージと呼びます。 XMS MessageConsumer オブジェクトは、有害メッセージを検出して、代替宛先にそれらを再転送します。

IBM MQ キュー・マネージャーは、それぞれのメッセージがバックアウトされた回数を記録しています。 この回数が、構成可能なしきい値に達すると、メッセージ・コンシューマーはそのメッセージを名前付きのバックアウト・キューに再キューイングします。 この再キューイングが何らかの理由により失敗すると、メッセージは入力キューから除去され、送達不能キューに再キューイングされるか、または廃棄されます。

XMS ConnectionConsumer オブジェクトは、同じ方法で、同じキュー・プロパティーを使用して、有害メッセージを処理します。 複数の接続コンシューマーが同じキューをモニターしている場合は、リキューが行われるしきい値の回数を超えても有害メッセージがアプリケーションに送達されることがあります。 この動作の原因は、接続コンシューマーが個別にキューをモニターして有害メッセージを再キューイングする方法にあります。

しきい値およびバックアウト・キューの名前は、IBM MQ キューの属性です。 これらの属性の名前は、BackoutThreshold および BackoutRequeueQName です。 これらの属性が適用されるキューは、以下のとおりです。

- Point-to-Point メッセージングの場合、これは基礎ローカル・キューです。 これは、メッセージ・コンシューマーおよび接続コンシューマーがキューの別名を使用する場合に重要です。
- IBM MQ メッセージング・プロバイダーの通常モードにおけるパブリッシュ/サブスクライブ・メッセージングの場合、これは、Topic の管理キューの作成元であるモデル・キューです。
- IBM MQ メッセージング・プロバイダーのマイグレーション・モードにおけるパブリッシュ/サブスクライブ・メッセージングの場合、これは、TopicConnectionFactory オブジェクトで定義された CCSUB キュー、または Topic オブジェクトで定義された CCDSUB キューです。

BackoutThreshold 属性および BackoutRequeueQName 属性を設定するには、次の MQSC コマンドを実行します。

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

パブリッシュ/サブスクライブ・メッセージングの場合、システムがサブスクリプションごとに動的キューを作成すると、これらの属性値は IBM MQ classes for JMS モデル・キュー SYSTEM.JMS.MODEL.QUEUE。これらの設定を変更するには、以下を使用できます。

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

バックアウトのしきい値がゼロの場合、ポイズン・メッセージの処理は不可となり、ポイズン・メッセージは入力キュー上に残ります。 それ以外の場合は、バックアウト・カウントがしきい値に達すると、メッセージは指定されたバックアウト・キューに送信されます。

バックアウト・カウントがしきい値に達してもメッセージをバックアウト・キューに入れることができないときは、メッセージは送達不能キューに送信されるか、メッセージが非永続の場合は廃棄されます。

この状況は、バックアウト・キューが定義されていない場合、または MessageConsumer オブジェクトがメッセージをバックアウト・キューに送信できない場合に発生します。

有害メッセージ処理を実行するためのシステムの構成

XMS .NET が **BOTHRESH** 属性と **BOQNAME** 属性の照会時に使用するキューは、実行しているメッセージングのスタイルによって次のように異なります。

- Point-to-Point メッセージングの場合、これは基礎ローカル・キューです。 これは、XMS .NET アプリケーションが別名キューとクラスター・キューのいずれかのメッセージをコンシュームしているときに重要になります。
- パブリッシュ/サブスクライブ・メッセージングの場合は、アプリケーションのメッセージを保持するための管理対象キューが作成されます。 XMS .NET は管理対象キューを照会して、**BOTHRESH** 属性と **BOQNAME** 属性の値を判別します。

  管理対象キューは、アプリケーションのサブスクライブ先の Topic オブジェクトに関連付けられているモデル・キューから作成され、モデル・キューから **BOTHRESH** 属性と **BOQNAME** 属性の値を継承します。使用されるモデル・キューは、受信側アプリケーションが永続サブスクリプションと非永続サブスクリプションのどちらを取得したかによって異なります。

  – 永続サブスクリプションに使用されるモデル・キューは、Topic の **MDURMDL** 属性によって指定されます。 この属性のデフォルト値は SYSTEM.DURABLE.MODEL.QUEUE です。
  – 非永続サブスクリプションの場合、使用されるモデル・キューは **MNDURMDL** 属性によって指定されます。 **MNDURMDL** 属性のデフォルト値は SYSTEM.NDURABLE.MODEL.QUEUE です。

BOTHRESH 属性および **BOQNAME** 属性を照会すると、XMS .NET によって次の処理が行われます。

- ローカル・キュー、または別名キューのターゲット・キューを開きます。

- **BOTHRESH** 属性と **BOQNAME** 属性を照会します。
- ローカル・キュー、または別名キューのターゲット・キューを閉じます。

ローカル・キュー、または別名キューのターゲット・キューを開くときに使用されるオープン・オプションは、使用されている IBM MQ のバージョンによって異なります。

- IBM MQ 9.1.0 Fix Pack 4 Long Term Support 以前および IBM MQ 9.1.4 Continuous Delivery 以前の場合、ローカル・キュー、または別名キューのターゲット・キューがクラスター・キューであると、XMS .NET によって、MQOO_INPUT_AS_Q_DEF、MQOO_INQUIRE、および MQOO_FAIL_IF_QUIESCING のオプションを使用してキューが開かれます。 これは、受信側アプリケーションを実行しているユーザーが、クラスター・キューのローカル・インスタンスに対する「照会および取得」アクセス権を持っていなければならないことを意味します。

  XMS .NET は、他のすべてのタイプのローカル・キューを、オープン・オプション MQOO_INQUIRE および MQOO_FAIL_IF_QUIESCING を使用して開きます。 XMS .NET が属性の値を照会するためには、受信側アプリケーションを実行しているユーザーがローカル・キューに対する照会アクセス権を持っていなければなりません。
- IBM MQ 9.1.5 および IBM MQ 9.1.0 Fix Pack 5 から XMS .NET を使用する場合、受信側アプリケーションを実行するユーザーは、キューのタイプに関係なく、ローカル・キューに対する照会アクセス権限を持っている必要があります。

有害メッセージをバックアウト・リキュー・キューまたはキュー・マネージャーの送達不能キューに移動するには、アプリケーションを実行するユーザーに put および passall 権限を付与する必要があります。

ASF での有害メッセージの処理
Application Server Facilities (ASF) を使用する場合は、MessageConsumer の代わりに ConnectionConsumer で有害メッセージを処理します。 ConnectionConsumer は、キューの BackoutThreshold および BackoutRequeueQName プロパティーに従ってメッセージをリキューします。

アプリケーションが ConnectionConsumers を使用している場合、メッセージがバックアウトされる状況は、アプリケーション・サーバーが提供するセッションによって異なります。

- セッションが非トランザクション化セッションで、AUTO_ACKNOWLEDGE または DUPS_OK_ACKNOWLEDGE が指定されている場合、メッセージがバックアウトされるのは、システム・エラーの後、またはアプリケーションが予期せずに終了した場合のみです。
- セッションが非トランザクション化セッションで、CLIENT_ACKNOWLEDGE が指定されている場合、無応答メッセージは、アプリケーション・サーバーが Session.recover() を呼び出すことによってバックアウトすることができます。

  一般に、MessageListener のクライアント実装またはアプリケーション・サーバーが Message.acknowledge() を呼び出します。 Message.acknowledge() は、これまでそのセッションで配信されたすべてのメッセージに応答します。
- セッションがトランザクション化セッションの場合、無応答メッセージは、アプリケーション・サーバーが Session.rollback() を呼び出すことによってバックアウトすることができます。

キュー・ブラウザー

アプリケーションは、キュー・ブラウザーを使用して、キュー上のメッセージを参照します。 その際にメッセージは除去されません。

キュー・ブラウザーを作成するには、アプリケーションは、ISession オブジェクトの Create Queue Browser メソッドを呼び出し、参照するキューを示す Destination オブジェクトをパラメーターとして指定します。 アプリケーションはメッセージ・セレクターを備えたキュー・ブラウザーまたはメッセージ・セレクターのないキュー・ブラウザーを作成できます。

キュー・ブラウザーの作成後、アプリケーションは、IQueueBrowser オブジェクトの GetEnumerator メソッドを呼び出して、キューにあるメッセージのリストを取得できます。 このメソッドは、Message オブジェクトのリストをカプセル化する列挙子を戻します。 リスト内の Message オブジェクトの順序は、メッセージがキューから取り出される順序と同じ です。 アプリケーションは列挙子を使用して、各メッセージを順番に参照できます。

メッセージがキューに書き込まれたり削除されたりすると、列挙子は動的に更新されます。 アプリケーションが IEnumerator.MoveNext() を呼び出してキュー上の次のメッセージを参照するたびに、そのメッセージにはキューの現在の内容が反映されます。

アプリケーションは、特定のキュー・ブラウザーに対して GetEnumerator メソッドを複数回呼び出すことができます。 呼び出しごとに、新しい列挙子が返されます。 したがって、アプリケーションは複数の列挙子を使用してキューのメッセージを参照し、キュー内の複数の位置を維持することができます。

アプリケーションはキュー・ブラウザーを使用して、キューから除去する適切なメッセージを検索し、メッセージ・セレクターを備えたメッセージ・コンシューマーを使用してメッセージを除去することができます。 メッセージ・セレクターは、JMSMessageID ヘッダー・フィールドの値に基づいてメッセージを選択できます。 このフィールドをはじめとする JMS メッセージ・ヘッダーのフィールドについては、668 ページの『XMS メッセージのヘッダー・フィールド』を参照してください。

リクエスター

アプリケーションはリクエスターを使用して要求メッセージを送信し、応答を待機して受信します。

多くのメッセージング・アプリケーションは、要求メッセージを送信して応答を待機するアルゴリズムに基づいています。 XMS の Requestor というクラスは、このようなスタイルのアプリケーションを開発するときに役立ちます。

リクエスターを作成するため、アプリケーションは Requestor クラスの Create Requestor コンストラクターを呼び出します。このとき、Session オブジェクトと、要求メッセージの送信先を示す Destination オブジェクトがパラメーターとして指定されます。 セッションがトランザクション化されていてはなりません。また、セッションには肯定応答モード XMSC_CLIENT_ACKNOWLEDGE は設定できません。 コンストラクターは、応答メッセージの送信先となる一時キューまたは一時トピックを自動的に作成します。

アプリケーションは、リクエスターの作成後に、Requestor オブジェクトの Request メソッドを呼び出して要求メッセージを送信し、 要求メッセージを受信したアプリケーションからの応答を待機して受信することができます。 応答の受信またはセッションの終了のいずれかが発生するまで、呼び出しは待機します。リクエスターが必要とする応答は、要求メッセージごとに 1 つのみです。

アプリケーションがリクエスターを閉じると、一時キューまたは一時トピックは削除されます。 ただし関連付けられているセッションは閉じません。

オブジェクトの削除

アプリケーションが、作成した XMS オブジェクトを削除すると、XMS は、このオブジェクトに割り振られていた内部リソースを解放します。

アプリケーションが XMS オブジェクトを作成すると、XMS はこのオブジェクトにメモリーやその他の内部リソースを割り振ります。 XMS は、XMS が内部リソースを解放した時点で、アプリケーションがオブジェクトの close メソッドまたは delete メソッドを呼び出して、明示的にオブジェクトを削除するまで、内部リソースを保存します。 アプリケーションが、既に削除されているオブジェクトを削除しようとすると、呼び出しは無視されます。

アプリケーションが Connection オブジェクトまたは Session オブジェクトを削除すると、XMS は特定の関連オブジェクトを自動的に削除し、その内部リソースを解放します。 これらは Connection オブジェクトまたは Session オブジェクトによって作成されたオブジェクトで、これらのオブジェクトから独立した機能を持っていません。 これらのオブジェクトを 646 ページの表 83 に示します。

注 : アプリケーションが従属セッションとの接続を終了すると、これらのセッションに従属するすべてのオブジェクトも削除されます。 従属オブジェクトがあるのは、Connection オブジェクトまたは Session オブジェクトのみです。

表 *83.* 自動的に削除されるオブジェクト

削除されたオブジェクト	**メソッド**	**自動的に削除される従属オブジェクト**
接続	Close Connection	ConnectionMetaData オブジェクトおよび Session オブジェクト

表 83. 自動的に削除されるオブジェクト (続き)		
削除されたオブジェクト	**メソッド**	**自動的に削除される従属オブジェクト**
Session	Close Session	MessageConsumer、MessageProducer、QueueBrowser、Requestor の各オブジェクト

XMS による管理 IBM MQ XA トランザクション

管理対象 IBM MQ XA トランザクションは、 XMS を介して使用できます。

XMS で XA トランザクションを使用するには、トランザクション化セッションを作成する必要があります。 XA トランザクションを使用する場合は、XMS セッションではなく、分散トランザクション・コーディネーター (DTC) のグローバル・トランザクションによって、トランザクションを制御します。 XA トランザクションを使用する場合、 XMS セッションで `Session.commit` または `Session.rollback` を発行することはできません。 その代わりに、DTC メソッド `Transscope.Commit` または `Transscope.Rollback` を使用して、トランザクションのコミットまたはロールバックを実行します。 XA トランザクションのためにセッションを使用している場合、そのセッションを使用して作成するプロデューサーまたはコンシューマーは、その XA トランザクションの一部でなければなりません。 そのプロデューサーまたはコンシューマーを XA トランザクションのスコープ外の操作のために使用することはできません。 つまり、XA トランザクションの外で `Producer.send` や `Consumer.receive` のような操作を実行するために使用することはできないということです。

以下の場合は、`IllegalStateException` 例外オブジェクトがスローされます。

- XA トランザクション化セッションを `Session.commit` または `Session.rollback` のために使用する場合。
- XA トランザクション化セッションで使用したことのあるプロデューサー・オブジェクトまたはコンシューマー・オブジェクトを XA トランザクションのスコープ外で使用する場合。

非同期コンシューマーでは、XA トランザクションはサポートされていません。

注 :

1. XA トランザクションがコミットされる前に、`Producer`、`Consumer`、`Session`、`Connection` のいずれかのオブジェクトを閉じる操作が実行される場合もあります。 そのような場合は、トランザクションに含まれているメッセージがロールバックされます。 さらに、XA トランザクションがコミットされる前に接続で障害が発生した場合も、トランザクションに含まれているすべてのメッセージがロールバックされます。 `Producer` オブジェクトの場合のロールバックは、メッセージがキューに書き込まれないという意味です。 `Consumer` オブジェクトの場合のロールバックは、メッセージがキューに残されるという意味です。
2. `Producer` オブジェクトが `TimeToLive` を指定してメッセージを `TransactionScope` に入れた場合は、その存続時間の経過後に `commit` を実行しても、`commit` の実行前にメッセージの有効期限が切れた状態になります。 その場合、`Consumer` オブジェクトがそのメッセージを受け取ることはできません。
3. `Session` オブジェクトを複数のスレッドで使用することはサポートされていません。 複数のスレッドで `Session` オブジェクトを共用するトランザクションの使用もサポートされていません。

XMS プリミティブ型

XMS には、Java の 8 個のプリミティブ型 (byte、short、int、long、float、double、char、および boolean) に相当するデータ型が用意されています。 これにより、XMS と JMS の間で、データの損失や破損が発生することなくメッセージを交換することができます。

648 ページの表 84 は、 Java 同等のデータ型、サイズ、および各 XMS プリミティブ型の最小値と最大値をリストしています。

表 84. XMS のデータ型とそれに対応する Java の型

XMS データ型	互換性のある Java データ型	サイズ	最小値	最大値
System.Boolean	boolean	32 ビット	false	true
System.SBYTE	byte	8 ビット	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 ビット	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 ビット	-2^7 (-128)	2^7-1 (127)
System.Int16	short	16 ビット	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 ビット	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	long	64 ビット	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	float	32 ビット	-3.402823E-38 (精度 7 桁)	3.402823E+38 (精度 7 桁)
System.Double	double	64 ビット	-1.79769313486231E-308 (精度 15 桁)	1.79769313486231E+308 (精度 15 桁)

プロパティー値のデータ型の暗黙的な変換

アプリケーションがプロパティーの値を取得するときに、 XMS によりこの値のデータ型を別のデータ型に変換できます。 多くのルールが、サポートされる変換と、XMS がその変換を実行する方法を規定します。

オブジェクトのプロパティーには名前と値が指定されていて、その値にはデータ型が関連付けられています。 プロパティーの値は、プロパティー・タイプ とも呼ばれます。

アプリケーションは、PropertyContext クラスのメソッドを使用してオブジェクトのプロパティーを取得および設定します。 プロパティー値を取得するため、アプリケーションは一般にプロパティー・タイプに対応したメソッドを呼び出します。 例えば、整数プロパティーの値を取得するには、アプリケーションは一般に GetIntProperty メソッドを呼び出します。

ただし、アプリケーションがプロパティーの値を取得する際に、 XMS によりこの値のデータ型を別のデータ型に変換できます。 例えば、整数プロパティーの値を取得するには、アプリケーションは GetStringProperty メソッドを呼び出すことで、そのプロパティーの値をストリングとして取得することができます。 XMS でサポートされている変換を 648 ページの表 85 に示します。

表 85. サポートされるプロパティー値のデータ型の変換

プロパティー・タイプ	サポートされているターゲット・データ・タイプ
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String

表 85. サポートされるプロパティー値のデータ型の変換 (続き)	
プロパティー・タイプ	**サポートされているターゲット・データ・タイプ**
System.Int16	System.String, System.Int32, System.Int64

以下の汎用ルールは、サポートされる変換を規定します。

- 変換中にデータが失われない限り、数値プロパティー値のデータ型を変換できます。 例えば、データ型 System.Int32 のプロパティーの値を、データ型 System.Int64 の値に変換することはできますが、データ型 System.Int16 の値に変換することはできません。
- どのデータ・タイプのプロパティー値でも、ストリングに変換できる。
- ストリング・プロパティー値は、ストリングが変換用に正しくフォーマットされていれば、任意の他のデータ・タイプに変換できます。 正しい形式になっていない ストリング・プロパティー値をアプリケーションが変換しようとすると、XMS から エラーが戻される場合があります。
- アプリケーションがサポートされていない変換を実行しようとすると、XMS からエラーが戻されます。

プロパティー値のデータ型が変換されるとき、以下のルールが適用されます。

- ブール型プロパティーの値をストリングに変換する場合、値 true はストリング "true"に変換され、値 false はストリング "false"に変換されます。
- ブール・プロパティー値を数値データ型 (System.SByte を含む) に変換すると、値 true は 1 に変換され、値 false は 0 に変換されます。
- ストリング・プロパティー値をブール値に変換する場合、ストリング "true" (大文字と小文字を区別しない) または "1" は true に変換され、ストリング "false" (大文字と小文字を区別しない) または "0" は false に変換されます。 その他のストリングはいずれも変換できません。
- ストリング・プロパティー値を、データ型 System.Int32、System.Int64、System.SByte、または System.Int16 の値に変換する場合、変換するストリングは以下の形式でなければなりません。

  [*blanks*][*sign*]*digits*

  ストリング・コンポーネントは、以下のように定義されます。

  blanks
  オプションの先行ブランク文字。

  sign
  オプションの正符号 (+) または負符号 (-) 文字。

  digits
  数字 (0 から 9 まで) の連続シーケンス。 少なくとも 1 つの数字が存在している必要があります。

  数字のシーケンスの後のストリングには数字以外の文字を含めることができますが、それらの文字の最初のものに達するとすぐに変換は停止します。 ストリングは 10 進整数を表すと想定されます。

  ストリングの形式が正しくないと、XMS からエラーが戻されます。
- ストリング・プロパティー値をデータ型 System.Double または System.Float の値に変換する場合、変換するストリングは以下の形式でなければなりません。

  [*blanks*][*sign*][*digits*][*point*[*d_digits*]][*e_char*[*e_sign*]*e_digits*]

  ストリング・コンポーネントは、以下のように定義されます。

  blanks
  (オプション) 先行ブランク文字。

  sign
  (オプション) 正符号 (+) または負符号 (-) 文字。

  digits
  数字 (0 から 9 まで) の連続シーケンス。 *digits* か *d_digits* のいずれかに、少なくとも 1 つの数字が必要です。

  point
  (オプション) 小数点 (.)。

d_digits
数字 (0 から 9 まで) の連続シーケンス。 *digits* か *d_digits* のいずれかに、少なくとも 1 つの数字が必要です。

e_char
指数文字。*E* または *e* のいずれかです。

e_sign
(オプション) 指数の正符号 (+) または負符号 (-) 文字。

e_digits
指数用の、数字 (0-9) の連続シーケンス。 ストリングに指数文字がある場合、1 つ以上の数字がなければなりません。

数字のシーケンスの後、または指数を表すオプションの文字の後のストリングには数字以外の文字を含めることができますが、それらの文字の最初のものに達するとすぐに変換は停止します。 ストリングは、10 の累乗の指数を持つ 10 進浮動小数点数を表すと想定されます。

ストリングの形式が正しくないと、XMS からエラーが戻されます。

- 数値プロパティー値 (データ型 System.SByte のプロパティー値を含む) をストリングに変換する場合、値は、その値に対応する ASCII 文字を含むストリングではなく、その値を 10 進数として表現するストリングに変換されます。 例えば、整数 65 はストリング "65" に変換され、ストリング "A" に変換されるのではありません。
- バイト配列プロパティー値をストリングに変換する場合、各バイトはバイトを表す 2 つの 16 進文字に変換されます。 例えば、バイト配列 {0xF1, 0x12, 0x00, 0xFF} がストリング "F11200FF" に変換されます。

プロパティー・タイプのデータ型変換は、Property クラスと PropertyContext クラスの両方のメソッドによりサポートされています。

イテレーター

イテレーターは、オブジェクトのリストとこのリストの現在位置を維持するカーソルをカプセル化します。 イテレーターの概念は、IBM MQ Message Service Client (XMS) for C/C++ の場合と同じように、IBM MQ Message Service Client (XMS) for .NET の IEnumerator インターフェースを使用することで実装されます。

イテレーターを作成すると、カーソルの位置は最初のオブジェクトの前になります。 アプリケーションは、イテレーターを使用して各オブジェクトを順番に取得します。

IBM MQ Message Service Client (XMS) for C/C++ の Iterator クラスは、Java の Enumerator クラスに相当します。 IBM MQ Message Service Client (XMS) for .NET は Java に類似し、IEnumerator インターフェースを使用します。

アプリケーションは、IEnumerator を使用して次のタスクを実行できます。

- メッセージのプロパティーを取得する
- マップ・メッセージの本文で名前と値の対を取得する
- キューに入っているメッセージを参照する
- 接続でサポートされている JMS 定義のメッセージ・プロパティーの名前を取得する

コード化文字セット ID

XMS .NET では、すべてのストリングがネイティブの .NET ストリングを使用して渡されます。 この場合のエンコード方式は固定であるため、解釈するためにこれ以上の情報は不要です。 そのため、XMS .NET アプリケーションに XMSC_CLIENT_CCSID プロパティーは必要ありません。

XMS エラーおよび例外コード

XMS では、障害を示す一定の範囲のエラー・コードが使用されています。 エラー・コードはリリースごとに異なる可能性があるため、本書では明示的には示していません。 本書で示されているのは XMS 例外コード (XMS_X_... の形式) だけです。 これは、このコードが XMS のリリース間で同一であるためです。

XMS を使用した自動 IBM MQ クライアント再接続

IBM WebSphere MQ 7.1 以降のクライアントをメッセージング・プロバイダーとして使用しているときに、ネットワーク、キュー・マネージャー、またはサーバーの障害の後に自動的に再接続するように XMS クライアントを構成します。

MQConnectionFactory クラスの WMQ_CONNECTION_NAME_LIST プロパティーと WMQ_CLIENT_RECONNECT_OPTIONS プロパティーを使用して、クライアント接続が自動的に再接続するように構成できます。 自動クライアント再接続では、接続で障害が発生した後にクライアントが再接続します。あるいは、オプションとしてキュー・マネージャーの停止後に自動クライアント再接続を利用することもできます。 クライアント・アプリケーションの設計によっては、自動再接続に適さないクライアント・アプリケーションもあります。

自動再接続が可能なクライアント接続は、接続が確立した時点で再接続が可能な状態になります。

注 : クライアント再接続オプション、クライアント再接続タイムアウト、および接続名前リストといったプロパティーは、クライアント・チャネル定義テーブル (CCDT) を使用して設定することも、mqclient.ini ファイルを介してクライアントの再接続を有効にすることによって設定することもできます。

注 : ConnectionFactory オブジェクトと CCDT の両方で再接続プロパティーを設定した場合の優先順位に関する規則は、以下のようになります。 ConnectionFactory オブジェクトで接続名リスト・プロパティーのデフォルト値が設定されている場合は、CCDT が優先されます。 接続名リストがデフォルト値に設定されていない場合は、ConnectionFactory オブジェクトで設定されているプロパティー値が優先されます。 接続名リストのデフォルト値は localhost(1414) です。

XMS .NET アプリケーションの作成

このセクションのトピックでは、XMS .NET アプリケーションを作成する場合に役立つ情報を記載します。

このタスクについて

このセクションでは、XMS .NET アプリケーションを作成する場合に固有の情報を記載します。 XMS アプリケーションを作成する場合の一般情報については、632 ページの『XMS アプリケーションの作成』を参照してください。

このセクションでは、以下のトピックについて説明します。

- 651 ページの『.NET のデータ型』
- 652 ページの『.NET における管理操作および非管理操作』
- 653 ページの『.NET での宛先』
- 653 ページの『.NET でのプロパティー』
- 654 ページの『.NET での存在しないプロパティーの処理』
- 655 ページの『.NET でのエラー処理』
- 655 ページの『.NET でのメッセージ・リスナーおよび例外リスナーの使用法』

.NET のデータ型

XMS .NET は、System.Boolean、System.Byte、System.SByte、System.Char、System.String、System.Single、System.Double、System.Decimal、System.Int16、System.Int32、System.Int64、System.UInt16、System.UInt32、System.UInt64、および System.Object をサポートしています。 XMS .NET のデータ・タイプは、XMS C/C++のデータ・タイプとは異なります。このトピックを使用して、対応するデータ・タイプを識別することができます。

下の表に、対応する XMS .NET と XMS C/C++ データ型を示し、簡単に説明します。

表 86. *XMS .NET* と *XMS C/C++* のデータ型

XMS .NET タイプ	**XMS C/C++ 型**	**説明**
System.SByte	xmsSBYTE xmsINT8	符号付き 8 ビット値
System.Byte	xmsBYTE xmsUINT8	符号なし 8 ビット値
System.Int16	xmsINT16 xmsSHORT	符号付き 16 ビット値
System.UInt16	xmsUINT16 xmsUSHORT	符号なし 16 ビット値
System.Int32	xmsINT32 xmsINT	符号付き 32 ビット値
System.UInt32	xmsUINT32 xmsUINT	符号なし 32 ビット値
System.Int64	xmsLONG xmsINT64	符号付き 64 ビット値
System.UInt64	xmsULONG xmsUINT64	符号なし 64 ビット値
System.Char	xmsCHAR16	符号なし 16 ビット文字 (.NET の場合は Unicode)
System.Single	xmsFLOAT	IEEE 32 ビット浮動小数点
System.Double	xmsDOUBLE	IEEE 64 ビット浮動小数点
System.Boolean	xmsBOOL	True/False (真/偽) の値
適用外	xmsCHAR	符号付きまたは符号なし 8 ビット値 (符号付きか符号なしかはプラットフォームによる)
System.Decimal	適用外	96 ビット符号付き整数× 10^0 から 10^{28} まで
System.Object	適用外	すべての型の基本
System.String	適用外	ストリング型

.NET における管理操作および非管理操作

管理コードは、.NET 共通言語ランタイム環境の中で排他的に実行され、そのランタイムによって提供されるサービスに完全に依存します。 アプリケーションが非管理に分類されるのは、そのアプリケーションの一部分が .NET 共通言語ランタイム環境の外部で実行されるか、または 外部にあるサービスを呼び出す場合です。

拡張機能の中には、現在のところ管理 .NET 環境の中ではサポートできないものがあります。

完全管理環境で現在サポートされていない機能をアプリケーションで使用する必要がある場合は、そのアプリケーションに実質的な変更を加えることなく、そのアプリケーションが非管理環境を使用するように

変更することができます。 ただし、その場合は XMS スタックで非管理コードが使用されることに注意してください。

IBM MQ キュー・マネージャーへの接続

WMQ_CM_CLIENT との管理接続では、非 TCP 通信およびチャネル圧縮はサポートされていません。 しかし、それらの接続は、非管理接続 (WMQ_CM_CLIENT_UNMANAGED) を使用することによってサポートされる場合があります。 詳細については、553 ページの『.NET アプリケーションの開発』を参照してください。

非管理環境で管理対象オブジェクトから接続ファクトリーを作成する場合、接続モードの値を手動で XMSC_WMQ_CM_CLIENT_UNMANAGED に変更する必要があります。

WebSphere Application Server サービス統合バス・メッセージング・エンジンとの接続

SSL プロトコル (HTTPS を含む) を使用することが必要な WebSphere Application Server サービス統合バス・メッセージング・エンジンとの接続は、現在のところ管理コードとしてサポートされていません。

.NET での宛先

.NET では、宛先はプロトコル・タイプに従って作成されるため、作成対象のプロトコル・タイプでのみ使用できます。

宛先を作成するための 2 つの関数が用意されており、1 つはトピック用、もう 1 つはキュー用です。

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

これらの関数は、API にある次の 2 つのオブジェクトで使用できます。

- ISession
- XMSFactoryFactory

どちらの場合も、これらのメソッドは、以下のフォーマットでパラメーターを含めることができる URI スタイル・ストリングを受け入れることができます。

```
"topic://some/topic/name?priority=5"
```

あるいは、これらのメソッドは宛先名のみ (つまり、topic:// または queue:// のプレフィックスとパラメーターのない名前) を解釈することもできます。

したがって、URI スタイルのストリング

```
CreateTopic("topic://some/topic/name");
```

は、次の宛先名と同じ結果になります。

```
CreateTopic("some/topic/name");
```

WebSphere Application Server サービス統合バス JMS においては、簡略形式でトピックを指定することもできます。つまり、次のように *topicname* と *topicspace* の両方を指定しますが、パラメーターは指定できません。

```
CreateTopic("topicspace:topicname");
```

.NET でのプロパティー

.NET アプリケーションは、オブジェクトのプロパティーを取得および設定するときに、PropertyContext インターフェースのメソッドを使用します。

PropertyContext インターフェースは、プロパティーを取得して設定するメソッドをカプセル化します。 これらのメソッドは、直接的または間接的に以下のクラスによって継承されます。

- BytesMessage
- 接続
- ConnectionFactory
- ConnectionMetaData
- Destination
- MapMessage
- メッセージ
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Session
- StreamMessage
- TextMessage

アプリケーションがプロパティーの値を設定した場合、プロパティーに設定されていた以前の値は新しい値によって置き換えられます。

XMS プロパティーについて詳しくは、XMS オブジェクトのプロパティーを参照してください。

使い勝手を向上させるため、XMS での XMS プロパティーの 名前と値は、XMSC という構造体の public 定数として事前定義されます。 それらの定数の名前は、XMSC.*constant* という形式です。例えば、XMSC.USERID (プロパティー名前定数) および XMSC.DELIVERY_AS_APP (値定数) のようになります。

さらに、IBM.XMS.MQC 構造体を使用することによって IBM MQ の定数にアクセスすることができます。 IBM.XMS ネーム・スペースが既にインポートされている場合は、MQC.*constant* の形式でそれらのプロパティーの値にアクセスできます。 例えば、MQC.MQRO_COA_WITH_FULL_DATA を使用できます。

さらに、 .NET の XMS .NET クラスと IBM MQ クラスの両方を使用し、 IBM.XMS および IBM.WMQ 名前空間の場合は、MQC 構造体の名前空間を完全に修飾して、各オカレンスが固有になるようにする必要があります。

現在のところ、高度な機能の一部は、管理 .NET 環境の中でサポートされていません。 詳しくは、652 ページの『.NET における管理操作および非管理操作』 を参照してください。

.NET での存在しないプロパティーの処理

XMS .NET での存在しないプロパティーの処理は、JMS 仕様と広範囲に整合しており、同時に XMS の C および C++ の実装環境とも整合しています。

JMS では、存在しないプロパティーにアクセスした場合、存在しない (ヌル) 値を要求された型にメソッドが変換しようとすると、Java システム例外が発生する可能性があります。 プロパティーが存在しない場合は、以下の例外が発生します。

- getStringProperty および getObjectProperty がヌルを戻す
- Boolean.valueOf(null) が false を戻すため、getBooleanProperty が false を戻す
- Integer.valueOf(null) が例外をスローするため、getIntProperty などが java.lang.NumberFormatException をスローする

プロパティーが XMS .NET に存在しない場合は、 以下の例外が発生します。

- GetStringProperty および GetObjectProperty (さらに GetBytesProperty) がヌルを戻す (Java の場合と同じ)
- GetBooleanProperty が System.NullReferenceException をスローする
- GetIntProperty などが System.NullReferenceException をスローする

この実装環境は Java とは異なりますが、JMS 仕様と広範囲に整合しており、 同時に XMS C および C++ のインターフェースとも整合しています。 Java 実装環境の場合と同様に、XMS .NET は、System.Convert 呼び出しから呼び出し元までのすべての例外に波及します。 ただし、Java とは異なり、XMS は、ヌルをシステム変換ルーチンに渡すことで .NET フレームワークの固有の動作を単に 使用するのではなく、NullReferenceExceptions を明示的にスローします。 アプリケーションがプロパティーを "abc" のような String に設定し、GetIntProperty を呼び出すと、Convert.ToInt32("abc") によってスローされた System.FormatException は呼び出し元に波及します。この振る舞いは Java と一致しています。 MessageFormatException がスローされるのは、setProperty と getProperty に使用した型が非互換である場合に限られます。 この振る舞いも Java と一致しています。

.NET でのエラー処理

XMS .NET 例外は、すべて System.Exception から派生します。 XMS メソッド呼び出しは、MessageFormatException などの特定の XMS 例外 、一般的な XMSException、または NullReferenceException などのシステム例外をスローすることができます。

特定の catch ブロックまたは汎用の `System.Exception` catch ブロックでこれらのエラーをキャッチするためのアプリケーションを作成します (どちらを使用するかは、アプリケーションの要件によって異なります)。

.NET でのメッセージ・リスナーおよび例外リスナーの使用法

.NET アプリケーションは、メッセージ・リスナーを使用してメッセージを非同期に受信し、例外リスナーを使用して接続の問題に関する 通知を非同期に受信します。

このタスクについて

メッセージ・および例外リスナーの両方の機能は、.NET と C++ の場合と同じです。ただし、いくつか実装に小さな相違点があります。

手順

- メッセージを非同期に受信するようにメッセージ・リスナーをセットアップするには、以下の手順に従ってください。
  a) メッセージ・リスナー代行のシグニチャーを突き合わせるメソッドを定義します。
  静的メソッドまたはインスタンス・メソッドのいずれかを定義できます。また、アクセス可能なクラスであればどのクラスでもメソッドを定義できます。 代行シグニチャーは以下のとおりです。

```
public delegate void MessageListener(IMessage msg);
```

  また、メソッドを以下のように定義できます。

```
void SomeMethodName(IMessage msg);
```

  b) 次の例のようなコードを使用して、このメソッドを代行としてインスタンス化します。

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

  c) 代行を 1 つ以上のコンシューマーに登録するため、コンシューマーの MessageListener プロパティーに代行を次のように設定します。

```
consumer.MessageListener = OnMsgMethod;
```

  MessageListener をヌルに戻すことで、代行を除去できます。

```
consumer.MessageListener = null;
```

- 例外リスナーをセットアップするには、以下の手順を実行します。
  例外リスナーは、メッセージ・リスナーとほぼ同様に機能しますが、代行定義が異なり、メッセージ・コンシューマーではなく接続に割り当てられています。 これは C++ と同一です。

  a) メソッドを定義します。
  代行シグニチャーは以下のとおりです。

```
public delegate void ExceptionListener(Exception ex);
```

  このため、定義されるメソッドは以下のようになります。

```
void SomeMethodName(Exception ex);
```

  b) 次の例のようなコードを使用して、このメソッドを代行としてインスタンス化します。

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) 代行を接続に登録するため、ExceptionListener プロパティーを設定します。

```
connection.ExceptionListener = OnExMethod ;
```

  ExceptionListener を次のようにリセットすることで、代行を除去できます。

```
null:  connection.ExceptionListener = null;
```

XMS .NET 管理対象オブジェクトでの作業

このセクションのトピックでは、管理対象オブジェクトに関する情報を記載します。 XMS アプリケーションは、中央の管理対象オブジェクト・リポジトリーからオブジェクト定義を取得し、それらを使用して接続ファクトリーと宛先を作成できます。

このタスクについて

このセクションでは、管理対象オブジェクトの作成と管理に役立つ情報を提供し、XMS によってサポートされる管理対象オブジェクト・リポジトリーの種類について説明します。 このセクションではまた、XMS アプリケーションが管理対象オブジェクト・リポジトリーへの接続を行って、必要な管理対象オブジェクトを取得する方法についても説明します。

このセクションでは、以下のトピックについて説明します。

- 657 ページの『XMS .NET サポートされる管理対象オブジェクト・リポジトリーのタイプ』
- 657 ページの『XMS .NET 管理対象オブジェクトのプロパティー・マッピング』
- 659 ページの『XMS .NET 管理対象 ConnectionFactory オブジェクトの必須プロパティー』
- 660 ページの『XMS .NET 管理対象 Destination オブジェクトの必須プロパティー』
- 661 ページの『XMS .NET 管理対象オブジェクトの作成』
- 661 ページの『XMS .NET InitialContext オブジェクトの作成』
- 662 ページの『XMS .NET InitialContext プロパティー』
- 662 ページの『XMS 初期コンテキストの URI フォーマット』
- 663 ページの『XMS .NET の JNDI ルックアップ Web サービス』
- 664 ページの『XMS .NET 管理対象オブジェクトの検索』

XMS .NET サポートされる管理対象オブジェクト・リポジトリーのタイプ

管理対象オブジェクトの中でも、ファイル・システムと LDAP は IBM MQ および WebSphere Application Server への接続に使用できるのに対し、COS ネーミングは WebSphere Application Server のみへの接続に使用できます。

ファイル・システム・オブジェクト・ディレクトリーは、シリアライズされた Java Naming Directory Interface (JNDI) オブジェクトの形式を使用します。 LDAP オブジェクト・ディレクトリーは、JNDI オブジェクトを含むディレクトリーです。 ファイル・システムおよび LDAP オブジェクト・ディレクトリーは、IBM MQ 以降で提供される IBM MQ Explorer によって管理できます。 ファイル・システムと LDAP オブジェクト・ディレクトリーの両方を使用して、 IBM MQ 接続ファクトリーと宛先を集中化することにより、クライアント接続を管理できます。 ネットワーク管理者は、同じ中央リポジトリーを参照し、中央リポジトリーの接続設定に行われた変更を反映するために自動的に更新される、複数のアプリケーションをデプロイすることができます。

COS ネーミング・ディレクトリーは、WebSphere Application Server service integration bus 接続ファクトリーおよび宛先を含んでおり、WebSphere Application Server 管理コンソールを使用して管理できます。 XMS アプリケーションが COS ネーミング・ディレクトリーからオブジェクトを取り出すには、JNDI ルックアップ Web サービスがデプロイされている必要があります。 この Web サービスは、すべての WebSphere Application Server service integration technologies で使用できるわけではありません。 詳しくは、製品資料を参照してください。

注 : オブジェクト・ディレクトリーに対する変更を有効にするには、アプリケーション接続を再始動します。

XMS .NET 管理対象オブジェクトのプロパティー・マッピング

XMS .NET アプリケーションを使用可能にして、IBM MQ JMS と WebSphere Application Server の接続ファクトリー・オブジェクト定義および宛先オブジェクト定義を使用するには、これらの定義から取り出したプロパティーを、定義に対応する XMS プロパティーで、かつ XMS 接続ファクトリーおよび宛先に設定できるプロパティーにマップする必要があります。

例えば、IBM MQ JMS 接続ファクトリーから取得したプロパティーを使用して XMS 接続ファクトリーを作成するには、その 2 つの間でプロパティーをマップする必要があります。

すべてのプロパティー・マッピングは、自動的に実行されます。

657 ページの表 87 には、接続ファクトリーおよび宛先の最も一般的なプロパティーのいくつかの間のマッピングを示します。 この表に示すプロパティーはほんの数組の例に過ぎず、示されているすべてのプロパティーがすべての接続タイプおよびサーバーに関連するわけではありません。

表 *87.* 接続ファクトリーと宛先のプロパティーの名前マッピングの例

IBM MQ JMS プロパティー名	**XMS プロパティー名**	**WebSphere Application Server service integration bus プロパティー名**
PERSISTENCE (PER)	XMSC_DELIVERY_MODE	
EXPIRY (EXP)	XMSC_TIME_TO_LIVE	
PRIORITY (PRI)	XMSC_PRIORITY	
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

注 : 658 ページの表 88 に示されているプロパティーは、JMS と XMS .NET に適用されます。

表 88. XMS .NET プロパティー					
Property	**オブジェクト・タイプ**				
	CF	**QCF**	**TCF**	**キュー**	**トピック**
APPLICATIONNAME	Y	Y	Y	N/A	N/A
ASYNCEXCEPTION	Y	Y	Y	N/A	N/A
CCDTURL	Y	Y	Y	N/A	N/A
CHANNEL	Y	Y	Y	N/A	N/A
CONNECTIONNAMELIST	Y	Y	Y	N/A	N/A
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/A	N/A
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/A	N/A
CLIENTID	N/A	Y	N/A	N/A	N/A
COMPHDR 659 ページの『1』	Y	N/A	Y	N/A	N/A
COMPMSG 659 ページの『1』	Y	Y	Y	N/A	N/A
接続 (CONNOPT) 659 ページの『1』	Y	Y	Y	N/A	N/A
CONNTAG 659 ページの『1』	Y	Y	Y	N/A	N/A
説明 659 ページの『1』	N/A	Y	N/A	Y	Y
EXPIRY 659 ページの『1』	N/A	N/A	N/A	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
HOSTNAME	N/A	Y	N/A	N/A	N/A
LOCALADDRESS	N/A	Y	N/A	N/A	N/A
PERSISTENCE	N/A	N/A	N/A	Y	Y
PORT	N/A	Y	N/A	N/A	N/A
優先順位 659 ページの『1』	N/A	N/A	N/A	Y	Y
プロバイダー・バージョン 659 ページの『1』	N/A	Y	N/A	N/A	N/A
QMANAGER	Y	Y	Y	Y	N/A

表 88. XMS .NET プロパティー (続き)					
Property	**オブジェクト・タイプ**				
	CF	**QCF**	**TCF**	**キュー**	**トピック**
キュー 659 ページの『1』	N/A	N/A	N/A	Y	N/A
SHARECONVALLOWED	Y	Y	Y	N/A	N/A
トピック 659 ページの『1』	N/A	N/A	N/A	N/A	Y
トランスポート 659 ページの『1』	N/A	Y	N/A	N/A	N/A

注:

1. これらのプロパティーにはアプリケーション・レベルのプロパティーはありませんが、管理対象プロパティーを使用してオプションで設定できます。

OutboundSNI プロパティー

V 9.3.0

IBM MQ 9.3.0 以降、XMSC_WMQ_OUTBOUND_SNI プロパティーを設定できます。これにより、アプリケーションの **OutboundSNI** プロパティーが設定されます。

XMSC_WMQ_OUTBOUND_SNI_PROPERTY には、以下の値が含まれます。

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL。これは、「CHANNEL」にマップされます。
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME。これは「HOSTNAME」にマップされます。
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK。これは「*」にマップされます。

さらに、MQOUTBOUND_SNI 環境変数を使用して **OutboundSNI** プロパティーを設定することもできます。この環境変数は以下の値を取ります。

- CHANNEL
- HOSTNAME
- *

注: 特定の値が設定されていない場合、プロパティーはデフォルトで XMSC_WMQ_OUTBOUND_SNI_CHANNEL になります。

管理対象ノードで **OutboundSNI** プロパティーを設定する際の優先順位は、以下のとおりです。

1. アプリケーション・レベル・プロパティー
2. 環境変数

非管理対象ノードの **OutboundSNI** プロパティーでは、 `mqclient.ini` のみがサポートされます。

XMS .NET 管理対象 ConnectionFactory オブジェクトの必須プロパティー

アプリケーションが接続ファクトリーを作成する場合には、メッセージング・サーバーへの接続を作成するために多くのプロパティーを定義する必要があります。

以下の表にリストするプロパティーは、メッセージング・サーバーへの接続を作成するためにアプリケーションで設定する必要がある最小限のプロパティーです。 接続の作成方法をカスタマイズする場合は、ご使用のアプリケーションで、必要に応じて ConnectionFactory オブジェクトの任意の追加プロパティーを設定できます。 詳しくは、 ConnectionFactory のプロパティーを参照してください。 有効なプロパティーの完全なリストが含まれています。

IBM MQ キュー・マネージャーへの接続

表 *89. IBM MQ* キュー・マネージャーへの接続で使用する管理対象 *ConnectionFactory* オブジェクトのプロパティー設定

必須の XMS プロパティー	必要となる同等の IBM MQ JMS プロパティー
XMSC_CONNECTION_TYPE	XMS は、接続ファクトリーのクラス名と TRANSPORT (TRAN) プロパティーからこのプロパティーを解決します。
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	キュー・マネージャーの名前

ブローカーへのリアルタイム接続

表 *90.* ブローカーへのリアルタイム接続で使用する管理対象 *ConnectionFactory* オブジェクトのプロパティー設定

必要な XMS	必要となる同等の IBM MQ JMS プロパティー
XMSC_CONNECTION_TYPE	XMS は、接続ファクトリーのクラス名と TRANSPORT (TRAN) プロパティーからこのプロパティーを解決します。
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

WebSphere Application Server service integration bus への接続

表 *91. WebSphere Application Server service integration bus* への接続で使用する管理対象 *ConnectionFactory* オブジェクトのプロパティー設定

XMS プロパティー	説明
XMSC_CONNECTION_TYPE	アプリケーションの接続先となるメッセージング・サーバーのタイプです。。これは、接続ファクトリーのクラス名から判別されます。
XMSC_WPM_BUS_NAME	接続ファクトリーの場合は、アプリケーションの接続先となるサービス統合バスの名前であり、宛先の場合は、その宛先が存在するサービス統合バスの名前です。

XMS .NET 管理対象 Destination オブジェクトの必須プロパティー

宛先を作成するアプリケーションは、管理対象 Destination オブジェクトで複数のプロパティーを設定する必要があります。

表 *92.* 管理対象 *Destination* オブジェクトのプロパティー設定

接続のタイプ	Property	説明
IBM MQ キュー・マネージャー	QUEUE (QU) TOPIC (TOP)	接続先のキュー アプリケーションが宛先として使用するトピック
ブローカーへのリアルタイム接続	TOPIC (TOP)	アプリケーションが宛先として使用するトピック
WebSphere Application Server service integration bus	topicName queueName	アプリケーションがトピックに接続している場合 アプリケーションがキューに接続している場合

XMS .NET 管理対象オブジェクトの作成

メッセージング・サーバーへの接続を作成するために XMS アプリケーションが必要とする ConnectionFactory および Destination オブジェクト定義は、適切な管理ツールを使用して作成する必要があります。

始める前に

XMS によってサポートされるさまざまな種類の管理対象オブジェクト・リポジトリーについて詳しくは、657 ページの『XMS .NET サポートされる管理対象オブジェクト・リポジトリーのタイプ』を参照してください。

このタスクについて

IBM MQ の管理対象オブジェクトを作成するには、IBM MQ Explorer または IBM MQ JMS 管理 (JMSAdmin) ツールを使用します。

IBM MQ または IBM Integration Bus の管理対象オブジェクトを作成するには、IBM MQ JMS 管理 (JMSAdmin) ツールを使用します。

WebSphere Application Server service integration bus の管理対象オブジェクトを作成するには、WebSphere Application Server 管理コンソールを使用します。

管理ツールでは、このプロパティーは短縮して **APPLICATIONNAME** または **APPNAME** とも呼ばれます。

注 : JMSAdmin を使用して TRANSPORT(UNMANAGED) を設定することはできません。 そのため、管理者が選択したアプリケーション名を使用して管理されていない XMS クライアントを取得するには、次のコマンドを入力する必要があります。

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

以下のステップで、管理対象のオブジェクトを作成するために行うことを要約します。

手順

1. 接続ファクトリーを作成し、必要なプロパティーを定義して、アプリケーションから選択対象のサーバーへの接続を作成します。

   接続を作成するために XMS で必要とされる最低限のプロパティーは、659 ページの『XMS .NET 管理対象 ConnectionFactory オブジェクトの必須プロパティー』で定義されています。
2. アプリケーションの接続先のメッセージング・サーバーで、必要な宛先を作成します。
   - IBM MQ キュー・マネージャーへの接続の場合は、キューまたはトピックを 1 つ作成します。
   - ブローカーへのリアルタイム接続の場合は、トピックを 1 つ作成します。
   - WebSphere Application Server service integration bus への接続の場合は、キューまたはトピックを 1 つ作成します。

   接続を作成するために XMS で必要とされる最低限のプロパティーは、660 ページの『XMS .NET 管理対象 Destination オブジェクトの必須プロパティー』で定義されています。

XMS .NET InitialContext オブジェクトの作成

アプリケーションは、管理対象オブジェクト・リポジトリーへの接続を作成するために使用される初期コンテキストを作成して、必要な管理対象オブジェクトを取得する必要があります。

このタスクについて

InitialContext オブジェクトは、そのリポジトリーへの接続をカプセル化します。 XMS API には、 以下のタスクを実行するためのメソッドが用意されています。

- InitialContext オブジェクトの作成
- 管理対象オブジェクト・リポジトリー内で、管理対象オブジェクトを検索する。

手順

- InitialContext オブジェクトの作成について詳しくは、.NET の InitialContext および InitialContext の プロパティーを参照してください。

XMS .NET InitialContext プロパティー

InitialContext コンストラクターのパラメーターには Uniform Resource Indicator (URI) で指定される、管理対象オブジェクトのリポジトリーのロケーションが含まれます。 アプリケーションがリポジトリーへの接続を確立するためには、URI に含まれる情報より多くの情報を指定することが必要な場合があります。

JNDI および .NET の XMS 実装では、追加情報をコンストラクターへの環境 Hashtable で指定します。

管理対象オブジェクト・リポジトリーのロケーションは、XMSC_IC_URL プロパティーで定義します。 このプロパティーは、通常は Create 呼び出しに渡されますが、検索の前に別のネーミング・ディレクトリーに接続するように変更できます。 FileSystem コンテキストまたは LDAP コンテキストの場合、このプロパティーはディレクトリーのアドレスを定義します。 COS ネーミングの場合、これは、これらのプロパティーを使用して JNDI ディレクトリーに接続する Web サービスのアドレスです。

以下のプロパティーは、JNDI ディレクトリーに接続する目的で使用するように、未変更のまま Web サービスに渡されます。

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

XMS 初期コンテキストの URI フォーマット

管理対象オブジェクトのリポジトリーのロケーションは、Uniform Resource Indicator (URI) で指定します。 URI のフォーマットは、コンテキストのタイプにより異なります。

FileSystem コンテキスト

FileSystem コンテキストの場合は、ファイル・システム・ベースのディレクトリーのロケーションが URL によって指定されます。 URL の構造は、RFC 1738 の「*Uniform Resource Locators (URL)*」に定義されています。 URL にはプレフィックス `file://` があり、プレフィックスの後の構文は、XMS が動作するシステム上で開くことができるファイルの有効な定義になっています。

この構文はプラットフォーム固有の構文にすることができます。また、分離文字として「/」または「¥」を使用できます。 「¥」を使用した場合、分離文字はそれぞれ追加の「¥」を使用してエスケープする必要があります。 こうすることにより、.NET フレームワークで、分離文字がそれ以降の文字列のエスケープ文字として解釈されるのを防止できます。

以下の例に、この構文が示されています。

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

LDAP コンテキスト

LDAP コンテキストの場合、URL の基本構造は RFC 2255 (*LDAP URL* の形式) によって定義され、大文字と小文字をを区別しない接頭部である `ldap://` が付けられます。

正確な構文を次に示します。

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

この構文は RFC で定義されているものと同様ですが、属性、スコープ、フィルター、拡張子のいずれにも対応していません。

この構文の例を以下に示します。

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

WSS コンテキスト

WSS コンテキストの場合、URL は、`http://` のプレフィックスが付く、Web サービス・エンドポイントの形式になります。

あるいは、`cosnaming://` または `wsvc://` というプレフィックスを使用することもできます。

これら 2 つのプレフィックスは、HTTP を介してアクセスされる URL を持つ WSS コンテキストを使用しているという意味として解釈されます。WSS コンテキストを使用すると、URL から直接かつ容易に初期コンテキスト・タイプを得ることができます。

この構文例には、以下が含まれています。

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

XMS .NET の JNDI ルックアップ Web サービス

COS ネーミング・ディレクトリーに XMS からアクセスするには、JNDI ルックアップ Web サービスを WebSphere Application Server service integration bus・サーバー上に デプロイする必要があります。 この Web サービスは、COS ネーミング・サービスから取り込んだ Java 情報を、XMS アプリケーションが読み取り可能な形式に変換します。

この Web サービスは、インストール・ディレクトリーにあるエンタープライズ・アーカイブ・ファイル SIBXJndiLookupEAR.ear に含まれています。 現行リリースの IBM MQ Message Service Client (XMS) for .NET の場合、 SIBXJndiLookupEAR.ear は *install_dir*\java\lib ディレクトリーにあります。 このサービスは、管理コンソールまたは wsaadmin スクリプト・ツールを使用して WebSphere Application Server service integration bus・サーバー内にインストールできます。 Web サービス・アプリケーションのデプロイについて詳しくは、製品資料を参照してください。

XMS アプリケーションの内部で Web サービスを定義するために必要なことは、InitialContext オブジェクトの XMSC_IC_URL プロパティーを Web サービスのエンドポイント URL に設定することだけです。 例えば、Web サービスが MyServer というサーバー・ホストにデプロイされている場合の Web サービス・エンドポイント URL の例は次のようになります。

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

XMSC_IC_URL プロパティーを設定すると、InitialContext ルックアップ呼び出しにより、定義済みのエンドポイントで Web サービスを呼び出し、さらに COS ネーミング・サービスから必要な管理対象オブジェクトを検索できます。

.NET アプリケーションは、Web サービスを使用できます。 サーバー・サイドのデプロイメントは、XMS C、/C++ の場合も、XMS .NET の場合も同じです。XMS .NET は、 Microsoft .NET Framework を介して直接 Web サービスを呼び出します。

XMS .NET 管理対象オブジェクトの検索

XMS は、InitialContext オブジェクトの作成時に指定されたアドレス、または InitialContext プロパティーに指定されているアドレスを使用して、リポジトリーから管理対象オブジェクトを取り出します。

検索するオブジェクトには、次のタイプの名前を指定できます。

- Destination オブジェクトを説明する単純名。例えば、SalesOrders というキューの宛先。
- 複合名。サブコンテキストで構成でき、「/」で区切られ、末尾はオブジェクト名にする必要があります。複合名の例は、「Warehouse/PickLists/DispatchQueue2」です。Warehouse および Picklists はネーミング・ディレクトリーのサブコンテキスト、DispatchQueue2 は Destination オブジェクトの名前を表します。

アプリケーションで最新バージョン以外の XMS を使用する場合

デフォルトでは、新しい XMS バージョンがインストールされると、以前のバージョンを使用しているアプリケーションは、再コンパイルする必要なく自動的に新しいバージョンに切り替わります。ただし、アプリケーション構成ファイルで属性を設定することで、アプリケーションが新しいバージョンを使用できないようにすることができます。

このタスクについて

複数バージョンの共存機能により、最新バージョンの XMS をインストールしても、以前のバージョン XMS が上書きされることはありません。 代わりに、同じ XMS .NET アセンブリーの複数のインスタンスがグローバル・アセンブリー・キャッシュ (GAC) 内に共存しますが、そのバージョン番号は異なります。 内部で、GAC は、ポリシー・ファイルを使用して、アプリケーション呼び出しを最新バージョンの XMS に転送します。 アプリケーションを実行するために再コンパイルは必要ありません。最新バージョンの XMS .NET に用意されている新機能を利用することも可能です。

手順

- アプリケーションで古い XMS .NET バージョンを使用する必要がある場合は、アプリケーション構成ファイルで `publisherpolicy` 属性を no に設定します。

  注 : アプリケーション構成ファイルは、そのファイルが関連する実行可能プログラムの名前とサフィックス .config から成る名前を持つファイルです。 例えば、text.exe のアプリケーション構成ファイルの名前は text.exe.config になります。

  ただし、いかなる時点でも、システムのアプリケーションはすべて、同じバージョンの XMS .NET を使用します。

XMS アプリケーションの通信の保護

このセクションでは、XMS アプリケーションが Secure Sockets Layer (SSL) により WebSphere Application Server service integration bus・メッセージング・エンジンまたは IBM MQ キュー・マネージャーに接続するためのセキュア通信をセットアップする方法について説明します。

このタスクについて

このセクションでは、以下のトピックについて説明します。

- 665 ページの『IBM MQ キュー・マネージャーとのセキュア接続』
- 665 ページの『IBM MQ キュー・マネージャーへの XMS 接続の CipherSuite および CipherSpec 名前マッピング』
- 666 ページの『WebSphere Application Server service integration bus メッセージング・エンジンとのセキュア接続』
- 667 ページの『WebSphere Application Server service integration bus との接続に関する CipherSuite および CipherSpec 名前マッピング』

IBM MQ キュー・マネージャーとのセキュア接続

XMS .NET アプリケーションが IBM MQ キュー・マネージャーとのセキュア接続を確立できるようにするには、関係するプロパティーが ConnectionFactory オブジェクトで定義されていることが必要です。

暗号化ネゴシエーションで使用されるプロトコルは、Secure Sockets Layer (SSL) または Transport Layer Security (TLS) のいずれかです。そのどちらを使用するかは、ConnectionFactory オブジェクトの中で指定されている CipherSuite によります。

以下の表に、SSL を使用した IBM MQ キュー・マネージャーへの接続に関する ConnectionFactory のプロパティーを、簡単な説明とともに示します。

表 *93. SSL* による *IBM MQ* キュー・マネージャーとの接続に関する *ConnectionFactory* のプロパティー

プロパティー名	説明
XMSC_WMQ_SSL_CERT_STORES	キュー・マネージャーとの SSL 接続で使用される証明書取り消しリスト (CRL) を保持するサーバーの位置。
XMSC_WMQ_SSL_CIPHER_SPEC	キュー・マネージャーとのセキュア接続で使用する CipherSpec の名前。
XMSC_WMQ_SSL_CIPHER_SUITE	キュー・マネージャーとの TLS 接続で使用される CipherSuite の名前。 セキュア接続のネゴシエーションで使用されるプロトコルは、指定されている CipherSuite によって異なります。
XMSC_WMQ_SSL_CRYPTO_HW	クライアント・システムに接続されている暗号ハードウェアに関する構成詳細情報。
XMSC_WMQ_SSL_FIPS_REQUIRED	このプロパティーの値は、非 FIPS 準拠暗号スイートをアプリケーションで使用できるかどうかを判別します。 このプロパティーが true (真) に設定されている場合、クライアント/サーバー接続には FIPS アルゴリズムだけが使用されます。
XMSC_WMQ_SSL_KEY_REPOSITORY	鍵および証明書が保存されている鍵データベース・ファイルの位置。
XMSC_WMQ_SSL_KEY_RESETCOUNT	KeyResetCount は、秘密鍵の再ネゴシエーションが実行されるまで、1 つの SSL 会話の中で送受信される暗号化されていないデータの合計バイト数を表します。
XMSC_WMQ_SSL_PEER_NAME	キュー・マネージャーとの SSL 接続で使用されるピア名。

IBM MQ キュー・マネージャーへの *XMS* 接続の *CipherSuite* および *CipherSpec* 名前マッピング

InitialContext は、JMSAdmin Connection Factory のプロパティー SSLCIPHERSUITE と、ほぼそれに相当する XMS の XMSC_WMQ_SSL_CIPHER_SPEC との間の変換を実行します。
XMSC_WMQ_SSL_CIPHER_SUITE には値を指定したが、XMSC_WMQ_SSL_CIPHER_SPEC の値は省略した場合、類似の変換が必要になります。

665 ページの表 94 に、使用できる CipherSpec と、JSSE CipherSuite でそれに相当するもののリストを示します。

表 *94.* 使用できる *CipherSpec* と *JSSE CipherSuite* でそれに相当するもの

CipherSpec	相当する JSSE CipherSuite
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA

表 94. 使用できる *CipherSpec* と *JSSE CipherSuite* でそれに相当するもの (続き)	
CipherSpec	**相当する JSSE CipherSuite**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

注 : Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

WebSphere Application Server service integration bus メッセージング・エンジンとのセキュア接続

XMS .NET アプリケーションが WebSphere Application Server service integration bus メッセージング・エンジンとのセキュア接続を確立できるようにするには、関係するプロパティーが ConnectionFactory オブジェクトで定義されていることが必要です。

XMS では、WebSphere Application Server service integration bus との接続について SSL と HTTPS がサポートされています。 SSL と HTTPS により、認証と機密性を考慮したセキュア接続が提供されます。

WebSphere セキュリティーの場合と同じように、XMS のセキュリティーは、JSSE セキュリティー標準規格および命名規則に準拠して構成されています。 それには、セキュア接続のネゴシエーション時に使用されるアルゴリズムを指定するための CipherSuite の使用も含まれています。 暗号化ネゴシエーションで使用されるプロトコルは、SSL または TLS のいずれかです。 そのどちらを使用するかは、ConnectionFactory オブジェクトの中で指定されている CipherSuite によります。

666 ページの表 95 に、ConnectionFactory オブジェクトで定義する必要のあるプロパティーを示します。

表 95. *WebSphere Application Server service integration bus* ・メッセージング・エンジンとのセキュア接続に関する *ConnectionFactory* のプロパティー	
プロパティー名	**説明**
XMSC_WPM_SSL_CIPHER_SUITE	WebSphere Application Server service integration bus メッセージング・エンジンとの TLS 接続で使用される CipherSuite の名前。 セキュア接続のネゴシエーションで使用されるプロトコルは、指定されている CipherSuite によって異なります。
XMSC_WPM_SSL_KEYRING_LABEL	サーバーによる認証で使用される証明書。

以下に、WebSphere Application Server service integration bus・メッセージング・エンジンとのセキュア接続に関する ConnectionFactory プロパティーの例を示します。

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

ここで chain_name を、BootstrapTunneledSecureMessaging または BootstrapSecureMessaging のいずれかに設定する必要があります。さらに、port_number は、ブートストラップ・サーバーが着信要求を listen するポートの番号です。

以下に、サンプル値が挿入された WebSphere Application Server service integration bus・メッセージング・エンジンへのセキュア接続に関する ConnectionFactory のプロパティーの例を示します。

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
```

```
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

WebSphere Application Server service integration bus との接続に関する *CipherSuite* および *CipherSpec* 名前マッピング

IBM Global Security Kit (GSKit) は CipherSuites ではなく CipherSpecs を使用するため、XMSC_WPM_SSL_CIPHER_SUITE プロパティーで指定された JSSE スタイルの CipherSuite 名を GSKit スタイルの CipherSpec 名にマップする必要があります。

667 ページの表 96 に、認識された CipherSuite のそれぞれに対応する CipherSpec を示します。

表 *96.* 使用できる *CipherSuite* とそれに対応する *CipherSpec*

CipherSuite	**CipherSpec で対応するもの**
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

注 : Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

XMS メッセージ

このセクションでは、XMS メッセージの構造とコンテンツについて説明するとともに、アプリケーションによる XMS メッセージの処理方法について説明します。

このセクションでは、以下のトピックについて説明します。

- 667 ページの『XMS メッセージのパーツ』
- 668 ページの『XMS メッセージのヘッダー・フィールド』
- 668 ページの『XMS メッセージのプロパティー』
- 671 ページの『XMS メッセージの本文』
- 675 ページの『メッセージ・セレクター』
- 675 ページの『XMS メッセージの IBM MQ メッセージへのマッピング』

XMS メッセージのパーツ

XMS メッセージは、ヘッダー、プロパティーのセット、および本文で構成されています。

ヘッダー
メッセージのヘッダーにはフィールドが含まれており、すべてのメッセージには同じセットのヘッダー・フィールドが含まれています。 XMS とアプリケーションは、ヘッダー・フィールドの値を使用して経路メッセージを識別します。 ヘッダー・フィールドについて詳しくは、668 ページの『XMS メッセージのヘッダー・フィールド』を参照してください。

プロパティーのセット
メッセージのプロパティーは、メッセージについての追加情報を指定します。 すべてのメッセージには、同じセットのヘッダー・フィールドがありますが、すべてのメッセージはプロパティーの異なるセットを持つことができます。 詳しくは、668 ページの『XMS メッセージのプロパティー』 を参照してください。

Body
メッセージの本文にはアプリケーション・データが含まれています。 詳しくは、671 ページの『XMS メッセージの本文』 を参照してください。

アプリケーションは、受信するメッセージを選択できます。 そのために、メッセージ・セレクターを使用して選択基準を指定します。 選択基準は、特定のヘッダー・フィールドの値とメッセージの任意のプロパティーの値に基づいて指定できます。 メッセージ・セレクターについて詳しくは、675 ページの『メッセージ・セレクター』を参照してください。

XMS メッセージのヘッダー・フィールド

XMS アプリケーションが、WebSphere JMS アプリケーションとの間でメッセージを交換できるようにするため、XMS メッセージのヘッダーには JMS メッセージのヘッダー・フィールドが含まれています。

これらのヘッダー・フィールドの名前は、プレフィックス JMS で始まります。 JMS メッセージ・ヘッダー・フィールドについて詳しくは、「*Java Message Service Specification*」を参照してください。

XMS は、JMS メッセージのヘッダー・フィールドを Message オブジェクトの属性として実装します。 各ヘッダー・フィールドには、値の設定および取得に使用する独自のメソッドがあります。 これらのメソッドについては、IMessage を参照してください。 ヘッダー・フィールドは、常時読み取り可能と書き込み可能です。

668 ページの表 97 に、JMS メッセージのヘッダー・フィールドのリストと、伝送されるメッセージに設定される各フィールドの値を示します。 一部のフィールドは、 アプリケーションがメッセージを送信するとき、または JMSRedelivered の場合にはアプリケーションがメッセージを 受信するときに、XMS によって自動的に設定されます。

表 *97. JMS* メッセージのヘッダー・フィールド.]

JMS メッセージのヘッダー・フィールド名	**送信されたメッセージ用の値の設定方法 (方法 [クラス] の形式)**
JMSCorrelationID	JMSCorrelationID の設定 [Message]
JMSDeliveryMode	送信 [MessageProducer]
JMSDestination	送信 [MessageProducer]
JMSExpiration	送信 [MessageProducer]
JMSMessageID	送信 [MessageProducer]
JMSPriority	送信 [MessageProducer]
JMSRedelivered	受信 [MessageConsumer]
JMSReplyTo	JMSReplyTo の設定 [Message]
JMSTimestamp	送信 [MessageProducer]
JMSType	JMSType の設定 [Message]

XMS メッセージのプロパティー

XMS は、3 種類の メッセージ・プロパティー (JMS 定義プロパティー、IBM 定義プロパティー、およびアプリケーション定義プロパティー) を サポートします。

XMS アプリケーションは、WebSphere JMS アプリケーションとの間で メッセージを交換できます。 これは、XMS が、次に示す Message オブジェクトの事前定義プロパティーを サポートするためです。

- WebSphere JMS がサポートするものと同じ JMS 定義のプロパティー。 このプロパティーの名前は、プレフィックス JMSX で始まります。
- WebSphere JMS がサポートするものと同じ IBM 定義のプロパティー。 このプロパティーの名前は、プレフィックス JMS_IBM_ で始まります。

それぞれの事前定義プロパティーには、2 つの名前があります。

- JMS 名 (JMS 定義プロパティーの場合) または WebSphere JMS 名 (IBM 定義プロパティーの場合)。

これは、JMS または WebSphere JMS で プロパティーが認識されるときの名前であり、このプロパティーを持つメッセージとともに伝送される名前でもあります。 XMS アプリケーションは、この名前を使用してメッセージ・セレクター式のプロパティーを識別します。

- メッセージ・セレクター式を除くすべての状況でプロパティーを識別するための XMS 名。 各 XMS 名は、`IBM.XMS.XMSC` クラスで名前付き定数として定義されています。 名前付き定数の値は、対応する JMS または WebSphere JMS 名になります。

事前定義プロパティーに加えて、XMS アプリケーションは、メッセージ・プロパティーの独自のセットを作成および使用できます。 これらのプロパティーは、アプリケーション定義プロパティー と呼ばれます。

アプリケーションがメッセージを作成した後のメッセージのプロパティーは、読み取り可能と書き込み可能です。 プロパティーは、アプリケーションがメッセージを送信した後も、読み取り可能と書き込み可能の状態のままになります。 アプリケーションがメッセージを受信するときのメッセージのプロパティーは、読み取り専用です。 メッセージのプロパティーが読み取り専用の場合にアプリケーションが Message クラスの `Clear Properties` メソッドを呼び出すと、プロパティーは読み取り可能および書き込み可能になります。 このメソッドにより、プロパティーもクリアされます。

受信したメッセージを、メッセージ・プロパティーのクリア後に転送するときの動作は、メッセージ・プロパティーがクリアされた標準の WMQ XMS for .NET BytesMessage を転送するときの動作と整合しています。

ただし、次のプロパティーが失われるため、これは推奨されません。

- JMS_IBM_Encoding プロパティー値。このプロパティーがないと、意味を持つようにメッセージ・データをデコードできなくなります。
- JMS_IBM_Format プロパティー値。このプロパティーがないと、(MQMD または新規の MQRFH2) メッセージ・ヘッダーと既存のヘッダーとの間のヘッダー・チェーニングが壊れます。

メッセージのすべてのプロパティーの値を判別するため、アプリケーションは、Message クラスの Get Properties メソッドを呼び出すことができます。 このメソッドは、Property オブジェクトのリストをカプセル化するイテレーターを作成します。そのイテレーターにおいて各 Property オブジェクトは、メッセージのプロパティーを表します。 その際、 アプリケーションは、Iterator クラスのメソッドを使用して各 Property オブジェクトを順番に取得し、Property クラスのメソッドを 使用して各プロパティーの名前、データ型、および値を取得できます。

メッセージの *JMS* 定義プロパティー

XMS と WebSphere JMS の両方で、1 つのメッセージに対し複数の JMS 定義プロパティーがサポートされています。

669 ページの表 98 に、メッセージの JMS 定義プロパティーのリストを示します。 このリストにあるプロパティーは、XMS と WebSphere JMS の両方でサポートされます。 JMS 定義プロパティーについては、「*Java Message Service Specification*」を参照してください。 JMS 定義プロパティーは、ブローカーへのリアルタイム接続では無効です。

この表では、各プロパティーのデータ型を明示し、伝送されるメッセージに設定されるプロパティーの値を示しています。 一部のプロパティーは、 アプリケーションがメッセージを送信するとき、または JMSXDeliveryCount の場合にはアプリケーションがメッセージを 受信するときに、XMS によって自動的に設定されます。

表 *98.* メッセージの *JMS* 定義プロパティー

JMS 定義プロパティーの XMS 名	**JMS の名前**	**データ・タイプ**	**送信されたメッセージ用の値の設定方法 (方法 [クラス] の形式)**
JMSX_APPID	JMSXAppID	System.String	送信 [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	受信 [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	ストリング・プロパティーの設定 [PropertyContext]

表 98. メッセージの *JMS* 定義プロパティー (続き)			
JMS 定義プロパティーの XMS 名	**JMS の名前**	**データ・タイプ**	**送信されたメッセージ用の値の設定方法 (方法 [クラス] の形式)**
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	整数の設定プロパティー [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	送信 [MessageProducer]

メッセージの *IBM* 定義プロパティー

XMS および WebSphere JMS では、1 つのメッセージに対して複数の IBM 定義プロパティーがサポートされています。

670 ページの表 99 は、 XMS と WebSphere JMS の両方でサポートされる、メッセージの IBM 定義済みプロパティーをリストしています。 IBM 定義プロパティーについて詳しくは、IBM MQ または WebSphere Application Server の製品資料を参照してください。

この表では、各プロパティーのデータ型を明示し、伝送されるメッセージに設定されるプロパティーの値を示しています。 一部のプロパティーは、アプリケーションがメッセージを送信するときに、XMS によって 自動的に設定されます。

表 99. メッセージの *IBM* 定義プロパティー			
IBM 定義プロパティーの XMS 名	**WebSphere JMS 名**	**データ・タイプ**	**送信されたメッセージ用の値の設定方法 (方法 [クラス] の形式)**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	受信 [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	受信 [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	受信 [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM DESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	受信 [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	ストリング・プロパティーの設定 [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	整数の設定プロパティー [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	送信 [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	送信 [MessageProducer]

表 99. メッセージの *IBM* 定義プロパティー (続き)

IBM 定義プロパティーの XMS 名	WebSphere JMS 名	データ・タイプ	送信されたメッセージ用の値の設定方法 (方法 [クラス] の形式)
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	送信 [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ ID	JMS_IBM_Report_Pass_Correl_ID	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	整数の設定プロパティー [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	送信 [MessageProducer]

メッセージのアプリケーション定義プロパティー

XMS アプリケーションは、メッセージ・プロパティーの独自のセットを作成および使用できます。 アプリケーションがメッセージを送信するときには、これらのプロパティーもメッセージと一緒に伝送されます。受信側のアプリケーションは、メッセージ・セレクターを使用することにより、これらのプロパティーの値に基づいて受信するメッセージを選択することができます。

WebSphere JMS アプリケーションが、XMS アプリケーションによって送信されるメッセージを選択および処理できるようにするには、アプリケーション定義プロパティーの名前がメッセージ・セレクター式の ID を形成する場合の規則に準拠している必要があります。 詳しくは、146 ページの『JMS のメッセージ・セレクター』を参照してください。 アプリケーション定義プロパティーの値のデータ型は、System.Boolean、System.SByte、System.Int16、System.Int32、System.Int64、System.Float、System.Double、または System.String のいずれかである必要があります。

XMS メッセージの本文

メッセージの本文にはアプリケーション・データが含まれています。 ただしメッセージは、本文が含まれずに、ヘッダー・フィールドとプロパティーのみで構成されることもあります。

XMS は、5 つのタイプのメッセージ本文をサポートします。

バイト
本文にはバイト・ストリームが含まれています。 この本文タイプのメッセージは、バイト・メッセージ と呼ばれます。 IBytesMessage インターフェースには、バイト・メッセージの本文を処理するメソッドが含まれています。

マップ
本文には、名前値のペアが 1 組含まれており、それぞれの値には関連付けられたデータ型があります。 この本文タイプのメッセージは、マップ・メッセージ と呼ばれます。 IMapMessage インターフェースには、マップ・メッセージの本文を処理するメソッドが含まれています。

オブジェクト
本文には、シリアライズされた Java または .NET オブジェクトが含まれています。 この本文タイプのメッセージは、オブジェクト・メッセージ と呼ばれます。 IObjectMessage インターフェースには、オブジェクト・メッセージの本文を処理するメソッドが含まれています。

ストリーム
本文には、値のストリームが含まれており、それぞれの値には関連付けられたデータ型があります。 この本文タイプのメッセージは、ストリーム・メッセージ と呼ばれます。 IStreamMessage インターフェースには、ストリーム・メッセージの本文を処理するメソッドが含まれています。

テキスト
本文にはストリングが含まれています。 この本文タイプのメッセージは、テキスト・メッセージ と呼ばれます。 ITextMessage インターフェースには、テキスト・メッセージの本文を処理するメソッドが含まれています。

IMessage インターフェースは、すべてのメッセージ・オブジェクトの親です。このインターフェースをメッセージング関数で使用することで、すべての XMS メッセージ・タイプを表すことができます。

それらの各データ型のサイズ、および最大値と最小値についての詳細は、648 ページの表 84 を参照してください。

バイト・メッセージ

バイト・メッセージの本文には、バイトのストリームが含まれています。 本文には実際のデータのみが含まれており、 このデータを解釈する役割は、送受信を行うアプリケーションに委ねられています。

バイト・メッセージは、XMS アプリケーションが、XMS または JMS のアプリケーション・プログラミング・インターフェースを使用していないアプリケーションとメッセージを交換する必要がある場合に役立ちます。

アプリケーションがバイト・メッセージを作成した後のメッセージの本文は、書き込み専用です。 アプリケーションは、.NET の IBytesMessage インターフェースの適切な書き込みメソッドを呼び出すことにより、アプリケーション・データを本文にアセンブルします。 アプリケーションが値をバイト・メッセージ・ストリームに書き込むたびに、その値は、アプリケーションによって書き込まれた前の値の直後にアセンブルされます。 XMS は、アセンブルされた最後のバイトの位置を記憶するために内部のカーソルを維持しています。

アプリケーションがメッセージを送信すると、メッセージの本文は読み取り専用になります。 このモードのとき、アプリケーションはメッセージを繰り返し送信できます。

アプリケーションがバイト・メッセージを受信するときのメッセージの本文は、読み取り専用です。 アプリケーションは、IBytesMessage インターフェースの適切な読み取りメソッドを使用して、バイト・メッセージ・ストリームのコンテンツを読み取ることができます。 アプリケーションはバイトを順番に読み取り、XMS は、読み取られた最後のバイトの位置を記憶するために内部のカーソルを維持しています。

バイト・メッセージの本文が書き込み可能のときに、アプリケーションが IBytesMessage インターフェースの Reset メソッドを呼び出すと、その本文は読み取り専用になります。 このメソッドはまた、バイト・メッセージ・ストリームの先頭でカーソルを位置変更します。

バイト・メッセージの本文が読み取り専用の場合に、アプリケーションが .NET の IMessage インターフェースの `Clear Body` メソッドを呼び出すと、その本文は書き込み可能になります。 このメソッドにより、本文もクリアされます。

マップ・メッセージ

マップ・メッセージの本文には、名前値のペアが 1 組含まれており、それぞれの値には関連付けられたデータ型があります。

それぞれの名前と値のペアにおいて、名前は値を識別するストリングであり、値は 674 ページの表 100 にリストされている XMS データ・タイプのいずれかを持つアプリケーション・データのエレメントです。 名前値ペアの順序は定義されていません。 MapMessage クラスには、名前値ペアを設定および取得するメソッドが含まれています。

アプリケーションは、その名前を指定することによって名前値ペアにランダムにアクセスできます。

.NET アプリケーションは、MapNames プロパティーを使用して、マップ・メッセージの本文にある名前の列挙を取得することができます。

アプリケーションが名前と値のペアから値を取得するとき、その値は XMS によって別のデータ型に変換される可能性があります。 例えば、マップ・メッセージの本文から整数を取得するには、アプリケーションは MapMessage クラスの GetString メソッドを呼び出すことで、その整数をストリングとして取得することができます。 サポートされる型変換は、XMS によってプロパティーの値が 1 つのデータ型から別のデータ型に変換される場合にサポートされる型変換と同じです。 サポートされる変換について詳しくは、648 ページの『プロパティー値のデータ型の暗黙的な変換』を参照してください。

アプリケーションがマップ・メッセージを作成した後のメッセージの本文は、読み取り可能と書き込み可能です。 本文は、アプリケーションがメッセージを送信した後も、読み取り可能と書き込み可能の状態のままになります。 アプリケーションがマップ・メッセージを受信するときのメッセージの本文は、読み取り専用です。 マップ・メッセージの本文が読み取り専用のときに、アプリケーションが Message クラスの Clear Body メソッドを呼び出すと、本文は読み取り可能と書き込み可能になります。 このメソッドにより、本文もクリアされます。

オブジェクト・メッセージ

オブジェクト・メッセージの本文には、シリアライズされた Java オブジェクトまたは .NET オブジェクトが含まれています。

XMS アプリケーションは、オブジェクト・メッセージを受信してヘッダー・フィールドとプロパティーを変更してから、そのメッセージを別の宛先に送信することができます。 またアプリケーションは、オブジェクト・メッセージの本文をコピーし、そのコピーを使用して別のオブジェクト・メッセージを形成することもできます。 XMS は、オブジェクト・メッセージの本文をバイトの配列として扱います。

アプリケーションがオブジェクト・メッセージを作成した後のメッセージの本文は、読み取り可能と書き込み可能です。 本文は、アプリケーションがメッセージを送信した後も、読み取り可能と書き込み可能の状態のままになります。 アプリケーションがオブジェクト・メッセージを受信するときのメッセージの本文は、読み取り専用です。 オブジェクト・メッセージの本文が読み取り専用のときに、アプリケーションが .NET の IMessage インターフェースの `Clear Body` メソッドを呼び出すと、その本文は読み取り可能および書き込み可能になります。 このメソッドにより、本文もクリアされます。

ストリーム・メッセージ

ストリーム・メッセージの本文には、値のストリームが含まれており、それぞれの値には関連付けられたデータ型があります。

値のデータ・タイプは、 674 ページの表 100 にリストされている XMS データ・タイプの 1 つです。

アプリケーションがストリーム・メッセージを作成した後のメッセージの本文は、書き込み可能です。 アプリケーションは、.NET の IStreamMessage インターフェースの適切な書き込みメソッドを呼び出すことにより、アプリケーション・データを本文にアセンブルします。 アプリケーションが値をメッセージ・ストリームに書き込むたびに、その値とデータ型は、アプリケーションによって書き込まれた前の値の直後にアセンブルされます。 XMS は、アセンブルされた最後の値の位置を記憶するために内部のカーソルを維持しています。

アプリケーションがメッセージを送信すると、メッセージの本文は読み取り専用になります。 このモードのとき、アプリケーションはメッセージの送信を複数回実行することができます。

アプリケーションがストリーム・メッセージを受信するときのメッセージの本文は、読み取り専用です。 アプリケーションは、.NET の IStreamMessage インターフェースの適切な読み取りメソッドを使用して、メッセージ・ストリームのコンテンツを読み取ることができます。 アプリケーションは、値を順番に読み取り、XMS は、読み取られた最後の値の位置を記憶するために内部のカーソルを維持しています。

アプリケーションがメッセージ・ストリームから値を読み取る場合、その値は XMS によって別のデータ・タイプに変換されることがあります。 例えば、メッセージ・ストリームから整数を読み取るには、アプリケーションは ReadString メソッドを呼び出すことで、その整数をストリングとして取得することができます。 サポートされる型変換は、XMS によってプロパティーの値が 1 つのデータ型から別のデータ型に変換される場合にサポートされる型変換と同じです。 サポートされる変換について詳しくは、648 ページの『プロパティー値のデータ型の暗黙的な変換』 を参照してください。

アプリケーションがメッセージ・ストリームから値を読み込もうとするときにエラーが発生すると、カーソルは次に進みません。 アプリケーションは、別のデータ型として値の読み取りを試みることにより、エラーから回復できます。

ストリーム・メッセージの本文が書き込み専用のときに、アプリケーションが XMS の IStreamMessage インターフェースの Reset メソッドを呼び出した場合、その本文は読み取り専用になります。 このメソッドはまた、メッセージ・ストリームの先頭でカーソルを位置変更します。

ストリーム・メッセージの本文が読み取り専用の場合に、アプリケーションが XMS の IMessage インターフェースの `Clear Body` メソッドを呼び出すと、その本文は書き込み専用になります。 このメソッドにより、本文もクリアされます。

テキスト・メッセージ

テキスト・メッセージの本文には、ストリングが含まれています。

アプリケーションがテキスト・メッセージを作成した後のメッセージの本文は、読み取り可能と書き込み可能です。 本文は、アプリケーションがメッセージを送信した後も、読み取り可能と書き込み可能の状態のままになります。 アプリケーションがテキスト・メッセージを受信するときのメッセージの本文は、読み取り専用です。 テキスト・メッセージの本文が読み取り専用のときに、アプリケーションが .NET の IMessage インターフェースの Clear Body メソッドを呼び出した場合、その本文は読み取り可能および書き込み可能になります。 このメソッドにより、本文もクリアされます。

アプリケーション・データの要素のデータ型

XMS アプリケーションが IBM MQ classes for JMS アプリケーションとメッセージを交換できるようにするには、両方のアプリケーションが、メッセージの本文にあるアプリケーション・データを同じように解釈できることが必要です。

この理由により、XMS アプリケーションによってメッセージの本文に書き込まれるアプリケーション・データの各エレメントは、674 ページの表 100 のリストにあるデータ型のいずれかを持っている必要があります。 データ型ごとに、以下の表に互換性のある Java データ型を示します。 XMS には、 これらのデータ型のみを持つアプリケーション・データのエレメントを書き込むためのメソッドが用意されています。

表 *100. Java* データ・タイプと互換性のある *XMS* データ・タイプ

XMS データ・タイプ	**意味**	**互換性のある Java データ型**
System.Boolean	ブール値 true または false	boolean
System.Char16	2 バイト文字	char
System.SByte	符号付き 8 ビット整数	byte
System.Int16	符号付き 16 ビット整数	short
System.Int32	符号付き 32 ビット整数	int
System.Int64	符号付き 64 ビット整数	long
System.Float	符号付き浮動小数点数	float
System.Double	符号付き倍精度浮動小数点数	double
System.String	文字ストリング	ストリング

これらの各データ型のサイズ、最大値、および最小値についての詳細は、647 ページの『XMS プリミティブ型』を参照してください。

メッセージ・セレクター

XMS アプリケーションは、メッセージ・セレクターを使用して、受信するメッセージを選択します。

アプリケーションは、メッセージ・コンシューマーを作成する場合、メッセージ・セレクター式をそのコンシューマーに関連付けることができます。 メッセージ・セレクター式により、選択基準を指定します。

アプリケーションが IBM WebSphere MQ 7.0 キュー・マネージャーに接続されている場合は、メッセージの選択はキュー・マネージャー側で行われます。 XMS では選択を行わず、単にキュー・マネージャーから受信したメッセージを配信します。そのため、パフォーマンスが向上します。

アプリケーションは、複数のメッセージ・コンシューマーを作成し、それぞれに独自のメッセージ・セレクター式を設定することができます。 着信メッセージが 複数のメッセージ・コンシューマーの選択基準を満たす場合、XMS はそれらの各コンシューマーに着信メッセージを配信します。

メッセージ・セレクター式は、メッセージの次のプロパティーを参照できます。

- JMS 定義プロパティー
- IBM 定義プロパティー
- アプリケーション定義プロパティー

次のメッセージ・ヘッダーのフィールドも参照できます。

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

ただし、メッセージ・セレクター式は、メッセージの本文のデータを参照できません。

次に、メッセージ・セレクター式の例を示します。

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS は、メッセージの優先順位が 3 より大きい場合にのみ、このメッセージ・セレクター式を使用してメッセージをメッセージ・コンシューマーに配信します。これは、値が `Jaguar;` のアプリケーション定義プロパティーの製造元と、値が `xj6` または `xj12.` の別のアプリケーション定義プロパティーのモデルです。

XMS でメッセージ・セレクター式を形成する場合の構文規則は、IBM MQ classes for JMS の場合と同じです。 メッセージ・セレクター式を構成する方法については、IBM MQ 製品資料の注記を参照してください。なお、メッセージ・セレクター式では、JMS 定義プロパティーの名前が JMS 名になるように、また IBM 定義プロパティーの名前が IBM MQ classes for JMS 名になるようにしてください。 メッセージ・セレクター式で、XMS 名を使用することはできません。

XMS メッセージの IBM MQ メッセージへのマッピング

XMS メッセージの JMS ヘッダー・フィールドとプロパティーは、IBM MQ メッセージのヘッダー構造のフィールドにマッピングされます。

XMS アプリケーションが IBM MQ キュー・マネージャーに接続されている場合、キュー・マネージャーに送信されるメッセージは、IBM MQ メッセージにマッピングされます。その方法は、類似の環境において IBM MQ classes for JMS メッセージが IBM MQ メッセージにマッピングされる場合と同じです。

Destination オブジェクトの XMSC_WMQ_TARGET_CLIENT プロパティーが XMSC_WMQ_TARGET_DEST_JMS に設定されている場合、宛先に送信されるメッセージの JMS ヘッダー・フィールドとプロパティーは、IBM MQ メッセージの MQMD および MQRFH2 ヘッダー構造のフィールドに

マッピングされます。 この方法で XMSC_WMQ_TARGET_CLIENT プロパティーを設定する場合は、メッセージを受信するアプリケーションが MQRFH2 ヘッダーを処理できることが前提となります。 したがって、受信側のアプリケーションは、MQRFH2 ヘッダーを処理するように設計されている別の XMS アプリケーション、IBM MQ classes for JMS アプリケーション、またはネイティブ IBM MQ アプリケーションとなります。

Destination オブジェクトの XMSC_WMQ_TARGET_CLIENT プロパティーが代わりに XMSC_WMQ_TARGET_DEST_MQ に設定されている場合、宛先に送信されるメッセージの JMS ヘッダー・フィールドとプロパティーは、IBM MQ メッセージの MQMD ヘッダー構造のフィールドにマッピングされます。 メッセージには MQRFH2 ヘッダーが含まれておらず、MQMD ヘッダー構造のフィールドにマッピングできない JMS ヘッダー・フィールドとプロパティーはすべて無視されます。 したがって、そのメッセージを受信するアプリケーションは、MQRFH2 ヘッダーを処理するようには設計されていないネイティブ IBM MQ アプリケーションにすることができます。

キュー・マネージャーから受信される IBM MQ メッセージは、XMS メッセージにマッピングされます。 その方法は、類似の環境において IBM MQ メッセージが IBM MQ classes for JMS メッセージにマッピングされる場合と同じです。

着信 IBM MQ メッセージに MQRFH2 ヘッダーがある場合、結果の XMS メッセージの本体のタイプは、MQRFH2 ヘッダーの mcd フォルダーに含まれる **Msd** プロパティーの値によって決まります。 MQRFH2 ヘッダーに **Msd** プロパティーが含まれていない場合、または IBM MQ メッセージに MQRFH2 ヘッダーがない場合、出力される XMS メッセージには、MQMD ヘッダーの *Format* フィールドの値によって判別されるタイプの本文が含まれています。 *Format* フィールドが MQFMT_STRING に設定されていれば、XMS メッセージはテキスト・メッセージです。 それ以外の場合、XMS メッセージはバイト・メッセージです。 IBM MQ メッセージに MQRFH2 ヘッダーがない場合は、MQMD ヘッダーのフィールドから派生可能な JMS ヘッダー・フィールドとプロパティーのみが設定されます。

IBM MQ classes for JMS メッセージを IBM MQ メッセージにマッピングする方法について詳しくは、150 ページの『JMS メッセージの IBM MQ メッセージへのマッピング』 を参照してください。

IBM MQ Message Service Client (XMS) for .NET アプリケーションからのメッセージ記述子の読み取りと書き込み

IBM MQ メッセージの StrucId と Version 以外のすべてのメッセージ記述子 (MQMD) フィールドにアクセスできます。BackoutCount は読み取り可能ですが、書き込むことはできません。

IBM MQ Message Service Client (XMS) for .NET に用意されているメッセージ属性は、XMS アプリケーションが MQMD フィールドを設定したり、IBM WebSphere MQ アプリケーションを駆動したりするときに役立ちます。

パブリッシュ/サブスクライブ・メッセージングを使用する場合は、いくつかの制約事項が適用されます。例えば、MsgID や CorrelId などの MQMD フィールドは、設定しても無視されます。

この機能は、 **PROVIDERVERSION** プロパティーが 6 に設定されている場合にも使用できません。

IBM MQ Message Service Client (XMS) for .NET アプリケーションから IBM MQ メッセージ・データへのアクセス

IBM MQ Message Service Client (XMS) for .NET アプリケーション内で JMSBytesMessage の本体として MQRFH2 ヘッダー (存在する場合) およびその他の IBM MQ ヘッダー (存在する場合) を含む完全な IBM MQ メッセージ・データにアクセスすることができます。

このトピックで説明されている機能は、IBM WebSphere MQ 7.0 以降のキュー・マネージャーに接続されていて、かつ IBM MQ メッセージング・プロバイダーが通常モードになっている場合のみ使用できます。

Destination オブジェクト・プロパティーによって、XMS アプリケーションが JMSBytesMessage の本文として IBM MQ メッセージ全体 (MQRFH2 ヘッダーが存在する場合は、このヘッダーも含む) にアクセスする方法が決定されます。

ALW AMQP クライアント・アプリケーションの開発

AMQP API の IBM MQ サポートにより、IBM MQ 管理者は AMQP チャネルを作成できます。 このチャネルを開始すると、AMQP クライアント・アプリケーションからの接続を受け入れるポート番号が定義されます。

AMQP チャネルは、 AIX, Linux, and Windows システムにインストールできます。 IBM i や z/OS では使用できません。

AMQP 1.0 クライアント・アプリケーションは、AMQP チャネルを使用してキュー・マネージャーに接続できます。

Apache Qpid JMS ライブラリーを使用したアプリケーションの開発

概要

Apache Qpid JMS ライブラリーは、AMQP 1.0 プロトコルを使用して、JMS 2 仕様の実装を提供します。

Apache Qpid JMS は、MQ Light メッセージング API とは異なる方法で、AMQP 1.0 プロトコルのいくつかの側面を使用します。 IBM MQ 9.2 は、IBM MQ AMQP チャネルにサポートを追加して、Apache Qpid JMS アプリケーションが IBM MQ に接続し、パブリッシュ/サブスクライブのメッセージングを行えるようにします。これには、共有サブスクリプションの使用も含まれます。

V 9.3.0 IBM MQ 9.3 は、IBM MQ AMQP チャネルにさらにサポートを追加して、Apache Qpid JMS アプリケーションが IBM MQ に接続し、Point-to-Point メッセージングを実行することができるようにします。 詳しくは、682 ページの『AMQP チャネルにおける Point-to-Point のサポート』を参照してください。

V 9.3.0 IBM MQ 9.3.0 は、IBM MQ AMQP チャネルにさらにキュー・ブラウズ・サポートを追加して、Apache Qpid JMS アプリケーションが IBM MQ に接続して、キューからのメッセージのブラウズを実行できるようにします。 詳しくは、682 ページの『AMQP チャネルにおける Point-to-Point のサポート』を参照してください。

V 9.3.0 IBM MQ 9.3.0 では、AMQP チャネル用に TMPMODEL および TMPQPRFX という 2 つのチャネル属性が追加されています。 これらの属性は、一時キューの作成時に使用されるモデル・キュー用と一時キュー接頭部用です。

他の IBM MQ アプリケーションとの相互通信

Apache Qpid JMS アプリケーションと他の IBM MQ アプリケーションとの間でメッセージを送信することができます。 例えば、Apache Qpid アプリケーションがトピックにメッセージをパブリッシュし、MQ Light アプリケーションが、サブスクリプションを作成することによってそのメッセージを受け取ることができます。

Apache Qpid JMS アプリケーションは、MQSUB API 呼び出しを使用して同じトピックにサブスクライブするなど、従来の IBM MQ アプリケーションでコンシュームされるメッセージをパブリッシュすることもできます。

同様に、Apache Qpid JMS アプリケーションは、従来の IBM MQ アプリケーションがメッセージをパブリッシュする IBM MQ トピックにサブスクライブすることができます。

また、両方のクライアントが同じ共有名とトピック・パターンを指定している限り、Apache Qpid JMS アプリケーションは、MQ Light アプリケーションとサブスクリプションを共有することもできます。

ただし、そのためには、Apache Qpid JMS アプリケーションでは接続時にクライアント ID を使用しないようにしてください。 そうすることで、両方のアプリケーションで使用される IBM MQ サブスクリプション名が確実に同じになります。

重要 : Apache Qpid JMS アプリケーションは、サブスクリプションを IBM MQ JMS アプリケーションと共有することはできません。

Apache Qpid JMS の制約事項

以下の JMS 機能がサポートされます。

- クライアント確認、自動確認、および重複 OK 確認モード (DUPS_OK_ACKNOWLEDGE)
  - 資格情報の使用を問わない接続
  - トピック宛先でのコンシューマーの作成
  - トピック宛先での永続コンシューマーの作成
  - トピック宛先での共有コンシューマーの作成
  - トピック宛先での共有永続コンシューマーの作成
  - クライアント確認モードと自動確認モード
  - メッセージ確認応答とセッション確認応答
  - 永続サブスクリプションからのアンサブスクライブ
  - V 9.3.0 一時キューの作成
  - V 9.3.0 キューまたは一時キュー宛先でのコンシューマーの作成
  - V 9.3.0 JMS MessageListeners
  - V 9.3.0 本体を受信する JMS コンシューマー (`Consumer.receiveBody()` という名前の JMS 2.0 メソッド)
  - V 9.3.0 以下の JMS メッセージ・タイプがサポートされています。
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - V 9.3.0 キューからのメッセージのブラウズ

以下の JMS 機能は AMQP クライアントによってサポートされていません。

- トランザクション化されたセッションおよびトランザクション化された JMSContexts の使用
  - メッセージ・セレクターの使用
  - **nolocal** 属性の使用。
  - トランザクション化されたセッションの使用
  - 送達遅延の使用
  - IBM MQ 9.3.0 で、キューからメッセージをブラウズします。
  - 同じクライアント ID とトピックで複数の永続サブスクリプションまたはコンシューマーを作成する
  - V 9.3.0 JMS 一時トピック
  - AMQP フィルターはサポートされていません。

V 9.3.3 IBM MQ 9.3.3 以降、以下の注記は Continuous Delivery ユーザーには適用されなくなりました。

重要 : V 9.3.0 クライアント確認: メッセージの未決済 AMQP 転送が使用される場合、つまり、メッセージのクライアント確認応答が必要な場合は、クライアント確認応答モードの使用時にメッセージ確認応答をタイムリーに送信することによってタイムリーに解決する必要があります。あるいは、キュー・マネージャー・プロパティー MARKINT (**MsgMarkBrowseInterval**) をより高い値に設定することを検討してください。

MsgMarkBrowseInterval のデフォルト値は 5 秒です。 アプリケーションがこのデフォルト値の範囲内で解決しない場合、重複したメッセージが表示される可能性があります。 メッセージが重複しないようにするには、 **MsgMarkBrowseInterval** の値を適宜増やして、無制限の時間間隔を表

すように NOLIMIT に設定する必要があります。 メッセージが解決される前にアプリケーションが異常終了または切断すると、メッセージは別のアプリケーションで使用可能になります。

詳しくは、 **MsgMarkBrowseInterval** を参照してください。 これはキュー・マネージャーのプロパティーであるため、設定した値は、そのキュー・マネージャーに接続されているすべてのアプリケーションに適用されます。

AMQP では、 **MsgMarkBrowseInterval** はキューに対してのみ有効であり、サブスクリプションに対しては有効ではありません。

サンプル AMQP クライアントのダウンロード

IBM MQ には AMQP クライアントは付属していませんが、 MQ Light クライアントをダウンロードすることも、Apache Qpid ライブラリーに基づいてオープン・ソース AMQP クライアントをダウンロードすることもできます。 詳しくは、 IBM MQ Light および Apache QPID を参照してください。

Apache Qpid ライブラリーに基づくその他のオープン・ソース AMQP クライアントをダウンロードすることもできます。 詳細については、https://qpid.apache.org/index.html を参照してください。

重要 : IBM サポートは、これらのお客様パッケージに対して構成または障害のサポートを提供することができません。使用法に関する質問やコード障害レポートは、それぞれのプロジェクトに送信する必要があります。

IBM MQ への AMQP クライアントのデプロイ

アプリケーションをデプロイする準備が整ったら、その他のエンタープライズ・アプリケーションのモニター機能、信頼性、およびセキュリティー機能のすべてを必要とします。 また、その他のエンタープライズ・アプリケーションとデータを交換することもできます。

AMQP クライアントをデプロイしたら、メッセージを IBM MQ アプリケーションと交換できます。 例えば、AMQP クライアントを使用して JavaScript ストリング・メッセージを送信する場合、 IBM MQ アプリケーションは MQ メッセージを受信します。この場合、MQMD の形式フィールドは MQSTR に設定されます。

AMQP チャネルの管理

その他の MQ チャネルと同じ方法で AMQP チャネルを管理できます。 MQSC コマンド、PCF コマンド・メッセージ、または IBM MQ Explorer を使用して、チャネルの定義、開始、停止、および管理を実行できます。 AMQP チャネルの作成および使用では、クライアントのキュー・マネージャーへの接続を定義および開始するためのサンプル・コマンドが提供されています。

AMQP チャネルが開始されたら、AMQP 1.0 クライアントを接続してそれをテストできます。 例えば、 MQ Light、Apache Qpid Proton、または Apache Qpid JMS です。

関連タスク

AMQP チャネルの作成および使用

AMQP クライアントの保護

ALW MQ Light、 Apache Qpid JMS、および AMQP (Advanced Message Queuing Protocol)

MQ Light クライアント、 Apache Qpid クライアント (Apache Proton など)、および Apache Qpid JMS API は、OASIS Standard AMQP 1.0 ワイヤー・プロトコルに基づいています。 AMQP は、送信側と受信側との間でメッセージが送信される方法を指定します。 アプリケーションが IBM MQ などのメッセージ・ブローカーにメッセージを送信するとき、そのアプリケーションは送信側として機能します。 IBM MQ が AMQP アプリケーションにメッセージを送信するとき、それは送信側として機能します。

AMQP には、以下のようないくつかの利点があります。

- オープンで標準化されたプロトコル
- 他のオープン・ソース AMQP 1.0 クライアントとの互換性

- 多数のオープン・ソース・クライアントの実装が可能

任意の AMQP 1.0 クライアントを AMQP チャネルに接続できますが、トランザクションや複数セッションなど、一部の AMQP 機能はサポートされません。

詳しくは、「AMQP.org の Web サイト」および「OASIS 規格 AMQP 1.0 PDF」を参照してください。

MQ Light および Apache Qpid JMS API には、以下のメッセージング機能があります。

- 最大 1 回のメッセージ・デリバリー
- 1 対多メッセージ・デリバリー
- トピック・ストリング宛先のアドレス指定
- メッセージと宛先の耐久性
- 複数のサブスクライバーによるワークロードの共有を可能にする共有の宛先
- ハングしたクライアントの問題を簡単に解決するためのクライアント・テークオーバー
- 構成可能なメッセージの先読み
- 構成可能なメッセージの確認応答

Apache Qpid JMS API の完全な資料は、「Qpid JMS」を参照してください。

関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW AMQP 1.0 サポート

AMQP チャネルは、AMQP 1.0 準拠のアプリケーションのサポート・レベルを提供します。

AMQP チャネルは AMQP 1.0 プロトコルのサブセットをサポートします。 AMQP 1.0 互換クライアントを IBM MQ AMQP チャネルに接続できます。 AMQP チャネルでサポートされるすべてのメッセージング機能を使用するには、特定の AMQP 1.0 フィールドの値を正しく設定する必要があります。

この情報によって、AMQP フィールドの形式の概要が決まり、AMQP チャネルでサポートされない AMQP 1.0 仕様の機能がリストされます。

AMQP 1.0 仕様の以下の機能は、その利用がサポートされていないか限定されています。

ATTACH フレーム

V 9.3.0

AMQP チャネルは、ATTACH フレームの機能に以下のいずれかが含まれていることを想定しています。

```
topic
temporary queue
queue
shared
```

機能はオブジェクトのタイプを意味し、マルチ機能の場合、機能を選択する優先順位はトピック、一時キュー、キューです。

予期される値が機能に含まれない場合、デフォルトの機能はトピックになります。 その他の機能は、すべて無視されます。

注 : 一部の AMQP クライアントはこれらの機能を設定せず、パブリッシュ/サブスクライブの IBM MQ デフォルト動作を取得します。 例えば、 Quarkus Reactive Messaging AMQP 1.0 コネクターは、バージョン 2.8.0CR1 以降の機能のみを設定します。

V 9.3.0 AMQP チャネルは、ソースまたはターゲットについて、ATTACH フレーム上の `distribution-Mode` に以下のいずれかが含まれていることを予期します。

- 移動

- copy

move は、破壊的な取得を示し、copy はブラウザーを示しています。

注: distribution-Mode が設定されていない場合、または コピー以外に設定されている場合は、移動 が想定されます。

リンク名

AMQP チャネルは、AMQP リンクの名前が以下の 5 つの形式のいずれかに従っていることを想定しています。

- プレーン・トピック (パブリッシュおよびサブスクライブ用)
  - メッセージのパブリッシュ: プレーン・トピック・ストリング (リンク名 "/sports/football"など) を使用すると、メッセージが /sports/football トピックにパブリッシュされます。
  - メッセージを受信するためにトピックにサブスクライブする場合: プレーン・トピック・ストリング (例えば、リンク名 "/sports/football" を指定すると、 /sports/football トピックにサブスクリプションが定義されます。
- 専用冗長トピック (サブスクライブ用)
  - "private:*topic string*" の形式でプライベート・サブスクリプションを記述する詳細トピック・ストリング (例: "private:/sports/football")。 この動作は、プレーン・トピック・ストリングと同じです。 private 宣言は、特定の AMQP クライアントに固有のサブスクリプションを、クライアント間で共有されるサブスクリプションと区別します。
- 共有冗長トピック (サブスクライブ用)
  - "share:*share name*:*topic string*" という形式で共有サブスクリプションを記述する詳細トピック・ストリング (例: "share:bbc:/sports/football")。
- V 9.3.0 キュー (プロデューサーとコンシューマー向けの point-to-point メッセージング用)
  - メッセージを送信するプロデューサー。キュー名の文字列を指定すると、プロデューサーはキューでメッセージを送信します。
  - メッセージを受信するコンシューマー。キュー名の文字列を指定すると、コンシューマーはキューからメッセージを受信します。
- V 9.3.0 ブランク (一時キューの point-to-point メッセージング用)
  - 一時キューでメッセージを送信するプロデューサー。ブランクを指定すると、プロデューサーは一時キューでメッセージを送信します。
  - 一時キューでメッセージを受信するコンシューマー。ブランクを指定すると、コンシューマーは一時キューからメッセージを受信します。

AMQP メッセージと IBM MQ メッセージとの間のマッピング方法について詳しくは、686 ページの『AMQP フィールドの IBM MQ フィールドへのマッピング (着信メッセージ)』を参照してください。

トピック・ストリング、共有名、およびクライアント ID の最大長

トピック・ストリング、共有名、およびクライアント ID の長さは 10237 バイト以内でなければなりません。 また、クライアント ID の最大長は 256 文字です。

これらの最大長は、以下を意味します。

- 共有名が短い場合は、トピック・ストリングを非常に長くすることができる。
- トピック・ストリングが短い場合は、共有名を長くすることができる。

コンテナー ID

AMQP チャネルは、AMQP オープン・パフォーマンスのコンテナー ID に固有の AMQP クライアント ID が含まれていることを予期します。 AMQP クライアント ID の最大長は 256 文字で、ID には英数字、パーセント記号 (%)、スラッシュ (/)、ピリオド (.)、および下線 (_) を使用できます。

セッション数

AMQP チャネルは、単一 AMQP セッションのみサポートしています。 AMQP クライアントが複数の AMQP セッションを作成しようとすると、エラー・メッセージが出て、チャネルから切断されます。

トランザクション

AMQP チャネルは AMQP トランザクションをサポートしていません。 AMQP 接続フレームが新規トランザクションを整合しようとしたり、AMQP 転送フレームが新規トランザクションを宣言しようとすると、エラー・メッセージが出て拒否されます。

送達状態

AMQP チャネルは、ファイル属性指定フレームの送達状態として Accepted、Released、または Modified のみサポートしています。 Modified 状態が使用されている場合、AMQP チャネルは undeliverable-here オプションをサポートしないことに注意してください。

関連タスク

AMQP チャネルの作成および使用

AMQP クライアントの保護

V 9.3.0 ALW AMQP チャネルにおける Point-to-Point のサポート

IBM MQ AMQP チャネルは、キューへのメッセージの送信とキューからのメッセージの受信をサポートします。

Apache Qpid™ JMS ライブラリーなどの AMQP クライアントは、AMQP 接続フレームの送信時に queue 機能または `temporary-queue` 機能を要求します。 これらの機能により、AMQP チャネルは、オブジェクトをキュー、一時キュー、またはトピックとして識別できます。 queue 機能または temporary-queue 機能のいずれか、またはどちらの機能もない場合、要求はトピックに対するものと見なされます。

IBM MQ AMQP チャネルは、以下のものに対するキュー・タイプをサポートします。

キューの送受信
メッセージをキューに送信して、キューから取り込むことができます。 メッセージの取り込みの場合、同期モードと非同期モードの両方がサポートされます。

V 9.3.0 キュー参照メッセージ
メッセージをキューに入れてキューからメッセージを取得するだけでなく、メッセージをキューから参照することもできます。

一時キューのサポート
メッセージを一時キューに送信して、一時キューから取り込むことができます。 一時キューの作成に使用されたのと同じ一時キュー・オブジェクトが一時キューの削除にも使用される場合は、一時キューの削除がサポートされていることに注意してください。

一時キューの作成時に、SYSTEM.DEFAULT.MODEL.QUEUE が使用され、一時キューの接頭部は `AMQP.*` になります。

デフォルトで、SYSTEM.DEFAULT.MODEL.QUEUE は一時動的キューですが、SYSTEM.DEFAULT.MODEL.QUEUE キューの **Definition type** プロパティーを使用して、キューを永続動的キューに変更することができます。

永続動的キュー
AMQP クライアント (Apache Qpid JMS ライブラリーなど) が、 **closed** 属性が *true* に設定された `detach` フレームを使用して要求を送信すると、永続動的キューが削除されます。

重要 :

Qpid JMS の動作:

Qpid JMS API コマンド (例、`javax.jms.TemporaryQueue.delete()` メソッド) を呼び出して、使用後にキューを破棄する必要があります。このプロセスにより、キューに存在するメッセージもクリアされます。

このようなコマンドを発行しない場合、接続を閉じても、キューにはメッセージが残ったままになります。

-

一時動的キュー

AMQP クライアントが接続を閉じると、一時動的キューが削除されます。

重要 :

Qpid JMS の動作:

Qpid JMS API コマンド (例、`javax.jms.TemporaryQueue.delete()` メソッド) を呼び出し、JMS 接続を閉じるか、または接続が切断されると、キューは削除され、すべてのメッセージが失われます。

`javax.jms.Session.createTemporaryQueue()` メソッドを使用して一時キューが作成されていても、 JMS セッション自体を閉じても、一時キューは削除されません。

-

関連タスク

AMQP チャネルの作成および使用

AMQP クライアントの保護

ALW AMQP および IBM MQ メッセージ・フィールドのマッピング

AMQP メッセージは、ヘッダー、デリバリー・アノテーション、メッセージ・アノテーション、プロパティー、アプリケーション・プロパティー、本文、フッターで構成されます。

AMQP メッセージは、以下の部分から構成されています。

ヘッダー

オプションのヘッダーには、メッセージの 5 つの固定属性が含まれます。

- **durable** - 耐久性の必要条件を指定します。
- **priority** - 相対メッセージ優先順位。
- **ttl** - 存続時間 (ミリ秒単位)。
- **first-acquirer** - これが true の場合、他のリンクによってメッセージは獲得されていません。
- **delivery-count** - 以前に失敗した送達試行回数。

デリバリー・アノテーション

オプション。 さまざまな対象者に対するメッセージの非標準ヘッダー属性を指定します。 デリバリー・アノテーションは、送信ピアから受信ピアに情報を伝達します。

メッセージ・アノテーション

オプション。 さまざまな対象者に対するメッセージの非標準ヘッダー属性を指定します。 メッセージ・アノテーションのセクションは、インフラストラクチャーを対象とするメッセージのプロパティーに使用され、すべての送達手順に伝搬される必要があります。

プロパティー

オプション。 この部分は、MQ メッセージ記述子と同等です。 これには、以下の固定フィールドが含まれています。

- **message-id** - アプリケーションのメッセージ ID
- **user-id** - 作成しているユーザーの ID
- **to** - メッセージの宛先ノードのアドレス
- **subject** - メッセージの件名
- **reply-to** - 送信が応答するノード

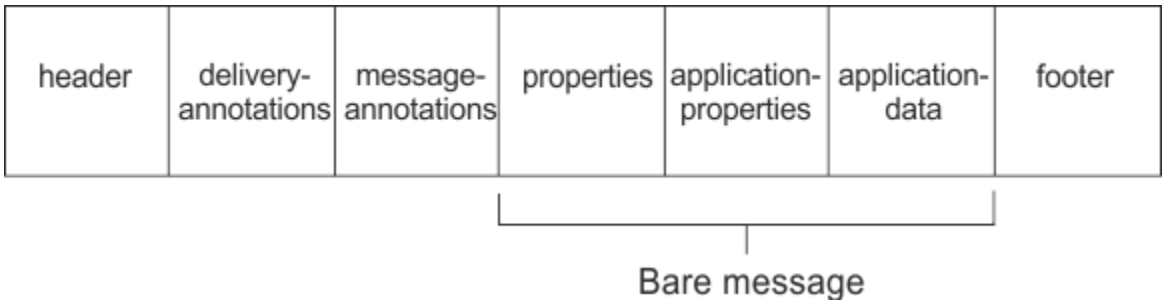
- **correlation-id** - アプリケーション相関 ID
- **content-type** - MIME コンテンツ・タイプ
- **content-encoding** - MIME コンテンツ・タイプ。 content-type の修飾子として使用します。
- **absolute-expiry-time** - このメッセージの有効期限が切れたと見なされる時間
- **creation-time** - このメッセージが作成された時間
- **group-id** - このメッセージが所属するグループ
- **group-sequence** - グループ内のこのメッセージのシーケンス番号
- **reply-to-group-id** - 応答メッセージが所属するグループ

アプリケーション・プロパティー
MQ メッセージ・プロパティーと同等です。

Body
MQ ユーザー・ペイロードと同等です。

フッター
オプション。 フッターは、ベア・メッセージ全体が作成された、または表示された後にのみ計算または評価を行えるメッセージまたは送達に関する詳細に使用されます (例えば、メッセージ・ハッシュ、HMAC、シグニチャー、暗号化の詳細)。

AMQP メッセージ形式を次の図に示します。



プロパティー、アプリケーション・プロパティー、およびアプリケーション・データ・パーツは、"ベア・メッセージ"として知られています。 これは、送信者によって送信されるメッセージで、変更できません。受信者には、ヘッダー、フッター、デリバリー・アノテーション、メッセージ・アノテーションを含むメッセージ全体が表示されます。

AMQP 1.0 メッセージ形式の詳細な説明については、OASIS 標準 (https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf) を参照してください。

関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW IBM MQ フィールドから AMQP フィールドへのマッピング (出力メッセージ)

IBM MQ メッセージが公開され、IBM MQ によって AMQP コンシューマーに送信されると、IBM MQ メッセージの属性の一部が、同等の AMQP メッセージ属性に伝搬されます。

ヘッダー

ヘッダーは、ヘッダー内の 5 つのフィールドの 1 つに、デフォルト以外の値が含まれている場合にのみ組み込まれます。 デフォルト以外の値を持つフィールドのみ、ヘッダーに含められます。 5 つのヘッダー・フィールドは、最初は同等の `mq_amqp.Hdr` プロパティー (設定されている場合) から取得され、以下の表に示すように変更されます。

表 *101.* ヘッダー・フィールドのマッピング

フィールド	デフォルト値	値
永続	false	MQMD.Persistence が MQPER_PERSISTENT に設定されている場合は true、それ以外の場合は false。
priority	4	mq_amqp.Hdr.Pri が設定されている場合はその値から。それ以外の場合、MQMD.Priority が設定されている場合は、その値からマップされます。 どちらも設定されていない場合は、4 に設定されます。
ttl	n/a	MQMD.Expiry (ミリ秒単位)。 MQMD.Expiry の値が MQEI_UNLIMITED の場合、AMQP ttl フィールドの最大値に設定されます。
first-acquirer	false	mq_amqp.Hdr.Fac が設定されている場合はその値からマップされ、それ以外の場合は false になります。
delivery-count	0	mq_amqp.Hdr.Dct が設定されている場合はその値からマップされ、それ以外の場合は 0 になります。

送達注釈 (delivery-annotation)

AMQP チャネルにより、必要に応じて設定されます。

メッセージ注釈 (message-annotation)

含まれない

プロパティー

同等の mq_amqp.Prp プロパティーが設定されている場合、「**プロパティー**」には変更なしでその値が取り込まれます。 メッセージがもともと AMQP メッセージではない (つまり、PutApplType が MQAT_AMQP ではない) 場合、プロパティー・セクションは以下の表で説明するように生成されます。

表 *102.* プロパティー・フィールドのマッピング

名前	値
message-id	MQMD.MsgId はバイナリーとして設定されます。
ユーザー ID	UTF-8 形式の MQMD.UserIdentifier は、ネットワーク・バイト・オーダーでバイナリーとして設定。
へ	メッセージの取得元のキュー、またはパブリケーションの場合はトピック・ストリング。
件名	設定なし。
応答先	MQMD.ReplyToQ (ブランク以外の場合)。それ以外の場合は、設定なし。
相関 ID (correlation-id)	MQMD.CorrelId は、ブランク以外の場合、バイナリーとして設定されます。 それ以外の場合は、設定なし。
コンテンツ・タイプ	設定なし。

表 102. プロパティー・フィールドのマッピング (続き)	
名前	**値**
コンテンツ・エンコーディング (content-encoding)	設定なし。
絶対有効期限 (absolute-expiry-time)	設定なし。
creation-time	MQMD.PutDate フィールドと MQMD.PutTime フィールドがタイム・スタンプ生成のために使用されます。
group-id	設定なし。
グループ順序	設定なし。
返信先グループ ID (reply-to-group-id)	設定なし。

アプリケーション・プロパティー (application-properties)

"usr" グループ内のすべての IBM MQ プロパティーが **application-properties** として追加されます。

body

AMQP チャネルは、変換を伴う get を実行して、IBM MQ ペイロードを UTF-8 に変換します。

IBM MQ ペイロードに AMQP メッセージが含まれない場合、IBM MQ ペイロードは本文でフォーマット MQFMT_STRING の単一ストリングのデータ・セクションとして設定されます (UTF-8 への変換が成功した場合)。それ以外の場合は、単一のバイナリー・データ・セクションとして設定されます。

AMQP フォーマットのメッセージが組み込まれる場合、これが本文として設定されます。 AMQP メッセージに先行する IBM MQ ヘッダー (メッセージ・ハンドルに返されるメッセージ・プロパティーを除外する) は、本文が AMQP シーケンスの場合、バイナリー値として前に付加されます。 それ以外の場合、IBM MQ ヘッダーは廃棄されます。

フッター

フッターは含まれません。

関連タスク

AMQP チャネルの作成および使用

AMQP クライアントの保護

関連資料

MQMD - メッセージ記述子

ALW AMQP フィールドの IBM MQ フィールドへのマッピング (着信メッセージ)

AMQP チャネルがメッセージを受信して、IBM MQ に配置すると、AMQP メッセージの一部の属性が同等の IBM MQ メッセージ属性に伝搬されます。

着信 AMQP メッセージをマッピングする場合、以下の制限が適用されます。

- プロパティー・パーツの message-id または correlation-id フィールドが uuid または ulong である場合、メッセージは拒否されます。
- message-annotations があると、メッセージはリジェクトされます。
- delivery-annotations セクションと footer セクションは許可されますが、 IBM MQ メッセージには伝搬されません。

以下のサブセクションは、AMQP メッセージの IBM MQ 式を示します。

メッセージ記述子

表 *103. AMQP* メッセージのメッセージ記述子

フィールド	値
StrucId	MQMD_STRUC_ID
バージョン	MQMD_VERSION_1
レポート	MQRO_NONE
MsgType	MQMT_DATAGRAM
Expiry	AMQP メッセージ・ヘッダーの `ttl` フィールドから取得される値
Feedback	MQFB_NONE
Encoding	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Format	「ペイロード」を参照。
優先順位	AMQP メッセージ・ヘッダーの `priority` フィールドから取得される値。 設定されている場合、最大値は 9 に制限されます。 設定されていない場合は、デフォルト値の 4 が取得されます。
Persistence	AMQP メッセージ・ヘッダーの `durable` フィールドが true に設定されている場合は、MQPER_PERSISTENT に設定します。 それ以外の場合は、MQPER_NOT_PERSISTENT に設定します。
MagId	キュー・マネージャーが固有の 24 バイトの MsgId を割り振ります。
CorrelId	AMQP プロパティーの `correlation-id` フィールドから取得される値 (設定されている場合)。 24 バイトのバイナリー値に設定されます。 それ以外の場合は、MQCI_NONE/ に設定されます。
BackoutCount	0
ReplyToQ	V 9.3.0 AMQP プロパティーの `reply-to` フィールドから取得される値 (設定されている場合)。 それ以外の場合は、"" に設定されます。
ReplyToQMgr	""
V 9.3.0 レポート	AMQP アプリケーション・プロパティーに設定されているすべての JMS IBM レポート・プロパティーから得られた値。
UserIdentifier	AMQP チャネルに接続する認証ユーザーの ID に設定されます。
AccountingToken	MQACT_NONE
ApplIdentityData	16 進数ストリング。 AMQP チャネルの MQ 接続 ID の末尾 8 バイトに設定されます。
PutApplType	MQAT_AMQP
PutApplName	
PutDate	AMQP プロパティーの `creation-time` フィールドから取得される値 (設定されている場合)。 それ以外の場合は、現在日付に設定されます。
PutTime	AMQP プロパティーの `creation-time` フィールドから取得される値 (設定されている場合)。 それ以外の場合は、現在時刻に設定されます。

表 103. AMQP メッセージのメッセージ記述子 (続き)	
フィールド	**値**
ApplOriginData	""

メッセージ・プロパティー

メッセージ・プロパティーを設定する理由は、次の 2 つです。

- AMQP メッセージの一部が、メッセージのペイロードに影響を与えずにキュー・マネージャーを通過できるようにするため。
- `application-properties` を選択できるようにします。

以下の表に、AMQP メッセージから設定されるプロパティーを示します。

表 104. AMQP メッセージ・プロパティー			
プロパティー名	**MQRFH2 名**	**タイプ**	**説明**
AMQPListener	mq_amqp.Lis	MQTYPE_STRING	AMQP チャネル用の識別ストリング。 これは、対象となるパーティーがメッセージを配置するバージョン (例えば、問題診断の際のサービス・チーム) を識別できるようにするため、メッセージ生成に使用されます。 この値は、キュー・マネージャーによって検証されず、外部に文書化してはなりません。
AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	AMQP メッセージのバージョン。 指定されない場合、「1.0」が想定されます。 この値は、キュー・マネージャーによって検証されません。
AMQPClient	mq_amqp.Cli	MQTYPE_STRING	API 用の識別ストリング。 これは、対象となるパーティーがメッセージを配置するバージョン (例えば、問題診断の際のサービス・チーム) を識別できるようにするため、AMQP メッセージのチャネルへの送信に使用されます。 この値は、キュー・マネージャーによって検証されず、外部に文書化してはなりません。
AMQPDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	AMQP メッセージ・ヘッダーの `durable` フィールドの値 (設定されている場合)。
AMQPPriority	mq_amqp.Hdr.Pri	MQTYPE_INT32	AMQP メッセージ・ヘッダーの `priority` フィールドの値 (設定されている場合)。
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	AMQP メッセージ・ヘッダーの `ttl` フィールドの値 (設定されている場合)。
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLEAN	AMQP メッセージ・ヘッダーの `first-acquirer` フィールドの値 (設定されている場合)。

表 *104. AMQP* メッセージ・プロパティー *(*続き*)*

プロパティー名	MQRFH2 名	タイプ	説明
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	AMQP メッセージ・ヘッダーの delivery-count フィールドの値 (設定されている場合)。
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_STRING	AMQP プロパティーの message-id フィールドの値 (ストリングとして設定されている場合)。
		MQTYPE_BYTE_STRING	AMQP プロパティーの message-id フィールドの値 (バイト・ストリングとして設定されている場合)。
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	AMQP プロパティーの user-id フィールドの値 (設定されている場合)。
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	AMQP プロパティーの to フィールドの値 (設定されている場合)。
AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	AMQP プロパティーの subject フィールドの値 (設定されている場合)。
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	AMQP プロパティーの reply-to フィールドの値 (設定されている場合)。
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	AMQP プロパティーの correlation-id フィールドの値 (ストリングとして設定されている場合)。
		MQTYPE_BYTE_STRING	AMQP プロパティーの correlation-id フィールドの値 (バイト・ストリングとして設定されている場合)。
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	AMQP プロパティーの content-type フィールドの値 (設定されている場合)。
AMQPContentEncoding	mq_amqp.Prp.Cneing	MQTYPE_STRING	AMQP プロパティーの content-encoding フィールドの値 (設定されている場合)。
AMQPAbsoluteExpiryTime	mq_amqp.Prp.Aet	MQTYPE_STRING	AMQP プロパティーの absolute-expiry-time フィールドの値 (設定されている場合)。
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	AMQP プロパティーの creation-time フィールドの値 (設定されている場合)。
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	AMQP プロパティーの group-id フィールドの値 (設定されている場合)。

表 104. AMQP メッセージ・プロパティー (続き)			
プロパティー名	**MQRFH2 名**	**タイプ**	**説明**
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	AMQP プロパティーの `group-sequence` フィールドの値 (設定されている場合)。
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_STRING	AMQP プロパティーの `reply-to-group-id` フィールドの値 (設定されている場合)。

AMQP メッセージの各「アプリケーション・プロパティー (application-properties)」は、IBM MQ メッセージ・プロパティーとして設定されます。 `application-properties` セクションは同じバイトごとに再構成する必要があるため、以下の制約事項が適用されます。

- MQSETMP 検証コードによってアプリケーション・プロパティーが拒否されると、メッセージが拒否されます。 以下に例を示します。
  - プロパティー名は MQ_MAX_PROPERTY_NAME_LENGTH の長さに制限されます。
  - プロパティー名は、Java Language Specification for Java Identifiers によって定義された規則に従う必要があります。
  - 設定可能な文書化された JMS プロパティーを例外として、プロパティー名は `JMS` または `usr.JMS` で始まっていてはなりません。
  - プロパティー名は SQL キーワードであってはなりません。
- Unicode 文字 U+002E (".") を含むアプリケーション・プロパティーは、メッセージが拒否される原因となります。 プロパティーは、JMS が使用するプロパティーの「usr」グループで表現可能でなければなりません。
- ヌル、ブール、バイト、短整数、整数、長整数、浮動小数点、倍精度、バイナリー、およびストリングのプロパティーのみがサポートされます。 その他の型のアプリケーション・プロパティーは、メッセージが拒否される原因となります。

V 9.3.0 `application-properties` を使用して、以下の JMS プロパティーを設定できます。

- JMS_IBM_REPORT_EXCEPTION
- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_REPORT_PASS_CORREL_ID
- JMS_IBM_REPORT_DISCARD_MSG

プロパティー名と値は、対応する 162 ページの『JMS プロバイダー固有のフィールドのマッピング』 詳細と一貫性があり、無効な値は無視されることに注意してください。

ペイロード

- 単一のバイナリー・データ・セクションを持つ AMQP body の場合、バイナリー・データ (AMQP ビットを除く) は、MQFMT_NONE の形式で IBM MQ ペイロードとして書き込まれます。
- 単一ストリング・データ・セクションを持つ AMQP body の場合、ストリング・データ (AMQP ビットを除く) は MQFMT_STRING 形式の IBM MQ ペイロードとして書き込まれます。
- それ以外の場合、AMQP body はペイロードをそのまま MQFMT_AMQP の形式で形成します。

関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW メッセージ送達の信頼性

このセクションでは、 MQ Light API と Apache Qpid JMS の信頼性機能を比較します。
関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW MQ Light メッセージの信頼性

MQ Light API には、AMQP アプリケーションとの間のメッセージ送達の信頼性を制御できるようにする 4 つの機能があります。

次のとおりです。

- 691 ページの『メッセージのサービスの品質 (QOS)』
- 692 ページの『サブスクライバーの自動確認』
- 692 ページの『サブスクリプション存続時間』
- 692 ページの『メッセージの持続性』

メッセージのサービスの品質 (QOS)

MQ Light API は、以下の 2 つのサービスの品質を提供します。

- 最高 1 回
- 最低 1 回

パブリッシャーとサブスクライバーで使用するサービスの品質を選択することができます。

MQ Light クライアントを使用している場合は、クライアントまたはサブスクライブの **qos** オプションを *QOS* 接続のオンへの変更 または *QOS AT_LEAST_ONCE (QOS AT_LEAST_ONCE)*に設定します。

別の AMQP クライアントを使用する場合は、達成したいサービスの品質に応じて、転送フレーム (パブリッシャーの場合) の **settled** 属性、または処理フレーム (サブスクライバーの場合) を 真 または 偽に設定します。

サービス品質は、メッセージが会話の sending 側から破棄されるタイミングを決定します。

パブリッシュ

パブリッシャーが **QOS** *0* を (多くても 1 回) 選択した場合、パブリッシャーはキュー・マネージャーからの確認応答を待たずに、メッセージのコピーを破棄します。

送信が完了する前にキュー・マネージャーへの接続に障害が起こると、メッセージがサブスクライバーによって受信されない場合があります。

パブリッシャーが **QOS** *1* を (少なくとも 1 回) 選択すると、パブリッシャーは、メッセージのコピーを破棄する前に、サブスクライバー・キューにメッセージが書き込まれたことをキュー・マネージャーが確認するまで待機します。

送信中にキュー・マネージャーへの接続に障害が起こると、パブリッシャーは、キュー・マネージャーへの再接続が行われた後にメッセージを再送信します。

サブスクライブ

サブスクライバーが **QOS** *0* を選択した場合、キュー・マネージャーはサブスクライバーからの確認応答を待たずに、メッセージのコピーを破棄します。

サブスクライバーがメッセージを受信する前にサブスクライバーへの接続に障害が起こると、そのメッセージは失われる可能性があります。

サブスクライバーが **QOS** *1* を選択すると、キュー・マネージャーはサブスクライバーからの確認応答を待ってから、メッセージのコピーを破棄します。 V 9.3.3 IBM MQ 9.3.3 以降、パフォーマンスを向上させるために、確認済みメッセージがバッチで削除されます。 詳しくは、694 ページの『確認済み AMQP メッセージをキューからバッチで削除』を参照してください。

サブスクライバーがメッセージを受信する前にサブスクライバーへの接続に障害が起こると、そのメッセージはキュー・マネージャーによって保持されます。 キュー・マネージャーは、キュー・マネージャーの再接続時にサブスクライバーにメッセージを再送信します。サブスクリプションが共用サブスクリプションである場合は別のサブスクライバーに再送信されます。

サブスクライバーの自動確認

サブスクライバーが **QOS** *1* を (少なくとも 1 回) 選択した場合、キュー・マネージャーがそのコピーを破棄する前に、各メッセージの受信を確認する必要があります。 サブスクライバーは、メッセージを確認するタイミングを決定できます。

auto-confirm を 真に設定すると、 MQ Light クライアントは、ネットワークを介してメッセージを正常に受信した後、各メッセージの配信を自動的に確認します。

これにより、ネットワーク障害が発生した場合、メッセージがアプリケーションに再送達されます。 ただし、MQ Light クライアントがメッセージの確認応答を出してからアプリケーションがそのメッセージを処理するまでの間にアプリケーションで障害が起こった場合は、アプリケーションがメッセージを失う可能性があります。

auto-confirm を 偽に設定すると、 MQ Light クライアントはメッセージの送達を自動的に確認しませんが、いつ確認するかを決定するためにアプリケーションに任せます。

これにより、アプリケーションは、メッセージの処理が完了して廃棄可能になったことを示す確認応答をキュー・マネージャーに出す前に、外部リソース (データベースやファイルなど) を更新できるようになります。

サブスクリプション存続時間

アプリケーションがサブスクライブするときに、そのアプリケーションは、サブスクリプション、およびそのサブスクリプションでメッセージが保管される宛先を、アプリケーションの切断後も引き続き存続させるかどうかを選択します。

MQ Light サブスクライブ・オプション **ttl** は、アプリケーションの切断後にサブスクリプションが存続する時間 (ミリ秒単位) を指定するために使用します。 アプリケーションがその時間より前に再接続されると、サブスクリプションが再開され、アプリケーションはそのサブスクリプションによるメッセージを引き続きコンシュームできます。

アプリケーションの再接続が行われずに存続時間の期間が経過すると、サブスクリプションは除去され、その宛先に保管されていたメッセージは、永続メッセージの場合でもすべて失われます。

メッセージが失われないようにすることが重要な場合、停止中にメッセージが失われることがないように、十分高い値の存続時間をアプリケーションに指定する必要があります。

メッセージの持続性

メッセージの持続性は、アプリケーションのパブリッシュとサブスクライブ、および IBM MQ トピック・オブジェクトの構成によって制御されます。

AMQP サブスクライバーが **QOS** *0* を (最大で 1 回) 使用し、非永続サブスクリプションを作成する場合、AMQP チャネルは、以下のテキストで説明されている他のオプションに関係なく、常に非永続メッセージをサブスクライバー・キューに書き込みます。

キュー・マネージャーが停止すると、サブスクリプションとメッセージの両方が失われることに注意してください。

AMQP パブリッシャーが AMQP **durable** ヘッダーを 真に設定すると、AMQP チャネルは永続メッセージをサブスクライバー・キューに書き込みます。

何らかの理由でキュー・マネージャーが停止した場合でも、サブスクライバーは、キュー・マネージャーの再始動時に引き続きメッセージを使用できます。

durable ヘッダーが設定されていない場合、AMQP チャネルは、関連する IBM MQ トピック・オブジェクトの **DEFPSIST** 属性に基づいて、パブリッシュされたメッセージの持続性を選択します。

デフォルトでは、これは SYSTEM.BASE.TOPIC であり、 いいえ (非永続) の **DEFPSIST** 属性を使用します。

重要 : 新しいバージョンの MQ Light クライアントでは、AMQP durable ヘッダーの設定はサポートされていません。

関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW Apache Qpid JMS のメッセージ信頼性

Apache Qpid™ JMS ライブラリーには、AMQP アプリケーションとの間のメッセージ送達の信頼性を制御できるようにする 4 つの機能があります。

これらは以下のとおりです。

- 693 ページの『パブリッシュ』 V 9.3.0 /point to point メッセージのプロデューサー
  - メッセージの有効期限
  - メッセージの持続性
- 693 ページの『サブスクライブ』
  - サブスクリプション永続性
  - セッション確認応答モード V 9.3.0 (コンシューマーの point to point メッセージングにも適用可能)

パブリッシュ

メッセージの有効期限

JMS プロデューサーの存続時間の値を設定すると、そのメッセージ・プロデューサーによってパブリッシュされるメッセージに指定された有効期限時刻に影響します。

JMS プロデューサーの存続時間の値が、有効期限が切れる前にメッセージがコンシュームされる十分な大きさになるようにします。

あるいは、存続時間の値を設定しないままにすると、メッセージがサブスクリプション・キューから有効期限切れになるのを防ぐことができます。

メッセージの持続性

JMS メッセージ・プロデューサーの送達モードを設定すると、指定されたトピックにパブリッシュされる IBM MQ メッセージの持続性が設定されます。

キュー・マネージャーが終了した場合、およびキュー・マネージャーで障害が発生した場合に保持する必要があるメッセージには、必ず **DeliveryMode**.PERSISTENT を使用してください。

サブスクライブ

サブスクリプション永続性

AMQP チャネルは、JMS のコンシューマーの作成メソッドの永続バージョンを使用することにより、永続サブスクリプションの作成をサポートします。

- **createDurableConsumer()**

- **createSharedDurableConsumer()**

セッション確認応答モード

コンシュームされたメッセージが IBM MQ サブスクリプション・キューから削除される前に完全に処理されたことを保証するには、 **Session** を使用して JMS セッションを作成します。CLIENT_ACKNOWLEDGE モードで、 **message.acknowledge()** メソッドを使用して、このメッセージと、このセッションで以前に受信した他のメッセージを確認します。

関連概念

AMQP クライアント・アプリケーションの開発

AMQP API の IBM MQ サポートにより、IBM MQ 管理者は AMQP チャネルを作成できます。 このチャネルを開始すると、AMQP クライアント・アプリケーションからの接続を受け入れるポート番号が定義されます。

V 9.3.3 確認済み AMQP メッセージをキューからバッチで削除

AMQP アプリケーションが QOS_AT_LEAST_ONCE (1) メッセージ配信を使用している場合、AMQP サービスは、アプリケーションからの確認応答を待機してから、そのメッセージをアプリケーションに送信した後に保持しているメッセージのコピーを破棄します。 IBM MQ 9.3.3 以降、確認済みのメッセージは、個別にではなくバッチでキューから削除されるため、パフォーマンスが向上します。

このタスクについて

Long Term Support および IBM MQ 9.3.3 より前の Continuous Delivery の場合、各メッセージは個別にキューから除去されます。

IBM MQ 9.3.3 以降、パフォーマンスを向上させるために、2 つのシステム・プロパティー **com.ibm.mq.AMQP.BATCHSZ** および **com.ibm.mq.AMQP.BATCHINT** を使用して、確認応答の処理をバッチで微調整できます。

com.ibm.mq.AMQP.BATCHSZ
この属性は、AMQP サービスがメッセージを削除する前に受信する確認応答の最大数を定義します。この値の範囲は 1 から 9999 までです。 無効な数値が設定された場合、または指定された数値が範囲外の場合は、デフォルト値の 50 が使用されます。

バッチ・サイズは、メッセージの転送方法には影響しません。 メッセージは常に個別に転送されますが、その後、AMQP サービスが確認応答を受信した後にバッチで削除されます。 バッチの実際のサイズは、 **com.ibm.mq.AMQP.BATCHINT** で指定された値より小さくすることができます。 例えば、 **com.ibm.mq.AMQP.BATCHINT** 属性によって設定された期間が経過すると、バッチが完了します。

com.ibm.mq.AMQP.BATCHINT
この属性は、AMQP サービスが確認済みメッセージをキューに保持する時間をミリ秒単位で定義します。 バッチが満杯でない場合、バッチはこの期間の後にクリアされます。 1 から 999 999 999 までの任意のミリ秒数を指定できます。 デフォルト値は 50 です。 この属性に値を指定しない場合は、デフォルト値の 50 が使用されます。

注 :

1. AMQP サービスがメッセージを破棄する前に確認応答を待機するかどうかは、アプリケーションがメッセージ配信に使用している以下の 2 つのサービス品質のいずれかによって決まります。
   - QOS (QOS_AT_MOST_ONCE 用) (0)

     AMQP アプリケーションがこのサービス品質を使用している場合、メッセージは確認応答されません。そのため、AMQP サービスは、メッセージをアプリケーションに送信した後、確認応答を待たずにメッセージを破棄します。
   - QOS AT_LEAST_ONCE (1)

     AMQP アプリケーションは、このサービス品質を使用している場合、メッセージを確認します。したがって、AMQP サービスは、各メッセージをアプリケーションに送信した後、アプリケーションから確認応答を受信するまで、各メッセージのコピーを保持します。 アプリケーションがメッセージを確

認する前に接続を切断するか切断すると、メッセージは他のアプリケーションで使用できるようになります。 AMQP サービスは、確認されるまでキューからメッセージを削除しません。

2. MQ Appliance **com.ibm.mq.AMQP.BATCHSZ** および **com.ibm.mq.AMQP.BATCHINT** システム・プロパティーは、IBM MQ Appliance では適用されません。 IBM MQ Appliance では、デフォルト値 50 が使用されます。

.

手順

com.ibm.mq.AMQP.BATCHSZ および **com.ibm.mq.AMQP.BATCHINT** システム・プロパティーを使用して、確認応答の処理をバッチ単位で微調整します。

IBM MQ 9.3.3 以降、キュー・マネージャーの作成時に、amqp_java.properties ファイルにシステム・プロパティーの以下のデフォルト値が含まれるようになりました。

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

消費されるメッセージ速度に応じて、パフォーマンスを向上させるために確認応答の処理をバッチで微調整することができます。 マイグレーションされたキュー・マネージャーの amqp_java.properties ファイルには、これらのプロパティーがありません。 したがって、マイグレーションされたキュー・マネージャーの場合、またはプロパティーが設定されていない場合は、デフォルト値が使用されます。 これらのプロパティーを追加して、最適化されたパフォーマンスを得るために値を微調整することができます。

以下のいずれかの条件が満たされると、確認済みメッセージはバッチで削除されます。

- 確認済みメッセージの数が **com.ibm.mq.AMQP.BATCHSZ** に達しました。
- バッチの開始後に **com.ibm.mq.AMQP.BATCHINT** を超過しました。
- アプリケーションは、前の 2 つの条件が満たされる前に、キューまたはトピックを切断またはクローズします。

ALW IBM MQ を使用した AMQP クライアントのトポロジー

IBM MQ と連携する AMQP クライアントの開発に役立つトポロジーの例。

関連タスク

AMQP チャネルの作成および使用

AMQP クライアントの保護

ALW IBM MQ 上で通信する AMQP クライアント

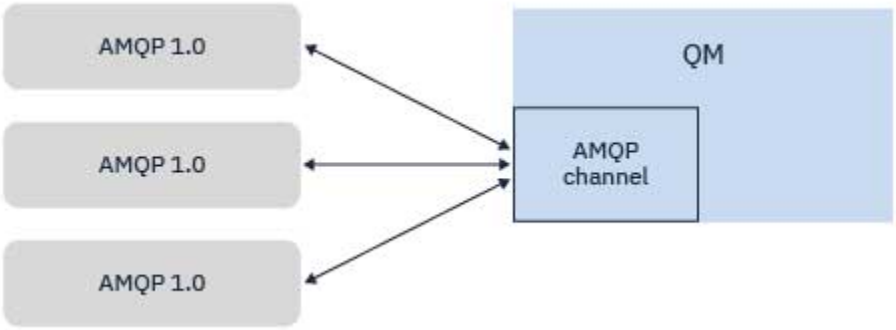
IBM MQ は、AMQP 1.0 に準拠するすべてのアプリケーションのメッセージング・プロバイダーとして使用できます。 任意の AMQP 1.0 クライアントを AMQP チャネルに接続できますが、トランザクションや複数セッションなど、一部の AMQP 機能はサポートされません。

1 つ以上の AMQP チャネルを定義することにより、AMQP 1.0 クライアントはキュー・マネージャーに接続し、トピック・ストリングにメッセージを送信できます。 クライアントは、トピック・パターンにサブスクライブして、そのパターンと一致するメッセージを受信することもできます。

以下のシナリオでは、メッセージを送受信するアプリケーションは AMQP 1.0 アプリケーションのみです。

アプリケーションは、トピック・ストリングにサブスクライブすることによって作成される宛先を持続するものとして、アプリケーションとキュー・マネージャーとの接続が一時的に切断された場合でもメッセージが失われないようにするかどうかを選択できます。

アプリケーションはまた、メッセージが宛先から消去される前に保存される期間も選択できます。



関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW IBM MQ アプリケーションとメッセージと交換する AMQP クライアント

AMQP チャネルを定義して開始することにより、AMQP 1.0 アプリケーションは、既存の MQ アプリケーションが受信するメッセージをパブリッシュできます。 AMQP チャネルを介してパブリッシュされるすべてのメッセージは、MQ キューではなく MQ トピックに送信されます。 MQSUB API 呼び出しを使用して既にサブスクリプションを作成した MQ アプリケーションは、その MQ アプリケーションが使用するトピック・ストリングやトピック・オブジェクトと AMQP クライアントによってパブリッシュされるトピック・ストリングとが一致する場合、AMQP 1.0 アプリケーションによってパブリッシュされたメッセージ受信します。

AMQP メッセージ・データ、属性、およびプロパティーは、MQ アプリケーションが受信する MQ メッセージに設定されています。 AMQP から MQ へのメッセージ・マッピングについて詳しくは、 686 ページの『AMQP フィールドの IBM MQ フィールドへのマッピング (着信メッセージ)』 を参照してください。

MQ アプリケーションが永続的なサブスクリプションを作成した場合、AMQP アプリケーションによってパブリッシュされるメッセージは、そのサブスクリプションをサポートするキューに保管されます。 その後、MQ アプリケーションがサブスクリプションを再開すると、アプリケーションはメッセージを受信します。 AMQP アプリケーションがメッセージ存続時間を指定している場合、MQ アプリケーションがその存続時間内に再接続しなければ、メッセージはキューで期限切れになります。

AMQP 1.0 アプリケーションは、既存の MQ アプリケーションによってパブリッシュされたメッセージをコンシュームすることもできます。 MQ アプリケーションによって MQ トピックまたはトピック・ストリングにパブリッシュされたメッセージは、そのアプリケーションがパブリッシュされたトピック・ストリングと一致するトピック・パターンによってサブスクライブされている場合、AMQP 1.0 アプリケーションが受信します。

AMQP 1.0 アプリケーションがサブスクリプションの存続時間値を指定している場合、AMQP アプリケーションがその存続時間よりも長く切断したときには、サブスクリプションはキュー・マネージャーで期限切れとなり、サブスクリプション・キューに保管されているすべてのメッセージは失われます。

MQMD フィールド、メッセージ・プロパティー、およびアプリケーション・データは、AMQP アプリケーションが受信する AMQP メッセージに設定されています。 MQ から AMQP へのメッセージ・マッピングについて詳しくは、 684 ページの『IBM MQ フィールドから AMQP フィールドへのマッピング (出力メッセージ)』 を参照してください。

関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

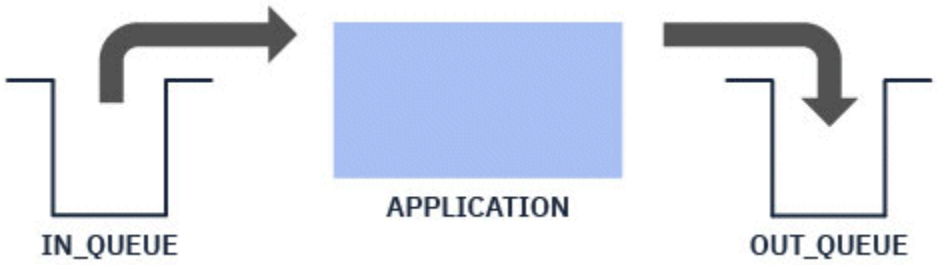
ALW IBM MQ キュー上のアプリケーションと直接対話するための AMQP クライアントの構成

V 9.3.0 IBM MQ AMQP 実装は、パブリッシュ/サブスクライブと Point-to-Point をサポートします。Point-to-Point をサポートしない AMQP クライアントの場合は、以下のステップを使用して、キューにメッセージを送信したり、キューからメッセージを受信したりします。

概要

例えば、入力キュー IN_QUEUE からメッセージを読み取り、それらのメッセージを出力キュー OUT_QUEUE に書き込むアプリケーションがあるとします。 AMQP クライアントは、 IN_QUEUE にメッセージを書き込んだり、 OUT_QUEUE からメッセージを取得したりすることができます。

注 : アプリケーション自体を変更する必要はありません。



AMQP パブリッシャーがメッセージをキューに書き込むには、対象となるキューを宛先として、AMQP クライアントのパブリッシュ先のトピック・ストリングに対して管理サブスクリプションを作成する必要があります。697 ページの『アプリケーションへのメッセージの送信方法:』を参照してください。

AMQP サブスクライバーがキューからメッセージを取得するには、このキューを、同じ名前を持つ別名キューに置き換えて、この別名キューのターゲットを、AMQP クライアントがサブスクライブしているトピック・ストリングを表すトピック・オブジェクトにする必要があります。698 ページの『アプリケーションからのメッセージの取得方法:』を参照してください。

アプリケーションへのメッセージの送信方法:

アプリケーションは既に IN_QUEUE からメッセージをピックアップしており、AMQP クライアントがメッセージをパブリッシュできるようにして、アプリケーションによって処理されるようにこのキューに移動できるようにします。

これを行うには、新しい管理サブスクリプションを作成します。このサブスクリプションのメッセージ受け取り元となるトピック・ストリングは、AMQP クライアントのパブリッシュ先のトピック・ストリングになります。 このサブスクリプションの宛先キューは、アプリケーション IN_QUEUE の入力キューになります。

その管理サブスクリプションの定義済みトピック・ストリングにパブリッシュされるメッセージは、定義済み宛先 (この場合は IN_QUEUE) にルーティングされます。

AMQP クライアントがトピック・ストリング /application/in にパブリッシュすると想定して、以下の MQSC コマンドを使用して管理サブスクリプション APP_IN を作成できます。

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

このオブジェクトを定義すると、 /application/in にパブリッシュされたすべてのメッセージが宛先 IN_QUEUE にルーティングされ、他のアプリケーションによってこのキューに書き込まれた他のメッセージと同じ方法でアプリケーションによって取り出されます。

アプリケーションからのメッセージの取得方法:

アプリケーションはメッセージを OUT_QUEUE に書き込み、そこで他のクライアントがメッセージをピックアップして処理することができます。

ただし今回の事例では、代わりに AMQP クライアントにメッセージを送信する必要があり、しかも AMQP クライアントはパブリッシュ/サブスクライブのみを使用するので、キューから直接メッセージを取得することができません。

これまでメッセージを受信していたクライアントをサブスクライブ側 AMQP クライアントに置き換えるには、AMQP クライアントによるサブスクライブ先のトピック・ストリング用のトピック・オブジェクトと、別名キューを作成する必要があります。

重要 : 別名キューを定義した後、AMQP クライアントがサブスクライブする機会を持つ前に生成側アプリケーションを開始すると、生成側アプリケーションが「キュー」(現在はトピック) に送信するメッセージは、サブスクライバーが存在しないために失われます。

この文書で説明している変更内容は、これまでメッセージを受信していたクライアントをサブスクライブ側 AMQP クライアントのみに置き換えるというものです。 AMQP と他のクライアントの組み合わせを使用してメッセージを取得するには、より大規模な変更が必要になります。

AMQP クライアントがトピック・ストリング /application/out にサブスクライブすると想定すると、以下の MQSC コマンドを使用してトピック・オブジェクト APP_OUT を定義できます。

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

このトピック・オブジェクトに配信されるメッセージは、同じトピック・ストリングにサブスクライブする AMQP クライアントに配信されます。

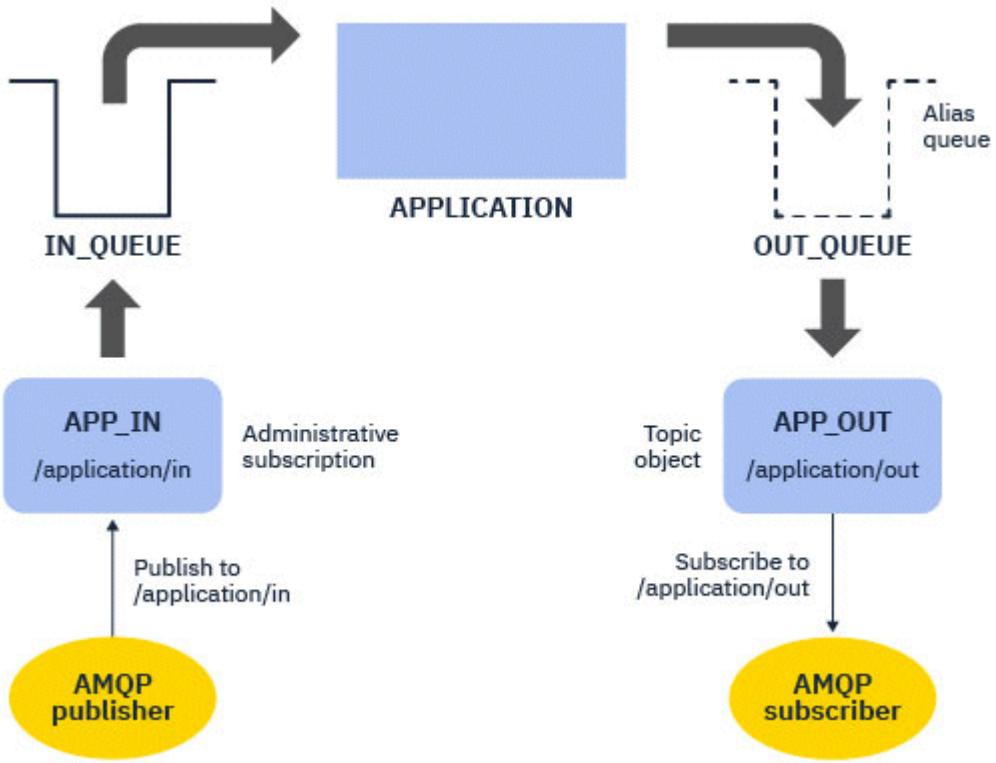
次に、アプリケーションによって OUT_QUEUE に書き込まれたメッセージがこの新しいトピック・オブジェクトに配信され、サブスクライブ・クライアントに送信されるようにする必要があります。

これを行うには、以下の MQSC コマンドを使用して、既存のキュー OUT_QUEUE を同じ名前の別名キューに置き換え、作成したばかりのトピック・オブジェクトのターゲット・タイプにします。

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

これで、アプリケーションによって OUT_QUEUE に書き込まれたメッセージは、キュー上で取り出されるのを待機しません。代わりに、この別名キューのターゲット、つまり新しいトピック・オブジェクト APP_OUT に配信されます。

このトピック・オブジェクト /application/out によって表されるトピック・ストリングにサブスクライブしている AMQP クライアントは、このトピック・オブジェクトに送信されたメッセージを別名キューから受信します。



関連タスク

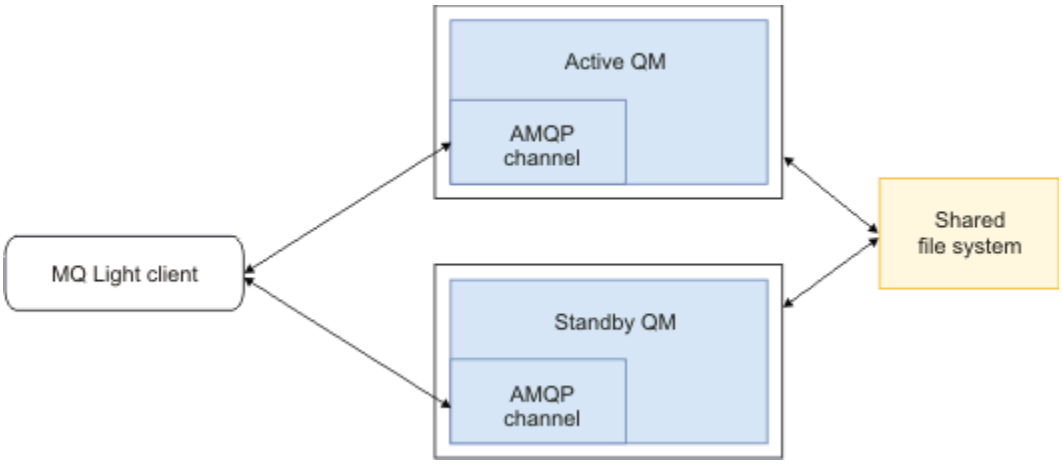
AMQP チャネルの作成および使用

AMQP クライアントの保護

ALW AMQP クライアントの高可用性のための構成

IBM MQ 複数インスタンス・キュー・マネージャーのアクティブ・インスタンスに接続し、高可用性 (HA) ペアの複数インスタンス・キュー・マネージャーのスタンバイ・インスタンスにフェイルオーバーするように AMQP 1.0 アプリケーションを構成できます。 そのためには、AMQP アプリケーションを 2 つの IP アドレスとポートのペアで構成します。

カスタム関数を使用して AMQP クライアント API を構成できます。この関数は、クライアントがサーバーへの接続を失った場合に呼び出されます。 この関数は、スタンバイ IBM MQ キュー・マネージャーなどの代替 IP アドレスに接続することも、元の IP アドレスに接続することもできます。 その他の AMQP クライアントでは、そのクライアントが複数接続エンドポイントの構成をサポートしている場合、 アプリケーションをホストとポートの 2 つのペアによって構成し、AMQP ライブラリーで提供される再接続機能を使用してスタンバイ・キュー・マネージャーに切り替えます。

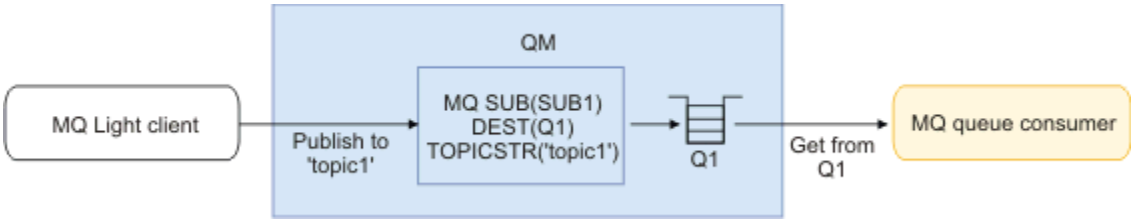


関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW AMQP クライアントのパブリッシュ/サブスクライブの構成

AMQP クライアントは、既存のアプリケーションによって読み取られる IBM MQ キューのメッセージを経路指定する IBM MQ サブスクリプションを持つトピックに対してパブリッシュすることができます。 キューから読み取るように構成されている既存の IBM MQ アプリケーションに AMQP 1.0 アプリケーションがメッセージを送信するようにするには、キュー・マネージャーで管理対象 IBM MQ サブスクリプションを定義する必要があります。

AMQP アプリケーションによって使用されるトピック・ストリングと一致するトピック・パターンを使用するように、サブスクリプションを構成します。 サブスクリプションの宛先を、IBM MQ アプリケーションがメッセージを取得または参照するキューの名前に設定します。

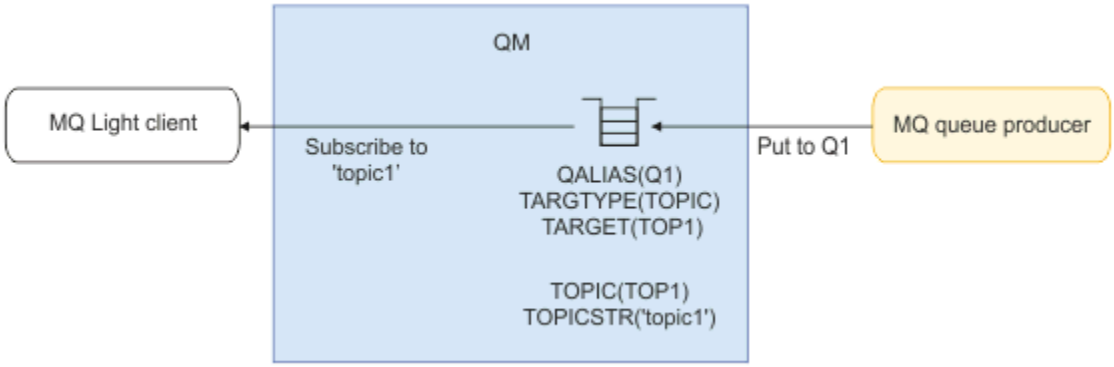


関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW キュー別名を使用して IBM MQ アプリケーションからのメッセージを受信する AMQP クライアント

AMQP クライアントはトピックをサブスクライブし、IBM MQ アプリケーションによって別名キューに put されたメッセージを受信します。 AMQP 1.0 アプリケーションが、キューにメッセージを書き込むように構成されている既存の IBM MQ アプリケーションからメッセージを受信するようにしたい場合は、キュー・マネージャーでキュー別名 (QALIAS) を定義する必要があります。

キュー別名は、IBM MQ アプリケーションが put のために開くキューと同じ名前でなければなりません。キュー別名は、TOPIC の基本タイプ、およびトピック・ストリングが AMQP アプリケーションによってサブスクライブされたトピック・パターンと一致する IBM MQ トピック・オブジェクトの基本オブジェクトを指定する必要があります。



関連タスク
AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW アプリケーション・サーバーとの間で要求を送信し応答をコンシュームする AMQP クライアント

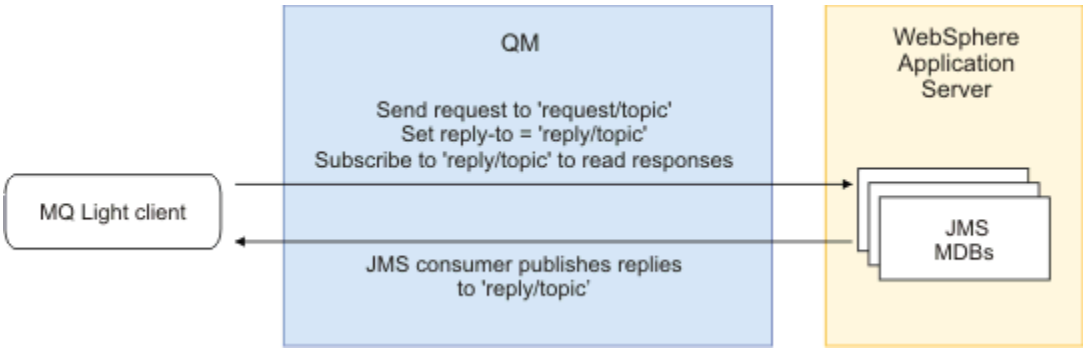
AMQP クライアントは、アプリケーション・サーバーで実行されているメッセージ駆動型 Bean に要求をサブミットし、応答トピックからの応答をコンシュームすることができます。 IBM MQ は、IBM MQ のパブリッシュするメッセージに応答先トピックを設定する AMQP 1.0 アプリケーションをサポートします。 AMQP メッセージに応答先属性を設定してパブリッシュされる場合、応答先フィールドの値は、JMS コンシューマーが受信する JMS プロパティーとして設定されます。 この設定により、JMS コンシューマーはメッセージから応答先トピックを読み取り、応答メッセージを AMQP クライアントに返信することができます。

JMS プロパティーは **JMSReplyTo** です。 AMQP 応答先ストリングは、以下のいずれかのタイプでなければなりません。

- トピック・ストリング。 例えば、`'reply/topic'`
- `amqp://host:port/[topic-string]`の形式の AMQP アドレス URL。 例えば、`amqp://localhost:5672/reply/topic`

reply-to フィールドに AMQP アドレス URL を指定すると、 **JMSReplyTo** プロパティーを設定する前に、URL の末尾にあるトピック・ストリング以外のすべてが削除されます。

AMQP 応答先アドレスから **JMSReplyTo** プロパティーへのマッピングについて詳しくは、 686 ページの『AMQP フィールドの IBM MQ フィールドへのマッピング (着信メッセージ)』 を参照してください。



関連タスク

AMQP チャネルの作成および使用
AMQP クライアントの保護

ALW MQ Light アプリケーションと Apache Qpid JMS アプリケーション間のインターオペラビリティー

MQ Light アプリケーションと Apache Qpid JMS アプリケーションは、同様の方法で動作します。トピックにサブスクライブする場合は、同じ命名規則に従う IBM MQ サブスクリプションを作成します。

専用の非共有サブスクリプション

アプリケーションによって作成される IBM MQ サブスクリプションの名前は、 `:private:<clientid>:<topicstring>`です。

別のクライアント ID を使用するアプリケーションは、他のアプリケーションによって作成されたサブスクリプションにアクセスすることはできません。これは、サブスクリプション名が自動的に生成され、AMQP クライアント ID が含まれているためです。

Apache Qpid JMS と MQ Light の両方のアプリケーションで、専用サブスクリプションにこの命名規則が使用されます。

グローバル共有サブスクリプション

AMQP クライアントによって作成されたグローバル共有 IBM MQ サブスクリプションの名前は、 `:share:<sharename>:<topicstring>`です。

異なる AMQP クライアント ID を持つ複数のアプリケーションが同じ共有名およびトピック・ストリングを指定している場合、それらのアプリケーションは 1 つのサブスクリプションを共有し、連携してそのサブスクリプションのメッセージを処理することができます。 サブスクリプションからメッセージをドレーンするワーカー・アプリケーションの数を拡張する場合は、このパターンを使用できます。

Apache Qpid JMS と MQ Light の両方のアプリケーションで、グローバル共有サブスクリプションにこの命名規則が使用されます。 Apache Qpid JMS の場合は、JMS 接続にクライアント ID を指定しないようにします。

Apache Qpid JMS ライブラリーは AMQP クライアント ID を自動的に生成しますが、このクライアント ID は IBM MQ サブスクリプションの命名の目的で使用されることはありません。

注 : グローバル共有サブスクリプションは、引き続き個々のキュー・マネージャーにスコープ設定されます。

専用共有サブスクリプション

AMQP クライアントによって作成された専用共有 IBM MQ サブスクリプションの名前は、 `:privateshare:<clientid>:<sharename>:<topicstring>`です。

単一の Apache Qpid JMS アプリケーションからの複数のスレッドが同じ共有名とトピック・ストリングを使用し、JMS 接続上でクライアント ID が構成されている場合、それらのスレッドは同じ IBM MQ サブスクリプション・オブジェクトを共有します。

ただし、他の Apache Qpid JMS 接続では、異なるクライアント ID を使用する必要があるため、サブスクリプションを共有することはできません。

MQ Light クライアントは、専用共有サブスクリプションの概念をサポートしていないため、Apache Qpid JMS アプリケーションによって作成される専用共有サブスクリプションからのメッセージをコンシュームすることはできません。

IBM MQ JMS サブスクリプション

IBM MQ JMS サブスクリプションは、AMQP チャネルとは異なる命名方式を使用します。 MQ Light アプリケーションおよび Apache Qpid JMS アプリケーションは、IBM MQ JMS アプリケーションとサブスクリプションを共有することはできません。

関連概念

AMQP クライアント・アプリケーションの開発

AMQP API の IBM MQ サポートにより、IBM MQ 管理者は AMQP チャネルを作成できます。 このチャネルを開始すると、AMQP クライアント・アプリケーションからの接続を受け入れるポート番号が定義されます。

ALW IBM MQ AMQP リスナー制御プロパティー

マルチスレッド・アプリケーションのパフォーマンスを向上させるために、AMQP プロパティー・ファイルでプロパティーを構成することにより、AMQP サービスが使用するワーカー・スレッドの数を調整できます。

AMQP リスナー・サービスのプロパティーは、以下のプロパティー・ファイルで構成できます。

- Windows Windows システムの場合: `amqp_win.properties` 。
- Linux AIX AIX and Linux システムの場合: `amqp_unix.properties` 。

構成できるプロパティーは以下のとおりです。

表 105. AMQP リスナー・サービスのプロパティー	
Property	**説明**
com.ibm.mq.MQXR.Workers	AMQP リスナー・サービスが作成するサーバー・ワーカー・スレッドの数。 この値を指定しない場合は、デフォルトでシステム上の論理プロセッサーの数と等しくなります。
MQIBindType	AMQP サービスのバインディング・タイプ。 FASTPATH、 SHARED、または ISOLATED のいずれかです。 デフォルトは FASTPATH です。

AMQP リスナー・サービスは、多数のワーカー・スレッド間でクライアント接続ワークロードのバランスを取ります。 AMQP サービスが使用するワーカー・スレッドの数は、 **com.ibm.mq.MQXR.Workers** プロパティーを使用して指定できます。

IBM MQ キュー・マネージャー管理者は、マルチスレッド・アプリケーションのパフォーマンスを向上させるために、ワーカー・スレッドの数を調整できます。 通常、最良のパフォーマンスは、ワーカー・スレッドの数がシステム上の論理プロセッサーの数と一致する場合に達成されます。 ただし、特定のマシン構成およびクライアント負荷特性では、これが常に該当するとは限りません。したがって、ワーカー・スレッド数の最適な値を見つけるには、チューニングの要素が必要になる場合があります。

チューニングを行う前に、クライアント・アプリケーションとそのワークロードの性質を十分に理解していることを確認してください。 さまざまなスレッド・カウントおよびベンチマーキングを使用してアプリケーションのパフォーマンスを測定することは、ワーカー・スレッドの数の最適な値を決定するのに役立ちます。

注 : MQ Appliance これらのプロパティーは、 IBM MQ Appliance では適用されません。 IBM MQ Appliance ではデフォルト値が使用されます。

IBM MQ での REST アプリケーションの開発

メッセージを送受信する REST アプリケーションを開発できます。 IBM MQ は、プラットフォームや能力に応じて、さまざまな REST API をサポートします。

以下のオプションは、IBM MQ との間でメッセージを送受信する際に選択できるオプションとして、IBM MQ でサポートされています。

- IBM MQ messaging REST API
- IBM z/OS Connect EE
- IBM Integration Bus
- DataPower

IBM MQ messaging REST API

messaging REST API を使用して IBM MQ メッセージを平文形式で送受信および参照できます。 messaging REST API はデフォルトでは有効です。

一般的なメッセージ・プロパティーを設定するために使用できる多くのさまざまな HTTP ヘッダーがサポートされています。

messaging REST API は完全に IBM MQ セキュリティーと一体化しています。 messaging REST API を使用するためには、ユーザーは mqweb サーバーで認証される必要があり、MQWebUser 役割のメンバーでなければなりません。

詳しくは、705 ページの『REST API を使用したメッセージング』を参照してください。 IBM Developer の Tutorial: Get started with the IBM MQ messaging REST API も参照してください。ここには、メッセージング REST API を使用するための Go および Node.js のサンプルがあります。

IBM z/OS 接続 EE

IBM z/OS Connect EE は、CICS または IMS トランザクション、IBM MQ キューおよびトピックなどの既存の z/OS 資産の上に REST API を作成できるようにする z/OS 製品です。 既存の z/OS 資産はユーザーから隠されます。 そのため、資産も資産を使用する既存のアプリケーションも変更することなく、資産を REST 対応にすることができます。

IBM z/OS Connect EE は、REST API によって使用される JSON データと、多くのメインフレーム・アプリケーションで予想される他の従来の言語構造 (例えば、COBOL) との間で変換するための自動データ変換を提供します。

Eclipse ベースの IBM z/OS Connect EE API ツールキットを使用して、照会パラメーターと URL パス・セグメントを使用し、IBM z/OS Connect EE ランタイムで送受信される JSON 形式を操作する包括的な RESTful API を構築できます。

IBM z/OS Connect EE は、IBM MQ サービス・プロバイダーを介して RESTful API として IBM MQ キューおよびトピックを公開するために使用することができます。 次の 2 つの異なるサービス・タイプがサポートされます。

- 単方向サービス: 1 つの IBM MQ 操作をキューまたはトピックに対して実行できる REST API を提供します。 厳密な構成によっては、HTTP 要求の結果としてメッセージがキューに送信される場合やトピックにパブリッシュされる場合があります。また、HTTP 要求によってメッセージがキューから破壊的に受信される場合もあります。または、HTTP 要求によって、メッセージがキューから破壊的に受信される場合があります。
- 双方向サービス: 要求応答スタイルのバックエンド・アプリケーションで使用するキューのペアに対して REST API を利用できます。 呼び出し元は HTTP 要求を双方向サービスに発行します。 HTTP 要求ペイロードは、JSON から従来の言語構造に変換され、要求キューに書き込まれ、そこでバックエンド・アプリケーションによって処理されます。そして、応答が応答キューに書き込まれます。 この応答は、サービスによって取り出され、従来の言語構造から JSON に変換され、POST 応答の本文として呼び出し元に送り返されます。

IBM z/OS Connect EE の詳細については、「z/OS Connect EE」を参照してください。

IBM MQ サービス・プロバイダーの詳細については、「IBM MQ サービス・プロバイダーの使用」を参照してください。

IBM Integration Bus

IBM Integration Bus は、 IBM の最先端の統合テクノロジーであり、サポートするメッセージ・フォーマットやプロトコルに関係なく、アプリケーションとシステムを相互に接続するために使用できます。

IBM Integration Bus は常に IBM MQ をサポートしており、 *HTTPInput* ノードおよび *HTTPRequest* ノードを提供します。これらのノードを使用して、 IBM MQ およびデータベースなどの他の多くのシステム上で RESTful インターフェースを構成できます。

IBM Integration Bus を使用すれば、IBM MQ の上にシンプルな REST インターフェースを提供できるだけでなく、さまざまなことが可能になります。 この機能を使用すると、ペイロードの高度な操作、エンリッチメントなど、他の多くの拡張機能を REST API の一部として利用できます。

詳しくは、テクノロジー・サンプル を参照してください。これは、XML ペイロードを予期する IBM MQ アプリケーションの上に、REST インターフェースを介して JSON を公開します。

DataPower

DataPower ゲートウェイはセキュリティー、制御、統合、および最適化したアクセスを、 IBM MQ を含む幅広いシステムに提供するのを助ける単一のマルチチャネル・ゲートウェイです。 ハードウェア形式と仮想形式の両方の要素として提供されています。

DataPower が提供するサービスの 1 つは、あるプロトコルの入力を取り、別のプロトコルで出力を生成するマルチプロトコル・ゲートウェイです。 特に DataPower は、HTTP(S) データを受け入れ、クライアント接続を介して IBM MQ に経路指定するよう構成するできます。これを使用して、IBM MQ 上に REST インターフェースを構築することができます。 変換などの他の DataPower サービスを使用して、REST インターフェースを機能拡張することも可能です。

詳しくは、マルチプロトコル・ゲートウェイを参照してください。

REST API を使用したメッセージング

messaging REST API を使用して、単純な Point-to-Point メッセージングおよびパブリッシュ・メッセージングを実行できます。 トピックへのメッセージのパブリッシュ、キューへのメッセージの送信、キュー上のメッセージの参照、およびキューからのメッセージの破壊的な取得を行うことができます。 messaging REST API で送受信される情報は平文形式です。

始める前に

注 :

- messaging REST API はデフォルトでは有効です。 messaging REST API を無効にして、すべてのメッセージングを禁止することができます。 messaging REST API を有効または無効にする方法について詳しくは、messaging REST API の構成を参照してください。
- messaging REST API は IBM MQ セキュリティーと一体化しています。 messaging REST API を使用するためには、ユーザーは mqweb サーバーで認証される必要があり、MQWebUser 役割のメンバーでなければなりません。 ユーザーは、指定されたキューまたはトピックへのアクセスも許可されている必要があります。 REST API のセキュリティーについて詳しくは、 IBM MQ Console および REST API のセキュリティーを参照してください。
- messaging REST API とともに Advanced Message Security (AMS) を使用する場合は、メッセージを通知するユーザーのコンテキストではなく、すべてのメッセージが mqweb サーバーのコンテキストを使用して暗号化されることに注意してください。
- メッセージを受信または参照する場合、IBM MQ MQSTR または JMS TextMessage 形式のメッセージのみがサポートされます。 その後、同期点下ですべてのメッセージが破壊的に受信され、未処理のメッセージはキューに残されます。 それらの有害メッセージを別の宛先に移動するように、IBM MQ キューを構成できます。 詳しくは、236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。
- messaging REST API には、トランザクション・サポートを使った「1 回限りメッセージ送達」の機能はありません。 HTTP POST が発行され、クライアントが HTTP 応答を受信する前に接続が失敗した場合、クライアントは、メッセージが指定のキューに送信されたのか、指定のトピックにパブリッシュされたのかをすぐには判別できません。 HTTP DELETE が発行されて、クライアントが HTTP 応答を受け取る前に接続が失敗した場合、メッセージはキューから破壊的に取得されて失われている可能性があります。破壊取得をロールバックする手段はないからです。
- V 9.3.0 IBM MQ 9.3.0 以降、着信ストリング内の改行は HTTP POST 操作によって削除されなくなりました。 旧バージョンを使用する REST アプリケーションでは、REST API を使用して送信または公開されるメッセージで改行を使用しないでください。改行は失われます。

手順

- 706 ページの『messaging REST API の使用開始』
- 708 ページの『messaging REST API の使用』
- REST API エラー処理
- REST API ディスカバリー
- REST API 各国語サポート

関連資料

メッセージング REST API に関する参照情報

関連情報

チュートリアル: IBM MQ メッセージング REST API の使用を開始する

messaging REST API の使用開始

messaging REST API の使用をすぐに開始して、cURL を使用していくつかのコマンド例を試してみてください。

始める前に

messaging REST API を使い始めるにあたって、このタスクに含まれる例には以下の要件があります。

- 例では、cURL を使用して REST 要求を送信することで、キューとの間でメッセージの書き込みと取得が行われます。 したがって、このタスクを実行するためには、ご使用のシステムに cURL がインストールされている必要があります。
- この例では、キュー・マネージャー QM1 を使用します。 同じ名前のキュー・マネージャーを作成するか、ご使用のシステム上の既存のキュー・マネージャーに置き換えてください。 キュー・マネージャーは mqweb サーバーと同じマシン上になければなりません。
- このタスクを実行するには、**dspmqweb** コマンドを使用するための特定の特権を持っているユーザーである必要があります。
  - z/OS z/OS の場合、**dspmqweb** コマンドを実行する権限と、mqwebuser.xml ファイルに対する書き込みアクセス権限を持っている必要があります。
  - Multi 他のすべてのオペレーティング・システムでは、特権ユーザーでなければなりません。

IBM i IBM i では、コマンドを QSHELL で実行する必要があります。

手順

1. mqweb サーバーに messaging REST API が構成されていることを確認します。
   - administrative REST API、 administrative REST API for MFT、 messaging REST API、または IBM MQ Console で使用するために mqweb サーバーが構成されていることを確認します。基本レジストリーを使用した mqweb サーバーの構成について詳しくは、 mqweb サーバーの基本構成を参照してください。
   - mqweb サーバーが既に構成されている場合は、 mqweb サーバーの基本構成のステップ 5 で、メッセージングを有効にするための適切なユーザーを追加したことを確認します。
     - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が true に設定されている場合、 messaging REST API のユーザーは MQWebUser 役割のメンバーでなければなりません。 MQWebAdmin 役割および MQWebAdminRO 役割は、messaging REST API には適用されません。 ユーザーは、OAM または RACF®を介してメッセージングに使用されるキューおよびトピックにアクセスする権限も持っている必要があります。
     - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が false に設定されている場合は、mqweb サーバーの開始に使用するユーザー ID に、OAM または RACF を介したメッセージングに使用されるキューへのアクセスを許可する必要があります。
2. z/OS
   z/OS では、 **dspmqweb** コマンドを使用できるように WLP_USER_DIR 環境変数を設定します。 次のコマンドを入力して、mqweb サーバー構成を指すように変数を設定します。

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   ここで、 *WLP_user_directory* は、 crtmqweb に渡されるディレクトリーの名前です。 例:

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

   詳しくは、mqweb サーバーの作成を参照してください。
3. 次のコマンドを入力して、 REST API URL を確認します。

   ```
   dspmqweb status
   ```

以下のステップの例では、REST API URL がデフォルトの URL `https://localhost:9443/ibmmq/rest/v2/` であることを前提としています。 デフォルト以外の URL を使用している場合は、以下の手順の URL を置き換えてください。

4. キュー・マネージャー QM1 にキュー MSGQ を作成します。 このキューがメッセージングに使用されます。 以下のいずれかの方法を使用します。

   - administrative REST API の mqsc リソースで POST 要求を使用し、 mqadmin ユーザーとして認証します。

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
     data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
     \"qlocal\",\"name\": \"MSGQ\"}"
     ```

   - MQSC コマンドを使用します。

     z/OS z/OS では、**runmqsc** コマンドではなく 2CR ソースを使用します。 詳しくは、IBM MQ for z/OS で MQSC コマンドおよび PCF コマンドを発行できるソースを参照してください。

     a. 次のコマンドを入力して、キュー・マネージャーに対して **runmqsc** を開始します。

        ```
        runmqsc QM1
        ```

     b. **DEFINE QLOCAL** MQSC コマンドを使用して、キューを作成します。

        ```
        DEFINE QLOCAL(MSGQ)
        ```

     c. 次のコマンドを入力して、**runmqsc** を終了します。

        ```
        end
        ```

5. mqweb サーバーの基本的な構成 のステップ 5 で mqwebuser.xml に追加したユーザーに、キュー MSGQ にアクセスする権限を付与します。 myuser が使用されているユーザーに置き換えます。

   - z/OS On z/OS:

     a. キューに対するアクセス権限をユーザーに付与します。

        ```
        RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
        PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
        ```

     b. キュー上のすべてのコンテキストを設定できるアクセス権限を mqweb 開始タスク・ユーザー ID に付与します。

        ```
        RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
        PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
        ```

   - Multi これ以外のすべてのオペレーティング・システムでは、ユーザーが mqm グループに含まれていれば、権限は既に付与されています。 そうでない場合は、以下のコマンドを入力します。

     a. 次のコマンドを入力して、キュー・マネージャーに対して **runmqsc** を開始します。

        ```
        runmqsc QM1
        ```

     b. **SET AUTHREC** MQSC コマンドを使用して、キューに対する表示、照会、取得、および書き込みの権限をユーザーに付与します。

        ```
        SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
        PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
        ```

     c. 次のコマンドを入力して、**runmqsc** を終了します。

        ```
        end
        ```

6. message リソースに対する POST 要求を使用して、 Hello World! という内容のメッセージをキュー・マネージャー QM1 のキュー MSGQ に書き込みます。 myuser および mypassword は、mqwebuser.xml のユーザー ID とパスワードに置き換えてください。

   基本認証が使用され、任意の値を持つ ibm-mq-rest-csrf-token HTTP ヘッダーが cURL REST 要求に設定されます。 この追加ヘッダーは、POST、PATCH、および DELETE の各要求に必要です。

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X POST
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
   plain;charset=utf-8" --data "Hello World!"
   ```

7. message リソースに対する DELETE 要求を使用して、キュー・マネージャー QM1 上のキュー MSGQ のキュー Hello World! からメッセージを破壊的に取得します。 myuser および mypassword は、mqwebuser.xml のユーザー ID とパスワードに置き換えてください。

   ```
   curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
   -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
   ```

   メッセージ Hello World! が返されます。

次のタスク

- この例では、基本認証を使用して要求を保護します。 代わりに、トークン・ベース認証またはクライアント・ベース認証を使用することもできます。 詳しくは、REST API でのクライアント証明書認証の使用、IBM MQ Console、および REST API でのトークン・ベースの認証の使用を参照してください。
- messaging REST API の使用法と照会パラメーターで URL を構成する方法について詳しくは、708 ページの『messaging REST API の使用』を参照してください。
- messaging REST API を使用するときは、パフォーマンスを最適化するためにキュー・マネージャーへの接続がプールされます。 最大プール・サイズ、およびプール内のすべての接続が使用中のときに実行されるアクションを構成できます。messaging REST API の構成を参照してください。
- 使用可能な messaging REST API リソースと使用可能なすべての照会パラメーターについては、messaging REST API に関する参照情報を参照してください。
- administrative REST API (IBM MQ 管理用の RESTful インターフェース) の検出: REST API を使用する管理。
- ブラウザー・ベースの GUI である IBM MQ Console については、IBM MQ Console を使用した管理を参照してください。

messaging REST API の使用

messaging REST API を使用する場合は、URL に対して HTTP メソッドを呼び出して IBM MQ メッセージを送受信します。 HTTP メソッド (POST など) は、URL で表されるオブジェクトに対して実行する操作のタイプを表します。 その操作に関する詳細は、照会パラメーター内にエンコードします。 操作の実行結果に関する情報は、一般には HTTP 応答の本体として返されます。

始める前に

messaging REST API を使用する前に、以下の点を考慮してください。

- messaging REST API を使用するには、mqweb サーバーで認証を行う必要があります。 HTTP 基本認証、クライアント証明書認証、またはトークン・ベースの認証を使用して、認証を行うことができます。 これらの認証方式の使用方法について詳しくは、IBM MQ Console および REST API セキュリティーを参照してください。
- REST API は大文字小文字を区別します。 例えば、キュー・マネージャーの名前が qmgr1 である場合、以下の URL の HTTP POST はエラーになります。

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- V 9.3.3 messaging REST API を使用してリモート・キュー・マネージャーに接続する場合は、キュー・マネージャー名ではなく、キュー・マネージャー接続の固有の名前を使用する必要があります。

- IBM MQ オブジェクト名で使用できる文字の一部は、URL 内に直接エンコードすることができません。そのような文字を正しくエンコードするためには、適切な URL エンコード方式を使用する必要があります。
  - スラッシュは、%2F としてエンコードする必要があります。
  - パーセント記号は %25 としてエンコードする必要があります。
  - ピリオドは、 %2E としてエンコードする必要があります。
  - 疑問符は %3F としてエンコードする必要があります。
- メッセージを受信または参照する場合、IBM MQ MQSTR および JMS TextMessage 形式のメッセージのみがサポートされます。 その後、同期点下ですべてのメッセージが破壊的に受信され、未処理のメッセージはキューに残されます。 それらの有害メッセージを別の宛先に移動するように、IBM MQ キューを構成できます。 詳しくは、236 ページの『IBM MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

このタスクについて

REST API を使用して IBM MQ キュー・オブジェクトに対してメッセージング・アクションを実行する場合は、まず、そのオブジェクトを表す URL を構成する必要があります。 どの URL も、要求を送信するホスト名とポートを示す接頭部で始まります。 URL の残りの部分は、特定のオブジェクト、またはそのオブジェクト (リソース) への経路を表します。

リソースに対して実行するメッセージング・アクションによって、URL に照会パラメーターが必要かどうかが決まります。 また、使用する HTTP メソッドや、追加情報を URL に送信したり URL から戻したりするかどうかも決まります。 追加情報を HTTP 要求に含める場合もあれば、HTTP 応答の一部として追加情報が返される場合もあります。

URL を構成したら、 HTTP 要求を IBM MQ に送信できます。 選択したプログラミング言語に組み込んだ HTTP 実装を使用して、要求を送信できます。 cURL などのコマンド・ライン・ツール、Web ブラウザーや Web ブラウザー・アドオンを使用して、要求を送信することもできます。

重要 : 少なくとも、手順 709 ページの『1.a』 と 709 ページの『1.b』 を実行する必要があります。

手順

1. URL を構成します。
   a) 以下のコマンドを入力して、接頭部 URL を特定します。

   ```
   dspmqweb status
   ```

   使用する URL には、 `/ibmmq/rest/` 句が含まれます。

   b) メッセージングに使用するキューおよび関連付けられたキュー・マネージャー・リソースを URL パスに追加します。

   メッセージング・リファレンスでは、括弧 *{}* で囲まれた URL 内で変数を識別することができます。詳しくは、/messaging/qmgr/{qmgrName}/queue/{queueName}/message を参照してください。

   例えば、キュー・マネージャー *QM1* に関連付けられたキュー *Q1* と対話するには、接頭部 URL に /qmgr および /queue を追加して、以下の URL を作成します。

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
   ```

   ヒント : V 9.3.3 キュー・マネージャーがリモート・キュー・マネージャーの場合は、キュー・マネージャー名の代わりにキュー・マネージャーの固有の名前を使用する必要があります。messaging REST API でリモート・キュー・マネージャーを使用するには、その前にリモート・キュー・マネージャーを構成する必要があります。 詳しくは、710 ページの『messaging REST API で使用するリモート・キュー・マネージャーのセットアップ』を参照してください。

   c) オプション: オプションの照会パラメーターを URL に追加します。

   疑問符 (?) を追加します。 URL に対する照会パラメーター、等号 =、および値。

例えば、次のメッセージが使用可能になるまで最大 30 秒間待機するには、次の URL を作成します。

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) オプション: オプションの照会パラメーターをさらに URL に追加します。

アンパーサンド & を URL に追加してから、 ステップ 1c を繰り返します。

2. URL に対して適切な HTTP メソッドを起動します。 オプションのメッセージ・ペイロードを指定し、認証のために適切なセキュリティー資格認定を提供します。 以下に例を示します。
   - 選択したプログラミング言語の HTTP/REST 実装を使用します。
   - REST クライアント・ブラウザー・アドオンや cURL などのツールを使用します。

V 9.3.3 messaging REST API で使用するリモート・キュー・マネージャーのセットアップ

messaging REST API を使用して、メッセージングのためにリモート・キュー・マネージャーに接続できます。 リモート・キュー・マネージャーに接続する前に、リモート・キュー・マネージャー構成をセットアップする必要があります。 その後、構成情報に定義されている固有の名前を使用して、リモート・キュー・マネージャーに接続できます。

始める前に

- administrative REST API、 administrative REST API for MFT、 messaging REST API、または IBM MQ Console で使用するために mqweb サーバーが構成されていることを確認します。基本レジストリーを使用した mqweb サーバーの構成について詳しくは、 mqweb サーバーの基本構成を参照してください。
- mqweb サーバーが既に構成されている場合は、 mqweb サーバーの基本構成のステップ 5 で、メッセージングを使用可能にするための適切なユーザーを追加したことを確認してください。 messaging REST API のユーザーは、 `MQWebUser` 役割のメンバーでなければなりません。 `MQWebAdmin` 役割および `MQWebAdminRO` 役割は、messaging REST API には適用されません。
  - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が `true` に設定されている場合、 `MQWebUser` 役割のユーザーは、OAM または RACF を介したメッセージングに使用されるキューおよびトピックへのアクセスを許可されている必要があります。
  - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が `false` に設定されている場合、mqweb サーバーの開始に使用されるユーザー ID に、OAM または RACF を介したメッセージングに使用されるキューおよびトピックへのアクセスを許可する必要があります。
- messaging REST API がリモート・キュー・マネージャーに接続するように構成されていることを確認します。 詳しくは、 messaging REST API の接続モードの構成を参照してください。

このタスクについて

messaging REST API を使用して、リモート・キュー・マネージャーに接続できます。 リモート・キュー・マネージャーは、別のシステム上のキュー・マネージャー、別のインストール済み環境のキュー・マネージャー、または mqweb サーバーと同じインストール済み環境のキュー・マネージャーにすることができます。

リモート・キュー・マネージャーに接続するには、以下の構成ステップを実行する必要があります。

- サーバー接続チャネルとリスナーを構成します。
- キュー・マネージャーにアクセスするための権限を適切なユーザーに付与します。
- キュー・マネージャーの接続情報を含む CCDT ファイルを作成します。
- **setmqweb remote** コマンドを使用して、接続情報を messaging REST API に追加します。

その後、キュー・マネージャー名の代わりにリソースの URL に固有の名前を指定することで、リモート・キュー・マネージャーを使用できます。

リモート・キュー・マネージャーをキュー・マネージャー・グループの一部として構成することもできます。 詳細については、713 ページの『メッセージング REST API で使用するキュー・マネージャー・グループのセットアップ』を参照してください。

手順

1. リモート・キュー・マネージャーで、キュー・マネージャーへのリモート接続を許可するサーバー接続チャネルを作成します。 コマンド行で **DEFINE CHANNEL** MQSC コマンドを使用して、サーバー接続チャネルを作成できます。
   例えば、キュー・マネージャー QM1 のサーバー接続チャネル QM1.SVRCONN を作成するには、次のコマンドを入力します。

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   DEFINE CHANNEL および使用可能なオプションについて詳しくは、DEFINE CHANNEL を参照してください。
2. 適切なユーザーがキュー・マネージャーへのアクセスを許可されていることを確認してください。 このユーザーには、メッセージングに使用するすべてのキューまたはトピックにアクセスする権限も必要です。 ユーザーには、キュー・マネージャーに対する connect、 inquire、 alternate user、および set context 権限が必要です。 UNIX, Linux, and Windows では、コマンド行で **setmqaut** 制御コマンドを使用します。 z/OS では、 RACF プロファイルを定義して、許可ユーザーにキュー・マネージャーへのアクセス権限を付与します。
   例えば、 UNIX, Linux, and Windows で、ユーザー exampleUser にキュー・マネージャー QM1 へのアクセスを許可するには、次のコマンドを入力します。

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   どのユーザーを許可する必要があるかについて詳しくは、716 ページの『messaging REST API が使用するセキュリティー・プリンシパルの決定』を参照してください。
3. ALW
   リモート・キュー・マネージャーにリスナーが存在しない場合は、コマンド行で **DEFINE LISTENER** MQSC コマンドを使用して、着信ネットワーク接続を受け入れるリスナーを作成します。
   例えば、リモート・キュー・マネージャー QM1 のポート 1414 にリスナー REMOTE.LISTENER を作成するには、次のコマンドを入力します。

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. コマンド行で **START LISTENER** MQSC コマンドを使用して、リスナーが実行されていることを確認します。

   ALW 例えば、 AIX, Linux, and Windows でキュー・マネージャー QM1 のリスナー REMOTE.LISTENER を開始するには、次のコマンドを入力します。

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS 例えば、z/OS でリスナーを開始するには、次のコマンドを入力します。

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   z/OS でリスナーを開始する前に、チャネル・イニシエーターのアドレス・スペースを開始する必要があります。
5. messaging REST API をホストする mqweb サーバーが実行されているシステムで、キュー・マネージャー接続情報を含む JSON CCDT ファイルを作成または更新します。

CCDT ファイルには、name、clientConnection、および type の情報を含める必要があります。 オプションで、transmissionSecurity 情報などの追加情報を含めることができます。 すべての CCDT チャネル属性定義について詳しくは、CCDT チャネル属性定義の完全なリストを参照してください。

以下の例は、リモート・キュー・マネージャー接続のための基本的な JSON CCDT ファイルを示しています。 チャネルの名前を、ステップ 711 ページの『1』で作成したサーバー接続チャネルの例と同じ名前に設定します。 接続ポートは、リスナーが使用するポートと同じ値に設定されます。 接続ホストは、リモート・キュー・マネージャー QM1 が実行されているシステムのホスト名に設定されます。

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

6. messaging REST API をホストする mqweb サーバーを実行しているインストール済み環境から、**setmqweb remote** コマンドを使用して、リモート・キュー・マネージャー情報を mqweb サーバー構成に追加します。

   少なくとも、以下のパラメーターを指定する必要があります。

   - **-qmgrName**。キュー・マネージャーの名前を指定します。
   - **-ccdtURL**。キュー・マネージャーの CCDT URL を指定します。
   - **-uniqueName**。ここで、キュー・マネージャーの固有の名前を指定します。 固有の名前は、同じ名前を持つ可能性があるため、別のキュー・マネージャーを識別するために存在してはならないリモート・キュー・マネージャーを区別するために使用されます。

   リモート・キュー・マネージャー接続に使用するユーザー名とパスワード、トラストストアと鍵ストアの詳細など、その他のいくつかのオプションを指定できます。 **setmqweb remote** コマンドで指定できるパラメーターの完全なリストについては、 setmqweb remote を参照してください。

   例えば、サンプルの CCDT ファイルを使用してリモート・キュー・マネージャー QM1 を追加するには、次のコマンドを入力します。

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

タスクの結果

messaging REST API でリモート・キュー・マネージャーを使用するには、キュー・マネージャー名の代わりにリソースの URL で固有の名前を使用します。

例

以下の例では、キュー・マネージャー QM1 のリモート・キュー・マネージャー接続をセットアップします。 ユーザー exampleUser に付与された許可に基づいてキュー・マネージャーを管理する権限を持つ IBM MQ Console 。 **setmqweb remote** を使用してキュー・マネージャー接続を構成するときに、このユーザーの資格情報が IBM MQ Console に提供されます。

1. リモート・キュー・マネージャー QM1 が存在するシステムで、サーバー接続チャネルとリスナーが作成されます。 リスナーが開始され、ユーザー exampleUser がキュー・マネージャーに接続し、メッセージングに使用されるキューにアクセスするための許可が付与されます。

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
```

```
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. mqweb サーバーが実行されているシステムで、以下の接続情報を含む QM1_ccdt.json ファイルが作成されます。

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. mqweb サーバーが実行されているシステムで、キュー・マネージャー QM1 の接続情報が mqweb サーバーに追加されます。 exampleUser の資格情報は、接続情報に含まれています。

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. messaging REST API は、リソース URL 内のキュー・マネージャー名の代わりにキュー・マネージャー接続の固有の名前を使用して、リモート・キュー・マネージャー QM1 に接続できます。

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.3.3 メッセージング REST API で使用するキュー・マネージャー・グループのセットアップ

messaging REST API を使用して、メッセージング用のキュー・マネージャー・グループに接続できます。キュー・マネージャー・グループに接続するには、その前にグループのリモート・キュー・マネージャー構成をセットアップする必要があります。 その後、構成情報に定義されている固有の名前を使用して、キュー・マネージャー・グループに接続できます。

始める前に

- administrative REST API、 administrative REST API for MFT、 messaging REST API、または IBM MQ Console で使用するために mqweb サーバーが構成されていることを確認します。基本レジストリーを使用した mqweb サーバーの構成について詳しくは、 mqweb サーバーの基本構成を参照してください。
- mqweb サーバーが既に構成されている場合は、 mqweb サーバーの基本構成のステップ 5 で、メッセージングを使用可能にするための適切なユーザーを追加したことを確認してください。 messaging REST API のユーザーは、 MQWebUser 役割のメンバーでなければなりません。 MQWebAdmin 役割および MQWebAdminRO 役割は、messaging REST API には適用されません。
  - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が true に設定されている場合、MQWebUser 役割のユーザーには、メッセージングに使用されるキューおよびトピックにアクセスする権限が必要です。 OAM または RACF を使用して、これらのユーザーに権限を与えることができます。
  - mqweb サーバー構成で **mqRestMessagingAdoptWebUserContext** が false に設定されている場合、mqweb サーバーを開始するユーザー ID には、メッセージングに使用されるキューおよびトピックにアクセスする権限が必要です。 OAM または RACF を使用して、このユーザーを許可することができます。

- messaging REST API がリモート・キュー・マネージャーに接続するように構成されていることを確認します。 詳しくは、 messaging REST API の接続モードの構成 を参照してください。

このタスクについて

キュー・マネージャー・グループを使用すると、グループ内の任意のキュー・マネージャーにアプリケーションを接続できます。 グループは、クライアント・チャネル定義テーブル (CCDT) 内の接続のセットとして定義されます。 MQCONN 呼び出しまたは MQCONNX 呼び出しを使用する場合は、キュー・マネージャー名の前にアスタリスクを付けて、グループを参照します。 messaging REST API では、キュー・マネージャー・グループに関連付けられた固有の名前を使用してグループを参照します。 キュー・マネージャー名の代わりに、固有の名前がリソースの URL に含まれます。 キュー・マネージャー・グループについて詳しくは、 928 ページの『CCDT のキュー・マネージャー・グループ』を参照してください。

リモート・キュー・マネージャーを個別に構成することもできます。 詳細については、710 ページの『messaging REST API で使用するリモート・キュー・マネージャーのセットアップ』を参照してください。

手順

1. グループ内の各リモート・キュー・マネージャーで、キュー・マネージャーへのリモート接続を許可するサーバー接続チャネルを作成します。 コマンド行で **DEFINE CHANNEL** MQSC コマンドを使用して、サーバー接続チャネルを作成できます。
   例えば、キュー・マネージャー QM1 のサーバー接続チャネル QM1.SVRCONN を作成するには、次のコマンドを入力します。

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   DEFINE CHANNEL および使用可能なオプションについて詳しくは、DEFINE CHANNEL を参照してください。
2. グループ内の各リモート・キュー・マネージャーで、適切なユーザーがキュー・マネージャーへのアクセスを許可されていることを確認します。 このユーザーには、メッセージングに使用するすべてのキューまたはトピックにアクセスする権限も必要です。 ユーザーには、キュー・マネージャーに対する connect、 inquire、 alternate user、および set context 権限が必要です。 UNIX, Linux, and Windows では、コマンド行で **setmqaut** 制御コマンドを使用します。 z/OS では、 RACF プロファイルを定義して、許可ユーザーにキュー・マネージャーへのアクセス権限を付与します。
   例えば、 UNIX, Linux, and Windows では、次のコマンドを入力して、キュー・マネージャー QM1: にアクセスする権限をユーザー exampleUser に付与します。

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   どのユーザーを許可する必要があるかについて詳しくは、 716 ページの『messaging REST API が使用するセキュリティー・プリンシパルの決定』を参照してください。
3. ALW
   グループ内の各リモート・キュー・マネージャーにリスナーが存在しない場合は、着信ネットワーク接続を受け入れるリスナーを作成します。 **DEFINE LISTENER** MQSC コマンドをコマンド行で使用して、リスナーを作成できます。
   例えば、リモート・キュー・マネージャー QM1 のポート 1414 にリスナー REMOTE.LISTENER を作成するには、次のコマンドを入力します。

   ```
   runmqsc QM1
   DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
   end
   ```
4. グループ内の各リモート・キュー・マネージャーで、コマンド行で **START LISTENER** MQSC コマンドを使用して、リスナーが実行されていることを確認します。

ALW 例えば、 AIX, Linux, and Windows でキュー・マネージャー QM1 のリスナー REMOTE.LISTENER を開始するには、次のコマンドを入力します。

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS 例えば、z/OS でリスナーを開始するには、次のコマンドを入力します。

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

z/OS でリスナーを開始する前に、チャネル・イニシエーターのアドレス・スペースを開始する必要があります。

5. messaging REST API をホストする mqweb サーバーが実行されているシステムで、JSON CCDT ファイルを作成します。 この JSON ファイルには、グループ内の各キュー・マネージャーの接続情報が含まれています。

   CCDT ファイルには、キュー・マネージャー接続ごとに name、 clientConnection、および type の情報を含める必要があります。 オプションで、transmissionSecurity 情報などの追加情報を含めることができます。 すべての CCDT チャネル属性定義について詳しくは、CCDT チャネル属性定義の完全なリストを参照してください。

   以下の例は、2 つのキュー・マネージャー接続用の基本的な JSON CCDT ファイルを示しています。 最初の接続は、キュー・マネージャー QM1 用です。 サーバー接続チャネルは QM1.SVRCONN で、リスナーはポート 1414 にあり、ホスト QM1.example.com で実行されます。 2 番目の接続は、キュー・マネージャー QM2 用です。 サーバー接続チャネルは QM2.SVRCONN で、リスナーはポート 1415 にあり、ホスト QM2.example.com で実行されます。 ただし、接続はキュー・マネージャー・グループ QMGRP の一部であるため、両方の接続の **queueManager** フィールドはキュー・マネージャー・グループの名前に設定されます。

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM1.example.com",
               "port": 1414
             }],
           "queueManager": "QMGRP"
         },
         "type": "clientConnection"
      }],
      "channel": [{
         "name": "QM2.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "QM2.example.com",
               "port": 1415
             }],
           "queueManager": "QMGRP"
         },
         "type": "clientConnection"
       }]
   }
   ```

6. messaging REST API をホストする mqweb サーバーを実行しているインストール済み環境から、**setmqweb remote** コマンドを使用して、キュー・マネージャー・グループを mqweb サーバー構成に追加します。

   少なくとも、以下のパラメーターを指定する必要があります。

   - **-qmgrName**。キュー・マネージャー・グループのグループ名を指定します。
   - **-ccdtURL**。キュー・マネージャーの CCDT URL を指定します。
   - **-uniqueName**。キュー・マネージャー・グループを識別する固有の名前を指定します。

- **-group**。キュー・マネージャー情報をグループの情報として設定します。

  接続に使用するユーザー名とパスワード、トラストストアと鍵ストアの詳細など、その他のいくつかのオプションを指定できます。 **setmqweb remote** コマンドで指定できるパラメーターの完全なリストについては、 setmqweb remote を参照してください。

  例えば、CCDT ファイルの例を使用してキュー・マネージャー・グループ QMGRP を追加するには、次のコマンドを入力します。

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

タスクの結果

リモート・キュー・マネージャー・グループは、リソース URL で固有の名前を使用することにより、messaging REST API で使用できます。 要求を完了するためにグループのキュー・マネージャーが選択され、要求を完了したキュー・マネージャーに関する情報が応答ヘッダー `ibm-mq-resolved-qmgr` に返されます。

V 9.3.3 messaging REST API が使用するセキュリティー・プリンシパルの決定

messaging REST API を使用する場合は、メッセージングのために接続するキュー・マネージャー、キュー、およびトピックにアクセスする権限を適切なユーザーに付与する必要があります。 許可が必要なユーザーは、mqweb サーバーの構成方法、および messaging REST API でリモート・キュー・マネージャーを使用しているかどうかによって異なります。

デフォルトでは、キュー・マネージャーへのアクセスを許可するために使用されるセキュリティー・プリンシパルは、messaging REST API を実行する mqweb サーバーを開始するユーザーです。 キューおよびトピックへのアクセスを許可するために使用されるセキュリティー・プリンシパルは、 messaging REST API にログインしているユーザーです。 ただし、別のセキュリティー・プリンシパルが使用されるように mqweb サーバーまたはリモート・キュー・マネージャー接続が構成されている可能性があります。

キュー・マネージャーへの接続に使用されるセキュリティー・プリンシパルの判別

ローカル・キュー・マネージャー接続の場合、キュー・マネージャーへの接続に使用されるセキュリティー・プリンシパルは、messaging REST API を実行する mqweb サーバーを開始するユーザーです。 リモート・キュー・マネージャー接続の場合、 messaging REST API は以下のセキュリティー・プリンシパルを使用して、キュー・マネージャーへのアクセスを優先順に許可します。 つまり、ユーザーがリモート・キュー・マネージャー構成内で複数の方法で指定されている場合、リストの最初のものが許可に使用されます。

1. セキュリティー・プリンシパルは、セキュリティー出口から採用されたユーザー・コンテキストです。
2. セキュリティー・プリンシパルは、リモート・キュー・マネージャーへの接続に使用されるサーバー接続チャネル上の CHLAUTH 規則の採用されたユーザー・コンテキストです。
3. セキュリティー・プリンシパルは、 messaging REST API のリモート・キュー・マネージャー構成に含まれるユーザー ID です。 このユーザー ID は、 **setmqweb remote** コマンドを使用してキュー・マネージャーを追加するときに、オプションでキュー・マネージャー接続情報に組み込まれます。
4. セキュリティー・プリンシパルは、 messaging REST API を実行する mqweb サーバーを開始するユーザーです。

messaging REST API で使用するリモート・キュー・マネージャーのセットアップについて詳しくは、710 ページの『messaging REST API で使用するリモート・キュー・マネージャーのセットアップ』を参照してください。

キューおよびトピックへの接続に使用されるセキュリティー・プリンシパルの判別

mqweb サーバー構成でプロパティーを設定して、 messaging REST API の使用時にキューおよびトピックへの接続を許可するために使用するセキュリティー・プリンシパルを決定できます。 このプロパティーは **mqRestMessagingAdoptWebUserContext** プロパティーです。 **dspmqweb properties** コマンドを使用して、このプロパティーの設定内容を表示できます。

- **mqRestMessagingAdoptWebUserContext** が true に設定されている場合、 messaging REST API は、messaging REST API にログインしているユーザーのユーザー ID を許可に使用します。 したがって、messaging REST API で使用するために mqweb サーバー構成に存在するユーザー ID は、キューおよびトピックへのアクセスを許可されている必要があるセキュリティー・プリンシパルです。
- **mqRestMessagingAdoptWebUserContext** が false に設定されている場合、 messaging REST API は、messaging REST API をホストする mqweb サーバーを開始したユーザーのユーザー ID を許可に使用します。 したがって、 messaging REST API をホストする mqweb サーバーを開始するユーザー ID と同じユーザー ID に、キューおよびトピックへのアクセス権限を付与する必要があります。

  キューおよびトピックがリモート・キュー・マネージャー上にある場合、許可に使用されるセキュリティー・プリンシパルは、キュー・マネージャー構成の設定によって決定される可能性があります。 以下のセキュリティー・プリンシパルを優先度順に使用できます。

  1. セキュリティー・プリンシパルは、セキュリティー出口から採用されたユーザー・コンテキストです。
  2. セキュリティー・プリンシパルは、リモート・キュー・マネージャーへの接続に使用されるサーバー接続チャネル上の CHLAUTH 規則の採用されたユーザー・コンテキストです。 例えば、MCAUSER パラメーターを使用するように、サーバー接続チャネルで CHLAUTH 規則を構成することができます。その後、キュー・マネージャーの使用を許可されているユーザー ID にすべての接続がマップされます。
  3. セキュリティー・プリンシパルは、キュー・マネージャーの AUTHINFO から採用されたユーザー・コンテキストです。 キュー・マネージャーの CONNAUTH 属性によって参照される AUTHINFO オブジェクトが **ADOPTCTX(yes)** を使用するように構成されている場合、キュー・マネージャーへの接続を許可するために使用されるセキュリティー・プリンシパルも、キューおよびトピックを許可するために使用されます。 例えば、このセキュリティー・プリンシパルは、 **setmqweb remote** コマンドの一部としてリモート・キュー・マネージャー接続情報に含まれているユーザー ID である場合があります。

関連情報

CHLAUTH

CONNAUTH

dspmqweb プロパティー

IBM MQ での MQI アプリケーションの開発

IBM MQ は、C、Visual Basic、COBOL、アセンブラー、RPG、pTAL、および PL/I のサポートを提供します。 これらのプロシージャー型言語は、Message Queue Interface (MQI) を使用してメッセージ・キューイング・サービスにアクセスします。

選択した言語でアプリケーションを作成する方法については、サブトピックを参照してください。

プロシージャー型言語の呼び出しインターフェースの概要については、呼び出しの記述を参照してください。 このトピックには、MQI 呼び出しのリストが含まれます。各呼び出しについて、これらの各言語での呼び出しのコーディング方法を示します。

IBM MQ では、アプリケーションの作成に役立つ、データ定義ファイルを提供しています。 詳細は、718 ページの『IBM MQ データ定義ファイル』を参照してください。

プログラムをコーディングするプロシージャー型言語を選択する場合は、そのプログラムが処理するメッセージの最大長を考慮してください。 プログラムが、既知の最大長をもつメッセージのみを処理する場合は、サポートされるどの言語ででもコーディングすることができます。 プログラムが処理するメッセージの最大長が分からない場合は、作成するアプリケーションが CICS、IMS、またはバッチのどれであるかによって、選択する言語が次のように決まります。

IMS およびバッチ
: 任意の容量のストレージを取得したり解放したりするには、これらの機能を提供する C、PL/I およびアセンブラー言語でプログラムをコーディングしてください。 また、COBOL を用いてプログラムをコーディングすることもできますが、ストレージを取得したり解放したりするには、アセンブラー言語、PL/I、または C のサブルーチンを使用してください。

CICS
CICS でサポートされる任意の言語でプログラムをコーディングしてください。 EXEC CICS インターフェースは、必要に応じてストレージを管理するための呼び出しを提供しています。

関連概念
15 ページの『オブジェクト指向のアプリケーション』
IBM MQ は、JMS、Java、C++、および .NET のサポートを提供します。 これらの言語およびフレームワークは、IBM MQ オブジェクト・モデルを使用します。これは、IBM MQ の呼び出しおよび構造体と同じ機能を持つクラスを提供します。

技術概要
7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

関連資料
アプリケーションの開発に関する参照情報

IBM MQ データ定義ファイル

IBM MQ では、アプリケーションの作成に役立つ、データ定義ファイルを提供しています。

データ定義ファイルは、以下の名前でも呼ばれます。

言語	データ定義
C	組み込みファイルまたはヘッダー・ファイル
Visual Basic	モジュール・ファイル (32 ビット版のみ)
COBOL	コピー・ファイル
アセンブラー	マクロ
PL/I	組み込みファイル

チャネル出口の作成に役立つデータ定義ファイルについては、IBM MQ COPY ファイル、ヘッダー・ファイル、インクルード・ファイル、およびモジュール・ファイルで説明されています。

インストール可能なサービス出口の作成に役立つデータ定義ファイルについては、942 ページの『ユーザー出口、API 出口、および IBM MQ インストール可能サービス』で説明されています。

C++ でサポートされるデータ定義ファイルについては、C++ の使用を参照してください。

IBM i

RPG でサポートされるデータ定義ファイルについては、 IBM i アプリケーション・プログラミングの参照情報 (ILE/RPG) を参照してください。

データ定義ファイルの名前には、接頭部 CMQ と、プログラム言語によって異なる接尾部が付いています。

接尾部	言語
a	アセンブラー言語
b	Visual Basic
c	C
l	COBOL (初期設定値なし)
p	PL/I
v	COBOL (デフォルト値設定あり)

インストール・ライブラリー

thlqual は、z/OS のインストール・ライブラリーの高水準修飾子を表します。

このトピックでは、以下の見出しのもとで、IBM MQ データ定義ファイルについて説明します。

- 719 ページの『C 言語組み込みファイル』
- 719 ページの『Visual Basic モジュール・ファイル』
- 719 ページの『COBOL コピー・ファイル』
- z/OS 721 ページの『System/390 アセンブラー言語マクロ』
- z/OS 721 ページの『PL/I 組み込みファイル』

C 言語組み込みファイル

IBM MQ C の組み込みファイルは、C ヘッダー・ファイルにリストされています。 これらのファイルは次のディレクトリーまたはライブラリーにインストールされます。

プラットフォーム	インストール・ディレクトリーまたはライブラリー
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/インク/
Windows Windows	*MQ_INSTALLATION_PATH*¥Tools¥c¥include
z/OS z/OS	**thlqual**.SCSQC370

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

注 : AIX and Linux では、組み込みファイルは `/usr/include` にシンボリック・リンクされます。

ディレクトリーの構造について詳しくは、ファイル・システム・サポートの計画を参照してください。

Visual Basic モジュール・ファイル

IBM MQ for Windows では、4 つの Visual Basic モジュール・ファイルが提供されます。

これは Visual Basic モジュール・ファイルにリストされており、次の場所にインストールされます。

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

COBOL コピー・ファイル

COBOL の場合、IBM MQ は、名前付きの定数を含む個別のコピー・ファイルと、各構造体ごとに 2 つのコピー・ファイルを提供します。

各構造体別の 2 つのコピー・ファイルがありますが、これは初期値がないものと、初期値があるものがそれぞれ提供されているからです。

- COBOL プログラムの WORKING-STORAGE SECTION では、構造体フィールドをデフォルト値に初期化するファイルを使用します。 これらの構造体は、文字「V」(値) を接尾部にもつコピー・ファイル名に定義されます。
- COBOL プログラムの LINKAGE SECTION では、初期値なしの構造体を使用します。 これらの構造体は、文字「L」(連係) が後ろに付いた名前をもつコピー・ファイルで定義されます。

IBM i IBM i のデータおよびインターフェース定義を含むコピー・ファイルが、MQI に対するプロトタイプ呼び出しを使用する ILE COBOL プログラム用に提供されています。 ファイルは、QMQM/

QCBLLESRC に入っており、これには接尾部として「L」(初期値を持たない構造体の場合) または「V」(初期値を持つ構造体の場合) の付いたメンバー名があります。

IBM MQ COBOL コピー・ファイルは、COBOL コピー・ファイルにリストされています。 次のディレクトリーにインストールされます。

プラットフォーム	インストール・ディレクトリーまたはライブラリー
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*¥Tools¥cobol¥copybook (Micro Focus COBOL 用) *MQ_INSTALLATION_PATH*¥Tools¥cobol¥copybook ¥VAcobol (IBM VisualAge® COBOL 用)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

必要なファイルのみをプログラムに組み込みます。 これを、レベル 01 の宣言後に 1 つまたは複数の COPY ステートメントによって行います。 つまり、必要に応じてプログラムに複数バージョンの構造体を組み込むことができます。 ただし、CMQV はファイル・サイズが大きいので注意してください。

次に、CMQMDV コピー・ファイルを組み込むための COBOL コードの例を示します。

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

各構造体の宣言は、レベル 01 の項目で始まります。つまり、構造体宣言の残りの部分にコピーする COPY ステートメントの前で、レベル 01 の宣言をコーディングすることによって、いくつかの構造体のインスタンスを宣言できることを意味します。 適切なインスタンスを参照するには、IN キーワードを使用してください。

次に、CMQMDV の 2 つのインスタンスを組み込むための COBOL コードの例を示します。

```
* Declare two instances of MQMD
01 MY-CMQMD.
COPY CMQMDV.
01 MY-OTHER-CMQMD.
COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

構造体を 4 バイト境界に位置合わせしてください。 COPY ステートメントを使用してレベル 01 項目以外の項目の後ろに構造体を組み込む場合には、その構造体がレベル 01 項目の開始から 4 バイトの倍数になっていることを確認してください。 これを無視すると、アプリケーションのパフォーマンスが低下する可能性があります。

構造体については、MQI で使用されるデータ・タイプを参照してください。 構造体内のフィールドの記述は、接頭部なしのフィールドの名前を示します。 COBOL プログラムでは、COBOL 宣言で示すように、フィールド名の接頭部に構造体の名前とそれに続くハイフンを付けてください。 構造体コピー・ファイル内のフィールドは、上記の方法で接頭部が付けられます。

構造体コピー・ファイル内の宣言でのフィールド名は、大文字です。 大文字と小文字を混在させたり、大文字の代わりに小文字を使用しても構いません。 例えば、MQGMO 構造体のフィールド *StrucId* は、COBOL 宣言およびコピー・ファイル内の MQGMO-STRUCID として示されます。

接尾部 V の構造体はすべてのフィールドに対して初期値で宣言されます。したがって、必要とされる値が初期値と異なる場合には、これらのフィールドのみを設定するだけで済みます。

System/390 アセンブラー言語マクロ

z/OS

IBM MQ for z/OS により、名前付きの定数を含む 2 つのアセンブラー言語マクロと各構造体を生成するための 1 つのマクロが提供されます。

これは、 z/OS Assembler コピー・ファイルにリストされており、**thlqual**.SCSQMACS にインストールされます。

これらのマクロは、次のようなコードを使用して呼び出されます。

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

PL/I 組み込みファイル

z/OS

IBM MQ for z/OS では、IBM MQ アプリケーションを PL/I で作成する際に必要なすべての定義を含んだ組み込みファイルを提供しています。

これらのファイルは、PL/I インクルード・ファイルにリストされており、**thlqual**.SCSQPLIC ディレクトリーにインストールされます。

IBM MQ スタブを各自のプログラムにリンクする場合には、それらのファイルをそのプログラムに組み込んでください (1028 ページの『実行するプログラムの作成』を参照)。 IBM MQ 呼び出しを動的にリンクしたい場合には、CMQP のみを組み込んでください (1035 ページの『IBM MQ スタブの動的呼び出し』 を参照)。 ダイナミック・リンクは、バッチ・プログラムおよび IMS プログラムに対してのみ行うことができます。

プロシージャー型キューイング・アプリケーションの作成

この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

以下のリンクを使用して、アプリケーションの作成についての詳細を確認してください。

- 722 ページの『Message Queue Interface の概要』
- 735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
- 742 ページの『オブジェクトのオープンとクローズ』
- 753 ページの『キューへのメッセージの書き込み』
- 768 ページの『キューからのメッセージの読み取り』
- 809 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
- 854 ページの『オブジェクト属性の照会と設定』
- 857 ページの『作業単位のコミットとバックアウト』
- 869 ページの『トリガーによる IBM MQ アプリケーションの開始』
- 888 ページの『MQI とクラスターの処理』
- z/OS 893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
- z/OS 70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』

関連概念

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

5 ページの『IBM MQ 用アプリケーションの開発』

メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

918 ページの『プロシージャー型クライアント・アプリケーションの作成』
IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

1005 ページの『プロシージャー型アプリケーションの構築』
IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

1042 ページの『プロシージャー型プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

関連タスク

1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』
これらのサンプル・プログラムは、プロシージャー型言語で作成されており、Message Queue Interface (MQI) の標準的な使用法を示しています。 異なるプラットフォーム上の IBM MQ プログラム。

Message Queue Interface の概要

メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

メッセージ・キュー・インターフェースは以下のもので成り立っています。

- 呼び出し (プログラムはこれを使用してキュー・マネージャーとその機能にアクセスすることができる)
- 構造体 (プログラムはこれを使用してキュー・マネージャーにデータを渡したり、データを読み取ったりする)
- 基本データ・タイプ (キュー・マネージャーにデータを渡したり、データを読み取ったりする)

z/OS IBM MQ for z/OS は以下のものも提供します。

- 2 つの特別な呼び出し (これらの呼び出しを介して、z/OS バッチ・プログラムは変更内容のコミットとバックアウトができる)
- IBM MQ for z/OS で提供される定数の値を定義する データ定義ファイル (コピー・ファイル、マクロ、インクルード・ファイル、およびヘッダー・ファイルと呼ばれることもある)。
- スタブ・プログラム (アプリケーションにリンク・エディットするためのもの)
- z/OS プラットフォーム上での MQI の使用方法を示すための一連のサンプル・プログラム。 このサンプルの詳細については、1166 ページの『z/OS 用サンプル・プログラムの使用』を参照してください。

IBM i IBM MQ for IBM i は以下のものも提供します。

- IBM MQ for IBM i で提供される定数の値を定義する データ定義ファイル (コピー・ファイル、マクロ、インクルード・ファイル、およびヘッダー・ファイルと呼ばれることもある)。
- 3 つのスタブ・プログラム (ILE C、ILE COBOL、および ILE RPG アプリケーションにリンク・エディットするためのもの)
- IBM i プラットフォーム上での MQI の使用方法を示すための一連のサンプル・プログラム。

AIX, Linux, and Windows システムは以下のものも提供します。

- IBM MQ for AIX, Linux, and Windows システム・プログラムが変更をコミットおよびバックアウトできる呼び出し。
- これらのプラットフォームで提供される定数の値を定義する組み込みファイル。
- アプリケーションにリンクするライブラリー・ファイル 。

- これらのプラットフォーム上での MQI の使用方法を示すための一連のサンプル・プログラム。 このサンプルの詳細については、1063 ページの『Multiplatforms でのサンプル・プログラムの使用』を参照してください。
- 外部トランザクション管理プログラムにバインドするためのサンプル・ソースおよび実行可能コード。

MQI の詳細については、以下のリンクを使用してください。

- 723 ページの『MQI 呼び出し』
- 724 ページの『同期点の呼び出し』
- 725 ページの『データ変換、データ・タイプ、データ定義、および構造体』
- 726 ページの『IBM MQ スタブ・プログラムおよびライブラリー・ファイル』
- 731 ページの『すべての呼び出しに共通のパラメーター』
- 731 ページの『バッファーの指定』
- z/OS 732 ページの『z/OS バッチの考慮事項』
- 733 ページの『AIX and Linux 信号処理』

関連概念

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQI 呼び出し

この情報を使用して、Message Queue Interface (MQI) での呼び出しについて学習します。

MQI の呼び出しは、次のように分類できます。

MQCONN、MQCONNX、および MQDISC
プログラムをキュー・マネージャーに接続したり (オプションを指定してまたは指定せずに)、キュー・マネージャーから切断したりするために使用します。 z/OS 用の CICS プログラムを作成する場合は、これらの呼び出しを使用する必要はありません。 ただし、作成するアプリケーションを他のプラットフォームに移植したい場合には、これらの呼び出しを使用することをお勧めします。

MQOPEN および MQCLOSE
キューなどのオブジェクトをオープンしたり、クローズするために使用します。

MQPUT および MQPUT1
メッセージをキューに書き込むために使用します。

MQGET
キュー上のメッセージをブラウズしたり、キューからメッセージを除去するために使用します。

MQSUB、MQSUBRQ
トピックにサブスクリプションを登録するためと、サブスクリプションに一致するパブリケーションを要求 するために使用します。

MQINQ
オブジェクトの属性を照会するために使用します。

MQSET
キューのいくつかの属性を設定するために使用します。 別のタイプのオブジェクトの属性を設定することはできません。

MQBEGIN、MQCMIT、および MQBACK
IBM MQ が作業単位の調整役であるときに、これらの呼び出しを使用します。 MQBEGIN は作業単位を開始します。 MQCMIT と MQBACK は作業単位を終了し、それぞれ、作業単位中に作成された更新をコミットするか、またはロールバックします。 IBM i IBM i コミットメント制御プログラムは、IBM MQ for IBM i のグローバル作業単位を調整するために使用します。 ネイティブのコミットメント制御開始、コミット、およびロールバック・コマンドが使用されます。

MQCRTMH、MQBUFMH、MQMHBUF、MQDLTMH
メッセージ・ハンドルを作成するため、メッセージ・ハンドルをバッファーに、またはバッファーをメッセージ・ハンドルに 変換するため、およびメッセージ・ハンドルを削除するために使用します。

MQSETMP、MQINQMP、MQDLTMP
メッセージ・ハンドルにメッセージ・プロパティーを設定するため、メッセージ・プロパティーを 照会するため、およびメッセージ・ハンドルからプロパティーを削除するために使用します。

MQCB、MQCB_FUNCTION、MQCTL
コールバック機能の登録および制御のために使用します。

MQSTAT
前の非同期書き込み操作に関する状況情報を取得するために 使用します。

MQI 呼び出しの説明については、呼び出しの記述を参照してください。

同期点の呼び出し
各種のプラットフォームにおける同期点の呼び出しについて理解するために、この情報を役立ててください。

次のようにして、同期点呼び出しを利用することができます。

IBM MQ for z/OS 呼び出し
z/OS

IBM MQ for z/OS は、MQCMIT 呼び出しと MQBACK 呼び出しを提供します。

最後の同期点以降のすべての MQGET および MQPUT 操作を永続的に行う (コミットする) か、バックアウトすることをキュー・マネージャーに知らせるために、これらの呼び出しを z/OS バッチ・プログラム内で用います。 他の環境で変更をコミットあるいはバックアウトする方法は、次のとおりです。

CICS
EXEC CICS SYNCPOINT および EXEC CICS SYNCPOINT ROLLBACK などのコマンドを使用します。

IMS
IOPCB に対する GU (get unique)、CHKP (checkpoint)、および ROLB (rollback) 呼び出しなどの IMS 同期点機能を使用します。

RRS

MQCMIT と MQBACK または SRRCMIT と SRRBACK のどちらか適切な組み合わせを使用します。(862 ページの『トランザクション管理とリカバリー可能リソース管理サービス』を参照してください)。

注 : SRRCMIT と SRRBACK は「ネイティブの」RRS コマンドであり、MQI 呼び出しではありません。

IBM i 呼び出し

IBM i

IBM MQ for IBM i は、MQCMIT コマンドと MQBACK コマンドを提供します。 IBM i の COMMIT および ROLLBACK コマンドか、あるいは IBM i コミットメント制御機能を開始させる他のコマンドまたは呼び出し (例えば、EXEC CICS SYNCPOINT) を使用することもできます。

AIX, Linux, and Windows プラットフォームでの IBM MQ 呼び出し

ALW

IBM MQ for AIX, Linux, and Windows は、MQCMIT 呼び出しと MQBACK 呼び出しを提供します。

同期点呼び出しをプログラムで使用して、最後の同期点以降のすべての MQGET および MQPUT 操作を永続的に行う (コミットする) か、バックアウトすることをキュー・マネージャーに通知します。 CICS 環境における変更をコミット、およびバックアウトするには、EXEC CICS SYNCPOINT および EXEC CICS SYNCPOINT ROLLBACK などのコマンドを使用してください。

データ変換、データ・タイプ、データ定義、および構造体

Message Queue Interface を使用する際の、データ変換、基本データ・タイプ、IBM MQ データ定義、および構造体について理解するために、この情報を使用します。

データ変換

MQXCNVC (文字変換) 呼び出しは、メッセージ文字データをある文字セットから別の文字セットに変換します。 IBM MQ for z/OS を除き、この呼び出しは、データ変換出口からのみ使用されます。

MQXCNVC 呼び出しで使用される構文については、MQXCNVC - 文字の変換を参照してください。また、データ変換出口の作成方法と呼び出し方法については、988 ページの『データ変換出口の作成』を参照してください。

基本データ・タイプ

サポートされるプログラム言語の場合、MQI は基本データ・タイプまたは構造化されていないフィールドを提供します。

これらのデータ・タイプについては、基本データ・タイプで詳しく説明されています。

IBM MQ データ定義

z/OS IBM MQ for z/OS では、COBOL コピー・ファイル、アセンブリー言語マクロ、単一 PL/I 組み込みファイル、単一 C 言語組み込みファイル、および C++ 言語組み込みファイルの形式で、データ定義を提供します。

IBM i IBM MQ for IBM i では、COBOL コピー・ファイル、RPG コピー・ファイル、C 言語組み込みファイル、および C++ 言語組み込みファイルの形式でデータ定義が提供されます。

IBM MQ によって提供されるデータ定義ファイルには、以下のものがあります。

- すべての IBM MQ 定数および戻りコードの定義
- IBM MQ 構造体およびデータ・タイプの定義
- 構造体の初期化用の定数
- 各呼び出しの関数原型 (PL/I および C 言語のみ)

IBM MQ データ定義ファイルの詳細については、718 ページの『IBM MQ データ定義ファイル』 を参照してください。

構造体

723 ページの『MQI 呼び出し』 のリストにある MQI 呼び出しで使用される構造体は、サポートされている各プログラム言語用のデータ定義ファイルに含まれています。IBM i z/OS IBM MQ for z/OS および IBM MQ for IBM i は、これらの構造体の一部のフィールドに入力する際に使用する定数を含むファイルを提供しています。 詳細については、 IBM MQ データ定義を参照してください。

構造体の要約については、 構造体データ・タイプ を参照してください。

IBM MQ スタブ・プログラムおよびライブラリー・ファイル

提供されるスタブ・プログラムおよびライブラリー・ファイルは、プラットフォームごとにここにリストされます。

実行可能アプリケーションを作成するときのスタブ・プログラムとライブラリー・ファイルの使用方法について詳しくは、1005 ページの『プロシージャー型アプリケーションの構築』 を参照してください。 C++ ライブラリー・ファイルへのリンクについては、「C++ の使用 *IBM MQ C++* の使用」を参照してください。

AIX *IBM MQ for AIX* ライブラリー・ファイル

IBM MQ for AIX では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

スレッド以外のアプリケーションでは、次のいずれかのライブラリーにリンクしてください。

表 *106.* スレッド化されていない *AIX* アプリケーション用のライブラリー・ファイル

ライブラリー・ファイル	**Environment**
libmqm.a	C 用サーバー
libmqic.a および libmqm.a	C 用クライアント
libmqmzf.a	C でインストール可能なサービス出口
libmqmxa.a	サーバー XA インターフェース
libmqmxa64.a	サーバー代替 XA インターフェース
libmqcxa.a	クライアント XA インターフェース
libmqcxa64.a	クライアント代替 XA インターフェース
libmqmcbrt.o	Micro Focus COBOL サポート用 IBM MQ 実行時ライブラリー
libmqmcb.a	COBOL 用サーバー
libmqicb.a	COBOL 用クライアント
libimqc23ia.a	C++ 用クライアント (XLC 16)
libimqs23ia.a	C++ 用サーバー (XLC 16)
V 9.3.5 libimqc23ca.a	C++ 用クライアント (XLC 17)
V 9.3.5 libimqs23ca.a	C++ 用サーバー (XLC 17)

V 9.3.5 "ia" を含むライブラリは XLC 16 コンパイラでビルドされており、"ca" を含むライブラリは XLC 17 コンパイラでビルドされている。

スレッド・アプリケーションでは、次のいずれかのライブラリーにリンクしてください。

表 107. スレッド化された *AIX* アプリケーション用のライブラリー・ファイル.

ライブラリー・ファイルと、それぞれのライブラリー・ファイルの環境をリストした、2 列で構成される表。

ライブラリー・ファイル	Environment
libmqm_r.a	C 用サーバー
libmqic_r.a and libmqm_r.a	C 用クライアント
libmqmzf_r.a	C でインストール可能なサービス出口
libmqmxa_r.a	サーバー XA インターフェース
libmqmxa64_r.a	サーバー代替 XA インターフェース
libmqcxa_r.a	クライアント XA インターフェース
libmqcxa64_r.a	クライアント代替 XA インターフェース
libimqc23ia_r.a	C++ 用クライアント (XLC 16)
libimqs23ia_r.a	C++ 用サーバー (XLC 16)
V 9.3.5 libimqc23ca_r.a	C++ 用クライアント (XLC 17)
V 9.3.5 libimqs23ca_r.a	C++ 用サーバー (XLC 17)

V 9.3.5 "ia" を含むライブラリは XLC 16 コンパイラでビルドされており、"ca" を含むライブラリは XLC 17 コンパイラでビルドされている。

注 : 複数のライブラリーにリンクすることはできません。 そのため、スレッド化ライブラリーと非スレッド化ライブラリーの両方に同時にリンクすることはできません。

IBM i *IBM MQ for IBM i* ライブラリー・ファイル

IBM MQ for IBM i では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクしてください。

スレッド化されていないアプリケーションでは、次のようになります。

表 108. スレッド化されていない *IBM i* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**Environment**
LIBMQM	サーバーおよびクライアント・サービス・プログラム
LIBMQIC	クライアント・サービス・プログラム
IMQB23I4	C++ 基本サービス・プログラム
IMQS23I4	C++ サーバー・サービス・プログラム
LIBMQMZF	C 用のインストール可能出口

スレッド化されたアプリケーションでは、次のようになります。

表 109. スレッド化された *IBM i* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**Environment**
LIBMQM_R	サーバーおよびクライアント・サービス・プログラム

表 109. スレッド化された IBM i アプリケーション用のライブラリー・ファイル (続き)	
ライブラリー・ファイル	**Environment**
IMQB23I4_R	C++ 基本サービス・プログラム
IMQS23I4_R	C++ サーバー・サービス・プログラム
LIBMQMZF_R	C 用のインストール可能出口
LIBMQIC_R	クライアント・サービス・プログラム

IBM MQ for IBM i では、C++ でアプリケーションを書き込むことができます。C++ アプリケーションのリンク方法、および C++ の使用に関するあらゆる状況の詳細については、C++ の使用を参照してください。

Linux *IBM MQ for Linux* ライブラリー・ファイル

IBM MQ for Linux では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

スレッド以外のアプリケーションでは、次のいずれかのライブラリーにリンクしてください。

表 110. スレッド化されていない Linux アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**Environment**
libmqm.so	C 用サーバー
libmqic.so および libmqm.so	C 用クライアント
libmqmzf.so	C でインストール可能なサービス出口
libmqmxa.so	サーバー XA インターフェース
libmqmxa64.so	サーバー代替 XA インターフェース
libmqcxa.so	クライアント XA インターフェース
libmqcxa64.so	クライアント代替 XA インターフェース
libimqc23gl.so	C++ 用クライアント
libimqs23gl.so	C++ 用サーバー

スレッド・アプリケーションでは、次のいずれかのライブラリーにリンクしてください。

表 111. スレッド化された Linux アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**Environment**
libmqm_r.so	C 用サーバー
libmqic_r.so and libmqm_r.so	C 用クライアント
libmqmzf_r.so	C でインストール可能なサービス出口
libmqmxa_r.so	サーバー XA インターフェース
libmqmxa64_r.so	サーバー代替 XA インターフェース
libmqcxa_r.so	クライアント XA インターフェース
libmqcxa64_r.so	クライアント代替 XA インターフェース
libimqc23gl_r.so	C++ 用クライアント
libimqs23gl_r.so	C++ 用サーバー

注 : 複数のライブラリーにリンクすることはできません。 そのため、スレッド化ライブラリーと非スレッド化ライブラリーの両方に同時にリンクすることはできません。

Windows *IBM MQ for Windows* ライブラリー・ファイル

IBM MQ for Windows では、ご使用のアプリケーションを実行している環境用に提供されている MQI ライブラリー・ファイルにプログラムをリンクする必要があります (オペレーティング・システムによって提供されるものに加えて)。

表 *112. Windows* アプリケーション用のライブラリー・ファイル

ライブラリー・ファイル	**Environment**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	C 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	C 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	C 用サーバー XA インターフェース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	C 用クライアント XA インターフェース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	C 用クライアント MTS (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	C (32 ビット) 用のサーバー TXSeries CICS サポート
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	C (32 ビット) 用のクライアント TXSeries CICS サポート
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	C 用インストール可能サービス出口 (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	IBM COBOL 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Micro Focus COBOL 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	IBM COBOL 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Micro Focus COBOL 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	C++ 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	C++ 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	C++ 用ベース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	C++ 用クライアント MTS (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	C 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	C 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	C 用サーバー XA インターフェース (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	C 用クライアント XA インターフェース (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	C 用クライアント MTS (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	IBM COBOL 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Micro Focus COBOL 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	IBM COBOL 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Micro Focus COBOL 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	C++ 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	C++ 用クライアント (64 ビット)

表 112. Windows アプリケーション用のライブラリー・ファイル (続き)	
ライブラリー・ファイル	**Environment**
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	C++ 用ベース (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	C++ 用クライアント MTS (64 ビット)

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

.NET プログラムをコンパイルする場合、amqmdnet.dll を使用します。 詳しくは、セクション 553 ページの『.NET アプリケーションの開発』内の 611 ページの『IBM MQ .NET プログラムのコンパイル』を参照してください。

次の各ファイルは、以前のリリースとの互換性を保つために出荷されます。

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS スタブ・プログラム*

IBM MQ for z/OS で作成されたプログラムを実行する場合は、そのプログラムと、アプリケーションを実行している環境用に IBM MQ for z/OS が提供するスタブ・プログラムをリンク・エディットしなければなりません。

そのスタブ・プログラムは、要求する呼び出し (IBM MQ for z/OS が処理できるもの) を処理する最初の段階を提供します。

IBM MQ for z/OS で提供されているスタブ・プログラムは、次のとおりです。

CSQBSTUB
z/OS バッチ・プログラム用のスタブ・プログラム

CSQBRRSI
MQI を経由して RRS を使用する z/OS バッチ・プログラム用のスタブ・プログラム

CSQBRSTB
直接 RRS を使用する z/OS バッチ・プログラム用のスタブ・プログラム

CSQCSTUB
CICS バッチ・プログラム用のスタブ・プログラム

CSQQSTUB
IMS バッチ・プログラム用のスタブ・プログラム

CSQXSTUB
分散キューイングの CICS 以外の出口用のスタブ・プログラム

CSQASTUB
データ変換出口用のスタブ・プログラム

重要 : 特定の環境用にリストされたもの以外のスタブ・プログラムを使用すると、予期しない結果が発生する可能性があります。

注 : CSQBRSTB スタブ・プログラムを使用する場合、SYS1.CSSLIB の ATRSCSS とリンク・エディットしてください。 (SYS1.CSSLIB は呼び出し可能サービス・ライブラリー (CSL) としても知られています。) RRS についての詳細は、862 ページの『トランザクション管理とリカバリー可能リソース管理サービス』を参照してください。

別の方法として、使用しているプログラムの中からスタブを動的に呼び出すことができます。 この手法については、1035 ページの『IBM MQ スタブの動的呼び出し』を参照してください。

IMS では、特殊な言語のインターフェース・モジュールの使用が必要になる場合もあります。このモジュールは IBM MQ によって提供されます。

同じ IMS MPP 領域にある CSQBSTUB および CSQQSTUB とリンク・エディットされているアプリケーションは実行しないでください。 これにより、DFS3607I や CSQQ005E メッセージなどの問題が発生する場合

があります。 アドレス・スペースの最初の MQCONN 呼び出しによって使用インターフェースが決まるため、トランザクション CSQQSTUB と CSQBSTUB は、異なる IMS メッセージ領域で実行する必要があります。

すべての呼び出しに共通のパラメーター

すべての呼び出しに共通のパラメーターには、ハンドルと戻りコードの 2 種類があります。

ハンドルの使用

すべての MQI 呼び出しは、1 つ以上のハンドル を使用します。 ハンドルは、 キュー・マネージャー、キューまたは他のオブジェクト、メッセージ、またはサブスクリプションを、呼び出しに応じて適切に識別します。

キュー・マネージャーと通信するプログラムには、そのキュー・マネージャーを識別するための固有 ID がなければなりません。 この ID は接続ハンドル と呼ばれ、*Hconn* と表される場合もあります。 CICS プログラムの場合は、接続ハンドルが常にゼロです。 他のすべてのプラットフォームやプログラムのスタイルでは、プログラムがキュー・マネージャーに接続するときに接続ハンドルが MQCONN 呼び出しまたは MQCONNX 呼び出しによって戻されます。 プログラムは、他の呼び出しを使用するときに、接続ハンドルを入力パラメーターとして渡します。

IBM MQ オブジェクトで作業するプログラムには、そのオブジェクトを識別するための固有の ID がなければなりません。 この ID はオブジェクト・ハンドル と呼ばれ、*Hobj* と表される場合もあります。 オブジェクト・ハンドルは、プログラムがオブジェクトを使用するためにオープンしたときに、MQOPEN 呼び出しによって戻されます。 プログラムは、後続の MQPUT、MQGET、MQINQ、MQSET、または MQCLOSE 呼び出しを使用するときに、オブジェクト・ハンドルを入力パラメーターとして渡します。

同様に、MQSUB 呼び出しはサブスクリプション・ハンドル (*Hsub* とも呼ばれる) を戻し、 このハンドルは、後続の MQGET、MQCB、または MQSUBRQ 呼び出し でサブスクリプションを識別するために使用されます。また、メッセージ・プロパティーを処理する一定の呼び出しは、メッセージ・ハンドル (*Hmsg* とも呼ばれる) を使用します。

戻りコードの理解

完了コードおよび理由コードは、各呼び出しの出力パラメーターとして戻されます。 これらをまとめて戻りコード と呼びます。

呼び出しが成功したかどうかを示すために、各呼び出しは、完了時に完了コード を戻します。 完了コードは通常、成功を示す MQCC_OK か、失敗を示す MQCC_FAILED です。 ある呼び出しは、中間の状態の MQCC_WARNING (一部成功) を戻すことがあります。

各呼び出しは、呼び出しの失敗や一部成功の理由を示す理由コード も戻します。 キューが満ぱいである、キューに対して読み取り操作が許されていない、ある特定のキューがキュー・マネージャーに対して定義されていない、などの状況を示す多数の理由コードがあります。 プログラムは、理由コードを用いて、どのように進むべきかを判断することができます。 例えば、入力データを変更してから再度呼び出しを実行するようユーザーにプロンプトを出したり、あるいはエラー・メッセージをユーザーに戻したりすることがあります。

完了コードが MQCC_OK のときは、理由コードは常に MQRC_NONE です。

各呼び出しに対する完了コードおよび理由コードが、その呼び出しの説明と共にリストされています。 呼び出しの記述で、リストから該当する呼び出しを選択し、参照してください。

修正処置のアイデアを含む詳細については、以下を参照してください。

- z/OS IBM MQ for z/OS のメッセージ、完了コード、および理由コードの IBM MQ for z/OS
- メッセージおよび理由コード (その他のすべての IBM MQ プラットフォームの場合)

バッファーの指定

キュー・マネージャーは、要求されたときだけバッファーを参照します。 呼び出し時にバッファーを必要としない場合、またはバッファーの長さが 0 の場合、バッファーへの NULL ポインターを使用することができます。

必要なバッファーのサイズを指定するとき、常にデータ長を使用します。

呼び出しからの出力を保持するためにバッファーを使用する (例えば、MQGET 呼び出し用のデータや、MQINQ 呼び出しで照会された属性の値を保持するために使用する) ときに、指定したバッファーが無効であったり読み取り専用ストレージであったりすると、キュー・マネージャーは理由コードを戻そうとします。 しかし、常に理由コードを戻せるとは限りません。

z/OS *z/OS バッチの考慮事項*

MQI を呼び出す z/OS バッチ・プログラムは、監視状態または問題プログラム状態のいずれかに含めることができます。

しかし、次の条件を満たす必要があります。

- プログラムはタスク・モードであって、サービス要求ブロック (SRB) モードであってはならない。
- プログラムは 1 次アドレス・スペース制御 (ASC) モードであって、アクセス・レジスター ASC モードであってはならない。
- プログラムは仮想記憶間モードであってはならない。 1 次アドレス・スペース番号 (ASN) は、2 次 ASN およびホーム ASN に等しくなければならない。
- プログラムは、MPF 出口プログラムとしては使用できない。
- z/OS ロックを保持することはできない。
- 機能リカバリー・ルーチン (FRR) を FRR スタックに入れることはできない。
- MQCONN または MQCONNX 呼び出しではどのプログラム状況ワード (PSW) キーでも有効にできるが (ただし、TCB キー内にあるストレージを使用してキーの互換性がある場合に限る)、MQCONN または MQCONNX によって戻される接続ハンドルを使用する後続の呼び出しは、次の条件を満たさなければならない。
  - MQCONN または MQCONNX 呼び出しで使用されたものと同じ PSW キーを持たなければならない。
  - 同じ PSW キーでアクセス可能な (必要な場合、書き込みのために) パラメーターを持たなければならない。
  - 同じタスク (TCB) の下で出さなくてはならない。ただし、タスクのサブタスクでは出してはならない。
- 24 ビットまたは 31 ビットのどちらのアドレッシング・モードであってもよい。 ただし、24 ビット・アドレッシング・モードが有効な場合は、パラメーター・アドレスを有効な 31 ビット・アドレスとして解釈しなければならない。

上記の条件が 1 つでも満たされないと、プログラム・チェックが起こる可能性があります。 場合によっては、呼び出しが失敗し、理由コードが戻されます。

Linux AIX *AIX and Linux 考慮事項*

AIX and Linux アプリケーションを開発する際に注意が必要な考慮事項。

Linux AIX *AIX and Linux システム内での fork システム呼び出し*

IBM MQ アプリケーションで fork システム呼び出しを使用するときには、以下の考慮事項にご注意ください。

アプリケーションが `fork` を使用する場合、そのアプリケーションの親プロセスは、 IBM MQ 呼び出し (例えば、 `MQCONN`) を行う前、または **`ImqQueueManager`** を使用して IBM MQ オブジェクトを作成する前に、`fork` を呼び出す必要があります。

アプリケーションが何らかの IBM MQ 呼び出しを行った後で子プロセスを作成する場合には、アプリケーション・コードは `fork()` を `exec()` と共に使用し、子が親の正確なコピーではなく、新しいインスタンスとなるようにしなければなりません。

アプリケーションが `exec()` を使用しないと、子プロセス内での IBM MQ API 呼び出しから MQRC_ENVIRONMENT_ERROR が返されます。

Linux AIX *AIX and Linux* 信号処理

一般に、AIX and Linux システムは、スレッド化されていない (プロセス) 環境からマルチスレッド環境に移行しています。 しかし、サポートされていても信号や信号処理がマルチスレッド環境に合っていなかったり、いくつもの制限が課せられていることがよく あります。

一般に、AIX and Linux システムは、スレッド化されていない (プロセス) 環境からマルチスレッド環境に移行しています。 スレッド化されていない環境では、たいていのアプリケーションは信号や信号処理を認識する必要はありませんでしたが、信号を使った場合にしか実現できない機能もありました。 マルチスレッド環境では、スレッド・ベースのプリミティブにより、これまでスレッド化されていない環境で信号を使って実現されていた機能がいくつかサポートされています。

しかし、サポートされていても信号や信号処理がマルチスレッド環境に合っていなかったり、いくつもの制限が課せられていることがよく あります。 こういった問題が発生するのは、アプリケーション・コードをマルチスレッド環境の (そのアプリケーションの一部として 稼働している) さまざまなミドルウェア・ライブラリーと統合していて、それらのライブラリーが独立して信号処理を行っている場合です。 プロセスごとに 定義された信号ハンドラーを保存して復元する従来の方法は、1 つのプロセス内に実行スレッドが 1 つしかない場合には有効でしたが、マルチスレッド環境では 機能しません。 その理由は、多数の実行スレッドがそれぞれプロセス全体のリソースの保存と復元を実行するために、予測不能な結果になるからです。

Linux AIX スレッド化されていないアプリケーション

各 MQI 機能は、それぞれ、次の信号に対する独自の信号ハンドラーを設定します。 これらの信号を処理するためにユーザーが作成したハンドラーは、MQI 機能呼び出しが実行されている間に置換されます。 これ以外の信号は、ユーザー作成のハンドラーを使って通常の方法で取り込むことができます。

各 MQI 機能は、それぞれ、次の信号に対する独自の信号ハンドラーを設定します。

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

これらの信号を処理するためにユーザーが作成したハンドラーは、MQI 機能呼び出しが実行されている間に置換されます。 これ以外の信号は、ユーザー作成のハンドラーを使って通常の方法で取り込むことができます。 ハンドラーをインストールしていない場合は、デフォルト・アクション (例えば、無視、メモリー・ダンプ、終了) がそのまま実行されます。

IBM MQ が同期信号 (SIGSEGV、SIGBUS、SIGFPE、SIGILL) の処理を行うと、MQI 機能呼び出しを行う前に、登録済みの任意の信号ハンドラーに信号を渡そうとします。

Linux AIX スレッド化されたアプリケーション

1 つのスレッドが IBM MQ に関連付けられていると見なされる期間は、MQCONN (または MQCONNX) が実行されてから MQDISC が実行されるまでの間です。

同期信号

同期信号は特定のスレッドで発生します。

AIX and Linux システムでは、安全確保のために、プロセス全体に渡って同期信号を処理する信号ハンドラーを設定することができます。 ただし、スレッドが IBM MQ に接続されている間に、IBM MQ はアプリケーション・プロセスにおいて以下の信号を処理する独自のハンドラーを設定します。

SIGBUS
SIGFPE
SIGSEGV
SIGILL

マルチスレッド・アプリケーションを作成している場合、各信号を処理する信号ハンドラーはプロセス全体で 1 つしか設定できません。 IBM MQ は、独自の同期信号ハンドラーをセットアップすると、各信号に

事前登録されたすべてのハンドラーを保存します。 IBM MQ が上記の信号の 1 つを処理した後、IBM MQ は、そのプロセス内の最初の IBM MQ 接続が行われた時点で有効だった信号ハンドラーの呼び出しを試行します。 すべてのアプリケーションのスレッドが IBM MQ から切断されると、事前に登録されたハンドラーが復元されます。

信号ハンドラーは IBM MQ によって保存および復元されるため、同じプロセス内の別のスレッドも IBM MQ に関連付けられている可能性がある場合は、アプリケーション・スレッドでこれらの信号を処理する信号ハンドラーを設定しないでください。

注 : スレッドが IBM MQ に接続している間にアプリケーションやミドルウェア・ライブラリー (アプリケーションの一部として稼働する) が信号ハンドラーを確立する場合、そのアプリケーションの信号ハンドラーは、信号の処理中に対応する IBM MQ ハンドラーを呼び出す必要があります。

信号ハンドラーの確立や復元では、最後に保管された信号ハンドラーから順番に復元するのが原則です。

- アプリケーションが、IBM MQ に接続した後で信号ハンドラーを確立した場合は、以前の信号ハンドラーを復元してから、IBM MQ へのアプリケーションの接続を切断してください。
- アプリケーションが、信号ハンドラーを確立してから IBM MQ に接続した場合は、IBM MQ へのアプリケーションの接続を切断してから、信号ハンドラーを復元してください。

注 : この、最後に保管された信号ハンドラーを最初に復元するという原則に従わなかった場合、アプリケーションでの信号処理に予期せぬ結果を招く可能性があり、アプリケーションによって信号が失われる恐れがあります。

非同期信号

IBM MQ では、クライアント・アプリケーションである場合を除いて、スレッド化されたアプリケーションで何らかの非同期信号が使用されることはありません。

スレッド化されたクライアント・アプリケーションに関する追加の考慮事項

IBM MQ は、サーバーへの入出力時に、以下の信号の処理を行います。 これらの信号は通信スタックで定義されています。 スレッドがキュー・マネージャーに接続されている間は、アプリケーションでこれらの信号に信号ハンドラーを確立してはなりません。

SIGPIPE (TCP/IP 用)

Linux AIX *MQI で AIX and Linux 信号処理を使用する際の追加の考慮事項*

AIX and Linux で信号処理に MQI を使用する場合、ファスト・パス・アプリケーション、信号ハンドラー内での MQI 機能呼び出し、MQI 呼び出し中の信号、ユーザー出口とインストール可能サービス、および VMS 出口ハンドラーに関する追加の考慮事項が存在します。

ファースト・パス (トラステッド) アプリケーション

ファースト・パス・アプリケーションは、IBM MQ が稼働しているプロセスと同じプロセスで稼働しています。つまり、ファースト・パス・アプリケーションは、マルチスレッド環境で稼働しています。

この環境では、IBM MQ は、同期信号 SIGSEGV、SIGBUS、SIGFPE、および SIGILL を処理します。 他のすべての信号は、ファースト・パス・アプリケーションが IBM MQ に接続されている間、ファースト・パス・アプリケーションに送達されることがあってはなりません。 これらの信号は、アプリケーションによってブロックまたは処理される必要があります。 ファースト・パス・アプリケーションがこのようなイベントを代行受信した場合は、必ずキュー・マネージャーを停止してから再起動してください。この処理をしない場合、キュー・マネージャーは未定義状態になる可能性があります。 MQCONNX のもとでファースト・パス・アプリケーションを実行する際の制約事項については、738 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』に詳しく記載されています。

信号ハンドラー内での MQI 機能呼び出し

信号ハンドラー内で MQI 機能を呼び出さないでください。

他の MQI 機能がアクティブになっているときに、信号ハンドラーから MQI 機能を呼び出そうとすると、MQRC_CALL_IN_PROGRESS が戻されます。 他にアクティブになっている MQI 機能がないときに信号ハンドラーから MQI 機能を呼び出そうとすると、選ばれた呼び出しのみをハンドラーから、またはハンドラー内で発行できるというオペレーティング・システムの制約事項のために、操作中のある時点でこの呼び出しが失敗する可能性があります。

C++ デストラクター方式では、プログラムの終了時に自動的に呼び出されることがあるため、MQI 機能の呼び出しを停止できない場合があります。 MQRC_CALL_IN_PROGRESS に関するエラーはすべて無視してください。 信号ハンドラーが exit() を呼び出すと、IBM MQ により同期点にあるコミットされていないメッセージは通常どおりバックアウトされ、オープンしているキューはすべてクローズされます。

MQI 呼び出し中の信号

MQI 機能は、コード EINTR およびそれに相当するコードをアプリケーション・プログラムに戻しません。

MQI の呼び出し中に信号が発生し、ハンドラーが *return* を呼び出した場合には、その信号が発生していない場合と同じように、呼び出しが続行されます。 特に、信号によって MQGET に割り込みをかけて、制御を即時にアプリケーションに戻すことはできません。 MQGET から抜け出したい場合は、キューを GET_DISABLED に設定します。あるいは、満了時刻を限定して (MQGMO_WAIT に gmo.WaitInterval を設定する)、MQGET 呼び出しをループに入れます。さらに、スレッド化されていない環境では信号ハンドラーを使用し、スレッド化された環境では信号ハンドラーに対応する機能を使用して、ループを中断するフラグを設定します。

AIX AIX 環境の IBM MQ では、信号によって割り込まれたシステム呼び出しは再始動する必要があります。 sigaction(2) を使用して独自の信号ハンドラーを確立する際には、新しいアクション構造体の sa_flags フィールドに SA_RESTART フラグを設定してください。 そうしない場合、IBM MQ が信号の割り込みを受けた任意の呼び出しを完了できなくなる場合があります。

ユーザー出口とインストール可能サービス

マルチスレッド環境で IBM MQ プロセスの一部として稼働しているユーザー出口とインストール可能サービスには、ファースト・パス・アプリケーションの場合と同じ制約事項があります。 これらの機能は永続的に IBM MQ に関連付けられると考えられるため、これらの機能で信号や非スレッド・セーフのオペレーティング・システム呼び出しを使用しないでください。

キュー・マネージャーへの接続とキュー・マネージャーからの切断

IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

この接続を行う方法は、プログラムが実行されるプラットフォームと環境に依存します。

Multi **IBM MQ for Multiplatforms**
これらの環境で実行されるプログラムは、キュー・マネージャーとの接続には MQCONN MQI 呼び出しを、切断には MQDISC 呼び出しをそれぞれ使用しなければなりません。 その代わりの方法として、プログラムは MQCONNX 呼び出しを使用することができます。

z/OS **IBM MQ for z/OS バッチ**
この環境で実行されるプログラムは、キュー・マネージャーとの接続には MQCONN MQI 呼び出しを、切断には MQDISC 呼び出しをそれぞれ使用しなければなりません。 その代わりの方法として、プログラムは MQCONNX 呼び出しを使用することができます。

z/OS バッチ・プログラムは、同一の TCB 上にある複数のキュー・マネージャーへの連続接続や同時接続を行えます。

z/OS **IMS**
IMS 制御領域が開始するときは、1 つ以上のキュー・マネージャーに接続します。 この接続は、IMS コマンドによって制御されます。 z/OS 上の IMS アダプターを制御する方法については、 IBM MQ for z/OS の管理を参照してください。 ただし、メッセージ・キューイング IMS プログラムの作成者は、

MQCONN MQI 呼び出しを用いて接続を希望する相手のキュー・マネージャーを指定する必要があります。 キュー・マネージャーから切断するには、MQDISC 呼び出しを用いることができます。

同期点を確立する IMS 呼び出し後、 別のユーザーのメッセージを処理する前に、IMS アダプターは、アプリケーションがハンドルをクローズしてキュー・マネージャーから切断したことを確認します。 860 ページの『IMS アプリケーションにおける同期点』を参照してください。

IMS バッチ・プログラムは、同一の TCB 上にある複数のキュー・マネージャーへの連続接続や同時接続を行えます。

z/OS 用の CICS Transaction Server

CICS プログラムは、CICS システム自体が接続しているので、キュー・マネージャーに接続するための作業を行う必要がありません。 通常、この接続は初期化時に自動的に設定されますが、IBM MQ for z/OS で提供されている CKQC トランザクションを使用することもできます。 CKQC について詳しくは、IBM MQ for z/OS の管理 を参照してください。

CICS タスクは、CICS 領域が接続されているキュー・マネージャーにのみ接続できます。

CICS プログラムは、MQI の接続呼び出しと切断呼び出し (MQCONN および MQDISC) を使用することもできます。 これらのアプリケーションを CICS 以外の環境に最小限の記録作業で移植できるようにするために、この方法をとることもできます。 ただし、これらの呼び出しは、 CICS 環境では 常に 正常に完了します。 つまり、戻りコードによる情報と、キュー・マネージャーとの実際の接続状態が異なる場合もあります。

TXSeries for Windows とオープン・システム

これらのプログラムは、CICS システム自体が接続しているので、キュー・マネージャーに接続するための作業を行う必要がありません。 したがって、一度に 1 つの接続だけがサポートされます。 CICS アプリケーションでは、接続ハンドルを取得するために MQCONN 呼び出しを発行しなければなりません。また、終了する前には MQDISC 呼び出しを発行しなければなりません。

キュー・マネージャーへの接続とキュー・マネージャーからの切断について詳しくは、以下のリンクを参照してください。

- 737 ページの『MQCONN 呼び出しを使用したキュー・マネージャーへの接続』
- 738 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』
- 742 ページの『MQDISC を使用したキュー・マネージャーからのプログラムの切断』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』

IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQCONN 呼び出しを使用したキュー・マネージャーへの接続

この情報を使用して、MQCONN 呼び出しを使用したキュー・マネージャーへの接続方法について学習します。

一般に、特定のキュー・マネージャー、またはデフォルトのキュー・マネージャーのどちらかへ接続できます。

- バッチ環境の IBM MQ for z/OS の場合、デフォルトのキュー・マネージャーは CSQBDEFV モジュールで指定されます。
- IBM MQ for Multiplatforms の場合、デフォルトのキュー・マネージャーは mqs.ini ファイルで指定されます。

別の方法として、z/OS MVS バッチ、TSO、および RRS 環境で、キュー共用グループ内の任意のキュー・マネージャーに接続できます。 MQCONN または MQCONNX 要求が、グループのアクティブ・メンバーのいずれか 1 つを選択します。

キュー・マネージャーに接続する場合、キュー・マネージャーはタスクに対してローカルでなければなりません。 キュー・マネージャーは IBM MQ アプリケーションと同じシステムに属する必要があります。

IMS 環境では、キュー・マネージャーを IMS 制御領域と、プログラムが使用する従属領域に接続しなければなりません。 デフォルト・キュー・マネージャーは、IBM MQ for z/OS のインストール時に CSQQDEFV モジュールで指定されます。

TXSeries CICS 環境と、TXSeries for Windows および AIX では、キュー・マネージャーを CICS に対する XA リソースとして定義する必要があります。

デフォルトのキュー・マネージャーに接続するには、すべてブランク文字からなる、またはヌル文字 (X'00') で始まる名前を指定して、MQCONN を呼び出します。

アプリケーションがキュー・マネージャーに正常に接続するには、アプリケーションが接続の許可を得ている必要があります。 詳しくは、保護を参照してください。

MQCONN の出力は以下のとおりです。

- 接続ハンドル (**Hconn**)
- 完了コード
- 理由コード

後続の MQI 呼び出しで接続ハンドルを使用してください。

アプリケーションが既にそのキュー・マネージャーに接続されていることを理由コードが示している場合は、アプリケーションが最初に接続されたときに戻されたものと同じ接続ハンドルが戻されます。 この状態ではアプリケーションは MQDISC 呼び出しを出す必要はありません。呼び出し側アプリケーションが接続の継続を期待しているためです。

接続ハンドルの有効範囲は、オブジェクト・ハンドルの有効範囲と同じです (744 ページの『MQOPEN 呼び出しを使用したオブジェクト・オープン』を参照)。

パラメーターの説明は、MQCONN の MQCONN 呼び出しに関する説明の中に含まれています。

呼び出しを出したときにキュー・マネージャーが静止状態の場合でも、あるいはキュー・マネージャーが終了状態の場合でも、MQCONN 呼び出しは失敗します。

MQCONN または MQCONNX の有効範囲

通常、MQCONN または MQCONNX 呼び出しの有効範囲は、発行元のスレッドです。 つまり、呼び出しから戻される接続ハンドルは、呼び出しを発行したスレッド内でのみ有効です。 そのハンドルを使用して一度に 1 つの呼び出しだけを実行できます。 別のスレッドがそのハンドルを使用している場合は、そのハンドルが無効なハンドルとして拒否されます。 アプリケーションに複数のスレッドがあって、それぞれが

IBM MQ 呼び出しを使用する場合には、それぞれのスレッドが MQCONN または MQCONNX を発行する必要があります。

1 つのプロセスで複数の MQCONN 呼び出しを発行する場合、各呼び出しを同一のキュー・マネージャーに対して発行する必要はありません。 ただし、一度に 1 つのスレッドから 1 つの IBM MQ 接続のみ行えます。 あるいは、単一スレッドからの複数の IBM MQ 接続と、任意のスレッドからの IBM MQ 接続を使用できるようにする 739 ページの『MQCONNX による共用 (スレッド独立) 接続』 を検討してください。[7]

アプリケーションをクライアントとして実行する場合は、1 つのスレッド内の複数のキュー・マネージャーに接続できます。

MQCONNX 呼び出しを使用したキュー・マネージャーへの接続

MQCONNX 呼び出しは、MQCONN 呼び出しに似ていますが、呼び出しの作業方法を制御するオプションを持っています。

MQCONNX に入力する場合は、キュー・マネージャー名 z/OS 、または z/OS 共用キュー・システムでのキュー共用グループ名を提供できます。 キュー・マネージャーへの接続方法を制御するオプションは、MQCNO と呼ばれる構造体で提供されます。

MQCONNX の出力は以下のとおりです。

- 接続ハンドル (Hconn)
- 完了コード
- 理由コード

後続の MQI 呼び出しで接続ハンドルを使用します。

MQCNO 構造体の *Options* フィールドに設定されている接続オプションを使用すると、接続のいくつかの属性を制御できます。 特に注意すべき点として、以下のオプションのグループがあります。

- バインディング・オプションを使用すると、 トラステッド・アプリケーション を作成できます。 トラステッド・アプリケーションは、 IBM MQ アプリケーションとローカル・キュー・マネージャー・エージェントが同じプロセスになることを意味します。 エージェント・プロセスではキュー・マネージャーにアクセスするためにインターフェースを使用する必要がなくなったので、これらのアプリケーションはキュー・マネージャーの拡張機能になります。 この動作は、MQCNO_FASTPATH_BINDING オプションを指定することによって要求されます。 トラステッド・アプリケーションに適用される制約事項について詳しくは、 738 ページの『トラステッド・アプリケーションの制約事項』 を参照してください。
- ハンドル共有オプションを使用すると、共有接続を作成できます。 共有接続は、同じプロセス内の異なるスレッド間でハンドルを共有できます。 共有接続について詳しくは、 739 ページの『MQCONNX による共用 (スレッド独立) 接続』 を参照してください。

MQCNO によって、アプリケーションはキュー・マネージャーへの接続の認証方法を制御することもできます。 認証資格情報は、MQCNO 構造体から参照される MQCSP 構造体で指定できます。

MQCONNX 呼び出しのパラメーター、および制御可能な接続属性について詳しくは、 MQCONNX-キュー・マネージャーの接続 (拡張)を参照してください。

トラステッド・アプリケーションの制約事項

トラステッド・アプリケーションに適用される制約事項。 すべてのプラットフォームに適用される制約事項もあれば、プラットフォーム固有の制約事項もあります。

T

- トラステッド・アプリケーションはキュー・マネージャーから明示的に切断する必要がある。
- **endmqm** コマンドを使用してキュー・マネージャーを終了する前に、トラステッド・アプリケーションを停止する必要があります。

[7] IBM MQ for AIX or Linux システムでマルチスレッド・アプリケーションを使用する場合は、アプリケーションがスレッドに十分なスタック・サイズを持っていることを確認する必要があります。 マルチスレッド・アプリケーションが MQI 呼び出しを作成する場合は、単独で作成する場合でも、他のシグナル・ハンドラー (例えば、CICS) を使用する場合でも、256 KB 以上のスタック・サイズを使用することを検討してください

- MQCNO_FASTPATH_BINDING と共に非同期信号とタイマー割り込み (sigkill など) を使用してはならない。
- すべてのプラットフォームにおいて、同じプロセス内の別のスレッドが別のキュー・マネージャーに接続されているときには、トラステッド・アプリケーション内のスレッドをキュー・マネージャーに接続できない。
- Linux AIX AIX and Linux システムでは、すべての MQI 呼び出しに有効なユーザー ID およびグループ ID として、mqm を使用する必要がある。 これらの ID は、認証が必要な非 MQI 呼び出し (例えば、ファイルのオープン) を行う前に変更できますが、次の MQI 呼び出しを行う前に mqm に戻しておく必要があります。
- IBM i On IBM i:
  1. トラステッド・アプリケーションは、QMQM ユーザー・プロファイルの下で実行する必要がある。 ユーザー・プロファイルが QMQM グループのメンバーであったり、プログラムが QMQM 権限を取得したりするだけでは十分ではない。 対話式ジョブにサインオンするために QMQM ユーザー・プロファイルを使用したり、トラステッド・アプリケーションを実行しているジョブのジョブ記述に QMQM ユーザー・プロファイルを指定したりできないことがある。 この場合の 1 つの方法は、IBM MQ プログラムの実行中に、 IBM i プロファイル・スワッピング API 関数 QSYGETPH、QWTSETP、および QSYRLSPH を使用して、ジョブの現行ユーザーを一時的に QMQM に変更することです。 これらの関数の詳細とその使用例は、 IBM i アプリケーション・プログラミング・インターフェース 資料の「セキュリティー API」セクションに記載されています。
  2. トラステッド・アプリケーションは、System-Request Option 2 で取り消したり、ENDJOB を使ってそれらが実行しているジョブを終了して取り消してはいけない。
- ALW AIX, Linux, and Windows システムでは、トラステッド 32 ビット・アプリケーションはサポートされていない。 トラステッド 32 ビット・アプリケーション を実行しようとすると、標準バウンド接続にダウングレードされる。

MQCONNX による共用 *(スレッド独立)* 接続

この情報を使用して、 MQCONNX との共有接続といくつかの使用上の注意について知ることができます。

注 : IBM MQ for z/OS ではサポートされない。

IBM MQ for z/OS 以外の IBM MQ プラットフォームでは、MQCONN で作成された接続は、その接続を作成したスレッドでのみ使用可能です。 MQCONNX 呼び出しでは、オプションを使用することで、プロセス内のすべてのスレッドで共用できる接続を作成することができます。 MQI 呼び出しを同じスレッド上で発行しなければならないトランザクション環境でアプリケーションが実行されている場合、以下のデフォルト・オプションを使用する必要があります。

MQCNO_HANDLE_SHARE_NONE
非共用接続を作成します。

その他のほとんどの環境では、スレッドに依存しない以下の共用接続オプションの 1 つを使用できます。

MQCNO_HANDLE_SHARE_BLOCK
共用接続を作成します。 MQCNO_HANDLE_SHARE_BLOCK 接続では、 現在別のスレッド上の MQI 呼び出しが接続を使用している場合は、その現行の MQI 呼び出しが完了するまで MQI 呼び出しを待機させます。

MQCNO_HANDLE_SHARE_NO_BLOCK
共用接続を作成します。 MQCNO_HANDLE_SHARE_NO_BLOCK 接続では、 現在別のスレッド上の MQI 呼び出しが接続を使用している場合は、MQI 呼び出しは MQRC_CALL_IN_PROGRESS の理由で即時に失敗します。

MTS (Microsoft Transaction Server) 環境では、MQCNO_HANDLE_SHARE_NONE がデフォルト値です。 MTS 環境では、MQCNO_HANDLE_SHARE_BLOCK がデフォルト値です。

接続ハンドルは MQCONNX 呼び出しから戻されます。 MQCONNX から戻されたハンドルは、各呼び出しに関連付けることによって、これより後に行われる、プロセス内のすべてのスレッドからの MQI 呼び出しに使用できます。 1 つの共用ハンドルを使用する各 MQI 呼び出しは、スレッドの枠を超えて逐次化されます。

例えば、共用ハンドルでは、次のような一連のアクティビティーが可能です。

1. スレッド 1 が MQCONNX を発行し、共用ハンドル *h1* を取得する
2. スレッド 1 がキューをオープンし、*h1* を使用して読み取り (get) 要求を発行する
3. スレッド 2 が *h1* を使用して書き込み (put) 要求を発行する
4. スレッド 3 が *h1* を使用して書き込み (put) 要求を発行する
5. スレッド 2 が *h1* を使用して MQDISC を発行する

ハンドルがいずれかのスレッドによって使用されている間、 他のスレッドでは、その接続にアクセスできなくなります。 先に発行された他のスレッドからの呼び出しが完了するまで次のスレッドを待機させておくことが可能な場面では、 MQCONNX にオプション MQCNO_HANDLE_SHARE_BLOCK を使用してください。

ただし、ブロックすると問題が生じる場合があります。 例えば、ステップ 740 ページの『2』 で、スレッド 1 が読み取り (get) 要求を発行し、これが未到着の可能性のあるメッセージを待機しているとします (待機状態の get 要求)。 この場合、スレッド 2 とスレッド 3 も、スレッド 1 でこの get 要求が完了するまで待機状態のまま (ブロックされた状態) になります。 そのハンドルで別の MQI 呼び出しが既に実行されている場合に MQI 呼び出しがエラーで戻るようにするには、 MQCONNX にオプション MQCNO_HANDLE_SHARE_NO_BLOCK を使用してください。

共用接続の使用上の注意

1. オブジェクトをオープンしたときに作成されるオブジェクト・ハンドル (Hobj) はすべて、Hconn と関連付けられます。それで、Hconn が共用される場合は、Hobj も共用されることになり、Hconn を使用するすべてのスレッドで使用できるようになります。 同様に、Hconn 下で開始されるすべての作業単位も Hconn に関連付けられます。それで、作業単位も、Hconn が共用されればスレッドの枠を超えて共用されることになります。
2. 共用 Hconn を切断するための MQDISC は、対応する MQCONNX を呼び出したスレッドに限らず、すべての スレッドから呼び出せます。 MQDISC は Hconn を終了させ、Hconn はどのスレッドからも使用できなくなります。
3. 単一スレッドで複数の共用 Hconn を並行して使用することもできます。例えば、MQPUT を使用して、ある共用 Hconn の下に 1 つのメッセージを put して、それから別の共用 Hconn を使用して別のメッセージを put することを、それぞれの操作を別々のローカル作業単位に属させたまま行うことができます。
4. 共用 Hconns は、グローバル作業単位では使用できません。

Multi *MQ_CONNECT_TYPE* を指定した *MQCONNX* 呼び出しオプションの使用

ここでは、さまざまな MQCONNX 呼び出しオプションと、それらが **MQ_CONNECT_TYPE** 環境変数でどのように使用されるかについて説明します。

注 : MQ_CONNECT_TYPE は、STANDARD バインディングに対してのみ有効です。 その他のバインディングの場合、 **MQ_CONNECT_TYPE** は無視されます。

IBM MQ for Multiplatforms では、環境変数 **MQ_CONNECT_TYPE** を、MQCONNX 呼び出しで使用される MQCNO 構造の Options フィールドに指定されたバインディングのタイプと組み合わせて使用できます。

表 *113.* ***MQ_CONNECT_TYPE*** 環境変数での *MQCONNX* 呼び出しオプションの使用方法

MQCONNX 呼び出しオプション	**MQ_CONNECT_TYPE 環境変数**	**結果**
STANDARD	UNDEFINED	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	ローカル	STANDARD

MQCNO_STANDARD_BINDING が指定されていない場合、MQCNO_NONE を使用でき、この場合、MQCNO_STANDARD_BINDING がデフォルトになります。

MQCONN および *MQCONNX* の認証と *ID*

このタスクを使用して、アプリケーションが IBM MQ への接続時に認証に使用される資格情報を提供する方法を学習します。

デフォルトのユーザー ID

アプリケーションがメッセージ・キュー・インターフェース (MQI) を使用して MQCONN または MQCONNX のいずれかで IBM MQ に接続する場合、ユーザー ID は常に確立され、接続に関連付けられます。

デフォルトでは、初期ユーザー ID は常に、アプリケーションが実行されているオペレーティング・システム・プロセスのユーザー ID です。 この初期 ID は、ローカルでバインドされたアプリケーション接続またはトラステッド・アプリケーション接続には十分である可能性があります。

アプリケーションが MQCONN 呼び出しを使用してキュー・マネージャーに接続すると、アプリケーションはデフォルト・ユーザー ID を変更できません。 ただし、以下のメカニズムによって、接続に関連付けられているユーザー ID を変更することができます。

- クライアント・サイドまたはサーバー・サイドのセキュリティー出口。
- キュー・マネージャーでのチャネル認証規則。
- TLS 相互認証中に確立されたクライアント・ユーザー ID。

MQCONNX を使用した資格情報の提供

MQCONNX により、アプリケーションは接続に関連付けられた ID をより詳細に制御できます。 アプリケーションは、MQCONNX に対する **ConnectOpts** パラメーターで指定された接続オプションの一部として、MQCSP 構造を提供できます。 MQCSP 構造体には、ユーザー ID を確立するために使用される資格情報を含めることができます。 IBM MQ は、MQCSP 構造で以下の資格情報をサポートします。

- ユーザー ID とパスワード。
- V 9.3.4 IBM MQ 9.3.4 以降、 AIX または Linux システム上で稼働するキュー・マネージャーにアプリケーションが接続する場合の認証トークン。

キュー・マネージャーの接続認証およびチャネル認証構成は、アプリケーションによって提供される資格情報の処理方法を制御します。 例えば、この構成は以下の側面に影響します。

- MQCSP 構造体内の資格情報が妥当性検査されるかどうか、および妥当性検査される方法。
- MQCSP 構造内の資格情報のユーザー ID が別のユーザー ID にマップされているかどうか。
- 認証されたユーザーがアプリケーションのコンテキストとして採用されるかどうか。

接続認証について詳しくは、 接続認証を参照してください。 チャネル認証について詳しくは、チャネル認証レコードを参照してください。

MQI を使用する C で作成されたいくつかのサンプル・プログラムは、MQCSP 構造を使用して認証資格情報を提供する方法を示しています。 詳しくは、以下のサンプル・プログラムを参照してください。

- 1095 ページの『読み取りサンプル・プログラム』
- 1107 ページの『書き込みサンプル・プログラム』
- 1083 ページの『ブラウザー・サンプル・プログラム』
- 1124 ページの『TLS サンプル・プログラム』

関連情報

MQCSP 構造を使用したユーザーの識別および認証
MQCSP - セキュリティー・パラメーター
ユーザーの識別および認証

MQDISC を使用したキュー・マネージャーからのプログラムの切断

この情報を使用して、MQDISC を使用したキュー・マネージャーからのプログラムの切断について学習します。

MQCONN または MQCONNX 呼び出しを用いてキュー・マネージャーに接続したプログラムが、すべてのキュー・マネージャーとの対話を終了したら、MQDISC 呼び出しを用いて接続を終了します。ただし、以下の場合を除きます。

- CICS Transaction Server for z/OS アプリケーションでは、MQCONNX が使用されていてアプリケーションの終了前に接続タグを除去する必要がない限り、呼び出しはオプションです。
- IBM MQ for IBM i では、オペレーティング・システムからサインオフする際に、暗黙の MQDISC 呼び出しが行われます。

MQDISC 呼び出しへの入力として、キュー・マネージャーに接続したときに MQCONN または MQCONNX によって戻された接続ハンドル (Hconn) を提供する必要があります。

z/OS 上の CICS を除き、MQDISC が呼び出された後は、接続ハンドル (Hconn) は無効になり、MQCONN または MQCONNX を再度呼び出すまで、これ以上 MQI 呼び出しを発行することはできません。 MQDISC は、依然オープンされ、このハンドルを用いるオブジェクトに対して暗黙的な MQCLOSE を行います。

z/OS に接続しているクライアントの場合、MQDISC 呼び出しが発行されると暗黙のコミットが行われますが、まだ開かれているキュー・ハンドルがクローズされることはありません。それらはチャネルが実際に終了したときにクローズされます。

IBM MQ for z/OS での接続に MQCONNX を使用した場合、MQDISC も MQCONNX が確立した接続タグの有効範囲を終了します。 しかし、CICS、IMS、または RRS アプリケーションでは、接続タグと関連したアクティブなリカバリー単位があれば、MQDISC は理由コード MQRC_CONN_TAG_NOT_RELEASED で拒否されます。

パラメーターの説明は、MQDISC の MQDISC 呼び出しに関する説明の中に含まれています。

MQDISC が発行されない場合

標準の非共用接続 (Hconn) は、作成スレッドが終了されるとクリーンアップされます。 しかし、共用接続は、プロセス全体が終了されるときにのみ、暗黙的にバックアウトされ、切断されます。 Hconn がまだ存在しているときに、その共用 Hconn を作成したスレッドが終了させられても、Hconn は引き続き使用可能です。

権限検査

MQCLOSE および MQDISC 呼び出しは、通常、権限検査は行いません。

正常なイベントの進行では、IBM MQ オブジェクトのオープンまたは接続の権限を持っているジョブが、そのオブジェクトをクローズしたり切断したりします。 IBM MQ オブジェクトへの接続またはオープンを行ったジョブの権限が取り消された場合でも、MQCLOSE および MQDISC 呼び出しは受け付けられます。

オブジェクトのオープンとクローズ

ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

以下のいずれかの操作を実行するには、まず、関連する IBM MQ オブジェクトを 開く 必要があります。

- メッセージをキューに書き込む
- メッセージをキューから読み取る (ブラウズまたは取り出し)
- オブジェクト属性を設定する
- 任意のオブジェクトの属性を照会する

オブジェクトをオープンするには、オブジェクトに対して行いたいことを指定するオプションを用いて、MQOPEN 呼び出しを使用します。 唯一の例外として、キューに単一メッセージを書き込みたいときは、キューを即時にクローズします。 この場合、MQPUT1 呼び出しを使用してオープン の段階を省略することができます (762 ページの『MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み』 を参照)。

MQOPEN 呼び出しを使用してオブジェクトをオープンするには、プログラムがキュー・マネージャーに接続されていなければなりません。 すべての環境に関する詳細については、735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』に説明があります。

オープンできる IBM MQ オブジェクトのタイプには、以下の 4 つがあります。

- キュー
- 名前リスト
- プロセス定義
- キュー・マネージャー

これらのオブジェクトは、MQOPEN 呼び出しを使用して、すべて同じ方法でオープンします。 IBM MQ オブジェクトについて詳しくは、「オブジェクト・タイプ」を参照してください。

同じオブジェクトを複数回オープンすることができますが、そのたびに新しいオブジェクト・ハンドルを取得します。 1 つのハンドルを用いてキュー上のメッセージをブラウズし、さらに別のハンドルを用いて同じキューからメッセージを除去したい場合があります。 これによって、同じオブジェクトをクローズして再オープンするためのリソースの消費を防ぎます。 また、メッセージのブラウズおよび 除去を同時に行うためにキューをオープンすることもできます。

さらに、単一の MQOPEN で複数のオブジェクトをオープンでき、MQCLOSE を使ってそれらをクローズできます。 この方法については、763 ページの『配布リスト』 を参照してください。

オブジェクトをオープンしようとするとき、MQOPEN 呼び出しで指定したオプションに対して、そのオブジェクトをオープンする権限を持っているかどうかをキュー・マネージャーが検査します。

プログラムがキュー・マネージャーから切断されるとき、オブジェクトは自動的にクローズされます。 IMS の環境では、プログラムが IMS の GU (get unique) 呼び出しの後、新しいユーザー用の処理を開始するとき、切断が強制的に実行されます。 IBM i プラットフォームでは、ジョブが終了したときに、オブジェクトは自動的にクローズされます。

プログラミングでは、オープンしたオブジェクトをクローズすることをお勧めします。 これを行うには、MQCLOSE 呼び出しを用います。

オブジェクトのオープンとクローズについて詳しくは、以下のリンクを参照してください。

- 744 ページの『MQOPEN 呼び出しを使用したオブジェクト・オープン』
- 751 ページの『動的キューの作成』
- 752 ページの『リモート・キューのオープン』
- 752 ページの『MQCLOSE 呼び出しを使用したオブジェクトのクローズ』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQOPEN 呼び出しを使用したオブジェクト・オープン

この情報を使用して、MQOPEN 呼び出しを使用したオブジェクトのオープンについて学習します。

MQOPEN 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。 z/OS 上の CICS アプリケーションの場合、定数 MQHC_DEF_HCONN (値ゼロ) を指定するか、MQCONN 呼び出しまたは MQCONNX 呼び出しによって戻される接続ハンドルを使用することができます。 他のプログラムの場合は、常に MQCONN 呼び出しか MQCONNX 呼び出しによって戻される接続ハンドルを用います。
- オープンしたいオブジェクトの記述。これはオブジェクト記述子構造体 (MQOD) を用いて行います。
- 呼び出しの処置を制御する 1 つまたは複数のオプション。

MQOPEN の出力は以下のとおりです。

- オブジェクトへのアクセスを表すオブジェクト・ハンドル。 後続の MQI 呼び出しへの入力にこれを使用します。
- 動的キューを作成する場合、修正されたオブジェクト記述子構造体 (ユーザーのプラットフォームでサポートされる)。
- 完了コード。
- 理由コード。

オブジェクト・ハンドルの有効範囲

オブジェクト・ハンドル (Hobj) の有効範囲は、接続ハンドル (Hconn) の有効範囲と同じです。

この点については、737 ページの『MQCONN または MQCONNX の有効範囲』と 739 ページの『MQCONNX による共用 (スレッド独立) 接続』を参照してください。 ただし、いくつかの環境では、追加の考慮事項があります。

CICS
CICS プログラムでは、MQOPEN 呼び出しを出したのと同じ CICS タスク内でのみ、オブジェクト・ハンドルを使用できます。

IMS および z/OS バッチ
IMS およびバッチ環境では、同一のタスク内ではハンドルを使用できますが、サブタスク内では使用できません。

MQOPEN 呼び出しのパラメーターの説明については、MQOPEN を参照してください。

以下の節では、MQOPEN の入力として指定しなければならない情報を取り上げます。

オブジェクトの識別 (MQOD 構造体)

MQOD 構造体を使用して、オープンしたいオブジェクトを識別します。 この構造体は MQOPEN 呼び出し用の入力パラメーターです。 (動的キューを作成するために MQOPEN 呼び出しを使用したときは、この構造体がキュー・マネージャーによって修正されます。)

MQOD 構造体の詳細については、MQOD を参照してください。

配布リストを生成するための MQOD 構造体の使用については、763 ページの『配布リスト』にある 764 ページの『MQOD 構造体の使用』を参照してください。

名前の解決
MQOPEN 呼び出しがキューとキュー・マネージャーの名前を解決する方法。

注 : キュー・マネージャーの別名は、RNAME フィールドを除いたリモート・キュー定義です。

IBM MQ キューをオープンするとき、MQOPEN 呼び出しによって、指定するキュー名におけるネーム・レゾリューション機能が実行されます。 これによって、キュー・マネージャーが以後の操作をどのキューに対して行うのかを決定します。 これは、オブジェクト記述子 (MQOD) に別名キューまたはリモート・キューの名前を指定したときは、呼び出しが名前をローカル・キューまたは伝送キューに解決することを意味します。 キューが任意のタイプの入力、ブラウズ、または設定用にオープンされている場合、ローカル・キューがあれば名前は解決されますが、ローカル・キューがなければ失敗に終わります。 キューが出力専用、照会専用、または出力照会両用にオープンされている場合のみ、非ローカル・キューに解決されます。
ネーム・レゾリューション処理の概要については、745 ページの表 114 を参照してください。
ObjectQMgrName に指定した名前が、*ObjectName* に指定した名前より前に 解決されます。

745 ページの表 114 は、キュー・マネージャー名の別名を定義するために、リモート・キューのローカル定義を使用する方法も示しています。 これにより、メッセージをリモート・キューに書き込むときに使用する伝送キューを選択できるので、例えば、多数のリモート・キュー・マネージャー向けのメッセージに対して単一の伝送キューを用いることができます。

次の表を使用するには、最初に左側の 2 つの列 (見出し **MQOD への入力**のもとにある) を読み、適切な事例を選択します。 次に、対応する行をすべて読み、指示に従います。 **解決済みの名前**列の指示に従い、**MQOD への入力**列に戻り、指示どおりに値を挿入するか、結果が提供された状態で表を終了することができます。
例えば、*ObjectName* を入力するように要求されたとします。

表 *114. MQOPEN* 使用時のキュー名の解決

MQOD への入力	**MQOD への入力**	**解決済みの名前**	**解決済みの名前**	**解決済みの名前**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
ブランクまたはローカル・キュー・マネージャー	CLUSTER 属性を含まないローカル・キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	該当しない (使用されたローカル・キュー)
ブランク・キュー・マネージャー	CLUSTER 属性を含むローカル・キュー	ワークロード管理で選択されているクラスター・キュー・マネージャー、または PUT で選択されている特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE および使用されたローカル・キュー SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ローカル・キュー・マネージャー	CLUSTER 属性を含むローカル・キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	該当しない (使用されたローカル・キュー)
ブランクまたはローカル・キュー・マネージャー	モデル・キュー	ローカル・キュー・マネージャー	生成された名前	該当しない (使用されたローカル・キュー)

表 *114. MQOPEN* 使用時のキュー名の解決 (続き)

MQOD への入力	**MQOD への入力**	**解決済みの名前**	**解決済みの名前**	**解決済みの名前**
ブランクまたはローカル・キュー・マネージャー	CLUSTER 属性が指定されている、または指定されていない別名キュー	別名キュー定義オブジェクト内の *ObjectQMgrName* を変更せず、*ObjectName* を *BaseQName* に設定して、ネーム・レゾリューションを再実行する。 *ObjectQMgrName* が指定されていてローカルに定義されている別名に解決してはならない。ただし、*ObjectQMgrName* がブランクであるクラスター別名 (他のキュー・マネージャーにホストされている) への解決は可能。		
ローカル・キュー・マネージャー	CLUSTER 属性を含む別名キュー	別名はローカルで定義されていないクラスター・キュー、または別名と同じ *ObjectName* を持つクラスター・キューに解決してはならない。		
ブランク・キュー・マネージャー	CLUSTER 属性を含む別名キュー	別名は、別名と同じ *ObjectName* を持つクラスター・キューに解決できる。		
ブランクまたはローカル・キュー・マネージャー	リモート・キューのローカル定義	*ObjectQMgrName* を *RemoteQMgrName* に、*ObjectName* を *RemoteQName* に設定して、ネーム・レゾリューションを再実行する。リモート・キューを解決してはならない。		非ブランクの場合は、*XmitQName* 属性の名前、ブランクの場合は、リモート・キュー定義オブジェクトの *RemoteQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ブランク・キュー・マネージャー	一致するローカル・オブジェクトなし。クラスター・キューは存在する。	ワークロード管理で選択されているクラスター・キュー・マネージャー、または PUT で選択されている特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ブランクまたはローカル・キュー・マネージャー	一致するローカル・オブジェクトなし。クラスター・キューは存在ない。		エラー。キューは存在ない。	適用外

表 *114. MQOPEN* 使用時のキュー名の解決 (続き)

MQOD への入力	**MQOD への入力**	**解決済みの名前**	**解決済みの名前**	**解決済みの名前**
ローカル・キュー・マネージャーと同じキュー共用グループ内のキュー・マネージャー名	ローカル共用キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	適用外
ローカル伝送キューの名前	(解決されない)	入力 *ObjectQMgrName*	入力 *ObjectName*	入力 *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
キュー・マネージャーの別名定義 (*RemoteQMgrName* がローカル・キュー・マネージャーの場合)	(解決されない。リモート・キュー)	*ObjectQMgrName* を *RemoteQMgrName* に設定して、ネーム・レゾリューションを再実行する。 リモート・キューに解決してはならない。	入力 *ObjectName*	非ブランクの場合は、*XmitQName* 属性の名前、ブランクの場合は、リモート・キュー定義オブジェクトの *RemoteQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
キュー・マネージャーがローカル・オブジェクトの名前でない。クラスター・キュー・マネージャーまたはキュー・マネージャー別名は存在する。	(解決されない)	*ObjectQMgrName* または PUT で選択された特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
キュー・マネージャーがローカル・オブジェクトの名前でない。クラスター・オブジェクトは存在していない。	(解決されない)	入力 *ObjectQMgrName*	入力 *ObjectName*	キュー・マネージャーの *DefXmitQName* 属性。*DefXmitQName* がサポートされる。 SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)

注 :

1. *BaseQName* は、別名キューの定義にある基本キューの名前です。
2. *RemoteQName* は、リモート・キューのローカル定義にあるリモート・キューの名前です。
3. *RemoteQMgrName* は、リモート・キューのローカル定義によるリモート・キュー・マネージャーの名前。
4. *XmitQName* は、リモート・キューのローカル定義による伝送キューの名前。
5. キュー共用グループ (QSG) の一部である IBM MQ for z/OS キュー・マネージャーの使用時、745 ページの表 114 のローカル・キュー・マネージャー名の代わりにキュー共用グループの名前を使用できます。

   ローカル・キュー・マネージャーがターゲット・キューをオープンできない場合、またはキューにメッセージを書き込めない場合、そのメッセージは、グループ内キューイングまたは IBM MQ チャネルのどちらかの方法で、指定の ObjectQMgrName に転送されます。
6. テーブルの *ObjectName* 列で、CLUSTER はキューの CLUSTER 属性と CLUSNL 属性の両方を表します。
7. ローカルおよびリモート・キュー・マネージャーが同じキュー共用グループにある場合、SYSTEM.QSG.TRANSMIT.QUEUE が使用されます。グループ内キューイングは使用可能になります。
8. 各クラスター送信側チャネルに対して異なるクラスター伝送キューを割り当てた場合、SYSTEM.CLUSTER.TRANSMIT.QUEUE はクラスター伝送キューの名前と同じではない可能性がありま

す。 複数のクラスター伝送キューについて詳しくは、クラスター化: クラスター伝送キューの構成方法の計画を参照してください。

9. キュー・マネージャーがローカル・オブジェクトの名前でなく、クラスター・キュー・マネージャーまたはキュー・マネージャー別名は存在する場合。

   ObjectQMgrName を使用してキュー・マネージャー名を指定しており、その宛先に到達するクラスター・チャネルが複数個存在し、ローカル・キュー・マネージャーがそれらのチャネルをそれぞれ異なるクラス ター名で認識している場合、宛先キューのクラスター名に関係なく、これらのどのチャネルもメッセージの移動に使用される可能性があります。

   そのキューに対するメッセージが、そのキューと同じクラスター名の付いたチャネル経由でのみ送信されることを予期していた場合、これは予期しない状況となります。

   ただし、この場合は **ObjectQMgrName** が優先されます。また、そのキュー・マネージャーに到達する可能性があるすべてのチャネルが属するクラスター名に関係なく、クラスター・ワークロード・バランシングではそれらのチャネルがすべて考慮に入れられます。

別名キューをオープンすると、別名が解決する基本キューもオープンされ、リモート・キューをオープンすると、伝送キューもオープンされます。 したがって、指定したキューも、それが解決されるキューも、他方がオープンされている間は削除できません。

別名キューはローカルに定義された別の別名キュー (クラスターで共有されているもの、またはそうでないもの) に解決することはできませんが、リモートに定義されたクラスター別名キューに解決することは許可されているため、 基本キューとして指定できます。

解決したキュー名と解決したキュー・マネージャー名は、MQOD の *ResolvedQName* フィールドと *ResolvedQMgrName* フィールドにそれぞれ格納されます。

分散キューイング環境でのネーム・レゾリューションの詳細については、キュー名解決についてを参照してください。

MQOPEN 呼び出しのオプションの使用

MQOPEN 呼び出しの **Options** パラメーターでは、オープンするオブジェクトに対するアクセスを制御するため、1 つまたは複数のオプションを選択しなければなりません。 これらのオプションを指定すると、次のことを行うことができます。

- キューをオープンし、そのキューに書き込まれるすべてのメッセージがそのキューの同じインスタンスに送信されるように指定する
- メッセージを書き込むためにキューをオープンする
- メッセージをブラウズするためにキューをオープンする
- メッセージを除去できるようにキューをオープンする
- 属性を照会し設定するためにオブジェクトをオープンする (ただし、設定できるのはキューの属性のみ)
- トピックまたはトピック・ストリングをオープンして、そこへメッセージをパブリッシュする
- コンテキスト情報をメッセージに関連付ける
- セキュリティー検査に使用する代替ユーザー ID を指名する
- キュー・マネージャーが静止状態であれば、呼び出しを制御する

クラスター・キュー用の *MQOPEN* オプション

キュー・ハンドルに使用されるバインディングは **DefBind** キュー属性から取得され、値は MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP を取ることができます。

MQPUT を使用してキューに書き込まれるすべてのメッセージが同じ経路をたどって同じキュー・マネージャーに送付されるようにするには、MQOPEN 呼び出しで MQOO_BIND_ON_OPEN オプションを使用します。

MQPUT の実行時に宛先が選択されるように指定するには、MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED オプションを使用します。

MQPUT を使用してキューに書き込まれるメッセージ・グループのすべてのメッセージが、同一の宛先インスタンスに割り振られるように指定するには、MQOPEN 呼び出しで MQOO_BIND_ON_GROUP オプションを使用します。

グループ内のすべてのメッセージが同じ宛先で処理されるように、クラスターで メッセージ・グループ を使用する場合は、 MQOO_BIND_ON_OPEN または MQOO_BIND_ON_GROUP のいずれかを指定する必要があります。

これらのオプションのいずれも指定しない場合、デフォルトの MQOO_BIND_AS_Q_DEF が使用されます。

MQOD にキュー・マネージャーの名前を指定すると、そのキュー・マネージャーのキューが選択されます。キュー・マネージャーの名前をブランクにすると、任意のインスタンスを選択できます。 詳細については、889 ページの『MQOPEN およびクラスター』を参照してください。

QALIAS 定義を使用してクラスター・キューをオープンすると、いくつかのキュー属性は基本キューではなく別名キューによって定義されます。 クラスター属性は、別名キューによって指定変更される基本キュー定義の属性に含まれます。 例えば、以下のスニペットでは、クラスター・キューは、MQOO_BIND_ON_OPEN ではなく MQOO_BIND_NOT_FIXED でオープンされます。 クラスター・キュー定義はクラスターを通じて公示され、別名キュー定義はキュー・マネージャーに対してローカルです。

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

メッセージを書き込むための *MQOPEN* オプション
メッセージを書き込むためにキューまたはトピックを開くには、MQOO_OUTPUT オプションを使用します。

メッセージをブラウズするための *MQOPEN* オプション
キューをオープンしてメッセージをブラウズできるようにするには、MQOO_BROWSE オプションと一緒に MQOPEN 呼び出しを使用します。

これは、キュー・マネージャーがキューの次のメッセージを 識別するために用いるブラウズ・カーソル を作成します。 詳しくは、 803 ページの『キュー上のメッセージのブラウズ』を参照してください。

注 :

1. リモート・キュー上のメッセージをブラウズすることはできません。MQOO_BROWSE オプションを用いてリモート・キューをオープンしないでください。
2. このオプションは、配布リストをオープンするときには指定できません。 配布リストの詳細については、763 ページの『配布リスト』を参照してください。
3. 共同ブラウズを使用している場合は、MQOO_CO_OP を MQOO_BROWSE と共に使用します。詳細は、Options を参照してください。

メッセージを除去するための *MQOPEN* オプション
キューからメッセージを除去するには、キューのオープンを制御する 3 つのオプションがあります。

どの MQOPEN 呼び出しにもそれらのうち 1 つしか使用できません。 これらのオプションは、プログラムのキューへのアクセスが排他的か共用かを定義します。 排他的アクセス は、キューをクローズするまでのことを意味します。 メッセージを削除できるのは、ユーザーだけです。 別のプログラムが、メッセージを除去するためにキューをオープンしようとすると、その MQOPEN 呼び出しは失敗に終わります。 共有アクセス は、複数のプログラムが削除できることを意味します。 キューからのメッセージ。

最もお勧めできる方法は、キューが定義されたときに決められたアクセスのタイプを受け入れることです。**Shareability** の設定に関係するキュー定義および **DefInputOpenOption** 属性。 このアクセスを受け入れるには、MQOO_INPUT_AS_Q_DEF オプションを使用します。 このオプションを使用するとき、これらの属性の設定によって アクセスのタイプがどのような影響を受けるかについては、750 ページの表 115 を参照してください。

表 115. キュー属性と *MQOPEN* 呼び出しのオプションがキューへのアクセスを及ぼす影響				
キューの属性		**MQOPEN のオプションによるアクセスのタイプ**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	共有	共有	排他的
SHAREABLE	EXCLUSIVE	排他的	共有	排他的
NOT_SHAREABLE*	SHARED*	排他的	排他的	排他的
NOT_SHAREABLE	EXCLUSIVE	排他的	排他的	排他的

注 : * 属性をこのように組み合わせてキューを定義することはできますが、デフォルトの入力オープン・オプションは、Shareability 属性によって上書きされます。

代替方法

- 他のプログラムがメッセージをキューから同時に除去することがあっても、アプリケーションが正常に機能することが分かっている場合は、MQOO_INPUT_SHARED オプションを使用してください。 750 ページの表 115 は、たとえこのオプションを指定しても、キューに対するアクセスが排他的になる特定の場合があることを示します。
- 他のプログラムがメッセージをキューから同時に除去できない場合のみ、アプリケーションが正常に機能することが分かっている場合は、MQOO_INPUT_EXCLUSIVE オプションを使用してください。

注 :

1. リモート・キューからはメッセージを除去できません。 したがって、MQOO_INPUT_* オプションのどれを使用してもリモート・キューをオープンすることはできません。
2. このオプションは、配布リストをオープンするときには指定できません。 詳しくは、763 ページの『配布リスト』を参照してください。

属性を設定し照会するための *MQOPEN* オプション
キューを開いてその属性を設定できるようにするには、MQOO_SET オプションを使用します。

他のタイプのオブジェクトの属性は設定できません (854 ページの『オブジェクト属性の照会と設定』を参照)。

オブジェクトの属性を照会するためにオブジェクトをオープンするには、MQOO_INQUIRE オプションを使用します。

注 : このオプションは、配布リストをオープンするときには指定できません。

メッセージ・コンテキストに関連する *MQOPEN* オプション
メッセージをキューに書き込むときに、コンテキスト情報とメッセージの関連付けができるようにしたい場合は、キューをオープンするときにメッセージ・コンテキスト・オプションの 1 つを使用しなければなりません。

これらのオプションを使用すると、メッセージを発信したユーザー に関連するコンテキスト情報と、メッセージを発信したアプリケーション に関連するコンテキスト情報とを区別できます。 また、メッセージをキューに書き込むときにコンテキストを設定するか、あるいはコンテキストを他のキュー・ハンドルから自動的に持ってくるかを選択することもできます。

関連概念
47 ページの『メッセージ・コンテキスト』
メッセージ・コンテキスト 情報により、メッセージを受信するアプリケーションは、そのメッセージの発信元についての情報を得ることができます。

759 ページの『メッセージ・コンテキスト情報の制御』
メッセージをキューに書き込むために MQPUT または MQPUT1 呼び出しを使用するときは、キュー・マネージャーがメッセージ記述子に何らかのデフォルト・コンテキスト情報を追加するように指定できます。 適切な許可レベルを持つアプリケーションは、余分のコンテキスト情報を追加できます。 MQPMO 構造体のオプション・フィールドを使用して、コンテキスト情報を制御できます。

代替ユーザー権限のための *MQOPEN* オプション
MQOPEN 呼び出しを使用してオブジェクトをオープンしようとすると、キュー・マネージャーはそのオブジェクトをオープンする権限を持っているか検査します。 ユーザーに権限がない場合、呼び出しは失敗します。

しかし、サーバー・プログラムは、サーバー自体の許可ではなく、作業対象のユーザーの許可をキュー・マネージャーに検査させることもできます。 これを行うには、MQOPEN 呼び出しの MQOO_ALTERNATE_USER_AUTHORITY オプションを使用し、MQOD 構造体の *AlternateUserId* フィールドに代替ユーザー ID を指定しなければなりません。 サーバーは、通常、サーバー自身が処理中のメッセージ内のコンテキスト情報からそのユーザー ID を取得します。

z/OS キュー・マネージャーの静止に関する *MQOPEN* オプション
キュー・マネージャーが静止状態のときに MQOPEN 呼び出しを使用した場合、使用環境によっては呼び出しが失敗することがあります。

z/OS 上の CICS 環境では、キュー・マネージャーが静止状態のときに MQOPEN 呼び出しを使用すると、呼び出しは常に失敗します。

他の z/OS および Multiplatforms 環境では、キュー・マネージャーが静止状態のときに MQOPEN 呼び出しの MQOO_FAIL_IF_QUIESCING オプションを使用する場合にのみ、その呼び出しは失敗します。

ローカル・キュー名の解決の *MQOPEN* オプション
ローカル・キュー、別名キュー、またはモデル・キューをオープンすると、ローカル・キューが返されます。

ただし、リモート・キューまたはクラスター・キューをオープンすると、MQOD 構造体の *ResolvedQName* および *ResolvedQMgrName* フィールドには、リモート・キュー定義で検出されたリモート・キューおよびリモート・キュー・マネージャーの名前、または選択されたリモート・クラスター・キューの名前が入ります。

MQOPEN 呼び出しの MQOO_RESOLVE_LOCAL_Q オプションを使用して、MQOD 構造体の *ResolvedQName* に、オープンされたローカル・キューの名前を入れます。 同様に、*ResolvedQMgrName* には、ローカル・キューをホスティングするローカル・キュー・マネージャーの名前が入ります。 このフィールドは MQOD 構造体のバージョン 3 でのみ使用可能です。 構造体がバージョン 3 より前の場合、エラーが戻されずに MQOO_RESOLVE_LOCAL_Q が無視されます。

例えば、リモート・キューを開くときに MQOO_RESOLVE_LOCAL_Q を指定すると、 *ResolvedQName* はメッセージが書き込まれる伝送キューの名前になります。 *ResolvedQMgrName* は、伝送キューをホスティングするローカル・キュー・マネージャーの名前です。

動的キューの作成
アプリケーションの終了後にそのキューが必要ない場合には、動的キューを使用してください。

例えば、応答先キューに動的キューを使用できます。 メッセージをキューに書き込む場合、MQMD 構造体の *ReplyToQ* フィールドに応答先キューの名前を指定します (755 ページの『MQMD 構造体を使用するメッセージの定義』 を参照)。

動的キューを作成するには、モデル・キューと呼ばれるテンプレートを MQOPEN 呼び出しと共に使用します。 IBM MQ コマンドまたは操作および制御パネルを使用して、モデル・キューを作成します。 作成される動的キューは、モデル・キューの属性を取り入れます。

MQOPEN を呼び出すときに、MQOD 構造体の *ObjectName* フィールドにモデル・キューの名前を指定します。 その呼び出しが完了すると、*ObjectName* フィールドが作成された動的キューの名前に設定されます。 また、*ObjectQMgrName* フィールドがローカル・キュー・マネージャーの名前に設定されます。

次の 3 つの方法で、作成する動的キューの名前を指定できます。

- MQOD 構造体の *DynamicQName* フィールドに、キューに付けたい名前をフルネームで指定する。
- 名前の (33 文字より少ない) 接頭部を指定し、キュー・マネージャーに名前の残りの部分を生成させる。 これは、キュー・マネージャーが固有な名前を生成するが、プログラマーもある程度の制御を行うことができる (例えば、各ユーザーにある一定の接頭部を使用させるか、ある特定の接頭部の付いた名前を持つ

キューに特別のセキュリティー区分を与える) ことを意味します。 この方法を使用するには、*DynamicQName* フィールドの最後の非ブランク文字にアスタリスク (*) を指定します。 動的キュー名に単一のアスタリスク (*) を指定しないでください。

- キュー・マネージャーに、キューに付けたい名前を完全な形で生成させる。 この方法を使用するには、*DynamicQName* フィールドの先頭文字位置にアスタリスク (*) を指定します。

これらの方法について詳しくは、DynamicQName フィールドの説明を参照してください。

動的キューについての詳細は、「動的キューおよびモデル・キュー」にあります。

リモート・キューのオープン

リモート・キューは、アプリケーションが接続しているキュー・マネージャーとは別のキュー・マネージャーが所有するキューです。

リモート・キューをオープンするときは、ローカル・キューの場合と同じように MQOPEN 呼び出しを使用します。 次の方法でキューの名前を指定できます。

1. MQOD 構造体の *ObjectName* フィールドには、 ローカル キュー・マネージャーに認識されているリモート・キューの名前を指定します。

   注: この場合は、 *ObjectQMgrName* フィールドをブランクのままにします。

2. MQOD 構造体の *ObjectName* フィールドに、 リモート キュー・マネージャーに認識されているリモート・キューの名前を指定します。 *ObjectQMgrName* フィールドに、以下のいずれかを指定します。

   - リモート・キュー・マネージャーと同じ名前の伝送キューの名前。 名前とその大文字小文字 (大文字のみ、小文字のみ、大文字小文字混合) は、正確に 一致していなければならない。
   - 宛先となるキュー・マネージャーや伝送キューを解決する、キュー・マネージャー別名オブジェクトの名前。

   これによって、メッセージの宛先と、そこに到達するために書き込まれる必要がある伝送キューがキュー・マネージャーに通知されます。

3. *DefXmitQname* がサポートされている場合は、MQOD 構造体の *ObjectName* フィールドに、 リモート キュー・マネージャーが認識しているリモート・キューの名前を指定します。

   注: *ObjectQMgrName* フィールドをリモート・キュー・マネージャーの名前に設定します (この場合、ブランクのままにすることはできません)。

MQOPEN を呼び出した場合にはローカル名だけが妥当性検査されます。最後の検査では、使用される伝送キューの存在が検査されます。

これらのメソッドは、745 ページの表 114 に要約されています。

MQCLOSE 呼び出しを使用したオブジェクトのクローズ

オブジェクトをクローズするには、MQCLOSE 呼び出しを使用します。

オブジェクトがキューの場合は、以下の点に注意してください。

- 一時動的キューをクローズする前に、空にする必要はありません。

  一時動的キューをクローズすると、その中に残っているメッセージと共にそのキューも削除されます。これは、たとえそのキューに対してコミットされていない MQGET、MQPUT、または MQPUT1 呼び出しがあっても同様です。

- IBM MQ for z/OS では、MQGMO_SET_SIGNAL オプションを持つ MQGET 要求がそのキューに対して未解決の場合は、それらの要求は取り消されます。
- MQOO_BROWSE オプションを用いてキューをオープンした場合は、ブラウズ・カーソルが破壊されます。

クローズは同期点と無関係なので、同期点の前でも後でもキューをクローズできます。

MQCLOSE 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。 オープンに使用したものと同じ接続ハンドルを使用するか、または z/OS 上の CICS アプリケーションの場合は、定数 MQHC_DEF_HCONN (値はゼロ) を指定することができます。

- クローズしたいオブジェクトのハンドル。 これは MQOPEN 呼び出しの出力から得られます。
- *Options* フィールドに MQCO_NONE を指定する (永続動的キューをクローズする場合以外)。
- メッセージがまだある場合でも、キュー・マネージャーがキューを削除するかどうかを判別する制御オプション (永続動的キューをクローズする場合)。

MQCLOSE の出力は以下のとおりです。

- 完了コード
- 理由コード
- MQHO_UNUSABLE_HOBJ という値にリセットされるオブジェクト・ハンドル

MQCLOSE 呼び出しのパラメーターの説明については、MQCLOSE を参照してください。

キューへのメッセージの書き込み

この情報を使用して、メッセージをキューに書き込む方法について学習します。

キューにメッセージを書き込むには MQPUT 呼び出しを使用します。 同じキューにいくつものメッセージを書き込むときは、最初の MQOPEN 呼び出しに続けて MQPUT を反復使用できます。 すべてのメッセージをキューに書き込んだら、MQCLOSE を呼び出します。

キュー・メッセージを 1 つだけ書き込んで、その後すぐにキューをクローズしたい場合は、MQPUT1 呼び出しを使用できます。 MQPUT1 は、次の一連の呼び出しと同じ機能を実行します。

- MQOPEN
- MQPUT
- MQCLOSE

しかし一般的には、複数のメッセージをキューに書き込むには、MQPUT 呼び出しを使用する方が効果的です。 これは、メッセージのサイズと、作業を行っているプラットフォームに応じて異なります。

メッセージをキューに書き込む方法について詳しくは、以下のリンクを参照してください。

- 754 ページの『MQPUT 呼び出しを使用したローカル・キューへのメッセージの書き込み』
- 759 ページの『リモート・キューへのメッセージの書き込み』
- 759 ページの『メッセージ・プロパティーの設定』
- 759 ページの『メッセージ・コンテキスト情報の制御』
- 762 ページの『MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み』
- 763 ページの『配布リスト』
- 768 ページの『書き込み呼び出しが失敗する場合』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQPUT 呼び出しを使用したローカル・キューへのメッセージの書き込み

ここでは、MQPUT 呼び出しを使用してローカル・キューにメッセージを書き込むための情報を取り上げます。

MQPUT 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル (Hconn)。
- キュー・ハンドル (Hobj)。
- キューに書き込みたいメッセージについての記述。 これは、メッセージ記述子構造体 (MQMD) の形式になります。
- 書き込みメッセージ・オプション構造体の形式による制御情報 (MQPMO)。
- メッセージ内に含まれるデータの長さ (MQLONG)。
- メッセージ・データそのもの。

MQPUT 呼び出しの出力は、以下のとおりです。

- 理由コード (MQLONG)
- 完了コード (MQLONG)

呼び出しが正常に完了した場合は、オプション構造体とメッセージ記述子構造体も戻します。 この呼び出しは、メッセージ送信先のキューおよびキュー・マネージャーの名前を示すために、オプション構造体を変更します。 書き込むメッセージの ID に対して、キュー・マネージャーが固有な値を生成するよう要求する場合 (MQMD 構造体の *MsgId* フィールドに 2 進数のゼロを指定することによって)、呼び出しは、この構造体を戻す前に *MsgId* フィールドにその値を入れます。 この値は、次に MQPUT を発行するまでにリセットしてください。

MQPUT 呼び出しの説明については、MQPUT を参照してください。

MQPUT 呼び出しの入力として指定しなければならない情報の詳しい説明については、以下のリンクを参照してください。

- 754 ページの『ハンドルの指定』
- 755 ページの『MQMD 構造体を使用するメッセージの定義』
- 755 ページの『MQPMO 構造体を使用するオプションの指定』
- 758 ページの『メッセージ内のデータ』
- 759 ページの『メッセージの書き込み: メッセージ・ハンドルの使用』

ハンドルの指定

z/OS アプリケーション上の CICS の接続ハンドル (*Hconn*) には、定数 MQHC_DEF_HCONN (値ゼロ) を指定するか、MQCONN または MQCONNX 呼び出しによって戻される接続ハンドルを使用することができます。

その他のアプリケーションの場合は、常に MQCONN 呼び出しまたは MQCONNX 呼び出しから返される接続ハンドルを使用します。

どの作業環境でも、MQOPEN 呼び出しから返される同じキュー・ハンドル (*Hobj*) を使用してください。

MQMD 構造体を使用するメッセージの定義

メッセージ記述子 (MQMD) は、MQPUT および MQPUT1 呼び出しに対する入出力パラメーターです。 これを使用して、キューに書き込むメッセージを定義します。

メッセージに対して MQPRI_PRIORITY_AS_Q_DEF または MQPER_PERSISTENCE_AS_Q_DEF が指定されていて、さらにキューがクラスター・キューである場合は、MQPUT で解決されるキューの値が使用されます。 そのキューが MQPUT では使用できないキューの場合、呼び出しは失敗します。 詳細については、キュー・マネージャー・クラスターの構成を参照してください。

注 : *MsgId* と *CorrelId* が固有なものとなるようにするため、新しいメッセージを置く前に MQPMO_NEW_MSG_ID と MQPMO_NEW_CORREL_ID を使用してください。 これらのフィールドの値は、MQPUT の正常終了時に戻されます。

MQMD で記述するメッセージ・プロパティーの概要については、18 ページの『IBM MQ メッセージ』を参照してください。その構造体そのものの説明については、MQMD を参照してください。

MQPMO 構造体を使用するオプションの指定

MQPMO (書き込みメッセージ・オプション) 構造体を使用して、MQPUT および MQPUT1 呼び出しにオプションを渡します。

以下の節を参考にして、この構造体の各フィールドに情報を入力してください。 構造体の説明については、MQPMO を参照してください。

この構造体には、以下のフィールドがあります。

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

これらのフィールドの内容は次のとおりです。

StrucId
構造体を書き込みメッセージ・オプション構造体として識別します。 これは 4 文字フィールドです。 必ず MQPMO_STRUC_ID を指定します。

バージョン
構造体のバージョン番号を記述します。 デフォルトは MQPMO_VERSION_1 です。 MQPMO_VERSION_2 を入力すると、配布リストを使用することができます (763 ページの『配布リスト』 を参照)。 MQPMO_VERSION_3 を 入力すると、メッセージ・ハンドルおよびメッセージ・プロパティーを使用できます。 MQPMO_CURRENT_VERSION を入力すると、アプリケーションが必ず最新のレベルを使用するように設定されます。

オプション
次のことを制御します。

- 書き込み操作を作業単位に含めるかどうか
- メッセージに関連付けるコンテキスト情報の量
- どこからコンテキスト情報を取り出すか
- キュー・マネージャーが静止状態のとき、呼び出しをエラーにするかどうか
- グループ化またはセグメント化が可能かどうか
- 新規のメッセージ ID と相関 ID の生成
- メッセージとセグメントをキューに書き込む順序
- ローカル・キュー名を解決するかどうか

Options フィールドをデフォルト値 (MQPMO_NONE) のままにしておくと、書き込むメッセージのコンテキスト情報がそれに関連付けられているデフォルトの情報になります。

また、呼び出しと同期点の関係は、次のようにプラットフォームによって決まります。 z/OS での同期点制御のデフォルト値は yes ですが、それ以外のプラットフォームでは no です。

Context
コンテキスト情報のコピー元にしたいキュー・ハンドルの名前を示します (*Options* フィールドで要求した場合)。

メッセージ・コンテキストについては、47 ページの『メッセージ・コンテキスト』を参照してください。 メッセージのコンテキスト情報を制御する MQPMO 構造体の使用方法については、759 ページの『メッセージ・コンテキスト情報の制御』を参照してください。

ResolvedQName
メッセージを受け取るためにオープンされたキューの名前 (別名の解決後の名前) を示します。 これは出力フィールドです。

ResolvedQMgrName
ResolvedQName のキューを所有するキュー・マネージャーの名前 (別名の解決後の名前) を示します。 これは出力フィールドです。

MQPMO では、配布リストに必要なフィールドも提供できます (763 ページの『配布リスト』を参照してください)。 この機能を使用したい場合は、バージョン 2 の MQPMO 構造体を使用します。 この構造体のフィールドを次に示します。

RecsPresent
このフィールドには、配布リスト内のキューの数、つまり存在する書き込みメッセージ・レコード (MQPMR)、および対応する応答レコード (MQRR) の数が入ります。

入力する値は、MQOPEN で指定したオブジェクト・レコードの数と同じにすることができます。 ただし、値が MQOPEN 呼び出しで提供されたオブジェクト・レコードの数より小さい場合、または、書き込みメッセージ・レコードを提供しない場合は、定義されていないキューの値として、メッセージ記述子提供のデフォルト値が使われます。 また、値が、提供されたオブジェクト・レコードの数より大きい場合は、超過した書き込みメッセージ・レコードが無視されます。

次のいずれかの方法を推奨します。

- 各宛先から報告または応答を受け取りたい場合は、MQOR 構造体にある値と同じ値を入力し、*MsgId* フィールドを含む MQPMR を使用する。 これらの *MsgId* フィールドをゼロに初期化する、または MQPMO_NEW_MSG_ID を指定する。

  メッセージをキューに書き込むと、キュー・マネージャーによって作成された *MsgId* の値を MQPMR で使用できるようになる。これらを使って、各報告または応答に関連付けられている宛先を識別できる。
- 報告および応答を受け取りたくない場合は、次のいずれかの方法を行う。

  1. 即時に失敗する宛先を識別したい場合は、MQOR 構造体にある値と同じ値を *RecsPresent* フィールドに入力し、これらの宛先を識別するために MQRR を指定しても構わない。 MQPMR は指定しない。

2. 失敗する宛先を識別したくない場合は、*RecsPresent* フィールドにゼロを入力し、MQPMR も MQRR も指定しない。

注 : MQPUT1 を使用している場合は、応答レコード・ポインターと応答レコード・オフセットの数がゼロでなければなりません。

書き込みメッセージ・レコード (MQPMR) と応答レコード (MQRR) の詳細については、MQPMR および MQRR を参照してください。

PutMsgRecFields
各書き込みメッセージ・レコード (MQPMR) に存在するレコードを示します。 これらのフィールドのリストについては、767 ページの『MQPMR 構造体の使用』を参照してください。

PutMsgRecOffset* と *PutMsgRecPtr
ポインター (一般に C で使用される) とオフセット (一般に COBOL で使用される) は、書き込みメッセージ・レコードのアドレッシングを行うために使用されます (MQPMR 構造体の概要については、767 ページの『MQPMR 構造体の使用』を参照してください)。

PutMsgRecPtr フィールドを使って最初の書き込みメッセージ・レコードを指し示すポインターを指定するか、または *PutMsgRecOffset* フィールドを使って最初の書き込みメッセージ・レコードのオフセットを指定します。 これは、MQPMO の先頭からのオフセットです。 *PutMsgRecFields* フィールドに従って、*PutMsgRecOffset* または *PutMsgRecPtr* にヌル以外の値を入力します。

ResponseRecOffset および ResponseRecPtr
ポインターとオフセットは、応答レコードのアドレッシングにも使用します (応答レコードの詳細については、766 ページの『MQRR 構造体の使用』を参照してください)。

ResponseRecPtr フィールドを使って最初の応答レコードを指し示すポインターを指定するか、または *ResponseRecOffset* フィールドを使って最初の応答レコードのオフセットを指定します。 これは、MQPMO 構造体の先頭からのオフセットです。 *ResponseRecOffset* または *ResponseRecPtr* に、ヌル以外の値を設定します。

注 : 配布リストにメッセージを書き込むために MQPUT1 を使用する場合は、*ResponseRecPtr* をヌルまたはゼロにし、*ResponseRecOffset* をゼロにする必要があります。

バージョン 3 の MQPMO 構造体には、以下のフィールドが追加で含まれて います。

OriginalMsgHandle
このフィールドをどのように使用できるのかは、*Action* フィールドの値に依存します。 新規メッセージを、関連付けられたメッセージ・プロパティーと共に書き込もうとしている場合は、 前に作成して、プロパティーを設定したメッセージ・ハンドルを、このフィールドに設定し ます。 前に取り出したメッセージに応えて、転送、応答、報告書生成を 行おうとしている場合は、このフィールドにはそのメッセージのメッセージ・ハンドル が含まれます。

NewMsgHandle
NewMsgHandle が指定されている場合、 そのハンドルに関連付けられたプロパティーが、*OriginalMsgHandle* に関連付けられたプロパティーをオーバーライドします。 詳しくは、Action (MQLONG) を参照してください。

アクション
このフィールドを使用して、実行される書き込みのタイプを指定します。 指定できる値と その意味は、次のとおりです。

MQACTP_NEW
これは他のどのメッセージにも関連していない新規メッセージである。

MQACTP_FORWARD
このメッセージは前に取り出され、今は転送されようとしている。

MQACTP_REPLY
このメッセージは、前に取り出されたメッセージへの応答である。

MQACTP_REPORT
このメッセージは、前に取り出されたメッセージの結果として生成 された報告書である。

詳しくは、Action (MQLONG) を参照してください。

PubLevel
このメッセージがパブリケーションである場合、それをどのサブスクリプションが受け取るのかを決定するために、 このフィールドを設定できます。 *SubLevel* がこの値 以下のサブスクリプションのみが、このパブリケーションを受け取ります。 デフォルト値 は 9 で、これは最高レベルであり、どの *SubLevel* のサブスクリプションでもこのパブリケーションを受け取ることができることを意味します。

メッセージ内のデータ

メッセージが入っているバッファーのアドレスを、MQPUT 呼び出しの **Buffer** パラメーターに指定します。 メッセージ内のデータとしては、何を入れても差し支えありません。 しかし、メッセージ内のデータ量は、これらを処理するアプリケーションのパフォーマンスに影響します。

データの最大サイズは、次の事項によって決定されます。

- キュー・マネージャーの **MaxMsgLength** 属性
- メッセージを書き込むキューの **MaxMsgLength** 属性
- IBM MQ によって追加される任意のメッセージ・ヘッダーのサイズ (送達不能ヘッダーの MQDLH と配布リスト・ヘッダーの MQDH を含む)

キュー・マネージャーの **MaxMsgLength** 属性は、キュー・マネージャーが処理できるメッセージのサイズを保持します。 V6 以降のすべての IBM MQ 製品において、この属性のデフォルトは 100 MB です。

この属性の値を判別するために、キュー・マネージャー・オブジェクトに MQINQ 呼び出しを使用します。大きなメッセージの場合は、この値を変更できます。

キューの **MaxMsgLength** 属性によって、キューに書き込むことができるメッセージの最大サイズが決まります。 この属性の値より大きいサイズのメッセージを書き込もうとすると、その MQPUT 呼び出しは失敗します。 リモート・キューにメッセージを書き込む場合、正常に書き込むことができるメッセージの最大サイズは、関係するキューおよびチャネルの **MaxMsgLength** 属性によって決まります。関係するキューとは、宛先のリモート・キューと経路上でメッセージが書き込まれる中間伝送キューを指します。

MQPUT 操作の場合は、メッセージのサイズが、キューとキュー・マネージャーの両方の **MaxMsgLength** 属性の値以下でなければなりません。 これらの属性の値は互いに独立していますが、キューの *MaxMsgLength* の値は、キュー・マネージャーの値以下にしておくことをお勧めします。

IBM MQ では、次の場合にメッセージにヘッダー情報を追加します。

- メッセージをリモート・キューに書き込むと、IBM MQ は伝送ヘッダー (MQXQH) 構造体をメッセージに追加します。 この構造体は、宛先キューとそれを所有するキュー・マネージャーの名前を含んでいます。
- IBM MQ がリモート・キューにメッセージを送達できない場合、メッセージを送達不能 (未配布メッセージ) キューに書き込もうとします。 そして、メッセージに MQDLH 構造体を追加します。 この構造体には、宛先キューの名前と、そのメッセージが送達不能キューに書き込まれた理由が収められています。
- メッセージを複数の宛先キューに送信する場合は、IBM MQ が MQDH ヘッダーをメッセージに追加します。 このヘッダーは、伝送キュー上の、配布リストに属するメッセージ内に存在するデータを記述します。 最大メッセージ長に最適値を選択するときには、この点を考慮してください。
- メッセージがセグメント化されている場合や、メッセージがグループに入っている場合、IBM MQ は MQMDE を追加することがあります。

これらの構造体の説明については、MQDH および MQMDE を参照してください。

メッセージ長の合計が、これらのキューに対して設定されている最大サイズに達しているときに、これらのヘッダーが追加されると、メッセージが大きくなりすぎて書き込み操作が失敗します。 書き込み操作が失敗する可能性を減らすには、次のようにしてください。

- メッセージ長を、伝送キューと送達不能キューの **MaxMsgLength** 属性より短くします。 最低でも MQ_MSG_HEADER_LENGTH 定数の値にします (大きな配布リストの場合は、さらに大きくします)。
- 送達不能キューの **MaxMsgLength** 属性を必ず、送達不能キューを所有するキュー・マネージャーの *MaxMsgLength* と同じに設定します。

キュー・マネージャーの属性とメッセージ・キューイングの定数については、キュー・マネージャーの属性を参照してください。

z/OS 分散キューイング環境で未配信メッセージが処理される方法については、「未配布/未処理のメッセージ」を参照してください。

メッセージの書き込み: メッセージ・ハンドルの使用

MQPMO 構造体では、*OriginalMsgHandle* と *NewMsgHandle* という 2 つのメッセージ・ハンドルを使用できます。 それらのメッセージ・ハンドルの関係は、MQPMO の *Action* フィールドの値によって定義します。

詳しくは、Action (MQLONG) を参照してください。 メッセージを書き込むためにメッセージ・ハンドルが絶対に必要というわけではありません。 その目的は、メッセージにプロパティーを関連付けることなので、メッセージ・ハンドルが必要になるのは、メッセージ・プロパティーを使用する場合に限られます。

リモート・キューへのメッセージの書き込み

メッセージを、ローカル・キューではなく、リモート・キュー (つまり、アプリケーションの接続先のキュー・マネージャー以外のキュー・マネージャーが所有するキュー) に書き込む場合は、キューをオープンする時のキューの名前の指定方法を検討する必要があります。これは、752 ページの『リモート・キューのオープン』で説明されています。 ローカル・キューに対する MQPUT または MQPUT1 呼び出しの使用方法には変更はありません。

リモート・キューおよび伝送キューの使用方法の詳細については、 IBM MQ 分散キューイング技法を参照してください。

メッセージ・プロパティーの設定

設定したい各プロパティーごとに MQSETMP を呼び出します。 メッセージを 書き込むときに、MQPMO 構造体のメッセージ・ハンドルおよびアクションの各フィールドを設定します。

メッセージにプロパティーを関連付けるには、メッセージにメッセージ・ハンドル がなければなりません。 メッセージ・ハンドルの作成は MQCRTMH 機能呼び出しを使用して行います。 設定したい各プロパティーごとにこのメッセージ・ハンドルを指定して MQSETMP を 呼び出します。 MQSETMP の使用法を示すためのサンプル・プログラム amqsstma.c が用意されています。

これが新規メッセージである場合、MQPUT または MQPUT1 を使用してキューに書き込むときに、MQPMO 内の OriginalMsgHandle フィールドをこのメッセージ・ハンドルの値に設定し、 MQPMO 内の Action フィールドを MQACTP_NEW (これは デフォルト値です) に設定します。

これが前に取り出したことのあるメッセージであり、転送または応答しようとしているか、それに応えて報告書を送信しようとしている 場合、元のメッセージ・ハンドルを MQPMO の OriginalMsgHandle フィールドに、 新規メッセージ・ハンドルを NewMsgHandle フィールドに入れます。 Action フィールドを、MQACTP_FORWARD、 MQACTP_REPLY、または MQACTP_REPORT のうち適当なものに設定します。

前に取り出したメッセージからの MQRFH2 ヘッダー内にプロパティーがある場合、MQBUFMH 呼び出しを使用して、それらのプロパティーをメッセージ・ハンドル・プロパティーに変換 することができます。

メッセージ・プロパティーを処理できない、IBM WebSphere MQ 7.0 より前のレベルのキュー・マネージャーのキューにメッセージを書き込む場合、プロパティーがどのように処理されるのかを指定するために、チャネル定義に PropertyControl パラメーターを設定できます。

メッセージ・コンテキスト情報の制御

メッセージをキューに書き込むために MQPUT または MQPUT1 呼び出しを使用するときは、キュー・マネージャーがメッセージ記述子に何らかのデフォルト・コンテキスト情報を追加するように指定できます。 適切な許可レベルを持つアプリケーションは、余分のコンテキスト情報を追加できます。 MQPMO 構造体のオプション・フィールドを使用して、コンテキスト情報を制御できます。

メッセージ・コンテキスト情報により、メッセージを受信するアプリケーションは、そのメッセージの発信元についての情報を得ることができます。 すべてのコンテキスト情報は、メッセージ記述子のコンテキスト・フィールド内に保管されます。 情報のタイプは、識別コンテキスト情報、起点コンテキスト情報、およびユーザー・コンテキスト情報に分けられます。

コンテキスト情報を制御するには、MQPMO 構造体の *Options* フィールドを使用します。

コンテキスト情報のためのオプションを指定しなかった場合、キュー・マネージャーは、メッセージ記述子の中に既に存在しているコンテキスト情報を、そのメッセージ用に生成された識別コンテキスト情報で

上書きします。 このフィールドの使用は、MQPMO_DEFAULT_CONTEXT オプションを指定するのと同じです。新しいメッセージを生成するときに、このデフォルト・コンテキスト情報が必要となることがあります (例えば、照会画面からのユーザー入力を処理するとき)。

メッセージに関連するコンテキスト情報が不要であれば、MQPMO_NO_CONTEXT オプションを使用してください。 メッセージをコンテキストなしで送る場合、IBM MQ による権限検査は、ブランクのユーザー ID を使用して行われます。 ブランクのユーザー ID には、IBM MQ リソースに対する明示的な権限を割り当てることはできませんが、特別な 'nobody' グループのメンバーとして扱われます。 特別なグループ nobody の詳細については、インストール可能サービス・インターフェースの参照情報を参照してください。

コンテキスト設定は、MQOPEN に続いて MQPUT を使用することによって行えます。 その際、以下のセクションに示されているように MQOO_ オプションおよび MQPMO_ オプションを使用します。 また、MQPUT1 だけを使用してコンテキスト設定を行うこともできます。 その場合には、以下のセクションに示されているように MQPMO_ オプションを選択することだけが必要です。

このトピックの続くセクションに、識別コンテキスト、ユーザー・コンテキスト、すべてのコンテキストの使用についての説明があります

- 760 ページの『識別コンテキストを渡す方法』
- 760 ページの『ユーザー・コンテキストを渡す方法』
- 761 ページの『すべてのコンテキストを渡す方法』
- 761 ページの『識別コンテキストを設定する方法』
- 761 ページの『ユーザー・コンテキストの設定』
- 761 ページの『すべてのコンテキストを設定する方法』

識別コンテキストを渡す方法

一般に、プログラムは、データが最終宛先に到着するまで、アプリケーション内でメッセージからメッセージに識別コンテキスト情報を渡さなければなりません。

プログラムは、データを変更するたびに起点コンテキスト情報を変更しなければなりません。 しかし、コンテキスト情報を変更または設定しようとするアプリケーションは、適切なレベルの許可を持っていなければなりません。 アプリケーションがキューをオープンするとき、キュー・マネージャーはこの許可を検査します。アプリケーションは、MQOPEN 呼び出しに対する適切なコンテキスト・オプションを用いる許可を持っていなければなりません。

アプリケーションがメッセージを入手し、このメッセージからのデータを処理し、(例えば別のアプリケーションに処理させるために) 変更済みデータを別のメッセージに入れる場合、アプリケーションは元のメッセージから新しいメッセージに識別コンテキスト情報を渡さなければなりません。 起点コンテキスト情報の作成はキュー・マネージャーに任せることができます。

元のメッセージのコンテキスト情報を保管するには、メッセージを読み取るキューをオープンするときに、MQOO_SAVE_ALL_CONTEXT オプションを使用してください。 これは、MQOPEN 呼び出しで使用する他のオプションに加えて使用します。 しかし、メッセージをブラウズするだけの場合は、コンテキスト情報を保管できないことに注意してください。

2 番目のメッセージを作成するときは、以下のようにします。

- (MQOO_OUTPUT オプションに加えて)、MQOO_PASS_IDENTITY_CONTEXT オプションを使ってキューをオープンする。
- 書き込みメッセージ・オプション構造体の *Context* フィールドに、コンテキスト情報の保管元のキューのハンドルを指定する。
- 書き込みメッセージ・オプション構造体の *Options* フィールドに、MQPMO_PASS_IDENTITY_CONTEXT オプションを指定する。

ユーザー・コンテキストを渡す方法

ユーザー・コンテキストのみを渡すよう選択することはできません。 メッセージを書き込むときにユーザー・コンテキストを 渡すには、MQPMO_PASS_ALL_CONTEXT を指定します。 ユーザー・コンテキスト内の すべてのプロパティーが、起点コンテキストと同じようにして渡されます。

MQPUT または MQPUT1 が実行されてコンテキストが引き渡されると、ユーザー・コンテキスト内の全プロパティーは、取得されたメッセージから書き込みメッセージに引き渡されます。 書き込みアプリケーションによって変更されたユーザー・コンテキスト・プロパティーは、すべて元の値で書き込まれます。 書き込みアプリケーションによって削除されたユーザー・コンテキスト・プロパティーは、すべて書き込みメッセージ内で復元されます。 書き込みアプリケーションによってメッセージに追加されたユーザー・コンテキスト・プロパティーは、すべて保存されます。

すべてのコンテキストを渡す方法

アプリケーションがメッセージを入手し、そのメッセージ・データを (変更せずに) 別のメッセージに入れる場合、アプリケーションは元のメッセージから新しいメッセージに、すべてのコンテキスト情報 (識別コンテキスト情報、起点コンテキスト情報、およびユーザー・コンテキスト情報) を渡さなければなりません。 これを行うアプリケーションの例としては、メッセージを 1 つのキューから別のキューへ移動するメッセージ・ムーバーがあります。

MQOPEN オプションの MQOO_PASS_ALL_CONTEXT と、書き込みメッセージ・オプションの MQPMO_PASS_ALL_CONTEXT を使用する以外は、識別コンテキストを渡す手順と同じ手順に従ってください。

識別コンテキストを設定する方法

メッセージの識別コンテキスト情報を設定したい場合は、次のようにします。

- MQOO_SET_IDENTITY_CONTEXT オプションを用いて、キューをオープンする。
- MQPMO_SET_IDENTITY_CONTEXT オプションを指定して、メッセージをキューに書き込む。 メッセージ記述子に、必要な識別コンテキスト情報をすべて指定する。

注 : MQOO_SET_IDENTITY_CONTEXT および MQPMO_SET_IDENTITY_CONTEXT オプションを使用して、いくつか (全部ではない) の識別コンテキスト・フィールドを設定する場合、キュー・マネージャーが他のいずれのフィールドも設定しないということを意識する必要があります。

メッセージ・コンテキスト・オプションのいずれかを変更するためには、呼び出しを発行するための適切な許可が必要です。 例えば、MQOO_SET_IDENTITY_CONTEXT または MQPMO_SET_IDENTITY_CONTEXT を使用するには、`+setid` アクセス権が必要です。

ユーザー・コンテキストの設定

ユーザー・コンテキスト内にプロパティーを設定するには、MQSETMP 呼び出しを行うときに メッセージ・プロパティー記述子 (MQPD) の Context フィールドを MQPD_USER_CONTEXT に設定します。

ユーザー・コンテキストにプロパティーを設定するために、特殊権限は必要ではありません。 ユーザー・コンテキストには、MQOO_SET_*または MQPMO_SET_* コンテキスト・オプションはありません。

すべてのコンテキストを設定する方法

メッセージの識別コンテキスト情報と起点コンテキスト情報の両方を設定したい場合は、次のようにします。

1. MQOO_SET_ALL_CONTEXT オプションを使用して、キューをオープンする。
2. MQPMO_SET_ALL_CONTEXT オプションを指定して、メッセージをキューに書き込む。 メッセージ記述子に、必要な識別コンテキスト情報と起点コンテキスト情報をすべて指定する。

各種のコンテキストの設定には、適切な許可が必要です。

関連概念

47 ページの『メッセージ・コンテキスト』
メッセージ・コンテキスト 情報により、メッセージを受信するアプリケーションは、そのメッセージの発信元についての情報を得ることができます。

関連資料

750 ページの『メッセージ・コンテキストに関連する MQOPEN オプション』

メッセージをキューに書き込むときに、コンテキスト情報とメッセージの関連付けができるようにしたい場合は、キューをオープンするときにメッセージ・コンテキスト・オプションの 1 つを使用しなければなりません。

MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み

キューにメッセージを 1 つ書き込んだ直後にキューをクローズするときは、MQPUT1 呼び出しを使用します。 例えば、サーバーのアプリケーションは、異なる各キューに応答を送信するとき、MQPUT1 呼び出しを用いることがあります。

MQPUT1 の機能は、MQOPEN、MQPUT、MQCLOSE を順番に呼び出した場合と同じです。 MQPUT 呼び出しと MQPUT1 呼び出しの構文上の唯一の違いは、MQPUT ではオブジェクト・ハンドルを指定するのに対して、MQPUT1 ではオブジェクト記述子構造体 (MQOD) を MQOPEN に定義されているように指定することです (744 ページの『オブジェクトの識別 (MQOD 構造体)』 を参照してください)。 これは、MQPUT1 を呼び出す時にはオープンするキューに関する情報を与えなければならないのに対して、MQPUT を呼び出すときはキューが既にオープンしていなければならないからです。

MQPUT1 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。
- オープンしたいオブジェクトの記述。 これは、オブジェクト記述子構造体 (MQOD) の形式です。
- キューに書き込みたいメッセージについての記述。 これは、メッセージ記述子構造体 (MQMD) の形式になります。
- 書き込みメッセージ・オプション構造体 (MQPMO) の形式による制御情報。
- メッセージ内に含まれるデータの長さ (MQLONG)。
- メッセージ・データのアドレス。

MQPUT1 の出力は以下のとおりです。

- 完了コード
- 理由コード

呼び出しが正常に完了した場合は、オプション構造体とメッセージ記述子構造体も戻します。 この呼び出しは、メッセージ送信先のキューおよびキュー・マネージャーの名前を示すために、オプション構造体を変更します。 (MQMD 構造体の *MsgId* フィールドに 2 進数のゼロを指定することによって)、書き込むメッセージの ID として固有な値をキュー・マネージャーに生成させる場合、この呼び出しは、この構造体を戻す前に *MsgId* フィールドにその値を入れます。

注 : モデル・キュー名で MQPUT1 を使用することはできません。しかし、いったんモデル・キューがオープンされれば、MQPUT1 を動的キューに対して発行できます。

MQPUT1 に対する入力パラメーターは次の 6 つです。

Hconn
これは接続ハンドルです。 CICS アプリケーションの場合は、(ゼロの値を持つ) 定数 MQHC_DEF_HCONN を指定するか、または MQCONN か MQCONNX 呼び出しによって戻される接続ハンドルを使用します。 他のプログラムの場合は、常に MQCONN 呼び出しか MQCONNX 呼び出しによって戻される接続ハンドルを用います。

ObjDesc
これはオブジェクト記述子構造体 (MQOD) です。

ObjectName および *ObjectQMgrName* フィールドには、メッセージを書き込みたいキューの名前と、このキューを所有するキュー・マネージャーの名前をそれぞれ指定します。

モデル・キューは使用できないため、MQPUT1 呼び出しに対する *DynamicQName* フィールドは無視されます。

AlternateUserId フィールドは、キューをオープンするための許可の検査に使う代替ユーザー ID を指定したい場合に使用します。

MsgDesc
これは、メッセージ記述子構造体 (MQMD) です。 MQPUT 呼び出しと同様に、この構造体を使用して、キューに書き込むメッセージを定義します。

PutMsgOpts
これは、書き込みメッセージ・オプション構造体 (MQPMO) です。 MQPUT 呼び出しのときと同様に、この構造体を使用してください (755 ページの『MQPMO 構造体を使用するオプションの指定』を参照)。

Options フィールドがゼロに設定されていると、キュー・マネージャーは、キューへのアクセス許可を検査するときに、独自のユーザー ID を使用します。 また、キュー・マネージャーは、MQOD 構造体の *AlternateUserId* フィールドに指定されている代替ユーザー ID を無視します。

BufferLength
これはメッセージの長さです。

Buffer
これは、メッセージのテキストを含むバッファーです。

クラスターを使用すると、MQPUT1 は MQOO_BIND_NOT_FIXED が有効になっている場合と同じ動作をします。 メッセージの送信先を判別する場合、各アプリケーションは MQOD 構造体ではなく MQPMO 構造体の解決済みフィールドを使用する必要があります。 詳細については、キュー・マネージャー・クラスターの構成を参照してください。

MQPUT1 呼び出しの説明については、MQPUT1 を参照してください。

配布リスト

IBM MQ for z/OS ではサポートされない。 配布リストを使用すると、単一の MQPUT または MQPUT1 呼び出しでメッセージを複数の宛先に書き込むことができます。単一の MQOPEN 呼び出しで複数のキューを開いた後、 単一の MQPUT 呼び出しでそれらの各キューにメッセージを書き込むことができます。 このプロセスで使用される MQI 構造体からの汎用情報の一部は、宛先リストに含まれている個々の宛先に関係する特定の情報に置き換えることができます。

重要 : 配布リストでは、トピック・オブジェクトを指す別名キューの使用はサポートされていません。 別名キューが配布リスト内のトピック・オブジェクトを指し示す場合、IBM MQ は MQRC_ALIAS_BASE_Q_TYPE_ERROR を戻します。

MQOPEN 呼び出しを使用すると、オブジェクト記述子 (MQOD) から汎用情報が取り込まれます。 *Version* フィールドに MQOD_VERSION_2 を指定し、*RecsPresent* フィールドにゼロより大きい値を指定すると、*Hobj* を、キューのハンドルではなく、(1 つまたは複数のキューの) リストのハンドルとして定義できます。この場合は、オブジェクト・レコード (MQOR) から特定の情報が提供されます。オブジェクト・レコードからは、宛先の詳細 (つまり、*ObjectName* と *ObjectQMgrName*) が提供されます。

オブジェクト・ハンドル (*Hobj*) は MQPUT 呼び出しに渡されるので、単一のキューではなくリストへの書き込みが可能になります。

メッセージをキューに書き込むと (MQPUT)、書き込みメッセージ・オプション構造体 (MQPMO) とメッセージ記述子構造体 (MQMD) から汎用情報が送り込まれます。 特定の情報は、書き込みメッセージ・レコード (MQPMR) の形式で提供されます。

応答レコード (MQRR) では、各宛先キューごとの完了コードと理由コードを受け取ることができます。

764 ページの図 56 には、配布リストの機能が示されています。

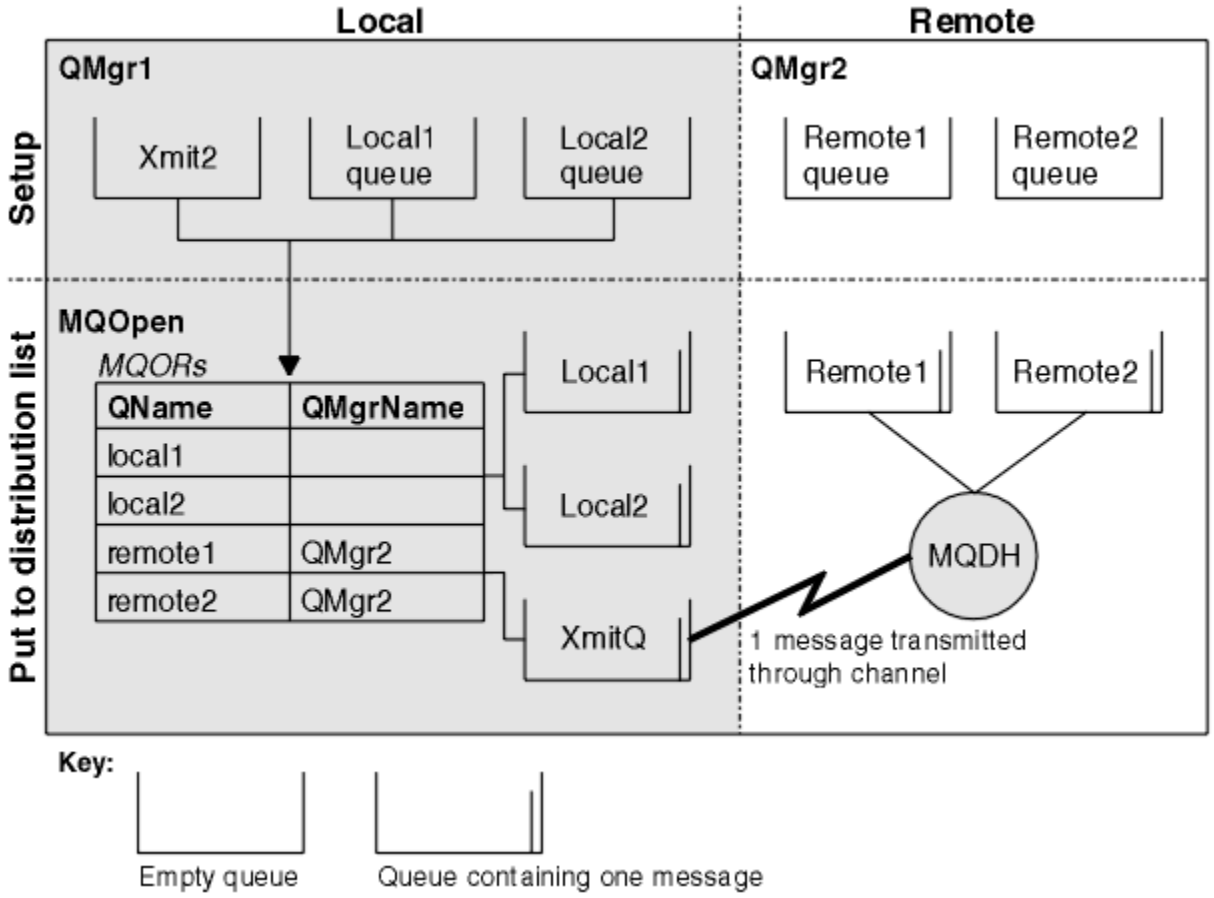


図 56. 配布リストの機能

配布リストのオープン
MQOPEN 呼び出しを使用して配布リストをオープンし、その呼び出しの各オプションを使用してそのリストの操作内容を指定します。

MQOPEN への入力として、次のものを提供する必要があります。

- 接続ハンドル (詳細は、753 ページの『キューへのメッセージの書き込み』を参照)
- オブジェクト記述子構造体 (MQOD) の汎用情報
- オープンしたい各キューの名前 (オブジェクト・レコード構造体 (MQOR) を使用)

MQOPEN の出力は以下のとおりです。

- 配布リストへのアクセスを表すオブジェクト・ハンドル
- 汎用完了コード
- 汎用理由コード
- 応答レコード (オプション) (各宛先の完了コードと理由を含む)

MQOD 構造体の使用

MQOD 構造体を使用して、オープンするキューを指定します。

配布リストを定義するには、*Version* フィールドに MQOD_VERSION_2 を指定し、*RecsPresent* フィールドにゼロより大きい値を指定し、*ObjectType* フィールドに MQOT_Q を指定する必要があります。
MQOD 構造体のすべてのフィールドについては、MQOD を参照してください。

MQOR 構造体の使用

各宛先ごとに MQOR 構造体を指定してください。

この構造体には、宛先キューとキュー・マネージャーの名前を組み込みます。 MQOD の *ObjectName* および *ObjectQMgrName* フィールドは、配布リストには使用しません。 1 つ以上のオブジェクト・レコードを組み込む必要があります。 *ObjectQMgrName* をブランクのままにすると、ローカル・キュー・マネージャーが使用されます。 これらのフィールドの詳細については、ObjectName および ObjectQMgrName を参照してください。

宛先キューを指定するには、2 つの方法があります。

- オフセット・フィールド *ObjectRecOffset* を使用する。

  この場合は、アプリケーションで、MQOD 構造体を組み込んだ独自の構造体を宣言し、その後に (必要な数の配列エレメントを使用して) MQOR レコードの配列を記述し、MQOD の開始点を基準にした配列内の最初のエレメントのオフセットを *ObjectRecOffset* に設定します。 このオフセットが正しいか、確かめてください。

  アプリケーションを実行するどの環境でも、プログラミング言語に用意されている組み込み機能を使用できる場合は、その組み込み機能を使用することをお勧めします。 COBOL プログラミング言語でその技法を使用するコード例を以下に示します。

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  対象のすべての環境で必要になる組み込み機能がプログラミング言語でサポートされていない場合は、定数 MQOD_CURRENT_LENGTH を使用します。 その技法を使用するコード例を以下に示します。

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  ただし、このコードが正しく動作するのは、MQOD 構造体と MQOR レコードの配列が隣接している場合に限られます。コンパイラーが MQOD と MQOR 配列の間にスキップ・バイトを挿入する場合は、*ObjectRecOffset* の格納値にそのバイト数を追加する必要があります。

  ポインター・データ型をサポートしていないプログラミング言語や、別の環境に移植できない方法でポインター・データ型を実装しているプログラミング言語 (COBOL プログラミング言語など) の場合は、*ObjectRecOffset* を使用することをお勧めします。

- ポインター・フィールド *ObjectRecPtr* を使用する。

  この場合は、アプリケーションで MQOD 構造体とは別に MQOR 構造体の配列を宣言でき、*ObjectRecPtr* に配列のアドレスを設定できます。 C プログラミング言語でその技法を使用するコード例を以下に示します。

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  別の環境に移植できる方法でポインター・データ型をサポートしているプログラミング言語 (C プログラミング言語など) の場合は、*ObjectRecPtr* を使用することをお勧めします。

どの技法を選択するにしても、*ObjectRecOffset* と *ObjectRecPtr* のいずれかを使用する必要があります。どちらもゼロの場合やどちらもゼロ以外の場合は、呼び出しが理由コード MQRC_OBJECT_RECORDS_ERROR で失敗します。

MQRR 構造体の使用

これらの構造体は、宛先ごとに記述します。つまり、配布リストのキューごとに *CompCode* フィールドと *Reason* フィールドを各応答レコードに組み込みます。 問題が存在する場所を確認できるようにするために、この構造体を使用する必要があります。

例えば、配布リストに 5 つの宛先キューが含まれている場合に、この構造体を使用しなければ、理由コード MQRC_MULTIPLE_REASONS を受け取ったときに、問題がどのキューに該当するのかを確認できません。一方、宛先ごとに完了コードと理由コードを受け取れば、エラーの場所を容易に判別できます。

MQRR 構造体の詳細については、MQRR を参照してください。

C で配布リストをオープンする方法を 766 ページの図 57 に示します。

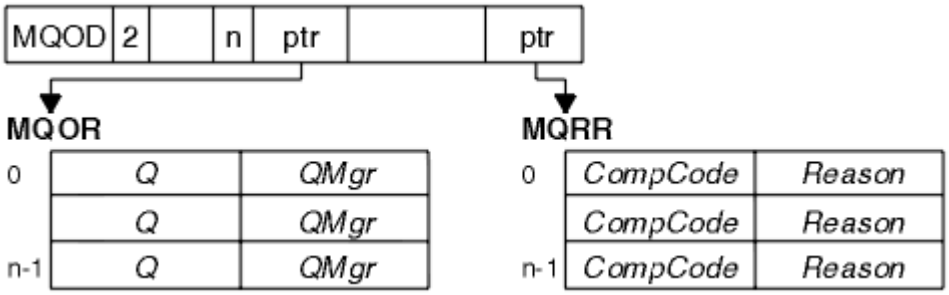


図 *57. C* による配布リストのオープン

COBOL で配布リストをオープンする方法を 766 ページの図 58 に示します。

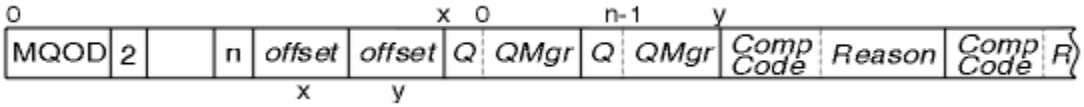


図 *58. COBOL* による配布リストのオープン

MQOPEN の各オプションの使用

配布リストをオープンするときには、以下のオプションを指定できます。

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (オプション)
- MQOO_ALTERNATE_USER_AUTHORITY (オプション)
- MQOO_*_CONTEXT (オプション)

これらのオプションの説明については、742 ページの『オブジェクトのオープンとクローズ』を参照してください。

配布リストへのメッセージの書き込み

配布リストにメッセージを書き込むには、MQPUT または MQPUT1 を使用します。

入力として、次のものを提供する必要があります。

- 接続ハンドル (詳細は、753 ページの『キューへのメッセージの書き込み』を参照)。
- オブジェクト・ハンドル。 MQOPEN を使って配布リストをオープンした場合は、*Hobj* を使用するとリストへの書き込みだけができます。
- メッセージ記述子構造体 (MQMD)。 この構造体の詳細については、MQMD を参照してください。
- 書き込みメッセージ・オプション構造体の形式による制御情報 (MQPMO)。 MQPMO 構造体のフィールドへの入力については、755 ページの『MQPMO 構造体を使用するオプションの指定』を参照してください。
- 書き込みメッセージ・レコード (MQPMR) の形式による制御情報。
- メッセージ内に含まれるデータの長さ (MQLONG)。

- メッセージ・データそのもの。

出力は以下のとおりです。

- 完了コード
- 理由コード
- 応答レコード (オプション)

MQPMR 構造体の使用

この構造体を使用するかどうかは任意です。この構造体では、MQMD で既に指定したフィールドとは別に指定したいと思っているいくつかのフィールドの宛先固有の情報を記述します。

これらのフィールドの説明については、MQPMR を参照してください。

各レコードの内容は、MQPMO の *PutMsgRecFields* フィールドで指定する情報によって異なります。 例えば、配布リストの使用法を示したサンプル・プログラム AMQSPTL0.C (説明については、1093 ページの『配布リスト・サンプル・プログラム』 を参照) では、MQPMR で *MsgId* と *CorrelId* の値を指定できるように記述内容が選択されています。 サンプル・プログラムのこの部分は、次のようになっています。

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

このプログラムでは、*MsgId* と *CorrelId* が配布リストの各宛先ごとに提供されます。 書き込みメッセージ・レコードは、配列として記述されています。

C で配布リストにメッセージを書き込む方法を 767 ページの図 59 に示します。

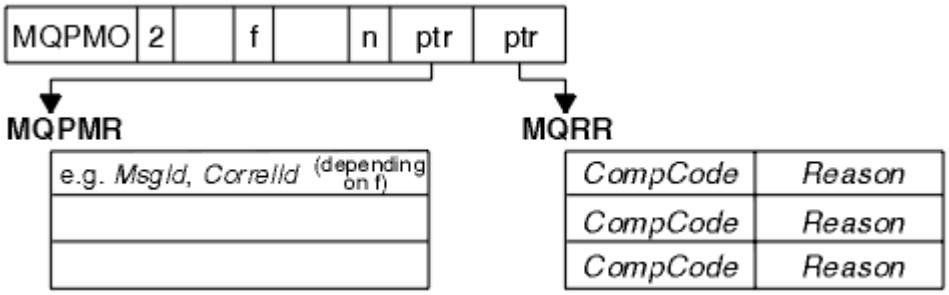


図 *59. C* による配布リストへのメッセージの書き込み

COBOL で配布リストにメッセージを書き込む方法を 767 ページの図 60 に示します。

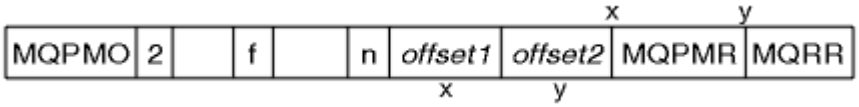


図 *60. COBOL* による配布リストへのメッセージの書き込み

MQPUT1 の使用

MQPUT1 を使用する場合の考慮点を以下に示します。

1. *ResponseRecOffset* フィールドと *ResponseRecPtr* フィールドの値は、ヌルまたはゼロでなければなりません。
2. 応答レコードが必要な場合は、MQOD から応答レコードのアドレス指定を行う必要があります。

書き込み呼び出しが失敗する場合

キューの特定の属性が、MQOPEN 呼び出しと MQPUT 呼び出しの間の期間にコマンドの FORCE オプションを使用して変更された場合、MQPUT 呼び出しは失敗し、MQRC_OBJECT_CHANGED 理由コードが戻されます。

キュー・マネージャーは、オブジェクト・ハンドルを無効としてマーク付けします。 このことは、MQPUT1 呼び出しの処理中に変更が行われたり、キュー名が解決された結果のキューにその変更が適用されたりする場合にも起こります。 このようにハンドルに影響を及ぼす属性は、MQOPEN の MQOPEN 呼び出しの節で一覧表示されています。 呼び出しが MQRC_OBJECT_CHANGED 理由レコードを戻す場合は、キューをクローズしてキューを再度オープンしてから、メッセージの書き込みを行います。

メッセージを書き込もうとしているキュー (または、キュー名が解決された結果のキュー) に対して書き込み操作が禁止されている場合は、MQPUT または MQPUT1 呼び出しは失敗し、MQRC_PUT_INHIBITED の理由コードが戻されます。 アプリケーションの設計として、他のプログラムがキューの属性を規則的に変更するようになっているのであれば、後で呼び出しをするとメッセージを正しく書き留める可能性があります。あとで呼び出しを再実行すると、書き込みに成功する場合もあります。ただし、ほかのプログラムがキューの属性を定期的に変更するようにアプリケーションが設計されていることが前提です。

さらに、メッセージを書き込もうとしているキューが満ぱいの場合は、MQPUT または MQPUT1 呼び出しは失敗し、MQRC_Q_FULL が戻されます。

(一時または永続) 動的キューが削除されていると、前に取得したオブジェクト・ハンドルを使用した MQPUT 呼び出しは失敗し、MQRC_Q_DELETED の理由コードを戻します。 この場合、オブジェクト・ハンドルは必要なくなるので、クローズすることをお勧めします。

配布リストの場合は、1 つの要求で複数の完了コードと理由コードが発生することがあります。 そのため、MQOPEN と MQPUT の *CompCode* と *Reason* の出力フィールドだけを使って配布リストを処理することはできません。

複数の宛先にメッセージを書き込むために配布リストを使用すると、応答レコードには、各宛先ごとに特定の *CompCode* と *Reason* が設定されます。 完了コード MQCC_FAILED を受け取った場合は、どの宛先キューにも正常にはメッセージが書き込まれていません。 完了コードが MQCC_WARNING の場合は、1 つ以上の宛先キューにメッセージが正常に書き込まれます。 戻りコード MQRC_MULTIPLE_REASONS を受け取った場合は、すべての宛先で理由コードが同じであるとは限りません。 したがって、エラーの原因となった 1 つまたは複数のキューと、それぞれの理由を判別できるように、MQRR 構造体を使用することをお勧めします。

キューからのメッセージの読み取り

この情報を使用して、キューからのメッセージの読み取りについて学習します。

次の 2 通りの方法で、キューからメッセージを読み取ることができます。

1. 他のプログラムがもうメッセージを見ることができないように、キューからメッセージを除去できます。
2. 元のメッセージをキューに残して、メッセージをコピーできます。 これを、ブラウズ といいます。 一度ブラウズされたメッセージは除去できます。

どちらの場合も、MQGET 呼び出しを使用しますが、最初のアプリケーションはキュー・マネージャーに接続されていなければなりません。また、(入力用、ブラウズ用、またはその両方用の) キューをオープンするために MQOPEN 呼び出しを使用する必要があります。 これらの操作については、735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』 および 742 ページの『オブジェクトのオープンとクローズ』 に記載されています。

キューをオープンすると、そのキューのメッセージを MQGET 呼び出しを繰り返し使用して、ブラウズしたり除去したりできます。 必要なメッセージをキューからすべて読み取ったら、MQCLOSE を呼び出します。

メッセージをキューから読み取る方法について詳しくは、以下のリンクを参照してください。

- 769 ページの『MQGET 呼び出しの使用によるキューからのメッセージの読み取り』
- 773 ページの『メッセージがキューから取り出される順序』
- 785 ページの『特定のメッセージの読み取り』

- 787 ページの『非永続メッセージのパフォーマンス向上』
- z/OS 791 ページの『索引のタイプ』
- 791 ページの『4 MB より長いメッセージの処理』
- 797 ページの『メッセージの待機』
- z/OS 798 ページの『信号機能』
- 799 ページの『バックアウトのスキップ』
- 802 ページの『アプリケーション・データの変換』
- 803 ページの『キュー上のメッセージのブラウズ』
- 809 ページの『MQGET 呼び出しが失敗する場合』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQGET 呼び出しの使用によるキューからのメッセージの読み取り

MQGET 呼び出しを使用すると、オープン・ローカル・キューのメッセージを読み取ることができます。 別のシステム上のキューのメッセージを読み取ることはできません。

MQGET 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。
- キュー・ハンドル。
- キューから読み取りたいメッセージの記述。 これは、メッセージ記述子 (MQMD) 構造体の形式です。
- 読み取りメッセージ・オプション (MQGMO) 構造体の形式による制御情報。
- メッセージを保持するため割り当てたバッファーのサイズ (MQLONG)。
- メッセージを書き込むストレージのアドレス。

MQGET の出力は以下のとおりです。

- 理由コード
- 完了コード
- 呼び出しが正常に完了した場合の、指定したバッファー域のメッセージ
- メッセージが検索されたキューの名前を示すように修正された、オプション構造体
- 検索されたメッセージを記述するためにフィールドの内容が修正された、メッセージ記述子構造体
- メッセージ長 (MQLONG)

MQGET 呼び出しの説明については、MQGET を参照してください。

以下の節では、MQGET 呼び出しへの入力として指定しなければならない情報を記述します。

- 770 ページの『接続ハンドルの指定』
- 770 ページの『MQMD 構造体および MQGET 呼び出しの使用によるメッセージの記述』
- 770 ページの『MQGMO 構造体を使用する MQGET オプションの指定』
- 773 ページの『バッファー域のサイズの指定』

接続ハンドルの指定

z/OS z/OS アプリケーション上の CICS の場合、定数 MQHC_DEF_HCONN (0 の値を持つ) を指定するか、MQCONN または MQCONNX 呼び出しによって戻された接続ハンドルを使用することができます。その他のアプリケーションの場合は、常に MQCONN 呼び出しまたは MQCONNX 呼び出しから返される接続ハンドルを使用します。

キュー・ハンドル (*Hobj*) は、MQOPEN を呼び出したときに戻されたものを使用します。

MQMD 構造体および MQGET 呼び出しの使用によるメッセージの記述

キューから読み取ろうとするメッセージを識別するには、メッセージ記述子構造体 (MQMD) を用いてください。

これは、MQGET 呼び出し用の入出力パラメーターです。 MQMD で記述するメッセージ・プロパティーの概要については、18 ページの『IBM MQ メッセージ』を参照してください。その構造体そのものの説明については、MQMD を参照してください。

キューから読み取りたいメッセージが分かっている場合は、785 ページの『特定のメッセージの読み取り』を参照してください。

特定のメッセージを指定しなければ、MQGET はキュー内の最初の メッセージを検索します。 773 ページの『メッセージがキューから取り出される順序』では、メッセージの優先順位、つまり、キューの **MsgDeliverySequence** 属性および MQGMO_LOGICAL_ORDER オプションによって、キューの中でメッセージ順序がどのように決められるかを説明しています。

注 : (例えば、キュー内のメッセージを順次に取り出すために) MQGET を複数回使用したい場合は、各呼び出しのあとに、この構造体の *MsgId* および *CorrelId* フィールドをヌルに設定しなければなりません。これにより、検索されたメッセージの ID のフィールドが消去されます。

ただし、メッセージをグループ化したい場合には、同一のグループのメッセージについては同じ *GroupId* 値を使用する必要があります。これにより、MQGET 呼び出しで前のメッセージと同じ ID をもつメッセージが検索され、あるグループに属するすべてのメッセージを取り出すことができます。

MQGMO 構造体を使用する MQGET オプションの指定

MQGMO 構造体は、オプションを MQGET 呼び出しに渡すための入出力変数です。 以下の節を参考にして、この構造体の フィールドに入力してください。

MQGMO 構造体の説明については、MQGMO を参照してください。

StrucId

StrucId は、構造体を読み取りメッセージ・オプション構造体として識別するために使用する 4 文字のフィールドです。 MQGMO_STRUC_ID の指定は必須です。

Version

Version は、構造体のバージョン番号を記述します。 MQGMO_VERSION_1 がデフォルト値です。「バージョン 2」フィールドを使用するか論理順序でメッセージを取得する場合は、MQGMO_VERSION_2 を指定します。 「バージョン 3」フィールドを使用するか論理順序でメッセージを取得する場合は、MQGMO_VERSION_3 を指定します。 MQGMO_CURRENT_VERSION は、最新のレベルを使用するようにアプリケーションを設定します。

Options

コード内では、任意の順序でオプションを選択できます。各オプションは、*Options* フィールド内のビットによって表されます。

Options フィールドは、次の事柄を制御します。

- MQGET 呼び出しが、完了前にメッセージがキューに到着するのを待機するかどうか (797 ページの『メッセージの待機』を参照)
- 読み取り操作が作業単位に含まれるかどうか
- 非持続メッセージを同期点以外で取り出すことによって、高速メッセージングを可能にするかどうか
- z/OS IBM MQ for z/OS で、検索されたメッセージが、バックアウトをスキップするものとしてマーク付けされるかどうか (799 ページの『バックアウトのスキップ』を参照)
- メッセージがキューから除去されるのか、単にブラウズされるのか
- メッセージの選択に、ブラウズ・カーソルを使用するか、それとも他の選択基準を使用するか
- メッセージ長がバッファーより長い場合でも、呼び出しが成功するかどうか
- z/OS IBM MQ for z/OS で、呼び出しを完了させるかどうか。 このオプションでは、メッセージ到着を通知する信号を設定するかどうかも設定します。
- キュー・マネージャーが静止状態のとき、呼び出しをエラーにするかどうか
- z/OS IBM MQ for z/OS で、接続が静止状態のときに呼び出しを失敗させるかどうか
- アプリケーションのメッセージ・データの変換が必要かどうか (802 ページの『アプリケーション・データの変換』を参照)
- メッセージとセグメントがキューから取り出される順序 z/OS (IBM MQ for z/OS を除く)
- 完全な論理メッセージのみが取り出し可能であるかどうか z/OS (IBM MQ for z/OS を除く)
- あるグループ内のすべての メッセージが使用可能な場合のみ、そのグループ内のメッセージを取り出せるようにするかどうか
- ある論理メッセージ内のすべて のセグメントが使用可能な場合にのみ、その論理メッセージ内のセグメントを取り出せるようにするかどうか z/OS (IBM MQ for z/OS を除く)

Options フィールドをデフォルト値 (MQGMO_NO_WAIT) にしておくと、MQGET 呼び出しは次のように機能します。

- 選択基準に合致するメッセージがキューにない場合、呼び出しはメッセージの到着を待たずに、即時完了します。 z/OS また、IBM MQ for z/OS での呼び出しは、このようなメッセージが到着したときの通知を要求する信号を設定しません。
- 呼び出しと同期点の関係は、プラットフォームによって次のように設定されています。

プラットフォーム	同期点の制御下にあるか
IBM i	いいえ
AIX and Linux システム	いいえ
z/OS z/OS z/OS	はい

プラットフォーム	同期点の制御下にあるか
Windows システム	いいえ

- z/OS IBM MQ for z/OS では、検索されたメッセージはバックアウトをスキップするものとしてマーク付けされません。
- 選択されたメッセージはキュー除去されます (ブラウズされない)。
- アプリケーション・メッセージのデータ変換は、必要ありません。
- メッセージがバッファーより長い場合は、呼び出しが失敗します。

WaitInterval
WaitInterval フィールドは、MQGMO_WAIT オプションを使用したときに、MQGET 呼び出しがキューにメッセージが到着するのを待機する最長時間 (ミリ秒) を指定します。 *WaitInterval* に指定した時間内にメッセージが到着しない場合は、呼び出しが完了し、選択基準に合致するメッセージがキュー上になかったことを示す理由コードを戻します。

z/OS IBM MQ for z/OS では、MQGMO_SET_SIGNAL オプションを使用する場合、*WaitInterval* フィールドには、シグナルが設定される時間を指定します。

これらのオプションの詳細については、797 ページの『メッセージの待機』 z/OS および 798 ページの『信号機能』 を参照してください。

z/OS ***Signal1***
Signal1 は、 IBM MQ for z/OS でのみサポートされます。

MQGMO_SET_SIGNAL オプションを使用して、適切なメッセージが到着したときにアプリケーションに通知するように要求する場合は、*Signal1* フィールドに信号のタイプを指定します。 他のすべてのプラットフォームの IBM MQ では、 *Signal1* フィールドは予約されており、その値は重要ではありません。

z/OS 詳細については、798 ページの『信号機能』を参照してください。

Signal2
Signal2 フィールドはすべてのプラットフォームで予約されており、その値は有効ではありません。

z/OS 詳しくは、798 ページの『信号機能』 を参照してください。

ResolvedQName
ResolvedQName は、キュー・マネージャーが、メッセージを検索したキューの名前 (別名を解決したあとの) を戻すための出力フィールドです。

MatchOptions
MatchOptions は、MQGET の選択基準を制御します。

GroupStatus
GroupStatus は、取り出したメッセージがグループに属しているかどうかを示します。

SegmentStatus
SegmentStatus は、取り出した項目が論理メッセージのセグメントかどうかを示します。

Segmentation
Segmentation は、取り出されたメッセージに対してセグメント化を実行できるかを示します。

MsgToken

MsgToken は、メッセージを一意に識別します。

ReturnedLength
ReturnedLength は、キュー・マネージャーが、戻されたメッセージ・データの長さ (バイト単位) を戻すための出力フィールドです。

MsgHandle
キューから取り出されるメッセージの プロパティーが設定される、メッセージのハンドル。 このハンドルは、 前に MQCRTMH 呼び出しで作成されたものです。 ハンドルに既に関連付けられているプロパティーは、メッセージを取り出す前にすべて クリアされます。

バッファー域のサイズの指定

MQGET 呼び出しの **BufferLength** パラメーターには、検索したメッセージ・データを保持するバッファー域のサイズを指定します。 この大きさを決める方法には、以下の 3 通りがあります。

1. このプログラムから出されるメッセージの長さが既に分かっている場合があります。 その場合は、そのサイズのバッファーを指定してください。

   しかし、メッセージがバッファーより長い場合でも、MQGET 呼び出しを完了させるには、MQGMO 構造体に MQGMO_ACCEPT_TRUNCATED_MSG オプションを使用できます。 その場合は、次のようにします。

   - バッファーには、保持できる分だけのメッセージが入る。
   - 呼び出しは警告の完了コードを戻す。
   - メッセージがキューから除去される (メッセージの残りの部分は切り捨てられる)、またはブラウズ・カーソルが次に進められる (キューをブラウズする場合)。
   - メッセージの実際の長さは *DataLength* に戻される。

   このオプションを指定しなくても、呼び出しはやはり警告を出して完了しますが、メッセージはキューから除去されません (またはブラウズ・カーソルは進みません)。
2. バッファー・サイズを見積もってください (場合によっては、ゼロ・バイトのサイズを指定することもできます)。MQGMO_ACCEPT_TRUNCATED_MSG オプションは使用しないでください。 MQGET 呼び出しが失敗した場合は (例えば、バッファーが小さすぎるために)、呼び出しの **DataLength** パラメーターにメッセージの長さが戻されます (それでも、バッファーには、収容しきれるだけのメッセージを保持していますが、呼び出しの処理は完了しません。) このメッセージの *MsgId* を記録しておき、あとで適切なサイズのバッファー域と最初の呼び出しで記録した *MsgId* を指定して、MQGET をもう一度呼び出してください。

   プログラムが、他のプログラムも使用しているキューを処理する場合は、別の MQGET 呼び出しを出す前に、他のプログラムの 1 つが必要なメッセージを除去してしまうこともあります。 その結果、このメッセージを探しているプログラムは、もはや存在しないメッセージの検索に時間を浪費することになります。 このことを避けるために、*BufferLength* をゼロに指定して MQGMO_ACCEPT_TRUNCATED_MSG オプションを使用し、必要なメッセージを検出するまでまずキューをブラウズします。 これにより、必要とするメッセージの下にブラウズ・カーソルが位置付けられます。 次に、MQGMO_MSG_UNDER_CURSOR オプションを指定して MQGET を再度呼び出すと、そのメッセージを取り出すことができます。 このブラウズ呼び出しと除去呼び出しの間に別のプログラムがそのメッセージを除去した場合は、ブラウズ・カーソルの下にメッセージがないので、2 回目の MQGET は即時に (キュー全体を検索しないで) 失敗します。
3. キューで受け入れられるメッセージの最大長は、そのキューの *MaxMsgLength* キュー 属性によって決定され、キュー・マネージャーで受け入れられるメッセージの最大長は、そのキュー・マネージャーの *MaxMsgLength* キュー・マネージャー 属性によって決定されます。 どんな長さのメッセージ受け取るのか分からない場合は、(MQINQ 呼び出しを使用して) **MaxMsgLength** 属性を照会してから、そのサイズのバッファーを指定してください。

   パフォーマンスが低下するのを避けるため、バッファー・サイズを、できるだけ実際のメッセージ・サイズに近いものにしてください。

   MaxMsgLength 属性の詳細については、792 ページの『最大メッセージ長の増加』を参照してください。

メッセージがキューから取り出される順序

キューからメッセージを取り出す順序を制御できます。 このセクションでは、オプションを確認します

優先順位

プログラムは、メッセージをキューに書き込むときに、優先順位を割り当てることができます (26 ページの『メッセージ優先順位』を参照)。 優先順位の同じメッセージは、コミットされた順ではなく、到着順にキューに保管されます。

キュー・マネージャーは、キューを厳密な FIFO (先入れ先出し) の順序か、優先順位での FIFO の順序で維持します。 これは、キューの **MsgDeliverySequence** 属性の設定値によって決められます。 キューに到着したメッセージは、同じ優先順位をもつ最後のメッセージのすぐあとに挿入されます。

プログラムは、キューから最初のメッセージを読み取ることも、優先順位を無視して特定のメッセージ読み取ることもできます。 例えば、前に送信した特定のメッセージに対する応答をプログラムが処理する場合があります。 詳しくは、785 ページの『特定のメッセージの読み取り』を参照してください。

あるアプリケーションがキューに一連のメッセージを書き込んだ場合、別のアプリケーションは、次の条件が満たされている場合は、それらのメッセージを書き込まれたときと同じ順序で取り出すことができます。

- すべてのメッセージの優先順位が同じである
- メッセージがすべて同じ作業単位内で、またはすべて作業単位外で書き込まれた
- キューは書き込みを行うアプリケーションに対してローカルである

これらの条件が満たされず、しかもある一定の順序で検索されるメッセージにアプリケーションが依存する場合は、アプリケーションはメッセージ・データ内に順序情報を含める必要があります。または、次のメッセージが送信される前に、アプリケーションはメッセージ受信の肯定応答を行う手段を確立しなければなりません。

z/OS IBM MQ for z/OS では、キュー属性 *IndexType* を使用して、キューに対する MQGET 操作の速度を向上させることができます。 詳しくは、791 ページの『索引のタイプ』を参照してください。

論理的な順序付けと物理的な順序付け

キューにあるメッセージの順序には、(優先順位レベルごとの) 物理 順序と論理 順序があります。

物理順序とは、メッセージがキューに到着した順序を指します。 論理順序では、1 つのグループ内のメッセージおよびセグメントのすべてが論理順序に基づいて、互いに隣接して、グループに属する最初の項目の物理位置によって決まる位置に並びます。

グループ、メッセージ、およびセグメントの説明については、44 ページの『メッセージ・グループ』を参照してください。 物理順序と論理順序は、次の理由で異なる場合があります。

- 複数のグループが異なるアプリケーションから 1 つの宛先に同じようなタイミングで到着すると、物理順序の区別がなくなる可能性がある。
- 単一グループの中でも、その中の一部のメッセージの再ルーティングや遅延によってメッセージの順序が乱れることがある。

例えば、775 ページの図 61 に示すような論理順序があるとします。

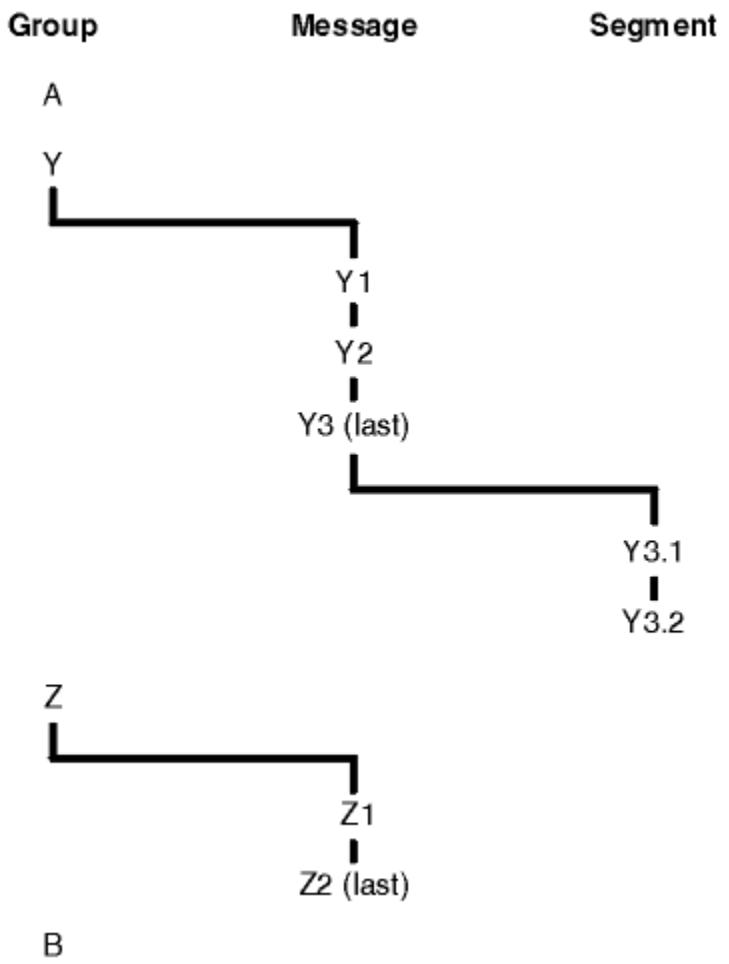


図 *61.* キューでの論理順序

これらのメッセージは、次に示す論理順序でキューに現れることがあります。

1. (グループに属していない) メッセージ A
2. グループ Y の論理メッセージ 1
3. グループ Y の論理メッセージ 2
4. グループ Y の (最後の) 論理メッセージ 3 のセグメント 1
5. グループ Y の (最後の) 論理メッセージ 3 の (最後の) セグメント 2
6. グループ Z の論理メッセージ 1
7. グループ Z の (最後の) 論理メッセージ 2
8. (グループに属していない) メッセージ B

しかし、物理順序はまったく異なる可能性があります。 各グループ内の最初の 項目の物理位置によって、そのグループ全体の論理位置が決まります。 例えば、グループ Y とグループ Z が同じようなタイミングで到着した場合に、グループ Z のメッセージ 2 がメッセージ 1 よりも優先されているとすれば、物理順序は 776 ページの図 62 のようになります。

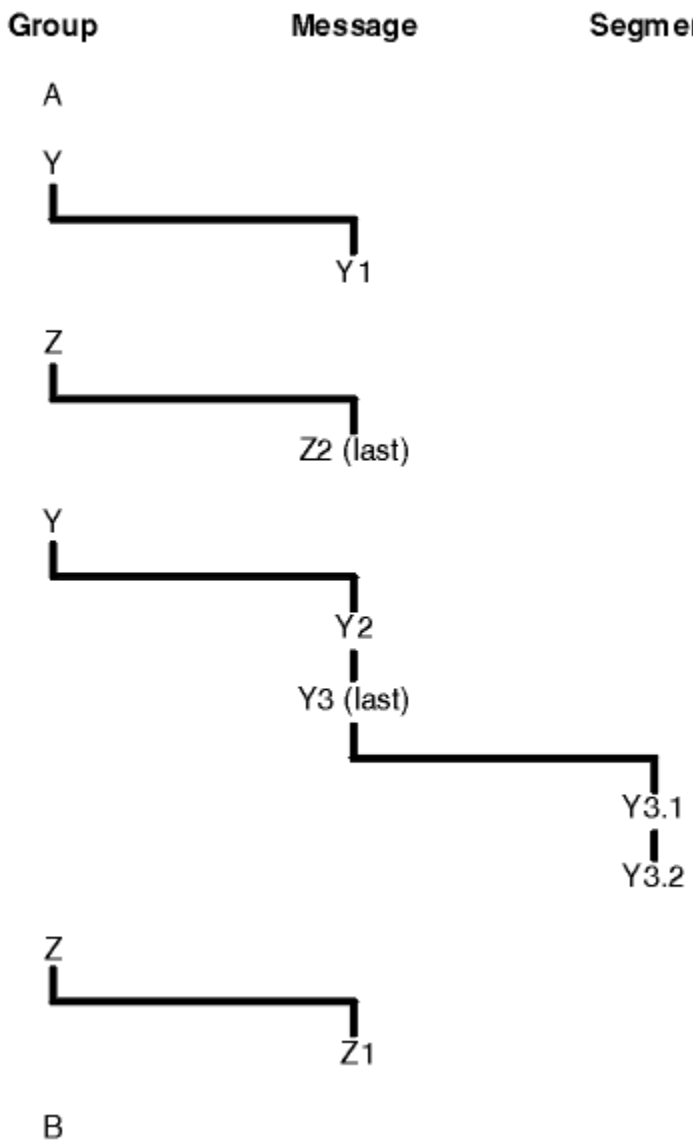


図 *62.* キューでの物理順序

これらのメッセージは、次に示す物理順序でキューに現れることがあります。

1. (グループに属していない) メッセージ A
2. グループ Y の論理メッセージ 1
3. グループ Z の論理メッセージ 2
4. グループ Y の論理メッセージ 2
5. グループ Y の (最後の) 論理メッセージ 3 のセグメント 1
6. グループ Y の (最後の) 論理メッセージ 3 の (最後の) セグメント 2
7. グループ Z の論理メッセージ 1
8. (グループに属していない) メッセージ B

注 : IBM MQ for z/OS では、キューが GROUPID によって索引付けされている場合、キューでのメッセージの物理順序は保証されていません。

メッセージを取得するときに MQGMO_LOGICAL_ORDER を指定することにより、物理順序ではなく論理順序でメッセージを取得できます。

MQGMO_BROWSE_FIRST および MQGMO_LOGICAL_ORDER を指定して MQGET 呼び出しを発行した場合は、 MQGMO_BROWSE_NEXT を指定して後続の MQGET 呼び出しを発行するときにも

MQGMO_LOGICAL_ORDER を指定する必要があります。 逆に、MQGMO_BROWSE_FIRST を指定して MQGET を発行するときに MQGMO_LOGICAL_ORDER を指定しなかった場合は、MQGMO_BROWSE_NEXT を指定して後続の MQGET を発行するときに、このオプションを指定しなければなりません。

MQGET 呼び出しでキューに入っているメッセージをブラウズするためにキュー・マネージャーが保持しているグループおよびセグメント情報は、MQGET 呼び出しでキューからメッセージを除去するためにキュー・マネージャーが保持しているグループおよびセグメント情報とは異なります。
MQGMO_BROWSE_FIRST を指定する場合、キュー・マネージャーはブラウズに関するグループ情報およびセグメント情報を無視して、現行のグループおよび現行の論理メッセージが存在しない場合と同じようにキューを走査します。

注 : MQGMO_LOGICAL_ORDER を指定せずに MQGET 呼び出しを使用することにより、メッセージ・グループ (またはグループに属していない論理メッセージ) の最後を越えて ブラウズすることは避けてください。 例えば、キュー上で、グループの最後のメッセージがグループの最初のメッセージよりも前に ある場合に、(次のグループの最初のメッセージを検出するために) *MsgSeqNumber* を 1 に設定して MQMO_MATCH_MSG_SEQ_NUMBER を指定し、MQGMO_BROWSE_NEXT を使用してグループの最後を越えてブラウズを行うと、既にブラウズしたグループの最初のメッセージに戻ってしまいます。 これは、即時に発生する可能性があります。あるいは、 (介入グループがある場合) 何度かの MQGET 呼び出し後に発生するかもしれません。

無限ループの発生を避けるには、ブラウズの際にキューを *2* 回 オープンして次のように操作します。

- 各グループの最初のメッセージだけをブラウズするには、最初のハンドルを使用します。
- 特定のグループのメッセージだけをブラウズするには、2 番目のハンドルを使用します。
- 特定のグループの各メッセージをブラウズする前に、MQMO_* オプションを使用して 2 番目のブラウズ・カーソルを最初のブラウズ・カーソルの位置まで移動します。
- MQGMO_BROWSE_NEXT によるグループの最後よりあとのブラウズは実行しないようにします。

これについての詳細は、MQGET、MQMD、 および MQI オプションの妥当性検査に関する規則を参照してください。

アプリケーションでブラウズを実行するときには、一般に、論理順序と物理順序のどちらか一方を選択します。 一方、この 2 つのモードを切り替えて使用する場合には、最初に MQGMO_LOGICAL_ORDER を指定してブラウズを発行すると論理順序による位置が設定されるので注意してください。

その時点でグループ内の最初の項目が存在しなければ、そのグループは論理順序の一部としては扱われません。

あるグループ内にいったんブラウズ・カーソルが置かれると、グループ内の最初のメッセージが除去されたとしても、そのブラウズ・カーソルは引き続き同じグループを指示します。 ただし、
MQGMO_LOGICAL_ORDER を使用してブラウズを発行した時点で最初の項目が存在しないグループにカーソルを移動することはできません。

MQPMO_LOGICAL_ORDER

MQPMO オプションは、アプリケーションがメッセージを論理メッセージのグループおよびセグメントに書き込む方法をキュー・マネージャーに通知します。 このオプションは、MQPUT 呼び出しでのみ指定できます。MQPUT1 呼び出しでは無効です。

MQPMO_LOGICAL_ORDER が指定されると、アプリケーションは後続の MQPUT 呼び出しを使用して次のことを行います。

1. 各論理メッセージ内のセグメントを、0 からセグメント・オフセットの小さい順に間を空けずに書き込む。
2. 論理メッセージ内のセグメントをすべて書き込んでから、その次の論理メッセージのセグメントを書き込む。
3. 各メッセージ・グループ内の論理メッセージを、1 からメッセージ順序番号の小さい順に間を空けずに書き込む。 IBM MQ はメッセージ・シーケンス番号を自動的にインクリメントします。
4. メッセージ・グループ内の論理メッセージをすべて書き込んでから、その次のメッセージ・グループの論理メッセージを書き込む。

アプリケーションはキュー・マネージャーにグループ内のメッセージと論理メッセージのセグメントを書き込む方法を指示したので、アプリケーションにおいて、MQPUT を呼び出すたびにグループの情報やセグメントの情報を維持および更新する必要はありません。その情報はキュー・マネージャーが維持および更新するからです。 具体的には、アプリケーションで MQMD の *GroupId*、*MsgSeqNumber*、および *Offset* フィールドを設定する必要がないということです。キュー・マネージャーがこれらのフィールドに適切な値を設定するからです。 アプリケーションが設定する必要があるのは、MQMD 内の *MsgFlags* フィールドだけです。これによりメッセージがグループに属している場合や、メッセージが論理メッセージのセグメントである場合を指示したり、グループ内の最後のメッセージまたは論理メッセージの最後のセグメントを指示したりできます。

メッセージ・グループまたは論理メッセージの開始後に MQPUT を呼び出すときは MQMD 中の *MsgFlags* にある適切な MQMF_* フラグを指定しなければなりません。 アプリケーションが、終了していないメッセージ・グループがある場合にグループ内にないメッセージを書き込もうとしたり、終了していない論理メッセージがある場合にセグメントではないメッセージを書き込もうとしたりすると、その呼び出しは失敗します。 また、状況に応じて理由コード MQRC_INCOMPLETE_GROUP または MQRC_INCOMPLETE_MSG が表示されます。 ただし、キュー・マネージャーは現在のメッセージ・グループまたは現在の論理メッセージの情報あるいはその両方を保存し、アプリケーションはメッセージを送信してこれらを終了します (アプリケーション・メッセージ・データなしも可能です)。このメッセージでは状況に応じて MQMF_LAST_MSG_IN_GROUP または MQMF_LAST_SEGMENT あるいはその両方を指定します。その後 MQPUT 呼び出しを再発行して、グループまたはセグメントにないメッセージを書き込みます。

776 ページの図 62 に、オプションとフラグの有効な組み合わせを示します。また、キュー・マネージャーがそれぞれの場合に応じて使用する、*GroupId*、*MsgSeqNumber*、および *Offset* の各フィールドの値を示します。 この表に示されていないオプションとフラグの組み合わせは無効です。 この表の列では、「どちらも」は「はい」または「いいえ」のどちらかを意味します。

LOG ORD
MQPMO_LOGICAL_ORDER オプションが呼び出しで指定されるかどうか。

MIG
MQMF_MSG_IN_GROUP または MQMF_LAST_MSG_IN_GROUP オプションが呼び出しで指定されるかどうか。

SEG
MQMF_SEGMENT または MQMF_LAST_SEGMENT オプションが呼び出しで指定されるかどうか。

SEG OK
MQMF_SEGMENTATION_ALLOWED オプションが呼び出しで指定されるかどうか。

Cur grp
現行のメッセージ・グループが呼び出しの前に存在するかどうか。

Cur log msg
現行の論理メッセージが呼び出しの前に存在するかどうかを示します。

その他の列
キュー・マネージャーが使用する値を示しています。「前の」という表現は、キュー・ハンドルに対して前のメッセージのフィールドで使用された値を示します。

表 116. 論理メッセージのグループおよびセグメントに関連した *MQPUT* オプション

指定するオプション	指定するオプション	指定するオプション	指定するオプション	呼び出しの前のグループおよび論理メッセージの状況	呼び出しの前のグループおよび論理メッセージの状況	キュー・マネージャーが使用する値	キュー・マネージャーが使用する値	キュー・マネージャーが使用する値
LOGORD	MIG	SEG	SEGOK	Curgrp	Curlogmsg	**GroupId**	**MsgSeqNumber**	**Offset**
はい	いいえ	いいえ	いいえ	いいえ	いいえ	MQGI_NONE	1	0
はい	いいえ	いいえ	はい	いいえ	いいえ	新しいグループ ID	1	0
はい	いいえ	はい	どちらも	いいえ	いいえ	新しいグループ ID	1	0
はい	いいえ	はい	どちらも	いいえ	はい	前のグループ ID	1	前のオフセット + 前のセグメント長
はい	はい	どちらも	どちらも	いいえ	いいえ	新しいグループ ID	1	0
はい	はい	どちらも	どちらも	はい	いいえ	前のグループ ID	前の順序番号 + 1	0
はい	はい	はい	どちらも	はい	はい	前のグループ ID	前の順序番号	前のオフセット + 前のセグメント長
いいえ	いいえ	いいえ	いいえ	どちらも	どちらも	MQGI_NONE	1	0
いいえ	いいえ	いいえ	はい	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	1	0
いいえ	いいえ	はい	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	1	フィールド内の値
いいえ	はい	いいえ	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	フィールド内の値	0
いいえ	はい	はい	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	フィールド内の値	フィールド内の値

注:

- MQPUT1 呼び出しでは、MQPMO_LOGICAL_ORDER は無効です。
- *MsgId* フィールドについては、MQPMO_NEW_MSG_ID または MQMI_NONE が指定されるとキュー・マネージャーは新しいメッセージ ID を生成し、それ以外の場合はフィールドの値を使用します。

- *CorrelId* フィールドについては、MQPMO_NEW_CORREL_ID が指定されるとキュー・マネージャーは新しい相関 ID を生成し、それ以外の場合はフィールドの値を使用します。

MQPMO_LOGICAL_ORDER を指定する場合、キュー・マネージャーでは、グループ内のすべてのメッセージおよび論理メッセージのセグメントを MQMD の *Persistence* フィールドに同じ値で書き込むことが必要です。 つまり、全部を持続にするか、または全部を非持続にする必要があります。 この条件が満たされないと、MQPUT 呼び出しは失敗し、理由コード MQRC_INCONSISTENT_PERSISTENCE が戻ります。

MQPMO_LOGICAL_ORDER オプションが作業単位に及ぼす影響は、以下のとおりです。

- グループ内または論理メッセージ内の最初の物理メッセージが 1 つの作業単位に書き込まれた場合、そのグループ内または論理メッセージ内の他の物理メッセージも、同じキュー・ハンドルが使用されていれば、すべて 1 つの作業単位に書き込む必要があります。 ただし、必ずしも同じ作業単位内に書き込む必要はありません。多数の物理メッセージから成るメッセージ・グループまたは論理メッセージを、キュー・ハンドルに対する 2 つ以上の連続した作業単位にまたがって分割することができます。
- グループ内または論理メッセージ内の最初の物理メッセージが 1 つの作業単位に書き込まれていない場合、同じキュー・ハンドルが使用されていれば、そのグループ内または論理メッセージ内の他の物理メッセージはどれも 1 つの作業単位に書き込むことができません。

これらの条件が満たされないと、MQPUT 呼び出しは失敗し、理由コード MQRC_INCONSISTENT_UOW が戻ります。

MQPMO_LOGICAL_ORDER が指定されている場合、MQPUT 呼び出しで提供される MQMD は、MQMD_VERSION_2 未満であってはいけません。 この状態が満たされない場合、呼び出しは失敗し、理由コード MQRC_WRONG_MD_VERSION が表示されます。

MQPMO_LOGICAL_ORDER を指定しないと、グループ内のメッセージおよび論理メッセージ内のセグメントは任意の順序で書き込まれます。また、必ずしも完全なメッセージ・グループおよび完全な論理メッセージを書き込む必要はありません。 *GroupId*、*MsgSeqNumber*、*Offset*、および *MsgFlags* フィールドが適切な値を持つようにするのは、アプリケーションの責任です。

システム障害が発生した後には、この手法を用いてメッセージ・グループまたは論理メッセージを途中から再始動します。 システムが再始動したら、アプリケーションで *GroupId*、*MsgSeqNumber*、*Offset*、*MsgFlags*、 および *Persistence* の各フィールドに適切な値を設定した後、必要に応じて MQPMO_SYNCPOINT または MQPMO_NO_SYNCPOINT を設定して MQPUT 呼び出しを発行できます。そのとき、MQPMO_LOGICAL_ORDER は指定しません。 この呼び出しが成功した場合、キュー・マネージャーはグループとセグメントの情報を保存し、キュー・ハンドルに対する後続の MQPUT 呼び出しで通常どおり MQPMO_LOGICAL_ORDER を指定できます。

MQPUT 呼び出しのためにキュー・マネージャーが保持しているグループおよびセグメント情報は、MQGET 呼び出しのためにキュー・マネージャーが保持しているグループおよびセグメント情報とは異なります。

キュー・ハンドルが指定されている場合には、アプリケーションでは、MQPMO_LOGICAL_ORDER を指定した MQPUT 呼び出しと MQPMO_LOGICAL_ORDER を指定していない MQPUT 呼び出しを組み合わせて使用できます。ただし、以下の点に注意してください。

- MQPMO_LOGICAL_ORDER を指定していない場合、MQPUT 呼び出しが成功するたびに、キュー・マネージャーは、キュー・ハンドルのグループおよびセグメント情報を、アプリケーションが指定する値に設定します。したがって、そのキュー・ハンドルに対してキュー・マネージャーで保持されていた既存のグループおよびセグメント情報がその値で置換されます。
- MQPMO_LOGICAL_ORDER を指定していない場合、現行のメッセージ・グループまたは論理メッセージが存在していても、呼び出しは失敗しません。呼び出しが成功しても MQCC_WARNING 完了コードが出される場合があります。 781 ページの表 117 に、発生する可能性のあるいくつかのケースを示しています。 これらの場合に、完了コードが MQCC_OK 以外であれば、理由コードは以下のいずれか (該当するもの) になります。
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG

– MQRC_INCONSISTENT_PERSISTENCE
– MQRC_INCONSISTENT_UOW

注 : キュー・マネージャーは、MQPUT1 呼び出しのためにグループ情報およびセグメント情報を確認しません。

表 *117.* グループおよびセグメント情報と整合性のない *MQPUT* または *MQCLOSE* 呼び出しの結果

現行の呼び出し	前の呼び出しが MQPMO_LOGICAL_ORDER を指定した MQPUT	前の呼び出しが MQPMO_LOGICAL_ORDER を指定しない MQPUT
MQPMO_LOGICAL_ORDER を指定した MQPUT	MQCC_FAILED	MQCC_FAILED
MQPMO_LOGICAL_ORDER を指定しない MQPUT	MQCC_WARNING	MQCC_OK
終了していないグループまたは論理メッセージを指定している MQCLOSE	MQCC_WARNING	MQCC_OK

メッセージおよびセグメントを論理順序で書き込むアプリケーションでは、最も簡単に使えるオプションとして MQPMO_LOGICAL_ORDER を指定します。 このオプションを指定すると、キュー・マネージャーがグループおよびセグメント情報を管理するので、アプリケーションでこの情報を管理する必要はなくなります。 しかし、特殊なアプリケーションでは、MQPMO_LOGICAL_ORDER オプションによって提供される以上の制御が必要な場合があります。その場合、MQPMO_LOGICAL_ORDER オプションを指定しないことができます。ただし、MQMD 内の *GroupId*、*MsgSeqNumber*、*Offset*、 および *MsgFlags* の各フィールドが正しく設定されていることを、各 MQPUT または MQPUT1 呼び出しの前に確認する必要があります。

例えば、受信した物理メッセージがグループに属していなくても、また論理メッセージのセグメントでなくても、そのメッセージを転送するアプリケーションでは、MQPMO_LOGICAL_ORDER を指定してはなりません。これには次の 2 つの理由があります。

- メッセージを取得して順番に書き込む場合、MQPMO_LOGICAL_ORDER を指定するとメッセージに新しいグループ ID が割り当てられます。こうなると、メッセージの発信元は、メッセージ・グループに対応する応答メッセージまたはレポート・メッセージとの相関をとることが困難になるか、不可能になります。
- 送信側のキュー・マネージャーと受信側のキュー・マネージャーの間にパスが複数あるような複雑なネットワークの場合には、物理メッセージが正しくない順序で到達することがあります。 MQGET 呼び出しに MQPMO_LOGICAL_ORDER と MQGMO_LOGICAL_ORDER をどちらも指定しなければ、転送側のアプリケーションでは、論理順序で次にあるメッセージを待たずに、それぞれの物理メッセージを到着と同時に取得して転送することができます。

グループ内のメッセージまたは論理メッセージのセグメントに関するレポート・メッセージを生成するアプリケーションでも、レポート・メッセージを書き込むときには MQPMO_LOGICAL_ORDER を指定してはなりません。

MQPMO_LOGICAL_ORDER はその他のどの MQPMO_* オプションとも組み合わせて指定できます。

論理的に順序付けされたグループのクラスター・キューへの書き込み (MQOO_BIND_ON_GROUP)

MQOO_BIND_ON_OPEN オプションは、このアプリケーションから出るすべてのメッセージ (したがって、すべてのグループ) が単一インスタンスに送付されることを保証します。 これには、アプリケーション・トラフィックがクラスター・キューの複数インスタンスにわたってロード・バランシングされないという欠点があります。 メッセージのグループをそのまま保持しているときにワークロード・バランシングを有効化するには、以下のオプションを設定する必要があります。

- MQPUT 呼び出しは MQPMO_LOGICAL_ORDER を指定する必要がある

- MQOPEN 呼び出しは、以下の 2 つのオプションのいずれかを指定する必要があります。
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF。さらに、キュー定義で DEFBIND(GROUP) を指定する必要があります。

こうすると、ワークロード・バランシングはメッセージのグループとグループの間 で行われ、キューの MQCLOSE と MQOPEN は必要なくなります。 グループとグループの間 ということは、MQMF_MSG_IN_GROUP が MQMD(v2) または MQMDE に設定されて、進行中のグループが部分的に完了することはないという意味です。 グループが進行中のときは、オブジェクト・ハンドルに解決されたキュー・マネージャーおよびキュー名が再利用されます。

前のメッセージが MQPMO_LOGICAL_ORDER または MQMF_MSG_IN_GROUP であったものの、現在のメッセージがグループの一部ではない場合、PUT 呼び出しは MQRC_INCOMPLETE_GROUP で失敗します。

個々の MQPUT に MQPMO_LOGICAL_ORDER が指定されていない場合、現在アクティブなグループがなければ、そのメッセージに対してワークロード・バランシングが実行されます (MQOPEN 呼び出しに MQOO_BIND_NOT_FIXED が指定されていた場合のように)。

MQOO_BIND_ON_GROUP を使用して宛先にバインドされたメッセージについては、再割り振りは実行されません。 再割り振りの詳細については、44 ページの『メッセージ・グループ』を参照してください。

論理メッセージのグループ化

論理メッセージをグループにまとめて使用するのは、次の 2 つの主な理由によります。

- メッセージは、特定の順番で処理しなければならないことがあります。
- グループ内の各メッセージは、関連した方法で処理しなければならないことがあります。

いずれの場合も、グループ全体を同じ読み取り側アプリケーション・インスタンスを使用して取り出します。

例えば、4 つの論理メッセージで構成されるグループがあるとします。 メッセージの書き込み側アプリケーションでは、次のような操作を行います。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

読み取り側アプリケーションは、グループ内の最初のメッセージに MQGMO_ALL_MSGS_AVAILABLE オプションを指定します。 これにより、グループ内のすべてのメッセージが到着して初めて処理が開始されるようになります。 MQGMO_ALL_MSGS_AVAILABLE オプションは、グループ内の後続のメッセージでは無視されます。

グループの最初の論理メッセージが取り出されたときに、MQGMO_LOGICAL_ORDER を使用して、グループの残りの論理メッセージが順序どおりに取り出されるようにすることができます。

そこで、読み取り側アプリケーションでは、次のような操作を行います。

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

メッセージのグループ化の詳しい例については、795 ページの『アプリケーションによる論理メッセージのセグメント化』および 783 ページの『複数の作業単位にわたるグループの書き込みと読み取り』を参照してください。

重要 : パブリッシュ/サブスクライブを使用してメッセージをトピックに送信する (またはメッセージをトピック別名に書き込む) 場合、メッセージのグループ化とセグメンテーションは許可されません。

サブスクリプションはパブリケーション・アクティビティーとは独立して作成および削除できるため、サブスクライバーがメッセージ・グループ全体またはメッセージのすべてのセグメントを受け取ることは保証できません。 RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP を参照してください。

メッセージのグループをすべてクラスター・キューの同じ宛先インスタンスに割り振るようにアプリケーションが要求できるようにする方法についての詳細は、DefBind を参照してください。

複数の作業単位にわたるグループの書き込みと読み取り

前出の例では、グループ全体が書き込まれ、作業単位がコミットされるまで、メッセージやセグメントをノードから送信したり (宛先がリモートの場合)、取り出したりする処理を開始することはできません。 このため、グループ全体を書き込むのに長時間かかる場合や、ノード上でのキューのスペースが限られている場合には、不都合が生じることがあります。 これを解決するには、グループをいくつかの作業単位に分けて書き込みます。

グループを複数の作業単位内で書き込む場合には、書き込み側アプリケーションで障害が発生したときでも、グループの一部がコミットされる可能性があります。 このため、アプリケーションでは、各作業単位でコミットされた状況に関する情報を保管し、再始動後にその情報を使用して不完全なグループの処理を再開できるようにしておく必要があります。 この情報を記録するのに最も便利な場所は状況 (STATUS) キューです。 1 つのグループ全体が正常に書き込まれると、状況キューは空になります。

セグメント化を伴う場合も、論理は同様です。 この場合、**StatusInfo** には *Offset* を入れる必要があります。

複数の作業単位にわたってグループを書き込む場合の記述例を次に示します。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

すべての作業単位がコミットされると、グループ全体が正常に書き込まれ、状況キューが空になります。 コミットされない作業単位がある場合は、状況に関する情報で指示されている個所からグループの処理を再開する必要があります。 最初の書き込みでは MQPMO_LOGICAL_ORDER を使用できませんが、以後の書き込みでは使用できます。

再始動処理の例を示します。

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...
```

```
else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

読み取り側アプリケーションでは、グループ全体が到着する前に、グループ内のメッセージの処理を開始したい場合があります。 これによって、グループ内のメッセージに関する応答時間が短縮されると共に、グループ全体を格納するストレージも必要なくなります。 このような利点を実現するために、メッセージ・グループごとにいくつかの作業単位を使用します。 リカバリー上の理由から、それぞれのメッセージは 1 つの作業単位内で取り出す必要があります。

そのためには、対応する書き込み側アプリケーションと同様に読み取り側アプリケーションでも、それぞれの作業単位がコミットされたときに、状況に関する情報がどこかに自動的に記録されなければなりません。 この場合も、この情報を記録するのに最も便利な場所は状況キューです。 1 つのグループ全体が正常に処理されると、状況キューは空になります。

注 : 中間作業単位については、作業単位ごとに 1 つの新しいメッセージ全体を書き込む代わりに、状況キューに対応するそれぞれの MQPUT でメッセージの 1 つのセグメントを書き込むように指定する (つまり、MQMF_SEGMENT フラグを設定する) ことによって、状況キューに対する MQGET を避けることができます。 最後の作業単位では、MQMF_LAST_SEGMENT を指定して最終セグメントを状況キューに書き込んだ上で、MQGMO_COMPLETE_MSG を指定した MQGET によって状況に関する情報をクリアします。

再始動処理では、単一の MQGET を使用して状況メッセージを読み取る代わりに、MQGMO_LOGICAL_ORDER によって、最後のメッセージに達するまで (つまり、それ以上セグメントが戻されなくなるまで) 状況キューをブラウズします。 再始動後の作業単位では、状況セグメントを書き込むときにオフセットを明示的に指定します。

以下の例では、アプリケーションのバッファーに、メッセージがセグメント化されているかどうかに関係なく、常にメッセージ全体を保持できるだけの大きさがあることを前提として、グループ内のメッセージのみについて考えます。 このため、それぞれの MQGET で MQGMO_COMPLETE_MSG を指定しています。 セグメント化を伴う場合も同様にします (この場合には、StatusInfo に *Offset* を入れる必要があります)。

分かりやすくするために、1 つの作業単位で最大 4 つのメッセージが取り出されるものとします。

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
```

```
      /* Come here if there was only 1 message in the group */
      MQCMIT
```

すべての作業単位がコミットされると、グループ全体が正常に取り出され、状況キューが空になります。 コミットされない作業単位がある場合は、状況に関する情報で指示されている個所からグループの処理を再開する必要があります。 最初の取り出しでは MQGMO_LOGICAL_ORDER を使用できませんが、以後の取り出しでは使用できます。

再始動処理の例を示します。

```
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if (Reason == MQRC_NO_MSG_AVAILABLE)
      /* Proceed to normal processing */
      ...

   else
      /* Group was terminated prematurely */
      /* The next message on the group must be retrieved by matching
         the sequence number and group ID with those retrieved from the
         status information. */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
            MQMD.GroupId      = value from Status message,
            MQMD.MsgSeqNumber = value from Status message plus 1
      msgs = 1
      /* Process this message */
      ...

      /* Now normal processing is resumed */
      /* Retrieve remaining messages in the group */
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )

         /* Process up to 4 messages in the group */
         GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                     | MQGMO_LOGICAL_ORDER
         do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
            MQGET
            msgs = msgs + 1
            /* Process this message */
            ...

         /* Have retrieved last message or 4 messages  */
         /* Update status message if not last in group */
         MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
         if ( GroupStatus == MQGS_MSG_IN_GROUP )
            StatusInfo = GroupId,MsgSeqNumber from MQMD
            MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
         MQCMIT
         msgs = 0
```

特定のメッセージの読み取り

キューから特定のメッセージを読み取る方法はいくつも あります。 MsgId および CorrelId での選択、GroupId、MsgSeqNumber、および Offset での選択、MsgToken での選択が可能です。 キューをオープンするときに選択ストリングを使用することもできます。

キューから特定のメッセージを読み取るには、MQMD 構造体の MsgId および CorrelId フィールドを使用してください。 ただし、アプリケーションは明示的にこれらのフィールドを設定することがあるので、指定する値が固有のメッセージを識別しない場合もあります。 これらのフィールドに設定可能な値に対して、検索されるメッセージを 785 ページの表 118 に示します。 MQGET 呼び出しの **GetMsgOpts** パラメーターに MQGMO_MSG_UNDER_CURSOR を指定した場合は、これらのフィールドは入力時に無視されます。

表 *118.* メッセージ *ID* と相関 *ID* の使用方法

検索するメッセージ	**MsgId**	**CorrelId**
キューの最初のメッセージ	MQMI_NONE	MQCI_NONE
MsgId に一致する最初のメッセージ	非ゼロ	MQCI_NONE
CorrelId に一致する最初のメッセージ	MQMI_NONE	非ゼロ

表 118. メッセージ *ID* と相関 *ID* の使用方法 (続き)		
検索するメッセージ	**MsgId**	**CorrelId**
MsgId および *CorrelId* に一致する最初のメッセージ	非ゼロ	非ゼロ

上記のいずれの場合も、最初 とは、選択基準を満たす最初のメッセージを意味します (ただし、MQGMO_BROWSE_NEXT が指定されている場合は別で、このときは選択基準を満たすメッセージ列上の次の メッセージを意味します)。

MQGET 呼び出しは、戻るときに *MsgId* フィールドと *CorrelId* フィールドに、戻されたメッセージ (存在する場合) のメッセージ ID と相関 ID を設定します。

MQMD 構造体の *Version* フィールドに 2 を設定すると、*GroupId*、*MsgSeqNumber*、および *Offset* の各フィールドが使用できます。 これらのフィールドに設定可能な値に対して、検索されるメッセージを 786 ページの表 119 に示します。

表 119. グループ *ID* の使用方法	
検索するメッセージ	**一致オプション**
キューの最初のメッセージ	MQMO_NONE
MsgId に一致する最初のメッセージ	MQMO_MATCH_MSG_ID
CorrelId に一致する最初のメッセージ	MQMO_MATCH_CORREL_ID
GroupId に一致する最初のメッセージ	MQMO_MATCH_GROUP_ID
MsgSeqNumber に一致する最初のメッセージ	MQMO_MATCH_MSG_SEQ_NUMBER
MsgToken に一致する最初のメッセージ	MQMO_MATCH_MSG_TOKEN
Offset に一致する最初のメッセージ	MQMO_MATCH_OFFSET

注:

1. MQMO_MATCH_XXX は、MQMD 構造体の *XXX* フィールドが一致する値に設定されていることを示します。
2. MQMO フラグをそれぞれ組み合わせて使用できます。 例えば、MQMO_MATCH_GROUP_ID、MQMO_MATCH_MSG_SEQ_NUMBER、および MQMO_MATCH_OFFSET を共に使用すると、GroupId、MsgSeqNumber、および Offset の各フィールドによって識別されるセグメントを指定できます。
3. MQGMO_LOGICAL_ORDER は、キュー・ハンドルに応じて制御される状態情報によって異なるので、MQGMO_LOGICAL_ORDER を指定した場合には、取り出されるメッセージが変わることがあります。 詳細については、774 ページの『論理的な順序付けと物理的な順序付け』 および Options を参照してください。

MQGET 呼び出しでは、通常、キューから最初のメッセージを検索します。 MQGET 呼び出しを使用するときに特定のメッセージを指定すると、キュー・マネージャーはそのメッセージを見つけるまでキューを検索しなければなりません。 これは、アプリケーションのパフォーマンスに影響を及ぼす可能性があります。

バージョン 2 以降の MQGMO 構造体を使用している場合で、MQMO_MATCH_MSG_ID または MQMO_MATCH_CORREL_ID フラグを指定していない場合は、MQGET ごとに MsgId フィールドや CorrelId フィールドをリセットする必要はありません。

z/OS IBM MQ for z/OS では、キュー属性 IndexType を使用して、キューに対する MQGET 操作の速度を向上させることができます。 詳しくは、791 ページの『索引のタイプ』 を参照してください。

特定のメッセージを、そのメッセージの MsgToken と、MatchOption MQMO_MATCH_MSG_TOKEN を MQGMO 構造体で 指定することによって、キューから読み取ることができます。 MsgToken は、そのメッセージを最初にキューに書き込んだ MQPUT 呼び出しに よって戻されるか、直前の MQGET 操作によって戻され、キュー・マネージャーが再始動 されない限り一定のままです。

キューにあるメッセージのうちの一部のメッセージにのみ関心がある場合、 MQOPEN または MQSUB 呼び出しで選択ストリングを使用することによって、 処理したいメッセージを指定することができます。 そうすると、MQGET は、その選択ストリングを 満足する次のメッセージを取り出します。 選択ストリングについて 詳しくは、31 ページの『Selectors』を参照してください。

非永続メッセージのパフォーマンス向上

クライアントは、サーバーからのメッセージを必要とする際、サーバーに向けて要求を送信します。 クライアントは、コンシュームするメッセージごとに要求を別々に送信します。 こうした要求メッセージの送信を省くことにより、非永続メッセージをコンシュームするクライアントのパフォーマンスを向上するには、先読み を使用するようにクライアントを構成します。 先読みにより、アプリケーションからの要求がなくてもクライアントへのメッセージ送信が可能となります。

先読みが使用可能な場合、メッセージはクライアント上の先読みバッファー というメモリーのバッファーに送信されます。 クライアントには、先読みが使用可能でオープンされた各キューごとに先読みバッファーがあります。 先読みバッファー内のメッセージは非永続メッセージです。 クライアントは定期的に、消費したデータ量に関する更新情報をサーバーに提供します。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、IBM MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアント・アプリケーションは、スレッド化された IBM MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みを使用すると、クライアント・アプリケーションから非永続メッセージをコンシュームする際のパフォーマンスを改善することができます。 このパフォーマンスの改善は、MQI アプリケーションと JMS アプリケーションの両方で有効です。 MQGET または非同期コンシュームを使用するクライアント・アプリケーションでは、非永続メッセージをコンシュームするときにパフォーマンス向上の効果が得られます。

クライアント・アプリケーションの設計によっては、先読みの使用が適していない場合があります。なぜなら、先読みの使用はすべてのオプションによってサポートされているわけではなく、またオプションによっては先読みの使用可能時に MQGET 呼び出し間で一貫性の確保が求められるためです。 クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納されているメッセージはそのクライアントの先読みバッファー内に保留されます。

変更前の選択基準で保留されたメッセージのバックログが不要になる場合、構成可能なパージ間隔をクライアント上で設定し、これらのメッセージをクライアントから自動的にパージすることができます。 パージ間隔は、クライアントによって決定される一連の先読みチューニング・オプションの 1 つです。 これらのオプションは、要件に合わせて調整することができます。

クライアント・アプリケーションが再始動されると、先読みバッファー内のメッセージは失われます。 逆に、先読みバッファーに移動されたメッセージが、その後に基礎となっているキューから削除されることもあります。これはバッファーからメッセージが除去される結果にはならないため、先読みバッファーを使用する MQGET 呼び出しは、もう存在していないメッセージを戻す可能性があります。

先読みはクライアントのバインディング用にのみ実行されます。 この属性は、他のすべてのバインディングでは無視されます。

先読みはトリガーに影響を及ぼしません。 メッセージがクライアントによって先読みされる際、トリガー・メッセージは生成されません。 先読みの有効時、アカウントおよび統計情報は先読みによって生成されません。 enabled.

パブリッシュ・サブスクライブ・メッセージングでの先読みの使用

サブスクライブ・アプリケーションがパブリケーションの送信先である宛先キューを指定すると、指定されたキューの DEFREADA 値が先読みのデフォルト値として使用されます。

サブスクライブ・アプリケーションが IBM MQ によるパブリケーションの送信先の管理を要求する場合、定義済みのモデル・キューに基づく動的キューとして管理キューが作成されます。 先読みのデフォルト値として使用されるのは、モデル・キューの DEFREADA 値です。 モデル・キューが、このトピックまたは親

トピックのために定義されていない限り、デフォルトのモデル・キューである SYSTEM.DURABLE.PUBLICATIONS.MODEL または SYSTEM.NONDURABLE.PUBLICATIONS.MODEL が使用されます。

関連概念

790 ページの『AIX 上の非持続メッセージのためのパフォーマンスの調整』
AIX V5.3 以降を使用している場合は、非持続メッセージのパフォーマンスを最大限に使用するように調整パラメーターを設定することを検討してください。

関連タスク

789 ページの『先読みの使用可能化および使用不能化』
デフォルトでは、先読みは使用不可に設定されています。 キューまたはアプリケーションのレベルで、先読みを使用可能に設定することができます。

関連資料

788 ページの『MQGET オプションと先読み』
先読みが使用可能になっている場合に、すべての MQGET オプションがサポートされるわけではありません。オプションによっては MQGET 呼び出し間の一貫性が求められます。

MQGET オプションと先読み

先読みが使用可能になっている場合に、すべての MQGET オプションがサポートされるわけではありません。オプションによっては MQGET 呼び出し間の一貫性が求められます。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、IBM MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアント・アプリケーションは、スレッド化された IBM MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

以下の表に、先読みの使用をサポートしているオプションおよび MQGET 呼び出し間の変更の可否を示します。

表 *120. MQGET* オプションと先読み

MQGET 値とオプション	**先読みの有効時に使用可能で、MQGET 呼び出し間で変更可能** [5]	**先読みが有効になっている場合に使用でき、MQGET 呼び出し間で変更できない** [1]	**先読みの有効時に使用不可の MQGET オプション** [2]
MQGET MQMD 値	MsgId [3] CorrelId [3]	Encoding CodedCharSetId	
MQGET MQGMO オプション	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED _MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

注 :

1. これらのオプションが MQGET 呼び出し間で変更された場合、MQRC_OPTIONS_CHANGED 理由コードが戻されます。

2. これらのオプションが最初の MQGET 呼び出しで指定されると、先読みは使用不可になります。 これらのオプションを後続の MQGET 呼び出しで指定すると、理由コード MQRC_OPTIONS_ERROR が戻されます。
3. クライアント・アプリケーションが MQGET 呼び出し間で MsgId および CorrelId の値を変更した場合、変更前の値を持つメッセージがクライアントに送信済みの可能性があり、コンシューム (または自動的にパージ) されるまでクライアントの先読みバッファー内に残されます。
4. MQGMO_MSG_UNDER_CURSOR は先読みでは使用できません。 キューのオープン時に、MQOO_BROWSE オプションと、MQOO_INPUT_SHARED オプションまたは MQOO_INPUT_EXCLUSIVE オプションの一方が指定されると、先読みは使用不可になります。
5. 先読みが使用可能に設定されている場合、最初の MQGET によって、 メッセージが参照されるのか、キューから取得されるのかが判別されます。 その後に、クライアント・アプリケーションが変更されたオプションで MQGET を使用すると (例えば、メッセージを最初に取得した後に参照しようとしたり、最初に参照した後に取得しようとしたりする場合)、MQRC_OPTIONS_CHANGED 理由コードが返されます。

クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納された、最初の選択基準に合致するメッセージは、クライアント・アプリケーションによってコンシュームされず、クライアントの先読みバッファー内で保留されます。 クライアントの先読みバッファーに多数の保留メッセージが格納されている場合、先読みによって得られる利点は失われ、コンシュームされるメッセージごとにサーバーへの要求を個別に行う必要があります。 先読みが効率的に使用されているかどうかを見極めるには、接続状況パラメーターである READA を使用します。

最初の MQGET 呼び出し時に不適合なオプションが指定された場合、アプリケーションからの要求により先読みを使用禁止にすることができます。 その状態になると、接続状況は先読み使用禁止中となります。

MQGET のこれらの制約事項のために、クライアント・アプリケーションの設計が先読みに適していないと判断した場合は、MQOPEN オプション MQOO_READ_AHEAD_NO を指定してください。 あるいは、開かれているキューのデフォルトの先読み値を NO または DISABLED に設定します。

先読みの使用可能化および使用不能化
デフォルトでは、先読みは使用不可に設定されています。 キューまたはアプリケーションのレベルで、先読みを使用可能に設定することができます。

このタスクについて

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、IBM MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアント・アプリケーションは、スレッド化された IBM MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みを使用可能に設定するには、以下のようにします。

- 先読みをキューのレベルで構成するには、キュー属性の DEFREADA を YES に設定します。
- 先読みをアプリケーションのレベルで構成するには、以下のようにします。
  - 可能な限りいつでも先読みを使用できるようにするには、MQOPEN 関数呼び出しで MQOO_READ_AHEAD オプションを使用します。 DEFREADA キュー属性が DISABLED に設定されている場合、クライアント・アプリケーションは先読みを使用できなくなります。
  - キューで先読みが使用可能である場合のみ先読みを使用するには、MQOPEN 関数呼び出しで MQOO_READ_AHEAD_AS_Q_DEF オプションを使用します。

クライアント・アプリケーション設計が先読みに適していない場合は、以下の方法で先読みを使用不可にすることができます。

- 次のように設定することにより、キューのレベルで先読みを使用不可にします。クライアント・アプリケーションから要求されない限り先読みを不使用にしておきたい場合は、キュー属性の DEFREADA を NO

に設定します。あるいは、クライアント・アプリケーションからの要求の有無に関係なく先読みを不使用にしておきたい場合は、DEFREADA を DISABLED に設定します。

- アプリケーションのレベルで先読みを使用不可にするには、MQOPEN 関数呼び出しで MQOO_NO_READ_AHEAD オプションを使用します。

2 種類の MQCLOSE オプションにより、先読みバッファーに格納されたメッセージをキューのクローズ時に処理する方法を設定することができます。

- 先読みバッファー内のメッセージを廃棄するには、MQCO_IMMEDIATE を使用します。
- キューのクローズ前に先読みバッファー内のメッセージをアプリケーションによってコンシュームさせるには、MQCO_QUIESCE を使用します。 MQCO_QUIESCE 付きで MQCLOSE を発行した際に、先読みバッファーにメッセージが残っていると、MQCC_WARNING と共に MQRC_READ_AHEAD_MSGS が戻ります。

AIX 上の非持続メッセージのためのパフォーマンスの調整
AIX V5.3 以降を使用している場合は、非持続メッセージのパフォーマンスを最大限に使用するように調整パラメーターを設定することを検討してください。

調整パラメーターがすぐに有効になるように設定するには、以下のコマンドを root ユーザーとして発行してください。

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

調整パラメーターがすぐに有効になり、リブート後も持続するように設定するには、以下のコマンドを root ユーザーとして発行してください。

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

通常は、非持続メッセージはメモリー内のみに保持されますが、状況によっては、AIX で非持続メッセージのディスクへの書き込みをスケジュールすることもできます。 ディスクに書き込むようにスケジュールされたメッセージは、ディスクへの書き込みが完了するまで MQGET が利用することはできません。 推奨される調整コマンドによって、このしきい値が異なります。 例えば、16 キロバイトのデータがキューに入れられた場合にメッセージをディスクに書き込むようにスケジュールする代わりに、マシン上の実記憶が満杯に近くなった場合にのみディスクへの書き込みが行われます。 これはグローバルな変更であり、他のソフトウェア・コンポーネントにも影響する可能性があります。

AIX では、マルチスレッド・アプリケーションを使用する場合や特に複数のプロセッサーを搭載したマシンで実行する場合は、アプリケーションを開始する前に、AIXTHREAD_SCOPE=S を mqm ID `.profile` に設定するか、環境内の AIXTHREAD_SCOPE=S を設定することを強くお勧めします。これにより、パフォーマンスが向上し、スケジューリングがより強固なものになります。 以下に例を示します。

```
export AIXTHREAD_SCOPE=S
```

AIXTHREAD_SCOPE=S を設定すると、デフォルト属性を使用して作成されたユーザー・スレッドが、システム全体の競合範囲内に入れられます。 システム全体の競合範囲で作成されたユーザー・スレッドは、カーネル・スレッドにバインドされ、カーネルによってスケジュールされます。 ベースとなるカーネル・スレッドは、他のユーザー・スレッドとは共用されません。

ファイル記述子

エージェント・プロセスなどのマルチスレッド・プロセスを実行しているときに、ファイル記述子のソフト限界に達することがあります。 この制限により、IBM MQ 理由コード MQRC_UNEXPECTED_ERROR (2195) が与えられます。十分なファイル記述子がある場合は、IBM MQ FFST™ ファイルが与えられます。

この問題は、処理できるファイル記述子の数を増やすことで回避できます。 これを行うには、mqm ユーザー ID の `/etc/security/limits` またはデフォルトのスタンザにある `nofiles` 属性を 10,000 に変更します。

システム・リソース限界

コマンド・プロンプトで以下のコマンドを使用して、データ・セグメントおよびスタック・セグメントのシステム・リソース限界を無制限に設定します。

```
ulimit -d unlimited
ulimit -s unlimited
```

索引のタイプ

キュー属性の *IndexType* は、該当するキューに対する MQGET 操作の速度を向上させるためにキュー・マネージャーが維持する索引のタイプを指定します。

注 : IBM MQ for z/OS でのみサポートされています。

次の 5 つのオプションがあります。

値	説明
NONE	索引を維持しません。 これは、メッセージを 順序どおりに取り出すときに使用します (774 ページの『優先順位』 を参照)。
GROUPID	グループ ID の索引は保持されます。 メッセージ・グループの論理順序を保つ必要がある場合は、必ずこの索引タイプを使用しなければなりません (774 ページの『論理的な順序付けと物理的な順序付け』 を参照)。
MSGID	メッセージ ID の索引は保持されます。 このオプションは、MQGET 呼び出しで *MsgId* フィールドを選択基準としてメッセージを取り出す場合に使用します (785 ページの『特定のメッセージの読み取り』 を参照)。
MSGTOKEN	メッセージ・トークンの索引は保持されます。
CORRELID	相関 ID の索引は保持されます。 このオプションは、MQGET 呼び出しで *CorrelId* フィールドを選択基準としてメッセージを取り出す場合に使用します (785 ページの『特定のメッセージの読み取り』 を参照)。

注 :

1. MSGID オプションまたは CORRELID オプションを使用して索引付けを行う場合は、関連する **MsgId** または **CorrelId** パラメーターを MQMD に設定してください。 これらのパラメーターを両方設定しても、速度は向上しません。
2. ブラウズするとき、キューが以下の条件のすべてに適合する場合、メッセージを見つけるために索引のメカニズムが使用されます。
   - 索引タイプは MSGID、CORRELID、または GROUPID
   - 同じタイプの ID でブラウズされる
   - 1 つの優先順位しかないメッセージがある
3. キュー (*MsgId* または *CorrelId* によって索引付けられた) に何千ものメッセージを入れるのは、再始動時に影響を与えるので避けてください (非持続メッセージは再始動時には削除されるので、この注意事項は非持続メッセージには該当しません。)
4. MSGTOKEN は、z/OS ワークロード管理プログラムによって管理されるキューを定義する目的で使用します。

IndexType 属性の詳しい説明については、IndexType を参照してください。 **IndexType** 属性のさらに詳しい説明については、65 ページの『z/OS アプリケーションの設計およびパフォーマンスに関する考慮事項』 を参照してください。

4 MB より長いメッセージの処理

メッセージが大きすぎて、アプリケーション、キュー、またはキュー・マネージャーでの処理には適さない場合があります。 環境に応じて、IBM MQ は 4 MB より長いメッセージを扱うためのいくつかの方法を提供しています。

V6 以降のすべての IBM MQ システムで、**MaxMsgLength** 属性を最大 100 MB まで増やすことができます。キューを使用するメッセージのサイズを反映するようにこの値を設定してください。 IBM MQ for z/OS 以外の IBM MQ システムでは、以下を行うこともできます。

1. セグメント化したメッセージを使用する。 (メッセージは、アプリケーションとキュー・マネージャーのどちらでもセグメント化できます。)
2. 参照メッセージを使用する。

この節の残りの部分では、この 3 つの方法について説明します。

最大メッセージ長の増加

キュー・マネージャー属性の **MaxMsgLength** では、キュー・マネージャーで処理できる 1 つのメッセージの最大長を指定します。 同様に、キュー属性の **MaxMsgLength** では、キューで処理できる 1 つのメッセージの最大長を指定します。 サポートされるデフォルトの最大メッセージ長は、使用している環境によって異なります。

大きなメッセージを処理する場合には、z/OS 以外のプラットフォームでこれらの属性を個別に変更できます。 このキュー・マネージャー属性の値は 32768 バイトから 100 MB の範囲内で設定できます。

重要 : IBM MQ for z/OS では、キュー・マネージャーの **MaxMsgLength** 属性が 100 MB にハードコーディングされています。

このキュー属性の値は、すべてのプラットフォームで 0 から 100 MB の範囲内で設定できます。

どちらか一方または両方の **MaxMsgLength** 属性を変更したあとで、変更が有効になるように、アプリケーションおよびチャネルを再始動してください。

これらの変更を行う場合、メッセージ長は、キューおよびキュー・マネージャーの **MaxMsgLength** 属性をどちらも超えてはなりません。 ただし、既存のメッセージの長さは、いずれかの属性より長くても構いません。

メッセージが大きすぎてキューで処理できない場合は、MQRC_MSG_TOO_BIG_FOR_Q が戻されます。 同様に、メッセージが大きすぎてキュー・マネージャーで処理できない場合は、MQRC_MSG_TOO_BIG_FOR_Q_MGR が戻されます。

大きなメッセージはセグメント化すると、処理が簡潔に行えます。 ただし、メッセージをセグメント化する場合、次のことを考慮してください。

- キュー・マネージャー間の均一性が多少損なわれること。 メッセージ・データの最大サイズは、メッセージが書き込まれる各キュー (伝送キューを含む) の *MaxMsgLength* によって決定されます。 多くの場合、特に伝送キューについては、この値のデフォルト値としてキュー・マネージャーの *MaxMsgLength* が使用されます。 このため、メッセージをリモート・キュー・マネージャーに送るときには、メッセージが大きすぎるかどうかを予測するのが困難になります。
- システム・リソースの使用が増大すること。 例えば、アプリケーションで従来より大きなバッファーが必要になるほか、一部のプラットフォームでは、共用ストレージの使用量が増大することも考えられます。 キューのストレージが影響を受けるのは、より大きなメッセージが実際にストレージを必要とする場合だけです。
- チャネルのバッチ処理に影響を与えること。 大きなメッセージであっても、バッチ・カウントでは単に 1 つのメッセージとしてカウントされます。一方、伝送時間は通常より長くかかるので、他のメッセージの応答時間が長くなることになります。

Multi メッセージのセグメント化

この情報を使用して、メッセージのセグメント化について学習します。 この機能は、IBM MQ for z/OS ではサポートされていません。また IBM MQ classes for JMS を使用するアプリケーションによってもサポートされていません。

トピック 792 ページの『最大メッセージ長の増加』 で説明した最大メッセージ長を大きくする方法には、いくつかの欠点があります。 また、最大メッセージ長を大きくしても、まだメッセージが大きすぎて、キューまたはキュー・マネージャーで処理できないことも考えられます。 このような場合には、メッセージ

をセグメント化することができます。 セグメント化については、44 ページの『メッセージ・グループ』を参照してください。

次の各節では、メッセージのセグメント化の一般的な使用法を紹介します。 書き込みと、除去を伴う読み取りでは、MQPUT 呼び出しまたは MQGET 呼び出しが必ず 1 つの作業単位の中で動作するものとします。 ネットワーク上に不完全なグループが生じる可能性を減らすため、常にこの方法を使用することを考慮してください。 キュー・マネージャーで単一フェーズ・コミットが行われることを前提としていますが、その他の調整方法も有効です。

また、読み取り側アプリケーションでは、複数のサーバーが同一のキューを処理している場合に、あるサーバーが (以前に MQGMO_ALL_MSGS_AVAILABLE または MQGMO_ALL_SEGMENTS_AVAILABLE を指定しているため) メッセージまたはセグメントの検索に失敗することがないように、各サーバーで同様のコードが実行されるものとします。

重要 : パブリッシュ/サブスクライブを使用してメッセージをトピックに送信する (またはメッセージをトピック別名に書き込む) 場合、メッセージのグループ化とセグメンテーションは許可されません。

サブスクリプションはパブリケーション・アクティビティーとは独立して作成および削除できるため、サブスクライバーがメッセージ・グループ全体またはメッセージのすべてのセグメントを受け取ることは保証できません。 RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP を参照してください。

複数の作業単位にわたるセグメント化メッセージの書き込みと読み取り

セグメント化したメッセージも、783 ページの『複数の作業単位にわたるグループの書き込みと読み取り』と同様の方法で複数の作業単位にわたって書き込んだり、読み取ったりできます。

ただし、グローバル作業単位では、セグメント化したメッセージを書き込んだり読み取ったりすることはできません。

Multi キュー・マネージャーによるセグメント化と再組み立て

ここでは、あるアプリケーションが書き込んだメッセージを、別のアプリケーションが取り出すという最も単純な場合を例に挙げて説明します。 メッセージは大きくなることがあります。書き込み側アプリケーションや読み取り側アプリケーションが 1 つのバッファーで処理できないほど大きくはならないものの、キュー・マネージャーや、メッセージが書き込まれるキューが処理できない大きさになることがあります。

これらのアプリケーションで必要な変更はごくわずかです。まず、書き込み側アプリケーションで、必要な場合にキュー・マネージャーによるセグメント化を許可します。これは次のように指定します。

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

また、読み取り側アプリケーションでは、セグメント化されているメッセージを再組み立てするようにキュー・マネージャーに要求します。これは、次のように指定します。

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

この単純なシナリオでは、アプリケーションは、MQPUT 呼び出しの前に GroupId フィールド を MQGI_NONE にリセットして、キュー・マネージャーが固有のグループ ID を各メッセージに対して生成 できるようにしなければなりません。 これが行われないと、無関係の複数のメッセージが 同じグループ ID を持ち、その結果として以降の処理が正しく行われなくなる 可能性があります。

アプリケーションのバッファーは、(MQGMO_ACCEPT_TRUNCATED_MSG オプションを指定した場合を除き) 再組み立て後のメッセージを格納できる大きさでなければなりません。

メッセージのセグメント化を可能にするようにキューの MAXMSGLEN 属性を変更する場合は、以下の点を考慮してください。

- ローカル・キューでサポートされる最小メッセージ・セグメントは 16 バイトです。
- 伝送キューの場合、MAXMSGLEN にはヘッダーに必要なスペースも含める必要があります。 伝送キューに書き込まれる可能性のあるメッセージ・セグメントでは、予期されるユーザー・データの最大長より少なくとも 4000 バイト大きい値を使用することを考慮してください。

データ変換が必要な場合には、読み取り側アプリケーションで MQGMO_CONVERT を指定して変換を実行しなければならないことがあります。 完成したメッセージと共にデータ変換出口が渡されるので、データ変換は簡単に行うことができます。 メッセージがセグメント化されており、データが不完全でデータ変換出口が変換を実行できないようなデータ形式になっている場合は、送信側チャネルでデータ変換を試行しないでください。

Multi アプリケーションによるセグメント化

キュー・マネージャーによるセグメント化では不十分な場合や、特定のセグメント境界でのデータ変換がアプリケーションで必要な場合に、アプリケーションによるセグメント化が使用されます。

アプリケーションによるセグメント化を使用する主な理由には、次の 2 つが考えられます。

1. メッセージが大きすぎて、アプリケーションが単一のバッファー内で処理できないため、キュー・マネージャーによるセグメント化だけでは不十分である。
2. 送信側チャネルでデータ変換を実行する必要がある。また、データは、個別のセグメントの変換を可能にするために書き込み側アプリケーションでセグメントの境界を定めなければならない形式である。

ただし、データ変換上の問題がない場合や、読み取り側アプリケーションで常に MQGMO_COMPLETE_MSG を使用する場合は、MQMF_SEGMENTATION_ALLOWED を指定することによってキュー・マネージャーによるセグメント化を許可することもできます。 次の例では、アプリケーションによってメッセージが 4 つのセグメントに分割されます。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

MQPMO_LOGICAL_ORDER を使用しない場合は、*Offset* および各セグメントの長さをアプリケーションで設定する必要があります。 この場合は、論理的な状態が自動的に保守されることはありません。

読み取り側アプリケーションでは、再組み立てされたメッセージをすべて保持できるだけの大きさのバッファーを確保できません。 したがって、読み取り側アプリケーションでは、セグメントを個別に処理できるようにする必要があります。

セグメント化されているメッセージの場合、このアプリケーションでは、該当する論理メッセージを構成するすべてのセグメントが存在しない限り、セグメントの処理を開始しないようにします。 このため、最初のセグメントについては MQGMO_ALL_SEGMENTS_AVAILABLE を指定します。
MQGMO_LOGICAL_ORDER を指定した場合に現行の論理メッセージが存在すると、
MQGMO_ALL_SEGMENTS_AVAILABLE は無視されます。

論理メッセージの最初のセグメントを取り出したあとは、MQGMO_LOGICAL_ORDER を使用して、論理メッセージの残りのセグメントが必ず順序どおりに取り出されるようにします。

異なるグループのメッセージについては考慮されません。 そのようなメッセージがあった場合は、キューの中で各メッセージの最初のセグメントが現れた順に処理されます。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi アプリケーションによる論理メッセージのセグメント化

メッセージは、グループ内の論理順序に従って保守する必要があります。また、それらの一部またはすべてが、アプリケーション・セグメンテーションを必要とするほど大きくなる場合があります。

次の例では、4 つの論理メッセージで構成されるグループを書き込みます。 3 番目のメッセージ以外は大きいので、書き込み側アプリケーションでセグメント化を行う必要があります。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

読み取り側アプリケーションでは、最初の MQGET で MQGMO_ALL_MSGS_AVAILABLE を指定します。 これは、グループ全体が使用可能になるまで、そのグループのメッセージおよびセグメントを一切取り出されないようにするためです。 グループ内の最初の物理メッセージが取り出された時点で、MQGMO_LOGICAL_ORDER を使用して、このグループのセグメントおよびメッセージが必ず順序どおりに取り出されるようにします。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

注 : MQGMO_LOGICAL_ORDER を指定した場合に、現行グループが存在すると、MQGMO_ALL_MSGS_AVAILABLE は無視されます。

参照メッセージ

この情報を使用して、参照メッセージの詳細について学習します。

注 : IBM MQ for z/OS ではサポートされていません。

この方法を使用すると、あるノードから別のノードへ大きなオブジェクトを転送するときに、送信元ノードまたは宛先ノードで IBM MQ キューにオブジェクトを格納する必要がありません。 この方法は、データが別の形式で存在する場合 (メール・アプリケーションなどの場合) には特に便利です。

この方法を使用するには、チャネルの両側でメッセージ出口を指定します。 これを行う方法については、984 ページの『チャネル・メッセージ出口プログラム』を参照してください。

IBM MQ では、参照メッセージ・ヘッダー (MQRMH) の形式が定義されています。 これについては、MQRMH を参照してください。 このヘッダーは定義済みの形式名によって識別され、ヘッダーの後には実際のデータが存在する場合があります。

大きなオブジェクトの転送を開始するために、アプリケーションは、参照メッセージ・ヘッダーだけで構成される、ヘッダーのあとにデータのないメッセージを書き込むことができます。 このメッセージがノードから送信されると、メッセージ出口が適切な方法でオブジェクトを取り出して参照メッセージに付加します。 次に、(以前より大きくなった) そのメッセージを送信側メッセージ・チャネル・エージェント (MCA) に戻します。これにより受信側 MCA への伝送が可能になります。

受信側 MCA には、別のメッセージ出口が構成されます。 このメッセージ出口はこのようなメッセージのいずれかを受信すると、付加されたオブジェクト・データを使用してオブジェクトを作成しますが、オブジェクトを付けずに 参照メッセージを渡します。 これで、アプリケーションが参照メッセージを受信できるようになります。このアプリケーションは、このノードでオブジェクト (あるいは、少なくともこの参照メッセージで表されているオブジェクトの一部) が作成されたことを認識します。

送信側メッセージ出口が参照メッセージに付加できるオブジェクト・データの最大長は、チャネルの あらかじめ決められた最大メッセージ長によって制限されます。 この出口から MCA に戻すことができるメッセージは、渡された各メッセージにつき 1 つだけなので、書き込み側アプリケーションでは複数のメッセージを書き込むことによって 1 つのオブジェクトを転送できます。 それぞれのメッセージでは、付加されるオブジェクトの論理 長およびオフセットを識別できるようにする必要があります。 ただし、オブジェクトの合計サイズまたはチャネルで許容される最大サイズが分からない場合は、書き込み側アプリケーションで 1 つのメッセージだけを書き込み、渡されたメッセージに可能な限りのデータを付加し終えたときには送信側メッセージ出口自身が伝送キューに次のメッセージを書き込むように、メッセージ送信出口を設計してください。

この方法を使用して大きなメッセージを処理するときには、次の点を考慮してください。

- MCA およびメッセージ出口は IBM MQ のユーザー ID のもとで実行します。 送信側でオブジェクトを取り出すため、または受信側でオブジェクトを作成するために、メッセージ出口 (つまりユーザー ID) はオブジェクトにアクセスする必要があります。オブジェクトが任意のユーザー ID でアクセス可能な場合にのみ、このアクセスは可能です。 ただし、この場合にはセキュリティー上の問題が生じます。
- バルク・データが付加された参照メッセージを、複数のキュー・マネージャーを経由して宛先に送る必要がある場合には、中間にある各ノードで IBM MQ キューにバルク・データが存在することになります。ただし、このような場合のための特別なサポートや出口を提供する必要はありません。
- 経路変更または送達不能キューイングが可能な場合には、メッセージ出口の設計が困難になります。 このような場合は、オブジェクトを構成する各部分が順番どおりに到着しないことがあります。
- 参照メッセージが宛先に到着すると、受信側メッセージ出口はオブジェクトを作成します。 しかし、この処理は MCA の作業単位と同期化されないため、バッチがバックアウトされた場合、そのオブジェクトの同じ部分を含む別の参照メッセージが後のバッチで到着すると、メッセージ出口でその部分を再作成しようとする可能性があります。 例えば、オブジェクトが一連のデータベース更新であれば、このような処理を許容することはできません。 このような場合は、メッセージ出口で適用済みの更新を記録したログを保存する必要があります。そのときに、IBM MQ キューを使用する必要がある場合があります。
- オブジェクト・タイプの特性によっては、オブジェクトが不要になったときに削除できるように、メッセージ出口とアプリケーションが連携して使用回数を保守しなければならないことがあります。 また、インスタンス ID が必要となる場合もあります。この ID を指定するためのフィールドは、参照メッセージ・ヘッダー (MQRMH を参照) にあります。
- 参照メッセージを配布リストとして書き込む場合には、結果として得られる配布リストまたはそのノードの宛先ごとに、オブジェクトが取り出し可能でなければなりません。 使用回数を保守しなければならないことがあります。 また、ノードが、リスト内の一部の宛先については最終ノードでありながら、その他の宛先については中間ノードである可能性も考慮する必要があります。
- バルク・データの変換は、(送信側チャネルで要求したとしても) 通常は行われません。 これは、メッセージ出口が呼び出される前に 変換が実行されるためです。 そのため、発信元の送信側チャネルで変換を要求しないでください。 参照メッセージが中間ノードを経由する場合、(要求があれば) バルク・データは中間ノードから送信されるときに変換されます。
- 参照メッセージをセグメント化することはできません。

MQRMH 構造体と MQMD 構造体の使用法

参照メッセージ・ヘッダーとメッセージ記述子の各フィールドの説明については、MQRMH および MQMD を参照してください。

MQMD 構造体では、*Format* フィールドに MQFMT_REF_MSG_HEADER を設定してください。 MQGET 呼び出しで MQHREF 形式を要求すると、この形式がそのあとに続くバルク・データと共に IBM MQ で自動的に変換されます。

MQRMH 構造体の *DataLogicalOffset* フィールドと *DataLogicalLength* フィールドの使用例を次に示します。

書き込み側アプリケーションでは、次のような参照メッセージを書き込みます。

- 物理データなし
- *DataLogicalLength* = 0 (このメッセージはオブジェクト全体を表す)
- *DataLogicalOffset* = 0

オブジェクトが 70 000 バイトの長さであると仮定すると、送信側メッセージ出口は 最初の 40 000 バイトを次のような内容の参照メッセージによってチャネル経由で送信します。

- MQRMH と、そのあとに続く 40 000 バイトの物理データ
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (オブジェクトの最初から)

次に、次のようなもう 1 つのメッセージを伝送キューに入れます。

- 物理データなし
- *DataLogicalLength* = 0 (オブジェクトの最後まで)。 ここには、値 30 000 を指定することもできます。
- *DataLogicalOffset* = 40000 (ここが開始点となる)

このメッセージ出口が、送信側メッセージ出口で確認されると、次のようにしてデータの残り 30,000 バイトを付加すると共に、フィールドを設定します。

- MQRMH と、そのあとに続く 30,000 バイトの物理データ
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (ここが開始点となる)

MQRMHF_LAST フラグも設定します。

参照メッセージの使用例を示すサンプル・プログラムの説明については、1063 ページの『Multiplatforms でのサンプル・プログラムの使用』を参照してください。

メッセージの待機

メッセージがキューに到着するまでプログラムを待機させる場合は、MQGMO 構造体の *Options* フィールドに、MQGMO_WAIT オプションを指定します。

MQGMO 構造体の *WaitInterval* フィールドを使用して、以下を指定します。 メッセージがキューに到着するのを MQGET 呼び出しに待たせる最大時間 (ミリ秒)。

メッセージがこの時間内に到着しない場合は、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE の理由コードを出して完了します。

WaitInterval フィールドに定数 MQWI_UNLIMITED を入れて、無限の待機時間を指定できます。 しかし、制御が及ばないイベントによって、プログラムが長時間待機させられる可能性もあるので、この定数を使用するときは注意してください。 IMS アプリケーションでは、無限の待機間隔を指定すると IMS システムを終了できなくなるため、無限の待機間隔は指定しないでください。 (IMS の終了時にはすべての従属領域の終了が必要です。) 代わりに、IMS アプリケーションでは有限の待機間隔を指定することができます。その間隔のあと、呼び出しがメッセージを検索しないで完了した場合は、待機オプションを指定した別の MQGET 呼び出しを出してください。

注 : 複数のプログラムがメッセージを除去する ために同じ共用キューで待機している場合は、メッセージの到着によってアクティブ化されるプログラムは 1 つだけです。 しかし、複数のプログラムがメッセージをブラウズするために待機している場合は、すべてのプログラムをアクティブ化することができます。 詳細については、 MQGMO の MQGMO 構造体の *Options* フィールドの説明を参照してください。

待機間隔が終了する前に、キューまたはキュー・マネージャーの状態が変更された場合は、次のようなことが起こります。

- キュー・マネージャーが静止状態に入り、MQGMO_FAIL_IF_QUIESCING オプションを使用している場合は、待機が取り消され、MQGET 呼び出しは MQRC_Q_MGR_QUIESCING の理由コードを出して完了する。このオプションを使用していない場合は、呼び出しが待機し続ける。

- z/OS z/OS で、接続 (CICS または IMS アプリケーション用) が静止状態になったときに MQGMO_FAIL_IF_QUIESCING オプションを使用すると、待機が取り消され、MQGET 呼び出しが完了して理由コード MQRC_CONN_QUIESCING が戻される。 このオプションを使用していない場合は、呼び出しが待機し続ける。
- キュー・マネージャーが強制的に停止されたか、取り消された場合は、MQGET 呼び出しが MQRC_Q_MGR_STOPPING または MQRC_CONNECTION_BROKEN のどちらかの理由コードで完了する。
- キュー (またはキュー名が解決された結果のキュー) の属性が変更されたために、読み取り要求が禁止されている場合は、待機が取り消され、MQGET 呼び出しは MQRC_GET_INHIBITED の理由コードで完了する。
- キュー (または、キュー名を解決して得られるキュー) の属性が、FORCE オプションを必要とする方法で変更された場合は、待機が取り消され、MQGET 呼び出しが完了して理由コード MQRC_OBJECT_CHANGED が戻される。

z/OS アプリケーションを複数のキューで待機させる場合は、IBM MQ for z/OS の信号機能を使用してください (798 ページの『信号機能』を参照)。 これらの処置が取られる状況の詳細については、MQGMO を参照してください。

信号機能

信号機能は、IBM MQ for z/OS でのみサポートされています。

信号機能は、MQGET 呼び出しのオプションです。信号機能を使用すると、あるキューに予期されるメッセージが到着したときにオペレーティング・システムからプログラムに通知する (つまり信号 を出す) ことができます。 これは、信号を待機している間もプログラムが他の作業を続行できるという点では、トピック 797 ページの『メッセージの待機』で説明した「待機付き読み取り」機能に似ています。 ただし、信号機能を使用している場合には、アプリケーション・スレッドを解放し、メッセージが到着したときの通知はオペレーティング・システムに完全に任せることができます。

信号の設定

信号を設定するには、MQGET 呼び出しで使用する MQGMO 構造体を次のように設定します。

1. *Options* フィールドに MQGMO_SET_SIGNAL オプションを設定する。
2. *WaitInterval* フィールドに信号の最長存続時間を設定する。 ここでは、IBM MQ がキューをモニターする時間の長さ (ミリ秒単位) を設定します。 無制限の存続時間を指定するには、MQWI_UNLIMITED 値を使用してください。

   注 : IMS アプリケーションでは、無限の待機間隔を指定すると IMS システムを終了できなくなるため、無限の待機間隔は指定しないでください。 (IMS の終了時にはすべての従属領域の終了が必要です。) 代わりに、IMS アプリケーションは一定の間隔で ECB の状態を調べることができます (ステップ 3 を参照)。 プログラムは、いくつかのキュー・ハンドルに同時に信号を設定できます。
3. *Signal1* フィールドにイベント制御ブロック (ECB) のアドレスを指定します。 これにより、信号の結果が通知されます。 ECB ストレージは、キューがクローズされるまで使用可能になっていなければなりません。

注 : MQGMO_SET_SIGNAL オプションを MQGMO_WAIT オプションと共に使用することはできません。

メッセージが到着したときに行われる処理

適切なメッセージが到着すると、ECB に完了コードが戻されます。

この完了コードで記述される情報は、次のいずれかです。

- 信号を設定したメッセージがキューに到着しました。 このメッセージは信号を要求したプログラムのために予約されていません。そのため、このプログラムで、メッセージを読み取るには、もう一度 MQGET 呼び出しを発行する必要があります。

  注 : 信号を受信してから、もう一度 MQGET 呼び出しを発行するまでの間に、他のアプリケーションがメッセージを読み取る可能性もあります。

- 設定された待機時間が満了しましたが、信号を設定したメッセージはキューに到着しませんでした。 IBM MQ は信号を取り消しました。
- 信号が取り消されました。 これは、例えば、キュー・マネージャーが停止した場合や、キューの属性が変更されて MQGET 呼び出しが許可されなくなった場合に起こります。

通知するメッセージが既にキュー上にある場合は、信号が設定されていないときと同様に MQGET 呼び出しが完了します。 また、エラーが即時に検出される場合も、呼び出しが完了し、戻りコードが設定されます。

呼び出しが受け入れられたとき、ただちに入手できるメッセージが存在しない場合は、他の作業を続けることができるように、制御がプログラムに戻されます。 メッセージ記述子内のどの出力フィールドも設定されませんが、**CompCode** パラメーターは MQCC_WARNING に、**Reason** パラメーターは MQRC_SIGNAL_REQUEST_ACCEPTED に設定されます。

IBM MQ が信号機能を使用して MQGET 呼び出しを発行したときにアプリケーションに戻す情報については、MQGET を参照してください。

ECB が通知されるのを待機している間、プログラムが他に行う作業がない場合は、次のものを使用して ECB を待機します。

- CICS Transaction Server for z/OS プログラムの場合は、EXEC CICS WAIT EXTERNAL コマンド
- バッチおよび IMS プログラムの場合は、z/OS WAIT マクロ

信号が設定されている間 (つまり、ECB がまだ通知されていない間) に、キューまたはキュー・マネージャーの状態が変更された場合は、次のようなことが起こります。

- キュー・マネージャーが静止状態に入り、MQGMO_FAIL_IF_QUIESCING オプションを使用されていたときは、信号が取り消される。 ECB は MQEC_Q_MGR_QUIESCING 完了コードで通知される。 このオプションが指定されていないときは、信号が設定されたままになります。
- キュー・マネージャーが強制的に停止されたか、取り消された場合は、信号は取り消される。 信号は、完了コード MQEC_WAIT_CANCELED によって送達されます。
- キュー (または、キュー名が解決された結果のキュー) の属性が変更されたために、読み取り要求が禁止されるようになった場合は、信号は取り消される。 信号は、完了コード MQEC_WAIT_CANCELED によって送達されます。

注 :

1. メッセージを除去するために、複数のプログラムが同じ共用キューに信号を設定していた場合は、メッセージの到着によってアクティブ化されるプログラムは 1 つだけです。 しかし、複数のプログラムがメッセージをブラウズするために待機している場合は、すべてのプログラムをアクティブ化することができます。 活動化するアプリケーションを決定する際にキュー・マネージャーが従う規則は、待機中のアプリケーションの規則と同じです。詳しくは、 MQGMO-メッセージ取得オプションの MQGMO 構造体の *Options* フィールドの説明を参照してください。
2. 同じメッセージを待機している MQGET 呼び出しが複数あって、待機オプションと信号オプションが混じっている場合は、各待機呼び出しが同等であると見なされます。 詳細については、 MQGMO-メッセージ読み取りオプションの MQGMO 構造体の *Options* フィールドの説明を参照してください。
3. ある条件の下では、MQGET 呼び出しによってメッセージが検索され、さらに、(同じメッセージの到着によって) 信号も送達されるということが起こる場合があります。 これは、(信号が送達されたために) プログラムが別の MQGET 呼び出しを出したときに、使用可能なメッセージがない場合もあることを意味します。 プログラムは、この状況をテストするように設計してください。

シグナルの設定方法については、Signal1 の MQGMO_SET_SIGNAL オプションと *Signal1* フィールドの説明を参照してください。

バックアウトのスキップ

MQGET 呼び出しに **MQGMO_MARK_SKIP_BACKOUT** オプションを指定すると、アプリケーション・プログラムが「*MQGET* - エラー - バックアウト」のループに入ることを回避できます。

注 : IBM MQ for z/OS でのみサポートされています。

作業単位の一部として、アプリケーション・プログラムは、キューからメッセージを読み取るために 1 つまたは複数の MQGET 呼び出しを発行することができます。 アプリケーション・プログラムがエラーを検

出した場合は、その作業単位をバックアウトできます。 これによって、その作業単位の中で更新されたすべてのリソースが、作業単位の開始前の状態に戻され、さらに MQGET 呼び出しによって検索されたメッセージも、元に戻されます。

元に戻されたメッセージは、そのあとにアプリケーション・プログラムによって発行される MQGET 呼び出しで使用可能になります。 通常、これにより、アプリケーション・プログラムに問題が生じることはありません。 しかし、バックアウトの原因となったエラーを回避することができない場合には、メッセージをキューに戻すことによって、アプリケーション・プログラムが「*MQGET* - エラー - バックアウト」のループに入る可能性があります。

この問題を避けるために、MQGET 呼び出しに MQGMO_MARK_SKIP_BACKOUT オプションを指定してください。 これは、MQGET 要求を、アプリケーションが開始するバックアウトに含まれない (つまり、バックアウトしてはならない) ものとしてマークを付けます。 このオプションを使用すると、バックアウトが起こったときに、他のリソースに対する更新は要求どおりにバックアウトされますが、マークの付いたメッセージは、新しい作業単位のもとで検索されたかのように扱われます。

アプリケーション・プログラムは、新しい作業単位をコミットしたり、新しい作業単位をバックアウトしたりするときに IBM MQ 呼び出しを発行する必要があります。 例えば、プログラムは、メッセージが廃棄されたことを発信元に通知するなどの例外処理を行い、作業単位をコミットしてキューからメッセージを除去することができます。新しい作業単位が (何らかの理由で) バックアウトされた場合は、メッセージがキューに戻されます。

1 つの作業単位内には、バックアウトのスキップとしてマークを付けられた MQGET 要求は 1 つしか許されませんが、そのマークが付いていない他のメッセージもいくつかあります。 あるメッセージにバックアウトのスキップ・マークを付けたあとは、その作業単位で MQGMO_MARK_SKIP_BACKOUT を指定した MQGET 呼び出しを発行しても、その呼び出しは MQRC_SECOND_MARK_NOT_ALLOWED 理由コードで失敗します。

注 :

1. マークの付いたメッセージは、それを含む作業単位がアプリケーションのバックアウト要求によって終了した場合にだけ、バックアウトをスキップします。 その作業単位が他の理由でバックアウトされた場合は、そのメッセージも、バックアウトのスキップ・マークが付いていないときと同様に、キューにバックアウトされます。
2. スキップのバックアウトは、RRS で制御された作業単位に関連する Db2 ストアード・プロシージャーではサポートされていません。 例えば、MQGMO_MARK_SKIP_BACKOUT オプションを指定した MQGET 呼び出しは、理由コード MQRC_OPTION_ENVIRONMENT_ERROR で失敗します。

801 ページの図 63 は、MQGET 要求がバックアウトをスキップすることを要求されたときに、アプリケーション・プログラムに含まれる典型的な一連のステップを示しています。

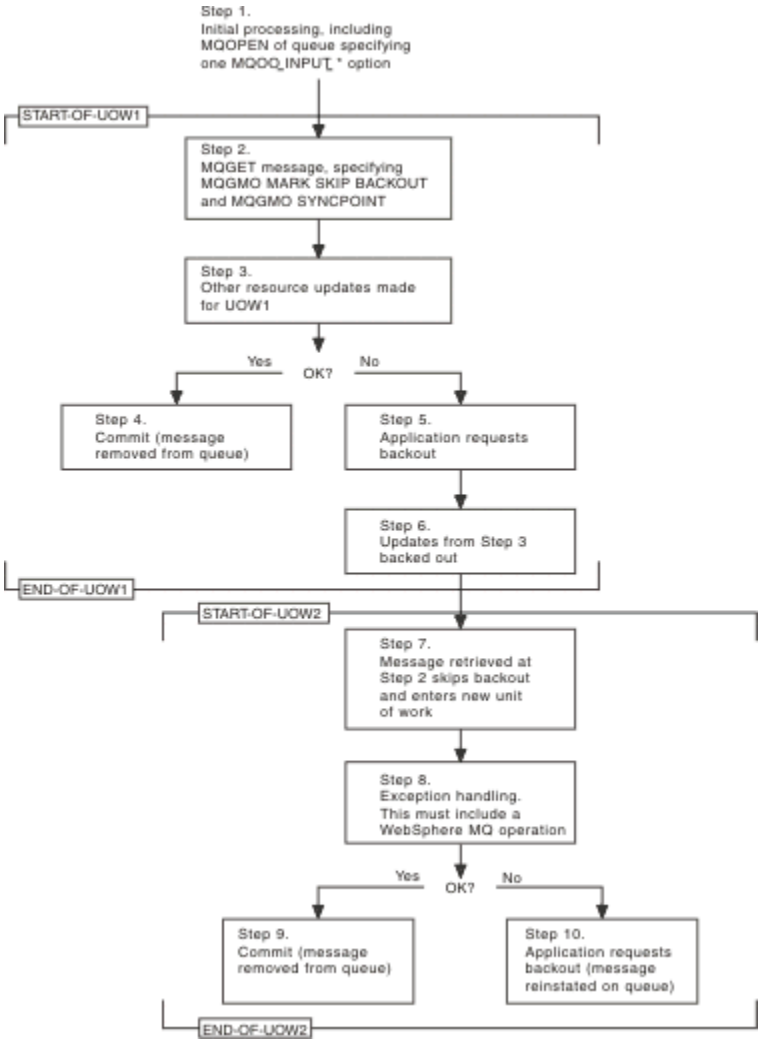


図 *63. MQGMO_MARK_SKIP_BACKOUT* の使用によるバックアウトのスキップ

801 ページの図 63 の各ステップは以下のとおりです。

ステップ 1
トランザクション内で初期処理が行われる。これには、キューをオープンするための MQOPEN 呼び出し (ステップ 2 でキューからメッセージを読み取るために MQOO_INPUT_* オプションの 1 つを指定した) が含まれます。

ステップ 2
MQGET が MQGMO_SYNCPOINT および MQGMO_MARK_SKIP_BACKOUT で呼び出される。 MQGMO_SYNCPOINT が必要なのは、MQGMO_MARK_SKIP_BACKOUT を有効にするためのに MQGET が作業単位内になければならないからです。 801 ページの図 63 では、この作業単位を UOW1 としています。

ステップ 3
他のリソースの更新が UOW1 の一部として行われる。 これには、さらに MQGET 呼び出し (MQGMO_MARK_SKIP_BACKOUT の指定がない) も含まれることがあります。

ステップ 4
ステップ 2 およびステップ 3 からの更新がすべて要求どおりに完了した。 アプリケーション・プログラムは更新をコミットし、UOW1 は終了します。 ステップ 2 で検索されたメッセージはキューから除去されます。

ステップ 5
ステップ 2 およびステップ 3 からの更新のいくつかが要求どおりに完了しなかった。 アプリケーション・プログラムは、これらのステップで行われた更新をバックアウトすることを要求します。

ステップ 6
ステップ 3 で行われた更新がバックアウトされる。

ステップ 7
ステップ 2 で行われた MQGET 要求が、バックアウトをスキップして、新しい作業単位 UOW2 の一部になる。

ステップ 8
UOW1 がバックアウトされたことに対応して、UOW2 が例外処理を行う (例えば、別のキューに対する MQPUT 呼び出しで問題が発生したために UOW1 がバックアウトされることを示す。)

ステップ 9
ステップ 8 が要求どおり完了し、アプリケーション・プログラムが活動をコミットし、UOW2 が終了した。 MQGET 要求が UOW2 の一部になっているので (ステップ 7 を参照)、このコミットによってメッセージがキューから除去されます。

ステップ 10
ステップ 8 が要求どおり完了しないので、アプリケーション・プログラムが UOW2 をバックアウトする。 読み取りメッセージ要求が UOW2 の一部になっているので (ステップ 7 を参照)、そのメッセージもバックアウトされ、キューに戻されます。 これにより、そのメッセージは、この同じアプリケーション・プログラムまたは別のアプリケーション・プログラムが発行する MQGET 呼び出しに対して、(キュー上の他のメッセージと同じように) 使用可能になります。

アプリケーション・データの変換

必要な場合は、MCA が、メッセージ記述子およびヘッダー・データを必要な文字セットとエンコード方式に変換します。 リンクの両端 (つまり、ローカル MCA またはリモート MCA) のいずれかで、変換が実行されます。

アプリケーションがメッセージをキューに書き込むと、ローカル・キュー・マネージャーにより、メッセージがキュー・マネージャーや MCA に処理される場合の制御を容易にするために、制御情報がメッセージ記述子に追加されます。 環境によっては、メッセージ・ヘッダー・データ・フィールドがローカル・システムの文字セットおよびエンコード方式で作成されます。

システム間でメッセージを移動させる場合、これらのアプリケーション・データを、受信側のシステムで必要とされる文字セットやエンコード方式に変換しなければならない場合があります。 このような変換は、受信側のシステムのアプリケーション・プログラム内から、または送信側のシステムの MCA によって実行できます。 受信側システムでデータ変換がサポートされている場合は、送信側システムで既に実行された変換の結果をそのまま使用せず、アプリケーション・プログラムを使用してアプリケーション・データを変換してください。

MQGET 呼び出しに渡す MQGMO 構造体の *Options* フィールドに MQGMO_CONVERT オプションを指定した場合に以下のすべての 条件が満たされると、アプリケーション・プログラム内でアプリケーション・データが変換されます。

- キュー上のメッセージに関連付けられている MQMD 構造体の *CodedCharSetId* フィールド・セットまたは *Encoding* フィールド・セットが、MQGET 呼び出しで指定された MQMD 構造体の *CodedCharSetId* フィールド・セットまたは *Encoding* フィールド・セットとは異なる。
- メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、MQFMT_NONE でない。
- MQGET 呼び出しで指定された *BufferLength* が、ゼロでない。
- メッセージ・データの長さが、ゼロでない。
- キュー・マネージャーが、メッセージと MQGET 呼び出しに関連付けられている MQMD 構造体に指定された *CodedCharSetId* フィールドと *Encoding* フィールドの間の変換をサポートしている。 サポートされているコード化文字セット ID とマシン・エンコードの詳細については、CodedCharSetId および Encoding を参照してください。
- キュー・マネージャーが、メッセージ形式の変換をサポートしている。 メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、組み込み形式のうちの 1 つである場合、キュー・マネージャーはそのメッセージを変換できます。 *Format* が、組み込み形式のうちの 1 つでない場合には、データ変換出口を作成してそのメッセージを変換する必要があります。

送信側の MCA でデータの変換を行う場合は、変換を必要とする各送信側チャネルまたはサーバー・チャネルの定義で CONVERT(YES) キーワードを指定してください。 データ変換に失敗した場合、メッセージは送信側のキュー・マネージャーの DLQ に送信され、MQDLH 構造体の *Feedback* フィールドにその理由が示されます。 メッセージを DLQ に書き込めない場合は、チャネルがクローズされ、未変換のメッセージが伝送キュー上に残ります。 送信側の MCA でなくアプリケーション内でデータ変換を行うと、この状態を回避できます。

規則として、組み込み形式またはデータ変換出口により文字 データとして記述されるメッセージ内のデータは、そのメッセージで使用されるコード化文字セットから要求されたコード化文字セットに変換され、また数値 フィールドは、要求されたエンコード方式に変換されます。

組み込み形式を変換する際に使用する変換処理規則について、また独自のデータ変換出口を作成する方法については、988 ページの『データ変換出口の作成』 を参照してください。 また、言語サポート・テーブルおよびサポートされているマシン・エンコードについては、各国語およびマシン・エンコードを参照してください。

EBCDIC 改行文字の変換

EBCDIC プラットフォームから ASCII プラットフォームに送信したデータと、再び戻されるデータとを同じにする必要がある場合は、EBCDIC 改行文字の変換を制御しなければなりません。

この操作は、変更されていない変換テーブルを IBM MQ に強制的に使用させる、プラットフォーム依存型のスイッチを使用して行います。ただし、結果的に動作の整合性が失われることもあるので注意が必要です。

この問題が発生するのは、EBCDIC 改行文字がプラットフォーム間または変換テーブル間で正しく変換されていないからです。 この場合には、ASCII プラットフォーム上でデータを表示しても、正しく形式表示されないことがあります。 この問題が発生した場合、例えば、RUNMQSC を使って ASCII プラットフォームから IBM i システムをリモート管理することは非常に困難です。

EBCDIC のフォーマット・データを ASCII フォーマットに変換することに関する詳細は、データ変換を参照してください。

キュー上のメッセージのブラウズ

この情報を使用して、MQGET 呼び出しを使用してキューにあるメッセージをブラウズする方法を理解します。

キューにあるメッセージをブラウズするために MQGET 呼び出しを使用するには、次のステップに従ってください。

1. MQOO_BROWSE オプションを指定した MQOPEN を呼び出して、キューをブラウズするためにオープンする。
2. キューの最初のメッセージを表示するには、MQGMO_BROWSE_FIRST オプションを指定して MQGET を呼び出す。 必要なメッセージを検出するには、MQGMO_BROWSE_NEXT オプションを指定して MQGET をそのメッセージをすべて検索するまで繰り返し呼び出します。

   メッセージをすべて見るために、各 MQGET 呼び出しのあとで、MQMD 構造体の *MsgId* および *CorrelId* フィールドをヌルに設定する必要があります。
3. MQCLOSE を呼び出して、キューをクローズする。

ブラウズ・カーソル

キューをブラウズするためにオープン (MQOPEN) すると、その呼び出しによってブラウズ・カーソルが設定されます。ブラウズ・カーソルは、ブラウズ・オプションの 1 つを用いる MQGET 呼び出しで使用されます。 ブラウズ・カーソルをキューの最初のメッセージの前に位置する論理ポインターと考えることができます。

同じキューに対していくつかの MQOPEN 要求を発行することによって、(単一のプログラムから) 複数のブラウズ・カーソルを活動化させることができます。

ブラウズするため、MQGET を呼び出すとき、MQGMO 構造体に次のオプションの 1 つを指定します。

MQGMO_BROWSE_FIRST
MQMD 構造体に指定された条件を満たす、最初のメッセージのコピーを読み取る。

MQGMO_BROWSE_NEXT
MQMD 構造体に指定された条件を満たす、次のメッセージのコピーを読み取る。

MQGMO_BROWSE_MSG_UNDER_CURSOR
カーソルによって現在ポイントされているメッセージ、 つまり、MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT オプションのどちらかを使用して最後に取り出されたメッセージのコピーを読み取る。

どの場合もメッセージはキューに残ります。

キューをオープンしたときは、ブラウズ・カーソルはキューの最初のメッセージの直前に論理的に置かれます。 このことは、MQOPEN 呼び出しのあと、即時に MQGET 呼び出しを行う場合は、MQGMO_BROWSE_NEXT オプションを使用して最初のメッセージをブラウズできることを意味します。 MQGMO_BROWSE_FIRST オプションを使用する必要はありません。

メッセージがキューからコピーされる順序は、キューの **MsgDeliverySequence** 属性によって決まります (詳細については、773 ページの『メッセージがキューから取り出される順序』 を参照してください。)

- 804 ページの『FIFO (先入れ先出し) 順序のキュー』
- 804 ページの『優先順位順序のキュー』
- 804 ページの『未コミット・メッセージ』
- 805 ページの『キュー順序の変更』
- 805 ページの『キューの索引の使用』

FIFO (先入れ先出し) 順序のキュー

この順序におけるキューの最初のメッセージは、一番長くキューに置かれていたメッセージです。

MQGMO_BROWSE_NEXT を使用すると、キューのメッセージを順番に読み取ります。 この順次のキューはメッセージを最後に置くので、ブラウズ中にはキューに書き込まれたどのメッセージでも見ることができます。 カーソルがキューの終わりに達したことを認識すると、ブラウズ・カーソルはその場に留まり、MQRC_NO_MSG_AVAILABLE を戻します。 その場合、次のメッセージを待ってカーソルをそのままにするか、または MQGMO_BROWSE_FIRST 呼び出しを使用してキューの始めにリセットすることができます。

優先順位順序のキュー

この順序にあるキューの最初のメッセージは、そのキュー上で最長であり、また MQOPEN 呼び出しが発行されたときに最高優先順位をもつメッセージです。

MQGMO_BROWSE_NEXT を使用すると、キューに入っているメッセージを読み取ることができます。

ブラウズ・カーソルは次のメッセージを指します。最初のメッセージの優先順位から始まり、優先順位の最も低いメッセージで終わります。 この間にキューに書き込まれたメッセージは、それが現行ブラウズ・カーソルで識別されているメッセージと同じ、もしくはそれより低い優先順位のメッセージである限り、すべてブラウズされます。

キューに書き込まれた、それより高い優先順位のメッセージは、次の場合にのみブラウズされます。

- ブラウズするキューを再びオープンする。この時点で新規のブラウズ・カーソルが確立されます。
- MQGMO_BROWSE_FIRST オプションを使用する。

未コミット・メッセージ

コミットされていないメッセージはブラウズでは認識されず、ブラウズ・カーソルはそのメッセージをスキップします。

1 つの作業単位内のメッセージは、その作業単位がコミットされるまでブラウズされません。 コミット時にはキューでのメッセージの位置は変わらないので、スキップされる、コミットされていないメッセージ

は、MQGMO_BROWSE_FIRST オプションを使用し、キューでの作業を再び行わない限り、コミット時 であっても表示されません。

キュー順序の変更

メッセージ送達順序を、メッセージがキューにある間に優先順位から FIFO に変更した場合、既にキューに入っているメッセージの順序は変更されません。 後からキューに追加されるメッセージは、そのキューのデフォルトの優先順位で処理されます。

キューの索引の使用

索引付きキューの中のメッセージ (持続メッセージと非持続メッセージのどちらか一方、あるいはその両方) の優先順位が単一の場合、キュー・マネージャーは、特定の形式でブラウズするときに索引を使用します。

注 : IBM MQ for z/OS でのみサポートされています。

索引付きキューの中のメッセージの優先順位が単一の場合、以下のいずれかの形式のブラウズが使用されます。

1. キューが MSGID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の MSGID を渡すブラウズ要求が処理されます。
2. キューが CORRELID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の CORRELID を渡すブラウズ要求が処理されます。
3. キューが GROUPID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の GROUPID を渡すブラウズ要求が処理されます。

ブラウズ要求が MQMD 構造体の MSGID、CORRELID、または GROUPID を渡さないものの、キューに索引が付けられていてメッセージが戻される場合には、メッセージの索引項目を見つけなければならず、その中の情報を使ってブラウズ・カーソルが更新されることになります。 選択の幅の広い索引値を使用する場合でも、ブラウズ要求に特に余分な処理は追加されません。

メッセージ長が分からない場合のメッセージのブラウズ

メッセージのサイズが分からないときには、*MsgId* フィールド、*CorrelId* フィールド、および *GroupId* フィールドを使用してメッセージを検索しなくても、次のように、
MQGMO_BROWSE_MSG_UNDER_CURSOR オプションを使用してメッセージをブラウズできます。

1. 次を指定して MQGET を発行する。
   - MQGMO_BROWSE_FIRST オプションまたは MQGMO_BROWSE_NEXT オプションのいずれか
   - MQGMO_ACCEPT_TRUNCATED_MSG オプション
   - バッファー長ゼロ

   注 : 別のプログラムが同じメッセージを受け取る可能性が高い場合には、MQGMO_LOCK オプションの併用を検討してください。 MQRC_TRUNCATED_MSG_ACCEPTED が戻されます。
2. 戻された *DataLength* を使用して必要なストレージを割り振る。
3. MQGMO_BROWSE_MSG_UNDER_CURSOR を指定して MQGET を発行する。

ポインターが指しているメッセージは、最後に検索されたメッセージです。ブラウズ・カーソルは移動されません。 MQGMO_LOCK オプションを使用してメッセージをロックするか、それとも MQGMO_UNLOCK オプションを使用してロック済みメッセージをアンロックするかを選択できます。

キューがオープンされてから、MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT オプションを指定した MQGET が発行されていない場合、呼び出しは失敗します。

ブラウズしたメッセージの除去

メッセージをブラウズするためだけでなく除去するためにキューをオープンしていた場合は、既にブラウズしたメッセージをキューから除去することができます。 (MQOPEN 呼び出しでは、MQOO_INPUT_* オプションの 1 つ、および MQOO_BROWSE オプションを指定する必要があります。)

メッセージを除去するには、MQGET を再度呼び出します。ただし、MQGMO 構造体の *Options* フィールドに MQGMO_MSG_UNDER_CURSOR を指定します。この場合、MQGET 呼び出しは MQMD 構造体の *MsgId*、 *CorrelId*、および *GroupId* フィールドを無視します。

ブラウズおよび除去のステップの間で、別のプログラムがキューからメッセージを除去してしまう場合があります。このとき、ブラウズ・カーソルの下のメッセージも除去されることもあります。 その場合、MQGET 呼び出しから、メッセージが利用できないことを示す理由コードが戻ります。

論理順序でのメッセージのブラウズ

774 ページの『論理的な順序付けと物理的な順序付け』では、キューにあるメッセージの論理順序と物理順序の相違について説明しています。 論理順序と物理順序の区別は、キューをブラウズするときに特に重要です。これは、ブラウズでは、通常はメッセージを削除しないため、またブラウズ操作が必ずしもキューの先頭から開始されるとは限らないためです。

アプリケーションで、(論理順序を使用して) あるグループのさまざまなメッセージをブラウズする場合には、論理順序に従って次のグループの先頭に進むことが重要です。これは、あるグループの最後のメッセージが、物理的には次のグループの最初のメッセージのあと に 置かれている場合もあるためです。 MQGMO_LOGICAL_ORDER オプションを使用すると、常に論理順序に従ってキューを走査することができます。

ブラウズ操作では、注意して MQGMO_ALL_MSGS_AVAILABLE (または MQGMO_ALL_SEGMENTS_AVAILABLE) を使用してください。 MQGMO_ALL_MSGS_AVAILABLE を指定した論理メッセージの場合を例に挙げて説明します。 このオプションを指定した場合、論理メッセージが使用可能になるのはグループ内の残りのメッセージがすべて存在する場合だけです。 この条件に当てはまらない場合、論理メッセージはスキップされます。 このため、欠落しているメッセージがあとから到着しても、次のメッセージをブラウズする操作では認識されない可能性があります。

例えば、次のような論理メッセージがあるとします。

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

この場合に、MQGMO_ALL_MSGS_AVAILABLE を指定してブラウズ関数を発行すると、グループ 456 の最初のメッセージが戻され、ブラウズ・カーソルはこの論理メッセージに 置かれたままになります。 次に、グループ 123 の 2 番目 (最後) のメッセージが到着したとします。

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456  <=== browse cursor
Logical message 2 (last)     of group 456
```

ここで、同じように次のメッセージをブラウズする関数を発行した場合は、グループ 123 の最初のメッセージがブラウズ・カーソルより前 にあるため、このグループのメッセージがすべて揃っていることが認識されません。

場合によっては (例えば、1 つのグループ全体が存在するときにメッセージを取り出して除去したような場合は)、MQGMO_ALL_MSGS_AVAILABLE を MQGMO_BROWSE_FIRST と共に使用することができます。 それ以外の場合は、欠落していたメッセージが新たに到着していないかどうかを確認するためにブラウズ走査を繰り返す必要があります。ただし、単に MQGMO_BROWSE_NEXT および MQGMO_ALL_MSGS_AVAILABLE を指定して MQGMO_WAIT を発行しても、このようなメッセージは取り出されません。 (これは、メッセージの走査が完了したあとで優先順位の高いメッセージが到着するような場合にも起こります。)

以下の節では、セグメント化されていないメッセージに関するブラウズ走査の例を紹介します。セグメント化されているメッセージをブラウズする場合も、原理は同様です。

グループ化されているメッセージのブラウズ

この例では、キューにある各メッセージをアプリケーションから論理順序に従ってブラウズします。

キューにあるメッセージは、グループ化されている場合があります。 グループ化されているメッセージについては、そのグループ内のメッセージがすべて到着するまで、アプリケーションではグループの処理を

一切開始しません。 このため、グループ内の最初のメッセージについては、MQGMO_ALL_MSGS_AVAILABLE を指定します。ただし、後続のメッセージについては、このオプション必要ありません。

この例では、MQGMO_WAIT を使用します。 ただし、新しいグループが到着した場合には、806 ページの『論理順序でのメッセージのブラウズ』で示した理由によって待機の条件は満たされますが、ブラウズ・カーソルが既にグループ内の最初の論理メッセージを通過したあとに残りのメッセージが到着した場合は、待機の条件は満たされません。 それでも、適切な間隔だけ待機することで、新しいメッセージまたはセグメントを待機する間、アプリケーションがたびたびループすることがなくなります。

走査を常に論理順序で実行するために、最初から最後まで MQGMO_LOGICAL_ORDER を使用します。 これは、除去を伴う MQGET の例とは対照的です。除去を伴う MQGET の例では、各グループが除去されるためにグループ内の最初の (または唯一の) メッセージを検索するときに MQGMO_LOGICAL_ORDER を使用していません。

アプリケーションのバッファーは、メッセージがセグメント化されているかどうかにかかわらず、常にメッセージ全体を保持できるだけの大きさであることが前提とされています。 このため、それぞれの MQGET で MQGMO_COMPLETE_MSG を指定しています。

あるグループ内の論理メッセージをブラウズする例を次に示します。

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
| MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

グループのブラウズは、MQRC_NO_MSG_AVAILABLE が戻るまで繰り返されます。

ブラウズおよび除去を伴う取り出し

この例では、アプリケーションはグループ内の各論理メッセージをブラウズしてから、そのグループを破壊的に取得するかどうかを決定します。

この例の最初の部分は、1 つ前の例と似ています。 ただし、この例では、あるグループ全体をブラウズしてから、必要に応じて最初に戻り、そのグループを取り出してキューから除去します。

この例では、各グループが除去されるので、グループ内の最初のメッセージまたは唯一のメッセージを検索するときに MQGMO_LOGICAL_ORDER は使用しません。

ブラウズと、削除を伴う取り出しの例を次に示します。

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
```

```
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
...
```

ブラウズされたメッセージの送達が繰り返されることの回避
オープンのオプションおよびメッセージ読み取りのオプションのうち特定のオプションを使用することによって、 メッセージにブラウズ済みのマークを付け、それらのメッセージを現行アプリケーションまたは他の連携アプリケーションが 再び取り出さないようにすることができます。 明示的または自動的にメッセージのマークを解除して、 ブラウズに使用できる状態に戻すことができます。

キューにあるメッセージをブラウズする場合、 メッセージを破壊的に読み取る場合に想定される順序とは異なる 順序でメッセージを取り出すことが可能です。 具体的には、同じメッセージを複数回ブラウズ できますが、もしそのメッセージがキューから削除されるのであればそれは不可能です。 これを 避けるため、メッセージにブラウズ済みのマークを付け、マークされた メッセージの取り出しを回避することができます。 これを、マーク付けブラウズ といいます。 ブラウズされた メッセージにマークを付けるには、メッセージ読み取りオプション MQGMO_MARK_BROWSE_HANDLE を 使用し、マークが付けられていないメッセージのみを取り出すには MQGMO_UNMARKED_BROWSE_MSG を使用します。
MQGMO_BROWSE_FIRST オプション、MQGMO_UNMARKED_BROWSE_MSG オプション、 および MQGMO_MARK_BROWSE_HANDLE オプションを組み合わせて使用し、繰り返し MQGET を発行すると、キューにある各メッセージを順々に取り出すことになります。 これは、 メッセージがスキップされないように MQGMO_BROWSE_FIRST が使用されていても、メッセージ送達の繰り返しを防止します。 これらのオプションの組み合わせが、1 つの定数 MQGMO_BROWSE_HANDLE で表されます。 まだブラウズされたことのないメッセージがキューにない場合、MQRC_NO_MSG_AVAILABLE が 戻されます。

複数のアプリケーションが同じキューをブラウズする場合、 それらのアプリケーションは、MQOO_CO_OP および MQOO_BROWSE オプションを指定してキューをオープンできます。 各 MQOPEN によって戻される オブジェクト・ハンドルは、連携グループの一部であると見なされます。
MQGMO_MARK_BROWSE_CO_OP オプションを指定した MQGET 呼び出しで戻される メッセージは、この連携ハンドル・セット用にマークされるものと見なされます。

あるメッセージに しばらくの間マークが付けられていた場合、キュー・マネージャーによって自動的に マーク解除が行われるようにし、もう一度ブラウズ可能にすることができます。 キュー・マネージャー 属性 MsgMarkBrowseInterval が、連携するハンドルのセット用にメッセージにマークが付いたままにしておく時間をミリ秒単位で指定します。 MsgMarkBrowseInterval の値が -1 の場合、 自動的にメッセージのマーク解除が行われることはないことを意味します。

単一プロセス または連携するプロセスのセットがメッセージのマーク付けを停止すると、 マークされていたすべてのメッセージがマーク解除されます。

連携ブラウズの例

1 つのディスパッチャー・アプリケーションの複数のコピー を実行して、キューにあるメッセージをブラウズし、各メッセージの内容に 基づいてコンシューマーを開始することができます。 各ディスパッチャーでは、 MQOO_CO_OP を指定してキューをオープンします。 これは、これらの複数のディスパッチャーが連携し、 互いに他のディスパッチャーのマーク付きメッセージに注意することを示します。 各ディスパッチャーは、 MQGMO_BROWSE_FIRST オプション、MQGMO_UNMARKED_BROWSE_MSG オプション、 および MQGMO_MARK_BROWSE_CO_OP オプションを指定して (1 つの定数 MQGMO_BROWSE_CO_OP を使用してこれらのオプションの組み合わせを表すことができます)、MQGET 呼び出しを繰り返します。 各ディスパッチャー・アプリケーション は、他の連携ディスパッチャーによってまだマークが付けられていないメッセージのみ を取り出します。 ディスパッチャーは、コンシューマーを初期化し、MQGET によって戻された MsgToken を 渡します。そうすると、コンシューマーはメッセージをキューから破壊的に取得

します。 コンシューマーがメッセージの MQGET をバックアウトする場合、 そのメッセージは、もうマークが付いていないので、いずれかのブラウザーが再ディスパッチできる ようになります。 コンシューマーがメッセージに対して MQGET を実行しない 場合、MsgMarkBrowseInterval が過ぎた後、キュー・マネージャーは、 連携するハンドルのセットのためにそのメッセージのマークを解除し、そのメッセージは再ディスパッチできるようになります。

同じ ディスパッチャー・アプリケーションの複数のコピーを使用するのではなく、 多数の異なるディスパッチャー・アプリケーション (それぞれがキューにある一部のメッセージの処理に適している) が キューをブラウズするようにすることもできます。 各ディスパッチャーでは、 MQOO_CO_OP を指定してキューをオープンします。 これは、これらの複数のディスパッチャーが連携し、 互いに他のディスパッチャーのマーク付きメッセージに注意することを示します。

- 1 つのディスパッチャーのメッセージ処理の順序が重要な場合、 各ディスパッチャーは、MQGMO_BROWSE_FIRST、 MQGMO_UNMARKED_BROWSE_MSG、および MQGMO_MARK_BROWSE_HANDLE オプション (または MQGMO_BROWSE_HANDLE) を指定して、MQGET 呼び出しを繰り返します。 ブラウズされたメッセージがこのディスパッチャーが処理するのに適している場合、 MQMO_MATCH_MSG_TOKEN、MQGMO_MARK_BROWSE_CO_OP と、 前の MQGET 呼び出しで戻された MsgToken を指定して、MQGET 呼び出しを実行します。 この呼び出しが 成功した場合、ディスパッチャーはコンシューマーを初期化し、コンシューマーに MsgToken を渡します。
- メッセージ処理の順序が重要でなく、ディスパッチャーがメッセージの大部分を 処理すると予期される場合、オプション MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG、および MQGMO_MARK_BROWSE_CO_OP (または MQGMO_BROWSE_CO_OP) を使用します。 ディスパッチャーは、処理できないメッセージをブラウズした場合、 オプション MQMO_MATCH_MSG_TOKEN、MQGMO_UNMARK_BROWSE_CO_OP と、前に戻された MsgToken を指定して MQGET を呼び出すことによって、 そのメッセージのマークを解除します。

MQGET 呼び出しが失敗する場合

キュー内の特定の属性が、MQOPEN と MQGET 呼び出しを発行する間のコマンドで FORCE オプションを使用して変更された場合、MQGET 呼び出しは失敗し、MQRC_OBJECT_CHANGED 理由コードが戻されます。

キュー・マネージャーは、オブジェクト・ハンドルを無効としてマーク付けします。 キュー名が解決された結果のキューにその変更が適用される場合にも同じことが起こります。 このようにハンドルに影響を及ぼす属性は、MQOPEN の MQOPEN 呼び出しの節で一覧表示されています。 呼び出しが理由コード MQRC_OBJECT_CHANGED を戻す場合は、キューをクローズして、キューを再度オープンしてから、メッセージの読み取りを再び行います。

メッセージを読み取ろうとするキュー (または、キュー名が解決された結果のキュー) に対して読み取り走査が禁止されている場合は、MQGET 呼び出しは失敗し、理由コード MQRC_GET_INHIBITED を戻します。これは、ブラウズするために MQGET 呼び出しを使用する場合でも起こります。 アプリケーションの設計が、他のプログラムがキューの属性を規則的に変更するようになっている場合、あとで MQGET 呼び出しを試行するときに、メッセージを正常に読み取りできる場合があります。

(一時または永続) 動的キューが削除されていると、前に取得したオブジェクト・ハンドルを使用した MQGET 呼び出しは失敗し、MQRC_Q_DELETED の理由コードを戻します。

パブリッシュ/サブスクライブ・アプリケーションの作成

パブリッシュ/サブスクライブ IBM MQ アプリケーションの作成を開始します。

パブリッシュ/サブスクライブの概念の概要については、「パブリッシュ/サブスクライブ・メッセージング」を参照してください。

さまざまなタイプのパブリッシュ/サブスクライブ・アプリケーションの作成については、以下のトピックを参照してください。

- 810 ページの『パブリッシャー・アプリケーションの作成』
- 817 ページの『サブスクライバー・アプリケーションの作成』
- 836 ページの『パブリッシュ/サブスクライブのライフ・サイクル』
- 841 ページの『パブリッシュ/サブスクライブのメッセージ・プロパティー』
- 843 ページの『メッセージの順序付け』

- 844 ページの『パブリケーションの代行受信』
- 852 ページの『パブリッシュのオプション』
- 852 ページの『サブスクリプション・オプション』

関連概念

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

5 ページの『IBM MQ 用アプリケーションの開発』
メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

721 ページの『プロシージャー型キューイング・アプリケーションの作成』
この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

918 ページの『プロシージャー型クライアント・アプリケーションの作成』
IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

1005 ページの『プロシージャー型アプリケーションの構築』
IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

1042 ページの『プロシージャー型プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

関連タスク

1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』
これらのサンプル・プログラムは、プロシージャー型言語で作成されており、Message Queue Interface (MQI) の標準的な使用法を示しています。 異なるプラットフォーム上の IBM MQ プログラム。

パブリッシャー・アプリケーションの作成

パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

単純な IBM MQ パブリッシャー・アプリケーションの書き込みは、メッセージをキュー (810 ページの表 121) に書き込むポイント・アプリケーションを指す IBM MQ ポイントを書き込むのと似ています。違いは、キューではなくトピックにメッセージを MQPUT することです。

表 *121. point-to-point* とパブリッシュ/サブスクライブの *IBM MQ* プログラム・パターン

ステップ	**point-to-point MQ 呼び出し**	**パブリッシュ MQ 呼び出し**
キュー・マネージャーに接続します。	MQCONN	MQCONN
キューをオープンする	MQOPEN	
トピックをオープンする		MQOPEN
メッセージを書き込む	MQPUT	MQPUT

表 121. *point-to-point* とパブリッシュ/サブスクライブの *IBM MQ* プログラム・パターン (続き)		
ステップ	**point-to-point MQ 呼び出し**	**パブリッシュ MQ 呼び出し**
トピックをクローズする		MQCLOSE
キューをクローズする	MQCLOSE	
キュー・マネージャーから切断する	MQDISC	MQDISC

具体的にするために、株価をパブリッシュする 2 つのアプリケーション 例があります。 キューへのメッセージ書き込みに似せてモデル化されている 1 つ目の例 (811 ページの『例 1: 固定トピックへのパブリッシャー』) では、管理者はキューを作成するのと同じような方法でトピック定義を 作成します。 プログラマーは、メッセージをキューではなくトピックに書き込むよう MQPUT を コーディングします。 2 つ目の例 (814 ページの『例 2: 可変トピックへのパブリッシャー』) では、プログラムと IBM MQ の相互作用のパターンは類似しています。 違いは、 管理者ではなくプログラマーが、メッセージが書き込まれる先のトピックを用意すること です。 実際には、これは、トピック・ストリング内容が定義されるか、 またはブラウザーからの入力などの他のソースによって提供されることを意味するのが一般的です。

関連概念

817 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする IBM MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

関連資料

DEFINE TOPIC

表示トピック

DISPLAYTPSTATUS

例 *1:* 固定トピックへのパブリッシャー

管理的に定義されたトピックへパブリッシュする方法を示すための IBM MQ プログラム。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

出力については、812 ページの図 65 を参照してください。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
                publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *64.* 固定トピックへの単純な *IBM MQ* パブリッシャー。

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *65. 1* つ目のパブリッシャー例からの出力サンプル

以下に選択したコード行は、IBM MQ 用のパブリッシャー・アプリケーション作成の各局面を示しています。

char topicNameDefault[] = "IBMSTOCKPRICE";
プログラム内でデフォルトのトピック名が定義されています。 プログラムの第 1 引数として、これと異なるトピック・オブジェクト の名前を指定すれば、これをオーバーライドできます。

MQCHAR resTopicStr[151];
resTopicStr は、td.ResObjectString.VSPtr によって ポイントされていて、解決後のトピック・ストリングを戻すために MQOPEN によって使用されます。 resTopicStr の長さ を td.ResObjectString.VSBufSize に渡される長さより 1 だけ長く して、ヌル終了のためのスペースを与えるようにします。

memset (resTopicStr, 0, sizeof(resTopicStr));
resTopicStr をヌルに初期設定して、 MQCHARV に戻される解決後のトピック・ストリングが 確実にヌル終了になるようにします。

td.ObjectType = MQOT_TOPIC
パブリッシュ/サブスクライブ用の新しいタイプのオブジェクト、トピック・オブジェクト があります。

td.Version = MQOD_VERSION_4;
新タイプのオブジェクトを使用するためには、バージョン *4* 以上のオブジェクト記述子を使用する必要があります。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
topicName は、トピック・オブジェクトの名前であり、 管理トピック・オブジェクトと呼ばれることもあります。 例では、IBM MQ エクスプローラーまたは次の MQSC コマンドを使用して、トピック・オブジェクトは前もって作成されている必要があります。

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
解決後のトピック・ストリングは、プログラム内の最後の printf でエコー出力 されます。 IBM MQ が解決されたストリングをプログラムに返すように、 MQCHARV ResObjectString 構造体をセットアップします。

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
出力用にトピックをオープンします。出力用にキューをオープンするのと同様です。

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
新規サブスクライバーがパブリケーションを受け取ることができるように したい場合、パブリッシャーに MQPMO_RETAIN を指定すると、 サブスクライバーを開始したときに、そのサブスクライバーは、開始前にパブリッシュされた 最後のパブリケーションを、最初に一致するパブリケーションとして受け取ります。 別の選択肢は、サブスクライバーが開始された後でのみパブリッシュされたパブリケーションを サブスクライバーに提供することです。 さらに、サブスクライバーには、 サブスクリプションに MQSO_NEW_PUBLICATIONS_ONLY を指定することで保存パブリケーションの受け取りを 拒否するというオプションもあります。

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
MQPUT に渡されるストリングの長さに 1 を加算することで、メッセージ・バッファーの一部として IBM MQ にヌル終了文字を渡します。

この例 1 は何を説明しているのでしょうか? この例は、point-to-point IBM MQ プログラム作成についての、試行および試験済みの従来のパターンにできるだけ似せて作成されています。 この IBM MQ プログラミング・パターンの重要なフィーチャーの 1 つは、どこにメッセージが送信されるのかをプログラマーが知らなくていいということです。 プログラマーの作業は、 キュー・マネージャーに接続し、受信者に配布されるメッセージをそのキュー・マネージャーに 渡すことです。 point-to-point パラダイムでは、 プログラマーは、管理者が構成したキュー (おそらく別名キュー) をオープン します。 別名キューは、メッセージを、ローカル・キュー・マネージャー またはリモート・キュー・マネージャーのターゲット・キューに経路指定します。 メッセージは、送信されるのを待つ間、ソースと宛先の間の任意の場所にあるキューに保管されます。

パブリッシュ/サブスクライブのパターンでは、 キューをオープンする代わりに、プログラマーはトピックをオープンします。 この例では、トピックは、管理者によってトピック・ストリングに 関連付けられま

す。 キュー・マネージャーは、パブリケーションを、 パブリケーションのトピック・ストリングと一致するサブスクリプションを持つ、 ローカルまたはリモートのサブスクライバーに、キューを使用して転送します。 保存パブリケーションの 場合、キュー・マネージャーは、現在はサブスクライバーを持っていなくても、 パブリケーションの最新コピーを保持します。 保存パブリケーションは、将来のサブスクライバーに 転送するために使用可能です。 パブリッシャー・アプリケーションは、パブリケーションを宛先に転送または選択するのに関与しません。 パブリッシャー・アプリケーションのタスクは、パブリケーションを作成して、管理者が定義したトピック に書き込むことです。

この固定トピックの例は、静的であり、 多くのパブリッシュ/サブスクライブ・アプリケーションの典型例ではありません。 この 例では、管理者がトピック・ストリングを定義し、パブリッシュが行われるトピックを変更する 必要があります。 通常、パブリッシュ/サブスクライブ・アプリケーションは、 トピック・ツリーの一部または全体のことを認識している必要があります。 トピックは頻繁に 変わることが多く、たとえそれほど変わらなくても、トピック組み合わせの数 は大きくなります。また、パブリッシュが行われる必要があるかもしれないトピック・ストリングの それぞれについてトピック・ノードを管理者が定義するのは、あまりにも 面倒です。 トピック・ストリングはパブリケーションより前には不明であることが多く、 パブリッシャー・アプリケーションは、パブリケーション内容からの情報を 使用してトピック・ストリングを指定したり、パブリッシュを行うトピック・ストリング についての情報を、ブラウザーからの入力など他のソースから利用したりします。 もっとダイナミックな スタイルでのパブリッシュを示すため、次の例では、パブリッシャー・アプリケーションの一部として、 トピックを動的に作成する方法を示します。

トピックは、 パブリッシャーとサブスクライバーを結合します。 トピックの命名およびトピック・ツリー内でのトピックの 編成に関する規則または体系の設計は、パブリッシュ/サブスクライブ・ソリューション開発 の重要なステップです。 トピック・ツリー編成のどの範囲までが パブリッシャー・プログラムとサブスクライバー・プログラムを結合し、 それらのプログラムをトピック・ツリーの内容に結合するのかを、注意深く見る必要があります。 トピック・ツリーを変更すると、パブリッシャー・アプリケーションおよび サブスクライバー・アプリケーションに影響があるかどうか、その 影響をどうすれば最小化できるかをよく検討してください。 IBM MQ パブリッシュ/サブスクライブ・モデルの体系に組み込まれているのは、トピックのルート部分またはルート・サブツリーを提供する、管理トピック・オブジェクトの概念です。 トピック・オブジェクトによって、トピック・ツリーのルート部分を管理的に定義するオプションが 提供されます。これによって、アプリケーション・プログラミングおよび操作が単純化され、その結果として保守容易性 が向上します。 例えば、 複数のパブリッシュ/サブスクライブ・アプリケーションをデプロイしようとしていて、それらのアプリケーションには、分離した複数のトピック・ツリーがある場合、 トピック・ツリーのルート部分を管理的に定義することによって、異なるアプリケーション間でトピック命名規則に一貫性がない場合でも、 各トピック・ツリーの分離を保証することができます。

実際には、パブリッシャー・アプリケーション は、この例のような固定トピックだけの使用から、 次の例のような可変トピックの使用まで、広い範囲をカバーします。 814 ページの『例 2: 可変トピックへのパブリッシャー』 は、 トピックおよびトピック・ストリングの使用の結合も例示しています。

関連概念

814 ページの『例 2: 可変トピックへのパブリッシャー』
プログラムで定義されたトピックへパブリッシュする方法を示す WebSphere MQ プログラム。

817 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする IBM MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

例 *2:* 可変トピックへのパブリッシャー
プログラムで定義されたトピックへパブリッシュする方法を示す WebSphere MQ プログラム。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

出力については、815 ページの図 67 を参照してください。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *66.* 可変トピックへの単純な *IBM MQ* パブリッシャー。

```
X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *67. 2* 番目のパブリッシャー例からの出力サンプル

この例について注意する点がいくつかあります。

char topicNameDefault[] = "STOCKS";
デフォルトのトピック名 STOCKS は、 トピック・ストリングの一部を定義します。 プログラムの第 1 引数として指定することで、このトピック名をオーバーライドできます。あるいは、 最初のパラメーターとして / を指定すれば、このトピック名の使用を除去することができます。

char topicString[101] = "IBM/PRICE";
IBM/PRICE は、デフォルトのトピック・ストリングです。 プログラム の第 2 引数として指定することで、このトピック・ストリングをオーバーライド できます。

キュー・マネージャーは、STOCKS トピック・オブジェクト "NYSE" によって提供されるトピック・ストリングを、プログラム "IBM/PRICE" によって提供されるトピック・ストリングと結合し、2 つのトピック・ストリングの間に "/" を挿入します。 結果は、解決されたトピック・ストリング "NYSE/IBM/PRICE" です。 結果のこのトピック・ストリングは、 IBMSTOCKPRICE トピック・オブジェクト内に定義されているものと 同じであり、効果もまったく同じです。

解決後のトピック・ストリングと関連付けられた管理トピック・オブジェクト は、パブリッシャーによって MQOPEN に渡されるトピック・オブジェクト と必ずしも同じでなくてもかまいません。 IBM MQ は、解決後のトピック・ストリング中のツリー暗黙指定を使用して、パブリケーションと関連付けられた属性を定義するのはどの管理トピック・オブジェクトなのかを解明します。

2 つのトピック・オブジェクト A と B があり、A がトピック "a"を定義し、B がトピック "a/b" (816 ページの図 68) を定義するとします。 パブリッシャー・プログラムがトピック・オブジェクト A を参照し、トピック・ストリング "b"を提供し、トピックをトピック・ストリング "a/b"に解決する場合、トピックが B に定義されているトピック・ストリング "a/b" と一致するため、パブリケーションはトピック・オブジェクト B からプロパティーを継承します。

if (strcmp(argv[1],"/"))
argv[1] はオプションで指定される topicName です。 "/" はトピック名としては無効です。ここでは、トピック名が存在せず、トピック・ストリング全体がプログラムによって提供されることを意味します。 815 ページの図 67 の 出力では、トピック・ストリング全体がプログラムによって動的に提供されることが示されています。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
デフォルトの場合、 IBM MQ エクスプローラーまたは次の MQSC コマンドを使用して、オプションの topicName を事前に作成しておく必要があります。

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
トピック・ストリングは、トピック記述子内の MQCHARV フィールドです。

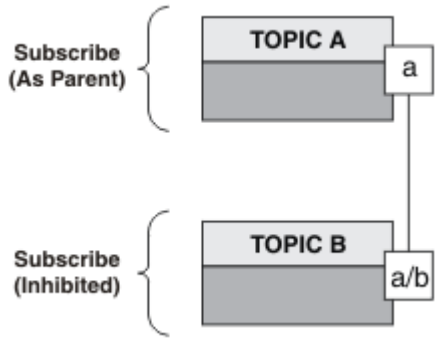


図 *68.* トピック・オブジェクトの関連

この例 2 は何を説明しているのでしょうか? コードは例 1 とよく似ています。 実際、違うのは 2 行のみですが、結果的には例 1 とは大きく異なるプログラムになっています。 プログラマーは、パブリケーションが送信される宛先を制御します。 サブスクライバー・アプリケーションの設計に使用される管理者入力が

最小限になると共に、パブリッシャーからサブスクライバーにパブリケーションを転送するために、トピックまたはキューが事前定義されている必要はありません。

point-to-point メッセージング・パラダイム では、メッセージが流れるには、その前にキューが定義されている必要があります。 パブリッシュ/サブスクライブの場合は、基礎にキューイング・システムを使用して IBM MQ がパブリッシュ/サブスクライブをインプリメントしていても、そうではありません。メッセージングとキューイングに関する、配信が保証されること、トランザクション化が可能であること、疎結合といった利点は、パブリッシュ/サブスクライブ・アプリケーションに継承されます。

設計者は、 パブリッシャー、サブスクライバー、プログラムが、基礎にあるトピック・ツリーに ついて認識する必要があるかどうか、また、サブスクライバー・プログラムが キューイングについて認識しているかどうかを決定しなければなりません。 次のサブスクライバー・アプリケーション例を 検討してください。 これらの例は、パブリッシャー例と一緒に使用されるように 設計されていて、通常は NYSE/IBM/PRICE にパブリッシュし、サブスクライブします。

関連概念

811 ページの『例 1: 固定トピックへのパブリッシャー』
管理的に定義されたトピックへパブリッシュする方法を示すための IBM MQ プログラム。

817 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする IBM MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

サブスクライバー・アプリケーションの作成

次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする IBM MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

817 ページの表 122 に、コンシューマーまたはサブスクライバーの 3 つのスタイルを、それらの特色を示している IBM MQ 機能呼び出しのシーケンスと共にリストします。

1. 最初のスタイルである MQ パブリケーション・コンシューマーは、 MQGET のみを実行する point-to-point MQ プログラムと同じです。 このアプリケーションには、 パブリケーションをコンシュームしているという認識はなく、単にキューから メッセージを読み取ります。 キューへのパブリケーションの転送を引き起こすサブスクリプションは、IBM MQ エクスプローラーまたはコマンドを使用して、管理的に作成されます。
2. 2 つ目のスタイルは、大部分のサブスクライバー・アプリケーションが優先的に使用するパターン です。 このサブスクライバー・アプリケーションは、サブスクリプションを作成し、その後で パブリケーションを取得します。 キュー管理はすべてキュー・マネージャーによって 実行されます。 これを管理サブスクライバー と呼びます。
3. 3 つ目のスタイルでは、サブスクライバー・アプリケーションは、 パブリケーションを保持するために使用されるキューを指定すること、そのキューのオープンとクローズを行うことと、サブスクリプションを発行することを選択して、キューをパブリケーションで満たします。 これを非管理サブスクライバー と呼びます。

これらのスタイルを理解する 1 つの方法は、817 ページの表 122 に リストされた各スタイルに対応する C プログラム例を検討することです。 これらの例は、 810 ページの『パブリッシャー・アプリケーションの作成』にあるパブリッシャー例と連動して実行できるよう設計されています。

表 *122. point-to-point vs.* サブスクライブ *IBM MQ* プログラム・パターン

ステップ	MQ メッセージ・コンシューマー	818 ページの『例 1: MQ パブリケーション・コンシューマー』	821 ページの『例 2: 管理 MQ サブスクライバー』	826 ページの『例 3: 非管理 MQ サブスクライバー』
キュー・マネージャーに接続します。	MQCONN	MQCONN	MQCONN	MQCONN

表 122. point-to-point vs. サブスクライブ IBM MQ プログラム・パターン (続き)				
ステップ	MQ メッセージ・コンシューマー	818 ページの『例 1: MQ パブリケーション・コンシューマー』	821 ページの『例 2: 管理 MQ サブスクライバー』	826 ページの『例 3: 非管理 MQ サブスクライバー』
キューをオープンする	MQOPEN	MQOPEN		MQOPEN
サブスクライブ			MQSUB	MQSUB
メッセージを読み取る	MQGET	MQGET	MQGET	MQGET
キューをクローズする	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
サブスクリプションをクローズする			MQCLOSE	MQCLOSE
キュー・マネージャーから切断する	MQDISC	MQDISC	MQDISC	MQDISC

MQCLOSE の使用は常にオプションであり、 リソースを解放するためか、MQCLOSE オプションを渡すためか、または単に MQOPEN との対称性のためです。 管理 MQ サブスクライバーのケースではサブスクリプション・キューがクローズされるときに MQCLOSE オプションを指定する必要はなく、また、対称性の議論は関係ないため、例 2: 管理 MQ サブスクライバーでは、サブスクリプション・キューは明示的にはクローズされません。

パブリッシュ/サブスクライブ・アプリケーションのパターンを理解する別の方法 は、関係するさまざまなエンティティー間の相互作用に注目することです。 ライフラインまたは UML シーケンス図は、相互作用を学習するためのいい方法です。 836 ページの『パブリッシュ/サブスクライブのライフ・サイクル』 に、3 つのライフライン例が記述されています。

例 *1: MQ* パブリケーション・コンシューマー
MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM MQ メッセージ・コンシューマーです。

この例のためのサブスクリプションおよびパブリケーションのキューを作成するため、以下のコマンドを実行するか、IBM MQ エクスプローラーを使用してオブジェクトを定義します。

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

IBMSTOCKPRICESUB サブスクリプションは、当パブリッシャー例のために作成された IBMSTOCK トピック・オブジェクト と、ローカル・キュー STOCKTICKER を参照します。 トピック・オブジェクト IBMSTOCK は、 サブスクリプション内で使用されるトピック・ストリング、NYSE/IBM/PRICE を定義します。 パブリケーションを受け取るのに使用されるトピック・オブジェクトおよびキュー は、サブスクリプションが作成される前に定義される必要があることに注意してください。

この MQ パブリケーション・コンシューマー・パターンには、以下のように多くの重要な面があります。

1. マルチプロセッシング: パブリケーション読み取りの作業が分配されます。 すべてのパブリケーションが、サブスクリプション・トピックと関連付けられた単一のキューに入れられ ます。 複数のコンシューマーが、MQOO_INPUT_SHARED を使用してそのキューをオープンできます。
2. 集中管理されるサブスクリプション。 アプリケーションは、独自のサブスクリプション・トピック または サブスクリプションを構成しません。パブリケーションがどこに送信されるのかについては、管理者が担当します。
3. サブスクリプション集中: 複数の異なるサブスクリプションを単一のキューに 送信できます。

4. サブスクリプション永続性: キューは、コンシューマーがアクティブかどうかに関係なく、すべての パブリケーションを受け取ります。
5. マイグレーションおよび共存: このコンシューマーのコードは、point-to-point のシナリオでもパブリッシュ/サブスクライブのシナリオでも 同じように機能します。

サブスクリプションは、トピック・ストリング NYSE/IBM/PRICE と キュー STOCKTICKER との間に関係を作成します。 パブリケーションは、現在保持されているパブリケーションも 含めて、サブスクリプションが作成された時点から STOCKTICKER に転送 されます。

管理的に作成されたサブスクリプションは、管理であることも非管理であることも可能です。 管理サブスクリプションは、作成されたらすぐに有効になり、 それは非管理サブスクリプションでも同様です。 上記に示した当パターンのすべての面が管理サブスクリプションに 当てはまるわけではありません。 826 ページの『例 3: 非管理 MQ サブスクライバー』を参照してください。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

820 ページの図 70 では結果が示されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *69. MQ* パブリケーション・コンシューマー

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

図 *70. MQ* パブリケーション・コンシューマーからの出力

標準 IBM MQ *C* 言語プログラミングに関するいくつかのヒントに留意してください。

memset(publication, 0, sizeof(publicationBuffer));
printf を使用するフォーマット設定を簡単にするため、メッセージに確実に末尾ヌルが入るようにします。 このパブリッシャー例では、 strlen(publication) に 1 を加算することによって、 MQPUT に渡されるメッセージ・バッファーに末尾ヌルを 組み込んでいます。 MQCHAR バッファーをヌルに設定することは、それらのバッファーを使用してストリングを保管する IBM MQ C プログラムのお勧めのプログラミング・スタイルです。こうすることで、バッファーを完全には満たさない文字配列の後にヌルが続くことが確実になります。

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
メッセージ・バッファーの末尾にヌルを 1 つ予約して、 if (messlen == strlen(publication)); が true の場合に、返されたメッセージの末尾にヌルが含まれるようにします。 このヒントは、 上記のヒントを補完するものであり、publication の内容で置き換えられない、最低でも 1 つのヌルが publicationBuffer 内に あることを確実にします。

関連概念

821 ページの『例 2: 管理 MQ サブスクライバー』
管理 MQ サブスクライバーは、大部分のサブスクライバー・アプリケーションが優先的に使用するパターンです。 管理対象サブスクリプションは、IBM MQ がサブスクリプションを処理するするもので、登録および登録解除を行います。 この例は、キュー、トピック、またはサブスクリプションの管理的な定義を必要としません。

826 ページの『例 3: 非管理 MQ サブスクライバー』
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 非管理サブスクリプションは、アプリケーションが責任を持つ場所です。 サブスクリプションが保管されるキューを指定します。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

810 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

例 *2:* 管理 *MQ* サブスクライバー
管理 MQ サブスクライバーは、大部分のサブスクライバー・アプリケーションが優先的に使用するパターンです。 管理対象サブスクリプションは、IBM MQ がサブスクリプションを処理するするもので、登録および登録解除を行います。 この例は、キュー、トピック、またはサブスクリプションの管理的な定義を必要としません。

この例で示すような最も単純な種類の管理サブスクライバーは、 通常は非永続 サブスクリプションを使用します。 例では、非永続サブスクリプション にフォーカスを合わせます。 サブスクリプションは、MQSUB からのサブスクリプション・ハンドルの存続期間の間しか存続しません。 サブスクリプションの 存続期間中にトピック・ストリングと一致したパブリケーションが、 サブスクリプション・キューに送信されます (保存パブリケーションも送信されることがあります。 それは、フラグ MQSO_NEW_PUBLICATIONS_ONLY が設定されていないか、デフォルトに なっていて、以前のパブリケーションでトピック・ストリングに一致するものが保持されていて、 かつ、そのパブリケーションが永続であるか、そのパブリケーション作成以降ずっとキュー・マネージャーが終了していない場合です)。

このパターンで永続 サブスクリプションを使用することもできます。 管理永続サブスクリプションが使用される典型的な理由は、 エラーが起こることなくサブスクライバーよりも長く持続する サブスクリプションを確立することよりも、むしろ信頼性のためです。 管理サブスクリプション、 非管理サブスクリプション、永続サブスクリプション、非永続サブスクリプションに関連付けられたそれぞれ異なるライフサイクルについて詳しくは、関連するトピックのセクションを参照してください。

永続サブスクリプションは永続パブリケーションと、 非永続サブスクリプションは非永続パブリケーションと関連付けられることが多い ことは確かですが、サブスクリプションの永続性とパブリケーションの永続性との 間には必須の関係はありません。 永続性の 4 通りの組み合わせのすべてが 可能です。

ここで検討している管理非永続のケースでは、キュー・マネージャー がサブスクリプション・キューを作成し、そのキューはクローズされるときに パージされ、削除されます。 パブリケーションは、非永続サブスクリプションが クローズされるときにキューから除去されます。

このコードで例示されている、管理非永続のパターンにおける重要な面を 以下に示します。

1. オンデマンドのサブスクリプション: サブスクリプション・トピック・ストリングは動的です。 これはアプリケーションによって実行時に供給されます。
2. 自律キュー: サブスクリプション・キューは、自己定義および自己管理を行います。
3. 自律サブスクリプションのライフ・サイクル: 非永続 サブスクリプション は、サブスクライバー・アプリケーションが存続している間だけ存在します。
   - 永続 管理サブスクリプションを定義すると、 永続サブスクリプション・キューができ、 アクティブなサブスクライバー・プログラムがなくても、 そのキューにパブリケーションが保管され続けます。 キュー・マネージャー がキューを削除 (および、取り出されなかったパブリケーションをキューから消去) する のは、アプリケーションまたは管理者がサブスクリプションを削除することを選択した 後でのみです。 サブスクリプションの削除は、 管理コマンドを使用するか、MQCO_REMOVE_SUB オプションを指定してサブスクリプションをクローズすることで実行できます。
   - 永続サブスクリプションに対しては SubExpiry を設定することを 検討してください。これを設定すると、パブリケーションがキューに送信されるのが中止され、 サブスクライバーは、サブスクリプションを除去しその結果キュー・マネージャーがキューおよびキューに滞留しているパブリケーションを削除する 前に、滞留している任意のパブリケーションをコンシュームできます。
4. トピック・ストリングの柔軟なデプロイメント: 管理的に定義されるトピックを使用して サブスクリプションのルート部分を定義することによって、サブスクリプション・トピック管理 が単純化されます。 これによって、トピック・ツリーの ルート部分がアプリケーションに対して隠蔽されます。 ルート部分を隠すこと によって、アプリケーションをデプロイする際に、別のインスタンスまたは別のアプリケーションで作成された別のトピック・ツリーにオーバーラップするようなトピック・ツリーを そのアプリケーションが作成してしまうことがなくなります。
5. 管理されるトピック: 管理的に定義されたトピック・オブジェクトと最初の部分が 一致するトピック・ストリングを使用して、パブリケーションはトピック・オブジェクトの属性 に従って管理されます。
   - 例えば、トピック・ストリングの最初の部分が、クラスター化されたトピック・オブジェクト に関連付けられたトピック・ストリングと一致する場合、 サブスクリプションは、そのクラスターの他のメンバーからパブリケーションを受け取ることができます。
   - 管理的に定義されたトピック・オブジェクトとプログラムで 定義されたサブスクリプションを選択的にマッチングすることで、 両方の利点を結合することができます。 管理者はトピックの属性を指定し、プログラマーはトピックの管理には関与せずに動的にサブトピックを定義します。
   - トピックと関連付けられた属性を提供するトピック・オブジェクトの マッチングに使用されるのは、結果のトピック・ストリングであり、 sd.Objectname に指定されたトピック・オブジェクトであるとは 限りませんが、それらは結局は 1 つであり、同じであるのが普通です。 814 ページの『例 2: 可変トピックへのパブリッシャー』を参照してください。

この例では、サブスクリプションを永続的にすることによって、 サブスクライバーが MQCO_KEEP_SUB オプションを指定してサブスクリプションをクローズした後、 パブリケーションはサブスクリプション・キューに引き続き送信されます。 サブスクライバーがアクティブでなくても、キューはパブリケーションを受け取り続けます。 この動作をオーバーライドするには、 MQSO_PUBLICATIONS_ON_REQUEST オプションを指定してサブスクリプションを 作成し、MQSUBRQ を使用して、保存パブリケーションを要求します。

MQCO_RESUME オプションを指定してサブスクリプションを オープンすることによって、後でサブスクリプションを再開できます。

MQSUB によって戻される キュー・ハンドル Hobj には多くの用途があります。 例では、キュー・ハンドルは、サブスクリプション・キューの名前を照会するのに 使用されています。 管理キューは、デフォルトのモデル・キュー SYSTEM.NDURABLE.MODEL.QUEUE または SYSTEM.DURABLE.MODEL.QUEUE を使用してオープンされます。 サブスクリプションと関連付けられたトピック・オブジェクトのプロパティーに応じて、トピックごとに独自の永続および非永続のモデル・キューを 用意することによって、デフォルトをオーバーライドできます。

モデル・キューから継承する属性に関係なく、 追加のサブスクリプションを作成するために管理キュー・ハンドルを再使用することはできません。 また、管理キュー用の別のハンドルを、戻されたキュー名を使用して管理キューを 2 度目にオープンすることによって 取得することもできません。 キューは、排他的入力用に オープン済みであるかのように動作します。

非管理キューのほうが、管理キューよりも柔軟性があります。 例えば、 非管理キューを共用することや、複数のサブスクリプションを 1 つのキューに 定義することができます。 次の例は、サブスクリプションを非管理サブスクリプション・キューと組み合わせる方法を示しています。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

825 ページの図 73 では結果が示されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){                /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
}
```

図 *71.* 管理 *MQ* サブスクライバー - パート *1:* 宣言および パラメーター処理

以下の説明は、この例の宣言に関する追加コメントです。

MQHOBJ Hobj = MQHO_NONE;
非永続の管理サブスクリプション・キューを、パブリケーションを受け取るために明示的に オープンすることはできませんが、キュー・マネージャーがキューをオープンするときに 戻されるオブジェクト・ハンドル用に、ストレージを割り振る必要が あります。 ハンドルを MQHO_OBJECT に初期設定することは重要です。 これは、キュー・マネージャーがサブスクリプション・キューのキュー・ハンドルを戻す 必要があることをキュー・マネージャーに示します。

MQSD sd = {MQSD_DEFAULT};
MQSUB で使用される、新規サブスクリプション記述子。

MQCHAR48 qName;
この例ではサブスクリプション・キューについて知っている必要は ありませんが、サブスクリプション・キューの名前を照会しています。MQINQ バインディング は C 言語では少し扱いにくいものなので、例のこの部分を学習用に役立てることができます。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

図 *72.* 管理 *MQ* サブスクライバー *-* パート *2:* コード本体

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020    "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020    "
Received publication "150"
Completion code 0 and Return code 0
```

図 *73. MQ* サブスクライバー

以下の説明は、この例のコードについての追加コメントです。

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
topicName がヌルまたはブランク *(*デフォルト値*)* の 場合、解決されたトピック・ストリングを計算するのにトピック名は使用されません。

sd.ObjectString.VSPtr = topicString;
事前定義されたトピック・オブジェクトを単独で使用するのではなく、 この例では、プログラマーは MQSUB によって結合された トピック・オブジェクトとトピック・ストリングを供給しています。 このトピック・ストリングは MQCHARV 構造体であることに注意してください。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQCHARV フィールドの長さを設定する代替方法。

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
トピック・ストリングの定義後、sd.Options フラグは細心の注意 を必要とします。 多くのオプションがありますが、この例では、最もよく使用されるオプションのみを指定しています。 他のオプションではデフォルト値を使用しています。

1. サブスクリプションは非永続 です。言い換えると、 このサブスクリプションの存続時間は、アプリケーション内でオープンしているサブスクリプションの存続期間なので、 MQSO_CREATE フラグを設定します。 読みやすくするために、*(*デフォルト*)* MQSO_NON_DURABLE フラグ も設定できます。
2. MQSO_CREATE を補完するのは、MQSO_RESUME です。 これら両方のフラグを一緒に設定することができます。キュー・マネージャーは、 新規サブスクリプションの作成または既存サブスクリプションの再開 のどちらか適当な方を行います。 ただし、MQSO_RESUME を 指定する場合は、再開するサブスクリプションがない場合でも、sd.SubName の MQCHARV 構造体の初期設定も 行う必要があります。 SubName の初期設定が 失敗すると、戻りコード 2440: MQRC_SUB_NAME_ERROR が MQSUB から出されます。

   注 : MQSO_RESUME は、 非永続管理サブスクリプションの場合は常に無視されます。しかし、 これを指定して、sd.SubName の MQCHARV 構造体を初期設定しないと、 エラーの原因になります。
3. さらに、サブスクリプションがどのようにオープンされるのかに影響する 3 番目の フラグ、MQSO_ALTER があります。 正しい許可が付与されている場合は、再開されたサブスクリプションのプロパティーは、 MQSUB に指定された他の属性と一致するように変更されます。

   注 : MQSO_CREATE、MQSO_RESUME、および MQSO_ALTER のうち少なくとも 1 つの フラグが指定されていなければなりません。 Options (MQLONG) を参照してください。 826 ページの『例 3: 非管理 MQ サブスクライバー』 に、これら 3 つのフラグのすべてを使用する例があります。
4. キュー・マネージャーが自動的にサブスクリプションを管理するよう、MQSO_MANAGED を 設定します。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
オプションで、ヌル終了ストリング用の MQCHARV 長さの 設定を省略して、代わりにヌル終了文字フラグを使用します。

sd.ResObjectString.VSPtr = resTopicStr;
結果のトピック・ストリングは、プログラム内の最初の printf で エコー出力されます。 解決されたストリングをプログラムに返すように MQCHARV ResObjectString for IBM MQ をセットアップします。

注: resTopicStringBuffer は memset(resTopicStr, 0, sizeof(resTopicStrBuffer)) でヌルに初期設定されています。 戻されるトピック・ストリングは、末尾ヌルで終了しません。

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
sd.ResObjectString のバッファー・サイズを、実際の サイズより 1 だけ小さい値に設定します。 これによって、解決後のトピック・ストリング がバッファー全体を満たす場合に、用意したヌル終了文字が上書きされないようになります。

注: トピック・ストリングが sizeof(resTopicStrBuffer)-1 より 長い場合、エラーは戻されません。 VSLength > VSBufSiz の場合でも、 sd.ResObjectString.VSLength に戻される長さは完全なストリングの長さであり、必ずしも戻されるストリングの長さではありません。 sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz をテストして、トピック・ストリングが完全であることを確認します。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQSUB 機能は、サブスクリプションを作成します。 非永続の場合、その名前には関心はないでしょうが、IBM MQ エクスプローラーで状況を検査することができます。 入力 として sd.SubName パラメーターを指定すること ができますが、それは名前が分かっている場合であり、当然、他のサブスクリプションと 名前が衝突しないようにする必要があります。

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
サブスクリプションとサブスクリプション・キューの両方のクローズは、オプションです。 この例では、サブスクリプションはクローズされますが、キューはクローズされません。 MQCLOSE MQCO_REMOVE_SUB オプション は、このケースではサブスクリプションは非永続であるため、デフォルトです。 MQCO_KEEP_SUB の使用はエラーです。

注: サブスクリプション・キュー は MQSUB ではクローズされず、ハンドル Hobj は、 MQCLOSE または MQDISC でキューがクローズされるまでは有効です。 アプリケーションが途中で終了してしまった場合、キューおよびサブスクリプション は、アプリケーションの終了から少し後に、キュー・マネージャーによってクリーンアップされます。

関連概念
818 ページの『例 1: MQ パブリケーション・コンシューマー』
MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM MQ メッセージ・コンシューマーです。

826 ページの『例 3: 非管理 MQ サブスクライバー』
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 非管理サブスクリプションは、アプリケーションが責任を持つ場所です。 サブスクリプションが保管されるキューを指定します。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

810 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

例 *3:* 非管理 *MQ* サブスクライバー
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 非管理サブスクリプションは、アプリケーションが責任を持つ場所です。 サブスクリプションが保管されるキューを指定します。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

非管理パターンは、非永続 サブスクリプションよりも 永続 サブスクリプションに関連付けられるほうが一般的です。 通常、非管理サブスクライバーによって 作成されるサブスクリプションのライフ・サイクルは、 サブスクライブを行うアプリケーション自体のライフ・サイクルとは無関係です。 サブスクリプションを 永続的にすることによって、サブスクリプションは、サブスクライブを行うアプリケーションがアクティブ でない場合でも、パブリケーションを受け取ります。

永続管理 サブスクリプションを作成しても同じ結果を 得ることができますが、アプリケーションによっては、キューおよびメッセージに関する制御と柔軟性を、 管理サブスクリプションの場合よりも必要とします。 永続管理サブスクリプションの場合、キュー・マネージャーは、サブスクリプション・トピックに一致するパブリケーション 用に永続キューを作成します。 そのキューおよび関連付けられたパブリケーションは、 サブスクリプションが削除されるときにキュー・マネージャーによって削除されます。

永続管理 サブスクリプションが使用されるのは、 アプリケーションとサブスクリプションのライフ・サイクルが基本的に同じであるが、 それを保証するのは難しいという場合が典型的です。 サブスクリプションを永続的にし、パブリッシャーに永続パブリケーションを 作成させるようにすると、もしキュー・マネージャーまたはサブスクライバーが途中で 終了してリカバリーが必要になっても、失われるメッセージはありません。

非 JMS アプリケーション、つまり、共有サブスクリプションを使用していない JMS アプリケーションの場合、キュー・マネージャーは、サブスクライバー用の永続管理サブスクリプション・キューを、キューの共用処理は可能でないような方法で、暗黙的にオープン します。 また、ご使用のアプリケーションが JMS 共有サブスクリプションを使用していない限り、各管理キューに対して複数のサブスクリプションを 作成することはできず、キューの名前に対する制御を十分に持っていないため、キューの 管理は容易ではありません。 以上の理由から、非管理 MQ サブスクライバー が管理 MQ サブスクライバーに比べて、 永続サブスクリプションを必要とするアプリケーションに適しているかどうかを検討してください。

833 ページの図 76 の コードは、非管理永続サブスクリプションのパターンを示しています。 例示の目的のため、このコードは非管理の非永続サブスクリプションも 作成します。 この例は、次のパターン・ファセットを 示しています。

- オンデマンドのサブスクリプション: サブスクリプション・トピック・ストリングは動的です。 これらはアプリケーションによって実行時に提供されます。
- 単純化されたサブスクリプション・トピック管理: 管理的に定義されるトピックを使用して、 サブスクリプション・トピック・ストリングのルート部分を定義することによって、 サブスクリプション・トピック管理が単純化されます。 これは、トピック・ツリーのルート部分を アプリケーションに対して隠します。 ルート部分を隠すことによって、 1 つのサブスクライバーを複数の異なるトピック・ツリーにデプロイできます。
- 柔軟なサブスクリプション管理: サブスクリプションを管理的に定義するか、 サブスクライバー・プログラム内でオンデマンドで作成することができます。 管理的に作成されたサブスクリプションとプログラムで作成されたサブスクリプションとの間には、 どのようにサブスクリプションが作成されたのかを示す属性を除いて、 違いはありません。 第 3 のタイプのサブスクリプションがあり、それは、サブスクリプションの配布のためにキュー・マネージャーによって自動的に 作成されるものです。 すべてのサブスクリプションは、IBM MQ エクスプローラーで表示されます。
- サブスクリプションとキューの柔軟な関連付け: MQSUB 機能によって、事前定義された ローカル・キューがサブスクリプションと関連付けられます。 サブスクリプションとキューを関連付けるための MQSUB の使用には、次のように いくつかの方法があります。
  - サブスクリプションを、既存のサブスクリプションが ない キュー (MQSO_CREATE + (Hobj from MQOPEN)) に関連付けます。
  - 新規 サブスクリプションを、既存のサブスクリプション MQSO_CREATE + (Hobj from MQOPEN) を持つキューに関連付けます。
  - 既存のサブスクリプションを別のキュー MQSO_ALTER + (Hobj from MQOPEN) に移動します。
  - 既存のキュー、MQSO_RESUME + (Hobj = MQHO_NONE)、または MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription) に関連付けられている既存のサブスクリプションを再開します。

- MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の 組み合わせ方を変えることによって、異なる値の sd.Options を使用する複数 バージョンの MQSUB をコーディングせずに、さまざまな異なる入力状態の サブスクリプションとキューに対応できます。
- あるいは、 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の特定の選択をコーディングすると、キュー・マネージャーはエラー (829 ページの表 123) を返します。 MQSUB への入力として提供されたサブスクリプションとキューの状態が sd.Options の値と矛盾する場合。 836 ページの図 82 は、 サブスクリプション X に対して、sd.Options フラグの個々の設定値を変え、3 種類の異なるオブジェクト・ハンドルを 渡して MQSUB を発行した場合の 結果を示します。

832 ページの図 75 に示されている プログラム例へのさまざまな入力を検討して、これらの異なる種類のエラーを理解してください。 表にリストされているケースに含まれていない 1 つの一般的なエラー RC = 2440 は、サブスクリプション名エラーです。 これが発生するのは、通常、ヌルまたは 無効なサブスクリプション名を MQSO_RESUME または MQSO_ALTER で渡すことが原因です。

- マルチプロセッシング: パブリケーションを読み取る作業を多くのコンシューマーと共有できます。 すべてのパブリケーションが、サブスクリプション・トピックと関連付けられた単一のキューに入れられます。 コンシューマーは、MQOPEN を使用してキューを直接オープンするのか、あるいは MQSUB を使用してサブスクリプションを再開するのかを選択できます。
- サブスクリプション集中: 複数のサブスクリプションを同じキューに 作成することができます。 この機能の使用には注意が必要です。これによって、 サブスクリプションのオーバーラップが発生したり、同じパブリケーションを複数回受け取る ことになる可能性があるためです。 MQSO_GROUP_SUB オプションを指定すると、 サブスクリプションのオーバーラップによって引き起こされるパブリケーション重複はなくなります。
- サブスクライバーとコンシューマーの分離: 例で示された 3 つのコンシューマー・モデル のほかに、コンシューマーをサブスクライバーから分離 するモデルがあります。 これは、非管理 MQ サブスクライバーの変形ですが、同じプログラム内で MQOPEN と MQSUB を 発行するのではなく、1 つのプログラムがパブリケーションにサブスクライブし、別のプログラムがそれら をコンシュームします。 例えば、サブスクライバーがパブリッシュ/サブスクライブ・クラスターの 一部であり、コンシューマーはキュー・マネージャー・クラスターの外部でキュー・マネージャーに 接続されているといった状況が考えられます。 リモート・キュー定義としてサブスクリプション・キューを 定義することによって、コンシューマーは、標準分散キューイングを通して パブリケーションを受け取ります。

これらのオプションの組み合わせを使用してコードを単純化する予定の場合は特に、 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の動作を理解することが重要です。 829 ページの表 123 を見てください。 この表には、MQSUB に異なるキュー・ハンドルを渡した場合の 結果と、834 ページの図 77 から 836 ページの図 82 に示されたプログラム例の実行結果が示されています。

表の構成に使用されるシナリオには、1 つのサブスクリプション X と 2 つのキュー A と B があります。 サブスクリプション名パラメーター sd.SubName は、X に設定され、キュー A に付属するサブスクリプションの名前に設定されます。 キュー B にはサブスクリプションが付属しません。

829 ページの表 123 では、MQSUB は、サブスクリプション X およびキュー・ハンドルをキュー A に渡されます。 サブスクリプション・オプションの結果は以下のとおりです。

- MQSO_CREATE は失敗します。なぜなら、 キュー・ハンドルに対応するキュー A には既に X へのサブスクリプションがあるためです。 この動作と正常な呼び出しを比較します。 正常な呼び出しは、キュー B がそれに接続している X へのサブスクリプションを持たないため、成功します。
- MQSO_RESUME が成功するのは、キュー・ハンドルが、X へのサブスクリプションを既に持っているキュー A に対応するためです。 これに対して、呼び出しは、サブスクリプション X がキュー A 上に存在しない場合は失敗します。
- MQSO_ALTER は、サブスクリプションおよびキューのオープンに関しては MQSO_RESUME と 同じように動作します。 しかし、MQSUB に渡されたサブスクリプション記述子内に 含まれている属性がサブスクリプションの属性と異なっている場合、MQSO_RESUME は 失敗しますが、MQSO_ALTER は、プログラム・インスタンスが属性を変更する許可を 持っていれば、成功します。 サブスクリプション内のトピック・ストリング を変更できないことに注意してください。しかし、 MQSUB は、エラーを戻すのではなく、サブスクリプション記述子内のトピック名とトピック・ストリングの値を 無視し、既存のサブスクリプション内の値を使用します。

次に、MQSUB がサブスクリプション X に渡され、キュー・ハンドルがキュー B に渡される場所である 829 ページの表 123 を確認します。 サブスクリプション・オプションの結果は以下のとおりです。

- MQSO_CREATE は成功し、キュー B に サブスクリプション X を作成します。これは キュー B 上の新しいサブスクリプションだからです。
- MQSO_RESUME は失敗します。 MQSUB はキュー B 上でサブスクリプション X を探して見つかりませんが、「*RC = 2428* - サブスクリプション *X* は存在しない」ではなく、「*RC = 2019* - サブスクリプション・キューがキュー・オブジェクト・ハンドルと一致しない」を返します。 3 番目のオプション MQSO_ALTER の動作は、 この予期しないエラーの理由を示唆します。 MQSUB は、 キュー・ハンドルが、サブスクリプションを持つキューをポイントしていることを予期しています。 MQSUB は初めにこれを検査してから、sd.SubName 内で指定されているサブスクリプションが存在するかどうかを検査します。
- MQSO_ALTER は成功し、 サブスクリプションをキュー A からキュー B に移動します。

キュー A にあるサブスクリプションのサブスクリプション名が sd.SubName 内のサブスクリプション名と一致しないケースは、この表には示されていません。 この場合の 呼び出しは *RC = 2428* - サブスクリプション *X* はキュー *A* に存在しない で失敗します。

表 *123.* さまざまなキュー・ハンドルとサブスクリプションの組み合わせでの *MQSUB* からのエラー

キュー・ハンドル	**キュー A** **サブスクリプション X** **キュー B** **サブスクリプションなし**	**キュー A** **サブスクリプションなし** **キュー B** **サブスクリプションなし**
キュー *A* の Hobj が MQSUB に渡される	**MQSO_CREATE** RC = 2432 - サブスクリプション X は既にキュー A に存在する **MQSO_RESUME** キュー A にあるサブスクリプション X を再開する **MQSO_ALTER** キュー A にあるサブスクリプション X を再開し、許可された変更を行う	**MQSO_CREATE** キュー A にサブスクリプション X を作成する **MQSO_RESUME** RC = 2428 - サブスクリプション X はキュー A に存在しない **MQSO_ALTER** RC = 2428 - サブスクリプション X はキュー A に存在しない
キュー *B* の Hobj が MQSUB に渡される	**MQSO_CREATE** キュー B に新規サブスクリプション X を作成する **MQSO_RESUME** RC = 2019 - サブスクリプション・キューがキュー・オブジェクト・ハンドルと一致しない **MQSO_ALTER** サブスクリプション X をキュー A からキュー B に移動する	**MQSO_CREATE** キュー B に新規サブスクリプション X を作成する **MQSO_RESUME** RC = 2428 - サブスクリプション X はキュー B に存在しない **MQSO_ALTER** RC = 2428 - サブスクリプション X はキュー B に存在しない

表 *123.* さまざまなキュー・ハンドルとサブスクリプションの組み合わせでの *MQSUB* からのエラー *(*続き*)*

キュー・ハンドル	**キュー A** **サブスクリプション X** **キュー B** **サブスクリプションなし**	**キュー A** **サブスクリプションなし** **キュー B** **サブスクリプションなし**
MQHO_NONE が MQSUB に渡される	**MQSO_CREATE** RC = 2019 - 不正なオブジェクト・ハンドル: 管理サブスクリプションを作成し、管理キューを作成するには、MQSO_MANAGED フラグを設定 してください **MQSO_RESUME** キュー A にあるサブスクリプション X を再開し、キュー A の Hobj を戻す **MQSO_ALTER** キュー A にあるサブスクリプション X を再開し、キュー A の Hobj を戻し、許可された変更を行う	**MQSO_CREATE** RC = 2019 - 不正なオブジェクト・ハンドル: 管理サブスクリプションを作成し、管理キューを作成するには、MQSO_MANAGED フラグを設定 してください **MQSO_RESUME** RC = 2428 - サブスクリプション X なし **MQSO_ALTER** RC = 2019 - 不正なオブジェクト・ハンドル: キュー A または B なし

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

図 *74.* 非管理 *MQ* サブスクライバー - パート *1:* 宣言

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
}
```

図 *75.* 非管理 *MQ* サブスクライバー *-* パート *2:* パラメーター処理

以下の説明は、この例でのパラメーター処理に関する追加コメントです。

switch((argv[5][0]))
例でデフォルトで使用されている MQSUB オプション設定の一部をオーバーライドする効果をテストするために、パラメーター 5 に A lter | C reate | R esume を入力することを選択できます。 この例で使用されているデフォルト設定は MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE です。

注 : MQSO_DURABLE を設定せずに MQSO_ALTER または MQSO_RESUME を 設定することは誤りであり、sd.SubName の設定は必須で、 再開または変更が可能なサブスクリプションを指していなければなりません。

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
デフォルトのサブスクリプション・キュー STOCKTICKER がヌル・ストリング で置き換えられた場合、MQSO_CREATE が設定されている 限り、この例は MQSO_MANAGED フラグを設定し、動的サブスクリプ

ション・キューを作成します。 5 番目のパラメーターに Alter or Resume が設定されている場合、例の動作は subscriptionName の値によって異なります。

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;

デフォルトのサブスクリプション IBMSTOCKPRICESUB が ヌル・ストリングで置き換えられた場合、この例は MQSO_DURABLE フラグを除去します。 他のパラメーターをデフォルト値にしてこの例を実行する 場合、STOCKTICKER 向けの 追加の一時サブスクリプションが作成され、重複するパブリケーションを受け取ります。 この例をパラメーターなしで次に実行 すると、1 つのみのパブリケーションをもう一度受け取ります。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
                gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

図 *76.* 非管理 *MQ* サブスクライバー *-* パート *3:* コード本体

以下の説明は、この例でのコードに関する追加コメントです。

if (strlen(subscriptionQueue))
サブスクリプション・キュー名がない場合、この例は Hobj の値 として MQHO_NONE を使用します。

MQOPEN(...);
サブスクリプション・キューがオープンされ、キュー・ハンドルが Hobj に保管されます。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQOPEN から渡された Hobj (または、サブスクリプション・キュー名がない場合は MQHO_NONE) を 使用して、サブスクリプションがオープンされます。 非管理キューの再開は、MQOPEN で明示的にオープンせずに、実行できます。

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
サブスクリプションがサブスクリプション・ハンドルを使用してクローズされます。 サブスクリプションが 永続的かどうかに基づいて、サブスクリプション は暗黙の MQCO_KEEP_SUB または MQCO_REMOVE_SUB で クローズされます。 永続サブスクリプションを MQCO_REMOVE_SUB でクローズすることはできますが、非永続サブスクリプションを MQCO_KEEP_SUB でクローズすることはできません。 MQCO_REMOVE_SUB のアクションは、 サブスクリプションを除去することであり、それによって、サブスクリプション・キューへのパブリケーションの送信は 停止されます。

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
サブスクリプションが非管理の場合、特別なアクションは実行されません。 キュー が管理されていて、サブスクリプションが 明示的または暗黙的な MQCO_REMOVE_SUB でクローズされた場合、すべての パブリケーションがキューから消去され、この時点でキューは削除されます。

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
受信したメッセージが、サブスクリプションのメッセージであることを確認してください。

この例の出力は、パブリッシュ/サブスクライブの各局面を示します。

834 ページの図 77 では、 NYSE/IBM/PRICE トピックに 130 をパブリッシュすることから例が開始します。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *77. NYSE/IBM/PRICE* に *130* をパブリッシュする

834 ページの図 78 では、 デフォルトのパラメーターを使用して例が実行され、 保存パブリケーション 130 を受け取ります。 836 ページの図 82 に示されているように、指定されたトピック・オブジェクトおよびトピック・ストリングは無視されます。 トピック・オブジェクトおよびトピック・ストリングは、サブスクリプション・オブジェクトがあれば、常にそこから 取得され、トピック・ストリングは不変です。 この例の 実際の動作は、MQSO_CREATE、MQSO_RESUME、および MQSO_ALTER がどのように選択され、組み合わされるのかに基づきます。 この例で選択されているオプションは MQSO_RESUME です。

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

図 *78.* 保存パブリケーションを受け取る

(835 ページの図 79) では、 永続サブスクリプションが既に保存パブリケーションを受け取ったため、パブリケーションは 受け取られません。 この例では、キュー名なしで、サブスクリプション名のみを指定することによって、サブスクリプションが 再開されています。 もしキュー名が指定されている場合は、最初にキューが オープンされ、ハンドルが MQSUB に渡されます。

注 : MQINQ からの 2038 エラー の原因は、MQSUB による STOCKTICKER の暗黙的な MQOPEN に MQOO_INQUIRE オプションが含まれていないことです。 キューを明示的にオープンすれば、MQINQ から 2038 戻りコードが戻されるのを回避できます。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

図 *79.* サブスクリプションの再開

835 ページの図 80 では、 この例は、宛先として STOCKTICKER を使用して、非永続、非管理のサブスクリプションを 作成します。 これは新規サブスクリプションであるため、保存パブリケーションを受け取ります。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

図 *80.* 新しい非管理、非永続のサブスクリプションで保存パブリケーションを 受け取る

835 ページの図 81 では、サブスクリプションのオーバーラップを説明するため、保存パブリケーションを変更する、別のパブリケーションが送信されます。 次に、新規の非永続、非管理のサブスクリプションが、サブスクリプション名を指定せずに作成されます。 保存パブリケーションは 2 回受け取られます。1 回は新規サブスクリプション用で、もう 1 回は、STOCKTICKER キューで依然としてアクティブな永続的 IBMSTOCKPRICESUB サブスクリプション用です。 この 例は、サブスクリプションを持つのはキューであり、 アプリケーションではないことを示しています。 アプリケーションのこの呼び出しでは IBMSTOCKPRICESUB サブスクリプション を参照しないにも関わらず、アプリケーションは、パブリケーションを 2 回受け取ります。 1 回は、管理的に作成された永続サブスクリプションからで、 もう 1 回は、アプリケーション自体で作成された非永続サブスクリプション からです。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

図 *81.* オーバーラップしているサブスクリプション

836 ページの図 82 の 例は、新規トピック・ストリングと既存のサブスクリプションの 指定が、サブスクリプションの変更という結果にならないことを示しています。

1. 最初のケースでは、Resume は、予想の通りに、既存の サブスクリプションを再開し、変更されたトピック・ストリングを無視します。
2. 2 番目のケースでは、 Alter はエラー RC = 2510, Topic not alterable を発生させます。

3. 3 番目の例では、 Create によってエラー RC = 2432, Sub already exists が発生します。

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

図 *82.* サブスクリプション・トピックは変更できない

関連概念

818 ページの『例 1: MQ パブリケーション・コンシューマー』
MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM MQ メッセージ・コンシューマーです。

821 ページの『例 2: 管理 MQ サブスクライバー』
管理 MQ サブスクライバーは、大部分のサブスクライバー・アプリケーションが優先的に使用するパターンです。 管理対象サブスクリプションは、IBM MQ がサブスクリプションを処理するするもので、登録および登録解除を行います。 この例は、キュー、トピック、またはサブスクリプションの管理的な定義を必要としません。

810 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

パブリッシュ/サブスクライブのライフ・サイクル

パブリッシュ/サブスクライブ・アプリケーションの設計時には、トピック、サブスクリプション、サブスクライバー、パブリケーション、パブリッシャー、およびキューの ライフ・サイクルを考慮してください。

オブジェクト (例えばサブスクリプション) のライフ・サイクルは、 作成で始まり、削除で終わります。 他の状態や変更が途中に含まれる場合も あります。例えば、一時的な中断、親トピックおよび子トピックの保有、 満了と削除などです。

キューなどの IBM MQ オブジェクトは、管理的に作成されるか、Programmable Command Format (PCF) を使用して管理プログラムによって作成されるのが、従来からの一般的な方法です。 パブリッシュ/サブスクライブで異なるのは、サブスクリプションの作成と削除を行うための MQSUB および MQCLOSE API verb が用意されていること、キューを作成して削除するだけでなく、 コンシュームされていないメッセージをクリーンアップするという、管理サブスクリプションの概念があること、 および、管理的に作成されたトピック・オブジェクトと、プログラムでまたは管理的に作成されたトピック・ストリングとの 間に関連があることです。

こうした豊富な機能によって、広範囲に及ぶパブリッシュ/サブスクライブ要件を満たすことができ、パブリッシュ/サブスクライブ・アプリケーションのいくつかの共通パターンの設計を単純化することができます。 例えば、管理サブスクリプションは、それを作成したプログラムの間だけ存続するように意図されているサブスクリプションの、プログラミングと管理の両方を単純化します。 非管理サブスクリプションは、サブスクライブするパブリケーションとコンシュームするパブリケーションの間の接続がより緩いプログラミングを単純化します。 サブスクリプションを中央に作成することは、コンシューマーへのルーティング・パブリケーション・トラフィックのパターンが集中化された制御のモデルに基づいている場合に有効です。 例えば、フライト情報を自動ゲートに送信して、一方ではプログラムで作成されたサブスクリプションが使用されることがあります。このサブスクリプションは、ゲート・スタッフがそのフライトの乗客レコードのサブスクライブを行う場合に、ゲートでフライト番号を入力することで使用されます。

この最後の例では、管理対象の永続サブスクリプションが適切に管理されている場合があります。それは、サブスクリプションが非常に頻繁に作成され、ゲートが閉じ、サブスクリプションをプログラマチックに削除できるときに、明確なエンドポイントを持つためです。また、何らかの理由でゲート・サブスクライバー・プログラムがダウンによって乗客レコードが失われるのを防ぐため、耐久性のあるものとなってい

ます。[8] ゲートへの乗客レコードのパブリッシュを始めるために、理想的な設計は、ゲート番号を使った乗客レコードのサブスクライブとゲート番号を使用したゲート搭乗開始イベントの公開の両方を行うゲート・アプリケーション用のものです。 パブリッシャーは、ゲート搭乗開始イベントに応えて、 乗客レコードをパブリッシュします。それらの レコードは、その後、必要とされる他の関係者にも送信されることが考えられます。例えば、 フライトが実施されたことを記録するために請求書作成部門に送られたり、 乗客の携帯電話にゲート番号をテキストで通知するために顧客サービスに送られます。

中央管理されるサブスクリプションは、各ゲートごとに事前定義されたキューを使用して乗客リストをゲートに転送する、 永続非管理モデルを使用できます。

パブリッシュ/サブスクライブのライフ・サイクルについての以下の 3 つの例では、管理非永続サブスクライバー、管理永続サブスクライバー、および非管理永続サブスクライバー と、サブスクリプション、トピック、キュー、パブリッシャー、キュー・マネージャーとの相互作用を示し、 管理プログラムとサブスクライバー・プログラム間でどのように操作が分配されるのかも 示します。

管理非永続サブスクライバー

838 ページの図 83 は、管理非永続サブスクリプションを作成し、サブスクリプションで識別されるトピックにパブリッシュされる 2 つのメッセージを取得して、終了するアプリケーションを示しています。 イタリック体のぼかしフォントでラベルが付いている、矢印が点線の相互作用は、 暗黙的なものです。

注意するべき点がいくつかあります。

1. このアプリケーションは、既にパブリッシュが 2 回行われたトピックにサブスクリプションを 作成します。 サブスクライバーは、最初のパブリケーションを受け取るときに、現在は保存パブリケーションである *2* 番目 のパブリケーションを 受け取ります。
2. キュー・マネージャーは、トピックに対するサブスクリプションを作成するだけでなく、一時サブスクリプション・キューも 作成します。
3. サブスクリプションには有効期限があります。 サブスクリプションが満了すると、トピックにあるパブリケーションはこのサブスクリプションにはもう送信されません が、サブスクライバーは、サブスクリプションが満了する前にパブリッシュされたメッセージを 引き続き取得します。 パブリケーションの満了は、サブスクリプションの満了に影響を受けません。
4. 4 番目のパブリケーションは、サブスクリプション・キューには入れられず、 従って最後の `MQGET` はパブリケーションを戻しません。
5. サブスクライバーは自身のサブスクリプションをクローズしますが、キューまたはキュー・マネージャー への接続はクローズしません。
6. キュー・マネージャーは、アプリケーションが終了した少し後に、クリーンアップを実行します。 サブスクリプションは管理非永続なので、サブスクリプション・キュー は削除されます。

[8] パブリッシャーは、他の起こりうる障害を回避するために、乗客レコードを持続メッセージとして送信する必要があります。

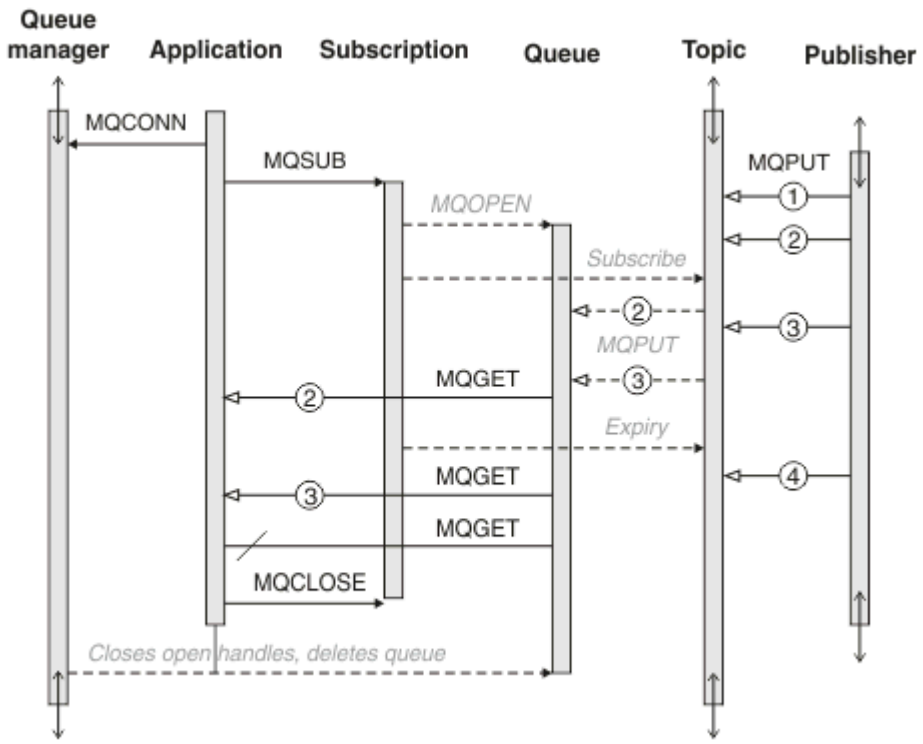


図 *83.* 管理非永続サブスクライバーのライフライン

管理永続サブスクライバー

管理永続サブスクライバー は、前の例をさらに 1 ステップ進めて、管理サブスクリプション がサブスクライブ・アプリケーションの終了および再始動の後も存続 することを示します。

注意するべき新しい点がいくつかあります。

1. この例では、前のとは違って、パブリケーション・トピックは、 サブスクリプション内に定義される前は、存在していませんでした。
2. サブスクライバーは、初めて終了する場合、 オプション MQCO_KEEP_SUB を指定してサブスクリプションをクローズします。 これは、管理永続サブスクリプション を暗黙的にクローズするデフォルトの動作です。
3. サブスクライバーがサブスクリプションを再開すると、サブスクリプション・キュー は再オープンされます。
4. 再オープンする前にキューに置かれた新規パブリケーション 2 は、サブスクリプションが除去された後でも、MQGET に対して使用可能 になっています。

   サブスクリプションが永続的であっても、パブリッシャーが送信したすべてのメッセージをサブスクライバーが確実に 受け取ることができるのは、サブスクリプションが永続サブスクリプションであり、メッセージが持続メッセージであるという両方 の条件が満たされる 場合のみです。 メッセージの持続性は、パブリッシャーによって 送信されるメッセージの MQMD 内の Persistent フィールド の設定に依存します。 サブスクライバーがこれを制御することはできません。
5. フラグ MQCO_REMOVE_SUB を設定してサブスクリプションをクローズすると、 サブスクリプションは除去され、それ以上のパブリケーションはサブスクリプション・キューに 置かれなくなります。 サブスクリプション・キューがクローズされると、 キュー・マネージャーは、未読のパブリケーション 3 を 除去した後、キューを削除します。 このアクションは、管理的にサブスクリプションを 削除することと等価です。

注 : キューを手動で削除しないでください。 また、MQCLOSE をオプション MQCO_DELETE または MQCO_PURGE_DELETE を指定して発行しないでください。 管理サブスクリプションの可視のインプリメンテーション詳細は、サポートされる IBM MQ インターフェースには含まれていません。 キュー・マネージャー管理は、完全な制御を持っていない場合、サブスクリプションを確実に制御することはできません。

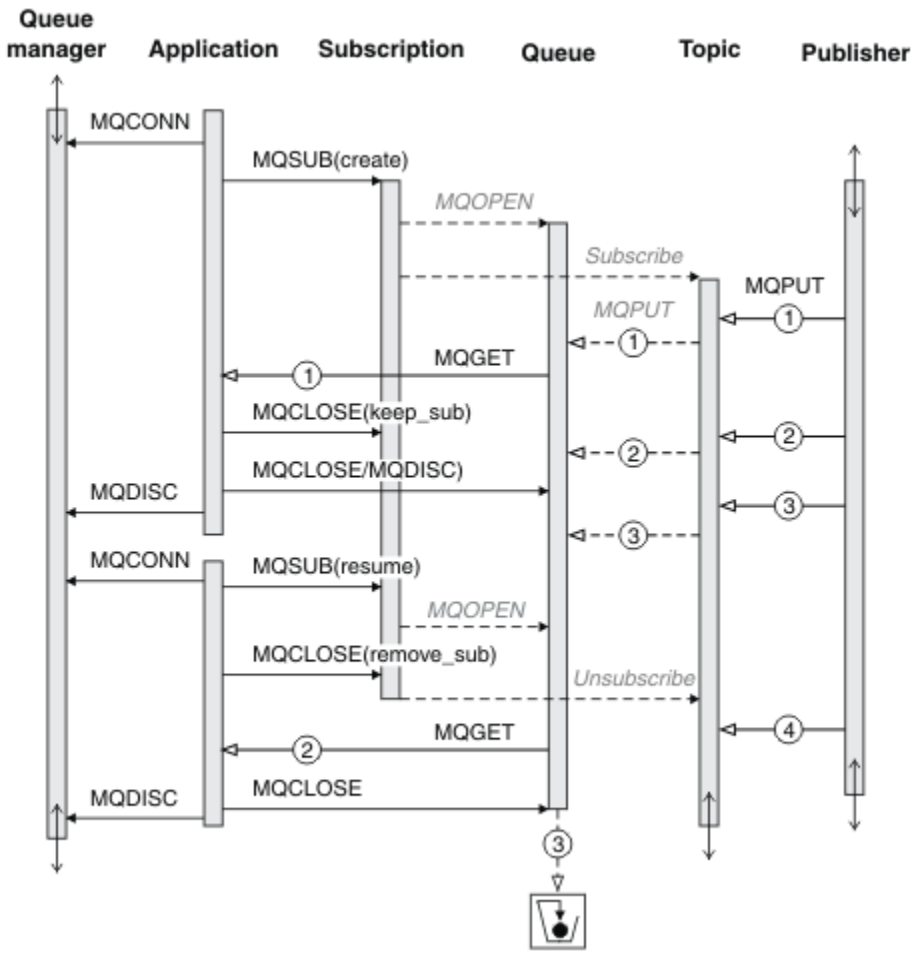


図 *84.* 管理永続サブスクライバーのライフライン

非管理永続サブスクライバー

3 番目の 例である非管理永続サブスクライバーには、管理者が追加されます。 これは、 管理者とパブリッシュ/サブスクライブ・アプリケーションとの相互作用 がどのように行われるのかを示す良い例です。

注意するべき点を以下にリストします。

1. パブリッシャーは、メッセージ 1 をトピックに 書き込みます。これは後で、サブスクリプション用に使用されるトピック・オブジェクトと関連付けられます。 トピック・オブジェクトは、ワイルドカードを使用してパブリッシュされたトピックと一致するトピック・ストリングを定義します。
2. トピックには保存パブリケーションが 1 つあります。
3. 管理者は、トピック・オブジェクト、キュー、およびサブスクリプションを作成します。 トピック・オブジェクトとキューは、サブスクリプションより前に定義されている必要があります。
4. アプリケーションは、サブスクリプションと関連付けられたキューを オープンし、MQSUB にキューのハンドルを渡します。 あるいは、その代わりに、 単にサブスクリプションをオープンし、キュー・ハンドル MQHO_NONE を渡すこともできます。 逆は正しくありません。 サブスクリプション名なしでキュ

ー・ハンドルだけを渡すことでサブスクリプションを再開することは できません。1 つのキューに複数のサブスクリプションがある可能性があるからです。

5. アプリケーションは、サブスクリプションを初めてオープンする場合でも、オプション MQSO_RESUME を 使用してサブスクリプションをオープンします。 管理的に作成されたサブスクリプション が再開されます。
6. サブスクライバーは、保存パブリケーション 1 を受け取ります。 パブリケーション 2 は、サブスクライバーがパブリケーションをどれも 受け取らないうちにパブリッシュされたにも関わらず、サブスクリプションが開始した後に パブリッシュされ、サブスクリプション・キューにある 2 番目のパブリケーションになっています。

   注 : 保存パブリケーション は、持続メッセージとしてパブリッシュされない場合、 キュー・マネージャーが再始動した後は失われます。
7. この例では、サブスクリプションは永続的です。 プログラムが 非管理非永続サブスクリプションを作成することが可能です。 これは管理者が実行できることでないことは明白です。
8. サブスクリプションをクローズする際のオプション MQCO_REMOVE_SUB の 効果は、管理者が削除したかのように、サブスクリプションを 削除することです。 これによって、それ以上のパブリケーションはキューに送信されなくなりますが、管理 永続サブスクリプションとは違って、キューがクローズされている場合であっても、既にキューにあるパブリケーションには影響しません。
9. 管理者は、後で残りのメッセージ 3 を削除し、 キューを削除します。

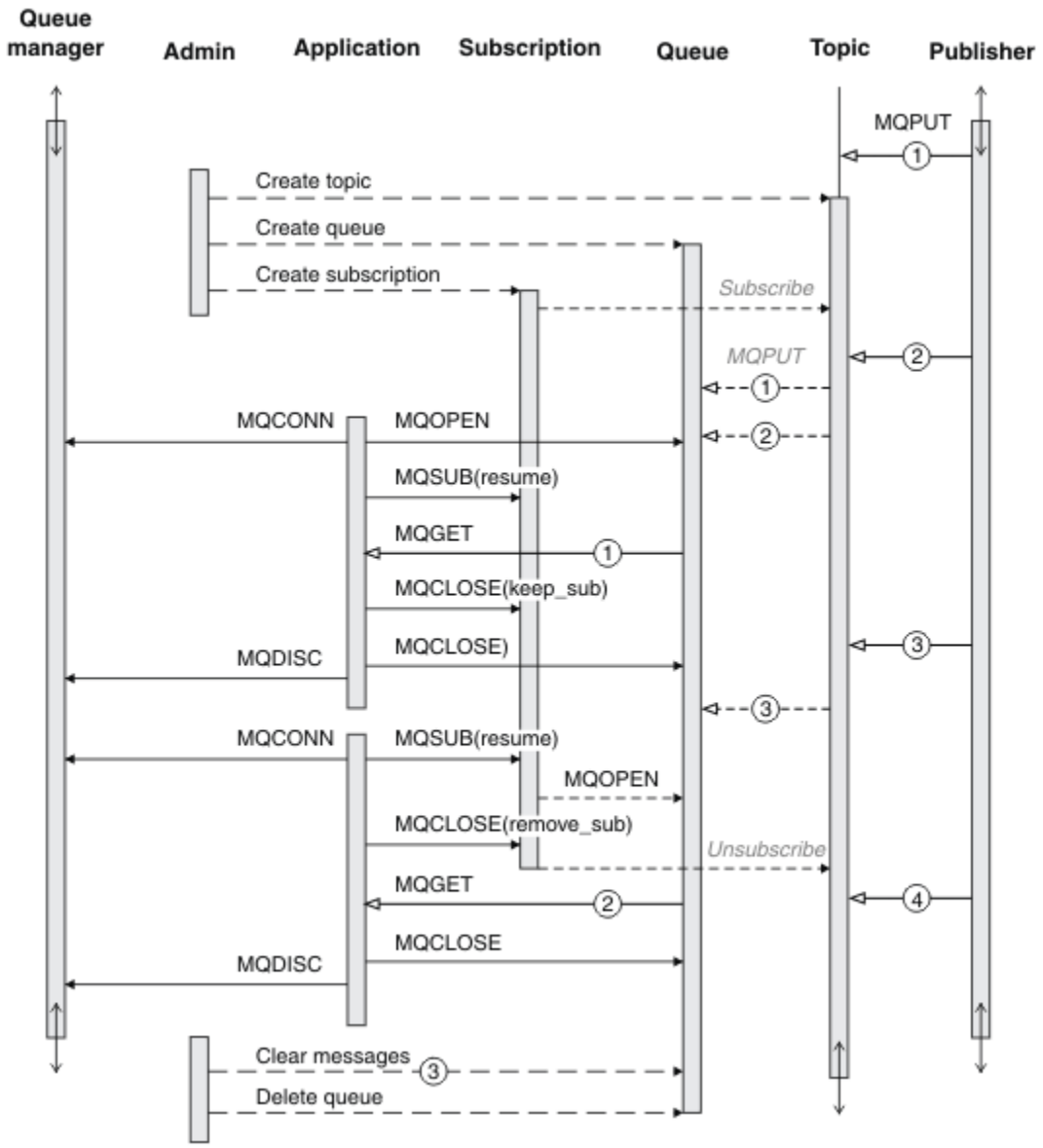


図 *85.* 非管理永続サブスクライバーのライフライン

非管理サブスクリプションの通常の パターンでは、キューとサブスクリプションのハウスキーピングは管理者に よって実行されます。 通常は、 アプリケーション・コードでプログラマチックに、管理サブスクライバーの動作をエミュレートしたり、キューとサブスクリプション を整理しようとはしません。 自身で管理ロジックを書く必要がある場合、 管理パターンを使用して同じ結果を得られるかどうかを 検討してください。 正確に同期する、完全に信頼できる管理コードをコーディングするのは、 容易なことではありません。 後で整理するほうが簡単です。 手動で行うか、または、メッセージ、サブスクリプション、およびキューを、状態に関わらず簡単に削除できることが確かであれば、 自動管理プログラムを使用できます。

パブリッシュ/サブスクライブのメッセージ・プロパティー

IBM MQ パブリッシュ/サブスクライブ・メッセージングにはいくつかのメッセージ・プロパティーが関係しています。

PubAccountingToken

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の AccountingToken フィールドに入る値です。 AccountingToken は、メッセージの識別コンテキ

ストの一部です。 メッセージ・コンテキストについて詳しくは、47 ページの『メッセージ・コンテキスト』を参照してください。 MQMD の AccountingToken フィールドについて詳しくは、AccountingToken を参照してください。

PubApplIdentityData

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の ApplIdentityData フィールドに入る値です。 ApplIdentityData は、メッセージの識別コンテキストの一部です。 メッセージ・コンテキストについて詳しくは、47 ページの『メッセージ・コンテキスト』を参照してください。 MQMD の ApplIdentityData フィールドについて詳しくは、ApplIdentityData を参照してください。

オプション MQSO_SET_IDENTITY_CONTEXT が指定されていない場合、デフォルト・コンテキスト情報としてこのサブスクリプションに対してパブリッシュされる各メッセージに設定される ApplIdentityData はブランクです。

オプション MQSO_SET_IDENTITY_CONTEXT が指定されている場合、PubApplIdentityData はユーザーによって生成され、このフィールドは、このサブスクリプションの各パブリケーションに設定される ApplIdentityData を含む入力フィールドになります。

PubPriority

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の Priority フィールドに入る値です。 MQMD の Priority フィールドについて詳しくは、Priority を参照してください。

値はゼロ以上でなければなりません。 ゼロは、最低優先順位です。 以下のような特殊値も使用できます。

- MQPRI_PRIORITY_AS_Q_DEF - MQSUB 呼び出しの Hobj フィールドでサブスクリプション・キューが提供されており、管理対象ハンドルではない場合、メッセージの優先順位はそのキューの DefProirity 属性から取られます。 そのようにして特定されたキューがクラスター・キューの場合、またはキュー名の解決パスに定義が複数ある場合は、MQMD の Priority の説明のとおり、優先順位は、パブリケーション・メッセージがキューに書き込まれるときに決定されます。 MQSUB 呼び出しが管理対象ハンドルを使用する場合、メッセージの優先順位は、サブスクライブ先のトピックに関連付けられたモデル・キューの DefPriority 属性から取られます。
- MQPRI_PRIORITY_AS_PUBLISHED - メッセージの優先順位は、元のパブリケーションの優先順位です。これはこのフィールドの初期値です。

SubCorrelId

重要 : 相関 ID は、階層内ではなく、パブリッシュ/サブスクライブ・クラスター内のキュー・マネージャー間でのみ受け渡し可能です。

このサブスクリプションに一致する送信されたパブリケーションにはすべて、メッセージ記述子内にこの相関 ID が含まれます。 複数のサブスクリプションが同じキューを使用してパブリケーションを取得する場合、相関 ID で MQGET を使用すると、特定のサブスクリプションに対するパブリケーションのみを取得できます。 この相関 ID はキュー・マネージャーまたはユーザーのいずれかによって生成されます。

オプション MQSO_SET_CORREL_ID が指定されていない場合、相関 ID はキュー・マネージャーによって生成され、このフィールドは、このサブスクリプションに対してパブリッシュされる各メッセージに設定される相関 ID を含む出力フィールドになります。

オプション MQSO_SET_CORREL_ID が指定されている場合、相関 ID はユーザーによって生成され、このフィールドは、このサブスクリプションの各パブリケーションに設定される相関 ID を含む入力フィールドになります。 この場合、フィールドの値が MQCI_NONE であれば、このサブスクリプションに対してパブリッシュされる各メッセージに設定される相関 ID はメッセージの元の書き込みによって作成された相関 ID です。

オプション MQSO_GROUP_SUB が指定されており、指定された相関 ID が、同じキューおよびオーバーラップ・トピック・ストリングを使用する既存のグループ化されたサブスクリプションと同じである場合、グループ内で最も有意なサブスクリプションのみがパブリケーションのコピーと共に提供されます。

SubUserData

これは、サブスクリプションのユーザー・データです。 このフィールドでサブスクリプションについて提供されるデータは、このサブスクリプションへ送信される各パブリケーションの MQSubUserData メッセージ・プロパティーとして含まれます。

パブリケーションのプロパティー

843 ページの表 124 は、パブリケーション・メッセージで提供されるパブリケーションのプロパティーのリストを示します。

これらのプロパティーは、**MQRFH2** フォルダーから直接アクセスするか、MQINQMP を使用して取り出すことができます。 MQINQMP は、照会するプロパティーの名前として、プロパティー名または **MQRFH2** 名を受け入れます。

表 *124.* パブリケーションのプロパティー

プロパティー名	MQRFH2 名	タイプ	説明
MQTopicString	mqps.Top	MQTYPE_STRING	トピック・ストリング
MQSubUserData	mqps.Sud	MQTYPE_STRING	サブスクライバー・ユーザー・データ
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	保存パブリケーション
MQPubOptions	mqps.Pub	MQTYPE_INT32	パブリケーション・オプション
MQPubLevel	mqps.Pbl	MQTYPE_INT32	パブリケーション・レベル
MQPubTime	mqpse.Pts	MQTYPE_STRING	パブリケーション時刻
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	パブリケーション・シーケンス番号
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	パブリッシャーによって追加されたストリング/整数型データ
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	次のメッセージ形式: MQRFH1 MQRFH2 PCF

メッセージの順序付け

特定のトピックで、メッセージは、パブリッシュ・アプリケーションから受信されるのと同じ順序で キュー・マネージャーによってパブリッシュされます (メッセージ優先順位に基づき再配列されることもあります)。

メッセージの順序付けは、通常、各サブスクライバーが、特定のパブリッシャーからの 特定のトピックについて、そのパブリッシャーがパブリッシュした順序で、 特定のキュー・マネージャーからメッセージを受信することを意味します。

ただし、IBM MQ のすべてのメッセージの場合と同様、メッセージの配信順序が乱れることはよくあります。 これは、次のような状況で 起こることがあります。

- ネットワーク内のリンクが切れて、以後のメッセージが別のリンクを通って 転送される場合
- キューが一時的にいっぱいになるか書き込み禁止になり、メッセージが 送達不能キューに入れられて配信が遅れたが、それ以後のメッセージは 問題なく配信された場合

- パブリッシャーとサブスクライバーがまだ作動しているときに、 管理者がキュー・マネージャーを削除したために、待機メッセージが送達不能キューに入れられ、サブスクリプションが中断された場合

以上のような状況にならない限り、 パブリケーションは必ず順序どおりに配信されます。

注 : パブリッシュ/サブスクライブでは、グループ化メッセージまたはセグメント化されたメッセージは使用できません。

パブリケーションの代行受信

パブリケーションをインターセプトして、変更した後、他のサブスクライバーに到達する前にリパブリッシュすることができます。

以下のいずれかのアクションを行うために、サブスクライバーに到達する前にパブリケーションをインターセプトすることもできます。

- メッセージに追加情報を付加する
- メッセージをブロックする
- メッセージを変換する

各メッセージに同じ操作を実行することもできますし、サブスクリプション、メッセージ、またはメッセージ・ヘッダーに応じて操作を変えることもできます。

関連資料

MQ_PUBLISH_EXIT - パブリッシュ出口

サブスクリプション・レベル

最終のサブスクライバーに到達する前にパブリケーションをインターセプトする、サブスクリプションのサブスクリプション・レベルを設定します。 インターセプト・サブスクライバーは、上位のサブスクリプション・レベルでサブスクライブし、下位のパブリケーション・レベルでリパブリッシュします。 最終のサブスクライバーに配信されるまでに、パブリケーションでメッセージ処理を実行するインターセプト・サブスクライバー・チェーンを作成します。

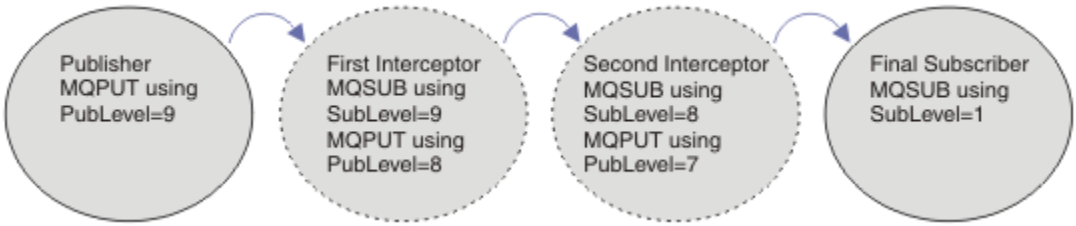


図 *86.* 一連のインターセプト・サブスクライバー

パブリケーションをインターセプトするには、**MQSD** SubLevel 属性を使用します。 インターセプトされたメッセージは、変換後、**MQPMO** PubLevel 属性を変更することによって下位のパブリケーション・レベルでリパブリッシュすることができます。 次に、メッセージは最終のサブスクライバーに送達されるか、または下位のサブスクリプション・レベルで中間サブスクライバーにより再度インターセプトされます。

インターセプト・サブスクライバーは通常、メッセージを変換してからリパブリッシュします。 一連のインターセプト・サブスクライバーがメッセージ・フローを形成します。 または、インターセプトされたパブリケーションをリパブリッシュしないこともできます。この場合、下位のサブスクリプション・レベルのサブスクライバーはメッセージを受け取りません。

インターセプターが他のどのサブスクライバーよりも先にパブリケーションを受け取るようにします。 インターセプターのサブスクリプション・レベルを、他のサブスクライバーよりも上位に設定します。 デフォルトで、サブスクライバーの SubLevel は 1 です。 最高値は 9 です。 パブリケーションは最低でも、一番高い SubLevel と同じくらいの高さの PubLevel で始まる必要があります。 最初はデフォルトの PubLevel である 9 でパブリッシュします。

- トピックでインターセプト・サブスクライバーを 1 つ設ける場合は、SubLevel を 9 に設定します。
- トピックに複数のインターセプト・アプリケーションを設ける場合は、次に続くインターセプト・サブスクライバーそれぞれについてより低い SubLevel を設定します。

- 最大で 8 つのインターセプト・アプリケーションを実装できます (サブスクリプション・レベルは 9 から下へ向かって 2 まで (両端を含む))。 メッセージの最終的な 受信側の SubLevel は 1 です。

最も高いサブスクリプション・レベルを持つインターセプターで、そのレベルがパブリケーションの PubLevel 以下のインターセプターが、最初にパブリケーションを受け取ります。 特定のサブスクリプション・レベルのトピックにつき、インターセプト・サブスクライバーを 1 つだけ構成します。 特定のサブスクリプション・レベルに複数のサブスクライバーを設定すると、パブリケーションの複数のコピーが最終のサブスクライブ・アプリケーションのセットに送信されます。

SubLevel が 0 に設定されたサブスクライバーはあらゆるパブリケーションの行き先となります。 つまり、メッセージを受け取る最終のサブスクライバーがない場合は、このサブスクライバーがパブリケーションを受け取ります。 SubLevel が 0 に設定されたサブスクライバーは、他のサブスクライバーが受け取らなかったパブリケーションをモニターするのに使用できます。

インターセプト・サブスクライバーのプログラミング

サブスクリプション・オプションは、845 ページの表 125 で説明するように使用します。

表 *125.* インターセプト・サブスクライバーのサブスクリプション・オプション

サブスクリプション・オプション	**注**
MQSO_SET_CORREL_ID と SubCorrelId を MQCI_NONE に設定する	インターセプトされたパブリケーションの CorrelId を、元のパブリケーションと同じままにします。 **注 :** 階層内のパブリケーションの相関 ID を受け渡すことはできません。 そのフィールドはキュー・マネージャーによって使用されます。
PubPriority を MQPRI_PRIORITY_AS_PUBLISHED に設定する	インターセプトされたパブリケーションの優先順位を、元のパブリケーションと同じままにします。

845 ページの表 125 に示すオプションは、すべてのインターセプト・サブスクライバーで使用される必要があります。 結果として、相関 ID とメッセージ優先順位が元のパブリッシャーの設定のままになります。

インターセプト・サブスクライバーがパブリケーションを処理する際に、自身のサブスクリプションの SubLevel よりも 1 つ低い PubLevel の同じトピックにメッセージをリパブリッシュします。 SubLevel が 9 に設定されたインターセプト・サブスクライバーの場合、8 の PubLevel でメッセージをリパブリッシュします。

メッセージを正しくリパブリッシュするには、元のパブリケーションからのいくつかの情報が必要となります。 元のメッセージで使用されているのと同じ **MQMD** を再使用し、MQPMO_PASS_ALL_CONTEXT を設定して、**MQMD** 内のすべての情報が次のサブスクライバーに渡されるようにします。 845 ページの表 126 に示されているメッセージ・プロパティーの値を、リパブリッシュされるメッセージの対応するフィールドにコピーします。 インターセプト・サブスクライバーはこれらの値を変更することができます。 OR 演算子を使用して、**MQPMO** に値を追加します。 書き込みメッセージ・オプションを結合するための オプション・フィールド。

管理対象パブリケーション・キューを使用するのではなく、明示的にパブリケーション・キューをオープンする必要があります。 管理対象キューに MQSO_SET_CORREL_ID を設定することはできません。 また、管理対象キューに MQOO_SAVE_ALL_CONTEXT を設定することもできません。 846 ページの『例』 に示されているコード・フラグメントを参照してください。

表 *126.* リパブリッシュされるメッセージの *MQPUT* 値

MQPUT によるメッセージのリパブリッシュ	**パブリケーション・メッセージの情報**
MQOD. ObjectString	メッセージ・プロパティー (message property) MQTopicString

表 126. リパブリッシュされるメッセージの *MQPUT* 値 (続き)	
MQPUT によるメッセージのリパブリッシュ	**パブリケーション・メッセージの情報**
MQPMO. Options	メッセージ・プロパティー (message property) MQPubOptions

最終のサブスクライバーでは、サブスクリプション・オプションを異なる設定にすることもできます。 例えば、パブリケーションの優先順位を、MQPRI_PRIORITY_AS_PUBLISHED にではなく、明示的に設定することができます。 最終のサブスクライバーの設定は、チェーン内の最終のインターセプト・サブスクライバーからのパブリケーションにのみ影響を与えます。

保存パブリケーション

保存パブリケーションは、インターセプトされた後、元のメッセージ書き込みオプションをリパブリッシュされるメッセージにコピーすることによって保持する必要があります。

MQPMO_RETAIN オプションはパブリッシャーによって設定されます。 845 ページの表 126 に示すように、各インターセプト・サブスクライバーは、リパブリッシュされるメッセージのメッセージ書き込みオプションに MQPubOptions を転送する必要があります。 メッセージ書き込みオプションをコピーすることによって、元のパブリッシャーによって設定されたオプション (パブリケーションを保持するかどうかを含め) が保持されます。

パブリケーションがインターセプト・サブスクライバー・チェーンを下方向に進んで最終のサブスクライバーに配信されると、最終的にこのパブリケーションは保持されます。 保存パブリケーションを要求する SubLevel 1 の新しいサブスクライバーが、これ以上のインターセプトなしでこのパブリケーションを受け取ります。 1 よりも高い SubLevel のサブスクライバーには、保存パブリケーションは送信されません。 結果として、保存パブリケーションがインターセプト・サブスクライバー・チェーンによって 2 度目に変更されることがありません。

例

ここでは、結合して 1 つのインターセプト・サブスクライバーを作成できるコード・フラグメントの例を示します。 ここでのコードは実動用の品質というよりも、簡潔であることを目的に記述されています。

846 ページの図 87 のプリプロセッサー・ディレクティブは、MQINQMP MQI 呼び出しが必要とするパブリケーション・メッセージから抽出される 2 つのプロパティーを定義します。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

図 *87.* プリプロセッサー・ディレクティブ

847 ページの図 88 は、コード・フラグメントで使用される宣言を示しています。 強調表示されている用語を除いて、宣言は IBM MQ アプリケーションに標準のものです。

強調表示されている書き込みオプションと取得オプションはすべてのコンテキストを渡すよう初期化されています。 強調表示されている MQTOPICSTRING と MQPUBOPTIONS は、プリプロセッサー・ディレクティブで定義されているプロパティー名の MQCHARV 初期化指定子です。 名前は MQINQMP に渡されます。

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

図 *88.* 宣言

宣言で容易に実行できない初期化を 848 ページの図 89 に示します。 強調表示されている値には説明が必要です。

SYSTEM.NDURABLE.MODEL.QUEUE
この例では、MQSUB を使用して管理非永続サブスクリプションをオープンするのではなく、モデル・キュー SYSTEM.NDURABLE.MODEL.QUEUE を使用して一時動的キューを作成します。 そのハンドルが MQSUB に渡されます。 キューを直接オープンすることによって、すべてのメッセージ・コンテキストを保存し、サブスクリプション・オプション MQSO_SET_CORREL_ID を設定することができます。

MQGMO_CURRENT_VERSION
ほとんどの IBM MQ 構造体について、現行バージョンを使用することが重要です。 gmo.MsgHandle などのフィールドは、最新バージョンの制御構造体でのみ使用可能です。

MQGMO_PROPERTIES_IN_HANDLE
元のパブリケーションで設定されているトピック・ストリングおよびメッセージ書き込みオプションは、インターセプト・サブスクライバーによってメッセージ・プロパティーを使用して取得されます。 または、メッセージで直接 **MQRFH2** 構造体が読み取られます。

MQSO_SET_CORREL_ID
MQSO_SET_CORREL_ID を次と組み合わせて使用します。

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

これらのオプションにより、相関 ID が渡されるようになります。 元のパブリッシャーによって設定されている相関 ID は、インターセプト・サブスクライバーが受け取ったパブリケーションの相関 ID フィールドに配置されます。 各インターセプト・サブスクライバーは、同じ相関 ID を渡します。 これにより、最終のサブスクライバーでは、同じ相関 ID を受け取るというオプションを持つことになります。

注 : パブリケーションがパブリッシュ/サブスクライブ階層を介して渡される場合、相関 ID は保持されることがありません。

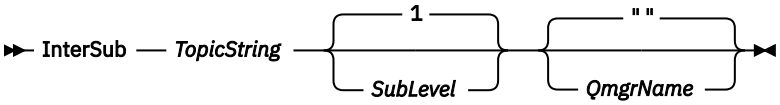
MQPRI_PRIORITY_AS_PUBLISHED
パブリケーションは、パブリッシュ時に使用されたのと同じメッセージ優先順位でパブリケーション・キューに配置されます。

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

図 *89.* 初期化

849 ページの図 90 に、コマンド行パラメーターを読み取り、初期化を完了し、インターセプト・サブスクリプションを作成するコード・フラグメントを示します。

次のコマンドを使用してプログラムを実行します。



エラー処理を可能な限り目立たなくするために、各 MQI 呼び出しからの理由コードが別の配列エレメントに格納されます。 完了コードがテストされた各呼び出し後に、値が MQCC_FAIL の場合は、制御が `do { } while(0)` コード・ブロックを出ます。

2 つの重要なコード行は次のとおりです。

pmo.PubLevel = sd.SubLevel - 1;
リパブリッシュされるメッセージのパブリケーション・レベルを、インターセプト・サブスクライバーのサブスクリプション・レベルより 1 レベル低く設定します。

gmo.MsgHandle = Hmsg;
MQGET がメッセージ・プロパティーを返せるように、メッセージ・ハンドルを指定します。

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

図 *90.* パブリケーションのインターセプトの準備

主なコード・フラグメント (850 ページの図 91) では、パブリケーション・キューからメッセージを取得します。 メッセージ・プロパティーを照会し、トピック・ストリングと元の **MQPMO** を使用してメッセージをリパブリッシュします。 パブリケーションの **オプション** ・プロパティー。

この例では、パブリケーションでは変換が実行されていません。 リパブリッシュされるパブリケーションのトピック・ストリングは、インターセプト・サブスクライバーがサブスクライブしたトピック・ストリングと必ず一致します。 インターセプト・サブスクライバーが、同じパブリケーション・キューに送信された複数のサブスクリプションのインターセプトを行う場合、異なるサブスクリプションと一致するパブリケーションを区別するために、トピック・ストリングを照会しなければならない場合があります。

MQINQMP の呼び出しが強調表示されています。 トピック・ストリングとパブリケーションのメッセージ書き込みオプションのプロパティーは、出力制御構造体に直接書き込まれます。 putOD.ObjectString の MQCHARV 長さフィールドを明示的な長さからヌル終了文字に変更する唯一の理由は、printf を使用してストリングを出力するためです。

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

図 *91.* パブリケーションのインターセプトおよびリパブリッシュ

850 ページの図 92 に最後のコードを示します。

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

図 *92.* 完了

パブリケーションおよび分散パブリッシュ/サブスクライブのインターセプト

分散パブリッシュ/サブスクライブ・トポロジーにインターセプト・サブスクライバーまたはパブリッシュ出口をデプロイする場合は、単純なパターンに従います。 インターセプト・サブスクライバーはパブリッシャーと同じキュー・マネージャーにデプロイし、パブリッシュ出口は最終のサブスクライバーと同じキュー・マネージャーにデプロイします。

851 ページの図 93 は、パブリッシュ/サブスクライブ・クラスター内で接続されている 2 つのキュー・マネージャーを示したものです。 パブリッシャーは、パブリケーション・レベル 9 でクラスター・トピックにパブリケーションを作成します。 番号付き矢印は、クラスター・トピックへのサブスクライバーへフローするときに、パブリケーションによって取られた一連のステップを示します。 パブリケーションは、Sublevel 9 を持つサブスクライバーによってインターセプトされ、Publevel 8 で再公開されます。 これは、Sublevel 8 のサブスクライバーによって再度インターセプトされます。 このサブスクライバーは、Publevel 7 でリパブリッシュします。 キュー・マネージャーによって提供されるプロキシー・サブスクライバーは、このパブリケーションをキュー・マネージャー B に転送します。キュー・マネージャー B には、最終のサブスクライバーだけではなくパブリッシュ出口もデプロイされています。 このパブリケーションは、パブリッシュ出口によって処理されてから、最終的に Sublevel 1 の最終のサブスクライバーに

よって受信されます。 インターセプト・サブスクライバーおよびパブリッシュ出口は、破線で囲んで表示しています。

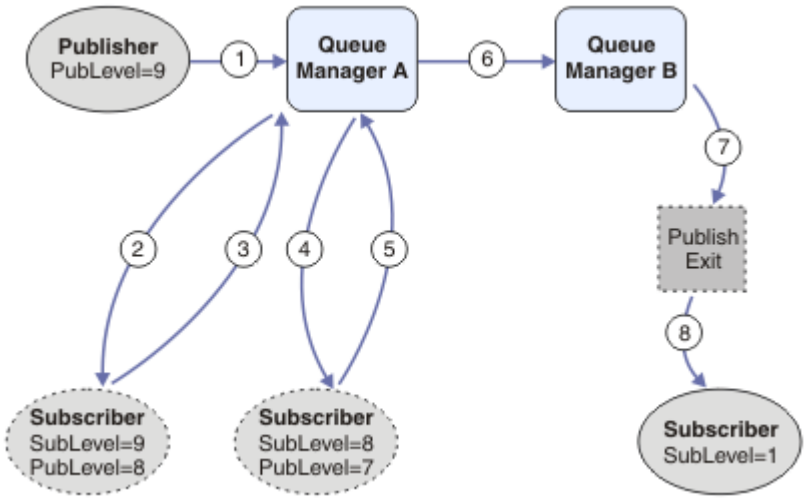


図 93. クラスター内のインターセプトおよびパブリッシュ出口

このような単純なパターンの目的は、パブリケーションを受信するすべてのサブスクライバーが、同一のパブリケーションを受信するようにすることです。 パブリケーションは、サブスクライバーがどこに接続されているかには関係なく、同じ変換シーケンスを経ます。 パブリッシャーまたは最終のサブスクライバーが接続されている場所によって変換シーケンスが変わるようなことは回避することが必要です。 ただし、各サブスクライバーに最終的に配信されるパブリケーションを調整することは妥当な例外といえます。 パブリケーションの最終配信先であるキューに基づいてパブリケーションをカスタマイズするには、パブリッシュ出口を使用します。

分散パブリッシュ/サブスクライブ・トポロジー内のどこにインターセプト・サブスクライバーおよびパブリッシュ出口をデプロイするかについては、慎重に検討する必要があります。 単純なパターンは、インターセプト・サブスクライバーをパブリッシャーと同じキュー・マネージャーにデプロイし、パブリッシュ出口を最終のサブスクライバーと同じキュー・マネージャーにデプロイすることです。

アンチパターン

852 ページの図 94 は、単純なパターンに従わなかった場合に予期しない状況になる可能性を示したものです。 最終のサブスクライバーがキュー・マネージャー A に追加され、さらに 2 つのインターセプト・サブスクライバーがキュー・マネージャー B に追加されて、デプロイメントが複雑化しています。

パブリケーションは PubLevel 7 でキュー・マネージャー B に転送され、そこで SubLevel 5 のサブスクライバーによってインターセプトされてから、SubLevel 1 の最終のサブスクライバーによってコンシュームされます。 パブリッシュ出口は、キュー・マネージャー B のインターセプト・コンシューマーと最終コンシューマーの両方にパブリケーションが渡される前にインターセプトします。 パブリケーションは、パブリッシュ出口によって処理されずに、キュー・マネージャー A の最終サブスクライバーに到達します。

パブリッシュ/サブスクライブ・トポロジーでは、プロキシー・サブスクライバーは SubLevel 1 でサブスクライブし、最後のインターセプト・サブスクライバーによって設定された PubLevel を渡します。 852 ページの図 94 では、その結果、パブリケーションは、キュー・マネージャー B にある SubLevel 9 を使用しているサブスクライバーにはインターセプトされません。

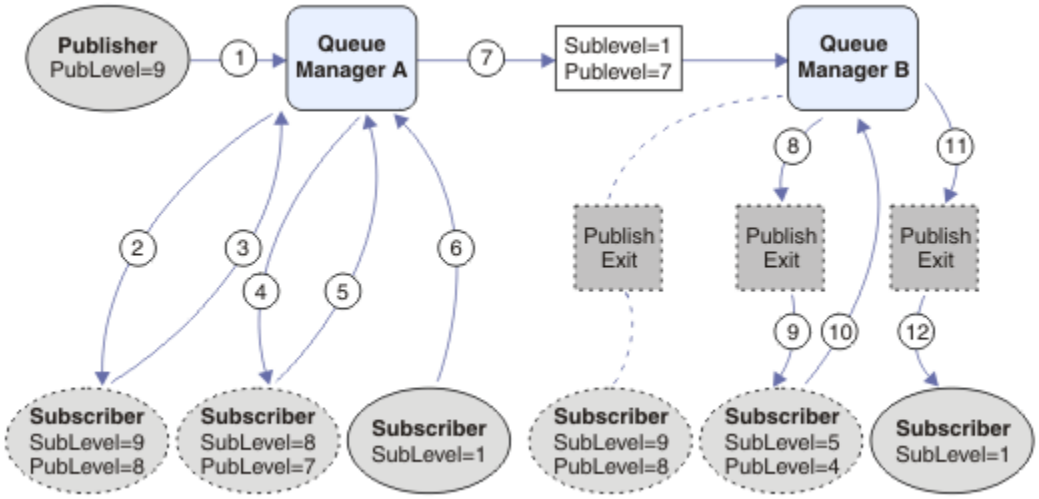


図 *94.* インターセプト・サブスクライバーの複雑なデプロイメント

パブリッシュのオプション

メッセージのパブリッシュ方法を制御するために使用できるオプションが幾つかあります。

サブスクライバーの応答情報の保留

サブスクライバーが受け取ったパブリケーションに対して応答できるようにしたくない場合、MQPMO_SUPPRESS_REPLYTO 書き込みメッセージ・オプションを使用して、MQMD の ReplyToQ フィールドと ReplyToQmgr フィールドの情報を保留できます。 このオプションを使用すると、キュー・マネージャーがパブリケーションを受け取るときに、いずれかのサブスクライバーに転送する前に MQMD の情報を削除します。

このオプションは、ReplyToQ を必要とするレポート・オプションと組み合わせて使用することはできません。組み合わせて使用すると、呼び出しは MQRC_MISSING_REPLY_TO_Q で失敗します。

パブリケーション・レベル

パブリケーション・レベルを使用すると、パブリケーションを受け取るサブスクライバーを制御できます。パブリケーション・レベルは、パブリケーションの宛先となるサブスクリプションのレベルを指示します。パブリケーションのパブリケーション・レベル以下のサブスクリプション・レベルのサブスクリプションで、そのうち最も高いサブスクリプション・レベルを持つサブスクリプションのみが、パブリケーションを受け取ります。 この値は、0 から 9 までの範囲でなければならず、0 が最低のパブリケーション・レベルです。 このフィールドの初期値は 9 です。 パブリッシュ・レベルおよびサブスクリプション・レベルの使用法の 1 つは、インターセプト・パブリケーションです。

パブリケーションがいずれかのサブスクライバーに送信されていないかどうかの確認

パブリケーションがいずれかのサブスクライバーに送信されていないかどうかを確認するには、MQPUT 呼び出しで MQPMO_WARN_IF_NO_SUBS_MATCHED 書き込みメッセージ・オプションを使用します。 PUT 操作によって完了コード MQCC_WARNING および理由コード MQRC_NO_SUBS_MATCHED が戻された場合、パブリケーションはどのサブスクリプションにも送信されませんでした。 PUT 操作で MQPMO_RETAIN オプションが指定されている場合、メッセージは保存され、それ以降に定義される、一致するサブスクリプションに送信されます。 分散パブリッシュ/サブスクライブ・システムでは、MQRC_NO_SUBS_MATCHED 理由コードが戻されるのは、キュー・マネージャーのトピックにプロキシー・サブスクリプションが登録されていない場合のみです。

サブスクリプション・オプション

メッセージのサブスクリプションの処理方法を制御するオプションがいくつかあります。

メッセージの持続性

キュー・マネージャーは、サブスクライバーに転送するパブリケーションの持続性を、パブリッシャーが設定したとおりに維持します。 パブリッシャーは、持続性を以下のいずれかのオプションに設定します。

0
非持続

1
永続

2
キュー/トピックの定義と同じ持続性

パブリッシュ/サブスクライブの場合、パブリッシャーがトピック・オブジェクトと **topicString** を解決済みトピック・オブジェクトにします。 パブリッシャーがキュー/トピック定義のとおりの持続性を指定している場合は、解決済みのトピック・オブジェクトのデフォルトの持続性がパブリケーションに設定されます。

保存パブリケーション

保存パブリケーションをいつ受け取るかを制御するために、サブスクライバーは次の 2 つのサブスクリプション・オプションを使用できます。

要求があったときのみパブリッシュ、MQSO_PUBLICATIONS_ON_REQUEST

いつパブリケーションを受け取るかをサブスクライバーに制御させるには、MQSO_PUBLICATIONS_ON_REQUEST サブスクリプション・オプションを使用できます。 次いでサブスクライバーは、MQSUBRQ 呼び出し (元の MQSUB 呼び出しから返される Hsub ハンドルを指定) を使用してトピックの保存パブリケーションの送信を要求することにより、いつパブリケーションを受け取るかを制御します。 MQSO_PUBLICATIONS_ON_REQUEST サブスクリプション・オプションを使用するサブスクライバーは、非保存パブリケーションを受け取りません。

MQSO_PUBLICATIONS_ON_REQUEST を指定する場合、 MQSUBRQ を使用してパブリケーションを取得する必要があります。 MQSO_PUBLICATIONS_ON_REQUEST を使用しない場合は、 パブリッシュされるときにメッセージを取得します。

サブスクライバーが MQSUBRQ 呼び出しを使用し、サブスクリプションのトピックにワイルドカードを使用すると、サブスクリプションはトピック・ツリー上の複数のトピックまたはノードと一致する可能性があり、それらのすべてと保存メッセージ (存在する場合) がサブスクライバーに送信されることになります。

特に、このオプションを永続サブスクリプションと併用すると便利な場合があります。というのは、永続的にサブスクライブする場合、サブスクライバー・アプリケーションが実行していなくても、キュー・マネージャーは継続的にパブリケーションをサブスクライバーに送信するからです。 このため、サブスクライバー・キューにメッセージが蓄積される可能性があります。 サブスクライバーの登録時に MQSO_PUBLICATIONS_ON_REQUEST オプションを使用すると、この蓄積を回避できます。 あるいは、アプリケーションにとって適切であるなら、非永続サブスクリプションを使用することによって、不要なメッセージの蓄積を回避することもできます。

サブスクリプションが永続的で、パブリッシャーが保存パブリケーションを使用する場合は、サブスクライバー・アプリケーションは MQSUBRQ 呼び出しを使用して、再始動後に状態情報をリフレッシュすることができます。 以後、サブスクライバーは MQSUBRQ 呼び出しを使用して、定期的にその状態をリフレッシュする必要があります。

このオプションを使用すると、MQSUB 呼び出しの結果としてパブリケーションが送信されることはありません。 元の永続サブスクリプションが MQSO_PUBLICATIONS_ON_REQUEST オプションを使用するように構成されている場合は、このサブスクリプションが切断してから再開されるときに、このオプションが使用されることになります。

新規パブリケーションのみ、MQSO_NEW_PUBLICATIONS_ONLY

トピックに保存パブリケーションが存在する場合、パブリケーションが行われた後でサブスクライバーがサブスクリプションを行うと、これはそのパブリケーションのコピーを受け取ることになります。

サブスクリプションよりも前に行われたパブリケーションを受け取ることを希望しないサブスクライバーは、MQSO_NEW_PUBLICATIONS_ONLY サブスクリプション・オプションを使用できます。

サブスクリプションのグループ化

パブリケーションを受け取るようにキューをセットアップしていて、 多くのオーバーラップ・サブスクリプションが同じキューにパブリケーションを送る場合に、 サブスクリプションのグループ化を検討してください。 この状態は、 オーバーラップ・サブスクリプションの例と類似しています。

トピックにサブスクライブするときにオプション MQSO_GROUP_SUB を設定することによって、 重複パブリケーションを受け取らないようにできます。 結果として、グループ内で複数のサブスクリプションが パブリケーションのトピックと一致するときは、1 つのサブスクリプションだけが キューへのパブリケーションの配置を受け持ちます。 パブリケーションのトピックと一致したその他のサブスクリプションは無視されます。

キューへのパブリケーションの配置を受け持つサブスクリプションは、ワイルドカードが出現するまでに、最も長く一致するトピック・ストリングを持つものを基にして選択されます。 このサブスクリプションは、最も一致度が高いサブスクリプションと考えることができます。 そのプロパティーは、MQSO_NOT_OWN_PUBS プロパティーがあるかどうかも含めて、パブリケーションに伝搬されます。 このプロパティーがある場合、その他の一致するサブスクリプションで MQSO_NOT_OWN_PUBS プロパティーがないとしても、 パブリケーションはキューに送達されません。

重複パブリケーションを回避するため、すべてのサブスクリプションを単一グループに置くことはできません。 グループ化したサブスクリプションは、 以下の条件を満たす必要があります。

1. いずれのサブスクリプションも管理対象ではない。
2. サブスクリプションのグループが同じキューにパブリケーションを送達する。
3. 各サブスクリプションが同じサブスクリプション・レベルにある必要がある。
4. グループ内の各サブスクリプションのパブリケーション・メッセージの 相関 ID が同じである。

   各サブスクリプションを同じ相関 ID のパブリケーション・メッセージにするには、 パブリケーション内に独自の相関 ID を作成するように MQSO_SET_CORREL_ID を設定し、 各サブスクリプションの **SubCorrelId** フィールドに同じ値を設定します。 **SubCorrelId** を値 MQCI_NONE に設定しないでください。

詳しくは、../refdev/q100080_.dita#q100080_/mqso_group_sub を参照してください。

オブジェクト属性の照会と設定

属性は、IBM MQ オブジェクトの性質を定義する特性です。

キュー・マネージャーがオブジェクトを処理する方法は属性の影響を受けます。 各タイプの IBM MQ オブジェクトの属性については、オブジェクトの属性で詳しく説明しています。

属性の一部には、オブジェクトが定義されたときに設定されて、IBM MQ コマンドを使用しないと変更できないものもあります。そのような属性の例としては、キューに書き込まれるメッセージのデフォルト優先順位があります。 その他の属性は、キュー・マネージャーの操作によって影響を受け、時間の経過で変わることがあります。その例として、キューの現在の長さがあります。

ほとんどの属性の現行値を MQINQ 呼び出しによって照会できます。 また、MQI は、キュー属性のいくつかを変更できる MQSET 呼び出しを提供します。 他のタイプのオブジェクトの属性を変更するときに、この MQI 呼び出しを使用することはできません。 代わりに、以下のいずれかのリソースを使用する必要があります。

- ALW MQSC 機能。MQSC コマンドを参照してください。
- IBM i CHGMQMx CL コマンド。これについては、 IBM i の CL コマンド・リファレンス、または MQSC 機能で説明されています。
- z/OS ALTER オペレーター・コマンド、または REPLACE オプションを指定した DEFINE コマンド (MQSC コマンドを参照)。

注 : この資料では、オブジェクト属性の名前を、MQINQ 呼び出しおよび MQSET 呼び出しで使用される形式で示しています。 IBM MQ コマンドを使用して属性を定義、変更、または表示する場合、トピック・リンクのコマンドの説明にあるキーワードを使用して属性を識別する必要があります。

MQINQ と MQSET 呼び出しの両方で、セレクターの配列を使用して識別します。 それらの属性は、照会または設定する属性です。 使用できる各属性ごとにセレクターがあります。 セレクター名には、属性の性質によって決められる次の接頭部があります。

表 *127.* セレクター名の接頭部

接頭部	**説明**
MQCA_	これらのセレクターは、文字データ (例えば、キューの名前) を含む属性を参照します。
MQIA_	これらのセレクターは、数値 (例えば、キュー上のメッセージ数を示す *CurrentQueueDepth*) または定数値 (キュー・マネージャーが同期点をサポートするかどうかを示す *SyncPoint*) を含む属性を参照します。

MQINQ または MQSET 呼び出しを使用するには、アプリケーションはキュー・マネージャーに接続されており、MQOPEN 呼び出しを使用して属性の設定または照会のためにオブジェクトをオープンしなければなりません。 これらの操作については、735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』および 742 ページの『オブジェクトのオープンとクローズ』に記載されています。

オブジェクト属性の照会および設定について詳しくは、以下のリンクを参照してください。

- 856 ページの『オブジェクト属性の照会』
- 856 ページの『MQINQ 呼び出しが失敗する場合』
- 857 ページの『キュー属性の設定』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

オブジェクト属性の照会

MQINQ 呼び出しを使用して、任意のタイプの IBM MQ の属性を照会します。

この呼び出しへの入力として、次のものを指定しなければなりません。

- 接続ハンドル。
- オブジェクト・ハンドル。
- セレクターの数。
- 属性セレクターの配列。各セレクターは、MQCA_* または MQIA_* の形式を使用します。各セレクターは、照会する値を持つ属性を表し、各セレクターは、オブジェクト・ハンドルが表すオブジェクトのタイプに対して有効である必要があります。 セレクターは、任意の順番で指定することができます。
- 照会する整数属性の値。 整数属性を照会しない場合は、ゼロを指定します。
- *CharAttrLength* の文字属性バッファーの長さ。 これは、少なくとも、各文字属性ストリングを保持するのに必要な長さの合計でなければなりません。 文字属性を照会しない場合は、ゼロを指定します。

MQINQ の出力は以下のとおりです。

- 配列にコピーされた一連の整数属性値。 値の数は *IntAttrCount* によって決まります。 *IntAttrCount* または *SelectorCount* のどちらかがゼロの場合、このパラメーターは使用されません。
- 文字属性が戻されるバッファー。 バッファーの長さは、**CharAttrLength** パラメーターによって指定されます。 *CharAttrLength* または *SelectorCount* のどちらかがゼロの場合、このパラメーターは使用されません。
- 完了コード。 完了コードが警告を示している場合、呼び出しの一部のみが完了したことを意味します。 この場合は、理由コードを調べてください。
- 理由コード。 部分完了の状況として、以下の 3 つの場合があります。
  - セレクターがキュー・タイプに適合しない。
  - 整数属性用に十分なスペースがない。
  - 文字属性用に十分なスペースがない。

  これらのうち複数の状況が起こった場合、最初に適合するものが戻されます。

また、出力や照会時にキューをオープンしても、そのキュー名が非ローカル・クラスター・キューの別名の場合、ユーザーが照会できるのはキュー名、キュー・タイプおよび共通属性だけです。
MQOO_BIND_ON_OPEN が使われている場合は、このコマンドで選択したキューの属性の値が共通属性の値になります。 MQOO_BIND_NOT_FIXED または MQOO_BIND_ON_GROUP が使用されている場合、あるいは MQOO_BIND_AS_Q_DEF が使用され、**DefBind** キュー属性が MQBND_BIND_NOT_FIXED である場合は、任意のクラスター・キューの属性値が共通属性の値になります。 詳細については、889 ページの『MQOPEN およびクラスター』および MQOPEN を参照してください。

注 : 呼び出しによって戻された値は、選択された属性のスナップショットです。 これらの属性は、プログラムが戻された値に基づいて処理を行う前に、変更できます。

MQINQ 呼び出しの説明については、MQINQ を参照してください。

MQINQ 呼び出しが失敗する場合

別名の属性について照会するために別名を開くと、基本キューの属性ではなく、別名キュー (別のキューにアクセスするために使用される IBM MQ オブジェクト) の属性が戻されます。

しかし、別名が識別する、基本キューの定義もキュー・マネージャーによってオープンされます。もし別のプログラムが基本キューの使用を、MQOPEN と MQINQ 呼び出しの間に変更するのであれば、MQINQ 呼び出しはエラーになり、MQRC_OBJECT_CHANGED 理由コードを戻します。 呼び出しは、別名キュー・オブジェクトの属性が変更されたときもエラーになります。

同様に、リモート・キューをオープンして属性を照会する場合、リモート・キューだけのローカル定義の属性が戻されます。

照会するキュー属性のタイプに対して無効なセレクターを 1 つ以上指定した場合、MQINQ 呼び出しが警告で完了し、出力は次のように設定されます。

- 整数属性については、*IntAttrs* の対応するエレメントが MQIAV_NOT_APPLICABLE に設定される。
- 文字属性については、*CharAttrs* ストリングの対応する部分がアスタリスクに設定される。

照会するオブジェクト属性のタイプに対して無効なセレクターを 1 つ以上指定した場合、MQINQ 呼び出しは失敗し、MQRC_SELECTOR_ERROR 理由コードが戻ります。

MQINQ を呼び出してモデル・キューを照会することはできません。MQSC 機能を使用するか、ご使用のプラットフォームで使用可能なコマンドを使用してください。

キュー属性の設定

この情報は、MQSET 呼び出しを使用してキュー属性を設定する方法について学習するのに使用します。

MQSET 呼び出しを使用して設定できるのは、次のキュー属性だけです。

- *InhibitGet* (リモート・キューのものを除く)
- *DistList* (z/OS の場合を除く)
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

MQSET 呼び出しは、MQINQ 呼び出しと同じパラメーターを持っています。 ただし、MQSET の場合は、完了コードと理由コードを除くすべてのパラメーターが入力パラメーターです。 部分完了という状況はありません。

注 : MQI では、ローカルに定義されたキュー以外の IBM MQ オブジェクトの属性は設定できません。

MQSET 呼び出しの詳細については、MQSET を参照してください。

作業単位のコミットとバックアウト

ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

このトピックでは以下の用語が使用されます。

- コミット
- バックアウト
- 同期点調整
- 同期点
- 作業単位
- 単一フェーズ・コミット
- 2 フェーズ・コミット

これらのトランザクション処理の用語を既に理解している場合は、859 ページの『IBM MQ アプリケーションでの同期点に関する考慮事項』に進んでください。

コミットとバックアウト

プログラムが作業単位内でメッセージをキューに書き込むとき、そのメッセージは、プログラムがその作業単位をコミットするまで他のプログラムには見えません。 作業単位をコミットするには、データの安全性を保護するためにすべての更新処理が正常終了する必要があります。 プログラムがエラーを

検出して、PUT 操作を永続的にしないと判断した場合、その作業単位をバックアウトできます。 プログラムがバックアウトを行うと、IBM MQ はその作業単位によってキューに書き込まれたメッセージを除去することにより、キューを復元します。 プログラムがコミットやバックアウト操作を実行する方法は、プログラムが実行されている環境に依存します。

同様に、プログラムが作業単位内でキューからメッセージを読み取ったときも、そのメッセージは、プログラムがその作業単位をコミットするまでキューに残っています。ただし、そのメッセージを他のプログラムが取り出すことはできません。 そのメッセージは、プログラムが作業単位をコミットしたときに、キューから永続的に削除されます。 プログラムが作業単位をバックアウトすると、IBM MQ は、メッセージが他のプログラムによって検索できるようにすることによって、キューを復元します。

同期点調整、同期点、作業単位

同期点調整 とは、データの保全性を失わずに作業単位をコミットまたはバックアウトする処理のことです。

変更をコミットするかバックアウトするかについての決定は、最も単純な場合、トランザクションの終了時に行われます。 ただし、トランザクション内の別の論理点でデータ変更を同期化すると、アプリケーションに対してさらに有益になる場合もあります。 これらの論理ポイントを *syncpoints* (または同期点) と呼びます。また、2 つの同期点間の更新セットを処理する期間を作業単位と呼びます。 1 つの作業単位に、複数の MQGET 呼び出しや MQPUT 呼び出しを含めることができます。

1 つの作業単位内のメッセージの最大数は、ALTER QMGR コマンドの MAXUMSGS 属性によって制御することができます。

単一フェーズ・コミット

単一フェーズ・コミット 処理とは、キューに対する変更を他のリソース管理プログラムとの間で調整せずに、プログラムがキューに対する更新をコミットできる処理です。

2 フェーズ・コミット

2 フェーズ・コミット処理とは、プログラムが IBM MQ のキューに対して行った更新を他のリソース (例えば、Db2 の制御下にあるデータベース) の更新と整合できる処理です。 このような処理のもとでは、すべてのリソースに対する更新が共にコミットまたはバックアウトされます。

作業単位の処理を支援するため、IBM MQ には **BackoutCount** という属性があります。 この属性のカウントは、作業単位内でメッセージがバックアウトされるたびに増えます。 同じメッセージで作業単位が何度も異常終了すると、*BackoutCount* の値は *BackoutThreshold* の値を超えてしまいます。この値は、キューの定義時に設定します。 このような状況が生じた場合、アプリケーションは、作業単位からメッセージを取り除いて、*BackoutRequeueQName* に定義された別のキューに書き込むことができます。 メッセージを移動すると、作業単位をコミットできます。

作業単位のコミットとバックアウトについて詳しくは、以下のリンクを参照してください。

- 859 ページの『IBM MQ アプリケーションでの同期点に関する考慮事項』
- z/OS 860 ページの『IBM MQ for z/OS アプリケーションにおける同期点』
- IBM i 862 ページの『CICS for IBM i アプリケーションの同期点』
- 863 ページの『IBM MQ for Multiplatforms の同期点』
- IBM i 867 ページの『IBM i 外部同期点管理プログラムへのインターフェース』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』

この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

IBM MQ アプリケーションでの同期点に関する考慮事項

この情報を使用して、IBM MQ アプリケーションでの同期点の使用について学びます。

2 フェーズ・コミットは次の環境でサポートされています。

- Multi IBM MQ for Multiplatforms
- z/OS z/OS 用の CICS Transaction Server
- z/OS TXSeries
- z/OS IMS/ESA®
- z/OS RRS を実行する z/OS バッチ
- X/Open XA インターフェースを使用するその他の外部コーディネーター

単一フェーズ・コミットは、以下の環境でサポートされています。

- Multi IBM MQ for Multiplatforms
- z/OS z/OS バッチ

外部インターフェースについて詳しくは、866 ページの『Multiplatforms の外部同期点管理プログラムへのインターフェース』、および The Open Group 発行の XA 資料「*CAE Specification Distributed Transaction Processing: The XA Specification*」を参照してください。 トランザクション管理プログラム (CICS、IMS、Encina、Tuxedo など) は、他のリカバリー可能リソースと整合する 2 フェーズ・コミットに参加できます。 これは、IBM MQ で提供されるキューイング機能が作業単位のスコープ内になり、トランザクション管理プログラムによる管理が可能になることを意味します。

IBM MQ に同梱されているサンプルは、IBM MQ 調整の XA 準拠データベースを示しています。 このサンプルの詳細については、1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』を参照してください。

IBM MQ アプリケーションでは、書き込みおよび読み取り呼び出しごとに、呼び出しを同期点の制御下に置くかどうかを指定できます。 書き込み操作が同期点の制御のもとに行われるようにするには、MQPUT の呼び出し時に MQPMO 構造体の *Options* フィールドに MQPMO_SYNCPOINT を指定してください。 読み取り操作の場合は、MQGMO 構造体の *Options* フィールドに MQGMO_SYNCPOINT 値を指定します。 明示的にオプションを選択しない場合は、デフォルトの処置はプラットフォームによって異なります。

- Multi デフォルトの同期点制御は NO です。
- z/OS デフォルトの同期点制御は YES です。

MQPUT1 呼び出しが MQPMO_SYNCPOINT を指定して発行されると、 書き込み操作を非同期に完了するように、デフォルトの動作が変更されます。 これによって、 戻される MQOD および MQMD の構造体内の特定のフィールドに未定義の値が含まれるようになるため、それらのフィールドに依存する一部のアプリケーションの動作が変更される場合があります。 アプリケーションでは、MQPMO_SYNC_RESPONSE を指定することで、書き込み操作を同期させて実行すること、および該当するすべてのフィールドの値を完成させることができます。

アプリケーションが同期点での MQPUT または MQGET への応答で MQRC_BACKED_OUT 理由コードを受け取った場合、アプリケーションは通常、MQBACK を使用して現行トランザクションをバックアウトし、その後で、適切であればトランザクション全体を再試行する必要があります。 アプリケーションが MQCMIT または MQDISC 呼び出しの応答で MQRC_BACKED_OUT を受け取った場合は、MQBACK を呼び出す必要はありません。

MQGET 呼び出しがバックアウトされるたびに、影響を受けるメッセージの MQMD 構造体の *BackoutCount* フィールドで、カウントが増えます。 *BackoutCount* のカウントが高いことは、メッセージがたびたびバックアウトされていることを示します。 これは、このメッセージに問題があることを示している場合があるため、調査が必要です。 *BackoutCount* について詳しくは、 BackoutCount を参照してください。

RRS を実行する z/OS バッチの場合を除いて、コミットされていない要求があるのに、プログラムが MQDISC 呼び出しを発行した場合は、暗黙の同期点が生じます。 プログラムが異常終了した場合は、暗黙のバックアウトが発生します。

z/OS z/OS では、プログラムが最初に MQDISC を呼び出さないと、正常終了しても暗黙の同期点が生じます。 MQ に接続された TCB が正常終了した場合、プログラムは正常終了したと見なされます。 z/OS UNIX System Services および言語環境プログラム (LE) の下で実行しているときには、異常終了または信号発信の場合にデフォルトの条件処理が起動します。 LE 条件処理ルーチンがエラー条件を処理し、TCB は正常に終了します。 これらの条件の下で、MQ は作業単位をコミットします。 詳しくは、「Language Environment の条件処理の入門」を参照してください。

z/OS IBM MQ for z/OS プログラムでは、バックアウトが起こった場合にメッセージをバックアウトしないように指定する MQGMO_MARK_SKIP_BACKOUT オプションを使用できます (*MQGET - エラー - バックアウト のループを防ぐため*)。 このオプションの使用方法については、799 ページの『バックアウトのスキップ』を参照してください。

キュー属性の変更 (MQSET 呼び出しまたはコマンドのいずれかによる) は、作業単位のコミットやバックアウトには影響されません。

z/OS *IBM MQ for z/OS アプリケーションにおける同期点*

このトピックでは、トランザクション・マネージャー (CICS および IMS) およびバッチ・アプリケーションで同期点を使用する方法を説明します。

z/OS *CICS Transaction Server for z/OS* アプリケーションにおける同期点

CICS アプリケーションで、EXEC CICS SYNCPOINT コマンドを使用して同期点を設定します。

以前の同期点に対するすべての変更をバックアウトするには、EXEC CICS SYNCPOINT ROLLBACK コマンドを使用できます。 詳細については、「*CICS* アプリケーション・プログラミング解説書」を参照してください。

作業単位内に他のリカバリー可能リソースが含まれている場合は、キュー・マネージャーは (CICS 同期点管理プログラムと共に) 2 フェーズ・コミット・プロトコルに関与します。それ以外の場合は、単一フェーズ・コミット・プロセスを実行します。

CICS アプリケーションが MQDISC 呼び出しを発行した場合、暗黙的な同期点は発生しません。 アプリケーションが正常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のコミットが行われます。 アプリケーションが異常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のバックアウトが行われます。

z/OS *IMS* アプリケーションにおける同期点

IMS アプリケーションでは、GU (固有の取得) などの IMS 呼び出しを IOPCB および CHKP (チェックポイント) に使用して、同期点を確立してください。

直前のチェックポイント以降のすべての変更をバックアウトするには、IMS ROLB (rollback) 呼び出しを使用できます。 詳しくは、IMS の資料を参照してください。

他のリカバリー可能リソースも作業単位に含まれている場合は、キュー・マネージャーが (IMS 同期点管理プログラムと共に) 2 フェーズ・コミット・プロトコルに関与します。

すべてのオープン状態のハンドルは、IMS アダプターによって同期点でクローズされます (バッチまたは非メッセージ・ドリブン BMP 環境の場合を除く)。 これは、別のユーザーが次の作業単位を開始することもあり、また、IBM MQ のセキュリティー検査が、(MQPUT または MQGET 呼び出し時でなく) MQCONN、MQCONNX および MQOPEN 呼び出し時に実行されるためです。

しかし、入力待ち (WFI (Wait-for-Input)) または疑似入力待ち (PWFI (pseudo Wait-for-Input)) 環境では、IMS は、次のメッセージが到着するか、QC 状況コードがアプリケーションに戻されるまでは、IBM MQ にハンドルをクローズするよう通知しません。 アプリケーションが IMS 領域内で待機していて、これらのハンドルのいずれかが、トリガーされたキューに属している場合、それらのキューはオープンしているのでトリガーは発生しません。 この理由のため、 WFI または PWFI 環境で実行するアプリケーションは、次のメッセージ用に IOPCB に対する GU を行う前に、明示的な MQCLOSE を キュー・ハンドルに対して実行する必要があります。

IMS アプリケーション (BMP または MPP) が MQDISC 呼び出しを発行した場合は、オープン・キューはクローズされますが、暗黙の同期点はとられません。 アプリケーションが正常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のコミットが行われます。 アプリケーションが異常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のバックアウトが行われます。

z/OS *z/OS アプリケーションにおける同期点*

バッチ・アプリケーションの場合、IBM MQ 同期点管理呼び出し (MQCMIT および MQBACK) を使用できます。 以前のバージョンとの互換性のために、CSQBCMT および CSQBBAK は同義語として使用することができます。

注 : IBM MQ や Db2 などのさまざまなリソース管理プログラムが管理するリソースの更新を、1 つの作業単位内でコミットまたはバックアウトする必要がある場合は、RRS を使用することができます。 詳しくは、862 ページの『トランザクション管理とリカバリー可能リソース管理サービス』を参照してください。

MQCMIT 呼び出しを使用する変更のコミット

入力として、MQCONN または MQCONNX 呼び出しによって戻される接続ハンドル (*Hconn*) が必要です。

MQCMIT からの出力は、完了コードと理由コードです。 同期点は完了したが、キュー・マネージャーが直前の同期点以降の書き込みおよび読み取り操作をバックアウトした場合は、この呼び出しは警告付きで完了します。

MQCMIT 呼び出しの正常な完了によって、次のことがキュー・マネージャーに対して示されます。つまり、アプリケーションが同期点に到達し、以前の同期点以降に行われた読み取りおよび書き込み操作は永続的なものとされました。

失敗の応答が戻ってきたからといって、必ずしも MQCMIT が完了しなかったことを意味するわけではありません。 例えば、アプリケーションが MQRC_CONNECTION_BROKEN を受け取ることがあります。

MQCMIT 呼び出しの説明については、MQCMIT を参照してください。

MQBACK 呼び出しを使用する変更のバックアウト

入力として、接続ハンドル (*Hconn*) を指定する必要があります。 MQCONN または MQCONNX 呼び出しによって戻されるハンドルを使用します。

MQBACK からの出力は完了コードと理由コードです。

この出力は、キュー・マネージャーに対して、アプリケーションが同期点に到達したこと、および最後の同期点以降に行われたすべての読み取りおよび書き込みがバックアウトされたことを示します。

MQBACK 呼び出しの説明については、MQBACK を参照してください。

トランザクション管理とリカバリー可能リソース管理サービス

トランザクション管理とリカバリー可能リソース管理サービス (RRS) は z/OS の機能の 1 つで、参加するすべてのリソース管理プログラムに 2 フェーズ同期点サポートを提供します。

アプリケーションは、 IBM MQ や Db2 などのさまざまな z/OS リソース・マネージャーによって管理されるリカバリー可能リソースを更新してから、それらの更新を単一の作業単位としてコミットまたはバックアウトすることができます。 RRS は、通常の実行時に必要な作業単位状況のログ記録を提供し、同期点処理を調整し、サブシステム再始動時に該当する作業単位状況情報を提供します。

IBM MQ for z/OS RRS 参加者サポートにより、バッチ、TSO、および Db2 ストアード・プロシージャー環境の IBM MQ アプリケーションは、 IBM MQ リソースと非 IBM MQ リソースの両方を更新できます (例えば、 Db2)。 単一の作業論理単位内で使用することができます。 RRS 参加プログラム・サポートについては、「*z/OS MVS プログラミング: リソース・リカバリー*」を参照してください。

IBM MQ アプリケーションでは、MQCMIT と MQBACK を使用することもできますし、同じ機能の RRS 呼び出しである SRRCMIT と SRRBACK を使用することもできます。 詳しくは、896 ページの『RRS バッチ・アダプター』を参照してください。

RRS の可用性

z/OS システムで RRS が活動状態になっていないと、RRS スタブ (CSQBRSTB または CSQBRRSI) とリンクされたプログラムが発行する IBM MQ 呼び出しはすべて、MQRC_ENVIRONMENT_ERROR を戻します。

Db2 ストアード・プロシージャー

RRS で Db2 ストアード・プロシージャーを使用する場合は、以下の点に注意してください。

- RRS を使用する Db2 ストアード・プロシージャーは、ワークロード・マネージャー (WLM) で管理する必要があります。
- Db2 が管理するストアード・プロシージャーが IBM MQ 呼び出しを含み、RRS スタブ (CSQBRSTB または CSQBRRSI) のいずれかとリンクされている場合、MQCONN または MQCONNX 呼び出しは MQRC_ENVIRONMENT_ERROR を戻します。
- WLM が管理するストアード・プロシージャーが IBM MQ 呼び出しを含み、RRS 以外のスタブとリンクされている場合、MQCONN または MQCONNX 呼び出しは MQRC_ENVIRONMENT_ERROR を戻します。ただし、ストアード・プロシージャー・アドレス・スペース開始後の最初の IBM MQ 呼び出しである場合は例外です。
- Db2 ストアード・プロシージャーが IBM MQ 呼び出しを含み、RRS 以外のスタブとリンクされている場合、そのストアード・プロシージャー内で更新された IBM MQ リソースは、ストアード・プロシージャー・アドレス・スペースが終了するか、後続のストアード・プロシージャーで (IBM MQ バッチ/TSO スタブを使って) MQCMIT が実行されるまで、コミットされません。
- 1 つのストアード・プロシージャーの複数のコピーを、同じアドレス・スペースで同時に実行することができます。 Db2 でストアード・プロシージャーのコピーを 1 つしか使用しないようにする場合は、プログラムを再入可能方式でコーディングしてください。 再入可能方式でコーディングされていないと、プログラムで使用する IBM MQ 呼び出しで MQRC_HCONN_ERROR が戻されることがあります。
- WLM で管理する Db2 ストアード・プロシージャーには、MQCMIT または MQBACK をコーディングしないでください。
- プログラムはすべて、Language Environment (言語環境プログラム) (LE) で実行するように設計してください。

IBM i *CICS for IBM i アプリケーションの同期点*

IBM MQ for IBM i は、CICS for IBM i 作業単位に参加します。 CICS for IBM i アプリケーションの中で MQI を使用して、現在の作業単位内にメッセージを配置および取得することができます。

EXEC CICS SYNCPOINT コマンドを使用して、 IBM MQ for IBM i 操作を含む同期点を設定できます。 前回の同期点まですべての変更をバックアウトする場合は、EXEC CICS SYNCPOINT ROLLBACK コマンドを使用できます。

CICS for IBM i アプリケーションで MQPMO_SYNCPOINT または MQGMO_SYNCPOINT オプションを設定した MQPUT、 MQPUT1、または MQGET を使用する場合は、 IBM MQ for IBM i が API コミットメント・リソースとしての登録を削除するまで、 CICS for IBM i をログオフできません。 キュー・マネージャーから切断する前に、保留中の書き込みまたは読み取り操作をすべてコミットするか、バックアウトしてください。 このようにすると、CICS for IBM i をログオフできるようになります。

Multi *IBM MQ for Multiplatforms の同期点*

同期点のサポートは、ローカルとグローバルという 2 種類の作業単位を対象とします。

ローカル 作業単位とは、IBM MQ キュー・マネージャーのリソースのみが更新対象のリソースとなる作業単位をいいます。 同期点調整は、キュー・マネージャー自身が単一フェーズ・コミット・プロシージャーにより行います。

グローバル 作業単位は、他のリソース管理プログラムのリソース (データベースなど) も更新する作業単位です。 IBM MQ は、このような作業単位を調整できます。 また、これらの作業単位は、外部コミットメント制御プログラムで調整することもできます。 以下に例を示します。

- 別のトランザクション・マネージャー
- IBM i IBM i コミットメント制御プログラム

データの保全性を失わないようにするために、2 フェーズ・コミット・プロシージャーを使用してください。 2 フェーズ・コミットは、XA 準拠トランザクション管理プログラムとデータベースでも提供されています。 以下に例を示します。

- TXSeries
- UDB
- IBM i IBM i コミットメント制御プログラム

ALW IBM MQ 製品は、2 フェーズ・コミット・プロセスを使用してグローバル作業単位を調整できます。

IBM i IBM MQ for IBM i は、WebSphere Application Server 環境の中で、グローバル作業単位のリソース管理プログラムとして作動することはできますが、トランザクション・マネージャーとして作動することはできません。

暗黙の同期点

持続メッセージを書き込むときには、同期点で持続メッセージを書き込むために IBM MQ が最適化されます。 複数のアプリケーションが同じキューに持続メッセージを書き込む場合には、それらのアプリケーションが同期点を使用するとパフォーマンスが向上します。 これは、同期点を使用して持続メッセージを書き込むと、キューの競合が少なくなるためです。

ImplSyncOpenOutput を使用すると、アプリケーションが同期点以外で持続メッセージを書き込んだ場合に、暗黙の同期点が追加されます。 これを使用すると、アプリケーションは暗黙の同期点を意識せずに実行できるので、パフォーマンスが向上します。

暗黙の同期点によってパフォーマンスが向上するのは、キューに書き込みを行うアプリケーションが複数存在する場合のみです。これは、この同期点によりキューの競合が減るためです。 このため、**ImplSyncOpenOutput** には、出力用にキューを開いているアプリケーションの最小数を指定します。この値に達すると、暗黙の同期点が追加されます。 デフォルト値は 2 です。 これは、**ImplSyncOpenOutput** を指定しない場合、複数のアプリケーションがキューに書き込みを行っている場合にのみ暗黙の同期点が追加されることを意味します。

詳しくは、チューニング・パラメーターを参照してください。

Multiplatforms でのローカル作業単位

キュー・マネージャーのみに関連のある作業単位のことをローカル 作業単位と呼びます。 同期点調整は、単一フェーズ・コミット・プロセスを使用して、キュー・マネージャー自体 (内部調整) によって提供されます。

ローカル作業単位を開始するには、アプリケーションが適切な同期点オプションを指定して MQGET、MQPUT、または MQPUT1 要求を発行します。 作業単位は MQCMIT によりコミットされるか、または MQBACK によりロールバックされます。 ただし、(故意にかどうかを問わず) アプリケーションとキュー・マネージャー間の接続が切断された場合にも、作業単位は終了します。

IBM MQ が調整するグローバル作業単位が活動状態のときでも、アプリケーションがキュー・マネージャーから切断されると (MQDISC)、作業単位のコミットが試みられます。 ただし、アプリケーションが切断されずに終了した場合には異常終了したものと見なされるため、作業単位はロールバックされます。

Multiplatforms でのグローバル作業単位
グローバル作業単位は、他のリソース管理プログラムのリソースも更新する必要がある場合に使用します。

この調整は、キュー・マネージャーの内部または外部で行うことができます。

内部同期点調整

キュー・マネージャーによるグローバル作業単位の調整は、IBM MQ for IBM i や IBM MQ for z/OS ではサポートされません。 これは、IBM MQ MQI client 環境ではサポートされません。

調整は、IBM MQ によって行われます。 グローバル作業単位を開始するには、アプリケーションが MQBEGIN 呼び出しを発行します。

MQBEGIN 呼び出しの入力として、接続ハンドル (*Hconn*) を必ず指定してください。このハンドルは MQCONN または MQCONNX 呼び出しから戻されます。 このハンドルは、IBM MQ キュー・マネージャーへの接続を表します。

アプリケーションは適切な同期点オプションを指定して MQGET、MQPUT、または MQPUT1 要求を発行します。 これは、ローカル・リソース、他のリソース管理プログラムのリソース、またはその両方を更新するグローバル作業単位を MQBEGIN により開始できることを意味します。 他のリソース管理プログラムのリソースの更新は、そのリソース管理プログラムの API を使用して行います。 ただし、MQI を使用して、他のキュー・マネージャーに属するキューを更新することはできません。 MQCMIT または MQBACK を発行してから、後続の作業単位 (ローカルまたはグローバル) を開始してください。

グローバル作業単位は MQCMIT によりコミットされます。MQCMIT により、その作業単位に関連のあるすべてのリソース管理プログラムの 2 フェーズ・コミットが開始されます。 2 フェーズ・コミット処理では、まず、すべてのリソース管理プログラム (Db2、Oracle、Sybase などの XA 準拠のデータベース・マネージャー) に、コミットできる状態であるかどうかを問い合わせます。 すべてのリソース管理プログラムがコミットできる状態である場合にのみ、作業単位をコミットするよう指示します。 いずれかのリソース管理プログラムからコミットできないと通知された場合は、各リソース管理プログラムに作業単位をバックアウトするよう指示します。 あるいは、MQBACK を使用して、すべてのリソース管理プログラムの更新をロールバックすることもできます。

グローバル作業単位が活動状態のときでもアプリケーションが切断されると (MQDISC)、作業単位はコミットされます。 ただし、アプリケーションが切断されずに終了した場合には異常終了したものと見なされるため、作業単位はロールバックされます。

MQBEGIN からの出力は完了コードと理由コードです。

MQBEGIN を使用してグローバル作業単位を開始した場合は、キュー・マネージャーを使用して構成されたすべての外部リソース管理プログラムが対象となります。 ただし、この呼び出しで作業単位を開始しても、次のどちらかの場合にこの呼び出しは完了して警告が発行されます。

- 参加するリソース管理プログラムがない場合 (つまり、キュー・マネージャーを使用して構成されたリソース管理プログラムがない場合)

または

- 1 つまたは複数のリソース管理プログラムが使用不可の場合

上記の場合は、作業単位を開始したときに使用可能であったリソース管理プログラムのみの更新を、作業単位に含めなければなりません。

いずれかのリソース管理プログラムが更新をコミットできない場合は、すべてのリソース管理プログラムが更新をロールバックするよう指示され、MQCMIT は警告付きで終了します。 まれに (通常はオペレーターの介入があった場合)、リソース管理プログラムの中で更新をコミットしたものとロールバックしたもの

があった場合、MQCMIT 呼び出しが失敗することがあります。この場合、作業は完了しますが、両方の処理結果が混ざり合って 生成されます。 このような障害が発生した場合は、キュー・マネージャーのエラー・ログでその原因を診断して、訂正処置を取ることができます。

グローバル作業単位の場合、MQCMIT 呼び出しが成功するのは、関連するすべてのリソース管理プログラムが更新をコミットした場合に限ります。

MQBEGIN 呼び出しの説明については、MQBEGIN を参照してください。

外部同期点調整

外部同期点調整は、IBM MQ 以外の同期点コーディネーター (CICS、Encina、Tuxedo など) が選択された場合に発生します。

この場合、IBM MQ for AIX, Linux, and Windows システムは、同期点コーディネーターに作業単位の結果の処理方法を登録し、コミットされない読み取りまたは書き込み操作を必要に応じてコミットまたはロールバックできるようにします。 外部同期点コーディネーターは、単一フェーズまたは 2 フェーズ・コミットメント・プロトコルが提供されたかどうかを判断します。

外部のコーディネーターを使用する場合、MQCMIT、MQBACK、および MQBEGIN は発行できません。 これらの関数を呼び出しても失敗し、理由コード MQRC_ENVIRONMENT_ERROR が戻されます。

外部整合の作業単位の開始方法は、同期点コーディネーターによって提供されるプログラミング・インターフェースによって異なります。 明示的な呼び出しが必要な場合もあります。 明示的な呼び出しが必要な場合､作業単位が開始されていないときに MQPMO_SYNCPOINT オプションを指定して MQPUT 呼び出しを発行すると、完了コード MQRC_SYNCPOINT_NOT_AVAILABLE が戻されます。

作業単位の有効範囲は、同期点コーディネーターによって判断されます。 アプリケーションとキュー・マネージャーとの間の接続状態は、アプリケーションが発行する MQI 呼び出しの成功か失敗かに影響を与えますが、作業単位の状況には影響はありません。 例えば、作業単位が活動状態のときでもアプリケーションはキュー・マネージャーとの接続を切断したり再接続したりすることが可能であり、さらに同じ作業単位内において別の MQGET および MQPUT 操作を実行することもできます。 これを、保留状態の切断といいます。

CICS の XA 機能を使用するかどうかに関係なく、 CICS プログラムで IBM MQ API 呼び出しを使用できます。 XA を使用しない場合、キューに対するメッセージの読み書きは、CICS アトミックの作業単位内では管理されません。 この方法を選択する 1 つの理由は、作業単位の全体の整合性が重要でないことです。

作業単位の整合性が重要である場合は、必ず XA を使用してください。 XA を使用すると、CICS で 2 フェーズ・コミット・プロトコルが使用されるため、作業単位内のすべてのリソースが一緒に更新されます。

トランザクション・サポートの設定に関する詳細は、「トランザクション・サポート・シナリオ」、および TXSeries CICS 資料を参照してください。例えば、「オープンシステム向けのマルチプラットフォーム *CICS* 管理ガイドの *TXSeries*」を参照してください。

Multi *Multiplatforms* での暗黙の同期点

暗黙の同期点をサポートすることによって、同期点以外で持続メッセージを書き込めるようになります。

持続メッセージを書き込むときには、同期点で持続メッセージを書き込むために IBM MQ が最適化されます。 複数のアプリケーションが同じキューに同時に持続メッセージを書き込む場合には、それらのアプリケーションが同期点を使用すると、通常はパフォーマンスが向上します。 これは、持続メッセージを書き込むときに同期点を使用すると、IBM MQ のロック戦略の効率が上がるためです。

qm.ini ファイルの **ImplSyncOpenOutput** パラメーターは、アプリケーションが同期点以外で持続メッセージを書き込むときに暗黙の同期点を追加できるかどうかを制御します。 これを使用すると、アプリケーションは暗黙の同期点を意識せずに実行できるので、パフォーマンスを向上させることができます。

暗黙の同期点によってパフォーマンスが向上するのは、同時にキューに書き込みを行うアプリケーションが複数存在する場合のみです。これは、この同期点によりロック競合が減るためです。
ImplSyncOpenOutput を指定すると、出力用にキューを開いているアプリケーションの数がこの最小値以上の場合に、暗黙の同期点を追加できます。 デフォルト値は *2* です。 つまり、**ImplSyncOpenOutput** を明示的に指定しない場合、暗黙の同期点は複数のアプリケーションが対象キューに書き込みを行うときにのみ追加されます。

暗黙の同期点を追加すると、統計にそのことが反映され、**runmqsc display conn** からのトランザクション出力で確認できます。

暗黙の同期点を追加しない場合には、**ImplSyncOpenOutput**=*OFF* を設定します。

詳しくは、チューニング・パラメーターを参照してください。

Multiplatforms の外部同期点管理プログラムへのインターフェース
IBM MQ for Multiplatforms は、X/Open XA インターフェースを使用する外部同期点管理プログラムによる、トランザクションの調整をサポートします。

一部の XA トランザクション管理プログラム (TXSeries) では、各 XA リソース管理プログラムがその名前を提供する必要があります。 これは、XA スイッチ構造体内の name というストリングです。

- ALW AIX, Linux, and Windows 上の IBM MQ のリソース・マネージャーの名前は MQSeries_XA_RMI です。
- IBM i IBM i の場合、リソース管理プログラムの名前は MQSeries XA RMI です。

XA インターフェースについて詳しくは、The Open Group が発行している XA 資料「*CAE Specification Distributed Transaction Processing: The XA Specification*」を参照してください。

XA 構成では、IBM MQ for Multiplatforms は XA リソース管理プログラムの役割を果たします。 XA 同期点コーディネーターは、一連の XA リソース管理プログラムを管理し、両方のリソース管理プログラムのトランザクションのコミットとバックアウトを同期します。 このように、XA 同期点コーディネーターは静的に登録されたリソース管理プログラムに対して機能します。

1. アプリケーションは、トランザクションを開始させたいことを同期点コーディネーターに通知する。
2. 同期点コーディネーターは、認識している任意のリソース管理プログラムに呼び出しを発行し、現行のトランザクションについて通知する。
3. アプリケーションは、現行のトランザクションに関連したリソース管理プログラムによって管理されているリソースを更新するために、呼び出しを発行する。
4. アプリケーションは、同期点コーディネーターがトランザクションをコミット、またはロールバックするように要求を出す。
5. 同期点コーディネーターは、2 フェーズ・コミット・プロトコルを使用して、各リソース管理プログラムに呼び出しを発行し、要求どおりにトランザクションを完了する。

XA 仕様は、各リソース管理プログラムに、XA Switch という構造体を提供するよう要求を出します。 この構造体では、リソース管理プログラムの機能と、同期点コーディネーターによって呼び出される関数を宣言します。

この構造体には次の 2 種類があります。

表 *128. XA* スイッチのバージョン

バージョン	**説明**
MQRMIXASwitch	静的 XA リソース管理
MQRMIXASwitchDynamic	動的 XA リソース管理

この構造体を含むライブラリーのリストについては、「IBM MQ XA スイッチの構造」を参照してください。

これらを XA 同期点コーディネーターにリンクさせるために使用する必要のある方式はコーディネーターによって定義されています。そのコーディネーターの文書を参照して、IBM MQ が XA 同期点コーディネーターと連携するための方法を判別してください。

同期点コーディネーターによる *xa_open* 呼び出しで渡される *xa_info* 構造体は、管理されるキュー・マネージャーの名前にすることができます。 これは、MQCONN または MQCONNX に渡されるキュー・マネージャー名と同じ形式であり、デフォルトのキュー・マネージャーが使用される場合にブランクになります。ただし、TPM と AXLIB という 2 つの追加のパラメーターを使用することができます。

TPM では、IBM MQ にトランザクション管理プログラム名 (例えば CICS など) を指定することができます。AXLIB では、XA AX 入り口点がある、トランザクション管理プログラム内の実際のライブラリー名を指定できます。

いずれかのパラメーターや、デフォルト以外のキュー・マネージャーを使用する場合は、QMNAME パラメーターを使用してキュー・マネージャー名を指定する必要があります。 詳細については、xa_open ストリングの CHANNEL、TRPTYPE、CONNAME、および QMNAME パラメーターを参照してください。

制約事項

1. グローバル作業単位は、共用 Hconn では許可されません (739 ページの『MQCONNX による共用 (スレッド独立) 接続』を参照)。
2. IBM i IBM MQ for IBM i は XA リソース管理プログラムの動的登録をサポートしません。

   サポートされる唯一のトランザクション・マネージャーは、WebSphere Application Server です。
3. Windows Windows システムでは、XA スイッチで宣言されるすべての関数は、_cdecl 関数として宣言されます。
4. 外部の同期点コーディネーターでは、一度に 1 つのキュー・マネージャーしか管理できません。 これは、各キュー・マネージャーに対してコーディネーターが効果的に接続されていることを前提としているので、一時点に許される接続は 1 つだけという規則の制約を受けるからです。

   注 : 注: JEE サーバーで実行される JMS クライアント・アプリケーション (CLIENT JEE アプリケーション) にはこの制限がないため、単一の JEE サーバー管理トランザクションで、同じトランザクション内の複数のキュー・マネージャーを調整できます。 ただし、バインディング・モードで実行される JMS サーバー・アプリケーションは、依然として一時点に許される接続は 1 つだけという規則の制約を受けます。
5. 同期点コーディネーターを使用して実行されるアプリケーションはすべて、コーディネーターによって管理されるキュー・マネージャーだけに接続できます。それは、これらアプリケーションが、既にそのキュー・マネージャーと有効な接続関係にあるためです。 これらのアプリケーションは、MQCONN または MQCONNX を発行して接続ハンドルを取得し、MQDISC を発行してから終了する必要があります。あるいは、TXSeries CICS に出口 UE014015 を使用することができます。

IBM i *IBM i 外部同期点管理プログラムへのインターフェース*

IBM MQ for IBM i は、ネイティブの IBM i コミットメント制御を外部同期点コーディネーターとして使用できます。

スレッド依存 (共用) 接続は、コミットメント制御では許可されません。 IBM i のコミットメント制御機能について詳しくは、「*IBM i Programming: Backup and Recovery Guide SC21-8079*」を参照してください。

IBM i コミットメント制御機能を開始するには、STRCMTCTL システム・コマンドを使用します。 コミットメント制御を終了するには、ENDCMTCTL システム・コマンドを使用します。

注 : コミットメント定義有効範囲 のデフォルト値は、*ACTGRP です。 IBM MQ for IBM i の場合は、これを *JOB と定義しなければなりません。 以下に例を示します。

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i では、IBM MQ リソースへの更新のみを含むローカル作業単位を実行することもできます。 ローカル作業単位を選択するか、それとも IBM i によって調整されるグローバル作業単位に関与するものすべてを選択するかどうかは、MQPMO_SYNCPOINT か MQGMO_SYNCPOINT、または MQBEGIN を指定して、アプリケーションが MQPUT、MQPUT1、または MQGET を呼び出す時に、各アプリケーションが決定します。 そのような呼び出しが最初に発行された時点でコミットメント制御がアクティブでない場合、IBM MQ はローカル作業単位を開始します。この後でコミットメント制御が開始されるかどうかに関係なく、IBM MQ に接続するための後続のすべての作業単位においても、ローカル作業単位が使用されます。ローカル作業単位をコミットするには、MQCMIT を使用します。 ローカル作業単位をバックアウトするには、MQBACK を使用します。 IBM i コミット、および CL コマンド COMMIT などのロールバック呼び出しは、IBM MQ ローカル作業単位には何の影響もありません。

IBM MQ for IBM i をネイティブ IBM i コミットメント制御と共に外部同期点コーディネーターとして使用したい場合は、コミットメント制御を持つすべてのジョブがアクティブであり、単一スレッド・ジョブで IBM MQ を使用していることを確認してください。 コミットメント制御が既に開始しているマルチスレッド・ジョブで、MQPMO_SYNCPOINT または MQGMO_SYNCPOINT を指定して、MQPUT、MQPUT1 または MQGET を呼び出すと、その呼び出しは理由コード MQRC_SYNCPOINT_NOT_AVAILABLE を出して失敗します。

マルチスレッド・ジョブでローカル作業単位、および MQCMIT と MQBACK 呼び出しを使用することが可能です。

コミットメント制御を開始した後、MQPMO_SYNCPOINT または MQGMO_SYNCPOINT を指定した MQPUT、MQPUT1 または MQGET を呼び出すと、IBM MQ for IBM i は、コミットメント定義に API コミットメント・リソースとして自身を追加します。 これが、通常、ジョブの最初の呼び出しになります。 特定のコミットメント定義の下に登録された API コミットメント・リソースがある間は、定義に対するコミットメント制御を終了することはできません。

IBM MQ for IBM i は、キュー・マネージャーから切断されたときに、API コミットメント・リソースとしての登録を除去します (ただし、現行の作業単位内に保留中の MQI 操作がない場合に限ります)。

現行の作業単位内に保留中の MQPUT、MQPUT1 または MQGET 操作がある間に、キュー・マネージャーから切断しようとした場合は、IBM MQ for IBM i が API コミットメント・リソースとして登録されたままになります。これは、次のコミットまたはロールバックが通知されるようにするためです。 次の同期点に達すると、IBM MQ for IBM i は必要に応じて変更をコミットまたはロールバックします。 作業単位が活動状態のときでもアプリケーションはキュー・マネージャーとの接続を切断したり再接続したりすることが可能であり、さらに同じ作業単位内において別の MQGET および MQPUT 操作を実行することもできます (保留状態の切断)。

そのコミットメント定義に対して ENDCMTCTL システム・コマンドを実行しようとした場合は、メッセージ CPF8355 が発行され、保留中の変更が活動状態であったことを知らせます。 このメッセージは、ジョブが終了したときにジョブ・ログにも示されます。 この状態を避けるためには、キュー・マネージャーから切断する前に、保留中のすべての IBM MQ for IBM i 操作をコミットまたはロールバックしてください。 このように、ENDCMTCTL の前に COMMIT コマンドまたは ROLLBACK コマンドを使用すると、コミットメント終了の制御を正常に完了することができます。

IBM i コミットメント制御を外部同期点コーディネーターとして使用する場合は、MQCMIT、MQBACK、および MQBEGIN 呼び出しを発行できません。 これらの関数を呼び出しても失敗し、理由コード MQRC_ENVIRONMENT_ERROR が戻されます。

作業単位をコミットまたはロールバック (つまりバックアウト) するには、コミットメント制御をサポートしているいずれかのプログラム言語を使用してください。 以下に例を示します。

- CL コマンド: COMMIT および ROLLBACK
- ILE C プログラミング関数: _Rcommit および _Rrollback
- ILE RPG: COMMIT および ROLBK
- COBOL/400®: COMMIT および ROLLBACK

IBM i コミットメント制御を外部同期点コーディネーターとして IBM MQ for IBM i で使用している場合、IBM i は、IBM MQ が関与する、2 フェーズ・コミット・プロトコルを実行します。 各作業単位は 2 フェーズでコミットされるので、最初のフェーズでコミットするように決定すると、キュー・マネージャーが 2 番目のフェーズに対して使用できなくなることがあります。 これは、例えばキュー・マネージャーの内部ジョブが終了した場合に起きることがあります。 この場合、コミットを実行するジョブ・ログに、コミットまたはロールバック操作が失敗したことを示すメッセージ CPF835F が含まれます。 この前に出されるメッセージは、この問題がコミット操作中かロールバック操作中のどちらで発生したかにかかわらず、問題の原因を、および失敗した作業単位の論理作業単位 ID (LUWID) を示します。

問題が、準備された作業単位のコミットまたはロールバック中に、IBM MQ API コミットメント・リソースの失敗により発生した場合、WRKMQMTRN コマンドを使用して操作を完了し、トランザクションの整合性を復元できます。 コマンドを使うには、コミットおよびバックアウトする作業単位の LUWID を知っている必要があります。

トリガーによる IBM MQ アプリケーションの開始

トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

キューを取り扱う一部の IBM MQ アプリケーションは絶えず稼働しているので、これらを使用してキューに到着したメッセージをいつでも取り出すことができます。 しかし、キューに到着するメッセージの数が予測できないときは、これが望ましくないこともあります。 この場合は、取り出すメッセージがないときでもアプリケーションがシステム・リソースを消費する可能性があります。

IBM MQ は、取り出すことができるメッセージがある場合に自動的にアプリケーションを開始できる機能を提供します。 この機能は、トリガー操作 と呼ばれます。

チャネルのトリガー操作については、チャネルのトリガー 操作を参照してください。

トリガー操作とは

キュー・マネージャーは、特定の条件を、トリガー・イベント を構成するものとして定義します。

トリガー操作がキューに対して有効になっている場合にトリガー・イベントが発生すると、キュー・マネージャーはトリガー・メッセージ を開始キュー に送信します。 開始キューにトリガー・メッセージがある場合、トリガー・イベントが発生したことを意味しています。

キュー・マネージャーが生成したトリガー・メッセージは、永続的ではありません。 これによりロギングが減少し (結果的にパフォーマンスが改善され)、再始動中の重複が最小限になります。その結果、再始動の時間が短縮されます。

開始キューを処理するプログラムは、トリガー・モニター・アプリケーション と呼ばれ、トリガー・メッセージを読み取り、トリガー・メッセージの情報に基づいて適切な処理を行います。 通常この処理によって、他のアプリケーションが開始され、トリガー・メッセージを生成したキューが処理されます。 キュー・マネージャーから見ると、トリガー・モニター・アプリケーションは特別なものではなく、キュー (開始キュー) からメッセージを読み取るアプリケーションの 1 つにすぎません。

トリガー操作がキューに対して有効になっている場合は、そのキューに関連するプロセス定義オブジェクト を作成できます。 このオブジェクトには、トリガー・イベントを発生させたメッセージを処理するアプリケーションについての情報が入っています。 プロセス定義オブジェクトが作成されると、キュー・マネージャーはこの情報を抽出し、トリガー・モニター・アプリケーションが使用できるようにこの情報をトリガー・メッセージに入れます。 キューと関連するプロセス定義の名前は、*ProcessName* ローカル・キュー属性によって指定されます。 各キューがそれぞれ異なるプロセス定義を指定することができます。また、いくつかのキューが同じプロセス定義を共有することもできます。

チャネルの開始をトリガーする場合、プロセス定義オブジェクトを定義する必要はありません。 伝送キュー定義が代わりに使用されます。

トリガー操作は、 AIX, Linux, and Windows 上で稼働する IBM MQ クライアントによってサポートされます。 クライアント環境で稼働するアプリケーションは、クライアント・ライブラリーにリンクすることを除いて、完全な IBM MQ 環境で稼働するアプリケーションと同じです。 ただし、トリガー・モニターおよび開始されるアプリケーションは、同じ環境にある必要があります。

トリガー操作には以下が必要です。

アプリケーション・キュー
: アプリケーション・キュー はローカル・キューであり、トリガー操作がオンに設定されていると、条件が合致した場合にはトリガー・メッセージを書き込むように要求します。

プロセス定義
: アプリケーション・キューにはプロセス定義オブジェクト を関連付けることができ、そこにアプリケーション・キューからメッセージを取得するアプリケーションの詳細が保持されます。 (属性のリストについては、プロセス定義の属性を参照してください。)

  トリガーでチャネルを開始する場合、プロセス定義オブジェクトを定義する必要はないことに注意してください。

伝送キュー
: **トリガーでチャネルを開始する場合、伝送キューが必要です。**

Linux 以外のプラットフォームの伝送キューの場合、伝送キューの *TriggerData* 属性で、開始するチャネルの名前を指定できます。 これは、チャネルをトリガーする際にプロセス定義の代わりに使用できます。ただし、プロセス定義が作成されていない場合にのみ使用されます。

トリガー・イベント

トリガー・イベント とは、キュー・マネージャーによってトリガー・メッセージが生成される起因となるイベントのことです。 これは、通常、アプリケーション・キューに到着するメッセージですが、他の場合にも発生することがあります。 例については、875 ページの『トリガー・イベントの条件』を参照してください。

IBM MQ には一連のオプションがあり、これらのオプションを使用してトリガー・イベントを引き起こす条件を制御できます (879 ページの『トリガー・イベントの制御』を参照)。

トリガー・メッセージ

キュー・マネージャーは、トリガー・イベントを認識するとトリガー・メッセージ を作成します。 キュー・マネージャーは、開始するアプリケーションに関する情報をトリガー・メッセージ内にコピーします。 この情報は、アプリケーション・キューおよびアプリケーション・キューに関連付けられているプロセス定義オブジェクトから得られます。

トリガー・メッセージは、固定形式です (887 ページの『トリガー・メッセージの形式』を参照)。

開始キュー

開始キュー は、キュー・マネージャーがトリガー・メッセージを書き込むローカル・キューです。 開始キューを、別名キューまたはモデル・キューにすることはできません。

1 つのキュー・マネージャーは、複数の開始キューを所有できます。各開始キューは、1 つ以上のアプリケーション・キューに関連付けられます。

z/OS 共用キュー、つまり、キュー共用グループ内のキュー・マネージャーがアクセスできるローカル・キューを、IBM MQ for z/OS の開始キューにすることができます。

トリガー・モニター

トリガー・モニター は、継続的に稼働しているプログラムで、1 つ以上の開始キューを処理します。 トリガー・メッセージが開始キューに到達すると、 トリガー・モニターがそのメッセージを検索します。トリガー・モニターは、トリガー・メッセージの情報を使用します。 トリガー・モニターはアプリケーション・キューに到着するメッセージを取り出すアプリケーションを開始するコマンドを発行して、アプリケーション・キューの名前といった、トリガー・メッセージ・ヘッダーに入っている情報をこのコマンドに渡します。

すべてのプラットフォームにおいて、チャネル・イニシエーターという特別なトリガー・モニターが、チャネルの開始を制御します。

z/OS z/OS では、チャネル・イニシエーターは通常、手動で開始されます。または、キュー・マネージャーの始動 JCL で CSQINP2 を変更することによってキュー・マネージャーが始動するときに自動的に実行することができます。

Multi マルチプラットフォーム では、チャネル・イニシエーターはキュー・マネージャー始動時に自動的に開始されるか、または **runmqchi** コマンドで手動で開始できます。

詳細については、883 ページの『トリガー・モニターによる開始キュー処理』を参照してください。

トリガー操作の働きを理解するには、871 ページの図 95 を参照してください。これは、トリガー・タイプ FIRST (MQTT_FIRST) の例です。

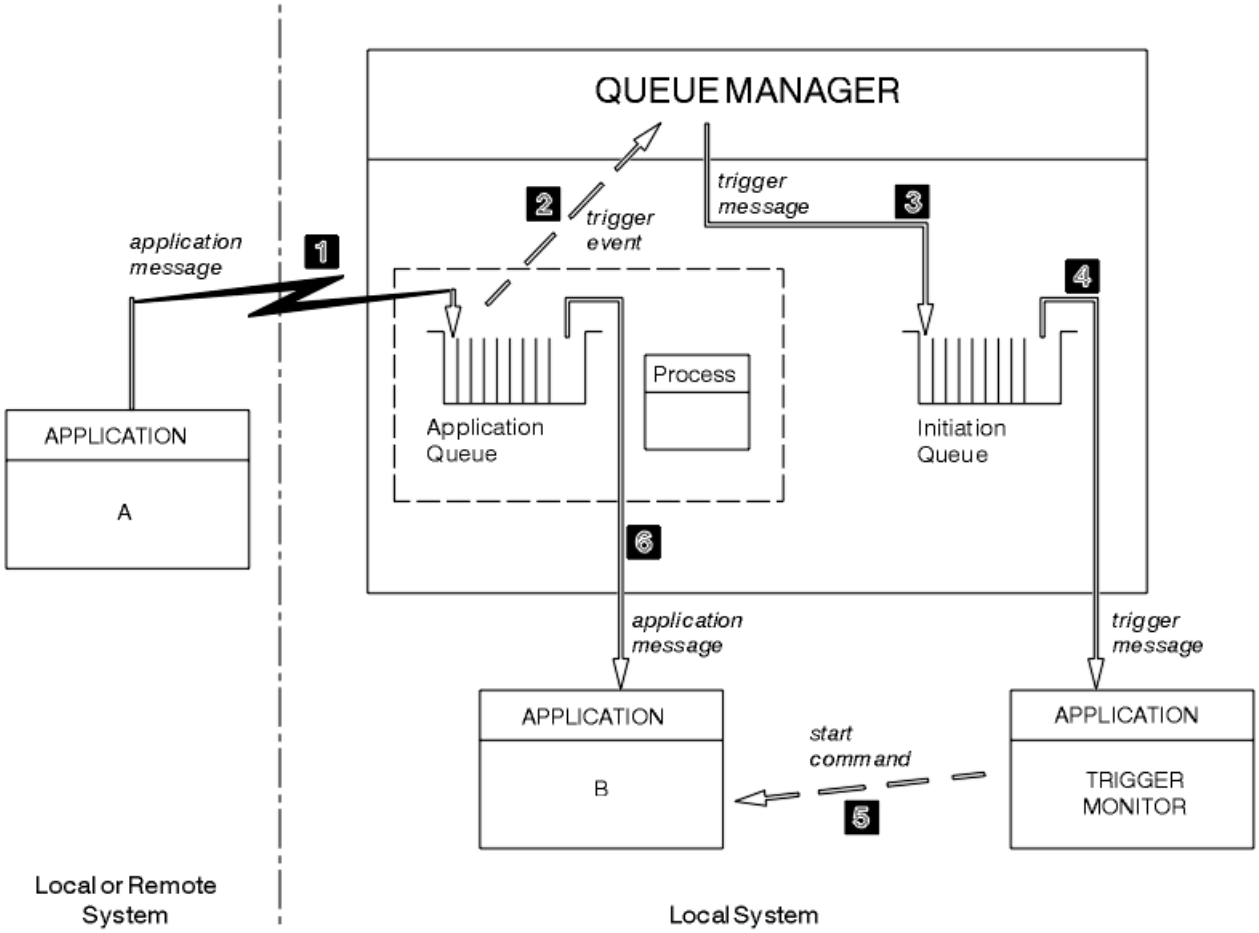


図 *95.* アプリケーションおよびトリガー・メッセージの流れ

871 ページの図 95 では、イベントの順序は以下のようになります。

1. アプリケーション A (キュー・マネージャーに対してローカルでもリモートでもよい) は、アプリケーション・キューにメッセージを書き込みます。 入力のためにこのキューをオープンしているアプリケーションはありません。 ただし、これが関係するのは、トリガー・タイプ FIRST および DEPTH のみです。
2. キュー・マネージャーは、この条件がトリガー・イベント生成の必要条件に合致するかどうかを検査する。 合致する場合は、トリガー・イベントが生成されます。 関連するプロセス定義オブジェクト内に保持されている情報が、トリガー・メッセージの作成時に使用されます。
3. キュー・マネージャーはトリガー・メッセージを作成し、このアプリケーション・キューに対応する開始キューにそれを書き込む。ただし、この処理が行われるのは、アプリケーション (トリガー・モニター) が入力のために開始キューをオープンする場合だけである。
4. トリガー・モニターは、開始キューからトリガー・メッセージを取り出します。
5. トリガー・モニターは、アプリケーション B (サーバー・アプリケーション) を開始するコマンドを発行します。
6. アプリケーション B はアプリケーション・キューをオープンし、メッセージを取り出します。

注:

1. 何らかのプログラムによって、アプリケーション・キューが既に入力のためにオープンされていて、FIRST または DEPTH に設定されているトリガー操作を持っている場合、トリガー・イベントが発生することはありません。キューが既に処理されているためです。
2. 入力のために開始キューがオープンされていない場合、キュー・マネージャーはトリガー・メッセージを生成せず、アプリケーションが入力のために開始キューをオープンするまで待ちます。

3. チャネルにトリガー操作を使用する場合は、トリガー・タイプとして FIRST または DEPTH を使用してください。
4. トリガーされたアプリケーションは、トリガー・モニターを開始したユーザー、CICS ユーザー、またはキュー・マネージャーを開始したユーザーのユーザー ID およびグループの下で実行されます。

ここまで、トリガー操作でのキュー間の関係は 1 対 1 対応のみでした。 872 ページの図 96 について考えます。

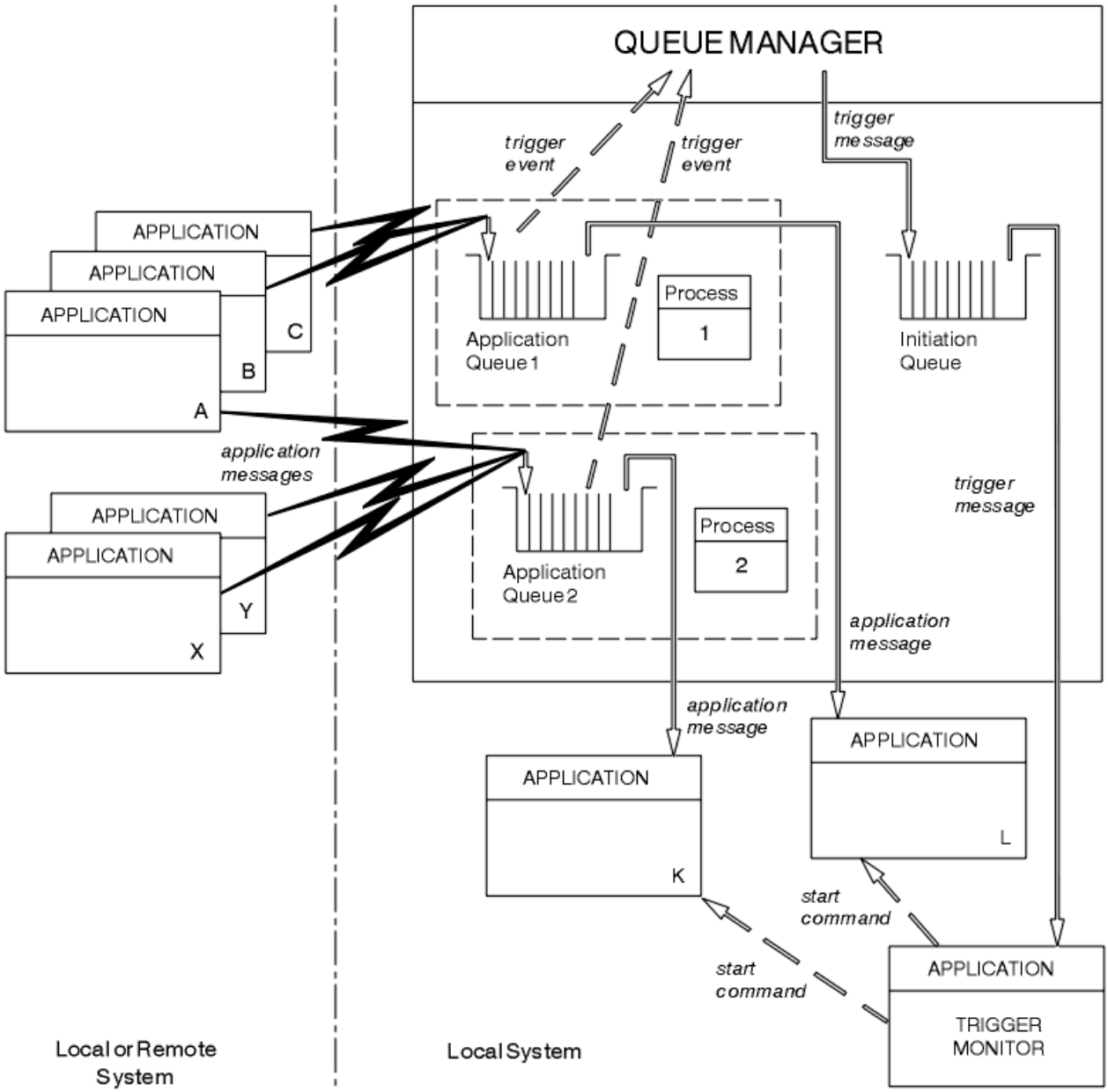


図 96. トリガー操作でのキューの関係

各アプリケーション・キューにはプロセス定義オブジェクトが関連付けられており、そこにメッセージを処理するアプリケーションについての詳細情報が保持されます。 キュー・マネージャーがトリガー・メッセージにこの情報を入れるので、必要な開始キューは 1 つだけです。 トリガー・モニターはこの情報をトリガー・メッセージから抽出し、関連するアプリケーションを開始して、各アプリケーション・キューのメッセージを処理します。

チャネルの開始をトリガーする場合、プロセス定義オブジェクトを定義する必要はないことに注意してください。 この場合は、伝送キュー定義により、トリガーするチャネルを判別できます。

以下のリンクを使用してトリガーを使用した IBM MQ アプリケーションの開始について詳しい情報を得ることができます。

- 873 ページの『トリガー操作の前提条件』
- 875 ページの『トリガー・イベントの条件』
- 879 ページの『トリガー・イベントの制御』
- 882 ページの『起動されたキューを使用するアプリケーションの設計』
- 883 ページの『トリガー・モニターによる開始キュー処理』
- 886 ページの『トリガー・メッセージの特性』
- 888 ページの『トリガー操作が動作しないとき』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

トリガー操作の前提条件

この情報を使用して、トリガー操作を使用する前に実行するステップについて学びます。

アプリケーションでトリガー操作を利用できるようにするには、以下のステップを実行します。

1. 以下のいずれかを実行します。
   a. アプリケーション・キューに対応する開始キューを作成する。 以下に例を示します。

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   または

   b. アプリケーションで使用できる既存のローカル・キュー名 (通常この名前は SYSTEM.DEFAULT.INITIATION.QUEUE、または、トリガー操作でチャネルを開始する場合は

SYSTEM.CHANNEL.INITQ) を調べて、アプリケーション・キューの *InitiationQName* フィールドにその名前を指定する。

2. 開始キューをアプリケーション・キューに関連付ける。 1 つのキュー・マネージャーで複数の開始キューを所有することができます。 いくつかのアプリケーション・キューを異なるプログラムで処理したいことがあります。この場合、各処理プログラムに対して 1 つの開始キューを使用することができますが、必ずしもそうする必要はありません。 以下に、アプリケーション・キューの作成例を示します。

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i 次に、開始キューを作成する IBM MQ for IBM i の CL プログラムの抜粋例を示します。

```
/*   Queue used by AMQSINQA                                        */
            CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                        QTYPE(*LCL)  REPLACE(*YES)                  +
                        MQMNAME                                     +
                        TEXT('queue for AMQSINQA')                  +
                        SHARE(*YES)                 /* Shareable   */+
                        DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                    +
                        TRGENBL(*YES)  /* Trigger control on      */+
                        TRGTYPE(*FIRST)/* Trigger on first message*/+
                        PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. アプリケーションを起動する場合は、プロセス定義オブジェクトを作成し、アプリケーション・キューを処理するアプリケーションに関する情報を入れる。 例えば、PAYR という CICS 給与計算トランザクションをトリガー操作によって開始するには、次のようにします。

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i 次に、プロセス定義オブジェクトを作成する IBM MQ for IBM i の CL プログラムの抜粋例を示します。

```
/*   Process definition                                            */
            CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                        REPLACE(*YES)                               +
                        MQMNAME                                     +
                        TEXT('trigger process for AMQSINQA')        +
                        ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                        APPID('AMQSINQA')    /* Program name      */
```

キュー・マネージャーは、トリガー・メッセージを作成すると、プロセス定義オブジェクトの属性から得た情報をトリガー・メッセージにコピーします。

プラットフォーム	プロセス定義オブジェクトを作成する場合
AIX, Linux, and Windows システム	DEFINE PROCESS を使用するか、SYSTEM.DEFAULT.PROCESS を使用して、ALTER PROCESS を使用して修正する

プラットフォーム	プロセス定義オブジェクトを作成する場合
z/OS z/OS	DEFINE PROCESS を使用する (ステップ 874 ページの『3』 のサンプル・コードを参照) か、または必要な操作および制御パネルを使用する
IBM i IBM i	コードを含む CL プログラムを使用する (ステップ 874 ページの『3』 を参照)

4. オプション: 伝送キュー定義を作成し、**ProcessName** 属性にはブランクを使用します。

   TrigData 属性については、起動するチャネルの名前を指定するか、ブランクのままにしてください。 IBM MQ for z/OS 以外では、ブランクのままにすると、チャネル・イニシエーターは、指定の伝送キューに関連するチャネルが見つかるまでチャネル定義ファイルを検索します。 キュー・マネージャーはトリガー・メッセージを作成するときに、伝送キュー定義の **TrigData** 属性から得た情報をトリガー・メッセージにコピーします。

5. アプリケーション・キューを処理するアプリケーションのプロパティーを指定するためにプロセス定義オブジェクトを作成した場合は、プロセス・オブジェクトをアプリケーション・キューに関連付ける。 これを行うには、アプリケーション・キューの **ProcessName** 属性にプロセス・オブジェクトの名前を指定します。

プラットフォーム	使用するコマンド
AIX, Linux, and Windows システム	ALTER QLOCAL
z/OS z/OS	ALTER QLOCAL
IBM i IBM i	CHGMQMQ

6. 定義した開始キューを処理するトリガー・モニター IBM i (または IBM MQ for IBM i のトリガー・サーバー) のインスタンスを開始する。 詳しくは、883 ページの『トリガー・モニターによる開始キュー処理』を参照してください。

未配布トリガー・メッセージについて知る必要がある場合、キュー・マネージャーに送達不能 (未配布メッセージ) キューが定義されていることを確認してください。 キュー名を *DeadLetterQName* キュー・マネージャー・フィールドに指定してください。

これで、アプリケーション・キューを定義するキュー・オブジェクトの属性を使用して、必要なトリガー条件を設定できます。 詳しくは、879 ページの『トリガー・イベントの制御』 を参照してください。

トリガー・イベントの条件

キュー・マネージャーは、本トピックで詳述する条件が満たされると、トリガー・メッセージを作成します。

このトピックで共用キューという語を用いる場合、それは IBM MQ for z/OS でのみ使用可能なキュー共用グループ内の共用キューを意味します。

以下の条件によって、キュー・マネージャーがトリガー・メッセージを作成します。

1. メッセージがキューに書き込まれた 場合。
2. メッセージが、キューのトリガー優先順位のしきい値と同等以上の優先順位を持っている場合。 この優先順位は、**TriggerMsgPriority** ローカル・キュー属性に設定され、それがゼロの場合は、すべてのメッセージが該当します。
3. 優先順位が *TriggerMsgPriority* と同等以上のキューのメッセージ数が、以前に次のような状態であった場合 (*TriggerType* に応じて異なる)。

- ゼロ (トリガー・タイプ MQTT_FIRST の場合)
- 任意の数 (トリガー・タイプ MQTT_EVERY の場合)
- *TriggerDepth* から 1 を引いた値 (トリガー・タイプ MQTT_DEPTH の場合)

注 :

a. 非共用ローカル・キューについては、キュー・マネージャーは、トリガー・イベントの条件が存在するかどうかを判断するときに、コミットされたメッセージとコミットされていないメッセージの両方をカウントします。 キューのメッセージがコミットされていないため、アプリケーションが検索を行う対象のメッセージがない場合には、結果的にアプリケーションが開始される場合があります。 この場合には、アプリケーションがメッセージの到着を待つようにするために、適切な *WaitInterval* を指定した待機オプションを使用することを考慮してください。

b. ローカル共用キューでは、コミットされたメッセージだけがキュー・マネージャーによってカウントされます。

4. タイプ FIRST または DEPTH のトリガー操作については、メッセージを除去するためにオープンされたアプリケーション・キューを持つプログラムがない場合 (つまり、**OpenInputCount** ローカル・キュー属性がゼロ)。

注 :

a. 共用キューでは、1 つのキューに対して複数のキュー・マネージャーがトリガー・モニターを実行している場合にのみ、特別な条件が適用されます。 この場合、1 つまたは複数のキュー・マネージャーが、入力用にオープンしたキューを共用している場合は、他のキュー・マネージャー上のトリガー基準は、*TriggerType* MQTT_FIRST、*TriggerMsgPriority* ゼロとして扱われます。 すべてのキュー・マネージャーが入力用のキューをクローズすると、トリガー条件はキュー定義で指定されている条件に戻ります。

この条件の影響を受けるシナリオの例として、アプリケーション・キュー A のために実行されるトリガー・モニターを含む複数のキュー・マネジャー QM1、QM2、および QM3 が挙げられます。 トリガーの条件を満たすメッセージが A に到着し、開始キューにトリガー・メッセージが生成されます。 QM1 上のトリガー・モニターが、トリガー・メッセージを取得し、アプリケーションをトリガーします。 トリガーされたアプリケーションが、共用入力用にアプリケーション・キューをオープンします。 この時点からアプリケーション・キュー A のトリガー条件が、キュー・マネージャー QM2 および QM3 上で *TriggerType* MQTT_FIRST、および *TriggerMsgPriority* ゼロと評価され、QM1 がアプリケーション・キューをクローズするまで続きます。

b. 共用キューについては、この条件は各キュー・マネージャーに適用されます。 つまり、あるキュー・マネージャーがキューのトリガー・メッセージを生成するには、そのキュー・マネージャーのキューの *OpenInputCount* がゼロでなければならないということです。 しかし、キュー共用グループ内に MQOO_INPUT_EXCLUSIVE オプションを使用してキューをオープンしているキュー・マネージャーが 1 つでもあれば、キュー共用グループ内のキュー・マネージャーがそのキューのトリガー・メッセージを生成することはありません。

トリガー条件の評価方法の変更は、トリガーされたアプリケーションが入力用にキューをオープンした場合に生じます。 トリガー・モニターが 1 つのみ実行されているシナリオでは、他のアプリケーションも同様に入力用にアプリケーション・キューをオープンするため、同じ影響を受ける可能性があります。 アプリケーション・キューが、トリガー・モニターによって起動されたアプリケーションによってオープンされたか、その他のアプリケーションによってオープンされたかは関係ありません。他のキュー・マネージャー上で入力用にキューがオープンされることが、トリガー条件を変更する原因となります。

5. IBM MQ for z/OS において、アプリケーション・キューの **Usage** 属性の値が MQUS_NORMAL である場合に、そのキューの読み取り要求が禁止されていない場合 (つまり、**InhibitGet** キュー属性が MQQA_GET_ALLOWED)。 また、起動されたアプリケーション・キューが MQUS_XMITQ の **Usage** 属性を持つキューである場合、そのキューに対する読み取り要求は禁止されません。
6. 以下のいずれかを実行します。
   - そのキューの **ProcessName** ローカル・キュー属性がブランクではなく、さらにその属性によって識別されるプロセス定義オブジェクトが作成されている場合。または、

- そのキューの **ProcessName** ローカル・キュー属性がブランクであり、そのキューが伝送キューである場合。 プロセス定義はオプションなので、**TriggerData** 属性には、開始するチャネルの名前が入る場合もあります。 この場合、トリガー・メッセージには次の値を持つ属性が入ります。
  - **QName**: キュー名
  - **ProcessName**: ブランク
  - **TriggerData**: トリガー・データ
  - **ApplType**: MQAT_UNKNOWN
  - **ApplId**: ブランク
  - **EnvData**: ブランク
  - **UserData**: ブランク

7. 開始キューが作成されていて、**InitiationQName** ローカル・キュー属性に指定されている場合。 さらに、次の状態になっている場合。
   - 読み取り要求が開始キューに対して禁止されていない (つまり、**InhibitGet** キュー属性の値が MQQA_GET_ALLOWED)。
   - その開始キューに対して、書き込み要求が禁止されていてはならない (つまり、**InhibitPut** キュー属性の値が MQQA_PUT_ALLOWED でなければならない)。
   - その開始キューの **Usage** 属性の値が MQUS_NORMAL でなければならない。
   - 動的キューがサポートされている環境では、開始キューは、論理的に削除されたとしてマークが付いている動的キューであってはならない。
8. 現在、メッセージの削除のためにオープンされた開始キューがトリガー・モニターにある (つまり、**OpenInputCount** ローカル・キュー属性がゼロより大きい) 場合。
9. アプリケーション・キューに対するトリガー制御 (**TriggerControl** ローカル・キュー属性) が MQTC_ON に設定されている場合。 これを行うには、キューを定義するときに **trigger** 属性を設定するか、ALTER QLOCAL コマンドを使用する。
10. トリガー・タイプ (**TriggerType** ローカル・キュー属性) が MQTT_NONE でない場合。

    必要条件がすべて満たされ、トリガー条件の原因となったメッセージが作業単位の一部として書き込まれる場合、作業単位がコミットされるか、またはトリガー・タイプ MQTT_FIRST の場合にはバックアウトされるかにかかわりなく、作業単位が完了するまではトリガー・モニター・アプリケーションによる取り出しにトリガー・メッセージを利用することはできません。
11. MQTT_FIRST または MQTT_DEPTH の **TriggerType** について、適切なメッセージがそのキューに入る場合。さらにそのキューが、次のどちらかの状態の場合。
    - 以前には空 (MQTT_FIRST) でなかった、または
    - **TriggerDepth** またはさらに他のメッセージ (MQTT_DEPTH) を持っていた

    また、MQTT_FIRST の場合に、このキューの最後のトリガー・メッセージが書き込まれてから十分な間隔 (**TriggerInterval** キュー・マネージャー属性) が経過していると、条件 875 ページの『2』 から 877 ページの『10』 (875 ページの『3』 を除く) が満たされます。

    これにより、キュー・サーバーはキューのメッセージをすべて処理する前に終了できます。 トリガー間隔の目的は、重複して生成されるトリガー・メッセージの数を減らすことです。

    注 : キュー・マネージャーをいったん停止して再始動すると、タイマー (**TriggerInterval**) はリセットされます。 2 つのトリガー・メッセージを生成できる間は、小さなウィンドウが表示されます。 このウィンドウは、メッセージが届いたときにキューのトリガー属性の設定が有効になっていて、キューが以前は空 (MQTT_FIRST) でなかった場合、あるいは **TriggerDepth** または他のメッセージ (MQTT_DEPTH) をもっていた場合に表示されます。
12. MQTT_FIRST または MQTT_DEPTH の **TriggerType** について、キューを処理する唯一のアプリケーションが MQCLOSE 呼び出しを発行し、少なくとも以下のメッセージが存在する場合。
    - 1 つの (MQTT_FIRST)、または

- **TriggerDepth** (MQTT_DEPTH)

これらのメッセージは十分な優先順位 (875 ページの『2』 の条件) のキューにあり、条件 876 ページの『6』 から 877 ページの『10』 も満たされている。

これにより、キュー・サーバーは MQGET 呼び出しを発行し、キューが空であることを検出し、終了できます。ただし、MQGET 呼び出しと MQCLOSE 呼び出しの間には、1 つまたは複数のメッセージが到着します。

注 :

a. アプリケーション・キューを処理するプログラムがすべてのメッセージを取り出さない場合は、閉じたループが発生する可能性があります。 プログラムがキューをクローズするたびに、キュー・マネージャーは、トリガー・モニターにサーバー・プログラムを再度開始させる、別のトリガー・メッセージを作成します。

b. アプリケーション・キューを処理するプログラムが、キューをクローズする前に、読み取り要求をバックアウトした (あるいは、プログラムが異常終了した) 場合にも、同様のことが起こります。 ただし、読み取り要求をバックアウトする前にプログラムがキューをクローズした場合、そのキューが別の理由で空であれば、トリガー・メッセージは作成されません。

c. このようなループが発生するのを防ぐには、MQMD の *BackoutCount* フィールドを使用して、繰り返しバックアウトされるメッセージを検出することができます。 詳しくは、46 ページの『バックアウトされるメッセージ』を参照してください。

13. MQSET またはコマンドを使用して以下の条件が満たされる場合。

a. • **TriggerControl** が MQTC_ON に変更される、または

- **TriggerControl** が既に MQTC_ON であり、**TriggerType**、**TriggerMsgPriority** または **TriggerDepth** (関係する場合) が変更され、

さらに、少なくとも以下のいずれかのメッセージが存在する場合。

- 1 つの (MQTT_FIRST か MQTT_EVERY)、または
- **TriggerDepth** (MQTT_DEPTH)

これらのメッセージは十分な優先順位 (875 ページの『2』 の条件) のキューにあり、条件 876 ページの『4』 から 877 ページの『10』 (877 ページの『8』 は除く) も満たされている。

トリガー操作が行われる条件が既に満たされている場合は、上記の条件により、アプリケーションまたはオペレーターがトリガー操作の基準を変更できます。

b. 開始キューの **InhibitPut** キュー属性が MQQA_PUT_INHIBITED から MQQA_PUT_ALLOWED に変更され、少なくとも次のメッセージが存在する場合。

- 1 つの (MQTT_FIRST か MQTT_EVERY)、または
- **TriggerDepth** (MQTT_DEPTH)

これらのメッセージには、任意のキューのうち、開始キューとなるキューの十分な優先順位 (875 ページの『2』 の条件) があり、条件 876 ページの『4』 から 877 ページの『10』 も満たされている。 (トリガー・メッセージは、条件を満たすキューごとに 1 つ生成される。)

開始キューでの MQQA_PUT_INHIBITED 条件のためにトリガー・メッセージは生成されませんでしたが、現在はこの条件が変更されています。

c. アプリケーション・キューの **InhibitGet** キュー属性の値が、MQQA_GET_INHIBITED から MQQA_GET_ALLOWED に変更され、少なくとも次のメッセージが存在する場合。

- 1 つの (MQTT_FIRST か MQTT_EVERY)、または
- **TriggerDepth** (MQTT_DEPTH)

これらのメッセージには、キューに十分な優先順位 (875 ページの『2』 の条件) があり、条件 876 ページの『4』 から 877 ページの『10』 (876 ページの『5』 を除く) も満たされている。

これにより、アプリケーション・キューからメッセージを取り出せる場合にだけそのアプリケーションをトリガーできます。

d. トリガー・モニター・アプリケーションが、開始キューからの入力のために MQOPEN 呼び出しを発行し、少なくとも次のメッセージが存在する場合。

- 1 つの (MQTT_FIRST か MQTT_EVERY)、または
- **TriggerDepth** (MQTT_DEPTH)

これらのメッセージは、任意のアプリケーション・キューのうち、開始キューとなるキューに十分な優先順位 (条件 875 ページの『2』) があり、条件 876 ページの『4』 から 877 ページの『10』 (877 ページの『8』 を除く) も満たされている。さらに、入力のために開始キューをオープンしているアプリケーションがほかにない。(トリガー・メッセージは、条件を満たすキューごとに 1 つ生成される。)

このため、トリガー・モニターが稼働していない間に、メッセージがキューに到着することができ、キュー・マネージャーが再始動し、トリガー・メッセージ (非持続性の) がなくなります。

14. MSGDLVSQ が正しく設定されている場合。 MSGDLVSQ=FIFO を設定した場合、メッセージは先入れ先出し方式でキューに送信されます。 メッセージの優先順位は無視され、キューのデフォルト優先順位がメッセージに割り当てられます。 **TriggerMsgPriority** にキューのデフォルト優先順位よりも高い優先順位を設定した場合、メッセージは起動されません。 **TriggerMsgPriority** にキューのデフォルト優先順位と同等以下の優先順位を設定した場合は、FIRST、EVERY、および DEPTH タイプのときにトリガー操作が実行されます。 これらのタイプについては、 879 ページの『トリガー・イベントの制御』の **TriggerType** フィールドの説明を参照してください。

MSGDLVSQ=PRIORITY を設定し、かつメッセージの優先順位が *TriggerMsgPriority* フィールドと同等以上に設定された場合は、メッセージはトリガー・イベントに影響を与えるだけです。 この場合は、FIRST、EVERY、および DEPTH タイプのときにトリガー操作が実行されます。 例えば、優先順位が **TriggerMsgPriority** に設定された優先順位より低いメッセージを 100 個書き込んだ場合、トリガー操作用のキューの有効なサイズはゼロのままです。 次に、優先順位が **TriggerMsgPriority** と同等以上に設定された別のメッセージをキューに書き込んだ場合、キューの有効なサイズは 0 から 1 になり、**TriggerType** が FIRST でなければならないという条件が満たされます。

注 :

1. ステップ 877 ページの『12』 以降の場合 (アプリケーション・キューに到着したメッセージ以外の何らかのイベントが発生したことによりトリガー・メッセージが生成された場合)、トリガー・メッセージは作業単位の一部として書き込まれません。 また、**TriggerType** が MQTT_EVERY で、アプリケーション・キューに 1 つ以上のメッセージがある場合には、トリガー・メッセージは 1 つだけ生成されます。
2. IBM MQ が MQPUT 中にメッセージをセグメント化する場合は、すべてのセグメントがキューに正常に配置されるまで、トリガー・イベントは処理されません。 ただし、メッセージ・セグメントがキューに配置されると、IBM MQ は、トリガー操作を行うために、それらのセグメントを個々のメッセージとして扱います。 例えば、1 つの論理メッセージが 3 つに分割される場合、論理メッセージが最初に MQPUT されてセグメント化されるときには、1 つのトリガー・イベントだけが処理されます。 しかし、3 つのそれぞれのセグメントのトリガー・イベントは、それぞれのセグメントが IBM MQ ネットワーク内を移動するときに処理されます。
3. IBM MQ for z/OS の場合、共用キューがトリガー用にセットアップされ、共用キューをホストするカップリング・ファシリティーへの接続が失われると、トリガー・イベントが生成され、メッセージが開始キューに書き込まれる可能性があります。 これは、トリガー操作のために元の共用キュー・セットアップにメッセージが書き込まれなかった場合でも発生する可能性があります。 これは、「リスト通知ベクトル」で説明されているように、IXLVECTR マクロによるビットの過剰表示が原因です。

トリガー・イベントの制御

アプリケーション・キューを定義する属性をいくつか使用することにより、トリガー・イベントを制御します。 この情報には、EVERY、FIRST、および DEPTH というトリガー・タイプの使用例も含まれています。

トリガー操作を使用可能にしたり使用禁止にしたりすることも、トリガー・イベントに影響を与えるメッセージの数または優先順位を選択することもできます。 これらの属性の詳細については、オブジェクトの属性に記載されています。

関連する属性は、次のとおりです。

TriggerControl
この属性を使用すると、アプリケーション・キューに対してトリガー操作を使用可能にしたり、使用禁止にしたりできます。

TriggerMsgPriority
メッセージがトリガー・イベントに影響を与えるために必要な最低の優先順位。
TriggerMsgPriority 属性よりも優先順位の低いメッセージがアプリケーション・キューに到着した場合は、キュー・マネージャーがトリガー・メッセージを作成するかどうかを判別するときに、そのメッセージは無視されます。 *TriggerMsgPriority* 属性をゼロに設定した場合は、すべてのメッセージがトリガー・イベントに影響を与えます。

TriggerType
トリガー・タイプ NONE (これは、*TriggerControl* を OFF に設定したのと同様に、トリガー操作を使用不可にする) のほかに、以下のトリガー・タイプを使用して、トリガー・イベントに対するキューの感知性を設定できます。

EVERY
メッセージがアプリケーション・キューに到着するたびに、トリガー・イベントを発生させます。 アプリケーションの複数のインスタンスを開始させたいときに、このトリガー・タイプを使用します。

FIRST
アプリケーション・キュー上のメッセージの数が 0 から 1 に変更されるときだけ、トリガー・イベントを発生させます。 処理プログラムを、最初のメッセージがキューに到着したときに開始させ、処理するメッセージがなくなるまで続行させてから終了したい場合に、このトリガー・タイプを使用します。 このキューは、必ず空になるまで処理しなければなりません。 881 ページの『トリガー・タイプ FIRST の特別な場合』も参照してください。

DEPTH
アプリケーション・キュー上のメッセージの数が **TriggerDepth** 属性の値に達したときだけ、トリガー・イベントを発生させます。 このトリガー・タイプの典型的な使用例は、一連の要求に対してすべての応答を受信したときにプログラムを開始させる場合です。

DEPTH によるトリガー操作 : DEPTH によるトリガー操作では、キュー・マネージャーはトリガー・メッセージを作成したあと、(*TriggerControl* 属性を使用して) トリガー操作を使用禁止にします。 アプリケーションは、この後、(MQSET 呼び出しによって) トリガー操作自体を再度使用可能にしなければなりません。

トリガー操作を使用禁止にする処置は、同期点制御の下では行われないので、作業単位をバックアウトしただけではトリガー操作を再度使用可能にすることはできません。 プログラムがトリガー・イベントを生じた書き込み要求をバックアウトしたか、あるいはプログラムが異常終了した場合は、MQSET 呼び出しまたは ALTER QLOCAL コマンドを使用してトリガー操作を再度使用可能にしなければなりません。

TriggerDepth
DEPTH によるトリガー操作を使用するときに、トリガー・イベントを発生させるキュー上のメッセージ数。

キュー・マネージャーがトリガー・メッセージを作成するために満たす必要のある条件は、875 ページの『トリガー・イベントの条件』に記述されています。

トリガー・タイプ EVERY の使用例

自動車保険の請求を生成するアプリケーションを考えてみてください。 アプリケーションは、毎回同じ応答先キューを指定して、数多くの保険会社に請求メッセージを送信します。 この応答先キューにトリガー・タイプ EVERY を設定して、応答が到着するたびに、応答を処理するサーバーのインスタンスが起動されるように設定できます。

トリガー・タイプ FIRST の使用例

多数の支店を持つ企業で、毎日の業務の詳細を本社あてに送信する場合を考えてみてください。 すべての支店が一日の作業終了時にこれを同時に行い、本社には全支店からの詳細データを処理するアプリケーシ

ョンがあります。 本社に到着する最初のメッセージは、このアプリケーションを開始させるトリガー・イベントを発生させることができます。 このアプリケーションは、そのキューにメッセージがなくなるまで処理を続けます。

トリガー・タイプ DEPTH の使用例

飛行機の予約確認、ホテルの予約確認、レンタカーの手配、さらにトラベラーズ・チェックの注文を行うための単一の要求を作成する旅行代理店のアプリケーションを想定します。 アプリケーションは、これらの項目を 4 つの要求メッセージに分けて、各メッセージを別々の宛先に送信できます。 その応答先キューにトリガー・タイプ DEPTH を (値を 4 に) 設定して、4 つの応答がすべて到着したときにだけ再始動するように設定できます。

4 つの応答のうち最後の応答より前に別のメッセージ (おそらく別の要求からのメッセージ) が応答先キューに到着した場合は、要求しているアプリケーションが早めに起動されてしまいます。 これを避けるため、DEPTH トリガーを使用して 1 つの要求に対する複数の応答を収集する場合は、各要求について常に新規の応答先キューを使用してください。

トリガー・タイプ FIRST の特別な場合

トリガー・タイプが FIRST の場合、アプリケーション・キューに別のメッセージが到着したときに既にメッセージがあると、キュー・マネージャーは通常は別のトリガー・メッセージを作成しません。

しかし、キューを処理するアプリケーションは、実際にはキューをオープンしない場合があります (例えば、アプリケーションがシステムの問題で終了する場合があります)。 正しくないアプリケーション名がプロセス定義オブジェクトに書き込まれている場合には、キューを処理するアプリケーションはメッセージをまったく取り上げません。 これらの場合、別のメッセージがアプリケーション・キューに到着する場合は、このメッセージ (およびキューの他のメッセージ) の処理のために稼働しているサーバーはありません。

これに対処するため、キュー・マネージャーは、以下の状況ではさらにトリガー・メッセージを作成します。

- アプリケーション・キューに別のメッセージが到着した場合。ただしこれは、キュー・マネージャーがそのキューに対するトリガー・メッセージを最後に作成してから事前に定義された時間が経過している場合だけです。 この時間は、キュー・マネージャー属性の *TriggerInterval* で定義されます。 デフォルト値は 999 999 999 ミリ秒です。
- IBM MQ for z/OS では、開かれている開始キューを指名したアプリケーション・キューが定期的に走査されます。 前回のトリガー・メッセージの送信から *TRIGINT* ミリ秒が経過しており、キューがトリガー・イベントの条件を満たしていて、CURDEPTH がゼロより大きい場合は、トリガー・メッセージが生成されます。 この処理は、バック・ストップ・トリガー操作と呼ばれます。

アプリケーションで使用するトリガー間隔の値を決めるときには、以下の点を考慮してください。

- *TriggerInterval* の値を低く設定している場合で、アプリケーション・キューにサービスを提供しているアプリケーションがない場合は、トリガー・タイプ FIRST がトリガー・タイプ EVERY のような振る舞いをすることがあります。 これは、メッセージがアプリケーション・キューに書き込まれる速度に依存し、その速度は他のシステム活動に依存することがあります。 これは、トリガー間隔が非常に小さいと、トリガー・タイプが (EVERY ではなく) FIRST であっても、メッセージがアプリケーション・キューに書き込まれるたびに別のトリガー・メッセージが生成されるからです。 (トリガー間隔がゼロのトリガー・タイプ FIRST は、トリガー・タイプ EVERY と同等です。)
- IBM MQ for z/OS において、*TRIGINT* の値を低く設定している場合で、トリガー・タイプ FIRST のアプリケーション・キューにサービスを提供しているアプリケーションがない場合は、開かれている開始キューを指名したアプリケーション・キューの定期的な走査が行われるたびに、バック・ストップ・トリガー操作によってトリガー・メッセージが生成されます。
- 作業単位がバックアウトされる場合 (トリガー・メッセージと 作業単位を参照)、トリガー間隔が高い値 (またはデフォルト値) に設定されていると、 その作業単位のバックアウト時にトリガー・メッセージが 1 つ生成されます。 しかし、トリガー間隔を小さい値かゼロに設定している (トリガー・タイプ FIRST がトリガー・タイプ EVERY のように機能する) 場合は、多数のトリガー・メッセージが生成される可能性があります。 その作業単位がバックアウトされると、すべてのトリガー・メッセージがやはり使用可能に

なります。 生成されるトリガー・メッセージの数は、トリガー間隔によって異なります。 トリガー間隔がゼロに設定されている場合、最大数のメッセージが生成されます。

起動されたキューを使用するアプリケーションの設計

これまでは、アプリケーションに対してトリガー操作を設定し、制御する方法を検討してきました。 ここでは、アプリケーションを設計するときに考慮すべきいくつかのヒントを示します。

トリガー・メッセージと作業単位

作業単位の一部ではないトリガー・イベントのために作成されたトリガー・メッセージは、作業単位の外部で開始キューに書き込まれます。その際、他のメッセージには依存せず、トリガー・モニターによる検索のために即時に利用可能になります。

作業単位に属するトリガー・イベントによって作成されたトリガー・メッセージは、その UOW が解決されたとき (作業単位がコミットまたはバックアウトされたとき) に、開始キューで取得可能になります。

キュー・マネージャーが開始キューにトリガー・メッセージを書き込むことに失敗した場合、トリガー・メッセージは送達不能 (未配布メッセージ) キューに書き込まれます。

注 :

1. キュー・マネージャーは、トリガー・イベントの条件が存在するかどうかを判断するときに、コミットされたメッセージとコミットされていないメッセージの両方をカウントします。

   タイプ FIRST または DEPTH のトリガー操作を使用すると、作業単位がバックアウトされた場合でもトリガー・メッセージが使用可能になります。これにより、必要な条件が満たされたとき、いつでもトリガー・メッセージが使用可能になります。 例えば、トリガー・タイプ FIRST で起動されるキューに対して、作業単位内で書き込み要求が行われたと想定します。 これによって、キュー・マネージャーはトリガー・メッセージを作成します。 別の作業単位から別の書き込み要求が行われても、さらに別のトリガー・イベントは発生しません。アプリケーション・キューのメッセージ数が 1 から 2 に変わったので、トリガー・イベントの条件を満たさないためです。 ここで、最初の作業単位がバックアウトされ、2 番目がコミットされたとしても、やはりトリガー・メッセージは作成されます。

   しかし、これは、トリガー・イベントの条件が満たされないときにも、トリガー・メッセージが作成される場合があることを意味します。 トリガー操作を使用するアプリケーションは、この状態を処理する用意を常にしておく必要があります。 MQGET 呼び出しで待機オプションを使用して、*WaitInterval* を適切な値に設定することをお勧めします。

   作成されたトリガー・メッセージは、作業単位がバックアウトされた場合でもコミットされた場合でも、常に取得可能になります。
2. 共用ローカル・キュー (つまり、キュー共用グループ内の共用キュー) については、キュー・マネージャーはコミットされたメッセージだけをカウントします。

起動されたキューからのメッセージの読み取り

トリガー操作を使用するアプリケーションを設計するときは、トリガー・モニターがプログラムを開始してから、アプリケーション・キューで他のメッセージが使用可能になるまでの間に遅延が生じる場合があることに注意してください。 これは、トリガー・イベントを発生するメッセージが他のメッセージより前にコミットされたときに起こることがあります。

メッセージが到着するまでの時間を考慮して、トリガー条件が設定されるキューからメッセージを除去する MQGET 呼び出しを使用するときは、必ず待機オプションを使用してください。 *WaitInterval* は、メッセージが書き込まれ、その書き込み呼び出しがコミットされるまでの妥当な最長時間を考慮した十分な値にしなければなりません。 メッセージがリモート・キュー・マネージャーから送られてくる場合は、この時間は次の値によって影響を受けます。

- コミットされる前に書き込まれるメッセージの数
- 通信リンクの速度と使用可能度
- メッセージのサイズ

待機オプションを指定した MQGET 呼び出しを使用するのが望ましい状態の例として、作業単位を説明したときと同じ例を考えてみます。 この例は、トリガー・タイプ FIRST で起動されるキューに対する、作業単位内の書き込み要求でした。 このイベントにより、キュー・マネージャーはトリガー・メッセージを作成します。 別の作業単位から別の書き込み要求が行われても、さらに別のトリガー・イベントは発生しません。アプリケーション・キューのメッセージ数が 0 から 1 に変わっていないためです。 ここで、最初の作業単位がバックアウトされ、2 番目がコミットされたとしても、やはりトリガー・メッセージは作成されます。 したがって、トリガー・メッセージは最初の作業単位がバックアウトされたときに作成されます。 2 番目のメッセージがコミットされるまでにかなりの遅延がある場合は、起動されたアプリケーションが待機しなければならない場合があります。

タイプ DEPTH のトリガー操作を使用すると、関連したメッセージがすべてコミットされた場合でも、遅延が生じることがあります。 **TriggerDepth** キュー属性に値 2 が指定されているとします。 2 つのメッセージがキューに到着すると、2 番目のメッセージによってトリガー・メッセージが作成されます。 しかし、2 番目のメッセージが最初にコミットされる場合は、その時点でトリガー・メッセージが使用可能になります。 トリガー・モニターはサーバー・プログラムを開始させますが、プログラムは、最初のメッセージがコミットされるまで 2 番目のメッセージしか検索できません。 このため、最初のメッセージが使用可能になるまで、プログラムは待機しなければならない場合があります。

待機期間が満了したときに取り出すメッセージがない場合は、アプリケーションが終了するような設計にしておく必要があります。 1 つ以上のメッセージが後から到着した場合は、アプリケーションが再トリガーされた時点でそれらのメッセージが処理されることになります。 この方式によって、アプリケーションがアイドル状態になって、不必要にリソースを使用することのないようにします。

トリガー・モニターによる開始キュー処理

キュー・マネージャーにとって、トリガー・モニターは、キューを処理する他のアプリケーションのような存在です。 ただし、トリガー・モニターは開始キューを提供します。

トリガー・モニターは、通常、絶えず稼働しているプログラムです。 トリガー・メッセージが開始キューに到着すると、トリガー・モニターはそのメッセージを検索します。 トリガー・モニターは、メッセージ内の情報を使用して、アプリケーション・キューにあるメッセージを処理するアプリケーションを開始させるコマンドを実行します。

トリガー・モニターは、それによって開始されるプログラムに十分な情報を渡して、そのプログラムが適正なアプリケーション・キューに適正な処置を行うことができるようにしなければなりません。

チャネル・イニシエーターは、メッセージ・チャネル・エージェントに応答する特別なタイプのトリガー・モニターの例です。 しかし、この場合には、トリガー・タイプ FIRST または DEPTH のどちらかを使用する必要があります。

ALW *AIX, Linux, and Windows* システム上のトリガー・モニター

このトピックでは、AIX, Linux, and Windows システム用に提供されるトリガー・モニターについて説明します。

次のトリガー・モニターがサーバー環境用に提供されています。

amqstrg0
これは、**runmqtrm** によって提供される機能のサブセットを提供するサンプル・トリガー・モニターです。 amqstrg0 についての詳細は、1063 ページの『Multiplatforms でのサンプル・プログラムの使用』を参照してください。

runmqtrm
このコマンドの構文は **runmqtrm** [*-m QMgrName*] [*-q InitQ*] です。ここで、 QMgrName はキュー・マネージャー、 InitQ は開始キューです。 デフォルト・キューは、デフォルト・キュー・マネージャーの SYSTEM.DEFAULT.INITIATION.QUEUE です。 該当するトリガー・メッセージ用のプログラムを呼び出します。 このトリガー・モニターは、デフォルトのアプリケーション・タイプをサポートします。

トリガー・モニターからオペレーティング・システムに渡されるコマンド・ストリングは、次のように作成されます。

1. 関連する PROCESS 定義 (作成される場合) の *ApplId*

2. 二重引用符で囲まれた MQTMC2 構造体
3. 関連する PROCESS 定義 (作成される場合) の *EnvData*

ApplId は実行するプログラムの名前を示します。この名前は、コマンド行に入力されたとおりに表示されます。

渡されるパラメーターは、MQTMC2 文字構造体です。 二重引用符に囲まれたこのストリングと正確に同じストリングを持つコマンド・ストリングが呼び出されます。したがって、システム・コマンドには 1 つのパラメーターとして受け付けられます。

トリガー・モニターは、開始したばかりのアプリケーションが完了するまでは、開始キューに別のメッセージがあるかどうかを検査しません。 アプリケーションに多くの処理がある場合は、到着するトリガー・メッセージの数にトリガー・モニターが追いつかないことがあります。 この解決方法は、次の 2 つです。

- 実行するトリガー・モニターを増やす
- 開始済みのアプリケーションのバックグラウンドで実行する

実行するトリガー・モニターの数を増やすと、一度に実行できるアプリケーションの最大数を制御できます。 アプリケーションをバックグラウンドで実行する場合は、IBM MQ による、実行可能なアプリケーションの数に関する制約はありません。

Linux AIX AIX and Linux 上で、開始されたアプリケーションをバックグラウンドで実行するには、PROCESS 定義の *EnvData* の末尾に & を配置します。

Windows システム上で、開始済みのアプリケーションをバックグラウンドで稼働させるには、*ApplId* フィールド内でアプリケーション名の前に START コマンドを追加します。 例えば、です

```
START ?B AMQSECHA
```

注 : Windows Windows のパスで、パス名の中にスペースが含まれている場合は、パスを引用符 (") で囲んで、それが 1 つの引数として扱われるようにする必要があります。 例えば、"C:\Program Files\Application Directory\Application.exe"などです。

以下は、パス中のファイル名にスペースが含まれている場合の APPLICID ストリングの例です。

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

この例の Windows START コマンドの構文には、二重引用符で囲まれた空のストリングがあります。 START コマンドは、引用符で囲われた最初の引数を新しいコマンドのタイトルとして扱うように指定します。 Windows がアプリケーションのパスを「タイトル」の引数と間違えないようにするため、コマンドのアプリケーション名の前に、二重引用符で囲ったタイトルのストリングを追加します。

以下のトリガー・モニターが、IBM MQ クライアント用に提供されています。

runmqtmc
これは、リンク先が IBM MQ MQI client・ライブラリーである点を除いて runmqtrm と同じです。

ALW *CICS* のトリガー・モニター

amqltmc0 トリガー・モニターが CICS 用に提供されています。 このトリガー・モニターは標準トリガー・モニター runmqtrm と同様に機能しますが、これを異なる方法で実行して、CICS トランザクションを起動します。

このトピックは、 Windows、 AIX and Linux x86-64 システムにのみ適用されます。

トリガー・モニターは CICS プログラムとして提供され、4 文字のトランザクション名で定義する必要があります。 4 文字の名前を入力して、トリガー・モニターを始動します。 デフォルトのキュー・マネージャー (qm.ini ファイルまたは IBM MQ for Windows の場合はレジストリーで指定)、および SYSTEM.CICS.INITIATION.QUEUE。

異なるキュー・マネージャーまたはキューを使用したい場合は、トリガー・モニター MQTMC2 構造体を作成します。これを行う場合、構造体はパラメーターとして追加するには長すぎるため、EXEC CICS START 呼び出しを使用してプログラムを作成する必要があります。 そこで、MQTMC2 構造体をデータとしてトリガー・モニター用の START 要求に渡します。

MQTMC2 構造体を使用する場合、それ以外のフィールドを参照しないため、トリガー・モニターに入力する必要があるのは、*StrucId*、*Version*、*QName*、および **QMgrName** パラメーターだけです。

メッセージは開始キューから読み取られ、EXEC CICS START を使用して CICS トランザクションの開始に使用されます。この際、トリガー・メッセージの APPL_TYPE は MQAT_CICS と想定しています。 開始キューからのメッセージの読み取りは、CICS 同期点制御の下で実行されます。

モニターの始動時、停止時、およびエラーの発生時に、メッセージが生成されます。 これらのメッセージは、CSMT 一時データ・キューに送られます。

表 *129.* トリガー・モニターの提供バージョン.

2 列の表。 1 列目にはトリガー・モニターのバージョンをリストし、2 列目には各バージョンの対象プラットフォームを示しています。

バージョン	**以下を使用してください。**
amqltmc0	TXSeries 回数: • AIX • Linux x86-64 システム
amqltmc4	Windows TXSeries for Windows 5.1
amqltmcc	CICS トリガー・モニターのクライアント・バインド・バージョン

他の環境用のトリガー・モニターが必要な場合は、キュー・マネージャーが開始キューに書き込むトリガー・メッセージを処理できるプログラムを作成してください。 このようなプログラムは、以下の操作を実行する必要があります。

1. メッセージが開始キューに到着するのを待機するために、MQGET 呼び出しを使用する。
2. 開始するアプリケーションの名前と、実行される環境を検索するために、トリガー・メッセージの MQTM 構造体のフィールドを調べる。
3. 環境に特有な開始コマンドを実行する。

   z/OS 例えば、z/OS バッチでは、ジョブを内部読み取りプログラムに実行依頼する。
4. 必要に応じて、MQTM 構造体を MQTMC2 構造体に変換する。
5. MQTMC2 または MQTM 構造体を開始済みのアプリケーションに渡す。 これにはユーザーのデータが含まれる場合がある。
6. アプリケーション・キューを、そのキューを処理するアプリケーションに対応させる。 これを行うには、キューの **ProcessName** 属性にプロセス定義オブジェクト (作成した場合) の名前を指定してください。 プロセス定義オブジェクトに名前を付けるには、 **DEFINE QLOCAL** コマンドまたは **ALTER QLOCAL** コマンドを使用できます。

   IBM i IBM i の場合は、CRTMQMQ または CHGMQMQ を指定することもできます。

トリガー・モニター・インターフェースの詳細については、MQTMC2 を参照してください。

IBM i *IBM i 上のトリガー・モニター*

IBM i では、**runmqtrm** 制御コマンドではなく、IBM MQ for IBM i CL コマンド **STRMQMTRM** を使用します。

STRMQMTRM コマンドを次のように使用します。

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

詳細は、runmqtrm と同じです。

以下のサンプル・プログラムも用意されています。これをモデルにして独自のトリガー・モニターを作成できます。

AMQSTRG4
これは、開始対象のプロセスのために IBM i ジョブを実行依頼するトリガー・モニターですが、これは各トリガー・メッセージに対応する追加処理があることを意味します。

AMQSERV4
これは、トリガー・サーバーです。 各トリガー・メッセージに対して、このサーバーは、トリガー・メッセージごとに、サーバー自体のジョブの処理のためのコマンドを実行し、CICS トランザクションを呼び出すことができます。

トリガー・モニターとトリガー・サーバーの両方とも、開始させるプログラムに MQTMC2 構造体を渡します。 この構造体の説明については、MQTMC2 を参照してください。 これらのサンプルはどちらも、ソース形式と実行可能形式の両方で提供されます。

これらのトリガー・モニターで起動では、ネイティブの IBM i プログラムしか起動できません。そのため、Java クラスは IFS にあるので、Java プログラムを直接トリガー操作することはできません。 しかし、Java プログラムを起動して TMC2 構造体を介して渡す CL プログラムをトリガー操作することによって、間接的に Java プログラムをトリガー操作することは可能です。 TMC2 構造体の最小サイズは 732 バイトです。

サンプル CLP のソースは次のとおりです。

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

トリガー・モニター・プログラム RUNMQTMC が IBM MQ MQI client 用に提供されています。

次のようにして RUNMQTMC を呼び出します。

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

トリガー・メッセージの特性

以下のトピックでは、トリガー・メッセージのその他の特性をいくつか説明します。

- 886 ページの『トリガー・メッセージの持続性と優先順位』
- 887 ページの『キュー・マネージャーの再始動とトリガー・メッセージ』
- 887 ページの『トリガー・メッセージとオブジェクト属性の変更』
- 887 ページの『トリガー・メッセージの形式』

トリガー・メッセージの持続性と優先順位

トリガー・メッセージは、持続性を持たせる必要はないので、持続的ではありません。

しかし、トリガー・イベントを生成する条件は持続的なので、これらの条件が満たされたときはいつでも、トリガー・メッセージが生成されます。 トリガー・メッセージが失われる場合でも、アプリケーション・キューのアプリケーション・メッセージはそのまま存在するので、条件がすべて満たされた場合にはすぐに、キュー・マネージャーがトリガー・メッセージを生成します。

作業単位がロールバックされる場合、その作業単位によって生成されるトリガー・メッセージはどれでも常に送達されます。

トリガー・メッセージの優先順位は、開始キューのデフォルト優先順位になります。

キュー・マネージャーの再始動とトリガー・メッセージ

キュー・マネージャーの再始動に続いて、入力のために開始キューが次にオープンされると、トリガー・メッセージは、それに対応するアプリケーション・キューにメッセージが存在し、かつトリガー操作のために定義されている場合に、この開始キューに書き込まれます。

トリガー・メッセージとオブジェクト属性の変更

トリガー・メッセージは、トリガー・イベントの発生時に効力があるトリガー属性の値に従って作成されます。

(生成原因となったメッセージが作業単位内で書き込まれたため) トリガー・モニターがすぐにトリガー・メッセージを使用できない場合には、その間にトリガー属性が変更されても、トリガー・メッセージには影響がありません。 特に、トリガー操作を使用禁止にした場合でも、いったん生成されたトリガー・メッセージは、使用禁止にはできません。 また、トリガー・メッセージが使用可能になった時点では、アプリケーション・キューが既に存在していない場合もあります。

トリガー・メッセージの形式

トリガー・メッセージの形式は、MQTM 構造体によって定義されます。

これは、以下のフィールドを持ち、これらのフィールドは、キュー・マネージャーがトリガー・メッセージを作成するとき、アプリケーション・キューおよびそのキューと関連する処理のオブジェクト定義にある情報を使用して書き込まれます。

StrucId
構造体の ID。

Version
構造体のバージョン。

QName
トリガー・イベントが発生したアプリケーション・キューの名前。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの **QName** 属性を使用して、このフィールドに値を入れます。

ProcessName
アプリケーション・キューに関連したプロセス定義オブジェクトの名前。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの **ProcessName** 属性を使用して、このフィールドに値を入れます。

TriggerData
トリガー・モニターが開始させるアプリケーションを識別する文字ストリング。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの **TriggerData** 属性を使用して、このフィールドに値を入れます。 IBM MQ for z/OS 以外の IBM MQ 製品では、このフィールドを使用して、トリガーされるチャネルの名前を指定できます。

ApplType
トリガー・モニターが開始させるアプリケーションのタイプ。 キュー・マネージャーがトリガー・メッセージを作成するとき、 *ProcessName* で識別されるプロセス定義オブジェクトの **ApplType** 属性を使用して、このフィールドに値を入れます。

ApplId
トリガー・モニターが開始されるアプリケーションを識別する文字ストリング。 キュー・マネージャーがトリガー・メッセージを作成するとき、*ProcessName* で識別されるプロセス定義オブジェクトの **ApplId** 属性を使用して、このフィールドに値を入れます。

CICS によって提供されるトリガー・モニター CKTI を使用する場合、プロセス定義オブジェクトの **ApplId** 属性は、CICS トランザクション ID です。

IBM MQ for z/OS によって提供される CSQQTRMN を使用する場合、プロセス定義オブジェクトの **ApplId** 属性は、IMS トランザクション ID です。

EnvData
トリガー・モニターが使用する環境関連データを含む文字フィールド。 キュー・マネージャーがトリガー・メッセージを作成するとき、*ProcessName* で識別されるプロセス定義オブジェクトの **EnvData** 属性を使用して、このフィールドに値を入れます。 CICS が提供したトリガー・モニター (CKTI) または IBM MQ for z/OS が提供したトリガー・モニター (CSQQTRMN) は、このフィールドを使用しませんが、他のトリガー・モニターは、このフィールドの使用を選択する場合があります。

UserData
トリガー・モニターが使用するユーザー・データを含む文字フィールド。 キュー・マネージャーがトリガー・メッセージを作成するとき、 *ProcessName* で識別されるプロセス定義オブジェクトの **UserData** 属性を使用して、このフィールドに値を入れます。 このフィールドは、起動されるチャネルの名前を指定するために使用できます。

トリガー・メッセージ構造体の詳細については、MQTM を参照してください。

トリガー操作が動作しないとき

トリガー・モニターがプログラムを開始できないか、またはキュー・マネージャーがトリガー・メッセージを送達できない場合は、プログラムは起動されません。 例えば、プロセス・オブジェクトのアプリケーション ID はプログラムがバックグラウンドで開始されるように指定する必要がありますが、これが指定されていない場合、トリガー・モニターはプログラムを開始できません。

トリガー・メッセージが作成されても開始キューに書き込むことができない (例えば、キューが満ぱいであるか、トリガー・メッセージの長さがその開始キューに指定されているメッセージの最大長を超えているため) 場合、そのトリガー・メッセージは代わりに送達不能 (未配布メッセージ) キューに書き込まれます。

送達不能キューへの PUT 操作を正常に完了できない場合は、トリガー・メッセージは廃棄され、警告メッセージが 

z/OS コンソールまたはシステム・オペレーターに送られるか、エラー・ログに書き込まれます。

トリガー・メッセージを送達不能キューに書き込むと、そのキューにトリガー・メッセージが生成される場合があります。 この 2 番目のトリガー・メッセージは、メッセージを送達不能キューに追加すると、破棄されます。

プログラムが正常に起動された場合でも、プログラムがキューからメッセージを受け取る前に異常終了した場合は、トレース・ユーティリティー (例えば、プログラムが CICS で動作している場合には CICS AUXTRACE) を使用して、障害の原因を調べてください。

MQI とクラスターの処理

呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

クラスターとともに使用できる呼び出しや戻りコードで使用可能なオプションの詳細については、 以下のリンクを参照してください。

- 889 ページの『MQOPEN およびクラスター』
- 890 ページの『MQPUT、MQPUT1、およびクラスター』
- 891 ページの『MQINQ およびクラスター』
- 892 ページの『MQSET およびクラスター』
- 892 ページの『戻りコード』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』

ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

MQOPEN およびクラスター

メッセージの書き込み先、または読み取り元のキューは、クラスター・キューがオープンされるときには、MQOPEN 呼び出しに依存します。

ターゲット・キューの選択

オブジェクト記述子 MQOD にキュー・マネージャー名を指定しない場合、キュー・マネージャーは、メッセージ送信先のキュー・マネージャーを選択します。 オブジェクト記述子にキュー・マネージャー名を指定した場合、メッセージは常に、選択したキュー・マネージャーに送信されます。

キュー・マネージャーがターゲット・キュー・マネージャーを選択している場合、その選択は、バインディング・オプション MQOO_BIND_* と、ローカル・キューが存在するかどうかによって異なります。 キューのローカル・インスタンスがある場合は、CLWLUSEQ 属性が ANY に設定されている場合を除き、常にリモート・インスタンスよりも先にオープンされます。 それ以外の場合、選択はバインディング・オプションによって異なります。グループ内のすべてのメッセージが同じ宛先で処理されるように、クラスターでメッセージ・グループを使用する場合は、 MQOO_BIND_ON_OPEN または MQOO_BIND_ON_GROUP のいずれかを指定する必要があります。

キュー・マネージャーは、ターゲット・キュー・マネージャーを選択する場合に、ワークロード管理アルゴリズムを使用して「ラウンドロビン」方式で選択します。 クラスターでのワークロード・バランシングを参照してください。

ワークロード・バランシング・アルゴリズムが使用されるタイミングは、次のようにクラスター・キューを開く方法によって異なります。

- MQOO_BIND_ON_OPEN - アルゴリズムは、アプリケーションがキューを開いた時点で 1 回使用されます。
- MQOO_BIND_NOT_FIXED - アルゴリズムは、それぞれのメッセージがキューに書き込まれるたびに使用されます。
- MQOO_BIND_ON_GROUP - アルゴリズムは、各メッセージ・グループの開始時点で 1 回ずつ使用されます。

MQOO_BIND_ON_OPEN

MQOPEN 呼び出しの MQOO_BIND_ON_OPEN オプションは、ターゲット・キュー・マネージャーを固定することを指定します。 クラスター内に同じキューのインスタンスが複数ある場合は、MQOO_BIND_ON_OPEN オプションを使用します。 MQOPEN 呼び出しから返されるオブジェクト・ハンドルを指定する、キューに入るすべてのメッセージは、同じキュー・マネージャーに送られます。

- メッセージに類縁性がある場合は、MQOO_BIND_ON_OPEN オプションを使用します。 例えば、メッセージのバッチをすべて同じキュー・マネージャーで処理する場合は、キューを開くときに MQOO_BIND_ON_OPEN を指定します。 IBM MQ は、そのキューに入ったすべてのメッセージで実行対象となるキュー・マネージャーおよびルートを固定します。
- MQOO_BIND_ON_OPEN オプションを指定した場合は、選択の対象となるキューの新しいインスタンス用にそのキューを再オープンする必要があります。

MQOO_BIND_NOT_FIXED
MQOPEN 呼び出しの MQOO_BIND_NOT_FIXED オプションは、ターゲット・キュー・マネージャーを固定しないことを指定します。 MQOPEN 呼び出しから返されるオブジェクト・ハンドルを指定する、キューに書き込まれるメッセージは、MQPUT の実行時にメッセージごとにキュー・マネージャーへの経路が指定されます。 すべてのメッセージを強制的に同じ宛先に書き込むことはしない場合は、MQOO_BIND_NOT_FIXED オプションを使用します。

- MQOO_BIND_NOT_FIXED と MQMF_SEGMENTATION_ALLOWED は、同時には指定しないでください。 そのようにすると、メッセージのセグメントが異なるキュー・マネージャーに送達され、クラスター全体に分散される可能性があります。

MQOO_BIND_ON_GROUP
アプリケーションが、メッセージのグループが同じ宛先インスタンスに割り当てられるように要求できるようにします。 このオプションは、キューの場合にのみ有効であり、クラスター・キューにのみ影響します。 クラスター・キューではないキューに対して指定された場合、このオプションは無視されます。

- MQPUT で MQPMO_LOGICAL_ORDER が指定されている場合、グループは、1 つの宛先にのみ経路指定されます。 MQOO_BIND_ON_GROUP が指定されても、メッセージが特定の論理グループに属していなければ、BIND_NOT_FIXED の動作が代わりに使用されます。

MQOO_BIND_AS_Q_DEF
MQOO_BIND_ON_OPEN、MQOO_BIND_NOT_FIXED、および MQOO_BIND_ON_GROUP のいずれも指定しない場合、デフォルト・オプションは MQOO_BIND_AS_Q_DEF です。 MQOO_BIND_AS_Q_DEF を使用すると、キュー・ハンドルに使用するバインディングは、DefBind キュー属性から取られます。

MQOPEN オプションの関連性

MQOPEN が成功するには、 MQOPEN オプション MQOO_BROWSE, MQOO_INPUT_*または MQOO_SET にクラスター・キューのローカル・インスタンスが必要です。

MQOPEN のオプションである MQOO_OUTPUT、MQOO_BIND_*、および MQOO_INQUIRE は、成功させるためにクラスター・キューのローカル・インスタンスを必要としません。

解決されたキュー・マネージャー名

MQOPEN の実行時にキュー・マネージャー名が解決されると、解決された名前がアプリケーションに返されます。 アプリケーションが以降の MQOPEN 呼び出しでこの名前の使用を試行する場合、アプリケーションがその名前へのアクセスを許可されていないことが分かる場合があります。

MQPUT、*MQPUT1*、およびクラスター

MQOPEN で MQOO_BIND_NOT_FIXED が指定された場合、ワークロード管理ルーチンは、MQPUT または MQPUT1 のどちらの宛先を選択するか決めます。

MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED が指定された場合、以降の MQPUT 呼び出しはそれぞれワークロード管理ルーチンを呼び出して、メッセージ送信先のキュー・マネージャーを決定できます。 宛先と経路はメッセージごとに選択されます。 メッセージが入った後でネットワークの状態が変わった場合は、宛先と経路が変わることがあります。 MQPUT1 呼び出しは常に、MQOO_BIND_NOT_FIXED が有効であると見なして動作します。 つまり、常にワークロード管理ルーチンを呼び出します。

ワークロード管理ルーチンがキュー・マネージャーを選択すると、ローカル・キュー・マネージャーは PUT 操作を完了します。 メッセージは異なる複数のキューに入れることができます。

1. 宛先がキューのローカル・インスタンスである場合、メッセージはローカル・キューに入ります。
2. 宛先がクラスター内のキュー・マネージャーである場合、メッセージはクラスター伝送キューに入ります。
3. 宛先がクラスター外部のキュー・マネージャーである場合、メッセージは、ターゲット・キュー・マネージャーと同じ名前を持つ伝送キューに入ります。

MQOPEN 呼び出しで MQOO_BIND_ON_OPEN が指定された場合、MQPUT 呼び出しは、宛先および経路が既に選択されているため、ワークロード管理ルーチンを呼び出しません。

MQINQ およびクラスター

どのクラスター・キューが照会の対象となるかは、MQOO_INQUIRE と組み合わせるオプションによって決まります。

キューに対して照会する前に、MQOPEN 呼び出しを使用してキューを開き、MQOO_INQUIRE を指定します。

クラスター・キューに対して照会を実行するには、MQOPEN 呼び出しを使用し、他のオプションと MQOO_INQUIRE を組み合わせます。 照会できる属性は、クラスター・キューのローカル・インスタンスがあるかどうか、およびキューがどのようにオープンしているかによって決まります。

- MQOO_BROWSE、MQOO_INPUT_*、または MQOO_SET を MQOO_INQUIRE と組み合わせる場合、オープンを成功させるには、クラスター・キューのローカル・インスタンスが必要です。 ローカル・インスタンスがあれば、ローカル・キューに有効なすべての属性を照会することができます。
- MQOO_OUTPUT を MQOO_INQUIRE と組み合わせて、 前述のオプションを他に何も指定しないと、オープンするインスタンスは次のいずれかになります。
  - ローカル・キュー・マネージャー上にあるインスタンス (もしある場合)。 ローカル・インスタンスがあれば、ローカル・キューに有効なすべての属性を照会することができます。
  - ローカル・キュー・マネージャー・インスタンスがない場合は、クラスター内の他の場所にあるインスタンス。 この場合は、次の属性のみ照会できます。 このときの QType 属性の値は MQQT_CLUSTER です。

    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

クラスター・キューの DefBind 属性を照会するには、MQINQ 呼び出しにセレクター MQIA_DEF_BIND を指定して使用します。 返される値は、MQBND_BIND_ON_OPEN または MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP のいずれかです。クラスターでグループを使用する場合は、MQBND_BIND_ON_OPEN または MQBND_BIND_ON_GROUP のいずれかを指定する必要があります。

キューのローカル・インスタンスの CLUSTER および CLUSNL 属性に対して照会を実行するには、MQINQ 呼び出しにセレクター MQCA_CLUSTER_NAME またはセレクター MQCA_CLUSTER_NAMELIST を指定して使用します。

注 : MQOPEN がバインドするキューを固定せずにクラスター・キューをオープンすると、MQINQ 呼び出しが連続して出されて、クラスター・キューのさまざまなインスタンスに対して照会が実行される可能性があります。

関連概念

748 ページの『クラスター・キュー用の MQOPEN オプション』

キュー・ハンドルに使用されるバインディングは **DefBind** キュー属性から取得され、値は MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP を取ることができます。

MQSET およびクラスター

MQOPEN オプションの MQOO_SET オプションには、クラスター・キューのローカル・インスタンスが必要です。これがなければ MQSET は成功しません。

MQSET 呼び出しを使用して、クラスター内の別の場所にあるキューの属性を設定することはできません。

クラスター属性で定義されたローカルの別名またはリモート・キューを開き、MQSET 呼び出しを使用することができます。 ローカルの別名またはリモート・キューの属性は、設定できます。 ターゲット・キューが別のキュー・マネージャーで定義されたクラスター・キューの場合でも問題になりません。

戻りコード

クラスター固有の戻りコード

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューのオープン、またはそこへのメッセージの書き込みのために発行されます。 キュー・マネージャーの ClusterWorkloadExit 属性によって定義されるクラスター・ワークロード出口は、予期せず失敗したり、時間内に応答しなかったりします。

IBM MQ for z/OS 上のシステム・ログにメッセージが書き込まれ、このエラーに関する詳細情報が提供されます。

このキュー・ハンドルに対する今後の MQOPEN、MQPUT、および MQPUT1 呼び出しは、ClusterWorkloadExit 属性がブランクであるものとして処理されます。

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

z/OS では、クラスター・ワークロード出口をロードできません。

システム・ログにメッセージが書き込まれ、ClusterWorkloadExit 属性はブランクと見なされて処理が続行されます。

Multi マルチプラットフォームでは、キュー・マネージャーに接続するために、MQCONN 呼び出しまたは MQCONNX 呼び出しが発行されます。 キュー・マネージャーの ClusterWorkloadExit 属性によって定義されるクラスター・ワークロード出口はロードすることができないため、呼び出しに失敗します。

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

有効な MQOO_OUTPUT オプションおよび MQOO_BIND_ON_OPEN オプションを指定した MQOPEN 呼び出しは、クラスター・キューに対して発行されます。 クラスター内のキューのインスタンスはすべて、InhibitPut 属性を MQQA_PUT_INHIBITED に設定することにより、現行では書き込みが禁止されています。 メッセージを受信するために使用可能なキュー・インスタンスがないため、MQOPEN 呼び出しに失敗します。

この理由コードは、次の記述が両方とも該当する場合にのみ戻されます。

- キューのローカル・インスタンスがない。 ローカル・インスタンスがあれば、 そのローカル・インスタンスが書き込み禁止になっていても MQOPEN 呼び出しは成功する。
- キューのクラスター・ワークロード出口がないか、またはクラスター・ワークロード出口はあるが、キュー・インスタンスを選択していない。 (クラスター・ワークロード出口がキュー・インスタンスを選択する場合には、たとえそのインスタンスが書き込みを禁止されていても、MQOPEN 呼び出しが成功します。)

MQOO_BIND_NOT_FIXED オプションが MQOPEN 呼び出しで指定されている場合、クラスター内のすべてのキューが書き込み禁止になっている場合であっても、呼び出しが成功することがあります。 ただし、後続の MQPUT 呼び出しは、この呼び出しの時点ですべてのキューが引き続き書き込み禁止になっている場合には、失敗する可能性があります。

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューのオープン、またはそこへのメッセージの書き込みのために発行されます。 フル・リポジトリー・キュー・マネージャーからの応答が必要であるのに応答がないため、キューの定義を正しく解決できません。
2. MQOPEN、 MQPUT、 MQPUT1 または MQSUB 呼び出しは、 PUBSCOPE (ALL) または SUBSCOPE (ALL) を指定したトピック・オブジェクトに対して発行されます。 完全リポジトリー・キュー・マネージャーからの応答が必要ですが、使用可能な応答がないため、クラスター・トピック定義を正しく解決できません。

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューに対して発行されます。 クラスタリングに必要なリソースを使用しようとしてエラーが発生しました。

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

MQPUT 呼び出しまたは MQPUT1 呼び出しは、クラスター・キューにメッセージを書き込むために発行されます。 呼び出し時に、クラスター内にキューのインスタンスはまったく存在しません。 MQPUT は失敗し、メッセージは送信されません。

キューをオープンする MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED が指定された場合、または MQPUT1 を使用してメッセージを書き込んだ場合に、このエラーが発生する可能性があります。

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、メッセージをオープンするか、クラスター・キューにメッセージを書き込むために発行されます。 クラスター・ワークロード出口は呼び出しを拒否します。

z/OS IBM MQ for z/OS 上でのアプリケーションの使用/作成方法

IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

ここでは、サポートされる各環境で稼働するプログラムに使用できる IBM MQ 機能について説明しています。 加えて、他の情報については以下のとおりです。

- IBM MQ-CICS bridge の使用については、「IBM MQ CICS と併用する方法」 を参照してください。
- IMS と IMS ブリッジの使用については、70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』を参照してください。

IBM MQ for z/OS 上でのアプリケーションの使用/作成方法について詳しくは、以下のリンクを参照してください。

- 894 ページの『環境依存の IBM MQ for z/OS 機能』
- 895 ページの『デバッグ機能、同期点サポート、およびリカバリー・サポート』
- 895 ページの『アプリケーション環境との IBM MQ for z/OS インターフェース』
- 897 ページの『z/OS UNIX System Services アプリケーションの作成』
- 900 ページの『共用キューを使用したアプリケーション・プログラミング』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』

ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

70 ページの『IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション』
この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

z/OS 環境依存の *IBM MQ for z/OS* 機能

IBM MQ for z/OS 機能については、この情報を参照してください。

IBM MQ for z/OS が実行される環境における IBM MQ 機能間の主な相違点は、以下のとおりです。

- IBM MQ for z/OS は、以下のトリガー・モニターを提供します。
  - CICS 環境で使用する CKTI
  - IMS 環境で使用する CSQQTRMN

  アプリケーションを上記以外の環境で開始するためには、独自のモジュールを作成する必要があります。
- 2 フェーズ・コミットを使用する同期点機能は、CICS および IMS 環境でサポートされます。 この機能は、トランザクション管理機能とリカバリー可能リソース管理サービス (RRS: Recoverable Resource Manager Service) を使用する z/OS バッチ環境でも使用できます。 また z/OS 環境では、IBM MQ 自体が単一フェーズ・コミットをサポートします。
- バッチおよび IMS 環境では、MQI はキュー・マネージャーにプログラムを接続したり、キュー・マネージャーからプログラムを切り離したり呼び出しを提供します。 プログラムは複数のキュー・マネージャーに接続できます。
- CICS システムは 1 つのキュー・マネージャーにだけ接続できます。 CICS システム開始ジョブでサブシステム名が定義される場合、CICS の開始時にこの処理を行うことができます。 CICS の環境で MQI の接続および切断機能を呼び出すことは可能ですが、効果はありません。
- API 交差出口を使用すると、プログラムはすべての MQI 呼び出しの処理に割り込むことができます。 この出口は CICS 環境でのみ使用可能です。
- マルチプロセッサー・システム上の CICS では、MQI 呼び出しが複数の z/OS TCB のもとで実行可能であるため、パフォーマンスが多少向上します。 詳しくは、「z/OS での計画 *IBM MQ for z/OS* 概説および計画ガイド」を参照してください。

これらの機能は、894 ページの表 130 に要約されています。

表 *130. z/OS* 環境の機能

	CICS	**IMS**	**Batch/TSO**
提供されるトリガー・モニター	はい	はい	いいえ
2 フェーズ・コミット	はい	はい	はい
単一フェーズ・コミット	はい	いいえ	はい

表 130. z/OS 環境の機能 (続き)			
	CICS	**IMS**	**Batch/TSO**
接続 / 切断 MQI 呼び出し	許容	はい	はい
API 交差出口 (API-crossing exit)	はい	いいえ	いいえ

注 : RRS を使用する Batch/TSO 環境では、2 フェーズ・コミットがサポートされています。

z/OS デバッグ機能、同期点サポート、およびリカバリー・サポート

この情報を使用して、プログラム・デバッグ機能、同期点サポート、およびリカバリー・サポートについて学習します。

プログラム・デバッグ機能

IBM MQ for z/OS は、すべての環境でプログラムをデバッグするために使用できるトレース機能を提供します。

さらに、CICS 環境では以下の機能を使用できます。

- CICS 実行診断機能 (CEDF)
- CICS トレース管理トランザクション (CETR)
- IBM MQ for z/OS API 交差出口

z/OS プラットフォームでは、ご使用のプログラミング言語によってサポートされる対話式デバッグ・ツールがすべて使用できます。

同期点のサポート

トランザクション処理環境では、トランザクションの処理が安全に行われるように、作業単位の開始と終了の同期化が必要です。

これは、 CICS 環境および IMS 環境の IBM MQ for z/OS によって完全にサポートされます。 完全サポートとは、CICS または IMS の制御下で、作業単位が一致してコミットされたり、バックアウトされるように、リソース管理プログラム間で連携が行われることを意味します。 リソース管理プログラムの例としては、Db2、CICS ファイル制御、IMS、および IBM MQ for z/OS があります。

z/OS バッチ・アプリケーションは、IBM MQ for z/OS 呼び出しを使用して単一フェーズ・コミット機能を与えることができます。 これは、他のリソース管理プログラムを参照しなくても、アプリケーション定義の一連のキュー操作をコミットまたはバックアウトすることができることを意味します。

トランザクション管理機能やリカバリー可能リソース管理サービス (RRS) を使用する z/OS バッチ環境では、2 フェーズ・コミット機能も使用できます。 詳細については、z/OS バッチ・アプリケーションにおける同期点を参照してください。

リカバリー・サポート

キュー・マネージャーと CICS または IMS システム間の接続が、トランザクションの間に切断された場合は、いくつかの作業単位は正しくバックアウトできない可能性があります。

ただし、これらの作業単位は、キュー・マネージャーと CICS または IMS システムとの接続が再確立されたとき、キュー・マネージャーによって (同期管理プログラムの制御下で) 解決されます。

z/OS アプリケーション環境との *IBM MQ for z/OS* インターフェース

異なる環境で実行されているアプリケーションがメッセージ・キューイング・ネットワークを介してメッセージを送受信できるようにするために、IBM MQ for z/OS は、サポートする各環境に対して アダプターを提供します。

これらのアダプターは、アプリケーション・プログラムと IBM MQ for z/OS サブシステムの間のインターフェースです。 これらのアダプターによってプログラムは MQI を使用できます。

z/OS バッチ・アダプター

この情報を使用して、バッチ・アダプターとそれがサポートするコミット・プロトコルについて学習します。

バッチ・アダプター は、以下の状態で稼働中のプログラムの IBM MQ for z/OS リソースへのアクセスを提供します。

- タスク (TCB) モード
- 障害状態または監視プログラム状態
- 1 次アドレス・スペース制御モード

プログラムを仮想記憶間モードにしてはなりません。

アプリケーション・プログラムと IBM MQ for z/OS との接続は、タスク・レベルで行われます。 アダプターは、アプリケーション・タスク制御ブロック (TCB) から IBM MQ for z/OS への単一接続スレッドを提供します。

このアダプターは、IBM MQ for z/OS が所有するリソースに対して行われた変更について単一フェーズ・コミット・プロトコルはサポートしますが、複数フェーズ・コミット・プロトコルはサポートしません。

z/OS *RRS* バッチ・アダプター

この情報を使用して、RRS バッチ・アダプターについて、および IBM MQ で提供される 2 つの RRS バッチ・アダプターについて学びます。

トランザクション管理やリカバリー可能リソース管理サービス (RRS) におけるバッチ・アダプターの考慮事項は、次のとおりです。

- コミット制御には z/OS RRS を使用する。
- 単一のタスクから、単一の z/OS インスタンス上で実行される複数の IBM MQ サブシステムへの同時接続をサポートします。
- 次のソフトウェアを考慮して、z/OS RRS に準拠したリカバリー可能マネージャーがリカバリー可能リソースにアクセスできるように、z/OS 全体の (z/OS RRS を使用した) コミットメント制御を行う。
  - RRS バッチ・アダプターを使って IBM MQ に接続するアプリケーション。
  - z/OS 上のワークロード・マネージャー (WLM) によって管理される Db2 ストアード・プロシージャー・アドレス・スペースで実行される Db2 ストアード・プロシージャー。
- 複数の TCB 間で IBM MQ バッチ・スレッドが交換接続できるようにする。

IBM MQ for z/OS には RRS バッチ・アダプターが 2 つあります。

CSQBRSTB
このアダプターを使用するには、IBM MQ アプリケーションのすべての MQCMIT ステートメントを SRRCMIT に、すべての MQBACK ステートメントを SRRBACK に変更する必要があります。 (CSQBRSTB とリンクするアプリケーションで MQCMIT や MQBACK を使うと、MQRC_ENVIRONMENT_ERROR を受け取ります。)

CSQBRRSI
このアダプターでは、IBM MQ アプリケーションで MQCMIT と MQBACK、あるいは SRRCMIT と SRRBACK の組み合わせのどちらかを使用することができます。

注 : CSQBRSTB および CSQBRRSI には、デフォルトの連係属性 AMODE(31) RMODE(ANY) が割り当てられています。 回線速度 16 MB 未満でどちらかのスタブをロードするアプリケーションは、まず、RMODE(24) でスタブを再リンクする必要があります。

マイグレーション

若干の変更で、または一切変更せずに、RRS 調整を使用できるように、既存の Batch/TSO IBM MQ アプリケーションをマイグレーションできます。

IBM MQ アプリケーションを CSQBRRSI アダプターでリンク・エディットすれば、IBM MQ に加え、RRS をサポートする他のリソース管理プログラムも含めた全環境で、MQCMIT および MQBACK は作業単位を同期点処理します。 IBM MQ アプリケーションを CSQBRSTB アダプターでリンク・エディットする場合は、

MQCMIT を SRRCMIT に、MQBACK を SRRBACK に変更します。 後者のアプローチには、同期点が IBM MQ リソースだけに限定されていないことが明示されるという利点があります。

z/OS *IMS アダプター*

IMS アダプターを IBM MQ for z/OS システムから使用している場合は、最大 100 MB の長さのメッセージに対応するための十分なストレージを IMS が取得できることを確認してください。

ユーザーへの注

IMS アダプター は、以下の IBM MQ for z/OS リソースへのアクセスを提供します。

- オンライン・メッセージ処理プログラム (MPP)
- 対話式ファースト・パス・プログラム (IFP)
- バッチ・メッセージ処理プログラム (BMP)

これらのリソースを使用するには、プログラムは、タスク (TCB) モードおよび問題プログラム状態で稼働していることが必要です。仮想記憶間モードまたはアクセス・レジスター・モードであってはなりません。

このアダプターには、アプリケーション・タスク制御ブロック (TCB) から IBM MQ への接続スレッドが備えられています。 アダプターは、IBM MQ for z/OS が所有するリソースに加えられた変更に対する 2 フェーズ・コミット・プロトコルをサポートします。このとき、同期点調整プログラムとして機能する IMS を使用します。

アダプターはまた、キュー上のあるトリガー条件が満たされたときにプログラムを自動的に開始できるトリガー・モニター・プログラムを提供します。 詳細については、869 ページの『トリガーによる IBM MQ アプリケーションの開始』を参照してください。

バッチ DL/I プログラムを作成する場合は、このトピックの z/OS バッチ・プログラムに関する説明に従ってください。

z/OS *z/OS UNIX System Services アプリケーションの作成*

バッチ・アダプターは、バッチおよび TSO アドレス・スペースからのキュー・マネージャー接続をサポートします。

バッチ・アドレス・スペースの場合、アダプターは、以下のように、そのアドレス・スペース内の複数の TCB からの接続をサポートします。

- MQCONN または MQCONNX 呼び出しにより、各 TCB から複数のキュー・マネージャーへ接続できます (ただし、TCB が一度に処理できるのは、ある特定のキュー・マネージャーへ接続する 1 つのインスタンスのみです)。
- 複数の TCB から 1 つのキュー・マネージャーへ接続することもできます (ただし、MQCONN または MQCONNX 呼び出しにより戻るキュー・マネージャー・ハンドルは発行元の TCB にバインドされ、他の TCB では使用できません)。

z/OS UNIX System Services では、次の 2 種類の pthread_create 呼び出しをサポートしています。

1. ヘビーウェイト・スレッド。このスレッドは TCB ごとに 1 つずつ実行されます。TCB は z/OS によってスレッドの開始時に接続 (ATTACH) され、終了時に切り離されます (DETACH)。
2. ミディアムウェイト・スレッド。このスレッドは TCB ごとに 1 つずつ実行されますが、長時間稼働している TCB のプールの 1 つでも構いません。 必要なアプリケーション・クリーンアップは、すべてアプリケーションで実行する必要があります。これは、アプリケーションがサーバーに接続していると、タスク (TCB) 終了時にサーバーが実行するデフォルトのスレッド終了処理が必ずしも発生するとは**限らない** からです。

ライトウェイト・スレッドはサポートしていません。 (アプリケーションで自身の作業要求をディスパッチする永続スレッドを作成する場合は、その **アプリケーション** で、次の作業要求を開始する前にリソースのクリーンアップを実行する必要があります。)

IBM MQ for z/OS は、バッチ・アダプターを使用して z/OS UNIX System Services スレッドを以下のようにサポートします。

1. ヘビーウェイト・スレッドについては、バッチ接続として完全サポートしています。 各スレッドは専用の TCB で実行します。この TCB はスレッドの開始時に接続され、終了時に切り離されます。 スレッド終了後、MQDISC 呼び出しを発行してください。これにより、IBM MQ for z/OS は標準タスク・クリーンアップを実行します。この処理では、未処理の作業単位のコミット (スレッドが正常終了した場合) またはバックアウト (スレッドが異常終了した場合) などを実行します。
2. ミディアムウェイト・スレッドについては完全にサポートしていますが、TCB を別のスレッドで再使用する場合、アプリケーションでは MQCMIT または MQBACK のあとに MQDISC 呼び出しを必ず発行してから次のスレッドを開始するようにしてください。 つまり、アプリケーションがプログラム割り込みハンドラーを設定してからアプリケーションが異常終了した場合は、TCB を別のスレッドで再使用する前に、割り込みハンドラーが MQCMIT および MQDISC 呼び出しを発行する必要があります。

注 : スレッド・モデルは、複数のスレッドからの共通 IBM MQ リソースへのアクセスをサポート **しません** 。

z/OS *z/OS での API 交差出口*

このトピックでは、プロダクト・センシティブ・プログラミング・インターフェースに関する情報を提供します。

出口とは、IBM 提供のコード内において、プログラマーが各自のコードを実行できる場所のことです。 IBM MQ for z/OS には、*API* 交差出口 があり、これを使用して MQI に対する呼び出しを代行受信したり、MQI 呼び出しの機能をモニターまたは修正したりすることができます。 この節では、API 交差出口の使用法と、IBM MQ for z/OS に付属して提供されるサンプル出口プログラムについて説明します。

このセクションは、CICS TS V3.1 以前のユーザーにのみ適用されます。 CICS TS V3.2 以降のユーザーは、CICS 製品資料の「 CICS IBM MQ との統合」セクションを参照してください。

注記

API 交差出口は、IBM MQ for z/OS の CICS アダプターによってのみ呼び出されます。 この出口プログラムは CICS アドレス・スペースで稼働します。

z/OS ユーザー固有の出口プログラムの作成

IBM MQ for z/OS で提供されているサンプル API 交差出口プログラム (CSQCAPX) を枠組みとして使用し、固有の出口プログラムを作成することができます。

これについては、899 ページの『サンプル API 交差出口プログラム、CSQCAPX』で説明されています。

出口プログラムを作成する際に、アプリケーションによって発行される MQI 呼び出しの名前を検索するには、MQXP 構造体の *ExitCommand* フィールドを調べます。 その呼び出しのパラメーターの数を検索するには、*ExitParmCount* フィールドを調べます。 16 バイトの *ExitUserArea* フィールドを使用して、アプリケーションが取得するすべての動的ストレージのアドレスを格納できます。 このフィールドは出口を何度呼び出しても変更されません。存続期間は CICS タスクと同じです。

CICS Transaction Server V3.2 を使用している場合は、出口プログラムをスレッド・セーフとして作成し、スレッド・セーフとして宣言する必要があります。 以前のリリースの CICS を使用している場合も、CICS Transaction Server V3.2 へのマイグレーションに備えて、出口プログラムをスレッド・セーフとして作成および宣言することをお勧めします。

ユーザーの出口プログラムは、MQXCC_SUPPRESS_FUNCTION または MQXCC_SKIP_FUNCTION を *ExitResponse* フィールドに戻すことによって、MQI 呼び出しの実行を制御できます。 その呼び出しを実行できるように許可するには (また、その呼び出しが完了したあとで出口プログラムを再び呼び出すには)、ユーザーの出口プログラムが MQXCC_OK を戻す必要があります。

出口プログラムは、MQI 呼び出しのあとに呼び出すと、その呼び出しによって設定された完了コードと理由コードを調べて、修正できます。

使用上の注意

ここでは、出口プログラムを作成する際に考慮する必要のあるいくつかの一般的な点について説明します。

- パフォーマンス上の理由から、アセンブラー言語でプログラムを作成してください。 IBM MQ for z/OS でサポートしている他の言語のうちのいずれかでプログラムを作成する場合には、自分でデータ定義ファイルを作成する必要があります。
- プログラムを AMODE(31) および RMODE(ANY) としてリンク・エディットします。
- 作成するプログラムに対して出口パラメーター・ブロックを定義するには、アセンブラー言語のマクロ CMQXPA を使用します。
- 出口プログラムや、出口プログラムから呼び出すプログラムを定義する際には、CONCURRENCY(THREADSAFE) を指定してください。
- CICS Transaction Server for z/OS ストレージ保護機構を使用している場合、プログラムを CICS 実行キーで実行する必要があります。 つまり、出口プログラムと制御を渡す先のプログラムの両方を定義するときには、EXECKEY(CICS) を指定する必要があります。 CICS 出口プログラムと CICS 記憶保護機能については、「*CICS* カストマイズの手引き」を参照してください。
- プログラムは、すべての API (例えば、 IMS、 Db2、および CICS) を使用できます。 CICS タスク関連ユーザー出口プログラムで使用できます。 また、MQCONN、MQCONNX、および MQDISC を除くすべての MQI 呼び出しを使用できます。 しかし、出口プログラム内の MQI 呼び出しは、出口プログラムを一度しか呼び出しません。
- 作成するプログラムでは、EXEC CICS SYNCPOINT または EXEC CICS SYNCPOINT ROLLBACK コマンドを発行できます。 しかし、これらのコマンドは、そのタスクによって行われた **すべての** 更新内容を出口が使用された点までコミットまたはロールバックするので、これらのコマンドを使用することはお勧めできません。
- 作成するプログラムは、EXEC CICS RETURN コマンドを実行して終了しなければなりません。 XCTL コマンドで制御を転送してはなりません。
- 出口は、IBM MQ for z/OS コードに対する拡張機能として作成されます。 出口によって、MQI を使用する IBM MQ for z/OS プログラムやトランザクションに障害が発生することがないようにする必要があります。 これらは、通常、接頭部 CSQ または CK で示されます。
- CSQCAPX が CICS に定義されると、CICS システムは、CICS が IBM MQ for z/OS に接続するときに出口プログラムのロードを試みます。 ロードが正常に行われると、メッセージ CSQC301I が CKQC パネルまたはシステム・コンソールに送られます。 ロードが失敗すると (例えば、ロード・モジュールが DFHRPL 連結のどのライブラリーにも存在しない場合)、メッセージ CSQC315 が CKQC パネルまたはシステム・コンソールに送られます。
- 連絡域内のパラメーターがアドレスであるため、出口プログラムは、CICS システムに対してローカルとして (つまり、リモート・プログラムとしてではなく) 定義されなければなりません。

z/OS サンプル *API* 交差出口プログラム、*CSQCAPX*

サンプル出口プログラムはアセンブラー言語プログラムとして提供されます。 ソース・ファイル (CSQCAPX) はライブラリー **thlqual**.SCSQASMS で提供されます (**thlqual** はインストール時に使用する高水準の修飾子)。 このソース・ファイルには、プログラム論理を記述する疑似コードが入っています。

このサンプル・プログラムには、初期化コードと、独自の出口プログラムを作成する際に使用できるレイアウトが入っています。

このサンプルでは、以下の作業を行う方法を示しています。

- 出口パラメーター・ブロックのセットアップ
- 呼び出しと出口パラメーター・ブロックのアドレッシング
- どの MQI 呼び出しに対して出口が呼び出されているかの判別
- 出口の呼び出しが MQI 呼び出しの処理の前か後かの判別
- CICS 一時記憶キューへのメッセージの書き込み
- 動的ストレージ獲得用マクロ DFHEIENT の使用による、再入可能性の維持
- CICS exec インターフェース制御ブロックに対する DFHEIBLK の使用
- エラー条件のトラップ
- 呼び出し側への戻り制御

サンプル出口プログラムの設計

サンプル出口プログラムでは、CICS 一時記憶キュー (CSQ1EXIT) にメッセージを書き込み、出口の操作を示します。

メッセージは、出口の呼び出しが MQI 呼び出しの前か後かを示します。 出口が MQI 呼び出しの後に呼び出される場合、メッセージにはその呼び出しによって戻る完了コードと理由コードが入っています。 このサンプルでは、CMQXPA マクロから名前付きの定数を使用して、入り口のタイプ (つまり、その呼び出しの前か後) について検査します。

このサンプルでは、モニター機能は実行せず、処理中の呼び出しのタイプを示しながら単にタイム・スタンプ付きのメッセージを CICS キューに入れます。 これによって、出口プログラムの正しい働きと共に、MQI のパフォーマンスが示されます。

注 : サンプル出口プログラムは、プログラムの実行中に行われる MQI 呼び出しごとに 6 つの EXEC CICS 呼び出しを発行します。 この出口プログラムを使用すると、IBM MQ for z/OS のパフォーマンスは低下します。

z/OS *API* 交差出口の準備と使用

サンプル出口は、ソース形式でのみ提供されます。

サンプル出口または独自に作成した出口プログラムを使用するには、他の CICS プログラムの場合と同様に、ロード・ライブラリーを作成します (1032 ページの『z/OS での CICS アプリケーションの構築』を参照)。

- CICS Transaction Server for z/OS および CICS for MVS™/ESA の場合、CICS システム定義 (CSD) データ・セットを更新するときに必要な定義はメンバー **thlqual**.SCSQPROC(CSQ4B100) 内にあります。

  注 : この定義では、接尾部 MQ を使用します。 この接尾部が社内で既に使用されている場合は、アセンブリー段階以前に変更する必要があります。

提供されているデフォルトの CICS プログラム定義を使用する場合は、出口プログラム CSQCAPX は **使用不可** 状態でインストールされます。 これは、出口プログラムを使用すると、パフォーマンスが大幅に低下する可能性があるからです。

API 交差出口を一時的にアクティブ化するには、次のようにします。

1. CICS マスター端末からコマンド **CEMT S PROGRAM(CSQCAPX) ENABLED** を発行します。
2. CKQC トランザクションを実行し、接続プルダウンのオプション 3 を使用して API 交差出口の状態を「**使用可能**」に変更する。

API 交差出口を永続的に使用可能にして IBM MQ for z/OS を実行する場合は、CICS Transaction Server for z/OS および CICS for MVS/ESA で、以下のいずれかを行います。

- STATUS(DISABLED) を STATUS(ENABLED) に変更して、メンバー CSQ4B100 内の CSQCAPX 定義を変更します。 CICS 提供のバッチ・プログラム DFHCSDUP を使用して、CICS CSD 定義を更新できます。
- 状況を DISABLED から ENABLED に変更することによって、CSQCAT1 グループ内の CSQCAPX 定義を変更できます。

どちらの場合も、グループを再インストールする必要があります。 これを行うには、CICS システムをコールド・スタートするか、あるいは CICS CEDA トランザクションを使用して、CICS の稼働中にグループを再インストールします。

注 : CEDA を使用すると、グループ内のいずれかのエントリーが現在使用中の場合に、エラーが発生する可能性があります。

プロダクト・センシティブ・プログラミング・インターフェース情報はこれで終わりです。

z/OS 共用キューを使用したアプリケーション・プログラミング

このトピックでは、共用キューを使用する新しいアプリケーションを設計するとき、また既存のアプリケーションを共用キュー環境にマイグレーションするときに、考慮に入れる必要のあるいくつかの要因についての情報を提供しています。

z/OS アプリケーションの逐次化

特定のタイプのアプリケーションでは、メッセージがキューに到着したときとまったく同じ順序でキューから取り出されるようにする必要がある場合があります。

例えば、データベースに対する更新のシャドーをリモート・システムに生成するのに IBM MQ を使用している場合は、レコードに対する更新を説明するメッセージは、そのレコードの挿入について説明するメッセージの後で処理される必要があります。 ローカル・キュー環境では、これは多くの場合、キューをオープンしているメッセージを読み取るアプリケーションによって行われます。その際、他の読み取りアプリケーションが同時にそのキューを処理することがないように、MQOO_INPUT_EXCLUSIVE オプションが指定されます。

IBM MQ では、アプリケーションがこれと同じ方法で共用キューを排他的にオープンできます。 ただし、アプリケーションがキューの区画から作業を行っている (例えば、すべてのデータベースに対する更新は同一キュー上にあるものの、テーブル A に対する更新の相関 ID が A で、テーブル B に対する更新の相関 ID が B である) 場合で、アプリケーションがテーブル A に対する更新のメッセージとテーブル B に対する更新のメッセージを同時に読み取る場合は、キューを排他的にオープンするという単純な機構は使用できません。

このタイプのアプリケーションが共用キューの高可用性を利用する場合は、1 次読み取りアプリケーションまたはキュー・マネージャーに障害が起こった場合に、同じ共用キューにアクセスし、2 次キュー・マネージャー上で稼働しているアプリケーションの別のインスタンスが、これを引き継ぐようにすることができます。

1 次キュー・マネージャーに障害が起こる場合は、次の 2 つのことが生じます。

- 共用キュー対等リカバリーによって、1 次アプリケーションからの更新が不完全であれば、その更新が完了されるか、バックアウトされる。
- 2 次アプリケーションが、キューの処理を引き継ぐ。

すべての未完了の作業単位が処理される前に 2 次アプリケーションが開始する場合がありますが、その場合は、2 次アプリケーションが順不同でメッセージを取り出すようになります。 この種の問題を解決するために、アプリケーションを逐次化アプリケーション にすることができます。

逐次化アプリケーションは、MQCONNX 呼び出しを使用して、アプリケーションに固有の接続タグを接続時に指定して、キュー・マネージャーに接続します。 アプリケーションが実行するすべての作業単位は、接続タグを使用してマークされます。 IBM MQ は、キュー共用グループ内の、同じ接続タグが指定された作業単位が逐次化されるようにします (MQCONNX 呼び出しでの逐次化オプションに従って)。

つまり、1 次アプリケーションが接続タグ `Database shadow retriever` を指定した MQCONNX 呼び出しを使用し、2 次テークオーバー・アプリケーションが同じ接続タグを指定した MQCONNX 呼び出しを使用しようとすると、2 次アプリケーションは、未解決の 1 次作業単位が完了するまで (この場合はピア・リカバリー)、2 次 IBM MQ に接続できません。

メッセージをキュー上の順序どおりに使用するアプリケーションでは、逐次化アプリケーション技法を使用することを考慮してください。 特に、次の点に注意してください。

- アプリケーションまたはキュー・マネージャーに障害が起こった後、そのアプリケーションを実行したときのコミットおよびバックアウト操作がすべて完了するまで、再始動してはならないアプリケーション。

  この場合、逐次化アプリケーション技法を使用できるのは、アプリケーションが同期点で作業する場合だけです。
- 同一アプリケーションの別のインスタンスが既に実行されている間は開始してはならないアプリケーション。

  この場合、逐次化アプリケーション技法が必要なのは、アプリケーションが排他的入力用にキューをオープンできない場合だけです。

注 : IBM MQ は、特定の基準が満たされるときにメッセージの順序が保持されることを保証するにすぎません。 これについては、MQGET を参照してください。

z/OS 共用キューでの使用に向かないアプリケーション

共用キューを使用している場合、IBM MQ の一部の機能はサポートされないため、これらの機能を使用するアプリケーションは、共用キュー環境に適していません。

共用キュー・アプリケーションを設計するときは次の点を考慮してください。

- キューに索引を付けられるのは、共用キューだけです。 キューから取得するメッセージの選択にメッセージ ID または相関 ID を使用するためには、キューが正しい値で索引付けされている必要があります。 メッセージ ID のみを使用してメッセージを選択している場合は、キューの索引タイプを MQIT_MSG_ID にする必要があります (ただし、MQIT_NONE も使用できます)。 相関 ID のみを使用してメッセージを選択している場合は、キューの索引タイプを MQIT_CORREL_ID にする必要があります。
- 一時動的キューを共用キューとして使用することはできません。 ただし、永続動的キューが使用できます。 共用動的キューは、PERMDYN (永続動的) キューと同じ方法で作成および破棄されますが、共用動的キューのモデルの DEFTYPE は SHAREDYN (共用動的) です。

z/OS 非アプリケーション・キューを共用するかどうかの決定

非アプリケーション・キューの共用については、この情報を参照してください。

アプリケーション・キュー以外のキューについても、共用することを考慮できます。

開始キュー

共用開始キューを定義する場合は、少なくとも 1 つのトリガー・モニターが実行されている場合は、キュー共用グループ内のすべてのキュー・マネージャー上でトリガー・モニターを実行する必要はありません。 (キュー共用グループ内の各キュー・マネージャー上でトリガー・モニターが実行している場合でも、共用開始キューを使用できます。)

共用アプリケーション・キューがあり、EVERY というトリガー・タイプ (またはトリガー間隔が短く、EVERY というトリガー・タイプと同じように動作する FIRST というトリガー・タイプ) を使用する場合は、開始キューは常に共用キューでなければなりません。 共用開始キューをどの場合に使用したらよいかについては、903 ページの表 131 を参照してください。

システム.* キュー

SYSTEM.ADMIN.* イベント・メッセージを共用キューとして保持するために使用されるキュー。 これは、例外が発生した場合にロード・バランシングを検査するのに便利です。 IBM MQ によって作成される各イベント・メッセージには、どのキュー・マネージャーによって作成されたかを示す相関 ID があります。

SYSTEM.QSG。* 共用チャネルに使用されるキュー、および共用キューとしてのグループ内キューイング。

SYSTEM.DEFAULT.LOCAL.QUEUE の定義を共用するように変更することもできますし、独自のデフォルト共用キュー定義を定義することもできます。 詳しくは、IBM MQ for z/OS のシステム・オブジェクトの定義 を参照してください。

他の SYSTEM.* を定義することはできません。 共用キューとしてのキュー。

z/OS 共用キューを使用するよう既存のアプリケーションをマイグレーションする

共用キュー環境では、理由コード、トリガー操作、および MQINQ API 呼び出しの動作が異なる場合があります。

既存のキューを共用キューにマイグレーションする方法については、 非共用キューの共用キューへのマイグレーション を参照してください。

既存のアプリケーションをマイグレーションするときは、以下の事柄を考慮してください。共用キュー環境では、動作が異なることがあります。

理由コード

共用キューを使用するよう既存のアプリケーションをマイグレーションするときは、発行される可能性のある新しい理由コードをチェックしてください。

トリガー

共用アプリケーション・キューを使用する場合、トリガー操作は、コミットされたメッセージに対してのみ機能します (非共用アプリケーション・キューでは、トリガー操作はすべてのメッセージに対して機能します)。

アプリケーションを開始するためにトリガー操作を使用する場合は、共用開始キューを使用できます。903 ページの表 131 では、使用する開始キューのタイプを決定する際に考慮しなければならない事柄を説明しています。

表 *131.* 共用開始キューを使用する場合

	非共用アプリケーション・キュー	**共用アプリケーション・キュー**
非共用開始キュー	旧リリースの場合。	トリガー・タイプ FIRST または DEPTH を使用する場合は、非共用開始キューと共用アプリケーション・キューを一緒に使用できます。 余分のトリガー・メッセージが生成される可能性がありますが、このセットアップは、長時間稼働するアプリケーション (CICS bridge など) のトリガー操作に向いており、高可用性を提供します。 トリガー・タイプ FIRST または DEPTH の場合は、トリガー・メッセージは、トリガー・モニターを実行しており、アプリケーション・キューを入力用にまだオープンしていないすべてのキュー・マネージャー上のアプリケーションのインスタンスを起動します。 キュー・マネージャーごとに 1 つのトリガー・メッセージが生成されます。特定のキュー・マネージャー上で、非共用ローカル開始キューに対して複数のトリガー・モニターを実行している場合、これらのモニターは競合してメッセージを処理します。

表 131. 共用開始キューを使用する場合 (続き)

	非共用アプリケーション・キュー	共用アプリケーション・キュー
共用開始キュー	共用開始キューと非共用アプリケーション・キューは一緒には使用できません。	トリガー・タイプが EVERY の場合、アプリケーションが共有アプリケーション・キューにメッセージを書き込むと、書き込み側のキュー・マネージャーは、トリガーに関係のあるキュー・マネージャー (すべてのイベント) を判別し、それらのキュー・マネージャーの 1 つに通知を送信します。 通知を受けたキュー・マネージャーでは、結果として開始キューに対するトリガー・メッセージが生成されます。 **注 :** トリガー・タイプが EVERY の共用アプリケーション・キューがある場合は、共用開始キューを使用してください。そうしない場合、特定の状況で (例えば、キュー・マネージャーで障害が発生した場合) トリガー・メッセージが失われることがあります。 FIRST や DEPTH のトリガー・タイプでは、それぞれのキュー・マネージャーによって、入力用にオープンしている指定の開始キューの入った 1 つのトリガー・メッセージが生成されます。 **注 :** トリガー・タイプが FIRST または DEPTH の場合、ビジー状態のトリガー・モニター・インスタンスは、それほどビジーでないトリガー・モニターが共用開始キュー内の複数のメッセージを処理する可能性を残しています。 したがって、ある特定のキュー・マネージャーに対して、サーバー・アプリケーションの複数インスタンスを開始できます。 これらの複数インスタンスは複数のトリガー・メッセージの処理の結果として開始されることに注意してください。 トリガー・タイプが FIRST または DEPTH の場合は通常、アプリケーション・インスタンスが既にアプリケーション・キューを処理していれば、アプリケーションが接続されているキュー・マネージャーによって別のトリガー・メッセージが生成されることはありません。

MQINQ

MQINQ 呼び出しを使用して共用キューについての情報を表示するときは、入力または出力用にキューをオープンしている MQOPEN 呼び出しの数の値は、その呼び出しを発行したキュー・マネージャーだけに関連しています。 キュー共用グループ内の、その他のキューをオープンしているキュー・マネージャーについての情報は作成されません。

z/OS IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーション

この情報は、IBM MQ を使用して IMS アプリケーションを作成する際に役立ちます。

- IMS アプリケーションで同期点と MQI 呼び出しを使用する場合は、70 ページの『IBM MQ を使用した IMS アプリケーションの作成』を参照してください。
- IBM MQ - IMS ブリッジを使用するアプリケーションを作成する場合は、74 ページの『IMS ブリッジ・アプリケーションの作成』を参照してください。

IBM MQ for z/OS 上の IMS および IMS ブリッジ・アプリケーションについて詳しくは、以下のリンクを参照してください。

- 70 ページの『IBM MQ を使用した IMS アプリケーションの作成』
- 74 ページの『IMS ブリッジ・アプリケーションの作成』

関連概念

722 ページの『Message Queue Interface の概要』
メッセージ・キュー・インターフェース (MQI) (Message Queue Interface (MQI)) コンポーネントについて説明します。

735 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』
IBM MQ プログラミング・サービスを使用するには、プログラムがキュー・マネージャーに接続していなければなりません。 この情報を使用して、キュー・マネージャーへの接続方法とキュー・マネージャーからの切断方法について学習します。

742 ページの『オブジェクトのオープンとクローズ』
ここでは、IBM MQ オブジェクトのオープンとクローズについて説明します。

753 ページの『キューへのメッセージの書き込み』
この情報を使用して、メッセージをキューに書き込む方法について学習します。

768 ページの『キューからのメッセージの読み取り』
この情報を使用して、キューからのメッセージの読み取りについて学習します。

854 ページの『オブジェクト属性の照会と設定』
属性は、IBM MQ オブジェクトの性質を定義する特性です。

857 ページの『作業単位のコミットとバックアウト』
ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

869 ページの『トリガーによる IBM MQ アプリケーションの開始』
トリガーについて、およびトリガーを使用して IBM MQ アプリケーションを開始する方法について理解します。

888 ページの『MQI とクラスターの処理』
呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

893 ページの『IBM MQ for z/OS 上でのアプリケーションの使用/作成方法』
IBM MQ for z/OS アプリケーションは、いくつもの異なる環境で稼働するプログラム群で構成することができます。 これは、複数の環境で使用可能な機能を利用できることを意味します。

z/OS *IBM MQ を使用した IMS アプリケーションの作成*

IMS アプリケーションで IBM MQ を使用する際には、さらに考慮事項があります。これには、どの MQ API 呼び出しを使用できるか、および同期点に使用されるメカニズムが含まれます。

IBM MQ for z/OS での IMS アプリケーションの作成について詳しくは、以下のリンクを参照してください。

- 71 ページの『IMS アプリケーションにおける同期点』
- 71 ページの『IMS アプリケーションにおける MQI 呼び出し』

制約事項

IMS アダプターを使用するアプリケーションが使用できる IBM MQ API 呼び出しには、いくつかの制約事項があります。

以下の IBM MQ API 呼び出しは、IMS アダプターを使用するアプリケーションではサポートされていません。

- MQCB
- MQCB_FUNCTION
- MQCTL

関連概念

74 ページの『IMS ブリッジ・アプリケーションの作成』
このトピックでは、IBM MQ - IMS ブリッジを使用するアプリケーションの作成について説明します。

z/OS *IMS* アプリケーションにおける同期点

IMS アプリケーションでは、IOPCB および CHKP (checkpoint) に対する GU (get unique) などの IMS 呼び出しを使用して同期点を確立します。

直前のチェックポイント以降のすべての変更をバックアウトするには、IMS ROLB (rollback) 呼び出しを使用できます。 詳しくは、 IMS 資料の ROLB 呼び出し を参照してください。

このキュー・マネージャーは、2 フェーズ・コミット・プロトコルの参加プログラムです。 IMS 同期点管理プログラムはそのためのコーディネーターです。

すべてのオープン状態のハンドルは、IMS アダプターによって同期点でクローズされます (バッチまたは非メッセージ・ドリブン BMP 環境の場合を除く)。 これは、別のユーザーが次の作業単位を開始することもあり、また、IBM MQ のセキュリティー検査が、(MQPUT または MQGET 呼び出し時でなく) MQCONN、MQCONNX および MQOPEN 呼び出し時に実行されるためです。

しかし、入力待ち (WFI (Wait-for-Input)) または疑似入力待ち (PWFI (pseudo Wait-for-Input)) 環境では、IMS は、次のメッセージが到着するか、QC 状況コードがアプリケーションに戻されるまでは、IBM MQ にハンドルをクローズするよう通知しません。 アプリケーションが IMS 領域内で待機していて、これらのハンドルのいずれかが、トリガーされたキューに属している場合、それらのキューはオープンしているのでトリガーは発生しません。 この理由のため、 WFI または PWFI 環境で実行するアプリケーションは、次のメッセージ用に IOPCB に対する GU を行う前に、明示的な MQCLOSE を キュー・ハンドルに対して実行する必要があります。

IMS アプリケーション (BMP または MPP) が MQDISC 呼び出しを発行した場合は、オープン・キューはクローズされますが、暗黙の同期点はとられません。 アプリケーションが正常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のコミットが行われます。 アプリケーションが異常終了した場合は、すべてのオープン・キューがクローズされ、暗黙のバックアウトが行われます。

z/OS *IMS* アプリケーションにおける *MQI* 呼び出し

この情報を使用して、サーバー・アプリケーションおよび照会アプリケーションにおける MQI 呼び出しの使用方法について学習します。

この節では、以下のタイプの IMS アプリケーションで MQI 呼び出しを使用する方法について説明します。

- 906 ページの『サーバー・アプリケーション』
- 908 ページの『照会アプリケーション』

サーバー・アプリケーション

MQI サーバー・アプリケーション・モデルの概要を次に示します。

```
Initialize/Connect
.
Open queue for input shared
.
Get message from IBM MQ queue
.
Do while Get does not fail
.
If expected message received
Process the message
Else
Process unexpected message
End if
.
Commit
.
Get next message from IBM MQ queue
.
End do
.
Close queue/Disconnect
.
END
```

サンプル・プログラム CSQ4ICB3 では、C/370 での、このモデルを使用した BMP の実装方式を示します。プログラムは、まず IMS、次に IBM MQ の順で通信を確立します。

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

IMS 初期設定では、プログラムがメッセージ・ドリブン、バッチ指向のいずれの BMP として呼び出されているかが判定され、それぞれの場合に応じて、IBM MQ キュー・マネージャーの接続とキュー・ハンドルが制御されます。

```
InitIMS
-------
Get the IO, Alternate and Database PCBs
Set MessageOriented to true

Call ctdli to handle status codes rather than abend
If call is successful (status code is zero)
While status code is zero
Call ctdli to get next message from IMS message queue
If message received
Do nothing
Else if no IOPBC
Set MessageOriented to false
Initialize error message
Build 'Started as batch oriented BMP' message
Call ReportCallError to output the message
End-if
Else if response is not 'no message available'
Initialize error message
Build 'GU failed' message
Call ReportCallError to output the message
Set return code to error
End-if
End-if
End-while
Else
Initialize error message
Build 'INIT failed' message
Call ReportCallError to output the message
Set return code to error
End-if

Return to calling function
```

IBM MQ 初期設定では、キュー・マネージャーに接続して、キューをオープンします。 メッセージ・ドリブンの BMP では、それぞれの IMS 同期点が取られてから呼び出されます。バッチ指向の BMP では、プログラムの始動時にのみ呼び出されます。

```
InitMQM
-------
Connect to the queue manager
If connect is successful
Initialize variables for the open call
Open the request queue
If open is not successful
Initialize error message
Build 'open failed' message
Call ReportCallError to output the message
Set return code to error
End-if
Else
Initialize error message
Build 'connect failed' message
Call ReportCallError to output the message
```

```
Set return code to error
End-if

Return to calling function
```

MPP 内のサーバー・モデルの実装方式は、MPP が呼び出しごとに 1 つの作業単位を処理するという事実に影響されます。 これは、同期点 (GU) がとられるとき、接続とキュー・ハンドルがクローズされ、次の IMS メッセージが送達されるからです。 この制限は、以下のいずれかの方法によって部分的に克服することができます。

- **1 つの作業単位内で多数のメッセージを処理する方法**

  これには、以下の処理が含まれます。

  – メッセージの読み取り
  – 必要な更新の処理
  – 応答の書き込み

  同期点がとられる時点で、1 つのループのなかで、すべてのメッセージが処理されるか、または最大数のメッセージ群が処理されるまで、上記の処理を行います。

  特定のタイプのアプリケーション (例えば、単純なデータベースの更新または照会) についてのみ、この方法が適用できます。 MQI 応答メッセージは、処理中の MQI メッセージの発信元の許可によって書き込むことができますが、IMS リソースの更新がセキュリティーに及ぼす影響には注意する必要があります。

- **MPP を呼び出すごとに 1 つのメッセージを処理し、使用可能なすべてのメッセージを処理するための MPP のスケジュールを複数作成できるようにする方法**

  IBM MQ IMS トリガー・モニター・プログラム (CSQQTRMN) は、IBM MQ キューにメッセージがあって、それにサービスを提供するアプリケーションがない場合に、MPP トランザクションをスケジュールするために使用します。

  トリガー・モニターが MPP を開始すると、キュー・マネージャー名とキュー名は、以下の COBOL コードの抜粋で示すように、プログラムに渡されます。

```
* Data definition extract
01 WS-INPUT-MSG.
05 IN-LL1         PIC S9(3) COMP.
05 IN-ZZ1         PIC S9(3) COMP.
05 WS-STRINGPARM      PIC X(1000).
01 TRIGGER-MESSAGE.
COPY CMQTMC2L.
*
* Code extract
GU-IOPCB SECTION.
MOVE SPACES TO WS-STRINGPARM.
CALL 'CBLTDLI' USING GU,
IOPCB,
WS-INPUT-MSG.
IF IOPCB-STATUS = SPACES
MOVE WS-STRINGPARM TO MQTMC.
*  ELSE handle error
*
* Now use the queue manager and queue names passed
DISPLAY 'MQTMC-QMGRNAME     ='
MQTMC-QMGRNAME OF MQTMC '='.
DISPLAY 'MQTMC-QNAME       ='
MQTMC-QNAME  OF MQTMC '='.
```

タスクの実行が長時間になることが予想されるサーバー・モデルは、バッチ処理領域内のサポートをお勧めします。ただし、BMP は CSQQTRMN で起動することができません。

照会アプリケーション

照会または更新作業を開始する一般的な IBM MQ アプリケーションは以下の処理を行います。

- ユーザーからのデータの収集
- 1 つ以上の IBM MQ メッセージの書き込み

- 応答メッセージの読み取り (待機しなければならない場合もある)
- ユーザーへの応答

IBM MQ キューに書き込まれたメッセージは、コミットされるまでは他の IBM MQ アプリケーションで使用できないため、それらのメッセージを同期点から外すか、あるいは IMS アプリケーションを 2 つのトランザクションに分割する必要があります。

照会で 1 つのメッセージの書き込みを処理する場合、*no syncpoint* オプションを使用できます。しかし、照会がより複雑な場合、またはリソースの更新が絡む場合には、同期点制御が稼働していないときに障害が発生すると、整合性上の問題が起こる可能性があります。

これを克服するために、プログラム間メッセージ通信を使用する MQI 呼び出しを使用して IMS MPP トランザクションを分割することができます。これについては、「*IMS* システム間連絡 *(ISC)*」を参照してください。 これによって、照会プログラムが MPP に実装できるようになります。

```
Initialize first program/Connect
.
Open queue for output
.
Put inquiry to IBM MQ queue
.
Switch to second IBM MQ program, passing necessary data in save
pack area (this commits the put)
.
END
.
.
Initialize second program/Connect
.
Open queue for input shared
.
Get results of inquiry from IBM MQ queue
.
Return results to originator
.
END
```

z/OS *IMS* ブリッジ・アプリケーションの作成

このトピックでは、IBM MQ - IMS ブリッジを使用するアプリケーションの作成について説明します。

IBM MQ-IMS ブリッジに関する情報については、「IMS ブリッジ」を参照してください。

IBM MQ for z/OS 上での IMS ブリッジ・アプリケーションの作成について詳しくは、以下のリンクを参照してください。

- 75 ページの『IMS ブリッジがメッセージを処理する方法』
- 916 ページの『IBM MQ を使用した IMS トランザクション・プログラムの作成』

関連概念

70 ページの『IBM MQ を使用した IMS アプリケーションの作成』
IMS アプリケーションで IBM MQ を使用する際には、さらに考慮事項があります。これには、どの MQ API 呼び出しを使用できるか、および同期点に使用されるメカニズムが含まれます。

z/OS *IMS* ブリッジがメッセージを処理する方法

IBM MQ-IMS ブリッジを使用して IMS アプリケーションにメッセージを送信する場合、メッセージを特殊なフォーマットで構成する必要があります。

また、ターゲット IMS システムの XCF グループとメンバー名を指定するストレージ・クラスで定義されている IBM MQ キューにもメッセージを書き込む必要があります。 これらは、MQ-IMS ブリッジ・キュー、または簡単に**ブリッジ**・キューとして知られています。

IBM MQ-IMS ブリッジを QSGDISP(QMGR) を指定して定義した場合、または QSGDISP(SHARED) と NOSHARE オプションを指定して定義した場合は、ブリッジ・キューに対する排他的な入力権限 (MQOO_INPUT_EXCLUSIVE) がブリッジになければなりません。

IMS アプリケーションにメッセージを送信する前に、ユーザーは IMS にサインオンする必要はありません。セキュリティー・チェックには、MQMD 構造体の *UserIdentifier* フィールドのユーザー ID が使われます。 チェックのレベルは、IBM MQ が IMS に接続されるときに決定されます。詳しくは、IMS ブリッジに関するアプリケーション・アクセス制御を参照してください。 これにより、疑似サインオンをインプリメントすることが可能になります。

IBM MQ - IMS ブリッジは、次のタイプのメッセージを受け付けます。

- メッセージには IMS トランザクション・データと MQIIH 構造体 (MQIIH を参照) が含まれています。

```
MQIIH LLZZ<trancode><data>[LLZZ<data>][LLZZ<data>]
```

注:

1. 角括弧 [ ] は、任意指定の複数セグメントを表します。
2. MQIIH 構造体を使用するには、MQMD 構造体の *Format* フィールドを MQFMT_IMS に設定してください。

- IMS トランザクション・データが入っているが、MQIIH 構造体は入っていないメッセージ。

```
LLZZ<trancode><data> \
[LLZZ<data>][LLZZ<data>]
```

IBM MQ は、メッセージ・データをチェックして、LL バイト数と MQIIH (存在する場合) の長さの合計が、メッセージの長さに等しいかどうかを確認します。

IBM MQ - IMS ブリッジは、ブリッジ・キューからメッセージを読み取ると、それらのメッセージを次のように処理します。

- メッセージに MQIIH 構造体が含まれている場合、ブリッジは MQIIH を検証し (MQIIH を参照)、OTMA ヘッダーを作成してから IMS にメッセージを送信します。 トランザクション・コードは、入力メッセージに指定されます。 これが LTERM の場合、IMS は応答として DFS1288E メッセージを出します。 トランザクション・コードがコマンドを表す場合、IMS は、そのコマンドを実行するか、もしくはメッセージを IMS 内のトランザクション用のキューに入れます。
- そのメッセージに IMS トランザクション・データは入っているが、MQIIH 構造体が入っていない場合は、IMS ブリッジは次のように仮定します。
  - トランザクション・コードはユーザー・データのバイト 5 から 12 までである。
  - トランザクションは非会話型である。
  - トランザクションはコミット・モード 0 (コミット後送信) である。
  - MQMD 内の *Format* は *MFSMapName* (入力上) として使用される。
  - セキュリティー・モードは MQISS_CHECK である。

  応答メッセージも MQIIH 構造体なしで作成され、MQMD の *Format* は IMS 出力の *MFSMapName* から取得されます。

IBM MQ - IMS ブリッジでは、各 IBM MQ キューにつき 1 つまたは 2 つの Tpipe を使用します。

- 同期化 Tpipe は、コミット・モード 0 (COMMIT_THEN_SEND) を使用するすべてのメッセージに使用されます (これらは、IMS /DIS TMEMBER クライアント TPIPE xxxx コマンドの状況フィールドに SYN と表示されます)。
- 非同期 Tpipe は、コミット・モード 1 (SEND_THEN_COMMIT) のメッセージに使用します。

Tpipe は、初めて使用するときに IBM MQ によって作成されます。 非同期 Tpipe は、IMS を再始動するまで削除されません。 同期 Tpipe は、IMS をコールド・スタートするまで削除されません。 ユーザーがこれらの Tpipe を削除することはできません。

IBM MQ - IMS ブリッジのメッセージの扱い方について詳しくは、以下のトピックを参照してください。

- 76 ページの『IBM MQ メッセージと IMS トランザクション・タイプの対応関係』
- 77 ページの『メッセージを IMS キューに書き込めない場合』

- 77 ページの『IMS ブリッジのフィードバック・コード』
- 77 ページの『IMS ブリッジからのメッセージ内の MQMD フィールド』
- 79 ページの『IMS ブリッジからのメッセージ内の MQIIH フィールド』
- 79 ページの『IMS からの応答メッセージ』
- 80 ページの『IMS トランザクションにおける代替応答 PCB の使用』
- 80 ページの『IMS からの非送信請求メッセージの送信』
- 80 ページの『メッセージのセグメント化』
- 80 ページの『IMS ブリッジとの間のメッセージのデータ変換』

関連概念

916 ページの『IBM MQ を使用した IMS トランザクション・プログラムの作成』

IBM MQ を介して IMS トランザクションを処理するために必要なコーディングは、 IMS トランザクションが必要とするメッセージ形式と、トランザクションが返すことができる応答の範囲によって異なります。しかし、アプリケーションが IMS 画面形式制御情報を扱うときには、考慮すべき点がいくつかあります。

z/OS *IBM MQ* メッセージと *IMS* トランザクション・タイプの対応関係

表は IBM MQ メッセージと IMS トランザクション・タイプの対応関係を示しています。

表 *132. IBM MQ* メッセージと *IMS* トランザクション・タイプの対応関係

IBM MQ メッセージ・タイプ	**コミット後送信 (モード 0) - 同期 IMS Tpipe を使用**	**送信後コミット (モード 1) - 非同期 IMS Tpipe を使用**
持続 IBM MQ メッセージ	• リカバリー可能な全機能トランザクション • リカバリー不能なトランザクションの場合は IMS で拒否	• ファースト・パス・トランザクション • 会話型トランザクション • 全機能トランザクション
非持続 IBM MQ メッセージ	• リカバリー不能な全機能トランザクション • リカバリー可能なトランザクションは IMS V8 および APAR PQ61404、および以降すべてのバージョンの IMS で許可	• ファースト・パス・トランザクション • 会話型トランザクション • 全機能トランザクション

注 : IMS コマンドは、コミット・モードが 0 の持続 IBM MQ メッセージを使用できません。 詳しくは、 コミット・モード (commitMode) を参照してください。

z/OS メッセージを *IMS* キューに書き込めない場合

メッセージを IMS キューに書き込めない場合に取られる処置について説明します。

メッセージを IMS キューに書き込めない場合は、IBM MQ によって次の処理が行われます。

- メッセージが無効なために IMS に書き込めない場合は、そのメッセージは送達不能キューに書き込まれ、メッセージがシステム・コンソールに送られます。
- メッセージが有効であるのに、IMS によって拒否された場合、IBM MQ はシステム・コンソールにエラー・メッセージを送信します。このメッセージは IMS センス・コードを含んでおり、IBM MQ メッセージは送達不能キューに書き込まれます。 IMS センス・コードが 001A の場合、IMS はその障害の理由を含む IBM MQ メッセージを応答先キューに送信します。

  注 : 上のリストのような環境では、IBM MQ が何らかの理由で送達不能キューにメッセージを書き込めないとき、メッセージは発信元の IBM MQ キューに返されます。 エラー・メッセージがシステム・コンソールに送られ、そのキューから、他のメッセージは送られません。

  メッセージを再送するには、次の **いずれか** を行ってください。

  – そのキューに対応する IMS 内の Tpipes を停止し、再始動する。

- そのキューを GET(DISABLED) に変更し、再び GET(ENABLED) に変更する。
- IMS または OTMA を停止、再始動する。
- IBM MQ サブシステムを停止し、再始動する。

- メッセージが、メッセージ・エラー以外の理由で IMS に拒否された場合、IBM MQ メッセージは元のキューに返され、IBM MQ はそのキューの処理を停止します。そして、システム・コンソールにはエラー・メッセージが送られます。

  例外報告メッセージが必要な場合は、そのブリッジは発信元の許可によりそのメッセージを応答先キューに書き込みます。 例外報告メッセージを応答先キューに書き込めない場合は、その報告メッセージはブリッジの許可により送達不能キューに書き込まれます。 メッセージ DLQ に書き込めない場合、そのメッセージは廃棄されます。

z/OS *IMS* ブリッジのフィードバック・コード

IMS センス・コードは通常、CSQ2001I のような IBM MQ コンソール・メッセージでは 16 進形式の出力となります (例えば、センス・コード 0x001F)。 IBM MQ フィードバック・コード (送達不能キューに書き込まれるメッセージの送達不能ヘッダーの中に見られる) は、10 進数です。

IMS ブリッジのフィードバック・コードの範囲は 301 から 399 まで、または、NACK センス・コード 0x001A の場合は 600 から 855 までです。 このフィードバック・コードは、IMS-OTMA センス・コードから次のようにマッピングされます。

1. IMS-OTMA センス・コードが 16 進数から 10 進数に変換されます。
2. 1 の計算結果に 300 が加算され、 IBM MQ *Feedback* コードが示されます。
3. IMS-OTMA センス・コード 0x001A、10 進数 26 は特殊なケースです。 範囲 600 から 855 までのフィードバック・コードが生成されます。

   a. IMS-OTMA 理由コードが 16 進数から 10 進数に変換されます。

   b. a の計算結果に 600 を足して、IBM MQ フィードバック・コードを作成します。

IMS-OTMA センス・コードについては、「NAK メッセージ用の OTMA センス・コード」を参照してください。

z/OS *IMS* ブリッジからのメッセージ内の *MQMD* フィールド

IMS ブリッジからのメッセージ内の MQMD フィールドについて説明します。

元のメッセージの MQMD は、OTMA ヘッダーの User Data セクションで、IMS によって渡されます。 メッセージが IMS から発信している場合、これは IMS 宛先解決出口によって構築されます。 IMS から受信されるメッセージの MQMD は次のように構築されます。

StrucID
"MD "

バージョン
MQMD_VERSION_1

レポート
MQRO_NONE

MsgType
MQMT_REPLY

Expiry
MQIIH の Flags フィールドで MQIIH_PASS_EXPIRATION が設定されている場合は、このフィールドには残りの満了時間が含まれ、設定されていない場合は、このフィールドは MQEI_UNLIMITED に設定されます。

Feedback
MQFB_NONE

Encoding
MQENC.Native (z/OS システムのエンコード)

CodedCharSetId
MQCCSI_Q_MGR (z/OS システムの CodedCharSetID)。

Format
入力メッセージの MQMD.Format が MQFMT_IMS の場合は MQFMT_IMS。その他の場合は、IOPCB.MODNAME。

優先順位
入力メッセージの MQMD.Priority。

Persistence
コミット・モードによって異なります。CM-1 の場合は、入力メッセージの MQMD.Persistence。CM-0 の場合は、IMS メッセージの回復可能性と一致します。

MsgId
MQRO_PASS_MSG_ID の場合、MQMD.MsgId。その他の場合は、New MsgId (デフォルト)。

CorrelId
MQRO_PASS_CORREL_ID の場合は、入力メッセージの MQMD.CorrelId。その他の場合は、入力メッセージの MQMD.MsgId (デフォルト)。

BackoutCount
0

ReplyToQ
ブランク

ReplyToQMgr
ブランク (MQPUT の間はキュー・マネージャーによってローカル・キュー・マネージャー名に設定されます)。

UserIdentifier
入力メッセージの MQMD.UserIdentifier。

AccountingToken
入力メッセージの MQMD.AccountingToken。

ApplIdentityData
入力メッセージの MQMD.ApplIdentityData。

PutApplType
エラーがない場合、MQAT_XCF。その他の場合は、MQAT_BRIDGE。

PutApplName
エラーがない場合、<XCFgroupName><XCFmemberName>。その他の場合は、QMGR 名。

PutDate
メッセージを書き込んだ日付

PutTime
メッセージを書き込んだ時刻

ApplOriginData
ブランク

z/OS *IMS* ブリッジからのメッセージ内の *MQIIH* フィールド

IMS ブリッジからのメッセージ内の MQIIH フィールドについて説明します。

IMS から受信されるメッセージの MQIIH は次のように構築されます。

StrucId
"IIH "

バージョン
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat がブランクでない場合は、入力メッセージの MQIIH.ReplyToFormat。ブランクの場合は、IOPCB.MODNAME。

フラグ
0

LTermOverride
OTMA ヘッダーの LTERM 名 (Tpipe)。

MFSMapName
OTMA ヘッダーのマップ名。

ReplyToFormat
ブランク

Authenticator
応答メッセージが MQ-IMS ブリッジ・キューに書き込まれる場合は入力メッセージの MQIIH.Authenticator。書き込まれない場合はブランク。

TranInstanceId
会話中の場合は、OTMA ヘッダーの会話 ID またはサーバー・トークン。 バージョンが V14 より前の IMS では、会話中でない場合、このフィールドは常にヌルになります。 IMS V14 以降では、会話中でない場合でも、このフィールドは IMS によって設定されることがあります。

TranState
会話中の場合は、"C"。それ以外の場合はブランク。

CommitMode
OTMA ヘッダーのコミット・モード ("0" または "1")。

SecurityScope
ブランク

Reserved
ブランク

z/OS *IMS* からの応答メッセージ

IMS トランザクション ISRT がその IOPCB へ送られると、メッセージは発信元の LTERM または TPIPE に戻されます。

これらは IBM MQ では応答メッセージとして表示されます。 IMS からの応答メッセージは、元のメッセージで指定された応答先キューに書き込まれます。 応答メッセージを応答先キューに書き込めない場合は、そのメッセージはブリッジの許可により送達不能キューに書き込まれます。 応答メッセージを送達不能キューに書き込めない場合は、メッセージが受信できないことを示す否定応答が IMS に送られます。 この時点で、そのメッセージに対する責任は IMS に戻ります。 コミット・モード 0 を使用している場合、Tpipe からのメッセージはブリッジには送られず、IMS キュー上に留まります。つまり、再始動するまではメッセージは送られません。 コミット・モード 1 を使用している場合、他の作業を続けることができます。

応答が MQIIH 構造体を持っている場合、その形式は MQFMT_IMS となります。この構造体を持っていない場合は、その形式はメッセージを挿入するときに使用される IMS MOD 名により指定されます。

z/OS *IMS* トランザクションにおける代替応答 *PCB* の使用

IMS トランザクションが代替応答 PCB (ALTPCB に対して ISRT を使用する場合、または変更可能な PCB に CHNG 呼び出しを発行する場合) を使用し、そのメッセージを転送する必要があるかどうかを判別するために、プリルーティング出口 (DFSYPRX0) が呼び出されます。

メッセージの再経路指定が必要な場合は、宛先解決出口 (DFSYDRU0) が呼び出され、宛先の確認とヘッダー情報の作成が行われます。これらの出口プログラムについては、IMS における OTMA 出口の使用および事前経路指定出口 DFSYPRX0 を参照してください。

出口でアクションが取られない限り、 IBM MQ キュー・マネージャーから開始された IMS トランザクションからのすべての出力は、IOPCB または ALTPCB に関係なく、同じキュー・マネージャーに戻されます。

z/OS *IMS* からの非送信請求メッセージの送信

IMS から IBM MQ キューにメッセージを送信するには、ISRT が ALTPCB に対して実行する IMS トランザクションを呼び出す必要があります。

事前経路指定および宛先解決出口を作成して、IMS からの非送信請求メッセージの経路を指定し、OTMA ユーザー・データを作成することによって、メッセージの MQMD が正確に構築されるようにする必要があります。 これらの出口プログラムについては、事前経路指定出口 DFSYPRX0 および宛先解決ユーザー出口を参照してください。

注 : IBM MQ - IMS ブリッジは、受信したメッセージが応答メッセージと非送信請求メッセージのどちらなのかを判断することができません。 いずれの場合であっても、メッセージも同じようにして扱い、そのメッセージとともに受信した OTMA UserData に基づいて、応答の MQMD および MQIIH を構築します。

非送信請求メッセージの場合は、Tpipe を新規に作成できます。 例えば、既存の IMS トランザクションを新規の LTERM (例: PRINT01) に変更したときに、実装システムでは出力を OTMA によって転送する必要がある場合、新規の Tpipe (この場合は PRINT01) が作成されます。 デフォルトの場合、作成されるのは非同期 Tpipe となります。 実装システムでメッセージをリカバリー可能にする必要がある場合は、宛先解決出口の出力フラグを設定してください。 詳細については、 *IMS* カスタマイズの手引き を参照してください。

z/OS メッセージのセグメント化

単一のセグメントまたは複数のセグメントの入力が予期されるとき、IMS トランザクションを定義できます。

発信元の IBM MQ アプリケーションは、MQIIH 構造体に続くユーザー入力を 1 つまたは複数の LLZZ データ・セグメントとして作成する必要があります。 IMS メッセージのすべてのセグメントは、1 回の MQPUT 呼び出しで送られる 1 つの IBM MQ メッセージの中に入っていなければなりません。

1 つの LLZZ データ・セグメントの最大長は、IMS/OTMA によって定義されます (32767 バイト)。 IBM MQ メッセージの全体の長さは、LL バイトの合計に、MQIIH 構造体の長さを加えた値となります。

応答のすべてのセグメントは、1 つの IBM MQ メッセージが入っています。

このほか、MQFMT_IMS_VAR_STRING 形式のメッセージについては、32 KB の制限があります。 ASCII 混合 CCSID メッセージ内のデータが EBCDIC 混合 CCSID メッセージに変換されるとき、SBCS 文字と DBCS 文字との間の遷移が発生するたびに、シフトイン・バイトまたはシフトアウト・バイトが追加されます。 32 KB の制限は、メッセージの最大サイズに適用されます。 つまり、メッセージの LL フィールドが 32 KB を超えることができないため、メッセージはシフトイン文字およびシフトアウト文字を含めて 32 KB を超えることができません。 メッセージを作成するアプリケーションでは、これを考慮に入れる必要があります。

z/OS *IMS* ブリッジとの間のメッセージのデータ変換

データ変換は、メッセージをそのストレージ・クラス用に定義された XCF 情報を持つ宛先キューに書き込むときに、分散キューイング機能 (必要なすべての出口を呼び出すことができる) またはグループ内キューイング・エージェント (出口の使用をサポートしない) によって実行されます。 データ変換は、パブリッシュ/サブスクライブによってメッセージがキューに送信された場合には行われません。

必要な出口はすべて、CSQXLIB DD ステートメントで参照されるデータ・セット内の分散キューイング機能から利用できなければなりません。 つまり、任意の IBM MQ プラットフォームから IBM MQ - IMS ブリッジを使用して IMS アプリケーションにメッセージを送信できます。

変換エラーがある場合、そのメッセージは未変換キューに書き込まれます。これは、IBM MQ - IMS ブリッジがそのヘッダー形式を認識できないため、最終的にそのブリッジによってエラーとして処理されます。 変換エラーが発生すると、エラー・メッセージが z/OS コンソールに送られます。

一般的なデータ変換の詳細については、988 ページの『データ変換出口の作成』を参照してください。

IBM MQ - IMS ブリッジへのメッセージの送信

変換が正しく行われるようにするため、キュー・マネージャーにメッセージの形式を指示する必要があります。

メッセージが MQIIH 構造体を持っている場合は、MQMD 内の *Format* は組み込み形式 MQFMT_IMS に設定し、MQIIH 内の *Format* はメッセージ・データを記述する形式の名前に設定する必要があります。 MQIIH がない場合は、MQMD 内の *Format* を所定の形式名に設定します。

データ (LLZZ 以外) がすべて文字データ (MQCHAR) の場合、形式名 (MQIIH または MQMD 内) として組み込み形式 MQFMT_IMS_VAR_STRING を使用します。 文字データでない場合は、独自の形式を使用します。また、その場合は、その形式にデータ変換出口も必要となります。 この出口では、データ自体に加えて、メッセージ内の LLZZ の変換も処理しなければなりません (ただし、メッセージの開始時に MQIIH を処理する必要はありません)。

アプリケーションで *MFSMapName* を使用する場合、MQFMT_IMS でメッセージを代わりに使用して、MQIIH の MFSMapName フィールドで IMS トランザクションに渡すマップ名を定義することができます。

IBM MQ - IMS ブリッジからのメッセージの受信

IMS に送信している元のメッセージに MQIIH 構造体がある場合は、応答メッセージにも MQIIH 構造体があります。

応答が正しく変換されるようにするには、次のステップに従ってください。

- 元のメッセージに MQIIH 構造体がある場合は、応答メッセージに指定したい形式を元のメッセージの MQIIH *ReplytoFormat* フィールドに指定します。 この値は応答メッセージの MQIIH *Format* フィールドに入れられます。 これは、出力データがすべて LLZZ< 文字データ> の形式の場合、特に役に立ちます。
- 元のメッセージに MQIIH 構造体がない場合は、応答メッセージに指定したい形式を IMS アプリケーションの ISRT 内の MFS MOD 名として IOPCB に指定します。

z/OS *IBM MQ* を使用した *IMS* トランザクション・プログラムの作成

IBM MQ を介して IMS トランザクションを処理するために必要なコーディングは、 IMS トランザクションが必要とするメッセージ形式と、トランザクションが返すことができる応答の範囲によって異なります。しかし、アプリケーションが IMS 画面形式制御情報を扱うときには、考慮すべき点がいくつかあります。

IMS トランザクションが 3270 画面から開始される場合、メッセージは IMS メッセージ形式サービスによって送られます。 これにより、トランザクションによって表示されるデータ・ストリームから、端末の依存関係がすべて除去されます。 トランザクションが OTMA によって開始される場合、MFS は関係しません。 アプリケーション論理が MFS でインプリメントされている場合は、これを新しいアプリケーションで再作成する必要があります。

IMS トランザクションによっては、エンド・ユーザーのアプリケーションで、特定の 3270 画面の動作 (例えば、誤ったデータを入力してしまったフィールドの強調表示) を修正することができます。 このタイプの情報は、プログラムによって修正する必要のある画面フィールドごとに 2 バイトの属性フィールドを IMS メッセージに追加することによって伝達されます。

このように、3270 を模倣するようにアプリケーションをコーディングする場合、メッセージの作成または受信時に、これらのフィールドを考慮に入れる必要があります。

以下の項目を処理するために、使用のプログラムに情報をコーディングする必要が生じることがあります。

- どのキーが押されたか (Enter や PF1 など)
- メッセージをアプリケーションに渡すときにカーソルがどこにあるか
- IMS アプリケーションによって属性フィールドが設定されているか
  - 高輝度、通常の輝度、またはゼロの輝度
  - カラー
  - 次に Enter キーを押すときに IMS はフィールドが戻ることを期待しているか
- IMS アプリケーションがヌル文字 (X'3F') をいずれかのフィールドで使用しているか

IMS メッセージに文字データだけ (LLZZ データ・セグメントとは別に) が入っており、MQIIH 構造体を使用している場合、MQMD 形式を MQFMT_IMS に設定し、MQIIH 形式を MQFMT_IMS_VAR_STRING に設定する必要があります。

IMS メッセージに文字データだけ (LLZZ データ・セグメントとは別に) が入っており、MQIIH 構造体を**使用していない** 場合、MQMD 形式を MQFMT_IMS_VAR_STRING に設定し、IMS アプリケーションが応答時に MODname MQFMT_IMS_VAR_STRING を指定するようにしてください。 問題が発生した場合 (例えば、ユーザーがトランザクションの使用を許可されていない場合)、IMS はエラー・メッセージを送信します。これには、DFSMOx の形式の MODname が含まれます。ここで、x は 1 から 5 の範囲の数値です。 これは MQMD.Format に書き込まれます。

IMS メッセージに 2 進データ、バック・データ、または浮動小数点データ (LLZZ データ・セグメントとは別に) が入っている場合、独自のデータ変換ルーチンをコーディングしてください。 IMS 画面のフォーマット設定については、「*IMS/ESA* アプリケーション・プログラミング: トランザクション管理プログラム」を参照してください。

IBM MQ を介して IMS トランザクションを処理するコードを作成する際には、以下のトピックを考慮してください。

- 917 ページの『IMS 会話型トランザクションを呼び出す IBM MQ アプリケーションの作成』
- 917 ページの『IMS コマンドを含むプログラムの作成』
- 918 ページの『トリガー』

IMS 会話型トランザクションを呼び出す IBM MQ アプリケーションの作成

この情報は、 IMS 会話型トランザクションを呼び出す IBM MQ アプリケーションを作成する際の考慮事項のガイドとして使用してください。

IMS 会話を呼び出すアプリケーションを作成する際には、以下の点を考慮してください。

- アプリケーション・メッセージに MQIIH 構造体を組み込みます。
- MQIIH の *CommitMode* を MQICM_SEND_THEN_COMMIT に設定します。
- 新規の会話を呼び出すには、MQIIH の *TranState* を MQITS_NOT_IN_CONVERSATION に設定します。
- 会話の 2 番目以降のステップを呼び出すには、*TranState* を MQITS_IN_CONVERSATION に設定し、*TranInstanceId* をその会話の前のステップに戻るフィールドの値に設定します。
- IMS から送信された元のメッセージが失われた場合でも、 IMS で *TranInstanceId* の値を見つける簡単な方法はありません。
- アプリケーションは、 IMS からのメッセージの *TranState* を検査して、 IMS トランザクションが会話を終了したかどうかを検査する必要があります。
- /EXIT を使用して会話を終了することができます。 また、*TranInstanceId* を引用符で囲み、*TranState* を MQITS_IN_CONVERSATION に設定し、会話を実行している IBM MQ キューを使用する必要があります。
- /HOLD や /REL を使用して、会話を保留または保留解除することはできません。
- IBM MQ - IMS ブリッジを介して呼び出される会話は、IMS が再始動すると、終了します。

IMS コマンドを含むプログラムの作成

アプリケーション・プログラムは、トランザクションの代わりに IBM MQ LLZZ コマンドの形式のメッセージを作成することができます。ここで、 *command* は /DIS TRAN PART または /DIS POOL ALL の形式です。

大部分の IMS コマンドは、この方法で実行できます。詳細については、「*IMS V11* コミュニケーションおよびコネクション」を参照してください。 コマンドの出力は、3270 端末に送信して表示されるテキスト形式で、IBM MQ 応答メッセージとして受信されます。

OTMA には、IMS 表示トランザクション・コマンドの特別な形式がインプリメントされており、設計済みの形式の出力を返します。 正確な形式は、「*IMS V11* コミュニケーションおよびコネクション」で定義されています。 IBM MQ メッセージからこの形式を呼び出すには、先に説明したようにメッセージ・データを構築し (例えば、/DIS TRAN PART)、MQIIH の TranState フィールドに MQITS_ARCHITECTED を設定する必要があります。 IMS はコマンドを処理し、設計済みの形式で応答を返します。 設計済みの応答には、テキスト形式の出力に含まれる見つかるすべての情報に加えて、トランザクションがリカバリー可能とリカバリー不能のどちらに定義されているかを示す情報が含まれています。

トリガー

IBM MQ - IMS ブリッジは、トリガー・メッセージをサポートしていません。

ストレージ・クラスを使用する開始キューを XCF パラメーターで定義する場合、そのキューに書き込まれたメッセージは、ブリッジに到達したときに拒否されます。

プロシージャー型クライアント・アプリケーションの作成

IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

IBM MQ クライアント環境では、アプリケーションを作成して実行できます。 アプリケーションを作成し、使用する IBM MQ MQI client にリンクする必要があります。 アプリケーションを作成およびリンクする方法は、使用しているプラットフォームおよびプログラミング言語に応じて異なります。 クライアント・アプリケーションの作成方法については、924 ページの『IBM MQ MQI clients 用のアプリケーションの作成』を参照してください。

特定の条件が満たされている場合は、コードを変更せずに、完全な IBM MQ 環境と IBM MQ MQI client 環境の両方で IBM MQ アプリケーションを実行できます。 IBM MQ クライアント環境でのアプリケーションの実行については、925 ページの『IBM MQ MQI client 環境でのアプリケーションの実行』を参照してください。

メッセージ・キュー・インターフェース (MQI) を使用して、IBM MQ MQI client 環境で実行するアプリケーションを作成する場合は、IBM MQ アプリケーションの処理が中断されないようにするために MQI 呼び出し時に適用する、付加的な制御があります。 これらの制御については、919 ページの『クライアント・アプリケーションでの MQI の使用法』を参照してください。

その他のアプリケーション・タイプをクライアント・アプリケーションとして準備および実行する方法については、以下のトピックを参照してください。

- 939 ページの『CICS および Tuxedo アプリケーションの準備と実行』
- 49 ページの『Microsoft Transaction Server アプリケーションの準備と実行』
- 941 ページの『IBM MQ JMS アプリケーションの準備と実行』

関連概念

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

5 ページの『IBM MQ 用アプリケーションの開発』
メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

721 ページの『プロシージャー型キューイング・アプリケーションの作成』
この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

809 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
パブリッシュ/サブスクライブ IBM MQ アプリケーションの作成を開始します。

1005 ページの『プロシージャー型アプリケーションの構築』
IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

1042 ページの『プロシージャー型プログラム・エラーの処理』

ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

関連タスク

1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』
これらのサンプル・プログラムは、プロシージャー型言語で作成されており、Message Queue Interface (MQI) の標準的な使用法を示しています。 異なるプラットフォーム上の IBM MQ プログラム。

クライアント・アプリケーションでの MQI の使用法

この一連のトピックでは、メッセージ・キュー・インターフェース (MQI) クライアント環境で実行する場合と、フル IBM MQ キュー・マネージャー環境で実行する場合で、IBM MQ アプリケーションの作成にどのような違いがあるかについて説明します。

アプリケーションを設計するときは、MQI 呼び出し中に加える必要がある制御を考慮して、IBM MQ アプリケーションの処理が中断されないようにしてください。

MQI を使用するアプリケーションを実行できるようになるには、まず特定の IBM MQ オブジェクトを作成する必要があります。 詳しくは、MQI を使用するアプリケーション・プログラムを参照してください。

クライアント・アプリケーションでのメッセージ・サイズの制限制限

キュー・マネージャーにはメッセージの最大長がありますが、クライアント・アプリケーションから送信できるメッセージの最大サイズは、チャネル定義により制限されます。

キュー・マネージャーの最大メッセージ長 (MaxMsgLength) 属性は、そのキュー・マネージャーが処理できるメッセージの最大長です。

Multi マルチプラットフォームでは、キュー・マネージャーの最大メッセージ長属性の値を大きくすることができます。 詳しくは、ALTER QMGR を参照してください。

キュー・マネージャーの MaxMsgLength の値は、MQINQ 呼び出しを使用して調べることができます。

MaxMsgLength 属性を変更した場合、新しい値より大きい長さのキュー、さらにはメッセージがすでにあるかどうかの検査は行われません。 この属性を変更したら、変更内容が有効になるよう、アプリケーションとチャネルを再始動してください。 これが完了すると、キュー・マネージャーまたはキューいずれかの MaxMsgLength を超える新規メッセージを生成することはできなくなります (キュー・マネージャーのセグメント化が許可されている場合を除きます)。

チャネル定義内の最大メッセージ長によって、クライアント接続で転送できるメッセージのサイズが制限されます。 IBM MQ アプリケーションが MQPUT 呼び出しまたは MQGET 呼び出しを使用して、これより大きいメッセージを処理しようとすると、アプリケーションにエラー・コードが戻されます。 チャネル定義の最大メッセージ・サイズ・パラメーターは、クライアント接続を介し、MQCB を使用してコンシュームされる最大メッセージ・サイズには影響しません。

関連概念

923 ページの『MQCONNX の使用法』
MQCONNX 呼び出しを使用して、MQCNO 構造内のチャネル定義 (MQCD) 構造体を指定することができます。

関連資料

最大メッセージ長 (MAXMSGL)
ALTER CHANNEL
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

クライアントまたはサーバー *CCSID* の選択

クライアント用のローカル・コード化文字セット ID (CCSID) を使用してください。 キュー・マネージャーは、必要な変換を実行します。 **MQCCSID** 環境変数を使用して、CCSID をオーバーライドできます。 アプリケーションが複数の PUT を実行する場合、最初の PUT の完了後に MQMD の CCSID およびエンコード・フィールドを上書きできます。

メッセージ・キュー・インターフェース (MQI) を介してアプリケーションからクライアント・スタブに渡されるデータは、IBM MQ MQI client 用にエンコードされた、ローカル CCSID のデータでなければなりま

せん。 接続されたキュー・マネージャーがデータの変換を要求する場合、その変換はキュー・マネージャーのクライアント・サポート・コードによって行われます。

IBM WebSphere MQ 7.0 以降のバージョンでは、キュー・マネージャーで変換できない場合は、Java クライアントで行うことができます。 373 ページの『IBM MQ classes for Java のクライアント接続』を参照してください。

クライアント・コードでは、クライアントの MQI を介する文字データは、そのワークステーション用に構成された CCSID のデータであると想定されます。 この CCSID がサポートされない CCSID であるか、必要な CCSID でない場合は、以下のいずれかのコマンドを使用して、 **MQCCSID** 環境変数でオーバーライドできます。

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

このパラメーターがプロファイル内に設定される場合、すべての MQI データは、コード・ページ 850 のデータであると見なされます。

注 : コード・ページ 850 についての想定は、メッセージ内のアプリケーション・データには適用されません。

メッセージ記述子 (MQMD) の後ろに IBM MQ ヘッダーがある複数の PUT をアプリケーションが実行する場合は、最初の PUT の完了後に MQMD の CCSID およびエンコード・フィールドが上書きされることに注意してください。

最初の PUT の後、これらのフィールドには、接続済みのキュー・マネージャーが IBM MQ ヘッダーを変換するために使用する値が入っています。 アプリケーションがこれらの値を必要な値にリセットするようにしてください。

クライアント・アプリケーションでの *MQINQ* の使用

MQINQ を使用して照会される値の一部は、クライアント・コードによって変更されます。

CCSID
これは、キュー・マネージャーの CCSID ではなく、クライアントの CCSID に設定されます。

MaxMsgLength
この値は、チャネル定義によって制限されると低減されます。 これは、次の値のうち低いほうの値になります。

- キュー定義で定義された値
- チャネル定義で定義された値

詳しくは、MQINQ を参照してください。

クライアント・アプリケーションでの同期点調整の使用

ベース・クライアント上で稼働するアプリケーションは MQCMIT および MQBACK を発行できますが、同期点制御の有効範囲は MQI リソースに限定されます。 外部トランザクション・マネージャーを拡張トランザクション・クライアントと共に使用できます。

IBM MQ 内におけるキュー・マネージャーの役割の 1 つは、アプリケーション内の同期点制御です。 アプリケーションが IBM MQ ベース・クライアント上で稼働している場合、そのアプリケーションは MQCMIT

および MQBACK を発行できますが、同期点制御の有効範囲は MQI リソースに限定されます。 IBM MQ verb MQBEGIN は、ベース・クライアント環境では無効です。

サーバー側で全機能を使用できるキュー・マネージャー環境で実行されるアプリケーションは、トランザクション・モニターを介して複数のリソース (データベースなど) を調整できます。 サーバー側では、IBM MQ 製品に付属のトランザクション・モニター、または他のトランザクション・モニター (CICS など) を使用できます。 ベース・クライアント・アプリケーションでトランザクション・モニターを使用することはできません。

外部トランザクション・マネージャーは、IBM MQ 拡張トランザクション・クライアントと共に使用できます。 拡張トランザクション・クライアントの概要を参照してください。 (詳細について記載されています。)

クライアント・アプリケーションでの先読みの使用

クライアントで先読みを使用することによって、クライアント・アプリケーションがメッセージを要求しなくても非永続メッセージがクライアントに送信されるようにすることができます。

クライアントは、サーバーからのメッセージを必要とする際、サーバーに向けて要求を送信します。 クライアントは、コンシュームするメッセージごとに要求を別々に送信します。 こうした要求メッセージを送信しなくても良いようにして、クライアントの非永続メッセージのコンシュームのパフォーマンスを改善するために、クライアントを先読みを使用するように構成できます。 先読みにより、アプリケーションからの要求がなくてもクライアントへのメッセージ送信が可能となります。

先読みを使用すると、クライアント・アプリケーションから非永続メッセージをコンシュームする際のパフォーマンスを改善することができます。 このパフォーマンスの改善は、MQI アプリケーションと JMS アプリケーションの両方で有効です。 MQGET または非同期コンシュームを使用するクライアント・アプリケーションでは、非永続メッセージをコンシュームする際にパフォーマンスが向上するという利点があります。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、IBM MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアント・アプリケーションは、スレッド化された IBM MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みが使用可能な場合、メッセージはクライアント上の先読みバッファーというメモリーのバッファーに送信されます。 クライアントには、先読みが使用可能でオープンされた各キューごとに先読みバッファーがあります。 先読みバッファー内のメッセージは非永続メッセージです。 クライアントは定期的に、消費したデータ量に関する更新情報をサーバーに提供します。

先読みの使用はすべてのオプションでサポートされているわけではないので、クライアント・アプリケーションの設計によっては、先読みの使用が適していない場合があります。 先読みが使用可能になる際、オプションによっては、MQGET 呼び出し間で一貫性の確保が求められます。 クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納されているメッセージはそのクライアントの先読みバッファー内に保留されます。 詳しくは、787 ページの『非永続メッセージのパフォーマンス向上』を参照してください。

先読みの構成は、IBM MQ クライアント構成ファイルの MessageBuffer スタンザで指定されている MaximumSize、PurgeTime、および UpdatePercentage という 3 つの属性によって制御されます。

クライアント・アプリケーションでの非同期書き込みの使用

非同期書き込みを使用すると、アプリケーションはキュー・マネージャーからの応答を待たずにキューにメッセージを書き込むことができます。 これを使用して、メッセージングのパフォーマンスを向上できる場合があります。

通常、アプリケーションは MQPUT または MQPUT1 を使用して 1 つ以上のメッセージをキューに書き込むと、キュー・マネージャーがその MQI 要求を処理したことを確認するのを待たなければなりません。 メッセージの非同期書き込みを選択することにより、特にクライアント・バインディングを使用するアプリケーションや、多数の小さいメッセージをキューに書き込むアプリケーションのメッセージングのパフォーマンスを向上させることができます。 アプリケーションにメッセージを非同期的に書き込ませる場合、キ

ュー・マネージャーは、呼び出しのたびに成功や失敗を返しません。その代わりに、周期的にエラーの確認を行うことができます。

メッセージをキューに非同期に書き込むには、MQPMO 構造体の *Options* フィールドの MQPMO_ASYNC_RESPONSE オプションを使用します。

非同期書き込みに適格でないメッセージは、キューに同期的に書き込まれます。

MQPUT または MQPUT1 で非同期書き込み応答を要求するときに出される CompCode および MQCC_OK および MQRC_NONE の Reason は、必ずしもメッセージがキューに正常に書き込まれたことを意味するものではありません。 MQPUT 呼び出しまたは MQPUT1 呼び出しごとの成功または失敗が、すぐには戻されないことがありますが、非同期呼び出しで最初に発生したエラーは、MQSTAT への呼び出しにより後から判別できます。

MQPMO_ASYNC_RESPONSE の詳細については、MQPMO オプションを参照してください。

非同期書き込みサンプル・プログラムで、使用可能ないくつかの機能を示しています。 このプログラムの機能および設計の詳細、および実行方法については、1081 ページの『非同期書き込みサンプル・プログラム』を参照してください。

クライアント・アプリケーションでの共用会話の使用

共用会話が使用できる環境では、会話は MQI チャネル・インスタンスを共用できます。

共用会話は 2 つのフィールドで制御されます。両方とも SharingConversations という名前で、1 つはチャネル定義 (MQCD) 構造体の一部であり、もう 1 つはチャネル出口パラメーター (MQCXP) 構造体の一部です。 MQCD の SharingConversations フィールドの値は整数で、チャネルと関連付けられた 1 つのチャネル・インスタンスを共用できる会話の最大数を決定します。 MQCXP の SharingConversations フィールドの値はブール値で、チャネル・インスタンスが現在共用されているかを示します。

共用会話が使用できない環境では、同一の MQCD を指定する新規クライアント接続でチャネル・インスタンスは共用されません。

新規クライアント・アプリケーション接続では、以下の条件に当てはまる場合にチャネル・インスタンスが共用されます。

- 共用会話用に、チャネル・インスタンスのクライアント接続側とサーバー接続側の両方が構成され、これらの値がチャネル出口でオーバーライドされない。
- クライアント接続 MQCD 値 (クライアント MQCONNX 呼び出しで、またはクライアント・チャネル定義テーブル (CCDT) から提供される値) は、既存のチャネル・インスタンスが最初に確立された時点で、クライアント MQCONNX 呼び出しで、または CCDT から提供されたクライアント接続 MQCD 値と完全に一致する。 元の MQCD が後から出口またはチャネル・ネゴシエーションで変更された可能性がありますが、変更が加えられる前にクライアント・システムに提供された値とは一致しています。
- サーバー・サイドでの共用会話の制限を超えていない。

新規クライアント・アプリケーション接続が、他の会話とのチャネル・インスタンスの共用を実行する際の基準に一致している場合は、この決定は、その会話で任意の出口が呼び出される前に行われます。 このような会話の出口で、チャネル・インスタンスが他の会話と共用されることには変わりはありません。 新規チャネル定義に一致する既存のチャネル・インスタンスがない場合は、新規チャネル・インスタンスが接続されます。

チャネル・ネゴシエーションが行われるのは、チャネル・インスタンス上の最初の会話に対してのみです。チャネル・インスタンスのネゴシエーション値はその段階で固定され、それ以降の会話開始時に変更することはできません。 TLS 認証も最初の会話に対してのみ行われます。

チャネル・インスタンスのクライアント接続側またはサーバー接続側のいずれかのソケットでの最初の会話に対する、任意のセキュリティー出口、送信出口または受信出口の初期設定中に、MQCD SharingConversations 値が変更された場合は、これらの出口がすべて初期設定された後に得られた新規の値が、チャネル・インスタンスの共用会話の値を決定する際に使用されます (最低値が優先されます)。

共用会話のネゴシエーション値がゼロの場合、チャネル・インスタンスは絶対に共用されません。 また、このフィールドをゼロに設定する出口プログラムも同じように各自のチャネル・インスタンス上で実行されます。

共用会話のネゴシエーション値がゼロより大きい場合は、MQCXP SharingConversations が後続の出口呼び出しについて TRUE に設定されます。これは、このチャネル・インスタンス上で、その他の出口プログラムが同時に実行可能になることを意味します。

チャネル出口プログラムを作成する場合は、共用会話に関与する可能性のあるチャネル・インスタンス上で実行するかどうかを考慮してください。 チャネル・インスタンスが共用会話に関与する可能性がある場合は、MQCD フィールドの変更による、チャネル出口の他のインスタンスに対する影響を考慮してください。すべての共用会話全体で、すべての MQCD フィールドは共通の値を持っています。 チャネル・インスタンスを確立した後で、出口プログラムが MQCD フィールドの変更を試行した場合、問題が発生する可能性があります。これは、チャネル・インスタンス上で実行中の出口プログラムの別のインスタンスが同時に同じフィールドの変更を試行している場合があるからです。 この状態が出口プログラムで発生する可能性がある場合は、出口コードの MQCD へのアクセスを直列化する必要があります。

会話を共用するために定義されているチャネルの処理をしているが、特定のチャネル・インスタンス上で共用は行わない場合は、チャネル・インスタンス上の最初の会話でのチャネル出口を初期設定する際に、SharingConversations の MQCD 値を 1 または 0 に設定してください。 SharingConversations 値の説明については、SharingConversations を参照してください。

例

共用会話は有効です。

出口プログラムを指定するクライアント接続チャネル定義を使用しています。

このチャネルをはじめて開始する場合は、初期設定時に出口プログラムが一部の MQCD パラメーターを変更します。 これらはチャネルの影響を受けるため、実行中のチャネルで使用している定義は現在、提供された元の定義とは異なります。 MQCXP SharingConversations パラメーターは TRUE に設定されています。

次回、このチャネルを使用してアプリケーションを接続する際に、会話は、同じ元のチャネル定義が使用されるため、以前開始されたチャネル・インスタンスで実行されます。 アプリケーションが 2 番目に接続するチャネル・インスタンスは、最初に接続したインスタンスと同じです。 この結果、アプリケーションは出口プログラムによって変更された定義を使用します。 2 番目の会話で出口プログラムが初期設定されると、MQCD フィールドは変更できますが、チャネルの影響は受けません。 これらの同じ特性が、チャネル・インスタンスを共用するすべての後続の会話に適用されます。

MQCONNX の使用法

MQCONNX 呼び出しを使用して、MQCNO 構造内のチャネル定義 (MQCD) 構造体を指定することができます。

これにより、呼び出し側のクライアント・アプリケーションは、実行時にクライアント接続チャネルの定義を指定できます。 詳しくは、 MQCNO を使用した IBM MQ MQI client でのクライアント接続チャネルの作成を参照してください。 MQCONNX の使用時にサーバーで発行される呼び出しは、サーバーのレベルおよびリスナー構成によって異なります。

MQCONNX をクライアントから使用すると、以下のオプションは無視されます。

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

使用できる MQCD 構造体は、使用する MQCD のバージョン番号によって異なります。 MQCD バージョン (MQCD_VERSION) について詳しくは、MQCD バージョンを参照してください。 MQCD 構造体を使用して、例えば、チャネル出口プログラムをサーバーに渡すことができます。 MQCD バージョン 3 以降を使用している場合は、その構造体を使用して、出口の配列をサーバーに渡すことができます。 この機能を使用して、既存の出口を変更するのではなく操作ごとの出口を追加することにより、同一メッセージに対して暗号化と圧縮などの複数の操作を実行できます。 MQCD 構造体の中の配列を指定しない場合は、単一出口のフィールドが検査されます。 チャネル出口プログラムの詳細については、967 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

MQCONNX の共用接続ハンドル

共有接続ハンドルを使用して、同じプロセス内の異なるスレッド間でハンドルを共有することができます。

共用接続ハンドルを指定すると、MQCONNX 呼び出しから戻された接続ハンドルを、プロセス内の任意のスレッドの後続の MQI 呼び出しに渡すことができます。

注 : IBM MQ MQI client 上で共用接続ハンドルを使用して、共用接続ハンドルをサポートしていないサーバー・キュー・マネージャーに接続することができます。

詳しくは、923 ページの『MQCONNX の使用法』 を参照してください。

IBM MQ MQI clients 用のアプリケーションの作成

IBM MQ MQI client 環境で、アプリケーションを作成して実行できます。 アプリケーションを作成し、使用する IBM MQ MQI client にリンクする必要があります。 アプリケーションを作成およびリンクする方法は、使用しているプラットフォームおよびプログラミング言語に応じて異なります。

アプリケーションがクライアント環境で実行されている場合は、アプリケーションを以下の表に示された言語で作成できます。

表 *133.* クライアント環境でサポートされているプログラミング言語

クライアント・プラットフォーム	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX AIX	はい	はい	はい			
IBM i IBM i	はい		はい		はい	
Linux Linux	はい	はい	はい			
Windows Windows	はい	はい	はい			はい

Multi *C アプリケーションと IBM MQ MQI client・コードとのリンク*

IBM MQ MQI client 上で実行する IBM MQ アプリケーションを作成した後、それを IBM MQ MQI client コードにリンクする必要があります。

アプリケーションは、次の 2 つの方法で IBM MQ MQI client・コードにリンクすることができます。

1. アプリケーションをキュー・マネージャーに接続して直接リンクする方法。この場合、キュー・マネージャーがアプリケーションと同じマシン上にある必要があります。
2. クライアント・ライブラリー・ファイルにリンクする方法。この場合ユーザーは、同じまたは異なるマシン上にあるキュー・マネージャーにアクセスできます。

IBM MQ には、次のような各環境用のクライアント・ライブラリー・ファイルがあります。

AIX **AIX**
libmqic.a ライブラリー (スレッド化されていないアプリケーション用)、または libmqic_r.a ライブラリー (スレッド化されたアプリケーション用)。

Linux **Linux**
libmqic.so ライブラリー (スレッド化されていないアプリケーション用)、または libmqic_r.so ライブラリー (スレッド化されたアプリケーション用)。

IBM i **IBM i**
クライアント・アプリケーションを、LIBMQIC クライアント・サービス・プログラム (スレッド化されていないアプリケーション用)、または LIBMQIC_R サービス・プログラム (スレッド化されたアプリケーション用) にバインドします。

Windows **Windows**
MQIC32.LIB.

ALW *C++ アプリケーションと IBM MQ MQI client・コードとのリンク*

C++ 内のクライアント上で実行するアプリケーションを作成できます。構築方法は環境によって異なります。

C++ アプリケーションのリンク方法について詳しくは、IBM MQ C++ プログラムの作成を参照してください。

C++ の使用に関するすべての詳細な説明は、C++ の使用を参照してください。

Multi *COBOL アプリケーションと IBM MQ MQI client・コードとのリンク*

IBM MQ MQI client 上で実行する COBOL アプリケーションを作成した後、そのアプリケーションを適切なライブラリーにリンクする必要があります。

IBM MQ には、次のような各環境用のクライアント・ライブラリー・ファイルがあります。

AIX
COBOL アプリケーションを、ライブラリー libmqicb.a (スレッド化されていないアプリケーション用)、または libmqicb_r.a (スレッド化されたアプリケーション用) とリンクします。

IBM i
COBOL クライアント・アプリケーションを、AMQCSTUB サービス・プログラム (スレッド化されていないアプリケーション用)、または AMQCSTUB_R サービス・プログラム (スレッド化されたアプリケーション用) にバインドします。

Windows
アプリケーション・コードを、32 ビット COBOL 用の MQICCBB ライブラリーとリンクします。 IBM MQ MQI client for Windows は、16 ビット COBOL をサポートしていません。

Windows *Visual Basic アプリケーションと IBM MQ MQI client・コードとのリンク*

Windows 上で IBM MQ MQI client コードと Microsoft Visual Basic アプリケーションをリンクできます。

Deprecated

IBM MQ 9.0 以降、Microsoft Visual Basic 6.0 に対するサポートは非推奨になりました。 IBM MQ classes for .NET は、推奨される置換テクノロジーです。 詳しくは、.NET アプリケーションの開発を参照してください。

Visual Basic アプリケーションを以下の組み込みファイルにリンクします。

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
PCF コマンド

CMQXB.bas
チャネル

Visual Basic コンパイラーでクライアントの `mqtype=2` を設定して、クライアント dll が正しく自動選択されるようにします。

MQIC32.dll
Windows 7、Windows 8、Windows 2008、および Windows 2012

関連概念

1057 ページの『Visual Basic でのコーディング』
Microsoft Visual Basic で IBM MQ プログラムをコーディングする際に考慮すべき情報。 Visual Basic は、Windows でのみサポートされます。

1025 ページの『Windows での Visual Basic プログラムの準備』
Microsoft Visual Basic プログラムを Windows で使用する場合に考慮する情報です。

IBM MQ MQI client 環境でのアプリケーションの実行

特定の条件が満たされている場合は、コードを変更せずに、完全な IBM MQ 環境と IBM MQ MQI client 環境の両方で IBM MQ アプリケーションを実行できます。

その条件は、次のとおりです。

- アプリケーションが複数のキュー・マネージャーに同時に接続する必要がない場合
- MQCONN または MQCONNX 呼び出しで、キュー・マネージャー名の先頭にアスタリスク (*) が付いていない場合
- アプリケーションは、「IBM MQ MQI client で実行されるアプリケーション」にリストされている例外のいずれかを使用する必要はありません。

注 : どの環境でアプリケーションを実行しなければならないかは、リンク・エディット時に使用するライブラリーによって決まります。

IBM MQ MQI client 環境で作業するときは、次の点に注意してください。

- IBM MQ MQI client 環境で実行されるアプリケーションごとに、サーバーに対してそれぞれ固有の接続が存在します。 アプリケーションは、MQCONN または MQCONNX 呼び出しを発行するたびにサーバーに対して接続を 1 つ確立します。
- アプリケーションは、メッセージを同期的に送受信します。 このことは、クライアント側で呼び出しを発行してからネットワークを介して完了コードおよび理由コードが戻されるまでに待ち時間があることを暗黙に示しています。
- すべてのデータ変換は、サーバーが実行します。なお、マシンで構成された CCSID のオーバーライドについては、MQCCSID も参照してください。

キュー・マネージャーへの *IBM MQ MQI client* ・アプリケーションの接続

IBM MQ MQI client 環境で実行中のアプリケーションは、さまざまな方法でキュー・マネージャーに接続することができます。 環境変数、MQCNO 構造、またはクライアント定義テーブルを使用できます。

IBM MQ クライアント環境で実行中のアプリケーションが MQCONN 呼び出しまたは MQCONNX 呼び出しを発行すると、クライアントは、アプリケーションが接続を確立する方法を識別します。 MQCONNX 呼び出しが、IBM MQ クライアントでアプリケーションから発行されると、MQI クライアント・ライブラリーは次の順序でクライアント・チャネル情報を検索します。

1. MQCNO 構造体の ClientConnOffset または ClientConnPtr フィールドの内容 (提供されている場合) を使用します。 これらのフィールドは、クライアント接続チャネルの定義として使用するチャネル定義構造体 (MQCD) を指定します。 接続詳細は、接続前出口を使用してオーバーライドできます。 詳しくは、998 ページの『リポジトリーから接続前出口を使用した接続定義の参照』を参照してください。
2. **MQSERVER** 環境変数が設定されている場合は、定義されているチャネルが使用されます。
3. mqclient.ini ファイルが定義されており、Channels スタンザに **ServerConnectionParms** 属性が含まれている場合は、そのファイルで定義されているチャネルが使用されます。 詳しくは、 IBM MQ MQI client 構成ファイル、 mqclient.ini 、およびクライアント構成ファイルの Channels スタンザを参照してください。
4. **MQCHLLIB** 環境変数および **MQCHLTAB** 環境変数が設定されている場合、それらが指すクライアント・チャネル定義テーブルが使用されます。 あるいは、 IBM MQ 9.0 以降では、 **MQCCDTURL** 環境変数は、 **MQCHLLIB** 環境変数と **MQCHLTAB** 環境変数の組み合わせを設定する機能と同等の機能を提供します。 **MQCCDTURL** が設定されている場合、それが指すクライアント・チャネル定義テーブルが使用されます。 詳しくは、 URL CCDT へのアクセスを参照してください。
5. mqclient.ini ファイルが定義されており、Channels スタンザに **ChannelDefinitionDirectory** 属性と **ChannelDefinitionFile** 属性が含まれている場合は、これらの属性を使用してクライアント・チャネル定義テーブルを見つけます。 詳しくは、 IBM MQ MQI client 構成ファイル、 mqclient.ini 、およびクライアント構成ファイルの Channels スタンザを参照してください。
6. 最後に、環境変数が設定されていない場合、クライアントは、mqs.ini ファイル内の AllQueueManagers スタンザの **DefaultPrefix** 属性から確立されたパスと名前を持つクライアント・チャネル定義テーブルを検索します。 詳しくは、 mqs.ini ファイルの AllQueueManagers スタンザを参照してください。

   クライアント・チャネル定義テーブルの検索が失敗した場合、クライアントは次のパスを使用します。

   - Linux AIX AIX and Linux 上: /var/mqm/AMQCLCHL.TAB

- Windows Windows 上: `C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\amqclchl.tab`
- IBM i IBM i 上: `/QIBM/UserData/mqm/@ipcc`
- MQ Appliance IBM MQ Appliance 上: *QMname*`_AMQCLCHL.TAB`。 `mqbackup:// URI` の下に表示されます。

以前のリストで説明された最初のオプション (MQCNO の `ClientConnOffset` または `ClientConnPtr` フィールドを使用したもの) は MQCONNX 呼び出しによってのみサポートされています。 アプリケーションが MQCONNX ではなく MQCONN を使用している場合、チャネル情報の検索は、リストに示されている順番で残りの 5 つの方法で行われます。 クライアントがチャネル情報を見つけられなかった場合、MQCONN または MQCONNX 呼び出しは失敗します。

MQCONN 呼び出しまたは MQCONNX 呼び出しが成功するためには、チャネル名 (クライアント接続用) が、サーバー側に定義されているサーバー接続チャネル名と一致していなければなりません。

関連概念
クライアント・チャネル定義テーブルへの Web アドレス指定可能アクセス
関連タスク
サーバーとクライアント間の接続の構成
関連資料
クライアント・チャネル定義テーブル
MQCNO - 接続オプション

環境変数を使用したクライアント・アプリケーションのキュー・マネージャーへの接続
クライアント・チャネル情報は、環境変数によって、クライアント環境で実行しているアプリケーションに提供できます。

IBM MQ MQI client 環境で実行中のアプリケーションは、次のさまざまな環境変数を使用してキュー・マネージャーに接続することができます。

MQSERVER
MQSERVER 環境変数は、最小チャネルを定義するために使用されます。 **MQSERVER** は、 IBM MQ サーバーのロケーションおよび使用する通信方式を指定します。

MQCHLLIB
MQCHLLIB 環境変数は、クライアント・チャネル定義テーブル (CCDT) を含むファイルへのディレクトリー・パスを指定します。 ファイルはサーバー上に作成されますが、IBM MQ MQI client・ワークステーションにコピーすることができます。

MQCHLTAB
MQCHLTAB 環境変数は、クライアント・チャネル定義テーブル (CCDT) を含むファイルの名前を指定します。

IBM MQ 9.0 以降、 **MQCCDTURL** 環境変数は、 **MQCHLLIB** 環境変数と **MQCHLTAB** 環境変数の組み合わせを設定する機能と同等の機能を提供します。 **MQCCDTURL** では、クライアント・チャネル定義テーブルを取得できる単一値として、ファイル、ftp、または http URL を指定できます。 詳しくは、クライアント・チャネル定義テーブルへの Web アドレス指定可能アクセスを参照してください。

MQCNO 構造体を使用したクライアント・アプリケーションのキュー・マネージャーへの接続
MQCONNX 呼び出しの MQCNO 構造体を使用して提供されるチャネル定義構造体 (MQCD) 内で、チャネルの定義を指定することができます。

詳しくは、 MQCNO を使用した IBM MQ MQI client でのクライアント接続チャネルの作成を参照してください。

クライアント・チャネル定義テーブルを使用したクライアント・アプリケーションのキュー・マネージャーへの接続
MQSC DEFINE CHANNEL コマンドを使用する場合、指定した詳細はクライアント・チャネル定義テーブル (CCDT) に入れられます。 MQCONN 呼び出しまたは MQCONNX 呼び出しの **QMgrName** パラメーターの内容によって、クライアントが接続されるキュー・マネージャーが判別されます。

このファイルは、アプリケーションが使用するチャネルを判別するためにクライアントがアクセスするファイルです。 適切なチャネル定義が複数ある場合、チャネルの選択には、クライアント・チャネル・ウェイト (CLNTWGHT) および接続アフィニティー (AFFINITY) のチャネル属性が関係します。

自動クライアント再接続の使用

追加のコードを作成しなくても、いくつかのコンポーネントを構成することによって、クライアント・アプリケーションが自動的に再接続するようにすることができます。

クライアントの自動再接続はインライン です。 接続はクライアント・アプリケーション・プログラムのどのポイントでも自動的に復元され、オブジェクトを開くためのハンドルがすべて復元されます。

対照的に、手動再接続では、クライアント・アプリケーションで MQCONN または MQCONNX を使用して接続を再作成し、オブジェクトを再オープンする必要があります。 クライアントの自動再接続は多くのクライアント・アプリケーションに適していますが、すべてのクライアント・アプリケーションに適しているわけではありません。

詳細については、自動クライアント再接続を参照してください。

クライアント・チャネル定義テーブルの役割

クライアント・チャネル定義テーブル (CCDT) には、クライアント接続チャネルの定義が含まれます。 これは、クライアント・アプリケーションが複数の代替キュー・マネージャーに接続する必要がある場合に特に役立ちます。

キュー・マネージャーを定義すると、クライアント・チャネル定義テーブルが作成されます。 同じファイルを、複数の IBM MQ クライアントが使用できます。

クライアント・アプリケーションが CCDT を使用する方法はいくつかあります。 CCDT は、クライアント・コンピューターにコピーできます。 複数のクライアントが共有する位置に、CCDT をコピーすることができます。 CCDT がサーバー上に置かれたまま、共有ファイルとしてクライアントからアクセス可能にすることができます。

IBM MQ 9.0 以降、URI を介してアクセスできる中央の場所で CCDT のホスティングが可能になり、デプロイされた各クライアントで個別に CCDT を更新する必要がなくなりました。

関連概念

クライアント・チャネル定義テーブルへの Web アドレス指定可能アクセス

関連タスク

クライアント接続チャネル定義へのアクセス

関連資料

クライアント・チャネル定義テーブル

CCDT のキュー・マネージャー・グループ

クライアント・チャネル定義テーブル (CCDT) で接続のセットをキュー・マネージャー・グループ として定義できます。 キュー・マネージャー・グループの一部であるキュー・マネージャーにアプリケーションを接続することができます。 これは、MQCONN または MQCONNX の呼び出し時に、 接頭部にアスタリスク付きのキュー・マネージャー名を使用することによって実行できます。

以下の理由から、複数のサーバー・マシンへの接続の定義を選択することがあります。

- 可用性を高めるために、実行中の一連のキュー・マネージャーのいずれかにクライアントを接続したい。
- 前回、正常に接続されたものと同じキュー・マネージャーにクライアントを再接続したいが、 接続に失敗した場合は別のキュー・マネージャーに接続したい。
- 接続に失敗した場合は、再度クライアント・プログラムで MQCONN を発行して、別のキュー・マネージャーにクライアント接続を再試行できるようにしたい。
- 接続に失敗した場合は、クライアント・コードを書かなくても、別のキュー・マネージャーにクライアントを自動的に再接続したい。
- スタンバイ・インスタンスを引き継ぐ場合は、クライアント・コードを書かなくても、 複数インスタンスのキュー・マネージャーの異なるインスタンスにクライアントを自動的に再接続したい。
- 一部のキュー・マネージャーには、他のキュー・マネージャーより多くのクライアントが接続されるように、多数のキュー・マネージャー全体のクライアント接続のバランスを取りたい。

- 接続数が多いことによって障害が発生する場合は、 多数のクライアントの再接続を複数のキュー・マネージャーに時間の経過に従って分散させたい。
- クライアント・アプリケーション・コードを変更せずにキュー・マネージャーを移動できるようにしたい。
- キュー・マネージャー名を認識する必要のないクライアント・アプリケーション・プログラムを作成したい。

別のキュー・マネージャーに接続することは、いつも適切であるというわけではありません。 例えば、拡張トランザクション・クライアントまたは WebSphere Application Server の Java クライアントは、予測可能なキュー・マネージャー・インスタンスに接続する必要がある場合があります。 IBM MQ classes for Java は自動クライアント再接続をサポートしていません。

キュー・マネージャー・グループは、クライアント・チャネル定義テーブル (CCDT) で定義されている接続セットです。 このセットは、それぞれのチャネル定義内に同じ値の **QMNAME** 属性を持つメンバーによって定義されます。

930 ページの図 97 は、クライアント接続テーブルを図形で表したものです。 これには、3 つのキュー・マネージャー・グループが示され、 そのうちの 2 つは、**QMNAME** (QM1) および **QMNAME** (QMGrp1) という名前で CCDT に記述されるキュー・マネージャー・グループで、 1 つは **QMNAME** (' ') で記述されるブランク (デフォルト・グループ) です。

1. キュー・マネージャー・グループ QM1 には 3 つのクライアント接続チャネルがあり、キュー・マネージャー QM1 および QM2 に接続されています。 QM1 は、2 つの異なるサーバーにある複数インスタンス・キュー・マネージャーである場合があります。
2. デフォルトのキュー・マネージャー・グループには、6 つのクライアント接続チャネルがあり、すべてのキュー・マネージャーに接続されています。
3. QMGrp1 には、QM4 および QM5 という 2 つのキュー・マネージャーへのクライアント接続チャネルがあります。

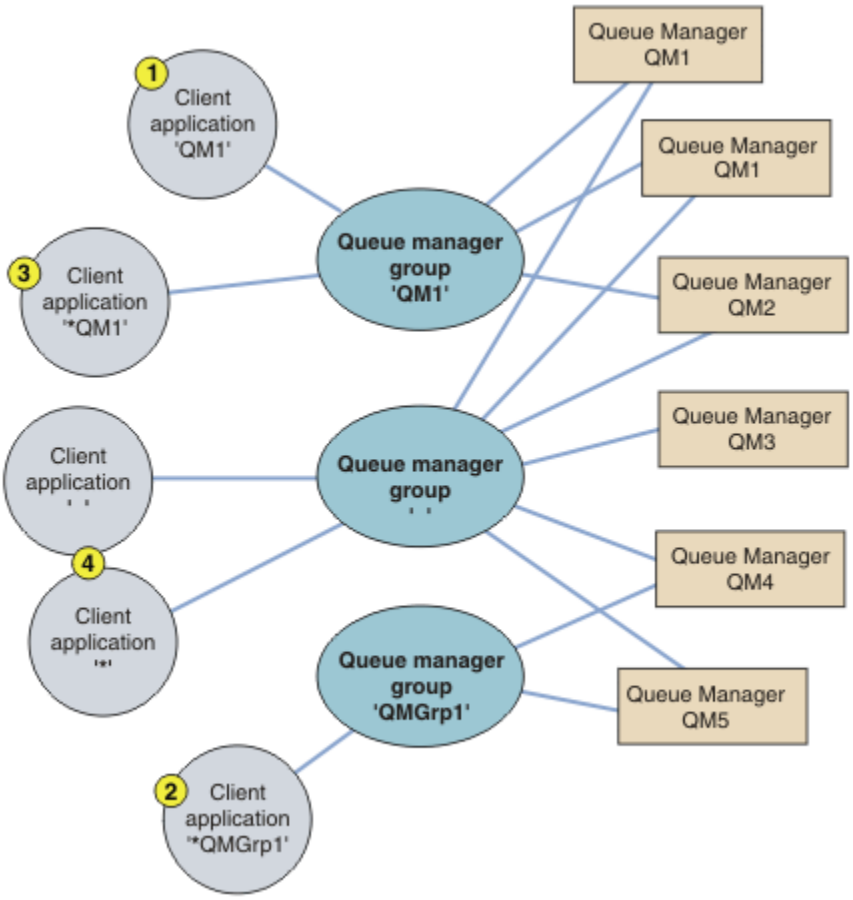


図 97. キュー・マネージャー・ グループ

このクライアント接続テーブルの 4 つの使用例について、930 ページの図 97 で番号を付けたクライアント・アプリケーションを用いて説明します。

1. 1 つ目の例では、クライアント・アプリケーションが、キュー・マネージャー名 QM1 を **QmgrName** パラメーターとして、MQCONN または MQCONNX MQI 呼び出しに渡します。 IBM MQ クライアント・コードによって、一致するキュー・マネージャー・グループ QM1 が選択されます。 グループには 3 つの接続チャネルがあり、IBM MQ MQI client はこれらのチャネルをそれぞれ順に使用して、QM1 と呼ばれる実行中のキュー・マネージャーへの接続用の IBM MQ リスナーを検出するまで、QM1 への接続を試行します。

   接続の試行順序は、クライアント接続の AFFINITY 属性の値およびクライアント・チャネルの重み付けによって決まります。 これらの制約では、可能性のある 3 つの接続と経過時間の両方を考慮して、 接続の試行順序がランダムに選択され、接続の負荷が分散されます。

   QM1 の実行中のインスタンスへの接続が確立されれば、 クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

2. 2 つ目の例では、クライアント・アプリケーションが、**QmgrName** パラメーターとして先頭にアスタリスクが付いたキュー・マネージャー名 *QMGrp1 を、MQCONN または MQCONNX MQI 呼び出しに渡します。 IBM MQ クライアントによって、一致するキュー・マネージャー・グループ QMGrp1 が選択されます。 このグループには 2 つのクライアント接続チャネルがあり、IBM MQ MQI client は各チャネルを順に使用して任意の キュー・マネージャーへの接続を試みます。 この例では、IBM MQ MQI client は正常な接続を確立する必要があります。接続先のキュー・マネージャーの名前は関係ありません。

   接続の試行順序についての規則は 1 つ目のケースと同じです。 唯一異なる点は、キュー・マネージャー名の先頭にアスタリスクが付いていることで、クライアントは、キュー・マネージャーの名前が関連していないことを示します。

QMGrp1 キュー・マネージャー・グループ内のチャネルを経由して接続された任意のキュー・マネージャーの実行中のインスタンスへの接続が確立されれば、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

3. 3 つ目の例は、**QmgrName** パラメーターの先頭にアスタリスクが付けられて *QM1 となっているため、基本的には 2 つ目の例と同じです。 この例は、クライアント・チャネル接続での接続先キュー・マネージャーを、1 つのチャネル定義の QMNAME 属性を単独で検査することでは決定できないことを示しています。 チャネル定義の **QMNAME** 属性が QM1 であっても、QM1 と呼ばれるキュー・マネージャーへの接続が必要であるとは限りません。 クライアント・アプリケーションの **QmgrName** パラメーターの先頭にアスタリスクが付いている場合、いずれのキュー・マネージャーも接続先にすることができます。

   このケースでは、QM1 または QM2 のいずれかの実行中のインスタンスへの接続が確立された場合に、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しが成功します。

4. 4 つ目の例は、デフォルト・グループの使用を示しています。 このケースでは、クライアント・アプリケーションが、**QmgrName** パラメーターとしてアスタリスク '*' またはブランク ' ' を、MQCONN または MQCONNX MQI 呼び出しに渡します。 クライアント・チャネル定義の規則により、ブランクの **QMNAME** 属性はデフォルトのキュー・マネージャー・グループを表し、ブランクまたはアスタリスクの付いた **QmgrName** パラメーターのいずれかが、ブランクの **QMNAME** 属性と一致します。

   この例では、デフォルトのキュー・マネージャー・グループに、すべてのキュー・マネージャーへのクライアント・チャネル接続があります。 デフォルトのキュー・マネージャー・グループを選択することにより、アプリケーションをグループ内のどのキュー・マネージャーにも接続できます。

   任意のキュー・マネージャーの実行中のインスタンスへの接続が確立された場合に、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

   注 : アプリケーションはブランクの **QmgrName** パラメーターを使用して、デフォルトのキュー・マネージャー・グループまたはデフォルトのキュー・マネージャーのいずれかに接続しますが、デフォルト・グループはデフォルトのキュー・マネージャーとは異なります。 デフォルトのキュー・マネージャー・グループの概念は、クライアント・アプリケーションにのみ関係しますが、デフォルトのキュー・マネージャーの概念はサーバー・アプリケーションに関係します。

2 番目または 3 番目のキュー・マネージャーに接続するチャネルも含め、クライアント接続チャネルは 1 つのキュー・マネージャーのみで定義してください。 これらのチャネルを 2 つのキュー・マネージャーで定義してから、2 つのクライアント・チャネル定義テーブルをマージしないようにしてください。 クライアントがアクセスできるのは、1 つのクライアント・チャネル定義テーブルのみです。

例

トピックの冒頭にある、キュー・マネージャー・グループを使用する理由を示すリストをもう一度見てください。 キュー・マネージャー・グループを使用することで、以下の機能がどのように提供されるのでしょうか。

キュー・マネージャー・セット内のいずれかのキュー・マネージャーに接続する。
セット内のすべてのキュー・マネージャーへの接続を使用してキュー・マネージャー・グループを定義し、先頭にアスタリスクの付いた **QmgrName** パラメーターを使用してグループに接続します。

同じキュー・マネージャーに再接続するが、最後に接続したキュー・マネージャーが使用不可の場合は、別のキュー・マネージャーに接続する。
前述のようにキュー・マネージャー・グループを定義しますが、各クライアント・チャネル定義で属性 **AFFINITY** (PREFERRED) を設定します。

接続に失敗した場合は、別のキュー・マネージャーへの接続を再試行する。
キュー・マネージャー・グループに接続して、接続が切れた場合、またはキュー・マネージャーで障害が発生した場合は、MQCONN または MQCONNX MQI 呼び出しを再発行します。

接続に失敗した場合は、別のキュー・マネージャーに自動的に再接続する。
MQCONNX **MQCNO** オプションの MQCNO_RECONNECT を使用してキュー・マネージャー・グループに接続します。

複数インスタンスのキュー・マネージャーの異なるインスタンスに自動的に再接続する。
前述の例と同じ操作を行います。 このケースでは、特定の複数インスタンス・キュー・マネージャーのインスタンスに接続するようにキュー・マネージャー・グループを制限する場合、 複数インスタンス・キュー・マネージャーのインスタンスのみへの接続を使用してグループを定義します。

クライアント・アプリケーションに、先頭にアスタリスクが付いていない **QmgrName** パラメーターを使用して、MQCONN または MQCONNX MQI 呼び出しを発行するように要求することもできます。 これが、クライアント・アプリケーションが名前の付いたキュー・マネージャーに接続できる唯一の方法です。 最後に、**MQCNO** オプションを MQCNO_RECONNECT_Q_MGR に設定できます。 このオプションでは、以前接続したものと同じキュー・マネージャーへの再接続を受け入れます。 この値を使用して、通常キュー・マネージャーの同じインスタンスへの再接続を制限することもできます。

一部のキュー・マネージャーには、他のキュー・マネージャーより多くのクライアントが接続されるように、キュー・マネージャー全体のクライアント接続のバランスを取る。
キュー・マネージャー・グループを定義し、接続を不均等に分散するように各クライアント・チャネル定義で **CLNTWGHT** 属性を設定します。

接続に失敗した場合またはキュー・マネージャーに障害が発生した場合、クライアント再接続の負荷を不均等に、時間の経過に従って分散する。
前述の例と同じ操作を行います。 IBM MQ MQI client はキュー・マネージャー全体で再接続をランダムに行い、時間の経過に従って再接続を分散します。

クライアント・コードを変更しないでキュー・マネージャーを移動する。
CCDT で、クライアント・アプリケーションをキュー・マネージャーの位置から分離させます。 CCDT は、クライアントで定義できるデータ・ファイルであり、共用ロケーションから読み取ったり、Web サーバーから取り出したりすることができます。 詳しくは、クライアント・チャネル定義テーブルを参照してください。

キュー・マネージャー名を認識しないクライアント・アプリケーションを作成する。
キュー・マネージャー・グループ名を使用して、組織内のクライアント・アプリケーションに関連し、かつ、キュー・マネージャーの名前ではなくソリューション・アーキテクチャーを反映するような、キュー・マネージャー・グループ名の命名規則を設定します。

z/OS キュー共用グループへの接続

キュー共用グループの一部であるキュー・マネージャーに、ご使用のアプリケーションを接続することができます。 これは、MQCONN 呼び出し時または MQCONNX 呼び出し時に、キュー・マネージャー名ではなくキュー共用グループ名を使用することによって実行できます。

キュー共用グループには、最大 4 文字の名前があります。 この名前はネットワーク内で固有であり、かつ、キュー・マネージャー名とは異なるものである必要があります。

クライアント・チャネル定義では、グループ内の使用可能なキュー・マネージャーに接続するときは、キュー共用グループの汎用インターフェースを使用する必要があります。 詳細については、キュー共用グループへのクライアントの接続を参照してください。 リスナーが接続するキュー・マネージャーが、キュー共用グループのメンバーであることを確認する検査が行われます。

共用キューについての詳細は、「共用キューおよびキュー共用グループ」を参照してください。

チャネルの加重とアフィニティーの例

この例では、ゼロ以外の ClientChannelWeights が使用された場合に、どのようにクライアント接続チャネルが選択されるかを示します。

ClientChannelWeight チャネル属性および ConnectionAffinity チャネル属性は、1 つの接続に関して使用できる適切なチャネルが複数存在する場合に、クライアント接続チャネルを選択する方法を制御します。 これらのチャネルは、高可用性またはワークロード・バランシング、あるいはその両方を提供するために、異なるキュー・マネージャーに接続するように構成されています。 複数のキュー・マネージャーのいずれかへの接続を引き起こす可能性のある MQCONN 呼び出しには、キュー・マネージャー名の前に、以下の場所で説明されているようにアスタリスクを付ける必要があります。 MQCONN 呼び出しの例: 1 の例。 キュー・マネージャー名にアスタリスク (*) が含まれます。

接続するのに適切なチャネルの候補は、QMNAME 属性が MQCONN 呼び出しで指定されるキュー・マネージャー名に一致するチャネルです。 接続に適用可能なすべてのチャネルで、ClientChannelWeight がゼ

ロ (デフォルト) の場合は、以下の例のようにアルファベット順で選択されます。 MQCONN 呼び出しの例: 1 の例。 キュー・マネージャー名にアスタリスク (*) が含まれます。

ゼロでない ClientChannelWeights が使用される場合については、以下の例で説明します。 このフィーチャーでは疑似ランダムなチャネル選択が行われるため、これらの例はアクションの厳密な順番ではなく、実行される可能性のある順番を示していることに注意してください。

例 *1. ConnectionAffinity* が *PREFERRED* に設定されている場合のチャネルの選択
この例では、ConnectionAffinity が PREFERRED に設定されている場合に、IBM MQ MQI client が CCDT からチャネルを選択する方法を示します。

この例では、複数のクライアント・マシンが、キュー・マネージャーにより提供されたクライアント・チャネル定義テーブル (CCDT) を使用しています。 CCDT には、次の属性を持つクライアント接続チャネルが含まれます (DEFINE CHANNEL コマンドの構文を使用して示します)。

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

アプリケーションは MQCONN(*CORE) を発行します。

QMNAME 属性が一致しないため、チャネル A はこの接続の候補にはなりません。 チャネル B、C、および D は候補として識別され、それぞれの加重に基づいた優先順序で並んでいます。 この例では、順序は C、B、D となります。 クライアントは、core2.ops.company.example でキュー・マネージャーへの接続を試みます。 このアドレスではキュー・マネージャーの名前は検査されません。これは、MQCONN 呼び出しで、キュー・マネージャー名にアスタリスクが含まれていたためです。

AFFINITY(PREFERRED) の場合、この特定のクライアント・マシンは接続のたびに、常に初期の優先順序でチャネルを並べることにご注意ください。 これは、接続が別のプロセスから、または別の時点で実行された場合であっても同じです。

この例では、core.2.ops.company.example のキュー・マネージャーには到達できません。 チャネル B が優先順位で次の順序にあるため、クライアントは core1.ops.company.example への接続を試行します。 さらに、チャネル C がデモートされ、優先順位の最下位になります。

2 番目の MQCONN(*CORE) 呼び出しが、同じアプリケーションから発行されます。 チャネル C は以前の接続によって降格されたため、現在最も優先されるチャネルは B です。 この接続は core1.ops.company.example に対して行われます。

同じクライアント・チャネル定義テーブルを共有する 2 番目のマシンでは、異なる初期の優先順序でチャネルを並べる場合があります。 例えば、D、B、C などです。 すべてのチャネルが動作する状態にある通常の環境では、このマシン上のアプリケーションは core3.ops.company.example に接続され、最初のマシン上のアプリケーションは core2.ops.company.example に接続されます。 これにより、複数のキュー・マネージャーを対象とした多数のクライアント間のワークロード・バランシングが可能になる一方で、個々のクライアントが同じキュー・マネージャー (使用可能な場合) に接続することができます。

例 *2. ConnectionAffinity* が *NONE* に設定されている場合のチャネルの選択
この例では、ConnectionAffinity が NONE に設定されている場合に、IBM MQ MQI client が CCDT からチャネルを選択する方法を示します。

この例では、複数のクライアントが、キュー・マネージャーにより提供されたクライアント・チャネル定義テーブル (CCDT) を使用しています。 CCDT には、次の属性を持つクライアント接続チャネルが含まれます (DEFINE CHANNEL コマンドの構文を使用して示します)。

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
```

```
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

アプリケーションは MQCONN(*CORE) を発行します。 前の例と同様、QMNAME が一致しないのでチャネル A は検討の対象にはなりません。 チャネル B、C、D はそれぞれの加重に基づいて選択されます。選択される確率はそれぞれ、50%、30%、20% です。 この例では、チャネル B が選択されると考えられます。作成される優先順位は永続的ではありません。

2 番目の MQCONN(*CORE) 呼び出しが実行されます。 この場合も、3 つの候補のチャネルのうちの 1 つが、前と同じ確率で選択されます。 この例では、チャネル C が選択されます。 しかし、core2.ops.company.example が応答しないため、残りの候補のチャネルの中から別のチャネルが選択されます。 チャネル B が選択され、アプリケーションが core1.ops.company.example に接続されます。

AFFINITY(NONE) が指定される場合、各 MQCONN 呼び出しは互いに独立しています。 したがって、このサンプル・アプリケーションが 3 番目の MQCONN(*CORE) 呼び出しを行う場合、切断されたチャネル C による接続をもう一度試行してから、B または D のいずれかを選択することになります。

MQCONN 呼び出しの例
MQCONN を使用して、特定のキュー・マネージャー、または複数のキュー・マネージャーのうちの 1 つに接続する例です。

次のそれぞれの例では、ネットワークは同じであり、同じ IBM MQ MQI client から 2 つのサーバーに定義された接続があります。 (以下の例では、MQCONN 呼び出しの代わりに MQCONNX 呼び出しを使用できます。)

サーバー・マシン上で 2 つのキュー・マネージャーが稼働中であり、一方は SALE、他方は SALE_BACKUP という名前です。

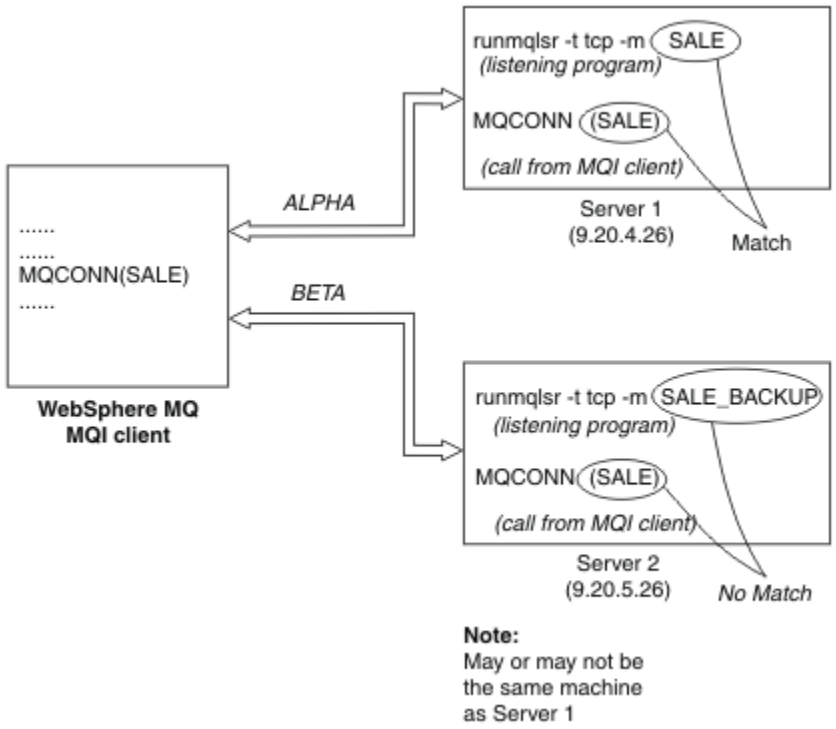


図 *98. MQCONN* の例

これらの例のチャネルについての定義は、次のとおりです。

SALE の定義:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP の定義:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

クライアント・チャネル定義は次のように要約できます。

名前	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

MQCONN の例で示される内容
この例では、バックアップ・システムとして複数のキュー・マネージャーを使用する例を示します。

サーバー 1 との通信リンクが一時的に切断されたとします。 バックアップ・システムとして複数のキュー・マネージャーを使用する例を示します。

それぞれの例で異なる MQCONN 呼び出しが使用され、次の規則を適用して、特定の例で何が起こっているかを説明します。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、MQCONN 呼び出しで指定された名前に対応するキュー・マネージャー名 (QMNAME フィールド) が検索されます。
2. 一致するものが見つかると、そのチャネル定義が使用されます。
3. 接続名 (CONNAME) で識別されたマシンに対するチャネルの開始が試行されます。 この試行が成功すると、アプリケーションは処理を続行します。 アプリケーションには、次のものが必要です。
   - サーバー上で稼働するリスナー
   - クライアントが接続しようとするキュー・マネージャーと同じキュー・マネージャーに接続されるリスナー (指定されている場合)
4. チャネルの開始の試行に失敗し、クライアント・チャネル定義テーブルに複数のエントリーがある (この例では 2 つのエントリーがある) 場合は、一致するものを求めてファイルがさらに検索されます。 一致するものが見つかると、ステップ 1 から処理が続行されます。
5. 一致するエントリーが見つからない場合、またはクライアント・チャネル定義テーブルのエントリーがそれ以外にないためチャネルが開始できなかった場合、アプリケーションは接続できません。 該当する理由コードおよび完了コードが MQCONN 呼び出しで戻されます。 アプリケーションでは、戻された理由コードおよび完了コードに基づいた処置を行うことができます。

例 *1.* キュー・マネージャー名にアスタリスク *(*)* が含まれる
この例では、アプリケーションがどのキュー・マネージャーに接続するかは関係ありません。 アプリケーションは、アスタリスクを含むキュー・マネージャー名に対して MQCONN 呼び出しを発行します。 適切なチャネルが選択されます。

アプリケーションは、次の呼び出しを発行します。

```
MQCONN (*SALE)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) で、アプリケーションの MQCONN 呼び出しに対応するキュー・マネージャー名 SALE が検索されます。
2. ALPHA および BETA のチャネル定義が見つかります。
3. いずれかのチャネルの CLNTWGHT 値が 0 の場合、そのチャネルが選択されます。 両方のチャネルの CLNTWGHT 値が 0 の場合、アルファベット順で先にあるチャネル ALPHA が選択されます。 いずれのチャネルの CLNTWGHT 値もゼロでない場合、その加重に基づいてどちらか一方のチャネルがランダムに選択されます。
4. チャネルの開始が試行されます。
5. チャネル BETA が選択された場合、その開始の試行は成功します。
6. チャネル ALPHA が選択された場合、通信リンクが切断されているため開始の試行は成功しません。 この場合は、以下のステップが適用されます。
   a. キュー・マネージャー名 SALE のその他の唯一のチャネルは BETA です。
   b. このチャネルの開始が試行されます。これは成功します。
7. リスナーが稼働しているかどうかの検査の結果、稼働しているリスナーが 1 つあることがわかります。これは SALE キュー・マネージャーに接続されていませんが、MQI 呼び出しのパラメーターにアスタリスク (*) が含まれているため、接続についての検査は行われません。 アプリケーションは SALE_BACKUP キュー・マネージャーに接続されて処理を続行します。

例 *2.* キュー・マネージャー名が指定されています
この例では、アプリケーションは特定のキュー・マネージャーに接続する必要があります。 アプリケーションは、そのキュー・マネージャー名に対して MQCONN 呼び出しを発行します。 適切なチャネルが選択されます。

次の MQI 呼び出しに示されているように、アプリケーションは SALE という名前の特定のキュー・マネージャーへの接続を要求します。

```
MQCONN (SALE)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、アプリケーションの MQCONN 呼び出しに対応するキュー・マネージャー名 SALE が検索されます。
2. 最初に見つかった一致するチャネル定義は ALPHA です。
3. チャネルの開始が試行されますが、通信リンクが切断されているため成功しません。
4. クライアント・チャネル定義テーブルが再び走査され、キュー・マネージャー名 SALE を検索すると、チャネル名 BETA が見つかります。
5. チャネルの試行が開始されます。これは成功します。
6. リスナーが稼働しているかどうかの検査の結果、稼働しているリスナーが 1 つあることがわかりますが、それは SALE キュー・マネージャーに接続されていません。
7. クライアント・チャネル定義テーブルには、これ以上エントリーがありません。 アプリケーションは続行できず、戻りコード MQRC_Q_MGR_NOT_AVAILABLE を受け取ります。

例 *3.* キュー・マネージャー名はブランクまたはアスタリスク *(*)* です
この例では、アプリケーションがどのキュー・マネージャーに接続するかは関係ありません。 アプリケーションは、キュー・マネージャー名にブランクのまたはアスタリスクを指定して、MQCONN を発行します。 適切なチャネルが選択されます。

これは、935 ページの『例 1. キュー・マネージャー名にアスタリスク (*) が含まれる』の場合と同じ方法で処理されます。

注 : このアプリケーションが IBM MQ MQI client 以外の環境で実行されていた場合、名前にブランクが指定されると、アプリケーションはデフォルトのキュー・マネージャーとの接続を試行します。 これは、アプリケーションがクライアント環境から実行されている場合には当てはまりません。アクセスされるキュー・マネージャーは、チャネルの接続先のリスナーと関連付けられているものです。

アプリケーションは、次の呼び出しを発行します。

```
MQCONN ("")
```

または

```
MQCONN (*)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、アプリケーションの MQCONN 呼び出しに対応するブランクのキュー・マネージャー名が検索されます。
2. チャネル名 ALPHA のエントリーには、SALE という定義内にキュー・マネージャー名があります。 これは、MQCONN 呼び出しのパラメーターと一致しません。ここでは、キュー・マネージャー名がブランクである必要があります。
3. 次のエントリーは、チャネル名 BETA のエントリーです。
4. 定義内の queue manager name は SALE です。 これも、キュー・マネージャー名がブランクである必要がある MQCONN 呼び出しのパラメーターとは一致しません。
5. クライアント・チャネル定義テーブルには、これ以上エントリーがありません。 アプリケーションは続行できず、戻りコード MQRC_Q_MGR_NOT_AVAILABLE を受け取ります。

クライアント環境でのトリガー操作

IBM MQ MQI clients で実行されている IBM MQ アプリケーションによって送信されるメッセージは、他のメッセージとまったく同じ方法でトリガー操作に寄与します。これらのメッセージは、サーバーとクライアントの両方でプログラムをトリガーするために使用できます。

トリガー操作については、チャネルのトリガー操作を参照してください。

トリガー・モニターと開始されるアプリケーションは、同じシステム上になければなりません。

クライアント環境におけるトリガーされたキューのデフォルトの特性は、サーバー環境のトリガーされたキューと同じです。 具体的には、z/OS キュー・マネージャーに対してローカルであるトリガーされたキューにメッセージを書き込むクライアント・アプリケーションに MQPMO 同期点制御オプションが指定されていない場合、メッセージは作業単位内に書き込まれます。 この場合は、トリガー条件が満たされても、トリガー・メッセージは同じ作業単位内の開始キューに書き込まれるため、トリガー・モニターはこの作業単位が終了するまでトリガー・メッセージを取り出せません。 この作業単位が終わるまで、トリガー操作のプロセスは起動されないことになります。

プロセス定義

トリガー設定がオンになっているキューに関連付けられているため、サーバー上でプロセス定義を定義する必要があります。

プロセス・オブジェクトは起動する必要があるものを定義します。 クライアントおよびサーバーが同じプラットフォーム上で実行されていない場合、トリガー・モニターが開始したプロセスは *ApplType* を定義しなければなりません。定義していないと、サーバーはデフォルトの定義 (つまり、通常サーバー・マシンに関連しているアプリケーションのタイプ) を選択し、障害を引き起こします。

例えば、トリガー・モニターが IBM MQ MQI client 上で稼働していて、他のオペレーティング・システム上のサーバーに要求を送信する場合は、MQAT_WINDOWS_NT が定義されている必要があります。定義されていないと、他のオペレーティング・システムでそのデフォルト定義が使用され、プロセスが失敗します。

トリガー・モニター

z/OS 用以外の IBM MQ 製品によって提供されるトリガー・モニターは、IBM i IBM i および AIX, Linux, and Windows・システム用のクライアント環境で稼働します。

トリガー・モニターを実行するには、次のいずれかのコマンドを発行します。

- IBM i On IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW AIX, Linux, and Windows プラットフォームの場合:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

デフォルトの開始キューは、デフォルトのキュー・マネージャーにある SYSTEM.DEFAULT.INITIATION.QUEUE です。 開始キューは、トリガー・モニターがトリガー・メッセージを検索する場所です。 次に、プログラムを呼び出して、適切なトリガー・メッセージを探します。 このトリガー・モニターは、デフォルトのアプリケーション・タイプをサポートします。また、クライアント・ライブラリーにリンクするという点を除くと、このトリガー・モニターは runmqtrm と同じです。

トリガー・モニターが作成するコマンド・ストリングは次のとおりです。

1. 関連プロセス定義の *ApplicId*。 *ApplicId* は実行するプログラムの名前です。この名前は、コマンド行に入力されたとおりに表示されます。
2. 開始キューから取得され、引用符で囲まれた MQTMC2 構造体。 引用符で囲まれたこのストリングと正確に同じストリングを持つコマンド・ストリングが開始されます。したがって、システム・コマンドには 1 つのパラメーターとして受け付けられます。
3. 関連プロセス定義の *EnvrData*。

トリガー・モニターは、開始したアプリケーションが完了するまでは、開始キューに別のメッセージがあるかどうかを検索しません。 アプリケーションに多くの処理がある場合は、到着するトリガー・メッセージの数にトリガー・モニターが追いつかないことがあります。 この状態に対応するには、次の 2 つの方法があります。

1. 実行するトリガー・モニターを増やす

   実行するトリガー・モニターの数を増やす場合、一度に実行できるアプリケーションの最大数を制御できます。

2. 開始済みのアプリケーションのバックグラウンドで実行する

   アプリケーションをバックグラウンドで実行する場合、IBM MQ では実行できるアプリケーションの数を制限しません。

AIX and Linux システムで、開始されたアプリケーションをバックグラウンドで実行するには、プロセス定義の *EnvrData* の末尾に & (アンパーサンド) を付ける必要があります。

CICS アプリケーション (z/OS 以外)
z/OS 以外の CICS アプリケーション・プログラムで MQCONN 呼び出し、または MQCONNX 呼び出しを発行する場合は、そのアプリケーション・プログラムが CEDA に RESIDENT として定義されている必要があります。 CICS サーバー・アプリケーションをクライアントとして再リンクすると、同期点サポートが失われるおそれがあります。

z/OS 以外の CICS アプリケーション・プログラムで MQCONN 呼び出し、または MQCONNX 呼び出しを発行する場合は、そのアプリケーション・プログラムが CEDA に RESIDENT として定義されている必要があります。 常駐コードをできるだけ小さくするために、別のプログラムにリンクして MQCONN 呼び出し、または MQCONNX 呼び出しを発行することができます。

MQSERVER 環境変数を使用してクライアント接続を定義する場合は、その変数を CICSENV.CMD ファイルの中で指定する必要があります。

IBM MQ アプリケーションは、IBM MQ サーバー環境で、または IBM MQ クライアント上で、コードを変更せずに実行できます。 ただし、 IBM MQ サーバー環境では、 CICS は同期点コーディネーターとして機能することができ、 **MQCMIT** および **MQBACK** ではなく、EXEC CICS SYNCPOINT および EXEC CICS SYNCPOINT ROLLBACK を使用します。 CICS アプリケーションが単純にクライアントとして再リンクされると、同期点サポートが失われます。 IBM MQ MQI client 上で実行されるアプリケーションには、**MQCMIT** および **MQBACK** を使用する必要があります。

ALW CICS および Tuxedo アプリケーションの準備と実行

CICS および Tuxedo アプリケーションをクライアント・アプリケーションとして実行するには、サーバー・アプリケーションで使用しているのとは異なるライブラリーを使用します。 アプリケーションの実行に使用されるユーザー ID も異なります。

IBM MQ MQI client アプリケーションとして実行するように CICS および Tuxedo アプリケーションを準備するには、 拡張トランザクション・クライアントの構成の説明に従ってください。

ただし、 IBM MQ で提供されるサンプル・プログラムを含め、特に CICS および Tuxedo アプリケーションの準備に関する情報は、 IBM MQ サーバー・システムで実行するアプリケーションを準備することを前提としていることに注意してください。 その結果、その情報は、サーバー・システム上で使用するための IBM MQ ライブラリーのみを参照しています。 クライアント・アプリケーションを準備するときには、次の手順を実行する必要があります。

- ご使用のアプリケーションが使用する言語バインディングに適したクライアント・システム・ライブラリーを使用する。 以下に例を示します。
  - Linux AIX AIX and Linux 上で C 言語で作成されたアプリケーションの場合、ライブラリー libmqm の代わりに libmqic を使用します。
  - Windows Windows システムでは、ライブラリー mqm.lib の代わりに mqic.lib を使用します。
- 939 ページの表 134 と 939 ページの表 135 に示されているサーバー・システム・ライブラリーではなく、それに相当するクライアント・システム・ライブラリーを使用する。 サーバー・システム・ライブラリーがこれらの表にリストされていない場合は、クライアント・システムで同じライブラリーを使用してください。

表 *134. AIX and Linux* でのクライアント・システム・ライブラリー

IBM MQ サーバー・システムのライブラリー	**IBM MQ クライアント・システム上で使用するための同等なライブラリー**
libmqmxa	libmqcxa

表 *135. Windows* システムでのクライアント・システム・ライブラリー

IBM MQ サーバー・システムのライブラリー	**IBM MQ クライアント・システム上で使用するための同等なライブラリー**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

クライアント・アプリケーションで使用されるユーザー ID

CICS の下で IBM MQ サーバー・アプリケーションを実行すると、通常は CICS ユーザーからトランザクションのユーザー ID に切り替わります。 ただし、 CICS の下で IBM MQ MQI client アプリケーションを実行する場合は、 CICS 特権権限が保持されます。

ALW *CICS と Tuxedo のサンプル・プログラム*

AIX, Linux, and Windows システムで使用する CICS および Tuxedo のサンプル・プログラム。

940 ページの表 136 は、 AIX and Linux クライアント・システムで使用するために提供されている CICS および Tuxedo のサンプル・プログラムをリストしています。 940 ページの表 137 では、Windows クライアント・システム用の同様の情報をリストしています。 これらの表では、プログラムの準備と実行に使用されるファイルもリストしています。 サンプル・プログラムの説明については、1084 ページの『CICS トランザクション・サンプル』および 1129 ページの『AIX, Linux, and Windows での TUXEDO サンプルの使用』を参照してください。

表 136. *AIX and Linux* クライアント・システム用のサンプル・プログラム		
説明	**Source**	**実行可能モジュール**
CICS プログラム	amqscic0.ccs	amqscicc
CICS プログラム用のヘッダー・ファイル	amqscih0.h	-
メッセージを書き込むための Tuxedo クライアント・プログラム	amqstxpx.c	-
メッセージを取得するための Tuxedo クライアント・プログラム	amqstxgx.c	-
2 つのクライアント・プログラム用の Tuxedo サーバー・プログラム	amqstxsx.c	-
Tuxedo プログラム用の UBBCONFIG ファイル	ubbstxcx.cfg	-
Tuxedo プログラム用のフィールド・テーブル・ファイル	amqstxvx.flds	-
Tuxedo プログラム用のビュー記述ファイル	amqstxvx.v	-

表 137. *Windows* クライアント・システム用のサンプル・プログラム		
説明	**Source**	**実行可能モジュール**
CICS トランザクション	amqscic0.ccs	amqscicc
CICS トランザクション用のヘッダー・ファイル	amqscih0.h	-
メッセージを書き込むための Tuxedo クライアント・プログラム	amqstxpx.c	-
メッセージを取得するための Tuxedo クライアント・プログラム	amqstxgx.c	-
2 つのクライアント・プログラム用の Tuxedo サーバー・プログラム	amqstxsx.c	-
Tuxedo プログラム用の UBBCONFIG ファイル	ubbstxcx.cfg	-
Tuxedo プログラム用のフィールド・テーブル・ファイル	amqstxvx.fld	-
Tuxedo プログラム用のビュー記述ファイル	amqstxvx.v	-
Tuxedo プログラム用の MAKE ファイル	amqstxmc.mak	-
Tuxedo プログラム用の ENVFILE ファイル	amqstxen.env	-

ALW エラー・メッセージ *AMQ5203 (CICS および Tuxedo アプリケーション用に変更)*

拡張トランザクション・クライアントを使用する CICS アプリケーションまたは Tuxedo アプリケーションを実行すると、標準診断メッセージが表示されることがあります。 これらのメッセージの 1 つが、拡張トランザクション・クライアントで使用するために変更されています。

IBM MQ エラー・ログ・ファイルに示される可能性のあるメッセージは、診断メッセージ: AMQ4000-9999 に記載されています。 メッセージ AMQ5203 は、拡張トランザクション・クライアントでの使用に合わせて変更されています。 変更されたメッセージの本文は、次のとおりです。

AMQ5203: XA インターフェースを呼び出しているときにエラーが発生しました。

説明

エラー番号は &2 です。ここで、1 の値は &1 に指定されたフラグ値が無効であったことを示します。2 は、同一のプロセス内で、スレッド化されたライブラリーとスレッド化されていないライブラリーの使用が試みられたことを示します。3 は、指定されたキュー・マネージャー名 '&3' にエラーがあったことを示します。4 は、&1 のリソース・マネージャー ID が無効であったことを示します。5 は、別のキュー・マネージャーが既に接続している場合に、'&3' という名前の 2 番目のキュー・マネージャーの使用が試行されたことを示します。6 は、アプリケーションがキュー・マネージャーに接続していない場合に、トランザクション・マネージャーが呼び出されたことを示します。7 は、別の呼び出しが進行中に、XA 呼び出しが行われたことを示します。8 は、xa_open 呼び出し内の xa_info ストリング '&4' に、パラメーター名 '&5' の無効なパラメーター値が含まれていたことを示します。9 は、xa_open 呼び出し内の xa_info ストリング '&4' で、パラメーター名が '&5' である必須パラメーターが欠落していることを示します。

ユーザーの応答

エラーを訂正して操作をやり直してください。

Windows Microsoft Transaction Server アプリケーションの準備と実行

MTS アプリケーションを、IBM MQ MQI client・アプリケーションとして実行するように準備するには、環境に応じて以下の指示に従ってください。

IBM MQ リソースにアクセスする Microsoft Transaction Server (MTS) アプリケーションの開発方法に関する一般情報については、 IBM MQ ヘルプ・センターの MTS に関するセクションを参照してください。

IBM MQ MQI client・アプリケーションとして動作するように MTS アプリケーションを準備するには、アプリケーションのコンポーネントごとに、次のいずれかを実行してください。

- コンポーネントが MQI に C 言語バインディングを使用する場合は、1022 ページの『Windows での C プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー mqic.lib ではなく、ライブラリー mqicxa.lib でリンクします。
- コンポーネントが IBM MQ C++ クラスを使用する場合は、548 ページの『Windows における C++ プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー imqc23vn.lib ではなく、ライブラリー imqx23vn.lib でリンクします。
- コンポーネントが MQI に Visual Basic 言語バインディングを使用する場合は、1025 ページの『Windows での Visual Basic プログラムの準備』 の指示に従います。ただし、Visual Basic プロジェクトを定義する際、「**条件付きコンパイル引数**」 フィールドに MqType=3 と入力します。

IBM MQ JMS アプリケーションの準備と実行

IBM MQ JMS アプリケーションは、WebSphere Application Server をトランザクション・マネージャーとして、クライアント・モードで実行できます。 特定の警告メッセージが表示される場合があります。

WebSphere Application Server をトランザクション・マネージャーとして使用して、 IBM MQ JMS アプリケーションをクライアント・モードで準備して実行するには、 82 ページの『IBM MQ classes for JMS/Jakarta Messaging の使用』 の指示に従ってください。

IBM MQ JMS クライアント・アプリケーションを実行すると、次の警告メッセージが表示される場合があります。

MQJE080

ライセンス・ユニットが足りません - setmqcap を実行してください (Insufficient license units - run setmqcap)

MQJE081

ライセンス・ユニット情報が入っているファイルの形式が誤っています - setmqcap を実行してください (File containing the license unit information is in the wrong format - run setmqcap)

MQJE082

ライセンス・ユニット情報が入っているファイルが見付かりませんでした - setmqcap を実行してください (File containing the license unit information could not be found - run setmqcap)

ユーザー出口、API 出口、および IBM MQ インストール可能サービス

このトピックには、これらのプログラムの使用および開発に関する情報へのリンクが含まれています。

ユーザー出口、API 出口、およびインストール可能サービスを使用してキューマネジャー機能を拡張する方法の概要については、「キュー・マネージャー機能」を参照してください。

出口およびインストール可能サービスの作成およびコンパイルについて詳しくは、サブトピックを参照してください。

関連概念

MQI チャネル用のチャネル出口プログラム

関連資料

API 出口参照

インストール可能サービス・インターフェースの参照情報

IBM i IBM i でのインストール可能サービス・インターフェースの参照情報

ALW AIX, Linux, and Windows での出口とインストール可能サービスの作成

AIX, Linux, and Windows 上の IBM MQ ライブラリーにリンクすることなく、出口を作成およびコンパイルすることができます。

このタスクについて

このトピックは、AIX, Linux, and Windows システムのみに適用されます。 他のプラットフォームの場合の出口とインストール可能サービスの作成について詳しくは、該当するプラットフォーム固有のトピックを参照してください。

IBM MQ がデフォルト以外の場所にインストールされている場合は、IBM MQ ライブラリーにリンクせずに出口を作成してコンパイルする必要があります。

これらの IBM MQ ライブラリーをリンクせずに、 AIX, Linux, and Windows システム上で出口を作成し、コンパイルすることができます。

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

これらのライブラリーにリンクしている既存の出口は、AIX and Linux システムで IBM MQ がデフォルトの場所にインストールされている限り、引き続き正しく動作します。

手順

1. cmqec.h ヘッダー・ファイルを組み込みます。

   このヘッダー・ファイルを組み込むと、cmqc.h、cmqxc.h、cmqzc.h の各ヘッダー・ファイルも自動的に組み込まれます。
2. MQI 呼び出しと DCI 呼び出しが MQIEP 構造体を経由するようにして、出口を作成します。 MQIEP 構造体の詳細については、MQIEP 構造体を参照してください。

   - インストール可能サービス
     - **Hconfig** パラメーターを使用して、MQZEP 呼び出しを参照します。
     - **Hconfig** パラメーターを使用する前に、**Hconfig** の最初の 4 バイトが MQIEP 構造体の **StrucId** と一致することを確認してください。

  - インストール可能サービス・コンポーネントを作成するための詳細については、MQIEP を参照してください。
- API 出口
  - **Hconfig** パラメーターを使用して、MQXEP 呼び出しを参照します。
  - **Hconfig** パラメーターを使用する前に、**Hconfig** の最初の 4 バイトが MQIEP 構造体の **StrucId** と一致することを確認してください。
  - API 出口を作成するための詳細については、960 ページの『API 出口の作成』を参照してください。
- チャネル出口
  - MQCXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQCXP のバージョン番号がバージョン 8 以上であることを確認してください。
  - チャネル出口を作成するための詳細については、970 ページの『チャネル出口プログラムの作成』を参照してください。
- データ変換出口
  - MQDXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQDXP のバージョン番号がバージョン 2 以上であることを確認してください。
  - **crtmqcvx** コマンドと amqsvfc0.c ソース・ファイルを使用して、**pEntryPoints** パラメーターを使用するデータ変換コードを作成できます。 995 ページの『IBM MQ for Windows 用のデータ変換出口の作成』および 993 ページの『IBM MQ for AIX or Linux システム用のデータ変換出口の作成』を参照してください。
  - **crtmqcvx** コマンドを使用して生成した既存のデータ変換出口がある場合は、更新後のコマンドを使用して出口を再生成する必要があります。
  - データ変換出口を作成するための詳細については、988 ページの『データ変換出口の作成』を参照してください。
- 事前接続出口
  - MQNXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQNXP のバージョン番号がバージョン 2 以上であることを確認してください。
  - 事前接続出口を作成するための詳細については、998 ページの『リポジトリーから接続前出口を使用した接続定義の参照』を参照してください。
- パブリッシュ出口
  - MQPSXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQPSXP のバージョン番号がバージョン 2 以上であることを確認してください。
  - パブリッシュ出口を作成するための詳細については、999 ページの『パブリッシュ出口の作成とコンパイル』を参照してください。
- クラスター・ワークロード出口
  - MQWXP 構造体の **pEntryPoints** パラメーターを使用して、MQXCLWLN 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQWXP のバージョン番号がバージョン 4 以上であることを確認してください。

– クラスター・ワークロード出口を作成するための詳細については、1001 ページの『クラスター・ワークロード出口の作成とコンパイル』を参照してください。

MQPUT を呼び出すチャネル出口の例を以下に示します。

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

その他の例については、1062 ページの『IBM MQ プロシージャー型サンプル・プログラムの使用』を参照してください。

3. 出口をコンパイルします。
   - IBM MQ ライブラリーにリンクしません。
   - IBM MQ ライブラリーへの埋め込み RPath を出口に組み込みません。
   - 出口をコンパイルするための詳細については、以下のいずれかのトピックを参照してください。
     – API 出口: 961 ページの『API 出口のコンパイル』。
     – チャネル出口、パブリッシュ出口、クラスター・ワークロード出口: 987 ページの『AIX, Linux, and Windows システムでのチャネル出口プログラムのコンパイル』。
     – データ変換出口: 988 ページの『データ変換出口の作成』。
4. 出口を以下のいずれかの場所に配置します。
   - 出口の構成時に完全に修飾する任意のパス
   - デフォルトの出口のパス。特定のインストール・ディレクトリーにあります。 例えば、*MQ_DATA_PATH*/exits/installation2 などです。
   - デフォルトの出口のパス

     デフォルトの出口パスは、32 ビット出口の場合は *MQ_DATA_PATH*/exits で、64 ビット出口の場合は *MQ_DATA_PATH*/exits64 です。 qm.ini ファイルまたは mqclient.ini ファイルでこれらのパスを変更することもできます。 詳細については、出口パスを参照してください。 Windows と Linux では、IBM MQ エクスプローラーを使用してパスを変更できます。

     a. キュー・マネージャー名を右クリックします。
     b.「**プロパティー...**」をクリックします。
     c.「**出口**」をクリックします。
     d.「出口デフォルト・パス」フィールドで、出口プログラムを保持するディレクトリーのパス名を指定します。

   特定のインストール・ディレクトリーとデフォルト・パス・ディレクトリーの両方に出口を配置すると、パスに名前が含まれている IBM MQ インストール環境では、特定のインストール・ディレクトリーにある出口が使用されます。 例えば、出口は /exits/installation2 と /exits に置かれますが、/exits/installation1 には置かれません。 IBM MQ インストール環境 installation2 では、/exits/installation2 の出口が使用されます。 IBM MQ インストール環境 installation1 では、/exits ディレクトリーの出口が使用されます。
5. 必要に応じて、出口を構成します。
   - インストール可能サービス: 953 ページの『構成サービスおよびコンポーネント』。
   - API 出口: 964 ページの『API 出口の構成』。
   - チャネル出口: 988 ページの『チャネル出口の構成』。
   - パブリッシュ出口: 1001 ページの『パブリッシュ出口の構成』。
   - 事前接続出口: クライアント構成ファイルの PreConnect スタンザ。

ALW *MQI* ライブラリーにリンクされていない *API* 出口

特定の環境では、MQIEP 関数ポインターを使用するよう再コーディングすることができない既存の API 出口を IBM MQ API ライブラリーとリンクしなければなりません。

これは、システムのランタイム・リンカーによって既存の API 出口が、関数ポインターをまだ読み込んでいないプログラムに正常に読み込まれるようになるために必要となります。

注 : この情報は、MQI 呼び出しを直接行う既存の API 出口に限定されます。 つまり、MQIEP を使用しない出口のみです。 可能な場合には、出口を再コーディングして MQIEP エントリー・ポイントを代わりに使用するように計画してください。

IBM MQ 8.0 以降、**runmqsc** が MQI ライブラリーと直接リンクされていないサンプル・プログラムです。

そのため、必要な IBM MQ API ライブラリーとリンクされていない、または MQIEP を使用するよう再コーディングされていない API 出口は **runmqsc** に読み込めません。

キュー・マネージャー・エラー・ログにエラーが表示されます。例えば、AMQ6175: `The system could not dynamically load the shared library` というメッセージが `undefined symbol: MQCONN` などの修飾テキストと一緒に表示されます。

また、AMQ7214: `The module for API Exit 'myexitname' could not be loaded` というメッセージが表示される場合もあります。

関連タスク

942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』
AIX, Linux, and Windows 上の IBM MQ ライブラリーにリンクすることなく、出口を作成およびコンパイルすることができます。

ALW AIX, Linux, and Windows 用のインストール可能サービスとコンポーネント

このセクションでは、インストール可能サービス、およびそれと関連した機能とコンポーネントについて紹介します。 これらの機能へのインターフェースは文書化されているため、お客さままたはソフトウェア・ベンダーがコンポーネントを供給することも可能です。

IBM MQ インストール可能サービスが提供されている主な理由は、以下のとおりです。

- IBM MQ 製品に付属のコンポーネントを使用するか、または他のコンポーネントで置換または追加するかを柔軟に選択できるようにするため。
- ベンダーが IBM MQ 製品の内部を変更しなくても、新規のテクノロジーなどを使用するコンポーネントを供給して参入できるようにするため。
- IBM MQ が新しいテクノロジーをより迅速かつ安価に利用できるようにすることで、製品を早期にかつ低価格で提供できるようにするため。

インストール可能サービス とサービス・コンポーネント は、IBM MQ 製品構成の一部です。 この構成の中心には、メッセージ・キュー・インターフェース (MQI) に関連する機能と規則を実装するキュー・マネージャーがあります。 この中心部には、その作業を実行するためのインストール可能サービス と呼ばれるいくつかのサービス機能が必要です。 インストール可能サービスは、以下のとおりです。

- 許可サービス
- ネーム・サービス

各インストール可能サービスは、1 つ以上のサービス・コンポーネント を使用して実装された関連機能セットです。 各コンポーネントは、適切に構築された一般的に利用可能なインターフェースを使用して呼び出されます。 これにより、独立系ソフトウェア・ベンダーおよび他のサード・パーティーがインストール可能コンポーネントを提供して、IBM MQ 製品に付属の機能を拡張したり置換したりできるようになります。 946 ページの表 138 は、使用可能なサービスおよびコンポーネントを一覧にまとめたものです。

表 138. インストール可能サービス・コンポーネントの要約			
インストール可能サービス	**提供コンポーネント**	**Function**	**要件**
許可サービス	オブジェクト権限マネージャー (OAM) (object authority manager (OAM))	コマンドおよび MQI 呼び出しの許可検査を行う。 ユーザーは独自のコンポーネントを作成して、OAM に追加したり、それを置き換えることができます。 例えば、ユーザー ID にキューをオープンする権限があるかどうかの確認など。	(適切なプラットフォーム許可機能を前提とする。)
ネーム・サービス	なし	指定のキューを所有するキュー・マネージャーの名前を検索するよう、キュー・マネージャーをサポートする。 • ユーザー定義	• サード・パーティーまたはユーザー作成のネーム・マネージャーで必要とされる。

インストール可能サービスのインターフェースは、インストール可能サービス・インターフェースの参照情報で説明されています。

関連タスク
インストール可能サービスの構成

サービス・コンポーネントの作成
このセクションでは、サービス、コンポーネント、エントリー・ポイント、および戻りコードの関係について説明します。

機能およびコンポーネント

各サービスは、関連機能のセットで構成されます。 例えば、ネーム・サービスには以下の機能が含まれます。

- キュー名を検索して、キューが定義されているキュー・マネージャーの名前を戻します。
- キュー名をサービスのディレクトリーに挿入します。
- キュー名をサービスのディレクトリーから削除します。

さらに、初期化と終了の機能も含まれています。

インストール可能サービスは、1 つ以上のサービス・コンポーネントによって 提供されます。 各コンポーネントは、そのサービスに定義されている機能の一部または全部を実行できます。 例えば、IBM MQ for AIX では、提供されている許可サービス・コンポーネントである OAM は、使用可能な機能すべてを実行します。 詳しくは、950 ページの『許可サービス・インターフェース』を参照してください。 コンポーネントは、サービスのインプリメントに必要な基礎となるリソースやソフトウェア (例えば、LDAP ディレクトリー) を管理する役割も果たしています。 構成ファイルは、コンポーネントをロードして、提供される機能ルーチンのアドレスを判別するための標準的な手段を用意してくれます。

947 ページの図 99 は、サービスとコンポーネントがどのように関係しているかを示しています。

- サービスは構成ファイル内のスタンザによってキュー・マネージャーに定義されます。
- 各サービスは、キュー・マネージャーに提供されているコードによってサポートされます。 ユーザーはこのコードを変更できないので、独自のサービスを作成することはできません。
- 各サービスは 1 つ以上のコンポーネントによってインプリメントされます。これらは製品に付属のものを使用したり、ユーザーで作成したりすることができます。 1 つのサービスに対して複数のコンポーネントを起動し、それぞれのコンポーネントにサービス内の異なる機能をサポートさせることもできます。

- エントリー・ポイントはサービス・コンポーネントをキュー・マネージャー内の サポートされているコードに接続します。

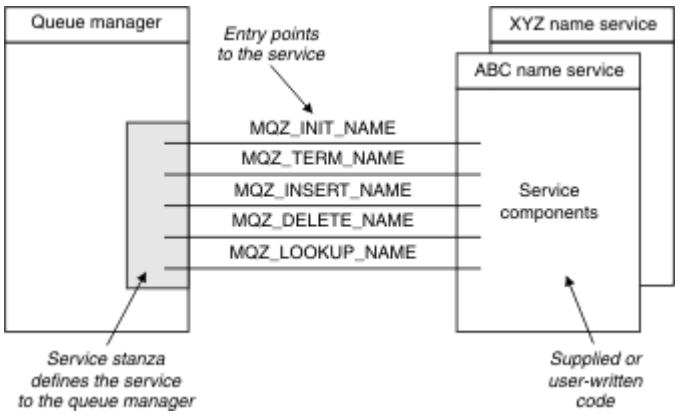


図 99. サービス、コンポーネント、およびエントリー・ポイントの簡単な関係図

エントリー・ポイント

各サービス・コンポーネントは、特定のインストール可能サービスをサポートする ルーチンのエントリー・ポイント・アドレスのリストによって表されます。 インストール可能サービスは、各ルーチンによって実行される機能を定義します。

構成する際にサービス・コンポーネントを配列する順序は、サービスの要求を満たすためにエントリー・ポイントが呼び出される順序を定義するものです。

提供されているヘッダー・ファイル `cmqzc.h` では、各サービスへの提供されているエントリー・ポイントに MQZID_ の接頭部があります。

サービスが存在する場合、サービスは事前定義の順序に読み込まれます。 次のリストは、サービスとその初期化の順序を示しています。

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` は、デフォルトで構成される唯一のサービスです。 `NameService` および `UserIdentifierService` を使用するには、それらを手動で構成する必要があります。

サービスおよびサービス・コンポーネントでは、1 対 1 または 1 対多のマッピングがあります。 サービスごとに、複数のサービス・コンポーネントを定義できます。 AIX and Linux システムでは、`ServiceComponent` スタンザのサービス値は、qm.ini ファイルにあるサービス・スタンザの Name 値と一致する必要があります。 Windows では、 `ServiceComponent` のサービス・レジストリー・キー値は、名前レジストリー・キー値と一致する必要があり、`HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` として定義されます。ここで、 *qmname* はキュー・マネージャーの名前です。

AIX and Linux システムでは、サービス・コンポーネントは qm.ini ファイルに定義されている順序で開始されます。 Windows では、Windows レジストリーが使用されるため、IBM MQ は **RegEnumKey** 呼び出しを実行し、値はアルファベット順に返されます。 そのため、Windows では、サービスはレジストリーで定義されているとおりにアルファベット順で呼び出されます。

`ServiceComponent` 定義の順序付けは重要です。 この順序付けにより、特定のサービスでコンポーネントが実行される順序が決まります。 例えば、 Windows 上の `AuthorizationService` は、`MQSeries.WindowsNT.auth.service` という名前のデフォルト OAM コンポーネントを使用して構成されます。 このサービスで追加のコンポーネントを定義して、デフォルトの OAM をオーバーライドすることができます。 `MQCACF_SERVICE_COMPONENT` を指定しない限り、アルファベット順で最初に来るコンポーネントが、要求を処理するために使用され、そのコンポーネントの名前が使用されます。

戻りコード

サービス・コンポーネントは、キュー・マネージャーに戻りコードを提供して、さまざまな状況を報告します。 それらは処理の成功または失敗を報告して、キュー・マネージャーが次のサービス・コンポーネントに進むかどうかを示します。 別の *Continuation* パラメーターが、この指示を伝えます。

コンポーネント・データ

単一のサービス・コンポーネントでは、データをさまざまな機能で共用することが必要になることがあります。 インストール可能なサービスは、サービス・コンポーネントの各呼び出しで受け渡しするためにオプションのデータ域を提供します。 このデータ域は、サービス・コンポーネントが専用に使用します。 これは、アドレス・スペースまたはプロセスが異なる場合でも、特定の機能のすべての呼び出しによって共用されます。 呼び出される場合は常にサービス・コンポーネントからアクセス可能であることを保証されます。 このデータ域のサイズを *ServiceComponent* スタンザで宣言する必要があります。

コンポーネントの初期化と終了

コンポーネントの初期化オプションおよび終了オプションの使用法。

コンポーネントの初期化ルーチンが呼び出されるときには、コンポーネントによってサポートされるエントリー・ポイントごとにキュー・マネージャーの **MQZEP** 関数を呼び出す必要があります。 **MQZEP** は、サービスにエントリー・ポイントを定義します。 未定義の出口点はすべて NULL と見なされます。

コンポーネントは、他の方法によって呼び出される前に、1 次初期化オプションによって必ず一度呼び出されます。

特定のプラットフォームでは、コンポーネントを 2 次初期化オプションで呼び出すことができます。 例えば、サービスにアクセスするオペレーティング・システム・プロセス、スレッド、またはタスクごとに一度呼び出すことができます。

2 次初期化が使用されると、次のようになります。

- コンポーネントを 2 次初期化のために複数回呼び出すことができます。 サービスが不要になったときは、そのような呼び出しごとに、対応する 2 次終了呼び出しが発行されます。

  ネーミング・サービスでは、これは MQZ_TERM_NAME 呼び出しです。

  許可サービスの場合、これは MQZ_TERM_AUTHORITY 呼び出しです。
- 1 次初期化および 2 次初期化のためにコンポーネントが呼び出されるたびに、エントリー・ポイントを (MQZEP を呼び出して) 再指定する必要があります。
- コンポーネントに対して使用されるコンポーネント・データのコピーは 1 つのみです。 2 次初期化ごとに異なるコピーが使用されるわけではありません。
- 2 次初期化が実行されるまで、そのコンポーネントが (状況に応じて、オペレーティング・システム・プロセス、スレッド、またはタスクからの) サービスに対する他の呼び出しのために起動されることはありません。
- コンポーネントは、**Version** パラメーターに 1 次初期化と 2 次初期化で同じ値を設定する必要があります。

コンポーネントは、不要になったときには必ず、1 次終了オプションにより一度呼び出されます。 このコンポーネントに対して、それ以降の呼び出しは行われません。

コンポーネントが 2 次初期化のために呼び出されている場合は、2 次終了オプションにより呼び出されます。

オブジェクト権限マネージャー *(OAM)*

IBM MQ 製品に付属の許可サービス・コンポーネントは、オブジェクト権限マネージャー (OAM) と呼ばれます。

デフォルトでは、OAM はアクティブで、制御コマンド **dspmqaut** (権限の表示)、**dmpmqaut** (権限のダンプ)、および **setmqaut** (権限の設定またはリセット) を処理することになります。

これらのコマンドの構文と使用方法については、制御コマンドを使用した IBM MQ for Multiplatforms の管理を参照してください。

OAM は、プリンシパルまたはグループのエンティティー で動作します。

- Linux AIX AIX and Linux システムの場合、プリンシパルはユーザー ID、またはユーザーの代理で実行されるアプリケーション・プログラムに関連付けられた ID です。グループはプリンシパルのシステム定義コレクションです。
- Windows Windows システムの場合、プリンシパルは Windows ユーザー ID、またはユーザーの代理で実行されるアプリケーション・プログラムに関連付けられた ID です。グループは Windows グループです。

権限はプリンシパル・レベルまたはグループ・レベルで認可または取り消しできます。

MQI 要求が行われるかコマンドが発行されると、OAM は、操作に関連するエンティティーに、要求された操作を実行する権限および指定されたキュー・マネージャーのリソースにアクセスする権限があるかどうかを確認します。

許可サービスによって、独自の許可サービス・コンポーネントを作成することにより、キュー・マネージャー用に備えられている権限検査を拡張または置換することができます。

ネーム・サービス
ネーム・サービスは、キュー・マネージャーにサポートを提供して、指定のキューを所有するキュー・マネージャーの名前を検索できるようにするインストール可能サービスです。 ネーム・サービスから検索できる他のキュー属性はありません。

ネーム・サービスにより、アプリケーションは出力用のリモート・キューをローカル・キューであるかのようにオープンすることが可能になります。 ネーム・サービスは、キュー以外のオブジェクトに対しては呼び出されません。

注 : リモート・キューの **Scope** 属性は CELL に設定されていなければなりません。

アプリケーションではキューをオープンする場合、キュー・マネージャーのディレクトリー内で最初にそのキューの名前を探します。 そこに存在しない場合、そのキュー名が認識されるものが見つかるまで、構成されているすべてのネーム・サービスを検索します。 名前が認識されるものが見つからない場合、オープンは失敗します。

ネーム・サービスはそのキューを所有するキュー・マネージャーを戻します。 その後キュー・マネージャーは、コマンドの最初の要求にキューおよびキュー・マネージャー名が指定されていた場合と同様に MQOPEN 要求を続行します。

ネーム・サービス・インターフェース (NSI) は、IBM MQ フレームワークの一部です。

ネーム・サービスの機能

キュー定義で **Scope** 属性がキュー・マネージャー (つまり MQSC の SCOPE(QMGR)) として指定されている場合、キュー定義は (すべてのキュー属性と共に) キュー・マネージャーのディレクトリーのみに保管されます。 これは、インストール可能サービスに置き換えることはできません。

キュー定義で **Scope** 属性がセル (つまり MQSC の SCOPE(CELL)) として指定されている場合も、キュー定義は (すべてのキュー属性と共に) キュー・マネージャーのディレクトリーに保管されます。 ただし、キューおよびキュー・マネージャー名はネーム・サービスにも保管されます。 この情報を保管できるサービスがない場合、*Scope* セルのあるキューは定義できません。

情報の保管先ディレクトリーは、サービスによって管理できます。 または、サービスが、基礎となるサービス (LDAP ディレクトリーなど) をこの目的で使用することもできます。 いずれの場合にも、ディレクトリーに保管される定義は、明示的に削除されるまで、コンポーネントおよびキュー・マネージャーが終了した後も保管され続ける必要があります。

注 :

1. ネーミング・ディレクトリー・セル内の異なるキュー・マネージャーに対するリモート・ホストのローカル・キュー定義に、(スコープが CELL の) メッセージを送信するには、チャネルを定義する必要があります。
2. リモート・キューのスコープが CELL となっている場合でも、リモート・キューから直接メッセージを取得することはできません。

3. スコープが CELL のキューを送信する場合、リモート・キュー定義は必要ありません。
4. ネーミング・サービスは主に宛先キューを定義します。 ただし、宛先キュー・マネージャーへの伝送キューと 1 対のチャネル定義は引き続き必要となります。 さらに、ローカル・システム上の伝送キューの名前は、スコープが CELL の宛先キューを所有するリモート・システム上のキュー・マネージャーの名前と同じでなければなりません。

   例えば、リモート・キュー・マネージャーの名前が QM01 である場合、ローカル・システム上の伝送キューの名前も QM01 でなければなりません。

許可サービス・インターフェース

許可サービスは、キュー・マネージャーで使用されるエントリー・ポイントを提供しています。

エントリー・ポイントは次のとおりです。

MQZ_AUTHENTICATE_USER
ユーザー ID およびパスワードを認証し、アイデンティティー・コンテキスト・フィールドを設定できます。

MQZ_CHECK_AUTHORITY
エンティティーが、指定されたオブジェクトに 1 つ以上の操作を実行する権限を所有しているかどうかを検査します。

MQZ_CHECK_PRIVILEGED
指定されたユーザーが特権ユーザーであるかどうかを検査します。

MQZ_COPY_ALL_AUTHORITY
参照されたオブジェクトに存在する現在のすべての許可を別のオブジェクトにコピーします。

MQZ_DELETE_AUTHORITY
指定されたオブジェクトに関連するすべての許可を削除します。

MQZ_ENUMERATE_AUTHORITY_DATA
指定された選択基準に一致するすべての権限データを検索します。

MQZ_FREE_USER
関連した割り振られたリソースを解放します。

MQZ_GET_AUTHORITY
指定されたオブジェクトに対してエンティティーがアクセスする権限を取得します。

MQZ_GET_EXPLICIT_AUTHORITY
名前付きグループが指定されたオブジェクトにアクセスする権限 (ただし、**nobody (なし)** グループの追加権限はなし) または名前付きプリンシパルの 1 次グループが指定されたオブジェクトにアクセスする権限を取得します。

MQZ_INIT_AUTHORITY
許可サービス・コンポーネントを初期化します。

MQZ_INQUIRE
サポートされている許可サービスの機能を照会します。

MQZ_REFRESH_CACHE
許可をすべてリフレッシュします。

MQZ_SET_AUTHORITY
指定されたオブジェクトに対してエンティティーが所有する権限を設定します。

MQZ_TERM_AUTHORITY
許可サービス・コンポーネントを終了します。

さらに、IBM MQ for Windows 上で、許可サービスはキュー・マネージャーで使用するために以下のエントリー・ポイントを提供しています。

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

これらのエントリー・ポイントは、Windows Security Identifier (NT SID) をサポートします。

これらの名前は、ヘッダー・ファイル cmqzc.h に **typedef** として指定され、コンポーネント機能のプロトタイピングに使用できます。

初期化関数 (**MQZ_INIT_AUTHORITY**) は、コンポーネントの主エントリー・ポイントでなければなりません。 他の機能は、初期化機能がコンポーネントのエントリー・ポイントのベクトルに追加したエントリー・ポイント・アドレスを介して起動されます。

ネーム・サービス・インターフェース
ネーム・サービスは、キュー・マネージャーが使用するエントリー・ポイントを提供します。

以下のエントリー・ポイントが提供されます。

MQZ_INIT_NAME
ネーム・サービス・コンポーネントを初期化します。

MQZ_TERM_NAME
ネーム・サービス・コンポーネントを終了します。

MQZ_LOOKUP_NAME
指定したキューのキュー・マネージャー名を検索します。

MQZ_INSERT_NAME
指定したキューを所有するキュー・マネージャーの名前を含む項目を、サービスが使用するディレクトリーに挿入します。

MQZ_DELETE_NAME
指定したキューの項目を、サービスが使用するディレクトリーから削除します。

複数のネーム・サービスが構成されている場合は、次のことが行われます。

- 検索では、キュー名が解決するまで、MQZ_LOOKUP_NAME 関数がリスト内の各サービスに対して呼び出されます (コンポーネントに検索を終了する指示が含まれていない場合)。
- 挿入では、MQZ_INSERT_NAME 関数がこの関数をサポートするリスト内の最初のサービスに対して呼び出されます。
- 削除では、MQZ_DELETE_NAME 関数がこの関数をサポートするリスト内の最初のサービスに対して呼び出されます。

挿入関数と削除関数をサポートするコンポーネントが複数あってはなりません。 ただし、検索のみをサポートするコンポーネントは可能であり、使用できます。 例えば、リスト内の最後のコンポーネントとして使用し、他のネーム・サービス・コンポーネントに認識されない任意の名前を、その名前を定義できるキュー・マネージャーに解決します。

C プログラミング言語では、typedef ステートメントを使用して、名前が関数データ型として定義されます。 これらを使用して、サービス関数をプロトタイプ化し、パラメーターが正しいことを確認できます。

インストール可能サービスに特定のすべての情報を含むヘッダー・ファイルは、C 言語では cmqzc.h です。

コンポーネントのメイン・エントリー・ポイントでなければならない初期化関数 (MQZ_INIT_NAME) とは別に、初期化関数が MQZEP 呼び出しによって追加したエントリー・ポイント・アドレスによっても関数が呼び出されます。

複数のサービス・コンポーネントの使用
1 つのサービスに複数のコンポーネントをインストールできます。 これにより、コンポーネントがサービスの実装の一部のみを提供して、残りの機能は他のコンポーネントが提供するようにすることができます。

複数のコンポーネントの使用例

ABC_name_serv および XYZ_name_serv という 2 つのネーム・サービス・コンポーネントを作成するとします。

ABC_name_serv
このコンポーネントは、サービス・ディレクトリーへの名前の挿入、およびサービス・ディレクトリーからの名前の削除をサポートしますが、キュー名の検索はサポートしません。

XYZ_name_serv
このコンポーネントはキュー名の検索をサポートしますが、サービス・ディレクトリーへの名前の挿入、およびサービス・ディレクトリーからの名前の削除はサポートしません。

コンポーネント ABC_name_serv は、キュー名のデータベースを保持し、2 つの単純なアルゴリズムを使用してサービス・ディレクトリーに名前を挿入するか、サービス・ディレクトリーから名前を削除します。

コンポーネント XYZ_name_serv は、そのコンポーネントの呼び出しに使用されたキュー名に対して、固定のキュー・マネージャー名を戻す、簡単なアルゴリズムを使用します。 キュー名のデータベースは保持されないので、挿入機能および削除機能はサポートされていません。

コンポーネントは、同じキュー・マネージャーにインストールされます。 *ServiceComponent* スタンザは、コンポーネント ABC_name_serv が最初に呼び出されるように配列されています。 コンポーネント・ディレクトリーでキューを挿入または削除する呼び出しは、コンポーネント ABC_name_serv によって処理されます。 このコンポーネントのみが、これらの機能をインプリメントします。 ただし、コンポーネント ABC_name_serv が解決できないルックアップ呼び出しは、ルックアップ専用コンポーネント XYZ_name_serv に渡されます。 このコンポーネントは、その簡単なアルゴリズムによってキュー・マネージャー名を提供します。

複数のコンポーネントを使用する際にエントリー・ポイントを省略する

複数のコンポーネントを使用してサービスを提供することにした場合、一部の機能を実装しないサービス・コンポーネントを設計することができます。 インストール可能サービスのフレームワークでは、何を省略できるかについての制限はありません。 ただし、ある特定のインストール可能サービスでは、機能を 1 つ以上省略すると、サービスの目的と論理的に矛盾する場合があります。

複数のコンポーネントと共に使用されるエントリー・ポイントの例

952 ページの表 139 に、2 つのコンポーネントがインストールされているインストール可能なネーム・サービスの例を示します。 各コンポーネントは、この特定のインストール可能サービスに関連した機能の異なるセットをサポートしています。 挿入機能については、ABC コンポーネントのエントリー・ポイントが最初に呼び出されます。 サービスに (**MQZEP** を使用して) 定義されていないエントリー・ポイントは、NULL であると見なされます。 この表には初期化のエントリー・ポイントも示されていますが、初期化はコンポーネントのメイン・エントリー・ポイントによって実行されるため、これは必要ありません。

キュー・マネージャーでは、インストール可能サービスを使用する必要が生じると、そのサービスに定義されたエントリー・ポイント (952 ページの表 139 の列に示されています) を使用します。 キュー・マネージャーは各コンポーネントを順番に検討して、必要な機能を実装するルーチンのアドレスを判別します。その後、ルーチンが存在すればそれを呼び出します。 処理が成功すると、結果および状況についてのすべての情報がキュー・マネージャーによって使用されます。

表 *139.* インストール可能サービスのエントリー・ポイントの例

機能番号	**ABC ネーム・サービス・コンポーネント**	**XYZ ネーム・サービス・コンポーネント**
MQZID_INIT_NAME (初期化)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (終了)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (挿入)	ABC_Insert()	NULL
MQZID_DELETE_NAME (削除)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (検索)	NULL	XYZ_Lookup()

ルーチンが存在しない場合、キュー・マネージャーはリストの次のコンポーネントに対して、このプロセスを繰り返します。 さらに、ルーチンが存在しても、処理が実行できなかったことを示すコードが戻され

た場合、使用可能な次のコンポーネントで試行を続けます。 サービス・コンポーネント内のルーチンが、操作を実行するための試行を続けないように指示するコードを戻す場合もあります。

構成サービスおよびコンポーネント

キュー・マネージャー構成ファイルを使用して、サービス・コンポーネントを構成します。ただし、各キュー・マネージャー構成ファイルがレジストリーに独自のスタンザを持つ Windows システムを除きます。

手順

1. キュー・マネージャー構成ファイル qm.ini にスタンザを追加して、サービスをキュー・マネージャーに定義し、モジュールの場所を指定します。
   - 使用される各サービスは、サービスをキュー・マネージャーに定義する Service スタンザ を持っている必要があります。 詳しくは、 qm.ini ファイルのサービス・スタンザを参照してください。
   - サービス内の各コンポーネントには、ServiceComponent スタンザが必要です。 このスタンザは、そのコンポーネントのコードを含むモジュールの名前とパスを識別します。 詳しくは、qm.ini ファイルの ServiceComponent スタンザを参照してください。

   オブジェクト権限マネージャー (OAM) と呼ばれる許可サービス・コンポーネントは、製品に同梱されています。 キュー・マネージャーを作成するとき、キュー・マネージャー構成ファイル (または Windows システムのレジストリー) が自動的に更新されて、許可サービスおよびデフォルト・コンポーネント (OAM) のための適切なスタンザを内蔵するようになります。 その他のコンポーネントについては、キュー・マネージャー構成ファイルを手動で構成する必要があります。

   各サービス・コンポーネントのコードは、プラットフォームが動的バインディングをサポートしている場合にはそれを使用して、キュー・マネージャーの始動時にキュー・マネージャーへロードされます。
2. キュー・マネージャーをいったん停止したあとで再始動して、コンポーネントをアクティブにします。

関連資料

qm.ini ファイルの Service スタンザ

qm.ini ファイルの ServiceComponent スタンザ

ユーザーの権限変更後の *OAM* のリフレッシュ

IBM MQ では、ユーザーの許可グループ・メンバーシップを変更した直後に OAM の許可グループ情報をリフレッシュし、キュー・マネージャーを停止して再始動することなく、オペレーティング・システム・レベルで行われた変更を反映できます。 これを行うには、**REFRESH SECURITY** コマンドを発行します。

注 : setmqaut コマンドを使用して権限を変更すると、OAM はそのような変更を即時に実装します。

キュー・マネージャーは、権限データを SYSTEM.AUTH.DATA.QUEUE というローカル・キューに保管します。 このデータは **amqzfuma.exe** によって管理されます。

関連資料

REFRESH SECURITY

IBM i IBM i でのインストール可能サービスとコンポーネント

この情報は、インストール可能サービスと、それに関連した機能およびコンポーネントについて学習するために使用します。 これらの機能へのインターフェースは文書化されているため、お客さままたはソフトウェア・ベンダーがコンポーネントを供給することも可能です。

IBM MQ インストール可能サービスが提供されている主な理由は、以下のとおりです。

- IBM MQ for IBM i 製品に付属のコンポーネントを使用するか、または他のコンポーネントで置換または追加するかを柔軟に選択できるようにするため。
- ベンダーが IBM MQ for IBM i 製品の内部を変更しなくても、新規のテクノロジーなどを使用するコンポーネントを供給して参入できるようにするため。
- IBM MQ が新しいテクノロジーをより迅速かつ安価に利用できるようにすることで、製品を早期にかつ低価格で提供できるようにするため。

インストール可能サービス とサービス・コンポーネント は、IBM MQ 製品構成の一部です。 この構成の中心には、メッセージ・キュー・インターフェース (MQI) に関連する機能と規則を実装するキュー・マネージャーがあります。 この中心部には、その作業を実行するためのインストール可能サービス と呼ばれるいくつかのサービス機能が必要です。 IBM MQ for IBM i で使用可能なインストール可能サービスは、許可サービスです。

各インストール可能サービスは、1 つ以上のサービス・コンポーネント を使用して実装された関連機能セットです。 各コンポーネントは、適切に構築された一般的に利用可能なインターフェースを使用して呼び出されます。 これにより、IBM MQ for IBM i で提供されるインストール可能コンポーネントを、独立系ソフトウェア・ベンダーやサード・パーティーが提供するインストール可能コンポーネントで強化または置換することが可能になります。 954 ページの表 140 に、許可サービスのサポートを要約します。

表 *140.* 許可サービス・コンポーネントの要約

提供コンポーネント	Function	要件
オブジェクト権限マネージャー (OAM)	コマンドおよび MQI 呼び出しの許可検査を行う。 ユーザーは独自のコンポーネントを作成して、OAM に追加したり、それを置き換えることができます。	(適切なプラットフォーム許可機能を前提とする。)
DCE ネーム・サービス・コンポーネント **注 :** DCE は、IBM MQ の V6.0 より前のバージョンでのみサポートされている。	• キュー・マネージャーによるキューの共用を許可、または • ユーザー定義 **注 :** 共用キューでは、**Scope** 属性を CELL に設定する必要がある。	• DCE が、提供されたコンポーネントで必要である。または • サード・パーティーまたはユーザー作成のネーム・マネージャーで必要とされる。

IBM i *IBM i での機能およびコンポーネント*

この情報は、IBM MQ for IBM i で使用できる機能、コンポーネント、エントリー・ポイント、戻りコード、およびコンポーネント・データについて理解するのに役立ちます。

各サービスは、関連機能のセットで構成されます。 例えば、ネーム・サービスには以下の機能が含まれます。

- キュー名を検索して、キューが定義されているキュー・マネージャーの名前を戻します。
- キュー名をサービスのディレクトリーに挿入します。
- キュー名をサービスのディレクトリーから削除します。

さらに、初期化と終了の機能も含まれています。

インストール可能サービスは、1 つ以上のサービス・コンポーネントによって 提供されます。 各コンポーネントは、そのサービスに定義されている機能の一部または全部を実行できます。 また、コンポーネントは、サービスを実装するのに必要な、基本となるすべてのリソースまたはソフトウェアの管理も行います。 構成ファイルは、コンポーネントをロードして、提供される機能ルーチンのアドレスを判別するための標準的な手段を用意してくれます。

サービスとコンポーネントは、次のように関連しています。

- サービスは構成ファイル内のスタンザによってキュー・マネージャーに定義されます。
- 各サービスは、キュー・マネージャーに提供されているコードによってサポートされます。 ユーザーはこのコードを変更できないので、独自のサービスを作成することはできません。
- 各サービスは 1 つ以上のコンポーネントによってインプリメントされます。これらは製品に付属のものを使用したり、ユーザーで作成したりすることができます。 1 つのサービスに対して複数のコンポーネントを起動し、それぞれのコンポーネントにサービス内の異なる機能をサポートさせることもできます。
- エントリー・ポイントはサービス・コンポーネントをキュー・マネージャー内の サポートされているコードに接続します。

エントリー・ポイント

各サービス・コンポーネントは、特定のインストール可能サービスをサポートする ルーチンのエントリー・ポイント・アドレスのリストによって表されます。 インストール可能サービスは、各ルーチンによって実行される機能を定義します。 構成する際にサービス・コンポーネントを配列する順序は、サービスの要求を満たすためにエントリー・ポイントが呼び出される順序を定義するものです。 提供されているヘッダー・ファイル cmqzc.h では、各サービスへの提供されているエントリー・ポイントに MQZID_ の接頭部があります。

戻りコード

サービス・コンポーネントは、キュー・マネージャーに戻りコードを提供して、さまざまな状況を報告します。 それらは処理の成功または失敗を報告して、キュー・マネージャーが次のサービス・コンポーネントに進むかどうかを示します。 別の *Continuation* パラメーターが、この指示を伝えます。

コンポーネント・データ

単一のサービス・コンポーネントでは、データをさまざまな機能で共用することが必要になることがあります。 インストール可能なサービスは、特定のサービス・コンポーネントの各呼び出しで受け渡しするためにオプションのデータ域を提供します。 このデータ域は、サービス・コンポーネントが専用に使用します。 指定の機能に対するすべての呼び出しは、異なるアドレス・スペースまたはプロセスから作成された呼び出しであっても、このデータ域を共用します。 呼び出される場合は常にサービス・コンポーネントからアクセス可能であることを保証されます。 このデータ域のサイズを *ServiceComponent* スタンザで宣言する必要があります。

IBM i *IBM i での初期化*

コンポーネントの初期化ルーチンが呼び出されるときには、コンポーネントによってサポートされるエントリー・ポイントごとにキュー・マネージャーの MQZEP 関数を呼び出す必要があります。 MQZEP は、サービスにエントリー・ポイントを定義します。 未定義の出口点はすべて NULL と見なされます。

1 次初期化
コンポーネントは、他の方法で呼び出される前に、常にこのオプションで 1 回呼び出されます。

2 次初期化
一部のプラットフォームでは、コンポーネントをこのオプションで呼び出すことができます。 例えば、サービスにアクセスするオペレーティング・システム・プロセス、スレッド、またはタスクごとに一度呼び出すことができます。

2 次初期化が使用されると、次のようになります。

- コンポーネントを 2 次初期化のために複数回呼び出すことができます。 サービスが不要になったときは、そのような呼び出しごとに、対応する 2 次終了呼び出しが発行されます。

  許可サービスの場合、これは MQZ_TERM_AUTHORITY 呼び出しです。
- 1 次初期化および 2 次初期化のためにコンポーネントが呼び出されるたびに、エントリー・ポイントを (MQZEP を呼び出して) 再指定する必要があります。
- コンポーネントに対して使用されるコンポーネント・データのコピーは 1 つのみです。 2 次初期化ごとに異なるコピーが使用されるわけではありません。
- 2 次初期化が実行されるまで、そのコンポーネントが (状況に応じて、オペレーティング・システム・プロセス、スレッド、またはタスクからの) サービスに対する他の呼び出しのために起動されることはありません。
- コンポーネントは、**Version** パラメーターに 1 次初期化と 2 次初期化で同じ値を設定する必要があります。

1 次終了
コンポーネントは、必要でなくなった場合に、常にこのオプションを使用して一度開始されます。 このコンポーネントに対して、それ以降の呼び出しは行われません。

2 次終了
コンポーネントは、2 次初期化用に開始された場合、このオプションを使用して開始されます。

IBM i *IBM i でのサービスとコンポーネントの構成*

サービス・コンポーネントは、キュー・マネージャー構成ファイルを使用して構成します。

手順

1. キュー・マネージャー構成ファイル qm.ini にスタンザを追加して、サービスをキュー・マネージャーに定義し、モジュールの場所を指定します。
   - 使用される各サービスは、サービスをキュー・マネージャーに定義する Service スタンザ を持っている必要があります。 詳しくは、 qm.ini ファイルのサービス・スタンザを参照してください。
   - サービス内の各コンポーネントには、ServiceComponent スタンザが必要です。 このスタンザは、そのコンポーネントのコードを含むモジュールの名前とパスを識別します。 詳しくは、qm.ini ファイルの ServiceComponent スタンザを参照してください。

   オブジェクト権限マネージャー (OAM) と呼ばれる許可サービス・コンポーネントは、製品に同梱されています。 キュー・マネージャーを作成すると、キュー・マネージャー構成ファイルが自動的に更新されて、許可サービスとデフォルトのコンポーネント (OAM) の該当するスタンザを組み込みます。 その他のコンポーネントについては、キュー・マネージャー構成ファイルを手動で構成する必要があります。

   各サービス・コンポーネントのコードは、プラットフォームが動的バインディングをサポートしている場合にはそれを使用して、キュー・マネージャーの始動時にキュー・マネージャーへロードされます。
2.

IBM i *IBM i での独自のサービス・コンポーネントの作成*

この情報を使用して、IBM MQ for IBM i でサービス・コンポーネントを作成する方法について学習します。

独自のサービス・コンポーネントを作成する方法は、以下のとおりです。

- ヘッダー・ファイル cmqzc.h が、使用するプログラムに組み込まれていることを確認します。
- プログラムをコンパイルし、共用ライブラリー (libmqm* および libmqmzf*) とリンクして、共用ライブラリーを作成します。

  注: エージェントはスレッド化された環境で実行できるため、OAM をスレッド化された環境で実行するように構築する必要があります。 これには、libmqm および libmqmzf のスレッド化されたバージョンを使用することも含まれています。
- キュー・マネージャー構成ファイルにスタンザを追加して、サービスをキュー・マネージャーに定義し、モジュールの位置を指定します。
- キュー・マネージャーをいったん停止したあとで再始動して、コンポーネントをアクティブにします。

IBM i *IBM i での許可サービス*

許可サービスは、キュー・マネージャーが許可機能を起動できるようにするインストール可能サービスです。例えば、ユーザー ID にキューをオープンする権限があるかどうかを確認します。

このサービスは、IBM MQ フレームワークの一部である、IBM MQ セキュリティー使用可能化インターフェース (SEI) のコンポーネントです。 以下の事柄について説明します。

- 956 ページの『オブジェクト権限マネージャー (OAM)』
- 957 ページの『オペレーティング・システムに対するサービスの定義』
- 957 ページの『許可サービス・スタンザの構成』
- 958 ページの『IBM i での許可サービス・インターフェース』

オブジェクト権限マネージャー (OAM)

IBM MQ 製品に付属の許可サービス・コンポーネントは、オブジェクト権限マネージャー (OAM) と呼ばれます。 デフォルトでは、OAM はアクティブになっており、以下の制御コマンドを処理します。

- **WRKMQMAUT** (権限の処理)
- **WRKMQMAUTD** (権限データの処理)

- **DSPMQMAUT** (オブジェクト権限の表示)
- **GRTMQMAUT** (オブジェクト権限の付与)
- **RVKMQMAUT** (オブジェクト権限の取り消し)
- **RFRMQMAUT** (セキュリティーのリフレッシュ)

これらのコマンドの構文とその使用法については、CL コマンド・ヘルプを参照してください。 OAM は、プリンシパルまたはグループのエンティティー で動作します。

MQI 要求が行われるかコマンドが発行されると、OAM は処理に関連するエンティティーの権限を検査して、次のアクションを実行することが可能であるかどうかを確認します。

- 要求された操作の実行。
- 指定されたキュー・マネージャー・リソースへのアクセス。

許可サービスによって、独自の許可サービス・コンポーネントを作成することにより、キュー・マネージャー用に備えられている権限検査を拡張または置換することができます。

オペレーティング・システムに対するサービスの定義

キュー・マネージャー構成ファイル qm.ini 内の許可サービス・スタンザは、許可サービスをキュー・マネージャーに対して定義します。 スタンザのタイプの詳細については、956 ページの『IBM i でのサービスとコンポーネントの構成』を参照してください。

許可サービス・スタンザの構成

On IBM MQ for IBM i:

プリンシパル
IBM i システム・ユーザー・プロファイル。

グループ
IBM i システム・グループ・プロファイル。

権限は、グループ・レベルでのみ認可または取り消しできます。 ユーザーの権限の認可または取り消し要求により、そのユーザーの 1 次グループが更新されます。

各キュー・マネージャーには、独自のキュー・マネージャー構成ファイルがあります。 例えば、キュー・マネージャー QMNAME のキュー・マネージャー構成ファイルのデフォルト・パスおよびファイル名は、/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini です。

デフォルト許可コンポーネントの *Service* スタンザおよび *ServiceComponent* スタンザは、qm.ini に自動的に追加されますが、これは WRKENVVAR によって指定変更することができます。 その他の *ServiceComponent* スタンザは、すべて手動で追加する必要があります。

例えば、キュー・マネージャー構成ファイル内の次のスタンザは、2 つの許可サービス・コンポーネントを定義します。

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

図 *100. IBM i での qm.ini* 内の許可サービス・スタンザ

最初のサービス・コンポーネント・スタンザ MQ.UNIX.authorization.service は、デフォルトの許可サービス・コンポーネントである OAM を定義します。 このスタンザを削除してキュー・マネージャーを再始動すると、OAM は使用不可となり、許可検査は行われません。

IBM i *IBM i* での許可サービス・インターフェース

許可サービス・インターフェースは、キュー・マネージャーで使用されるいくつかのエントリー・ポイントを提供しています。

MQZ_AUTHENTICATE_USER
ユーザー ID およびパスワードを認証し、アイデンティティー・コンテキスト・フィールドを設定できます。

MQZ_CHECK_AUTHORITY
エンティティーが、指定されたオブジェクトに 1 つ以上の操作を実行する権限を所有しているかどうかを検査します。

MQZ_COPY_ALL_AUTHORITY
参照されたオブジェクトに存在する現在のすべての許可を別のオブジェクトにコピーします。

MQZ_DELETE_AUTHORITY
指定されたオブジェクトに関連するすべての許可を削除します。

MQZ_ENUMERATE_AUTHORITY_DATA
指定された選択基準に一致するすべての権限データを検索します。

MQZ_FREE_USER
関連した割り振られたリソースを解放します。

MQZ_GET_AUTHORITY
指定されたオブジェクトに対してエンティティーがアクセスする権限を取得します。

MQZ_GET_EXPLICIT_AUTHORITY
名前付きグループが指定されたオブジェクトにアクセスする権限 (ただし、**nobody (なし)** グループの追加権限はなし) または名前付きプリンシパルの 1 次グループが指定されたオブジェクトにアクセスする権限を取得します。

MQZ_INIT_AUTHORITY
許可サービス・コンポーネントを初期化します。

MQZ_INQUIRE
サポートされている許可サービスの機能を照会します。

MQZ_REFRESH_CACHE
許可をすべてリフレッシュします。

MQZ_SET_AUTHORITY
指定されたオブジェクトに対してエンティティーが所有する権限を設定します。

MQZ_TERM_AUTHORITY
許可サービス・コンポーネントを終了します。

これらのエントリー・ポイントは、Windows Security Identifier (NT SID) をサポートします。

これらの名前は、ヘッダー・ファイル cmqzc.h に **typedef** として指定され、コンポーネント機能のプロトタイピングに使用できます。

初期化関数 (**MQZ_INIT_AUTHORITY**) は、コンポーネントの主エントリー・ポイントでなければなりません。 他の機能は、初期化機能がコンポーネントのエントリー・ポイントのベクトルに追加したエントリー・ポイント・アドレスを介して起動されます。

詳しくは、956 ページの『IBM i での独自のサービス・コンポーネントの作成』を参照してください。

Multi マルチプラットフォームでの API 出口の作成とコンパイル

API 出口を使用すると、MQPUT および MQGET などの IBM MQ API 呼び出しの動作を変更するコードを作成して、これらの呼び出しの直前または直後に、そのコードを挿入することができます。

注 : z/OS IBM MQ for z/OS ではサポートされない。

API 出口を使用する理由

各アプリケーションには実行すべき特定の ジョブがあり、アプリケーションのコードは、そのタスクをできるだけ効率的に実行しなければなりません。 もっと高いレベルで考えるなら、特定のキュー・マネージャーを使用するすべてのアプリケーション用に、キュー・マネージャーに対して標準またはビジネス・プロセスを 適用したいと思うことがあるかもしれません。 個々のアプリケーションより上のレベルで実行することにより、影響を受ける各アプリケーションのコードを 変更しないで済むため、効率が良くなります。

以下は、API 出口が役立つ場合のある分野に関するいくつかの提案です。

セキュリティー
セキュリティー上の理由で、アプリケーションがキューまたはキュー・マネージャーにアクセスする許可があることを検査するための認証を提供できます。 また、個々の API 呼び出し、またはそれらの呼び出しが使用するパラメーターをも認証することによって、アプリケーションによる API の使用を監視することもできます。

柔軟性
柔軟性を持たせるため、ビジネス環境中のデータに依存するアプリケーションを変更せずに、その環境で敏速な変更に応答することができます。 例えば、金利、為替レート、または製造環境におけるコンポーネントの価格などの変更に応答する API 出口を持つことができます。

キューまたはキュー・マネージャーの使用状況のモニタリング
キューまたはキュー・マネージャーの使用をモニターする場合、アプリケーションおよびメッセージの流れのトレース、API 呼び出しにおけエラーのログ記録、アカウントの目的での監査証跡のセットアップ、または計画の目的での使用法統計の収集などを実行できます。

API 出口の実行時に起きる事柄

出口プログラムを作成して、IBM MQ に対して識別しておけば、キュー・マネージャーは登録済みのポイントで自動的に出口コードを呼び出します。

実行する API 出口ルーチンは、Multiplatforms のスタンザで識別されます。 このトピックでは、構成ファイル `mqs.ini` および `qm.ini` 内のスタンザについて説明します。

次の 3 つの場所でルーチンを定義できます。

1. mqs.ini ファイルにある ApiExitCommon は、キュー・マネージャー起動時に適用される、IBM MQ 全体のルーチンを識別します。 これらのルーチンは、個々のキュー・マネージャーに定義されるルーチンによって、オーバーライドすることができます (このリストの項目 959 ページの『3』 を参照)。
2. ApiExit テンプレート (mqs.ini ファイル内) は、新しいキュー・マネージャーの作成時に ApiExit ローカル・セット (このリストの項目 959 ページの『3』 を参照) にコピーされる IBM MQ 全体のルーチンを識別します。
3. qm.ini ファイルにある ApiExitLocal は、特定のキュー・マネージャーに適用されるルーチンを識別します。

新しいキュー・マネージャーが作成されると、mqs.ini の ApiExitTemplate 定義が、その新しいキュー・マネージャーの qm.ini の ApiExitLocal 定義にコピーされます。 キュー・マネージャーが開始すると、ApiExitCommon および ApiExitLocal 定義が使用されます。 両方とも同じ名前のルーチンを識別する場合、ApiExitLocal 定義が ApiExitCommon 定義を置き換えます。 964 ページの『API 出口の構成』 で説明されている Sequence 属性は、スタンザで定義されているルーチンの実行順序を決定します。

IBM MQ の複数のインストール済み環境にわたる API 出口の使用

以前のバージョンの IBM MQ 用に作成された API 出口が、すべてのバージョンで動作するように使用されていることを確認してください。これは、 IBM WebSphere MQ 7.1 で出口に対して行われた変更が以前のバージョンでは動作しない可能性があるためです。 出口に加えられた変更について詳しくは、942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』を参照してください。

API 出口 amqsaem および amqsaxe 用に提供されているサンプルは、出口の作成時に必要な変更を反映しています。 アプリケーションの起動の前に、正しい IBM MQ ライブラリー (アプリケーションが関連付けら

れているキュー・マネージャーのインストール済み環境に対応するもの) がクライアント・アプリケーションにリンクされていることを確認する必要があります。

Multi *API* 出口の作成

C プログラミング言語を使用して、各 API 呼び出しに出口を作成できます。

使用可能な出口

以下のように、各 API 呼び出しで出口を使用できます。

- MQCB。指定したオブジェクト・ハンドルにコールバックを再登録し、コールバックに対するアクティベーションと変更を制御します。
- MQCTL。接続にオープンされたオブジェクト・ハンドルに対する制御アクションを実行します。
- MQCONN/MQCONNX。後続の API 呼び出しで使用される、キュー・マネージャーの接続ハンドルを提供します。
- MQDISC。キュー・マネージャーから切断します。
- MQBEGIN。グローバル作業単位 (UOW) を開始します。
- MQBACK。UOW をバックアウトします。
- MQCMIT。UOW をコミットします。
- MQOPEN。後続のアクセス用に IBM MQ リソースをオープンします。
- MQCLOSE。アクセス用に既にオープンされている IBM MQ リソースをクローズします。
- MQGET。アクセス用に既にオープンされているキューからメッセージを取得します。
- MQPUT1。メッセージをキューに入れます。
- MQPUT。アクセス用に既にオープンされているキューにメッセージを入れます。
- MQINQ。アクセス用に既にオープンされている IBM MQ リソースの属性に対して照会を実行します。
- MQSET。アクセス用に既にオープンされているキューの属性を設定します。
- MQSTAT。状況情報を取得します。
- MQSUB。特定のトピックに対して、アプリケーションのサブスクリプションを登録します。
- MQSUBRQ。サブスクリプションに対して要求を実行します。

MQ_CALLBACK_EXIT は、コールバック処理の前と後に実行する出口機能を提供します。 詳しくは、コールバック - MQ_CALLBACK_EXIT を参照してください。

API 出口の作成

API 出口内で、呼び出しは一般的な形式を取ります。

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

ここで、*call* は MQ 接頭部を省略した MQI 呼び出し名、例えば PUT、GET などです。 *parameters* は出口の機能を制御し、主に出口と外部制御ブロック MQAXP (API 出口パラメーター構造体) および MQAXC (API 出口コンテキスト構造体) の間の通信を提供します。 *context* は、API 出口が呼び出されたコンテキストを記述し、*ApiCallParameters* は MQI 呼び出しに対するパラメーターを表します。

API 出口の作成に役立つように、サンプル出口 amqsaxe0.c が提供されています。この出口は、指定したファイルにトレース・エントリーを生成します。 出口の作成時に、このサンプルを開始点として使用できます。 サンプル出口の使用方法について詳しくは、1079 ページの『API 出口サンプル・プログラム』を参照してください。

API 出口呼び出し、外部制御ブロック、および関連トピックについて詳しくは、API 出口参照を参照してください。

出口の作成、コンパイル、および構成に関する一般情報については、942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』を参照してください。

API 出口でのメッセージ・ハンドルの使用

API 出口がどのメッセージ・プロパティーに対してアクセス権を持つのかを 制御することができます。 プロパティーには ExitMsgHandle が関連付けられています。 書き込み出口で設定される プロパティーは書き込まれるメッセージに設定されますが、読み取り出口で取り出されるプロパティー はアプリケーションに戻されません。

Function を MQXF_INIT に設定し **ExitReason** を MQXR_CONNECTION に設定した MQXEP MQI 呼び出しを使用して、MQ_INIT_EXIT 出口機能を登録する とき、MQXEPO 構造体を **ExitOpts** パラメーターとして渡します。 MQXEPO 構造体に含まれる ExitProperties フィールドは、出口に対して使用可能にする プロパティーのセットを指定します。 これは、プロパティーの接頭部を表す文字ストリング (MQRFH2 フォルダー名に対応) として 指定されます。

各 API 出口は 1 つの MQAXP 構造体を受け取り、そこに ExitMsgHandle フィールドが 含まれています。 このフィールドは、IBM MQ によって生成される、接続ごとに固有の値に設定されます。 従って、同じ接続にある同じタイプまたは異なるタイプの API 出口間で、 ハンドルは変化しません。

ExitReason が MQXR_BEFORE の MQ_PUT_EXIT または MQ_PUT1_EXIT、つまり、 メッセージを書き込む前に実行される API 出口では、出口が完了するときに ExitMsgHandle と関連付けられているすべてのプロパティー (メッセージ記述子プロパティーを除く) が、書き込まれるメッセージに設定されます。 これが起こるのを 防ぐには、ExitMsgHandle を MQHM_NONE に設定します。 また、異なるメッセージ・ハンドルを 指定することもできます。

MQ_GET_EXIT および MQ_CALLBACK_EXIT では、 ExitMsg ハンドルのプロパティーがクリアされ、MQ_INIT_EXIT の登録時に ExitProperties フィールドに指定されたプロパティー (メッセージ記述子プロパティー以外) が取り込まれます。 これらのプロパティーは、読み取りアプリケーションに 使用可能にはされません。 読み取りアプリケーションが MQGMO (メッセージ読み取りのオプション) フィールド にメッセージ・ハンドルを指定した場合、そのハンドルと関連付けられているプロパティーはすべて (メッセージ記述子プロパティーも含めて)、API 出口で使用可能です。 ExitMsgHandle にプロパティーが設定されるのを 防止するには、MQHM_NONE に設定します。

注 : 出口メッセージ・プロパティーを処理するには、以下のようにします。

- MQ_GET_EXIT 関数の後には、出口に対して事前 MQ_GET_EXIT 関数を定義する必要があります。
- MQ_CALLBACK_EXIT 関数の前に、出口の MQ_CB_EXIT 関数を定義する必要があります。

V 9.3.2 IBM MQ 9.3.2 以降、 MQ-GET-EXIT および MQ_CALLBACK_EXIT に関する以前のステートメントは適用されなくなりました。

API 出口でのメッセージ・ハンドルの使用法を示すためのサンプル・プログラム amqsaem0.c が 用意されています。

関連資料

ユーザー出口、API 出口、およびインストール可能サービス参照

Multi *API* 出口のコンパイル

出口を作成した後、その出口を次のようにコンパイルし、リンクします。

次の例は、1079 ページの『API 出口サンプル・プログラム』で説明されるサンプル・プログラムに使用されるコマンドを示します。 Windows システム以外のプラットフォームの場合、サンプル API 出口コードを *MQ_INSTALLATION_PATH*/samp で見つけることができ、コンパイルされてリンクされた共有ライブラリーを *MQ_INSTALLATION_PATH*/samp/bin で見つけることができます。

Windows Windows システムの場合、サンプル API 出口コードは *MQ_INSTALLATION_PATH* \Tools\c\Samples にあります。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされたディレクトリーを表します。

注 : 64 ビット・アプリケーションのプログラミングに関するガイダンスは、64 ビット・プラットフォームでのコーディング標準にリストされています。

マルチキャスト・クライアントの場合、API 出口とデータ変換出口はクライアント・サイドで実行可能であることが必要になりました。これは、一部のメッセージがキュー・マネージャーを経由しない可能性があ

るためです。 以下のライブラリーは、サーバー・パッケージだけでなくクライアント・パッケージにも含まれています。

表 141. クライアント・パッケージとサーバー・パッケージの両方に含まれているライブラリー

オペレーティング・システム	ライブラリー
AIX	32 ビットおよび 64 ビット: libmqm.a および libmqm_r.a
IBM i	LIBMQM および LIBMQM_R
Linux	32 ビットおよび 64 ビット: libmqm.so および libmqm_r.so
Windows	32 ビットおよび 64 ビット: mqm.dll および mqm.pdb

Linux AIX *AIX and Linux システムでの API 出口のコンパイル*

AIX and Linux システムでの API 出口のコンパイル方法について例を示します。

どのプラットフォームでも、モジュールへの入り口点は MQStart です。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

AIX 上

AIX

次のいずれかのコマンドを発行して API 出口ソース・コードをコンパイルします。

32 ビット・アプリケーション

非スレッド化

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション

非スレッド化

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Linux 上

Linux

次のいずれかのコマンドを発行して API 出口ソース・コードをコンパイルします。

31 ビット・アプリケーション
非スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

32 ビット・アプリケーション
非スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション
非スレッド化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Windows システムでの API 出口のコンパイル*
サンプル API 出口プログラム amqsaxe0.c を Windows 上でのコンパイルおよびリンク

マニフェスト・ファイルは、コンパイルしたアプリケーションまたは DLL に埋め込むことのできる、バージョンその他の情報を含むオプション XML 文書です。

そのような文書がない場合は、**mt** コマンドの -manifest *manifest.file* パラメーターを省略できます。

964 ページの図 101 または 964 ページの図 102 の例でコマンドを調整して、Windows 上の amqsaxe0.c をコンパイルおよびリンクします。 コマンドは、Microsoft Visual Studio 2008、2010、または 2012 で動作します。 この例では、C:¥Program Files¥IBM¥MQ\tools\c\samples ディレクトリーが現行ディレクトリーであると想定しています。

32 ビット

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def



amqsaxe0.obj \
   /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

図 *101. 32* ビット *Windows* での *amqsaxe0.c* のコンパイルおよびリンク

64 ビット

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \



amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

図 *102. 64* ビット *Windows* での *amqsaxe0.c* のコンパイルおよびリンク

関連概念

1079 ページの『API 出口サンプル・プログラム』
サンプル API 出口は、 **MQAPI_TRACE_LOGFILE** 環境変数で定義された接頭部を持つユーザー指定ファイルに MQI トレースを生成します。

IBM i *IBM i での API 出口のコンパイル*
IBM i で API 出口をコンパイルします。

出口は次のように作成されます (C 言語の場合)。

1. CRTCMOD を使用してモジュールを作成します。 パラメーター TERASPACE(*YES *TSIFC) を組み込んで、テラスペースを使用するようにコンパイルします。
2. CRTSRVPGM を使用してモジュールからサービス・プログラムを作成します。 これをマルチスレッド API 出口のサービス・プログラム QMQM/LIBMQMZF_R にバインドする必要があります。

API 出口の構成

構成情報を変更することにより、API 出口を使用できるように IBM MQ を構成します。

構成情報を変更するには、出口ルーチンとその実行順序を定義するスタンザを変更する必要があります。この情報は、以下の方法で変更できます。

- Windows Linux Windows および Linux (x86 および x86-64 プラットフォーム) での IBM MQ Explorer の使用。
- Windows Windows で **amqmdain** コマンドを使用する。
- Multi マルチプラットフォームでの mqs.ini および qm.ini ファイルの直接的な使用。

mqs.ini ファイルには、特定のノード上のすべてのキュー・マネージャーに関連する情報が含まれます。これは次の場所にあります。

– Linux AIX AIX and Linux 上の /var/mqm ディレクトリー内。
– Windows Windows システムの HKLM\SOFTWARE\IBM\WebSphere MQ キーで指定された WorkPath 内。
– IBM i IBM i 上の /QIBM/UserData/mqm ディレクトリー内。

qm.ini ファイルには、特定のキュー・マネージャーに関連する情報が含まれます。 各キュー・マネージャーごとに、1 つのキュー・マネージャー構成ファイルがあります。これは、キュー・マネージャーが占有するディレクトリー・ツリーのルートに保持されます。 例えば、QMNAME という名前のキュー・マネージャーの構成ファイルのパスと名前は、次のとおりです。

IBM i IBM i システムの場合:

```
/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
```

Linux AIX AIX and Linux システムの場合:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

Windows Windows システムの場合:

```
C: ¥ProgramData¥IBM¥MQ\qmgrs\QMNAME\qm.ini
```

構成ファイルを編集する前に、必要な場合に復元できるようにバックアップをとっておいてください。

構成ファイルは次のどちらの方法でも編集できます。

– 自動。これには、ノード上のキュー・マネージャーの構成を変更するコマンドを使用します。
– 手動。これには、標準テキスト・エディターを使用します。

構成ファイル属性のどれかに誤った値を設定した場合、その値は無視され、問題を示すオペレーター・メッセージが表示されます その結果、その属性をまったく指定しなかった場合と同じになります。

構成するスタンザ

変更する必要があるスタンザは、以下のとおりです。

ApiExitCommon

出口の下にある IBM MQ プロパティー・ページ上の mqs.ini および IBM MQ Explorer で定義されています。

いずれかのキュー・マネージャーが開始されると、このスタンザ内の属性は読み取られ、qm.ini で定義された API 出口によってオーバーライドされます。

ApiExitTemplate

出口の下にある IBM MQ プロパティー・ページ上の mqs.ini および IBM MQ Explorer で定義されています。

キュー・マネージャーが作成されると、このスタンザの属性が ApiExitLocal スタンザの下の新たに作成された qm.ini ファイルにコピーされます。

ApiExitLocal

出口の下にあるキュー・マネージャー・プロパティー・ページの qm.ini および IBM MQ Explorer で定義されます。

キュー・マネージャーが開始されると、ここで定義された API 出口は、mqs.ini で定義されているデフォルトをオーバーライドします。

スタンザの属性

- API 出口を命名するには、以下の属性を使用します。

  Name=ApiExit_name
  MQAXP 構造体の ExitInfoName フィールドで、API 出口に渡される API 出口の記述名。

  この名前は一意で、48 文字以内であり、さらに IBM MQ オブジェクト名 (例えばキュー名) に有効な文字のみを含む必要があります。

- 実行する API 出口コードのモジュールとエントリー・ポイントを識別するには、以下の属性を使用します。

  Function=function_name
  API 出口コードを含むモジュールへの関数エントリー・ポイントの名前。 このエントリー・ポイントは MQ_INIT_EXIT 関数です。

  このフィールドの長さは MQ_EXIT_NAME_LENGTH に限定されます。

  Module=module_name
  API 出口コードを含むモジュール。

  このフィールドにモジュールの絶対パス名が入っている場合、それがそのまま使用されます。

  このフィールドにモジュール名のみが含まれている場合、そのモジュールは qm.ini の ExitPath 内の ExitsDefaultPath 属性を使用して位置指定されます。

  異なるスレッド化ライブラリーをサポートするプラットフォームで、API 出口モジュールの非スレッド化バージョンとスレッド化バージョンの両方を提供する必要があります。 スレッド化バージョンは、接尾部 _r を持っている必要があります。 IBM MQ アプリケーション・スタブのスレッド化バージョンは、指定のモジュールがロードされる前にその名前に _r を暗黙的に追加します。

  このフィールドの長さは、プラットフォームがサポートする最大パス長に限定されます。

- 必要により出口にデータを渡すには、以下の属性を使用します。

  Data=data_name
  MQAXP 構造体の ExitData フィールドで API 出口に渡されるデータ。

  この属性を指定すると、先行および末尾のブランクが除去されて残りのストリングは 32 文字に切り捨てられ、その結果が出口に渡されます。 この属性を省略すると、デフォルト値の 32 文字のブランクが出口に渡されます。

  このフィールドの最大長は、32 文字です。

- 他の出口に関するこの出口のシーケンスを識別するには、以下の属性を使用します。

  Sequence=sequence_number
  その他の API 出口に関してこの API 出口が呼び出される順序。 小さなシーケンス番号の出口は、より大きなシーケンス番号の出口よりも先に呼び出されます。 出口のシーケンス番号は連続である必要はありません。 つまり、1、2、3 の順序は、7、42、1096 の順序と同じ結果となります。 2 つの出口のシーケンス番号が同じ場合は、キュー・マネージャーが最初に呼び出す出口を決定します。 MQAXP の ExitChainAreaPtr で示される ExitChainArea に時刻またはマーカーを設定するかまたは独自のログ・ファイルを作成して、イベントの後に呼び出された出口を判別できます。

  この属性は、符号なし数値です。

サンプル・スタンザ

サンプル mqs.ini ファイルには、次のスタンザが含まれています。

ApiExitTemplate
このスタンザは、記述名 OurPayrollQueueAuditor、モジュール名 auditor、およびシーケンス番号 2 を持つ出口を定義します。 123 のデータ値が出口に渡されます。

ApiExitCommon
このスタンザは、記述名 MQPoliceman、モジュール名 tmqp、およびシーケンス番号 1 を持つ出口を定義します。 渡されたデータは命令 (CheckEverything) です。

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

以下のサンプル qm.ini ファイルには、記述名 ClientApplicationAPIchecker、モジュール名 ClientAppChecker、および順序番号 3 が指定された出口の ApiExitLocal 定義が含まれています。

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

メッセージング・チャネルのためのチャネル出口プログラム

ここでの一連のトピックには、メッセージ・チャネル用の IBM MQ チャネル出口プログラムについての情報が含まれています。

メッセージ・チャネル・エージェント (MCA) でデータ変換出口を呼び出すこともできます。 データ変換出口の作成の詳細については、988 ページの『データ変換出口の作成』を参照してください。

この情報の一部は、IBM MQ MQI clients をキュー・マネージャーに接続する MQI チャネルの出口にも適用されます。 詳細については、MQI チャネル用のチャネル出口プログラムを参照してください。

チャネル出口プログラムは、MCA プログラムの実行中に定義された位置で呼び出されます。

一部のユーザー出口プログラムは、相互補完的に機能します。 例えば、伝送されるメッセージを暗号化するために、送信側 MCA によってあるユーザー出口プログラムが呼び出された場合、その逆の処理を行うために受信側で補完的な処理が実行される必要があります。

967 ページの表 142 には、チャネル・タイプごとに使用可能なチャネル出口のタイプを示しています。

表 *142.* チャネル・タイプで使用可能なチャネル出口

チャネル・タイプ	メッセージ出口	Message-retry exit (メッセージ再試行出口)	受信出口	セキュリティー出口	送信出口	Auto-definition exit (自動定義出口)
送信側チャネル	はい		はい	はい	はい	
サーバー・チャネル	はい		はい	はい	はい	
クラスター送信側チャネル	はい		はい	はい	はい	はい

表 142. チャネル・タイプで使用可能なチャネル出口 (続き)						
チャネル・タイプ	**メッセージ出口**	**Message-retry exit (メッセージ再試行出口)**	**受信出口**	**セキュリティー出口**	**送信出口**	**Auto-definition exit (自動定義出口)**
受信側チャネル	はい	はい	はい	はい	はい	はい
要求側チャネル	はい	はい	はい	はい	はい	
クラスター受信側チャネル	はい	はい	はい	はい	はい	はい
クライアント接続チャネル			はい	はい	はい	
サーバー接続チャネル			はい	はい	はい	はい

注 : z/OS

1. z/OS では、自動定義出口はクラスター送信側チャネルおよびクラスター受信側チャネルにのみ適用されます。

クライアント上でチャネル出口を実行する場合、MQSERVER 環境変数は使用できません。 代わりに、クライアント・チャネル定義テーブルに説明されているクライアント・チャネル定義テーブル (CCDT) を作成し、参照します。

処理の概要

MCA がチャネル出口プログラムを使用する方法の概要

始動すると MCA は、開始ダイアログを交換して処理を同期化させます。 次に、セキュリティー出口を含むデータ交換に切り替えます。 開始フェーズが完了してメッセージを転送できるようにするために、これらの出口は正常に終了する必要があります。

セキュリティー検査フェーズは、968 ページの図 103 に示すような処理ループです。

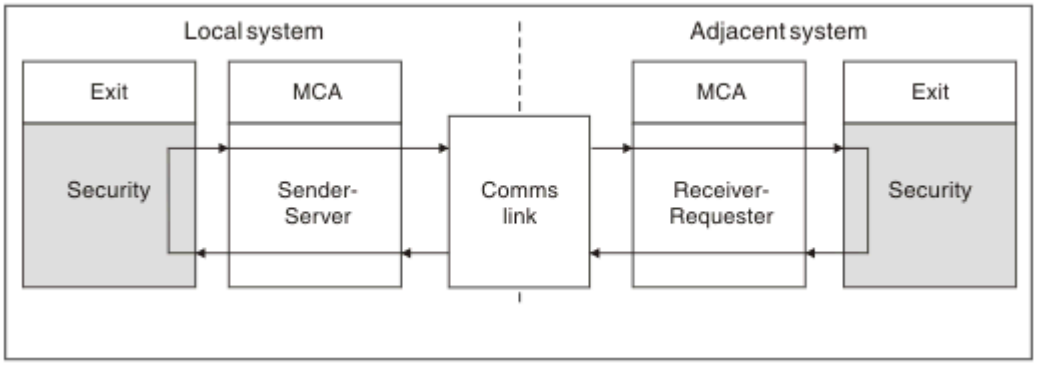


図 *103.* セキュリティー出口ループ

969 ページの図 104 に示すように、メッセージ転送フェーズ中に、送信側 MCA が伝送キューからメッセージを読み取り、メッセージ出口を呼び出し、次に送信出口を呼び出し、受信側 MCA にメッセージを送信します。

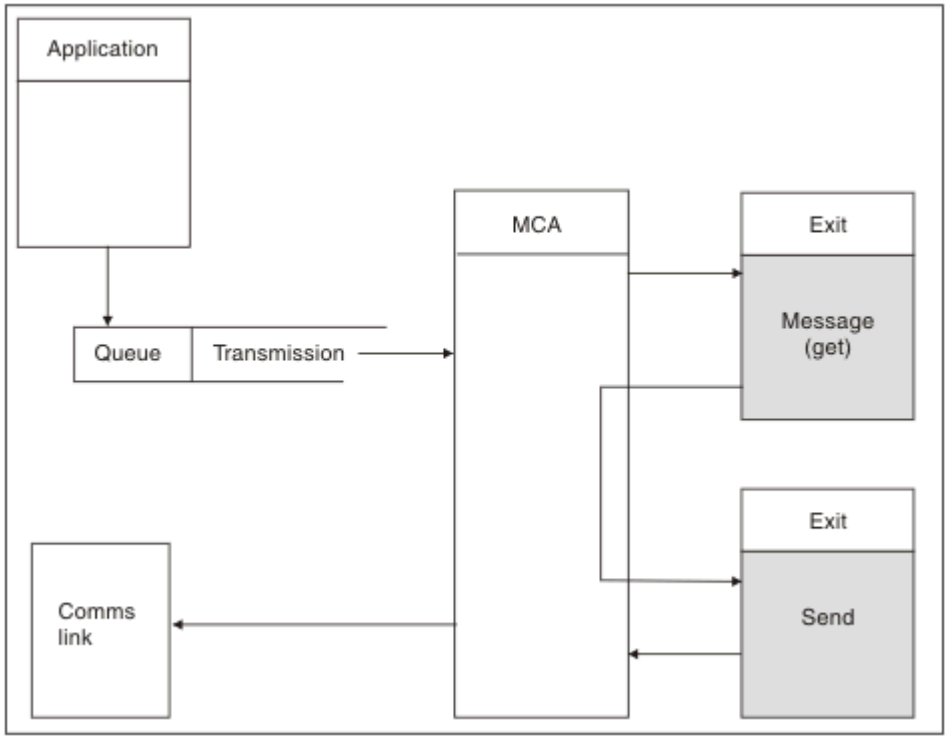


図 *104.* メッセージ・チャネルの送信側での送信出口の例

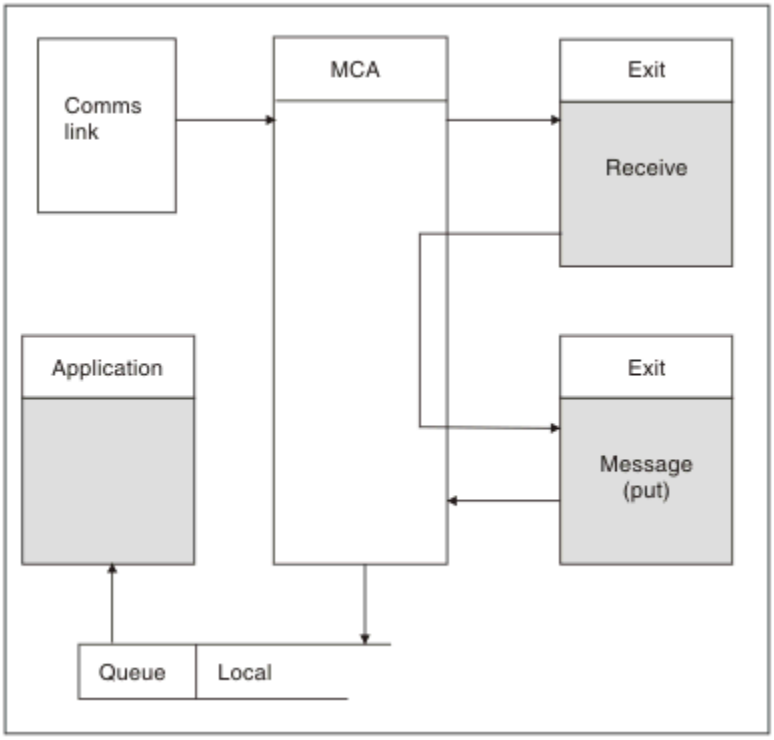


図 *105.* メッセージ・チャネルの受信側での受信出口の例

969 ページの図 105 に示すように、受信側 MCA は通信リンクからメッセージを受信し、受信出口を呼び出し、次にメッセージ出口を呼び出し、メッセージをローカル・キューに入れます (メッセージ出口が呼び出される前に、受信出口を複数回呼び出すことができます)。

チャネル出口プログラムの作成

チャネル出口プログラムの作成に役立つ情報を 以下に記載します。

ユーザー出口プログラムおよびチャネル出口プログラムでは、すべての MQI 呼び出し (ただし、この後のセクションで説明されているものを除く) を使用できます。 MQ V7 以降では、MQCXP 構造体バージョン 7 以降に接続ハンドル hConn が含まれているため、MQCONN を発行する代わりにこれを使用することができます。 これより前のバージョンで接続ハンドルを取得するには、MQCONN を発行する必要があります (チャネルそのものが既にキュー・マネージャーに接続されているため MQRC_ALREADY_CONNECTED 警告が返されても構いません)。

チャネル出口はスレッド・セーフでなければならないことに注意してください。

クライアント接続チャネル上の出口の場合、出口が接続しようとするキュー・マネージャーは、その出口がどのようにリンクされているかによって異なります。 出口が MQM.LIB (IBM i の場合は QMQM/LIBMQM) にリンクされている場合に、 MQCONN 呼び出しでキュー・マネージャー名を指定しないと、 出口はそのシステム上のデフォルトのキュー・マネージャーに接続しようとします。 出口が MQM.LIB (IBM i の場合は QMQM/LIBMQM) にリンクされていて、MQCD の QMgrName フィールドを介して出口に渡されたキュー・マネージャーの名前を指定すると、出口はそのキュー・マネージャーに接続しようとします。 出口が MQIC.LIB またはそれ以外のライブラリーにリンクされている場合は、キュー・マネージャー名を指定するかどうかに関係なく MQCONN 呼び出しは失敗します。

チャネル出口に渡された hConn に関連付けられているトランザクションの状態を変更することは 避ける必要があります。つまり、MQCMIT、 MQBACK、または MQDISC verb にチャネル hConn を指定して使用しないでください。また、 チャネル hConn を指定して MQBEGIN verb を使用することもできません。

MQCONNX に MQCNO_HANDLE_SHARE_BLOCK または MQCNO_HANDLE_SHARE_NO_BLOCK を指定して、 新規 IBM MQ 接続を作成する場合は、 ユーザーの責任において、その接続が適切に管理され、キュー・マネージャーから正常に切断されるようにします。 例えば、チャネル出口が、呼び出される度に (切断することなく) キュー・マネージャーへの接続を新規作成すると、 接続ハンドルが蓄積され、エージェント・スレッドの数が増加することになります。

出口は MCA 自体と同じスレッドで実行され、同じ接続ハンドルを使用します。 つまり、出口は MCA と同じ UOW 内で実行され、同期点の下で行われた呼び出しは、すべてバッチの終わりにチャネルによってコミットまたはバックアウトされます。

したがって、チャネル・メッセージ出口は、元のメッセージが入っているバッチがコミットされたときにのみ そのキューにコミットされる通知メッセージを送信できます。 つまり、チャネル・メッセージ出口から同期点 MQI 呼び出しを発行することが可能です。

チャネル出口は、MQCD のフィールドを変更できます。 ただし、リストされている状況を除いて、これらの変更に応じて動作することはありません。 チャネル出口プログラムが MQCD データ構造体のフィールドを変更する場合、新規の値は、IBM MQ チャネル・プロセスにより無視されます。 しかし新規の値は MQCD に残り、出口チェーンの残りの出口、およびチャネル・インスタンスを共用する会話に渡されます。 詳しくは、チャネル出口での MQCD フィールドの変更を参照してください。

また、C で作成したプログラムの場合、チャネル出口プログラムでは再入不能 C ライブラリー関数は使用しないでください。

Linux AIX 複数のチャネル出口ライブラリーを同時に使用する場合、2 つの異なる出口のコードに同一名の関数が含まれていれば、一部の UNIX and Linux プラットフォームでは問題が発生する可能性があります。 チャネル出口がロードされる場合、動的ローダーは出口ライブラリーの関数名を、ライブラリーのロード先のアドレスに解決します。 2 つの出口ライブラリーが別個の関数を定義し、それらが偶然同一の名前になった場合、この解決プロセスでは、一方のライブラリーの関数名が他方の関数を使用するように誤って解決してしまう可能性があります。 この問題が起きる場合は、影響を受けないように、必要な出口と MQStart 関数だけしかエクスポートしないようにリンカーに指定してください。 他の関数には、 それ自身の出口ライブラリー以外の関数によって使用されないようにするために、ローカル可視性を付与する必要があります。 詳細については、リンカーに関する資料 を参照してください。

すべての出口は、チャネル出口パラメーター構造体 (MQCXP)、チャネル定義構造体 (MQCD)、準備済みのデータ・バッファー、データ長パラメーター、バッファー長パラメーターを指定して呼び出されます。 したがって、バッファー長を超過しないように注意してください。

- メッセージ出口の場合、チャネル間で送信されるのに必要な最大長のメッセージと MQXQH 構造体の長さが許可されます。
- 送信出口および受信出口の場合、許可される最大バッファーは次のとおりです。

  LU 6.2
  32 KB

  TCP:
  IBM i IBM i 16 KB

  IBM i その他 32 KB

  注 : 使用可能な最大の長さは、この長さよりも 2 バイト小さくなります。 詳細については、MaxSegmentLength で戻される値をチェックしてください。 MaxSegmentLength の詳細については、MaxSegmentLength を参照してください。

  NetBIOS:
  64 KB

  SPX:
  64 KB

  注 : 送信側チャネルの受信出口および受信側の送信出口は、TCP 用に 2 KB のバッファーを使用します。
- セキュリティー出口の場合、分散キューイング機構は、4000 バイトのバッファーを割り当てます。

出口から、関連パラメーターと共に代替バッファーを戻すことは可能です。 呼び出しの詳細については、967 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

z/OS *z/OS* におけるチャネル出口プログラムの作成

以下の情報を使用すると、z/OS のチャネル出口プログラムの作成およびコンパイルに役立ちます。

次の場合には、z/OS LINK による場合のように、出口が開始します。

- 許可がない問題プログラム状態
- 1 次アドレス・スペース制御モード
- 非仮想記憶間モード
- 非アクセス登録モード
- 31 ビット・アドレッシング・モード

リンク・エディット済みモジュールは、チャネル・イニシエーター・アドレス・スペース・プロシージャーの CSQXLIB DD ステートメントで指定されたデータ・セット内に格納しなければなりません。ロード・モジュール名はチャネル定義内で出口名として指定されたものです。

z/OS のチャネル出口を作成するときは、以下の規則が適用されます。

- 出口はアセンブラーまたは C で書く必要があります。C が使用されている場合は、「z/OS C/C++ プログラミング・ガイド」で説明されているシステム出口の C システム・プログラミング環境に準拠している必要があります。
- 出口は、CSQXLIB DD ステートメントで定義された許可されていないライブラリーからロードされます。CSQXLIB に DISP=SHR が指定されている場合は、チャネル・イニシエーターの実行中に出口を更新できます。 チャネルを再始動すると、新しいバージョンが使用されます。
- 出口は、再入可能であり、仮想ストレージ内のどこでも実行可能である必要があります。
- 出口は、戻り時に、環境をリセットし、入り口の時と同じ状態にする必要があります。
- 出口は、取得したストレージをすべて解放するか、または次の出口呼び出しによって解放するようにする必要があります。

  複数の呼び出しに渡って持続するストレージの場合は、z/OS STORAGE サービスを使用するか、あるいはシステム・プログラミング C の 4kmalc ライブラリー関数を使用します。

  この機能に関する小差いは、4kmalc() --「割り当てページ」-「割り当てられたストレージ」を参照してください。

- すべての IBM MQ MQI 呼び出し (MQCMIT または CSQBCMT および MQBACK または CSQBBAK を除く) を使用できます。 それらは、MQCONN (キュー・マネージャー名はブランク) の後に入れます。 これらの呼び出しを使用する場合は、出口をスタブ CSQXSTUB を指定してリンク・エディットする必要があります。

  この規則の例外は、セキュリティー・チャネル出口が MQI 呼び出しのコミットとバックアウトを発行できることです。 このような呼び出しを発行するには、MQCMIT または CSQBCMT および MQBACK または CSQBBAK の代わりに動詞として CSQXCMT および CSQXBAK をコーディングします。
- IBM WebSphere MQ 7.0 以降のスタブ CSQXSTUB を使用するすべての出口は、PDS-E 形式の CSQXLIB ロード・ライブラリーでリンク・エディットされている必要があります。
- システム・サービスを使用すると他の一部またはすべてのチャネルの処理に重大な影響を与えることになるため、出口は待機を生じさせるシステム・サービスを使用してはなりません。 通常、多くのチャネルが単一の TCB の下で実行します。 待機を生じさせる出口で何らかの処理を行い、かつ MQXWAIT を使用しない場合は、それらすべてのチャネルが待機状態になります。 チャネルが待機しても機能上は問題ありませんが、パフォーマンスにはマイナスの影響となる可能性があります。 ほとんどの SVC には待機が含まれているので、それらの使用を避ける必要があります。ただし、以下の SVC は例外です。

  – GETMAIN/FREEMAIN/STORAGE

  – LOAD/DELETE

  したがって、一般的には、SVC、PC、および I/O は回避されます。 代わりに、MQXWAIT 呼び出しを使用します。
- ESTAE や SPIE のエラー処理は IBM MQ が 実行するエラー処理に干渉することがあるので、接続するサブタスク内でなければ、出口は ESTAE も SPIE も発行しません。 つまり、IBM MQ がエラーからリカバリーできない可能性がある、または出口プログラムがエラー情報をすべては受け取らない可能性があるということです。
- MQXWAIT 呼び出し (MQXWAIT を参照) は、入出力および他のイベントを待機するサービスを提供します。このサービスを使用する場合、出口はリンケージ・スタックを使用してはなりません。

  ノンブロッキング機能を提供しない入出力および他の機能または待機する ECB の場合、別個のサブタスクを ATTACH して接続し、その完了を MQXWAIT まで待機する必要があります。この技法に伴う処理のため、この機能の使用はセキュリティー出口に限られます。
- MQDISC MQI 呼び出しによって、出口プログラム内で暗黙的コミットが実行されることはありません。チャネル・プロセスのコミットは、チャネル・プロトコルが要求する場合にのみ行われます。

以下の出口サンプルは、IBM MQ for z/OS と共に提供されています。

CSQ4BAX0
このサンプルはアセンブラーで書かれており、MQXWAIT の使用法を図示しています。

CSQ4BCX1 および CSQ4BCX2
これらのサンプルは C 言語で書かれており、パラメーターのアクセス方法を図示しています。

CSQ4BCX3 および CSQ4BAX3
これらのサンプルは、それぞれ C 言語およびアセンブラーで書かれています。

CSQ4BCX3 サンプルは、プリコンパイルにより SCSQAUTH LOADLIB に入れられます。 このサンプルは、出口自体において変更の必要なしで機能するはずです。 LOADLIB (例えば MY.TEST.LOADLIB という名前の) を作成し、そこに SCSQAUTH(CSQ4BCX3) メンバーをコピーすることができます。

クライアント接続にセキュリティー出口をセットアップするには、以下の手順を実行します。

1. チャネル・イニシエーターが使用するユーザー ID 用に有効な OMVS セグメントを確立します。

   これにより、IBM MQ for z/OS チャネル・イニシエーターは、出口処理を促進するために、z/OS UNIX System Services (z/OS UNIX) ソケット・インターフェースで TCP/IP を使用できるようになります。接続クライアントのユーザー ID に OMVS セグメントを定義することは不要である点にご注意ください。
2. 出口コード自体が、プログラム制御環境でのみ実行することを確認してください。

これは、CHINIT アドレス・スペースにロードされるあらゆるものがプログラム制御されたライブラリー (つまり STEPLIB 内のすべてのライブラリー)、CSQXLIB で指定されたライブラリー、および以下からロードされる必要があることを意味します。

```
++hlq++.SCSQANLx
++hlq++.SCSQMVR1
++hlq++.SCSQAUTH
```

プログラム制御されたロード・ライブラリーを設定するには、次の例のようなコマンドを使用します。

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
```

その後、次のコマンドを発行して、プログラム制御環境をアクティブ化またはリフレッシュできます。

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. 次のコマンドを発行して、出口 LOADLIB を CSQXLIB DD (CHINIT 始動プロシージャー内) に追加します。

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

これにより、指定されたチャネルの出口がアクティブになります。

4. 外部セキュリティー・マネージャー (ESM) は、プログラム制御されるその他すべてのライブラリーをリストしますが、ESM も C ライブラリーもプログラム制御される必要がありません。

サンプル CSQ4BCX3 を使用したセキュリティー出口のセットアップについて詳しくは、IBM MQ for z/OS サーバー接続チャネルを参照してください。

CSQ4BCX4
このサンプルは C で作成され、MQCXP 内の **RemoteProduct** フィールドおよび **RemoteVersion** フィールドを使用してデモンストレーションされます。

関連概念

973 ページの『IBM i でのチャネル出口プログラムの作成』
以下の情報を使用すると、IBM i のチャネル出口プログラムの作成およびコンパイルに役立ちます。

974 ページの『AIX, Linux, and Windows におけるチャネル出口プログラムの作成』
以下の情報は、AIX, Linux, and Windows システム用のチャネル出口プログラムを作成する際に使用することができます。

関連資料

IBM MQ for z/OS サーバー接続チャネル

IBM i *IBM i でのチャネル出口プログラムの作成*

以下の情報を使用すると、IBM i のチャネル出口プログラムの作成およびコンパイルに役立ちます。

出口は、ILE C、ILE RPG、または ILE COBOL の言語で作成されたプログラム・オブジェクトです。 出口プログラム名とそれらのライブラリー名は、チャネル定義で指定されます。

出口プログラムを作成およびコンパイルするときは、以下の条件を考慮してください。

- プログラムはスレッド・セーフであり、ILE C、ILE RPG、または ILE COBOL コンパイラーによって作成されている必要があります。 ILE RPG の場合は、THREAD(*SERIALIZE) 制御仕様を指定する必要があります。ILE COBOL の場合は、PROCESS ステートメントの THREAD オプションに SERIALIZE を指定する必要があります。 またプログラムは、ILE C および ILE RPG の場合は QMQM/LIBMQM_R、ILE COBOL の場合は AMQ0STUB_R という、スレッド化された IBM MQ ライブラリーにバインドする必要があります。RPG または COBOL アプリケーションをスレッド・セーフにすることに関する追加情報については、該当する言語のプログラマーズ・ガイドを参照してください。

- IBM MQ for IBM i では、出口プログラムのテラスペース・サポートが有効になっている必要があります (テラスペースは OS/400 V4R4 に導入された共有メモリーの形式です)。 ILE RPG および COBOL コンパイラーの場合、OS/400 V4R4 以降でコンパイルされたすべてのプログラムでは、サポートが有効になっています。 C の場合、プログラムは CRTCMOD または CRTBNDC コマンドの TERASPACE(*YES *TSIFC) オプションを指定してコンパイルする必要があります。
- ポインターをそれ独自のバッファー・スペースに戻す出口では、指示されているオブジェクトが、チャネル出口プログラムの時間幅を超えて存在するようにする必要があります。 ポインターは、プログラム・スタックの変数のアドレスや、プログラム・ヒープの変数のアドレスとすることができません。 このポインターは、システムから獲得されるものでなければなりません。 例として、ユーザー出口に作成されるユーザー・スペースがあります。 チャネル出口プログラムによって割り振られたデータ域が、プログラムの終了時に引き続き MCA で使用可能になるようにするには、チャネル出口は呼び出し側の活動化グループまたは名前付き活動化グループ内で実行される必要があります。 これは、CRTPGM の ACTGRP パラメーターをユーザー定義の値または *CALLER に設定することによって行います。 プログラムをこのように作成すれば、チャネル出口プログラムは動的メモリーを割り振って、このメモリーを指すポインターを MCA に戻すことができるようになります。

関連概念

974 ページの『AIX, Linux, and Windows におけるチャネル出口プログラムの作成』
以下の情報は、AIX, Linux, and Windows システム用のチャネル出口プログラムを作成する際に使用することができます。

971 ページの『z/OS におけるチャネル出口プログラムの作成』
以下の情報を使用すると、z/OS のチャネル出口プログラムの作成およびコンパイルに役立ちます。

ALW *AIX, Linux, and Windows* におけるチャネル出口プログラムの作成

以下の情報は、AIX, Linux, and Windows システム用のチャネル出口プログラムを作成する際に使用することができます。

942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』 に概説されている手順に従います。 以下のチャネル出口固有の情報から、該当するものを使用してください。

出口は C で作成されなければならず、Windows の DLL です。

出口にダミー MQStart() ルーチンを定義して、ライブラリーのエントリー・ポイントとして `MQStart` を指定します。 974 ページの図 106 に、プログラムへの入り口のセットアップ方法を示しています。

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

図 *106.* チャネル出口のサンプル・ソース・コード

Visual C++ を使用して Windows 用のチャネル出口を作成する場合は、ユーザー独自の DEF ファイルを作成する必要があります。 作成方法を 974 ページの図 107 に示します。 チャネル出口プログラムの作成について詳しくは、970 ページの『チャネル出口プログラムの作成』 を参照してください。

```
EXPORTS
ChannelExit
```

図 *107. Windows* 用のサンプル *DEF* ファイル

関連概念

973 ページの『IBM i でのチャネル出口プログラムの作成』

以下の情報を使用すると、IBM i のチャネル出口プログラムの作成およびコンパイルに役立ちます。

971 ページの『z/OS におけるチャネル出口プログラムの作成』
以下の情報を使用すると、z/OS のチャネル出口プログラムの作成およびコンパイルに役立ちます。

チャネル・セキュリティー出口プログラム
セキュリティー出口プログラムを使用して、チャネルの反対側のパートナーが正しいかを確かめることができます。 これを認証と呼びます。

チャネルがセキュリティー出口を使用する必要があることを指定するには、チャネル定義の **SCYEXIT** フィールドに出口名を指定します。

注 : チャネル認証レコードを使用して認証を行うこともできます。 チャネル認証レコードの高い柔軟性により、 特定のユーザーおよびチャネルからのキュー・マネージャーへのアクセスを防止し、 リモート・ユーザーを IBM MQ ユーザー ID にマップすることができます。 IBM MQ では TLS もサポートしているため、ユーザー認証や、使用データの暗号化およびデータ保全性検査を行うことができます。 TLS の詳細については、IBM MQ での TLS セキュリティー・プロトコルを参照してください。 これらよりもさらに高度な (または異なる) 形式のセキュリティー処理や、別のタイプの検査やセキュリティー・コンテキストの確立を必要とする場合は、 セキュリティー出口を作成することを検討してください。

サブジェクト DN 属性および発行者 DN 属性は、次のチャネル状況属性に表示されます。

- SSLPEER (PCF セレクター MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF セレクター MQCACH_SSL_CERT_ISSUER_NAME)

これらの値はチャネル状況コマンドによって返されるだけでなく、以下にリストするチャネル・セキュリティー出口にもデータが渡されます。

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

セキュリティー出口は C または Java で作成できます。

チャネル・セキュリティー出口プログラムは、 MCA 処理サイクル内の以下の箇所で呼び出されます。

- MCA 開始および終了時
- チャネルの開始時に初期のデータ折衝が終了した直後。 チャネルの受信側またはサーバーは、リモート側のセキュリティー出口へ送達されるメッセージを提供することによって、リモート側とのセキュリティー・メッセージ交換を開始する場合があります。 また、このメッセージ交換を拒否することもあります。 リモート側から受信したセキュリティー・メッセージを処理するために、出口プログラムが再度呼び出されます。
- チャネルの開始時に初期のデータ折衝が終了した直後。 チャネルの送信側または要求側は、リモート側から送信されてきたセキュリティー・メッセージを処理するか、リモート側がセキュリティー交換を開始できない場合にそれを開始します。 その後受信する可能性のあるすべてのセキュリティー・メッセージを処理するために、出口プログラムが 再度呼び出されます。

要求側チャネルが MQXR_INIT_SEC で呼び出されることはありません。 チャネルは、セキュリティー出口プログラムがあることをサーバーに通知します。そこでサーバーはセキュリティー出口を開始する機会を得ます。 セキュリティー出口がない場合、サーバーは要求側に通知し、長さゼロのフローが出口プログラムに返されます。

注 : ゼロ長のセキュリティー・メッセージを送信することは避けてください。

セキュリティー出口プログラムによって交換されたデータの例を、図 976 ページの図 108 から 978 ページの図 111 に示します。 これらの例は、受信側のセキュリティー出口と送信側のセキュリティー出口が関連して発生するイベントの順序を示しています。 図の中で行が連続することは、時間の経過を表しています。 場合によっては、受信側と送信側のイベントが相関されないために、イベントが同時に発生したりあるいは別々のときに発生することもあります。 また、別の場合には、ある出口プログラムのイベントが原因となって、他の出口プログラムで補足的なイベントが後から発生することもあります。 例えば、976 ページの図 108 では次のようになっています。

1. 受信側と送信側がそれぞれ MQXR_INIT によって呼び出されたものの、それらの呼び出しが相関されなかったために、それらの呼び出しが同時に行われるか別々に行われるかわかりません。

2. 次に、受信側が MQXR_INIT_SEC によって呼び出されましたが、送信側出口で補足的なイベントを必要としない MQXCC_OK が戻されました。
3. 次に、送信側が MQXR_INIT_SEC によって呼び出されました。 この呼び出しは、MQXR_INIT_SEC によって呼び出された受信側の呼び出しとは相関されません。 送信側は MQXCC_SEND_SEC_MSG を戻し、受信側出口で補足的なイベントが発生します。
4. その後、受信側が MQXR_SEC_MSG によって呼び出され、MQXCC_SEND_SEC_MSG を戻すと、 送信側出口で補足的なイベントが発生します。
5. その後、送信側が MQXR_SEC_MSG によって呼び出され、受信側出口での補足的なイベントを必要としない MQXCC_OK を戻します。

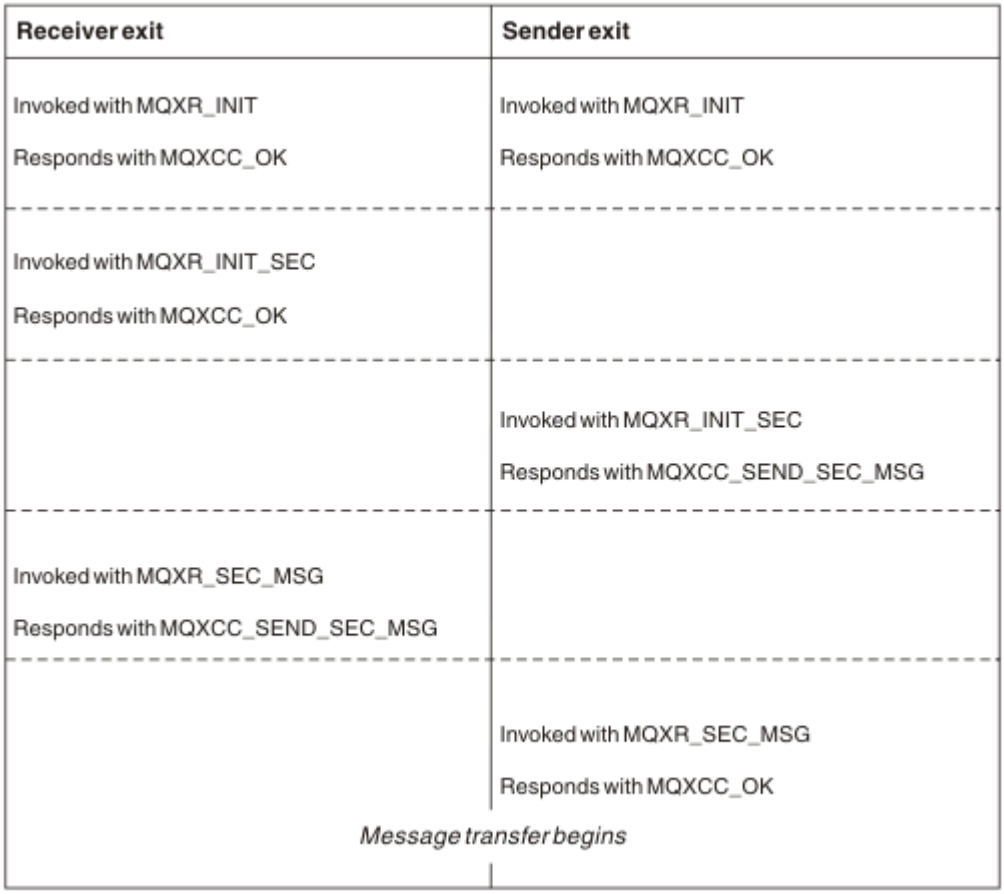


図 *108.* 合意に基づいて送信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

図 *109.* 合意なしで送信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

図 *110.* 合意に基づいて受信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

図 *111.* 合意なしで受信側から開始される交換

チャネル・セキュリティー出口には、伝送ヘッダーを除き、セキュリティー出口プログラムが作成したセキュリティー・データが入っているエージェント・バッファーが渡されます。 このデータは、どちらかのチャネルでセキュリティー検査を行えるような、任意の適切なデータになります。

メッセージ・チャネルの送信側および受信側の両方のセキュリティー出口プログラムは、どの呼び出しにも次の 2 つの応答コードのうちいずれかを返すことができます。

- セキュリティー交換はエラーなしで終了
- チャネルを非表示にして、閉じる

注 :

1. チャネル・セキュリティー出口は、通常は対で作動します。 該当するチャネルを定義するときには必ず、チャネルの両側で互換性のある出口プログラムを指定するようにしてください。
2. IBM i IBM i では、 `Use adopted authority` (USEADPAUT = *YES) でコンパイルされたセキュリティー出口プログラムは、QMQM 権限または QMQMADM 権限を借用できます。 この機能を出口が使用することによって、システムにセキュリティー上のリスクがもたらされることのないように注意してください。
3. チャネルの反対側で証明書が提供される TLS チャネルでは、セキュリティー出口は、SSLPeerNamePtr がアクセスする MQCD フィールドでこの証明書の所有者の識別名を受け取り、SSLRemCertIssNamePtr がアクセスする MQCXP フィールドで発行者の識別名を受け取ります。 この名前を保管する目的は以下のとおりです。
   - TLS チャネル間のアクセスを制限すること
   - この名前に基づいて MQCD.MCAUserIdentifier を変更すること

関連概念

Transport Layer Security (TLS) の概念

関連資料

チャネル認証レコード

セキュリティー出口の作成

セキュリティー出口のスケルトン・コードを使用して、セキュリティー出口を作成できます。

979 ページの図 112 は、セキュリティー出口の作成方法を示しています。

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

図 *112.* セキュリティー出口のスケルトン・コード

標準の IBM MQ エントリー・ポイント MQStart が存在しなければなりませんが、関数の実行には必要とされません。 関数の名前 (この例では EntryPoint) は変更可能ですが、ライブラリーのコンパイルおよびリンク時にはその関数をエクスポートする必要があります。 上記の例のように、ポインター pChannelExitParms は PMQCXP にキャストする必要があり、pChannelDefinition は PMQCD にキャストする必要があります。 チャネル出口の呼び出しおよびパラメーターの使用に関する一般情報については、MQ_CHANNEL_EXIT を参照してください。 これらのパラメーターは、セキュリティー出口で以下のように使用されます。

PMQVOID pChannelExitParms
入出力

MQCXP 構造体へのポインター - フィールドにアクセスするために PMQCXP にキャストします。 この構造体は、出口と MCA との間の通信に使用されます。 MQCXP の以下のフィールドは、セキュリティー出口には特に重要です。

ExitReason
セキュリティー出口に、セキュリティー交換における現行状態を通知する。 取るべき処置を決定する際に使用される。

ExitResponse
セキュリティー交換の次のステージを指示する、MCA への応答。

ExitResponse2
MCA がセキュリティー出口の応答を解釈する方法を制御する、追加の制御フラグ。

ExitUserArea
呼び出しから次の呼び出しまでの間、状態を維持するためにセキュリティー出口が使用できる、16 バイト (最大) のストレージ。

ExitData
チャネル定義の SCYDATA フィールドで指定されるデータを含みます (32 バイト。右側にブランクで埋め込み)。

PMQVOID pChannelDefinition
入出力

MQCD 構造体へのポインター - フィールドにアクセスするために PMQCD にキャストします。 このパラメーターにはチャネルの定義が含まれます。 MQCD の以下のフィールドは、セキュリティー出口には特に重要です。

ChannelName
チャネル名 (20 バイト。右側にブランクで埋め込み)。

ChannelType
チャネル・タイプを定義するコード。

MCA ユーザー ID
このグループの 3 つのフィールドは、チャネル定義で指定された MCAUSER フィールドの値に初期化されます。 セキュリティー出口により指定された、これらのフィールドのユーザー ID はすべて、アクセス制御に使用されます (SDR、SVR、CLNTCONN、および CLUSSDR チャネルには適用されない)。

MCAUserIdentifier
ID の最初の 12 バイト (右側にブランクで埋め込み)。

LongMCAUserIdPtr
フルサイズの ID (必ずしも NULL 終了ではない) が含まれるバッファーへのポインターは、MCAUserIdentifier に優先します。

LongMCAUserIdLength
LongMCAUserIdPtr により指されるストリングの長さ - LongMCAUserIdPtr が設定されている場合に設定する必要があります。

リモート・ユーザー ID
CLNTCONN/SVRCONN のチャネルのペアにのみ適用されます。 CLNTCONN セキュリティー出口が定義されていない場合、これら 3 つのフィールドはクライアント MCA により初期化されます。そのため、認証用および MCA ユーザー ID の指定時に SVRCONN セキュリティー出口が使用できる、クライアントの環境からのユーザー ID が含まれる場合があります。 CLNTCONN セキュリティー出口が定義されている場合、これらのフィールドは初期化されず、CLNTCONN セキュリティー出口により設定できます。またはセキュリティー・メッセージをクライアントからサーバーへのユーザー ID の受け渡しに使用できます。

RemoteUserIdentifier
ID の最初の 12 バイト (右側にブランクで埋め込み)。

LongRemoteUserIdPtr
フルサイズの ID (必ずしも NULL 終了ではない) が含まれるバッファーへのポインターは、RemoteUserIdentifier に優先します。

LongRemoteUserIdLength
LongRemoteUserIdPtr により指されるストリングの長さ - LongRemoteUserIdPtr が設定されている場合に設定する必要があります。

PMQLONG pDataLength
入出力

MQLONG へのポインター。 セキュリティー出口の呼び出し時に、AgentBuffer に含まれるセキュリティー出口の長さが入ります。 セキュリティー出口により、AgentBuffer または ExitBuffer で送信されるメッセージの長さに設定する必要があります。

PMQLONG pAgentBufferLength
入力

MQLONG へのポインター。 セキュリティー出口の呼び出し時に AgentBuffer に含まれるデータの長さ。

PMQVOID pAgentBuffer
入出力

セキュリティー出口の呼び出し時に、これはパートナー出口から送信されるメッセージを指します。 MQCXP 構造体内の ExitResponse2 に MQXR2_USE_AGENT_BUFFER フラグが設定されている場合 (デフォルト)、セキュリティー出口ではこのパラメーターを、送信されるメッセージ・データを指すように設定する必要があります。

PMQLONG pExitBufferLength
入出力

MQLONG へのポインター。 このパラメーターはセキュリティー出口の最初の呼び出し時に 0 に初期化され、戻される値は、セキュリティー交換中のセキュリティー出口への呼び出しから呼び出しまでの間、維持されます。

PMQPTR pExitBufferAddr
入出力

このパラメーターはセキュリティー出口の最初の呼び出し時に NULL ポインターに初期化され、戻される値は、セキュリティー交換中のセキュリティー出口への呼び出しから呼び出しまでの間、維持されます。 MQXR2_USE_EXIT_BUFFER フラグが MQCXP 構造体内の ExitResponse2 に設定されている場合、セキュリティー出口ではこのパラメーターを、送信されるメッセージ・データを指すように設定する必要があります。

CLNTCONN/SVRCONN チャネルのペアおよび他のチャネルのペアで定義されたセキュリティー出口における動作の相違点
セキュリティー出口は、すべてのタイプのチャネルで定義できます。 ただし、CLNTCONN/SVRCONN チャネルのペアで定義されたセキュリティー出口の動作は、他のチャネルのペアで定義されたセキュリティー出口と少し異なります。

CLNTCONN チャネルのセキュリティー出口は、パートナー SVRCONN 出口による処理のために、または SVRCONN セキュリティー出口が定義されておらず SVRCONN の MCAUSER フィールドが設定されていない場合の OAM 許可のために、チャネル定義でリモート・ユーザー ID を設定できます。

CLNTCONN セキュリティー出口が定義されていない場合、チャネル定義のリモート・ユーザー ID は、クライアント MCA によってクライアント環境のユーザー ID に設定されます (ブランクの場合もあります)。

CLNTCONN と SVRCONN のチャネルのペアで定義されたセキュリティー出口間でのセキュリティー交換は、SVRCONN セキュリティー出口が、MQXCC_OK の ExitResponse を戻す場合に正常に完了します。 他のチャネルのペア間でのセキュリティー交換は、交換を開始したセキュリティー出口が、MQXCC_OK の ExitResponse を戻す場合に正常に完了します。

ただし、MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse コードを使用して、セキュリティー交換の継続を強制することができます。MQXCC_SEND_AND_REQUEST_SEC_MSG の ExitResponse が CLNTCONN または SVRCONN セキュリティー出口によって戻された場合、パートナー出口はセキュリティー・メッセージ (MQXCC_OK やヌル応答ではない) を送信することによって応答する必要があり、そうしない場合はチャネルは終了します。 他のタイプのチャネルで定義されたセキュリティー出口の場合、MQXCC_SEND_AND_REQUEST_SEC_MSG への応答としてパートナー・セキュリティー出口から

MQXCC_OK の ExitResponse が戻されると、ヌル応答が戻されたようにセキュリティー交換は継続し、チャネルは終了しません。

SSPI セキュリティー出口
IBM MQ for Windows は、セキュリティー・サービス・プログラミング・インターフェース (SSPI) を使用して IBM MQ チャネルの認証を行う、セキュリティー出口を提供します。 SSPI には、Windows の統合セキュリティー機能が備えられています。

このセキュリティー出口は、IBM MQ クライアントおよび IBM MQ サーバーの両方で使用されます。

セキュリティー・パッケージは、security.dll または secur32.dll のいずれかからロードします。 これらの DLL は、ご使用のオペレーティング・システム (OS) で提供されています。

片方向認証は Windows で提供されているもので、NTLM 認証サービスを使用します。 双方向認証は Windows 2000 で提供され、Kerberos 認証サービスを使用します。

セキュリティー出口プログラムは、ソースおよびオブジェクト形式で提供されます。 オブジェクト・コードをそのまま使用することもできますし、あるいはソース・コードを開始点として使用して独自のユーザー出口プログラムを作成することもできます。 SSPI セキュリティー出口のオブジェクトまたはソース・コードの使用について詳しくは、1140 ページの『Windows での SSPI セキュリティー出口の使用』を参照してください。

チャネル送信および受信出口プログラム
送信出口と受信出口を使用して、データの圧縮や圧縮解除などの作業を実行することができます。 連続して実行する送信および受信出口プログラムのリストを指定できます。

チャネル送信出口プログラムと受信出口プログラムは、MCA の処理サイクルの次の場所で呼び出されます。

- 送信出口プログラムと受信出口プログラムは、MCA の開始時に初期設定のために呼び出され、MCA の終了時には終了のために呼び出されます。
- 送信出口プログラムは、伝送がリンクを介して送信される直前に、チャネルの一方または他方の側 (1 つのメッセージ転送の伝送の送信元となる側) で呼び出されます。 メッセージ・チャネルでは一方向にのみメッセージが送信されるにもかかわらず、両方向で出口を使用できる理由についての説明は、注 4 を参照してください。
- 受信出口プログラムは、伝送がリンクから取り出された直後に、チャネルの一方または他方の側 (1 つのメッセージ転送の伝送が受信された側) で呼び出されます。 メッセージ・チャネルでは一方向にのみメッセージが送信されるにもかかわらず、両方向で出口を使用できる理由についての説明は、注 4 を参照してください。

1 つのメッセージ転送に対して多くの伝送がある場合があり、メッセージが受信側のメッセージ出口に到達する前に送信出口プログラムと受信出口プログラムの多くの反復がある可能性があります。

チャネル送信出口および受信出口プログラムには、通信リンクから送信されたり受信されたりした伝送データが入っているエージェント・バッファーが渡されます。 送信出口プログラムの場合、バッファーの最初の 8 バイトは MCA による使用のために予約されており、変更できません。 プログラムが別のバッファーを返す場合、新規バッファーにはこれらの最初の 8 バイトが存在する必要があります。 これらの出口プログラムに提示されるデータの形式は定義されていません。

適切な応答コードが送信および受信の出口プログラムによって戻される必要があります。 それ以外の応答は、MCA 異常終了 (アベンド) の原因になります。

注 : 送信出口または受信出口から同期点の内部で MQGET、MQPUT、または MQPUT1 呼び出しを発行しないでください。

注 :

1. 通常、送信出口と受信出口は、ペアで使用します。 例えば、送信出口がデータを圧縮して受信出口がデータを圧縮解除したり、送信出口がデータを暗号化して受信出口がデータを暗号化解除したりすることがあります。 該当するチャネルを定義するときには必ず、チャネルの両側で互換性のある出口プログラムを指定するようにしてください。
2. チャネルの圧縮がオンの場合、出口には圧縮データが渡されます。

3. チャネル送信出口および受信出口は、状況メッセージなどのアプリケーション・データ以外のメッセージ・セグメントに対して呼び出されることがあります。 開始ダイアログ中またはセキュリティー検査フェーズで呼び出されることはありません。
4. メッセージ・チャネルは一方向にのみメッセージを送信しますが、チャネル制御データ (ハートビートやバッチ処理の終了など) は両方向に流れるため、これらの出口も両方向で利用可能です。 ただし、初期チャネル開始データ・フローのなかには、任意の出口による処理対象から除外されるものもあります。
5. 送信出口と受信出口が、順序を無視して呼び出される場合があります。例えば、一連の出口プログラムの実行中や、出口プログラムの他にセキュリティー出口も実行している場合です。 したがって、データを処理するために受信出口が最初に呼び出されたとき、対応する送信出口を通過して渡されていないデータを受け取る可能性があります。 受信出口が、操作が必要であることを最初にチェックせずに圧縮解除などの操作をただ実行した場合、予期しない結果になることがあります。

   送信出口および受信出口をコーディングする場合は、受信中のデータが対応する送信出口によって処理済みであるということを受信出口が検査できるような方法で行う必要があります。 推奨される実行方法は、出口プログラムを以下のようにコーディングすることです。

   - 送信出口はデータの 9 番目のバイトの値を 0 に設定し、すべてのデータを 1 バイトだけシフトさせた後で操作を実行します (最初の 8 バイトは MCA で使用するために予約されています)。
   - 受信出口は、バイト 9 の値が 0 のデータを受信すると、そのデータを送信出口から送られてきたデータであると認識します。 0 を削除し、補数演算を実行し、結果データを 1 バイトだけシフトして戻します。
   - 受信出口は、バイト 9 が 0 以外のデータを受信すると、送信出口は稼働していないと見なし、データを変更せずに呼び出し側に送り返す。

   セキュリティー出口の使用中、セキュリティー出口によってチャネルが終了した場合、対応する受信出口が存在しないのに送信出口が呼び出される可能性があります。 この問題を回避する 1 つの方法は、出口がチャネルを終了するときに、MQCD.SecurityUserData または MQCD.SendUserData などのフラグを設定するようにセキュリティー出口をコーディングすることです。 次に、送信出口はこのフィールドを検査し、フラグが設定されていない場合のみデータを処理する必要があります。 このチェックによって送信出口によるデータの不要な変更が回避され、セキュリティー出口が変更済みデータを受信した場合に発生する可能性がある変換エラーを回避することができます。

チャネル送信出口プログラム - スペースの予約
送信および受信出口を使用して、データを変換してから伝送することができます。 チャネル送信出口プログラムでは、変換バッファー内でスペースを予約することにより、変換に関する独自のデータを追加できます。

このデータは受信出口プログラムによって処理されてから、バッファーから除去されます。 例えば、データを暗号化し、暗号化解除のセキュリティー・キーを追加することもできます。

スペースの予約および使用の方法

初期設定で送信出口プログラムが呼び出されたときに、MQXCP の *ExitSpace* フィールドを、予約するバイト数に設定します。 詳しくは、MQCXP を参照してください。 *ExitSpace* を設定できるのは、初期設定の間、つまり *ExitReason* の値が MQXR_INIT であるときだけです。 *ExitReason* が MQXR_XMIT に設定され、伝送の直前に送信出口が呼び出されると、伝送バッファー内で *ExitSpace* バイトが予約されます。 *ExitSpace* は、z/OS ではサポートされていません。

送信出口は、予約されたすべてのスペースを使用する必要はありません。 *ExitSpace* のバイト数まで使用でき、伝送バッファーがいっぱいでなければ、出口は予約されている量を超えて使用できます。
ExitSpace の値の設定時には、伝送バッファー内にメッセージ・データ用として最低でも 1 KB は残しておかなければなりません。 予約されたスペースが大量のデータ用に使用される場合、チャネル・パフォーマンスに影響することがあります。

伝送バッファーの長さは通常 32KB です。 ただし、チャネルが TLS を使用する場合には、RFC 6101 および関連する TLS 標準のファミリーで規定されている最大レコード長に収まるようにするため、伝送バッフ

ァー・サイズが 15,352 バイトに減少します。 さらに 1024 バイトが IBM MQ による使用のために予約されるので、送信出口で使用可能な伝送バッファーの最大スペースは 14,328 バイトとなります。

チャネルの受信側で行われる事柄

チャネル受信出口プログラムは、対応する送信出口と互換性と持つようにセットアップしなければなりません。 受信出口は予約されたスペース内のバイト数を認識していなければならず、そのスペース内のデータを除去しなければなりません。

複数の送信出口

連続して実行する送信および受信出口プログラムのリストを指定できます。 IBM MQ は、すべての送信出口によって予約されたスペースの合計を保持します。 このスペースの合計では、伝送バッファー内にメッセージ・データ用として最低でも 1 KB は残しておかなければなりません。

以下の例では、連続して呼び出される 3 つの送信出口に、スペースが割り振られる方法が示されています。

1. 初期設定時の呼び出しで、以下のことが行われます。
   - 送信出口 A で 1 KB が予約されます。
   - 送信出口 B で 2 KB が予約されます。
   - 送信出口 C で 3 KB が予約されます。
2. 最大伝送サイズは 32 KB で、ユーザー・データの長さは 5 KB です。
3. 出口 A は 5 KB のデータで呼び出されます。出口 B および C に 5 KB が予約されているため、27 KB までが使用可能です。 出口 A は、それが予約した量である 1 KB を追加します。
4. 出口 B は 6 KB のデータで呼び出されます。出口 C に 3 KB が予約されているため、29 KB までが使用可能です。 出口 B は、それが予約した 2 KB よりも少ない 1 KB を追加します。
5. 出口 C が 7 KB のデータで呼び出されます。最大で 32 KB まで使用可能です。 出口 C は、それが予約した 3 KB より大きい 10 KB を追加します。 データの合計 17 KB は最大値の 32 KB より小さいため、この量は有効です。

TLS を使用するチャネルに対する最大送信バッファー・サイズは 32Kb ではなく 15,352 バイトです。 その理由は、基礎となるセキュア・ソケット伝送セグメントが 16Kb に限定されており、そのスペースの一部が TLS レコード・オーバーヘッドに必要とされるためです。 さらに 1024 バイトが IBM MQ による使用のために予約されるので、送信出口で使用可能な伝送バッファーの最大スペースは 14,328 バイトとなります。

チャネル・メッセージ出口プログラム
チャネル・メッセージ出口を使用して、リンクの暗号化、着信ユーザー ID の妥当性検査または置換、メッセージ・データの変換、ジャーナル処理、および参照メッセージ処理などのタスクを実行できます。 連続して実行するメッセージ出口プログラムのリストを指定できます。

チャネル・メッセージ出口プログラムは、MCA の処理サイクルの次の場所で呼び出されます。

- MCA 開始および終了時
- 送信側 MCA が MQGET 呼び出しを発行した直後
- 受信側 MCA が MQPUT 呼び出しを発行する前

メッセージ出口には、伝送キュー・ヘッダー MQXQH およびキューから検索されたアプリケーション・メッセージ・テキストが入っているエージェント・バッファーが渡されます MQXQH の形式は、MQXQH - 伝送キュー・ヘッダーにあります。

参照メッセージ (送信される他のオブジェクトを指すヘッダーのみを含んだメッセージ) を使用する場合、メッセージ出口はヘッダー MQRMH を認識します。 このヘッダーは、オブジェクトを識別し、該当する方法でオブジェクトを検索し、それをヘッダーに追加します。そして、そのヘッダーを受信側 MCA へ伝送するために MCA に渡します。 受信側 MCA にある別のメッセージ出口は、このメッセージが参照メッセージであることを認識し、オブジェクトを抽出し、ヘッダーを宛先キューに渡します。 参照メッセージと、参照メッセージを処理するメッセージ出口のサンプルについては、795 ページの『参照メッセージ』および 1110 ページの『参照メッセージ・サンプルの実行』を参照してください。

メッセージ出口は、次のような応答を戻します。

- メッセージを送信する (GET 出口)。 メッセージは出口によって変更された可能性があります (この場合は MQXCC_OK を戻します)。
- メッセージをキューに書き込む (PUT 出口)。 メッセージは出口によって変更された可能性があります (この場合は MQXCC_OK を戻します)。
- メッセージを処理しない。 メッセージは、MCA によって送達不能キュー (未配布メッセージ・キュー) に入ります。
- チャネルをクローズする。
- 不良戻りコードが戻される。この場合には MCA が異常終了します。

注 :

1. メッセージが部分に分割された場合であっても、メッセージ出口は、転送される完全な各メッセージにつき 1 回呼び出されます。
2. Linux AIX AIX または Linux でメッセージ出口を指定した場合、ユーザー ID を自動的に小文字に変換する機能 (ここを参照) は働きません。
3. 出口は MCA 自体と同じスレッドで実行されます。 また、同じ接続ハンドルを使用するため、MCA と同じ作業単位 (UOW) 内で実行されます。 したがって、同期点で行われた呼び出しは、すべてバッチの終わりにチャネルによってコミットまたはバックアウトされます。 例えば、あるチャネル・メッセージ出口プログラムは別のチャネル・メッセージ出口プログラムに通知メッセージを送信でき、これらのメッセージは、元のメッセージを含んでいるバッチがコミットされたときに限りキューにコミットされます。

   したがって、チャネル・メッセージ出口プログラムから同期点 MQI 呼び出しを発行できます。

メッセージ出口の外側でのメッセージ変換
メッセージ出口を呼び出す前に、受信側 MCA はメッセージの変換を実行します。 このトピックでは、変換を実行するために使用されるアルゴリズムについて説明します。

処理されるヘッダー

変換ルーチンは、メッセージ出口が呼び出される前に受信側の MCA 内で実行します。 変換ルーチンはメッセージの冒頭にある MQXQH ヘッダーから始まります。 その後、変換ルーチンは、MQXQH に続くチェーン・ヘッダーを処理し、必要に応じて変換を実行します。 チェーン・ヘッダーは、受信側のメッセージ出口に渡された MQCXP データの HeaderLength パラメーターに含まれるオフセットを超えて拡張できます。 以下のヘッダーは同じ場所で変換されます。

- MQXQH (形式名「MQXMIT」)
- MQMD (このヘッダーは MQXQH の一部で形式名なし)
- MQMDE (形式名「MQHMDE」)
- MQDH (形式名「MQHDIST」)
- MQWIH (形式名「MQHWIH」)

以下のヘッダーは変換されませんが、MCA がチェーン・ヘッダーの処理を続行するときにステップオーバーされます。

- MQDLH (形式名「MQDEAD」)
- 「MQH」の 3 文字で開始される形式名を持つ任意のヘッダー (例えば「MQHRF」) で、それ以外の方法で言及されないもの

ヘッダーの処理方法

各 IBM MQ ヘッダーの Format パラメーターは MCA によって読み取られます。 Format パラメーターはヘッダー内の 8 バイトで、名前を含む 8 つの 1 バイト文字です。

その後、MCA は各ヘッダーに続くデータを名前付きタイプとして解釈します。 Format が、IBM MQ データ変換に適格なヘッダー・タイプの名前であれば、データは変換されます。 これが非 MQ データを表す別の名前である場合 (例えば MQFMT_NONE または MQFMT_STRING)、MCA はヘッダーの処理を停止します。

MQCXP HeaderLength について

メッセージ出口に提供される MQCXP データ内の HeaderLength パラメーターは、メッセージの最初にある MQXQH (MQMD を含む)、MQMDE、および MQDH ヘッダーの合計長です。 これらのヘッダーは、「Format」の名前と長さを使用してチェーニングされます。

MQWIH

チェーニングされたヘッダーは、HeaderLength を超えてユーザー・データ域まで拡張できます。 MQWIH ヘッダー (ある場合) は、HeaderLength を超えて表示されるそのようなヘッダーの 1 つです。

チェーン・ヘッダー内に MQWIH ヘッダーがある場合、受信側のメッセージ出口が呼び出される前に、このヘッダーは同じ場所で変換されます。

チャネル・メッセージ再試行出口プログラム

チャネル・メッセージ再試行出口は、ターゲット・キューのオープンに失敗したときに呼び出されます。 この出口を使用して、再試行を行う環境、再試行の回数、再試行の頻度を決めることができます。

この出口は、MCA 開始および終了時にチャネルの受信側でも呼び出されます。

チャネル・メッセージ再試行出口には、伝送キュー・ヘッダー MQXQH と、キューから検索されたアプリケーション・メッセージ・テキストが入っているエージェント・バッファーが渡されます。 MQXQH の形式は、MQXQH の概要に示されています。

この出口はすべての理由コードについて呼び出され、MCA が再試行する必要がある理由コードと、再試行の回数および間隔を判別します (チャネルが定義されたときに設定されたメッセージ再試行カウントの値が MQCD の出口に渡されますが、出口はこの値を無視することができます)。

MQCXP の MsgRetryCount フィールドの値は、出口が呼び出されるたびに MCA によって増分され、出口は MQCXP の MsgRetryInterval フィールドに入っている待ち時間を示した MQXCC_OK、あるいは MQXCC_SUPPRESS_FUNCTION のいずれかを戻します。 再試行は、出口が MQCXP の ExitResponse フィールドに MQXCC_SUPPRESS_FUNCTION を戻すまで無限に続きます。 これらの完了コードに対して MCA が行う処置については、MQCXP を参照してください。

すべての再試行が失敗した場合は、メッセージは送達不能キューに書き込まれます。 使用可能な送達不能キューが 1 つもない場合、チャネルは停止します。

チャネル用にメッセージ再試行出口を定義しないときに、MQRC_Q_FULL などの一時的なものと思われる障害が起こると、MCA は、チャネルが定義されたときに設定されたメッセージ再試行カウントとメッセージ再試行間隔を使用します。 障害が永続的なもので、障害を処理する出口プログラムを定義していない場合、メッセージは送達不能キューに書き込まれます。

チャネル自動定義出口プログラム

受信側チャネルまたはサーバー接続チャネルを開始する要求を受信してもそのチャネル定義が存在しない場合、チャネル自動定義出口を使用することができます (IBM MQ for z/OS の場合を除く)。 また、いずれのプラットフォームにおいても、この出口を呼び出すことにより、クラスター送信側チャネルとクラスター受信側チャネルの両方で、チャネルのインスタンスの定義変更ができるようになります。

z/OS 以外のすべてのプラットフォームでは、受信側チャネルまたはサーバー接続チャネルを開始する要求が受信されてチャネル定義が存在しない場合、チャネル自動定義出口を呼び出すことができます。 これを使用して、自動的に定義される受信側の SYSTEM.AUTO.RECEIVER、または、サーバー接続チャネルの SYSTEM.AUTO.SVRCON 用に提供されるデフォルト定義を変更できます。 チャネル定義を自動的に作成する方法については、チャネルの準備を参照してください。

チャネル自動定義出口は、クラスター送信側チャネル開始の要求が受信された場合にも呼び出せます。 この出口を呼び出すことによって、クラスター送信側チャネルとクラスター受信側チャネルの両方で、チャネルのインスタンスの定義変更ができるようになります。 この場合、出口は IBM MQ for z/OS にも適用されます。 チャネル自動定義出口は通常、メッセージ出口名 (MSGEXIT、RCVEXIT、SCYEXIT、および

SENDEXIT) の変更に使用されます。出口名のフォーマットはプラットフォームによって異なるためです。チャネル自動定義出口名が指定されていない場合の z/OS でのデフォルト動作は、フォーム *[path]/librarynameの(function)* という分散出口名を検査し、存在する場合、関数 (またはライブラリー名) から最高 8 文字を取得します。 z/OS では、チャネル自動定義出口プログラムは、MsgExit、MsgUserData、SendExit、SendUserData、ReceiveExit、および ReceiveUserData フィールド自体ではなく、MsgExitPtr、MsgUserDataPtr、SendExitPtr、SendUserDataPtr、ReceiveExitPtr、および ReceiveUserDataPtr で指定されるフィールドを変更する必要があります。

詳細については、自動定義チャネルの処理を参照してください。

他のチャネル出口の場合は、次のようなパラメーター・リストを指定します。

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

ChannelExitParms については、MQCXP で説明しています。 ChannelDefinition については、MQCD で説明しています。

MQCD には、デフォルトのチャネル定義で使用されている値が (出口によって変更されていない場合は) 入ります。 出口は、フィールドのサブセットだけを変更することができます。詳細については、MQ_CHANNEL_AUTO_DEF_EXIT を参照してください。 ただし、他のフィールドを変更しようとしてもエラーは起こりません。

チャネル自動定義出口は、MQXCC_OK または MQXCC_SUPPRESS_FUNCTION の応答を戻します。 どちらの応答も戻らない場合、MCA は MQXCC_SUPPRESS_FUNCTION が戻ったと想定して処理を続行します。これは自動定義が異常終了したため、新しいチャネル定義が作成されずチャネルを開始できないということです。

ALW *AIX, Linux, and Windows システムでのチャネル出口プログラムのコンパイル*

次の例を使用して、AIX, Linux, and Windows システムでのチャネル出口プログラムをコンパイルします。

Windows

Windows

チャネル出口プログラムのコンパイラーおよびリンカー・コマンド - Windows

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

AIX and Linux システム

Linux AIX

これらの例では、exit はライブラリー名で、ChannelExit は関数名です。 AIX では、エクスポート・ファイルは exit.exp と呼ばれます。 これらの名前は、チャネル定義によって、MQCD チャネル定義で説明されている形式で出口プログラムを参照するために使用されます。 DEFINE CHANNEL コマンドの MSGEXIT パラメーターも参照してください。

AIX チャネル出口のコンパイラーおよびリンカー・コマンドのサンプル - AIX

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Linux においてキュー・マネージャーが 32 ビットの場合のチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Linux においてキュー・マネージャーが 64 ビットの場合のチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

クライアントでは、32 ビットまたは 64 ビット出口を使用できます。 この出口は、mqic_r にリンクされている必要があります。

AIX AIX では、IBM MQ によって呼び出されるすべての関数をエクスポートする必要があります。この Make ファイルのサンプル・エクスポート・ファイル:

```
#
!channelExit
MQStart
```

チャネル出口の構成

チャネル出口を呼び出すには、チャネル定義でそのチャネルに名前を指定する必要があります。

チャネル出口には、チャネル定義で名前を付ける必要があります。 この命名は、チャネルを最初に定義するときに行うことができます。または、例えば MQSC コマンドの ALTER CHANNEL を使用して、後で情報を追加することができます。 MQCD チャネル・データ構造体でチャネル出口名を指定することもできます。出口名のフォーマットは、IBM MQ プラットフォームによって異なります。詳しくは、MQCD または MQSC コマンドを参照してください。

チャネル定義にユーザー出口プログラム名が含まれない場合、そのユーザー出口は呼び出されません。

チャネル自動定義出口は、キュー・マネージャーの特性であり、個別チャネルの特性ではありません。 この出口が呼び出されるようにするには、キュー・マネージャーの定義で名前を指定する必要があります。キュー・マネージャー定義を変更するには、MQSC コマンドの ALTER QMGR を使用します。

データ変換出口の作成

この一連のトピックには、データ変換出口を作成する方法に関する情報が含まれています。

注 : MQSeries for VSE/ESA ではサポートされていません。

MQPUT を実行する際、アプリケーションはメッセージのメッセージ記述子 (MQMD) を作成します。MQMD は、作成されるプラットフォームに関係なく、その内容が IBM MQ に理解できるものでなければならないので、システムによって自動的に変換されます。

ただし、アプリケーション・データは自動的に変換されません。 CodedCharSetId フィールドおよび Encoding フィールドが異なるプラットフォーム間 (例えば ASCII と EBCDIC の間) で文字データがやり取りされている場合、メッセージの変換はアプリケーション側で調整する必要があります。 アプリケーションのデータ変換は、キュー・マネージャー自体またはデータ変換出口 と呼ばれるユーザー出口プログラムで実行できます。 アプリケーション・データが組み込み形式の 1 つ (MQFMT_STRING など) である場合、キュー・マネージャー自体が、組み込み変換ルーチンの 1 つを使って、データ変換を実行できます。 このトピックには、アプリケーション・データが組み込まれている形式とは異なる場合のために IBM MQ が提供する、データ変換機能に関する情報が含まれています。

MQGET 呼び出しの処理中に、制御をデータ変換出口に渡すことができます。 これは、最終宛先に到着する前に異なるプラットフォーム間でデータが変換されないようにします。 ただし、最終宛先が MQGET でのデータ変換をサポートしていないプラットフォームであれば、データをその最終宛先に送る送信側のチャネルで CONVERT(YES) を指定する必要があります。 これによって、間違いなく IBM MQ で伝送中にデータが変換されます。 このとき、送信側のチャネルを定義しているシステムには、必ずデータ変換出口が設定されていなければなりません。

MQGET 呼び出しは、アプリケーションによって直接発行されます。 MQMD 中の CodedCharSetId および Encoding フィールドを、必要な文字セットおよびエンコードに設定してください。 アプリケーションがキュー・マネージャーと同じ文字セットおよびエンコードを使用する場合は、CodedCharSetId を

MQCCSI_Q_MGR に、Encoding を MQENC_NATIVE に設定します。 MQGET 呼び出しの完了後、これらのフィールドには戻されるメッセージ・データに該当する値が指定されます。 変換が失敗した場合、これらのフィールドの値が必要な値とは異なることがあります。 その場合、アプリケーションは、これらのフィールドを、各 MQGET 呼び出しの前に必要とされた値にリセットしなければなりません。

MQGET 呼び出し用にデータ変換出口が呼び出されるために必要な条件は、MQGET で定義されています。

データ変換出口に渡される MQ_DATA_CONV_EXIT 呼び出しと MQDXP 構造体のパラメーターと詳細な使用上の注意については、データ変換を参照してください。

アプリケーション・データをエンコード方式の異なるマシン間および異なる CCSID 間で変換するプログラムは、IBM MQ データ変換インターフェース (DCI) に必ず準拠していなければなりません。

マルチキャスト・クライアントの場合、API 出口とデータ変換出口はクライアント・サイドで実行可能であることが必要になりました。これは、一部のメッセージがキュー・マネージャーを経由しない可能性があるためです。 以下のライブラリーは、サーバー・パッケージだけでなくクライアント・パッケージにも含まれています。

表 *143.* クライアント・パッケージとサーバー・パッケージの両方に含まれているライブラリー

オペレーティング・システム	**ライブラリー**
AIX	32 ビットおよび 64 ビット: libmqm.a および libmqm_r.a
IBM i	LIBMQM および LIBMQM_R
Linux	32 ビットおよび 64 ビット: libmqm.so および libmqm_r.so
Windows	32 ビットおよび 64 ビット: mqm.dll および mqm.pdb

データ変換出口の起動

データ変換出口とは、MQGET 呼び出しの処理中に制御を受け取るユーザー作成出口です。

この出口は、次の記述が当てはまる場合に呼び出さされます。

- MQGMO_CONVERT オプションが、MQGET 呼び出しで指定されている。
- メッセージ・データの一部または全部が、必要な文字セットまたはエンコードになっていない。
- メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、MQFMT_NONE でない。
- MQGET 呼び出しで指定された *BufferLength* が、ゼロでない。
- メッセージ・データの長さが、ゼロでない。
- メッセージに、ユーザー定義フォーマットのデータが入っている。 メッセージ全体がユーザー定義フォーマットになっていることもあれば、その前に 1 つまたは複数の組み込みフォーマットが来ることもあります。 例えば、ユーザー定義フォーマットの前に、MQFMT_DEAD_LETTER_HEADER フォーマットが来ることがあります。 出口は、ユーザー定義フォーマットだけを変換するために呼び出されます。一方、キュー・マネージャーは、ユーザー定義フォーマットの前に来るすべての組み込みフォーマットを変換します。

  また、ユーザー作成出口が組み込みフォーマットを変換するために呼び出されることもありますが、これは組み込み変換ルーチンが組み込みフォーマットを正常に変換できなかった場合に限られます。

その他にも条件があり、MQ_DATA_CONV_EXIT の MQ_DATA_CONV_EXIT 呼び出しの使用上の注意で詳しく説明されています。

MQGET 呼び出しの詳細については、MQGET を参照してください。 MQXCNVC 以外のデータ変換出口は、MQI 呼び出しを使用できません。

出口の新規コピーがロードされるのは、アプリケーションがキュー・マネージャーに接続された後、*Format* を使用する最初のメッセージを検索しようとするときです。 これ以外のときにも、前にロードされたコピーをキュー・マネージャーが既に廃棄した場合には、新規コピーがロードされることがあります。

データ変換出口は、MQGET 呼び出しを発行するプログラムの環境と同様の環境で稼働します。 ユーザー・アプリケーションと同様に、プログラムはメッセージ変換をサポートしない宛先キュー・マネージャーにメッセージを送信する MCA (メッセージ・チャネル・エージェント) であっても構いません。 この環境には、該当する場合、アドレス・スペースとユーザー・プロファイルが組み込まれます。 出口は、キュー・マネージャーの環境で稼働しないので、そのキュー・マネージャーの保全性を損なうことはありません。

z/OS 上でのデータ変換

z/OS

z/OS では、以下の点に注意してください。

- 出口プログラムを記述できるのはアセンブリー言語だけです。
- 出口プログラムは、ストレージのあらゆる場所で実行できる、再入力可能なプログラムでなければなりません。
- 出口プログラムは、出口時に環境を入り口時の環境に復元し、占有していたストレージをすべて解放しなければなりません。
- 出口プログラムは待機 (WAIT) することも、ESTAE や SPIE を発行することもできません。
- 通常、出口プログラムは、次の状態で z/OS LINK によって呼び出されたかのように呼び出されます。
  - 許可がない問題プログラム状態
  - 1 次アドレス・スペース制御モード
  - 非仮想記憶間モード
  - 非アクセス・レジスター・モード
  - 31 ビット・アドレッシング・モード
  - TCB-PRB モード
- CICS アプリケーションで使用される場合には、出口は、EXEC CICS LINK によって呼び出され、CICS プログラミング規則に適合していなければなりません。 これらのパラメーターは、CICS 連絡域 (COMMAREA) でポインター (アドレス) によって渡されます。

  推奨されてはいないものの、以下の点に注意して、ユーザー出口プログラムで CICS API 呼び出しを使用することもできます。

  - その結果によっては、MCA で宣言された作業単位に影響することもあるので、同期点を発行しないでください。
  - IBM MQ for z/OS 以外のリソース・マネージャーによって制御されているリソース (CICS Transaction Server によって制御されているリソースを含む) は更新しないでください。

  CONVERT=YES が設定されたチャネルの場合、出口は CSQXLIB DD ステートメントで参照されたデータ・セットからロードされます。 IBM MQ CICS ブリッジに関する MQ 提供の出口である CSQCBDCI と CSQCBDCO は、SCSQAUTH 内にあります。

IBM i 用のデータ変換出口プログラムの作成

IBM i 用の MQ データ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. メッセージ・フォーマットに名前を付けます。 その名前は、必ず MQMD の *Format* フィールド内に入る長さにしてください。 *Format* 名の先頭には、ブランクは指定できません。また、名前の後ろのブランクは無視されます。 *Format* の長さは 8 文字のみであるため、このオブジェクトの名前は、ブランク以外の、8 文字までの文字で構成しなければなりません。 メッセージを 1 つ送信するたびに、必ずこの名前を指定してください (この例では Format という名前を使っています)。
2. メッセージを表す構造体を作成します。 この例は、有効な構文に記載されています。
3. CVTMQMDTA コマンドを介してこの構造体を実行し、データ変換出口用のコードのフラグメントを作成します。

CVTMQMDTA コマンドで生成された関数は、ファイル QMQM/H(AMQSVMHA) に入っているマクロを使用します。 これらのマクロは、すべての構造体がパックされていることを前提として作成されています。そうでない場合は、これらを修正する必要があります。

4. 提供されたスケルトン・ソース・ファイル QMQMSAMP/QCSRC(AMQSVFC4) のコピーを取り、そのファイル名を変更します (ここでは、EXIT_MOD という名前を使っています。)
5. ソース・ファイルから以下のコメント・ボックスを検索し、その中の指示どおりにコードを挿入します。

   a. ソース・ファイルの最後の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   ここに、ステップ 990 ページの『3』 で生成したコードのフラグメントを挿入します。

   b. ソース・ファイルのほぼ中央に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   このあとには、コメント化された ConverttagSTRUCT 関数の呼び出しが続きます。

   この関数の名前をステップ 991 ページの『5.a』 で追加した関数の名前に変更します。 コメント文字を削除して、関数を活動状態にします。 関数がいくつかある場合は、各関数ごとに呼び出しを作成します。

   c. ソース・ファイルの最初の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   ここに、ステップ 991 ページの『5.a』 で追加した関数のプロトタイプ・ステートメントを挿入します。

   文字データを含むメッセージの場合には、生成されたコードにより MQXCNVC が呼び出されます。サービス・プログラム QMQM/LIBMQM をバインドすれば、これを解決できます。

6. 次のようにして、ソース・モジュール EXIT_MOD をコンパイルします。

   ```
   CRTCMOD MODULE(library/EXIT_MOD) +
   SRCFILE(QCSRC) +
   TERASPACE(*YES *TSIFC)
   ```

7. プログラムを作成してリンクします。

   スレッド化されていないアプリケーションの場合は、以下のようにします。

   ```
   CRTPGM PGM(library/Format)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
   ```

   スレッド環境では、基本環境用のデータ変換出口に加えて、データ変換出口をもう 1 つ作成します。 このロード可能オブジェクトの後には _R を付ける必要があります。 MQXCNVC への呼び出しを解決するには、LIBMQM_R ライブラリーを使用します。 スレッド環境では、どちらのロード可能オブジェクトも必要です。

   ```
   CRTPGM PGM(library/Format_R)  +
      MODULE(library/EXIT_MOD)  +
      BNDSRVPGM(QMQM/LIBMQM_R)  +
      ACTGRP(QMQM)  +
      USRPRF(*USER)
   ```

8. IBM MQ ジョブ用のライブラリー・リストに出力します。 実動時には、データ変換出口プログラムを QSYS 内に格納しておくことをお勧めします。

注 :

1. CVTMQMDTA がパックされた構造体を使用している場合には、すべての IBM MQ アプリケーションは、必ず _Packed 修飾子を使用しなければなりません。
2. データ変換出口プログラムは、必ず再入可能でなければなりません。
3. MQXCNVC は、データ変換出口から発行できるただ 1 つの MQI 呼び出しです。
4. 出口がユーザーの権限で稼働できるように、ユーザー・プロファイル・コンパイラー・オプションを *USER に設定して出口プログラムをコンパイルします。
5. IBM MQ for IBM i のすべてのユーザー出口では、テラスペース・メモリーが使用可能でなければなりません。CRTCMOD および CRTBNDC コマンドで TERASPACE(*YES *TSIFC) を指定する必要があります。

IBM MQ for z/OS 用のデータ変換出口プログラムの作成

IBM MQ for z/OS 用のデータ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. 提供されたソース・スケルトン CSQ4BAX9 (CICS 以外の環境の場合) または CSQ4CAX9 (CICS 環境の場合) を開始点として利用する。
2. CSQUCVX ユーティリティーを実行する。
3. 変換したいメッセージ内で構造体を生成できるように、CSQ4BAX9 または CSQ4CAX9 のプロローグにある指示に従って、CSQUCVX ユーティリティーで生成されたルーチンを組み込む。
4. このユーティリティーは、データ構造体がパックされておらず、データの暗黙の位置合わせが受け入れられ、さらにそれらの構造体がフルワード境界で開始されるものと仮定する。また、バイトは必要に応じて (有効な構文の例における ID と VERSION の間でのように) スキップされます。 構造体がパックされている場合は、生成された CMQXCALA マクロを省略してください。 したがって、すべてのフィールドを指定し、どのバイトもスキップしないように構造体を宣言することを考慮してください。有効な構文の例では、ID と VERSION の間にフィールド "MQBYTE DUMMY;" を追加してください。
5. 提供された出口は、変換されるメッセージ形式よりも入力バッファーの方が短い場合に、エラーを戻す。 出口は、可能な限り多くの完了フィールドを変換しますが、このエラーにより、未変換メッセージがアプリケーションに戻されます。 短い入力バッファーを部分的なフィールドも含めて可能な限り変換できるようにしたい場合には、CSQXCDFA マクロ上の TRUNC= 値を YES に変更します。 アプリケーションは、切り捨てを処理しなければなりません。
6. その他の必要な特殊処理コードを追加する。
7. プログラムの名前を自分のデータ形式に変更する。
8. バッチ・アプリケーション・プログラムなどのプログラムをコンパイルおよびリンク・エディットする (そのプログラムが CICS アプリケーションで使用するためのものでない限り)。 ユーティリティーで生成されたマクロは、ライブラリー **thlqual**.SCSQMACS に入っています。

   メッセージに文字データが入っている場合には、生成済みのコードにより、MQXCNVC が呼び出されます。 ユーザー出口でこの呼び出しを使用する場合は、これを出口スタブ・プログラム CSQASTUB でリンク・エディットします。 これは言語にも環境にも依存しないスタブです。 別の方法として、動的呼び出し名 CSQXCNVC を使用して、スタブを動的にロードできます。 詳しくは、1035 ページの『IBM MQ スタブの動的呼び出し』を参照してください。

   リンク・エディット済みのモジュールを、アプリケーション・ロード・ライブラリーに配置し、またチャネル・イニシエーターから開始されるタスク・プロシージャーの CSQXLIB DD ステートメントで参照されるデータ・セットに配置します。
9. 出口が CICS アプリケーションで使用するためのものであれば、必要な場合には CSQASTUB を含む、CICS アプリケーション・プログラムなどのプログラムをコンパイルおよびリンク・エディットする。 そのプログラムを、CICS アプリケーション・プログラム・ライブラリーに配置します。 通常の方法でそのプログラムを CICS に定義し、その定義に EXECKEY(CICS) を指定します。

注 : LE/370 ランタイム・ライブラリーは、CSQUCVX ユーティリティーを実行するために必要ですが (ステップ 992 ページの『2』 を参照)、データ変換出口自体をリンク・エディットまたは実行するためには必要ありません (ステップ 992 ページの『8』 および 992 ページの『9』 を参照)。

IBM MQ - IMS ブリッジ内でのデータ変換については、 74 ページの『IMS ブリッジ・アプリケーションの作成』 を参照してください。

Linux AIX *IBM MQ for AIX or Linux システム用のデータ変換出口の作成*

IBM MQ for AIX or Linux システム用のデータ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. メッセージ・フォーマットに名前を付けます。 その名前は、必ず MQMD の *Format* フィールド内に入る長さにし、英大文字で表記 (例えば MYFORMAT) してください。 *Format* 名の先頭には、ブランクは指定できません。 また、名前の後ろのブランクは無視されます。 *Format* の長さは 8 文字のみであるため、このオブジェクトの名前は、ブランク以外の、8 文字までの文字で構成しなければなりません。 メッセージを 1 つ送信するたびに、必ずこの名前を指定してください。

   スレッド化環境でデータ変換出口を使用する場合は、ロード可能オブジェクトの後に、それがスレッド化バージョンであることを示す _r を付ける必要があります。
2. メッセージを表す構造体を作成します。 この例は、有効な構文に記載されています。
3. `crtmqcvx` コマンドを使用してこの構造体を実行し、データ変換出口用のコードのフラグメントを作成します。

   `crtmqcvx` コマンドで生成される関数は、すべての構造体がパックされていることを前提とするマクロを使用します。そうなっていない場合は、修正する必要があります。
4. 提供されているスケルトン・ソース・ファイルをコピーし、ファイルの名前をステップ 993 ページの『1』 で設定したメッセージ・フォーマットの名前に変更します。 スケルトン・ソース・ファイル、およびそのコピーは読み取り専用です。

   スケルトン・ソース・ファイルは、amqsvfc0.c という名前のファイルです。
5. IBM MQ for AIX には、amqsvfc.exp という名前のスケルトン・エクスポート・ファイルも用意されています。 このファイルをコピーして、ファイル名を MYFORMAT.EXP に変更します。
6. このスケルトンでは、*MQ_INSTALLATION_PATH*/inc ディレクトリー内にサンプル・ヘッダー・ファイル amqsvmha.h が組み込まれています。ここで *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。 組み込みパスがこのディレクトリーを指して、このファイルを選択するようになっていることを確認してください。

   amqsvmha.h ファイルには、`crtmqcvx` コマンドで生成されたコードが使用するマクロが入っています。 変換される文字データに構造体が入っている場合には、これらのマクロにより、MQXCNVC が呼び出されます。
7. ソース・ファイルから以下のコメント・ボックスを検索し、その中の指示どおりにコードを挿入します。

   a. ソース・ファイルの最後の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   ここに、ステップ 993 ページの『3』 で生成したコードのフラグメントを挿入します。

   b. ソース・ファイルのほぼ中央に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   このあとには、コメント化された `ConverttagSTRUCT` 関数の呼び出しが続きます。

この関数の名前をステップ 993 ページの『7.a』 で追加した関数の名前に変更します。 コメント文字を削除して、関数を活動状態にします。 関数がいくつかある場合は、各関数ごとに呼び出しを作成します。

c. ソース・ファイルの最初の方に、以下で始まるコメント・ボックスがあります。

```
/* Insert the function prototypes for the functions produced by */
```

ここに、ステップ 993 ページの『3』 で追加した関数のプロトタイプ・ステートメントを挿入します。

8. MQStart を入り口点に使用して、出口を共用ライブラリーとしてコンパイルします。 その方法については、994 ページの『AIX and Linux システムでのデータ変換出口のコンパイル』 を参照してください。
9. 出口ディレクトリーに出力を入れます。 デフォルトの出口ディレクトリーは、32 ビット・システムの場合は /var/mqm/exits で、64 ビット・システムの場合は /var/mqm/exits64 です。 qm.ini ファイルまたは mqclient.ini ファイルでこれらのディレクトリーを変更できます。 このパスは、それぞれのキュー・マネージャーで設定できます。出口の検索は、このパスでのみ行われます。

注 :

1. crtmqcvx がパックされた構造体を使用している場合は、すべての IBM MQ アプリケーションをこの方法でコンパイルする必要があります。
2. データ変換出口プログラムは、必ず再入可能でなければなりません。
3. MQXCNVC は、データ変換出口から発行できるただ 1 つの MQI 呼び出しです。

Linux AIX *AIX and Linux システムでのデータ変換出口のコンパイル*

AIX and Linux システムでのデータ変換出口のコンパイル方法について例を示します。

どのプラットフォームでも、モジュールへの入り口点は MQStart です。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

AIX

AIX

次のいずれかのコマンドを発行して出口ソース・コードをコンパイルします。

32 ビット・アプリケーション

非スレッド化

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション

非スレッド化

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
```

```
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

次のいずれかのコマンドを発行して出口ソース・コードをコンパイルします。

31 ビット・アプリケーション

非スレッド化

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

32 ビット・アプリケーション

非スレッド化

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション

非スレッド化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

スレッド化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Windows *IBM MQ for Windows 用のデータ変換出口の作成*

IBM MQ for Windows 用のデータ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. メッセージ・フォーマットに名前を付けます。 その名前は、必ず MQMD の *Format* フィールド内に入る長さにしてください。 *Format* 名の先頭には、ブランクは指定できません。 また、名前の後ろのブラ

ンクは無視されます。 *Format* の長さは 8 文字のみであるため、このオブジェクトの名前は、ブランク以外の、8 文字までの文字で構成しなければなりません。

サンプル・ディレクトリー *MQ_INSTALLATION_PATH*\Tools\C\Samples には、 amqsvfcn.def という名前の .DEF ファイルも用意されています。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーです。 このファイルのコピーを取り、そのファイル名を、例えば MYFORMAT.DEF に変更します。 作成している DLL の名前と MYFORMAT.DEF に指定されている名前は、必ず同じにしてください。 MYFORMAT.DEF にある名前 FORMAT1 を、新しい形式名で上書きします。

メッセージを 1 つ送信するたびに、必ずこの名前を指定してください。

2. メッセージを表す構造体を作成します。 この例は、有効な構文に記載されています。
3. crtmqcvx コマンドを使用してこの構造体を実行し、データ変換出口用のコードのフラグメントを作成します。

   CRTMQCVX コマンドで生成された関数は、すべての構造体がパックされていることを前提として作成されたマクロを使用します。そうなっていない場合は、修正する必要があります。
4. 提供されているスケルトン・ソース・ファイル amqsvfc0.c をコピーし、ファイルの名前をステップ 995 ページの『1』 で設定したメッセージ・フォーマットの名前に変更します。

   amqsvfc0.c は *MQ_INSTALLATION_PATH*\Tools\C\Samples にあります。ここで、 *MQ_INSTALLATION_PATH* は IBM MQ がインストールされているディレクトリーです。 (デフォルトのインストール・ディレクトリーは C:¥Program Files¥IBM¥MQ です。)

   スケルトンには、 *MQ_INSTALLATION_PATH*\Tools\C\include ディレクトリーにサンプル・ヘッダー・ファイル amqsvmha.h が含まれています。 組み込みパスがこのディレクトリーを指して、このファイルを選択するようになっていることを確認してください。

   amqsvmha.h ファイルには、CRTMQCVX コマンドで生成されたコードが使用するマクロが入っています。 変換される文字データに構造体が入っている場合には、これらのマクロにより、MQXCNVC が呼び出されます。
5. ソース・ファイルから以下のコメント・ボックスを検索し、その中の指示どおりにコードを挿入します。

   a. ソース・ファイルの最後の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   ここに、ステップ 996 ページの『3』 で生成したコードのフラグメントを挿入します。

   b. ソース・ファイルのほぼ中央に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   このあとには、コメント化された ConverttagSTRUCT 関数の呼び出しが続きます。

   この関数の名前をステップ 996 ページの『5.a』 で追加した関数の名前に変更します。 コメント文字を削除して、関数を活動状態にします。 関数がいくつかある場合は、各関数ごとに呼び出しを作成します。

   c. ソース・ファイルの最初の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   ここに、ステップ 996 ページの『3』 で追加した関数のプロトタイプ・ステートメントを挿入します。
6. 次のコマンド・ファイルを作成します。

   ```
   cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
   MYFORMAT.C
   ```

```
MYFORMAT.DEF
```

ここで *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーです。

7. このコマンド・ファイルを発行して、出口を DLL ファイルとしてコンパイルします。
8. IBM MQ データ・ディレクトリーの下にある出口サブディレクトリーに出力を入れます。 32 ビット・システムに出口をインストールするためのデフォルト・ディレクトリーは *MQ_DATA_PATH*\Exits で、64 ビット・システムの場合は *MQ_DATA_PATH*\Exits64 になります。

   データ変換出口の検索では、レジストリー内パスが使われます。 レジストリー・フォルダーは次のとおりです。

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

   レジストリー・キーは ExitsDefaultPath です。 このパスは、それぞれのキュー・マネージャーで設定できます。出口の検索は、このパスでのみ行われます。

注 :

1. CRTMQCVX がパックされた構造体を使用している場合は、すべての IBM MQ アプリケーションをこの方法でコンパイルする必要があります。
2. データ変換出口プログラムは、必ず再入可能でなければなりません。
3. MQXCNVC は、データ変換出口から発行できるただ 1 つの MQI 呼び出しです。

Windows *Windows* オペレーティング・システムにおける出口ロード・ファイルとスイッチ・ロード・ファイル

IBM WebSphere MQ for Windows 7.5 キュー・マネージャー・プロセスは 32 ビットです。 そのため、64 ビット・アプリケーションを使用する際に、いくつかのタイプの出口ロード・ファイルや XA スイッチ・ロード・ファイルで、32 ビット・バージョンのものもキュー・マネージャーで使用できるようにしておく必要があります。 32 ビット・バージョンの出口ロード・ファイルまたは XA スイッチ・ロード・ファイルが必要であるのに、それが使用可能になっていない場合は、関連する API 呼び出しまたはコマンドが失敗します。

ExitPath の qm.ini file では、2 つの属性がサポートされています。 これらは ExitsDefaultPath=*MQ_INSTALLATION_PATH*\exits および ExitsDefaultPath64=*MQ_INSTALLATION_PATH*\exits64 です。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。 これらの属性を使用すると、適切なライブラリーを確実に見つけることができます。 さらに、出口が IBM MQ クラスターで使用される場合は、この属性を使用することによって、リモート・システム上に適切なライブラリーが確実に見つかるようにすることができます。

以下の表はさまざまなタイプの出口ロード・ファイルとスイッチ・ロード・ファイルをリストしたもので、それぞれ 32 ビット・アプリケーションが使用されるときと 64 ビット・アプリケーションが使用されるときに、32 ビット・バージョンと 64 ビット・バージョンのどちら (あるいは両方) が必要になるかを示しています。

ファイル・タイプ	32 ビット・アプリケーション	64 ビット・アプリケーション
API 出口 (API exit)	32 ビットおよび 64 ビット	64 ビット
データ変換出口	32 ビット	64 ビット
サーバー・チャネル出口 (全タイプ)	64 ビット	64 ビット
クライアント・チャネル出口 (全タイプ)	32 ビット	64 ビット
インストール可能サービス出口	64 ビット	64 ビット

ファイル・タイプ	32 ビット・アプリケーション	64 ビット・アプリケーション
クラスター WLM 出口	64 ビット	64 ビット
Pub/Sub ルーティング出口	64 ビット	64 ビット
データベース・スイッチ・ロード・ファイル	32 ビットおよび 64 ビット	64 ビット
外部トランザクション・マネージャー AX ライブラリー	32 ビット	64 ビット
事前接続出口	32 ビット	64 ビット

リポジトリーから接続前出口を使用した接続定義の参照

IBM MQ MQI clients は、リポジトリーを検索して、接続前出口ライブラリーを使用して接続定義を取得するように構成できます。

概要

クライアント・アプリケーションは、クライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続できます。 一般に、CCDT ファイルは中心にあるネットワーク・ファイル・サーバーにあり、 それを参照するクライアントが存在します。 CCDT ファイルを参照するさまざまなクライアント・アプリケーションを管理することは難しいため、柔軟なアプローチとして、クライアント定義をグローバル・リポジトリー (LDAP ディレクトリー、WebSphere Registry and Repository、または他のリポジトリー) に保管することができます。 クライアント接続定義をリポジトリーに保管すると、より簡単にクライアント接続定義を管理できるようになり、アプリケーションが適切で最も新しいクライアント接続定義にアクセスできるようになります。

MQCONN/X 呼び出しの実行時に、IBM MQ MQI client はアプリケーションが指定した 接続前出口ライブラリーをロードし、接続定義を取り出すために出口機能を呼び出します。 その後、取り出された接続定義を使用して、キュー・マネージャーへの接続が確立されます。 呼び出される出口ライブラリーと関数の詳細は、mqclient.ini 構成ファイルで指定されます。

構文

```
void MQ_PRECONNECT_EXIT (pExitParms, pQMgrName, ppConnectOpts, pCompCode, pReason);
```

パラメーター

pExitParms
タイプ: PMQNXP 入出力

PreConnection 出口パラメーター構造体。

この構造体は、出口の呼び出し側によって割り振られて維持されます。

pQMgrName
タイプ: PMQCHAR 入出力

キュー・マネージャーの名前。

入力では、このパラメーターは、**QMgrName** パラメーターを介して MQCONN API 呼び出しに提供されるフィルター・ストリングです。 このフィールドはブランクであるか、内容が明示的に指定されているか、特定のワイルドカード文字が含まれる場合があります。 このフィールドは出口によって変更されます。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。

ppConnectOpts
タイプ: ppConnectOpts 入出力

MQCONNX のアクションを制御するオプション。

これは、 MQCONN API 呼び出しのアクションを制御する MQCNO 接続オプション構造へのポインターです。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。 MQCNO 構造がアプリケーションによって元々提供されなかった場合を含め、MQI クライアントは常に MQCNO 構造を出口に提供します。 アプリケーションが MQCNO 構造を提供した場合、 クライアントは複製を作成して出口に渡し、そこで構造が変更されます。 クライアントは MQCNO の所有権を保持します。

MQCNO を介して参照される MQCD は、配列を介して提供されたすべての接続定義より優先されます。 クライアントはキュー・マネージャーに接続するために MQCNO 構造体を使用し、その他の定義は無視されます。

pCompCode
タイプ: PMQLONG 入出力

完了コード。

出口完了コードを受け取る MQLONG へのポインター。 値は、次のいずれかでなければなりません。

- MQCC_OK - 正常終了。
- MQCC_WARNING - 警告 (部分的に完了)
- MQCC_FAILED - 呼び出しの失敗。

pReason
タイプ: PMQLONG 入出力

pCompCode を修飾する理由。

出口理由コードを受け取る MQLONG へのポインター。 完了コードが MQCC_OK の場合、以下の値だけが有効です。

- MQRC_NONE - (0, x'000') 報告する理由はありません。

完了コードが MQCC_FAILED または MQCC_WARNING の場合、出口関数は理由コード・フィールドを任意の有効な MQRC_* 値に設定できます。

C 言語での呼び出し

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

パブリッシュ出口の作成とコンパイル

キュー・マネージャーでパブリッシュ出口を構成し、パブリッシュされたメッセージがサブスクライバーによって受信される前にその内容を変更できるようにすることが可能です。 また、メッセージ・ヘッダーの変更や、メッセージをサブスクリプションに送信させないことも可能です。

注 : パブリッシュ出口は z/OS ではサポートされていません。

パブリッシュ出口を使用して、サブスクライバーに送達されるメッセージを検査および変更できます。

- 各サブスクライバーにパブリッシュされるメッセージの内容を検査する
- 各サブスクライバーにパブリッシュされるメッセージの内容を変更する
- メッセージが入れられるキューを変更する
- サブスクライバーへのメッセージの送達を停止する

パブリッシュ出口の作成

942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』の手順を使用すると、出口の作成およびコンパイルに役立ちます。

パブリッシュ出口のプロバイダーは、出口で実行する内容を定義します。 ただし、この出口は、MQPSXP で定義された規則に従う必要があります。

IBM MQ では、MQ_PUBLISH_EXIT エントリー・ポイントの実装は提供されません。 C 言語の typedef 宣言が用意されています。 typedef を使用することにより、ユーザー作成出口のパラメーターを正しく宣言します。 次の例は、typedef 宣言の使用方法を示しています。

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
/* C language statements to perform the function of the exit */
}
```

パブリッシュ出口は、以下の操作の結果として、キュー・マネージャー・プロセスで実行されます。

- 1 つ以上のサブスクライバーにメッセージが送信されるパブリッシュ操作
- 1 つ以上の保存メッセージが送信されるサブスクライブ操作
- 1 つ以上の保存メッセージが送信されるサブスクリプション要求操作

接続のためにパブリッシュ出口が呼び出された場合、初めての呼び出し時に *ExitReason* コードが MQXR_INIT に設定されます。 パブリッシュ出口を使用したら、接続が切断される前に、*ExitReason* コードを MQXR_TERM に指定して、その出口が呼び出されます。

パブリッシュ出口が構成されていても、キュー・マネージャーの開始時にロードできない場合、そのキュー・マネージャーについてはパブリッシュ/サブスクライブ・メッセージ操作が禁止されます。 問題を修正するか、パブリッシュ/サブスクライブ・メッセージングが再び使用可能になる前にキュー・マネージャーを再始動する必要があります。

パブリッシュ出口に必要な IBM MQ の各接続は、出口をロードできないか、または出口を初期化できない場合があります。 出口をロードまたは初期化できない場合、パブリッシュ出口に必要なパブリッシュ/サブスクライブ操作は、その接続には使用不可になります。 IBM MQ の理由コード MQRC_PUBLISH_EXIT_ERROR により、操作に失敗します。

パブリッシュ出口が呼び出されるコンテキストは、アプリケーションによるキュー・マネージャーへの接続です。 ユーザー・データ域は、パブリッシュ操作を実行している接続ごとにキュー・マネージャーによって維持されます。 出口は、各接続のユーザー・データ域に情報を保持できます。

パブリッシュ出口は一部の MQI 呼び出しを使用できます。 使用できる MQI 呼び出しは、メッセージ・プロパティーを操作するものだけです。 これらの呼び出しを以下に示します。

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

パブリッシュ出口で宛先キュー・マネージャーまたはキュー名が変更された場合、新規の権限検査は実行されません。

パブリッシュ出口のコンパイル

パブリッシュ出口は動的にロードされるライブラリーであり、チャネル出口と考えることができます。 出口のコンパイルについては、942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』を参照してください。

サンプル・パブリッシュ出口

サンプル出口プログラムは、amqspse0.c と呼ばれます。 出口が初期化、パブリッシュ、または終了の中のどの操作のために呼び出されたかに応じて、別のメッセージがログ・ファイルに書き込まれます。 また、ストレージを適切に割り振りまたは解放するための出口ユーザー域フィールドの使用方法を示します。

パブリッシュ出口の構成

特定の属性を定義して、パブリッシュ出口を構成する必要があります。

Windows および Linux では、IBM MQ エクスプローラーを使用して属性を定義できます。 属性は、「キュー・マネージャー・プロパティー」ページの「パブリッシュ/サブスクライブ」の下で定義されます。

AIX and Linux システム上の qm.ini ファイルでパブリッシュ出口を構成するには、 PublishSubscribe というスタンザを作成します。 PublishSubscribe スタンザの属性は次のとおりです。

PublishExitPath=[path]|*module_name*
パブリッシュ出口コードを含むモジュールの名前とパス。 このフィールドの最大長は MQ_EXIT_NAME_LENGTH です。 デフォルトでは、パブリッシュ出口はありません。

PublishExitFunction=*function_name*
パブリッシュ出口コードを含むモジュールへの関数エントリー・ポイントの名前。 このフィールドの最大長は MQ_EXIT_NAME_LENGTH です。

IBM i IBM i では、プログラムが使用される場合には、PublishExitFunction を省略します。

PublishExitData=*string*
キュー・マネージャーがパブリッシュ出口を呼び出している場合、入力として MQPSXP 構造体が渡されます。 **PublishExitData** 属性を使用して指定されたデータは、構造の *ExitData* フィールドに入ります。 ストリングの最大長は、MQ_EXIT_DATA_LENGTH 個の文字になります。 デフォルトは 32 個のブランク文字です。

クラスター・ワークロード出口の作成とコンパイル

クラスターのワークロード管理をカスタマイズするには、クラスター・ワークロード出口プログラムを作成します。 メッセージをルーティングするときに、1 日のさまざまな時間のチャネル使用コストやメッセージの内容を考慮に入れる場合があります。 これらは、標準ワークロード管理アルゴリズムでは考慮されていない要因です。

ほとんどの場合、ワークロード管理アルゴリズムはニーズを満たします。 しかし、ワークロード管理を調整する独自のユーザー出口プログラムを使用する場合のために、IBM MQ には、クラスター・ワークロード出口というユーザー出口が組み込まれています。

ネットワークやメッセージに関する何らかの特定の情報を持っていて、それをワークロード・バランシングの操作に利用できる場合があります。 大容量のチャネルや低コストのネットワーク経路がわかっている場合や、メッセージをその内容に応じてルーティングしたい場合があります。 クラスター・ワークロード出口プログラムを作成するか、サード・パーティーが提供するものを使用するかを決定できます。

クラスター・ワークロード出口は、クラスター・キューへのアクセス時に呼び出されます。 MQOPEN、MQPUT1、および MQPUT によって呼び出されます。

MQOO_BIND_ON_OPEN が指定されている場合は、MQOPEN 時に選択されたターゲット・キュー・マネージャーに固定されます。 この場合は、出口は 1 回だけ実行されます。

MQOPEN 時にターゲット・キュー・マネージャーが固定されていない場合、ターゲット・キュー・マネージャーは MQPUT 呼び出しの時点で選択されます。 ターゲット・キュー・マネージャーが使用不可の場合、またはメッセージがまだ伝送キューにある間にターゲット・キュー・マネージャーに障害が起こった場合は、出口が再び呼び出されます。 新しいターゲット・キュー・マネージャーが選択されます。 メッセージが転

送されている間にメッセージ・チャネルに障害が起こり、メッセージがバックアウトされた場合は、新しいターゲット・キュー・マネージャーが選択されます。

Multi マルチプラットフォーム では、次回にキュー・マネージャーが開始したときに、キュー・マネージャーが新しいクラスター・ワークロード出口をロードします。

キュー・マネージャー定義にクラスター・ワークロード出口プログラム名が設定されていない場合は、 そのクラスター・ワークロード出口は呼び出されません。

クラスター・ワークロード出口の出口パラメーター構造体 MQWXP には、次のようなさまざまなデータが渡されます。

- メッセージ定義構造体 MQMD。
- メッセージ長パラメーター。
- メッセージのコピーまたはメッセージの一部。

非 z/OS プラットフォームでは、CLWLMode=FAST を使用した場合、オペレーティング・システムの各プロセスは、それぞれ独自の出口のコピーをロードします。 キュー・マネージャーへの接続が異なると、呼び出される出口のコピーも異なる可能性があります。 出口がデフォルトのセーフ・モード CLWLMode=SAFE で実行される場合は、独自の別プロセスで出口の単一コピーが実行されます。

クラスター・ワークロード出口の作成

z/OS z/OS の場合のクラスター・ワークロード出口の作成については、1004 ページの『IBM MQ for z/OS のクラスター・ワークロード出口プログラミング』を参照してください。

IBM MQ 9.1.0 以降、クラスター・ワークロード出口は、キュー・マネージャー・アドレス・スペースではなく、チャネル・イニシエーター・アドレス・スペースで実行されるようになりました。 クラスター・ワークロード出口がある場合は、キュー・マネージャー開始タスク・プロシージャーから CSQXLIB DD ステートメントを削除し、クラスター・ワークロード出口を格納するデータ・セットをチャネル・イニシエーター開始タスク・プロシージャーの CSQXLIB 連結に追加する必要があります。

Multi Multiplatforms の場合、クラスター・ワークロード出口で MQI 呼び出しを使用することはできません。 それ以外の点では、クラスター・ワークロード出口プログラムの作成とコンパイルの規則は、チャネル出口プログラムに適用される規則と同様です。 942 ページの『AIX, Linux, and Windows での出口とインストール可能サービスの作成』のステップ、およびサンプル・プログラム 1002 ページの『クラスター・ワークロード出口のサンプル』は、出口を作成およびコンパイルする場合に役立ちます。

チャネル出口の詳細については、970 ページの『チャネル出口プログラムの作成』を参照してください。

クラスター・ワークロード出口の構成

クラスター・ワークロード出口は、キュー・マネージャー定義で指定します。これは、ALTER QMGR コマンドにクラスター・ワークロード出口属性を指定することによって行います。 以下に例を示します。

```
ALTER QMGR CLWLEXIT(myexit)
```

関連資料

クラスター・ワークロード出口呼び出しとデータ構造体

クラスター・ワークロード出口のサンプル

IBM MQ にはクラスター・ワークロード出口プログラムのサンプルが組み込まれています。 このサンプルは、コピーして、独自のプログラムの基礎として使用することができます。

z/OS **IBM MQ for z/OS**

サンプルのクラスター・ワークロード出口プログラムは、アセンブラーおよび C で提供されています。 アセンブラー・バージョンは CSQ4BAF1 と呼ばれ、ライブラリー thlqual.SCSQASMS にあります。 C バージョンは CSQ4BCF1 という名前で、ライブラリー thlqual.SCSQC37S にあります。 thlqual

は、ご使用のインストール済み環境での IBM MQ データ・セットのターゲット・ライブラリー高位修飾子です。

Multi **IBM MQ for Multiplatforms**

サンプル・クラスター・ワークロード出口プログラムは C で提供されており、`amqswlm0.c` と呼ばれます。 以下の場所にあります。

表 *144. Multiplatforms* におけるクラスター・ワークロード出口プログラムのサンプルの場所

プラットフォーム	ファイル・パス
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`
IBM i IBM i	qmqm ライブラリー

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

このサンプル出口は、キュー・マネージャーが使用不可になった場合を除いて、すべてのメッセージを特定のキュー・マネージャーに経路指定します。 キュー・マネージャーに障害が起こった場合は、メッセージは別のキュー・マネージャーに経路指定されます。

メッセージの送信先とするキュー・マネージャーを示します。 キュー・マネージャー定義の `CLWLDATA` 属性に、クラスター受信チャネルの名前を指定します。 以下に例を示します。

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

出口を有効にするには、`CLWLEXIT` 属性に絶対パスと名前を指定します。

Linux AIX On AIX and Linux:

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows On Windows:

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS On z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

x は、使用するバージョンのプログラミング言語により、`'A'` または `'C'` とします。

IBM i IBM i で以下のいずれかのコマンドを使用します。

- MQSC コマンドを使用する場合:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  プログラム名とライブラリー名はいずれも 10 文字で指定します。なお、名前が 10 文字以下の場合は、左詰めに入力して残りをブランクで埋めます。
- CL コマンドを使用する場合:

```
CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
```

このように指定すると、IBM MQ は、提供されているワークロード管理アルゴリズムを使用する代わりにこの出口を呼び出して、選択されたキュー・マネージャーにすべてのメッセージを経路指定します。

z/OS IBM MQ for z/OS のクラスター・ワークロード出口プログラミング

クラスター・ワークロード出口は、z/OS **LINK** コマンドによって呼び出されたかのように見えます。 出口は、数多くの厳格なプログラミング規則に従っています。 ワークロード出口では、待機を引き起こすほとんどの SVC コマンドや、STAE または ESTAE を使用しないでください。

クラスター・ワークロード出口は、以下の状態では z/OS **LINK** によって呼び出されたかのように見えます。

- 許可がない問題プログラム状態
- 1 次アドレス・スペース制御モード
- 非仮想記憶間モード
- 非アクセス登録モード
- 31 ビット・アドレッシング・モード
- 記憶キー 8
- プログラム・キー・マスク 8
- TCB キー 8

チャネル・イニシエーターの開始タスク・プロシージャーの CSQXLIB DD ステートメントで指定されたデータ・セットに、リンク・エディットされたモジュールを入れます。 ロード・モジュールの名前は、キュー・マネージャー定義にワークロード出口名として指定されているものです。

IBM MQ for z/OS のワークロード出口を作成するときは、以下の規則が適用されます。

- 出口はアセンブラーまたは C で書く必要があります。 C を使用する場合は、*z/OS C/C++* プログラミング・ガイド SC09-4765 で説明されているように、システム出口の C システム・プログラミング環境に準拠している必要があります。
- MQXCLWLN 呼び出しを使用する場合は、CSQMFCLW (*thlqual*.SCSQLOAD に指定) でリンク・エディットします。
- 出口は、CSQXLIB DD ステートメントで定義された許可されていないライブラリーからロードされます。CSQXLIB に DISP=SHR が指定されている場合は、キュー・マネージャーの実行中に出口を更新し、次の MQCONN スレッドをキュー・マネージャーが開始するときに新バージョンを使用することができます。
- 出口は、再入可能であり、仮想ストレージ内のどこでも実行可能である必要があります。
- 出口は、戻り時に、環境をエントリー時の環境にリセットする必要があります。
- 出口は、入手したストレージをすべて解放するか、後の出口呼び出しで必ず解放されるようにする必要があります。
- MQI 呼び出しは許可されません。
- 出口には、待機を引き起こすおそれのあるシステム・サービスは使用しないでください。待機状態がキュー・マネージャーのパフォーマンスを著しく低下させることになります。 したがって、一般的に、SVC、PC、または I/O の使用は避けてください。
- 出口では、出口が接続するサブタスク内から発行する場合を除き、ESTAE および SPIE のいずれも発行しないでください。

注 : 出口の実行内容について絶対的な制約事項はありません。 ただし、ほとんどの SVC は待機を引き起こすため、次のコマンドを除き、これを避けてください。

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

ESTAE および ESPIE のエラー処理は IBM MQ が実行するエラー処理に干渉することがあるので、ESTAE および ESPIE は使用しないでください。 IBM MQ がエラーから回復できないか、出口プログラムがすべてのエラー情報を受信できない可能性があります。

システム・パラメーター EXITLIM は、出口を実行できる時間の長さを制限します。 EXITLIM のデフォルト値は 30 秒です。 戻りコード MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA') が表示された場合は、出口がループしている可能性があります。 出口の実行に 30 秒以上の時間が必要と思われる場合は、EXITLIM の値を増やしてください。

プロシージャー型アプリケーションの構築

IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

AIX AIX でのプロシージャー型アプリケーションの構築

AIX の資料では、作成したプログラムから実行可能アプリケーションを作成する方法について説明します。

このトピックでは、AIX で実行する IBM MQ for AIX アプリケーションの構築時に実行する必要がある追加タスクと、標準タスクに対する変更について説明します。 C、C++、および COBOL がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

IBM MQ for AIX を使用して実行可能なアプリケーションを作成するときに必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しをコーディングすると共に、使用する言語用の IBM MQ for AIX 組み込みファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、718 ページの『IBM MQ データ定義ファイル』を参照してください。

スレッド化されたサーバーまたはスレッド化されたクライアント・アプリケーションを実行するときは、環境変数 AIXTHREAD_SCOPE=S を設定してください。

AIX *AIX での C プログラムの作成*

このトピックでは、AIX での C プログラムの作成に必要なライブラリーのリンクについて説明します。

プリコンパイルされた C プログラムは、*MQ_INSTALLATION_PATH*/samp/bin ディレクトリーで提供されています。 ANSI コンパイラーを使用して、次のコマンドを実行してください。 64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

32 ビット・アプリケーションの場合:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

amqsput0 はサンプル・プログラムを示します。

64 ビット・アプリケーションの場合:

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

amqsput0 はサンプル・プログラムを示します。

V 9.3.5 XLC 17 コンパイラーを使用する 32 ビット・アプリケーションの場合:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

amqsput0 はサンプル・プログラムを示します。

V 9.3.5 XLC 17 コンパイラーを使用する 64 ビット・アプリケーションの場合:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

amqsput0 はサンプル・プログラムを示します。

C++ プログラム用の VisualAge C/C++ コンパイラーを使用している場合は、オプション -q namemangling=v5 を組み込んで、ライブラリーのリンク時にすべての IBM MQ シンボルが解決されるようにする必要があります。

IBM MQ MQI client for AIX のみがインストールされたマシン上でプログラムを使用したい場合は、そのプログラムを再コンパイルして、クライアント・ライブラリー (-lmqic) にリンクします。

ライブラリーのリンク

以下に、必要なライブラリーのリストを示します。

- プログラムを、IBM MQ が提供する適切なライブラリーとリンクしてください。

  スレッド以外の環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm.a	C 用サーバー
libmqic.a & libmqm.a	C 用クライアント

  スレッド環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm_r.a	C 用サーバー
libmqic_r.a & libmqm_r.a	C 用クライアント

  例えば、1 つのコンパイル単位から簡単な IBM MQ スレッド・アプリケーションを作成する場合は、次のようなコマンドを実行します。

  32 ビット・アプリケーションの場合:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

  amqsput0 はサンプル・プログラムを示します。

  64 ビット・アプリケーションの場合:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  amqsput0 はサンプル・プログラムを示します。

  V 9.3.5 XLC 17 コンパイラーを使用する 32 ビット・アプリケーションの場合:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

  amqsput0 はサンプル・プログラムを示します。

  V 9.3.5 XLC 17 コンパイラーを使用する 64 ビット・アプリケーションの場合:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

  amqsput0 はサンプル・プログラムを示します。

IBM MQ MQI client for AIX のみがインストールされたマシン上でプログラムを使用したい場合は、そのプログラムを再コンパイルして、クライアント・ライブラリー (`-lmqic`) にリンクします。

注:

1. 複数のライブラリーにリンクすることはできません。 そのため、スレッド化ライブラリーと非スレッド化ライブラリーの両方に同時にリンクすることはできません。
2. インストール可能なサービスを作成している場合 (詳細については IBM MQ の管理を参照)、スレッド化されていないアプリケーションの `libmqmzf.a` ライブラリー、およびスレッド化されたアプリケーション内の `libmqmzf_r.a` ライブラリーにリンクする必要があります。
3. IBM TXSeries、Encina、または BEA Tuxedo のような XA 準拠のトランザクション・マネージャーによって外部調整用のアプリケーションを作成している場合、スレッド化されていないアプリケーションの `libmqmxa.a` (トランザクション・マネージャーが long 型を 64 ビットとして扱う場合は `libmqmxa64.a`) や `libmqz.a` ライブラリー、およびスレッド化されたアプリケーションの `libmqmxa_r.a` (または `libmqmxa64_r.a`) および `libmqz_r.a` ライブラリーにリンクする必要があります。
4. トラステッド・アプリケーションは、IBM MQ のスレッド・ライブラリーにリンクする必要があります。 しかし、IBM MQ for AIX or Linux システム上のトラステッド・アプリケーションのスレッドに関しては、1 回の操作につき 1 つのスレッドのみ接続できます。
5. IBM MQ ライブラリーは、他のどの製品ライブラリーよりも先にリンクしなければなりません。

AIX *AIX での COBOL プログラムの作成*

IBM COBOL Set および Micro Focus COBOL を使用して AIX で COBOL プログラムを作成する際には、この情報を使用してください。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

- 32 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy32
  ```

  また、シンボリック・リンクは次のディレクトリーに作成されます。

  ```
  MQ_INSTALLATION_PATH/inc
  ```

- 64 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

以下の例では、**COBCPY** 環境変数を次のように設定します。

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

(32 ビット・アプリケーションの場合) および

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

(64 ビット・アプリケーションの場合)

プログラムを次のいずれかのライブラリー・ファイルとリンクする必要があります。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqmcb.a	COBOL 用サーバー (スレッド化されていないアプリケーションの場合)
libmqmcb_r.a	COBOL 用サーバー (スレッド化されたアプリケーションの場合)

ライブラリー・ファイル	プログラム / 出口タイプ
libmqicb.a	COBOL 用クライアント (スレッド化されていないアプリケーションの場合)
libmqicb_r.a	COBOL 用クライアント (スレッド化されたアプリケーションの場合)

作成するプログラムにより、IBM COBOL SET コンパイラーまたは Micro Focus COBOL コンパイラーを使用できます (以下参照)。

- amqm で始まるプログラムの場合には Micro Focus COBOL コンパイラーを使用します。および
- amq0 で始まるプログラムの場合にはどちらのコンパイラーを使用しても構いません。

IBM COBOL Set for AIX の使用による COBOL プログラムの作成

COBOL サンプル・プログラムが IBM MQ に付属しています。 このプログラムをコンパイルするには、以下のリストの該当するコマンドを入力してください。

32 ビットの非スレッド・サーバー・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

32 ビットの非スレッド・クライアント・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

32 ビットのスレッド・サーバー・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

32 ビットのスレッド・クライアント・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

64 ビットの非スレッド・サーバー・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

64 ビットの非スレッド・クライアント・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

64 ビットのスレッド・サーバー・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

64 ビットのスレッド・クライアント・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Micro Focus COBOL の使用による COBOL プログラムの作成

プログラムをコンパイルする前に環境変数を次のように設定します。

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Micro Focus COBOL を使用して 32 ビット COBOL プログラムをコンパイルするには、次のように入力します。

- COBOL 用サーバー

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- COBOL 用クライアント

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- COBOL 用スレッド・サーバー

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- COBOL 用スレッド・クライアント

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Micro Focus COBOL を使用して 64 ビット COBOL プログラムをコンパイルするには、次のように入力します。

- COBOL 用サーバー

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- COBOL 用クライアント

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- COBOL 用スレッド・サーバー

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- COBOL 用スレッド・クライアント

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

ここで amqminqx はサンプル・プログラムです。

設定する必要のある環境変数については、Micro Focus COBOL に関する資料を参照してください。

AIX AIX での CICS アプリケーション・プログラムの作成

AIX で CICS プログラムを作成する際には、以下の情報を使用してください。

XA スイッチ・モジュールを使用して CICS を IBM MQ とリンクします。 XA スイッチ構造体の詳細については、XA スイッチ構造体を参照してください。

他のトランザクション・メッセージ用の XA スイッチを開発できるよう、サンプルのソース・コード・ファイルが提供されています。 提供されているスイッチ・ロード・モジュールの名前は 1010 ページの表 145 にリストされています。

表 145. AIX 上の CICS アプリケーション・プログラムの必須コード: XA 初期化ルーチン		
説明	**C (ソース)**	**C (実行可能) - XAD.Stanza に追加する。 XAD.Stanza**
XA 初期化ルーチン	amqzscix.c	*amqzsc* - AIX の場合は CICS

製品に付属している IBM MQ スイッチ・ロード・ファイル *amqzsc* の事前ビルド・バージョンを使用します。

C トランザクションは、必ずスレッド・セーフ IBM MQ ライブラリー *libmqm_r.a* にリンクしてください。 COBOL ライブラリー *libmqmcb_r.a.*を使用して、COBOL トランザクションを作成します。

CICS トランザクションのサポートについては、「IBM MQ の管理 IBM MQ システム管理ガイド」に詳細情報が記載されています。

AIX *TXSeries CICS* のサポート

IBM MQ on AIX は、XA インターフェースを使用して TXSeries CICS をサポートします。 CICS アプリケーションがスレッド化されたバージョンの IBM MQ ライブラリーにリンクされていることを確認してください。

IBM COBOL Set for AIX または Micro Focus COBOL を使用して CICS プログラムを実行できます。 以下のセクションでは、IBM COBOL Set for AIX および icro Focus COBOL 上で実行する CICS プログラムの違いを説明します。

C または COBOL のいずれかで同じ CICS 領域にロードされる IBM MQ プログラムを作成します。 C および COBOL の MQI 呼び出しを組み合わせて同じ CICS 領域内に入れることはできません。 たいていの場合、使用される 2 番目の言語の MQI 呼び出しは失敗し、理由コード MQRC_HOBJ_ERROR が出されます。

IBM COBOL Set for AIX を使用する CICS COBOL プログラムの準備

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

IBM COBOL を使用する場合は、次のステップに従ってください。

1. 次の環境変数をエクスポートする。

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                -e _iwz_cobol_main  \
```

   LIB はコンパイラー指示を示します。

2. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l IBMCOB yourprog.ccp
```

Micro Focus COBOL の使用による CICS COBOL プログラムの作成

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

Micro Focus COBOL を使用する場合は、次のステップに従ってください。

1. 次のコマンドを使用して、ランタイム・ライブラリーに IBM MQ の COBOL ランタイム・ライブラリー・モジュールを追加する。

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

注: `cicsmkcobol` を使用すると、IBM MQ では COBOL アプリケーションから C プログラミング言語の MQI 呼び出しを行うことはできません。

既存のアプリケーションにこの種の呼び出しがある場合は、これらの関数を COBOL アプリケーションから `myMQ.so` などの独自のライブラリーに移動することをお勧めします。 関数を移動した後は、CICS 用の COBOL アプリケーションをビルドするときに IBM MQ ライブラリー `libmqmcbrt.o` を組み込まないでください。

さらに、COBOL アプリケーションで COBOL MQI 呼び出しが行われない場合は、`libmqmz_r` と `cicsmkcobol` をリンクしないでください。

これにより、Micro Focus COBOL 言語メソッド・ファイルが作成され、CICS ランタイム COBOL ライブラリーが IBM MQ for AIX or Linux システムを呼び出すことができるようになります。

注: `cicsmkcobol` は、以下の製品のいずれかをインストールする場合にのみ実行してください。

- Micro Focus COBOL の新規バージョンまたは新規リリース
- CICS for AIX の新規バージョンまたは新規リリース
- サポートされているデータベース製品の新規バージョンまたは新規リリース (COBOL トランザクション専用)
- IBM MQ の新規バージョンまたは新規リリース

2. 次の環境変数をエクスポートする。

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l COBOL -e yourprog.ccp
```

CICS C プログラムの作成

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

CICS の標準機能を使用して、次のステップに従って CICS C プログラムを作成してください。

1. 次の環境変数の **いずれか** をエクスポートする。
   - LDFLAGS = "-L/*MQ_INSTALLATION_PATH*lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L*MQ_INSTALLATION_PATH*lib -lmqm_r" export USERLIB
2. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l C amqscic0.ccs
```

CICS C サンプル・トランザクション

AIX IBM MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。 トランザクションは、伝送キュー SYSTEM.SAMPLE.CICS からメッセージを読み取ります。デフォルトのキュー・マネージャー上の WORKQUEUE で、メッセージの伝送ヘッダーに含まれているキュー名を持つローカル・キューにそれらを配置します。 障害はすべて、キュー SYSTEM.SAMPLE.CICS に送信されます。DLQ。 サンプル MQSC スクリプト AMQSCIC0.TST を使用して、これらのキューやサンプル入力キューを作成します。

IBM i IBM i でのプロシージャー型アプリケーションの構築

IBM i 資料では、 iSeries または System i システム上の IBM i で実行するために、作成したプログラムから実行可能アプリケーションを作成する方法について説明しています。

このトピックでは、 IBM i システムで実行する IBM MQ for IBM i プロシージャー型アプリケーションの構築時に実行する必要がある追加タスク、および標準タスクに対する変更について説明します。 COBOL、C、

C++、Java、および RPG の各プログラム言語がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。 Java プログラムの作成については、IBM MQ classes for Java の使用を参照してください。

実行可能な IBM MQ for IBM i アプリケーションの作成時に必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しを記述すると共に、使用する言語用の IBM MQ for IBM i データ定義ファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、718 ページの『IBM MQ データ定義ファイル』を参照してください。

IBM i *IBM i での C プログラムの作成*

IBM MQ for IBM i は、100 MB までのサイズのメッセージをサポートします。 ILE C でコーディングされたアプリケーション・プログラムで、16 MB より大きい IBM MQ メッセージをサポートするものは、これらのメッセージ用に十分なメモリーを割り振るために Teraspace コンパイラー・オプションを使用する必要があります。

C コンパイラー・オプションの詳細については、「*WebSphere Development Studio ILE C/C++ Programmer's Guide*」を参照してください。

C モジュールをコンパイルするために、IBM i コマンド **CRTCMOD** を使用することができます。 コンパイルする際には、インクルード・ファイル (QMQM) を含むライブラリーがライブラリー・リスト内にあることを確認してください。

次に、**CRTPGM** コマンドを使用して、コンパイラーの出力とサービス・プログラムとをバインドする必要があります。

表 *146.* スレッド化されていない環境およびスレッド化された環境での *CRTPGM* の例

環境のタイプ	**コマンド**	**プログラム / 出口タイプ**

スレッド化されていない環境	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM)`
		C 用サーバーまたはクライアント

スレッド化された環境	`CRTPGM PGM( pgmname )`
	`MODULE( pgmname )`
	`BNDSRVPGM(QMQM/LIBMQM_R)`
		C 用サーバーまたはクライアント

pgmname は、プログラムの名前です。

1012 ページの表 147 は、スレッド化されていない環境およびスレッド化された環境で IBM i 上に C プログラムを作成する際に必要なライブラリーをリストしています。

表 *147.* スレッド化されていない環境およびスレッド化された環境に必要なライブラリー

環境のタイプ	**ライブラリー・ファイル**	**プログラム / 出口タイプ**
スレッド化されていない環境	LIBMQM	C 用サーバー
	LIBMQIC & LIBMQM	C 用クライアント
スレッド化された環境	LIBMQM_R	C 用サーバー
	LIBMQIC_R & LIBMQM_R	C 用クライアント

IBM i *IBM i での COBOL プログラムの作成*

IBM i での COBOL プログラムの作成について、および COBOL プログラム内から MQI にアクセスする方法について説明します。

このタスクについて

COBOL プログラムの中から MQI にアクセスする場合、IBM MQ for IBM i には、サービス・プログラムによって提供される、結合プロシージャー呼び出しインターフェースがあります。 これは、IBM MQ for IBM i のすべての MQI 機能へのアクセス、スレッド化されたアプリケーションのサポートを提供するものです。 このインターフェースは、ILE COBOL コンパイラーでのみ使用できます。

MQI 機能には、標準の COBOL CALL 構文を使用してアクセスできます。

MQI で使用する名前付き定数と構造定義を含む COBOL コピー・ファイルは、ソース物理ファイル QMQM/QCBLLESRC に含まれています。

COBOL コピー・ファイルは、単一引用符 (') をストリング区切り文字として使用します。 IBM i COBOL コンパイラーでは、区切り文字は引用符 (") であると想定しています。 コンパイラーが警告メッセージを生成しないようにするには、コマンド **CRTCBLPGM**、 **CRTBNDCBL**、または **CRTCBLMOD** で OPTION (*APOST) を指定します。

コンパイラーが COBOL コピー・ファイル内の単一引用符 (') をストリング区切り文字として処理できるようにするには、コンパイラー・オプション ¥APOST を使用します。

注 : 動的呼び出しインターフェースは、IBM MQ 9.0 以降には装備されていません。

結合プロシージャー呼び出しインターフェースを使用するには、以下の手順を実行します。

手順

1. パラメーターを指定している **CRTCBLMOD** コンパイラーを使用して、モジュールを作成します。

   ```
   LINKLIT(*PRC)
   ```

2. **CRTPGM** コマンドを使用して、適切なパラメーターを指定してプログラム・オブジェクトを作成します。

   スレッド化されていないアプリケーションでは、次のようになります。

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   スレッド化されたアプリケーションでは、次のようになります。

   ```
   BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
   ```

   注 : V4R4 ILE COBOL コンパイラーを使用して作成され、PROCESS ステートメントに THREAD(SERIALIZE) オプションを含むプログラムの場合を除き、COBOL プログラムではスレッド化された IBM MQ ライブラリーを使用しないでください。 この方法で COBOL プログラムがスレッド・セーフにされていても、THREAD(SERIALIZE) は、モジュール・レベルで COBOL プロシージャーのシリアライゼーションを強制し、パフォーマンス全体に影響する可能性があるため、アプリケーションの設計時には注意してください。

   詳細については、「*WebSphere Development Studio: ILE COBOL Programmer's Guide*」および「*WebSphere Development Studio: ILE COBOL Reference*」を参照してください。

   CICS アプリケーションのコンパイルについて詳しくは、「*CICS for IBM i Application Programming Guide*」(SC41-5454) を参照してください。

IBM i *IBM i での CICS プログラムの準備*

IBM i で CICS プログラムを作成する際に必要なステップについて説明します。

EXEC CICS ステートメントと MQI 呼び出しを含むプログラムを作成するには、次のステップを実行してください。

1. 必要に応じて、CRTCICSMAP コマンドを使用してマップを作成します。

2. EXEC CICS コマンドをネイティブ言語ステートメントに変換します。 C プログラムの場合は CRTCICSC コマンドを使用します。 COBOL プログラムの場合は CRTCICSCBL コマンドを使用します。

   CICSOPT(*NOGEN) を CRTCICSC または CRTCICSCBL コマンドに含めます。 これにより処理が停止され、適切な CICS と IBM MQ サービス・プログラムを含めることができるようになります。 このコマンドは、デフォルトでコードを QTEMP/QACYCICS に書き込みます。
3. CRTCMOD コマンド (C プログラムの場合) または CRTCBLMOD コマンド (COBOL プログラムの場合) を使用して、ソース・コードをコンパイルします。
4. CRTPGM を使用して、コンパイル済みコードを適切な CICS および IBM MQ サービス・プログラムにリンクします。 これにより、実行可能プログラムが作成されます。

このようなコードの例を次に示します (これは、付属している CICS サンプル・プログラムをコンパイルします)。

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *IBM i での RPG プログラムの作成*

IBM MQ for IBM i を使用している場合は、RPG でアプリケーションを作成できます。

詳細については、1061 ページの『RPG での IBM MQ プログラムのコーディング (IBM i のみ)』、および IBM i アプリケーション・プログラミング・リファレンス (ILE/RPG) を参照してください。

IBM i *IBM i の SQL プログラミングに関する考慮事項*

SQL を使用して IBM i でアプリケーションを作成する際に必要なステップについて説明します。

プログラムに EXEC SQL ステートメントと MQI 呼び出しが含まれている場合、次のステップを実行してください。

1. EXEC SQL コマンドをネイティブ言語ステートメントに変換します。 C プログラムの場合は CRTSQLCI コマンドを使用します。 COBOL プログラムの場合は CRTSQLCBLI コマンドを使用します。

   OPTION(*NOGEN) を CRTSQLCI または CRTSQLCBLI コマンドに含めます。 これにより処理が停止され、適切な IBM MQ サービス・プログラムを含めることができるようになります。 このコマンドは、デフォルトでコードを QTEMP/QSQLTEMP に書き込みます。
2. CRTCMOD コマンド (C プログラムの場合) または CRTCBLMOD コマンド (COBOL プログラムの場合) を使用して、ソース・コードをコンパイルします。
3. CRTPGM を使用して、コンパイル済みコードを適切な IBM MQ サービス・プログラムにリンクします。 これにより、実行可能プログラムが作成されます。

このようなコードの例を次に示します (これは、ライブラリー SQLUSER にあるプログラム SQLTEST をコンパイルします)。

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM    PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Linux でのプロシージャー型アプリケーションの構築

ここでは、実行する IBM MQ for Linux アプリケーションの構築時に実行する必要がある追加作業、および標準作業に対する変更点について説明します。

C および C++ がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

Linux *Linux での C プログラムの作成*

プリコンパイルされた C プログラムは、 *MQ_INSTALLATION_PATH*/samp/bin ディレクトリーで提供されます。 ソース・コードからサンプルを作成するには、gcc コンパイラーを使用します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

通常の環境で作業してください。 64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

ライブラリーのリンク

Linux で C プログラムを作成するときに必要なライブラリーを以下の表にリストします。

- プログラムを、IBM MQ が提供する適切なライブラリーとリンクする必要があります。

  スレッド化されていない環境では、以下のいずれかのライブラリーにのみリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm.so	C 用サーバー
libmqic.so & libmqm.so	C 用クライアント

  スレッド化された環境では、以下のいずれかのライブラリーにのみリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm_r.so	C 用サーバー
libmqic_r.so & libmqm_r.so	C 用クライアント

注:

1. 複数のライブラリーにリンクすることはできません。 そのため、スレッド化ライブラリーと非スレッド化ライブラリーの両方に同時にリンクすることはできません。
2. インストール可能サービス (詳細については IBM MQ の管理を参照) を書き込む場合は、 `libmqmzf.so` ライブラリーにリンクする必要があります。
3. IBM TXSeries、Encina、または BEA Tuxedo のような XA 準拠のトランザクション・マネージャーによって外部調整用のアプリケーションを作成している場合、スレッド化されていないアプリケーションの `libmqmxa.so` (トランザクション・マネージャーが long 型を 64 ビットとして扱う場合は `libmqmxa64.so`) や `libmqz.so` ライブラリー、およびスレッド化されたアプリケーションの `libmqmxa_r.so` (または `libmqmxa64_r.so`) および `libmqz_r.so` ライブラリーにリンクする必要があります。
4. IBM MQ ライブラリーは、他のどの製品ライブラリーよりも先にリンクしなければなりません。

Linux *31* ビット・アプリケーションの構築

このトピックでは、さまざまな環境で 31 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、31 ビット、非スレッド

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C クライアント・アプリケーション、31 ビット、スレッド

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー・アプリケーション、31 ビット、非スレッド

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C サーバー・アプリケーション、31 ビット、スレッド

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ クライアント・アプリケーション、31 ビット、非スレッド

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

C++ クライアント・アプリケーション、31 ビット、スレッド

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、31 ビット、非スレッド

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

C++ サーバー・アプリケーション、31 ビット、スレッド

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、31 ビット、非スレッド

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C クライアント出口、31 ビット、スレッド

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー出口、31 ビット、非スレッド

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C サーバー出口、31 ビット、スレッド

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *32* ビット・アプリケーションの構築

このトピックでは、さまざまな環境で 32 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、32 ビット、非スレッド

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C クライアント・アプリケーション、32 ビット、スレッド

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー・アプリケーション、32 ビット、非スレッド

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C サーバー・アプリケーション、32 ビット、スレッド

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ クライアント・アプリケーション、32 ビット、非スレッド

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

C++ クライアント・アプリケーション、32 ビット、スレッド

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、32 ビット、非スレッド

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

C++ サーバー・アプリケーション、32 ビット、スレッド

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、32 ビット、非スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
```

```
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

C クライアント出口、32 ビット、スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー出口、32 ビット、非スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

C サーバー出口、32 ビット、スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *64 ビット・アプリケーションの構築*

このトピックでは、さまざまな環境で 64 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、64 ビット、非スレッド

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

C クライアント・アプリケーション、64 ビット、スレッド

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

C サーバー・アプリケーション、64 ビット、非スレッド

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

C サーバー・アプリケーション、64 ビット、スレッド

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

C++ クライアント・アプリケーション、64 ビット、非スレッド

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

C++ クライアント・アプリケーション、64 ビット、スレッド

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、64 ビット、非スレッド

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

C++ サーバー・アプリケーション、64 ビット、スレッド

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、64 ビット、非スレッド

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

C クライアント出口、64 ビット、スレッド

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C サーバー出口、64 ビット、非スレッド

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

C サーバー出口、64 ビット、スレッド

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *Linux での COBOL プログラムの作成*

Linux で COBOL プログラムを作成し、IBM COBOL for Linux on x86 および Micro Focus COBOL を使用して COBOL プログラムを作成する方法について説明します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

1. 32 ビット COBOL コピーブックは、以下のディレクトリーにインストールされます。

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

また、シンボリック・リンクは次のディレクトリーに作成されます。

```
MQ_INSTALLATION_PATH/inc
```

2. 64 ビット・プラットフォームでは、64 ビット COBOL コピーブックは以下のディレクトリーにインストールされます。

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

3. 以下の例では、COBCPY を次のように設定します。

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

32 ビット・アプリケーションの場合:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

64 ビット・アプリケーションの場合。

プログラムを次のいずれかとリンクする必要があります。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqmcb.so	COBOL 用サーバー
libmqicb.so	COBOL 用クライアント
libmqmcb_r.so	COBOL 用サーバー (スレッド化されたアプリケーションの場合)
libmqicb_r.so	COBOL 用クライアント (スレッド化されたアプリケーションの場合)

IBM COBOL for Linux on x86 を使用した COBOL プログラムの作成

IBM MQ には、サンプル COBOL プログラムが用意されています。 このプログラムをコンパイルするには、以下のリストの該当するコマンドを入力してください。

32 ビット非スレッド・サーバー・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

32 ビット非スレッド・クライアント・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

32 ビット・スレッド・サーバー・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

32 ビット・スレッド・クライアント・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

Micro Focus COBOL の使用による COBOL プログラムの作成

プログラムをコンパイルする前に環境変数を次のように設定します。

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Micro Focus COBOL を使用して 32 ビット COBOL プログラム (サポートされている場合) をコンパイルするには、次のように入力します。

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Micro Focus COBOL を使用して 64 ビット COBOL プログラムをコンパイルするには、次のように入力します。

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

ここで amqsput はサンプル・プログラムです。

必要な環境変数については、Micro Focus COBOL に関する資料を参照してください。

Windows Windows でのプロシージャー型アプリケーションの構築

Windows システムの資料では、ユーザーが作成したプログラムから実行可能アプリケーションを構築する方法を説明します。

このトピックでは、 Windows システムで実行する IBM MQ for Windows アプリケーションを構築する際に実行する必要がある追加タスクと、標準タスクに対する変更について説明します。 C、C++、COBOL、および Visual Basic プログラム言語がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

IBM MQ for Windows を使用して実行可能なアプリケーションを作成するときに必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しをコーディングすると共に、使用する言語用の IBM MQ for Windows 組み込みファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。詳しくは、718 ページの『IBM MQ データ定義ファイル』を参照してください。

Windows *Windows での 64 ビット・アプリケーションの構築*

32 ビット・アプリケーションと 64 ビット・アプリケーションの両方が、IBM MQ for Windows でサポートされています。 32 ビット形式と 64 ビット形式の両方の IBM MQ 実行可能ファイルとライブラリー・ファイルが提供されているので、処理しているアプリケーションに応じて該当するバージョンを使用してください。

実行可能ファイルとライブラリー

32 ビット・バージョンと 64 ビット・バージョンの IBM MQ ライブラリーは、以下の場所で提供されています。

表 *148. IBM MQ* ライブラリーの場所

ライブラリーのバージョン	ライブラリー・ファイルを含むディレクトリー
32 ビット	*MQ_INSTALLATION_PATH* \Tools\Lib
64 ビット	*MQ_INSTALLATION_PATH* \Tools\Lib64

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

32 ビット・アプリケーションは、マイグレーション後も引き続き正常に機能します。 32 ビット・ファイルは、以前のバージョンの製品と同じディレクトリーにあります。

64 ビット・バージョンを作成する場合は、*MQ_INSTALLATION_PATH* \Tools\Lib64 のライブラリー・ファイルを使用するように環境が構成されていることを確認する必要があります。 LIB 環境変数が、32 ビット・ライブラリーを含むフォルダーを参照するように設定されていないことを確認してください。

Windows *Windows での C プログラムの作成*

通常の Windows 環境で作業してください。IBM MQ for Windows には特別な環境は必要ありません。

64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

- プログラムを、IBM MQ が提供する適切なライブラリーとリンクしてください。

  ライブラリー・ファイル　プログラム / 出口タイプ

  MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib　32 ビット C 用サーバー

  MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib　32 ビット C 用クライアント

  MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib　32 ビット C 用クライアント、トランザクション調整あり

  MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib　64 ビット C 用サーバー

  MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib　64 ビット C 用クライアント

  MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib　64 ビット C 用クライアント、トランザクション調整あり

  MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

  次のコマンドは、amqsget0 サンプル・プログラムをコンパイルする例を示しています (Microsoft Visual C++ コンパイラーを使用)。

  32 ビット・アプリケーションの場合:

  ```
  cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
  ```

  64 ビット・アプリケーションの場合:

  ```
  cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
  ```

  注:

  - インストール可能なサービス (詳細は、IBM MQ の管理を参照) を作成する場合は、mqmzf.lib ライブラリーにリンクする必要があります。
  - XA 準拠のトランザクション管理プログラム (IBM TXSeries Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成している場合は mqmxa.lib または mqmxa.lib ライブラリーにリンクする必要があります。
  - CICS 出口を作成している場合は、mqmcics4.lib ライブラリーにリンクしてください。
  - IBM MQ ライブラリーは、他のどの製品ライブラリーよりも先にリンクしなければなりません。

- DLL は指定したパス (PATH) 内になければなりません。
- 可能な限りいつでも小文字を使用すると、IBM MQ for Windows システムから、小文字の使用が必要になる IBM MQ for AIX or Linux システムへと移行できます。

CICS および Transaction Server プログラムの作成

CICS IBM MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。 これは、CICS の標準機能を使用して作成します。 例えば、TXSeries for Windows 2000 の場合は次のようになります。

1. 環境変数を設定する (次のコードを 1 行で入力する)。

```
set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. USERLIB 環境変数を設定する。

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. 次のように入力して、サンプル・プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

これについては、「*Transaction Server for Windows NT Application Programming Guide*」*(CICS)* で説明されています。 *V4*。

CICS トランザクションのサポートの詳細については、IBM MQ の管理を参照してください。

Windows *Windows での COBOL プログラムの作成*

この情報を使用して、Windows での COBOL プログラムの作成、および CICS および Transaction Server プログラムの作成について学習します。

1. 32 ビットの COBOL コピーブックは、ディレクトリー *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook にインストールされます。
2. 64 ビット COBOL コピーブックは、ディレクトリー *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64 にインストールされます。
3. 以下の例では、CopyBook を次のように設定します。

```
CopyBook
```

   32 ビット・アプリケーションの場合:

```
CopyBook64
```

   64 ビット・アプリケーションの場合。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

Windows システムで COBOL プログラムを作成するには、IBM MQ で提供される次のいずれかのライブラリーにプログラムをリンクしてください。

ライブラリー・ファイル	プログラムまたは出口タイプ
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	Micro Focus COBOL 用 32 ビット・サーバー
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Micro Focus COBOL 用 32 ビット・クライアント

ライブラリー・ファイル	プログラムまたは出口タイプ
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	Micro Focus COBOL 用 64 ビット・サーバー
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	Micro Focus COBOL 用 64 ビット・クライアント

MQI クライアント環境でプログラムを実行するときは、DOSCALLS ライブラリーが COBOL ライブラリーまたは IBM MQ ライブラリーより前に来るようにしてください。

Micro Focus COBOL の使用による COBOL プログラムの作成

mqmcbb および mqiccbb ライブラリーではなく、 mqmcb.lib または mqiccb.lib のいずれかを使用して、既存の 32 ビット IBM MQ Micro Focus COBOL プログラムを再リンクします。

例えば、Micro Focus COBOL を使用してサンプル・プログラム amq0put0 をコンパイルするには、次のようにします。

1. COBCPY 環境変数を設定して、宛先を IBM MQ COBOL コピーブックに指定する (次のコードを 1 行で入力する)。

```
set COBCPY= MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. プログラムをコンパイルしてオブジェクト・ファイルを作成する。

```
cobol amq0put0 LITLINK
```

3. オブジェクト・ファイルを実行時システムにリンクする。
   - LIB 環境変数を設定して、宛先をコンパイラー COBOL ライブラリーに指定する。
   - IBM MQ サーバーで使用する場合には、以下の記述形式によりオブジェクト・ファイルをリンクする。

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - または、IBM MQ クライアントで使用する場合には、以下の記述形式によりオブジェクト・ファイルをリンクする。

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

CICS および Transaction Server プログラムの作成

IBM VisualAge COBOL を使用する TXSeries for Windows NT, V5.1 プログラムをコンパイルおよびリンクするには次のように行います。

1. 環境変数を設定する (次のコードを 1 行で入力する)。

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. USERLIB 環境変数を設定する。

```
set USERLIB=MQMCBB.LIB
```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l IBMCOB myprog.ccp
```

詳細については、「*Transaction Server for Windows NT, V4 Application Programming Guide*」を参照してください。

Micro Focus COBOL を使用する CICS for Windows V5 プログラムをコンパイルおよびリンクするには次のように行います。

- INCLUDE 変数を設定する。

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- COBCPY 環境変数を設定する。

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- COBOL オプションを設定する。
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

そして、以下のコードを実行します。

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Windows での Visual Basic プログラムの準備*

Microsoft Visual Basic プログラムを Windows で使用する場合に考慮する情報です。

Deprecated IBM MQ 9.0 以降、Microsoft Visual Basic 6.0 に対するサポートは非推奨になりました。 IBM MQ classes for .NET は、推奨される置換テクノロジーです。 詳しくは、.NET アプリケーションの開発を参照してください。

注 : 64 ビット・バージョンの Visual Basic モジュール・ファイルは提供されていません。

Windows で Visual Basic プログラムを準備するには、以下のことを行います。

1. 新規プロジェクトを作成する。
2. 製品に付属のモジュール・ファイル CMQB.BAS をプロジェクトに追加する。
3. 必要に応じて、次のモジュール・ファイルを追加する。
   - CMQBB.BAS: MQAI サポート
   - CMQCFB.BAS: PCF サポート
   - CMQXB.BAS: チャネル出口サポート
   - CMQPSB.BAS: パブリッシュ/サブスクライブ

Visual Basic 内から MQCONNXAny 呼び出しを使用する方法については、1057 ページの『Visual Basic でのコーディング』 を参照してください。

プロジェクト・コードでは、MQI 呼び出しの前に、必ずプロシージャー MQ_SETDEFAULTS を呼び出す必要があります。 このプロシージャーにより、MQI 呼び出しに必要なデフォルトの構造体がセットアップされます。

プロジェクトをコンパイルまたは実行する前に、条件付きコンパイル変数 *MqType* を設定して、 IBM MQ サーバーまたはクライアントのどちらを作成するかを指定します。 以下のように、 Visual Basic プロジェクトの *MqType* を、サーバーの場合は 1、クライアントの場合は 2 に設定します。

1.「Project (プロジェクト)」メニューを選択する。

2. 「*Name* プロパティー」(*Name* は現行プロジェクト名) を選択する。
3. ダイアログ・ボックスで「Make (作成)」タブを選択する。
4. サーバーの場合は、「Conditional Compilation Arguments (条件付きコンパイル引数)」フィールドで次のように入力する。

```
MqType=1
```

クライアントの場合は、次のように入力する。

```
MqType=2
```

関連概念

1057 ページの『Visual Basic でのコーディング』
Microsoft Visual Basic で IBM MQ プログラムをコーディングする際に考慮すべき情報。 Visual Basic は、Windows でのみサポートされます。

関連資料

925 ページの『Visual Basic アプリケーションと IBM MQ MQI client・コードとのリンク』
Windows 上で IBM MQ MQI client コードと Microsoft Visual Basic アプリケーションをリンクできます。

Windows *SSPI* セキュリティー出口

IBM MQ for Windows には、IBM MQ MQI client 用および IBM MQ サーバー用のセキュリティー出口が備えられています。 これはチャネル出口プログラムであり、セキュリティー・サービス・プログラミング・インターフェース (SSPI) を使用することによって、IBM MQ チャネルの認証を行うことができます。 SSPI には、Windows システムの統合セキュリティー機能が備えられています。

セキュリティー・パッケージは、security.dll または secur32.dll のいずれかからロードします。 これらの DLL は、ご使用のオペレーティング・システム (OS) で提供されています。

片方向認証は、NTLM 認証サービスを使用して備えられています。 双方向認証は Kerberos 認証サービスを使用して備えられています。

セキュリティー出口プログラムは、ソースおよびオブジェクト形式で提供されます。 オブジェクト・コードをそのまま使用することもできますし、あるいはソース・コードを開始点として使用して独自のユーザー出口プログラムを作成することもできます。

1140 ページの『Windows での SSPI セキュリティー出口の使用』も参照してください。

セキュリティー出口の概要

セキュリティー出口は、2 つのセキュリティー出口プログラムの間のセキュア接続を形成します。このうち、1 つのプログラムはメッセージ・チャネル・エージェント (MCA) を送信するためのもので、もう 1 つは MCA を受信するためのものです。

セキュア接続を開始する方のプログラム、つまり、MCA セッションが確立された後に制御を得る最初のプログラムは、コンテキスト・イニシエーター と呼ばれます。 これのパートナーとなるプログラムは、コンテキスト・アクセプター と呼ばれます。

以下の表は、チャネル・タイプ (コンテキスト・イニシエーターと、それに関連したコンテキスト・アクセプター) の一例を示しています。

表 *149.* コンテキスト・イニシエーターとそれに関連したコンテキスト・アクセプター

コンテキスト・イニシエーター	**コンテキスト・アクセプター**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

セキュリティー出口プログラムには以下の 2 つの入り口点があります。

- **SCY_NTLM**

  NTLM 認証サービス (片方向認証を実行する) を使用します。 NTLM を使用すると、サーバーはクライアントの ID を検証することができます。 クライアントがサーバーの ID を検証したり、1 つのサーバーが他のサーバーを検証したりすることはできません。 NTLM 認証は、サーバーの真正を前提とするネットワーク環境用に設計されたものです。

- **SCY_KERBEROS**

  これは Kerberos 相互認証サービスを使用します。 Kerberos プロトコルは、ネットワーク環境内のサーバーが本物であることを前提としません。 ネットワーク接続の両側のパーティーが他のパーティーの ID を検証できます。 つまり、サーバーはクライアントや他のサーバーの ID を検証でき、クライアントはサーバーの ID を検証できるということです。

セキュリティー出口の実行内容

このトピックでは、SSPI チャネル出口プログラムの実行内容について説明します。

提供されているチャネル出口プログラムは、セッションの確立時に、パートナー・システムの片方向認証か双方向 (相互) 認証のいずれかを提供します。 特定のチャネルに関して、各出口プログラムには関連したプリンシパル があります (ユーザー ID と同様。1027 ページの『IBM MQ アクセス制御および Windows プリンシパル』を参照してください)。 2 つの出口プログラム間の接続は、2 つのプリンシパル間の関連とも言うことができます。

基本となるセッションが確立された後、2 つのセキュリティー出口プログラム (1 つは MCA の送信用、もう 1 つは MCA の受信用) の間のセキュア接続が確立されます。 以下の順序で操作が行われます。

1. 各プログラムは、明示的なログイン操作などによって、特定のプリンシパルと関連付けられます。
2. コンテキスト・イニシエーターは、セキュリティー・パッケージのパートナー (Kerberos の場合は名前つきパートナー) とのセキュア接続を要求し、トークン (token1 と呼ばれる) を受け取ります。 既に確立されている基本セッションを使用して、トークンがパートナー・プログラムに送信されます。
3. パートナー・プログラム (コンテキスト・アクセプター) は token1 をセキュリティー・パッケージに渡します。これにより、コンテキスト・イニシエーターが本物かどうか検査されます。 NTLM の場合、接続はこれで確立されます。
4. Kerberos が提供するセキュリティー出口 (つまり、相互認証) の場合、セキュリティー・パッケージは 2 つ目のトークン (token2 と呼ばれる) を生成します。コンテキスト・アクセプターは基本セッションを使用することによってこのトークンをコンテキスト・イニシエーターに戻します。
5. コンテキスト・イニシエーターは token2 を使用して、コンテキスト・アクセプターが本物かどうか検査します。
6. この段階で両方のアプリケーションにおいてパートナーの真正が証明されると、セキュア (認証済み) 接続が確立されます。

IBM MQ アクセス制御および Windows プリンシパル

IBM MQ が提供するアクセス制御は、ユーザーとグループを基にします。 Windows が提供する認証は、ユーザーや servicePrincipalName (SPN) などの、プリンシパルを基にします。 servicePrincipalName の場合、1 つのユーザーに関して多数が関連付けられる場合があります。

SSPI セキュリティー出口では、認証の際に、関連した Windows プリンシパルが使用されます。 Windows の認証が成功すると、出口は Windows プリンシパルに関連したユーザー ID を IBM MQ に渡し、アクセス制御が行えるようにします。

認証に関連する Windows プリンシパルは、使用される認証タイプによって異なります。

- NTLM 認証の場合、コンテキスト・イニシエーター用の Windows プリンシパルは、実行中のプロセスに関連したユーザー ID になります。 この認証は片方向なので、コンテキスト・アクセプターに関連したプリンシパルは関係ありません。
- Kerberos 認証の場合の CLNTCONN チャネルでは、Windows プリンシパルは、実行中のプロセスに関連したユーザー ID になります。 それ以外のチャネルでは、Windows プリンシパルは、QueueManagerName に次の接頭部を追加して形成される servicePrincipalName になります。

```
ibmMQSeries/
```

z/OS z/OS でのプロシージャー型アプリケーションの構築

CICS、IMS、および z/OS の資料は、これらの環境で実行されるアプリケーションの構築方法を説明しています。

以下のトピックでは、追加の作業および標準作業に対する変更点について説明します。これらの作業は、これらの環境用の IBM MQ for z/OS アプリケーションの構築時に行う必要があります。 COBOL、C、C++、アセンブラー、PL/I の各プログラム言語がサポートされています。 (C++ アプリケーションの作成については、C++ の使用を参照してください。)

実行可能な IBM MQ for z/OS アプリケーションを作成するときに必要な作業は、プログラムを作成するプログラム言語とアプリケーションが実行される環境によって異なります。

プログラムで MQI 呼び出しを記述すると共に、使用する言語用の IBM MQ for z/OS データ定義ファイルを組み込むために、適切な言語ステートメントを追加してください。 したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、718 ページの『IBM MQ データ定義ファイル』を参照してください。

注記

thlqual は、z/OS のインストール・ライブラリーの高水準修飾子を表します。

z/OS 実行するプログラムの作成

IBM MQ アプリケーション用のプログラムの作成後、実行可能なアプリケーションを作成するには、そのプログラムをコンパイルまたはアセンブルする必要があります。さらに、結果として得られるオブジェクト・コードを、サポートする各環境ごとに IBM MQ for z/OS が提供するスタブ・プログラムとリンク・エディットする必要があります。

プログラムの作成方法は、アプリケーションが実行される環境 (バッチ、CICS、IMS (BMP または MPP)、または Linux または z/OS UNIX System Services) と、z/OS のインストール先でのデータ・セットの構造によって異なります。

1035 ページの『IBM MQ スタブの動的呼び出し』では、IBM MQ スタブをリンク・エディットする代わりにプログラムで MQI 呼び出しを行う方法について説明します。 この方法は、すべての言語および環境で使用できるわけではありません。

プログラムが動作しているバージョンの IBM MQ for z/OS のスタブ・プログラムよりも高いレベルのスタブ・プログラムをリンク・エディットしないでください。 例えば、MQSeries for OS/390® V5.2 で動作しているプログラムを IBM MQ for z/OS V7 で提供されるスタブ・プログラムとリンク・エディットしてはなりません。

z/OS *64* ビット *C* アプリケーションの構築

z/OS では、64 ビット C アプリケーションは LP64 コンパイラーおよびバインダー・オプションを使用して構築します。 IBM MQ for z/OS *cmqc.h* ヘッダー・ファイルは、このオプションがコンパイラーに提供されると認識し、64 ビット操作に適した IBM MQ データ型および構造体を生成します。

必要な調整セマンティックに適したダイナミック・リンク・ライブラリー (DLL) を使用するには、このオプションを指定して C コードを構築する必要があります。 これを行うには、以下の表に定義されている適切なサイド・デックを使用して、コンパイル済みコードをバインドします。

表 150. 各調整セマンティックに必要なサイド・デック名	
調整	**サイド・デック名**
シングル・フェーズ・コミット MQI	CSQBMQ2X
RRS verb を使用する RRS 調整の 2 フェーズ・コミット	CSQBRR2X
MQI verb を使用する RRS 調整の 2 フェーズ・コミット	CSQBRI2X

注 : 31 ビット C アプリケーションの場合は、1031 ページの『31 ビット Language Environment または XPLINK を使用した z/OS バッチ・アプリケーションの構築』で説明されているように、呼び出しインターフェース (Language Environment または XPLINK) のコンパイラー・オプションも設定します。 64 ビット C アプリケーションの場合、サポートされるリンケージは XPLINK のみであるため、呼び出しインターフェースは指定しません。

*z/OS XL C/C++*で提供される EDCQCB JCL プロシージャーを使用して、次のようにバッチ・ジョブとして単一フェーズ・コミット IBM MQ プログラムを作成します。

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',             < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',             < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//         DD
//         DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

z/OS UNIX System Services において RRS で調整されたプログラムを構築するには、次のようにコンパイルしてリンクします。

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *z/OS* バッチ・アプリケーションの構築

z/OS バッチ・アプリケーションの構築方法および構築時に考慮すべき手順について説明します。

z/OS バッチで実行する IBM MQ for z/OS 用のアプリケーションを作成するには、以下のタスクを実行するジョブ制御言語 (JCL) を作成します。

1. プログラムをコンパイル (またはアセンブル) して、オブジェクト・コードを作成する。 コンパイル用の JCL には、SYSLIB ステートメントが含まれていなければなりません。この SYSLIB ステートメントによって製品データ定義ファイルがコンパイラーで利用できるようになります。 データ定義は、以下の IBM MQ for z/OS ライブラリーで提供されます。
   - COBOL の場合、**thlqual**.SCSQCOBC
   - アセンブラー言語の場合、**thlqual**.SCSQMACS
   - C の場合、**thlqual**.SCSQC370
   - PL/I の場合、**thlqual**.SCSQPLIC
2. C アプリケーションの場合は、ステップ 1029 ページの『1』 で作成したオブジェクト・コードを事前にリンクする。
3. PL/I アプリケーションの場合は、コンパイラー・オプション EXTRN(SHORT) を使用する。
4. ステップ 1029 ページの『1』 (C アプリケーションの場合はステップ 1029 ページの『2』) で作成したオブジェクト・コードをリンク・エディットして、ロード・モジュールを作成する。 オブジェクト・コードをリンク・エディットする場合には、IBM MQ for z/OS バッチ・スタブ・プログラム (CSQBSTUB、または RRS スタブ・プログラム CSQBRRSI もしくは CSQBRSTB のどちらか) を組み込む必要があります。

CSQBSTUB
IBM MQ for z/OS が提供する単一フェーズ・コミット

CSQBRRSI
RRS が MQI を使用して提供する 2 フェーズ・コミット

CSQBRSTB
RRS が直接提供する 2 フェーズ・コミット

注 :

a. CSQBRSTB を使用する場合は、アプリケーションを SYS1.CSSLIB の ATRSCSS とリンク・エディットする必要もあります。 1030 ページの図 113 と 1030 ページの図 114 に、これを行うための JCL の一部を示します。 スタブは使用する言語に依存しないもので、ライブラリー **thlqual**.SCSQLOAD で提供されます。

b. アプリケーションが Language Environment で実行されている場合は、1031 ページの『31 ビット Language Environment または XPLINK を使用した z/OS バッチ・アプリケーションの構築』で説明されているように、代わりに Language Environment DLL を使用してリンク・エディットを行う必要があります。

5. ロード・モジュールをアプリケーション・ロード・ライブラリーに格納する。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

図 *113.* バッチ環境でオブジェクト・モジュールをリンク・エディットするための *JCL* の一部 *(*単一フェーズ・コミットを使用する場合*)*

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
⋮
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
⋮
/*
```

図 *114.* バッチ環境でオブジェクト・モジュールをリンク・エディットするための *JCL* の一部 *(2* フェーズ・コミットを使用する場合*)*

バッチ・プログラムまたは RRS プログラムを実行するには、ライブラリー **thlqual**.SCSQAUTH および **thlqual**.SCSQLOAD を STEPLIB または JOBLIB データ・セット連結に組み込む必要があります。

TSO プログラムを実行するには、ライブラリー **thlqual**.SCSQAUTH および **thlqual**.SCSQLOAD を、TSO セッションで使用する STEPLIB に組み込む必要があります。

z/OS UNIX System Services シェルからバッチ・プログラムを実行するには、 次のように、ライブラリー **thlqual**.SCSQAUTH および **thlqual**.SCSQLOAD を $HOME?.profile 内の STEPLIB 指定に追加します。

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *31 ビット Language Environment または XPLINK を使用した z/OS バッチ・アプリケーションの構築*

IBM MQ for z/OS には、アプリケーションのリンク・エディット時に使用する必要があるダイナミック・リンク・ライブラリー (DLL)・セットが用意されています。

アプリケーションが以下の呼び出しインターフェースのいずれかを使用できるようにするライブラリーには、2 つのバリアントがあります。

- 31 ビット Language Environment 呼び出しインターフェース。
- 31 ビット XPLINK 呼び出しインターフェース。 z/OS XPLINK は、C アプリケーションに使用できるハイパフォーマンスの呼び出し規則です。 z/OS 2.2 の資料の XPLINK | NOXPLINK を参照してください。

DLL を使用するには、 アプリケーションを、これまでのバージョンで提供されていたスタブではなく、 サイドデッキ と呼ばれるものに対してバインドまたはリンクします。 サイドデッキは、(SCSQLOAD ライブラリーではなく) SCSQDEFS ライブラリーにあります。

表 *151.* ダイナミック・リンク・ライブラリーのバリアント

コミット	31 ビット Language Environment DLL	31 ビット XPLINK DLL	同等のスタブ名
1 フェーズ・コミット MQI ライブラリー	CSQBMQ1	CSQBMQ1X	CSQBSTUB
RRS トランザクション制御 verb を使用して RRS 調整を行う 2 フェーズ・コミット	CSQBRR1	CSQBRR1X	CSQBRSTB
MQI トランザクション制御 verb を使用して MQI 調整を行う 2 フェーズ・コミット	CSQBRI1	CSQBRI1X	CSQBRRSI

注 : すべてのサイドデッキには、 CSQASTUB を含めることによってすでに解決された、データ変換エントリー・ポイント MQXCNVC の定義が含まれます。

共通の問題:

- アプリケーションが非同期メッセージのコンシューム (MQCB、MQCTL、または MQSUB 呼び出し) を使用し、前述の DLL インターフェースを使用しない場合、次のメッセージがジョブ・ログに出力されます。

  ```
  CSQB001E z/OS バッチまたは z/OS UNIX System Services で実行する Language Environment プログラムは、IBM MQ に対して DLL インターフェースを使用する必要があります
  ```

  解決方法: 前述のように、スタブではなくサイドデッキを使用して、アプリケーションを再構築してください。
- プログラムのビルド時に、次のメッセージが出力されます。

  ```
  IEW2469E セクション ユーザー・コード から MQAPI-名前 への参照の属性が、以下の属性と一致しません。
  ターゲット・シンボル
  ```

  理由: これは、XPLINK プログラムを、V701 (以降) のバージョンの cmqc.h でコンパイルしたが、サイドデッキを使用してバインドしていないことを意味します。

  解決方法: SCSQLOAD からのスタブではなく、SCSQDEFS からの適切なサイドデッキに対してバインドするように、 プログラムのビルド・ファイルを変更してください。

次のサンプル JCL では、31 ビット Language Environment DLL 呼び出しインターフェースを使用できるように C プログラムをコンパイルおよびリンク・エディットする方法を示します。

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
```

```
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

注 : コンパイルでは **DLL** オプションを使用します。 リンク・エディットでは **DYNAM=DLL** オプションを使用し、**CSQBMQ1** ライブラリーを参照します。

次のサンプル JCL では、31 ビット XPLINK DLL 呼び出しインターフェースを使用できるように C プログラムをコンパイルおよびリンク・エディットする方法を示します。

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

注 : コンパイルでは **XPLINK** および **DLL** のオプションを使用します。 リンク・エディットでは **DYNAM=DLL** オプションを使用し、**CSQBMQ1X** ライブラリーを参照します。

コンパイル・オプション DLL がモジュール内の各プログラムに追加されていることを確認します。 IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED などのメッセージは、すべてのプログラムが DLL オプションを使用してコンパイルされていることを確認するのに必要な指示です。

z/OS *z/OS での CICS アプリケーションの構築*

z/OS で CICS アプリケーションを構築する際には、この情報を使用してください。

CICS の下で実行される IBM MQ for z/OS 用のアプリケーションをビルドするには、以下を行う必要があります。

- プログラム内の CICS コマンドを、プログラムの残りの部分の作成に使用する言語に変換する。
- 変換プログラムからの出力をコンパイルまたはアセンブルして、オブジェクト・コードを作成する。
  - PL/I プログラムの場合は、コンパイラー・オプション EXTRN(SHORT) を使用する。
  - C アプリケーションの場合、アプリケーションが XPLINK を使用していない場合は、コンパイラー・オプション DEFINE (MQ_OS_LINKAGE=1) を使用します。
- オブジェクト・コードをリンク・エディットして、ロード・モジュールを作成する。

CICS では、これらのステップを順に実行するプロシージャーを、サポートしている各プログラム言語ごとに提供しています。

- CICS Transaction Server for z/OS については、「*CICS Transaction Server for z/OS System Definition Guide*」にこれらのプロシージャーの使用方法が説明されており、「*CICS/ESA Application Programming Guide*」に変換プロセスに関する詳細な説明があります。

次のものを組み込む必要があります。

- コンパイル (またはアセンブリー) 段階の SYSLIB ステートメントに、製品データ定義ファイルをコンパイラーで利用できるようにするステートメントを組み込む。 データ定義は、以下の IBM MQ for z/OS ライブラリーで提供されます。
  - COBOL の場合、**thlqual**.SCSQCOBC
  - アセンブラー言語の場合、**thlqual**.SCSQMACS
  - C の場合、**thlqual**.SCSQC370
  - PL/I の場合、**thlqual**.SCSQPLIC
- リンク・エディット JCL に、IBM MQ for z/OS CICS スタブ・プログラム (CSQCSTUB) を組み込む。 1033 ページの図 115 に、これを行うための JCL コードの一部を示します。 スタブは使用する言語に依存しないもので、ライブラリー **thlqual**.SCSQLOAD で提供されます。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

図 *115. CICS* 環境でオブジェクト・モジュールをリンク・エディットするための *JCL* の一部

- CICS T3.2 以降の CICS バージョンの場合、または IBM MQ メッセージ・プロパティー API、IBM MQ API MQCB、MQCTL、MQSTAT、MQSUB、または MQSUBR を使用する場合、CSQCSTUB を提供された IBM MQ ではなく、DFHMQSTB というスタブを提供された CICS でオブジェクト・コードをリンク・エディットする必要があります。 CICS 用の IBM MQ プログラムの作成について詳しくは、 CICS 製品資料の IBM MQ MQI 呼び出しにアクセスするための API スタブ・プログラム を参照してください。

これらのステップが完了したら、アプリケーション・ロード・ライブラリーにロード・モジュールを格納し、プログラムを通常の方法で CICS に定義します。

CICS プログラムを実行する前に、システム管理者はそれを IBM MQ プログラムおよびトランザクションとしてプログラムを CICS に定義する必要があります。その後、標準的な方法でそれを実行することができます。

z/OS *IMS (BMP または MPP) アプリケーションの構築*

IMS (BMP または MPP) アプリケーションの構築について説明します。

DL/I バッチ・プログラムを作成する場合は、1029 ページの『z/OS バッチ・アプリケーションの構築』を参照してください。 IMS で (BMP または MPP として) 実行する他のアプリケーションを構築するには、以下の作業を行う JCL を作成してください。

1. プログラムをコンパイル (またはアセンブル) して、オブジェクト・コードを作成する。 コンパイル用の JCL には、SYSLIB ステートメントが含まれていなければなりません。この SYSLIB ステートメントによって製品データ定義ファイルがコンパイラーで利用できるようになります。 データ定義は、以下の IBM MQ for z/OS ライブラリーで提供されます。
   - COBOL の場合、**thlqual**.SCSQCOBC
   - アセンブラー言語の場合、**thlqual**.SCSQMACS
   - C の場合、**thlqual**.SCSQC370
   - PL/I の場合、**thlqual**.SCSQPLIC

2. C アプリケーションの場合は、ステップ 1033 ページの『1』 で作成したオブジェクト・モジュールを事前にリンクする。
3. PL/I プログラムの場合は、コンパイラー・オプション EXTRN(SHORT) を使用する。
4. C アプリケーションの場合、アプリケーションが XPLINK を使用していない場合は、コンパイラー・オプション DEFINE (MQ_OS_LINKAGE=1) を使用します。.
5. ステップ 1033 ページの『1』 (または C/370 アプリケーションの場合はステップ 1034 ページの『2』) で作成したオブジェクト・コードをリンク・エディットして、ロード・モジュールを作成します。
   a. IMS 言語インターフェース・モジュール (DFSLI000) を組み込む。
   b. IBM MQ for z/OS の IMS スタブ・プログラム (CSQQSTUB) を組み込む。 1034 ページの図 116 に、これを行うための JCL の一部を示します。 スタブは使用する言語に依存しないもので、ライブラリー **thlqual**.SCSQLOAD で提供されます。

      注 : COBOL を使用している場合は、NODYNAM コンパイラー・オプションを選択して、リンク・エディット・プログラムが CSQQSTUB の参照を解決できるようにしてください。ただし、1035 ページの『IBM MQ スタブの動的呼び出し』 で説明するダイナミック・リンクを使用する場合を除きます。
6. ロード・モジュールをアプリケーション・ロード・ライブラリーに格納する。

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

図 *116. IMS* 環境でオブジェクト・モジュールをリンク・エディットするための *JCL* の一部

IMS プログラムを実行する前に、システム管理者はそれを IBM MQ プログラムおよびトランザクションとして IMS に定義する必要があります。それから、標準的な方法でそれを実行することができます。

z/OS *z/OS UNIX System Services* アプリケーションの構築

z/OS UNIX System Services アプリケーションの構築について説明します。

z/OS UNIX System Services の下で実行される IBM MQ for z/OS 用の C アプリケーションをビルドするには、以下のようにアプリケーションをコンパイルしてリンクします。

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

thlqual はインストール先で使用する高水準の修飾子です。

C プログラムを実行するには、`.profile` ファイル (ルート・ディレクトリーに存在している必要がある) に、以下のように追加する必要があります。

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

変更が認識されるには、z/OS UNIX System Services を出て、再度 z/OS UNIX System Services に入る必要があることに注意してください。

複数のシェルを実行する場合は、以下のように export という語を行の先頭に追加します。

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

これが正常に完了すると、CSQBSTUB にリンクして、IBM MQ 呼び出しを発行できます。

1035 ページの『IBM MQ スタブの動的呼び出し』では、IBM MQ スタブをリンク・エディットする代わりにプログラムで MQI 呼び出しを行う方法について説明します。 この方法は、すべての言語および環境で使用できるわけではありません。

プログラムが動作しているバージョンの IBM MQ for z/OS のスタブ・プログラムよりも高いレベルのスタブ・プログラムをリンク・エディットしないでください。 例えば、IBM WebSphere MQ for z/OS 7.1 で動作しているプログラムを IBM MQ for z/OS 8.0 で提供されるスタブ・プログラムとリンク・エディットしてはなりません。

z/OS *IBM MQ スタブの動的呼び出し*

IBM MQ スタブ・プログラムをオブジェクト・コードとリンク・エディットする代わりに、プログラム内からスタブを動的に呼び出すことができます。

これは、バッチ、IMS、CICS の各環境で行うことができます。 この機能は、RRS 環境ではサポートされていません。 アプリケーション・プログラムで RRS を使用して更新を整合させる場合は、1039 ページの『RRS の考慮事項』を参照してください。

しかし、この動的呼び出しは、以下の問題を伴います。

- プログラムがより複雑になる。
- プログラムの実行時により多くのストレージが必要になる。
- プログラムのパフォーマンスが低下する。
- 他の環境で、同じプログラムを使用できなくなる。

スタブを動的に呼び出す場合には、実行時に適切なスタブ・プログラムとその別名がなければなりません。それには、IBM MQ for z/OS のデータ・セット SCSQLOAD を以下のものに組み込みます。

- バッチおよび IMS の場合、JCL の STEPLIB 連結。
- CICS の場合、CICS DFHRPL 連結。

IMS の場合、動的スタブを含むライブラリー (IMS アダプターのセットアップ の IMS アダプターのインストールに関する情報で説明されているように作成されている) であることを確認します。 領域 JCL の STEPLIB 連結内のデータ・セット SCSQLOAD の前にあります。

スタブを動的に呼び出す場合には、1035 ページの表 152 に示す名前を使用します。 PL/I では、プログラムで使用している呼び出し名のみを宣言します。

表 *152.* ダイナミック・リンクのための呼び出し名

MQI 呼び出し	**バッチ (非 RRS) 動的呼び出し名**	**CICS 動的呼び出し名**	**IMS 動的呼び出し名**
MQBACK	CSQBBACK	サポートされていない	サポート対象外
MQBUFMH	CSQBBFMH	CSQCBFMH[1]	MQBUFMH
MQCB	CSQBCB	CSQCCB[1]	サポート対象外
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	サポートされていない	サポート対象外
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH[1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL[1]	サポート対象外
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH[1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP[1]	MQDLTMP

表 152. ダイナミック・リンクのための呼び出し名 (続き)			
MQI 呼び出し	**バッチ (非 RRS) 動的呼び出し名**	**CICS 動的呼び出し名**	**IMS 動的呼び出し名**
MQGet	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP[1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF[1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP[1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT[1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB[1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR[1]	MQSUBRQ

注 : 1. これらの API 呼び出しは、CICS TS 3.2 以降の使用時にのみ使用可能であり、CICS に付属する CSQCSTUB を使用する必要があります。 CICS TS 3.2 の場合、APAR PK66866 が適用されている必要があります。 CICS TS 4.1 の場合、APAR PK89844 が適用されている必要があります。

この技法の使用方法の例については、次の図を参照してください。

- バッチと COBOL: 1036 ページの図 117 を参照してください
- CICS と COBOL: 1037 ページの図 118 を参照してください
- IMS と COBOL: 1037 ページの図 119 を参照してください
- バッチとアセンブラー: 1037 ページの図 120 を参照してください
- CICS とアセンブラー: 1037 ページの図 121 を参照してください
- IMS とアセンブラー: 1038 ページの図 122 を参照してください
- バッチと C: 1038 ページの図 123
- CICS と C: 1038 ページの図 124 を参照してください
- IMS と C: 1038 ページの図 125 を参照してください
- バッチと PL/I: 1038 ページの図 126 を参照してください
- IMS と PL/I: 1039 ページの図 127 を参照してください

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQBOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

図 *117.* バッチ環境で *COBOL* を使用したダイナミック・リンク

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

図 *118. CICS* 環境で *COBOL* を使用したダイナミック・リンク

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

図 *119. IMS* 環境で *COBOL* を使用したダイナミック・リンク

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

図 *120.* バッチ環境でのアセンブリー言語を使用した動的リンク

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

図 *121. CICS* 環境でのアセンブリー言語を使用した動的リンク

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

図 *122. IMS* 環境でのアセンブリー言語を使用した動的リンク

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

図 *123.* バッチ環境で *C* 言語を使用したダイナミック・リンク

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

図 *124. CICS* 環境で *C* 言語を使用したダイナミック・リンク

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

図 *125. IMS* 環境で *C* 言語を使用したダイナミック・リンク

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

図 *126.* バッチ環境で *PL/I* を使用したダイナミック・リンク

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

図 *127. IMS* 環境で *PL/I* を使用したダイナミック・リンク

z/OS *RRS* の考慮事項

アプリケーション・プログラムで RRS を使用して更新を調整する場合に、この情報を考慮してください。

IBM MQ では、RRS 調整を必要とするバッチ・プログラム用のスタブが 2 種類提供されています。896 ページの『RRS バッチ・アダプター』を参照してください。 後続の API 呼び出しの動作の違いは、MQCONN または MQCONNX API 上のスタブ・ルーチンによって渡される情報から、バッチ・アダプターによって MQCONN 時に決まります。 つまり、適切なスタブを使用して IBM MQ への初期接続が行われた場合は、RRS 調整を必要とするバッチ・プログラムのために動的 API 呼び出しを使用できることを意味します。 この点を以下の例に示します。

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS プログラムのデバッグ

この情報を使用して、TSO および CICS プログラムのデバッグについて、および CICS トレースの詳細について学習します。

IBM MQ for z/OS アプリケーション・プログラムのデバッグを容易にするための主な手段として、各 API 呼び出しによって戻される理由コードがあります。 これらのコードのリストと、修正アクションのアイデアについては、以下を参照してください。

- IBM MQ for z/OS のメッセージ、完了コード、および理由コードの IBM MQ for z/OS
- メッセージおよび理由コード (その他のすべての IBM MQ プラットフォームの場合)

このトピックでは、特定の環境で使用するその他のデバッグ・ツールも示します。

TSO プログラムのデバッグ

次の対話式デバッグ・ツールは、TSO プログラムで利用できます。

- TEST ツール
- VS COBOL II 対話式デバッグ・ツール
- C および PL/I プログラム用 INSPECT 対話式デバッグ・ツール

CICS プログラムのデバッグ

CICS 実行診断機能 (CEDF) を使用すると、プログラムまたはプログラム作成プロシージャーを変更しなくても、CICS プログラムを対話形式でテストすることができます。

EDF の詳細については、「*CICS Transaction Server for z/OS CICS Application Programming Guide*」を参照してください。

CICS トレース

CICS トレースの活動を制御するには、CICS トレース管理トランザクション (CETR) を使用すると簡単に行うことができます。

CETR の詳細については、「*CICS Transaction Server for z/OS CICS-Supplied Transactions*」という資料を参照してください。

CICS トレースが活動状態にあるかどうかを判別するには、CKQC パネルを使用して接続状況を表示します。このパネルにはトレース番号も表示されます。

CICS トレース・エントリーの解釈については、1040 ページの表 153 を参照してください。

これらの値の CICS トレース・エントリーは、AP0 *xxx* です (*xxx* は、CICS アダプターが使用可能になったときに指定されたトレース番号です)。 CSQCTEST を除くすべてのトレース・エントリーは、CSQCTRUE によって発行されます。 CSQCTEST は、CSQCRST および CSQCDSP によって発行されます。

表 *153. CICS* アダプターのトレース・エントリー

名前	説明	トレース順序	トレース・データ
CSQCABNT	異常終了	END_THREAD ABNORMAL を IBM MQ に発行する前。 これは作業の終了によるものであるため、暗黙のバックアウトはアプリケーションによって行うことができます。 その場合は、END_THREAD 呼び出しに ROLLBACK 要求が含まれています。	作業単位情報 作業の状況を知りたいときは、この情報を使用できます。 (例えば、DISPLAY THREAD コマンドまたは IBM MQ for z/OS ログ印刷ユーティリティーによって生成された出力と、照合することができます。)
CSQCBACK	同期点のバックアウト	BACKOUT を IBM MQ for z/OS に出す前。 これは、アプリケーションからの明示的バックアウト要求によります。	作業単位情報
CSQCCCRC	完了コードと理由コード	API 呼び出しからの異常戻り後	完了コードと理由コード
CSQCCOMM	同期点のコミット	COMMIT を IBM MQ for z/OS に出す前。 これは、単一フェーズ・コミット要求、または 2 フェーズ・コミット要求の第 2 フェーズによるものです。 この要求は、アプリケーションからの明示的な同期点要求によるものです。	作業単位情報

表 153. CICS アダプターのトレース・エントリー (続き)

名前	説明	トレース順序	トレース・データ
CSQCEXER	解決の実行	EXECUTE_RESOLVE を IBM MQ for z/OS に出す前。	EXECUTE_RESOLVE を発行する作業単位の作業単位情報。 これは、再同期プロセスにおける最後の未確定作業単位です。
CSQCGETW	読み取り待機	CICS 待機を発行する前	待機する対象の ECB のアドレス
CSQCGMGD	読み取りメッセージ・データ	MQGET からの正常な戻り後	最大 40 バイトまでのメッセージ・データ
CSQCGMGH	読み取りメッセージ・ハンドル	MQGET を IBM MQ for z/OS に発行する前。	オブジェクト・ハンドル
CSQCGMGI	読み取りメッセージ ID	MQGET からの正常な戻り後	メッセージのメッセージ ID と相関 ID
CSQCINDL	未確定リスト	2 番目の INQUIRE_INDOUBT からの正常な戻り後	未確定作業単位リスト
CSQCINDO	IBM の使用専用		
CSQCINDS	未確定リスト・サイズ	最初の INQUIRE_INDOUBT からの正常な戻り後、かつ未確定リストが空でない。	リストの長さ。 64 で割ると未確定の作業単位の数が得られます。
CSQCINQH	INQ ハンドル	MQINQ を IBM MQ for z/OS に発行する前。	オブジェクト・ハンドル
CSQCLOSH	CLOSE ハンドル	MQCLOSE を IBM MQ for z/OS に発行する前。	オブジェクト・ハンドル
CSQCLOST	後処理情報なし	再同期プロセス時に、CICS は、再始動されたために再同期される作業単位に関する後処理情報がないことをアダプターに通知する。	再同期される作業単位の CICS に認識されている作業単位 ID
CSQCNIND	後処理が未確定でない	再同期プロセス時に、CICS は、再同期される作業単位が未確定になっていない (つまり、依然として実行中であるものと思われる) ことをアダプターに通知する。	再同期される作業単位の CICS に認識されている作業単位 ID
CSQCNORT	正常終了	END_THREAD NORMAL を IBM MQ for z/OS に出す前。 これは作業の終了によるものであるため、アプリケーションが暗黙の同期点のコミットを行う場合があります。 COMMIT 要求は、この場合、END_THREAD 呼び出しに組み込まれます。	作業単位情報
CSQCOPNH	OPEN ハンドル	MQOPEN からの正常な戻り後	オブジェクト・ハンドル
CSQCOPNO	OPEN オブジェクト	MQOPEN を IBM MQ for z/OS に発行する前。	オブジェクト名
CSQCPMGD	書き込みメッセージ・データ	MQPUT を IBM MQ for z/OS に発行する前。	最大 40 バイトまでのメッセージ・データ

表 153. CICS アダプターのトレース・エントリー (続き)

名前	説明	トレース順序	トレース・データ
CSQCPMGH	書き込みメッセージ・ハンドル	MQPUT を IBM MQ for z/OS に発行する前。	オブジェクト・ハンドル
CSQCPMGI	書き込みメッセージ ID	IBM MQ for z/OS からの正常な MQPUT の後。	メッセージのメッセージ ID と相関 ID
CSQCPREP	同期点の作成	2 フェーズ・コミット処理の最初のフェーズにおいて PREPARE を IBM MQ for z/OS に発行する前。この呼び出しは、分散キューイング機能要素から API 呼び出しとして出されることもあります。	作業単位情報
CSQCP1MD	単一書き込みメッセージ・データ	MQPUT1 を IBM MQ for z/OS に発行する前。	最大 40 バイトのデータからなるメッセージ
CSQCP1MI	単一書き込みメッセージ ID	MQPUT1 からの正常な戻り後	メッセージのメッセージ ID と相関 ID
CSQCP1ON	単一書き込みオブジェクト名	MQPUT1 を IBM MQ for z/OS に発行する前。	オブジェクト名
CSQCRBAK	解決済みバックアウト	RESOLVE_ROLLBACK を IBM MQ for z/OS に出す前。	作業単位情報
CSQCRCMT	解決済みコミット	RESOLVE_COMMIT を IBM MQ for z/OS に出す前。	作業単位情報
CSQCRMIR	RMI 応答	特定の呼び出しから CICS RMI (リソース管理プログラム・インターフェース) に戻る前	体系化された RMI 応答値。 その意味は、呼び出しの種類によって異なります。 これらの値については、「*CICS Transaction Server for z/OS Customization Guide*」に記述されています。 呼び出しの種類を突き止めるには、CICS RMI コンポーネントが作成した先行のトレース項目を見てください。
CSQCRSYN	再同期	作業に対して再同期プロセスが開始する前	再同期される作業単位の CICS に認識されている作業単位 ID
CSQCSETH	SET ハンドル	MQSET を IBM MQ for z/OS に発行する前。	オブジェクト・ハンドル
CSQCTASE	IBM の使用専用		
CSQCTEST	トレース・テスト	ユーザーが指定したトレース番号または接続のトレース状況を検査するために EXEC CICS ENTER TRACE 呼び出しで使用される。	データなし
CSQCDCFF	IBM の使用専用		

プロシージャー型プログラム・エラーの処理

ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

キュー・マネージャーは、可能なときはいつでも、MQI 呼び出しが行われるとすぐにどのようなエラーでも戻します。 これらはローカルで判別されたエラー です。

メッセージをリモート・キューに送信するときは、MQI 呼び出しが行われた時にエラーが明らかにならないことがあります。 この場合は、エラーを識別したキュー・マネージャーが、別のメッセージを発信元のプログラムに送ってエラーを報告します。 これらはリモートで判別されたエラー です。

ローカルで判別されたエラー

MQI 呼び出しの失敗、システムの中断、および間違ったデータを含むメッセージといった、ローカルで判別されるエラーについての情報を示します。

キュー・マネージャーが即時に報告できるエラーで、最も一般的な原因として次の 3 つが挙げられます。

- MQI 呼び出しが失敗した (例えば、キューが満ぱいのために)。
- 実行中のアプリケーションがキュー・マネージャーなどのシステム部分に依存している状態で、このようなシステム部分の運用の中断が発生した。
- メッセージに、正しく処理できないデータが含まれている。

非同期書き込み機能を使用している場合、エラーは即時に 報告されません。 MQSTAT 呼び出しを使用して、前の非同期書き込み操作に関する状況情報を取得 してください。

MQI 呼び出しの失敗

キュー・マネージャーは MQI 呼び出しのどのようなコーディング・エラーでもただちに報告することができます。 そのときは、事前定義された戻りコードのセットが使用されます。 戻りコードは、完了コードと理由コードに分けられます。

呼び出しが成功したかどうかを示すために、キュー・マネージャーは呼び出しが完了したときに完了コード を戻します。 完了コードには、成功、部分完了、および呼び出しの失敗を示す 3 種類のコードがあります。 また、キュー・マネージャーは、部分完了または呼び出しの失敗の理由を示す理由コード も戻します。

各呼び出しに対する完了コードおよび理由コードは、戻りコードにその呼び出しの説明と共に記載されています。 修正処置のアイデアを含む詳細については、以下を参照してください。

- z/OS IBM MQ for z/OS のメッセージ、完了コード、および理由コードの IBM MQ for z/OS
- メッセージおよび理由コード (その他のすべての IBM MQ プラットフォームの場合)

各呼び出しから発生する可能性のあるすべての戻りコードを処理できるように、プログラムを設計してください。

System i の中断

キュー・マネージャーがシステム障害から回復しなければならない場合に、それに接続するアプリケーションが割り込みを検知しないことがあります。 しかし、このような割り込みが起きてもユーザーのデータが失われないようにアプリケーションを設計しなければなりません。

データの一貫性を維持するために使用できる方法は、キュー・マネージャーが稼働しているプラットフォームによって次のように異なります。

z/OS

CICS および IMS 環境では、CICS または IMS によって管理される作業単位内で、MQPUT および MQGET 呼び出しを行うことができます。 バッチ環境では、MQPUT および MQGET 呼び出しを同じ方法で行うことができますが、以下を使用して同期点を宣言する必要があります。

- IBM MQ for z/OS の MQCMIT および MQBACK 呼び出し (857 ページの『作業単位のコミットとバックアウト』 を参照)、または
- z/OS トランザクション管理とリカバリー可能リソース管理サービス (RRS)。これは 2 フェーズ同期点サポートを提供します。 RRS を使用すると、単一の作業論理単位内で、IBM MQ 製品リソースと RRS で使用可能な他の製品リソース (Db2 ストアード・プロシージャー・リソースなど) を、どちらも

更新することができます。 RRS 同期点サポートについては、862 ページの『トランザクション管理とリカバリー可能リソース管理サービス』を参照してください。

IBM i
IBM i コミットメント制御によって管理されるグローバルな作業単位内で、MQPUT および MQGET 呼び出しを行うことができます。 同期点を宣言するには、ネイティブの IBM i COMMIT および ROLLBACK コマンドを使用するか、または言語固有のコマンドを使用します。 ローカル作業単位は、MQCMIT および MQBACK 呼び出しを使用して IBM MQ によって管理されます。

AIX, Linux, and Windows システム
これらの環境では、MQPUT および MQGET 呼び出しを通常の方法で行うことができますが、MQCMIT および MQBACK 呼び出しを使用して同期点を宣言する必要があります (857 ページの『作業単位のコミットとバックアウト』を参照してください)。 CICS 環境では、CICS が管理する作業単位内で MQPUT および MQGET 呼び出しを行えるので、MQCMIT および MQBACK コマンドは使用不可になります。

失いたくないデータをすべて送達するには、持続メッセージを使用してください。 キュー・マネージャーが障害から回復する必要がある場合は持続メッセージはキューに再び戻されます。 ALW IBM MQ on AIX, Linux, and Windows の場合、アプリケーション内の MQGET または MQPUT 呼び出しは、すべてのログ・ファイルがいっぱいになる時点で失敗し、メッセージ MQRC_RESOURCE_PROBLEM が出されます。 AIX, Linux, and Windows でのログ・ファイルについて詳しくは、IBM MQ の管理を参照してください。
z/OS z/OS の場合は、z/OS での計画を参照してください。

アプリケーションの実行中に、オペレーターがキュー・マネージャーを停止させる場合は、通常、休止オプションが使用されます。 キュー・マネージャーは静止状態となり、アプリケーションはその状態で作業を続行できますが、できるだけ早くすぐに終了させなければなりません。 小規模で高速のアプリケーションは、多くの場合、静止状態を無視して正常に終了するまで続行できます。 実行時間の長いアプリケーションや、メッセージの到着を待機しているアプリケーションでは、MQOPEN、MQPUT、MQPUT1、および MQGET 呼び出しを使用する場合には、静止状態のときは失敗 オプションを設定することが必要です。 これらのオプションは、キュー・マネージャーが静止しているときは呼び出しが失敗することを意味します。 ただし、アプリケーションには、静止状態を無視する呼び出しを出して正しく終了するだけの時間がある場合もあります。 このようなアプリケーションは、それ自体が行った変更をコミットまたはバックアウトしてから終了することもできます。

キュー・マネージャーが強制停止 (つまり、静止状態のない停止) された場合は、アプリケーションが MQI 呼び出しを 行うと MQRC_CONNECTION_BROKEN という理由コードが出されます。 アプリケーションを終了するか、 IBM i IBM MQ for IBM i、AIX, Linux, and Windows システムの場合は、代わりに MQDISC 呼び出しを発行します。

誤りデータを含むメッセージ

使用中のアプリケーションで作業単位が使用されている場合に、プログラムがキューから取り出したメッセージを正常に処理できないときは、MQGET 呼び出しがバックアウトされます。

キュー・マネージャーはそれが起こった回数をカウントして、メッセージ記述子の *BackoutCount* フィールドに保持します。 このカウントは影響を受ける各メッセージの記述子に保持されます。 このカウントによりアプリケーションの効率に関する貴重な情報が得られます。 バックアウト・カウントが時間の経過に伴って増加するメッセージは、繰り返し拒否されます。 この現象が生じる理由を分析し、それに応じてそのようなメッセージを処理できるよう、アプリケーションを設計してください。

z/OS IBM MQ for z/OS で、バックアウト・カウントをキュー・マネージャーの再始動時に残せるように、**HardenGetBackout** 属性を MQQA_BACKOUT_HARDENED に設定してください。これを無視すると、キュー・マネージャーが再始動しなければならない場合に、各メッセージの正確なバックアウト・カウントが維持されません。 属性をこのように設定すると、余分な処理が付け加えられます。

IBM MQ (IBM i IBM i、AIX, Linux, and Windows システム用) では、バックアウト・カウントは常にキュー・マネージャーの再始動時に保持されます。

z/OS また、IBM MQ for z/OS では、作業単位内でキューからメッセージを除去したときに、1 つのメッセージにマークを付けて、その作業単位がアプリケーションによってバックアウトされる場合に、そのメッセージが再び使用可能にならないようにすることができます。 マークの付いたメッセージは、新し

い作業単位で取り出された場合と同様に扱われます。 MQGMO_MARK_SKIP_BACKOUT オプションを使用して、バックアウトをスキップするメッセージにマークを付けます。(MQGMO 構造で) MQGET 呼び出しを使用する場合。 この手法については、799 ページの『バックアウトのスキップ』を参照してください。

問題判別用の報告メッセージの使用

リモート・キュー・マネージャーは、MQI 呼び出しを行ったときに、エラー (メッセージをキューに入れられない場合など) を報告することはできませんが、メッセージをどのように処理したかを知らせる報告メッセージを出力できます。

アプリケーション内では報告メッセージを作成 (MQPUT) することができ、オプションを選択してそれらを受信することもできます (この場合、別のアプリケーションかキュー・マネージャーによって送信されます)。

報告メッセージの作成

報告メッセージによって、アプリケーションは送られたメッセージを処理できないということを別のアプリケーションに通知できます。

ただし、*Report* フィールドを最初に分析して、メッセージを送信したアプリケーションが問題通知の対象となるかどうかを判断する必要があります。 報告メッセージが必要と判断した場合は、さらに次の事項を決定する必要があります。

- 元のメッセージをすべて含めるか、先頭の 100 バイトのデータだけを含めるか、それとも元のメッセージを一切含めないか。
- 元のメッセージをどのように処理するか。 廃棄することも、送達不能キューに入れることもできます。
- *MsgId* フィールドおよび *CorrelId* フィールドの内容は、同様に必要か。

作成される報告メッセージの理由を示すには、*Feedback* フィールドを使用してください。 報告メッセージは、アプリケーションの応答先キューに書き込みます。 詳細については、Feedback を参照してください。

報告メッセージの要求および受信 (MQGET)

メッセージを別のアプリケーションに送信するときは、*Report* フィールドを設定して必要なフィードバックを指示しない限り、問題が通知されることはありません。 使用可能なオプションについては、レポート・フィールドの構造を参照してください。

キュー・マネージャーは、報告メッセージをアプリケーションの応答先キューに必ず書き込みますが、使用する独自のアプリケーションにより同じ処理を実行することをお勧めします。 報告メッセージ機能を使用するときは、応答先キューの名前をメッセージのメッセージ記述子に指定してください。そうしないと、MQPUT 呼び出しは失敗します。

アプリケーションには、応答先キューをモニターし、そこに到着するメッセージを処理する手順が含まれている必要があります。 報告メッセージには、元のメッセージが全部または先頭の 100 バイトだけ入っているか、または元のメッセージはまったく入っていないことを念頭においてください。

キュー・マネージャーは、報告メッセージの *Feedback* フィールドを設定することにより、宛先キューが存在しないなどのエラーの原因を示します。 使用しているプログラムも同様の処理を行います。

報告メッセージの詳細については、20 ページの『レポート・メッセージ』を参照してください。

リモートで判別されたエラー

メッセージをリモート・キューに送信すると、ローカル・キュー・マネージャーが MQI 呼び出しを処理した際にエラーを検出しなかった場合でも、リモート・キュー・マネージャーによるメッセージの処理内容は、別の要因によって左右される可能性があります。

例えば、宛先に指定しているキューが満ぱいであったり、存在していない場合もあります。 メッセージが、宛先キューへの経路上にある他の中間キュー・マネージャーによって処理されなければならない場合は、これらのプログラムはエラーを検出する可能性があります。

メッセージ送達時の問題

MQPUT 呼び出しが失敗した場合、キューにメッセージを再度書き込む、メッセージを送信側に戻す、またはメッセージを送達不能キューに書き込むかのいずれかを行うことができます。

各オプションにはそれぞれの利点がありますが、宛先キューが満杯であったことが原因で MQPUT が失敗した場合には、必ずしもメッセージの書き込みを再試行する必要はありません。 この場合は、メッセージを送達不能キューに書き込むことにより、あとで正しい宛先キューにメッセージを送達できます。

メッセージ送達の再試行

MsgRetryCount および *MsgRetryInterval* という属性がそのチャネルに設定されていた場合、またはチャネルが使用する再試行出口プログラム (チャネルの属性である *MsgRetryExitId* フィールドに名前が保管されているプログラム) が存在する場合、リモート・キュー・マネージャーは、メッセージが送達不能キューに書き込まれる前に、キューにメッセージを再度書き込もうとします。

MsgRetryExitId フィールドがブランクの場合、*MsgRetryCount* および *MsgRetryInterval* という属性の値が使用されます。

MsgRetryExitId フィールドがブランクではない場合、この名前の出口プログラムが実行されます。 ユーザー独自の出口プログラムの使用について詳しくは、967 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

送信側への戻りメッセージ

送信側にメッセージを戻すには、元のメッセージをすべて取り込むように、生成される報告メッセージを要求します。

報告メッセージのオプションの詳細については、20 ページの『レポート・メッセージ』を参照してください。

送達不能 (未配布メッセージ) キューの使用

キュー・マネージャーは、メッセージを送達できないと、メッセージをその送達不能キューに書き込もうとします。 このキューは、キュー・マネージャーのインストール時に定義する必要があります。

ご使用のプログラムも送達不能キューを使用できますが、この使用方法はキュー・マネージャーによる使用方法と同様です。 送達不能キューの名前は、(MQOPEN 呼び出しを使用して) キュー・マネージャーのオブジェクトをオープンし、(MQINQ 呼び出しを使用して) **DeadLetterQName** 属性について問い合わせることによって確認できます。

キュー・マネージャーは、メッセージをこのキューに書き込む際に、メッセージにヘッダーを追加します。そのフォーマットは送達不能ヘッダー (MQDLH) 構造体で説明されます。 MQDLH - 送達不能ヘッダーを参照してください。 このヘッダーには、宛先キューの名前およびメッセージが送達不能キューに書き込まれた理由が入っています。 メッセージを目的のキューに書き込むときは、必ずヘッダーを除去し、問題を解決しておいてください。 さらに、キュー・マネージャーは、メッセージ記述子 (MQMD) の *Format* フィールドを変更することにより、メッセージに MQDLH 構造体が格納されていることを示します。

MQDLH 構造体

送達不能キューに書き込んだすべてのメッセージに、MQDLH 構造体を追加することをお勧めします。ただし、特定の IBM MQ 製品により提供された送達不能キュー・ハンドラーを使用する場合は、メッセージに MQDLH 構造体を追加する必要があります。

メッセージにヘッダーを追加すると、送達不能キューに対してメッセージが長くなりすぎることがあります。そのため、メッセージが、送達不能キューに許容される最大サイズよりも、少なくとも MQ_MSG_HEADER_LENGTH 定数の値だけ短いことを常に確認してください。 キューに書き込み可能なメッセージの最大サイズは、そのキューの **MaxMsgLength** 属性の値により決まります。 送達不能キューの場合、この属性をキュー・マネージャーによる最大許容値に設定してください。 使用中のアプリケーションがメッセージを送達できず、さらにメッセージが長すぎて送達不能キューに書き込むことができない場合は、MQDLH 構造体の説明に記述されている推奨事項に従ってください。

送達不能キューがモニターされ、そのキューに到着するすべてのメッセージが処理されることを確認してください。 送達不能キュー・ハンドラーはバッチ・ユーティリティーとして稼働し、送達不能キュー上の

選択メッセージに関するさまざまな処理を実行するために使用できます。 詳細については、1047 ページの『送達不能キュー処理』を参照してください。

データ変換が必要な場合に、MQGET 呼び出しの MQGMO_CONVERT オプションを使用すると、キュー・マネージャーはヘッダー情報を変換します。 メッセージを書き込むプロセスが MCA の場合、元のメッセージのすべてのテキストがヘッダーのあとに書き込まれます。

送達不能キューに書き込まれるメッセージがこのキューには長すぎる場合、メッセージが切り捨てられる場合があります。 こうした状況を示す一例としては、送達不能キュー上のメッセージがこのキューの **MaxMsgLength** 属性の値と同じ長さである場合が考えられます。

送達不能キュー処理

ここでは、送達不能キュー処理を使用する際の汎用プログラミング・インターフェース情報を示します。

送達不能キューの処理は、ローカル・システムの要件によって異なりますが、仕様を作成する際には次の点を考慮してください。

- メッセージは、その中に送達不能キューのヘッダーがあるので識別できます。これは、MQMD の形式フィールドの値が MQFMT_DEAD_LETTER_HEADER だからです。
- CICS を使用する IBM MQ for z/OS では、MCA がこのメッセージを送達不能キューに書き込む場合、*PutApplType* フィールドは MQAT_CICS になり、 *PutApplName* フィールドは CICS システムの *ApplId* とそれに続く MCA のトランザクション名になります。
- このメッセージが送達不能キューに送られる理由は、送達不能キュー・ヘッダーの *Reason* フィールドに入ります。
- 送達不能キュー・ヘッダーには、宛先キュー名とキュー・マネージャー名の詳細が収められます。
- 送達不能キュー・ヘッダーには、メッセージが宛先キューに書き込まれる前にメッセージ記述子内に復元しなければならないフィールドが収められています。 次のとおりです。
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- メッセージ記述子は、元のアプリケーションによる PUT と同じです。 ただし、上記の 3 つのフィールド (Encoding、CodedCharSetId、および Format) を除きます。

送達不能キュー・アプリケーションでは、次の 1 つ以上の操作を実行する必要があります。

- *Reason* フィールドを調べます。 次の理由により、MCA がメッセージを書き込んだ場合があるからです。
  - メッセージがチャネルの最大メッセージ・サイズより長くなった。

    理由は MQRC_MSG_TOO_BIG_FOR_CHANNEL です。
  - メッセージがその宛先キューに書き込まれなかった。

    理由は、MQPUT 操作により戻る可能性があるすべての MQRC_* 理由コードとなります。
  - ユーザー出口で、このアクションが要求された。

    理由コードは、ユーザー出口により提供されるか、デフォルトの MQRC_SUPPRESSED_BY_EXIT となります。
- 可能な場合は、メッセージをその予定の宛先に転送することを試みる。
- メッセージが送達不能となった理由は明確になったが、すぐに修正できないとき、ある程度の間メッセージを保存してから廃棄する。
- 問題が特定されたら、問題の訂正を管理者に指示する。
- 破壊されているメッセージや処理できないメッセージを廃棄する。

送達不能キューから回復させたメッセージを処理するには、次の 2 つの方法があります。

1. メッセージの宛先がローカル・キューの場合は、次のようにします。
   - アプリケーション・データを抽出するために必要なコード変換を実行する。

- そのデータがローカル関数の場合は、このデータのコード変換を実行する。
- すべてのメッセージ記述子の詳細を復元して、結果として生成されたメッセージをローカル・キューに書き込む。

2. メッセージの宛先がリモート・キューの場合は、メッセージをそのキューに書き込む。

分散キューイング環境での未配布メッセージの処理方法については、メッセージを送達できない場合の処理を参照してください。

マルチキャスト・プログラミング

この情報は、キュー・マネージャーへの接続や例外報告などの、IBM MQ Multicast のプログラミング・タスクについて学習するために使用します。

IBM MQ Multicast は、ユーザーに対しては可能な限り透過的でありながら、既存のアプリケーションとは互換性があるように設計されています。 COMMINFO オブジェクトを定義し、TOPIC オブジェクトの **MCAST** および **COMMINFO** パラメーターを設定すると、マルチキャストを使用するために既存の IBM MQ アプリケーションを大幅に再作成する必要はなくなります。 ただし、考慮すべきいくつかの制約 (詳細は 1048 ページの『マルチキャストと MQI』を参照) やセキュリティー問題 (詳細はマルチキャストのセキュリティーを参照) が存在する場合があります。

マルチキャストと MQI

ここでは、主なメッセージ・キュー・インターフェース (MQI) 概念およびそれらと IBM MQ マルチキャストとの関連について説明します。

マルチキャスト・サブスクリプションは非永続です。物理キューが使用されず、永続サブスクリプションによって作成されるオフライン・メッセージがどこにも保管されないためです。

アプリケーションがマルチキャスト・トピックをサブスクライブすると、あたかもキューのハンドルであるかのように、取り込み (MQGET) に使用できるオブジェクト・ハンドルが返されます。 これは、管理マルチキャスト・サブスクリプション (MQSO_MANAGED で作成されたサブスクリプション) のみがサポートされることを意味します。 つまり、サブスクリプションを作成してキューにあるメッセージを「指す」ことはできません。 そのため、サブスクリプション呼び出しで返されたオブジェクト・ハンドルからメッセージを取り込まなければならないことになります。 クライアントでは、メッセージはクライアントによって取り込まれるまでメッセージ・バッファーに保管されます。詳しくは、クライアント構成ファイルの MessageBuffer スタンザを参照してください。 クライアントがパブリッシュ速度に追いつかない場合は、必要に応じてメッセージが廃棄されます (最も古いメッセージを最初に廃棄)。

アプリケーションがマルチキャストを使用するかどうかは通常、管理上の決定です (トピック・オブジェクトの MCAST 属性を設定することによって指定)。 パブリッシュ・アプリケーションがマルチキャストが使用されないようにする必要がある場合は、MQOO_NO_MULTICAST オプションを使用できます。 同様に、サブスクライブ・アプリケーションは、MQSO_NO_MULTICAST オプションを使用してサブスクライブすることによって、マルチキャストが使用されないようにすることができます。

IBM MQ マルチキャストは、メッセージ・セレクターの使用をサポートします。 セレクターは、アプリケーションが、選択ストリングが表す SQL92 照会を満たすプロパティーが指定されたメッセージにのみ含まれているインタレストを登録するのに使用されます。 メッセージ・セレクターについて詳しくは、31 ページの『Selectors』を参照してください。

次の表は、主な MQI 概念のすべておよびそれらとマルチキャストとの関連をリストしたものです。

表 *154. MQI* の概念と、マルチキャストとの関係

MQI 概念	**マルチキャストの使用を試みたときのアクション**	**理由コード**
ゼロ長メッセージの書き込み	拒否	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
グループ化	拒否	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

表 154. *MQI* の概念と、マルチキャストとの関係 (続き)

MQI 概念	マルチキャストの使用を試みたときのアクション	理由コード
Segmentation	拒否	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
配布リスト	拒否	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	トピック・ハンドルについては拒否。トピックの MQINQ および MQSET はサポートされていません。	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
MQINQ	管理対象ハンドルについては受け入れ。 照会できるのは Current Depth のみです。	• 値が Current Depth の場合、適用できる理由コードはありません。 • 値が Current Depth 以外の値である場合、理由コードは 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR です。
MQSET	すべてのハンドルについて拒否	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
トランザクション (XA または非 XA)	拒否	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
メッセージ参照	拒否	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
メッセージのロック	拒否	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
マークによる参照	拒否	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
コンテキストの受け渡し	拒否	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	拒否。 マルチキャストのみのトピックに MQPUT1 を試行することは無効です。	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
永続サブスクリプション	トピックに「マルチキャストのみ」のマークが付いている場合は拒否。それ以外の場合は非マルチキャスト・サブスクリプションが行われます。	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED

表 *154. MQI* の概念と、マルチキャストとの関係 *(*続き*)*		
MQI 概念	**マルチキャストの使用を試みたときのアクション**	**理由コード**
TopicString > 255	拒否。 トピック・ストリングが 255 文字より大きい場合は、クライアントで拒否されます。	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
非管理サブスクリプション	トピックに「マルチキャストのみ」のマークが付いている場合は拒否。それ以外の場合は非マルチキャスト・サブスクリプションが行われます。	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	拒否	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

以下の項目は、この表の MQI 概念の一部をさらに詳しく説明したものであると同時に、表にない MQI 概念の一部の情報でもあります。

メッセージの持続性
非永続マルチキャスト・サブスクライバーの場合、パブリッシャーからの永続メッセージはリカバリー不能な形で配信されます。

メッセージ切り捨て
メッセージ切り捨てがサポートされます。つまり、アプリケーションは以下の処理を行えることになります。

1. MQGET を発行する。
2. MQRC_TRUNCATED_MSG_FAILED を取得する。
3. より大きいバッファーを割り振る。
4. MQGET を再発行してメッセージを取得する。

サブスクリプション有効期限
サブスクリプション期限はサポートされていません。 期限を設定しようとしても無視されます。

マルチキャストの高可用性

この情報は、IBM MQ Multicast の継続的な対等通信操作を理解するために使用します。IBM MQ は IBM MQ キュー・マネージャーに接続しますが、メッセージはそのキュー・マネージャー内を流れません。

マルチキャスト・トピック・オブジェクトの MQOPEN または MQSUB を実行するには、キュー・マネージャーへの接続を行う必要がありますが、メッセージ自体はキュー・マネージャー内を流れません。 したがって、マルチキャスト・トピック・オブジェクトに対する MQOPEN または MQSUB が完了した後は、キュー・マネージャーへの接続が失われている場合でも、マルチキャスト・メッセージの送信を続行することは可能です。 次の 2 つのモードの操作があります。

キュー・マネージャーに通常の接続を行う
キュー・マネージャーへの接続が存在している間は、マルチキャスト通信は可能です。 接続が失敗した場合は、通常の MQI 規則が適用されます。例えば、マルチキャスト・オブジェクト・ハンドルへの MQPUT は 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN を戻します。

キュー・マネージャーにクライアント接続の再接続を行う
再接続サイクル中にも、マルチキャスト通信は可能です。 これはつまり、キュー・マネージャーへの接続が切断された場合でも、マルチキャスト・メッセージの書き込みとコンシュームは影響を受けないということです。 クライアントはキュー・マネージャーへの再接続を試行し、再接続が失敗した場合には、接続ハンドルは破壊され、マルチキャストの呼び出しを含むすべての MQI 呼び出しは失敗します。 詳細については、自動クライアント再接続を参照してください。

いずれかのアプリケーションが明示的に MQDISC を発行している場合、すべてのマルチキャスト・サブスクリプションとオブジェクト・ハンドルは閉じられます。

マルチキャストによる対等通信操作の続行

クライアント間の対等通信の利点の 1 つは、メッセージがキュー・マネージャー内を流れる必要がないということですしたがって、キュー・マネージャーへの接続が切断された場合でも、メッセージ転送は続行します。 以下の制限は、このモードの継続メッセージ要件に適用されます。

- 継続操作のために、MQCNO_RECONNECT_* オプションの 1 つを使用して接続を行う必要があります。 このプロセスは、通信セッションが切断されても、実際の接続ハンドルは破壊されておらず、代わりに再接続状態になることを意味します。 再接続が失敗する場合には、接続ハンドルは破壊され、それ以降のすべての MQI 呼び出しは実行できなくなります。
- このモードでは、MQPUT、MQGET、MQINQ、および非同期コンシュームのみがサポートされます。 すべての MQOPEN、MQCLOSE、または MQDISC 動詞は、完了するためにキュー・マネージャーへの再接続が必要です。
- キュー・マネージャーへの状況フローは停止します。したがって、キュー・マネージャーについて示される状態は、既に古くなっているかまたは失効している場合があります。 つまり、クライアントがメッセージを送受信するとしても、キュー・マネージャーの状況が分からないという意味です。 詳細については、マルチキャスト・アプリケーションのモニターを参照してください。

マルチキャスト・メッセージング用の MQI でのデータ変換

この情報は、IBM MQ Multicast メッセージングで、データ変換が行われる方法について理解するために役立ちます。

IBM MQ Multicast はコネクションレスの共有プロトコルであるため、各クライアントがデータ変換に関する特定の要求を行うことはできません。 同じマルチキャスト・ストリームにサブスクライブしているクライアントはすべて同じバイナリー・データを受け取ります。したがって、IBM MQ データ変換が必要な場合は、変換は各クライアントでローカルに実行されます。

データはクライアント上で IBM MQ Multicast トラフィック用に変換されます。 **MQGMO_CONVERT** オプションが指定されている場合、データ変換は要求どおりに実行されます。 ユーザー定義のフォーマットでは、クライアントにデータ変換出口がインストールされている必要があります。クライアント・パッケージとサーバー・パッケージに現在入っているライブラリーの詳細については、988 ページの『データ変換出口の作成』を参照してください。

データ変換の管理については、Multicast メッセージングに関するデータ変換を使用可能にするを参照してください。

データ変換の詳細については、データ変換を参照してください。

データ変換出口および `ClientExitPath` の詳細については、クライアント構成ファイルの ClientExitPath スタンザを参照してください。

マルチキャスト例外報告

この情報は、IBM MQ Multicast イベント・ハンドラーと、IBM MQ Multicast 例外の報告について学習するために使用します。

IBM MQ Multicast は、イベント・ハンドラーを呼び出して、標準 IBM MQ イベント・ハンドラー・メカニズムを使用して報告されるマルチキャスト・イベントを報告することで、問題判別を支援します。

個々のマルチキャスト・イベントの結果として、複数の IBM MQ イベントが呼び出される可能性があります。これは同じマルチキャスト送信側または受信側を使用する複数の MQHCONN 接続ハンドルが存在する

ことがあるためです。 ただし、それぞれのマルチキャスト例外では、IBM MQ 接続当たり 1 つのみのイベント・ハンドラーが呼び出されます。

IBM MQ MQCBDO_EVENT_CALL 定数により、アプリケーションは IBM MQ イベントのみを受け取るためにコールバックを登録することができます。さらに、MQCBDO_MC_EVENT_CALL により、アプリケーションはマルチキャスト・イベントのみを受け取るためにコールバックを登録することができます。 両方の定数を使用する場合、両方のタイプのイベントを受け取ります。

マルチキャスト・イベントの要求

IBM MQ Multicast イベントは、cbd.Options フィールド内の MQCBDO_MC_EVENT_CALL 定数を使用します。 以下の例は、マルチキャスト・イベントの要求方法を示しています。

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

MQCBDO_MC_EVENT_CALL オプションを cbd.Options フィールドに指定する場合、接続レベル・イベントではなく IBM MQ Multicast イベントのみがイベント・ハンドラーに送信されます。 どちらのタイプのイベントもイベント・ハンドラーに送信されるように要求するには、アプリケーションにより cbd.Options フィールドに MQCBDO_EVENT_CALL 定数を、MQCBDO_MC_EVENT_CALL 定数と共に指定する必要があります。以下に例を示します。

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

この定数のどちらも使用しない場合、接続レベル・イベントのみがイベント・ハンドラーに送信されます。

Options フィールドの値の詳細については、Options (MQLONG) を参照してください。

マルチキャスト・イベント・フォーマット

IBM MQ Multicast 例外には、コールバック関数の **Buffer** パラメーターで返されるいくつかのサポート情報が含まれます。 **Buffer** ポインターはポインターの配列を指し、MQCBC.DataLength フィールドは配列のサイズをバイト単位で指定します。 配列の最初の要素は、必ずイベントの短いテキスト記述を指します。 イベントのタイプに応じてさらにパラメーターを指定することもできます。 以下の表は、例外をリストしています。

表 *155.* マルチキャスト・イベント・コードの説明

イベント・コード	説明	追加データ
MQMCEV_PACKET_LOSS	リカバリー不能のパケット・ロス	失われたパケットの数
MQMCEV_HEARTBEAT_TIMEOUT	ハートビート制御パケットの長期間の欠落	N/A
MQMCEV_VERSION_CONFLICT	より新しいプロトコル・バージョン・パケットの受信	N/A
MQMCEV_RELIABILITY	送信側および受信側の異なる信頼性モード	N/A
MQMCEV_CLOSED_TRANS	トピック送信は 1 ソースで閉じられる	N/A
MQMCEV_STREAM_ERROR	ストリームでエラーが検出された	N/A

表 155. マルチキャスト・イベント・コードの説明 (続き)

イベント・コード	説明	追加データ
MQMCEV_NEW_SOURCE	新しいソースがトピックの送信を開始する	ソース構造体
MQMCEV_RECEIVE_QUEUE_TRIMMED	時間切れまたはスペース切れが原因で PacketQ から削除されたパケットの数	トリムされたパケットの数
MQMCEV_PACKET_LOSS_NACK_EXPIRE	NACK 満了に起因するリカバリー不能パケット・ロス	失われたパケットの数
MQMCEV_ACK_RETRIES_EXCEEDED	**max_ack_retries** を超過した後に履歴から削除されたパケットの数	削除されたパケットの数
MQMCEV_STREAM_SUSPEND_NACK	このトピックで受け入れられたストリームで NACK が中断状態である	中断ストリーム ID ストリームが中断されるミリ秒単位の時間
MQMCEV_STREAM_RESUME_NACK	ストリームでの中断後に NACK は再開される	ストリーム ID
MQMCEV_STREAM_EXPELLED	このトピックで受け入れられたストリームが追放要求が原因で拒否された	ストリーム ID
MQMCEV_FIRST_MESSAGE	ソースからの最初のメッセージ	メッセージ番号
MQMCEV_LATE_JOIN_FAILURE	遅延結合セッションを開始できなかった	N/A
MQMCEV_MESSAGE_LOSS	リカバリー不能のメッセージ損失	失われたメッセージの数
MQMCEV_SEND_PACKET_FAILURE	マルチキャスト送信側がマルチキャスト・パケットを送信できなかった	N/A
MQMCEV_REPAIR_DELAY	マルチキャスト受信側が未解決の NAK の修復パケットを受け取らなかった	N/A
MQMCEV_MEMORY_ALERT_ON	受信側の受信バッファーが満杯である	バッファー・プール使用率
MQMCEV_MEMORY_ALERT_OFF	受信側の受信バッファーが通常に戻っている	バッファー・プール使用率
MQMCEV_NACK_ALERT_ON	受信側の修復パケット要求率が最高水準点に達した	秒当たりのパケット数での、現在の修復要求率
MQMCEV_NACK_ALERT_OFF	受信側の修復パケット要求率が通常に戻っている	秒当たりのパケット数での、現在の修復要求率
MQMCEV_REPAIR_ALERT_ON	送信側の修復パケット送信率が最高水準点に達した	N/A
MQMCEV_REPAIR_ALERT_OFF	送信側の修復パケット送信率が通常に戻っている	N/A

表 155. マルチキャスト・イベント・コードの説明 (続き)		
イベント・コード	**説明**	**追加データ**
MQMCEV_SHM_DEST_UNUSABLE	送信側トピック宛先により使用される共有メモリー領域が使用不可であることが検出された	N/A
MQMCEV_SHM_PORT_UNUSABLE	受信側インスタンスにより使用される共有メモリー・ポートが使用不可であることが検出された	N/A
MQMCEV_CCT_GETTIME_FAILED	調整クラスター時間からの取得時間が失敗した	N/A
MQMCEV_DEST_INTERFACE_FAILURE	送信側トピック宛先により使用されるネットワーク・インターフェースに障害が発生し、バックアップのネットワーク・インターフェースが使用不可である	
MQMCEV_DEST_INTERFACE_FAILOVER	送信側トピック宛先により使用されるネットワーク・インターフェースに障害が発生し、別のインターフェースへのフェイルオーバーが正常に実行された	
MQMCEV_PORT_INTERFACE-FAILURE	受信側 rmmPort により使用されるネットワーク・インターフェースに障害が発生し、バックアップのネットワーク・インターフェースが使用不可である (または同じように障害が発生している)	RMM 構成
MQMCEV_PORT_INTERFACE_FAILOVER	受信側 rmmPort により使用されるネットワーク・インターフェースに障害が発生し、別のインターフェースへのフェイルオーバーが正常に実行された	RMM 構成

C によるコーディング

IBM MQ プログラムを C 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

- 1054 ページの『MQI 呼び出しのパラメーター』
- 1055 ページの『未定義データ・タイプのパラメーター』
- 1055 ページの『データ・タイプ』
- 1055 ページの『2 進ストリングの取り扱い』
- 1055 ページの『文字ストリングの取り扱い』
- 1056 ページの『構造体の初期値』
- 1056 ページの『動的構造体の初期値』
- 1057 ページの『C++ からの使用』

MQI 呼び出しのパラメーター

入力のみ でタイプが MQHCONN、MQHOBJ、MQHMSG、MQLONG のいずれかであるパラメーターは、値を使って渡されます。他のすべてのパラメーターでは、パラメーターのアドレス が値を使って渡されます。

アドレスを使って渡されるパラメーターのすべてを、機能を呼び出すたびに指定しなければならないわけではありません。 特定のパラメーターが必要でない場合は、パラメーター・データのアドレスの代わりに、機能呼び出し時にパラメーターとしてヌル・ポインターを指定できます。 これが可能なパラメーターは、呼び出し記述子で識別されます。

関数の値として戻るパラメーターはありません。C の用語では、これはすべての関数が void を戻す、と言います。

関数の属性は、MQENTRY マクロ変数によって定義されます。このマクロ変数の値は、環境に応じて異なります。

未定義データ・タイプのパラメーター

MQGET 関数、MQPUT 関数、および MQPUT1 関数にはそれぞれ、未定義データ型の **Buffer** パラメーターがあります。 このパラメーターは、アプリケーションのメッセージ・データを送受信するために使用されます。

この種類のパラメーターは、C の例で MQBYTE の配列として示されています。 この方法でパラメーターを宣言することはできますが、通常は、これらのメッセージ内のデータのレイアウトを記述する構造体として宣言する方が便利です。 関数仮パラメーターは、void を指し示すポインターとして宣言されるので、どのようなデータのアドレスでも関数呼び出し時にパラメーターとして指定できます。

データ・タイプ

すべてのデータ型は、typedef ステートメントによって定義されます。

各データ型には、対応するポインター・データ型も定義されます。 ポインター・データ・タイプの名前は、基本データ・タイプ、またはポインターを指示するためにその接頭部に P の付いた構造体データ・タイプの名前です。 ポインターの属性は、MQPOINTER マクロ変数によって定義されます。このマクロ変数の値は、環境に応じて異なります。 以下のコードはポインター・データ型を宣言する方法を示しています。

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

2 進ストリングの取り扱い

2 進データのストリングは、いずれかの MQBYTEn データ・タイプとして宣言されます。

このタイプのフィールドのコピー、比較、または設定を行うときには、C 関数の memcpy、memcmp、または memset を使用してください。

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

ストリング関数の strcpy、strcmp、strncpy、または strncmp は、MQBYTE24 として宣言されたデータに対しては正しく機能しないので、使用しないでください。

文字ストリングの取り扱い

キュー・マネージャーは、文字データをアプリケーションに戻すときに、文字データがフィールドの定義された長さになるように必ずブランク文字を埋め込みます。 キュー・マネージャーはヌル文字で終了する

ストリングを戻しませんが、入力値として使用することはできます。 したがって、そのようなストリングのコピー、比較、または連結を行うときは、ストリング関数の strncpy、strncmp、または strncat を用いてください。

ストリングがヌル文字で終了することを要求するストリング関数 (strcpy、strcmp、および strcat) を使用してはなりません。 また、ストリングの長さを判別するために関数 strlen を使用してはなりません。代わりに、関数 sizeof を使用してフィールドの長さを判別してください。

構造体の初期値

組み込みファイル <cmqc.h> は、構造体のインスタンスを宣言する際に、その構造体の初期値の設定に使用できる各種のマクロ変数を定義します。 これらのマクロ変数では、MQxxx_DEFAULT の形式の名前を使用します。 この MQxxx は構造体の名前を表します。 次のように使用します。

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

文字フィールドには、MQI により有効な特定値が定義されるものもあります (例えば、MQMD 内の *StrucId* フィールドや *Format* フィールドの場合)。 それぞれの有効値に対して、次の 2 つのマクロ変数が提供されます。

- 定義されたフィールドの長さと完全に一致する長さのストリング (暗黙指定のヌルを除く) として有効値を定義するマクロ変数。 以下の例では、記号 ¬ は単一のブランク文字を表します。

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  memcpy 関数および memcmp 関数でこの形式を使用します。
- 有効値を char (文字) の配列として定義するマクロ変数。このマクロ変数の名前は、_ARRAY を接尾部にもつストリングです。 以下に例を示します。

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  構造体のインスタンスが MQMD_DEFAULT マクロ変数で提供される値と異なる値で宣言された場合には、この書式を使用して、当該フィールドを初期化します。

動的構造体の初期値

構造体のインスタンスの変数番号が必要な場合、インスタンスは通常 calloc または malloc 関数を使用して動的に取得した主ストレージに作成されます。

そのような構造体におけるフィールドを初期化するには、次の技法をお勧めします。

1. 構造体を初期化するために、適切な MQxxx_DEFAULT マクロ変数を使用して、構造体のインスタンスを宣言する。 このインスタンスが他のインスタンスのモデル になります。

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   static または auto キーワードを宣言にコーディングして、モデルのインスタンスの存続期間を必要に応じて静的、または動的にします。
2. calloc または malloc 関数を使用して、構造体の動的インスタンスのストレージを取得する。

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. memcpy 関数を使用して、モデルのインスタンスを動的インスタンスにコピーする。

```
memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                  /* initialize dynamic instance */
```

C++ からの使用

C++ プログラム言語の場合、ヘッダー・ファイルには、C++ コンパイラーが使用されるときだけ含まれる、以下の追加のステートメントが入っています。

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Visual Basic でのコーディング

Microsoft Visual Basic で IBM MQ プログラムをコーディングする際に考慮すべき情報。 Visual Basic は、Windows でのみサポートされます。

注 :

Stabilized IBM WebSphere MQ 7.0 以降、.NET 環境外の場合、Visual Basic (VB) のサポートは IBM WebSphere MQ 6.0 レベルで構成されています。 IBM WebSphere MQ 7.0 以降で追加された最新機能の大部分は VB アプリケーションでは使用できません。 VB.NET でプログラミングする場合は、 .NET の IBM MQ クラスを使用してください。 詳細については、.NET アプリケーションの開発を参照してください。

Deprecated IBM MQ 9.0 以降、Microsoft Visual Basic 6.0 に対するサポートは非推奨になりました。 IBM MQ classes for .NET は、推奨される置換テクノロジーです。

Visual Basic と IBM MQ の間のやり取りでバイナリー・データが意図せずに変換されてしまうのを避けるため、MQSTRING ではなく MQBYTE 定義を使用してください。 CMQB.BAS には、C「バイト」定義と同等で、IBM MQ 構造体においてこれらの定義を使用する、いくつかの新しい MQBYTE タイプが定義されています。 例えば、MQMD (メッセージ記述子) 構造体には、MQBYTE24 として MsgId (メッセージ ID) が定義されています。

Visual Basic には、ポインターのデータ・タイプがありません。そのため、他の IBM MQ データ構造体への参照は、ポインターではなく、オフセットによって行われます。 2 つのコンポーネント構造体からなる複合構造体を宣言し、呼び出しの際にその複合構造体を指定します。 IBM MQ における Visual Basic のサポートには、これを可能にするための MQCONNXAny 呼び出しが含まれており、クライアント・アプリケーションでクライアント接続のチャネル・プロパティーを指定できます。 ここでは、通常の MQCNO 構造体の代わりに、タイプを持たない構造体 (MQCNOCD) が受け入れられます。

MQCNOCD 構造体は、MQCNO の後ろに MQCD が付いた複合構造体です。 この構造体は、出口ヘッダー・ファイル CMQXB で宣言されます。 MQCNOCD 構造体の初期化には、ルーチン MQCNOCD_DEFAULTS を使用してください。 MQCONNX 呼び出しを行うサンプルが提供されています (amqscnxb.vbp)。

MQCONNXAny は MQCONNX と同じパラメーターを持ちますが、MQCONNXAny の場合は、**ConnectOpts** パラメーターが MQCNO データ・タイプではなく Any データ・タイプとして宣言されます。 これにより、関数は、MQCNO 構造体でも MQCNOCD 構造体でも受け入れられるようになります。 この関数は、メインのヘッダー・ファイル CMQB で宣言されます。

関連概念

1025 ページの『Windows での Visual Basic プログラムの準備』
Microsoft Visual Basic プログラムを Windows で使用する場合に考慮する情報です。

関連資料

925 ページの『Visual Basic アプリケーションと IBM MQ MQI client・コードとのリンク』
Windows 上で IBM MQ MQI client コードと Microsoft Visual Basic アプリケーションをリンクできます。

COBOL によるコーディング

IBM MQ プログラムを COBOL 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

名前付き定数

定数の名前は、その一部に下線文字 (_) が付いて示されています。 COBOL の場合は、下線文字の代わりにハイフン文字 (-) を使用しなければなりません。 文字ストリング値をもつ定数は、単一引用符 (') をストリング区切り文字として使用します。 コンパイラーがこの文字を受け入れるには、コンパイラー・オプション APOST を使用します。

コピー・ファイル CMQV には、名前付き定数の宣言レベル 10 の項目として入っています。 この定数を使用するには、レベル 01 の項目を明示的に宣言してから、COPY ステートメントを使用して次のように定数の宣言にコピーします。

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

ただし、この方法を使用すると、定数が参照されない場合でもプログラム内のストレージを占有してしまいます。 定数が同じ実行単位内にある多数の個別プログラムに組み込まれると、定数のコピーが多数存在することになります。これによって主ストレージのかなりの容量が使用されることがあります。 レベル 01 の宣言に GLOBAL 文節を追加することによって、このような状況を避けることができます。

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

これによって、実行単位内の *1* セットの定数のためだけにストレージが割り振られます。そして定数は、レベル 01 の宣言を含むプログラムだけでなく、実行単位内の どの プログラムからでも参照できるようになります。

構造体のアライメントの確保

MQ 呼び出しの開始時に引き渡される IBM MQ 構造体が、確実にワード境界に調整されるように注意を払ってください。 ワード境界は、32 ビット・プロセスでは 4 バイト、64 ビット・プロセスでは 8 バイト、128 ビット・プロセス (IBM i) では 16 バイトです。

可能な場合は、すべての IBM MQ 構造体をまとめて配置し、すべての境界が合うように調整してください。

System/390 アセンブラー言語によるコーディング (Message Queue Interface)

IBM MQ for z/OS プログラムをアセンブラー言語でコーディングするときは、以下の節で述べられている事項に注意してください。

- 1059 ページの『名前』
- 1059 ページの『MQI 呼び出しの使用』
- 1059 ページの『定数の宣言』
- 1059 ページの『構造体の名前の指定』
- 1060 ページの『構造体の形式の指定』
- 1060 ページの『リスト出力の有無の制御』
- 1060 ページの『フィールドへの初期値の指定』
- 1060 ページの『再入可能プログラムの作成』
- 1061 ページの『CEDF の使用』

名前

呼び出しの説明にあるパラメーターの名前と、構造体の説明にあるフィールドの名前は、大/小文字混合で示されています。 IBM MQ と共に提供されるアセンブラー言語のマクロでは、すべての名前が大文字になっています。

MQI 呼び出しの使用

MQI はコール・インターフェースであるため、アセンブラー言語のプログラムは OS の連係規則に従わなければなりません。

特に、MQI 呼び出しを出す前は、アセンブラー言語のプログラムは少なくとも 18 フルワードの保管域にレジスター R13 を指定しなければなりません。 この保管域は、呼び出されたプログラム用のストレージを提供します。 これは、呼び出し側のレジスター類を内容が破壊されないように保管しておき、戻ったときにこれらのレジスターの内容を復元します。

注 : このことは、CICS アセンブラー言語のプログラムにとって重要です。このプログラムでは、DFHEIENT マクロを使用して動的ストレージを設定しますが、デフォルトの DATAREG を R13 から他のレジスターに指定変更するためです。 CICS リソース管理プログラム・インターフェースは、スタブから制御を受け取るときに、レジスターの現在の内容を R13 が指摘するアドレスに保管します。 この目的のための保管域を予約しておかないと、予測不能な結果となり、CICS で異常終了を引き起こす可能性があります。

定数の宣言

ほとんどの定数は、マクロ CMQA 内で等価として宣言されます。

ただし、以下の定数は、等価として定義することはできません。また、デフォルトのオプションを使用してマクロを呼び出しても組み込まれません。

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

これの定数を組み込むには、マクロを呼び出すときに キーワード EQUONLY=NO を追加してください。

CMQA は多重宣言に対して保護されます。したがって、何度でも組み込むことができます。 ただし、キーワード EQUONLY は、マクロが最初に組み込まれるときにしか有効ではありません。

構造体の名前の指定

構造体の複数のインスタンスを削除可能にする場合、構造体を生成するマクロは、各フィールドの接頭部にユーザーによる指定が可能なストリングと下線文字 (_) を付けます。

マクロを呼び出すときにはそのストリングを指定します。 ストリングを指定しないと、マクロ構造体の名前を使用して、接頭部を作成します。

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

呼び出しの記述に記載されている構造体の宣言は、デフォルトの接頭部を示しています。

構造体の形式の指定

マクロは、以下の 2 つの形式のうちいずれかで構造体宣言を生成することができます。これらは、DSECT パラメーターによって制御されます。

DSECT=YES
アセンブラー言語の DSECT 命令を使用して、新規のデータ・セクションを開始します。構造体の定義は、DSECT ステートメントの直後に行います。 ストレージが割り振られないため、初期化はできません。 マクロ呼び出しのラベルがデータ・セクションの名前として使用されます。ラベルが指定されていない場合は、構造体の名前が使用されます。

DSECT=NO
アセンブラー言語の DC 命令を使用して、ルーチン内の現行の位置に構造体を定義します。 このフィールドは、マクロの呼び出しに関連するパラメーターをコーディングすることにより指定できる値で初期化されます。 マクロの呼び出しで値が指定されないフィールドは、デフォルト値で初期化されます。

DSECT パラメーターを指定しないと、DSECT=NO であると解釈されます。

リスト出力の有無の制御

以下のように LIST パラメーターを使用して、アセンブラー言語のリストで構造体宣言の出力を制御することができます。

LIST=YES
構造体宣言は、アセンブラー言語のリストに出力されます。

LIST=NO
構造体宣言は、アセンブラー言語のリストに出力されません。 LIST パラメーターを指定しないと、LIST=NO であると解釈されます。

フィールドへの初期値の指定

構造体内のフィールドを初期化するのに使用する値を、フィールドの名前を (接頭部なしで) コーディングすることによって指定できます。このとき値は必要な値を伴った、マクロの呼び出し時のパラメーターとしてコーディングします。

例えば、メッセージ記述子の構造体を MQMT_REQUEST で初期化された *MsgType* フィールド、およびストリング MY_REPLY_TO_QUEUE で初期化された *ReplyToQ* フィールドによって宣言するために、次に示すコードを使用することができます。

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

名前付きの定数 (または等価の値) をマクロの呼び出し時の値として指定する場合には、CMQA マクロを使用してその名前付き定数を定義してください。 値が文字ストリングの場合には、 単一引用符 (' ') で囲まないでください。

再入可能プログラムの作成

IBM MQ MQ は入力と出力の両方に構造体を使用します。 プログラムを再入可能のままにしておきたい場合は、次のようにします。

1. 構造体の作業用ストレージ・バージョンを DSECT として定義するか、構造体を既に定義された DSECT 内にインラインに定義する。 次にその DSECT を、取得したストレージに次のものを用いてコピーする。
   - バッチおよび TSO プログラムの場合は、STORAGE または GETMAIN z/OS アセンブラー・マクロ
   - CICS の場合は、作業用ストレージ DSECT (DFHEISTG) または EXEC CICS GETMAIN コマンド

   これらの作業用ストレージ構造体を正しく初期化するには、対応する構造体の定数バージョンを作業用ストレージ・バージョンにコピーしてください。

   注 : MQMD および MQXQH 構造体の長さは、それぞれ 256 バイトを超えます。 これらの構造体をストレージにコピーするには、MVCL アセンブラー命令を使います。
2. CALL マクロの LIST 形式 (MF=L) を使用して、ストレージ内にスペースを予約する。 MQI 呼び出しを行うために CALL マクロを使用するときは、1061 ページの『CEDF の使用』 の例に示されるように、前に予約されているストレージを使用して、マクロの EXECUTE 形式 (MF=E) を使います。 この具体的な実行例については、IBM MQ と共に出荷されるアセンブラー言語サンプル・プログラムを参照してください。

プログラムが再入可能かどうかを判別するには、アセンブラー言語の RENT オプションを使用してください。

再入可能プログラムの作成については、「*z/OS MVS アプリケーション開発ガイド: アセンブラー言語プログラム*」を参照してください。

CEDF の使用

CICS 提供のトランザクション (CEDF (CICS 実行診断機能)) をプログラムのデバッグに有効利用したい場合は、例えば次のように、各 CALL ステートメントに ,VL キーワードを追加してください。

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

上記の例は、再入可能なアセンブラー言語コードであり、ここにある PARMAREA は、指定した作業用ストレージ内の領域です。

MQI 呼び出しの使用

MQI はコール・インターフェースであるため、アセンブラー言語のプログラムは OS の連係規則に従わなければなりません。 特に、MQI 呼び出しを出す前は、アセンブラー言語のプログラムは少なくとも 18 フルワードの保管域にレジスター R13 を指定しなければなりません。 この保管域は、呼び出されたプログラム用のストレージを提供します。 これは、呼び出し側のレジスター類を内容が破壊されないように保管しておき、戻ったときにこれらのレジスターの内容を復元します。

注 : このことは、CICS アセンブラー言語のプログラムにとって重要です。このプログラムでは、DFHEIENT マクロを使用して動的ストレージを設定しますが、デフォルトの DATAREG を R13 から他のレジスターに指定変更するためです。 CICS リソース管理プログラム・インターフェースは、スタブから制御を受け取るときに、レジスターの現在の内容を R13 が指摘するアドレスに保管します。 この目的のための適切な保管域を予約しておかないと、予測不能な結果となり、CICS で異常終了を引き起こす可能性があります。

IBM i RPG での IBM MQ プログラムのコーディング (IBM i のみ)

IBM MQ 資料では、呼び出しのパラメーター、データ・タイプの名前、構造体のフィールド、および定数の名前をすべてロング・ネームで記述しています。 RPG では、これらの名前が 6 文字以内の大文字に短縮されます。

例えば、フィールド *MsgType* は、RPG では *MDMT* になります。 詳細については、IBM i アプリケーション・プログラミングの参照情報 (ILE/RPG)を参照してください。

PL/I でのコーディング (z/OS のみ)

PL/I で IBM MQ のコーディングを行うときに役立つ情報です。

構造体

構造体は BASED 属性で宣言されるので、プログラムが構造体の 1 つまたは複数のインスタンスを宣言しない限りはどのストレージも占有してはなりません。

構造体インスタンスは、次の例のように like 属性を用いて宣言することができます。

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

構造体のフィールドは、INITIAL 属性で宣言されます。したがって、構造体のインスタンスを宣言するために like 属性が使用されるときは、そのインスタンスは該当の構造体に定義されている初期値を継承します。 設定する必要があるのは、初期値と異なる値を必要とするフィールドだけです。

PL/I では、大文字と小文字を区別しないので、呼び出し名、構造体フィールドの名前、および定義の名前を小文字、大文字、または大文字小文字混合のいずれでも記述できます。

名前付き定数

名前付き定数は、マクロ変数として宣言されます。その結果、プログラムによって参照されない名前付き定数は、コンパイル済みプロシージャーでどのストレージも占有しません。

しかし、プログラムをコンパイルするときに、ソースをマクロ・プリプロセッサーによって処理させるコンパイラー・オプションを指定しなければなりません。

マクロ変数は、たとえ数値を表すものでも、すべて文字変数です。 これは、データ・タイプに矛盾が生じるように思われますが、マクロ処理プログラムによってマクロ変数が置換された後でも、何の矛盾も生じません。次に例を示します。

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

IBM MQ プロシージャー型サンプル・プログラムの使用

これらのサンプル・プログラムは、プロシージャー型言語で作成されており、Message Queue Interface (MQI) の標準的な使用法を示しています。 異なるプラットフォーム上の IBM MQ プログラム。

このタスクについて

次の 2 セットのサンプルがあります。

- 分散システムと IBM i 用のサンプル・プログラム。
- z/OS 用のサンプル・プログラム。

手順

- 以下のリンクを使用して、サンプル・プログラムについての詳細を確認してください。
  - 1063 ページの『Multiplatforms でのサンプル・プログラムの使用』
  - z/OS 1166 ページの『z/OS 用サンプル・プログラムの使用』

関連概念

7 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM MQ アプリケーションを作成することができます。 IBM MQ アプリケーションの設計と記述を開始する前に、IBM MQ の基本概念について理解しておいてください。

5 ページの『IBM MQ 用アプリケーションの開発』

メッセージを送受信するためのアプリケーション、およびキュー・マネージャーや関連リソースを管理するためのアプリケーションを開発できます。 IBM MQ は、さまざまな言語やフレームワークで作成されたアプリケーションをサポートします。

49 ページの『IBM MQ アプリケーションの設計上の考慮事項』
プラットフォームや環境を、アプリケーションによってどのように利用できるか判断したら、IBM MQ によって提供される機能の使用方法を判別する必要があります。

721 ページの『プロシージャー型キューイング・アプリケーションの作成』
この情報を使用して、キューイング・アプリケーションの作成、キュー・マネージャーへの接続およびキュー・マネージャーからの切断、パブリッシュ/サブスクライブ、およびオブジェクトの開閉について説明します。

918 ページの『プロシージャー型クライアント・アプリケーションの作成』
IBM MQ でプロシージャー型言語を使用してクライアント・アプリケーションを作成するために知っておくべき内容。

809 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
パブリッシュ/サブスクライブ IBM MQ アプリケーションの作成を開始します。

1005 ページの『プロシージャー型アプリケーションの構築』
IBM MQ アプリケーションをプロシージャー型言語のいずれかで作成し、そのアプリケーションを複数のさまざまなプラットフォームで実行することができます。

1042 ページの『プロシージャー型プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

Multi Multiplatforms でのサンプル・プログラムの使用

これらのプロシージャー型サンプル・プログラムは製品に同梱されています。 サンプルは C および COBOL で作成されており、メッセージ・キュー・インターフェース (MQI) の一般的な使用法を示します。

このタスクについて

このサンプルは、一般的なプログラミング技法の解説を目的としたものではありません。したがって、実動プログラムには組み込んだほうがよいエラー・チェックの一部が省略されています。

すべてのサンプルのソース・コードが、この製品で提供されています。このソースには、プログラムによって示されるメッセージ・キューイング技法を説明するコメントが含まれています。

IBM i RPG プログラミングについては、IBM i アプリケーション・プログラミングの参照情報 (ILE/RPG) を参照してください。

サンプル・プログラム名は接頭部 amq で始まっています。 名前の中の 4 番目の文字は、プログラム言語と、必要な場合にはコンパイラーを示します。

- s: C 言語
- 0: IBM および Micro Focus コンパイラーでの COBOL 言語
- i: IBM コンパイラーのみでの COBOL 言語
- m: Micro Focus コンパイラーのみでの COBOL 言語

実行可能ファイルの 8 番目の文字は、サンプルをローカル・バインディング・モードまたはクライアント・モードのどちらで実行するかを示します。 8 番目の文字がない場合、サンプルはローカル・バインディング・モードで実行します。 8 番目の文字が「c」である場合、サンプルはクライアント・モードで実行します。

サンプル・アプリケーションを実行するには、その前にキュー・マネージャーを作成して構成する必要があります。 キュー・マネージャーがクライアント接続を受け入れるようにセットアップするには、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』を参照してください。

手順

- 以下のリンクを使用して、サンプル・プログラムについての詳細を確認してください。
  - 1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』
  - 1073 ページの『サンプル・プログラムの作成と実行』
  - 1079 ページの『API 出口サンプル・プログラム』
  - 1080 ページの『非同期コンシューム・サンプル・プログラム』
  - 1081 ページの『非同期書き込みサンプル・プログラム』
  - 1082 ページの『ブラウズ・サンプル・プログラム』
  - 1083 ページの『ブラウザー・サンプル・プログラム』
  - 1084 ページの『CICS トランザクション・サンプル』
  - 1084 ページの『Connect サンプル・プログラム』
  - 1086 ページの『データ変換サンプル・プログラム』
  - 1086 ページの『データベース調整サンプル』
  - 1093 ページの『送達不能キュー・ハンドラーのサンプル』
  - 1093 ページの『配布リスト・サンプル・プログラム』
  - 1094 ページの『エコー・サンプル・プログラム』
  - 1095 ページの『読み取りサンプル・プログラム』
  - 1096 ページの『高可用性のサンプル・プログラム』
  - 1101 ページの『照会サンプル・プログラム』
  - 1102 ページの『メッセージ処理サンプル・プログラムの照会プロパティー』
  - 1102 ページの『パブリッシュ/サブスクライブのサンプル・プログラム』
  - 1106 ページの『パブリッシュ出口サンプル・プログラム』
  - 1107 ページの『書き込みサンプル・プログラム』
  - 1109 ページの『参照メッセージ・サンプル・プログラム』
  - 1117 ページの『要求サンプル・プログラム』
  - 1123 ページの『設定サンプル・プログラム』
  - 1124 ページの『TLS サンプル・プログラム』
  - 1127 ページの『トリガー・サンプル・プログラム』
  - 1129 ページの『AIX, Linux, and Windows での TUXEDO サンプルの使用』
  - 1140 ページの『Windows での SSPI セキュリティー出口の使用』
  - 1141 ページの『リモート・キューを使用するサンプルの実行』
  - 1141 ページの『クラスター・キュー・モニターのサンプル・プログラム (AMQSCLM)』
  - 1151 ページの『接続エンドポイント検索 (CEPL) のサンプル・プログラム』

関連概念

528 ページの『C++ サンプル・プログラム』
メッセージの取得と書き込みのデモ用に 4 つのサンプル・プログラムが提供されています。

関連タスク

1166 ページの『z/OS 用サンプル・プログラムの使用』
IBM MQ for z/OS と共に提供されるプロシージャー型サンプル・アプリケーションは、Message Queue Interface (MQI) の一般的な使用法を示しています。

Multi *Multiplatforms のサンプル・プログラムで示されている機能*

一連の表に、IBM MQ のサンプル・プログラムで示される技法について説明します。

すべてのサンプルは、MQOPEN および MQCLOSE 呼び出しを使用してキューをオープンおよびクローズするので、これらの技法はこの表では別々に記載されていません。 以下の中から、該当するプラットフォームが含まれている見出しを参照してください。

z/OS z/OS プラットフォームについては、1166 ページの『z/OS 用サンプル・プログラムの使用』を参照してください。

Linux AIX *AIX and Linux* システム用のサンプル
IBM MQ for AIX or Linux のサンプル・プログラムで示される技法。

IBM MQ for AIX or Linux のサンプル・プログラムが保管されている場所については、1076 ページの『AIX and Linux でのサンプル・プログラムの作成と実行』 を参照してください。

1065 ページの表 156 この表は、提供されている C および COBOL ソース・ファイル、およびサーバーまたはクライアント実行可能プログラムが含まれているかどうかをリストしています。

表 *156. AIX and Linux* での *MQI (C* および *COBOL)* の使用方法を示すサンプル・プログラム.

4 列の表。 1 列目には、サンプルに示されている技法をリストしています。 2 列目および 3 列目には、1 列目にリストした各技法が示されている C サンプルと COBOL サンプルをそれぞれリストしています。 4 列目は、サーバーの C 実行可能プログラムが含まれているかどうか、5 列目は、クライアントの C 実行可能プログラムが含まれているかどうかを示しています。

技法	C (ソース) (1067 ページの『1』)	COBOL (ソース) (1067 ページの『2』)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
パブリッシュ/サブスクライブ・インターフェースの使用	amqspuba amqssuba amqssbxa	サンプルなし	amqspub amqssub amqssbx	サンプルなし
MQPUT 呼び出しを使用するメッセージの書き込み	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 呼び出しを使用する単一メッセージの書き込み	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
配布リストへのメッセージの書き込み (1067 ページの『3』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
要求メッセージに対する応答	amqsinqa	amqminqx amqiinqx	amqsinq	サンプルなし
参照によるメッセージの読み取り (待機なし)	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
メッセージの読み取り (時間制限付きの待機)	amqsget0	amq0get0	amqsget	amqsgetc
メッセージの読み取り (無制限の待機)	amqstrg0	サンプルなし	amqstrg	amqstrgc
メッセージの読み取り (データ変換あり)	amqsecha	サンプルなし	amqsech	サンプルなし
キューへの参照メッセージの書き込み (1067 ページの『3』)	amqsprma	サンプルなし	amqsprm	amqsprmc
キューからの参照メッセージの読み取り (1067 ページの『3』)	amqsgrma	サンプルなし	amqsgrm	amqsgrmc
参照メッセージのチャネル出口 (1067 ページの『3』)	amqsqrma amqsxrma	サンプルなし	amqsxrm	サンプルなし
メッセージの最初の 20 文字のブラウズ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージ全体のブラウズ	amqsbcg0	サンプルなし	amqsbcg	amqsbcgc

表 156. *AIX and Linux* での *MQI (C* および *COBOL)* の使用方法を示すサンプル・プログラム.

4 列の表。 1 列目には、サンプルに示されている技法をリストしています。 2 列目および 3 列目には、1 列目にリストした各技法が示されている C サンプルと COBOL サンプルをそれぞれリストしています。 4 列目は、サーバーの C 実行可能プログラムが含まれているかどうか、5 列目は、クライアントの C 実行可能プログラムが含まれているかどうかを示しています。

(続き)

技法	C (ソース) (1067 ページの『1』)	COBOL (ソース) (1067 ページの『2』)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
共用入力キューの使用	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
排他的入力キューの使用	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 呼び出しの使用	amqsinqa	amqminqx amqiinqx	amqsinq	サンプルなし
MQSET 呼び出しの使用	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
応答先キューの使用	amqsreq0	amq0req0	amqsreq	amqsreqc
メッセージ例外の要求	amqsreq0	amq0req0	amqsreq	サンプルなし
切り捨てられたメッセージの受け入れ	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
解決されたキュー名の使用	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
処理のトリガー操作	amqstrg0	サンプルなし	amqstrg	amqstrgc
データ変換の使用	(1067 ページの『4』)	サンプルなし	サンプルなし	サンプルなし
SQL を使用する単一データベースにアクセスする IBM MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	サンプルなし	サンプルなし
SQL を使用する 2 つのデータベースにアクセスする IBM MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	サンプルなし	サンプルなし
CICS トランザクション (1067 ページの『5』)	amqscic0.ccs	サンプルなし	amqscic0	サンプルなし
Encina トランザクション (1067 ページの『3』)	amqsxae0	サンプルなし	amqsxae0	サンプルなし
メッセージを書き込む TUXEDO トランザクション (1067 ページの『6』)	amqstxpx	サンプルなし	サンプルなし	サンプルなし
メッセージを読み取る TUXEDO トランザクション (1067 ページの『6』)	amqstxgx	サンプルなし	サンプルなし	サンプルなし
TUXEDO のサーバー (1067 ページの『6』)	amqstxsx	サンプルなし	サンプルなし	サンプルなし
送達不能キュー・ハンドラー	ディレクトリー ./tools/c/Samples/dlq (1067 ページの『7』)	サンプルなし	amqsdlq	サンプルなし

表 156. *AIX and Linux* での *MQI (C* および *COBOL)* の使用方法を示すサンプル・プログラム.

4 列の表。 1 列目には、サンプルに示されている技法をリストしています。 2 列目および 3 列目には、1 列目にリストした各技法が示されている C サンプルと COBOL サンプルをそれぞれリストしています。 4 列目は、サーバーの C 実行可能プログラムが含まれているかどうか、5 列目は、クライアントの C 実行可能プログラムが含まれているかどうかを示しています。

(続き)

技法	C (ソース) (1067 ページの『1』)	COBOL (ソース) (1067 ページの『2』)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
MQI クライアントからのメッセージの書き込み	サンプルなし	サンプルなし	サンプルなし	amqsputc
MQI クライアントからのメッセージの読み取り	サンプルなし	サンプルなし	サンプルなし	amqsgetc
MQCONNX を使用するキュー・マネージャーへの接続	amqscnxc	サンプルなし	サンプルなし	amqscnxc
API 出口の使用	amqsaxe0	サンプルなし	amqsaxe	サンプルなし
クラスター・ワークロード・バランシング出口	amqswlm0	サンプルなし	amqswlm	サンプルなし
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	amqsapt0	サンプルなし	amqsapt	amqsaptc
再接続可能クライアント	amqsphac amqsghac amqsmhac	サンプルなし	適用外	amqsphac amqsghac amqsmhac
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用	amqscbf0	サンプルなし	amqscbf	amqscbfc
MQCONNX での TLS 接続情報の指定	amqssslc	サンプルなし	適用外	amqssslc

注 :

1. IBM MQ MQI client の実行可能なバージョンのサンプルは、サーバー環境で実行されるサンプルと同じソースを共用します。
2. 「amqm」で始まるプログラムは Micro Focus COBOL コンパイラーで、「amqi」で始まるプログラムは IBM COBOL コンパイラーで、また「amq0」で始まるプログラムはこのどちらかで、それぞれコンパイルしてください。
3. AIX IBM MQ for AIX でのみサポートされています。
4. AIX IBM MQ for AIX でこのプログラムは `amqsvfc0.c` と呼ばれます。
5. AIX CICS は IBM MQ for AIX によってのみサポートされます。
6. Linux TUXEDO は System p 上の IBM MQ for Linux でサポートされていません。
7. 送達不能キュー・ハンドラーのソースは、複数のファイルから構成され、個別のディレクトリーに提供されます。

AIX and Linux システムのサポートの詳細については、IBM MQ のシステム要件を参照してください。

Windows *IBM MQ for Windows* のサンプル

IBM MQ for Windows のサンプル・プログラムで示される技法。

1068 ページの表 157 は、提供されている C および COBOL ソース・ファイル、およびサーバーまたはクライアント実行可能プログラムが含まれているかどうかをリストしています。

表 157. *MQI* の使用方法を示す *IBM MQ for Windows* サンプル・プログラム *(C* および *COBOL)*

技法	C (ソース)	COBOL (ソース)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
パブリッシュ/サブスクライブ・インターフェースの使用	amqspuba amqssuba amqssbxa	サンプルなし	amqspub amqssub amqssbx	サンプルなし
MQPUT 呼び出しを使用するメッセージの書き込み	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 呼び出しを使用する単一メッセージの書き込み	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
配布リストへのメッセージの書き込み	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
要求メッセージに対する応答	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
メッセージを読み取る (待機間隔なし)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージの読み取り (時間制限付きの待機)	amqsget0	amq0get0	amqsget	amqsgetc
メッセージの読み取り (無制限の待機)	amqstrg0	サンプルなし	amqstrg	amqstrgc
メッセージの読み取り (データ変換あり)	amqsecha	サンプルなし	amqsech	amqsechc
キューへの参照メッセージの書き込み	amqsprma	サンプルなし	amqsprm	amqsprmc
キューからの参照メッセージの読み取り	amqsgrma	サンプルなし	amqsgrm	amqsgrmc
参照メッセージのチャネル出口	amqsqrma amqsxrma	サンプルなし	amqsxrm	サンプルなし
メッセージの最初の 20 文字のブラウズ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージ全体のブラウズ	amqsbcg0	サンプルなし	amqsbcg	amqsbcgc
共用入力キューの使用	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
排他的入力キューの使用	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 呼び出しの使用	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
MQSET 呼び出しの使用	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
MQINQMP 呼び出しの使用	amqsiqma	サンプルなし	サンプルなし	サンプルなし
応答先キューの使用	amqsreq0	amq0req0	amqsreq	amqsreqc
メッセージ例外の要求	amqsreq0	amq0req0	amqsreq	amqsreqc
切り捨てられたメッセージの受け入れ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
解決されたキュー名の使用	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
処理のトリガー操作	amqstrg0	サンプルなし	amqstrg	amqstrgc

表 157. *MQI* の使用方法を示す *IBM MQ for Windows* サンプル・プログラム *(C* および *COBOL) (*続き*)*

技法	C (ソース)	COBOL (ソース)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
データ変換の使用	amqsvfc0	サンプルなし	サンプルなし	サンプルなし
SQL を使用する単一データベースにアクセスする IBM MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	サンプルなし	サンプルなし
SQL を使用する 2 つのデータベースにアクセスする IBM MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	サンプルなし	サンプルなし
メッセージを書き込む TUXEDO トランザクション	amqstxpx	サンプルなし	サンプルなし	サンプルなし
メッセージを読み取る TUXEDO トランザクション	amqstxgx	サンプルなし	サンプルなし	サンプルなし
TUXEDO のサーバー	amqstxsx	サンプルなし	サンプルなし	サンプルなし
送達不能キュー・ハンドラー	ディレクトリー ./tools/c/Samples/dlq (1070 ページの『1』)	サンプルなし	amqsdlq	サンプルなし
IBM MQ MQI client からのメッセージの書き込み	サンプルなし	サンプルなし	サンプルなし	amqsputc
IBM MQ MQI client からのメッセージの読み取り	サンプルなし	サンプルなし	サンプルなし	amqsgetc
MQCONNX を使用するキュー・マネージャーへの接続	amqscnxc	サンプルなし	サンプルなし	amqscnxc
API 出口の使用	amqsaxe0	サンプルなし	amqsaxe	サンプルなし
クラスター・ワークロード・バランシング	amqswlm0	サンプルなし	amqswlm	サンプルなし
SSPI セキュリティー・ルーチン	amqsspin	サンプルなし	amqrspin.dll	amqrspin.dll
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	amqsapt0	サンプルなし	amqsapt	amqsaptc
再接続可能クライアント	amqsphac amqsghac amqsmhac	サンプルなし	適用外	amqsphac amqsghac amqsmhac
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用	amqscbf0	サンプルなし	amqscbf	amqscbfc

表 157. MQI の使用方法を示す IBM MQ for Windows サンプル・プログラム (C および COBOL) (続き)				
技法	**C (ソース)**	**COBOL (ソース)**	**サーバー (C 実行可能プログラム)**	**クライアント (C 実行可能プログラム)**
MQCONNX での TLS 接続情報の指定	amqssslc	サンプルなし	適用外	amqssslc

注 :

1. 送達不能キュー・ハンドラーのソースは、複数のファイルから構成され、個別のディレクトリーに提供されます。

Windows *IBM MQ for Windows 用の Visual Basic のサンプル*

Windows システム上の IBM MQ 用のサンプル・プログラムによって示される技法。

1070 ページの表 158 に、IBM MQ for Windows サンプル・プログラムで示される技法についてまとめます。

プロジェクトには複数のファイルが含まれている可能性があります。 Visual Basic 内のプロジェクトをオープンすると、その他のファイルは自動的に読み込まれます。 実行可能プログラムはありません。

mqtrivc.vbp 以外のサンプル・プロジェクトはすべて、IBM MQ サーバーとともに使用されるように設定されています。 IBM MQ クライアントで使用されるようにサンプル・プロジェクトを変更する方法については、1025 ページの『Windows での Visual Basic プログラムの準備』を参照してください。

表 158. MQI の使用方法を示す IBM MQ for Windows サンプル・プログラム (Visual Basic)	
技法	**プロジェクト・ファイル名**
MQPUT 呼び出しを使用するメッセージの書き込み	amqsputb.vbp
MQGET 呼び出しを使用するメッセージの読み取り	amqsgetb.vbp
MQGET 呼び出しを使用するキューのブラウズ	amqsbcgb.vbp
単純な MQGET および MQPUT サンプル (クライアント)	mqtrivc.vbp
単純な MQGET および MQPUT サンプル (サーバー)	mqtrivs.vbp
MQPUT および MQGET を使用するストリングおよびユーザー定義構造体の、書き込みおよび読み取り	strings.vbp
PCF 構造体を使用したチャネルの開始および停止	pcfsamp.vbp
MQAI を使用するキューの作成	amqsaicq.vbp
MQAI を使用するキュー・マネージャーのキューのリスト表示	amqsailq.vbp
MQAI を使用するイベントのモニター	amqsaiem.vbp

IBM i *IBM i のサンプル*

IBM i システム上の IBM MQ 用のサンプル・プログラムによって示される技法。

1071 ページの表 159 に、IBM MQ for IBM i サンプル・プログラムで示される技法についてまとめます。 技法の中には複数のサンプル・プログラムで使用されているものもありますが、表には 1 つのプログラムのみを示しています。

表 159. *IBM i での MQI (C および COBOL)* の使用方法を示すサンプル・プログラム

技法	C (ソース) (1072 ページの『1』)	COBOL (ソース) (1072 ページの『2』)	RPG (ソース) (1072 ページの『3』)	Client (実行可能プログラム) (4)
MQPUT 呼び出しを使用するメッセージの書き込み	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	AMQSPUTC
MQPUT 呼び出しを使用する、データ・ファイルからのメッセージの書き込み	AMQSPUT4	サンプルなし	サンプルなし	サンプルなし
MQPUT1 呼び出しを使用する単一メッセージの書き込み	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
配布リストへのメッセージの書き込み	AMQSPTL4	サンプルなし	サンプルなし	AMQSPTLC
要求メッセージに対する応答	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
メッセージを読み取る (待機間隔なし)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
メッセージの読み取り (時間制限付きの待機)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
メッセージの読み取り (無制限の待機)	AMQSTRG4	サンプルなし	AMQ3TRG4	AMQSTRGC
メッセージの読み取り (データ変換あり)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
キューへの参照メッセージの書き込み	AMQSPRM4	サンプルなし	サンプルなし	AMQSPRMC
キューからの参照メッセージの読み取り	AMQSGRM4	サンプルなし	サンプルなし	AMQSGRMC
参照メッセージのチャネル出口	AMQSQRM4, AMQSXRM4	サンプルなし	サンプルなし	サンプルなし
メッセージ出口	AMQSCMX4	サンプルなし	サンプルなし	サンプルなし
メッセージの最初の 49 文字のブラウズ	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
メッセージ全体のブラウズ	AMQSBCG4	サンプルなし	サンプルなし	AMQSBCGC
共用入力キューの使用	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
排他的入力キューの使用	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
MQINQ 呼び出しの使用	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
MQSET 呼び出しの使用	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
応答先キューの使用	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
メッセージ例外の要求	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
切り捨てられたメッセージの受け入れ	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
解決されたキュー名の使用	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
処理のトリガー操作	AMQSTRG4	サンプルなし	AMQ3TRG4	AMQSTRGC
トリガー・サーバー	AMQSERV4	サンプルなし	AMQ3SRV4	サンプルなし
トリガー・サーバーの使用 (CICS トランザクションを含む)	AMQSERV4	サンプルなし	AMQ3SRV4	サンプルなし
データ変換の使用	AMQSVFC4	サンプルなし	サンプルなし	サンプルなし
API 出口の使用	AMQSAXE0	サンプルなし	サンプルなし	サンプルなし
クラスター・ワークロード・バランシング	AMQSWLM0	サンプルなし	サンプルなし	サンプルなし

表 159. *IBM i での MQI (C および COBOL)* の使用方法を示すサンプル・プログラム (続き)

技法	C (ソース) (1072 ページの『1』)	COBOL (ソース) (1072 ページの『2』)	RPG (ソース) (1072 ページの『3』)	Client (実行可能プログラム) (4)
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	AMQSAPT0	サンプルなし	サンプルなし	AMQSAPTC
パブリッシュ/サブスクライブ・インターフェースの使用	AMQSPUBA, AMQSSUBA, AMQSSBXA	サンプルなし	サンプルなし	AMQSPUBC, AMQSSUBC, AMQSSBXC
再接続可能なクライアント (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	サンプルなし	サンプルなし	サンプルなし
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用 (5)	AMQSCBFO	サンプルなし	サンプルなし	サンプルなし
MQCONNX での TLS 接続情報の指定	AMQSSSLC	サンプルなし	サンプルなし	AMQSSSLC
MQCONNX を使用するキュー・マネージャーへの接続	AMQSCNXC	サンプルなし	サンプルなし	AMQSCNXC
MQINQMP を使用したメッセージ・キューからのメッセージ・ハンドルのプロパティーの照会	AMQISQMA (MQISQMA)	サンプルなし	サンプルなし	AMQISQMC (SQL)
MQSETMP を使用してメッセージ・ハンドルのプロパティーを設定し、それをメッセージ・キューに書き込む	AMQSSQMA (MQSSQMA)	サンプルなし	サンプルなし	AMQSSQMC (SQL)

注:

1. C サンプル・プログラムのソースはファイル QMQMSAMP/QCSRC 内にあります。 組み込みファイルはメンバーとしてファイル QMQM/H 中にあります。
2. COBOL サンプル・プログラムのソースは、ファイル QMQMSAMP/QCBLLESRC 内にあります。 メンバーの名前は AMQ0xxx4 (xxx はサンプル機能を示す) になります。
3. RPG サンプル・プログラムのソースは QMQMSAMP/QRPGLESRC 内にあります。 メンバーの名前は AMQ3xxx4 (xxx はサンプル機能を示す) になります。 コピー・メンバーは QMQM/QRPGLESRC に存在します。 個々のメンバー名には接尾部 G が付けられます。
4. IBM MQ MQI client の実行可能なバージョンのサンプルは、サーバー環境で実行されるサンプルと同じソースを共用します。 クライアント環境のサンプルのソースは、サーバーのものと同じです。 IBM MQ MQI client のサンプルはクライアント・ライブラリー LIBMQIC にリンクされ、 IBM MQ サーバー・サンプルはサーバー・ライブラリー LIBMQM にリンクされます。
5. 再接続可能なクライアントのサンプル・アプリケーションおよび非同期コンシューマー・アプリケーションのクライアント実行可能プログラムを実行する必要がある場合、それをスレッド化ライブラリー LIBMQIC_R を使用してコンパイルおよびリンクする必要があります。 このため、このプログラムはスレッド化環境で実行される必要があります。 環境変数 QIBM_MULTI_THREADED を 'Y' に設定し、アプリケーションを qsh から実行します。

   詳しくは、 Java および JMS を使用した IBM MQ のセットアップ を参照してください。

詳しくは、 1075 ページの『IBM i でのサンプル・プログラムの作成と実行』 を参照してください。

これらのほかに、IBM MQ for IBM i サンプル・オプションには、サンプル・データ・ファイルが含まれます。サンプル・データ・ファイルは、サンプル・プログラム AMQSDATA およびサンプル CL プログラム (管理タスクを実行する) への入力として使用します。 CL サンプルについては、IBM i の管理を参照してください。 サンプル CL プログラム amqsamp4 を使用すると、このトピックに示すサンプル・プログラム用のキューを作成できます。

Multi サンプル・プログラムの作成と実行

いくつかの初期準備が完了したら、サンプル・プログラムを実行できます。

このタスクについて

サンプル・プログラムを実行する前に、あらかじめキュー・マネージャーを作成する必要があります。また、必要なキューも作成しなければなりません。 例えば、COBOL サンプルを実行する場合などには、追加の準備が必要となることもあります。 必要な準備が完了したら、サンプル・プログラムを実行できます。

手順

サンプル・プログラムの準備と実行の方法については、次のトピックを参照してください。

- 1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』
- 1075 ページの『IBM i でのサンプル・プログラムの作成と実行』
- 1076 ページの『AIX and Linux でのサンプル・プログラムの作成と実行』
- 1077 ページの『Windows でのサンプル・プログラムの作成と実行』

Multi クライアント接続を受け入れるようにキュー・マネージャーを構成する *(Multiplatforms)*

サンプル・アプリケーションを実行するには、その前にキュー・マネージャーを作成する必要があります。その後、クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるように、キュー・マネージャーを構成できます。

始める前に

キュー・マネージャーが既に存在しており、開始していることを確認します。 MQSC コマンドを実行することにより、チャネル認証レコードが既に使用可能になっているかどうかを判別します。

```
DISPLAY QMGR CHLAUTH
```

重要 : このタスクは、チャネル認証レコードが使用可能になっていることを前提としています。 このキュー・マネージャーが他のユーザーやアプリケーションによって使用されている場合、この設定を変更すると、他のすべてのユーザーとアプリケーションが影響を受けます。 キュー・マネージャーがチャネル認証レコードを利用しない場合には、ステップ 4 を代替認証方式 (例えば、セキュリティー出口など) に置き換えて、MCAUSER をステップ 1073 ページの『1』 で取得する *non-privileged-user-id* に設定することができます。

アプリケーションが使用すると予期されるチャネル名を把握し、アプリケーションがそのチャネルを使用できるようにする必要があります。 また、アプリケーションが使用すると予期されるオブジェクト (例えば、キューやトピック) も把握し、アプリケーションがこれらのオブジェクトを使用できるようにする必要もあります。

このタスクについて

このタスクにより、キュー・マネージャーに接続するクライアント・アプリケーションで使用する、非特権ユーザー ID が作成されます。 クライアント・アプリケーションがこのユーザー ID を使用して必要とするチャネルとキューを使用できるようにするためにのみ、アクセス権限が付与されます。

手順

1. キュー・マネージャーが実行されているシステムでユーザー ID を取得します。 このタスクの場合、このユーザー ID は特権管理ユーザーにすることはできません。 このユーザー ID の権限は、クライアント接続をキュー・マネージャーで実行するためのものです。
2. 以下のコマンドを使用してリスナー・プログラムを開始します。

   qmgr-name は、キュー・マネージャーの名前です。
   nnnn は、選択したポート番号です。

a) ALW
AIX, Linux, and Windows システムの場合:

```
runmqlsr -t tcp -m qmgr-name -p nnnn
```

b) IBM i
IBM i の場合:

```
STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
```

3. アプリケーションが SYSTEM.DEF.SVRCONN を使用する場合、このチャネルは既に定義済みです。 アプリケーションが別のチャネルを使用する場合は、それを次の MQSC コマンドを発行して作成します。

```
DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Channel for use by sample programs')
```

channel-name は、チャネルの名前です。

4. ご使用のクライアント・システムの IP アドレスのみにチャネルの使用を許可するチャネル認証規則を、以下のように MQSC コマンドを発行して作成します。

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

説明:

channel-name は、チャネルの名前です。
client-machine-IP-address は、クライアント・システムの IP アドレスです。 サンプル・クライアント・アプリケーションがキュー・マネージャーと同じマシン上で実行されているときに、そのアプリケーションが「localhost」を使用して接続しようとしているのであれば、IP アドレス「127.0.0.1」を使用します。 複数のさまざまなクライアント・マシンが接続することになっている場合、単一の IP アドレスではなく、パターンや範囲を使用することができます。 詳細については、汎用 IP アドレスを参照してください。
non-privileged-user-id は、ステップ 1073 ページの『1』 で取得したユーザー ID です。

5. アプリケーションが SYSTEM.DEFAULT.LOCAL.QUEUE を使用する場合、このキューは既に定義済みです。 アプリケーションが別のキューを使用する場合は、それを次の MQSC コマンドを発行して作成します。

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

queue-name は、キューの名前です。

6. 以下の MQSC コマンドを発行して、キュー・マネージャーに接続して照会を実行するための権限を付与します。

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

non-privileged-user-id は、ステップ 1073 ページの『1』 で取得したユーザー ID です。

7. アプリケーションが Point-to-Point アプリケーションである場合 (つまりキューを使用する場合)、キューを使用したメッセージの照会、書き込み、および読み取りを、ユーザー ID を使用して行えるようにするために、以下のように MQSC コマンドを発行してアクセス権限を付与します。

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

説明:

queue-name は、キューの名前です。
non-privileged-user-id は、ステップ 1073 ページの『1』 で取得したユーザー ID です。

8. アプリケーションがパブリッシュ/サブスクライブ・アプリケーションである場合 (つまりトピックを使用する場合) 、使用するユーザー ID による、トピックを使用したパブリッシュ/サブスクライブを行えるようにするために、以下のように MQSC コマンドを発行してアクセス権限を付与します。

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

説明:

non-privileged-user-id は、ステップ 1073 ページの『1』 で取得したユーザー ID です。
これにより、トピック・ツリー内のあらゆるトピックへのアクセス権限が *non-privileged-user-id* に付与されます。または、**DEFINE TOPIC** を使用してトピック・オブジェクトを定義し、そのトピック・オブジェクトにより参照されるトピック・ツリーの一部のみへのアクセス権限を付与することもできます。 詳細については、トピックへのユーザー・アクセスの制御を参照してください。

次のタスク

これで、クライアント・アプリケーションはキュー・マネージャーに接続し、キューを使用してメッセージの書き込みや読み取りができるようになりました。

関連概念

ALW AIX, Linux, and Windows 上の IBM MQ オブジェクトへのアクセス権限の付与

関連資料

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC
IBM i IBM i 上の IBM MQ 権限

IBM i *IBM i* でのサンプル・プログラムの作成と実行

IBM i でサンプル・プログラムを実行する前に、あらかじめキュー・マネージャーを作成する必要があります。また、必要なキューも作成しなければなりません。 COBOL サンプルを実行する場合には、追加の準備が必要になることがあります。

このタスクについて

IBM MQ for IBM i のサンプル・プログラムのソースは、QCSRC、QCLSRC、QCBLLESRC、および QRPGLESRC のメンバーとしてライブラリー QMQMSAMP に提供されています。

サンプルを実行するときに、自分自身のキューを使用するか、あるいはサンプル・プログラム AMQSAMP4 を実行して、いくつかのサンプル・キューを作成することができます。 AMQSAMP4 プログラムのソースは、ライブラリー QMQMSAMP のファイル QCLSRC に入っています。 それを、CRTCLPGM コマンドを使用してコンパイルできます。

サンプルを実行するには、ライブラリー QMQM で提供される C の実行可能バージョンを使用するか、他の IBM MQ アプリケーションと同様にコンパイルします。

手順

1. キュー・マネージャーを作成し、デフォルトの定義をセットアップします。
   サンプル・プログラムを実行するには、その前にこの操作を行う必要があります。 キュー・マネージャーの作成について詳しくは、IBM MQ の管理を参照してください。 クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるようにキュー・マネージャーを構成する方法について詳しくは、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』 を参照してください。

2. ライブラリー QMQMSAMP のファイル AMQSDATA 内のメンバー PUT からのデータを使用してサンプル・プログラムの 1 つを呼び出すには、以下のようなコマンドを使用します。

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

注 : コンパイル済みのモジュールで IFS ファイル・システムを使用するには、CRTCMOD にオプション SYSIFCOPT (*IFSIO) を指定し、パラメーターとして渡されるファイル名を以下の形式で指定しなければなりません。

```
home/me/myfile
```

3. COBOL バージョンの Inquire、Set、Echo の例を使用する場合、サンプルを実行する前にプロセス定義を変更してください。

   照会、設定、およびエコーのサンプルでは、サンプル定義によってこれらのサンプルの C バージョンが起動されます。 COBOL バージョンを使用したい場合は、以下のプロセス定義を変更する必要があります。

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   IBM i では、**CHGMQMPRC** コマンド (詳細については MQ プロセスの変更 (CHGMQMPRC) を参照) を使用するか、あるいは **AMQSAMP4** コマンドを代わりの定義で編集、実行します。

4. サンプル・プログラムを実行します。

   各サンプルで必要なパラメーターについて詳しくは、それぞれのサンプルの説明を参照してください。

   注 : COBOL サンプル・プログラムでは、キュー名をパラメーターとして渡す場合、必要に応じてブランク文字を埋め、合計で 48 文字にしなければなりません。 48 文字以外の場合には、理由コード 2085 でプログラムが異常終了します。

関連資料

1070 ページの『IBM i のサンプル』
IBM i システム上の IBM MQ 用のサンプル・プログラムによって示される技法。

Linux AIX *AIX and Linux* でのサンプル・プログラムの作成と実行

AIX and Linux でサンプル・プログラムを実行する前に、あらかじめキュー・マネージャーを作成する必要があります。また、必要なキューも作成しなければなりません。 COBOL サンプルを実行する場合には、追加の準備が必要になることがあります。

このタスクについて

インストール時にデフォルト値を使用した場合、IBM MQ (AIX and Linux システム用) のサンプル・ファイルは、1076 ページの表 160 に記載されているディレクトリーに格納されます。

表 *160. IBM MQ (AIX and Linux* システム用*)* のサンプルの格納場所

内容	**ディレクトリー**
ソース・ファイル	*MQ_INSTALLATION_PATH*/samp
送達不能キュー・ハンドラーのソース・ファイル	*MQ_INSTALLATION_PATH*/samp/dlq
実行可能ファイル	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

サンプルで作業を行うために一連のキューが必要となります。 独自のキューを使用するか、またはサンプル MQSC ファイル amqscos0.tst を実行してキューを 1 セット作成します。 このサンプルを実行するに

は、提供されている実行可能バージョンを使用するか、あるいは ANSI コンパイラーを使用して他のアプリケーションの場合と同様にソース・バージョンをコンパイルします。

手順

1. キュー・マネージャーを作成し、デフォルトの定義をセットアップします。

   サンプル・プログラムを実行するには、その前にこの操作を行う必要があります。 キュー・マネージャーの作成について詳しくは、IBM MQ の管理を参照してください。 クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるようにキュー・マネージャーを構成する方法について詳しくは、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』を参照してください。
2. 独自のキューを使用していない場合、サンプル MQSC ファイル amqscos0.tst を実行してキューを 1 セット作成します。

   AIX and Linux システムでこれを行うには、次のように入力します。

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   `sampobj.out` ファイルを調べて、エラーがないことを確認してください。
3. COBOL バージョンの Inquire、Set、Echo の例を使用する場合、サンプルを実行する前にプロセス定義を変更してください。

   照会、設定、およびエコーのサンプルでは、サンプル定義によってこれらのサンプルの C バージョンが起動されます。 COBOL バージョンを使用したい場合は、以下のプロセス定義を変更する必要があります。

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Windows では、このために `amqscos0.tst` ファイルを編集し、C 実行可能ファイル名を COBOL 実行可能ファイル名に変更してから **runmqsc** コマンドを使用してサンプルを実行します。
4. サンプル・プログラムを実行します。

   サンプルを実行するには、その名前に続けて任意のパラメーターを入力します。以下に例を示します。

   ```
   amqsput myqueue qmanagername
   ```

   この例では、*myqueue* がメッセージを入れるキューの名前で、*qmanagername* が *myqueue* を所有するキュー・マネージャーです。

   各サンプルで必要なパラメーターについて詳しくは、それぞれのサンプルの説明を参照してください。

   注 : COBOL サンプル・プログラムでは、キュー名をパラメーターとして渡す場合、必要に応じてブランク文字を埋め、合計で 48 文字にしなければなりません。 48 文字以外の場合には、理由コード 2085 でプログラムが異常終了します。

関連資料

1065 ページの『AIX and Linux システム用のサンプル』
IBM MQ for AIX or Linux のサンプル・プログラムで示される技法。

Windows *Windows* でのサンプル・プログラムの作成と実行

Windows でサンプル・プログラムを実行する前に、あらかじめキュー・マネージャーを作成する必要があります。また、必要なキューも作成しなければなりません。 COBOL サンプルを実行する場合には、追加の準備が必要になることがあります。

このタスクについて

インストール時にデフォルト値を使用した場合、IBM MQ for Windows のサンプル・ファイルは、1078 ページの表 161 に記載されているディレクトリーに格納されます。 デフォルトの場合、インストール・ドライブは <c:> です。

表 *161. IBM MQ for Windows* 用のサンプルの格納場所	
内容	**ディレクトリー**
C ソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥c¥Samples
送達不能ハンドラーのサンプルのソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥DLQ
COBOL ソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥Cobol¥Samples
C の実行可能ファイル [1]	*MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥Bin (32 ビット・バージョン) *MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥Bin64 (64 ビット・バージョン)
サンプル MQSC ファイル	*MQ_INSTALLATION_PATH*¥Tools¥MQSC¥Samples
Visual Basic ソース・コード	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET のサンプル	*MQ_INSTALLATION_PATH*¥Tools¥dotnet¥Samples

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

注 : いくつかの C 実行可能ファイル・サンプルの 64 ビット・バージョンを使用できます。

サンプルで作業を行うために一連のキューが必要となります。 独自のキューを使用するか、またはサンプル MQSC ファイル amqscos0.tst を実行してキューを 1 セット作成します。 このサンプルを実行するには、提供されている実行可能バージョンを使用するか、または IBM MQ for Windows の他のアプリケーションの場合と同様にソース・バージョンをコンパイルします。

手順

1. キュー・マネージャーを作成し、デフォルトの定義をセットアップします。

   サンプル・プログラムを実行するには、その前にこの操作を行う必要があります。 キュー・マネージャーの作成について詳しくは、IBM MQ の管理を参照してください。 クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるようにキュー・マネージャーを構成する方法について詳しくは、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』 を参照してください。
2. 独自のキューを使用していない場合、サンプル MQSC ファイル amqscos0.tst を実行してキューを 1 セット作成します。

   Windows システムでこれを行うには、次のように入力します。

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   `sampobj.out` ファイルを調べて、エラーがないことを確認してください。 このファイルは現行ディレクトリーにあります。
3. COBOL バージョンの Inquire、Set、Echo の例を使用する場合、サンプルを実行する前にプロセス定義を変更してください。

   照会、設定、およびエコーのサンプルでは、サンプル定義によってこれらのサンプルの C バージョンが起動されます。 COBOL バージョンを使用したい場合は、以下のプロセス定義を変更する必要があります。

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   Windows では、このために `amqscos0.tst` ファイルを編集し、C 実行可能ファイル名を COBOL 実行可能ファイル名に変更してから **runmqsc** コマンドを使用してサンプルを実行します。

4. サンプル・プログラムを実行します。
   サンプルを実行するには、その名前に続けて任意のパラメーターを入力します。以下に例を示します。

   ```
   amqsput myqueue qmanagername
   ```

   この例では、*myqueue* がメッセージを入れるキューの名前で、*qmanagername* が *myqueue* を所有するキュー・マネージャーです。

   各サンプルで必要なパラメーターについて詳しくは、それぞれのサンプルの説明を参照してください。

   注 : COBOL サンプル・プログラムでは、キュー名をパラメーターとして渡す場合、必要に応じてブランク文字を埋め、合計で 48 文字にしなければなりません。 48 文字以外の場合には、理由コード 2085 でプログラムが異常終了します。

関連資料

1067 ページの『IBM MQ for Windows のサンプル』
IBM MQ for Windows のサンプル・プログラムで示される技法。

1070 ページの『IBM MQ for Windows 用の Visual Basic のサンプル』
Windows システム上の IBM MQ 用のサンプル・プログラムによって示される技法。

API 出口サンプル・プログラム

サンプル API 出口は、 **MQAPI_TRACE_LOGFILE** 環境変数で定義された接頭部を持つユーザー指定ファイルに MQI トレースを生成します。

API 出口について詳しくは、958 ページの『マルチプラットフォームでの API 出口の作成とコンパイル』を参照してください。

Source
amqsaxe0.c

バイナリー
amqsaxe

サンプル出口の構成

1. qm.ini ファイルの ApiExit ローカル・スタンザ に以下の情報を追加します。

   Windows 以外のプラットフォーム

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   ここで、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーを表します。

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   ここで、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーを表します。

2. **MQAPI_TRACE_LOGFILE** 環境変数を設定します。

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. アプリケーションを実行します。

出力ファイルは、 MqiTrace.*pid*.*tid*.log のような名前で /tmp ディレクトリーに作成されます。

非同期コンシューム・サンプル・プログラム

amqscbf サンプル・プログラムは、MQCB および MQCTL を使用して、複数のキューからのメッセージを非同期的にコンシュームする方法を示します。

amqscbf は、C ソース・コードとして提供され、AIX, Linux, and Windows プラットフォーム上の クライアントおよびサーバー用のバイナリー実行可能プログラムです。

このプログラムはコマンド行から開始され、以下のオプション・パラメーターを使用します。

```
Usage: [Options] Queue Name {queue_name}
  where Options are:
    -m Queue Manager Name
    -o Open options
    -r Reconnect Type
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

複数のキューからメッセージを読み取るために、複数のキュー名を指定します (このサンプルでは、最大 10 個のキューが サポートされています。)

注 : Reconnect type は、クライアント・プログラムに対してのみ有効です。

例

以下の例は、amqscbf がサーバー・プログラムとして実行され、QL1 から 1 つのメッセージを読み取り、その後停止される様子を示しています。

IBM MQ Explorer を使用して、テスト・メッセージを QL1 に書き込みます。 Enter キーを押して、 プログラムを停止します。

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

amqscbf が示す操作

このサンプルは、複数のキューからのメッセージを到着順に読み取る方法 を示します。 この方法では、同期 MQGET を使用して、さらに多くのコードが必要となります。 非同期コンシュームの場合、ポーリングは不要であり、スレッドおよびストレージの管理は IBM MQ によって実行されます。「実環境の」例では、エラーの対処が 必要となります。このサンプルでは、エラーはコンソールに書き出されます。

サンプル・コードには以下のステップがあります。

1. 単一のメッセージ・コンシューム・コールバック関数を定義する。

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{ ... }
```

2. キュー・マネージャーに接続する。

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. 入力キューを開き、各入力キューを MessageConsumer コールバック関数と関連付ける。

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   cbd.CallbackFunction は、各キューごとに設定する必要はありません。これは入力専用フィールドです。 ただし、異なるコールバック関数を各キューに関連付けることができます。
4. メッセージのコンシュームを開始する。

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. ユーザーが Enter キーを押すまで待ってから、メッセージのコンシュームを停止する。

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最後に、キュー・マネージャーから切断する。

```
MQDISC(&Hcon,&CompCode,&Reason);
```

非同期書き込みサンプル・プログラム

amqsapt サンプルの実行および非同期書き込みサンプル・プログラムの設計について説明します。

非同期書き込みサンプル・プログラムは、非同期 MQPUT 呼び出しを使用してキューにメッセージを書き込み、次いで MQSTAT 呼び出しを使用して状況情報を取得します。 各種プラットフォームにおけるこのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

amqsapt サンプルの実行

このプログラムでは、パラメーターを以下の 6 つまで設定できます。

1. 宛先キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)
3. オープン・オプション (オプション)
4. クローズ・オプション (オプション)
5. 宛先キュー・マネージャーの名前 (オプション)
6. 動的キューの名前 (オプション)

キュー・マネージャーを指定しないと、amqsapt はデフォルトのキュー・マネージャーに接続されます。

非同期書き込みサンプル・プログラムの設計

このプログラムは、提供される出力オプション、あるいは MQOO_OUTPUT および MQOO_FAIL_IF_QUIESCING オプション付きで MQOPEN 呼び出しを使用して、メッセージを入れる宛先キューをオープンします。

キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。 プログラムを簡潔に保つには、これ以降の MQI 呼び出しで、プログラムが多数のオプションに対してデフォルト値を使用するようにします。

入力の各行ごとに、プログラムはテキストをバッファーに読み込み、MQPMO_ASYNC_RESPONSE 付きの MQPUT 呼び出しを使用します。こうして、その行のテキストが入ったデータグラム・メッセージを作成し、非同期で宛先キューに書き込みます。 プログラムは、入力が終了するか、MQPUT 呼び出しが失敗するまで処理を続行します。 プログラムは、入力が終了すると、MQCLOSE 呼び出しを使用して、キューをクローズします。

続いてプログラムは、MQSTAT 呼び出しを発行し、MQSTS 構造体が戻ります。また、正常に書き込まれたメッセージの数、警告が出されて書き込まれたメッセージの数、および失敗の回数を収めたメッセージを表示します。

ブラウズ・サンプル・プログラム

ブラウズ・サンプル・プログラムは、MQGET 呼び出しを使用してキュー上のメッセージを参照します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

ブラウズ・サンプル・プログラムの設計

このプログラムは、MQOO_BROWSE オプション付きの MQOPEN 呼び出しを使用して、宛先キューをオープンします。 キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。

キュー上のメッセージごとに、プログラムは MQGET 呼び出しを使用してキューからメッセージをコピーし、メッセージに含まれているデータを表示します。 MQGET 呼び出しでは次のオプションを使用します。

MQGMO_BROWSE_NEXT
MQOPEN 呼び出しのあと、ブラウズ・カーソルはキューの最初のメッセージの前に論理的に位置付けられるので、最初に呼び出しが行われると、このオプションによって、 **最初の** メッセージが戻されます。

MQGMO_NO_WAIT
このプログラムは、キューにメッセージがない場合は、待機しません。

MQGMO_ACCEPT_TRUNCATED_MSG
この MQGET 呼び出しでは、固定サイズのバッファーを指定します。 メッセージがこのバッファーよりも大きい場合、このプログラムは、メッセージの切り捨てが行われたことを警告すると共に、その切り捨てられたメッセージを表示します。

このプログラムは、各 MQGET 呼び出しの後に MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアする必要がある方法を示しています。これは、これらのフィールドを、呼び出しが取り出すメッセージに含まれている値に設定するためです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

このプログラムはキューの終わりまで処理を継続します。キューの終わりに到達すると、MQGET 呼び出しは、MQRC_NO_MSG_AVAILABLE 理由コードを戻し、プログラムは警告メッセージを表示します。 MQGET 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

続いて、プログラムは、MQCLOSE 呼び出しを使用してキューをクローズします。

AIX, Linux, and Windows のブラウズ・サンプル・プログラム

AIX, Linux, and Windows でのブラウズ・サンプル・プログラムについて学習する際に、このトピックを参照してください。

C バージョンのプログラムでは、次の 2 つのパラメーターをとります。

1. ソース・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、このプログラムはデフォルトのキュー・マネージャーに接続します。 例えば、それぞれ次のように入力します。

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

この例では、`myqueue` がメッセージを表示するキューの名前で、`qmanagername` が `myqueue` を所有するキュー・マネージャーとなります。

C サンプルの実行時に `qmanagername` を省略すると、デフォルトのキュー・マネージャーがそのキューを所有すると見なされます。

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target queue
```

各メッセージの最初の 50 文字のみが表示され、その場合は - - - truncated が続きます。

IBM i でのブラウズ・サンプル・プログラム
各プログラムは、呼び出されると、指定したキューのすべてのメッセージのコピーを取り出します。コピー元のメッセージはそのままキューに残ります。

提供されたキュー SYSTEM.SAMPLE.LOCAL を使用することができます。最初に書き込みサンプル・プログラムを実行して、キューにメッセージを入れます。 キュー SYSTEM.SAMPLE.ALIAS を使用することができます。これは、同一のローカル・キューの別名です。 プログラムは、キューの終わりに到達するか、MQI 呼び出しが異常終了するまで処理を継続します。

C のサンプルでは、Windows システムのサンプルと同様の方法で、通常は 2 番目のパラメーターとしてキュー・マネージャー名を指定することができます。 以下に例を示します。

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

キュー・マネージャーを指定しないと、このプログラムはデフォルトのキュー・マネージャーに接続します。 このことは、RPG のサンプルにも関係があります。 ただし、RPG サンプルでは、デフォルトを受け入れるよりもキュー・マネージャー名を指定する必要があります。

ALW ブラウザー・サンプル・プログラム

ブラウザー・サンプル・プログラムは、キュー上のすべてのメッセージのメッセージ記述子およびメッセージ内容フィールドの両方を読み取って、書き込みます。

このサンプル・プログラムは、単に技法を説明するだけではなく、ユーティリティーとして作成されています。 これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

このプログラムは、以下の定位置パラメーターを受け入れます。

1. ソース・キューの名前 (必須)
2. キュー・マネージャーの名前 (必須)
3. プロパティー用のオプション・パラメーター (オプション)

以下の環境変数を使用して、キュー・マネージャーでの認証に使用する資格情報を指定します。

MQSAP ユーザー ID
キュー・マネージャーでの認証にユーザー ID とパスワードを使用する場合は、接続認証に使用するユーザー ID に設定します。 プログラムは、ユーザー ID に付随するパスワードの入力を求めるプロンプトを出します。

V 9.3.4 Linux AIX **MQSAP トークン**
キュー・マネージャーで認証する認証トークンを指定する場合は、非ブランク値に設定します。 プログラムは、認証トークンの入力を求めるプロンプトを出します。 認証トークンは、クライアント・バインディングを使用する **amqsbcgc** サンプルでのみ使用できます。

これらのプログラムを実行するには、以下のいずれかのコマンドを発行します。

- amqsbcg *myqueue qmanagername*
- amqsbcgc *myqueue qmanagername*

この例では、*myqueue* がメッセージをブラウズするキューの名前で、*qmanagername* が *myqueue* を所有するキュー・マネージャーです。

このプログラムは、各メッセージをキューから読み取り、次のものを stdout に書き込みます。

- 形式化メッセージ記述子フィールド

• メッセージ・データ (16 進数でダンプされるが、可能な場合は、文字形式)

表 *162.* プロパティー・パラメーターで許容される値

値	動作
0	デフォルトの動作。 アプリケーションに送られるプロパティーは、メッセージの取得元の **PropertyControl** キュー属性に応じて異なります。

1	メッセージ・ハンドルが作成されて、MQGET と共に使用されます。 メッセージ記述子 (またはエクステンション) に含まれるプロパティーを除くメッセージ・プロパティーは、メッセージ記述子と同様の方式で表示されます。 以下に例を示します。
	`****Message properties****`
	`  property name: property value`
	または、使用可能なプロパティーが存在しない場合は、次のようになります。
	`****Message properties****`
	`  None`
	メッセージ記述子の場合と同様に、数値は `printf` を使って表示され、ストリング値は単一引用符で囲まれ、バイト・ストリングは X および単一引用符で囲まれます。

2	MQGMO_NO_PROPERTIES が指定され、メッセージ記述子プロパティーだけが戻されます。
3	MQGMO_PROPERTIES_FORCE_MQRFH2 が指定され、すべてのプロパティーがメッセージ・データ内に戻されます。
4	MQGMO_PROPERTIES_COMPATIBILITY が指定され、IBM MQ プロパティーが含まれるかどうかに応じてすべてのプロパティーを戻すことができます。含まれない場合、プロパティーは破棄されます。

プログラムは、メッセージの最初の 65535 文字に制限され、それより長いメッセージが読み取られると、理由 `truncated msg` で失敗します。

このユーティリティーからの出力の例については、キューのブラウズを参照してください。

CICS トランザクション・サンプル

サンプルの CICS トランザクション・プログラムは、ソース・コードに amqscic0.ccs、実行可能なバージョンに amqscic0 という名前が付いて提供されています。 標準の CICS 機能を使用してトランザクションを構築できます。

ご使用のプラットフォームで必要となるコマンドの詳細については、1005 ページの『プロシージャー型アプリケーションの構築』 を参照してください。

トランザクションは、伝送キュー SYSTEM.SAMPLE.CICS からメッセージを読み取ります。WORKQUEUE をデフォルト・キュー・マネージャーに指定し、それらをローカル・キューに入れます。その名前は、メッセージの伝送ヘッダーに含まれています。 障害はすべて、キュー SYSTEM.SAMPLE.CICS に送信されます。DLQ。

注 : サンプル MQSC スクリプト amqscic0.tst を使用すると、これらのキューやサンプル入力キューを作成できます。

Connect サンプル・プログラム

Connect サンプル・プログラムによって、クライアントからの MQCONNX 呼び出しとそのオプションを調べることができます。 このサンプルでは、MQCONNX 呼び出しを使用してキュー・マネージャーに接続し、MQINQ 呼び出しを使用してキュー・マネージャーの名前を照会し、それを表示します。 また、amqscnxc サンプルの実行について説明します。

注 : Connect サンプル・プログラムは、クライアント・サンプルです。 このプログラムをコンパイルしてサーバー上で実行することができますが、その機能はクライアント上でのみ意味をもち、クライアント実行可能ファイルのみが提供されます。

amqscnxc サンプルの実行

Connect サンプル・プログラムのコマンド行構文は次のようになります。

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

パラメーターはオプションであり、その順序は重要ではありません。ただし、QMgrName を指定する場合は、最後に置く必要があります。 パラメーターとして次のものがあります。

ConnName
サーバー・キュー・マネージャーの TCP/IP 接続名

TCP/IP 接続名を指定しないと、*ClientConnPtr* が NULL に設定されて MQCONNX が発行されます。

SvrconnChannelName
サーバー接続チャネルの名前

TCP/IP 接続名だけを指定し、サーバー接続チャネルを指定しないと (この反対は不可)、サンプルは SYSTEM.DEF.SVRCONN という名前を使用します。

ユーザー
接続認証に使用するユーザー名

これを指定すると、プログラムによりユーザー ID に対応するパスワードを入力するよう求められます。

QMgrName
宛先キュー・マネージャーの名前

宛先キュー・マネージャーを指定しないと、サンプルは与えられた TCP/IP 接続名を listen している側のキュー・マネージャーに接続します。

注 : パラメーターとして疑問符だけを入力するか、誤ったパラメーターを入力すると、プログラムの使用方法を説明するメッセージが表示されます。

コマンド行オプションなしでサンプルを実行すると、MQSERVER 環境変数の内容が接続情報を判別するために使用されます (この例では、MQSERVER が SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com に設定されます。) 以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

以下のように、このサンプルを実行し、TCP/IP 接続名とサーバー接続チャネル名のみを指定し、宛先キュー・マネージャー名を指定しないと、次のようになります。

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

デフォルトのキュー・マネージャー名が使用され、以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

以下のように、このサンプルを実行し、TCP/IP 接続名と宛先キュー・マネージャー名を指定すると、次のようになります。

```
amqscnxc -x machine.site.company.com MACHINE
```

以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

データ変換サンプル・プログラム

データ変換サンプル・プログラムは、データ変換出口ルーチンのスケルトンです。 データ変換サンプルの設計について説明します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

データ変換サンプルの設計

各データ変換出口ルーチンは、1 つの指定されたメッセージ形式を変換します。 このルーチンのスケルトンは、データ変換出口生成ユーティリティー・プログラムによって生成されたコードのラッパーとして設計されています。

このユーティリティーは、データ構造体ごとに 1 つのコード・フラグメントを生成します。 このような構造体がいくつか集まって 1 つの形式を形成しています。 したがって、複数のコード・フラグメントがこのスケルトンに付加されて、形式全体にわたってデータ変換を実行するためのルーチンが生成されます。

続いて、このプログラムは、変換の成否を検査し、呼び出し側に必要な値を戻します。

データベース調整サンプル

ここでは以下の 2 つのサンプルが提供されます。これらは IBM MQ が IBM MQ の更新分とデータベースの更新分の双方を、同一の作業単位内でどのように調整できるかを示します。

サンプルは、以下のとおりです。

1. AMQSXAS0 (C の場合) または AMQ0XAS0 (COBOL の場合)。これは、IBM MQ 作業単位内の単一データベースを更新します。
2. AMQSXAG0 (C の場合) または AMQ0XAG0 (COBOL の場合)、AMQSXAB0 (C の場合) または AMQ0XAB0 (COBOL の場合)、および AMQSXAF0 (C の場合) または AMQ0XAF0 (COBOL の場合)。これらは共に IBM MQ 作業単位内における 2 つのデータベースを更新し、複数のデータベースにアクセスする方法を示します。 これらのサンプルは MQBEGIN 呼び出し、混合 SQL 呼び出しおよび IBM MQ 呼び出しの使用方法、さらにデータベースに接続する場所とタイミングを示すために提供されます。

1087 ページの図 128 には、提供されたサンプルを使用してデータベースを更新する方法を示します。

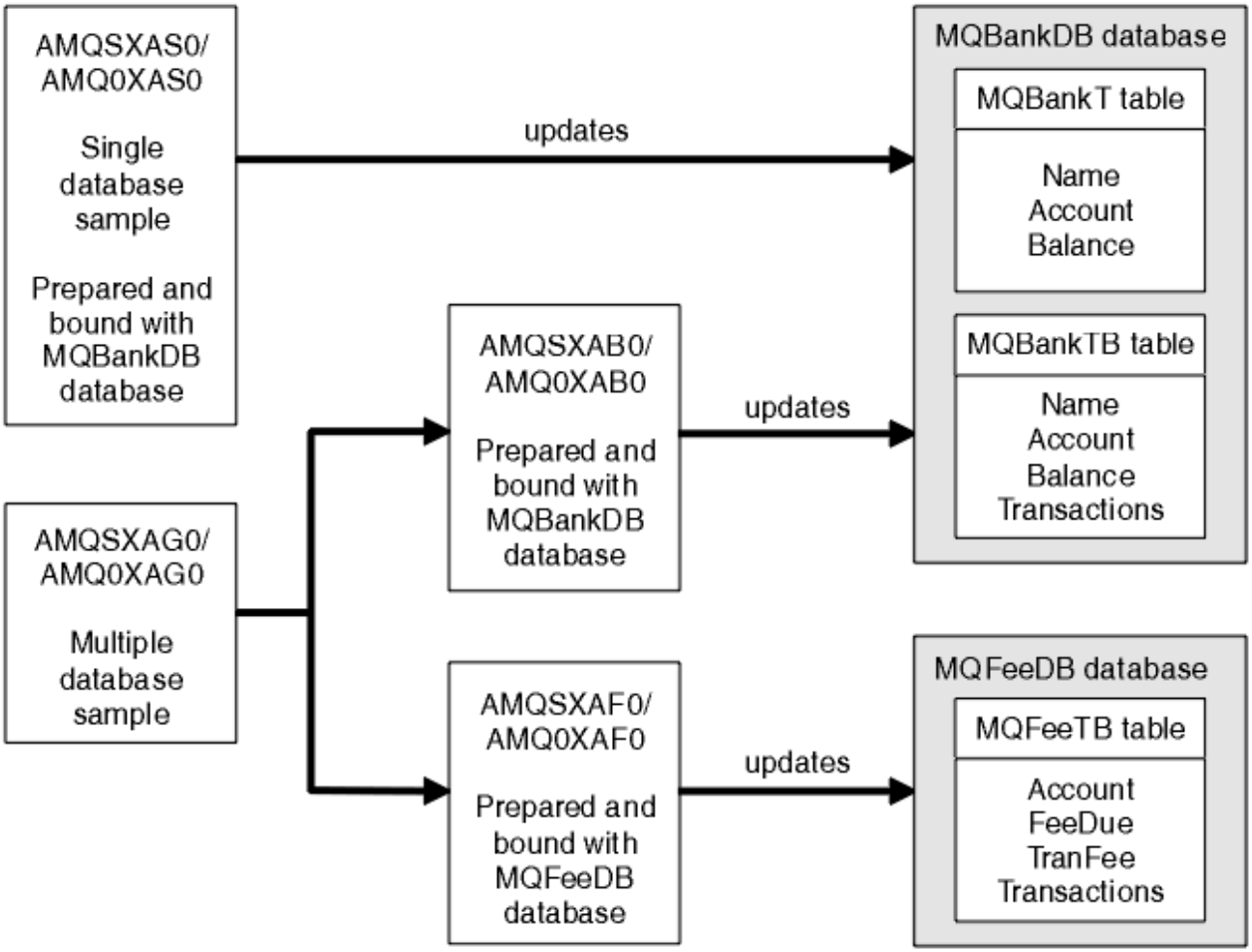


図 *128.* データベース調整サンプル

プログラムは (同期点で) キューからメッセージを読み取り、次にメッセージ内の情報を使用してデータベースから関連のある情報を得ると、これを更新します。 すると、データベースの新規状況が出力されます。

プログラム・ロジックは次のとおりです。

1. プログラム実引数から入力キューの名前を使用します。
2. MQCONN を使用して、デフォルトのキュー・マネージャー (または C で与えられる名前) に接続します。
3. 障害のない間に、MQOPEN を使用して入力のためにキューをオープンします。
4. MQBEGIN を使用して作業単位を開始します。
5. MQGET を使用して、同期点でキューから次のメッセージを読み取ります。
6. データベースから情報を読み取ります。
7. データベースの情報を更新します。
8. MQCMIT を使用して変更をコミットします。
9. 更新された情報を出力します (利用できるメッセージがない場合は障害として数え、ループを終了します)。
10. MQCLOSE を使用してキューをクローズします。
11. MQDISC を使用してキューから切断します。

サンプルは SQL カーソルを使用するので、データベースから読み取った内容 (つまり、複数インスタンス) は、メッセージが処理されている間ロックされます。したがって、これらのプログラムの複数インスタンスは同時に実行可能です。 カーソルは明示的にオープンされますが、MQCMIT 呼び出しによって暗黙的にクローズされます。

単一データベースのサンプル (AMQSXAS0 または AMQ0XAS0) には SQL CONNECT ステートメントはなく、データベースへの接続は MQBEGIN 呼び出しで IBM MQ によって暗黙的に行われます。 複数データベースのサンプル (AMQSXAG0 または AMQ0XAG0、AMQSXAB0 または AMQ0XAB0、および AMQSXAF0 または AMQ0XAF0) には SQL CONNECT ステートメントがあります。これは、データベース製品には活動状態の接続を 1 つしか許可しないものもあるからです。 上記の内容が使用するデータベース製品では問題にならない場合、または複数のデータベース製品の中の 1 つのデータベースにアクセスしている場合は、SQL CONNECT ステートメントは削除できます。

サンプルは IBM Db2 データベース製品で作成されているので、他のデータベース製品で作業するには、多少の変更が必要になる可能性があります。

SQL エラー検査は、Db2 で提供される UTIL.C および CHECKERR.CBL のルーチンを使用しています。 これらをコンパイルするか、またはコンパイルおよびリンクの前にこれらを置き換える必要があります。

注 : SQL エラー検査に Micro Focus COBOL のソース CHECKERR.MFC を使用している場合は、AMQ0XAS0 が正しくリンクできるようにプログラム ID を大文字、つまり CHECKERR に変更してください。

データベースと表の作成

サンプルをコンパイルする前に、データベースと表を作成します。

データベースを作成するには、ご使用のデータベース製品にとって標準的な方法を使用します。例えば、次のように入力します。

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

SQL ステートメントを使用して次のように表を作成します。

C の場合

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

COBOL の場合

```
EXEC SQL CREATE TABLE
  MQBankT(Name         VARCHAR(40) NOT NULL,
          Account      INTEGER     NOT NULL,
          Balance      INTEGER     NOT NULL,
          PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name         VARCHAR(40) NOT NULL,
           Account      INTEGER     NOT NULL,
           Balance      INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER   NOT NULL,
          FeeDue       INTEGER   NOT NULL,
          TranFee      INTEGER   NOT NULL,
          Transactions INTEGER,
```

```
          PRIMARY KEY (Account))
  END-EXEC.
```

SQL ステートメントを使用して次のように表にデータを入力します。

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
.
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
.
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
.
```

注 : COBOL の場合、同じ SQL ステートメントを使用しますが、各行の終わりに END_EXEC を追加します。

サンプルのプリコンパイル、コンパイル、およびリンク
C および COBOL でのサンプルのプリコンパイル、コンパイル、およびリンクについて説明します。

.SQC ファイル (C 言語) および .SQB ファイル (COBOL) をプリコンパイルし、適切なデータベースにバインドして、.C ファイルまたは .CBL ファイルを作成します。 これを行うには、ご使用のデータベース製品の標準的な方法を使用します。

C の場合のプリコンパイル

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

COBOL の場合のプリコンパイル

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

コンパイルとリンク

以下のサンプル・コマンドでは、記号 *DB2TOP* および *MQ_INSTALLATION_PATH* が使用されています。 *DB2TOP* は、Db2 製品のインストール・ディレクトリーを表します。 *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

- AIX の場合、ディレクトリー・パスは以下のとおりです。

```
/usr/lpp/db2_05_00
```

- Windows システムの場合、ディレクトリー・パスは、製品のインストール時に選択したパスによって変わります。 デフォルト設定を選択した場合のパスは以下のとおりです。

```
c:\sqllib
```

注 : Windows システムでリンク・コマンドを実行する場合は、その前に Db2 ライブラリーと IBM MQ ライブラリーへのパスが LIB 環境変数に含まれていることを確認してください。

以下のファイルを一時ディレクトリーにコピーしてください。

- IBM MQ インストール済み環境からの amqsxag0.c ファイル

  注 : このファイルは、以下のディレクトリーにあります。

  - AIX and Linux システムの場合:

    ```
    MQ_INSTALLATION_PATH/samp/xatm
    ```

  - Windows システムの場合:

    ```
    MQ_INSTALLATION_PATH\tools\c\samples\xatm
    ```

- .sqc ソース・ファイル、amqsxas0.sqc、amqsxaf0.sqc、および amqsxab0.sqc をプリコンパイルすることで取得した .c ファイル。
- ご使用の Db2 インストール済み環境のファイル util.c および util.h。

  注 : これらのファイルは、以下のディレクトリーにあります。

  ```
  DB2TOP/samples/c
  ```

お使いのプラットフォーム用の以下のコンパイラー・コマンドを使用して、各 .c ファイルのオブジェクト・ファイルを構築します。

- AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Windows システム

```
cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
FILENAME.c
```

お使いのプラットフォーム用の以下のリンク・コマンドを使用して、amqsxag0 実行可能ファイルを構築します。

- AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows Windows システム

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
/out:amqsxag0.exe
```

お使いのプラットフォーム用の以下のコンパイル・コマンドおよびリンク・コマンドを使用して、amqsxas0 実行可能ファイルを構築します。

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Windows システム

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

追加情報

AIX AIX 上で作業中に Oracle にアクセスしたい場合、xlc_r コンパイラーを使用して libmqm_r.a にリンクしてください。

サンプルの実行
この情報を使用して、C および COBOL でデータベース調整サンプルを実行する前にキュー・マネージャーを構成する方法について学習します。

サンプルを実行するには、まずキュー・マネージャーをご使用中のデータベース製品で構成します。 その方法については、シナリオ 1: キュー・マネージャーによる調整の実行を参照してください。

以下のタイトルでは、C および COBOL でのサンプルの実行方法について説明しています。

- 1091 ページの『C サンプル』
- 1092 ページの『COBOL サンプル』

C サンプル

メッセージはキューから読み取るため、次の形式をとることが必要です。

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT を使用してキューにメッセージを書き込むことができます。

データベース調整サンプルは、次の 2 つのパラメーターを使用します。

1. キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

singDBQ というキューと共に singDBQM という単一データベースのサンプル用にキュー・マネージャーを作成して構成したと想定すると、Fred Bloggs 氏のアカウントを 50 ずつ増やすには、次のようにします。

```
AMQSPUT singDBQ singDBQM
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=50 WHERE Account=1
```

キューには複数のメッセージを書き込むことができます。

```
AMQSXAS0 singDBQ singDBQM
```

そこで Fred Bloggs 氏のアカウントの更新状況が出力されます。

multDBQ というキューと共に multDBQM という複数データベースのサンプル用にキュー・マネージャーを作成して構成したと想定すると、Mary Brown さんのアカウントを 75 ずつ減らすには、次のようにします。

```
AMQSPUT multDBQ multDBQM
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=-75 WHERE Account=3
```

キューには複数のメッセージを書き込むことができます。

```
AMQSXAG0 multDBQ multDBQM
```

そこで Mary Brown さんのアカウントの更新状況が出力されます。

COBOL サンプル

メッセージはキューから読み取るため、次の形式をとることが必要です。

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

分かりやすくするため、`Balance change` は符号付きの 8 桁の数に、`Account` は 8 桁の数にしてください。

AMQSPUT のサンプルを使用してキューにメッセージを書き込むことができます。

サンプルはパラメーターを使用せず、デフォルトのキュー・マネージャーを使用します。 キュー・マネージャーは、サンプルのうち常に 1 つだけを実行できるように構成できます。 singDBQ というキューと共に単一データベースのサンプル用にデフォルトのキュー・マネージャーを構成したと想定すると、Fred Bloggs 氏のアカウントを 50 ずつ増やすには、次のようにします。

```
AMQSPUT singDBQ
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

キューには複数のメッセージを書き込むことができます。

```
AMQ0XAS0
```

キューの名前を次のとおり入力してください。

```
singDBQ
```

そこで Fred Bloggs 氏のアカウントの更新状況が出力されます。

multDBQ というキューと共に複数データベースのサンプル用にデフォルトのキュー・マネージャーを構成したと想定すると、Mary Brown さんのアカウントを 75 ずつ減らすには、次のようにします。

```
AMQSPUT multDBQ
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

キューには複数のメッセージを書き込むことができます。

```
AMQ0XAG0
```

キューの名前を次のとおり入力してください。

```
multDBQ
```

そこで Mary Brown さんのアカウントの更新状況が出力されます。

送達不能キュー・ハンドラーのサンプル

サンプルの送達不能キュー・ハンドラーが提供されています。実行可能バージョンの名前は amqsdlq です。 **RUNMQDLQ** とは異なる送達不能キュー・ハンドラーが必要な場合は、サンプルのソースをベースとして使用することができます。

サンプルは、この製品内で提供される送達不能ハンドラーと同様ですが、トレースとエラー報告は異なります。 次の 2 つの環境変数が利用できます。

ODQ_TRACE
トレースをオンに切り替えるには、 YES または yes に設定します。

ODQ_MSG
エラー・メッセージおよび通知メッセージを含むファイルの名前を設定します。 提供されるファイルの名前は amqsdlq.msg です。

プラットフォームに応じ、**export** コマンドまたは **set** コマンドを使用して、これらの変数をご使用の環境で有効にする必要があります。また、**unset** コマンドを使用してトレースをオフにします。

独自の要件に合わせて、エラー・メッセージ・ファイル amqsdlq.msg を変更することができます。 このサンプルは、メッセージを IBM MQ エラー・ログ・ファイル **ではなく** stdout に書き込みます。

送達不能ハンドラーの動作および実行方法について詳しくは、ご使用のプラットフォームの「IBM MQ 送達不能キューでのメッセージの処理」または「システム管理ガイド」を参照してください。

配布リスト・サンプル・プログラム

配布リスト・サンプル amqsptl0 によって、複数のメッセージ・キューにメッセージを書き込む例が得られます。 これは MQPUT サンプルである amqsput0 に基づいています。

配布リスト・サンプル amqsptl0 の実行

配布リスト・サンプルは、書き込みサンプルと同様の方法で実行されます。

これは以下のパラメーターをとります。

- キューの名前
- キュー・マネージャーの名前

これらの値は組にして入力します。 以下に例を示します。

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

キューは MQOPEN によってオープンされ、メッセージは MQPUT によってキューに書き込まれます。 キュー名またはキュー・マネージャー名が認識されない場合は、理由コードが戻ります。

メッセージがキュー・マネージャー間を流れるように、チャネルを定義してください。 サンプル・プログラムでは、チャネル定義まで行いません。

配布リスト・サンプルの設計

書き込みメッセージ・レコード (MQPMR) は、宛先ごとにメッセージの属性を指定します。 サンプルが *MsgId* および *CorrelId* に値を設定すると、これらは MQMD 構造体で指定された値を指定変更します。

MQPMO 構造体の *PutMsgRecFields* フィールドは、次のように MQPMR にあるフィールドを示します。

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

次に、サンプルは応答レコードとオブジェクト・レコードを割り当てます。 オブジェクト・レコード (MQOR) には、1 組以上の名前かつ偶数の名前が必要です。つまり、*ObjectName* と *ObjectQMgrName* です。

次の段階では、MQCONN を使用したキュー・マネージャーへの接続が必要になります。 サンプルは MQOR 内の最初のキューに関連するキュー・マネージャーへの接続を試みます。これが失敗すると、サンプルはオブジェクト・レコードをくまなく調べます。 キュー・マネージャーに接続できない場合はその旨を通知し、このサンプル・プログラムは実行を終了します。

宛先キューは MQOPEN によってオープンされ、メッセージは MQPUT によってこれらのキューに書き込まれます。 問題や障害が発生した場合は、応答レコード (MQRR) に報告されます。

最後に、宛先キューは MQCLOSE によってクローズされ、プログラムは MQDISC によってキュー・マネージャーから切断されます。 *CompCode* および *Reason* を示す呼び出しごとに、同じ応答レコードが使用されます。

エコー・サンプル・プログラム

Echo サンプル・プログラムは、メッセージ・キューから応答キューへのメッセージをエコー出力します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

これらのプログラムは、起動されたプログラムとして実行することを意図したものです。

IBM i、AIX, Linux, and Windows・システムでは、その入力は、宛先キューおよびそのキュー・マネージャーの名前を含む MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

IBM i IBM i でトリガー・プロセスを機能させるには、使用したいエコー・サンプル・プログラムが、キュー SYSTEM.SAMPLE.ECHO これを行うには、使用したいエコー・サンプル・プログラムの名前を、プロセス定義 SYSTEM.SAMPLE.ECHOPROCESS の *ApplId* フィールドに指定します。 (このために、CHGMQMPRC コマンドを使用できます。詳細については、MQ プロセスの変更 (CHGMQMPRC) を参照してください。) サンプル・キューのトリガー・タイプは FIRST です。このため、要求サンプルを実行する前にメッセージが既にキュー上にある場合、エコー・サンプルは送信したメッセージによって起動されません。

正しく定義を設定したら、まず、1 つのジョブで AMQSERV4 を始動し、続いて別のジョブで AMQSREQ4 を始動します。 AMQSERV4 の代わりに AMQSTRG4 を使用しても構いませんが、ジョブの送信が遅れる可能性があるため、処理の流れを追うのが難しくなります。

SYSTEM.SAMPLE.ECHO キューにメッセージを送信するには、要求サンプル・プログラムを使用してください。 エコー・サンプル・プログラムは、要求メッセージ内のデータを含む応答メッセージを、要求メッセージで指定した応答先キューに送信します。

エコー・サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、要求キューと呼びます)。 このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG、MQGMO_CONVERT、および MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求

メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各入力行ごとに、プログラムはテキストをバッファーに読み込み、MQPUT1 呼び出しを使用して、その行のテキストが入った要求メッセージを応答先キューに入れます。

MQGET 呼び出しが失敗すると、このプログラムは、メッセージ記述子の *Feedback* フィールドに、MQGET によって戻される理由コードを設定して、報告メッセージを応答先キューに入れます。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

IBM i IBM i では、このプログラムは、IBM MQ for IBM i 以外のプラットフォームからキューに送信されたメッセージにも応答できますが、この状況においてサンプルは提供されません。 ECHO プログラムを作動させるには、次のようにします。

- **Format**、**Encoding**、および **CCSID** パラメーターを正しく指定して、プログラムを作成し、テキスト要求メッセージを送信します。

  ECHO プログラムは、必要に応じて、キュー・マネージャーにメッセージ・データの変換を行うよう要求します。
- ユーザーが作成したプログラムが応答に対して同様の変換を行わない場合には、IBM MQ for IBM i の送信チャネル上で CONVERT(*YES) と指定します。

読み取りサンプル・プログラム

読み取りサンプル・プログラムは、MQGET 呼び出しを使用してキューからメッセージを取得します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

読み取りサンプル・プログラムの設計

このプログラムは、MQOO_INPUT_AS_Q_DEF オプション付きの MQOPEN 呼び出しを使用して、宛先キューをオープンします。 プログラムは、キューをオープンできない場合に、MQOPEN 呼び出しによって戻される理由コードを含むエラー・メッセージを表示します。

キュー上の各メッセージごとに、プログラムは MQGET 呼び出しを使用して、キューからメッセージを除去し、そのメッセージに含まれるデータを表示します。 MQGET 呼び出しは、MQGMO_WAIT オプションを使用して、*WaitInterval* を 15 秒に指定します。したがって、キュー上にメッセージがないと、プログラムは指定された時間まで待機します。 メッセージがこの時間内に到着しない場合、呼び出しは失敗し、MQRC_NO_MSG_AVAILABLE 理由コードを戻します。

このプログラムは、それぞれの MQGET 呼び出しのあとに MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアすべき方法を示します。これは、この呼び出しによって、これらのフィールドに、このプログラムが取り出すメッセージに含まれる値が設定されるからです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

この MQGET 呼び出しでは、固定サイズのバッファーを指定します。 メッセージがこのバッファーよりも大きい場合には、呼び出しは失敗し、プログラムが停止します。

このプログラムは、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE 理由コードを戻すか、あるいは MQGET 呼び出しが失敗するまで、処理を続行します。 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

続いて、プログラムは、MQCLOSE 呼び出しを使用してキューをクローズします。

amqsget および *amqsgetc* のサンプルの実行

これらのプログラムはそれぞれ以下の定位置パラメーターを受け入れます。

1. ソース・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーが指定されていない場合、 **amqsget** はデフォルトのキュー・マネージャーに接続し、 **amqsgetc** は *MQSERVER* 環境変数またはクライアント・チャネル定義ファイルによって識別されるキュー・マネージャーに接続します。

3. オープン・オプション (オプション)

   オープン・オプションが指定されない場合、サンプルでは、これら 2 つのオプションの組み合わせである値 8193 が使用されます。

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING

4. クローズ・オプション (オプション)

   クローズ・オプションが指定されない場合、サンプルでは、MQCO_NONE である値 0 が使用されます。

以下の環境変数を使用して、キュー・マネージャーでの認証に使用する資格情報を指定します。

MQSAP ユーザー ID
キュー・マネージャーでの認証にユーザー ID とパスワードを使用する場合は、接続認証に使用するユーザー ID に設定します。 プログラムは、ユーザー ID に付随するパスワードの入力を求めるプロンプトを出します。

V 9.3.4 Linux AIX **MQSAP トークン**
キュー・マネージャーで認証する認証トークンを指定する場合は、非ブランク値に設定します。 プログラムは、認証トークンの入力を求めるプロンプトを出します。 認証トークンは、クライアント・バインディングを使用する **amqsgetc** サンプルでのみ使用できます。

これらのプログラムを実行するには、それぞれ次のように入力します。

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

この例では、*myqueue* がプログラムがメッセージを取得するキューの名前で、*qmanagername* が *myqueue* を所有するキュー・マネージャーとなります。

amqsget および amqsgetc の使用

amqsget は、共有メモリーを使用してキュー・マネージャーへのローカル接続を実行します。そのため、**amqsgetc** は (同じシステム上のキュー・マネージャーに接続している場合でも) クライアント・スタイルの接続を実行しますが、キュー・マネージャーが存在するシステムでのみ実行できます。

amqsgetc を使用する場合、キュー・マネージャー・ホストまたは IP アドレスとキュー・マネージャー・リスナー・ポートに関して実際にキュー・マネージャーに到達する方法についてのアプリケーションの詳細を提供する必要があります。

通常、これは、 **MQSERVER** 環境変数を使用するか、クライアント・チャネル定義テーブルを使用して接続の詳細を定義することによって行います。クライアント・チャネル定義テーブルは、環境変数を使用して **amqsgetc** に提供することもできます。例えば、 MQCCDTURL を参照してください。

MQSERVER を使用して、ポート 1414 でリスナーが実行されているキュー・マネージャーにローカルで接続し、デフォルトのサーバー接続チャネルを使用する例を以下に示します。

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

高可用性のサンプル・プログラム

amqsghac、**amqsphac**、および **amqsmhac** の各高可用性サンプル・プログラムは、自動クライアント再接続を使用して、キュー・マネージャーの障害後のリカバリーを例示します。 **amqsfhac** では、ネットワーク・ストレージを使用するキュー・マネージャーが、障害後もデータ保全性を維持していることを検査します。

amqsghac、**amqsphac**、および **amqsmhac** の各プログラムは、コマンド行から開始され、組み合わせて使用することにより、複数インスタンス・キュー・マネージャーのいずれかのインスタンスに障害が発生した後の再接続を実例によって確認できます。

また、**amqsghac**、**amqsphac**、および **amqsmhac** の各サンプルを使用して、単一インスタンス・キュー・マネージャー (通常、キュー・マネージャー・グループに構成されています) へのクライアント再接続の例を確認することもできます。

サンプルを単純化して構成しやすくするために、ここでは、開始され、停止された後に、再開される単一インスタンス・キュー・マネージャーに再接続するサンプル・プログラムが示されています。1099 ページの『キュー・マネージャーのセットアップおよび制御』を参照してください。

amqsfhac を **amqmfsck** と並行して使用して、ファイル・システムの整合性を検査します。 詳しくは、**amqmfsck** (ファイル・システム検査) と共有ファイル・システムの動作の検証を参照してください。

amqsphac *queueName* [*qMgrName*]

- **amqsphac** は IBM MQ MQI client・アプリケーションです。 これは一連のメッセージを、各メッセージ間に 2 秒の遅延時間を設定してキューに書き込み、イベント・ハンドラーに送信されたイベントを表示します。
- メッセージをキューに書き込むために同期点は使用されません。
- 同一のキュー・マネージャー・グループに属する任意のキュー・マネージャーに再接続できます。

amqsghac *queueName* [*qMgrName*]

- **amqsghac** は IBM MQ MQI client・アプリケーションです。 これはキューからメッセージを取得し、イベント・ハンドラーに送信されたイベントを表示します。
- メッセージをキューから取得するために同期点は使用されません。
- 同一のキュー・マネージャー・グループに属する任意のキュー・マネージャーに再接続できます。

amqsmhac -s *sourceQueueName* -t *targetQueueName* [-m *qMgrName*] [-w *waitInterval*]

- **amqsmhac** は IBM MQ MQI client・アプリケーションです。 これはあるキューから別のキューにメッセージをコピーします。デフォルトでは、プログラムの終了前に受け取った最後のメッセージから 15 分の待機間隔が設定されています。
- メッセージは同期点内でコピーされます。
- 再接続は、同一のキュー・マネージャーに対してのみ設定できます。

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (0 | 1 | 2)

- **amqsfhac** は IBM MQ MQI client・アプリケーションです。 これは、ネットワーク・ストレージを使用する IBM MQ 複数インスタンス・キュー・マネージャー (NAS やクラスター・ファイル・システムなど) が、データ保全性を維持していることを確認します。 「共有ファイル・システムの動作の検証」の手順に従って **amqsfhac** を実行します。
- これは、*QueueManagerName* に接続するときに、MQCNO_RECONNECT_Q_MGR オプションを使用します。 さらに、キュー・マネージャーがフェイルオーバーするときには、自動的に再接続します。
- これは *InTransactionCount***RepeatCount* 持続メッセージを *QueueName* に書き込みます。この間に、キュー・マネージャーを何度でもフェイルオーバーさせることができます。 そのたびに **amqsfhac** はキュー・マネージャーに再接続し、続行します。 テストの目的は、メッセージが失われていないことを確認することです。
- *InTransactionCount* メッセージは各トランザクション内に書き込まれます。 トランザクションは、*RepeatCount* の回数だけ繰り返されます。 トランザクション内で障害が発生した場合、 **amqsfhac** がキュー・マネージャーに再接続すると、 **amqsfhac** はロールバックし、トランザクションを再実行依頼します。
- さらにこれはメッセージを *SideQueueName* に書き込みます。 また、*SideQueueName* を使用して、すべてのメッセージが正常にコミットされているか、または *QueueName* からロールバックされているかを検査します。 不整合を検出した場合は、エラー・メッセージを書き込みます。

- 最後のパラメーターを (0|1|2) に設定することで、**amqsfhac** からの出力トレースの量を変更します。

  0
  出力量は最小です。

  1
  出力量は中程度です。

  2
  出力量は最大です。

クライアント接続の構成

これらのサンプルを実行するには、クライアントと サーバーの接続チャネルを構成する必要があります。クライアント検証プロシージャーに、クライアントのテスト環境をセットアップする方法が示されています。

別の方法として、以下の例に提供されている構成を使用することもできます。

amqsghac、amqsphac、および amqsmhac の使用例

この例では、単一インスタンス・キュー・マネージャーを使用した再接続可能クライアントの例を示します。

メッセージは **amqsphac** によってキュー SOURCE に書き込まれ、**amqsmhac** によって TARGET に転送され、**amqsghac** によって TARGET から取り出されます。1098 ページの図 129 を参照してください。

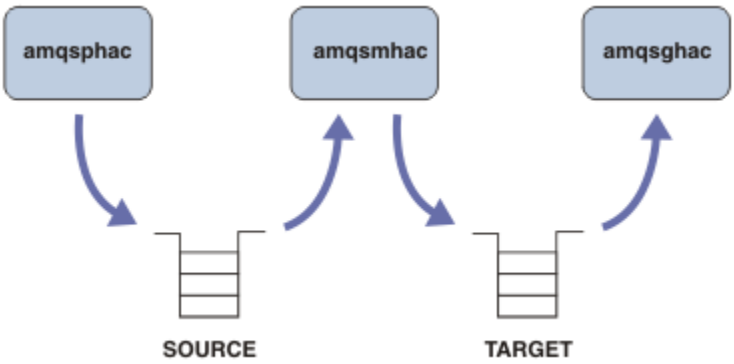


図 *129.* 再接続可能クライアントのサンプル

以下のステップに従って、このサンプルを実行してください。

1. 以下のコマンドを含む hasamples.tst ファイルを作成します。

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. コマンド・プロンプトで以下のコマンドを入力する。

   a. `crtmqm QM1`

   b. `strmqm QM1`

   c. `runmqsc QM1 < hasamples.tst`

3. 環境変数 **MQCHLLIB** を AMQCLCHL.TAB クライアント・チャネル定義ファイルへのパス (例えば、 SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc) に設定します。
4. **MQCHLLIB** を設定した 3 つの新規ウィンドウを開く。例えば Windows では、前述のコマンド・プロンプトで **start** を 3 回入力し、各ウィンドウでそれぞれプログラムを実行します。 (1099 ページの『キュー・マネージャーのセットアップおよび制御』 のステップ 1100 ページの『5』 を参照してください。)
5. コマンド endmqm -r -p QM1 を入力してキュー・マネージャーを停止してから、クライアントが再接続できるようにします。
6. コマンド strmqm QM1 を入力して、キュー・マネージャーを再始動します。

Windows で **amqsghac**、 **amqsphac**、および **amqsmhac** サンプルを実行した結果を以下の例に示します。

キュー・マネージャーのセットアップおよび制御

1. キュー・マネージャーを作成します。

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   後で **MQCHLLIB** 変数を設定するために、データ・ディレクトリーを覚えておいてください。
2. キュー・マネージャーを始動します。

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. キューとチャネルを作成し、リスナー・ポートを変更して、リスナーとチャネルを開始する。

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. クライアント・チャネル・テーブルをクライアントに認識されるようにする。

ステップ 1099 ページの『1』 の **crtmqm** コマンドから返されたデータ・ディレクトリーを使用し、そのディレクトリーに @ipcc ディレクトリーを追加して、**MQCHLLIB** 変数を設定します。

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. 他のウィンドウでサンプル・プログラムを開始する。

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. キュー・マネージャーを終了してから、再始動する。

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

関連タスク
共有ファイル・システムの動作の検証
関連資料
amqmfsck (ファイル・システム検査)

照会サンプル・プログラム

照会サンプル・プログラムでは、MQINQ 呼び出しを使用して、一部のキュー属性について照会します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

これらのプログラムは、トリガーされるプログラムとして実行することを意図したものです。したがって、IBM MQ for Multiplatforms では、その入力は MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 この構造体には、照会される属性を持つ宛先キューの名前が入っています。 C バージョンもキュー・マネージャー名を使用します。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

トリガー・プロセスを処理するには、使用したい照会サンプル・プログラムがキュー SYSTEM.SAMPLE.INQ に到着したメッセージによって起動されているか確認してください。 これを行うには、使用したい照会サンプル・プログラムの名前を、プロセス定義 SYSTEM.SAMPLE.INQPROCESS の *ApplicId* フィールドに指定します。 IBM i IBM i では、これを行うために CHGMQMPRC コマンドを使用できます。詳細については、MQ プロセスの変更 (CHGMQMPRC) を参照してください。 サンプル・キューのトリガー・タイプは FIRST です。ユーザーが要求サンプルを実行する前にメッセージが既にキュー上にある場合、照会サンプルはユーザーが送信したメッセージによって起動されません。

正しく定義を設定したら、以下のようにします。

- ALW AIX, Linux, and Windows の場合、1 つのセッションで **runmqtrm** プログラムを開始後、別のセッションで amqsreq プログラムを開始します。
- IBM i IBM i の場合、1 つのセッションで AMQSERV4 プログラムを開始後に、別のセッションで AMQSREQ4 プログラムを開始します。 AMQSERV4 の代わりに AMQSTRG4 を使用しても構いませんが、ジョブの送信が遅れる可能性があるため、処理の流れを追うのが難しくなります。

要求サンプル・プログラムを使用して、それぞれがキュー名のみを含む要求メッセージを、キュー SYSTEM.SAMPLE.INQ に送信します。 各要求メッセージごとに、照会サンプル・プログラムは、要求メッセージで指定したキューの情報を含む応答メッセージを送信します。 応答は、要求メッセージで指定した応答先キューに送信されます。

IBM i IBM i では、サンプルの入力ファイル・メンバー QMQMSAMP.AMQSDATA(INQ) を使用すると、名前を付けられた最後のキューがないので、サンプルは異常終了の理由コードと共に報告メッセージを戻します。

照会サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、このキューを要求キュー と呼びます。) このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各要求メッセージが要求キューから除去されたら、このプログラムは、そのデータに含まれるそのキュー (宛先キュー と呼びます) の名前を読み込み、MQOO_INQ オプション付きの MQOPEN 呼び出しを使用して、そのキューをオープンします。 次に、このプログラムは、MQINQ 呼び出しを使用して、宛先キューの *InhibitGet*、**CurrentQDepth**、および **OpenInputCount** 属性値を照会します。

MQINQ 呼び出しが正常に行われると、このプログラムは MQPUT1 呼び出しを使用して、応答メッセージを応答先キューに入れます。 このメッセージには、3 つの属性値が含まれます。

MQOPEN または MQINQ 呼び出しが異常終了すると、このプログラムは MQPUT1 呼び出しを使用して、報告メッセージを応答先キューに入れます。 この報告メッセージのメッセージ記述子の *Feedback* フィールドには、MQOPEN または MQINQ の失敗した呼び出しによって、戻される理由コードが入ります。

MQINQ 呼び出し後、このプログラムは MQCLOSE 呼び出しを使用して、宛先キューをクローズします。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

メッセージ処理サンプル・プログラムの照会プロパティー

AMQSIQMA は、メッセージ・ハンドルのプロパティーをメッセージ・キューから照会するためのサンプル C プログラムであり、MQINQMP API 呼び出しの使用例です。

このサンプルでは、メッセージ・ハンドルが作成され、それが MQGMO 構造体の MsgHandle フィールドに書き込まれます。 次に、1 つのメッセージが読み取られ、そのメッセージ・ハンドルに設定されているすべてのプロパティーが照会および表示されます。

```
C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

パブリッシュ/サブスクライブのサンプル・プログラム

パブリッシュ/サブスクライブのサンプル・プログラムでは、IBM MQ のパブリッシュ機能およびサブスクライブ機能の使用法を示します。

IBM MQ パブリッシュ/サブスクライブ・インターフェースに対してプログラムを作成する方法を示す C 言語のサンプル・プログラムが 3 つあります。 また、古いインターフェースを使用する C のサンプルのほか、Java のサンプルもあります。 Java のサンプルでは、com.ibm.mq.jar にある IBM MQ パブリッシュ/サブスクライブ・インターフェースと、com.ibm.mqjms にある JMS パブリッシュ/サブスクライブ・インターフェースを使用しています。 JMS のサンプルについては、このトピックでは説明しません。

C

C サンプル・フォルダー内のパブリッシャー・サンプル amqspub を検索します。 最初のパラメーターとして任意のトピック名を指定し、2 番目のパラメーターとしてキュー・マネージャーの名前 (省略可能) を指定して、このサンプルを実行します。 例えば、amqspub mytopic QM3 と入力します。 また、amqspubc というクライアント・バージョンもあります。 このクライアント・バージョンを実行する場合は、まず、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』で詳細を確認してください。

パブリッシャーはデフォルトのキュー・マネージャーに接続し、「target topic is mytopic」という応答を出力します。 これ以降、このウィンドウに入力する行は、mytopic にパブリッシュされます。

同じディレクトリー内の別のコマンド・ウィンドウを開き、サブスクライバー・プログラム amqssub を実行し、同じトピック名とオプションのキュー・マネージャー名を指定して、それを実行します。 例えば、amqssub mytopic QM3 と入力します。

サブスクライバーからの応答として「Calling MQGET : 30 seconds wait time」が出力されます。 これ以降、パブリッシャーに入力する行は、サブスクライバーに出力として表示されます。

もう 1 つ別のコマンド・ウィンドウでもう 1 つ別のサブスクライバーを始動し、2 つのサブスクライバーがパブリケーションを受け取る様子を見てください。

パラメーターに関する完全な説明とオプションの設定に関する情報は、サンプルのソース・コードを参照してください。 サブスクライバーのオプション・フィールドの値については、トピック Options (MQLONG) に説明があります。

別のサブスクライバー・サンプル amqssbx があり、コマンド行スイッチとして追加のサブスクリプション・オプションを提供しています。

amqssbx -d mysub -t mytopic -k と入力してサブスクライバーを起動すると、サブスクライバーが終了した後も保持される永続サブスクリプションを使用できます。

サブスクリプションをテストするために、パブリッシャーを使用して別のアイテムをパブリッシュしてみてください。 30 秒間待機して、サブスクライバーが終了するのを待ちます。 同じトピックでさらにいくつかアイテムをパブリッシュします。 サブスクライバーを再始動します。 サブスクライバーが実行中でなかった間にパブリッシュされた最終アイテムが、サブスクライバーが再始動した後すぐに表示されます。

C レガシー

キューに入れられたコマンドについて例を示す C のサンプルのセットが他にいくつかあります。 それらのサンプルのうちいくつかは、当初は MQ0C Supportpac の一部として提供されていました。 これらのサンプルで示されている機能は、互換性の目的で完全にサポートされています。

キューに入れられたコマンドのインターフェースを使用することはお勧めできません。 パブリッシュ/サブスクライブの API よりずっと複雑で、キューに入れられたコマンドの複雑なプログラミングを行うことに見合うだけの機能上のメリットもありません。 しかし、キューに入れられたコマンドの方式が適していると考えられるケースとして、そのインターフェースを既に使用している場合や、従来と異なる MQSUB 呼び出しを作成するよりも、複雑なメッセージを作成して汎用的な MQPUT を呼び出す方が容易であるようなプログラミング環境を使用している場合があります。

追加のサンプルは、samples フォルダー内の pubsub サブディレクトリーにあります。

サンプルには、1103 ページの表 163 にリストされている 6 つのタイプがあります。

表 *163.* パブリッシュ/サブスクライブに関する既存のサンプル *C* プログラムのカテゴリー

カテゴリー	**プログラム**	**コメント**
RFH1	amqssr1a.c amqspr1a.c	RFH1 形式のメッセージを使用して作成した、単純なパブリッシュ/サブスクライブの例。
RFH2	amqssr2a.c amqspr2a.c	RFH2 形式のメッセージを使用して作成した、単純なパブリッシュ/サブスクライブの例。
MQAI サンプル	amqsppca.c amqsspca.c	PCF コマンドと MQAI コマンド・インターフェースを使用して作成した、単純なパブリッシュ/サブスクライブの例。
RFH1 を使用する MA0C 結果サービス	amqsgama.c amqsresa.c	RFH1 ヘッダーを使用して作成した結果サービス 1. amqsgama.tst および amqsresa.tst で定義されているキューが必要です 2. amqsresa は amqsgama より前に開始される必要があります
RFH2 を使用する MA0C 結果サービス	amqsgr2a.c amqsrr2a.c	RFH2 ヘッダーを使用して作成した結果サービス 1. amqsgama.tst および amqsresa.tst で定義されているキューが必要です 2. amqsresa は amqsgama より前に開始される必要があります
ルーティング出口によるパブリッシュ/サブスクライブの例	amqspsra.c	パブリッシュ/サブスクライブするメッセージについて、ルーティング出口でキューまたはキュー・マネージャーの宛先を変更する方法を示しています。

Java 用のサンプル・プログラム

Java サンプル MQPubSubApiSample.java は、パブリッシャーとサブスクライバーを 1 つのプログラムで結合します。 そのソースおよびコンパイル済みクラス・ファイルは、wmqjava サンプル・フォルダーにあります。

クライアント・モードで実行する場合は、まず、1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』で詳細を確認してください。

構成済みの Java 環境がある場合は、Java コマンドを使用してコマンド・ラインからサンプルを実行します。 Java プログラミング・ワークベンチが既にセットアップされている IBM MQ Explorer Eclipse ワークスペースからサンプルを実行することもできます。

サンプル・プログラムを実行するために、プログラムのプロパティーの一部に変更が必要な場合があります。 それには、JVM に対してパラメーターを指定したり、ソースを編集したりします。

1104 ページの『MQPubSubApiSample Java サンプルの実行』の 説明に、Eclipse ワークスペースからのサンプル実行方法が記述されています。

MQPubSubApiSample Java サンプルの実行
Eclipse プラットフォームの Java 開発ツールを使用して MQPubSubApiSample を実行する方法。

始める前に

Eclipse ワークベンチを開きます。 新しいワークスペース・ディレクトリーを作成し、 それを選択します。 ウェルカム・ウィンドウをクローズします。

1073 ページの『クライアント接続を受け入れるようにキュー・マネージャーを構成する (Multiplatforms)』のステップを行ってから、クライアントとして実行します。

このタスクについて

パブリッシュ/サブスクライブの Java サンプル・プログラムは、IBM MQ MQI client Java プログラムです。 このサンプルは、ポート 1414 で listen するデフォルトのキュー・マネージャー を使用して、変更なしで実行します。 このタスクでは、この単純なケースを説明し、さらに、このサンプルをさまざまな異なる IBM MQ 構成に適合させるには、パラメーターをどのように指定し、サンプルをどのように変更すればいいのかを、一般用語を使って説明します。 例では、Windows での実行を示します。 他のプラットフォームの場合はファイル・パスが 異なります。

手順

1. Java サンプル・プログラムをインポートします
   a) ワークベンチで、 **ウィンドウ** > 「**パースペクティブを開く**」 > 「**その他**」 > **Java** をクリックし、「**OK**」をクリックします。
   b)「**パッケージ・エクスプローラー**」ビューに切り替えます。
   c)「**パッケージ・エクスプローラー** 」ビューの空白で右クリックします。 「**新規**」 > 「**Java プロジェクト**」をクリックします。
   d) **Project name** フィールドに、 MQ Java Samples と入力します。 「**次へ**」をクリックします。
   e)「 **Java Settings** 」パネルで、 「**ライブラリー**」 タブに切り替えます。
   f)「**外部 JAR の追加**」をクリックします。
   g) *MQ_INSTALLATION_PATH* が IBM MQ インストール・フォルダーである *MQ_INSTALLATION_PATH* \java\lib を参照して、com.ibm.mq.jar および com.ibm.mq.jmqi.jar を選択します。
   h)「**オープン**」 > 「**終了**」をクリックします。
   i)「**パッケージ・エクスプローラー**」ビューで src を右クリックします。
   j) 選択 **インポート ...** > **一般** > **ファイル・システム** > **次へ** > **参照**... 次に、パス *MQ_INSTALLATION_PATH* \tools\wmqjava\samples を参照します。ここで、 *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。

k)「**インポート**」パネル(1105 ページの図 130) で samples をクリックします(チェック・ボックスは選択しません)。

l) MQPubSubApiSample.java を選択します。 **Into folder** フィールドには MQ Java Samples/src が含まれている必要があります。 「**終了**」をクリックします。

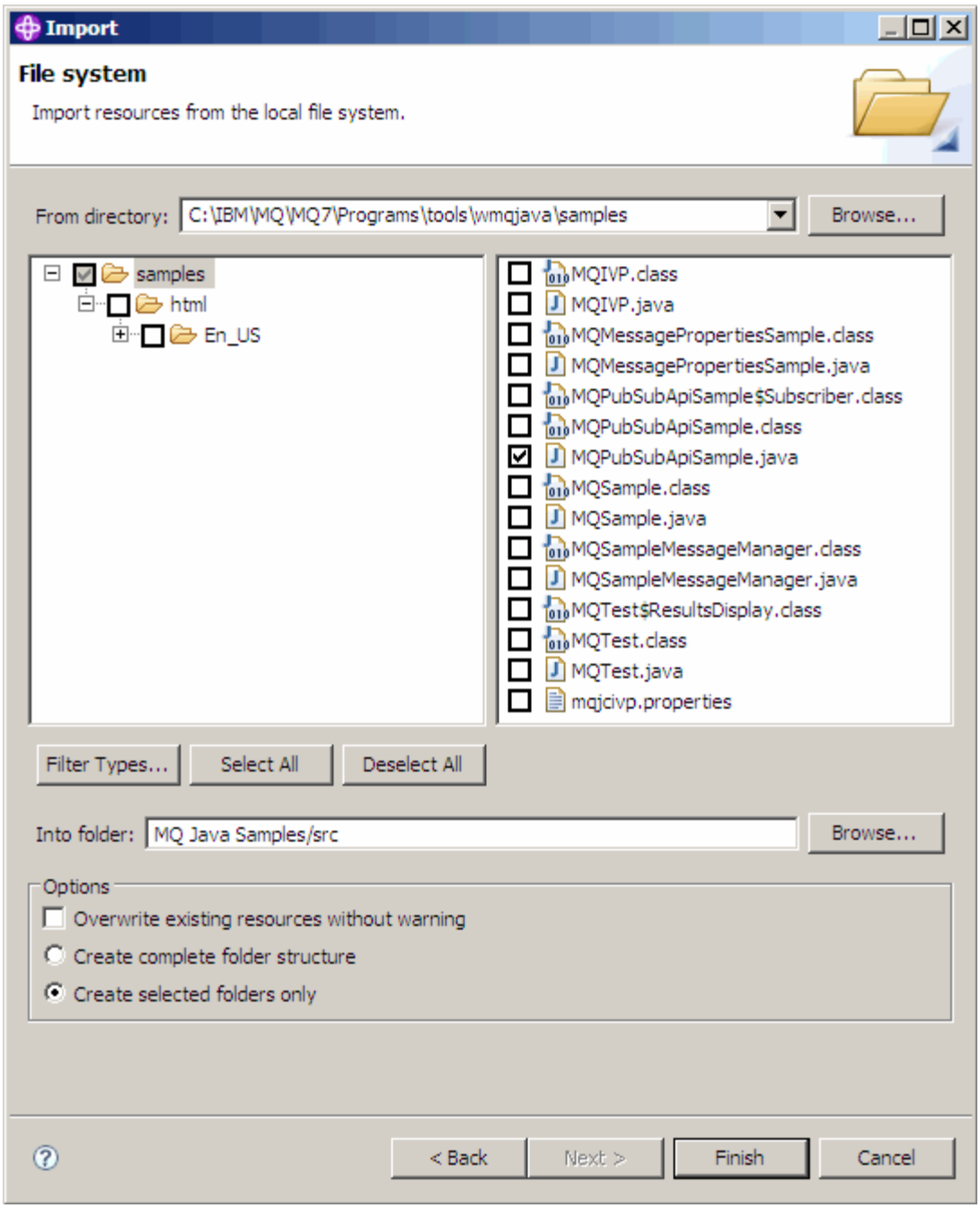


図 *130.* ファイル・システムのインポート

2. パブリッシュ/サブスクライブのサンプル・プログラムを実行します。

このプログラムを実行する方法には、デフォルトのパラメーターを 変更する必要があるかどうかによって、2 とおりの方法があります。

- 最初の選択肢は、変更を加えずにプログラムを実行することです。
  - ワークスペースのメイン・メニューで、src フォルダーを展開します。 「**MQPubSubApiSample.java**」をクリックして、1 として**実行します** > 。 **Java アプリケーション**

- 2 番目の選択肢は、パラメーター付きでプログラムを実行するか、または使用している環境に応じてソース・コードを変更してプログラムを実行することです。
  - MQPubSubApiSample.java を開き、MQPubSubApiSample コンストラクターを調べます。
  - プログラムの属性を変更します。

    これらの属性は、-D JVM スイッチを使用するか、ソース・コードを編集することによって System p プロパティーを提供することによって変更できます。

    - topicObject
    - queueManagerName
    - subscriberCount

    以下の属性は、コンストラクターでのソース・コードの編集によってのみ 変更できます。

    - ホスト名
    - port
    - channel

    System p プロパティーを設定するには、アクセサーにデフォルト値をコーディングします。例えば、次のようにします。

    ```
    queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
    "QM3");
    ```

    あるいは、 以下の手順のように、-D オプションを使用して JVM にパラメーターを 指定します。

  a. 設定したい System.Property のフルネームをコピーします。 例えば、com.ibm.mq.pubSubSample.queueManagerName です。
  b. ワークスペースで、「**実行**」 >「**実行ダイアログを開く**」を右クリックします。「**アプリケーションの作成、管理、および実行**」 で Java「アプリケーション」をダブルクリックし、「**(x) = 引数**」 タブをクリックします。
  c.「**VM 引数:**」 ペインで、 -D と入力し、 System.property 名 com.ibm.mq.pubSubSample.queueManagerName に続けて =QM3 を貼り付けます。 「**適用**」 >「**実行**」をクリックします。
  d. コンマ区切りリストとして、またはペイン内の追加行 (コンマ区切りなし) として、 さらに引数を追加します。

     例えば、-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6。

パブリッシュ出口サンプル・プログラム

AMQSPSE0 は、サブスクライバーに送信される前にパブリケーションをインターセプトする出口のサンプル C プログラムです。 これにより、出口は、例えば、メッセージ・ヘッダー、ペイロードまたは宛先を変更したり、メッセージがサブスクライバーにパブリッシュされないようにすることができます。

サンプルを実行するには、以下のタスクを実行します。

1. キュー・マネージャーを以下のように構成します。

- Linux AIX AIX and Linux システムの場合、以下のようなスタンザを qm.ini ファイルに追加します。

  ```
  PublishSubscribe:
  PublishExitPath=Module
  PublishExitFunction=EntryPoint
  ```

  ここで、モジュールは *MQ_INSTALLATION_PATH*/samp/bin/amqspse です。
  MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

- Windows Windows の場合、これに相当する属性をレジストリーに設定します。

2. モジュールが IBM MQ にアクセス可能であること確認します。
3. キュー・マネージャーを再始動して、構成を取り入れます。
4. トレースされるアプリケーション・プロセスで、トレース・ファイルが書き込まれる場所を記述します。 以下に例を示します。

   - Linux AIX AIX and Linux システムでは、ディレクトリー `/var/mqm/trace` が存在することを確認し、 **MQPSE_TRACE_LOGFILE** 環境変数をエクスポートします。

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows Windows の場合は、ディレクトリー `C:\temp` が存在することを確認し、 **MQPSE_TRACE_LOGFILE** 環境変数を設定します。

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

書き込みサンプル・プログラム

書き込みサンプル・プログラムは、MQPUT 呼び出しを使用して、メッセージをキューに入れます。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

書き込みサンプル・プログラムの設計

このプログラムは、MQOO_OUTPUT オプション付きの MQOPEN 呼び出しを使用して、メッセージを入れる宛先キューをオープンします。

キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。 プログラムを簡潔に保つには、これ以降の MQI 呼び出しで、プログラムが多数のオプションに対してデフォルト値を使用するようにします。

各入力行ごとに、プログラムはテキストをバッファーに読み込み、MQPUT 呼び出しを使用して、その行のテキストが入ったデータグラム・メッセージを作成します。 プログラムは、入力が終了するか、MQPUT 呼び出しが異常終了するまで処理を続行します。 プログラムは、入力が終了すると、MQCLOSE 呼び出しを使用して、キューをクローズします。

書き込みサンプル・プログラムの実行

amqsput および amqsputc のサンプルの実行

ALW

amqqsput サンプルは、ローカル・バインディングを使用してメッセージを書き込むためのプログラムであり、 **amqsputc** サンプルは、クライアント・バインディングを使用してメッセージを書き込むためのプログラムです。 これらのプログラムはそれぞれ以下の定位置パラメーターを受け入れます。

1. 宛先キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

   キュー・マネージャーが指定されていない場合、 **amqsput** はデフォルトのキュー・マネージャーに接続し、 **amqsputc** は *MQSERVER* 環境変数またはクライアント・チャネル定義ファイルによって識別されるキュー・マネージャーに接続します。
3. オープン・オプション (オプション)

   オープン・オプションが指定されない場合、サンプルでは、これら 2 つのオプションの組み合わせである値 8208 が使用されます。

   - MQOO_OUTPUT

- MQOO_FAIL_IF_QUIESCING

4. クローズ・オプション (オプション)

   クローズ・オプションが指定されない場合、サンプルでは、MQCO_NONE である値 0 が使用されます。

5. 宛先キュー・マネージャーの名前 (オプション)

   宛先キュー・マネージャーが指定されない場合、MQOD の ObjectQMgrName フィールドはブランクのままになります。

6. 動的キューの名前 (オプション)

   動的キューの名前が指定されない場合、MQOD の DynamicQName フィールドはブランクのままになります。

以下の環境変数を使用して、キュー・マネージャーでの認証に使用する資格情報を指定します。

MQSAP ユーザー ID
キュー・マネージャーでの認証にユーザー ID とパスワードを使用する場合は、接続認証に使用するユーザー ID に設定します。 プログラムは、ユーザー ID に付随するパスワードの入力を求めるプロンプトを出します。

V 9.3.4 Linux AIX **MQSAP トークン**
キュー・マネージャーで認証する認証トークンを指定する場合は、非ブランク値に設定します。 プログラムは、認証トークンの入力を求めるプロンプトを出します。 認証トークンは、クライアント・バインディングを使用する **amqsputc** サンプルでのみ使用できます。

これらのプログラムを実行するには、以下のいずれかのコマンドを発行します。

- amqsput *myqueue qmanagername*
- amqsputc *myqueue qmanagername*

この例では、*myqueue* がメッセージを入れるキューの名前で、*qmanagername* が *myqueue* を所有するキュー・マネージャーです。

amq0put サンプルの実行

ALW

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target queue
```

プログラムは、StdIn からの入力を受け付け、各入力行を宛先キューに付加します。 ブランク行は、これ以上データがないことを示します。

AMQSPUT4 C サンプルの実行 (IBM i)

IBM i

C プログラム AMQSPUT4 (IBM i プラットフォームでのみ使用可能) は、ソース・ファイルのメンバーからデータを読み込んで、メッセージを作成します。

プログラムを始動する時に、ファイルの名前をパラメーターとして指定する必要があります。 ファイルの構造体は、次のように指定する必要があります。

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

書き込みサンプルの入力サンプルは、ライブラリー QMQMSAMP のファイル AMQSDATA 内のメンバー PUT で提供されます。

注 : キュー名の場合、大文字と小文字の区別があることに注意してください。 サンプル・ファイル作成プログラム AMQSAMP4 によって作成されたキューの名前はすべて、大文字で作成されます。

上記の C プログラムは、メッセージをファイルの最初の行で名前が指定されたキューに入れます。提供されているキュー SYSTEM.SAMPLE.LOCAL を使用することもできます。 このプログラムは、以下に示すファイルの各行のテキストを、個別のデータグラム・メッセージに入れ、そのファイルの終わりでブランク行を読み取ると停止します。

データ・ファイルの例を使用すると、コマンドは次のようになります。

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

AMQ0PUT4 COBOL サンプルの実行 (IBM i)

IBM i

COBOL プログラム AMQ0PUT4 (IBM i プラットフォームでのみ使用可能) は、キーボードからのデータを受け入れて、メッセージを作成します。

プログラムを開始するためには、プログラムを呼び出してプログラム・パラメーターとして宛先キューの名前を指定してください。 プログラムは、キーボードから受け入れた入力をバッファーに入れ、テキストの各行ごとにデータグラム・メッセージを作成します。 キーボードでブランク行を入力するとプログラムは停止します。

参照メッセージ・サンプル・プログラム

参照メッセージ・サンプルにより、オブジェクトをソース・ノードまたは宛先ノードのいずれかで IBM MQ キューに保管する必要がなく、あるノードから別のノード (通常は異なるシステム上にあるもの) にラージ・オブジェクトを転送することができます。

参照メッセージをキューへ書き込み、メッセージ出口で受信し、キューから取り出す方法を示すために、一連のサンプル・プログラムが提供されます。 サンプル・プログラムは参照メッセージを使用してファイルを移動します。 データベースなどの他のオブジェクトを移動したい場合、またはセキュリティー検査を実行したい場合は、システムが提供するサンプルである amqsxrm に基づいて、独自の出口を定義してください。

どのバージョンの参照メッセージ出口サンプル・プログラムを使用するかは、チャネルが実行されているプラットフォームによって異なります。

- すべてのプラットフォーム上で、送信側で amqsxrma を使用します。
- 受信側が IBM i 以外のプラットフォームで実行されている場合には、受信側で amqsxrma を使用します。
- IBM i 受信側が IBM i で実行されている場合は、amqsxrm4 を使用します。

.

IBM i *IBM i* を使用する場合の注意

サンプル・メッセージ出口を使用して参照メッセージを受信するには、IFS または任意のサブディレクトリーのルート・ファイル・システムにあるファイルを指定して、ストリーム・ファイルを作成できるようにします。

IBM i 上では、サンプル・メッセージ出口はファイルを作成し、データを EBCDIC コードに変換して、コード・ページをご使用のシステム・コード・ページに設定します。 CPYFRMSTMF コマンドにより、このファイルを QSYS.LIB ファイル・システムにコピーすることができます。 以下に例を示します。

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

CPYFRMSTMF コマンドではファイルは作成されません。 このコマンドを実行する前にファイルを作成しておく必要があります。

QSYS.LIB のファイルを送信する場合、サンプルに変更を加える必要はありません。 その他のファイル・システムの場合は、MQRMH 構造体の CodedCharSetId フィールドに指定された CCSID と、送信する大量データが一致することを確認してください。

統合ファイル・システムを使用する場合は、SYSIFCOPT(*IFSIO) オプション・セット付きのプログラム・モジュールを作成してください。 データベースまたは固定長レコード・ファイルを移動したい場合は、この製品で提供されるサンプル AMQSXRM4 をもとに独自の出口を定義してください。

データベース・ファイルの推奨される転送方法は、CPYTOSTMF コマンドを使用してデータベース・ファイルを IFS 構造に変換してから、IFS ファイルを添付した参照メッセージを送信する方法です。 IFS 内部から参照することによってデータベース・ファイル を転送するが、データベース・ファイルを IFS 構造に変換しない場合は、メンバー 名を指定する必要があります。 この方法を選択した場合、データ保全性は保証されません。

Multi 参照メッセージ・サンプルの実行

この例を使用して、参照メッセージ・サンプル・アプリケーション AMQSPRM (AIX, Linux, and Windows 上の場合) または AMQSPRMA (IBM i 上の場合) の実行方法を確認します。 この例は、参照メッセージをキューに書き込み、メッセージ出口で受信し、キューから取り出す方法を示しています。

参照メッセージ・サンプルは、次のように実行します。

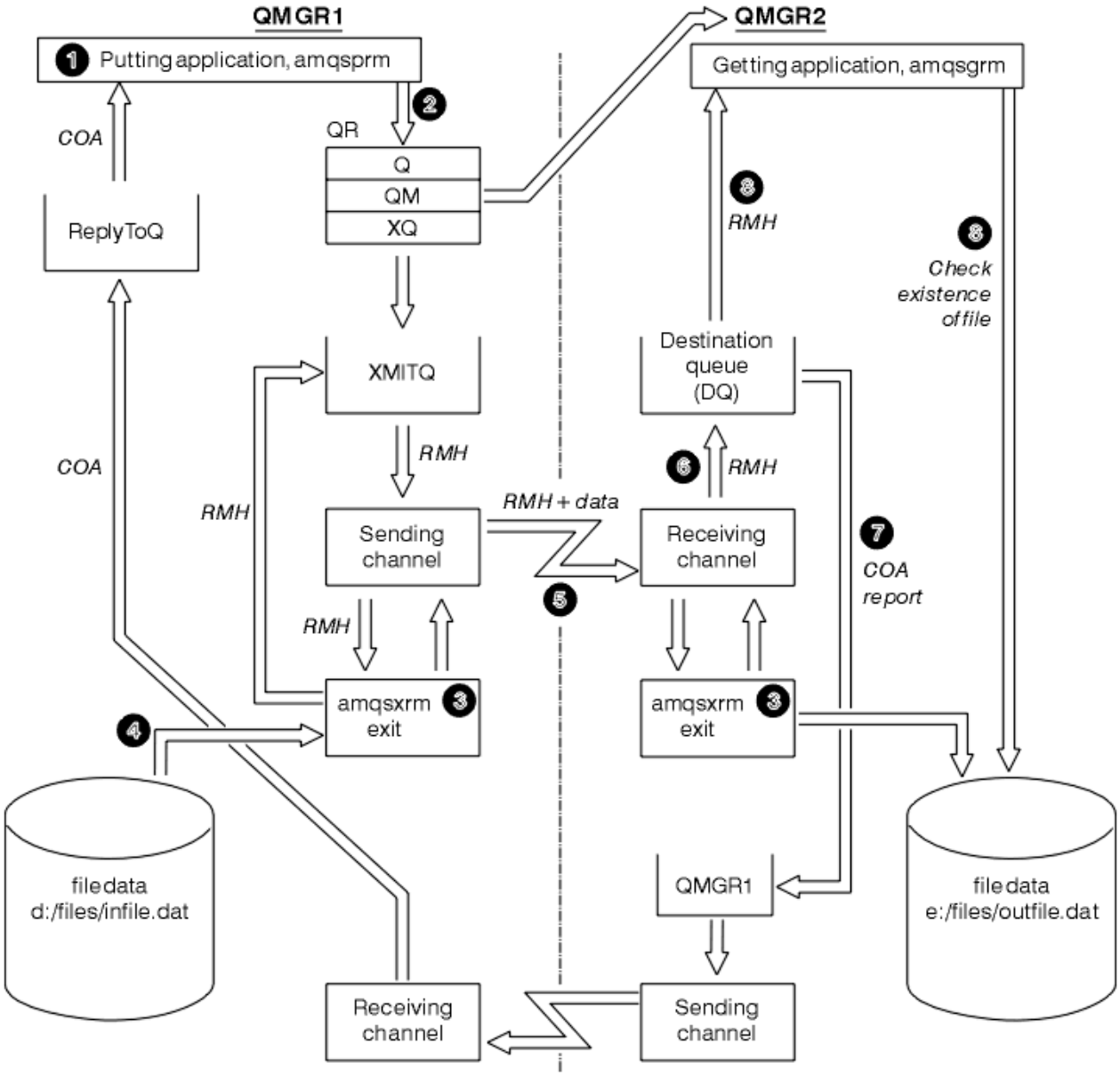


図 *131.* 参照メッセージ・サンプルの実行

1. 環境を設定してリスナー、チャネル、およびトリガー・モニターを始動し、チャネルおよびキューを定義します。

   ここでは、送信側のマシンを MACHINE1、そのマシン上のキュー・マネージャーを QMGR1 とし、受信側のマシンを MACHINE2、そのマシン上のキュー・マネージャーを QMGR2 とした場合を例に、参照メッセージの設定方法について説明します。

   注 : 以下に示す定義により、参照メッセージを作成して、オブジェクト・タイプ FLATFILE のファイルをキュー・マネージャー QMGR1 から QMGR2 へ送信し、AMQSPRM (IBM i の場合は AMQSPRMA)の呼び出しでの定義に従ってファイルを再作成することができます。 参照メッセージ (ファイル・データを含む) はチャネル CHL1 および伝送キュー XMITQ によって送信され、キュー DQ に入れられます。 例外報告および COA 報告はチャネル REPORT および伝送キュー QMGR1 によって QMGR1 に戻されます。

   参照メッセージを受信するアプリケーション (AMQSGRM または IBM i の場合は AMQSGRMA) は、開始キュー INITQ およびプロセス PROC によって起動されます。 マシンのタイプおよび IBM MQ 製品のインストール先によっては、CONNAME フィールドが正しく設定されていること、および MSGEXIT フィールドにディレクトリー構造が設定されていることを確認してください。

IBM i MQSC の定義では、出口を定義するときに AIX 形式を使用してきました。したがって、MQSC を IBM i で使用する場合は、これらの定義を適宜修正する必要があります。 メッセージ・データ FLATFILE については大文字と小文字の区別があることに注意してください。大文字で指定しないとサンプルは動作しません。

マシン MACHINE1、キュー・マネージャー QMGR1 の場合

MQSC 構文

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **IBM i コマンド構文**

注 : キュー・マネージャー名を指定しないと、デフォルトのキュー・マネージャーがシステムに使用されます。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
            REPLACE(*YES) TRPTYPE(*TCP) +
            CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
            MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ   QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
            REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*RCVR) +
            MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ   QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
            REPLACE(*YES) RMTQNAME(DQ) +
            RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

マシン MACHINE2、キュー・マネージャー QMGR2 の場合

MQSC 構文

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i **IBM i コマンド構文**

注 : IBM i IBM i では、キュー・マネージャー名を指定しないと、デフォルトのキュー・マネージャーがシステムに使用されます。

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
```

```
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. IBM MQ オブジェクトの作成が終わったら、以下の作業を行います。
   a. (該当するプラットフォームの場合には) 送信側および受信側のキュー・マネージャーでリスナーを始動します。
   b. チャネル CHL1 および REPORT を始動します。
   c. 受信側のキュー・マネージャーで、開始キュー INITQ のトリガー・モニターを始動します。
3. 以下のパラメーターを使用して、コマンド行から書き込み参照メッセージ・サンプル・プログラム AMQSPRM (IBM i の場合は AMQSPRMA) を呼び出します。

   -m
   ローカル・キュー・マネージャーの名前。省略すると、デフォルトのキュー・マネージャーが設定されます。

   -i
   ソース・ファイルの名前と格納場所

   -o
   宛先ファイルの名前と格納場所

   -q
   キューの名前

   -g
   -q パラメーターで定義されたキューのあるキュー・マネージャーの名前。 デフォルトで、-m パラメーターで指定されたキュー・マネージャーになります。

   -t
   オブジェクト・タイプ

   -w
   待機間隔。つまり、受信側のキュー・マネージャーから出る例外報告および COA 報告の待機時間。

   例えば、上記のように定義されたオブジェクトを用いてサンプルを使用するには、パラメーターを次のように指定します。

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   待機時間を長くすると、大きなファイルをネットワークで送信する場合、メッセージを書き込んでいるプログラムがタイムアウトになる前にファイルが送信されるよう余裕をとっておくことができます。

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

IBM i ユーザー : 

IBM i の場合、以下のステップを実行します。

   a. 以下のコマンドを使用します。

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

ここでは、元のファイル rmsg1 が IFS ディレクトリー /refmsgs にあり、宛先ファイルを rmsgx としてターゲット・システムの IFS ディレクトリー /refmsgs に置くことを前提とします。

b. CRTDIR コマンドを使って独自のディレクトリーを作成します。ルート・ディレクトリーは使用しないでください。

c. データを書き込むプログラムを呼び出すときには、IFS での命名規則に従って出力ファイル名を指定してください。例えば、/TEST/FILENAME と指定すると、TEST というディレクトリー内に FILENAME というファイルが作成されます。

注:

IBM i IBM i では、パラメーターを指定するとき、順方向の斜線 (/) またはダッシュ (-) のどちらも使用できます。 以下に例を示します。

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX AIX and Linux プラットフォームの場合、宛先ファイルのディレクトリーを示すのに、円記号を 1 つではなく 2 つ (¥¥) 使用する必要があります。 したがって、**amqsprm** コマンドは次のように表示されます。

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

書き込み参照メッセージ・プログラムを実行すると、以下のことが行われます。

- 参照メッセージは、キュー・マネージャー QMGR1 でキュー QR に書き込まれます。
- ソース・ファイルとパスは d:\files\infile.dat で、コマンド例が発行されたシステム上に存在します。
- キュー QR がリモート・キューである場合、参照メッセージは別のシステム上の別のキュー・マネージャーに送信され、そこで名前とパス e:\files\outfile.dat を使用してファイルが作成されます。 このファイルの内容はソース・ファイルの内容と同じです。
- amqsprm は宛先キュー・マネージャーからの COA 報告を 30 秒間待機します。
- オブジェクト・タイプは flatfile なので、キュー QR からのメッセージを移動するのに使用されたチャネルは、このオブジェクト・タイプを *MsgData* フィールドに指定しなければなりません。

4. ユーザーのチャネルを定義する場合は、送信用と受信用の両方のメッセージ出口を amqsxrm に選択します。

Windows これは Windows で次のように定義されます。

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX これは AIX and Linux で次のように定義されます。

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

パス名を指定する場合は、完全な名前を指定してください。 パス名を省略すると、プログラムは、qm.ini ファイルで指定されたパス (または、IBM MQ for Windows で、レジストリーに指定されたパス) であると見なされます。

5. チャネル出口は参照メッセージ・ヘッダーを読み取り、参照するファイルを検索します。
6. 次にチャネル出口はファイルをセグメント化できます。その後、チャネルにファイルをヘッダーと共に送ります。

Linux AIX AIX and Linux では、サンプル・メッセージ出口がターゲット・ディレクトリーにファイルを作成できるように、そのディレクトリーのグループ所有者を「mqm」に変更してくだ

さい。 さらに、ターゲット・ディレクトリーの許可を変更して、mqm グループ・メンバーがターゲット・ディレクトリーに書き込めるようにします。 ファイルのデータは IBM MQ キューには格納されません。

7. ファイルの最後のセグメントが受信メッセージ出口によって処理されると、参照メッセージは amqsprm で指定された宛先キューに書き込まれます。 このキューが起動されると (つまり、その定義が **Trigger**、**InitQ**、および **Process** のキュー属性を指定すると)、宛先キューの PROC パラメーターで指定されたプログラムも起動されます。 トリガーされるプログラムは、 **Process** 属性の ApplId フィールドで定義する必要があります。
8. 参照メッセージが宛先キュー (DQ) に到達すると、COA 報告が書き込みアプリケーション (amqsprm) に返送されます。
9. 読み取り参照メッセージ・サンプルである amqsgrm は、入力トリガー・メッセージで指定されたキューからメッセージを読み込み、ファイルの有無を確認します。

書き込み参照メッセージ・サンプル *(amqsprma.c、AMQSPRM4)* の設計

このトピックでは、書き込み参照メッセージ・サンプルの詳細について説明します。

このサンプルはファイルを参照する参照メッセージを作成し、そのメッセージを指定されたキューに書き込みます。

1. サンプルは MQCONN を使用してローカル・キュー・マネージャーと接続します。
2. 次にサンプルは MQOPEN を使用して、報告メッセージを受信するために使用されるモデル・キューをオープンします。
3. サンプルはファイルを移動するのに必須の値が含まれた参照メッセージを作成します。この値とは、例えば、送信側ファイルの名前および宛先ファイルの名前、そしてオブジェクト・タイプなどです。 例として、 IBM MQ に付属しているサンプルは、 QMGR1 から QMGR2 にファイル d:\x\file.in を送信し、以下のパラメーターを使用して d:\y\file.out としてファイルを再作成するための参照メッセージを作成します。

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   ここで、QR は QMGR2 の宛先キューを参照するリモート・キューの定義を表します。

   注 : AIX and Linux プラットフォームの場合、宛先ファイルのディレクトリーを示すのに、円記号を 1 つではなく 2 つ (¥¥) 使用します。 したがって、**amqsprm** コマンドは次のように表示されます。

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```

4. 参照メッセージは、(ファイルのデータを除いて) /q パラメーターで指定されたキューに書き込まれます。 これがリモート・キューの場合、メッセージは対応する伝送キューに書き込まれます。
5. サンプルは、/w パラメーターで指定された時間 (デフォルトは 15 秒) COA 報告を待機します。これは例外報告と共にローカル・キュー・マネージャー (QMGR1) で作成された動的キューに返送されます。

参照メッセージ出口サンプル *(amqsxrma.c、AMQSXRM4)* の設計

このサンプルは、チャネル定義のメッセージ出口ユーザー・データ・フィールド内のオブジェクト・タイプと一致するオブジェクト・タイプを持つ参照メッセージを認識します。

これらのメッセージについては、以下の 2 つのことが起こります。

- 送信側チャネルまたはサーバー・チャネルでは、指定された長さのデータを、指定されたファイルの指定されたオフセットから、参照メッセージのあとのエージェント・バッファーに残っている空間にコピーします。 ァイルの終わりに達していない場合、参照メッセージは *DataLogicalOffset* フィールドの更新後に伝送キューに戻されます。
- 要求側チャネルまたは受信側チャネルでは、 *DataLogicalOffset* フィールドがゼロで、指定されたファイルが存在しない場合、そのファイルが作成されます。 参照メッセージの次にくるデータは、指定されたファイルの最後に追加されます。 参照メッセージが指定されたファイルの最後のメッセージではな

い場合、参照メッセージは廃棄されます。 最後のメッセージである場合は、参照メッセージは追加データなしでチャネル出口に戻り、宛先キューに書き込まれます。

送信側チャネルおよびサーバー・チャネルでは、入力参照メッセージの *DataLogicalLength* フィールドがゼロの場合、ファイルの残りの部分 (*DataLogicalOffset* からファイルの終わりまで) がチャネルを介して送信されます。 上記のフィールドがゼロでない場合は、指定された長さだけが送信されます。

エラーが発生した場合 (例えば、サンプルがファイルをオープンできない場合) は、MQCXP。*ExitResponse* が MQXCC_SUPPRESS_FUNCTION に設定されているため、処理中のメッセージは、宛先キューに進む代わりに送達不能キューに書き込まれます。 フィードバック・コードが MQCXP に戻されます。 *Feedback* 。レポート・メッセージのメッセージ記述子の *Feedback* フィールドにメッセージを書き込んだアプリケーションに返されます。 これは、書き込み側アプリケーションが MQMD の *Report* フィールドに MQRO_EXCEPTION を設定することによって例外レポートを要求したためです。

参照メッセージのエンコードまたは *CodedCharacterSetId* (CCSID) がキュー・マネージャーのものと異なる場合、参照メッセージはローカル・エンコードおよび CCSID に変換されます。 システムが提供するサンプルである amqsprm では、オブジェクトの形式は MQFMT_STRING なので、amqsxrm はオブジェクト・データがファイルに書き込まれる前に受信側でローカル CCSID に変換します。

このファイルにマルチバイト文字 (例えば、DBCS または Unicode) が含まれる場合は、転送されるファイルの形式を MQFMT_STRING として指定しないでください。 これは、ファイルが送信側でセグメント化されるときに、マルチバイト文字が分割される可能性があるからです。 このようなファイルを転送し、変換するためには、参照メッセージ出口でファイルを変換せずに、転送が完了した時点で受信側でそのファイルを変換するように、そのファイルを MQFMT_STRING 以外の形式で指定してください。

参照メッセージ出口サンプルのコンパイル
参照メッセージ出口サンプルをコンパイルするには、IBM MQ がインストールされているプラットフォーム用のコマンドを使用します。

MQ_INSTALLATION_PATH は、IBM MQ がインストールされている上位ディレクトリーを表します。

amqsxrma をコンパイルするには、以下のコマンドを使用します。

AIX 上

AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

IBM i 上

IBM i

```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

注 :

1. IFS ファイル・システムを使用するようにモジュールを作成するには、オプション SYSIFCOPT(*IFSIO) を追加してください。
2. スレッド化されていないチャネルと共に使用されるプログラムを作成するには、以下のコマンドを使用します。CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. スレッド化チャネルと共に使用されるプログラムを作成するには、以下のコマンドを使用します。CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

Linux 上

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

Windows 上

Windows IBM MQ では、サーバー・パッケージだけでなくクライアント・パッケージでも mqm ライブラリーが提供されるようになったため、以下の例では `mqmvx.lib` の代わりに `mqm.lib` が使用されています。

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

関連概念

970 ページの『チャネル出口プログラムの作成』
チャネル出口プログラムの作成に役立つ情報を 以下に記載します。

読み取り参照メッセージ・サンプル *(amqsgrma.c、AMQSGRM4)* の設計
このトピックでは、読み取り参照メッセージ・サンプルの設計について説明します。

プログラム・ロジックは次のとおりです。

1. サンプルは起動され、キュー名およびキュー・マネージャー名を入力トリガー・メッセージから抜き出します。
2. 次に、サンプルは MQCONN を使用して指定されたキュー・マネージャーに接続し、MQOPEN を使用して指定されたキューをオープンします。
3. サンプルはループ内に 15 秒の待機時間を持つ MQGET 呼び出しを発行し、キューからメッセージを読み取ります。
4. メッセージが参照メッセージの場合、サンプルは転送されたファイルの有無を確認します。
5. そこでサンプルはキューをクローズし、キュー・マネージャーから切断します。

要求サンプル・プログラム

要求サンプル・プログラムは、クライアント / サーバー処理を示します。 このサンプルは、要求メッセージをサーバー・プログラムによって処理する宛先サーバー・キューに入れるクライアントです。 サーバー・プログラムが応答先キューに応答メッセージを書き込むのを待機します。

要求サンプルは、MQPUT 呼び出しを使用して、一連の要求メッセージを宛先サーバー・キューに入れます。 これらのメッセージは、ローカル・キュー SYSTEM.SAMPLE.REPLY を応答先キュー (ローカル・キューまたはリモート・キューのいずれか) として指定します。 プログラムは応答メッセージを受信するまで待機し、そのメッセージを表示します。 応答は、宛先サーバー・キューがサーバー・アプリケーションによって処理中であるか、あるいはアプリケーションがその目的で起動される場合にのみ送信されます (照会、設定、およびエコー・サンプル・プログラムは起動されるように設計されています)。 C サンプルは、最初の応答が到着するまで、1 分間 (COBOL サンプルは 5 分間) 待機し (サーバー・アプリケーションが起動される時間を与えるため)、以降の応答については 15 秒間待機しますが、両サンプルは共に応答がなくても終了することができます。 要求サンプル・プログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

要求サンプル・プログラムの実行

amqsreq0.c、amqsreq、および amqsreqc のサンプルの実行

C バージョンのプログラムでは、次の 3 つのパラメーターをとります。

1. 宛先サーバー・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

3. 応答キュー (オプション)

例えば、それぞれ次のように入力します。

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

この例では、`myqueue` が宛先サーバー・キューの名前、`qmanagername` が `myqueue` を所有するキュー・マネージャーの名前、そして `replyqueue` が応答キューの名前です。

キュー・マネージャーの名前を省略すると、デフォルトのキュー・マネージャーがキューを所有すると見なされます。 応答キューの名前を省略すると、デフォルトの応答キューが指定されます。

amq0req0.cbl サンプルの実行

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target server queue
```

このプログラムでは、StdIn からの入力値を使用し、テキストの各行を要求メッセージの内容として使用しながら、各行を宛先サーバー・キューに付加します。 このプログラムは、ヌル行を読み込んだ時点で終了します。

AMQSREQ4 サンプルの実行

C プログラムは、stdin (キーボード) からデータを取り込むことによってメッセージを作成します。入力が終了した時点でブランクが現れます。 このプログラムは、最高で 3 つのパラメーターを取ります。それらは、宛先キュー名 (必須)、キュー・マネージャー名 (オプション)、応答先キュー名 (オプション) です。 キュー・マネージャー名が指定されない場合は、デフォルトのキュー・マネージャーが使用されます。 応答先キューが指定されない場合は、SYSTEM.SAMPLE.REPLY キューが使用されます。

次の例は、C サンプル・プログラムの呼び出し例です。ここでは、応答先キューを指定し、デフォルトのキュー・マネージャーを使用します。

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

注 : キュー名の場合、大文字と小文字の区別があることに注意してください。 サンプル・ファイル作成プログラム AMQSAMP4 によって作成されたキューの名前はすべて、大文字で作成されます。

AMQ0REQ4 サンプルの実行

COBOL プログラムは、キーボードからのデータを受け入れて、メッセージを作成します。 プログラムを始動するには、始動するプログラムを呼び出し、宛先キューの名前をパラメーターとして指定します。 プログラムは、キーボードから受け入れた入力をバッファーに入れ、テキストの各行ごとに要求メッセージを作成します。 キーボードでブランク行を入力するとプログラムは停止します。

トリガー操作を使用した要求サンプルの実行

サンプルがトリガー操作および、照会、設定、またはエコーのいずれかのサンプル・プログラムと共に使用される場合、入力行は、起動されたプログラムにアクセスさせたいキューの名前でなければなりません。

ALW *AIX, Linux, and Windows* でトリガー操作を使用する要求サンプルの実行

AIX, Linux, and Windows において、1 つのセッションでトリガー・モニター・プログラム RUNMQTRM を開始した後、別のセッションで amqsreq プログラムを開始します。

トリガー操作を使用してサンプルを実行するには、次のようにします。

1. 1 つのセッションでトリガー・モニター・プログラム RUNMQTRM を始動します (開始キュー SYSTEM.SAMPLE.TRIGGER が使用できます)。
2. 別のセッションで amqsreq プログラムを始動します。
3. 宛先サーバー・キューが定義されているか確認します。

   要求サンプルがメッセージを入れるための宛先サーバー・キューとして使用可能なサンプル・キューには、次のものがあります。

   - SYSTEM.SAMPLE.INQ - 照会サンプル・プログラム用
   - SYSTEM.SAMPLE.SET - 設定サンプル・プログラム用
   - SYSTEM.SAMPLE.ECHO - エコー・サンプル・プログラム用

   これらのキューには、FIRST というトリガー・タイプがありますが、要求サンプルを実行する前にキュー上に既にメッセージがある場合には、サーバー・アプリケーションはユーザーが送信したメッセージによって起動されません。
4. 照会、設定、またはエコー・サンプル・プログラムを使用するために、キューが定義されているか確認します。

これは、要求サンプルがメッセージを送信するときにトリガー・モニターは作動可能であることを意味します。

注 : RUNMQSC および amqscos0.tst ファイルを使用して作成されたサンプル・プロセス定義によって C のサンプルが起動されます。 COBOL バージョンを使用するには、amqscos0.tst のプロセス定義を変更し、この更新済みファイルと共に RUNMQSC を使用してください。

1120 ページの図 132 は、要求サンプルと照会サンプルを一緒に使用する方法を示しています。

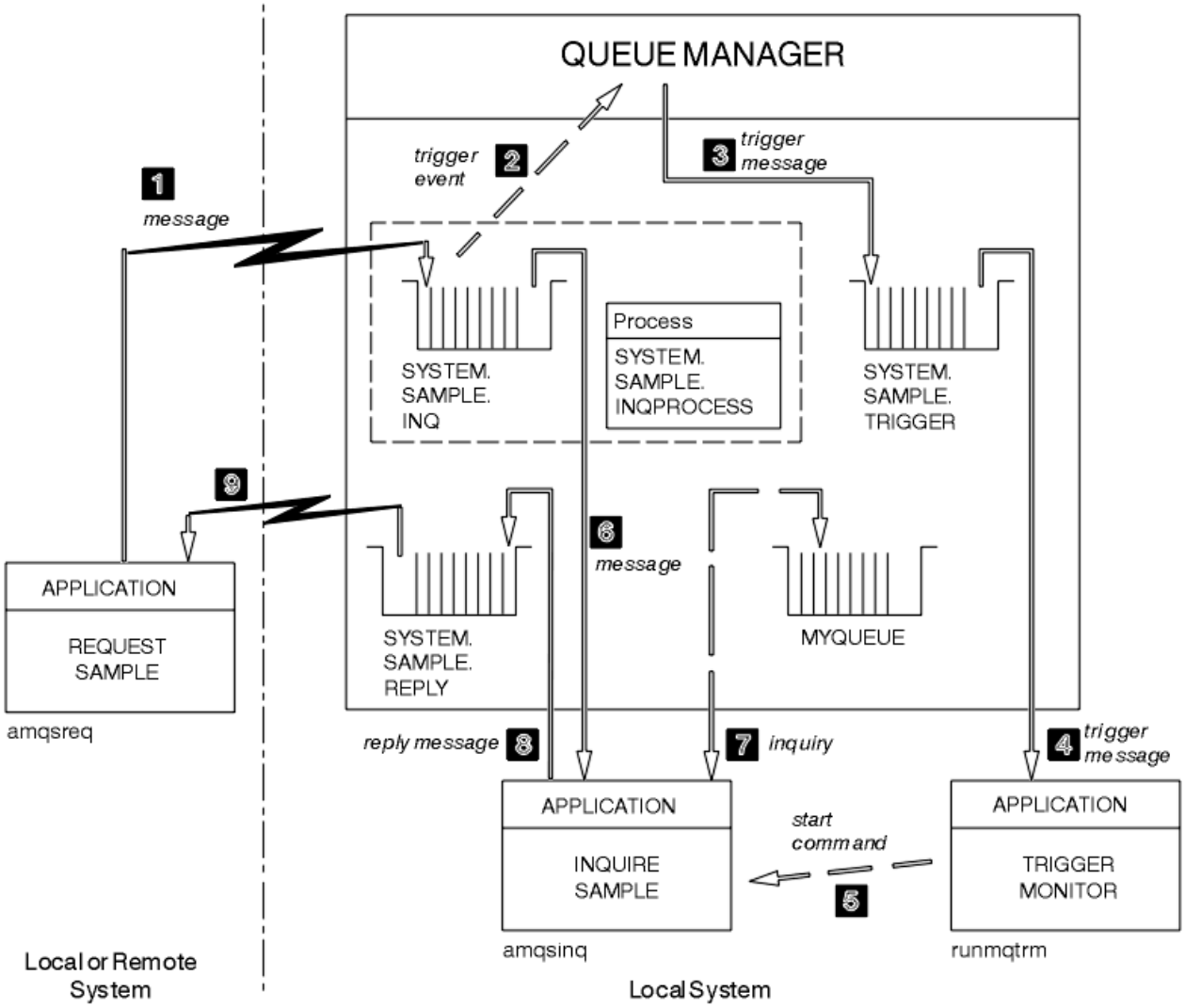


図 *132.* トリガー操作を使用する要求および照会サンプル

1120 ページの図 132 では、要求サンプルはメッセージを宛先サーバー・キュー SYSTEM.SAMPLE.INQ に入れ、照会サンプルはキュー MYQUEUE を照会します。 また、amqscos0.tst を実行した時に定義したいずれかのサンプル・キュー、または定義したそれ以外のキューを照会サンプルとして使用することもできます。

注 : 1120 ページの図 132 の数字は、イベントの発生する順序を示しています。

トリガー操作を使用して、要求サンプルおよび照会サンプルを実行するには、次のようにします。

1. 使用したいキューが定義されているか検査します。 amqscos0.tst を実行してサンプル・キューを定義すると共に、キュー MYQUEUE を定義します。
2. トリガー・モニター・コマンド RUNMQTRM を実行します。

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. 要求サンプルを実行します。

```
amqsreq SYSTEM.SAMPLE.INQ
```

注 : プロセス・オブジェクトは起動する必要があるものを定義します。 クライアントおよびサーバーが同じプラットフォーム上で実行されていない場合、トリガー・モニターが開始したプロセスは

ApplType を定義しなければなりません。定義していないと、サーバーはデフォルトの定義 (つまり、通常サーバー・マシンに関連しているアプリケーションのタイプ) を選択し、障害を引き起こします。

アプリケーション・タイプのリストについては、ApplType を参照してください。

4. 照会サンプルに使用したいキューの名前を入力します。

```
MYQUEUE
```

5. ブランク行を入力します (要求プログラムを終了するため)。
6. ここで、要求サンプルは照会プログラムが MYQUEUE から獲得したデータを含むメッセージを表示します。

複数のキューを使用できます。この場合、ステップ 1121 ページの『4』 で他のキューの名前を入力します。

トリガー操作の詳細については、869 ページの『トリガーによる IBM MQ アプリケーションの開始』を参照してください。

IBM i *IBM i でトリガー操作を使用する要求サンプルの実行*

IBM i において、1 つのジョブでサンプル・トリガー・サーバー AMQSERV4 を開始した後、別のジョブで AMQSREQ4 を開始します。 つまり、要求サンプル・プログラムがメッセージを送信する時点で、トリガー・サーバーが作動可能になっています。

注 :

1. AMQSAMP4 によって作成されたサンプル定義により、C バージョンのサンプルが起動されます。 COBOL バージョンを起動したい場合は、プロセス定義 SYSTEM.SAMPLE.ECHOPROCESS、SYSTEM.SAMPLE.INQPROCESS、および SYSTEM.SAMPLE.SETPROCESS を変更してください。 これらの変更は、CHGMQMPRC コマンド (詳細については、MQ プロセスの変更 (CHGMQMPRC) を参照) を使用して行うことも、AMQSAMP4 の独自バージョンを編集し実行することで行うこともできます。
2. AMQSERV4 のソース・コードは、C 言語のみで提供されています。 ただし、コンパイル済みのバージョン (COBOL サンプルで使用可能) がライブラリー QMQM に提供されています。

要求メッセージを以下のサンプル・サーバー・キューに入れることができます。

- SYSTEM.SAMPLE.ECHO (エコー・サンプル・プログラム用)
- SYSTEM.SAMPLE.INQ (照会サンプル・プログラム用)
- SYSTEM.SAMPLE.SET (設定サンプル・プログラム用)

SYSTEM.SAMPLE.ECHO プログラムのフローチャートを 1123 ページの図 133 に示します。 データ・ファイルの例を使用すると、このサーバーへの C プログラムの要求コマンドは次のようになります。

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

注 : このサンプル・キューのトリガー・タイプは FIRST です。したがって、要求サンプルを実行する前にメッセージが既にキュー上にある場合、サーバー・アプリケーションは送信したメッセージによって起動されません。

例をさらに試したい場合には、以下の方法があります。

- AMQSERV4 の代わりに AMQSTRG4 (または、コマンド行でこれに相当する STRMQMTRM。詳細については、MQ トリガー・モニターの開始 (STRMQMTRM) を参照) を使用してジョブを実行依頼します。ただし、ジョブの実行依頼が遅れる可能性があるため、処理の流れを追うのが難しくなります。
- SYSTEM.SAMPLE.INQUIRE および SYSTEM.SAMPLE.SET サンプル・プログラムを実行します。 データ・ファイルの例を使用すると、これらのサーバーへの C プログラムの要求コマンドはそれぞれ次のようになります。

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

これらのサンプル・キューのトリガー・タイプも FIRST になります。

要求サンプル・プログラムの設計

次に、このプログラムは、宛先サーバー・キューをオープンして、メッセージを入れられるようにします。このプログラムでは、MQOO_OUTPUT オプション付きの MQOPEN 呼び出しを使用します。 キューをオープンできない場合、プログラムは MQOPEN 呼び出しによって戻された理由コードを含むエラー・メッセージを表示します。

その後プログラムは、応答メッセージを読み取るために、SYSTEM.SAMPLE.REPLY と呼ばれる応答先キューをオープンします。 これに関して、このプログラムは、MQOO_INPUT_EXCLUSIVE オプション付きの MQOPEN 呼び出しを使用します。 プログラムは、キューをオープンできない場合に、MQOPEN 呼び出しによって戻される理由コードを含むエラー・メッセージを表示します。

次にプログラムは、入力行ごとにテキストをバッファー中に読み込み、MQPUT 呼び出しを使用してその行のテキストを含む要求メッセージを作成します。 この呼び出しで、プログラムは MQRO_EXCEPTION_WITH_DATA 報告オプションを使用して、要求メッセージについて送信されるすべての報告メッセージに 100 バイトまでのメッセージ・データを入れるように要求します。 プログラムは、入力が終了するか、MQPUT 呼び出しが異常終了するまで処理を続行します。

その後プログラムは、MQGET 呼び出しを使用してキューから応答メッセージを除去し、応答に含まれるデータを表示します。 MQGET 呼び出しでは、MQGMO_WAIT、MQGMO_CONVERT、および MQGMO_ACCEPT_TRUNCATED オプションを使用します。 *WaitInterval* は、最初の応答では、COBOL バージョンでは 5 分、C バージョンでは 1 分となります (サーバーのアプリケーションを起動する時間を与えるため)。以降の応答では、15 秒となります。 キューにメッセージがない場合、プログラムはこれらの期間待ちます。 メッセージがこの時間内に到着しない場合、呼び出しは失敗し、MQRC_NO_MSG_AVAILABLE 理由コードを戻します。 この呼び出しでは、MQGMO_ACCEPT_TRUNCATED_MSG オプションも使用します。これによって、宣言されたバッファー・サイズよりも大きいメッセージは切り捨てられます。

このプログラムは、それぞれの MQGET 呼び出しのあとに MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアする方法を示します。これは、この呼び出しによって、これらのフィールドに、このプログラムが取り出すメッセージに含まれる値が設定されるからです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

このプログラムは、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE 理由コードを戻すか、あるいは MQGET 呼び出しが失敗するまで、処理を続行します。 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

次に、このプログラムは、MQCLOSE 呼び出しを使用して、宛先サーバー・キューと応答先キューの両方をクローズします。

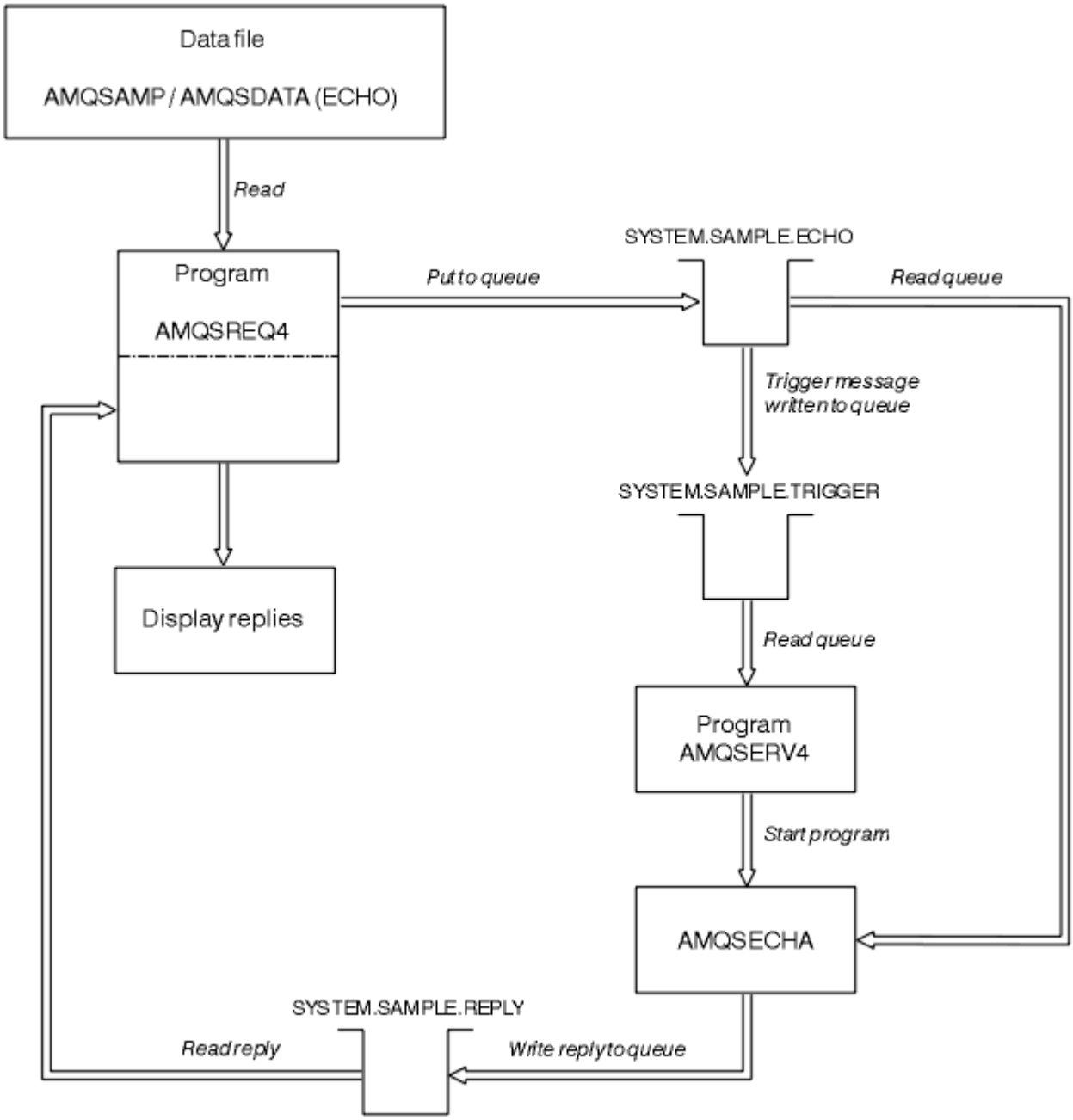


図 *133.* サンプル *IBM i* クライアント/サーバー *(*エコー*)* プログラムのフローチャート

設定サンプル・プログラム

設定サンプル・プログラムは、MQSET 呼び出しを使用してキューの **InhibitPut** 属性を変更することによって、キュー上での書き込み操作を禁止します。 また、設定サンプル・プログラムの設計について説明します。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

このプログラムは、トリガーされるプログラムとして実行することを意図したものです。したがって、その入力は、照会される属性を持つ宛先キューの名前を含む MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 C バージョンもキュー・マネージャー名を使用します。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

トリガー・プロセスを処理するには、使用したい設定サンプル・プログラムがキュー SYSTEM.SAMPLE.SET に到着するメッセージによって起動されているか確認してください。 これを行うには、使用したい設定サ

ンプル・プログラムの名前を、プロセス定義 SYSTEM.SAMPLE.SETPROCESS の *ApplicId* フィールドに指定します。 サンプル・キューのトリガー・タイプは FIRST です。要求サンプルを実行する前にメッセージが既にキュー上にある場合、設定サンプルはユーザーが送信したメッセージによって起動されません。

正しく定義を設定したら、以下のようにします。

- ALW AIX, Linux, and Windows システムの場合、1 つのセッションで **runmqtrm** プログラムを開始後、別のセッションで amqsreq プログラムを開始します。
- IBM i IBM i の場合、1 つのセッションで AMQSERV4 プログラムを開始後に、別のセッションで AMQSREQ4 プログラムを開始します。 AMQSERV4 の代わりに AMQSTRG4 を使用しても構いませんが、ジョブの送信が遅れる可能性があるため、処理の流れを追うのが難しくなります。

要求サンプル・プログラムを使用して、それぞれがキュー名のみを含む要求メッセージをキュー SYSTEM.SAMPLE.SET に送信します。 各要求メッセージごとに、設定サンプル・プログラムは、指定のキューで書き込み操作が禁止されたことの確認を含む応答メッセージを送信します。 応答は、要求メッセージで指定した応答先キューに送信されます。

設定サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、このキューを要求キュー と呼びます。) このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各要求メッセージが要求キューから除去されたら、このプログラムは、そのデータに含まれるそのキュー (宛先キュー と呼びます) の名前を読み込み、MQOO_SET オプション付きの MQOPEN 呼び出しを使用して、そのキューをオープンします。 次に、このプログラムは、MQSET 呼び出しを使用して、宛先キューの **InhibitPut** 属性値を MQQA_PUT_INHIBITED に設定します。

MQSET 呼び出しが正常に行われると、このプログラムは MQPUT1 呼び出しを使用して、応答メッセージを応答先キューに入れます。 このメッセージには、ストリング `PUT inhibited` が含まれています。

MQOPEN または MQSET 呼び出しが失敗すると、プログラムは MQPUT1 呼び出しを使用して、`report` メッセージを応答先キューに書き込みます。 この報告メッセージのメッセージ記述子の *Feedback* フィールドには、MQOPEN または MQSET の失敗した呼び出しによって、戻される理由コードが入ります。

MQSET 呼び出し後、このプログラムは MQCLOSE 呼び出しを使用して、宛先キューをクローズします。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

TLS サンプル・プログラム

AMQSSSLC はサンプルの C プログラムであり、MQCONNX 呼び出しで TLS クライアント接続情報を提供する MQCNO 構造体および MQSCO 構造体の使用方法を示します。 このプログラムを使用すると、クライアント MQI アプリケーションで、クライアント・チャネル定義テーブル (CCDT) を使用せずに実行時にそのクライアント接続チャネルの定義および TLS 設定が提供されます。

接続名が指定されると、プログラムは MQCD 構造体にクライアント接続チャネル定義を構成します。

キー・リポジトリー・ファイルの語幹名が指定されると、プログラムは MQSCO 構造体を構成します。OCSP レスポンダー URL も指定された場合、プログラムは認証情報レコード MQAIR 構造体を構成します。

次に、プログラムは MQCONNX を使用してキュー・マネージャーに接続します。 接続先のキュー・マネージャーの名前が照会され、表示されます。

このプログラムは、MQI クライアント・アプリケーションとしてリンクされることを前提としています。ただし、通常の MQI アプリケーションとしてリンクすることはできます。この場合、ローカル・キュー・マネージャーに接続するだけで、クライアント接続情報は無視されます。

V 9.3.0 V 9.3.0 鍵リポジトリーにアクセスするためのパスフレーズがファイルにスタッシュされていない場合は、アプリケーションの実行時に **amqssslc** にパスフレーズを提供する必要があります。パスフレーズは、以下のいずれかの方法で指定できます。

- パスフレーズの入力を求めるプロンプトを **amqssslc** に要求する、または
- *MQKEYRPWD* 環境変数を使用する。
- クライアント構成ファイルでの **SSLKeyRepositoryPassword** 属性の使用

IBM MQ MQI client アプリケーションに鍵リポジトリー・パスワードを提供する方法について詳しくは、IBM MQ MQI client on AIX, Linux, and Windows 用の鍵リポジトリー・パスワードの提供 を参照してください。

amqssslc は、以下のパラメーターを受け入れます。これらはすべてオプションです。

-m QmgrName
接続先のキュー・マネージャーの名前

-c ChannelName
使用するチャネルの名前

-x ConnName
サーバー接続名

TLS パラメーター:

V 9.3.0 V 9.3.0 **-k KeyReposFileName**
鍵リポジトリー・ファイルの名前。 ファイル拡張子が指定されていない場合は、.kdb であると想定されます。 以下に例を示します。

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
キュー・マネージャー上の SVRCONN チャネル定義の **SSLCIPH** に対応する TLS チャネル CipherSpec ストリング。

-f
FIPS 140-2 認証アルゴリズムのみを使用する必要があることを指定します。

-b VALUE1[,VALUE2...]
Suite B 準拠アルゴリズムのみを使用する必要があることを指定します。 このパラメーターは、NONE、128_BIT、192_BIT の 1 つ以上の値の、コンマ区切りのリストです。 これらの値は、 **MQSUITEB** 環境変数の値、およびクライアント構成ファイル SSL スタンザの同等の **EncryptionPolicySuiteB** 設定の値と同じ意味を持ちます。

-p Policy
使用する証明書妥当性検査ポリシーを指定します。 以下のいずれかの値を選択できます。

ANY
セキュア・ソケット・ライブラリーでサポートされる証明書妥当性検査ポリシーのいずれかにおいて、その証明書チェーンが有効であると見なされる場合に、それらのポリシーのそれぞれを適用し、証明書チェーンを受け入れます。 この設定は、最新の証明書標準に準拠しない旧式のデジタル証明書との後方互換性を最大にするために使用できます。

RFC5280
RFC 5280 準拠の証明書妥当性検査ポリシーのみを適用します。 この設定は、ANY 設定よりも厳密に妥当性検査しますが、一部の旧式のデジタル証明書を拒否します。

デフォルト値は ANY です。

-l CertLabel
セキュア接続に使用する証明書ラベル。

注 : 値は小文字で指定する必要があります。

V 9.3.0 V 9.3.0 **-w**
指定する鍵リポジトリー・パスフレーズの入力を求めるプロンプトを **amqssslc** が出すことを指定します。

V 9.3.0 V 9.3.0 **-i**
amqssslc が、提供される鍵リポジトリー・パスフレーズの暗号化に使用される初期鍵の入力を求めるプロンプトを出すことを指定します。

鍵リポジトリー・パスフレーズが **runmqicred** ユーティリティーを使用して暗号化されたときに初期鍵ファイルが指定された場合は、このオプションを指定します。

OCSP 証明書の失効のパラメーター:

-o URL
OCSP レスポンダー URL

以下のいずれかの環境変数を設定して、キュー・マネージャーでの認証に使用される資格情報を提供することもできます。

MQSAP ユーザー ID
キュー・マネージャーでの認証にユーザー ID とパスワードを使用する場合は、接続認証に使用するユーザー ID に設定します。 プログラムは、ユーザー ID に付随するパスワードの入力を求めるプロンプトを出します。

V 9.3.4 Linux AIX **MQSAP トークン**
キュー・マネージャーで認証する認証トークンを指定する場合は、非ブランク値に設定します。 プログラムは、認証トークンの入力を求めるプロンプトを出します。

TLS サンプル・プログラムの実行
TLS サンプル・プログラムを実行するには、まず最初に TLS 環境をセットアップする必要があります。 その後、いくつかのパラメーターを指定して、コマンド行からサンプルを実行します。

このタスクについて
以下の手順では、個人証明書を使用するサンプル・プログラムを実行します。 コマンドを変更することにより、例えば、CA 証明書を使用して、OCSP レスポンダーを使用した CA 証明書の状況の検査を行うことができます。 サンプル内の説明を参照してください。

手順
1. QM1 という名前で、キュー・マネージャーを作成する。 詳しくは、「crtmqm」を参照してください。
2. キュー・マネージャーのキー・リポジトリーを作成する。 詳しくは、AIX, Linux, and Windows での鍵リポジトリーのセットアップを参照してください。
3. クライアントのキー・リポジトリーを作成する。 このリポジトリーに *clientkey.kdb* という名前を付けます。

   鍵リポジトリーの作成時に、鍵リポジトリー・パスワードをファイルにスタッシュします。
4. キュー・マネージャーの個人証明書を作成する。 詳しくは、AIX, Linux, and Windows での自己署名個人証明書の作成を参照してください。
5. クライアントの個人証明書を作成する。
6. サーバーのキー・リポジトリーから個人証明書を抽出し、クライアント・リポジトリーに追加する。 詳しくは、AIX, Linux, and Windows で鍵リポジトリーから自己署名証明書の公開部分を抽出する、および AIX, Linux, and Windows システムで CA 証明書 (または自己署名証明書の公開部分) を鍵リポジトリーに追加するを参照してください。
7. クライアントのキー・リポジトリーから個人証明書を抽出し、サーバーのキー・リポジトリーに追加する。
8. 以下のように、MQSC コマンドを使用してサーバー接続チャネルを作成する。

```
DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
```

詳しくは、「サーバー接続チャネル」を参照してください。

9. キュー・マネージャー上のチャネル・リスナーを定義および開始する。 詳しくは、「DEFINE LISTENER」および「START LISTENER」を参照してください。
10. 以下のコマンドを使用して、サンプル・プログラムを実行する。

V 9.3.0 V 9.3.0

```
AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
-k "C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\clientkey.kdb" -s
TLS_RSA_WITH_AES_128_CBC_SHA256
-o http://dummy.OCSP.responder
```

タスクの結果

サンプル・プログラムは以下の操作を実行します。

1. 指定されたオプションを使用して、指定されたキュー・マネージャーまたはデフォルトのキュー・マネージャーに接続します。
2. キュー・マネージャーを開いて、その名前を照会します。
3. キュー・マネージャーを閉じます。
4. キュー・マネージャーから切断します。

サンプル・プログラムが正常に実行されると、以下の例のような出力が表示されます。

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:&#xa5;Program Files&#xa5;IBM&#xa5;MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

サンプル・プログラムで問題が発生した場合、該当するエラー・メッセージが表示されます。例えば、無効な OCSP レスポンダー URL を指定すると、以下のようなメッセージを受け取ります。

```
MQCONNX ended with reason code 2553
```

理由コードのリストについては、 API 完了コードと理由コードを参照してください。

トリガー・サンプル・プログラム

トリガー・サンプルに提供されている関数は、**runmqtrm** プログラム内のトリガー・モニターに提供されているトリガー・サンプルのサブセットです。

これらのプログラムの名前については、1064 ページの『Multiplatforms のサンプル・プログラムで示されている機能』を参照してください。

トリガー・サンプルの設計

トリガー・サンプル・プログラムは、MQOO_INPUT_AS_Q_DEF オプション付きの MQOPEN 呼び出しを使用して、開始キューをオープンします。 このプログラムは、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプション (無制限の待機時間を指定する) 付きの MQGET 呼び出しを使用して、開始キューからメッセージを取得します。 このプログラムは、それぞれの MQGET 呼び出しがメッセージを順に取得する前に、*MsgId* フィールドおよび *CorrelId* フィールドをクリアします。

開始キューからメッセージが取り出されると、このプログラムはそのメッセージのサイズを検査して、MQTM 構造体のサイズと同じであるか確認します。 このテストが失敗すると、プログラムは警告を表示します。

トリガー・メッセージが有効である場合、トリガー・サンプルは、*ApplicId*、*EnvrData*、*Version*、および *ApplType* の各フィールドからデータをコピーします。 これらのフィールドのうち最後の 2 つは数値フィールドです。したがって、このプログラムは、IBM i、AIX, Linux, and Windows・システム用の MQTMC2 構造体で使用するために文字置換を行います。

トリガー・サンプルは、トリガー・メッセージの *ApplicId* フィールドで指定したアプリケーションに開始コマンドを実行し、MQTMC2 または MQTMC (トリガー・メッセージの文字バージョン) 構造体を渡します。

- ALW AIX, Linux, and Windows システムでは、*EnvData* フィールドは呼び出し側のコマンド・ストリングへの拡張として使用されます。
- IBM i IBM i では、このフィールドは、例えばジョブ優先順位やジョブ記述のようなジョブ提示パラメーターとして使用されます。

最後に、このプログラムは開始キューをクローズします。

IBM i でのトリガー・サンプル・プログラムの終了

IBM i

トリガー・モニター・プログラムは、sysrequest オプション 2 (ENDRQS) またはトリガー・キューからの読み取りを禁止することによって終了できます。

サンプル・トリガー・キューを使用すると、コマンドは次のようになります。

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

重要 : このキューでトリガーを再び開始する前に、次のコマンドを入力しなければなりません。

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

トリガー・サンプル・プログラムの実行
このトピックでは、トリガー・サンプル・プログラムの実行について説明します。

amqstrg0.c、amqstrg、および amqstrgc のサンプルの実行

このプログラムは次の 2 つのパラメーターをとります。

1. 開始キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、このプログラムはデフォルトのキュー・マネージャーに接続します。 サンプルの開始キューは、amqscos0.tst を実行する時には既に定義されています。そのキューの名前は SYSTEM.SAMPLE.TRIGGER で、このプログラムを実行する時に使用できます。

注 : このサンプルの関数は、runmqtrm プログラムで提供される完全なトリガー関数のサブセットです。

AMQSTRG4 サンプルの実行

IBM i

このトリガー・モニターは、IBM i 環境で使用されるものです。 このモニターは、開始するアプリケーションごとに 1 つの IBM i ジョブを実行依頼します。 これは、各トリガー・メッセージに関連した追加処理があることを意味します。

AMQSTRG4 (QCSRC の場合) には 2 つのパラメーターがあります。このパラメーターは、処理する開始キューの名前と、キュー・マネージャーの名前 (オプション) です。 AMQSAMP4 (QCLSRC の場合) は、サンプ

ル・プログラムを試行するときに使用できるサンプル開始キュー SYSTEM.SAMPLE.TRIGGER を定義します。

サンプルのトリガー・キューを使用すると、発行するコマンドは次のようになります。

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

または、CL でこれに相当する STRMQMTRM を使用することもできます。詳細については、MQ トリガー・モニターの開始 (STRMQMTRM) を参照してください。

AMQSERV4 サンプルの実行

IBM i

このトリガー・サーバーは、IBM i 環境で使用されるものです。 トリガー・メッセージごとに、それ自身のジョブの中で開始コマンドを実行して、指定されたアプリケーションを開始します。 このトリガー・サーバーは、CICS トランザクションを呼び出すことができます。

AMQSERV4 には 2 つのパラメーターがあります。このパラメーターは、処理する開始キューの名前と、キュー・マネージャーの名前 (オプション) です。 AMQSAMP4 は、サンプル・プログラムを試行するときに使用できるサンプル開始キュー SYSTEM.SAMPLE.TRIGGER を定義します。

サンプルのトリガー・キューを使用する時には、以下のコマンドを発行します。

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

トリガー・サーバーの設計
トリガー・サーバーの設計は、いくつかの例外を除いて、トリガー・モニターの設計と似ています。

トリガー・サーバーの設計は、トリガー・サーバーが次のことを行う以外は、トリガー・モニターと同様です。

- MQAT_CICS と同様に MQAT_OS400 アプリケーションを許可します。
- IBM i IBM i ジョブを実行依頼するのではなく、独自のジョブで IBM i アプリケーションを呼び出す (または STRCICSUSR を使用して CICS アプリケーションを開始する)。
- CICS アプリケーションについては、*EnvData* を代用します。例えば、STRCICSUSR コマンド内のトリガー・メッセージから CICS 領域を指定するような場合です。
- 共用する入力に対して開始キューをオープンし、多数のトリガー・サーバーを同時に実行できるようにします。

注 : AMQSERV4 によって開始されるプログラムは、MQDISC 呼び出しを使用してはなりません。これによってトリガー・サーバーが停止されてしまうからです。 AMQSERV4 によって開始されるプログラムが MQCONN 呼び出しを使用すると、MQRC_ALREADY_CONNECTED 理由コードを受け取ります。

ALW *AIX, Linux, and Windows での TUXEDO サンプルの使用*

TUXEDO の書き込みおよび読み取りサンプル・プログラムおよび TUXEDO でのサーバー環境の構築について学習します。

始める前に

これらのサンプルを実行するには、サーバー環境を構築しておく必要があります。

このタスクについて

注 : このセクション全体で、複数行にまたがる長いコマンドを分割するため、円記号 (¥) が使用されています。 この文字は入力しないでください。 各コマンドは単一行として入力してください。

ALW サーバー環境の構築
さまざまなプラットフォーム用の IBM MQ のサーバー環境を構築する方法について説明します。

始める前に

TUXEDO 作業環境があるものとします。

AIX *AIX のサーバー環境の構築 (32 ビット)*
IBM MQ for AIX (32 ビット) のサーバー環境を構築する方法。

手順

1. サーバー環境が構築されているディレクトリー (例えば、APPDIR) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。ここで、TUXDIR は TUXEDO のルート・ディレクトリーで、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされている上位ディレクトリーを表します。

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib
```

3. 以下の行を TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /APPDIR/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

次のタスク

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

AIX *AIX のサーバー環境の構築 (64 ビット)*

IBM MQ for AIX (64 ビット) のサーバー環境を構築する方法。

手順

1. サーバー環境が構築されているディレクトリー (例えば、APPDIR) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。ここで、TUXDIR は TUXEDO のルート・ディレクトリーを表し、*MQ_INSTALLATION_PATH* は IBM MQ がインストールされている上位ディレクトリーを表します。

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH/inc -I /APPDIR -L MQ_INSTALLATION_PATH/lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS= MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/APPDIR/amqstxvx.V
$ export LIBPATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. 以下の行を TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /APPDIR/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

次のタスク

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

Windows *Windows のサーバー環境の構築 (32 ビット)*
IBM MQ for Windows のサーバー環境の構築 (32 ビット)。

このタスクについて

注 : 以下で *VARIABLES* として示されているフィールドをディレクトリー・パスに変更します。

表 *164.* ディレクトリー・パスに変更するフィールド

フィールド	ディレクトリー・パス
MQMDIR	IBM MQ のインストール時に指定されたディレクトリー・パス (例: `g:\Program Files\IBM\MQ`)。
TUXDIR	TUXEDO がインストールされたときに指定されたディレクトリー・パス (例: `f:\tuxedo`)。
APPDIR	サンプル・アプリケーション用に使用されるディレクトリー・パス。例: `f:\tuxedo\apps\mqapp`

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

図 *134. IBM MQ for Windows* 用 *ubbstxcn.cfg* ファイルの例

注 : マシン名 *MachineName* とディレクトリー・パスをそれぞれご使用のインストール済み環境に合わせて変更します。 また、キュー・マネージャー名 *MYQUEUEMANAGER* も、接続するキュー・マネージャーの名前に変更してください。

IBM MQ for Windows のサンプル ubbconfig ファイルは、1133 ページの図 134 にリストされています。これは、IBM MQ サンプル・ディレクトリーで ubbstxcn.cfg として提供されます。

IBM MQ for Windows に提供されたサンプル Make ファイル (1134 ページの図 135 を参照) は ubbstxmn.mak と呼ばれ、IBM MQ サンプル・ディレクトリーに保持されます。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

図 *135. IBM MQ for Windows* 用サンプル *TUXEDO makefile*

サーバー環境とサンプルを構築するには、次のステップを実行します。

手順

1. 次のようにサンプル・アプリケーションを作成するアプリケーション・ディレクトリーを作成します。

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 以下のサンプル・ファイルを IBM MQ サンプル・ディレクトリーからアプリケーション・ディレクトリーにコピーします。

   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg

3. これらのファイルをそれぞれに編集して、インストール時に使用するディレクトリー名とディレクトリー・パスを設定します。
4. ubbstxcn.cfg を編集し (1133 ページの図 134 を参照)、接続先のマシン名とキュー・マネージャーの詳細を追加します。
5. 以下の行を TUXEDO ファイル *TUXDIR*udataobj\rm に追加します。

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   ファイル内では、新規のエントリーは 1 行でなければなりません。
6. 以下の環境変数を設定します。

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. TUXEDO 用 TLOG デバイスを作成します。

これを行うには、tmadmin -c を呼び出し、次のコマンドを入力します。

```
crdl -z APPDIR\TLOG
```

8. 現行ディレクトリーを *APPDIR* に設定し、サンプル Make ファイル amqstxmn.mak を外部プロジェクト Make ファイルとして呼び出します。 例えば、Microsoft Visual C++ では次のコマンドを実行します。

```
msvc amqstxmn.mak
```

「**build (構築)**」を選択して、すべてのサンプル・プロ グラムを作成します。

Windows *Windows のサーバー環境の構築 (64 ビット)*
IBM MQ for Windows (64 ビット) のサーバー環境を構築する方法。

このタスクについて

注 : 以下で *VARIABLES* として示されているフィールドをディレクトリー・パスに変更します。

表 *165.* ディレクトリー・パスに変更するフィールド

フィールド	ディレクトリー・パス
MQMDIR	IBM MQ のインストール時に指定されたディレクトリー・パス (例: g:\Program Files\IBM\MQ)。
TUXDIR	TUXEDO がインストールされたときに指定されたディレクトリー・パス (例: f:\tuxedo)。
APPDIR	サンプル・アプリケーション用に使用されるディレクトリー・パス。例: f:\tuxedo\apps\mqapp

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

図 *136. IBM MQ for Windows* 用 *ubbstxcn.cfg* ファイルの例

注 : マシン名 *MachineName* とディレクトリー・パスをそれぞれご使用のインストール済み環境に合わせて変更します。 また、キュー・マネージャー名 *MYQUEUEMANAGER* も、接続するキュー・マネージャーの名前に変更してください。

IBM MQ for Windows 用のサンプル ubbconfig ファイルのリストを、1136 ページの図 136 に示します。 これは、IBM MQ サンプル・ディレクトリーで ubbstxcn.cfg として提供されます。

IBM MQ for Windows に提供されたサンプル Make ファイル (1137 ページの図 137 を参照) は ubbstxmn.mak と呼ばれ、IBM MQ サンプル・ディレクトリーに保持されます。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

図 *137. IBM MQ for Windows* 用サンプル *TUXEDO makefile*

サーバー環境とサンプルを構築するには、次のステップを実行します。

手順

1. 次のようにサンプル・アプリケーションを作成するアプリケーション・ディレクトリーを作成します。

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 以下のサンプル・ファイルを IBM MQ サンプル・ディレクトリーからアプリケーション・ディレクトリーにコピーします。

   - amqstxmn.mak
   - amqstxen.env
   - ubbstxcn.cfg

3. これらのファイルをそれぞれに編集して、インストール時に使用するディレクトリー名とディレクトリー・パスを設定します。
4. ubbstxcn.cfg を編集し (1136 ページの図 136 を参照)、接続先のマシン名とキュー・マネージャーの詳細を追加します。
5. 以下の行を TUXEDO ファイル *TUXDIR*udataobj\rm に追加

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   ファイル内では、新規のエントリーは 1 行でなければなりません。
6. 以下の環境変数を設定します。

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. TUXEDO 用 TLOG デバイスを作成します。 これを行うには、tmadmin -c を呼び出し、次のコマンドを入力します。

```
crdl -z APPDIR\TLOG
```

8. 現行ディレクトリーを *APPDIR* に設定し、サンプル Make ファイル `amqstxmn.mak` を外部プロジェクト Make ファイルとして呼び出します。 例えば、Microsoft Visual C++ では次のコマンドを実行します。

```
msvc amqstxmn.mak
```

「**build (構築)**」を選択して、すべてのサンプル・プロ グラムを作成します。

ALW *TUXEDO* のサンプル・サーバー・プログラム

サンプル・サーバー・プログラム (amqstxsx) は、書き込み (amqstxpx.c) および読み取り (amqstxgx.c) サンプル・プログラムと共に実行するように設計されています。 サンプル・サーバー・プログラムは、TUXEDO を始動すると自動的に実行されます。

注 : TUXEDO を始動する前にキュー・マネージャーを開始する必要があります。

このサンプル・サーバーでは 2 つの TUXEDO サービス MPUT1 と MGET1 を提供しています。

- MPUT1 サービスは書き込みサンプルによって作動し、同期点で MQPUT1 を使用して、TUXEDO によって制御される作業単位にメッセージを書き込みます。 このサービスでは、書き込みサンプルによって提供されるパラメーター QName および Message Text をとります。
- MGET1 サービスは、メッセージを取得するたびにキューをオープン、およびクローズします。 このサービスでは、読み取りサンプルによって提供されるパラメーター QName および Message Text をとります。

エラー・メッセージ、理由コード、および状況メッセージはすべて、TUXEDO ログ・ファイルに書き込まれます。

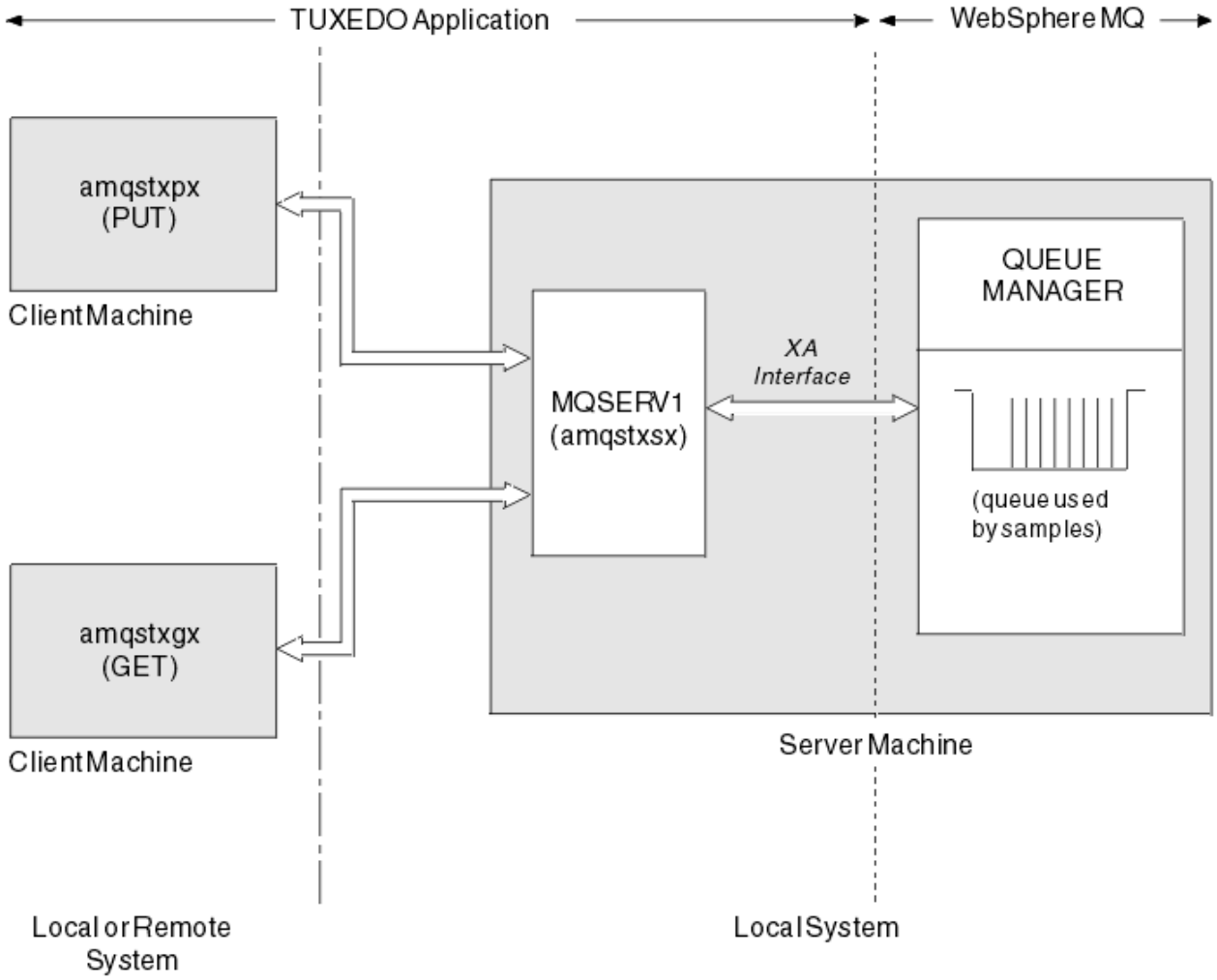


図 *138. TUXEDO* サンプルの処理の流れ

ALW *TUXEDO* の書き込みサンプル・プログラム

このサンプルにより、キューに複数回メッセージを書き込むことができます。バッチでは、リソース・マネージャーとして TUXEDO を使用した同期点を示すことができます。

書き込みサンプルが正常に動作するには、サンプル・サーバー・プログラム `amqstxsx` が実行されていなければなりません。このサーバー・サンプル・プログラムはキュー・マネージャーへの接続を行い、XA インターフェースを使用します。 このサンプルを実行するには、次のように入力します。

- `doputs -n queuename -b batchsize -c trancount -t message`

以下に例を示します。

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

これによって、6 つのバッチでそれぞれ 5 個ずつ、合計で 30 個のメッセージを `myqueue` という名前のキューに入れることができます。 何らかの問題があれば、そのバッチ単位でメッセージを元に戻し、問題がなければ、それらのメッセージをコミットします。

エラー・メッセージはすべて、TUXEDO ログ・ファイルおよび stderr に書き込まれます。 理由コードはすべて、stderr に書き込まれます。

ALW *TUXEDO* の読み取りサンプル

このサンプルを使用すると、バッチ内のキューからメッセージを取得することができます。

GET サンプルを成功させるには、サンプル・サーバー・プログラム amqstxsx を実行する必要があります。 このサンプル・サーバー・プログラムはキュー・マネージャーに接続し、XA インターフェースを使用します。 サンプルを実行するには、以下のコマンドを入力します。

- dogets -n queuename -b batchsize -c trancount

以下に例を示します。

- dogets -n myqueue -b 6 -c 4

これによって、myqueue という名前のキューから、6 つのバッチでそれぞれ 4 個ずつ、合計で 24 個のメッセージを取り出すことができます。 30 個のメッセージを myqueue に入れる書き込みサンプル・プログラムを実行したあとでこのサンプル・プログラムを実行すると、myqueue には 6 個のメッセージしか残りません。 バッチ数とバッチ・サイズがメッセージの書き込みと読み取りとで異なっても差し支えありません。

エラー・メッセージはすべて、TUXEDO ログ・ファイルおよび stderr に書き込まれます。 理由コードはすべて、stderr に書き込まれます。

Windows *Windows での SSPI セキュリティー出口の使用*

このトピックでは、Windows システムで SSPI チャネル出口プログラムを使用する方法について説明します。 付属する出口コードの形式は、オブジェクトとソースの 2 つです。

オブジェクト・コード

オブジェクト・コード・ファイルは amqrspin.dll と呼ばれます。 お客様およびサーバーの両方の場合、それは IBM MQ for Windows の標準パーツとして *MQ_INSTALLATION_PATH*/exits/*INSTALLATION_NAME* フォルダーにインストールされます。 例えば、C:¥Program Files¥IBM¥MQ\exits\installation2 などです。 これは、標準のユーザー出口としてロードされます。 提供されるセキュリティー・チャネル出口を実行して、チャネル定義にある認証サービスを使用することができます。

これを行うには、次のいずれかを指定してください。

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

制限付きチャネルのサポートを提供するには、SVRCONN チャネルで以下を指定します。

```
SCYDATA('remote_principal_name')
```

remote_principal_name は、*DOMAIN*¥*user* という形式になります。 ソース・チャネルは、リモート・プリンシパルの名前が *remote_principal_name* と一致する場合にのみ確立されます。

Kerberos セキュリティー・ドメイン内で稼働するシステム間で、提供されているチャネル出口プログラムを使用するには、キュー・マネージャー用に **servicePrincipalName** を作成してください。

ソース・コード

出口ソース・コード・ファイルは amqsspin.c と呼ばれます。 これは C:¥Program Files¥IBM¥MQ\Tools\c\Samples に存在します。

ソース・コードを変更するには、変更されたソースを再コンパイルしなければなりません。

このソースのコンパイルとリンクは、関連したプラットフォームの他のチャネル出口と同じように行います。ただし、SSPI ヘッダーはコンパイル時にアクセスする必要があり、SSPI セキュリティー・ライブラリーは関連した他の推奨ライブラリーとともにリンク時にアクセスする必要があります。

以下のコマンドを実行する前に、cl.exe、Visual C++ ライブラリー、および include フォルダーがパス内で使用可能であることを確認してください。 以下に例を示します。

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

注 : ソース・コードにはトレースやエラー処理のためのものが組み込まれていません。 ソース・コードを変更して使用する場合、独自のトレースおよびエラー処理ルーチンを追加してください。

リモート・キューを使用するサンプルの実行

接続されたキュー・マネージャーでサンプルを実行することにより、リモート・キューイングを実証できます。

プログラム amqscos0.tst は、OTHER という名前のリモート・キュー・マネージャーを使用するリモート・キュー (SYSTEM.SAMPLE.REMOTE) のローカル定義を行います。 このサンプル定義を使用するには、OTHER を 2 番目に使用したいキュー・マネージャーの名前に変更します。 また、2 つのキュー・マネージャーの間のメッセージ・チャネルも設定する必要があります。この方法については、チャネルの定義を参照してください。

要求サンプル・プログラムは、それ自体のローカル・キュー・マネージャー名を、自らが送信するメッセージの *ReplyToQMgr* フィールドに入れます。 照会サンプルおよび設定サンプルは、この 2 つのサンプルが処理する要求メッセージの *ReplyToQ* フィールドおよび *ReplyToQMgr* フィールドで名前が指定されたキューおよびメッセージ・キュー・マネージャーに応答メッセージを送信します。

クラスター・キュー・モニターのサンプル・プログラム *(AMQSCLM)*

このサンプルでは、IBM MQ に組み込まれたクラスター・ワークロード・バランシング機能を使用して、コンシューム側アプリケーションが接続されているキューのインスタンスにメッセージを送信します。 この自動送信により、コンシューム側アプリケーションが接続されていないクラスター・キューのインスタンスにメッセージが蓄積されることを防ぎます。

概要

複数のキュー・マネージャーで同じキューに対して異なる定義を使用するクラスターをセットアップできます。 この構成は、可用性およびワークロード・バランシングの改善という利点をもたらします。 ただし、IBM MQ には、接続されているアプリケーションの状態に応じて動的にクラスター全体にメッセージを分散させる機能は組み込まれていません。 このことから、確実にメッセージが処理されるようにするには、常に、キューのあらゆるインスタンスにコンシューム側アプリケーションを接続しなければなりません。

クラスター・キュー・モニター・サンプル・プログラムは、接続されているアプリケーションの状態をモニターします。 このプログラムは、コンシューム側アプリケーションが接続されているクラスター・キューのインスタンスにメッセージを送信するために、組み込みワークロード・バランシング構成を動的に調整します。 特定の状況では、このプログラムを使用することにより、常にキューのあらゆるインスタンスにコンシューム側アプリケーションを接続しなければならないという必要性を緩和できます。 また、コンシューム側アプリケーションが接続されていないキューのインスタンスに入れられた状態のメッセージは再送されます。 メッセージの再送により、一時的にシャットダウンされているコンシューム側アプリケーションにもメッセージをルーティングできます。

このプログラムの使用が意図されているのは、コンシューム側アプリケーションが頻繁に接続されて切り離されるのではなく、長時間実行される場合です。

クラスター・キュー・モニター・サンプル・プログラムは、C サンプル・ファイル amqsclma.c のコンパイル済みの実行可能プログラムです。

クラスターおよびワークロードについて詳しくは、クラスターによるワークロードの管理を参照してください。

AMQSCLM: サンプルを使用するための設計と計画

クラスター・キュー・モニター・サンプル・プログラムの動作、サンプル・プログラムを実行するシステムをセットアップするときに考慮すべき点、およびサンプル・ソース・コードをどのように変更できるかについて説明します。

設計

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションが接続されているローカル・クラスター・キューをモニターします。 プログラムは、ユーザーによって指定されたキューをモニターします。 キューの名前は、APP.TEST01 のように固有にすることも、または総称にすることもできます。 総称名は、PCF (プログラマブル・コマンド・フォーマット) に準拠する形式でなければなりません。 総称名の例としては、APP.TEST* や APP* などがあります。

クラスター内でモニター対象のローカル・キューのインスタンスを所有する各キュー・マネージャーには、クラスター・キュー・モニター・サンプル・プログラムの 1 つのインスタンスを接続する必要があります。

動的メッセージ・ルーティング

クラスター・キュー・モニター・サンプル・プログラムは、あるキューのコンシューマーがいるかどうかを、そのキューの **IPPROCS** 値 (入力処理カウント用に開かれる) を使用して判別します。 値が 0 より大きいときは、キューに少なくとも 1 つのコンシューム側アプリケーションが接続していることを示します。 この場合、キューはアクティブです。 値が 0 であるときは、キューに接続しているコンシューム側プログラムがないことを示します。 この場合、キューは非アクティブです。

クラスター内に複数のインスタンスがあるクラスター・キューの場合、IBM MQ では、各キュー・インスタンスのクラスター・ワークロード優先順位プロパティー **CLWLPRTY** を使用して、メッセージを送信する宛先のインスタンスが決定されます。 IBM MQ は、 **CLWLPRTY** 値が最も高いキューの使用可能なインスタンスにメッセージを送信します。

クラスター・キュー・モニター・サンプル・プログラムは、ローカル **CLWLPRTY** の値を 1 に設定することによって、クラスター・キューをアクティブ化します。 このプログラムは、**CLWLPRTY** 値を 0 に設定することにより、クラスター・キューを非アクティブ化します。

IBM MQ クラスタリング・テクノロジーは、クラスター・キューの更新された **CLWLPRTY** プロパティーを、クラスター内の関連するすべてのキュー・マネージャーに伝搬します。 例:

- メッセージをキューに書き込むアプリケーションが接続されているキュー・マネージャー
- 同じクラスター内に同じ名前のローカル・キューを所有するキュー・マネージャー

この伝搬は、クラスターのフル・リポジトリー・キュー・マネージャーを使用して行われます。 クラスター・キューの新規メッセージは、クラスター内で最も大きい **CLWLPRTY** 値を持つインスタンスに送信されます。

既にキューに入れられたメッセージの転送

CLWLPRTY の値を動的に変更した場合、この変更は新規メッセージのルーティングに適用されます。 コンシューマーが接続されていないキュー・インスタンスのキューに既に入れられているメッセージ、または **CLWLPRTY** 値の変更がクラスター内に伝搬される前にワークロード・バランシング・メカニズムを経由したメッセージには、この動的な変更が適用されません。 その結果、非アクティブなキューにあるメッセージは、コンシューム側アプリケーションによって処理されることなく、そのまま残ってしまいます。 この問題を解決するために、クラスター・キュー・モニター・サンプル・プログラムでは、コンシューマーを持たないローカル・キューのメッセージを取得して、そのメッセージを同じキューのコンシューマーが接続されているリモート・インスタンスに送信できます。

クラスター・キュー・モニター・サンプル・プログラムは、一度メッセージを取得して (**MQGET** を使用する) からそのメッセージを同じクラスター・キューに書き込む (**MQPUT** を使用する) という方法で、非アクティブなローカル・キューのメッセージを 1 つ以上のアクティブなリモート・キューに転送します。 この方法で転送されることで、IBM MQ クラスター・ワークロード管理によって、ローカル・キュー・インスタンスよりも大きい CLWLPRTY 値を持つ別のターゲット・インスタンスが選択されます。 メッセージが転送されても、メッセージのパーシスタンスとコンテキストは保持されます。 メッセージの順序とバインディング・オプションは保持されません。

計画

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションの接続性に変更があったとき、クラスター構成を変更します。 変更は、クラスター・キュー・モニター・サンプル・プログラムがキューをモニターしているキュー・マネージャーから、クラスター内のフル・リポジトリー・キュー・マネージャーに伝送されます。 フル・リポジトリー・キュー・マネージャーは構成の更新を処理し、その情報をクラスター内の関係するすべてのキュー・マネージャーに再送します。 関係するキュー・マネージャーに含まれるのは、同じ名前のクラスター・キューを所有するキュー・マネージャー (クラスター・キュー・モニター・サンプル・プログラムのインスタンスが実行中)、およびアプリケーションが過去 30 日間にクラスター・キューを開いてメッセージを書き込んだキュー・マネージャーです。

変更はクラスター内で非同期に処理されます。 そのため、変更が行われると、ある期間、クラスター内で認識される構成がキュー・マネージャーごとに異なるという状況も起こりえます。

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションの接続と切り離しが頻繁には行われない、コンシューム側アプリケーションが長期間継続して実行されるようなシステムにのみ適しています。 コンシューム側アプリケーションが短期間だけ接続されるようなシステムのモニターに使用すると、構成更新の配布に要する待ち時間のために、コンシューマーが接続されているキューをクラスター内のキュー・マネージャー側で正しく認識できなくなってしまう可能性があります。 この待ち時間によって、メッセージが正しくルーティングされない事態も発生しかねません。

多数のキューをモニターしている場合、すべてのキューで、接続されているコンシューマーでの変更の速度が比較的遅くなり、これによりクラスター内のクラスター構成トラフィックが増加します。 クラスター構成トラフィックが増えると、以下の 1 つ以上キュー・マネージャーに過大な負荷がかかってしまうことがあります。

- クラスター・キュー・モニター・サンプル・プログラムを実行中のキュー・マネージャー
- フル・リポジトリー・キュー・マネージャー
- メッセージをキューに書き込むアプリケーションが接続されているキュー・マネージャー
- 同じクラスター内に同じ名前のローカル・キューを所有するキュー・マネージャー

フル・リポジトリー・キュー・マネージャーのプロセッサー使用量を評価する必要があります その他のプロセッサー使用量は、フル・リポジトリー・キュー SYSTEM.CLUSTER.COMMAND.QUEUE のメッセージ・トラフィックによって把握できます。 このキューにメッセージがたまっている場合は、フル・リポジトリー・キュー・マネージャーがシステムのクラスター構成変更の速度のペースに合わせることができていないことを示しています。

クラスター・キュー・モニター・サンプル・プログラムが多数のキューをモニターしている場合、サンプル・プログラムとキュー・マネージャーによって実行される一定量の作業があります。 この作業は、接続されているコンシューマーに何も変更がなくても実行されるものです。 **-i** 引数を変更して、モニターの周期の頻度を減らし、ローカル・システムでサンプル・プログラムのプロセッサー使用量を減らすことができます。

過大なアクティビティーを検出するための助けとして、クラスター・キュー・モニター・サンプル・プログラムはポーリング間隔ごとの平均処理時間、処理経過時間、および構成変更の数のレポートを報告します。 レポートは情報メッセージ **CLM0045I** として、30 分または 600 ポーリングのどちらか早いほうの間隔で送信されます。

クラスター・キュー・モニター使用の要件

クラスター・キュー・モニター・サンプル・プログラムには要件と制限があります。 提供されているサンプル・ソース・コードは、その使用方法に関する制限の一部を変更することができます。 このセクションに挙げた例で、加えることができる変更について詳しく説明します。

- クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションが接続されている、またはされていない状態にあるキューをモニターする用途で設計されています。 システムでコンシューム側アプリケーションの接続と切り離しが頻繁に行われると、 サンプル・プログラムはクラスター全体に過大なクラスター構成アクティビティーを生じさせる可能性があります。 このことは、クラスター内のキュー・マネージャーのパフォーマンスに影響を与えかねません。

- クラスター・キュー・モニター・サンプル・プログラムは、基礎となる IBM MQ システムおよびクラスター・テクノロジーに依存しています。 モニターするキューの数、モニターの頻度、および各キューの状態が変化する頻度は、システム全体にかかる負荷に影響します。 モニターするキューおよびモニターのポーリング間隔を選択する際には、これらの要素を考慮してください。
- クラスター内のモニター対象のキューのインスタンスを所有するすべてのキュー・マネージャーには、クラスター・キュー・モニター・サンプル・プログラムの 1 つのインスタンスが接続している必要があります。 サンプル・プログラムを、クラスター内のキューを所有しないキュー・マネージャーに接続する必要はありません。
- クラスター・キュー・モニター・サンプル・プログラムを適切な権限で実行して、必要な IBM MQ リソースすべてにアクセスできるようにします。 例:
  - 接続先のキュー・マネージャー
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - メッセージ転送を実行するときにモニターするすべてのキュー
- コマンド・サーバーは、各キュー・マネージャーに対して、クラスター・キュー・モニター・サンプル・プログラムが接続された状態で実行する必要があります。
- クラスター・キュー・モニター・サンプル・プログラムの各インスタンスが接続先のキュー・マネージャー上のローカル (非クラスター) キューを排他使用できることが必要です。 このローカル・キューは、サンプル・プログラムの制御、およびキュー・マネージャーのコマンド・サーバーに対して実行した照会の応答メッセージを受信するために使用されます。
- クラスター・キュー・モニター・サンプル・プログラムの単一インスタンスがモニターするすべてのキューは、同一のクラスター内になければなりません。 キュー・マネージャーが持つモニター対象のキューが複数のクラスターに分散している場合は、サンプル・プログラムの複数インスタンスが必要です。 それらの各インスタンスに、制御および応答メッセージ用のローカル・キューが必要です。
- モニターするキューはすべてが単一のクラスター内になければなりません。 クラスター名前リストを使用するように構成されたキューは、モニターされません。
- 非アクティブ・キューからのメッセージ転送を有効にするかどうかはオプションです。 この指定は、クラスター・キュー・モニター・サンプル・プログラムの特定のインスタンスがモニターするすべてのキューに適用されます。 モニター対象のキューの一部だけにメッセージ転送を有効にする必要がある場合は、クラスター・キュー・モニター・サンプル・プログラムのインスタンスが 2 つ必要になります。 一方のサンプル・プログラムでメッセージ転送を有効にし、もう一方のサンプル・プログラムでメッセージ転送を無効にします。 このとき、サンプル・プログラムの各インスタンスに、制御および応答メッセージ用のローカル・キューが必要です。
- デフォルトでは、IBM MQ クラスター・ワークロード・バランシングが、書き込みアプリケーションが接続されている同じキュー・マネージャーにある、クラスター・キューのインスタンスにメッセージを送信します。 次の環境で、ローカル・キューが非アクティブである間は、この機能を無効にしてください。
  - 書き込みアプリケーションが、モニター対象である非アクティブなキューのインスタンスを所有するキュー・マネージャーに接続されている
  - キューに入れられたメッセージが、非アクティブなキューのからアクティブなキューに転送中である

  CLWLUSEQ 値を ANY に設定することにより、キューのワークロード・バランシングのローカル設定を静的に無効にすることができます。 この構成の場合、ローカルのコンシューム側アプリケーションがあっても、ローカル・キューに書き込まれたメッセージをローカル・キュー・インスタンスとリモート・キュー・インスタンスに分散してワークロードのバランスを取ります。 または、クラスター・キュー・モニター・サンプル・プログラムの構成を一時的に変更し、コンシューマーがキューに接続されていない間、**CLWLUSEQ** 値を ANY に設定することもできます。結果として、あるキューがアクティブである間、ローカル・メッセージだけがそのキューのローカル・インスタンスに送信されます。
- IBM MQ システムおよびアプリケーションが、モニター対象のキューまたは使用中のチャネルに **CLWLPRTY** を使用することはできません。 使用した場合は、**CLWLPRTY** キュー属性に対するクラスター・キュー・モニター・サンプル・プログラムのアクションによって、望ましくない結果が生じる可能性があります。
- クラスター・キュー・モニター・サンプル・プログラムは、ランタイム情報をログに記録して、一式のレポート・ファイルを作成します。 これらのレポートを保管するディレクトリーが必要であり、このディ

レクトリーに書き込む権限をクラスター・キュー・モニター・サンプル・プログラムが持っている必要があります。

AMQSCLM: サンプルの準備と実行

クラスター・キュー・モニター・サンプルは、キュー・マネージャーにローカルに接続して実行することも、チャネルを介して接続したクライアントとして実行することもできます。 このサンプルはキュー・マネージャーが実行されるたびに実行する必要があり、ローカルに実行する場合には、キュー・マネージャーを使用してサンプルを自動的に開始および停止するキュー・マネージャー・サービスとして構成できます。

始める前に

クラスター・キュー・モニター・サンプルを実行する前に、以下のステップを実行する必要があります。

1. サンプルの内部使用のために、それぞれのキュー・マネージャーで作業キューを作成します。

   サンプルのそれぞれのインスタンスには、排他的な内部使用のために、ローカルの非クラスター・キューが必要です。 キューの名前は選択できます。 次の例では、名前は AMQSCLM.CONTROL.QUEUE を使用します。 例えば、Windows の場合、このキューは次の **MQSC** コマンドを使用して作成できます。

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   MAXDEPTH と **MAXMSGL** の値はデフォルトのまま使用できます。

2. エラー・ログと情報メッセージ・ログのためにディレクトリーを作成します。

   サンプルは、レポート・ファイルに診断メッセージを書き込みます。 ユーザーは、それらのファイルを保管するディレクトリーを選択する必要があります。 例えば、Windows の場合は、次のコマンドを使用してディレクトリーを作成できます。

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   サンプルが作成するレポート・ファイルには、次の命名規則があります。

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (オプション) クラスター・キューモニター・サンプルを IBM MQ サービスとして定義します。

   キューをモニターするには、サンプルは常に実行されている必要があります。 クラスター・キュー・モニター・サンプルが常に実行されるようにするには、サンプルをキュー・マネージャー・サービスとして定義します。 サンプルをサービスとして定義するということは、キュー・マネージャーが開始すると AMQSCLM も開始することを意味します。 次の例を使用すると、クラスター・キュー・モニター・サンプルを IBM MQ サービスとして定義できます。

   ```
   define service(AMQSCLM) +
         descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
         control(qmgr) +
         servtype(server) +
         startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
         startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
   c:\AMQSCLM\rpts') +
         stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
         stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

定義	説明
service	サービス名を指定します。 サービス名は選択できます。
descr	サービスのテキスト記述を指定します。
control	サービスがキュー・マネージャーと同時に開始および停止することを示します。

定義	説明
servtype	このキュー・マネージャーに対して、一度に 1 つのサーバー・サービス・オブジェクト (つまり、1 つのインスタンスのみ) が実行可能であることを示します。
startcmd	プログラムの場所と名前を指定します。
startarg	サンプルの引数を指定します。 *+QMNAME+* が使用されているため、キュー・マネージャーの名前は自動的に置き換わります。
stdout	標準の出力がリダイレクトされる完全修飾ファイル名です。 サンプルがこのファイルに書き込むのは、サンプルが終了したことを確認するメッセージのみです。 サンプルでこれを実行しているのは、サンプル終了プロセスの初期段階で、標準エラー・ファイルが既に閉じられているためです。
stderr	標準のエラー出力がリダイレクトされる完全修飾ファイル名です。 サンプルは標準エラー・ファイルに、サンプルが終了する前のエラー・メッセージを書き込みます。

このタスクについて

このタスクでは、さまざまな方法でクラスター・キュー・モニター・サンプルを開始および停止できます。また、モニター対象のキューの統計情報を含むレポート・ファイルを生成するモードで、サンプルを実行することもできます。

サンプル・プログラムは、次のコマンドを使用して実行できます。

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

次の表では、クラスター・キュー・モニター・サンプルに使用できる引数とその詳細をリストします。

引数	変数	詳細
-m	**QMgrName**	モニター対象のキュー・マネージャー。
-c	**ClusterName**	モニター対象のキューが含まれるクラスター。
-q	**QNameMask**	モニター対象のキュー (複数の場合あり)。 末尾の * は、0 文字以上の末尾文字と一致する名前を持つすべてのキューをモニターします。
-f	**QListFile**	モニター対象のキュー名またはキュー名マスクのリストを含むファイルの、絶対パスおよびファイル名。 ファイルの各行には 1 つのキュー名/マスクが含まれている必要があります。 **-q** または **-f** のいずれかを指定できますが、両方は指定できません。
-r	**MonitorQName**	サンプルが排他的に使用しているローカル・キュー。
-l	**ReportDir**	ログに記録された情報メッセージを一連のラップに保管するディレクトリー・パス [9] レポート・ファイル。
-t		(オプション) キューに入れられたメッセージを、非アクティブなローカル・キューからアクティブなキューに転送できるようにします。 転送を有効にしないと、クラスターに入れられる新規のメッセージのみが、キューのアクティブなインスタンスに動的に経路指定されます。
-u	**ActiveVal**	(オプション) モニター対象のキュー・インスタンスが非アクティブな場合は、その **CLWLUSEQ** プロパティーを ANY に、アクティブな場合は **ActiveVal** の値に自動的に切り替えます。 **ActiveVal** には、LOCAL または QMGR のいずれかを指定できます。 この引数が、書き込み側のアプリケーションが同じキュー・マネージャーに接続されているシステム、またはメッセージの転送が有効なシステムで設定されていない場合、モニター対象のキューは **CLWLUSEQ** の値が ANY に設定されているか、QMGR に設定されていて ANY に設定されているキュー・マネージャーを使用する必要があります。
-i	**Interval**	(オプション) モニターがキューを確認する時間間隔 (秒)。 デフォルトは 300 秒 (5 分) です。
-d		(オプション) 追加の診断が出力されるようにします。 デバッグ出力は、システムを最初に構成するとき、またはサンプル・コードを操作するときに役立つ場合があります。
-s		(オプション) 各間隔についての最小限の統計情報が出力されるようにします。
-v		(オプション) レポート・ファイルに加えて、レポート情報を standard out に記録します。

引数リストの例:

```
-m QMGR1 -c CLUS1   -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1   -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

キュー・リスト・ファイルの例:

```
Q1
QUEUE.*
```

[9] キュー・マネージャーとキューの組み合わせごとに、固定サイズのログ・ファイルが生成されます。これは、いっぱいになったら、上書きされます。 ロガーは常に同じファイルに書き込み、そのファイルの前の 2 つのバージョンも保持します。

```
ABC
ABD
```

手順

1. クラスター・キュー・モニター・サンプルを開始します。 サンプルは、以下のいずれかの方法で開始できます。
   - 適切なユーザー権限でコマンド・プロンプトを使用する。
   - サンプルが IBM MQ サービスとして構成されている場合は、MQSC **START SERVICE** コマンドを使用します。

   引数リストは、どちらの場合も同じです。

   サンプルは、プログラムが初期化されてから 10 秒間はキューのモニターを開始しません。 コンシューマー・アプリケーションは、この遅延によって最初にモニター対象のキューに接続できるため、キューをアクティブな状態に変更する無駄を回避できます。
2. クラスター・キュー・モニター・サンプルを停止します。 サンプルは、キュー・マネージャーが停止したり、停止中または静止中であったり、キュー・マネージャーへの接続が切断されたりすると自動的に停止します。 サンプルは、以下の方法で、キュー・マネージャーを終了させることなく停止することができます。
   - サンプルが排他的に使用しているローカル・キューを、Get 機能が使用できないように構成する。
   - サンプルが排他的に使用しているローカル・キューに、**CorrelId** が "STOP CLUSTER MONITOR¥0¥0¥0¥0" のメッセージを送信する。
   - サンプル・プロセスを終了する。 これは、アクティブなキューに転送されている非永続メッセージの損失をもたらす可能性があります。 また、これにより、サンプルが使用しているローカル・キューが、終了後の数秒間オープンの状態になる可能性があります。 この状況では、クラスター・キュー・モニター・サンプルの新規のインスタンスは、直ちに開始することができません。

   サンプルが IBM MQ サービスとして開始された場合、**STOP SERVICE** は何の効果も及ぼしません。 その場合は、キュー・マネージャー内の構成済みの **STOP SERVICE** メカニズムとして記述される終了メソッドの 1 つを使用できます。

次のタスク

サンプルの状況を確認します。

レポート作成を有効にした場合は、レポート・ファイルで状況を確認できます。 最新のレポート・ファイルを確認するには、次のコマンドを使用します。

```
QMgrName.ClusterName.RPT01.LOG
```

古いレポート・ファイルを確認するには、次のコマンドを使用します。

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

レポート・ファイルは、最大サイズ (約 1 MB) まで拡大します。 RPT01 ファイルがいっぱいになると、新しい RPT01 ファイルが作成されます。 古い RPT01 ファイルは、RPT02 に名前変更されます。 RPT02 は、RPT03 に名前変更されます。 古い RPT03 は廃棄されます。

サンプルは、以下の状況で情報メッセージを作成します。

- 開始時
- 終了時
- キューを、**ACTIVE** または **INACTIVE** とマーク付けするとき
- メッセージを、非アクティブなキューからアクティブなインスタンスにリキューするとき

サンプルは、注意が必要な問題を報告する場合は、エラー・メッセージ *CLMnnnnE* を作成します。

サンプルは、30 分おきに、各ポーリング間隔の平均処理時間と経過した処理時間を報告します。 この情報は、メッセージ CLM0045I に含まれます。

統計メッセージが有効 (**-s**) な場合、サンプルは、それぞれのキュー・チェックについて以下の情報を報告します。

- キューの処理にかかった時間 (ミリ秒)
- 確認されたキューの数
- アクティブ/非アクティブへの変更の数
- 転送されたメッセージの数

この情報は、メッセージ CLM0048I で報告されます。

デバッグ・モードでは、レポート・ファイルは急速に拡大して、短期間で循環する場合があります。 この状況では、ファイルは 1 MB のサイズ制限を超える場合があります。

AMQSCLM: トラブルシューティング
以下のセクションでは、サンプルの使用時に起こる可能性のあるシナリオについて記載します。 各シナリオについての可能性のある説明、およびその解決方法の選択肢について説明します。

シナリオ: AMQSCLM が開始しない

可能性のある説明: 構文が正しくない。

アクション: 標準のエラー出力に正しい構文が書き込まれているか確認してください。

可能性のある説明: キュー・マネージャーが使用可能ではない。

アクション: レポート・ファイルに、メッセージ ID CLM0010E が書き込まれていないか確認してください。

可能性のある説明: レポート・ファイル (複数の場合あり) を開けない、または作成できない。

アクション: 標準のエラー出力に、初期化時のエラー・メッセージが書き込まれていないか確認してください。

シナリオ: AMQSCLM がキューを ACTIVE または INACTIVE に変更しない

可能性のある説明: キューがモニター対象のキューのリストにない。

アクション: **-q** および **-f** パラメーターの値を確認してください。

可能性のある説明: キューが適切なクラスター内のローカル・キューではない。

アクション: キューが適切なクラスター内のローカル・キューであることを確認してください。

可能性のある説明: AMQSCLM がこのキュー・マネージャーとクラスターに対して実行されていない。

アクション: AMQSCLM を、関連するキュー・マネージャーとクラスターに対して開始してください。

可能性のある説明: キューにはコンシューマーが存在しないため、キューは INACTIVE、**CLWLPRTY**=0 の状態である。 または、コンシューマーが少なくとも 1 つ存在するため、ACTIVE、**CLWLPRTY** >=1 の状態である。

アクション: コンシューマー・アプリケーションがキューに接続されていることを確認してください。

可能性のある説明: キュー・マネージャーのコマンド・サーバーが実行されていない。

アクション: レポート・ファイルにエラーが書き込まれていないか確認してください。

シナリオ: メッセージが INACTIVE キューに対して経路指定されていない

可能性のある説明: メッセージが非アクティブなキューを所有するキュー・マネージャーに直接書き込まれている。また、キューの **CLWLUSEQ** 値が ANY ではなく、AMQSCLM に対して **-u** 引数が使用されていない。

アクション: 関連するキュー・マネージャーの **CLWLUSEQ** 値を確認するか、AMQSCLM に対して **-u** 引数が使用されるようにしてください。

可能性のある説明: いずれのキュー・マネージャーにもアクティブなキューがない。 キューがアクティブになるまで、メッセージに対してすべての非アクティブなキューの間での均一なワークロード・バランシングが行われます。

アクション: すべてのキュー・マネージャーのキューの状況を確認してください。

可能性のある説明: メッセージが非アクティブなキューを所有するクラスター内の異なるキュー・マネージャーに書き込まれている。また、更新された **CLWLPRTY** 値 0 が、書き込み側のアプリケーションのキュー・マネージャーに伝搬されていない。

アクション: モニター対象のキュー・マネージャーとフル・リポジトリー・キュー・マネージャー間のクラスター・チャネルが実行されていることを確認してください。 また、書き込み側のキュー・マネージャーとフル・リポジトリー・キュー・マネージャー間のチャネルが実行されていることを確認してください。 さらに、モニター対象のキュー・マネージャー、書き込み側のキュー・マネージャー、およびフル・リポジトリー・キュー・マネージャーのエラー・ログを確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブである (CLWLPRTY=1) が、ローカル・キュー・マネージャーからのクラスター送信側チャネルが実行されていないため、それらのキュー・インスタンスにメッセージを経路指定できない。

アクション: アクティブなキュー・インスタンスがあるリモート・キュー・マネージャー (1 つ以上) への、ローカル・キュー・マネージャーからのクラスター送信側チャネルの状況を確認してください。

シナリオ: AMQSCLM が非アクティブなキューからメッセージを転送しない

可能性のある説明: メッセージの転送が有効 (**-t**) になっていない。

アクション: メッセージの転送が有効 (**-t**) になっていることを確認してください。

可能性のある説明: キューがモニター対象のキューのリストにない。

アクション: **-q** および **-f** パラメーターの値を確認してください。

可能性のある説明: AMQSCLM がこのキュー・マネージャー、または同じキューのインスタンスを所有するクラスター内の他のキュー・マネージャーに対して実行されていない。

アクション: AMQSCLM を開始してください。

可能性のある説明: キューが **CLWLUSEQ**=LOCAL または **CLWLUSEQ**=QMGR を持っている。また、**-u** 引数が設定されていない。

アクション: **-u** パラメーターを設定するか、キューまたはキュー・マネージャーの構成を ANY に変更してください。

可能性のある説明: クラスター内にキューのアクティブなインスタンスがない。

アクション: **CLWLPRTY** 値が 1 以上のキューのインスタンスがあるか確認してください。

可能性のある説明: リモート・キュー・インスタンスにはコンシューマーがある (**IPPROCS** >=1) が、AMQSCLM がそれらのリモート・インスタンスをモニターしていないため、コンシューマーはそれらのキュー・マネージャー上で非アクティブ (**CLWLPRTY**=0) である。

アクション: AMQSCLM がそれらのキュー・マネージャー上で実行されていることを確認してください。さらに (あるいは)、**-q** および **-f** パラメーターの値を確認して、キューがモニター対象のキューのリストにあることを確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブ (**CLWLPRTY**=1) だが、ローカル・キュー・マネージャーでは非アクティブ (**CLWLPRTY**=0) と思われる。 この状況は、更新された **CLWLPRTY** の値が、このキュー・マネージャーに伝搬されていないことが原因で発生しています。

アクション: リモート・キュー・マネージャーが、クラスター内の少なくとも 1 つのフル・リポジトリー・キュー・マネージャーに接続されていることを確認してください。 また、フル・リポジトリー・キュー・マネージャーが正常に機能していることを確認してください。 さらに、フル・リポジトリー・キュー・マ

ネージャーとモニター対象のキュー・マネージャー間のチャネルが実行されていることを確認してください。

可能性のある説明: メッセージがコミットされていないため、取得可能ではない。

アクション: 送信側のアプリケーションが正常に機能していることを確認してください。

可能性のある説明: AMQSCLM には、メッセージが入れられるローカル・キューへのアクセス権限がない。

アクション: AMQSCLM が、キューに対する十分なアクセス権限を持つユーザーとして実行しているかどうかを確認します。

可能性のある説明: キュー・マネージャーのコマンド・サーバーが実行されていない。

アクション: キュー・マネージャーのコマンド・サーバーを開始してください。

可能性のある説明: AMQSCLM にエラーが発生した。

アクション: レポート・ファイルにエラーが書き込まれていないか確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブである (CLWLPRTY=1) が、ローカル・キュー・マネージャーからのクラスター送信側チャネルが実行されていないため、それらのキュー・インスタンスにメッセージを転送できない。 この場合は、amqsclm レポート・ログ内に CLM0030W 警告が同時に記録されていることがよくあります。

アクション: アクティブなキュー・インスタンスがあるリモート・キュー・マネージャー (1 つ以上) への、ローカル・キュー・マネージャーからのクラスター送信側チャネルの状況を確認してください。

ALW 接続エンドポイント検索 *(CEPL)* のサンプル・プログラム

IBM MQ の接続エンドポイント検索サンプルは、IBM MQ ユーザーが Tivoli Directory Server などの LDAP リポジトリーから接続定義を取得する手段を提供する、シンプルながらも強力な出口モジュールを備えています。

CEPL を使用するには、Tivoli Directory Server v6.3 クライアントがインストールされている必要があります。

このサンプルを使用するには、サポート対象のプラットフォームで IBM MQ を管理するための実用的な知識が必要です。

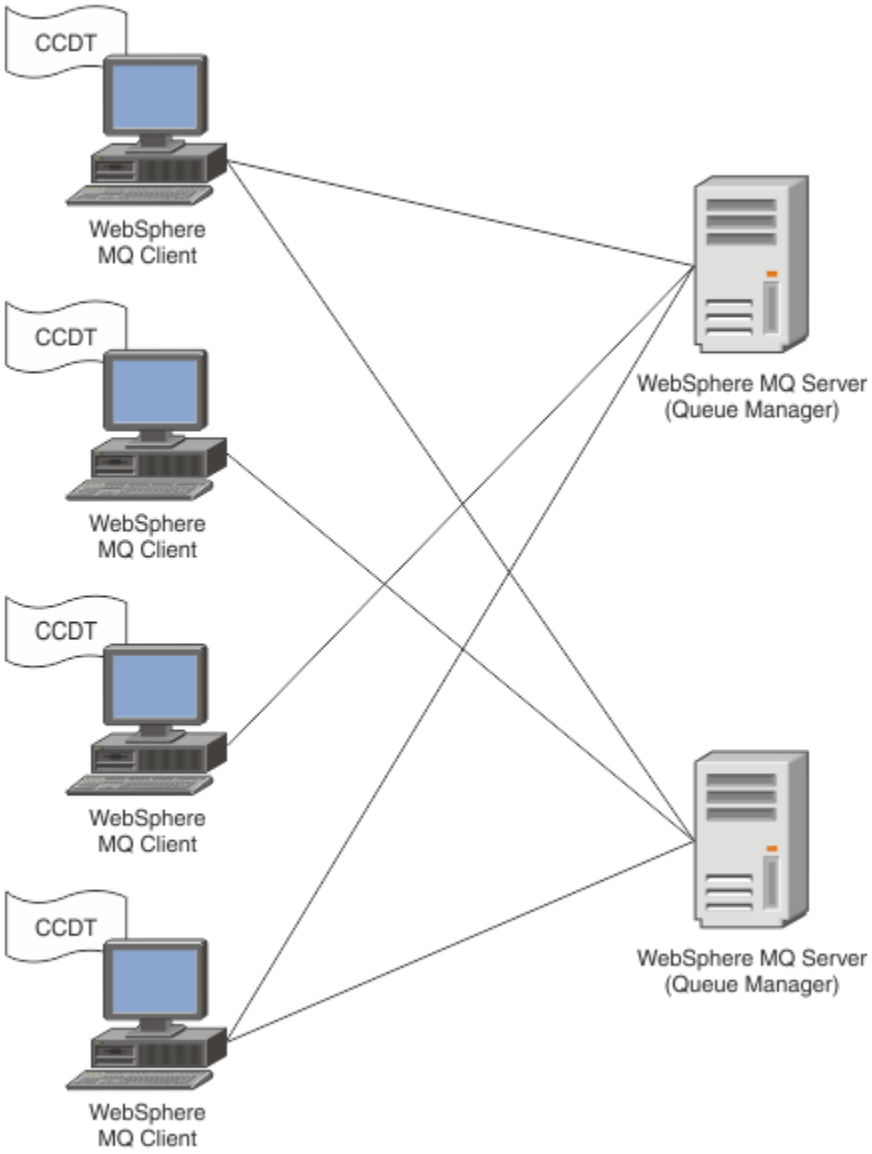
Windows Linux AIX 概要

保守および管理を補助するクライアント接続定義を保管するため、LDAP (Lightweight Directory Access Protocol) ディレクトリーなどのグローバル・リポジトリーを構成します。

IBM MQ クライアント・アプリケーションを使用して、クライアント接続定義テーブル (CCDT) を介し、キュー・マネージャーへの接続を確立します。

CCDT の作成には、標準的な IBM MQ MQSC 管理インターフェースを使用します。 クライアント接続定義を作成するには、定義内に含まれるデータがキュー・マネージャーに制限されていなくても、ユーザーはキュー・マネージャーに接続する必要があります。 生成される CCDT ファイルは、クライアント・マシン

およびアプリケーションに手動で配布する必要があります。

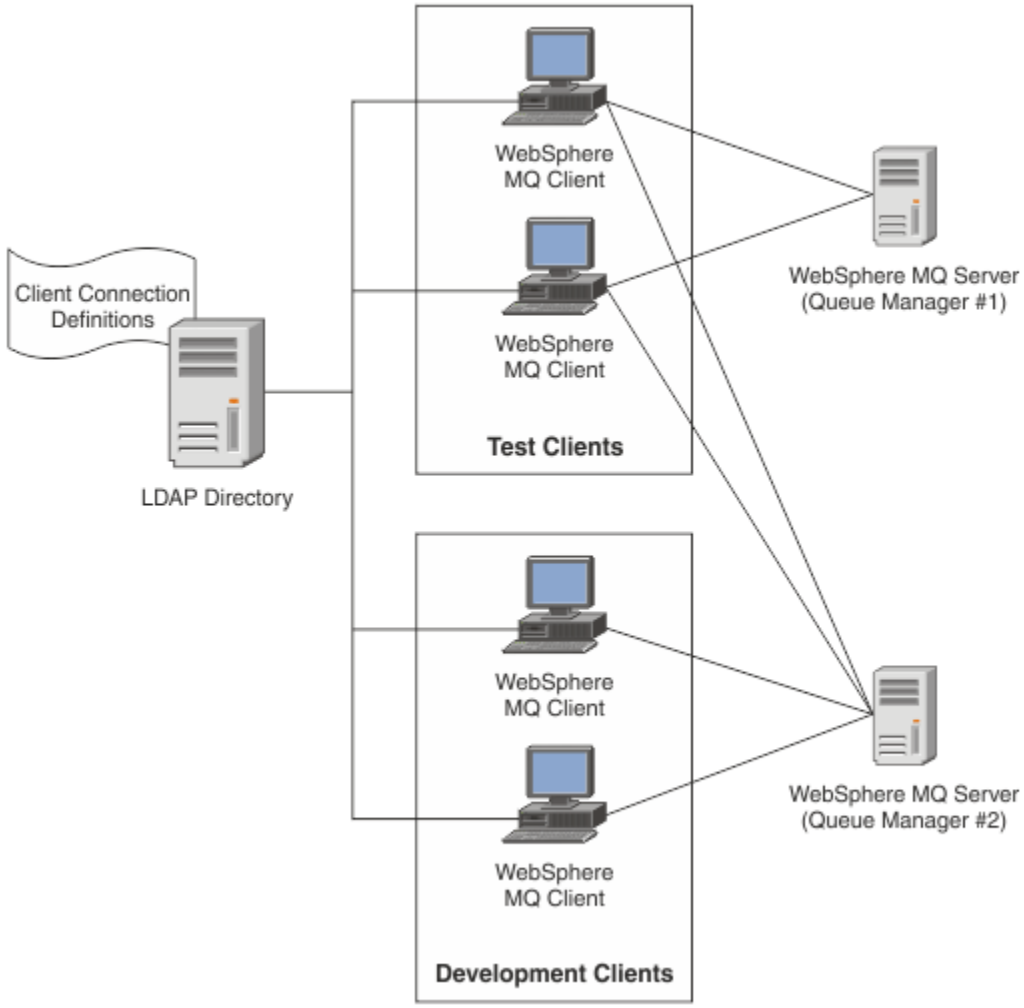


CCDT ファイルは、各 IBM MQ クライアントに手動で配布する必要があります。 何千ものクライアントがローカルにもグローバルにも存在できるような場合は、保守や管理がすぐに煩雑になります。 各クライアントに、そこで使用できる適切なクライアント定義が確実にあるようにするには、より柔軟なアプローチが必要です。

そのようなアプローチの 1 つとして、クライアント接続定義を LDAP (Lightweight Directory Access Protocol) ディレクトリーのようなグローバル・リポジトリーに保管することが挙げられます。 LDAP ディレクトリーには、追加のセキュリティー機能、索引付け機能、および検索機能もあるので、それを使用して各クライアントは自分に関連のある接続定義にのみアクセスできます。

LDAP ディレクトリーは、特定のユーザー・グループが特定の定義のみを使用できるように構成できます。 例えば、開発クライアントはキュー・マネージャー #2 にのみアクセス可能なのに対し、テスト・クライア

ントはキュー・マネージャー #1 と #2 の両方にアクセスできます。



出口モジュールは、LDAP リポジトリー (例えば IBM Tivoli Directory Server) を検索して、チャネル定義を取り出すことができます。 これらの接続定義を使用して、IBM MQ クライアント・アプリケーションはキュー・マネージャーへの接続を確立できます。

出口モジュールは、MQCONN/MQCONNX 呼び出しの際にチャネル定義を LDAP リポジトリーから取得できるようにする、接続前出口モジュールです。

出口モジュールおよびスキーマは、以下のお客様が実装できます。

- 既存の CCDT ファイル・ベース・テクノロジーを使用するスキルの基礎が既にあり、管理および配布のコストを軽減したいお客様。
- クライアント接続定義を配布するための独自の適切なテクノロジーを既に採用している、既存のお客様。
- 現在クライアント接続ソリューションを何も採用しておらず、IBM MQ が提供するフィーチャーを使用したい、新規または既存のお客様。
- 現行の LDAP ビジネス・アーキテクチャーに即してメッセージング・モデルを直接使用または調整したい、新規または既存のお客様。

ALW サポートされる環境

接続エンドポイント検索サンプルを実行する前に、サポート対象オペレーティング・システムと関連ソフトウェアがあることを確認してください。

IBM MQ 接続エンドポイント検索のサンプル・プログラムには、以下のソフトウェアが必要です。

- IBM WebSphere MQ 7.0 以降

- Tivoli Directory Server クライアント V6.3 以降

サポートされるオペレーティング・システム:

1. Windows (7/8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 および v5 (System p)
   - SUSE v9 および v10 (System p)
   - RHEL v4 および v5 (x86-6432 ビットおよび 64 ビット)
   - SUSE v9 および v10 (x86-6432 ビットおよび 64 ビット)

注 : 以下のプラットフォームでは、このサンプルは使用できません。

- z/OS
- IBM i

ALW インストールと構成
出口モジュールと接続エンドポイント・スキーマのインストールと構成。

出口モジュールのインストール

IBM MQ のインストール時に、出口モジュールは `tools/samples/c/preconnexit/bin` の下にインストールされます。 32 ビット・プラットフォームの場合は、出口モジュールを使用する前に `exit/`*`installation_name`*`/` をコピーする必要があります。 64 ビット・プラットフォームで出口モジュールを使用するには、その前に出口モジュールを exit64/*installation_name*/ にコピーしておく必要があります。

接続エンドポイント・スキーマのインストール

この出口は、接続エンドポイント・スキーマ `ibm-amq.schema` を使用します。 出口を使用するには、その前に任意の LDAP サーバーにスキーマ・ファイルをインポートしておく必要があります。 スキーマのインポート後は、属性の値を追加する必要があります。

以下は、接続エンドポイント・スキーマのインポート方法の例です。 この例は、IBM Tivoli Directory Server (ITDS) を使用することを前提としています。

- IBM Tivoli Directory Server が実行されていることを確認してから、`ibm-amq.schema` ファイルを ITDS サーバーにコピーまたは FTP します。
- ITDS サーバーで次のコマンドを入力して、スキーマを ITDS ストアにインストールします。*LDAP ID* と *LDAP* パスワードは、それぞれ LDAP サーバーのルート DN とパスワードです。

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- コマンド・ウィンドウで次のコマンドを入力するか、第三者製のツールでスキーマを参照して、スキーマを確認します。

```
ldapsearch objectclass=ibm-amqClientConnection
```

スキーマ・ファイルのインポートについて詳しくは、LDAP サーバーの資料を参照してください。

構成

PreConnect という名前の新しいセクションを、クライアント構成ファイル (例えば、`mqclient.ini`) に追加する必要があります。 PreConnect セクションには、以下のキーワードが含まれています。

モジュール
API 出口コードを含むモジュールの名前。 このフィールドにモジュールの絶対パスが指定されていると、そのまま使用されます。 それ以外の場合は、IBM MQ インストール環境の `exit` または `exit64` フォルダーが検索されます。

Function
`LdapPreConnect` 出口コードが入っているライブラリーへの機能エントリー・ポイントの名前。 関数定義は、企業の関数プロトタイプに従います。

重要 : 実際の出口エントリー・ポイントを指定するときは、関数ステートメント内の引用符を除去する必要があります。

データ
チャネル定義を含む LDAP リポジトリーの URI。

以下のスニペットは、`mqclient.ini` ファイルで必要とされる変更の例です。

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW 出口およびスキーマの概要

キュー・マネージャーへの接続を確立するために使用される構文およびパラメーター

IBM MQ 9.3 は、出口モジュール内のエントリー・ポイントとして以下の構文を定義しています。

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

MQCONN/X 呼び出しの実行中、IBM MQ C クライアントは関数構文の実装を含む出口モジュールをロードします。 その後、チャネル定義を検索するために出口関数が起動されます。 取り出されたチャネル定義が、キュー・マネージャーへの接続を確立するために使用されます。

パラメーター

pExitParms
タイプ: PMQNXP 入出力

PreConnection 出口パラメーター構造体。 この構造体は、出口の呼び出し側によって割り振られて維持されます。

```
struct tagMQNXP
{
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     ExitId;              /* Type of exit */
  MQLONG     ExitReason;          /* Reason for invoking exit */
  MQLONG     ExitResponse;        /* Response from exit */
  MQLONG     ExitResponse2;       /* Secondary response from exit */
  MQLONG     Feedback;            /* Feedback code (reserved) */
  MQLONG     ExitDataLength;      /* Exit data length */
  PMQCHAR    pExitDataPtr;        /* Exit data */
  MQPTR      pExitUserAreaPtr;    /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;      /* Array of pointers to MQCDs */
  MQLONG     MQCDArrayCount;      /* Number of entries found */
  MQLONG     MaxMQCDVersion;      /* Maximum MQCD version */
};
```

pQMgrName
タイプ: PMQCHAR 入出力

キュー・マネージャーの名前。 入力では、このパラメーターは、 **QMgrName** パラメーターを介して MQCONN API 呼び出しに提供されるフィルター・ストリングです。 このフィールドはブランクであるか、内容が明示的に指定されているか、特定のワイルドカード文字が含まれる場合があります。 このフィールドは出口によって変更されます。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。

ppConnectOpts
タイプ: ppConnectOpts 入出力

MQCONNX のアクションを制御するオプション。 これは、MQCONN API 呼び出しのアクションを制御する MQCNO 接続オプション構造へのポインターです。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。 MQCNO 構造がアプリケーションによって元々提供されなかった場合を含め、MQI クライアントは常に MQCNO 構造を出口に提供します。 アプリケーションが MQCNO 構造を提供した場合、 クライアントは複製を作成して出口に渡し、そこで構造が変更されます。 クライアントは MQCNO の所有権を保持します。 MQCNO を介して参照される MQCD は、配列を介して提供されたすべての接続定義より優先されます。 クライアントはキュー・マネージャーに接続するために MQCNO 構造体を使用し、その他の定義は無視されます。

pCompCode
タイプ: PMQLONG 入出力

完了コード。 出口完了コードを受け取る MQLONG へのポインター。 値は、次のいずれかでなければなりません。

- MQCC_OK - 正常終了。
- MQCC_WARNING - 警告 (部分的に完了)
- MQCC_FAILED - 呼び出しの失敗。

pReason
タイプ: PMQLONG 入出力

pCompCode を修飾する理由。 出口理由コードを受け取る MQLONG へのポインター。 完了コードが MQCC_OK の場合、唯一の有効値は次のとおりです。MQRC_NONE - (0, x'000') 報告する理由はありません。

完了コードが MQCC_FAILED または MQCC_WARNING の場合、出口関数は理由コード・フィールドを任意の有効な MQRC_* 値に設定できます。

ALW *MQ LDAP* コンテキスト情報
出口は、コンテキスト情報に以下のデータ構造体を使用します。

MQNLDAPCTX
MQNLDAPCTX 構造体は、以下の C プロトタイプを持ちます。

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX 接続エンドポイント検索出口の作成のためのサンプル・コード
このサンプル・コード・スニペットを使用して、AIX、または Linux、または Windows でソースをコンパイルできます。

ソースのコンパイル

任意の LDAP クライアント・ライブラリー (例えば、IBM Tivoli Directory Server V6.3 クライアント・ライブラリー) を使用して、ソースをコンパイルできます。 この資料は、Tivoli Directory Server V6.3 クライアント・ライブラリーを使用することを前提としています。

注 : 接続前出口ライブラリーは、以下の LDAP サーバーを使用してサポートされています。

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

以下のコード・スニペットでは、出口をコンパイルする方法について説明します。

Windows **Windows プラットフォームでの出口のコンパイル**

出口ソースをコンパイルする場合は、以下のスニペットを使用できます。

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 \
              /DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
$(CC) $(CCARGS) $*.c
```

注 : Microsoft Visual Studio 2003 コンパイラーでコンパイルされた IBM Tivoli Directory Server V6.3 クライアント・ライブラリーを使用している場合、 Microsoft Visual Studio 2012 以降のコンパイラーで IBM Tivoli Directory Server V6.3 クライアント・ライブラリーをコンパイルしているときに、警告を受け取ることがあります。

Linux AIX **AIX、または Linux での出口のコンパイル**

次のコード・スニペットは、Linux で出口ソースをコンパイルするためのものです。 AIX では、一部のコンパイラーのオプションが異なる場合があります。

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server には、静的リンク・ライブラリーと動的リンク・ライブラリーの両方が付属していますが、使用できるライブラリーのタイプは 1 つのみです。 このスクリプトは、静的ライブラリーを使用することを前提としています。

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl
```

```
amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *PreConnect* 出口モジュールの呼び出し

PreConnect 出口モジュールは、次の 3 種類の理由コードで呼び出せます。 LDAP サーバーへの接続を初期化および確立する場合は MQXR_INIT 理由コード。 LDAP サーバーからチャネル定義を取得する場合は MQXR_PRECONNECT 理由コード。 出口をクリーンアップする場合は MQXR_TERM 理由コード。

MQXR_INIT

LDAP サーバーとの接続を開始および確立する場合は、MQXR_INIT 理由コードを使用して出口を呼び出します。

MQXR_INIT 呼び出しの前に、MQNXP 構造体の pExitDataPtr フィールドには、`mqclient.ini` ファイル (つまり、LDAP) 内の PreConnect スタンザからのデータ属性が取り込まれます。

LDAP URL は、少なくともプロトコル、ホスト名、ポート番号、および検索のベース DN で構成されています。 出口は pExitDataPtr フィールドに含まれる LDAP URL を解析し、MQNLDAPCTX LDAP Lookup Context 構造体を割り振ってそれを適宜に入力します。 この構造体のアドレスは、pExitUserAreaPtr フィールドに保管されます。 LDAP URL を正しく解析できなかった場合は、MQCC_FAILED エラーが発生します。

この時点で、出口は **MQNLDAPCTX** パラメーターを使用して LDAP サーバーに接続し、バインドします。 その結果として発生する LDAP API ハンドルもこの構造体に保管されます。

MQXR_PRECONNECT

LDAP サーバーからチャネル定義を取得する場合は、MQXR_PRECONNECT 理由コードを使用して出口モジュールを呼び出します。

出口は、LDAP サーバーから所定のフィルターと一致するチャネル定義を検索します。 **QMgrNameparameter** に特定のキュー・マネージャー名が含まれている場合、指定されたキュー・マネージャー名と一致する **ibm-amqQueueManagerName** LDAP 属性値を持つすべてのチャネル定義が検索によって返されます。

QMgrName パラメーターが '*' または ' の場合 ' (ブランク) の場合、検索は、 **ibm-amqIsClientDefault Connection** エンドポイント属性が はいに設定されているすべてのチャネル定義を返します。

検索が成功すると、出口は MQCD 定義を 1 つまたは 1 配列作成し、呼び出し元に返します。

MQXR_TERM

出口をクリーンアップする場合は、この理由コードを使用して出口を呼び出します。 このクリーンアップの間、出口は LDAP サーバーとの接続を切断し、出口が割り振りおよび保守するすべてのメモリー (MQNLDAPCTX 構造体、ポインター配列、およびそれによって参照されるすべての MQCD を含む) を解放します。 その他のフィールドはすべてデフォルト値に設定されます。 **pQMgrName** および **ppConnectOpts** 出口パラメーターは、MQXR_TERM 理由コードを伴う出口の実行時には使用されず、NULL の場合があります。

関連資料

クライアント構成ファイルの PreConnect スタンザ

ALW *LDAP* スキーマ

クライアント接続データは、LDAP (Lightweight Directory Access Protocol) ディレクトリーというグローバル・リポジトリーに保管されます。 IBM MQ クライアントは、LDAP ディレクトリーを使用して接続定義を取得します。 LDAP ディレクトリー内の IBM MQ クライアント接続定義の構造体は LDAP スキーマといいます。 LDAP スキーマは、属性タイプの定義、オブジェクト・クラスの定義、およびサーバーの判断基準 (フィルターまたは属性値のアサーションがエントリーの属性と一致するかどうか、操作を許可、追加、および変更するかどうか) となるその他の情報の集合体です。

LDAP ディレクトリーへのデータの保管

クライアント接続定義は、接続ポイントというディレクトリー・ツリー内の特定のブランチにあります。 接続ポイントには、LDAP ディレクトリー内のその他のノードのように、それに関連付けられた識別名 (DN) が付けられています。 このノードは、ディレクトリーで実行するすべての照会の開始点として使用するこ

とができます。 クライアント接続定義のサブセットを返すよう LDAP ディレクトリーを照会する場合は、フィルターを使用します。 サブツリーへのアクセスは、ディレクトリー・ツリーの他の部分 (例えば、ユーザー、部門、またはグループ) で付与された権限に基づいて、制限することができます。

独自の属性およびクラスの定義

LDAP スキーマを変更して、クライアント・チャネル定義を保管します。 すべての LDAP データ定義には、オブジェクトと属性が必要です。 オブジェクトと属性は、オブジェクトまたは属性を一意的に識別するオブジェクト ID (OID) 番号によって識別されます。 LDAP スキーマ内のすべてのクラスは、最上位のオブジェクトから直接的または間接的に継承されます。 クライアント・チャネル定義オブジェクトには、最上位のオブジェクトの属性が含まれています。 すべての LDAP データ定義には、以下のオブジェクトと属性が必要です。

- オブジェクト定義は、LDAP 属性の集合体です。
- 属性は LDAP データ型です。

各属性の説明、および属性が通常の IBM MQ プロパティーにどのようにマップされるかについては、LDAP 属性を参照してください。

ALW *LDAP* 属性

定義された LDAP 属性は IBM MQ 固有の属性であり、クライアント接続プロパティーに直接マップされます。

IBM MQ クライアント・チャネル・ディレクトリーのストリング属性

以下の表では、文字ストリング属性とそれらがマップされる IBM MQ プロパティーをリストします。 これらの属性は、directoryString (サブセットとして IA5/ASCII を含む可変バイトのエンコード・システムである、UTF-8 エンコードの Unicode) 構文の値を保持することができます。 構文は、そのオブジェクト ID (OID) 番号によって指定されます。

表 *166. IBM MQ* クライアント・チャネル・ディレクトリーのストリング属性

LDAP 属性	**説明**	**IBM MQ プロパティー**
CN	チャネル名と定義されるキュー・マネージャー名で構成される共通名。	
ibm-amqChannelName	チャネル定義の名前。	CHANNEL
ibm-amqConnectionName	通信接続 ID。	CONNAME
ibm-amqDescription	チャネルの説明。	DESCR
ibm-amqLocalAddress	チャネルのローカル通信アドレス。	LOCLADDR
ibm-amqModeName	LU 6.2 モード名。	MODENAME
ibm-amqPassword	使用可能なパスワード。	PASSWORD
ibm-amqQueueManagerName	IBM MQ クライアント・アプリケーションが接続を要求できるキュー・マネージャーまたはキュー・マネージャー・グループの名前。	QMNAME
ibm-amqSecurityExitUserData	セキュリティー出口に渡されるユーザー・データ。	SCYDATA
ibm-amqSecurityExitName	チャネル・セキュリティー出口によって実行される出口プログラムの名前。	SCYEXIT
ibm-amqSslCipherSpec	TLS 接続用の単一の CipherSpec。	SSLCIPH
ibm-amqSslPeerName	IBM MQ チャネルの相手側にあるピア・キュー・マネージャーまたはクライアントからの証明書の識別名 (DN) を確認します。	SSLPEER
ibm-amqTransactionProgramName	トランザクション・プログラム名。	TPNAME

表 166. *IBM MQ* クライアント・チャネル・ディレクトリーのストリング属性 (続き)		
LDAP 属性	**説明**	**IBM MQ プロパティー**
ibm-amqUserID	リモート MCA との保護 SNA セッションの開始を試行するときに MCA が使用するユーザー ID。	ユーザー ID

IBM MQ クライアント接続の整数属性

事前定義値を持つ属性 (例えば、列挙型) は、標準の整数として保管されます。 これらの値は、関連付けられた定数名ではなく、整数値として LDAP ディレクトリーに保管されます。

表 167. *IBM MQ* クライアント・チャネル・ディレクトリーの整数属性		
LDAP 属性	**説明**	**IBM MQ プロパティー**
ibm-amqConnectionAffinity	同じキュー・マネージャー名を使用して複数回接続するクライアント・アプリケーションが、同じクライアント・チャネルを使用するかどうかを決定します。	AFFINITY
ibm-amqClientChannelWeight	どのクライアント接続チャネル定義を使用するかに影響を与える加重。	CLNTWGHT
ibm-amqHeartBeatInterval	伝送キューにメッセージがないときに送信側の MCA から渡されるハートビート・フローの間隔の概算時間。	HBINT
ibm-amqKeepAliveInterval	チャネルのタイムアウト値。	KAINT
ibm-amqMaximumMessageLength	チャネル上で送信可能なメッセージの最大長。	MAXMSGL
ibm-amqSharingConversations	各 TCP/IP チャネル・インスタンスを共用する会話の最大数。	SHARECNV
ibm-amqTransportType	使用するトランスポート・タイプ。	TRPTYPE

IBM MQ クライアント・チャネルのブール属性

このブール属性は、どの IBM MQ プロパティーにもマップされません。 この属性の構文はブール値を示します。

表 168. *IBM MQ* クライアント・チャネルのブール属性	
LDAP 属性	**説明**
ibm-amqIsClientDefault	このブール属性は、ibm-amqQueueManagerName 属性が定義されていないエントリーの検索の問題を解決するために定義します。

IBM MQ クライアント・チャネルのリスト属性

IBM MQ プロパティーは、単一値のコンマ区切りリスト属性として LDAP ディレクトリー内に保管されます。 これらの属性は、他のディレクトリー・ストリング属性と同じ方法で定義されます。 以下の表では、リスト属性とそれらがマップされる IBM MQ プロパティーについて説明します。

表 169. *IBM MQ* クライアント・チャネルのリスト属性		
LDAP 属性	**説明**	**IBM MQ プロパティー**
ibm-amqHeaderCompression	チャネルがサポートするヘッダー・データ圧縮技法のリスト。	COMPHDR

表 169. IBM MQ クライアント・チャネルのリスト属性 (続き)		
LDAP 属性	**説明**	**IBM MQ プロパティー**
ibm-amqMessageCompression	チャネルがサポートするメッセージ・データ圧縮技法のリスト。	COMPMSG
ibm-amqSendExitUserData	送信出口に渡されるユーザー・データ。	SENDDATA
ibm-amqSendExitUserName	チャネル送信出口によって実行される出口プログラムの名前。	SENDEXIT
ibm-amqReceiveExitUserData	受信出口に渡されるユーザー・データ。	RCVDATA
ibm-amqReceiveExitName	チャネル受信ユーザー出口によって実行されるユーザー出口プログラムの名前。	RCVEXIT

ALW 共通名

共通名 (CN) は、チャネル名と定義されるキュー・マネージャー名で構成されます。

これは既存の属性です。

CN の形式は次のとおりです。

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

以下に例を示します。

```
CN=TC1(QM_T1)
```

この属性の値は 1 つしか指定できません。

これはストリング属性で、値は大/小文字を区別しません。 サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルター (サブストリング (例えば、CN=jim* (CN は属性)) を使用し、1 つ以上のワイルドカードを含む) での属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqChannelName*

この属性は、チャネル定義の名前を指定します。

この属性は、大/小文字を区別しない、最大 20 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルター (サブストリングを使用し、1 つ以上のワイルドカードを含む) での属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqDescription*

この LDAP 属性は、チャネルの説明を提供します。

この属性は、大/小文字を区別しない、最大 64 バイトの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqConnectionName*

この LDAP 属性は、通信接続 ID です。 チャネルが使用する特定の通信リンクを指定します。

この属性は、大/小文字を区別しない、最大 264 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqLocalAddress*
この属性は、チャネルのローカル通信アドレスを指定します。

この属性は、大/小文字を区別しない、最大 48 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqModeName*
この属性は、LU 6.2 接続に使用します。 これは、通信セッションの割り振りが実行されるときに、接続のセッションの特性について追加の定義を提供します。

この属性は、大/小文字を区別しない、8 文字ちょうどの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqPassword*
この LDAP 属性は、リモート MCA との保護 LU 6.2 セッションの開始を試行するときに MCA が使用可能なパスワードを指定します。

この属性は、最大 12 桁の単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqQueueManagerName*
この属性は、IBM MQ クライアント・アプリケーションが接続を要求できるキュー・マネージャーまたはキュー・マネージャー・グループの名前を指定します。

この属性は、大/小文字を区別しない、最大 48 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

関連資料
1164 ページの『ibm-amqIsClientDefault』
このブール属性は、ibm-amqQueueManagerName 属性が定義されていないエントリーの検索の問題を解決します。

ALW *ibm-amqSecurityExitUserData*
この LDAP 属性は、セキュリティー出口に渡されるユーザー・データを指定します。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqSecurityExitName*
この LDAP 属性は、チャネル・セキュリティー出口によって実行される出口プログラムの名前を指定します。

有効なチャネル・セキュリティー出口がない場合は、これをブランクのままにします。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqSslCipherSpec*
この LDAP 属性は、TLS 接続用の単一の CipherSpec を指定します。

この属性は、大/小文字を区別しない、最大 32 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqSslPeerName*
この LDAP 属性は、IBM MQ チャネルの相手側にあるピア・キュー・マネージャーまたはクライアントからの証明書の識別名 (DN) を確認するために使用します。

この LDAP 属性は、大/小文字を区別しない、最大 1024 バイトの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqTransactionProgramName*
この LDAP 属性は、トランザクション・プログラム名を指定します。 これは、LU 6.2 接続に使用されます。

この属性は、大/小文字を区別しない、最大 64 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqUserID*
この LDAP 属性は、リモート MCA との保護 SNA セッションの開始を試行するときに MCA が使用するユーザー ID を指定します。

この属性は、大/小文字を区別しない、12 文字ちょうどの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ALW *ibm-amqConnectionAffinity*
この LDAP 属性は、同じキュー・マネージャー名を使用して複数回接続するクライアント・アプリケーションが、同じクライアント・チャネルを使用するかどうかを指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqClientChannelWeight*
この LDAP 属性は、どのクライアント接続チャネル定義を使用するかに影響を与える加重を指定します。

クライアント・チャネル加重属性は、複数の適切な定義が使用可能な場合に、クライアント・チャネル定義の選択にバイアスをかけるために使用します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqHeartBeatInterval*
この LDAP 属性によって、伝送キューにメッセージがなくなったときに送信 MCA からハートビート・フローが渡される間の時間の近似値を指定することができます。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。 デフォルト値は 1 です。 デフォルトは、現在の MQSERVER 環境変数操作で設定されます。

ALW *ibm-amqKeepAliveInterval*
この LDAP 属性は、チャネルのタイムアウト値を指定するために使用します。

この属性の値は、チャネルのキープアライブ・タイミングを指定する通信スタックに渡されます。 これは、チャネルごとに別々のキープアライブ値を指定するために使用できます。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqMaximumMessageLength*
この LDAP 属性は、チャネル上で送信可能なメッセージの最大長を指定します。

現在の MQSERVER 環境変数の操作により、この属性のデフォルト値は 104857600 です。 この属性は単一の整数値を持ち、既存の属性ではありません。

ALW *ibm-amqSharingConversations*
この LDAP 属性は、各 TCP/IP チャネル・インスタンスを共用する会話の最大数を指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqTransportType*
この LDAP 属性は、使用するトランスポート・タイプを指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ALW *ibm-amqIsClientDefault*
このブール属性は、ibm-amqQueueManagerName 属性が定義されていないエントリーの検索の問題を解決します。

通常、接続前出口モジュールは検索条件に ibm-amqQueueManagerName 属性の値を指定して LDAP サーバーを検索します。 このようにして照会すると、ibm-amqQueueManagerName 属性値が MQCONN/X 呼び出しに指定されたキュー・マネージャーの名前と一致するすべての項目が返されます。 ただし、クライアント・チャネル定義テーブル (CCDT) を使用する場合は、MQCONN/X 呼び出しでキュー・マネージャー名をブランクに設定したり、名前の前にアスタリスク (*) を付けたりすることができます。キュー・マネージャーの名前がブランクの場合、クライアントはデフォルトのキュー・マネージャーに接続します。 キュー・マネージャーの名前の先頭にアスタリスク (*) が付いているときは、クライアントは任意のキュー・マネージャーに接続します。

同様に、項目内の ibm-amqQueueManagerName 属性を未定義のままにしておくことができます。 この場合、このエンドポイント情報を使用するクライアントがどのキュー・マネージャーにも接続できることが予期されています。 例えば、ある項目に次の行が含まれているとします。

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

この例では、クライアントは myhost 上で実行中の指定されたキュー・マネージャーに接続することを試みます。

しかし、LDAP サーバーでは、定義されていない属性値の検索は実行されません。 例えば、ある項目に ibm-amqQueueManagerName 以外の接続情報が含まれている場合、検索結果にこの項目は含まれません。 この問題を克服するために、ibm-amqIsClientDefault を設定することができます。 これはブール属性で、定義されていない場合は FALSE の値をとるものと見なされます。

ibm-amqQueueManagerName が定義されていなくても検索結果に含める項目については、ibm-amqIsClientDefault を TRUE に設定します。 MQCONN/X への呼び出しでキュー・マネージャー名としてブランクまたはアスタリスク (*) が指定されている場合、接続前出口は LDAP サーバーを検索して、ibm-amqIsClientDefault 属性値が TRUE に設定されているすべての項目を探します。

注 : ibm-amqIsClientDefault が TRUE に設定されている場合は、ibm-amqQueueManagerName 属性を設定または定義しないでください。

関連資料

1162 ページの『ibm-amqQueueManagerName』
この属性は、IBM MQ クライアント・アプリケーションが接続を要求できるキュー・マネージャーまたはキュー・マネージャー・グループの名前を指定します。

ALW *ibm-amqHeaderCompression*
この LDAP 属性は、チャネルでサポートされるヘッダー・データ圧縮技法のリストです。

この属性の最大サイズは 48 文字です。 これは、既存の属性ではありません。

この属性の値は 1 つしか指定できません。

このリスト属性は、コンマ区切り形式のディレクトリー・ストリングとして指定します。 例えば、**ibm-amqHeaderCompression** に指定される値は `0` であり、これは NONE にマップされます。 最大許容制限を超える値は、クライアントに無視されます。 例えば、ibm-amqHeaderCompression はリストに最大 2 つの整数を含みます。

ALW *ibm-amqMessageCompression*
この LDAP 属性は、チャネルでサポートされるメッセージ・データ圧縮技法のリストです。

この属性の最大サイズは 48 文字です。 これは、既存の属性ではありません。

この属性は複数の値をサポートしません。

このリスト属性は、コンマ区切り形式のディレクトリー・ストリングとして指定します。 例えば、この属性に指定される値は 1、2、4 で、これは基礎となる圧縮シーケンス RLE、ZLIBFAST、および ZLIBHIGH にマップされます。

最大許容制限を超える値は、クライアントに無視されます。 例えば、ibm-amqMessageCompression はリストに最大 16 つの整数を含みます。

ALW *ibm-amqSendExitUserData*
この LDAP 属性は、送信出口に渡されるユーザー・データを指定します。

この LDAP 属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqSendExitName と **ibm-amqSendExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ALW *ibm-amqSendExitName*
この LDAP 属性は、チャネル送信出口によって実行される出口プログラムの名前を指定します。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqSendExitName と **ibm-amqSendExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ALW *ibm-amqReceiveExitUserData*
この LDAP 属性は、受信出口に渡されるユーザー・データを指定します。

一連の受信出口を実行できます。 一連の出口のユーザー・データのストリングは、コンマ、スペース、またはその両方で区切られます。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : **ibm-amqReceiveExitName** と **ibm-amqReceiveExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ALW *ibm-amqReceiveExitName*
この LDAP 属性は、チャネル受信ユーザー出口によって実行されるユーザー出口プログラムの名前を指定します。

この属性は、連続して実行されるプログラムの名前のリストです。 有効なチャネル受信ユーザー出口がない場合には、ブランクのままにしておいてください。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : **ibm-amqReceiveExitName** と **ibm-amqReceiveExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

z/OS z/OS 用サンプル・プログラムの使用

IBM MQ for z/OS と共に提供されるプロシージャー型サンプル・アプリケーションは、Message Queue Interface (MQI) の一般的な使用法を示しています。

このタスクについて

IBM MQ for z/OS では、988 ページの『データ変換出口の作成』で説明されているように、サンプル・データ変換出口も提供しています。

すべてのサンプル・アプリケーションはソース形式で提供されます。いくつかのサンプル・アプリケーションは実行可能形式ででも提供されます。 このソース・モジュールには、プログラム論理を記述する疑似コードが含まれています。

注 : サンプル・アプリケーションのなかには、基本的なパネル起動型のインターフェースをもつものがありますが、これらはアプリケーションの外観の設計方法を示すことを目的としたものではありません。 非プログラム式端末用のパネル起動型のインターフェースの設計方法については、*SAA Common User Access: Basic Interface Design Guide* (SC26-4583) およびその付録 (GG22-9508) を参照してください。 これらは、アプリケーション内およびアプリケーション間の両面で、整合性のあるアプリケーションを設計する上の手引きとなります。

手順

- 以下のリンクを使用して、サンプル・プログラムについての詳細を確認してください。
  - 1167 ページの『z/OS のサンプル・アプリケーションで示されている機能』
  - 1173 ページの『z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行』
  - 1176 ページの『z/OS における TSO 環境用サンプル・アプリケーションの作成』
  - 1178 ページの『z/OS での CICS 環境用のサンプル・アプリケーションの準備』
  - 1182 ページの『z/OS での IMS 環境用のサンプル・アプリケーションの準備』
  - 1183 ページの『z/OS における書き込みサンプル』
  - 1185 ページの『z/OS における読み取りサンプル』
  - 1188 ページの『z/OS でのブラウズ・サンプル』
  - 1190 ページの『z/OS におけるメッセージ印刷サンプル』
  - 1193 ページの『z/OS におけるキュー属性サンプル』
  - 1194 ページの『z/OS におけるメール管理プログラム・サンプル』

- 1201 ページの『z/OS における信用小切手サンプル』
- 1213 ページの『z/OS におけるメッセージ・ハンドラー・サンプル』
- 1217 ページの『z/OS における非同期書き込みサンプル』
- 1218 ページの『z/OS におけるバッチ非同期コンシューム・サンプル』
- 1219 ページの『z/OS における CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル』
- 1222 ページの『z/OS におけるパブリッシュ/サブスクライブのサンプル』
- 1224 ページの『z/OS におけるメッセージ・プロパティーの設定および照会のサンプル』

関連タスク

1063 ページの『Multiplatforms でのサンプル・プログラムの使用』

これらのプロシージャー型サンプル・プログラムは製品に同梱されています。 サンプルは C および COBOL で作成されており、メッセージ・キュー・インターフェース (MQI) の一般的な使用法を示します。

z/OS *z/OS* のサンプル・アプリケーションで示されている機能

このセクションでは、それぞれのサンプル・アプリケーションで示される MQI 機能について要約し、各サンプルを書き込むプログラミング言語、および各サンプルを実行する環境を示します。

z/OS *z/OS* における書き込みサンプル

書き込みサンプルは、MQPUT 呼び出しを使用してキューにメッセージを書き込む方法を示しています。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

このプログラムは COBOL および C で提供され、バッチ環境および CICS 環境で稼働します。 バッチ・アプリケーションについては 1174 ページの表 172 を、 CICS アプリケーションについては 1179 ページの表 179 を参照してください。

z/OS *z/OS* における読み取りサンプル

Get サンプルは、MQGET 呼び出しを使用してキューからメッセージを取得する方法を示しています。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

このプログラムは COBOL および C で提供され、バッチ環境および CICS 環境で稼働します。 バッチ・アプリケーションについては 1174 ページの表 172 を、 CICS アプリケーションについては 1179 ページの表 179 を参照してください。

z/OS *z/OS* におけるブラウズ・サンプル

ブラウズ・サンプルは、「ブラウズ」オプションを使用してメッセージを検索し、そのメッセージを印刷し、キューにあるメッセージ間を移動する方法を示します。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN

- MQOPEN
- メッセージをブラウズするための MQGET
- MQCLOSE
- MQDISC

このプログラムは、COBOL、アセンブラー、PL/I、および C の各言語で提供されています。 このアプリケーションは、バッチ環境で実行されます。 バッチ・アプリケーションについては、1175 ページの表 173 を参照してください。

z/OS *z/OS* におけるメッセージ印刷サンプル

メッセージ印刷サンプルは、そのメッセージ記述子のすべてのフィールドと共に、キューからメッセージを除去する方法およびメッセージのデータを印刷する方法を示します。 オプションで、各メッセージに関連付けられたすべてのメッセージ・プロパティーを表示することができます。

ソース・モジュール内の 2 行からコメント文字を除去することによって、キュー上のメッセージを除去するのではなく、ブラウズするようにプログラムを変更できます。 このプログラムは、キューにメッセージを入れるアプリケーションでの問題を診断するときに便利です。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- キューからメッセージを除去するための MQGET (ブラウズのためのオプション付き)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

このプログラムは、C 言語で提供されています。 このアプリケーションは、バッチ環境で実行されます。 バッチ・アプリケーションについては、1175 ページの表 174 を参照してください。

z/OS *z/OS* におけるキュー属性サンプル

キュー属性サンプルでは、IBM MQ for z/OS オブジェクト属性の値を照会および設定する方法を説明します。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

このプログラムは、COBOL、アセンブラー、および C の各言語で提供されています。 このアプリケーションは、CICS 環境で実行されます。 CICS アプリケーションについては、 1179 ページの表 180 を参照してください。

z/OS *z/OS* におけるメール管理プログラム・サンプル

メール管理プログラム・サンプルを使用する際の考慮事項を示します。

メール管理プログラム・サンプルでは、以下の技法について示します。

- 別名キューの使用
- 一時的な動的キューを作成するためのモデル・キューの使用
- 応答先キューの使用
- CICS 環境およびバッチ環境での同期点の使用

- システム・コマンド入力キューへのコマンドの送信
- 戻りコードのテスト
- リモート・キューのローカル定義を使用する方法や、リモート・キュー・マネージャーの名前付きキューにメッセージを直接入れる方法によるリモート・キュー・マネージャーへのメッセージの送信

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

このアプリケーションには次の 3 つのバージョンがあります。

- COBOL で作成された CICS アプリケーション
- COBOL で作成された TSO アプリケーション
- C で作成された TSO アプリケーション

TSO アプリケーションでは、IBM MQ for z/OS バッチ・アダプターが使用され、いくつかの ISPF パネルが組み込まれています。

TSO アプリケーションについては 1177 ページの表 177 を、CICS アプリケーションについては 1180 ページの表 181 を参照してください。

z/OS *z/OS* における信用小切手サンプル

ここでは、信用小切手サンプルを使用する際の考慮事項について説明します。

信用小切手サンプルは、以下の技法を示すプログラムです。

- 複数の環境で実行されるアプリケーションの開発
- 一時的な動的キューを作成するためのモデル・キューの使用
- 相関 ID の使用
- コンテキスト情報の設定と引き渡し
- メッセージ優先順位と持続的な有効性の使用
- トリガー操作によるプログラムの起動
- 応答先キューの使用
- 別名キューの使用
- 送達不能キューの使用
- 名前リストの使用
- 戻りコードのテスト

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQOPEN
- MQPUT
- MQPUT1
- 待機および信号オプションの使用によるメッセージのブラウズと読み取り、および特定のメッセージ読み取りのための MQGET
- MQINQ
- MQSET

- MQCLOSE

このサンプルは、独立型の CICS アプリケーションとして実行できます。 しかし、CICS および IMS の両環境で提供される機能を使用するメッセージ・キューイング・アプリケーションの設計方法を示すために、1 つのモジュールが IMS バッチ・メッセージ処理プログラムとしても提供されます。

CICS プログラムは C および COBOL で提供されます。 単一の IMS プログラムが C で提供されます。

CICS アプリケーションの場合は 1180 ページの表 182 を、IMS アプリケーションの場合は 1182 ページの表 184 を参照してください。

z/OS *z/OS* におけるメッセージ・ハンドラー・サンプル

メッセージ・ハンドラー・サンプルでは、キュー上のメッセージのブラウズ、転送、および削除を行うことができます。

このアプリケーションでは、次の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

このプログラムは、C および COBOL の各プログラム言語で提供されています。 このアプリケーションは TSO 環境で実行されます。 TSO アプリケーションについては、1178 ページの表 178 を参照してください。

z/OS *z/OS* における分散キューイング出口サンプル

分散キューイング出口サンプルのソース・プログラムを表の形式で示します。

次の表に、分散キューイング出口サンプルのソース・プログラムの名前を示します。

表 *170.* 分散キューイング出口サンプルのソース

メンバー名	対象言語	説明	ライブラリーで提供されるもの
CSQ4BAX0	アセンブラー	ソース・プログラム	SCSQASMS
CSQ4BCX1	C	ソース・プログラム	SCSQC37S
CSQ4BCX2	C	ソース・プログラム	SCSQC37S
CSQ4BCX4	C	ソース・プログラム	SCSQC37S

注 : ソース・プログラムは CSQXSTUB とリンク・エディットされています。

z/OS *z/OS* におけるデータ変換出口サンプル

データ変換出口ルーチンにはスケルトンが提供され、IBM MQ には MQXCNVC 呼び出しを説明するサンプルが提供されています。

次の表に、データ変換出口サンプルのソース・プログラムの名前を示しています。

表 *171.* データ変換出口サンプルのソース *(*アセンブラー言語のみ*)*

メンバー名	説明	ライブラリーで提供されるもの
CSQ4BAX8	ソース・プログラム	SCSQASMS

表 171. データ変換出口サンプルのソース (アセンブラー言語のみ) (続き)		
メンバー名	**説明**	**ライブラリーで提供されるもの**
CSQ4BAX9	ソース・プログラム	SCSQASMS
CSQ4CAX9	ソース・プログラム	SCSQASMS

注 : ソース・プログラムは CSQASTUB とリンク・エディットされています。

詳しくは、 988 ページの『データ変換出口の作成』 を参照してください。

z/OS *z/OS におけるパブリッシュ/サブスクライブのサンプル*

パブリッシュ/サブスクライブのサンプル・プログラムでは、IBM MQ のパブリッシュ機能およびサブスクライブ機能の使用法を示します。

4 つの C プログラミング言語および 2 つの COBOL プログラミング言語のサンプル・プログラムがあり、これらは IBM MQ パブリッシュ/サブスクライブ・インターフェースのプログラム作成方法を示します。

これらのアプリケーションでは、以下の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

パブリッシュ/サブスクライブのサンプル・プログラムは、C および COBOL プログラミング言語で提供されます。 サンプル・アプリケーションはバッチ環境で実行されます。 バッチ・アプリケーションについては、パブリッシュ/サブスクライブのサンプルを参照してください。

z/OS クライアント接続を受け入れるようにキュー・マネージャーを構成する *(z/OS)*

サンプル・アプリケーションを実行するには、その前にキュー・マネージャーを作成する必要があります。その後、クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるように、キュー・マネージャーを構成できます。

始める前に

キュー・マネージャーが既に存在しており、開始していることを確認します。 MQSC コマンドを実行することにより、チャネル認証レコードが既に使用可能になっているかどうかを判別します。

```
DISPLAY QMGR CHLAUTH
```

重要 : このタスクは、チャネル認証レコードが使用可能になっていることを前提としています。 このキュー・マネージャーが他のユーザーやアプリケーションによって使用されている場合、この設定を変更すると、他のすべてのユーザーとアプリケーションが影響を受けます。 キュー・マネージャーがチャネル認証レコードを利用しない場合には、ステップ 4 を代替認証方式 (例えば、セキュリティー出口など) に置き換えて、MCAUSER をステップ 1172 ページの『1』 で取得する *non-privileged-user-id* に設定することができます。

アプリケーションが使用すると予期されるチャネル名を把握し、アプリケーションがそのチャネルを使用できるようにする必要があります。 また、アプリケーションが使用すると予期されるオブジェクト (例え

ば、キューやトピック) も把握し、アプリケーションがこれらのオブジェクトを使用できるようにする必要もあります。

このタスクについて

このタスクにより、キュー・マネージャーに接続するクライアント・アプリケーションで使用する、非特権ユーザー ID が作成されます。 クライアント・アプリケーションがこのユーザー ID を使用して必要とするチャネルとキューを使用できるようにするためにのみ、アクセス権限が付与されます。

手順

1. キュー・マネージャーが実行されているシステムでユーザー ID を取得します。

   このタスクの場合、このユーザー ID は特権管理ユーザーにすることはできません。 このユーザー ID の権限は、クライアント接続をキュー・マネージャーで実行するためのものです。
2. リスナー・プログラムを開始します。

   a) チャネル・イニシエーターが開始されていることを確認します。 そうでない場合は、 **START CHINIT** コマンドを発行して開始します。

   b) 次のコマンドを発行して、リスナー・プログラムを開始します。

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   nnnn は、選択したポート番号です。
3. アプリケーションが SYSTEM.DEF.SVRCONN を使用する場合、このチャネルは既に定義済みです。 アプリケーションが別のチャネルを使用する場合は、以下のようにして、それを MQSC コマンドを発行して作成します。

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name は、チャネルの名前です。
4. ご使用のクライアント・システムの IP アドレスのみにチャネルの使用を許可するチャネル認証規則を、以下のように MQSC コマンドを発行して作成します。

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   説明:

   channel-name は、チャネルの名前です。

   client-machine-IP-address は、クライアント・システムの IP アドレスです。 サンプル・クライアント・アプリケーションがキュー・マネージャーと同じマシン上で実行されているときに、そのアプリケーションが「localhost」を使用して接続しようとしているのであれば、IP アドレス「127.0.0.1」を使用します。 複数のさまざまなクライアント・マシンが接続することになっている場合、単一の IP アドレスではなく、パターンや範囲を使用することができます。 詳細については、汎用 IP アドレスを参照してください。

   non-privileged-user-id は、ステップ 1172 ページの『1』 で取得したユーザー ID です。
5. アプリケーションが SYSTEM.DEFAULT.LOCAL.QUEUE を使用する場合、このキューは既に定義済みです。 アプリケーションが別のキューを使用する場合は、以下のようにして、それを MQSC コマンドを発行して作成します。

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   queue-name は、キューの名前です。
6. キュー・マネージャーへの接続と照会のアクセス権限を以下のように付与します。

a) チャネル・イニシエーターが開始されていることを確認します。 まだ開始されていない場合は、START CHINIT コマンドを発行してチャネル・イニシエーターを開始します。

b) 例えば次のコマンドを発行して、TCP リスナーを開始します。

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

nnnn は、選択したポート番号です。

7. アプリケーションが Point-to-Point アプリケーションである場合 (つまりキューを使用する場合)、キューを使用したメッセージの照会、書き込み、および読み取りを、ユーザー ID を使用して行えるようにするために、以下のように MQSC コマンドを発行してアクセス権限を付与します。

   次の RACF コマンドを発行します。

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

   説明:

   qmgr-name は、キュー・マネージャーの名前です。
   queue-name は、キューの名前です。
   non-privileged-user-id は、ステップ 1172 ページの『1』 で取得したユーザー ID です。

8. アプリケーションがパブリッシュ/サブスクライブ・アプリケーションである場合 (つまりトピックを使用する場合) 、使用するユーザー ID による、トピックを使用したパブリッシュ/サブスクライブを行えるようにするために、以下のように RACF コマンドを発行してアクセス権限を付与します。

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

   説明:

   qmgr-name は、キュー・マネージャーの名前です。
   non-privileged-user-id は、ステップ 1172 ページの『1』 で取得したユーザー ID です。
   これにより、トピック・ツリー内のあらゆるトピックへのアクセス権限が *non-privileged-user-id* に付与されます。または、**DEFINE TOPIC** を使用してトピック・オブジェクトを定義し、そのトピック・オブジェクトにより参照されるトピック・ツリーの一部のみへのアクセス権限を付与することもできます。 詳しくは、トピックへのユーザー・アクセスの制御を参照してください。

次のタスク

これで、クライアント・アプリケーションはキュー・マネージャーに接続し、キューを使用してメッセージの書き込みや読み取りができるようになりました。

関連概念

z/OS IBM MQ 上で z/OS オブジェクトを処理する権限

関連資料

SET CHLAUTH
DEFINE CHANNEL
DEFINE QLOCAL
SET AUTHREC

z/OS z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行

バッチ環境で実行されるサンプル・アプリケーションを準備するには、バッチ IBM MQ for z/OS アプリケーションを構築するときと同じ手順を実行します。

これらのステップは、1029 ページの『z/OS バッチ・アプリケーションの構築』に記述してあります。

あるいは、実行可能形式のサンプルが提供される場合には、それを thlqual.SCSQLOAD ロード・ライブラリーから実行することもできます。

注 : アセンブラー言語バージョンのブラウズ・サンプルでは、データ制御ブロック (DCB) を使用します。したがって、RMODE(24) を使用してリンク・エディットする必要があります。

使用するライブラリー・メンバーは、1174 ページの表 172、1175 ページの表 173、1175 ページの表 174、および 1175 ページの表 175 に示されています。

使用しようとするサンプルに提供されている実行 JCL を編集する必要があります (1174 ページの表 172、1175 ページの表 173、1175 ページの表 174、および 1175 ページの表 175 を参照)。

提供される JCL 内の PARM ステートメントには、修正の必要があるパラメーターが多数含まれています。C サンプル・プログラムを実行する場合は、パラメーターをスペースで区切ります。アセンブラー、COBOL、および PL/I サンプル・プログラムを実行する場合は、パラメーターをコンマで区切ります。 例えば、キュー・マネージャーの名前が CSQ1 で、LOCALQ1 という名前のキューでアプリケーションを実行したい場合、COBOL、PL/I、およびアセンブラー言語の JCL において、PARM ステートメントを次のように指定してください。

```
PARM=(CSQ1,LOCALQ1)
```

C 言語の JCL では、PARM ステートメントを次のように指定してください。

```
PARM=('CSQ1 LOCALQ1')
```

この時点で、ジョブの実行依頼ができる状態になりました。

z/OS *z/OS* におけるサンプル・バッチ・アプリケーションの名前

サンプル・バッチ・アプリケーション用に提供されるプログラムの要約を示します。

バッチ・アプリケーション・プログラムは次の表に要約されています。

- 1174 ページの表 172 書き込みおよび読み取りのサンプル
- 1175 ページの表 173 ブラウズ・サンプル
- 1175 ページの表 174 メッセージ印刷サンプル
- 1175 ページの表 175 パブリッシュ/サブスクライブのサンプル
- 1176 ページの表 176 その他のサンプル

表 *172.* バッチ書き込みおよび読み取りのサンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4BCJ1	C	ソース・プログラムの読み取り	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	ソース・プログラムの書き込み	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	CSQ4BCJ1 および CSQBCK1 のサンプル実行 JCL	SCSQPROC	なし
CSQ4BVJ1	COBOL	ソース・プログラムの読み取り	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	ソース・プログラムの書き込み	SCSQCOBS	SCSQLOAD

表 *172.* バッチ書き込みおよび読み取りのサンプル *(*続き*)*

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4BVJR	COBOL	CSQBVJ1 および CSQBVK1 のサンプル実行 JCL	SCSQPROC	なし

表 *173.* バッチ・ブラウズ・サンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4BVA1	COBOL	ソース・プログラム	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	CSQ4BVA1 のサンプル実行 JCL	SCSQPROC	なし
CSQ4BAA1	アセンブラー	ソース・プログラム	SCSQASMS	SCSQLOAD
CSQ4BAAR	アセンブラー	CSQ4BAA1 のサンプル実行 JCL	SCSQPROC	なし
CSQ4BCA1	C	ソース・プログラム	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	CSQ4BCA1 のサンプル実行 JCL	SCSQPROC	なし
CSQ4BPA1	PL/I	ソース・プログラム	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	CSQ4BPA1 のサンプル実行 JCL	SCSQPROC	なし

表 *174.* バッチ・メッセージ印刷のサンプル *(C* 言語のみ*)*

メンバー名	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4BCG1	ソース・プログラム	SCSQC37S	SCSQLOAD
CSQ4BCGR	CSQ4BCG1 のサンプル実行 JCL	SCSQPROC	なし
CSQ4BCL1	ソース・プログラムのブラウズ	SCSQC37S	SCSQLOAD
CSQ4BCLR	CSQ4BCL1 のサンプル実行 JCL	SCSQPROC	なし

表 *175.* パブリッシュ/サブスクライブのサンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	SCSQPROC の JCL	ライブラリーで提供される実行可能ファイル
CSQ4BCP1	C	トピックへのパブリッシュのソース・プログラム	SCSQC37S	CSQ4BCPP	SCSQLOAD

表 175. パブリッシュ/サブスクライブのサンプル (続き)					
メンバー名	**対象言語**	**説明**	**ライブラリーで提供されるソース・ファイル**	**SCSQPROC の JCL**	**ライブラリーで提供される実行可能ファイル**
CSQ4BCP2	C	トピックへのサブスクライブおよびメッセージの読み取りのソース・プログラム	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	ユーザーが指定した宛先を使用したトピックへのサブスクライブおよびメッセージの読み取りのソース・プログラム	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	拡張オプションを使用したトピックへのサブスクライブおよびメッセージの読み取りのソース・プログラム	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	トピックへのパブリッシュのソース・プログラム	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	トピックへのサブスクライブおよびメッセージの読み取りのソース・プログラム	SCSQCOBS	CSQ4BVPS	SCSQLOAD

表 176. その他のサンプル					
メンバー名	**対象言語**	**説明**	**ライブラリーで提供されるソース・ファイル**	**SCSQPROC の JCL**	**ライブラリーで提供される実行可能ファイル**
CSQ4BCS1	C	非同期コンシュームのソース・プログラム	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	非同期書き込みおよび状況チェックのソース・プログラム	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	メッセージ・プロパティーの照会のソース・プログラム	SCSQC37S	CSQ4BCMP	SCSQLOAD
CSQ4BCM2	C	メッセージ・プロパティーの設定のソース・プログラム	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *z/OS における TSO 環境用サンプル・アプリケーションの作成*

TSO 環境で実行されるサンプル・アプリケーションを準備するには、バッチ IBM MQ for z/OS アプリケーションを構築するときと同じ手順を実行します。

これらのステップは、1029 ページの『z/OS バッチ・アプリケーションの構築』に記述してあります。 使用するライブラリー・メンバーは、1177 ページの表 177 に示してあります。

あるいは、実行可能形式のサンプルが提供される場合には、それを thlqual.SCSQLOAD ロード・ライブラリーから実行することもできます。

メール管理プログラムのサンプル・アプリケーションについて、それが使用するキューがシステム上で使用できるか確認します。 キューはメンバー **thlqual**.SCSQPROC(CSQ4CVD) で定義されます。 これらのキューを常に利用できるようにするには、これらのメンバーを CSQINP2 初期化入力データ・セットに追加するか、CSQUTIL プログラムを使用して、これらのキュー定義をロードします。

z/OS *z/OS* におけるサンプル *TSO* アプリケーションの名前

サンプル TSO アプリケーションごとに提供されるプログラムの名前と、ソース、JCL、および実行可能ファイル (メッセージ・ハンドラー・サンプルの場合のみ) があるライブラリーの名前を示します。

TSO アプリケーション・プログラムは次の表に要約されています。

- 1177 ページの表 177 メール管理プログラム・サンプル
- 1178 ページの表 178 メッセージ・ハンドラー・サンプル

これらのサンプルは、ISPF パネルを使用します。 したがって、このプログラムをリンク・エディットするときは、ISPF スタブである ISPLINK を組み込む必要があります。

表 *177. TSO* メール管理プログラム・サンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル
CSQ4CVD	言語に依存しない	IBM MQ for z/OS オブジェクト定義	SCSQPROC
CSQ40	言語に依存しない	ISPF メッセージ	SCSQMSGE
CSQ4RVD1	COBOL	CSQ4TVD1 を開始するための CLIST	SCSQCLST
CSQ4TVD1	COBOL	メニュー・プログラムのソース・プログラム	SCSQCOBS
CSQ4TVD2	COBOL	メール読み取りプログラムのソース・プログラム	SCSQCOBS
CSQ4TVD4	COBOL	メール送信プログラムのソース・プログラム	SCSQCOBS
CSQ4TVD5	COBOL	ニックネーム・プログラムのソース・プログラム	SCSQCOBS
CSQ4VDP1-6	COBOL	パネル定義	SCSQPNLA
CSQ4VD0	COBOL	データ定義	SCSQCOBC
CSQ4VD1	COBOL	データ定義	SCSQCOBC
CSQ4VD2	COBOL	データ定義	SCSQCOBC
CSQ4VD4	COBOL	データ定義	SCSQCOBC
CSQ4RCD1	C	CSQ4TCD1 を開始するための CLIST	SCSQCLST
CSQ4TCD1	C	メニュー・プログラムのソース・プログラム	SCSQC37S
CSQ4TCD2	C	メール読み取りプログラムのソース・プログラム	SCSQC37S
CSQ4TCD4	C	メール送信プログラムのソース・プログラム	SCSQC37S

表 177. TSO メール管理プログラム・サンプル (続き)			
メンバー名	**対象言語**	**説明**	**ライブラリーで提供されるソース・ファイル**
CSQ4TCD5	C	ニックネーム・プログラムのソース・プログラム	SCSQC37S
CSQ4CDP1-6	C	パネル定義	SCSQPNLA
CSQ4TC0	C	組み込みファイル	SCSQC370

表 178. TSO メッセージ・ハンドラー・サンプル				
メンバー名	**対象言語**	**説明**	**ライブラリーで提供されるソース・ファイル**	**ライブラリーで提供される実行可能ファイル**
CSQ4TCH0	C	データ定義	SCSQC370	なし
CSQ4TCH1	C	ソース・プログラム	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	ソース・プログラム	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	ソース・プログラム	SCSQC37S	SCSQLOAD
CSQ4RCH1	C および COBOL	CSQ4TCH1 または CSQ4TVH1 を開始するための CLIST	SCSQCLST	なし
CSQ4CHP1	C および COBOL	パネル定義	SCSQPNLA	なし
CSQ4CHP2	C および COBOL	パネル定義	SCSQPNLA	なし
CSQ4CHP3	C および COBOL	パネル定義	SCSQPNLA	なし
CSQ4CHP9	C および COBOL	パネル定義	SCSQPNLA	なし
CSQ4TVH0	COBOL	データ定義	SCSQCOBC	なし
CSQ4TVH1	COBOL	ソース・プログラム	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	ソース・プログラム	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	ソース・プログラム	SCSQCOBS	SCSQLOAD

z/OS z/OS での CICS 環境用のサンプル・アプリケーションの準備

CICS サンプル・プログラムを実行する前に、LOGMODE を 32702 にして CICS にログオンしてください。これは、サンプル・プログラムが 3270 モードの 2 画面を使用するように書き込まれているためです。

CICS 環境で実行するサンプル・アプリケーションを作成するには、以下のステップに従ってください。

1. BMS 画面定義ソースをアセンブルして、サンプルにシンボリック記述マップおよび物理画面マップを作成する。BMS 画面定義ソースは、ライブラリー **thlqual**.SCSQMAPS で提供されますが、**thlqual** はインストール先で使用する高水準の修飾子を示します。 これらのマップに名前を付けるときは、BMS 画面定義ソース名 (書き込みおよび読み取りサンプル・プログラムでは利用不可) を使用しますが、その名前の最後の文字は省略してください。
2. CICS 用の IBM MQ for z/OS アプリケーションを作成するときと同じステップに従ってアプリケーションを作成する。 これらのステップは、1032 ページの『z/OS での CICS アプリケーションの構築』に記述してあります。 使用するライブラリー・メンバーは、1179 ページの表 179、1179 ページの表 180、1180 ページの表 181、および 1180 ページの表 182 に示されています。

   あるいは、実行可能形式のサンプルが提供される場合には、それを thlqual.SCSQCICS ロード・ライブラリーから実行することもできます。

3. CICS システム定義 (CSD) データ・セットを更新して、CICS でマップ・セット、プログラム、およびトランザクションを識別できるようにする。 必要な定義は、メンバー **thlqual**.SCSQPROC(CSQ4S100) 内にあります。 これを行う方法については、 CICS Transaction Server for z/OS 4.1 製品資料 (CICS Transaction Server for z/OS 4.1、 CICS-IBM MQ アダプター) の「 *CICS-IBM MQ* アダプター 」セクションを参照してください。

   注 : 信用小切手サンプル・アプリケーションでは、サンプルが使用する VSAM データ・セットが作成されていないと、この段階でエラー・メッセージを受け取ります。

4. 信用小切手およびメール管理プログラムのサンプル・アプリケーションについて、それらが使用するキューがシステム上で使用できるか確認する。 信用小切手サンプルの場合、メンバー **thlqual**.SCSQPROC(CSQ4CVB) for COBOL、および **thlqual**.SCSQPROC(CSQ4CCB) for C で定義されています。 メール・マネージャー・サンプルの場合、メンバー **thlqual**.SCSQPROC (CSQ4CVD) に定義されています。 これらのキューを常に利用できるようにするには、これらのメンバーを CSQINP2 初期化入力データ・セットに追加するか、CSQUTIL プログラムを使用して、これらのキュー定義をロードします。

   キュー属性サンプル・アプリケーションについては、他のサンプル・アプリケーションで提供される 1 つまたは複数のキューを使用することができます。 代わりに、独自のキューを使用することもできます。 しかし提供される形式では、このサンプルは、その名前の最初の 8 バイトが文字 CSQ4SAMP であるキューによってのみ機能します。

z/OS *z/OS 上のサンプル CICS アプリケーションの名前*

このトピックでは、サンプル CICS アプリケーション用に提供されるプログラムの要約を示します。

CICS アプリケーション・プログラムは次の表に要約されています。

- 1179 ページの表 179 書き込みおよび読み取りのサンプル
- 1179 ページの表 180 キュー属性サンプル
- 1180 ページの表 181 メール管理プログラム・サンプル (COBOL のみ)
- 1180 ページの表 182 信用小切手サンプル
- 1182 ページの表 183 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル

表 *179. CICS* 書き込みおよび読み取りのサンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4CCK1	C	ソース・プログラムの書き込み	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	ソース・プログラムの読み取り	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	ソース・プログラムの読み取り	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	ソース・プログラムの書き込み	SCSQCOBS	SCSQCICS
CSQ4S100	言語に依存しない	CICS システム定義データ・セット	SCSQPROC	なし

表 *180. CICS* キュー属性サンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4CVC1	COBOL	ソース・プログラム	SCSQCOBS	SCSQCICS

表 180. CICS キュー属性サンプル (続き)

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル	ライブラリーで提供される実行可能ファイル
CSQ4VMSG	COBOL	メッセージ定義	SCSQCOBC	なし
CSQ4VCMS	COBOL	BMS 画面定義	SCSQMAPS	SCSQCICS (名前付き CSQ4ACM)
CSQ4CAC1	アセンブラー	ソース・プログラム	SCSQASMS	SCSQCICS
CSQ4AMSG	アセンブラー	メッセージ定義	SCSQMACS	なし
CSQ4ACMS	アセンブラー	BMS 画面定義	SCSQMAPS	SCSQCICS (名前付き CSQ4ACM)
CSQ4CCC1	C	ソース・プログラム	SCSQC37S	SCSQCICS
CSQ4CMSG	C	メッセージ定義	SCSQC370	なし
CSQ4CCMS	C	BMS 画面定義	SCSQMAPS	SCSQCICS (名前付き CSQ4ACM)
CSQ4S100	言語に依存しない	CICS システム定義データ・セット	SCSQPROC	なし

表 181. CICS メール管理プログラム・サンプル (COBOL のみ)

メンバー名	説明	ライブラリーで提供されるソース・ファイル
CSQ4CVD	IBM MQ for z/OS オブジェクト定義	SCSQPROC
CSQ4CVD1	メニュー・プログラムのソース	SCSQCOBS
CSQ4CVD2	メール読み取りプログラムのソース	SCSQCOBS
CSQ4CVD3	メッセージ表示プログラムのソース	SCSQCOBS
CSQ4CVD4	メール送信プログラムのソース	SCSQCOBS
CSQ4CVD5	ニックネーム・プログラムのソース	SCSQCOBS
CSQ4VDMS	BMS 画面定義ソース	SCSQMAPS
CSQ4S100	CICS システム定義データ・セット	SCSQPROC
CSQ4VD0	データ定義	SCSQCOBC
CSQ4VD3	データ定義	SCSQCOBC
CSQ4VD4	データ定義	SCSQCOBC

表 182. CICS 信用小切手サンプル

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル
CSQ4CVB	言語に依存しない	IBM MQ オブジェクト定義	SCSQPROC

表 *182. CICS* 信用小切手サンプル *(*続き*)*

メンバー名	対象言語	説明	ライブラリーで提供されるソース・ファイル
CSQ4CCB	言語に依存しない	IBM MQ オブジェクト定義	SCSQPROC
CSQ4CVB1	COBOL	ユーザー・インターフェース・プログラムのソース	SCSQCOBS
CSQ4CVB2	COBOL	信用アプリケーション管理プログラムのソース	SCSQCOBS
CSQ4CVB3	COBOL	当座預金口座プログラムのソース	SCSQCOBS
CSQ4CVB4	COBOL	配布プログラムのソース	SCSQCOBS
CSQ4CVB5	COBOL	代理店照会プログラムのソース	SCSQCOBS
CSQ4CCB1	C	ユーザー・インターフェース・プログラムのソース	SCSQC37S
CSQ4CCB2	C	信用アプリケーション管理プログラムのソース	SCSQC37S
CSQ4CCB3	C	当座預金口座プログラムのソース	SCSQC37S
CSQ4CCB4	C	配布プログラムのソース	SCSQC37S
CSQ4CCB5	C	代理店照会プログラムのソース	SCSQC37S
CSQ4CB0	C	組み込みファイル	SCSQC370
CSQ4CBMS	C	BMS 画面定義ソース	SCSQMAPS
CSQ4VBMS	COBOL	BMS 画面定義ソース	SCSQMAPS
CSQ4VB0	COBOL	データ定義	SCSQCOBC
CSQ4VB1	COBOL	データ定義	SCSQCOBC
CSQ4VB2	COBOL	データ定義	SCSQCOBC
CSQ4VB3	COBOL	データ定義	SCSQCOBC
CSQ4VB4	COBOL	データ定義	SCSQCOBC
CSQ4VB5	COBOL	データ定義	SCSQCOBC
CSQ4VB6	COBOL	データ定義	SCSQCOBC
CSQ4VB7	COBOL	データ定義	SCSQCOBC
CSQ4VB8	COBOL	データ定義	SCSQCOBC
CSQ4BAQ	言語に依存しない	VSAM データ・セットのソース	SCSQPROC
CSQ4FILE	言語に依存しない	CSQ4CVB3 によって使用される VSAM データ・セットを作成する JCL	SCSQPROC
CSQ4S100	言語に依存しない	CICS システム定義データ・セット	SCSQPROC

表 183. CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル		
メンバー名	**説明**	**ライブラリーで提供されるソース・ファイル**
CSQ4CVCN	単純メッセージ・コンシューム・プログラムのソース	SCSQCOBS
CSQ4CVCT	制御メッセージ・コンシューム・プログラムのソース	SCSQCOBS
CSQ4CVEV	イベント・ハンドラー・プログラムのソース	SCSQCOBS
CSQ4CVPT	メッセージ書き込みクライアント・プログラムのソース	SCSQCOBS
CSQ4CVRG	登録クライアント・プログラムのソース	SCSQCOBS
CSQ4S100	CICS システム定義データ・セット	SCSQPROC

z/OS z/OS での IMS 環境用のサンプル・アプリケーションの準備

信用小切手サンプル・アプリケーションの一部は、IMS 環境で実行できます。

このアプリケーションの一部を CICS サンプルで実行できるように作成する場合、まず 1178 ページの『z/OS での CICS 環境用のサンプル・アプリケーションの準備』 に記載されているステップに従ってください。

その後、次のステップに従ってください。

1. IMS 用の IBM MQ for z/OS アプリケーションを作成するときと同じステップに従ってアプリケーションを作成する。 これらのステップは、1033 ページの『IMS (BMP または MPP) アプリケーションの構築』 に記述してあります。 使用するライブラリー・メンバーは、1182 ページの表 184 に示してあります。
2. IMS でアプリケーション・プログラムおよびデータベースを識別できるようにする。 そのために、サンプルが PSBGEN、DBDGEN、ACB 定義、IMSGEN、および IMSDALOC ステートメントで提供されます。
3. この目的のために提供されるサンプル JCL (CSQ4ILDB) を調整、実行することによって、データベース CSQ4CA をロードする。 この JCL は、ファイル CSQ4BAQ からのデータによりデータベースをロードします。 IMS 制御領域をデータベース CSQ4CA の DD ステートメントで更新します。
4. この目的のために提供されるサンプル JCL を調整、実行することによって、当座預金口座プログラムをバッチ・メッセージ処理 (BMP) プログラムとして起動する。 この JCL は、バッチ向き BMP プログラムを起動します。 このプログラムをメッセージ向き BMP プログラムとして実行するには、IN= ステートメントを含む JCL 内の該当する行からコメント文字を除去します。

z/OS z/OS 上のサンプル IMS アプリケーションの名前

ここでは、信用小切手サンプル IMS アプリケーション用に提供されるソースおよび JCL をリストした表を示します。

表 184. 信用小切手 IMS サンプルのソースおよび JCL (C のみ)		
メンバー名	**説明**	**ライブラリーで提供されるもの**
CSQ4CVB	IBM MQ オブジェクト定義	SCSQPROC
CSQ4ICB3	当座預金口座プログラムのソース	SCSQC37S
CSQ4ICBL	当座預金口座データベースをロードするためのソース	SCSQC37S
CSQ4CBI	データ定義	SCSQC370

表 184. 信用小切手 IMS サンプルのソースおよび JCL (C のみ) (続き)		
メンバー名	**説明**	**ライブラリーで提供されるもの**
CSQ4PSBL	データベース・ロード・プログラムの PSBGEN JCL	SCSQPROC
CSQ4PSB3	当座預金口座プログラムの PSBGEN JCL	SCSQPROC
CSQ4DBDS	データベース CSQ4CA の DBDGEN JCL	SCSQPROC
CSQ4GIMS	CSQ4IVB3 および CSQ4CA の IMSGEN マクロ定義	SCSQPROC
CSQ4ACBG	CSQ4IVB3 のアプリケーション制御ブロック (ACB) 定義	SCSQPROC
CSQ4BAQ	データベースのソース	SCSQPROC
CSQ4ILDB	データベース・ロード・ジョブのサンプル実行 JCL	SCSQPROC
CSQ4ICBR	当座預金口座プログラムのサンプル実行 JCL	SCSQPROC
CSQ4DYNA	データベースの IMSDALOC マクロ定義	SCSQPROC

z/OS z/OS における書き込みサンプル

書き込みサンプル・プログラムは、MQPUT 呼び出しを使用して、メッセージをキューに入れます。

このソース・プログラムは、C および COBOL によりバッチ環境および CICS 環境で提供されています (1174 ページの表 172 および 1179 ページの表 179 を参照)。

書き込みサンプルの設計

プログラム・ロジックの流れを以下に示します。

1. MQCONN 呼び出しを使用して、キュー・マネージャーに接続する。 この呼び出しが異常終了したら、完了コードおよび理由コードを出力し、処理を停止します。

   注 : このサンプルを CICS 環境で実行する場合は、MQCONN 呼び出しを発行する必要はありません。 MQCONN 呼び出しを発行すると、DEF_HCONN が戻されます。 以降の MQI 呼び出しでは、接続ハンドル MQHC_DEF_HCONN を使用できます。
2. MQOO_OUTPUT オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。 この呼び出しへの入力で、このプログラムは、ステップ 1185 ページの『1』 で戻される接続ハンドルを使用します。 オブジェクト記述子構造体 (MQOD) では、このプログラムは、パラメーターとしてこのプログラムに渡されるキュー名フィールドを除くすべてのフィールドに対してデフォルト値を使用します。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力し、処理を停止します。
3. 必要な数のメッセージがキューに入れられるまで MQPUT 呼び出しを発行するループをプログラム内で作成する。 MQPUT 呼び出しが異常終了すると、そのループは速やかに中止され、これ以降 MQPUT 呼び出しは試行されなくなり、完了コードと理由コードが戻されます。
4. ステップ 1185 ページの『2』 で戻されるオブジェクト・ハンドルを指定した MQCLOSE 呼び出しを使用して、キューをクローズする。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力します。
5. ステップ 1185 ページの『1』 で戻される接続ハンドルを指定した MQDISC 呼び出しを使用して、キュー・マネージャーから切り離す。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力します。

   注 : このサンプルを CICS 環境で実行する場合は、MQDISC 呼び出しを発行する必要はありません。

z/OS *z/OS* におけるバッチ環境用の書き込みサンプル

このトピックでは、バッチ環境用の書き込みサンプルの考慮事項を示します。

このサンプルを実行するには、1173 ページの『z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行』で説明しているようにサンプル JCL を編集し、実行してください。

これらのプログラムは、EXEC PARM で次のパラメーター (C ではスペースで区切り、COBOL ではコンマで区切る) をとります。

1. キュー・マネージャー名 (4 文字)
2. 宛先キューの名前 (48 文字)
3. メッセージの数 (最大 4 桁)
4. メッセージに書き込む埋め込み文字 (1 文字)
5. メッセージに書き込む文字数 (最大 4 桁)
6. メッセージの持続的有効性 (1 文字: 持続性の場合は P、非持続性の場合は N)

これらのパラメーターのうちいずれかを不正確に入力すると、該当するエラー・メッセージを受け取ります。

サンプルからのメッセージは SYSPRINT データ・セットに書き込まれます。

使用上の注意

- このサンプルを簡潔に保つために、言語バージョン間には機能上の相違が多少あります。 しかし、サンプル実行 JCL、CSQ4BCJR、および CSQ4BVJR に示されるパラメーターのレイアウトを使用すれば、これらの相違を最小限に抑えることができます。 これらの相違は MQI とは何の関係もありません。
- CSQ4BCK1 を用いると送信するメッセージの数およびメッセージの長さを 5 桁以上で入力することができます。
- 2 つの数値フィールドには、範囲 1 から 9999 までの任意の数字を入力します。 入力する値は正の値でなければなりません。 例えば、1 つのメッセージを書き込む場合、値として 1、01、001、または 0001 と入力することができます。 非数値または負の値を入力すると、エラー・メッセージが戻る場合があります。 例えば、-1 を入力すると、COBOL プログラムは 1 バイトのメッセージを送りますが、C プログラムはエラー・メッセージを戻します。
- CSQ4BCK1 および CSQ4BVK1 の両プログラムでは、メッセージに持続的有効性を持たせたい場合に、持続パラメーター ++PER++ に P と入力する必要があります。 P を入力しないと、そのメッセージは非持続メッセージとなります。

z/OS *z/OS* 上の *CICS* 環境用の書き込みサンプル

このトピックでは、CICS 環境用の書き込みサンプルの考慮事項を示します。

このトランザクションは、コンマで区切られた以下のパラメーターをとります。

1. メッセージの数 (最大 4 桁)
2. メッセージに書き込む埋め込み文字 (1 文字)
3. メッセージに書き込む文字数 (最大 4 桁)
4. メッセージの持続的有効性 (1 文字: 持続性の場合は P、非持続性の場合は N)
5. 宛先キューの名前 (48 文字)

これらのパラメーターのうちいずれかを不正確に入力すると、該当するエラー・メッセージを受け取ります。

COBOL サンプルの場合、次のように入力して、CICS 環境で書き込みサンプルを呼び出します。

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

C サンプルの場合、次のように入力して、CICS 環境で書き込みサンプルを呼び出します。

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

サンプルからのメッセージはすべて画面上に表示されます。

使用上の注意

- このサンプルを簡潔に保つために、言語バージョン間には機能上の相違が多少あります。 これらの相違は MQI とは何の関係もありません。
- 48 文字よりも長いキュー名を入力すると、48 文字を超える部分は切り捨てられますが、エラー・メッセージは戻されません。
- このトランザクションに入る前に、「CLEAR」キーを押します。
- 2 つの数値フィールドには、範囲 1 から 9999 までの任意の数値を入力します。 入力する値は正の値でなければなりません。 例えば、1 つのメッセージを書き込む場合、値として 1、01、001、または 0001 と入力することができます。 非数値または負の値を入力すると、エラー・メッセージが戻る場合があります。 例えば、-1 を入力すると、COBOL プログラムは 1 バイトのメッセージを送り、C プログラムは malloc() からのエラーで異常終了します。
- CSQ4CCK1 および CSQ4CVK1 の両プログラムでは、メッセージに持続的有効性を持たせる必要がある場合には、持続パラメーターに P と入力します。 非持続メッセージの場合には、持続パラメーターに N と入力します。 それ以外の値を入力すると、エラー・メッセージが戻されます。
- プログラム起動時に設定されたパラメーターを除くすべてのパラメーターに対してデフォルト値が使用されるため、このメッセージは同期点で書き込まれます。

z/OS *z/OS における読み取りサンプル*

Get サンプル・プログラムは、MQGET 呼び出しを使用するキューからメッセージを取得します。

このソース・プログラムは、C および COBOL によりバッチ環境および CICS 環境で提供されています (1174 ページの表 172 および 1179 ページの表 179 を参照)。

z/OS *z/OS における読み取りサンプルの設計*

読み取りサンプルの設計、および考慮すべき使用上の注意について説明します。

プログラム・ロジックの流れを以下に示します。

1. MQCONN 呼び出しを使用して、キュー・マネージャーに接続する。 この呼び出しが異常終了したら、完了コードおよび理由コードを出力し、処理を停止します。

   注 : このサンプルを CICS 環境で実行する場合は、MQCONN 呼び出しを発行する必要はありません。 MQCONN 呼び出しを発行すると、DEF_HCONN が戻されます。 以降の MQI 呼び出しでは、接続ハンドル MQHC_DEF_HCONN を使用できます。
2. MQOO_INPUT_SHARED オプションおよび MQOO_BROWSE オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。 この呼び出しへの入力で、このプログラムは、ステップ 1185 ページの『1』 で戻される接続ハンドルを使用します。 オブジェクト記述子構造体 (MQOD) では、このプログラムは、パラメーターとしてこのプログラムに渡されるキュー名フィールドを除くすべてのフィールドに対してデフォルト値を使用します。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力し、処理を停止します。
3. 必要な数のメッセージがキューから取り出されるまで MQGET 呼び出しを発行するループをプログラム内で作成する。 MQGET 呼び出しが異常終了すると、そのループは速やかに中止され、これ以降 MQGET 呼び出しは試行されなくなり、完了コードと理由コードが戻されます。 MQGET 呼び出しで次のオプションが指定されます。

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT または MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST および MQGMO_BROWSE_NEXT

これらのオプションについては、MQGET に説明があります。 各メッセージごとに、メッセージ番号に続いてメッセージの長さとメッセージ・データが出力されます。

4. ステップ 1185 ページの『2』 で戻されるオブジェクト・ハンドルを指定した MQCLOSE 呼び出しを使用して、キューをクローズする。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力します。
5. ステップ 1185 ページの『1』 で戻される接続ハンドルを指定した MQDISC 呼び出しを使用して、キュー・マネージャーから切り離す。 MQOPEN 呼び出しが異常終了したら、完了コードおよび理由コードを出力します。

注 : このサンプルを CICS 環境で実行する場合は、MQDISC 呼び出しを発行する必要はありません。

使用上の注意

- このサンプルを簡潔に保つために、言語バージョン間には機能上の相違が多少あります。 しかし、サンプル実行 JCL、CSQ4BCJR、および CSQ4BVJR に示されるパラメーターのレイアウトを使用すれば、これらの相違を最小限に抑えることができます。 これらの相違は MQI とは何の関係もありません。
- CSQ4BCJ1 によって取り出すメッセージの数を 5 桁以上で入力することができます。
- 64 KB を超えるメッセージの部分は切り捨てられます。
- CSQ4BCJ1 による表示は、最初のヌル (¥0) 文字が表示されるまでに限られます。したがって、CSQ4BCJ1 で正しく表示できるのは文字メッセージのみです。
- メッセージ数の数値フィールドには、範囲 1 から 9999 までの任意の数字を入力します。 入力する値は正の値でなければなりません。 例えば、1 つのメッセージを読み取る場合、値として 1、01、001、または 0001 と入力することができます。 非数値または負の値を入力すると、エラー・メッセージが戻る場合があります。 例えば、-1 を入力すると、COBOL プログラムは 1 つのメッセージを取り出しますが、C プログラムはメッセージを取り出しません。
- CSQ4BCJ1 および CSQ4BVJ1 の両プログラムでは、メッセージをブラウズしたい場合には、読み取りパラメーター ++GET++ に B と入力します。
- CSQ4BCJ1 および CSQ4BVJ1 の両プログラムでは、メッセージを同期点で取り出すには、同期点パラメーター ++SYNC++ に S と入力します。

z/OS *z/OS におけるバッチ環境用の読み取りサンプル*

このサンプルを実行するには、1173 ページの『z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行』 で説明しているようにサンプル JCL を編集し、実行してください。

これらのプログラムは、EXEC PARM で次のパラメーター (C ではスペースで区切り、COBOL ではコンマで区切る) をとります。

1. キュー・マネージャー名 (4 文字)
2. 宛先キューの名前 (48 文字)
3. 読み取るメッセージの数 (最大 4 桁)
4. ブラウズ/読み取りメッセージ・オプション (1 文字: ブラウズするには B で、メッセージの破壊読み出しには D)
5. 同期点制御 (1 文字: 同期点ありの場合は S で、同期点なしの場合は N)

これらのパラメーターのうちいずれかを不正確に入力すると、該当するエラー・メッセージを受け取ります。

サンプルからの出力は SYSPRINT データ・セットに書き込まれます。

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
```

```
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *z/OS 上の CICS 環境の読み取りサンプル*

CICS 環境用の読み取りサンプルの特殊な考慮事項を示します。

このトランザクションは、EXEC PARM でコンマで区切られた以下のパラメーターをとります。

1. 読み取るメッセージの数 (最大 4 桁)
2. ブラウズ/読み取りメッセージ・オプション (1 文字: ブラウズするには B で、メッセージの破壊読み出しには D)
3. 同期点制御 (1 文字: 同期点ありの場合は S で、同期点なしの場合は N)
4. 宛先キューの名前 (48 文字)

これらのパラメーターのうちいずれかを不正確に入力すると、該当するエラー・メッセージを受け取ります。

COBOL サンプルの場合、次のように入力して、CICS 環境で読み取りサンプルを呼び出します。

```
MVGT,9999,B,S,QUEUE.NAME
```

C サンプルの場合、次のように入力して、CICS 環境で書き込みサンプルを呼び出します。

```
MCGT,9999,B,S,QUEUE.NAME
```

メッセージはキューから取り出されると、CICS トランザクションと同じ名前で CICS の一時記憶キューに入れられます (例えば C サンプルの場合は MCGT)。

以下に読み取りサンプルの出力例を示します。

```
**************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
**************************** BOTTOM OF QUEUE **********************
```

使用上の注意

- このサンプルを簡潔に保つために、言語バージョン間には機能上の相違が多少あります。 これらの相違は MQI とは何の関係もありません。
- 48 文字よりも長いキュー名を入力すると、48 文字を超える部分は切り捨てられますが、エラー・メッセージは戻されません。
- このトランザクションに入る前に、「CLEAR」キーを押します。
- CSQ4CCJ1 による表示は、最初のヌル (¥0) 文字が表示されるまでに限られます。したがって、CSQ4CCJ1 で正しく表示できるのは文字メッセージのみです。
- 数値フィールドには、範囲 1 から 9999 までの任意の数値を入力します。 入力する値は正の値でなければなりません。 例えば、1 つのメッセージを読み取る場合、値として 1、01、001、または 0001 と入力することができます。 非数値または負の値を入力すると、エラー・メッセージが戻る場合があります。
- C で 24 526 バイトより長いメッセージ、COBOL で 9 950 バイトより 長いメッセージは、切り捨てられます。 これは、CICS の一時記憶キューの使用法によるものです。
- CSQ4CCK1 および CSQ4CVK1 の両方のプログラムで、メッセージをブラウズする場合は、get パラメーターに B を入力します。それ以外の場合は、D を入力します。 これは、破壊的 MQGET 呼び出しを実行します。 それ以外の値を入力すると、エラー・メッセージが戻されます。

- CSQ4CCJ1 および CSQ4CVJ1 の両プログラムでは、メッセージを同期点で取り出すには、同期点パラメーターに S と入力します。 同期点パラメーターに N を入力すると、MQGET 呼び出しは同期点以外の時点で発行されます。 それ以外の値を入力すると、エラー・メッセージが戻されます。

z/OS z/OS でのブラウズ・サンプル

ブラウズ・サンプルは、MQGET 呼び出しを使用してキュー上のメッセージを表示する方法を示すバッチ・アプリケーションです。

アプリケーションは、キュー内のすべてのメッセージを網羅して、各メッセージの最初の 80 バイト分の文字を出力します。 このアプリケーションを使用すると、キュー上のメッセージを変更せずに参照することができます。

ソース・プログラムおよびサンプル実行 JCL は、COBOL、アセンブラー、PL/I、および C の各言語で提供されます (1175 ページの表 173 を参照)。

アプリケーションを開始するには、1173 ページの『z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行』に記述されているサンプル実行 JCL を編集し、実行してください。 実行 JCL にキューの名前を指定することによって、いずれかのキューのメッセージを参照できます。

アプリケーションを実行して、キューにいくつかのメッセージがある場合、出力データ・セットは以下のような形になります。

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

キュー上にメッセージがない場合、データ・セットには見出しと `End of report` というメッセージのみが出力されます。 MQI 呼び出しでエラーが発生すると、完了コードおよび理由コードが出力データ・セットに追加されます。

z/OS z/OS におけるブラウズ・サンプルの設計

ブラウズ・サンプル・アプリケーションは、単一のプログラム・モジュールを使用します。各プログラム・モジュールはサポートされる各プログラム言語で提供されます。

プログラム・ロジックの流れを以下に示します。

1. 出力データ・セットをオープンし、報告書のタイトル行を出力する。 キュー・マネージャーおよびキューの名前が実行 JCL から渡されているかを検査します。 両方の名前が渡されている場合には、それらの名前を含む報告書の各行を出力します。 それらの名前が渡されていない場合には、エラー・メッセージを出力し、出力データ・セットをクローズし、処理を停止します。

   JCL から渡されるパラメーターをプログラムがテストする方法は、プログラムが書かれる言語に依存します。詳細については、1189 ページの『z/OS における言語依存設計上の考慮事項』を参照してください。
2. MQCONN 呼び出しを使用して、キュー・マネージャーに接続する。 この呼び出しが失敗すると、完了コードおよび理由コードを出力し、出力データ・セットをクローズしてから、処理を停止します。
3. MQOO_BROWSE オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。 この呼び出しへの入力で、このプログラムは、ステップ 1188 ページの『2』 で戻される接続ハンドルを使用します。 オブジェクト記述子構造体 (MQOD) では、このプログラムは、(ステップ 1188 ページの『1』 で渡される) キュー名以外のすべてのフィールドに対してデフォルト値を使用します。 この呼び出しが失

敗すると、完了コードおよび理由コードを出力し、出力データ・セットをクローズしてから、処理を停止します。

4. MQGET 呼び出しを使用して、キューの最初のメッセージをブラウズする。 この呼び出しへの入力で、このプログラムは次のように指定します。
   - ステップ 1188 ページの『2』 およびステップ 1188 ページの『3』 からの接続ハンドルおよびキュー・ハンドル
   - すべてのフィールドに初期値が設定された MQMD 構造体
   - 次の 2 つのオプション
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - メッセージからコピーしたデータを保持する 80 バイトのサイズのバッファー

   MQGMO_ACCEPT_TRUNCATED_MSG オプションを使用することによって、メッセージのサイズが呼び出しで指定された 80 バイトのバッファーより大きい場合でも、その呼び出しを完了させることができます。 メッセージがバッファーよりも大きいと、そのメッセージはバッファーに収まるように切り捨てられて、これを示すために完了コードと理由コードが設定されます。 サンプルは、報告書を読みやすくするためにメッセージの 81 文字目以降を切り捨てるように設計されています。 バッファー・サイズは DEFINE ステートメントによって設定されます。したがって、必要に応じて簡単にバッファー・サイズを変更できます。
5. MQGET 呼び出しが失敗するまで、以下のループを実行する。
   a. 以下のものを示す報告書の行を出力する。
      - メッセージの順序番号 (ブラウズ操作のカウント)
      - メッセージの実際の長さ (切り捨てられる前の長さ)。 この値は、MQGET 呼び出しの DataLength フィールドに戻されます。
      - メッセージ・データの先頭の 80 バイト

   b. MQMD 構造体の MsqId および CorrelId フィールドをヌルにリセットする。

   c. 次の 2 つのオプション付きの MQGET 呼び出しを使用して、次のメッセージを表示する。
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. MQGET 呼び出しが失敗したら、理由コードをチェックして、その失敗の原因が、ブラウズ・カーソルがキューの終わりに達したためかどうかを調べる。 この場合、End of report というメッセージを出力し、ステップ 1189 ページの『7』 に進みます。それ以外の場合は、完了コードおよび理由コードを出力し、出力データ・セットをクローズしてから、処理を停止します。
7. ステップ 1188 ページの『3』 で戻されるオブジェクト・ハンドルを指定した MQCLOSE 呼び出しを使用して、キューをクローズする。
8. ステップ 1188 ページの『2』 で戻される接続ハンドルを指定した MQDISC 呼び出しを使用して、キュー・マネージャーから切り離す。
9. 出力データ・セットをクローズし、処理を停止する。

z/OS *z/OS における言語依存設計上の考慮事項*

ソース・モジュールは、4 つのプログラミング言語でブラウズ・サンプルに提供されます。

ソース・モジュール間では 2 つの大きな相違があります。

- 実行 JCL から渡されるパラメーターをテストする際に、COBOL、PL/I、およびアセンブラー言語のモジュールはコンマ文字 (,) を検索します。 JCL が PARM=(,LOCALQ1) を渡すと、アプリケーションはデフォルトのキュー・マネージャーでキュー LOCALQ1 をオープンしようとします。. コンマの後ろに名前がない場合 (またはコンマがない場合)、アプリケーションはエラーを戻します。 C モジュールはコンマを検索しません。 JCL が単一のパラメーター (例えば、PARM=('LOCALQ1')) を渡す場合、C モジュールはこれをデフォルト・キュー・マネージャーのキュー名として使用します。

- アセンブラー言語モジュールを簡潔に保つため、出力報告書を作成するときに日付形式 *yy/ddd* (例えば、05/116) を使用します。 その他のモジュールでは、*mm/dd/yy* の形式のカレンダー日付を使用します。

z/OS *z/OS におけるメッセージ印刷サンプル*

メッセージ印刷サンプルは、MQGET 呼び出しを使用してキューからすべてのメッセージを削除する方法を示すバッチ・アプリケーションです。

メッセージ印刷サンプルは、以下の 3 つのパラメーターを使用します。

1. キュー・マネージャーの名前
2. ソース・キューの名前
3. プロパティーを指定するオプション・パラメーター

また、このサンプルでは、それぞれのメッセージごとに、メッセージ記述子フィールドに続いてメッセージ・データを出力します。 このプログラムは、データを 16 進数および文字 (印刷可能な場合) の両方で出力します。 文字が印刷可能でない場合、プログラムはその文字をピリオド (.) で置き換えます。メッセージをキューで書き込むアプリケーションで問題を診断するときに、このプログラムを使用できます。

プロパティー・パラメーターで許容される値は、次のとおりです。

表 *185.* プロパティー・パラメーターで許容される値

値	動作
0	デフォルトの動作。 アプリケーションに送られるプロパティーは、メッセージの取得元の **PropertyControl** キュー属性に応じて異なります。

1	メッセージ・ハンドルが作成されて、MQGET と共に使用されます。 メッセージ記述子 (またはエクステンション) に含まれるプロパティーを除くメッセージ・プロパティーは、メッセージ記述子と同様の方式で表示されます。 以下に例を示します。
	`****Message properties****`
	`  property name: property value`
	または、使用可能なプロパティーが存在しない場合は、次のようになります。
	`****Message properties****`
	`  None`
	メッセージ記述子の場合と同様に、数値は `printf` を使って表示され、ストリング値は単一引用符で囲まれ、バイト・ストリングは X および単一引用符で囲まれます。

2	MQGMO_NO_PROPERTIES が指定され、メッセージ記述子プロパティーだけが戻されます。
3	MQGMO_PROPERTIES_FORCE_MQRFH2 が指定され、すべてのプロパティーがメッセージ・データ内に戻されます。
4	MQGMO_PROPERTIES_COMPATIBILITY が指定され、IBM MQ プロパティーが含まれるかどうかに応じてすべてのプロパティーを戻すことができます。含まれない場合、プロパティーは破棄されます。

メッセージをキューから除去するのではなくブラウズするように、アプリケーションを変更することができます。 これを行うには、1191 ページの『z/OS におけるメッセージ印刷サンプルの設計』 で示すように、-DBROWSE オプション付きでコンパイルして BROWSE マクロを定義します。 SCSQLOAD ライブラリーに実行可能コードが用意されています。 モジュール CSQ4BCG0 は -DBROWSE 付きでビルドされています。モジュール CSQ4BCG1 はキューを破壊的に読み取ります。

このアプリケーションには、C 言語で書かれた単一のソース・プログラムがあります。 また、サンプル実行 JCL コードも提供されます (1175 ページの表 174 を参照)。

アプリケーションを開始するには、1173 ページの『z/OS におけるバッチ環境用のサンプル・アプリケーションの作成と実行』に記述されているサンプル実行 JCL を編集し、実行してください。 アプリケーショ

ンを実行して、キューにいくつかのメッセージがある場合、出力データ・セットは 1191 ページの図 139 に示すようになります。

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR   '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C54040404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C54040404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '   '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

図 *139.* メッセージ印刷サンプル・アプリケーションからの報告書の例

z/OS *z/OS* におけるメッセージ印刷サンプルの設計

メッセージ印刷サンプル・アプリケーションは、C 言語で作成された単一のプログラムを使用します。

プログラム・ロジックの流れを以下に示します。

1. キュー・マネージャーおよびキューの名前が実行 JCL から渡されているかを検査します。 それらの名前が渡されていない場合には、エラー・メッセージを出力し、処理を停止します。
2. MQCONN 呼び出しを使用して、キュー・マネージャーに接続する。 この呼び出しが失敗すると、完了コードおよび理由コードを出力し、処理を停止します。正常に呼び出された場合は、キュー・マネージャーの名前を出力します。
3. MQOO_INPUT_SHARED オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。

   注 : アプリケーションがメッセージをキューから削除するのではなく参照するようにする場合は、-DBROWSE を使用してサンプルをコンパイルするか、ソースの冒頭に `#define BROWSE` を追加しま

す。 これによりマクロ・プリプロセッサーは、コンパイル時に MQOO_BROWSE オプションを選択するための行をプログラムに追加します。

この呼び出しへの入力で、このプログラムは、ステップ 1191 ページの『2』 で戻される接続ハンドルを使用します。 オブジェクト記述子構造体 (MQOD) では、このプログラムは、(ステップ 1191 ページの『1』 で渡される) キュー名以外のすべてのフィールドに対してデフォルト値を使用します。 この呼び出しが失敗すると、完了コードおよび理由コードを出力し、処理を停止します。正常に呼び出された場合は、キューの名前を出力します。

4. メッセージ・プロパティーを取得するためにメッセージ・ハンドルを使用する 場合、MQCRTMH を使用して、後続の MQGET 呼び出しで使用できるようにハンドルを作成する。 この呼び出しが正常終了しない場合、完了コードと理由コードを出力し、処理を停止します。
5. メッセージ読み取りオプションを、メッセージ・プロパティーの要求アクションを反映するように設定 する。
6. MQGET 呼び出しが失敗するまで、以下のループを実行する。
   a. バッファーをブランクに初期化する。これによって、メッセージ・データは、バッファー内にある既存のデータによって破壊されません。
   b. MQMD 構造体の `MsgId` および `CorrelId` フィールドをヌルに設定する。これによって、MQGET 呼び出しはキューにある最初のメッセージを選択します。
   c. MQGET 呼び出しを使用して、キューからメッセージを読み取る。 この呼び出しへの入力で、このプログラムは次のように指定します。
      - ステップ 1191 ページの『2』 および 1191 ページの『3』 からの接続ハンドルおよびオブジェクト・ハンドル
      - すべてのフィールドが初期値に設定された MQMD 構造体。 (`MsgId` および `CorrelId` は、MQGET 呼び出しごとにヌルにリセットされます。)
      - オプション MQGMO_NO_WAIT

        注 : アプリケーションがメッセージをキューから削除するのではなく参照するようにする場合は、-DBROWSE を使用してサンプルをコンパイルするか、ソースの冒頭に `#define BROWSE` を追加します。 これによりマクロ・プリプロセッサーは、コンパイル時に MQGMO_BROWSE_NEXT オプションを選択するための行をプログラムに追加します。 このオプションが、ブラウズ・カーソルが現行のオブジェクト・ハンドルで以前に使用されていないキューに対する呼び出しで使用されると、ブラウズ・カーソルは論理的に、最初のメッセージの前に置かれます。
      - メッセージからコピーしたデータを保持する 64 KB のサイズのバッファー
   d. printMD サブルーチンを呼び出す。 このサブルーチンは、メッセージ記述子内の各フィールドの名前とその内容を出力します。
   e. ステップ 1192 ページの『4』 でメッセージ・ハンドルを作成した場合、 printProperties サブルーチンを呼び出して、メッセージ・プロパティーを表示する。
   f. メッセージの長さと、それに続くメッセージ・データを出力する。 メッセージ・データの各行は、以下の形式となります。
      - データのこの部分の相対位置 (16 進数)
      - 16 バイトの 16 進データ
      - 印刷可能な場合、文字形式の同じ 16 バイトのデータ (印刷できない文字はピリオドに置換されます)。
7. MQGET 呼び出しが失敗した場合、理由コードをチェックして、失敗した原因が、キューにメッセージがなかったためかどうかを調べる。 このような場合、「No more messages」というメッセージを出力します。それ以外の場合は、完了コードおよび理由コードを出力します。 いずれの場合でも、1193 ページの『9』 のステップに進みます。

注 : MQGET 呼び出しは、64 KB を超えるデータからなるメッセージを検出すると、異常終了します。大きなメッセージを処理できるようにプログラムを変更するには、以下のいずれかを行います。

- MQGMO_ACCEPT_TRUNCATED_MSG オプションを MQGET 呼び出しに追加する。これによって、その呼び出しは最初の 64 KB のデータを読み取り、残りを廃棄します。

- この大きさのメッセージを検出したら、キューに残しておくようにする。
- バッファーのサイズを大きくする。

8. ステップ 1192 ページの『4』 でメッセージ・ハンドルを作成した 場合、MQDLTMH を呼び出してそれを削除する。
9. ステップ 1191 ページの『3』 で戻されるオブジェクト・ハンドルを指定した MQCLOSE 呼び出しを使用して、キューをクローズする。
10. ステップ 1191 ページの『2』 で戻される接続ハンドルを指定した MQDISC 呼び出しを使用して、キュー・マネージャーから切り離す。

z/OS z/OS におけるキュー属性サンプル

キュー属性サンプルは、会話型の CICS アプリケーションで、MQINQ 呼び出しおよび MQSET 呼び出しの使用方法を示します。

このサンプルでは、キューの **InhibitPut** 属性および **InhibitGet** 属性の値を照会する方法と、プログラムがキューにメッセージを書き込んだり、キューからメッセージを読み込んだりすることができないようにそれらの属性の値を変更する方法を示します。 プログラムをテストするときには、上記の方法によりキューをロック する場合もあります。

独自のキューによる不慮の妨害を防ぐために、このサンプルは、名前の最初の 8 バイトに文字 CSQ4SAMP があるキュー・オブジェクトでのみ作動するようになっています。 ただし、ソース・コードには、この制限を解除する方法を示すコメントが付いています。

ソース・プログラムは COBOL、アセンブラー、および C の各言語で提供されています (1179 ページの表 180 を参照)。

アセンブラー言語バージョンのサンプルでは、再入可能コードを使用しています。 これを行うには、そのバージョンのサンプル内の各 MQI 呼び出しのコードに次のような MF キーワードが含まれることに注意してください。

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(VL キーワードは、プログラムのデバッグ用の CICS 実行診断機能 (CEDF) 提供のトランザクションを使用できることを意味します。) 再入可能プログラムを作成する方法の詳細については、System/390 アセンブラー言語によるコーディングを参照してください。

アプリケーションを開始するには、CICS システムを始動し、次の CICS トランザクションを使用します。

- COBOL の場合、MVC1
- アセンブラー言語の場合、MAC1
- C の場合、MCC1

ステップ 3 で述べた CSD データ・セットを変更することによって、これらのトランザクションのうちのどれでも、その名前を変更することができます。

サンプルの設計

サンプルを開始すると、以下のフィールドが含まれた画面マップを表示します。

- キューの名前
- ユーザー要求 (有効なアクションは、照会、許可、または禁止)
- キューに対する書き込み操作の現在の状況
- キューに対する読み取り操作の現在の状況

最初の 2 つのフィールドは、ユーザー入力用のフィールドです。 最後の 2 つのフィールドには、アプリケーションによって値が設定されます。これらのフィールドには、ワード INHIBITED または ALLOWED が表示されます。

アプリケーションは、最初の 2 つのフィールドに入力した値の妥当性を検査します。 アプリケーションは、キュー名が文字 CSQ4SAMP で始まり、3 つの有効な要求のうちの 1 つを「Action (アクション)」フィール

ドに入力しているか検査します。 アプリケーションは、入力した値をすべて大文字に変換します。したがって、小文字を含む名前付きのキューは使用できません。

「**アクション**」 フィールドに inquire と入力すると、プログラム・ロジックのフローは次のようになります。

1. MQOO_INQUIRE オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。
2. セレクター MQIA_INHIBIT_GET および MQIA_INHIBIT_PUT を使用して、MQINQ を呼び出す。
3. MQCLOSE 呼び出しを使用して、キューをクローズする。
4. MQINQ 呼び出しの **IntAttrs** パラメーターに戻される属性を分析し、ワード「INHIBITED」または「ALLOWED」を必要に応じて、関係する画面フィールドに移動する。

「**アクション**」 フィールドに inhibit と入力すると、プログラム・ロジックのフローは次のようになります。

1. MQOO_SET オプション付きの MQOPEN 呼び出しを使用して、キューをオープンする。
2. セレクター MQIA_INHIBIT_GET および MQIA_INHIBIT_PUT を使用し、**IntAttrs** パラメーターに値 MQQA_GET_INHIBITED および MQQA_PUT_INHIBITED を指定して、MQSET を呼び出す。
3. MQCLOSE 呼び出しを使用して、キューをクローズする。
4. ワード「INHIBITED」を関係する画面フィールドに移動する。

「**アクション**」 フィールドに allow と入力すると、アプリケーションは、禁止要求の場合と同様の処理を実行します。 唯一の違いは、属性の設定と画面上に表示されるワードだけです。

アプリケーションがキューを開くと、キュー・マネージャーへのデフォルトの接続ハンドルが使用されます。(CICS は、CICS システムを始動するときに、キュー・マネージャーへの接続を確立します。) アプリケーションは、この段階で以下のエラーをトラップすることができます。

- アプリケーションがキュー・マネージャーに接続されていません。
- キューが存在しません。
- 当該ユーザーはこのキューへのアクセスを許可されていません。
- アプリケーションはキューのオープンを許可されていません。

その他の MQI エラーについては、アプリケーションは完了コードと理由コードを表示します。

z/OS *z/OS* におけるメール管理プログラム・サンプル

メール管理プログラム・サンプル・アプリケーションは、単一環境内および異なる環境にまたがって、メッセージの送受信を実際に試行する一連のプログラムです。 このアプリケーションは、異なるキュー・マネージャーを使用している場合でも、ユーザーがメッセージを交換できる単純な電子メーリング・システムです。

このアプリケーションは、MQOPEN 呼び出しを使用し、システム・コマンド入力キューに IBM MQ for z/OS コマンドを入れることによって、キューを作成する方法を示します。

このアプリケーションには次の 3 つのバージョンがあります。

- COBOL で作成された CICS アプリケーション
- COBOL で作成された TSO アプリケーション
- C で作成された TSO アプリケーション

z/OS *z/OS* におけるメール管理プログラム・サンプルの作成

メール管理プログラムは、2 つの環境で実行するバージョンで提供されます。 アプリケーションを実行する前に行うべき準備は、使用する環境によって異なります。

ユーザーは、サインオン・ユーザー ID が各システム上で同一になっている限り、TSO および CICS の両方からメール・キューとニックネーム・キューにアクセスできます。

別のキュー・マネージャーにメッセージを送信するためには、そのキュー・マネージャーにメッセージ・チャネルを設定しておく必要があります。 これを行うには、 チャネル制御機能で説明されている IBM MQ のチャネル制御機能を使用します。

TSO 環境用のサンプルの作成

次のステップを行います。

1. 1176 ページの『z/OS における TSO 環境用サンプル・アプリケーションの作成』で説明するように、サンプルを作成する。
2. サンプル用に提供されている CLIST を調整して、次のものを定義する。
   - パネルの位置
   - メッセージ・ファイルの位置
   - ロード・モジュールの位置
   - アプリケーションで使用したいキュー・マネージャーの名前

   個別の CLIST が各言語バージョンのサンプルごとに提供されています。
   - COBOL バージョン用: CSQ4RVD1
   - C バージョン用: CSQ4RCD1
3. アプリケーションによって使用されるキューがキュー・マネージャーにあるか確認します (キューは CSQ4CVD で定義されます。)

注 : VS COBOL II では ISPF でのマルチタスキングをサポートしていません。 これは、メール管理プログラム・サンプル・アプリケーションを分割画面の一方の側でしか使用できないことを意味します。 これを行うと、その結果は予測できません。

z/OS *z/OS* におけるメール管理プログラム・サンプルの実行

CICS Transaction Server for z/OS 環境でサンプルを開始するには、トランザクション MAIL を実行します。 CICS にまだサインオンしていない場合、アプリケーションはメールの送信先となるユーザー ID を入力するよう促します。

アプリケーションは、開始されると、メール・キューをオープンします。 このキューが存在しない場合は、アプリケーションがキューを作成します。 メール・キューの名前の形式は CSQ4SAMP.MAILMGR。 *userid*。 ここで、 *userid* は環境によって異なります。

TSO の場合
ユーザーの TSO ID

CICS 内
ユーザーの CICS サインオン、またはメール管理プログラムの開始時に入力を指示されたときにユーザーが入力したユーザー ID

メール管理プログラムが使用するキュー名の全部が大文字でなければなりません。

アプリケーションは、以下のためのオプションが含まれたメニュー・パネルを表示します。

- 着信メール読み取り
- 送信メール
- ニックネーム の作成

また、メニュー・パネルは、メール・キュー上に待機しているメッセージの数も表示します。 それぞれのメニュー・オプションがパネルを表示します。

着信メール読み取り
メール管理プログラムは、メール・キューにあるメッセージのリストを表示します。 (キュー上の最初の 99 個のメッセージのみが表示されます。) このパネルの例については、1199 ページの図 142 を参照してください。 このリストからメッセージを選択すると、そのメッセージの内容が表示されます (1200 ページの図 143 を参照)。

送信メール
パネルは入力を促します。

- メッセージ送信先ユーザーの名前
- メール・キューを所有するキュー・マネージャーの名前
- メッセージのテキスト

ユーザー名フィールドには、ユーザー ID またはメール管理プログラムを使用して作成したニックネームが入力できます。 ユーザーのメール・キューが使用中のキュー・マネージャーと同じキュー・マネージャーによって所有されている場合に、キュー・マネージャー名フィールドをブランクのままにしておくことができます。また、ユーザー名フィールドにニックネームを入力した場合は、キュー・マネージャー名フィールドをブランクにしておかなければなりません。

- ユーザー名のみを指定すると、プログラムは最初にその名前がニックネームであると見なし、その名前で定義されたオブジェクトにメッセージを送信します。 そのようなニックネームがない場合、プログラムはその名前のローカル・キューにメッセージを送信しようとします。
- ユーザー名とキュー・マネージャー名の両方を指定すると、プログラムはこれらの 2 つの名前で定義されたメール・キューにメッセージを送信します。

例えば、メッセージをリモート・キュー・マネージャー QM12 上のユーザー JONESM に送信したい場合、次の 2 つの方法のいずれかでメッセージを送信できます。

- 両方のフィールドを使用して、キュー・マネージャー QM12 のユーザー JONESM を指定する。
- そのユーザーのニックネーム (例えば、MARY) を定義したら、ユーザー名フィールドに MARY を入れ、キュー・マネージャー名フィールドには何も入れずに、そのユーザーにメッセージを送信する。

ニックネーム の作成
頻繁に連絡をとるユーザーにメッセージを送信するときに使用できる、覚えやすい名前を定義することができます。 相手のユーザーのユーザー ID とそのメール・キューを所有するキュー・マネージャーの名前を入力するように促されます。

ニックネームは、 CSQ4SAMP.MAILMGR。 *userid.nickname*。ここで、 *userid* はユーザー自身のユーザー ID、 *nickname* は使用するニックネームです。 このように構成された名前によって、ユーザーはそれぞれ独自のニックネームのセットを持つことができます。

プログラムが作成するキューのタイプは、 「ニックネームの作成」パネルの各フィールドをユーザーがどのように入力するかによって異なります。

- ユーザー名のみを指定する場合、またはそのキュー・マネージャー名がメール管理プログラムに接続されるキュー・マネージャーの名前と同じである場合、プログラムは別名キューを作成します。
- ユーザー名とキュー・マネージャー名の両方を指定すると (また、そのキュー・マネージャーがメール管理プログラムに接続されるキュー・マネージャーでない場合)、プログラムはリモート・キューのローカル定義を作成します。 プログラムは、この定義により解決されるキューの有無を検査しません。また、リモート・キュー・マネージャーが存在するかどうかも検査しません。

例えば、ユーザー ID が SMITHK で、JONESM というユーザー (リモート・キュー・マネージャー QM12 を使用する) に対して MARY というニックネームを作成すると、ニックネーム・プログラムは、CSQ4SAMP.MAILMGR.SMITHK.MARY という名前のリモート・キューのローカル定義を作成します。 この定義は、 キュー・マネージャー QM12 の CSQ4SAMP.MAILMGR.JONESM である Mary のメール・キューに解決されます。 キュー・マネージャー QM12 を使用している場合、プログラムは同じ名前 (CSQ4SAMP.MAILMGR.SMITHK.MARY) の別名キューを代わりに作成します。

C バージョンの TSO アプリケーションでは、ISPF のメッセージ処理機能を COBOL バージョンより多く活用しています。 C バージョンと COBOL バージョンでは異なるエラー・メッセージが表示されます。

z/OS *z/OS* におけるメール管理プログラム・サンプルの設計

以下のセクションでは、メール・マネージャー・サンプル・アプリケーションを構成する各プログラムについて説明します。

プログラムとアプリケーションが使用するパネルとの関係は、1197 ページの図 140 (TSO バージョンの場合) および 1198 ページの図 141 (CICS Transaction Server for z/OS バージョンの場合) に示されています。

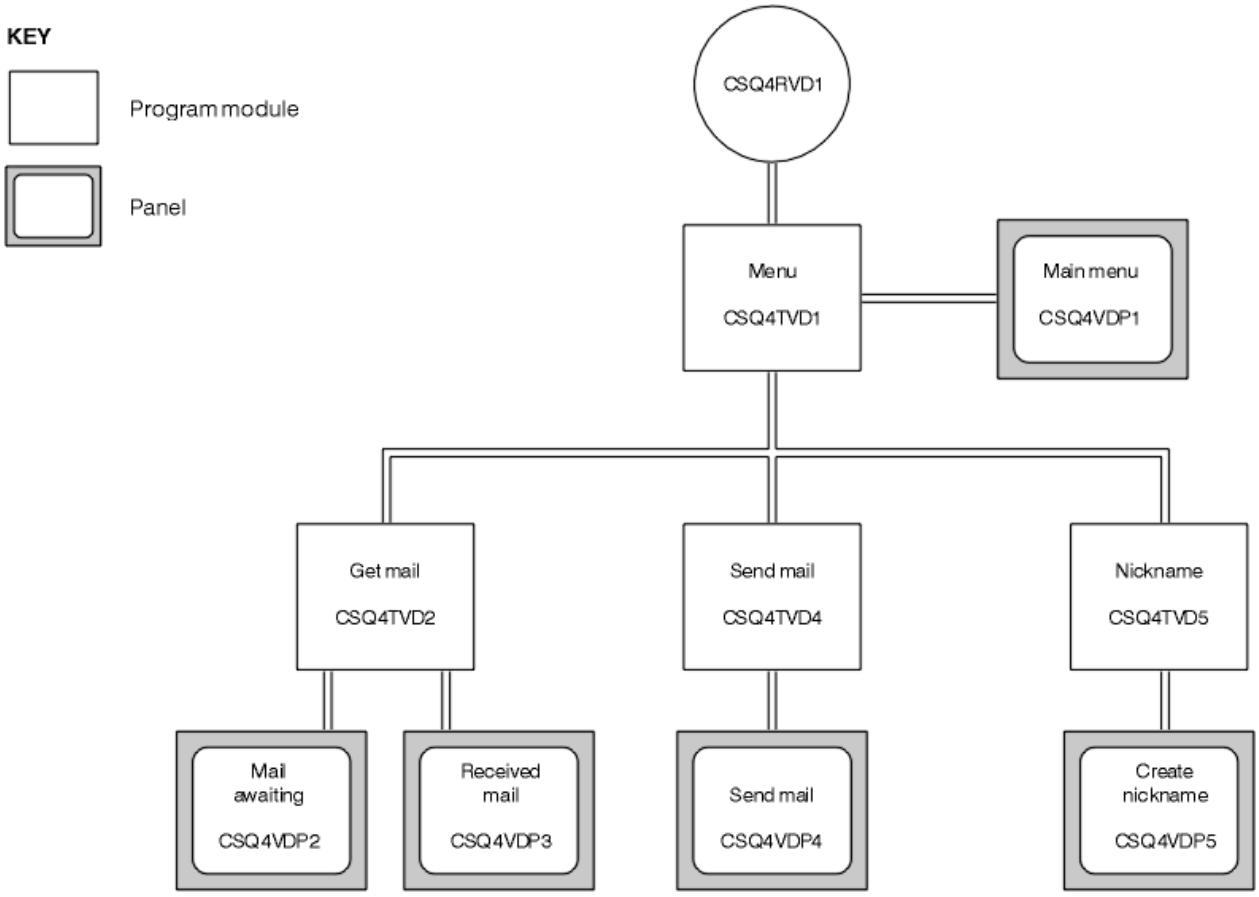


図 *140. TSO* バージョンのメール管理プログラムのプログラムとパネル

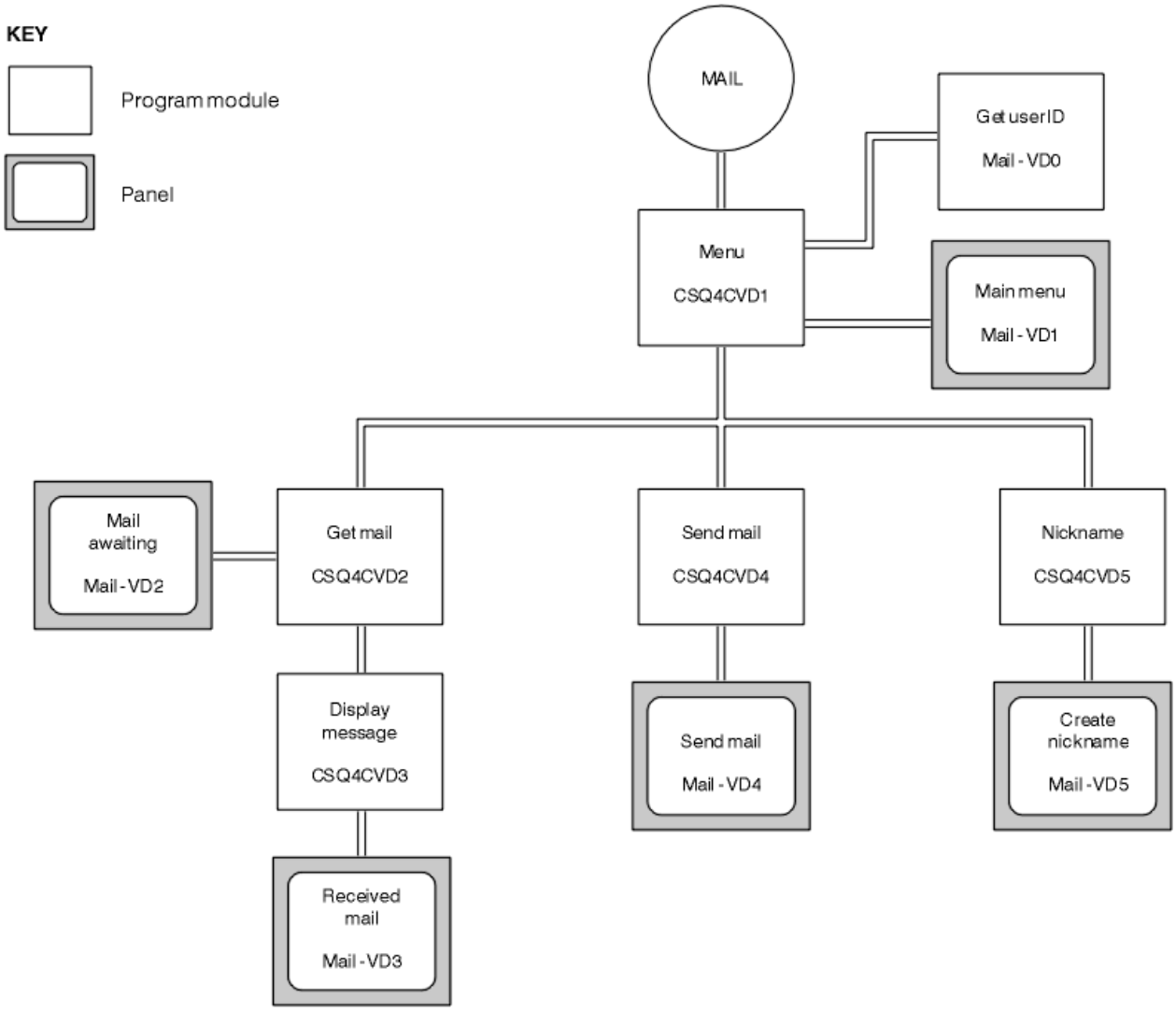


図 *141. CICS* バージョンのメール管理プログラムのプログラムとパネル

z/OS *z/OS* におけるメニュー・プログラム

TSO 環境では、メニュー・プログラムは CLIST によって呼び出されます。 CICS 環境では、プログラムはトランザクション MAIL によって呼び出されます。

メニュー・プログラム (TSO の場合は CSQ4TVD1、CICS の場合は CSQ4CVD1) は、一連のプログラムの中の初期プログラムです。 メニュー・プログラムは、メニュー (TSO の場合は CSQ4VDP1、CICS の場合は VD1) を表示し、メニューから他のプログラムが選択されると、そのプログラムを呼び出します。

メニュー・プログラムは、最初にユーザー ID を取得します。

- CICS バージョンのプログラムで、ユーザーが CICS に対してサインオンしている場合、ユーザー ID は CICS コマンド ASSIGN USERID を使用することにより得られます。 ユーザーがサインオンしていない場合、このプログラムは、サインオン・パネル (CSQ4VD0) を表示して、ユーザーにユーザー ID を入力するよう促します。 このプログラム内ではセキュリティー処理はありません。ユーザーはどのユーザー ID も指定できます。
- TSO バージョンでは、ユーザーの ID は CLIST 内の TSO から得られます。 このユーザー ID は、ISPF 共用プール内の変数としてメニュー・プログラムに渡されます。

プログラムは、ユーザー ID を取得した後、ユーザーがメール・キュー (CSQ4SAMP.MAILMGR。 *userid*)。メール・キューが存在しない場合、このプログラムはシステム・コマンド入力キューにメッセージを入れることによって、キューを作成します。 メッセージには、IBM MQ for z/OS コマンド DEFINE QLOCAL が入っています。 このコマンドが使用するオブジェクト定義は、キューの最大サイズを 9999 個に設定します。

また、このプログラムは、一時動的キューを作成して、システム・コマンド入力キューからの応答を処理します。 これを行う場合、プログラムは MQOPEN 呼び出しを使用して、動的キューに対するテンプレートとして SYSTEM.DEFAULT.MODEL.QUEUE を指定します。 キュー・マネージャーは、接頭部 CSQ4SAMP 付きの名前を使って一時動的キューを作成します。名前の残りの部分は、キュー・マネージャーによって生成されます。

次に、プログラムはユーザーのメール・キューをオープンし、キューの現行のサイズを照会して、キュー上のメッセージ数を検索します。 これを行う場合、プログラムは MQIA_CURRENT_Q_DEPTH セレクターを指定した MQINQ 呼び出しを使用します。

続いて、このプログラムはメニューを表示して、ユーザーが選択した項目を処理するループを実行します。PF3 キーを押すと、このループは停止します。 有効な選択を行うと、適切なプログラムが開始されます。行った選択が有効でないと、エラー・メッセージが表示されます。

z/OS *z/OS におけるメール読み取りプログラムとメッセージ表示機能プログラム*

TSO バージョンのアプリケーションでは、メール読み取りおよびメッセージ表示機能は同一のプログラムで実行されます (CSQ4TVD2)。 アプリケーションの CICS バージョンでは、これらの関数は別々のプログラム (CSQ4CVD2 および CSQ4CVD3) によって実行されます。

「メール待機」パネル (TSO の場合は CSQ4VDP2、CICS の場合は VD2) (例については、1199 ページの図 142 を参照) では、ユーザーのメール・キューにあるすべてのメッセージを表示します。 このリストを作成する場合、プログラムは MQGET 呼び出しを使用して、キュー上のすべてのメッセージをブラウズし、各メッセージに関する情報を保管します。 表示される情報のほかに、プログラムは各メッセージの `MsgId` および `CorrelId` を記録します。

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

図 *142.* メッセージ待ちリストを示すパネルの例

「Mail Awaiting (メール待機)」パネルから、ユーザーは 1 つのメッセージを選択し、そのメッセージの内容を表示することができます (例については、1200 ページの図 143 を参照してください)。 プログラムは、すべてのメッセージをブラウズしたときに記録した `MsgId` と `CorrelId` を使用して、MQGET 呼び出しによりこのメッセージをキューから除去します。 この MQGET 呼び出しは、MQGMO_SYNCPOINT オプションを使用して実行されます。 プログラムは、メッセージの内容を表示してから、同期点を宣言します。これによって、MQGET 呼び出しはコミットされ、メッセージは存在しなくなります。

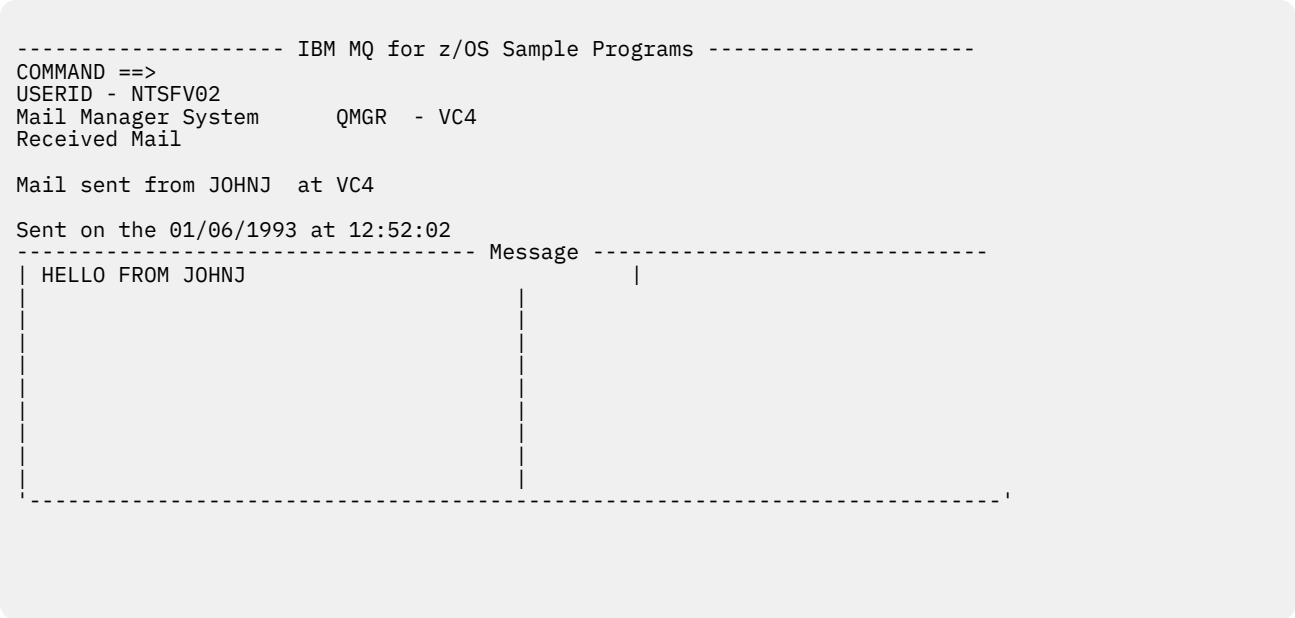


図 *143.* メッセージの内容を示すパネルの例

今回、メール管理プログラムで提供されている機能に対する拡張として目立ったのは、メッセージの内容を表示したあとでオプションとしてそのメッセージをキューに残しておくこともできるようになった点です。 これを行う場合、メッセージ表示後に、キューからメッセージを除去する MQGET 呼び出しをバックアウトする必要があります。

z/OS *z/OS* におけるメール送信プログラム

ユーザーーが「メールの送信」パネル (CSQ4VDP4 for TSO、VD4 for CICS) を完了すると、Send-Mail プログラム (CSQ4TVD4 for TSO、CSQ4CVD4 for CICS) は、受信側のメール・キューにメッセージを表示します。

これを行う場合、プログラムは MQPUT1 呼び出しを使用します。 メッセージの宛先は、「メール送信」パネル内のフィールドをユーザーがどのように設定したかによって異なります。

- ユーザーがユーザー名のみを指定している場合、プログラムは最初に、その名前がニックネームであると見なし、メッセージをその名前で定義されたオブジェクトに送信します。 そのようなニックネームがない場合、プログラムはその名前のローカル・キューにメッセージを送信しようとします。
- ユーザーがユーザー名とキュー・マネージャー名の両方を指定している場合、プログラムはこれらの 2 つの名前で定義されているメール・キューにメッセージを送信します。

このプログラムはブランク・メッセージを受け付けません。各メッセージ・テキスト行から先行するブランクを除去します。

MQPUT1 呼び出しが成功すると、プログラムはユーザー名と、メッセージが入れられたキュー・マネージャー名を示すメッセージを表示します。 この呼び出しが失敗すると、プログラムは特に、キューまたはキュー・マネージャーが存在しないことを示す理由コードを検査します。これらは、MQRC_UNKNOWN_OBJECT_NAME および MQRC_UNKNOWN_OBJECT_Q_MGR です。 このプログラムは、これらの各エラーについては、特有のエラー・メッセージを表示します。他のエラーについては、プログラムはこの呼び出しによって戻される完了コードと理由コードを表示します。

z/OS *z/OS* におけるニックネーム・プログラム

ユーザーがニックネームを定義する場合、ニックネーム・プログラム (TSO の場合は CSQ4TVD5、CICS の場合は CSQ4CVD5) はその名前の一部としてのニックネームを持つキューを作成します。

プログラムは、メッセージをシステム・コマンド入力キューに入れることによって、これを行います。 メッセージには、 IBM MQ for z/OS コマンド DEFINE QALIAS または DEFINE QREMOTE が含まれています。プログラムが作成するキューのタイプは、「ニックネームの作成」パネル (TSO の場合は CSQ4VDP5、CICS の場合は VD5) の各フィールドをユーザーがどのように設定したかによって異なります。

- ユーザー名のみを指定しているか、またはキュー・マネージャー名がメール管理プログラムと接続しているキュー・マネージャーの名前と同じである場合、プログラムは別名キューを作成します。
- ユーザー名とキュー・マネージャー名の両方を指定している場合 (かつ、キュー・マネージャーがメール管理プログラムと接続しているキュー・マネージャーと異なる場合)、プログラムはリモート・キューのローカル定義を作成します。 プログラムは、この定義により解決されるキューの有無を検査しません。 また、リモート・キュー・マネージャーが存在するかどうかも検査しません。

また、このプログラムは、一時動的キューを作成して、システム・コマンド入力キューからの応答を処理します。

キュー・マネージャーが、プログラムが予期する理由 (例えば、キューが既に存在する場合など) により、ニックネーム・キューを作成できない場合、プログラムは特有のエラー・メッセージを表示します。 キュー・マネージャーが、プログラムが予期しない理由によりニックネーム・キューを作成できない場合、プログラムはコマンド・サーバーによって戻されるエラー・メッセージを最大 2 つまで表示します。

注 : ニックネーム・プログラムは各ニックネームごとに、別名キューまたはリモート・キューのローカル定義のみを作成します。 これらのキュー名により解決されるローカル・キューは、ニックネームに含まれるユーザー ID がメール管理プログラム・アプリケーションを開始するために使用されるときにのみ作成されます。

z/OS *z/OS における信用小切手サンプル*

信用小切手サンプル・アプリケーションは、IBM MQ for z/OS で提供される多数の機能の使用方法を示す一連のプログラムです。 これは、アプリケーションの多くのコンポーネント・プログラムが、メッセージ・キューイング技法を使用して、互いにメッセージを渡す方法を示します。

このサンプルは、独立型の CICS アプリケーションとして実行できます。 しかし、CICS および IMS の両環境で提供される機能を使用するメッセージ・キューイング・アプリケーションの設計方法を示すために、1 つのモジュールが IMS バッチ・メッセージ処理プログラムとしても提供されます。 このサンプルへの拡張機能については、1212 ページの『z/OS での信用小切手サンプルに対する IMS 拡張』で説明しています。

また、複数のキュー・マネージャー上でサンプルを実行し、アプリケーションの各インスタンス間でメッセージを送信することもできます。 これを行うには、1212 ページの『z/OS において複数のキュー・マネージャーを使用する信用小切手サンプル』を参照してください。

CICS プログラムは C および COBOL で提供されます。 単一の IMS プログラムは C でのみ送達されます。 提供されたデータ・セットは、1180 ページの表 182 および 1182 ページの表 184 に表示されます。

このアプリケーションは、銀行の顧客から貸付を求められたときに、そのリスクを評価する方法を示します。 このアプリケーションは、銀行が貸付要求をどのように処理できるかを次の 2 つの方法で示します。

- 顧客を直接扱う場合、銀行員は口座と信用リスク情報に即時にアクセスする必要があります。
- 申込書を扱う場合、銀行員は口座と信用リスク情報に関する一連の要求を実行依頼しておいて、あとでその応答を処理することができます。

メッセージ・キューイング技法を明確にするために、アプリケーションでの財務面およびセキュリティー面についての詳細は簡略化してあります。

z/OS *z/OS における信用小切手サンプルの作成と実行*

信用小切手サンプルを作成し、実行するには、以下のステップに従います。

1. いくつかの例となる口座についての情報を保持する VSAM データ・セットを作成する。 これを行うには、データ・セット CSQ4FILE で提供される JCL (ジョブ制御言語) を編集し、実行します。
2. 1178 ページの『z/OS での CICS 環境用のサンプル・アプリケーションの準備』に示されるステップに従います。 (サンプルの IMS 拡張機能を使用したい場合に必要な追加の作業ステップについては、1212 ページの『z/OS での信用小切手サンプルに対する IMS 拡張』で説明しています。)
3. CKTI トリガー・モニターの開始 (IBM MQ for z/OS に付属) これは、キュー CSQ4SAMP.INITIATION.QUEUE(CICS トランザクション CKQC を使用)
4. アプリケーションを開始するには、CICS システムを始動し、トランザクション MVB1 を使用する。
5. 最初のパネルから「**Immediate (即時)**」または「**Batch (バッチ)**」照会を選択する。

即時照会のパネルとバッチ照会のパネルは似ています。1202 ページの図 144 に即時照会のパネルを示します。

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . . . . . . ____________________
Social security number ___ __ ____
Bank account name . . ______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : ____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

図 *144.* 信用小切手サンプル・アプリケーションの即時照会パネル

6. 適切なフィールドに口座番号と貸付金額を入力する。 これらのフィールドに入力する情報についての指針は、1202 ページの『照会パネルでの情報の入力』を参照してください。

照会パネルでの情報の入力

信用小切手サンプル・アプリケーションは、照会パネルの「**要求金額**」フィールドに入力されたデータが整数の形式になっているかを検査します。

次のいずれかの口座番号を入力すると、このアプリケーションは、VSAM データ・セット CSQ4BAQ 内の該当する口座名、平均口座残高、および信用評価指数を検索します。

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

その他のフィールドには、どのような情報を入力することも、何も入力しないこともできます。 アプリケーションは入力された情報をすべて保持し、生成する報告書に同じ情報を戻します。

z/OS *z/OS* における信用小切手サンプルの設計

このセクションでは、信用小切手サンプル・アプリケーションを構成する各プログラムの設計について説明します。

アプリケーションの設計時に検討されたいくつかの技法については、1209 ページの『z/OS における信用小切手サンプルの設計上の考慮事項』を参照してください。

1204 ページの図 145 では、アプリケーションを構成するプログラムと、これらのプログラムが使用するキューを示します。 この図では、分かりやすくするために、キュー名の接頭部 CSQ4SAMP はすべて省略されています。

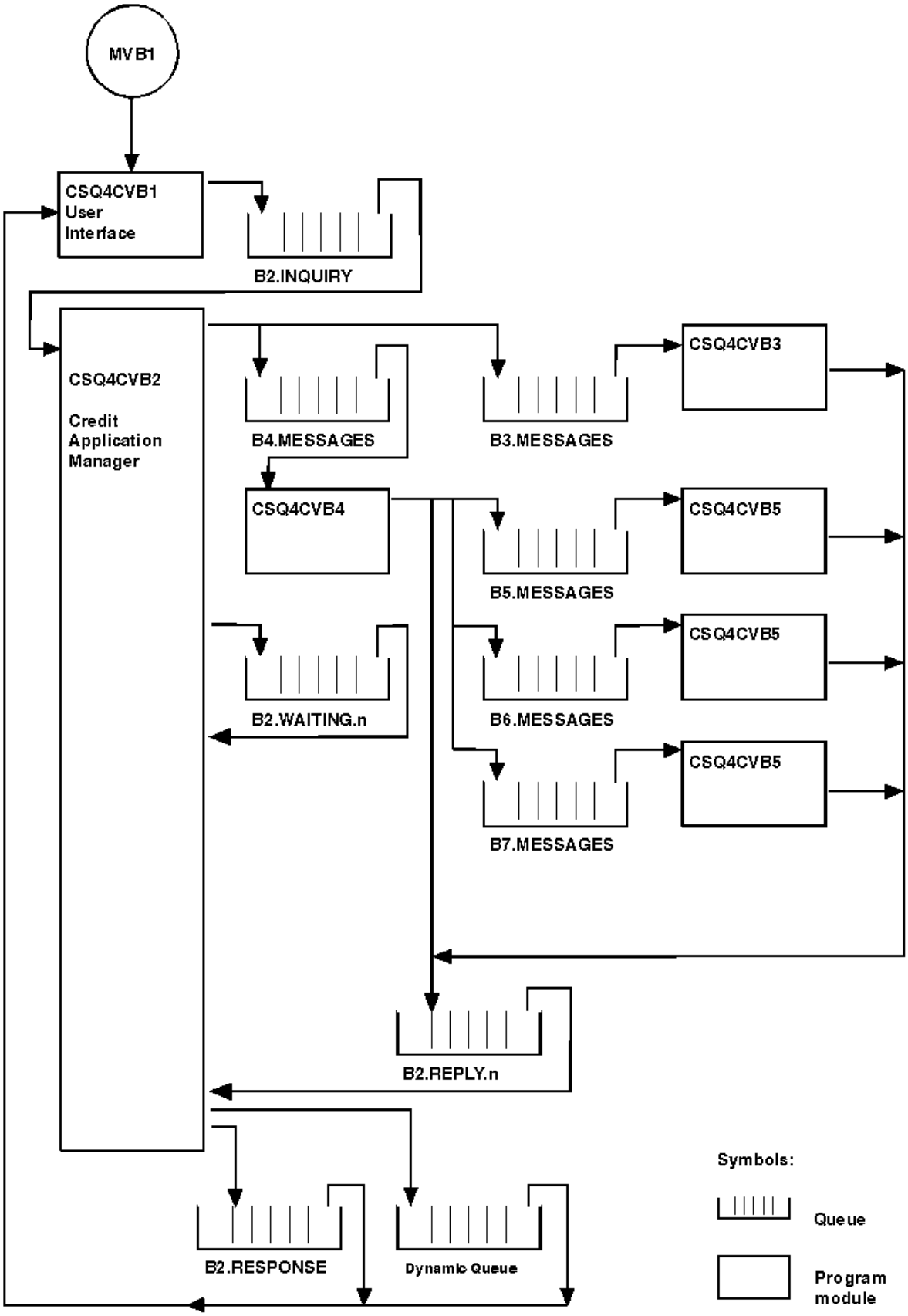


図 *145.* 信用小切手サンプル・アプリケーションのプログラムとキュー *(COBOL* プログラムのみ*)*

z/OS *z/OS におけるユーザー・インターフェース・プログラム (CSQ4CVB1)*

会話型モード CICS トランザクション MVB1 を開始すると、アプリケーションのユーザー・インターフェース・プログラムが開始されます。

このプログラムは、照会メッセージをキュー CSQ4SAMP.B2.INQUIRY に入れ、これらの照会に対する応答を、照会時に指定する応答先キューから取得します。 ユーザー・インターフェースから、即時照会およびバッチ照会を実行依頼することができます。

- 即時照会の場合、プログラムは応答先キューとして使用する一時動的キューを作成します。 これは、それぞれの照会ごとに応答先キューを持つということを意味します。
- バッチ照会の場合、ユーザー・インターフェース・プログラムは、キュー CSQ4SAMP.B2.RESPONSE から応答を得ます。 簡潔化するために、プログラムは、すべての照会に対する応答を 1 つの応答先キューから取得します。 銀行では MVB1 の各ユーザーごとに別々の応答先キューを使用したい場合があることは容易に分かります。そうすればユーザーは、それぞれ自分が開始した照会だけに対する応答を参照できるからです。

バッチ・モードと即時モードにおいて、アプリケーションで使用するメッセージの特性の重要な違いは、以下のとおりです。

- バッチ・モードで作業している場合、メッセージの優先順位は低くなります。したがって、メッセージは、即時モードで入力される貸付要求よりもあとに処理されます。 また、メッセージには持続的有効性があります。したがって、アプリケーションまたはキュー・マネージャーを再始動しなければならない場合に、メッセージは回復されます。
- 即時モードで作業している場合、メッセージの優先順位は高くなります。したがって、メッセージは、バッチ・モードで入力される貸付要求よりも先に処理されます。 また、メッセージには持続的有効性がありません。したがって、アプリケーションまたはキュー・マネージャーを再始動しなければならない場合に、メッセージは廃棄されます。

すべての場合、貸付要求メッセージの特性は、アプリケーション全体に伝えられます。 例えば、優先順位の高い要求から生じたメッセージはすべて、優先順位が高くなります。

z/OS *z/OS における信用アプリケーション管理プログラム (CSQ4CVB2)*

信用アプリケーション管理プログラム (CAM) は、信用小切手アプリケーションのほとんどの処理を実行します。

CAM は、トリガー・イベントがキュー CSQ4SAMP.B2.INQUIRY またはキュー CSQ4SAMP.B2.REPLY で発生したときに、CKTI トリガー・モニター (IBM MQ for z/OS に付属) によって開始されます。 *n*。ここで、*n* は応答キューのセットの 1 つを識別する整数です。 トリガー・メッセージには、トリガー・イベントが発生したキューの名前を含むデータが入っています。

CAM は CSQ4SAMP.B2.WAITING.n という形式の名前をもつキューを使用して、処理する照会についての情報を格納します。 それぞれが応答先キューと対になるようにキューには名前が付けられます。例えば、キュー CSQ4SAMP.B2.WAITING.3 には、特定の照会に関する入力データが含まれており、キュー CSQ4SAMP.B2.REPLY.3 には、すべて同じ照会に関係する一連の応答メッセージ (データベースを照会するプログラムからの応答) が含まれています。 この設計の背後にある理由を理解するには、1210 ページの『CAM での個別の照会および応答キューの使用』を参照してください。

開始論理

トリガー・イベントがキュー CSQ4SAMP.B2.INQUIRY 上で発生すると、CAM はキューを共用アクセス・キューとしてオープンします。 続いて、CAM は空の応答キューが見つかるまで、それぞれの応答キューをオープンしようとします。 空の応答キューが見つからない場合、CAM はそのことを記録し、正常に終了します。

トリガー・イベントがキュー CSQ4SAMP.B2.REPLY.n 上で発生すると、CAM はキューを排他的アクセス・キューとしてオープンします。 戻りコードにより、オブジェクトが既に使用中であることが報告されると、CAM は正常終了します。 これ以外のエラーが発生すると、CAM はそのエラーを記録し、処理を終了します。 CAM は対応する待機キューと照会キューをオープンしてから、メッセージの取得と処理を開始します。 待機キューから、CAM は部分的に完了された照会の詳細を回復します。

このサンプルを簡潔にするため、使用するキューの名前はプログラムに保持されています。 ビジネス環境では、キュー名はプログラムによってアクセスされるファイル内に保持されることになります。

照会キューからのメッセージの読み取り

CAM は最初に MQGMO_SET_SIGNAL オプション付きの MQGET 呼び出しを使用して、照会キューからメッセージを読み取ろうとします。 メッセージが即座に利用できる場合、そのメッセージは処理されます。メッセージが利用できない場合には、信号が設定されます。

次に、CAM は、同じオプション付きの MQGET 呼び出しを再び使用して、応答キューからメッセージを読み取ろうとします。 メッセージが即座に利用できる場合、そのメッセージは処理されます。メッセージが利用できない場合には、信号が設定されます。

両方の信号が設定されると、プログラムはいずれかの信号が通知されるまで待機します。 メッセージが利用可能であることを示す信号が通知されると、そのメッセージが取り出され、処理されます。 信号が期限切れになるかキュー・マネージャーが終了されると、プログラムは終了します。

CAM によって取り出されたメッセージの処理

CAM によって取り出されるメッセージは、次の 4 つのタイプのいずれかになります。

- 照会メッセージ
- 応答メッセージ
- 伝搬メッセージ
- 予期しないメッセージまたは不要なメッセージ

CAM は、1207 ページの『z/OS における CAM によって取り出されたメッセージの処理』で説明されているようにこれらのメッセージを処理します。

応答の送信

CAM は 1 つの照会に対して予期するすべての応答を受信すると、その応答を処理し、単一の応答メッセージを作成します。 これによって、CorrelId がすべて同一である応答メッセージにあるすべてのデータを 1 つのメッセージに統合します。 この応答は、元の貸付要求で指定された応答先キューに入れられます。 応答メッセージは、最終の応答メッセージの取り出しを含む同じ作業単位内に入れられます。 これは、キュー CSQ4SAMP.B2.WAITING.n 上に完了されたメッセージが存在しないようにすることにより、リカバリー処理を単純化するものです。

部分的に完了された照会のリカバリー

CAM は、受信したすべてのメッセージをキュー CSQ4SAMP.B2.WAITING.n にコピーします。 CAM は、メッセージ記述子のフィールドを次のように設定します。

- *Priority* の値はメッセージのタイプによって決定されます。
  - 要求メッセージでは、優先順位 = 3
  - データグラムでは、優先順位 = 2
  - 応答メッセージでは、優先順位 = 1
- *CorrelId* には、貸付要求メッセージの *MsgId* 値が設定されます。
- 他の MQMD フィールドは、受信したメッセージのフィールドから値がコピーされます。

照会が既に完了している場合は、応答処理時に、特定の照会に関するメッセージが待機キューから除去されます。 したがって、待機キューには常に、進行中の照会に関係するすべてのメッセージが入っています。これらのメッセージは、プログラムを再始動しなければならない場合に、進行中の照会の詳細を回復するために使用します。 伝搬や応答メッセージの前に照会メッセージが回復されるように、異なる優先順位が設定されています。

z/OS *z/OS における CAM によって取り出されたメッセージの処理*

信用アプリケーション管理プログラム (CAM) によって取り出されるメッセージは、4 つのタイプのいずれかになります。 CAM がメッセージを処理する方法は、メッセージのタイプによって異なります。

CAM によって取り出されるメッセージは、次の 4 つのタイプのいずれかになります。

- 照会メッセージ
- 応答メッセージ
- 伝搬メッセージ
- 予期しないメッセージまたは不要なメッセージ

CAM はこれらのメッセージを次のように処理します。

照会メッセージ

照会メッセージは、ユーザー・インターフェース・プログラムから送られてきます。 貸付要求ごとに照会メッセージが作成されます。

すべての貸付要求について、CAM は顧客の当座預金口座の平均残高を要求します。 CAM は、要求メッセージを別名キュー CSQ4SAMP.B2.OUTPUT.ALIAS に入れることによって、これを実行します。 このキュー名は、当座預金口座プログラム CSQ4CVB3 によって処理されるキュー CSQ4SAMP.B3.MESSAGES に解決されます。 CAM がメッセージをこの別名キューに入れることによって、応答先キューに適切な CSQ4SAMP.B2.REPLY.n キューが指定されます。 プログラム CSQ4CVB3 を、異なる名前の基本キューを処理する別のプログラムに簡単に置換できるように、別名キューがここで使用されます。 これを行うには、その名前が新規のキューに解決されるように別名キューを再定義します。 また、別名キューと基本キューに異なるアクセス許可を割り当てることもできます。

ユーザーが 10000 単位を超える貸付を要求すると、CAM は他のデータベースについても検査を開始します。 要求メッセージを配布プログラム CSQ4CVB4 で処理されるキュー CSQ4SAMP.B4.MESSAGES に入れることによって、これを行います。 このキューを処理するプロセスは、メッセージを、その他のレコード (クレジット・カードのヒストリー、普通預金口座、抵当支払いなど) にアクセスできるプログラムが処理するキューに伝搬します。 これらのプログラムからのデータは、書き込み操作で指定された応答先キューに戻されます。 さらに、伝搬メッセージはこのプログラムによって応答先キューに送信され、送信済みの伝搬メッセージの数を指定します。

ビジネス環境において、配布プログラムは、提供されているデータの形式を再設定して、他のタイプの銀行預金口座で求められる形式と一致させるようにします。

ここで述べるキューはすべて、リモート・システム上に置くことができます。

各照会メッセージごとに、CAM はメモリー常駐型のレコード照会テーブル (IRT) への入力を開始します。 このレコードには、次のものが含まれます。

- 照会メッセージの `MsgId`
- ReplyExp フィールドでは、予想される応答の数 (送信されたメッセージの数に等しい)
- ReplyRec フィールドでは、受信した応答の数 (この時点ではゼロ)
- PropsOut フィールドでは、伝搬メッセージが予想されるかの指示

CAM は以下の設定で照会メッセージを待機キューにコピーします。

- `Priority` に 3 を設定する。
- `CorrelId` に照会メッセージの `MsgId` 値を設定する。
- 他のメッセージ記述子フィールドには照会メッセージの記述子フィールド値を設定する。

伝搬メッセージ

伝搬メッセージには、配布プログラムが照会を転送したキューの数が入っています。 伝搬メッセージは次のように処理されます。

1. IRT 中の適切なレコードの ReplyExp フィールドに、送信したメッセージ数を付加する。 この情報はメッセージ内にあります。
2. IRT 中のレコードの ReplyRec フィールドを 1 つずつ増やす。
3. IRT 中のレコードの PropsOut フィールドを 1 つずつ減らす。

4. 待機キューにメッセージをコピーする。 CAM は Priority に 2 を設定し、メッセージ記述子の他のフィールドに伝搬メッセージの記述子フィールド値を設定します。

応答 メッセージ

応答メッセージには、当座預金口座プログラムへの要求の 1 つに対する応答、またはいずれかの代理店照会プログラムへの要求の 1 つに対する応答が入っています。 応答メッセージは次のように処理されます。

1. IRT 中のレコードの ReplyRec フィールドを 1 つずつ増やす。
2. Priority に 1 を設定し、メッセージ記述子の他のフィールドに応答メッセージの記述子フィールド値を設定して、メッセージを待機キューにコピーする。
3. ReplyRec = ReplyExp かつ PropsOut = 0 の場合、MsgComplete フラグを設定する。

その他のメッセージ

アプリケーションは他のメッセージを予期しません。 ただし、アプリケーションは、システムからのメッセージ・ブロードキャスト、または不明な CorrelIds が含まれる応答メッセージを受信することがあります。

CAM はこれらのメッセージをキュー CSQ4SAMP.DEAD.QUEUE に入れます。このキューの中でメッセージを調べることができます。 この書き込み操作が失敗すると、メッセージは失われ、プログラムは処理を続行します。 この部分のプログラムの設計の詳細については、1210 ページの『サンプルが予期しないメッセージを処理する方法』を参照してください。

z/OS *z/OS における当座預金口座プログラム (CSQ4CVB3)*

当座預金口座プログラムは、キュー CSQ4SAMP.B3.MESSAGES 上のトリガー・イベントによって開始されます。 このプログラムは、キューをオープンした後、待機オプションを指定した MQGET 呼び出しを使用してキューからメッセージを取得し、待機間隔は 30 秒に設定されます。

このプログラムは、VSAM データ・セット CSQ4BAQ の中から、貸付要求メッセージの口座番号を検索します。 このプログラムは、対応する口座名、平均残高、および信用評価指数を取り出すか、口座番号がデータ・セットにないことを通知します。

次に、このプログラムは、応答メッセージを (MQPUT1 呼び出しを使用して)、貸付要求メッセージで指定された応答先キューに入れます。 応答メッセージについて、プログラムは以下の処理を行います。

- 貸付要求メッセージの CorrelId をコピーする。
- MQPMO_PASS_IDENTITY_CONTEXT オプションを使用する。

プログラムは、待機間隔が時間切れになるまで、キューからメッセージを取得し続けます。

z/OS *z/OS における配布プログラム (CSQ4CVB4)*

この配布プログラムは、キュー CSQ4SAMP.B4.MESSAGES 上のトリガー・イベントによって開始されます。

クレジット・カードのヒストリー、普通預金口座、抵当支払いなどのレコードにアクセスできる他の代理店への貸付要求の配布をシミュレートする場合、このプログラムは、すべてのキューにある同一のメッセージのコピーを、名前リスト CSQ4SAMP.B4.NAMELIST に入れます。 これらのキューは 3 つあり、CSQ4SAMP.B *n*.MESSAGES という形式の名前が付けられています。ここで、*n* は 5、6、または 7 です。 ビジネス・アプリケーションでは、エージェンシーが別の場所に配置されている可能性があり、これらのキューはリモート・キューになる可能性があります。 これを示すためにサンプル・アプリケーションを修正したい場合は、1212 ページの『z/OS において複数のキュー・マネージャーを使用する信用小切手サンプル』を参照してください。

配布プログラムは次のステップに従って処理を実行します。

1. 名前リストの中から、このプログラムが使用するキューの名前を取得する。 このプログラムは、MQINQ 呼び出しを使用して、名前リスト・オブジェクトの属性を照会することによって、これを行います。
2. これらのキューおよび CSQ4SAMP.B4.MESSAGES をオープンする。
3. キュー CSQ4SAMP.B4.MESSAGES 上にメッセージがなくなるまで、以下のループを実行する。
   a. 待機オプション付きで、待機間隔を 30 秒に設定した MQGET 呼び出しを使用して、メッセージを取得する。

b. 名前リストに示されている各キューにメッセージを入れ、適切な CSQ4SAMP.B2.REPLY.n キューの名前を応答先キューに指定する。 このプログラムは、貸付要求メッセージの *CorrelId* をこれらのコピー・メッセージにコピーし、MQPUT 呼び出しで MQPMO_PASS_IDENTITY_CONTEXT オプションを使用します。

c. データグラム・メッセージをキュー CSQ4SAMP.B2.REPLY.n に送信して、正常に入れた、メッセージの数を示す。

d. 同期点を宣言する。

z/OS *z/OS における代理店照会プログラム (CSQ4CVB5/CSQ4CCB5)*

代理店照会プログラムは COBOL プログラムと C プログラムの両方で提供されています。 両方のプログラム共に設計は同じです。 これは、異なるタイプのプログラムが IBM MQ アプリケーション内で簡単に共存できること、およびそのようなアプリケーションを構成するプログラム・モジュールを容易に置き換えることができることを示しています。

プログラムのインスタンスは、以下の 3 つのキューのうち、いずれかのキュー上のトリガー・イベントによって開始します。

- COBOL プログラム (CSQ4CVB5) の場合
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- C プログラム (CSQ4CCB5) の場合、キュー CSQ4SAMP.B8.MESSAGES

注 : C プログラムを使用したい場合は、名前リスト CSQ4SAMP.B4.NAMELIST の定数を変更して、キュー CSQ4SAMP.B7.MESSAGES を CSQ4SAMP.B8.MESSAGES に置換する必要があります。 これを行う場合、次のいずれかを使用できます。

- IBM MQ for z/OS 操作および制御パネル
- ALTER NAMELIST コマンド
- CSQUTIL ユーティリティー

適切なキューをオープンしたら、このプログラムは、待機オプション付きで、待機間隔を 30 秒に設定した MQGET 呼び出しを使用して、キューからメッセージを取得します。

このプログラムは、VSAM データ・セット CSQ4BAQ の中から、貸付要求メッセージに渡された口座番号を検索することによって、代理店のデータベースの検索をシミュレートします。 続いて、このプログラムは、処理中のキューの名前を含む応答と、信用評価指数を作成します。 この処理を単純化するために、信用評価指数が無作為に選択されます。

応答メッセージを入れるときに、プログラムは MQPUT1 呼び出しを使用して、次の処理を行います。

- 貸付要求メッセージの `CorrelId` をコピーする。
- MQPMO_PASS_IDENTITY_CONTEXT オプションを使用する。

プログラムは、貸付要求メッセージで指定された応答先キューに、応答メッセージを送信します。 (応答先キューを所有するキュー・マネージャーの名前も、貸付要求メッセージに指定されます。)

z/OS *z/OS における信用小切手サンプルの設計上の考慮事項*

信用小切手サンプルの設計の考慮事項を示します。

このトピックでは、以下の情報について説明します。

CAM での個別の照会および応答キューの使用

このアプリケーションは、照会と応答の両方に対して単一のキューを使用することもできますが、以下のような理由から別々のキューを使用するように設計されました。

- プログラムが最大数の照会を処理しているときに、それ以上の照会があると、それらの照会はキューに残されます。 単一のキューを使用している場合、処理しきれない照会をキューから取り出して、別の場所に格納することが必要な場合があります。
- メッセージの通信量がそれを保証できるほど十分である場合、CAM の他のインスタンスを自動的に始動して、同一の照会キューを処理することは可能です。 しかし、プログラムは進行中の照会の記録をとる必要があり、これを行うために、プログラムはすべての応答を開始した照会に戻す必要があります。 1 つのキューのみを使用する場合、プログラムはメッセージをブラウズして、そのメッセージがこのプログラム用であるか、または別のプログラム用であるかを確認する必要があります。 これによって、操作効率が大幅に低下します。

  アプリケーションは、応答先キューと待機キューを対で使用することによって、複数の CAM をサポートし、処理中の照会を効率的に回復することができます。
- プログラムは信号を利用して、複数のキューを効率的に待つことができます。

サンプルがエラーを処理する方法

ユーザー・インターフェース・プログラムは、発生したエラーをユーザーに直接報告することによって、エラーを処理します。

これ以外のプログラムには、ユーザー・インターフェースがありません。 したがって、別の方法でエラーを処理しなければなりません。また、多くの場合において (例えば、MQGET 呼び出しが失敗した場合)、これらのプログラムはアプリケーションのユーザーの ID を識別できません。

これらのプログラムは、エラー・メッセージを CSQ4SAMP と呼ばれる CICS 一時記憶キューに入れます。 CICS で提供するトランザクション CEBR を使用して、このキューをブラウズできます。 また、このプログラムは、エラー・メッセージを CICS CSML ログに書き込むこともできます。

サンプルが予期しないメッセージを処理する方法

メッセージ・キューイング・アプリケーションを設計する場合には、キューが予期しないメッセージを受け取ったときの処理方法を決定する必要があります。

基本的に次の 2 通りの方法があります。

- アプリケーションは、予期しないメッセージの処理が終了するまで、作業を中断します。 これは多くの場合、アプリケーションがオペレーターに通知し、アプリケーションを終了して、自動的に再始動しないようにすることを意味します (トリガーをオフに設定することによって、これを行うことができます)。 この方法は、アプリケーションに関するすべての処理が、単一の予期しないメッセージによって停止される可能性があり、そのアプリケーションを再始動するには、オペレーターの介入が必要なことを意味します。
- アプリケーションが処理を続行するには、処理しているキューからメッセージを除去して、そのメッセージを別の場所に入れます。 このメッセージを入れておく最も良い場所は、システム上の送達不能キューです。

2 番目の方法にすると、

- オペレーターまたは別のプログラムは、送達不能キュー上にあるメッセージを調べて、そのメッセージがどこから送信されてきたかを見つけ出す必要があります。
- 送達不能キューに入れることが不可能な場合、その予期しないメッセージは失われます。
- 予期しないメッセージが送達不能キューに設定されているメッセージの限度より大きいか、またはプログラムのバッファー・サイズよりも大きい場合、そのメッセージは切り捨てられます。

外部からの影響を最小限に抑えて、アプリケーションがすべての照会を円滑に処理できるように、信用小切手サンプル・アプリケーションでは、2 番目の方法を使用します。 サンプルを、同一のキュー・マネージャーを使用する別のアプリケーションと区別しておくため、信用小切手サンプルは、システムの送達不能キューを使用しません。代わりに、このサンプルの送達不能キューを使用します。 このキューの名前は

CSQ4SAMP.DEAD.QUEUE です。 このサンプルは、ソース・プログラムに対して提供されるバッファー領域よりも大きいメッセージの余長をすべて切り捨てます。 ここで、ブラウズ・サンプル・アプリケーションを使用して、このキューにあるメッセージをブラウズしたり、メッセージ印刷サンプル・アプリケーションを使用して、そのメッセージとメッセージ記述子を印刷することができます。

ただし、サンプルを拡張して、複数のキュー・マネージャーに対して実行できるようにしたい場合には、予期しないメッセージまたは送達できないメッセージは、そのキュー・マネージャーよって、システムの送達不能キューに入れることができます。

サンプルが同期点を使用する方法

信用小切手サンプル・アプリケーション内のプログラムは、同期点を宣言して、以下のことを可能にします。

- 予期される各メッセージに対する応答として、1 つの応答メッセージのみが送信されます。
- 予期しないメッセージの複数のコピーが、サンプルの送達不能キューに入れられないようにします。
- CAM が、その待機キューから持続的有効性のあるメッセージを読み取ることによって、部分的に完了された照会すべての状態を回復できるようにします。

このためには、単一の作業単位を使用して、メッセージの読み取り、そのメッセージの処理、および後続の書き込み操作をカバーします。

サンプルがメッセージ・コンテキスト情報を使用する方法

ユーザー・インターフェース・プログラム (CSQ4CVB1) は、メッセージを送信するときに、MQPMO_DEFAULT_CONTEXT オプションを使用します。 このことは、キュー・マネージャーが識別コンテキスト情報および起点コンテキスト情報の両方を生成することを意味します。 キュー・マネージャーは、この情報をプログラム (MVB1) を開始したトランザクションおよびトランザクションを開始したユーザー ID から取得します。

CAM は、照会メッセージを送信するときに、MQPMO_PASS_IDENTITY_CONTEXT オプションを使用します。 このことは、キューに入れられるメッセージの識別コンテキスト情報は、元の照会メッセージの識別コンテキストからコピーされることを意味します。 このオプションを使用することにより、起点コンテキスト情報がキュー・マネージャーによって生成されます。

CAM は、応答メッセージを送信するときに、MQPMO_ALTERNATE_USER_AUTHORITY オプションを使用します。 これによって、キュー・マネージャーは、CAM が応答先キューをオープンするときのセキュリティー検査のために、代わりのユーザー ID を使用します。 CAM は、元の照会メッセージの実行依頼元のユーザー ID を使用します。 このことは、ユーザーは、自分自身が発信した照会に対する応答のみを参照することが許可されていることを意味します。 代わりのユーザー ID は、元の照会メッセージのメッセージ記述子内の識別コンテキスト情報から得られます。

照会プログラム (CSQ4CVB3/4/5) は、応答メッセージを送信するときに、MQPMO_PASS_IDENTITY_CONTEXT オプションを使用します。 このことは、キューに入れられるメッセージの識別コンテキスト情報は、元の照会メッセージの識別コンテキストからコピーされることを意味します。 このオプションを使用することにより、起点コンテキスト情報がキュー・マネージャーによって生成されます。

注 : MVB3/4/5 トランザクションに関係するユーザー ID は、B2.REPLY.n キューへのアクセスが必要となります。 これらのユーザー ID は、処理中の要求に関連付けられたユーザー ID とは異なる場合があります。 このセキュリティー上の危険を回避するために、照会プログラムは、応答をキューに入れるときに、MQPMO_ALTERNATE_USER_AUTHORITY オプションを使用することができます。 これは、個々の MVB1 ユーザーは、B2.REPLY.n キューをオープンするための許可が必要となることを意味します。

CAM でのメッセージ ID および相関 ID の使用

アプリケーションは、同時に処理している活動状態のすべての照会の進捗状況をモニターする必要があります。 これを行うために、アプリケーションは各貸付要求メッセージに固有のメッセージ ID を使用して、それぞれの照会に関してアプリケーションにあるすべての情報を関連付けます。

CAM は、照会メッセージの MsgId を、その照会のために送信するすべての要求メッセージの CorrelId にコピーします。 サンプル内のその他のプログラム (CSQ4CVB3 から 5) は、受信する各メッセージの CorrelId を、その応答メッセージの CorrelId にコピーします。

z/OS z/OS において複数のキュー・マネージャーを使用する信用小切手サンプル

信用小切手サンプル・アプリケーションを使用して、2 つのキュー・マネージャーと CICS システム (それぞれのキュー・マネージャーが異なる CICS システムに接続されている) にサンプルをインストールすることにより、分散キューイングを実証することができます。

サンプル・プログラムがインストールされ、トリガー・モニター (CKTI) が各システムで実行されているときに、以下のことを行う必要があります。

1. 2 つのキュー・マネージャー間に通信リンクを設定する。 この方法については、分散キューイングの構成を参照してください。
2. 一方のキュー・マネージャーで、(もう一方のキュー・マネージャー上にある) 使用したい各リモート・キューのローカル定義を作成する。 これらのキューを任意の CSQ4SAMP.B n.MESSAGES とすることができます。ここで、*n* は 3、5、6、または 7 です。 (これらは、チェックリスト・アカウント・プログラムと代理店照会プログラムによって提供されるキューです。) これを行う方法については、 DEFINE QREMOTE および DEFINE キューを参照してください。
3. 使用したいリモート・キューの名前を含められるように、名前リスト (CSQ4SAMP.B4.NAMELIST) の定義を変更する。 この方法については、DEFINE NAMELIST を参照してください。

z/OS z/OS での信用小切手サンプルに対する *IMS* 拡張

当座預金口座プログラムは、IMS バッチ・メッセージ処理 (BMP) プログラムとして提供されます。 このプログラムは、C 言語で書かれています。

プログラムは、 CICS バージョンと同じ機能を実行します。ただし、アカウント情報を取得するために、プログラムは VSAM ファイルではなく IMS データベースを読み取ります。 CICS バージョンの当座預金口座プログラムを IMS バージョンに置換しても、アプリケーションの使用方法は変わりません。

IMS バージョンを作成し、実行するには、次のことを行う必要があります。

1. 1201 ページの『z/OS における信用小切手サンプルの作成と実行』のステップを実行します。
2. 1182 ページの『z/OS での IMS 環境用のサンプル・アプリケーションの準備』のステップを実行します。
3. 別名キュー CSQ4SAMP.B2.OUTPUT.ALIAS の定義を変更して、(キュー CSQ4SAMP.B3.MESSAGES の代わりに) キュー CSQ4SAMP.B3.IMS.MESSAGES に解決されるようにする。 これを行う場合、次のいずれかを使用できます。
   - IBM MQ for z/OS 操作および制御パネル
   - ALTER QALIAS コマンド

IMS 当座預金口座プログラムを使用する別の方法は、配布プログラムからメッセージを受信するキューの 1 つをこのプログラムが使用する方法です。 信用小切手サンプル・アプリケーションの配布形式には、これらのキューのうちの 3 つ (B5/6/7.MESSAGES) があり、これら 3 つのすべてのキューは代理店照会プログラムが使用します。 このプログラムは、VSAM データ・セット内を検索します。 VSAM データ・セットと IMS データベースの使用方法を比較するために、IMS 当座預金口座プログラムがこれらのキューのいずれかを処理するようにできます。 これを行うには、名前リスト CSQ4SAMP.B4.NAMELIST の定義を変更して、CSQ4SAMP.B*n*.MESSAGES キューの 1 つを CSQ4SAMP.B3.IMS.MESSAGES キューに置換する必要があります。 次のいずれかを使用できます。

- IBM MQ for z/OS 操作および制御パネル
- ALTER NAMELIST コマンド

この時点で、CICS トランザクション MVB1 からサンプルを実行できます。 操作や応答は変わりません。 IMS BMP は、停止メッセージを受信するか、5 分間非活動状態のままであると停止します。

IMS 当座預金口座プログラム (CSQ4ICB3) の設計

このプログラムは BMP として実行されます。 いずれかの IBM MQ メッセージが送信される前に、JCL を使用してこのプログラムを開始してください。

このプログラムは、IMS データベースの中から貸付要求メッセージの口座番号を検索します。 このプログラムは、対応する口座名、平均残高、および信用評価指数を取り出します。

このプログラムは、データベースの検索結果を、処理中の IBM MQ メッセージで指定された応答先キューに送信します。 応答を作成するトランザクションが正しいキューを処理中であるか確認できるように、戻されたメッセージは、口座の種類と検索の結果を受信したメッセージに付加します。 このメッセージは 3 行 (1 行の長さは 79 文字) 分のスペースに表示されます。以下に例を示します。

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

メッセージ指向 BMP として実行する場合、プログラムは IMS メッセージ・キューをドレーンしてから、IBM MQ for z/OS キューからメッセージを読み取り、それらを処理します。 IMS メッセージ・キューから受信する情報はありません。 ハンドルがクローズされたため、このプログラムは、それぞれのチェックポイント後にキュー・マネージャーに再接続します。

バッチ指向の BMP で実行する場合、ハンドルがクローズされていないため、このプログラムは、それぞれのチェックポイント後もキュー・マネージャーに接続されたままとなります。

z/OS *z/OS におけるメッセージ・ハンドラー・サンプル*

TSO 用メッセージ・ハンドラー・サンプル・アプリケーションによって、キュー上のメッセージを表示、転送、および削除できます。 このサンプルは C および COBOL で利用できます。

サンプルの作成と実行

次のステップを行います。

1. 1176 ページの『z/OS における TSO 環境用サンプル・アプリケーションの作成』 で説明するように、サンプルを作成する。
2. このサンプル用に提供されている CLIST (CSQ4RCH1) を調整して、パネルの位置、メッセージ・ファイルの位置、およびロード・モジュールの位置を定義する。

CLIST CSQ4RCH1 を使用して、C および COBOL の両方のバージョンのサンプルを実行できます。 CSQ4RCH1 の提供されているバージョンは、C バージョンを実行し、COBOL バージョンを実行するのに必要な調整に関する指示を含んでいます。

注 :

1. このサンプルでは、サンプル・キュー定義は提供されていません。
2. VS COBOL II では ISPF でのマルチタスキングをサポートしていません。したがって、分割画面の両側でメッセージ・ハンドラー・サンプル・アプリケーションを使用しないでください。 これを行うと、その結果は予測できません。

z/OS *z/OS におけるメッセージ・ハンドラー・サンプルの使用*

サンプルをインストールし、調整済みの CLIST CSQ4RCH1 から呼び出すと、1214 ページの図 146 に示す画面が表示されます。

```
---------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :


Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

図 *146.* メッセージ・ハンドラー・サンプルの初期画面

表示するキュー・マネージャー名およびキュー名 (大文字小文字の区別あり) を入力してください。これにより、メッセージ・リスト画面が表示されます (1214 ページの図 147 参照)。

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
******************************* Bottom of data ********************************
```

図 *147.* メッセージ・ハンドラー・サンプル用のメッセージ・リスト画面

この画面にはキューにあるメッセージが先頭から 99 個表示され、それぞれについて次のフィールドが表示されます。

Msg No
メッセージ番号

Put Date MM/DD/YYYY
メッセージがキューに書き込まれた日付 (グリニッジ標準時)

Put Time HH:MM:SS
メッセージがキューに書き込まれた時刻 (グリニッジ標準時)

Format Name
MQMD.Format フィールド

ユーザー ID
MQMD.UserIdentifier フィールド

Put Application Type
MQMD.PutApplType フィールド

Put Application Name
MQMD.PutApplName フィールド

キューに入っているメッセージの総数も表示されます。

この画面からは、メッセージを、カーソル位置ではなく番号によって選択し、表示できます。 例については、1215 ページの図 148 を参照してください。

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
--------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence   : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount   : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr   : `VM03                          `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                            `
PutApplType   : 000000002
PutApplName   : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .............`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
******************************** Bottom of data ********************************
```

図 *148.* 選択したメッセージの表示結果

メッセージが表示されれば、削除したり、キューに残しておいたり、あるいは別のキューに転送したりできます。 「Forward to Q Mgr」フィールドおよび「Forward to Queue」フィールドは、MQMD からの値を使用して初期設定します。これらのフィールドはメッセージを転送する前に変更できます。

メッセージは MsgId と CorrelId をキーとして使用して取得されるため、サンプル設計では、固有の MsgId / CorrelId の組み合わせを持つメッセージのみを選択して表示することができます。 キーが固有でない場合は、サンプルは選択されたメッセージを確実に検索することができなくなります。

注 : メッセージをブラウズするために SCSQCLST(CSQ4RCH1) サンプルを使用すると、呼び出しのたびにメッセージのバックアウト・カウントが増加します。 このサンプルの動作を変更するには、サンプルをコピーし、必要に応じて内容を変更します。 このバックアウト・カウントに依存する他のアプリケーションが、このカウント増加の影響を受ける場合があることにご注意ください。

z/OS *z/OS におけるメッセージ・ハンドラー・サンプルの設計*
このトピックでは、メッセージ・ハンドラー・サンプル・アプリケーションを構成している各プログラムの設計について説明します。

オブジェクト妥当性検査プログラム

ここでは、有効なキュー名とキュー・マネージャー名が必要となります。

キュー・マネージャー名を指定しないと、デフォルトのキュー・マネージャーが使用されます (使用可能な場合)。 ローカル・キューのみが使用できます。キューのタイプを検査するために MQINQ が発行され、そのキューがローカルでない場合、エラーが報告されます。 キューを正常にオープンできない場合、あるいは MQGET 呼び出しがキューで禁止されている場合、エラー・メッセージが戻り、このエラー・メッセージには CompCode および Reason を示す戻りコードが入っています。

メッセージ・リスト・プログラム

このプログラムは、キューに入っているメッセージのリストを書き込み日、書き込み時刻、メッセージ形式などのメッセージに関する情報と共に表示します。

リストに格納される最大メッセージ数は 99 個です。 この最大数を超える数のメッセージがキューにある場合は、現行キューのサイズも表示されます。 表示するメッセージを選択する場合は、入力フィールドにそのメッセージ番号を入力します (デフォルト値は 01 です)。 入力が誤っていると、該当するエラー・メッセージを受け取ります。

メッセージ内容プログラム

このプログラムは、メッセージの内容を表示します。

その内容は形式設定され、次の 2 つの部分に分かれています。

1. メッセージ記述子
2. メッセージ・バッファー

メッセージ記述子の部分には、各フィールドの内容がそれぞれ別の行に表示されます。

メッセージ・バッファーの部分は、その内容によって形式設定されます。 バッファーが送達不能ヘッダー (MQDLH) または伝送キュー・ヘッダー (MQXQH) を保持している場合は、それらのヘッダーは形式設定され、バッファーの前に表示されます。

バッファー・データの形式設定を行う前に、タイトル行でメッセージのバッファー長 (バイト単位) を表示します。 最大のバッファー・サイズは 32768 バイトで、これを超えるメッセージの部分は切り捨てられます。 メッセージの最初の 32768 バイトのみが表示されることを示すメッセージと共にバッファー全体のサイズが表示されます。

バッファー・データの形式は 2 つの方法で設定されます。

1. バッファー先頭からのオフセットで出力したあとに、バッファー・データを 16 進数で表示する。
2. バッファー・データを再び EBCDIC 値として表示する。 EBCDIC 値を出力できない場合は、代わりにピリオド (.) が出力されます。

削除する場合には D と入力し、アクション・フィールドに転送する場合は F と入力します。 メッセージを転送する場合は、`forward-to queue` と `queue manager name` を正しく設定する必要があります。 これらのフィールドのデフォルト値は、メッセージ記述子の ReplyToQ および ReplyToQMgr フィールドから読み込まれます。

メッセージを転送すると、バッファーに格納されているヘッダー・ブロックはすべて削除されます。 メッセージは、正常に転送されると、元のキューから除去されます。 無効なアクションを実行しようすると、エラー・メッセージが表示されます。

CSQ4CHP9 というヘルプ・パネルの例も提供されています。

z/OS z/OS における非同期書き込みサンプル

非同期書き込みサンプル・プログラムは、非同期 MQPUT 呼び出しを使用してキューにメッセージを書き込みます。 サンプルでは、MQSTAT 呼び出しを使用して状況情報も取得します。

非同期書き込みアプリケーションでは、以下の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

サンプル・プログラムは C プログラミング言語で提供されます。

非同期書き込みアプリケーションはバッチ環境で実行されます。 バッチ・アプリケーションについては、その他のサンプルを参照してください。

このトピックでは、非同期コンシューム・プログラムの設計および CSQ4BCS2 サンプルの実行についても説明します。

- 1217 ページの『CSQ4BCS2 サンプルの実行』
- 1217 ページの『非同期書き込みサンプル・プログラムの設計』

CSQ4BCS2 サンプルの実行

このサンプル・プログラムでは、パラメーターを以下の 6 つまで設定できます。

1. 宛先キューの名前 (必須)。
2. キュー・マネージャーの名前 (オプション)。
3. オープン・オプション (オプション)。
4. クローズ・オプション (オプション)。
5. 宛先キュー・マネージャーの名前 (オプション)。
6. 動的キューの名前 (オプション)。

キュー・マネージャーを指定しないと、CSQ4BCS2 はデフォルトのキュー・マネージャーに接続されます。メッセージ内容は標準入力 (**SYSIN DD**) で提供されます。

プログラムを実行するサンプル JCL が CSQ4BCSP にあります。

非同期書き込みサンプル・プログラムの設計

プログラムは、提供される出力オプション付き、または MQOO_OUTPUT および MQOO_FAIL_IF_QUIESCING オプション付きで MQOPEN 呼び出しを使用して、メッセージを書き込む宛先キューをオープンします。

キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。 この呼び出しおよび後続の MQI 呼び出しに関してプログラムを簡潔に保つため、多数のオプションに対してデフォルト値を使用します。

入力行ごとに、プログラムはテキストをバッファーに読み込み、MQPMO_ASYNC_RESPONSE 付きの MQPUT 呼び出しを使用して、その行のテキストが入ったデータグラム・メッセージを作成し、非同期で宛先キューにメッセージを書き込みます。 プログラムは、入力が終了するか、MQPUT 呼び出しが失敗するまで処理を続行します。 プログラムは、入力が終了すると、MQCLOSE 呼び出しを使用して、キューをクローズします。

続いてプログラムは、MQSTS 構造体が戻される MQSTAT 呼び出しを発行し、正常に書き込まれたメッセージの数、警告が出されて書き込まれたメッセージの数、および失敗の回数が含まれたメッセージを表示します。

注 : MQPUT エラーが MQSTAT 呼び出しによって検出されるときに何が発生しているかを監視するには、宛先キューの MAXDEPTH を低い値に設定します。

z/OS *z/OS* におけるバッチ非同期コンシューム・サンプル

CSQ4BCS1 サンプル・プログラムは、C で提供され、複数のキューからのメッセージを非同期的にコンシュームするために MQCB および MQCTL を使用する方法を示します。

非同期コンシューム・サンプルはバッチ環境で実行されます。 バッチ・アプリケーションについては、その他のサンプルを参照してください。

CICS 環境で実行される COBOL サンプルもあります。1219 ページの『z/OS における CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル』 を参照してください。

これらのアプリケーションでは、以下の MQI 呼び出しを使用します。

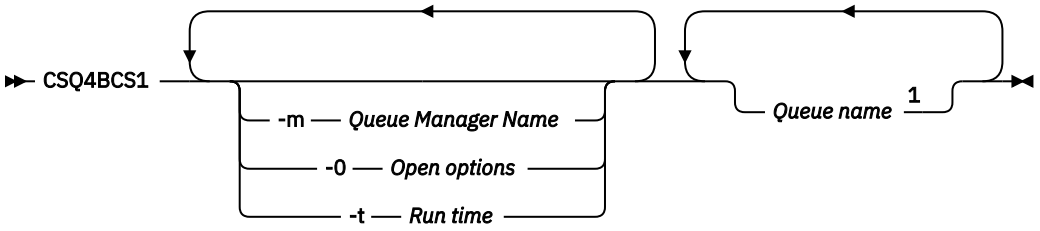
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

このトピックでは、以下の見出しの情報についても説明します。

- 1218 ページの『CSQ4BCS1 サンプルの実行』
- 1218 ページの『 バッチ非同期コンシューム・サンプル・プログラムの設計』

CSQ4BCS1 サンプルの実行

このサンプル・プログラムは、以下の構文に従っています。



注:

[1] このサンプルでは、最大 10 個のキューがサポートされます。 複数のキューからメッセージを読み取るためには、複数のキュー名を指定します。

このプログラムを実行するサンプル JCL が CSQ4BCSC にあります。

バッチ非同期コンシューム・サンプル・プログラムの設計

このサンプルは、複数のキューからのメッセージを到着順に読み取る方法 を示します。 この方法では、同期 MQGET を使用して、さらに多くのコードが必要となる場合があります。 非同期コンシュームの場合、ポーリングは不要であり、スレッドおよびストレージの管理は IBM MQ によって実行されます。 サンプル・プログラムでは、エラーはコンソールに書き込まれます。

サンプル・コードには以下のステップがあります。

1. 単一のメッセージ・コンシューム・コールバック関数を定義する。

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
```

```
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. キュー・マネージャーに接続する。

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. 入力キューを開き、各入力キューを MessageConsumer コールバック関数と関連付ける。

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   cbd.CallbackFunction は、各キューごとに設定する必要はありません。これは入力専用フィールドです。 キューごとに異なるコールバック関数を関連付けることができます。

4. メッセージのコンシュームを開始する。

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. ユーザーが Enter キーを押すまで待ってからメッセージのコンシュームを停止する。

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最後に、キュー・マネージャーから切断する。

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS z/OS における CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル

非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル・プログラムでは、CICS 内での非同期コンシュームの使用法と、パブリッシュ機能およびサブスクライブ機能の使用法を示します。

登録クライアント・プログラムでは、3 つのコールバック・ハンドラー (1 つのイベント・ハンドラーと 2 つのメッセージ・コンシューマー) を登録し、非同期コンシュームを開始します。 メッセージング・クライアント・プログラムでは、キューにメッセージを書き込んだり、2 つのメッセージ・コンシューマー (CSQ4CVCN および CSQ4CVCT) によるコンシューム用に CICS コンソールから適切なメッセージをパブリッシュしたりします。

サンプルの動作を実行時に制御するため、メッセージ・コンシューマーの 1 つに、受信するメッセージを使用して、いずれかのコールバック・ハンドラーを中断、再開、または登録解除するように指示できます。また、MQCTL STOP を発行して、制御下にある非同期コンシュームを終了させることもできます。 もう 1 つのメッセージ・コンシューマーは、トピックをサブスクライブするように登録されます。

各プログラムでは、サンプルの動作を表示するために適切なポイントで COBOL DISPLAY ステートメントが発行されます。

これらのアプリケーションでは、以下の MQI 呼び出しを使用します。

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

これらのプログラムは COBOL 言語で提供されます。 CICS アプリケーションについては、 CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル を参照してください。

また、このトピックでは次の情報も含まれます。

- 1220 ページの『セットアップ』
- 1220 ページの『登録クライアント CSQ4CVRG』
- 1220 ページの『イベント・ハンドラー CSQ4CVEV』
- 1220 ページの『単純メッセージ・コンシューマー CSQ4CVCN』
- 1220 ページの『制御メッセージ・コンシューマー CSQ4CVCT』
- 1221 ページの『メッセージング・クライアント CSQ4CVPT』

セットアップ

メッセージ・コンシューマーによって使用されるキュー名およびトピック名は、登録クライアント・プログラムおよびメッセージング・クライアント・プログラムにハードコーディングされています。

キュー **SAMPLE.CONTROL.QUEUE** は、サンプルの実行前に、CICS 領域に関連付けられているキュー・マネージャーに定義する必要があります。 トピック **News/Media/Movies** は、必要に応じて定義できますが、存在しない場合は、実行時にデフォルトの管理オブジェクト下に作成されます。

グループ CSQ4SAMP のインストールによって、CICS プログラムおよびトランザクション定義をインストールすることができます。

登録クライアント CSQ4CVRG

登録クライアント・プログラムは、CICS トランザクション MVRG 下で開始する必要があります。 入力は取りません。

登録クライアントは、開始されると、MQCB を使用して以下のコールバック・ハンドラーを登録します。

- CSQ4CVEV (イベント・ハンドラー)
- CSQ4CVCN (トピック **News/Media/Movies** のメッセージ・コンシューマー)
- CSQ4CVCT (キュー **SAMPLE.CONTROL.QUEUE** のメッセージ・コンシューマー)

登録クライアントは、登録される 3 つすべてのコールバック・ハンドラーの名前を含むデータ構造体、および 2 つのメッセージ・コンシューマーに関連付けられるオブジェクト・ハンドルを CSQ4CVCT に渡します。

登録クライアントはコールバック・ハンドラーを登録すると、MQCTL START_WAIT を発行して非同期コンシュームを開始し、制御が戻る (例えば、コールバック・ハンドラーの 1 つが MQCTL STOP を発行する) まで中断します。

イベント・ハンドラー CSQ4CVEV

イベント・ハンドラーは、呼び出されると、呼び出しタイプ (START など) を示すメッセージを表示します。 IBM MQ 理由コード CONNECTION_QUIESCING のために呼び出されると、イベント・ハンドラーは MQCTL STOP を発行し、非同期コンシュームを終了して制御を登録クライアントに返します。

単純メッセージ・コンシューマー CSQ4CVCN

このメッセージ・コンシューマーは、呼び出されると、呼び出しタイプ (REGISTER など) を示すメッセージを表示します。 MSG_REMOVED 呼び出しタイプの場合、メッセージ・コンシューマーはインバウンド・メッセージを取得し、それを CICS ジョブ・ログに出力します。

制御メッセージ・コンシューマー CSQ4CVCT

このメッセージ・コンシューマーは、呼び出されると、呼び出しタイプ (START など) を示すメッセージを表示します。 MSG_REMOVED 呼び出しタイプのために呼び出されると、このメッセージ・コンシューマー

はインバウンド・メッセージ、および登録クライアントにより渡されたデータ構造体を取得します。 メッセージの内容に基づいて、以下のいずれかに対する該当の MQCB コマンドまたは MQCTL コマンドを発行します。

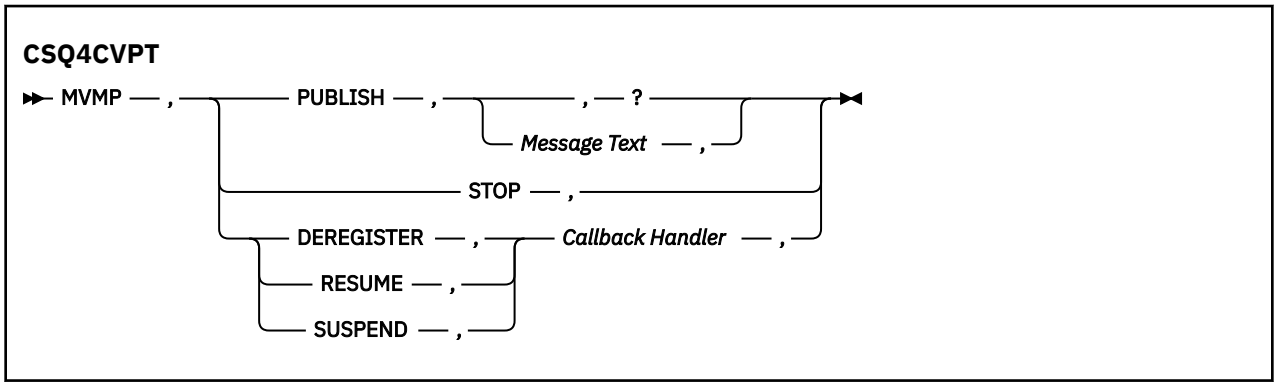
- 非同期コンシュームの停止 (登録クライアントに制御を返す)
- 指定されたコールバック・ハンドラー (これ自体も含む) の中断、再開、または登録解除

メッセージング・クライアント CSQ4CVPT

メッセージング・クライアントには以下の 2 つの機能があります。

- メッセージ・コンシューマー CSQ4CVCN によるコンシュームのトピックにメッセージをパブリッシュします。
- 制御メッセージ・コンシューマー CSQ4CVCT によるコンシュームのキューに制御メッセージを書き込みます。これによりサンプルの動作を変更できます。

メッセージング・クライアント・プログラムは、CICS トランザクション下で CICS コンソールから開始する必要があり、以下の構文のコマンド行入力を取ります。



PUBLISH
単純メッセージ・コンシューマーによるコンシューム用の保存メッセージとしてメッセージ・テキスト (またはデフォルト・メッセージ) をパブリッシュします。

STOP
非同期コンシュームを停止します。

DEREGISTER
コールバック・ハンドラーを登録解除します。

RESUME
指定したコールバック・ハンドラーを再開します。

SUSPEND
指定したコールバック・ハンドラーを中断します。

入力フィールドは定位置のコンマ区切りです。 キーワードおよびコールバック・ハンドラー名には大/小文字の区別はありません。

例：

表 *186.* 入力例

例	**説明**
MVMP,PUBLISH,,	デフォルト・メッセージのパブリッシュ
MVMP,publish,A short message,	指定したテキストのパブリッシュ
MVMP,STOP,	非同期コンシュームの停止
MVMP,DEREGISTER,CSQ4CVEV,	イベント・ハンドラーの登録解除
MVMP,resume,csq4cvcn,	単純メッセージ・コンシューマーの再開

表 186. 入力例 (続き)	
例	**説明**
MVMP,SUSPEND,CSQ4CVEV,	イベント・ハンドラーの中断

ここで、MVMP はメッセージング・クライアント・プログラム CSQ4CVPT に関連付けられた CICS トランザクションです。

注 :

- すべてのコールバック・ハンドラーを中断または登録解除すると、登録クライアントにより発行された START_WAIT が終了し、制御が登録クライアントに戻されてタスクは終了します。
- 制御コールバック・ハンドラーの中断または登録解除は、意図的に防止されてはいませんが、これをすると、サンプルの動作をそれ以降制御することができなくなります。

z/OS z/OS におけるパブリッシュ/サブスクライブのサンプル

パブリッシュ/サブスクライブのサンプル・プログラムでは、IBM MQ のパブリッシュ機能およびサブスクライブ機能の使用法を示します。

4 つの C プログラミング言語および 2 つの COBOL プログラミング言語のサンプル・プログラムがあり、これらは IBM MQ パブリッシュ/サブスクライブ・インターフェースのプログラム作成方法を示します。 これらのプログラムは C 言語および COBOL 言語で提供されます。 アプリケーションはバッチ環境で実行されます。バッチ・アプリケーションについては、パブリッシュ/サブスクライブのサンプルを参照してください。

CICS 環境で実行される COBOL サンプルもあります。1219 ページの『z/OS における CICS 非同期コンシュームおよびパブリッシュ/サブスクライブのサンプル』 を参照してください。

このトピックでは、パブリッシュ/サブスクライブのサンプル・プログラムの実行方法についても説明します。 サンプル・プログラムには次のようなものがあります。

- 1222 ページの『CSQ4BCP1 サンプルの実行』
- 1222 ページの『CSQ4BCP2 サンプルの実行』
- 1223 ページの『CSQ4BCP3 サンプルの実行』
- 1223 ページの『CSQ4BCP4 サンプルの実行』
- 1224 ページの『CSQ4BVP1 サンプルの実行』
- 1224 ページの『CSQ4BVP2 サンプルの実行』

CSQ4BCP1 サンプルの実行

このプログラムは C で作成されており、トピックに対してメッセージをパブリッシュします。 このプログラムを実行する前に、サブスクライバー・サンプルの 1 つを開始します。

このプログラムでは、パラメーターを以下の 4 つまで設定できます。

1. 宛先トピック・ストリングの名前 (必須)。
2. キュー・マネージャーの名前 (オプション)。
3. オープン・オプション (オプション)。
4. クローズ・オプション (オプション)。

キュー・マネージャーを指定しないと、CSQ4BCP1 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BCPP にあります。

メッセージ内容は標準入力 (**SYSIN DD**) で提供されます。

CSQ4BCP2 サンプルの実行

このプログラムは C で作成されており、トピックに対してサブスクライブし、受け取ったメッセージを出力します。

このプログラムでは、パラメーターを以下の 3 つまで設定できます。

1. 宛先トピック・ストリングの名前 (必須)。
2. キュー・マネージャーの名前 (オプション)。
3. MQSD サブスクリプション・オプション (オプション)。

キュー・マネージャーを指定しないと、CSQ4BCP2 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BCPS にあります。

CSQ4BCP3 サンプルの実行

このプログラムは C で作成されており、ユーザー指定の宛先キューを使用してトピックに対してサブスクライブし、受け取ったメッセージを出力します。

このプログラムでは、パラメーターを以下の 4 つまで設定できます。

1. 宛先トピック・ストリングの名前 (必須)。
2. 宛先の名前 (必須)。
3. キュー・マネージャーの名前 (オプション)。
4. MQSD サブスクリプション・オプション (オプション)。

キュー・マネージャーを指定しないと、CSQ4BCP3 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BCPD にあります。

CSQ4BCP4 サンプルの実行

このプログラムは C で作成されており、MQSUB 呼び出しの拡張オプションの使用を許可した状態でトピックにサブスクライブしてメッセージを取得します。単純な MQSUB サンプル CSQ4BCP2 の内容を拡張したものです。 メッセージ・ペイロードだけでなく、メッセージごとのメッセージ・プロパティーも受け取られて表示されます。

このプログラムでは、以下のようにパラメーターの変数セットを設定できます。

- **-t** *Topic string*.
- **-o** *Topic object name*.

  重要 : -t または **-o** のいずれか、あるいはその両方が必要です。
- **-m** *Queue manager name* (オプション)。
- **-b** *Connection binding type* (オプション)。ここで、 *type* には以下のいずれかの値を指定できます。
  - *standard* : MQCNO_STANDARD_BINDING。これがデフォルト値です。
  - 共有: MQCNO_SHARED_BINDING
  - *fastpath*: MQCNO_FASTPATH_BINDING
  - 分離: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (オプション)。
- **-w** *Wait interval on MQGET in seconds* (オプション)。 *seconds* には、以下のいずれかの値を指定できます。
  - unlimited: MQWI_UNLIMITED
  - none: 待機なし
  - *n*: 秒単位での待機間隔
  - 値を指定しない: 値を指定しないとデフォルトの 30 秒になります。
- **-d** *Subscription name* (オプション)。 名前付きの永続サブスクリプションを作成または再開します。
- **-k** (オプション)。 MQCLOSE の永続サブスクリプションを保持します。

キュー・マネージャーを指定しないと、CSQ4BCP4 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BCPE にあります。

CSQ4BVP1 サンプルの実行

このプログラムは COBOL で作成されており、トピックに対してメッセージをパブリッシュします。 このプログラムを実行する前に、サブスクライバー・サンプルの 1 つを開始します。

このプログラムには、パラメーターはありません。 **SYSIN DD** で、入力トピック名、キュー・マネージャー名、およびメッセージ内容が提供されます。

キュー・マネージャーを指定しないと、CSQ4BVP1 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BVPP にあります。

CSQ4BVP2 サンプルの実行

このプログラムは COBOL で作成されており、トピックに対してサブスクライブし、受け取ったメッセージを表示します。

このプログラムには、パラメーターはありません。 **SYSIN DD** で、トピック名およびキュー・マネージャー名の入力が提供されます。

キュー・マネージャーを指定しないと、CSQ4BVP1 はデフォルトのキュー・マネージャーに接続されます。プログラムを実行するサンプル JCL が CSQ4BVPP にあります。

z/OS *z/OS* におけるメッセージ・プロパティーの設定および照会のサンプル

メッセージ・プロパティー・サンプル・プログラムでは、メッセージ・ハンドルへのユーザー定義プロパティーの追加、およびそのメッセージに関連付けられるプロパティーの照会について示されます。

これらのアプリケーションでは、以下の MQI 呼び出しを使用します。

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

これらのプログラムは C 言語で提供されます。 アプリケーションはバッチ環境で実行されます。 バッチ・アプリケーションについては、その他のサンプルを参照してください。

CSQ4BCM1 プログラムは、メッセージ・ハンドルのプロパティーをメッセージ・キューから照会するために使用され、MQINQMP API 呼び出しの使用例になっています。 サンプルでは、キューから 1 つのメッセージが読み取られた後、すべてのメッセージ・ハンドル・プロパティーがプリントされます。

CSQ4BCM2 プログラムは、メッセージ・ハンドルのプロパティーをメッセージ・キューに設定するために使用され、MQSETMP API 呼び出しの使用例になっています。 サンプルでは、メッセージ・ハンドルが作成され、それが MQGMO 構造体の `MsgHandle` フィールドに設定されます。 その後、メッセージがキューに書き込まれます。

メッセージ・プロパティーの照会およびプリントのその他の例が、CSQ4BCG1 サンプル・プログラムおよび CSQ4BCP4 サンプル・プログラムに含まれます。

このトピックの以下の見出しでは、メッセージ・プロパティーの設定および照会サンプルの実行についても説明します。

- 1225 ページの『CSQ4BCM1 サンプルの実行』
- 1225 ページの『CSQ4BCM2 サンプルの実行』

CSQ4BCM1 サンプルの実行

このプログラムでは、パラメーターを以下の 4 つまで設定できます。

1. 宛先キューの名前 (必須)。
2. キュー・マネージャーの名前 (オプション)。
3. オープン・オプション (オプション)。
4. クローズ・オプション (オプション)。

CSQ4BCM2 サンプルの実行

このプログラムでは、パラメーターを以下の 6 つまで設定できます。

1. 宛先キューの名前 (必須)。
2. キュー・マネージャーの名前 (オプション)。
3. オープン・オプション (オプション)。
4. クローズ・オプション (オプション)。
5. 宛先キュー・マネージャーの名前 (オプション)。
6. 動的キューの名前 (オプション)。

プロパティー名、値、およびメッセージ内容は、標準入力 (**SYSIN DD**) を介して提供されます。 プログラムを実行するためのサンプル JCL があります。これは CSQ4BCMP にあります。

Managed File Transfer 用アプリケーションの開発

Managed File Transfer で実行するプログラムを指定し、 Managed File Transfer で Apache Ant を使用し、ユーザー出口で Managed File Transfer をカスタマイズし、エージェント・コマンド・キューにメッセージを書き込むことによって Managed File Transfer を制御します。

MFT で実行するプログラムの指定

Managed File Transfer Agent が実行されているシステムで、プログラムを実行することができます。 ファイル転送要求の一部として、転送の開始前または終了後のいずれかにプログラムを実行するように指定することができます。 また、管理対象呼び出し要求を実行依頼することで、ファイル転送要求に含まれないプログラムを開始することも可能です。

このタスクについて

以下の 5 つのシナリオで、プログラムの実行を指定することができます。

- 転送要求の一部として、転送の開始前に、ソース・エージェントで転送を開始します。
- 転送要求の一部として、転送の開始前に宛先エージェントで転送を開始します。
- 転送要求の一部として、転送の完了後にソース・エージェントで。
- 転送要求の一部として、転送の完了後に宛先エージェントで。
- 転送要求の一部としてではなく実行する。 プログラム実行の要求を、エージェントに実行依頼することができます。 このシナリオは、管理対象呼び出しと呼ばれる場合があります。

ユーザー出口とプログラム呼び出しは、以下の順序で呼び出されます。

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
```

```
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

注 :

1. **DestinationTransferEndExits** は、転送が正常に完了したか、部分的に正常に完了した場合にのみ実行されます。
2. **postDestinationCall** は、転送が正常に完了したか、部分的に正常に完了した場合にのみ実行されます。
3. **SourceTransferEndExits** は、成功した転送、部分的に成功した転送、または失敗した転送に対して実行されます。
4. **postSourceCall** は、以下の場合にのみ呼び出されます。
   - 転送は取り消されませんでした。
   - 成功した結果または部分的に成功した結果があります。
   - 宛先転送後プログラムは正常に実行されました。

手順

- 以下のいずれかのオプションを使用して、実行するプログラムを指定します。

  Apache Ant タスクを使用する

  `fte:filecopy`、`fte:filemove`、および `fte:call` Ant のいずれかのタスクを使用して、プログラムを開始します。 Ant タスクを使用すると、 `fte:presrc`、 `fte:predst`、 `fte:postdst`、`fte:postsrc`、および `fte:command` のネストされたエレメントを使用して、5 つのシナリオのいずれでもプログラムを指定できます。 詳しくは、プログラム呼び出しのネスト・エレメントを参照してください。

  ファイル転送要求メッセージを編集する

  転送要求によって生成された XML を編集することができます。 この方法を使用すると、**preSourceCall**、**postSourceCall**、**preDestinationCall**、**postDestinationCall**、および **managedCall** エレメントを XML ファイルに追加することにより、5 つのシナリオのいずれでもプログラムを実行できます。 その後、例えば **fteCreateTransfer -td** パラメーターを付けるなどして、この変更した XML ファイルを新規ファイル転送要求の転送定義として使用します。 詳細は、「MFT エージェント呼び出し要求メッセージの例」を参照してください。

  fteCreateTransfer コマンドを使用する

  fteCreateTransfer コマンドを使用して、開始するプログラムを指定することができます。 このコマンドを使用して、最初の 4 つのシナリオで転送要求の一部としてプログラムを実行するように指定することはできますが、管理対象呼び出しを開始することはできません。 使用するパラメーターについては、**fteCreateTransfer**: 新規ファイル転送の開始を参照してください。 このコマンドの使用例については、fteCreateTransfer を使用してプログラムを開始する例を参照してください。

関連資料

commandPath MFT プロパティー

管理対象呼び出し

Managed File Transfer (MFT) エージェントは、通常、ファイルまたはメッセージの転送に使用されます。 これらは 管理対象転送と呼ばれます。 エージェントは、ファイルやメッセージを転送することなく、コマンド、スクリプト、または JCL を実行するためにも使用できます。 この機能は、 管理対象呼び出しと呼ばれます。

管理対象呼び出し要求は、以下のいくつかの方法でエージェントに実行依頼できます。

- fte: call Ant タスクを使用する。

- コマンドまたはスクリプトを実行するタスク XML を使用してリソース・モニターを構成する。 詳しくは、 コマンドおよびスクリプトを開始するためのモニター・タスクの構成 を参照してください。
- XML メッセージをエージェントのコマンド・キューに直接書き込む。 Managed Call XML スキーマについて詳しくは、 ファイル転送要求メッセージ・フォーマット を参照してください。

管理対象呼び出しの場合、エージェント・プロパティー **commandPath** に、実行されるコマンドまたはスクリプトを含むディレクトリーを指定する必要があります。

管理対象呼び出しは、エージェントの **commandPath** で指定されていないディレクトリーにあるコマンドやスクリプトを実行することはできません。 これは、エージェントが悪意のあるコードを実行しないようにするためです。

重要 : これを確実にするために、 **commandPath** を指定すると、デフォルトで以下のようになります。

- 既存のエージェント・サンドボックスは、エージェントの開始時にエージェントによって構成されます。 これにより、転送のアクセスを拒否したディレクトリーのリストに、すべての **commandPath** ディレクトリーが自動的に追加されます。
- エージェントの始動時に既存のユーザー・サンドボックスが更新され、すべての **commandPath** ディレクトリー (およびそのサブディレクトリー) が <exclude> エレメントとして <read> エレメントと <write> エレメントに追加されます。
- エージェントがエージェント・サンドボックスもユーザー・サンドボックスも使用するように構成されていない場合は、エージェントの開始時に、拒否されたディレクトリーとして指定された **commandPath** ディレクトリーを持つ新しいエージェント・サンドボックスが作成されます。

さらに、エージェントの権限検査を有効にして、許可されたユーザーのみが管理対象呼び出し要求を実行依頼できるようにすることもできます。 これについて詳しくは、 MFT エージェント・アクションに対するユーザー権限の制限を参照してください。

管理対象呼び出しの一部として呼び出されるコマンド、スクリプト、または JCL は、エージェントによってモニターされる外部プロセスとして実行されます。 プロセスが終了すると、管理対象呼び出しが完了し、**fte:call** Ant タスクを呼び出したエージェントまたは Ant スクリプトのいずれかが、プロセスからの戻りコードを使用できるようになります。

管理対象呼び出しが **fte:call** Ant タスクによって開始された場合、 Ant スクリプトは戻りコードの値を検査して、管理対象呼び出しが成功したかどうかを判別できます。

その他のすべてのタイプの管理対象呼び出しでは、管理対象呼び出しが正常に完了したことを示すために使用する戻りコード値を指定できます。 エージェントは、外部プロセスの終了時に、プロセスからの戻りコードをこれらの戻りコードと比較します。

注 : 管理対象呼び出しは外部プロセスとして実行されるため、いったん開始すると取り消すことはできません。

管理対象コールとソース転送スロット

エージェントには、 拡張エージェント・プロパティー: 転送制限で説明されているように、エージェント・プロパティー **maxSourceTransfers** で指定された数のソース転送スロットが含まれています。

管理対象呼び出しまたは管理対象転送が実行されるたびに、ソース転送スロットが占有されます。 このスロットは、管理対象通話または管理対象転送が完了すると解放されます。

エージェントが新規の管理対象呼び出し要求または管理対象転送要求のいずれかを受信したときに、すべてのソース転送スロットが使用中である場合、スロットが使用可能になるまで、要求はエージェントによってキューに入れられます。

管理対象呼び出しが管理対象転送を開始する場合 (例えば、管理対象呼び出しが Ant スクリプトを実行し、その Ant スクリプトが fte: filecopy タスクまたは fte: filemove タスクを使用してファイルを転送する場合)、以下の 2 つのソース転送スロットが必要です。

- 1 つは管理対象転送用
- 1 つは管理対象呼び出し用

この状況では、管理対象転送が完了するのに長い時間がかかるか、リカバリーに入ると、管理対象転送が完了するか、取り消されるか、 **transferRecoveryTimeout** が原因でタイムアウトになるまで、2 つのソース転送スロットが占有されることに注意してください。 **transferRecoveryTimeout** について詳しくは、 転送リカバリー・タイムアウトの概念 を参照してください。 これにより、エージェントが処理できる他の管理対象転送または管理対象呼び出しの数が制限される可能性があります。

このため、長期間ソース転送スロットを占有しないようにするために、管理対象呼び出しの設計を検討する必要があります。

管理対象呼び出しでの REST API の使用

V 9.3.0 V 9.3.0

HTTP GET および HTTP POST verb は、管理対象呼び出しを使用可能にするためにサポートされており、バージョン 3 の REST API でのみ機能します。

その他の verb (例えば、 HTTP DELETE and HTTP UPDATE など) はサポートされておらず、使用しようとすると HTTP 405 エラー・コードが返されます。

重要 : 一度実行依頼された管理対象呼び出しは、 REST API を使用して取り消すことはできません。

Apache Ant と MFT の併用

Managed File Transfer は、ファイル転送機能を Apache Ant ツールに組み込むために使用できるタスクを提供します。

fteAnt コマンドを使用して、既に構成済みの Managed File Transfer 環境で Ant タスクを実行できます。 Ant スクリプトのファイル転送 Ant タスクを使用し、逐次実行形式のスクリプト言語から複雑なファイル転送操作を統合することができます。

Apache Ant について詳しくは、Apache Ant プロジェクトの web ページ (https://ant.apache.org/) を参照してください。

関連概念

1228 ページの『MFT とともに Ant スクリプトの使用を開始する』
Managed File Transfer で Ant スクリプトを使用すると、解釈されたスクリプト言語から複雑なファイル転送操作を調整することができます。

fteAnt: MFT で Ant タスクを実行します。

関連資料

1229 ページの『MFT のサンプル Ant タスク』
Managed File Transfer のインストール済み環境には、いくつかのサンプル Ant スクリプトが用意されています。 これらのサンプルは、ディレクトリー *MQ_INSTALLATION_PATH*/mqft/samples/fteant にあります。 各サンプル・スクリプトには init ターゲットが含まれています。ご使用の構成でこれらのスクリプトを実行するには、 init ターゲットに設定されているプロパティーを編集します。

MFT とともに Ant スクリプトの使用を開始する

Managed File Transfer で Ant スクリプトを使用すると、解釈されたスクリプト言語から複雑なファイル転送操作を調整することができます。

Ant スクリプト

Ant スクリプト (またはビルド・ファイル) は、1 つ以上のターゲットを定義している XML 文書です。 これらのターゲットには、実行するタスク・エレメントが含まれます。 Managed File Transfer は、ファイル転送機能を Apache Ant に組み込むために使用できるタスクを提供します。 Ant スクリプトについて詳しくは、Apache Ant プロジェクトの Web ページ (https://ant.apache.org/) を参照してください。

Managed File Transfer タスクを使用する Ant スクリプトの例は、ディレクトリー *MQ_INSTALLATION_PATH*/mqft/samples/fteant の製品のインストールとともに提供されます。

プロトコル・ブリッジ・エージェントでは、Ant スクリプトはプロトコル・ブリッジ・エージェントのシステムで実行されます。 これらの Ant スクリプトは、FTP または SFTP サーバー上のファイルには直接アクセスすることはありません。

名前空間

名前空間は、ファイル転送用の Ant タスクを、同じ名前を共有している可能性のある別の Ant タスクと区別するために使用します。 名前空間は、Ant スクリプトのプロジェクト・タグに定義します。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

属性 `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` は、Ant に対し、ライブラリー `com.ibm.wmqfte.ant.taskdefs` にある接頭部 `fte` が付いたタスクの定義を探すよう命令します。

`fte` を名前空間の接頭部として使用する必要はなく、どの値でも使用できます。 名前空間の接頭部 `fte` は、すべての例およびサンプル Ant スクリプトで使用されます。

Ant スクリプトの実行

ファイル転送 Ant タスクを含む Ant スクリプトを実行するには、**fteAnt** コマンドを使用します。 以下に例を示します。

```
fteAnt -file ant_script_location/ant_script_name
```

詳しくは、 **fteAnt**: MFT での Ant タスクの実行を参照してください。

戻りコード

ファイル転送 Ant タスクは、Managed File Transfer コマンドと同じ戻りコードを戻します。 詳しくは、MFT の戻りコードを参照してください。

関連資料

fteAnt: MFT で Ant タスクを実行します。

1229 ページの『MFT のサンプル Ant タスク』

Managed File Transfer のインストール済み環境には、いくつかのサンプル Ant スクリプトが用意されています。 これらのサンプルは、ディレクトリー *MQ_INSTALLATION_PATH*/mqft/samples/fteant にあります。 各サンプル・スクリプトには `init` ターゲットが含まれています。ご使用の構成でこれらのスクリプトを実行するには、 `init` ターゲットに設定されているプロパティーを編集します。

MFT のサンプル Ant タスク

Managed File Transfer のインストール済み環境には、いくつかのサンプル Ant スクリプトが用意されています。 これらのサンプルは、ディレクトリー *MQ_INSTALLATION_PATH*/mqft/samples/fteant にあります。 各サンプル・スクリプトには `init` ターゲットが含まれています。ご使用の構成でこれらのスクリプトを実行するには、 `init` ターゲットに設定されているプロパティーを編集します。

E メール

email サンプルは、ファイルを転送し、転送が失敗した場合は指定した E メール・アドレスに E メールを送信する Ant タスクの使用例を示しています。 このスクリプトは、ソース・エージェントと宛先エージェントがアクティブであり、Managed File Transfer ping タスクを使用して転送を処理できることを確認します。 両方のエージェントがアクティブである場合、スクリプトは Managed File Transfer fte:filecopy タスクを使用して、元のファイルを削除することなく、ソース・エージェントと宛先エージェントとの間でファイル

を転送します。 転送が失敗すると、スクリプトは標準 Ant email タスクを使用して、失敗についての情報を含む E メールを送信します。

ハブ

ハブ・サンプルは、hubcopy.xml と hubprocess.xml. という 2 つのスクリプトで構成されています。 hubcopy.xml スクリプトは、Ant スクリプトを使用して「ハブ・アンド・スポーク」のスタイル・トポロジーを作成する方法を示しています。 このサンプルでは、2 つのファイルがスポーク・マシン上で実行中のエージェントからハブ・マシン上で実行中のエージェントに転送されます。 両方のファイルは同時に転送されます。転送が完了すると、ハブ・マシンで hubprocess.xml Ant スクリプトが実行され、ファイルが処理されます。 両方のファイルが正しく転送されると、Ant スクリプトはそれらのファイルの内容を連結します。 ファイルが正しく転送されない場合、Ant スクリプトは転送されたすべてのファイル・データを削除してクリーンアップします。 この例を正しく機能させるには、hubprocess.xml スクリプトをハブ・エージェントのコマンド・パスに配置する必要があります。 エージェントのコマンド・パスの設定について詳しくは、『commandPath MFT プロパティー』を参照してください。

librarytransfer (IBM i プラットフォームのみ)

IBM i

IBM i librarytransfer サンプルは、Ant タスクを使用して、1 つの IBM i システム上の IBM i ライブラリーを 2 番目の IBM i システムに転送する方法を示しています。

IBM i librarytransfer サンプルは、IBM i 上のネイティブ保存ファイル・サポートと、Managed File Transfer で使用可能な事前定義 Ant タスクを使用して、2 つの IBM i システム間でネイティブ・ライブラリー・オブジェクトを転送します。 このサンプルは、 Managed File Transfer filecopy タスクで < presrc> ネスト・エレメントを使用して、ソース・エージェント・システム上の要求されたライブラリーを一時保存ファイルに保存する実行可能スクリプト librarysave.sh を呼び出します。 保存ファイルは、filecopy Ant タスクによって宛先エージェント・システムに移動されます。宛先エージェント・システムでは、< postdst> ネスト・エレメントを使用して実行可能スクリプト libraryrestore.sh を呼び出し、保存ファイルに保存されているライブラリーを宛先システムに復元します。

IBM i このサンプルを実行する前に、librarytransfer.xml ファイルで説明されているようにいくつかの構成を完了する必要があります。 また、2 つの IBM i マシン上に Managed File Transfer 環境が稼働している必要があります。 セットアップには、1 つ目の IBM i マシンで実行されるソース・エージェントと、2 つ目の IBM i マシンで実行される宛先エージェントを含める必要があります。 2 つのエージェントは相互通信が可能でなければなりません。

IBM i librarytransfer サンプルは、次の 3 つのファイルで構成されます。

- librarytransfer.xml
- librarysave.sh (<presrc> 実行可能スクリプト)
- libraryrestore.sh (<postdst> 実行可能スクリプト)

サンプル・ファイルは、以下のディレクトリーにあります。/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer

IBM i ユーザーは、このサンプルを実行するために、以下のステップを実行する必要があります。

1. Qshell セッションを開始します。 IBM i コマンド・ウィンドウで、 STRQSH と入力します。
2. 以下のように、ディレクトリーを bin ディレクトリーに変更します。

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 必要な構成を行った後、次のコマンドを使用してサンプルを実行します。

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer (IBM i プラットフォームのみ)

IBM i physicalfiletransfer サンプルは、Ant タスクを使用して、1 つの IBM i システム上のライブラリーから 2 番目の IBM i システム上のライブラリーにソース物理ファイルまたはデータベース・ファイルを転送する方法を示しています。

IBM i physicalfiletransfer サンプルは、IBM i 上のネイティブ保存ファイル・サポートと、Managed File Transfer で使用可能な事前定義 Ant タスクを使用して、2 つの IBM i システム間で完全なソース物理ファイルおよびデータベース・ファイルを転送します。 このサンプルでは、 Managed File Transfer filecopy タスク内で < presrc> ネスト・エレメントを使用して、実行可能スクリプト `physicalfilesave.sh` を呼び出し、ソース・エージェント・システム上のライブラリーから要求されたソース物理ファイルまたはデータベース・ファイルを一時保存ファイルに保存します。 保存ファイルは、filecopy Ant タスクによって宛先エージェント・システムに移動されます。宛先エージェント・システムでは、< postdst> ネストされたエレメントを使用して実行可能スクリプト `physicalfilerestore.sh` を呼び出し、保存ファイル内のファイル・オブジェクトを宛先システム上の指定されたライブラリーに復元します。

IBM i このサンプルを実行する前に、`physicalfiletransfer.xml` ファイルで説明されているようにいくつかの構成を完了する必要があります。 また、2 つの IBM i システム上に Managed File Transfer 環境が稼働している必要があります。 セットアップには、1 つ目の IBM i システムで実行されるソース・エージェントと、2 つ目の IBM i システムで実行される宛先エージェントを含める必要があります。 2 つのエージェントは相互通信が可能でなければなりません。

IBM i physicalfiletransfer サンプルは、次の 3 つのファイルで構成されます。

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (<presrc> 実行可能スクリプト)
- `physicalfilerestore.sh` (<postdst> 実行可能スクリプト)

サンプル・ファイルは、以下のディレクトリーにあります。`/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i ユーザーは、このサンプルを実行するために、以下のステップを実行する必要があります。

1. Qshell セッションを開始します。 IBM i コマンド・ウィンドウで、 STRQSH と入力します。
2. 以下のように、ディレクトリーを `bin` ディレクトリーに変更します。

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. 必要な構成を行った後、次のコマンドを使用してサンプルを実行します。

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

timeout

timeout サンプルは、ファイル転送を試行し、指定したタイムアウト値より長くかかる場合は転送を取り消す Ant タスクの使用例を示しています。 スクリプトは、Managed File Transfer fte:filecopy タスクを使用してファイル転送を開始します。 この転送の結果は保留になります。 スクリプトは、Managed File Transfer fte:awaitoutcome Ant タスクを使用して、指定された秒数の間、転送の完了を待機します。 指定された時間内に転送が完了しない場合、ファイル転送の取り消しに Managed File Transfer fte:cancel Ant タスクが使用されます。

vsamtransfer

z/OS

z/OS vsamtransfer サンプルは、 Ant タスクを使用して、 Managed File Transfer を使用して VSAM データ・セットから別の VSAM データ・セットに転送する方法を示しています。 Managed File Transfer

は、現在、VSAM データ・セットの転送をサポートしていません。 サンプル・スクリプトは、 presrc プログラム呼び出しのネストされたエレメント を使用して実行可能ファイル datasetcopy.sh を呼び出すことにより、VSAM データ・レコードを順次データ・セットにアンロードします。 スクリプトは、Managed File Transfer fte:filemove タスクを使用して、ソース・エージェントから宛先エージェントに順次データ・セットを転送します。 その後、スクリプトは postdst プログラム呼び出しのネストされたエレメント を使用して loadvsam.jcl スクリプトを呼び出します。 この JCL (ジョブ制御言語) スクリプトは、転送されたデータ・セット・レコードを宛先の VSAM データ・セットにロードします。 このサンプルでは、宛先呼び出しに JCL を使用してこの言語オプションを説明します。 代わりに、2 番目のシェル・スクリプトを使用しても、同じ結果を得ることができます。

z/OS このサンプルでは、ソースおよび宛先データ・セットは VSAM である必要はありません。 ソースと宛先のデータ・セットが同じタイプであれば、サンプルはどのデータ・セットでも機能します。

z/OS このサンプルを正しく機能させるには、datasetcopy.sh スクリプトをソース・エージェントのコマンド・パスに置き、loadvsam.jcl スクリプトを宛先エージェントのコマンド・パスに置く必要があります。 エージェントのコマンド・パスの設定について詳しくは、『commandPath MFT プロパティー』を参照してください。

zip

zip サンプルは、zip.xml と zipfiles.xml の 2 つのスクリプトで構成されています。 このサンプルは、Managed File Transfer fte: filemove タスク内で presrc nested 要素 を使用して、ファイル転送移動操作を実行する前に Ant スクリプトを実行する方法を示しています。 zip.xml スクリプト内の presrc ネスト・エレメントによって呼び出される zipfiles.xml スクリプトは、ディレクトリーの内容を圧縮します。 zip.xml スクリプトは、圧縮ファイルを転送します。 このサンプルでは、zipfiles.xml Ant スクリプトがソース・エージェントのコマンド・パスに存在している必要があります。 これは、zipfiles.xml Ant スクリプトに、ソース・エージェントでディレクトリーの内容を圧縮するために使用されるターゲットが含まれているためです。 エージェントのコマンド・パスの設定について詳しくは、『commandPath MFT プロパティー』を参照してください。

関連概念

1228 ページの『MFT とともに Ant スクリプトの使用を開始する』
Managed File Transfer で Ant スクリプトを使用すると、解釈されたスクリプト言語から複雑なファイル転送操作を調整することができます。

関連資料

fteAnt: MFT で Ant タスクを実行します。

ユーザー出口での MFT のカスタマイズ

Managed File Transfer のフィーチャーは、ユーザー出口ルーチンと呼ばれる独自のプログラムを使用してカスタマイズできます。

重要 : ユーザー出口内のコードは IBM ではサポートされません。そのコードに関する問題は、最初に、お客様の企業または出口を提供したベンダーのいずれかが調査する必要があります。

Managed File Transfer は、ユーザーが作成したプログラム (ユーザー出口ルーチン) に Managed File Transfer が制御を渡すことができる、コード内のポイントを提供します。 これらのポイントは、ユーザー出口ポイント (出口点) と呼ばれます。 その後 Managed File Transfer は、作成したプログラムが処理を終了した時に制御を再開できます。 ユーザー出口は必ずしも使用する必要はありませんが、特定の要件を満たすために Managed File Transfer システムの機能を拡張およびカスタマイズする場合に便利です。

ファイル転送の処理中にソース・システムでユーザー出口を呼び出せるポイントが 2 つあり、ファイル転送の処理中に宛先システムでユーザー出口を呼び出せるポイントが 2 つあります。 以下の表は、これらの各ユーザー出口点と、出口点を利用するために実装する必要がある Java インターフェースを要約しています。

表 *187.* ソース側および宛先側の出口点と *Java* インターフェースの要約	
出口点	**実装する Java インターフェース**
ソース側の出口点:	
ファイル転送全体が開始する前	SourceTransferStartExit.java インターフェース
ファイル転送全体の完了後	SourceTransferEndExit.java インターフェース
宛先側の出口点:	
ファイル転送全体が開始する前	DestinationTransferStartExit.java インターフェース
ファイル転送全体の完了後	DestinationTransferEndExit.java インターフェース

ユーザー出口は次の順序で起動されます。

1. SourceTransferStartExit
2. DestinationTransferStartExit
3. DestinationTransferEndExit
4. SourceTransferEndExit

SourceTransferStartExit および DestinationTransferStartExit 出口で行われた変更内容は、以降の出口への入力として伝搬されます。 例えば、SourceTransferStartExit 出口が転送メタデータを変更する場合、変更内容は他の出口への入力転送メタデータに反映されます。

ユーザー出口とプログラム呼び出しは、以下の順序で呼び出されます。

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

注 :

1. **DestinationTransferEndExits** は、転送が正常に完了したか、部分的に正常に完了した場合にのみ実行されます。
2. **postDestinationCall** は、転送が正常に完了したか、部分的に正常に完了した場合にのみ実行されます。
3. **SourceTransferEndExits** は、成功した転送、部分的に成功した転送、または失敗した転送に対して実行されます。
4. **postSourceCall** は、以下の場合にのみ呼び出されます。
   - 転送は取り消されませんでした。
   - 成功した結果または部分的に成功した結果があります。
   - 宛先転送後プログラムは正常に実行されました。

ユーザー出口のビルド

ユーザー出口を構築するためのインターフェースは、*MQ_INSTALL_DIRECTORY*/mqft/lib/com.ibm.wmqfte.exitroutines.api.jar に含まれています。 出口をビルドする際には、この .jar ファイルをクラスパスに含める必要があります。 出口を実行するには、出口を .jar ファイルとして抽出してから、 以下のセクションで示すディレクトリーにその .jar ファイルを配置します。

ユーザー出口の場所

ユーザー出口ルーチンは、次の 2 つの候補となる場所に保管できます。

- exits ディレクトリー それぞれの agent ディレクトリーの下位に exits ディレクトリーがあります。 例 : var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits
- exitClassPath プロパティーを設定して代わりの場所を指定できます。 exits ディレクトリーと exitClassPath によって設定されたクラスパスの両方に出口クラスがある場合、exits ディレクトリー内のクラスは優先されます。これは、両方のロケーションに同じ名前のクラスがある場合、exits ディレクトリー内のクラスが優先されることを意味します。

ユーザー出口を使用するようにエージェントを構成する

エージェントが呼び出すユーザー出口を指定するために設定可能なエージェント・プロパティーが 4 つあります。 これらのエージェント・プロパティーは、 sourceTransferStartExitClasses、 sourceTransferEndExitClasses、 destinationTransferStartExitClasses、および destinationTransferEndExitClasses です。 これらのプロパティーの使用方法については、ユーザー出口用の MFT エージェント・プロパティーを参照してください。

プロトコル・ブリッジ・エージェントでのユーザー出口の実行

ソース・エージェントが出口を呼び出すと、転送対象のソース・アイテムのリストが出口に渡されます。 通常のエージェントの場合、これは完全修飾ファイル名のリストです。 ファイルはローカル (またはマウントを介してアクセス可能) であるため、出口はファイルにアクセスして暗号化することができます。

ただし、プロトコル・ブリッジ・エージェントの場合、リスト内の項目は次の形式になります。

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

リスト内の項目ごとに、出口は最初にファイル・サーバーに接続し、(FTP や FTPS または SFTP プロトコルを使用して) ファイルをダウンロードし、ローカルで暗号化してから、暗号化されたファイルをファイル・サーバーにアップロードする必要があります。

Connect:Direct ブリッジ・エージェントでのユーザー出口の実行

Connect:Direct® ブリッジ・エージェントでユーザー出口を実行することはできません。

関連概念

1234 ページの『MFT のソースと宛先のユーザー出口』

MFT ユーザー出口のメタデータ

MFT ユーザー出口の Java インターフェース

関連資料

1239 ページの『MFT ユーザー出口のリモート・デバッグの使用可能化』

ユーザー出口を作成する際、 コード中の問題を見つけるために有用なデバッガーを使用したい場合があります。

1240 ページの『MFT ソース転送ユーザー出口のサンプル』

1241 ページの『プロトコル・ブリッジ資格情報ユーザー出口のサンプル』

MFT リソース・モニター・ユーザー出口

ユーザー出口用の MFT エージェント・プロパティー

MFT のソースと宛先のユーザー出口

ディレクトリー分離文字

ソース・ファイル仕様のディレクトリー分離文字は、**fteCreateTransfer** コマンドまたは IBM MQ Explorer でどのようにディレクトリー分離文字を指定したかに関係なく、常にスラッシュ (/) 文字を使用し

て表されます。 出口を作成するときはそのことを考慮に入れる必要があります。 例えば、 `c:\a\b.txt` というソース・ファイルが存在することを確認し、 **fteCreateTransfer** コマンドまたは IBM MQ Explorer を使用してこのソース・ファイルを指定した場合、ファイル名は実際には `c:/a/b.txt` として保管されていることに注意してください。したがって、 `c:\a\b.txt` という元のストリングを検索しても、一致するものは見つかりません。

ソース側の出口点

ファイル転送全体が開始する前

この出口は、保留中の転送のリスト内の次の項目に転送要求があり、その転送が開始されようとしている時に、ソース・エージェントにより呼び出されます。

この出口点の使用例としては、各段階で外部コマンドを使用してエージェントが読み取り/書き込み権限を持つディレクトリーにファイルを送る、宛先システム上のファイルを名前変更する、などがあります。

次の引数をこの出口に渡します。

- ソース・エージェント名
- 宛先エージェント名
- 環境メタデータ
- 転送メタデータ
- ファイル仕様 (ファイル・メタデータを含む)

この出口から次のデータが返されます。

- 更新された転送メタデータ。 項目を追加、変更、および削除できます。
- ファイル仕様の更新されたリスト。ソース・ファイル名と宛先ファイル名のペアで構成されます。 項目を追加、変更、および削除できます。
- 転送を続行するかどうかを指定する標識。
- 転送ログに挿入するストリング。

この出口点でユーザー出口コードを呼び出す SourceTransferStartExit.java インターフェースを実装します。

ファイル転送全体の完了後

この出口は、ファイル転送全体が完了した後にソース・エージェントにより呼び出されます。

この出口点の使用例として、転送が完了したことを通知するための E メールまたは IBM MQ メッセージの送信など、完了タスクの実行があります。

次の引数をこの出口に渡します。

- 転送出口の結果
- ソース・エージェント名
- 宛先エージェント名
- 環境メタデータ
- 転送メタデータ
- ファイル結果

この出口から次のデータが返されます。

- 転送ログに挿入する更新されたストリング。

この出口点でユーザー出口コードを呼び出す SourceTransferEndExit.java インターフェースを実装します。

宛先側の出口点

ファイル転送全体が開始する前

この出口点の使用例として、宛先での許可の検証があります。

次の引数をこの出口に渡します。

- ソース・エージェント名
- 宛先エージェント名
- 環境メタデータ
- 転送メタデータ
- ファイル仕様

この出口から次のデータが返されます。

- 更新された一連の宛先ファイル名。 項目の変更はできますが、追加または削除はできません。
- 転送を続行するかどうかを指定する標識。
- 転送ログに挿入するストリング。

この出口点でユーザー出口コードを呼び出す DestinationTransferStartExit.java インターフェースを実装します。

ファイル転送全体の完了後

この出口点の使用例として、転送されたファイルを使用するバッチ処理の開始、転送に失敗した場合の E メールの送信などがあります。

次の引数をこの出口に渡します。

- 転送出口の結果
- ソース・エージェント名
- 宛先エージェント名
- 環境メタデータ
- 転送メタデータ
- ファイル結果

この出口から次のデータが返されます。

- 転送ログに挿入する更新されたストリング。

この出口点でユーザー出口コードを呼び出す DestinationTransferEndExit.java インターフェースを実装します。

関連概念

MFT ユーザー出口の Java インターフェース

関連資料

1239 ページの『MFT ユーザー出口のリモート・デバッグの使用可能化』
ユーザー出口を作成する際、 コード中の問題を見つけるために有用なデバッガーを使用したい場合があります。

1240 ページの『MFT ソース転送ユーザー出口のサンプル』

MFT リソース・モニター・ユーザー出口

MFT 転送入出力ユーザー出口の使用

Managed File Transfer 転送入出力ユーザー出口を使用してカスタム・コードを構成し、Managed File Transfer 転送用の基礎ファイル・システムの入出力処理を実行することができます。

MFT 転送では、通常、適宜のファイル・システムと対話するための 1 つの組み込み入出力プロバイダーがエージェントによって選択され、転送が行われます。 組み込み入出力プロバイダーは、次のタイプのファイル・システムをサポートします。

- 通常の UNIX タイプおよび Windows タイプのファイル・システム
- z/OS z/OS 順次および区分データ・セット (z/OS のみ)
- IBM i IBM i ネイティブ保存ファイル (IBM i のみ)
- IBM MQ キュー
- リモート FTP および SFTP プロトコル・サーバー (プロトコル・ブリッジ・エージェントのみ)
- リモート Connect:Direct ノード (Connect:Direct ブリッジ・エージェントのみ)

サポートされていないファイル・システムまたはカスタム入出力動作が必要なファイル・システム用に、転送入出力ユーザー出口を作成することができます。

転送入出力ユーザー出口は、ユーザー出口に既存のインフラストラクチャーを使用します。 しかし、この転送入出力ユーザー出口が他のユーザー出口と異なるのは、 各ファイルの転送処理中に転送入出力ユーザー出口の機能に対して複数回のアクセスが行われるということです。

エージェント・プロパティー IOExitClasses (`agent.properties` ファイル内) を使用して、ロードする入出力出口クラスを指定します。 各出口クラスは、次のようにコンマで区切ります。

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

転送入出力ユーザー出口の Java インターフェースは、次のとおりです。

IOExit
入出力出口が使用されているかどうかを判別するために使用されるメインの入り口点。 このインスタンスによって、IOExitPath インスタンスが作成されます。

エージェント・プロパティー IOExitClasses には IOExit 入出力出口インターフェースを指定するだけで十分です。

IOExitPath
抽象インターフェースを表します。例えば、1 つのデータ・コンテナー、または一式のデータ・コンテナーを表すワイルドカードなどです。 このインターフェースを実装するクラス・インスタンスを作成することはできません。 このインターフェースは、パスを検査したり、派生パスをリストすることができます。 IOExitResourcePath インターフェースと IOExitWildcardPath インターフェースは IOExitPath を拡張したものです。

IOExitChannel
IOExitPath リソースとの間でのデータの読み書きを有効にします。

IOExitRecordChannel
レコード単位 IOExitPath リソースの IOExitChannel インターフェースを拡張して、IOExitPath リソースとの間でレコード単位のデータの読み書きを有効にします。

IOExitLock
共用アクセスまたは排他的アクセスの IOExitPath リソースでのロックを表します。

z/OS **IOExitRecordResourcePath**
IOExitResourcePath インターフェースを拡張して、レコード単位ファイル (z/OS データ・セットなど) のデータ・コンテナーを表せるようにします。 このインターフェースを使用して、データの位置を指定したり、読み書き操作のための IOExitRecordChannel インスタンスを作成したりすることができます。

IOExitResourcePath
IOExitPath インターフェースを拡張して、ファイルやディレクトリーなどのデータ・コンテナーを表せるようにします。 このインターフェースを使用して、データの位置を指定することができます。 このインターフェースでディレクトリーを表す場合は、パスのリストが返されるようにするために listPaths メソッドを使用できます。

IOExitWildcardPath
IOExitPath インターフェースを拡張して、ワイルドカードを使用したパスを表せるようにします。 このインターフェースを使用して、複数の IOExitResourcePaths のマッチングが行えます。

IOExitProperties
Managed File Transfer が入出力の特定の側面の IOExitPath を処理する方法を決めるプロパティーを指定します。 例えば、中間ファイルを使用するかどうか、または転送が再開された場合にリソースを先頭から再読み取りするかどうかを指定します。

関連概念
1232 ページの『ユーザー出口での MFT のカスタマイズ』
Managed File Transfer のフィーチャーは、ユーザー出口ルーチンと呼ばれる独自のプログラムを使用してカスタマイズできます。

関連資料
IOExit.java インターフェース
IOExitChannel.java インターフェース
IOExitLock.java インターフェース
IOExitPath.java インターフェース
IOExitProperties.java インターフェース
IOExitRecordChannel.java インターフェース
z/OS IOExitRecordResourcePath.java インターフェース
IOExitResourcePath.java インターフェース
IOExitWildcardPath.java インターフェース
MFTagent.properties ファイル

IBM i IBM i ユーザー出口でのサンプル MFT

Managed File Transfer では、IBM i に固有のサンプル・ユーザー出口がインストール済み環境に用意されています。 これらのサンプルは、ディレクトリー *MQMFT_install_dir*/samples/ioexit-IBMi および *MQMFT_install_dir*/samples/userexit-IBMi にあります。

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit サンプル・ユーザー出口は、IBM i の QDLS ファイル・システム内のファイルを転送します。 出口がインストールされると、/QDLS で始まるファイルへの転送では、自動的にその出口が使用されます。

この出口をインストールするには、以下のステップを実行します。

1. com.ibm.wmqfte.samples.ibmi.ioexits.jar ファイルを *WMQFTE_install_dir*/samples/ioexit-IBMi ディレクトリーからエージェントの exits ディレクトリーにコピーします。
2. com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit を IOExitClasses プロパティーに追加します。
3. エージェントを再始動します。

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit
com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit サンプル・ユーザー出口は、MFT ファイル・モニターのように動作し、IBM i ライブラリーから物理ファイル・メンバーを自動的に転送します。

この出口を実行するには、(例えば **-md** パラメーターを使用して)「library.qsys.monitor」メタデータ・フィールドに値を指定します。 このパラメーターでは、ファイル・メンバーへの IFS スタイルのパスが使用され、ファイルおよびメンバーのワイルドカードを含めることができます。 例えば、/QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR、/QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR、/QSYS.LIB/FOO.LIB/*.FILE/*.MBR のようになります。

また、このサンプル出口には、オプションのメタデータ・フィールド「naming.scheme.qsys.monitor」もあり、転送中に使用される命名体系の判別に使用できます。 デフォルトでは、このフィールドは

「unix」に設定されています。これにより、宛先ファイルは FOO.MBR と呼ばれます。 値「ibmi」を指定して、 IBM i FTP FILE.MEMBER スキーム (例えば、 /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR は BAR.BAZ。

この出口をインストールするには、以下のステップを実行します。

1. com.ibm.wmqfte.samples.ibmi.userexits.jar ファイルを *WMQFTE_install_dir*/samples/userexit-IBMi ディレクトリーからエージェントの exits ディレクトリーにコピーします。
2. com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit を agent.properties ファイルの sourceTransferStartExitClasses プロパティーに追加します。
3. エージェントを再始動します。

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit サンプル・ユーザー出口は、ソース・ファイル・メンバーが転送の一環として削除される時に空のファイル・オブジェクトを削除します。 IBM i ファイル・オブジェクトは多くのメンバーを保持している可能性があるため、MFT ではファイル・オブジェクトはディレクトリーのように扱われます。 したがって、MFT を使用してファイル・オブジェクトに移動操作を実行することはできません。移動操作は、メンバー・レベルのみでサポートされます。 その結果、メンバーに対して移動操作を行うと、空のファイルが残されます。 これらの空のファイルを転送要求の一環として削除する場合に、このサンプル出口を使用します。

「empty.file.delete」メタデータに「true」を指定して FTEFileMember を転送すると、このサンプル出口は、親ファイルが空の場合にはその親ファイルを削除します。

この出口をインストールするには、以下のステップを実行します。

1. com.ibm.wmqfte.samples.ibmi.userexits.jar ファイルを *WMQFTE_install_dir*/samples/userexit-IBMi からエージェントの exits ディレクトリーにコピーします。
2. com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit を agent.properties ファイル内の sourceTransferStartExitClasses プロパティーに追加します。
3. エージェントを再始動します。

関連資料

1236 ページの『MFT 転送入出力ユーザー出口の使用』
Managed File Transfer 転送入出力ユーザー出口を使用してカスタム・コードを構成し、Managed File Transfer 転送用の基礎ファイル・システムの入出力処理を実行することができます。

ユーザー出口用の MFT エージェント・プロパティー

MFT ユーザー出口のリモート・デバッグの使用可能化

ユーザー出口を作成する際、 コード中の問題を見つけるために有用なデバッガーを使用したい場合があります。

出口は、エージェントを実行する Java 仮想マシン内部で実行されるため、統合開発環境に通常組み込まれている直接的なデバッグ・サポートを使用することができません。 しかし、JVM のリモート・デバッグを使用可能にして、 適切なリモート・デバッガーに接続することができます。

リモート・デバッグを有効にするには、標準 JVM パラメーター **-Xdebug** および **-Xrunjdwp** を使用します。 これらのプロパティーは、**BFG_JVM_PROPERTIES** 環境変数によって、エージェントを実行する JVM に渡されます。 例えば、AIX and Linux では、以下のコマンドによって、エージェントを開始し、JVM が TCP ポート 8765 でデバッガー接続を listen するようにします。

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

エージェントは、デバッガーが接続するまで開始しません。 Windows では、 **export** コマンドの代わりに **set** コマンドを使用してください。

また、デバッガーと JVM の間で他の通信方式を使用することもできます。 例えば、JVM に接続するのではなく、逆に JVM がデバッガーへの接続を開始することもできます。 また、TCP の代わりに、共有メモリーを使用することもできます。 詳しくは、Java Platform Debugger Architecture 資料を参照してください。

エージェントをリモート・デバッグ・モードで開始する場合は、 **-F** (フォアグラウンド) パラメーターを使用する必要があります。

Eclipse デバッガーの使用

以下のステップは、 Eclipse 開発環境のリモート・デバッグ機能に適用されます。 また、JPDA 準拠の他のリモート・デバッガーを使用することもできます。

1. 「**実行**」 > 「**デバッグ・ダイアログを開く**」 (Eclipse のバージョンによっては 「**実行**」 > 「**デバッグの構成**」 または「**実行**」 > 「**デバッグ・ダイアログ**」)
2. 構成タイプのリストにある「**リモート Java アプリケーション**」をダブルクリックして、 デバッグ構成を作成します。
3. 構成フィールドに入力し、デバッグ構成を保存します。 エージェントの JVM をデバッグ・モードで既に開始している場合は、すぐに JVM に 接続できます。

MFT ソース転送ユーザー出口のサンプル

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
```

```
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();

            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

プロトコル・ブリッジ資格情報ユーザー出口のサンプル

このユーザー出口のサンプルの使用法については、出口クラスを使用したファイル・サーバーの資格情報のマップを参照してください。

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {
```

```
    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
                          System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                         initialisationResult = false;
                     }
                 }
             }

             if (initialisationResult) {
                 // Populate the map of mqUserId to server credentials from the properties
                 final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                  while ( propertyNames.hasMoreElements()) {
                     final Object name = propertyNames.nextElement();
                     if (name instanceof String ) {
                         final String mqUserId = ((String)name).trim();
                          // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
```

```
                            serverPassword = valueTokenizer.nextToken().trim();
                        }
                        // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                        // Insert the credentials into the map
                        credentialsMap.put(mqUserId, credentials);
                    }
                }

            }
        }

        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

プロトコル・ブリッジ・プロパティー・ユーザー出口のサンプル

このユーザー出口のサンプルの使用法については、ProtocolBridgePropertiesExit2: プロトコル・ファイル・サーバー・プロパティーの検索を参照してください

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
```

```
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
```

```
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
```

```
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

エージェント・コマンド・キューにメッセージを PUT することによる MFT の制御

エージェント・コマンド・キューにメッセージを PUT することによって、Managed File Transfer を制御するアプリケーションを作成することができます。

エージェントのコマンド・キューにメッセージを PUT し、エージェントが次のいずれかの操作を実行するように要求できます。

- ファイル転送の作成
- スケジュール済みファイル転送の作成
- ファイル転送を取り消す
- スケジュール済みファイル転送の取り消し
- コマンドの呼び出し
- モニターの作成
- モニターの削除
- ping を返し、エージェントがアクティブであることを示す

これらのいずれかの操作をエージェントが実行するよう要求するには、メッセージが次のいずれかのスキーマに準拠した XML 形式でなければなりません。

FileTransfer.xsd
この形式のメッセージは、ファイル転送またはスケジュール済みファイル転送の作成、コマンドの呼び出し、およびファイル転送またはスケジュール済みファイル転送の取り消しに使用できます。 詳しくは、ファイル転送要求メッセージ・フォーマットを参照してください。

Monitor.xsd
この形式のメッセージはリソース・モニターの作成または削除に使用できます。 詳しくは、MFT モニター要求メッセージ・フォーマットを参照してください。

PingAgent.xsd
この形式のメッセージは、エージェントがアクティブであることを検査するよう、エージェントを ping するのに使用できます。 詳しくは、Ping MFT エージェント要求メッセージ・フォーマットを参照してください。

エージェントは要求メッセージに対して応答を返します。 応答メッセージは、要求メッセージに定義されている応答キューに PUT されます。 応答メッセージは、次のスキーマで定義された XML 形式です。

Reply.xsd
詳しくは、MFT エージェント応答メッセージ・フォーマットを参照してください。

MQ Telemetry 用アプリケーションの開発

テレメトリー・アプリケーションは、センス装置や制御装置を、インターネット上および企業内の使用可能な他の情報源と統合します。

MQ Telemetry のアプリケーションの開発には、設計パターン、工夫された例、サンプル・プログラム、プログラミングの概念、参照情報を使用します。

関連概念

MQ Telemetry

Telemetry のユースケース

関連タスク

MQ Telemetry のインストール

MQ Telemetry の管理

MQ Telemetry の問題のトラブルシューティング

関連資料

MQ Telemetry リファレンス

IBM MQ Telemetry Transport サンプル・プログラム

サンプル・スクリプトは、サンプル IBM MQ Telemetry Transport v3 クライアント・アプリケーション (`mqttv3app.jar`) を処理する場合に提供されます。IBM MQ 8.0.0　以降、サンプル・クライアント・アプリケーションは MQ Telemetry に組み込まれなくなりました。 これは、(現在使用できない) IBM Messaging Telemetry Clients SupportPac の一部でした。 今後も同様のサンプル・アプリケーションを Eclipse Paho と MQTT.org から自由に利用できます。

最新情報とダウンロードについては、以下のリソースを参照してください。

- Eclipse パホ プロジェクトと MQTT.org には、プログラミング言語の範囲の最新のテレメトリー・クライアントとサンプルの無料ダウンロードがあります。 これらのサイトを使用すると、IBM MQ Telemetry Transport のパブリッシュやサブスクライブを行ったり、セキュリティー機能を追加したりするためのサンプル・プログラムを開発するのに役立ちます。
- IBM Messaging Telemetry Clients SupportPac はダウンロードできなくなりました。 以前にダウンロードしたものがある場合は、以下の内容が含まれています。
  - IBM Messaging Telemetry Clients SupportPac の MA9B バージョンは、コンパイル済みのサンプル・アプリケーション (`mqttv3app.jar`) と、関連するクライアント・ライブラリー (`mqttv3.jar`) が含まれていました。それらは次のディレクトリーで提供されていました。
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - この SupportPac の MA9C バージョンでは、`/SDK/` ディレクトリーとコンテンツが削除されました。
    - サンプル・アプリケーション (`mqttv3app.jar`) のソースのみが提供されました。 次のディレクトリー内にありました。

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - コンパイル済みのクライアント・ライブラリーが引き続き提供されていました。 次のディレクトリー内にありました。

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

(現在使用できない) IBM Messaging Telemetry Clients SupportPac のコピーが引き続き存在する場合、サンプル・アプリケーションのインストールと実行については、コマンド行を使用した MQ Telemetry の検証を参照してください。

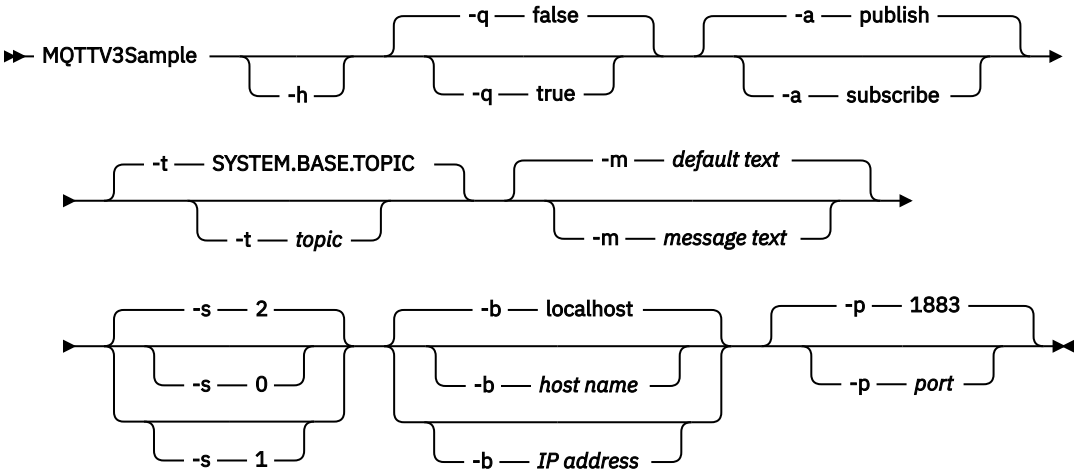
MQTTV3Sample プログラム

MQTTV3Sample プログラムのサンプル構文およびパラメーターに関する参照情報。

目的

MQTTV3Sample プログラムは、メッセージをパブリッシュし、トピックにサブスクライブするために使用できます。 このサンプル・プログラムの入手方法については、1247 ページの『IBM MQ Telemetry Transport サンプル・プログラム』を参照してください。

MQTTV3Sample syntax



パラメーター

-h
このヘルプ・テキストを出力して終了します。

-q
デフォルト・モードの false を使用する代わりに抑止モードを設定します。

-a
デフォルト・アクションのパブリッシュを想定する代わりにパブリッシュまたはサブスクライブを設定します。

-t
デフォルトのトピックにパブリッシュまたはサブスクライブする代わりに、トピックにパブリッシュまたはサブスクライブします。

-m
デフォルトのパブリケーション・テキスト「Hello from an MQTT v3 application」を送信する代わりに、メッセージ・テキストをパブリッシュします。

-s
デフォルトの QoS 2 を使用する代わりに QoS を設定します。

-b
デフォルトのホスト名の localhost に接続する代わりに、このホスト名または IP アドレスに接続します。

-p
デフォルトの 1883 を使用する代わりに、このポートを使用します。

MQTTV3Sample プログラムの実行

Windows 上で、トピックにサブスクライブするには、次のコマンドを使用します。

```
run MQTTV3Sample -a subscribe
```

Windows 上で、メッセージをパブリッシュするには、次のコマンドを使用します。

```
run MQTTV3Sample
```

MQTT クライアント・プログラミングの概念

このセクションで説明する概念は、MQTT protocol のクライアント・ライブラリーを理解するために役立ちます。 この概念は、クライアント・ライブラリーに付属している API ドキュメンテーションを補足するものです。

最新情報とダウンロードについては、以下のリソースを参照してください。

- Eclipse パホ プロジェクトと MQTT.org には、プログラミング言語の範囲の最新のテレメトリー・クライアントとサンプルの無料ダウンロードがあります。 これらのサイトを使用すると、IBM MQ Telemetry Transport のパブリッシュやサブスクライブを行ったり、セキュリティー機能を追加したりするためのサンプル・プログラムを開発するのに役立ちます。
- IBM Messaging Telemetry Clients SupportPac はダウンロードできなくなりました。 以前にダウンロードしたものがある場合は、以下の内容が含まれています。
  - IBM Messaging Telemetry Clients SupportPac の MA9B バージョンは、コンパイル済みのサンプル・アプリケーション (`mqttv3app.jar`) と、関連するクライアント・ライブラリー (`mqttv3.jar`) が含まれていました。それらは次のディレクトリーで提供されていました。
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - この SupportPac の MA9C バージョンでは、`/SDK/` ディレクトリーとコンテンツが削除されました。
    - サンプル・アプリケーション (`mqttv3app.jar`) のソースのみが提供されました。 次のディレクトリー内にありました。

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - コンパイル済みのクライアント・ライブラリーが引き続き提供されていました。 次のディレクトリー内にありました。

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

MQTT クライアントを開発して実行するには、これらのリソースをクライアント装置にコピーまたはインストールする必要があります。 別個のクライアント・ランタイムをインストールする必要はありません。

クライアントのライセンス条件は、クライアントの接続先のサーバーに関連付けられます。

MQTT クライアント・ライブラリーは、MQTT protocol の参照実装です。 ユーザー独自のクライアントを、さまざまな装置プラットフォームに適したさまざまな言語で実装できます。 IBM MQ Telemetry Transport のフォーマットおよびプロトコルを参照してください。

API ドキュメンテーションでは、クライアントがどの MQTT サーバーに接続しているかについて想定していません。 クライアントの動作は、異なるサーバーに接続するときに若干の違いが出る可能性があります。以下の説明は、IBM MQ テレメトリー・サービスに接続している場合のクライアントの動作について示しています。

MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離

します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

コールバック

注 : MqttCallback に対する最新の変更については、 Eclipse Paho Web サイトを参照してください。 例えば、MqttCallback は、クライアントの Paho バージョンではインターフェースとして定義され、非同期メソッドは Paho MqttAsyncClient クラスによって提供されます。

MqttCallback インターフェースには、次の 3 つのコールバック・メソッドがあります。

connectionLost(java.lang.Throwable cause)

connectionLost は、通信エラーが原因で接続が切断されたときに呼び出されます。 また、接続が確立された後にサーバー上でエラーが発生したためにサーバーが接続を切断した場合にも呼び出されます。 サーバー・エラーは、キュー・マネージャーのエラー・ログに記録されます。 サーバーはクライアントとの接続を切断し、クライアントは MqttCallback.connectionLost を呼び出します。

クライアント・アプリケーションと同じスレッドで例外としてスローされる唯一のリモート・エラーは、MqttClient.connect からの例外です。 接続の確立後にサーバーによって検出されたエラーは、 MqttCallback.connectionLost コールバック・メソッドに throwables として報告されます。

connectionLost の原因となる代表的なサーバー・エラーとして、許可エラーがあります。 例えば、トピックに関してパブリッシュが許可されていないクライアントの代わりに、テレメトリー・サーバーがそのトピックに関してパブリッシュを行おうとすることがあります。 その場合、MQCC_FAIL 条件コードをテレメトリー・サーバーに戻す原因となるものはすべて、接続切断の原因となる可能性があります。

deliveryComplete(IMqttDeliveryToken token)

deliveryComplete は、送達トークンをクライアント・アプリケーションに戻すために MQTT クライアントによって呼び出されます。1257 ページの『送達トークン』を参照してください。 送達トークンの利用により、コールバックはメソッド token.getMessage を使ってパブリッシュ・メッセージにアクセスすることができます。

deliveryComplete メソッドによって呼び出された後、アプリケーション・コールバックが MQTT クライアントに制御を戻すと、配信が完了します。 送達が完了するまで、QoS が 1 または 2 に設定されたメッセージはパーシスタンス・クラスによって保存されます。

deliveryComplete の呼び出しは、アプリケーションとパーシスタンス・クラスの間の同期点になります。 deliveryComplete メソッドが同じメッセージに対して 2 度呼び出されることはありません。

アプリケーション・コールバックが deliveryComplete から MQTT クライアントに戻ると、クライアントは QoS 1 または 2 のメッセージに対して MqttClientPersistence.remove を呼び出します。 MqttClientPersistence.remove は、パブリッシュされたメッセージのローカルで保管されたコピーを削除します。

トランザクション処理の観点から見ると、deliveryComplete の呼び出しは、送達をコミットする単一フェーズのトランザクションです。 コールバック中に処理が失敗した場合、クライアントの再始動時に MqttClientPersistence.remove が再び呼び出され、パブリッシュ・メッセージのローカル・コピーを削除します。 コールバックは再び呼び出されません。 送達されたメッセージのログを保管するためにコールバックを使用する場合、ログと MQTT クライアントを同期することはできません。 ログを確実に保管するには、MqttClientPersistence クラスでログを更新してください。

送達トークンとメッセージは、メイン・アプリケーション・スレッドおよび MQTT クライアントによって参照されます。 MQTT クライアントは送達の完了時に MqttMessage オブジェクトを間接参照し、クライアントの切断時に送達トークン・オブジェクトを間接参照します。 送達が完了した後、MqttMessage オブジェクトに対してガーベッジ・コレクションを実行することができます (クライアント・アプリケーションがそれを間接参照する場合)。 セッションが切断された後、送達トークンに対してガーベッジ・コレクションを実行することができます。

メッセージがパブリッシュされた後で、IMqttDeliveryToken 属性および MqttMessage 属性を取得することができます。 メッセージがパブリッシュされた後で MqttMessage 属性の設定を試みた場合、結果は未定義です。
MQTT クライアントは、同じ ClientIdentifier を使用して前のセッションに再接続した場合、送達確認の処理を続行します。 1253 ページの『クリーン・セッション』を参照。 MQTT クライアント・アプリケーションは、前回のセッションに対しては MqttClient.CleanSession を false に設定し、新規セッションでは false に設定する必要があります。 保留中の送達に関して、MQTT クライアントは新規セッションで新しい送達トークンとメッセージ・オブジェクトを作成します。 MqttClientPersistence クラスを使用してオブジェクトをリカバリーします。 古い送達トークンとメッセージへの参照をアプリケーション・クライアントが依然として保持している場合は、それらを間接参照してください。 前回のセッションで開始されてこのセッションで完了する送達がある場合、アプリケーション・コールバックが新規セッションで呼び出されます。
保留中の送達が完了するとき、アプリケーション・クライアントの接続後にアプリケーション・コールバックが呼び出されます。 アプリケーション・クライアントは、接続前に MqttClient.getPendingDeliveryTokens メソッドを使って保留中の送達を取り出すことができます。
最初に、クライアント・アプリケーションは、パブリッシュされるメッセージ・オブジェクトとそのペイロード・バイト配列を作成しました。 MQTT クライアントはそれらのオブジェクトを参照します。 メソッド token.getMessage で送達トークンによって戻されるメッセージ・オブジェクトは、クライアントによって作成されたメッセージ・オブジェクトと必ずしも同じではありません。 新しい MQTT クライアント・インスタンスが送達トークンを再作成する場合、MqttClientPersistence クラスは MqttMessage オブジェクトを再作成します。 一貫性を保つために、token.isCompleted が true の場合、メッセージ・オブジェクトがアプリケーション・クライアントと MqttClientPersistence クラスのどちらによって作成されたかに関係なく、token.getMessage は null を返します。

messageArrived(String topic, MqttMessage message)

messageArrived は、サブスクリプション・トピックに一致するクライアントのパブリケーションが到着したときに呼び出されます。 topic は、サブスクリプション・フィルターではなくパブリケーション・トピックです。 フィルターにワイルドカードが含まれる場合、これら 2 つは異なる可能性があります。
クライアントによって作成された複数のサブスクリプションにトピックが一致する場合、クライアントはパブリケーションの複数コピーを受信します。 サブスクライブ先でもあるトピックに対してクライアントがパブリッシュを行うと、クライアントは自分のパブリケーションのコピーを受信します。
QoS が 1 または 2 のメッセージが送信される場合、そのメッセージは、 MQTT クライアントが messageArrived を呼び出す前に MqttClientPersistence クラスによって保管されます。 messageArrived は deliveryComplete のように動作します。これはパブリケーションのために一度だけ呼び出され、 messageArrived が MQTT クライアントに戻ると、MqttClientPersistence.remove によってパブリケーションのローカル・コピーが削除されます。 messageArrived が MQTT クライアントに戻ると、 MQTT クライアントはトピックおよびメッセージへの参照を除去します。 アプリケーション・クライアントがオブジェクトへの参照を保持していない場合、トピックおよびメッセージ・オブジェクトに対してガーベッジ・コレクションが実行されます。

コールバック、スレッド、およびクライアント・アプリケーションの同期

MQTT クライアントはメイン・アプリケーション・スレッドとは別のスレッドでコールバック・メソッドを呼び出します。 コールバック用のスレッドは、クライアント・アプリケーションではなく、MQTT クライアントによって作成されます。

MQTT クライアントはコールバック・メソッドを同期します。 一度に実行されるコールバック・メソッドのインスタンスは 1 つだけです。 同期により、どのパブリケーションが送達されたかを記録するオブジェクトの更新が容易になります。 実行される MqttCallback.deliveryComplete のインスタンスは一度に 1 つなので、追加の同期を行うことなく安全に記録を更新することができます。 また、パブリケーションが一度に 1 つだけ到着する場合も同様です。 messageArrived メソッドのコードでは、同期せずにオ

ブジェクトを更新することができます。 別のスレッドで、記録 (または更新されるオブジェクト) を参照している場合には、記録またはオブジェクトを同期します。

送達トークンは、メイン・アプリケーション・スレッドとパブリケーションの送達の間の同期メカニズムを提供します。 メソッド token.waitForCompletion は、特定のパブリケーションの送達が完了するか、オプションのタイムアウトが満了するまで待機します。 次の方法で token.waitForCompletion を使用してパブリケーションを一度に 1 つずつ処理できます。

MqttCallback.deliveryComplete メソッドと同期する。 MqttCallback.deliveryComplete が MQTT クライアントに戻るときにのみ、token.waitForCompletion が再開します。 このメカニズムを使用することで、メイン・アプリケーション・スレッドでコードが実行される前に、MqttCallback.deliveryComplete で実行されるコードを同期できます。

それぞれのパブリケーションが送達されるのを待たずにパブリッシュしながら、すべてのパブリケーションが送達されたときに確認するにはどうしたらよいでしょうか? 単一スレッドでパブリッシュを行う場合には、最後に送信されるパブリケーションが最後の送達になります。

サーバーに送信される要求の同期

1252 ページの表 188 は、サーバーに要求を送信する MQTT Java クライアントのメソッドについて説明しています。 アプリケーション・クライアントで無期限のタイムアウトが設定されない限り、クライアントがサーバーを無限に待つことはありません。 クライアントがハングする場合、それはアプリケーション・プログラミングの問題、または MQTT クライアントの障害です。

表 *188.* サーバーに要求を送信するメソッドの同期の振る舞い

メソッド	同期	タイムアウト間隔
MqttClient.Connect	サーバーとの接続が確立されるのを待ちます。	デフォルトの 30 秒 (またはパラメーターによる設定値) になると、例外をスローします。
MqttClient.Disconnect	MQTT クライアントが行わなければならない処理を完了して TCP/IP セッションが切断されるのを待ちます。	
MqttClient.Subscribe	Subscribe または UnSubscribe メソッドの完了を待ちます。	
MqttClient.UnSubscribe		
MqttClient.Publish	MQTT クライアントに要求を渡した後、直ちにアプリケーション・スレッドに戻ります。	なし。
IMqttDeliveryToken.waitForCompletion	送達トークンが戻されるのを待ちます。	無期限。またはパラメーターで設定される値。

関連概念

クリーン・セッション

MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できま

す。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

クリーン・セッション

MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

`MqttClient.connect` メソッドを使用して MQTT クライアント・アプリケーションに接続すると、クライアントは、クライアント ID とサーバーのアドレスを使用して接続を識別します。 サーバーは、前回のサーバーへの接続のセッション情報が保存されているかどうかを調べます。 前回のセッションがまだ存在し、`cleanSession=true` の場合、クライアント側とサーバー側の前回のセッション情報は消去されます。 `cleanSession=false` の場合、前回のセッションが再開されます。 前回のセッションが存在しない場合、新規セッションが開始されます。

注 : IBM MQ Administrator は開かれているセッションを強制的に閉じ、すべてのセッション情報を削除します。 クライアントが `cleanSession=false` でセッションを再び開く場合、新規セッションが開始されます。

パブリケーション

デフォルトの MqttConnectOptions を使用するか、クライアントに接続する前に MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にそのクライアントで保留になっているすべてのパブリケーションの送達が削除されます。

クリーン・セッションの設定は、QoS=0 で送信されるパブリケーションには影響を与えません。 QoS=1 と QoS=2 の場合、cleanSession=true を使用するとパブリケーションが失われる可能性があります。

サブスクリプション

クライアントを接続する前に、デフォルトの MqttConnectOptions を使用するか、MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にクライアントの古いサブスクリプションはすべて削除されます。 セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続前に MqttConnectOptions.cleanSession を false に設定した場合、クライアントが作成するサブスクリプションは、接続前にクライアントに存在していたすべてのサブスクリプションに追加されます。クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

cleanSession 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。 そのデフォルト・モード cleanSession=true では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。 それに代わる cleanSession=false モードでは、サブスクリプションは永続的になります。 クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。 クライアントは、再接続時にすべての未送達パブリケーションを受信します。 接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

接続する前に cleanSession モードを設定する必要があります。このモードはセッション全体で有効です。 その設定を変更するには、クライアントを切断し、再接続する必要があります。 モードを cleanSession=false の使用から cleanSession=true に変更すると、クライアントの以前のサブスクリプション、および受信されていないパブリケーションはすべて破棄されます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション

パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

クライアント ID は、MQTT システムの管理に使用されます。 管理対象のクライアントは幾十万にもなる可能性があるので、特定のクライアントを迅速に識別する必要があります。 例えば、あるデバイスに誤動作が発生し、お客様がヘルプ・デスクに連絡していることが通知されるとします。 お客様は、デバイスを識別できる必要があり、その識別を、通常、クライアントに接続されているサーバーと相関させる必要がある場合があります。

MQTT クライアント接続を参照するとき、各接続にはクライアント ID のラベルが付いています。 この ID をデバイスおよびサーバーにマップする最善の方法を決定するには、以下の質問を自分に投げかけてください。

- 各デバイスをクライアント ID およびサーバーにマップするデータベースを保守し、使用すると便利になりますか？
- デバイスの名前は、デバイスが接続されているサーバーを識別することができますか？
- クライアント ID を物理デバイスにマップするルックアップ・テーブルが必要ですか？
- クライアント ID は、特定の装置、ユーザー、またはそのクライアントで実行しているアプリケーションを識別しますか。
- お客様が障害のあるデバイスを新しいデバイスと置き換える場合、新しいデバイスは古いデバイスと同じ ID を持っていますか？または新しい ID を割り振っていますか？(物理デバイスを変更し、同じ ID を保持する場合、突出したパブリケーションとアクティブなサブスクリプションは、自動的に新しいデバイスに転送されます。)

クライアント ID が固有であることを確認するためのシステムも必要です。また、クライアント上で ID を設定するための信頼性の高いプロセスを持つ必要があります。 クライアント・デバイスが「ブラック・ボックス」の場合、ユーザー・インターフェースを使用せずに、クライアント ID を使用してデバイスを製作することも、デバイスをアクティブ化する前にそのデバイスを構成するソフトウェア・インストールおよび構成プロセスを有することもできます。

ID を短い固有のもののままにするために、48 ビットのデバイス MAC アドレスからクライアント ID を作成することができます。 転送サイズが重大な問題ではない場合は、残りの 17 バイトを使用してアドレスを管理しやすくすることができます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

送達トークン

クライアントがトピックにでパブリッシュするときに、新しい送達トークンが作成されます。 送達トークンは、パブリケーションの送達をモニターしたり、送達が完了するまでクライアント・アプリケーションをブロックしたりするために使用します。

トークンは MqttDeliveryToken オブジェクトです。 これは MqttTopic.publish() メソッドを呼び出すことによって作成されます。 クライアント・セッションが切断され、送達が完了するまで MQTT クライアントによって保存されます。

トークンは通常、送達が完了しているかどうかを調べるために使用します。 戻されるトークンを使用して token.waitForCompletion を呼び出すことにより、送達が完了するまでクライアント・アプリケーションをブロックします。 または、MqttCallBack ハンドラーを提供します。 パブリケーションの送達の一部として予期するすべての確認応答を MQTT クライアントが受信すると、クライアントは MqttCallBack.deliveryComplete を呼び出して送達トークンをパラメーターとして渡します。

送達が完了するまで、戻される送達トークンを使用して token.getMessage を呼び出すことにより、パブリケーションを調べることができます。

完了送達

送達の完了は非同期で、パブリケーションに関連付けられているサービス品質によって異なります。

最高 1 回

QoS=0

送達は、MqttTopic.publish からの戻り時に即時に完了します。
MqttCallback.deliveryComplete が即時に呼び出されます。

最低 1 回

QoS=1

送達は、パブリケーションに対する確認応答をキュー・マネージャーから受信したときに完了します。 確認応答を受信すると、MqttCallback.deliveryComplete が呼び出されます。 通信が遅かったり不安定であったりする場合は、MqttCallback.deliveryComplete が呼び出される前にメッセージが複数回送達される可能性があります。

正確に 1 回

QoS=2

パブリケーションがサブスクライバーにパブリッシュされたという完了メッセージをクライアントが受け取ると、送達が完了します。 パブリケーション・メッセージを受信した後すぐに MqttCallback.deliveryComplete が呼び出されます。 完了メッセージを待ちません。

まれに、クライアント・アプリケーションが MqttCallback.deliveryComplete から MQTT クライアントに正常に戻らないことがあります。 送達が完了したことは、MqttCallback.deliveryComplete が呼び出されたことで分かります。 クライアントが同じセッションを再始動する場合、MqttCallback.deliveryComplete は再び呼び出されません。

未完了送達

送達が完了せずにクライアント・セッションが切断された場合、クライアントを再接続して送達を完了することができます。 メッセージの送達は、MqttConnectionOptions 属性が false に設定されたセッションでメッセージがパブリッシュされた場合のみ完了できます。

同じクライアント ID およびサーバー・アドレスを使ってクライアントを作成してから接続します。 その際、cleanSession MqttConnectionOptions 属性は再び false に設定します。 cleanSession を true に設定すると、保留中の送達トークンが廃棄されてしまいます。

MqttClient.getPendingDeliveryTokens を呼び出すことにより、保留中の送達があるかどうかを調べることができます。 MqttClient.getPendingDeliveryTokens はクライアントに接続する前に呼び出すことができます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

遺言パブリケーション

MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

遺言用のトピックを作成します。 `MQTTManagement/Connections/server URI/client identifer/Lost` のようなトピックを作成することができます。

`MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` メソッドを使用して、「遺言」をセットアップします。

`lastWillPayload` メッセージにタイム・スタンプを作成することを検討します。 クライアントおよび接続環境の識別に役立つその他のクライアント情報を組み込みます。 `MqttClient` コンストラクターに `MqttConnectionOptions` オブジェクトを渡します。

`lastWillQos` を 1 または 2 に設定し、IBM MQ のメッセージを持続にして、確実に送信されるようにします。 最後に失われた接続情報を保存するには、`lastWillRetained` を `true` に設定します。

接続が予期せず終了した場合に、「遺言」パブリケーションがサブスクライバーに送信されます。 これは、クライアントが `MqttClient.disconnect` メソッドを呼び出さずに接続が終了した場合に送信されます。

接続をモニターするには、「遺言」パブリケーションを他のパブリケーションで補完して、接続およびプログラムされた切断を記録します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション

MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション

パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質

MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

MQTT におけるメッセージ持続性には、メッセージが転送される方法と、それが持続メッセージとして IBM MQ のキューに入れられるかどうかの 2 つの面があります。

1. MQTT クライアントは、メッセージ持続性とサービス品質とを結合させます。 メッセージのために選択するサービス品質に応じて、メッセージは永続的になります。 メッセージ永続性は、必要なサービス品質を実装するために必要です。

   "at most once", QoS=0 を指定すると、クライアントは、メッセージがパブリッシュされるとすぐにそれを破棄します。 メッセージのアップストリーム処理に障害が生じた場合、メッセージは再送信されません。 クライアントがアクティブのままであっても、メッセージは再送信されません。 QoS=0 のメッセージの動作は、IBM MQ の高速非持続メッセージと同じです。

   メッセージは、QoS が 1 または 2 に指定されてクライアントによってパブリッシュされる場合、持続的になります。 メッセージはローカルに保管され、"at least once", QoS=1、または "exactly once", QoS=2 送達を保証する必要がなくなったときにのみ、クライアントから廃棄されます。

2. メッセージは、QoS が 1 または 2 にマーク付けされた場合、持続メッセージとして IBM MQ のキューに入れられます。 QoS=0 としてマーク付けされた場合は、非持続メッセージとして IBM MQ のキューに入れられます。 IBM MQ では、メッセージ・チャネルの NPMSPEED 属性が FAST に設定されていない限り、非持続メッセージはキュー・マネージャー間で "1 回だけ"転送されます。

永続パブリケーションはクライアント・アプリケーションによって受け取られるまでクライアント上に保管されます。 QoS=2 の場合、アプリケーションのコールバックから制御が戻ったときに、パブリケーションはクライアントから廃棄されます。 QoS=1 の場合、障害が生じたときは、アプリケーションがパブリケーションを再び受け取ることがあります。 QoS=0 の場合、コールバックがパブリケーションを受け取る回数は 1 回以内となります。 パブリケーション時に障害が生じている場合やクライアントが切断されている場合には、パブリケーションを受け取らないことがあります。

トピックにサブスクライブするときに、サブスクライバーがメッセージを受け取る際の QoS を、その持続性の能力に合うように低くすることができます。 より大きな QoS で作成されたパブリケーションは、サブスクライバーが要求したものの中で最高の QoS で送信されます。

メッセージの保管

小さな装置にデータ・ストレージを実装する方法は、さまざまに大きく異なります。 MQTT クライアントが管理するストレージに持続メッセージを一時保存するモデルは、遅すぎたりストレージ要求が大きすぎたりすることがあります。 モバイル・デバイスでは、モバイル・オペレーティング・システムによって MQTT メッセージに適したストレージ・サービスが提供される場合があります。

小型の装置の制約に適合するように柔軟性を提供するために、MQTT クライアントには 2 つの持続性インターフェースがあります。 インターフェースは永続メッセージの保管に関連した操作を定義します。 それらのインターフェースは、MQTT client for Java の API ドキュメンテーションで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、MQTT クライアント・プログラミング・リファレンスを参照してください。装置に適合するようにインターフェースを実装できます。 Java SE で実行される MQTT クライアントには、永続メッセージをファイル・システムに保管するインターフェースのデフォルト実装があります。 これは `java.io` パッケージを使用します。

パーシスタンス (持続性) クラス

`MqttClientPersistence`
`MqttClient` コンストラクターのパラメーターとして、 `MqttClientPersistence` の実装のインスタンスを MQTT クライアントに渡します。 `MqttClientPersistence` パラメーターを `MqttClient` コンストラクターから省略した場合、MQTT クライアントはクラス `MqttDefaultFilePersistence` を使用して持続メッセージを保管します。

`MqttPersistable`
`MqttClientPersistence` は、ストレージ・キーを使用して `MqttPersistable` オブジェクトを取得および配置します。 `MqttDefaultFilePersistence` を使用していない場合は、`MqttPersistable` の実装および `MqttClientPersistence` の実装を提供する必要があります。

`MqttDefaultFilePersistence`

MQTT クライアントは、`MqttDefaultFilePersistence` クラスを提供します。 クライアント・アプリケーションで `MqttDefaultFilePersistence` をインスタンス化する場合、永続メッセージを保管するディレクトリーを `MqttDefaultFilePersistence` コンストラクターのパラメーターとして提供できます。

あるいは、MQTT クライアントは `MqttDefaultFilePersistence` のインスタンスを生成し、ファイルを以下のデフォルト・ディレクトリーに配置することもできます。

`client identifier` `-tcp` *`hostname portnumber`*

以下の文字がディレクトリー名ストリングから削除されます。

`"\", "\\", "/", ":"と" "`

このディレクトリーへのパスは、システム・プロパティー `rcp.data` の値です。`rcp.data` が設定されていない場合は、パスはシステム・プロパティー `usr.data` の値です。ここで、

- `rcp.data` は、OSGi または Eclipse Rich Client Platform (RCP) のインストールに関連したプロパティーです。
- `usr.data` は、アプリケーションを開始した Java コマンドが起動されたディレクトリーです。

関連概念
MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプション

も含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

パブリケーション

パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MqttMessage には、ペイロードとしてのバイト配列があります。 メッセージはできる限り小さくしてください。 MQTT protocol で許可されるメッセージの最大長は 250 MB です。

通常、MQTT クライアント・プログラムは java.lang.String または java.lang.StringBuffer を使用してメッセージ・コンテンツを扱います。 MqttMessage クラスには、ペイロードをストリングに変換するための便利な toString メソッドがあります。 java.lang.String または java.lang.StringBuffer からバイト配列のペイロードを作成するには、getBytes メソッドを使用します。

getBytes メソッドは、ストリングをプラットフォームのデフォルト文字セットに変換します。 デフォルト文字セットは、通常 UTF-8 です。 テキストだけを含む MQTT パブリケーションは、通常 UTF-8 でエン

コードされています。 デフォルト文字セットをオーバーライドするには、メソッド getBytes("UTF8") を使用します。

IBM MQ では、MQTT パブリケーションを jms-bytes メッセージとして受信します。 メッセージには、<mqtt> フォルダーおよび <mqps> フォルダーのある MQRFH2 フォルダーが含まれます。 <mqtt> フォルダーには、clientId、msgId および qos が含まれますが、このコンテンツは将来的に変更される可能性があります。

MqttMessage には 3 つの追加的な属性 (サービス品質、保持されるかどうか、および重複であるかどうか) があります。 重複フラグは、サービス品質が「最低 1 回」または「正確に 1 回」のときにだけ設定されます。 このメッセージが既に送信されたが、MQTT クライアントによって素早く認知されない場合、重複属性が true に設定された状態でそのメッセージが再送信されます。

パブリッシュ

MQTT クライアント・アプリケーションにパブリケーションを作成するには、MqttMessage を作成します。 そのペイロード、サービスの品質、および保持されるかどうかを設定して、MqttTopic.publish(MqttMessage message) メソッドを呼び出します。MqttDeliveryToken が返され、パブリケーションの完了は非同期です。

あるいは、パブリケーション作成時に、MQTT クライアントは MqttTopic.publish(byte [] payload, int qos, boolean retained) メソッドのパラメーターから一時メッセージ・オブジェクトを作成することができます。

パブリケーションのサービス品質が「最低 1 回」または「正確に 1 回」、つまり QoS=1 または QoS=2 である場合、MQTT クライアントは MqttClientPersistence インターフェースを呼び出します。 送達トークンをアプリケーションに戻す前に、メッセージを保管するために MqttClientPersistence を呼び出します。

MqttDeliveryToken.waitForCompletion メソッドを使用することで、アプリケーションは、メッセージがサーバーに送達されるまでブロックするよう選択できます。 あるいは、アプリケーションはブロックなしで続行することもできます。 パブリケーションがブロッキングなしで配信されるかどうかを確認する場合は、MqttCallback を実装するコールバック・クラスのインスタンスを MQTT クライアントに登録します。 MQTT クライアントは、パブリケーションが送達された直後に MqttCallback.deliveryComplete メソッドを呼び出します。 サービス品質に応じて、QoS=0 であれば送達はほぼ即時に行われ、QoS=2 であれば幾らかの時間がかかります。

送達が完了したかどうかポーリングするには、MqttDeliveryToken.isComplete メソッドを使用します。 MqttDeliveryToken.isComplete の値が false である間は、MqttDeliveryToken.getMessage を呼び出してメッセージの内容を取得できます。 MqttDeliveryToken.isComplete を呼び出した結果が true の場合、メッセージは既に廃棄されているので、MqttDeliveryToken.getMessage を呼び出すと NULL ポインター例外がスローされます。 MqttDeliveryToken.getMessage と MqttDeliveryToken.isComplete とが同期するような機能は組み込まれていません。

保留中の送達トークンをすべて受け取る前にクライアントが切断した場合、クライアントの新しいインスタンスは、接続の前に保留中の送達トークンを照会することができます。 クライアントが接続するまで新しい送達は完了しないので、MqttDeliveryToken.getMessage を安全に呼び出すことができます。 どのパブリケーションがまだ送達されていないかを検出するには、MqttDeliveryToken.getMessage メソッドを使用します。 MqttConnectOptions.cleanSession をデフォルト値 true に設定して接続すると、保留中の送達トークンは廃棄されます。

サブスクライブ

キュー・マネージャーには、MQTT サブスクライバーに送るパブリケーションを作成する役割があります。 キュー・マネージャーは、MQTT クライアントによって作成されたサブスクリプション内のトピック・フィルターが、パブリケーション内のトピック・ストリングと一致するかどうかを確認します。 この一致は完全一致にすることも、または一致にワイルドカードを含めることもできます。 パブリケーションがキュー・マネージャーによってサブスクライバーに転送される前に、キュー・マネージャーは、パブリケーシ

ョンに関連付けられたトピック属性を調べます。 ワイルドカード文字を含むトピック・ストリングを使用したサブスクライブ操作で説明されている検索手順に従い、管理トピック・オブジェクトによってサブスクライブ権限がユーザーに与えられているかどうかを識別します。

MQTT クライアントは、「最低 1 回」のサービス品質でパブリケーションを受け取ると、`MqttCallback.messageArrived` メソッドを呼び出してパブリケーションを処理します。 パブリケーションのサービス品質が「正確に 1 回」つまり QoS=2 である場合、MQTT クライアントは、メッセージを受け取ると `MqttClientPersistence` インターフェースを呼び出してそれを保管します。 その後、`MqttCallback.messageArrived` を呼び出します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントによって提供されるサービス品質

MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

パブリケーションのサービス品質は、`MqttMessage` の属性です。 これはメソッド `MqttMessage.setQos` によって設定されます。

メソッド `MqttClient.subscribe` は、トピックのクライアントに送信されるパブリケーションに適用されるサービス品質を下げることができます。 サブスクライバーに転送されるパブリケーションのサービス品質は、パブリケーションのサービス品質と異なる場合があります。 2 つのうちの低い方の値がパブリケーションの転送に使用されます。

最高 1 回
`QoS=0`

メッセージは最高 1 回送信されるか、まったく送信されません。 ネットワークでのメッセージの送信は確認応答されません。
メッセージは保管されません。 クライアントが切断されるか、サーバーで障害が発生すると、メッセージが失われる可能性があります。
`QoS=0` が最も速い転送モードです。 これは「応答不要送信」と呼ばれることがあります。
MQTT protocol は、`QoS=0` でパブリケーションをクライアントに転送するためにサーバーを必要としません。 サーバーがパブリケーションを受信したときにクライアントが切断される場合、サーバーによってはパブリケーションが廃棄される可能性があります。 テレメトリー (MQXR) サービスは、`QoS=0` で送信されるメッセージを廃棄しません。 そのメッセージは非持続メッセージとして保管され、キュー・マネージャーが停止する場合にのみ廃棄されます。

最低 1 回
`QoS=1`

`QoS=1` はデフォルトの転送モードです。
メッセージは常に、最低 1 回送信されます。 送信側が確認応答を受信しない場合、確認応答が受信されるまで、`DUP` フラグが設定されたメッセージが再送信されます。 結果として、受信側に同じメッセージが複数回送信されて、受信側がそれを複数回処理することになる可能性があります。
メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。
メッセージは、受信側によって処理された後、受信側から削除されます。 受信側がブローカーの場合、メッセージはそのサブスクライバーにパブリッシュされます。 受信側がクライアントの場合、メッセージはサブスクライバー・アプリケーションに送信されます。 メッセージを削除した後、受信側は送信側に確認応答を送信します。
送信側が受信側から確認応答を受信した後、そのメッセージは送信側から削除されます。

正確に 1 回
`QoS=2`

メッセージは常に、正確に 1 回送信されます。
メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。
`QoS=2` は最も安全ですが、最も遅い転送モードです。 メッセージを送信側から削除する前に、送信側と受信側の間で少なくとも 2 組の伝送を取ります。 最初の伝送の後に受信側のメッセージを処理することができます。
1 組目の伝送で、送信側はメッセージを伝送し、メッセージが保管されたという確認応答を受信側から取得します。 送信側が確認応答を受信しない場合、確認応答が受信されるまで、`DUP` フラグが設定されたメッセージが再送信されます。

2 組目の伝送で、送信側はメッセージ "PUBREL" の処理を完了できることを受信側に伝えます。 送信側が "PUBREL" メッセージの確認応答を受信しない場合、確認応答が受信されるまで "PUBREL" メッセージが再送信されます。 送信側は、 "PUBREL" メッセージに対する確認応答を受信すると、保存していたメッセージを削除します。
受信側は、メッセージを再処理しない場合には、最初または 2 番目のフェーズでメッセージを処理することができます。 受信側がブローカーの場合、サブスクライバーにメッセージをパブリッシュします。 受信側がクライアントの場合、サブスクライバー・アプリケーションにメッセージを送信します。 受信側は、メッセージの処理を完了したという完了メッセージを送信側に送り戻します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター

トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

保存パブリケーションおよび MQTT クライアント

トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

`MqttMessage.setRetained` メソッドを使用して、トピックのパブリケーションが保存されるかどうかを指定します。

保存パブリケーションを作成または更新するときには、1 または 2 の QoS を使用してパブリケーションを送信します。 QoS を 0 の値で送信すると、IBM MQ は非永続保存パブリケーションを作成します。 キュー・マネージャーが停止する場合、パブリケーションは保存されません。

保存パブリケーションを持つトピックに非保存パブリケーションをパブリッシュした場合、保存パブリケーションは影響を受けません。 現在のサブスクライバーは新しいパブリケーションを受信します。 新しいサブスクライバーは、まず保存パブリケーションを受信し、次に新しいパブリケーションがあればそれを受信します。

保存パブリケーションを使用して、最新の測定値を記録することができます。 トピックへの新規サブスクライバーは、即時に最新の測定値を受信します。 サブスクライバーが最後にパブリケーション・トピックにサブスクライブした後に新しい測定が行われていない場合、サブスクライバーが再度サブスクライブすると、そのトピックの最新の保存パブリケーションを再び受信します。

保存パブリケーションを削除する方法として、次の 2 つのオプションがあります。

- **CLEAR TOPICSTR** MQSC コマンドを実行します。
- 長さがゼロの保存パブリケーションを作成します。 MQTT 3.1.1 仕様で指定されているように、長さがゼロの保存メッセージがトピックにパブリッシュされると、そのトピックの保存メッセージはクリアされます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

サブスクリプション

トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MqttClient.subscribe メソッドを使用してサブスクリプションを作成します。 その際に、1 つ以上のトピック・フィルターおよびサービス品質パラメーターを渡します。 サービス品質パラメーターは、サブスクライバーがメッセージを受け取るために使用する用意のある最大のサービス品質を設定します。 このクライアントに送るメッセージを、それよりも高いサービス品質で送信することはできません。 サービス品質は、メッセージがパブリッシュされたときの元の値およびサブスクリプションに指定されたレベルよりも低く設定されます。 メッセージを受け取るためのデフォルトのサービス品質は、QoS=1 つまり「最低 1 回」です。

サブスクリプション要求そのものは、QoS=1 で送信されます。

MQTT クライアントが MqttCallback.messageArrived メソッドを呼び出すときに、パブリケーションはサブスクライバーによって受け取られます。 messageArrived メソッドは、メッセージをサブスクライバーにパブリッシュするときに使用されたトピック・ストリングも渡します。

MqttClient.unsubscribe メソッドを使用して、サブスクリプション (またはサブスクリプションのセット) を削除できます。

IBM MQ コマンドは、サブスクリプションを削除できます。 IBM MQ Explorer を使用するか、**runmqsc** または PCF コマンドを使用して、サブスクリプションをリストします。 すべての MQTT クライアントのサブスクリプションの名前が示されます。 これらには、次の形式の名前が付けられます。
ClientIdentifier:*Topic name*

クライアントを接続する前に、デフォルトの MqttConnectOptions を使用するか、MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にクライアントの

古いサブスクリプションはすべて削除されます。 セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続前に MqttConnectOptions.cleanSession を false に設定した場合、クライアントが作成するサブスクリプションは、接続前にクライアントに存在していたすべてのサブスクリプションに追加されます。 クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

cleanSession 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。 そのデフォルト・モード cleanSession=true では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。 それに代わる cleanSession=false モードでは、サブスクリプションは永続的になります。 クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。 クライアントは、再接続時にすべての未送達パブリケーションを受信します。 接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

接続する前に cleanSession モードを設定する必要があります。このモードはセッション全体で有効です。 その設定を変更するには、クライアントを切断し、再接続する必要があります。 モードを cleanSession=false の使用から cleanSession=true に変更すると、クライアントの以前のサブスクリプション、および受信されていないパブリケーションはすべて破棄されます。

アクティブなサブスクリプションに一致するパブリケーションは、パブリッシュされるとすぐにクライアントに送信されます。 クライアントが切断されている場合は、そのクライアントが同じサーバーに同じクライアント ID を使用して再接続すれば、 そして MqttConnectOptions.cleanSession が false に設定されていれば送信されます。

特定のクライアントのサブスクリプションは、クライアント ID によって識別されます。 クライアントを別のクライアント装置から同じサーバーに再接続して、同じサブスクリプションの処理を続行し、未送達のパブリケーションを受け取ることもできます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション

MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション

MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター

トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM MQ のトピック・ストリングの構文とほぼ同じです。

トピック・ストリングは、パブリケーションをサブスクライバーに送信するために使われます。 トピック・ストリングを作成するには、メソッド `MqttClient.getTopic(java.lang.String topicString)` を使用します。

トピック・フィルターは、トピックにサブスクライブし、パブリケーションを受信するために使われます。 トピック・フィルターにはワイルドカードを含めることができます。 ワイルドカードを使用すると、複数のトピックにサブスクライブできます。 トピック・フィルターを作成するには、例えば `MqttClient.subscribe(java.lang.String topicFilter)` などのサブスクリプション・メソッドを使用します。

トピック・ストリング

IBM MQ トピック・ストリングの構文については、「トピック・ストリング」で説明されています。 MQTT トピック・ストリングの構文は、 MQTT client for Java の API 資料の `MqttClient` クラスで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、 MQTT クライアント・プログラミング・リファレンスを参照してください。

それぞれの種類のトピック・ストリングの構文は、ほとんど同じです。 ただし、小さな違いが 4 つあります。

1. MQTT クライアントによって IBM MQ に送信されるトピック・ストリングは、キュー・マネージャー名の規則に従う必要があります。
2. 最大長が異なります。 IBM MQ トピック・ストリングは 10,240 文字に制限されます。 MQTT クライアントは、最大 65535 バイトのトピック・ストリングを作成することができます。
3. MQTT クライアントによって作成されるトピック・ストリングには、ヌル文字を含めることができません。
4. IBM Integration Bus では、ヌルのトピック・レベル `'...//...'` は無効です。 IBM MQ ではヌルのトピック・レベルがサポートされています。

IBM MQ パブリッシュ/サブスクライブとは異なり、`mqttv3` プロトコルには管理トピック・オブジェクトの概念がありません。 トピック・オブジェクトおよびトピック・ストリングからトピック・ストリングを構成することはできません。 ただし、トピック・ストリングは IBM MQ の管理トピックにマップされます。 管理トピックに関連付けられているアクセス制御は、パブリケーションがトピックにパブリッシュされるか、廃棄されるかを決定します。 サブスクライバーへの転送時にパブリケーションに適用される属性は、管理トピックの属性の影響を受けます。

トピック・フィルター

IBM MQ トピック・フィルターの構文については、「トピック・ベースのワイルドカード・スキーム」で説明されています。 MQTT クライアントで構成できるトピック・フィルターの構文は、 MQTT client for Java の API 資料の `MqttClient` クラスで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、 MQTT クライアント・プログラミング・リファレンスを参照してください。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよびテレメトリー (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、"at least once" と "exactly once" の配信、およびパブリケーションの "exactly once" 受信を保証するために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID
クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション・メッセージは、"at least once" または "exactly once" のサービス品質で送信されると、永続化されます。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントは、パブリケーションを IBM MQ および MQTT クライアントに提供するために、「at most once」、「at least once」、「exactly once」という 3 段階のサービス品質を提供します。 MQTT クライアントが IBM MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
トピックには、保存パブリケーションが 1 つだけあります。 保存パブリケーションを持つトピックへのサブスクリプションを作成すると、パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られま

す。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

IBM MQ を使用した Microsoft Windows Communication Foundation アプリケーションの開発

IBM MQ 用の Microsoft Windows Communication Foundation (WCF) カスタム・チャネルは、WCF クライアントとサービスの間でメッセージを送受信します。

関連概念

1272 ページの『.NET を使用する WCF 用の IBM MQ カスタム・チャネルの概要』
IBM MQ カスタム・チャネルは、Microsoft Windows Communication Foundation (WCF) 統一プログラミング・モデルを使用するトランスポート・チャネルです。

1276 ページの『WCF 用の IBM MQ カスタム・チャネルの使用』
Windows Communication Foundation (WCF) 用の IBM MQ カスタム・チャネルを使用するプログラマーが利用できる情報の概要。

1295 ページの『WCF サンプルの使用』
Windows Communication Foundation (WCF) サンプルでは、IBM MQ カスタム・チャネルの使用方法の簡単な例をいくつか紹介します。

FFST: WCF XMS 第 1 障害サポート・テクノロジー

関連タスク

IBM MQ の WCF カスタム・チャネルのトレース

IBM MQ の問題に関する WCF カスタム・チャネルのトラブルシューティング

.NET を使用する WCF 用の IBM MQ カスタム・チャネルの概要

IBM MQ カスタム・チャネルは、Microsoft Windows Communication Foundation (WCF) 統一プログラミング・モデルを使用するトランスポート・チャネルです。

Microsoft.NET 3 で導入された Microsoft Windows Communication Foundation フレームワークにより、.NET アプリケーションおよびサービスをそれらの接続に使われているトランスポートやプロトコルから独立して開発することができ、代わりのトランスポートまたは構成を、サービスまたはアプリケーションが配置された環境に応じて使用することができるようになります。

接続は、以下の必要な組み合わせを含むチャネル・スタックを作成することで、 WCF によって実行時に管理されます。

- プロトコル要素: オプションの一連の要素。 WS-* 標準などのプロトコルをサポートするように 1 つ以上の要素を追加することも、 要素を追加しないこともできます。
- メッセージ・エンコーダー: ワイヤー形式でメッセージを直列化するように制御する、スタック内の必須要素。
- トランスポート・チャネル: 直列化したメッセージをそのエンドポイントにトランスポートさせる、スタック内の必須要素。

IBM MQ カスタム・チャネルは、トランスポート・チャネルであるため、WCF カスタム・バインディングを使用して、アプリケーションでの必要に応じてメッセージ・エンコーダーおよびオプションのプロトコルとペアにする必要があります。 このように、WCF を使用するように開発されたアプリケーションは、IBM MQ のカスタム・チャネルを使用して、Microsoft が提供する組み込みトランスポートを使用するのと同じ方法でデータを送受信することができます。これにより、IBM MQ の非同期、スケーラブル、および高信頼性メッセージング機能との簡単な統合が可能になります。 サポートされる機能の完全なリストについては、1276 ページの『WCF カスタム・チャネルのフィーチャーと機能』を参照してください。

WCF 用の IBM MQ カスタム・チャネルはいつ、どういう理由で使用するのか

IBM MQ カスタム・チャネルを使用することにより、Microsoft が提供する組み込みトランスポートと同じ方法で WCF クライアントとサービスの間でデータを送受信できます。これによって、アプリケーションは、WCF 統一プログラミング・モデル内の IBM MQ のフィーチャーにアクセスできます。

WCF 用の IBM MQ カスタム・チャネル用の標準的な使用パターン・シナリオでは、ネイティブの IBM MQ メッセージを伝送するための非 SOAP インターフェースとして使用されます。

非 SOAP/非 JMS メッセージ (Pure MQMessage) フォーマットを使用して伝送されるメッセージ

ネイティブ IBM MQ メッセージを伝送するための非 SOAP インターフェースとして WCF 用の IBM MQ カスタム・チャネルを使用する場合、メッセージは IBM MQ の非 SOAP/ 非 JMS メッセージ (Pure MQMessage) フォーマットを使用して伝送されます。

WCF ユーザーはサービスを開始できます。 言い換えれば、サービス・ユーザーは MQMessage を使用して IBM MQ キューにメッセージを送信できることになります。 アプリケーションは、MQMD フィールドおよびペイロードの取得と設定が可能です。 メッセージが IBM MQ キューで使用可能な場合、このメッセージは、 AIX、 Linux、 Windows、または z/OS で実行されている任意の WCF サービス・アプリケーションまたは非 WCF アプリケーション (C アプリケーションや Java アプリケーションなど) によって処理できます。

WCF 用の IBM MQ カスタム・チャネルのソフトウェア要件

このトピックでは、WCF 用の IBM MQ カスタム・チャネルのソフトウェア要件の概要について説明します。 WCF 用の IBM MQ カスタム・チャネルは、IBM WebSphere MQ 7.0 以降のキュー・マネージャーにのみ接続可能です。

ランタイム環境の要件

- Microsoft.NET Framework v4.7.2 以上がホスト・マシンにインストールされている必要があります。
- *Java and .NET Messaging and Web Services* が、IBM MQ インストーラーによる作業の一環としてデフォルトでインストールされます。 このコンポーネントにより、カスタム・チャネルに必要な .NET アセンブリーがグローバル・アセンブリー・キャッシュにインストールされます。

注 : IBM MQ をインストールする前に Microsoft .NET Framework V4.7.2 以降がインストールされていない場合、 IBM MQ 製品のインストールはエラーなしで続行されますが、 IBM MQ classes for .NET は使用できません。 IBM MQ をインストールした後に .NET Framework がインストールされた場合、 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd スクリプトを実行することで、IBM MQ.NET アセンブリーを登録する必要があります。ここで *WMQInstallDir* は、IBM MQ がインストールされているディレクトリーです。 このスクリプトにより、必要なアセンブリーがグローバル・アセンブリー・キャッシュ (GAC) にインストールされます。 実行するアクションを記録する一連の amqi*.log ファイルは、%TEMP% ディレクトリーで作成されます。 .NET が以前のバージョン (例えば、.NET V3.5) から V4.7.2 以上にアップグレードされている場合は、 amqiRegisterdotNet.cmd スクリプトを再実行する必要はありません。

開発環境の要件

- Microsoft Visual Studio 2015 または Windows Software Development Kit for .NET 4.7.2 以降。
- サンプル・ソリューション・ファイルを作成するには、Microsoft.NET Framework V4.7.2 以上をホスト・マシンにインストールする必要があります。

WCF 用の IBM MQ カスタム・チャネル: 何がインストールされるのか

IBM MQ カスタム・チャネルは、Microsoft Windows Communication Foundation (WCF) 統一プログラミング・モデルを使用するトランスポート・チャネルです。 カスタム・チャネルは、 デフォルトでインストールの一環としてインストールされます。

WCF 用の IBM MQ カスタム・チャネル

カスタム・チャネルおよびその依存関係は、デフォルトでインストールされる Java and .NET Messaging and Web Services コンポーネント内に含まれています。 IBM MQ を IBM MQ 8.0 より前のバージョンからアップグレードするとき、Java and .NET Messaging and Web Services コンポ

ーネントが以前のインストール済み環境に以前、インストールされていた場合は、デフォルトで WCF 用の IBM MQ カスタム・チャネルが更新によってインストールされます。

`.NET Messaging and Web Services` コンポーネントには、`IBM.XMS.WCF.dll` ファイルと `IBM.WMQ.WCF.dll` ファイルが含まれます。これらのファイルは、WCF インターフェース・クラスを含むメイン・カスタム・チャネル・アセンブリーです。 これらのファイルはグローバル・アセンブリー・キャッシュ (GAC) でインストールされ、以下のディレクトリーでも使用できるようになります。 *MQ_INSTALLATION_PATH* `\bin`。ここで、*MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーです。

次の表は、カスタム・チャネルを使用するために必要なキー・クラスを要約したものです。

表 *189.* カスタム・チャネルを使用するために必要なキー・クラス

	SOAP/JMS インターフェース (既存)	**非 SOAP/非 JMS インターフェース (IBM MQ 8.0 から)**
Custom Channel Assembly	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Transport Binding Name	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Transport Binding Importer:	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Transport Binding Config	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Samples(Oneway)	SimpleOneWay_Client、SimpleOneWay_Service	MQMessaging_OneWay_Client、MQMessaging_OneWay_Service
Samples(RequestReply)	SimpleRequestReply_Client、SimpleRequestReply_Service	MQMessaging_RequestReply_Client、MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll は、SOAP/JMS インターフェースと非 SOAP/非 JMS インターフェースの両方をサポートします。 新規アプリケーションの開発では、両方のインターフェースをサポートする IBM.WMQ.WCF アセンブリーを使用することをお勧めします。

MQSTR 形式のメッセージの送信

要求メッセージのタイプが MQSTR の場合、応答メッセージを MQSTR 形式で送信することを選択できます。

応答メッセージの形式を変更するには、追加の URI パラメーター **replyMessageFormat** を使用する必要があります。 サポートされている値は、以下のとおりです。

""

" " がデフォルト値です。

応答メッセージはバイト (MQMFT_NONE) 形式になります。 以下に例を示します。

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

応答メッセージは MQSTR (MQMFT_STRING) 形式になります。 以下に例を示します。

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

注 :

1. **replyMessageFormat** の値は、大/小文字を区別しません。
2. " " と *MQSTR* 以外の値を使用すると、無効なパラメーター値の例外が発生します。

IBM MQ カスタム・チャネルのサンプル

これらのサンプルでは、WCF 用の IBM MQ カスタム・チャネルの使用方法についていくつかの簡単な例を紹介します。 サンプルとその関連ファイルは、*MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf ディレクトリーにあります。ここで、*MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。 IBM MQ カスタム・チャネルのサンプルについて詳しくは、1295 ページの『WCF サンプルの使用』を参照してください。

svcutil.exe.config

svcutil.exe.config は、Microsoft WCF svcutil クライアント・プロキシー生成ツールを有効化してカスタム・チャネルを認識するために必要な構成設定の例です。 svcutil.exe.config ファイルは、*MQ_INSTALLATION_PATH* \tools\wcf\docs\examples\ ディレクトリーにあります。ここで、*MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。 svcutil.exe.config の使用方法の詳細については、1292 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』を参照してください。

WCF のアーキテクチャー

WCF 用の IBM MQ カスタム・チャネルは、 IBM Message Service Client for .NET (XMS .NET) API の上に統合されています。

SOAP/JMS インターフェース

WCF アーキテクチャーを以下の図に示します。

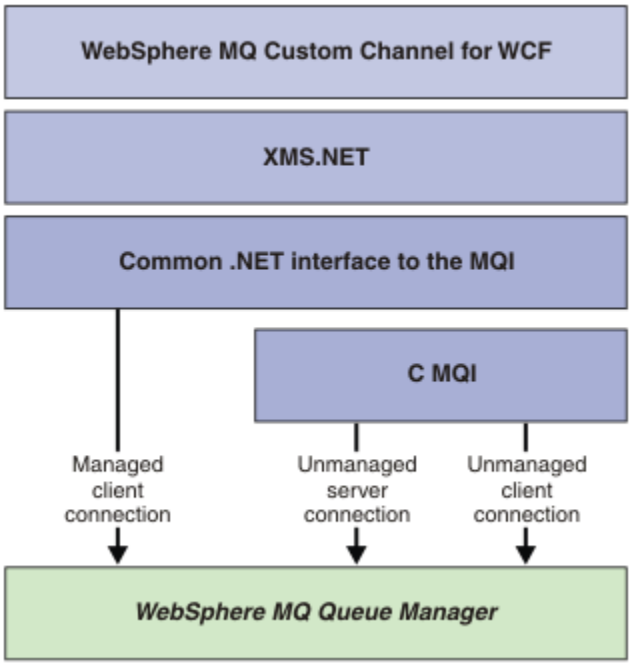


図 *149. SOAP/JMS* インターフェース対応の *WCF* アーキテクチャー

必要なコンポーネントはすべて、製品インストールでデフォルトでインストールされます。

接続は以下の 3 つです。

- 管理対象クライアント接続
- 非管理対象サーバー接続
- 非管理対象クライアント接続

これらの接続について詳しくは、1282 ページの『WCF 接続オプション』を参照してください。

非 SOAP/非 JMS インターフェース

WCF 用の IBM MQ カスタム・チャネルは、SOAP/JMS インターフェース (IBM WebSphere MQ 7.0.1 から使用可能) と非 SOAP/非 JMS インターフェースの両方をサポートします。

WCF アーキテクチャーを以下の図に示します。

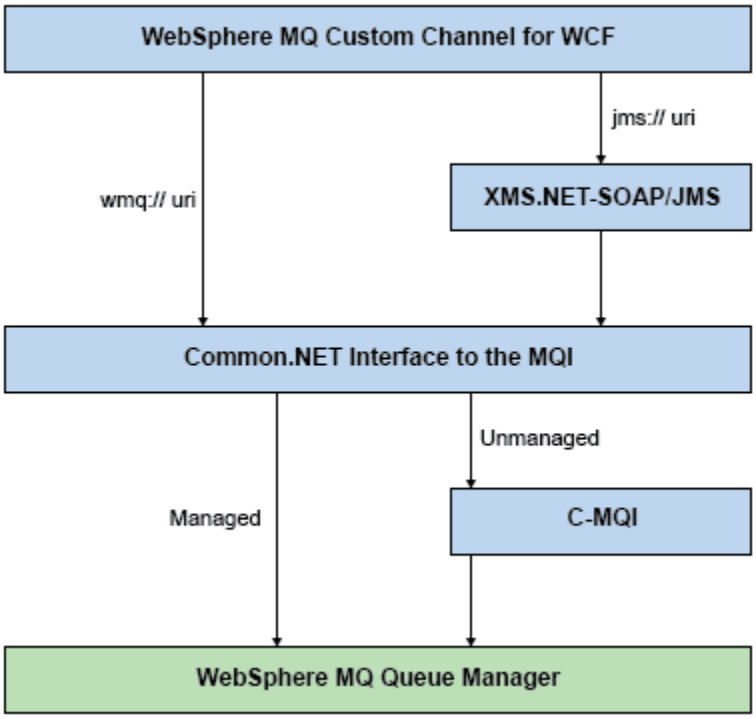


図 *150.* 非 *SOAP/*非 *JMS* インターフェース対応の *WCF* アーキテクチャー

WCF 用の IBM MQ カスタム・チャネルの使用

Windows Communication Foundation (WCF) 用の IBM MQ カスタム・チャネルを使用するプログラマーが利用できる情報の概要。

Microsoft Windows Communication Foundation は、Microsoft.NET Framework 3 の Web サービスおよびメッセージング・サポートをサポートします。 IBM MQ は、 Microsoft によって提供される組み込みチャネルと同じ方法で、 .NET Framework 3 の WCF 内でカスタム・チャネルとして使用できます。

カスタム・チャネルを介して転送されるメッセージは、 IBM MQ の SOAP over JMS 実装に従ってフォーマット設定されます。 アプリケーションは、WCF または JMS サービス・インフラストラクチャー上の WebSphere SOAP によってホストされるサービスと通信できます。

WCF カスタム・チャネルのフィーチャーと機能

WCF カスタム・チャネルのフィーチャーと機能については、以下のトピックを参照してください。

WCF カスタム・チャネルの形状

IBM MQ を Microsoft Windows Communication Foundation (WCF) カスタム・チャネル内で使用できるカスタム・チャネル形状の概要。

WCF 用の IBM MQ カスタム・チャネルは、次の 2 つのチャネル形状をサポートします。

- 片方向
- 要求/応答

WCF は、ホストされているサービス・コントラクトに応じて、 自動的にチャネル形状を選択します。

IsOneWay パラメーターのみを使用するメソッドを含むコントラクトは、 片方向チャネル形状でサービスが提供されます。例えば、次のようになります。

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

片方向と要求/応答のメソッドが混在するコントラクト、またはすべてが要求/応答のメソッドを含むコントラクトは、要求/応答チャネル形状でサービスが提供されます。 以下に例を示します。

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

注 : 片方向と要求/応答のメソッドが同じコントラクト内で混在するときには、 意図したとおりに動作するかどうかを確認する必要があります。 特に、混合環境内で作業する際に、 片方向メソッドはサービスからヌル応答を受信するまで待機するため、このことは重要です。

片方向チャネル

WCF 用の IBM MQ 片方向カスタム・チャネルは、例えば、片方向チャネル形状を使用して WCF クライアントからメッセージを送信する場合に使用されます。 チャネルは、例えば、クライアント・キュー・マネージャーから WCF サービス上のキューに送信するなど、 単一方向にのみメッセージを送信できます。

要求/応答チャネル

WCF 用の IBM MQ 要求/応答カスタム・チャネルは、例えば、双方向に非同期でメッセージを送信する場合に使用されます。非同期メッセージングには、同じクライアント・インスタンスを使用する必要があります。 チャネルは、例えば、クライアント・キュー・マネージャーから WCF サービス上のキューに送信するなど、 単一方向にメッセージを送信し、次に、WCF からクライアント・キュー・マネージャー上のキューに応答メッセージを送信できます。

WCF URI パラメーターの名前と値

SOAP/JMS インターフェースと非 SOAP/非 JMS インターフェースにおける URI パラメーターの名前と値。

SOAP/JMS インターフェース

connectionFactory
connectionFactory パラメーターは必須です。

initialContextFactory
initialContextFactory パラメーターは必須で、WebSphere Application Server およびその他の製品との互換性を保つために、その値を「com.ibm.mq.jms.Nojndi」に設定する必要があります。

非 SOAP/非 JMS インターフェース

URI フォーマットは MA93 仕様と同様です。 IBM MQ IRI 仕様の詳細については、サポートパック - MA93 を参照してください。

IBM MQ URI 構文

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

IBM MQ IRI の例

次のサンプル IRI は、example.com というマシンのポート 1414 に対して IBM MQ TCP クライアント・バインディング接続を使用できること、およびキュー・マネージャー QM1 上の SampleQ というキューに持続要求メッセージを置けることをサービス要求元に伝えます。 IRI は、SampleReplyQ というキューにサービス・プロバイダーが応答を置くことを指定しています。

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

TLS 対応の接続にするには

WCF クライアント/サービスを使用してセキュア (TLS) 接続を行うには、URI 内で以下のプロパティーに適切な値を設定します。 接頭部に「*」の付いたプロパティーはすべて、セキュア接続を行うために必須です。

- **sslKeyRepository**: *SYSTEM または *USER
- ***sslCipherSpec**: 有効な CipherSpec (例えば TLS_RSA_WITH_AES_128_CBC_SHA256)
- **sslCertRevocationCheck**: true または false
- **sslKeyResetCount**: 32kb より大きい値
- **sslPeerName:** サーバー証明書の識別名

以下に例を示します。

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

WCF カスタム・チャネルの保証配信

保証配信により、サービス要求または応答が確実に実施され、失われることはありません。

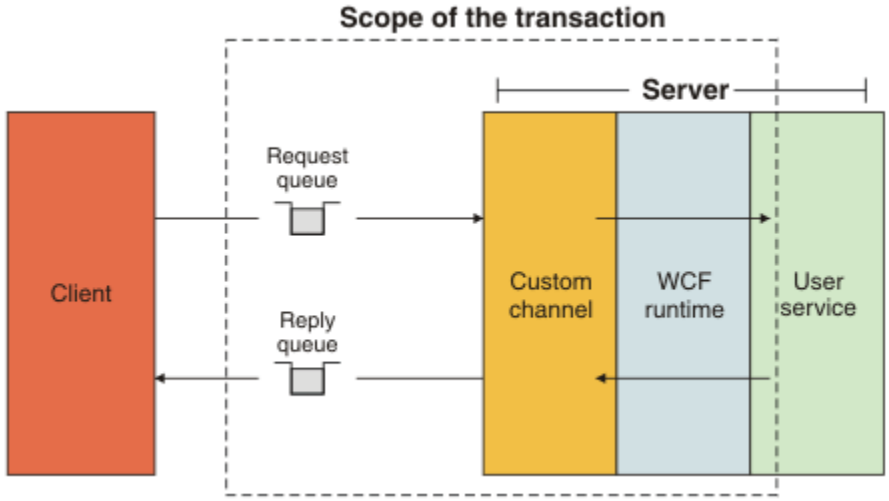
要求メッセージの受信および応答メッセージの送信は、ローカル・トランザクションの同期点下で行われ、ランタイム障害の場合には、これらのメッセージをロールバックすることができます。 ランタイム障害の例としては、サービスから未処理例外がスローされた、メッセージをサービスにディスパッチできなかった、応答メッセージを配信できなかった、などがあります。

AssuredDelivery は、サービスで受信された要求メッセージ、およびサービスから送信された応答メッセージを、ランタイム障害が発生しても失わないことを保証する、サービス契約で指定できる保証配信属性です。

システム障害や電源異常が発生した場合もメッセージを確実に保持するためには、メッセージを持続メッセージとして送信する必要があります。 持続メッセージを使用するには、クライアント・アプリケーションのエンドポイント URI でこのオプションを指定しておく必要があります。

分散トランザクションはサポートされず、トランザクションのスコープが、IBM MQ で実行される要求および応答メッセージ処理の枠を超えることはありません。 サービス内で実行された処理はすべて、メッセー

ジの再受信の原因となった障害の結果として再実行できます。 次の図に、トランザクションのスコープを示します。



以下の例に示すように、AssuredDelivery 属性をサービス・クラスに適用することにより、確実性のある送達が有効になります。

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

AssuredDelivery 属性を使用するときは、次の点に注意してください。

- メッセージがロールバックされて再受信された場合に障害が再発生する可能性があるとチャネルが判断すると、そのメッセージは有害メッセージとして扱われ、再処理用の要求キューには返されません。 例えば、受信メッセージが正しくフォーマットされていなかったり、サービスにディスパッチできなかったりする場合です。 サービス操作からスローされた未処理例外は、メッセージの再配信回数が、要求キューのバックアウトしきい値プロパティーで指定された最大回数に達するまで、必ず再送信されます。 詳しくは、1280 ページの『WCF カスタム・チャネル有害メッセージ』を参照してください。
- チャネルは、トランザクションの整合性を確保するために単一実行スレッドを使用して、各要求メッセージの読み取り、処理、および応答をアトミック操作として実行します。 サービス操作を並行して実行できるように、チャネルは WCF がチャネルの複数のインスタンスを作成することを可能にします。 要求の処理で使用可能なチャネル・インスタンスの数は、バインディング・プロパティー MaxConcurrentCalls で制御されます。 詳しくは、1288 ページの『WCF バインディング構成オプション』を参照してください。
- 保証配信機能は、IOperationInvoker および IErrorHandler の両方の WCF 拡張ポイントを使用します。 これらの拡張ポイントを外部で使用するアプリケーションは、事前に登録済みの拡張ポイントが確実に呼び出されるようにする必要があります。 IErrorHandler に対してこれを実行できないと、エラーが報告されない可能性が生じます。 IOperationInvoker でこれを実行できなかった場合は、WCF が応答を停止する可能性があります。

WCF カスタム・チャネル・セキュリティー

WCF 用の IBM MQ カスタム・チャネルは、キュー・マネージャーへの非管理対象クライアント接続の場合にのみ TLS の使用をサポートします。

クライアント・チャネル定義テーブル (CCDT) 内の項目を使用して TLS を指定します。 CCDT について詳しくは、クライアント・チャネル定義テーブルを参照してください。

WCF クライアント・チャネル定義テーブル *(CCDT)*

WCF 用の IBM MQ カスタム・チャネルは、クライアント・チャネル定義テーブル (CCDT) の使用による、クライアント接続の接続情報の構成をサポートしています。

CCDT は、次の 2 つの環境変数によって制御します。

- *MQCHLLIB* で、テーブルが置かれているディレクトリーを指定します。
- *MQCHLTAB* で、テーブルのファイル名を指定します。

これらの環境変数が定義されている場合、これらの環境変数は、URI で指定されているクライアント接続詳細に優先します。

クライアント・チャネル定義テーブルについて詳しくは、クライアント・チャネル定義テーブルを参照してください。

WCF カスタム・チャネル有害メッセージ

サービスが要求メッセージの処理に失敗したり、応答キューへの応答メッセージの配信に失敗したりすると、そのメッセージは有害メッセージとして扱われます。

有害要求メッセージ

要求メッセージを処理できないと、そのメッセージは有害メッセージとして扱われます。 このアクションにより、サービスは、処理不能の同じメッセージを二度と受信しなくなります。 処理不能の要求メッセージを有害メッセージとして扱うためには、次のいずれかの状況が当てはまる必要があります。

- メッセージのバックアウト・カウントが、要求キューで指定されたバックアウトしきい値を超えた場合。これが発生するのは、サービスに対して保証配信が指定されている場合だけです。 保証配信について詳しくは、1278 ページの『WCF カスタム・チャネルの保証配信』を参照してください。
- メッセージが正しくフォーマットされていないため、SOAP over JMS メッセージとして解釈できなかった場合。

有害応答メッセージ

サービスが応答キューへの応答メッセージの配信に失敗すると、その応答メッセージは有害メッセージとして扱われます。 応答メッセージの場合は、このアクションにより、後で問題判別に役立てるために応答メッセージを取得することが可能になります。

有害メッセージの処理

有害メッセージに対するアクションは、キュー・マネージャーの構成、およびメッセージのレポート・オプションに設定されている値によって異なります。 SOAP over JMS では、次のレポート・オプションがデフォルトで要求メッセージに対し設定されています。これらのレポート・オプションは構成できません。

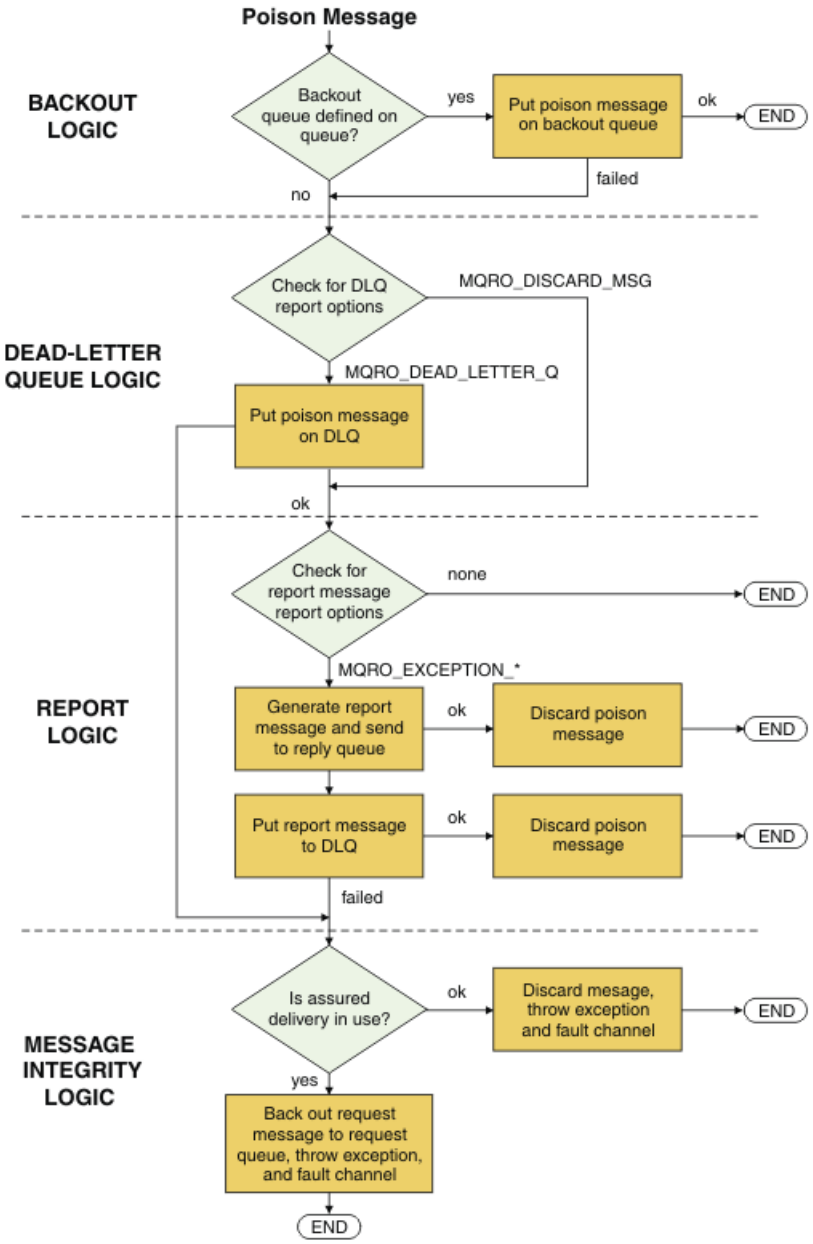
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

SOAP over JMS では、次のレポート・オプションがデフォルトで応答メッセージに対し設定されています。このレポート・オプションは構成できません。

- MQRO_DEAD_LETTER_Q

メッセージの送信元が WCF 以外の場合は、その送信元の資料を参照してください。

次の図に、有効なアクション、および有害メッセージ処理が失敗した場合に取られるステップを示します。



IBM MQ メッセージの WCF アプリケーション対応能力

非 SOAP/非 JMS (すなわち IBM MQ) メッセージの WCF アプリケーション対応能力

非 SOAP/非 JMS インターフェースの場合、IBM MQ メッセージの WCF アプリケーション対応能力は、以下のとおりです。

- WCF アプリケーションは、どのような IBM MQ アプリケーションでも処理できる IBM MQ 基本メッセージを送受信できます。
- WCF アプリケーションは MQMD およびペイロードの更新を完全に制御できます。
- WCF クライアントは、任意の IBM MQ クライアント (C、 Java、 JMS、および .NET クライアントなど) がコンシュームできる IBM MQ メッセージを送信できます。

非 SOAP/非 JMS インターフェースの場合、WCF では、メッセージ・ペイロードとメッセージに関する MQMD の設定に、以下のクラスを使用する必要があります。

- WmqStringMessage (String タイプのペイロードの場合)
- WmqBytesMessage (Bytes タイプのペイロードの場合)
- WmqXmlMessage (XML タイプのペイロードの場合)

メッセージのペイロードを設定するには、ペイロードのタイプに応じて、WmqStringMessage、WmqBytesMessage、または WmqXmlMessage クラスの **Data** プロパティーを使用します。 例えば、String タイプのペイロードを設定するには、次のコードを使用します。

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

WCF 接続オプション

WCF 用の IBM MQ カスタム・チャネルをキュー・マネージャーに接続するモードは 3 つあります。 どのタイプの接続が最も必要に適しているかを考慮してください。

接続オプションについて詳しくは、578 ページの『接続の違い』を参照してください。

WCF アーキテクチャーについて詳しくは、1275 ページの『WCF のアーキテクチャー』を参照してください。

非管理対象クライアント接続

このモードでの接続では、ローカル・マシンまたはリモート・マシンで実行されている IBM MQ サーバーに IBM MQ クライアントとして接続します。

WCF 用の IBM MQ カスタム・チャネルを IBM MQ クライアントとして使用するには、そのカスタム・チャネルを IBM MQ MQI client と一緒に IBM MQ サーバーにインストールしても、または別個のマシンにインストールしても構いません。

非管理対象サーバー接続

サーバー・バインディング・モードで使用する場合、WCF 用の IBM MQ カスタム・チャネルは、ネットワーク経由で通信するのではなく、キュー・マネージャー API を使用します。 バインディング接続を使用すると、ネットワーク接続を使用した場合よりも、IBM MQ アプリケーションのパフォーマンスが向上します。

バインディング接続を使用するには、WCF 用の IBM MQ カスタム・チャネルを IBM MQ サーバーにインストールする必要があります。

管理対象クライアント接続

このモードでの接続では、ローカル・マシンまたはリモート・マシンで実行されている IBM MQ サーバーに IBM MQ クライアントとして接続します。

このモードで接続する .NET 3 用の IBM MQ カスタム・チャネル・クラスは、 .NET 管理対象コード内に残り、ネイティブ・サービスへの呼び出しを行いません。 管理対象コードについて詳しくは、Microsoft の資料を参照してください。

管理対象クライアントの使用には、いくつかの制限があります。 これらの制限事項について詳しくは、578 ページの『管理対象クライアント接続』を参照してください。

WCF 用の IBM MQ カスタム・チャネルの作成および構成

WCF 用の IBM MQ カスタム・チャネルは、Microsoft によって提供されるトランスポート WCF チャネルと同じ方法で機能します。 WCF 用の IBM MQ カスタム・チャネルは、2 つの方法のどちらかで作成できます。

このタスクについて

IBM MQ カスタム・チャネルは、WCF トランスポート・チャネルとして WCF に統合されるため、メッセージ・エンコーダーおよびオプションのプロトコル・チャネルとペアにする必要があります。それによって、

アプリケーションで使用できる完全なチャネル・スタックを作成することができます。 完全なチャネル・スタックを正常に作成するためには、以下の 2 つの要素が必要になります。

1. バインディング定義: アプリケーション・チャネル・スタックの作成に必要な要素を指定します。 それらの要素には、トランスポート・チャネル、メッセージ・エンコーダー、および任意のプロトコル、 さらに一般的な構成設定が含まれます。 カスタム・チャネルの場合、バインディング定義は WCF カスタム・バインディングの形式で作成する必要があります。
2. エンドポイント定義: サービス・コントラクトをバインディング定義にリンクし、アプリケーションが接続可能な宛先を示す実際の接続 URI も提供します。 カスタム・チャネルの場合、この URI は SOAP over JMS URI の形式になります。

これらの定義は、次の 2 つの方法のいずれかで作成できます。

- 管理: 定義は、アプリケーション構成ファイルに詳細を指定することによって作成されます (例: `app.config`)。
- プログラマチック: 定義はアプリケーション・コードから直接作成されます。

定義を作成する際にどちらの方法を使用するかについては、以下のように、アプリケーションの要件に基づいて決定する必要があります。

- 構成のための管理方法では、アプリケーションを再作成しなくても、サービスおよびクライアントのデプロイメント後の詳細を柔軟に変更することができます。
- 「プログラマチック」による構成方法では、構成エラーに対する保護が強化され、 また実行時に構成を動的に生成することができます。

アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、*WCF* カスタム・チャネルを管理的に作成する

WCF 用の IBM MQ カスタム・チャネルは、トランスポート・レベルの WCF チャネルです。 このカスタム・チャネルを使用するには、エンドポイントおよびバインディングを定義する必要があります。これらの定義は、アプリケーション構成ファイルにバインディングおよびエンドポイント情報を指定することにより定義することができます。

WCF 用の IBM MQ カスタム・チャネル (トランスポート・レベルの WCF チャネル) を構成して使用するには、 バインディングおよびエンドポイント定義を定義する必要があります。 バインディング定義はチャネルの構成情報を保持し、エンドポイント定義は接続の詳細を保持します。 これらの定義は、以下の 2 つの方法で作成できます。

- アプリケーション・コードからプログラマチックに直接作成する。詳しくは、『1285 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』』を参照してください。
- アプリケーション構成ファイルで詳細を指定することにより、管理的に作成する。この方法については、以下の手順で説明します。

通常、クライアントまたはサービス・アプリケーションの構成ファイルには *yourappname*`.exe.config` という名前が付けられます。ここで、*yourappname* はアプリケーションの名前です。 アプリケーション構成ファイルは、以下の方法で `SvcConfigEditor.exe` という Microsoft サービス構成エディター・ツールを使用して、非常に簡単に変更できます。

- `SvcConfigEditor.exe` 構成エディター・ツールを開始します。 このツールのデフォルトのインストール・ロケーションは *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` です。ここで、ドライブ*:* はインストール・ドライブの名前です。

ステップ 1: バインディング要素拡張を追加して、WCF がカスタム・チャネルを検出できるようにする

1. 「**拡張**」 > 「**拡張機能**」 > 「**バインディング要素**」と右クリックしてメニューを開き、「**新規**」を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *190.* 新規バインディング要素フィールド	
フィールド	**値**
名前	IBM.XMS.WCF.SoapJmsIbmTransportChannel
タイプ	グローバル・アセンブリー・キャッシュ (GAC) で IBM.XMS.WCF.dll にナビゲートし、IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig を選択します。

ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する

1. 「**バインディング**」を右クリックしてメニューを開き、「**新規バインディング構成**」を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *191.* 新規バインディング構成フィールド	
フィールド	**値**
名前	CustomBinding_WMQ
BindingElement 1	textMessageEncoding (MessageVersion: Soap11)
BindingElement 2	IBM.XMS.WCF.SoapJmsIbmTransportChannel

ステップ 3: バインディング・プロパティーを指定する

1. 『1284 ページの『ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する』』で作成したバインディングから *IBM.XMS.WCF.SoapJmsIbmTransportChannel* トランスポート・バインディングを選択します。
2. 『1288 ページの『WCF バインディング構成オプション』』の説明に従って、プロパティーのデフォルト値に必要な変更を加えます。

ステップ 4: エンドポイント定義を作成する

1284 ページの『ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する』で作成したカスタム・バインディングを参照し、サービスの接続詳細を提供するエンドポイント定義を作成します。 この情報を指定する方法は、エンドポイント定義がクライアント・アプリケーション用の定義か、サービス・アプリケーション用の定義かによって異なります。

クライアント・アプリケーションの場合、以下のようにして、エンドポイント定義をクライアント・セクションに追加します。

1. 「**クライアント**」 > 「**エンドポイント**」と右クリックしてメニューを開き、「**新規クライアント・エンドポイント**」を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *192.* 新規クライアント・エンドポイント・フィールド	
フィールド	**値**
名前	Endpoint_WMQ
アドレス	サービスにアクセスするために必要な *WMQ* 接続の詳細を記述する *SOAP/JMS URI*。 詳細は、『1287 ページの『WCF 用の IBM MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』』を参照してください。

表 *192.* 新規クライアント・エンドポイント・フィールド *(続き)*	
フィールド	**値**
Binding	customBinding
BindingConfiguration	CustomBinding_WMQ
Contract	サービス・コントラクト・インターフェースの名前

サービス・アプリケーションの場合、以下のようにして、サービス定義をサービス・セクションに追加します。

1. 「**サービス**」を右クリックしてメニューを開き、「**新規サービス**」を選択してから、ホスト対象のサービス・クラスを選択します。
2. エンドポイント定義を新規サービスの「**エンドポイント**」セクションに追加し、以下の表に示されているようにフィールドに入力します。

表 *193.* 新規サービス・エンドポイント・フィールド	
フィールド	**値**
名前	Endpoint_WMQ
アドレス	サービスにアクセスするために必要な *WMQ* 接続の詳細を記述する *SOAP/JMS URI*。 詳細は、『1287 ページの『WCF 用の IBM MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』』を参照してください。
Binding	customBinding
BindingConfiguration	CustomBinding_WMQ
Contract	サービス実装クラスの名前

バインディング情報とエンドポイント情報をプログラマチックに指定することにより、*WCF* カスタム・チャネルを作成する

WCF 用の IBM MQ カスタム・チャネルは、トランスポート・レベルの WCF チャネルです。 カスタム・チャネルを使用するには、エンドポイントおよびバインディングを定義する必要があります。これらは、アプリケーション・コードから直接プログラマチックに定義することができます。

WCF 用の IBM MQ カスタム・チャネル (トランスポート・レベルの WCF チャネル) を構成して使用するには、 バインディングおよびエンドポイント定義を定義する必要があります。 バインディング定義はチャネルの構成情報を保持し、エンドポイント定義は接続の詳細を保持します。 詳しくは、1295 ページの『WCF サンプルの使用』を参照してください。

これらの定義は、以下の 2 つの方法で作成できます。

- アプリケーション構成ファイルで詳細を指定することにより、管理的に作成する。この方法については、1283 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』で説明します。
- アプリケーション・コードからプログラマチックに直接作成する。 この方法については、以下のサブトピックで説明します。

バインディング情報とエンドポイント情報のプログラマチック定義: *SOAP/JMS* インターフェース

SOAP/JMS インターフェースの場合、エンドポイントとバインディングをアプリケーション・コードからプログラマチックに直接定義できます。

このタスクについて

バインディング情報とエンドポイント情報をプログラマチックに指定するには、以下のステップを実行して、必要なコードをアプリケーションに追加します。

手順

1. 次のコードをアプリケーションに追加することによって、チャネルのトランスポート・バインディング要素のインスタンスを作成します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. 例えば、次のコードをアプリケーションに追加して ClientConnectionMode を設定することによって、必要なバインディング・プロパティーを設定します。

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. 次のコードをアプリケーションに追加することによって、トランスポート・チャネルとメッセージ・エンコーダーを組み合わせるカスタム・バインディングを作成します。

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. SOAP/JMS URI を作成します。

   サービスにアクセスするために必要な IBM MQ 接続の詳細を記述する SOAP/JMS URI は、エンドポイント・アドレスとして指定する必要があります。 指定するアドレスは、チャネルがサービス・アプリケーションに使用されるかクライアント・アプリケーションに使用されるかによって異なります。

   - クライアント・アプリケーションの場合は、以下のように、SOAP/JMS URI を EndpointAddress として作成する必要があります。

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - サービス・アプリケーションの場合は、以下のように、SOAP/JMS URI を URI として作成する必要があります。

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     エンドポイント・アドレスについて詳しくは、1287 ページの『WCF 用の IBM MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』を参照してください。

バインディング情報とエンドポイント情報のプログラマチック定義*:* 非 *SOAP*/非 *JMS* インターフェース
非 SOAP/非 JMS インターフェースの場合、エンドポイントとバインディングをアプリケーション・コードからプログラマチックに直接定義できます。

このタスクについて

バインディング情報とエンドポイント情報をプログラマチックに指定するには、以下のステップを実行して、必要なコードをアプリケーションに追加します。

手順

1. 次のコードをアプリケーションに追加することによって、WmqBinding を作成します。

```
WmqBinding binding = new WmqBinding();
```

このコードは、非 SOAP/非 JMS インターフェースに必要な WmqMsgEncodingElement と WmqIbmTransportBindingElement を組み合わせるバインディングを作成します。

2. サービスにアクセスするために必要な IBM MQ 接続詳細を記述した wmq:// URI を指定します。

   wmq:// URI の指定の仕方は、チャネルがサービス・アプリケーションに使用されるかクライアント・アプリケーションに使用されるかによって異なります。

   - クライアント・アプリケーションの場合は、以下のように、wmq:// URI を EndpointAddress として作成する必要があります。

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - サービス・アプリケーションの場合は、以下のように、wmq:// URI を URI として作成する必要があります。

     ```
     Uri sampleAddress = new Uri(
     "wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

WCF 用の IBM MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット

Web サービスは、ロケーションと接続の詳細を示す Universal Resource Identifier (URI) を使用して指定されます。 URI フォーマットは、SOAP/JMS インターフェースを使用するか非 SOAP/非 JMS インターフェースを使用するかによって異なります。

SOAP/JMS インターフェース

IBM MQ transport for SOAP でサポートされる URI フォーマットでは、ターゲット・サービスにアクセスする際に SOAP/IBM MQ 固有のパラメーターとオプションを包括的に制御できます。 フォーマットは、WebSphere Application Server および CICS と互換性があり、それらの製品の両方を使って IBM MQ の統合を容易にします。

URI 構文は以下のとおりです。

```
jms:/queue? name=value&name=value...
```

ここで、name はパラメーター名で、*value* は適切な値です。また、name = *value* エレメントは、2 回目以降に出現するときにアンパーサンド (&) が付いている場合は、何回も繰り返すことができます。

パラメーター名は、IBM MQ オブジェクトの名前と同様、大文字小文字を区別します。 いずれかのパラメーターを複数回指定した場合は、最後に出現した該当パラメーターが有効になります。つまり、クライアント・アプリケーションは、URI にパラメーター値を付加することにより、パラメーター値を指定変更できます。 認識されない追加のパラメーターが含まれている場合、それらは無視されます。

URI を XML ストリングで保管する場合、 アンパーサンド文字を "&" と記述しなければなりません。同様に、スクリプト内に URI がコーディングされている場合は、**&** などのエスケープ文字に注意してください。そうでない場合は、シェルによって解釈されます。

次に、Axis サービス用の単純な URI の例を示します。

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

以下に、.NET サービス用の簡単な URI の例を示します。

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

必須パラメーターのみが提供され (targetService は .NET サービスでのみ必須)、connectionFactory にはオプションが指定されていません。

次の Axis の例では、connectionFactory にいくつかのオプションが含まれます。

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

この Axis の例では、connectionFactory の sslPeerName オプションも指定しています。 sslPeerName 自体の値には、 名前と値のペア、および意味のある組み込みブランクが含まれます。

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

非 SOAP/非 JMS インターフェース

非 SOAP/非 JMS インターフェース対応の URI フォーマットでは、ターゲット・サービスにアクセスする際に IBM MQ 固有のパラメーターとオプションを包括的に制御できます。

URI 構文は以下のとおりです。

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

この IRI は、example.com というマシンのポート 1415 に対して IBM MQ TCP クライアント・バインディング接続を使用できること、およびキュー・マネージャー MOTOR.INS 上の INS.QUOTE.REQUEST というキューに持続要求メッセージを置けることをサービス要求元に伝えます。 IRI は、キュー・マネージャー BRANCH452 上の INS.QUOTE.REPLY というキューにサービス・プロバイダーが応答を置くことを指定しています。 URI フォーマットは SupportPac MA93 での仕様と同様です。 IBM MQ IRI 仕様について詳しくは、 SupportPac MA93: IBM MQ -サービス定義 を参照してください。

WCF バインディング構成オプション

カスタム・チャネルのバインディング情報に構成オプションを適用する方法には、2 とおりあります。 プロパティーを管理的に設定するか、プログラマチックに設定します。

バインディング構成オプションは、以下の 2 つの方法のいずれかで設定できます。

1. 管理: バインディング・プロパティー設定は、アプリケーション構成ファイル (例: app.config) 内のカスタム・バインディング定義のトランスポート・セクションに指定する必要があります。
2. プログラマチック: カスタム・バインディングの初期化中にプロパティーを指定するように、 アプリケーション・コードを変更する必要があります。

バインディング・プロパティーを管理的に設定する

バインディング・プロパティー設定は、アプリケーション構成ファイル (例: app.config) 内で指定できます。 この構成ファイルは、**svcutil** によって、次の例のように生成されます。

SOAP/JMS インターフェース

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

非 SOAP/非 JMS インターフェース

```
<customBinding>
```

```
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

バインディング・プロパティーをプログラマチックに設定する

クライアント接続モードを指定するために WCF バインディング・プロパティーを追加するには、カスタム・バインディングの初期化時にプロパティーを指定するようにサービス・コードを変更する必要があります。

以下の例を使用して、非管理のクライアント接続モードを指定します。

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

WCF バインディング・プロパティー

表 *194.* 管理的またはプログラマチックに設定する際のバインディング・プロパティーの値

プロパティー名	クライアント・アプリケーションまたはサービス・アプリケーション	管理値	プログラムによる値	説明
maxBufferPoolSize	両方	0 から 64 ビットの符号付き整数	0 から 64 ビットの符号付き整数	チャネル・インスタンスの WCF メッセージ・バッファーの保管のために使用可能なメモリーの最大サイズを指定します。
maxMessageSize	両方	1 から 32 ビットの符号付き整数	1 から 32 ビットの符号付き整数	個別の WCF メッセージに使用可能な最大メモリーを指定します。
clientConnectionMode	両方	0 (デフォルト値) 1	AS_URI (デフォルト値) CLIENT_UNMANAGED	トランスポート・チャネルのクライアント接続モードを指定します。 0 は、クライアント接続モードが URI に指定されたとおりであることを意味します。 クライアント接続が使用されている場合にのみ使用します。 クライアント接続モードが、URI に指定されたとおりであることを指定します。 クライアント接続モードが設定されていない場合、0 がデフォルト値です。 1 は、クライアント接続モードが非管理クライアントであることを意味します。 クライアント接続が使用されている場合にのみ使用します。

表 194. 管理的またはプログラマチックに設定する際のバインディング・プロパティーの値 (続き)

プロパティー名	クライアント・アプリケーションまたはサービス・アプリケーション	管理値	プログラムによる値	説明
MaxConcurrentCalls	クライアント	範囲は 0 から 2 147 483 647 です。 16 がデフォルト値です。	範囲は 0 から 2 147 483 647 です。 16 がデフォルト値です。	このプロパティーは、個々のクライアント・プロキシーに対して同時に実行できる並行操作の最大数を定義します。 それよりも多くの操作が開始された場合、進行中の操作が完了するかタイムアウトになるまで操作はキューに入れられます。 この設定を使用して、個々のプロキシーが取り込むことができる スレッドおよびリソースの最大数を制御できます。 0 はこの制限を取り外し、 すべての操作を並行して試行できます。
MaxConcurrentCalls	サービス	範囲は 1 から 2 147 483 647 です。 16 がデフォルト値です。	範囲は 1 から 2 147 483 647 です。 16 がデフォルト値です。	このプロパティーが使用されるのは、確約済みデリバリー・フィーチャーが使用可能である場合のみです (確約済みデリバリーについて詳しくは、1278 ページの『WCF カスタム・チャネルの保証配信』を参照してください)。 これは、所定のエンドポイントに同時に進行できる並行操作の最大数を指定します。 この設定を変更するときには、注意が必要です。 それぞれの並行操作には追加リソースが必要とされ、特に、要求を実行するための、カスタム・チャネルの新規インスタンスおよびスレッド・プールからの関連スレッドが必要です。 過剰な割り振りは、生産性を低下させ、 パフォーマンスに重大な影響を及ぼすことがあります。 このプロパティーをサポートするには、スレッド・プールを適切に構成する必要があります。

WCF 用サービスの作成とホスティング

WCF サービスを作成および構成する方法を説明する Microsoft Windows Communication Foundation (WCF) サービスの概要。

WCF およびそれを使用する WCF サービス用の IBM MQ カスタム・チャネルは、以下の方法によってホストすることができます。

- セルフ・ホスティング

- Windows サービス

WCF 用の IBM MQ カスタム・チャネルは、Windows Process Activation Service にホストできません。

以下の各トピックには、必要なステップが示された簡単なセルフ・ホスティングの例が記載されています。詳細情報と最新情報が記載された Microsoft WCF オンライン文書は、Microsoft MSDN の Web サイト (https://msdn.microsoft.com) にあります。

1 番目の方法を使用した *WCF* サービス・アプリケーションの作成*:* アプリケーション構成ファイルを使用して管理的にセルフ・ホスティングする

アプリケーション構成ファイルを作成したら、サービスのインスタンスを開き、指定されたコードをアプリケーションに追加します。

始める前に

『1283 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』』の説明に従って、サービスのアプリケーション構成ファイルを作成または編集します。

このタスクについて

1. サービス・ホストでサービスのインスタンスを生成し、インスタンスを開きます。 サービス・タイプは、サービス構成ファイルに指定されているサービス・タイプと同じでなければなりません。
2. 以下のコードをご使用のアプリケーションに追加します。

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

2 番目の方法を使用した *WCF* サービス・アプリケーションの作成*:* アプリケーションから直接プログラマチックにセルフ・ホスティングする

バインディング・プロパティーを追加し、必要なサービス・クラスのインスタンスを使用してサービス・ホストを作成し、そのサービスを開きます。

始める前に

1. カスタム・チャネルの IBM.XMS.WCF.dll ファイルへの参照をプロジェクトに追加します。IBM.XMS.WCF.dll は *WMQInstallDir*\bin にあります。ここで *WMQInstallDir* は IBM MQ がインストールされているディレクトリーを指します。
2. *using* ステートメントを IBM.XMS.WCF 名前空間に追加します。例えば、次のとおりです。using IBM.XMS.WCF
3. 『1285 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』』の説明に従って、チャネル・バインディング要素のインスタンスとエンドポイントを作成します。

このタスクについて

チャネルのバインディング・プロパティーへの変更が必要な場合は、以下のステップを実行します。

1. 以下の例に示すように、バインディング・プロパティーを transportBindingElement に追加します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
```

```
connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 必要なサービス・クラスのインスタンスを使用してサービス・ホストを作成します。

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. サービスを開きます。

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

HTTP エンドポイントを使用したメタデータの公開

WCF 用の IBM MQ カスタム・チャネルを使用するよう構成されたサービスのメタデータを公開するための手順です。

このタスクについて

サービス・メタデータが公開されている必要がある場合 (`svcutil` などのツールがオフラインの WSDL ファイルからではなく、実行中のサービスから直接アクセスできるようにするため)、HTTP エンドポイントを使用してサービス・メタデータを公開する必要があります。 以下のステップを使用して、このエンドポイントを追加できます。

1. メタデータを ServiceHost に公開する必要のある基底アドレスを追加します。例えば、次のとおりです。

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. サービスが開かれる前に、以下のコードを ServiceHost に追加します。

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

タスクの結果

これで、メタデータは以下のアドレスで使用可能になります。`http://localhost:8000/MyService`

WCF 用のクライアント・アプリケーションの作成

Microsoft Windows Communication Foundation (WCF) クライアント・アプリケーションの生成および作成の概要です。

WCF サービス用のクライアント・アプリケーションを作成できます。通常、クライアント・アプリケーションの生成は、Microsoft ServiceModel メタデータ・ユーティリティー・ツール (Svcutil.exe) を使用して、アプリケーションで直接使用できる必要な構成ファイルとプロキシー・ファイルを作成することにより行います。

svcutil ツールを使用して、実行中のサービスのメタデータにより *WCF* クライアント・プロキシーおよびアプリケーション構成ファイルを生成する

Microsoft svcutil.exe ツールを使用して、WCF 用の IBM MQ カスタム・チャネルを使用するよう構成されたサービスのクライアントを生成する手順です。

始める前に

svcutil ツールを使用して、アプリケーションで直接使用できる必要な構成およびプロキシー・ファイルを作成するには、 以下の 3 つの前提条件があります。

- svcutil ツールを開始する前に、WCF サービスが実行されている必要があります。
- WCF サービスは、実行中のサービスからクライアントを直接生成するために、IBM MQ カスタム・チャネル・エンドポイント参照に加えて、HTTP ポートを使用してそのメタデータを公開する必要があります。
- カスタム・チャネルが svcutil の構成データに登録されている必要があります。

このタスクについて

以下のステップでは、IBM MQ カスタム・チャネルを使用するように構成されているが、別個の HTTP ポートを介して実行時にそのメタデータの公開も行う、サービスのクライアントを生成する方法について説明します。

1. WCF サービスを開始します (サービスは、svcutil ツールを開始する前に実行されている必要があります)。
2. インストールのルートから `svcutil.exe` 構成ファイルからの詳細をアクティブな svcutil 構成ファイルへ追加します。そのため、一般的に `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` svcutil は IBM MQ カスタム・チャネルを認識します。
3. コマンド・プロンプトから svcutil を実行します。例えば、次のようになります。

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. 生成された `app.config` ファイルおよび `YourService.cs` ファイルを Microsoft Visual Studio クライアント・プロジェクトにコピーします。

次のタスク

サービス・メタデータを直接取得できない場合は、svcutil を使用して、代わりに wsdl からクライアント・ファイルを生成できます。 詳しくは、『1293 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』を参照してください。

svcutil ツールを使用して、*WSDL* による *WCF* クライアント・プロキシーおよびアプリケーション構成ファイルを生成する

サービスのメタデータを使用できない場合に、WSDL から WCF クライアントを生成するための手順です。

サービスのメタデータを直接取り出して、実行中のサービスのメタデータからクライアントを生成することができない場合は、svcutil を使用して、代わりに WSDL からクライアント・ファイルを生成できます。以下の変更を WSDL に加えて、IBM MQ カスタム・チャネルが使用されるように指定する必要があります。

1. 以下の名前空間定義およびポリシー情報を追加します。

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. 新規ポリシー・セクションを参照するようバインディング・セクションを変更し、基礎となるバインディング要素から transport 定義をすべて除去します。

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. コマンド・プロンプトから svcutil を実行します。例えば、次のようになります。

```
svcutil /language:C# /r:  MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
/config:app.config  MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service
\soap.server.stockQuoteAxis_Wmq.wsdl
```

アプリケーション構成ファイルによる、クライアント・プロキシーを使用する *WCF* クライアント・アプリケーションの作成

始める前に

『1283 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』』の説明に従って、クライアント用のアプリケーション構成ファイルを作成または編集します。

このタスクについて

クライアント・プロキシーのインスタンスを生成して開きます。 生成したプロキシーに渡されるパラメーターは、クライアント構成ファイルに指定されたエンドポイント名 (Endpoint_WMQ など) と同じにする必要があります。

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

プログラマチック構成による、クライアント・プロキシーを使用する *WCF* クライアント・アプリケーションの作成

始める前に

1. カスタム・チャネルの IBM.XMS.WCF.dll ファイルへの参照をプロジェクトに追加します。IBM.XMS.WCF.dll は、*WMQInstallDir*\bin ディレクトリーにあります。ここで *WMQInstallDir* は、IBM MQ がインストールされているディレクトリーです。
2. *using* ステートメントを IBM.XMS.WCF 名前空間に追加します。例えば、次のとおりです。using IBM.XMS.WCF

3. 1285 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』の説明に従って、チャネルのバインディング要素とエンドポイントのインスタンスを作成します。

このタスクについて

チャネルのバインディング・プロパティーの変更が必要な場合は、以下のステップを実行します。

1. 以下の図に示すように、バインディング・プロパティーを transportBindingElement に追加します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 以下の図に示すようにクライアント・プロキシーを作成します。ここで、*binding* および *endpoint address* は、ステップ 1 で構成されたバインディングとエンドポイント・アドレスで、次のように渡されます。

```
MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
           try {
                  myClient.myMethod("HelloWorld!");
                  myClient.Close();
           }
           catch (TimeoutException e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
           catch (CommunicationException e) {
               Console.Out.WriteLine(e);
                                myClient.Abort();
           }
           catch (Exception e) {
               Console.Out.WriteLine(e);
               myClient.Abort();
           }
```

WCF サンプルの使用

Windows Communication Foundation (WCF) サンプルでは、IBM MQ カスタム・チャネルの使用方法の簡単な例をいくつか紹介します。

サンプル・プロジェクトを作成するには、Microsoft .NET 3.5 SDK または Microsoft Visual Studio 2008 のいずれかが必要です。

単純な片方向クライアント/サーバー WCF サンプル

このサンプルは、片方向チャネル形状を使用して WCF クライアントから Windows Communication Foundation (WCF) サービスを開始するために使用される IBM MQ カスタム・チャネルを示します。

このタスクについて

このサービスは、ストリングをコンソールに出力する単一メソッドを実装します。 1292 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』 で説明されているように svcutil ツールを使用し、別々に公開された HTTP エンドポイントからサービス・メタデータを取得することでクライアントが生成されました。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config ファイル内のクライアント・アプリケーション上および *MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs ファイル

内のサービス・アプリケーションで対応する値も変更する必要があります。ここで *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。 JMS エンドポイント URI のフォーマットについて詳しくは、 IBM MQ 製品資料の「 *IBM MQ Transport for SOAP* 」を参照してください。 サンプル・ソリューションおよびソースを変更する必要がある場合は、IDE (例えば、Microsoft Visual Studio 8 以降など) が必要になります。

手順

1. *QM1*
2. *SampleQ*
3. リスナーが以下のメッセージを待つためにサービスを開始します。 *MQ_INSTALLATION_PATH*\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリーです。
4. クライアントを 1 回実行: *MQ_INSTALLATION_PATH* \tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。

   クライアント・アプリケーションは 5 回ループして、5 つのメッセージを *SampleQ* に送信します。

タスクの結果

サービス・アプリケーションは *SampleQ* からメッセージを取得し、画面上に `Hello World` を 5 回表示します。

次のタスク

単純な応答-要求クライアント/サーバー WCF サンプル

このサンプルは、要求/応答チャネル・シェイプを使用して WCF クライアントから Windows Communication Foundation (WCF) サービスを開始するために使用される IBM MQ カスタム・チャネルを示しています。

このタスクについて

このサービスは、2 つの数値を加算および減算するいくつかの単純な計算器メソッドを提供し、結果を返します。 1292 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』 で説明されているように `svcutil` ツールを使用し、別々に公開された HTTP エンドポイントからサービス・メタデータを取得することでクライアントが生成されました。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\client\app.config ファイル内のクライアント・アプリケーションの対応する値、および *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs ファイル内のサービス・アプリケーションで値を変更する必要があります。そこで。*MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。 JMS エンドポイント URI のフォーマットについて詳しくは、IBM MQ 製品資料の「 *IBM MQ Transport for SOAP* 」を参照してください。 サンプル・ソリューションおよびソースを変更する必要がある場合は、IDE (例えば、Microsoft Visual Studio 8 以降など) が必要になります。

手順

1. *QM1*
2. *SampleQ*
3. *SampleReplyQ*

4. リスナーが以下のメッセージを待つためにサービスを開始します。*MQ_INSTALLATION_PATH*\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリーです。
5. クライアントを 1 回実行: *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。

タスクの結果

クライアントが実行されると、以下のプロセスが開始され、4 回繰り返されて、合計で 5 つのメッセージが各方向に送信されます。

1. クライアントは、*SampleQ* に要求メッセージを書き込み、応答を待ちます。
2. サービスは *SampleQ* から要求メッセージを取得します。
3. サービスは、メッセージの内容を使用して値を加算および減算します。
4. サービスはその結果を *SampleReplyQ* 上のメッセージに書き込んで、クライアントが新規メッセージを書き込むのを待ちます。
5. クライアントは *SampleReplyQ* からメッセージを取得し、結果を画面上に表示します。

次のタスク

WCF クライアントから IBM MQ によってホストされる .NET サービスへの接続のサンプル

.NET と Java の両方について、サンプル・クライアント・アプリケーションおよびサンプル・サービス・プロキシー・アプリケーションが用意されています。 これらのサンプルは、株価の照会要求を受け取って、その株価を提供する Stock Quote サービスに基づいています。

始める前に

サンプルでは、.NET SOAP over JMS サービスのホスティング環境が IBM MQ に正しくインストールされ、構成されていて、ローカル・キュー・マネージャーからアクセス可能であることが必要です。

.NET SOAP over JMS サービスのホスティング環境が IBM MQ に正しくインストールおよび構成され、ローカル・キュー・マネージャーからアクセス可能になったら、追加の構成ステップを実行する必要があります。

1. **WMQSOAP_HOME** 環境変数を IBM MQ インストール・ディレクトリーに設定します (例: C:¥Program Files¥IBM¥MQ)。
2. Java コンパイラー javac が使用可能であり、PATH 上にあることを確認してください。
3. インストール・イメージの prereqs/axis ディレクトリーから IBM MQ 実動ディレクトリーにファイル axis.jar をコピーします (例: C:¥Program Files¥IBM¥MQ\java\lib\soap)。
4. PATH に追加します。*MQ_INSTALLATION_PATH*\Java\lib。ここで *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーを表します。例: C:¥Program Files¥IBM¥MQ
5. .NET のロケーションが *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd で正しく指定されていることを確かめてください。ここで *MQ_INSTALLATION_PATH* は IBM MQ がインストールされているディレクトリーを表します。例: C:¥Program Files¥IBM¥MQ。 .NET の場所は、例えば set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin のように指定できます。

前述のステップが完了したら、サービスをテストして実行します。

1. Soap over JMS 作業ディレクトリーにナビゲートします。
2. 以下のいずれかのコマンドを入力して、検証テストを実行し、サービス・リスナーを実行状態にします。
   - .NET の場合: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold 。ここで、 *MQ_INSTALLATION_PATH* は、 IBM MQ がインストールされているディレクトリーを表します。
   - AXIS の場合: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold 。ここで *MQ_INSTALLATION_PATH* は、 IBM MQ がインストールされているディレクトリーを表します。

hold 引数を指定すると、テストの完了後にリスナーが実行されたままになります。

この構成中にエラーが報告される場合は、以下の方法ですべての変更を除去すれば、手順を再開することができます。

1. 生成された SOAP over JMS ディレクトリーを削除します。
2. キュー・マネージャーを削除してください。

このタスクについて

このサンプルは、片方向チャネル形状を使用した、WCF クライアントから IBM MQ で提供される .NET SOAP over JMS サンプル・サービスへの接続を示しています。 このサービスは、テキスト・ストリングをコンソールに出力する単純な StockQuote の例を実装します。

クライアントは、1293 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』で説明しているように、 クライアント・ファイルを生成するために WSDL を使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\app.config ファイル内のクライアント・アプリケーション上で対応する値、および *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl ファイル内のサービス・アプリケーション上で対応する値も変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーを表します。 JMS エンドポイント URI のフォーマット設定について詳しくは、IBM MQ 製品資料の「*SOAP* のための *IBM MQ* トランスポート」を参照してください。

手順

クライアントを一度実行: *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリーを表します。
クライアント・アプリケーションは 5 回ループして、5 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得して、画面上に Hello World を 5 回表示します。

WCF クライアントから IBM MQ によってホストされる Axis Java サービスへの接続サンプル

Java と .NET の両方について、サンプル・クライアント・アプリケーションおよびサンプル・サービス・プロキシー・アプリケーションが用意されています。 これらのサンプルは、株価の照会要求を受け取って、その株価を提供する Stock Quote サービスに基づいています。

始める前に

このサンプルでは、.NET SOAP over JMS サービスのホスティング環境が IBM MQ に正しくインストールされ、構成されていて、ローカル・キュー・マネージャーからアクセス可能であることが必要です。

.NET SOAP over JMS サービスのホスティング環境が IBM MQ に正しくインストールおよび構成され、ローカル・キュー・マネージャーからアクセス可能になったら、追加の構成ステップを実行する必要があります。

1. **WMQSOAP_HOME** 環境変数を IBM MQ インストール・ディレクトリーに設定します (例: C:¥Program Files¥IBM¥MQ)。
2. Java コンパイラー javac が使用可能であり、PATH 上にあることを確認してください。
3. ファイル axis.jar を、インストール・イメージの prereqs/axis ディレクトリーから IBM MQ インストール・ディレクトリーにコピーします。
4. PATH に追加します。*MQ_INSTALLATION_PATH*\Java\lib。ここで *MQ_INSTALLATION_PATH* は、IBM MQ がインストールされているディレクトリーを表します。例: C:¥Program Files¥IBM¥MQ
5. .NET のロケーションが *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd で正しく指定されていることを確かめてください。ここで *MQ_INSTALLATION_PATH* は IBM MQ がインストールされているディレクトリーを表します。例: C:¥Program Files¥IBM¥MQ。 .NET の場所は、例えば set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin のように指定できます。

前述のステップが完了したら、サービスをテストして実行します。

1. Soap over JMS 作業ディレクトリーにナビゲートします。
2. 以下のいずれかのコマンドを入力して、検証テストを実行し、サービス・リスナーを実行状態にします。
   - .NET の場合: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold 。ここで、 *MQ_INSTALLATION_PATH* は、 IBM MQ がインストールされているディレクトリーを表します。
   - AXIS の場合: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold 。ここで *MQ_INSTALLATION_PATH* は、 IBM MQ がインストールされているディレクトリーを表します。

hold 引数を指定すると、テストの完了後にリスナーが実行されたままになります。

この構成中にエラーが報告される場合は、以下の方法ですべての変更を除去すれば、手順を再開することができます。

1. 生成された SOAP over JMS ディレクトリーを削除します。
2. キュー・マネージャーを削除してください。

このタスクについて

このサンプルは、片方向チャネル形状を使用した、IBM MQ で提供される WCF クライアントから Axis Java SOAP over JMS サンプル・サービスへの接続を示しています。 このサービスは、単純な StockQuote の例を実装します。この例では、現在のディレクトリーに保存されているファイルにテキスト・ストリングが出力されます。

クライアントは、1293 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』で説明しているように、 クライアント・ファイルを生成するために WSDL を使用して生成されています。

このサンプルは、このパラグラフで説明しているように、特定のリソース名を使用して構成されています。リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\app.config ファイルのクライアント・アプリケーションの対応する値、および *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.ws

dl ファイルのサービス・アプリケーションで、対応する値を変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーを表します。

手順

クライアントを一度実行: *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリーを表します。
クライアント・アプリケーションは 5 回ループして、5 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得して、現行ディレクトリーのファイルに Hello World を 5 回追加します。

WCF クライアントから WebSphere Application Server でホストされる Java サービスへのサンプル

WebSphere Application Server 6 には、サンプル・クライアント・アプリケーションおよびサンプル・サービス・プロキシー・アプリケーションが用意されています。 要求/応答サービスも用意されています。

始める前に

このサンプルでは、以下の IBM MQ 構成を使用する必要があります。

表 *195.* 必須の *IBM MQ* 構成

オブジェクト	**必須名**
キュー・マネージャー	QM1
ローカル・キュー	HelloWorld
ローカル・キュー	HelloWorldReply

このサンプルでは、WebSphere Application Server 6 がホスティングする環境を正しくインストールし、構成している必要もあります。 WebSphere Application Server 6 は、デフォルトでバインディング・モード接続を使用して IBM MQ に接続します。 そのため、WebSphere Application Server 6 は、キュー・マネージャーと同じマシンにインストールする必要があります。

WAS 環境を構成したら、以下の追加構成ステップを行う必要があります。

1. WebSphere Application Server JNDI リポジトリーに以下の JNDI オブジェクトを作成します。
   a. HelloWorld という名前の JMS キュー宛先
      - JNDI 名を jms/HelloWorld に設定します。
      - キュー名を HelloWorld に設定します。
   b. HelloWorldQCF という名前の JMS キュー接続ファクトリー
      - JNDI 名を jms/HelloWorldQCF に設定します。
      - キュー・マネージャー名を QM1 に設定します。
   c. WebServicesReplyQCF という名前の JMS キュー接続ファクトリー
      - JNDI 名を jms/WebServicesReplyQCF に設定します。
      - キュー・マネージャー名を QM1 に設定します。
2. 次の構成で HelloWorldPort というメッセージ・リスナー・ポートを WebSphere Application Server に作成します。
   - 接続ファクトリーの JNDI 名を jms/HelloWorldQCF に設定します。

- 宛先の JNDI 名を jms/HelloWorld に設定します。

3. 以下のようにして、Web サービスの HelloWorldEJBEAR.ear アプリケーションを WebSphere アプリケーション・サーバーにインストールします。
   a. 「**アプリケーション**」 > 「**新しいアプリケーション**」 > 「**新しいエンタープライズ・アプリケーション**」とクリックします。
   b. *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear へ移動します。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリです。
   c. ウィザードのデフォルト・オプションを何も変更せずに、 アプリケーションをインストールした後でアプリケーション・サーバーを再始動してください。

WAS の構成が完了したら、以下のようにサービスを一度実行してテストします。

1. Soap over JMS 作業ディレクトリーにナビゲートします。
2. コマンド *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\TestClient.exe を入力してサンプルを実行します。ここで、 *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。

このタスクについて

このサンプルは、WCF クライアントから、 IBM MQ に含まれている WCF サンプルで提供されている WebSphere Application Server SOAP over JMS サンプル・サービスへの、要求/応答チャネル・シェイプを使用した接続を示しています。 メッセージ・フローは、WCF と WebSphere Application Server の間を、IBM MQ キューを使用して送信されます。 このサービスは HelloWorld(...) メソッドを実装します。このメソッドは、ストリングを取り、クライアントにグリーティングを戻します。

クライアントは、『1292 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』で説明しているように、別個に公開された HTTP エンドポイントからサービス・メタデータを取得する svcutil ツールを使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\app.config ファイルのクライアント・アプリケーション、および *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear のサービス・アプリケーションで対応する値も変更する必要があります。ここで *MQ_INSTALLATION_PATH* は、IBM MQ のインストール・ディレクトリーです。

サービスおよびクライアントは、IBM Developer の記事「*Building a JMS web service using SOAP over JMS and WebSphere Studio*」で概説されているサービスおよびクライアントに基づいています。 IBM MQ WCF カスタム・チャネルと互換性のある SOAP over JMS Web サービスの開発について詳しくは、「https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html」を参照してください。

手順

クライアントを 1 回実行: *MQ_INSTALLATION_PATH* \tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで *MQ_INSTALLATION_PATH* は IBM MQ のインストール・ディレクトリーです。

クライアント・アプリケーションは、両方のサービス・メソッドを同時に開始して、 2 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得し、クライアント・アプリケーションがコンソールに出力する HelloWorld(...) メソッド呼び出しへの応答を提供します。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

IBM 本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
日本アイ・ビー・エム株式会社
法務・知的財産
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
〒 103-8510
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 INTERNATIONAL BUSINESS MACHINES CORPORATION は、法律上の瑕疵担保責任、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 "" 国または地域によっては、法律の強行規定により、保証責任の制限が 禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものでは ありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用すること ができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他の ライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で 決定されたものです。 そのため、他の操作環境で得られた結果は、異なる可能性があります。 一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。 実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、 もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。 したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、 あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、 名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性が あることをほのめかしたり、保証することはできません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

プログラミング・インターフェース情報 (提供されている場合) は、このプログラムで使用するアプリケーション・ソフトウェアの作成を支援することを目的としています。

本書には、プログラムを作成するユーザーが WebSphere MQ のサービスを使用するためのプログラミング・インターフェースに関する情報が記載されています。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。 診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

重要 : この診断、修正、およびチューニング情報は、変更される可能性があるため、プログラミング・インターフェースとして使用しないでください。

商標

IBM、 IBM ロゴ、 ibm.com®は、世界の多くの国で登録された IBM Corporation の商標です。 現時点での IBM の商標リストについては、 "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml をご覧ください。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds 氏の米国およびその他の国における登録商標です。

この製品には、 Eclipse Project (https://www.eclipse.org/) により開発されたソフトウェアが含まれています。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

部品番号:

(1P) P/N: