7.5

IBM WebSphere MQ 用のアプリケーションの開発

注記

本書および本書で紹介する製品をご使用になる前に、1129 ページの『特記事項』に記載されている情報をお読みください。

本書は、 IBM® WebSphere® MQ バージョン 7 リリース 5、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

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

目次

アプリケーションの開発 7
アプリケーション開発の概念 8
MQI を使用するアプリケーション・プログラム 9
IBM WebSphere MQ メッセージ 9
Microsoft Transaction Server アプリケーションの準備と実行 40
IBM WebSphere MQ を WebSphere Application Server で使用する 40
トランザクション・サポートのシナリオ 41
使用する言語の決定 78
IBM WebSphere MQ データ定義ファイル 80
C によるコーディング 82
COBOL によるコーディング 85
pTAL によるコーディング 86
Visual Basic によるコーディング 87
IBM WebSphere MQ オブジェクト・モデル 87
JMS または Java の使用 89
IBM WebSphere MQ アプリケーションの設計 89
メッセージの設計 92
アプリケーション設計とパフォーマンス 93
IBM WebSphere MQ の高度な技法 94
IBM WebSphere MQ プログラムのサンプル 96
分散プラットフォームにおけるサンプル・プログラム 96
キューイング・アプリケーションの作成 193
Message Queue Interface の概要 194
キュー・マネージャーへの接続とキュー・マネージャーからの切断 206
オブジェクトのオープンとクローズ 214
キューへのメッセージの書き込み 224
キューからのメッセージの読み取り 240
パブリッシュ/サブスクライブ・アプリケーションの作成 277
オブジェクト属性の照会と設定 321
作業単位のコミットとバックアウト 324
トリガーによる IBM WebSphere MQ アプリケーションの開始 330
MQI とクラスターの処理 348
クライアント・アプリケーションの作成 353
クライアント・アプリケーションのメッセージ・キュー・インターフェース (MQI) の使用 354
IBM WebSphere MQ MQI クライアント用のアプリケーションの作成 359
IBM WebSphere MQ MQI クライアント環境でのアプリケーションの実行 361
CICS および Tuxedo アプリケーションの準備と実行 373
Microsoft Transaction Server アプリケーションの準備と実行 375
IBM WebSphere MQ JMS アプリケーションの準備と実行 375
ユーザー出口、API 出口、およびインストール可能サービス 376
出口とインストール可能サービスの作成とコンパイル 376
IBM WebSphere MQ アプリケーションの構築 429
AIX でのアプリケーションの構築 430
HP Integrity NonStop Server でのアプリケーションの構築 436
HP-UX でのアプリケーションの構築 442
Linux でのアプリケーションの構築 448
Solaris でのアプリケーションの構築 454
Windows システムでのアプリケーションの構築 461
IBM WebSphere MQ for Windows での LDAP サービスの使用 468
IBM WebSphere MQ Telemetry アプリケーションの開発 475
IBM WebSphere MQ Telemetry サンプル・プログラム 475
Java を使用した最初のパブリッシャーの作成 478

Java を使用した非同期パブリッシャーの作成 484
Java を使用したリカバリー可能な非同期パブリッシャーの作成 489
Java を使用したサブスクライバーの作成 495
JAAS を使用する MQTT クライアントの認証 500
自己署名証明書を使用する SSL 接続の認証 506
証明書チェーンを使用する SSL 接続の認証 511
C を使用した最初のパブリッシャーの作成 515
C を使用した非同期パブリッシャーの作成 519
C を使用したサブスクライバーの作成 523
クライアント・プログラミングの概念 528
C クライアント・プログラミングの概念 549
プログラム・エラーの処理 553
ローカルで判別されたエラー 553
問題判別用の報告メッセージの使用 554
リモートで判別されたエラー 555
マルチキャスト・プログラミング 557
マルチキャストと Message Queue Interface 557
キュー・マネージャーへのマルチキャスト接続 560
マルチキャスト・メッセージングのデータ変換のプログラミング 560
マルチキャスト例外報告 561
.NET の使用 564
IBM WebSphere MQ classes for .NET の概要 565
IBM WebSphere MQ.NET プログラムの作成およびデプロイ 580
Microsoft Windows Communication Foundation (WCF) 用の IBM WebSphere MQ カスタム・チャネル 599
.NET 3 を使用した WCF 用の IBM WebSphere MQ カスタム・チャネルの紹介 600
WCF 用の IBM WebSphere MQ カスタム・チャネルの使用 604
WCF サンプルの使用 621
IBM WebSphere MQ 用の WCF カスタム・チャネルの問題判別 627
C++ の使用 634
サンプル・プログラム 637
C++ 言語の概念 641
C++ でのメッセージング 645
IBM WebSphere MQ C++ プログラムの作成 652
IBM WebSphere MQ classes for Java の使用 658
IBM WebSphere MQ classes for Java の概要 659
IBM WebSphere MQ classes for Java のインストールと構成 661
プログラマー向けの概要 673
IBM WebSphere MQ classes for Java アプリケーションの作成 674
IBM WebSphere MQ classes for JMS の使用 721
IBM WebSphere MQ classes for JMS の概要 723
IBM WebSphere MQ classes for JMS のインストールと構成 724
プログラマー向けの概要 801
IBM WebSphere MQ classes for JMS アプリケーションの作成 809
アプリケーション・サーバー機構 (ASF) 930
IBM WebSphere MQ JMS 管理ツールの使用 938
IBM WebSphere MQ Explorer for JMS 構成の使用 946
WebSphere MQ Headers パッケージの使用 947
WebSphere MQ classes for Java での使用 948
WebSphere MQ classes for JMS との併用 948
IBM WebSphere MQ での Web サービスの使用 950
IBM WebSphere MQ transport for SOAP 950
IBM WebSphere MQ bridge for HTTP 1028
Component Object Model インターフェース (IBM WebSphere MQ Automation Classes for ActiveX) の使用 1038
IBM WebSphere MQ Automation Classes for ActiveX を使用した設計とプログラミング 1039
IBM WebSphere MQ Automation Classes for ActiveX リファレンス 1044
トラブルシューティング 1110

MQAI との間の ActiveX インターフェース 1115
IBM WebSphere MQ Automation Classes for ActiveX スターター・サンプルについて 1124

特記事項 1129
プログラミング・インターフェース情報 1130
商標 1130

アプリケーションの開発

IBM WebSphere MQ には、ビジネス・プロセスをサポートするために必要なメッセージを送受信するアプリケーションを開発するためのいくつかの手段が用意されています。 キュー・マネージャーや関連リソースを管理するためのアプリケーションを開発することもできます。

IBM WebSphere MQ のアプリケーションを開発する前に、IBM WebSphere MQ 技術概要 IBM WebSphere MQ 技術概要で取り上げられている概念をよく理解するようにしてください。

さまざまなプログラミング言語で IBM WebSphere MQ のアプリケーションを開発できます。 サポートされているプログラミング言語とそれぞれの特色については、78 ページの『使用するプログラミング言語の決定』を参照してください。

以下の各セクションでは、それぞれのプラットフォームの場合に作成できる IBM WebSphere MQ のアプリケーションのタイプをまとめています。

z/OS 以外のプラットフォームの IBM WebSphere MQ

ここでは、 IBM WebSphere MQ で作成できるアプリケーションのタイプについて説明します。

IBM WebSphere MQ の製品はキュー・マネージャーとアプリケーション・イネーブラーです。 各製品は、IBM メッセージ・キュー・インターフェース (MQI) をサポートしており、プログラムは、これを通じてキューにメッセージを書き込んだり、キューからメッセージを読み取ったりすることができます。

非 z/OS プラットフォーム上の IBM WebSphere MQ では、次のようなアプリケーションを作成できます。

- 同じオペレーティング・システム下で実行中の他のアプリケーションにメッセージを送信する。 アプリケーションは、同じシステム上と別のシステム上のどちらにあっても構いません。
- 他の IBM WebSphere MQ プラットフォーム上で実行されるアプリケーションにメッセージを送信する。
- CICS® for TXSeries® for AIX®、TXSeries for HP-UX、TXSeries for Solaris、および TXSeries for Windows システム・アプリケーションからメッセージ・キューイングを使用する。
- Encina for AIX、HP-UX、Solaris、および Windows システムからメッセージ・キューイングを使用する。
- Tuxedo for AIX、AT & T、 HP-UX、Solaris、および Windows システム内からメッセージ・キューイングを使用します。
- IBM WebSphere MQ をトランザクション・マネージャーとして使用し、IBM WebSphere MQ の作業単位で外部リソース・マネージャーによって行われた更新を調整する。 以下の外部リソース管理プログラムがサポートされており、X/Open XA インターフェースに準拠しています。
  - DB2®
  - Informix®
  - Oracle
  - Sybase
- いくつかのメッセージを、コミットまたはバックアウトすることのできる 1 つの作業単位として、まとめて処理する。
- 全機能を使用できる IBM WebSphere MQ 環境、または次に示すプラットフォーム上の IBM WebSphere MQ MQI クライアント環境から実行する。
  - UNIX and Linux® システム
  - Windows

関連概念

機密保護

アプリケーション開発の概念

選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

IBM WebSphere MQ アプリケーションの設計と作成を開始する前に、IBM WebSphere MQ の基本概念をよく理解して、技術概要のトピックを参照してください。 IBM WebSphere MQ 用に作成できるアプリケーションのタイプについては、 7 ページの『アプリケーションの開発』を参照してください。

アプリケーション開発に固有の IBM WebSphere MQ の概念については、以下のリンクを使用してください。

- 9 ページの『IBM WebSphere MQ メッセージ』
- Point-to-Point メッセージング
- WebSphere MQ パブリッシュ/サブスクライブ・メッセージングの紹介
- 354 ページの『クライアント・アプリケーションでのメッセージ・キュー・インターフェース (MQI) の使用』
- 950 ページの『WebSphere MQ での Web サービスの使用』
- 398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』
- 41 ページの『トランザクション・サポートのシナリオ』

MQI を使用するアプリケーションを実行できるようになるには、まず特定の IBM WebSphere MQ オブジェクトを作成する必要があります。 詳しくは、9 ページの『MQI を使用するアプリケーション・プログラム』を参照してください。

関連概念

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

96 ページの『WebSphere MQ プログラムのサンプル』
この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

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

353 ページの『クライアント・アプリケーションの作成』
WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

721 ページの『WebSphere MQ classes for JMS の使用』
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) は、 WebSphere MQ に付属する JMS プロバイダーです。 javax.jms パッケージで定義されたインターフェースの実装に加えて、WebSphere MQ classes for JMS は JMS API への 2 セットの拡張機能を提供します。

1038 ページの『Component Object Model インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用』
WebSphere MQ Automation Classes for ActiveX (MQAX) は ActiveX コンポーネント群で、 WebSphere MQ にアクセスするためにアプリケーションで使用できるクラスを提供します。

658 ページの『WebSphere MQ classes for Java の使用』
WebSphere MQ classes for Java を使用すると、Java 環境で WebSphere MQ を使用できます。 Java アプリケーションは、 WebSphere MQ classes for Java または WebSphere MQ classes for JMS のいずれかを使用して、 WebSphere MQ リソースにアクセスできます。

564 ページの『.NET の使用』
.NET プログラミング・フレームワークで作成されたプログラムは、WebSphere MQ classes for .NET によって、WebSphere MQ に WebSphere MQ MQI クライアントとして接続するか、または WebSphere MQ サーバーに直接接続することができます。

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

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

MQI を使用するアプリケーション・プログラム

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

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

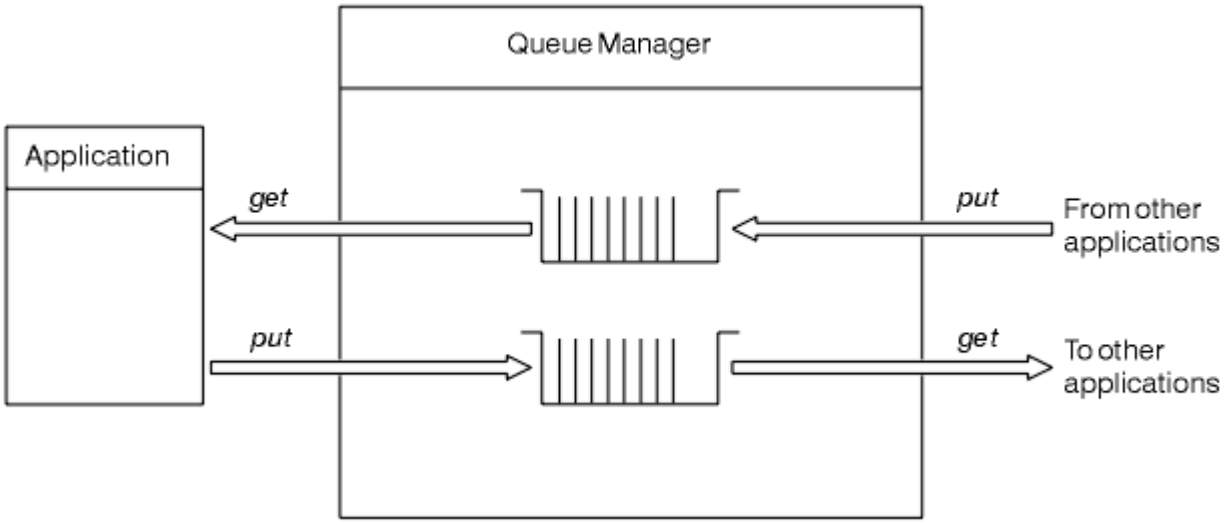


図 1. キュー、メッセージ、およびアプリケーション

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

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

- キュー・マネージャーが存在しており、実行されている。
- 最初のアプリケーション・キュー (ここからメッセージが取り出される) が定義されている。
- 2 番目のキュー (アプリケーションはここにメッセージを書き込む) も定義されている。
- アプリケーションが、キュー・マネージャーに接続可能である。 このためには、アプリケーションは IBM WebSphere MQ にリンクされている必要があります。 429 ページの『IBM WebSphere MQ アプリケーションの構築』を参照してください。
- 最初のキューにメッセージを書き込むアプリケーションも、キュー・マネージャーに接続する。 また、それらのアプリケーションがリモートである場合も、伝送キューとチャネルは共にセットアップされている。 9 ページの図 1 には、システムのこの部分は示されていません。

IBM WebSphere MQ メッセージ

この情報では、IBM WebSphere MQ メッセージの概念、 メッセージの各部分、およびメッセージ記述子について紹介します。

IBM WebSphere MQ のメッセージは次の 2 つの部分で構成されます。

- メッセージ・プロパティー
- アプリケーション・データ

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

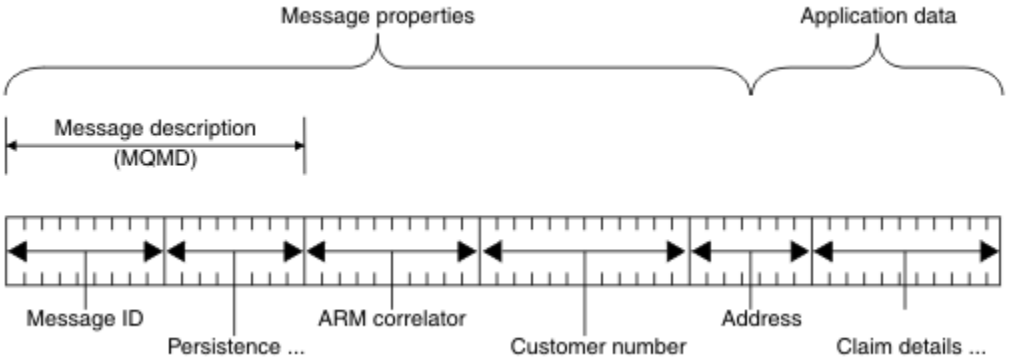


図 2. メッセージの表現

WebSphere MQ メッセージで 運ばれるアプリケーション・データは、データ変換がそれに対して行われない限り、キュー・マネージャーによって変更されることはありません。 また、WebSphere MQ では、 このデータの内容についていかなる制限も課されません。 各メッセージ内のデータの長さは、 キューとキュー・マネージャーのいずれの *MaxMsgLength* 属性の値を超えることもできません。

WebSphere MQ for AIX、 WebSphere MQ for HP-UX、 WebSphere MQ for Linux、 WebSphere MQ for Solaris、 WebSphere MQ for Windows の場合、 *MaxMsgLength* のデフォルトは 100 MB (104 857 600 バイト) です。

一部の環境では、メッセージを *MaxMsgLength* 属性の値より少し短くします。 詳しくは、229 ページの『メッセージ内のデータ』を参照してください。

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

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

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

メッセージの発信元に関する情報が含まれる MQMD 内のフィールドを使用する方法については、38 ページの『メッセージ・コンテキスト』を参照してください。

メッセージ記述子にはさまざまなバージョンがあります。 メッセージのグループ化とセグメント化に関する追加情報 (35 ページの『メッセージ・グループ』を参照) は、メッセージ記述子 (または MQMDE) のバージョン 2 で提供されます。 これは、追加のフィールドがあることを除いてバージョン 1 のメッセージ記述子と同じです。 これについては、MQMDE - 拡張メッセージ記述子を参照してください。

メッセージのタイプ

IBM WebSphere MQ によって定義されるメッセージには、4 つのタイプがあります。

これら 4 つのメッセージは次のとおりです。

- データグラム
- 要求メッセージ
- 応答メッセージ
- 報告メッセージ

- 報告メッセージのタイプ
- 報告メッセージのオプション

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

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

データグラム

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

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

要求メッセージ

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

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

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

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

応答メッセージ

他のメッセージに応答するときは、応答メッセージ を使用します。

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

レポート・メッセージ

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

報告メッセージは、以下により生成できます。

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

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

報告メッセージのタイプ

キューにメッセージを書き込む場合、次の受信を選択できます。

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

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

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

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

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

報告メッセージのオプション

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

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

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

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

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

- 元のメッセージの *MsgId* または *CorrelId* のいずれかを報告メッセージの *CorrelId* フィールドにコピーするように要求できます。 デフォルトのアクションでは、メッセージ ID がコピーされます。 MQRO_COPY_MSG_ID_TO_CORRELID を使用すると、メッセージの送信側は応答または報告メッセージを元のメッセージと相関できます。 応答メッセージまたは報告メッセージの相関 ID は、元のメッセージのメッセージ ID と同じになります。
- 報告メッセージに対して新規の *MsgId* を生成するか、または元のメッセージの *MsgId* を報告メッセージの *MsgId* フィールドにコピーするかのどちらかを要求できます。 デフォルトのアクションでは新規のメッセージ ID が生成されます。 MQRO_NEW_MSG_ID を使用すると、システムにある各メッセージに異なるメッセージ ID が与えられ、それぞれのメッセージをシステム内の他のすべてのメッセージと明確に区別できるようになります。
- 特殊なアプリケーションは MQRO_PASS_MSG_ID または MQRO_PASS_CORREL_ID を使用する必要が生じるかもしれません。 しかし、例えば、同じメッセージ ID のある複数のメッセージがキューに入っている場合などに、キューからメッセージを読み取るアプリケーションが正しく機能するように設計する必要があります。

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

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

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

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

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

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

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

報告とセグメント化されたメッセージ

WebSphere MQ for z/OS® ではサポートされない。

メッセージがセグメント化された場合 (メッセージのセグメント化については 262 ページの『メッセージのセグメント化』 を参照)、報告を生成するように求めると、メッセージがセグメント化されていないときよりも多くの報告を受け取ることがあります。

WebSphere MQ によって生成される報告の場合

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

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

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

アプリケーションによって生成される報告の場合

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

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

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

- MQMDE
- MQDLH
- MQXQH
- MQIIH
- MQH*

MQH* は、文字 MQH で始まるあらゆる名前を表します。

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

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

報告の取得

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

MQGMO_COMPLETE_MSG が指定された MQGET は、1 つの完全な元のメッセージを表すのに十分な数のレポート・メッセージ (COA などの単一タイプで、同じ *GroupId* を持つもの) がキューに存在する場合に満足されます。 これは、報告メッセージ自体に元のデータが完全には含まれていないときにも言えることです。各報告メッセージ内の *OriginalLength* フィールドは、その報告メッセージが表す元のデータの長さを示します (これは、データ自体が存在しなくても同様です)。

MQGMO_COMPLETE_MSG を指定した MQGMO_COMPLETE_MSG を指定した MQGET は同じ *Feedback* コードを持っている場合のみレポート・メッセージを再アセンブルするため、キューに複数の異なるレポート・タイプ (例えば、COA と COD の両方) が存在する場合でも、この手法を使用できます。 ただし、通常、例外レポートにこの手法を使用することはできません。一般的に、例外レポートの *Feedback* コードが異なるためです。

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

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

次に 1 つの方法を示します。

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

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

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

バックレベルのキュー・マネージャー

セグメンテーションをサポートするキュー・マネージャーによってレポートが生成されるが、セグメンテーションをサポート しない キュー・マネージャーで受信される場合、ゼロ、100 バイト、またはメッセージ内の元のデータすべてに加えて、レポート・データには常に MQMDE 構造体 (レポートによって表される *Offset* および *OriginalLength* を識別する) が含まれます。

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

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

メッセージ制御情報およびメッセージ・データの形式

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

メッセージ制御情報の形式

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

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

メッセージ・データの形式

次のいずれかを指定できます。

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

上記を指定するには、以下のフィールドを使用します。

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

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

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

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

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

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

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

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

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

アプリケーション・データの変換

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

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

注 : MQFMT_NONE が指定されたメッセージは変換されません。

送信側のキュー・マネージャーでの変換

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

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

組み込み形式

これには以下が含まれます。

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

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

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

アプリケーション定義の形式

ユーザー定義の形式の場合、アプリケーションのデータ変換は、データ変換出口プログラムで行う必要があります (詳細は、416 ページの『データ変換出口の作成』を参照してください)。 クライアント / サーバー環境では、出口プログラムはサーバーでロードされ、そこで変換が行われます。

受信側のキュー・マネージャーでの変換

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

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

コード化文字セット

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

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

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

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

複数のプラットフォームにまたがるアプリケーションを作成するときは、アプリケーション・データの変換、形式名、およびユーザー出口を考慮する必要があります。 データ変換出口の呼び出し方法と書き込み方法については、416 ページの『データ変換出口の作成』を参照してください。

メッセージ優先順位

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

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

キューの *DefPriority* 属性は、そのキューに入れられるメッセージのデフォルト優先順位の値を設定します。 この値は、キューの作成時に設定されますが、後で変更可能です。 別名キューおよびリモート・キューのローカル定義は、それらが解決される基本キューとは異なるデフォルト優先順位を持つ可能性があります。 解決経路 (216 ページの『名前の解決』を参照) に複数のキュー定義がある場合、デフォルト優先順位には、オープン・コマンドで指定されるキューの *DefPriority* 属性の (書き込み操作時の) 値が使用されます。

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

メッセージ・プロパティー

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

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

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

メッセージ・プロパティーとメッセージ長

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

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

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

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

プロパティーが含まれるバッファーの場合、 アプリケーションは *MaxMsgLength* の値より大きいバッファーを使用してメッセージを正常に書き込むことができます。 なぜなら、MQRFH2 エレメントとして表されている場合であっても、メッセージ・プロパティーはメッセージの長さにカウントされないからです。

MQRFH2 ヘッダー・フィールド がプロパティーの長さに加えられるのは、1 つ以上のフォルダーが含まれていて、ヘッダー内のすべてのフォルダーが プロパティーを含んでいる場合のみです。 MQRFH2 ヘッダーに 1 つ以上のフォルダーが含まれていて、 どのフォルダーもプロパティーを含んでいない場合、代わりに MQRFH2 ヘッダー・フィールドが メッセージ長に加算されます。

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

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

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

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

バージョン 7.0 より前の製品のキュー・マネージャーに流れる場合、メッセージのプロパティーは (メッセージ記述子内のものを除き) メッセージの長さにカウントされます。 したがって、必要に応じ、バージョン 7.0 より前のシステムに向かうチャネルの *MaxMsgLength* 属性の値を大きくして、 各メッセージについてより多くのデータを送信できるように補償してください。 あるいは、キューまたはキュー・マネージャーの *MaxMsgLength* を小さくして、システム中で送信されるデータの全体のレベルを一定に保つこともできます。

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

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

プロパティー名

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

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

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

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

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

すべてのプロパティー名は、Java ID の Java 言語仕様で定義されている規則に従う必要があります。ただし、Unicode 文字 U+002E (.) は名前の一部として許可されますが、開始は許可されません。 Java ID の規則は、プロパティー名の JMS 仕様に含まれている規則と同等です。

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

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

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

- `mcd`
- `jms`
- `usr`
- `mq`
- `sib`
- `wmq`
- `Root`
- `Body`
- `Properties`

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

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

プロパティー名の制約事項

プロパティーに名前を付けるときには、一定の規則を守らなければなりません。

以下の制約事項が、プロパティー名に適用されます。

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

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

プロパティー	**(の) 同義語**
JMSCorrelationID	ルート .MQMD.CorrelId または jms.Cid
JMSDeliveryMode	ルート .MQMD.Persistence または jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	ルート .MQMD.Expiry または jms.Exp
JMSMessageID	ルート .MQMD.MsgId
JMSPriority	ルート .MQMD.Priority または jms.Pri
JMSRedelivered	ルート .MQMD.BackoutCount
JMSReplyTo (URI としてエンコードされたストリング)	ルート .MQMD.ReplyToQ または Root .MQMD.ReplyToQMgr または jms.Rto
JMSTimestamp	ルート .MQMD.PutDate または Root .MQMD.PutTime または jms.Tms
JMSType	mcd.Type または mcd.Set または mcd.Fmt

プロパティー	(の) 同義語
JMSXAppID	ルート .MQMD.PutApplName
JMSXDeliveryCount	ルート .MQMD.BackoutCount
JMSXGroupID	ルート .MQMD.GroupId または jms.Gid
JMSXGroupSeq	ルート .MQMD.MsgSeqNumber または jms.Seq
JMSXUserID	ルート .MQMD.UserIdentifier

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

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

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

これらの の制約事項は、以下のように検証メカニズムによって実施されます。

- メッセージ・ハンドルの作成時に妥当性検査が要求された場合、 MQSETMP - メッセージ・プロパティーの設定 呼び出しを使用してプロパティーを設定すると、プロパティー名が妥当性検査されます。 プロパティーの妥当性検査の試行が行われ、プロパティー名の指定にエラーがあるために失敗した場合、完了コードは MQCC_FAILED で、理由は以下のとおりです。
  - 理由 1 から 4 の場合、MQRC_PROPERTY_NAME_ERROR
  - 理由 5 の場合、MQRC_MIXED_CONTENT_NOT_ALLOWED
- MQRFH2 エレメントとして直接指定されたプロパティーの名前については、MQPUT 呼び出しにより妥当性検査が行われるとは限りません。

プロパティーとしてのメッセージ記述子フィールド

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

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

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

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

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

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

キュー・マネージャーに受け入れられる MQMDE でメッセージ・データが始まる場合、Root.MQMD.<Field> 表記を使って MQMDE フィールドにアクセスできます。 この場合 MQMDE フィールドは、プロパティーの観点で論理上 MQMD の一部として扱われます。 MQMDE の概要のセクション『MQPUT 呼び出しおよび MQPUT1 呼び出しで MQMDE を指定する場合』を参照してください。

プロパティーのデータ型と値

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

プロパティー値のデータ型は、以下のいずれかの値でなければなりません。

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

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

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

キューからのメッセージの選択

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

Selectors

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

MQSUB および MQOPEN 関数呼び出しを使用する選択

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

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

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

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

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

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

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

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

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

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

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

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

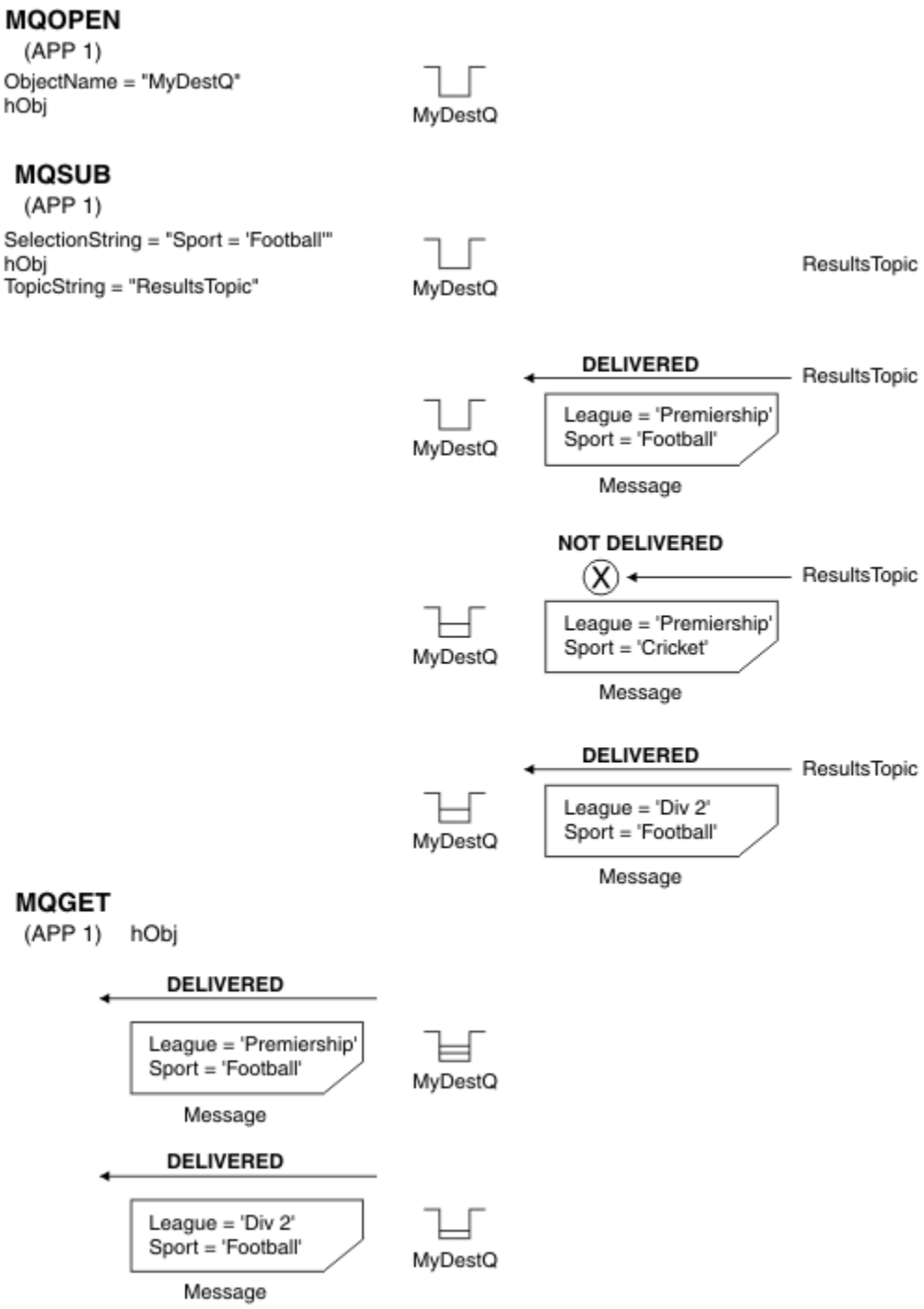


図 *3. MQSUB* 呼び出しを使用する選択

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

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

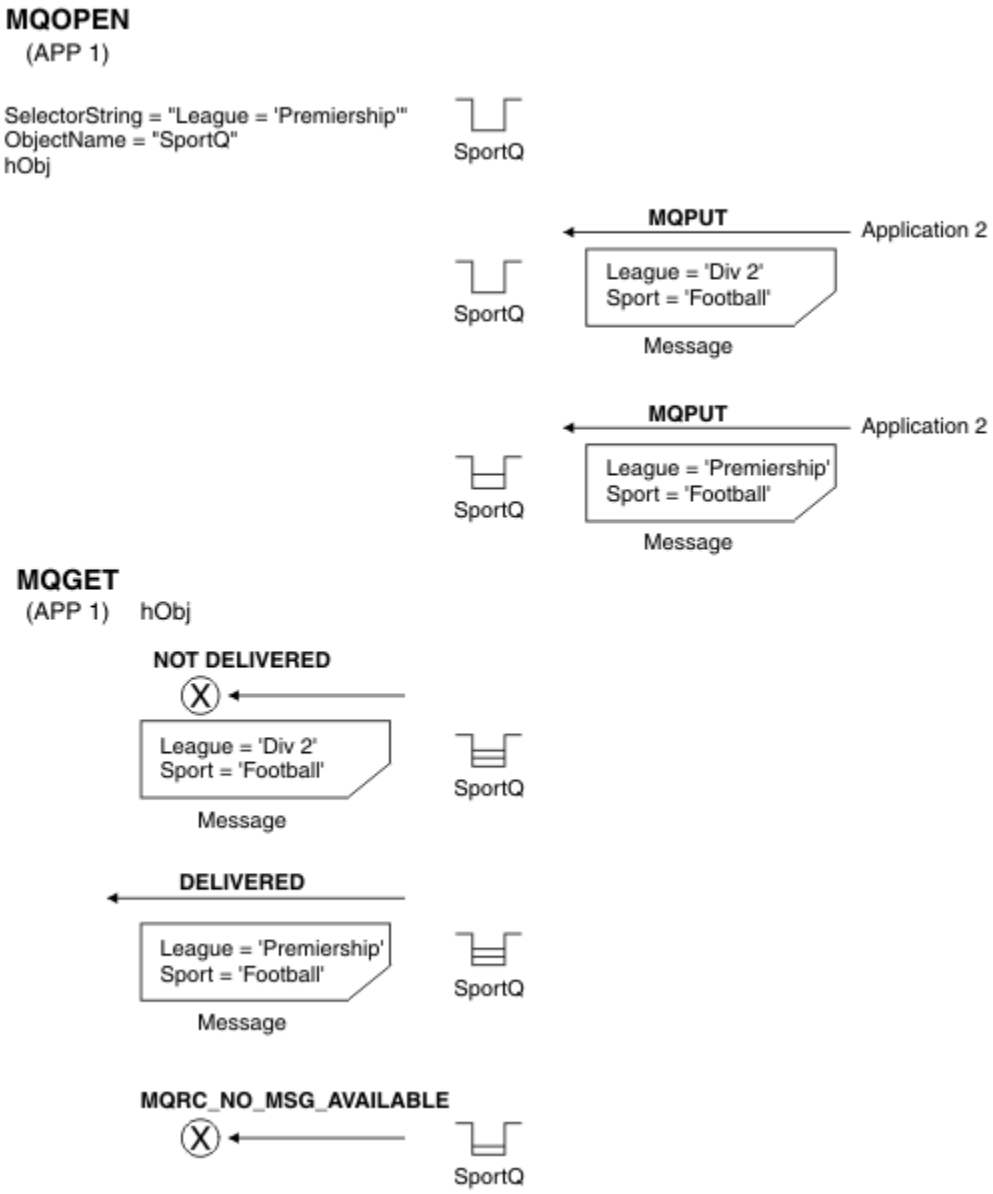


図 *4. MQOPEN* 呼び出しを使用する選択

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

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

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

選択動作
IBM WebSphere MQ 選択動作の概要。

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

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

メッセージ・プロパティーに対するマッチングが行われる前に、選択ストリングが TRUE または FALSE のいずれかに解決される可能性があります。 例えば、選択文字列が "TRUE <>FALSE"に設定されている場合などです。 こうした早期の評価の実行が保証されるのは、選択ストリング内にメッセージ・プロパティー参照がない場合のみです。

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

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

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

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

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

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

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

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

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

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

メッセージング・パフォーマンス

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

選択が JMSCorrelationID または JMSMessageID に基づいている場合にディープ・キューのメッセージ選択を最適化するには、 JMSCorrelationID = ... または JMSMessageID = ... の形式の選択ストリングを使用し、1 つのプロパティーのみを参照します。

この方法により、JMSCorrelationID での選択のパフォーマンスが著しく向上します。JMSMessageID の場合は、パフォーマンスがわずかに向上します。

複雑なセレクターの使用

セレクターに多数のコンポーネントが含まれる場合があります。例:

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

このような複雑なセレクターを使用すると、パフォーマンスへの影響が深刻になったり、リソース要件が過大になったりする場合があります。 したがって、IBM WebSphere MQ は、あまりにも複雑でシステム・リソース不足を招き得るセレクターを処理しないことで、システムを保護します。 一部のプラットフォームでは約 100 回のテストの後に保護が発生することがあるので、 この数に近いコンポーネントを含むセレクターを使用すると、障害が生じる可能性があります。 多数のコンポーネントを含むセレクターを使用する場合は、保護の制限に達しないことを確認するために、適切なプラットフォームで十分に試行とテストを行うことをお勧めします。

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

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

関連概念

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

メッセージの内容の選択
選択したメッセージ・ペイロードの内容 (コンテンツ・フィルタリング) に基づいてサブスクライブすることができますが、そのようなサブスクリプションにどのメッセージを送信するかを、WebSphere MQ で直接判断することはできません。 このようなメッセージを処理するには、代わりに、IBM Integration Bus などの拡張メッセージ選択プロバイダーが必要です。

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

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

WebSphere MQ は、メッセージ・セレクターが提示された時点で、その構文が正しいかどうか を検証します。 選択ストリングの構文が正しくないか、 プロパティー名が有効でない場合で、かつ拡張メッセージ選択プロバイダーが使用可能でない場合に、 アプリケーションに MQRC_SELECTION_NOT_AVAILABLE が戻されます。 選択ストリングの構文が正しくないか、 プロパティー名が有効でない場合、サブスクリプションが再開されると、 アプリケーションに MQRC_SELECTOR_SYNTAX_ERROR が戻されます。 プロパティーが設定されたときに (MQCMHO_VALIDATE の代わりに MQCMHO_NONE を設定することによって) プロパティー名の妥当性検査が 使用不可に設定されていた場合、その後にアプリケーションが、無効なプロパティー名を使うメッセージを書き込むと、 そのメッセージは決して選択されることはありません。

セレクターには、以下のものを含めることができます。

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

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

- "myBytes = "0x0AFC23"" = TRUE

以下のストリング選択は一致しません。

- "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (バイトの数が 2 の倍数 でないため)
- "myBytes = "0x0AFC2300"" = FALSE (比較では後続ゼロは意味がある ため)
- "myBytes = "0x000AFC23"" = FALSE (比較では先行ゼロは意味がある ため)
- "myBytes = "0x23FC0A"" = FALSE (エンディアン種別は考慮されないため)

– 16 進数はゼロで始まり、大文字または小文字の x が続きます。 リテラルの残りの部分は、1 つ以上の有効な 16 進数文字となります。 例は、0xA、0xAF、0X2020 です。

– 先行ゼロの後に 0-7 の範囲の 1 つ以上の数字が続いているものは、 常に 8 進数の開始と解釈されます。 接頭部がゼロの 10 進数で表すことはできません。例えば 09 の場合、9 は有効な 8 進数字ではないため構文エラーが戻されます。 8 進数字の例は、0177、0713 です。

– 正確な数字リテラルは、57、 -957、+62 など、小数点なしの数値です。 正確な数字リテラルには、末尾に大文字または小文字の L を付けることができます。これが数値の格納または解釈の方法に影響を与えることはありません。 WebSphere MQ は、-9,223,372,036,854,775,808 から 9,223,372,036,854,775,807 までの範囲の正確な数字をサポートします。

– 近似数値リテラルは、7E3 または -57.9E2 などの浮動小数における数値、または 7.、-95.7、または +6.2 などの小数部を持つ数値です。 WebSphere MQ は、-1.797693134862315E+308 から 1.797693134862315E+308 までの範囲の数値をサポートします。

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

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

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

– ブール・リテラルの TRUE および FALSE。

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

- ID:

ID は可変長文字シーケンスで、必ず有効な ID 開始文字で始まり、ゼロまたはそれ以上の有効な ID 部分文字が続きます。 ID 名の規則は、メッセージ・プロパティー名の規則と同じです。詳細については、19 ページの『プロパティー名』 および 20 ページの『プロパティー名の制約事項』 を参照してください。

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

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

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

プロパティー名または MQMD フィールド名にマップする JMS フィールド名およびプロパティー名も、選択ストリングでの有効な ID です。 WebSphere MQ は、認識された JMS フィールドおよびプロパティー名 をメッセージ・プロパティー値にマップします。 詳しくは、812 ページの『JMS のメッセージ・セレクター』を参照してください。 例えば、選択ストリング "JMSPriority >=" は、現行メッセージの jms フォルダーにある 「1 次」 プロパティーを選択します。

- オーバーフロー/アンダーフロー:

  10 進数および近似数値の両方で、 以下は定義されていません。

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

  これらの条件は検査されません。

- 空白文字:

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

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

- 式:
  – セレクターは、条件式です。 true に評価されるセレクターは一致し、false または unknown に評価されるセレクターは一致しません。
  – 演算式は、それ自体と、算術演算、ID (ID 値は数値リテラルとして扱われる)、および数値リテラルから構成されています。
  – 条件式は、それ自体と、比較演算、および論理演算から構成されています。
- 標準の括弧 () (式が評価される順序を設定する) がサポートされています。
- 論理演算子 (優先順位どおりに列挙): NOT、AND、OR。
- 比較演算子: =、 >、 >=、 <、 <=、 <> (等しくない)。
  – 2 つのバイト・ストリングが等しいのは、両ストリングが同じ長さで、バイトのシーケンスが等しい場合のみ です。
  – 同じタイプの値のみを比較できます。 1 つの例外として、正確な数値と近似数値を比較することは有効です (必要な型変換は、Java 数値プロモーションの規則によって定義されます)。 異なるタイプを比較する試みがある場合、セレクターは常に false です。
  – ストリングとブールの比較は、= および <> に制限されます。 2 つのストリングは、それらのストリングに含まれている文字シーケンスがまったく同じ場合にのみ等しくなります。
- 算術演算子 (優先順位どおりに列挙):

  - 単項 +、-。
  - 乗算 *、および除算 /。
  - 加算 +、および減算 -。
  - ヌル値での算術演算はサポートされていません。 ヌル値での算術演算が試行される場合、完全セレクターは常に false です。
  - 算術演算では、Java 数値プロモーションを使用する必要があります。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比較演算子:
  - `Age BETWEEN 15 and 19` は `age >= 15 AND age <= 19` と同等です。
  - `Age NOT BETWEEN 15 and 19` は `age < 15 OR age > 19` と同等です。
  - BETWEEN 演算の式のいずれかがヌルである場合、演算の値は false です。 NOT BETWEEN 演算の式のいずれかが NULL である場合、演算の値は true です。
- ID がストリング値または NULL 値を持つ場合の ID [NOT] IN (string-literal1, string-literal2,...) 比較演算子。
  - `Country IN ('UK', 'US', 'France')` は `'UK'` の場合には true であり、`'Peru'` の場合には false です。 これは、式 `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')` と同等です。
  - `Country NOT IN ('UK', 'US', 'France')` は `'UK'` の場合には false であり、`'Peru'` の場合には true です。 これは、式 `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))` と同等です。
  - IN または NOT IN 演算の ID がヌルである場合、演算の値は不明です。
- identifier [NOT] LIKE *pattern-value* [ESCAPE *escape-character*] 比較演算子。 identifier にはストリング値があります。 *pattern-value* はストリング・リテラルです。ここで、_ は単一文字を表しており、% は文字シーケンス (空のシーケンスを含む) を表しています。 その他のすべての文字はそれ自体を表しています。 オプションの *escape-character* は単一の文字ストリング・リテラルです。この文字は、*pattern-value* 内の _ および % の特殊な意味をエスケープするために使用されます。 LIKE 演算子は、2 つのストリング値の比較のみに使用する必要があります。
  - `phone LIKE '12%3'` は、123 および 12993 の場合には true で、1234 の場合には false です。
  - `word LIKE 'l_se'` は、lose の場合には true で、loose の場合には false です。
  - `underscored LIKE '\_%' ESCAPE '\'` は _foo の場合には true であり、bar の場合には false です。
  - `phone NOT LIKE '12%3'` は、123 および 12993 の場合には false で、1234 の場合には true です。
  - LIKE 操作または NOT LIKE 操作の ID が NULL である場合、操作の値は不明です。

  注: LIKE 演算子は、2 つのストリング値の比較のために使用する必要があります。 Root.MQMD.CorrelId の値は、文字ストリングではなく、24 バイトのバイト配列です。 セレクター・ストリング `Root.MQMD.CorrelId LIKE 'ABC%'` は、構文的には有効であるとしてパーサーによって受け入れられますが、false と評価されます。 このため、バイト配列を文字ストリングと比較するときには、LIKE を使用することはできません。
- identifier IS NULL 比較演算子は、NULL ヘッダー・フィールド値、または欠落しているプロパティー値をテストします。
- identifier IS NOT NULL 比較演算子は、ヌル以外のヘッダー・フィールド値またはプロパティー値の存在をテストします。
- ヌル値

  NULL 値を含むセレクター式の評価は、以下のように SQL 92 NULL セマンティクスによって定義されます。
  - SQL は、 NULL 値を不明として扱います。

– 不明値を持つ比較または算術は、常に、不明値を生じさせます。
– IS NULL および IS NOT NULL 演算子は、不明値を TRUE および FALSE 値に変換します。

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

表 *1.* ロジックが *A AND B* の場合のブール演算子の結果

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

表 *2.* ロジックが *A OR B* の場合のブール演算子の結果

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

表 *3.* ロジックが *NOT A* の場合のブール演算子の結果

演算子 A	**結果 (NOT A)**
T	F
F	T
U	U

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

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

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

SQL コメントは、サポートされていません。

関連概念

選択動作
IBM WebSphere MQ 選択動作の概要。

メッセージの内容の選択
選択したメッセージ・ペイロードの内容 (コンテンツ・フィルタリング) に基づいてサブスクライブすることができますが、そのようなサブスクリプションにどのメッセージを送信するかを、WebSphere MQ で直接判断することはできません。 このようなメッセージを処理するには、代わりに、IBM Integration Bus などの拡張メッセージ選択プロバイダーが必要です。

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

関連資料

MsgHandle
MQBUFMH - バッファーからメッセージ・ハンドルへの変換

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

- 等価は単一の等号文字を使用してテストされます。例えば、 `a = b` は正しいですが、 `a == b` は正しくありません。
- 多くのプログラミング言語で使用される、「等しくない」ことを表す演算子は `!=` です。 この表記は `<>`の有効な同義語ではありません。例えば、 `a <> b` は有効ですが、 `a != b` は無効です。
- 単一引用符が認識されるのは、' (U+0027) 文字が使用されている場合のみです。 同様に、二重引用符はバイト・ストリングを囲むために使用される場合にのみ有効であり、" (U+0022) 文字を使用する必要があります。
- 記号 &、 &&、 | 、および || は、論理積/論理和の同義語ではありません。例えば、 `a && b` は `a AND b` として指定する必要があります。
- ワイルドカード文字 `*` および `?` は、`%` および `_` の同義語ではありません。
- `20 < b < 30` などの複合式を含むセレクターは無効です。 演算子の優先順位が同じ場合、パーサーは左から右の順に評価します。 したがって、この例は `(20 < b) < 30` になりますが、これには意味がありません。 代わりに、式を `(b > 20) AND (b < 30)` と記述する必要があります。
- バイト・ストリングは二重引用符で囲む必要があります。単一引用符が使用された場合、バイト・ストリングはストリング・リテラルであると見なされます。 `0x` に続く文字の数 (文字が表す数ではない) は、2 の倍数でなければなりません。
- キーワード `IS` は等号の同義語ではありません。 したがって、選択ストリング `a IS 3` および `b IS 'red'` は無効です。 `IS` キーワードは、 `IS NULL` および `IS NOT NULL` のケースをサポートするためにのみ存在します。

関連概念

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

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

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

メッセージ・プロパティー名は、任意の有効な Unicode 文字のシーケンスを 使用して指定できます。 UTF-8 でエンコードされた選択ストリング内に含まれるメッセージ・プロパティー名 は、マルチバイト文

字を含んでいる場合でも妥当性検査されます。 マルチバイト UTF-8 の妥当性検査は厳密であり、有効な UTF-8 シーケンス がメッセージ・プロパティー名に使用されるようにする必要があります。

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

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

関連概念

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

メッセージの内容の選択
選択したメッセージ・ペイロードの内容 (コンテンツ・フィルタリング) に基づいてサブスクライブすることができますが、そのようなサブスクリプションにどのメッセージを送信するかを、WebSphere MQ で直接判断することはできません。 このようなメッセージを処理するには、代わりに、IBM Integration Bus などの拡張メッセージ選択プロバイダーが必要です。

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

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

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

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

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

関連概念

選択動作
IBM WebSphere MQ 選択動作の概要。

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

IBM WebSphere MQ メッセージの非同期コンシューム

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

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

メッセージの非同期コンシュームでは、以下の用語が使用されます。

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

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

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

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

以下の図は、この機能の使用例を示しています。

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

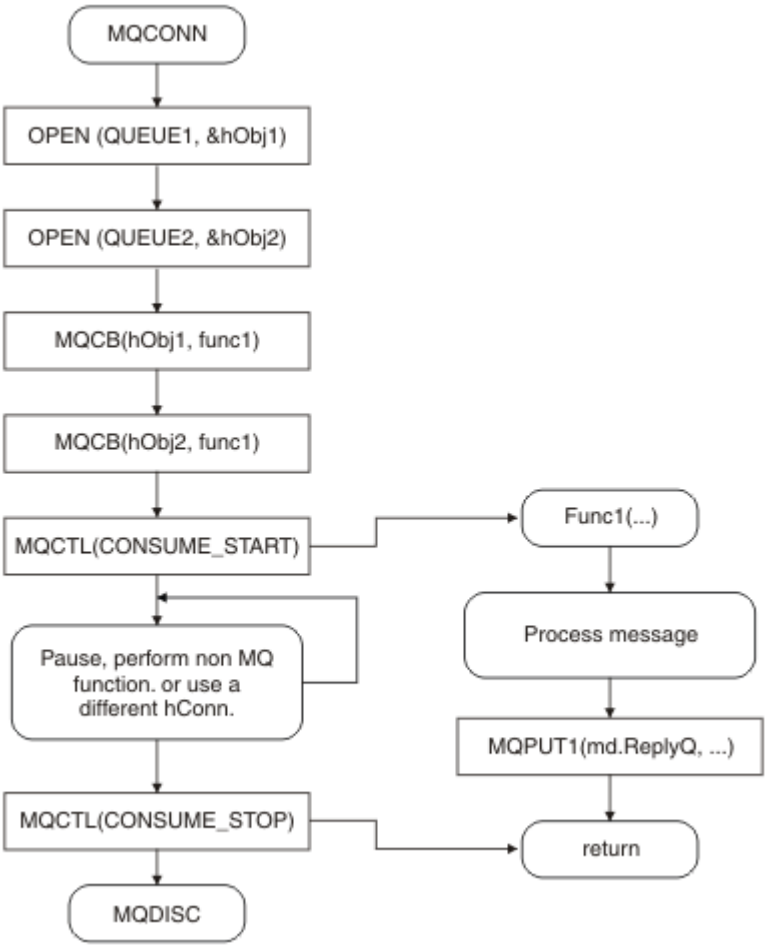


図 *5. 2* つのキューから消費する標準的なメッセージ・ドリブン・アプリケーション

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

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

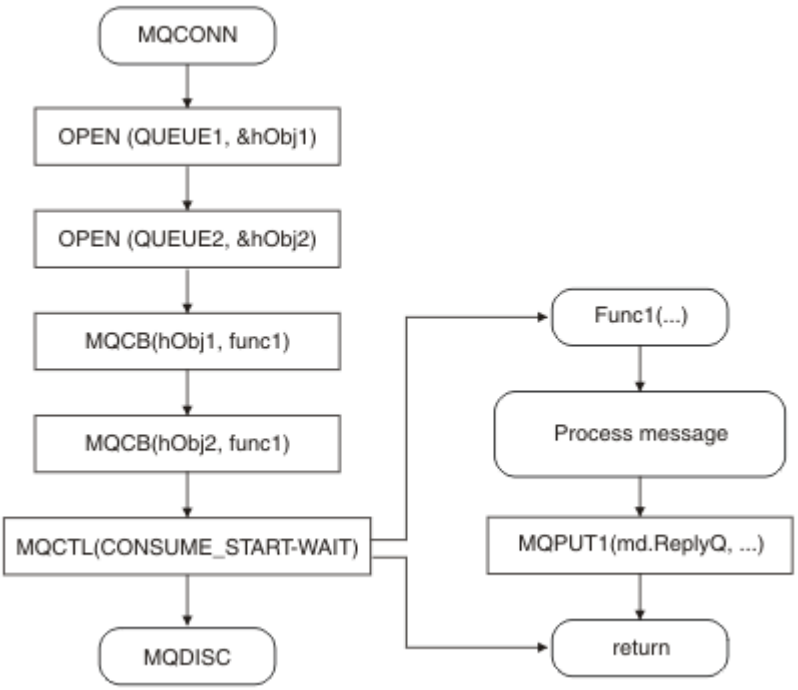


図 *6. 2* つのキューから消費する単一スレッドのメッセージ・ドリブン・アプリケーション

メッセージ・グループ

メッセージは、メッセージを順序付けするためにグループ化できます。

メッセージ・グループを使用すると、複数のメッセージを相互に関連したものとしてマークすることができます。また、グループに論理順序を適用することができます (245 ページの『論理的な順序付けと物理的な順序付け』 を参照)。 z/OS 以外のプラットフォームでは、関連する概念である 262 ページの『メッセージのセグメント化』 を使用すると、大きなメッセージを小さなセグメントに分割することができます。 グループ化またはセグメント化したメッセージは、トピックに書き込む際に使用することはできません。

グループ内の階層は、次のようになります。

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

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

次の 35 ページの図 7 は論理メッセージのグループを示しています。

図 *7.* 論理メッセージのグループ

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

論理メッセージ

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

グループ内の論理メッセージを次の目的で使用します。

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

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

セグメント

セグメントは、書き込み用または読み取り用のアプリケーション、あるいはキュー・マネージャー (メッセージが処理されるときに介入するキュー・マネージャーなど) にとって大きすぎるメッセージを処理するために使用されます。 詳しくは、262 ページの『メッセージのセグメント化』 を参照してください。

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

各セグメントは、1 つの物理メッセージから成り、このメッセージがグループに属している場合もあります (36 ページの図 8 にグループ内のメッセージの例を示します)。 セグメントは、論理的には 1 つのメッセージの一部分で、同じメッセージの個々のセグメントについては、MQMD 内の *MsgId*、*Offset*、*SegmentFlag* の各フィールドが異なるだけです。 セグメントが到着に失敗した場合は、理由コード MQRC_INCOMPLETE_GROUP または MQRC_INCOMPLETE_MSG が必要に応じて戻されます。

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

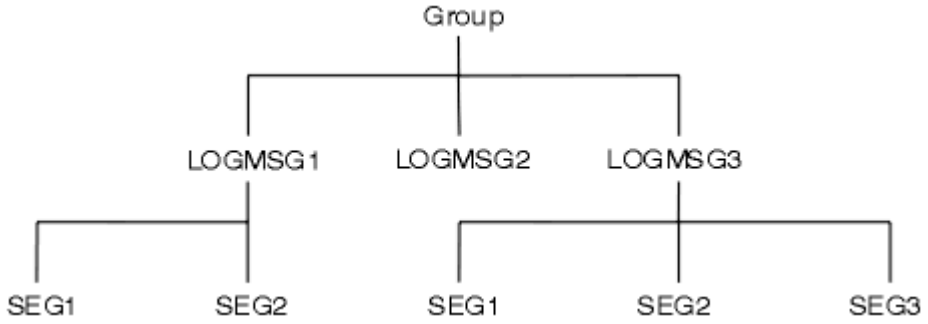


図 *8.* セグメント化されたメッセージ

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

論理メッセージと物理メッセージについては、245 ページの『論理的な順序付けと物理的な順序付け』 を参照してください。 メッセージのセグメント化の詳細については、262 ページの『メッセージのセグメント化』 を参照してください。

メッセージの持続性

持続メッセージはログとキュー・データ・ファイルに書き込まれます。

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

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

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

- 現在の作業単位外の持続メッセージは、書き込みまたは読み取り操作ごとにディスクに書き込まれます。324 ページの『作業単位のコミットとバックアウト』を参照してください。
- UNIX システム上の IBM WebSphere MQ 、 Linux システム上の IBM WebSphere MQ 、 および IBM WebSphere MQ for Windows では、現行の作業単位内の持続メッセージは、作業単位がコミットされたときにのみログに記録されます (作業単位に多数のキュー操作が含まれる可能性があります)。

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

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

送達できないメッセージ

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

以下のように行えます。

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

詳細については、553 ページの『プログラム・エラーの処理』を参照してください。

バックアウトされるメッセージ

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

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

WebSphere MQ for WebSphere MQ for Windows、 WebSphere MQ on UNIX システム、 WebSphere MQ on Linux システム バックアウト・カウントは、常にキュー・マネージャーの再始動後も存続します。 *HardenGetBackout* 属性に対する変更は、すべて無視されます。

メッセージのコミットおよびバックアウトの詳細については、324 ページの『作業単位のコミットとバックアウト』を参照してください。

応答先キューおよびキュー・マネージャー

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

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

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

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

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

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

注 : キュー・マネージャーにメッセージの送達を複数回試行させるようにし、送達できなかった場合にそのメッセージを破棄するよう要求することができます。 送達できなかったメッセージを破棄できなかった場合、リモート・キュー・マネージャーはそのメッセージを送達不能 (未配布メッセージ) キューに入れます (556 ページの『送達不能 (未配布メッセージ) キューの使用』を参照)。

メッセージ・コンテキスト

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

取り出しを行うアプリケーションは次のような処理を行うことがあります。

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

メッセージをキューに書き込むために MQPUT または MQPUT1 呼び出しを使用するときは、キュー・マネージャーがメッセージ記述子に何らかのデフォルト・コンテキスト情報を追加するように指定できます。適切な許可レベルを持つアプリケーションは、余分のコンテキスト情報を追加できます。 コンテキスト情報の指定方法の詳細については、231 ページの『コンテキスト情報の制御』を参照してください。

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

- 送達時の確認
- Expiry

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

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

identity コンテキスト

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

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

WebSphere MQ for Windows でメッセージが作成されると、 Windows システム・セキュリティー ID (SID) が *AccountingToken* フィールドに保管されます。 SID の使用目的は、*UserIdentifier* フィールドの補足とユーザーの認証の確立です。

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

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

origin コンテキスト

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

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

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

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

WebSphere MQ オブジェクト

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

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

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

関連概念

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

関連資料

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

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

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

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

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

- コンポーネントが MQI に C 言語バインディングを使用する場合は、461 ページの『Windows での C プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー mqic.lib ではなく、ライブラリー mqicxa.lib でリンクします。
- コンポーネントが WebSphere MQ C++ クラスを使用する場合は、658 ページの『Windows における C++ プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー imqc23vn.lib ではなく、ライブラリー imqx23vn.lib でリンクします。
- コンポーネントが MQI に Visual Basic 言語バインディングを使用する場合は、466 ページの『Windows での Visual Basic プログラムの作成』 の指示に従います。ただし、Visual Basic プロジェクトを定義する際、「**条件付きコンパイル引数**」 フィールドに `MqType=3` と入力します。
- コンポーネントが WebSphere MQ Automation Classes for ActiveX (MQAX) を使用する場合は、値 `mqic32xa.dll` を使用して環境変数 GMQ_MQ_LIB を定義します。

  この環境変数は、アプリケーション内から定義するか、有効範囲がシステム全体になるように定義することができます。 ただし、この環境変数をシステム全体として定義すると、アプリケーション内からその環境変数を定義しない既存の MQAX アプリケーションが、誤った動作をする可能性があります。

IBM WebSphere MQ と WebSphere Application Server の使用

このトピックでは、IBM WebSphere MQ を WebSphere Application Server で使用する方法について知ることができます。

WebSphere Application Server で実行されている Java で作成されたアプリケーションは、Java Messaging Service (JMS) 仕様を使用してメッセージングを実行できます。 この環境での Point-to-Point メッセージングは、IBM WebSphere MQ キュー・マネージャーによって提供される場合があります。

IBM WebSphere MQ キュー・マネージャーを使用して Point-to-Point メッセージングを提供する利点は、接続する JMS アプリケーションが IBM WebSphere MQ ネットワークの機能を十分に使用できるという点です。これにより、アプリケーションは、多数のプラットフォームで実行されているキュー・マネージャーとメッセージを交換することができます。

アプリケーションは、キュー接続ファクトリー・オブジェクト用にクライアント・トランスポート または バインディング・トランスポート を 使用できます。 バインディング・トランスポート の場合、キュー・マネージャーは接続を要求するアプリケーションのローカルに存在している必要があります。 キュー・マネージャーがアプリケーションに対してローカルでない場合、アプリケーションが別のマシンまたはイメージで実行されているキュー・マネージャーに接続できるようにするには、クライアント接続機構 をインストールする必要があります。

デフォルトでは、IBM WebSphere MQ キュー上に保持された JMS メッセージは、MQRFH2 ヘッダーを 使用して JMS メッセージ・ヘッダー情報の一部を保持します。 多数のレガシー IBM WebSphere MQ アプリケーションは、これらのヘッダー付きメッセージを処理できず、独自の特性ヘッダー (例えば、CICS ブリッジ用 MQCIH、または IBM WebSphere MQ Workflow アプリケーション用 MQWIH) が必要です。 これらの特別な考慮事項について詳しくは、815 ページの『JMS メッセージの WebSphere MQ メッセージへのマッピング』を参照してください。

トランザクション・サポートのシナリオ

トランザクション・サポートを使用すると、信頼性の高い方法でアプリケーションからデータベースを処理できるようになります。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

このセクションでは、トランザクション・サポートについて説明します。 データベース製品を含む IBM WebSphere MQ をアプリケーションで利用可能にするために必要な作業には、アプリケーション・プログラミングおよびシステム管理の分野が含まれます。 この情報および 324 ページの『作業単位のコミットとバックアウト』を使用してください。

まずトランザクションを形成する作業単位の概要を示し、次に IBM WebSphere MQ がトランザクションをデータベースで調整できるようにする方法を説明します。

関連概念

41 ページの『作業単位の紹介』
このトピックでは、作業単位、コミット、バックアウト、および同期点についての一般的な概念を紹介および定義します。 また、グローバル作業単位について説明する 2 つのシナリオも記載しています。

IBM WebSphere MQ と HP NonStop TMF

作業単位の紹介

このトピックでは、作業単位、コミット、バックアウト、および同期点についての一般的な概念を紹介および定義します。 また、グローバル作業単位について説明する 2 つのシナリオも記載しています。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

プログラムが作業単位内のキューにメッセージを書き込むと、それらのメッセージは、そのプログラムが作業単位をコミット したときにのみ、他のプログラムから見えるようになります。 作業単位をコミットするには、データの安全性を保護するためにすべての更新処理が正常終了する必要があります。

プログラムがエラーを検出し、PUT 操作を永続的にしないと決めた場合、プログラムは、その作業単位をバックアウト できます。 プログラムがバックアウトを行うと、WebSphere MQ はその作業単位によってキューに書き込まれたメッセージを除去することにより、キューを復元します。

同様に、プログラムが作業単位内の 1 つ以上のキューからメッセージを読み取ると、それらのメッセージは、プログラムがその作業単位をコミットするまでキューに留まりますが、それらのメッセージを他のプログラムで取り出すことはできません。 それらのメッセージは、プログラムが作業単位をコミットしたときに、キューから永続的に削除されます。 プログラムが作業単位をバックアウトすると、WebSphere MQ は、メッセージが他のプログラムによって検索できるようにすることによって、キューを復元します。

変更をコミットまたはバックアウトする決定は、最も単純な場合、タスクの終了時に行われます。 しかし、アプリケーションにとってはタスク内の他の論理点でデータ変更の同期をとる方が好都合な場合があります。 論理点は同期点と呼ばれ、2 つの同期点間での一連の更新を処理する期間を作業単位 といいます。 1 つの作業単位に、複数の MQGET 呼び出しや MQPUT 呼び出しを含めることができます。

WebSphere MQ では、ローカル 作業単位とグローバル 作業単位を区別する必要があります。

ローカル作業単位
WebSphere MQ キューへの書き込みおよびキューからの読み取りのみを行う作業単位。各作業単位の調整は、単一フェーズ・コミット・プロセスを使用してキュー・マネージャー内で行われます。

更新されるリソースが、単一の WebSphere MQ キュー・マネージャーによって管理されるキューのみである場合は、ローカル作業単位を使用します。 更新のコミットには MQCMIT verb を、更新のバックアウトには MQBACK をそれぞれ使用します。

ローカル作業単位の使用に関して、ログ管理以外にシステム管理タスクはありません。 ご使用のアプリケーションで、MQCMIT および MQBACK と共に MQPUT 呼び出しと MQGET 呼び出しを使用する場合は、MQPMO_SYNCPOINT オプションと MQGMO_SYNCPOINT オプションを使用してみてください。 (ログ管理については、ログ・ファイルを管理するを参照してください。)

グローバル作業単位
リレーショナル・データベース内のテーブルなど、他のリソースの更新も行われる作業単位。 複数のリソース・マネージャー が関与するときは、2 フェーズ・コミット・プロセスを使用してグローバル作業単位を調整するトランザクション・マネージャー・ソフトウェアが必要です。

Db2、Oracle、Sybase、および Informix などのリレーショナル・データベース・マネージャー・ソフトウェアの更新も必要な場合に、グローバル作業単位を使用します。

グローバル作業単位を使用するシナリオはいくつかあります。 ここでは 2 つのシナリオについて説明します。

1. 最初のシナリオでは、キュー・マネージャー自体がトランザクション・マネージャーとして機能します。 このシナリオでは、MQI verb がグローバル作業単位を制御します。グローバル作業単位は、MQBEGIN verb を使用してアプリケーションで開始され、次いで MQCMIT を使用してコミットされるか、MQBACK を使用してバックアウトされます。
2. 2 つ目のシナリオでは、トランザクション・マネージャー役割は、TXSeries、Encina、または Tuxedo など、他のソフトウェアによって実行されます。 このシナリオでは、トランザクション・マネージャー・ソフトウェアによって提供される API を使用して、作業単位を制御します (例えば、TXSeries の場合は EXEC CICS SYNCPOINT)。

以上の 2 つのシナリオで編成された、グローバル作業単位を使用するために必要なすべての手順について、以下のセクションで説明します。

- 42 ページの『シナリオ 1: キュー・マネージャーが調整を行う』
- 69 ページの『シナリオ 2: 他のソフトウェアが調整を行う』

シナリオ 1: キュー・マネージャーが調整を行う

シナリオ 1 では、キュー・マネージャーがトランザクション・マネージャーとして機能します。 このシナリオでは、MQI verb がグローバル作業単位を制御します。グローバル作業単位は、MQBEGIN verb を使用してアプリケーションで開始され、次いで MQCMIT を使用してコミットされるか、MQBACK を使用してバックアウトされます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

分離レベル

IBM WebSphere MQ では、データベース内に実装されたトランザクション分離設計に応じて、データベースが更新される前にキュー上のメッセージが表示されることがあります。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

IBM WebSphere MQ キュー・マネージャーが XA トランザクション・マネージャーとして稼働している場合、XA リソース・マネージャーに対する更新を調整するには、以下のコミット・プロトコルに従います。

1. すべての XA リソース・マネージャーを準備します。
2. IBM WebSphere MQ キュー・マネージャー・リソース・マネージャーをコミットします。
3. その他のリソース・マネージャーをコミットします。

ステップ 2 と 3 の間に、キューにコミットされているメッセージをアプリケーションが認識することがありますが、データベース内の対応する行はこのメッセージを反映していません。

アプリケーションのデータベース API 呼び出しが保留中の更新の完了を待つようにデータベースが構成されている場合、このことは問題にはなりません。

データベースの構成を変えると、この問題を解決できます。 必要な構成のタイプは、「分離レベル」と呼ばれます。 分離レベルについて詳しくは、データベースの資料を参照してください。 あるいは、以下の逆順でリソース・マネージャーをコミットするようにキュー・マネージャーを構成することもできます。

1. すべての XA リソース・マネージャーを準備します。
2. その他のリソース・マネージャーをコミットします。
3. IBM WebSphere MQ キュー・マネージャー・リソース・マネージャーをコミットします。

プロトコルを変更すると、IBM WebSphere MQ キュー・マネージャーは最後にコミットされるので、キューからメッセージを読み取るアプリケーションは、対応するデータベースの更新の完了後のみメッセージを認識します。

この変更されたプロトコルを使用するようにキュー・マネージャーを構成するには、**AMQ_REVERSE_COMMIT_ORDER** 環境変数を設定します。

この環境変数は、**strmqm** を実行してキュー・マネージャーを開始する環境で設定します。 例えば、キュー・マネージャーを開始する直前にシェルで以下のコマンドを実行します。

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

注 : この環境変数を設定すると、トランザクションごとに追加のログ・エントリーが発生する可能性があるので、各トランザクションのパフォーマンスに少々影響があります。

データベースの調整

キュー・マネージャーでグローバル作業単位自体を調整している場合は、データベースの更新を作業単位に統合できます。 つまり MQI と SQL の混合アプリケーションを作成し、MQCMIT verb および MQBACK verb を使用して、キューとデータベースへの変更をまとめてコミットまたはロールバックすることができます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

キュー・マネージャーは、「*X/Open Distributed Transaction Processing: The XA Specification*」に記載されている 2 フェーズ・コミット・プロトコルを使用してこれを行います。 作業単位をコミットする場合、キュー・マネージャーはまず作業単位にかかわっている各データベース・マネージャーに対して、更新する準備ができているかどうかを確認します。 キュー・マネージャーをはじめ、作業単位にかかわっているすべてのプログラムおよびデータベースがコミットできる状態にある場合にのみ、キューおよびデータベースの更新がすべてコミットされます。 更新できない状態にあるプログラムが 1 つでもあると、作業単位はバックアウトされます。

通常、グローバル作業単位は、以下の方式 (疑似コード) でアプリケーションにインプリメントされます。

MQBEGIN
MQGET (メッセージ・オプションにフラグ MQGMO_SYNCPOINT を組み込む)
MQPUT (メッセージ・オプションにフラグ MQPMO_SYNCPOINT を組み込む)
SQL INSERT
MQCMIT

MQBEGIN は、グローバル作業単位の先頭を指示するためのものです。 MQCMIT は、2 フェーズ・コミット・プロトコルを使用して、グローバル作業単位の終わりを指示し、関与するすべてのリソース・マネージャーについてこの終わりを完了するためのものです。

作業単位 (トランザクション ともいいます) が MQCMIT を使用して正常に完了されると、その作業単位で取られたすべてのアクションが永続的または不可逆的にされます。 何らかの理由で、作業単位が失敗した場合は、すべてのアクションがバックアウトされます。 1 つの作業単位内の 1 つのアクションを永続的にし、別のアクションをバックアウトすることはできません。 これが、作業単位の原理です。つまり、作業単位内のすべてのアクションを永続的にするか、すべてを非永続的にするかになります。

注 :

1. アプリケーション・プログラマーは、MQBACK を呼び出すことによって作業単位を強制的にバックアウトできます。 MQCMIT が呼び出される前にアプリケーションまたはデータベースが失敗 した場合は、キュー・マネージャーで作業単位をバックアウトすることもできます。
2. アプリケーションが MQCMIT を呼び出さずに MQDISC を呼び出した場合、キュー・マネージャーは、MQCMIT が呼び出されている場合と同様に振る舞い、作業単位をコミットします。

MQBEGIN から MQCMIT までの間では、キュー・マネージャーは、そのリソースを更新するためにデータベースへの呼び出しを行いません。 すなわち、データベースのテーブルを変更するには、コードを作成 (例えば、疑似コードでの SQL INSERT) する方法しかありません。

コミット・プロトコル中にキュー・マネージャーとデータベース・マネージャーとの間の連絡が断たれた場合には、完全リカバリー・サポートを使用できます。 データベース・マネージャーが未確定のうちに使用できなくなった場合、つまりコミットできる状態になっているがコミットまたはバックアウトの決定をまだ受信していない場合、キュー・マネージャーはその結果がデータベースに正常に送信できるまで作業単位の実行結果を保管しておきます。 同様に、コミット操作が完了していないままの状態でキュー・マネージャーが終了した場合には、キュー・マネージャーの再開時にはこの未完了の状態が保管されています。アプリケーションが突然終了した場合、作業単位の保全性に影響はありませんが、45 ページの表 5 に記載されているとおり、結果は、アプリケーションが終了したプロセスの場所によって異なります。

データベースまたはアプリケーション・プログラムが失敗した時点で起こることを、以下の表に要約します。

表 4. データベース・サーバーが失敗したときに起こること

失敗発生時	**結果**
アプリケーションが MQCMIT を呼び出す前。	作業単位がバックアウトされます。
アプリケーションによる MQCMIT の呼び出し中で、すべてのデータベースが正しく準備されていることを示す**前**。	理由コード MQRC_BACKED_OUT が返され、作業単位はバックアウトされます。
アプリケーションによる MQCMIT の呼び出し中で、すべてのデータベースが正しく準備されていることを示した**後**、ただし、正常にコミットしたことを示す前。	理由コード MQRC_OUTCOME_PENDING が返され、作業単位は、キュー・マネージャーによりリカバリー可能な状態に保持されます。
アプリケーションによる MQCMIT の呼び出し中で、すべてのデータベースが正しくコミットしたことを示した**後**。	理由コード MQRC_NONE が返され、作業単位はコミットされます。
アプリケーションが MQCMIT を呼び出した後。	理由コード MQRC_NONE が返され、作業単位はコミットされます。

表 5. アプリケーション・プログラムが失敗したときに起こること	
失敗発生時	**結果**
アプリケーションが MQCMIT を呼び出す前。	作業単位がバックアウトされます。
アプリケーションによる MQCMIT の呼び出し中で、キュー・マネージャーがアプリケーションの MQCMIT 要求を受信する**前**。	作業単位がバックアウトされます。
アプリケーションによる MQCMIT の呼び出し中で、キュー・マネージャーがアプリケーションの MQCMIT 要求を受信した**後**。	キュー・マネージャーは、2 フェーズ・コミットを使用してコミットを試みます (作業単位のそれぞれの部分を正しく実行およびコミットするデータベース製品の影響を受けます)。

MQCMIT から戻される途中の理由コードが MQRC_OUTCOME_PENDING の場合、作業単位は、データベース・サーバーとの連絡を再確立し、データベース・サーバーに作業単位のその部分をコミットするよう指示できるようになるまでキュー・マネージャーによって保管されます。 リカバリーの方法とその時期については、61 ページの『XA リソース・マネージャーとの連絡が失われる際の考慮事項』を参照してください。

キュー・マネージャーは、「*X/Open Distributed Transaction Processing: The XA Specification* 」に記載されている XA インターフェースを使用してデータベース・マネージャーと通信します。 これらの関数呼び出しの例として、xa_open、xa_start、 xa_end、xa_prepare、xa_commit があります。 トランザクション・マネージャー とリソース・マネージャー という用語は、XA 仕様で使用されているのと同じ意味で使用されています。

制限対象機能
データベース調整サポートに対する制約事項があります。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

次の制約事項が適用されます。

- WebSphere MQ の作業単位内でのデータベース更新の調整は、MQI クライアント・アプリケーションでは**サポートされていません**。 クライアント・アプリケーションでの MQBEGIN の使用は失敗します。 MQBEGIN を呼び出すプログラムは、キュー・マネージャーと同じマシン上でサーバー・アプリケーションとして実行する必要があります。

  注 : サーバー・アプリケーションとは、必要な WebSphere MQ サーバー・ライブラリーとリンクされているプログラムです。また、クライアント・アプリケーションとは、必要な WebSphere MQ クライアント・ライブラリーとリンクされているプログラムです。 プログラムのコンパイルおよびリンクについて詳しくは、359 ページの『WebSphere MQ MQI クライアント用のアプリケーションの作成』および 429 ページの『IBM WebSphere MQ アプリケーションの構築』を参照してください。
- データベース・クライアントがキュー・マネージャーと同じマシン上にインストールされており、それがこの機能をサポートしている限り、データベース・サーバーをキュー・マネージャー・サーバーとは別のマシンに置くことは可能です。 データベース製品のクライアント・ソフトウェアを 2 フェーズ・コミット・システムに使用できるかどうかを判別するには、データベース製品の資料を参照してください。
- キュー・マネージャーは (シナリオ 2 のグローバル作業単位に関与する目的で) リソース・マネージャーとして動作しますが、あるキュー・マネージャーにそのシナリオ 1 のグローバル作業単位内で別のキュー・マネージャーを調整させることはできません。

スイッチ・ロード・ファイル

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

スイッチ・ロード・ファイルは、 IBM WebSphere MQ アプリケーションおよびキュー・マネージャーのコードによってロードされる共有ライブラリー (Windows システム上の DLL) です。 これは、データベースのクライアント共用ライブラリーのロードを単純化し、XA 関数にポインターを返すためのものです。

スイッチ・ロード・ファイルの詳細は、キュー・マネージャーを始動する前に指定する必要があります。詳細は、Windows および UNIX and Linux システムの qm.ini ファイルに記載されています。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、IBM WebSphere MQ Explorer を使用して qm.ini ファイルを更新します。
- その他のすべてのシステムでは、qm.ini ファイルを直接編集します。

スイッチ・ロード・ファイルの C ソースは、シナリオ 1 のグローバル作業単位をサポートする場合、IBM WebSphere MQ のインストールにより提供されます。 ソースには MQStart という関数が含まれています。 スイッチ・ロード・ファイルがロードされると、キュー・マネージャーによってこの関数が呼び出され、この関数から *XA* スイッチ と呼ばれる構造体のアドレスが戻されます。

XA スイッチ構造体は、データベース・クライアント共用ライブラリーに入っており、46 ページの表 6 に記載されているように、多数の関数ポインターが含まれています。

表 *6. XA* スイッチ関数ポインター

関数ポインター名	**XA 関数**	**目的**
xa_open_entry	xa_open	データベースへの接続
xa_close_entry	xa_close	データベースからの切断
xa_start_entry	xa_start	グローバル作業単位のブランチの開始
xa_end_entry	xa_end	グローバル作業単位のブランチの中断
xa_rollback_entry	xa_rollback	グローバル作業単位のブランチのロールバック
xa_prepare_entry	xa_prepare	グローバル作業単位のブランチをコミットするための準備
xa_commit_entry	xa_commit	グローバル作業単位のブランチのコミット
xa_recover_entry	xa_recover	データベースに未確定の作業単位があるかどうかの判定
xa_forget_entry	xa_forget	データベースがグローバル作業単位のブランチの記録を消去できるようにする
xa_complete_entry	xa_complete	グローバル作業単位のブランチの完了

アプリケーションでの最初の MQBEGIN 呼び出し時に、MQBEGIN の一部として実行される IBM WebSphere MQ コードによりスイッチ・ロード・ファイルがロードされ、データベース共用ライブラリー内の xa_open 関数が呼び出されます。 同様に、キュー・マネージャーの始動時や、後続のその他の場面で、一部のキュー・マネージャー・プロセスによりスイッチ・ロード・ファイルがロードされ、xa_open が呼び出されます。

動的登録 を使用すると、xa_* 呼び出しの回数を低減できます。 この最適化手法の詳しい説明については、66 ページの『XA 動的登録』を参照してください。

データベース調整のためのシステムの構成

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

キュー・マネージャーが調整するグローバル作業単位でデータベース・マネージャーを使用するには、事前に次の作業を行ってください。 これらの作業について、以下で説明します。

- 47 ページの『データベース・プロダクトのインストールおよび構成』
- 47 ページの『スイッチ・ロード・ファイルの作成』
- 48 ページの『キュー・マネージャーへの構成情報の追加』
- 50 ページの『アプリケーションの作成および変更』
- 50 ページの『システムのテスト』

データベース・プロダクトのインストールおよび構成
データベース・プロダクトをインストールして構成するには、プロダクト独自の資料を参照してください。 このセクションの以下のトピックでは、構成に関する一般的な問題と、それらが WebSphere MQ とデータベースの間の相互協調処理にどのように関連しているかを説明します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

データベースの接続

キュー・マネージャーへの標準接続を確立するアプリケーションは、個別のローカル・キュー・マネージャー・エージェント・プロセスのスレッドと関連付けられます。 (ファースト・パス 接続でない接続が、このコンテキストの標準 接続です。 詳細については、208 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』を参照してください。

アプリケーションによって MQBEGIN が出されたときは、このアプリケーションとエージェント・プロセスの両方がデータベース・クライアント・ライブラリー内の xa_open 関数を呼び出します。 この呼び出しに対する応答で、データベース・クライアント・ライブラリー・コードは、アプリケーションとキュー・マネージャー・プロセスの両方から 作業単位に必要となるデータベースに接続 します。 アプリケーションがキュー・マネージャーに接続している限り、これらのデータベースへの接続は維持されます。

このことは、データベースによってサポートされるユーザーまたは接続の数が限られている場合には重要な考慮事項となります。1 つのアプリケーション・プログラムをサポートするのに、データベースに対して 2 つの接続が確立されるためです。

クライアント/サーバー構成

WebSphere MQ キュー・マネージャーおよびアプリケーション・プロセスにロードされるデータベース・クライアント・ライブラリーは、そのサーバーとの間で送受信が**可能でなければなりません**。 以下の事項について確認してください。

- データベースのクライアント/サーバー構成ファイルに正しい詳細がある
- 関係のある環境変数がキュー・マネージャー**および**アプリケーション・プロセスの環境で設定されている

スイッチ・ロード・ファイルの作成
WebSphere MQ には、サポートされているデータベース・マネージャー用のスイッチ・ロード・ファイルの作成に使用する、サンプルの Make ファイルが付属しています。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

サンプルの Make ファイルは、スイッチ・ロード・ファイルの作成に必要なすべての関連 C ソース・ファイルと共に、次のディレクトリーにインストールされます。

- WebSphere MQ for Windows の場合、*MQ_INSTALLATION_PATH*\tools\c\samples\xatm\ ディレクトリー内
- WebSphere MQ for UNIX and Linux システムの場合、*MQ_INSTALLATION_PATH*/samp/xatm/ ディレクトリー内

スイッチ・ロード・ファイルの作成に使用するサンプル・ソース・モジュールは次のとおりです。

- db2swit.c (DB2 用)
- oraswit.c (Oracle 用)
- infswit.c (Informix 用)
- sybswit.c (Sybase 用)

スイッチ・ロード・ファイルを生成する際に、32 ビット・スイッチ・ロード・ファイルを /var/mqm/exits にインストールし、64 ビット・スイッチ・ロード・ファイルを /var/mqm/exits64 にインストールします。

32 ビット・キュー・マネージャーを使用している場合は、サンプルの Make ファイル xaswit.mak により、32 ビット・スイッチ・ロード・ファイルが /var/mqm/exits にインストールされます。

64 ビット・キュー・マネージャーを使用している場合は、サンプルの Make ファイル xaswit.mak により、32 ビット・スイッチ・ロード・ファイルが /var/mqm/exits に、64 ビット・スイッチ・ロード・ファイルが /var/mqm/exits64 にインストールされます。

ファイル・セキュリティー

ご使用のオペレーティング・システムで、WebSphere MQ の制御が及ばない原因により、WebSphere MQ によるスイッチ・ロード・ファイルのロードが失敗することがあります。 この場合、エラー・メッセージが WebSphere MQ エラー・ログに書き込まれます。また、MQBEGIN 呼び出しが失敗する可能性があります。 ご使用のオペレーティング・システムでスイッチ・ロード・ファイルのロードが失敗しないようにするには、次の要件を満たす必要があります。

1. スイッチ・ロード・ファイルは、qm.ini ファイルで指定された場所で使用可能である必要があります。
2. スイッチ・ロード・ファイルは、キュー・マネージャー・プロセスおよびアプリケーション・プロセスを含む、このファイルをロードする必要のあるすべてのプロセスによってアクセス可能である必要があります。
3. データベース製品から提供されるライブラリーを含む、スイッチ・ロード・ファイルが依存するすべてのライブラリーは、存在していてアクセス可能である必要があります。

キュー・マネージャーへの構成情報の追加

データベース・マネージャー用のスイッチ・ロード・ファイルを作成し、それを安全な場所に保管したら、その場所をキュー・マネージャーに指定する必要があります。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

場所を指定するには、以下のステップを実行します。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、 WebSphere MQ エクスプローラーを使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。
- 他のすべてのシステムにおいて、キュー・マネージャーの qm.ini ファイルにある、XAResourceManager スタンザのスイッチ・ロード・ファイルの詳細を指定します。

キュー・マネージャーで調整する予定のデータベースについて XAResourceManager スタンザを追加してください。 通常、データベースは 1 つのみなので、XAResourceManager スタンザも 1 つです。 複数のデータベースを含む複雑な構成についての詳細は、60 ページの『複数のデータベースの構成』を参照してください。 XAResourceManager スタンザの属性は次のとおりです。

Name=name
リソース・マネージャーを識別する、ユーザーが選択したストリング。 結果的に、それにより、XAResourceManager スタンザに名前が付けられます。 キュー・マネージャーの名前は必須指定属性で、長さは 31 文字までです。

ユーザーが選択する名前は固有のものでなければなりません。この qm.ini ファイル内にある、この名前を持つ XAResourceManager スタンザは 1 つでなければなりません。 また、この名前は、分かりやすい名前にしてください。`dspmqtrn` コマンドの実行時に、キュー・マネージャーがこの名前を使用して、キュー・マネージャー・エラー・ログ・メッセージと出力の両方でこのリソース・マネージャーを参照するためです。 (詳細については、63 ページの『dspmqtrn コマンドを使用した未解決の作業単位の表示』を参照してください。)

名前を選択し、キュー・マネージャーを始動したら、名前属性を変更しないでください。 構成の変更についての詳細は、65 ページの『構成情報の変更』を参照してください。

SwitchFile=name
これは、以前に作成した XA スイッチ・ロード・ファイルの名前です。 この属性は必須指定属性です。キュー・マネージャーおよび WebSphere MQ アプリケーション・プロセス内のコードは、次の 2 つの時点でスイッチ・ロード・ファイルのロードを試みます。

1. キュー・マネージャーの始動時
2. WebSphere MQ アプリケーション・プロセスで初めて MQBEGIN を呼び出したとき

スイッチ・ロード・ファイルのセキュリティー属性および許可属性により、これらのプロセスがこのアクションを実行できるようにする必要があります。

XAOpenString=string
これは、WebSphere MQ コードがその呼び出しに入れてデータベース・マネージャーの xa_open 関数に渡すデータ・ストリングです。 この属性はオプションです。省略すると、ゼロ長のストリングが指定されたものと見なされます。

キュー・マネージャーおよび WebSphere MQ アプリケーション・プロセス内のコードは、次の 2 つの時点で xa_open 関数を呼び出します。

1. キュー・マネージャーの始動時
2. WebSphere MQ アプリケーション・プロセスで初めて MQBEGIN を呼び出したとき

このストリングの形式は、各データベース・プロダクトに固有のものであり、そのプロダクトの資料で詳しく説明しています。 一般に、xa_open ストリングには、キュー・マネージャーとアプリケーション・プロセスの両方でデータベースに接続できるようにするために認証情報 (ユーザー名とパスワード) が含まれています。

XACloseString=string
これは、WebSphere MQ コードがその呼び出しに入れてデータベース・マネージャーの xa_close 関数に渡すデータ・ストリングです。 この属性はオプションです。省略すると、ゼロ長のストリングが指定されたものと見なされます。

キュー・マネージャーおよび WebSphere MQ アプリケーション・プロセス内のコードは、次の 2 つの時点で xa_close 関数を呼び出します。

1. キュー・マネージャーの始動時
2. 先に MQBEGIN を呼び出してから、WebSphere MQ アプリケーション・プロセスで MQDISC を呼び出すとき

このストリングの形式は、各データベース・プロダクトに固有のものであり、そのプロダクトの資料で詳しく説明しています。 一般に、このストリングは空であるので、通常、XACloseString 属性を XAResourceManager スタンザから除外します。

ThreadOfControl=THREAD|PROCESS
ThreadOfControl 値は THREAD または PROCESS です。 キュー・マネージャーはこの属性を使用してシリアライズします。 この属性はオプションです。省略すると、値 PROCESS が指定されたものと見なされます。

データベース・クライアント・コードにより、シリアライゼーションなしでスレッドが XA 関数を呼び出せる場合、ThreadOfControl の値として THREAD を使用できます。 キュー・マネージャーは、必要に応じて、同時に複数のスレッドからデータベース・クライアント共用ライブラリーの XA 関数を呼び出せると想定しています。

データベース・クライアント・コードによりスレッドがこの方法でその XA 関数を呼び出せない場合、ThreadOfControl の値は PROCESS でなければなりません。 この場合、キュー・マネージャーは、データベース・クライアント共用ライブラリーへのすべての呼び出しをシリアライズするため、特定のプロセス内から行われる呼び出しは一度に 1 つのみです。 アプリケーションが複数のスレッドを使用して実行する場合に、同様のシリアライゼーションを実行するようにしなければならないこともあります。

データベース・プロダクトがこの方法でマルチスレッド・プロセスに対処できるかどうかというこの問題は、そのプロダクトのベンダーの問題です。 ThreadOfControl 属性を THREAD または PROCESS に設定できるかどうかについては、該当するデータベース・プロダクトの資料で詳しく説明しています。 できれば、ThreadOfControl を THREAD に設定することをお勧めします。 未確定の場合、安全性のより高い オプションは、PROCESS に設定することです。ただし、THREAD を使用した場合に可能なパフォーマンスの向上は実現されません。

アプリケーションの作成および変更

グローバル作業単位のインプリメント方法を示します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

WebSphere MQ のインストールにより提供されるシナリオ 1 のグローバル作業単位用のサンプル・アプリケーション・プログラムについては、41 ページの『作業単位の紹介』で説明されています。

通常、グローバル作業単位は、以下の方式 (疑似コード) でアプリケーションにインプリメントされます。

```
MQBEGIN
MQGET
MQPUT
SQL INSERT
MQCMIT
```

MQBEGIN は、グローバル作業単位の先頭を指示するためのものです。 MQCMIT は、2 フェーズ・コミット・プロトコルを使用して、グローバル作業単位の終わりを指示し、関与するすべてのリソース・マネージャーについてこの終わりを完了するためのものです。

MQBEGIN から MQCMIT までの間では、キュー・マネージャーは、そのリソースを更新するためにデータベースへの呼び出しを行いません。 すなわち、データベースのテーブルを変更するには、コードを作成 (例えば、疑似コードでの SQL INSERT) する方法しかありません。

キュー・マネージャーの役割は、データベースに関する限り、グローバル作業単位の開始時、終了時、およびグローバル作業単位をコミットまたはロールバックするかどうかをデータベースに指示することです。

アプリケーションに関する限り、キュー・マネージャーは、グローバル作業単位に対して、リソース・マネージャー (リソースがキュー上のメッセージである場合) およびトランザクション・マネージャーという 2 つの役割を担います。

付属のサンプル・プログラムから始め、これらのプログラム内で実行されるさまざまな WebSphere MQ 呼び出しおよびデータベース API 呼び出しの動作を確認してください。 関係する API 呼び出しについては、96 ページの『WebSphere MQ プログラムのサンプル』、MQI で使用されるデータ・タイプ、およびデータベース固有の資料 (データベース固有の API の場合) に詳しく記載されています。

システムのテスト

アプリケーションおよびシステムが正しく構成されていることを確認する方法は、テスト時にそれらを実行する方法のみです。 付属のサンプル・プログラムの 1 つを作成して実行することにより、システムの構成 (キュー・マネージャーとデータベース間での通信が正常に行えるかどうか) をテストできます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

Db2 の構成

DB2 のサポートおよび構成情報。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

サポートされる Db2 のレベルは、IBM WebSphere MQ の詳細なシステム要件 ページで定義されています。

注 : Db2 の 32 ビット・インスタンスは、キュー・マネージャーが 64 ビットのプラットフォームではサポートされていません。

以下の作業を行います。

1. 環境変数の設定の確認
2. Db2 スイッチ・ロード・ファイルの作成
3. リソース・マネージャー構成情報の追加
4. 必要な場合に、Db2 構成パラメーターの変更

この情報を、46 ページの『データベース調整のためのシステムの構成』 に記載されている一般情報と共にお読みください。

警告 : UNIX and Linux プラットフォームで db2profile を実行する場合、環境変数 LIBPATH および LD_LIBRARY_PATH が設定されます。 これらの環境変数を unset することをお勧めします。該当する「 スタートアップ・ガイド 」を参照してください。

Db2 環境変数の設定の確認

Db2 環境変数が、アプリケーション・プロセス**だけでなく**、キュー・マネージャー・プロセスに対しても設定されていることを確認してください。 特に、キュー・マネージャーを**始動する前に**、DB2INSTANCE 環境変数を必ず設定してください。 DB2INSTANCE 環境変数は、更新する Db2 データベースが含まれている Db2 インスタンスを識別します。 以下に例を示します。

- UNIX and Linux システムでは、以下を使用します。

```
export DB2INSTANCE=db2inst1
```

- Windows システムでは、以下を使用します。

```
set DB2INSTANCE=DB2
```

Db2 データベースを使用した Windows では、キュー・マネージャーを開始できるようにするために、ユーザー MUSR_MQADMIN を DB2USERS グループに追加する必要があります。

Db2 スイッチ・ロード・ファイルの作成

Db2 スイッチ・ロード・ファイルを作成する最も簡単な方法は、サンプル・ファイル xaswit.mak を使用する方法です。 このファイルは、各種のデータベース・プロダクトについてスイッチ・ロード・ファイルを作成するために WebSphere MQ により提供されます。

Windows システムの場合、 xaswit.mak はディレクトリー *MQ_INSTALLATION_PATH*\tools\c\samples\xatm に入っています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。 Microsoft Visual C++ を使用して Db2 スイッチ・ロード・ファイルを作成するには、以下を使用します。

```
nmake /f xaswit.mak db2swit.dll
```

生成されたスイッチ・ファイルは、 c:\Program Files\IBM\WebSphere MQ\exits に置かれます。

xaswit.mak は、ディレクトリー *MQ_INSTALLATION_PATH*/samp/xatm に入っています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。

xaswit.mak を編集し、使用している Db2 のバージョンに該当する行を コメント解除 してください。 その後で、次のコマンドを使用して MAKE ファイルを実行します。

```
make -f xaswit.mak db2swit
```

生成された 32 ビットのスイッチ・ロード・ファイルは、 /var/mqm/exits にあります。

生成された 64 ビット・スイッチ・ロード・ファイルは、 /var/mqm/exits64 にあります。

Db2 用のリソース・マネージャー構成情報の追加

キュー・マネージャー用の構成情報を変更して、Db2 をグローバル作業単位に参加するプログラムとして宣言する必要があります。 この方法による構成情報の変更は、48 ページの『キュー・マネージャーへの構成情報の追加』で詳しく説明されています。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、 WebSphere MQ エクスプローラーを使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。
- 他のすべてのシステムにおいて、キュー・マネージャーの qm.ini ファイルにある、XAResourceManager スタンザのスイッチ・ロード・ファイルの詳細を指定します。

52 ページの図 9 は、UNIX サンプルで、調整されるデータベースが mydbname と呼ばれる XAResourceManager 項目を示しています。この名前が XAOpenString で指定されています。

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=mydbname,myuser,mypasswd,toc=t
  ThreadOfControl=THREAD
```

図 *9. Db2* 用サンプル *XAResourceManager* 項目 *(UNIX* プラットフォーム*)*

注 :

1. ThreadOfControl=THREAD は、バージョン 8 より前の Db2 バージョンでは使用できません。 ThreadOfControl および XAOpenString パラメーター toc を、以下のいずれかの組み合わせに設定します。
   - ThreadOfControl=THREAD および toc=t
   - ThreadOfControl=PROCESS および toc=p

   jdbcdb2 XA スイッチ・ロード・ファイルを使用して JDBC/JTA 調整を有効にする場合は、 ThreadOfControl=PROCESS および toc=p を使用する必要があります。

Db2 構成パラメーターの変更

キュー・マネージャーが調整する各 Db2 データベースごとに、データベース権限を設定し、tp_mon_name パラメーターを変更し、maxappls パラメーターをリセットする必要があります。 これを行うには、以下のステップを実行します。

データベース特権の設定

キュー・マネージャー・プロセスは、UNIX and Linux システム上では有効なユーザーおよびグループ mqm で稼働します。 Windows システムでは、それらのプロセスは、キュー・マネージャーを始動したユーザーとして稼働します。 このユーザーは、次のいずれかになります。

1. strmqm コマンドを発行したユーザー
2. IBM MQSeries® Service COM サーバーを実行するユーザー

特に指定のない限り、このユーザーは MUSR_MQADMIN と呼ばれます。

xa_open ストリングでユーザー名とパスワードを指定しなかった場合、 Db2 は xa_open 呼び出しを認証するために **キュー・マネージャーを実行しているユーザー** を使用します。 このユーザー (例えば、UNIX and Linux システム上のユーザー mqm) がデータベース内に最小限の特権を持っていない場合、データベースは xa_open 呼び出しの認証を拒否します。

同じことが、アプリケーション・プロセスにも当てはまります。 xa_open ストリングにユーザー名とパスワードを指定しなかった場合、Db2 は、アプリケーションが実行されているユーザーを使用して、最初の MQBEGIN 中に作成される xa_open 呼び出しを認証します。 さらに、このユーザーがデータベース内に最小限の特権を持っていないと、これは機能しません。

例えば、以下の Db2 コマンドを発行して、mydbname データベース内で mqm ユーザーに接続権限を付与してください。

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

セキュリティーの詳細については、61 ページの『セキュリティーに関する考慮事項』を参照してください。

Windows TP_MON_NAME パラメーターの変更

Db2 for Windows システムの場合のみ、TP_MON_NAME 構成パラメーターを変更して、 Db2 が動的登録のためにキュー・マネージャーを呼び出すために使用する DLL を指定します。

コマンド db2 update dbm cfg using TP_MON_NAME mqmax を使用して、Db2 がキュー・マネージャーを呼び出すために使用するライブラリーとして、MQMAX.DLL を指定します。 これは PATH 内のディレクトリーに入っていなければなりません。

maxappls パラメーターのリセット

場合によっては *maxappls* パラメーターの設定を確認する必要があります。このパラメーターにより、1 つのデータベースに接続できるアプリケーションの数が制限されます。 47 ページの『データベース・プロダクトのインストールおよび構成』を参照してください。

Oracle の構成

Oracle のサポートと構成に関する情報。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

以下のステップを完了します。

1. 環境変数の設定の確認
2. Oracle スイッチ・ロード・ファイルの作成
3. リソース・マネージャー構成情報の追加
4. 必要な場合には、Oracle 構成パラメーターの変更

IBM WebSphere MQ がサポートする Oracle の現行レベル・リストは、IBM WebSphere MQ の詳細なシステム要件に関するページにあります。

Oracle 環境変数の設定の確認

Oracle 環境変数が、アプリケーション・プロセスだけでなく、キュー・マネージャー・プロセスに対しても設定されていることを確認します。 特に、キュー・マネージャーを始動する前に、必ず以下の環境変数を設定してください。

ORACLE_HOME
Oracle ホーム・ディレクトリー。 例えば、UNIX and Linux システムでは、以下を使用します。

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```

Windows システムでは、以下を使用します。

```
set ORACLE_HOME=c:\oracle\ora81
```

ORACLE_SID
使用している Oracle SID。 クライアント/サーバーの接続に Net8 を使用している場合、この環境変数を設定する必要はありません。 お手元の Oracle 資料を参照してください。

次の例は、UNIX and Linux システムにおけるこの環境変数の設定例です。

```
export ORACLE_SID=sid1
```

Windows システム上での同等値は、次のとおりです。

```
set ORACLE_SID=sid1
```

注 : PATH 環境変数は、バイナリー・ディレクトリー (ORACLE_INSTALL_DIR/VERSION/32BIT_NAME/bin や ORACLE_INSTALL_DIR/VERSION/64BIT_NAME/bin など) を含むように設定する必要があります。そうでない場合、マシンに oraclient ライブラリーがないことを示すメッセージが表示される場合があります。

Windows 64 ビット・システムでキュー・マネージャーを実行する場合は、 64 ビットと 32 ビットの両方の Oracle クライアントをインストールする必要があります。 両方のクライアントをインストールする必要があるのは、キュー・マネージャーが、32 ビットのスイッチ・ロード・ファイルを使用する 32 ビット・プロセスとして稼働し、そのスイッチ・ロード・ファイルによって 32 ビットの Oracle クライアント dll が開始されるためです。

64 ビットのキュー・マネージャーによってロードされたスイッチ・ロード・ファイルは、 Oracle 64 ビット・クライアント・ライブラリーにアクセスする必要があります。 IBM WebSphere MQ が Windows 64 ビット・システムで実行されている場合、 32 ビットのキュー・マネージャーは、 32 ビットの Oracle クライアントにアクセスする必要があります。

Oracle スイッチ・ロード・ファイルの作成

Oracle スイッチ・ロード・ファイルを作成するには、サンプル・ファイル xaswit.mak を使用します。 このファイルは、各種のデータベース製品用にスイッチ・ロード・ファイルを作成するために IBM WebSphere MQ により提供されます。 Windows システムでは、 xaswit.mak はディレクトリー C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm にあります。 Microsoft Visual C++ で Oracle スイッチ・ロード・ファイルを作成するには、 nmake /f xaswit.mak oraswit.dll を使用します。

生成されたスイッチ・ファイルは *MQ_INSTALLATION_PATH*\exits に置かれます。
MQ_INSTALLATION_PATH IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

xaswit.mak は、ディレクトリー *MQ_INSTALLATION_PATH*/samp/xatm にあります。
MQ_INSTALLATION_PATH IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

xaswit.mak を編集し、使用している Oracle のバージョンに対して適切な行をコメント解除してください。 その後で、次のコマンドを使用して MAKE ファイルを実行します。

```
make -f xaswit.mak oraswit
```

生成された 32 ビット・スイッチ・ロード・ファイルは、/var/mqm/exits に保管されます。

生成された 64 ビット・スイッチ・ロード・ファイルは、/var/mqm/exits64 に保管されます。

Oracle 用のリソース・マネージャー構成情報の追加

キュー・マネージャー用の構成情報を変更し、Oracle をグローバル作業単位に参加するプログラムとして宣言する必要があります。 キュー・マネージャー用の構成情報をこのように変更する方法については、48 ページの『キュー・マネージャーへの構成情報の追加』で詳しく説明されています。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、IBM WebSphere MQ Explorer を使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。
- 他のすべてのシステムでは、キュー・マネージャーの qm.ini ファイルにある XAResourceManager スタンザでスイッチ・ロード・ファイルの詳細を指定します。

55 ページの図 10 は、XAResourceManager 項目を示した UNIX and Linux システムのサンプルです。 XA オープン・ストリングに LogDir を追加して、すべてのエラーおよびトレース情報のログが同一の場所に記録されるようにする必要があります。

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

図 *10. Oracle* 用サンプル *XAResourceManager* 項目 *(UNIX and Linux* プラットフォーム*)*

注 :

1. 55 ページの図 10 では、xa_open ストリングと 4 つのパラメーターが使用されています。 Oracle の資料に説明されているように、追加のパラメーターを指定できます。
2. IBM WebSphere MQ パラメーターの ThreadOfControl=THREAD を使用する場合は、XAResourceManager スタンザで Oracle パラメーター +threads=true を使用する必要があります。

xa_open ストリングの詳細については、「*Oracle8 Server Application Developer's Guide*」を参照してください。

Oracle 構成パラメーターの変更

キュー・マネージャーが調整する各 Oracle データベースごとに、最大セッション数を確認してデータベース特権を設定する必要があります。 これを行うには、以下のステップを完了します。

最大セッションの確認
キュー・マネージャーにより制御されるプロセスでは、接続を増やす必要があることを考慮して、LICENSE_MAX_SESSIONS および PROCESSES の設定値を確認する必要がある場合があります。 詳細については、47 ページの『データベース・プロダクトのインストールおよび構成』を参照してください。

データベース特権の設定
xa_open ストリングに指定される Oracle ユーザー名は、Oracle 資料に記載されているとおり、DBA_PENDING_TRANSACTIONS ビューへのアクセス権限が付与されているものでなければなりません。

必要な特権は、以下のコマンド例を使用して付与できます。

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

Informix の構成

Informix のサポートと構成に関する情報。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

以下のステップを完了します。

1. 適切な Informix クライアント SDK がインストールされていることを確認します。
   - 32 ビットのキュー・マネージャーとアプリケーションでは、32 ビット Informix クライアント SDK が必要です。
   - 64 ビットのキュー・マネージャーとアプリケーションでは、64 ビット Informix クライアント SDK が必要です。
2. Informix データベースが 正しく作成されていることを確認します。
3. 環境変数の設定の確認
4. Informix スイッチ・ロード・ファイルを作成します。
5. リソース・マネージャー構成情報の追加

WebSphere MQ でサポートされる Informix のレベルの現行リストについては、IBM WebSphere MQ の詳細なシステム要件 のページを参照してください。

Informix データベースが正しく作成されていることの確認

WebSphere MQ キュー・マネージャーによって調整されるすべての Informix データベースは、`log` パラメーターを指定して作成する必要があります。 以下に例を示します。

```
create database mydbname with log;
```

WebSphere MQ キュー・マネージャーは、作成時に `log` パラメーターが指定されていない Informix データベースを調整できません。 作成時に `log` パラメーターが指定されていない Informix データベースをキュー・マネージャーが調整しようとすると、 Informix への xa_open 呼び出しは失敗し、いくつかの FFST エラーが生成されます。

Informix 環境変数の設定の確認

Informix 環境変数が、アプリケーション・プロセス**だけでなく**、キュー・マネージャー・プロセスに対しても設定されていることを確認してください。 特に、キュー・マネージャーを始動する**前**に、必ず以下の環境変数を設定してください。

INFORMIXDIR
Informix プロダクト・インストールのディレクトリー。

- 32 ビット UNIX and Linux アプリケーションの場合は、以下のコマンドを使用します。

  ```
  export INFORMIXDIR=/opt/informix/32-bit
  ```

- 64 ビット UNIX and Linux アプリケーションの場合は、以下のコマンドを使用します。

  ```
  export INFORMIXDIR=/opt/informix/64-bit
  ```

- Windows アプリケーションの場合は、以下のコマンドを使用します。

  ```
  set INFORMIXDIR=c:\informix
  ```

32 ビットと 64 ビットの両方のアプリケーションをサポートする必要がある 64 ビットのキュー・マネージャーを使用しているシステムの場合は、32 ビットと 64 ビット両方の Informix クライアント SDK

をインストールする必要があります。 スイッチ・ロード・ファイルを作成する場合に使用するサンプル MAKE ファイル xaswit.mak も、両方の製品インストール・ディレクトリーを設定します。

INFORMIXSERVER
Informix サーバーの名前。 例えば、UNIX and Linux システムでは、以下を使用します。

```
export INFORMIXSERVER=hostname_1
```

Windows システムでは、以下を使用します。

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG
Informix サーバーの構成ファイルの名前。 例えば、UNIX and Linux システムでは、以下を使用します。

```
export ONCONFIG=onconfig.hostname_1
```

Windows システムでは、以下を使用します。

```
set ONCONFIG=onconfig.hostname_1
```

Informix スイッチ・ロード・ファイルの作成

Informix スイッチ・ロード・ファイルを作成するには、サンプル・ファイル xaswit.mak を使用します。 このファイルは、各種のデータベース製品用にスイッチ・ロード・ファイルを作成するために WebSphere MQ により提供されます。 Windows システムの場合、 xaswit.mak はディレクトリー *MQ_INSTALLATION_PATH*\tools\c\samples\xatm に入っています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。 Microsoft Visual C++ を使用して Informix スイッチ・ロード・ファイルを作成するには、以下を使用します。

```
nmake /f xaswit.mak infswit.dll
```

生成されたスイッチ・ファイルは、 c:\Program Files\IBM\WebSphere MQ\exits に置かれます。

xaswit.mak は、ディレクトリー *MQ_INSTALLATION_PATH*/samp/xatm に入っています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。

xaswit.mak を編集し、使用している Informix のバージョンに該当する行を コメント解除 してください。 その後で、次のコマンドを使用して MAKE ファイルを実行します。

```
make -f xaswit.mak infswit
```

生成された 32 ビット・スイッチ・ロード・ファイルは、/var/mqm/exits に保管されます。

生成された 64 ビット・スイッチ・ロード・ファイルは、/var/mqm/exits64 に保管されます。

Informix 用のリソース・マネージャー構成情報の追加

キュー・マネージャー用の構成情報を変更し、Informix をグローバル作業単位に参加するプログラムとして宣言する必要があります。 キュー・マネージャー用の構成情報をこのように変更する方法については、48 ページの『キュー・マネージャーへの構成情報の追加』で詳しく説明されています。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、 WebSphere MQ エクスプローラーを使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。
- 他のすべてのシステムでは、キュー・マネージャーの qm.ini ファイルにある XAResourceManager スタンザでスイッチ・ロード・ファイルの詳細を指定します。

58 ページの図 11 は、UNIX サンプルで、調整されるデータベースが mydbname と呼ばれる qm.ini の XAResourceManager 項目を示しています。この名前が XAOpenString で指定されています。

```
XAResourceManager:
  Name=myinformix
  SwitchFile=infswit
  XAOpenString=DB=mydbname@myinformixserver\;USER=myuser\;PASSWD=mypasswd
  ThreadOfControl=THREAD
```

図 *11. Informix* 用サンプル *XAResourceManager* 項目 *(UNIX* プラットフォーム*)*

注 : デフォルトでは、UNIX プラットフォーム上のサンプル xaswit.mak は、スレッド化された Informix ライブラリーを使用するスイッチ・ロード・ファイルを作成します。 これらの Informix ライブラリーを使用するときには、ThreadOfControl が THREAD に設定されていることを確認する必要があります。 58 ページの図 11 では、qm.ini ファイルの XAResourceManager スタンザ属性 ThreadOfControl が THREAD に設定されています。 THREAD が指定されている場合、スレッド化された Informix ライブラリーおよび WebSphere MQ のスレッド化された API ライブラリーを使用してアプリケーションを構築する必要があります。

XAOpenString 属性には、 データベース名の後に @ 記号、その後に Informix サーバー名を付けた値を指定する必要があります。

スレッド化されていない Informix ライブラリーを使用する場合は、qm.ini ファイルの XAResourceManager スタンザの ThreadOfControl 属性を PROCESS に設定する必要があります。 さらに、サンプル xaswit.mak に以下の変更を加える必要があります。

1. スレッド化されていないスイッチ・ロード・ファイルの生成をアンコメントする。
2. スレッド化されたスイッチ・ロード・ファイルの生成をコメント化する。

Sybase 構成

Sybase のサポートおよび構成情報。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

以下のステップを完了します。

1. 例えば XA DTM オプションをインストールすることにより、Sybase XA ライブラリーをインストールしていることを確認します。
2. 環境変数の設定の確認
3. Sybase XA サポートの使用可能化
4. Sybase スイッチ・ロード・ファイルの作成
5. リソース・マネージャー構成情報の追加

WebSphere MQ がサポートする Sybase のレベルの現行リストは、IBM WebSphere MQ の詳細なシステム要件 ページで提供されています。

Sybase 環境変数の設定の確認

Sybase 環境変数が、アプリケーション・プロセス**だけでなく**、キュー・マネージャー・プロセスに対しても設定されていることを確認します。 特に、キュー・マネージャーを始動する**前**に、必ず以下の環境変数を設定してください。

SYBASE
Sybase プロダクト・インストールの位置。 例えば、UNIX and Linux システムでは、以下を使用します。

```
export SYBASE=/sybase
```

Windows システムでは、以下を使用します。

```
set SYBASE=c:\sybase
```

SYBASE_OCS
Sybase クライアント・ファイルをインストールした SYBASE の下のディレクトリー。 例えば、UNIX and Linux システムでは、以下を使用します。

```
export SYBASE_OCS=OCS-12_0
```

Windows システムでは、以下を使用します。

```
set SYBASE_OCS=OCS-12_0
```

Sybase XA サポートの使用可能化

Sybase XA 構成ファイル $SYBASE/$SYBASE_OCS/xa_config 内で、更新する Sybase サーバーへの接続ごとに論理リソース・マネージャー (Logical Resource Manager: LRM) を定義します。 $SYBASE/$SYBASE_OCS/xa_config の内容例を、59 ページの図 12 に示します。

```
# The first line must always be a comment
[xa]
  LRM=lrmname
  server=servername
```

図 *12. $SYBASE/$SYBASE_OCS/xa_config* の内容例

Sybase スイッチ・ロード・ファイルの作成

Sybase スイッチ・ロード・ファイルを作成するには、WebSphere MQ に付属しているサンプル・ファイルを使用します。 Windows システムの場合、 xaswit.mak はディレクトリー C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm に入っています。 Microsoft Visual C++ で Sybase スイッチ・ロード・ファイルを作成するには、以下を使用します。

```
nmake /f xaswit.mak sybswit.dll
```

生成されたスイッチ・ファイルは、 c:\Program Files\IBM\WebSphere MQ\exits に置かれます。

xaswit.mak は、ディレクトリー *MQ_INSTALLATION_PATH*/samp/xatm に入っています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。

xaswit.mak を編集し、使用している Sybase のバージョンに対して適切な行をコメント解除 してください。 その後で、次のコマンドを使用して MAKE ファイルを実行します。

```
make -f xaswit.mak sybswit
```

生成された 32 ビットのスイッチ・ロード・ファイルは、 /var/mqm/exits にあります。

生成された 64 ビット・スイッチ・ロード・ファイルは、 /var/mqm/exits64 にあります。

Sybase 用のリソース・マネージャー構成情報の追加

キュー・マネージャー用の構成情報を変更して、Sybase をグローバル作業単位に参加するプログラムとして宣言する必要があります。 構成情報の変更は、48 ページの『キュー・マネージャーへの構成情報の追加』で詳しく説明されています。

- Windows および Linux (x86 および x86-64 プラットフォーム) システムでは、 WebSphere MQ エクスプローラーを使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。
- 他のすべてのシステムにおいて、キュー・マネージャーの qm.ini ファイルにある、XAResourceManager スタンザのスイッチ・ロード・ファイルの詳細を指定します。

60 ページの図 13 は UNIX and Linux のサンプルを示しています。このサンプルでは、Sybase XA 構成ファイル $SYBASE/$SYBASE_OCS/xa_config 内の *lrmname* LRM 定義と関連付けられたデータベースを使用します。 XA 関数呼び出しをログに記録する場合は、ログ・ファイル名を組み込みます。

```
XAResourceManager:
  Name=mysybase
  SwitchFile=sybswit
  XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
  ThreadOfControl=THREAD
```

図 *13. Sybase* 用サンプル *XAResourceManager* 項目 *(UNIX and Linux* プラットフォーム*)*

Sybase でのマルチスレッド・プログラムの使用

Sybase への更新を組み込む WebSphere MQ グローバル作業単位でマルチスレッド化プログラムを使用する場合は、ThreadOfControl パラメーターに値 THREAD を使用する**必要があります**。 さらに、プログラム (およびスイッチ・ロード・ファイル) とスレッド・セーフな Sybase ライブラリー (_r バージョン) とをリンクしていることを確認してください。 ThreadOfControl パラメーターでの値 THREAD の使用については、60 ページの図 13 で示されています。

複数のデータベースの構成

複数のデータベースに対する更新がグローバル作業単位に含まれるようにキュー・マネージャーを構成するには、データベースごとに XAResourceManager スタンザを追加します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

すべてのデータベースが同じデータベース・マネージャーによって管理される場合、データベースは各スタンザにより個別に定義されます。 各スタンザは同一の *SwitchFile* を指定しますが、*XAOpenString* の内容は異なります。これは、各スタンザにより、更新されるデータベースの名前が指定されるためです。 例えば、 61 ページの図 14 に示すスタンザは、 UNIX and Linux システム上の Db2 データベース *MQBankDB* および *MQFeeDB* を使用してキュー・マネージャーを構成します。

重要 : 複数のスタンザで同じデータベースを指定することはできません。 どのような環境においても、このような構成は機能しません。このような構成を試した場合は失敗します。

when the MQ code makes its second xa_open call in any process in this environment, the database software fails the second xa_open with a -5 error, XAER_INVAL という形式のエラーを受け取ります。

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=DB2 MQFeeDB
  SwitchFile=db2swit
  XAOpenString=MQFeeDB
```

図 *14.* 複数の *Db2* データベース用サンプル *XAResourceManager* 項目

更新されるデータベースが異なるデータベース・マネージャーによって管理される場合、それぞれに対して XAResourceManager スタンザを追加します。 この場合、各スタンザには異なる *SwitchFile* が指定されます。 例えば、*MQFeeDB* が DB2 ではなく Oracle により管理されている場合、UNIX and Linux システムで次のスタンザを使用します。

```
XAResourceManager:
  Name=DB2 MQBankDB
  SwitchFile=db2swit
  XAOpenString=MQBankDB

XAResourceManager:
  Name=Oracle MQFeeDB
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/myuser/mypassword+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

図 *15. DB2* および *Oracle* データベース用サンプル *XAResourceManager* 項目

原則として、1 つのキュー・マネージャーで構成できるデータベース・インスタンスの数に制限はありません。

注 : グローバル作業単位内の複数のデータベース更新に Informix データベースを組み込む場合のサポートについて詳しくは、製品の README ファイルを確認してください。

セキュリティーに関する考慮事項

XA モデルでのデータベースの実行に関する考慮事項。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

以下の情報は、参考のためにのみ記載しています。 いかなる場合でも、XA モデルでのデータベースの運用に関するセキュリティーの含意事項については、データベース・マネージャーに添付されている資料を参照してください。

アプリケーション・プロセスは、MQBEGIN によってグローバル作業単位の開始を指示します。 アプリケーションが実行する最初の MQBEGIN 呼び出しでは、各参加データベースのクライアント・ライブラリー・コードが xa_open エントリー・ポイントで呼び出されて、参加データベースにすべて接続されます。 すべてのデータベース・マネージャーが、ユーザー ID およびパスワードを XAOpenString に指定できるメカニズムを提供しています。 認証情報が流れるのは、このときだけです。

UNIX and Linux プラットフォームで MQI を呼び出す場合、ファースト・パス・アプリケーションを実行するには、mqm の有効なユーザー ID を使用しなければならないことに注意してください。

XA リソース・マネージャーとの連絡が失われる際の考慮事項

キュー・マネージャーは、データベース・マネージャーが使用できない状況でもその影響をあまり受けません。 これは、データベース・サーバーとは別にキュー・マネージャーを始動および停止できることを意味しています。 連絡が復元されるとき、キュー ・マネージャーとデータベースは再同期されます。 rsvmqtrn コマンドを使用して、未確定の作業単位を手動で解決することもできます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

通常の運用では、構成手順の完了後には、最小限の管理作業のみが必要とされます。 キュー・マネージャーは、データベース・マネージャーが使用できない状況でもその影響を受けることが少ないため、管理ジョブは単純なものになっています。 具体的には、次のような特徴があります。

- キュー・マネージャーを始動する際に、事前に各データベース・マネージャーを始動する必要はありません。
- いずれかのデータベース・マネージャーが使用できなくなった場合に、キュー・マネージャーを停止して再始動する必要はありません。

これにより、データベース・サーバーとは独立してキュー・マネージャーを始動および停止できます。

キュー・マネージャーとデータベース間の連絡が失われた場合は、両方が使用可能になった後に、再同期を取る必要があります。 再同期とは、該当するデータベースが関連する未確定の作業単位を完了するプロセスです。 通常、再同期は自動的に実行されるため、ユーザーが介入する必要はありません。 キュー・マネージャーはデータベースに対して、未確定の作業単位のリストを要求します。 キュー・マネージャーは次に、未確定の作業単位をそれぞれコミットまたはロールバックするようにデータベースに指示します。

キュー・マネージャーが始動すると、各データベースと再同期を取ります。 あるデータベースが使用できない状態になった場合、このデータベースが使用可能な状態に戻ったとキュー・マネージャーが次に認識した時点で、キュー・マネージャーが再同期を取る必要があるのはこのデータベースのみです。

MQBEGIN を使用して新しいグローバル作業単位が開始されると、キュー・マネージャーは、以前に使用できない状態にあったデータベースとの連絡を自動的に回復します。 これは、データベース・クライアント・ライブラリー内の xa_open 関数を呼び出して行われます。 この xa_open の呼び出しが失敗した場合、MQBEGIN は、完了コード MQCC_WARNING と理由コード MQRC_PARTICIPANT_NOT_AVAILABLE を戻します。 MQBEGIN 呼び出しは、後で再試行できます。

データベースへの更新に関与するグローバル作業単位で、MQBEGIN 中に失敗したことが示される場合、この作業単位の試行を続行しないでください。 そのデータベースに接続して更新を行うことはできません。 行える操作は、プログラムを終了するか、データベースが再度使用可能な状態になることを期待して MQBEGIN を定期的に再試行することのみです。

別の方法として、`rsvmqtrn` コマンドを使用して、未確定の作業単位をすべて明示的に解決することもできます。

未確定の作業単位

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

準備するようデータベース・マネージャーが通知を受けた後にキュー・マネージャーとの連絡を失った場合、データベースに未確定の作業単位が残されることがあります。 データベース・サーバーは、キュー・マネージャーから実行結果 (コミットまたはロールバック) を受け取るまでは、更新に関連するデータベース・ロックを保持する必要があります。

このロックにより、他のアプリケーションがデータベース・レコードの更新または読み取りを行えなくなるため、できるだけ早く再同期を実行する必要があります。

何らかの理由により、キュー・マネージャーによるデータベースの自動再同期よりも早い時点で再同期を行わなければならない場合には、データベース・マネージャーに付属の機能を使用してデータベースの更新を手動でコミットまたはロールバックすることもできます。 「*X/Open Distributed Transaction Processing: The XA Specification*」では、これをヒューリスティック 判定の実行と呼んでいます。 データ保全性を損なう可能性があるため、これはできるだけ使用しないでください。例えば、その他のすべての参加プログラムがそれぞれの更新をコミット済みであるときに、誤ってデータベースの更新をロールバックしてしまう可能性があります。

キュー・マネージャーを再始動するか、データベースが既に再始動している場合には `rsvmqtrn` コマンドを使用して、自動再同期を開始するほうがはるかに良い方法です。

dspmqtrn コマンドを使用した未解決の作業単位の表示

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

データベース・マネージャーが使用不可になっている場合には、**dspmqtrn** コマンドを使用して、該当するデータベースが関与する未解決のグローバル作業単位の状態を確認できます。

dspmqtrn コマンドを実行すると、1 つ以上の参加プログラムが未確定の状態になっている作業単位のみが表示されます。 参加プログラムは、準備済みの更新をコミットするかロールバックするかについてキュー・マネージャーからの決定を待機しています。

これらのグローバル作業単位ごとに、それぞれの参加プログラムの状態が **dspmqtrn** からの出力に表示されます。 特定のリソース・マネージャーのリソースを更新しなかった作業単位は表示されません。

未確定の作業単位に関して、リソース・マネージャーは、次のいずれかが済んだ状態と見なされます。

準備済み
リソース・マネージャーでは更新をコミットする準備ができています。

コミット済み
リソース・マネージャーによる更新のコミットが既に実行されています。

ロールバック済み
リソース・マネージャーによる更新のロールバックが既に実行されています。

参加済み
リソース・マネージャーは参加プログラムですが、更新の準備、コミット、およびロールバックが済んでいません。

キュー・マネージャーは、再始動時に、XAResourceManager スタンザを持つ各データベースに、その未確定のグローバル作業単位のリストを要求します。 データベースが再開されていない場合、または使用できない状態の場合、キュー・マネージャーは、それらの作業単位の最終結果をデータベースにまだ送信できません。 未確定の作業単位の結果は、その後初めてデータベースが再度使用できる状態になったときに、データベースに送信されます。

この場合、データベース・マネージャーは、再同期が行われるまで、準備済み 状態にあるとして報告されます。

dspmqtrn コマンドにより未確定の作業単位が表示される場合には常に、参加している可能性のあるリソース・マネージャーすべてのリストが最初に示されます。 これらのリソース・マネージャーには固有 ID である *RMId* が割り振られています。この ID は、未確定の作業単位に関してリソース・マネージャーの状態が報告される場合に、リソース・マネージャーの名前 の代わりに使用されます。

dspmqtrn による出力の例に、次のコマンドの発行結果を示します。

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
    XID: formatID 5067085, gtrid_length 12, bqual_length 4
         gtrid [3291A50600002013746573 74]
         bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

Transaction number は、rsvmqtrn コマンドで使用できるトランザクションの ID です。 AMQ7056 メッセージの詳細については、AMQ7000-7999: WebSphere MQ 製品を参照してください。 *XID* 変数は、*X/Open XA* 仕様の一部です。この仕様の最新情報については、 https://publications.opengroup.org/c193 を参照してください。

図 *16. dspmqtrn* による出力の例

dspmqtrn による出力の例の出力は、キュー・マネージャーに 3 つのリソース・マネージャーが関連付けられていることを示しています。 最初のリソース・マネージャー 0 は、キュー・マネージャー自身です。 その他の 2 つのリソース・マネージャー・インスタンスは、 MQBankDB データベースと MQFeeDB Db2 データベースです。

この例では、未確定の作業単位は 1 つのみです。 3 つのリソース・マネージャーすべてについてメッセージが出されます。これは、作業単位内のキュー・マネージャーと両方の Db2 データベースに対して更新が行われたことを意味します。

キュー・マネージャー、つまりリソース・マネージャー **0** に対して行われた更新は、コミット済み の状態です。 Db2 データベースに対する更新は 準備済み 状態です。つまり、 *MQBankDB* および *MQFeeDB* データベースに対する更新をコミットするために呼び出される前に、 Db2 が使用不可になっている必要があります。

未確定の作業単位には、XID (トランザクション *ID*) という外部 ID が付いています。 これは、グローバル作業単位の部分を識別するためにキュー・マネージャーによって Db2 に渡されるデータです。

rsvmqtrn コマンドを使用した未解決の作業単位の解決

未解決の作業単位は、キュー・マネージャーと DB2 が再同期したときに完了します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

64 ページの図 16 に示す出力には、両方の DB2 データベースに対してコミットの決定がまだ送信されていない未確定の作業単位が 1 つあることが示されています。

この作業単位を完了するには、DB2 が次に使用可能な状態になった時点で、キュー・マネージャーと DB2 を再同期する必要があります。 キュー・マネージャーは、新しい作業単位の開始を利用して、DB2 との連絡を再度獲得します。 または、**rsvmqtrn** コマンドを使用して明示的に再同期を実行するよう、キュー・マネージャーに指示することもできます。

未確定の作業単位に関連するすべてのデータベース・ロックをできるだけ早く解放するために、DB2 が再始動した直後にこの指示を行ってください。 キュー・マネージャーに対してすべての未確定の作業単位を解決するよう指示する -a オプションを使用します。 次の例では、DB2 が再始動しているため、キュー・マネージャーが未確定の作業単位を解決できます。

```
> rsvmqtrn -m MY_QMGR -a
Any in-doubt transactions have been resolved.
```

実行結果の混在とエラー

キュー・マネージャーは 2 フェーズ・コミット・プロトコルを使用しますが、これにより作業単位の実行結果が混在する可能性が完全に排除されるわけではありません。 参加プログラムのうち、一部が更新をコミットし、一部が更新をバックアウトした場合に、このような混在が発生します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

完了した作業単位の実行結果が混在していると、深刻な影響が出ます。1 つの作業単位として更新すべきだったのに更新されなかった共用リソースは、整合性のない状態になってしまうためです。

実行結果の混在は、キュー・マネージャーによって未確定の作業単位を解決せずに、ヒューリスティック判定に基づいて作業単位の処理を行った場合に主に発生します。 この判定は、キュー・マネージャーの制御を受けません。

キュー・マネージャーは実行結果の混在を検出するたびに、FFST 情報を作成し、エラー・ログに障害を記録して、次の 2 つのメッセージのいずれかを表示します。

- データベース・マネージャーがコミットの代わりにロールバックを行った場合には次のメッセージが表示されます。

```
AMQ7606 A transaction has been committed but one or more resource
        managers have rolled back.
```

- データベース・マネージャーがロールバックの代わりにコミットを行った場合には次のメッセージが表示されます。

```
AMQ7607 A transaction has been rolled back but one or more resource
        managers have committed.
```

さらに、ヒューリスティック判定によって障害が発生したデータベースを識別するメッセージが表示されます。 この場合、ユーザーは障害のあるデータベースとの整合性をローカルで復元する必要があります。この手順は複雑です。まず、誤ってコミットまたはロールバックした更新を分離し、次にデータベースの変更を手動で取り消すか、再実行しなければなりません。

構成情報の変更

キュー・マネージャーが正常に始動し、グローバル作業単位を調整する準備ができた後は、リソース・マネージャーの構成情報を変更しないでください。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

構成情報は、変更する必要があればいつでも変更できますが、変更内容が有効になるのはキュー・マネージャーの再始動後です。

データベースに対するリソース・マネージャー構成情報を削除すると、実際にはキュー・マネージャーがデータベース・マネージャーと連絡をとる機能が除去されることになります。

どのリソース・マネージャー構成情報でも、*Name* 属性を **変更しないでください** 。 この属性により、キュー・マネージャーに対してデータベース・マネージャーのインスタンスが一意的に識別されます。 この固有の ID が変更されると、キュー・マネージャーは、データベースが削除され、まったく新しいインスタンスが追加されているものと想定します。 その場合でも、キュー・マネージャーは未解決の作業単位をその古い *Name* に関連付けており、データベースを未確定の状態にする可能性があります。

データベース・マネージャーのインスタンスの削除

構成からデータベースを完全に削除する必要がある場合は、キュー・マネージャーを再始動する前に、データベースが未確定状態でないことを確認してください。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

データベース製品には、未確定のトランザクションをリストするためのコマンドが用意されています。 未確定のトランザクションがある場合、まず、キュー・マネージャーとデータベースを再同期させます。 これを行うには、キュー・マネージャーを始動します。 **rsvmqtrn** コマンドか、未確定の作業単位を表示するためのデータベース固有のコマンドを使用して、再同期が行われたかどうかを確認できます。 再同期が行われたことを確認できたら、キュー・マネージャーを終了し、データベースの構成情報を削除します。

この手順に従わない場合、キュー・マネージャーはそのデータベースに関与するすべての未確定の作業単位を記憶したままになります。 キュー・マネージャーの再始動時に毎回警告メッセージ AMQ7623 が出されます。 今後、キュー・マネージャーを使ってこのデータベースの構成を再び行うことがない場合は、**rsvmqtrn** コマンドの -r オプションを使用して、未確定トランザクションへのこのデータベースの関与に関する記憶を消去するようキュー・マネージャーに指示します。 キュー・マネージャーは、すべての参加プログラムで未確定トランザクションが完了した場合にのみ、これらのトランザクションに関する記憶を消去します。

いくつかのリソース・マネージャー構成情報を一時的に削除する必要がある場合があります。 UNIX and Linux システムでは、後で簡単に復元できるように、スタンザをコメント化して構成情報を一時的に削除するのが最も良い方法です。 キュー・マネージャーが特定のデータベースまたはデータベース・マネージャーと連絡を取るたびにエラーが発生する場合、この処置を行うことをお勧めします。 対象のリソース・マネージャー構成情報を一時的に削除すると、キュー・マネージャーはその他すべての参加プログラムが関与するグローバル作業単位を開始できます。 XAResourceManager スタンザのコメント化の例を以下に示します。

```
# This database has been temporarily removed
#XAResourceManager:
#  Name=mydb2
#  SwitchFile=db2swit
#  XAOpenString=mydbname,myuser,mypassword,toc=t
#  ThreadOfControl=THREAD
```

図 *17. UNIX and Linux* システムでの *XAResourceManager* スタンザのコメント化

Windows システムでは、WebSphere MQ Explorer を使用して、データベース・マネージャー・インスタンスに関する情報を削除します。 *Name* フィールドを復元する際に、正しい名前を入力するよう、十分注意してください。 間違った名前を入力した場合、65 ページの『構成情報の変更』に記載されているとおり、未確定問題が発生することがあります。

XA 動的登録

XA 仕様には、トランザクション・マネージャーがリソース・マネージャーに対して実行する xa_* 呼び出しの回数を減らす仕組みがあります。 この最適化の仕組みを動的登録 といいます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

動的登録は DB2 でサポートされています。 他のデータベースでもサポートしているものがあります。詳しくは、ご使用のデータベース製品の資料を参照してください。

動的登録最適化はなぜ有用なのでしょうか。 アプリケーションには、データベース・テーブルの更新が含まれているグローバル作業単位もあれば、そのような更新が含まれていないグローバル作業単位もあります。 データベースのテーブルに対して永続的な更新が行われていない場合、MQCMIT 中に発生するコミット・プロトコルにそのデータベースを組み込む必要はありません。

データベースが動的登録をサポートしているかどうかに関係なく、アプリケーションは WebSphere MQ 接続での最初の MQBEGIN 呼び出し中に xa_open を呼び出します。 アプリケーションは、後続の MQDISC 呼び出しで xa_close も呼び出します。 後続の XA 呼び出しのパターンは、データベースが動的登録をサポートしているかどうかによって異なります。

データベースが動的登録をサポートしない場合
作業単位内のそのデータベースのテーブルに対して永続的な更新を行ったかどうかにかかわらず、各グローバル作業単位には、WebSphere MQ コードによって実行されてデータベース・クライアント・ライブラリーに入れられたいくつかの XA 関数呼び出しが含まれます。 これには以下が含まれます。

- アプリケーション・プロセスからの xa_start および xa_end。 これらは、グローバル作業単位の始めと終わりを宣言するために使用されます。
- キュー・マネージャー・エージェント・プロセス amqzlaa0 からの xa_prepare、xa_commit、および xa_rollback。 これらは、グローバル作業単位の結果、つまりコミットまたはロールバックの決定を送信するために使用されます。

さらに、キュー・マネージャー・エージェント・プロセスは、最初の MQBEGIN 中に xa_open も呼び出します。

データベースが動的登録をサポートする場合
WebSphere MQ コードは、必要な XA 関数呼び出しのみを実行します。 データベース・リソースに対する永続的更新が**伴わない**グローバル作業単位の場合、データベースへの XA 呼び出しは**ありません**。 そのような永続的更新が**伴う**グローバル作業単位の場合、次の呼び出しが行われます。

- アプリケーション・プロセスからの、グローバル作業単位の終わりを宣言する xa_end。
- キュー・マネージャー・エージェント・プロセス amqzlaa0 からの xa_prepare、xa_commit、および xa_rollback。 これらは、グローバル作業単位の結果、つまりコミットまたはロールバックの決定を送信するために使用されます。

動的登録が機能するためには、データベースが現行のグローバル作業単位に含める永続的更新を実行したことを WebSphere MQ に通知する仕組みがデータベースに用意されていることが不可欠です。
WebSphere MQ には、この目的のために ax_reg 関数が用意されています。

アプリケーション・プロセスで実行するデータベースのクライアント・コードは、ax_reg 関数を検出してそれを呼び出し、現行のグローバル作業単位内で永続的な作業を実行したという事実を動的に登録 します。 この ax_reg 呼び出しに対する応答として、WebSphere MQ は、データベースが参加したことを記録します。 これがこの WebSphere MQ 接続での最初の ax_reg 呼び出しである場合、キュー・マネージャー・エージェント・プロセスは xa_open を呼び出します。

データベース・クライアント・コードは、プロセスで実行中に (例えば SQL UPDATE 呼び出し中や、データベースのクライアント API が関与するあらゆる呼び出し中に)、この ax_reg 呼び出しを実行します。

エラー状態
XA 動的登録では、分かりにくい障害がキュー・マネージャーで発生する可能性があります。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

一般的な例としては、キュー・マネージャーを始動する前にデータベース環境変数を正しく設定することを忘れたために、キュー・マネージャーの xa_open に対する呼び出しが失敗する場合があります。 どのグローバル作業単位も使用できません。

このようなことを回避するために、キュー・マネージャーを始動する前に関連する環境変数を設定してあることを確認してください。 ご使用のデータベース製品の資料、および 51 ページの『Db2 の構成』、53 ページの『Oracle の構成』、および 58 ページの『Sybase 構成』に記載されているアドバイスを確認してください。

すべてのデータベース製品において、キュー・マネージャーは、リカバリー・セッションの一部として、キュー・マネージャーの始動時に xa_open を一度呼び出します (61 ページの『XA リソース・マネージャーとの連絡が失われる際の考慮事項』で説明しています)。 データベース環境変数を間違って設定した場合、この xa_open 呼び出しは失敗しますが、これはキュー・マネージャーが始動できない原因ではありません。 キュー・マネージャーが始動できないのは、データベース・サーバーが使用できない状態であることを示すのにデータベース・クライアント・ライブラリーが同じ xa_open エラー・コードを使用するためです。 キュー・マネージャーは、該当のデータベースが関与するグローバル作業単位の外側でデータ処理の続行を開始できるはずなので、WebSphere MQ では、これを重大なエラーとしては扱いません。

xa_open への後続の呼び出しは、WebSphere MQ 接続での最初の MQBEGIN 中 (動的登録が使用されていない場合) またはデータベース・クライアント・コードによる WebSphere MQ 提供の ax_reg 関数への呼び出し中 (動的登録が使用されている場合) に、キュー・マネージャーから実行されます。

エラー条件の **タイミング** (場合によっては FFST レポート) は、動的登録を使用しているかどうかによって異なります。

- 動的登録を使用している場合、MQBEGIN 呼び出しは正常に実行されますが、SQL UPDATE (または類似の) データベース呼び出しは失敗します。
- 動的登録を使用していない場合、MQBEGIN 呼び出しは失敗します。

アプリケーション・プロセスおよびキュー・マネージャー・プロセスに環境変数が正しく設定されていることを確認してください。

XA 呼び出しの要約
ここでは、グローバル作業単位を制御するさまざまな MQI 呼び出しの結果として、データベース・クライアント・ライブラリー内の XA 関数に対して行われる呼び出しのリストを示します。 ここでは、XA 仕様で記述されるプロトコルの完全な説明ではなく、概要を示します。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

xa_start 呼び出し、および xa_end 呼び出しは、必ずアプリケーション・プロセス内で WebSphere MQ コードによって行われるのに対し、xa_prepare、xa_commit、および xa_rollback は、必ずキュー・マネージャー・エージェント・プロセス amqzlaa0 から呼び出されます。

この表に示されている xa_open 呼び出しと xa_close 呼び出しはすべて、アプリケーション・プロセスから実行されます。 キュー・マネージャー・エージェント・プロセスは、67 ページの『エラー状態』で説明されている状況で、xa_open を呼び出します。

表 *7. XA* 関数呼び出しの要約

MQI 呼び出し	**動的登録付きで行われる XA 呼び出し**	**動的登録なしで行われる XA 呼び出し**
最初の MQBEGIN	`xa_open`	`xa_open` `xa_start`
後続の MQBEGIN	XA 呼び出しなし	`xa_start`

MQCMIT (ax_reg は現行のグローバル作業単位中に**呼び出されない**)	XA 呼び出しなし	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (ax_reg は現行のグローバル作業単位中に**呼び出される**)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		適用されません。 非動的モードでは ax_reg への呼び出しは行われません。

MQBACK (ax_reg は現行のグローバル作業単位中に**呼び出されない**)	XA 呼び出しなし	`xa_end` `xa_rollback`

表 7. *XA* 関数呼び出しの要約 *(続き)*

MQI 呼び出し	**動的登録付きで行われる XA 呼び出し**	**動的登録なしで行われる XA 呼び出し**
MQBACK (ax_reg は現行のグローバル作業単位中に**呼び出される**)	`xa_end` `xa_rollback`	適用されません。 非動的モードでは ax_reg への呼び出しは行われません。
MQDISC。MQCMIT または MQBACK が最初に呼び出されたもの。 それらが呼び出されなかった場合、MQCMIT 処理は、MQDISC 中に最初に行われます。	`xa_close`	`xa_close`

注 :

1. MQCMIT では、`xa_prepare` が正常に実行された場合に `xa_commit` が呼び出されます。 そうでない場合は、`xa_rollback` が呼び出されます。

シナリオ 2: 他のソフトウェアが調整を行う

シナリオ 2 では、外部トランザクション・マネージャーがグローバル作業単位を調整し、トランザクション・マネージャーの API の制御を受けてそれらを開始してコミットします。 MQBEGIN、MQCMIT、および MQBACK の各 verb は使用できません。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

ここでは、次の事項を含め、このシナリオについて説明します。

- 69 ページの『外部同期点調整』
- 72 ページの『CICS の使用』
- 76 ページの『Microsoft Transaction Server (COM+) の使用』

IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) は、HP NonStop Transaction Management Facility (TMF) を使用してグローバル作業単位を調整できます。 詳しくは、HP NonStop TMF の使用を参照してください。

外部同期点調整

グローバル作業単位は、外部 X/Open XA 準拠のトランザクション・マネージャーによって調整することもできます。 この場合、WebSphere MQ キュー・マネージャーは作業単位に関与していますが、作業単位の調整は行いません。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

外部トランザクション・マネージャーによって調整されるグローバル作業単位の制御のフローは次のとおりです。

1. トランザクションを開始することをアプリケーションが外部同期点調整プログラム (TXSeries など) に通知します。
2. 同期点調整プログラムが WebSphere MQ などの既知のリソース・マネージャーに現行トランザクションを通知します。
3. アプリケーションは、現行トランザクションに関連付けられたリソース・マネージャーに対して呼び出しを実行します。 例えば、アプリケーションは MQGET 呼び出しを WebSphere MQ に対して実行することがあります。
4. アプリケーションは外部同期点調整プログラムにコミット要求またはバックアウト要求を出します。

5. 同期点調整プログラムは、通常 2 フェーズ・コミット・プロトコルを使用して、適切な呼び出しを各リソース・マネージャーに出し、トランザクションを完了します。

WebSphere MQ が関与するトランザクションの 2 フェーズ・コミット・プロセスを提供する外部同期点調整プログラムのサポートされるレベルは、IBM WebSphere MQ の詳細なシステム要件で定義されています。

このセクションの後半では、外部作業単位を使用可能にする方法について説明します。

IBM WebSphere MQ XA スイッチ構造

外部から調整される作業単位に関与する各リソース・マネージャーには、XA スイッチ構造体がなければなりません。 この構造体により、リソース・マネージャーの機能と、同期点調整プログラムによって呼び出される関数の両方が定義されます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

IBM WebSphere MQ には、この構造体について、次の 2 種類が用意されています。

- *MQRMIXASwitch* (静的 XA リソース管理用)
- *MQRMIXASwitchDynamic* (動的 XA リソース管理用)

ご使用のトランザクション・マネージャーの資料を参照して、静的リソース管理インターフェースまたは動的リソース管理インターフェースのいずれを使用するかを決定してください。 トランザクション・マネージャーでサポートされている場合は必ず、動的 XA リソース管理を使用することをお勧めします。

64 ビットのトランザクション・マネージャーの中には、XA 仕様にある *long* 型を 64 ビットとして扱うものもあれば、32 ビットとして扱うものもあります。 WebSphere MQ は両方のモデルをサポートします。

- ご使用のトランザクション・マネージャーが 32 ビットである場合や、ご使用のトランザクション・マネージャーは 64 ビットであるが *long* 型を 32 ビットとして扱う場合は、70 ページの表 8 にリストされているスイッチ・ロード・ファイルを使用してください。
- ご使用のトランザクション・マネージャーが 64 ビットで、 *long* 型を 64 ビットとして扱う場合は、71 ページの表 9 にリストされているスイッチ・ロード・ファイルを使用してください。

long 型を 64 ビットとして扱う既知の 64 ビットのトランザクション・マネージャーのリストは、71 ページの表 10 に示されています。 ご使用のトランザクション・マネージャーが使用するモデルが不明な場合は、トランザクション・マネージャーの資料を参照してください。

表 *8. XA* スイッチ・ロード・ファイル名

プラットフォーム	**スイッチ・ロード・ファイル名 (サーバー)**	**スイッチ・ロード・ファイル名 (拡張トランザクション・クライアント)**
Windows	*mqmxa.dll*	*mqcxa.dll*
AIX (スレッドなし)	*libmqmxa.a*	*libmqcxa.a*
AIX (スレッドあり)	*libmqmxa_r.a*	*libmqcxa_r.a*
HP-UX (スレッドなし)	*libmqmxa.so*	*libmqcxa.so*
HP-UX (スレッドあり)	*libmqmxa_r.so*	*libmqcxa_r.so*
Linux (スレッドなし)	*libmqmxa.so*	*libmqcxa.so*
Linux (スレッド化)	*libmqmxa_r.so*	*libmqcxa_r.so*
Solaris	*libmqmxa.so*	*libmqcxa.so*

表 9. 64 ビットの代替 XA スイッチ・ロード・ファイル名		
プラットフォーム	**スイッチ・ロード・ファイル名 (サーバー)**	**スイッチ・ロード・ファイル名 (拡張トランザクション・クライアント)**
AIX (スレッドなし)	*libmqmxa64.a*	*libmqcxa64.a*
AIX (スレッドあり)	*libmqmxa64_r.a*	*libmqcxa64_r.a*
HP-UX (スレッドなし)	*libmqmxa64.so*	*libmqcxa64.so*
HP-UX (スレッドあり)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Linux (スレッドなし)	*libmqmxa64.so*	*libmqcxa64.so*
Linux (スレッド化)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Solaris	*libmqmxa64.so*	*libmqcxa64.so*

表 10. 64 ビットの代替スイッチ・ロード・ファイルを必要とする 64 ビットのトランザクション・マネージャー
トランザクション・マネージャー
Tuxedo

一部の外部同期点調整プログラム (CICS 以外) では、XA スイッチ構造体の名前フィールドに作業単位に関与する各リソース・マネージャーの名前を指定する必要があります。 WebSphere MQ のリソース・マネージャーの名前は MQSeries_XA_RMI です。

WebSphere MQ XA スイッチ構造体が同期点調整プログラムにリンクする方法は、同期点調整プログラムが定義します。 WebSphere MQ XA スイッチ構造体と CICS とのリンクの詳細については、72 ページの『CICS の使用』に記載されています。 WebSphere MQ XA スイッチ構造体とその他の XA 準拠同期点調整プログラムとのリンクの詳細については、各プログラムに付属の資料を参照してください。

WebSphere MQ を XA 準拠同期点調整プログラムと共に使用する場合には必ず、次の事項を考慮してください。

- 同期点調整プログラムによる xa_open 呼び出しに渡される xa_info 構造体には必ず、WebSphere MQ キュー・マネージャーの名前が含まれています。 この名前の形式は、MQCONN 呼び出しに渡されるキュー・マネージャーの名前と同じです。 xa_open 呼び出しに渡される名前がブランクの場合、デフォルトのキュー・マネージャーが使用されます。

  あるいは、xa_info 構造体に *TPM* パラメーターおよび *AXLIB* パラメーターの値を含める方法もあります。 *TPM* パラメーターは、使用されるトランザクション・マネージャーを指定します。 有効な値は、CICS、TUXEDO、および ENCINA です。 *AXLIB* パラメーターは、トランザクション・マネージャーの ax_reg 関数および ax_unreg 関数を含むライブラリーの名前を指定します。 これらのパラメーターの詳細については、拡張トランザクション・クライアントの構成を参照してください。 xa_info 構造体にこれらのパラメーターのいずれかが含まれる場合、デフォルトのキュー・マネージャーが使用されていない限り、キュー・マネージャー名が *QMNAME* パラメーターで指定されます。
- 外部同期点調整プログラムのインスタンスによって調整されるトランザクションには、一度に 1 つのキュー・マネージャーのみが参加できます。 これにより同期点調整プログラムは、キュー・マネージャーと有効な接続関係を持つことができ、サポートされる接続は一度につき 1 つという規則が適用されます。
- 外部同期点調整プログラムへの呼び出しを含むあらゆるアプリケーションが接続できるキュー・マネージャーは、この外部同期点調整プログラムが管理するトランザクションに参加しているキュー・マネージャーのみです (このようなアプリケーションは既にキュー・マネージャーと有効な接続関係にあるためです)。 ただし、このようなアプリケーションは、MQCONN 呼び出しを実行して接続ハンドルを獲得し、終了前に MQDISC 呼び出しを実行する必要があります。

- リソースの更新が外部同期点調整プログラムによって調整されるキュー・マネージャーは、外部同期点調整プログラムより前に始動する必要があります。 同様に、同期点調整プログラムはキュー・マネージャーより前に終了しなければなりません。
- 外部同期点調整プログラムが異常終了した場合、この同期点調整プログラムを**再始動する前** にキュー・マネージャーを停止して再始動し、異常終了時にコミットされていなかったすべてのメッセージング操作が適切に解決されるようにします。

CICS の使用

CICS は、TXSeries のエレメントの 1 つです。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

XA 準拠 (かつ 2 フェーズ・コミット・プロセスを使用) の TXSeries のバージョンは、 IBM WebSphere MQ の詳細なシステム要件 で定義されています。

WebSphere MQ は、他のトランザクション・マネージャーもサポートします。 サポートされているソフトウェアの現行のリストについては、IBM WebSphere MQ の詳細なシステム要件を参照してください。

2 フェーズ・コミット・プロセスの要件

WebSphere MQ で CICS 2 フェーズ・コミット・プロセスを使用する場合の 2 フェーズ・コミット・プロセスの要件。 これらの要件は、z/OS には適用されません。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

次の要件に注意してください。

- WebSphere MQ および CICS は同一の物理マシン上になければなりません。
- WebSphere MQ は、 WebSphere MQ MQI クライアント上の CICS をサポートしません。
- XAD リソース定義スタンザに指定されている名前を使用して、キュー・マネージャーを開始する必要があります。 **前** 「CICS」を開始しようとします。 CICS 領域に WebSphere MQ 用の XAD リソース定義スタンザを追加してある場合には、このようにキュー・マネージャーを始動しないと、CICS を始動できません。
- 単一の CICS 領域から一度にアクセスできる WebSphere MQ キュー・マネージャーは 1 つのみです。
- CICS トランザクションは、`MQCONN` 要求を実行しないと WebSphere MQ リソースにアクセスできません。 `MQCONN` 呼び出しで名前が指定される WebSphere MQ キュー・マネージャーは、CICS 領域の XAD リソース定義スタンザの XAOpen 項目に指定されているものでなければなりません。 この項目がブランクである場合、`MQCONN` 要求ではデフォルトのキュー・マネージャーを指定しなければなりません。
- WebSphere MQ リソースにアクセスする CICS トランザクションは、 CICS に戻る前に、トランザクションから `MQDISC` 呼び出しを発行する必要があります。 これを行わない場合、キューがオープンの状態で、CICS アプリケーション・サーバーが接続されたままになることがあります。 さらに、タスク終了出口 (76 ページの『サンプル・タスク終了出口』を参照) をインストールしないと、CICS アプリケーション・サーバーが後で (おそらく次のトランザクション中に) 異常終了する可能性があります。
- CICS ユーザー ID (cics) が mqm グループのメンバーであり、CICS コードが WebSphere MQ を呼び出す権限を持っていることを確認する必要があります。

  CICS 環境で実行されるトランザクションの場合、キュー・マネージャーは、次のような許可方法およびコンテキストの決定方法を採用しています。

  - キュー・マネージャーは、CICS がトランザクションを実行するユーザー ID を照会します。 このユーザー ID はオブジェクト権限マネージャーにより検査されたもので、コンテキスト情報に使用されます。
  - メッセージ・コンテキストでは、アプリケーション・タイプは MQAT_CICS です。
  - コンテキスト内のアプリケーション名は、CICS トランザクション名からコピーされます。

一般 *XA* サポート
General XA は IBM i でサポートされません。 CICS と WebSphere MQ (UNIX and Linux システム用) のリンクを可能にする XA スイッチ・ロード・モジュールが提供されます。 また、用意されているサンプル・ソース・コード・ファイルを使用して、他のトランザクション・メッセージ用の XA スイッチを作成できます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

用意されているスイッチ・ロード・モジュールの名前は以下のとおりです。

表 *11. CICS* アプリケーションの必須コード*: XA* 初期化ルーチン

C (ソース)	**C (exec) - 以下のいずれかを追加してください。** **XAD.Stanza**
amqzscix.c	amqzsc - TXSeries (版) AIX バージョン 5.1、 amqzsc - TXSeries for HP-UX バージョン 5.1 amqzsc - TXSeries for Sun Solaris バージョン 5.1
amqzscin.c	mqmc4swi - TXSeries (版) Windows バージョン 5.1

TXSeries for Multiplatforms で使用するためのライブラリーの作成
TXSeries for Multiplatforms で使用するためのライブラリーを作成するときには、以下の情報を使用してください。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

事前作成スイッチ・ロード・ファイル は、XA プロトコルを使用する 2 フェーズ・コミット・トランザクションを必要とする CICS プログラムで使用できる共用ライブラリー (Windows システムでは *DLL* と呼ばれます) です。 この事前作成ライブラリーの名前は、CICS アプリケーションの必須コード: XA 初期化ルーチンの表に記載されています。 また、サンプル・ソース・コードは、以下のディレクトリーにあります。

表 *12. Windows*、*UNIX and Linux* オペレーティング・システムでのインストール・ディレクトリー

プラットフォーム	**ディレクトリー**	**ソース・ファイル**
UNIX and Linux	*MQ_INSTALLATION_PATH*/samp/	amqzscix.c
Windows	*MQ_INSTALLATION_PATH*¥Tools¥c¥Samples	amqzscin.c

ここで、*MQ_INSTALLATION_PATH* は、 IBM WebSphere MQ をインストールしたディレクトリーです。

サンプル・ソースからスイッチ・ロード・ファイルを作成するには、ご使用のオペレーティング・システムに該当する指示に従ってください。

AIX
以下のコマンドを発行します。

```
export MQM_HOME=/usr/mqm
echo "amqzscix" > tmp.exp
xlc_r $MQM_HOME/samp/amqzscix.c -I/usr/lpp/cics/include -I$MQM_HOME/inc -e amqzscix -bE:tmp.exp -bM:SRE
-o amqzsc /usr/lpp/cics/lib/regxa_swxa.o -L$MQM_HOME/lib -L/usr/lpp/cics/lib -lcicsrt -lEncina
-lEncServer -lpthreads -lsarpc -lmqmcics_r -lmqmxa_r -lmqzi_r -lmqmcs_r
rm tmp.exp
```

Solaris
以下のコマンドを発行します。

```
/opt/SUNWspro/bin/cc -s -l/opt/encina/include amqzscix.c -G -o amqzscix  -e
    CICS_XA_Init -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib
    -L/opt/dcelocal/lib /opt/cics/lib/reqxa_swxa.o
    -lmqmcics -lmqmxa -lmqzi -lmqmcs -lmqmzse -lcicsrt -lEncina -lEncSfs -ldce
```

HP-UX
以下のコマンドを発行します。

```
cc -c -s -I/opt/encina/include MQ_INSTALLATION_PATH/samp/amqzscix.c -Aa +z -o  amqzscix.o ld -b
    -o amqzscix amqzscix.o /opt/cics/lib/regxa_swxa.o +e CICS_XA_Init \
    -LMQ_INSTALLATION_PATH/lib -L/opt/encina/lib -L/opt/cics/lib
    -lmqmxa_r -lmqzi_r -lmqmcs_r -lmqmzse -ldbm -lc -lm
```

Linux プラットフォーム
以下のコマンドを発行します。

```
gcc -m32 -shared -fPIC -o amqzscix amqzscix.c
    \-IMQ_INSTALLATION_PATH/inc -I CICS_INSTALLATION_PATH/include
    \-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
    \-Wl,-rpath=/usr/lib -Wl,-rpath-link,/usr/lib -Wl,--no-undefined
    -Wl,--allow-shlib-undefined \-L CICS_LIB_PATH/regxa_swxa.o \-lpthread -ldl -lc
    -shared -lmqzi_r -lmqmxa_r -lmqmcics_r -ldl -lc
```

Windows
次のステップを行います。

1. cl コマンドを使用し、少なくとも以下の変数でコンパイルして、amqzscin.obj をビルドします。

   ```
   cl.exe -c -IEncinaPath\include -IMQ_INSTALLATION_PATH\include -Gz -LD amqzscin.c
   ```

2. 以下の行を含む、mqmc1415.def という名前のモジュール定義ファイルを作成します。

   ```
   LIBRARY MQMC4SWI
   EXPORTS
   CICS_XA_Init
   ```

3. **lib** コマンドを使用し、少なくとも以下のオプションを使用してエクスポート・ファイルとインポート・ライブラリーを作成します。

   ```
   lib -def:mqmc4swi.def -out:mqmc4swi.lib
   ```

   lib コマンドが正常に実行されると、mqmc4swi.exp ファイルもまた作成されます。

4. link コマンドを使用し、少なくとも以下のオプションを使用して mqmc4swi.dll を作成します。

   ```
   link.exe -dll -nod -out:mqmc4swi.dll
       amqzscin.obj CicsPath\lib\regxa_swxa.obj
       mqmc4swi.exp mqmcics4.lib
      CicsPath\lib\libcicsrt.lib
      DcePath\lib\libdce.lib DcePath\lib\pthreads.lib
      EncinaPath\lib\libEncina.lib
      EncinaPath\lib\libEncServer.lib
      msvcrt.lib kernel32.lib
   ```

IBM WebSphere MQ XA のサポートおよび *Tuxedo*
IBM WebSphere MQ Windows では、UNIX and Linux システムは、xa_start で Tuxedo によって調整される XA アプリケーションを無期限にブロックできます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

これが発生するのは、単一のグローバル・トランザクションの中で Tuxedo によって調整される複数のプロセスが、同じトランザクション・ブランチ ID (XID) を使用して IBM WebSphere MQ にアクセスしようとする場合のみです。 グローバル・トランザクションの各プロセスに対して、IBM WebSphere MQ で使用するための XID としてそれぞれ異なるものを Tuxedo が割り当てる場合は、この問題が発生することはありません。

この問題を回避するには、単一のグローバル・トランザクション ID (gtrid) のもとで IBM WebSphere MQ にアクセスする Tuxedo 内の各アプリケーションを、独自の Tuxedo サーバー・グループ内で構成するようにしてください。 同じサーバー・グループに属するプロセスは、リソース・マネージャーにアクセスする際に単一の gtrid の代わりに同じ XID を使用するため、IBM WebSphere MQ において xa_start でブロッキングの影響を受けることになります。 異なるサーバー・グループに属するプロセスは、別々の XID を使用してリソース・マネージャーにアクセスするので、IBM WebSphere MQ でそのトランザクション処理を直列化する必要はありません。

CICS 2 フェーズ・コミット・プロセスの使用可能化
CICS で 2 フェーズ・コミット・プロセスを使用して MQI 呼び出しが関与するトランザクションを調整できるようにするには、CICS XAD リソース定義スタンザ・エントリーを CICS 領域に追加します。 このトピックは、 z/OS には適用されないことに注意してください。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

WebSphere MQ for Windows の XAD スタンザ・エントリーを追加する例を示します。<Drive> は、WebSphere MQ がインストールされるドライブ (例えば、D:) です。

```
cicsadd -cxad -r<cics_region> \
     ResourceDescription="MQM XA Product Description" \
     SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
     XAOpen=<queue_manager_name>
```

拡張トランザクション・クライアントでは、スイッチ・ロード・ファイル mqcc4swi.dll を使用します。

WebSphere MQ for UNIX and Linux システムの XAD スタンザ・エントリーを追加する例です。
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされる上位ディレクトリーを表します。

```
cicsadd -cxad -r<cics_region> \
      ResourceDescription="MQM XA Product Description" \
      SwitchLoadFile="MQ_INSTALLATION_PATH/lib/amqzsc" \
      XAOpen=<queue_manager_name>
```

拡張トランザクション・クライアントでは、スイッチ・ロード・ファイル amqczsc を使用します。

cicsadd コマンドの使用法については、「*CICS* 管理リファレンス」またはご使用のプラットフォーム用の「*CICS* 管理ガイド」を参照してください。

WebSphere MQ の呼び出しを CICS トランザクションに組み込むことができます。この場合、WebSphere MQ リソースのコミットおよびロールバックは CICS の指示により実行されます。 このサポートはクライアント・アプリケーションでは使用できません。

WebSphere MQ リソースにアクセスするには、 CICS トランザクションから MQCONN を発行する **必要があります** 。その後、出口で対応する MQDISC を発行する必要があります。

CICS ユーザー出口の使用可能化
CICS ユーザー出口ポイント (通常、ユーザー出口 と呼ばれる) とは、CICS モジュール内で CICS が制御をユーザー作成プログラム (ユーザー出口プログラム) に渡し、ユーザー出口プログラムが処理を完了すると CICS に制御が戻るポイントです。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

CICS ユーザー出口を使用する前に、ご使用のプラットフォーム用の「*CICS* 管理ガイド」を参照してください。

サンプル・タスク終了出口
WebSphere MQ では、CICS タスク終了出口のサンプル・ソース・コードが提供されています。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

サンプル・ソース・コードは、以下のディレクトリーにあります。

表 *13. CICS* タスク終了出口		
プラットフォーム	**ディレクトリー**	**ソース・ファイル**
UNIX and Linux システム	*MQ_INSTALLATION_PATH*/サンプ	amqzscgx.c
Windows	*MQ_INSTALLATION_PATH*¥Tools¥c¥Samples	amqzscgn.c

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

各ソース・ファイルの先頭付近のコメント中に、サンプル・タスク終了出口の作成に関する説明が記載されています。

タスクの通常終了時および異常終了時 (同期点がとられた後) に、CICS からこの出口が呼び出されます。 出口プログラムでは、リカバリー可能作業は実行できません。

これらの関数を使用するのは、WebSphere MQ においてと、CICS バージョンが XA インターフェースをサポートしている CICS コンテキストにおいてのみです。 CICS は、これらのライブラリーを "プログラム" または "ユーザー出口"と呼びます。

CICS にはいくつかのユーザー出口があり、amqzscgx (使用されている場合) は CICS で "タスク終了ユーザー出口 (UE014015)"(出口番号 15) として定義され、使用可能になっています。

タスク終了出口が CICS から呼び出された時点で、そのタスクの終了状態が CICS から WebSphere MQ に既に通知されており、WebSphere MQ が適切な操作 (コミットまたはロールバック) を実行しています。 出口が実行する内容は、クリーンアップのための MQDISC の発行のみです。

タスク終了出口を使用するために CICS システムをインストールして構成する目的の 1 つは、障害のあるアプリケーション・コードの何らかの影響を受けないようにシステムを保護することにあります。 例えば、CICS トランザクションが最初に MQDISC を呼び出さずに異常終了した場合に、タスク終了出口がインストールされていないと、続けて (約 10 秒以内に) CICS 領域のリカバリー不能な障害が発生することがあります。 その原因は、cicsas プロセスで稼働する WebSphere MQ の正常性スレッドが、ポストされずにしかもクリーンアップおよび戻りのための時間も与えられなくなることにあります。 その徴候としては、FFST レポートを /var/mqm/errors (Windows ではこれに相当する場所) に書き込んだうえで cicsas プロセスが即時に終了することがあります。

Microsoft Transaction Server (COM+) の使用
COM + (Microsoft Transaction Server) は、ユーザーが標準的な中間層サーバーでビジネス・ロジック・アプリケーションを実行できるように設計されています。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

重要な情報については、Windows 上のプライマリー・インストールでのみ使用できる機能 を参照してください。

COM+ は作業を アクティビティーに分割します。通常、口座 *A* から口座 *B* への資金の振替などの、ビジネス・ロジックに基づく短い独立したチャンクです。 COM+ はオブジェクト方向、特に COM に大きく依存します。緩やかに COM + のアクティビティーは COM (ビジネス) オブジェクトによって表されます。

COM+ は、オペレーティング・システムの統合パーツの 1 つです。 Windows 2000 および Windows XP で COM+ を使用するには、Hotfix Q313582 (COM+ Rollup Package 19.1 ともいいます) が必要です。

COM+ は以下 3 つのサービスをビジネス・オブジェクト管理者に提供し、ビジネス・オブジェクト・プログラマーのかなりの負荷を軽減します。

- トランザクション管理
- 機密保護
- リソース・プール

一般に COM+ は、COM+ 内に保持されているオブジェクトに対する COM クライアントであるフロントエンド・コードと、データベースなどのバックエンド・サービスと共に使用され、WebSphere MQ によって COM+ ビジネス・オブジェクトとバックエンドの間がブリッジングされます。

フロントエンド・コードは、スタンドアロン・プログラムか、Microsoft Internet Information Server (IIS) をホストとするアクティブ・サーバー・ページ (ASP) にすることができます。 フロントエンド・コードは、COM の接続を通して、COM+ およびビジネス・オブジェクトと同じコンピューターに置くことができます。また、フロントエンド・コードは、DCOM の接続を通して、異なるコンピューターに置くことができます。状況別に異なるクライアントを使用して、同じ COM+ ビジネス・オブジェクトにアクセスすることもできます。

バックエンド・コードは、COM+ およびビジネス・オブジェクトと同じコンピューターに置くか、WebSphere MQ でサポートされている任意のプロトコルの接続を通して異なるコンピューターに置くことができます。

グローバル作業単位の有効期限切れ

事前に構成された非アクティブ間隔が経過した後に、グローバル作業単位の有効期限が切れるようキュー・マネージャーを構成することができます。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

この動作を有効にするには、以下の環境変数を設定します。

- *AMQ_TRANSACTION_EXPIRY_RESCAN*= < 再スキャン間隔 (ミリ秒)>
- *AMQ_XA_TRANSACTION_EXPIRY*= < タイムアウト間隔 (ミリ秒)>

重要 : 環境変数は、XA 仕様の表 6-4 の *Idle* 状態のトランザクションにのみ影響を与えます。 つまり、アプリケーション・スレッドに関連付けられていないものの、外部トランザクション・マネージャー・ソフトウェアによって **xa_prepare** 関数呼び出しがまだ呼び出されていないトランザクションです。

外部トランザクション・マネージャーは、準備、コミット、またはロールバックされるトランザクションのみをログに記録します。 何らかの理由で外部トランザクション・マネージャーがダウンして復帰した場合、準備、コミット、およびロールバックされたトランザクションは完了されますが、まだ準備されていないすべてのアクティブ・トランザクションは孤立します。 これを防ぐには *AMQ_XA_TRANSACTION_EXPIRY* を設定します。その際、アプリケーションが MQI トランザクション API 呼び出しを行い、他のリソース・マネージャーでのトランザクション作業を実行してトランザクションを完了するまでの予想される時間間隔を考慮に入れます。

AMQ_XA_TRANSACTION_EXPIRY 満了後の適切な時点で確実にクリーンアップするには、*AMQ_TRANSACTION_EXPIRY_RESCAN* の値を *AMQ_XA_TRANSACTION_EXPIRY* 間隔の値よりも小さい値に設定してください。*AMQ_XA_TRANSACTION_EXPIRY* 間隔の中で再スキャンが複数回にわたって発生するのが理想的です。

回復単位後処理

WebSphere MQ for z/OS は、リカバリー単位処理を提供します。 この機能によって、2 フェーズ・コミット・トランザクションの 2 番目のフェーズを駆動可能にするかどうかを構成できます。 例えば、リカバリー中に、同じキュー共用グループ (QSG) 内の別のキュー・マネージャーに接続される場合などです。

注 : このトピックは、 IBM MQ Version 8.0 以降のバージョンでも使用できます。 ただし、 "「バージョンの変更」" リスト・ボックスを使用して新しいバージョンに切り替えることはできません。 新しいバージョンのトピックに移動するには、ブラウザーの URL ボックスでバージョン番号を編集します。

リカバリー単位処理は、WebSphere MQ for z/OS V7.0.1 以降でサポートされます。

回復単位後処理

リカバリー単位処理は、アプリケーションの接続およびそれ以降に開始されるすべてのトランザクションに関係するものです。 可能なリカバリー単位処理には、次の 2 つがあります。

- GROUP リカバリー単位処理は、トランザクション・アプリケーションが論理的にキュー共用グループに接続されていて、特定のキュー・マネージャーに対するアフィニティーがないことを識別します。 QSG 内のいずれかのキュー・マネージャーに接続されている場合に、開始される 2 フェーズ・コミット・トランザクションのうち、コミット・プロセスのフェーズ 1 を完了している (つまり、未確定である) ものを照会して解決できます。 リカバリー・シナリオでは、これは、トランザクション調整プログラムが、同じキュー・マネージャーに再接続する必要がない (使用不可の可能性がある) ことを意味します。
- QMGR リカバリー単位処理は、アプリケーションに接続先のキュー・マネージャーに対する直接のアフィニティーがあり、このアプリケーションが開始するすべてのトランザクションにもこの処理があることを識別します。

  リカバリー・シナリオでは、そのキュー・マネージャーがキュー共用グループに属するかどうかに関係なく、トランザクション調整プログラムは同じキュー・マネージャーに再接続して、未確定トランザクションをすべて照会して解決する必要があります。

使用するプログラミング言語の決定

この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

IBM WebSphere MQ では、以下のプロシージャー型のプログラミング言語をサポートしています。

- C
- Visual Basic (Windows システムのみ)
- COBOL

これらの言語は、メッセージ・キュー・インターフェース (MQI) を使用して、メッセージ・キューイング・サービスにアクセスします。 これらの言語のサポートの詳細については、79 ページの『WebSphere MQ とプロシージャー型言語の併用』を参照してください。

IBM WebSphere MQ は、以下をサポートします。

- .NET
- ActiveX
- C++
- JAVA
- JMS

これらの言語は、IBM WebSphere MQ オブジェクト・モデルを使用します。これは、WebSphere MQ の呼び出しおよび構造体と同じ機能を提供しますが、オブジェクト指向の環境により合ったプログラミングの方法であるクラスを備えています。 IBM WebSphere MQ オブジェクト・モデルを使用する一部の言語は、メッセージ・キュー・インターフェース (MQI) で使用できない追加の機能を提供します。 これらの言語のサポートの詳細については、79 ページの『WebSphere MQ によるオブジェクト指向プログラミング』を参照してください。

WebSphere MQ とプロシージャー型言語の併用

選択した言語でアプリケーションを作成する方法については、以下のリンクを参照してください。

- 82 ページの『C によるコーディング』
- 87 ページの『Visual Basic によるコーディング』
- 85 ページの『COBOL によるコーディング』

プロシージャー型言語の呼び出しインターフェースの概要については、呼び出しの記述を参照してください。 このトピックには、MQI 呼び出しのリストが含まれます。各呼び出しについて、これらの各言語での呼び出しのコーディング方法を示します。

WebSphere MQ では、アプリケーションの作成に役立つ、データ定義ファイルを提供しています。 詳細は、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

プログラムをコーディングする言語を選択できる場合は、そのプログラムが処理するメッセージの最大長を考慮してください。 プログラムが、既知の最大長をもつメッセージのみを処理する場合は、サポートされるどのプログラム言語ででもコーディングすることができます。 しかし、プログラムが処理するメッセージの最大長が分からない場合は、作成するアプリケーションが CICS、IMS、またはバッチのどれであるかによって、選択する言語が次のように決められます。

IMS およびバッチ
任意の容量のストレージを取得したり解放したりするには、これらの機能を提供する C、PL/I およびアセンブラー言語でプログラムをコーディングしてください。 また、COBOL を用いてプログラムをコーディングすることもできますが、ストレージを取得したり解放したりするには、アセンブラー言語、PL/I、または C のサブルーチンを使用してください。

CICS
CICS でサポートされる任意の言語でプログラムをコーディングしてください。 EXEC CICS インターフェースは、必要に応じてストレージを管理するための呼び出しを提供しています。

WebSphere MQ によるオブジェクト指向プログラミング

IBM WebSphere MQ Object Model を使用する言語およびプログラミング・フレームワークの中には、メッセージ・キュー・インターフェース (MQI) では利用できない追加の機能を提供しているものがあります。 IBM WebSphere MQ オブジェクト・モデルによって提供されるクラス、メソッド、およびプロパティーの詳細については、87 ページの『IBM WebSphere MQ オブジェクト・モデル』を参照してください。

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

C++
IBM WebSphere MQ では、WebSphere MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。 WebSphere MQ オブジェクト・モデルを C++ で使用するプログラムのコーディングについては、C++ の使用 を参照してください。 Message Service Clients for C/C++ and .NET には、Java Message Service (JMS) と同じインターフェース・セットを持つ XMS というアプリケーション・プログラミング・インターフェース (API) が用意されています。 API。

JAVA
Java での WebSphere MQ オブジェクト・モデルを使用したプログラムのコーディングについては、Java の使用 を参照してください。 IBM WebSphere MQ classes for Java と IBM WebSphere MQ クラスの違いについて、どちらを使用するか決める際の参考のために、89 ページの『IBM WebSphere MQ classes for Java または IBM WebSphere MQ classes for JMS を使用する必要がありますか?』を参照してください

JMS
Websphere MQ は、Java Message Service (JMS) 仕様をインプリメントするクラスも提供します。 Websphere MQ classes for JMS の詳細については、Java の使用を参照してください。 IBM WebSphere MQ classes for Java と IBM WebSphere MQ クラスの違いについて、どちらを使用するか決める際の参

考のために、89 ページの『IBM WebSphere MQ classes for Java または IBM WebSphere MQ classes for JMS を使用する必要がありますか?』を参照してください

C/C++ および .NET 用の Message Service Client は、 Java Message Service (JMS) API と同じインターフェース・セットを持つ、XMS という名前のアプリケーション・プログラミング (API) を提供します。

ActiveX

WebSphere MQ ActiveX は通常、MQAX と呼ばれます。 MQAX は、WebSphere MQ for Windows の一部として含まれています。 ActiveX のサポートは、WebSphere MQ バージョン 6.0 レベルで固定されています。 バージョン 6.0 より後の WebSphere MQ で導入された機能を 活用するには、代わりに .NET を使用することを検討してください。 ActiveX で WebSphere MQ オブジェクト・モデルを使用するプログラムのコーディングについては、 コンポーネント・オブジェクト・モデル・インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用 を参照してください。

関連概念

技術概要

7 ページの『アプリケーションの開発』

IBM WebSphere MQ には、ビジネス・プロセスをサポートするために必要なメッセージを送受信するアプリケーションを開発するためのいくつかの手段が用意されています。 キュー・マネージャーや関連リソースを管理するためのアプリケーションを開発することもできます。

8 ページの『アプリケーション開発の概念』

選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

関連資料

アプリケーション開発のリファレンス

IBM WebSphere MQ データ定義ファイル

IBM WebSphere MQ では、アプリケーションの作成に役立つ、データ定義ファイルを提供しています。

データ定義ファイルは、以下の名前でも呼ばれます。

言語	データ定義
C	組み込みファイルまたはヘッダー・ファイル
Visual Basic	モジュール・ファイル (32 ビット版のみ)
COBOL	コピー・ファイル
アセンブラー	マクロ
PL/I	組み込みファイル

チャネル出口の作成に役立つデータ定義ファイルについては、WebSphere MQ COPY ファイル、ヘッダー・ファイル、インクルード・ファイル、およびモジュール・ファイルで説明されています。

インストール可能なサービス出口の作成に役立つデータ定義ファイルについては、376 ページの『ユーザー出口、API 出口、および WebSphere MQ インストール可能サービス』で説明されています。

C++ でサポートされるデータ定義ファイルについては、C++ の使用を参照してください。

データ定義ファイルの名前には、接頭部 CMQ と、プログラム言語によって異なる接尾部が付いています。

接尾部	言語
a	アセンブラー言語
b	Visual Basic
c	C
l	COBOL (初期設定値なし)

接尾部	言語
p	PL/I
v	COBOL (デフォルト値設定あり)

インストール・ライブラリー

thlqual は、z/OS のインストール・ライブラリーの高水準修飾子を表します。

このトピックでは、以下の見出しのもとで、WebSphere MQ データ定義ファイルについて説明します。

- 81 ページの『C 言語組み込みファイル』
- 81 ページの『Visual Basic モジュール・ファイル』
- 81 ページの『COBOL コピー・ファイル』

C 言語組み込みファイル

WebSphere MQ C の組み込みファイルは、C ヘッダー・ファイルにリストされています。 これらのファイルは次のディレクトリーまたはライブラリーにインストールされます。

プラットフォーム	インストール・ディレクトリーまたはライブラリー
UNIX プラットフォーム	*MQ_INSTALLATION_PATH*/inc/
Windows システム	*MQ_INSTALLATION_PATH*\Tools\c\include

ここで、*MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

注 : UNIX プラットフォームでは、組み込みファイルはシンボリックに `/usr/include` にリンクされます。

ディレクトリーの構造について詳しくは、 ファイル・システム・サポートの計画 を参照してください。

Visual Basic モジュール・ファイル

WebSphere MQ for Windows では、4 つの Visual Basic モジュール・ファイルが提供されます。

これは Visual Basic モジュール・ファイルにリストされており、次の場所にインストールされます。

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

COBOL コピー・ファイル

COBOL の場合、WebSphere MQ は、名前付きの定数を含む個別のコピー・ファイルと、各構造体ごとに 2 つのコピー・ファイルを提供します。

各構造体別の 2 つのコピー・ファイルがありますが、これは初期値がないものと、初期値があるものがそれぞれ提供されているからです。

- COBOL プログラムの WORKING-STORAGE SECTION では、構造体フィールドをデフォルト値に初期化するファイルを使用します。 これらの構造体は、文字「V」(値) を接尾部にもつコピー・ファイル名に定義されます。
- COBOL プログラムの LINKAGE SECTION では、初期値なしの構造体を使用します。 これらの構造体は、文字「L」(連係) が後ろに付いた名前をもつコピー・ファイルで定義されます。

WebSphere MQ COBOL コピー・ファイルは、COBOL コピー・ファイルにリストされています。 次のディレクトリーにインストールされます。

プラットフォーム	インストール・ディレクトリーまたはライブラリー
その他の UNIX プラットフォーム	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*¥Tools¥cobol¥copybook (Micro Focus COBOL 用) *MQ_INSTALLATION_PATH*¥Tools¥cobol¥copybook¥VAcobol (IBM VisualAge® COBOL 用)

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

必要なファイルのみをプログラムに組み込みます。 これを、レベル 01 の宣言後に 1 つまたは複数の COPY ステートメントによって行います。 つまり、必要に応じてプログラムに複数バージョンの構造体を組み込むことができます。 ただし、CMQV はファイル・サイズが大きいので注意してください。

次に、CMQMDV コピー・ファイルを組み込むための COBOL コードの例を示します。

```
01 MQM-MESSAGE-DESCRIPTOR.
   COPY CMQMDV.
```

各構造体の宣言は、レベル 01 の項目で始まります。つまり、構造体宣言の残りの部分にコピーする COPY ステートメントの前で、レベル 01 の宣言をコーディングすることによって、いくつかの構造体のインスタンスを宣言できることを意味します。 適切なインスタンスを参照するには、IN キーワードを使用してください。

次に、CMQMDV の 2 つのインスタンスを組み込むための COBOL コードの例を示します。

```
* Declare two instances of MQMD
 01  MY-CMQMD.
     COPY CMQMDV.
 01  MY-OTHER-CMQMD.
     COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-CMQMD
     MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-CMQMD.
```

構造体を 4 バイト境界に位置合わせしてください。 COPY ステートメントを使用してレベル 01 項目以外の項目の後ろに構造体を組み込む場合には、その構造体がレベル 01 項目の開始から 4 バイトの倍数になっていることを確認してください。 これを無視すると、アプリケーションのパフォーマンスが低下する可能性があります。

構造体については、MQI で使用されるデータ・タイプを参照してください。 構造体内のフィールドの記述は、接頭部なしのフィールドの名前を示します。 COBOL プログラムでは、COBOL 宣言で示すように、フィールド名の接頭部に構造体の名前とそれに続くハイフンを付けてください。 構造体コピー・ファイル内のフィールドは、上記の方法で接頭部が付けられます。

構造体コピー・ファイル内の宣言でのフィールド名は、大文字です。 大文字と小文字を混在させたり、大文字の代わりに小文字を使用しても構いません。 例えば、MQGMO 構造体のフィールド *StrucId* は、COBOL 宣言およびコピー・ファイル内の MQGMO-STRUCID として示されます。

接尾部 V の構造体はすべてのフィールドに対して初期値で宣言されます。したがって、必要とされる値が初期値と異なる場合には、これらのフィールドのみを設定するだけで済みます。

C によるコーディング

WebSphere MQ プログラムを C 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

- 83 ページの『MQI 呼び出しのパラメーター』
- 83 ページの『未定義データ・タイプのパラメーター』
- 83 ページの『データ・タイプ』

- 83 ページの『2 進ストリングの取り扱い』
- 84 ページの『文字ストリングの取り扱い』
- 84 ページの『構造体の初期値』
- 84 ページの『動的構造体の初期値』
- 85 ページの『C++ からの使用』

MQI 呼び出しのパラメーター

入力のみ でタイプが MQHCONN、MQHOBJ、MQHMSG、MQLONG のいずれかであるパラメーターは、値を使って渡されます。他のすべてのパラメーターでは、パラメーターのアドレス が値を使って渡されます。

アドレスを使って渡されるパラメーターのすべてを、機能を呼び出すたびに指定しなければならないわけではありません。 特定のパラメーターが必要でない場合は、パラメーター・データのアドレスの代わりに、機能呼び出し時にパラメーターとしてヌル・ポインターを指定できます。 これが可能なパラメーターは、呼び出し記述子で識別されます。

関数の値として戻るパラメーターはありません。C の用語では、これはすべての関数が void を戻す、と言います。

関数の属性は、MQENTRY マクロ変数によって定義されます。このマクロ変数の値は、環境に応じて異なります。

未定義データ・タイプのパラメーター

MQGET 関数、MQPUT 関数、および MQPUT1 関数にはそれぞれ、未定義データ・タイプの *Buffer* パラメーターがあります。 このパラメーターは、アプリケーションのメッセージ・データを送受信するために使用されます。

この種類のパラメーターは、C の例で MQBYTE の配列として示されています。 この方法でパラメーターを宣言することはできますが、通常は、これらのメッセージ内のデータのレイアウトを記述する構造体として宣言する方が便利です。 関数仮パラメーターは、void を指し示すポインターとして宣言されるので、どのようなデータのアドレスでも関数呼び出し時にパラメーターとして指定できます。

データ・タイプ

すべてのデータ型は、typedef ステートメントによって定義されます。

各データ型には、対応するポインター・データ型も定義されます。 ポインター・データ・タイプの名前は、基本データ・タイプ、またはポインターを指示するためにその接頭部に P の付いた構造体データ・タイプの名前です。 ポインターの属性は、MQPOINTER マクロ変数によって定義されます。このマクロ変数の値は、環境に応じて異なります。 以下のコードはポインター・データ型を宣言する方法を示しています。

```
#define MQPOINTER                   /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

2 進ストリングの取り扱い

2 進データのストリングは、いずれかの MQBYTEn データ・タイプとして宣言されます。

このタイプのフィールドのコピー、比較、または設定を行うときには、C 関数の memcpy、memcmp、または memset を使用してください。

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
```

```
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

ストリング関数の strcpy、strcmp、strncpy、または strncmp は、MQBYTE24 として宣言されたデータに対しては正しく機能しないので、使用しないでください。

文字ストリングの取り扱い

キュー・マネージャーは、文字データをアプリケーションに戻すときに、文字データがフィールドの定義された長さになるように必ずブランク文字を埋め込みます。 キュー・マネージャーはヌル文字で終了するストリングを戻しませんが、入力値として使用することはできます。 したがって、そのようなストリングのコピー、比較、または連結を行うときは、ストリング関数の strncpy、strncmp、または strncat を用いてください。

ストリングがヌル文字で終了することを要求するストリング関数 (strcpy、strcmp、および strcat) を使用してはなりません。 また、ストリングの長さを判別するために関数 strlen を使用してはなりません。代わりに、関数 sizeof を使用してフィールドの長さを判別してください。

構造体の初期値

組み込みファイル <cmqc.h> は、構造体のインスタンスを宣言する際に、その構造体の初期値の設定に使用できる各種のマクロ変数を定義します。 これらのマクロ変数では、MQxxx_DEFAULT の形式の名前を使用します。 この MQxxx は構造体の名前を表します。 次のように使用します。

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

文字フィールドには、MQI により有効な特定値が定義されるものもあります (例えば、MQMD 内の *StrucId* フィールドや *Format* フィールドの場合)。 それぞれの有効値に対して、次の 2 つのマクロ変数が提供されます。

- 定義されたフィールドの長さと完全に一致する長さのストリング (暗黙指定のヌルを除く) として有効値を定義するマクロ変数。 例えば、シンボル ¬ はブランク文字を表します。

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  memcpy 関数および memcmp 関数でこの形式を使用します。
- 有効値を char (文字) の配列として定義するマクロ変数。このマクロ変数の名前は、_ARRAY を接尾部にもつストリングです。 以下に例を示します。

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  構造体のインスタンスが MQMD_DEFAULT マクロ変数で提供される値と異なる値で宣言された場合には、この書式を使用して、当該フィールドを初期化します。

動的構造体の初期値

構造体のインスタンスの変数番号が必要な場合、インスタンスは通常 calloc または malloc 関数を使用して動的に取得した主ストレージに作成されます。

そのような構造体におけるフィールドを初期化するには、次の技法をお勧めします。

1. 構造体を初期化するために、適切な MQxxx_DEFAULT マクロ変数を使用して、構造体のインスタンスを宣言する。 このインスタンスが他のインスタンスのモデル になります。

```
MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                  /* declare model instance */
```

   static または auto キーワードを宣言にコーディングして、モデルのインスタンスの存続期間を必要に応じて静的、または動的にします。
2. calloc または malloc 関数を使用して、構造体の動的インスタンスのストレージを取得する。

```
PMQMD InstancePtr;
InstancePtr = malloc(sizeof(MQMD));
                                  /* get storage for dynamic instance */
```

3. memcpy 関数を使用して、モデルのインスタンスを動的インスタンスにコピーする。

```
memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                  /* initialize dynamic instance */
```

C++ からの使用

C++ プログラム言語の場合、ヘッダー・ファイルには、C++ コンパイラーが使用されるときだけ含まれる、以下の追加のステートメントが入っています。

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

COBOL によるコーディング

WebSphere MQ プログラムを COBOL 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

名前付き定数

定数の名前は、その一部に下線文字 (_) が付いて示されています。 COBOL の場合は、下線文字の代わりにハイフン文字 (-) を使用しなければなりません。 文字ストリング値をもつ定数は、単一引用符 (') をストリング区切り文字として使用します。 コンパイラーがこの文字を受け入れるには、コンパイラー・オプション APOST を使用します。

コピー・ファイル CMQV には、名前付き定数の宣言レベル 10 の項目として入っています。 この定数を使用するには、レベル 01 の項目を明示的に宣言してから、COPY ステートメントを使用して次のように定数の宣言にコピーします。

```
WORKING-STORAGE SECTION.
01  MQM-CONSTANTS.
    COPY CMQV.
```

ただし、この方法を使用すると、定数が参照されない場合でもプログラム内のストレージを占有してしまいます。 定数が同じ実行単位内にある多数の個別プログラムに組み込まれると、定数のコピーが多数存在

することになります。これによって主ストレージのかなりの容量が使用されることがあります。 レベル 01 の宣言に GLOBAL 文節を追加することによって、このような状況を避けることができます。

```
* Declare a global structure to hold the constants
 01  MQM-CONSTANTS GLOBAL.
     COPY CMQV.
```

これによって、実行単位内の *1* セットの定数のためだけにストレージが割り振られます。そして定数は、レベル 01 の宣言を含むプログラムだけでなく、実行単位内の どの プログラムからでも参照できるようになります。

構造体のアライメントの確保

MQ 呼び出しの開始時に引き渡される IBM WebSphere MQ 構造体が、確実にワード境界に調整されるように注意を払ってください。 ワード境界は、32 ビット・プロセスでは 4 バイト、64 ビット・プロセスでは 8 バイト、128 ビット・プロセス (IBM i) では 16 バイトです。

可能な場合は、すべての IBM WebSphere MQ 構造体をまとめて配置し、すべての境界が合うように調整してください。

pTAL によるコーディング

IBM WebSphere MQ プログラムを pTAL 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

HP Integrity NonStop Server

IBM WebSphere MQ 構造体の定義と初期化

IBM WebSphere MQ 構造体用の pTAL 構造体定義が、^DEF で終わる名前で提供されています。 例えば IBM WebSphere MQ メッセージ記述子 (MQMD) 構造体および IBM WebSphere MQ メッセージ書き出しオプション (MQPMO) 構造体を作成するために、以下の pTAL 宣言をコーディングすることができます。

```
    STRUCT MYMD(MQMD^DEF);        ! Declare an MQMD structure
    STRUCT MYPMO(MQPMO^DEF);      ! Declare an MQPMO structure
```

IBM WebSphere MQ は、 IBM WebSphere MQ 構造体をデフォルト値で初期化するために、 ^DEFAULT で終わる名前を pTAL DEFINE に提供しています。 宣言される MQMD および MQPMO 構造体にデフォルト値を割り当てるために、以下の pTAL ステートメントがコーディングされます。

```
    MQMD^DEFAULT(MYMD);           ! Assign default values to an MQMD structure
    MQPMO^DEFAULT(MYPMO);         ! Assign default values to an MQPMO structure
```

同様のコードを使って、他の IBM WebSphere MQ 構造体を宣言および初期化することができます。

pTAL と CRE

pTAL プログラムは共通ランタイム環境を初期化できないため、C 言語または COBOL のメインルーチンと共に使用する必要があります。

IBM WebSphere MQ に備わっている pTAL サンプルは、AMQSPTM0.C という C 言語メインライン・ルーチンを使用します

MQCHAR データ型のパラメーター

MQGET、MQPUT、および MQPUT1 プロシージャーにはそれぞれ、MQCHAR .EXT データ・タイプの **Buffer** パラメーターがあります。 このパラメーターは、アプリケーションのメッセージ・データを送受信するために使用されます。

この種類のパラメーターは、pTAL サンプルでストリングの配列として示されます。 この方法でパラメーターを宣言することはできますが、通常は、これらのメッセージ内のデータのレイアウトを記述する構造体

として宣言する方が便利です。 プロシージャー・パラメーターは MQCHAR .EXT として宣言されますが、任意のデータのアドレスをプロシージャー呼び出しでパラメーターとして指定できます。

文字ストリングの取り扱い

キュー・マネージャーは、文字データをアプリケーションに戻すときに、文字データがフィールドの定義された長さになるように必ずブランク文字を埋め込みます。 キュー・マネージャーはヌル文字で終了するストリングを戻しませんが、入力値として使用することはできます。

Visual Basic によるコーディング

WebSphere MQ プログラムを Visual Basic 言語でコーディングするときは、以下の節で述べられている事項に注意してください。

注 : .NET 環境の外部では、WebSphere MQ での Visual Basic (VB) の サポートは V6.0 レベルで固定されています。 WebSphere MQ 7.0 以降で追加 された最新機能の大部分は VB アプリケーションでは使用できません。 VB.NET で プログラミングする場合､WebSphere MQ .NET クラスを使用してください。 詳細については、.NET の使用を参照してください。

Visual Basic は、Windows でのみサポートされています。

Visual Basic と WebSphere MQ の間のやり取りでバイナリー・データが意図せずに変換されてしまうのを避けるため、MQSTRING ではなく MQBYTE 定義を使用してください。 CMQB.BAS には、C「バイト」定義と同等で、WebSphere MQ 構造体においてこれらの定義を使用する、いくつかの新しい MQBYTE タイプが定義されています。 例えば、MQMD (メッセージ記述子) 構造体には、MQBYTE24 として MsgId (メッセージ ID) が定義されています。

Visual Basic には、ポインターのデータ・タイプがありません。そのため、他の WebSphere MQ データ構造体への参照は、ポインターではなく、オフセットによって行われます。 2 つのコンポーネント構造体からなる複合構造体を宣言し、呼び出しの際にその複合構造体を指定します。 WebSphere MQ における Visual Basic のサポートには、これを可能にするための MQCONNXAny 呼び出しが含まれており、クライアント・アプリケーションでクライアント接続のチャネル・プロパティーを指定できます。 ここでは、通常の MQCNO 構造体の代わりに、タイプを持たない構造体 (MQCNOCD) が受け入れられます。

MQCNOCD 構造体は、MQCNO の後ろに MQCD が付いた複合構造体です。 この構造体は、出口ヘッダー・ファイル CMQXB で宣言されます。 MQCNOCD 構造体の初期化には、ルーチン MQCNOCD_DEFAULTS を使用してください。 MQCONNX 呼び出しを行うサンプルが提供されています (amqscnxb.vbp)。

MQCONNXAny は MQCONNX と同じパラメーターを持ちますが、MQCONNXAny の場合は、*ConnectOpts* パラメーターが MQCNO データ・タイプではなく Any データ・タイプとして宣言されます。 これにより、関数は、MQCNO 構造体でも MQCNOCD 構造体でも受け入れられるようになります。 この関数は、メインのヘッダー・ファイル CMQB で宣言されます。

IBM WebSphere MQ オブジェクト・モデル

IBM WebSphere MQ オブジェクト・モデルは、クラス、メソッド、およびプロパティーで構成されます。この情報を使用して、これらの概念について学習します。

IBM WebSphere MQ オブジェクト・モデルのコンポーネントは次のとおりです。

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

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

以下のトピックでは、各 WebSphere MQ オブジェクト・モデルについて詳しく説明します。

- 88 ページの『クラス』
- 89 ページの『オブジェクト参照子』
- 89 ページの『戻りコード』

クラス

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

実際のモデルは、サポートされるオブジェクト指向環境によって若干異なります。

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

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

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

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

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

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

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

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

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

オブジェクト参照子

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

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

戻りコード

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

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

IBM WebSphere MQ classes for Java または IBM WebSphere MQ classes for JMS を使用する必要がありますか?

Java アプリケーションは、 IBM WebSphere MQ classes for Java または IBM WebSphere MQ classes for JMS のいずれかを使用して、IBM WebSphere MQ リソースにアクセスできます。 それぞれのアプローチにはそれぞれの利点があります。

IBM WebSphere MQ classes for Java は、メッセージ・キュー・インターフェース (MQI)、ネイティブ IBM WebSphere MQ API をカプセル化し、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用します。一方、 IBM WebSphere MQ classes for Java Message Service は、Sun の Java Message Service (JMS) インターフェースを実装します。

Java 以外の環境でプロシージャー型言語またはオブジェクト指向言語を使用して IBM WebSphere MQ に精通している場合は、 IBM WebSphere MQ classes for Java を使用して既存の知識を Java 環境に転送できます。 また、IBM WebSphere MQ classes for JMS では十分に利用できない、IBM WebSphere MQ の機能を全範囲にわたって活用することもできます。

IBM WebSphere MQ に慣れていない場合、または JMS の経験が既にある場合には、IBM WebSphere MQ classes for JMS を使用することにより、JMS API を使用して IBM WebSphere MQ リソースにアクセスするほうが便利かもしれません。 JMS は、 Java Platform, Enterprise Edition (Java EE) プラットフォームの不可欠な部分でもあります。 Java EE アプリケーションは、メッセージ駆動型 Bean (MDB) を使用してメッセージを非同期で処理でき、MDB は JMS メッセージのみを処理できます。 JMS は、 Java EE が IBM WebSphere MQ などの非同期メッセージング・システムと対話するための標準メカニズムでもあります。Java EE 準拠のすべてのアプリケーション・サーバーには JMS プロバイダーが含まれている必要があります。そのため、JMS を使用して異なるアプリケーション・サーバー間で通信したり、アプリケーションを変更せずに JMS プロバイダー間でアプリケーションを移植したりすることができます。

IBM WebSphere MQ アプリケーションの設計

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

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

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

- サーバー
- クライアント
- パブリッシュ/サブスクライブ

- Web サービス
- ユーザー出口、API 出口、およびインストール可能サービス

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

プログラミング言語

IBM WebSphere MQ では、アプリケーションの作成において、多数のプロシージャー型およびオブジェクト指向のプログラミング言語をサポートしています。 詳細については、78 ページの『使用するプログラミング言語の決定』を参照してください。

複数のプラットフォームに対応するアプリケーション

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

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

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

キューのタイプ

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

キュー・マネージャー・クラスターの使用

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

メッセージのタイプ

単純なメッセージ用にデータグラムを用いる場合もあれば、別の状況では要求メッセージ (応答を期待するメッセージ) を用いる場合もあります。 あるメッセージには異なった優先順位を割り当てたい場合もあります。 メッセージの設計について詳しくは、92 ページの『メッセージの設計』を参照してください。

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

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

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

IBM WebSphere MQ プログラムの制御
あるプログラムを自動的に開始したり、特定のメッセージがキューに入るまでプログラムを待機させたい場合があります (IBM WebSphere MQ トリガー操作 機能を使用します。この機能については、330 ページの『トリガーによる IBM WebSphere MQ アプリケーションの開始』を参照)。 あるいは、キュー上のメッセージが十分な速さで処理されない場合に、アプリケーションの別のインスタンスを開始することもできます (インスツルメンテーション・イベントで説明されているように、 IBM WebSphere MQ インスツルメンテーション・イベント 機能を使用します)。

IBM WebSphere MQ クライアント上でのアプリケーションの稼働
MQI の全機能は、クライアント環境でサポートされています。これにより、再リンク対象の大部分の IBM WebSphere MQ アプリケーションは、IBM WebSphere MQ MQI クライアント上で稼働できます。IBM WebSphere MQ MQI クライアント上のアプリケーションは、MQI ライブラリーではなく、MQIC ライブラリーにリンクしてください。

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

詳しくは、 361 ページの『IBM WebSphere MQ MQI クライアント環境でのアプリケーションの実行』を参照してください。

アプリケーションのパフォーマンス
設計上の決定事項がアプリケーションのパフォーマンスに影響を与える場合があります。 IBM WebSphere MQ アプリケーションのパフォーマンスを向上させるための推奨事項については、93 ページの『アプリケーション設計とパフォーマンス』 を参照してください。

IBM WebSphere MQ の高度な手法
より高機能なアプリケーションでは、応答の対応付けや IBM WebSphere MQ コンテキスト情報の生成と送信など、IBM WebSphere MQ の高度な手法を使用することをお勧めします。 詳しくは、94 ページの『IBM WebSphere MQ の高度な手法』を参照してください。

データの保護およびデータの整合性の維持
メッセージと共に渡されるコンテキスト情報を使用して、そのメッセージが許容ソースから送信されたものかどうかを調べることができます。 また、IBM WebSphere MQ やご使用のオペレーティング・システムで提供される同期点処理機能を使用して、データが確実に他のリソースとの整合性を持ち続けるようにすることができます (詳細は、324 ページの『作業単位のコミットとバックアウト』 を参照してください)。 IBM WebSphere MQ メッセージの 持続性 機能を使用して、重要なメッセージを確実に配信することができます。

IBM WebSphere MQ アプリケーションのテスト

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

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

関連概念
IBM WebSphere MQ 技術概説
8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

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

353 ページの『クライアント・アプリケーションの作成』

WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

564 ページの『.NET の使用』
.NET プログラミング・フレームワークで作成されたプログラムは、WebSphere MQ classes for .NET によって、WebSphere MQ に WebSphere MQ MQI クライアントとして接続するか、または WebSphere MQ サーバーに直接接続することができます。

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

721 ページの『WebSphere MQ classes for JMS の使用』
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) は、 WebSphere MQ に付属する JMS プロバイダーです。 javax.jms パッケージで定義されたインターフェースの実装に加えて、WebSphere MQ classes for JMS は JMS API への 2 セットの拡張機能を提供します。

658 ページの『WebSphere MQ classes for Java の使用』
WebSphere MQ classes for Java を使用すると、Java 環境で WebSphere MQ を使用できます。 Java アプリケーションは、 WebSphere MQ classes for Java または WebSphere MQ classes for JMS のいずれかを使用して、 WebSphere MQ リソースにアクセスできます。

1038 ページの『Component Object Model インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用』
WebSphere MQ Automation Classes for ActiveX (MQAX) は ActiveX コンポーネント群で、 WebSphere MQ にアクセスするためにアプリケーションで使用できるクラスを提供します。

メッセージの設計

メッセージを設計する際には、この情報に示される事柄を考慮してください。

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

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

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

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

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

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

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

キュー・マネージャーの再始動がメッセージに与える影響
キュー・マネージャーを再始動すると、そのキュー・マネージャーはすべての持続メッセージを保持し、必要に応じて WebSphere MQ ログ・ファイルからそれらのメッセージを回復します。 非持続メッセージや一時動的キューは、保存されません。 廃棄したくないメッセージは、作成時に持続として定義する必要があります。 WebSphere MQ for Windows または WebSphere MQ (UNIX and Linux システム用) のアプリケーションを作成するとき、アプリケーションがログ・ファイルの限界まで実行される

ような設計となる危険性を減らすため、ログ・ファイルの割り振りに関してシステムがどのようにセットアップされているかを確認してください。

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

受け取るキューの量
1 つのメッセージを複数のキューに入れる必要がある場合は、配布リストを使用する方法 と、トピックにパブリッシュする方法があります。

アプリケーション設計とパフォーマンス

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

効率的なアプリケーションを設計するための方法を以下に示します。

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

メッセージ長の影響

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

メッセージ持続の影響

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

特定メッセージの検索

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

可変長メッセージを含んでいるキュー

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

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

同期点の頻度

1 つの同期点内で多数の MQPUT 呼び出しまたは MQGET 呼び出しをコミットなしで発行するプログラムは、パフォーマンス上の問題を引き起こす可能性があります。 影響を受けるキューは、現在アクセス不能なメッセージで満杯になり、他のタスクはそれらのメッセージを取得するために待機することがあります。これは、使用されるストレージの面からも、 メッセージを取得しようとするタスクでスレッドが拘束されるという面からも、 好ましいことではありません。

MQPUT1 呼び出しの使用

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

使用するスレッドの数

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

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

IBM WebSphere MQ の高度な手法

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

メッセージの待機

キューを扱うプログラムは、次のような方法でメッセージを待機できます。

- メッセージが到着するか、または指定の時間間隔が経過するまで待機する (267 ページの『メッセージの待機』を参照)。
- メッセージの到着時に実行されるコールバック出口を確立します。 詳細は、33 ページの『IBM WebSphere MQ メッセージの非同期コンシューム』を参照してください。
- キューに対して、メッセージが到着しているかどうかを検査するための呼び出しを周期的に出す (ポーリング)。 これはパフォーマンスに影響する可能性があるため、通常は推奨されません。

応答の対応付け

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

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

コンテキスト情報の設定と使用

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

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

コンテキスト情報の設定と使用の詳細については、38 ページの『メッセージ・コンテキスト』を参照してください。

WebSphere MQ プログラムの自動的な開始

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

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

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

トリガー操作の詳細については、330 ページの『トリガーによる IBM WebSphere MQ アプリケーションの開始』を参照してください。 トリガー操作は、プログラムを自動的に開始する方法の一つにすぎません。 例えば、WebSphere MQ 以外の機能を使ってタイマーで自動的にプログラムを開始することもできます。

WebSphere MQ はサービス・オブジェクトを使用して、キュー・マネージャーの開始時に WebSphere MQ プログラムを開始するように定義できます。 詳細は、「サービス・オブジェクト」を参照してください。

WebSphere MQ の報告の生成

アプリケーション内では、次の報告を要求できます。

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

これらについては、11 ページの『レポート・メッセージ』を参照してください。

クラスターおよびメッセージ類似性

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

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

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

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

詳しくは、 メッセージの類縁性の処理 を参照してください。

WebSphere MQ プログラムのサンプル

この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

- 96 ページの『分散プラットフォームにおけるサンプル・プログラム』

関連概念

8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

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

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

353 ページの『クライアント・アプリケーションの作成』
WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

950 ページの『WebSphere MQ での Web サービスの使用』
IBM WebSphere MQ transport for SOAP または IBM WebSphere MQ bridge for HTTP を使用して、Web サービス用の IBM WebSphere MQ アプリケーションを開発できます。

277 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
パブリッシュ/サブスクライブ WebSphere MQ アプリケーションの作成を開始します。

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

553 ページの『プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

分散プラットフォームにおけるサンプル・プログラム

このトピックでは、C および COBOL で作成された、 IBM WebSphere MQ に付属のサンプル・プログラムについて説明します。 このサンプルでは、メッセージ・キュー・インターフェース (MQI) の一般的な使用法を示します。

このサンプルは、一般的なプログラミング技法の解説を目的としたものではありません。したがって、実動プログラムには組み込んだほうがよいエラー・チェックの一部が省略されています。 それでも、これらのサンプルは、メッセージ・キューイング・プログラムの基礎として使用するのには適しています。

すべてのサンプルのソース・コードが、この製品で提供されています。このソースには、プログラムによって示されるメッセージ・キューイング技法を説明するコメントが含まれています。

C++ サンプル・プログラム: C++ で利用できるサンプル・プログラムについては、C++ の使用を参照してください。

サンプル・プログラム名は接頭部 amq で始まっています。 名前の中の 4 番目の文字は、プログラム言語と、必要な場合にはコンパイラーを示します。

s	C 言語
0	IBM および Micro Focus コンパイラーでの COBOL 言語
i	IBM コンパイラーのみでの COBOL 言語
m	Micro Focus コンパイラーのみでの COBOL 言語

実行可能ファイルの 8 番目の文字は、サンプルをローカル・バインディング・モードまたはクライアント・モードのどちらで実行するかを示します。 8 番目の文字がない場合、サンプルはローカル・バインディング・モードで実行します。 8 番目の文字が「c」である場合、サンプルはクライアント・モードで実行します。 キュー・マネージャーがクライアント接続を受け入れるようにセットアップするには、詳細について、108 ページの『サンプル・プログラムの作成と実行』を参照してください。

以下のリンクを使用して、サンプル・プログラムについての詳細を確認してください。

- 98 ページの『サンプル・プログラムの中で示されている機能』
- 133 ページの『パブリッシュ/サブスクライブのサンプル・プログラム』
- 139 ページの『書き込みサンプル・プログラム』
- 126 ページの『配布リスト・サンプル・プログラム』
- 115 ページの『ブラウズ・サンプル・プログラム』
- 116 ページの『ブラウザー・サンプル・プログラム』
- 127 ページの『読み取りサンプル・プログラム』
- 140 ページの『参照メッセージ・サンプル・プログラム』
- 146 ページの『要求サンプル・プログラム』
- 132 ページの『照会サンプル・プログラム』
- 133 ページの『メッセージ処理サンプル・プログラムの照会プロパティー』
- 150 ページの『設定サンプル・プログラム』
- 127 ページの『エコー・サンプル・プログラム』
- 118 ページの『データ変換サンプル・プログラム』
- 154 ページの『トリガー・サンプル・プログラム』
- 114 ページの『非同期書き込みサンプル・プログラム』
- 119 ページの『データベース調整サンプル』
- 117 ページの『CICS トランザクション・サンプル』
- 155 ページの『TUXEDO サンプル』
- 126 ページの『送達不能キュー・ハンドラーのサンプル』
- 117 ページの『Connect サンプル・プログラム』
- 112 ページの『API 出口サンプル・プログラム』
- 168 ページの『Windows システムでの SSPI セキュリティー出口の使用』
- 169 ページの『リモート・キューを使用するサンプルの実行』
- 169 ページの『クラスター・キュー・モニターのサンプル・プログラム (AMQSCLM)』
- 179 ページの『接続エンドポイント検索 (CEPL) のサンプル・プログラム』

サンプル・プログラムの中で示されている機能

一連の表に、WebSphere MQ のサンプル・プログラムで示される技法について説明します。

すべてのサンプルは、MQOPEN および MQCLOSE 呼び出しを使用してキューをオープンおよびクローズするので、これらの技法はこの表では別々に記載されていません。 以下の中から、該当するプラットフォームが含まれている見出しを参照してください。

UNIX and Linux システム用のサンプル

このトピックでは、WebSphere MQ (UNIX and Linux システム用) のサンプル・プログラムで示される技法について説明します。

UNIX および Linux システム上の WebSphere MQ 用のサンプル・プログラムが保管されている場所については、110 ページの『UNIX システムでのサンプル・プログラムの作成と実行』 を参照してください。

98 ページの表 14 この表は、提供されている C および COBOL ソース・ファイル、およびサーバーまたはクライアント実行可能プログラムが含まれているかどうかをリストしています。

表 *14. MQI* の使用方法を示す *WebSphere MQ (UNIX and Linux* 用*)* サンプル・プログラム *(C* および *COBOL)*

技法	**C (ソース) (100 ページの『1』)**	**COBOL (ソース) (100 ページの『2』)**	**サーバー (C 実行可能プログラム)**	**クライアント (C 実行可能プログラム) (100 ページの『3』)**
パブリッシュ/サブスクライブ・インターフェースの使用	amqspuba amqssuba amqssbxa	サンプルなし	amqspub amqssub amqssbx	サンプルなし
MQPUT 呼び出しを使用するメッセージの書き込み	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 呼び出しを使用する単一メッセージの書き込み	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
配布リストへのメッセージの書き込み (100 ページの『4』)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
要求メッセージに対する応答	amqsinqa	amqminqx amqiinqx	amqsinq	サンプルなし
メッセージを読み取る (待機間隔なし)	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
メッセージの読み取り (時間制限付きの待機)	amqsget0	amq0get0	amqsget	amqsgetc
メッセージの読み取り (無制限の待機)	amqstrg0	サンプルなし	amqstrg	amqstrgc
メッセージの読み取り (データ変換あり)	amqsecha	サンプルなし	amqsech	サンプルなし
キューへの参照メッセージの書き込み (100 ページの『4』)	amqsprma	サンプルなし	amqsprm	amqsprmc
キューからの参照メッセージの読み取り (100 ページの『4』)	amqsgrma	サンプルなし	amqsgrm	amqsgrmc
参照メッセージのチャネル出口 (100 ページの『4』)	amqsqrma amqsxrma	サンプルなし	amqsxrm	サンプルなし
メッセージの最初の 20 文字のブラウズ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージ全体のブラウズ	amqsbcg0	サンプルなし	amqsbcg	amqsbcgc

表 14. *MQI* の使用方法を示す *WebSphere MQ (UNIX and Linux* 用*)* サンプル・プログラム *(C* および *COBOL) (*続き*)*

技法	C (ソース) (100 ページの『1』)	COBOL (ソース) (100 ページの『2』)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム) (100 ページの『3』)
共用入力キューの使用	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
排他的入力キューの使用	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 呼び出しの使用	amqsinqa	amqminqx amqiinqx	amqsinq	サンプルなし
MQSET 呼び出しの使用	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
応答先キューの使用	amqsreq0	amq0req0	amqsreq	amqsreqc
メッセージ例外の要求	amqsreq0	amq0req0	amqsreq	サンプルなし
切り捨てられたメッセージの受け入れ	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
解決されたキュー名の使用	amqsgbr0	amq0gbr0	amqsgbr	サンプルなし
処理のトリガー操作	amqstrg0	サンプルなし	amqstrg	amqstrgc
データ変換の使用	(100 ページの『5』)	サンプルなし	サンプルなし	サンプルなし
SQL を使用する単一データベースにアクセスする WebSphere MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	サンプルなし	サンプルなし
SQL を使用する 2 つのデータベースにアクセスする WebSphere MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	サンプルなし	サンプルなし
CICS トランザクション (100 ページの『6』)	amqscic0.ccs	サンプルなし	amqscic0	サンプルなし
Encina トランザクション (100 ページの『4』)	amqsxae0	サンプルなし	amqsxae0	サンプルなし
メッセージを書き込む TUXEDO トランザクション (100 ページの『7』)	amqstxpx	サンプルなし	サンプルなし	サンプルなし
メッセージを読み取る TUXEDO トランザクション (100 ページの『7』)	amqstxgx	サンプルなし	サンプルなし	サンプルなし
TUXEDO のサーバー (100 ページの『7』)	amqstxsx	サンプルなし	サンプルなし	サンプルなし
送達不能キュー・ハンドラー	ディレクトリー ./tools/c/Samples/dlq (100 ページの『8』)	サンプルなし	amqsdlq	サンプルなし
MQI クライアントからのメッセージの書き込み	サンプルなし	サンプルなし	サンプルなし	amqsputc

表 14. MQI の使用方法を示す WebSphere MQ (UNIX and Linux 用) サンプル・プログラム (C および COBOL) (続き)

技法	C (ソース) (100 ページの『1』)	COBOL (ソース) (100 ページの『2』)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム) (100 ページの『3』)
MQI クライアントからのメッセージの読み取り	サンプルなし	サンプルなし	サンプルなし	amqsgetc
MQCONNX を使用するキュー・マネージャーへの接続	amqscnxc	サンプルなし	サンプルなし	amqscnxc
API 出口の使用	amqsaxe0	サンプルなし	amqsaxe	サンプルなし
クラスター・ワークロード・バランシング出口	amqswlm0	サンプルなし	amqswlm	サンプルなし
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	amqsapt0	サンプルなし	amqsapt	amqsaptc
再接続可能クライアント	amqsphac amqsghac amqsmhac	サンプルなし	適用外	amqsphac amqsghac amqsmhac
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用	amqscbf0	サンプルなし	amqscbf	amqscbfc
MQCONNX での SSL/TLS 接続情報の指定	amqssslc	サンプルなし	適用外	amqssslc

注 :

1. WebSphere MQ MQI クライアントの実行可能なバージョンのサンプルは、サーバー環境で実行されるサンプルと同じソースを共用します。
2. 「amqm」で始まるプログラムは Micro Focus COBOL コンパイラーで、「amqi」で始まるプログラムは IBM COBOL コンパイラーで、また「amq0」で始まるプログラムは Micro Focus COBOL か IBM COBOL コンパイラーのどちらかで、それぞれコンパイルしてください。
3. WebSphere MQ for HP-UX では、WebSphere MQ MQI クライアントの実行可能なバージョンのサンプルはありません。
4. WebSphere MQ for AIX、WebSphere MQ for HP-UX、および WebSphere MQ for Solaris でのみサポートされています。
5. WebSphere MQ for AIX、WebSphere MQ for HP-UX、および WebSphere MQ for Solaris では、このプログラムは `amqsvfc0.c` と呼ばれています。
6. CICS は、WebSphere MQ for AIX および WebSphere MQ for HP-UX でのみサポートされています。
7. TUXEDO は、System p 上の WebSphere MQ for Linux ではサポートされません。
8. 送達不能キュー・ハンドラーのソースは、複数のファイルから構成され、個別のディレクトリーに提供されます。

UNIX and Linux システムのサポートに関する詳細情報は、WebSphere MQ システム要件ページ IBM WebSphere MQ で入手可能です。

HP Integrity NonStop Server 用の IBM WebSphere MQ クライアントのサンプル

このトピックでは、IBM WebSphere MQ クライアント (HP Integrity NonStop Server システム用) のサンプル・プログラムで示される技法について説明します。

101 ページの表 15 この表には、付属する C、COBOL、および pTAL のソース・サンプル・プログラムをリストしています。

表 15. C、COBOL、および pTAL の使用方法を示す IBM WebSphere MQ (HP Integrity NonStop Server 用) サンプル・プログラム

技法	C				COBOL		pTAL	
	OSS (ソース)	OSS (実行可能ファイル)	Guardian (ソース)	Guardian (実行可能ファイル)	OSS (ソース)	Guardian (ソース)	OSS (ソース)	Guardian (ソース)
パブリッシュ/サブスクライブ・インターフェースの使用	amqspuba.c amqssbxa.c amqssuba.c amqspse0.c	amqspubc amqssbxc amqssubc	MQSPUBC MQSSBXC MQSSUBC	AMQSPUBC AMQSSBXC AMQSSUBC	amq0pub0.cbl amq0sub0.cbl	MQSPUBL MQSSUBL	amqtpub0.tal amqtsub0.tal	MQSPUBT MQSSUBT
MQPUT 呼び出しを使用するメッセージの書き込み	amqsput0.c	amqsputc	MQSPUTC	AMQSPUTC	amq0put0.cbl	MQSPUTL	amqtput0.tal	MQSPUTT
MQPUT1 呼び出しを使用する単一メッセージの書き込み	amqsecha.c	amqsechc	MQSECHC	AMQSECHC			amqtech0.tal	MQSECHT
配布リストへのメッセージの書き込み	amqsptl0.c	amqsptlc	MQSPTLC	AMQSPTLC	amq0ptl0.cbl	MQSPTLL		
要求メッセージに対する応答	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
メッセージを読み取る (待機間隔なし)	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
メッセージの読み取り (時間制限付きの待機)	amqsget0.c	amqsgetc	MQSGETC	AMQSGETC	amq0get0.cbl	MQSGETL	amqtget0.tal	MQSGETT

表 15. C、COBOL、および pTAL の使用方法を示す IBM WebSphere MQ (HP Integrity NonStop Server 用) サンプル・プログラム (続き)

技法	C				COBOL		pTAL	
メッセージの読み取り (無制限の待機)	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
メッセージの読み取り (データ変換あり)	amqsecha.c	amqsechc	MQSECHC	AMQSECHC				
キューへの参照メッセージの書き込み	amqsprma.c	amqsprmc	MQSPRMC	AMQSPRMC				
キューからの参照メッセージの読み取り	amqsgrma.c	amqsgrmc	MQSGRMC	AMQSGRMC				
参照メッセージのチャネル出口	amqsqrma.c amqsxrma.c		MQSQRMC MQSXRMC					
メッセージの最初の 20 文字のブラウズ	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
メッセージ全体のブラウズ	amqsbcg0.c	amqsbcgc	MQSBCGC	AMQSBCGC				
共用入力キューの使用	amqsinqa.c	amqsinqc	MQSINQC	MQSINQC				
排他的入力キューの使用	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
MQINQ 呼び出しの使用	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
MQSET 呼び出しの使用	amqsseta.c	amqssetc	MQSSETC	AMQSSETC				

表 *15. C*、*COBOL*、および *pTAL* の使用方法を示す *IBM WebSphere MQ (HP Integrity NonStop Server* 用*)* サンプル・プログラム *(*続き*)*

技法	C				COBOL		pTAL	
応答先キューの使用	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
メッセージ例外の要求	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
切り捨てられたメッセージの受け入れ	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
解決されたキュー名の使用	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
処理のトリガー操作	amqstrg0.c	amqstrgc	MQSTRGC	AMQSTRGC				
データ変換の使用	amqsvfc0.c							
送達不能キュー・ハンドラー (1)	ディレクトリー ./samp/dlq							
MQCONNX を使用するキュー・マネージャーへの接続	amqscnxc.c	amqscnxc	MQSCNXC					
API 出口の使用	amqsaxe0.c amqsaem0.c							
クラスター・ワークロード・バランシング出口	amqswlm0.c		MQSWLMC					
クラスター・キュー・モニター	amqsclma.c							

表 15. C、COBOL、および pTAL の使用方法を示す IBM WebSphere MQ (HP Integrity NonStop Server 用) サンプル・プログラム (続き)

技法	C				COBOL		pTAL	
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	amqsapt0.c	amqsaptc	MQSAPTC	MQSAPTC				
再接続可能クライアント	amqsghac.c amqsmhac.c amqsphac.c	amqsghac amqsmhac amqsphac	MQSGHAC MQSMHAC MQSPHAC MQSFHAC	AMQSGHAC AMQSMHAC AMQSPHAC AMQSFHAC				
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用	amqscbf0.c	amqscbfc						
MQCONNX での SSL/TLS 接続情報の指定	amqssslc.c	amqssslc	MQSSSLC	AMQSSSLC				
アクティビティー・トレース	amqsact0.c	amqsactc	MQSACTC	AMQSACTC				
メッセージ・プロパティー	amqsiqma.c amqsstma.c	amqsiqmc amqsstmc	MQSIQMC MQSSTMC	AMQSIQMC AMQSSTMC				
コマンド・サーバー	amqsstop.c		MQSSTOC					
ログ・イベント	amqslog0.c	amqslogc	MQSLOGC	AMQSLOGC				

表 15. C、COBOL、および pTAL の使用方法を示す IBM WebSphere MQ (HP Integrity NonStop Server 用) サンプル・プログラム (続き)

技法	C				COBOL		pTAL	
会計	amqsmon0.c	amqsmonc	MQSMONC	AMQSMONC				
管理インターフェース	amqsaicq.c amqsaiem.c amqsailq.c							
pTAL を呼び出すための C 言語 main 関数の例			MQSPTMC					

注:

1. 送達不能キュー・ハンドラーのソースは、複数のファイルから構成され、個別のディレクトリーに提供されます。
2. HP Integrity NonStop Server プラットフォームでの IBM WebSphere MQ クライアント用アプリケーションの開発については、以下を参照してください。
   - 436 ページの『HP Integrity NonStop Server でのアプリケーションの構築』
     - 439 ページの『HP Integrity NonStop Server での C プログラムの作成』
     - 440 ページの『COBOL プログラムの作成』
     - 441 ページの『pTAL プログラムの作成』

IBM WebSphere MQ for Windows のサンプル

このトピックでは、IBM WebSphere MQ for Windows のサンプル・プログラムで示される技法について説明します。

105 ページの表 16 この表は、提供されている C および COBOL ソース・ファイル、およびサーバーまたはクライアント実行可能プログラムが含まれているかどうかをリストしています。

表 16. MQI の使用方法を示す IBM WebSphere MQ for Windows サンプル・プログラム (C および COBOL)

技法	C (ソース)	COBOL (ソース)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
パブリッシュ/サブスクライブ・インターフェースの使用	amqspuba amqssuba amqssbxa	サンプルなし	amqspub amqssub amqssbx	サンプルなし
MQPUT 呼び出しを使用するメッセージの書き込み	amqsput0	amq0put0	amqsput	amqsputc
MQPUT1 呼び出しを使用する単一メッセージの書き込み	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
配布リストへのメッセージの書き込み	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc

表 16. *MQI* の使用方法を示す *IBM WebSphere MQ for Windows* サンプル・プログラム *(C* および *COBOL) (*続き*)*

技法	C (ソース)	COBOL (ソース)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
要求メッセージに対する応答	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
メッセージを読み取る (待機間隔なし)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージの読み取り (時間制限付きの待機)	amqsget0	amq0get0	amqsget	amqsgetc
メッセージの読み取り (無制限の待機)	amqstrg0	サンプルなし	amqstrg	amqstrgc
メッセージの読み取り (データ変換あり)	amqsecha	サンプルなし	amqsech	amqsechc
キューへの参照メッセージの書き込み	amqsprma	サンプルなし	amqsprm	amqsprmc
キューからの参照メッセージの読み取り	amqsgrma	サンプルなし	amqsgrm	amqsgrmc
参照メッセージのチャネル出口	amqsqrma amqsxrma	サンプルなし	amqsxrm	サンプルなし
メッセージの最初の 20 文字のブラウズ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
メッセージ全体のブラウズ	amqsbcg0	サンプルなし	amqsbcg	amqsbcgc
共用入力キューの使用	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
排他的入力キューの使用	amqstrg0	amq0req0	amqstrg	amqstrgc
MQINQ 呼び出しの使用	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
MQSET 呼び出しの使用	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
MQINQMP 呼び出しの使用	amqsiqma	サンプルなし	サンプルなし	サンプルなし
応答先キューの使用	amqsreq0	amq0req0	amqsreq	amqsreqc
メッセージ例外の要求	amqsreq0	amq0req0	amqsreq	amqsreqc
切り捨てられたメッセージの受け入れ	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
解決されたキュー名の使用	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
処理のトリガー操作	amqstrg0	サンプルなし	amqstrg	amqstrgc
データ変換の使用	amqsvfc0	サンプルなし	サンプルなし	サンプルなし
SQL を使用する単一データベースにアクセスする WebSphere MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxas0.sqc DB2 amqsxas0.ec Informix	amq0xas0.sqb	サンプルなし	サンプルなし
SQL を使用する 2 つのデータベースにアクセスする WebSphere MQ (XA 準拠のデータベース・マネージャーを調整)	amqsxag0.c amqsxab0.sqc DB2 amqsxaf0.sqc DB2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	サンプルなし	サンプルなし

表 16. *MQI* の使用方法を示す *IBM WebSphere MQ for Windows* サンプル・プログラム *(C* および *COBOL) (*続き*)*

技法	C (ソース)	COBOL (ソース)	サーバー (C 実行可能プログラム)	クライアント (C 実行可能プログラム)
メッセージを書き込む TUXEDO トランザクション	amqstxpx	サンプルなし	サンプルなし	サンプルなし
メッセージを読み取る TUXEDO トランザクション	amqstxgx	サンプルなし	サンプルなし	サンプルなし
TUXEDO のサーバー	amqstxsx	サンプルなし	サンプルなし	サンプルなし
送達不能キュー・ハンドラー	ディレクトリー ./tools/c/Samples/dlq (107 ページの『1』)	サンプルなし	amqsdlq	サンプルなし
WebSphere MQ MQI クライアントからのメッセージの書き込み	サンプルなし	サンプルなし	サンプルなし	amqsputc
WebSphere MQ MQI クライアントからのメッセージの読み取り	サンプルなし	サンプルなし	サンプルなし	amqsgetc
MQCONNX を使用するキュー・マネージャーへの接続	amqscnxc	サンプルなし	サンプルなし	amqscnxc
API 出口の使用	amqsaxe0	サンプルなし	amqsaxe	サンプルなし
クラスター・ワークロード・バランシング	amqswlm0	サンプルなし	amqswlm	サンプルなし
SSPI セキュリティー・ルーチン	amqsspin	サンプルなし	amqrspin.dll	amqrspin.dll
MQSTAT 呼び出しを使用するメッセージの非同期書き込みおよび状況の取得	amqsapt0	サンプルなし	amqsapt	amqsaptc
再接続可能クライアント	amqsphac amqsghac amqsmhac	サンプルなし	適用外	amqsphac amqsghac amqsmhac
複数のキューからメッセージを非同期にコンシュームするためのメッセージ・コンシューマーの使用	amqscbf0	サンプルなし	amqscbf	amqscbfc
MQCONNX での SSL/TLS 接続情報の指定	amqssslc	サンプルなし	適用外	amqssslc

注 :

1. 送達不能キュー・ハンドラーのソースは、複数のファイルから構成され、個別のディレクトリーに提供されます。

IBM WebSphere MQ for Windows 用の *Visual Basic* のサンプル

このトピックでは、IBM WebSphere MQ for Windows の Visual Basic サンプル・プログラムで示される技法について説明します。

108 ページの表 17 に、IBM WebSphere MQ for Windows サンプル・プログラムで示される技法についてまとめます。

プロジェクトには複数のファイルが含まれている可能性があります。 Visual Basic 内のプロジェクトをオープンすると、その他のファイルは自動的に読み込まれます。 実行可能プログラムはありません。

mqtrivc.vbp 以外のサンプル・プロジェクトはすべて、IBM WebSphere MQ サーバーとともに使用されるように設定されています。 IBM WebSphere MQ クライアントで使用されるようにサンプル・プロジェクトを変更する方法については、466 ページの『Windows での Visual Basic プログラムの作成』を参照してください。

表 *17. MQI* の使用方法を示す *IBM WebSphere MQ for Windows* サンプル・プログラム *(Visual Basic)*

技法	**プロジェクト・ファイル名**
MQPUT 呼び出しを使用するメッセージの書き込み	amqsputb.vbp
MQGET 呼び出しを使用するメッセージの読み取り	amqsgetb.vbp
MQGET 呼び出しを使用するキューのブラウズ	amqsbcgb.vbp
単純な MQGET および MQPUT サンプル (クライアント)	mqtrivc.vbp
単純な MQGET および MQPUT サンプル (サーバー)	mqtrivs.vbp
MQPUT および MQGET を使用するストリングおよびユーザー定義構造体の、書き込みおよび読み取り	strings.vbp
PCF 構造体を使用したチャネルの開始および停止	pcfsamp.vbp
MQAI を使用するキューの作成	amqsaicq.vbp
MQAI を使用するキュー・マネージャーのキューのリスト表示	amqsailq.vbp
MQAI を使用するイベントのモニター	amqsaiem.vbp

サンプル・プログラムの作成と実行

クライアント・モードで実行するアプリケーションからの着信接続要求を安全に受け入れるように、キュー・マネージャーを構成します。

始める前に

キュー・マネージャーが既に存在しており、開始していることを確認します。 以下のようにしてチャネル認証レコードが既に使用可能になっているかどうかを判別します。

```
DISPLAY QMGR CHLAUTH
```

このタスクは、チャネル認証レコードが使用可能になっていることを前提としています。 このキュー・マネージャーが他のユーザーやアプリケーションによって使用されている場合、この設定を変更すると、他のすべてのユーザーとアプリケーションが影響を受けます。 キュー・マネージャーがチャネル認証レコードを利用しない場合には、ステップ 109 ページの『4』 を代替認証方式 (例えば、セキュリティー出口など) に置き換えて、MCAUSER をステップ 109 ページの『1』 で取得する *non-privileged-user-id* に設定することができます。

アプリケーションが使用すると予期されるチャネル名を把握し、アプリケーションがそのチャネルを使用できるようにする必要があります。 また、アプリケーションが使用すると予期されるオブジェクト (例えば、キューやトピック) も把握し、アプリケーションがこれらのオブジェクトを使用できるようにする必要もあります。

このタスクについて

このタスクにより、キュー・マネージャーに接続するクライアント・アプリケーションで使用する、非特権ユーザー ID が作成されます。 クライアント・アプリケーションがこのユーザー ID を使用して必要とするチャネルとキューを使用できるようにするためにのみ、アクセス権限が付与されます。

手順

1. キュー・マネージャーが実行されているシステムでユーザー ID を取得します。 このタスクの場合、このユーザー ID は特権管理ユーザーにすることはできません。 このユーザー ID の権限は、クライアント接続をキュー・マネージャーで実行するためのものです。
2. 以下のコマンドを使用してリスナー・プログラムを開始します。

   qmgr は、キュー・マネージャーの名前です。
   nnnn は、選択したポート番号です。

   a) UNIX および Windows システムの場合:

   ```
   runmqlsr -t tcp -m qmgr -p nnnn
   ```

3. アプリケーションが SYSTEM.DEF.SVRCONN を使用する場合、このチャネルは既に定義済みです。 アプリケーションが別のチャネルを使用する場合は、以下のようにして、それを MQSC コマンドを発行して作成します。

   ```
   DEFINE CHANNEL('channel-name') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   channel-name は、チャネルの名前です。
4. ご使用のクライアント・システムの IP アドレスのみにチャネルの使用を許可するチャネル認証規則を、以下のように MQSC コマンドを発行して作成します。

   ```
   SET CHLAUTH('channel-name') TYPE(ADDRESSMAP) ADDRESS('client-machine-IP-address') +
   MCAUSER('non-privileged-user-id')
   ```

   channel-name は、チャネルの名前です。
   client-machine-IP-address は、クライアント・システムの IP アドレスです。
   サンプル・クライアント・アプリケーションがキュー・マネージャーと同じマシン上で実行されているときに、そのアプリケーションが「localhost」を使用して接続しようとしているのであれば、IP アドレス「127.0.0.1」を使用します。 複数のさまざまなクライアント・マシンが接続することになっている場合、単一の IP アドレスではなく、パターンや範囲を使用することができます。 詳細については、汎用 IP アドレスを参照してください。
   non-privileged-user-id は、ステップ 109 ページの『1』 で取得したユーザー ID です。
5. アプリケーションが SYSTEM.DEFAULT.LOCAL.QUEUE を使用する場合、このキューは既に定義済みです。 アプリケーションが別のキューを使用する場合は、以下のようにして、それを MQSC コマンドを発行して作成します。

   ```
   DEFINE QLOCAL('queue-name') DESCR('Queue for use by sample programs')
   ```

   queue-name は、キューの名前です。
6. キュー・マネージャーへの接続と照会のアクセス権限を以下のように付与します。

   a) UNIX および Windows システムの場合、MQSC コマンドを実行します。

   ```
   SET AUTHREC OBJTYPE(QMGR) PRINCIPAL('non-privileged-user-id') +
   AUTHADD(CONNECT, INQ)
   ```

   non-privileged-user-id は、ステップ 109 ページの『1』 で取得したユーザー ID です。
7. アプリケーションが Point-to-Point アプリケーションである場合 (つまりキューを使用する場合)、キューを使用したメッセージの照会、書き込み、および読み取りを、ユーザー ID を使用して行えるようにするために、以下のように MQSC コマンドを発行してアクセス権限を付与します。

   a) UNIX および Windows システムの場合、MQSC コマンドを実行します。

   ```
   SET AUTHREC PROFILE('queue-name') OBJTYPE(QUEUE) +
   PRINCIPAL('non-privileged-user-id') AUTHADD(PUT, GET, INQ, BROWSE)
   ```

   queue-name は、キューの名前です。
   non-privileged-user-id は、ステップ 109 ページの『1』 で取得したユーザー ID です。

8. アプリケーションがパブリッシュ/サブスクライブ・アプリケーションである場合 (つまりトピックを使用する場合)、使用するユーザー ID による、トピックを使用したパブリッシュ/サブスクライブを行えるようにするために、以下のように MQSC コマンドを発行してアクセス権限を付与します。

   a) UNIX および Windows システムの場合、MQSC コマンドを実行します。

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL('non-privileged-user-id') AUTHADD(PUB, SUB)
```

   non-privileged-user-id は、ステップ 109 ページの『1』 で取得したユーザー ID です。
   これにより、トピック・ツリー内のあらゆるトピックへのアクセス権限が *non-privileged-user-id* に付与されます。または、**DEFINE TOPIC** を使用してトピック・オブジェクトを定義し、そのトピック・オブジェクトにより参照されるトピック・ツリーの一部のみへのアクセス権限を付与することもできます。 詳細については、トピックへのユーザー・アクセスの制御を参照してください。

次のタスク

これで、クライアント・アプリケーションはキュー・マネージャーに接続し、キューを使用してメッセージの書き込みや読み取りができるようになりました。

関連タスク

UNIX または Linux システムおよび Windows での WebSphere MQ オブジェクトへのアクセス権限の付与

関連資料

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

UNIX システムでのサンプル・プログラムの作成と実行

表 *18. WebSphere MQ (UNIX and Linux* システム用*)* のサンプルの格納場所

内容	ディレクトリー
ソース・ファイル	*MQ_INSTALLATION_PATH*/samp
送達不能キュー・ハンドラーのソース・ファイル	*MQ_INSTALLATION_PATH*/samp/dlq
実行可能ファイル	*MQ_INSTALLATION_PATH*/samp/bin
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

インストール時にデフォルト値を使用した場合、WebSphere MQ (UNIX and Linux システム用) のサンプル・ファイルは、110 ページの表 18 に記載されているディレクトリーに格納されます。 このサンプルを実行するには、提供されている実行可能バージョンを使用するか、あるいは ANSI コンパイラーを使用して他のアプリケーションの場合と同様にソース・バージョンをコンパイルします。 これを行う方法については、111 ページの『サンプル・プログラムの実行』 を参照してください。

Windows システムでのサンプル・プログラムの作成と実行

表 *19. WebSphere MQ for Windows* 用のサンプルの格納場所

内容	ディレクトリー
C ソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥C¥Samples
送達不能ハンドラーのサンプルのソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥DLQ
COBOL ソース・コード	*MQ_INSTALLATION_PATH*¥Tools¥Cobol¥Samples

表 19. WebSphere MQ for Windows 用のサンプルの格納場所 (続き)	
内容	**ディレクトリー**
C の実行可能ファイル [1]	*MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥Bin (32 ビット・バージョン) *MQ_INSTALLATION_PATH*¥Tools¥C¥Samples¥Bin64 (64 ビット・バージョン)
サンプル MQSC ファイル	*MQ_INSTALLATION_PATH*¥Tools¥MQSC¥Samples
Visual Basic ソース・コード	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET サンプル	*MQ_INSTALLATION_PATH*¥Tools¥dotnet¥Samples

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

注:

1. いくつかの C 実行可能ファイル・サンプルの 64 ビット・バージョンを使用できます。

WebSphere MQ for Windows のサンプル・ファイルは、インストール時にデフォルトが使用された場合は、110 ページの表 19 にリストされているディレクトリーにあります。インストール・ドライブのデフォルトは < c:> です。サンプルを実行するには、提供されている実行可能バージョンを使用するか、他の WebSphere MQ for Windows アプリケーションと同様にソース・バージョンをコンパイルします。 この方法については、111 ページの『サンプル・プログラムの実行』を参照してください。

サンプル・プログラムの実行

各種プラットフォームにわたってサンプル・プログラムを実行する場合に、このトピックを参照してください。

いずれのサンプル・プログラムを実行する場合でも、キュー・マネージャーを作成し、デフォルト定義を設定してください。 これについては、管理で説明されています。

Windows、UNIX、および Linux プラットフォームの場合

サンプルで作業を行うために一連のキューが必要となります。 独自のキューを使用するか、またはサンプル MQSC ファイル amqscos0.tst を実行してキューを 1 セット作成します。

UNIX and Linux システムでこれを行うには、次のように入力します。

- `runmqsc QManagerName <amqscos0.tst >/tmp/sampobj.out`

`sampobj.out` ファイルを検査して、エラーがないことを確認します。

Windows システムでこれを行うには、次のように入力します。

- `runmqsc QManagerName  <amqscos0.tst > sampobj.out`

`sampobj.out` ファイルを検査して、エラーがないことを確認します。 このファイルは現行ディレクトリーにあります。

この時点で、サンプル・アプリケーションを実行できます。 次のように、サンプル・アプリケーションの名前を入力し、そのあとにパラメーターを入力します。

- `amqsput myqueue qmanagername`

この例では、`myqueue` がメッセージを入れるキューの名前で、`qmanagername` が `myqueue` を所有するキュー・マネージャーです。

各サンプルで使用されるパラメーターについては、個々のサンプルの説明を参照してください。

キュー名の長さ

COBOL サンプル・プログラムでは、キュー名をパラメーターとして渡す場合、必要に応じてブランク文字を埋め、合計で 48 文字にしなければなりません。 48 文字以外の場合には、理由コード 2085 でプログラムが異常終了します。

照会、設定、およびエコーのサンプル

照会、設定、およびエコーのサンプルでは、サンプル定義によってこれらのサンプルの C バージョンが起動されます。

COBOL バージョンを使用したい場合は、以下のプロセス定義を変更する必要があります。

- SYSTEM.SAMPLE.INQPROCESS
- SYSTEM.SAMPLE.SETPROCESS
- SYSTEM.SAMPLE.ECHOPROCESS

Windows、UNIX and Linux システムでこれを行うには、`amqscos0.tst` ファイルを編集し、C 実行可能ファイル名を COBOL 実行可能ファイル名に変更します。それから、上記のように **runmqsc** コマンドを実行します。

API 出口サンプル・プログラム

サンプル API 出口は、ユーザー指定ファイルに MQI トレースを生成し、これに MQAPI_TRACE_LOGFILE 環境変数で定義した接頭部を付けます。

API 出口について詳しくは、389 ページの『API 出口の作成とコンパイル』を参照してください。

ソース
amqsaxe0.c

バイナリー
amqsaxe

サンプル出口の構成

1. 以下のものを qm.ini ファイルに追加します。

   Windows 以外のプラットフォーム

   ```
   ApiExitLocal:
      Sequence=100
      Function=EntryPoint
      Module=MQ_INSTALLATION_PATH/samp/bin/amqsaxe
      Name=SampleApiExit
   ```

   ここで、*MQ_INSTALLATION_PATH* は、IBM WebSphere MQ がインストールされているディレクトリーを表します。

   Windows

   ```
   ApiExitLocal:
      Sequence=100
      Function=EntryPoint
      Module=MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
      Name=SampleApiExit
   ```

   ここで、*MQ_INSTALLATION_PATH* は、IBM WebSphere MQ がインストールされているディレクトリーを表します。

2. 環境変数を設定します。

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. アプリケーションを実行します。

   /tmp ディレクトリーに出力ファイルが作成されます。そのファイル名は MqiTrace.<pid>.<tid>.log のようになります。

非同期コンシューム・サンプル・プログラム

amqscbf サンプル・プログラムは、MQCB および MQCTL を使用して、複数のキューからのメッセージを非同期的にコンシュームする方法を示します。

amqscbf は、C ソース・コードとして提供され、Windows、UNIX and Linux プラットフォーム上の クライアントおよびサーバー用のバイナリー実行可能プログラムです。

このプログラムはコマンド行から開始され、以下のオプション・パラメーターを使用します。

```
Usage: [Options] <Queue Name> { <Queue Name> }
  where Options are:
    -m <Queue Manager Name>
    -o <Open options>
    -r <Reconnect Type>
       d Reconnect Disabled
       r Reconnect
       m Reconnect Queue Manager
```

複数のキューからメッセージを読み取るために、複数のキュー名を指定します (このサンプルでは、最大 10 個のキューが サポートされています。)

注 : クライアント・プログラムでは「*Reconnect Type*」のみが有効です。

例

以下の例は、amqscbf がサーバー・プログラムとして実行され、QL1 から 1 つのメッセージを読み取り、その後停止される様子を示しています。

WebSphere MQ Explorer を使用して、テスト・メッセージを QL1 に書き込みます。 Enter キーを押して、プログラムを停止します。

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

amqscbf が示す操作

このサンプルは、複数のキューからのメッセージを到着順に読み取る方法 を示します。 この方法では、同期 MQGET を使用して、さらに多くのコードが必要となります。 非同期コンシュームの場合、ポーリングは不要であり、スレッドおよびストレージの管理は WebSphere MQ によって実行されます。 「実環境の」例では、エラーの対処が 必要となります。このサンプルでは、エラーはコンソールに書き出されます。

サンプル・コードには以下のステップがあります。

1. 単一のメッセージ・コンシューム・コールバック関数を定義する。

   ```
   void MessageConsumer(MQHCONN      hConn,
                        MQMD    * pMsgDesc,
                        MQGMO   * pGetMsgOpts,
                        MQBYTE  * Buffer,
                        MQCBC   * pContext)
   { ... }
   ```

2. キュー・マネージャーに接続する。

   ```
   MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
   ```

3. 入力キューを開き、各入力キューを MessageConsumer コールバック関数と関連付ける。

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   cbd.CallbackFunction は、各キューごとに設定する必要はありません。これは入力専用フィールドです。 ただし、異なるコールバック関数を各キューに関連付けることができます。

4. メッセージのコンシュームを開始する。

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. ユーザーが Enter キーを押すまで待ってから、メッセージのコンシュームを停止する。

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. 最後に、キュー・マネージャーから切断する。

```
MQDISC(&Hcon,&CompCode,&Reason);
```

非同期書き込みサンプル・プログラム

amqsapt サンプルの実行および非同期書き込みサンプル・プログラムの設計について説明します。

非同期書き込みサンプル・プログラムは、非同期 MQPUT 呼び出しを使用してキューにメッセージを書き込み、次いで MQSTAT 呼び出しを使用して状況情報を取得します。 各種プラットフォームにおけるこのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』 を参照してください。

amqsapt サンプルの実行

このプログラムでは、パラメーターを以下の 6 つまで設定できます。

1. 宛先キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)
3. オープン・オプション (オプション)
4. クローズ・オプション (オプション)
5. 宛先キュー・マネージャーの名前 (オプション)
6. 動的キューの名前 (オプション)

キュー・マネージャーを指定しないと、amqsapt はデフォルトのキュー・マネージャーに接続されます。

非同期書き込みサンプル・プログラムの設計

このプログラムは、提供される出力オプション、あるいは MQOO_OUTPUT および MQOO_FAIL_IF_QUIESCING オプション付きで MQOPEN 呼び出しを使用して、メッセージを入れる宛先キューをオープンします。

キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。 プログラムを簡潔に保つには、これ以降の MQI 呼び出しで、プログラムが多数のオプションに対してデフォルト値を使用するようにします。

入力の各行ごとに、プログラムはテキストをバッファーに読み込み、MQPMO_ASYNC_RESPONSE 付きの MQPUT 呼び出しを使用します。こうして、その行のテキストが入ったデータグラム・メッセージを作成し、非同期で宛先キューに書き込みます。 プログラムは、入力が終了するか、MQPUT 呼び出しが失敗するまで処理を続行します。 プログラムは、入力が終了すると、MQCLOSE 呼び出しを使用して、キューをクローズします。

続いてプログラムは、MQSTAT 呼び出しを発行し、MQSTS 構造体が戻ります。また、正常に書き込まれたメッセージの数、警告が出されて書き込まれたメッセージの数、および失敗の回数を収めたメッセージを表示します。

ブラウズ・サンプル・プログラム

ブラウズ・サンプル・プログラムは、MQGET 呼び出しを使用してキュー上のメッセージを参照します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

ブラウズ・サンプル・プログラムの設計

このプログラムは、MQOO_BROWSE オプション付きの MQOPEN 呼び出しを使用して、宛先キューをオープンします。 キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。

キュー上のメッセージごとに、プログラムは MQGET 呼び出しを使用してキューからメッセージをコピーし、メッセージに含まれているデータを表示します。 MQGET 呼び出しでは次のオプションを使用します。

MQGMO_BROWSE_NEXT
MQOPEN 呼び出しのあと、ブラウズ・カーソルはキューの最初のメッセージの前に論理的に位置付けられるので、最初に呼び出しが行われると、このオプションによって、**最初の** メッセージが戻されます。

MQGMO_NO_WAIT
このプログラムは、キューにメッセージがない場合は、待機しません。

MQGMO_ACCEPT_TRUNCATED_MSG
この MQGET 呼び出しでは、固定サイズのバッファーを指定します。 メッセージがこのバッファーよりも大きい場合、このプログラムは、メッセージの切り捨てが行われたことを警告すると共に、その切り捨てられたメッセージを表示します。

このプログラムは、各 MQGET 呼び出しの後に MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアする方法を示しています。これは、この呼び出しにより、これらのフィールドが、取得するメッセージに含まれる値に設定されるためです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

このプログラムはキューの終わりまで処理を継続します。キューの終わりに到達すると、MQGET 呼び出しは、MQRC_NO_MSG_AVAILABLE 理由コードを戻し、プログラムは警告メッセージを表示します。 MQGET 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

続いて、プログラムは、MQCLOSE 呼び出しを使用してキューをクローズします。

UNIX、Linux および Windows システム

UNIX、Linux および Windows システムでのブラウズ・サンプル・プログラムについて学習する際に、このトピックを参照してください。

C バージョンのプログラムでは、次の 2 つのパラメーターをとります。

1. ソース・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、このプログラムはデフォルトのキュー・マネージャーに接続します。 例えば、それぞれ次のように入力します。

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

この例では、`myqueue` がメッセージを表示するキューの名前で、`qmanagername` が `myqueue` を所有するキュー・マネージャーとなります。

C サンプルの実行時に `qmanagername` を省略すると、デフォルトのキュー・マネージャーがそのキューを所有すると見なされます。

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target queue
```

各メッセージの最初の 50 文字のみが表示され、21 文字以降は、「- - - truncated」と表示されます。

ブラウザー・サンプル・プログラム

ブラウザー・サンプル・プログラムは、キュー上のすべてのメッセージのメッセージ記述子およびメッセージ内容フィールドの両方を読み取って、書き込みます。

このサンプル・プログラムは、単に技法を説明するだけではなく、ユーティリティーとして作成されています。 これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

このプログラムは、次のようなパラメーターを受け入れます。

1. ソース・キューの名前
2. キュー・マネージャーの名前
3. プロパティー用のオプション・パラメーター。

このプログラムの最初の 2 つの入力パラメーターは、必須です。 例えば、以下のいずれかの方法でプログラムを開始します。

- `amqsbcg myqueue qmanagername`
- `amqsbcgc myqueue qmanagername`

この例では、myqueue がメッセージをブラウズするキューの名前で、qmanagername が myqueue を所有するキュー・マネージャーです。

このプログラムは、各メッセージをキューから読み取り、次のものを stdout に書き込みます。

- 形式化メッセージ記述子フィールド
- メッセージ・データ (16 進数でダンプされるが、可能な場合は、文字形式)

プロパティー・パラメーターで許容される値は、次のとおりです。

値	動作
0	デフォルトの動作 (V6 と同じ)。 アプリケーションに送られるプロパティーは、メッセージの取得元の *PropertyControl* キュー属性に応じて異なります。

1	メッセージ・ハンドルが作成されて、MQGET と共に使用されます。 メッセージ記述子 (またはエクステンション) に含まれるプロパティーを除くメッセージ・プロパティーは、メッセージ記述子と同様の方式で表示されます。 以下に例を示します。
	`****Message properties****`
	`  <property name> : <property value>`
	または、使用可能なプロパティーが存在しない場合は、次のようになります。
	`****Message properties****`
	`  None`
	メッセージ記述子の場合と同様に、数値は printf を使って表示され、ストリング値は単一引用符で囲まれ、バイト・ストリングは X および単一引用符で囲まれます。

2	MQGMO_NO_PROPERTIES が指定され、メッセージ記述子プロパティーだけが戻されます。
3	MQGMO_PROPERTIES_FORCE_MQRFH2 が指定され、すべてのプロパティーがメッセージ・データ内に戻されます。
4	MQGMO_PROPERTIES_COMPATIBILITY が指定され、バージョン 6 プロパティーが含まれるかどうかに応じてすべてのプロパティーを戻すことができます。 含まれない場合、プロパティーは破棄されます。

このプログラムでは、メッセージの先頭からの 65535 文字が出力されるという制限があり、それより長いメッセージを読み取ると、メッセージが切り捨てられた という理由で異常終了します。

このユーティリティーからの出力の例については、管理を参照してください。

CICS トランザクション・サンプル

サンプルの CICS トランザクション・プログラムは、ソース・コードに amqscic0.ccs、実行可能なバージョンに amqscic0 という名前が付いて提供されています。 トランザクションは、CICS の標準機能を使用して作成できます。

ご使用のプラットフォームで必要となるコマンドの詳細については、429 ページの『IBM WebSphere MQ アプリケーションの構築』 を参照してください。

トランザクションは、デフォルトのキュー・マネージャーの伝送キュー SYSTEM.SAMPLE.CICS.WORKQUEUE からメッセージを読み取り、メッセージの伝送ヘッダーに名前があるローカル・キューにそれらを配置します。 エラーはすべて、キュー SYSTEM.SAMPLE.CICS.DLQ に送信されます。

注 : サンプル MQSC スクリプト amqscic0.tst を使用すると、これらのキューやサンプル入力キューを作成できます。

Connect サンプル・プログラム

Connect サンプル・プログラムによって、クライアントからの MQCONNX 呼び出しとそのオプションを調べることができます。 このサンプルでは、MQCONNX 呼び出しを使用してキュー・マネージャーに接続し、MQINQ 呼び出しを使用してキュー・マネージャーの名前を照会し、それを表示します。 また、amqscnxc サンプルの実行について説明します。

注 : Connect サンプル・プログラムは、クライアント・サンプルです。 このプログラムをコンパイルしてサーバー上で実行することができますが、その機能はクライアント上でのみ意味をもち、クライアント実行可能ファイルのみが提供されます。

amqscnxc サンプルの実行

Connect サンプル・プログラムのコマンド行構文は次のようになります。

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [QMgrName]
```

パラメーターはオプションであり、その順序は重要ではありません。ただし、QMgrName を指定する場合は、最後に置く必要があります。 パラメーターとして次のものがあります。

ConnName
サーバー・キュー・マネージャーの TCP/IP 接続名

SvrconnChannelName
サーバー接続チャネルの名前

QMgrName
宛先キュー・マネージャーの名前

TCP/IP 接続名を指定しないと、*ClientConnPtr* が NULL に設定されて MQCONNX が発行されます。 TCP/IP 接続名だけを指定し、サーバー接続チャネルを指定しないと (この反対は不可)、サンプルは SYSTEM.DEF.SVRCONN という名前を使用します。 宛先キュー・マネージャーを指定しないと、サンプルは与えられた TCP/IP 接続名を listen している側のキュー・マネージャーに接続します。

注 : パラメーターとして疑問符だけを入力するか、誤ったパラメーターを入力すると、プログラムの使用方法を説明するメッセージが表示されます。

コマンド行オプションなしでサンプルを実行すると、MQSERVER 環境変数の内容が接続情報を判別するために使用されます (この例では、MQSERVER が SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com に設定されます。) 以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

以下のように、このサンプルを実行し、TCP/IP 接続名とサーバー接続チャネル名のみを指定し、宛先キュー・マネージャー名を指定しないと、次のようになります。

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

デフォルトのキュー・マネージャー名が使用され、以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to the default queue manager
using the server connection channel SYSTEM.ADMIN.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

以下のように、このサンプルを実行し、TCP/IP 接続名と宛先キュー・マネージャー名を指定すると、次のようになります。

```
amqscnxc -x machine.site.company.com MACHINE
```

以下のような出力が表示されます。

```
Sample AMQSCNXC start
Connecting to queue manager MACHINE
using the server connection channel SYSTEM.DEF.SVRCONN
on connection name machine.site.company.com.
Connection established to queue manager MACHINE

Sample AMQSCNXC end
```

データ変換サンプル・プログラム

データ変換サンプル・プログラムは、データ変換出口ルーチンのスケルトンです。 データ変換サンプルの設計について説明します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

データ変換サンプルの設計

各データ変換出口ルーチンは、1 つの指定されたメッセージ形式を変換します。 このルーチンのスケルトンは、データ変換出口生成ユーティリティー・プログラムによって生成されたコードのラッパーとして設計されています。

このユーティリティーは、データ構造体ごとに 1 つのコード・フラグメントを生成します。 このような構造体がいくつか集まって 1 つの形式を形成しています。 したがって、複数のコード・フラグメントがこのスケルトンに付加されて、形式全体にわたってデータ変換を実行するためのルーチンが生成されます。

続いて、このプログラムは、変換の成否を検査し、呼び出し側に必要な値を戻します。

データベース調整サンプル

ここでは以下の 2 つのサンプルが提供されます。これらは WebSphere MQ が WebSphere MQ の更新分とデータベースの更新分の双方を、同一の作業単位内でどのように調整できるかを示します。

サンプルは、以下のとおりです。

1. AMQSXAS0 (C の場合) または AMQ0XAS0 (COBOL の場合)。これは、WebSphere MQ 作業単位内の単一データベースを更新します。
2. AMQSXAG0 (C の場合) または AMQ0XAG0 (COBOL の場合)、AMQSXAB0 (C の場合) または AMQ0XAB0 (COBOL の場合)、および AMQSXAF0 (C の場合) または AMQ0XAF0 (COBOL の場合)。これらは共に WebSphere MQ 作業単位内における 2 つのデータベースを更新し、複数のデータベースにアクセスする方法を示します。 これらのサンプルは MQBEGIN 呼び出し、混合 SQL および WebSphere MQ 呼び出しの使用方法、さらにデータベースに接続する時間と場所を示すために提供されます。

119 ページの図 18 には、提供されたサンプルを使用してデータベースを更新する方法を示します。

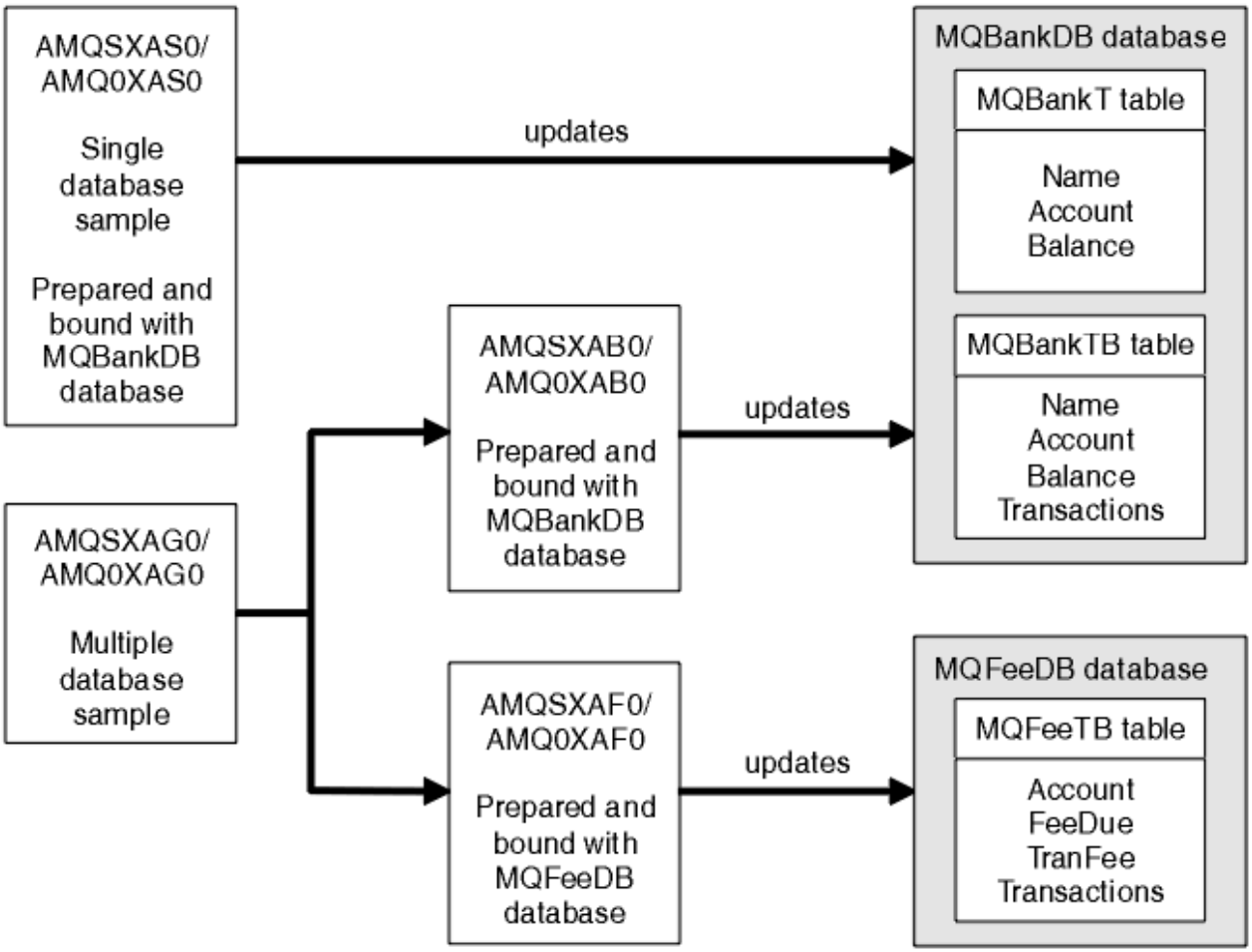


図 *18.* データベース調整サンプル

プログラムは (同期点で) キューからメッセージを読み取り、次にメッセージ内の情報を使用してデータベースから関連のある情報を得ると、これを更新します。 すると、データベースの新規状況が出力されます。

プログラム・ロジックは次のとおりです。

1. プログラム実引数から入力キューの名前を使用します。
2. MQCONN を使用して、デフォルトのキュー・マネージャー (または C で与えられる名前) に接続します。
3. 障害のない間に、MQOPEN を使用して入力のためにキューをオープンします。
4. MQBEGIN を使用して作業単位を開始します。

5. MQGET を使用して、同期点でキューから次のメッセージを読み取ります。
6. データベースから情報を読み取ります。
7. データベースの情報を更新します。
8. MQCMIT を使用して変更をコミットします。
9. 更新された情報を出力します (利用できるメッセージがない場合は障害として数え、ループを終了します)。
10. MQCLOSE を使用してキューをクローズします。
11. MQDISC を使用してキューから切断します。

サンプルは SQL カーソルを使用するので、データベースから読み取った内容 (つまり、複数インスタンス) は、メッセージが処理されている間ロックされます。したがって、これらのプログラムの複数インスタンスは同時に実行可能です。 カーソルは明示的にオープンされますが、MQCMIT 呼び出しによって暗黙的にクローズされます。

単一データベースのサンプル (AMQSXAS0 または AMQ0XAS0) には SQL CONNECT ステートメントはなく、データベースへの接続は MQBEGIN 呼び出しで WebSphere MQ によって暗黙的に行われます。 複数データベースのサンプル (AMQSXAG0 または AMQ0XAG0、AMQSXAB0 または AMQ0XAB0、および AMQSXAF0 または AMQ0XAF0) には SQL CONNECT ステートメントがあります。これは、データベース製品には活動状態の接続を 1 つしか許可しないものもあるからです。 上記の内容が使用するデータベース製品では問題にならない場合、または複数のデータベース製品の中の 1 つのデータベースにアクセスしている場合は、SQL CONNECT ステートメントは削除できます。

サンプルは IBM DB2 データベース製品で作成されているので、他のデータベース製品で作業するには、多少の変更が必要になる可能性があります。

SQL エラー検査は、DB2 で提供される UTIL.C および CHECKERR.CBL のルーチンを使用しています。 これらをコンパイルするか、またはコンパイルおよびリンクの前にこれらを置き換える必要があります。

注 : SQL エラー検査に Micro Focus COBOL のソース CHECKERR.MFC を使用している場合は、AMQ0XAS0 が正しくリンクできるようにプログラム ID を大文字、つまり CHECKERR に変更してください。

データベースと表の作成

サンプルをコンパイルする前に、データベースと表を作成します。

データベースを作成するには、ご使用のデータベース製品にとって標準的な方法を使用します。例えば、次のように入力します。

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

SQL ステートメントを使用して次のように表を作成します。

C の場合

```
EXEC SQL CREATE TABLE MQBankT(Name         VARCHAR(40) NOT NULL,
                              Account      INTEGER     NOT NULL,
                              Balance      INTEGER     NOT NULL,
                              PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQBankTB(Name         VARCHAR(40) NOT NULL,
                               Account      INTEGER     NOT NULL,
                               Balance      INTEGER     NOT NULL,
                               Transactions INTEGER,
                               PRIMARY KEY (Account));

EXEC SQL CREATE TABLE MQFeeTB(Account      INTEGER     NOT NULL,
                              FeeDue       INTEGER     NOT NULL,
                              TranFee      INTEGER     NOT NULL,
                              Transactions INTEGER,
                              PRIMARY KEY (Account));
```

COBOL の場合

```
 EXEC SQL CREATE TABLE
   MQBankT(Name      VARCHAR(40) NOT NULL,
           Account   INTEGER     NOT NULL,
           Balance   INTEGER     NOT NULL,
           PRIMARY KEY (Account))
    END-EXEC.

EXEC SQL CREATE TABLE
  MQBankTB(Name      VARCHAR(40) NOT NULL,
           Account   INTEGER     NOT NULL,
           Balance   INTEGER     NOT NULL,
           Transactions INTEGER,
           PRIMARY KEY (Account))
  END-EXEC.

EXEC SQL CREATE TABLE
  MQFeeTB(Account      INTEGER     NOT NULL,
          FeeDue       INTEGER     NOT NULL,
          TranFee      INTEGER     NOT NULL,
          Transactions INTEGER,
          PRIMARY KEY (Account))
  END-EXEC.
```

SQL ステートメントを使用して次のように表にデータを入力します。

```
EXEC SQL INSERT INTO MQBankT VALUES ('Mr Fred Bloggs',1,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Mrs S Smith',2,0);
EXEC SQL INSERT INTO MQBankT VALUES ('Ms Mary Brown',3,0);
⋮
EXEC SQL INSERT INTO MQBankTB VALUES ('Mr Fred Bloggs',1,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Mrs S Smith',2,0,0);
EXEC SQL INSERT INTO MQBankTB VALUES ('Ms Mary Brown',3,0,0);
⋮
EXEC SQL INSERT INTO MQFeeTB VALUES (1,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (2,0,50,0);
EXEC SQL INSERT INTO MQFeeTB VALUES (3,0,50,0);
⋮
```

注 : COBOL の場合、同じ SQL ステートメントを使用しますが、各行の終わりに END_EXEC を追加します。

サンプルのプリコンパイル、コンパイル、およびリンク

C および COBOL でのサンプルのプリコンパイル、コンパイル、およびリンクについて説明します。

.C または .CBL ファイルを作るには、.SQC ファイル (C の場合) および .SQB ファイル (COBOL の場合) をプリコンパイルし、該当するデータベースにバインドします。 これを行うには、ご使用のデータベース製品の標準的な方法を使用します。

C の場合のプリコンパイル

```
db2 connect to MQBankDB
db2 prep AMQSXAS0.SQC
db2 connect reset

db2 connect to MQBankDB
db2 prep AMQSXAB0.SQC
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQSXAF0.SQC
db2 connect reset
```

COBOL の場合のプリコンパイル

```
db2 connect to MQBankDB
db2 prep AMQ0XAS0.SQB bindfile target ibmcob
db2 bind AMQ0XAS0.BND
db2 connect reset
```

```
db2 connect to MQBankDB
db2 prep AMQ0XAB0.SQB bindfile target ibmcob
db2 bind AMQ0XAB0.BND
db2 connect reset

db2 connect to MQFeeDB
db2 prep AMQ0XAF0.SQB bindfile target ibmcob
db2 bind AMQ0XAF0.BND
db2 connect reset
```

コンパイルとリンク

以下のサンプル・コマンドは、シンボル *<DB2TOP>* および *MQ_INSTALLATION_PATH* を使用します。 *<DB2TOP>* は、 DB2 製品のインストール・ディレクトリーを表します。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。

- AIX の場合、ディレクトリー・パスは以下のとおりです。

```
/usr/lpp/db2_05_00
```

- HP-UX および Solaris の場合、ディレクトリー・パスは以下のとおりです。

```
/opt/IBMdb2/V5.0
```

- Windows システムの場合、ディレクトリー・パスは、製品のインストール時に選択したパスによって変わります。 デフォルト設定を選択した場合のパスは以下のとおりです。

```
c:\sqllib
```

注 : Windows システムでリンク・コマンドを実行する場合は、その前に DB2 ライブラリーと WebSphere MQ ライブラリーへのパスが LIB 環境変数に含まれていることを確認してください。

以下のファイルを一時ディレクトリーにコピーしてください。

- WebSphere MQ インストール済み環境からの amqsxag0.c ファイル

  注 : このファイルは、以下のディレクトリーにあります。

  – UNIX and Linux システムの場合:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows システムの場合:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- .sqc ソース・ファイル、 amqsxas0.sqc, amqsxaf0.sqc 、および amqsxab0.sqc のプリコンパイルによって取得した .c ファイル
- DB2 インストール済み環境からのファイル util.c および util.h 。

  注 : これらのファイルは、以下のディレクトリーにあります。

  ```
  <DB2TOP>/samples/c
  ```

ご使用のプラットフォームに応じて、以下のコンパイラー・コマンドを使用し、.c ファイルごとにオブジェクト・ファイルを構築してください。

- AIX

```
xlc_r -IMQ_INSTALLATION_PATH/inc -I
```

```
<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- HP-UX

```
cc -Aa +z -IMQ_INSTALLATION_PATH/inc -I
<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- Solaris

```
cc -Aa -KPIC -mt -IMQ_INSTALLATION_PATH
/inc -I<DB2TOP>/include -c -o
 <FILENAME>.o <FILENAME>.c
```

- Windows システム

```
cl /c /IMQ_INSTALLATION_PATH\tools\c\include /I
<DB2TOP>\include
 <FILENAME>.c
```

ご使用のプラットフォームに応じて、以下のリンク・コマンドを使用し、amqsxag0 実行可能ファイルを構築してください。

- AIX

```
xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib
 -lmqm util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- HP-UX Revision 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread -lcl
 /lib/crt0.o util.o amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Solaris

```
cc -mt -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib
 -lmqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxaf0.o amqsxab0.o amqsxag0.o -o amqsxag0
```

- Windows システム

```
link util.obj amqsxaf0.obj amqsxab0.obj amqsxag0.obj mqm.lib db2api.lib
 /out:amqsxag0.exe
```

ご使用のプラットフォームに応じて、以下のコンパイル・コマンドとリンク・コマンドを使用し、amqsxas0 実行可能ファイルを構築してください。

- AIX

```
xlc_r -H512 -T512 -L<DB2TOP>/lib -ldb2
 -LMQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- HP-UX Revision 11i

```
ld -E -L<DB2TOP>/lib -ldb2 -LMQ_INSTALLATION_PATH/lib -lmqm -lc -lpthread
 -lcl /lib/crt0.o util.o amqsxas0.o -o amqsxas0
```

- Solaris

```
cc -mt -L<DB2TOP>/lib -ldb2-LMQ_INSTALLATION_PATH/lib
```

```
 -lqm -lthread -lsocket -lc -lnsl -ldl util.o
  amqsxas0.o -o amqsxas0
```

- Windows システム

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

追加情報

AIX や HP-UX 上で作業中に Oracle にアクセスしたい場合、xlc_r コンパイラーを使用して libmqm_r.a にリンクしてください。

サンプルの実行

この情報を使用して、C および COBOL でデータベース調整サンプルを実行する前にキュー・マネージャーを構成する方法について学習します。

サンプルを実行するには、まずキュー・マネージャーをご使用中のデータベース製品で構成します。 これを行う方法については、42 ページの『シナリオ 1: キュー・マネージャーが調整を行う』を参照してください。

以下のタイトルでは、C および COBOL でのサンプルの実行方法について説明しています。

- 124 ページの『C サンプル』
- 125 ページの『COBOL サンプル』

C サンプル

メッセージはキューから読み取るため、次の形式をとることが必要です。

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT を使用してキューにメッセージを書き込むことができます。

データベース調整サンプルは、次の 2 つのパラメーターを使用します。

1. キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

singDBQ というキューと共に singDBQM という単一データベースのサンプル用にキュー・マネージャーを作成して構成したと想定すると、Fred Bloggs 氏のアカウントを 50 ずつ増やすには、次のようにします。

```
  AMQSPUT singDBQ singDBQM
```

次に、以下のメッセージを入力してください。

```
  UPDATE Balance change=50 WHERE Account=1
```

キューには複数のメッセージを書き込むことができます。

```
  AMQSXAS0 singDBQ singDBQM
```

そこで Fred Bloggs 氏のアカウントの更新状況が出力されます。

multDBQ というキューと共に multDBQM という複数データベースのサンプル用にキュー・マネージャーを作成して構成したと想定すると、Mary Brown さんのアカウントを 75 ずつ減らすには、次のようにします。

```
  AMQSPUT multDBQ multDBQM
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=-75 WHERE Account=3
```

キューには複数のメッセージを書き込むことができます。

```
AMQSXAG0 multDBQ multDBQM
```

そこで Mary Brown さんのアカウントの更新状況が出力されます。

COBOL サンプル

メッセージはキューから読み取るため、次の形式をとることが必要です。

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

分かりやすくするため、`Balance change` は符号付きの 8 桁の数に、`Account` は 8 桁の数にしてください。

AMQSPUT のサンプルを使用してキューにメッセージを書き込むことができます。

サンプルはパラメーターを使用せず、デフォルトのキュー・マネージャーを使用します。 キュー・マネージャーは、サンプルのうち常に 1 つだけを実行できるように構成できます。 singDBQ というキューと共に単一データベースのサンプル用にデフォルトのキュー・マネージャーを構成したと想定すると、Fred Bloggs 氏のアカウントを 50 ずつ増やすには、次のようにします。

```
AMQSPUT singDBQ
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

キューには複数のメッセージを書き込むことができます。

```
AMQ0XAS0
```

キューの名前を次のとおり入力してください。

```
singDBQ
```

そこで Fred Bloggs 氏のアカウントの更新状況が出力されます。

multDBQ というキューと共に複数データベースのサンプル用にデフォルトのキュー・マネージャーを構成したと想定すると、Mary Brown さんのアカウントを 75 ずつ減らすには、次のようにします。

```
AMQSPUT multDBQ
```

次に、以下のメッセージを入力してください。

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

キューには複数のメッセージを書き込むことができます。

```
AMQ0XAG0
```

キューの名前を次のとおり入力してください。

```
multDBQ
```

そこで Mary Brown さんのアカウントの更新状況が出力されます。

送達不能キュー・ハンドラーのサンプル

サンプルの送達不能キュー・ハンドラーが提供されています。実行可能なバージョンの名前は amqsdlq です。 RUNMQDLQ とは異なる送達不能キュー・ハンドラーを使用する場合、サンプルのソースをベースとして使用することができます。

サンプルは、この製品内で提供される送達不能ハンドラーと同様ですが、トレースとエラー報告は異なります。 次の 2 つの環境変数が利用できます。

ODQ_TRACE
YES または yes を設定すると、トレースがオンになります。

ODQ_MSG
エラー・メッセージおよび通知メッセージを含むファイルの名前を設定します。 amqsdlq.msg という名前のファイルが提供されています。

プラットフォームに応じ、**export** コマンドまたは **set** コマンドを使用して、これらの変数をご使用の環境で有効にする必要があります。また、**unset** コマンドを使用してトレースをオフにします。

エラー・メッセージ・ファイル amqsdlq.msg を各自の条件に合わせて修正することができます。 このサンプルは、メッセージを WebSphere MQ エラー・ログ・ファイル **ではなく** stdout に書き込みます。

送達不能ハンドラーの機能とその実行方法については、管理またはご使用のプラットフォームの「システム管理ガイド」を参照してください。

配布リスト・サンプル・プログラム

配布リスト・サンプル amqsptl0 によって、複数のメッセージ・キューにメッセージを書き込む例が得られます。 これは MQPUT サンプルである amqsput0 に基づいています。

配布リスト・サンプル amqsptl0 の実行

配布リスト・サンプルは、書き込みサンプルと同様の方法で実行されます。

これは以下のパラメーターをとります。

- キューの名前
- キュー・マネージャーの名前

これらの値は組にして入力します。 以下に例を示します。

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

キューは MQOPEN によってオープンされ、メッセージは MQPUT によってキューに書き込まれます。 キュー名またはキュー・マネージャー名が認識されない場合は、理由コードが戻ります。

メッセージがキュー・マネージャー間を流れるように、チャネルを定義してください。 サンプル・プログラムでは、チャネル定義まで行いません。

配布リスト・サンプルの設計

書き込みメッセージ・レコード (MQPMR) は、宛先ごとにメッセージの属性を指定します。 サンプルが *MsgId* および *CorrelId* に値を設定すると、これらは MQMD 構造体で指定された値を指定変更します。

MQPMO 構造体の *PutMsgRecFields* フィールドは、次のように MQPMR にあるフィールドを示します。

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

次に、サンプルは応答レコードとオブジェクト・レコードを割り当てます。 オブジェクト・レコード (MQOR) には、1 組以上の名前かつ偶数の名前が必要です。つまり、*ObjectName* と *ObjectQMgrName* です。

次の段階では、MQCONN を使用したキュー・マネージャーへの接続が必要になります。 サンプルは MQOR 内の最初のキューに関連するキュー・マネージャーへの接続を試みます。これが失敗すると、サンプルはオブジェクト・レコードをくまなく調べます。 キュー・マネージャーに接続できない場合はその旨を通知し、このサンプル・プログラムは実行を終了します。

宛先キューは MQOPEN によってオープンされ、メッセージは MQPUT によってこれらのキューに書き込まれます。 問題や障害が発生した場合は、応答レコード (MQRR) に報告されます。

最後に、宛先キューは MQCLOSE によってクローズされ、プログラムは MQDISC によってキュー・マネージャーから切断されます。 *CompCode* および *Reason* を示す呼び出しごとに、同じ応答レコードが使用されます。

エコー・サンプル・プログラム

Echo サンプル・プログラムは、メッセージ・キューから応答キューへのメッセージをエコー出力します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

これらのプログラムは、起動されたプログラムとして実行することを意図したものです。

UNIX、Linux、および Windows システムでは、その入力は、宛先キューおよびそのキュー・マネージャーの名前を含む MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

正しく定義を設定したら、まず、1 つのジョブで AMQSERV4 を始動し、続いて別のジョブで AMQSREQ4 を始動します。 AMQSERV4 の代わりに AMQSTRG4 を使用しても構いませんが、ジョブの送信が遅れる可能性があるため、処理の流れを追うのが難しくなります。

SYSTEM.SAMPLE.ECHO キューにメッセージを送信するには、要求サンプル・プログラムを使用してください。 エコー・サンプル・プログラムは、要求メッセージ内のデータを含む応答メッセージを、要求メッセージで指定した応答先キューに送信します。

エコー・サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、このキューを要求キュー と呼びます。) このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG、MQGMO_CONVERT、および MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各入力行ごとに、プログラムはテキストをバッファーに読み込み、MQPUT1 呼び出しを使用して、その行のテキストが入った要求メッセージを応答先キューに入れます。

MQGET 呼び出しが失敗すると、このプログラムは、メッセージ記述子の *Feedback* フィールドに、MQGET によって戻される理由コードを設定して、報告メッセージを応答先キューに入れます。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

読み取りサンプル・プログラム

読み取りサンプル・プログラムは、MQGET 呼び出しを使用してキューからメッセージを取得します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

読み取りサンプル・プログラムの設計

このプログラムは、MQOO_INPUT_AS_Q_DEF オプション付きの MQOPEN 呼び出しを使用して、宛先キューをオープンします。 プログラムは、キューをオープンできない場合に、MQOPEN 呼び出しによって戻される理由コードを含むエラー・メッセージを表示します。

キュー上の各メッセージごとに、プログラムは MQGET 呼び出しを使用して、キューからメッセージを除去し、そのメッセージに含まれるデータを表示します。 MQGET 呼び出しは、MQGMO_WAIT オプションを使用して、*WaitInterval* を 15 秒に指定します。したがって、キュー上にメッセージがないと、プログラムは指定された時間まで待機します。 メッセージがこの時間内に到着しない場合、呼び出しは失敗し、MQRC_NO_MSG_AVAILABLE 理由コードを戻します。

このプログラムは、それぞれの MQGET 呼び出しのあとに MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアすべき方法を示します。これは、この呼び出しによって、これらのフィールドに、このプログラムが取り出すメッセージに含まれる値が設定されるからです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

この MQGET 呼び出しでは、固定サイズのバッファーを指定します。 メッセージがこのバッファーよりも大きい場合には、呼び出しは失敗し、プログラムが停止します。

このプログラムは、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE 理由コードを戻すか、あるいは MQGET 呼び出しが失敗するまで、処理を続行します。 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

続いて、プログラムは、MQCLOSE 呼び出しを使用してキューをクローズします。

amqsget および *amqsgetc* のサンプルの実行

これらのプログラムはそれぞれ次の 2 つのパラメーターをとります。

1. ソース・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、amqsget はデフォルトのキュー・マネージャーに接続され、amqsgetc は環境変数またはクライアント・チャネル定義ファイルによって識別されるキュー・マネージャーに接続されます。

これらのプログラムを実行するには、それぞれ次のように入力します。

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

この例では、`myqueue` がプログラムがメッセージを取得するキューの名前で、`qmanagername` が `myqueue` を所有するキュー・マネージャーとなります。

`qmanagername` を省略すると、プログラムはデフォルト値をとります。MQI クライアントの場合には、環境変数またはクライアント・チャネル定義ファイルによって識別されるキュー・マネージャーをとります。

高可用性のサンプル・プログラム

amqsghac、**amqsphac**、および **amqsmhac** の各高可用性サンプル・プログラムは、自動クライアント再接続を使用して、キュー・マネージャーの障害後のリカバリーを例示します。 **amqsfhac** では、ネットワーク・ストレージを使用するキュー・マネージャーが、障害後もデータ保全性を維持していることを検査します。

amqsghac、**amqsphac**、および **amqsmhac** の各プログラムは、コマンド行から開始され、組み合わせて使用することにより、複数インスタンス・キュー・マネージャーのいずれかのインスタンスに障害が発生した後の再接続を実例によって確認できます。

また、**amqsghac**、**amqsphac**、および **amqsmhac** の各サンプルを使用して、単一インスタンス・キュー・マネージャー (通常、キュー・マネージャー・グループに構成されています) へのクライアント再接続の例を確認することもできます。

サンプルを単純化して構成しやすくするために、ここでは、開始され、停止された後に、再開される単一インスタンス・キュー・マネージャーに再接続するサンプル・プログラムが示されています。131 ページの『キュー・マネージャーのセットアップおよび制御』を参照してください。

amqsfhac を **amqmfsck** と並行して使用して、ファイル・システムの整合性を検査します。 詳しくは、**amqmfsck** (ファイル・システム検査) および 共有ファイル・システムの動作の検証 を参照してください。

amqsphac *queueName* [*qMgrName*]

- **amqsphac** は IBM WebSphere MQ MQI client・アプリケーションです。 これは一連のメッセージを、各メッセージ間に 2 秒の遅延時間を設定してキューに書き込み、イベント・ハンドラーに送信されたイベントを表示します。
- メッセージをキューに書き込むために同期点は使用されません。
- 同一のキュー・マネージャー・グループに属する任意のキュー・マネージャーに再接続できます。

amqsghac *queueName* [*qMgrName*]

- **amqsghac** は IBM WebSphere MQ MQI client・アプリケーションです。 これはキューからメッセージを取得し、イベント・ハンドラーに送信されたイベントを表示します。
- メッセージをキューから取得するために同期点は使用されません。
- 同一のキュー・マネージャー・グループに属する任意のキュー・マネージャーに再接続できます。

amqsmhac -s *sourceQueueName* **-t** *targetQueueName* [**-m** *qMgrName*] [**-w** *waitInterval*]

- **amqsmhac** は IBM WebSphere MQ MQI client・アプリケーションです。 これはあるキューから別のキューにメッセージをコピーします。デフォルトでは、プログラムの終了前に受け取った最後のメッセージから 15 分の待機間隔が設定されています。
- メッセージは同期点内でコピーされます。
- 再接続は、同一のキュー・マネージャーに対してのみ設定できます。

amqsfhac *QueueManagerName QueueName SideQueueName InTransactionCount RepeatCount* (**0|1|2**)

- **amqsfhac** は IBM WebSphere MQ MQI client・アプリケーションです。 これは、ネットワーク・ストレージを使用する IBM WebSphere MQ 複数インスタンス・キュー・マネージャー (NAS やクラスター・ファイル・システムなど) が、データ保全性を維持していることを確認します。 「共有ファイル・システムの動作の検証」の手順に従って **amqsfhac** を実行します。
- これは、*QueueManagerName* に接続するときに、MQCNO_RECONNECT_Q_MGR オプションを使用します。 さらに、キュー・マネージャーがフェイルオーバーするときには、自動的に再接続します。
- これは *InTransactionCount***RepeatCount* 持続メッセージを *QueueName* に書き込みます。この間に、キュー・マネージャーを何度でもフェイルオーバーさせることができます。 そのたびに **amqsfhac** はキュー・マネージャーに再接続し、続行します。 テストの目的は、メッセージが失われていないことを確認することです。
- *InTransactionCount* メッセージは各トランザクション内に書き込まれます。 トランザクションは、*RepeatCount* の回数だけ繰り返されます。 トランザクション内で障害が発生した場合、 **amqsfhac** がキュー・マネージャーに再接続すると、 **amqsfhac** はロールバックし、トランザクションを再実行依頼します。
- さらにこれはメッセージを *SideQueueName* に書き込みます。 また、*SideQueueName* を使用して、すべてのメッセージが正常にコミットされているか、または *QueueName* からロールバックされているかを検査します。 不整合を検出した場合は、エラー・メッセージを書き込みます。
- 最後のパラメーターを (0|1|2) に設定することで、**amqsfhac** からの出力トレースの量を変更します。

  0
  出力量は最小です。

  1
  出力量は中程度です。

2
出力量は最大です。

クライアント接続の構成

これらのサンプルを実行するには、クライアントと サーバーの接続チャネルを構成する必要があります。クライアント検証プロシージャーに、クライアントのテスト環境をセットアップする方法が示されています。 クライアントのインストールの検査を参照してください。

別の方法として、以下の例に提供されている構成を使用することもできます。

amqsghac、amqsphac、および amqsmhac の使用例

この例では、単一インスタンス・キュー・マネージャーを使用した再接続可能クライアントの例を示します。

メッセージは **amqsphac** によってキュー SOURCE に書き込まれ、**amqsmhac** によって TARGET に転送され、**amqsghac** によって TARGET から取り出されます。130 ページの図 19 を参照してください。

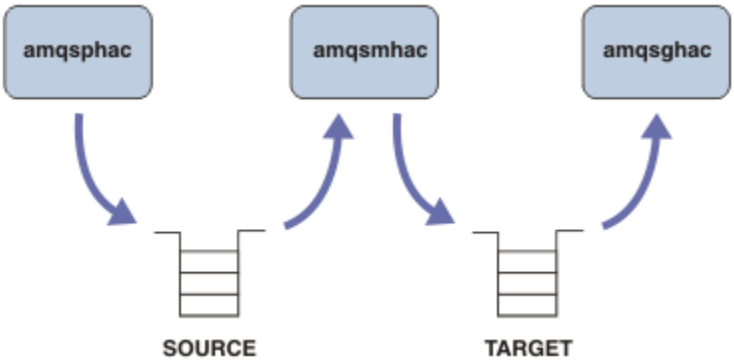


図 *19.* 再接続可能クライアントのサンプル

以下のステップに従って、このサンプルを実行してください。

1. 以下のコマンドを含む hasamples.tst ファイルを作成します。

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) +
       MCAUSER(MUSR_MQADMIN) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
       CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
ALTER  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) +
       PORT(2345)
START  LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
START  CHANNEL(CHANNEL1)
```

2. コマンド・プロンプトで以下のコマンドを入力する。
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. 環境変数 **MQCHLLIB** を AMQCLCHL.TAB クライアント・チャネル定義ファイルへのパス (例えば、 SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc) に設定します。
4. **MQCHLLIB** を設定した 3 つの新規ウィンドウを開く。例えば Windows では、前述のコマンド・プロンプトで **start** を 3 回入力し、各ウィンドウでそれぞれプログラムを実行します。 (131 ページの『キュー・マネージャーのセットアップおよび制御』 のステップ 131 ページの『5』 を参照してください。)
5. コマンド endmqm -r -p QM1 を入力してキュー・マネージャーを停止してから、クライアントが再接続できるようにします。

6. コマンド strmqm QM1 を入力して、キュー・マネージャーを再始動します。

Windows で **amqsghac**、**amqsphac**、および **amqsmhac** の各サンプルを実行したときの結果を、以下の例に示します。

キュー・マネージャーのセットアップおよび制御

1. キュー・マネージャーを作成します。

```
C:\>crtmqm QM1
WebSphere MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   後で **MQCHLLIB** 変数を設定するために、データ・ディレクトリーを覚えておいてください。

2. キュー・マネージャーを始動します。

```
C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

3. キューとチャネルを作成し、リスナー・ポートを変更して、リスナーとチャネルを開始する。

```
C:\>runmqsc QM1 < hasamples.tst
5724-H72 (C) Copyright IBM Corp. 1994, 2024. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: WebSphere MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: WebSphere MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: WebSphere MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: WebSphere MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: WebSphere MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start WebSphere MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start WebSphere MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. クライアント・チャネル・テーブルをクライアントに認識されるようにする。

   ステップ 131 ページの『1』 の **crtmqm** コマンドから返されたデータ・ディレクトリーを使用し、そのディレクトリーに @ipcc ディレクトリーを追加して、**MQCHLLIB** 変数を設定します。

```
C:\>SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. 他のウィンドウでサンプル・プログラムを開始する。

```
C:\>start amqsphac SOURCE QM1
C:\>start amqsmhac -s SOURCE -t TARGET -m QM1
C:\>start amqsghac TARGET QM1
```

6. キュー・マネージャーを終了してから、再始動する。

```
C:\>endmqm -r -p QM1
```

```
Waiting for queue manager 'QM1' to end.
WebSphere MQ queue manager 'QM1' ending.
WebSphere MQ queue manager 'QM1' ended.

C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
<Message 3>
message <Message 4>
message <Message 5>
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message <Message 3>
message <Message 4>
message <Message 5>
```

関連タスク

共有ファイル・システムの動作の検証

関連資料

amqmfsck (ファイル・システム検査)

照会サンプル・プログラム

照会サンプル・プログラムでは、MQINQ 呼び出しを使用して、一部のキュー属性について照会します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

これらのプログラムは、トリガーされるプログラムとして実行することを意図したものです。したがって、IBM i、Windows、UNIX and Linux システムでは、その入力は MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 この構造体には、照会される属性を持つ宛先キューの名前が入っています。 C バージョンもキュー・マネージャー名を使用します。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

トリガー・プロセスを処理するには、使用したい照会サンプル・プログラムがキュー SYSTEM.SAMPLE.INQ に到着したメッセージによって起動されているか確認してください。 これを行うには、使用したい照会サ

ンプル・プログラムの名前を、プロセス定義 SYSTEM.SAMPLE.INQPROCESS の *ApplicId* フィールドに指定します。 サンプル・キューのトリガー・タイプは FIRST です。ユーザーが要求サンプルを実行する前にメッセージが既にキュー上にある場合、照会サンプルはユーザーが送信したメッセージによって起動されません。

正しく定義を設定したら、以下のようにします。

- UNIX、Linux および Windows システムの場合、1 つのセッションで **runmqtrm** プログラムを開始後、別のセッションで amqsreq プログラムを開始します。

要求サンプル・プログラムを使用して、それぞれがキュー名のみを含む要求メッセージを、キュー SYSTEM.SAMPLE.INQ に送信します。 各要求メッセージごとに、照会サンプル・プログラムは、要求メッセージで指定したキューの情報を含む応答メッセージを送信します。 応答は、要求メッセージで指定した応答先キューに送信されます。

照会サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、このキューを要求キュー と呼びます。) このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各要求メッセージが要求キューから除去されたら、このプログラムは、そのデータに含まれるそのキュー (宛先キュー と呼びます) の名前を読み込み、MQOO_INQ オプション付きの MQOPEN 呼び出しを使用して、そのキューをオープンします。 次に、このプログラムは、MQINQ 呼び出しを使用して、宛先キューの *InhibitGet*、*CurrentQDepth*、および *OpenInputCount* 属性値を照会します。

MQINQ 呼び出しが正常に行われると、このプログラムは MQPUT1 呼び出しを使用して、応答メッセージを応答先キューに入れます。 このメッセージには、3 つの属性値が含まれます。

MQOPEN または MQINQ 呼び出しが異常終了すると、このプログラムは MQPUT1 呼び出しを使用して、報告メッセージを応答先キューに入れます。 この報告メッセージのメッセージ記述子の *Feedback* フィールドには、MQOPEN または MQINQ の失敗した呼び出しによって、戻される理由コードが入ります。

MQINQ 呼び出し後、このプログラムは MQCLOSE 呼び出しを使用して、宛先キューをクローズします。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

メッセージ処理サンプル・プログラムの照会プロパティー

AMQSIQMA は、メッセージ・ハンドルのプロパティーをメッセージ・キューから照会するためのサンプル C プログラムであり、MQINQMP API 呼び出しの使用例です。

このサンプルでは、メッセージ・ハンドルが作成され、それが MQGMO 構造体の MsgHandle フィールドに書き込まれます。 次に、1 つのメッセージが読み取られ、そのメッセージ・ハンドルに設定されているすべてのプロパティーが照会および表示されます。

```
C:\Program Files\IBM\WebSphere MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name <MyProp> value <MyValue>
message text <Hello world!>
Sample AMQSIQMA end
```

パブリッシュ/サブスクライブのサンプル・プログラム

パブリッシュ/サブスクライブのサンプル・プログラムは、WebSphere MQ でパブリッシュ機能とサブスクライブ機能を使用する方法を例示しています。

WebSphere MQ パブリッシュ/サブスクライブ・インターフェースに対してプログラムを作成する方法を示す C 言語のサンプル・プログラムが 3 つあります。 また、古いインターフェースを使用する C のサンプルのほか、Java のサンプルもあります。 Java のサンプルでは、com.ibm.mq.jar にある WebSphere MQ パブリッシュ/サブスクライブ・インターフェースと、com.ibm.mqjms にある JMS パブリッシュ/サブスクライブ・インターフェースを使用しています。 JMS のサンプルについては、このトピックでは説明しません。

C

C のサンプル・フォルダーからパブリッシャーのサンプル amqspub を見つけてください。 最初のパラメーターとして任意のトピック名を指定し、2 番目のパラメーターとしてキュー・マネージャーの名前 (省略可能) を指定して、このサンプルを実行します。 例えば、`amqspub mytopic QM3` と入力します。 amqspubc と呼ばれるクライアント・バージョンもあります。 このクライアント・バージョンを実行する場合は、まず、108 ページの『サンプル・プログラムの作成と実行』で詳細を確認してください。

パブリッシャーはデフォルトのキュー・マネージャーに接続し、「`target topic is mytopic`」という応答を出力します。 これ以降、このウィンドウに入力する行は、mytopic にパブリッシュされます。

同じディレクトリー内の別のコマンド・ウィンドウを開き、サブスクライバー・プログラム amqssub を実行し、同じトピック名とオプションのキュー・マネージャー名を指定して、それを実行します。 例えば、`amqssub mytopic QM3` です。

サブスクライバーは、`Calling MQGET : 30 seconds wait time` という出力で応答します。 これ以降、パブリッシャーに入力する行は、サブスクライバーに出力として表示されます。

もう 1 つ別のコマンド・ウィンドウでもう 1 つ別のサブスクライバーを始動し、2 つのサブスクライバーがパブリケーションを受け取る様子を見てください。

パラメーターに関する完全な説明とオプションの設定に関する情報は、サンプルのソース・コードを参照してください。 サブスクライバーのオプション・フィールドの値については、トピック Options (MQLONG) に説明があります。

もう 1 つのサブスクライバー・サンプル amqssbx では、コマンド・ライン・スイッチとして追加のサブスクリプション・オプションを指定できます。

`amqssbx -d mysub -t mytopic -k` と入力してサブスクライバーを起動すると、サブスクライバーが終了した後も保持される永続サブスクリプションを使用できます。

サブスクリプションをテストするために、パブリッシャーを使用して別のアイテムをパブリッシュしてみてください。 30 秒間待機して、サブスクライバーが終了するのを待ちます。 同じトピックでさらにいくつかアイテムをパブリッシュします。 サブスクライバーを再始動します。 サブスクライバーが実行中でなかった間にパブリッシュされた最終アイテムが、サブスクライバーが再始動した後すぐに表示されます。

C レガシー

キューに入れられたコマンドについて例を示す C のサンプルのセットが他にいくつかあります。 それらのサンプルのうちいくつかは、当初は MQOC Supportpac の一部として提供されていました。 これらのサンプルで示されている機能は、互換性の目的で完全にサポートされています。

キューに入れられたコマンドのインターフェースを使用することはお勧めできません。 パブリッシュ/サブスクライブの API よりずっと複雑で、キューに入れられたコマンドの複雑なプログラミングを行うことに見合うだけの機能上のメリットもありません。 しかし、キューに入れられたコマンドの方式が適していると考えられるケースとして、そのインターフェースを既に使用している場合や、従来と異なる MQSUB 呼び出しを作成するよりも、複雑なメッセージを作成して汎用的な MQPUT を呼び出す方が容易であるようなプログラミング環境を使用している場合があります。

追加のサンプルは、samples フォルダーの pubsub サブディレクトリーにあります。

サンプルには、135 ページの表 20 にリストされている 6 つのタイプがあります。

表 20. パブリッシュ/サブスクライブに関する既存のサンプル C プログラムのカテゴリー		
カテゴリー	**プログラム**	**コメント**
RFH1	amqssr1a.c amqspr1a.c	RFH1 形式のメッセージを使用して作成した、単純なパブリッシュ/サブスクライブの例。
RFH2	amqssr2a.c amqspr2a.c	RFH2 形式のメッセージを使用して作成した、単純なパブリッシュ/サブスクライブの例。
MQAI サンプル	amqsppca.c amqsspca.c	PCF コマンドと MQAI コマンド・インターフェースを使用して作成した、単純なパブリッシュ/サブスクライブの例。
RFH1 を使用する MA0C 結果サービス	amqsgama.c amqsresa.c	RFH1 ヘッダーを使用して作成した結果サービス 1. amqsgama.tst および amqsresa.tst で定義されるキューが必要です。 2. amqsresa は amqsgama より前に開始される必要があります
RFH2 を使用する MA0C 結果サービス	amqsgr2a.c amqsrr2a.c	RFH2 ヘッダーを使用して作成した結果サービス 1. amqsgama.tst および amqsresa.tst で定義されるキューが必要です。 2. amqsresa は amqsgama より前に開始される必要があります
ルーティング出口によるパブリッシュ/サブスクライブの例	amqspsra.c	パブリッシュ/サブスクライブするメッセージについて、ルーティング出口でキューまたはキュー・マネージャーの宛先を変更する方法を示しています。

JAVA

Java のサンプル MQPubSubApiSample.java では、パブリッシャーとサブスクライバーを 1 つのプログラムにまとめてあります。 ソース・ファイルとコンパイル済みのクラス・ファイルは、wmqjava サンプル・フォルダーにあります。

クライアント・モードで実行する場合は、まず、108 ページの『サンプル・プログラムの作成と実行』で詳細を確認してください。

構成済みの Java 環境がある場合は、Java コマンドを使用してコマンド・ラインからサンプルを実行します。 あるいは、Java プログラミング・ワークベンチを既にセットアップした WebSphere MQ エクスプローラーの Eclipse ワークスペースからサンプルを実行することもできます。

サンプル・プログラムを実行するために、プログラムのプロパティーの一部に変更が必要な場合があります。 それには、JVM に対してパラメーターを指定したり、ソースを編集したりします。

135 ページの『MQPubSubApiSample Java サンプルの実行』の 説明に、Eclipse ワークスペースからのサンプル実行方法が記述されています。

MQPubSubApiSample Java サンプルの実行

Eclipse プラットフォームの Java 開発ツールを使用して MQPubSubApiSample を実行する方法。

始める前に

Eclipse ワークベンチを開きます。 新しいワークスペース・ディレクトリーを作成し、 それを選択します。 ウェルカム・ウィンドウをクローズします。

108 ページの『サンプル・プログラムの作成と実行』のステップを行ってから、クライアントとして実行します。

このタスクについて

Java パブリッシュ/サブスクライブのサンプル・プログラムは、WebSphere MQ MQI クライアント Java プログラムです。 このサンプルは、ポート 1414 で listen するデフォルトのキュー・マネージャー を使用して、変更なしで実行します。 このタスクでは、 この単純なケースを説明し、さらに、このサンプルをさまざまな異なる WebSphere MQ 構成に適合させるには、パラメーターをどのように指定し、サンプルをどのように変更すればいいのかを、一般用語を使って説明します。 例では、Windows での実行を示します。 他のプラットフォームの場合はファイル・パスが 異なります。

手順

1. Java サンプル・プログラムのインポート
   a) ワークベンチで、 **ウィンドウ** > 「**パースペクティブを開く**」 > 「**その他**」 > **Java** をクリックし、「**OK**」をクリックします。
   b)「**パッケージ・エクスプローラー**」ビューに切り替えます。
   c)「**パッケージ・エクスプローラー** 」ビューの空白で右クリックします。 「**新規**」 > 「**Java プロジェクト**」をクリックします。
   d) **Project name** フィールドに、 MQ Java Samples と入力します。 「**次へ**」をクリックします。
   e)「 **Java Settings** 」パネルで、 「**ライブラリー**」 タブに切り替えます。
   f)「**外部 JAR の追加**」をクリックします。
   g) *MQ_INSTALLATION_PATH*\java\lib を参照します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ インストール・フォルダーであり、 com.ibm.mq.jar および com.ibm.mq.jmqi.jar を選択します。
   h)「**オープン**」 > 「**終了**」
   i)「**パッケージ・エクスプローラー** 」ビューで src を右クリックします。
   j) 選択 **インポート ...** > **一般** > **ファイル・システム** > **次へ** > **参照**... 次に、パス *MQ_INSTALLATION_PATH*\tools\wmqjava\samples を参照します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ インストール・ディレクトリーです。
   k)「**インポート**」パネル (137 ページの図 20) で、samples をクリックします (チェック・ボックスは選択しない)。
   l) MQPubSubApiSample.java を選択します。 **Into folder** フィールドには MQ Java Samples/src が含まれている必要があります。 「**完了**」をクリックする。

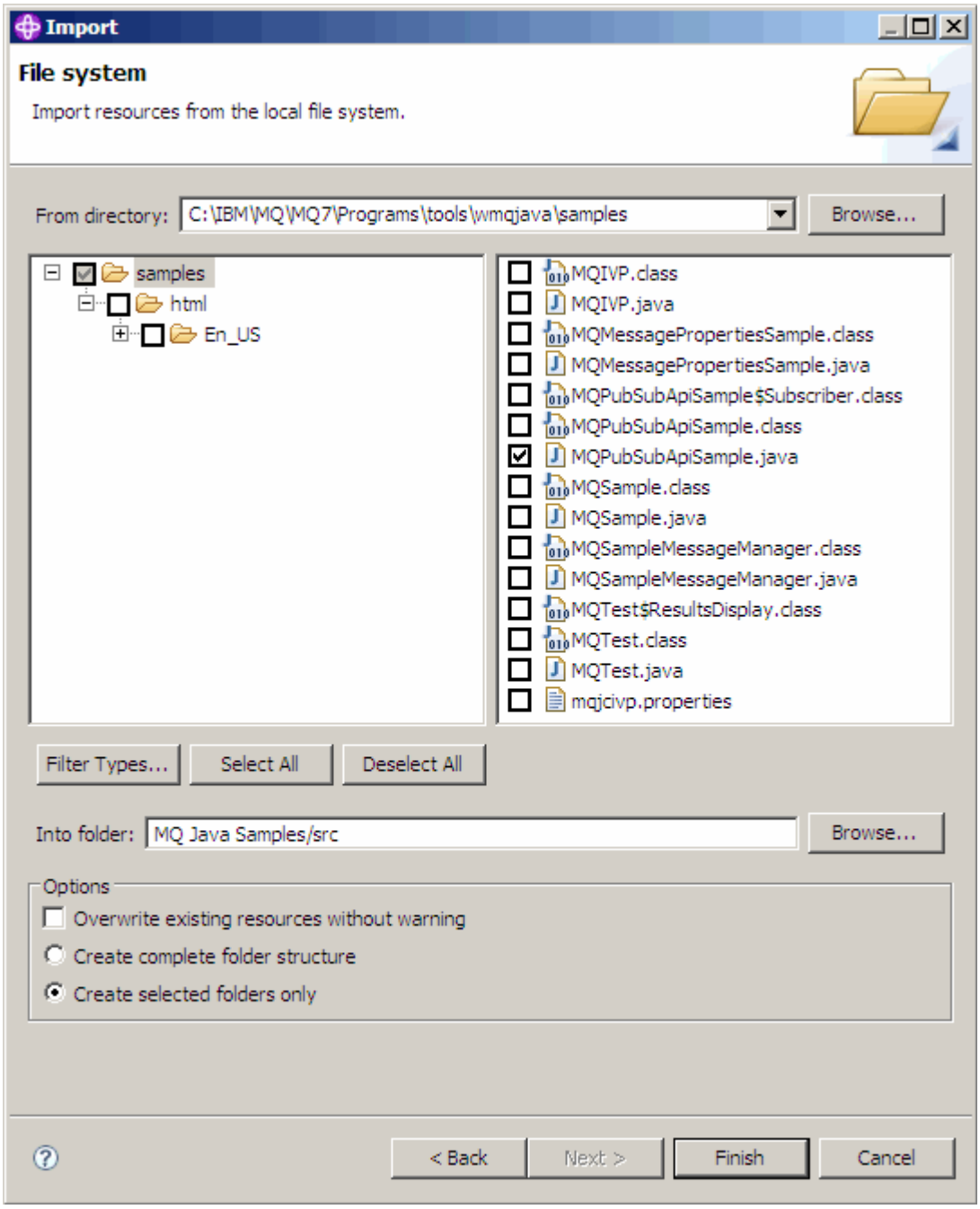


図 *20.* ファイル・システムのインポート

2. パブリッシュ/サブスクライブのサンプル・プログラムを実行します。

   このプログラムを実行する方法には、デフォルトのパラメーターを 変更する必要があるかどうかによって、2 とおりの方法があります。

   - 最初の選択肢は、変更を加えずにプログラムを実行することです。
     - ワークスペースのメインメニューで、`src` フォルダーを展開します。 **MQPubSubApiSample.javaRun-as > 1 を右クリックします。 Java アプリケーション**
   - 2 番目の選択肢は、パラメーター付きでプログラムを実行するか、または使用している環境に応じてソース・コードを変更してプログラムを実行することです。
     - `MQPubSubApiSample.java` をオープンし、`MQPubSubApiSample` コンストラクター を調べます。

– プログラムの属性を変更します。

-D JVM スイッチを使用するか、ソース・コードを編集してシステム・プロパティーのデフォルト値を指定することによって、以下の属性を変更できます。

- topicObject
- queueManagerName
- subscriberCount

以下の属性は、コンストラクターでのソース・コードの編集によってのみ 変更できます。

- ホスト名
- port
- channel

システム・プロパティーを設定するには、以下のように、アクセサーにデフォルト値を コーディングします。

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

あるいは、 以下の手順のように、-D オプションを使用して JVM にパラメーターを 指定します。

a. 設定したい System.Property のフルネームをコピーします。 例えば、com.ibm.mq.pubSubSample.queueManagerName です。

b. ワークスペースで、「**実行**」 >「**実行**」**ダイアログを開く**を右クリックします。「**アプリケーションの作成、管理、および実行**」 で「Java アプリケーション」をダブルクリックし、 「**(x) = 引数**」 タブをクリックします。

c.「**VM 引数:**」ペインで、-D を 入力し、System.property 名 com.ibm.mq.pubSubSample.queueManagerName を 貼り付け、その後に =QM3 を付けます。「**適用**」 >「**実行**」をクリックします。

d. コンマ区切りリストとして、またはペイン内の追加行 (コンマ区切りなし) として、 さらに引数を追加します。

例:-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3, -Dcom.ibm.mq.pubSubSample.subscriberCount=6。

パブリッシュ出口サンプル・プログラム

AMQSPSE0 は、サブスクライバーに送信される前にパブリケーションをインターセプトする出口のサンプル C プログラムです。 これにより、出口は、例えば、メッセージ・ヘッダー、ペイロードまたは宛先を変更したり、メッセージがサブスクライバーにパブリッシュされないようにすることができます。

サンプルを実行するには、以下のタスクを実行します。

1. キュー・マネージャーを以下のように構成します。
   - UNIX and Linux システムの場合、以下のようなスタンザを qm.ini ファイルに追加します。

     ```
     PublishSubscribe:
               PublishExitPath=<Module>
               PublishExitFunction=EntryPoint
     ```

     ここで、モジュールは *MQ_INSTALLATION_PATH*/samp/bin/amqspse です。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。 Windows の場合、これに相当する属性をレジストリーに設定します。
2. モジュールが WebSphere MQ にアクセス可能であること確認します。
3. キュー・マネージャーを再始動して、構成を取り入れます。
4. トレースされるアプリケーション・プロセスで、トレース・ファイルが書き込まれる場所を記述します。 以下に例を示します。

- UNIX and Linux システムの場合、ディレクトリー /var/mqm/trace が存在することを確認し、以下の環境変数をエクスポートします。

```
export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
```

- Windows の場合、ディレクトリー C:\temp が存在することを確認し、以下の環境変数を設定します。

```
set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
```

書き込みサンプル・プログラム

書き込みサンプル・プログラムは、MQPUT 呼び出しを使用して、メッセージをキューに入れます。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

書き込みサンプル・プログラムの設計

このプログラムは、MQOO_OUTPUT オプション付きの MQOPEN 呼び出しを使用して、メッセージを入れる宛先キューをオープンします。

キューをオープンできない場合、このプログラムは MQOPEN 呼び出しから戻される理由コードの入ったエラー・メッセージを出力します。 プログラムを簡潔に保つには、これ以降の MQI 呼び出しで、プログラムが多数のオプションに対してデフォルト値を使用するようにします。

各入力行ごとに、プログラムはテキストをバッファーに読み込み、MQPUT 呼び出しを使用して、その行のテキストが入ったデータグラム・メッセージを作成します。 プログラムは、入力が終了するか、MQPUT 呼び出しが異常終了するまで処理を続行します。 プログラムは、入力が終了すると、MQCLOSE 呼び出しを使用して、キューをクローズします。

書き込みサンプル・プログラムの実行

amqsput および amqsputc のサンプルの実行

これらのプログラムにはそれぞれに次の 2 つのパラメーターがあります。

1. 宛先キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、amqsput はデフォルトのキュー・マネージャーに接続され、amqsputc は環境変数またはクライアント・チャネル定義ファイルによって識別されるキュー・マネージャーに接続されます。 これらのプログラムを実行するには、それぞれ次のように入力します。

- `amqsput myqueue qmanagername`
- `amqsputc myqueue qmanagername`

この例では、myqueue がメッセージを入れるキューの名前で、qmanagername が myqueue を所有するキュー・マネージャーです。

amq0put サンプルの実行

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target queue
```

プログラムは、StdIn からの入力を受け付け、各入力行を宛先キューに付加します。 ブランク行は、これ以上データがないことを示します。

参照メッセージ・サンプル・プログラム

参照メッセージ・サンプルによって、大きなオブジェクトをあるノードから別のノードへ (通常は異なるシステム間で) 転送できます。この際、発信元ノードまたは宛先ノードのいずれの WebSphere MQ キューにもオブジェクトを格納する必要はありません。

参照メッセージをキューへ書き込み、メッセージ出口で受信し、キューから取り出す方法を示すために、一連のサンプル・プログラムが提供されます。 サンプル・プログラムは参照メッセージを使用してファイルを移動します。 データベースなどの他のオブジェクトを移動したい場合、またはセキュリティー検査を実行したい場合は、システムが提供するサンプルである amqsxrm に基づいて、独自の出口を定義してください。 以下の節では、参照メッセージ・サンプル・プログラムについて説明します。

どのバージョンの参照メッセージ出口サンプル・プログラムを使用するかは、チャネルが実行されているプラットフォームによって異なります。 すべてのプラットフォーム上で、送信側で amqsxrma を使用します。 受信側が WebSphere MQ 製品 (WebSphere MQ for IBM i の下で実行されている場合は、 amqsxrm4 を使用します。

参照メッセージ・サンプルの実行

この情報を使用して、参照メッセージ・サンプル・プログラムを実行する方法について学習します。

参照メッセージ・サンプルは、次のように実行します。

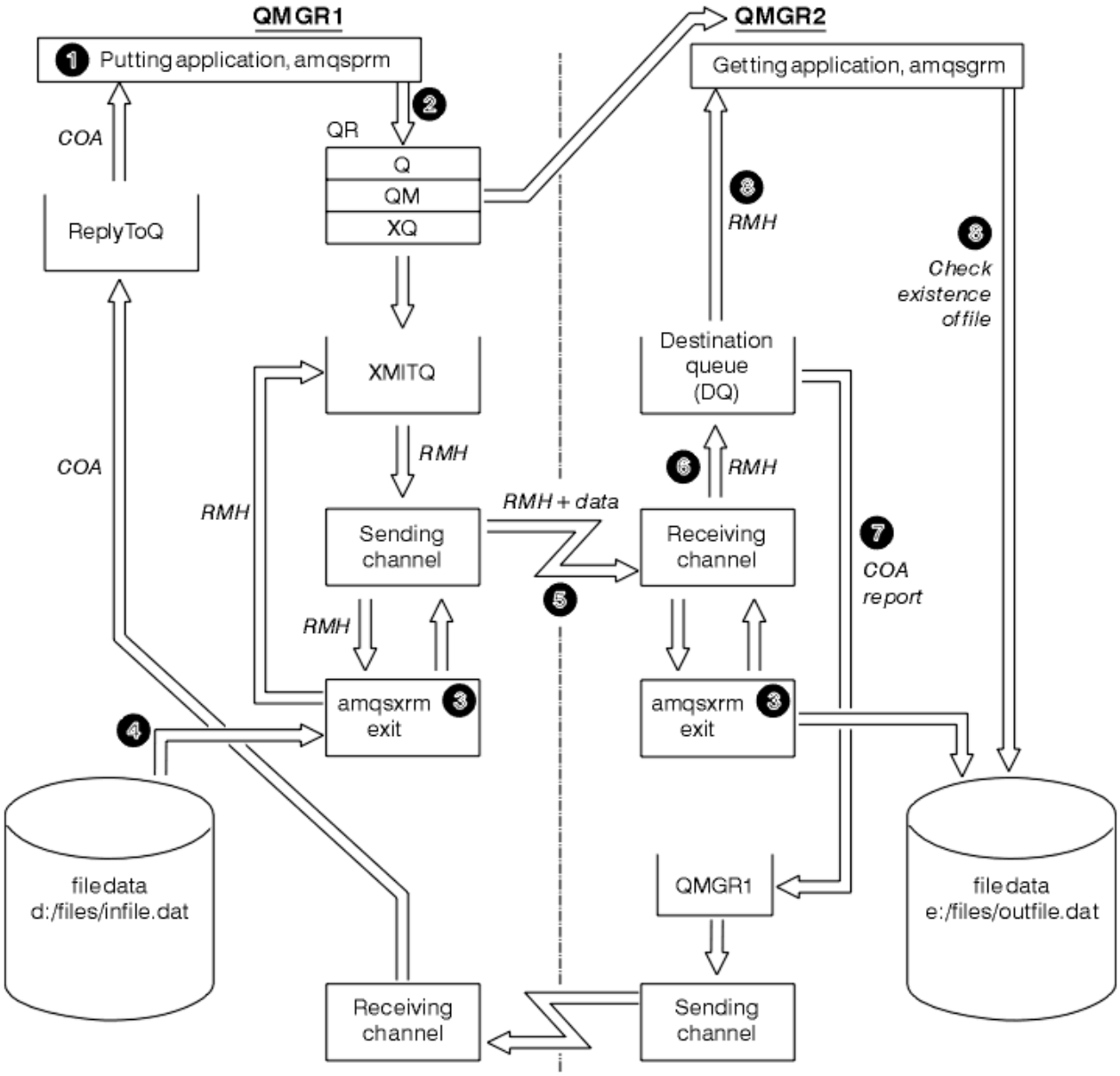


図 *21.* 参照メッセージ・サンプルの実行

1. 環境を設定してリスナー、チャネル、およびトリガー・モニターを始動し、チャネルおよびキューを定義します。

   ここでは、送信側のマシンを MACHINE1、そのマシン上のキュー・マネージャーを QMGR1 とし、受信側のマシンを MACHINE2、そのマシン上のキュー・マネージャーを QMGR2 とした場合を例に、参照メッセージ・サンプルの設定方法について説明します。

   注 : 以下に示す定義により、参照メッセージを作成して、オブジェクト・タイプ FLATFILE のファイルをキュー・マネージャー QMGR1 から QMGR2 へ送信し、AMQSPRM (IBM i の場合は AMQSPRMA) の呼び出しでの定義に従ってファイルを再作成することができます。 参照メッセージ (ファイル・データを含む) はチャネル CHL1 および伝送キュー XMITQ によって送信され、キュー DQ に入れられます。 例外報告および COA 報告はチャネル REPORT および伝送キュー QMGR1 によって QMGR1 に戻されます。

   参照メッセージを受信するアプリケーション (AMQSGRM) は、開始キュー INITQ およびプロセス PROC によって起動されます。 マシンのタイプおよび WebSphere MQ 製品のインストール先によっては、CONNAME フィールドが正しく設定されていること、および MSGEXIT フィールドにディレクトリー構造が設定されていることを確認してください。

MQSC の定義では、出口を定義するときに AIX 形式を使用していましたが、。 メッセージ・データ FLATFILE については大文字と小文字の区別があることに注意してください。大文字で指定しないとサンプルは動作しません。

マシン MACHINE1、キュー・マネージャー QMGR1 の場合

MQSC 構文

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

注 : キュー・マネージャー名を指定しないと、デフォルトのキュー・マネージャーがシステムに使用されます。

```
CRTMQMCHL  CHLNAME(CHL1) CHLTYPE(*SDR) MQMNAME(QMGR1) +
             REPLACE(*YES) TRPTYPE(*TCP) +
             CONNAME('MACHINE2(60501)') TMQNAME(XMITQ) +
             MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMQ    QNAME(XMITQ) QTYPE(*LCL) MQMNAME(QMGR1) +
             REPLACE(*YES) USAGE(*TMQ)

CRTMQMCHL  CHLNAME(REPORT) CHLTYPE(*RCVR) +
               MQMNAME(QMGR1) REPLACE(*YES) TRPTYPE(*TCP)

CRTMQMQ    QNAME(QR) QTYPE(*RMT) MQMNAME(QMGR1) +
             REPLACE(*YES) RMTQNAME(DQ) +
             RMTMQMNAME(QMGR2) TMQNAME(XMITQ)
```

マシン MACHINE2、キュー・マネージャー QMGR2 の場合

MQSC 構文

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

2. WebSphere MQ オブジェクトの作成が終わったら、以下の作業を行います。
   a. (該当するプラットフォームの場合には) 送信側および受信側のキュー・マネージャーでリスナーを始動します。
   b. チャネル CHL1 および REPORT を始動します。
   c. 受信側のキュー・マネージャーで、開始キュー INITQ のトリガー・モニターを始動します。
3. 以下のパラメーターを使用して、コマンド行から書き込み参照メッセージ・サンプル・プログラム AMQSPRM を呼び出します。

   -m　ローカル・キュー・マネージャーの名前。省略すると、デフォルトのキュー・マネージャーが設定されます。

   -i　ソース・ファイルの名前と格納場所

-o 宛先ファイルの名前と格納場所

-q キューの名前

-g -q パラメーターに指定されたキューのあるキュー・マネージャーの名前。

-t オブジェクト・タイプ

-w 待機間隔。つまり、受信側のキュー・マネージャーから出る例外報告および COA 報告の待機時間。

例えば、上記のように定義されたオブジェクトを用いてサンプルを使用するには、パラメーターを次のように指定します。

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

待機時間を長くすると、大きなファイルをネットワークで送信する場合、メッセージを書き込んでいるプログラムがタイムアウトになる前にファイルが送信されるよう余裕をとっておくことができます。

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

注 : UNIX and Linux プラットフォームの場合、宛先ファイルのディレクトリーを示すのに、円記号を 1 つではなく 2 つ (¥¥) 使用する必要があります。 したがって、**amqsprm** コマンドは次のように表示されます。

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

書き込み参照メッセージ・プログラムを実行すると、以下のことが行われます。

- 参照メッセージは、キュー・マネージャー QMGR1 でキュー QR に書き込まれます。
- ソース・ファイルとパスは `d:\files\infile.dat` で、コマンド例が発行されたシステム上に存在します。
- キュー QR がリモート・キューである場合、参照メッセージは別のシステム上の別のキュー・マネージャーに送信され、そこで名前とパス `e:\files\outfile.dat` を使用してファイルが作成されます。 このファイルの内容はソース・ファイルの内容と同じです。
- amqsprm は宛先キュー・マネージャーからの COA 報告を 30 秒間待機します。
- オブジェクト・タイプは `flatfile` なので、キュー QR からのメッセージを移動するのに使用されたチャネルは、このオブジェクト・タイプを *MsgData* フィールドに指定しなければなりません。

4. ユーザーのチャネルを定義する場合は、送信用と受信用の両方のメッセージ出口を amqsxrm に選択します。 これは、WebSphere MQ for Windows で次のように定義されます。

```
msgexit('pathname\amqsxrm.dll(MsgExit)')
```

これは、WebSphere MQ for AIX、WebSphere MQ for HP-UX、および WebSphere MQ for Solaris で次のように定義されます。

```
msgexit('pathname/amqsxrm(MsgExit)')
```

パス名を指定する場合は、完全な名前を指定してください。 パス名を省略すると、プログラムは `qm.ini` ファイルで指定されたパス (または、WebSphere MQ for Windows では、レジストリーに指定されたパス) にあると想定します。

5. チャネル出口は参照メッセージ・ヘッダーを読み取り、参照するファイルを検索します。
6. 次にチャネル出口はファイルをセグメント化できます。その後、チャネルにファイルをヘッダーと共に送ります。 WebSphere MQ for AIX、WebSphere MQ for HP-UX、および WebSphere MQ for Solaris では、サンプル・メッセージ出口がターゲット・ディレクトリーにファイルを作成できるように、そのデ

ィレクトリーのグループ所有者を「mqm」に変更してください。 さらに、ターゲット・ディレクトリーの許可を変更して、mqm グループ・メンバーがターゲット・ディレクトリーに書き込めるようにします。 ファイルのデータは WebSphere MQ キューには格納されません。

7. ファイルの最後のセグメントが受信メッセージ出口によって処理されると、参照メッセージは amqsprm で指定された宛先キューに書き込まれます。 このキューが起動されると (つまり、その定義が *Trigger*、*InitQ*、および *Process* のキュー属性を指定すると)、宛先キューの PROC パラメーターで指定されたプログラムも起動されます。 起動されるプログラムは *Process* 属性の *ApplId* フィールドで定義する必要があります。
8. 参照メッセージが宛先キュー (DQ) に到達すると、COA 報告が書き込みアプリケーション (amqsprm) に返送されます。
9. 読み取り参照メッセージ・サンプルである amqsgrm は、入力トリガー・メッセージで指定されたキューからメッセージを読み込み、ファイルの有無を確認します。

書き込み参照メッセージ・サンプル (*amqsprma.c*、*AMQSPRM4*) の設計

このトピックでは、書き込み参照メッセージ・サンプルの詳細について説明します。

このサンプルはファイルを参照する参照メッセージを作成し、そのメッセージを指定されたキューに書き込みます。

1. サンプルは MQCONN を使用してローカル・キュー・マネージャーと接続します。
2. 次にサンプルは MQOPEN を使用して、報告メッセージを受信するために使用されるモデル・キューをオープンします。
3. サンプルはファイルを移動するのに必須の値が含まれた参照メッセージを作成します。この値とは、例えば、送信側ファイルの名前および宛先ファイルの名前、そしてオブジェクト・タイプなどです。 例として、 WebSphere MQ に付属しているサンプルは、以下のパラメーターを使用して、ファイル `d:\x\file.in` を `QMGR1` から `QMGR2` に送信し、ファイルを `d:\y\file.out` として再作成するための参照メッセージを作成します。

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   ここで、`QR` は `QMGR2` の宛先キューを参照するリモート・キューの定義を表します。

   注 : UNIX and Linux プラットフォームの場合、宛先ファイルのディレクトリーを示すのに、円記号を 1 つではなく 2 つ (¥¥) 使用します。 したがって、**amqsprm** コマンドは次のように表示されます。

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```

4. 参照メッセージは、(ファイルのデータを除いて) /q パラメーターで指定されたキューに書き込まれます。 これがリモート・キューの場合、メッセージは対応する伝送キューに書き込まれます。
5. サンプルは、/w パラメーターで指定された時間 (デフォルトは 15 秒) COA 報告を待機します。これは例外報告と共にローカル・キュー・マネージャー (QMGR1) で作成された動的キューに返送されます。

参照メッセージ出口サンプル (*amqsxrma.c*、*AMQSXRM4*) の設計

このサンプルは、チャネル定義のメッセージ出口ユーザー・データ・フィールド内のオブジェクト・タイプと一致するオブジェクト・タイプを持つ参照メッセージを認識します。

これらのメッセージについては、以下の 2 つのことが起こります。

- 送信側チャネルまたはサーバー・チャネルでは、指定された長さのデータを、指定されたファイルの指定されたオフセットから、参照メッセージのあとのエージェント・バッファーに残っている空間にコピーします。 ファイルの終わりに到達しない場合は、参照メッセージは *DataLogicalOffset* フィールドを更新後、伝送キューに書き戻されます。
- 要求側チャネルまたは受信側チャネルでは、*DataLogicalOffset* フィールドがゼロで、指定されたファイルがない場合、そのファイルを作成します。 参照メッセージの次にくるデータは、指定されたファイルの最後に追加されます。 参照メッセージが指定されたファイルの最後のメッセージではない場合、

参照メッセージは廃棄されます。 最後のメッセージである場合は、参照メッセージは追加データなしでチャネル出口に戻り、宛先キューに書き込まれます。

送信側チャネルおよびサーバー・チャネルについては、入力参照メッセージ内の *DataLogicalLength* フィールドがゼロの場合、ファイルの残りの部分、つまり、*DataLogicalOffset* からファイルの最後まではチャネルに沿って送信されます。 上記のフィールドがゼロでない場合は、指定された長さだけが送信されます。

エラーが発生した場合 (例えば、サンプルがファイルをオープンできない場合) は、MQCXP。*ExitResponse* が MQXCC_SUPPRESS_FUNCTION に設定されているため、処理中のメッセージは、宛先キューに進む代わりに送達不能キューに書き込まれます。 フィードバック・コードが MQCXP に戻されます。*Feedback* 。レポート・メッセージのメッセージ記述子の *Feedback* フィールドにメッセージを書き込んだアプリケーションに返されます。 これは、書き込みアプリケーションが MQMD の *Report* フィールドに MQRO_EXCEPTION を設定することにより例外報告を要求したためです。

参照メッセージのエンコードまたは *CodedCharacterSetId* (CCSID) がキュー・マネージャーのものと異なる場合、参照メッセージはローカル・エンコード方式および CCSID に変換されます。 システムが提供するサンプルである amqsprm では、オブジェクトの形式は MQFMT_STRING なので、amqsxrm はオブジェクト・データがファイルに書き込まれる前に受信側でローカル CCSID に変換します。

このファイルにマルチバイト文字 (例えば、DBCS または Unicode) が含まれる場合は、転送されるファイルの形式を MQFMT_STRING として指定しないでください。 これは、ファイルが送信側でセグメント化されるときに、マルチバイト文字が分割される可能性があるからです。 このようなファイルを転送し、変換するためには、参照メッセージ出口でファイルを変換せずに、転送が完了した時点で受信側でそのファイルを変換するように、そのファイルを MQFMT_STRING 以外の形式で指定してください。

参照メッセージ出口サンプルのコンパイル

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

amqsxrma をコンパイルするには、以下のコマンドを使用します。

AIX の場合

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

HP-UX の場合

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -IMQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -LMQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

Linux 上

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

Solaris の場合

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm

 -lsocket
-lnsl -ldl
```

Windows の場合

WebSphere MQ では、サーバー・パッケージだけでなくクライアント・パッケージも含めて mqm ライブラリーが提供されるようになったため、以下の例では、 mqmvx.lib の代わりに mqm.lib を使用します。

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

チャネル出口の作成とコンパイルに関する一般情報については、401 ページの『チャネル出口プログラムの作成』を参照してください。

読み取り参照メッセージ・サンプル (*amqsgrma.c*、*AMQSGRM4*) の設計

このトピックでは、読み取り参照メッセージ・サンプルの設計について説明します。

プログラム・ロジックは次のとおりです。

1. サンプルは起動され、キュー名およびキュー・マネージャー名を入力トリガー・メッセージから抜き出します。
2. 次に、サンプルは MQCONN を使用して指定されたキュー・マネージャーに接続し、MQOPEN を使用して指定されたキューをオープンします。
3. サンプルはループ内に 15 秒の待機時間を持つ MQGET 呼び出しを発行し、キューからメッセージを読み取ります。
4. メッセージが参照メッセージの場合、サンプルは転送されたファイルの有無を確認します。
5. そこでサンプルはキューをクローズし、キュー・マネージャーから切断します。

要求サンプル・プログラム

要求サンプル・プログラムは、クライアント / サーバー処理を示します。 このサンプルは、要求メッセージをサーバー・プログラムによって処理する宛先サーバー・キューに入れるクライアントです。 サーバー・プログラムが応答先キューに応答メッセージを書き込むのを待機します。

要求サンプルは、MQPUT 呼び出しを使用して、一連の要求メッセージを宛先サーバー・キューに入れます。 これらのメッセージは、ローカル・キュー SYSTEM.SAMPLE.REPLY を応答先キュー (ローカル・キューまたはリモート・キューのいずれか) として指定します。 プログラムは応答メッセージを受信するまで待機し、そのメッセージを表示します。 応答は、宛先サーバー・キューがサーバー・アプリケーションによって処理中であるか、あるいはアプリケーションがその目的で起動される場合にのみ送信されます (照会、設定、およびエコー・サンプル・プログラムは起動されるように設計されています)。 C サンプルは、最初の応答が到着するまで、1 分間 (COBOL サンプルは 5 分間) 待機し (サーバー・アプリケーションが起動される時間を与えるため)、以降の応答については 15 秒間待機しますが、両サンプルは共に応答がなくても終了することができます。 要求サンプル・プログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

要求サンプル・プログラムの実行

amqsreq0.c、amqsreq、および amqsreqc のサンプルの実行

C バージョンのプログラムでは、次の 3 つのパラメーターをとります。

1. 宛先サーバー・キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)
3. 応答キュー (オプション)

例えば、それぞれ次のように入力します。

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

この例では、myqueue が宛先サーバー・キューの名前、qmanagername が myqueue を所有するキュー・マネージャーの名前、そして replyqueue が応答キューの名前です。

キュー・マネージャーの名前を省略すると、デフォルトのキュー・マネージャーがキューを所有すると見なされます。 応答キューの名前を省略すると、デフォルトの応答キューが指定されます。

amq0req0.cbl サンプルの実行

COBOL バージョンには、パラメーターがありません。 プログラムはデフォルトのキュー・マネージャーに接続し、これを実行すると、次のように入力が促されます。

```
Please enter the name of the target server queue
```

このプログラムでは、StdIn からの入力値を使用し、テキストの各行を要求メッセージの内容として使用しながら、各行を宛先サーバー・キューに付加します。 このプログラムは、ヌル行を読み込んだ時点で終了します。

AMQSREQ4 サンプルの実行

C プログラムは、stdin (キーボード) からデータを取り込むことによってメッセージを作成します。入力が終了した時点でブランクが現れます。 このプログラムは、最高で 3 つのパラメーターを取ります。それらは、宛先キュー名 (必須)、キュー・マネージャー名 (オプション)、応答先キュー名 (オプション) です。 キュー・マネージャー名が指定されない場合は、デフォルトのキュー・マネージャーが使用されます。 応答先キューが指定されない場合は、SYSTEM.SAMPLE.REPLY キューが使用されます。

次の例は、C サンプル・プログラムの呼び出し例です。ここでは、応答先キューを指定し、デフォルトのキュー・マネージャーを使用します。

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

注 : キュー名の場合、大文字と小文字の区別があることに注意してください。 サンプル・ファイル作成プログラム AMQSAMP4 によって作成されたキューの名前はすべて、大文字で作成されます。

AMQ0REQ4 サンプルの実行

COBOL プログラムは、キーボードからのデータを受け入れて、メッセージを作成します。 プログラムを始動するには、始動するプログラムを呼び出し、宛先キューの名前をパラメーターとして指定します。 プログラムは、キーボードから受け入れた入力をバッファーに入れ、テキストの各行ごとに要求メッセージを作成します。 キーボードでブランク行を入力するとプログラムは停止します。

トリガー操作を使用した要求サンプルの実行

サンプルがトリガー操作および、照会、設定、またはエコーのいずれかのサンプル・プログラムと共に使用される場合、入力行は、起動されたプログラムにアクセスさせたいキューの名前でなければなりません。

UNIX、*Linux* および *Windows* システム

トリガー操作を使用してサンプルを実行するには、次のようにします。

1. 1 つのセッションでトリガー・モニター・プログラム RUNMQTRM を始動します (開始キュー SYSTEM.SAMPLE.TRIGGER が使用できます)。
2. 別のセッションで amqsreq プログラムを始動します。
3. 宛先サーバー・キューが定義されているか確認します。

   要求サンプルがメッセージを入れるための宛先サーバー・キューとして使用可能なサンプル・キューには、次のものがあります。

   - SYSTEM.SAMPLE.INQ - 照会サンプル・プログラム用
   - SYSTEM.SAMPLE.SET - 設定サンプル・プログラム用
   - SYSTEM.SAMPLE.ECHO - エコー・サンプル・プログラム用

これらのキューには、FIRST というトリガー・タイプがありますが、要求サンプルを実行する前にキュー上に既にメッセージがある場合には、サーバー・アプリケーションはユーザーが送信したメッセージによって起動されません。

4. 照会、設定、またはエコー・サンプル・プログラムを使用するために、キューが定義されているか確認します。

これは、要求サンプルがメッセージを送信するときにトリガー・モニターは作動可能であることを意味します。

注 : RUNMQSC および amqscos0.tst ファイルを使用して作成されたサンプル・プロセス定義によって C のサンプルが起動されます。 COBOL バージョンを使用するには、amqscos0.tst のプロセス定義を変更し、この更新済みファイルと共に RUNMQSC を使用してください。

148 ページの図 22 は、要求サンプルと照会サンプルを一緒に使用する方法を示しています。

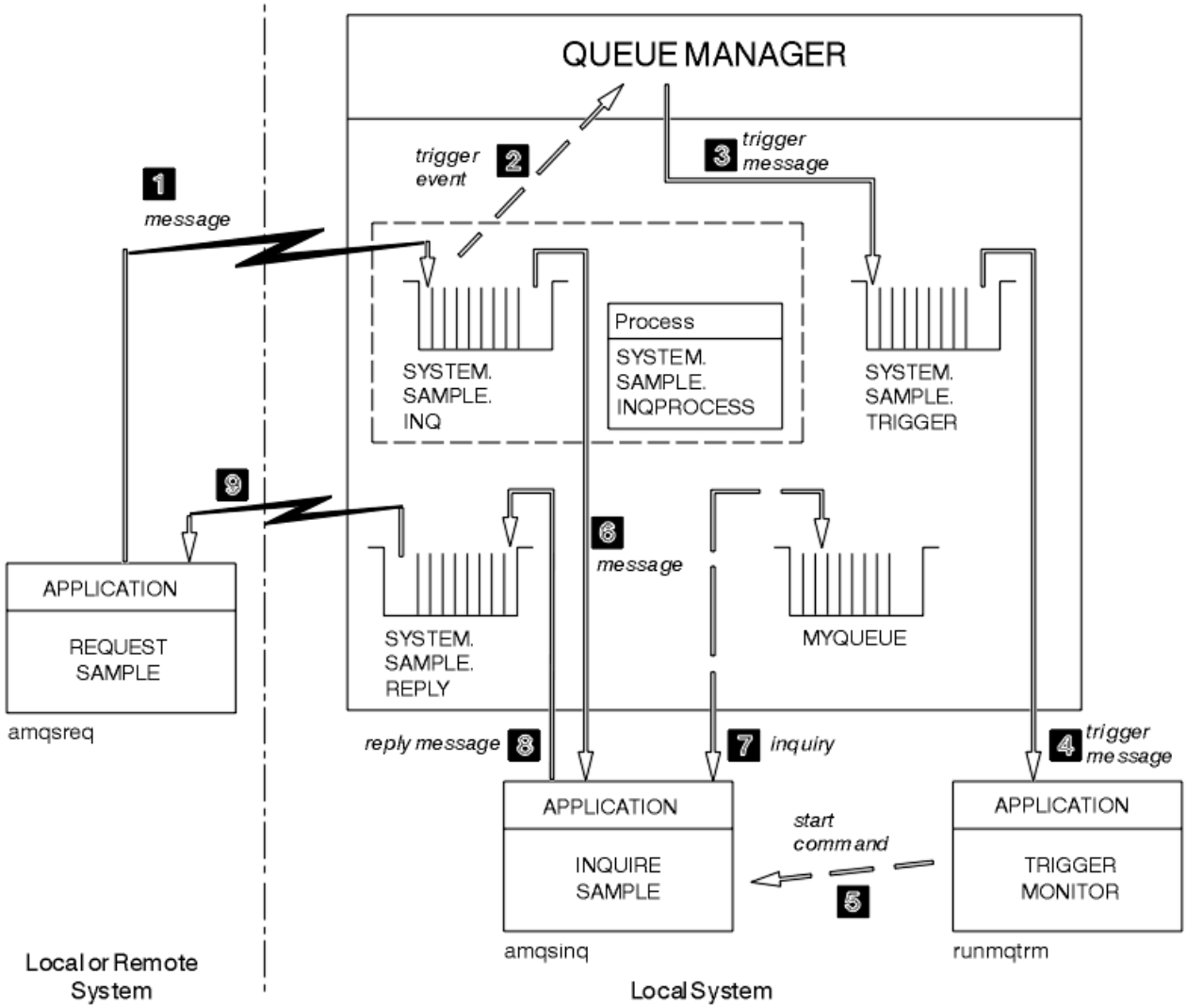


図 22. トリガー操作を使用する要求および照会サンプル

148 ページの図 22 では、要求サンプルはメッセージを宛先サーバー・キュー SYSTEM.SAMPLE.INQ に入れ、照会サンプルはキュー MYQUEUE を照会します。 また、amqscos0.tst を実行した時に定義したいずれかのサンプル・キュー、または定義したそれ以外のキューを照会サンプルとして使用することもできます。

注 : 148 ページの図 22 の数字は、イベントの発生する順序を示しています。

トリガー操作を使用して、要求サンプルおよび照会サンプルを実行するには、次のようにします。

1. 使用したいキューが定義されているか検査します。 amqscos0.tst を実行してサンプル・キューを定義すると共に、キュー MYQUEUE を定義します。

2. トリガー・モニター・コマンド RUNMQTRM を実行します。

```
RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
```

3. 要求サンプルを実行します。

```
amqsreq SYSTEM.SAMPLE.INQ
```

注: プロセス・オブジェクトは起動する必要があるものを定義します。 クライアントおよびサーバーが同じプラットフォーム上で実行されていない場合、トリガー・モニターが開始したプロセスは *ApplType* を定義しなければなりません。定義していないと、サーバーはデフォルトの定義 (つまり、通常サーバー・マシンに関連しているアプリケーションのタイプ) を選択し、障害を引き起こします。

アプリケーション・タイプのリストについては、ApplType を参照してください。

4. 照会サンプルに使用したいキューの名前を入力します。

```
MYQUEUE
```

5. ブランク行を入力します (要求プログラムを終了するため)。
6. ここで、要求サンプルは照会プログラムが MYQUEUE から獲得したデータを含むメッセージを表示します。

複数のキューを使用できます。この場合、ステップ 149 ページの『4』 で他のキューの名前を入力します。

トリガー操作の詳細については、330 ページの『トリガーによる IBM WebSphere MQ アプリケーションの開始』 を参照してください。

要求サンプル・プログラムの設計

次に、このプログラムは、宛先サーバー・キューをオープンして、メッセージを入れられるようにします。このプログラムでは、MQOO_OUTPUT オプション付きの MQOPEN 呼び出しを使用します。 キューをオープンできない場合、プログラムは MQOPEN 呼び出しによって戻された理由コードを含むエラー・メッセージを表示します。

その後プログラムは、応答メッセージを読み取るために、SYSTEM.SAMPLE.REPLY と呼ばれる応答先キューをオープンします。 これに関して、このプログラムは、MQOO_INPUT_EXCLUSIVE オプション付きの MQOPEN 呼び出しを使用します。 プログラムは、キューをオープンできない場合に、MQOPEN 呼び出しによって戻される理由コードを含むエラー・メッセージを表示します。

次にプログラムは、入力行ごとにテキストをバッファー中に読み込み、MQPUT 呼び出しを使用してその行のテキストを含む要求メッセージを作成します。 この呼び出しで、プログラムは MQRO_EXCEPTION_WITH_DATA 報告オプションを使用して、要求メッセージについて送信されるすべての報告メッセージに 100 バイトまでのメッセージ・データを入れるように要求します。 プログラムは、入力が終了するか、MQPUT 呼び出しが異常終了するまで処理を続行します。

その後プログラムは、MQGET 呼び出しを使用してキューから応答メッセージを除去し、応答に含まれるデータを表示します。 MQGET 呼び出しでは、MQGMO_WAIT、MQGMO_CONVERT、および MQGMO_ACCEPT_TRUNCATED オプションを使用します。 *WaitInterval* は、最初の応答では、COBOL バージョンでは 5 分、C バージョンでは 1 分となります (サーバーのアプリケーションを起動する時間を与えるため)。以降の応答では、15 秒となります。 キューにメッセージがない場合、プログラムはこれらの期間待ちます。 メッセージがこの時間内に到着しない場合、呼び出しは失敗し、MQRC_NO_MSG_AVAILABLE 理由コードを戻します。 この呼び出しでは、MQGMO_ACCEPT_TRUNCATED_MSG オプションも使用します。これによって、宣言されたバッファー・サイズよりも大きいメッセージは切り捨てられます。

このプログラムは、それぞれの MQGET 呼び出しのあとに MQMD 構造体の *MsgId* および *CorrelId* フィールドをクリアする方法を示します。これは、この呼び出しによって、これらのフィールドに、このプログラムが取り出すメッセージに含まれる値が設定されるからです。 これらのフィールドをクリアすることは、MQGET 呼び出しを連続して行った場合に、メッセージがキューに保持された順に取り出されることを意味します。

このプログラムは、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE 理由コードを戻すか、あるいは MQGET 呼び出しが失敗するまで、処理を続行します。 呼び出しが失敗すると、このプログラムは、理由コードを含むエラー・メッセージを表示します。

次に、このプログラムは、MQCLOSE 呼び出しを使用して、宛先サーバー・キューと応答先キューの両方をクローズします。

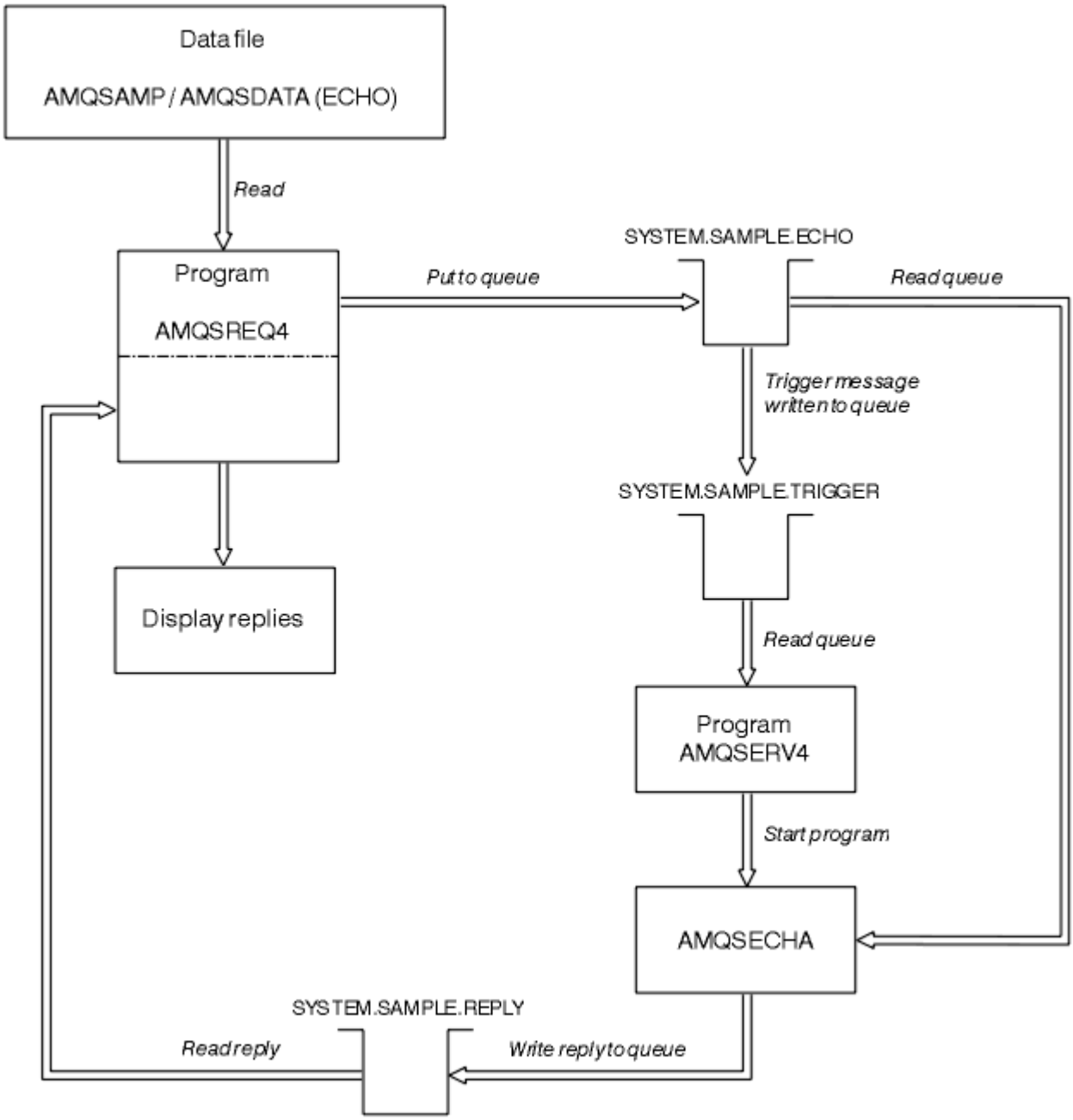


図 *23.* サンプル *IBM i* クライアント/サーバー *(*エコー*)* プログラムのフローチャート

設定サンプル・プログラム

設定サンプル・プログラムは、MQSET 呼び出しを使用してキューの *InhibitPut* 属性を変更することによって、キュー上での書き込み操作を禁止します。 また、設定サンプル・プログラムの設計について説明します。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

このプログラムは、トリガーされるプログラムとして実行することを意図したものです。したがって、その入力は、照会される属性を持つ宛先キューの名前を含む MQTMC2 (トリガー・メッセージ) 構造体のみとなります。 C バージョンもキュー・マネージャー名を使用します。 COBOL バージョンでは、デフォルトのキュー・マネージャーを使用します。

トリガー・プロセスを処理するには、使用したい設定サンプル・プログラムがキュー SYSTEM.SAMPLE.SET に到着するメッセージによって起動されているか確認してください。 これを行うには、使用したい設定サンプル・プログラムの名前を、プロセス定義 SYSTEM.SAMPLE.SETPROCESS の *ApplicId* フィールドに指定します。 サンプル・キューのトリガー・タイプは FIRST です。要求サンプルを実行する前にメッセージが既にキュー上にある場合、設定サンプルはユーザーが送信したメッセージによって起動されません。

正しく定義を設定したら、以下のようにします。

- UNIX、Linux および Windows システムの場合、1 つのセッションで **runmqtrm** プログラムを開始後、別のセッションで amqsreq プログラムを開始します。
- IBM i の場合、1 つのセッションで AMQSERV4 プログラムを開始後に、別のセッションで AMQSREQ4 プログラムを開始します。 AMQSERV4 の代わりに AMQSTRG4 を使用しても構いませんが、ジョブの送信が遅れる可能性があるため、処理の流れを追うのが難しくなります。

要求サンプル・プログラムを使用して、それぞれがキュー名のみを含む要求メッセージをキュー SYSTEM.SAMPLE.SET に送信します。 各要求メッセージごとに、設定サンプル・プログラムは、指定のキューで書き込み操作が禁止されたことの確認を含む応答メッセージを送信します。 応答は、要求メッセージで指定した応答先キューに送信されます。

設定サンプル・プログラムの設計

このプログラムは、始動時に渡されたトリガー・メッセージ構造体で名前が指定されたキューをオープンします。 (分かりやすくするため、このキューを要求キュー と呼びます。) このプログラムは、MQOPEN 呼び出しを使用して、共用する入力に対してこのキューをオープンします。

プログラムは、MQGET 呼び出しを使用して、このキューからメッセージを除去します。 この呼び出しでは、5 秒間の待機時間を指定した、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプションを使用します。 このプログラムは、各メッセージの記述子をテストして、要求メッセージであるか確認します。要求メッセージでない場合は、このプログラムはそのメッセージを廃棄して、警告メッセージを表示します。

各要求メッセージが要求キューから除去されたら、このプログラムは、そのデータに含まれるそのキュー (宛先キュー と呼びます) の名前を読み込み、MQOO_SET オプション付きの MQOPEN 呼び出しを使用して、そのキューをオープンします。 次に、このプログラムは、MQSET 呼び出しを使用して、宛先キューの *InhibitPut* 属性値を MQQA_PUT_INHIBITED に設定します。

MQSET 呼び出しが正常に行われると、このプログラムは MQPUT1 呼び出しを使用して、応答メッセージを応答先キューに入れます。 このメッセージには、ストリング `PUT inhibited` が含まれています。

MQOPEN または MQSET 呼び出しが失敗すると、プログラムは MQPUT1 呼び出しを使用して、`report` メッセージを応答先キューに書き込みます。 この報告メッセージのメッセージ記述子の *Feedback* フィールドには、MQOPEN または MQSET の失敗した呼び出しによって、戻される理由コードが入ります。

MQSET 呼び出し後、このプログラムは MQCLOSE 呼び出しを使用して、宛先キューをクローズします。

要求キューにメッセージが残っていない場合、このプログラムはそのキューをクローズし、キュー・マネージャーとの接続を切り離します。

SSL/TLS サンプル・プログラム

AMQSSLC は、MQCONNX 呼び出しで MQCNO および MQSCO 構造体を使用して SSL/TLS クライアント接続情報を提供する方法を示すサンプル C プログラムです。 このプログラムを使用すると、クライアント MQI アプリケーションで、クライアント・チャネル定義テーブル (CCDT) を使用せずに実行時にそのクライアント接続チャネルの定義および SSL/TLS 設定が提供されます。

接続名が指定されると、プログラムは MQCD 構造体にクライアント接続チャネル定義を構成します。

キー・リポジトリー・ファイルの語幹名が指定されると、プログラムは MQSCO 構造体を構成します。OCSP レスポンダー URL も指定された場合、プログラムは認証情報レコード MQAIR 構造体を構成します。

次に、プログラムは MQCONNX を使用してキュー・マネージャーに接続します。 接続先のキュー・マネージャーの名前が照会され、表示されます。

このプログラムは、MQI クライアント・アプリケーションとしてリンクされることを前提としています。ただし、通常の MQI アプリケーションとしてリンク可能です。 単にローカル・キュー・マネージャーに接続し、クライアント接続情報は無視します。

AMQSSLC は、以下のパラメーターを受け入れます。これらはすべてオプションです。

-m QmgrName
接続先のキュー・マネージャーの名前

-c ChannelName
使用するチャネルの名前

-x ConnName
サーバー接続名

SSL/TLS パラメーター:

-k KeyReposStem
キー・リポジトリー・ファイルの語幹名。 これは、.kdb 接尾部なしのファイルへの絶対パスです。 以下に例を示します。

```
/home/user/client
C:\User\client
```

-s CipherSpec
キュー・マネージャーの SVRCONN チャネル定義の SSLCIPH に対応する SSL/TLS チャネル CipherSpec ストリング。

-f
FIPS 140-2 認証アルゴリズムのみを使用する必要があることを指定します。

-b VALUE1[,VALUE2...]
Suite B 準拠アルゴリズムのみを使用する必要があることを指定します。 このパラメーターは、NONE、128_BIT、192_BIT の 1 つ以上の値の、コンマ区切りのリストです。 これらの値は、MQSUITEB 環境変数の対応する値、およびクライアント構成ファイルの SSL スタンザにある同等の EncryptionPolicySuiteB 設定と同じ意味を持ちます。

-p Policy
使用する証明書妥当性検査ポリシーを指定します。 以下のいずれかの値を選択できます。

ANY
セキュア・ソケット・ライブラリーでサポートされる証明書妥当性検査ポリシーのいずれかにおいて、その証明書チェーンが有効であると見なされる場合に、それらのポリシーのそれぞれを適用し、証明書チェーンを受け入れます。 この設定は、最新の証明書標準に準拠しない旧式のデジタル証明書との後方互換性を最大にするために使用できます。

RFC5280
RFC 5280 準拠の証明書妥当性検査ポリシーのみを適用します。 この設定は、ANY 設定よりも厳密に妥当性検査しますが、一部の旧式のデジタル証明書を拒否します。

デフォルト値は ANY です。

OCSP 証明書の失効のパラメーター:

-o URL
OCSP レスポンダー URL

SSL/TLS サンプル・プログラムの実行

SSL/TLS サンプル・プログラムを実行するには、まず最初に SSL または TLS 環境をセットアップする 必要があります。 その後、いくつかのパラメーターを指定して、コマンド行からサンプルを実行します。

このタスクについて

以下の手順では、個人証明書を使用するサンプル・プログラムを実行します。 コマンドを変更することにより、例えば、CA 証明書を使用して、OCSP レスポンダーを使用した CA 証明書の状況の検査を行うことができます。 サンプル内の説明を参照してください。

手順

1. QM1 という名前で、キュー・マネージャーを作成する。 詳しくは、「crtmqm」を参照してください。
2. キュー・マネージャーのキー・リポジトリーを作成する。 詳しくは、UNIX, Linux, and Windows システムでの鍵リポジトリーのセットアップを参照してください。
3. クライアントのキー・リポジトリーを作成する。 このリポジトリーに *clientkey.kdb* という名前を付けます。
4. キュー・マネージャーの個人証明書を作成する。 詳しくは、UNIX, Linux, and Windows システムでの自己署名個人証明書の作成を参照してください。
5. クライアントの個人証明書を作成する。
6. サーバーのキー・リポジトリーから個人証明書を抽出し、クライアント・リポジトリーに追加する。 詳しくは、 UNIX、 Linux および Windows システムでの鍵リポジトリーからの自己署名証明書の公開部分の抽出 、および UNIX、 Linux または Windows システムでの鍵リポジトリーへの CA 証明書 (または自己署名証明書の公開部分) の追加を参照してください。
7. クライアントのキー・リポジトリーから個人証明書を抽出し、サーバーのキー・リポジトリーに追加する。
8. 以下のように、MQSC コマンドを使用してサーバー接続チャネルを作成する。

   ```
   DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(NULL_SHA)
   ```

   詳しくは、「サーバー接続チャネル」を参照してください。
9. キュー・マネージャー上のチャネル・リスナーを定義および開始する。 詳しくは、「DEFINE LISTENER」および「START LISTENER」を参照してください。
10. 以下のコマンドを使用して、サンプル・プログラムを実行する。

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "c:\Program Files\IBM\WebSphere MQ\clientkey" -s NULL_SHA
    -o http://dummy.OCSP.responder
    ```

タスクの結果

サンプル・プログラムは以下の操作を実行します。

1. 指定されたオプションを使用して、指定されたキュー・マネージャーまたはデフォルトのキュー・マネージャーに接続します。
2. キュー・マネージャーを開いて、その名前を照会します。
3. キュー・マネージャーを閉じます。
4. キュー・マネージャーから切断します。

サンプル・プログラムが正常に実行されると、以下の例のような出力が表示されます。

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using SSL CipherSpec NULL_SHA
Using SSL key repository stem c:\Program Files\IBM\WebSphere MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

サンプル・プログラムで問題が発生した場合、該当するエラー・メッセージが表示されます。例えば、無効な OCSP レスポンダー URL を指定すると、以下のようなメッセージを受け取ります。

```
MQCONNX ended with reason code 2553
```

理由コードのリストについては、API 理由コードを参照してください。

トリガー・サンプル・プログラム

トリガー・サンプルに提供されている関数は、**runmqtrm** プログラム内のトリガー・モニターに提供されているトリガー・サンプルのサブセットです。

これらのプログラムの名前については、98 ページの『サンプル・プログラムの中で示されている機能』を参照してください。

トリガー・サンプルの設計

トリガー・サンプル・プログラムは、MQOO_INPUT_AS_Q_DEF オプション付きの MQOPEN 呼び出しを使用して、開始キューをオープンします。 このプログラムは、MQGMO_ACCEPT_TRUNCATED_MSG オプションおよび MQGMO_WAIT オプション (無制限の待機時間を指定する) 付きの MQGET 呼び出しを使用して、開始キューからメッセージを取得します。 このプログラムは、それぞれの MQGET 呼び出しがメッセージを順に取得する前に、*MsgId* フィールドおよび *CorrelId* フィールドをクリアします。

開始キューからメッセージが取り出されると、このプログラムはそのメッセージのサイズを検査して、MQTM 構造体のサイズと同じであるか確認します。 このテストが失敗すると、プログラムは警告を表示します。

トリガー・メッセージが有効である場合、トリガー・サンプルは、*ApplicId*、*EnvrData*、*Version*、および *ApplType* の各フィールドからデータをコピーします。 これらのフィールドのうち最後の 2 つは数値フィールドです。したがって、このプログラムは、UNIX、Linux、および Windows システム用の MQTMC2 構造体で使用するために文字置換を行います。

トリガー・サンプルは、トリガー・メッセージの *ApplicId* フィールドで指定したアプリケーションに開始コマンドを実行し、MQTMC2 または MQTMC (トリガー・メッセージの文字バージョン) 構造体を渡します。 UNIX、Linux および Windows システムでは、*EnvData* フィールドは呼び出し側のコマンド・ストリングへの拡張として使用されます。

最後に、このプログラムは開始キューをクローズします。

トリガー・サンプル・プログラムの実行

このトピックでは、トリガー・サンプル・プログラムの実行について説明します。

amqstrg0.c、amqstrg、および amqstrgc のサンプルの実行

このプログラムは次の 2 つのパラメーターをとります。

1. 開始キューの名前 (必須)
2. キュー・マネージャーの名前 (オプション)

キュー・マネージャーを指定しないと、このプログラムはデフォルトのキュー・マネージャーに接続します。 サンプルの開始キューは、amqscos0.tst を実行する時には既に定義されています。そのキューの名前は SYSTEM.SAMPLE.TRIGGER で、このプログラムを実行する時に使用できます。

注 : このサンプルの関数は、**runmqtrm** プログラムで提供される完全なトリガー関数のサブセットです。

トリガー・サーバーの設計

トリガー・サーバーの設計は、トリガー・サーバーが次のことを行う以外は、トリガー・モニターと同様です。

- MQAT_CICS と同様に MQAT_OS400 アプリケーションを許可します。

- CICS アプリケーションについては、*EnvData* を代用します。例えば、STRCICSUSR コマンド内のトリガー・メッセージから CICS 領域を指定するような場合です。
- 共用する入力に対して開始キューをオープンし、多数のトリガー・サーバーを同時に実行できるようにします。

注 : AMQSERV4 によって開始されるプログラムは、MQDISC 呼び出しを使用してはなりません。これによってトリガー・サーバーが停止されてしまうからです。 AMQSERV4 によって開始されるプログラムが MQCONN 呼び出しを使用すると、MQRC_ALREADY_CONNECTED 理由コードを受け取ります。

TUXEDO サンプル

TUXEDO の書き込みおよび読み取りサンプル・プログラムおよび TUXEDO でのサーバー環境の構築について学習します。

これらのサンプルを実行するには、サーバー環境を構築しておく必要があります。

注 : このトピック全体で、複数行にまたがる長いコマンドの分割に円記号 (¥) が使用されています。この文字は入力しないでください。 各コマンドは単一行として入力してください。

サーバー環境の構築

さまざまなプラットフォーム用の WebSphere MQ のサーバー環境を構築する方法について説明します。

TUXEDO 作業環境があるものとします。

WebSphere MQ for AIX (32 ビット*)* 用のサーバー環境の構築

1. サーバー環境が構築されているディレクトリー (例えば、<APPDIR>) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。ここで、TUXDIR は TUXEDO のルート・ディレクトリーであり、*MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH□inc -I /<APPDIR> -L MQ_INSTALLATION_PATH□lib"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/<APPDIR>/amqstxvx.V
$ export LIBPATH=$TUXDIR□lib:MQ_INSTALLATION_PATH□lib:□lib
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.a
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /<APPDIR>/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for AIX (64 ビット*)* 用のサーバー環境の構築

1. サーバー環境が構築されているディレクトリー (例えば、<APPDIR>) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。ここで、TUXDIR は TUXEDO のルート・ディレクトリーであり、*MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

```
$ export CFLAGS="-I MQ_INSTALLATION_PATH□inc -I /<APPDIR> -L
MQ_INSTALLATION_PATH□lib64"
$ export LDOPTS="-lmqm"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=/<APPDIR>/amqstxvx.V
$ export LIBPATH=$TUXDIR□lib64:MQ_INSTALLATION_PATH□lib64:□lib64
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a \
       -r MQSeries_XA_RMI -s MPUT2:MPUT
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.a
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /<APPDIR>/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for Solaris (32 ビット*)* 用のサーバー環境の構築

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. サーバー環境を構築するディレクトリー (例えば、*APPDIR*) を作成し、このディレクトリーですべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。この場合、TUXDIR は TUXEDO のルート・ディレクトリーです。

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs  -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so  \
```

```
$ buildclient -o dogets  -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /APPDIR/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for Solaris (64 ビット) 用のサーバー環境の構築

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. サーバー環境が構築されているディレクトリー (例えば、<APPDIR>) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。この場合、TUXDIR は TUXEDO のルート・ディレクトリーです。

```
$ export CFLAGS="-I /<APPDIR>"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR□lib:MQ_INSTALLATION_PATH□lib:□lib64
$ export LD_LIBRARY_PATH=$TUXDIR□lib64:MQ_INSTALLATION_PATH□lib64:□lib64
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
```

```
        -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
        -r MQSeries_XA_RMI -s MPUT2:MPUT \
        -s MGET2:MGET \
        -v -bshm
        -l -ldl
$ buildclient -o doputs  -f amqstxpx.c \
        -f MQ_INSTALLATION_PATH/lib64/libmqm.so  \

$ buildclient -o dogets  -f amqstxgx.c \
        -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /<APPDIR>/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. Tuxedo を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for HP-UX (32 ビット*)* 用のサーバー環境の構築

注 : 32 ビットの TUXEDO サーバー環境は、Itanium プラットフォームでのみ構築できます。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. サーバー環境が構築されているディレクトリー (例えば、<APPDIR>) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。この場合、TUXDIR は TUXEDO のルート・ディレクトリーです。

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

mkfldhdr および viewc コマンドを実行した後、TUXEDO アプリケーション・ディレクトリーに amqstxvx.h ヘッダー・ファイルが作成されます。 このファイルを TUXEDO アプリケーション・ディレクトリーから TUXEDO インクルード・ディレクトリーにコピーしてから、以下のコマンドを実行します。

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /<APPDIR>/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. TUXEDO を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for HP-UX (64 ビット*)* 用のサーバー環境の構築

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. サーバー環境が構築されているディレクトリー (例えば、<APPDIR>) を作成し、このディレクトリー内のすべてのコマンドを実行します。
2. 以下の環境変数をエクスポートします。この場合、TUXDIR は TUXEDO のルート・ディレクトリーです。

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
```

```
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. 以下のものを TUXEDO ファイル udataobj/RM に追加します。

   HP-UX IA64 (IPF) プラットフォームでは次のようにします。

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

   注 : HP-UX IA64 (IPF) プラットフォームに付属している WebSphere MQ ライブラリーのファイル名拡張子は .so です。

4. 次のコマンドを実行します。

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   mkfldhdr および viewc コマンドを実行した後、TUXEDO アプリケーション・ディレクトリーに amqstxvx.h ヘッダー・ファイルが作成されます。 このファイルを TUXEDO アプリケーション・ディレクトリーから TUXEDO インクルード・ディレクトリーにコピーしてから、以下のコマンドを実行します。

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
```

   HP-UX IA64 (IPF) プラットフォームでは次のようにします。

```
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. ubbstxcx.cfg を編集し、必要に応じてマシン名、作業ディレクトリー、およびキュー・マネージャーに関する詳細を追加します。

```
$ tmloadcf    -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. TLOGDEVICE を作成します。次のように入力します。

```
$tmadmin -c
```

   プロンプトが出されます。 このプロンプトで次のように入力します。

```
> crdl -z /<APPDIR>/TLOG1
```

7. キュー・マネージャーを始動します。

```
$ strmqm
```

8. TUXEDO を始動します。

```
$ tmboot -y
```

これで、doputs および dogets プログラムを使用して、キューにメッセージを書き込んだり、キューからメッセージを取り出すことができます。

WebSphere MQ for Windows (32 ビット) 用のサーバー環境の構築

注 : 以下に、<> で示すフィールドをディレクトリー・パスに変更します。

< MQMDIR>	WebSphere MQ のインストール時に指定されたディレクトリー・パス (例えば、`g:\Program Files\IBM\WebSphere MQ`)
< TUXDIR>	TUXEDO のインストール時に指定されたディレクトリー・パス (例: `f:\tuxedo`)
< APPDIR>	サンプル・アプリケーションに使用するディレクトリー・パス (例: `f:\tuxedo\apps\mqapp`)

サーバー環境およびサンプルを構築するには、次のようにします。

1. 次のようにサンプル・アプリケーションを作成するアプリケーション・ディレクトリーを作成します。

```
f:\tuxedo\apps\mqapp
```

2. 以下のサンプル・ファイルを WebSphere MQ サンプル・ディレクトリーからアプリケーション・ディレクトリーにコピーします。

```
amqstxmn.mak
amqstxen.env
ubbstxcn.cfg
```

3. これらのファイルをそれぞれに編集して、インストール時に使用するディレクトリー名とディレクトリー・パスを設定します。
4. ubbstxcn.cfg (163 ページの図 24 を参照) を編集して、接続したいマシン名とキュー・マネージャーの詳細を追加します。
5. 以下のものを TUXEDO ファイル <TUXDIR>udataobj¥rm に追加します。

```
MQSeries_XA_RMI;MQRMIXASwitchDynamic;
   <MQMDIR>\tools\lib\mqmxa.lib <MQMDIR>\tools\lib\mqm.lib
```

ここで、< MQMDIR> は、前の例に示されているように置き換えられます。 ここでは 2 行で記載していますが、ファイル内では、新規のエントリーは 1 行でなければなりません。

6. 以下の環境変数を設定します。

```
TUXDIR=<TUXDIR>
TUXCONFIG=<APPDIR>\tuxconfig
FIELDTBLS=<MQMDIR>\tools\c\samples\amqstxvx.fld
LANG=C
```

7. TUXEDO 用 TLOG デバイスを作成します。 これを行うには、 `tmadmin -c` を呼び出して、次のコマンドを入力します。

```
crdl -z <APPDIR>\TLOG
```

<*APPDIR*> は置き換えられます。

8. 現行ディレクトリーを <APPDIR> に設定し、サンプル makefile (amqstxmn.mak) を外部プロジェクト makefile として呼び出します。 例えば、Microsoft Visual C++ では次のコマンドを実行します。

```
msvc amqstxmn.mak
```

「**build (構築)**」を選択して、すべてのサンプル・プロ グラムを作成します。

```
*RESOURCES
IPCKEY        99999
UID           0
GID           0
MAXACCESSERS  20
MAXSERVERS    20
MAXSERVICES   50
MASTER        SITE1
MODEL         SHM
LDBAL         N

*MACHINES
<MachineName> LMID=SITE1
              TUXDIR="f:\tuxedo"
              APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
              ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
              TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
              ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
              TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
              TLOGNAME=TLOG
              TYPE="i386NT"
              UID=0
              GID=0

*GROUPS
GROUP1
        LMID=SITE1  GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSeries_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1     SRVGRP=GROUP1 SRVID=1
MQSERV2     SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

図 *24. WebSphere MQ for Windows* 用 *ubbstxcn.cfg* ファイルの例

注 : ディレクトリー名とディレクトリー・パスは、インストール済み環境に合わせて変更してください。 また、キュー・マネージャー名 MYQUEUEMANAGER も、接続するキュー・マネージャーの名前に変更してください。 追加する必要のあるその他の情報は、<> で示してあります。

WebSphere MQ for Windows 用 ubbconfig ファイルの例を 163 ページの図 24 に示してあります。 これは、ubbstxcn.cfg として WebSphere MQ サンプル・ディレクトリーに提供されています。

WebSphere MQ for Windows 用に提供されているサンプル Make ファイル (164 ページの図 25 を参照) は、 ubbstxmn.mak という名前で、 WebSphere MQ サンプル・ディレクトリーに保持されています。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSeries_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

図 *25. WebSphere MQ for Windows* 用サンプル *TUXEDO makefile*

WebSphere MQ for Windows (64 ビット*)* 用のサーバー環境の構築

注 : 以下に、<> で示すフィールドをディレクトリー・パスに変更します。

< MQMDIR>	WebSphere MQ のインストール時に指定されたディレクトリー・パス (例えば、`g:\Program Files\IBM\WebSphere MQ`)
< TUXDIR>	TUXEDO のインストール時に指定されたディレクトリー・パス (例: `f:\tuxedo`)
< APPDIR>	サンプル・アプリケーションに使用するディレクトリー・パス (例: `f:\tuxedo\apps\mqapp`)

サーバー環境およびサンプルを構築するには、次のようにします。

1. 次のようにサンプル・アプリケーションを作成するアプリケーション・ディレクトリーを作成します。

   ```
   f:\tuxedo\apps\mqapp
   ```

2. 以下のサンプル・ファイルを WebSphere MQ サンプル・ディレクトリーからアプリケーション・ディレクトリーにコピーします。

   ```
   amqstxmn.mak
   amqstxen.env
   ubbstxcn.cfg
   ```

3. これらのファイルをそれぞれに編集して、インストール時に使用するディレクトリー名とディレクトリー・パスを設定します。
4. `ubbstxcn.cfg` (165 ページの図 26 を参照) を編集して、接続したいマシン名とキュー・マネージャーの詳細を追加します。
5. 以下のものを TUXEDO ファイル `<TUXDIR>udataobj\rm` に追加します。

   ```
   MQSeries_XA_RMI;MQRMIXASwitchDynamic;
      <MQMDIR>\tools\lib64\mqmxa64.lib <MQMDIR>\tools\lib64\mqm.lib
   ```

<MQMDIR> は置き換えられます。 ここでは 2 行で記載していますが、ファイル内では、新規のエントリーは 1 行でなければなりません。

6. 以下の環境変数を設定します。

```
TUXDIR=<TUXDIR>
TUXCONFIG=<APPDIR>\tuxconfig
FIELDTBLS=<MQMDIR>\tools\c\samples\amqstxvx.fld
LANG=C
```

7. TUXEDO 用 TLOG デバイスを作成します。 これを行うには、 `tmadmin -c` を呼び出して、次のコマンドを入力します。

```
crdl -z <APPDIR>\TLOG
```

前のサンプルで示されているように、*<APPDIR>* は置き換えられます。

8. 現行ディレクトリーを <APPDIR> に設定し、サンプル makefile (amqstxmn.mak) を外部プロジェクト makefile として呼び出します。 例えば、Microsoft Visual C++ では次のコマンドを実行します。

```
msvc amqstxmn.mak
```

「**build (構築)**」を選択して、すべてのサンプル・プロ グラムを作成します。

```
*RESOURCES
IPCKEY        99999
UID           0
GID           0
MAXACCESSERS  20
MAXSERVERS    20
MAXSERVICES   50
MASTER        SITE1
MODEL         SHM
LDBAL         N

*MACHINES
<MachineName> LMID=SITE1
              TUXDIR="f:\tuxedo"
              APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
              ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
              TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
              ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
              TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
              TLOGNAME=TLOG
              TYPE="i386NT"
              UID=0
              GID=0

*GROUPS
GROUP1
        LMID=SITE1  GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSeries_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1     SRVGRP=GROUP1 SRVID=1
MQSERV2     SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

図 *26. WebSphere MQ for Windows* 用 *ubbstxcn.cfg* ファイルの例

注 : ディレクトリー名とディレクトリー・パスは、インストール済み環境に合わせて変更してください。 また、キュー・マネージャー名 MYQUEUEMANAGER も、接続するキュー・マネージャーの名前に変更してください。 追加する必要のあるその他の情報は、<> で示してあります。

WebSphere MQ for Windows 用 ubbconfig ファイルの例を 165 ページの図 26 に示してあります。 これは、ubbstxcn.cfg として WebSphere MQ サンプル・ディレクトリーに提供されています。

WebSphere MQ for Windows 用に提供されているサンプル Make ファイル (166 ページの図 27 を参照) は、 ubbstxmn.mak という名前で、 WebSphere MQ サンプル・ディレクトリーに保持されています。

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSeries_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

図 *27. WebSphere MQ for Windows* 用サンプル *TUXEDO makefile*

TUXEDO のサンプル・サーバー・プログラム

サンプル・サーバー・プログラム (amqstxsx) は、書き込み (amqstxpx.c) および読み取り (amqstxgx.c) サンプル・プログラムと共に実行するように設計されています。 サンプル・サーバー・プログラムは、TUXEDO を始動すると自動的に実行されます。

注 : TUXEDO を始動する **前に** キュー・マネージャーを開始する必要があります。

このサンプル・サーバーでは 2 つの TUXEDO サービス MPUT1 と MGET1 を提供しています。

- MPUT1 サービスは書き込みサンプルによって作動し、同期点で MQPUT1 を使用して、TUXEDO によって制御される作業単位にメッセージを書き込みます。 このサービスでは、書き込みサンプルによって提供されるパラメーター QName および Message Text をとります。
- MGET1 サービスは、メッセージを取得するたびにキューをオープン、およびクローズします。 このサービスでは、読み取りサンプルによって提供されるパラメーター QName および Message Text をとります。

エラー・メッセージ、理由コード、および状況メッセージはすべて、TUXEDO ログ・ファイルに書き込まれます。

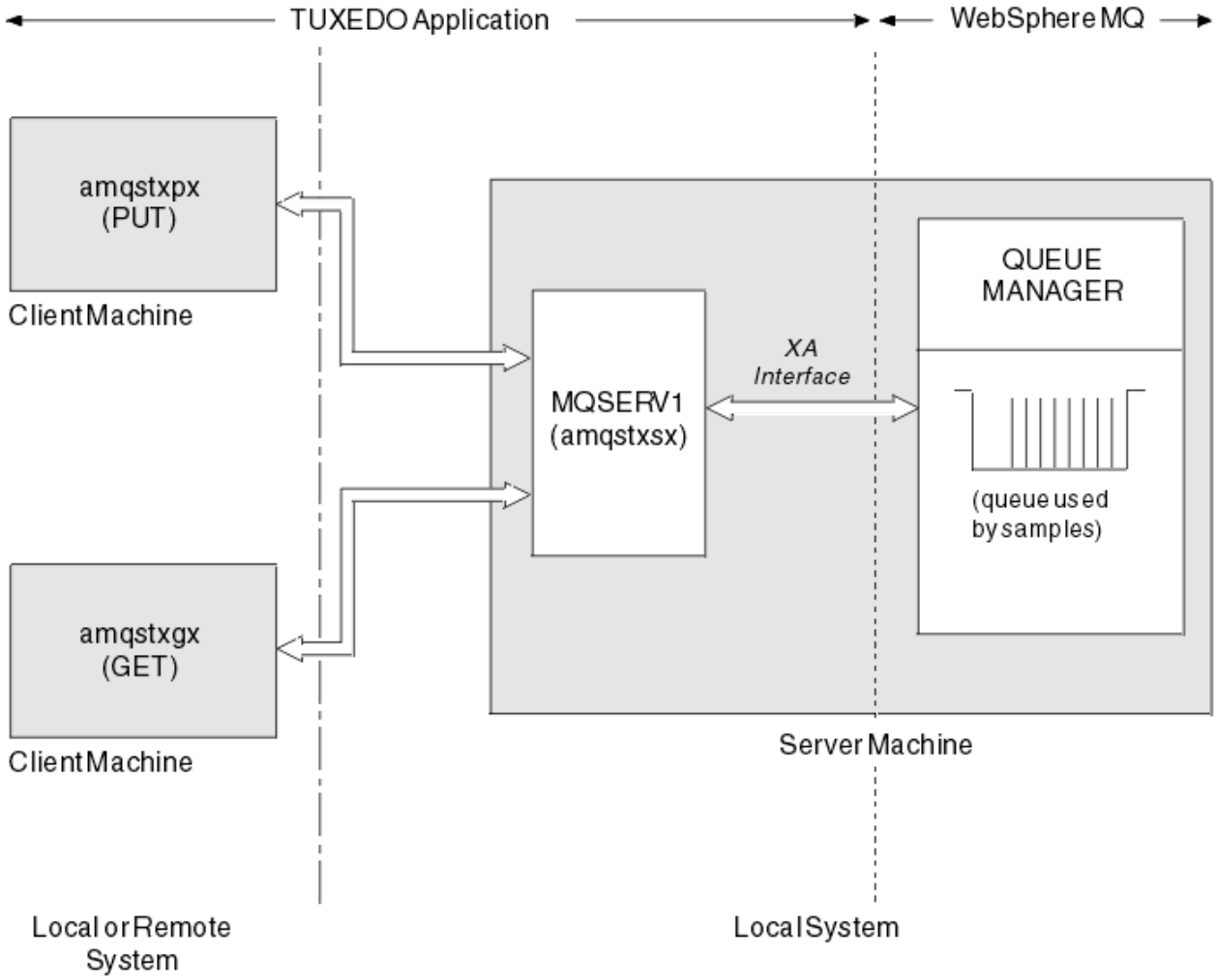


図 *28. TUXEDO* サンプルの処理の流れ

TUXEDO の書き込みサンプル・プログラム

このサンプルにより、キューに複数回メッセージを書き込むことができます。バッチでは、リソース・マネージャーとして TUXEDO を使用した同期点を示すことができます。

書き込みサンプルが正常に動作するには、サンプル・サーバー・プログラム `amqstxsx` が実行されていなければなりません。このサーバー・サンプル・プログラムはキュー・マネージャーへの接続を行い、XA インターフェースを使用します。 このサンプルを実行するには、次のように入力します。

- `doputs -n queuename -b batchsize -c trancount -t message`

以下に例を示します。

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

これによって、6 つのバッチでそれぞれ 5 個ずつ、合計で 30 個のメッセージを `myqueue` という名前のキューに入れることができます。 何らかの問題があれば、そのバッチ単位でメッセージを元に戻し、問題がなければ、それらのメッセージをコミットします。

エラー・メッセージはすべて、TUXEDO ログ・ファイルおよび stderr に書き込まれます。 理由コードはすべて、stderr に書き込まれます。

TUXEDO の読み取りサンプル

このサンプルを使用すると、バッチ内のキューからメッセージを取得することができます。

書き込みサンプルが正常に動作するには、サンプル・サーバー・プログラム amqstxsx が実行されていなければなりません。このサーバー・サンプル・プログラムはキュー・マネージャーへの接続を行い、XA インターフェースを使用します。 このサンプルを実行するには、次のように入力します。

- dogets -n queuename -b batchsize -c trancount

以下に例を示します。

- dogets -n myqueue -b 6 -c 4

これによって、myqueue という名前のキューから、6 つのバッチでそれぞれ 4 個ずつ、合計で 24 個のメッセージを取り出すことができます。 30 個のメッセージを myqueue に入れる書き込みサンプル・プログラムを実行したあとでこのサンプル・プログラムを実行すると、myqueue には 6 個のメッセージしか残りません。 バッチ数とバッチ・サイズがメッセージの書き込みと読み取りとで異なっても差し支えありません。

エラー・メッセージはすべて、TUXEDO ログ・ファイルおよび stderr に書き込まれます。 理由コードはすべて、stderr に書き込まれます。

Windows システムでの SSPI セキュリティー出口の使用

このトピックでは、Windows システムで SSPI チャネル出口プログラムを使用する方法について説明します。 付属する出口コードの形式は、オブジェクトとソースの 2 つです。

オブジェクト・コード

オブジェクト・コード・ファイルの名前は amqrspin.dll です。 これは、クライアントとサーバーの両方で、WebSphere MQ for Windows の標準部分として *MQ_INSTALLATION_PATH*\exits*INSTALLATION_NAME* フォルダーにインストールされます 例えば、C:\Program Files\IBM\WebSphere MQ\exits\installation2 などです。 これは、標準のユーザー出口としてロードされます。 提供されるセキュリティー・チャネル出口を実行して、チャネル定義にある認証サービスを使用することができます。

これを行うには、次のいずれかを指定してください。

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

制限付きチャネルのサポートを提供するには、SVRCONN チャネルで以下を指定します。

```
SCYDATA('remote_principal_name')
```

remote_principal_name は、DOMAIN¥user という形式になります。 ソース・チャネルは、リモート・プリンシパルの名前が *remote_principal_name* と一致する場合にのみ確立されます。

Kerberos セキュリティー・ドメイン内で稼働するシステム間で、提供されているチャネル出口プログラムを使用するには、キュー・マネージャー用に servicePrincipalName を作成してください。

ソース・コード

出口ソース・コード・ファイルの名前は amqsspin.c です。 これは C:\Program Files\IBM\WebSphere MQ\Tools\c\Samples にあります。

ソース・コードを変更するには、変更されたソースを再コンパイルしなければなりません。

このソースのコンパイルとリンクは、関連したプラットフォームの他のチャネル出口と同じように行います。ただし、SSPI ヘッダーはコンパイル時にアクセスする必要があり、SSPI セキュリティー・ライブラリーは関連した他の推奨ライブラリーとともにリンク時にアクセスする必要があります。

以下のコマンドを実行する前に、cl.exe、Visual C++ ライブラリー、および include フォルダーのパスが通っていることを確認します。 以下に例を示します。

```
cl /VERBOSE /LD /MT /I<path_to_Microsoft_platform_SDK\include>
/I<path_to_WebSphere MQ\tools\c\include> amqsspin.c /DSECURITY_WIN32
-link  /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

注 : ソース・コードにはトレースやエラー処理のためのものが組み込まれていません。 ソース・コードを変更して使用する場合、独自のトレースおよびエラー処理ルーチンを追加してください。

リモート・キューを使用するサンプルの実行

接続されたキュー・マネージャーでサンプルを実行することにより、リモート・キューイングを実証できます。

プログラム amqscos0.tst は、OTHER という名前のリモート・キュー・マネージャーを使用するリモート・キュー (SYSTEM.SAMPLE.REMOTE) のローカル定義を行います。 このサンプル定義を使用するには、OTHER を 2 番目に使用したいキュー・マネージャーの名前に変更します。 また、2 つのキュー・マネージャーの間のメッセージ・チャネルも設定する必要があります。この方法については、チャネルの定義を参照してください。

要求サンプル・プログラムは、それ自体のローカル・キュー・マネージャー名を、自らが送信するメッセージの *ReplyToQMgr* フィールドに入れます。 照会サンプルおよび設定サンプルは、この 2 つのサンプルが処理する要求メッセージの *ReplyToQ* フィールドおよび *ReplyToQMgr* フィールドで名前が指定されたキューおよびメッセージ・キュー・マネージャーに応答メッセージを送信します。

クラスター・キュー・モニターのサンプル・プログラム (AMQSCLM)

このサンプルでは、IBM WebSphere MQ に組み込まれたクラスター・ワークロード・バランシング機能を使用して、コンシューム側アプリケーションが接続されているキューのインスタンスにメッセージを送信します。 この自動送信により、コンシューム側アプリケーションが接続されていないクラスター・キューのインスタンスにメッセージが蓄積されることを防ぎます。

概要

複数のキュー・マネージャーで同じキューに対して異なる定義を使用するクラスターをセットアップできます。 この構成は、可用性およびワークロード・バランシングの改善という利点をもたらします。 ただし、IBM WebSphere MQ には、接続されているアプリケーションの状態に応じて動的にクラスター全体にメッセージを分散させる機能は組み込まれていません。 このことから、確実にメッセージが処理されるようにするには、常に、キューのあらゆるインスタンスにコンシューム側アプリケーションを接続しなければなりません。

クラスター・キュー・モニター・サンプル・プログラムは、接続されているアプリケーションの状態をモニターします。 このプログラムは、コンシューム側アプリケーションが接続されているクラスター・キューのインスタンスにメッセージを送信するために、組み込みワークロード・バランシング構成を動的に調整します。 特定の状況では、このプログラムを使用することにより、常にキューのあらゆるインスタンスにコンシューム側アプリケーションを接続しなければならないという必要性を緩和できます。 また、コンシューム側アプリケーションが接続されていないキューのインスタンスに入れられた状態のメッセージは再送されます。 メッセージの再送により、一時的にシャットダウンされているコンシューム側アプリケーションにもメッセージをルーティングできます。

このプログラムの使用が意図されているのは、コンシューム側アプリケーションが頻繁に接続されて切り離されるのではなく、長時間実行される場合です。

クラスター・キュー・モニター・サンプル・プログラムは、C サンプル・ファイル amqsclma.c のコンパイル済み実行可能プログラムです。

クラスターおよびワークロードについて詳しくは、クラスターによるワークロードの管理を参照してください。

AMQSCLM: サンプルを使用するための設計と計画

クラスター・キュー・モニター・サンプル・プログラムの動作、サンプル・プログラムを実行するシステムをセットアップするときに考慮すべき点、およびサンプル・ソース・コードをどのように変更できるかについて説明します。

デザイン

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションが接続されているローカル・クラスター・キューをモニターします。 プログラムは、ユーザーによって指定されたキューをモニターします。 キューの名前は、APP.TEST01 のように固有にすることも、または総称にすることもできます。 総称名は、PCF (プログラマブル・コマンド・フォーマット) に準拠する形式でなければなりません。 総称名の例としては、APP.TEST* や APP* などがあります。

クラスター内でモニター対象のローカル・キューのインスタンスを所有する各キュー・マネージャーには、クラスター・キュー・モニター・サンプル・プログラムの 1 つのインスタンスを接続する必要があります。

動的メッセージ・ルーティング

クラスター・キュー・モニター・サンプル・プログラムは、あるキューのコンシューマーがいるかどうかを、そのキューの **IPPROCS** 値 (入力処理カウント用に開かれる) を使用して判別します。 値が 0 より大きいときは、キューに少なくとも 1 つのコンシューム側アプリケーションが接続していることを示します。 この場合、キューはアクティブです。 値が 0 であるときは、キューに接続しているコンシューム側プログラムがないことを示します。 この場合、キューは非アクティブです。

クラスター内に複数のインスタンスがあるクラスター・キューの場合、WebSphere MQ では、各キュー・インスタンスのクラスター・ワークロード優先順位プロパティー **CLWLPRTY** を使用して、メッセージを送信する宛先のインスタンスが決定されます。 **CLWLPRTY** 値が最も大きい使用可能なキューのインスタンスに、WebSphere MQ はメッセージを送信します。

クラスター・キュー・モニター・サンプル・プログラムは、ローカル **CLWLPRTY** の値を 1 に設定することによって、クラスター・キューをアクティブ化します。 このプログラムは、**CLWLPRTY** 値を 0 に設定することにより、クラスター・キューを非アクティブ化します。

クラスター・キューの **CLWLPRTY** プロパティーが更新されると、WebSphere MQ クラスタリング・テクノロジーによって、この更新がクラスター内の関係するすべてのキュー・マネージャーに伝搬されます。 例:

- メッセージをキューに書き込むアプリケーションが接続されているキュー・マネージャー
- 同じクラスター内に同じ名前のローカル・キューを所有するキュー・マネージャー

この伝搬は、クラスターのフル・リポジトリー・キュー・マネージャーを使用して行われます。 クラスター・キューの新規メッセージは、クラスター内で最も大きい **CLWLPRTY** 値を持つインスタンスに送信されます。

既にキューに入れられたメッセージの転送

CLWLPRTY の値を動的に変更した場合、この変更は新規メッセージのルーティングに適用されます。 コンシューマーが接続されていないキュー・インスタンスのキューに既に入れられているメッセージ、または **CLWLPRTY** 値の変更がクラスター内に伝搬される前にワークロード・バランシング・メカニズムを経由したメッセージには、この動的な変更が適用されません。 その結果、非アクティブなキューにあるメッセージは、コンシューム側アプリケーションによって処理されることなく、そのまま残ってしまいます。 この問題を解決するために、クラスター・キュー・モニター・サンプル・プログラムでは、コンシューマーを持たないローカル・キューのメッセージを取得して、そのメッセージを同じキューのコンシューマーが接続されているリモート・インスタンスに送信できます。

クラスター・キュー・モニター・サンプル・プログラムは、一度メッセージを取得して (**MQGET** を使用する) からそのメッセージを同じクラスター・キューに書き込む (**MQPUT** を使用する) という方法で、非アクティブなローカル・キューのメッセージを 1 つ以上のアクティブなリモート・キューに転送します。 この方法で転送されることで、WebSphere MQ クラスター・ワークロード管理によって、ローカル・キュー・インスタンスよりも大きい CLWLPRTY 値を持つ別のターゲット・インスタンスが選択されます。 メッセー

ジが転送されても、メッセージのパーシスタンスとコンテキストは保持されます。 メッセージの順序とバインディング・オプションは保持されません。

計画

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションの接続性に変更があったとき、クラスター構成を変更します。 変更は、クラスター・キュー・モニター・サンプル・プログラムがキューをモニターしているキュー・マネージャーから、クラスター内のフル・リポジトリー・キュー・マネージャーに伝送されます。 フル・リポジトリー・キュー・マネージャーは構成の更新を処理し、その情報をクラスター内の関係するすべてのキュー・マネージャーに再送します。 関係するキュー・マネージャーに含まれるのは、同じ名前のクラスター・キューを所有するキュー・マネージャー (クラスター・キュー・モニター・サンプル・プログラムのインスタンスが実行中)、およびアプリケーションが過去 30 日間にクラスター・キューを開いてメッセージを書き込んだキュー・マネージャーです。

変更はクラスター内で非同期に処理されます。 そのため、変更が行われると、ある期間、クラスター内で認識される構成がキュー・マネージャーごとに異なるという状況も起こりえます。

クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションの接続と切り離しが頻繁には行われない、コンシューム側アプリケーションが長期間継続して実行されるようなシステムにのみ適しています。 コンシューム側アプリケーションが短期間だけ接続されるようなシステムのモニターに使用すると、構成更新の配布に要する待ち時間のために、コンシューマーが接続されているキューをクラスター内のキュー・マネージャー側で正しく認識できなくなってしまう可能性があります。 この待ち時間によって、メッセージが正しくルーティングされない事態も発生しかねません。

多数のキューをモニターしている場合、すべてのキューで、接続されているコンシューマーでの変更の速度が比較的遅くなり、これによりクラスター内のクラスター構成トラフィックが増加します。 クラスター構成トラフィックが増えると、以下の 1 つ以上キュー・マネージャーに過大な負荷がかかってしまうことがあります。

- クラスター・キュー・モニター・サンプル・プログラムを実行中のキュー・マネージャー
- フル・リポジトリー・キュー・マネージャー
- メッセージをキューに書き込むアプリケーションが接続されているキュー・マネージャー
- 同じクラスター内に同じ名前のローカル・キューを所有するキュー・マネージャー

フル・リポジトリー・キュー・マネージャーのプロセッサー使用量を評価する必要があります その他のプロセッサー使用量は、フル・リポジトリー・キュー SYSTEM.CLUSTER.COMMAND.QUEUE のメッセージ・トラフィックによって把握できます。 このキューにメッセージがたまっている場合は、フル・リポジトリー・キュー・マネージャーがシステムのクラスター構成変更の速度のペースに合わせることができていないことを示しています。

クラスター・キュー・モニター・サンプル・プログラムが多数のキューをモニターしている場合、サンプル・プログラムとキュー・マネージャーによって実行される一定量の作業があります。 この作業は、接続されているコンシューマーに何も変更がなくても実行されるものです。 **-i** 引数を変更して、モニターの周期の頻度を減らし、ローカル・システムでサンプル・プログラムのプロセッサー使用量を減らすことができます。

過大なアクティビティーを検出するための助けとして、クラスター・キュー・モニター・サンプル・プログラムはポーリング間隔ごとの平均処理時間、処理経過時間、および構成変更の数のレポートを報告します。 レポートは情報メッセージ **CLM0045I** として、30 分または 600 ポーリングのどちらか早いほうの間隔で送信されます。

クラスター・キュー・モニター使用の要件

クラスター・キュー・モニター・サンプル・プログラムには要件と制限があります。 提供されているサンプル・ソース・コードは、その使用方法に関する制限の一部を変更することができます。 このセクションに挙げた例で、加えることができる変更について詳しく説明します。

- クラスター・キュー・モニター・サンプル・プログラムは、コンシューム側アプリケーションが接続されている、またはされていない状態にあるキューをモニターする用途で設計されています。 システムでコンシューム側アプリケーションの接続と切り離しが頻繁に行われると、 サンプル・プログラムはクラス

ター全体に過大なクラスター構成アクティビティーを生じさせる可能性があります。 このことは、クラスター内のキュー・マネージャーのパフォーマンスに影響を与えかねません。

- クラスター・キュー・モニター・サンプル・プログラムは、基礎となる WebSphere MQ システムおよびクラスター・テクノロジーに依存しています。 モニターするキューの数、モニターの頻度、および各キューの状態が変化する頻度は、システム全体にかかる負荷に影響します。 モニターするキューおよびモニターのポーリング間隔を選択する際には、これらの要素を考慮してください。
- クラスター内のモニター対象のキューのインスタンスを所有するすべてのキュー・マネージャーには、クラスター・キュー・モニター・サンプル・プログラムの 1 つのインスタンスが接続している必要があります。 サンプル・プログラムを、クラスター内のキューを所有しないキュー・マネージャーに接続する必要はありません。
- クラスター・キュー・モニター・サンプル・プログラムを適切な権限で実行して、必要な WebSphere MQ リソースすべてにアクセスできるようにします。 例:
  - 接続先のキュー・マネージャー
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - メッセージ転送を実行するときにモニターするすべてのキュー
- コマンド・サーバーは、各キュー・マネージャーに対して、クラスター・キュー・モニター・サンプル・プログラムが接続された状態で実行する必要があります。
- クラスター・キュー・モニター・サンプル・プログラムの各インスタンスが接続先のキュー・マネージャー上のローカル (非クラスター) キューを排他使用できることが必要です。 このローカル・キューは、サンプル・プログラムの制御、およびキュー・マネージャーのコマンド・サーバーに対して実行した照会の応答メッセージを受信するために使用されます。
- クラスター・キュー・モニター・サンプル・プログラムの単一インスタンスがモニターするすべてのキューは、同一のクラスター内になければなりません。 キュー・マネージャーが持つモニター対象のキューが複数のクラスターに分散している場合は、サンプル・プログラムの複数インスタンスが必要です。 それらの各インスタンスに、制御および応答メッセージ用のローカル・キューが必要です。
- モニターするキューはすべてが単一のクラスター内になければなりません。 クラスター名前リストを使用するように構成されたキューは、モニターされません。
- 非アクティブ・キューからのメッセージ転送を有効にするかどうかはオプションです。 この指定は、クラスター・キュー・モニター・サンプル・プログラムの特定のインスタンスがモニターするすべてのキューに適用されます。 モニター対象のキューの一部だけにメッセージ転送を有効にする必要がある場合は、クラスター・キュー・モニター・サンプル・プログラムのインスタンスが 2 つ必要になります。 一方のサンプル・プログラムでメッセージ転送を有効にし、もう一方のサンプル・プログラムでメッセージ転送を無効にします。 このとき、サンプル・プログラムの各インスタンスに、制御および応答メッセージ用のローカル・キューが必要です。
- デフォルトでは、WebSphere MQ クラスター・ワークロード・バランシングが、書き込みアプリケーションが接続されている同じキュー・マネージャーにある、クラスター・キューのインスタンスにメッセージを送信します。 次の環境で、ローカル・キューが非アクティブである間は、この機能を無効にしてください。
  - 書き込みアプリケーションが、モニター対象である非アクティブなキューのインスタンスを所有するキュー・マネージャーに接続されている
  - キューに入れられたメッセージが、非アクティブなキューのからアクティブなキューに転送中である

  CLWLUSEQ 値を ANY に設定することにより、キューのワークロード・バランシングのローカル設定を静的に無効にすることができます。 この構成の場合、ローカルのコンシューム側アプリケーションがあっても、ローカル・キューに書き込まれたメッセージをローカル・キュー・インスタンスとリモート・キュー・インスタンスに分散してワークロードのバランスを取ります。 または、クラスター・キュー・モニター・サンプル・プログラムの構成を一時的に変更し、コンシューマーがキューに接続されていない間、**CLWLUSEQ** 値を ANY に設定することもできます。結果として、あるキューがアクティブである間、ローカル・メッセージだけがそのキューのローカル・インスタンスに送信されます。
- WebSphere MQ システムおよびアプリケーションが、モニター対象のキューまたは使用中のチャネルに **CLWLPRTY** を使用することはできません。 使用した場合は、**CLWLPRTY** キュー属性に対するクラスタ

ー・キュー・モニター・サンプル・プログラムのアクションによって、望ましくない結果が生じる可能性があります。

- クラスター・キュー・モニター・サンプル・プログラムは、ランタイム情報をログに記録して、一式のレポート・ファイルを作成します。 これらのレポートを保管するディレクトリーが必要であり、このディレクトリーに書き込む権限をクラスター・キュー・モニター・サンプル・プログラムが持っている必要があります。

AMQSCLM: サンプルの準備と実行

クラスター・キュー・モニター・サンプルを実行するには、クライアント・モードで実行されているアプリケーションからの着信接続要求を安全に受け入れるよう、キュー・マネージャーを構成する必要があります。

始める前に

クラスター・キュー・モニター・サンプルを実行する前に、以下のステップを実行する必要があります。

1. サンプルの内部使用のために、それぞれのキュー・マネージャーで作業キューを作成します。

   サンプルのそれぞれのインスタンスには、排他的な内部使用のために、ローカルの非クラスター・キューが必要です。 キューの名前は選択できます。 次の例では、名前は AMQSCLM.CONTROL.QUEUE を使用します。 例えば、Windows の場合、このキューは次の **MQSC** コマンドを使用して作成できます。

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   MAXDEPTH と **MAXMSGL** の値はデフォルトのまま使用できます。

2. エラー・ログと情報メッセージ・ログのためにディレクトリーを作成します。

   サンプルは、レポート・ファイルに診断メッセージを書き込みます。 ユーザーは、それらのファイルを保管するディレクトリーを選択する必要があります。 例えば、Windows の場合は、次のコマンドを使用してディレクトリーを作成できます。

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   サンプルが作成するレポート・ファイルには、次の命名規則があります。

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (オプション) クラスター・キューモニター・サンプルを IBM WebSphere MQ サービスとして定義します。

   キューをモニターするには、サンプルは常に実行されている必要があります。 クラスター・キュー・モニター・サンプルが常に実行されるようにするには、サンプルをキュー・マネージャー・サービスとして定義します。 サンプルをサービスとして定義するということは、キュー・マネージャーが開始すると AMQSCLM も開始することを意味します。 次の **RUNMQSC** の例を使用すると、クラスター・キュー・モニター・サンプルを IBM WebSphere MQ サービスとして定義できます。

   ```
   define service(AMQSCLM) +
       descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
       control(qmgr) +
       servtype(server) +
       startcmd('<Install Root>\tools\c\samples\Bin\AMQSCLM.exe') +
    startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l c:\AMQSCLM\rpts') +
    stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
    stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
   ```

   ここで、<Install Root> は、インストール済み環境の場所を表します。

定義	説明
service	サービス名を指定します。 サービス名は選択できます。
descr	サービスのテキスト記述を指定します。

定義	説明
control	サービスがキュー・マネージャーと同時に開始および停止することを示します。
servtype	このキュー・マネージャーに対して、一度に 1 つのサーバー・サービス・オブジェクト (つまり、1 つのインスタンスのみ) が実行可能であることを示します。
startcmd	プログラムの場所と名前を指定します。
startarg	サンプルの引数を指定します。 *+QMNAME+* が使用されているため、キュー・マネージャーの名前は自動的に置き換わります。
stdout	標準の出力がリダイレクトされる完全修飾ファイル名です。 サンプルがこのファイルに書き込むのは、サンプルが終了したことを確認するメッセージのみです。 サンプルでこれを実行しているのは、サンプル終了プロセスの初期段階で、標準エラー・ファイルが既に閉じられているためです。
stderr	標準のエラー出力がリダイレクトされる完全修飾ファイル名です。 サンプルは標準エラー・ファイルに、サンプルが終了する前のエラー・メッセージを書き込みます。

このタスクについて

このタスクでは、さまざまな方法でクラスター・キュー・モニター・サンプルを開始および停止できます。また、モニター対象のキューの統計情報を含むレポート・ファイルを生成するモードで、サンプルを実行することもできます。

サンプル・プログラムは、次のコマンドを使用して実行できます。

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask | -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

次の表では、クラスター・キュー・モニター・サンプルに使用できる引数とその詳細をリストします。

引数	変数	詳細
-m	**QMgrName**	モニター対象のキュー・マネージャー。
-c	**ClusterName**	モニター対象のキューが含まれるクラスター。
-q	**QNameMask**	モニター対象のキュー (複数の場合あり)。 末尾の * は、0 文字以上の末尾文字と一致する名前を持つすべてのキューをモニターします。
-f	**QListFile**	モニター対象のキュー名マスクのキュー名のリストを含む、ファイルの絶対パスおよびファイル名。 ファイルの各行には 1 つのキュー名/マスクが含まれている必要があります。 **-q** または **-f** のいずれかを指定できますが、両方は指定できません。
-r	**MonitorQName**	サンプルが排他的に使用しているローカル・キュー。
-l	**ReportDir**	ログに記録された情報メッセージを一連のラッピング < fn> に保管するディレクトリー・パス。キュー・マネージャーとキューの組み合わせごとに、特定のサイズで上限が設定されたレポート・ファイルが生成されます。 ロガーは常に同じファイルに書き込みますが、ファイルの以前の 2 つのバージョンも保持します。 < /fn> レポート・ファイル。
-t		(オプション) キューに入れられたメッセージを、非アクティブなローカル・キューからアクティブなキューに転送できるようにします。 転送を有効にしないと、クラスターに入れられる新規のメッセージのみが、キューのアクティブなインスタンスに動的に経路指定されます。
-u	**ActiveVal**	(オプション) モニター対象のキュー・インスタンスが非アクティブな場合はその **CLWLUSEQ** プロパティーを ANY に、アクティブな場合は **ActiveVal** プロパティーに自動的に切り替えます。 **ActiveVal** には、LOCAL または QMGR のいずれかを指定できます。 この引数が、書き込み側のアプリケーションが同じキュー・マネージャーに接続されているシステム、またはメッセージの転送が有効なシステムで設定されていない場合、モニター対象のキューは **CLWLUSEQ** の値が ANY に設定されているか、QMGR に設定されていて ANY に設定されているキュー・マネージャーを使用する必要があります。
-i	**Interval**	(オプション) モニターがキューを確認する時間間隔 (秒)。 デフォルトは 300 秒 (5 分) です。
-d		(オプション) 追加の診断が出力されるようにします。 デバッグ出力は、システムを最初に構成するとき、またはサンプル・コードを操作するときに役立つ場合があります。
-s		(オプション) 各間隔についての最小限の統計情報が出力されるようにします。
-v		(オプション) レポート・ファイルに加えて、レポート情報を standard out に記録します。

引数リストの例:

```
-m QMGR1 -c CLUS1 -f c:\QList.txt -r CLMQ -l c:\amqsclm\rpts -s
-m QMGR2 -c CLUS1 -q ABC* -r CLMQ -l c:\amqsclm\rpts -i 600
-m QMGR1 -c CLUSDEV -q QUEUE.* -r CLMQ -l c:\amqsclm\rpts -t -u QMGR -d
```

キュー・リスト・ファイルの例:

```
Q1
QUEUE.*
ABC
ABD
```

手順

1. クラスター・キュー・モニター・サンプルを開始します。 サンプルは、以下のいずれかの方法で開始できます。
   - 適切なユーザー権限でコマンド・プロンプトを使用する。
   - サンプルが IBM WebSphere MQ サービスとして構成されている場合は、MQSC **START SERVICE** コマンドを使用します。

   引数リストは、どちらの場合も同じです。

   サンプルは、プログラムが初期化されてから 10 秒間はキューのモニターを開始しません。 コンシューマー・アプリケーションは、この遅延によって最初にモニター対象のキューに接続できるため、キューをアクティブな状態に変更する無駄を回避できます。
2. クラスター・キュー・モニター・サンプルを停止します。 サンプルは、キュー・マネージャーが停止したり、停止中または静止中であったり、キュー・マネージャーへの接続が切断されたりすると自動的に停止します。 サンプルは、以下の方法で、キュー・マネージャーを終了させることなく停止することができます。
   - サンプルが排他的に使用しているローカル・キューを、Get 機能が使用できないように構成する。
   - サンプルが排他的に使用しているローカル・キューに、**CorrelId** が "STOP CLUSTER MONITOR¥0¥0¥0¥0" のメッセージを送信する。
   - サンプル・プロセスを終了する。 これは、アクティブなキューに転送されている非永続メッセージの損失をもたらす可能性があります。 また、これにより、サンプルが使用しているローカル・キューが、終了後の数秒間オープンの状態になる可能性があります。 この状況では、クラスター・キュー・モニター・サンプルの新規のインスタンスは、直ちに開始することができません。

   サンプルが IBM WebSphere MQ サービスとして開始された場合、**STOP SERVICE** は何の効果も及ぼしません。 その場合は、キュー・マネージャー内の構成済みの **STOP SERVICE** メカニズムとして記述される終了メソッドの 1 つを使用できます。

次のタスク

サンプルの状況を確認します。

レポート作成を有効にした場合は、レポート・ファイルで状況を確認できます。 最新のレポート・ファイルを確認するには、次のコマンドを使用します。

```
QMgrName.ClusterName.RPT01.LOG
```

古いレポート・ファイルを確認するには、次のコマンドを使用します。

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

レポート・ファイルは、最大サイズ (約 1 MB) まで拡大します。 RPT01 ファイルがいっぱいになると、新しい RPT01 ファイルが作成されます。 古い RPT01 ファイルは、RPT02 に名前変更されます。 RPT02 は、RPT03 に名前変更されます。 古い RPT03 は破棄されます。

サンプルは、以下の状況で情報メッセージを作成します。

- 開始時
- 終了時
- キューを、**ACTIVE** または **INACTIVE** とマーク付けするとき
- メッセージを、非アクティブなキューからアクティブなインスタンスにリキューするとき

サンプルは、注意が必要な問題を報告する場合は、エラー・メッセージ *CLMnnnnE* を作成します。

サンプルは、30 分おきに、各ポーリング間隔の平均処理時間と経過した処理時間を報告します。 この情報は、メッセージ CLM0045I に含まれます。

統計メッセージが有効 (**-s**) な場合、サンプルは、それぞれのキュー・チェックについて以下の情報を報告します。

- キューの処理にかかった時間 (ミリ秒)
- 確認されたキューの数
- アクティブ/非アクティブへの変更の数
- 転送されたメッセージの数

この情報は、メッセージ CLM0048I で報告されます。

デバッグ・モードでは、レポート・ファイルは急速に拡大して、短期間で循環する場合があります。 この状況では、ファイルは 1 MB のサイズ制限を超える場合があります。

AMQSCLM: トラブルシューティング

以下のセクションでは、サンプルの使用時に起こる可能性のあるシナリオについて記載します。 各シナリオについての可能性のある説明、およびその解決方法の選択肢について説明します。

シナリオ: AMQSCLM が開始しない

可能性のある説明: 構文が正しくない。

アクション: 標準のエラー出力に正しい構文が書き込まれているか確認してください。

可能性のある説明: キュー・マネージャーが使用可能ではない。

アクション: レポート・ファイルに、メッセージ ID CLM0010E が書き込まれていないか確認してください。

可能性のある説明: レポート・ファイル (複数の場合あり) を開けない、または作成できない。

アクション: 標準のエラー出力に、初期化時のエラー・メッセージが書き込まれていないか確認してください。

シナリオ: AMQSCLM がキューを ACTIVE または INACTIVE に変更しない

可能性のある説明: キューがモニター対象のキューのリストにない。

アクション: **-q** および **-f** パラメーターの値を確認してください。

可能性のある説明: キューが適切なクラスター内のローカル・キューではない。

アクション: キューが適切なクラスター内のローカル・キューであることを確認してください。

可能性のある説明: AMQSCLM がこのキュー・マネージャーとクラスターに対して実行されていない。

アクション: AMQSCLM を、関連するキュー・マネージャーとクラスターに対して開始してください。

可能性のある説明: キューにはコンシューマーが存在しないため、キューは INACTIVE、**CLWLPRTY**=0 の状態である。 あるいは、少なくとも 1 つのコンシューマーがあるため、ACTIVE **CLWLPRTY**> =1 のままです。

アクション: コンシューマー・アプリケーションがキューに接続されていることを確認してください。

可能性のある説明: キュー・マネージャーのコマンド・サーバーが実行されていない。

アクション: レポート・ファイルにエラーが書き込まれていないか確認してください。

シナリオ: メッセージが INACTIVE キューに対して経路指定されていない

可能性のある説明: メッセージが非アクティブなキューを所有するキュー・マネージャーに直接書き込まれている。また、キューの **CLWLUSEQ** 値が ANY ではなく、AMQSCLM に対して **-u** 引数が使用されていない。

アクション: 関連するキュー・マネージャーの **CLWLUSEQ** 値を確認するか、AMQSCLM に対して **-u** 引数が使用されるようにしてください。

可能性のある説明: いずれのキュー・マネージャーにもアクティブなキューがない。 キューがアクティブになるまで、メッセージに対してすべての非アクティブなキューの間での均一なワークロード・バランシングが行われます。

アクション: すべてのキュー・マネージャーのキューの状況を確認してください。

可能性のある説明: メッセージが非アクティブなキューを所有するクラスター内の異なるキュー・マネージャーに書き込まれている。また、更新された **CLWLPRTY** 値 0 が、書き込み側のアプリケーションのキュー・マネージャーに伝搬されていない。

アクション: モニター対象のキュー・マネージャーとフル・リポジトリー・キュー・マネージャー間のクラスター・チャネルが実行されていることを確認してください。 また、書き込み側のキュー・マネージャーとフル・リポジトリー・キュー・マネージャー間のチャネルが実行されていることを確認してください。 さらに、モニター対象のキュー・マネージャー、書き込み側のキュー・マネージャー、およびフル・リポジトリー・キュー・マネージャーのエラー・ログを確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブである (CLWLPRTY=1) が、ローカル・キュー・マネージャーからのクラスター送信側チャネルが実行されていないため、それらのキュー・インスタンスにメッセージを経路指定できない。

アクション: アクティブなキュー・インスタンスがあるリモート・キュー・マネージャー (1 つ以上) への、ローカル・キュー・マネージャーからのクラスター送信側チャネルの状況を確認してください。

シナリオ: AMQSCLM が非アクティブなキューからメッセージを転送しない

可能性のある説明: メッセージの転送が有効 (**-t**) になっていない。

アクション: メッセージの転送が有効 (**-t**) になっていることを確認してください。

可能性のある説明: キューがモニター対象のキューのリストにない。

アクション: **-q** および **-f** パラメーターの値を確認してください。

可能性のある説明: AMQSCLM がこのキュー・マネージャー、または同じキューのインスタンスを所有するクラスター内の他のキュー・マネージャーに対して実行されていない。

アクション: AMQSCLM を開始してください。

可能性のある説明: キューが **CLWLUSEQ**=LOCAL または **CLWLUSEQ**=QMGR を持っている。また、**-u** 引数が設定されていない。

アクション: **-u** パラメーターを設定するか、キューまたはキュー・マネージャーの構成を ANY に変更してください。

可能性のある説明: クラスター内にキューのアクティブなインスタンスがない。

アクション: **CLWLPRTY** 値が 1 以上のキューのインスタンスがあるか確認してください。

可能性のある説明:リモート・キュー・インスタンスにはコンシューマー (**IPPROCS**> = 1) がありますが、AMQSCLM がそれらのリモート・インスタンスをモニターしていないため、それらのキュー・マネージャーでは非アクティブです (**CLWLPRTY**= 0)。

アクション: AMQSCLM がそれらのキュー・マネージャー上で実行されていることを確認してください。さらに (あるいは)、**-q** および **-f** パラメーターの値を確認して、キューがモニター対象のキューのリストにあることを確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブ (**CLWLPRTY**=1) だが、ローカル・キュー・マネージャーでは非アクティブ (**CLWLPRTY**=0) と思われる。 この状況は、更新された **CLWLPRTY** の値が、このキュー・マネージャーに伝搬されていないことが原因で発生しています。

アクション: リモート・キュー・マネージャーが、クラスター内の少なくとも 1 つのフル・リポジトリー・キュー・マネージャーに接続されていることを確認してください。 また、フル・リポジトリー・キュー・マネージャーが正常に機能していることを確認してください。 さらに、フル・リポジトリー・キュー・マネージャーとモニター対象のキュー・マネージャー間のチャネルが実行されていることを確認してください。

可能性のある説明: メッセージがコミットされていないため、取得可能ではない。

アクション: 送信側のアプリケーションが正常に機能していることを確認してください。

可能性のある説明: AMQSCLM には、メッセージが入れられるローカル・キューへのアクセス権限がない。

アクション: このシナリオは、AMQSCLM が、キューに対する十分なアクセス権限を持つユーザーとして実行されていないことが原因である可能性があります。

可能性のある説明: キュー・マネージャーのコマンド・サーバーが実行されていない。

アクション: キュー・マネージャーのコマンド・サーバーを開始してください。

可能性のある説明: AMQSCLM にエラーが発生した。

アクション: レポート・ファイルにエラーが書き込まれていないか確認してください。

可能性のある説明: リモート・キュー・インスタンスはアクティブである (CLWLPRTY=1) が、ローカル・キュー・マネージャーからのクラスター送信側チャネルが実行されていないため、それらのキュー・インスタンスにメッセージを転送できない。 この場合は、amqsclm レポート・ログ内に CLM0030W 警告が同時に記録されていることがよくあります。

アクション: アクティブなキュー・インスタンスがあるリモート・キュー・マネージャー (1 つ以上) への、ローカル・キュー・マネージャーからのクラスター送信側チャネルの状況を確認してください。

接続エンドポイント検索 (CEPL) のサンプル・プログラム

IBM WebSphere MQ の接続エンドポイント検索サンプルは、 WebSphere MQ ユーザーが Tivoli Directory Server などの LDAP リポジトリーから接続定義を取得する手段を提供する、シンプルかつ強力な出口モジュールを備えています。

CEPL を使用するには、Tivoli Directory Server v6.3 クライアントがインストールされている必要があります。

このサンプルを使用するには、サポート対象のプラットフォームで WebSphere MQ を管理するための実用的な知識が必要です。

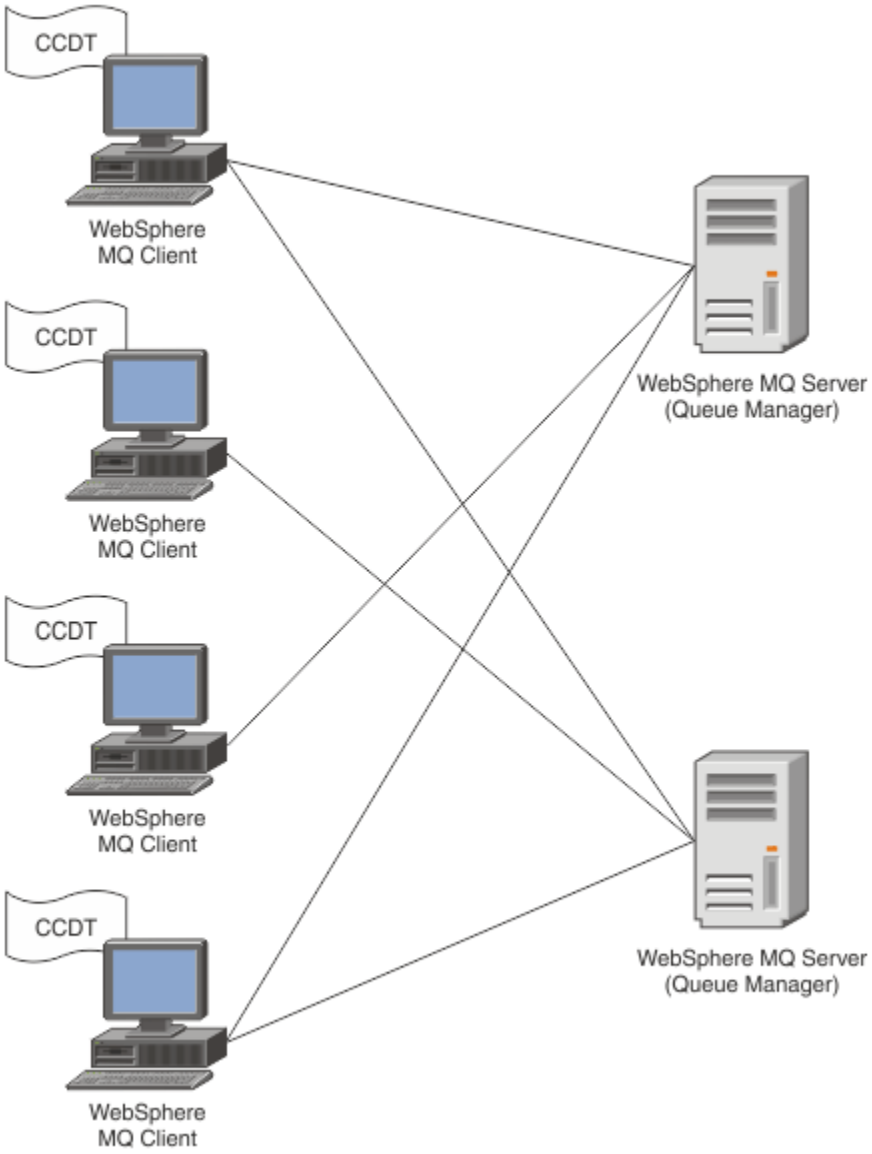
概要

保守および管理を補助するクライアント接続定義を保管するため、LDAP (Lightweight Directory Access Protocol) ディレクトリーなどのグローバル・リポジトリーを構成します。

IBM WebSphere MQ クライアント・アプリケーションを使用して、クライアント接続定義テーブル (CCDT) を介し、キュー・マネージャーへの接続を確立します。

CCDT の作成には、標準的な WebSphere MQ MQSC 管理インターフェースを使用します。 クライアント接続定義を作成するには、定義内に含まれるデータがキュー・マネージャーに制限されていなくても、ユーザーはキュー・マネージャーに接続する必要があります。 生成される CCDT ファイルは、クライアント・

マシンおよびアプリケーションに手動で配布する必要があります。

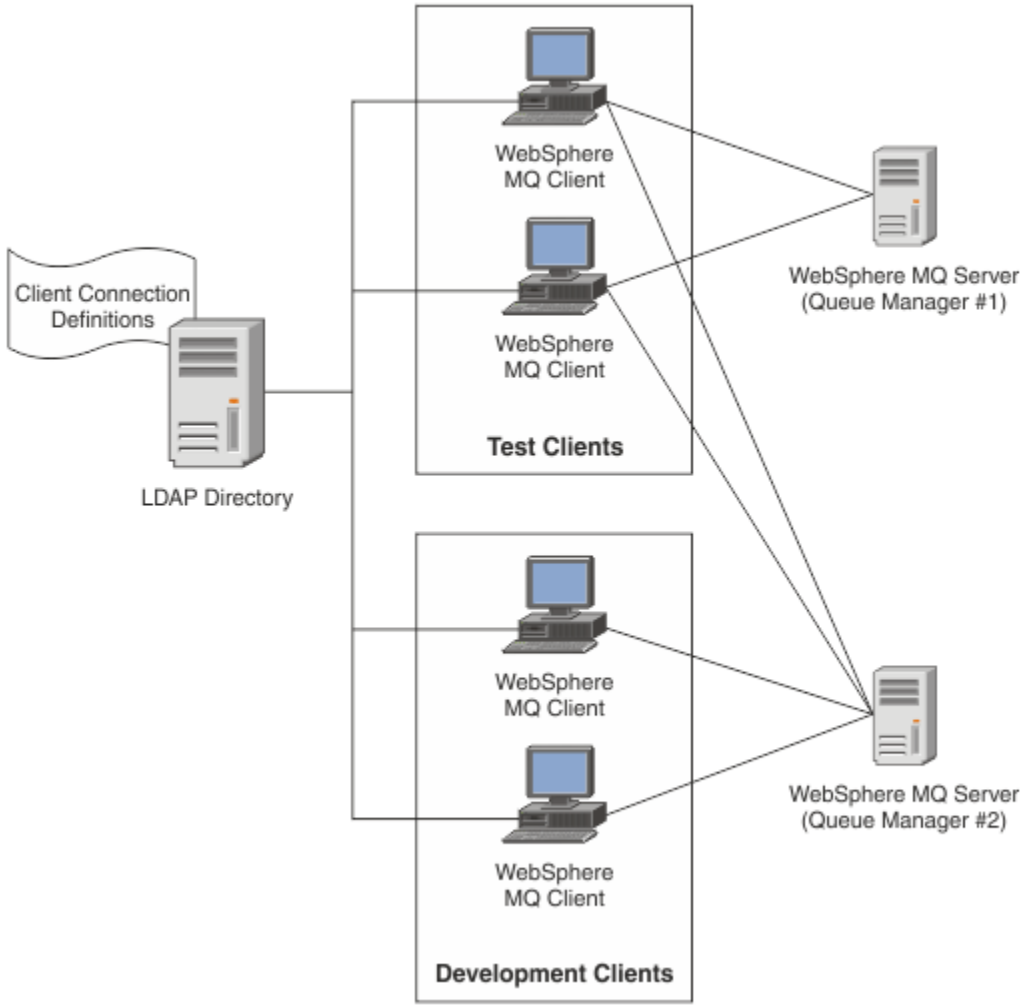


CCDT ファイルは、各 WebSphere MQ クライアントに手動で配布する必要があります。 何千ものクライアントがローカルにもグローバルにも存在できるような場合は、保守や管理がすぐに煩雑になります。 各クライアントに、そこで使用できる適切なクライアント定義が確実にあるようにするには、より柔軟なアプローチが必要です。

そのようなアプローチの 1 つとして、クライアント接続定義を LDAP (Lightweight Directory Access Protocol) ディレクトリーのようなグローバル・リポジトリーに保管することが挙げられます。 LDAP ディレクトリーには、追加のセキュリティー機能、索引付け機能、および検索機能もあるので、それを使用して各クライアントは自分に関連のある接続定義にのみアクセスできます。

LDAP ディレクトリーは、特定のユーザー・グループが特定の定義のみを使用できるように構成できます。 例えば、開発クライアントはキュー・マネージャー #2 にのみアクセス可能なのに対し、テスト・クライア

ントはキュー・マネージャー #1 と #2 の両方にアクセスできます。



出口モジュールは、LDAP リポジトリー (例えば IBM Tivoli Directory Server) を検索して、チャネル定義を取り出すことができます。 これらの接続定義を使用して、WebSphere MQ クライアント・アプリケーションはキュー・マネージャーへの接続を確立できます。

出口モジュールは、MQCONN/MQCONNX 呼び出しの際にチャネル定義を LDAP リポジトリーから取得できるようにする、接続前出口モジュールです。

出口モジュールおよびスキーマは、以下のお客様が実装できます。

- 既存の CCDT ファイル・ベース・テクノロジーを使用するスキルの基礎が既にあり、管理および配布のコストを軽減したいお客様。
- クライアント接続定義を配布するための独自の適切なテクノロジーを既に採用している、既存のお客様。
- 現在クライアント接続ソリューションを何も採用しておらず、IBM WebSphere MQ が提供するフィーチャーを使用したい、新規または既存のお客様。
- 現行の LDAP ビジネス・アーキテクチャーに即してメッセージング・モデルを直接使用または調整したい、新規または既存のお客様。

サポートされる環境

接続エンドポイント検索サンプルを実行する前に、サポート対象オペレーティング・システムと関連ソフトウェアがあることを確認してください。

IBM WebSphere MQ 接続エンドポイント検索のサンプル・プログラムには、以下のソフトウェアが必要です。

- IBM WebSphere MQ V7.0 以降

- Tivoli Directory Server クライアント V6.3 以降

サポートされるオペレーティング・システム:

1. Windows (XP/2003/2008)
2. Solaris (SPARC および x86-64)
3. AIX
4. Linux
   - RHEL v4 および v5 (System p)
   - SUSE v9 および v10 (System p)
   - RHEL v4 および v5 (System x32 ビットおよび x64 ビット)
   - SUSE v9 および v10 (System x32 ビットおよび x64 ビット)
5. HP IA64

注 : サンプル・プログラムは、z/OS、i/5、および HP PARISC プラットフォームでは使用できません。

インストールと構成

出口モジュールと接続エンドポイント・スキーマのインストールと構成。

出口モジュールのインストール

WebSphere MQ のインストール時、出口モジュールは `tools/samples/c/preconnexit/bin` にインストールされます。 32 ビット・プラットフォームの場合、出口モジュールを使用するには、その前に出口モジュールを `exit/<install name>/` にコピーする必要があります。 64 ビット・プラットフォームで出口モジュールを使用するには、その前に出口モジュールを exit64/<installation name>/ にコピーしておく必要があります。

接続エンドポイント・スキーマのインストール

出口は接続エンドポイント・スキーマ *ibm-amq.schema* を使用します。 出口を使用するには、その前に任意の LDAP サーバーにスキーマ・ファイルをインポートしておく必要があります。 スキーマのインポート後は、属性の値を追加する必要があります。

以下は、接続エンドポイント・スキーマのインポート方法の例です。 この例は、IBM Tivoli Directory Server (ITDS) を使用することを前提としています。

- IBM Tivoli Directory Server が実行されていることを確認し、ITDS サーバーに *ibm-amq.schema* ファイルをコピーするか FTP で転送します。
- ITDS サーバーで次のコマンドを入力して、スキーマを ITDS ストアにインストールします。LDAP ID と LDAP パスワードは、それぞれ LDAP サーバーのルート DN とパスワードです。

  ldapadd -D "LDAP ID" -w "LDAP password" -f ibm-amq.schema
- コマンド・ウィンドウで次のコマンドを入力するか、第三者製のツールでスキーマを参照して、スキーマを確認します。

  ldapsearch objectclass=ibm-amqClientConnection

スキーマ・ファイルのインポートについて詳しくは、LDAP サーバーの資料を参照してください。

構成

クライアント構成ファイル (例えば、 *mqclient.ini* ファイル) には、「**PreConnect**」という新しいセクションを追加する必要があります。 PreConnect セクションには、以下のキーワードが含まれています。

Module : API 出口コードを含むモジュールの名前。 このフィールドにモジュールの絶対パスを指定すると、そのモジュールが使用されます。その他の場合は、WebSphere MQ のインストール済み環境にある *exit* または *exit64* フォルダーが検索されます。

Function : PreConnect 出口コードを含むライブラリーへの関数のエントリー・ポイントの名前。 関数定義は、MQ_PRECONNECT_EXIT プロトタイプに従います。

Data : チャネル定義を含む LDAP リポジトリーの URI。

次のスニペットは、*mqclient.ini* ファイルに必要な変更の例です。

```
PreConnect:
  Module=amqlcelp
  Function=PreConnectExit
  Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
  Sequence=1
```

出口およびスキーマの概要

キュー・マネージャーへの接続を確立するために使用される構文およびパラメーター

WebSphere MQ v7.5 は、出口モジュールのエントリー・ポイントとして以下の構文を定義します。

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP  pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

MQCONN/X 呼び出しの実行中、WebSphere MQ C クライアントは関数構文の実装を含む出口モジュールをロードします。 その後、チャネル定義を検索するために出口関数が起動されます。 取り出されたチャネル定義が、キュー・マネージャーへの接続を確立するために使用されます。

パラメーター

pExitParms

タイプ: PMQNXP 入出力

PreConnection 出口パラメーター構造体。 この構造体は、出口の呼び出し側によって割り振られて維持されます。

```
struct tagMQNXP
{
  MQCHAR4    StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    ExitId;             /* Type of exit */
  MQLONG    ExitReason;         /* Reason for invoking exit */
  MQLONG    ExitResponse;       /* Response from exit */
  MQLONG    ExitResponse2;      /* Secondary response from exit */
  MQLONG    Feedback;           /* Feedback code (reserved) */
  MQLONG    ExitDataLength;        /* Exit data length */
  PMQCHAR      pExitDataPtr;           /* Exit data */
  MQPTR        pExitUserAreaPtr;       /* Exit user area */
  PMQCD *    ppMQCDArrayPtr;        /* Array of pointers to MQCDs */
  MQLONG    MQCDArrayCount;        /* Number of entries found */
  MQLONG    MaxMQCDVersion;        /* Maximum MQCD version */
};
```

pQMgrName

タイプ: PMQCHAR 入出力

キュー・マネージャーの名前。 入力において、このパラメーターは、**QMgrName** パラメーターを介して MQCONN API 呼び出しに提供されたフィルター・ストリングです。 このフィールドはブランクであるか、内容が明示的に指定されているか、特定のワイルドカード文字が含まれる場合があります。 このフィールドは出口によって変更されます。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。

ppConnectOpts

タイプ: ppConnectOpts 入出力

MQCONNX のアクションを制御するオプション。 これは、MQCONN API 呼び出しのアクションを制御する MQCNO 接続オプション構造へのポインターです。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。 MQCNO 構造がアプリケーションによって元々提供されなかった場合を含め、MQI クライアントは常に MQCNO 構造を出口に提供します。 アプリケーションが MQCNO 構造を提供した場合、 クライアントは複製を作成して出口に渡し、そこで構造が変更されます。 クライアントは MQCNO の所有権を保持します。 MQCNO を介して参照される MQCD は、配列を

介して提供されたすべての接続定義より優先されます。 クライアントはキュー・マネージャーに接続するために MQCNO 構造体を使用し、その他の定義は無視されます。

pCompCode
タイプ: PMQLONG 入出力

完了コード 出口完了コードを受け取る MQLONG へのポインター。 値は、次のいずれかでなければなりません。

MQCC_OK - 正常終了。

MQCC_WARNING - 警告 (部分的に完了)

MQCC_FAILED - 呼び出しの失敗。

pReason
タイプ: PMQLONG 入出力

pCompCode を修飾する理由。 出口理由コードを受け取る MQLONG へのポインター。 完了コードが MQCC_OK の場合、以下の値だけが有効です。

MQRC_NONE - (0, x'000') 報告する理由はありません。

完了コードが MQCC_FAILED または MQCC_WARNING の場合、出口関数は理由コード・フィールドを任意の有効な MQRC_* 値に設定できます。

MQ LDAP コンテキスト情報
出口は、コンテキスト情報に以下のデータ構造体を使用します。

MQNLDAPCTX
MQNLDAPCTX 構造体は、以下の C プロトタイプを持ちます。

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4    StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  LDAP *    objectDirectory;    /* LDAP Instance */
  MQLONG       ldapVersion;        /* Which LDAP version to use? */
  MQLONG       port;            /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL    ssl;            /* SSL enabled? */
  MQCHAR *    host;            /* Hostname of LDAP server */
  MQCHAR *    password;        /* Password of LDAP server */
  MQCHAR *    searchFilter;        /* LDAP search filter */
  MQCHAR *    baseDN;            /* Base Distinguished Name */
  MQCHAR *    charSet;        /* Character set */
};
```

接続エンドポイント検索出口の作成のためのサンプル・コード
Windows および分散プラットフォームでソースをコンパイルするためのコード・スニペット。

ソースのコンパイル

任意の LDAP クライアント・ライブラリー (例えば、IBM Tivoli Directory Server 6.3 クライアント・ライブラリー) を使用して、ソースをコンパイルできます。 この資料は、Tivoli Directory Server 6.3 クライアント・ライブラリーを使用することを前提としています。

注 : 接続前出口ライブラリーは、以下の LDAP サーバーを使用してテストされています。

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

以下のコード・スニペットでは、Windows および他の分散プラットフォームで出口をコンパイルする方法について説明します。

Windows プラットフォームでの出口のコンパイル

Windows で出口ソースをコンパイルする場合は、以下のスニペットを使用できます。

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib  \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 /
DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
              $(CC) $(CCARGS) $*.c
```

注 : IBM Tivoli Directory Server 6.3 クライアント・ライブラリーを Microsoft Visual Studio 2003 コンパイラーでコンパイルした場合、 Microsoft Visual Studio 2005 以上のコンパイラーを使用して IBM Tivoli Directory Server 6.3 クライアント・ライブラリーをコンパイルすると、警告が表示されることがあります。

他の分散プラットフォームでの出口のコンパイル

他の分散プラットフォーム (例えば、Linux) で出口ソースをコンパイルする場合は、以下のスニペットを使用できます。 他の分散プラットフォームでは、一部のコンパイラーのオプションが異なる場合があります。

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server には、静的リンク・ライブラリーと動的リンク・ライブラリーの両方が付属していますが、使用できるライブラリーの形式は 1 つのみです。 このスクリプトは、静的ライブラリーを使用することを前提としています。

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl

amqlcepl: amqlcel0.c
        $(CC) -o cepl amqlcel0.c  -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

出口モジュールの呼び出し

接続前出口モジュールは、3 種類の理由コードを使用して呼び出すことができます。 このセクションでは、それぞれの出口理由コードの詳細を説明します。

MQXR_INIT

LDAP サーバーとの接続を開始および確立する場合は、MQXR_INIT 理由コードを使用して出口を呼び出します。

MQXR_INIT 呼び出しの前、MQNXP 構造体の *pExitDataPtr* フィールドには、*mqclient.ini* ファイル内の PreConnect スタンザからの Data 属性 (例えば、LDAP) が入力されています。

LDAP URL は、少なくともプロトコル、ホスト名、ポート番号、および検索のベース DN で構成されています。 出口は *pExitDataPtr* フィールドに含まれる LDAP URL を解析し、MQNLDAPCTX LDAP Lookup Context 構造体を割り振ってそれを適宜に入力します。 この構造体のアドレスは、*pExitUserAreaPtr* フ

ィールドに保管されます。 LDAP URL を正しく解析できなかった場合は、MQCC_FAILED エラーが発生します。

この時点では、出口は MQNLDAPCTX パラメーターを使用して LDAP サーバーに接続およびバインドします。 その結果として発生する LDAP API ハンドルもこの構造体に保管されます。

MQXR_PRECONNECT
LDAP サーバーからチャネル定義を取得する場合は、MQXR_PRECONNECT 理由コードを使用して出口モジュールを呼び出します。

出口は、LDAP サーバーから所定のフィルターと一致するチャネル定義を検索します。 *QMgrName* パラメーターに特定のキュー・マネージャー名が含まれている場合、*ibm-amqQueueManagerName* LDAP 属性値が所定のキュー・マネージャー名と一致するすべてのチャネル定義が検索結果として返されます。

QMgrName パラメーターが「*」または「」(ブランク) の場合、*ibm-amqIsClientDefault* 接続エンドポイント属性が true に設定されているすべてのチャネル定義が検索結果として返されます。

検索が成功すると、出口は MQCD 定義を 1 つまたは 1 配列作成し、呼び出し元に返します。

MQXR_TERM
出口をクリーンアップする場合は、この理由コードを使用して出口を呼び出します。 この間、出口は LDAP サーバーとの接続を切断し、出口が割り振りおよび保守するすべてのメモリーを解放します。 これには、*MQNLDAPCTX* 構造体、ポインターの配列、およびそれが参照する MQCD が含まれます。 その他のフィールドはすべてデフォルト値に設定されます。 *pQMgrName* および *ppConnectOpts* 出口パラメーターは、*MQXR_TERM* の間は使用されず、場合によっては NULL になります。

LDAP スキーマ
クライアント接続データは、LDAP (Lightweight Directory Access Protocol) ディレクトリーというグローバル・リポジトリーに保管されます。 WebSphere MQ クライアントは、LDAP ディレクトリーを使用して接続定義を取得します。 LDAP ディレクトリー内の WebSphere MQ クライアント接続定義の構造体は LDAP スキーマといいます。 LDAP スキーマは、属性タイプの定義、オブジェクト・クラスの定義、およびサーバーの判断基準 (フィルターまたは属性値のアサーションがエントリーの属性と一致するかどうか、操作を許可、追加、および変更するかどうか) となるその他の情報の集合体です。

LDAP ディレクトリーへのデータの保管

クライアント接続定義は、接続ポイントというディレクトリー・ツリー内の特定のブランチにあります。 接続ポイントには、LDAP ディレクトリー内のその他のノードのように、それに関連付けられた識別名 (DN) が付けられています。 このノードは、ディレクトリーで実行するすべての照会の開始点として使用することができます。 クライアント接続定義のサブセットを返すよう LDAP ディレクトリーを照会する場合は、フィルターを使用します。 サブツリーへのアクセスは、ディレクトリー・ツリーの他の部分 (例えば、ユーザー、部門、またはグループ) で付与された権限に基づいて、制限することができます。

独自の属性およびクラスの定義
LDAP スキーマを変更して、クライアント・チャネル定義を保管します。 すべての LDAP データ定義には、オブジェクトと属性が必要です。 オブジェクトと属性は、オブジェクトまたは属性を一意的に識別するオブジェクト ID (OID) 番号によって識別されます。 LDAP スキーマ内のすべてのクラスは、最上位のオブジェクトから直接的または間接的に継承されます。 クライアント・チャネル定義オブジェクトには、最上位のオブジェクトの属性が含まれています。 すべての LDAP データ定義には、以下のオブジェクトと属性が必要です。

- オブジェクト定義は、LDAP 属性の集合体です。
- 属性は LDAP データ型です。

各属性の説明、および属性が通常の WebSphere MQ プロパティーにどのようにマップされるかについては、LDAP 属性を参照してください。

LDAP 属性
定義された LDAP 属性は WebSphere MQ 固有の属性であり、クライアント接続プロパティーに直接マップされます。

WebSphere MQ クライアント・チャネル・ディレクトリーのストリング属性

以下の表では、文字ストリング属性とそれらがマップされる WebSphere MQ プロパティーをリストします。 これらの属性は、directoryString (サブセットとして IA5/ASCII を含む可変バイトのエンコード・システムである、UTF-8 エンコードの Unicode) 構文の値を保持することができます。 構文は、そのオブジェクト ID (OID) 番号によって指定されます。

表 *21. WebSphere MQ* クライアント・チャネル・ディレクトリーのストリング属性

LDAP 属性	**説明**	**WebSphere MQ プロパティー**
CN	チャネル名と定義されるキュー・マネージャー名で構成される共通名。	
ibm-amqChannelName	チャネル定義の名前。	CHANNEL
ibm-amqConnectionName	通信接続 ID。	CONNAME
ibm-amqDescription	チャネルの説明。	DESCR
ibm-amqLocalAddress	チャネルのローカル通信アドレス。	LOCLADDR
ibm-amqModeName	LU 6.2 のモード名。	MODENAME
ibm-amqPassword	使用可能なパスワード。	パスワード
ibm-amqQueueManagerName	WebSphere MQ クライアント・アプリケーションが接続を要求できるキュー・マネージャーまたはキュー・マネージャー・グループの名前。	QMNAME
ibm-amqSecurityExitUserData	セキュリティー出口に渡されるユーザー・データ。	SCYDATA
ibm-amqSecurityExitName	チャネル・セキュリティー出口によって実行される出口プログラムの名前。	SCYEXIT
ibm-amqSslCipherSpec	SSL 接続用の単一の CipherSpec。	SSLCIPH
ibm-amqSslPeerName	WebSphere MQ チャネルの相手側にあるピア・キュー・マネージャーまたはクライアントからの証明書の識別名 (DN) を確認します。	SSLPEER
ibm-amqTransactionProgramName	トランザクション・プログラム名。	TPNAME
ibm-amqUserID	リモート MCA との保護 SNA セッションの開始を試行するときに MCA が使用するユーザー ID。	ユーザー ID

WebSphere MQ クライアント接続の整数属性

事前定義値を持つ属性 (例えば、列挙型) は、標準の整数として保管されます。 これらの値は、関連付けられた定数名ではなく、整数値として LDAP ディレクトリーに保管されます。

表 *22. WebSphere MQ* クライアント・チャネル・ディレクトリーの整数属性

LDAP 属性	**説明**	**WebSphere MQ プロパティー**
ibm-amqConnectionAffinity	同じキュー・マネージャー名を使用して複数回接続するクライアント・アプリケーションが、同じクライアント・チャネルを使用するかどうかを決定します。	AFFINITY
ibm-amqClientChannelWeight	どのクライアント接続チャネル定義を使用するかに影響を与える加重。	CLNTWGHT

表 22. *WebSphere MQ* クライアント・チャネル・ディレクトリーの整数属性 (続き)		
LDAP 属性	**説明**	**WebSphere MQ プロパティー**
ibm-amqHeartBeatInterval	伝送キューにメッセージがないときに送信側の MCA から渡されるハートビート・フローの間隔の概算時間。	HBINT
ibm-amqKeepAliveInterval	チャネルのタイムアウト値。	KAINT
ibm-amqMaximumMessageLength	チャネル上で送信可能なメッセージの最大長。	MAXMSGL
ibm-amqSharingConversations	各 TCP/IP チャネル・インスタンスを共用する会話の最大数。	SHARECNV
ibm-amqTransportType	使用するトランスポート・タイプ。	TRPTYPE

WebSphere MQ クライアント・チャネルのブール属性

このブール属性は、どの WebSphere MQ プロパティーにもマップされません。 この属性の構文はブール値を示します。

表 23. *WebSphere MQ* クライアント・チャネルのブール属性	
LDAP 属性	**説明**
ibm-amqIsClientDefault	このブール属性は、ibm-amqQueueManagerName 属性が定義されていないエントリーの検索の問題を解決するために定義します。

WebSphere MQ クライアント・チャネルのリスト属性

WebSphere MQ プロパティーは、単一値のコンマ区切りリスト属性として LDAP ディレクトリー内に保管されます。 これらの属性は、他のディレクトリー・ストリング属性と同じ方法で定義されます。 以下の表では、リスト属性とそれらがマップされる WebSphere MQ プロパティーについて説明します。

表 24. *WebSphere MQ* クライアント・チャネルのリスト属性		
LDAP 属性	**説明**	**WebSphere MQ プロパティー**
ibm-amqHeaderCompression	チャネルがサポートするヘッダー・データ圧縮技法のリスト。	COMPHDR
ibm-amqMessageCompression	チャネルがサポートするメッセージ・データ圧縮技法のリスト。	COMPMSG
ibm-amqSendExitUserData	送信出口に渡されるユーザー・データ。	SENDDATA
ibm-amqSendExitUserName	チャネル送信出口によって実行される出口プログラムの名前。	SENDEXIT
ibm-amqReceiveExitUserData	受信出口に渡されるユーザー・データ。	RCVDATA
ibm-amqReceiveExitName	チャネル受信ユーザー出口によって実行されるユーザー出口プログラムの名前。	RCVEXIT

共通名

共通名 (CN) は、チャネル名と定義されるキュー・マネージャー名で構成されます。

これは既存の属性です。

CN の形式は次のとおりです。

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

以下に例を示します。

```
CN=TC1(QM_T1)
```

この属性の値は 1 つしか指定できません。

これはストリング属性で、値は大/小文字を区別しません。 サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルター (サブストリング (例えば、CN=jim* (CN は属性)) を使用し、1 つ以上のワイルドカードを含む) での属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqChannelName
この属性は、チャネル定義の名前を指定します。

この属性は、大/小文字を区別しない、最大 20 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルター (サブストリングを使用し、1 つ以上のワイルドカードを含む) での属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqDescription
この LDAP 属性は、チャネルの説明を提供します。

この属性は、大/小文字を区別しない、最大 64 バイトの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqConnectionName
この LDAP 属性は、通信接続 ID です。 チャネルが使用する特定の通信リンクを指定します。

この属性は、大/小文字を区別しない、最大 264 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqLocalAddress
この属性は、チャネルのローカル通信アドレスを指定します。

この属性は、大/小文字を区別しない、最大 48 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqModeName
この属性は、LU 6.2 接続に使用します。 これは、通信セッションの割り振りが実行されるときに、接続のセッションの特性について追加の定義を提供します。

この属性は、大/小文字を区別しない、8 文字ちょうどの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqPassword
この LDAP 属性は、リモート MCA との保護 LU 6.2 セッションの開始を試行するときに MCA が使用可能なパスワードを指定します。

この属性は、最大 12 桁の単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqQueueManagerName
この属性は、WebSphere MQ クライアント・アプリケーションが接続を要求できるキュー・マネージャーまたはキュー・マネージャー・グループの名前を指定します。

この属性は、大/小文字を区別しない、最大 48 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqSecurityExitUserData
この LDAP 属性は、セキュリティー出口に渡されるユーザー・データを指定します。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqSecurityExitName
この LDAP 属性は、チャネル・セキュリティー出口によって実行される出口プログラムの名前を指定します。

有効なチャネル・セキュリティー出口がない場合は、これをブランクのままにします。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqSslCipherSpec
この LDAP 属性は、SSL 接続用の単一の CipherSpec を指定します。

この属性は、大/小文字を区別しない、最大 32 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqSslPeerName
この LDAP 属性は、WebSphere MQ チャネルの相手側にあるピア・キュー・マネージャーまたはクライアントからの証明書の識別名 (DN) を確認するために使用します。

この LDAP 属性は、大/小文字を区別しない、最大 1024 バイトの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqTransactionProgramName
この LDAP 属性は、トランザクション・プログラム名を指定します。 これは、LU 6.2 接続に使用されます。

この属性は、大/小文字を区別しない、最大 64 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqUserID
この LDAP 属性は、リモート MCA との保護 SNA セッションの開始を試行するときに MCA が使用するユーザー ID を指定します。

この属性は、大/小文字を区別しない、12 文字ちょうどの単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

ibm-amqConnectionAffinity
この LDAP 属性は、同じキュー・マネージャー名を使用して複数回接続するクライアント・アプリケーションが、同じクライアント・チャネルを使用するかどうかを指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqClientChannelWeight
この LDAP 属性は、どのクライアント接続チャネル定義を使用するかに影響を与える加重を指定します。

クライアント・チャネル加重属性は、複数の適切な定義が使用可能な場合に、クライアント・チャネル定義の選択にバイアスをかけるために使用します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqHeartBeatInterval
この LDAP 属性によって、伝送キューにメッセージがなくなったときに送信 MCA からハートビート・フローが渡される間の時間の近似値を指定することができます。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。 デフォルト値は 1 です。 デフォルトは、現在の MQSERVER 環境変数操作で設定されます。

ibm-amqKeepAliveInterval
この LDAP 属性は、チャネルのタイムアウト値を指定するために使用します。

この属性の値は、チャネルのキープアライブ・タイミングを指定する通信スタックに渡されます。 これは、チャネルごとに別々のキープアライブ値を指定するために使用できます。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqMaximumMessageLength
この LDAP 属性は、チャネル上で送信可能なメッセージの最大長を指定します。

現在の MQSERVER 環境変数の操作により、この属性のデフォルト値は 104857600 です。 この属性は単一の整数値を持ち、既存の属性ではありません。

ibm-amqSharingConversations
この LDAP 属性は、各 TCP/IP チャネル・インスタンスを共用する会話の最大数を指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqTransportType
この LDAP 属性は、使用するトランスポート・タイプを指定します。

この属性は、単一の整数値を持ちます。 これは、既存の属性ではありません。

ibm-amqIsClientDefault
このブール属性は、ibm-amqQueueManagerName 属性が定義されていないエントリーの検索の問題を解決します。

通常、接続前出口モジュールは検索条件に ibm-amqQueueManagerName 属性の値を指定して LDAP サーバーを検索します。 このようにして照会すると、ibm-amqQueueManagerName 属性値が MQCONN/X 呼び出しに指定されたキュー・マネージャーの名前と一致するすべての項目が返されます。 ただし、クライアント・チャネル定義テーブル (CCDT) を使用する場合は、MQCONN/X 呼び出しでキュー・マネージャー名をブランクに設定したり、名前の前にアスタリスク (*) を付けたりすることができます。キュー・マネージャーの名前がブランクの場合、クライアントはデフォルトのキュー・マネージャーに接続します。 キュー・マネージャーの名前の先頭にアスタリスク (*) が付いているときは、クライアントは任意のキュー・マネージャーに接続します。

同様に、項目内の ibm-amqQueueManagerName 属性を未定義のままにしておくことができます。 この場合、このエンドポイント情報を使用するクライアントがどのキュー・マネージャーにも接続できることが予期されています。 例えば、ある項目に次の行が含まれているとします。

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

この例では、クライアントは myhost 上で実行中の指定されたキュー・マネージャーに接続することを試みます。

しかし、LDAP サーバーでは、定義されていない属性値の検索は実行されません。 例えば、ある項目に ibm-amqQueueManagerName 以外の接続情報が含まれている場合、検索結果にこの項目は含まれません。 この問題を克服するために、ibm-amqIsClientDefault を設定することができます。 これはブール属性で、定義されていない場合は FALSE の値をとるものと見なされます。

ibm-amqQueueManagerName が定義されていなくても検索結果に含める項目については、ibm-amqIsClientDefault を TRUE に設定します。 MQCONN/X への呼び出しでキュー・マネージャー名としてブランクまたはアスタリスク (*) が指定されている場合、接続前出口は LDAP サーバーを検索して、ibm-amqIsClientDefault 属性値が TRUE に設定されているすべての項目を探します。

注 : ibm-amqIsClientDefault が TRUE に設定されている場合は、ibm-amqQueueManagerName 属性を設定または定義しないでください。

ibm-amqHeaderCompression
この LDAP 属性は、チャネルでサポートされるヘッダー・データ圧縮技法のリストです。

この属性の最大サイズは 48 文字です。 これは、既存の属性ではありません。

この属性の値は 1 つしか指定できません。

このリスト属性は、コンマ区切り形式のディレクトリー・ストリングとして指定します。 例えば、**ibm-amqHeaderCompression** に指定される値は 0 で、これは NONE にマップされます。 最大許容制限を超える値は、クライアントに無視されます。 例えば、ibm-amqHeaderCompression はリストに最大 2 つの整数を含みます。

ibm-amqMessageCompression
この LDAP 属性は、チャネルでサポートされるメッセージ・データ圧縮技法のリストです。

この属性の最大サイズは 48 文字です。 これは、既存の属性ではありません。

この属性は複数の値をサポートしません。

このリスト属性は、コンマ区切り形式のディレクトリー・ストリングとして指定します。 例えば、この属性に指定される値は 1、2、4 で、これは基礎となる圧縮シーケンス RLE、ZLIBFAST、および ZLIBHIGH にマップされます。

最大許容制限を超える値は、クライアントに無視されます。 例えば、ibm-amqMessageCompression はリストに最大 16 つの整数を含みます。

ibm-amqSendExitUserData
この LDAP 属性は、送信出口に渡されるユーザー・データを指定します。

この LDAP 属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqSendExitName と **ibm-amqSendExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ibm-amqSendExitName
この LDAP 属性は、チャネル送信出口によって実行される出口プログラムの名前を指定します。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqSendExitName と **ibm-amqSendExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ibm-amqReceiveExitUserData
この LDAP 属性は、受信出口に渡されるユーザー・データを指定します。

一連の受信出口を実行できます。 一連の出口のユーザー・データのストリングは、コンマ、スペース、またはその両方で区切られます。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqReceiveExitName と **ibm-amqReceiveExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

ibm-amqReceiveExitName
この LDAP 属性は、チャネル受信ユーザー出口によって実行されるユーザー出口プログラムの名前を指定します。

この属性は、連続して実行されるプログラムの名前のリストです。 有効なチャネル受信ユーザー出口がない場合には、ブランクのままにしておいてください。

この属性は、大/小文字を区別しない、最大 999 文字の単一のストリング値を持ちます。 これは、既存の属性ではありません。

サブストリング・マッチングは無視されます。 サブストリング・マッチングは、検索フィルターでの属性の動作を指定する、サブスキーマで使用されるマッチング規則です。

注 : ibm-amqReceiveExitName と **ibm-amqReceiveExitUserData** は、ペアで同期させる必要があります。 また、ユーザー・データは、出口名と同期させる必要があります。 そのため、一方を指定した場合、もう一方も (それにデータが含まれていない場合でも) 対称的に指定する必要があります。

キューイング・アプリケーションの作成

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

以下のリンクを使用して、アプリケーションの作成についての詳細を確認してください。

関連概念

8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

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

96 ページの『WebSphere MQ プログラムのサンプル』
この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

353 ページの『クライアント・アプリケーションの作成』
WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

950 ページの『WebSphere MQ での Web サービスの使用』
IBM WebSphere MQ transport for SOAP または IBM WebSphere MQ bridge for HTTP を使用して、Web サービス用の IBM WebSphere MQ アプリケーションを開発できます。

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

553 ページの『プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

Message Queue Interface の概要

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

メッセージ・キュー・インターフェースは以下のもので成り立っています。

- 呼び出し (プログラムはこれを使用してキュー・マネージャーとその機能にアクセスすることができる)
- 構造体 (プログラムはこれを使用してキュー・マネージャーにデータを渡したり、データを読み取ったりする)
- 基本データ・タイプ (キュー・マネージャーにデータを渡したり、データを読み取ったりする)

WebSphere MQ for Windows および WebSphere MQ (UNIX and Linux システム用) は以下のものも提供します。

- WebSphere MQ for Windows および WebSphere MQ (UNIX and Linux システム用) のプログラムが変更をコミットおよびバックアウトできるような呼び出し。
- これらのプラットフォームで提供される定数の値を定義する組み込みファイル。
- アプリケーションにリンクするライブラリー・ファイル。
- これらのプラットフォーム上での MQI の使用方法を示すための一連のサンプル・プログラム。 このサンプルの詳細については、96 ページの『分散プラットフォームにおけるサンプル・プログラム』を参照してください。
- 外部トランザクション管理プログラムにバインドするためのサンプル・ソースおよび実行可能コード。

MQI の詳細については、以下のリンクを使用してください。

- 195 ページの『MQI 呼び出し』
- 196 ページの『同期点の呼び出し』
- 196 ページの『データ変換、データ・タイプ、データ定義、および構造体』
- 197 ページの『IBM WebSphere MQ スタブ・プログラムおよびライブラリー・ファイル』
- 201 ページの『すべての呼び出しに共通のパラメーター』
- 202 ページの『バッファーの指定』
- 203 ページの『UNIX and Linux 信号処理』

関連概念

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

214 ページの『オブジェクトのオープンとクローズ』

ここでは、WebSphere MQ オブジェクトのオープンとクローズについて説明します。

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

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

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

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

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

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

MQI 呼び出し

この情報を使用して、MQI での呼び出しについて学びます。

MQI の呼び出しは、次のように分類できます。

MQCONN、MQCONNX、および MQDISC
プログラムをキュー・マネージャーに接続したり (オプションを指定してまたは指定せずに)、キュー・マネージャーから切断したりするために使用します。 z/OS 用の CICS プログラムを作成する場合は、これらの呼び出しを使用する必要はありません。 ただし、作成するアプリケーションを他のプラットフォームに移植したい場合には、これらの呼び出しを使用することをお勧めします。

MQOPEN および MQCLOSE
キューなどのオブジェクトをオープンしたり、クローズするために使用します。

MQPUT および MQPUT1
メッセージをキューに書き込むために使用します。

MQGET
キュー上のメッセージをブラウズしたり、キューからメッセージを除去するために使用します。

MQSUB、MQSUBRQ
トピックにサブスクリプションを登録するためと、サブスクリプションに一致するパブリケーションを要求 するために使用します。

MQINQ
オブジェクトの属性を照会するために使用します。

MQSET
キューのいくつかの属性を設定するために使用します。 別のタイプのオブジェクトの属性を設定することはできません。

MQBEGIN、MQCMIT、および MQBACK
WebSphere MQ が作業単位の調整役であるときに、これらの呼び出しを使用します。 MQBEGIN は作業単位を開始します。 MQCMIT と MQBACK は作業単位を終了し、それぞれ、作業単位中に作成された更新をコミットするか、またはロールバックします。 ネイティブのコミットメント制御開始、コミット、およびロールバック・コマンドが使用されます。

MQCRTMH、MQBUFMH、MQMHBUF、MQDLTMH
メッセージ・ハンドルを作成するため、メッセージ・ハンドルをバッファーに、またはバッファーをメッセージ・ハンドルに 変換するため、およびメッセージ・ハンドルを削除するために使用します。

MQSETMP、MQINQMP、MQDLTMP
メッセージ・ハンドルにメッセージ・プロパティーを設定するため、メッセージ・プロパティーを 照会するため、およびメッセージ・ハンドルからプロパティーを削除するために使用します。

MQCB、MQCB_FUNCTION、MQCTL
コールバック機能の登録および制御のために使用します。

MQSTAT
前の非同期書き込み操作に関する状況情報を取得するために 使用します。

MQI 呼び出しの説明については、呼び出しの記述を参照してください。

同期点の呼び出し

各種のプラットフォームにおける同期点の呼び出しについて理解するために、この情報を役立ててください。

次のようにして、同期点呼び出しを利用することができます。

Windows、UNIX、および Linux プラットフォームでの IBM WebSphere MQ 呼び出し

次の製品では、MQCMIT 呼び出しと MQBACK 呼び出しが提供されます。

- IBM WebSphere MQ for Windows
- UNIX and Linux システム上の IBM WebSphere MQ

同期点呼び出しをプログラムで使用して、最後の同期点以降のすべての MQGET および MQPUT 操作を永続的に行う (コミットする) か、バックアウトすることをキュー・マネージャーに通知します。 CICS 環境における変更をコミット、およびバックアウトするには、EXEC CICS SYNCPOINT および EXEC CICS SYNCPOINT ROLLBACK などのコマンドを使用してください。

データ変換、データ・タイプ、データ定義、および構造体

Message Queue Interface を使用する際の、データ変換、基本データ・タイプ、WebSphere MQ データ定義、および構造体について理解するために、この情報を使用します。

データ変換

MQXCNVC (文字変換) 呼び出しは、メッセージ文字データをある文字セットから別の文字セットに変換します。 WebSphere MQ for z/OS を除き、この呼び出しは、データ変換出口からのみ使用されます。

MQXCNVC 呼び出しで使用される構文については、MQXCNVC - 文字の変換を参照してください。また、データ変換出口の作成方法と呼び出し方法については、416 ページの『データ変換出口の作成』を参照してください。

基本データ・タイプ

サポートされるプログラム言語の場合、MQI は基本データ・タイプまたは構造化されていないフィールドを提供します。

これらのデータ・タイプについては、基本データ・タイプで詳しく説明されています。

WebSphere MQ データ定義

WebSphere MQ によって提供されるデータ定義ファイルには、以下のものがあります。

- すべての WebSphere MQ 定数および戻りコードの定義
- WebSphere MQ 構造体およびデータ・タイプの定義
- 構造体の初期化用の定数
- 各呼び出しの関数原型 (PL/I および C 言語のみ)

WebSphere MQ データ定義ファイルの詳細については、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

構造体

195 ページの『MQI 呼び出し』のリストにある MQI 呼び出しで使用される構造体は、サポートされている各プログラム言語用のデータ定義ファイルに含まれています。

構造体の要約については、構造体データ・タイプの要約を参照してください。

IBM WebSphere MQ スタブ・プログラムおよびライブラリー・ファイル

提供されるスタブ・プログラムおよびライブラリー・ファイルは、プラットフォームごとにここにリストされます。

実行可能アプリケーションを作成するときのスタブ・プログラムとライブラリー・ファイルの使用方法について詳しくは、429 ページの『IBM WebSphere MQ アプリケーションの構築』を参照してください。 C++ ライブラリー・ファイルへのリンクについては、「C++ の使用 *WebSphere MQ C++* の使用」を参照してください。

IBM WebSphere MQ for Windows

IBM WebSphere MQ for Windows では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

表 *25. Windows* アプリケーション用のライブラリー・ファイル

ライブラリー・ファイル	**環境**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	C 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	C 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	C 用サーバー XA インターフェース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	C 用クライアント XA インターフェース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	C 用クライアント MTS (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib	C 用サーバー TXSeries CICS サポート (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib	C 用クライアント TXSeries CICS サポート (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	C 用インストール可能サービス出口 (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	IBM COBOL 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Micro Focus COBOL 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	IBM COBOL 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Micro Focus COBOL 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	C++ 用サーバー (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	C++ 用クライアント (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	C++ 用ベース (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	C++ 用クライアント MTS (32 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	C 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	C 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	C 用サーバー XA インターフェース (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	C 用クライアント XA インターフェース (64 ビット)

表 25. *Windows* アプリケーション用のライブラリー・ファイル (続き)	
ライブラリー・ファイル	**環境**
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	C 用クライアント MTS (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	IBM COBOL 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Micro Focus COBOL 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	IBM COBOL 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Micro Focus COBOL 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	C++ 用サーバー (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	C++ 用クライアント (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	C++ 用ベース (64 ビット)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	C++ 用クライアント MTS (64 ビット)

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

.NET プログラムをコンパイルするには、`amqmdnet.dll` を使用します。 詳しくは、セクション 564 ページの『.NET の使用』内の 599 ページの『WebSphere MQ .NET プログラムのコンパイル』を参照してください。

次の各ファイルは、以前のリリースとの互換性を保つために出荷されます。

mqic32.lib
mqic32xa.lib

IBM WebSphere MQ for AIX

IBM WebSphere MQ for AIX では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

スレッド化されていないアプリケーションでは、次のようになります。

表 26. スレッド化されていない *AIX* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm.a	C 用サーバー
libmqic.a & **libmqm.a**	C 用クライアント
libmqmzf.a	C でインストール可能なサービス出口
libmqmxa.a	サーバー XA インターフェース
libmqmxa64.a	サーバー代替 XA インターフェース
libmqcxa.a	クライアント XA インターフェース
libmqcxa64.a	クライアント代替 XA インターフェース

表 *26.* スレッド化されていない *AIX* アプリケーション用のライブラリー・ファイル *(*続き*)*	
ライブラリー・ファイル	**環境**
libmqmcbrt.o	Micro Focus COBOL サポート用 WebSphere MQ 実行時ライブラリー
libmqmcb.a	COBOL 用サーバー
libmqicb.a	COBOL 用クライアント
libimqc23ia.a	C++ 用クライアント
libimqs23ia.a	C++ 用サーバー

スレッド化されたアプリケーションでは、次のようになります。

表 *27.* スレッド化された *AIX* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm_r.a	C 用サーバー
libmqic_r.a & **libmqm_r.a**	C 用クライアント
libmqmzf_r.a	C でインストール可能なサービス出口
libmqmxa_r.a	サーバー XA インターフェース
libmqmxa64_r.a	サーバー代替 XA インターフェース
libmqcxa_r.a	クライアント XA インターフェース
libmqcxa64_r.a	クライアント代替 XA インターフェース
libimqc23ia_r.a	C++ 用クライアント
libimqs23ia_r.a	C++ 用サーバー

IBM WebSphere MQ HP-UX

IBM WebSphere MQ for HP-UX では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

IA64 (IPF) プラットフォーム

スレッド化されていないアプリケーションでは、次のようになります。

表 *28.* スレッド化されていない *HP-UX* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm.so	C 用サーバー
libmqic.so & **libmqm.so**	C 用クライアント
libmqmzf.so	C でインストール可能なサービス出口
libmqmxa.so	サーバー XA インターフェース
libmqmxa64.so	サーバー代替 XA インターフェース
libmqcxa.so	クライアント XA インターフェース
libmqcxa64.so	クライアント代替 XA インターフェース
libimqi23ah.so	C++

表 *28.* スレッド化されていない *HP-UX* アプリケーション用のライブラリー・ファイル *(*続き*)*	
ライブラリー・ファイル	**環境**
libmqmcbrt.o	Micro Focus COBOL サポート用 WebSphere MQ 実行時ライブラリー
libmqmcb.so	COBOL 用サーバー
libmqicb.so	COBOL 用クライアント

スレッド化されたアプリケーションでは、次のようになります。

表 *29.* スレッド化された *HP-UX* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm_r.so	C 用サーバー
libmqmzf_r.so & **libmqm_r.so**	C でインストール可能なサービス出口
libmqmxa_r.so	サーバー XA インターフェース
libmqmxa64_r.so	サーバー代替 XA インターフェース
libmqcxa_r.so	クライアント XA インターフェース
libmqcxa64_r.so	クライアント代替 XA インターフェース
libimqi23ah_r.so	C++

IBM WebSphere MQ の Linux

IBM WebSphere MQ for Linux では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行する環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

スレッド化されていないアプリケーションでは、次のようになります。

表 *30.* スレッド化されていない *Linux* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm.so	C 用サーバー
libmqic.so & **libmqm.so**	C 用クライアント
libmqmzf.so	C でインストール可能なサービス出口
libmqmxa.so	サーバー XA インターフェース
libmqmxa64.so	サーバー代替 XA インターフェース
libmqcxa.so	クライアント XA インターフェース
libmqcxa64.so	クライアント代替 XA インターフェース
libimqc23gl.so	C++ 用クライアント
libimqs23gl.so	C++ 用サーバー

スレッド化されたアプリケーションでは、次のようになります。

表 *31.* スレッド化された *Linux* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm_r.so	C 用サーバー

表 31. スレッド化された *Linux* アプリケーション用のライブラリー・ファイル (続き)	
ライブラリー・ファイル	**環境**
libmqic_r.so & **libmqm_r.so**	C 用クライアント
libmqmzf_r.so	C でインストール可能なサービス出口
libmqmxa_r.so	サーバー XA インターフェース
libmqmxa64_r.so	サーバー代替 XA インターフェース
libmqcxa_r.so	クライアント XA インターフェース
libmqcxa64_r.so	クライアント代替 XA インターフェース
libimqc23gl_r.so	C++ 用クライアント
libimqs23gl_r.so	C++ 用サーバー

IBM WebSphere MQ For Solaris

IBM WebSphere MQ for Solaris では、オペレーティング・システムによって提供されるライブラリー・ファイルに加え、アプリケーションを実行している環境用に提供される MQI ライブラリー・ファイルにプログラムをリンクする必要があります。

表 32. *Solaris* アプリケーション用のライブラリー・ファイル	
ライブラリー・ファイル	**環境**
libmqm.so	C 用サーバーおよびクライアント
libmqmzse.so	C 用
libmqic.so	C 用クライアント
libmqmcs.so	C 用共通サービス
libmqmzf.so	C でインストール可能なサービス出口
libmqmxa.so	サーバー XA インターフェース
libmqmxa64.so	サーバー代替 XA インターフェース
libmqcxa.so	クライアント XA インターフェース
libmqcxa64.so	クライアント代替 XA インターフェース
libimqc23as.a	C++ 用クライアント
libimqs23as.a	C++ 用サーバー

すべての呼び出しに共通のパラメーター

すべての呼び出しに共通のパラメーターには、ハンドルと戻りコードの 2 種類があります。

ハンドルの使用

すべての MQI 呼び出しは、1 つ以上のハンドル を使用します。 ハンドルは、 キュー・マネージャー、キューまたは他のオブジェクト、メッセージ、またはサブスクリプションを、呼び出しに応じて適切に識別します。

キュー・マネージャーと通信するプログラムには、そのキュー・マネージャーを識別するための固有 ID がなければなりません。 この ID は接続ハンドル と呼ばれ、*Hconn* と表される場合もあります。 CICS プログラムの場合は、接続ハンドルが常にゼロです。 他のすべてのプラットフォームやプログラムのスタイルでは、プログラムがキュー・マネージャーに接続するときに接続ハンドルが MQCONN 呼び出しまたは

MQCONNX 呼び出しによって戻されます。 プログラムは、他の呼び出しを使用するときに、接続ハンドルを入力パラメーターとして渡します。

WebSphere MQ オブジェクトで作業するプログラムには、そのオブジェクトを識別するための固有の ID がなければなりません。 この ID はオブジェクト・ハンドル と呼ばれ、*Hobj* と表される場合もあります。 オブジェクト・ハンドルは、プログラムがオブジェクトを使用するためにオープンしたときに、MQOPEN 呼び出しによって戻されます。 プログラムは、後続の MQPUT、MQGET、MQINQ、MQSET、または MQCLOSE 呼び出しを使用するときに、オブジェクト・ハンドルを入力パラメーターとして渡します。

同様に、MQSUB 呼び出しはサブスクリプション・ハンドル (*Hsub* とも呼ばれる) を戻し、 このハンドルは、後続の MQGET、MQCB、または MQSUBRQ 呼び出し でサブスクリプションを識別するために使用されます。また、メッセージ・プロパティーを処理する一定の呼び出しは、メッセージ・ハンドル (*Hmsg* とも呼ばれる) を使用します。

戻りコードの理解

完了コードおよび理由コードは、各呼び出しの出力パラメーターとして戻されます。 これらをまとめて戻りコード と呼びます。

呼び出しが成功したかどうかを示すために、各呼び出しは、完了時に完了コード を戻します。 完了コードは通常、成功を示す MQCC_OK か、失敗を示す MQCC_FAILED です。 ある呼び出しは、中間の状態の MQCC_WARNING (一部成功) を戻すことがあります。

各呼び出しは、呼び出しの失敗や一部成功の理由を示す理由コード も戻します。 キューが満ぱいである、キューに対して読み取り操作が許されていない、ある特定のキューがキュー・マネージャーに対して定義されていない、などの状況を示す多数の理由コードがあります。 プログラムは、理由コードを用いて、どのように進むべきかを判断することができます。 例えば、入力データを変更してから再度呼び出しを実行するようユーザーにプロンプトを出したり、あるいはエラー・メッセージをユーザーに戻したりすることがあります。

完了コードが MQCC_OK のときは、理由コードは常に MQRC_NONE です。

各呼び出しに対する完了コードおよび理由コードが、その呼び出しの説明と共にリストされています。 呼び出しの記述で、リストから該当する呼び出しを選択し、参照してください。

修正処置のアイデアを含む詳細については、以下を参照してください。

- 理由コード (その他すべての WebSphere MQ プラットフォームの場合)

バッファーの指定

キュー・マネージャーは、要求されたときだけバッファーを参照します。 呼び出し時にバッファーを必要としない場合、またはバッファーの長さが 0 の場合、バッファーへの NULL ポインターを使用することができます。

必要なバッファーのサイズを指定するとき、常にデータ長を使用します。

呼び出しからの出力を保持するためにバッファーを使用する (例えば、MQGET 呼び出し用のデータや、MQINQ 呼び出しで照会された属性の値を保持するために使用する) ときに、指定したバッファーが無効であったり読み取り専用ストレージであったりすると、キュー・マネージャーは理由コードを戻そうとします。 しかし、常に理由コードを戻せるとは限りません。

UNIX and Linux の考慮事項

注意の必要な考慮事項。

UNIX and Linux アプリケーションを開発するときには、以下の点にご注意ください。

UNIX and Linux システム内での *fork* システム呼び出し

IBM WebSphere MQ アプリケーションで fork システム呼び出しを使用するときには、以下の考慮事項にご注意ください。

アプリケーションが fork を使用する場合、そのアプリケーションの親プロセスは、 IBM WebSphere MQ 呼び出し (例えば、 MQCONN) を行う前、または **ImqQueueManager** を使用して IBM WebSphere MQ オブジェクトを作成する前に、 fork を呼び出す必要があります。

アプリケーションが何らかの IBM WebSphere MQ 呼び出しを行った後で子プロセスを作成する場合には、アプリケーション・コードは fork() を exec() と共に使用し、子が親の正確なコピーではなく、新しいインスタンスとなるようにしなければなりません。

アプリケーションが exec() を使用しないと、子プロセス内での IBM WebSphere MQ API 呼び出しから MQRC_ENVIRONMENT_ERROR が返されます。

UNIX and Linux 信号処理

この情報は、WebSphere MQ for z/OS または WebSphere MQ for Windows には適用されません。

一般に、UNIX、Linux および IBM i システムは、スレッド化されていない (プロセス) 環境からマルチスレッド環境に移行しています。 スレッド化されていない環境では、たいていのアプリケーションは信号や信号処理を認識する必要はありませんでしたが、信号を使った場合にしか実現できない機能もありました。 マルチスレッド環境では、スレッド・ベースのプリミティブにより、これまでスレッド化されていない環境で信号を使って実現されていた機能がいくつかサポートされています。

しかし、サポートされていても信号や信号処理がマルチスレッド環境に合っていなかったり、いくつもの制限が課せられていることがよく あります。 こういった問題が発生するのは、アプリケーション・コードをマルチスレッド環境の (そのアプリケーションの一部として 稼働している) さまざまなミドルウェア・ライブラリーと統合していて、それらのライブラリーが独立して信号処理を行っている場合です。 プロセスごとに 定義された信号ハンドラーを保存して復元する従来の方法は、1 つのプロセス内に実行スレッドが 1 つしかない場合には有効でしたが、マルチスレッド環境では 機能しません。 その理由は、多数の実行スレッドがそれぞれプロセス全体のリソースの保存と復元を実行するために、予測不能な結果になるからです。

スレッド化されていないアプリケーション

Solaris では、アプリケーションが単一スレッドしか使用していなかったとしてもすべてのアプリケーションがスレッド化されていると見なされるため、Solaris にはこの節の内容は適用されません。

各 MQI 機能は、それぞれ、次の信号に対する独自の信号ハンドラーを設定します。

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

これらの信号を処理するためにユーザーが作成したハンドラーは、MQI 機能呼び出しが実行されている間に置換されます。 これ以外の信号は、ユーザー作成のハンドラーを使って通常の方法で取り込むことができます。 ハンドラーをインストールしていない場合は、デフォルト・アクション (例えば、無視、メモリー・ダンプ、終了) がそのまま実行されます。

WebSphere MQ が同期信号 (SIGSEGV、SIGBUS、SIGFPE、SIGILL) の処理を行うと、MQI 機能呼び出しを行う前に、登録済みの任意の信号ハンドラーに信号を渡そうとします。

スレッド化されたアプリケーション

1 つのスレッドが WebSphere MQ に関連付けられていると見なされる期間は、MQCONN (または MQCONNX) が実行されてから MQDISC が実行されるまでの間です。

同期信号

同期信号は特定のスレッドで発生します。

UNIX and Linux システムでは、安全確保のために、プロセス全体に渡って同期信号を処理する信号ハンドラーを設定することができます。 ただし、スレッドが WebSphere MQ に接続されている間に、WebSphere MQ はアプリケーション・プロセスにおいて以下の信号を処理する独自のハンドラーを設定します。

SIGBUS
SIGFPE
SIGSEGV
SIGILL

マルチスレッド・アプリケーションを作成している場合、各信号を処理する信号ハンドラーはプロセス全体で 1 つしか設定できません。 WebSphere MQ は、独自の同期信号ハンドラーをセットアップすると、各信号に事前登録されたすべてのハンドラーを保存します。 WebSphere MQ がリストされたシグナルの 1 つを処理した後、WebSphere MQ は、プロセス内の最初の WebSphere MQ 接続の時点で有効であったシグナル・ハンドラーを呼び出そうとします。 すべてのアプリケーションのスレッドが WebSphere MQ から切断されると、事前に登録されたハンドラーが復元されます。

信号ハンドラーは WebSphere MQ によって保存および復元されるため、同じプロセス内の別のスレッドも WebSphere MQ に関連付けられている可能性がある場合は、アプリケーション・スレッドでこれらの信号を処理する信号ハンドラーを設定しないでください。

注 : スレッドが WebSphere MQ に接続している間にアプリケーションやミドルウェア・ライブラリー (アプリケーションの一部として稼働する) が信号ハンドラーを確立する場合、そのアプリケーションの信号ハンドラーは、信号の処理中に対応する WebSphere MQ ハンドラーを呼び出す必要があります。

信号ハンドラーの確立や復元では、最後に保管された信号ハンドラーから順番に復元するのが原則です。

- アプリケーションが、WebSphere MQ に接続した後で信号ハンドラーを確立した場合は、以前の信号ハンドラーを復元してから、WebSphere MQ へのアプリケーションの接続を切断してください。
- アプリケーションが、信号ハンドラーを確立してから WebSphere MQ に接続した場合は、WebSphere MQ へのアプリケーションの接続を切断してから、信号ハンドラーを復元してください。

注 : この、最後に保管された信号ハンドラーを最初に復元するという原則に従わなかった場合、アプリケーションでの信号処理に予期せぬ結果を招く可能性があり、アプリケーションによって信号が失われる恐れがあります。

非同期信号

WebSphere MQ では、クライアント・アプリケーションである場合を除いて、スレッド化されたアプリケーションで何らかの非同期信号が使用されることはありません。

スレッド化されたクライアント・アプリケーションに関する追加の考慮事項

WebSphere MQ は、サーバーへの入出力時に、以下の信号の処理を行います。 これらの信号は通信スタックで定義されています。 スレッドがキュー・マネージャーに接続されている間は、アプリケーションでこれらの信号に信号ハンドラーを確立してはなりません。

SIGPIPE (TCP/IP 用)

その他の考慮事項
UNIX 信号処理を使用するときは、以下の考慮事項に注意してください。

ファースト・パス (トラステッド) アプリケーション

ファースト・パス・アプリケーションは、WebSphere MQ が稼働しているプロセスと同じプロセスで稼働しています。つまり、ファースト・パス・アプリケーションは、マルチスレッド環境で稼働しています。

この環境では、WebSphere MQ は、同期信号 SIGSEGV、SIGBUS、SIGFPE、および SIGILL を処理します。他のすべての信号は、ファースト・パス・アプリケーションが WebSphere MQ に接続されている間、ファースト・パス・アプリケーションに送達されることがあってはなりません。 これらの信号は、アプリケーションによってブロックまたは処理される必要があります。 ファースト・パス・アプリケーションがこのようなイベントを代行受信した場合は、必ずキュー・マネージャーを停止してから再起動してください。この処理をしない場合、キュー・マネージャーは未定義状態になる可能性があります。 MQCONNX のもとでファースト・パス・アプリケーションを実行する際の制約事項については、208 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』に詳しく記載されています。

信号ハンドラー内での MQI 機能呼び出し

信号ハンドラー内で MQI 機能を呼び出さないでください。

他の MQI 機能がアクティブになっているときに、信号ハンドラーから MQI 機能を呼び出そうとすると、MQRC_CALL_IN_PROGRESS が戻されます。 他にアクティブになっている MQI 機能がないときに信号ハンドラーから MQI 機能を呼び出そうとすると、選ばれた呼び出しのみをハンドラーから、またはハンドラー内で発行できるというオペレーティング・システムの制約事項のために、操作中のある時点でこの呼び出しが失敗する可能性があります。

C++ デストラクター方式では、プログラムの終了時に自動的に呼び出されることがあるため、MQI 機能の呼び出しを停止できない場合があります。 MQRC_CALL_IN_PROGRESS に関するエラーはすべて無視してください。 信号ハンドラーが exit() を呼び出すと、WebSphere MQ により同期点にあるコミットされていないメッセージは通常どおりバックアウトされ、オープンしているキューはすべてクローズされます。

MQI 呼び出し中の信号

MQI 機能は、コード EINTR およびそれに相当するコードをアプリケーション・プログラムに戻しません。

MQI の呼び出し中に信号が発生し、ハンドラーが *return* を呼び出した場合には、その信号が発生していない場合と同じように、呼び出しが続行されます。 特に、信号によって MQGET に割り込みをかけて、制御を即時にアプリケーションに戻すことはできません。 MQGET から抜け出したい場合は、キューを GET_DISABLED に設定します。あるいは、満了時刻を限定して (MQGMO_WAIT に gmo.WaitInterval を設定する)、MQGET 呼び出しをループに入れます。さらに、スレッド化されていない環境では信号ハンドラーを使用し、スレッド化された環境では信号ハンドラーに対応する機能を使用して、ループを中断するフラグを設定します。

AIX 環境の WebSphere MQ では、信号によって割り込まれたシステム呼び出しは再始動する必要があります。 sigaction(2) を使用して独自の信号ハンドラーを確立するときは、新しいアクション構造の sa_flags フィールドに SA_RESTART フラグを設定してください。これを行わないと、WebSphere MQ が信号の割り込みを受けた任意の呼び出しを完了できなくなる場合があります。

ユーザー出口とインストール可能サービス

マルチスレッド環境で WebSphere MQ プロセスの一部として稼働しているユーザー出口とインストール可能サービスには、ファースト・パス・アプリケーションの場合と同じ制約事項があります。 これらの機能は永続的に WebSphere MQ に関連付けられると考えられるため、これらの機能で信号や非スレッド・セーフのオペレーティング・システム呼び出しを使用しないでください。

VMS 出口ハンドラー

SYS$DCLEXH システム・サービスを使用して、WebSphere MQ アプリケーションの出口ハンドラーをインストールできます。

出口ハンドラーは、イメージが終了するときに制御を受けます。 通常、Exit ($EXIT) サービスまたは Force Exit ($FORCEX) サービスを呼び出したときにイメージが終了します。 $FORCEX はユーザー・モードのターゲット・プロセスを中断します。 続いて、$DCLEXH によって確立されたすべてのユーザー・モード出口ハンドラーは、確立したのと逆順で実行し始めます。 出口ハンドラーおよび $FORCEX についての詳細は、「*VMS* プログラミング概念マニュアル *(VMS Programming Concepts Manual)*」および「*VMS* システム・サービス・マニュアル *(VMS System Services Manual)*」を参照してください。

出口ハンドラー内から MQI 機能を呼び出した場合、イメージが終了した方法によって機能の動作が異なってきます。 他の MQI 機能がアクティブになっているときにイメージが終了すると、MQRC_CALL_IN_PROGRESS が戻されます。

他の MQI 機能がアクティブになっておらず、アップコールが WebSphere MQ アプリケーションで使用不可になっている場合、出口ハンドラー内から MQI 機能を呼び出すことができます。 アップコールが WebSphere MQ アプリケーションで使用可能になっていると、これは失敗し、理由コード MQRC_HCONN_ERROR が出されます。

通常、MQCONN または MQCONNX 呼び出しの有効範囲は、発行元のスレッドです。 アップコールが使用可能の場合、出口ハンドラーは別々のスレッドとして実行し、接続ハンドルは共用できません。

出口ハンドラーは、ターゲット・プロセスの割り込まれたコンテキスト内で開始されます。 ハンドラーが行ったアクションが、その呼び出し元の非同期的に割り込まれたコンテキストに対して、安全で信頼できるかどうかは、アプリケーション次第です。

キュー・マネージャーへの接続とキュー・マネージャーからの切断

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

この接続を行う方法は、プログラムが実行されるプラットフォームと環境に依存します。

z/OS バッチ、WebSphere MQ for IBM i、WebSphere MQ (UNIX システム用)、WebSphere MQ (Linux システム用)、および WebSphere MQ for Windows

これらの環境で実行されるプログラムは、キュー・マネージャーとの接続には MQCONN MQI 呼び出しを、切断には MQDISC 呼び出しをそれぞれ使用しなければなりません。 その代わりの方法として、プログラムは MQCONNX 呼び出しを使用することができます。

z/OS バッチ・プログラムは、同一の TCB 上にある複数のキュー・マネージャーへの連続接続や同時接続を行えます。

IMS

IMS 制御領域が開始するときは、1 つ以上のキュー・マネージャーに接続します。 この接続は、IMS コマンドによって制御されます。 ただし、メッセージ・キューイング IMS プログラムの作成者は、MQCONN MQI 呼び出しを用いて接続を希望する相手のキュー・マネージャーを指定する必要があります。 キュー・マネージャーから切断するには、MQDISC 呼び出しを用いることができます。

同期点を確立する IMS 呼び出し後、 別のユーザーのメッセージを処理する前に、IMS アダプターは、アプリケーションがハンドルをクローズしてキュー・マネージャーから切断したことを確認します。

IMS プログラムは、同一の TCB 上にある複数のキュー・マネージャーへの連続接続や同時接続を行えます。

CICS Transaction Server for z/OS および CICS for MVS/ESA

CICS プログラムは、CICS システム自体が接続しているので、キュー・マネージャーに接続するための作業を行う必要がありません。 通常、この接続は初期化のときに自動的に作成されますが、CKQC トランザクションを使用することもできます。 これは WebSphere MQ for z/OS で提供されています。

CICS タスクは、CICS 領域自体が接続されているキュー・マネージャーにのみ接続できます。

注 : CICS プログラムは、MQI の接続呼び出しと切断呼び出し (MQCONN および MQDISC) を使用することもできます。 これらのアプリケーションを CICS 以外の環境に最小限の記録作業で移植できるようにするために、この方法をとることもできます。 ただし、 CICS 環境では、これらの呼び出しは 常に正常に完了します。 つまり、戻りコードによる情報と、キュー・マネージャーとの実際の接続状態が異なる場合もあります。

TXSeries for Windows とオープン・システム

これらのプログラムは、CICS システム自体が接続しているので、キュー・マネージャーに接続するための作業を行う必要がありません。 したがって、一度に 1 つの接続だけがサポートされます。 CICS アプリケーションでは、接続ハンドルを取得するために MQCONN 呼び出しを発行しなければなりません。また、終了する前には MQDISC 呼び出しを発行しなければなりません。

キュー・マネージャーへの接続とキュー・マネージャーからの切断について詳しくは、以下のリンクを参照してください。

- 207 ページの『MQCONN 呼び出しを使用したキュー・マネージャーへの接続』
- 208 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』
- 213 ページの『MQDISC を使用したキュー・マネージャーからのプログラムの切断』

関連概念

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

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

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

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

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

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

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

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

MQCONN 呼び出しを使用したキュー・マネージャーへの接続

この情報を使用して、MQCONN 呼び出しを使用したキュー・マネージャーへの接続方法について学習します。

一般に、特定のキュー・マネージャー、またはデフォルトのキュー・マネージャーのどちらかへ接続できます。

- バッチ環境の IBM WebSphere MQ for z/OS の場合、デフォルトのキュー・マネージャーは CSQBDEFV モジュールで指定されます。
- IBM WebSphere MQ for Windows、IBM i、UNIX、および Linux システムの場合、デフォルトのキュー・マネージャーは mqs.ini ファイルで指定されます。

別の方法として、z/OS MVS™ バッチ、TSO、および RRS 環境で、キュー共用グループ内の任意のキュー・マネージャーに接続できます。 MQCONN または MQCONNX 要求が、グループのアクティブ・メンバーのいずれか 1 つを選択します。

キュー・マネージャーに接続する場合、キュー・マネージャーはタスクに対してローカルでなければなりません。 キュー・マネージャーは IBM WebSphere MQ アプリケーションと同じシステムに属する必要があります。

IMS 環境では、キュー・マネージャーを IMS 制御領域と、プログラムが使用する従属領域に接続しなければなりません。 デフォルト・キュー・マネージャーは、IBM WebSphere MQ for z/OS のインストール時に CSQQDEFV モジュールに指定されます。

TXSeries CICS 環境と、TXSeries for Windows および AIX では、キュー・マネージャーを CICS に対する XA リソースとして定義する必要があります。

デフォルトのキュー・マネージャーに接続するには、すべてブランク文字からなる、またはヌル文字 (X'00') で始まる名前を指定して、MQCONN を呼び出します。

アプリケーションがキュー・マネージャーに正常に接続するには、アプリケーションが接続の許可を得ている必要があります。 詳しくは、セキュリティーを参照してください。

MQCONN の出力は以下のとおりです。

- 接続ハンドル (**Hconn**)
- 完了コード
- 理由コード

後続の MQI 呼び出しで接続ハンドルを使用してください。

アプリケーションが既にそのキュー・マネージャーに接続されていることを理由コードが示している場合は、アプリケーションが最初に接続されたときに戻されたものと同じ接続ハンドルが戻されます。 この状態ではアプリケーションは MQDISC 呼び出しを出す必要はありません。呼び出し側アプリケーションが接続の継続を期待しているためです。

接続ハンドルの有効範囲は、オブジェクト・ハンドルの有効範囲と同じです (215 ページの『MQOPEN 呼び出しを使用したオブジェクト・オープン』を参照)。

パラメーターの説明は、MQCONN の MQCONN 呼び出しに関する説明の中に含まれています。

呼び出しを出したときにキュー・マネージャーが静止状態の場合でも、あるいはキュー・マネージャーが終了状態の場合でも、MQCONN 呼び出しは失敗します。

MQCONN または MQCONNX の有効範囲

通常、MQCONN または MQCONNX 呼び出しの有効範囲は、発行元のスレッドです。 つまり、呼び出しから戻される接続ハンドルは、呼び出しを発行したスレッド内でのみ有効です。 そのハンドルを使用して一度に 1 つの呼び出しだけを実行できます。 別のスレッドがそのハンドルを使用している場合は、そのハンドルが無効なハンドルとして拒否されます。 アプリケーションに複数のスレッドがあって、それぞれが IBM WebSphere MQ 呼び出しを使用する場合には、それぞれのスレッドが MQCONN または MQCONNX を発行する必要があります。

1 つのプロセスで複数の MQCONN 呼び出しを発行する場合、各呼び出しを同一のキュー・マネージャーに対して発行する必要はありません。 ただし、一度に 1 つのスレッドから 1 つの WebSphere MQ 接続のみ行えます。 あるいは、単一スレッドからの複数の WebSphere MQ 接続と、任意のスレッドからの WebSphere MQ 接続を使用できるようにする 212 ページの『MQCONNX による共用 (スレッド独立) 接続』 を検討してください。[1]

アプリケーションをクライアントとして実行する場合は、1 つのスレッド内の複数のキュー・マネージャーに接続できます。

MQCONNX 呼び出しを使用したキュー・マネージャーへの接続

MQCONNX 呼び出しは、MQCONN 呼び出しに似ていますが、呼び出しの作業方法を制御するオプションを持っています。

MQCONNX に入力する場合は、キュー・マネージャー名を提供するか、または z/OS 共用キュー・システムでのキュー共用グループ名を提供できます。 MQCONNX の出力は以下のとおりです。

- 接続ハンドル (Hconn)
- 完了コード
- 理由コード

後続の MQI 呼び出しで接続ハンドルを使用します。

MQCONNX のすべてのパラメーターの説明は、MQCONNX に記載されています。 *Options* フィールドを使用すると、任意のバージョンの MQCNO に対して STANDARD_BINDING、FASTPATH_BINDING、SHARED_BINDING、または ISOLATED_BINDING を設定できます。 また、MQCONNX 呼び出しを使用して共用 (スレッド独立) 接続を作成することも可能です。 詳細は、212 ページの『MQCONNX による共用 (スレッド独立) 接続』を参照してください。

MQCNO_STANDARD_BINDING

デフォルトでは、MQCONNX は (MQCONN と同様に)、WebSphere MQ アプリケーションとローカル・キュー・マネージャー・エージェントが別々のプロセスで実行される 2 つの論理スレッドを暗黙的に指定します。 WebSphere MQ アプリケーションは WebSphere MQ 操作の要求を出し、ローカル・キュ

[1] UNIX and Linux システム上の IBM WebSphere MQ でマルチスレッド・アプリケーションを使用する場合は、アプリケーションにスレッド用の十分なスタック・サイズがあることを確認する必要があります。 マルチスレッド・アプリケーションが MQI 呼び出しを作成する場合は、単独で作成する場合でも、他のシグナル・ハンドラー (例えば、CICS) を使用する場合でも、256 KB 以上のスタック・サイズを使用することを検討してください

ー・マネージャー・エージェントは、その要求にサービスを提供します。 これは、MQCONNX 呼び出しで MQCNO_STANDARD_BINDING オプションによって定義されます。

MQCNO_STANDARD_BINDING を指定すると、 MQCONNX 呼び出しは、qm.ini または Windows レジストリーに定義されている、キュー・マネージャーの DefaultBindType 属性の値に応じて、MQCNO_SHARED_BINDING または MQCNO_ISOLATED_BINDING のいずれかを使用します。

これがデフォルト値です。

mqm ライブラリーにリンクしている場合、まずはデフォルトのバインド・タイプを使用した標準のサーバー接続が試行されます。 基礎となるサーバー・ライブラリーのロードに失敗した場合、代わりにクライアント接続が試行されます。

- MQ_CONNECT_TYPE 環境変数が指定されている場合は、MQCONN の、もしくは MQCNO_STANDARD_BINDING が指定されている場合は MQCONNX の振る舞いを変更するために以下のオプションを指定できます。 (例外は、関連する変更をアプリケーションに加えることなく管理者がファスト・パス接続をダウングレードできるよう、 MQ_CONNECT_TYPE を LOCAL または STANDARD に設定して MQCNO_FASTPATH_BINDING が指定された場合です。

値	意味
CLIENT	クライアント接続のみが試行されます。
FASTPATH	この値は以前のリリースではサポートされていましたが、現在は指定されても無視されます。
ローカル	サーバー接続のみが試行されます。 ファスト・パス接続が、通常のサーバー接続にダウングレードされます。
STANDARD	以前のリリースとの互換性のためにサポートされます。 現在この値は、LOCAL として処理されます。

- MQCONN の呼び出し時に MQ_CONNECT_TYPE 環境変数が設定されていない場合、 デフォルトのバインド・タイプを使用した標準サーバー接続が試行されます。 サーバー・ライブラリーのロードに失敗した場合、クライアント接続が試みられます。

MQCNO_FASTPATH_BINDING

トラステッド・アプリケーション では、WebSphere MQ アプリケーションとローカル・キュー・マネージャー・エージェントが同じプロセスになるように暗黙的に指定します。 エージェント・プロセスではキュー・マネージャーにアクセスするためにインターフェースを使用する必要がなくなったので、これらのアプリケーションはキュー・マネージャーの拡張機能になります。 これは、MQCONNX 呼び出しで MQCNO_FASTPATH_BINDING オプションによって定義されます。

トラステッド・アプリケーションは、WebSphere MQ のスレッド・ライブラリーにリンクする必要があります。 承認されたアプリケーションとして実行されるように WebSphere MQ アプリケーションをセットアップする方法については、MQCNO オプションを参照してください。

このオプションによって、 最大のパフォーマンスが提供されます。

注 : このオプションを使用すると、キュー・マネージャーの整合性が損なわれます。キュー・マネージャーのストレージの上書きに対する保護はありません。 また、このオプションは、キュー・マネージャー内のメッセージやその他のデータが影響を受ける可能性のあるエラーがアプリケーションにある場合にも適用されます。 このオプションを使う前に、これらの問題を検討してください。

MQCNO_SHARED_BINDING

このオプションを指定して、アプリケーションとローカル・キュー・マネージャー・エージェントが別々のプロセスで実行するようにします。 これにより、キュー・マネージャーの整合性は保持されます。つまり、キュー・マネージャーが誤ったプログラムから保護されます。 ただし、アプリケーションおよびローカル・キュー・マネージャー・エージェントはいくつかのリソースを共用します。

このオプションは、 キュー・マネージャーの整合性の保護と MQI 呼び出しのパフォーマンスの両方の観点から、MQCNO_FASTPATH_BINDING と MQCNO_ISOLATED_BINDING の中間になります。

キュー・マネージャーがこのタイプのバインディングをサポートしていない場合、MQCNO_SHARED_BINDING は無視されます。 処理は、オプションが指定されなかったものとして続行します。

MQCNO_SHARED_BINDING を使用してアプリケーションをローカル・キュー・マネージャーに接続した場合、そのアプリケーションの実行中にキュー・マネージャーを停止することができます。 アプリケーションがまだ実行されている間にキュー・マネージャーを再始動すると、キュー・マネージャーを開始しようとする操作がエラー AMQ7018 で失敗します。キュー・マネージャーが必要とするリソースを、まだアプリケーションが保持しているからです。

キュー・マネージャーを開始するためには、アプリケーションを停止する必要があります。

MQCNO_ISOLATED_BINDING

MQCNO_SHARED_BINDING の場合と同様、このオプションを指定して、アプリケーションとローカル・キュー・マネージャー・エージェントが別々のプロセスで実行するようにします。 ただし、この場合、アプリケーション・プロセスとローカル・キュー・マネージャー・エージェントがリソースを共用しないという意味で、互いに独立します。

これは、キュー・マネージャーの整合性を保護するために最も安全なオプションですが、MQI 呼び出しのパフォーマンスは最も遅くなります。

キュー・マネージャーがこのタイプのバインディングをサポートしていない場合、MQCNO_ISOLATED_BINDING は無視されます。 処理は、オプションが指定されなかったものとして続行します。

MQCNO_CLIENT_BINDING

このオプションを指定すると、アプリケーションはクライアント接続のみを試行します。 このオプションには以下の制約があります。

- MQCNO_CLIENT_BINDING は、z/OS 上では MQRC_OPTIONS_ERROR を出して拒否されます。
- MQCNO_CLIENT_BINDING は、MQCNO_STANDARD_BINDING 以外の MQCNO バインディング・オプションとともに指定されると MQRC_OPTIONS_ERROR を出して拒否されます。
- MQCNO_CLIENT_BINDING は、バインド・タイプを選択する独自のメカニズムを持っている Java では使用できません。
- V 7.5.0.7 IBM WebSphere MQ Version 7.5.0、フィックスパック 7 より前では、MQCNO_CLIENT_BINDING は、バインド・タイプを選択する独自のメカニズムを持っている .NET では使用できません。 Version 7.5.0, Fix Pack 7 以降では、.NET を使用するときの MQCNO_CLIENT_BINDING の制限はなくなりました。
- MQCONNX の呼び出し時に MQ_CONNECT_TYPE 環境変数が設定されていない場合、デフォルトのバインド・タイプを使用した標準サーバー接続が試行されます。 サーバー・ライブラリーのロードに失敗した場合、クライアント接続が試みられます。

MQCNO_LOCAL_BINDING

このオプションを指定すると、アプリケーションはサーバー接続を試行します。MQCNO_FASTPATH_BINDING、MQCNO_ISOLATED_BINDING、または MQCNO_SHARED_BINDING のいずれかが同時に指定されていると、接続はそちらのタイプに代わります。それについてはこのセクションで説明されています。 そうでない場合は、デフォルトのバインド・タイプを使用した通常のサーバー接続が試行されます。 MQCNO_LOCAL_BINDING には以下の制約があります。

- MQCNO_LOCAL_BINDING は、z/OS 上では無視されます。
- MQCNO_LOCAL_BINDING は、MQCNO_RECONNECT_AS_DEF 以外の MQCNO 再接続オプションとともに指定されると MQRC_OPTIONS_ERROR を出して拒否されます。
- MQCNO_LOCAL_BINDING は、バインド・タイプを選択する独自のメカニズムを持っている Java では使用できません。

- V 7.5.0.7 IBM WebSphere MQ Version 7.5.0、フィックスパック 7 より前では、MQCNO_LOCAL_BINDING は、バインド・タイプを選択する独自のメカニズムを持っている .NET では使用できません。 Version 7.5.0, Fix Pack 7 以降では、.NET を使用するときの MQCNO_LOCAL_BINDING の制限はなくなりました。
- MQCONNX の呼び出し時に MQ_CONNECT_TYPE 環境変数が設定されていない場合、デフォルトのバインド・タイプを使用した標準サーバー接続が試行されます。 サーバー・ライブラリーのロードに失敗した場合、クライアント接続が試みられます。

z/OS では、これらのオプションが許容されますが、実行されるのは標準のバインド済みの接続だけです。 z/OS 用の MQCNO バージョン 3 では、次の 4 つの代わりのオプションが許可されます。

MQCNO_SERIALIZE_CONN_TAG_QSG

これによって、アプリケーションは、いつでもアプリケーションのインスタンスが一度に 1 つだけ、キュー共用グループで実行されるように要求することができます。 これは、アプリケーションから指定または派生させられた値とともに接続タグの使用を登録することによって実現されます。 バージョン 3 の MQCNO では、128 バイトの文字ストリングが指定されます。

MQCNO_RESTRICT_CONN_TAG_QSG

これは、アプリケーションが複数のプロセス (または TCB) で構成され、それぞれがキュー・マネージャーに接続できる場合に使用されます。 接続は、タグが現在使用されていない場合、または要求アプリケーションが同じ処理範囲内にある場合にのみ許可されます。 これはタグの所有者と同じキュー共用グループ内にある MVS アドレス・スペースです。

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

これは MQCNO_SERIALIZE_CONN_TAG_QSG と似ていますが、要求タグが既に使用中かどうかを確認するため、ローカル・キュー・マネージャーだけに問い合わせが行われます。

MQCNO_RESTRICT_CONN_TAG_Q_MGR

これは MQCNO_RESTRICT_CONN_TAG_QSG と似ていますが、要求タグが既に使用中かどうかを確認するため、ローカル・キュー・マネージャーだけに問い合わせが行われます。

トラステッド・アプリケーションの制約事項

以下の制約事項は、トラステッド・アプリケーションに適用されます。

- トラステッド・アプリケーションはキュー・マネージャーから明示的に切断する必要がある。
- トラステッド・アプリケーションは、キュー・マネージャーを終了する前に endmqm コマンドを使って停止する必要がある。
- MQCNO_FASTPATH_BINDING と共に非同期信号とタイマー割り込み (`sigkill` など) を使用してはならない。
- すべてのプラットフォームにおいて、同じプロセス内の別のスレッドが別のキュー・マネージャーに接続されているときには、トラステッド・アプリケーション内のスレッドをキュー・マネージャーに接続できない。
- WebSphere MQ (UNIX and Linux システム用) では、すべての MQI 呼び出しに有効なユーザー ID およびグループ ID として、mqm を使用する必要がある。 これらの ID は、認証が必要な非 MQI 呼び出し (例えば、ファイルのオープン) を行う前に変更できますが、次の MQI 呼び出しを行う前に mqm に戻しておく必要があります。
- WebSphere MQ for HP-UX で、マルチスレッドのファースト・パス・アプリケーションが、デフォルトよりも大きなスタック・サイズを設定する必要が生じると思われる。 256 KB のサイズを使用する。
- WebSphere MQ for Windows では、トラステッド 64 ビット・アプリケーションはサポートされていない。 トラステッド 64 ビット・アプリケーションを実行しようとすると、 標準バウンド接続にダウングレードされる。
- WebSphere MQ on UNIX and Linux システムでは、トラステッド 32 ビット・アプリケーションはサポートされません。 トラステッド 32 ビット・アプリケーション を実行しようとすると、標準バウンド接続にダウングレードされる。

MQCONNX による共用 (スレッド独立) 接続

この情報を使用して、 MQCONNX との共有接続といくつかの使用上の注意について知ることができます。

注 : WebSphere MQ for z/OS ではサポートされない。

WebSphere MQ for z/OS 以外の WebSphere MQ プラットフォームの場合、 MQCONN によって作成された接続は、 接続を作成したスレッドでのみ使用できます。 MQCONNX 呼び出しでは、オプションを使用することで、プロセス内のすべてのスレッドで共用できる接続を作成することができます。 MQI 呼び出しを同じスレッド上で発行しなければならないトランザクション環境でアプリケーションが実行されている場合、以下のデフォルト・オプションを使用する必要があります。

MQCNO_HANDLE_SHARE_NONE
非共用接続を作成します。

その他のほとんどの環境では、スレッドに依存しない以下の共用接続オプションの 1 つを使用できます。

MQCNO_HANDLE_SHARE_BLOCK
共用接続を作成します。 MQCNO_HANDLE_SHARE_BLOCK 接続では、 現在別のスレッド上の MQI 呼び出しが接続を使用している場合は、その現行の MQI 呼び出しが完了するまで MQI 呼び出しを待機させます。

MQCNO_HANDLE_SHARE_NO_BLOCK
共用接続を作成します。 MQCNO_HANDLE_SHARE_NO_BLOCK 接続では、 現在別のスレッド上の MQI 呼び出しが接続を使用している場合は、MQI 呼び出しは MQRC_CALL_IN_PROGRESS の理由で即時に失敗します。

MTS (Microsoft Transaction Server) 環境では、MQCNO_HANDLE_SHARE_NONE がデフォルト値です。 MTS 環境では、MQCNO_HANDLE_SHARE_BLOCK がデフォルト値です。

接続ハンドルは MQCONNX 呼び出しから戻されます。 MQCONNX から戻されたハンドルは、各呼び出しに関連付けることによって、これより後に行われる、プロセス内のすべてのスレッドからの MQI 呼び出しに使用できます。 1 つの共用ハンドルを使用する各 MQI 呼び出しは、スレッドの枠を超えて逐次化されます。

例えば、共用ハンドルでは、次のような一連のアクティビティーが可能です。

1. スレッド 1 が MQCONNX を発行し、共用ハンドル *h1* を取得する
2. スレッド 1 がキューをオープンし、*h1* を使用して読み取り (get) 要求を発行する
3. スレッド 2 が *h1* を使用して書き込み (put) 要求を発行する
4. スレッド 3 が *h1* を使用して書き込み (put) 要求を発行する
5. スレッド 2 が *h1* を使用して MQDISC を発行する

ハンドルがいずれかのスレッドによって使用されている間、 他のスレッドでは、その接続にアクセスできなくなります。 先に発行された他のスレッドからの呼び出しが完了するまで次のスレッドを待機させておくことが可能な場面では、 MQCONNX にオプション MQCNO_HANDLE_SHARE_BLOCK を使用してください。

ただし、ブロックすると問題が生じる場合があります。 例えば、ステップ 212 ページの『2』 で、スレッド 1 が読み取り (get) 要求を発行し、これが未到着の可能性のあるメッセージを待機しているとします (待機状態の get 要求)。 この場合、スレッド 2 とスレッド 3 も、スレッド 1 でこの get 要求が完了するまで待機状態のまま (ブロックされた状態) になります。 そのハンドルで別の MQI 呼び出しが既に実行されている場合に MQI 呼び出しがエラーで戻るようにするには、 MQCONNX にオプション MQCNO_HANDLE_SHARE_NO_BLOCK を使用してください。

共用接続の使用上の注意

1. オブジェクトをオープンしたときに作成されるオブジェクト・ハンドル (Hobj) はすべて、Hconn と関連付けられます。それで、Hconn が共用される場合は、Hobj も共用されることになり、Hconn を使用するすべてのスレッドで使用できるようになります。 同様に、Hconn 下で開始されるすべての作業単位も Hconn に関連付けられます。それで、作業単位も、Hconn が共用されればスレッドの枠を超えて共用されることになります。

2. 共用 Hconn を切断するための MQDISC は、対応する MQCONNX を呼び出したスレッドに限らず、すべての スレッドから呼び出せます。 MQDISC は Hconn を終了させ、Hconn はどのスレッドからも使用できなくなります。
3. 単一スレッドで複数の共用 Hconn を並行して使用することもできます。例えば、MQPUT を使用して、ある共用 Hconn の下に 1 つのメッセージを put して、それから別の共用 Hconn を使用して別のメッセージを put することを、それぞれの操作を別々のローカル作業単位に属させたまま行うことができます。
4. 共用 Hconns は、グローバル作業単位では使用できません。

MQ_CONNECT_TYPE を指定した *MQCONNX* 呼び出しオプションの使用

この情報を使用して、MQ_CONNECT_TYPE と共に使用されるさまざまな MQCONNX 呼び出しオプションを理解します。

WebSphere MQ for IBM i、WebSphere MQ for Windows、 および WebSphere MQ (UNIX and Linux システム用) では、 MQCONNX 呼び出しで使用される MQCNO 構造体の *Options* フィールドに指定されたバインディングのタイプと組み合わせて、環境変数 MQ_CONNECT_TYPE を使用 することができます。

表 *33. MQ_CONNECT_TYPE* 環境変数

MQCONNX 呼び出しオプション	**MQ_CONNECT_TYPE 環境変数**	**結果**
STANDARD	UNDEFINED	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	ローカル	STANDARD

MQCNO_STANDARD_BINDING が指定されていない場合、MQCNO_NONE を使用でき、この場合、MQCNO_STANDARD_BINDING がデフォルトになります。

MQDISC を使用したキュー・マネージャーからのプログラムの切断

この情報を使用して、MQDISC を使用したキュー・マネージャーからのプログラムの切断について学習します。

MQCONN または MQCONNX 呼び出しを用いてキュー・マネージャーに接続したプログラムが、すべてのキュー・マネージャーとの対話を終了したら、MQDISC 呼び出しを用いて接続を終了します。ただし、以下の場合を除きます。

- CICS Transaction Server for z/OS アプリケーションでは、MQCONNX が使用されていてアプリケーションの終了前に接続タグを除去する必要がない限り、呼び出しはオプションです。
- WebSphere MQ for IBM i では、オペレーティング・システムからサインオフする際に、暗黙の MQDISC 呼び出しが行われます。

MQDISC 呼び出しへの入力として、キュー・マネージャーに接続したときに MQCONN または MQCONNX によって戻された接続ハンドル (Hconn) を提供する必要があります。

z/OS 上の CICS を除き、MQDISC が呼び出された後は、接続ハンドル (Hconn) は無効になります。MQCONN または MQCONNX を再度呼び出すまで、これ以上 MQI 呼び出しを発行することはできません。 MQDISC は、依然オープンされ、このハンドルを用いるオブジェクトに対して暗黙的な MQCLOSE を行います。

WebSphere MQ for z/OS での接続に MQCONNX を使用した場合、MQDISC も MQCONNX が確立した接続タグの有効範囲を終了します。 しかし、CICS、IMS、または RRS アプリケーションでは、接続タグと関連したアクティブなリカバリー単位があれば、MQDISC は理由コード MQRC_CONN_TAG_NOT_RELEASED で拒否されます。

パラメーターの説明は、MQDISC の MQDISC 呼び出しに関する説明の中に含まれています。

MQDISC が発行されない場合

標準の非共用接続 (Hconn) は、作成スレッドが終了されるとクリーンアップされます。 しかし、共用接続は、プロセス全体が終了されるときにのみ、暗黙的にバックアウトされ、切断されます。 Hconn がまだ存在しているときに、その共用 Hconn を作成したスレッドが終了させられても、Hconn は引き続き使用可能です。

権限検査

MQCLOSE および MQDISC 呼び出しは、通常、権限検査は行いません。

正常なイベントの進行では、WebSphere MQ オブジェクトのオープンまたは接続の権限を持っているジョブが、そのオブジェクトをクローズしたり切断したりします。 WebSphere MQ オブジェクトへの接続またはオープンを行ったジョブの権限が取り消された場合でも、MQCLOSE および MQDISC 呼び出しは受け付けられます。

オブジェクトのオープンとクローズ

ここでは、WebSphere MQ オブジェクトのオープンとクローズについて説明します。

以下の操作を実行するには、まず、関連する WebSphere MQ オブジェクトを 開く 必要があります。

- メッセージをキューに書き込む
- メッセージをキューから読み取る (ブラウズまたは取り出し)
- オブジェクト属性を設定する
- 任意のオブジェクトの属性を照会する

オブジェクトをオープンするには、オブジェクトに対して行いたいことを指定するオプションを用いて、MQOPEN 呼び出しを使用します。 唯一の例外として、キューに単一メッセージを書き込みたいときは、キューを即時にクローズします。 この場合、MQPUT1 呼び出しを使用してオープン の段階を省略することができます (233 ページの『MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み』 を参照)。

MQOPEN 呼び出しを使用してオブジェクトをオープンするには、プログラムがキュー・マネージャーに接続されていなければなりません。 すべての環境に関する詳細については、206 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』 に説明があります。

オープンできる WebSphere MQ オブジェクトのタイプには、以下の 4 つがあります。

- キュー
- 名前リスト
- プロセス定義
- キュー・マネージャー

これらのオブジェクトは、MQOPEN 呼び出しを使用して、すべて同じ方法でオープンします。 WebSphere MQ オブジェクトの詳細については、オブジェクトを参照してください。

同じオブジェクトを複数回オープンすることができますが、そのたびに新しいオブジェクト・ハンドルを取得します。1 つのハンドルを用いてキュー上のメッセージをブラウズし、さらに別のハンドルを用いて同じキューからメッセージを除去したい場合があります。 これによって、同じオブジェクトをクローズして再オープンするためのリソースの消費を防ぎます。 また、メッセージのブラウズおよび 除去を同時に行うためにキューをオープンすることもできます。

さらに、単一の MQOPEN で複数のオブジェクトをオープンでき、MQCLOSE を使ってそれらをクローズできます。 この方法については、234 ページの『配布リスト』 を参照してください。

オブジェクトをオープンしようとするとき、MQOPEN 呼び出しで指定したオプションに対して、そのオブジェクトをオープンする権限を持っているかどうかをキュー・マネージャーが検査します。

プログラムがキュー・マネージャーから切断されるとき、オブジェクトは自動的にクローズされます。 IMS の環境では、プログラムが IMS の GU (get unique) 呼び出しの後、新しいユーザー用の処理を開始するとき、切断が強制的に実行されます。 IBM i プラットフォームでは、ジョブが終了したときに、オブジェクトは自動的にクローズされます。

プログラミングでは、オープンしたオブジェクトをクローズすることをお勧めします。 これを行うには、MQCLOSE 呼び出しを用います。

オブジェクトのオープンとクローズについて詳しくは、以下のリンクを参照してください。

- 215 ページの『MQOPEN 呼び出しを使用したオブジェクト・オープン』
- 223 ページの『動的キューの作成』
- 223 ページの『リモート・キューのオープン』
- 224 ページの『MQCLOSE 呼び出しを使用したオブジェクトのクローズ』

関連概念

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

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

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

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

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

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

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

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

MQOPEN 呼び出しを使用したオブジェクト・オープン

この情報を使用して、MQOPEN 呼び出しを使用したオブジェクトのオープンについて学習します。

MQOPEN 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。 z/OS 上の CICS アプリケーションの場合、定数 MQHC_DEF_HCONN (値はゼロ) を指定するか、MQCONN または MQCONNX 呼び出しによって戻される接続ハンドルを使用することができます。他のプログラムの場合は、常に MQCONN 呼び出しか MQCONNX 呼び出しによって戻される接続ハンドルを用います。
- オープンしたいオブジェクトの記述。これはオブジェクト記述子構造体 (MQOD) を用いて行います。
- 呼び出しの処置を制御する 1 つまたは複数のオプション。

MQOPEN の出力は以下のとおりです。

- オブジェクトへのアクセスを表すオブジェクト・ハンドル。 後続の MQI 呼び出しへの入力にこれを使用します。
- 動的キューを作成する場合、修正されたオブジェクト記述子構造体 (ユーザーのプラットフォームでサポートされる)。
- 完了コード。
- 理由コード。

オブジェクト・ハンドルの有効範囲

オブジェクト・ハンドル (Hobj) の有効範囲は、接続ハンドル (Hconn) の有効範囲と同じです。

この点については、208 ページの『MQCONN または MQCONNX の有効範囲』と 212 ページの『MQCONNX による共用 (スレッド独立) 接続』を参照してください。 ただし、いくつかの環境では、追加の考慮事項があります。

CICS
CICS プログラムでは、MQOPEN 呼び出しを出したのと同じ CICS タスク内でのみ、オブジェクト・ハンドルを使用できます。

IMS と z/OS バッチ
IMS およびバッチ環境では、同一のタスク内ではハンドルを使用できますが、サブタスク内では使用できません。

MQOPEN 呼び出しのパラメーターの説明については、MQOPEN を参照してください。

以下の節では、MQOPEN の入力として指定しなければならない情報を取り上げます。

オブジェクトの識別 (MQOD 構造体)

MQOD 構造体を使用して、オープンしたいオブジェクトを識別します。 この構造体は MQOPEN 呼び出し用の入力パラメーターです。 (動的キューを作成するために MQOPEN 呼び出しを使用したときは、この構造体がキュー・マネージャーによって修正されます。)

MQOD 構造体の詳細については、MQOD を参照してください。

配布リストを生成するための MQOD 構造体の使用については、234 ページの『配布リスト』にある 236 ページの『MQOD 構造体の使用』を参照してください。

名前の解決

MQOPEN 呼び出しがキューとキュー・マネージャーの名前を解決する方法。

注 : キュー・マネージャーの別名は、RNAME フィールドを除いたリモート・キュー定義です。

WebSphere MQ キューをオープンするとき、MQOPEN 呼び出しによって、指定するキュー名におけるネーム・レゾリューション機能が実行されます。 これによって、キュー・マネージャーが以後の操作をどのキューに対して行うのかを決定します。 これは、オブジェクト記述子 (MQOD) に別名キューまたはリモート・キューの名前を指定したときは、呼び出しが名前をローカル・キューまたは伝送キューに解決することを意味します。 キューが任意のタイプの入力、ブラウズ、または設定用にオープンされている場合、ローカル・キューがあれば名前は解決されますが、ローカル・キューがなければ失敗に終わります。 キューが出力専用、照会専用、または出力照会両用にオープンされている場合のみ、非ローカル・キューに解決されます。 ネーム・レゾリューション処理の概要については、217 ページの表 34 を参照してください。 *ObjectQMgrName* に指定した名前は、 *ObjectName* に指定した名前より 前に 解決されます。

217 ページの表 34 は、キュー・マネージャー名の別名を定義するために、リモート・キューのローカル定義を使用する方法も示しています。 これにより、メッセージをリモート・キューに書き込むときに使用する伝送キューを選択できるので、例えば、多数のリモート・キュー・マネージャー向けのメッセージに対して単一の伝送キューを用いることができます。

次の表を使用するには、最初に左側の 2 つの列 (見出し **MQOD への入力**のもとにある) を読み、適切な事例を選択します。 次に、対応する行をすべて読み、指示に従います。 **解決済みの名前**列の指示に従い、**MQOD への入力**列に戻り、指示どおりに値を挿入するか、結果が提供された状態で表を終了することができます。 例えば、*ObjectName* を入力するように要求されたとします。

表 *34. MQOPEN* 使用時のキュー名の解決

MQOD への入力		**解決済みの名前**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**伝送キュー**
ブランクまたはローカル・キュー・マネージャー	CLUSTER 属性を含まないローカル・キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	該当しない (使用されたローカル・キュー)
ブランク・キュー・マネージャー	CLUSTER 属性を含むローカル・キュー	ワークロード管理で選択されているクラスター・キュー・マネージャー、または PUT で選択されている特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE および使用されたローカル・キュー SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ローカル・キュー・マネージャー	CLUSTER 属性を含むローカル・キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	該当しない (使用されたローカル・キュー)
ブランクまたはローカル・キュー・マネージャー	モデル・キュー	ローカル・キュー・マネージャー	生成された名前	該当しない (使用されたローカル・キュー)
ブランクまたはローカル・キュー・マネージャー	CLUSTER 属性が指定されている、または指定されていない別名キュー	別名キュー定義オブジェクト内の *ObjectQMgrName* を変更せず、*ObjectName* を *BaseQName* に設定して、ネーム・レゾリューションを再実行する。 *ObjectQMgrName* が指定されていてローカルに定義されている別名に解決してはならない。ただし、*ObjectQMgrName* がブランクであるクラスター別名 (他のキュー・マネージャーにホストされている) への解決は可能。		
ローカル・キュー・マネージャー	CLUSTER 属性を含む別名キュー	別名はローカルで定義されていないクラスター・キュー、または別名と同じ *ObjectName* を持つクラスター・キューに解決してはならない。		
ブランク・キュー・マネージャー	CLUSTER 属性を含む別名キュー	別名は、別名と同じ *ObjectName* を持つクラスター・キューに解決できる。		

表 *34. MQOPEN* 使用時のキュー名の解決 *(*続き*)*

MQOD への入力		**解決済みの名前**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**伝送キュー**
ブランクまたはローカル・キュー・マネージャー	リモート・キューのローカル定義	*ObjectQMgrName* を *RemoteQMgrName* に、*ObjectName* を *RemoteQName* に設定して、ネーム・レゾリューションを再実行する。リモート・キューを解決してはならない。		非ブランクの場合は、*XmitQName* 属性の名前、ブランクの場合は、リモート・キュー定義オブジェクトの *RemoteQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ブランク・キュー・マネージャー	一致するローカル・オブジェクトなし。クラスター・キューは存在する。	ワークロード管理で選択されているクラスター・キュー・マネージャー、または PUT で選択されている特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
ブランクまたはローカル・キュー・マネージャー	一致するローカル・オブジェクトなし。クラスター・キューは存在ない。		エラー。キューは存在ない。	適用外
ローカル・キュー・マネージャーと同じキュー共用グループ内のキュー・マネージャー名	ローカル共用キュー	ローカル・キュー・マネージャー	入力 *ObjectName*	適用外
ローカル伝送キューの名前	(解決されない)	入力 *ObjectQMgrName*	入力 *ObjectName*	入力 *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
キュー・マネージャーの別名定義 (*RemoteQMgrName* がローカル・キュー・マネージャーの場合)	(解決されない。リモート・キュー)	*ObjectQMgrName* を *RemoteQMgrName* に設定して、ネーム・レゾリューションを再実行する。 リモート・キューに解決してはならない。	入力 *ObjectName*	非ブランクの場合は、*XmitQName* 属性の名前、ブランクの場合は、リモート・キュー定義オブジェクトの *RemoteQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)
キュー・マネージャーがローカル・オブジェクトの名前でない。クラスター・キュー・マネージャーまたはキュー・マネージャー別名は存在する。	(解決されない)	*ObjectQMgrName* または PUT で選択された特定のクラスター・キュー・マネージャー	入力 *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)

表 *34. MQOPEN* 使用時のキュー名の解決 *(続き)*				
MQOD への入力		**解決済みの名前**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**伝送キュー**
キュー・マネージャーがローカル・オブジェクトの名前でない。クラスター・オブジェクトは存在していない。	(解決されない)	入力 *ObjectQMgrName*	入力 *ObjectName*	キュー・マネージャーの *DefXmitQName* 属性。*DefXmitQName* がサポートされる。 SYSTEM.QSG.TRANSMIT.QUEUE (注を参照)

注 :

1. *BaseQName* は、別名キューの定義による基本キューの名前。
2. *RemoteQName* は、リモート・キューのローカル定義によるリモート・キューの名前。
3. *RemoteQMgrName* は、リモート・キューのローカル定義によるリモート・キュー・マネージャーの名前。
4. *XmitQName* は、リモート・キューのローカル定義による伝送キューの名前。
5. キュー共用グループ (QSG) の一部である WebSphere MQ for z/OS キュー・マネージャーの使用時、217 ページの表 34 のローカル・キュー・マネージャー名の代わりに QSG の名前を使用できます。

   ローカル・キュー・マネージャーがターゲット・キューをオープンできない場合、またはキューにメッセージを書き込めない場合、そのメッセージは、グループ内キューイングまたは WebSphere MQ チャネルのどちらかの方法で、指定の ObjectQMgrName に転送されます。
6. テーブルの *ObjectName* 列で、CLUSTER はキューの CLUSTER 属性と CLUSNL 属性の両方を表します。
7. ローカルおよびリモート・キュー・マネージャーが同じキュー共用グループにある場合、SYSTEM.QSG.TRANSMIT.QUEUE が使用されます。グループ内キューイングは使用可能になります。
8. 各クラスター送信側チャネルに対して異なるクラスター伝送キューを割り当てた場合、SYSTEM.CLUSTER.TRANSMIT.QUEUE はクラスター伝送キューの名前と同じではない可能性があります。 複数のクラスター伝送キューについて詳しくは、クラスター化: クラスター伝送キューの構成方法の計画を参照してください。
9. キュー・マネージャーがローカル・オブジェクトの名前でなく、クラスター・キュー・マネージャーまたはキュー・マネージャー別名は存在する場合。

   ObjectQMgrName を使用してキュー・マネージャー名を指定しており、その宛先に到達するクラスター・チャネルが複数個存在し、ローカル・キュー・マネージャーがそれらのチャネルをそれぞれ異なるクラス ター名で認識している場合、宛先キューのクラスター名に関係なく、これらのどのチャネルもメッセージの移動に使用される可能性があります。

   そのキューに対するメッセージが、そのキューと同じクラスター名の付いたチャネル経由でのみ送信されることを予期していた場合、これは予期しない状況となります。

   ただし、この場合は **ObjectQMgrName** が優先されます。また、そのキュー・マネージャーに到達する可能性があるすべてのチャネルが属するクラスター名に関係なく、クラスター・ワークロード・バランシングではそれらのチャネルがすべて考慮に入れられます。

別名キューをオープンすると、別名が解決する基本キューもオープンされ、リモート・キューをオープンすると、伝送キューもオープンされます。 したがって、指定したキューも、それが解決されるキューも、他方がオープンされている間は削除できません。

別名キューはローカルに定義された別の別名キュー (クラスターで共有されているもの、またはそうでないもの) に解決することはできませんが、リモートに定義されたクラスター別名キューに解決することは許可されているため、 基本キューとして指定できます。

解決したキュー名と解決したキュー・マネージャー名は、MQOD の *ResolvedQName* フィールドと *ResolvedQMgrName* フィールドにそれぞれ格納されます。

分散キューイング環境でのネーム・レゾリューションの詳細については、キュー名解決についてを参照してください。

MQOPEN 呼び出しのオプションの使用

MQOPEN 呼び出しの *Options* パラメーターでは、オープンするオブジェクトに対するアクセスを制御するため、1 つまたは複数のオプションを選択しなければなりません。 これらのオプションを指定すると、次のことを行うことができます。

- キューをオープンし、そのキューに書き込まれるすべてのメッセージがそのキューの同じインスタンスに送信されるように指定する
- メッセージを書き込むためにキューをオープンする
- メッセージをブラウズするためにキューをオープンする
- メッセージを除去できるようにキューをオープンする
- 属性を照会し設定するためにオブジェクトをオープンする (ただし、設定できるのはキューの属性のみ)
- トピックまたはトピック・ストリングをオープンして、そこへメッセージをパブリッシュする
- コンテキスト情報をメッセージに関連付ける
- セキュリティー検査に使用する代替ユーザー ID を指名する
- キュー・マネージャーが静止状態であれば、呼び出しを制御する

クラスター・キュー用の *MQOPEN* オプション

キュー・ハンドルに使用されるバインディングは *DefBind* キュー属性から取得され、値は MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP を取ることができます。

MQPUT を使用してキューに書き込まれるすべてのメッセージが同じ経路をたどって同じキュー・マネージャーに送付されるようにするには、MQOPEN 呼び出しで MQOO_BIND_ON_OPEN オプションを使用します。

MQPUT の実行時に宛先が選択されるように指定するには、MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED オプションを使用します。

MQPUT を使用してキューに書き込まれるメッセージ・グループのすべてのメッセージが、同一の宛先インスタンスに割り振られるように指定するには、MQOPEN 呼び出しで MQOO_BIND_ON_GROUP オプションを使用します。

クラスターで メッセージ・グループ を使用する場合は、グループ内のすべてのメッセージが同じ宛先で処理されるように、 MQOO_BIND_ON_OPEN または MQOO_BIND_ON_GROUP のいずれかを指定する必要があります。

これらのオプションのいずれも指定しない場合、デフォルトの MQOO_BIND_AS_Q_DEF が使用されます。

MQOD にキュー・マネージャーの名前を指定すると、そのキュー・マネージャーのキューが選択されます。キュー・マネージャーの名前をブランクにすると、任意のインスタンスを選択できます。 詳細については、349 ページの『MQOPEN およびクラスター』を参照してください。

QALIAS 定義を使用してクラスター・キューをオープンすると、いくつかのキュー属性は基本キューではなく別名キューによって定義されます。 クラスター属性は、別名キューによって指定変更される基本キュー定義の属性に含まれます。 例えば、以下のスニペットでは、クラスター・キューは MQOO_BIND_ON_OPEN ではなく MQOO_BIND_NOT_FIXED で開かれます。 クラスター・キュー定義はクラスターを通じて公示され、別名キュー定義はキュー・マネージャーに対してローカルです。

```
DEFINE QLOCAL(CLQ1)  CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

メッセージを書き込むための *MQOPEN* オプション

メッセージを書き込むためにキューまたはトピックを開くには、MQOO_OUTPUT オプションを使用します。

メッセージをブラウズするための *MQOPEN* オプション
キューをオープンしてメッセージをブラウズできるようにするには、MQOO_BROWSE オプションと一緒に MQOPEN 呼び出しを使用します。

これは、キュー・マネージャーがキューの次のメッセージを 識別するために用いるブラウズ・カーソル を作成します。 詳しくは、271 ページの『キュー上のメッセージのブラウズ』を参照してください。

注 :

1. リモート・キュー上のメッセージをブラウズすることはできません。MQOO_BROWSE オプションを用いてリモート・キューをオープンしないでください。
2. このオプションは、配布リストをオープンするときには指定できません。 配布リストの詳細については、234 ページの『配布リスト』を参照してください。
3. 共同ブラウズを使用している場合は、MQOO_CO_OP を MQOO_BROWSE と共に使用します。詳細は、Options を参照してください。

メッセージを除去するための *MQOPEN* オプション
キューからメッセージを除去するには、キューのオープンを制御する 3 つのオプションがあります。

どの MQOPEN 呼び出しにもそれらのうち 1 つしか使用できません。 これらのオプションは、プログラムのキューへのアクセスが排他的か共用かを定義します。 排他的アクセス は、キューをクローズするまでのことを意味します。メッセージを削除できるのは、ユーザーだけです。 別のプログラムが、メッセージを除去するためにキューをオープンしようとすると、その MQOPEN 呼び出しは失敗に終わります。 共有アクセス は、複数のプログラムが削除できることを意味します。キューからのメッセージ。

最もお勧めできる方法は、キューが定義されたときに決められたアクセスのタイプを受け入れることです。キュー定義には、 *Shareability* の設定と、*DefInputOpenOption* 属性。 このアクセスを受け入れるには、MQOO_INPUT_AS_Q_DEF オプションを使用します。 このオプションを使用するとき、これらの属性の設定によって アクセスのタイプがどのような影響を受けるかについては、221 ページの表 35 を参照してください。

表 *35.* キュー属性と *MQOPEN* 呼び出しのオプションがキューへのアクセスを及ぼす影響

キューの属性		**MQOPEN のオプションによるアクセスのタイプ**		
Shareability	*DefInputOpenOption*	**AS_Q_DEF**	**SHARED**	**EXCLUSIVE**
SHAREABLE	SHARED	共用	共用	排他的
SHAREABLE	EXCLUSIVE	排他的	共用	排他的
NOT_SHAREABLE*	SHARED*	排他的	排他的	排他的
NOT_SHAREABLE	EXCLUSIVE	排他的	排他的	排他的

注 : * 属性をこのように組み合わせてキューを定義することはできますが、デフォルトの入力オープン・オプションは、Shareability 属性によって上書きされます。

代替方法

- 他のプログラムがメッセージをキューから同時に除去することがあっても、アプリケーションが正常に機能することが分かっている場合は、MQOO_INPUT_SHARED オプションを使用してください。 221 ページの表 35 は、たとえこのオプションを指定しても、キューに対するアクセスが排他的になる特定の場合があることを示します。
- 他のプログラムがメッセージをキューから同時に除去できない場合のみ、アプリケーションが正常に機能することが分かっている場合は、MQOO_INPUT_EXCLUSIVE オプションを使用してください。

注 :

1. リモート・キューからはメッセージを除去できません。 したがって、MQOO_INPUT_* オプションのどれを使用してもリモート・キューをオープンすることはできません。
2. このオプションは、配布リストをオープンするときには指定できません。 詳しくは、234 ページの『配布リスト』を参照してください。

属性を設定し照会するための *MQOPEN* オプション
キューを開いてその属性を設定できるようにするには、MQOO_SET オプションを使用します。

他のタイプのオブジェクトの属性は設定できません (321 ページの『オブジェクト属性の照会と設定』を参照)。

オブジェクトの属性を照会するためにオブジェクトをオープンするには、MQOO_INQUIRE オプションを使用します。

注 : このオプションは、配布リストをオープンするときには指定できません。

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

これらのオプションを使用すると、メッセージを発信したユーザー に関連するコンテキスト情報と、メッセージを発信したアプリケーション に関連するコンテキスト情報とを区別できます。 また、メッセージをキューに書き込むときにコンテキストを設定するか、あるいはコンテキストを他のキュー・ハンドルから自動的に持ってくるかを選択することもできます。

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

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

代替ユーザー権限のための *MQOPEN* オプション
MQOPEN 呼び出しを使用してオブジェクトをオープンしようとすると、キュー・マネージャーはそのオブジェクトをオープンする権限を持っているか検査します。 ユーザーに権限がない場合、呼び出しは失敗します。

しかし、サーバー・プログラムは、サーバー自体の許可ではなく、作業対象のユーザーの許可をキュー・マネージャーに検査させることもできます。 これを行うには、MQOPEN 呼び出しの MQOO_ALTERNATE_USER_AUTHORITY オプションを使用し、MQOD 構造体の *AlternateUserId* フィールドに代替ユーザー ID を指定しなければなりません。 サーバーは、通常、サーバー自身が処理中のメッセージ内のコンテキスト情報からそのユーザー ID を取得します。

キュー・マネージャーの静止に関する *MQOPEN* オプション
z/OS の CICS 環境では、キュー・マネージャーが静止状態のときに MQOPEN 呼び出しを使用すると、呼び出しは常に失敗します。

他の z/OS 環境、IBM i、Windows システム、および UNIX and Linux システム環境では、キュー・マネージャーが静止状態のときに MQOPEN 呼び出しの MQOO_FAIL_IF_QUIESCING オプションを使用する場合にのみ、その呼び出しは失敗します。

ローカル・キュー名の解決の *MQOPEN* オプション
ローカル・キュー、別名キュー、またはモデル・キューをオープンすると、ローカル・キューが返されます。

しかし、リモート・キューまたはクラスター・キューをオープンすると、MQOD 構造の *ResolvedQName* および *ResolvedQMgrName* フィールドには、リモート・キュー定義にあるリモート・キューおよびリモート・キュー・マネージャーの名前、または選択したリモート・クラスター・キューが入ります。

MQOPEN 呼び出しの MQOO_RESOLVE_LOCAL_Q オプションを使用して、MQOD 構造の *ResolvedQName* に、オープンされたローカル・キュー名が入るようにします。 同様に *ResolvedQMgrName* には、ローカル・キューをホスティングするローカル・キュー・マネージャー名が入ります。 このフィールドは MQOD

構造体のバージョン 3 でのみ使用可能です。 構造体がバージョン 3 より前の場合、エラーが戻されずに MQOO_RESOLVE_LOCAL_Q が無視されます。

例えば、リモート・キューをオープンするときに MQOO_RESOLVE_LOCAL_Q を指定すると、メッセージが書き込まれる伝送キューの名前は *ResolvedQName* となります。 伝送キューをホスティングするローカル・キュー・マネージャーの名前は *ResolvedQMgrName* となります。

動的キューの作成

アプリケーションの終了後にそのキューが必要ない場合には、動的キューを使用してください。

例えば、応答先キューに動的キューを使用できます。 メッセージをキューに書き込む場合、MQMD 構造体の *ReplyToQ* フィールドに応答先キューの名前を指定します (226 ページの『MQMD 構造体を使用するメッセージの定義』 を参照)。

動的キューを作成するには、モデル・キューと呼ばれるテンプレートを MQOPEN 呼び出しと共に使用します。 WebSphere MQ コマンドまたは操作および制御パネルを使用して、モデル・キューを作成します。 作成される動的キューは、モデル・キューの属性を取り入れます。

MQOPEN を呼び出すときに、MQOD 構造体の *ObjectName* フィールドにモデル・キューの名前を指定します。 その呼び出しが完了すると、*ObjectName* フィールドが作成された動的キューの名前に設定されます。 また、*ObjectQMgrName* フィールドがローカル・キュー・マネージャーの名前に設定されます。

次の 3 つの方法で、作成する動的キューの名前を指定できます。

- MQOD 構造体の *DynamicQName* フィールドに、キューに付けたい名前をフルネームで指定する。
- 名前の (33 文字より少ない) 接頭部を指定し、キュー・マネージャーに名前の残りの部分を生成させる。 これは、キュー・マネージャーが固有な名前を生成するが、プログラマーもある程度の制御を行うことができる (例えば、各ユーザーにある一定の接頭部を使用させるか、ある特定の接頭部の付いた名前を持つキューに特別のセキュリティー区分を与える) ことを意味します。 この方法を使用するには、*DynamicQName* フィールドの最後の非ブランク文字にアスタリスク (*) を指定します。 動的キュー名に単一のアスタリスク (*) を指定しないでください。
- キュー・マネージャーに、キューに付けたい名前を完全な形で生成させる。 この方法を使用するには、*DynamicQName* フィールドの先頭文字位置にアスタリスク (*) を指定します。

これらの方法について詳しくは、DynamicQName フィールドの説明を参照してください。

動的キューについて詳しくは、 動的キューとモデル・キュー を参照してください。

リモート・キューのオープン

リモート・キューは、アプリケーションが接続しているキュー・マネージャーとは別のキュー・マネージャーが所有するキューです。

リモート・キューをオープンするときは、ローカル・キューの場合と同じように MQOPEN 呼び出しを使用します。 次の方法でキューの名前を指定できます。

1. MQOD 構造体の *ObjectName* フィールドに、リモート・キューの名前 (ローカル・キュー・マネージャーに識別されているもの) を指定する。

   注: この場合、*ObjectQMgrName* フィールドはブランクのままにしてください。
2. MQOD 構造体の *ObjectName* フィールドに、リモート・キューの名前 (リモート・キュー・マネージャーに識別されているもの) を指定する。 *ObjectQMgrName* フィールドには、次のどちらかを指定する。

   - リモート・キュー・マネージャーと同じ名前の伝送キューの名前。 名前とその大文字小文字 (大文字のみ、小文字のみ、大文字小文字混合) は、正確に 一致していなければならない。
   - 宛先となるキュー・マネージャーや伝送キューを解決する、キュー・マネージャー別名オブジェクトの名前。

   これによって、メッセージの宛先と、そこに到達するために書き込まれる必要がある伝送キューがキュー・マネージャーに通知されます。

3. *DefXmitQname* がサポートされている場合、MQOD 構造体の *ObjectName* フィールドに、リモート・キューの名前 (リモート・キュー・マネージャーに識別されているもの) を指定する。

   注: *ObjectQMgrName* フィールドには、リモート・キュー・マネージャーの名前を設定してください (この場合は、ブランクにしておくことはできません)。

MQOPEN を呼び出した場合にはローカル名だけが妥当性検査されます。最後の検査では、使用される伝送キューの存在が検査されます。

これらのメソッドは、217 ページの表 34 に要約されています。

MQCLOSE 呼び出しを使用したオブジェクトのクローズ

オブジェクトをクローズするには、MQCLOSE 呼び出しを使用します。

オブジェクトがキューの場合は、以下の点に注意してください。

- 一時動的キューをクローズする前に、空にする必要はありません。

  一時動的キューをクローズすると、その中に残っているメッセージと共にそのキューも削除されます。これは、たとえそのキューに対してコミットされていない MQGET、MQPUT、または MQPUT1 呼び出しがあっても同様です。
- WebSphere MQ for z/OS では、MQGMO_SET_SIGNAL オプションを持つ MQGET 要求がそのキューに対して未解決の場合は、それらの要求は取り消されます。
- MQOO_BROWSE オプションを用いてキューをオープンした場合は、ブラウズ・カーソルが破壊されます。

クローズは同期点と無関係なので、同期点の前でも後でもキューをクローズできます。

MQCLOSE 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。 オープンに使用したものと同じ接続ハンドルを使用するか、または z/OS 上の CICS アプリケーションの場合は、定数 MQHC_DEF_HCONN (値はゼロ) を指定することができます。
- クローズしたいオブジェクトのハンドル。 これは MQOPEN 呼び出しの出力から得られます。
- *Options* フィールドに MQCO_NONE を指定する (永続動的キューをクローズする場合以外)。
- メッセージがまだある場合でも、キュー・マネージャーがキューを削除するかどうかを判別する制御オプション (永続動的キューをクローズする場合)。

MQCLOSE の出力は以下のとおりです。

- 完了コード
- 理由コード
- MQHO_UNUSABLE_HOBJ という値にリセットされるオブジェクト・ハンドル

MQCLOSE 呼び出しのパラメーターの説明については、MQCLOSE を参照してください。

キューへのメッセージの書き込み

この情報を使用して、メッセージをキューに書き込む方法について学習します。

キューにメッセージを書き込むには MQPUT 呼び出しを使用します。 同じキューにいくつものメッセージを書き込むときは、最初の MQOPEN 呼び出しに続けて MQPUT を反復使用できます。 すべてのメッセージをキューに書き込んだら、MQCLOSE を呼び出します。

キュー・メッセージを 1 つだけ書き込んで、その後すぐにキューをクローズしたい場合は、MQPUT1 呼び出しを使用できます。 MQPUT1 は、次の一連の呼び出しと同じ機能を実行します。

- MQOPEN
- MQPUT
- MQCLOSE

しかし一般的には、複数のメッセージをキューに書き込むには、MQPUT 呼び出しを使用する方が効果的です。 これは、メッセージのサイズと、作業を行っているプラットフォームに応じて異なります。

メッセージをキューに書き込む方法について詳しくは、以下のリンクを参照してください。

- 225 ページの『MQPUT 呼び出しを使用したローカル・キューへのメッセージの書き込み』
- 230 ページの『リモート・キューへのメッセージの書き込み』
- 230 ページの『メッセージ・プロパティーの設定』
- 231 ページの『コンテキスト情報の制御』
- 233 ページの『MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み』
- 234 ページの『配布リスト』
- 239 ページの『書き込み呼び出しが失敗する場合』

関連概念

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

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

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

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

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

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

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

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

MQPUT 呼び出しを使用したローカル・キューへのメッセージの書き込み

ここでは、MQPUT 呼び出しを使用してローカル・キューにメッセージを書き込むための情報を取り上げます。

MQPUT 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル (Hconn)。
- キュー・ハンドル (Hobj)。
- キューに書き込みたいメッセージについての記述。 これは、メッセージ記述子構造体 (MQMD) の形式になります。
- 書き込みメッセージ・オプション構造体の形式による制御情報 (MQPMO)。
- メッセージ内に含まれるデータの長さ (MQLONG)。
- メッセージ・データそのもの。

MQPUT 呼び出しの出力は、以下のとおりです。

- 理由コード (MQLONG)

- 完了コード (MQLONG)

呼び出しが正常に完了した場合は、オプション構造体とメッセージ記述子構造体も戻します。 この呼び出しは、メッセージ送信先のキューおよびキュー・マネージャーの名前を示すために、オプション構造体を変更します。 書き込むメッセージの ID に対して、キュー・マネージャーが固有な値を生成するよう要求する場合 (MQMD 構造体の *MsgId* フィールドに 2 進数のゼロを指定することによって)、呼び出しは、この構造体を戻す前に *MsgId* フィールドにその値を入れます。 この値は、次に MQPUT を発行するまでにリセットしてください。

MQPUT 呼び出しの説明については、MQPUT を参照してください。

MQPUT 呼び出しの入力として指定しなければならない情報の詳しい説明については、以下のリンクを参照してください。

- 226 ページの『ハンドルの指定』
- 226 ページの『MQMD 構造体を使用するメッセージの定義』
- 226 ページの『MQPMO 構造体を使用するオプションの指定』
- 229 ページの『メッセージ内のデータ』
- 230 ページの『メッセージの書き込み: メッセージ・ハンドルの使用』

ハンドルの指定

z/OS アプリケーションの CICS の接続ハンドル (*Hconn*) には、定数 MQHC_DEF_HCONN (値ゼロ) を指定するか、MQCONN 呼び出しまたは MQCONNX 呼び出しによって戻される接続ハンドルを使用することができます。 その他のアプリケーションの場合は、常に MQCONN 呼び出しまたは MQCONNX 呼び出しから返される接続ハンドルを使用します。

どの作業環境でも、MQOPEN 呼び出しから返される同じキュー・ハンドル (*Hobj*) を使用してください。

MQMD 構造体を使用するメッセージの定義

メッセージ記述子 (MQMD) は、MQPUT および MQPUT1 呼び出しに対する入出力パラメーターです。 これを使用して、キューに書き込むメッセージを定義します。

メッセージに対して MQPRI_PRIORITY_AS_Q_DEF または MQPER_PERSISTENCE_AS_Q_DEF が指定されていて、さらにキューがクラスター・キューである場合は、MQPUT で解決されるキューの値が使用されます。 そのキューが MQPUT では使用できないキューの場合、呼び出しは失敗します。 詳細については、キュー・マネージャー・クラスターの構成を参照してください。

注 : *MsgId* と *CorrelId* が固有なものとなるようにするため、新しいメッセージを置く前に MQPMO_NEW_MSG_ID と MQPMO_NEW_CORREL_ID を使用してください。 これらのフィールドの値は、MQPUT の正常終了時に戻されます。

MQMD で記述するメッセージ・プロパティーの概要については、9 ページの『IBM WebSphere MQ メッセージ』を参照してください。その構造体そのものの説明については、MQMD を参照してください。

MQPMO 構造体を使用するオプションの指定

MQPMO (書き込みメッセージ・オプション) 構造体を使用して、MQPUT および MQPUT1 呼び出しにオプションを渡します。

以下の節を参考にして、この構造体の各フィールドに情報を入力してください。 構造体の説明については、MQPMO を参照してください。

この構造体には、以下のフィールドがあります。

- *StrucId*
- *Version*
- *Options*
- *Context*

- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

これらのフィールドの内容は次のとおりです。

StrucId
構造体を書き込みメッセージ・オプション構造体として識別します。 これは 4 文字フィールドです。 必ず MQPMO_STRUC_ID を指定します。

バージョン
構造体のバージョン番号を記述します。 デフォルトは MQPMO_VERSION_1 です。 MQPMO_VERSION_2 を入力すると、配布リストを使用することができます (234 ページの『配布リスト』 を参照)。 MQPMO_VERSION_3 を 入力すると、メッセージ・ハンドルおよびメッセージ・プロパティーを使用できます。 MQPMO_CURRENT_VERSION を入力すると、アプリケーションが必ず最新のレベルを使用するように設定されます。

オプション
次のことを制御します。

- 書き込み操作を作業単位に含めるかどうか
- メッセージに関連付けるコンテキスト情報の量
- どこからコンテキスト情報を取り出すか
- キュー・マネージャーが静止状態のとき、呼び出しをエラーにするかどうか
- グループ化またはセグメント化が可能かどうか
- 新規のメッセージ ID と相関 ID の生成
- メッセージとセグメントをキューに書き込む順序
- ローカル・キュー名を解決するかどうか

Options フィールドをデフォルト値 (MQPMO_NONE) のままにしておくと、書き込むメッセージのコンテキスト情報がそれに関連付けられているデフォルトの情報になります。

また、呼び出しと同期点の関係は、次のようにプラットフォームによって決まります。 z/OS での同期点制御のデフォルト値は yes ですが、それ以外のプラットフォームでは no です。

Context
コンテキスト情報のコピー元にしたいキュー・ハンドルの名前を示します (*Options* フィールドで要求した場合)。

メッセージ・コンテキストについては、38 ページの『メッセージ・コンテキスト』を参照してください。 メッセージのコンテキスト情報を制御する MQPMO 構造体の使用方法については、231 ページの『コンテキスト情報の制御』を参照してください。

ResolvedQName
メッセージを受け取るためにオープンされたキューの名前 (別名の解決後の名前) を示します。 これは出力フィールドです。

ResolvedQMgrName
ResolvedQName のキューを所有するキュー・マネージャーの名前 (別名の解決後の名前) を示します。 これは出力フィールドです。

MQPMO では、配布リストに必要なフィールドも提供できます (234 ページの『配布リスト』を参照してください)。 この機能を使用したい場合は、バージョン 2 の MQPMO 構造体を使用します。 この構造体のフィールドを次に示します。

RecsPresent
このフィールドには、配布リスト内のキューの数、つまり存在する書き込みメッセージ・レコード (MQPMR)、および対応する応答レコード (MQRR) の数が入ります。

入力する値は、MQOPEN で指定したオブジェクト・レコードの数と同じにすることができます。 ただし、値が MQOPEN 呼び出しで提供されたオブジェクト・レコードの数より小さい場合、または、書き込みメッセージ・レコードを提供しない場合は、定義されていないキューの値として、メッセージ記述子提供のデフォルト値が使われます。 また、値が、提供されたオブジェクト・レコードの数より大きい場合は、超過した書き込みメッセージ・レコードが無視されます。

次のいずれかの方法を推奨します。

- 各宛先から報告または応答を受け取りたい場合は、MQOR 構造体にある値と同じ値を入力し、*MsgId* フィールドを含む MQPMR を使用する。 これらの *MsgId* フィールドをゼロに初期化する、または MQPMO_NEW_MSG_ID を指定する。

  メッセージをキューに書き込むと、キュー・マネージャーによって作成された *MsgId* の値を MQPMR で使用できるようになる。これらを使って、各報告または応答に関連付けられている宛先を識別できる。
- 報告および応答を受け取りたくない場合は、次のいずれかの方法を行う。
  1. 即時に失敗する宛先を識別したい場合は、MQOR 構造体にある値と同じ値を *RecsPresent* フィールドに入力し、これらの宛先を識別するために MQRR を指定しても構わない。 MQPMR は指定しない。
  2. 失敗する宛先を識別したくない場合は、*RecsPresent* フィールドにゼロを入力し、MQPMR も MQRR も指定しない。

注 : MQPUT1 を使用している場合は、応答レコード・ポインターと応答レコード・オフセットの数がゼロでなければなりません。

書き込みメッセージ・レコード (MQPMR) と応答レコード (MQRR) の詳細については、MQPMR および MQRR を参照してください。

PutMsgRecFields
各書き込みメッセージ・レコード (MQPMR) に存在するレコードを示します。 これらのフィールドのリストについては、238 ページの『MQPMR 構造体の使用』を参照してください。

PutMsgRecOffset* と *PutMsgRecPtr
ポインター (一般に C で使用される) とオフセット (一般に COBOL で使用される) は、書き込みメッセージ・レコードのアドレッシングを行うために使用されます (MQPMR 構造体の概要については、238 ページの『MQPMR 構造体の使用』を参照してください)。

PutMsgRecPtr フィールドを使って最初の書き込みメッセージ・レコードを指し示すポインターを指定するか、または *PutMsgRecOffset* フィールドを使って最初の書き込みメッセージ・レコードのオフセットを指定します。 これは、MQPMO の先頭からのオフセットです。 *PutMsgRecFields* フィールドに従って、*PutMsgRecOffset* または *PutMsgRecPtr* にヌル以外の値を入力します。

ResponseRecOffset および ResponseRecPtr
ポインターとオフセットは、応答レコードのアドレッシングにも使用します (応答レコードの詳細については、237 ページの『MQRR 構造体の使用』を参照してください)。

ResponseRecPtr フィールドを使って最初の応答レコードを指し示すポインターを指定するか、または *ResponseRecOffset* フィールドを使って最初の応答レコードのオフセットを指定します。 これは、MQPMO 構造体の先頭からのオフセットです。 *ResponseRecOffset* または *ResponseRecPtr* に、ヌル以外の値を設定します。

注 : 配布リストにメッセージを書き込むために MQPUT1 を使用する場合は、*ResponseRecPtr* をヌルまたはゼロにし、*ResponseRecOffset* をゼロにする必要があります。

バージョン 3 の MQPMO 構造体には、以下のフィールドが追加で含まれて います。

OriginalMsgHandle
このフィールドをどのように使用できるのかは、*Action* フィールドの値に依存します。 新規メッセージを、関連付けられたメッセージ・プロパティーと共に書き込もうとしている場合は、 前に作成して、プロパティーを設定したメッセージ・ハンドルを、このフィールドに設定し ます。 前に取り出したメッセージに応えて、転送、応答、報告書生成を 行おうとしている場合は、このフィールドにはそのメッセージのメッセージ・ハンドル が含まれます。

NewMsgHandle
NewMsgHandle が指定されている場合、 そのハンドルに関連付けられたプロパティーが、*OriginalMsgHandle* に関連付けられたプロパティーをオーバーライドします。 詳しくは、Action (MQLONG) を参照してください。

Action
このフィールドを使用して、実行される書き込みのタイプを指定します。 指定できる値と その意味は、次のとおりです。

MQACTP_NEW
これは他のどのメッセージにも関連していない新規メッセージである。

MQACTP_FORWARD
このメッセージは前に取り出され、今は転送されようとしている。

MQACTP_REPLY
このメッセージは、前に取り出されたメッセージへの応答である。

MQACTP_REPORT
このメッセージは、前に取り出されたメッセージの結果として生成 された報告書である。

詳しくは、Action (MQLONG) を参照してください。

PubLevel
このメッセージがパブリケーションである場合、それをどのサブスクリプションが受け取るのかを決定するために、 このフィールドを設定できます。 *SubLevel* がこの値 以下のサブスクリプションのみが、このパブリケーションを受け取ります。 デフォルト値 は 9 で、これは最高レベルであり、どの *SubLevel* のサブスクリプションでもこのパブリケーションを受け取ることができることを意味します。

メッセージ内のデータ

メッセージが入っているバッファーのアドレスを、MQPUT 呼び出しの *Buffer* パラメーターに指定します。 メッセージ内のデータとしては、何を入れても差し支えありません。 しかし、メッセージ内のデータ量は、これらを処理するアプリケーションのパフォーマンスに影響します。

データの最大サイズは、次の事項によって決定されます。

- キュー・マネージャーの *MaxMsgLength* 属性
- メッセージを書き込むキューの *MaxMsgLength* 属性
- WebSphere MQ によって追加される任意のメッセージ・ヘッダーのサイズ (送達不能ヘッダーの MQDLH と配布リスト・ヘッダーの MQDH を含む)

キュー・マネージャーの *MaxMsgLength* 属性は、キュー・マネージャーが処理できるメッセージのサイズを保持します。 V6 以降のすべての WebSphere MQ 製品において、この属性のデフォルトは 100 MB です。

この属性の値を判別するために、キュー・マネージャー・オブジェクトに MQINQ 呼び出しを使用します。大きなメッセージの場合は、この値を変更できます。

キューの *MaxMsgLength* 属性によって、キューに書き込むことができるメッセージの最大サイズが決まります。 この属性の値より大きいサイズのメッセージを書き込もうとすると、その MQPUT 呼び出しは失敗します。 リモート・キューにメッセージを書き込む場合、正常に書き込むことができるメッセージの最大サイズは、関係するキューおよびチャネルの *MaxMsgLength* 属性によって決まります。関係するキューとは、宛先のリモート・キューと経路上でメッセージが書き込まれる中間伝送キューを指します。

MQPUT 操作の場合は、メッセージのサイズが、キューとキュー・マネージャーの両方の *MaxMsgLength* 属性の値以下でなければなりません。 これらの属性の値は互いに独立していますが、キューの *MaxMsgLength* の値は、キュー・マネージャーの値以下にしておくことをお勧めします。

WebSphere MQ では、次の場合にメッセージにヘッダー情報を追加します。

- メッセージをリモート・キューに書き込むと、WebSphere MQ は伝送ヘッダー (MQXQH) 構造体をメッセージに追加します。 この構造体は、宛先キューとそれを所有するキュー・マネージャーの名前を含んでいます。
- WebSphere MQ がリモート・キューにメッセージを送達できない場合、メッセージを送達不能 (未配布メッセージ) キューに書き込もうとします。 そして、メッセージに MQDLH 構造体を追加します。 この構造体には、宛先キューの名前と、そのメッセージが送達不能キューに書き込まれた理由が収められています。
- メッセージを複数の宛先キューに送信する場合は、WebSphere MQ が MQDH ヘッダーをメッセージに追加します。 このヘッダーは、伝送キュー上の、配布リストに属するメッセージ内に存在するデータを記述します。 最大メッセージ長に最適値を選択するときには、この点を考慮してください。
- メッセージがセグメント化されている場合や、メッセージがグループに入っている場合、WebSphere MQ は MQMDE を追加することがあります。

これらの構造体の説明については、MQDH および MQMDE を参照してください。

メッセージ長の合計が、これらのキューに対して設定されている最大サイズに達しているときに、これらのヘッダーが追加されると、メッセージが大きくなりすぎて書き込み操作が失敗します。 書き込み操作が失敗する可能性を減らすには、次のようにしてください。

- メッセージ長を、伝送キューと送達不能キューの *MaxMsgLength* 属性より短くします。 最低でも MQ_MSG_HEADER_LENGTH 定数の値にします (大きな配布リストの場合は、さらに大きくします)。
- 送達不能キューの *MaxMsgLength* 属性を必ず、送達不能キューを所有するキュー・マネージャーの *MaxMsgLength* と同じに設定します。

キュー・マネージャーの属性とメッセージ・キューイングの定数については、キュー・マネージャーの属性を参照してください。

メッセージの書き込み: メッセージ・ハンドルの使用

MQPMO 構造体では、*OriginalMsgHandle* と *NewMsgHandle* という 2 つのメッセージ・ハンドルを使用できます。 それらのメッセージ・ハンドルの関係は、MQPMO の *Action* フィールドの値によって定義します。

詳しくは、Action (MQLONG) を参照してください。 メッセージを書き込むためにメッセージ・ハンドルが絶対に必要というわけではありません。 その目的は、メッセージにプロパティーを関連付けることなので、メッセージ・ハンドルが必要になるのは、メッセージ・プロパティーを使用する場合に限られます。

リモート・キューへのメッセージの書き込み

メッセージを、ローカル・キューではなく、リモート・キュー (つまり、アプリケーションの接続先のキュー・マネージャー以外のキュー・マネージャーが所有するキュー) に書き込む場合は、キューをオープンする時のキューの名前の指定方法を検討する必要があります。これについては、223 ページの『リモート・キューのオープン』で説明されています。 ローカル・キューに対する MQPUT または MQPUT1 呼び出しの使用方法には変更はありません。

リモート・キューおよび伝送キューの使用について詳しくは、WebSphere MQ 分散メッセージング技法 を参照してください。

メッセージ・プロパティーの設定

設定したい各プロパティーごとに MQSETMP を呼び出します。 メッセージを 書き込むときに、MQPMO 構造体のメッセージ・ハンドルおよびアクションの各フィールドを設定します。

メッセージにプロパティーを関連付けるには、メッセージにメッセージ・ハンドル がなければなりません。 メッセージ・ハンドルの作成は MQCRTMH 機能呼び出しを使用して行います。 設定したい各プロパティー

ごとにこのメッセージ・ハンドルを指定して MQSETMP を 呼び出します。 MQSETMP の使用法を示すためのサンプル・プログラム amqsstma.c が用意されています。

これが新規メッセージである場合、MQPUT または MQPUT1 を使用してキューに書き込むときに、MQPMO 内の OriginalMsgHandle フィールドをこのメッセージ・ハンドルの値に設定し、 MQPMO 内の Action フィールドを MQACTP_NEW (これは デフォルト値です) に設定します。

これが前に取り出したことのあるメッセージであり、転送または応答しようとしているか、それに応えて報告書を送信しようとしている 場合、元のメッセージ・ハンドルを MQPMO の OriginalMsgHandle フィールドに、 新規メッセージ・ハンドルを NewMsgHandle フィールドに入れます。 Action フィールドを、MQACTP_FORWARD、 MQACTP_REPLY、または MQACTP_REPORT のうち適当なものに設定します。

前に取り出したメッセージからの MQRFH2 ヘッダー内にプロパティーがある場合、MQBUFMH 呼び出しを使用して、それらのプロパティーをメッセージ・ハンドル・プロパティーに変換 することができます。

メッセージ・プロパティーを処理できない、WebSphere MQ バージョン 7.0 より前のレベルのキュー・マネージャーのキューに メッセージを書き込む場合、プロパティーがどのように処理されるのかを指定するため、チャネル定義 に PropertyControl パラメーターを設定することができます。

コンテキスト情報の制御

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

コンテキスト情報を制御するには、MQPMO 構造体の *Options* フィールドを使用します。

このフィールドを使用しない場合、キュー・マネージャーは、メッセージ記述子の中に既に存在しているコンテキスト情報を、そのメッセージ用に生成された識別コンテキスト情報で上書きします。 このフィールドの使用は、MQPMO_DEFAULT_CONTEXT オプションを指定するのと同じです。新しいメッセージを生成するときに、このデフォルト・コンテキスト情報が必要となることがあります (例えば、照会画面からのユーザー入力を処理するとき)。

メッセージに関連するコンテキスト情報が不要であれば、MQPMO_NO_CONTEXT オプションを使用してください。 メッセージをコンテキストなしで送る場合、IBM WebSphere MQ による権限検査は、ブランクのユーザー ID を使用して行われます。 ブランクのユーザー ID には、IBM WebSphere MQ リソースに対する明示的な権限を割り当てることはできませんが、特別な 'nobody' グループのメンバーとして扱われます。特殊グループ nobody について詳しくは、 インストール可能サービス・インターフェースの参照情報を参照してください。

メッセージに関連するコンテキスト情報が不要であれば、MQPMO_NO_CONTEXT オプションを使用してください。

このトピックの続くセクションに、識別コンテキスト、ユーザー・コンテキスト、すべてのコンテキストの使用についての説明があります

- 231 ページの『識別コンテキストを渡す方法』
- 232 ページの『ユーザー・コンテキストを渡す方法』
- 232 ページの『すべてのコンテキストを渡す方法』
- 232 ページの『識別コンテキストを設定する方法』
- 233 ページの『ユーザー・コンテキストの設定』
- 233 ページの『すべてのコンテキストを設定する方法』

識別コンテキストを渡す方法

一般に、プログラムは、データが最終宛先に到着するまで、アプリケーション内でメッセージからメッセージに識別コンテキスト情報を渡さなければなりません。

プログラムは、データを変更するたびに起点コンテキスト情報を変更しなければなりません。 しかし、コンテキスト情報を変更または設定しようとするアプリケーションは、適切なレベルの許可を持っていなければなりません。 アプリケーションがキューをオープンするとき、キュー・マネージャーはこの許可を検

査します。アプリケーションは、MQOPEN 呼び出しに対する適切なコンテキスト・オプションを用いる許可を持っていなければなりません。

アプリケーションがメッセージを入手し、このメッセージからのデータを処理し、(例えば別のアプリケーションに処理させるために) 変更済みデータを別のメッセージに入れる場合、アプリケーションは元のメッセージから新しいメッセージに識別コンテキスト情報を渡さなければなりません。 起点コンテキスト情報の作成はキュー・マネージャーに任せることができます。

元のメッセージのコンテキスト情報を保管するには、メッセージを読み取るキューをオープンするときに、MQOO_SAVE_ALL_CONTEXT オプションを使用してください。 これは、MQOPEN 呼び出しで使用する他のオプションに加えて使用します。 しかし、メッセージをブラウズするだけの場合は、コンテキスト情報を保管できないことに注意してください。

2 番目のメッセージを作成するときは、以下のようにします。

- (MQOO_OUTPUT オプションに加えて)、MQOO_PASS_IDENTITY_CONTEXT オプションを使ってキューをオープンする。
- 書き込みメッセージ・オプション構造体の *Context* フィールドに、コンテキスト情報の保管元のキューのハンドルを指定する。
- 書き込みメッセージ・オプション構造体の *Options* フィールドに、MQPMO_PASS_IDENTITY_CONTEXT オプションを指定する。

ユーザー・コンテキストを渡す方法

ユーザー・コンテキストのみを渡すよう選択することはできません。 メッセージを書き込むときにユーザー・コンテキストを 渡すには、MQPMO_PASS_ALL_CONTEXT を指定します。 ユーザー・コンテキスト内の すべてのプロパティーが、起点コンテキストと同じようにして渡されます。

MQPUT または MQPUT1 が実行されてコンテキストが引き渡されると、ユーザー・コンテキスト内の全プロパティーは、取得されたメッセージから書き込みメッセージに引き渡されます。 書き込みアプリケーションによって変更されたユーザー・コンテキスト・プロパティーは、すべて元の値で書き込まれます。 書き込みアプリケーションによって削除されたユーザー・コンテキスト・プロパティーは、すべて書き込みメッセージ内で復元されます。 書き込みアプリケーションによってメッセージに追加されたユーザー・コンテキスト・プロパティーは、すべて保存されます。

すべてのコンテキストを渡す方法

アプリケーションがメッセージを入手し、そのメッセージ・データを (変更せずに) 別のメッセージに入れる場合、アプリケーションは元のメッセージから新しいメッセージに、すべてのコンテキスト情報 (識別コンテキスト情報、起点コンテキスト情報、およびユーザー・コンテキスト情報) を渡さなければなりません。 これを行うアプリケーションの例としては、メッセージを 1 つのキューから別のキューへ移動するメッセージ・ムーバーがあります。

MQOPEN オプションの MQOO_PASS_ALL_CONTEXT と、書き込みメッセージ・オプションの MQPMO_PASS_ALL_CONTEXT を使用する以外は、識別コンテキストを渡す手順と同じ手順に従ってください。

識別コンテキストを設定する方法

メッセージの識別コンテキスト情報を設定したい場合は、次のようにします。

- MQOO_SET_IDENTITY_CONTEXT オプションを用いて、キューをオープンする。
- MQPMO_SET_IDENTITY_CONTEXT オプションを指定して、メッセージをキューに書き込む。 メッセージ記述子に、必要な識別コンテキスト情報をすべて指定する。

注 : MQOO_SET_IDENTITY_CONTEXT および MQPMO_SET_IDENTITY_CONTEXT オプションを使用して、いくつか (全部ではない) の識別コンテキスト・フィールドを設定する場合、キュー・マネージャーが他のいずれのフィールドも設定しないということを意識する必要があります。

メッセージ・コンテキスト・オプションのいずれかを変更するためには、呼び出しを発行するための適切な許可が必要です。 例えば、MQOO_SET_IDENTITY_CONTEXT または MQPMO_SET_IDENTITY_CONTEXT を使用するには、+setid アクセス権が必要です。

ユーザー・コンテキストの設定

ユーザー・コンテキスト内にプロパティーを設定するには、MQSETMP 呼び出しを行うときに メッセージ・プロパティー記述子 (MQPD) の Context フィールドを MQPD_USER_CONTEXT に設定します。

ユーザー・コンテキストにプロパティーを設定するために、特殊権限は必要ではありません。 ユーザー・コンテキストには、MQOO_SET_*または MQPMO_SET_* コンテキスト・オプションはありません。

すべてのコンテキストを設定する方法

メッセージの識別コンテキスト情報と起点コンテキスト情報の両方を設定したい場合は、次のようにします。

1. MQOO_SET_ALL_CONTEXT オプションを使用して、キューをオープンする。
2. MQPMO_SET_ALL_CONTEXT オプションを指定して、メッセージをキューに書き込む。 メッセージ記述子に、必要な識別コンテキスト情報と起点コンテキスト情報をすべて指定する。

各種のコンテキストの設定には、適切な許可が必要です。

関連概念

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

関連資料

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

MQPUT1 呼び出しを使用したキューへの単一メッセージの書き込み

キューにメッセージを 1 つ書き込んだ直後にキューをクローズするときは、MQPUT1 呼び出しを使用します。 例えば、サーバーのアプリケーションは、異なる各キューに応答を送信するとき、MQPUT1 呼び出しを用いることがあります。

MQPUT1 の機能は、MQOPEN、MQPUT、MQCLOSE を順番に呼び出した場合と同じです。 MQPUT 呼び出しと MQPUT1 呼び出しの構文上の唯一の違いは、MQPUT ではオブジェクト・ハンドルを指定するのに対して、MQPUT1 ではオブジェクト記述子構造体 (MQOD) を MQOPEN に定義されているように指定することです (216 ページの『オブジェクトの識別 (MQOD 構造体)』 を参照してください)。 これは、MQPUT1 を呼び出す時にはオープンするキューに関する情報を与えなければならないのに対して、MQPUT を呼び出すときはキューが既にオープンしていなければならないからです。

MQPUT1 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。
- オープンしたいオブジェクトの記述。 これは、オブジェクト記述子構造体 (MQOD) の形式です。
- キューに書き込みたいメッセージについての記述。 これは、メッセージ記述子構造体 (MQMD) の形式になります。
- 書き込みメッセージ・オプション構造体 (MQPMO) の形式による制御情報。
- メッセージ内に含まれるデータの長さ (MQLONG)。
- メッセージ・データのアドレス。

MQPUT1 の出力は以下のとおりです。

- 完了コード

- 理由コード

呼び出しが正常に完了した場合は、オプション構造体とメッセージ記述子構造体も戻します。 この呼び出しは、メッセージ送信先のキューおよびキュー・マネージャーの名前を示すために、オプション構造体を変更します。 (MQMD 構造体の *MsgId* フィールドに 2 進数のゼロを指定することによって)、書き込むメッセージの ID として固有な値をキュー・マネージャーに生成させる場合、この呼び出しは、この構造体を戻す前に *MsgId* フィールドにその値を入れます。

注 : モデル・キュー名で MQPUT1 を使用することはできません。しかし、いったんモデル・キューがオープンされれば、MQPUT1 を動的キューに対して発行できます。

MQPUT1 に対する入力パラメーターは次の 6 つです。

Hconn
これは接続ハンドルです。 CICS アプリケーションの場合は、(ゼロの値を持つ) 定数 MQHC_DEF_HCONN を指定するか、または MQCONN か MQCONNX 呼び出しによって戻される接続ハンドルを使用します。 他のプログラムの場合は、常に MQCONN 呼び出しか MQCONNX 呼び出しによって戻される接続ハンドルを用います。

ObjDesc
これはオブジェクト記述子構造体 (MQOD) です。

ObjectName および *ObjectQMgrName* フィールドには、メッセージを書き込みたいキューの名前と、このキューを所有するキュー・マネージャーの名前をそれぞれ指定します。

モデル・キューは使用できないため、MQPUT1 呼び出しに対する *DynamicQName* フィールドは無視されます。

AlternateUserId フィールドは、キューをオープンするための許可の検査に使う代替ユーザー ID を指定したい場合に使用します。

MsgDesc
これは、メッセージ記述子構造体 (MQMD) です。 MQPUT 呼び出しと同様に、この構造体を使用して、キューに書き込むメッセージを定義します。

PutMsgOpts
これは、書き込みメッセージ・オプション構造体 (MQPMO) です。 MQPUT 呼び出しのときと同様に、この構造体を使用してください (226 ページの『MQPMO 構造体を使用するオプションの指定』を参照)。

Options フィールドがゼロに設定されていると、キュー・マネージャーは、キューへのアクセス許可を検査するときに、独自のユーザー ID を使用します。 また、キュー・マネージャーは、MQOD 構造体の *AlternateUserId* フィールドに指定されている代替ユーザー ID を無視します。

BufferLength
これはメッセージの長さです。

Buffer
これは、メッセージのテキストを含むバッファーです。

クラスターを使用すると、MQPUT1 は MQOO_BIND_NOT_FIXED が有効になっている場合と同じ動作をします。 メッセージの送信先を判別する場合、各アプリケーションは MQOD 構造体ではなく MQPMO 構造体の解決済みフィールドを使用する必要があります。 詳細については、キュー・マネージャー・クラスターの構成を参照してください。

MQPUT1 呼び出しの説明については、MQPUT1 を参照してください。

配布リスト

WebSphere MQ for z/OS ではサポートされない。 配布リストを使用すると、単一の MQPUT または MQPUT1 呼び出しでメッセージを複数の宛先に書き込むことができます。単一の MQOPEN 呼び出しで複数のキューを開いた後、 単一の MQPUT 呼び出しでそれらの各キューにメッセージを書き込むことができます。 このプロセスで使用される MQI 構造体からの汎用情報の一部は、宛先リストに含まれている個々の宛先に関係する特定の情報に置き換えることができます。

V 7.5.0.8

重要 : 配布リストでは、トピック・オブジェクトを指す別名キューの使用はサポートされていません。 Version 7.5.0, Fix Pack 8 以降では、 別名キューが配布リスト内のトピック・オブジェクトを指している場合、IBM WebSphere MQ は MQRC_ALIAS_BASE_Q_TYPE_ERROR を返します。

MQOPEN 呼び出しを使用すると、オブジェクト記述子 (MQOD) から汎用情報が取り込まれます。 *Version* フィールドに MQOD_VERSION_2 を指定し、*RecsPresent* フィールドにゼロより大きい値を指定すると、*Hobj* を、キューのハンドルではなく、(1 つまたは複数のキューの) リストのハンドルとして定義できます。この場合は、オブジェクト・レコード (MQOR) から特定の情報が提供されます。オブジェクト・レコードからは、宛先の詳細 (つまり、*ObjectName* と *ObjectQMgrName*) が提供されます。

オブジェクト・ハンドル (*Hobj*) は MQPUT 呼び出しに渡されるので、単一のキューではなくリストへの書き込みが可能になります。

メッセージをキューに書き込むと (MQPUT)、書き込みメッセージ・オプション構造体 (MQPMO) とメッセージ記述子構造体 (MQMD) から汎用情報が送り込まれます。 特定の情報は、書き込みメッセージ・レコード (MQPMR) の形式で提供されます。

応答レコード (MQRR) では、各宛先キューごとの完了コードと理由コードを受け取ることができます。

235 ページの図 29 には、配布リストの機能が示されています。

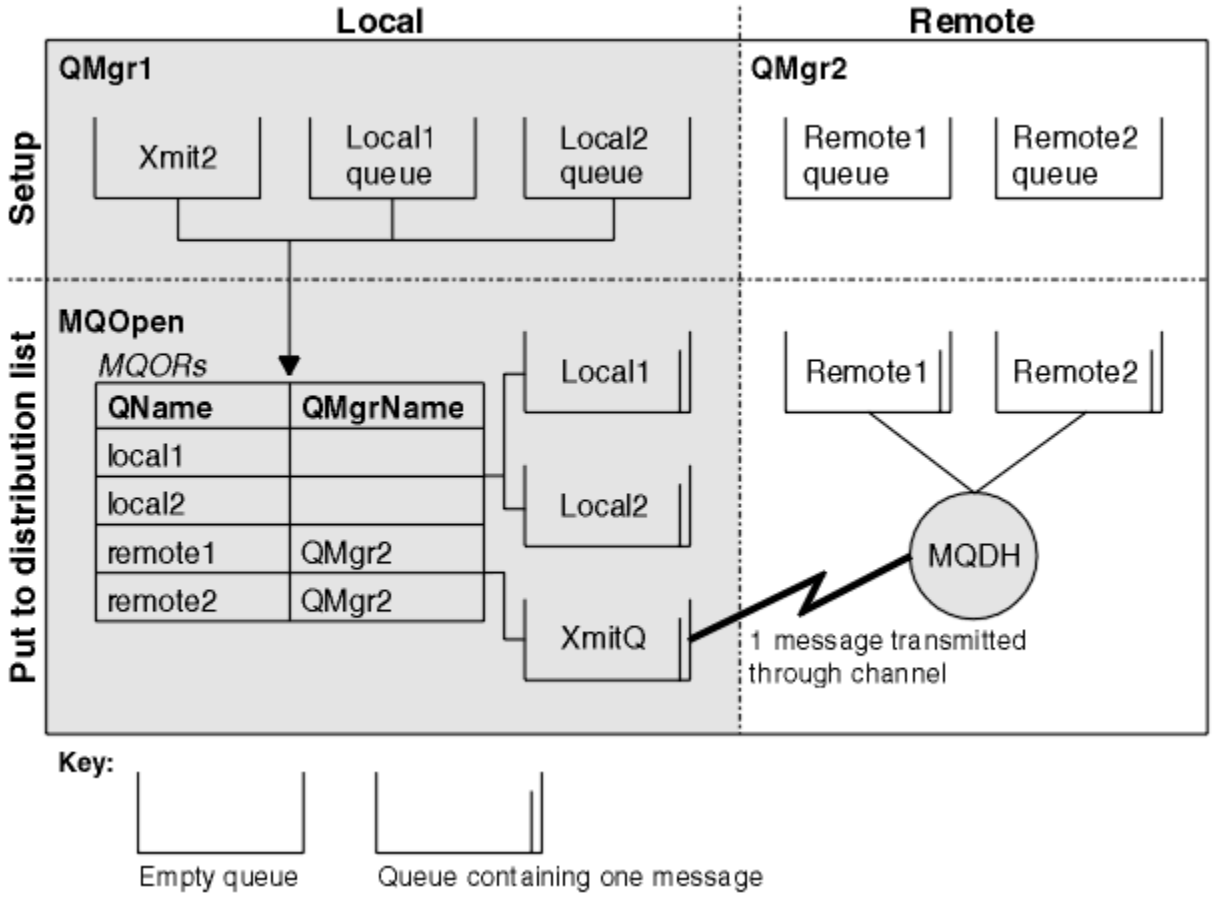


図 *29.* 配布リストの機能

配布リストのオープン

MQOPEN 呼び出しを使用して配布リストをオープンし、その呼び出しの各オプションを使用してそのリストの操作内容を指定します。

MQOPEN への入力として、次のものを提供する必要があります。

- 接続ハンドル (詳細は、224 ページの『キューへのメッセージの書き込み』 を参照)

- オブジェクト記述子構造体 (MQOD) の汎用情報
- オープンしたい各キューの名前 (オブジェクト・レコード構造体 (MQOR) を使用)

MQOPEN の出力は以下のとおりです。

- 配布リストへのアクセスを表すオブジェクト・ハンドル
- 汎用完了コード
- 汎用理由コード
- 応答レコード (オプション) (各宛先の完了コードと理由を含む)

MQOD 構造体の使用

MQOD 構造体を使用して、オープンするキューを指定します。

配布リストを定義するには、*Version* フィールドに MQOD_VERSION_2 を指定し、*RecsPresent* フィールドにゼロより大きい値を指定し、*ObjectType* フィールドに MQOT_Q を指定する必要があります。 MQOD 構造体のすべてのフィールドについては、MQOD を参照してください。

MQOR 構造体の使用

各宛先ごとに MQOR 構造体を指定してください。

この構造体には、宛先キューとキュー・マネージャーの名前を組み込みます。 MQOD の *ObjectName* および *ObjectQMgrName* フィールドは、配布リストには使用しません。 1 つ以上のオブジェクト・レコードを組み込む必要があります。 *ObjectQMgrName* をブランクのままにすると、ローカル・キュー・マネージャーが使用されます。 これらのフィールドの詳細については、ObjectName および ObjectQMgrName を参照してください。

宛先キューを指定するには、2 つの方法があります。

- オフセット・フィールド *ObjectRecOffset* を使用する。

  この場合は、アプリケーションで、MQOD 構造体を組み込んだ独自の構造体を宣言し、その後に (必要な数の配列エレメントを使用して) MQOR レコードの配列を記述し、MQOD の開始点を基準にした配列内の最初のエレメントのオフセットを *ObjectRecOffset* に設定します。 このオフセットが正しいか、確かめてください。

  アプリケーションを実行するどの環境でも、プログラミング言語に用意されている組み込み機能を使用できる場合は、その組み込み機能を使用することをお勧めします。 COBOL プログラミング言語でその技法を使用するコード例を以下に示します。

```
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE LENGTH OF MY-MQOD TO MQOD-OBJECTRECOFFSET.
```

  対象のすべての環境で必要になる組み込み機能がプログラミング言語でサポートされていない場合は、定数 MQOD_CURRENT_LENGTH を使用します。 その技法を使用するコード例を以下に示します。

```
01  MY-MQ-CONSTANTS.
    COPY CMQV.
01  MY-OPEN-DATA.
    02 MY-MQOD.
       COPY CMQODV.
    02 MY-MQOR-TABLE OCCURS 100 TIMES.
       COPY CMQORV.
    MOVE MQOD-CURRENT-LENGTH TO MQOD-OBJECTRECOFFSET.
```

  ただし、このコードが正しく動作するのは、MQOD 構造体と MQOR レコードの配列が隣接している場合に限られます。コンパイラーが MQOD と MQOR 配列の間にスキップ・バイトを挿入する場合は、*ObjectRecOffset* の格納値にそのバイト数を追加する必要があります。

ポインター・データ型をサポートしていないプログラミング言語や、別の環境に移植できない方法でポインター・データ型を実装しているプログラミング言語 (COBOL プログラミング言語など) の場合は、*ObjectRecOffset* を使用することをお勧めします。

- ポインター・フィールド *ObjectRecPtr* を使用する。

  この場合は、アプリケーションで MQOD 構造体とは別に MQOR 構造体の配列を宣言でき、*ObjectRecPtr* に配列のアドレスを設定できます。 C プログラミング言語でその技法を使用するコード例を以下に示します。

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  別の環境に移植できる方法でポインター・データ型をサポートしているプログラミング言語 (C プログラミング言語など) の場合は、*ObjectRecPtr* を使用することをお勧めします。

どの技法を選択するにしても、*ObjectRecOffset* と *ObjectRecPtr* のいずれかを使用する必要があります。どちらもゼロの場合やどちらもゼロ以外の場合は、呼び出しが理由コード MQRC_OBJECT_RECORDS_ERROR で失敗します。

MQRR 構造体の使用

これらの構造体は、宛先ごとに記述します。つまり、配布リストのキューごとに *CompCode* フィールドと *Reason* フィールドを各応答レコードに組み込みます。 問題が存在する場所を確認できるようにするために、この構造体を使用する必要があります。

例えば、配布リストに 5 つの宛先キューが含まれている場合に、この構造体を使用しなければ、理由コード MQRC_MULTIPLE_REASONS を受け取ったときに、問題がどのキューに該当するのかを確認できません。一方、宛先ごとに完了コードと理由コードを受け取れば、エラーの場所を容易に判別できます。

MQRR 構造体の詳細については、MQRR を参照してください。

C で配布リストをオープンする方法を 237 ページの図 30 に示します。

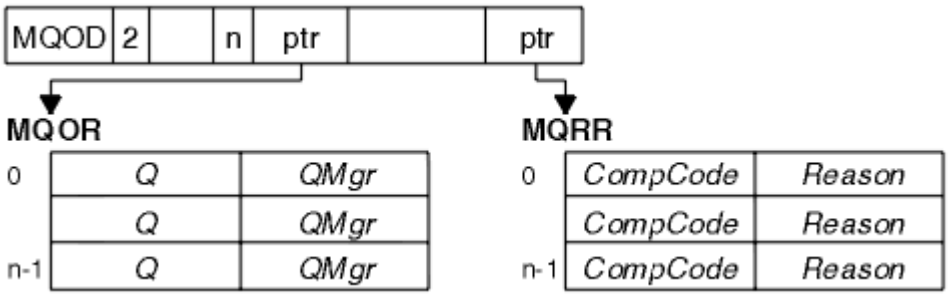


図 *30. C* による配布リストのオープン

COBOL で配布リストをオープンする方法を 237 ページの図 31 に示します。

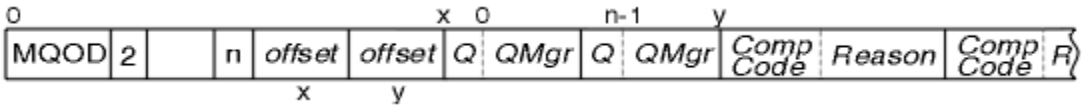


図 *31. COBOL* による配布リストのオープン

MQOPEN の各オプションの使用

配布リストをオープンするときには、以下のオプションを指定できます。

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (オプション)
- MQOO_ALTERNATE_USER_AUTHORITY (オプション)

- MQOO_*_CONTEXT (オプション)

これらのオプションの説明については、214 ページの『オブジェクトのオープンとクローズ』を参照してください。

配布リストへのメッセージの書き込み

配布リストにメッセージを書き込むには、MQPUT または MQPUT1 を使用します。

入力として、次のものを提供する必要があります。

- 接続ハンドル (詳細は、224 ページの『キューへのメッセージの書き込み』を参照)。
- オブジェクト・ハンドル。 MQOPEN を使って配布リストをオープンした場合は、*Hobj* を使用するとリストへの書き込みだけができます。
- メッセージ記述子構造体 (MQMD)。 この構造体の詳細については、MQMD を参照してください。
- 書き込みメッセージ・オプション構造体の形式による制御情報 (MQPMO)。 MQPMO 構造体のフィールドへの入力については、226 ページの『MQPMO 構造体を使用するオプションの指定』を参照してください。
- 書き込みメッセージ・レコード (MQPMR) の形式による制御情報。
- メッセージ内に含まれるデータの長さ (MQLONG)。
- メッセージ・データそのもの。

出力は以下のとおりです。

- 完了コード
- 理由コード
- 応答レコード (オプション)

MQPMR 構造体の使用

この構造体を使用するかどうかは任意です。この構造体では、MQMD で既に指定したフィールドとは別に指定したいと思っているいくつかのフィールドの宛先固有の情報を記述します。

これらのフィールドの説明については、MQPMR を参照してください。

各レコードの内容は、MQPMO の *PutMsgRecFields* フィールドで指定する情報によって異なります。 例えば、配布リストの使用法を示したサンプル・プログラム AMQSPTL0.C (説明については、126 ページの『配布リスト・サンプル・プログラム』を参照) では、MQPMR で *MsgId* と *CorrelId* の値を指定できるように記述内容が選択されています。 サンプル・プログラムのこの部分は、次のようになっています。

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

このプログラムでは、*MsgId* と *CorrelId* が配布リストの各宛先ごとに提供されます。 書き込みメッセージ・レコードは、配列として記述されています。

C で配布リストにメッセージを書き込む方法を 239 ページの図 32 に示します。

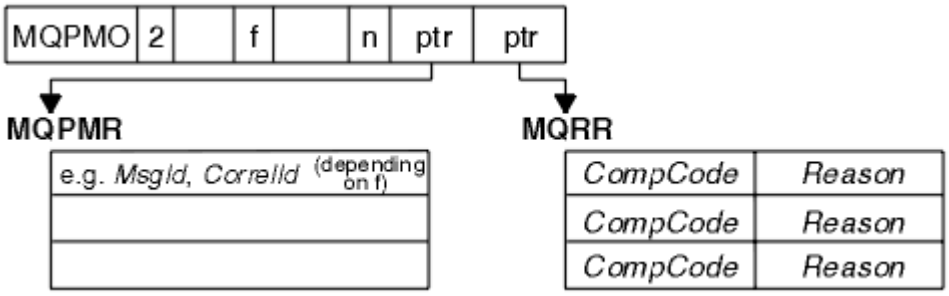


図 *32. C* による配布リストへのメッセージの書き込み

COBOL で配布リストにメッセージを書き込む方法を 239 ページの図 33 に示します。

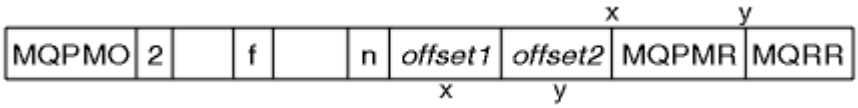


図 *33. COBOL* による配布リストへのメッセージの書き込み

MQPUT1 の使用

MQPUT1 を使用する場合の考慮点を以下に示します。

1. *ResponseRecOffset* フィールドと *ResponseRecPtr* フィールドの値は、ヌルまたはゼロでなければなりません。
2. 応答レコードが必要な場合は、MQOD から応答レコードのアドレス指定を行う必要があります。

書き込み呼び出しが失敗する場合

キューの特定の属性が、MQOPEN 呼び出しと MQPUT 呼び出しの間の期間にコマンドの FORCE オプションを使用して変更された場合、MQPUT 呼び出しは失敗し、MQRC_OBJECT_CHANGED 理由コードが戻されます。

キュー・マネージャーは、オブジェクト・ハンドルを無効としてマーク付けします。 このことは、MQPUT1 呼び出しの処理中に変更が行われたり、キュー名が解決された結果のキューにその変更が適用されたりする場合にも起こります。 このようにハンドルに影響を及ぼす属性は、MQOPEN の MQOPEN 呼び出しの節で一覧表示されています。 呼び出しが MQRC_OBJECT_CHANGED 理由レコードを戻す場合は、キューをクローズしてキューを再度オープンしてから、メッセージの書き込みを行います。

メッセージを書き込もうとしているキュー (または、キュー名が解決された結果のキュー) に対して書き込み操作が禁止されている場合は、MQPUT または MQPUT1 呼び出しは失敗し、MQRC_PUT_INHIBITED の理由コードが戻されます。 アプリケーションの設計として、他のプログラムがキューの属性を規則的に変更するようになっているのであれば、後で呼び出しをするとメッセージを正しく書き留める可能性があります。あとで呼び出しを再実行すると、書き込みに成功する場合もあります。ただし、ほかのプログラムがキューの属性を定期的に変更するようにアプリケーションが設計されていることが前提です。

さらに、メッセージを書き込もうとしているキューが満ぱいの場合は、MQPUT または MQPUT1 呼び出しは失敗し、MQRC_Q_FULL が戻されます。

(一時または永続) 動的キューが削除されていると、前に取得したオブジェクト・ハンドルを使用した MQPUT 呼び出しは失敗し、MQRC_Q_DELETED の理由コードを戻します。 この場合、オブジェクト・ハンドルは必要なくなるので、クローズすることをお勧めします。

配布リストの場合は、1 つの要求で複数の完了コードと理由コードが発生することがあります。 そのため、MQOPEN と MQPUT の *CompCode* と *Reason* の出力フィールドだけを使って配布リストを処理することはできません。

複数の宛先にメッセージを書き込むために配布リストを使用すると、応答レコードには、各宛先ごとに特定の *CompCode* と *Reason* が設定されます。 完了コード MQCC_FAILED を受け取った場合は、どの宛先キューにも正常にはメッセージが書き込まれていません。 完了コードが MQCC_WARNING の場合は、1 つ以上の宛先キューにメッセージが正常に書き込まれます。 戻りコード MQRC_MULTIPLE_REASONS を受け

取った場合は、すべての宛先で理由コードが同じであるとは限りません。 したがって、エラーの原因となった 1 つまたは複数のキューと、それぞれの理由を判別できるように、MQRR 構造体を使用することをお勧めします。

キューからのメッセージの読み取り

この情報を使用して、キューからのメッセージの読み取りについて学習します。

次の 2 通りの方法で、キューからメッセージを読み取ることができます。

1. 他のプログラムがもうメッセージを見ることができないように、キューからメッセージを除去できます。
2. 元のメッセージをキューに残して、メッセージをコピーできます。 これを、ブラウズ といいます。 一度ブラウズされたメッセージは除去できます。

どちらの場合も、MQGET 呼び出しを使用しますが、最初のアプリケーションはキュー・マネージャーに接続されていなければなりません。また、(入力用、ブラウズ用、またはその両方用の) キューをオープンするために MQOPEN 呼び出しを使用する必要があります。 これらの操作については、206 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』および 214 ページの『オブジェクトのオープンとクローズ』に記載されています。

キューをオープンすると、そのキューのメッセージを MQGET 呼び出しを繰り返し使用して、ブラウズしたり除去したりできます。 必要なメッセージをキューからすべて読み取ったら、MQCLOSE を呼び出します。

メッセージをキューから読み取る方法について詳しくは、以下のリンクを参照してください。

- 241 ページの『MQGET 呼び出しの使用によるキューからのメッセージの読み取り』
- 245 ページの『メッセージがキューから取り出される順序』
- 256 ページの『特定のメッセージの読み取り』
- 257 ページの『非永続メッセージのパフォーマンス向上』
- 262 ページの『4 MB より長いメッセージの処理』
- 267 ページの『メッセージの待機』
-
- 268 ページの『バックアウトのスキップ』
- 270 ページの『アプリケーション・データの変換』
- 271 ページの『キュー上のメッセージのブラウズ』
- 277 ページの『MQGET 呼び出しが失敗する場合』

関連概念

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

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

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

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

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

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

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

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

MQGET 呼び出しの使用によるキューからのメッセージの読み取り

MQGET 呼び出しを使用すると、オープン・ローカル・キューのメッセージを読み取ることができます。 別のシステム上のキューのメッセージを読み取ることはできません。

MQGET 呼び出しへの入力として、次のものを提供する必要があります。

- 接続ハンドル。
- キュー・ハンドル。
- キューから読み取りたいメッセージの記述。 これは、メッセージ記述子 (MQMD) 構造体の形式です。
- 読み取りメッセージ・オプション (MQGMO) 構造体の形式による制御情報。
- メッセージを保持するため割り当てたバッファーのサイズ (MQLONG)。
- メッセージを書き込むストレージのアドレス。

MQGET の出力は以下のとおりです。

- 理由コード
- 完了コード
- 呼び出しが正常に完了した場合の、指定したバッファー域のメッセージ
- メッセージが検索されたキューの名前を示すように修正された、オプション構造体
- 検索されたメッセージを記述するためにフィールドの内容が修正された、メッセージ記述子構造体
- メッセージ長 (MQLONG)

MQGET 呼び出しの説明については、MQGET を参照してください。

以下の節では、MQGET 呼び出しへの入力として指定しなければならない情報を記述します。

- 241 ページの『接続ハンドルの指定』
- 241 ページの『MQMD 構造体および MQGET 呼び出しの使用によるメッセージの記述』
- 242 ページの『MQGMO 構造体を使用する MQGET オプションの指定』
- 244 ページの『バッファー域のサイズの指定』

接続ハンドルの指定

z/OS アプリケーション上の CICS の場合、定数 MQHC_DEF_HCONN (値ゼロ) を指定するか、MQCONN 呼び出しまたは MQCONNX 呼び出しによって戻される接続ハンドルを使用することができます。 その他のアプリケーションの場合は、常に MQCONN 呼び出しまたは MQCONNX 呼び出しから返される接続ハンドルを使用します。

キュー・ハンドル (*Hobj*) は、MQOPEN を呼び出したときに戻されたものを使用します。

MQMD 構造体および MQGET 呼び出しの使用によるメッセージの記述

キューから読み取ろうとするメッセージを識別するには、メッセージ記述子構造体 (MQMD) を用いてください。

これは、MQGET 呼び出し用の入出力パラメーターです。 MQMD で記述するメッセージ・プロパティーの概要については、9 ページの『IBM WebSphere MQ メッセージ』を参照してください。その構造体そのものの説明については、MQMD を参照してください。

キューから読み取りたいメッセージが分かっている場合は、256 ページの『特定のメッセージの読み取り』を参照してください。

特定のメッセージを指定しなければ、MQGET はキュー内の最初の メッセージを検索します。 245 ページの『メッセージがキューから取り出される順序』では、メッセージの優先順位、つまり、キューの *MsgDeliverySequence* 属性および MQGMO_LOGICAL_ORDER オプションによって、キューの中でメッセージ順序がどのように決められるかを説明しています。

注 : (例えば、キュー内のメッセージを順次に取り出すために) MQGET を複数回使用したい場合は、各呼び出しのあとに、この構造体の *MsgId* および *CorrelId* フィールドをヌルに設定しなければなりません。これにより、検索されたメッセージの ID のフィールドが消去されます。

ただし、メッセージをグループ化したい場合には、同一のグループのメッセージについては同じ *GroupId* 値を使用する必要があります。これにより、MQGET 呼び出しで前のメッセージと同じ ID をもつメッセージが検索され、あるグループに属するすべてのメッセージを取り出すことができます。

MQGMO 構造体を使用する MQGET オプションの指定

MQGMO 構造体は、オプションを MQGET 呼び出しに渡すための入出力変数です。 以下の節を参考にして、この構造体の フィールドに入力してください。

MQGMO 構造体の説明については、MQGMO を参照してください。

StrucId
StrucId は、構造体を読み取りメッセージ・オプション構造体として識別するために使用する 4 文字のフィールドです。 MQGMO_STRUC_ID の指定は必須です。

Version
Version は、構造体のバージョン番号を記述します。 MQGMO_VERSION_1 がデフォルト値です。「バージョン 2」フィールドを使用するか論理順序でメッセージを取得する場合は、MQGMO_VERSION_2 を指定します。「バージョン 3」フィールドを使用するか論理順序でメッセージを取得する場合は、MQGMO_VERSION_3 を指定します。 MQGMO_CURRENT_VERSION は、最新のレベルを使用するようにアプリケーションを設定します。

Options
コード内では、任意の順序でオプションを選択できます。各オプションは、*Options* フィールド内のビットによって表されます。

Options フィールドは、次の事柄を制御します。

- MQGET 呼び出しが、完了前にメッセージがキューに到着するのを待機するかどうか (267 ページの『メッセージの待機』を参照)
- 読み取り操作が作業単位に含まれるかどうか
- 非持続メッセージを同期点以外で取り出すことによって、高速メッセージングを可能にするかどうか
- WebSphere MQ for z/OS で、検索されたメッセージが、バックアウトをスキップするものとしてマーク付けされるかどうか (268 ページの『バックアウトのスキップ』を参照)
- メッセージがキューから除去されるのか、単にブラウズされるのか
- メッセージの選択に、ブラウズ・カーソルを使用するか、それとも他の選択基準を使用するか
- メッセージ長がバッファーより長い場合でも、呼び出しが成功するかどうか
- WebSphere MQ for z/OS で、呼び出しを完了させるかどうか。 このオプションでは、メッセージ到着を通知する信号を設定するかどうかも設定します。
- キュー・マネージャーが静止状態のとき、呼び出しをエラーにするかどうか
- WebSphere MQ for z/OS で、接続が静止状態のときに呼び出しを失敗させるかどうか
- アプリケーションのメッセージ・データの変換が必要かどうか (270 ページの『アプリケーション・データの変換』を参照)
- キューからメッセージとセグメント (WebSphere MQ for z/OS を除く) を取り出す順序
- WebSphere MQ for z/OS を除き、完全な論理メッセージのみが取り出し可能であるかどうか
- あるグループ内のすべての メッセージが使用可能な場合のみ、そのグループ内のメッセージを取り出せるようにするかどうか

- WebSphere MQ for z/OS を除き、ある論理メッセージ内のすべて のセグメントが使用可能な場合にのみ、その論理メッセージ内のセグメントを取り出せるようにするかどうか

Options フィールドをデフォルト値 (MQGMO_NO_WAIT) にしておくと、MQGET 呼び出しは次のように機能します。

- 選択基準に合致するメッセージがキューにない場合、呼び出しはメッセージの到着を待たずに、即時完了します。 また、WebSphere MQ for z/OS での呼び出しは、このようなメッセージが到着したときの通知を要求する信号を設定しません。
- 呼び出しと同期点の関係は、プラットフォームによって次のように設定されています。

プラットフォーム	同期点の制御下にあるか
IBM i	No
UNIX and Linux システム	No
z/OS	Yes
Windows システム	No

- WebSphere MQ for z/OS では、検索されたメッセージはバックアウトをスキップするものとしてマーク付けされません。
- 選択されたメッセージはキュー除去されます (ブラウズされない)。
- アプリケーション・メッセージのデータ変換は、必要ありません。
- メッセージがバッファーより長い場合は、呼び出しが失敗します。

WaitInterval
WaitInterval フィールドは、MQGMO_WAIT オプションを使用した場合に、MQGET 呼び出しがキューにメッセージが到着するのを待機する最大時間 (ミリ秒単位) を指定します。 *WaitInterval* に指定した時間内にメッセージが到着しない場合は、呼び出しが完了し、選択基準に合致するメッセージがキュー上になかったことを示す理由コードを戻します。

WebSphere MQ for z/OS で、MQGMO_SET_SIGNAL オプションを使用した場合は、この *WaitInterval* フィールドに、信号が設定される時間を指定します。

これらのオプションの詳細については、267 ページの『メッセージの待機』 を参照してください。

Signal1
Signal1 は、WebSphere MQ for z/OS および MQSeries for HP Integrity NonStop Server でのみサポートされています。

MQGMO_SET_SIGNAL オプションを使用して、適切なメッセージが到着したときにアプリケーションに通知するように要求する場合は、*Signal1* フィールドにシグナルのタイプを指定します。 上記以外のすべてのプラットフォームの WebSphere MQ では、*Signal1* フィールドは予約フィールドとなるので、その値は有効ではありません。

Signal2
Signal2 フィールドはすべてのプラットフォームで予約されており、その値は有効ではありません。

ResolvedQName
ResolvedQName は、キュー・マネージャーが、メッセージを検索したキューの名前 (別名を解決したあとの) を戻すための出力フィールドです。

MatchOptions
MatchOptions は、MQGET の選択基準を制御します。

GroupStatus
GroupStatus は、取り出したメッセージがグループに属しているかどうかを示します。

SegmentStatus
SegmentStatus は、取り出した項目が論理メッセージのセグメントかどうかを示します。

Segmentation
Segmentation は、取り出されたメッセージに対してセグメント化を実行できるかを示します。

MsgToken

MsgToken は、メッセージを一意に識別します。

ReturnedLength
ReturnedLength は、キュー・マネージャーが、戻されたメッセージ・データの長さ (バイト単位) を戻すための出力フィールドです。

MsgHandle
キューから取り出されるメッセージの プロパティーが設定される、メッセージのハンドル。 このハンドルは、 前に MQCRTMH 呼び出しで作成されたものです。 ハンドルに既に関連付けられているプロパティーは、メッセージを取り出す前にすべて クリアされます。

バッファー域のサイズの指定

MQGET 呼び出しの *BufferLength* パラメーターには、検索したメッセージ・データを保持するバッファー域のサイズを指定します。 この大きさを決める方法には、以下の 3 通りがあります。

1. このプログラムから出されるメッセージの長さが既に分かっている場合があります。 その場合は、そのサイズのバッファーを指定してください。

   しかし、メッセージがバッファーより長い場合でも、MQGET 呼び出しを完了させるには、MQGMO 構造体に MQGMO_ACCEPT_TRUNCATED_MSG オプションを使用できます。 その場合は、次のようにします。

   - バッファーには、保持できる分だけのメッセージが入る。
   - 呼び出しは警告の完了コードを戻す。
   - メッセージがキューから除去される (メッセージの残りの部分は切り捨てられる)、またはブラウズ・カーソルが次に進められる (キューをブラウズする場合)。
   - メッセージの実際の長さは *DataLength* に戻される。

   このオプションを指定しなくても、呼び出しはやはり警告を出して完了しますが、メッセージはキューから除去されません (またはブラウズ・カーソルは進みません)。
2. バッファー・サイズを見積もってください (場合によっては、ゼロ・バイトのサイズを指定することもできます)。MQGMO_ACCEPT_TRUNCATED_MSG オプションは使用しないでください。 MQGET 呼び出しが失敗した場合は (例えば、バッファーが小さすぎるために)、呼び出しの *DataLength* パラメーターにメッセージの長さが戻されます (それでも、バッファーには、収容しきれるだけのメッセージを保持していますが、呼び出しの処理は完了しません。) このメッセージの *MsgId* を記録しておき、あとで適切なサイズのバッファー域と最初の呼び出しで記録した *MsgId* を指定して、MQGET をもう一度呼び出してください。

   プログラムが、他のプログラムも使用しているキューを処理する場合は、別の MQGET 呼び出しを出す前に、他のプログラムの 1 つが必要なメッセージを除去してしまうこともあります。 その結果、このメッセージを探しているプログラムは、もはや存在しないメッセージの検索に時間を浪費することになります。 このことを避けるために、*BufferLength* をゼロに指定して MQGMO_ACCEPT_TRUNCATED_MSG オプションを使用し、必要なメッセージを検出するまでまずキューをブラウズします。 これにより、必要とするメッセージの下にブラウズ・カーソルが位置付けられます。 次に、MQGMO_MSG_UNDER_CURSOR オプションを指定して MQGET を再度呼び出すと、そのメッセージを取り出すことができます。 このブラウズ呼び出しと除去呼び出しの間に別のプログラムがそのメッセージを除去した場合は、ブラウズ・カーソルの下にメッセージがないので、2 回目の MQGET は即時に (キュー全体を検索しないで) 失敗します。
3. キューで受け入れられるメッセージの最大長は、そのキューの *MaxMsgLength* キュー 属性によって決定され、キュー・マネージャーで受け入れられるメッセージの最大長は、そのキュー・マネージャーの *MaxMsgLength* キュー・マネージャー 属性によって決定されます。 どんな長さのメッセージ受け取るのか分からない場合は、(MQINQ 呼び出しを使用して) *MaxMsgLength* 属性を照会してから、そのサイズのバッファーを指定してください。

パフォーマンスが低下するのを避けるため、バッファー・サイズを、できるだけ実際のメッセージ・サイズに近いものにしてください。

MaxMsgLength 属性の詳細については、262 ページの『最大メッセージ長の増加』を参照してください。

メッセージがキューから取り出される順序

キューからメッセージを取り出す順序を制御できます。 このセクションでは、オプションを確認します

Priority

プログラムは、メッセージをキューに書き込むときに、優先順位を割り当てることができます (17 ページの『メッセージ優先順位』を参照)。 優先順位の同じメッセージは、コミットされた順ではなく、到着順にキューに保管されます。

キュー・マネージャーは、キューを厳密な FIFO (先入れ先出し) の順序か、優先順位での FIFO の順序で維持します。 これは、キューの *MsgDeliverySequence* 属性の設定値によって決められます。キューに到着したメッセージは、同じ優先順位をもつ最後のメッセージのすぐあとに挿入されます。

プログラムは、キューから最初のメッセージを読み取ることも、優先順位を無視して特定のメッセージ読み取ることもできます。 例えば、前に送信した特定のメッセージに対する応答をプログラムが処理する場合があります。 詳細については、256 ページの『特定のメッセージの読み取り』を参照してください。

あるアプリケーションがキューに一連のメッセージを書き込んだ場合、別のアプリケーションは、次の条件が満たされている場合は、それらのメッセージを書き込まれたときと同じ順序で取り出すことができます。

- すべてのメッセージの優先順位が同じである
- メッセージがすべて同じ作業単位内で、またはすべて作業単位外で書き込まれた
- キューは書き込みを行うアプリケーションに対してローカルである

これらの条件が満たされず、しかもある一定の順序で検索されるメッセージにアプリケーションが依存する場合は、アプリケーションはメッセージ・データ内に順序情報を含める必要があります。または、次のメッセージが送信される前に、アプリケーションはメッセージ受信の肯定応答を行う手段を確立しなければなりません。

論理的な順序付けと物理的な順序付け

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

物理順序とは、メッセージがキューに到着した順序を指します。 論理順序では、1 つのグループ内のメッセージおよびセグメントのすべてが論理順序に基づいて、互いに隣接して、グループに属する最初の項目の物理位置によって決まる位置に並びます。

グループ、メッセージ、およびセグメントの説明については、35 ページの『メッセージ・グループ』を参照してください。 物理順序と論理順序は、次の理由で異なる場合があります。

- 複数のグループが異なるアプリケーションから 1 つの宛先に同じようなタイミングで到着すると、物理順序の区別がなくなる可能性がある。
- 単一グループの中でも、その中の一部のメッセージの再ルーティングや遅延によってメッセージの順序が乱れることがある。

例えば、246 ページの図 34 に示すような論理順序があるとします。

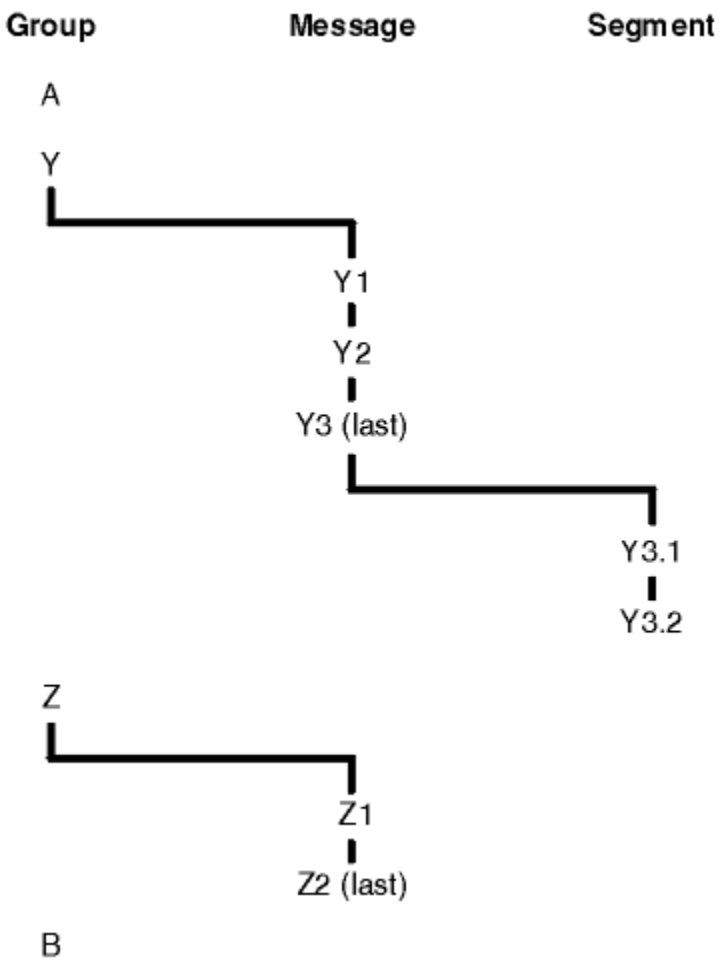


図 *34.* キューでの論理順序

これらのメッセージは、次に示す論理順序でキューに現れることがあります。

1. (グループに属していない) メッセージ A
2. グループ Y の論理メッセージ 1
3. グループ Y の論理メッセージ 2
4. グループ Y の (最後の) 論理メッセージ 3 のセグメント 1
5. グループ Y の (最後の) 論理メッセージ 3 の (最後の) セグメント 2
6. グループ Z の論理メッセージ 1
7. グループ Z の (最後の) 論理メッセージ 2
8. (グループに属していない) メッセージ B

しかし、物理順序はまったく異なる可能性があります。 各グループ内の最初の 項目の物理位置によって、そのグループ全体の論理位置が決まります。 例えば、グループ Y とグループ Z が同じようなタイミングで到着した場合に、グループ Z のメッセージ 2 がメッセージ 1 よりも優先されているとすれば、物理順序は 247 ページの図 35 のようになります。

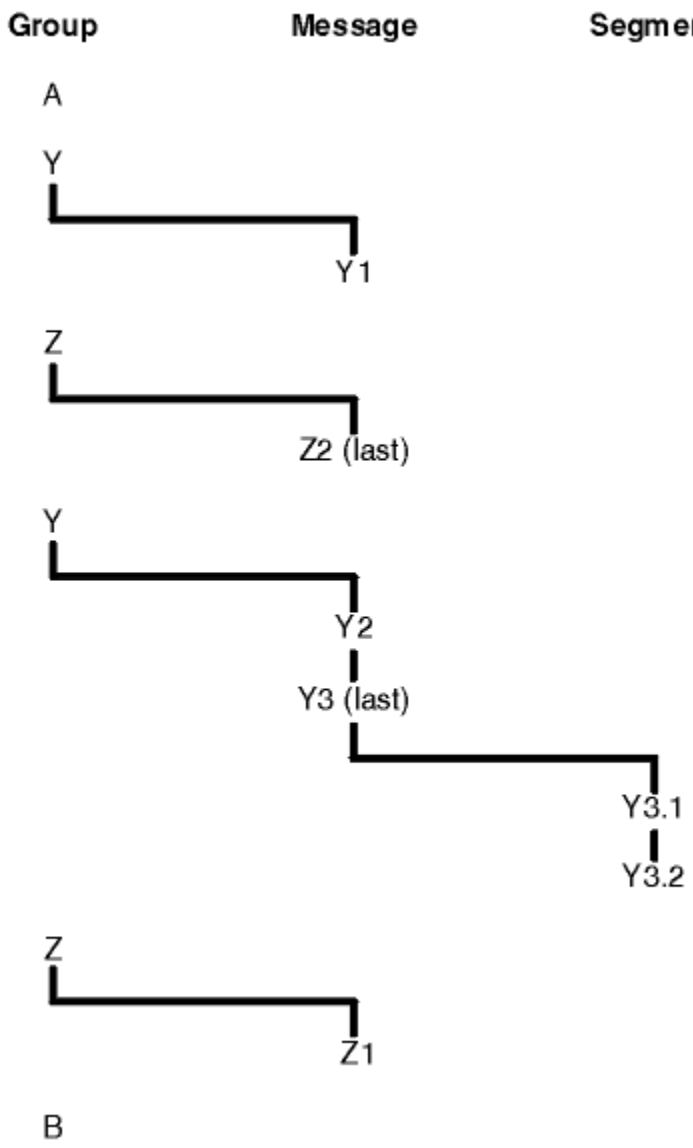


図 *35.* キューでの物理順序

これらのメッセージは、次に示す物理順序でキューに現れることがあります。

1. (グループに属していない) メッセージ A
2. グループ Y の論理メッセージ 1
3. グループ Z の論理メッセージ 2
4. グループ Y の論理メッセージ 2
5. グループ Y の (最後の) 論理メッセージ 3 のセグメント 1
6. グループ Y の (最後の) 論理メッセージ 3 の (最後の) セグメント 2
7. グループ Z の論理メッセージ 1
8. (グループに属していない) メッセージ B

注 : IBM WebSphere MQ for z/OS では、キューが GROUPID によって索引付けされている場合、キューでのメッセージの物理順序は保証されていません。

メッセージを取得するときに MQGMO_LOGICAL_ORDER を指定することにより、物理順序ではなく論理順序でメッセージを取得できます。

MQGMO_BROWSE_FIRST および MQGMO_LOGICAL_ORDER を指定して MQGET 呼び出しを発行した場合は、 MQGMO_BROWSE_NEXT を指定して後続の MQGET 呼び出しを発行するときにも

MQGMO_LOGICAL_ORDER を指定する必要があります。 逆に、MQGMO_BROWSE_FIRST を指定して MQGET を発行するときに MQGMO_LOGICAL_ORDER を指定しなかった場合は、MQGMO_BROWSE_NEXT を指定して後続の MQGET を発行するときに、このオプションを指定しなければなりません。

MQGET 呼び出しでキューに入っているメッセージをブラウズするためにキュー・マネージャーが保持しているグループおよびセグメント情報は、MQGET 呼び出しでキューからメッセージを除去するためにキュー・マネージャーが保持しているグループおよびセグメント情報とは異なります。
MQGMO_BROWSE_FIRST を指定する場合、キュー・マネージャーはブラウズに関するグループ情報およびセグメント情報を無視して、現行のグループおよび現行の論理メッセージが存在しない場合と同じようにキューを走査します。

注 : MQGMO_LOGICAL_ORDER を指定せずに MQGET 呼び出しを使用することにより、メッセージ・グループ (またはグループに属していない論理メッセージ) の最後を越えて ブラウズすることは避けてください。 例えば、キュー上で、グループの最後のメッセージがグループの最初のメッセージよりも前に ある場合に、(次のグループの最初のメッセージを検出するために) *MsgSeqNumber* を 1 に設定して MQMO_MATCH_MSG_SEQ_NUMBER を指定し、MQGMO_BROWSE_NEXT を使用してグループの最後を越えてブラウズを行うと、既にブラウズしたグループの最初のメッセージに戻ってしまいます。 これは、即時に発生する可能性があります。あるいは、 (介入グループがある場合) 何度かの MQGET 呼び出し後に発生するかもしれません。

無限ループの発生を避けるには、ブラウズの際にキューを *2* 回 オープンして次のように操作します。

- 各グループの最初のメッセージだけをブラウズするには、最初のハンドルを使用します。
- 特定のグループのメッセージだけをブラウズするには、2 番目のハンドルを使用します。
- 特定のグループの各メッセージをブラウズする前に、MQMO_* オプションを使用して 2 番目のブラウズ・カーソルを最初のブラウズ・カーソルの位置まで移動します。
- MQGMO_BROWSE_NEXT によるグループの最後よりあとのブラウズは実行しないようにします。

これについての詳細は、MQGET、MQMD、 および MQI オプションの妥当性検査に関する規則を参照してください。

アプリケーションでブラウズを実行するときには、一般に、論理順序と物理順序のどちらか一方を選択します。 一方、この 2 つのモードを切り替えて使用する場合には、最初に MQGMO_LOGICAL_ORDER を指定してブラウズを発行すると論理順序による位置が設定されるので注意してください。

その時点でグループ内の最初の項目が存在しなければ、そのグループは論理順序の一部としては扱われません。

あるグループ内にいったんブラウズ・カーソルが置かれると、グループ内の最初のメッセージが除去されたとしても、そのブラウズ・カーソルは引き続き同じグループを指示します。 ただし、MQGMO_LOGICAL_ORDER を使用してブラウズを発行した時点で最初の項目が存在しないグループにカーソルを移動することはできません。

MQPMO_LOGICAL_ORDER

MQPMO オプションは、アプリケーションがメッセージを論理メッセージのグループおよびセグメントに書き込む方法をキュー・マネージャーに通知します。 このオプションは、MQPUT 呼び出しでのみ指定できます。MQPUT1 呼び出しでは無効です。

MQPMO_LOGICAL_ORDER が指定されると、アプリケーションは後続の MQPUT 呼び出しを使用して次のことを行います。

1. 各論理メッセージ内のセグメントを、0 からセグメント・オフセットの小さい順に間を空けずに書き込む。
2. 論理メッセージ内のセグメントをすべて書き込んでから、その次の論理メッセージのセグメントを書き込む。
3. 各メッセージ・グループ内の論理メッセージを、1 からメッセージ順序番号の小さい順に間を空けずに書き込む。IBM WebSphere MQ メッセージ・シーケンス番号を自動的に増分します。
4. メッセージ・グループ内の論理メッセージをすべて書き込んでから、その次のメッセージ・グループの論理メッセージを書き込む。

アプリケーションはキュー・マネージャーにグループ内のメッセージと論理メッセージのセグメントを書き込む方法を指示したので、アプリケーションにおいて、MQPUT を呼び出すたびにグループの情報やセグメントの情報を維持および更新する必要はありません。その情報はキュー・マネージャーが維持および更新するからです。 具体的には、アプリケーションで MQMD の *GroupId*、*MsgSeqNumber*、および *Offset* フィールドを設定する必要がないということです。キュー・マネージャーがこれらのフィールドに適切な値を設定するからです。 アプリケーションが設定する必要があるのは、MQMD 内の *MsgFlags* フィールドだけです。これによりメッセージがグループに属している場合や、メッセージが論理メッセージのセグメントである場合を指示したり、グループ内の最後のメッセージまたは論理メッセージの最後のセグメントを指示したりできます。

メッセージ・グループまたは論理メッセージの開始後に MQPUT を呼び出すときは MQMD 中の *MsgFlags* にある適切な MQMF_* フラグを指定しなければなりません。 アプリケーションが、終了していないメッセージ・グループがある場合にグループ内にないメッセージを書き込もうとしたり、終了していない論理メッセージがある場合にセグメントではないメッセージを書き込もうとしたりすると、その呼び出しは失敗します。 また、状況に応じて理由コード MQRC_INCOMPLETE_GROUP または MQRC_INCOMPLETE_MSG が表示されます。 ただし、キュー・マネージャーは現在のメッセージ・グループまたは現在の論理メッセージの情報あるいはその両方を保存し、アプリケーションはメッセージを送信してこれらを終了します (アプリケーション・メッセージ・データなしも可能です)。このメッセージでは状況に応じて MQMF_LAST_MSG_IN_GROUP または MQMF_LAST_SEGMENT あるいはその両方を指定します。その後 MQPUT 呼び出しを再発行して、グループまたはセグメントにないメッセージを書き込みます。

247 ページの図 35 は、有効なオプションとフラグの組み合わせ、および各ケースでキュー・マネージャーが使用する *GroupId*、*MsgSeqNumber*、および *Offset* フィールドの値を示しています。 この表に示されていないオプションとフラグの組み合わせは無効です。 この表の列では、「どちらも」は「はい」または「いいえ」のどちらかを意味します。

LOG ORD
MQPMO_LOGICAL_ORDER オプションが呼び出しで指定されるかどうか。

MIG
MQMF_MSG_IN_GROUP または MQMF_LAST_MSG_IN_GROUP オプションが呼び出しで指定されるかどうか。

SEG
MQMF_SEGMENT または MQMF_LAST_SEGMENT オプションが呼び出しで指定されるかどうか。

SEG OK
MQMF_SEGMENTATION_ALLOWED オプションが呼び出しで指定されるかどうか。

Cur grp
現行のメッセージ・グループが呼び出しの前に存在するかどうか。

Cur log msg
現行の論理メッセージが呼び出しの前に存在するかどうかを示します。

その他の列
キュー・マネージャーが使用する値を示しています。「前の」という表現は、キュー・ハンドルに対して前のメッセージのフィールドで使用された値を示します。

表 *36.* 論理メッセージのグループおよびセグメントに関連した *MQPUT* オプション

指定するオプション				呼び出しの前のグループおよび論理メッセージの状況		キュー・マネージャーが使用する値		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***
Yes	いいえ	いいえ	いいえ	いいえ	いいえ	MQGI_NONE	1	0
Yes	いいえ	いいえ	Yes	いいえ	いいえ	新しいグループ ID	1	0

表 36. 論理メッセージのグループおよびセグメントに関連した *MQPUT* オプション (続き)

指定するオプション				呼び出しの前のグループおよび論理メッセージの状況		キュー・マネージャーが使用する値		
LOG ORD	**MIG**	**SEG**	**SEG OK**	**Cur grp**	**Cur log msg**	***GroupId***	***MsgSeqNumber***	***Offset***
Yes	いいえ	Yes	どちらも	いいえ	いいえ	新しいグループ ID	1	0
Yes	いいえ	Yes	どちらも	いいえ	Yes	前のグループ ID	1	前のオフセット + 前のセグメント長
Yes	Yes	どちらも	どちらも	いいえ	いいえ	新しいグループ ID	1	0
Yes	Yes	どちらも	どちらも	Yes	いいえ	前のグループ ID	前の順序番号 + 1	0
Yes	Yes	Yes	どちらも	Yes	Yes	前のグループ ID	前の順序番号	前のオフセット + 前のセグメント長
いいえ	いいえ	いいえ	いいえ	どちらも	どちらも	MQGI_NONE	1	0
いいえ	いいえ	いいえ	Yes	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	1	0
いいえ	いいえ	Yes	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	1	フィールド内の値
いいえ	Yes	いいえ	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	フィールド内の値	0
いいえ	Yes	Yes	どちらも	どちらも	どちらも	MQGI_NONE の場合は新しいグループ ID、その他はフィールド内の値	フィールド内の値	フィールド内の値

注 :

- MQPUT1 呼び出しでは、MQPMO_LOGICAL_ORDER は無効です。
- *MsgId* フィールドについては、MQPMO_NEW_MSG_ID または MQMI_NONE が指定されるとキュー・マネージャーは新しいメッセージ ID を生成し、それ以外の場合はフィールドの値を使用します。
- *CorrelId* フィールドについては、MQPMO_NEW_CORREL_ID が指定されるとキュー・マネージャーは新しい相関 ID を生成し、それ以外の場合はフィールドの値を使用します。

MQPMO_LOGICAL_ORDER を指定する場合、キュー・マネージャーでは、グループ内のすべてのメッセージおよび論理メッセージのセグメントを MQMD の *Persistence* フィールドに同じ値で書き込むことが必要です。 つまり、全部を持続にするか、または全部を非持続にする必要があります。 この条件が満たされないと、MQPUT 呼び出しは失敗し、理由コード MQRC_INCONSISTENT_PERSISTENCE が戻ります。

MQPMO_LOGICAL_ORDER オプションが作業単位に及ぼす影響は、以下のとおりです。

- グループ内または論理メッセージ内の最初の物理メッセージが 1 つの作業単位に書き込まれた場合、そのグループ内または論理メッセージ内の他の物理メッセージも、同じキュー・ハンドルが使用されていれば、すべて 1 つの作業単位に書き込む必要があります。 ただし、必ずしも同じ作業単位内に書き込む必要はありません。多数の物理メッセージから成るメッセージ・グループまたは論理メッセージを、キュー・ハンドルに対する 2 つ以上の連続した作業単位にまたがって分割することができます。
- グループ内または論理メッセージ内の最初の物理メッセージが 1 つの作業単位に書き込まれていない場合、同じキュー・ハンドルが使用されていれば、そのグループ内または論理メッセージ内の他の物理メッセージはどれも 1 つの作業単位に書き込むことができません。

これらの条件が満たされないと、MQPUT 呼び出しは失敗し、理由コード MQRC_INCONSISTENT_UOW が戻ります。

MQPMO_LOGICAL_ORDER が指定されている場合、MQPUT 呼び出しで提供される MQMD は、MQMD_VERSION_2 未満であってはいけません。 この状態が満たされない場合、呼び出しは失敗し、理由コード MQRC_WRONG_MD_VERSION が表示されます。

MQPMO_LOGICAL_ORDER を指定しないと、グループ内のメッセージおよび論理メッセージ内のセグメントは任意の順序で書き込まれます。また、必ずしも完全なメッセージ・グループおよび完全な論理メッセージを書き込む必要はありません。 *GroupId*、*MsgSeqNumber*、*Offset*、および *MsgFlags* フィールドが適切な値を持つようにするのは、アプリケーションの責任です。

システム障害が発生した後には、この手法を用いてメッセージ・グループまたは論理メッセージを途中から再始動します。 システムが再始動したら、アプリケーションで *GroupId*、*MsgSeqNumber*、*Offset*、*MsgFlags*、 および *Persistence* の各フィールドに適切な値を設定した後、必要に応じて MQPMO_SYNCPOINT または MQPMO_NO_SYNCPOINT を設定して MQPUT 呼び出しを発行できます。そのとき、MQPMO_LOGICAL_ORDER は指定しません。 この呼び出しが成功した場合、キュー・マネージャーはグループとセグメントの情報を保存し、キュー・ハンドルに対する後続の MQPUT 呼び出しで通常どおり MQPMO_LOGICAL_ORDER を指定できます。

MQPUT 呼び出しのためにキュー・マネージャーが保持しているグループおよびセグメント情報は、MQGET 呼び出しのためにキュー・マネージャーが保持しているグループおよびセグメント情報とは異なります。

キュー・ハンドルが指定されている場合には、アプリケーションでは、MQPMO_LOGICAL_ORDER を指定した MQPUT 呼び出しと MQPMO_LOGICAL_ORDER を指定していない MQPUT 呼び出しを組み合わせて使用できます。ただし、以下の点に注意してください。

- MQPMO_LOGICAL_ORDER を指定していない場合、MQPUT 呼び出しが成功するたびに、キュー・マネージャーは、キュー・ハンドルのグループおよびセグメント情報を、アプリケーションが指定する値に設定します。したがって、そのキュー・ハンドルに対してキュー・マネージャーで保持されていた既存のグループおよびセグメント情報がその値で置換されます。
- MQPMO_LOGICAL_ORDER を指定していない場合、現行のメッセージ・グループまたは論理メッセージが存在していても、呼び出しは失敗しません。呼び出しが成功しても MQCC_WARNING 完了コードが出される場合があります。252 ページの表 37 発生する可能性のあるさまざまなケースを示しています。 これらの場合に、完了コードが MQCC_OK 以外であれば、理由コードは以下のいずれか (該当するもの) になります。
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW

  注 : キュー・マネージャーは、MQPUT1 呼び出しのためにグループ情報およびセグメント情報を確認しません。

表 37. グループおよびセグメント情報と整合性のない *MQPUT* または *MQCLOSE* 呼び出しの結果		
現行の呼び出し	**前の呼び出しが MQPMO_LOGICAL_ORDER を指定した MQPUT**	**前の呼び出しが MQPMO_LOGICAL_ORDER を指定しない MQPUT**
MQPMO_LOGICAL_ORDER を指定した MQPUT	MQCC_FAILED	MQCC_FAILED
MQPMO_LOGICAL_ORDER を指定しない MQPUT	MQCC_WARNING	MQCC_OK
終了していないグループまたは論理メッセージを指定している MQCLOSE	MQCC_WARNING	MQCC_OK

メッセージおよびセグメントを論理順序で書き込むアプリケーションでは、最も簡単に使えるオプションとして MQPMO_LOGICAL_ORDER を指定します。 このオプションを指定すると、キュー・マネージャーがグループおよびセグメント情報を管理するので、アプリケーションでこの情報を管理する必要はなくなります。 しかし、特殊なアプリケーションでは、MQPMO_LOGICAL_ORDER オプションによって提供される以上の制御が必要な場合があります。その場合、MQPMO_LOGICAL_ORDER オプションを指定しないことができます。ただし、MQMD 内の *GroupId*、*MsgSeqNumber*、*Offset*、 および *MsgFlags* の各フィールドが正しく設定されていることを、各 MQPUT または MQPUT1 呼び出しの前に確認する必要があります。

例えば、受信した物理メッセージがグループに属していなくても、また論理メッセージのセグメントでなくても、そのメッセージを転送するアプリケーションでは、MQPMO_LOGICAL_ORDER を指定してはなりません。これには次の 2 つの理由があります。

- メッセージを取得して順番に書き込む場合、MQPMO_LOGICAL_ORDER を指定するとメッセージに新しいグループ ID が割り当てられます。こうなると、メッセージの発信元は、メッセージ・グループに対応する応答メッセージまたはレポート・メッセージとの相関をとることが困難になるか、不可能になります。
- 送信側のキュー・マネージャーと受信側のキュー・マネージャーの間にパスが複数あるような複雑なネットワークの場合には、物理メッセージが正しくない順序で到達することがあります。 MQGET 呼び出しに MQPMO_LOGICAL_ORDER と MQGMO_LOGICAL_ORDER をどちらも指定しなければ、転送側のアプリケーションでは、論理順序で次にあるメッセージを待たずに、それぞれの物理メッセージを到着と同時に取得して転送することができます。

グループ内のメッセージまたは論理メッセージのセグメントに関するレポート・メッセージを生成するアプリケーションでも、レポート・メッセージを書き込むときには MQPMO_LOGICAL_ORDER を指定してはなりません。

MQPMO_LOGICAL_ORDER はその他のどの MQPMO_* オプションとも組み合わせて指定できます。

論理的に順序付けされたグループのクラスター・キューへの書き込み (MQOO_BIND_ON_GROUP)

MQOO_BIND_ON_OPEN オプションは、このアプリケーションから出るすべてのメッセージ (したがって、すべてのグループ) が単一インスタンスに送付されることを保証します。 これには、アプリケーション・トラフィックがクラスター・キューの複数インスタンスにわたってロード・バランシングされないという欠点があります。 メッセージのグループをそのまま保持しているときにワークロード・バランシングを有効化するには、以下のオプションを設定する必要があります。

- MQPUT 呼び出しは MQPMO_LOGICAL_ORDER を指定する必要がある
- MQOPEN 呼び出しは、以下の 2 つのオプションのいずれかを指定する必要があります。
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF。さらに、キュー定義で DEFBIND(GROUP) を指定する必要があります。

こうすると、ワークロード・バランシングはメッセージのグループとグループの間で行われ、キューの MQCLOSE と MQOPEN は必要なくなります。 グループとグループの間 ということは、MQMF_MSG_IN_GROUP が MQMD(v2) または MQMDE に設定されて、進行中のグループが部分的に完了することはないという意味です。 グループが進行中のときは、オブジェクト・ハンドルに解決されたキュー・マネージャーおよびキュー名が再利用されます。

前のメッセージが MQPMO_LOGICAL_ORDER または MQMF_MSG_IN_GROUP であったものの、現在のメッセージがグループの一部ではない場合、PUT 呼び出しは MQRC_INCOMPLETE_GROUP で失敗します。

個々の MQPUT に MQPMO_LOGICAL_ORDER が指定されていない場合、現在アクティブなグループがなければ、そのメッセージに対してワークロード・バランシングが実行されます (MQOPEN 呼び出しに MQOO_BIND_NOT_FIXED が指定されていた場合のように)。

MQOO_BIND_ON_GROUP を使用して宛先にバインドされたメッセージについては、再割り振りは実行されません。 再割り振りの詳細については、35 ページの『メッセージ・グループ』を参照してください。

論理メッセージのグループ化

論理メッセージをグループにまとめて使用するのは、次の 2 つの主な理由によります。

- メッセージは、特定の順番で処理しなければならないことがあります。
- グループ内の各メッセージは、関連した方法で処理しなければならないことがあります。

いずれの場合も、グループ全体を同じ読み取り側アプリケーション・インスタンスを使用して取り出します。

例えば、4 つの論理メッセージで構成されるグループがあるとします。 メッセージの書き込み側アプリケーションでは、次のような操作を行います。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP

MQCMIT
```

読み取り側アプリケーションは、グループ内の最初のメッセージに MQGMO_ALL_MSGS_AVAILABLE オプションを指定します。 これにより、グループ内のすべてのメッセージが到着して初めて処理が開始されるようになります。 MQGMO_ALL_MSGS_AVAILABLE オプションは、グループ内の後続のメッセージでは無視されます。

グループの最初の論理メッセージが取り出されたときに、MQGMO_LOGICAL_ORDER を使用して、グループの残りの論理メッセージが順序どおりに取り出されるようにすることができます。

そこで、読み取り側アプリケーションでは、次のような操作を行います。

```
/* Wait for the first message in a group, or a message not in a group */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Process each remaining message in the group */
   ...

MQCMIT
```

メッセージのグループ化の詳しい例については、264 ページの『アプリケーションによる論理メッセージのセグメント化』および 253 ページの『複数の作業単位にわたるグループの書き込みと読み取り』を参照してください。

メッセージのグループをすべてクラスター・キューの同じ宛先インスタンスに割り振るようにアプリケーションが要求できるようにする方法についての詳細は、DefBind を参照してください。

複数の作業単位にわたるグループの書き込みと読み取り

前出の例では、グループ全体が書き込まれ、作業単位がコミットされるまで、メッセージやセグメントをノードから送信したり (宛先がリモートの場合)、取り出したりする処理を開始することはできません。 このため、グループ全体を書き込むのに長時間かかる場合や、ノード上でのキューのスペースが限られている場合には、不都合が生じることがあります。 これを解決するには、グループをいくつかの作業単位に分けて書き込みます。

グループを複数の作業単位内で書き込む場合には、書き込み側アプリケーションで障害が発生したときでも、グループの一部がコミットされる可能性があります。 このため、アプリケーションでは、各作業単位でコミットされた状況に関する情報を保管し、再始動後にその情報を使用して不完全なグループの処理を再開できるようにしておく必要があります。 この情報を記録するのに最も便利な場所は状況 (STATUS) キューです。 1 つのグループ全体が正常に書き込まれると、状況キューは空になります。

セグメント化を伴う場合も、論理は同様です。 この場合、 StatusInfo に *Offset* を含める必要があります。

複数の作業単位にわたってグループを書き込む場合の記述例を次に示します。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* First UOW */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Next and subsequent UOWs */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
StatusInfo = GroupId,MsgSeqNumber from MQMD
MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
MQCMIT

/* Last UOW */
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
MQCMIT
```

すべての作業単位がコミットされると、グループ全体が正常に書き込まれ、状況キューが空になります。 コミットされない作業単位がある場合は、状況に関する情報で指示されている個所からグループの処理を再開する必要があります。 最初の書き込みでは MQPMO_LOGICAL_ORDER を使用できませんが、以後の書き込みでは使用できます。

再始動処理の例を示します。

```
MQGET (StatusInfo from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
   ...

else
   /* Group was terminated prematurely */
   Set GroupId, MsgSeqNumber in MQMD to values from Status message
   PMO.Options = MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

   /* Now normal processing is resumed.
      Assume this is not the last message */
   PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
   StatusInfo = GroupId,MsgSeqNumber from MQMD
   MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
```

読み取り側アプリケーションでは、グループ全体が到着する前に、グループ内のメッセージの処理を開始したい場合があります。 これによって、グループ内のメッセージに関する応答時間が短縮されると共に、グループ全体を格納するストレージも必要なくなります。 このような利点を実現するために、メッセージ・グループごとにいくつかの作業単位を使用します。 リカバリー上の理由から、それぞれのメッセージは 1 つの作業単位内で取り出す必要があります。

そのためには、対応する書き込み側アプリケーションと同様に読み取り側アプリケーションでも、それぞれの作業単位がコミットされたときに、状況に関する情報がどこかに自動的に記録されなければなりません。 この場合も、この情報を記録するのに最も便利な場所は状況キューです。 1 つのグループ全体が正常に処理されると、状況キューは空になります。

注 : 中間作業単位については、作業単位ごとに 1 つの新しいメッセージ全体を書き込む代わりに、状況キューに対応するそれぞれの MQPUT でメッセージの 1 つのセグメントを書き込むように指定する (つまり、MQMF_SEGMENT フラグを設定する) ことによって、状況キューに対する MQGET を避けることができます。 最後の作業単位では、MQMF_LAST_SEGMENT を指定して最終セグメントを状況キューに書き込んだ上で、MQGMO_COMPLETE_MSG を指定した MQGET によって状況に関する情報をクリアします。

再始動処理では、単一の MQGET を使用して状況メッセージを読み取る代わりに、MQGMO_LOGICAL_ORDER によって、最後のメッセージに達するまで (つまり、それ以上セグメントが戻されなくなるまで) 状況キューをブラウズします。 再始動後の作業単位では、状況セグメントを書き込むときにオフセットを明示的に指定します。

以下の例では、アプリケーションのバッファーに、メッセージがセグメント化されているかどうかに関係なく、常にメッセージ全体を保持できるだけの大きさがあることを前提として、グループ内のメッセージのみについて考えます。 このため、それぞれの MQGET で MQGMO_COMPLETE_MSG を指定しています。セグメント化を伴う場合も同様にします (この場合には、StatusInfo に *Offset* を入れる必要があります)。

分かりやすくするために、1 つの作業単位で最大 4 つのメッセージが取り出されるものとします。

```
msgs = 0    /* Counts messages retrieved within UOW */
/* Should be no status message at this point */

/* Retrieve remaining messages in the group */
do while ( GroupStatus == MQGS_MSG_IN_GROUP )

   /* Process up to 4 messages in the group */
   GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
               | MQGMO_LOGICAL_ORDER
   do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
      MQGET
      msgs = msgs + 1
      /* Process this message */
      ...
   /* end while

   /* Have retrieved last message or 4 messages  */
   /* Update status message if not last in group */
   MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
   if ( GroupStatus == MQGS_MSG_IN_GROUP )
      StatusInfo = GroupId,MsgSeqNumber from MQMD
      MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
   MQCMIT
   msgs = 0
/* end while

if ( msgs > 0 )
   /* Come here if there was only 1 message in the group */
   MQCMIT
```

すべての作業単位がコミットされると、グループ全体が正常に取り出され、状況キューが空になります。コミットされない作業単位がある場合は、状況に関する情報で指示されている個所からグループの処理を再開する必要があります。 最初の取り出しでは MQGMO_LOGICAL_ORDER を使用できませんが、以後の取り出しでは使用できます。

再始動処理の例を示します。

```
MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
   /* Proceed to normal processing */
```

```
      ...

   else
      /* Group was terminated prematurely */
      /* The next message on the group must be retrieved by matching
         the sequence number and group id with those retrieved from the
         status information. */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID | MQMO_MATCH_MSG_SEQ_NUMBER,
            MQMD.GroupId      = value from Status message,
            MQMD.MsgSeqNumber = value from Status message plus 1
      msgs = 1
      /* Process this message */
      ...

      /* Now normal processing is resumed */
      /* Retrieve remaining messages in the group */
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )

         /* Process up to 4 messages in the group */
         GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
                     | MQGMO_LOGICAL_ORDER
         do while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) )
            MQGET
            msgs = msgs + 1
            /* Process this message */
            ...

         /* Have retrieved last message or 4 messages  */
         /* Update status message if not last in group */
         MQGET (from STATUS queue) GMO.Options = MQGMO_SYNCPOINT
         if ( GroupStatus == MQGS_MSG_IN_GROUP )
            StatusInfo = GroupId,MsgSeqNumber from MQMD
            MQPUT (StatusInfo to STATUS queue) PMO.Options = MQPMO_SYNCPOINT
         MQCMIT
         msgs = 0
```

特定のメッセージの読み取り

キューから特定のメッセージを読み取る方法はいくつも あります。 MsgId および CorrelId での選択、GroupId、MsgSeqNumber、および Offset での選択、MsgToken での選択が可能です。 キューをオープンするときに選択ストリングを使用することもできます。

キューから特定のメッセージを読み取るには、MQMD 構造体の *MsgId* および *CorrelId* フィールドを使用してください。 ただし、アプリケーションは明示的にこれらのフィールドを設定することがあるので、指定する値が固有のメッセージを識別しない場合もあります。 これらのフィールドに設定可能な値に対して、検索されるメッセージを 256 ページの表 38 に示します。 MQGET 呼び出しの *GetMsgOpts* パラメーターに MQGMO_MSG_UNDER_CURSOR を指定した場合は、これらのフィールドは入力時に無視されます。

表 *38.* メッセージ *ID* と相関 *ID* の使用方法

検索するメッセージ	***MsgId***	***CorrelId***
キューの最初のメッセージ	MQMI_NONE	MQCI_NONE
MsgId に一致する最初のメッセージ	非ゼロ	MQCI_NONE
CorrelId に一致する最初のメッセージ	MQMI_NONE	非ゼロ
MsgId および *CorrelId* に一致する最初のメッセージ	非ゼロ	非ゼロ

上記のいずれの場合も、最初 とは、選択基準を満たす最初のメッセージを意味します (ただし、MQGMO_BROWSE_NEXT が指定されている場合は別で、このときは選択基準を満たすメッセージ列上の次の メッセージを意味します)。

MQGET 呼び出しは、戻るときに *MsgId* フィールドと *CorrelId* フィールドに、戻されたメッセージ (存在する場合) のメッセージ ID と相関 ID を設定します。

MQMD 構造体の *Version* フィールドに 2 を設定すると、*GroupId*、*MsgSeqNumber*、および *Offset* の各フィールドが使用できます。257 ページの表 39 は、これらのフィールドの可能な設定について、どのメッセージが検索されるかを示しています。

表 39. グループ *ID* の使用方法	
検索するメッセージ	**一致オプション**
キューの最初のメッセージ	MQMO_NONE
MsgId に一致する最初のメッセージ	MQMO_MATCH_MSG_ID
CorrelId に一致する最初のメッセージ	MQMO_MATCH_CORREL_ID
GroupId に一致する最初のメッセージ	MQMO_MATCH_GROUP_ID
MsgSeqNumber に一致する最初のメッセージ	MQMO_MATCH_MSG_SEQ_NUMBER
MsgToken に一致する最初のメッセージ	MQMO_MATCH_MSG_TOKEN
Offset に一致する最初のメッセージ	MQMO_MATCH_OFFSET

注:

1. MQMO_MATCH_XXX は、MQMD 構造体の *XXX* フィールドが一致する値に設定されていることを示します。
2. MQMO フラグをそれぞれ組み合わせて使用できます。 例えば、MQMO_MATCH_GROUP_ID、MQMO_MATCH_MSG_SEQ_NUMBER、および MQMO_MATCH_OFFSET を共に使用すると、*GroupId*、*MsgSeqNumber*、および *Offset* の各フィールドによって識別されるセグメントを指定できます。
3. MQGMO_LOGICAL_ORDER は、キュー・ハンドルに応じて制御される状態情報によって異なるので、MQGMO_LOGICAL_ORDER を指定した場合には、取り出されるメッセージが変わることがあります。 詳細については、245 ページの『論理的な順序付けと物理的な順序付け』 および Options を参照してください。

MQGET 呼び出しでは、通常、キューから最初のメッセージを検索します。 MQGET 呼び出しを使用するときに特定のメッセージを指定すると、キュー・マネージャーはそのメッセージを見つけるまでキューを検索しなければなりません。 これは、アプリケーションのパフォーマンスに影響を及ぼす可能性があります。

バージョン 2 以降の MQGMO 構造体を使用している場合で、MQMO_MATCH_MSG_ID または MQMO_MATCH_CORREL_ID フラグを指定していない場合は、MQGET ごとに *MsgId* フィールドや *CorrelId* フィールドをリセットする必要はありません。

特定のメッセージを、そのメッセージの MsgToken と、MatchOption MQMO_MATCH_MSG_TOKEN を MQGMO 構造体で 指定することによって、キューから読み取ることができます。 MsgToken は、そのメッセージを最初にキューに書き込んだ MQPUT 呼び出しに よって戻されるか、直前の MQGET 操作によって戻され、キュー・マネージャーが再始動 されない限り一定のままです。

キューにあるメッセージのうちの一部のメッセージにのみ関心がある場合、MQOPEN または MQSUB 呼び出しで選択ストリングを使用することによって、 処理したいメッセージを指定することができます。 そうすると、MQGET は、その選択ストリングを 満足する次のメッセージを取り出します。 選択ストリングについて 詳しくは、22 ページの『Selectors』 を参照してください。

非永続メッセージのパフォーマンス向上

クライアントは、サーバーからのメッセージを必要とする際、サーバーに向けて要求を送信します。 クライアントは、コンシュームするメッセージごとに要求を別々に送信します。 こうした要求メッセージの送信を省くことにより、非永続メッセージをコンシュームするクライアントのパフォーマンスを向上するには、先読み を使用するようにクライアントを構成します。 先読みにより、アプリケーションからの要求がなくてもクライアントへのメッセージ送信が可能となります。

先読みが使用可能な場合、メッセージはクライアント上の先読みバッファー というメモリーのバッファーに送信されます。 クライアントには、先読みが使用可能でオープンされた各キューごとに先読みバッファーがあります。 先読みバッファー内のメッセージは非永続メッセージです。 クライアントは定期的に、消費したデータ量に関する更新情報をサーバーに提供します。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、WebSphere MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアントとリモート・キュー・マネージャーの両方が、WebSphere MQ バージョン 7 以降でなければなりません。
- クライアント・アプリケーションは、スレッド化された WebSphere MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みを使用すると、クライアント・アプリケーションから非永続メッセージをコンシュームする際のパフォーマンスを改善することができます。 このパフォーマンスの改善は、MQI アプリケーションと JMS アプリケーションの両方で使用することができます。 MQGET または非同期コンシュームを使用するクライアント・アプリケーションでは、非永続メッセージをコンシュームするときにパフォーマンス向上の効果が得られます。

クライアント・アプリケーションの設計によっては、先読みの使用が適していない場合があります。なぜなら、先読みの使用はすべてのオプションによってサポートされているわけではなく、またオプションによっては先読みの使用可能時に MQGET 呼び出し間で一貫性の確保が求められるためです。 クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納されているメッセージはそのクライアントの先読みバッファー内に保留されます。

変更前の選択基準で保留されたメッセージのバックログが不要になる場合、構成可能なパージ間隔をクライアント上で設定し、これらのメッセージをクライアントから自動的にパージすることができます。 パージ間隔は、クライアントによって決定される一連の先読みチューニング・オプションの 1 つです。 これらのオプションは、要件に合わせて調整することができます。

クライアント・アプリケーションが再始動されると、先読みバッファー内のメッセージは失われます。 逆に、先読みバッファーに移動されたメッセージが、その後に基礎となっているキューから削除されることもあります。これはバッファーからメッセージが除去される結果にはならないため、先読みバッファーを使用する MQGET 呼び出しは、もう存在していないメッセージを戻す可能性があります。

先読みはクライアントのバインディング用にのみ実行されます。 この属性は、他のすべてのバインディングでは無視されます。

先読みはトリガーに影響を及ぼしません。 メッセージがクライアントによって先読みされる際、トリガー・メッセージは生成されません。 先読みの有効時、アカウントおよび統計情報は先読みによって生成されません。 enabled.

パブリッシュ・サブスクライブ・メッセージングでの先読みの使用

サブスクライブ・アプリケーションがパブリケーションの送信先である宛先キューを指定すると、指定されたキューの DEFREADA 値が先読みのデフォルト値として使用されます。

サブスクライブ・アプリケーションが WebSphere MQ によるパブリケーションの送信先の管理を要求する場合、定義済みのモデル・キューに基づく動的キューとして管理キューが作成されます。 先読みのデフォルト値として使用されるのは、モデル・キューの DEFREADA 値です。 モデル・キューが、このトピックまたは親トピックのために定義されていない限り、デフォルトのモデル・キューである SYSTEM.DURABLE.PUBLICATIONS.MODEL または SYSTEM.NONDURABLE.PUBLICATIONS.MODEL が使用されます。

関連概念

261 ページの『AIX 上の非持続メッセージのためのパフォーマンスの調整』
AIX V5.3 以降を使用している場合は、非持続メッセージのパフォーマンスを最大限に使用するように調整パラメーターを設定することを検討してください。

関連タスク

260 ページの『先読みの使用可能化および使用不能化』

デフォルトでは、先読みは使用不可に設定されています。 キューまたはアプリケーションのレベルで、先読みを使用可能に設定することができます。

関連資料

259 ページの『MQGET オプションと先読み』
先読みが使用可能になっている場合に、すべての MQGET オプションがサポートされるわけではありません。オプションによっては MQGET 呼び出し間の一貫性が求められます。

MQGET オプションと先読み

先読みが使用可能になっている場合に、すべての MQGET オプションがサポートされるわけではありません。オプションによっては MQGET 呼び出し間の一貫性が求められます。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、WebSphere MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアントとリモート・キュー・マネージャーの両方が、WebSphere MQ バージョン 7 以降でなければなりません。
- クライアント・アプリケーションは、スレッド化された WebSphere MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

以下の表に、先読みの使用をサポートしているオプションおよび MQGET 呼び出し間の変更の可否を示します。

表 40. MQGET オプションと先読み

	先読みの有効時に使用可能で、MQGET 呼び出し間で変更可能 [5]	**先読みの有効時に使用可能だが、MQGET 呼び出し間で変更不可 [1]**	**先読みの有効時に使用不可の MQGET オプション [2]**
MQGET MQMD 値	MsgId[3] CorrelId[3]	Encoding CodedCharSetId	
MQGET MQGMO オプション	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_TRUNCATED _MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR[4] • MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ AVAILABLE

注 :

1. これらのオプションが MQGET 呼び出し間で変更された場合、MQRC_OPTIONS_CHANGED 理由コードが戻されます。
2. これらのオプションが最初の MQGET 呼び出しで指定されると、先読みは使用不可になります。 これらのオプションを後続の MQGET 呼び出しで指定すると、理由コード MQRC_OPTIONS_ERROR が戻されます。
3. クライアント・アプリケーションが MQGET 呼び出し間で MsgId および CorrelId の値を変更した場合、変更前の値を持つメッセージがクライアントに送信済みの可能性があり、コンシューム (または自動的にパージ) されるまでクライアントの先読みバッファー内に残されます。
4. MQGMO_MSG_UNDER_CURSOR は先読みでは使用できません。 キューのオープン時に、MQOO_BROWSE オプションと、MQOO_INPUT_SHARED オプションまたは MQOO_INPUT_EXCLUSIVE オプションの一方が指定されると、先読みは使用不可になります。
5. 先読みが使用可能に設定されている場合、最初の MQGET によって、 メッセージが参照されるのか、キューから取得されるのかが判別されます。 その後に、クライアント・アプリケーションが変更されたオ

プションで MQGET を使用すると (例えば、メッセージを最初に取得した後に参照しようとしたり、最初に参照した後に取得しようとしたりする場合)、MQRC_OPTIONS_CHANGED 理由コードが返されます。

クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納された、最初の選択基準に合致するメッセージは、クライアント・アプリケーションによってコンシュームされず、クライアントの先読みバッファー内で保留されます。 クライアントの先読みバッファーに多数の保留メッセージが格納されている場合、先読みによって得られる利点は失われ、コンシュームされるメッセージごとにサーバーへの要求を個別に行う必要があります。 先読みが効率的に使用されているかどうかを見極めるには、接続状況パラメーターである READA を使用します。

最初の MQGET 呼び出し時に不適合なオプションが指定された場合、アプリケーションからの要求により先読みを使用禁止にすることができます。 その状態になると、接続状況は先読み使用禁止中となります。

MQGET のこれらの制約事項のために、クライアント・アプリケーションの設計が先読みに適していないと判断した場合は、MQOPEN オプション MQOO_READ_AHEAD_NO を指定してください。 あるいは、開かれているキューのデフォルトの先読み値を NO または DISABLED に設定します。

先読みの使用可能化および使用不能化

デフォルトでは、先読みは使用不可に設定されています。 キューまたはアプリケーションのレベルで、先読みを使用可能に設定することができます。

このタスクについて

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、WebSphere MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアントとリモート・キュー・マネージャーの両方が、WebSphere MQ バージョン 7 以降でなければなりません。
- クライアント・アプリケーションは、スレッド化された WebSphere MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みを使用可能に設定するには、以下のようにします。

- 先読みをキューのレベルで構成するには、キュー属性の DEFREADA を YES に設定します。
- 先読みをアプリケーションのレベルで構成するには、以下のようにします。
  - 可能な限りいつでも先読みを使用できるようにするには、MQOPEN 関数呼び出しで MQOO_READ_AHEAD オプションを使用します。 DEFREADA キュー属性が DISABLED に設定されている場合、クライアント・アプリケーションは先読みを使用できなくなります。
  - キューで先読みが使用可能である場合のみ先読みを使用するには、MQOPEN 関数呼び出しで MQOO_READ_AHEAD_AS_Q_DEF オプションを使用します。

クライアント・アプリケーション設計が先読みに適していない場合は、以下の方法で先読みを使用不可にすることができます。

- 次のように設定することにより、キューのレベルで先読みを使用不可にします。クライアント・アプリケーションから要求されない限り先読みを不使用にしておきたい場合は、キュー属性の DEFREADA を NO に設定します。あるいは、クライアント・アプリケーションからの要求の有無に関係なく先読みを不使用にしておきたい場合は、DEFREADA を DISABLED に設定します。
- アプリケーションのレベルで先読みを使用不可にするには、MQOPEN 関数呼び出しで MQOO_NO_READ_AHEAD オプションを使用します。

2 種類の MQCLOSE オプションにより、先読みバッファーに格納されたメッセージをキューのクローズ時に処理する方法を設定することができます。

- 先読みバッファー内のメッセージを廃棄するには、MQCO_IMMEDIATE を使用します。

- キューのクローズ前に先読みバッファー内のメッセージをアプリケーションによってコンシュームさせるには、MQCO_QUIESCE を使用します。 MQCO_QUIESCE 付きで MQCLOSE を発行した際に、先読みバッファーにメッセージが残っていると、MQCC_WARNING と共に MQRC_READ_AHEAD_MSGS が戻ります。

AIX 上の非持続メッセージのためのパフォーマンスの調整

AIX V5.3 以降を使用している場合は、非持続メッセージのパフォーマンスを最大限に使用するように調整パラメーターを設定することを検討してください。

調整パラメーターがすぐに有効になるように設定するには、以下のコマンドを root ユーザーとして発行してください。

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

調整パラメーターがすぐに有効になり、リブート後も持続するように設定するには、以下のコマンドを root ユーザーとして発行してください。

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

通常は、非持続メッセージはメモリー内のみに保持されますが、状況によっては、AIX で非持続メッセージのディスクへの書き込みをスケジュールすることもできます。 ディスクに書き込むようにスケジュールされたメッセージは、ディスクへの書き込みが完了するまで MQGET が利用することはできません。 推奨される調整コマンドによって、このしきい値が異なります。 例えば、16 キロバイトのデータがキューに入れられた場合にメッセージをディスクに書き込むようにスケジュールする代わりに、マシン上の実記憶が満杯に近くなった場合にのみディスクへの書き込みが行われます。 これはグローバルな変更であり、他のソフトウェア・コンポーネントにも影響する可能性があります。

AIX 上で、マルチスレッド・アプリケーションを使用しており、特に複数のプロセッサーを搭載したマシン上で実行している場合には、パフォーマンスの向上および確実なスケジューリングのために、アプリケーションを始動する前に、mqm id `.profile` で AIXTHREAD_SCOPE=S を設定するか、ご使用の環境で AIXTHREAD_SCOPE=S を設定することを強くお勧めします。 以下に例を示します。

```
export AIXTHREAD_SCOPE=S
```

AIXTHREAD_SCOPE=S を設定すると、デフォルト属性を使用して作成されたユーザー・スレッドが、システム全体の競合範囲内に入れられます。 システム全体の競合範囲で作成されたユーザー・スレッドは、カーネル・スレッドにバインドされ、カーネルによってスケジュールされます。 ベースとなるカーネル・スレッドは、他のユーザー・スレッドとは共用されません。

ファイル記述子

エージェント・プロセスなどのマルチスレッド・プロセスを実行しているときに、ファイル記述子のソフト限界に達することがあります。 この制限により、 IBM WebSphere MQ 理由コード MQRC_UNEXPECTED_ERROR（2195）と、十分なファイル記述子がある場合は IBM WebSphere MQ FFST™ ファイルが得られます。

この問題は、処理できるファイル記述子の数を増やすことで回避できます。 これを行うには、mqm ユーザー ID の `/etc/security/limits` またはデフォルトのスタンザにある `nofiles` 属性を 10,000 に変更します。

システム・リソース限界

コマンド・プロンプトで以下のコマンドを使用して、データ・セグメントおよびスタック・セグメントのシステム・リソース限界を無制限に設定します。

```
ulimit -d unlimited
ulimit -s unlimited
```

4 MB より長いメッセージの処理

メッセージが大きすぎて、アプリケーション、キュー、またはキュー・マネージャーでの処理には適さない場合があります。 環境に 応じて、WebSphere MQ は 4 MB より長いメッセージを扱うためのいくつかの方法を提供しています。

V6 以降のすべての WebSphere MQ システムで、*MaxMsgLength* 属性を最大 100 MB まで増やすことができます。 キューを使用するメッセージのサイズを反映するようにこの値を設定してください。
WebSphere MQ for z/OS 以外の WebSphere MQ システムでは、以下を行うこともできます。

1. セグメント化したメッセージを使用する。 (メッセージは、アプリケーションとキュー・マネージャーのどちらでもセグメント化できます。)
2. 参照メッセージを使用する。

この節の残りの部分では、この 3 つの方法について説明します。

最大メッセージ長の増加

キュー・マネージャー属性の *MaxMsgLength* では、キュー・マネージャーで処理できる 1 つのメッセージの最大長を指定します。 同様に、キュー属性の *MaxMsgLength* では、キューで処理できる 1 つのメッセージの最大長を指定します。 サポートされるデフォルトの 最大メッセージ長は、使用している環境によって異なります。

大きなメッセージを処理する場合には、この 2 つの属性を個別に変更できます。 このキュー・マネージャー属性の値は 32768 バイトから 100 MB の範囲内で設定できます。キュー属性の値は 0 から 100 MB の範囲内で設定できます。

どちらか一方または両方の *MaxMsgLength* 属性を変更したあとで、変更が有効になるように、アプリケーションおよびチャネルを再始動してください。

これらの変更を行う場合、メッセージ長は、キューおよびキュー・マネージャーの *MaxMsgLength* 属性をどちらも超えてはなりません。 ただし、既存のメッセージの長さは、いずれかの属性より長くても構いません。

メッセージが大きすぎてキューで処理できない場合は、MQRC_MSG_TOO_BIG_FOR_Q が戻されます。 同様に、メッセージが大きすぎてキュー・マネージャーで処理できない場合は、
MQRC_MSG_TOO_BIG_FOR_Q_MGR が戻されます。

大きなメッセージはセグメント化すると、処理が簡潔に行えます。 ただし、メッセージをセグメント化する場合、次のことを考慮してください。

- キュー・マネージャー間の均一性が多少損なわれること。 メッセージ・データの最大サイズは、メッセージが書き込まれる各キュー (伝送キューを含む) の *MaxMsgLength* によって決定されます。 多くの場合、特に伝送キューについては、この値のデフォルト値としてキュー・マネージャーの *MaxMsgLength* が使用されます。 このため、メッセージをリモート・キュー・マネージャーに送るときには、メッセージが大きすぎるかどうかを予測するのが困難になります。
- システム・リソースの使用が増大すること。 例えば、アプリケーションで従来より大きなバッファーが必要になるほか、一部のプラットフォームでは、共用ストレージの使用量が増大することも考えられます。 キューのストレージが影響を受けるのは、より大きなメッセージが実際にストレージを必要とする場合だけです。
- チャネルのバッチ処理に影響を与えること。 大きなメッセージであっても、バッチ・カウントでは単に 1 つのメッセージとしてカウントされます。一方、伝送時間は通常より長くかかるので、他のメッセージの応答時間が長くなることになります。

メッセージのセグメント化

この情報を使用して、メッセージのセグメント化について学習します。

注 : IBM WebSphere MQ for z/OS 、または IBM WebSphere MQ classes for JMS を使用するアプリケーションではサポートされません。

トピック 262 ページの『最大メッセージ長の増加』で説明した最大メッセージ長を大きくする方法には、いくつかの欠点があります。 また、最大メッセージ長を大きくしても、まだメッセージが大きすぎて、キューまたはキュー・マネージャーで処理できないことも考えられます。 このような場合には、メッセージをセグメント化することができます。 セグメント化については、35 ページの『メッセージ・グループ』を参照してください。

次の各節では、メッセージのセグメント化の一般的な使用法を紹介します。 書き込みと、除去を伴う読み取りでは、MQPUT 呼び出しまたは MQGET 呼び出しが必ず 1 つの作業単位の中で動作するものとします。 ネットワーク上に不完全なグループが生じる可能性を減らすため、常にこの方法を使用することを考慮してください。 キュー・マネージャーで単一フェーズ・コミットが行われることを前提としていますが、その他の調整方法も有効です。

また、読み取り側アプリケーションでは、複数のサーバーが同一のキューを処理している場合に、あるサーバーが (以前に MQGMO_ALL_MSGS_AVAILABLE または MQGMO_ALL_SEGMENTS_AVAILABLE を指定しているため) メッセージまたはセグメントの検索に失敗することがないように、各サーバーで同様のコードが実行されるものとします。

複数の作業単位にわたるセグメント化メッセージの書き込みと読み取り

セグメント化したメッセージも、253 ページの『複数の作業単位にわたるグループの書き込みと読み取り』と同様の方法で複数の作業単位にわたって書き込んだり、読み取ったりできます。

ただし、グローバル作業単位では、セグメント化したメッセージを書き込んだり読み取ったりすることはできません。

キュー・マネージャーによるセグメント化と再組み立て

ここでは、あるアプリケーションが書き込んだメッセージを、別のアプリケーションが取り出すという最も単純な場合を例に挙げて説明します。 メッセージは大きくなることがあります。書き込み側アプリケーションや読み取り側アプリケーションが 1 つのバッファーで処理できないほど大きくはならないものの、キュー・マネージャーや、メッセージが書き込まれるキューが処理できない大きさになることがあります。

これらのアプリケーションで必要な変更はごくわずかです。まず、書き込み側アプリケーションで、必要な場合にキュー・マネージャーによるセグメント化を許可します。これは次のように指定します。

```
PMO.Options = (existing options)
MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

また、読み取り側アプリケーションでは、セグメント化されているメッセージを再組み立てするようにキュー・マネージャーに要求します。これは、次のように指定します。

```
GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
MQGET
```

この単純なシナリオでは、アプリケーションは、MQPUT 呼び出しの前に GroupId フィールド を MQGI_NONE にリセットして、キュー・マネージャーが固有のグループ ID を各メッセージに対して生成 できるようにしなければなりません。 これが行われないと、無関係の複数のメッセージが 同じグループ ID を持ち、その結果として以降の処理が正しく行われなくなる 可能性があります。

アプリケーションのバッファーは、(MQGMO_ACCEPT_TRUNCATED_MSG オプションを指定した場合を除き) 再組み立て後のメッセージを格納できる大きさでなければなりません。

メッセージのセグメント化を可能にするようにキューの MAXMSGLEN 属性を変更する場合は、以下の点を考慮してください。

- ローカル・キューでサポートされる最小メッセージ・セグメントは 16 バイトです。
- 伝送キューの場合、MAXMSGLEN にはヘッダーに必要なスペースも含める必要があります。 伝送キューに書き込まれる可能性のあるメッセージ・セグメントでは、予期されるユーザー・データの最大長より少なくとも 4000 バイト大きい値を使用することを考慮してください。

データ変換が必要な場合には、読み取り側アプリケーションで MQGMO_CONVERT を指定して変換を実行しなければならないことがあります。 完成したメッセージと共にデータ変換出口が渡されるので、データ変換は簡単に行うことができます。 メッセージがセグメント化されており、データが不完全でデータ変換出口が変換を実行できないようなデータ形式になっている場合は、送信側チャネルでデータ変換を試行しないでください。

アプリケーションによるセグメント化

キュー・マネージャーによるセグメント化では不十分な場合や、特定のセグメント境界でのデータ変換がアプリケーションで必要な場合に、アプリケーションによるセグメント化が使用されます。

アプリケーションによるセグメント化を使用する主な理由には、次の 2 つが考えられます。

1. メッセージが大きすぎて、アプリケーションが単一のバッファー内で処理できないため、キュー・マネージャーによるセグメント化だけでは不十分である。
2. 送信側チャネルでデータ変換を実行する必要がある。また、データは、個別のセグメントの変換を可能にするために書き込み側アプリケーションでセグメントの境界を定めなければならない形式である。

ただし、データ変換上の問題がない場合や、読み取り側アプリケーションで常に MQGMO_COMPLETE_MSG を使用する場合は、MQMF_SEGMENTATION_ALLOWED を指定することによってキュー・マネージャーによるセグメント化を許可することもできます。 次の例では、アプリケーションによってメッセージが 4 つのセグメントに分割されます。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT

MQCMIT
```

MQPMO_LOGICAL_ORDER を使用しない場合は、*Offset* および各セグメントの長さをアプリケーションで設定する必要があります。 この場合は、論理的な状態が自動的に保守されることはありません。

読み取り側アプリケーションでは、再組み立てされたメッセージをすべて保持できるだけの大きさのバッファーを確保できません。 したがって、読み取り側アプリケーションでは、セグメントを個別に処理できるようにする必要があります。

セグメント化されているメッセージの場合、このアプリケーションでは、該当する論理メッセージを構成するすべてのセグメントが存在しない限り、セグメントの処理を開始しないようにします。 このため、最初のセグメントについては MQGMO_ALL_SEGMENTS_AVAILABLE を指定します。
MQGMO_LOGICAL_ORDER を指定した場合に現行の論理メッセージが存在すると、
MQGMO_ALL_SEGMENTS_AVAILABLE は無視されます。

論理メッセージの最初のセグメントを取り出したあとは、MQGMO_LOGICAL_ORDER を使用して、論理メッセージの残りのセグメントが必ず順序どおりに取り出されるようにします。

異なるグループのメッセージについては考慮されません。 そのようなメッセージがあった場合は、キューの中で各メッセージの最初のセグメントが現れた順に処理されます。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

アプリケーションによる論理メッセージのセグメント化

メッセージは、グループ内の論理順序に従って保守する必要があります。また、それらの一部またはすべてが、アプリケーション・セグメンテーションを必要とするほど大きくなる場合があります。

次の例では、4 つの論理メッセージで構成されるグループを書き込みます。 3 番目のメッセージ以外は大きいので、書き込み側アプリケーションでセグメント化を行う必要があります。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP      | MQMF_LAST_SEGMENT

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP | MQMF_LAST_SEGMENT

MQCMIT
```

読み取り側アプリケーションでは、最初の MQGET で MQGMO_ALL_MSGS_AVAILABLE を指定します。 これは、グループ全体が使用可能になるまで、そのグループのメッセージおよびセグメントを一切取り出されないようにするためです。 グループ内の最初の物理メッセージが取り出された時点で、MQGMO_LOGICAL_ORDER を使用して、このグループのセグメントおよびメッセージが必ず順序どおりに取り出されるようにします。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

do while ( (GroupStatus   != MQGS_LAST_MSG_IN_GROUP) ||
             (SegmentStatus != MQGS_LAST_SEGMENT) )
   MQGET
   /* Process a segment or complete logical message.  Use the GroupStatus
      and SegmentStatus information to see what has been returned */
   ...

MQCMIT
```

注 : MQGMO_LOGICAL_ORDER を指定した場合に、現行グループが存在すると、MQGMO_ALL_MSGS_AVAILABLE は無視されます。

参照メッセージ

この情報を使用して、参照メッセージの詳細について学習します。

注 : WebSphere MQ for z/OS ではサポートされていません。

この方法を使用すると、あるノードから別のノードへ大きなオブジェクトを転送するときに、送信元ノードまたは宛先ノードで WebSphere MQ キューにオブジェクトを格納する必要がありません。 この方法は、データが別の形式で存在する場合 (メール・アプリケーションなどの場合) には特に便利です。

この方法を使用するには、チャネルの両側でメッセージ出口を指定します。 これを行う方法については、412 ページの『チャネル・メッセージ出口プログラム』を参照してください。

WebSphere MQ では、参照メッセージ・ヘッダー (MQRMH) の形式が定義されています。 これについては、MQRMH を参照してください。 このヘッダーは定義済みの形式名によって識別され、ヘッダーの後には実際のデータが存在する場合があります。

大きなオブジェクトの転送を開始するために、アプリケーションは、参照メッセージ・ヘッダーだけで構成される、ヘッダーのあとにデータのないメッセージを書き込むことができます。 このメッセージがノードから送信されると、メッセージ出口が適切な方法でオブジェクトを取り出して参照メッセージに付加します。 次に、(以前より大きくなった) そのメッセージを送信側メッセージ・チャネル・エージェント (MCA) に戻します。これにより受信側 MCA への伝送が可能になります。

受信側 MCA には、別のメッセージ出口が構成されます。 このメッセージ出口はこのようなメッセージのいずれかを受信すると、付加されたオブジェクト・データを使用してオブジェクトを作成しますが、オブジェクトを付けずに 参照メッセージを渡します。 これで、アプリケーションが参照メッセージを受信できる

ようになります。このアプリケーションは、このノードでオブジェクト (あるいは、少なくともこの参照メッセージで表されているオブジェクトの一部) が作成されたことを認識します。

送信側メッセージ出口が参照メッセージに付加できるオブジェクト・データの最大長は、チャネルの あらかじめ決められた最大メッセージ長によって制限されます。 この出口から MCA に戻すことができるメッセージは、渡された各メッセージにつき 1 つだけなので、書き込み側アプリケーションでは複数のメッセージを書き込むことによって 1 つのオブジェクトを転送できます。 それぞれのメッセージでは、付加されるオブジェクトの論理 長およびオフセットを識別できるようにする必要があります。 ただし、オブジェクトの合計サイズまたはチャネルで許容される最大サイズが分からない場合は、書き込み側アプリケーションで 1 つのメッセージだけを書き込み、渡されたメッセージに可能な限りのデータを付加し終えたときには送信側メッセージ出口自身が伝送キューに次のメッセージを書き込むように、メッセージ送信出口を設計してください。

この方法を使用して大きなメッセージを処理するときには、次の点を考慮してください。

- MCA およびメッセージ出口は WebSphere MQ のユーザー ID のもとで実行します。 送信側でオブジェクトを取り出すため、または受信側でオブジェクトを作成するために、メッセージ出口 (つまりユーザー ID) はオブジェクトにアクセスする必要があります。オブジェクトが任意のユーザー ID でアクセス可能な場合にのみ、このアクセスは可能です。 ただし、この場合にはセキュリティー上の問題が生じます。
- バルク・データが付加された参照メッセージが、宛先に達する前に複数のキュー・マネージャーを経由する必要がある場合は、その間にあるノードの WebSphere MQ キューにバルク・データが 存在します 。 ただし、このような場合のための特別なサポートや出口を提供する必要はありません。
- 経路変更または送達不能キューイングが可能な場合には、メッセージ出口の設計が困難になります。 このような場合は、オブジェクトを構成する各部分が順番どおりに到着しないことがあります。
- 参照メッセージが宛先に到着すると、受信側メッセージ出口はオブジェクトを作成します。 しかし、この処理は MCA の作業単位と同期化されないため、バッチがバックアウトされた場合、そのオブジェクトの同じ部分を含む別の参照メッセージが後のバッチで到着すると、メッセージ出口でその部分を再作成しようとする可能性があります。 例えば、オブジェクトが一連のデータベース更新であれば、このような処理を許容することはできません。 このような場合は、メッセージ出口で適用済みの更新を記録したログを保存する必要があります。そのときに、WebSphere MQ キューを使用する必要がある場合があります。
- オブジェクト・タイプの特性によっては、オブジェクトが不要になったときに削除できるように、メッセージ出口とアプリケーションが連携して使用回数を保守しなければならないことがあります。 また、インスタンス ID が必要となる場合もあります。この ID を指定するためのフィールドは、参照メッセージ・ヘッダー (MQRMH を参照) にあります。
- 参照メッセージを配布リストとして書き込む場合には、結果として得られる配布リストまたはそのノードの宛先ごとに、オブジェクトが取り出し可能でなければなりません。 使用回数を保守しなければならないことがあります。 また、ノードが、リスト内の一部の宛先については最終ノードでありながら、その他の宛先については中間ノードである可能性も考慮する必要があります。
- バルク・データの変換は、(送信側チャネルで要求したとしても) 通常は行われません。 これは、メッセージ出口が呼び出される前に 変換が実行されるためです。 そのため、発信元の送信側チャネルで変換を要求しないでください。 参照メッセージが中間ノードを経由する場合、(要求があれば) バルク・データは中間ノードから送信されるときに変換されます。
- 参照メッセージをセグメント化することはできません。

MQRMH 構造体と MQMD 構造体の使用法

参照メッセージ・ヘッダーとメッセージ記述子の各フィールドの説明については、MQRMH および MQMD を参照してください。

MQMD 構造体では、*Format* フィールドに MQFMT_REF_MSG_HEADER を設定してください。 MQGET 呼び出しで MQHREF 形式を要求すると、この形式がそのあとに続くバルク・データと共に WebSphere MQ で自動的に変換されます。

MQRMH 構造体の *DataLogicalOffset* フィールドと *DataLogicalLength* フィールドの使用例を次に示します。

書き込み側アプリケーションでは、次のような参照メッセージを書き込みます。

- 物理データなし
- *DataLogicalLength* = 0 (このメッセージはオブジェクト全体を表す)
- *DataLogicalOffset* = 0

オブジェクトが 70 000 バイトの長さであると仮定すると、送信側メッセージ出口は 最初の 40 000 バイトを次のような内容の参照メッセージによってチャネル経由で送信します。

- MQRMH と、そのあとに続く 40 000 バイトの物理データ
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (オブジェクトの最初から)

次に、次のようなもう 1 つのメッセージを伝送キューに入れます。

- 物理データなし
- *DataLogicalLength* = 0 (オブジェクトの最後まで)。 ここには、値 30 000 を指定することもできます。
- *DataLogicalOffset* = 40000 (ここが開始点となる)

このメッセージ出口が、送信側メッセージ出口で確認されると、次のようにしてデータの残り 30,000 バイトを付加すると共に、フィールドを設定します。

- MQRMH と、そのあとに続く 30,000 バイトの物理データ
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (ここが開始点となる)

MQRMHF_LAST フラグも設定します。

参照メッセージの使用例を示すサンプル・プログラムの説明については、96 ページの『分散プラットフォームにおけるサンプル・プログラム』を参照してください。

メッセージの待機

メッセージがキューに到着するまでプログラムを待機させる場合は、MQGMO 構造体の *Options* フィールドに、MQGMO_WAIT オプションを指定します。

MQGMO 構造体の *WaitInterval* フィールドを使用して、以下を指定します。メッセージがキューに到着するのを MQGET 呼び出しに待たせる最大時間 (ミリ秒)。

メッセージがこの時間内に到着しない場合は、MQGET 呼び出しが MQRC_NO_MSG_AVAILABLE の理由コードを出して完了します。

WaitInterval フィールドに定数 MQWI_UNLIMITED を入れて、無限の待機時間を指定できます。 しかし、制御が及ばないイベントによって、プログラムが長時間待機させられる可能性もあるので、この定数を使用するときは注意してください。 IMS アプリケーションでは、無限の待機間隔を指定すると IMS システムを終了できなくなるため、無限の待機間隔は指定しないでください。 (IMS の終了時にはすべての従属領域の終了が必要です。) 代わりに、IMS アプリケーションでは有限の待機間隔を指定することができます。その間隔のあと、呼び出しがメッセージを検索しないで完了した場合は、待機オプションを指定した別の MQGET 呼び出しを出してください。

注 : 複数のプログラムがメッセージを除去する ために同じ共用キューで待機している場合は、メッセージの到着によってアクティブ化されるプログラムは 1 つだけです。 しかし、複数のプログラムがメッセージをブラウズするために待機している場合は、すべてのプログラムをアクティブ化することができます。 詳細については、MQGMO にある MQGMO 構造体の *Options* フィールドに関する説明を参照してください。

待機間隔が終了する前に、キューまたはキュー・マネージャーの状態が変更された場合は、次のようなことが起こります。

- キュー・マネージャーが静止状態に入り、MQGMO_FAIL_IF_QUIESCING オプションを使用している場合は、待機が取り消され、MQGET 呼び出しは MQRC_Q_MGR_QUIESCING の理由コードを出して完了する。このオプションを使用していない場合は、呼び出しが待機し続ける。

- キュー・マネージャーが強制的に停止されたか、取り消された場合は、MQGET 呼び出しが MQRC_Q_MGR_STOPPING または MQRC_CONNECTION_BROKEN のどちらかの理由コードで完了する。
- キュー (またはキュー名が解決された結果のキュー) の属性が変更されたために、読み取り要求が禁止されている場合は、待機が取り消され、MQGET 呼び出しは MQRC_GET_INHIBITED の理由コードで完了する。
- キュー (または、キュー名を解決して得られるキュー) の属性が、FORCE オプションを必要とする方法で変更された場合は、待機が取り消され、MQGET 呼び出しが完了して理由コード MQRC_OBJECT_CHANGED が戻される。

これらの処置が取られる状況の詳細については、MQGMO を参照してください。

バックアウトのスキップ

MQGET 呼び出しに **MQGMO_MARK_SKIP_BACKOUT** オプションを指定すると、アプリケーション・プログラムが「*MQGET - エラー - バックアウト*」のループに入ることを回避できます。

注 : WebSphere MQ for z/OS でのみサポートされています。

作業単位の一部として、アプリケーション・プログラムは、キューからメッセージを読み取るために 1 つまたは複数の MQGET 呼び出しを発行することができます。 アプリケーション・プログラムがエラーを検出した場合は、その作業単位をバックアウトできます。 これによって、その作業単位の中で更新されたすべてのリソースが、作業単位の開始前の状態に戻され、さらに MQGET 呼び出しによって検索されたメッセージも、元に戻されます。

元に戻されたメッセージは、そのあとにアプリケーション・プログラムによって発行される MQGET 呼び出しで使用可能になります。 通常、これにより、アプリケーション・プログラムに問題が生じることはありません。 しかし、バックアウトの原因となったエラーを回避することができない場合には、メッセージをキューに戻すことによって、アプリケーション・プログラムが「*MQGET - エラー - バックアウト*」のループに入る可能性があります。

この問題を避けるために、MQGET 呼び出しに MQGMO_MARK_SKIP_BACKOUT オプションを指定してください。 これは、MQGET 要求を、アプリケーションが開始するバックアウトに含まれない (つまり、バックアウトしてはならない) ものとしてマークを付けます。 このオプションを使用すると、バックアウトが起こったときに、他のリソースに対する更新は要求どおりにバックアウトされますが、マークの付いたメッセージは、新しい作業単位のもとで検索されたかのように扱われます。

アプリケーション・プログラムは、新しい作業単位をコミットしたり、新しい作業単位をバックアウトしたりするときに WebSphere MQ 呼び出しを発行する必要があります。 例えば、プログラムは、メッセージが廃棄されたことを発信元に通知するなどの例外処理を行い、作業単位をコミットしてキューからメッセージを除去することができます。新しい作業単位が (何らかの理由で) バックアウトされた場合は、メッセージがキューに戻されます。

1 つの作業単位内には、バックアウトのスキップとしてマークを付けられた MQGET 要求は 1 つしか許されませんが、そのマークが付いていない他のメッセージもいくつかあります。 あるメッセージにバックアウトのスキップ・マークを付けたあとは、その作業単位で MQGMO_MARK_SKIP_BACKOUT を指定した MQGET 呼び出しを発行しても、その呼び出しは MQRC_SECOND_MARK_NOT_ALLOWED 理由コードで失敗します。

注 :

1. マークの付いたメッセージは、それを含む作業単位がアプリケーションのバックアウト要求によって終了した場合にだけ、バックアウトをスキップします。 その作業単位が他の理由でバックアウトされた場合は、そのメッセージも、バックアウトのスキップ・マークが付いていないときと同様に、キューにバックアウトされます。
2. スキップのバックアウトは、RRS で制御された作業単位に関連する DB2 ストアード・プロシージャーではサポートされていません。 例えば、MQGMO_MARK_SKIP_BACKOUT オプションを指定した MQGET 呼び出しは、理由コード MQRC_OPTION_ENVIRONMENT_ERROR で失敗します。

269 ページの図 36 は、MQGET 要求がバックアウトをスキップすることを要求されたときに、アプリケーション・プログラムに含まれる典型的な一連のステップを示しています。

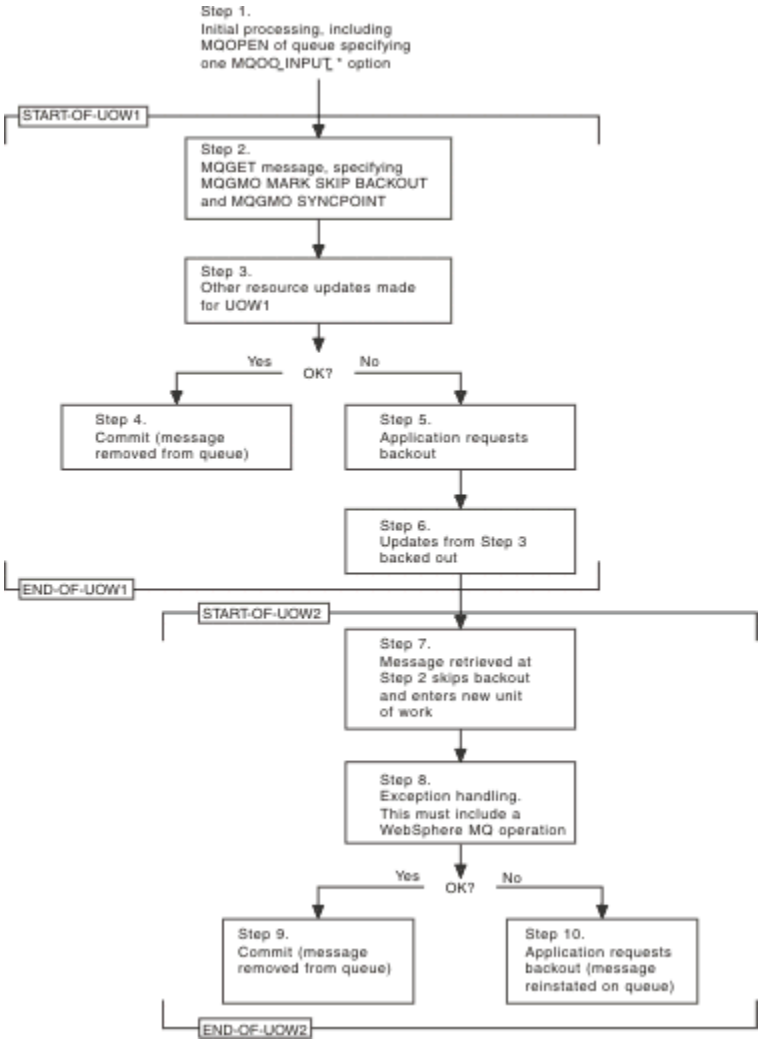


図 *36. MQGMO_MARK_SKIP_BACKOUT* の使用によるバックアウトのスキップ

269 ページの図 36 の各ステップは以下のとおりです。

ステップ 1
トランザクション内で初期処理が行われる。これには、キューをオープンするための MQOPEN 呼び出し (ステップ 2 でキューからメッセージを読み取るために MQOO_INPUT_* オプションの 1 つを指定した) が含まれます。

ステップ 2
MQGET が MQGMO_SYNCPOINT および MQGMO_MARK_SKIP_BACKOUT で呼び出される。 MQGMO_SYNCPOINT が必要なのは、MQGMO_MARK_SKIP_BACKOUT を有効にするためのに MQGET が作業単位内になければならないからです。 269 ページの図 36 では、この作業単位を UOW1 としています。

ステップ 3
他のリソースの更新が UOW1 の一部として行われる。 これには、さらに MQGET 呼び出し (MQGMO_MARK_SKIP_BACKOUT の指定がない) も含まれることがあります。

ステップ 4
ステップ 2 およびステップ 3 からの更新がすべて要求どおりに完了した。 アプリケーション・プログラムは更新をコミットし、UOW1 は終了します。 ステップ 2 で検索されたメッセージはキューから除去されます。

ステップ 5
ステップ 2 およびステップ 3 からの更新のいくつかが要求どおりに完了しなかった。 アプリケーション・プログラムは、これらのステップで行われた更新をバックアウトすることを要求します。

ステップ 6
ステップ 3 で行われた更新がバックアウトされる。

ステップ 7
ステップ 2 で行われた MQGET 要求が、バックアウトをスキップして、新しい作業単位 UOW2 の一部になる。

ステップ 8
UOW1 がバックアウトされたことに対応して、UOW2 が例外処理を行う (例えば、別のキューに対する MQPUT 呼び出しで問題が発生したために UOW1 がバックアウトされることを示す。)

ステップ 9
ステップ 8 が要求どおり完了し、アプリケーション・プログラムが活動をコミットし、UOW2 が終了した。 MQGET 要求が UOW2 の一部になっているので (ステップ 7 を参照)、このコミットによってメッセージがキューから除去されます。

ステップ 10
ステップ 8 が要求どおり完了しないので、アプリケーション・プログラムが UOW2 をバックアウトする。 読み取りメッセージ要求が UOW2 の一部になっているので (ステップ 7 を参照)、そのメッセージもバックアウトされ、キューに戻されます。 これにより、そのメッセージは、この同じアプリケーション・プログラムまたは別のアプリケーション・プログラムが発行する MQGET 呼び出しに対して、(キュー上の他のメッセージと同じように) 使用可能になります。

アプリケーション・データの変換

必要な場合は、MCA が、メッセージ記述子およびヘッダー・データを必要な文字セットとエンコード方式に変換します。 リンクの両端 (つまり、ローカル MCA またはリモート MCA) のいずれかで、変換が実行されます。

アプリケーションがメッセージをキューに書き込むと、ローカル・キュー・マネージャーにより、メッセージがキュー・マネージャーや MCA に処理される場合の制御を容易にするために、制御情報がメッセージ記述子に追加されます。 環境によっては、メッセージ・ヘッダー・データ・フィールドがローカル・システムの文字セットおよびエンコード方式で作成されます。

システム間でメッセージを移動させる場合、これらのアプリケーション・データを、受信側のシステムで必要とされる文字セットやエンコード方式に変換しなければならない場合があります。 このような変換は、受信側のシステムのアプリケーション・プログラム内から、または送信側のシステムの MCA によって実行できます。 受信側システムでデータ変換がサポートされている場合は、送信側システムで既に実行された変換の結果をそのまま使用せず、アプリケーション・プログラムを使用してアプリケーション・データを変換してください。

アプリケーション・プログラム内でのアプリケーション・データの変換は、MQGET 呼び出しに渡される MQGMO 構造体の *Options* フィールドで MQGMO_CONVERT オプションが指定されており、かつ以下のすべての 条件が満たされている場合に実行されます。

- キュー上のメッセージに関連付けられている MQMD 構造体の *CodedCharSetId* フィールド・セットまたは *Encoding* フィールド・セットが、MQGET 呼び出しで指定された MQMD 構造体の *CodedCharSetId* フィールド・セットまたは *Encoding* フィールド・セットとは異なる。
- メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、MQFMT_NONE でない。
- MQGET 呼び出しで指定された *BufferLength* が、ゼロでない。
- メッセージ・データの長さが、ゼロでない。
- キュー・マネージャーが、メッセージと MQGET 呼び出しに関連付けられている MQMD 構造体に指定された *CodedCharSetId* フィールドと *Encoding* フィールドの間の変換をサポートしている。 サポートされているコード化文字セット ID とマシン・エンコードの詳細については、CodedCharSetId および Encoding を参照してください。
- キュー・マネージャーが、メッセージ形式の変換をサポートしている。 メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、組み込み形式のうちの 1 つである場合、キュー・マネージャーはそのメッセージを変換できます。 *Format* が、組み込み形式のうちの 1 つでない場合には、データ変換出口を作成してそのメッセージを変換する必要があります。

送信側の MCA でデータの変換を行う場合は、変換を必要とする各送信側チャネルまたはサーバー・チャネルの定義で CONVERT(YES) キーワードを指定してください。 データ変換に失敗した場合、メッセージは送信側のキュー・マネージャーの DLQ に送信され、MQDLH 構造体の *Feedback* フィールドにその理由が示されます。 メッセージを DLQ に書き込めない場合は、チャネルがクローズされ、未変換のメッセージが伝送キュー上に残ります。 送信側の MCA でなくアプリケーション内でデータ変換を行うと、この状態を回避できます。

規則として、組み込み形式またはデータ変換出口により文字 データとして記述されるメッセージ内のデータは、そのメッセージで使用されるコード化文字セットから要求されたコード化文字セットに変換され、また数値 フィールドは、要求されたエンコード方式に変換されます。

組み込み形式を変換する際に使用する変換処理規則について、また独自のデータ変換出口を作成する方法については、416 ページの『データ変換出口の作成』 を参照してください。 また、言語サポート・テーブルおよびサポートされているマシン・エンコードについては、各国語およびマシン・エンコードを参照してください。

EBCDIC 改行文字の変換

EBCDIC プラットフォームから ASCII プラットフォームに送信したデータと、再び戻されるデータとを同じにする必要がある場合は、EBCDIC 改行文字の変換を制御しなければなりません。

この操作は、変更されていない変換テーブルを WebSphere MQ に強制的に使用させる、プラットフォーム依存型のスイッチを使用して行います。ただし、結果的に動作の整合性が失われることもあるので注意が必要です。

この問題が発生するのは、EBCDIC 改行文字がプラットフォーム間または変換テーブル間で正しく変換されていないからです。 この場合には、ASCII プラットフォーム上でデータを表示しても、正しく形式表示されないことがあります。 この問題が発生した場合、例えば、RUNMQSC を使って ASCII プラットフォームから IBM i システムをリモート管理することは非常に困難です。

EBCDIC のフォーマット・データを ASCII フォーマットに変換することに関する詳細は、データ変換を参照してください。

キュー上のメッセージのブラウズ

この情報を使用して、MQGET 呼び出しを使用してキューにあるメッセージをブラウズする方法を理解します。

キューにあるメッセージをブラウズするために MQGET 呼び出しを使用するには、次のステップに従ってください。

1. MQOO_BROWSE オプションを指定した MQOPEN を呼び出して、キューをブラウズするためにオープンする。
2. キューの最初のメッセージを表示するには、MQGMO_BROWSE_FIRST オプションを指定して MQGET を呼び出す。 必要なメッセージを検出するには、MQGMO_BROWSE_NEXT オプションを指定して MQGET をそのメッセージをすべて検索するまで繰り返し呼び出します。

   メッセージをすべて見るために、各 MQGET 呼び出しのあとで、MQMD 構造体の *MsgId* および *CorrelId* フィールドをヌルに設定する必要があります。
3. MQCLOSE を呼び出して、キューをクローズする。

ブラウズ・カーソル

キューをブラウズするためにオープン (MQOPEN) すると、その呼び出しによってブラウズ・カーソルが設定されます。ブラウズ・カーソルは、ブラウズ・オプションの 1 つを用いる MQGET 呼び出しで使用されます。 ブラウズ・カーソルをキューの最初のメッセージの前に位置する論理ポインターと考えることができます。

同じキューに対していくつかの MQOPEN 要求を発行することによって、(単一のプログラムから) 複数のブラウズ・カーソルを活動化させることができます。

ブラウズするため、MQGET を呼び出すとき、MQGMO 構造体に次のオプションの 1 つを指定します。

MQGMO_BROWSE_FIRST
MQMD 構造体に指定された条件を満たす、最初のメッセージのコピーを読み取る。

MQGMO_BROWSE_NEXT
MQMD 構造体に指定された条件を満たす、次のメッセージのコピーを読み取る。

MQGMO_BROWSE_MSG_UNDER_CURSOR
カーソルによって現在ポイントされているメッセージ、 つまり、MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT オプションのどちらかを使用して最後に取り出されたメッセージのコピーを読み取る。

どの場合もメッセージはキューに残ります。

キューをオープンしたときは、ブラウズ・カーソルはキューの最初のメッセージの直前に論理的に置かれます。 このことは、MQOPEN 呼び出しのあと、即時に MQGET 呼び出しを行う場合は、MQGMO_BROWSE_NEXT オプションを使用して最初のメッセージをブラウズできることを意味します。MQGMO_BROWSE_FIRST オプションを使用する必要はありません。

メッセージがキューからコピーされる順序は、キューの *MsgDeliverySequence* 属性によって決まります (詳細については、245 ページの『メッセージがキューから取り出される順序』を参照してください。)

- 272 ページの『FIFO (先入れ先出し) 順序のキュー』
- 272 ページの『優先順位順序のキュー』
- 272 ページの『未コミット・メッセージ』
- 273 ページの『キュー順序の変更』
- 273 ページの『キューの索引の使用』

FIFO (先入れ先出し) 順序のキュー

この順序におけるキューの最初のメッセージは、一番長くキューに置かれていたメッセージです。

MQGMO_BROWSE_NEXT を使用すると、キューのメッセージを順番に読み取ります。 この順次のキューはメッセージを最後に置くので、ブラウズ中にはキューに書き込まれたどのメッセージでも見ることができます。 カーソルがキューの終わりに達したことを認識すると、ブラウズ・カーソルはその場に留まり、MQRC_NO_MSG_AVAILABLE を戻します。 その場合、次のメッセージを待ってカーソルをそのままにするか、または MQGMO_BROWSE_FIRST 呼び出しを使用してキューの始めにリセットすることができます。

優先順位順序のキュー

この順序にあるキューの最初のメッセージは、そのキュー上で最長であり、また MQOPEN 呼び出しが発行されたときに最高優先順位をもつメッセージです。

MQGMO_BROWSE_NEXT を使用すると、キューに入っているメッセージを読み取ることができます。

ブラウズ・カーソルは次のメッセージを指します。最初のメッセージの優先順位から始まり、優先順位の最も低いメッセージで終わります。 この間にキューに書き込まれたメッセージは、それが現行ブラウズ・カーソルで識別されているメッセージと同じ、もしくはそれより低い優先順位のメッセージである限り、すべてブラウズされます。

キューに書き込まれた、それより高い優先順位のメッセージは、次の場合にのみブラウズされます。

- ブラウズするキューを再びオープンする。この時点で新規のブラウズ・カーソルが確立されます。
- MQGMO_BROWSE_FIRST オプションを使用する。

未コミット・メッセージ

コミットされていないメッセージはブラウズでは認識されず、ブラウズ・カーソルはそのメッセージをスキップします。

1 つの作業単位内のメッセージは、その作業単位がコミットされるまでブラウズされません。 コミット時にはキューでのメッセージの位置は変わらないので、スキップされる、コミットされていないメッセージ

は、MQGMO_BROWSE_FIRST オプションを使用し、キューでの作業を再び行わない限り、コミット時 であっても表示されません。

キュー順序の変更

メッセージ送達順序を、メッセージがキューにある間に優先順位から FIFO に変更した場合、既にキューに入っているメッセージの順序は変更されません。 後からキューに追加されるメッセージは、そのキューのデフォルトの優先順位で処理されます。

キューの索引の使用

索引付きキューの中のメッセージ (持続メッセージと非持続メッセージのどちらか一方、あるいはその両方) の優先順位が単一の場合、キュー・マネージャーは、特定の形式でブラウズするときに索引を使用します。

注 : WebSphere MQ for z/OS でのみサポートされています。

索引付きキューの中のメッセージの優先順位が単一の場合、以下のいずれかの形式のブラウズが使用されます。

1. キューが MSGID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の MSGID を渡すブラウズ要求が処理されます。
2. キューが CORRELID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の CORRELID を渡すブラウズ要求が処理されます。
3. キューが GROUPID で索引を付けられている場合、宛先メッセージを見つけるために索引を使用して MQMD 構造体の GROUPID を渡すブラウズ要求が処理されます。

ブラウズ要求が MQMD 構造体の MSGID、CORRELID、または GROUPID を渡さないものの、キューに索引が付けられていてメッセージが戻される場合には、メッセージの索引項目を見つけなければならず、その中の情報を使ってブラウズ・カーソルが更新されることになります。 選択の幅の広い索引値を使用する場合でも、ブラウズ要求に特に余分な処理は追加されません。

メッセージ長が分からない場合のメッセージのブラウズ

メッセージのサイズが分からないときには、*MsgId* フィールド、*CorrelId* フィールド、および *GroupId* フィールドを使用してメッセージを検索しなくても、次のように、
MQGMO_BROWSE_MSG_UNDER_CURSOR オプションを使用してメッセージをブラウズできます。

1. 次を指定して MQGET を発行する。
   - MQGMO_BROWSE_FIRST オプションまたは MQGMO_BROWSE_NEXT オプションのいずれか
   - MQGMO_ACCEPT_TRUNCATED_MSG オプション
   - バッファー長ゼロ

   注 : 別のプログラムが同じメッセージを受け取る可能性が高い場合には、MQGMO_LOCK オプションの併用を検討してください。 MQRC_TRUNCATED_MSG_ACCEPTED が戻されます。
2. 戻された *DataLength* を使用して必要なストレージを割り振る。
3. MQGMO_BROWSE_MSG_UNDER_CURSOR を指定して MQGET を発行する。

ポインターが指しているメッセージは、最後に検索されたメッセージです。ブラウズ・カーソルは移動されません。 MQGMO_LOCK オプションを使用してメッセージをロックするか、それとも MQGMO_UNLOCK オプションを使用してロック済みメッセージをアンロックするかを選択できます。

キューがオープンされてから、MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT オプションを指定した MQGET が発行されていない場合、呼び出しは失敗します。

ブラウズしたメッセージの除去

メッセージをブラウズするためだけでなく除去するためにキューをオープンしていた場合は、既にブラウズしたメッセージをキューから除去することができます。 (MQOPEN 呼び出しでは、MQOO_INPUT_* オプションの 1 つ、および MQOO_BROWSE オプションを指定する必要があります。)

メッセージを除去するには、MQGET をもう一度呼び出してください。ただし、このときには MQGMO 構造体の *Options* フィールドに MQGMO_MSG_UNDER_CURSOR を指定してください。この場合の MQGET 呼び出しでは、MQMD 構造体の *MsgId* フィールド、*CorrelId* フィールド、および *GroupId* フィールドが無視されます。

ブラウズおよび除去のステップの間で、別のプログラムがキューからメッセージを除去してしまう場合があります。このとき、ブラウズ・カーソルの下のメッセージも除去されることもあります。 その場合、MQGET 呼び出しから、メッセージが利用できないことを示す理由コードが戻ります。

論理順序でのメッセージのブラウズ

245 ページの『論理的な順序付けと物理的な順序付け』では、キューにあるメッセージの論理順序と物理順序の相違について説明しています。 論理順序と物理順序の区別は、キューをブラウズするときに特に重要です。これは、ブラウズでは、通常はメッセージを削除しないため、またブラウズ操作が必ずしもキューの先頭から開始されるとは限らないためです。

アプリケーションで、(論理順序を使用して) あるグループのさまざまなメッセージをブラウズする場合には、論理順序に従って次のグループの先頭に進むことが重要です。これは、あるグループの最後のメッセージが、物理的には次のグループの最初のメッセージのあと に 置かれている場合もあるためです。 MQGMO_LOGICAL_ORDER オプションを使用すると、常に論理順序に従ってキューを走査することができます。

ブラウズ操作では、注意して MQGMO_ALL_MSGS_AVAILABLE (または MQGMO_ALL_SEGMENTS_AVAILABLE) を使用してください。 MQGMO_ALL_MSGS_AVAILABLE を指定した論理メッセージの場合を例に挙げて説明します。 このオプションを指定した場合、論理メッセージが使用可能になるのはグループ内の残りのメッセージがすべて存在する場合だけです。 この条件に当てはまらない場合、論理メッセージはスキップされます。 このため、欠落しているメッセージがあとから到着しても、次のメッセージをブラウズする操作では認識されない可能性があります。

例えば、次のような論理メッセージがあるとします。

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

この場合に、MQGMO_ALL_MSGS_AVAILABLE を指定してブラウズ関数を発行すると、グループ 456 の最初のメッセージが戻され、ブラウズ・カーソルはこの論理メッセージに 置かれたままになります。 次に、グループ 123 の 2 番目 (最後) のメッセージが到着したとします。

```
Logical message 1 (not last) of group 123
Logical message 2 (last)     of group 123
Logical message 1 (not last) of group 456  <=== browse cursor
Logical message 2 (last)     of group 456
```

ここで、同じように次のメッセージをブラウズする関数を発行した場合は、グループ 123 の最初のメッセージがブラウズ・カーソルより前 にあるため、このグループのメッセージがすべて揃っていることが認識されません。

場合によっては (例えば、1 つのグループ全体が存在するときにメッセージを取り出して除去したような場合は)、MQGMO_ALL_MSGS_AVAILABLE を MQGMO_BROWSE_FIRST と共に使用することができます。 それ以外の場合は、欠落していたメッセージが新たに到着していないかどうかを確認するためにブラウズ走査を繰り返す必要があります。ただし、単に MQGMO_BROWSE_NEXT および MQGMO_ALL_MSGS_AVAILABLE を指定して MQGMO_WAIT を発行しても、このようなメッセージは取り出されません。 (これは、メッセージの走査が完了したあとで優先順位の高いメッセージが到着するような場合にも起こります。)

以下の節では、セグメント化されていないメッセージに関するブラウズ走査の例を紹介します。セグメント化されているメッセージをブラウズする場合も、原理は同様です。

グループ化されているメッセージのブラウズ

この例では、キューにある各メッセージをアプリケーションから論理順序に従ってブラウズします。

キューにあるメッセージは、グループ化されている場合があります。 グループ化されているメッセージについては、そのグループ内のメッセージがすべて到着するまで、アプリケーションではグループの処理を一切開始しません。 このため、グループ内の最初のメッセージについては、MQGMO_ALL_MSGS_AVAILABLE を指定します。ただし、後続のメッセージについては、このオプション必要ありません。

この例では、MQGMO_WAIT を使用します。 ただし、新しいグループが到着した場合には、274 ページの『論理順序でのメッセージのブラウズ』で示した理由によって待機の条件は満たされますが、ブラウズ・カーソルが既にグループ内の最初の論理メッセージを通過したあとに残りのメッセージが到着した場合は、待機の条件は満たされません。 それでも、適切な間隔だけ待機することで、新しいメッセージまたはセグメントを待機する間、アプリケーションがたびたびループすることがなくなります。

走査を常に論理順序で実行するために、最初から最後まで MQGMO_LOGICAL_ORDER を使用します。 これは、除去を伴う MQGET の例とは対照的です。除去を伴う MQGET の例では、各グループが除去されるためにグループ内の最初の (または唯一の) メッセージを検索するときに MQGMO_LOGICAL_ORDER を使用していません。

アプリケーションのバッファーは、メッセージがセグメント化されているかどうかにかかわらず、常にメッセージ全体を保持できるだけの大きさであることが前提とされています。 このため、それぞれの MQGET で MQGMO_COMPLETE_MSG を指定しています。

あるグループ内の論理メッセージをブラウズする例を次に示します。

```
/* Browse the first message in a group, or a message not in a group */
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
MQGET GMO.MatchOptions = MQMO_MATCH_MSG_SEQ_NUMBER, MD.MsgSeqNumber = 1
/* Examine first or only message */
 ...

GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group */
   ...
```

グループのブラウズは、MQRC_NO_MSG_AVAILABLE が戻るまで繰り返されます。

ブラウズおよび除去を伴う取り出し

この例では、アプリケーションはグループ内の各論理メッセージをブラウズしてから、そのグループを破壊的に取得するかどうかを決定します。

この例の最初の部分は、1 つ前の例と似ています。 ただし、この例では、あるグループ全体をブラウズしてから、必要に応じて最初に戻り、そのグループを取り出してキューから除去します。

この例では、各グループが除去されるので、グループ内の最初のメッセージまたは唯一のメッセージを検索するときに MQGMO_LOGICAL_ORDER は使用しません。

ブラウズと、削除を伴う取り出しの例を次に示します。

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MESSAGES_AVAILABE | MQGMO_WAIT
do while ( GroupStatus == MQGS_MSG_IN_GROUP )
   MQGET
   /* Examine each remaining message in the group (or as many as
      necessary to decide whether to get it destructively) */
   ...

if ( we want to retrieve the group destructively )

   if ( GroupStatus == ' ' )
      /* We retrieved an ungrouped message */
      GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = 0
      /* Process the message */
      ...

   else
      /* We retrieved one or more messages in a group.  The browse cursor */
```

```
      /* will not normally be still on the first in the group, so we have */
      /* to match on the GroupId and MsgSeqNumber = 1.                    */
      /* Another way, which works for both grouped and ungrouped messages,*/
      /* would be to remember the MsgId of the first message when it was  */
      /* browsed, and match on that.                                      */
      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
      MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                             | MQMO_MATCH_MSG_SEQ_NUMBER,
           (MQMD.GroupId      = value already in the MD)
            MQMD.MsgSeqNumber = 1
      /* Process first or only message */
      ...

      GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                  | MQGMO_LOGICAL_ORDER
      do while ( GroupStatus == MQGS_MSG_IN_GROUP )
         MQGET
         /* Process each remaining message in the group */
         ...
```

ブラウズされたメッセージの送達が繰り返されることの回避

オープンのオプションおよびメッセージ読み取りのオプションのうち特定のオプションを使用することによって、 メッセージにブラウズ済みのマークを付け、それらのメッセージを現行アプリケーションまたは他の連携アプリケーションが 再び取り出さないようにすることができます。 明示的または自動的にメッセージのマークを解除して、 ブラウズに使用できる状態に戻すことができます。

キューにあるメッセージをブラウズする場合、 メッセージを破壊的に読み取る場合に想定される順序とは異なる 順序でメッセージを取り出すことが可能です。 具体的には、同じメッセージを複数回ブラウズ できますが、もしそのメッセージがキューから削除されるのであればそれは不可能です。 これを 避けるため、メッセージにブラウズ済みのマークを付け、マークされた メッセージの取り出しを回避することができます。 これを、マーク付けブラウズ といいます。 ブラウズされた メッセージにマークを付けるには、メッセージ読み取りオプション MQGMO_MARK_BROWSE_HANDLE を 使用し、マークが付けられていないメッセージのみを取り出すには MQGMO_UNMARKED_BROWSE_MSG を使用します。
MQGMO_BROWSE_FIRST オプション、MQGMO_UNMARKED_BROWSE_MSG オプション、 および MQGMO_MARK_BROWSE_HANDLE オプションを組み合わせて使用し、繰り返し MQGET を発行すると、キューにある各メッセージを順々に取り出すことになります。 これは、 メッセージがスキップされないように MQGMO_BROWSE_FIRST が使用されていても、メッセージ送達の繰り返しを防止します。 これらのオプションの組み合わせが、1 つの定数 MQGMO_BROWSE_HANDLE で表されます。 まだブラウズされたことのないメッセージがキューにない場合、MQRC_NO_MSG_AVAILABLE が 戻されます。

複数のアプリケーションが同じキューをブラウズする場合、 それらのアプリケーションは、MQOO_CO_OP および MQOO_BROWSE オプションを指定してキューをオープンできます。 各 MQOPEN によって戻される オブジェクト・ハンドルは、連携グループの一部であると見なされます。
MQGMO_MARK_BROWSE_CO_OP オプションを指定した MQGET 呼び出しで戻される メッセージは、この連携ハンドル・セット用にマークされるものと見なされます。

あるメッセージに しばらくの間マークが付けられていた場合、キュー・マネージャーによって自動的に マーク解除が行われるようにし、もう一度ブラウズ可能にすることができます。 キュー・マネージャー 属性 MsgMarkBrowseInterval が、連携するハンドルのセット用にメッセージにマークが付いたままにしておく時間をミリ秒単位で指定します。 MsgMarkBrowseInterval の値が -1 の場合、 自動的にメッセージのマーク解除が行われることはないことを意味します。

単一プロセス または連携するプロセスのセットがメッセージのマーク付けを停止すると、 マークされていたすべてのメッセージがマーク解除されます。

連携ブラウズの例

1 つのディスパッチャー・アプリケーションの複数のコピー を実行して、キューにあるメッセージをブラウズし、各メッセージの内容に 基づいてコンシューマーを開始することができます。 各ディスパッチャーでは、 MQOO_CO_OP を指定してキューをオープンします。 これは、これらの複数のディスパッチャーが連携し、 互いに他のディスパッチャーのマーク付きメッセージに注意することを示します。 各ディスパッチャーは、 MQGMO_BROWSE_FIRST オプション、MQGMO_UNMARKED_BROWSE_MSG オプション、 および MQGMO_MARK_BROWSE_CO_OP オプションを指定して (1 つの定数 MQGMO_BROWSE_CO_OP を使用してこれらのオプションの組み合わせを表すことができます)、MQGET 呼び出しを繰り返します。 各ディスパッチャー・アプリケーション は、他の連携ディスパッチャーによってまだマークが付けられてい

ないメッセージのみ を取り出します。 ディスパッチャーは、コンシューマーを初期化し、MQGET によって戻された MsgToken を 渡します。そうすると、コンシューマーはメッセージをキューから破壊的に取得します。 コンシューマーがメッセージの MQGET をバックアウトする場合、 そのメッセージは、もうマークが付いていないので、いずれかのブラウザーが再ディスパッチできる ようになります。 コンシューマーがメッセージに対して MQGET を実行しない 場合、MsgMarkBrowseInterval が過ぎた後、キュー・マネージャーは、 連携するハンドルのセットのためにそのメッセージのマークを解除し、そのメッセージは再ディスパッチできるようになります。

同じ ディスパッチャー・アプリケーションの複数のコピーを使用するのではなく、 多数の異なるディスパッチャー・アプリケーション (それぞれがキューにある一部のメッセージの処理に適している) が キューをブラウズするようにすることもできます。 各ディスパッチャーでは、 MQOO_CO_OP を指定してキューをオープンします。 これは、これらの複数のディスパッチャーが連携し、 互いに他のディスパッチャーのマーク付きメッセージに注意することを示します。

- 1 つのディスパッチャーのメッセージ処理の順序が重要な場合、 各ディスパッチャーは、MQGMO_BROWSE_FIRST、 MQGMO_UNMARKED_BROWSE_MSG、および MQGMO_MARK_BROWSE_HANDLE オプション (または MQGMO_BROWSE_HANDLE) を指定して、MQGET 呼び出しを繰り返します。 ブラウズされたメッセージがこのディスパッチャーが処理するのに適している場合、 MQMO_MATCH_MSG_TOKEN、MQGMO_MARK_BROWSE_CO_OP と、 前の MQGET 呼び出しで戻された MsgToken を指定して、MQGET 呼び出しを実行します。 この呼び出しが 成功した場合、ディスパッチャーはコンシューマーを初期化し、コンシューマーに MsgToken を渡します。
- メッセージ処理の順序が重要でなく、ディスパッチャーがメッセージの大部分を 処理すると予期される場合、オプション MQGMO_BROWSE_FIRST、MQGMO_UNMARKED_BROWSE_MSG、および MQGMO_MARK_BROWSE_CO_OP (または MQGMO_BROWSE_CO_OP) を使用します。 ディスパッチャーは、処理できないメッセージをブラウズした場合、 オプション MQMO_MATCH_MSG_TOKEN、MQGMO_UNMARK_BROWSE_CO_OP と、前に戻された MsgToken を指定して MQGET を呼び出すことによって、 そのメッセージのマークを解除します。

MQGET 呼び出しが失敗する場合

キュー内の特定の属性が、MQOPEN と MQGET 呼び出しを発行する間のコマンドで FORCE オプションを使用して変更された場合、MQGET 呼び出しは失敗し、MQRC_OBJECT_CHANGED 理由コードが戻されます。

キュー・マネージャーは、オブジェクト・ハンドルを無効としてマーク付けします。 キュー名が解決された結果のキューにその変更が適用される場合にも同じことが起こります。 このようにハンドルに影響を及ぼす属性は、MQOPEN の MQOPEN 呼び出しの節で一覧表示されています。 呼び出しが理由コード MQRC_OBJECT_CHANGED を戻す場合は、キューをクローズして、キューを再度オープンしてから、メッセージの読み取りを再び行います。

メッセージを読み取ろうとするキュー (または、キュー名が解決された結果のキュー) に対して読み取り走査が禁止されている場合は、MQGET 呼び出しは失敗し、理由コード MQRC_GET_INHIBITED を戻します。これは、ブラウズするために MQGET 呼び出しを使用する場合でも起こります。 アプリケーションの設計が、他のプログラムがキューの属性を規則的に変更するようになっている場合、あとで MQGET 呼び出しを試行するときに、メッセージを正常に読み取りできる場合があります。

(一時または永続) 動的キューが削除されていると、前に取得したオブジェクト・ハンドルを使用した MQGET 呼び出しは失敗し、MQRC_Q_DELETED の理由コードを戻します。

パブリッシュ/サブスクライブ・アプリケーションの作成

パブリッシュ/サブスクライブ WebSphere MQ アプリケーションの作成を開始します。

パブリッシュ/サブスクライブの概念の概要については、Introduction to WebSphere MQ publish/subscribe messaging を参照してください。

さまざまなタイプのパブリッシュ/サブスクライブ・アプリケーションの作成については、以下のトピックを参照してください。

- 278 ページの『パブリッシャー・アプリケーションの作成』
- 285 ページの『サブスクライバー・アプリケーションの作成』
- 303 ページの『パブリッシュ/サブスクライブのライフ・サイクル』

- 308 ページの『パブリッシュ/サブスクライブのメッセージ・プロパティー』
- 310 ページの『メッセージの順序付け』
- 311 ページの『パブリケーションの代行受信』
- 319 ページの『パブリッシュのオプション』
- 319 ページの『サブスクリプション・オプション』

関連概念

8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

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

96 ページの『WebSphere MQ プログラムのサンプル』
この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

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

353 ページの『クライアント・アプリケーションの作成』
WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

950 ページの『WebSphere MQ での Web サービスの使用』
IBM WebSphere MQ transport for SOAP または IBM WebSphere MQ bridge for HTTP を使用して、Web サービス用の IBM WebSphere MQ アプリケーションを開発できます。

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

553 ページの『プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

パブリッシャー・アプリケーションの作成

パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

単純な WebSphere MQ パブリッシャー・アプリケーションを作成することは、キューにメッセージを書き込む WebSphere MQ Point-to-Point アプリケーションを作成することに似ています (278 ページの表 41)。違いは、キューではなくトピックにメッセージを MQPUT することです。

表 *41. point-to-point* とパブリッシュ/サブスクライブの *WebSphere MQ* プログラム・パターン

手順	**point-to-point MQ 呼び出し**	**パブリッシュ MQ 呼び出し**
キュー・マネージャーに接続します。	MQCONN	MQCONN
キューをオープンする	MQOPEN	

表 41. point-to-point とパブリッシュ/サブスクライブの WebSphere MQ プログラム・パターン (続き)		
手順	**point-to-point MQ 呼び出し**	**パブリッシュ MQ 呼び出し**
トピックをオープンする		MQOPEN
メッセージを書き込む	MQPUT	MQPUT
トピックをクローズする		MQCLOSE
キューをクローズする	MQCLOSE	
キュー・マネージャーから切断する	MQDISC	MQDISC

具体的にするために、株価をパブリッシュする 2 つのアプリケーション 例があります。 キューへのメッセージ書き込みに似せてモデル化されている 1 つ目の例 (279 ページの『例 1: 固定トピックへのパブリッシャー』) では、管理者はキューを作成するのと同じような方法でトピック定義を 作成します。 プログラマーは、メッセージをキューではなくトピックに書き込むよう MQPUT を コーディングします。 2 つ目の例 (282 ページの『例 2: 可変トピックへのパブリッシャー』) では、プログラム と WebSphere MQ の相互作用のパターンは類似しています。 違いは、 管理者ではなくプログラマーが、メッセージが書き込まれる先のトピックを用意すること です。 実際には、これは、トピック・ストリング内容が定義されるか、 またはブラウザーからの入力などの他のソースによって提供されることを意味するのが一般的です。

関連概念

285 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする WebSphere MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

関連資料

DEFINE TOPIC

DISPLAY TOPIC

DISPLAY TPSTATUS

例 *1:* 固定トピックへのパブリッシャー

管理的に定義されたトピックへパブリッシュする方法を示すための WebSphere MQ プログラム。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

出力については、280 ページの図 38 を参照してください。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
               publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *37.* 固定トピックへの単純な *WebSphere MQ* パブリッシャー。

```
X:\Publish1\Debug>PublishStock
Optional parameters:  TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters:  TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *38. 1* つ目のパブリッシャー例からの出力サンプル

以下に選択したコード行は、WebSphere MQ 用の パブリッシャー・アプリケーション作成の各局面を示しています。

char topicNameDefault[] = "IBMSTOCKPRICE";
プログラム内でデフォルトのトピック名が定義されています。 プログラムの第 1 引数として、これと異なるトピック・オブジェクト の名前を指定すれば、これをオーバーライドできます。

MQCHAR resTopicStr[151];
resTopicStr は、td.ResObjectString.VSPtr によって ポイントされていて、解決後のトピック・ストリングを戻すために MQOPEN によって使用されます。 resTopicStr の長さ を td.ResObjectString.VSBufSize に渡される長さより 1 だけ長く して、ヌル終了のためのスペースを与えるようにします。

memset (resTopicStr, 0, sizeof(resTopicStr));
resTopicStr をヌルに初期設定して、 MQCHARV に戻される解決後のトピック・ストリングが 確実にヌル終了になるようにします。

td.ObjectType = MQOT_TOPIC
パブリッシュ/サブスクライブ用の新しいタイプのオブジェクト、トピック・オブジェクト があります。

td.Version = MQOD_VERSION_4;
新タイプのオブジェクトを使用するためには、バージョン *4* 以上のオブジェクト記述子を使用する必要があります。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
topicName は、トピック・オブジェクトの名前であり、 管理トピック・オブジェクトと呼ばれることもあります。 例では、 WebSphere MQ エクスプローラーまたは次の MQSC コマンドを使用して、トピック・オブジェクトは前もって作成されている 必要があります。

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
解決後のトピック・ストリングは、プログラム内の最後の printf でエコー出力 されます。 解決後のストリングを WebSphere MQ がプログラムに戻すための MQCHARV ResObjectString 構造体 をセットアップします。

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
出力用にトピックをオープンします。出力用にキューをオープンするのと同様です。

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
新規サブスクライバーがパブリケーションを受け取ることができるように したい場合、パブリッシャーに MQPMO_RETAIN を指定すると、 サブスクライバーを開始したときに、そのサブスクライバーは、開始前にパブリッシュされた 最後のパブリケーションを、最初に一致するパブリケーションとして受け取ります。 別の選択肢は、サブスクライバーが開始された後でのみパブリッシュされたパブリケーションを サブスクライバーに提供することです。 さらに、サブスクライバーには、 サブスクリプションに MQSO_NEW_PUBLICATIONS_ONLY を指定することで保存パブリケーションの受け取りを 拒否するというオプションもあります。

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
MQPUT に渡されるストリングの長さに 1 を加算 することで、メッセージ・バッファーの一部として WebSphere MQ にヌル終了 文字を渡します。

この例 1 は何を説明しているのでしょうか? この 例は、point-to-point WebSphere MQ プログラム作成についての、試行および試験済みの 従来のパターンにできるだけ似せて作成されています。 この WebSphere MQ プログラミング・パターンの 重要なフィーチャーの 1 つは、どこにメッセージが送信されるのかをプログラマーが 知らなくていいということです。 プログラマーの作業は、 キュー・マネージャーに接続し、受信者に配布されるメッセージをそのキュー・マネージャーに 渡すことです。 point-to-point パラダイムでは、 プログラマーは、管理者が構成したキュー (おそらく別名キュー) をオープン します。 別名キューは、メッセージを、ローカル・キュー・マネージャー またはリモート・キュー・マネージャーのターゲット・キューに経路指定します。 メッセージは、送信されるのを待つ間、ソースと宛先の間の任意の場所にあるキューに保管されます。

パブリッシュ/サブスクライブのパターンでは、 キューをオープンする代わりに、プログラマーはトピックをオープンします。 この例では、トピックは、管理者によってトピック・ストリングに 関連付けられま

す。 キュー・マネージャーは、パブリケーションを、 パブリケーションのトピック・ストリングと一致するサブスクリプションを持つ、 ローカルまたはリモートのサブスクライバーに、キューを使用して転送します。 保存パブリケーションの 場合、キュー・マネージャーは、現在はサブスクライバーを持っていなくても、 パブリケーションの最新コピーを保持します。 保存パブリケーションは、将来のサブスクライバーに 転送するために使用可能です。 パブリッシャー・アプリケーションは、パブリケーションを宛先に転送または選択するのに関与しません。 パブリッシャー・アプリケーションのタスクは、パブリケーションを作成して、管理者が定義したトピック に書き込むことです。

この固定トピックの例は、静的であり、 多くのパブリッシュ/サブスクライブ・アプリケーションの典型例ではありません。 この 例では、管理者がトピック・ストリングを定義し、パブリッシュが行われるトピックを変更する 必要があります。 通常、パブリッシュ/サブスクライブ・アプリケーションは、 トピック・ツリーの一部または全体のことを認識している必要があります。 トピックは頻繁に 変わることが多く、たとえそれほど変わらなくても、トピック組み合わせの数 は大きくなります。また、パブリッシュが行われる必要があるかもしれないトピック・ストリングの それぞれについてトピック・ノードを管理者が定義するのは、あまりにも 面倒です。 トピック・ストリングはパブリケーションより前には不明であることが多く、 パブリッシャー・アプリケーションは、パブリケーション内容からの情報を 使用してトピック・ストリングを指定したり、パブリッシュを行うトピック・ストリング についての情報を、ブラウザーからの入力など他のソースから利用したりします。 もっとダイナミックな スタイルでのパブリッシュを示すため、次の例では、パブリッシャー・アプリケーションの一部として、 トピックを動的に作成する方法を示します。

トピックは、 パブリッシャーとサブスクライバーを結合します。 トピックの命名およびトピック・ツリー内でのトピックの 編成に関する規則または体系の設計は、パブリッシュ/サブスクライブ・ソリューション開発 の重要なステップです。 トピック・ツリー編成のどの範囲までが パブリッシャー・プログラムとサブスクライバー・プログラムを結合し、 それらのプログラムをトピック・ツリーの内容に結合するのかを、注意深く見る必要があります。 トピック・ツリーを変更すると、パブリッシャー・アプリケーションおよび サブスクライバー・アプリケーションに影響があるかどうか、その 影響をどうすれば最小化できるかをよく検討してください。 WebSphere MQ パブリッシュ/サブスクライブ・モデル の体系に組み込まれているのは、 トピックのルート部分またはルート・サブツリーを提供する、管理トピック・オブジェクトの概念です。 トピック・オブジェクトによって、トピック・ツリーのルート部分を管理的に定義するオプションが 提供されます。これによって、アプリケーション・プログラミングおよび操作が単純化され、その結果として保守容易性 が向上します。 例えば、 複数のパブリッシュ/サブスクライブ・アプリケーションをデプロイ しようとしていて、それらのアプリケーションには、分離した複数のトピック・ツリーがある場合、 トピック・ツリーのルート部分を管理的に定義することによって、異なるアプリケーション間でトピック命名規則に一貫性がない場合でも、 各トピック・ツリーの分離を保証することができます。

実際には、パブリッシャー・アプリケーション は、この例のような固定トピックだけの使用から、 次の例のような可変トピックの使用まで、広い範囲をカバーします。 282 ページの『例 2: 可変トピックへのパブリッシャー』 は、 トピックおよびトピック・ストリングの使用の結合も例示しています。

関連概念

282 ページの『例 2: 可変トピックへのパブリッシャー』
プログラムで定義されたトピックへパブリッシュする方法を示す WebSphere MQ プログラム。

285 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする WebSphere MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

例 2: 可変トピックへのパブリッシャー

プログラムで定義されたトピックへパブリッシュする方法を示す WebSphere MQ プログラム。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

出力については、284 ページの図 40 を参照してください。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication,
topicName, topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *39.* 可変トピックへの単純な *WebSphere MQ* パブリッシャー。

```
X:\Publish2\Debug>PublishStock
Provide parameters:  TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters:  TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *40. 2* 番目のパブリッシャー例からの出力サンプル

この例について注意する点がいくつかあります。

char topicNameDefault[] = "STOCKS";
デフォルトのトピック名 STOCKS は、 トピック・ストリングの一部を定義します。 プログラムの第 1 引数として指定することで、このトピック名をオーバーライドできます。あるいは、 最初のパラメーターとして / を指定すれば、このトピック名の使用を除去することができます。

char topicString[101] = "IBM/PRICE";
IBM/PRICE は、デフォルトのトピック・ストリングです。 プログラム の第 2 引数として指定することで、このトピック・ストリングをオーバーライド できます。

キュー・マネージャーは、 STOCKS トピック・オブジェクト "NYSE"によって提供されるトピック・ストリングを、プログラム "IBM/PRICE" によって提供されるトピック・ストリングと結合し、2 つのトピック・ストリングの間に "/" を挿入します。 結果は、解決されたトピック・ストリング "NYSE/IBM/PRICE"になります。 結果のこのトピック・ストリングは、IBMSTOCKPRICE トピック・オブジェクト内に定義されているものと 同じであり、効果もまったく同じです。

解決後のトピック・ストリングと関連付けられた管理トピック・オブジェクト は、パブリッシャーによって MQOPEN に渡されるトピック・オブジェクト と必ずしも同じでなくてもかまいません。 WebSphere MQ は、解決後のトピック・ストリング中のツリー暗黙指定を 使用して、パブリケーションと関連付けられた属性を定義するのはどの管理トピック・オブジェクトなのか を解明します。

2 つのトピック・オブジェクト A と B があり、 A がトピック "a"を定義し、 B がトピック "a/b" (285 ページの図 41) を定義するとします。 パブリッシャー・プログラムがトピック・オブジェクト A を参照し、トピック・ストリング "b"を提供し、トピックをトピック・ストリング "a/b"に解決する場合、トピックが B に定義されているトピック・ストリング "a/b" と一致するため、パブリケーションはトピック・オブジェクト B からプロパティーを継承します。

if (strcmp(argv[1],"/"))
argv[1] はオプションで指定される topicName です。 "/" は トピック名としては無効であり、ここでは、トピック名がなく、 トピック・ストリング全体がプログラムによって用意されることを表します。 284 ページの図 40 の 出力では、トピック・ストリング全体がプログラムによって動的に提供されることが示されています。

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
デフォルトのケースでは、WebSphere MQ エクスプローラーまたは次のコマンドを使用して、オプションの topicName が前もって 作成されている必要があります。

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
トピック・ストリングは、トピック記述子内の MQCHARV フィールドです。

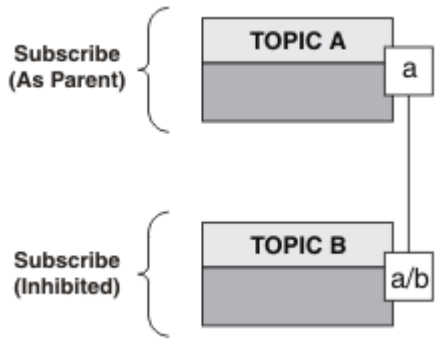


図 *41.* トピック・オブジェクトの関連

この例 2 は何を説明しているのでしょうか? コードは例 1 とよく似ています。 実際、違うのは 2 行のみですが、結果的には例 1 とは大きく異なるプログラムになっています。 プログラマーは、パブリケーションが送信される宛先を制御します。 サブスクライバー・アプリケーションの設計に使用される管理者入力が最小限になると共に、パブリッシャーからサブスクライバーにパブリケーションを転送するために、トピックまたはキューが事前定義されている必要はありません。

point-to-point メッセージング・パラダイム では、メッセージが流れるには、その前にキューが定義されている必要があります。 パブリッシュ/サブスクライブの場合は、基礎にキューイング・システムを使用して WebSphere MQ がパブリッシュ/サブスクライブをインプリメントしていても、そうではありません。メッセージングと キューイングに関する、配信が保証されること、トランザクション化が可能であること、疎結合といった利点は、パブリッシュ/サブスクライブ・アプリケーション に継承されます。

設計者は、 パブリッシャー、サブスクライバー、プログラムが、基礎にあるトピック・ツリーに ついて認識する必要があるかどうか、また、サブスクライバー・プログラムが キューイングについて認識しているかどうかを決定しなければなりません。 次のサブスクライバー・アプリケーション例を 検討してください。 これらの例は、パブリッシャー例と一緒に使用されるように 設計されていて、通常は `NYSE/IBM/PRICE` にパブリッシュし、サブスクライブします。

関連概念

279 ページの『例 1: 固定トピックへのパブリッシャー』
管理的に定義されたトピックへパブリッシュする方法を示すための WebSphere MQ プログラム。

285 ページの『サブスクライバー・アプリケーションの作成』
次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする WebSphere MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

サブスクライバー・アプリケーションの作成

次の 3 つの例を考察して、サブスクライバー・アプリケーションの作成を開始します。1 つは、キューからメッセージをコンシュームする WebSphere MQ アプリケーション、もう 1 つは、キューに関する知識を何も必要としない、サブスクリプションを作成するアプリケーション、最後の 1 つは、キューイングとサブスクリプションの両方を使用する例です。

286 ページの表 42 に、 コンシューマーまたはサブスクライバーの 3 つのスタイルを、それらの特色を示している WebSphere MQ 機能呼び出しのシーケンスと共にリストします。

1. 最初のスタイルである MQ パブリケーション・コンシューマーは、 MQGET のみを実行する point-to-point MQ プログラムと同じです。 このアプリケーションには、 パブリケーションをコンシュームしているという認識はなく、単にキューから メッセージを読み取ります。 キューへのパブリケーションの転送を引き起こすサブスクリプション は、WebSphere MQ エクスプローラーまたはコマンドを使用して、管理的に作成されます。

2. 2 つ目のスタイルは、大部分のサブスクライバー・アプリケーションが優先的に使用するパターン です。 このサブスクライバー・アプリケーションは、サブスクリプションを作成し、その後で パブリケーションを取得します。 キュー管理はすべてキュー・マネージャーによって 実行されます。
3. 3 つ目のスタイルでは、サブスクライバー・アプリケーションは、 パブリケーションに使用されるキューのオープンとクローズを行うことと、サブスクリプションを発行することを選択して、キューをパブリケーションで満たします。

これらのスタイルを理解する 1 つの方法は、スタイルごとに 286 ページの表 42 にリストされている C プログラムの例を検討することです。 これらの例は、 278 ページの『パブリッシャー・アプリケーションの作成』 にあるパブリッシャー例と連動して実行できるよう設計されています。

表 *42. point-to-point vs.* サブスクライブ *WebSphere MQ* プログラム・パターン

手順	MQ メッセージ・コンシューマー	286 ページの『例 1: MQ パブリケーション・コンシューマー』	289 ページの『例 2: 管理対象 MQ サブスクライバー』	294 ページの『例 3: 非管理 MQ サブスクライバー』
キュー・マネージャーに接続します。	MQCONN	MQCONN	MQCONN	MQCONN
キューをオープンする	MQOPEN	MQOPEN		MQOPEN
サブスクライブ			MQSUB	MQSUB
メッセージを読み取る	MQGET	MQGET	MQGET	MQGET
キューをクローズする	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
サブスクリプションをクローズする			MQCLOSE	MQCLOSE
キュー・マネージャーから切断する	MQDISC	MQDISC	MQDISC	MQDISC

MQCLOSE の使用は常にオプションであり、 リソースを解放するためか、MQCLOSE オプションを渡すためか、または単に MQOPEN との対称性のためです。 管理 MQ サブスクライバーのケースではサブスクリプション・キューがクローズされるときに MQCLOSE オプションを指定する必要はなく、また、対称性の議論は関係ないため、例 2: 管理 MQ サブスクライバーでは、サブスクリプション・キューは明示的にはクローズされません。

パブリッシュ/サブスクライブ・アプリケーションのパターンを理解する別の方法 は、関係するさまざまなエンティティー間の相互作用に注目することです。 ライフラインまたは UML シーケンス図は、相互作用を学習するためのいい方法です。 303 ページの『パブリッシュ/サブスクライブのライフ・サイクル』 に、3 つのライフライン例が記述されています。

例 *1: MQ* パブリケーション・コンシューマー

MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM WebSphere MQ メッセージ・コンシューマーです。

この例のためのサブスクリプションおよびパブリケーションのキューを作成 するため、以下のコマンドを実行するか、WebSphere MQ エクスプローラーを使用してオブジェクトを定義します。

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

IBMSTOCKPRICESUB サブスクリプションは、当パブリッシャー例のために作成された IBMSTOCK トピック・オブジェクト と、ローカル・キュー STOCKTICKER を参照します。 トピック・オブジェクト IBMSTOCK は、 サブスクリプション内で使用されるトピック・ストリング、NYSE/IBM/PRICE を定義します。 パブ

リケーションを受け取るのに使用されるトピック・オブジェクトおよびキュー は、サブスクリプションが作成される前に定義される必要があることに注意してください。

この MQ パブリケーション・コンシューマー・パターンには、以下のように多くの重要な面があります。

1. マルチプロセッシング: パブリケーション読み取りの作業が分配されます。 すべてのパブリケーションが、サブスクリプション・トピックと関連付けられた単一のキューに入れられ ます。 複数のコンシューマーが、MQOO_INPUT_SHARED を使用してそのキューをオープンできます。
2. 集中管理されるサブスクリプション。 アプリケーションは、独自のサブスクリプション・トピック またはサブスクリプションを構成しません。パブリケーションがどこに送信されるのかについては、管理者が担当します。
3. サブスクリプション集中: 複数の異なるサブスクリプションを単一のキューに 送信できます。
4. サブスクリプション永続性: キューは、コンシューマーがアクティブかどうかに関係なく、すべての パブリケーションを受け取ります。
5. マイグレーションおよび共存: このコンシューマーのコードは、point-to-point のシナリオでもパブリッシュ/サブスクライブのシナリオでも 同じように機能します。

サブスクリプションは、トピック・ストリング NYSE/IBM/PRICE と キュー STOCKTICKER との間に関係を作成します。 パブリケーションは、現在保持されているパブリケーションも 含めて、サブスクリプションが作成された時点から STOCKTICKER に転送 されます。

管理的に作成されたサブスクリプションは、管理であることも非管理であることも可能です。 管理サブスクリプションは、作成されたらすぐに有効になり、 それは非管理サブスクリプションでも同様です。 上記に示した当パターンのすべての面が管理サブスクリプションに 当てはまるわけではありません。 294 ページの『例 3: 非管理 MQ サブスクライバー』を参照してください。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

288 ページの図 43 では結果が示されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    MQCHAR      publicationBuffer[101];
    MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
    MQCHAR48 qmName = "";                       /* Use default queue manager    */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN;       /* connection handle            */
    MQHOBJ   Hobj = MQHO_NONE;                  /* object handle sub queue      */
    MQLONG   CompCode = MQCC_OK;                /* completion code              */
    MQLONG   Reason = MQRC_NONE;                /* reason code                  */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};               /* Unmanaged subscription queue */
    MQMD     md = {MQMD_DEFAULT};               /* Message Descriptor           */
    MQGMO    gmo = {MQGMO_DEFAULT};             /* Get message options          */
    char *   publication=publicationBuffer;
    char *   subscriptionQueue = subscriptionQueueDefault;

    switch(argc){                     /* Replace defaults with args if provided */
        default:
            subscriptionQueue = argv[1]
        case(1):
            printf("Optional parameter: subscriptionQueue\n");
    }

    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
        MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publication, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
                  &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

図 *42. MQ* パブリケーション・コンシューマー

```
X:\Subscribe1\Debug>Subscribe1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

図 *43. MQ* パブリケーション・コンシューマーからの出力

標準 WebSphere MQ *C* 言語プログラミングに関するいくつかの ヒントに留意してください。

memset(publication, 0, sizeof(publicationBuffer));
printf を使用するフォーマット設定を簡単にするため、メッセージに確実に末尾ヌルが入るようにします。 このパブリッシャー例では、 strlen(publication) に 1 を加算することによって、 MQPUT に渡されるメッセージ・バッファーに末尾ヌルを 組み込んでいます。 MQCHAR バッファーをヌルに設定することは、それらのバッファーを使用してストリングを保管する IBM WebSphere MQ C プログラムのお勧めのプログラミング・スタイルです。こうすることで、バッファーを完全には満たさない文字配列の後にヌルが続くことが確実になります。

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
メッセージ・バッファーの末尾にヌルを 1 つ予約して、 "if (messlen == strlen(publication));" が true の場合に、返されたメッセージの末尾にヌルが含まれるようにします。 このヒントは、 上記のヒントを補完するものであり、publication の内容で置き換えられない、最低でも 1 つのヌルが publicationBuffer 内に あることを確実にします。

関連概念

289 ページの『例 2: 管理対象 MQ サブスクライバー』
管理対象 MQ サブスクライバーは、ほとんどのサブスクライバー・アプリケーションで推奨されるパターンです。 この例では、キュー、 トピック 、または サブスクリプションの管理定義は 必要ありません 。

294 ページの『例 3: 非管理 MQ サブスクライバー』
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

278 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

例 *2:* 管理対象 *MQ* サブスクライバー

管理対象 MQ サブスクライバーは、ほとんどのサブスクライバー・アプリケーションで推奨されるパターンです。 この例では、キュー、 トピック 、または サブスクリプションの管理定義は 必要ありません 。

この最も単純な 種類 の 管理サブスクライバーは、通常、 非永続 サブスクリプションを使用します。 例では、非永続サブスクリプション にフォーカスを合わせます。 サブスクリプションは、 MQSUB からのサブスクリプション・ハンドルの存続期間 のみ 存続します。 サブスクリプションの存続期間中にトピック・ストリングに一致する すべての パブリケーションがサブスクリプション・キューに送信されます (フラグ MQSO_NEW_PUBLICATIONS_ONLY が設定されていないか、デフォルト設定されていない場合、トピック・ストリングに一致する以前のパブリケーションが保持されている場合、パブリケーションが作成されてからキュー・マネージャーが終了していない場合は、保存パブリケーションも送信されます)。

このパターンで永続 サブスクリプションを使用することもできます。 通常、管理対象永続サブスクリプションが使用される場合は、エラーが発生することなくサブスクライバーよりも存続するサブスクリプションを確立するためではなく、信頼性の理由で行われます。 管理対象、非管理対象、 永続 、および 非永続 サブスクリプションに関連するさまざまなライフサイクルについて詳しくは、関連トピックのセクションを参照してください。

永続サブスクリプションは永続パブリケーションと、 非永続サブスクリプションは非永続パブリケーションと関連付けられることが多い ことは確かですが、サブスクリプションの永続性とパブリケーションの永続性との 間には必須の関係はありません。 永続性の 4 通りの組み合わせのすべてが 可能です。

ここで検討している管理非永続のケースでは、キュー・マネージャー がサブスクリプション・キューを作成し、そのキューはクローズされるときに パージされ、削除されます。 パブリケーションは、非永続サブスクリプションが クローズされるときにキューから除去されます。

このコードで例示されている、管理非永続のパターンにおける重要な面を 以下に示します。

1. 要求時サブスクリプション: サブスクリプション・トピック・ストリングは動的です。 これはアプリケーションによって実行時に供給されます。

2. 自律キュー: サブスクリプション・キューは、自己定義および自己管理を行います。
3. 自己管理サブスクリプション・ライフサイクル: 非-永続 サブスクリプションは、サブスクライバー・アプリケーションの期間中のみ存在します。
   - 永続 管理サブスクリプションを定義すると、永続サブスクリプション・キューが作成され、アクティブなサブスクライバー・プログラムがない状態でパブリケーションが引き続き保管されます。 キュー・マネージャー がキューを削除 (および、取り出されなかったパブリケーションをキューから消去) する のは、アプリケーションまたは管理者がサブスクリプションを削除することを選択した 後でのみです。 サブスクリプションの削除は、 管理コマンドを使用するか、MQCO_REMOVE_SUB オプションを指定してサブスクリプションをクローズすることで実行できます。
   - 永続サブスクリプションの 設定 SubExpiry を検討してください。これにより、パブリケーションがキューに送信されなくなり、サブスクライバーがサブスクリプションを削除する前に残りのパブリケーションをコンシュームできるようになり、キュー・マネージャーがキューとそのキューに残っているパブリケーションを削除するようになります。
4. トピック・ストリングの柔軟なデプロイメント: 管理的に定義されるトピックを使用して サブスクリプションのルート部分を定義することによって、サブスクリプション・トピック管理 が単純化されます。 これによって、トピック・ツリーの ルート部分がアプリケーションに対して隠蔽されます。 ルート・パートを非表示にすることにより、別のインスタンスまたは別のアプリケーションによって作成された別のトピック・ツリーと重複するトピック・ツリーをアプリケーションが誤って作成することなく、アプリケーションをデプロイすることができます。
5. 管理対象トピック: 最初の部分が管理上定義されたトピック・オブジェクトと一致するトピック・ストリングを使用すると、トピック オブジェクトの属性に従ってパブリケーションが管理されます。
   - の例では、トピック・ストリングの最初の部分がクラスター・トピック・オブジェクトに関連付けられたトピック・ストリングと一致する場合、サブスクリプションはクラスターの他のメンバーからパブリケーションを受け取ることができます。
   - 管理的に定義されたトピック・オブジェクトとプログラムで 定義されたサブスクリプションを選択的にマッチングすることで、 両方の利点を結合することができます。 管理者はトピックの属性を提供し、プログラマーはトピックの管理を気にすることなく「sub-topics」を動的に定義します。
   - これは、 トピックに関連付けられた属性を提供するトピック・オブジェクトと突き合わせるために使用される結果のトピック・ストリングであり、通常は同じであることが分かりますが、必ずしも sd.Objectname で指定されたトピック・オブジェクトとは限りません。 282 ページの『例 2: 可変トピックへのパブリッシャー』を参照してください。

この例では、サブスクリプションを永続的なものにすることにより、サブスクライバーが MQCO_KEEP_SUB オプションを指定してサブスクリプションをクローズした後も、パブリケーションは引き続きサブスクリプション・キューに送信されます。 サブスクライバーがアクティブでなくても、キューはパブリケーションを 受け取り続けます。 この動作をオーバーライドするには、MQSO_PUBLICATIONS_ON_REQUEST オプションを指定してサブスクリプションを 作成し、MQSUBRQ を使用して、保存パブリケーションを要求します。

MQCO_RESUME オプションを指定してサブスクリプションを オープンすることによって、後でサブスクリプションを再開できます。

MQSUB によって戻される キュー・ハンドル Hobj には多くの用途があります。 例では、キュー・ハンドルは、サブスクリプション・キューの名前を照会するのに 使用されています。 管理キューは、デフォルトのモデル・キュー SYSTEM.NDURABLE.MODEL.QUEUE または SYSTEM.DURABLE.MODEL.QUEUE を使用してオープンされます。 サブスクリプションに関連付けられたトピック・オブジェクトのプロパティーとして、トピックごとに独自の永続モデル・キューおよび非永続モデル・キューを提供することにより、 デフォルトをオーバーライドできます。

モデル・キューから継承する属性に関係なく、 追加のサブスクリプションを作成するために管理キュー・ハンドルを再使用することはできません。 また、管理キュー用の別のハンドルを、戻されたキュー名を使用して管理キューを 2 度目にオープンすることによって 取得することもできません。 キューは、排他的 入力のためにオープンされたかのように動作します。

非管理キューのほうが、管理キューよりも柔軟性があります。 例えば、 非管理キューを共用することや、複数のサブスクリプションを 1 つのキューに 定義することができます。 次の例である 294 ページの『例

3: 非管理 MQ サブスクライバー』 に、 サブスクリプションを非管理サブスクリプション・キューと結合する方法が示されています。

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

---

292 ページの図 46 では結果が示されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault     = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = "";                 /* Use default queue manager     */
    MQCHAR48 qName = "";                  /* Allocate to query queue name  */
    char     publicationBuffer[101];      /* Allocate to receive messages  */
    char     resTopicStrBuffer[151];   /* Allocate to resolve topic string */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle             */
    MQHOBJ   Hobj = MQHO_NONE;            /* publication queue handle      */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle           */
    MQLONG   CompCode = MQCC_OK;          /* completion code               */
    MQLONG   Reason = MQRC_NONE;          /* reason code                   */
    MQLONG   messlen = 0;
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor       */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor            */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options           */

    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

    switch(argc){               /* Replace defaults with args if provided */
        default:
          topicString = argv[2];
        case(2):
          if (strcmp(argv[1],"/"))       /* "/" invalid = No topic object  */
             topicName = argv[1];
          else
             *topicName = '\0';
        case(1):
             printf("Optional parameters: topicName, topicString\nValues \"%s\"  \"%s\"\n",
                   topicName, topicString);
    }
```

図 *44.* 管理対象 *MQ* サブスクライバー-パート *1:* 宣言およびパラメーター処理。

以下の説明は、この例の宣言に関する追加コメントです。

MQHOBJ Hobj = MQHO_NONE;
パブリケーションを受け取るために非永続管理サブスクリプション・キューを明示的に開くことはできませんが、キュー・マネージャーがキューを開くときに返すオブジェクト・ハンドル用のストレージを割り振る必要があります。 MQHO_OBJECT に対するハンドルを初期化することが重要です。 これ は、キュー・ハンドルをサブスクリプション・キューに戻す必要があることをキュー・マネージャーに示します。

MQSD sd = {MQSD_DEFAULT};
MQSUB で使用される、新規サブスクリプション記述子。

MQCHAR48 qName;
例 はサブスクリプション・キューの知識を必要と n しますが、この例ではサブスクリプション・キューの名前を照会します。 MQINQ バインディングは C 言語では少し扱いにくいので、例のこの部分を参考にしてください。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING ;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from \"%-0.48s\"\n",
               gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publicationBuffer, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer-1),
                publication, &messlen, &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
    return;
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs];                /* Array of integer attributes  */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
          &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}
```

図 *45.* 管理対象 *MQ* サブスクライバー-パート *2:* コード本体。

```
W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS"  "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403300020        "
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values ""  "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from
"SYSTEM.MANAGED.NDURABLE.48A0AC7403310020        "
Received publication "150"
Completion code 0 and Return code 0
```

図 *46.* 管理対象 *MQ* サブスクライバーからの出力

以下の説明は、この例のコードについての追加コメントです。

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
topicName がヌルまたはブランク (デフォルト値) の 場合、解決されたトピック・ストリングを計算するのにトピック名は使用されません。

sd.ObjectString.VSPtr = topicString;
事前定義されたトピック・オブジェクトを単独で使用するのではなく、 この例では、プログラマーは MQSUB によって結合された トピック・オブジェクトとトピック・ストリングを供給しています。 このトピック・ストリングは MQCHARV 構造体であることに注意してください。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQCHARV フィールドの長さを設定する代替方法。

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
トピック・ストリングの定義後、sd.Options フラグは細心の注意 を必要とします。 多くのオプションがあります。 この例では、最もよく使用されるオプションのみを指定しています。その他のオプションは のデフォルトのままです。

1. サブスクリプションは 非永続であるため、つまり、アプリケーション内のオープン・サブスクリプションの存続時間があるため、 MQSO_CREATE フラグを設定します。 読みやすくするために、 (デフォルト) MQSO_NON_DURABLE フラグを設定することもできます。
2. MQSO_CREATE を補完するのは、MQSO_RESUME です。 両方のフラグを一緒に設定することができます。キュー・マネージャーは、 新規 サブスクリプションを作成するか、既存のサブスクリプションを再開するか、どちらか該当する方を行います。 ただし、MQSO_RESUME を 指定する場合は、再開するサブスクリプションがない場合でも、sd.SubName の MQCHARV 構造体の初期設定も 行う必要があります。 SubName の初期設定が 失敗すると、戻りコード 2440: MQRC_SUB_NAME_ERROR が MQSUB から出されます。

   注: MQSO_RESUME は、 非永続管理サブスクリプションの場合は常に無視されます。しかし、 これを指定して、sd.SubName の MQCHARV 構造体を初期設定しないと、 エラーの原因になります。
3. さらに、サブスクリプションがどのようにオープンされるのかに影響する 3 番目の フラグ、MQSO_ALTER があります。 正しい許可が付与されている場合は、再開されたサブスクリプションのプロパティーは、 MQSUB に指定された他の属性と一致するように変更されます。

   注: MQSO_CREATE、MQSO_RESUME、および MQSO_ALTER のうち少なくとも 1 つの フラグが指定されていなければなりません。 Options (MQLONG) を参照してください。 294 ページの『例 3: 非管理 MQ サブスクライバー』 に、これら 3 つのフラグのすべてを使用する例があります。
4. キュー・マネージャーが自動的にサブスクリプションを管理するよう、MQSO_MANAGED を 設定します。

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
オプションで、ヌル終了ストリング用の MQCHARV 長さの 設定を省略して、代わりにヌル終了文字フラグを使用します。

sd.ResObjectString.VSPtr = resTopicStr;
結果のトピック・ストリングは、プログラム内の最初の printf で エコー出力されます。 WebSphere MQ が解決後のストリングをプログラムに戻すための MQCHARV ResObjectString を セットアップします。

注: resTopicStringBuffer は memset(resTopicStr, 0, sizeof(resTopicStrBuffer)) でヌルに初期設定されています。 戻されるトピック・ストリングは、末尾ヌルで終了しません。

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
sd.ResObjectString のバッファー・サイズを、実際の サイズより 1 だけ小さい値に設定します。この は、解決されたトピック・ストリングがバッファー全体を満杯にした場合に、提供された NULL 終止符を上書きしないようにします。

注: トピック・ストリングが sizeof(resTopicStrBuffer)-1 より 長い場合、エラーは戻されません。 VSLength > VSBufSiz の場合でも、 sd.ResObjectString.VSLength に戻される長さは完全なストリングの長さであり、必ずしも戻されるストリングの長さではありません。

sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz をテストして、トピック・ストリングが完全であることを確認します。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQSUB 機能は、サブスクリプションを作成します。 非永続 の場合、その名前には関心はないでしょうが、WebSphere MQ エクスプローラーで状況を検査することができます。 sd.SubName パラメーターを input として指定すると、検索する名前が分かります。当然、他のサブスクリプションとの名前の競合を避ける必要があります。

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
サブスクリプションとサブスクリプション・キューの両方のクローズは、オプションです。 この例では、サブスクリプションはクローズされますが、キューはクローズされません。 MQCLOSE MQCO_REMOVE_SUB オプション は、このケースではサブスクリプションは非永続であるため、デフォルトです。 MQCO_KEEP_SUB の使用はエラーです。

注 : サブスクリプション・キュー は MQSUB ではクローズされず、ハンドル Hobj は、MQCLOSE または MQDISC でキューがクローズされるまでは有効です。 アプリケーションが途中で終了してしまった場合、キューおよびサブスクリプション は、アプリケーションの終了から少し後に、キュー・マネージャーによってクリーンアップされます。

関連概念
286 ページの『例 1: MQ パブリケーション・コンシューマー』
MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM WebSphere MQ メッセージ・コンシューマーです。

294 ページの『例 3: 非管理 MQ サブスクライバー』
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

278 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

例 *3:* 非管理 *MQ* サブスクライバー
非管理サブスクライバーは、サブスクライバー・アプリケーションの 重要なクラスの 1 つです。 これを使用して、パブリッシュ/サブスクライブの利点 を、パブリケーションのキューイングとコンシュームの制御と結合することができます。 この例 では、サブスクリプションとキューを結合するさまざまな方法を示します。

非管理パターンは、非永続 サブスクリプションよりも 永続 サブスクリプションに関連付けられるほうが一般的です。 通常、非管理サブスクライバーによって 作成されるサブスクリプションのライフ・サイクルは、 サブスクライブを行うアプリケーション自体のライフ・サイクルとは無関係です。 サブスクリプションを 永続的にすることによって、サブスクリプションは、サブスクライブを行うアプリケーションがアクティブ でない場合でも、パブリケーションを受け取ります。

永続管理 サブスクリプションを作成しても同じ結果を 得ることができますが、アプリケーションによっては、キューおよびメッセージに関する制御と柔軟性を、 管理サブスクリプションの場合よりも必要とします。 永続管理サブスクリプションの場合、キュー・マネージャーは、サブスクリプション・トピックに一致するパブリケーション 用に永続キューを作成します。 そのキューおよび関連付けられたパブリケーションは、 サブスクリプションが削除されるときにキュー・マネージャーによって削除されます。

永続管理 サブスクリプションが使用されるのは、 アプリケーションとサブスクリプションのライフ・サイクルが基本的に同じであるが、 それを保証するのは難しいという場合が典型的です。 サブスクリプションを永続的にし、パブリッシャーに永続パブリケーションを 作成させるようにすると、もしキュー・マネージャーまたはサブスクライバーが途中で 終了してリカバリーが必要になっても、失われるメッセージはありません。

キュー・マネージャーは、サブスクライバー用の永続管理サブスクリプション・キューを、 キューの共用処理は可能でないような方法で、暗黙的にオープン します。 また、各管理キューに対して複数のサブスクリプションを 作成することはできず、キューの名前に対する制御を十分に持っていないため、キューの 管理は容易ではありません。 以上の理由から、非管理 MQ サブスクライバー が管理 MQ サブスクライバーに比べて、 永続サブスクリプションを必要とするアプリケーションに適しているかどうかを検討してください。

300 ページの図 49 の コードは、非管理永続サブスクリプションのパターンを示しています。 例示の目的のため、このコードは非管理の非永続サブスクリプションも 作成します。 この例は、次のパターン・ファセットを 示しています。

- オンデマンドのサブスクリプション: サブスクリプション・トピック・ストリングは動的です。 これらはアプリケーションによって実行時に提供されます。
- 単純化されたサブスクリプション・トピック管理: 管理的に定義されるトピックを使用して、 サブスクリプション・トピック・ストリングのルート部分を定義することによって、 サブスクリプション・トピック管理が単純化されます。 これは、トピック・ツリーのルート部分を アプリケーションに対して隠します。 ルート部分を隠すことによって、 1 つのサブスクライバーを複数の異なるトピック・ツリーにデプロイできます。
- 柔軟なサブスクリプション管理: サブスクリプションを管理的に定義するか、 サブスクライバー・プログラム内でオンデマンドで作成することができます。 管理的に作成されたサブスクリプションとプログラムで作成されたサブスクリプションとの間には、 どのようにサブスクリプションが作成されたのかを示す属性を除いて、 違いはありません。 第 3 のタイプのサブスクリプションがあり、それは、サブスクリプションの配布のためにキュー・マネージャーによって自動的に 作成されるものです。 すべてのサブスクリプションは、WebSphere MQ エクスプローラーで表示されます。
- サブスクリプションとキューの柔軟な関連付け: MQSUB 機能によって、事前定義された ローカル・キューがサブスクリプションと関連付けられます。 サブスクリプションとキューを関連付けるための MQSUB の使用には、次のように いくつかの方法があります。
  - サブスクリプションを、既存のサブスクリプションが ない キュー (MQSO_CREATE + (Hobj from MQOPEN)) に関連付けます。
  - 新規 サブスクリプションを、既存のサブスクリプション MQSO_CREATE + (Hobj from MQOPEN) を持つキューに関連付けます。
  - 既存のサブスクリプションを別のキュー MQSO_ALTER + (Hobj from MQOPEN) に移動します。
  - 既存のキュー、MQSO_RESUME + (Hobj = MQHO_NONE)、または MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription) に関連付けられている既存のサブスクリプションを再開します。
  - MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の 組み合わせ方を変えることによって、異なる値の sd.Options を使用する複数 バージョンの MQSUB をコーディングせずに、さまざまな異なる入力状態の サブスクリプションとキューに対応できます。
  - あるいは、 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の特定の選択をコーディングすることにより、 MQSUB への入力として提供されたサブスクリプションとキューの状態が sd.Options の値と矛盾する場合、キュー・マネージャーはエラー (297 ページの表 43) を返します。 303 ページの図 55 sd.Options フラグの個々の設定が異なるサブスクリプション X に対して MQSUB を発行し、3 つの異なるオブジェクト・ハンドルを渡した結果を示しています。

  299 ページの図 48 に示されている プログラム例へのさまざまな入力を検討して、これらの異なる種類のエラーを理解してください。 表にリストされているケースに含まれていない 1 つの一般的なエラー RC = 2440 は、サブスクリプション名エラーです。 これが発生するのは、通常、ヌルまたは 無効なサブスクリプション名を MQSO_RESUME または MQSO_ALTER で渡すことが原因です。
- マルチプロセッシング: パブリケーションを読み取る作業を多くのコンシューマーと共有できます。 すべてのパブリケーションが、サブスクリプション・トピックと関連付けられた単一のキューに入れられます。 コンシューマーは、MQOPEN を使用してキューを直接オープンするのか、あるいは MQSUB を使用してサブスクリプションを再開するのかを選択できます。
- サブスクリプション集中: 複数のサブスクリプションを同じキューに 作成することができます。 この機能の使用には注意が必要です。これによって、 サブスクリプションの「オーバーラップ」が発生したり、

同じパブリケーションを複数回受け取る ことになる可能性があるためです。 MQSO_GROUP_SUB オプションを指定すると、 サブスクリプションのオーバーラップによって引き起こされるパブリケーション重複はなくなります。

- サブスクライバーとコンシューマーの分離: 例で示された 3 つのコンシューマー・モデル のほかに、コンシューマーをサブスクライバーから分離 するモデルがあります。 これは非管理対象 MQ サブスクライバーのバリエーションですが、同じプログラムで MQOPEN と MQSUB を発行するのではなく、1 つのプログラムがパブリケーションにサブスクライブし、別のプログラムがそれらをコンシュームします。 例えば、サブスクライバーがパブリッシュ/サブスクライブ・クラスターの 一部であり、コンシューマーはキュー・マネージャー・クラスターの外部でキュー・マネージャーに 接続されているといった状況が考えられます。 リモート・キュー定義としてサブスクリプション・キューを 定義することによって、コンシューマーは、標準分散キューイングを通して パブリケーションを受け取ります。

これらのオプションの組み合わせを使用してコードを単純化する予定の場合は特に、 MQSO_CREATE | MQSO_RESUME | MQSO_ALTER の動作を理解することが重要です。 297 ページの表 43 を見てください。 この表には、MQSUB に異なるキュー・ハンドルを渡した場合の 結果と、301 ページの図 50 から 303 ページの図 55 に示されたプログラム例の実行結果が示されています。

表の構成に使用されるシナリオには、1 つのサブスクリプション X と 2 つのキュー A および B があります。サブスクリプション名パラメーター sd.SubName は、X に設定され、キュー A に付属するサブスクリプションの名前に設定されます。 キュー B にはサブスクリプションが付属しません。

297 ページの表 43 では、 MQSUB にサブスクリプション X とキュー・ハンドル A が渡されます。 サブスクリプション・オプションの結果は以下のとおりです。

- MQSO_CREATE が失敗するのは、キュー・ハンドルが、X へのサブスクリプションを既に持っているキュー A に対応しているためです。 この動作と正常な呼び出しを比較します。 正常な呼び出しは、キュー B がそれに接続している X へのサブスクリプションを持たないため、成功します。
- MQSO_RESUME が成功するのは、キュー・ハンドルが、X へのサブスクリプションを既に持っているキュー A に対応するためです。 対照的に、サブスクリプション X がキュー A に存在しない場合、呼び出しは失敗します。
- MQSO_ALTER は、サブスクリプションとキューのオープンに関して MQSO_RESUME と同じように動作します。 ただし、 MQSUB に渡されるサブスクリプション記述子に含まれている属性がサブスクリプションの属性と異なる場合、 MQSO_RESUME は失敗します。一方、 MQSO_ALTER は、プログラム・インスタンスが属性を変更する許可を持っている限り成功します。 サブスクリプション内のトピック・ストリングを変更することはできませんが、 MQSUB はエラーを返すのではなく、サブスクリプション記述子内のトピック名とトピック・ストリングの値を無視し、既存のサブスクリプション内の値を使用します。

次に、MQSUB がサブスクリプション X に渡され、キュー・ハンドルがキュー B に渡される場所である 297 ページの表 43 を確認します。 サブスクリプション・オプションの結果は以下のとおりです。

- MQSO_CREATE は成功し、キュー B 上にサブスクリプション X を作成します。これは、これがキュー B 上の新規サブスクリプションであるためです。
- MQSO_RESUME は失敗します。 MQSUB はキュー B 上でサブスクリプション X を探して見つかりませんが、「*RC = 2428* - サブスクリプション *X* は存在しない」ではなく、「*RC = 2019* - サブスクリプション・キューがキュー・オブジェクト・ハンドルと一致しない」を返します。 3 番目のオプション MQSO_ALTER の動作は、 この予期しないエラーの理由を示唆します。 MQSUB は、 キュー・ハンドルが、サブスクリプションを持つキューをポイントしていることを予期しています。 MQSUB は初めにこれを検査してから、sd.SubName 内で指定されているサブスクリプションが存在するかどうかを検査します。
- MQSO_ALTER は成功し、 サブスクリプションをキュー A からキュー B に移動します。

キュー A にあるサブスクリプションのサブスクリプション名が sd.SubName 内のサブスクリプション名と一致しないケースは、この表には示されていません。 この場合の 呼び出しは *RC = 2428* - サブスクリプション *X* はキュー *A* に存在しない で失敗します。

表 *43.* さまざまなキュー・ハンドルとサブスクリプションの組み合わせでの *MQSUB* からのエラー

	キュー A **サブスクリプション X** **キュー B** **サブスクリプションなし**	**キュー A** **サブスクリプションなし** **キュー B** **サブスクリプションなし**
キュー *A* の Hobj が MQSUB に渡される	**MQSO_CREATE** RC = 2432 - サブスクリプション X は既にキュー A に存在する **MQSO_RESUME** キュー A にあるサブスクリプション X を再開する **MQSO_ALTER** キュー A にあるサブスクリプション X を再開し、許可された変更を行う	**MQSO_CREATE** キュー A にサブスクリプション X を作成する **MQSO_RESUME** RC = 2428 - サブスクリプション X はキュー A に存在しない **MQSO_ALTER** RC = 2428 - サブスクリプション X はキュー A に存在しない
キュー *B* の Hobj が MQSUB に渡される	**MQSO_CREATE** キュー B に新規サブスクリプション X を作成する **MQSO_RESUME** RC = 2019 - サブスクリプション・キューがキュー・オブジェクト・ハンドルと一致しない **MQSO_ALTER** サブスクリプション X をキュー A からキュー B に移動する	**MQSO_CREATE** キュー B に新規サブスクリプション X を作成する **MQSO_RESUME** RC = 2428 - サブスクリプション X はキュー B に存在しない **MQSO_ALTER** RC = 2428 - サブスクリプション X はキュー B に存在しない
MQHO_NONE が MQSUB に渡される	**MQSO_CREATE** RC = 2019 - 不正なオブジェクト・ハンドル: 管理サブスクリプションを作成し、管理キューを作成するには、MQSO_MANAGED フラグを設定 してください **MQSO_RESUME** キュー A にあるサブスクリプション X を再開し、キュー A の Hobj を戻す **MQSO_ALTER** キュー A にあるサブスクリプション X を再開し、キュー A の Hobj を戻し、許可された変更を行う	**MQSO_CREATE** RC = 2019 - 不正なオブジェクト・ハンドル: 管理サブスクリプションを作成し、管理キューを作成するには、MQSO_MANAGED フラグを設定 してください **MQSO_RESUME** RC = 2428 - サブスクリプション X なし **MQSO_ALTER** RC = 2019 - 不正なオブジェクト・ハンドル: キュー A または B なし

注 : 実動での使用向けではなく、読みやすくするため、コンパクトなコーディング・スタイルが 採用されています。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault            = "STOCKS";
    char     topicStringDefault[]        = "IBM/PRICE";
    char     subscriptionNameDefault[]   = "IBMSTOCKPRICESUB";
    char     subscriptionQueueDefault[]  = "STOCKTICKER";
    char     publicationBuffer[101];      /* Allocate to receive messages     */
    char     resTopicStrBuffer[151];      /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";                 /* Default queue manager            */
    MQCHAR48 qName = "";                  /* Allocate storage for MQINQ       */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                */
    MQHOBJ   Hobj = MQHO_NONE;            /* subscription queue handle        */
    MQHOBJ   Hsub = MQSO_NONE;            /* subscription handle              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                  */
    MQLONG   Reason = MQRC_NONE;          /* reason code                      */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};         /* Unmanaged subscription queue     */
    MQSD     sd = {MQSD_DEFAULT};         /* Subscription Descriptor          */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor               */
    MQGMO   gmo = {MQGMO_DEFAULT};        /* get message options              */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE |
MQSO_FAIL_IF_QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName =  subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));
```

図 *47.* 非管理 *MQ* サブスクライバー *-* パート *1:* 宣言

```
    switch(argc){                   /* Replace defaults with args if provided */
default:
        switch((argv[5][0])) {
case('A'):  sdOptions =  MQSO_ALTER  | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('C'):  sdOptions =  MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
case('R'):  sdOptions =  MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF_QUIESCING;
            break;
default:    ;
        }
case(5):
    if (strcmp(argv[4],"/")) /* "/" invalid = No subscription  */
        subscriptionQueue = argv[4];
    else {
        *subscriptionQueue = '\0';
        if (argc > 5) {
            if (argv[5][0] == 'C') {
                sdOptions = sdOptions + MQSO_MANAGED;
            }
        }
        else
            sdOptions = sdOptions + MQSO_MANAGED;
    }

case(4):
    if (strcmp(argv[3],"/")) /* "/" invalid = No subscription  */
        subscriptionName = argv[3];
    else {
        *subscriptionName = '\0';
        sdOptions = sdOptions - MQSO_DURABLE;
    }
case(3):
    if (strcmp(argv[2],"/")) /* "/" invalid = No topic string  */
        topicString = argv[2];
    else
        *topicString = '\0';
case(2):
    if (strcmp(argv[1],"/")) /* "/" invalid = No topic object  */
        topicName = argv[1];
    else
        *topicName = '\0';
case(1):
    sd.Options = sdOptions;
    printf("Optional parameters: "
    printf("topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|
R(esume)\n");
    printf("Values \"%-.48s\"  \"%s\" \"%s\"  \"%-.48s\" sd.Options=%d\n",
           topicName, topicString, subscriptionName, subscriptionQueue, sd.Options);
    }
```

図 *48.* 非管理 *MQ* サブスクライバー *-* パート *2:* パラメーター処理

以下の説明は、この例でのパラメーター処理に関する追加コメントです。

switch((argv[5][0]))
例でデフォルトで使用されている MQSUB オプション設定の一部をオーバーライドする効果をテストするために、パラメーター 5 に Alter | C reate | Resume を入力することを選択できます。 この例で使用されているデフォルト設定は MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE です。

注 : MQSO_DURABLE を設定せずに MQSO_ALTER または MQSO_RESUME を設定するとエラーになります。 sd.SubName を設定して、再開または変更できるサブスクリプションを参照する必要があります。

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
デフォルトのサブスクリプション・キュー STOCKTICKER がヌル・ストリング で置き換えられた場合、MQSO_CREATE が設定されている 限り、この例は MQSO_MANAGED フラグを設定し、動的サブスクリプション・キューを作成します。 5 番目のパラメーターに Alter or Resume が設定されている場合、例の動作は subscriptionName の値によって異なります。

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;
デフォルトのサブスクリプション IBMSTOCKPRICESUB がヌル・ストリングに置き換えられると、この例では MQSO_DURABLE フラグが除去されます。 他のパラメーターをデフォルト値にしてこの例を実行する 場合、STOCKTICKER 向けの 追加の一時サブスクリプションが作成され、重複するパブリケーションを受け取ります。 この例をパラメーターなしで次に実行 すると、1 つのみのパブリケーションをもう一度受け取ります。

```
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        if (strlen(subscriptionQueue)) {
            strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
            MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING | MQOO_INQUIRE,
                  &Hobj, &CompCode, &Reason);
            if (CompCode != MQCC_OK) break;
        }
        strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        sd.ObjectString.VSPtr = topicString;
        sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
        sd.SubName.VSPtr = subscriptionName;
        sd.SubName.VSLength = MQVS_NULL_TERMINATED;
        sd.ResObjectString.VSPtr = resTopicStr;
        sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
        MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
        memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
        inquireQname(Hconn, Hobj, qName);
        printf("Waiting %d seconds for publications matching \"%s\" from %-0.48s\n",
                gmo.WaitInterval/1000, resTopicStr, qName);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding       = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publication), publication, &messlen,
&CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors    */
    MQLONG intAttrs[_intAttrs];            /* Array of integer attributes           */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
&CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strncpy(qName, "unknown queue", MQ_Q_NAME_LENGTH);
    }
    return;
}
```

図 *49.* 非管理 *MQ* サブスクライバー - パート *3:* コード本体

以下の説明は、この例でのコードに関する追加コメントです。

if (strlen(subscriptionQueue))
サブスクリプション・キュー名がない場合、この例では Hobj の値として MQHO_NONE を使用します。

MQOPEN(...);
サブスクリプション・キューがオープンされ、キュー・ハンドルが Hobj に保存されます。

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
MQOPEN から渡された Hobj (または、サブスクリプション・キュー名がない場合は MQHO_NONE) を 使用して、サブスクリプションがオープンされます。 非管理キューの再開は、MQOPEN で明示的にオープンせずに、実行できます。

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
サブスクリプションがサブスクリプション・ハンドルを使用してクローズされます。 サブスクリプションが永続的であるかどうかに応じて、サブスクリプションは暗黙の MQCO_KEEP_SUB または MQCO_REMOVE_SUB でクローズされます。 MQCO_REMOVE_SUB を使用して永続サブスクリプションをクローズすることはできますが、 MQCO_KEEP_SUB を使用して非永続サブスクリプションをクローズすることは できません 。 MQCO_REMOVE_SUB のアクションは、サブスクリプション・キューに送信されるパブリケーションを停止するサブスクリプションを除去することです。

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
サブスクリプションが非管理の場合、特別なアクションは実行されません。 キュー が管理されていて、サブスクリプションが 明示的または暗黙的な MQCO_REMOVE_SUB でクローズされた場合、すべての パブリケーションがキューから消去され、この時点でキューは削除されます。

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
受信したメッセージが、サブスクリプションのメッセージであることを確認してください。

この例の出力は、パブリッシュ/サブスクライブの各局面を示します。

301 ページの図 50 の例では、 NYSE/IBM/PRICE トピックに 130 をパブリッシュすることから開始します。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters:  TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

図 *50. NYSE/IBM/PRICE* に *130* をパブリッシュする

301 ページの図 51 では、デフォルトのパラメーターを使用して例が実行され、保存パブリケーション 130 を受け取ります。 303 ページの図 55 に示されているように、指定されたトピック・オブジェクトおよびトピック・ストリングは無視されます。 トピック・オブジェクトおよびトピック・ストリングは、サブスクリプション・オブジェクトがあれば、常にそこから 取得され、トピック・ストリングは不変です。 この例の実際の動作は、 MQSO_CREATE、 MQSO_RESUME、および MQSO_ALTER の選択または組み合わせによって異なります。 この例で選択されているオプションは MQSO_RESUME です。

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS"  "IBM/PRICE" "IBMSTOCKPRICESUB"  "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

図 *51.* 保存パブリケーションを受け取る

(302 ページの図 52) では、 永続サブスクリプションが既に保存パブリケーションを受け取ったため、パブリケーションは 受け取られません。 この例では、キュー名なしで、サブスクリプション名のみを指定することによって、サブスクリプションが 再開されています。 もしキュー名が指定されている場合は、最初にキューが オープンされ、ハンドルが MQSUB に渡されます。

注 : MQINQ からの 2038 エラーは、 MQOO_INQUIRE オプションを含まない MQSUB による STOCKTICKER の暗黙的な MQOPEN が原因です。 キューを明示的にオープンすれば、MQINQ から 2038 戻りコードが戻されるのを回避できます。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS"  "IBM/PRICE" "IBMSTOCKPRICESUB"  "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

図 *52.* サブスクリプションの再開

302 ページの図 53 では、 この例は、宛先として STOCKTICKER を使用して、非永続、非管理のサブスクリプションを 作成します。 これは新規サブスクリプションであるため、保存パブリケーションを受け取ります。

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS"  "IBM/PRICE" ""  "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

図 *53.* 新しい非管理、非永続のサブスクリプションで保存パブリケーションを 受け取る

302 ページの図 54 では、サブスクリプションのオーバーラップを説明するため、保存パブリケーションを変更する、別のパブリケーションが送信されます。 次に、新規の非永続、非管理のサブスクリプションが、サブスクリプション名を指定せずに作成されます。 保存パブリケーションは 2 回受け取られます。1 回は新規サブスクリプション用で、もう 1 回は、STOCKTICKER キューで依然としてアクティブな永続的 IBMSTOCKPRICESUB サブスクリプション用です。 この 例は、サブスクリプションを持つのはキューであり、 アプリケーションではないことを示しています。 アプリケーションのこの呼び出しでは IBMSTOCKPRICESUB サブスクリプション を参照しないにも関わらず、アプリケーションは、パブリケーションを 2 回受け取ります。 1 回は、管理的に作成された永続サブスクリプションからで、 もう 1 回は、アプリケーション自体で作成された非永続サブスクリプション からです。

```
W:\Subscribe3\Debug>..\..\Publish2\Debug\publishstock
Provide parameters:  TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|
C(reate)|R(esume)
Values "STOCKS"  "IBM/PRICE" ""  "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

図 *54.* オーバーラップしているサブスクリプション

303 ページの図 55 の 例は、新規トピック・ストリングと既存のサブスクリプションの 指定が、サブスクリプションの変更という結果にならないことを示しています。

1. 最初のケースでは、Resume は、予想の通りに、既存の サブスクリプションを再開し、変更されたトピック・ストリングを無視します。
2. 2 番目のケースでは、 Alter はエラー RC = 2510, Topic not alterable を発生させます。
3. 3 番目の例では、 Create によってエラー RC = 2432, Sub already exists が発生します。

```
W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values ""  "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB"  "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values ""  "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB"  "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values ""  "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB"  "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432
```

図 *55.* サブスクリプション・トピックは変更できない

関連概念

286 ページの『例 1: MQ パブリケーション・コンシューマー』
MQ パブリケーション・コンシューマーは、自身でトピックにサブスクライブしない、IBM WebSphere MQ メッセージ・コンシューマーです。

289 ページの『例 2: 管理対象 MQ サブスクライバー』
管理対象 MQ サブスクライバーは、ほとんどのサブスクライバー・アプリケーションで推奨されるパターンです。 この例では、キュー、 トピック 、または サブスクリプションの管理定義は 必要ありません 。

278 ページの『パブリッシャー・アプリケーションの作成』
パブリッシャー・アプリケーションをコーディングする前に、まず 2 つの例を理解 してください。 最初の例は、キューにメッセージを書き込む point-to-point アプリケーション にできるだけ近くなるようモデル化されていて、2 つ目の例は、トピックを動的に作成する方法を 示しています (これはパブリッシャー・アプリケーションではより一般的なパターンです)。

パブリッシュ/サブスクライブのライフ・サイクル

パブリッシュ/サブスクライブ・アプリケーションの設計時には、トピック、サブスクリプション、サブスクライバー、パブリケーション、パブリッシャー、およびキューの ライフ・サイクルを考慮してください。

オブジェクト (例えばサブスクリプション) のライフ・サイクルは、 作成で始まり、削除で終わります。 他の状態や変更が途中に含まれる場合も あります。例えば、一時的な中断、親トピックおよび子トピックの保有、 満了と削除などです。

キューなどの WebSphere MQ オブジェクトは、管理的に作成されるか、 Programmable Command Format (PCF) を使用して管理プログラムによって作成されるのが、従来からの一般的な方法です。 パブリッシュ/サブスクライブで異なるのは、サブスクリプションの作成と削除を行うための MQSUB および MQCLOSE API verb が 用意されていること、キューを作成して削除するだけでなく、 コンシュームされていないメッセージをクリーンアップするという、管理サブスクリプションの概念があること、 および、管理的に作成されたトピック・オブジェクトと、プログラムでまたは管理的に作成されたトピック・ストリングとの 間に関連があることです。

こうした豊富な機能によって、広範囲に及ぶパブリッシュ/サブスクライブ要件を満たすことができ、 パブリッシュ/サブスクライブ・アプリケーションのいくつかの共通パターンの設計を単純化することができます。 例えば、管理サブスクリプションは、それを作成したプログラムの間だけ存続するように意図されているサブスクリプションの、プログラミングと管理の両方を単純化します。 非管理サブスクリプションは、サブスクライブするパブリケーションとコンシュームするパブリケーションの間の接続がより緩いプログラミングを単純化します。 サブスクリプションを中央に作成することは、コンシューマーへのルーティング・パブリケーション・トラフィックのパターンが集中化された制御のモデルに基づいている場合に有効です。 例えば、フライト情報を自動ゲートに送信して、一方ではプログラムで作成されたサブスクリプションが使用されることがあります。このサブスクリプションは、ゲート・スタッフがそのフライトの乗客レコードのサブスクライブを行う場合に、ゲートでフライト番号を入力することで使用されます。

この最後の例では、管理対象の永続サブスクリプションが適切に管理されている場合があります。それは、サブスクリプションが非常に頻繁に作成され、ゲートが閉じ、サブスクリプションをプログラマチックに削除できるときに、明確なエンドポイントを持つためです。また、何らかの理由でゲート・サブスクライバー・プログラムがダウンによって乗客レコードが失われるのを防ぐため、耐久性のあるものとなっています。[2] ゲートへの乗客レコードのパブリッシュを始めるために、理想的な設計は、ゲート番号を使った乗客レコードのサブスクライブとゲート番号を使用したゲート搭乗開始イベントの公開の両方を行うゲート・アプリケーション用のものです。 パブリッシャーは、ゲート搭乗開始イベントに応えて、 乗客レコー

ドをパブリッシュします。それらの レコードは、その後、必要とされる他の関係者にも送信されることが考えられます。例えば、 フライトが実施されたことを記録するために請求書作成部門に送られたり、 乗客の携帯電話にゲート番号をテキストで通知するために顧客サービスに送られます。

中央管理されるサブスクリプションは、各ゲートごとに事前定義されたキューを使用して乗客リストをゲートに転送する、 永続非管理モデルを使用できます。

パブリッシュ/サブスクライブのライフ・サイクルについての以下の 3 つの例では、 管理非永続サブスクライバー、管理永続サブスクライバー、および非管理永続サブスクライバー と、サブスクリプション、トピック、キュー、パブリッシャー、キュー・マネージャーとの相互作用を示し、 管理プログラムとサブスクライバー・プログラム間でどのように操作が分配されるのかも 示します。

管理非永続サブスクライバー

305 ページの図 56 は、管理非永続サブスクリプションを作成し、サブスクリプションで識別されるトピックにパブリッシュされる 2 つのメッセージを取得して、終了するアプリケーションを示しています。 イタリック体のぼかしフォントでラベルが付いている、矢印が点線の相互作用は、 暗黙的なものです。

注意するべき点がいくつかあります。

1. このアプリケーションは、既にパブリッシュが 2 回行われたトピックにサブスクリプションを 作成します。 サブスクライバーは、最初のパブリケーションを受け取るときに、現在は保存パブリケーションである 2 番目 のパブリケーションを 受け取ります。
2. キュー・マネージャーは、トピックに対するサブスクリプションを作成するだけでなく、一時サブスクリプション・キューも 作成します。
3. サブスクリプションには有効期限があります。 サブスクリプションが満了すると、トピックにあるパブリケーションはこのサブスクリプションにはもう送信されません が、サブスクライバーは、サブスクリプションが満了する前にパブリッシュされたメッセージを 引き続き取得します。 パブリケーションの満了は、サブスクリプションの満了に影響を受けません。
4. 4 番目のパブリケーションは、サブスクリプション・キューには入れられず、 従って最後の MQGET はパブリケーションを戻しません。
5. サブスクライバーは自身のサブスクリプションをクローズしますが、キューまたはキュー・マネージャー への接続はクローズしません。
6. キュー・マネージャーは、アプリケーションが終了した少し後に、クリーンアップを実行します。 サブスクリプションは管理非永続なので、サブスクリプション・キュー は削除されます。

[2] パブリッシャーは、他の起こりうる障害を回避するために、乗客レコードを持続メッセージとして送信する必要があります。

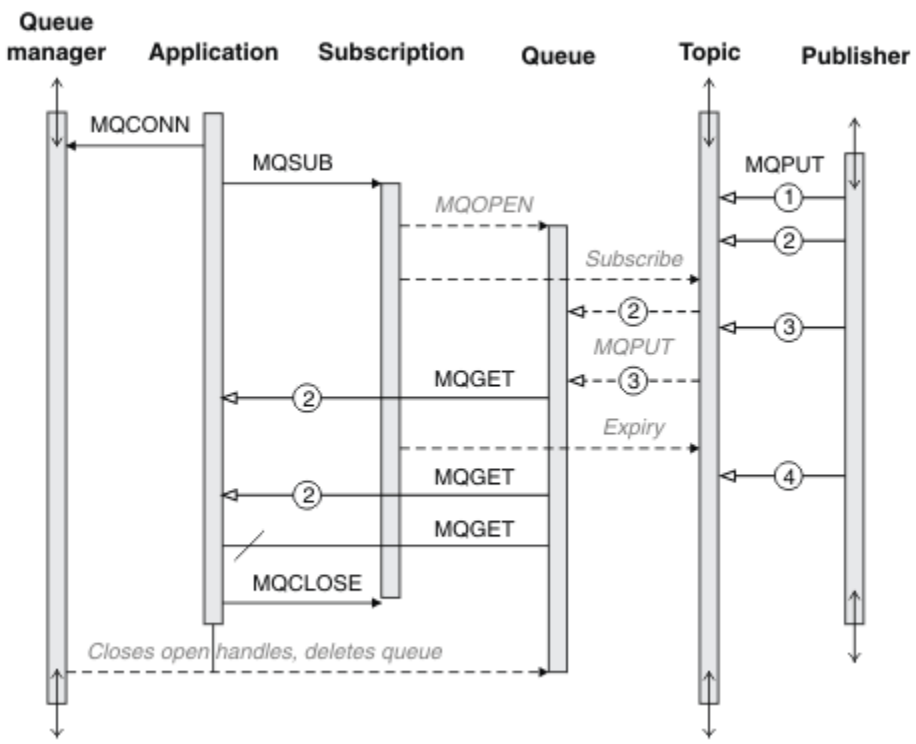


図 *56.* 管理非永続サブスクライバーのライフライン

管理永続サブスクライバー

管理永続サブスクライバー は、前の例をさらに 1 ステップ進めて、管理サブスクリプション がサブスクライブ・アプリケーションの終了および再始動の後も存続 することを示します。

注意するべき新しい点がいくつかあります。

1. この例では、前のとは違って、パブリケーション・トピックは、 サブスクリプション内に定義される前は、存在していませんでした。
2. サブスクライバーは、初めて終了する場合、 オプション MQCO_KEEP_SUB を指定してサブスクリプションをクローズします。 これは､管理永続サブスクリプション を暗黙的にクローズするデフォルトの動作です。
3. サブスクライバーがサブスクリプションを再開すると、サブスクリプション・キュー は再オープンされます。
4. 再オープンする前にキューに置かれた新規パブリケーション 2 は、サブスクリプションが除去された後でも、MQGET に対して使用可能 になっています。

   サブスクリプションが永続的であっても、パブリッシャーが送信したすべてのメッセージをサブスクライバーが確実に 受け取ることができるのは、サブスクリプションが永続サブスクリプションであり、メッセージが持続メッセージであるという両方 の条件が満たされる 場合のみです。 メッセージの持続性は、パブリッシャーによって 送信されるメッセージの MQMD 内の Persistent フィールド の設定に依存します。 サブスクライバーがこれを制御することはできません。
5. フラグ MQCO_REMOVE_SUB を設定してサブスクリプションをクローズすると、 サブスクリプションは除去され、それ以上のパブリケーションはサブスクリプション・キューに 置かれなくなります。 サブスクリプション・キューがクローズされると、 キュー・マネージャーは、未読のパブリケーション 3 を 除去した後、キューを削除します。 このアクションは、管理的にサブスクリプションを 削除することと等価です。

注 : キューを手動で削除しないでください。 また、MQCLOSE をオプション MQCO_DELETE または MQCO_PURGE_DELETE を指定して発行しないでください。 管理サブスクリプションの可視のインプリメンテーション詳細 は、サポートされる WebSphere MQ インターフェースには 含まれていません。 キュー・マネージャー管理は、完全な制御を持っていない場合、 サブスクリプションを確実に制御することはできません。

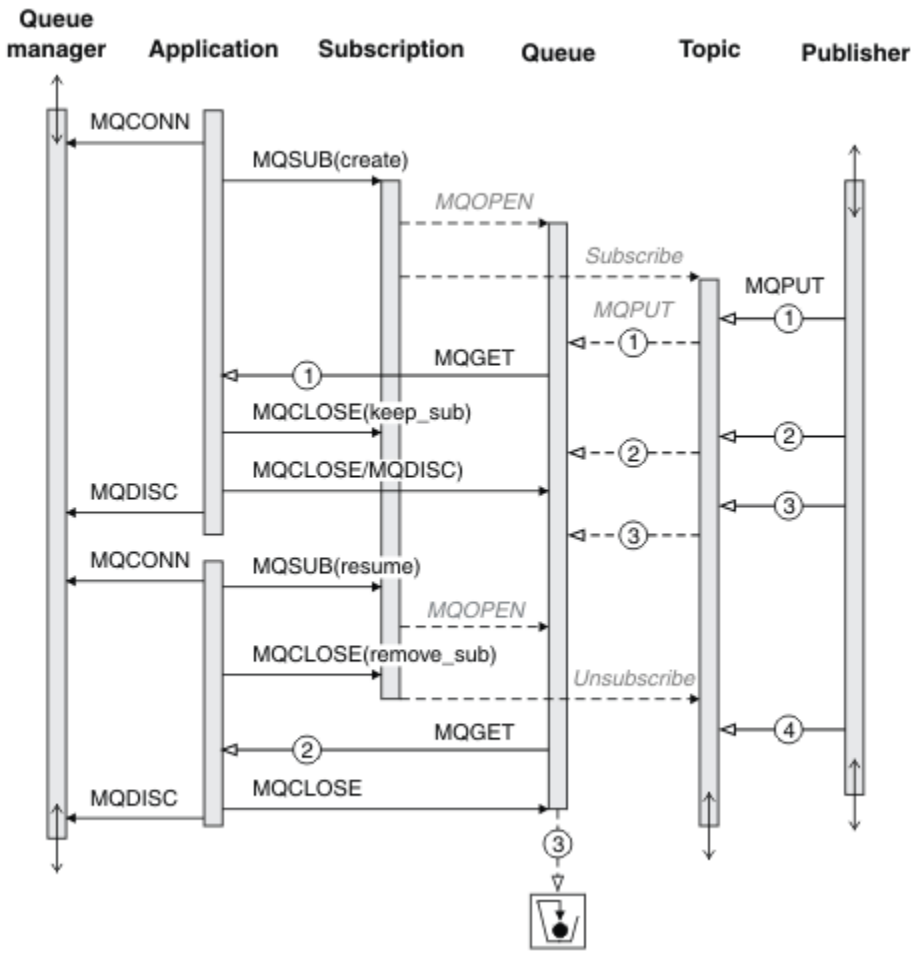


図 *57.* 管理永続サブスクライバーのライフライン

非管理永続サブスクライバー

3 番目の 例である非管理永続サブスクライバーには、管理者が追加されます。 これは、 管理者とパブリッシュ/サブスクライブ・アプリケーションとの相互作用 がどのように行われるのかを示す良い例です。

注意するべき点を以下にリストします。

1. パブリッシャーは、メッセージ 1 をトピックに 書き込みます。これは後で、サブスクリプション用に使用されるトピック・オブジェクトと関連付けられます。 トピック・オブジェクトは、ワイルドカードを使用してパブリッシュされたトピックと一致するトピック・ストリングを定義します。
2. トピックには保存パブリケーションが 1 つあります。
3. 管理者は、トピック・オブジェクト、キュー、およびサブスクリプションを作成します。 トピック・オブジェクトとキューは、サブスクリプションより前に定義されている必要があります。
4. アプリケーションは、サブスクリプションと関連付けられたキューを オープンし、MQSUB にキューのハンドルを渡します。 あるいは、その代わりに、 単にサブスクリプションをオープンし、キュー・ハンドル MQHO_NONE を渡すこともできます。 逆は正しくありません。 サブスクリプション名なしでキュ

ー・ハンドルだけを渡すことでサブスクリプションを再開することは できません。1 つのキューに複数のサブスクリプションがある可能性があるからです。

5. アプリケーションは、サブスクリプションを初めてオープンする場合でも、オプション MQSO_RESUME を 使用してサブスクリプションをオープンします。 管理的に作成されたサブスクリプション が再開されます。
6. サブスクライバーは、保存パブリケーション 1 を受け取ります。 パブリケーション 2 は、サブスクライバーがパブリケーションをどれも 受け取らないうちにパブリッシュされたにも関わらず、サブスクリプションが開始した後に パブリッシュされ、サブスクリプション・キューにある 2 番目のパブリケーションになっています。

   注 : 保存パブリケーション は、持続メッセージとしてパブリッシュされない場合、 キュー・マネージャーが再始動した後は失われます。
7. この例では、サブスクリプションは永続的です。 プログラムが 非管理非永続サブスクリプションを作成することが可能です。 これは管理者が実行できることでないことは明白です。
8. サブスクリプションをクローズする際のオプション MQCO_REMOVE_SUB の 効果は、管理者が削除したかのように、サブスクリプションを 削除することです。 これによって、それ以上のパブリケーションはキューに送信されなくなりますが、管理 永続サブスクリプションとは違って、キューがクローズされている場合であっても、既にキューにあるパブリケーションには影響しません。
9. 管理者は、後で残りのメッセージ 3 を削除し、 キューを削除します。

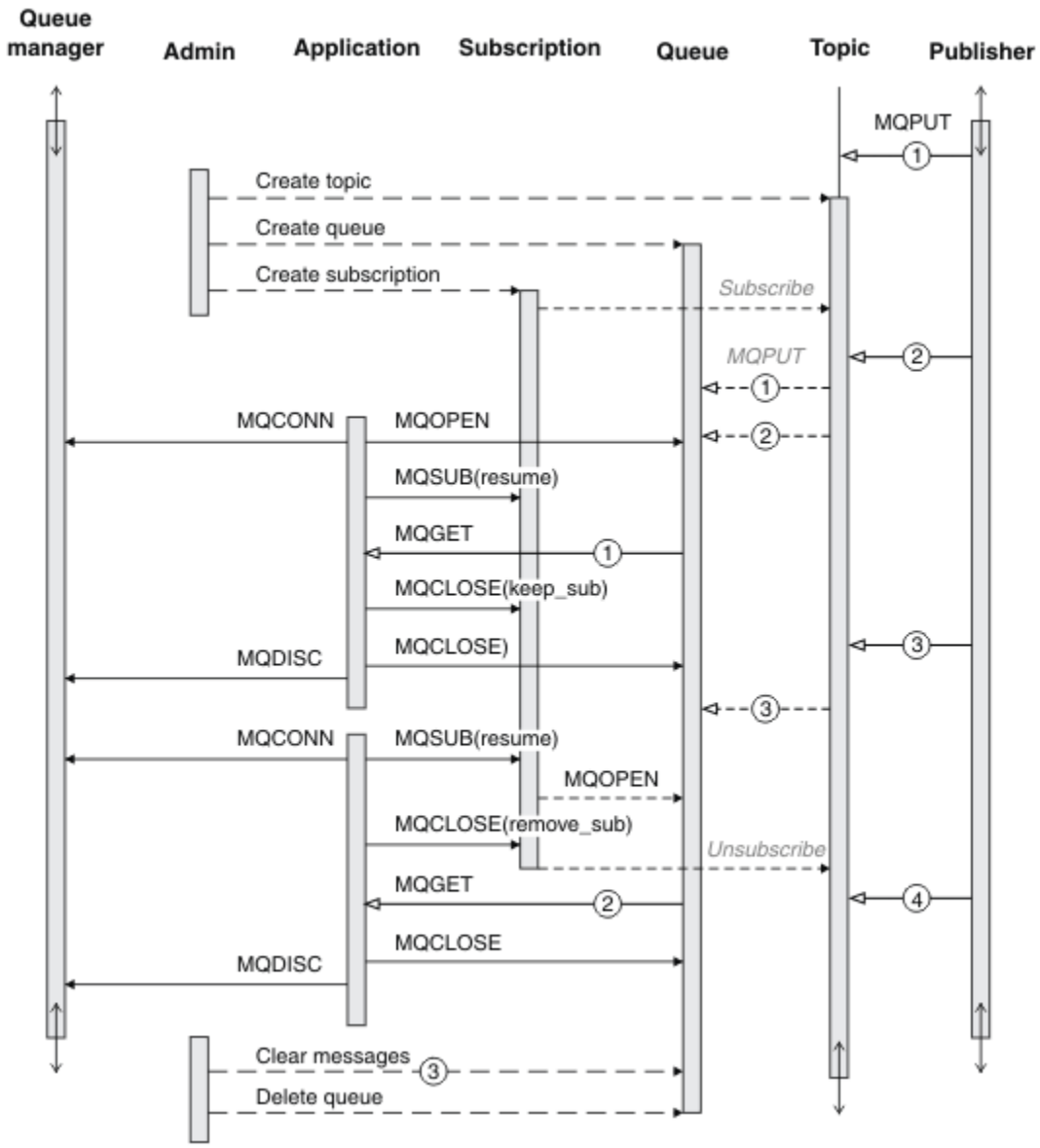


図 *58.* 非管理永続サブスクライバーのライフライン

非管理サブスクリプションの通常の パターンでは、キューとサブスクリプションのハウスキーピングは管理者に よって実行されます。 通常は、 アプリケーション・コードでプログラマチックに、管理サブスクライバーの動作をエミュレートしたり、キューとサブスクリプション を整理しようとはしません。 自身で管理ロジックを書く必要がある場合、 管理パターンを使用して同じ結果を得られるかどうかを 検討してください。 正確に同期する、完全に信頼できる管理コードをコーディングするのは、 容易なことではありません。 後で整理するほうが簡単です。 手動で行うか、または、メッセージ、サブスクリプション、およびキューを、状態に関わらず簡単に削除できることが確かであれば、 自動管理プログラムを使用できます。

パブリッシュ/サブスクライブのメッセージ・プロパティー

WebSphere MQ パブリッシュ/サブスクライブ・メッセージングにはいくつかのメッセージ・プロパティーが関係しています。

PubAccountingToken

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の AccountingToken フィールドに入る値です。 AccountingToken は、メッセージの識別コンテキ

ストの一部です。 メッセージ・コンテキストについて詳しくは、38 ページの『メッセージ・コンテキスト』を参照してください。 MQMD の AccountingToken フィールドについて詳しくは、AccountingToken を参照してください。

PubApplIdentityData

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の ApplIdentityData フィールドに入る値です。 ApplIdentityData は、メッセージの識別コンテキストの一部です。 メッセージ・コンテキストについて詳しくは、38 ページの『メッセージ・コンテキスト』を参照してください。 MQMD の ApplIdentityData フィールドについて詳しくは、ApplIdentityData を参照してください。

オプション MQSO_SET_IDENTITY_CONTEXT が指定されていない場合、デフォルト・コンテキスト情報としてこのサブスクリプションに対してパブリッシュされる各メッセージに設定される ApplIdentityData はブランクです。

オプション MQSO_SET_IDENTITY_CONTEXT が指定されている場合、PubApplIdentityData はユーザーによって生成され、このフィールドは、このサブスクリプションの各パブリケーションに設定される ApplIdentityData を含む入力フィールドになります。

PubPriority

これは、このサブスクリプションに一致するすべてのパブリケーション・メッセージのメッセージ記述子 (MQMD) の Priority フィールドに入る値です。 MQMD の Priority フィールドについて詳しくは、Priority を参照してください。

値はゼロ以上でなければなりません。 ゼロは、最低優先順位です。 以下のような特殊値も使用できます。

- MQPRI_PRIORITY_AS_Q_DEF - MQSUB 呼び出しの Hobj フィールドでサブスクリプション・キューが提供されており、管理対象ハンドルではない場合、メッセージの優先順位はそのキューの DefProirity 属性から取られます。 そのようにして特定されたキューがクラスター・キューの場合、またはキュー名の解決パスに定義が複数ある場合は、MQMD の Priority の説明のとおり、優先順位は、パブリケーション・メッセージがキューに書き込まれるときに決定されます。 MQSUB 呼び出しが管理対象ハンドルを使用する場合、メッセージの優先順位は、サブスクライブ先のトピックに関連付けられたモデル・キューの DefPriority 属性から取られます。
- MQPRI_PRIORITY_AS_PUBLISHED - メッセージの優先順位は、元のパブリケーションの優先順位です。これはこのフィールドの初期値です。

SubCorrelId

重要 : 相関 ID は、階層内ではなく、パブリッシュ/サブスクライブ・クラスター内のキュー・マネージャー間でのみ受け渡し可能です。

このサブスクリプションに一致する送信されたパブリケーションにはすべて、メッセージ記述子内にこの相関 ID が含まれます。 複数のサブスクリプションが同じキューを使用してパブリケーションを取得する場合、相関 ID で MQGET を使用すると、特定のサブスクリプションに対するパブリケーションのみを取得できます。 この相関 ID はキュー・マネージャーまたはユーザーのいずれかによって生成されます。

オプション MQSO_SET_CORREL_ID が指定されていない場合、相関 ID はキュー・マネージャーによって生成され、このフィールドは、このサブスクリプションに対してパブリッシュされる各メッセージに設定される相関 ID を含む出力フィールドになります。

オプション MQSO_SET_CORREL_ID が指定されている場合、相関 ID はユーザーによって生成され、このフィールドは、このサブスクリプションの各パブリケーションに設定される相関 ID を含む入力フィールドになります。 この場合、フィールドの値が MQCI_NONE であれば、このサブスクリプションに対してパブリッシュされる各メッセージに設定される相関 ID はメッセージの元の書き込みによって作成された相関 ID です。

オプション MQSO_GROUP_SUB が指定されており、指定された相関 ID が、同じキューおよびオーバーラップ・トピック・ストリングを使用する既存のグループ化されたサブスクリプションと同じである場合、グループ内で最も有意なサブスクリプションのみがパブリケーションのコピーと共に提供されます。

SubUserData

これは、サブスクリプションのユーザー・データです。 このフィールドでサブスクリプションについて提供されるデータは、このサブスクリプションへ送信される各パブリケーションの MQSubUserData メッセージ・プロパティーとして含まれます。

パブリケーションのプロパティー

310 ページの表 44 は、パブリケーション・メッセージで提供されるパブリケーションのプロパティーのリストを示します。

これらのプロパティーは、**MQRFH2** フォルダーから直接アクセスするか、MQINQMP を使用して取り出すことができます。 MQINQMP は、照会するプロパティーの名前として、プロパティー名または **MQRFH2** 名を受け入れます。

表 *44.* パブリケーションのプロパティー

プロパティー名	MQRFH2 名	タイプ	説明
MQTopicString	mqps.Top	MQTYPE_STRING	トピック・ストリング
MQSubUserData	mqps.Sud	MQTYPE_STRING	サブスクライバー・ユーザー・データ
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	保存パブリケーション
MQPubOptions	mqps.Pub	MQTYPE_INT32	パブリケーション・オプション
MQPubLevel	mqps.Pbl	MQTYPE_INT32	パブリケーション・レベル
MQPubTime	mqpse.Pts	MQTYPE_STRING	パブリケーション時刻
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	パブリケーション・シーケンス番号
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	パブリッシャーによって追加されたストリング/整数型データ
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	次のメッセージ形式: MQRFH1 MQRFH2 PCF

メッセージの順序付け

特定のトピックで、メッセージは、パブリッシュ・アプリケーションから受信されるのと同じ順序で キュー・マネージャーによってパブリッシュされます (メッセージ優先順位に基づき再配列されることもあります)。

メッセージの順序付けは、通常、各サブスクライバーが、特定のパブリッシャーからの 特定のトピックについて、そのパブリッシャーがパブリッシュした順序で、 特定のキュー・マネージャーからメッセージを受信することを意味します。

ただし、WebSphere MQ のすべてのメッセージの場合と同様、メッセージの配信順序が 乱れることはよくあります。 これは、次のような状況で 起こることがあります。

- ネットワーク内のリンクが切れて、以後のメッセージが別のリンクを通って 転送される場合
- キューが一時的にいっぱいになるか書き込み禁止になり、メッセージが 送達不能キューに入れられて配信が遅れたが、それ以後のメッセージは 問題なく配信された場合

- パブリッシャーとサブスクライバーがまだ作動しているときに、 管理者がキュー・マネージャーを削除したために、待機メッセージが送達不能キューに入れられ、サブスクリプションが中断された場合

以上のような状況にならない限り、 パブリケーションは必ず順序どおりに配信されます。

注 : パブリッシュ/サブスクライブでは、グループ化メッセージまたはセグメント化されたメッセージは使用できません。

パブリケーションの代行受信

パブリケーションをインターセプトして、変更した後、他のサブスクライバーに到達する前にリパブリッシュすることができます。

以下のいずれかのアクションを行うために、サブスクライバーに到達する前にパブリケーションをインターセプトすることもできます。

- メッセージに追加情報を付加する
- メッセージをブロックする
- メッセージを変換する

各メッセージに同じ操作を実行することもできますし、サブスクリプション、メッセージ、またはメッセージ・ヘッダーに応じて操作を変えることもできます。

関連資料

MQ_PUBLISH_EXIT - パブリッシュ出口

サブスクリプション・レベル

最終のサブスクライバーに到達する前にパブリケーションをインターセプトする、サブスクリプションのサブスクリプション・レベルを設定します。 インターセプト・サブスクライバーは、上位のサブスクリプション・レベルでサブスクライブし、下位のパブリケーション・レベルでリパブリッシュします。 最終のサブスクライバーに配信されるまでに、パブリケーションでメッセージ処理を実行するインターセプト・サブスクライバー・チェーンを作成します。

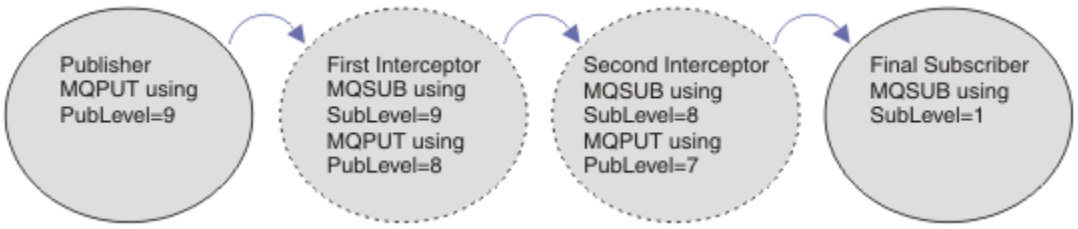


図 *59.* 一連のインターセプト・サブスクライバー

パブリケーションをインターセプトするには、**MQSD** SubLevel 属性を使用します。 インターセプトされたメッセージは、変換後、**MQPMO** PubLevel 属性を変更することによって下位のパブリケーション・レベルでリパブリッシュすることができます。 次に、メッセージは最終のサブスクライバーに送達されるか、または下位のサブスクリプション・レベルで中間サブスクライバーにより再度インターセプトされます。

インターセプト・サブスクライバーは通常、メッセージを変換してからリパブリッシュします。 一連のインターセプト・サブスクライバーがメッセージ・フローを形成します。 または、インターセプトされたパブリケーションをリパブリッシュしないこともできます。この場合、下位のサブスクリプション・レベルのサブスクライバーはメッセージを受け取りません。

インターセプターが他のどのサブスクライバーよりも先にパブリケーションを受け取るようにします。 インターセプターのサブスクリプション・レベルを、他のサブスクライバーよりも上位に設定します。 デフォルトで、サブスクライバーの SubLevel は 1 です。 最高値は 9 です。 パブリケーションは最低でも、一番高い SubLevel と同じくらいの高さの PubLevel で始まる必要があります。 最初はデフォルトの PubLevel である 9 でパブリッシュします。

- トピックでインターセプト・サブスクライバーを 1 つ設ける場合は、SubLevel を 9 に設定します。

- トピックに複数のインターセプト・アプリケーションを設ける場合は、次に続くインターセプト・サブスクライバーそれぞれについてより低い SubLevel を設定します。
- 最大で 8 つのインターセプト・アプリケーションを実装できます (サブスクリプション・レベルは 9 から下へ向かって 2 まで (両端を含む))。 メッセージの最終的な 受信側の SubLevel は 1 です。

最も高いサブスクリプション・レベルを持つインターセプターで、そのレベルがパブリケーションの PubLevel 以下のインターセプターが、最初にパブリケーションを受け取ります。 特定のサブスクリプション・レベルのトピックにつき、インターセプト・サブスクライバーを 1 つだけ構成します。 特定のサブスクリプション・レベルに複数のサブスクライバーを設定すると、パブリケーションの複数のコピーが最終のサブスクライブ・アプリケーションのセットに送信されます。

SubLevel が 0 に設定されたサブスクライバーはあらゆるパブリケーションの行き先となります。 つまり、メッセージを受け取る最終のサブスクライバーがない場合は、このサブスクライバーがパブリケーションを受け取ります。 SubLevel が 0 に設定されたサブスクライバーは、他のサブスクライバーが受け取らなかったパブリケーションをモニターするのに使用できます。

インターセプト・サブスクライバーのプログラミング

サブスクリプション・オプションは、312 ページの表 45 で説明するように使用します。

表 *45.* インターセプト・サブスクライバーのサブスクリプション・オプション

サブスクリプション・オプション	**注**
MQSO_SET_CORREL_ID と SubCorrelId を MQCI_NONE に設定する	インターセプトされたパブリケーションの CorrelId を、元のパブリケーションと同じままにします。 **注 :** 階層内のパブリケーションの相関 ID を受け渡すことはできません。 そのフィールドはキュー・マネージャーによって使用されます。
PubPriority を MQPRI_PRIORITY_AS_PUBLISHED に設定する	インターセプトされたパブリケーションの優先順位を、元のパブリケーションと同じままにします。

312 ページの表 45 に示すオプションは、すべてのインターセプト・サブスクライバーで使用される必要があります。 結果として、相関 ID とメッセージ優先順位が元のパブリッシャーの設定のままになります。

インターセプト・サブスクライバーがパブリケーションを処理する際に、自身のサブスクリプションの SubLevel よりも 1 つ低い PubLevel の同じトピックにメッセージをリパブリッシュします。 SubLevel が 9 に設定されたインターセプト・サブスクライバーの場合、8 の PubLevel でメッセージをリパブリッシュします。

メッセージを正しくリパブリッシュするには、元のパブリケーションからのいくつかの情報が必要となります。 元のメッセージで使用されているのと同じ **MQMD** を再使用し、MQPMO_PASS_ALL_CONTEXT を設定して、**MQMD** 内のすべての情報が次のサブスクライバーに渡されるようにします。 313 ページの表 46 に示されているメッセージ・プロパティーの値を、リパブリッシュされるメッセージの対応するフィールドにコピーします。 インターセプト・サブスクライバーはこれらの値を変更することができます。 OR 演算子を使用して **MQPMO**.Options フィールドに追加の値を追加し、メッセージ書き込みオプションを結合します。

管理対象パブリケーション・キューを使用するのではなく、明示的にパブリケーション・キューをオープンする必要があります。 管理対象キューに MQSO_SET_CORREL_ID を設定することはできません。 また、管理対象キューに MQOO_SAVE_ALL_CONTEXT を設定することもできません。 313 ページの『例』 に示されているコード・フラグメントを参照してください。

表 46. リパブリッシュされるメッセージの *MQPUT* 値	
MQPUT によるメッセージのリパブリッシュ	**パブリケーション・メッセージの情報**
MQOD.ObjectString	メッセージ・プロパティー MQTopicString
MQPMO.Options	メッセージ・プロパティー MQPubOptions

最終のサブスクライバーでは、サブスクリプション・オプションを異なる設定にすることもできます。 例えば、パブリケーションの優先順位を、MQPRI_PRIORITY_AS_PUBLISHED にではなく、明示的に設定することができます。 最終のサブスクライバーの設定は、チェーン内の最終のインターセプト・サブスクライバーからのパブリケーションにのみ影響を与えます。

保存パブリケーション

保存パブリケーションは、インターセプトされた後、元のメッセージ書き込みオプションをリパブリッシュされるメッセージにコピーすることによって保持する必要があります。

MQPMO_RETAIN オプションはパブリッシャーによって設定されます。 各インターセプト・サブスクライバーは、 313 ページの表 46 に示すように、リパブリッシュされたメッセージのメッセージ書き込みオプションに MQPubOptions を転送する必要があります。 メッセージ書き込みオプションをコピーすることによって、元のパブリッシャーによって設定されたオプション (パブリケーションを保持するかどうかを含め) が保持されます。

パブリケーションがインターセプト・サブスクライバー・チェーンを下方向に進んで最終のサブスクライバーに配信されると、最終的にこのパブリケーションは保持されます。 保存パブリケーションを要求する SubLevel 1 の新しいサブスクライバーが、これ以上のインターセプトなしでこのパブリケーションを受け取ります。 1 よりも高い SubLevel のサブスクライバーには、保存パブリケーションは送信されません。 結果として、保存パブリケーションがインターセプト・サブスクライバー・チェーンによって 2 度目に変更されることがありません。

例

ここでは、結合して 1 つのインターセプト・サブスクライバーを作成できるコード・フラグメントの例を示します。 ここでのコードは実動用の品質というよりも、簡潔であることを目的に記述されています。

313 ページの図 60 のプリプロセッサー・ディレクティブは、MQINQMP MQI 呼び出しが必要とするパブリケーション・メッセージから抽出される 2 つのプロパティーを定義します。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
#define      MQPUBOPTIONS        (MQPTR)(char*) "MQPubOptions",\
                                 0,\
                                 12,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
#define      MQTOPICSTRING       (MQPTR)(char*) "MQTopicString",\
                                 0,\
                                 13,\
                                 MQVS_NULL_TERMINATED,\
                                 MQCCSI_APPL
```

図 *60.* プリプロセッサー・ディレクティブ

314 ページの図 61 は、コード・フラグメントで使用される宣言を示しています。 強調表示されている用語を除いて、宣言は WebSphere MQ アプリケーションに標準のものです。

強調表示されている書き込みオプションと取得オプションはすべてのコンテキストを渡すよう初期化されています。 強調表示されている MQTOPICSTRING と MQPUBOPTIONS は、プリプロセッサー・ディレクティブで定義されているプロパティー名の MQCHARV 初期化指定子です。 名前は MQINQMP に渡されます。

```
int main(int argc, char **argv) {
    MQLONG   Reason            = MQRC_NONE;
    MQLONG   CompCode          = MQCC_OK;
    MQHCONN  Hcon              = MQHC_UNUSABLE_HCONN;
    MQCHAR   QMName[49]        = " ";
    MQCMHO   CrtMsgHOpts       = {MQCMHO_DEFAULT};
    MQHMSG   Hmsg              = MQHM_NONE;
    MQMD     md                = {MQMD_DEFAULT};
    MQHOBJ   gHobj             = MQHO_NONE;
    MQOD     getOD             = {MQOD_DEFAULT};
    MQGMO    gmo               = {MQGMO_DEFAULT};
    MQLONG   GO_Options        = MQOO_INPUT_AS_Q_DEF
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_SAVE_ALL_CONTEXT;
    MQLONG   GC_Options        = MQCO_DELETE_PURGE;
    MQHOBJ   Hsub              = MQHO_NONE;
    MQSD     sd                = {MQSD_DEFAULT};
    MQLONG   SC_Options        = MQCO_NONE;
    MQHOBJ   pHobj             = MQHO_NONE;
    MQOD     putOD             = {MQOD_DEFAULT};
    MQLONG   PO_Options        = MQOO_OUTPUT
                               | MQOO_FAIL_IF_QUIESCING
                               | MQOO_PASS_ALL_CONTEXT;
    MQLONG   PC_Options        = MQCO_NONE;
    MQPMO    pmo               = {MQPMO_DEFAULT};
    MQIMPO   InqPropOpts       = {MQIMPO_DEFAULT};
    MQPD     PropDesc          = {MQPD_DEFAULT};
    MQLONG   Type              = MQTYPE_AS_SET;
    MQCHARV  TopStrProp        = {MQTOPICSTRING};
    MQCHARV  PubOptProp        = {MQPUBOPTIONS};
    MQLONG   DataLength        = 0;
    MQBYTE   buffer[256]       = "";
    MQLONG   buflen            = sizeof(buffer) - 1;
    MQLONG   messlen           = 0;
    char     TopStrBuf[256]    = "Initial value";
    int      i                 = 0;
```

図 *61.* 宣言

宣言で容易に実行できない初期化を 315 ページの図 62 に示します。 強調表示されている値には説明が必要です。

SYSTEM.NDURABLE.MODEL.QUEUE
この例では、MQSUB を使用して管理非永続サブスクリプションをオープンするのではなく、モデル・キュー SYSTEM.NDURABLE.MODEL.QUEUE を使用して一時動的キューを作成します。 そのハンドルが MQSUB に渡されます。 キューを直接オープンすることによって、すべてのメッセージ・コンテキストを保存し、サブスクリプション・オプション MQSO_SET_CORREL_ID を設定することができます。

MQGMO_CURRENT_VERSION
ほとんどの WebSphere MQ 構造体について、現行バージョンを使用することが重要です。 gmo.MsgHandle などのフィールドは、最新バージョンの制御構造体でのみ使用可能です。

MQGMO_PROPERTIES_IN_HANDLE
元のパブリケーションで設定されているトピック・ストリングおよびメッセージ書き込みオプションは、インターセプト・サブスクライバーによってメッセージ・プロパティーを使用して取得されます。 または、メッセージで直接 **MQRFH2** 構造体が読み取られます。

MQSO_SET_CORREL_ID
MQSO_SET_CORREL_ID を次と組み合わせて使用します。

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

これらのオプションにより、相関 ID が渡されるようになります。 元のパブリッシャーによって設定されている相関 ID は、インターセプト・サブスクライバーが受け取ったパブリケーションの相関 ID フィールドに配置されます。 各インターセプト・サブスクライバーは、同じ相関 ID を渡します。 これにより、最終のサブスクライバーでは、同じ相関 ID を受け取るというオプションを持つことになります。

注 : パブリケーションがパブリッシュ/サブスクライブ階層を介して渡される場合、相関 ID は保持されることがありません。

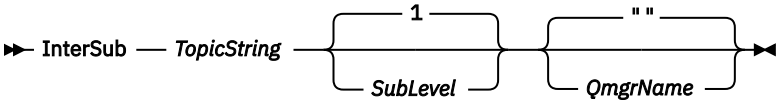
MQPRI_PRIORITY_AS_PUBLISHED
パブリケーションは、パブリッシュ時に使用されたのと同じメッセージ優先順位でパブリケーション・キューに配置されます。

```
strncpy(getOD.ObjectName,    "SYSTEM.NDURABLE.MODEL.QUEUE",
        sizeof(getOD.ObjectName));
gmo.Version                = MQGMO_VERSION_4;
gmo.Options                = MQGMO_WAIT
                           | MQGMO_PROPERTIES_IN_HANDLE
                           | MQGMO_CONVERT;
gmo.WaitInterval           = 30000;
sd.Options                 = MQSO_CREATE
                           | MQSO_FAIL_IF_QUIESCING
                           | MQSO_SET_CORREL_ID;
sd.PubPriority             = MQPRI_PRIORITY_AS_PUBLISHED;
sd.Version                 = MQSD_VERSION_1;
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
putOD.ObjectType             = MQOT_TOPIC;
putOD.ObjectString.VSPtr     = &TopStrBuf;
putOD.ObjectString.VSBufSize = sizeof(TopStrBuf);
putOD.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
putOD.ObjectString.VSCCSID   = MQCCSI_APPL;
putOD.Version                = MQOD_VERSION_4;
pmo.Version                = MQPMO_VERSION_3;
```

図 *62.* 初期化

316 ページの図 63 に、コマンド行パラメーターを読み取り、初期化を完了し、インターセプト・サブスクリプションを作成するコード・フラグメントを示します。

次のコマンドを使用してプログラムを実行します。



エラー処理を可能な限り目立たなくするために、各 MQI 呼び出しからの理由コードが別の配列エレメントに格納されます。 完了コードがテストされた各呼び出し後に、値が MQCC_FAIL の場合は、制御が `do { } while(0)` コード・ブロックを出ます。

2 つの重要なコード行は次のとおりです。

pmo.PubLevel = sd.SubLevel - 1;
リパブリッシュされるメッセージのパブリケーション・レベルを、インターセプト・サブスクライバーのサブスクリプション・レベルより 1 レベル低く設定します。

gmo.MsgHandle = Hmsg;
MQGET がメッセージ・プロパティーを返せるように、メッセージ・ハンドルを指定します。

```
do {
   printf("Intercepting subscriber start\n");
   if (argc < 2) {
       printf("Required parameter missing - topic string\n");
       exit(99);
   } else {
       sd.ObjectString.VSPtr    = argv[1];
       sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
       printf("TopicString = %s\n", sd.ObjectString.VSPtr);
   }
   if (argc > 2) {
       sd.SubLevel  = atoi(argv[2]);
       pmo.PubLevel = sd.SubLevel - 1;
       printf("SubLevel is %d, PubLevel is %d\n", sd.SubLevel, pmo.PubLevel);
   }
   if (argc > 3)
       strncpy(QMName, argv[3], sizeof(QMName));
   MQCONN(QMName, &Hcon, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &getOD, GO_Options, &gHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQSUB(Hcon, &sd, &gHobj, &Hsub, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCRTMH(Hcon, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   gmo.MsgHandle = Hmsg;
```

図 *63.* パブリケーションのインターセプトの準備

主なコード・フラグメント (317 ページの図 64) では、パブリケーション・キューからメッセージを取得します。 このコードでは、メッセージ・プロパティーを照会し、トピック・ストリングと、パブリケーションの元の **MQPMO**.option プロパティーを使用してメッセージをリパブリッシュします。

この例では、パブリケーションでは変換が実行されていません。 リパブリッシュされるパブリケーションのトピック・ストリングは、インターセプト・サブスクライバーがサブスクライブしたトピック・ストリングと必ず一致します。 インターセプト・サブスクライバーが、同じパブリケーション・キューに送信された複数のサブスクリプションのインターセプトを行う場合、異なるサブスクリプションと一致するパブリケーションを区別するために、トピック・ストリングを照会しなければならない場合があります。

MQINQMP の呼び出しが強調表示されています。 トピック・ストリングとパブリケーションのメッセージ書き込みオプションのプロパティーは、出力制御構造体に直接書き込まれます。 putOD.ObjectString の MQCHARV 長さフィールドを明示的な長さからヌル終了文字に変更する唯一の理由は、printf を使用してストリングを出力するためです。

```
while (CompCode != MQCC_FAILED) {
   memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
   md.Encoding       = MQENC_NATIVE;
   md.CodedCharSetId = MQCCSI_Q_MGR;
   printf("MQGET : %d seconds wait time\n", gmo.WaitInterval/1000);
   MQGET(Hcon, gHobj, &md, &gmo, buflen, buffer, &messlen,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   buffer[messlen] = '\0';
   MQINQMP(Hcon, Hmsg,    &InqPropOpts, &TopStrProp, &PropDesc, &Type,
       putOD.ObjectString.VSBufSize, putOD.ObjectString.VSPtr,
       &(putOD.ObjectString.VSLength), &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   memset((void *)((MQLONG)(putOD.ObjectString.VSPtr)
       + putOD.ObjectString.VSLength),'\0',1);
   putOD.ObjectString.VSLength = MQVS_NULL_TERMINATED;
   MQINQMP(Hcon, Hmsg, &InqPropOpts, &PubOptProp, &PropDesc, &Type,
       sizeof(pmo.Options), &(pmo.Options), &DataLength,
       &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQOPEN(Hcon, &putOD, PO_Options, &pHobj, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   printf("Republish message <%s> on topic <%s> with options %d\n",
       buffer, putOD.ObjectString.VSPtr, pmo.Options);
   MQPUT(Hcon, pHobj, &md, &pmo, messlen, buffer, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
       break;
   MQCLOSE(Hcon, &pHobj, PC_Options, &CompCode, &Reason);
   if (CompCode == MQCC_FAILED)
   break;
}
```

図 *64.* パブリケーションのインターセプトおよびリパブリッシュ

317 ページの図 65 に最後のコードを示します。

```
    } while (0);
    if (CompCode == MQCC_FAILED && Reason != MQRC_NO_MSG_AVAILABLE)
        printf("MQI Call failed with reason code %d\n", Reason);
    if (Hsub!= MQHO_NONE)
        MQCLOSE(Hcon, &Hsub, SC_Options, &CompCode, &Reason);
    if (Hcon!= MQHC_UNUSABLE_HCONN)
        MQDISC(&Hcon, &CompCode, &Reason);
}
```

図 *65. Completion*

パブリケーションおよび分散パブリッシュ/サブスクライブのインターセプト

分散パブリッシュ/サブスクライブ・トポロジーにインターセプト・サブスクライバーまたはパブリッシュ出口をデプロイする場合は、単純なパターンに従います。 インターセプト・サブスクライバーはパブリッシャーと同じキュー・マネージャーにデプロイし、パブリッシュ出口は最終のサブスクライバーと同じキュー・マネージャーにデプロイします。

318 ページの図 66 は、パブリッシュ/サブスクライブ・クラスター内で接続されている 2 つのキュー・マネージャーを示したものです。 パブリッシャーは、パブリケーション・レベル 9 でクラスター・トピックにパブリケーションを作成します。 番号付き矢印は、クラスター・トピックへのサブスクライバーへフローするときに、パブリケーションによって取られた一連のステップを示します。 パブリケーションは、Sublevel 9 を持つサブスクライバーによってインターセプトされ、Publevel 8 で再公開されます。 これは、Sublevel 8 のサブスクライバーによって再度インターセプトされます。 このサブスクライバーは、Publevel 7 でリパブリッシュします。 キュー・マネージャーによって提供されるプロキシー・サブスクライバーは、このパブリケーションをキュー・マネージャー B に転送します。キュー・マネージャー B には、最終のサブスクライバーだけではなくパブリッシュ出口もデプロイされています。 このパブリケーションは、パブリッシュ出口によって処理されてから、最終的に Sublevel 1 の最終のサブスクライバーによって受信されます。 インターセプト・サブスクライバーおよびパブリッシュ出口は、破線で囲んで表示しています。

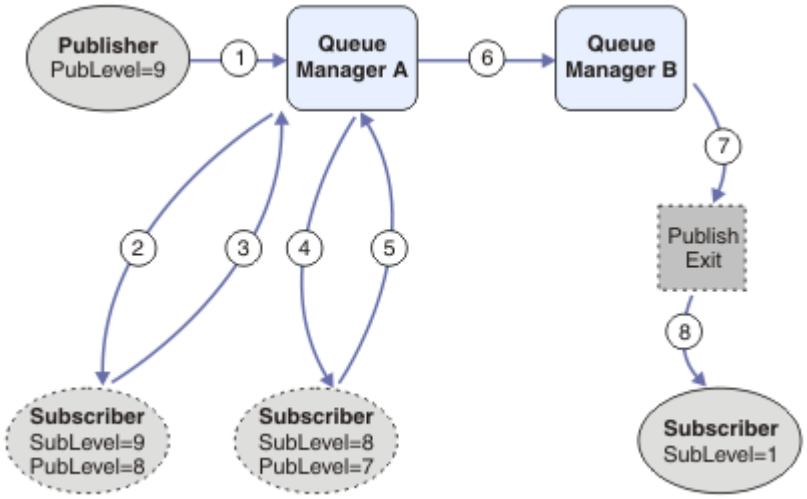


図 66. クラスター内のインターセプトおよびパブリッシュ出口

このような単純なパターンの目的は、パブリケーションを受信するすべてのサブスクライバーが、同一のパブリケーションを受信するようにすることです。 パブリケーションは、サブスクライバーがどこに接続されているかには関係なく、同じ変換シーケンスを経ます。 パブリッシャーまたは最終のサブスクライバーが接続されている場所によって変換シーケンスが変わるようなことは回避することが必要です。 ただし、各サブスクライバーに最終的に配信されるパブリケーションを調整することは妥当な例外といえます。 パブリケーションの最終配信先であるキューに基づいてパブリケーションをカスタマイズするには、パブリッシュ出口を使用します。

分散パブリッシュ/サブスクライブ・トポロジー内のどこにインターセプト・サブスクライバーおよびパブリッシュ出口をデプロイするかについては、慎重に検討する必要があります。 単純なパターンは、インターセプト・サブスクライバーをパブリッシャーと同じキュー・マネージャーにデプロイし、パブリッシュ出口を最終のサブスクライバーと同じキュー・マネージャーにデプロイすることです。

アンチパターン

319 ページの図 67 は、単純なパターンに従わなかった場合に予期しない状況になる可能性を示したものです。 最終のサブスクライバーがキュー・マネージャー A に追加され、さらに 2 つのインターセプト・サブスクライバーがキュー・マネージャー B に追加されて、デプロイメントが複雑化しています。

パブリケーションは PubLevel 7 でキュー・マネージャー B に転送され、そこで SubLevel 5 のサブスクライバーによってインターセプトされてから、SubLevel 1 の最終のサブスクライバーによってコンシュームされます。 パブリッシュ出口は、キュー・マネージャー B のインターセプト・コンシューマーと最終コンシューマーの両方にパブリケーションが渡される前にインターセプトします。 パブリケーションは、パブリッシュ出口によって処理されずに、キュー・マネージャー A の最終サブスクライバーに到達します。

パブリッシュ/サブスクライブ・トポロジーでは、プロキシー・サブスクライバーは SubLevel 1 でサブスクライブし、最後のインターセプト・サブスクライバーによって設定された PubLevel を渡します。 319 ページの図 67 では、その結果、パブリケーションは、キュー・マネージャー B にある SubLevel 9 を使用しているサブスクライバーにはインターセプトされません。

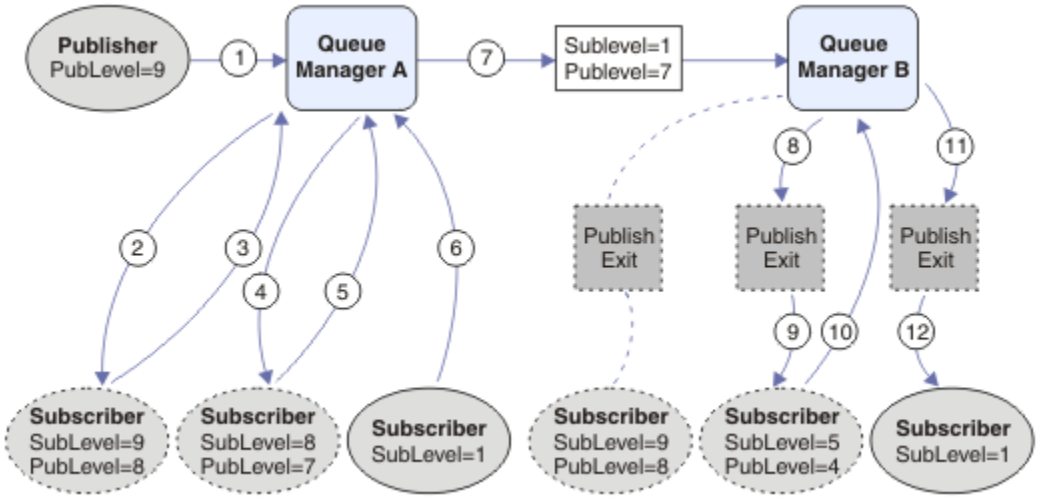


図 67. インターセプト・サブスクライバーの複雑なデプロイメント

パブリッシュのオプション

メッセージのパブリッシュ方法を制御するために使用できるオプションが幾つかあります。

サブスクライバーの応答情報の保留

サブスクライバーが受け取ったパブリケーションに対して応答できるようにしたくない場合、MQPMO_SUPPRESS_REPLYTO 書き込みメッセージ・オプションを使用して、MQMD の ReplyToQ フィールドと ReplyToQmgr フィールドの情報を保留できます。 このオプションを使用すると、キュー・マネージャーがパブリケーションを受け取るときに、いずれかのサブスクライバーに転送する前に MQMD の情報を削除します。

このオプションは、ReplyToQ を必要とするレポート・オプションと組み合わせて使用することはできません。組み合わせて使用すると、呼び出しは MQRC_MISSING_REPLY_TO_Q で失敗します。

パブリケーション・レベル

パブリケーション・レベルを使用すると、パブリケーションを受け取るサブスクライバーを制御できます。パブリケーション・レベルは、パブリケーションの宛先となるサブスクリプションのレベルを指示します。パブリケーションのパブリケーション・レベル以下のサブスクリプション・レベルのサブスクリプションで、そのうち最も高いサブスクリプション・レベルを持つサブスクリプションのみが、パブリケーションを受け取ります。 この値は、0 から 9 までの範囲でなければならず、0 が最低のパブリケーション・レベルです。 このフィールドの初期値は 9 です。 パブリッシュ・レベルおよびサブスクリプション・レベルの使用法の 1 つは、インターセプト・パブリケーションです。

パブリケーションがいずれかのサブスクライバーに送信されていないかどうかの確認

パブリケーションがいずれかのサブスクライバーに送信されていないかどうかを確認するには、MQPUT 呼び出しで MQPMO_WARN_IF_NO_SUBS_MATCHED 書き込みメッセージ・オプションを使用します。 PUT 操作によって完了コード MQCC_WARNING および理由コード MQRC_NO_SUBS_MATCHED が戻された場合、パブリケーションはどのサブスクリプションにも送信されませんでした。 PUT 操作で MQPMO_RETAIN オプションが指定されている場合、メッセージは保存され、それ以降に定義される、一致するサブスクリプションに送信されます。 分散パブリッシュ/サブスクライブ・システムでは、MQRC_NO_SUBS_MATCHED 理由コードが戻されるのは、キュー・マネージャーのトピックにプロキシー・サブスクリプションが登録されていない場合のみです。

サブスクリプション・オプション

メッセージのサブスクリプションの処理方法を制御するオプションがいくつかあります。

メッセージの持続性

キュー・マネージャーは、サブスクライバーに転送するパブリケーションの持続性を、パブリッシャーが設定したとおりに維持します。 パブリッシャーは、持続性を以下のいずれかのオプションに設定します。

0
非持続

1
永続

2
キュー/トピックの定義と同じ持続性

パブリッシュ/サブスクライブの場合、パブリッシャーがトピック・オブジェクトと **topicString** を解決済みトピック・オブジェクトにします。 パブリッシャーがキュー/トピック定義のとおりの持続性を指定している場合は、解決済みのトピック・オブジェクトのデフォルトの持続性がパブリケーションに設定されます。

保存パブリケーション

保存パブリケーションをいつ受け取るかを制御するために、サブスクライバーは次の 2 つのサブスクリプション・オプションを使用できます。

要求があったときのみパブリッシュ、MQSO_PUBLICATIONS_ON_REQUEST

いつパブリケーションを受け取るかをサブスクライバーに制御させるには、MQSO_PUBLICATIONS_ON_REQUEST サブスクリプション・オプションを使用できます。 次いでサブスクライバーは、MQSUBRQ 呼び出し (元の MQSUB 呼び出しから返される Hsub ハンドルを指定) を使用してトピックの保存パブリケーションの送信を要求することにより、いつパブリケーションを受け取るかを制御します。 MQSO_PUBLICATIONS_ON_REQUEST サブスクリプション・オプションを使用するサブスクライバーは、非保存パブリケーションを受け取りません。

MQSO_PUBLICATIONS_ON_REQUEST を指定する場合、 MQSUBRQ を使用してパブリケーションを取得する必要があります。 MQSO_PUBLICATIONS_ON_REQUEST を使用しない場合は、 パブリッシュされるときにメッセージを取得します。

サブスクライバーが MQSUBRQ 呼び出しを使用し、サブスクリプションのトピックにワイルドカードを使用すると、サブスクリプションはトピック・ツリー上の複数のトピックまたはノードと一致する可能性があり、それらのすべてと保存メッセージ (存在する場合) がサブスクライバーに送信されることになります。

特に、このオプションを永続サブスクリプションと併用すると便利な場合があります。というのは、永続的にサブスクライブする場合、サブスクライバー・アプリケーションが実行していなくても、キュー・マネージャーは継続的にパブリケーションをサブスクライバーに送信するからです。 このため、サブスクライバー・キューにメッセージが蓄積される可能性があります。 サブスクライバーの登録時に MQSO_PUBLICATIONS_ON_REQUEST オプションを使用すると、この蓄積を回避できます。 あるいは、アプリケーションにとって適切であるなら、非永続サブスクリプションを使用することによって、不要なメッセージの蓄積を回避することもできます。

サブスクリプションが永続的で、パブリッシャーが保存パブリケーションを使用する場合は、サブスクライバー・アプリケーションは MQSUBRQ 呼び出しを使用して、再始動後に状態情報をリフレッシュすることができます。 以後、サブスクライバーは MQSUBRQ 呼び出しを使用して、定期的にその状態をリフレッシュする必要があります。

このオプションを使用すると、MQSUB 呼び出しの結果としてパブリケーションが送信されることはありません。 元の永続サブスクリプションが MQSO_PUBLICATIONS_ON_REQUEST オプションを使用するように構成されている場合は、このサブスクリプションが切断してから再開されるときに、このオプションが使用されることになります。

新規パブリケーションのみ、MQSO_NEW_PUBLICATIONS_ONLY

トピックに保存パブリケーションが存在する場合、パブリケーションが行われた後でサブスクライバーがサブスクリプションを行うと、これはそのパブリケーションのコピーを受け取ることになります。

サブスクリプションよりも前に行われたパブリケーションを受け取ることを希望しないサブスクライバーは、MQSO_NEW_PUBLICATIONS_ONLY サブスクリプション・オプションを使用できます。

サブスクリプションのグループ化

パブリケーションを受け取るようにキューをセットアップしていて、 多くのオーバーラップ・サブスクリプションが同じキューにパブリケーションを送る場合に、 サブスクリプションのグループ化を検討してください。 この状態は、 オーバーラップ・サブスクリプションの例と類似しています。

トピックにサブスクライブするときにオプション MQSO_GROUP_SUB を設定することによって、 重複パブリケーションを受け取らないようにできます。 結果として、グループ内で複数のサブスクリプションが パブリケーションのトピックと一致するときは、1 つのサブスクリプションだけが キューへのパブリケーションの配置を受け持ちます。 パブリケーションのトピックと一致したその他のサブスクリプションは無視されます。

キューへのパブリケーションの配置を受け持つサブスクリプションは、ワイルドカードが出現するまでに、最も長く一致するトピック・ストリングを持つものを基にして選択されます。 このサブスクリプションは、最も一致度が高いサブスクリプションと考えることができます。 そのプロパティーは、MQSO_NOT_OWN_PUBS プロパティーがあるかどうかも含めて、パブリケーションに伝搬されます。 このプロパティーがある場合、その他の一致するサブスクリプションで MQSO_NOT_OWN_PUBS プロパティーがないとしても、 パブリケーションはキューに送達されません。

重複パブリケーションを回避するため、すべてのサブスクリプションを単一グループに置くことはできません。 グループ化したサブスクリプションは、 以下の条件を満たす必要があります。

1. いずれのサブスクリプションも管理対象ではない。
2. サブスクリプションのグループが同じキューにパブリケーションを送達する。
3. 各サブスクリプションが同じサブスクリプション・レベルにある必要がある。
4. グループ内の各サブスクリプションのパブリケーション・メッセージの 相関 ID が同じである。

   各サブスクリプションを同じ相関 ID のパブリケーション・メッセージにするには、 パブリケーション内に独自の相関 ID を作成するように MQSO_SET_CORREL_ID を設定し、 各サブスクリプションの **SubCorrelId** フィールドに同じ値を設定します。 **SubCorrelId** を値 MQCI_NONE に設定しないでください。

詳細については、../com.ibm.mq.ref.dev.doc/q100080_.dita#q100080_/mqso_group_sub を参照してください。

オブジェクト属性の照会と設定

属性は、WebSphere MQ オブジェクトの性質を定義する特性です。

キュー・マネージャーがオブジェクトを処理する方法は属性の影響を受けます。 各タイプの WebSphere MQ オブジェクトの属性については、オブジェクトの属性で詳しく説明しています。

属性の一部には、オブジェクトが定義されたときに設定されて、WebSphere MQ コマンドを使用しないと変更できないものもあります。そのような属性の例としては、キューに書き込まれるメッセージのデフォルト優先順位があります。 その他の属性は、キュー・マネージャーの操作によって影響を受け、時間の経過で変わることがあります。その例として、キューの現在の長さがあります。

ほとんどの属性の現行値を MQINQ 呼び出しによって照会できます。 また、MQI は、キュー属性のいくつかを変更できる MQSET 呼び出しを提供します。 他のタイプのオブジェクトの属性を変更するときに、この MQI 呼び出しを使用することはできません。その場合は、代わりに次のものを使用してください。

Windows Linux UNIX **WebSphere MQ for Windows、UNIX、および Linux プラットフォームの場合**

MQSC 機能。MQSC リファレンスを参照してください。

注 : この資料では、オブジェクト属性の名前を、MQINQ 呼び出しおよび MQSET 呼び出しで使用される形式で示しています。 WebSphere MQ コマンドを使用して属性を定義、変更、または表示する場合、トピック・リンクのコマンドの説明にあるキーワードを使用して属性を識別する必要があります。

MQINQ と MQSET 呼び出しの両方で、セレクターの配列を使用して識別します。それらの属性は、照会または設定する属性です。 使用できる各属性ごとにセレクターがあります。 セレクター名には、属性の性質によって決められる次の接頭部があります。

MQCA_	これらのセレクターは、文字データ (例えば、キューの名前) を含む属性を参照します。
MQIA_	これらのセレクターは、数値 (例えば、キュー上のメッセージ数を示す *CurrentQueueDepth*) または定数値 (キュー・マネージャーが同期点をサポートするかどうかを示す *SyncPoint*) を含む属性を参照します。

MQINQ または MQSET 呼び出しを使用するには、アプリケーションはキュー・マネージャーに接続されており、MQOPEN 呼び出しを使用して属性の設定または照会のためにオブジェクトをオープンしなければなりません。 これらの操作については、206 ページの『キュー・マネージャーへの接続とキュー・マネージャーからの切断』および 214 ページの『オブジェクトのオープンとクローズ』に記載されています。

オブジェクト属性の照会および設定について詳しくは、以下のリンクを参照してください。

- 322 ページの『オブジェクト属性の照会』
- 323 ページの『MQINQ 呼び出しが失敗する場合』
- 324 ページの『キュー属性の設定』

関連概念

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

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

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

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

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

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

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

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

オブジェクト属性の照会

MQINQ 呼び出しを使用して、任意のタイプの IBM WebSphere MQ の属性を照会します。

この呼び出しへの入力として、次のものを指定しなければなりません。

- 接続ハンドル。
- オブジェクト・ハンドル。
- セレクターの数。
- 属性セレクターの配列。各セレクターは、MQCA_* または MQIA_* の形式を使用します。各セレクターは、照会する値を持つ属性を表し、各セレクターは、オブジェクト・ハンドルが表すオブジェクトのタイプに対して有効である必要があります。 セレクターは、任意の順番で指定することができます。

- 照会する整数属性の値。 整数属性を照会しない場合は、ゼロを指定します。
- *CharAttrLength* の文字属性バッファーの長さ。 これは、少なくとも、各文字属性ストリングを保持するのに必要な長さの合計でなければなりません。 文字属性を照会しない場合は、ゼロを指定します。

MQINQ の出力は以下のとおりです。

- 配列にコピーされた一連の整数属性値。 値の数は *IntAttrCount* によって決まります。 *IntAttrCount* または *SelectorCount* のどちらかがゼロの場合、このパラメーターは使用されません。
- 文字属性が戻されるバッファー。 バッファーの長さは、*CharAttrLength* パラメーターによって指定されます。 *CharAttrLength* または *SelectorCount* のどちらかがゼロの場合、このパラメーターは使用されません。
- 完了コード。 完了コードが警告を示している場合、呼び出しの一部のみが完了したことを意味します。 この場合は、理由コードを調べてください。
- 理由コード。 部分完了の状況として、以下の 3 つの場合があります。
  - セレクターがキュー・タイプに適合しない。
  - 整数属性用に十分なスペースがない。
  - 文字属性用に十分なスペースがない。

  これらのうち複数の状況が起こった場合、最初に適合するものが戻されます。

また、出力や照会時にキューをオープンしても、そのキュー名が非ローカル・クラスター・キューの別名の場合、ユーザーが照会できるのはキュー名、キュー・タイプおよび共通属性だけです。 MQOO_BIND_ON_OPEN が使われている場合は、このコマンドで選択したキューの属性の値が共通属性の値になります。 MQOO_BIND_NOT_FIXED または MQOO_BIND_ON_GROUP が使用されている場合、あるいは MQOO_BIND_AS_Q_DEF が使用され、*DefBind* キュー属性が MQBND_BIND_NOT_FIXED である場合は、任意のクラスター・キューの属性値が共通属性の値になります。 詳細については、349 ページの『MQOPEN およびクラスター』および MQOPEN を参照してください。

注 : 呼び出しによって戻された値は、選択された属性のスナップショットです。 これらの属性は、プログラムが戻された値に基づいて処理を行う前に、変更できます。

MQINQ 呼び出しの説明については、MQINQ を参照してください。

MQINQ 呼び出しが失敗する場合

属性を照会するために別名をオープンした場合は、基本キューの属性ではなく、別名キュー (他のキューにアクセスするために使用される WebSphere MQ オブジェクト) の属性が戻されます。

しかし、別名が識別する、基本キューの定義もキュー・マネージャーによってオープンされます。もし別のプログラムが基本キューの使用を、MQOPEN と MQINQ 呼び出しの間に変更するのであれば、MQINQ 呼び出しはエラーになり、MQRC_OBJECT_CHANGED 理由コードを戻します。 呼び出しは、別名キュー・オブジェクトの属性が変更されたときもエラーになります。

同様に、リモート・キューをオープンして属性を照会する場合、リモート・キューだけのローカル定義の属性が戻されます。

照会するキュー属性のタイプに対して無効なセレクターを 1 つ以上指定した場合、MQINQ 呼び出しが警告で完了し、出力は次のように設定されます。

- 整数属性については、*IntAttrs* の対応するエレメントが MQIAV_NOT_APPLICABLE に設定される。
- 文字属性については、*CharAttrs* ストリングの対応する部分がアスタリスクに設定される。

照会するオブジェクト属性のタイプに対して無効なセレクターを 1 つ以上指定した場合、MQINQ 呼び出しは失敗し、MQRC_SELECTOR_ERROR 理由コードが戻ります。

MQINQ を呼び出してモデル・キューを照会することはできません。MQSC 機能を使用するか、ご使用のプラットフォームで使用可能なコマンドを使用してください。

キュー属性の設定

この情報は、MQSET 呼び出しを使用してキュー属性を設定する方法について学習するのに使用します。

MQSET 呼び出しを使用して設定できるのは、次のキュー属性だけです。

- *InhibitGet* (リモート・キューのものを除く)
- *DistList* (z/OS の場合を除く)
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

MQSET 呼び出しは、MQINQ 呼び出しと同じパラメーターを持っています。 ただし、MQSET の場合は、完了コードと理由コードを除くすべてのパラメーターが入力パラメーターです。 部分完了という状況はありません。

注 : MQI では、ローカルに定義されたキュー以外の WebSphere MQ オブジェクトの属性は設定できません。

MQSET 呼び出しの詳細については、MQSET を参照してください。

作業単位のコミットとバックアウト

ここでは、作業単位で発生したリカバリー可能な取得操作および書き込み操作をコミットおよび取り消す方法について説明します。

このトピックでは以下の用語が使用されます。

- Commit
- バックアウト
- 同期点調整
- 同期点
- 作業単位
- 単一フェーズ・コミット
- 2 フェーズ・コミット

これらのトランザクション処理の用語を既に理解している場合は、326 ページの『IBM WebSphere MQ アプリケーションでの同期点に関する考慮事項』に進んでください。

コミットとバックアウト

プログラムが作業単位内でメッセージをキューに書き込むとき、そのメッセージは、プログラムがその作業単位をコミットするまで他のプログラムには見えません。 作業単位をコミットするには、データの安全性を保護するためにすべての更新処理が正常終了する必要があります。 プログラムがエラーを検出して、PUT 操作を永続的にしないと判断した場合、その作業単位をバックアウトできます。 プログラムがバックアウトを行うと、IBM WebSphere MQ はその作業単位によってキューに書き込まれたメッセージを除去することにより、キューを復元します。 プログラムがコミットやバックアウト操作を実行する方法は、プログラムが実行されている環境に依存します。

同様に、プログラムが作業単位内でキューからメッセージを読み取ったときも、そのメッセージは、プログラムがその作業単位をコミットするまでキューに残っています。ただし、そのメッセージを他のプログラムが取り出すことはできません。 そのメッセージは、プログラムが作業単位をコミットしたときに、キューから永続的に削除されます。 プログラムが作業単位をバックアウトすると、IBM WebSphere MQ は、メッセージが他のプログラムによって検索できるようにすることによって、キューを復元します。

同期点調整、同期点、作業単位

同期点調整 とは、データの保全性を失わずに作業単位をコミットまたはバックアウトする処理のことです。

変更をコミットするかバックアウトするかについての決定は、最も単純な場合、トランザクションの終了時に行われます。 ただし、トランザクション内の別の論理点でデータ変更を同期化すると、アプリケーションに対してさらに有益になる場合もあります。 これらの論理ポイントは 同期点 (または 同期点) と呼ばれ、2 つの同期点の間の一連の更新を処理する期間は 作業単位と呼ばれます。 1 つの作業単位に、複数の MQGET 呼び出しや MQPUT 呼び出しを含めることができます。 作業単位内のメッセージの最大数は、z/OS 以外の他のプラットフォームでは ALTER QMGR コマンドの MAXUMSGS 属性を使用して制御できます。 これらのコマンドについて詳しくは、「MQSC リファレンス *WebSphere MQ Script (MQSC)* コマンド・リファレンス」を参照してください。

単一フェーズ・コミット

単一フェーズ・コミット 処理とは、キューに対する変更を他のリソース管理プログラムとの間で調整せずに、プログラムがキューに対する更新をコミットできる処理です。

2 フェーズ・コミット

2 フェーズ・コミット処理とは、プログラムが IBM WebSphere MQ のキューに対して行った更新を他のリソース (例えば、DB2 の制御下にあるデータベース) の更新と整合できる処理です。 このような処理のもとでは、すべてのリソースに対する更新が共にコミットまたはバックアウトされます。

作業単位の処理を支援するため、IBM WebSphere MQ には *BackoutCount* という属性があります。 この属性のカウントは、作業単位内でメッセージがバックアウトされるたびに増えます。 同じメッセージで作業単位が何度も異常終了すると、*BackoutCount* の値は *BackoutThreshold* の値を超えてしまいます。 この値は、キューの定義時に設定します。 このような状況が生じた場合、アプリケーションは、作業単位からメッセージを取り除いて、*BackoutRequeueQName* に定義された別のキューに書き込むことができます。 メッセージを移動すると、作業単位をコミットできます。

作業単位のコミットとバックアウトについて詳しくは、以下のリンクを参照してください。

- 326 ページの『IBM WebSphere MQ アプリケーションでの同期点に関する考慮事項』
- 327 ページの『UNIX, Linux, and Windows システムでの IBM WebSphere MQ の同期点』

関連概念

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

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

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

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

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

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

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

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

IBM WebSphere MQ アプリケーションでの同期点に関する考慮事項

この情報を使用して、IBM WebSphere MQ アプリケーションでの同期点の使用について学びます。

2 フェーズ・コミットは、次の製品でサポートされます。

- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ (Linux 用)
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- CICS (MVS/ESA 版) 4.1
- CICS Transaction Server for z/OS
- TXSeries
- IMS/ESA®
- X/Open XA インターフェースを使用するその他の外部コーディネーター

単一フェーズ・コミットは、次の製品でサポートされます。

- WebSphere MQ (UNIX システム用)
- WebSphere MQ for Windows

注 : 外部インターフェースの詳細については、329 ページの『外部同期点管理プログラムへのインターフェース』および The Open Group 発行の XA 資料「*CAE Specification Distributed Transaction Processing: The XA Specification*」を参照してください。 トランザクション管理プログラム (CICS、IMS、Encina、Tuxedo など) は、他のリカバリー可能リソースと整合する 2 フェーズ・コミットに参加できます。 これは、WebSphere MQ で提供されるキューイング機能が作業単位で有効になり、トランザクション管理プログラムによる管理が可能になることを意味します。

WebSphere MQ に同梱されているサンプルは、WebSphere MQ 調整の XA 準拠データベースを示しています。 このサンプルの詳細については、96 ページの『分散プラットフォームにおけるサンプル・プログラム』を参照してください。

WebSphere MQ アプリケーションでは、書き込みおよび読み取り呼び出しごとに、呼び出しを同期点の制御下に置くかどうかを指定できます。 書き込み操作が同期点の制御のもとに行われるようにするには、MQPUT の呼び出し時に MQPMO 構造体の *Options* フィールドに MQPMO_SYNCPOINT を指定してください。 読み取り操作の場合は、MQGMO 構造体の *Options* フィールドに MQGMO_SYNCPOINT 値を指定します。 明示的にオプションを選択しないと、デフォルトの処置が取られます。これは、プラットフォームによって異なります。 同期点制御のデフォルト値は、「no」です。

MQPUT1 呼び出しが MQPMO_SYNCPOINT を指定して発行されると、 書き込み操作を非同期に完了するように、デフォルトの動作が変更されます。 これによって、 戻される MQOD および MQMD の構造体内の特定のフィールドに未定義の値が含まれるようになるため、それらのフィールドに依存する一部のアプリケーションの動作が変更される場合があります。 アプリケーションでは、MQPMO_SYNC_RESPONSE を指定することで、書き込み操作を同期させて実行すること、および該当するすべてのフィールドの値を完成させることができます。

アプリケーションが同期点での MQPUT または MQGET への応答で MQRC_BACKED_OUT 理由コードを受け取った場合、アプリケーションは通常、MQBACK を使用して現行トランザクションをバックアウトし、その後で、適切であればトランザクション全体を再試行する必要があります。 アプリケーションが MQCMIT または MQDISC 呼び出しの応答で MQRC_BACKED_OUT を受け取った場合は、MQBACK を呼び出す必要はありません。

MQGET 呼び出しがバックアウトされるたびに、影響を受けるメッセージの MQMD 構造体の *BackoutCount* フィールドで、カウントが増えます。 *BackoutCount* のカウントが高いことは、メッセージがたびたびバックアウトされていることを示します。 これは、このメッセージに問題があることを示している場合があるため、調査が必要です。 *BackoutCount* について詳しくは、BackoutCount を参照してください。

コミットされていない要求があるのに、プログラムが MQDISC 呼び出しを発行した場合は、暗黙の同期点が生じます。 プログラムが異常終了した場合は、暗黙のバックアウトが発生します。

キュー属性の変更 (MQSET 呼び出しまたはコマンドのいずれかによる) は、作業単位のコミットやバックアウトには影響されません。

UNIX, Linux, and Windows システムでの IBM WebSphere MQ の同期点

同期点のサポートは、ローカルとグローバルという 2 種類の作業単位を対象とします。

ローカル 作業単位とは、WebSphere MQ キュー・マネージャーのリソースのみが更新対象のリソースとなる作業単位をいいます。 同期点調整は、キュー・マネージャー自身が単一フェーズ・コミット・プロシージャーにより行います。

グローバル 作業単位は、他のリソース管理プログラムのリソース (データベースなど) も更新する作業単位です。 WebSphere MQ は、このような作業単位を調整できます。 また、これらの作業単位は、別のトランザクション管理プログラムや IBM i コミットメント制御プログラムなどの外部コミットメント制御プログラムで調整することもできます。

データの保全性を失わないようにするために、2 フェーズ・コミット・プロシージャーを使用してください。 2 フェーズ・コミットは、XA 準拠トランザクション管理プログラムとデータベース (例えば IBM TXSeries および UDB など)、でも提供されています。 WebSphere MQ 製品 (ただし、WebSphere MQ for IBM i および WebSphere MQ for z/OS を除く) は、2 フェーズ・コミット・プロセスを使用してグローバル作業単位を調整できます。

ローカル作業単位

キュー・マネージャーのみに関連のある作業単位のことをローカル 作業単位と呼びます。 同期点調整は、単一フェーズ・コミット・プロセスを使用して、キュー・マネージャー自体 (内部調整) によって提供されます。

ローカル作業単位を開始するには、アプリケーションが適切な同期点オプションを指定して MQGET、MQPUT、または MQPUT1 要求を発行します。 作業単位は MQCMIT によりコミットされるか、または MQBACK によりロールバックされます。 ただし、(故意にかどうかを問わず) アプリケーションとキュー・マネージャー間の接続が切断された場合にも、作業単位は終了します。

WebSphere MQ が調整するグローバル作業単位が活動状態のときでも、アプリケーションがキュー・マネージャーから切断されると (MQDISC)、作業単位のコミットが試みられます。 ただし、アプリケーションが切断されずに終了した場合には異常終了したものと見なされるため、作業単位はロールバックされます。

グローバル作業単位

グローバル作業単位は、他のリソース管理プログラムのリソースも更新する必要がある場合に使用します。

この調整は、キュー・マネージャーの内部または外部で行うことができます。

内部同期点調整

キュー・マネージャーによるグローバル作業単位の調整は、WebSphere MQ for IBM i や WebSphere MQ for z/OS ではサポートされません。 この調整は WebSphere MQ MQI クライアント環境ではサポートされません。

調整は、WebSphere MQ によって行われます。 グローバル作業単位を開始するには、アプリケーションが MQBEGIN 呼び出しを発行します。

MQBEGIN 呼び出しの入力として、接続ハンドル (*Hconn*) を必ず指定してください。このハンドルは MQCONN または MQCONNX 呼び出しから戻されます。 このハンドルは、WebSphere MQ キュー・マネージャーへの接続を表します。

アプリケーションは適切な同期点オプションを指定して MQGET、MQPUT、または MQPUT1 要求を発行します。 これは、ローカル・リソース、他のリソース管理プログラムのリソース、またはその両方を更新するグローバル作業単位を MQBEGIN により開始できることを意味します。 他のリソース管理プログラムのリソースの更新は、そのリソース管理プログラムの API を使用して行います。 ただし、MQI を使用して、他のキュー・マネージャーに属するキューを更新することはできません。 MQCMIT または MQBACK を発行してから、後続の作業単位 (ローカルまたはグローバル) を開始してください。

グローバル作業単位は MQCMIT によりコミットされます。MQCMIT により、その作業単位に関連のあるすべてのリソース管理プログラムの 2 フェーズ・コミットが開始されます。 2 フェーズ・コミット処理では、まず、すべてのリソース管理プログラム (DB2、Oracle、Sybase などの XA 準拠のデータベース・マネージャー) に、コミットできる状態であるかどうかを問い合わせます。 すべてのリソース管理プログラムがコミットできる状態である場合にのみ、作業単位をコミットするよう指示します。 いずれかのリソース管理プログラムからコミットできないと通知された場合は、各リソース管理プログラムに作業単位をバックアウトするよう指示します。 あるいは、MQBACK を使用して、すべてのリソース管理プログラムの更新をロールバックすることもできます。

グローバル作業単位が活動状態のときでもアプリケーションが切断されると (MQDISC)、作業単位はコミットされます。 ただし、アプリケーションが切断されずに終了した場合には異常終了したものと見なされるため、作業単位はロールバックされます。

MQBEGIN からの出力は完了コードと理由コードです。

MQBEGIN を使用してグローバル作業単位を開始した場合は、キュー・マネージャーを使用して構成されたすべての外部リソース管理プログラムが対象となります。 ただし、この呼び出しで作業単位を開始しても、次のどちらかの場合にこの呼び出しは完了して警告が発行されます。

- 参加するリソース管理プログラムがない場合 (つまり、キュー・マネージャーを使用して構成されたリソース管理プログラムがない場合)

または

- 1 つまたは複数のリソース管理プログラムが使用不可の場合

上記の場合は、作業単位を開始したときに使用可能であったリソース管理プログラムのみの更新を、作業単位に含めなければなりません。

いずれかのリソース管理プログラムが更新をコミットできない場合は、すべてのリソース管理プログラムが更新をロールバックするよう指示され、MQCMIT は警告付きで終了します。 まれに (通常はオペレーターの介入があった場合)、リソース管理プログラムの中で更新をコミットしたものとロールバックしたものがあった場合、MQCMIT 呼び出しが失敗することがあります。この場合、作業は完了しますが、両方の処理結果が混ざり合って 生成されます。 このような障害が発生した場合は、キュー・マネージャーのエラー・ログでその原因を診断して、訂正処置を取ることができます。

グローバル作業単位の場合、MQCMIT 呼び出しが成功するのは、関連するすべてのリソース管理プログラムが更新をコミットした場合に限ります。

MQBEGIN 呼び出しの説明については、MQBEGIN を参照してください。

外部同期点調整

外部同期点調整は、WebSphere MQ 以外の同期点コーディネーター (CICS、Encina、Tuxedo など) が選択された場合に発生します。

この場合、WebSphere MQ (UNIX and Linux システム用) および WebSphere MQ for Windows は、同期点コーディネーターに作業単位の結果の処理方法を登録し、コミットされない読み取りまたは書き込み操作を必要に応じてコミットまたはロールバックできるようにします。 外部同期点コーディネーターは、単一フェーズまたは 2 フェーズ・コミットメント・プロトコルが提供されたかどうかを判断します。

外部のコーディネーターを使用する場合、MQCMIT、MQBACK、および MQBEGIN は発行できません。 これらの関数を呼び出しても失敗し、理由コード MQRC_ENVIRONMENT_ERROR が戻されます。

外部整合の作業単位の開始方法は、同期点コーディネーターによって提供されるプログラミング・インターフェースによって異なります。 明示的な呼び出しが必要な場合もあります。 明示的な呼び出しが必要な場合、作業単位が開始されていないときに MQPMO_SYNCPOINT オプションを指定して MQPUT 呼び出しを発行すると、完了コード MQRC_SYNCPOINT_NOT_AVAILABLE が戻されます。

作業単位の有効範囲は、同期点コーディネーターによって判断されます。 アプリケーションとキュー・マネージャーとの間の接続状態は、アプリケーションが発行する MQI 呼び出しの成功か失敗かに影響を与えますが、作業単位の状況には影響はありません。 例えば、作業単位が活動状態のときでもアプリケーションはキュー・マネージャーとの接続を切断したり再接続したりすることが可能であり、さらに同じ作業単位内において別の MQGET および MQPUT 操作を実行することもできます。 これを、保留状態の切断といいます。

CICS の XA 機能を使用するかどうかに関係なく、 CICS プログラムで WebSphere MQ API 呼び出しを使用できます。 XA を使用しない場合、キューに対するメッセージの読み書きは、CICS アトミックの作業単位内では管理されません。 この方法を選択する 1 つの理由は、作業単位の全体の整合性が重要でないことです。

作業単位の整合性が重要である場合は、必ず XA を使用してください。 XA を使用すると、CICS で 2 フェーズ・コミット・プロトコルが使用されるため、作業単位内のすべてのリソースが一緒に更新されます。

トランザクション・サポートのセットアップについて詳しくは、41 ページの『トランザクション・サポートのシナリオ』を参照してください。また、TXSeries CICS 資料 (例えば、*TXSeries for Multiplatforms CICS Administration Guide for Open Systems*) も参照してください。

外部同期点管理プログラムへのインターフェース

WebSphere MQ (UNIX and Linux システム)、および WebSphere MQ for Windows は、X/Open XA インターフェースを使用する外部同期点管理プログラムによるトランザクションの調整をサポートします。

一部の XA トランザクション管理プログラム (TXSeries) では、各 XA リソース管理プログラムがその名前を提供する必要があります。 これは、XA スイッチ構造体内の name というストリングです。 UNIX、 Linux 、および Windows システム上の WebSphere MQ のリソース・マネージャーは、MQSeries_XA_RMI と命名されます。 XA インターフェースについて詳しくは、The Open Group が発行している XA 資料「*CAE Specification Distributed Transaction Processing: The XA Specification*」を参照してください。

XA 構成では、 UNIX、 Linux、および Windows システム上の WebSphere MQ は、XA Resource Manager の役割を果たします。 XA 同期点コーディネーターは、一連の XA リソース管理プログラムを管理し、両方のリソース管理プログラムにおけるトランザクションのコミットとバックアウトを同期できます。 このように、XA 同期点コーディネーターは静的に登録されたリソース管理プログラムに対して機能します。

1. アプリケーションは、トランザクションを開始させたいことを同期点コーディネーターに通知する。
2. 同期点コーディネーターは、認識している任意のリソース管理プログラムに呼び出しを発行し、現行のトランザクションについて通知する。
3. アプリケーションは、現行のトランザクションに関連したリソース管理プログラムによって管理されているリソースを更新するために、呼び出しを発行する。
4. アプリケーションは、同期点コーディネーターがトランザクションをコミット、またはロールバックするように要求を出す。
5. 同期点コーディネーターは、2 フェーズ・コミット・プロトコルを使用して、各リソース管理プログラムに呼び出しを発行し、要求どおりにトランザクションを完了する。

XA 仕様は、各リソース管理プログラムに、*XA Switch* という構造体を提供するよう要求を出します。 この構造体では、リソース管理プログラムの能力と同期点コーディネーターによって呼び出される機能を宣言します。

この構造体には次の 2 種類があります。

MQRMIXASwitch	静的 XA リソース管理
MQRMIXASwitchDynamic	動的 XA リソース管理

この構造体が含まれているライブラリーのリストについては、70 ページの『IBM WebSphere MQ XA スイッチ構造』を参照してください。

これらを XA 同期点コーディネーターにリンクさせるために使用する必要のある方式はコーディネーターによって定義されています。そのコーディネーターの文書を参照して、WebSphere MQ が XA 同期点コーディネーターと連携するための方法を判別してください。

同期点コーディネーターによる *xa_open* 呼び出しで渡される *xa_info* 構造体は、管理されるキュー・マネージャーの名前にすることができます。 これは、MQCONN または MQCONNX に渡されるキュー・マネージャー名と同じ形式であり、デフォルトのキュー・マネージャーが使用される場合にブランクになります。ただし、TPM と AXLIB という 2 つの追加のパラメーターを使用することができます。

TPM では、WebSphere MQ にトランザクション管理プログラム名 (例えば CICS など) を指定することができます。 AXLIB では、XA AX 入り口点がある、トランザクション管理プログラム内の実際のライブラリー名を指定できます。

いずれかのパラメーターや、デフォルト以外のキュー・マネージャーを使用する場合は、QMNAME パラメーターを使用してキュー・マネージャー名を指定する必要があります。 詳細については、xa_open ストリングの CHANNEL、TRPTYPE、CONNAME、および QMNAME パラメーターを参照してください。

制限対象機能

1. グローバル作業単位は、共用 Hconn では許可されません (212 ページの『MQCONNX による共用 (スレッド独立) 接続』を参照)。
2. Windows システムでは、XA スイッチで宣言されるすべての関数は、_cdecl 関数として宣言されます。
3. 外部の同期点コーディネーターでは、一度に 1 つのキュー・マネージャーしか管理できません。 これは、各キュー・マネージャーに対してコーディネーターが効果的に接続されていることを前提としているので、一時点に許される接続は 1 つだけという規則の制約を受けるからです。

   注 : 注: JEE サーバーで実行される JMS クライアント・アプリケーション (CLIENT JEE アプリケーション) はこの制約を受けないため、 単一の JEE サーバーで管理されるトランザクションは、同じトランザクションで複数のキュー・マネージャーを調整できます。 ただし、バインディング・モードで実行される JMS サーバー・アプリケーションは、依然として一時点に許される接続は 1 つだけという規則の制約を受けます。
4. 同期点コーディネーターを使用して実行されるアプリケーションはすべて、コーディネーターによって管理されるキュー・マネージャーだけに接続できます。それは、これらアプリケーションが、既にそのキュー・マネージャーと有効な接続関係にあるためです。 これらのアプリケーションは、MQCONN または MQCONNX を発行して接続ハンドルを取得し、MQDISC を発行してから終了する必要があります。あるいは、TXSeries CICS に出口 UE014015 を使用することができます。

トリガーによる IBM WebSphere MQ アプリケーションの開始

トリガーについて、およびトリガーを使用して IBM WebSphere MQ アプリケーションを開始する方法について理解します。

キューを取り扱う一部の WebSphere MQ アプリケーションは絶えず稼働しているので、これらを使用してキューに到着したメッセージをいつでも取り出すことができます。 しかし、キューに到着するメッセージの数が予測できないときは、これが望ましくないこともあります。 この場合は、取り出すメッセージがないときでもアプリケーションがシステム・リソースを消費する可能性があります。

WebSphere MQ は、取り出すことができるメッセージがある場合に自動的にアプリケーションを開始できる機能を提供します。 この機能は、トリガー操作 と呼ばれます。

チャネルのトリガー操作については、チャネルのトリガー 操作を参照してください。

トリガー操作とは

キュー・マネージャーは、特定の条件を、トリガー・イベント を構成するものとして定義します。

トリガー操作がキューに対して有効になっている場合にトリガー・イベントが発生すると、キュー・マネージャーはトリガー・メッセージ を開始キュー に送信します。 開始キューにトリガー・メッセージがある場合、トリガー・イベントが発生したことを意味しています。

キュー・マネージャーが生成したトリガー・メッセージは、永続的ではありません。 これによりロギングが減少し (結果的にパフォーマンスが改善され)、再始動中の重複が最小限になります。その結果、再始動の時間が短縮されます。

開始キューを処理するプログラムは、トリガー・モニター・アプリケーション と呼ばれ、トリガー・メッセージを読み取り、トリガー・メッセージの情報に基づいて適切な処理を行います。 通常この処理によって、他のアプリケーションが開始され、トリガー・メッセージを生成したキューが処理されます。 キュー・マネージャーから見ると、トリガー・モニター・アプリケーションは特別なものではなく、キュー (開始キュー) からメッセージを読み取るアプリケーションの 1 つにすぎません。

トリガー操作がキューに対して有効になっている場合は、そのキューに関連するプロセス定義オブジェクト を作成できます。 このオブジェクトには、トリガー・イベントを発生させたメッセージを処理するアプリケーションについての情報が入っています。 プロセス定義オブジェクトが作成されると、キュー・マネージャーはこの情報を抽出し、トリガー・モニター・アプリケーションが使用できるようにこの情報をトリガー・メッセージに入れます。 キューと関連するプロセス定義の名前は、*ProcessName* ローカル・キュー属性によって指定されます。 各キューがそれぞれ異なるプロセス定義を指定することができます。また、いくつかのキューが同じプロセス定義を共有することもできます。

チャネルの開始をトリガーする場合、プロセス定義オブジェクトを定義する必要はありません。 伝送キュー定義が代わりに使用されます。

トリガー操作は、次の環境で稼働する WebSphere MQ クライアントによってサポートされます。

- UNIX and Linux システム
- Windows システム

クライアント環境で稼働するアプリケーションは、クライアント・ライブラリーにリンクすることを除いて、完全な WebSphere MQ 環境で稼働するアプリケーションと同じです。 ただし、トリガー・モニターおよび開始されるアプリケーションは、同じ環境にある必要があります。

トリガー操作には以下が必要です。

アプリケーション・キュー
アプリケーション・キュー はローカル・キューであり、トリガー操作がオンに設定されていると、条件が合致した場合にはトリガー・メッセージを書き込むように要求します。

プロセス定義
アプリケーション・キューにはプロセス定義オブジェクト を関連付けることができ、そこにアプリケーション・キューからメッセージを取得するアプリケーションの詳細が保持されます。 (属性のリストについては、プロセス定義の属性を参照してください。)

トリガーでチャネルを開始する場合、プロセス定義オブジェクトを定義する必要はないことに注意してください。

伝送キュー
トリガーでチャネルを開始する場合、伝送キューが必要です。

AIX、 HP-UX、 IBM i、Solaris、 z/OS、または Windows システム上の伝送キューの場合、伝送キューの *TriggerData* 属性に、開始するチャネルの名前を指定できます。 これは、チャネルをトリガーする際にプロセス定義の代わりに使用できます。ただし、プロセス定義が作成されていない場合にのみ使用されます。

トリガー・イベント
トリガー・イベント とは、キュー・マネージャーによってトリガー・メッセージが生成される起因となるイベントのことです。 これは、通常、アプリケーション・キューに到着するメッセージですが、他の場合にも発生することがあります (336 ページの『トリガー・イベントの条件』を参照)。 WebSphere MQ には、トリガー・イベントの原因となる条件を制御できるようにする一連のオプションがあります (340 ページの『トリガー・イベントの制御』を参照)。

トリガー・メッセージ
キュー・マネージャーは、トリガー・イベントを認識するとトリガー・メッセージ を作成します (336 ページの『トリガー・イベントの条件』を参照)。 キュー・マネージャーは、開始するアプリケーションに関する情報をトリガー・メッセージ内にコピーします。 この情報は、アプリケーション・キューおよびアプリケーション・キューに関連付けられているプロセス定義オブジェクトから得られます。 トリガー・メッセージは、固定形式です (347 ページの『トリガー・メッセージの形式』を参照)。

開始キュー
開始キュー は、キュー・マネージャーがトリガー・メッセージを書き込むローカル・キューです。 開始キューを、別名キューまたは モデル・キューにすることはできません。 1 つのキュー・マネージャーは、複数の開始キューを所有できます。各開始キューは、1 つ以上のアプリケーション・キューに関連付けられます。 共用キュー、つまり、キュー共用グループ内のキュー・マネージャーがアクセスできるローカル・キューを、WebSphere MQ for z/OS の開始キューにすることができます。

トリガー・モニター

トリガー・モニター は、継続的に稼働しているプログラムで、1 つ以上の開始キューを処理します。 トリガー・メッセージが開始キューに到達すると、 トリガー・モニターがそのメッセージを検索します。 トリガー・モニターは、トリガー・メッセージの情報を使用します。 トリガー・モニターはアプリケーション・キューに到着するメッセージを取り出すアプリケーションを開始するコマンドを発行して、アプリケーション・キューの名前といった、トリガー・メッセージ・ヘッダーに入っている情報をこのコマンドに渡します。

すべてのプラットフォームにおいて、チャネル・イニシエーターという特別なトリガー・モニターが、チャネルの開始を制御します。 z/OS では、チャネル・イニシエーターは通常、手動で開始されるか、キュー・マネージャー始動 JCL の CSQINP2 を変更することによって、キュー・マネージャーが開始する際に自動的に開始されます。 他のプラットフォームでは、キュー・マネージャー始動時に自動的に開始されるか、runmqchi コマンドで手動で開始できます。

(詳細については、344 ページの『トリガー・モニターによる開始キュー処理』を参照してください。)

トリガー操作の働きを理解するには、332 ページの図 68 を参照してください。これは、トリガー・タイプ FIRST (MQTT_FIRST) の例です。

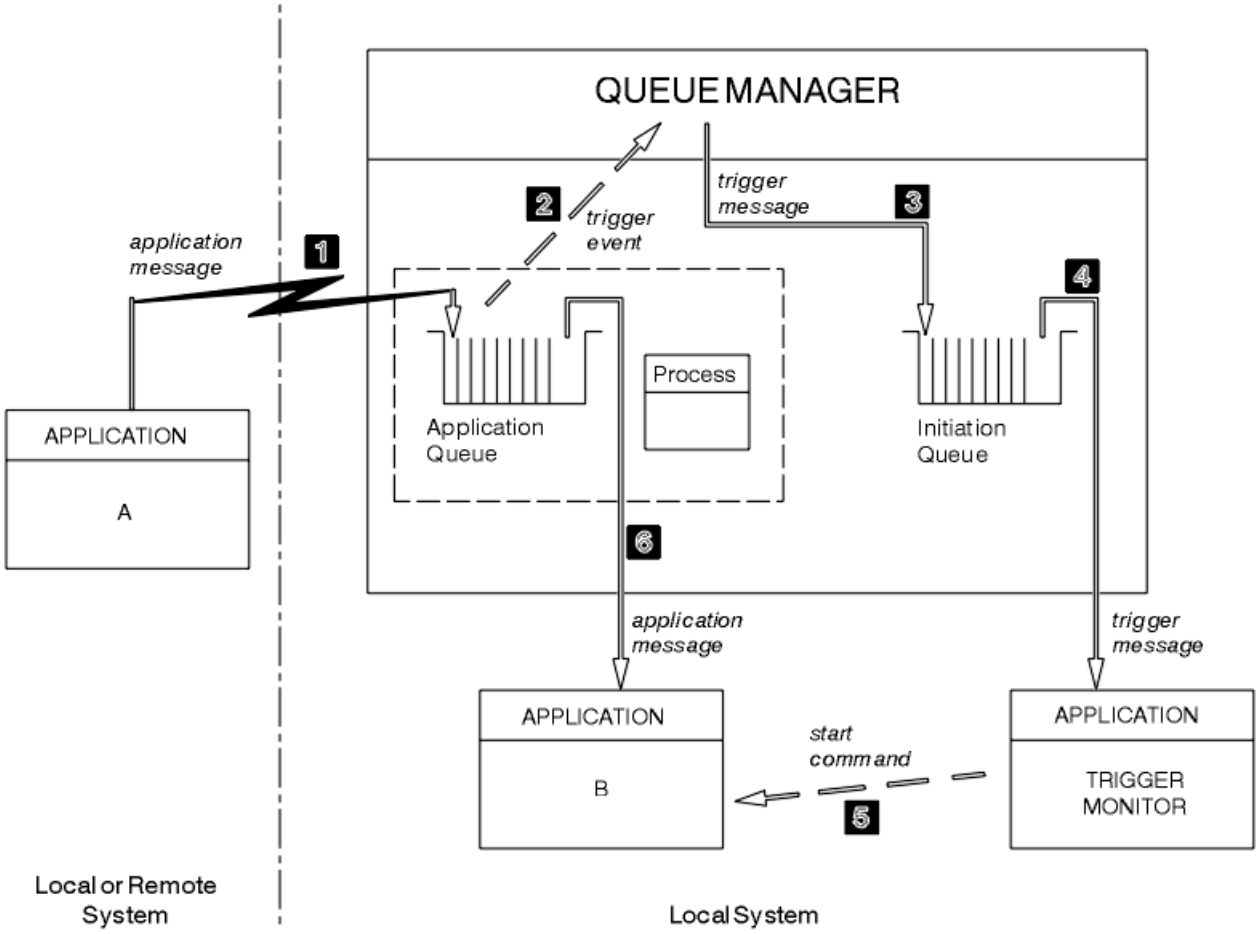


図 *68.* アプリケーションおよびトリガー・メッセージの流れ

332 ページの図 68 では、イベントの順序は以下のようになります。

1. アプリケーション A (キュー・マネージャーに対してローカルでもリモートでもよい) は、アプリケーション・キューにメッセージを書き込みます。 入力のためにこのキューをオープンしているアプリケーションはありません。 ただし、これが関係するのは、トリガー・タイプ FIRST および DEPTH のみです。
2. キュー・マネージャーは、この条件がトリガー・イベント生成の必要条件に合致するかどうかを検査する。 合致する場合は、トリガー・イベントが生成されます。 関連するプロセス定義オブジェクト内に保持されている情報が、トリガー・メッセージの作成時に使用されます。

3. キュー・マネージャーはトリガー・メッセージを作成し、このアプリケーション・キューに対応する開始キューにそれを書き込む。ただし、この処理が行われるのは、アプリケーション (トリガー・モニター) が入力のために開始キューをオープンする場合だけである。
4. トリガー・モニターは、開始キューからトリガー・メッセージを取り出します。
5. トリガー・モニターは、アプリケーション B (サーバー・アプリケーション) を開始するコマンドを発行します。
6. アプリケーション B はアプリケーション・キューをオープンし、メッセージを取り出します。

注 :

1. 何らかのプログラムによって、アプリケーション・キューが既に入力のためにオープンされていて、FIRST または DEPTH に設定されているトリガー操作を持っている場合、トリガー・イベントが発生することはありません。キューが既に処理されているためです。
2. 入力のために開始キューがオープンされていない場合、キュー・マネージャーはトリガー・メッセージを生成せず、アプリケーションが入力のために開始キューをオープンするまで待ちます。
3. チャネルにトリガー操作を使用する場合は、トリガー・タイプとして FIRST または DEPTH を使用してください。
4. トリガーされたアプリケーションは、トリガー・モニターを開始したユーザー、CICS ユーザー、またはキュー・マネージャーを開始したユーザーのユーザー ID およびグループの下で実行されます。

ここまで、トリガー操作でのキュー間の関係は 1 対 1 対応のみでした。 334 ページの図 69 について考えます。

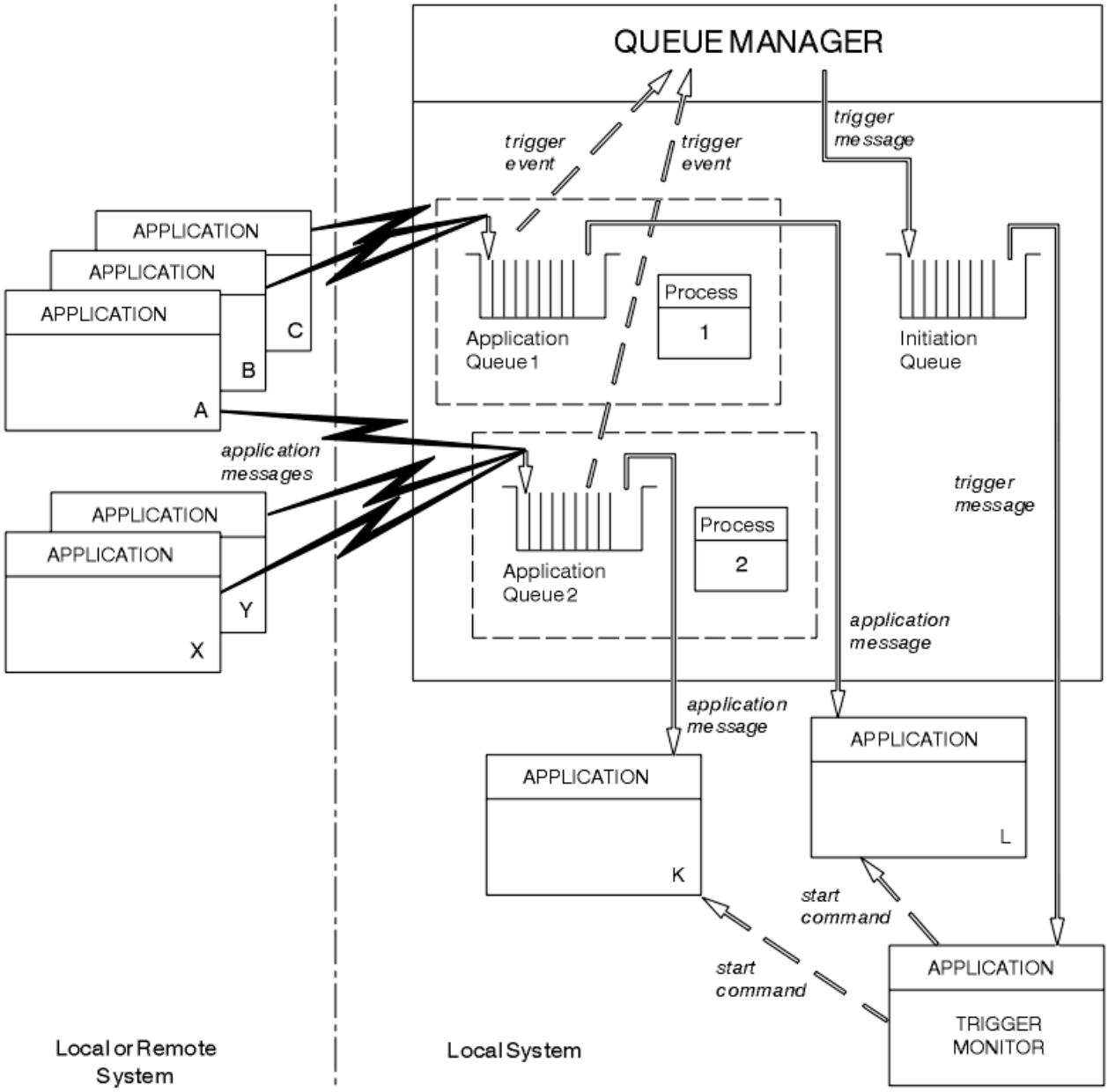


図 69. トリガー操作でのキューの関係

各アプリケーション・キューにはプロセス定義オブジェクトが関連付けられており、そこにメッセージを処理するアプリケーションについての詳細情報が保持されます。 キュー・マネージャーがトリガー・メッセージにこの情報を入れるので、必要な開始キューは 1 つだけです。 トリガー・モニターはこの情報をトリガー・メッセージから抽出し、関連するアプリケーションを開始して、各アプリケーション・キューのメッセージを処理します。

チャネルの開始をトリガーする場合、プロセス定義オブジェクトを定義する必要はないことに注意してください。 この場合は、伝送キュー定義により、トリガーするチャネルを判別できます。

以下のリンクを使用してトリガーを使用した WebSphere MQ アプリケーションの開始について詳しい情報を得ることができます。

- 335 ページの『トリガー操作の前提条件』
- 336 ページの『トリガー・イベントの条件』
- 340 ページの『トリガー・イベントの制御』
- 343 ページの『起動されたキューを使用するアプリケーションの設計』

- 344 ページの『トリガー・モニターによる開始キュー処理』
- 346 ページの『トリガー・メッセージの特性』
- 348 ページの『トリガー操作が動作しないとき』

関連概念

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

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

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

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

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

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

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

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

トリガー操作の前提条件

この情報を使用して、トリガー操作を使用する前に実行するステップについて学びます。

アプリケーションでトリガー操作を利用できるようにするには、以下のステップを実行します。

1. 次のいずれかの場合:
   a. アプリケーション・キューに対応する開始キューを作成する。 以下に例を示します。

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
       LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
       DESCR ('initiation queue description')
```

   または

   b. アプリケーションで使用できる既存のローカル・キュー名 (通常この名前は SYSTEM.DEFAULT.INITIATION.QUEUE、または、トリガー操作でチャネルを開始する場合は SYSTEM.CHANNEL.INITQ) を調べて、アプリケーション・キューの *InitiationQName* フィールドにその名前を指定する。

2. 開始キューをアプリケーション・キューに関連付ける。 1 つのキュー・マネージャーで複数の開始キューを所有することができます。 いくつかのアプリケーション・キューを異なるプログラムで処理したいことがあります。この場合、各処理プログラムに対して 1 つの開始キューを使用することができますが、必ずしもそうする必要はありません。 以下に、アプリケーション・キューの作成例を示します。

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ ('initiation.queue')          +
PROCESS ('process.name')            +
```

```
TRIGGER                                 +
TRIGTYPE (FIRST)
```

3. アプリケーションを起動する場合は、プロセス定義オブジェクトを作成し、アプリケーション・キューを処理するアプリケーションに関する情報を入れる。 例えば、PAYR という CICS 給与計算トランザクションをトリガー操作によって開始するには、次のようにします。

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

   キュー・マネージャーは、トリガー・メッセージを作成すると、プロセス定義オブジェクトの属性から得た情報をトリガー・メッセージにコピーします。

プラットフォーム	プロセス定義オブジェクトを作成する場合
UNIX、 Linux、および Windows システム	DEFINE PROCESS を使用するか、SYSTEM.DEFAULT.PROCESS を使用して、ALTER PROCESS を使用して修正する

4. オプション: 伝送キュー定義を作成し、*ProcessName* 属性にはブランクを使用します。

   TrigData 属性については、起動するチャネルの名前を指定するか、ブランクのままにしてください。 IBM WebSphere MQ for z/OS 以外では、ブランクのままにすると、チャネル・イニシエーターは、指定の伝送キューに関連するチャネルが見つかるまでチャネル定義ファイルを検索します。 キュー・マネージャーはトリガー・メッセージを作成するときに、伝送キュー定義の *TrigData* 属性から得た情報をトリガー・メッセージにコピーします。

5. アプリケーション・キューを処理するアプリケーションのプロパティーを指定するためにプロセス定義オブジェクトを作成した場合は、プロセス・オブジェクトをアプリケーション・キューに関連付ける。 これを行うには、アプリケーション・キューの *ProcessName* 属性にプロセス・オブジェクトの名前を指定します。

プラットフォーム	使用するコマンド
UNIX、 Linux、および Windows システム	ALTER QLOCAL

6. 定義した開始キューを処理するトリガー・モニター のインスタンスを開始する。 詳しくは、344 ページの『トリガー・モニターによる開始キュー処理』を参照してください。

未配布トリガー・メッセージについて知る必要がある場合、キュー・マネージャーに送達不能 (未配布メッセージ) キューが定義されていることを確認してください。 キュー名を *DeadLetterQName* キュー・マネージャー・フィールドに指定してください。

これで、アプリケーション・キューを定義するキュー・オブジェクトの属性を使用して、必要なトリガー条件を設定できます。 詳細については、340 ページの『トリガー・イベントの制御』を参照してください。

トリガー・イベントの条件

このトピックで共用キューという語を用いる場合、それは WebSphere MQ for z/OS でのみ使用可能なキュー共用グループ内の共用キューを意味します。

キュー・マネージャーは、以下の条件が満たされた場合にトリガー・メッセージを作成します。

1. メッセージがキューに書き込まれた 場合。
2. メッセージが、キューのトリガー優先順位のしきい値と同等以上の優先順位を持っている場合。 この優先順位は、*TriggerMsgPriority* ローカル・キュー属性に設定され、それがゼロの場合は、すべてのメッセージが該当します。

3. 優先順位が *TriggerMsgPriority* と同等以上のキューのメッセージ数が、以前に次のような状態であった場合 (*TriggerType* に応じて異なる)。
   - ゼロ (トリガー・タイプ MQTT_FIRST の場合)
   - 任意の数 (トリガー・タイプ MQTT_EVERY の場合)
   - *TriggerDepth* から 1 を引いた値 (トリガー・タイプ MQTT_DEPTH の場合)

   注 :

   a. 非共用ローカル・キューについては、キュー・マネージャーは、トリガー・イベントの条件が存在するかどうかを判断するときに、コミットされたメッセージとコミットされていないメッセージの両方をカウントします。 キューのメッセージがコミットされていないため、アプリケーションが検索を行う対象のメッセージがない場合には、結果的にアプリケーションが開始される場合があります。 この場合には、アプリケーションがメッセージの到着を待つようにするために、適切な *WaitInterval* を指定した待機オプションを使用することを考慮してください。

   b. ローカル共用キューでは、コミットされたメッセージだけがキュー・マネージャーによってカウントされます。
4. タイプ FIRST または DEPTH のトリガー操作については、メッセージを除去するためにオープンされたアプリケーション・キューを持つプログラムがない場合 (つまり、*OpenInputCount* ローカル・キュー属性がゼロ)。

   注 :

   a. 共用キューでは、1 つのキューに対して複数のキュー・マネージャーがトリガー・モニターを実行している場合にのみ、特別な条件が適用されます。 この場合、1 つまたは複数のキュー・マネージャーが、入力用にオープンしたキューを共用している場合は、他のキュー・マネージャー上のトリガー基準は、*TriggerType* MQTT_FIRST、*TriggerMsgPriority* ゼロとして扱われます。 すべてのキュー・マネージャーが入力用のキューをクローズすると、トリガー条件はキュー定義で指定されている条件に戻ります。

      この条件の影響を受けるシナリオの例として、アプリケーション・キュー A のために実行されるトリガー・モニターを含む複数のキュー・マネジャー QM1、QM2、および QM3 が挙げられます。 トリガーの条件を満たすメッセージが A に到着し、開始キューにトリガー・メッセージが生成されます。 QM1 上のトリガー・モニターが、トリガー・メッセージを取得し、アプリケーションをトリガーします。 トリガーされたアプリケーションが、共用入力用にアプリケーション・キューをオープンします。 この時点からアプリケーション・キュー A のトリガー条件が、キュー・マネージャー QM2 および QM3 上で *TriggerType* MQTT_FIRST、および *TriggerMsgPriority* ゼロと評価され、QM1 がアプリケーション・キューをクローズするまで続きます。

   b. 共用キューについては、この条件は各キュー・マネージャーに適用されます。 つまり、あるキュー・マネージャーがキューのトリガー・メッセージを生成するには、そのキュー・マネージャーのキューの *OpenInputCount* がゼロでなければならないということです。 しかし、キュー共用グループ内に MQOO_INPUT_EXCLUSIVE オプションを使用してキューをオープンしているキュー・マネージャーが 1 つでもあれば、キュー共用グループ内のキュー・マネージャーがそのキューのトリガー・メッセージを生成することはありません。

      トリガー条件の評価方法の変更は、トリガーされたアプリケーションが入力用にキューをオープンした場合に生じます。 トリガー・モニターが 1 つのみ実行されているシナリオでは、他のアプリケーションも同様に入力用にアプリケーション・キューをオープンするため、同じ影響を受ける可能性があります。 アプリケーション・キューが、トリガー・モニターによって起動されたアプリケーションによってオープンされたか、その他のアプリケーションによってオープンされたかは関係ありません。他のキュー・マネージャー上で入力用にキューがオープンされることが、トリガー条件を変更する原因となります。
5. WebSphere MQ for z/OS において、アプリケーション・キューの *Usage* 属性の値が MQUS_NORMAL である場合に、そのキューの読み取り要求が禁止されていない場合 (つまり、*InhibitGet* キュー属性が MQQA_GET_ALLOWED)。 また、起動されたアプリケーション・キューが MQUS_XMITQ の *Usage* 属性を持つキューである場合、そのキューに対する読み取り要求は禁止されません。
6. 次のいずれかの場合:

- そのキューの *ProcessName* ローカル・キュー属性がブランクではなく、さらにその属性によって識別されるプロセス定義オブジェクトが作成されている場合。または、
- そのキューの *ProcessName* ローカル・キュー属性がブランクであり、そのキューが伝送キューである場合。 プロセス定義はオプションなので、*TriggerData* 属性には、開始するチャネルの名前が入る場合もあります。 この場合、トリガー・メッセージには次の値を持つ属性が入ります。
  - *QName*: キュー名
  - *ProcessName*: ブランク
  - *TriggerData*: トリガー・データ
  - *ApplType*: MQAT_UNKNOWN
  - *ApplId*: ブランク
  - *EnvData*: ブランク
  - *UserData*: ブランク

7. 開始キューが作成されていて、*InitiationQName* ローカル・キュー属性に指定されている場合。 さらに、次の状態になっている場合。
   - 読み取り要求が開始キューに対して禁止されていない (つまり、*InhibitGet* キュー属性が MQQA_GET_ALLOWED)。
   - その開始キューに対して、書き込み要求が禁止されていてはならない (つまり、*InhibitPut* キュー属性が MQQA_PUT_ALLOWED でなければならない)。
   - その開始キューの *Usage* 属性が MQUS_NORMAL でなければならない。
   - 動的キューがサポートされている環境では、開始キューは、論理的に削除されたとしてマークが付いている動的キューであってはならない。
8. 現在、メッセージの削除のためにオープンされた開始キューがトリガー・モニターにある (つまり、*OpenInputCount* ローカル・キュー属性がゼロより大きい) 場合。
9. アプリケーション・キューに対するトリガー制御 (*TriggerControl* ローカル・キュー属性) が MQTC_ON に設定されている場合。 これを行うには、キューを定義するときに *trigger* 属性を設定するか、ALTER QLOCAL コマンドを使用する。
10. トリガー・タイプ (*TriggerType* ローカル・キュー属性) が MQTT_NONE でない場合。

    必要条件がすべて満たされ、トリガー条件の原因となったメッセージが作業単位の一部として書き込まれる場合、作業単位がコミットされるか、またはトリガー・タイプ MQTT_FIRST の場合にはバックアウトされるかにかかわりなく、作業単位が完了するまではトリガー・モニター・アプリケーションによる取り出しにトリガー・メッセージを利用することはできません。
11. MQTT_FIRST または MQTT_DEPTH の *TriggerType* について、適切なメッセージがそのキューに入る場合。さらにそのキューが、次のどちらかの状態の場合。
    - 以前には空 (MQTT_FIRST) でなかった、または
    - *TriggerDepth* またはさらに他のメッセージ (MQTT_DEPTH) を持っていた

    さらに (条件 337 ページの『3』 を除く) 336 ページの『2』 から 338 ページの『10』 までの条件を満たし、MQTT_FIRST である場合に、このキューに対してトリガー・メッセージが最後に書き込まれてから、十分な時間 (*TriggerInterval* キュー・マネージャー属性) が経過した場合。

    これにより、キュー・サーバーはキューのメッセージをすべて処理する前に終了できます。 トリガー間隔の目的は、重複して生成されるトリガー・メッセージの数を減らすことです。

    注 : キュー・マネージャーをいったん停止して再始動すると、タイマー (*TriggerInterval*) はリセットされます。 2 つのトリガー・メッセージを生成できる間は、小さなウィンドウが表示されます。このウィンドウは、メッセージが届いたときにキューのトリガー属性の設定が有効になっていて、キューが以前は空 (MQTT_FIRST) でなかった場合、あるいは *TriggerDepth* または他のメッセージ (MQTT_DEPTH) をもっていた場合に表示されます。

12. MQTT_FIRST または MQTT_DEPTH の *TriggerType* について、キューを処理する唯一のアプリケーションが MQCLOSE 呼び出しを発行し、少なくとも以下のメッセージが存在する場合。

    - 1 つの (MQTT_FIRST)、または
    - *TriggerDepth* (MQTT_DEPTH)

    これらのメッセージは十分な優先順位 (336 ページの『2』 の条件) のキューにあり、条件 337 ページの『6』 から 338 ページの『10』 も満たされている。

    これにより、キュー・サーバーは MQGET 呼び出しを発行し、キューが空であることを検出し、終了できます。ただし、MQGET 呼び出しと MQCLOSE 呼び出しの間には、1 つまたは複数のメッセージが到着します。

    注 :

    a. アプリケーション・キューを処理するプログラムがすべてのメッセージを取り出さない場合は、閉じたループが発生する可能性があります。 プログラムがキューをクローズするたびに、キュー・マネージャーは、トリガー・モニターにサーバー・プログラムを再度開始させる、別のトリガー・メッセージを作成します。

    b. アプリケーション・キューを処理するプログラムが、キューをクローズする前に、読み取り要求をバックアウトした (あるいは、プログラムが異常終了した) 場合にも、同様のことが起こります。 ただし、読み取り要求をバックアウトする前にプログラムがキューをクローズした場合、そのキューが別の理由で空であれば、トリガー・メッセージは作成されません。

    c. このようなループが発生するのを防ぐには、MQMD の *BackoutCount* フィールドを使用して、繰り返しバックアウトされるメッセージを検出することができます。 詳細については、37 ページの『バックアウトされるメッセージ』を参照してください。

13. MQSET またはコマンドを使用して以下の条件が満たされる場合。

    a. - *TriggerControl* が MQTC_ON に変更される、または
       - *TriggerControl* が既に MQTC_ON であり、*TriggerType*、*TriggerMsgPriority* または *TriggerDepth* (関係する場合) が変更され、

       さらに、少なくとも以下のいずれかのメッセージが存在する場合。

       - 1 つの (MQTT_FIRST か MQTT_EVERY)、または
       - *TriggerDepth* (MQTT_DEPTH)

       これらのメッセージは十分な優先順位 (336 ページの『2』 の条件) のキューにあり、条件 337 ページの『4』 から 338 ページの『10』 (338 ページの『8』 は除く) も満たされている。

       トリガー操作が行われる条件が既に満たされている場合は、上記の条件により、アプリケーションまたはオペレーターがトリガー操作の基準を変更できます。

    b. 開始キューの *InhibitPut* キュー属性が MQQA_PUT_INHIBITED から MQQA_PUT_ALLOWED に変更され、少なくとも次のメッセージが存在する場合。

       - 1 つの (MQTT_FIRST か MQTT_EVERY)、または
       - *TriggerDepth* (MQTT_DEPTH)

       これらのメッセージには、任意のキューのうち、開始キューとなるキューの十分な優先順位 (336 ページの『2』 の条件) があり、条件 337 ページの『4』 から 338 ページの『10』 も満たされている。 (トリガー・メッセージは、条件を満たすキューごとに 1 つ生成される。)

       開始キューでの MQQA_PUT_INHIBITED 条件のためにトリガー・メッセージは生成されませんでしたが、現在はこの条件が変更されています。

    c. アプリケーション・キューの *InhibitGet* キュー属性が、MQQA_GET_INHIBITED から MQQA_GET_ALLOWED に変更され、少なくとも次のメッセージが存在する場合。

       - 1 つの (MQTT_FIRST か MQTT_EVERY)、または
       - *TriggerDepth* (MQTT_DEPTH)

これらのメッセージには、キューに十分な優先順位 (336 ページの『2』 の条件) があり、条件 337 ページの『4』 から 338 ページの『10』 (337 ページの『5』 を除く) も満たされている。

これにより、アプリケーション・キューからメッセージを取り出せる場合にだけそのアプリケーションをトリガーできます。

d. トリガー・モニター・アプリケーションが、開始キューからの入力のために MQOPEN 呼び出しを発行し、少なくとも次のメッセージが存在する場合。

- 1 つの (MQTT_FIRST か MQTT_EVERY)、または
- *TriggerDepth* (MQTT_DEPTH)

これらのメッセージは、任意のアプリケーション・キューのうち、開始キューとなるキューに十分な優先順位 (条件 336 ページの『2』) があり、条件 337 ページの『4』 から 338 ページの『10』 (338 ページの『8』 を除く) も満たされている。さらに、入力のために開始キューをオープンしているアプリケーションがほかにない。(トリガー・メッセージは、条件を満たすキューごとに 1 つ生成される。)

このため、トリガー・モニターが稼働していない間に、メッセージがキューに到着することができ、キュー・マネージャーが再始動し、トリガー・メッセージ (非持続性の) がなくなります。

14. MSGDLVSQ が正しく設定されている場合。 MSGDLVSQ=FIFO を設定した場合、メッセージは先入れ先出し方式でキューに送信されます。 メッセージの優先順位は無視され、キューのデフォルト優先順位がメッセージに割り当てられます。 *TriggerMsgPriority* にキューのデフォルト優先順位よりも高い優先順位を設定した場合、メッセージは起動されません。 *TriggerMsgPriority* にキューのデフォルト優先順位と同等以下の優先順位を設定した場合は、FIRST、EVERY、および DEPTH タイプのときにトリガー操作が実行されます。 これらのタイプの詳細については、340 ページの『トリガー・イベントの制御』の *TriggerType* フィールドの説明を参照してください。

MSGDLVSQ=PRIORITY を設定し、かつメッセージの優先順位が *TriggerMsgPriority* フィールドと同等以上に設定された場合は、メッセージはトリガー・イベントに影響を与える だけです。 この場合は、FIRST、EVERY、および DEPTH タイプのときにトリガー操作が実行されます。 例えば、優先順位が *TriggerMsgPriority* に設定された優先順位より低いメッセージを 100 個書き込んだ場合、トリガー操作用のキューの有効なサイズはゼロのままです。 次に、優先順位が *TriggerMsgPriority* と同等以上に設定された別のメッセージをキューに書き込んだ場合、キューの有効なサイズは 0 から 1 になり、*TriggerType* が FIRST でなければならないという条件が満たされます。

注 :

1. ステップ 339 ページの『12』 以降の場合 (アプリケーション・キューに到着したメッセージ以外の何らかのイベントが発生したことによりトリガー・メッセージが生成された場合)、トリガー・メッセージは作業単位の一部として書き込まれません。 また、*TriggerType* が MQTT_EVERY で、アプリケーション・キューに 1 つ以上のメッセージがある場合には、トリガー・メッセージは 1 つだけ生成されます。
2. WebSphere MQ が MQPUT 中にメッセージをセグメント化する場合は、すべてのセグメントがキューに正常に配置されるまで、トリガー・イベントは処理されません。 ただし、メッセージ・セグメントがキューに配置されると、WebSphere MQ は、トリガー操作を行うために、それらのセグメントを個々のメッセージとして扱います。 例えば、1 つの論理メッセージが 3 つに分割される場合、論理メッセージが最初に MQPUT されてセグメント化されるときには、1 つのトリガー・イベントだけが処理されます。しかし、3 つのそれぞれのセグメントのトリガー・イベントは、それぞれのセグメントが WebSphere MQ ネットワーク内を移動するときに処理されます。

トリガー・イベントの制御

アプリケーション・キューを定義する属性をいくつか使用することにより、トリガー・イベントを制御します。 この情報には、EVERY、FIRST、および DEPTH というトリガー・タイプの使用例も含まれています。

トリガー操作を使用可能にしたり使用禁止にしたりすることも、トリガー・イベントに影響を与えるメッセージの数または優先順位を選択することもできます。 これらの属性の詳細については、オブジェクトの属性に記載されています。

関連する属性は、次のとおりです。

TriggerControl
この属性を使用すると、アプリケーション・キューに対してトリガー操作を使用可能にしたり、使用禁止にしたりできます。

TriggerMsgPriority
メッセージがトリガー・イベントに影響を与えるために必要な最低の優先順位。
TriggerMsgPriority より優先順位の低いメッセージがアプリケーション・キューに到着した場合、キュー・マネージャーは、トリガー・メッセージを作成するかどうかを決定するときにそのメッセージを無視します。 *TriggerMsgPriority* 属性をゼロに設定した場合は、すべてのメッセージがトリガー・イベントに影響を与えます。

TriggerType
トリガー・タイプ NONE (これは、 *TriggerControl* を OFF に設定したのと同様に、トリガー操作を使用不可にする) のほかに、以下のトリガー・タイプを使用して、トリガー・イベントに対するキューの感知性を設定できます。

EVERY	メッセージがアプリケーション・キューに到着するたびに、トリガー・イベントを発生させます。 アプリケーションの複数のインスタンスを開始させたいときに、このトリガー・タイプを使用します。
FIRST	アプリケーション・キュー上のメッセージの数が 0 から 1 に変更されるときだけ、トリガー・イベントを発生させます。 処理プログラムを、最初のメッセージがキューに到着したときに開始させ、処理するメッセージがなくなるまで続行させてから終了したい場合に、このトリガー・タイプを使用します。 このキューは、必ず空になるまで処理しなければなりません。 342 ページの『トリガー・タイプ FIRST の特別な場合』も参照してください。
DEPTH	トリガー・イベントは、アプリケーション・キュー上のメッセージの数が *TriggerDepth* 属性の値に達したときにのみ発生します。 このトリガー・タイプの典型的な使用例は、一連の要求に対してすべての応答を受信したときにプログラムを開始させる場合です。 **DEPTH によるトリガー操作 :** 項目数によるトリガー操作では、キュー・マネージャーは、トリガー・メッセージを作成した後、(< xph> < pv>TriggerControl< /pv> < /xph> 属性を使用して) トリガー操作を使用不可にします。 アプリケーションは、この後、(MQSET 呼び出しによって) トリガー操作自体を再度使用可能にしなければなりません。 トリガー操作を使用禁止にする処置は、同期点制御の下では行われないので、作業単位をバックアウトしただけではトリガー操作を再度使用可能にすることはできません。 プログラムがトリガー・イベントを生じた書き込み要求をバックアウトしたか、あるいはプログラムが異常終了した場合は、MQSET 呼び出しまたは ALTER QLOCAL コマンドを使用してトリガー操作を再度使用可能にしなければなりません。

TriggerDepth
DEPTH によるトリガー操作を使用するときに、トリガー・イベントを発生させるキュー上のメッセージ数。

キュー・マネージャーがトリガー・メッセージを作成するために満たす必要のある条件は、336 ページの『トリガー・イベントの条件』に記述されています。

トリガー・タイプ EVERY の使用例

自動車保険の請求を生成するアプリケーションを考えてみてください。 アプリケーションは、毎回同じ応答先キューを指定して、数多くの保険会社に請求メッセージを送信します。 この応答先キューにトリガー・タイプ EVERY を設定して、応答が到着するたびに、応答を処理するサーバーのインスタンスが起動されるように設定できます。

トリガー・タイプ FIRST の使用例

多数の支店を持つ企業で、毎日の業務の詳細を本社あてに送信する場合を考えてみてください。 すべての支店が一日の作業終了時にこれを同時に行い、本社には全支店からの詳細データを処理するアプリケーションがあります。 本社に到着する最初のメッセージは、このアプリケーションを開始させるトリガー・イベントを発生させることができます。 このアプリケーションは、そのキューにメッセージがなくなるまで処理を続けます。

トリガー・タイプ DEPTH の使用例

飛行機の予約確認、ホテルの予約確認、レンタカーの手配、さらにトラベラーズ・チェックの注文を行うための単一の要求を作成する旅行代理店のアプリケーションを想定します。 アプリケーションは、これらの項目を 4 つの要求メッセージに分けて、各メッセージを別々の宛先に送信できます。 その応答先キューにトリガー・タイプ DEPTH を (値を 4 に) 設定して、4 つの応答がすべて到着したときにだけ再始動するように設定できます。

4 つの応答のうち最後の応答より前に別のメッセージ (おそらく別の要求からのメッセージ) が応答先キューに到着した場合は、要求しているアプリケーションが早めに起動されてしまいます。 これを避けるため、DEPTH トリガーを使用して 1 つの要求に対する複数の応答を収集する場合は、各要求について常に新規の応答先キューを使用してください。

トリガー・タイプ FIRST の特別な場合

トリガー・タイプが FIRST の場合、アプリケーション・キューに別のメッセージが到着したときに既にメッセージがあると、キュー・マネージャーは通常は別のトリガー・メッセージを作成しません。

しかし、キューを処理するアプリケーションは、実際にはキューをオープンしない場合があります (例えば、アプリケーションがシステムの問題で終了する場合があります)。 正しくないアプリケーション名がプロセス定義オブジェクトに書き込まれている場合には、キューを処理するアプリケーションはメッセージをまったく取り上げません。 これらの場合、別のメッセージがアプリケーション・キューに到着する場合は、このメッセージ (およびキューの他のメッセージ) の処理のために稼働しているサーバーはありません。

これに対処するため、キュー・マネージャーは、以下の状況ではさらにトリガー・メッセージを作成します。

- アプリケーション・キューに別のメッセージが到着した場合。ただしこれは、キュー・マネージャーがそのキューに対するトリガー・メッセージを最後に作成してから事前に定義された時間が経過している場合だけです。 この時間は、キュー・マネージャー属性の *TriggerInterval* で定義されます。 デフォルト値は 999 999 999 ミリ秒です。
- WebSphere MQ for z/OS では、開かれている開始キューを指名したアプリケーション・キューが定期的に走査されます。 前回のトリガー・メッセージの送信から *TRIGINT* ミリ秒が経過しており、キューがトリガー・イベントの条件を満たしていて、CURDEPTH がゼロより大きい場合は、トリガー・メッセージが生成されます。 この処理は、バック・ストップ・トリガー操作と呼ばれます。

アプリケーションで使用するトリガー間隔の値を決めるときには、以下の点を考慮してください。

- *TriggerInterval* の値を低く設定している場合で、アプリケーション・キューにサービスを提供しているアプリケーションがない場合は、トリガー・タイプ FIRST がトリガー・タイプ EVERY のような振る舞いをすることがあります。 これは、メッセージがアプリケーション・キューに書き込まれる速度に依存し、その速度は他のシステム活動に依存することがあります。 これは、トリガー間隔が非常に小さいと、トリガー・タイプが (EVERY ではなく) FIRST であっても、メッセージがアプリケーション・キューに書き込まれるたびに別のトリガー・メッセージが生成されるからです。 (トリガー間隔がゼロのトリガー・タイプ FIRST は、トリガー・タイプ EVERY と同等です。)
- WebSphere MQ for z/OS において、*TRIGINT* の値を低く設定している場合で、トリガー・タイプ FIRST のアプリケーション・キューにサービスを提供しているアプリケーションがない場合は、開かれている開始キューを指名したアプリケーション・キューの定期的な走査が行われるたびに、バック・ストップ・トリガー操作によってトリガー・メッセージが生成されます。
- 作業単位がバックアウトされる場合 (トリガー・メッセージと 作業単位を参照)、トリガー間隔が高い値 (またはデフォルト値) に設定されていると、 その作業単位のバックアウト時にトリガー・メッセージが 1 つ生成されます。 しかし、トリガー間隔を小さい値かゼロに設定している (トリガー・タイプ FIRST が

トリガー・タイプ EVERY のように機能する) 場合は、多数のトリガー・メッセージが生成される可能性があります。 その作業単位がバックアウトされると、すべてのトリガー・メッセージがやはり使用可能になります。 生成されるトリガー・メッセージの数は、トリガー間隔によって異なります。 トリガー間隔がゼロに設定されている場合、最大数のメッセージが生成されます。

起動されたキューを使用するアプリケーションの設計

これまでは、アプリケーションに対してトリガー操作を設定し、制御する方法を検討してきました。 ここでは、アプリケーションを設計するときに考慮すべきいくつかのヒントを示します。

トリガー・メッセージと作業単位

作業単位の一部ではないトリガー・イベントのために作成されたトリガー・メッセージは、作業単位の外部で開始キューに書き込まれます。その際、他のメッセージには依存せず、トリガー・モニターによる検索のために即時に利用可能になります。

作業単位に属するトリガー・イベントによって作成されたトリガー・メッセージは、その UOW が解決されたとき (作業単位がコミットまたはバックアウトされたとき) に、開始キューで取得可能になります。

キュー・マネージャーが開始キューにトリガー・メッセージを書き込むことに失敗した場合、トリガー・メッセージは送達不能 (未配布メッセージ) キューに書き込まれます。

注 :

1. キュー・マネージャーは、トリガー・イベントの条件が存在するかどうかを判断するときに、コミットされたメッセージとコミットされていないメッセージの両方をカウントします。

   タイプ FIRST または DEPTH のトリガー操作を使用すると、作業単位がバックアウトされた場合でもトリガー・メッセージが使用可能になります。これにより、必要な条件が満たされたとき、いつでもトリガー・メッセージが使用可能になります。 例えば、トリガー・タイプ FIRST で起動されるキューに対して、作業単位内で書き込み要求が行われたと想定します。 これによって、キュー・マネージャーはトリガー・メッセージを作成します。 別の作業単位から別の書き込み要求が行われても、さらに別のトリガー・イベントは発生しません。アプリケーション・キューのメッセージ数が 1 から 2 に変わったので、トリガー・イベントの条件を満たさないためです。 ここで、最初の作業単位がバックアウトされ、2 番目がコミットされたとしても、やはりトリガー・メッセージは作成されます。

   しかし、これは、トリガー・イベントの条件が満たされない ときにも、トリガー・メッセージが作成される場合があることを意味します。 トリガー操作を使用するアプリケーションは、この状態を処理する用意を常にしておく必要があります。 MQGET 呼び出しで待機オプションを使用して、*WaitInterval* を適切な値に設定することをお勧めします。

   作成されたトリガー・メッセージは、作業単位がバックアウトされた場合でもコミットされた場合でも、常に取得可能になります。
2. 共用ローカル・キュー (つまり、キュー共用グループ内の共用キュー) については、キュー・マネージャーはコミットされたメッセージだけをカウントします。

起動されたキューからのメッセージの読み取り

トリガー操作を使用するアプリケーションを設計するときは、トリガー・モニターがプログラムを開始してから、アプリケーション・キューで他のメッセージが使用可能になるまでの間に遅延が生じる場合があることに注意してください。 これは、トリガー・イベントを発生するメッセージが他のメッセージより前にコミットされたときに起こることがあります。

メッセージが到着するまでの時間を考慮して、トリガー条件が設定されるキューからメッセージを除去する MQGET 呼び出しを使用するときは、必ず待機オプションを使用してください。 *WaitInterval* は、メッセージが書き込まれ、その書き込み呼び出しがコミットされるまでの妥当な最長時間を考慮した十分な値にしなければなりません。 メッセージがリモート・キュー・マネージャーから送られてくる場合は、この時間は次の値によって影響を受けます。

- コミットされる前に書き込まれるメッセージの数
- 通信リンクの速度と使用可能度

- メッセージのサイズ

待機オプションを指定した MQGET 呼び出しを使用するのが望ましい状態の例として、作業単位を説明したときと同じ例を考えてみます。 この例は、トリガー・タイプ FIRST で起動されるキューに対する、作業単位内の書き込み要求でした。 このイベントにより、キュー・マネージャーはトリガー・メッセージを作成します。 別の作業単位から別の書き込み要求が行われても、さらに別のトリガー・イベントは発生しません。アプリケーション・キューのメッセージ数が 0 から 1 に変わっていないためです。 ここで、最初の作業単位がバックアウトされ、2 番目がコミットされたとしても、やはりトリガー・メッセージは作成されます。 したがって、トリガー・メッセージは最初の作業単位がバックアウトされたときに作成されます。 2 番目のメッセージがコミットされるまでにかなりの遅延がある場合は、起動されたアプリケーションが待機しなければならない場合があります。

タイプ DEPTH のトリガー操作を使用すると、関連したメッセージがすべてコミットされた場合でも、遅延が生じることがあります。 *TriggerDepth* キュー属性の値が 2 であるとします。 2 つのメッセージがキューに到着すると、2 番目のメッセージによってトリガー・メッセージが作成されます。 しかし、2 番目のメッセージが最初にコミットされる場合は、その時点でトリガー・メッセージが使用可能になります。 トリガー・モニターはサーバー・プログラムを開始させますが、プログラムは、最初のメッセージがコミットされるまで 2 番目のメッセージしか検索できません。 このため、最初のメッセージが使用可能になるまで、プログラムは待機しなければならない場合があります。

待機期間が満了したときに取り出すメッセージがない場合は、アプリケーションが終了するような設計にしておく必要があります。 1 つ以上のメッセージが後から到着した場合は、アプリケーションが再トリガーされた時点でそれらのメッセージが処理されることになります。 この方式によって、アプリケーションがアイドル状態になって、不必要にリソースを使用することのないようにします。

トリガー・モニターによる開始キュー処理

キュー・マネージャーにとって、トリガー・モニターは、キューを処理する他のアプリケーションのような存在です。 ただし、トリガー・モニターは開始キューを提供します。

トリガー・モニターは、通常、絶えず稼働しているプログラムです。 トリガー・メッセージが開始キューに到着すると、トリガー・モニターはそのメッセージを検索します。 トリガー・モニターは、メッセージ内の情報を使用して、アプリケーション・キューにあるメッセージを処理するアプリケーションを開始させるコマンドを実行します。

トリガー・モニターは、それによって開始されるプログラムに十分な情報を渡して、そのプログラムが適正なアプリケーション・キューに適正な処置を行うことができるようにしなければなりません。

チャネル・イニシエーターは、メッセージ・チャネル・エージェントに応答する特別なタイプのトリガー・モニターの例です。 しかし、この場合には、トリガー・タイプ FIRST または DEPTH のどちらかを使用する必要があります。

UNIX および *Windows* システム上のトリガー・モニター

このトピックでは、UNIX システムおよび Windows システム用に提供されるトリガー・モニターについて説明します。

次のトリガー・モニターがサーバー環境用に提供されています。

amqstrg0
これは、**runmqtrm** によって提供される機能のサブセットを提供するサンプル・トリガー・モニターです。 amqstrg0 についての詳細は、96 ページの『分散プラットフォームにおけるサンプル・プログラム』を参照してください。

runmqtrm
このコマンドの構文は **runmqtrm** [*-m QMgrName*] [*-q InitQ*] です。ここで、QMgrName はキュー・マネージャー、 InitQ は開始キューです。 デフォルト・キューは、デフォルト・キュー・マネージャーの SYSTEM.DEFAULT.INITIATION.QUEUE です。 該当するトリガー・メッセージ用のプログラムを呼び出します。 このトリガー・モニターは、デフォルトのアプリケーション・タイプをサポートします。

トリガー・モニターからオペレーティング・システムに渡されるコマンド・ストリングは、次のように作成されます。

1. 関連する PROCESS 定義 (作成される場合) の *ApplId*
2. 二重引用符で囲まれた MQTMC2 構造体
3. 関連する PROCESS 定義 (作成される場合) の *EnvData*

ApplId は実行するプログラムの名前を示します。この名前は、コマンド行に入力されたとおりに表示されます。

渡されるパラメーターは、MQTMC2 文字構造体です。 二重引用符に囲まれたこのストリングと正確に同じストリングを持つコマンド・ストリングが呼び出されます。したがって、システム・コマンドには 1 つのパラメーターとして受け付けられます。

トリガー・モニターは、開始したばかりのアプリケーションが完了するまでは、開始キューに別のメッセージがあるかどうかを検査しません。 アプリケーションに多くの処理がある場合は、到着するトリガー・メッセージの数にトリガー・モニターが追いつかないことがあります。 この解決方法は、次の 2 つです。

- 実行するトリガー・モニターを増やす
- 開始済みのアプリケーションのバックグラウンドで実行する

実行するトリガー・モニターの数を増やすと、一度に実行できるアプリケーションの最大数を制御できます。 アプリケーションをバックグラウンドで実行する場合は、WebSphere MQ による、実行可能なアプリケーションの数に関する制約はありません。

Windows システム上で、開始済みのアプリケーションをバックグラウンドで稼働させるには、*ApplId* フィールド内でアプリケーション名の前に START コマンドを追加します。 以下に例を示します。

```
START ?B AMQSECHA
```

UNIX システムで、開始されたアプリケーションをバックグラウンドで実行するには、PROCESS 定義の *EnvData* の末尾に & を配置します。

注 : Windows のパスで、パス名の中にスペースが含まれている場合は、パスを引用符 (") で囲んで、それが 1 つの引数として扱われるようにする必要があります。 例えば、" C:\Program Files\Application Directory\Application.exe"などです。

以下は、パス中のファイル名にスペースが含まれている場合の APPLICID ストリングの例です。

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

この例の Windows START コマンドの構文には、二重引用符で囲まれた空のストリングがあります。 START コマンドは、引用符で囲われた最初の引数を新しいコマンドのタイトルとして扱うように指定します。 Windows がアプリケーションのパスを「タイトル」の引数と間違えないようにするため、コマンドのアプリケーション名の前に、二重引用符で囲ったタイトルのストリングを追加します。

以下のトリガー・モニターが、WebSphere MQ クライアント用に提供されています。

runmqtmc
これは、リンク先が WebSphere MQ MQI クライアント・ライブラリーである点を除いて runmqtrm と同じです。

CICS の場合
amqltmc0 トリガー・モニターが CICS 用に提供されています。 このトリガー・モニターは標準トリガー・モニター runmqtrm と同様に機能しますが、これを異なる方法で実行して、 CICS トランザクションを起動します。

このトピックは Windows、UNIX、および Linux システムにのみ適用されます。

これは CICS プログラムとして提供され、4 文字のトランザクション名で定義する必要があります。 4 文字の名前を入力して、トリガー・モニターを始動します。 デフォルトのキュー・マネージャー (qm.ini ファイル、または WebSphere MQ for Windows の場合はレジストリーで指定)、および SYSTEM.CICS.INITIATION.QUEUE。

異なるキュー・マネージャーまたはキューを使用したい場合は、トリガー・モニター MQTMC2 構造体を作成します。これを行う場合、構造体はパラメーターとして追加するには長すぎるため、EXEC CICS START 呼び出しを使用してプログラムを作成する必要があります。 そこで、MQTMC2 構造体をデータとしてトリガー・モニター用の START 要求に渡します。

MQTMC2 構造体を使用する場合、それ以外のフィールドを参照しないため、トリガー・モニターに入力する必要があるのは、*StrucId*、*Version*、*QName*、および *QMgrName* パラメーターだけです。

メッセージは開始キューから読み取られ、EXEC CICS START を使用して CICS トランザクションの開始に使用されます。この際、トリガー・メッセージの APPL_TYPE は MQAT_CICS と想定しています。 開始キューからのメッセージの読み取りは、 CICS 同期点制御の下で実行されます。

モニターの始動時、停止時、およびエラーの発生時に、メッセージが生成されます。 これらのメッセージは、CSMT 一時データ・キューに送られます。

以下に、トリガー・モニターの提供バージョンを示します。

バージョン	**以下を使用してください。**
amqltmc0	TXSeries for AIX、HP-UX、および Sun Solaris バージョン 5.1
amqltmc4	TXSeries for Windows バージョン 5.1
amqltmcc	CICS トリガー・モニターのクライアント・バインド・バージョン

他の環境用のトリガー・モニターが必要な場合は、キュー・マネージャーが開始キューに書き込むトリガー・メッセージを処理できるプログラムを作成してください。 このようなプログラムは、以下の操作を実行する必要があります。

1. メッセージが開始キューに到着するのを待機するために、MQGET 呼び出しを使用する。
2. 開始するアプリケーションの名前と、実行される環境を検索するために、トリガー・メッセージの MQTM 構造体のフィールドを調べる。
3. 環境に特有な開始コマンドを実行する。
4. 必要に応じて、MQTM 構造体を MQTMC2 構造体に変換する。
5. MQTMC2 または MQTM 構造体を開始済みのアプリケーションに渡す。 これにはユーザーのデータが含まれる場合がある。
6. アプリケーション・キューを、そのキューを処理するアプリケーションに対応させる。 これを行うには、キューの *ProcessName* 属性にプロセス定義オブジェクト (作成した場合) の名前を指定してください。

トリガー・モニター・インターフェースの詳細については、MQTMC2 を参照してください。

トリガー・メッセージの特性

以下のトピックでは、トリガー・メッセージのその他の特性をいくつか説明します。

- 346 ページの『トリガー・メッセージの持続性と優先順位』
- 347 ページの『キュー・マネージャーの再始動とトリガー・メッセージ』
- 347 ページの『トリガー・メッセージとオブジェクト属性の変更』
- 347 ページの『トリガー・メッセージの形式』

トリガー・メッセージの持続性と優先順位

トリガー・メッセージは、持続性を持たせる必要はないので、持続的ではありません。

しかし、トリガー・イベントを生成する条件は持続的なので、これらの条件が満たされたときはいつでも、トリガー・メッセージが生成されます。 トリガー・メッセージが失われる場合でも、アプリケーション・キューのアプリケーション・メッセージはそのまま存在するので、条件がすべて満たされた場合にはすぐに、キュー・マネージャーがトリガー・メッセージを生成します。

作業単位がロールバックされる場合、その作業単位によって生成されるトリガー・メッセージはどれでも常に送達されます。

トリガー・メッセージの優先順位は、開始キューのデフォルト優先順位になります。

キュー・マネージャーの再始動とトリガー・メッセージ

キュー・マネージャーの再始動に続いて、入力のために開始キューが次にオープンされると、トリガー・メッセージは、それに対応するアプリケーション・キューにメッセージが存在し、かつトリガー操作のために定義されている場合に、この開始キューに書き込まれます。

トリガー・メッセージとオブジェクト属性の変更

トリガー・メッセージは、トリガー・イベントの発生時に効力があるトリガー属性の値に従って作成されます。

(生成原因となったメッセージが作業単位内で書き込まれたため) トリガー・モニターがすぐにトリガー・メッセージを使用できない場合には、その間にトリガー属性が変更されても、トリガー・メッセージには影響がありません。 特に、トリガー操作を使用禁止にした場合でも、いったん生成されたトリガー・メッセージは、使用禁止にはできません。 また、トリガー・メッセージが使用可能になった時点では、アプリケーション・キューが既に存在していない場合もあります。

トリガー・メッセージの形式

トリガー・メッセージの形式は、MQTM 構造体によって定義されます。

これは、以下のフィールドを持ち、これらのフィールドは、キュー・マネージャーがトリガー・メッセージを作成するとき、アプリケーション・キューおよびそのキューと関連する処理のオブジェクト定義にある情報を使用して書き込まれます。

StrucId
構造体の ID。

Version
構造体のバージョン。

QName
トリガー・イベントが発生したアプリケーション・キューの名前。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの *QName* 属性を使用して、このフィールドに値を入れます。

ProcessName
アプリケーション・キューに関連したプロセス定義オブジェクトの名前。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの *ProcessName* 属性を使用して、このフィールドに値を入れます。

TriggerData
トリガー・モニターが開始させるアプリケーションを識別する文字ストリング。 キュー・マネージャーがトリガー・メッセージを作成するときに、アプリケーション・キューの *TriggerData* 属性を使用して、このフィールドに値を入れます。 WebSphere MQ 製品 (WebSphere MQ for z/OS を除く) では、このフィールドを使用して、トリガーされるチャネルの名前を指定できます。

ApplType
トリガー・モニターが開始させるアプリケーションのタイプ。 キュー・マネージャーがトリガー・メッセージを生成するとき、これは *ProcessName* で識別されるプロセス定義オブジェクトの *ApplType* 属性を使用して、このフィールドに値を入れます。

ApplId
トリガー・モニターが開始されるアプリケーションを識別する文字ストリング。 キュー・マネージャーがトリガー・メッセージを生成するとき、これは *ProcessName* で識別されるプロセス定義オブジェクトの *ApplId* 属性を使用して、このフィールドに値を入れます。 WebSphere MQ for z/OS が提供するトリガー・モニター CKTI または CSQQTRMN を使用する場合、プロセス定義オブジェクトの *ApplId* 属性は、CICS または IMS トランザクション ID です。

EnvData
トリガー・モニターが使用する環境関連データを含む文字フィールド。 キュー・マネージャーがトリガー・メッセージを生成するとき、これは *ProcessName* で識別されるプロセス定義オブジェクトの *EnvData* 属性を使用して、このフィールドに値を入れます。 WebSphere MQ for z/OS が提供するトリガー・モニター (CKTI または CSQQTRMN) は、このフィールドを使用しませんが、他のトリガー・モニターは使用することを選択する場合があります。

UserData
トリガー・モニターが使用するユーザー・データを含む文字フィールド。 キュー・マネージャーがトリガー・メッセージを生成するとき、これは *ProcessName* で識別されるプロセス定義オブジェクトの *UserData* 属性を使用して、このフィールドに値を入れます。 このフィールドは、起動されるチャネルの名前を指定するために使用できます。

トリガー・メッセージ構造体の詳細については、MQTM を参照してください。

トリガー操作が動作しないとき

トリガー・モニターがプログラムを開始できないか、またはキュー・マネージャーがトリガー・メッセージを送達できない場合は、プログラムは起動されません。 例えば、プロセス・オブジェクトのアプリケーション ID はプログラムがバックグラウンドで開始されるように指定する必要がありますが、これが指定されていない場合、トリガー・モニターはプログラムを開始できません。

トリガー・メッセージが作成されても開始キューに書き込むことができない (例えば、キューが満ぱいであるか、トリガー・メッセージの長さがその開始キューに指定されているメッセージの最大長を超えているため) 場合、そのトリガー・メッセージは代わりに送達不能 (未配布メッセージ) キューに書き込まれます。

送達不能キューへの PUT 操作を正常に完了できない場合は、トリガー・メッセージは廃棄され、警告メッセージが z/OS コンソールまたはシステム・オペレーターに送られるか、エラー・ログに書き込まれます。

トリガー・メッセージを送達不能キューに書き込むと、そのキューにトリガー・メッセージが生成される場合があります。 この 2 番目のトリガー・メッセージは、メッセージを送達不能キューに追加すると、破棄されます。

プログラムが正常に起動された場合でも、プログラムがキューからメッセージを受け取る前に異常終了した場合は、トレース・ユーティリティー (例えば、プログラムが CICS で動作している場合には CICS AUXTRACE) を使用して、障害の原因を調べてください。

MQI とクラスターの処理

呼び出しと戻りコードには、クラスター化に関連する特殊なオプションがあります。

クラスターとともに使用できる呼び出しや戻りコードで使用可能なオプションの詳細については、 以下のリンクを参照してください。

- 349 ページの『MQOPEN およびクラスター』
- 350 ページの『MQPUT、MQPUT1、およびクラスター』
- 350 ページの『MQINQ およびクラスター』
- 351 ページの『MQSET およびクラスター』
- 352 ページの『戻りコード』

関連概念

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

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

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

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

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

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

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

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

MQOPEN およびクラスター

メッセージの書き込み先、または読み取り元のキューは、クラスター・キューがオープンされるときには、MQOPEN 呼び出しに依存します。

ターゲット・キューの選択

オブジェクト記述子 MQOD にキュー・マネージャー名を指定しない場合、キュー・マネージャーは、メッセージ送信先のキュー・マネージャーを選択します。 オブジェクト記述子にキュー・マネージャー名を指定した場合、メッセージは常に、選択したキュー・マネージャーに送信されます。

キュー・マネージャーがターゲット・キュー・マネージャーを選択している場合、その選択は、バインディング・オプション MQOO_BIND_* と、ローカル・キューが存在するかどうかによって異なります。 キューのローカル・インスタンスがある場合は、CLWLUSEQ 属性が ANY に設定されている場合を除き、常にリモート・インスタンスよりも先にオープンされます。 それ以外の場合、その選択はバインディング・オプションに依存します。 クラスターで メッセージ・グループ を使用する場合は、グループ内のすべてのメッセージが同じ宛先で処理されるように、 MQOO_BIND_ON_OPEN または MQOO_BIND_ON_GROUP のいずれかを指定する必要があります。

キュー・マネージャーがターゲット・キュー・マネージャーを選択している場合、ワークロード管理アルゴリズムを使用して、「ラウンドロビン」方式で選択します。ワークロード・バランシングを参照してください。

MQOO_BIND_ON_OPEN
MQOPEN 呼び出しの MQOO_BIND_ON_OPEN オプションは、ターゲット・キュー・マネージャーを固定することを指定します。 クラスター内に同じキューのインスタンスが複数ある場合は、MQOO_BIND_ON_OPEN オプションを使用します。 MQOPEN 呼び出しから返されるオブジェクト・ハンドルを指定する、キューに入るすべてのメッセージは、同じキュー・マネージャーに送られます。

- メッセージに類縁性がある場合は、MQOO_BIND_ON_OPEN オプションを使用します。 例えば、メッセージのバッチをすべて同じキュー・マネージャーで処理する場合は、キューを開くときに MQOO_BIND_ON_OPEN を指定します。 IBM WebSphere MQ は、そのキューに入ったすべてのメッセージで実行対象となるキュー・マネージャーおよびルートを固定します。
- MQOO_BIND_ON_OPEN オプションを指定した場合は、選択の対象となるキューの新しいインスタンス用にそのキューを再オープンする必要があります。

MQOO_BIND_NOT_FIXED
MQOPEN 呼び出しの MQOO_BIND_NOT_FIXED オプションは、ターゲット・キュー・マネージャーを固定しないことを指定します。 MQOPEN 呼び出しから返されるオブジェクト・ハンドルを指定する、キューに書き込まれるメッセージは、MQPUT の実行時にメッセージごとにキュー・マネージャーへの経路が指定されます。 すべてのメッセージを強制的に同じ宛先に書き込むことはしない場合は、MQOO_BIND_NOT_FIXED オプションを使用します。

- MQOO_BIND_NOT_FIXED と MQMF_SEGMENTATION_ALLOWED は、同時には指定しないでください。 そのようにすると、メッセージのセグメントが異なるキュー・マネージャーに送達され、クラスター全体に分散される可能性があります。

MQOO_BIND_ON_GROUP
アプリケーションが、メッセージのグループが同じ宛先インスタンスに割り当てられるように要求できるようにします。 このオプションは、キューの場合にのみ有効であり、クラスター・キューにのみ影響します。 クラスター・キューではないキューに対して指定された場合、このオプションは無視されます。

- MQPUT で MQPMO_LOGICAL_ORDER が指定されている場合、グループは、1 つの宛先にのみ経路指定されます。 MQOO_BIND_ON_GROUP が指定されても、メッセージが特定のグループに属していなければ、BIND_NOT_FIXED の動作が代わりに使用されます。

MQOO_BIND_AS_Q_DEF
MQOO_BIND_ON_OPEN、MQOO_BIND_NOT_FIXED、および MQOO_BIND_ON_GROUP のいずれも指定しない場合、デフォルト・オプションは MQOO_BIND_AS_Q_DEF です。 MQOO_BIND_AS_Q_DEF を使用すると、キュー・ハンドルに使用するバインディングは、DefBind キュー属性から取られます。

MQOPEN オプションの関連性

MQOPEN が成功するには、 MQOPEN オプション MQOO_BROWSE, MQOO_INPUT_*または MQOO_SET にクラスター・キューのローカル・インスタンスが必要です。

MQOPEN のオプションである MQOO_OUTPUT、MQOO_BIND_*、および MQOO_INQUIRE は、成功させるためにクラスター・キューのローカル・インスタンスを必要としません。

解決されたキュー・マネージャー名

MQOPEN の実行時にキュー・マネージャー名が解決されると、解決された名前がアプリケーションに返されます。 アプリケーションが以降の MQOPEN 呼び出しでこの名前の使用を試行する場合、アプリケーションがその名前へのアクセスを許可されていないことが分かる場合があります。

MQPUT、MQPUT1、およびクラスター

MQOPEN で MQOO_BIND_NOT_FIXED が指定された場合、ワークロード管理ルーチンは、MQPUT または MQPUT1 のどちらの宛先を選択するか決めます。

MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED が指定された場合、以降の MQPUT 呼び出しはそれぞれワークロード管理ルーチンを呼び出して、メッセージ送信先のキュー・マネージャーを決定できます。 宛先と経路はメッセージごとに選択されます。 メッセージが入った後でネットワークの状態が変わった場合は、宛先と経路が変わることがあります。 MQPUT1 呼び出しは常に、MQOO_BIND_NOT_FIXED が有効であると見なして動作します。 つまり、常にワークロード管理ルーチンを呼び出します。

ワークロード管理ルーチンがキュー・マネージャーを選択すると、ローカル・キュー・マネージャーは PUT 操作を完了します。 メッセージは異なる複数のキューに入れることができます。

1. 宛先がキューのローカル・インスタンスである場合、メッセージはローカル・キューに入ります。
2. 宛先がクラスター内のキュー・マネージャーである場合、メッセージはクラスター伝送キューに入ります。
3. 宛先がクラスター外部のキュー・マネージャーである場合、メッセージは、ターゲット・キュー・マネージャーと同じ名前を持つ伝送キューに入ります。

MQOPEN 呼び出しで MQOO_BIND_ON_OPEN が指定された場合、MQPUT 呼び出しは、宛先および経路が既に選択されているため、ワークロード管理ルーチンを呼び出しません。

MQINQ およびクラスター

どのクラスター・キューが照会の対象となるかは、MQOO_INQUIRE と組み合わせるオプションによって決まります。

キューに対して照会する前に、MQOPEN 呼び出しを使用してキューを開き、MQOO_INQUIRE を指定します。

クラスター・キューに対して照会を実行するには、MQOPEN 呼び出しを使用し、他のオプションと MQOO_INQUIRE を組み合わせます。 照会できる属性は、クラスター・キューのローカル・インスタンスがあるかどうか、およびキューがどのようにオープンしているかによって決まります。

- MQOO_BROWSE、MQOO_INPUT_*、または MQOO_SET を MQOO_INQUIRE と組み合わせる場合、オープンを成功させるには、クラスター・キューのローカル・インスタンスが必要です。 ローカル・インスタンスがあれば、ローカル・キューに有効なすべての属性を照会することができます。
- MQOO_OUTPUT を MQOO_INQUIRE と組み合わせて、 前述のオプションを他に何も指定しないと、オープンするインスタンスは次のいずれかになります。
  - ローカル・キュー・マネージャー上にあるインスタンス (もしある場合)。 ローカル・インスタンスがあれば、ローカル・キューに有効なすべての属性を照会することができます。
  - ローカル・キュー・マネージャー・インスタンスがない場合は、クラスター内の他の場所にあるインスタンス。 この場合は、次の属性のみ照会できます。 このときの QType 属性の値は MQQT_CLUSTER です。
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

クラスター・キューの DefBind 属性を照会するには、MQINQ 呼び出しにセレクター MQIA_DEF_BIND を指定して使用します。 返される値は、MQBND_BIND_ON_OPEN または MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP のいずれかです。 クラスターでグループを使用する場合は、MQBND_BIND_ON_OPEN または MQBND_BIND_ON_GROUP のいずれかを指定する必要があります。

キューのローカル・インスタンスの CLUSTER および CLUSNL 属性に対して照会を実行するには、MQINQ 呼び出しにセレクター MQCA_CLUSTER_NAME またはセレクター MQCA_CLUSTER_NAMELIST を指定して使用します。

注 : MQOPEN がバインドするキューを固定せずにクラスター・キューをオープンすると、MQINQ 呼び出しが連続して出されて、クラスター・キューのさまざまなインスタンスに対して照会が実行される可能性があります。

関連概念

220 ページの『クラスター・キュー用の MQOPEN オプション』

キュー・ハンドルに使用されるバインディングは *DefBind* キュー属性から取得され、値は MQBND_BIND_ON_OPEN、MQBND_BIND_NOT_FIXED、または MQBND_BIND_ON_GROUP を取ることができます。

MQSET およびクラスター

MQOPEN オプションの MQOO_SET オプションには、クラスター・キューのローカル・インスタンスが必要です。これがなければ MQSET は成功しません。

MQSET 呼び出しを使用して、クラスター内の別の場所にあるキューの属性を設定することはできません。

クラスター属性で定義されたローカルの別名またはリモート・キューを開き、MQSET 呼び出しを使用することができます。 ローカルの別名またはリモート・キューの属性は、設定できます。 ターゲット・キューが別のキュー・マネージャーで定義されたクラスター・キューの場合でも問題になりません。

戻りコード

クラスター固有の戻りコード

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューのオープン、またはそこへのメッセージの書き込みのために発行されます。 キュー・マネージャーの ClusterWorkloadExit 属性によって定義されるクラスター・ワークロード出口は、予期せず失敗したり、時間内に応答しなかったりします。

WebSphere MQ for z/OS 上のシステム・ログにメッセージが書き込まれ、このエラーに関する詳細情報が提供されます。

このキュー・ハンドルに対する今後の MQOPEN、MQPUT、および MQPUT1 呼び出しは、ClusterWorkloadExit 属性がブランクであるものとして処理されます。

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

z/OS では、クラスター・ワークロード出口をロードできません。

システム・ログにメッセージが書き込まれ、ClusterWorkloadExit 属性はブランクと見なされて処理が続行されます。

z/OS 以外のプラットフォームでは、キュー・マネージャーに接続するために、MQCONN 呼び出しまたは MQCONNX 呼び出しが発行されます。 キュー・マネージャーの ClusterWorkloadExit 属性によって定義されるクラスター・ワークロード出口はロードすることができないため、呼び出しに失敗します。

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

有効な MQOO_OUTPUT オプションおよび MQOO_BIND_ON_OPEN オプションを指定した MQOPEN 呼び出しは、クラスター・キューに対して発行されます。 クラスター内のキューのインスタンスはすべて、InhibitPut 属性を MQQA_PUT_INHIBITED に設定することにより、現行では書き込みが禁止されています。 メッセージを受信するために使用可能なキュー・インスタンスがないため、MQOPEN 呼び出しに失敗します。

この理由コードは、次の条件が両方とも存在する場合にだけ戻ります。

- キューのローカル・インスタンスがない。 ローカル・インスタンスがあれば、 そのローカル・インスタンスが書き込み禁止になっていても MQOPEN 呼び出しは成功する。
- キューのクラスター・ワークロード出口がないか、またはクラスター・ワークロード出口はあるが、キュー・インスタンスを選択していない。 (クラスター・ワークロード出口がキュー・インスタンスを選択する場合には、たとえそのインスタンスが書き込みを禁止されていても、MQOPEN 呼び出しが成功します。)

MQOO_BIND_NOT_FIXED オプションが MQOPEN 呼び出しで指定されている場合、クラスター内のすべてのキューが書き込み禁止になっている場合であっても、呼び出しが成功することがあります。 ただし、後続の MQPUT 呼び出しは、この呼び出しの時点ですべてのキューが引き続き書き込み禁止になっている場合には、失敗する可能性があります。

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューのオープン、またはそこへのメッセージの書き込みのために発行されます。 フル・リポジトリー・キュー・マネージャーからの応答が必要であるのに応答がないため、キューの定義を正しく解決できません。
2. MQOPEN、MQPUT、MQPUT1 または MQSUB 呼び出しは、PUBSCOPE(ALL) または SUBSCOPE(ALL) を指定したトピック・オブジェクトに対して発行されます。 完全リポジトリー・キュー・マネージャーからの応答が必要ですが、使用可能な応答がないため、クラスター・トピック定義を正しく解決できません。

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、クラスター・キューに対して発行されます。 クラスタリングに必要なリソースを使用しようとしてエラーが発生しました。

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

MQPUT 呼び出しまたは MQPUT1 呼び出しは、クラスター・キューにメッセージを書き込むために発行されます。 呼び出し時に、クラスター内にキューのインスタンスはまったく存在しません。 MQPUT は失敗し、メッセージは送信されません。

キューをオープンする MQOPEN 呼び出しで MQOO_BIND_NOT_FIXED が指定された場合、または MQPUT1 を使用してメッセージを書き込んだ場合に、このエラーが発生する可能性があります。

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

MQOPEN、MQPUT、または MQPUT1 の呼び出しは、メッセージをオープンするか、クラスター・キューにメッセージを書き込むために発行されます。 クラスター・ワークロード出口は呼び出しを拒否します。

クライアント・アプリケーションの作成

WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

WebSphere MQ クライアント環境では、アプリケーションを作成して実行できます。 アプリケーションは、作成して、使用する WebSphere MQ MQI クライアントにリンクする必要があります。 アプリケーションを作成およびリンクする方法は、使用しているプラットフォームおよびプログラミング言語に応じて異なります。 クライアント・アプリケーションの作成方法については、359 ページの『WebSphere MQ MQI クライアント用のアプリケーションの作成』を参照してください。

特定の条件を満たす場合には、WebSphere MQ のフル環境と WebSphere MQ MQI クライアント環境の両方で、コードを変換せずに WebSphere MQ アプリケーションを実行できます。 WebSphere MQ クライアント環境でのアプリケーションの実行については、361 ページの『IBM WebSphere MQ MQI クライアント環境でのアプリケーションの実行』を参照してください。

メッセージ・キュー・インターフェース (MQI) を使用して、WebSphere MQ MQI クライアント環境で実行するアプリケーションを作成する場合は、MQI 呼び出し時に強制的に WebSphere MQ アプリケーションの処理が中断されないようにする追加の制御があります。 これらの制御については、354 ページの『クライアント・アプリケーションでのメッセージ・キュー・インターフェース (MQI) の使用』を参照してください。

その他のアプリケーション・タイプをクライアント・アプリケーションとして準備および実行する方法については、以下のトピックを参照してください。

- 373 ページの『CICS および Tuxedo アプリケーションの準備と実行』
- 40 ページの『Microsoft Transaction Server アプリケーションの準備と実行』
- 375 ページの『WebSphere MQ JMS アプリケーションの準備と実行』

関連概念

8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

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

96 ページの『WebSphere MQ プログラムのサンプル』
この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

193 ページの『キューイング・アプリケーションの作成』

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

950 ページの『WebSphere MQ での Web サービスの使用』
IBM WebSphere MQ transport for SOAP または IBM WebSphere MQ bridge for HTTP を使用して、Web サービス用の IBM WebSphere MQ アプリケーションを開発できます。

277 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
パブリッシュ/サブスクライブ WebSphere MQ アプリケーションの作成を開始します。

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

553 ページの『プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

クライアント・アプリケーションでのメッセージ・キュー・インターフェース (MQI) の使用

この一連のトピックでは、WebSphere MQ MQI クライアント環境で実行する場合と WebSphere MQ キュー・マネージャー環境で実行する場合で、WebSphere MQ アプリケーションの作成にどのような違いがあるかについて説明します。

アプリケーションを設計するときは、MQI 呼び出し中に加える必要がある制御を考慮して、WebSphere MQ アプリケーションの処理が中断されないようにしてください。

クライアント・アプリケーションでのメッセージ・サイズの制限制限

キュー・マネージャーにはメッセージの最大長がありますが、クライアント・アプリケーションから送信できるメッセージの最大サイズは、チャネル定義により制限されます。

キュー・マネージャーの最大メッセージ長 (MaxMsgLength) 属性は、そのキュー・マネージャーが処理できるメッセージの最大長です。

z/OS 以外のプラットフォームでは、キュー・マネージャーの最大メッセージ長属性の値を大きくすることができます。 詳しくは、ALTER QMGR を参照してください。

キュー・マネージャーの MaxMsgLength の値は、MQINQ 呼び出しを使用して調べることができます。

MaxMsgLength 属性を変更した場合、新しい値より大きい長さのキュー、さらにはメッセージがすでにあるかどうかの検査は行われません。 この属性を変更したら、変更内容が有効になるよう、アプリケーションとチャネルを再始動してください。 これが完了すると、キュー・マネージャーまたはキューいずれかの MaxMsgLength を超える新規メッセージを生成することはできなくなります (キュー・マネージャーのセグメント化が許可されている場合を除きます)。

チャネル定義内の最大メッセージ長によって、クライアント接続で転送できるメッセージのサイズが制限されます。 WebSphere MQ アプリケーションが MQPUT 呼び出しまたは MQGET 呼び出しを使用して、これより大きいメッセージを処理しようとすると、アプリケーションにエラー・コードが戻されます。 チャネル定義の最大メッセージ・サイズ・パラメーターは、クライアント接続を介し、MQCB を使用してコンシュームされる最大メッセージ・サイズには影響しません。

クライアントまたはサーバーのコード化文字セット ID (CCSID) の選択

クライアント用のローカル CCSID を使用します。 キュー・マネージャーは、必要な変換を実行します。 MQCCSID 環境変数を使用して、CCSID をオーバーライドします。 アプリケーションが複数の PUT を実行する場合、最初の PUT の完了後に MQMD の CCSID およびエンコード・フィールドを上書きできます。

MQI を介してアプリケーションからクライアント・スタブに渡されるデータは、WebSphere MQ MQI クライアント用にエンコードされた、ローカル CCSID のデータでなければなりません。 接続されたキュー・マネージャーがデータの変換を要求する場合、その変換はキュー・マネージャーのクライアント・サポート・コードによって行われます。

ただし、キュー・マネージャーで変換できない場合は、V7 の Java クライアントで行うことができます。 674 ページの『 WebSphere MQ classes for Java クライアント接続』を参照してください。

クライアント・コードでは、クライアントの MQI を介する文字データは、そのワークステーション用に構成された CCSID のデータであると想定されます。 この CCSID がサポートされていない CCSID であるか、必要とされる CCSID ではない場合、以下のコマンドのいずれかを使用して、この CCSID を MQCCSID 環境変数でオーバーライドできます。

- Windows の場合:

```
SET MQCCSID=850
```

- UNIX システムの場合:

```
export MQCCSID=850
```

このパラメーターがプロファイル内に設定される場合、すべての MQI データは、コード・ページ 850 のデータであると見なされます。

注 : コード・ページ 850 についての想定は、メッセージ内のアプリケーション・データには適用されません。

メッセージ記述子 (MQMD) の後ろに WebSphere MQ ヘッダーがある複数の PUT をアプリケーションが実行する場合は、最初の PUT の完了後に MQMD の CCSID およびエンコード・フィールドが上書きされることに注意してください。

最初の PUT の後、これらのフィールドには、接続済みのキュー・マネージャーが WebSphere MQ ヘッダーを変換するために使用する値が入っています。 アプリケーションがこれらの値を必要な値にリセットするようにしてください。

クライアント・アプリケーションでの MQINQ の使用

MQINQ を使用して照会される値の一部は、クライアント・コードによって変更されます。

CCSID
これは、キュー・マネージャーの CCSID ではなく、クライアントの CCSID に設定されます。

MaxMsgLength
この値は、チャネル定義によって制限されると低減されます。 これは、次の値のうち低いほうの値になります。

- キュー定義で定義された値
- チャネル定義で定義された値

詳しくは、MQINQ を参照してください。

クライアント・アプリケーションでの同期点調整の使用

ベース・クライアント上で稼働するアプリケーションは MQCMIT および MQBACK を発行できますが、同期点制御の有効範囲は MQI リソースに限定されます。 外部トランザクション・マネージャーを拡張トランザクション・クライアントと共に使用できます。

WebSphere MQ 内におけるキュー・マネージャーの役割の 1 つは、アプリケーション内の同期点制御です。アプリケーションが WebSphere MQ ベース・クライアント上で稼働している場合、そのアプリケーションは MQCMIT および MQBACK を発行できますが、同期点制御の有効範囲は MQI リソースに限定されます。WebSphere MQ verb MQBEGIN は、ベース・クライアント環境では無効です。

サーバー側で全機能を使用できるキュー・マネージャー環境で実行されるアプリケーションは、トランザクション・モニターを介して複数のリソース (データベースなど) を調整できます。 サーバー側では、WebSphere MQ 製品に付属のトランザクション・モニター、または他のトランザクション・モニター (CICS など) を使用できます。 ベース・クライアント・アプリケーションでトランザクション・モニターを使用することはできません。

外部トランザクション・マネージャーは、WebSphere MQ 拡張トランザクション・クライアントと共に使用できます。 拡張トランザクション・クライアントの概要を参照してください。 (詳細について記載されています。)

クライアント・アプリケーションでの先読みの使用

クライアントで先読みを使用することによって、クライアント・アプリケーションがメッセージを要求しなくても非永続メッセージがクライアントに送信されるようにすることができます。

クライアントは、サーバーからのメッセージを必要とする際、サーバーに向けて要求を送信します。 クライアントは、コンシュームするメッセージごとに要求を別々に送信します。 こうした要求メッセージを送信しなくても良いようにして、クライアントの非永続メッセージのコンシュームのパフォーマンスを改善するために、クライアントを先読みを使用するように構成できます。 先読みにより、アプリケーションからの要求がなくてもクライアントへのメッセージ送信が可能となります。

先読みを使用すると、クライアント・アプリケーションから非永続メッセージをコンシュームする際のパフォーマンスを改善することができます。 このパフォーマンスの改善は、MQI アプリケーションと JMS アプリケーションの両方で使用することができます。 MQGET または非同期コンシュームを使用するクライアント・アプリケーションでは、非永続メッセージをコンシュームする際にパフォーマンスが向上するという利点があります。

MQOO_READ_AHEAD を使用して MQOPEN を呼び出すときに、特定の条件が満たされている場合にのみ、WebSphere MQ クライアントは先読みを使用可能にします。 それらの条件には、以下のものが含まれます。

- クライアントとリモート・キュー・マネージャーの両方が、WebSphere MQ バージョン 7 以降でなければなりません。
- クライアント・アプリケーションは、スレッド化された WebSphere MQ MQI クライアント・ライブラリーに対してコンパイルおよびリンクされている必要があります。
- クライアント・チャネルが TCP/IP プロトコルを使用している必要があります。
- チャネルでは、クライアントとサーバー両方のチャネル定義で、SharingConversations (SHARECNV) がゼロ以外に設定されていなければなりません。

先読みが使用可能な場合、メッセージはクライアント上の先読みバッファーというメモリーのバッファーに送信されます。 クライアントには、先読みが使用可能でオープンされた各キューごとに先読みバッファーがあります。 先読みバッファー内のメッセージは非永続メッセージです。 クライアントは定期的に、消費したデータ量に関する更新情報をサーバーに提供します。

先読みの使用はすべてのオプションでサポートされているわけではないので、クライアント・アプリケーションの設計によっては、先読みの使用が適していない場合があります。 先読みが使用可能になる際、オプションによっては、MQGET 呼び出し間で一貫性の確保が求められます。 クライアントが MQGET 呼び出し間に選択基準を変更した場合、先読みバッファーに格納されているメッセージはそのクライアントの先読みバッファー内に保留されます。 詳細については、257 ページの『非永続メッセージのパフォーマンス向上』を参照してください。

先読みの構成は、WebSphere MQ クライアント構成ファイルの MessageBuffer スタンザで指定されている MaximumSize、PurgeTime、および UpdatePercentage という 3 つの属性によって制御されます。

クライアント・アプリケーションでの非同期書き込みの使用

非同期書き込みを使用すると、アプリケーションはキュー・マネージャーからの応答を待たずにキューにメッセージを書き込むことができます。 これを使用して、メッセージングのパフォーマンスを向上できる場合があります。

通常、アプリケーションは MQPUT または MQPUT1 を使用して 1 つ以上のメッセージをキューに書き込むと、キュー・マネージャーがその MQI 要求を処理したことを確認するのを待たなければなりません。 メッセージの非同期書き込みを選択することにより、特にクライアント・バインディングを使用するアプリケーションや、多数の小さいメッセージをキューに書き込むアプリケーションのメッセージングのパフォーマンスを向上させることができます。 アプリケーションにメッセージを非同期的に書き込ませる場合、キュー・マネージャーは、呼び出しのたびに成功や失敗を返しません。その代わりに、周期的にエラーの確認を行うことができます。

メッセージをキューに非同期に書き込むには、MQPMO 構造体の *Options* フィールドの MQPMO_ASYNC_RESPONSE オプションを使用します。

非同期書き込みに適格でないメッセージは、キューに同期的に書き込まれます。

MQPUT または MQPUT1 で非同期書き込み応答を要求するときに出される CompCode および MQCC_OK および MQRC_NONE の Reason は、必ずしもメッセージがキューに正常に書き込まれたことを意味するものではありません。 MQPUT 呼び出しまたは MQPUT1 呼び出しごとの成功または失敗が、すぐには戻されないことがありますが、非同期呼び出しで最初に発生したエラーは、MQSTAT への呼び出しにより後から判別できます。

MQPMO_ASYNC_RESPONSE の詳細については、MQPMO オプションを参照してください。

非同期書き込みサンプル・プログラムで、使用可能ないくつかの機能を示しています。 このプログラムの機能および設計の詳細、および実行方法については、114 ページの『非同期書き込みサンプル・プログラム』を参照してください。

クライアント・アプリケーションでの共用会話の使用

共用会話が使用できる環境では、会話は MQI チャネル・インスタンスを共用できます。

共用会話は 2 つのフィールドで制御されます。両方とも SharingConversations という名前で、1 つはチャネル定義 (MQCD) 構造体の一部であり、もう 1 つはチャネル出口パラメーター (MQCXP) 構造体の一部です。 MQCD の SharingConversations フィールドの値は整数で、チャネルと関連付けられた 1 つのチャネル・インスタンスを共用できる会話の最大数を決定します。 MQCXP の SharingConversations フィールドの値はブール値で、チャネル・インスタンスが現在共用されているかを示します。

共用会話が使用できない環境では、同一の MQCD を指定する新規クライアント接続でチャネル・インスタンスは共用されません。

新規クライアント・アプリケーション接続では、以下の条件に当てはまる場合にチャネル・インスタンスが共用されます。

- 共用会話用に、チャネル・インスタンスのクライアント接続側とサーバー接続側の両方が構成され、これらの値がチャネル出口でオーバーライドされない。
- クライアント接続 MQCD 値 (クライアント MQCONNX 呼び出しで、またはクライアント・チャネル定義テーブル (CCDT) から提供される値) は、既存のチャネル・インスタンスが最初に確立された時点で、クライアント MQCONNX 呼び出しで、または CCDT から提供されたクライアント接続 MQCD 値と完全に一致する。 元の MQCD が後から出口またはチャネル・ネゴシエーションで変更された可能性がありますが、変更が加えられる前にクライアント・システムに提供された値とは一致しています。
- サーバー・サイドでの共用会話の制限を超えていない。

新規クライアント・アプリケーション接続が、他の会話とのチャネル・インスタンスの共用を実行する際の基準に一致している場合は、この決定は、その会話で任意の出口が呼び出される前に行われます。 このような会話の出口で、チャネル・インスタンスが他の会話と共用されることには変わりはありません。 新規チャネル定義に一致する既存のチャネル・インスタンスがない場合は、新規チャネル・インスタンスが接続されます。

チャネル・ネゴシエーションが行われるのは、チャネル・インスタンス上の最初の会話に対してのみです。チャネル・インスタンスのネゴシエーション値はその段階で固定され、それ以降の会話開始時に変更することはできません。 TLS/SSL 認証も最初の会話に対してのみ行われます。

チャネル・インスタンスのクライアント接続側またはサーバー接続側のいずれかのソケットでの最初の会話に対する、任意のセキュリティー出口、送信出口または受信出口の初期設定中に、MQCD SharingConversations 値が変更された場合は、これらの出口がすべて初期設定された後に得られた新規の値が、チャネル・インスタンスの共用会話の値を決定する際に使用されます (最低値が優先されます)。

共用会話のネゴシエーション値がゼロの場合、チャネル・インスタンスは絶対に共用されません。 また、このフィールドをゼロに設定する出口プログラムも同じように各自のチャネル・インスタンス上で実行されます。

共用会話のネゴシエーション値がゼロより大きい場合は、MQCXP SharingConversations が後続の出口呼び出しについて TRUE に設定されます。これは、このチャネル・インスタンス上で、その他の出口プログラムが同時に実行可能になることを意味します。

チャネル出口プログラムを作成する場合は、共用会話に関与する可能性のあるチャネル・インスタンス上で実行するかどうかを考慮してください。 チャネル・インスタンスが共用会話に関与する可能性がある場合は、MQCD フィールドの変更による、チャネル出口の他のインスタンスに対する影響を考慮してください。すべての共用会話全体で、すべての MQCD フィールドは共通の値を持っています。 チャネル・インスタンスを確立した後で、出口プログラムが MQCD フィールドの変更を試行した場合、問題が発生する可能性があります。これは、チャネル・インスタンス上で実行中の出口プログラムの別のインスタンスが同時に同じフィールドの変更を試行している場合があるからです。 この状態が出口プログラムで発生する可能性がある場合は、出口コードの MQCD へのアクセスを直列化する必要があります。

会話を共用するために定義されているチャネルの処理をしているが、特定のチャネル・インスタンス上で共用は行わない場合は、チャネル・インスタンス上の最初の会話でのチャネル出口を初期設定する際に、SharingConversations の MQCD 値を 1 または 0 に設定してください。 SharingConversations 値の説明については、SharingConversations を参照してください。

例

共用会話は有効です。

出口プログラムを指定するクライアント接続チャネル定義を使用しています。

このチャネルをはじめて開始する場合は、初期設定時に出口プログラムが一部の MQCD パラメーターを変更します。 これらはチャネルの影響を受けるため、実行中のチャネルで使用している定義は現在、提供された元の定義とは異なります。 MQCXP SharingConversations パラメーターは TRUE に設定されています。

次回、このチャネルを使用してアプリケーションを接続する際に、会話は、同じ元のチャネル定義が使用されるため、以前開始されたチャネル・インスタンスで実行されます。 アプリケーションが 2 番目に接続するチャネル・インスタンスは、最初に接続したインスタンスと同じです。 この結果、アプリケーションは出口プログラムによって変更された定義を使用します。 2 番目の会話で出口プログラムが初期設定されると、MQCD フィールドは変更できますが、チャネルの影響は受けません。 これらの同じ特性が、チャネル・インスタンスを共用するすべての後続の会話に適用されます。

MQCONNX の使用法

MQCONNX 呼び出しを使用して、MQCNO 構造内のチャネル定義 (MQCD) 構造体を指定することができます。

これにより、呼び出し側のクライアント・アプリケーションは、実行時にクライアント接続チャネルの定義を指定できます。 詳細については、MQCONNX 呼び出しでの MQCNO 構造体の使用を参照してください。 MQCONNX の使用時にサーバーで発行される呼び出しは、サーバーのレベルおよびリスナー構成によって異なります。

MQCONNX をクライアントから使用すると、以下のオプションは無視されます。

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

使用できる MQCD 構造体は、使用する MQCD のバージョン番号によって異なります。 MQCD バージョン (MQCD_VERSION) について詳しくは、MQCD バージョンを参照してください。 MQCD 構造体を使用して、例えば、チャネル出口プログラムをサーバーに渡すことができます。 MQCD バージョン 3 以降を使用している場合は、その構造体を使用して、出口の配列をサーバーに渡すことができます。 この機能を使用して、既存の出口を変更するのではなく操作ごとの出口を追加することにより、同一メッセージに対して暗号化と圧縮などの複数の操作を実行できます。 MQCD 構造体の中の配列を指定しない場合は、単一出口のフィールドが検査されます。 チャネル出口プログラムの詳細については、398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

MQCONNX の共用接続ハンドル

共有接続ハンドルを使用して、同じプロセス内の異なるスレッド間でハンドルを共有することができます。

共用接続ハンドルを指定すると、MQCONNX 呼び出しから戻された接続ハンドルを、プロセス内の任意のスレッドの後続の MQI 呼び出しに渡すことができます。

注 : WebSphere MQ MQI クライアント上の共用接続ハンドルを使用して、共用接続ハンドルをサポートしていないサーバー・キュー・マネージャーに接続することができます。

詳細については、358 ページの『MQCONNX の使用法』を参照してください。

WebSphere MQ MQI クライアント用のアプリケーションの作成

アプリケーションは、WebSphere MQ MQI クライアント環境で作成および実行することができます。 アプリケーションは、作成して、使用する WebSphere MQ MQI クライアントにリンクする必要があります。 アプリケーションを作成およびリンクする方法は、使用しているプラットフォームおよびプログラミング言語に応じて異なります。

アプリケーションがクライアント環境で実行されている場合は、アプリケーションを以下の表に示された言語で作成できます。

表 47. クライアント環境でサポートされているプログラミング言語

クライアント・プラットフォーム	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Yes	Yes	Yes			
HP Integrity NonStop Server	Yes		Yes	Yes		
HP-UX	Yes	Yes	Yes			
Linux	Yes	Yes	Yes			
Solaris	Yes	Yes	Yes			
Windows	Yes	Yes	Yes			Yes

これらの言語でクライアント・アプリケーションをリンクまたは作成するための指示については、関連トピックを参照してください。

C アプリケーションと WebSphere MQ MQI クライアント・コードとのリンク

WebSphere MQ MQI クライアント上で実行する WebSphere MQ アプリケーションを作成したら、そのアプリケーションをキュー・マネージャーにリンクする必要があります。

アプリケーションをキュー・マネージャーにリンクするには、次の 2 つの方法があります。

1. 直接リンクする方法。この場合、キュー・マネージャーがアプリケーションと同じワークステーション上にある必要があります。
2. クライアント・ライブラリー・ファイルにリンクする方法。この場合ユーザーは、同じまたは異なるワークステーション上にあるキュー・マネージャーにアクセスできます。

WebSphere MQ には、次のような各環境用のクライアント・ライブラリー・ファイルが用意されています。

AIX
libmqic.a ライブラリー (スレッド化されていないアプリケーション用)、または libmqic_r.a ライブラリー (スレッド化されたアプリケーション用)。

HP-UX
libmqic.sl ライブラリー (スレッド化されていないアプリケーション用)、または libmqic_r.sl ライブラリー (スレッド化されたアプリケーション用)。

Linux
libmqic.so ライブラリー (スレッド化されていないアプリケーション用)、または libmqic_r.so ライブラリー (スレッド化されたアプリケーション用)。

Solaris
libmqic.so.

Solaris 用の WebSphere MQ MQI クライアントのみがインストールされたワークステーション上でプログラムを使用する場合は、そのプログラムを次のように再コンパイルしてクライアント・ライブラリーにリンクする必要があります。

```
$ /opt/SUNWspro/bin/cc -o <prog> <prog> c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

パラメーターは、示されているとおりに正しい順序で入力しなければなりません。

Windows
MQIC32.LIB.

C++ アプリケーションと WebSphere MQ MQI クライアント・コードとのリンク

C++ 内のクライアント上で実行するアプリケーションを作成できます。構築方法は環境によって異なります。

C++ アプリケーションのリンク方法について詳しくは、WebSphere MQ C++ プログラムの作成を参照してください。

C++ の使用に関するすべての詳細な説明は、C++ の使用を参照してください。

COBOL アプリケーションと IBM WebSphere MQ MQI クライアント・コードとのリンク

IBM WebSphere MQ MQI クライアント上で実行する COBOL アプリケーションを作成した後、そのアプリケーションを適切なライブラリーにリンクする必要があります。

IBM WebSphere MQ には、次のような各環境用のクライアント・ライブラリー・ファイルがあります。

AIX
COBOL アプリケーションを、ライブラリー libmqicb.a (スレッド化されていないアプリケーション用)、または libmqicb_r.a (スレッド化されたアプリケーション用) とリンクします。

HP-UX
COBOL アプリケーションを、ライブラリー libmqicb.sl (スレッド化されていないアプリケーション用)、または libmqicb_r.sl (スレッド化されたアプリケーション用) とリンクします。

Linux
COBOL アプリケーションを、ライブラリー libmqicb.so (スレッド化されていないアプリケーション用)、または libmqicb_r.so (スレッド化されたアプリケーション用) とリンクします。

Solaris
COBOL アプリケーションを、ライブラリー libmqicb.so (スレッド化されていないアプリケーション用)、または libmqicb_r.so (スレッド化されたアプリケーション用) とリンクします。

Windows
アプリケーション・コードを、32 ビット COBOL 用の MQICCBB ライブラリーとリンクします。
Windows 用の IBM WebSphere MQ MQI クライアントは、16 ビット COBOL をサポートしていません。

Visual Basic アプリケーションと WebSphere MQ MQI クライアント・コードとのリンク

Visual Basic アプリケーションを WebSphere MQ MQI クライアント・コード (Windows) にリンクすることができます。

Visual Basic アプリケーションを以下の組み込みファイルにリンクします。

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
PCF コマンド

CMQXB.bas
チャネル

Visual Basic コンパイラーで、クライアントに対して mqtype=2 と設定して、次のクライアント DLL が正しく自動選択されるようにします。

MQIC32.dll
Windows 2000、 Windows XP および Windows 2003

IBM WebSphere MQ MQI クライアント環境でのアプリケーションの実行

特定の条件を満たす場合には、IBM WebSphere MQ のフル環境と IBM WebSphere MQ MQI クライアント環境の両方で、コードを変換せずに IBM WebSphere MQ アプリケーションを実行できます。

その条件は、次のとおりです。

- アプリケーションが複数のキュー・マネージャーに同時に接続する必要がない場合
- MQCONN または MQCONNX 呼び出しで、キュー・マネージャー名の先頭にアスタリスク (*) が付いていない場合
- アプリケーションは、「IBM WebSphere MQ MQI クライアント上で実行されるアプリケーション」にリストされている例外を使用する必要はありません。

注 : どの環境でアプリケーションを実行しなければならないかは、リンク・エディット時に使用するライブラリーによって決まります。

IBM WebSphere MQ MQI クライアント環境で作業するときは、次の点に注意してください。

- IBM WebSphere MQ MQI クライアント環境で実行されるアプリケーションは、サーバーに対してそれぞれ固有の接続を行っています。 アプリケーションは、MQCONN または MQCONNX 呼び出しを発行するたびにサーバーに対して接続を 1 つ確立します。
- アプリケーションは、メッセージを同期的に送受信します。 このことは、クライアント側で呼び出しを発行してからネットワークを介して完了コードおよび理由コードが戻されるまでに待ち時間があることを暗黙に示しています。
- すべてのデータ変換は、サーバーが実行します。なお、マシンで構成された CCSID のオーバーライドについては、MQCCSID も参照してください。

キュー・マネージャーへの IBM WebSphere MQ MQI クライアント・アプリケーションの接続

IBM WebSphere MQ MQI クライアント環境で実行中のアプリケーションは、さまざまな方法でキュー・マネージャーに接続することができます。 環境変数、MQCNO 構造、またはクライアント定義テーブルを使用できます。

IBM WebSphere MQ クライアント環境で実行中のアプリケーションが MQCONN 呼び出しまたは MQCONNX 呼び出しを発行すると、クライアントは、アプリケーションが接続を確立する方法を識別します。 MQCONNX 呼び出しが、IBM WebSphere MQ クライアントでアプリケーションから発行されると、MQI クライアント・ライブラリーは次の順序でクライアント・チャネル情報を検索します。

1. MQCNO 構造体の *ClientConnOffset* または *ClientConnPtr* フィールドの内容 (提供されている場合) を使用します。 これらのフィールドは、クライアント接続チャネルの定義として使用するチャネル定義構造体 (MQCD) を指定します。 接続詳細は、接続前出口を使用してオーバーライドできます。 詳しくは、424 ページの『リポジトリーから接続前出口を使用した接続定義の参照』を参照してください。
2. MQSERVER 環境変数が設定されていると、その環境変数で定義されているチャネルが使用されます。
3. mqclient.ini ファイルが定義され、ServerConnectionParms が含まれている場合、そのファイルが定義するチャネルが使用されます。 詳しくは、構成ファイルを使用したクライアントの構成およびクライアント構成ファイルの CHANNELS スタンザを参照してください。

4. MQCHLLIB 環境変数および MQCHLTAB 環境変数が設定されていると、これらの環境変数が示すクライアント・チャネル定義テーブルが使用されます。
5. `mqclient.ini` ファイルが定義されており、ChannelDefinitionDirectory 属性および ChannelDefinitionFile 属性が含まれている場合、これらの属性は、クライアント・チャネル定義テーブルを見つけるために使用されます。 詳しくは、構成ファイルを使用したクライアントの構成およびクライアント構成ファイルの CHANNELS スタンザを参照してください。
6. 最後に、環境変数が設定されない場合、クライアントは、`mqs.ini` ファイル内の `DefaultPrefix` で設定されたパスと名前でクライアント・チャネル定義テーブルを検索します。 クライアント定義テーブルの検索が失敗した場合、クライアントは次のパスを使用します。
   - UNIX and Linux システム: `/var/mqm/AMQCLCHL.TAB`
   - Windows: `C:\Program Files\IBM\Websphere MQ\amqclchl.tab`

以前のリストで説明された最初のオプション (MQCNO の *ClientConnOffset* または *ClientConnPtr* フィールドを使用したもの) は MQCONNX 呼び出しによってのみサポートされています。 アプリケーションが MQCONNX ではなく MQCONN を使用している場合、チャネル情報の検索は、リストに示されている順番で残りの 5 つの方法で行われます。 クライアントがチャネル情報を見つけられなかった場合、MQCONN または MQCONNX 呼び出しは失敗します。

MQCONN 呼び出しまたは MQCONNX 呼び出しが成功するためには、チャネル名 (クライアント接続用) が、サーバー側に定義されているサーバー接続チャネル名と一致していなければなりません。

アプリケーションから MQRC_Q_MGR_NOT_AVAILABLE という戻りコードが戻り、エラー・ログ・ファイルに「`AMQ9517 - File damaged (AMQ9517 - ファイルが壊れています)`」というエラー・メッセージが記録された場合は、移行およびクライアント・チャネル定義テーブル (CCDT) を参照してください。

関連概念
クライアント・チャネル定義テーブル
関連タスク
サーバーとクライアント間の接続の構成
関連資料
MQSERVER
MQCHLLIB
MQCHLTAB

環境変数を使用したクライアント・アプリケーションのキュー・マネージャーへの接続

クライアント・チャネル情報は、MQSERVER、MQCHLLIB、および MQCHLTAB 環境変数によって、クライアント環境で実行しているアプリケーションに提供できます。

これらの変数の詳細については、MQSERVER、MQCHLLIB、および MQCHLTAB を参照してください。

MQCNO 構造体を使用したクライアント・アプリケーションのキュー・マネージャーへの接続

MQCONNX 呼び出しの MQCNO 構造体を使用して提供されるチャネル定義構造体 (MQCD) 内で、チャネルの定義を指定することができます。

詳しくは、 MQCONNX 呼び出しでの MQCNO 構造体の使用 を参照してください。

クライアント・チャネル定義テーブルを使用したクライアント・アプリケーションのキュー・マネージャーへの接続

MQSC DEFINE CHANNEL コマンドを使用する場合、指定した詳細はクライアント・チャネル定義テーブル (CCDT) に入れられます。 MQCONN 呼び出しまたは MQCONNX 呼び出しの *QMgrName* パラメーターの内容によって、クライアントが接続されるキュー・マネージャーが判別されます。

このファイルは、アプリケーションが使用するチャネルを判別するためにクライアントがアクセスするファイルです。 適切なチャネル定義が複数ある場合、チャネルの選択には、クライアント・チャネル・ウェイト (CLNTWGHT) および接続アフィニティー (AFFINITY) のチャネル属性が関係します。

クライアント・チャネル定義テーブルの役割

クライアント・チャネル定義テーブル (CCDT) には、クライアント接続チャネルの定義が含まれます。 これは、クライアント・アプリケーションが複数の代替キュー・マネージャーに接続する必要がある場合に特に役立ちます。

キュー・マネージャーを定義すると、クライアント・チャネル定義テーブルが作成されます。

注 : 同じファイルを、複数の IBM WebSphere MQ クライアントが使用できます。 MQCHLLIB および MQCHLTAB IBM WebSphere MQ 環境変数を使用して、このファイルの異なるバージョンにアクセスします。 環境変数については、WebSphere MQ 環境変数の使用を参照してください。

CCDT のキュー・マネージャー・グループ

クライアント・チャネル定義テーブル (CCDT) で接続のセットをキュー・マネージャー・グループ として定義できます。 キュー・マネージャー・グループの一部であるキュー・マネージャーにアプリケーションを接続することができます。 これは、MQCONN または MQCONNX の呼び出し時に、 接頭部にアスタリスク付きのキュー・マネージャー名を使用することによって実行できます。

以下の理由から、複数のサーバー・マシンへの接続の定義を選択することがあります。

- 可用性を高めるために、実行中の一連のキュー・マネージャーのいずれかにクライアントを接続したい。
- 前回、正常に接続されたものと同じキュー・マネージャーにクライアントを再接続したいが、 接続に失敗した場合は別のキュー・マネージャーに接続したい。
- 接続に失敗した場合は、再度クライアント・プログラムで MQCONN を発行して、別のキュー・マネージャーにクライアント接続を再試行できるようにしたい。
- 接続に失敗した場合は、クライアント・コードを書かなくても、別のキュー・マネージャーにクライアントを自動的に再接続したい。
- スタンバイ・インスタンスを引き継ぐ場合は、クライアント・コードを書かなくても、 複数インスタンスのキュー・マネージャーの異なるインスタンスにクライアントを自動的に再接続したい。
- 一部のキュー・マネージャーには、他のキュー・マネージャーより多くのクライアントが接続されるように、多数のキュー・マネージャー全体のクライアント接続のバランスを取りたい。
- 接続数が多いことによって障害が発生する場合は、 多数のクライアントの再接続を複数のキュー・マネージャーに時間の経過に従って分散させたい。
- クライアント・アプリケーション・コードを変更せずにキュー・マネージャーを移動できるようにしたい。
- キュー・マネージャー名を認識する必要のないクライアント・アプリケーション・プログラムを作成したい。

別のキュー・マネージャーに接続することは、いつも適切であるというわけではありません。 例えば、WebSphere Application Server の拡張トランザクション・クライアントまたは Java クライアントは、予測可能なキュー・マネージャー・インスタンスに接続する必要がある場合があります。 クライアントの自動再接続機能は、WebSphere MQ classes for Java ではサポートされていません。

キュー・マネージャー・グループは、クライアント・チャネル定義テーブル (CCDT) で定義されている接続セットです。 このセットは、それぞれのチャネル定義内に同じ値の **QMNAME** 属性を持つメンバーによって定義されます。

364 ページの図 70 は、クライアント接続テーブルを図形で表したものです。 これには、3 つのキュー・マネージャー・グループが示され、 そのうちの 2 つは、**QMNAME**(QM1) および **QMNAME**(QMGrp1) という名前で CCDT に記述されるキュー・マネージャー・グループで、 1 つは **QMNAME**(' ') で記述されるブランク (デフォルト・グループ) です。

1. キュー・マネージャー・グループ QM1 には 3 つのクライアント接続チャネルがあり、キュー・マネージャー QM1 および QM2 に接続されています。 QM1 は、2 つの異なるサーバーにある複数インスタンス・キュー・マネージャーである場合があります。
2. デフォルトのキュー・マネージャー・グループには、6 つのクライアント接続チャネルがあり、すべてのキュー・マネージャーに接続されています。

3. QMGrp1 には、QM4 および QM5 という 2 つのキュー・マネージャーへのクライアント接続チャネルがあります。

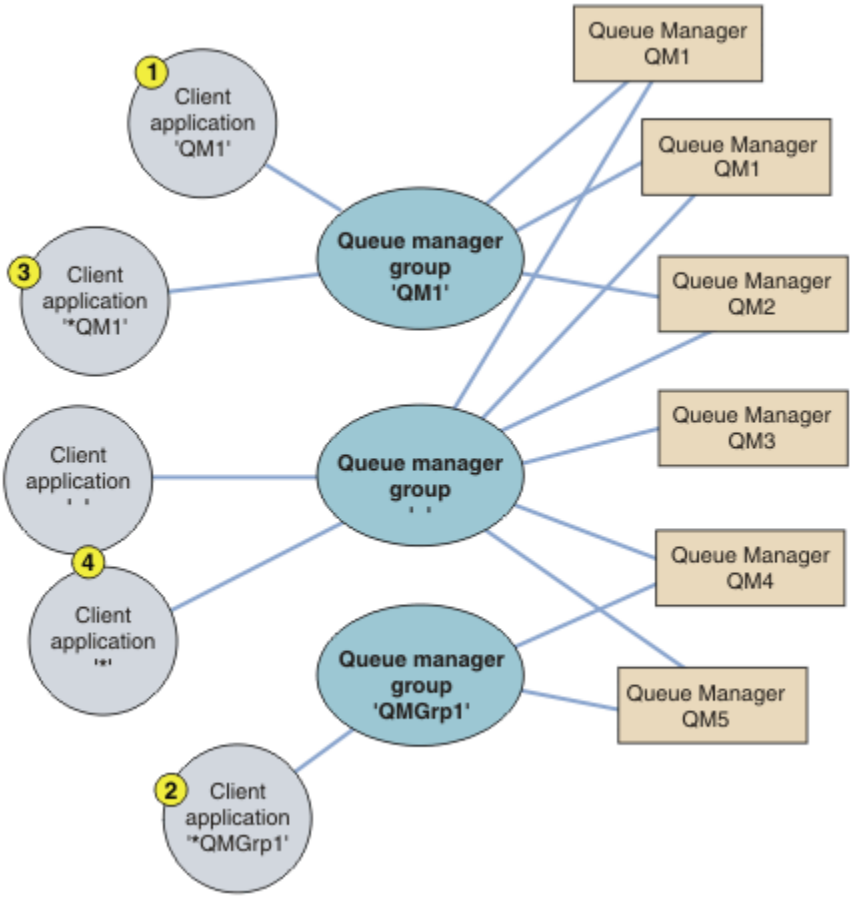


図 70. キュー・マネージャー・ グループ

このクライアント接続テーブルの 4 つの使用例について、364 ページの図 70 で番号を付けたクライアント・アプリケーションを用いて説明します。

1. 1 つ目の例では、クライアント・アプリケーションが、キュー・マネージャー名 QM1 を **QmgrName** パラメーターとして、MQCONN または MQCONNX MQI 呼び出しに渡します。 WebSphere MQ クライアント・コードによって、一致するキュー・マネージャー・グループ QM1 が選択されます。 グループには 3 つの接続チャネルがあり、WebSphere MQ MQI クライアントはこれらのチャネルをそれぞれ順に使用して、QM1 と呼ばれる実行中のキュー・マネージャーへの接続用の WebSphere MQ リスナーを検出するまで、QM1 への接続を試行します。

   接続の試行順序は、クライアント接続の AFFINITY 属性の値およびクライアント・チャネルの重み付けによって決まります。 これらの制約では、可能性のある 3 つの接続と経過時間の両方を考慮して、 接続の試行順序がランダムに選択され、接続の負荷が分散されます。

   QM1 の実行中のインスタンスへの接続が確立されれば、 クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

2. 2 つ目の例では、クライアント・アプリケーションが、**QmgrName** パラメーターとして先頭にアスタリスクが付いたキュー・マネージャー名 *QMGrp1 を、MQCONN または MQCONNX MQI 呼び出しに渡します。 WebSphere MQ クライアントによって、一致するキュー・マネージャー・グループ QMGrp1 が選択されます。 このグループには 2 つのクライアント接続チャネルがあり、WebSphere MQ MQI クライアントは各チャネルを順に使用して任意の キュー・マネージャーへの接続を試みます。 この例では、WebSphere MQ MQI クライアントは正常な接続を確立する必要があります。接続先のキュー・マネージャーの名前は関係ありません。

接続の試行順序についての規則は 1 つ目のケースと同じです。 唯一異なる点は、キュー・マネージャー名の先頭にアスタリスクが付いていることで、クライアントは、キュー・マネージャーの名前が関連していないことを示します。

QMGrp1 キュー・マネージャー・グループ内のチャネルを経由して接続された任意のキュー・マネージャーの実行中のインスタンスへの接続が確立されれば、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

3. 3 つ目の例は、**QmgrName** パラメーターの先頭にアスタリスクが付けられて *QM1 となっているため、基本的には 2 つ目の例と同じです。 この例は、クライアント・チャネル接続での接続先キュー・マネージャーを、1 つのチャネル定義の QMNAME 属性を単独で検査することでは決定できないことを示しています。 チャネル定義の **QMNAME** 属性が QM1 であっても、QM1 と呼ばれるキュー・マネージャーへの接続が必要であるとは限りません。 クライアント・アプリケーションの **QmgrName** パラメーターの先頭にアスタリスクが付いている場合、いずれのキュー・マネージャーも接続先にすることができます。

   このケースでは、QM1 または QM2 のいずれかの実行中のインスタンスへの接続が確立された場合に、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しが成功します。

4. 4 つ目の例は、デフォルト・グループの使用を示しています。 このケースでは、クライアント・アプリケーションが、**QmgrName** パラメーターとしてアスタリスク '*' またはブランク ' ' を、MQCONN または MQCONNX MQI 呼び出しに渡します。 クライアント・チャネル定義の規則により、ブランクの **QMNAME** 属性はデフォルトのキュー・マネージャー・グループを表し、ブランクまたはアスタリスクの付いた **QmgrName** パラメーターのいずれかが、ブランクの **QMNAME** 属性と一致します。

   この例では、デフォルトのキュー・マネージャー・グループに、すべてのキュー・マネージャーへのクライアント・チャネル接続があります。 デフォルトのキュー・マネージャー・グループを選択することにより、アプリケーションをグループ内のどのキュー・マネージャーにも接続できます。

   任意のキュー・マネージャーの実行中のインスタンスへの接続が確立された場合に、クライアント・アプリケーションが発行した MQCONN または MQCONNX 呼び出しは成功します。

   注 : アプリケーションはブランクの **QmgrName** パラメーターを使用して、デフォルトのキュー・マネージャー・グループまたはデフォルトのキュー・マネージャーのいずれかに接続しますが、デフォルト・グループはデフォルトのキュー・マネージャーとは異なります。 デフォルトのキュー・マネージャー・グループの概念は、クライアント・アプリケーションにのみ関係しますが、デフォルトのキュー・マネージャーの概念はサーバー・アプリケーションに関係します。

2 番目または 3 番目のキュー・マネージャーに接続するチャネルも含め、クライアント接続チャネルは 1 つのキュー・マネージャーのみで定義してください。 これらのチャネルを 2 つのキュー・マネージャーで定義してから、 2 つのクライアント・チャネル定義テーブルをマージしないようにしてください。 クライアントがアクセスできるのは、1 つのクライアント・チャネル定義テーブルのみです。

例

トピックの冒頭にある、キュー・マネージャー・グループを使用する理由を示すリストをもう一度見てください。 キュー・マネージャー・グループを使用することで、以下の機能がどのように提供されるのでしょうか。

キュー・マネージャー・セット内のいずれかのキュー・マネージャーに接続する。
セット内のすべてのキュー・マネージャーへの接続を使用してキュー・マネージャー・グループを定義し、先頭にアスタリスクの付いた **QmgrName** パラメーターを使用してグループに接続します。

同じキュー・マネージャーに再接続するが、最後に接続したキュー・マネージャーが使用不可の場合は、別のキュー・マネージャーに接続する。
前述のようにキュー・マネージャー・グループを定義しますが、各クライアント・チャネル定義で属性 **AFFINITY**(PREFERRED) を設定します。

接続に失敗した場合は、別のキュー・マネージャーへの接続を再試行する。
キュー・マネージャー・グループに接続して、接続が切れた場合、またはキュー・マネージャーで障害が発生した場合は、MQCONN または MQCONNX MQI 呼び出しを再発行します。

接続に失敗した場合は、別のキュー・マネージャーに自動的に再接続する。
MQCONNX **MQCNO** オプションの MQCNO_RECONNECT を使用してキュー・マネージャー・グループに接続します。

複数インスタンスのキュー・マネージャーの異なるインスタンスに自動的に再接続する。
前述の例と同じ操作を行います。 このケースでは、特定の複数インスタンス・キュー・マネージャーのインスタンスに接続するようにキュー・マネージャー・グループを制限する場合、 複数インスタンス・キュー・マネージャーのインスタンスのみへの接続を使用してグループを定義します。

クライアント・アプリケーションに、先頭にアスタリスクが付いていない **QmgrName** パラメーターを使用して、MQCONN または MQCONNX MQI 呼び出しを発行するように要求することもできます。 これが、クライアント・アプリケーションが名前の付いたキュー・マネージャーに接続できる唯一の方法です。 最後に、**MQCNO** オプションを MQCNO_RECONNECT_Q_MGR に設定できます。 このオプションでは、以前接続したものと同じキュー・マネージャーへの再接続を受け入れます。 この値を使用して、通常キュー・マネージャーの同じインスタンスへの再接続を制限することもできます。

キュー・マネージャー間でクライアント接続の バランスを取る 。一部のキュー・マネージャーには、他のキュー・マネージャーよりも多くのクライアントが接続されます。
キュー・マネージャー・グループを定義し、接続を不均等に分散するように各クライアント・チャネル定義で **CLNTWGHT** 属性を設定します。

接続に失敗した場合またはキュー・マネージャーに障害が発生した場合、クライアント再接続の負荷を不均等に、時間の経過に従って分散する。
前述の例と同じ操作を行います。 WebSphere MQ MQI クライアントはキュー・マネージャー全体で再接続をランダムに行い、時間の経過に従って再接続を分散します。

クライアント・コードを変更しないでキュー・マネージャーを移動する。
CCDT で、クライアント・アプリケーションをキュー・マネージャーの位置から分離させます。

各クライアントにクライアント接続テーブルを分散するか、 参照先のクライアントごとの共用ファイル・システムに CCDT を配置するかを選択できます。 あるいは、MQCONNX MQI 呼び出しでサポートされる CCDT のプログラム版を使用して、 CCDT をクライアント・アプリケーションに渡すためのサービスを呼び出します。

キュー・マネージャー名を認識しないクライアント・アプリケーションを作成する。
キュー・マネージャー・グループ名を使用して、組織内のクライアント・アプリケーションに関連し、かつ、キュー・マネージャーの名前ではなくソリューション・アーキテクチャーを反映するような、キュー・マネージャー・グループ名の命名規則を設定します。

キュー共用グループへの接続
キュー共用グループの一部であるキュー・マネージャーに、ご使用のアプリケーションを接続することができます。 これは、MQCONN 呼び出し時または MQCONNX 呼び出し時に、キュー・マネージャー名ではなくキュー共用グループ名を使用することによって実行できます。

キュー共用グループには最大 4 文字の名前が付けられています。 この名前はネットワーク内で固有であり、かつ、キュー・マネージャー名とは異なるものである必要があります。

クライアント・チャネル定義では、グループ内の使用可能なキュー・マネージャーに接続するときは、キュー共用グループの汎用インターフェースを使用する必要があります。 詳細については、キュー共用グループへのクライアントの接続を参照してください。 リスナーが接続するキュー・マネージャーが、キュー共用グループのメンバーであることを確認する検査が行われます。

チャネルの加重とアフィニティーの例
この例では、ゼロ以外の ClientChannelWeights が使用された場合に、どのようにクライアント接続チャネルが選択されるかを示します。

ClientChannelWeight チャネル属性および ConnectionAffinity チャネル属性は、1 つの接続に関して使用できる適切なチャネルが複数存在する場合に、クライアント接続チャネルを選択する方法を制御します。 これらのチャネルは、高可用性またはワークロード・バランシング、あるいはその両方を提供するために、異なるキュー・マネージャーに接続するように構成されています。 複数のキュー・マネージャーのいずれかに接続する可能性がある MQCONN 呼び出しでは、MQCONN 呼び出しの例: 例 1 で説明されているように、キュー・マネージャー名の前にアスタリスクを付ける必要があります。 キュー・マネージャー名にアスタリスク (*) が含まれます。

接続するのに適切なチャネルの候補は、QMNAME 属性が MQCONN 呼び出しで指定されるキュー・マネージャー名に一致するチャネルです。 接続に適用可能なすべてのチャネルで、ClientChannelWeight がゼロ (デフォルト) の場合は、以下の例のようにアルファベット順で選択されます。 MQCONN 呼び出しの例: 1 の例。 キュー・マネージャー名にアスタリスク (*) が含まれます。

ゼロでない ClientChannelWeights が使用される場合については、以下の例で説明します。 このフィーチャーでは疑似ランダムなチャネル選択が行われるため、これらの例はアクションの厳密な順番ではなく、実行される可能性のある順番を示していることに注意してください。

例 *1. ConnectionAffinity* が *PREFERRED* に設定されている場合のチャネルの選択
この例では、ConnectionAffinity が PREFERRED に設定されている場合に、WebSphere MQ MQI クライアントが CCDT からチャネルを選択する方法を示します。

この例では、複数のクライアント・マシンが、キュー・マネージャーにより提供されたクライアント・チャネル定義テーブル (CCDT) を使用しています。 CCDT には、次の属性を持つクライアント接続チャネルが含まれます (DEFINE CHANNEL コマンドの構文を使用して示します)。

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(PREFERRED)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(PREFERRED)
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(PREFERRED)
```

アプリケーションは MQCONN(*CORE) を発行します。

QMNAME 属性が一致しないため、チャネル A はこの接続の候補にはなりません。 チャネル B、C、および D は候補として識別され、それぞれの加重に基づいた優先順序で並んでいます。 この例では、順序は C、B、D となります。 クライアントは、core2.ops.company.example でキュー・マネージャーへの接続を試みます。 このアドレスではキュー・マネージャーの名前は検査されません。これは、MQCONN 呼び出しで、キュー・マネージャー名にアスタリスクが含まれていたためです。

AFFINITY(PREFERRED) の場合、この特定のクライアント・マシンは接続のたびに、常に初期の優先順序でチャネルを並べることにご注意ください。 これは、接続が別のプロセスから、または別の時点で実行された場合であっても同じです。

この例では、core.2.ops.company.example のキュー・マネージャーには到達できません。 チャネル B が優先順位で次の順序にあるため、クライアントは core1.ops.company.example への接続を試行します。 さらに、チャネル C がデモートされ、優先順位の最下位になります。

2 番目の MQCONN(*CORE) 呼び出しが、同じアプリケーションから発行されます。 チャネル C は以前の接続によって降格されたため、現在最も優先されるチャネルは B です。 この接続は core1.ops.company.example に対して行われます。

同じクライアント・チャネル定義テーブルを共有する 2 番目のマシンでは、異なる初期の優先順序でチャネルを並べる場合があります。 例えば、D、B、C などです。 すべてのチャネルが動作する状態にある通常の環境では、このマシン上のアプリケーションは core3.ops.company.example に接続され、最初のマシン上のアプリケーションは core2.ops.company.example に接続されます。 これにより、複数のキュー・マネージャーを対象とした多数のクライアント間のワークロード・バランシングが可能になる一方で、個々のクライアントが同じキュー・マネージャー (使用可能な場合) に接続することができます。

例 *2. ConnectionAffinity* が *NONE* に設定されている場合のチャネルの選択
この例では、ConnectionAffinity が NONE に設定されている場合に、WebSphere MQ MQI クライアントが CCDT からチャネルを選択する方法を示します。

この例では、複数のクライアントが、キュー・マネージャーにより提供されたクライアント・チャネル定義テーブル (CCDT) を使用しています。 CCDT には、次の属性を持つクライアント接続チャネルが含まれます (DEFINE CHANNEL コマンドの構文を使用して示します)。

```
CHANNEL(A) QMNAME(DEV) CONNAME(devqm.it.company.example)
CHANNEL(B) QMNAME(CORE) CONNAME(core1.ops.company.example) CLNTWGHT(5) +
AFFINITY(NONE)
CHANNEL(C) QMNAME(CORE) CONNAME(core2.ops.company.example) CLNTWGHT(3) +
AFFINITY(NONE)
```

```
CHANNEL(D) QMNAME(CORE) CONNAME(core3.ops.company.example) CLNTWGHT(2) +
AFFINITY(NONE)
```

アプリケーションは MQCONN(*CORE) を発行します。 前の例と同様、QMNAME が一致しないのでチャネル A は検討の対象にはなりません。 チャネル B、C、D はそれぞれの加重に基づいて選択されます。選択される確率はそれぞれ、50%、30%、20% です。 この例では、チャネル B が選択されると考えられます。作成される優先順位は永続的ではありません。

2 番目の MQCONN(*CORE) 呼び出しが実行されます。 この場合も、3 つの候補のチャネルのうちの 1 つが、前と同じ確率で選択されます。 この例では、チャネル C が選択されます。 しかし、core2.ops.company.example が応答しないため、残りの候補のチャネルの中から別のチャネルが選択されます。 チャネル B が選択され、アプリケーションが core1.ops.company.example に接続されます。

AFFINITY(NONE) が指定される場合、各 MQCONN 呼び出しは互いに独立しています。 したがって、このサンプル・アプリケーションが 3 番目の MQCONN(*CORE) 呼び出しを行う場合、切断されたチャネル C による接続をもう一度試行してから、B または D のいずれかを選択することになります。

MQCONN 呼び出しの例

MQCONN を使用して、特定のキュー・マネージャー、または複数のキュー・マネージャーのうちの 1 つに接続する例です。

次のそれぞれの例では、ネットワークは同じであり、同じ WebSphere MQ MQI クライアントから 2 つのサーバーに定義された接続があります。 (以下の例では、MQCONN 呼び出しの代わりに MQCONNX 呼び出しを使用できます。)

サーバー・マシン上で 2 つのキュー・マネージャーが稼働中であり、一方は SALE、他方は SALE_BACKUP という名前です。

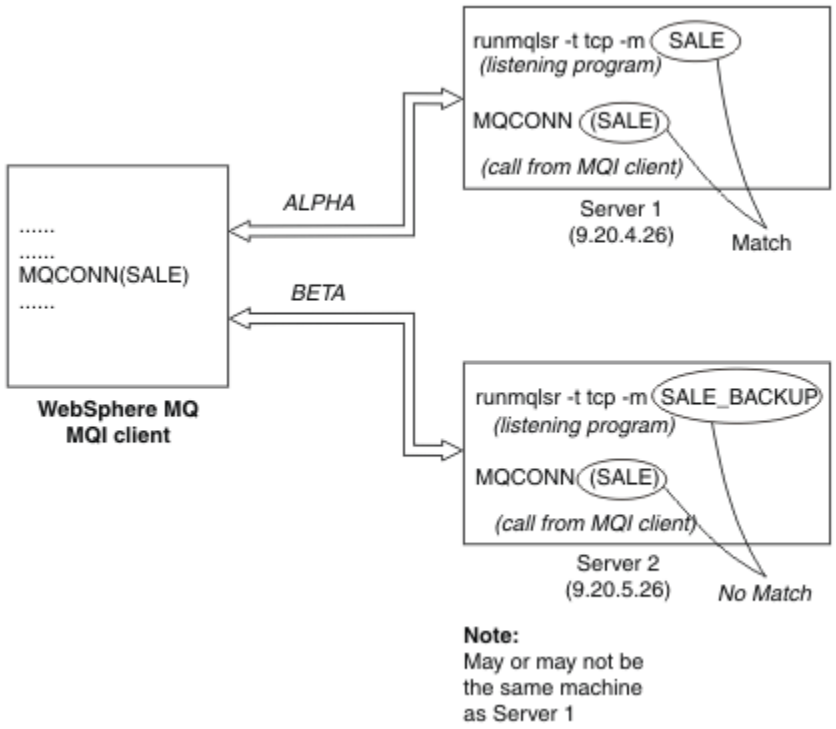


図 *71. MQCONN* の例

これらの例のチャネルについての定義は、次のとおりです。

SALE の定義:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('WebSphere MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('WebSphere MQ MQI client connection to server 2') +
QMNAME(SALE)
```

SALE_BACKUP の定義:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')
```

クライアント・チャネル定義は次のように要約できます。

名前	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	SALE
BETA	CLNTCONN	TCP	9.20.5.26	SALE

MQCONN の例で示される内容
この例では、バックアップ・システムとして複数のキュー・マネージャーを使用する例を示します。

サーバー 1 との通信リンクが一時的に切断されたとします。 バックアップ・システムとして複数のキュー・マネージャーを使用する例を示します。

それぞれの例で異なる MQCONN 呼び出しが使用され、次の規則を適用して、特定の例で何が起こっているかを説明します。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、MQCONN 呼び出しで指定された名前に対応するキュー・マネージャー名 (QMNAME フィールド) が検索されます。
2. 一致するものが見つかると、そのチャネル定義が使用されます。
3. 接続名 (CONNAME) で識別されたマシンに対するチャネルの開始が試行されます。 この試行が成功すると、アプリケーションは処理を続行します。 アプリケーションには、次のものが必要です。
   - サーバー上で稼働するリスナー
   - クライアントが接続しようとするキュー・マネージャーと同じキュー・マネージャーに接続されるリスナー (指定されている場合)
4. チャネルの開始の試行に失敗し、クライアント・チャネル定義テーブルに複数のエントリーがある (この例では 2 つのエントリーがある) 場合は、一致するものを求めてファイルがさらに検索されます。 一致するものが見つかると、ステップ 1 から処理が続行されます。
5. 一致するエントリーが見つからない場合、またはクライアント・チャネル定義テーブルのエントリーがそれ以外にないためチャネルが開始できなかった場合、アプリケーションは接続できません。 該当する理由コードおよび完了コードが MQCONN 呼び出しで戻されます。 アプリケーションでは、戻された理由コードおよび完了コードに基づいた処置を行うことができます。

例 *1.* キュー・マネージャー名にアスタリスク *(*)* が含まれる
この例では、アプリケーションがどのキュー・マネージャーに接続するかは関係ありません。 アプリケーションは、アスタリスクを含むキュー・マネージャー名に対して MQCONN 呼び出しを発行します。 適切なチャネルが選択されます。

アプリケーションは、次の呼び出しを発行します。

```
MQCONN (*SALE)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) で、アプリケーションの MQCONN 呼び出しに対応するキュー・マネージャー名 SALE が検索されます。
2. ALPHA および BETA のチャネル定義が見つかります。
3. いずれかのチャネルの CLNTWGHT 値が 0 の場合、そのチャネルが選択されます。 両方のチャネルの CLNTWGHT 値が 0 の場合、アルファベット順で先にあるチャネル ALPHA が選択されます。 いずれのチャネルの CLNTWGHT 値もゼロでない場合、その加重に基づいてどちらか一方のチャネルがランダムに選択されます。
4. チャネルの開始が試行されます。
5. チャネル BETA が選択された場合、その開始の試行は成功します。
6. チャネル ALPHA が選択された場合、通信リンクが切断されているため開始の試行は成功しません。 この場合は、以下のステップが適用されます。
   a. キュー・マネージャー名 SALE のその他の唯一のチャネルは BETA です。
   b. このチャネルの開始が試行されます。これは成功します。
7. リスナーが稼働しているかどうかの検査の結果、稼働しているリスナーが 1 つあることがわかります。これは SALE キュー・マネージャーに接続されていませんが、MQI 呼び出しのパラメーターにアスタリスク (*) が含まれているため、接続についての検査は行われません。 アプリケーションは SALE_BACKUP キュー・マネージャーに接続されて処理を続行します。

例 *2.* キュー・マネージャー名が指定されています
この例では、アプリケーションは特定のキュー・マネージャーに接続する必要があります。 アプリケーションは、そのキュー・マネージャー名に対して MQCONN 呼び出しを発行します。 適切なチャネルが選択されます。

次の MQI 呼び出しに示されているように、アプリケーションは SALE という名前の特定のキュー・マネージャーへの接続を要求します。

```
MQCONN (SALE)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、アプリケーションの MQCONN 呼び出しに対応するキュー・マネージャー名 SALE が検索されます。
2. 最初に見つかった一致するチャネル定義は ALPHA です。
3. チャネルの開始が試行されますが、通信リンクが切断されているため成功しません。
4. クライアント・チャネル定義テーブルが再び走査され、キュー・マネージャー名 SALE を検索すると、チャネル名 BETA が見つかります。
5. チャネルの試行が開始されます。これは成功します。
6. リスナーが稼働しているかどうかの検査の結果、稼働しているリスナーが 1 つあることがわかりますが、それは SALE キュー・マネージャーに接続されていません。
7. クライアント・チャネル定義テーブルには、これ以上エントリーがありません。 アプリケーションは続行できず、戻りコード MQRC_Q_MGR_NOT_AVAILABLE を受け取ります。

例 *3.* キュー・マネージャー名はブランクまたはアスタリスク *(*)* です
この例では、アプリケーションがどのキュー・マネージャーに接続するかは関係ありません。 アプリケーションは、キュー・マネージャー名にブランクのまたはアスタリスクを指定して、MQCONN を発行します。 適切なチャネルが選択されます。

これは、369 ページの『例 1. キュー・マネージャー名にアスタリスク (*) が含まれる』の場合と同じ方法で処理されます。

注 : このアプリケーションが WebSphere MQ MQI クライアント以外の環境で実行されていた場合、名前にブランクが指定されると、アプリケーションはデフォルトのキュー・マネージャーとの接続を試行します。これは、アプリケーションがクライアント環境から実行されている場合には当てはまりません。アクセスされるキュー・マネージャーは、チャネルの接続先のリスナーと関連付けられているものです。

アプリケーションは、次の呼び出しを発行します。

```
MQCONN ("")
```

または

```
MQCONN (*)
```

前述の規則に従うと、この例での処理は次のようになります。

1. クライアント・チャネル定義テーブル (CCDT) がチャネル名のアルファベット順に走査され、アプリケーションの MQCONN 呼び出しに対応するブランクのキュー・マネージャー名が検索されます。
2. チャネル名 ALPHA のエントリーには、SALE という定義内にキュー・マネージャー名があります。 これは、MQCONN 呼び出しのパラメーターと一致しません。ここでは、キュー・マネージャー名がブランクである必要があります。
3. 次のエントリーは、チャネル名 BETA のエントリーです。
4. 定義内の queue manager name は SALE です。 これも、キュー・マネージャー名がブランクである必要がある MQCONN 呼び出しのパラメーターとは一致しません。
5. クライアント・チャネル定義テーブルには、これ以上エントリーがありません。 アプリケーションは続行できず、戻りコード MQRC_Q_MGR_NOT_AVAILABLE を受け取ります。

クライアント環境でのトリガー操作

WebSphere MQ MQI クライアント上で実行中の WebSphere MQ アプリケーションが送信したメッセージは、他のすべてのメッセージとまったく同じようにトリガー操作に使用でき、これを使用して、サーバーとクライアントの両方でプログラムを起動 (トリガー) することができます。

トリガー操作については、チャネルのトリガー操作を参照してください。

トリガー・モニターと開始されるアプリケーションは、同じシステム上になければなりません。

クライアント環境におけるトリガーされたキューのデフォルトの特性は、サーバー環境のトリガーされたキューと同じです。 具体的には、z/OS キュー・マネージャーに対してローカルであるトリガーされたキューにメッセージを書き込むクライアント・アプリケーションに MQPMO 同期点制御オプションが指定されていない場合、メッセージは作業単位内に書き込まれます。 この場合は、トリガー条件が満たされても、トリガー・メッセージは同じ作業単位内の開始キューに書き込まれるため、トリガー・モニターはこの作業単位が終了するまでトリガー・メッセージを取り出せません。 この作業単位が終わるまで、トリガー操作のプロセスは起動されないことになります。

プロセス定義

トリガー設定がオンになっているキューに関連付けられているため、サーバー上でプロセス定義を定義する必要があります。

プロセス・オブジェクトは起動する必要があるものを定義します。 クライアントおよびサーバーが同じプラットフォーム上で実行されていない場合、トリガー・モニターが開始したプロセスは *ApplType* を定義しなければなりません。定義していないと、サーバーはデフォルトの定義 (つまり、通常サーバー・マシンに関連しているアプリケーションのタイプ) を選択し、障害を引き起こします。

例えば、トリガー・モニターが Windows クライアント上で稼働していて、他のオペレーティング・システム上のサーバーに要求を送信する場合は、MQAT_WINDOWS_NT が定義されている必要があります。定義されていないと、他のオペレーティング・システムでそのデフォルト定義が使用され、プロセスが失敗します。

トリガー・モニター

z/OS WebSphere MQ 以外の製品によって提供されるトリガー・モニターは、UNIX、Linux 、および Windows システムのクライアント環境で実行されます。

トリガー・モニターを実行するには、次のいずれかのコマンドを発行します。

- Windows Linux UNIX Windows、UNIX、および Linux プラットフォームの場合:

```
runmqtmc [-m QMgrName] [-q InitQ]
```

デフォルトの開始キューは、デフォルトのキュー・マネージャーにある SYSTEM.DEFAULT.INITIATION.QUEUE です。 開始キューは、トリガー・モニターがトリガー・メッセージを検索する場所です。 次に、プログラムを呼び出して、適切なトリガー・メッセージを探します。 このトリガー・モニターは、デフォルトのアプリケーション・タイプをサポートします。また、クライアント・ライブラリーにリンクするという点を除くと、このトリガー・モニターは runmqtrm と同じです。

トリガー・モニターが作成するコマンド・ストリングは次のとおりです。

1. 関連プロセス定義の *ApplicId*。 *ApplicId* は実行するプログラムの名前です。この名前は、コマンド行に入力されたとおりに表示されます。
2. 開始キューから取得され、引用符で囲まれた MQTMC2 構造体。 引用符で囲まれたこのストリングと正確に同じストリングを持つコマンド・ストリングが開始されます。したがって、システム・コマンドには 1 つのパラメーターとして受け付けられます。
3. 関連プロセス定義の *EnvrData*。

トリガー・モニターは、開始したアプリケーションが完了するまでは、開始キューに別のメッセージがあるかどうかを検索しません。 アプリケーションに多くの処理がある場合は、到着するトリガー・メッセージの数にトリガー・モニターが追いつかないことがあります。 この状態に対応するには、次の 2 つの方法があります。

1. 実行するトリガー・モニターを増やす

   実行するトリガー・モニターの数を増やす場合、一度に実行できるアプリケーションの最大数を制御できます。
2. 開始済みのアプリケーションのバックグラウンドで実行する

   アプリケーションをバックグラウンドで実行する場合、WebSphere MQ では実行できるアプリケーションの数を制限しません。

UNIX and Linux システムで、開始されたアプリケーションをバックグラウンドで実行するには、プロセス定義の *EnvrData* の末尾に & (アンパーサンド) を付ける必要があります。

CICS アプリケーション (z/OS 以外)

z/OS 以外の CICS アプリケーション・プログラムで MQCONN 呼び出し、または MQCONNX 呼び出しを発行する場合は、そのアプリケーション・プログラムが CEDA に RESIDENT として定義されている必要があります。 CICS サーバー・アプリケーションをクライアントとして再リンクすると、同期点サポートが失われるおそれがあります。

z/OS 以外の CICS アプリケーション・プログラムで MQCONN 呼び出し、または MQCONNX 呼び出しを発行する場合は、そのアプリケーション・プログラムが CEDA に RESIDENT として定義されている必要があります。 常駐コードをできるだけ小さくするために、別のプログラムにリンクして MQCONN 呼び出し、または MQCONNX 呼び出しを発行することができます。

MQSERVER 環境変数を使用してクライアント接続を定義する場合は、その変数を CICSENV.CMD ファイルの中で指定する必要があります。

WebSphere MQ アプリケーションは、WebSphere MQ サーバー環境で、または WebSphere MQ クライアント上で、コードを変更せずに実行できます。 ただし、WebSphere MQ サーバー環境では、CICS は同期点調整プログラムとして機能することができ、ユーザーは MQCMIT および MQBACK の代わりに EXEC CICS SYNCPOINT および EXEC CICS SYNCPOINT ROLLBACK を使用します。 CICS アプリケーションが単純にクライアントとして再リンクされると、同期点サポートが失われます。 WebSphere MQ MQI クライアント上で実行しているアプリケーションに対しては、MQCMIT および MQBACK を使用する必要があります。

CICS および Tuxedo アプリケーションの準備と実行

CICS および Tuxedo アプリケーションをクライアント・アプリケーションとして実行するには、サーバー・アプリケーションで使用しているのとは異なるライブラリーを使用します。 アプリケーションの実行に使用されるユーザー ID も異なります。

WebSphere MQ MQI クライアント・アプリケーションとして実行するために CICS および Tuxedo アプリケーションを準備するには、 拡張トランザクション・クライアントの構成の手順に従ってください。

ただし、特に CICS および Tuxedo アプリケーションの準備に関する情報 (WebSphere MQ で提供されるサンプル・プログラムを含む) は、 WebSphere MQ サーバー・システムで実行するアプリケーションを準備することを前提としています。 その結果、その情報は、サーバー・システム上で使用するための WebSphere MQ ライブラリーのみを参照しています。 クライアント・アプリケーションを準備するときには、次の手順を実行する必要があります。

- ご使用のアプリケーションが使用する言語バインディングに適したクライアント・システム・ライブラリーを使用する。 例えば、AIX、HP-UX、または Solaris 上で C で作成されたアプリケーションの場合、ライブラリー libmqm の代わりに libmqic を使用します。 Windows システムでは、ライブラリー mqm.lib の代わりに mqic.lib を使用します。
- 373 ページの表 48 (AIX、HP-UX、および Solaris の場合) および 373 ページの表 49 (Windows システムの場合) に表示されているサーバー・システム・ライブラリーではなく、それに相当するクライアント・システム・ライブラリーを使用する。 サーバー・システム・ライブラリーがこれらの表にリストされていない場合は、クライアント・システムで同じライブラリーを使用してください。

表 *48. AIX*、*HP-UX*、および *Solaris* でのクライアント・システム・ライブラリー

WebSphere MQ サーバー・システムのライブラリー	**WebSphere MQ クライアント・システム上で使用するための同等なライブラリー**
libmqmxa	libmqcxa

表 *49. Windows* システムでのクライアント・システム・ライブラリー

WebSphere MQ サーバー・システムのライブラリー	**WebSphere MQ クライアント・システム上で使用するための同等なライブラリー**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

クライアント・アプリケーションで使用されるユーザー ID

CICS 下で WebSphere MQ サーバー・アプリケーションを実行しているとき、通常は CICS ユーザーからトランザクションのユーザー ID に切り替わります。 しかし、CICS 下で WebSphere MQ MQI クライアント・アプリケーションを実行しているときには、CICS 特権を持つ権限が保持されます。

CICS と Tuxedo のサンプル・プログラム

AIX、 HP-UX、Solaris、および Windows システムで使用する CICS および Tuxedo のサンプル・プログラム。

374 ページの表 50 は、 AIX、 HP-UX、および Solaris クライアント・システムで使用するために提供されている CICS および Tuxedo のサンプル・プログラムをリストしています。 374 ページの表 51 では、Windows クライアント・システム用の同様の情報をリストしています。 これらの表では、プログラムの準備と実行に使用されるファイルもリストしています。 サンプル・プログラムの説明については、117 ページの『CICS トランザクション・サンプル』および 155 ページの『TUXEDO サンプル』を参照してください。

表 *50. AIX*、*HP-UX*、および *Solaris* クライアント・システム用のサンプル・プログラム		
説明	**ソース**	**実行可能モジュール**
CICS プログラム	amqscic0.ccs	amqscicc
CICS プログラム用のヘッダー・ファイル	amqscih0.h	-
メッセージを書き込むための Tuxedo クライアント・プログラム	amqstxpx.c	-
メッセージを取得するための Tuxedo クライアント・プログラム	amqstxgx.c	-
2 つのクライアント・プログラム用の Tuxedo サーバー・プログラム	amqstxsx.c	-
Tuxedo プログラム用の UBBCONFIG ファイル	ubbstxcx.cfg	-
Tuxedo プログラム用のフィールド・テーブル・ファイル	amqstxvx.flds	-
Tuxedo プログラム用のビュー記述ファイル	amqstxvx.v	-

表 *51. Windows* クライアント・システム用のサンプル・プログラム		
説明	**ソース**	**実行可能モジュール**
CICS トランザクション	amqscic0.ccs	amqscicc
CICS トランザクション用のヘッダー・ファイル	amqscih0.h	-
メッセージを書き込むための Tuxedo クライアント・プログラム	amqstxpx.c	-
メッセージを取得するための Tuxedo クライアント・プログラム	amqstxgx.c	-
2 つのクライアント・プログラム用の Tuxedo サーバー・プログラム	amqstxsx.c	-
Tuxedo プログラム用の UBBCONFIG ファイル	ubbstxcx.cfg	-
Tuxedo プログラム用のフィールド・テーブル・ファイル	amqstxvx.fld	-
Tuxedo プログラム用のビュー記述ファイル	amqstxvx.v	-
Tuxedo プログラム用の MAKE ファイル	amqstxmc.mak	-
Tuxedo プログラム用の ENVFILE ファイル	amqstxen.env	-

エラー・メッセージ AMQ5203 (CICS および Tuxedo アプリケーション用に変更)

拡張トランザクション・クライアントを使用する CICS アプリケーションまたは Tuxedo アプリケーションを実行すると、標準診断メッセージが表示されることがあります。 これらのメッセージの 1 つが、拡張トランザクション・クライアントで使用するために変更されています。

WebSphere MQ エラー・ログ・ファイルに示される可能性のあるメッセージは、診断メッセージ: AMQ4000-9999 に記載されています。 メッセージ AMQ5203 は、拡張トランザクション・クライアントでの使用に合わせて変更されています。 変更されたメッセージの本文は、次のとおりです。

AMQ5203: XA インターフェースを呼び出しているときにエラーが発生しました。

説明

エラー番号は &2 です。ここで、1 の値は &1 に指定されたフラグ値が無効であったことを示します。2 は、同一のプロセス内で、スレッド化されたライブラリーとスレッド化されていないライブラリーの使用が試みられたことを示します。3 は、指定されたキュー・マネージャー名 '&3' にエラーがあったことを示します。4 は、&1 のリソース・マネージャー ID が無効であったことを示します。5 は、別のキュー・マネージャーが既に接続している場合に、'&3' という名前の 2 番目のキュー・マネージャーの使用が試行されたことを示します。6 は、アプリケーションがキュー・マネージャーに接続していない場合に、トランザクション・マネージャーが呼び出されたことを示します。7 は、別の呼び出しが進行中に、XA 呼び出しが行われたことを示します。8 は、xa_open 呼び出し内の xa_info ストリング '&4' に、パラメーター名 '&5' の無効なパラメーター値が含まれていたことを示します。9 は、xa_open 呼び出し内の xa_info ストリング '&4' で、パラメーター名が '&5' である必須パラメーターが欠落していることを示します。

ユーザーの応答

エラーを訂正して操作をやり直してください。

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

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

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

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

- コンポーネントが MQI に C 言語バインディングを使用する場合は、461 ページの『Windows での C プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー mqic.lib ではなく、ライブラリー mqicxa.lib でリンクします。
- コンポーネントが WebSphere MQ C++ クラスを使用する場合は、658 ページの『Windows における C++ プログラムの作成』 の指示に従ってください。ただし、コンポーネントは、ライブラリー imqc23vn.lib ではなく、ライブラリー imqx23vn.lib でリンクします。
- コンポーネントが MQI に Visual Basic 言語バインディングを使用する場合は、466 ページの『Windows での Visual Basic プログラムの作成』 の指示に従います。ただし、Visual Basic プロジェクトを定義する際、「**条件付きコンパイル引数**」 フィールドに `MqType=3` と入力します。
- コンポーネントが WebSphere MQ Automation Classes for ActiveX (MQAX) を使用する場合は、値 `mqic32xa.dll` を使用して環境変数 GMQ_MQ_LIB を定義します。

  この環境変数は、アプリケーション内から定義するか、有効範囲がシステム全体になるように定義することができます。 ただし、この環境変数をシステム全体として定義すると、アプリケーション内からその環境変数を定義しない既存の MQAX アプリケーションが、誤った動作をする可能性があります。

WebSphere MQ JMS アプリケーションの準備と実行

WebSphere MQ JMS アプリケーションは、WebSphere Application Server をトランザクション・マネージャーとして、クライアント・モードで実行できます。 特定の警告メッセージが表示される場合があります。

WebSphere Application Server をトランザクション・マネージャーとして使用する場合、WebSphere MQ JMS アプリケーションをクライアント・モードで準備および実行するには、721 ページの『WebSphere MQ classes for JMS の使用』 の指示に従ってください。

WebSphere MQ JMS クライアント・アプリケーションを実行する場合、次の警告メッセージが表示される場合があります。

MQJE080

ライセンス・ユニットが足りません - setmqcap を実行してください (Insufficient license units - run setmqcap)

MQJE081
ライセンス・ユニット情報が入っているファイルの形式が誤っています - setmqcap を実行してください (File containing the license unit information is in the wrong format - run setmqcap)

MQJE082
ライセンス・ユニット情報が入っているファイルが見付かりませんでした - setmqcap を実行してください (File containing the license unit information could not be found - run setmqcap)

ユーザー出口、API 出口、および WebSphere MQ インストール可能サービス

ユーザー出口、API 出口、またはインストール可能サービスを使用して、キュー・マネージャーの機能を拡張できます。 このトピックには、これらのプログラムの使用および開発に関する情報へのリンクが含まれています。

ユーザー出口、API 出口、およびインストール可能サービスを使用してキューマネジャー機能を拡張する方法の概要については、「キュー・マネージャー機能」を参照してください。

出口およびインストール可能サービスの作成およびコンパイルについての詳細は、376 ページの『出口とインストール可能サービスの作成とコンパイル』を参照してください。

関連概念
MQI チャネル用のチャネル出口プログラム
関連資料
API 出口参照
インストール可能サービス・インターフェースの参照情報

出口とインストール可能サービスの作成とコンパイル

UNIX、 Linux 、および Windows 上の IBM WebSphere MQ ライブラリーにリンクすることなく、出口を作成してコンパイルすることができます。

このタスクについて

Windows Linux UNIX このトピックは、 Windows、 UNIX and Linux システムにのみ適用されます。 他のプラットフォームの場合の出口とインストール可能サービスの作成について詳しくは、該当するプラットフォーム固有のトピックを参照してください。

IBM WebSphere MQ がデフォルト以外の場所にインストールされている場合は、IBM WebSphere MQ ライブラリーにリンクせずに出口を作成してコンパイルする必要があります。

以下の IBM WebSphere MQ ライブラリーのいずれにもリンクせずに、 Windows、 UNIX and Linux システム上で出口を作成し、コンパイルすることができます。

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

これらのライブラリーにリンクしている既存の出口は、UNIX and Linux システムで IBM WebSphere MQ がデフォルトの場所にインストールされている限り、引き続き正しく動作します。

手順

1. cmqec.h ヘッダー・ファイルを組み込みます。

このヘッダー・ファイルを組み込むと、cmqc.h、cmqxc.h、cmqzc.h の各ヘッダー・ファイルも自動的に組み込まれます。

2. MQI 呼び出しと DCI 呼び出しが MQIEP 構造体を経由するようにして、出口を作成します。 MQIEP 構造体の詳細については、MQIEP 構造体を参照してください。

   - インストール可能サービス
     - **Hconfig** パラメーターを使用して、MQZEP 呼び出しを参照します。
     - **Hconfig** パラメーターを使用する前に、**Hconfig** の最初の 4 バイトが MQIEP 構造体の **StrucId** と一致することを確認してください。
     - インストール可能サービス・コンポーネントを作成するための詳細については、MQIEP を参照してください。
   - API 出口
     - **Hconfig** パラメーターを使用して、MQXEP 呼び出しを参照します。
     - **Hconfig** パラメーターを使用する前に、**Hconfig** の最初の 4 バイトが MQIEP 構造体の **StrucId** と一致することを確認してください。
     - API 出口を作成するための詳細については、391 ページの『API 出口の作成』を参照してください。
   - チャネル出口
     - MQCXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
     - **pEntryPoints** を使用する前に、MQCXP のバージョン番号がバージョン 8 以上であることを確認してください。
     - チャネル出口を作成するための詳細については、401 ページの『チャネル出口プログラムの作成』を参照してください。
   - データ変換出口
     - MQDXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
     - **pEntryPoints** を使用する前に、MQDXP のバージョン番号がバージョン 2 以上であることを確認してください。
     - **crtmqcvx** コマンドと amqsvfc0.c ソース・ファイルを使用して、**pEntryPoints** パラメーターを使用するデータ変換コードを作成できます。 422 ページの『WebSphere MQ for Windows 用のデータ変換出口の作成』および 418 ページの『WebSphere MQ (UNIX and Linux システム用) 用のデータ変換出口の作成』を参照してください。
     - **crtmqcvx** コマンドを使用して生成した既存のデータ変換出口がある場合は、更新後のコマンドを使用して出口を再生成する必要があります。
     - データ変換出口を作成するための詳細については、416 ページの『データ変換出口の作成』を参照してください。
   - 事前接続出口
     - MQNXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
     - **pEntryPoints** を使用する前に、MQNXP のバージョン番号がバージョン 2 以上であることを確認してください。
     - 事前接続出口を作成するための詳細については、424 ページの『リポジトリーから接続前出口を使用した接続定義の参照』を参照してください。
   - パブリッシュ出口
     - MQPSXP 構造体の **pEntryPoints** パラメーターを使用して、MQI 呼び出しと DCI 呼び出しを参照します。
     - **pEntryPoints** を使用する前に、MQPSXP のバージョン番号がバージョン 2 以上であることを確認してください。

- パブリッシュ出口を作成するための詳細については、426 ページの『パブリッシュ出口の作成とコンパイル』を参照してください。
- クラスター・ワークロード出口
  - MQWXP 構造体の **pEntryPoints** パラメーターを使用して、MQXCLWLN 呼び出しを参照します。
  - **pEntryPoints** を使用する前に、MQWXP のバージョン番号がバージョン 4 以上であることを確認してください。
  - クラスター・ワークロード出口を作成するための詳細については、428 ページの『クラスター・ワークロード出口の作成とコンパイル』を参照してください。

MQPUT を呼び出すチャネル出口の例を以下に示します。

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

その他の例については、96 ページの『WebSphere MQ プログラムのサンプル』を参照してください。

3. 出口をコンパイルします。
   - IBM WebSphere MQ ライブラリーにリンクしません。
   - IBM WebSphere MQ ライブラリーへの埋め込み RPath を出口に組み込みません。
   - 出口をコンパイルするための詳細については、以下のいずれかのトピックを参照してください。
     - API 出口: 392 ページの『API 出口のコンパイル』。
     - チャネル出口、パブリッシュ出口、クラスター・ワークロード出口: 415 ページの『Windows、UNIX and Linux システムでのチャネル出口プログラムのコンパイル』。
     - データ変換出口: 416 ページの『データ変換出口の作成』。
4. 出口を以下のいずれかの場所に配置します。
   - 出口の構成時に完全に修飾する任意のパス
   - デフォルトの出口のパス。特定のインストール・ディレクトリーにあります。 例えば、*MQ_DATA_PATH*/exits/installation2 のようになります。
   - デフォルトの出口のパス

     デフォルトの出口のパスは、32 ビット出口の場合が *MQ_DATA_PATH*/exits、64 ビット出口の場合が *MQ_DATA_PATH*/exits64 になります。 qm.ini ファイルまたは mqclient.ini ファイルでこれらのパスを変更することもできます。 詳細については、出口パスを参照してください。 Windows および Linux では、 WebSphere MQ エクスプローラーを使用してパスを変更できます。

     a. キュー・マネージャー名を右クリックします。
     b.「**プロパティー...**」をクリックします。
     c.「**出口**」をクリックします。
     d.「出口デフォルト・パス」フィールドで、出口プログラムを保持するディレクトリーのパス名を指定します。

   特定のインストール・ディレクトリーとデフォルト・パス・ディレクトリーの両方に出口を配置すると、パスに名前が含まれている WebSphere MQ インストール環境では、特定のインストール・ディレクトリーにある出口が使用されます。 例えば、/exits/installation2 と /exits に出口を配置し、/exits/installation1 には出口を配置しない場合があるとします。 WebSphere MQ インストール環境 installation2 では、/exits/installation2 の出口が使用されます。 WebSphere MQ インストール環境 installation1 では、/exits ディレクトリーの出口が使用されます。
5. 必要に応じて、出口を構成します。
   - インストール可能サービス: 386 ページの『構成サービスおよびコンポーネント』。

- API 出口: 396 ページの『API 出口の構成』。
- チャネル出口: 416 ページの『チャネル出口の構成』。
- パブリッシュ出口: 427 ページの『パブリッシュ出口の構成』。
- 事前接続出口: 425 ページの『クライアント構成ファイルの PreConnect スタンザ』。

UNIX、Linux、および Windows 用のインストール可能サービスとコンポーネント

このセクションでは、インストール可能サービス、およびそれと関連した機能とコンポーネントについて紹介します。 これらの機能へのインターフェースは文書化されているため、お客さままたはソフトウェア・ベンダーがコンポーネントを供給することも可能です。

WebSphere MQ インストール可能サービスが提供されている主な理由は、以下の とおりです。

- WebSphere MQ 製品に付属のコンポーネントを使用するか、または他のコンポーネントで置換または追加するか を柔軟に選択できるようにするため。
- ベンダーが WebSphere MQ 製品の内部を変更しなくても、新規のテクノロジーなどを使用する コンポーネントを供給して参入できるようにするため。
- WebSphere MQ が新しいテクノロジーをより迅速かつ安価に利用できるようにすることで、 製品を早期にかつ低価格で提供できるようにするため。

インストール可能サービス とサービス・コンポーネント は、WebSphere MQ 製品構成の 一部です。 この構成の中心には、メッセージ・キュー・インターフェース (MQI) に関連する機能と規則を実装するキュー・マネージャーがあります。 この中心部には、その作業を実行するためのインストール可能サービス と呼ばれるいくつかのサービス機能が必要です。 インストール可能サービスは、以下のとおりです。

- 許可サービス
- ネーム・サービス

各インストール可能サービスは、1 つ以上のサービス・コンポーネント を使用して実装された関連機能セットです。 各コンポーネントは、適切に構築された一般的に利用可能なインターフェースを使用して呼び出されます。 これにより、独立系ソフトウェア・ベンダーおよび他のサード・パーティーがインストール可能コンポーネントを提供して、WebSphere MQ 製品に付属の機能を拡張したり置換したりできるようになります。 379 ページの表 52 は、使用可能なサービスおよびコンポーネントを一覧にまとめたものです。

表 *52.* インストール可能サービス・コンポーネントの要約

インストール可能サービス	**提供コンポーネント**	**関数**	**要件**
許可サービス	オブジェクト権限マネージャー (OAM) (object authority manager (OAM))	コマンドおよび MQI 呼び出しの許可検査を行う。 ユーザーは独自のコンポーネントを作成して、OAM に追加したり、それを置き換えることができます。 例えば、ユーザー ID にキューをオープンする権限があるかどうかの確認など。	(適切なプラットフォーム許可機能を前提とする。)
ネーム・サービス	なし	指定のキューを所有するキュー・マネージャーの名前を検索するよう、キュー・マネージャーをサポートする。 • ユーザー定義 **注 :** 共用キューでは、*Scope* 属性を CELL に設定する必要がある。	• サード・パーティーまたはユーザー作成のネーム・マネージャーで必要とされる。

インストール可能サービスのインターフェースは、インストール可能サービス・インターフェースの参照情報で説明されています。

サービス・コンポーネントの作成

このセクションでは、サービス、コンポーネント、エントリー・ポイント、および戻りコードの関係について説明します。

機能およびコンポーネント

各サービスは、関連機能のセットで構成されます。 例えば、ネーム・サービスには以下の機能が含まれます。

- キュー名を検索して、キューが定義されているキュー・マネージャーの名前を戻します。
- キュー名をサービスのディレクトリーに挿入します。
- キュー名をサービスのディレクトリーから削除します。

さらに、初期化と終了の機能も含まれています。

インストール可能サービスは、1 つ以上のサービス・コンポーネントによって 提供されます。 各コンポーネントは、そのサービスに定義されている機能の一部または全部を実行できます。 例えば、WebSphere MQ for AIX では、提供されている許可サービス・コンポーネントである OAM は 使用可能な機能すべてを実行します。 詳しくは、383 ページの『許可サービス・インターフェース』を参照してください。 コンポーネントは、サービスのインプリメントに必要な基礎となるリソースやソフトウェア (例えば、LDAP ディレクトリー) を管理する役割も果たしています。 構成ファイルは、コンポーネントをロードして、提供される機能ルーチンのアドレスを判別するための標準的な手段を用意してくれます。

380 ページの図 72 は、サービスとコンポーネントがどのように関係しているかを示しています。

- サービスは構成ファイル内のスタンザによってキュー・マネージャーに定義されます。
- 各サービスは、キュー・マネージャーに提供されているコードによってサポートされます。 ユーザーはこのコードを変更できないので、独自のサービスを作成することはできません。
- 各サービスは 1 つ以上のコンポーネントによってインプリメントされます。これらは製品に付属のものを使用したり、ユーザーで作成したりすることができます。 1 つのサービスに対して複数のコンポーネントを起動し、それぞれのコンポーネントにサービス内の異なる機能をサポートさせることもできます。
- エントリー・ポイントはサービス・コンポーネントをキュー・マネージャー内の サポートされているコードに接続します。

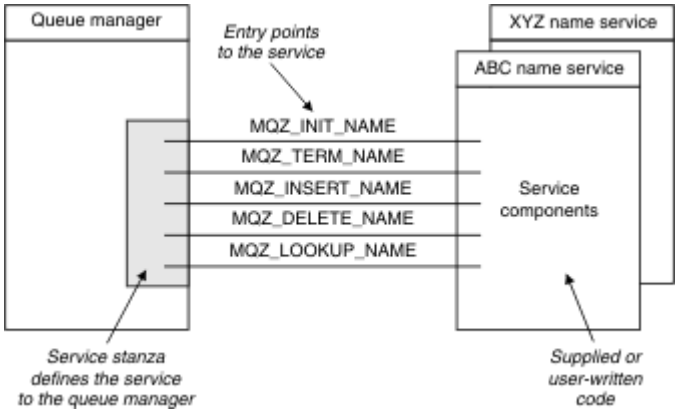


図 72. サービス、コンポーネント、およびエントリー・ポイントの簡単な関係図

エントリー・ポイント

各サービス・コンポーネントは、特定のインストール可能サービスをサポートする ルーチンのエントリー・ポイント・アドレスのリストによって表されます。 インストール可能サービスは、各ルーチンによって実行される機能を定義します。

構成する際にサービス・コンポーネントを配列する順序は、サービスの要求を満たすためにエントリー・ポイントが呼び出される順序を定義するものです。

提供されているヘッダー・ファイル cmqzc.h では、各サービスへの提供されているエントリー・ポイントに MQZID_ の接頭部があります。

サービスが存在する場合、サービスは事前定義の順序に読み込まれます。 次のリストは、サービスとその初期化の順序を示しています。

1. NameService
2. AuthorizationService
3. UserIdentifierService

AuthorizationService は、デフォルトで構成される唯一のサービスです。 NameService および UserIdentifierService を使用するには、それらを手動で構成する必要があります。

サービスおよびサービス・コンポーネントでは、1 対 1 または 1 対多のマッピングがあります。 サービスごとに、複数のサービス・コンポーネントを定義できます。 UNIX and Linux システムでは、ServiceComponent スタンザのサービス値は、qm.ini ファイルにあるサービス・スタンザの Name 値と一致する必要があります。 Windows では、 ServiceComponent のサービス・レジストリー・キー値は、名前レジストリー・キー値と一致する必要があり、HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\Configuration\QueueManager*qmname*\ として定義されます。ここで、 *qmname* はキュー・マネージャーの名前です。

UNIX and Linux システムでは、サービス・コンポーネントは qm.ini ファイルに定義されている順序で開始されます。 Windows では、Windows レジストリーが使用されるため、WebSphere MQ は **RegEnumKey** 呼び出しを実行し、値はアルファベット順に返されます。 そのため、Windows では、サービスはレジストリーで定義されているとおりにアルファベット順で呼び出されます。

ServiceComponent 定義の順序付けは重要です。 この順序付けにより、特定のサービスでコンポーネントが実行される順序が決まります。 例えば、Windows で AuthorizationService は、MQSeries.WindowsNT.auth.service という名前のデフォルトの OAM コンポーネントで構成されています。 このサービスで追加のコンポーネントを定義して、デフォルトの OAM をオーバーライドすることができます。 MQCACF_SERVICE_COMPONENT を指定しない限り、アルファベット順で最初に来るコンポーネントが、要求を処理するために使用され、そのコンポーネントの名前が使用されます。

戻りコード

サービス・コンポーネントは、キュー・マネージャーに戻りコードを提供して、さまざまな状況を報告します。 それらは処理の成功または失敗を報告して、キュー・マネージャーが次のサービス・コンポーネントに進むかどうかを示します。 別の *Continuation* パラメーターが、この指示を伝えます。

コンポーネント・データ

単一のサービス・コンポーネントでは、データをさまざまな機能で共用することが必要になることがあります。 インストール可能なサービスは、サービス・コンポーネントの各呼び出しで受け渡しするためにオプションのデータ域を提供します。 このデータ域は、サービス・コンポーネントが専用に使用します。 これは、アドレス・スペースまたはプロセスが異なる場合でも、特定の機能のすべての呼び出しによって共用されます。 呼び出される場合は常にサービス・コンポーネントからアクセス可能であることを保証されます。 このデータ域のサイズを *ServiceComponent* スタンザで宣言する必要があります。

コンポーネントの初期化と終了

コンポーネントの初期化オプションおよび終了オプションの使用法。

コンポーネントの初期化ルーチンが呼び出されるときには、コンポーネントによってサポートされるエントリー・ポイントごとにキュー・マネージャーの **MQZEP** 関数を呼び出す必要があります。 **MQZEP** は、サービスにエントリー・ポイントを定義します。 未定義の出口点はすべて NULL と見なされます。

コンポーネントは、他の方法によって呼び出される前に、1 次初期化オプションによって必ず一度呼び出されます。

特定のプラットフォームでは、コンポーネントを 2 次初期化オプションで呼び出すことができます。 例えば、サービスにアクセスするオペレーティング・システム・プロセス、スレッド、またはタスクごとに一度呼び出すことができます。

2 次初期化が使用されると、次のようになります。

- コンポーネントを 2 次初期化のために複数回呼び出すことができます。 サービスが不要になったときは、そのような呼び出しごとに、対応する 2 次終了呼び出しが発行されます。

  ネーミング・サービスでは、これは MQZ_TERM_NAME 呼び出しです。

  許可サービスの場合、これは MQZ_TERM_AUTHORITY 呼び出しです。
- 1 次初期化および 2 次初期化のためにコンポーネントが呼び出されるたびに、エントリー・ポイントを (MQZEP を呼び出して) 再指定する必要があります。
- コンポーネントに対して使用されるコンポーネント・データのコピーは 1 つのみです。 2 次初期化ごとに異なるコピーが使用されるわけではありません。
- 2 次初期化が実行されるまで、そのコンポーネントが (状況に応じて、オペレーティング・システム・プロセス、スレッド、またはタスクからの) サービスに対する他の呼び出しのために起動されることはありません。
- コンポーネントは、*Version* パラメーターに 1 次初期化と 2 次初期化で同じ値を設定する必要があります。

コンポーネントは、不要になったときには必ず、1 次終了オプションにより一度呼び出されます。 このコンポーネントに対して、それ以降の呼び出しは行われません。

コンポーネントが 2 次初期化のために呼び出されている場合は、2 次終了オプションにより呼び出されます。

オブジェクト権限マネージャー *(OAM)*

WebSphere MQ 製品に付属の許可サービス・コンポーネントは、オブジェクト権限マネージャー (OAM) と呼ばれます。

デフォルトでは、OAM はアクティブであり、制御コマンドの **dspmqaut** (権限の表示)、**dmpmqaut** (権限のダンプ)、および **setmqaut** (権限の設定とリセット) を処理します。

これらのコマンドの構文と使用方法については、制御コマンドで説明されています。

OAM は、プリンシパルまたはグループのエンティティー で動作します。

- UNIX and Linux システムの場合:
  - プリンシパルはユーザー ID、またはユーザーの代理で実行されるアプリケーション・プログラムに関連付けられた ID です。
  - グループは UNIX または Linux システムにより定義されるプリンシパルの集合です。
  - 権限は、グループ・レベルでのみ認可または取り消しできます。 ユーザーの権限の認可または取り消し要求により、そのユーザーの 1 次グループが更新されます。
- Windows システムの場合:
  - プリンシパルは Windows ユーザー ID、またはユーザーの代理で実行されるアプリケーション・プログラムに関連付けられた ID です。
  - グループは Windows グループです。
  - 権限はプリンシパル・レベルまたはグループ・レベルで認可または取り消しできます。

MQI 要求が行われるかコマンドが発行されると、OAM は処理に関連するエンティティーの権限を検査して、次のことが可能であるかどうかを確認します。

- 要求された操作の実行。
- 指定されたキュー・マネージャー・リソースへのアクセス。

許可サービスによって、独自の許可サービス・コンポーネントを作成することにより、キュー・マネージャー用に備えられている権限検査を拡張または置換することができます。

ネーム・サービス
ネーム・サービスは、キュー・マネージャーにサポートを提供して、指定のキューを所有するキュー・マネージャーの名前を検索できるようにするインストール可能サービスです。 ネーム・サービスから検索できる他のキュー属性はありません。

ネーム・サービスにより、アプリケーションは出力用のリモート・キューをローカル・キューであるかのようにオープンすることが可能になります。 ネーム・サービスは、キュー以外のオブジェクトに対しては呼び出されません。

注 : リモート・キューの *Scope* 属性は CELL に設定されて**いなければなりません**。

アプリケーションではキューをオープンする場合、キュー・マネージャーのディレクトリー内で最初にそのキューの名前を探します。 そこに存在しない場合、そのキュー名が認識されるものが見つかるまで、構成されているすべてのネーム・サービスを検索します。 名前が認識されるものが見つからない場合、オープンは失敗します。

ネーム・サービスはそのキューを所有するキュー・マネージャーを戻します。 その後キュー・マネージャーは、コマンドの最初の要求にキューおよびキュー・マネージャー名が指定されていた場合と同様に MQOPEN 要求を続行します。

ネーム・サービス・インターフェース (NSI) は、WebSphere MQ フレームワークの一部です。

ネーム・サービスの機能

キュー定義で *Scope* 属性がキュー・マネージャー (つまり MQSC の SCOPE(QMGR)) として指定されている場合、キュー定義は (すべてのキュー属性と共に) キュー・マネージャーのディレクトリーのみに保管されます。 これは、インストール可能サービスに置き換えることはできません。

キュー定義で *Scope* 属性がセル (つまり MQSC の SCOPE(CELL)) として指定されている場合も、キュー定義は (すべてのキュー属性と共に) キュー・マネージャーのディレクトリーに保管されます。 ただし、キューおよびキュー・マネージャー名はネーム・サービスにも保管されます。 この情報を保管できるサービスがない場合、*Scope* セルのあるキューは定義できません。

情報の保管先ディレクトリーは、サービスによって管理できます。 または、サービスが、基礎となるサービス (LDAP ディレクトリーなど) をこの目的で使用することもできます。 いずれの場合にも、ディレクトリーに保管される定義は、明示的に削除されるまで、コンポーネントおよびキュー・マネージャーが終了した後も保管され続ける必要があります。

注 :

1. ネーミング・ディレクトリー・セル内の異なるキュー・マネージャーに対するリモート・ホストのローカル・キュー定義に、(スコープが CELL の) メッセージを送信するには、チャネルを定義する必要があります。
2. リモート・キューのスコープが CELL となっている場合でも、リモート・キューから直接メッセージを取得することはできません。
3. スコープが CELL のキューを送信する場合、リモート・キュー定義は必要ありません。
4. ネーミング・サービスは主に宛先キューを定義します。 ただし、宛先キュー・マネージャーへの伝送キューと 1 対のチャネル定義は引き続き必要となります。 さらに、ローカル・システム上の伝送キューの名前は、スコープが CELL の宛先キューを所有するリモート・システム上のキュー・マネージャーの名前と同じでなければなりません。

   例えば、リモート・キュー・マネージャーの名前が QM01 である場合、ローカル・システム上の伝送キューの名前も QM01 でなければなりません。

許可サービス・インターフェース
許可サービスは、キュー・マネージャーで使用されるエントリー・ポイントを提供しています。

エントリー・ポイントは次のとおりです。

MQZ_AUTHENTICATE_USER
ユーザー ID およびパスワードを認証し、アイデンティティー・コンテキスト・フィールドを設定できます。

MQZ_CHECK_AUTHORITY
エンティティーが、指定されたオブジェクトに 1 つ以上の操作を実行する権限を所有しているかどうかを検査します。

MQZ_CHECK_PRIVILEGED
指定されたユーザーが特権ユーザーであるかどうかを検査します。

MQZ_COPY_ALL_AUTHORITY
参照されたオブジェクトに存在する現在のすべての許可を別のオブジェクトにコピーします。

MQZ_DELETE_AUTHORITY
指定されたオブジェクトに関連するすべての許可を削除します。

MQZ_ENUMERATE_AUTHORITY_DATA
指定された選択基準に一致するすべての権限データを検索します。

MQZ_FREE_USER
関連した割り振られたリソースを解放します。

MQZ_GET_AUTHORITY
指定されたオブジェクトに対してエンティティーがアクセスする権限を取得します。

MQZ_GET_EXPLICIT_AUTHORITY
名前付きグループが指定されたオブジェクトにアクセスする権限 (ただし、**nobody (なし)** グループの追加権限はなし) または名前付きプリンシパルの 1 次グループが指定されたオブジェクトにアクセスする権限を取得します。

MQZ_INIT_AUTHORITY
許可サービス・コンポーネントを初期化します。

MQZ_INQUIRE
サポートされている許可サービスの機能を照会します。

MQZ_REFRESH_CACHE
許可をすべてリフレッシュします。

MQZ_SET_AUTHORITY
指定されたオブジェクトに対してエンティティーが所有する権限を設定します。

MQZ_TERM_AUTHORITY
許可サービス・コンポーネントを終了します。

さらに、WebSphere MQ for Windows 上で、許可サービスはキュー・マネージャーで使用するために以下のエントリー・ポイントを提供しています。

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

これらのエントリー・ポイントは、Windows Security Identifier (NT SID) を サポートします。

これらの名前は、ヘッダー・ファイル `cmqzc.h` に **typedef** として指定され、コンポーネント機能のプロトタイピングに使用できます。

初期化関数 (**MQZ_INIT_AUTHORITY**) は、コンポーネントの主エントリー・ポイントでなければなりません。 他の機能は、初期化機能がコンポーネントのエントリー・ポイントのベクトルに追加したエントリー・ポイント・アドレスを介して起動されます。

ネーム・サービス・インターフェース
ネーム・サービスは、キュー・マネージャーが使用するエントリー・ポイントを提供します。

以下のエントリー・ポイントが提供されます。

MQZ_INIT_NAME
ネーム・サービス・コンポーネントを初期化します。

MQZ_TERM_NAME
ネーム・サービス・コンポーネントを終了します。

MQZ_LOOKUP_NAME
指定したキューのキュー・マネージャー名を検索します。

MQZ_INSERT_NAME
指定したキューを所有するキュー・マネージャーの名前を含む項目を、サービスが使用するディレクトリーに挿入します。

MQZ_DELETE_NAME
指定したキューの項目を、サービスが使用するディレクトリーから削除します。

複数のネーム・サービスが構成されている場合は、次のことが行われます。

- 検索では、キュー名が解決するまで、MQZ_LOOKUP_NAME 関数がリスト内の各サービスに対して呼び出されます (コンポーネントに検索を終了する指示が含まれていない場合)。
- 挿入では、MQZ_INSERT_NAME 関数がこの関数をサポートするリスト内の最初のサービスに対して呼び出されます。
- 削除では、MQZ_DELETE_NAME 関数がこの関数をサポートするリスト内の最初のサービスに対して呼び出されます。

挿入関数と削除関数をサポートするコンポーネントが複数あってはなりません。 ただし、検索のみをサポートするコンポーネントは可能であり、使用できます。 例えば、リスト内の最後のコンポーネントとして使用し、他のネーム・サービス・コンポーネントに認識されない任意の名前を、その名前を定義できるキュー・マネージャーに解決します。

C プログラミング言語では、typedef ステートメントを使用して、名前が関数データ型として定義されます。 これらを使用して、サービス関数をプロトタイプ化し、パラメーターが正しいことを確認できます。

インストール可能サービスに特定のすべての情報を含むヘッダー・ファイルは、C 言語では cmqzc.h です。

コンポーネントのメイン・エントリー・ポイントでなければならない初期化関数 (MQZ_INIT_NAME) とは別に、初期化関数が MQZEP 呼び出しによって追加したエントリー・ポイント・アドレスによっても関数が呼び出されます。

複数のサービス・コンポーネントの使用
1 つのサービスに複数のコンポーネントをインストールできます。 これにより、コンポーネントがサービスの実装の一部のみを提供して、残りの機能は他のコンポーネントが提供するようにすることができます。

複数のコンポーネントの使用例

ABC_name_serv および XYZ_name_serv という 2 つのネーム・サービス・コンポーネントを作成するとします。

ABC_name_serv
このコンポーネントは、サービス・ディレクトリーへの名前の挿入、およびサービス・ディレクトリーからの名前の削除をサポートしますが、キュー名の検索はサポートしません。

XYZ_name_serv
このコンポーネントはキュー名の検索をサポートしますが、サービス・ディレクトリーへの名前の挿入、およびサービス・ディレクトリーからの名前の削除はサポートしません。

コンポーネント ABC_name_serv は、キュー名のデータベースを保持し、2 つの単純なアルゴリズムを使用してサービス・ディレクトリーに名前を挿入するか、サービス・ディレクトリーから名前を削除します。

コンポーネント XYZ_name_serv は、そのコンポーネントの呼び出しに使用されたキュー名に対して、固定のキュー・マネージャー名を戻す、簡単なアルゴリズムを使用します。 キュー名のデータベースは保持されないので、挿入機能および削除機能はサポートされていません。

コンポーネントは、同じキュー・マネージャーにインストールされます。 *ServiceComponent* スタンザは、コンポーネント ABC_name_serv が最初に呼び出されるように配列されています。 コンポーネント・ディレクトリーでキューを挿入または削除する呼び出しは、コンポーネント ABC_name_serv によって処理されます。 このコンポーネントのみが、これらの機能をインプリメントします。 ただし、コンポーネント ABC_name_serv が解決できないルックアップ呼び出しは、ルックアップ専用コンポーネント

XYZ_name_serv に渡されます。 このコンポーネントは、その簡単なアルゴリズムによってキュー・マネージャー名を提供します。

複数のコンポーネントを使用する際にエントリー・ポイントを省略する

複数のコンポーネントを使用してサービスを提供することにした場合、一部の機能を実装しないサービス・コンポーネントを設計することができます。 インストール可能サービスのフレームワークでは、何を省略できるかについての制限はありません。 ただし、ある特定のインストール可能サービスでは、機能を 1 つ以上省略すると、サービスの目的と論理的に矛盾する場合があります。

複数のコンポーネントと共に使用されるエントリー・ポイントの例

386 ページの表 53 に、2 つのコンポーネントがインストールされているインストール可能なネーム・サービスの例を示します。 各コンポーネントは、この特定のインストール可能サービスに関連した機能の異なるセットをサポートしています。 挿入機能については、ABC コンポーネントのエントリー・ポイントが最初に呼び出されます。 サービスに (**MQZEP** を使用して) 定義されていないエントリー・ポイントは、NULL であると見なされます。 この表には初期化のエントリー・ポイントも示されていますが、初期化はコンポーネントのメイン・エントリー・ポイントによって実行されるため、これは必要ありません。

キュー・マネージャーでは、インストール可能サービスを使用する必要が生じると、そのサービスに定義されたエントリー・ポイント (386 ページの表 53 の列に示されています) を使用します。 キュー・マネージャーは各コンポーネントを順番に検討して、必要な機能を実装するルーチンのアドレスを判別します。 その後、ルーチンが存在すればそれを呼び出します。 処理が成功すると、結果および状況についてのすべての情報がキュー・マネージャーによって使用されます。

表 *53.* インストール可能サービスのエントリー・ポイントの例

機能番号	**ABC ネーム・サービス・コンポーネント**	**XYZ ネーム・サービス・コンポーネント**
MQZID_INIT_NAME (初期化)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (終了)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (挿入)	ABC_Insert()	NULL
MQZID_DELETE_NAME (削除)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (検索)	NULL	XYZ_Lookup()

ルーチンが存在しない場合、キュー・マネージャーはリストの次のコンポーネントに対して、このプロセスを繰り返します。 さらに、ルーチンが存在しても、処理が実行できなかったことを示すコードが戻された場合、使用可能な次のコンポーネントで試行を続けます。 サービス・コンポーネント内のルーチンが、操作を実行するための試行を続けないように指示するコードを戻す場合もあります。

構成サービスおよびコンポーネント

キュー・マネージャー構成ファイルを使用して、サービス・コンポーネントを構成します。ただし、各キュー・マネージャー構成ファイルがレジストリーに独自のスタンザを持つ Windows システムを除きます。

1. キュー・マネージャー構成ファイルにスタンザを追加して、キュー・マネージャーへのサービスを定義し、モジュールの場所を指定します。

   使用される各サービスは、サービスをキュー・マネージャーに定義する *Service* スタンザ を持っている必要があります。

   サービス内の各コンポーネントには、*ServiceComponent* スタンザが必要です。 これにより、そのコンポーネントのコードを含むモジュールの名前とパスを識別します。

   詳しくは、387 ページの『Service スタンザの形式』および 387 ページの『Service component スタンザの形式』を参照してください。

   オブジェクト権限マネージャー (OAM) と呼ばれる許可サービス・コンポーネントは、製品に同梱されています。 キュー・マネージャーを作成するとき、キュー・マネージャー構成ファイル (または Windows

システムのレジストリー) が自動的に更新されて、許可サービスおよびデフォルト・コンポーネント (OAM) のための適切なスタンザを内蔵するようになります。 その他のコンポーネントについては、キュー・マネージャー構成ファイルを手動で構成する必要があります。

各サービス・コンポーネントのコードは、プラットフォームが動的バインディングをサポートしている場合にはそれを使用して、キュー・マネージャーの始動時にキュー・マネージャーへロードされます。

2. キュー・マネージャーをいったん停止したあとで再始動して、コンポーネントをアクティブにします。

Service スタンザの形式

Service スタンザには、サービスの名前と、サービスに定義されたエントリー・ポイントの数が格納されています。

スタンザの形式は次のとおりです。

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

ここで、

<service_name>
サービスの名前。 これはサービスによって定義されます。

<entries>
このサービス用として定義するエントリー・ポイントの数。 これには、初期化エントリー・ポイントと終了エントリー・ポイントも含まれています。

Windows システムでの *Service* スタンザの形式

Windows システムでは、*Service* スタンザに SecurityPolicy 属性が含まれます。

スタンザの形式は次のとおりです。

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

ここで、

<service_name>
サービスの名前。 これはサービスによって定義されます。

<entries>
このサービス用として定義するエントリー・ポイントの数。 これには、初期化エントリー・ポイントと終了エントリー・ポイントも含まれています。

<policy>
NTSIDsRequired (Windows セキュリティー ID) または Default。 NTSIDsRequired を指定しない場合、Default 値が使用されます。 この属性は、Name に AuthorizationService の値が設定されている場合にのみ有効です。

388 ページの『許可サービス・スタンザの構成: Windows システム』も参照してください。

Service component スタンザの形式

サービス・コンポーネント・スタンザの形式は次のとおりです。

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

ここで、

<service_name>
サービスの名前。 これは、Service スタンザで指定された Name と一致する必要があります。

<component_name>
サービス・コンポーネントの記述名を指定します。 これは、固有名でなければならず、しかも WebSphere MQ オブジェクトの名前 (例えばキュー名) として有効な文字のみが使用されていなければなりません。 この名前は、サービスで生成されるオペレーター・メッセージに表示されます。 企業の商標や、それに類似した識別性の高いストリングで始まる名前を使用することをお勧めします。

<module_name>
このコンポーネントのコードを含むモジュールの名前。

<size>
各呼び出しでコンポーネントに渡される コンポーネント・データ域のバイト単位のサイズ。 コンポーネント・データが必要ない場合は、ゼロを指定します。

これらの 2 つのスタンザは任意の順序で指定できます。 スタンザにあるスタンザ・キーも、任意の順序で指定できます。 これらのスタンザの両方に、すべてのスタンザ・キーが存在する必要があります。 スタンザ・キーが重複する場合は、最後のキーが使用されます。

起動時に、キュー・マネージャーは構成ファイル内の各サービス・コンポーネント項目を順番に処理します。 その後、指定のコンポーネント・モジュールをロードし、コンポーネントのエントリー・ポイント (これはコンポーネントの初期化のエントリー・ポイントでなければなりません) を呼び出して、それに構成ハンドルを渡します。

許可サービス・スタンザの構成: *UNIX and Linux* システム

UNIX and Linux システムでは、各キュー・マネージャーに専用のキュー・マネージャー構成ファイルがあります。

例えば、キュー・マネージャー QMNAME のキュー・マネージャー構成ファイルのデフォルトのパスおよびファイル名は `/var/mqm/qmgrs/QMNAME/qm.ini` です。

デフォルト許可コンポーネントの *Service* スタンザおよび *ServiceComponent* スタンザは、`qm.ini` に自動的に追加されますが、これは `mqsnoaut` によって指定変更することができます。 その他の *ServiceComponent* スタンザは、すべて手動で追加する必要があります。

例えば、キュー・マネージャー構成ファイルに含まれる以下のスタンザは、WebSphere MQ for AIX 上の 2 つの許可サービス・コンポーネントを定義します。 *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

```
Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=MQ_INSTALLATION_PATH/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

図 *73. qm.ini* 内の *UNIX and Linux* 許可サービス・スタンザ

サービス・コンポーネント・スタンザ (`MQSeries.UNIX.auth.service`) では、デフォルトの許可サービス・コンポーネントである OAM を定義します。 このスタンザを削除してキュー・マネージャーを再始動すると、OAM は使用不可となり、許可検査は行われません。

許可サービス・スタンザの構成: *Windows* システム

WebSphere MQ for Windows では、各キュー・マネージャーについて、レジストリー内にそれぞれ専用のスタンザが 1 つあります。

デフォルト許可コンポーネントの *Service* スタンザおよび *ServiceComponent* スタンザはレジストリーに自動的に追加されますが、これは mqsnoaut を使用して指定変更することができます。 その他の *ServiceComponent* スタンザは、すべて手動で追加する必要があります。

さらに、SecurityPolicy 属性も WebSphere MQ サービスを使用して追加できます。
SsecurityPolicy 属性が適用されるのは、*Service* スタンザに指定されているサービスが、許可サービス、つまりデフォルトの OAM である場合のみです。 SecurityPolicy 属性は、各キュー・マネージャーの セキュリティー・ポリシーを指定するために使用できます。 指定できる値は以下のとおりです。

Default
デフォルトのセキュリティー・ポリシーを有効にする場合は、Default を指定します。 ある特定のユーザー ID の Windows セキュリティー ID (NT SID) が OAM に渡されなかった場合は、関連のセキュリティー・データベース内の検索によって、該当する SID が取得されます。

NTSIDsRequired
セキュリティー検査を行うときに、NT SID を OAM に渡す必要があります。

Service スタンザの形式については、387 ページの『Windows システムでの Service スタンザの形式』を参照してください。 セキュリティーに関するより一般的な情報については、Windows、UNIX and Linux システムでのセキュリティーのセットアップを参照してください。

サービス・コンポーネント・スタンザ (MQSeries.WindowsNT.auth.service) は、デフォルトの許可サービス・コンポーネントである OAM を定義します。 このスタンザを削除してキュー・マネージャーを再始動すると、OAM は使用不可となり、許可検査は行われません。

ネーム・サービス・スタンザの構成*: Unix* システムおよび *Linux* システム
ここに簡略説明を入力してください。先頭パラグラフと要約に使用されます。

ネーム・サービスのための UNIX and Linux 構成ファイル・スタンザである以下の例では、(架空の) ABC 社によって提供されるネーム・サービス・コンポーネントを指定します。

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

図 *74. qm.ini* のネーム・サービス・スタンザ *(UNIX and Linux* システムの場合*)*

注 : Windows システムでは、ネーム・サービス・スタンザ情報はレジストリーに保管されます。

ユーザーの権限変更後の *OAM* のリフレッシュ
WebSphere MQ では、ユーザーの許可グループ・メンバーシップを変更した直後に OAM の許可グループ情報をリフレッシュし、キュー・マネージャーを停止して再始動することなく、オペレーティング・システム・レベルで行われた変更を反映できます。 これを行うには、**REFRESH SECURITY** コマンドを発行します。

注 : setmqaut コマンドを使用して権限を変更すると、OAM はそのような変更を即時に実装します。

キュー・マネージャーは、権限データを SYSTEM.AUTH.DATA.QUEUE というローカル・キューに保管します。 このデータは amqzfuma.exe によって管理されます。

関連資料
REFRESH SECURITY

API 出口の作成とコンパイル

API 出口を使用すると、MQPUT および MQGET などの WebSphere MQ API 呼び出しの動作を変更するコードを作成して、これらの呼び出しの直前または直後に、そのコードを 挿入することができます。

注 : WebSphere MQ for z/OS ではサポートされない。

API 出口を使用する理由

各アプリケーションには実行すべき特定の ジョブがあり、アプリケーションのコードは、そのタスクをできるだけ効率的に実行しなければなりません。 もっと高いレベルで考えるなら、特定のキュー・ マネージャーを使用する**すべての** アプリケーション用に、キュー・マネージャーに対して標準またはビジネス・プロセスを 適用したいと思うことがあるかもしれません。 個々のアプリケーションより上のレベルで実行することにより、影響を受ける各アプリケーションのコードを 変更しないで済むため、効率が良くなります。

以下は、API 出口が役立つ場合のある分野に関するいくつかの提案です。

- セキュリティー 上の理由で、アプリケーションがキューまたはキュー・マネージャーにアクセスする許可があることを検査するための認証を提供できます。 また、個々の API 呼び出し、またはそれらの呼び出しが使用するパラメーターをも認証することによって、アプリケーションによる API の使用を監視することもできます。
- 柔軟性 を持たせるため、ビジネス環境中のデータに依存するアプリケーションを変更せずに、その環境で敏速な変更に応答することができます。 例えば、金利、為替レート、または製造環境におけるコンポーネントの価格などの変更に応答する API 出口を持つことができます。
- キューまたはキュー・マネージャーの使用をモニター する場合、アプリケーションおよびメッセージの流れのトレース、API 呼び出しにおけエラーのログ記録、アカウントの目的での監査証跡のセットアップ、または計画の目的での使用法統計の収集などを実行できます。

API 出口の実行時に起きる事柄

出口プログラムを作成して、WebSphere MQ に対して識別しておけば、キュー・マネージャーは登録済みのポイントで自動的に出口コードを呼び出します。

実行する API 出口ルーチンは、 IBM i、 Windows、 UNIX and Linux の各システムのスタンザで識別されます。 このトピックでは、構成ファイル mqs.ini および qm.ini 内のスタンザについて説明します。

次の 3 つの場所でルーチンを定義できます。

1. mqs.ini ファイルにある ApiExitCommon は、キュー・マネージャー起動時に適用される、WebSphere MQ 全体のルーチンを識別します。 これらのルーチンは、個々のキュー・マネージャーに定義されるルーチンによって、オーバーライドすることができます (このリストの項目 390 ページの『3』 を参照)。
2. ApiExit テンプレート (mqs.ini ファイル内) は、新規キュー・マネージャーの作成時に ApiExitLocal セット (このリストの項目 390 ページの『3』 を参照) にコピーされる、 WebSphere MQ 全体のルーチンを識別します。
3. qm.ini ファイルにある ApiExitLocal は、特定のキュー・マネージャーに適用されるルーチンを識別します。

新しいキュー・マネージャーが作成されると、mqs.ini の ApiExitTemplate 定義が、その新しいキュー・マネージャーの qm.ini の ApiExitLocal 定義にコピーされます。 キュー・マネージャーが開始すると、ApiExitCommon および ApiExitLocal 定義が使用されます。 両方とも同じ名前のルーチンを識別する場合、ApiExitLocal 定義が ApiExitCommon 定義を置き換えます。 396 ページの『API 出口の構成』で説明されている Sequence 属性により、スタンザで定義されたルーチンの実行順序が決まります。

WebSphere MQ の複数のインストール済み環境にわたる API 出口の使用

バージョン 7.1 の出口に加えた変更により、以前のバージョンと協働しなくなる場合があるので、すべてのバージョンで機能するように以前のバージョンの WebSphere MQ 用に書かれた API 出口を使用していることを確認してください。 出口に加えられた変更について詳しくは、376 ページの『出口とインストール可能サービスの作成とコンパイル』を参照してください。

API 出口 amqsaem および amqsaxe 用に提供されているサンプルは、出口の作成時に必要な変更を反映しています。 アプリケーションの起動の前に、正しい WebSphere MQ ライブラリー (アプリケーションが関連付けられているキュー・マネージャーのインストール済み環境に対応するもの) がクライアント・アプリケーションにリンクされていることを確認する必要があります。

API 出口の作成

C プログラミング言語を使用して、各 API 呼び出しに出口を作成できます。

以下のように、各 API 呼び出しで出口を使用できます。

- MQCB。指定したオブジェクト・ハンドルにコールバックを再登録し、コールバックに対するアクティベーションと変更を制御します。
- MQCTL。接続にオープンされたオブジェクト・ハンドルに対する制御アクションを実行します。
- MQCONN/MQCONNX。後続の API 呼び出しで使用される、キュー・マネージャーの接続ハンドルを提供します。
- MQDISC。キュー・マネージャーから切断します。
- MQBEGIN。グローバル作業単位 (UOW) を開始します。
- MQBACK。UOW をバックアウトします。
- MQCMIT。UOW をコミットします。
- MQOPEN。後続のアクセス用に WebSphere MQ リソースをオープンします。
- MQCLOSE。アクセス用に既にオープンされている WebSphere MQ リソースをクローズします。
- MQGET。アクセス用に既にオープンされているキューからメッセージを取得します。
- MQPUT1。メッセージをキューに入れます。
- MQPUT。アクセス用に既にオープンされているキューにメッセージを入れます。
- MQINQ。アクセス用に既にオープンされている WebSphere MQ リソースの属性に対して照会を実行します。
- MQSET。アクセス用に既にオープンされているキューの属性を設定します。
- MQSTAT。状況情報を取得します。
- MQSUB。特定のトピックに対して、アプリケーションのサブスクリプションを登録します。
- MQSUBRQ。サブスクリプションに対して要求を実行します。

MQ_CALLBACK_EXIT は、コールバック処理の前と後に実行する出口機能を提供します。 詳しくは、コールバック - MQ_CALLBACK_EXIT を参照してください。

API 出口内で、呼び出しは一般的な形式を取ります。

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

ここで、*call* は MQ 接頭部を省略した MQI 呼び出し名、例えば PUT、GET などです。 *parameters* は出口の機能を制御し、主に出口と外部制御ブロック MQAXP (API 出口パラメーター構造体) および MQAXC (API 出口コンテキスト構造体) の間の通信を提供します。 *context* は、API 出口が呼び出されたコンテキストを記述し、*ApiCallParameters* は MQI 呼び出しに対するパラメーターを表します。

API 出口の作成に役立つように、サンプル出口 amqsaxe0.c が提供されています。この出口は、指定したファイルにトレース・エントリーを生成します。 出口の作成時に、このサンプルを開始点として使用できます。 サンプル出口の使用方法について詳しくは、112 ページの『API 出口サンプル・プログラム』を参照してください。

API 出口呼び出し、外部制御ブロック、および関連トピックについて詳しくは、API 出口参照を参照してください。

出口の作成、コンパイル、および構成に関する一般情報については、376 ページの『出口とインストール可能サービスの作成とコンパイル』を参照してください。

API 出口でのメッセージ・ハンドルの使用

API 出口がどのメッセージ・プロパティーに対してアクセス権を持つのかを 制御することができます。 プロパティーには ExitMsgHandle が関連付けられています。 書き込み出口で設定される プロパティーは書き込まれるメッセージに設定されますが、読み取り出口で取り出されるプロパティー はアプリケーションに戻されません。

Function を MQXF_INIT に設定し **ExitReason** を MQXR_CONNECTION に設定した MQXEP MQI 呼び出しを使用して、MQ_INIT_EXIT 出口機能を登録する とき、MQXEPO 構造体を **ExitOpts** パラメーターとして渡します。 MQXEPO 構造体に含まれる ExitProperties フィールドは、出口に対して使用可能にする プロパティーのセットを指定します。 これは、プロパティーの接頭部を表す文字ストリング (MQRFH2 フォルダー名に対応) として 指定されます。

各 API 出口は 1 つの MQAXP 構造体を受け取り、そこに ExitMsgHandle フィールドが 含まれています。 このフィールドは、WebSphere MQ によって生成される、接続ごとに 固有の値に設定されます。 従って、同じ接続にある同じタイプまたは異なるタイプの API 出口間で、 ハンドルは変化しません。

ExitReason が MQXR_BEFORE の MQ_PUT_EXIT または MQ_PUT1_EXIT、つまり、 メッセージを書き込む前に実行される API 出口では、出口が完了するときに ExitMsgHandle と関連付けられているすべてのプロパティー (メッセージ記述子プロパティーを除く) が、書き込まれるメッセージに設定されます。 これが起こるのを 防ぐには、ExitMsgHandle を MQHM_NONE に設定します。 また、異なるメッセージ・ハンドルを 指定することもできます。

MQ_GET_EXIT では、ExitMsgHandle のプロパティーはクリアされ、 MQ_INIT_EXIT が登録されたときに ExitProperties フィールドに指定されたプロパティー (メッセージ記述子プロパティーを除く) が 設定されます。 これらのプロパティーは、読み取りアプリケーションに 使用可能にはされません。 読み取りアプリケーションが MQGMO (メッセージ読み取りのオプション) フィールド にメッセージ・ハンドルを指定した場合、 そのハンドルと関連付けられているプロパティーはすべて (メッセージ記述子プロパティーも含めて)、 API 出口で使用可能です。 ExitMsgHandle にプロパティーが設定されるのを 防止するには、MQHM_NONE に設定します。

API 出口でのメッセージ・ハンドルの使用法を示すためのサンプル・プログラム amqsaem0.c が 用意されています。

API 出口のコンパイル

出口を作成した後、その出口を次のようにコンパイルし、リンクします。

次の例は、112 ページの『API 出口サンプル・プログラム』で説明されるサンプル・プログラムに使用されるコマンドを示します。 Windows システム以外のプラットフォームの場合、サンプル API 出口コードは *MQ_INSTALLATION_PATH*/samp にあり、コンパイル済みおよびリンク済みの共用ライブラリーは *MQ_INSTALLATION_PATH*/samp/bin にあります。 Windows システムの場合、サンプル API 出口コードは *MQ_INSTALLATION_PATH*\Tools\c\Samples にあります。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされたディレクトリーを表します。

ユーザーへの注 :

1. 64 ビット・アプリケーションのプログラミングに関するガイダンスは、64 ビット・プラットフォームでのコーディング標準にリストされています。

マルチキャスト・クライアントの導入とともに、 API 出口とデータ変換出口はクライアント・サイドで実行可能であることが必要になりました。 これは、一部のメッセージがキュー・マネージャーを経由しない可能性があるためです。 以下のライブラリーは、現在ではサーバー・パッケージだけでなくクライアント・パッケージの一部にもなっています。

表 *54.* 現在ではクライアント・パッケージとサーバー・パッケージの両方に含まれているライブラリー

オペレーティング・システム	**ライブラリー**
Windows	32 ビットおよび 64 ビット: mqm.dll および mqm.pdb
Linux & HP-UX	32 ビットおよび 64 ビット: libmqm.so および libmqm_r.so
AIX	32 ビットおよび 64 ビット: libmqm.a および libmqm_r.a
Solaris	32 ビットおよび 64 ビット: libmqm.so

Unix および *Linux* システムでの *API* 出口のコンパイル
UNIX および Linux システムで API 出口をコンパイルする方法の例。

どのプラットフォームでも、モジュールへの入り口点は MQStart です。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

AIX の場合

次のいずれかのコマンドを発行して API 出口ソース・コードをコンパイルします。

32 ビット・アプリケーション
非スレッド化

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション
非スレッド化

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

HP-UX Itanium プラットフォームの場合

32 ビット・アプリケーション
非スレッド化

API 出口ソース・コードをコンパイルします。

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

API 出口ソース・コードをリンクします。

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

スレッド化

API 出口ソース・コードをコンパイルします。

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

API 出口ソース・コードをリンクします。

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

64 ビット・アプリケーション
非スレッド化

API 出口ソース・コードをコンパイルします。

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

API 出口ソース・コードをリンクします。

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

スレッド化

API 出口ソース・コードをコンパイルします。

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

API 出口ソース・コードをリンクします。

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

Linux 上

次のいずれかのコマンドを発行して API 出口ソース・コードをコンパイルします。

31 ビット・アプリケーション
非スレッド化

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

32 ビット・アプリケーション
非スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション
非スレッド化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Solaris の場合

次のいずれかのコマンドを発行して API 出口ソース・コードをコンパイルします。

32 ビット・アプリケーション
SPARC プラットフォーム

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64 プラットフォーム

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

64 ビット・アプリケーション
SPARC プラットフォーム

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64 プラットフォーム

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Windows システムの場合:
Windows でサンプル API 出口プログラム amqsaxe0.c をコンパイルしてリンクします。

マニフェスト・ファイルは、コンパイルしたアプリケーションまたは DLL に埋め込むことのできる、バージョンその他の情報を含むオプション XML 文書です。

そのような文書がない場合は、**mt** コマンドの -manifest *manifest.file* パラメーターを省略できます。

395 ページの図 75 または 396 ページの図 76 の例のコマンドを、Windows 上の amqsaxe0.c をコンパイルしてリンクするように調整します。 コマンドは、Microsoft Visual Studio 2005、2008、または 2010 で処理されます。 この例では、 WebSphere MQ C:\Program Files\IBM\WebSphere MQ\tools\c\samples ディレクトリーが現行ディレクトリーであることを前提としています。

32 ビット

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def

amqsaxe0.obj \
     /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

図 *75. 32* ビットの *Windows* 上での *amqsaxe0.c* のコンパイルおよびリンク

64 ビット

```
cl /c /nologo /MD /Foamqsaxe0.obj amqsaxe0.c

link /nologo /dll /def:amqsaxe.def \
     /libpath:..\..\lib64 \


amqsaxe0.obj /manifest /out:amqsaxe.dll

mt -nologo -manifest amqsaxe.dll.manifest \
   -outputresource:amqsaxe.dll;2
```

図 *76. 64* ビットの *Windows* 上での *amqsaxe0.c* のコンパイルおよびリンク

関連概念

112 ページの『API 出口サンプル・プログラム』
サンプル API 出口は、ユーザー指定ファイルに MQI トレースを生成し、これに MQAPI_TRACE_LOGFILE 環境変数で定義した接頭部を付けます。

API 出口の構成

構成情報を変更することにより、API 出口を使用できるように IBM WebSphere MQ を構成します。

構成情報を変更するには、出口ルーチンとその実行順序を定義するスタンザを変更する必要があります。この情報は、以下の方法で変更できます。

- IBM WebSphere MQ Explorer の使用 (Windows および Linux (x86 および x86-64 プラットフォーム) の場合)
- **amqmdain** コマンドの使用 (Windows の場合)
- mqs.ini および qm.ini ファイルを直接使用する (Windows、UNIX and Linux システムの場合)。

  mqs.ini ファイルには、特定のノードのすべてのキュー・マネージャーに 関連する情報が含まれています。 これは、 UNIX and Linux の場合は /var/mqm ディレクトリーにあり、 Windows システムの場合は HKLM\SOFTWARE\IBM\WebSphere MQ キーで指定された WorkPath にあります。

  qm.ini ファイルには、特定のキュー・マネージャーに関係のある情報が含まれています。 各キュー・マネージャーごとに、1 つのキュー・マネージャー構成ファイルがあります。これは、キュー・マネージャーが占有するディレクトリー・ツリーのルートに保持されます。 例えば、QMNAME という名前のキュー・マネージャーの構成ファイルのパスと名前は、次のとおりです。

  UNIX and Linux システムの場合:

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows システムの場合:

  ```
  C:\Program Files\IBM\WebSphere MQ\qmgrs\QMNAME\qm.ini
  ```

  構成ファイルを編集する前に、必要な場合に復元できるようにバックアップをとっておいてください。

  構成ファイルの編集方法には次の 2 つがあります。

  – 自動。これには、ノード上のキュー・マネージャーの構成を変更するコマンドを使用します。

  – 手動。これには、標準テキスト・エディターを使用します。

  構成ファイル属性のどれかに誤った値を設定した場合、その値は無視され、問題を示すオペレーター・メッセージが表示されます (その結果、その属性をまったく指定しなかった場合と同じになります。)

構成するスタンザ

変更する必要があるスタンザは、以下のとおりです。

ApiExitCommon

mqs.ini と、IBM WebSphere MQ Explorer の IBM WebSphere MQ プロパティー・ページにある「出口」で定義されます。

キュー・マネージャーが始動すると、このスタンザの属性が読み取られ、qm.ini に定義された API 出口で指定変更されます。

ApiExitTemplate

mqs.ini と、IBM WebSphere MQ Explorer の IBM WebSphere MQ プロパティー・ページにある「出口」で定義されます。

キュー・マネージャーが作成されると、このスタンザの属性が ApiExitLocal スタンザの下の新たに作成された qm.ini ファイルにコピーされます。

ApiExitLocal

qm.ini と、IBM WebSphere MQ Explorer のキュー・マネージャー・プロパティー・ページにある「出口」で定義されます。

キュー・マネージャーが始動すると、ここで定義した API 出口により mqs.ini に定義された デフォルトが指定変更されます。

スタンザの属性

- API 出口を命名するには、以下の属性を使用します。

  Name=ApiExit_name
  MQAXP 構造体の ExitInfoName フィールドで、API 出口に渡される API 出口の記述名。

  この名前は一意で、48 文字以内であり、さらに IBM WebSphere MQ オブジェクト名 (例えばキュー名) に有効な文字のみを含む必要があります。

- 実行する API 出口コードのモジュールとエントリー・ポイントを識別するには、以下の属性を使用します。

  Function=function_name
  API 出口コードを含むモジュールへの関数エントリー・ポイントの名前。 このエントリー・ポイントは MQ_INIT_EXIT 関数です。

  このフィールドの長さは MQ_EXIT_NAME_LENGTH に限定されます。

  Module=module_name
  API 出口コードを含むモジュール。

  このフィールドにモジュールの絶対パス名が入っている場合、それがそのまま使用されます。

  このフィールドにモジュール名のみが含まれている場合、そのモジュールは qm.ini の `ExitPath` 内の `ExitsDefaultPath` 属性を使用して位置指定されます。

  異なるスレッド化ライブラリーをサポートするプラットフォームで、API 出口モジュールの非スレッド化バージョンとスレッド化バージョンの両方を提供する必要があります。 スレッド化バージョンは、接尾部 `_r` を持っている必要があります。 IBM WebSphere MQ アプリケーション・スタブのスレッド化バージョンは、指定のモジュールがロードされる前にその名前に `_r` を暗黙的に追加します。

  このフィールドの長さは、プラットフォームがサポートする最大パス長に限定されます。

- 必要により出口にデータを渡すには、以下の属性を使用します。

  Data=data_name
  MQAXP 構造体の ExitData フィールドで API 出口に渡されるデータ。

  この属性を指定すると、先行および末尾のブランクが除去されて残りのストリングは 32 文字に切り捨てられ、その結果が出口に渡されます。 この属性を省略すると、デフォルト値の 32 文字のブランクが出口に渡されます。

  このフィールドの最大長は、32 文字です。

- 他の出口に関するこの出口のシーケンスを識別するには、以下の属性を使用します。

Sequence=sequence_number
その他の API 出口に関してこの API 出口が呼び出される順序。 小さなシーケンス番号の出口は、より大きなシーケンス番号の出口よりも先に呼び出されます。 出口のシーケンス番号は連続である必要はありません。 つまり、1、2、3 の順序は、7、42、1096 の順序と同じ結果となります。 2 つの出口のシーケンス番号が同じ場合は、キュー・マネージャーが最初に呼び出す出口を決定します。 MQAXP の ExitChainAreaPtr で示される ExitChainArea に時刻またはマーカーを設定するかまたは独自のログ・ファイルを作成して、イベントの後に呼び出された出口を判別できます。

この属性は、符号なし数値です。

サンプル・スタンザ

サンプル mqs.ini ファイルには、次のスタンザが含まれています。

ApiExitTemplate
このスタンザは、記述名 OurPayrollQueueAuditor、モジュール名 auditor、およびシーケンス番号 2 を持つ出口を定義します。 123 のデータ値が出口に渡されます。

ApiExitCommon
このスタンザは、記述名 MQPoliceman、モジュール名 tmqp、およびシーケンス番号 1 を持つ出口を定義します。 渡されるデータは命令 (CheckEverything) です。

```
mqs.ini

   ApiExitTemplate:
      Name=OurPayrollQueueAuditor
      Sequence=2
      Function=EntryPoint
      Module=/usr/ABC/auditor
      Data=123
   ApiExitCommon:
      Name=MQPoliceman
      Sequence=1
      Function=EntryPoint
      Module=/usr/MQPolice/tmqp
      Data=CheckEverything
```

以下のサンプル qm.ini ファイルには、記述名 ClientApplicationAPIchecker、モジュール名 ClientAppChecker、および順序番号 3 が指定された出口の ApiExitLocal 定義が含まれています。

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

メッセージング・チャネルのためのチャネル出口プログラム

ここでの一連のトピックには、メッセージ・チャネル用の WebSphere MQ チャネル出口プログラムについての情報が含まれています。

メッセージ・チャネル・エージェント (MCA) でデータ変換出口を呼び出すこともできます。 データ変換出口の作成の詳細については、416 ページの『データ変換出口の作成』を参照してください。

この情報の一部は、WebSphere MQ MQI クライアントをキュー・マネージャーに接続する MQI チャネルの出口にも適用されます。 詳細については、MQI チャネル用のチャネル出口プログラムを参照してください。

チャネル出口プログラムは、MCA プログラムの実行中に定義された位置で呼び出されます。

一部のユーザー出口プログラムは、相互補完的に機能します。 例えば、伝送されるメッセージを暗号化するために、送信側 MCA によってあるユーザー出口プログラムが呼び出された場合、その逆の処理を行うために受信側で補完的な処理が実行される必要があります。

399 ページの表 55 には、チャネル・タイプごとに使用可能なチャネル出口のタイプを示しています。

表 55. チャネル・タイプで使用可能なチャネル出口

チャネル・タイプ	メッセージ出口	Message-retry exit (メッセージ再試行出口)	受信出口	セキュリティー出口	送信出口	Auto-definition exit (自動定義出口)
送信側チャネル	Yes		Yes	Yes	Yes	
サーバー・チャネル	Yes		Yes	Yes	Yes	
クラスター送信側チャネル	Yes		Yes	Yes	Yes	Yes
受信側チャネル	Yes	Yes	Yes	Yes	Yes	Yes
要求側チャネル	Yes	Yes	Yes	Yes	Yes	
クラスター受信側チャネル	Yes	Yes	Yes	Yes	Yes	Yes
クライアント接続チャネル			Yes	Yes	Yes	
サーバー接続チャネル			Yes	Yes	Yes	Yes

クライアント上でチャネル出口を実行する場合、MQSERVER 環境変数は使用できません。 代わりに、クライアント・チャネル定義テーブル の説明に従って、クライアント・チャネル定義テーブル (CCDT) を作成して参照してください。

処理の概要

MCA がチャネル出口プログラムを使用する方法の概要

始動すると MCA は、開始ダイアログを交換して処理を同期化させます。 次に、セキュリティー出口を含むデータ交換に切り替えます。 開始フェーズが完了してメッセージを転送できるようにするために、これらの出口は正常に終了する必要があります。

セキュリティー検査フェーズは、399 ページの図 77 に示すような処理ループです。

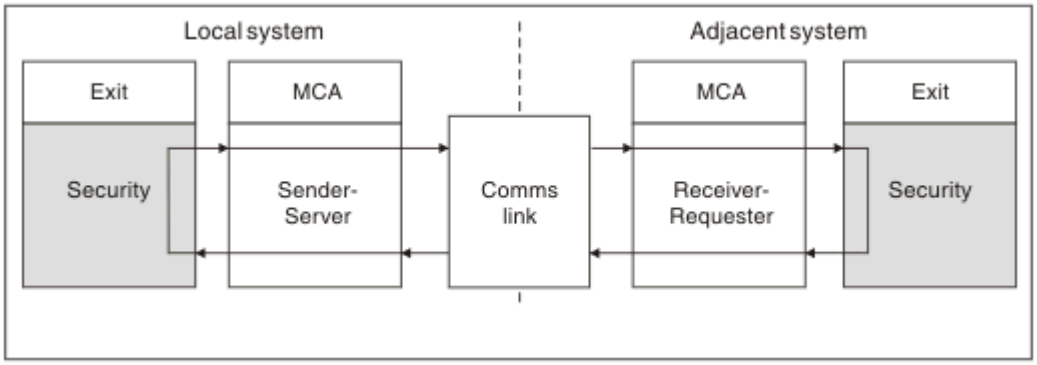


図 77. セキュリティー出口ループ

400 ページの図 78 に示すように、メッセージ転送フェーズ中に、送信側 MCA が伝送キューからメッセージを読み取り、メッセージ出口を呼び出し、次に送信出口を呼び出し、受信側 MCA にメッセージを送信します。

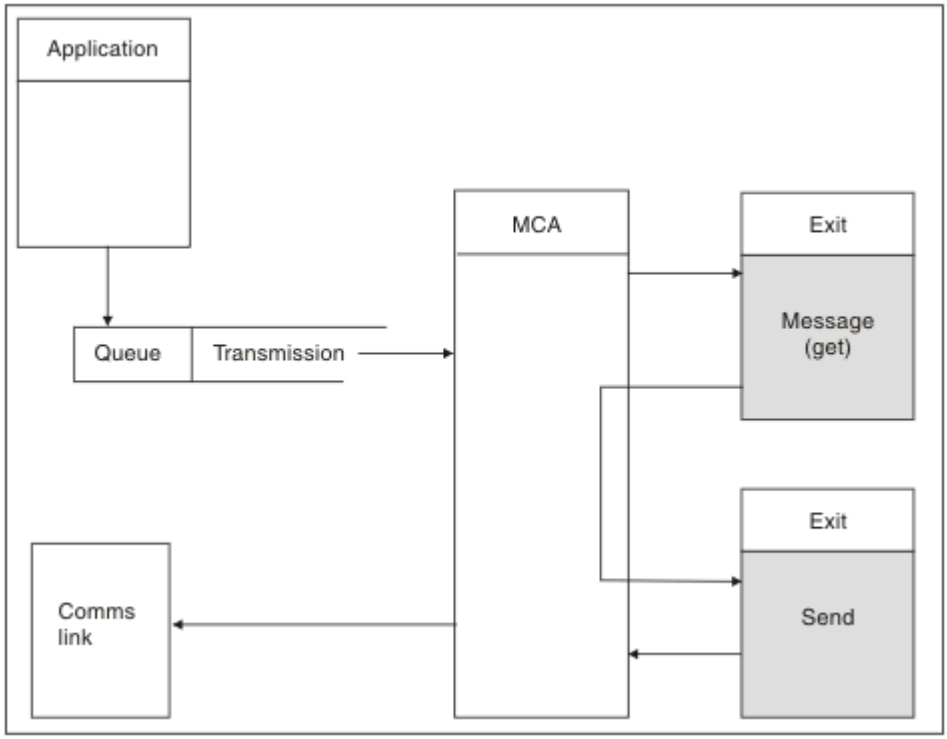


図 78. メッセージ・チャネルの送信側での送信出口の例

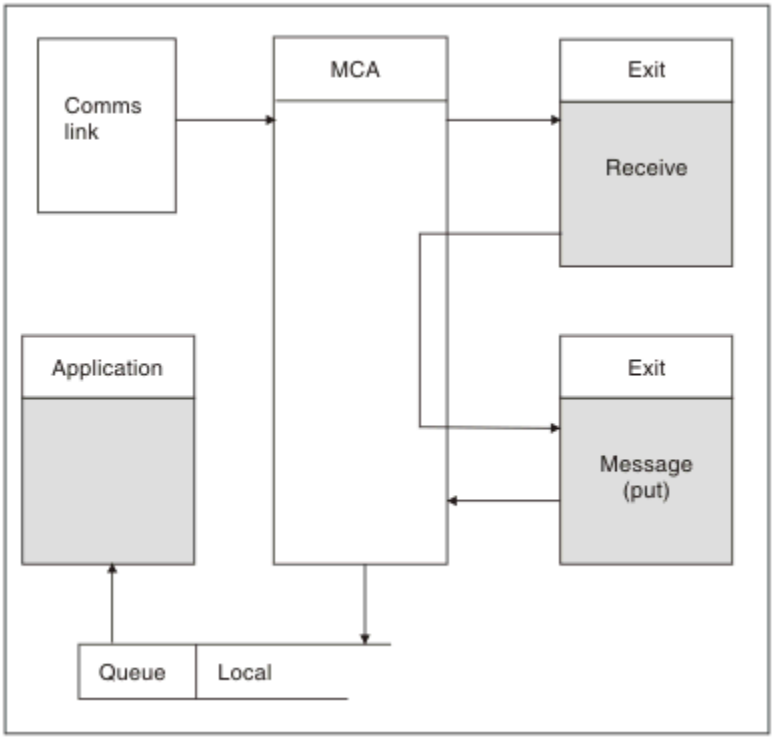


図 79. メッセージ・チャネルの受信側での受信出口の例

400 ページの図 79 に示すように、受信側 MCA は通信リンクからメッセージを受信し、受信出口を呼び出し、次にメッセージ出口を呼び出し、メッセージをローカル・キューに入れます (メッセージ出口が呼び出される前に、受信出口を複数回呼び出すことができます)。

チャネル出口プログラムの作成

チャネル出口プログラムの作成に役立つ情報を 以下に記載します。

ユーザー出口プログラムおよびチャネル出口プログラムでは、すべての MQI 呼び出し (ただし、この後のセクションで説明されているものを除く) を使用できます。 MQ V7 以降では、MQCXP 構造体バージョン 7 以降に接続ハンドル hConn が含まれているため、MQCONN を発行する代わりにこれを使用することができます。 これより前のバージョンで接続ハンドルを取得するには、MQCONN を発行する必要があります (チャネルそのものが既にキュー・マネージャーに接続されているため MQRC_ALREADY_CONNECTED 警告が返されても構いません)。

チャネル出口はスレッド・セーフでなければならないことに注意してください。

クライアント接続チャネル上の出口の場合、出口が接続しようとするキュー・マネージャーは、その出口がどのようにリンクされているかによって異なります。 出口が MQM.LIB にリンクされている場合に、MQCONN 呼び出しでキュー・マネージャー名を指定しないと、出口はそのシステム上のデフォルトのキュー・マネージャーに接続しようとします。 出口が MQM.LIB にリンクされていて、MQCD の QMgrName フィールドを介して出口に渡されたキュー・マネージャーの名前を指定すると、出口はそのキュー・マネージャーに接続しようとします。 出口が MQIC.LIB またはそれ以外のライブラリーにリンクされている場合は、キュー・マネージャー名を指定するかどうかに関係なく MQCONN 呼び出しは失敗します。

チャネル出口に渡された hConn に関連付けられているトランザクションの状態を変更することは 避ける必要があります。つまり、MQCMIT、 MQBACK、または MQDISC verb にチャネル hConn を指定して使用しないでください。また、 チャネル hConn を指定して MQBEGIN verb を使用することもできません。

MQCONNX に MQCNO_HANDLE_SHARE_BLOCK または MQCNO_HANDLE_SHARE_NO_BLOCK を指定して、 新規 IBM WebSphere MQ 接続を作成する場合は、 ユーザーの責任において、その接続が適切に管理され、キュー・マネージャーから正常に切断されるようにします。 例えば、チャネル出口が、呼び出される度に (切断することなく) キュー・マネージャーへの接続を新規作成すると、 接続ハンドルが蓄積され、エージェント・スレッドの数が増加することになります。

出口は MCA 自体と同じスレッドで実行され、同じ接続ハンドルを使用します。 つまり、出口は MCA と同じ UOW 内で実行され、同期点の下で行われた呼び出しは、すべてバッチの終わりにチャネルによってコミットまたはバックアウトされます。

したがって、チャネル・メッセージ出口は、元のメッセージが入っているバッチがコミットされたときにのみ そのキューにコミットされる通知メッセージを送信できます。 つまり、チャネル・メッセージ出口から同期点 MQI 呼び出しを発行することが可能です。

チャネル出口は、MQCD のフィールドを変更できます。 ただし、リストされている状況を除いて、これらの変更に応じて動作することはありません。 チャネル出口プログラムが MQCD データ構造体のフィールドを変更する場合、新規の値は、IBM WebSphere MQ チャネル・プロセスにより無視されます。 しかし新規の値は MQCD に残り、出口チェーンの残りの出口、およびチャネル・インスタンスを共用する会話に渡されます。 詳しくは、チャネル出口での MQCD フィールドの変更を参照してください。

また、C で作成したプログラムの場合、チャネル出口プログラムでは再入不能 C ライブラリー関数は使用しないでください。

複数のチャネル出口ライブラリーを同時に使用する場合、2 つの異なる出口のコードに同一名の関数が含まれていれば、一部の UNIX and Linux プラットフォームでは問題が発生する可能性があります。 チャネル出口がロードされる場合、動的ローダーは出口ライブラリーの関数名を、ライブラリーのロード先のアドレスに解決します。 2 つの出口ライブラリーが別個の関数を定義し、それらが偶然同一の名前になった場合、この解決プロセスでは、一方のライブラリーの関数名が他方の関数を使用するように誤って解決してしまう可能性があります。 この問題が起きる場合は、影響を受けないように、必要な出口と MQStart 関数だけしかエクスポートしないようにリンカーに指定してください。 他の関数には、 それ自身の出口ライブラリー以外の関数によって使用されないようにするために、ローカル可視性を付与する必要があります。 詳細については、リンカーに関する資料 を参照してください。

すべての出口は、チャネル出口パラメーター構造体 (MQCXP)、チャネル定義構造体 (MQCD)、準備済みのデータ・バッファー、データ長パラメーター、バッファー長パラメーターを指定して呼び出されます。 したがって、バッファー長を超過しないように注意してください。

- メッセージ出口の場合、チャネル間で送信されるのに必要な最大長のメッセージと MQXQH 構造体の長さが許可されます。
- 送信出口および受信出口の場合、許可される最大バッファーは次のとおりです。

  LU 6.2
  32 KB

  TCP:
  32 KB

  注 : 使用可能な最大の長さは、この長さよりも 2 バイト小さくなります。 詳細については、MaxSegmentLength で戻される値をチェックしてください。 MaxSegmentLength の詳細については、MaxSegmentLength を参照してください。

  NetBIOS:
  64 KB

  SPX:
  64 KB

  注 : 送信側チャネルの受信出口および受信側の送信出口は、TCP 用に 2 KB のバッファーを使用します。
- セキュリティー出口の場合、分散キューイング機構は、4000 バイトのバッファーを割り当てます。

出口から、関連パラメーターと共に代替バッファーを戻すことは可能です。 呼び出しの詳細については、398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

Windows、*UNIX and Linux* システムでのチャネル出口プログラムの作成

以下の情報は、Windows、UNIX and Linux システム用のチャネル出口プログラムを作成する際に使用することができます。

376 ページの『出口とインストール可能サービスの作成とコンパイル』に概説されている手順に従います。以下のチャネル出口固有の情報から、該当するものを使用してください。

出口は C で作成されなければならず、Windows の DLL です。

出口にダミー MQStart() ルーチンを定義して、ライブラリーのエントリー・ポイントとして MQStart を指定します。 402 ページの図 80 に、プログラムへの入り口のセットアップ方法を示しています。

```
#include <cmqec.h>

void MQStart() {;} /* dummy entry point - for consistency only */
void MQENTRY ChannelExit ( PMQCXP  pChannelExitParms,
                           PMQCD   pChannelDefinition,
                           PMQLONG pDataLength,
                           PMQLONG pAgentBufferLength,
                           PMQVOID pAgentBuffer,
                           PMQLONG pExitBufferLength,
                           PMQPTR  pExitBufferAddr)
{
... Insert code here
}
```

図 *80.* チャネル出口のサンプル・ソース・コード

Visual C++ を使用して Windows 用のチャネル出口を作成する場合は、ユーザー独自の DEF ファイルを作成する必要があります。 作成方法を 402 ページの図 81 に示します。 チャネル出口プログラムの作成について詳しくは、401 ページの『チャネル出口プログラムの作成』を参照してください。

```
EXPORTS
ChannelExit
```

図 *81. Windows* 用のサンプル *DEF* ファイル

チャネル・セキュリティー出口プログラム
セキュリティー出口プログラムを使用して、チャネルの反対側のパートナーが正しいかを確かめることができます。 これを認証と呼びます。 チャネルがセキュリティー出口を使用する必要があることを指定するには、チャネル定義の SCYEXIT フィールドに出口名を指定します。

注 : チャネル認証レコードを使用して認証を行うこともできます。 チャネル認証レコードの高い柔軟性により、 特定のユーザーおよびチャネルからのキュー・マネージャーへのアクセスを防止し、 リモート・ユーザーを IBM WebSphere MQ ユーザー ID にマップすることができます。 IBM WebSphere MQ では SSL および TLS もサポートしているため、 ユーザー認証や、使用データの暗号化およびデータ保全性検査を行うことができます。 SSL と TLS の詳細については、WebSphere MQ での SSL および TLS のサポートを参照してください。 これらよりもさらに高度な (または異なる) 形式のセキュリティー処理や、別のタイプの検査やセキュリティー・コンテキストの確立を必要とする場合は、 セキュリティー出口を作成することを検討してください。

IBM WebSphere MQ Version 7.1 より前に作成されたセキュリティー出口で注目に値する点として、IBM WebSphere MQ の旧バージョンは、基礎となるセキュア・ソケット・プロバイダー (例えば GSKit) を照会し、リモート・パートナーの証明書のサブジェクト識別名 (SSLPEER) および発行者識別名 (SSLCERTI) を判別していました。 IBM WebSphere MQ Version 7.1 では、一連の新規セキュリティー属性のサポートが追加されました。 これらの属性にアクセスするために IBM WebSphere MQ Version 7.1 は証明書の DER エンコードを取得し、サブジェクト DN および発行者 DN を判別するためにそれを使用します。 サブジェクト DN 属性および発行者 DN 属性は、次のチャネル状況属性に表示されます。

- SSLPEER (PCF セレクター MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (PCF セレクター MQCACH_SSL_CERT_ISSUER_NAME)

これらの値はチャネル状況コマンドによって返されるだけでなく、以下にリストするチャネル・セキュリティー出口にもデータが渡されます。

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

IBM WebSphere MQ Version 7.1 では、サブジェクト DN に SERIALNUMBER 属性も含まれ、これにはリモート・パートナーの証明書のシリアル番号が含まれます。 さらに一部の DN 属性は、以前のリリースとは異なる順序で返されます。 その結果、Version 7.1 では、SSLPEER フィールドと SSLCERTI フィールドの構成が以前のリリースとは異なるものに変更されています。そのため、これらのフィールドに依存するセキュリティー出口またはアプリケーションを検査し、更新することを推奨します。

チャネル定義の SSLPEER フィールドを経由して指定された既存の WebSphere MQ ピア名フィルターは影響を受けることなく、引き続き以前のリリースと同様に作動します。 その理由は、WebSphere MQ ピア名マッチング・アルゴリズムが、チャネル定義を変更することなく既存の SSLPEER フィルターを処理するよう更新されたことにあります。 この変更の影響を受ける可能性が最も高いのは、PCF プログラミング・インターフェースによって返されるサブジェクト DN および発行者 DN 値に依存するセキュリティー出口とアプリケーションです。

セキュリティー出口は C または Java で作成できます。

チャネル・セキュリティー出口プログラムは、 MCA 処理サイクル内の以下の箇所で呼び出されます。

- MCA 開始および終了時
- チャネルの開始時に初期のデータ折衝が終了した直後。 チャネルの受信側またはサーバーは、リモート側のセキュリティー出口へ送達されるメッセージを提供することによって、リモート側とのセキュリティー・メッセージ交換を開始する場合があります。 また、このメッセージ交換を拒否することもあります。リモート側から受信したセキュリティー・メッセージを処理するために、出口プログラムが再度呼び出されます。
- チャネルの開始時に初期のデータ折衝が終了した直後。 チャネルの送信側または要求側は、リモート側から送信されてきたセキュリティー・メッセージを処理するか、リモート側がセキュリティー交換を開始できない場合にそれを開始します。 その後受信する可能性のあるすべてのセキュリティー・メッセージを処理するために、出口プログラムが 再度呼び出されます。

要求側チャネルが MQXR_INIT_SEC で呼び出されることはありません。 チャネルは、セキュリティー出口プログラムがあることをサーバーに通知します。そこでサーバーはセキュリティー出口を開始する機会を

得ます。 セキュリティー出口がない場合、サーバーは要求側に通知し、長さゼロのフローが出口プログラムに返されます。

注 : ゼロ長のセキュリティー・メッセージを送信することは避けてください。

セキュリティー出口プログラムによって交換されたデータの例を、図 404 ページの図 82 から 406 ページの図 85 に示します。 これらの例は、 受信側のセキュリティー出口と送信側のセキュリティー出口が関連して発生するイベントの順序を示しています。 図の中で行が連続することは、時間の経過を表しています。 場合によっては、受信側と送信側のイベントが相関されないために、イベントが同時に発生したりあるいは別々のときに発生することもあります。 また、別の場合には、ある出口プログラムのイベントが原因となって、他の出口プログラムで補足的なイベントが後から発生することもあります。 例えば、404 ページの図 82 では次のようになっています。

1. 受信側と送信側がそれぞれ MQXR_INIT によって呼び出されたものの、それらの呼び出しが相関されなかったために、それらの呼び出しが同時に行われるか別々に行われるかわかりません。
2. 次に、受信側が MQXR_INIT_SEC によって呼び出されましたが、送信側出口で補足的なイベントを必要としない MQXCC_OK が戻されました。
3. 次に、送信側が MQXR_INIT_SEC によって呼び出されました。 この呼び出しは、MQXR_INIT_SEC によって呼び出された受信側の呼び出しとは相関されません。 送信側は MQXCC_SEND_SEC_MSG を戻し、受信側出口で補足的なイベントが発生します。
4. その後、受信側が MQXR_SEC_MSG によって呼び出され、MQXCC_SEND_SEC_MSG を戻すと、 送信側出口で補足的なイベントが発生します。
5. その後、送信側が MQXR_SEC_MSG によって呼び出され、受信側出口での補足的なイベントを必要としない MQXCC_OK を戻します。

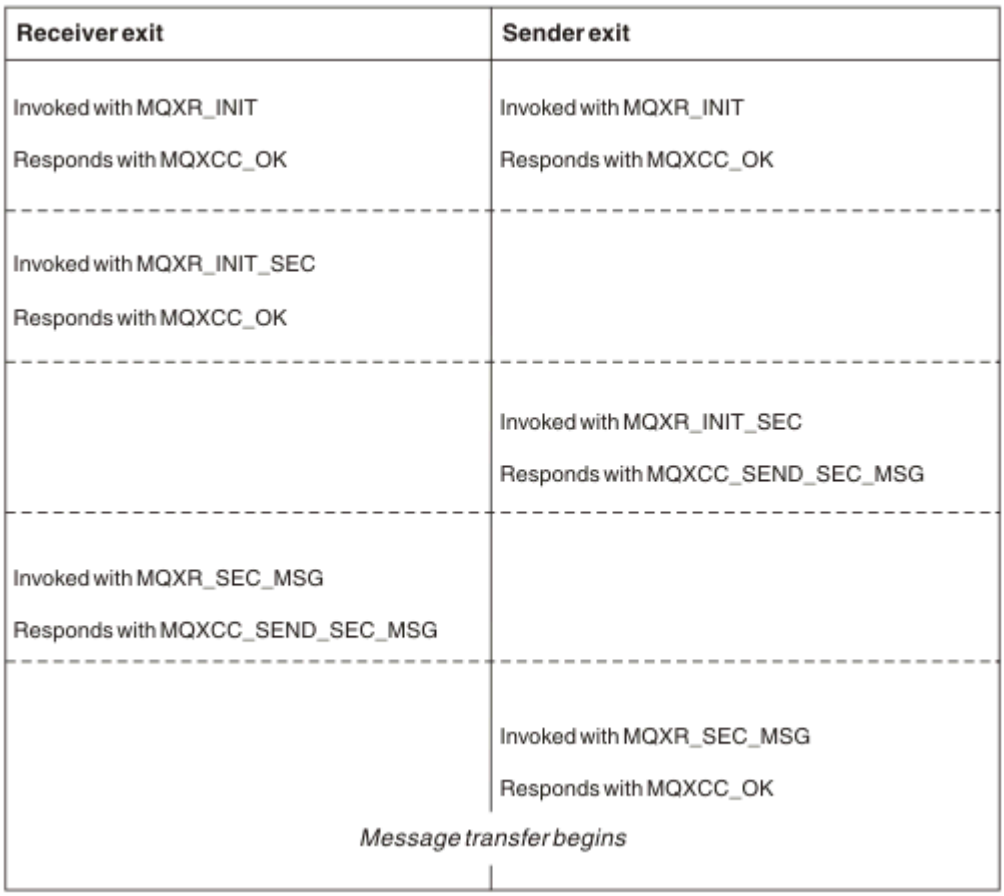


図 *82.* 合意に基づいて送信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

図 *83.* 合意なしで送信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

図 *84.* 合意に基づいて受信側から開始される交換

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

図 *85.* 合意なしで受信側から開始される交換

チャネル・セキュリティー出口には、伝送ヘッダーを除き、セキュリティー出口プログラムが作成したセキュリティー・データが入っているエージェント・バッファーが渡されます。 このデータは、どちらかのチャネルでセキュリティー検査を行えるような、任意の適切なデータになります。

メッセージ・チャネルの送信側および受信側の両方のセキュリティー出口プログラムは、どの呼び出しにも次の 2 つの応答コードのうちいずれかを返すことができます。

- セキュリティー交換はエラーなしで終了
- チャネルを非表示にして、閉じる

注 :

1. チャネル・セキュリティー出口は、通常は対で作動します。 該当するチャネルを定義するときには必ず、チャネルの両側で互換性のある出口プログラムを指定するようにしてください。
2. IBM i では、 「Use adopted authority (借用権限を使用)」(USEADPAUT=*YES) でコンパイルされたセキュリティー出口プログラムは、QMQM または QMQMADM 権限を借用できます。 この機能を出口が使用することによって、システムにセキュリティー上のリスクがもたらされることのないように注意してください。
3. チャネルの反対側で証明書が提供される SSL チャネルでは、セキュリティー出口は、SSLPeerNamePtr がアクセスする MQCD フィールドでこの証明書の所有者の識別名を受け取り、SSLRemCertIssNamePtr がアクセスする MQCXP フィールドで発行者の識別名を受け取ります。 この名前を保管する目的は以下のとおりです。
   - SSL チャネル間のアクセスを制限すること
   - この名前に基づいて MQCD.MCAUserIdentifier を変更すること

関連概念

チャネル認証レコード

Secure Sockets Layer (SSL) および Transport Layer Security (TLS) の概念

セキュリティー出口の作成

セキュリティー出口のスケルトン・コードを使用して、セキュリティー出口を作成できます。

407 ページの図 86 は、セキュリティー出口の作成方法を示しています。

```
void MQENTRY MQStart() {;}
void MQENTRY EntryPoint (PMQVOID pChannelExitParms,
                         PMQVOID pChannelDefinition,
                         PMQLONG pDataLength,
                         PMQLONG pAgentBufferLength,
                         PMQVOID pAgentBuffer,
                         PMQLONG pExitBufferLength,
                         PMQPTR  pExitBufferAddr)
{
  PMQCXP pParms = (PMQCXP)pChannelExitParms;
  PMQCD  pChDef = (PMQCD)pChannelDefinition;
  /* TODO: Add Security Exit Code Here */

}
```

図 *86.* セキュリティー出口のスケルトン・コード

標準 WebSphere MQ 入り口点 MQStart が存在しなければなりませんが、関数の実行には必要とされません。 関数の名前 (この例では EntryPoint) は変更可能ですが、ライブラリーのコンパイルおよびリンク時にはその関数をエクスポートする必要があります。 上記の例のように、ポインター pChannelExitParms は PMQCXP にキャストする必要があり、pChannelDefinition は PMQCD にキャストする必要があります。 チャネル出口の呼び出しおよびパラメーターの使用に関する一般情報については、MQ_CHANNEL_EXIT を参照してください。 これらのパラメーターは、セキュリティー出口で以下のように使用されます。

PMQVOID pChannelExitParms
入出力

MQCXP 構造体へのポインター - フィールドにアクセスするために PMQCXP にキャストします。 この構造体は、出口と MCA との間の通信に使用されます。 MQCXP の以下のフィールドは、セキュリティー出口には特に重要です。

ExitReason
セキュリティー出口に、セキュリティー交換における現行状態を通知する。 取るべき処置を決定する際に使用される。

ExitResponse
セキュリティー交換の次のステージを指示する、MCA への応答。

ExitResponse2
MCA がセキュリティー出口の応答を解釈する方法を制御する、追加の制御フラグ。

ExitUserArea
呼び出しから次の呼び出しまでの間、状態を維持するためにセキュリティー出口が使用できる、16 バイト (最大) のストレージ。

ExitData
チャネル定義の SCYDATA フィールドで指定されるデータを含みます (32 バイト。右側にブランクで埋め込み)。

PMQVOID pChannelDefinition
入出力

MQCD 構造体へのポインター - フィールドにアクセスするために PMQCD にキャストします。 このパラメーターにはチャネルの定義が含まれます。 MQCD の以下のフィールドは、セキュリティー出口には特に重要です。

ChannelName
チャネル名 (20 バイト。右側にブランクで埋め込み)。

ChannelType
チャネル・タイプを定義するコード。

MCA ユーザー ID
このグループの 3 つのフィールドは、チャネル定義で指定された MCAUSER フィールドの値に初期化されます。 セキュリティー出口により指定された、これらのフィールドのユーザー ID はすべて、アクセス制御に使用されます (SDR、SVR、CLNTCONN、および CLUSSDR チャネルには適用されない)。

MCAUserIdentifier
ID の最初の 12 バイト (右側にブランクで埋め込み)。

LongMCAUserIdPtr
フルサイズの ID (必ずしも NULL 終了ではない) が含まれるバッファーへのポインターは、MCAUserIdentifier に優先します。

LongMCAUserIdLength
LongMCAUserIdPtr により指されるストリングの長さ - LongMCAUserIdPtr が設定されている場合に設定する必要があります。

リモート・ユーザー ID
CLNTCONN/SVRCONN のチャネルのペアにのみ適用されます。 CLNTCONN セキュリティー出口が定義されていない場合、これら 3 つのフィールドはクライアント MCA により初期化されます。そのため、認証用および MCA ユーザー ID の指定時に SVRCONN セキュリティー出口が使用できる、クライアントの環境からのユーザー ID が含まれる場合があります。 CLNTCONN セキュリティー出口が定義されている場合、これらのフィールドは初期化されず、CLNTCONN セキュリティー出口により設定できます。またはセキュリティー・メッセージをクライアントからサーバーへのユーザー ID の受け渡しに使用できます。

RemoteUserIdentifier
ID の最初の 12 バイト (右側にブランクで埋め込み)。

LongRemoteUserIdPtr
フルサイズの ID (必ずしも NULL 終了ではない) が含まれるバッファーへのポインターは、RemoteUserIdentifier に優先します。

LongRemoteUserIdLength
LongRemoteUserIdPtr により指されるストリングの長さ - LongRemoteUserIdPtr が設定されている場合に設定する必要があります。

PMQLONG pDataLength
入出力

MQLONG へのポインター。 セキュリティー出口の呼び出し時に、AgentBuffer に含まれるセキュリティー出口の長さが入ります。 セキュリティー出口により、AgentBuffer または ExitBuffer で送信されるメッセージの長さに設定する必要があります。

PMQLONG pAgentBufferLength
入出力

MQLONG へのポインター。 セキュリティー出口の呼び出し時に AgentBuffer に含まれるデータの長さ。

PMQVOID pAgentBuffer
入出力

セキュリティー出口の呼び出し時に、これはパートナー出口から送信されるメッセージを指します。 MQCXP 構造体内の ExitResponse2 に MQXR2_USE_AGENT_BUFFER フラグが設定されている場合 (デフォルト)、セキュリティー出口ではこのパラメーターを、送信されるメッセージ・データを指すように設定する必要があります。

PMQLONG pExitBufferLength
入出力

MQLONG へのポインター。 このパラメーターはセキュリティー出口の最初の呼び出し時に 0 に初期化され、戻される値は、セキュリティー交換中のセキュリティー出口への呼び出しから呼び出しまでの間、維持されます。

PMQPTR pExitBufferAddr
入出力

このパラメーターはセキュリティー出口の最初の呼び出し時に NULL ポインターに初期化され、戻される値は、セキュリティー交換中のセキュリティー出口への呼び出しから呼び出しまでの間、維持されます。 MQXR2_USE_EXIT_BUFFER フラグが MQCXP 構造体内の ExitResponse2 に設定されている場合、セキュリティー出口ではこのパラメーターを、送信されるメッセージ・データを指すように設定する必要があります。

CLNTCONN/SVRCONN チャネルのペアおよび他のチャネルのペアで定義されたセキュリティー出口における動作の相違点
セキュリティー出口は、すべてのタイプのチャネルで定義できます。 ただし、CLNTCONN/SVRCONN チャネルのペアで定義されたセキュリティー出口の動作は、他のチャネルのペアで定義されたセキュリティー出口と少し異なります。

CLNTCONN チャネルのセキュリティー出口は、パートナー SVRCONN 出口による処理のために、または SVRCONN セキュリティー出口が定義されておらず SVRCONN の MCAUSER フィールドが設定されていない場合の OAM 許可のために、チャネル定義でリモート・ユーザー ID を設定できます。

CLNTCONN セキュリティー出口が定義されていない場合、チャネル定義のリモート・ユーザー ID は、クライアント MCA によってクライアント環境のユーザー ID に設定されます (ブランクの場合もあります)。

CLNTCONN と SVRCONN のチャネルのペアで定義されたセキュリティー出口間でのセキュリティー交換は、SVRCONN セキュリティー出口が、MQXCC_OK の ExitResponse を戻す場合に正常に完了します。 他のチャネルのペア間でのセキュリティー交換は、交換を開始したセキュリティー出口が、MQXCC_OK の ExitResponse を戻す場合に正常に完了します。

ただし、MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse コードを使用して、セキュリティー交換の継続を強制することができます。MQXCC_SEND_AND_REQUEST_SEC_MSG の ExitResponse が CLNTCONN または SVRCONN セキュリティー出口によって戻された場合、パートナー出口はセキュリティー・メッセージ (MQXCC_OK やヌル応答ではない) を送信することによって応答する必要があり、そうしない場合はチャネルは終了します。 他のタイプのチャネルで定義されたセキュリティー出口の場合、MQXCC_SEND_AND_REQUEST_SEC_MSG への応答としてパートナー・セキュリティー出口から

MQXCC_OK の ExitResponse が戻されると、ヌル応答が戻されたようにセキュリティー交換は継続し、チャネルは終了しません。

SSPI セキュリティー出口
WebSphere MQ for Windows は、セキュリティー・サービス・プログラミング・インターフェース (SSPI) を使用して WebSphere MQ チャネルの認証を行う、セキュリティー出口を提供します。 SSPI には、Windows の統合セキュリティー機能が備えられています。

このセキュリティー出口は、WebSphere MQ クライアントおよび WebSphere MQ サーバーの両方で使用されます。

セキュリティー・パッケージは、security.dll または secur32.dll のいずれかからロードします。 これらの DLL は、ご使用のオペレーティング・システム (OS) で提供されています。

片方向認証は Windows で提供されているもので、NTLM 認証サービスを使用します。 双方向認証は Windows 2000 で提供され、Kerberos 認証サービスを使用します。

セキュリティー出口プログラムは、ソースおよびオブジェクト形式で提供されます。 オブジェクト・コードをそのまま使用することもできますし、あるいはソース・コードを開始点として使用して独自のユーザー出口プログラムを作成することもできます。 SSPI セキュリティー出口のオブジェクトまたはソース・コードの使用について詳しくは、168 ページの『Windows システムでの SSPI セキュリティー出口の使用』を参照してください。

チャネル送信および受信出口プログラム
送信出口と受信出口を使用して、データの圧縮や圧縮解除などの作業を実行することができます。 連続して実行する送信および受信出口プログラムのリストを指定できます。

チャネル送信出口プログラムと受信出口プログラムは、MCA の処理サイクルの次の場所で呼び出されます。

- 送信出口プログラムと受信出口プログラムは、MCA の開始時に初期設定のために呼び出され、MCA の終了時には終了のために呼び出されます。
- 送信出口プログラムは、伝送がリンクを介して送信される直前に、チャネルの一方または他方の側 (1 つのメッセージ転送の伝送の送信元となる側) で呼び出されます。 メッセージ・チャネルでは一方向にのみメッセージが送信されるにもかかわらず、両方向で出口を使用できる理由についての説明は、注 4 を参照してください。
- 受信出口プログラムは、伝送がリンクから取り出された直後に、チャネルの一方または他方の側 (1 つのメッセージ転送の伝送が受信された側) で呼び出されます。 メッセージ・チャネルでは一方向にのみメッセージが送信されるにもかかわらず、両方向で出口を使用できる理由についての説明は、注 4 を参照してください。

1 つのメッセージ転送に対して多くの伝送がある場合があり、メッセージが受信側のメッセージ出口に到達する前に送信出口プログラムと受信出口プログラムの多くの反復がある可能性があります。

チャネル送信出口および受信出口プログラムには、通信リンクから送信されたり受信されたりした伝送データが入っているエージェント・バッファーが渡されます。 送信出口プログラムの場合、バッファーの最初の 8 バイトは MCA による使用のために予約されており、変更できません。 プログラムが別のバッファーを返す場合、新規バッファーにはこれらの最初の 8 バイトが存在する必要があります。 これらの出口プログラムに提示されるデータの形式は定義されていません。

適切な応答コードが送信および受信の出口プログラムによって戻される必要があります。 それ以外の応答は、MCA 異常終了 (アベンド) の原因になります。

注 : 送信出口または受信出口から同期点の内部で MQGET、MQPUT、または MQPUT1 呼び出しを発行しないでください。

注 :

1. 通常、送信出口と受信出口は、ペアで使用します。 例えば、送信出口がデータを圧縮して受信出口がデータを圧縮解除したり、送信出口がデータを暗号化して受信出口がデータを暗号化解除したりすることがあります。 該当するチャネルを定義するときには必ず、チャネルの両側で互換性のある出口プログラムを指定するようにしてください。
2. チャネルの圧縮がオンの場合、出口には圧縮データが渡されます。

3. チャネル送信出口および受信出口は、状況メッセージなどのアプリケーション・データ以外のメッセージ・セグメントに対して呼び出されることがあります。 開始ダイアログ中またはセキュリティー検査フェーズで呼び出されることはありません。
4. メッセージ・チャネルは一方向にのみメッセージを送信しますが、チャネル制御データ (ハートビートやバッチ処理の終了など) は両方向に流れるため、これらの出口も両方向で利用可能です。 ただし、初期チャネル開始データ・フローのなかには、任意の出口による処理対象から除外されるものもあります。
5. 送信出口と受信出口が、順序を無視して呼び出される場合があります。例えば、一連の出口プログラムの実行中や、出口プログラムの他にセキュリティー出口も実行している場合です。 したがって、データを処理するために受信出口が最初に呼び出されたとき、対応する送信出口を通過して渡されていないデータを受け取る可能性があります。 受信出口が、操作が必要であることを最初にチェックせずに圧縮解除などの操作をただ実行した場合、予期しない結果になることがあります。

   送信出口および受信出口をコーディングする場合は、受信中のデータが対応する送信出口によって処理済みであるということを受信出口が検査できるような方法で行う必要があります。 推奨される実行方法は、出口プログラムを以下のようにコーディングすることです。

   - 送信出口はデータの 9 番目のバイトの値を 0 に設定し、すべてのデータを 1 バイトだけシフトさせた後で操作を実行します (最初の 8 バイトは MCA で使用するために予約されています)。
   - 受信出口は、バイト 9 の値が 0 のデータを受信すると、そのデータを送信出口から送られてきたデータであると認識します。 0 を削除し、補数演算を実行し、結果データを 1 バイトだけシフトして戻します。
   - 受信出口は、バイト 9 が 0 以外のデータを受信すると、送信出口は稼働していないと見なし、データを変更せずに呼び出し側に送り返す。

   セキュリティー出口の使用中、セキュリティー出口によってチャネルが終了した場合、対応する受信出口が存在しないのに送信出口が呼び出される可能性があります。 この問題を回避する 1 つの方法は、出口がチャネルを終了するときに、MQCD.SecurityUserData または MQCD.SendUserData などのフラグを設定するようにセキュリティー出口をコーディングすることです。 次に、送信出口はこのフィールドを検査し、フラグが設定されていない場合のみデータを処理する必要があります。 このチェックによって送信出口によるデータの不要な変更が回避され、セキュリティー出口が変更済みデータを受信した場合に発生する可能性がある変換エラーを回避することができます。

チャネル送信出口プログラム - スペースの予約
送信および受信出口を使用して、データを変換してから伝送することができます。 チャネル送信出口プログラムでは、変換バッファー内でスペースを予約することにより、変換に関する独自のデータを追加できます。

このデータは受信出口プログラムによって処理されてから、バッファーから除去されます。 例えば、データを暗号化し、暗号化解除のセキュリティー・キーを追加することもできます。

スペースの予約および使用の方法

初期設定で送信出口プログラムが呼び出されたときに、MQXCP の *ExitSpace* フィールドを、予約するバイト数に設定します。 詳しくは、MQCXP を参照してください。 *ExitSpace* を設定できるのは、初期設定の間、つまり *ExitReason* の値が MQXR_INIT であるときだけです。 *ExitReason* が MQXR_XMIT に設定され、伝送の直前に送信出口が呼び出されると、伝送バッファー内で *ExitSpace* バイトが予約されます。 *ExitSpace* は、z/OS ではサポートされていません。

送信出口は、予約されたすべてのスペースを使用する必要はありません。 *ExitSpace* のバイト数まで使用でき、伝送バッファーがいっぱいでなければ、出口は予約されている量を超えて使用できます。
ExitSpace の値の設定時には、伝送バッファー内にメッセージ・データ用として最低でも 1 KB は残しておかなければなりません。 予約されたスペースが大量のデータ用に使用される場合、チャネル・パフォーマンスに影響することがあります。

チャネルの受信側で行われる事柄

チャネル受信出口プログラムは、対応する送信出口と互換性と持つようにセットアップしなければなりません。 受信出口は予約されたスペース内のバイト数を認識していなければならず、そのスペース内のデータを除去しなければなりません。

複数の送信出口

連続して実行する送信および受信出口プログラムのリストを指定できます。 WebSphere MQ は、すべての送信出口によって予約されたスペースの合計を保持します。 このスペースの合計では、伝送バッファー内にメッセージ・データ用として最低でも 1 KB は残しておかなければなりません。

以下の例では、連続して呼び出される 3 つの送信出口に、スペースが割り振られる方法が示されています。

1. 初期設定時の呼び出しで、以下のことが行われます。
   - 送信出口 A で 1 KB が予約されます。
   - 送信出口 B で 2 KB が予約されます。
   - 送信出口 C で 3 KB が予約されます。
2. 最大伝送サイズは 32 KB で、ユーザー・データの長さは 5 KB です。
3. 出口 A は 5 KB のデータで呼び出されます。出口 B および C に 5 KB が予約されているため、27 KB までが使用可能です。 出口 A は、それが予約した量である 1 KB を追加します。
4. 出口 B は 6 KB のデータで呼び出されます。出口 C に 3 KB が予約されているため、29 KB までが使用可能です。 出口 B は、それが予約した 2 KB よりも少ない 1 KB を追加します。
5. 出口 C が 7 KB のデータで呼び出されます。最大で 32 KB まで使用可能です。 出口 C は、それが予約した 3 KB より大きい 10 KB を追加します。 データの合計 17 KB は最大値の 32 KB より小さいため、この量は有効です。

チャネル・メッセージ出口プログラム
チャネル・メッセージ出口を使用して、リンクの暗号化、着信ユーザー ID の妥当性検査または置換、メッセージ・データの変換、ジャーナル処理、および参照メッセージ処理などのタスクを実行できます。 連続して実行するメッセージ出口プログラムのリストを指定できます。

チャネル・メッセージ出口プログラムは、MCA の処理サイクルの次の場所で呼び出されます。

- MCA 開始および終了時
- 送信側 MCA が MQGET 呼び出しを発行した直後
- 受信側 MCA が MQPUT 呼び出しを発行する前

メッセージ出口には、伝送キュー・ヘッダー MQXQH およびキューから検索されたアプリケーション・メッセージ・テキストが入っているエージェント・バッファーが渡されます (MQXQH の形式は、MQXQH にあります)。 参照メッセージ (送信される他のオブジェクトを指すヘッダーのみを含むメッセージ) を使用する場合、メッセージ出口はヘッダー MQRMH を認識します。 このヘッダーは、オブジェクトを識別し、該当する方法でオブジェクトを検索し、それをヘッダーに追加します。そして、そのヘッダーを受信側 MCA へ伝送するために MCA に渡します。 受信側 MCA にある別のメッセージ出口は、このメッセージが参照メッセージであることを認識し、オブジェクトを抽出し、ヘッダーを宛先キューに渡します。 参照メッセージと、参照メッセージを処理するメッセージ出口のサンプルについては、265 ページの『参照メッセージ』および 140 ページの『参照メッセージ・サンプルの実行』を参照してください。

メッセージ出口は、次のような応答を戻します。

- メッセージを送信する (GET 出口)。 メッセージは出口によって変更された可能性があります (この場合は MQXCC_OK を戻します)。
- メッセージをキューに書き込む (PUT 出口)。 メッセージは出口によって変更された可能性があります (この場合は MQXCC_OK を戻します)。
- メッセージを処理しない。 メッセージは、MCA によって送達不能キュー (未配布メッセージ・キュー) に入ります。

- チャネルをクローズする。
- 不良戻りコードが戻される。この場合には MCA が異常終了します。

注 :

1. メッセージが部分に分割された場合であっても、メッセージ出口は、転送される完全な各メッセージにつき 1 回呼び出されます。
2. UNIX システムでは、何らかの理由でメッセージ出口を指定した場合、ユーザー ID を自動的に小文字に変換する機能は働きません。 UNIX and Linux システムでのオブジェクトのセキュリティーを参照してください。
3. 出口は MCA 自体と同じスレッドで実行されます。 また、同じ接続ハンドルを使用するため、MCA と同じ作業単位 (UOW) 内で実行されます。 したがって、同期点の下で行われた呼び出しは、すべてバッチの終わりにチャネルによってコミットまたはバックアウトされます。 例えば、あるチャネル・メッセージ出口プログラムは別のチャネル・メッセージ出口プログラムに通知メッセージを送信でき、これらのメッセージは、元のメッセージを含んでいるバッチがコミットされたときに限りキューにコミットされます。

   したがって、チャネル・メッセージ出口プログラムから同期点 MQI 呼び出しを発行することが可能です。

メッセージ出口の外側でのメッセージ変換
メッセージ出口を呼び出す前に、受信側 MCA はメッセージの変換を実行します。 このトピックでは、変換を実行するために使用されるアルゴリズムについて説明します。

処理されるヘッダー

変換ルーチンは、メッセージ出口が呼び出される前に受信側の MCA 内で実行します。 変換ルーチンはメッセージの冒頭にある MQXQH ヘッダーから始まります。 その後、変換ルーチンは、MQXQH に続くチェーン・ヘッダーを処理し、必要に応じて変換を実行します。 チェーン・ヘッダーは、受信側のメッセージ出口に渡された MQCXP データの HeaderLength パラメーターに含まれるオフセットを超えて拡張できます。 以下のヘッダーは同じ場所で変換されます。

- MQXQH (形式名 "MQXMIT ")
- MQMD (このヘッダーは MQXQH の一部で形式名なし)
- MQMDE (形式名 "MQHMDE ")
- MQDH (形式名 "MQHDIST ")
- MQWIH (形式名 "MQHWIH ")

以下のヘッダーは変換されませんが、MCA がチェーン・ヘッダーの処理を続行するときにステップオーバーされます。

- MQDLH (形式名 "MQDEAD ")
- 3 文字の「MQH」で始まる形式名を持つすべてのヘッダー (例えば、「MQHRF 」) 特に言及されていない

ヘッダーの処理方法

各 WebSphere MQ ヘッダーの Format パラメーターは MCA によって読み取られます。 Format パラメーターはヘッダー内の 8 バイトで、名前を含む 8 つの 1 バイト文字です。

その後、MCA は各ヘッダーに続くデータを名前付きタイプとして解釈します。 Format が、WebSphere MQ データ変換に適格なヘッダー・タイプの名前であれば、データは変換されます。 これが非 MQ データを表す別の名前である場合 (例えば MQFMT_NONE または MQFMT_STRING)、MCA はヘッダーの処理を停止します。

MQCXP HeaderLength について

メッセージ出口に提供される MQCXP データ内の HeaderLength パラメーターは、メッセージの最初にある MQXQH (MQMD を含む)、MQMDE、および MQDH ヘッダーの合計長です。 これらのヘッダーは、「Format」の名前と長さを使用してチェーニングされます。

MQWIH

チェーニングされたヘッダーは、HeaderLength を超えてユーザー・データ域まで拡張できます。 MQWIH ヘッダー (ある場合) は、HeaderLength を超えて表示されるそのようなヘッダーの 1 つです。

チェーン・ヘッダー内に MQWIH ヘッダーがある場合、受信側のメッセージ出口が呼び出される前に、このヘッダーは同じ場所で変換されます。

チャネル・メッセージ再試行出口プログラム

チャネル・メッセージ再試行出口は、ターゲット・キューのオープンに失敗したときに呼び出されます。 この出口を使用して、再試行を行う環境、再試行の回数、再試行の頻度を決めることができます。

この出口は、MCA 開始および終了時にチャネルの受信側でも呼び出されます。

チャネル・メッセージ再試行出口には、伝送キュー・ヘッダー MQXQH と、キューから検索されたアプリケーション・メッセージ・テキストが入っているエージェント・バッファーが渡されます。 MQXQH の形式は、MQXQH の概要に示されています。

この出口はすべての理由コードについて呼び出され、MCA が再試行する必要がある理由コードと、再試行の回数および間隔を判別します (チャネルが定義されたときに設定されたメッセージ再試行カウントの値が MQCD の出口に渡されますが、出口はこの値を無視することができます)。

MQCXP の MsgRetryCount フィールドの値は、出口が呼び出されるたびに MCA によって増分され、出口は MQCXP の MsgRetryInterval フィールドに入っている待ち時間を示した MQXCC_OK、あるいは MQXCC_SUPPRESS_FUNCTION のいずれかを戻します。 再試行は、出口が MQCXP の ExitResponse フィールドに MQXCC_SUPPRESS_FUNCTION を戻すまで無限に続きます。 これらの完了コードに対して MCA が行う処置については、MQCXP を参照してください。

すべての再試行が失敗した場合は、メッセージは送達不能キューに書き込まれます。 使用可能な送達不能キューが 1 つもない場合、チャネルは停止します。

チャネル用にメッセージ再試行出口を定義しないときに、MQRC_Q_FULL などの一時的なものと思われる障害が起こると、MCA は、チャネルが定義されたときに設定されたメッセージ再試行カウントとメッセージ再試行間隔を使用します。 障害が永続的なもので、障害を処理する出口プログラムを定義していない場合、メッセージは送達不能キューに書き込まれます。

チャネル自動定義出口プログラム

受信側チャネルまたはサーバー接続チャネルを開始する要求を受信してもそのチャネル定義が存在しない場合、チャネル自動定義出口を使用することができます (WebSphere MQ for z/OS の場合を除く)。 また、いずれのプラットフォームにおいても、この出口を呼び出すことにより、クラスター送信側チャネルとクラスター受信側チャネルの両方で、チャネルのインスタンスの定義変更ができるようになります。

z/OS 以外のすべてのプラットフォームでは、受信側チャネルまたはサーバー接続チャネルを開始する要求が受信されてチャネル定義が存在しない場合、チャネル自動定義出口を呼び出すことができます。 これを使用して、自動的に定義される受信側の SYSTEM.AUTO.RECEIVER、または、サーバー接続チャネルの SYSTEM.AUTO.SVRCON 用に提供されるデフォルト定義を変更できます。 チャネル定義を自動的に作成する方法については、チャネルの準備を参照してください。

チャネル自動定義出口は、クラスター送信側チャネル開始の要求が受信された場合にも呼び出せます。 この出口を呼び出すことによって、クラスター送信側チャネルとクラスター受信側チャネルの両方で、チャネルのインスタンスの定義変更ができるようになります。 この場合、出口は WebSphere MQ for z/OS にも適用されます。 チャネル自動定義出口は通常、メッセージ出口名 (MSGEXIT、RCVEXIT、SCYEXIT、および SENDEXIT) の変更に使用されます。出口名のフォーマットはプラットフォームによって異なるためです。 チャネル自動定義出口名が指定されていない場合の z/OS でのデフォルト動作は、フォーム *[path]/libraryname(function)* という分散出口名を検査し、存在する場合、関数 (またはライブラリー名) から最高 8 文字を取得します。 z/OS では、チャネル自動定義出口プログラムは、MsgExit、MsgUserData、SendExit、SendUserData、ReceiveExit、および ReceiveUserData フィールド自体ではなく、MsgExitPtr、

MsgUserDataPtr、SendExitPtr、SendUserDataPtr、ReceiveExitPtr、および ReceiveUserDataPtr で指定されるフィールドを変更する必要があります。

詳しくは、チャネルの自動定義を参照してください。

他のチャネル出口の場合は、次のようなパラメーター・リストを指定します。

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

ChannelExitParms については、MQCXP で説明しています。 ChannelDefinition については、MQCD で説明しています。

MQCD には、デフォルトのチャネル定義で使用されている値が (出口によって変更されていない場合は) 入ります。 出口は、フィールドのサブセットだけを変更することができます。詳細については、MQ_CHANNEL_AUTO_DEF_EXIT を参照してください。 ただし、他のフィールドを変更しようとしてもエラーは起こりません。

チャネル自動定義出口は、MQXCC_OK または MQXCC_SUPPRESS_FUNCTION の応答を戻します。 どちらの応答も戻らない場合、MCA は MQXCC_SUPPRESS_FUNCTION が戻ったと想定して処理を続行します。これは自動定義が異常終了したため、新しいチャネル定義が作成されずチャネルを開始できないということです。

Windows、 UNIX and Linux システムでのチャネル出口プログラムのコンパイル

次の例を使用して、Windows、UNIX and Linux システムでのチャネル出口プログラムをコンパイルします。

Windows

Windows

Windows チャネル出口プログラムのコンパイラーおよびリンカー・コマンド:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

UNIX および Linux システム

Linux UNIX

これらの例では、exit はライブラリー名で、ChannelExit は関数名です。 AIX では、エクスポート・ファイルは exit.exp と呼ばれます。 これらの名前は、チャネル定義によって、MQCD チャネル定義で説明されている形式で出口プログラムを参照するために使用されます。 DEFINE CHANNEL コマンドの MSGEXIT パラメーターも参照してください。

AIX におけるチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

HP-UX におけるチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

Linux プラットフォームにおいてキュー・マネージャーが 32 ビットの場合のチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux プラットフォームにおいてキュー・マネージャーが 64 ビットの場合のチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Solaris におけるチャネル出口のコンパイラーおよびリンカー・コマンドのサンプル

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

クライアントでは、32 ビットまたは 64 ビット出口を使用できます。 この出口は、mqic_r にリンクされている必要があります。

AIX では、IBM WebSphere MQ によって呼び出されるすべての関数をエクスポートする必要があります。 この Make ファイルのサンプル・エクスポート・ファイル:

```
#!
channelExit
MQStart
```

チャネル出口の構成

チャネル出口を呼び出すには、チャネル定義でそのチャネルに名前を指定する必要があります。

チャネル出口には、チャネル定義で名前を付ける必要があります。 この命名は、チャネルを最初に定義するときに行うことができます。または、例えば MQSC コマンドの ALTER CHANNEL を使用して、後で情報を追加することができます。 MQCD チャネル・データ構造体でチャネル出口名を指定することもできます。出口名のフォーマットは、IBM WebSphere MQ プラットフォームによって異なります。詳細については、MQCD またはスクリプト (MQSC) コマンドを参照してください。

チャネル定義にユーザー出口プログラム名が含まれない場合、そのユーザー出口は呼び出されません。

チャネル自動定義出口は、キュー・マネージャーの特性であり、個別チャネルの特性ではありません。 この出口が呼び出されるようにするには、キュー・マネージャーの定義で名前を指定する必要があります。キュー・マネージャー定義を変更するには、MQSC コマンドの ALTER QMGR を使用します。

データ変換出口の作成

この一連のトピックには、データ変換出口を作成する方法に関する情報が含まれています。

注 : MQSeries for VSE/ESA ではサポートされていません。

MQPUT を実行する際、アプリケーションはメッセージのメッセージ記述子 (MQMD) を作成します。 MQMD は、作成されるプラットフォームに関係なく、その内容が WebSphere MQ に理解できるものでなければならないので、システムによって自動的に変換されます。

ただし、アプリケーション・データは自動的に変換されません。 *CodedCharSetId* フィールドおよび *Encoding* フィールドが異なるプラットフォーム間 (例えば ASCII と EBCDIC の間) で文字データがやり取りされている場合、メッセージの変換はアプリケーション側で調整する必要があります。 アプリケーションのデータ変換は、キュー・マネージャー自体またはデータ変換出口 と呼ばれるユーザー出口プログラムで実行できます。 アプリケーション・データが組み込み形式の 1 つ (MQFMT_STRING など) である場合、キュー・マネージャー自体が、組み込み変換ルーチンの 1 つを使って、データ変換を実行できます。 このトピックには、 アプリケーション・データが組み込まれている形式とは異なる場合のために WebSphere MQ が提供する、 データ変換機能に関する情報が含まれています。

MQGET 呼び出しの処理中に、制御をデータ変換出口に渡すことができます。 これは、最終宛先に到着する前に異なるプラットフォーム間でデータが変換されないようにします。 ただし、最終宛先が MQGET でのデータ変換をサポートしていないプラットフォームであれば、データをその最終宛先に送る送信側のチャネルで CONVERT(YES) を指定する必要があります。 これによって、間違いなく WebSphere MQ で伝送中にデータが変換されます。 このとき、送信側のチャネルを定義しているシステムには、必ずデータ変換出口が設定されていなければなりません。

MQGET 呼び出しは、アプリケーションによって直接発行されます。 MQMD 中の *CodedCharSetId* および *Encoding* フィールドを、必要な文字セットおよびエンコードに設定してください。 アプリケーションがキュー・マネージャーと同じ文字セットおよびエンコードを使用する場合は、*CodedCharSetId* を MQCCSI_Q_MGR に、*Encoding* を MQENC_NATIVE に設定します。 MQGET 呼び出しの完了後、これらのフィールドには戻されるメッセージ・データに該当する値が指定されます。 変換が失敗した場合、これらのフィールドの値が必要な値とは異なることがあります。 その場合、アプリケーションは、これらのフィールドを、各 MQGET 呼び出しの前に必要とされた値にリセットしなければなりません。

MQGET 呼び出し用にデータ変換出口が呼び出されるために必要な条件は、MQGET で定義されています。

データ変換出口に渡される MQ_DATA_CONV_EXIT 呼び出しと MQDXP 構造体のパラメーターと詳細な使用上の注意については、データ変換を参照してください。

アプリケーション・データをエンコード方式の異なるマシン間および異なる CCSID 間で変換するプログラムは、WebSphere MQ データ変換インターフェース (DCI) に必ず準拠していなければなりません。

マルチキャスト・クライアントの導入とともに、 API 出口とデータ変換出口はクライアント・サイドで実行可能であることが必要になりました。 これは、一部のメッセージがキュー・マネージャーを経由しない可能性があるためです。 以下のライブラリーは、現在ではサーバー・パッケージだけでなくクライアント・パッケージの一部にもなっています。

表 *56.* 現在ではクライアント・パッケージとサーバー・パッケージの両方に含まれているライブラリー

オペレーティング・システム	**ライブラリー**
Windows	32 ビットおよび 64 ビット: mqm.dll および mqm.pdb
Linux & HP-UX	32 ビットおよび 64 ビット: libmqm.so および libmqm_r.so
AIX	32 ビットおよび 64 ビット: libmqm.a および libmqm_r.a
Solaris	32 ビットおよび 64 ビット: libmqm.so

データ変換出口の起動

データ変換出口とは、MQGET 呼び出しの処理中に制御を受け取るユーザー作成出口です。

次のことが当てはまる場合に、この出口が呼び出されます。

- MQGMO_CONVERT オプションが、MQGET 呼び出しで指定されている。
- メッセージ・データの一部または全部が、必要な文字セットまたはエンコードになっていない。
- メッセージに関連付けられている MQMD 構造体の *Format* フィールドが、MQFMT_NONE でない。
- MQGET 呼び出しで指定された *BufferLength* が、ゼロでない。
- メッセージ・データの長さが、ゼロでない。
- メッセージに、ユーザー定義フォーマットのデータが入っている。 メッセージ全体がユーザー定義フォーマットになっていることもあれば、その前に 1 つまたは複数の組み込みフォーマットが来ることもあります。 例えば、ユーザー定義フォーマットの前に、MQFMT_DEAD_LETTER_HEADER フォーマットが来ることがあります。 出口は、ユーザー定義フォーマットだけを変換するために呼び出されます。一方、キュー・マネージャーは、ユーザー定義フォーマットの前に来るすべての組み込みフォーマットを変換します。

  また、ユーザー作成出口が組み込みフォーマットを変換するために呼び出されることもありますが、これは組み込み変換ルーチンが組み込みフォーマットを正常に変換できなかった場合に限られます。

その他にも条件があり、MQ_DATA_CONV_EXIT の MQ_DATA_CONV_EXIT 呼び出しの使用上の注意で詳しく説明されています。

MQGET 呼び出しの詳細については、MQGET を参照してください。 MQXCNVC 以外のデータ変換出口は、MQI 呼び出しを使用できません。

出口の新規コピーがロードされるのは、アプリケーションがキュー・マネージャーに接続された後、*Format* を使用する最初のメッセージを検索しようとするときです。 これ以外のときにも、前にロードされたコピーをキュー・マネージャーが既に廃棄した場合には、新規コピーがロードされることがあります。

データ変換出口は、MQGET 呼び出しを発行するプログラムの環境と同様の環境で稼働します。 ユーザー・アプリケーションと同様に、プログラムはメッセージ変換をサポートしない宛先キュー・マネージャーにメッセージを送信する MCA (メッセージ・チャネル・エージェント) であっても構いません。 この環境には、該当する場合、アドレス・スペースとユーザー・プロファイルが組み込まれます。 出口は、キュー・マネージャーの環境で稼働しないので、そのキュー・マネージャーの保全性を損なうことはありません。

WebSphere MQ (UNIX and Linux システム用) 用のデータ変換出口の作成

WebSphere MQ (UNIX and Linux システム用) のデータ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. メッセージ・フォーマットに名前を付けます。 その名前は、必ず MQMD の *Format* フィールド内に入る長さにし、英大文字で表記 (例えば MYFORMAT) してください。 *Format* 名の先頭には、ブランクは指定できません。 また、名前の後ろのブランクは無視されます。 *Format* の長さは 8 文字のみであるため、このオブジェクトの名前は、ブランク以外の、8 文字までの文字で構成しなければなりません。 メッセージを 1 つ送信するたびに、必ずこの名前を指定してください。

   スレッド化環境でデータ変換出口を使用する場合は、ロード可能オブジェクトの後に、それがスレッド化バージョンであることを示す _r を付ける必要があります。
2. メッセージを表す構造体を作成します。 この例は、有効な構文に記載されています。
3. `crtmqcvx` コマンドを使用してこの構造体を実行し、データ変換出口用のコードのフラグメントを作成します。

   `crtmqcvx` コマンドで生成される関数は、すべての構造体がパックされていることを前提とするマクロを使用します。そうなっていない場合は、修正する必要があります。
4. 提供されているスケルトン・ソース・ファイルをコピーし、ファイルの名前をステップ 418 ページの『1』 で設定したメッセージ・フォーマットの名前に変更します。 スケルトン・ソース・ファイル、およびそのコピーは読み取り専用です。

   スケルトン・ソース・ファイルは、amqsvfc0.c という名前のファイルです。
5. WebSphere MQ for AIX には、amqsvfc.exp という名前のスケルトン・エクスポート・ファイルも用意されています。 このファイルをコピーして、ファイル名を MYFORMAT.EXP に変更します。
6. このスケルトンでは、*MQ_INSTALLATION_PATH*/inc ディレクトリー内にサンプル・ヘッダー・ファイル amqsvmha.h が組み込まれています。ここで *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。 組み込みパスがこのディレクトリーを指して、このファイルを選択するようになっていることを確認してください。

   amqsvmha.h ファイルには、`crtmqcvx` コマンドで生成されたコードが使用するマクロが入っています。 変換される文字データに構造体が入っている場合には、これらのマクロにより、MQXCNVC が呼び出されます。
7. ソース・ファイルから以下のコメント・ボックスを検索し、その中の指示どおりにコードを挿入します。

   a. ソース・ファイルの最後の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   ここに、ステップ 418 ページの『3』 で生成したコードのフラグメントを挿入します。

   b. ソース・ファイルのほぼ中央に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   このあとには、コメント化された `ConverttagSTRUCT` 関数の呼び出しが続きます。

この関数の名前をステップ 418 ページの『7.a』 で追加した関数の名前に変更します。 コメント文字を削除して、関数を活動状態にします。 関数がいくつかある場合は、各関数ごとに呼び出しを作成します。

c. ソース・ファイルの最初の方に、以下で始まるコメント・ボックスがあります。

```
/* Insert the function prototypes for the functions produced by */
```

ここに、上記のステップ 418 ページの『3』 で追加した関数のプロトタイプ・ステートメントを挿入します。

8. MQStart を入り口点に使用して、出口を共用ライブラリーとしてコンパイルします。 その方法については、419 ページの『UNIX and Linux システムでのデータ変換出口のコンパイル』を参照してください。
9. 出口ディレクトリーに出力を入れます。 デフォルトの出口ディレクトリーは、32 ビット・システムの場合は /var/mqm/exits で、64 ビット・システムの場合は /var/mqm/exits64 です。 qm.ini ファイルまたは mqclient.ini ファイルでこれらのディレクトリーを変更できます。 このパスは、それぞれのキュー・マネージャーで設定できます。出口の検索は、このパスでのみ行われます。

注 :

1. crtmqcvx がパックされた構造体を使用している場合は、すべての WebSphere MQ アプリケーションをこの方法でコンパイルする必要があります。
2. データ変換出口プログラムは、必ず再入可能でなければなりません。
3. MQXCNVC は、データ変換出口から発行できるただ *1* つの MQI 呼び出しです。

UNIX and Linux システムでのデータ変換出口のコンパイル
UNIX and Linux システムでのデータ変換出口のコンパイル方法について例を示します。

どのプラットフォームでも、モジュールへの入り口点は MQStart です。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

AIX

次のいずれかのコマンドを発行して出口ソース・コードをコンパイルします。

32 ビット・アプリケーション
非スレッド化

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション
非スレッド化

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
    MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

HP-UX Itanium プラットフォーム

次のいずれかのコマンド・セットを発行して出口ソース・コードをコンパイルおよびリンクします。

32 ビット・アプリケーション

非スレッド化

出口ソース・コードをコンパイルします。

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

出口オブジェクトをリンクします。

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

スレッド化

出口ソース・コードをコンパイルします。

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

出口オブジェクトをリンクします。

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
    -lpthread
rm MYFORMAT.o
```

64 ビット・アプリケーション

非スレッド化

出口ソース・コードをコンパイルします。

```
 c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

出口オブジェクトをリンクします。

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

スレッド化

出口ソース・コードをコンパイルします。

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

出口オブジェクトをリンクします。

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

Linux

次のいずれかのコマンドを発行して出口ソース・コードをコンパイルします。

31 ビット・アプリケーション
非スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

32 ビット・アプリケーション
非スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

64 ビット・アプリケーション
非スレッド化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

スレッド化

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Solaris

次のいずれかのコマンドを発行して出口ソース・コードをコンパイルします。

32 ビット・アプリケーション
SPARC プラットフォーム

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

x86-64 プラットフォーム

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

64 ビット・アプリケーション
SPARC プラットフォーム

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

x86-64 プラットフォーム

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64  -lsocket -lnsl -ldl
```

WebSphere MQ for Windows 用のデータ変換出口の作成

WebSphere MQ for Windows 用のデータ変換出口プログラムを作成する際に検討するステップについての情報です。

次のステップを行います。

1. メッセージ・フォーマットに名前を付けます。 その名前は、必ず MQMD の *Format* フィールド内に入る長さにしてください。 *Format* 名の先頭には、ブランクは指定できません。 また、名前の後ろのブランクは無視されます。 *Format* の長さは 8 文字のみであるため、このオブジェクトの名前は、ブランク以外の、8 文字までの文字で構成しなければなりません。

   amqsvfcn.def という名前の .DEF ファイルも、サンプル・ディレクトリー *MQ_INSTALLATION_PATH*\Tools\C\Samples にあります。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。 このファイルのコピーを取り、そのファイル名を、例えば MYFORMAT.DEF に変更します。 作成している DLL の名前と MYFORMAT.DEF に指定されている名前は、必ず同じにしてください。 MYFORMAT.DEF にある名前 FORMAT1 を、新しい形式名で上書きします。

   メッセージを 1 つ送信するたびに、必ずこの名前を指定してください。
2. メッセージを表す構造体を作成します。 この例は、有効な構文に記載されています。
3. crtmqcvx コマンドを使用してこの構造体を実行し、データ変換出口用のコードのフラグメントを作成します。

   CRTMQCVX コマンドで生成された関数は、すべての構造体がパックされていることを前提として作成されたマクロを使用します。そうなっていない場合は、修正する必要があります。
4. 提供されているスケルトン・ソース・ファイル amqsvfc0.c をコピーし、ファイルの名前をステップ 422 ページの『1』 で設定したメッセージ・フォーマットの名前に変更します。

   amqsvfc0.c は *MQ_INSTALLATION_PATH*\Tools\C\Samples にあります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ がインストールされているディレクトリーです。 (デフォルトのインストール・ディレクトリーは C:\Program Files\IBM\WebSphere MQ です。)

   スケルトンには、 *MQ_INSTALLATION_PATH*\Tools\C\include ディレクトリーにサンプル・ヘッダー・ファイル amqsvmha.h が含まれています。 組み込みパスがこのディレクトリーを指して、このファイルを選択するようになっていることを確認してください。

   amqsvmha.h ファイルには、CRTMQCVX コマンドで生成されたコードが使用するマクロが入っています。 変換される文字データに構造体が入っている場合には、これらのマクロにより、MQXCNVC が呼び出されます。
5. ソース・ファイルから以下のコメント・ボックスを検索し、その中の指示どおりにコードを挿入します。

   a. ソース・ファイルの最後の方に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   ここに、ステップ 422 ページの『3』 で生成したコードのフラグメントを挿入します。

   b. ソース・ファイルのほぼ中央に、以下で始まるコメント・ボックスがあります。

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   このあとには、コメント化された ConverttagSTRUCT 関数の呼び出しが続きます。

   この関数の名前をステップ 422 ページの『5.a』 で追加した関数の名前に変更します。 コメント文字を削除して、関数を活動状態にします。 関数がいくつかある場合は、各関数ごとに呼び出しを作成します。

   c. ソース・ファイルの最初の方に、以下で始まるコメント・ボックスがあります。

```
/* Insert the function prototypes for the functions produced by */
```

ここに、ステップ 422 ページの『3』 で追加した関数のプロトタイプ・ステートメントを挿入します。

6. 次のコマンド・ファイルを作成します。

```
cl -I MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C

MYFORMAT.DEF
```

ここで *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされているディレクトリーです。

7. このコマンド・ファイルを発行して、出口を DLL ファイルとしてコンパイルします。
8. WebSphere MQ データ・ディレクトリーの下にある出口サブディレクトリーに出力を入れます。 32 ビット・システムに出口をインストールするためのデフォルト・ディレクトリーは *MQ_DATA_PATH*\Exits で、64 ビット・システムの場合は *MQ_DATA_PATH*\Exits64 になります。

データ変換出口の検索では、レジストリー内パスが使われます。 レジストリー・フォルダーは次のとおりです。

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\Configuration\ClientExitPa
th\
```

レジストリー・キーは ExitsDefaultPath です。 このパスは、それぞれのキュー・マネージャーで設定できます。出口の検索は、このパスでのみ行われます。

注 :

1. CRTMQCVX がパックされた構造体を使用している場合は、すべての WebSphere MQ アプリケーションをこの方法でコンパイルする必要があります。
2. データ変換出口プログラムは、必ず再入可能でなければなりません。
3. MQXCNVC は、データ変換出口から発行できるただ *1* つの MQI 呼び出しです。

Windows オペレーティング・システムにおける出口ロード・ファイルとスイッチ・ロード・ファイル

IBM WebSphere MQ for Windows Version 7.5 キュー・マネージャー・プロセスは 32 ビットです。 そのため、64 ビット・アプリケーションを使用する際に、いくつかのタイプの出口ロード・ファイルや XA スイッチ・ロード・ファイルで、32 ビット・バージョンのものもキュー・マネージャーで使用できるようにしておく必要があります。 32 ビット・バージョンの出口ロード・ファイルまたは XA スイッチ・ロード・ファイル が必要であるのに、それが使用可能になっていない場合は、関連する API 呼び出しまたはコマンドが失敗します。

ExitPath の qm.ini file では、2 つの属性がサポートされています。 これらは ExitsDefaultPath=*MQ_INSTALLATION_PATH*\exits および ExitsDefaultPath64=*MQ_INSTALLATION_PATH*\exits64 です。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。 これらの属性を使用すると、適切なライブラリーを確実に見つけることができます。 さらに、出口が WebSphere MQ クラスターで使用される場合は、この属性を使用することによって、リモート・システム上に適切なライブラリーが確実に見つかるようにすることができます。

以下の表はさまざまなタイプの出口ロード・ファイルとスイッチ・ロード・ファイルをリストしたもので、それぞれ 32 ビット・アプリケーションが使用されるときと 64 ビット・アプリケーションが使用されるときに、32 ビット・バージョンと 64 ビット・バージョンのどちら (あるいは両方) が必要になるかを示しています。

ファイル・タイプ	32 ビット・アプリケーション	64 ビット・アプリケーション
API 交差出口 (API-crossing exit)	32 ビット	32 ビットおよび 64 ビット

ファイル・タイプ	32 ビット・アプリケーション	64 ビット・アプリケーション
データ変換出口	32 ビット	64 ビット
サーバー・チャネル出口 (全タイプ)	32 ビット	32 ビット
クライアント・チャネル出口 (全タイプ)	32 ビット	64 ビット
インストール可能サービス出口	32 ビット	32 ビット
サービス・トレース・モジュール	32 ビット	32 ビットおよび 64 ビット
クラスター WLM 出口	32 ビット	32 ビット
Pub/Sub ルーティング出口	32 ビット	32 ビット
データベース・スイッチ・ロード・ファイル	32 ビット	32 ビットおよび 64 ビット
外部トランザクション・マネージャー AX ライブラリー	32 ビット	64 ビット

リポジトリーから接続前出口を使用した接続定義の参照

WebSphere MQ MQI クライアントは、リポジトリーを検索して、接続前出口ライブラリーを使用して接続定義を取得するように構成できます。

概要

クライアント・アプリケーションは、 クライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続できます。 一般に、CCDT ファイルは中心にあるネットワーク・ファイル・サーバーにあり、 それを参照するクライアントが存在します。 CCDT ファイルを参照するさまざまなクライアント・アプリケーションを管理することは難しいため、柔軟なアプローチとして、クライアント定義をグローバル・リポジトリー (LDAP ディレクトリー、WebSphere Registry and Repository、または他のリポジトリー) に保管することができます。 クライアント接続定義をリポジトリーに保管すると、より簡単にクライアント接続定義を管理できるようになり、アプリケーションが適切で最も新しいクライアント接続定義にアクセスできるようになります。

MQCONN/X 呼び出しの実行時に、IBM WebSphere MQ MQI client はアプリケーションが指定した 接続前出口ライブラリーをロードし、接続定義を取り出すために出口機能を呼び出します。 その後、取り出された接続定義を使用して、キュー・マネージャーへの接続が確立されます。 呼び出される出口ライブラリーと関数の詳細は、mqclient.ini 構成ファイルで指定されます。

構文

```
void MQ_PRECONNECT_EXIT (pExitParms, pQMgrName, ppConnectOpts, pCompCode, pReason);
```

Parameters

pExitParms

タイプ: PMQNXP 入出力

PreConnection 出口パラメーター構造体。

この構造体は、出口の呼び出し側によって割り振られて維持されます。

pQMgrName

タイプ: PMQCHAR 入出力

キュー・マネージャーの名前。

入力において、このパラメーターは、**QMgrName** パラメーターを介して MQCONN API 呼び出しに提供されたフィルター・ストリングです。 このフィールドはブランクであるか、内容が明示的に指定されているか、特定のワイルドカード文字が含まれる場合があります。 このフィールドは出口によって変更されます。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。

ppConnectOpts
タイプ: ppConnectOpts 入出力

MQCONNX のアクションを制御するオプション。

これは、 MQCONN API 呼び出しのアクションを制御する MQCNO 接続オプション構造へのポインターです。 出口が MQXR_TERM を使用して呼び出された場合、このパラメーターは NULL です。 MQCNO 構造がアプリケーションによって元々提供されなかった場合を含め、MQI クライアントは常に MQCNO 構造を出口に提供します。 アプリケーションが MQCNO 構造を提供した場合、 クライアントは複製を作成して出口に渡し、そこで構造が変更されます。 クライアントは MQCNO の所有権を保持します。

MQCNO を介して参照される MQCD は、配列を介して提供されたすべての接続定義より優先されます。 クライアントはキュー・マネージャーに接続するために MQCNO 構造体を使用し、その他の定義は無視されます。

pCompCode
タイプ: PMQLONG 入出力

完了コード

出口完了コードを受け取る MQLONG へのポインター。 値は、次のいずれかでなければなりません。

- MQCC_OK - 正常終了。
- MQCC_WARNING - 警告 (部分的に完了)
- MQCC_FAILED - 呼び出しの失敗。

pReason
タイプ: PMQLONG 入出力

pCompCode を修飾する理由。

出口理由コードを受け取る MQLONG へのポインター。 完了コードが MQCC_OK の場合、以下の値だけが有効です。

- MQRC_NONE - (0, x'000') 報告する理由はありません。

完了コードが MQCC_FAILED または MQCC_WARNING の場合、出口関数は理由コード・フィールドを任意の有効な MQRC_* 値に設定できます。

C 言語での呼び出し

```
void MQ_PRECONNECT_EXIT (&ExitParms, &QMgrName, &pConnectOpts, &CompCode, &Reason);
```

Parameter

```
PMQNXP  pExitParms   /*PreConnect exit parameter structure*/
PMQCHAR pQMgrName    /*Name of the queue manager*/
PPMQCNO ppConnectOpts/*Options controlling the action of MQCONNX*/
PMQLONG pCompCode    /*Completion code*/
PMQLONG pReason      /*Reason qualifying pCompCode*/
```

クライアント構成ファイルの *PreConnect* スタンザ

PreConnect スタンザを使用して、mqclient.ini ファイルで PreConnect 出口を構成します。

次の属性を PreConnect スタンザに含めることができます。

Data=< ***URL*****>**
接続定義が保管されるリポジトリーの URL。 例えば、LDAP サーバーを使用しているときには、次のようになります。

Data = ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com

Function=*<myFunc>*
PreConnect 出口コードを含むライブラリーへの、関数のエントリー・ポイントの名前。

関数定義は、PreConnect 出口のプロトタイプ MQ_PRECONNECT_EXIT に従います。

このフィールドの最大長は MQ_EXIT_NAME_LENGTH です。

Module=*< amqldapi>*
API 出口コードを含むモジュールの名前。

このフィールドにモジュールの絶対パス名が指定されていると、そのまま使用されます。

Sequence の場合:*< sequence_number>*
その他の出口に関してこの出口が呼び出される順序。 小さなシーケンス番号の出口は、より大きなシーケンス番号の出口よりも先に呼び出されます。 出口のシーケンス番号は連続である必要はありません。つまり、1、2、3 の順序は、7、42、1096 の順序と同じ結果となります。 この属性は、符号なし数値です。

mqclient.ini ファイル内で複数の PreConnect スタンザを定義できます。 各出口の処理順序は、スタンザの Sequence 属性によって決定されます。

パブリッシュ出口の作成とコンパイル

キュー・マネージャーでパブリッシュ出口を構成し、パブリッシュされたメッセージがサブスクライバーによって受信される前にその内容を変更できるようにすることが可能です。 また、メッセージ・ヘッダーの変更や、メッセージをサブスクリプションに送信させないことも可能です。

パブリッシュ出口は z/OS ではサポートされていません。

パブリッシュ出口を使用して、サブスクライバーに送達されるメッセージを検査および変更できます。

- 各サブスクライバーにパブリッシュされるメッセージの内容を検査する
- 各サブスクライバーにパブリッシュされるメッセージの内容を変更する
- メッセージが入れられるキューを変更する
- サブスクライバーへのメッセージの送達を停止する

パブリッシュ出口の作成

376 ページの『出口とインストール可能サービスの作成とコンパイル』の手順を使用すると、出口の作成およびコンパイルに役立ちます。

パブリッシュ出口のプロバイダーは、出口で実行する内容を定義します。 ただし、この出口は、MQPSXP で定義された規則に従う必要があります。

WebSphere MQ では、MQ_PUBLISH_EXIT エントリー・ポイントの実装は提供されません。 C 言語の typedef 宣言が用意されています。 typedef を使用することにより、ユーザー作成出口のパラメーターを正しく宣言します。 次の例は、typedef 宣言の使用方法を示しています。

```
#include "cmqec.h"

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                            PMQPBC  pPubContext,
                            PMQSBC  pSubContext )
{
  /* C language statements to perform the function of the exit */
}
```

パブリッシュ出口は、以下の操作の結果として、キュー・マネージャー・プロセスで実行されます。

- 1 つ以上のサブスクライバーにメッセージが送信されるパブリッシュ操作
- 1 つ以上の保存メッセージが送信されるサブスクライブ操作

• 1 つ以上の保存メッセージが送信されるサブスクリプション要求操作

接続のためにパブリッシュ出口が呼び出された場合、初めての呼び出し時に *ExitReason* コードが MQXR_INIT に設定されます。 パブリッシュ出口を使用したら、接続が切断される前に、*ExitReason* コードを MQXR_TERM に指定して、その出口が呼び出されます。

パブリッシュ出口が構成されていても、キュー・マネージャーの開始時にロードできない場合、そのキュー・マネージャーについてはパブリッシュ/サブスクライブ・メッセージ操作が禁止されます。 問題を修正するか、パブリッシュ/サブスクライブ・メッセージングが再び使用可能になる前にキュー・マネージャーを再始動する必要があります。

パブリッシュ出口に必要な WebSphere MQ の各接続は、出口をロードできないか、または出口を初期化できない場合があります。 出口をロードまたは初期化できない場合、パブリッシュ出口に必要なパブリッシュ/サブスクライブ操作は、その接続には使用不可になります。 WebSphere MQ の理由コード MQRC_PUBLISH_EXIT_ERROR により、操作に失敗します。

パブリッシュ出口が呼び出されるコンテキストは、アプリケーションによるキュー・マネージャーへの接続です。 ユーザー・データ域は、パブリッシュ操作を実行している接続ごとにキュー・マネージャーによって維持されます。 出口は、各接続のユーザー・データ域に情報を保持できます。

パブリッシュ出口は一部の MQI 呼び出しを使用できます。 使用できる MQI 呼び出しは、メッセージ・プロパティーを操作するものだけです。 これらの呼び出しを以下に示します。

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

パブリッシュ出口で宛先キュー・マネージャーまたはキュー名が変更された場合、新規の権限検査は実行されません。

パブリッシュ出口のコンパイル

パブリッシュ出口は動的にロードされるライブラリーであり、チャネル出口と考えることができます。 出口のコンパイルについては、376 ページの『出口とインストール可能サービスの作成とコンパイル』を参照してください。

サンプル・パブリッシュ出口

サンプル出口プログラムは、amqspse0.c と呼ばれます。 出口が初期化、パブリッシュ、または終了の中のどの操作のために呼び出されたかに応じて、別のメッセージがログ・ファイルに書き込まれます。 また、ストレージを適切に割り振りまたは解放するための出口ユーザー域フィールドの使用方法を示します。

パブリッシュ出口の構成

特定の属性を定義して、パブリッシュ出口を構成する必要があります。

Windows および Linux では、 WebSphere MQ エクスプローラーを使用して属性を定義できます。 属性は、「キュー・マネージャー・プロパティー」ページの「パブリッシュ/サブスクライブ」の下で定義されます。

UNIX システムおよび LINUX システムで、qm.ini ファイルでパブリッシュ出口を構成するには、PublishSubscribe という名前のスタンザを作成します。 PublishSubscribe スタンザの属性は次のとおりです。

PublishExitPath=[path]|*module_name*
パブリッシュ出口コードを含むモジュールの名前とパス。 このフィールドの最大長は MQ_EXIT_NAME_LENGTH です。 デフォルトでは、パブリッシュ出口はありません。

PublishExitFunction=*function_name*
パブリッシュ出口コードを含むモジュールへの関数エントリー・ポイントの名前。 このフィールドの最大長は MQ_EXIT_NAME_LENGTH です。

PublishExitData=*string*
キュー・マネージャーがパブリッシュ出口を呼び出している場合、入力として MQPSXP 構造体が渡されます。 *PublishExitData* 属性を使用して指定されたデータは、構造の *ExitData* フィールドに入ります。 ストリングの最大長は、MQ_EXIT_DATA_LENGTH 個の文字になります。 デフォルトは 32 個のブランク文字です。

クラスター・ワークロード出口の作成とコンパイル

クラスターのワークロード管理をカスタマイズするには、クラスター・ワークロード出口プログラムを作成します。 メッセージをルーティングするときに、1 日のさまざまな時間のチャネル使用コストやメッセージの内容を考慮に入れる場合があります。 これらは、標準ワークロード管理アルゴリズムでは考慮されていない要因です。

ほとんどの場合、ワークロード管理アルゴリズムはニーズを満たします。 しかし、 ワークロード管理を調整する独自のユーザー出口プログラムを使用する場合のために、WebSphere MQ には、 クラスター・ワークロード出口というユーザー出口が組み込まれています。

ネットワークやメッセージに関する何らかの特定の情報を持っていて、それをワークロード・バランシングの操作に利用できる場合があります。 大容量のチャネルや低コストのネットワーク経路がわかっている場合や、メッセージをその内容に応じてルーティングしたい場合があります。 クラスター・ワークロード出口プログラムを作成するか、サード・パーティーが提供するものを使用するかを決定できます。

クラスター・ワークロード出口は、クラスター・キューへのアクセス時に呼び出されます。 MQOPEN、MQPUT1、および MQPUT によって呼び出されます。

MQOO_BIND_ON_OPEN が指定されている場合は、MQOPEN 時に選択されたターゲット・キュー・マネージャーに固定されます。 この場合は、出口は 1 回だけ実行されます。

MQOPEN 時にターゲット・キュー・マネージャーが固定されていない場合、ターゲット・キュー・マネージャーは MQPUT 呼び出しの時点で選択されます。 ターゲット・キュー・マネージャーが使用不可の場合、またはメッセージがまだ伝送キューにある間にターゲット・キュー・マネージャーに障害が起こった場合は、出口が再び呼び出されます。 新しいターゲット・キュー・マネージャーが選択されます。 メッセージが転送されている間にメッセージ・チャネルに障害が起こり、メッセージがバックアウトされた場合は、新しいターゲット・キュー・マネージャーが選択されます。

z/OS 以外のプラットフォームでは、キュー・マネージャーは次回に開始されたときに 新しいクラスター・ワークロード出口をロードします。

キュー・マネージャー定義にクラスター・ワークロード出口プログラム名が設定されていない場合は、 そのクラスター・ワークロード出口は呼び出されません。

クラスター・ワークロード出口の出口パラメーター構造体 MQWXP には、次のようなさまざまなデータが渡されます。

- メッセージ定義構造体 MQMD。
- メッセージ長パラメーター。
- メッセージのコピーまたはメッセージの一部。

非 z/OS プラットフォームでは、CLWLMode=FAST を使用した場合、オペレーティング・システムの各プロセスは、それぞれ独自の出口のコピーをロードします。 キュー・マネージャーへの接続が異なると、呼び出される出口のコピーも異なる可能性があります。 出口がデフォルトのセーフ・モード CLWLMode=SAFE で実行される場合は、独自の別プロセスで出口の単一コピーが実行されます。

クラスター・ワークロード出口の作成

z/OS 以外のプラットフォームの場合、クラスター・ワークロード出口で MQI 呼び出しを使用してはなりません。 それ以外の点では、クラスター・ワークロード出口プログラムの作成とコンパイルの規則は、チャネル出口プログラムに適用される規則と同様です。 376 ページの『出口とインストール可能サービスの作

成とコンパイル』のステップ、およびサンプル・プログラム 429 ページの『クラスター・ワークロード出口のサンプル』は、出口を作成およびコンパイルする場合に役立ちます。

チャネル出口の詳細については、401 ページの『チャネル出口プログラムの作成』を参照してください。

クラスター・ワークロード出口の構成

ALTER QMGR コマンドでクラスター・ワークロード出口属性を指定することにより、キュー・マネージャー定義でクラスター・ワークロード出口に名前を付けます。 以下に例を示します。

```
ALTER QMGR CLWLEXIT(myexit)
```

クラスター・ワークロード出口のサンプル

WebSphere MQ にはクラスター・ワークロード出口プログラムのサンプルが組み込まれています。 このサンプルは、コピーして、独自のプログラムの基礎として使用することができます。

z/OS 以外のプラットフォームの場合

サンプル・クラスター・ワークロード出口プログラムは C で提供されており、amqswlm0.c と呼ばれます。 以下の場所にあります。

表 57. クラスター・ワークロード出口プログラムのサンプルの場所 (z/OS 以外)

プラットフォーム	ファイル・パス
AIX、HP-UX、Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Windows	*MQ_INSTALLATION_PATH*\Tools\c\Samples

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

このサンプル出口は、キュー・マネージャーが使用不可になった場合を除いて、すべてのメッセージを特定のキュー・マネージャーに経路指定します。 キュー・マネージャーに障害が起こった場合は、メッセージは別のキュー・マネージャーに経路指定されます。

メッセージの送信先とするキュー・マネージャーを示します。 キュー・マネージャー定義の CLWLDATA 属性に、クラスター受信チャネルの名前を指定します。 以下に例を示します。

```
ALTER QMGR CLWLDATA('my-cluster-name.my-queue-manager')
```

出口を有効にするには、CLWLEXIT 属性に絶対パスと名前を指定します。

UNIX and Linux システムの場合:

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

Windows の場合:

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

このように指定すると、WebSphere MQ は、提供されているワークロード管理アルゴリズムを使用する代わりに この出口を呼び出して、選択されたキュー・マネージャーにすべてのメッセージを経路指定します。

IBM WebSphere MQ アプリケーションの構築

この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

AIX でのアプリケーションの構築

AIX の資料では、作成したプログラムから実行可能なアプリケーションを作成する方法について説明しています。

このトピックでは、 AIX の下で実行する WebSphere MQ for AIX アプリケーションを構築する際に実行する必要がある追加タスク、および標準タスクに対する変更について説明します。 C、C++、および COBOL がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

WebSphere MQ for AIX を使用して実行可能なアプリケーションを作成するときに必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しをコーディングすると共に、使用する言語用の WebSphere MQ for AIX 組み込みファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。 詳細については、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

スレッド化されたサーバーまたはスレッド化されたクライアント・アプリケーションを実行するときは、環境変数 AIXTHREAD_SCOPE=S を設定してください。

AIX での C プログラムの作成

このトピックでは、AIX での C プログラムの作成に必要なライブラリーのリンクについて説明します。

事前にコンパイルした C プログラムは、*MQ_INSTALLATION_PATH*/samp/bin ディレクトリーの中にあります。 ANSI コンパイラーを使用して、次のコマンドを実行してください。 64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

32 ビット・アプリケーションの場合:

```
  $ xlc_r -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

amqsput0 はサンプル・プログラムを示します。

64 ビット・アプリケーションの場合:

```
  $ xlc_r -q64 -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

amqsput0 はサンプル・プログラムを示します。

C++ プログラム用の VisualAge C/C++ コンパイラーを使用している場合は、オプション -q namemangling=v5 を組み込んで、ライブラリーのリンク時にすべての WebSphere MQ シンボルが解決されるようにする必要があります。

WebSphere MQ MQI client for AIX のみがインストールされているマシンでプログラムを使用する場合は、プログラムを再コンパイルして、代わりにクライアント・ライブラリー (-lmqic) にリンクしてください。

ライブラリーのリンク

以下に、必要なライブラリーのリストを示します。

- プログラムを、WebSphere MQ が提供する適切なライブラリーとリンクしてください。

  スレッド以外の環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm.a	C 用サーバー
libmqic.a & libmqm.a	C 用クライアント

スレッド環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm_r.a	C 用サーバー
libmqic_r.a & libmqm_r.a	C 用クライアント

例えば、1 つのコンパイル単位から簡単な WebSphere MQ スレッド・アプリケーションを作成する場合は、次のようなコマンドを実行します。

32 ビット・アプリケーションの場合:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

amqsput0 はサンプル・プログラムを示します。

64 ビット・アプリケーションの場合:

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm_r
```

amqsput0 はサンプル・プログラムを示します。

WebSphere MQ MQI client for AIX のみがインストールされているマシンでプログラムを使用する場合は、プログラムを再コンパイルして、代わりにクライアント・ライブラリー (-lmqic) にリンクしてください。

注:

1. インストール可能なサービス (詳細については、管理を参照) を作成する場合、スレッド化されていないアプリケーションでは libmqmzf.a ライブラリーに、スレッド化されたアプリケーションでは libmqmzf_r.a ライブラリーにリンクする必要があります。
2. XA 準拠のトランザクション管理プログラム (IBM TXSeries、Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成している場合は、非スレッド・アプリケーションの場合には libmqmxa.a (または トランザクション管理プログラムが「long」タイプを 64 ビットとして扱う場合は libmqmxa64.a) および libmqz.a ライブラリーにリンクし、スレッド・アプリケーションの場合には libmqmxa_r.a (または libmqmxa64_r.a) および libmqz_r.a ライブラリーにリンクする必要があります。
3. トラステッド・アプリケーションは、WebSphere MQ のスレッド・ライブラリーにリンクする必要があります。 ただし、 UNIX and Linux システム上の WebSphere MQ のトラステッド・アプリケーションでは、一度に 1 つのスレッドしか接続できません。
4. WebSphere MQ ライブラリーは、その他の製品ライブラリーの前にリンクしなければなりません。

AIX での COBOL プログラムの作成

IBM COBOL Set および Micro Focus COBOL を使用して、AIX で COBOL プログラムを作成する方法について説明します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

- 32 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

また、シンボリック・リンクは次のディレクトリーに作成されます。

```
MQ_INSTALLATION_PATH/inc
```

- 64 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

以下の例では、**COBCPY** 環境変数を次のように設定します。

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

(32 ビット・アプリケーションの場合) および

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

(64 ビット・アプリケーションの場合)

プログラムを次のいずれかのライブラリー・ファイルとリンクする必要があります。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqmcb.a	COBOL 用サーバー (スレッド化されていないアプリケーションの場合)
libmqmcb_r.a	COBOL 用サーバー (スレッド化されたアプリケーションの場合)
libmqicb.a	COBOL 用クライアント (スレッド化されていないアプリケーションの場合)
libmqicb_r.a	COBOL 用クライアント (スレッド化されたアプリケーションの場合)

作成するプログラムにより、IBM COBOL SET コンパイラーまたは Micro Focus COBOL コンパイラーを使用できます (以下参照)。

- amqm で始まるプログラムの場合には Micro Focus COBOL コンパイラーを使用します。および
- amq0 で始まるプログラムの場合にはどちらのコンパイラーを使用しても構いません。

IBM COBOL Set for AIX の使用による COBOL プログラムの作成

COBOL サンプル・プログラムが IBM WebSphere MQ に付属しています。 このプログラムをコンパイルするには、以下のリストの該当するコマンドを入力してください。

32 ビットの非スレッド・サーバー・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-I<COBCPY>
```

32 ビットの非スレッド・クライアント・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-I<COBCPY>
```

32 ビットのスレッド・サーバー・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

32 ビットのスレッド・クライアント・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

64 ビットの非スレッド・サーバー・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -I<COBCPY>
```

64 ビットの非スレッド・クライアント・アプリケーション

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -I<COBCPY>
```

64 ビットのスレッド・サーバー・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

64 ビットのスレッド・クライアント・アプリケーション

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

Micro Focus COBOL の使用による COBOL プログラムの作成

プログラムをコンパイルする前に環境変数を次のように設定します。

```
export COBCPY=<COBCPY>
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Micro Focus COBOL を使用して 32 ビット COBOL プログラムをコンパイルするには、次のように入力します。

- COBOL 用サーバー

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- COBOL 用クライアント

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- COBOL 用スレッド・サーバー

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- COBOL 用スレッド・クライアント

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Micro Focus COBOL を使用して 64 ビット COBOL プログラムをコンパイルするには、次のように入力します。

- COBOL 用サーバー

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- COBOL 用クライアント

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- COBOL 用スレッド・サーバー

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- COBOL 用スレッド・クライアント

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

amqminqx はサンプル・プログラムを示します。

設定する必要のある環境変数については、Micro Focus COBOL に関する資料を参照してください。

AIX での CICS アプリケーション・プログラムの準備

AIX で CICS プログラムを作成する際には、以下の情報を使用してください。

CICS を IBM WebSphere MQ にリンクできるように、XA スイッチ・モジュールが提供されています。

表 *58. AIX* 上の *CICS* アプリケーション・プログラムの必須コード*: XA* 初期化ルーチン

説明	**C (ソース)**	**C (実行可能) - XAD.Stanza に追加する。 XAD.Stanza**
XA 初期化ルーチン	amqzscix.c	*amqzsc* - CICS for AIX

製品に付属している IBM WebSphere MQ スイッチ・ロード・ファイル *amqzsc* の事前作成バージョンを使用します。

C トランザクションは、必ずスレッド・セーフ IBM WebSphere MQ ライブラリー *libmqm_r.a* にリンクしてください。 COBOL ライブラリー *libmqmcb_r.a.*を使用して、COBOL トランザクションを作成します。

CICS トランザクションのサポートの詳細については、管理を参照してください。

TXSeries CICS サポート

AIX 上の IBM WebSphere MQ は、XA インターフェースを使用して TXSeries CICS をサポートします。 CICS アプリケーションがスレッド化されたバージョンの IBM WebSphere MQ ライブラリーにリンクされていることを確認してください。

IBM COBOL Set for AIX または Micro Focus COBOL を使用して CICS プログラムを実行できます。 以下のセクションでは、CICS プログラムを、IBM COBOL Set for AIX で実行する場合と Micro Focus COBOL で実行する場合の相違点について説明します。

C または COBOL のいずれかの同じ CICS 領域にロードされる WebSphere MQ プログラムを作成します。 C および COBOL の MQI 呼び出しを組み合わせて同じ CICS 領域内に入れることはできません。 たいてい

の場合、使用される 2 番目の言語の MQI 呼び出しは失敗し、理由コード MQRC_HOBJ_ERROR が出されます。

IBM COBOL Set for AIX を使用した CICS COBOL プログラムの作成

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

IBM COBOL を使用する場合は、次のステップに従ってください。

1. 次の環境変数をエクスポートする。

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp  \
                -IMQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include  \
                -e _iwz_cobol_main  \
```

   LIB はコンパイラー指示を示します。
2. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l IBMCOB <yourprog>.ccp
```

Micro Focus COBOL の使用による CICS COBOL プログラムの作成

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

Micro Focus COBOL を使用する場合は、次のステップに従ってください。

1. 次のコマンドを使用して、ランタイム・ライブラリーに IBM WebSphere MQ の COBOL ランタイム・ライブラリー・モジュールを追加する。

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

   注: cicsmkcobol を使用すると、IBM WebSphere MQ では COBOL アプリケーションから C プログラミング言語の MQI 呼び出しを行うことはできません。

   既存のアプリケーションにこの種の呼び出しがある場合は、これらの関数を COBOL アプリケーションから myMQ.so などの独自のライブラリーに移動することをお勧めします。 関数を移動した後は、CICS 用の COBOL アプリケーションをビルドするときに IBM WebSphere MQ ライブラリー libmqmcbrt.o を組み込まないでください。

   さらに、COBOL アプリケーションで COBOL MQI 呼び出しが行われない場合は、libmqmz_r と cicsmkcobol をリンクしないでください。

   これにより、Micro Focus COBOL 言語メソッド・ファイルが作成され、CICS の実行時 COBOL ライブラリーが IBM WebSphere MQ (UNIX and Linux システム用) を呼び出すことができるようになります。

   注: cicsmkcobol は、以下の製品のいずれかをインストールする場合にのみ実行してください。

   - Micro Focus COBOL の新規バージョンまたは新規リリース
   - CICS for AIX の新しいバージョンまたはリリース
   - サポートされているデータベース製品の新規バージョンまたは新規リリース (COBOL トランザクション専用)
   - IBM WebSphere MQ の新しいバージョンまたはリリース
2. 次の環境変数をエクスポートする。

```
COBCPY=MQ_INSTALLATION_PATH/inc  export COBCPY
```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l COBOL -e <yourprog>.ccp
```

CICS C プログラムの作成

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

標準の CICS 機能を使用して CICS C プログラムを作成します。

1. 次の環境変数の**いずれか** をエクスポートする。
   - LDFLAGS = "-L/*MQ_INSTALLATION_PATH*lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L*MQ_INSTALLATION_PATH*lib -lmqm_r" export USERLIB
2. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l C amqscic0.ccs
```

CICS C サンプル・トランザクション

AIX IBM WebSphere MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。 トランザクションは、デフォルトのキュー・マネージャーの伝送キュー SYSTEM.SAMPLE.CICS.WORKQUEUE からメッセージを読み取り、メッセージの伝送ヘッダーに名前があるローカル・キューにそれらを入れます。 エラーはすべて、キュー SYSTEM.SAMPLE.CICS.DLQ に送信されます。 サンプル MQSC スクリプト AMQSCIC0.TST を使用して、これらのキューやサンプル入力キューを作成します。

HP Integrity NonStop Server でのアプリケーションの構築

ここでは、 HP Integrity NonStop Server の下で実行する IBM WebSphere MQ Client for HP Integrity NonStop Server アプリケーションをビルドするときに実行する必要がある追加タスクと、標準タスクに対する変更について説明します。

C、COBOL、および pTAL がサポートされています。

OSS および Guardian ヘッダーとパブリック・ライブラリー

OSS および Guardian ヘッダーとパブリック・ライブラリーのリストを示します。 OSS ヘッダー、OSS パブリック実行可能/パブリック・インポート・ライブラリー、Guardian ヘッダー、Guardian パブリック実行可能/パブリック・インポート・ライブラリーがリストされます。

437 ページの『OSS ヘッダー』

437 ページの『OSS パブリック実行可能/パブリック・インポート・ライブラリー』

438 ページの『Guardian ヘッダー』

439 ページの『Guardian パブリック実行可能/パブリック・インポート・ライブラリー』

OSS ヘッダー

表 59. OSS ヘッダー		
オブジェクト	**ロケーション**	**説明**
cmqbc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqfc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqec.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqpsc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqxc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqzc.h	<mqinstall>/inc	IBM WebSphere MQ C 言語ヘッダー (OSS)
cmqcobol.cpy	<mqinstall>/inc	IBM WebSphere MQ COBOL コピーブック (OSS)
cmqbt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL ヘッダー (OSS)
cmqcft.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL ヘッダー (OSS)
cmqpst.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL ヘッダー (OSS)
cmqt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL ヘッダー (OSS)
cmqxt.tal	<mqinstall>/inc	IBM WebSphere MQ pTAL ヘッダー (OSS)

OSS パブリック実行可能/パブリック・インポート・ライブラリー

表 60. OSS パブリック実行可能/パブリック・インポート・ライブラリー		
オブジェクト	**ロケーション**	**説明**
libmqic.so	<mqinstall>/bin	IBM WebSphere MQ パブリック実行可能ライブラリー (OSS 非スレッド)
libmqic_r.so	<mqinstall>/bin	IBM WebSphere MQ パブリック実行可能ライブラリー (OSS マルチスレッド)
libmqic.so	<mqinstall>/lib	IBM WebSphere MQ パブリック・インポート・ライブラリー (OSS 非スレッド)

表 60. OSS パブリック実行可能/パブリック・インポート・ライブラリー (続き)		
オブジェクト	**ロケーション**	**説明**
libmqic_r.so	<mqinstall>/lib	IBM WebSphere MQ パブリック・インポート・ライブラリー (OSS マルチスレッド)
mqicb	<mqinstall>/lib	IBM WebSphere MQ パブリック・インポート・ライブラリー (COBOL 用、OSS)

Guardian ヘッダー

表 61. Guardian ヘッダー		
オブジェクト	**ロケーション**	**説明**
cmqbch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqfch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqech	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqpsch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqxch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqzch	<mqinstall>/inc/G	IBM WebSphere MQ C 言語ヘッダー (Guardian)
cmqcobol	<mqinstall>/inc/G	IBM WebSphere MQ COBOL コピーブック (Guardian)
cmqbt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL ヘッダー (Guardian)
cmqcft	<mqinstall>/inc/G	IBM WebSphere MQ pTAL ヘッダー (Guardian)
cmqpst	<mqinstall>/inc/G	IBM WebSphere MQ pTAL ヘッダー (Guardian)
cmqt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL ヘッダー (Guardian)
cmqxt	<mqinstall>/inc/G	IBM WebSphere MQ pTAL ヘッダー (Guardian)

Guardian パブリック実行可能/パブリック・インポート・ライブラリー

表 62. *Guardian* パブリック実行可能/パブリック・インポート・ライブラリー		
オブジェクト	**ロケーション**	**説明**
mqic	<mqinstall>/bin/G	IBM WebSphere MQ パブリック実行可能ライブラリー (Guardian)
mqicb	<mqinstall>/lib/G	IBM WebSphere MQ パブリック・インポート・ライブラリー (COBOL 用、Guardian)

HP Integrity NonStop Server での C プログラムの作成

このトピックでは、HP Integrity NonStop Server で C プログラムを作成する際の考慮事項について、アプリケーションを構築するときに使用するコマンドの例 (OSS C コンパイラーを使用する場合、および Guardian C コンパイラーを使用する場合) と共に説明します。

事前にコンパイルした C プログラムは、MQ_INSTALLATION_PATH/opt/mqm/samp/bin ディレクトリーの中にあります。 ソース・コードからサンプルを作成するには、c89 コンパイラーを使用します。

プログラムを、IBM WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。 以下の表は、HP Integrity NonStop Server で C プログラムを作成する際にリンクする必要があるライブラリーをリストしています。

表 63. . HP Integrity NonStop Server リンク・ライブラリー	
ライブラリー	**説明**
libmqic.so	OSS 非スレッド化
libmqic_r.so	OSS マルチスレッド化
mqic	Guardian

マルチスレッド化されたネイティブ IBM WebSphere MQ アプリケーションでは、Posix ユーザー・スレッド (PUT) 機能を使用する必要があります。 この製品には、標準 Posix スレッド (SPT) のサポートはありません。

OSS C コンパイラーを使用したアプリケーションの構築

このセクションでは、OSS コンパイラーの使用時に、OSS または Guardian をターゲットとするプログラムを構築するために使用するコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

以下の例では、非スレッド化 C クライアント OSS アプリケーションを構築します。

```
c89 -Wsystype=oss -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
```

以下の例では、マルチスレッド C クライアント OSS アプリケーションを構築します。

```
c89 -Wsystype=oss -D_PUT_MODEL_ -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic_r -lput
```

以下の例では、Guardian C クライアント・アプリケーションを構築します。

```
c89 -Wsystype=guardian -o /G/vol/subvol/amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
```

Guardian C コンパイラーを使用したアプリケーションの構築

このセクションでは、Guardian コンパイラーの使用時に Guardian をターゲットとするプログラムを構築するために使用するコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている Guardian ボリュームおよびサブボリュームを表します。

以下の例では、Guardian C クライアント・アプリケーションを構築します。

```
CCOMP /in AMQSPUT0/ AMQSPUTC;&
       runnable,systype guardian,nolist,&
       ssv0 "$system.system",&
       ssv1 "MQINSTALLATION_SUBVOL",&
       ld(-LMQINSTALLATION_SUBVOL -lmqic)
```

COBOL プログラムの作成

このトピックでは、IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) の C プログラムを作成する際の考慮事項について説明します。 ここでは、アプリケーションを構築するために使用するコマンドの例 (OSS ECOBOL コンパイラーを使用する場合、および Guardian ECOBOL コンパイラーを使用する場合) を示します。

ソース・コードから COBOL サンプルを作成するには、ECOBOL コンパイラーを使用します。

以下の表は、HP Integrity NonStop Server で COBOL プログラムを作成する際に必要なライブラリーをリストしています。 プログラムを、IBM WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。

表 *64.* . HP Integrity NonStop Server リンク・ライブラリー

ライブラリー	**説明**
libmqic.so	OSS 非スレッド化
mqic	Guardian

キュー・マネージャーに接続する COBOL アプリケーションを実行するときには、最初に *SAVE-ENVIRONMENT* 変数を ON に設定する必要があります。 *SAVE-ENVIRONMENT* 変数を ON に設定するには、以下のようにします。

- OSS の場合、以下のコマンドを入力します。

```
export SAVE-ENVIRONMENT=ON
```

- Guardian の場合、以下のコマンドを入力します。

```
param SAVE-ENVIRONMENT ON
```

SAVE-ENVIRONMENT 変数を オンに設定しない場合、アプリケーションがキュー・マネージャーへの接続を試行すると、理由コード 2058 (080A) (RC2058): MQRC_Q_MGR_NAME_ERROR で失敗します。

OSS ECOBOL コンパイラーを使用したアプリケーションの構築

このセクションでは、OSS ECOBOL コンパイラーの使用時に、OSS または Guardian をターゲットとするプログラムを構築するために使用するコマンドの例を示します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

以下の例では、COBOL クライアント OSS アプリケーションを構築します。

```
    ecobol -Wsystype=oss
           -Wcobol="ansi;port"
```

```
          -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
          -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
          -LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
          -o amq0put0
          MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

以下の例では、COBOL クライアント Guardian アプリケーションを構築します。

```
   ecobol -Wsystype=guardian
          -Wcobol="ansi;port;save all"
          -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
          -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
          -LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
          -o amq0put0
          MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

Guardian ECOBOL コンパイラーを使用したアプリケーションの構築

このセクションでは、Guardian ECOBOL コンパイラーの使用時に Guardian をターゲットとするプログラムを構築するために使用するコマンドの例を示します。

MQ_INSTALLATION_SUBVOL は、IBM WebSphere MQ がインストールされている Guardian ボリュームおよびサブボリュームを表します。

以下の例では、COBOL クライアント Guardian アプリケーションを構築します。

```
   ECOBOL /in MQSPUTL/ MQSPUT,MQINSTALLATION_SUBVOL.cmqcobol;
          call-shared;ansi;port;save all;nolist;runnable;
          consult MQINSTALLATION_SUBVOL.mqicb;
          eld(-LMQINSTALLATION_SUBVOL -lmqic)
```

pTAL プログラムの作成

IBM WebSphere MQ クライアント (HP Integrity NonStop Server プラットフォーム用) の pTAL プログラムの作成について学習します。

ソース・コードから pTAL サンプルを作成するには EPTAL コンパイラーを使用します。

注 :

- pTAL IBM WebSphere MQ アプリケーションでは、C または COBOL 言語で書かれたメインルーチンを使用する必要があります。
- pTAL アプリケーションは Guardian でのみ作成可能です。

以下の表は、HP Integrity NonStop Server で pTAL プログラムを作成する際に必要なライブラリーをリストしています。 プログラムを、IBM WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。

表 *65.* . HP Integrity NonStop Server リンク・ライブラリー

ライブラリー	説明
mqic	Guardian

Guardian EPTAL コンパイラーを使用したアプリケーションの構築

このセクションでは、Guardian EPTAL コンパイラーの使用時に Guardian をターゲットとするプログラムを構築するために使用できるコマンドの例を示します。

MQINSTALLATION_SUBVOL は、IBM WebSphere MQ のインストール場所である Guardian ボリュームおよびサブボリュームを表しています。

pTAL IBM WebSphere MQ アプリケーションでは、C または COBOL 言語で書かれたメインルーチンを使用する必要があります。

以下の例では、pTAL クライアント Guardian アプリケーションを構築します。

```
ASSIGN SSV0, $SYSTEM.SYSTEM
ASSIGN SSV1, MQINSTALLATION_SUBVOL

EPTAL /in MQINSTALLATION_SUBVOL.MQSPUTT/ MQSPUTO;nolist

CCOMP /in MQINSTALLATION_SUBVOL.MQSPTMC/ MQSPUT;
       runnable,systype guardian,extensions,nolist,
       ssv0 "$system.system",
       ssv1 "MQINSTALLATION_SUBVOL",
       eld(MQSPUTO -LMQINSTALLATION_SUBVOL -lmqic)
```

HP-UX でのアプリケーションの構築

ここでは、追加の作業および標準作業に対する変更点について説明します。これらの作業は、HP-UX 上で実行する WebSphere MQ for HP-UX アプリケーションの構築時に行う必要があります。

C、C++、および COBOL がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

WebSphere MQ for HP-UX を使用して実行可能なアプリケーションを作成するときに必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しをコーディングすると共に、使用する言語用の WebSphere MQ for HP-UX 組み込みファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

このトピック全体で、複数行にまたがる長いコマンドの分割に円記号 (¥) が使用されています。 実際に入力する場合は、この文字を使用しないで、各コマンドを 1 行に入力してください。

HP-UX での C プログラムの作成

このトピックでは、HP-UX で C プログラムを作成する際の考慮事項を IA64 (IPF) プラットフォームの例と共に説明します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

通常の環境で作業してください。 事前にコンパイルした C プログラムは、*MQ_INSTALLATION_PATH*/samp/bin ディレクトリーの中にあります。

64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

SSL を使用するには、POSIX スレッドを使用して、HP-UX 上に WebSphere MQ MQI クライアントをビルドする必要があります。

以下の例を考慮します。

- 442 ページの『IA64 (IPF) プラットフォーム』
- 444 ページの『ライブラリーのリンク』

IA64 (IPF) プラットフォーム

IA64(IPF) プラットフォーム上で、amqsput0、cliexit、および srvexit の例を ビルドします。

以下の例は、サンプル・プログラム amqsput0 を 非スレッド化 32 ビット環境でクライアント・アプリケーションとしてビルドします。

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

以下の例は、サンプル・プログラム amqsput0 を スレッド化 32 ビット環境でクライアント・アプリケーションとしてビルドします。

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

以下の例は、サンプル・プログラム amqsput0 を 非スレッド化 64 ビット環境でクライアント・アプリケーションとしてビルドします。

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

以下の例は、サンプル・プログラム amqsput0 を スレッド化 64 ビット環境でクライアント・アプリケーションとしてビルドします。

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

以下の例は、サンプル・プログラム amqsput0 を非スレッド化 32 ビット環境でサーバー・アプリケーションとしてビルドします。

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

以下の例は、サンプル・プログラム amqsput0 を スレッド化 32 ビット環境でサーバー・アプリケーションとしてビルドします。

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

以下の例は、サンプル・プログラム amqsput0 を 非スレッド化 64 ビット環境でサーバー・アプリケーションとしてビルドします。

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

以下の例は、サンプル・プログラム amqsput0 を スレッド化 64 ビット環境でサーバー・アプリケーションとしてビルドします。

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

以下の例は、非スレッド化 32 ビット環境でクライアント出口 cliexit を ビルドします。

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

以下の例は、スレッド化 32 ビット環境でクライアント出口 cliexit を ビルドします。

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

以下の例は、非スレッド化 64 ビット環境でクライアント出口 cliexit を ビルドします。

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

以下の例は、スレッド化 64 ビット環境でクライアント出口 cliexit をビルドします。

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

以下の例は、非スレッド化 32 ビット環境でサーバー出口 srvexit をビルドします。

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

以下の例は、スレッド化 32 ビット環境でサーバー出口 srvexit を ビルドします。

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

以下の例は、非スレッド化 64 ビット環境でサーバー出口 srvexit をビルドします。

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c
-IMQ_INSTALLATION_PATHMQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

以下の例は、スレッド化 64 ビット環境でサーバー出口 srvexit をビルドします。

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

ライブラリーのリンク

プログラムを、WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。

以下の表は、さまざまな環境で使用するライブラリーを示しています。

ハードウェア・プラットフォーム	スレッド環境または非スレッド環境のどちらか	プログラム / 出口タイプ	ライブラリー・ファイル
IA64 (IPF)	スレッド化	C 用サーバーおよびクライアント	libmqm_r.so
IA64 (IPF)	スレッド化	C 用クライアント	libmqic_r.so
IA64 (IPF)	非スレッド化	C 用サーバーおよびクライアント	libmqm.so
IA64 (IPF)	非スレッド化	C 用クライアント	libmqic.so

注 :

1. インストール可能なサービス (詳細は、管理を参照) を作成する場合は、`libmqmzf.sl` ライブラリーにリンクする必要があります。
2. XA 準拠のトランザクション・マネージャー (IBM TXSeries Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成しているときは、非スレッド・アプリケーションの場合には `libmqmxa.sl` (または、トランザクション・マネージャーが「long」タイプを 64 ビットとして扱う場合は `libmqmxa64.sl`) および `libmqz.sl` ライブラリーにリンクし、スレッド・アプリケーションの場合には `libmqmxa_r.sl` (または `libmqmxa64_r.sl`) および `libmqz_r.sl` ライブラリーにリンクする必要があります。
3. WebSphere MQ ライブラリーは、その他の製品ライブラリーの前にリンクしなければなりません。

HP-UX での COBOL プログラムの作成

HP-UX での COBOL プログラムの作成、IA64 (IPF) プラットフォーム上の WebSphere MQ での Micro Focus Server Express の使用、および WebSphere MQ MQI クライアント環境でのプログラムの実行について説明します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

ユーザーへの注

1. 32 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   また、シンボリック・リンクは次のディレクトリーに作成されます。

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 64 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 以下の例では、COBCPY を次のように設定します。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   (32 ビット・アプリケーションの場合) および

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   (64 ビット・アプリケーションの場合)

Micro Focus コンパイラーを用いてプログラムをコンパイルします。 構造体を宣言するコピー・ファイルは、*MQ_INSTALLATION_PATH*/inc ディレクトリーにあります。次に例を示します。

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

32 ビット・プログラムのコンパイル:

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

64 ビット・プログラムのコンパイル:

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

amqsput はサンプル・プログラムを示します。

実行時スタック・サイズが十分な値に指定されているかどうか確認してください。推奨値は最低 16 KB です。

プログラムを、WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。 以下の表は、さまざまな環境で使用するライブラリーを示しています。

ハードウェア・プラットフォーム	プログラム / 出口タイプ	ライブラリー・ファイル
IA64 (IPF)	COBOL 用サーバー	libmqmcb.so
IA64 (IPF)	COBOL 用クライアント	libmqicb.so
IA64 (IPF)	スレッド化されたアプリケーション	libmqmcb_r.so

Micro Focus Server Express の IA64 (IPF) プラットフォーム上の WebSphere MQ での使用

HP/IPF プラットフォームでの Micro Focus Server Express と WebSphere MQ の併用について詳しくは、447 ページの『IA64 (IPF) 上の WebSphere MQ for HP-UX によってサポートされるアドレス・スペース・モデル』 を参照してください。

WebSphere MQ MQI クライアント環境で実行するプログラム

LU 6.2 を使用して、MQI クライアントをサーバーに接続する場合は、アプリケーションを libsna.a (SNAplusAPI 製品の一部) にリンクしてください。 コンパイルおよびリンクのコマンドに `-lV3` オプションおよび `-lstr` オプション を使用してください。

- `-lV3` オプションは、プログラムが AT & T 信号ライブラリーにアクセスできるようにします (SNAplusAPI は AT & T 信号を使用します)。
- `-lstr` オプションは、プログラムをストリーム・コンポーネントにリンクします。

HP-UX での CICS プログラムの作成

HP-UX での CICS トランザクション・プログラムの作成について説明します。

CICS のサンプル・トランザクション amqscic0.ccs を作成するには、次のコマンドを実行してください。

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

CICS を WebSphere MQ とリンクできる XA スイッチ・モジュールが提供されます。

表 *66. CICS* アプリケーションの主要コード *(HP-UX)*

説明	C (ソース)	C (実行可能)
XA 初期化ルーチン	amqzscix.c	amqzsc

CICS トランザクションのサポートの詳細については、管理を参照してください。

TXSeries CICS サポート

HP-UX 上の WebSphere MQ は、XA インターフェースを使用して TXSeries CICS をサポートします。 CICS アプリケーションがスレッド化されたバージョンの MQ ライブラリーにリンクされていることを確認してください。

C または COBOL のいずれかの同じ CICS 領域にロードされる WebSphere MQ プログラムを作成します。 C および COBOL の MQI 呼び出しを組み合わせて同じ CICS 領域内に入れることはできません。 たいていの場合、使用される 2 番目の言語の MQI 呼び出しは失敗し、理由コード `MQRC_HOBJ_ERROR` が出されます。

CICS C サンプル・トランザクション

CICS 用の WebSphere MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。トランザクションは、デフォルトのキュー・マネージャーの伝送キュー SYSTEM.SAMPLE.CICS.WORKQUEUE からメッセージを読み取り、メッセージの伝送ヘッダーに名前があるローカル・キューにそれらを入れます。 エラーはすべて、キュー SYSTEM.SAMPLE.CICS.DLQ に送信されます。 サンプル MQSC スクリプト AMQSCIC0.TST を使用して、これらのキューやサンプル入力キューを作成します。

Micro Focus COBOL の使用による CICS COBOL プログラムの作成

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

Micro Focus COBOL を使用する場合は、次のステップに従ってください。

1. 次のコマンドを使用して、実行時ライブラリーに WebSphere MQ の COBOL 実行時ライブラリー・モジュールを追加する。

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   注: `cicsmkcobol` を使用すると、WebSphere MQ では COBOL アプリケーションから C プログラミング言語の MQI 呼び出しを行うことはできません。

   既存のアプリケーションにこの種の呼び出しがある場合は、これらの関数を COBOL アプリケーションから `myMQ.so` などの独自のライブラリーに移動することをお勧めします。 これらの関数を移動した後に、CICS 用の COBOL アプリケーションをビルドする際、WebSphere MQ ライブラリー `libmqmcbrt.o` を組み込まないでください。

   さらに、COBOL アプリケーションで COBOL MQI 呼び出しが行われない場合は、`libmqmz_r` と `cicsmkcobol` をリンクしないでください。

   これにより、Micro Focus COBOL 言語メソッド・ファイルが作成され、CICS の実行時 COBOL ライブラリーが WebSphere MQ (UNIX and Linux システム用) を呼び出すことができるようになります。

   注: `cicsmkcobol` は、以下の製品のいずれかをインストールする場合にのみ実行してください。

   - Micro Focus COBOL の新規バージョンまたは新規リリース
   - CICS for HP-UX の新規バージョンまたは新規リリース
   - サポートされているデータベース製品の新規バージョンまたは新規リリース (COBOL トランザクション専用)
   - WebSphere MQ の新規バージョンまたは新規リリース
2. 次の環境変数をエクスポートする。

   ```
   COBCPY=MQ_INSTALLATION_PATH/inc  export COBCPY
   ```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

   ```
   cicstcl -l COBOL -e <yourprog>.ccp
   ```

IA64 (IPF) 上の WebSphere MQ for HP-UX によってサポートされるアドレス・スペース・モデル

HP-UX は、WebSphere MQ アプリケーションで利用できるアドレス・スペース・モデルを複数提供しています。

HP-UX は、以下の 2 つのアドレス・スペース・モデルをサポートしています。

- MGAS - Mostly Global Address space (これはデフォルトで、WebSphere MQ によって使用されます)
- MPAS - Mostly Private Address space

WebSphere MQ に接続するアプリケーションは、MGAS または MPAS アドレス・スペース・モデルを使用できます。 MPAS モデルを使用してビルドされ、共用メモリーを使用して WebSphere MQ に接続するアプリケーションでは、多少のパフォーマンス・コストが発生する可能性があります。これは、WebSphere MQ によって使用される共用メモリー・ページを MPAS プログラムの仮想アドレス・スペースにマッピングする際の非効率性が原因です。

Micro Focus Server Express を使用してビルドされる COBOL アプリケーションは、デフォルトでは MPAS モデルを使用します。

chatr プログラムを使用して、プログラムで使用されるアドレッシング・モデルを検査および変更できます。

32 ビット MPAS プログラムから WebSphere MQ への接続で問題が 発生する場合、MGAS アドレッシング・モデルを使用するか、またはご使用のアプリケーションを 32 ビット MPAS アプリケーション ではなく 64 ビット MPAS アプリケーションとしてビルドすることを検討してください。

MGAS および MPAS アドレス・スペース・モデルについて詳しくは、HP-UX の資料を参照してください。

Linux でのアプリケーションの構築

ここでは、実行する WebSphere MQ for Linux アプリケーションの構築時に実行する必要がある追加タスク、および標準タスクに対する変更について説明します。

C および C++ がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

Linux での C プログラムの作成

プリコンパイルされた C プログラムは、 *MQ_INSTALLATION_PATH*/samp/bin ディレクトリーで提供されます。 ソース・コードからサンプルを作成するには、gcc コンパイラーを使用します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

通常の環境で作業してください。 64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

ライブラリーのリンク

Linux で C プログラムを作成するときに必要なライブラリーを以下の表にリストします。

- プログラムを、WebSphere MQ が提供する適切なライブラリーとリンクする必要があります。

  スレッド以外の環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm.so	C 用サーバー
libmqic.so & libmqm.so	C 用クライアント

  スレッド環境では、次のいずれかのライブラリーにリンクしてください。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm_r.so	C 用サーバー
libmqic_r.so & libmqm_r.so	C 用クライアント

注 :

1. インストール可能なサービス (詳細は、管理を参照) を作成する場合は、libmqmzf.so ライブラリーにリンクする必要があります。
2. XA 準拠のトランザクション管理プログラム (IBM TXSeries Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成している場合は、非スレッド・アプリケーションの場合には libmqmxa.so (または、トランザクション管理プログラムが「long」タイプを 64 ビットとして扱う場合は libmqmxa64.so) および libmqz.so ライブラリーにリンクし、スレッド・アプリケーションの場合には libmqmxa_r.so (または libmqmxa64_r.so) および libmqz_r.so ライブラリーにリンクする必要があります。
3. WebSphere MQ ライブラリーは、その他の製品ライブラリーの前にリンクしなければなりません。

31 ビット・アプリケーションの構築

このトピックでは、さまざまな環境で 31 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、31 ビット、非スレッド

```
gcc -m31  -o famqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C クライアント・アプリケーション、31 ビット、スレッド

```
gcc -m31  -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー・アプリケーション、31 ビット、非スレッド

```
gcc -m31  -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C サーバー・アプリケーション、31 ビット、スレッド

```
gcc -m31  -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ クライアント・アプリケーション、31 ビット、非スレッド

```
g++ -m31 -fsigned-char  -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

C++ クライアント・アプリケーション、31 ビット、スレッド

```
g++ -m31 -fsigned-char  -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、31 ビット、非スレッド

```
g++ -m31 -fsigned-char  -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

C++ サーバー・アプリケーション、31 ビット、スレッド

```
g++ -m31 -fsigned-char  -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、31 ビット、非スレッド

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

C クライアント出口、31 ビット、スレッド

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

C サーバー出口、31 ビット、非スレッド

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm
```

C サーバー出口、31 ビット、スレッド

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32_r srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm_r -lpthread
```

32 ビット・アプリケーションの構築

このトピックでは、さまざまな環境で 32 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、32 ビット、非スレッド

```
gcc -m32 -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

C クライアント・アプリケーション、32 ビット、スレッド

```
gcc -m32 -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

C サーバー・アプリケーション、32 ビット、非スレッド

```
gcc -m32 -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C サーバー・アプリケーション、32 ビット、スレッド

```
gcc -m32 -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

C++ クライアント・アプリケーション、32 ビット、非スレッド

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

C++ クライアント・アプリケーション、32 ビット、スレッド

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、32 ビット、非スレッド

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

C++ サーバー・アプリケーション、32 ビット、スレッド

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、32 ビット、非スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

C クライアント出口、32 ビット、スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

C サーバー出口、32 ビット、非スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

C サーバー出口、32 ビット、スレッド

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
lmqm_r -lpthread
```

64 ビット・アプリケーションの構築

このトピックでは、さまざまな環境で 64 ビット・プログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、64 ビット、非スレッド

```
gcc -m64  -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

C クライアント・アプリケーション、64 ビット、スレッド

```
gcc -m64  -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

C サーバー・アプリケーション、64 ビット、非スレッド

```
gcc -m64  -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

C サーバー・アプリケーション、64 ビット、スレッド

```
gcc -m64  -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

C++ クライアント・アプリケーション、64 ビット、非スレッド

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

C++ クライアント・アプリケーション、64 ビット、スレッド

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

C++ サーバー・アプリケーション、64 ビット、非スレッド

```
g++ -m64 -fsigned-char  -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

C++ サーバー・アプリケーション、64 ビット、スレッド

```
g++ -m64 -fsigned-char  -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

C クライアント出口、64 ビット、非スレッド

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

C クライアント出口、64 ビット、スレッド

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64_r cliexit.c
```

```
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

C サーバー出口、64 ビット、非スレッド

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

C サーバー出口、64 ビット、スレッド

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64_r srvexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux での COBOL プログラムの作成

Linux での COBOL プログラムの作成について、および Micro Focus COBOL を使用した COBOL プログラムの作成について説明します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. 32 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   また、シンボリック・リンクは次のディレクトリーに作成されます。

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 64 ビット・プラットフォームでは、64 ビット COBOL コピーブックは次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 以下の例では、COBCPY を次のように設定します。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   (32 ビット・アプリケーションの場合) および

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   (64 ビット・アプリケーションの場合)

プログラムを次のいずれかとリンクする必要があります。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqmcb.so	COBOL 用サーバー
libmqicb.so	COBOL 用クライアント
libmqmcb_r.so	COBOL 用サーバー (スレッド化されたアプリケーションの場合)

ライブラリー・ファイル	プログラム / 出口タイプ
libmqicb_r.so	COBOL 用クライアント (スレッド化されたアプリケーションの場合)

Micro Focus COBOL の使用による COBOL プログラムの作成

プログラムをコンパイルする前に環境変数を次のように設定します。

```
export COBCPY=<COBCPY>
export LIB=MQ_INSTALLATION_PATHlib:$LIB
```

Micro Focus COBOL を使用して、サポートされている場合に 32 ビット COBOL プログラムをコンパイルするには、次のように入力します。

```
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Micro Focus COBOL を使用して 64 ビット COBOL プログラムをコンパイルするには、次のように入力します。

```
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtvP amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

`amqsput` はサンプル・プログラムを示します。

必要な環境変数については、Micro Focus COBOL に関する資料を参照してください。

Solaris でのアプリケーションの構築

ここでは、追加の作業および標準作業に対する変更点について説明します。これらの作業は、Solaris 上で実行する WebSphere MQ for Solaris アプリケーションの構築時に行う必要があります。

COBOL、C、および C++ の各プログラム言語がサポートされています。 C++ プログラムの作成については、C++ の使用を参照してください。

ソース・コードで MQI 呼び出しを記述すると共に、適切な組み込みファイルを追加する必要があります。したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

このトピック全体で、複数行にまたがる長いコマンドの分割に円記号 (¥) が使用されています。 実際に入力する場合は、この文字を使用しないで、各コマンドを 1 行に入力してください。

Solaris での C プログラムの作成

事前にコンパイルした C プログラムは、*MQ_INSTALLATION_PATH*/samp/bin ディレクトリーの中にあります。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

Solaris 用の WebSphere MQ MQI クライアントのみがインストールされたマシン上でプログラムを使用したい場合は、そのプログラムをコンパイルして、クライアント・ライブラリー (`-lmqic`) にリンクします。

サポートされていないコンパイラー ?usr?ucb?cc を使用しても、アプリケーションが正常にコンパイルおよび リンクされることがあります。 しかし、そのアプリケーションの実行時に、キュー・マネージャーに接続しようとすると、アプリケーションは異常終了します。

注 : FIPS 140-2 準拠操作用に構成された 32 ビット Solaris x86 SSL および TLS クライアントでは、 Intel システムでの稼働時に障害が起きます。 この障害は、FIPS 140-2 準拠の GSKit-Crypto Solaris x86 32 ビット・ライブラリー・ファイルが Intel チップ・セットをロードしないことが原因で発生します。 影響を受けたシステムでは、クライアント・エラー・ログにエラー AMQ9655 が記録されます。 この問題を解決するには、FIPS 140-2 準拠を無効にするか、またはクライアント・アプリケーション 64 ビットを再コンパイルします (64 ビット・コードは影響を受けないため)。

ライブラリーのリンク

使用しているアプリケーションのタイプに合った WebSphere MQ ライブラリーとリンクする必要があります。

ライブラリー・ファイル	プログラム / 出口タイプ
libmqm.so	C 用サーバー
libmqic.so & libmqm.so	C 用クライアント

注 :

1. インストール可能なサービス (詳細は、管理を参照) を作成する場合は、`libmqmzf.so` ライブラリーにリンクしてください。
2. XA 準拠のトランザクション管理プログラム (IBM TXSeries Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成している場合は、`libmqmxa.so` (または、トランザクション管理プログラムが「long」タイプを 64 ビットとして扱う場合は `libmqmxa64.so`) および `libmqz.so` ライブラリーにリンクする必要があります。
3. WebSphere MQ ライブラリーは、その他の製品ライブラリーの前にリンクしなければなりません。

x86-64 でのアプリケーションの構築

このトピックでは、x86-64 プラットフォーム上のさまざまな環境でプログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、32 ビット

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C クライアント・アプリケーション、64 ビット

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

C サーバー・アプリケーション、32 ビット

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C サーバー・アプリケーション、64 ビット

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

C++ クライアント・アプリケーション、32 ビット

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

C++ クライアント・アプリケーション、64 ビット

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++ サーバー・アプリケーション、32 ビット

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++ サーバー・アプリケーション、64 ビット

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C クライアント出口、32 ビット

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

C クライアント出口、64 ビット

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C サーバー出口、32 ビット

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

C サーバー出口、64 ビット

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

SPARC でのアプリケーションの構築

このトピックでは、SPARC プラットフォーム上のさまざまな環境でプログラムのビルドに使用されるコマンドの例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

C クライアント・アプリケーション、32 ビット

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

C クライアント・アプリケーション、64 ビット

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C サーバー・アプリケーション、32 ビット

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

C サーバー・アプリケーション、64 ビット

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

C++ クライアント・アプリケーション、32 ビット

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

C++ クライアント・アプリケーション、64 ビット

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

C++ サーバー・アプリケーション、32 ビット

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

C++ サーバー・アプリケーション、64 ビット

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

C クライアント出口、32 ビット

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

C クライアント出口、64 ビット

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

C サーバー出口、32 ビット

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

C サーバー出口、64 ビット

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Solaris での COBOL プログラムの作成

Solaris での COBOL プログラムの作成について説明します。

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

1. 32 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   また、シンボリック・リンクは次のディレクトリーに作成されます。

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 64 ビット COBOL コピーブックは、次のディレクトリーにインストールされます。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. 以下の例では、COBCPY を次のように設定します。

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   (32 ビット・アプリケーションの場合) および

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   (64 ビット・アプリケーションの場合)

Micro Focus コンパイラーを用いてプログラムをコンパイルします。 構造体を宣言するコピー・ファイルは、*MQ_INSTALLATION_PATH*/inc ディレクトリーにあります。次に例を示します。

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

32 ビット・プログラムのコンパイル:

- $ cob32 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqmcb

  COBOL 用サーバー

- $ cob32 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqicb

  COBOL 用クライアント
- $ cob32 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqmcb_r

  COBOL 用スレッド・サーバー
- $ cob32 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib -lmqicb_r

  COBOL 用スレッド・クライアント

64 ビット・プログラムのコンパイル:

- $ cob64 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqmcb

  COBOL 用サーバー
- $ cob64 -xv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqicb

  COBOL 用クライアント
- $ cob64 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqmcb_r

  COBOL 用スレッド・サーバー
- $ cob64 -xtv *amqs0put0.cbl* -L *MQ_INSTALLATION_PATH*/lib64 -lmqicb_r

  COBOL 用スレッド・クライアント

amqs0put0.cbl はサンプル・プログラムを示します。

プログラムを次のいずれかとリンクする必要があります。

- libmqmcb.so

  COBOL 用サーバー
- libmqicb.so

  COBOL 用クライアント

Solaris での CICS プログラムの作成

Solaris での CICS プログラムの作成について説明します。

CICS を WebSphere MQ とリンクできる XA スイッチ・モジュールが提供されます。

表 *67. CICS* アプリケーションの主要コード *(Solaris)*

説明	**C (ソース)**	**C (実行可能)**
XA 初期化ルーチン	amqzscix.c	amqzsc - TXSeries for Solaris

トランザクションは、スレッド・セーフの WebSphere MQ ライブラリー libmqm.so と必ずリンクしてください。

CICS トランザクションのサポートの詳細については、管理を参照してください。

TXSeries CICS サポート

WebSphere MQ for Solaris は、XA インターフェースを使用して TXSeries CICS をサポートします。

C または COBOL のいずれかの同じ CICS 領域にロードされる WebSphere MQ プログラムを作成します。 C および COBOL の MQI 呼び出しを組み合わせて同じ CICS 領域内に入れることはできません。 たいていの場合、使用される 2 番目の言語の MQI 呼び出しは失敗し、理由コード MQRC_HOBJ_ERROR が出されます。

Micro Focus COBOL の使用による CICS COBOL プログラムの作成

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

Micro Focus COBOL を使用する場合は、次のステップに従ってください。

1. 次のコマンドを使用して、実行時ライブラリーに WebSphere MQ の COBOL 実行時ライブラリー・モジュールを追加する。

```
cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
```

   注: `cicsmkcobol` を使用すると、WebSphere MQ では COBOL アプリケーションから C プログラミング言語の MQI 呼び出しを行うことはできません。

   既存のアプリケーションにこの種の呼び出しがある場合は、これらの関数を COBOL アプリケーションから `myMQ.so` などの独自のライブラリーに移動してください。 これらの機能を移動した後は、 CICS 用の COBOL アプリケーションをビルドするときに、 WebSphere MQ ライブラリー `libmqmcbrt.o` は組み込まれません。

   さらに、COBOL アプリケーションで COBOL MQI 呼び出しが行われない場合は、`libmqmz_r` と `cicsmkcobol` をリンクしないでください。

   これにより、Micro Focus COBOL 言語メソッド・ファイルが作成され、CICS の実行時 COBOL ライブラリーが WebSphere MQ (UNIX and Linux システム用) を呼び出すことができるようになります。

   注: `cicsmkcobol` は、以下の製品のいずれかをインストールする場合にのみ実行してください。

   - Micro Focus COBOL の新規バージョンまたは新規リリース
   - TXSeries for Solaris の新規バージョンまたは新規リリース
   - サポートされているデータベース製品の新規バージョンまたは新規リリース (COBOL トランザクション専用)
   - WebSphere MQ の新規バージョンまたは新規リリース
2. 次の環境変数をエクスポートする。

```
COBCPY=MQ_INSTALLATION_PATH/inc  export COBCPY
```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l COBOL -e <yourprog>.ccp
```

CICS C プログラムの作成

CICS の標準機能を使用して、次のステップに従って CICS C プログラムを作成してください。

1. 次の環境変数の**いずれか** をエクスポートする。
   - LDFLAGS = "-L*MQ_INSTALLATION_PATH*lib -lmqm_r" export LDFLAGS
   - USERLIB = "-L*MQ_INSTALLATION_PATH*lib -lmqm_r" export USERLIB
2. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l C amqscic0.ccs
```

CICS C サンプル・トランザクション

CICS 用の WebSphere MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。 トランザクションは、デフォルトのキュー・マネージャーの伝送キュー

SYSTEM.SAMPLE.CICS.WORKQUEUE からメッセージを読み取り、メッセージの伝送ヘッダーに名前があるローカル・キューにそれらを入れます。 エラーはすべて、キュー SYSTEM.SAMPLE.CICS.DLQ に送信されます。 サンプル MQSC スクリプト AMQSCIC0.TST を使用して、これらのキューやサンプル入力キューを作成します。

Windows システムでのアプリケーションの構築

Windows システムの資料では、作成したプログラムから実行可能なアプリケーションを構築する方法について説明しています。

このトピックでは、Windows システムで実行する WebSphere MQ for Windows アプリケーションを構築する際に実行する必要がある追加タスクと、標準タスクに対する変更について説明します。 ActiveX、C、C++、COBOL、および Visual Basic プログラム言語がサポートされています。 ActiveX プログラムの作成については、コンポーネント・オブジェクト・モデル・インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用を参照してください。 C++ プログラムの作成については、C++ の使用を参照してください。

WebSphere MQ for Windows を使用して実行可能なアプリケーションを作成するときに必要な作業は、ソース・コードを作成するプログラム言語により異なります。 ソース・コードで MQI 呼び出しをコーディングすると共に、使用する言語用の WebSphere MQ for Windows 組み込みファイルを組み込むために、適切な言語ステートメントを追加する必要があります。 したがって、これらのファイルの内容をよく理解しておいてください。 詳しくは、80 ページの『IBM WebSphere MQ データ定義ファイル』を参照してください。

Windows での 64 ビット・アプリケーションの構築

32 ビット・アプリケーションと 64 ビット・アプリケーションの両方が、IBM WebSphere MQ for Windows Version 7.5 でサポートされています。 32 ビット形式と 64 ビット形式の両方の IBM WebSphere MQ 実行可能ファイルとライブラリー・ファイルが提供されているので、処理しているアプリケーションに応じて該当するバージョンを使用してください。

実行可能ファイルとライブラリー

32 ビット・バージョンと 64 ビット・バージョンの IBM WebSphere MQ ライブラリーは、以下の場所で提供されています。

表 *68. IBM WebSphere MQ* ライブラリーの場所

ライブラリーのバージョン	ライブラリー・ファイルを含むディレクトリー
32 ビット	*MQ_INSTALLATION_PATH*\Tools\Lib
64 ビット	*MQ_INSTALLATION_PATH*\Tools\Lib64

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

32 ビット・アプリケーションは、マイグレーション後も引き続き正常に機能します。 32 ビット・ファイルは、以前のバージョンの製品と同じディレクトリーにあります。

64 ビット・バージョンを作成する場合は、ご使用の環境が *MQ_INSTALLATION_PATH*\Tools\Lib64 のライブラリー・ファイルを使用するように構成されていることを確認する必要があります。 LIB 環境変数が、32 ビット・ライブラリーを含むフォルダーを参照するように設定されていないことを確認してください。

Windows での C プログラムの作成

通常の Windows 環境で作業してください。WebSphere MQ for Windows には特別な環境は必要ありません。

64 ビット・アプリケーションのプログラミングの詳細については、64 ビット・プラットフォームでのコーディング標準を参照してください。

- プログラムを、WebSphere MQ が提供する適切なライブラリーとリンクしてください。

  ライブラリー・ファイル　プログラム / 出口タイプ

  MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib　32 ビット C 用サーバー

  MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib　32 ビット C 用クライアント

  MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib　32 ビット C 用クライアント、トランザクション調整あり

  MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib　64 ビット C 用サーバー

  MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib　64 ビット C 用クライアント

  MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib　64 ビット C 用クライアント、トランザクション調整あり

  MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

  次のコマンドは、amqsget0 サンプル・プログラムをコンパイルする例を示しています (Microsoft Visual C++ コンパイラーを使用)。

  32 ビット・アプリケーションの場合:

  ```
  cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
  ```

  64 ビット・アプリケーションの場合:

  ```
  cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
  ```

  注:

  – インストール可能なサービス (詳細は、管理を参照) を作成する場合は、mqmzf.lib ライブラリーにリンクする必要があります。
  – XA 準拠のトランザクション管理プログラム (IBM TXSeries、Encina、BEA Tuxedo など) による外部調整用のアプリケーションを作成している場合は mqmxa.lib または mqmxa.lib ライブラリーにリンクする必要があります。
  – CICS 出口を作成している場合は、mqmcics4.lib ライブラリーにリンクしてください。
  – WebSphere MQ ライブラリーは、その他の製品ライブラリーの前にリンクしなければなりません。
- DLL は指定したパス (PATH) 内になければなりません。
- 小文字を使用できるところでは小文字を使用すれば、WebSphere MQ for Windows から WebSphere MQ (UNIX and Linux システム用) に移行できます。UNIX では、小文字を使用する必要があります。

CICS および Transaction Server プログラムの作成

CICS 用の WebSphere MQ トランザクションのサンプル C ソースは、AMQSCIC0.CCS により提供されます。これは、CICS の標準機能を使用して作成します。 例えば、TXSeries for Windows 2000 の場合は、以下のようにします。

1. 環境変数を設定する (次のコードを 1 行で入力する)。

```
set CICS_IBMC_FLAGS=-IMQ_INSTALLATION_PATH\Tools\C\Include;
%CICS_IBMC_FLAGS%
```

2. USERLIB 環境変数を設定する。

```
set USERLIB=MQM.LIB;%USERLIB%
```

3. 次のように入力して、サンプル・プログラムの変換、コンパイル、およびリンクを行う。

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

これについては、「*Transaction Server for Windows NT Application Programming Guide (CICS) V4*」に記載されています。

CICS トランザクションのサポートの詳細については、管理を参照してください。

Windows での COBOL プログラムの作成

この情報を使用して、Windows での COBOL プログラムの作成、および CICS および Transaction Server プログラムの作成について学習します。

1. 32 ビット COBOL コピーブックは、ディレクトリー *MQ_INSTALLATION_PATH*\Tools\cobol\CopyBook にインストールされます。
2. 64 ビット COBOL コピーブックは、ディレクトリー *MQ_INSTALLATION_PATH*\Tools\cobol\CopyBook64 にインストールされます。
3. 以下の例では、CopyBook を次のように設定します。

```
CopyBook
```

   (32 ビット・アプリケーションの場合) および

```
CopyBook64
```

   (64 ビット・アプリケーションの場合)

MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

Windows システムで COBOL プログラムを作成するには、IBM WebSphere MQ で提供される次のいずれかのライブラリーにプログラムをリンクしてください。

ライブラリー・ファイル	プログラムまたは出口タイプ
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb	IBM 用 32 ビット・サーバー
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb	Micro Focus COBOL 用 32 ビット・サーバー
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb	IBM COBOL 用 32 ビット・クライアント
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb	Micro Focus COBOL 用 32 ビット・クライアント

ライブラリー・ファイル	プログラムまたは出口タイプ
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb	IBM 用 64 ビット・サーバー
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb	Micro Focus COBOL 用 64 ビット・サーバー
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb	IBM COBOL 用 64 ビット・クライアント
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb	Micro Focus COBOL 用 64 ビット・クライアント

MQI クライアント環境でプログラムを実行するときは、DOSCALLS ライブラリーが COBOL ライブラリーまたは IBM WebSphere MQ ライブラリーより前に来るようにしてください。

作成するプログラムにより、IBM COBOL SET コンパイラーまたは Micro Focus COBOL コンパイラーを使用できます (以下参照)。

- amqi で始まるプログラムの場合には IBM COBOL SET コンパイラーを使用します。
- amqm で始まるプログラムの場合には Micro Focus COBOL コンパイラーを使用します。および
- amq0 で始まるプログラムの場合にはどちらのコンパイラーを使用しても構いません。

IBM および Micro Focus COBOL

mqmcb.lib または mqiccb.lib のいずれかを使用して、既存の 32 ビット IBM WebSphere MQ Micro Focus COBOL プログラムを再リンクします。mqmcbb ライブラリーや mqiccbb ライブラリーは使用しません。

例えば、 IBM VisualAge COBOL を使用してサンプル・プログラム amq0put0 をコンパイルするには、次のようにします。

1. IBM WebSphere MQ VisualAge COBOL コピーブックへのパスを指定した SYSLIB 環境変数を設定する (次のコードを 1 行で入力する)。

```
set SYSLIB=MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook\VAcobol;%SYSLIB%
```

2. IBM WebSphere MQ サーバー上で使用する場合:

```
cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
Tools\Lib\mqmcbb.lib"
```

3. IBM WebSphere MQ クライアント上で使用する場合:

```
cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
Tools\Lib\mqiccbb.lib"
```

注: コンパイラー・オプション CALLINT(SYSTEM) を使用する必要がありますが、これは、cob2 のデフォルト値になっています。

例えば、Micro Focus COBOL を使用して amq0put0 というサンプル・プログラムをコンパイルするには、以下のステップに従ってください。

1. COBCPY 環境変数を設定して、宛先を IBM WebSphere MQ COBOL コピーブックに指定する (次のコードを 1 行で入力する)。

```
set COBCPY=MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. プログラムをコンパイルしてオブジェクト・ファイルを作成する。

```
cobol amq0put0 LITLINK
```

3. オブジェクト・ファイルを実行時システムにリンクする。
   - LIB 環境変数を設定して、宛先をコンパイラー COBOL ライブラリーに指定する。
   - IBM WebSphere MQ サーバーで使用する場合には、以下の記述形式によりオブジェクト・ファイルをリンクする。

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - または、IBM WebSphere MQ クライアントで使用する場合には、以下の記述形式によりオブジェクト・ファイルをリンクする。

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

CICS および Transaction Server プログラムの作成

IBM VisualAge COBOL を使用して TXSeries for Windows NT V5.1 プログラムをコンパイルおよびリンクするには、以下のようにします。

1. 環境変数を設定する (次のコードを 1 行で入力する)。

   ```
   set CICS_IBMCOB_FLAGS=MQ_INSTALLATION_PATH\
   Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
   ```

2. USERLIB 環境変数を設定する。

   ```
   set USERLIB=MQMCBB.LIB
   ```

3. 次のように入力して、プログラムの変換、コンパイル、およびリンクを行う。

   ```
   cicstcl -l IBMCOB myprog.ccp
   ```

詳細については、「*Transaction Server for Windows NT, V4 Application Programming Guide*」を参照してください。

Micro Focus COBOL を使用して CICS for Windows V5 プログラムをコンパイルし、リンクするには、以下のステップに従ってください。

- INCLUDE 変数を設定する。

  ```
  set
  INCLUDE=<drive>:\<programname>\ibm\websphere\tools\c\include;
          <drive>:\opt\cics\include;%INCLUDE%
  ```

- COBCPY 環境変数を設定する。

  ```
   setCOBCPY=<drive>:\<programname>\ibm\websphere\tools\cobol\copybook;
             <drive>:\opt\cics\include
  ```

- COBOL オプションを設定する。
  - set
  - COBOPTS=/LITLINK /NOTRUNC

そして、以下のコードを実行します。

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
```

```
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows での Visual Basic プログラムの作成

Windows で Visual Basic プログラムを使用する際は、以下の情報を参照してください。

注 : 64 ビット・バージョンの Visual Basic モジュール・ファイルは提供されていません。

Windows 上で、Visual Basic プログラムを作成する手順は、次のとおりです。

1. 新規プロジェクトを作成する。
2. 製品に付属のモジュール・ファイル CMQB.BAS をプロジェクトに追加する。
3. 必要に応じて、次のモジュール・ファイルを追加する。

CMQBB.BAS	MQAI サポート
CMQCFB.BAS	PCF サポート
CMQXB.BAS	チャネル出口サポート
CMQPSB.BAS	パブリッシュ/サブスクライブ

Visual Basic 内から MQCONNXAny 呼び出しを使用する方法について詳しくは、87 ページの『Visual Basic によるコーディング』を参照してください。

プロジェクト・コードでは、MQI 呼び出しの前に、必ずプロシージャー MQ_SETDEFAULTS を呼び出す必要があります。 このプロシージャーにより、MQI 呼び出しに必要なデフォルトの構造体がセットアップされます。

プロジェクトをコンパイルまたは実行する前に、条件付きコンパイル変数 *MqType* を設定して、WebSphere MQ サーバーまたはクライアントを作成するかどうかを指定します。 Visual Basic プロジェクト内の *MqType* の値を次のように設定します。サーバーの場合は 1 で、クライアントの場合は 2 です。

1.「Project (プロジェクト)」メニューを選択する。
2.「*Name* プロパティー」(*Name* は現行プロジェクト名) を選択する。
3. ダイアログ・ボックスで「Make (作成)」タブを選択する。
4. サーバーの場合は、「Conditional Compilation Arguments (条件付きコンパイル引数)」フィールドで次のように入力する。

```
MqType=1
```

クライアントの場合は、次のように入力する。

```
MqType=2
```

SSPI セキュリティー出口

WebSphere MQ for Windows には、WebSphere MQ MQI クライアント用および WebSphere MQ サーバー用のセキュリティー出口が備えられています。 これはチャネル出口プログラムであり、セキュリティー・サービス・プログラミング・インターフェース (SSPI) を使用することによって、WebSphere MQ チャネルの認証を行うことができます。 SSPI には、Windows システムの統合セキュリティー機能が備えられています。

セキュリティー・パッケージは、security.dll または secur32.dll のいずれかからロードします。 これらの DLL は、ご使用のオペレーティング・システム (OS) で提供されています。

片方向認証は、NTLM 認証サービスを使用して備えられています。 双方向認証は Kerberos 認証サービスを使用して備えられています。

セキュリティー出口プログラムは、ソースおよびオブジェクト形式で提供されます。 オブジェクト・コードをそのまま使用することもできますし、あるいはソース・コードを開始点として使用して独自のユーザー出口プログラムを作成することもできます。

168 ページの『Windows システムでの SSPI セキュリティー出口の使用』も参照してください。

セキュリティー出口の概要

セキュリティー出口は、2 つのセキュリティー出口プログラムの間のセキュア接続を形成します。このうち、1 つのプログラムはメッセージ・チャネル・エージェント (MCA) を送信するためのもので、もう 1 つは MCA を受信するためのものです。

セキュア接続を開始する方のプログラム、つまり、MCA セッションが確立された後に制御を得る最初のプログラムは、コンテキスト・イニシエーター と呼ばれます。 これのパートナーとなるプログラムは、コンテキスト・アクセプター と呼ばれます。

以下の表は、チャネル・タイプ (コンテキスト・イニシエーターと、それに関連したコンテキスト・アクセプター) の一例を示しています。

表 69. コンテキスト・イニシエーターとそれに関連したコンテキスト・アクセプター

コンテキスト・イニシエーター	コンテキスト・アクセプター
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	MQCHT_SENDER
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

セキュリティー出口プログラムには以下の 2 つの入り口点があります。

- **SCY_NTLM**

  NTLM 認証サービス (片方向認証を実行する) を使用します。 NTLM を使用すると、サーバーはクライアントの ID を検証することができます。 クライアントがサーバーの ID を検証したり、1 つのサーバーが他のサーバーを検証したりすることはできません。 NTLM 認証は、サーバーの真正を前提とするネットワーク環境用に設計されたものです。

- **SCY_KERBEROS**

  これは Kerberos 相互認証サービスを使用します。 Kerberos プロトコルは、ネットワーク環境内のサーバーが本物であることを前提としません。 ネットワーク接続の両側のパーティーが他のパーティーの ID を検証できます。 つまり、サーバーはクライアントや他のサーバーの ID を検証でき、クライアントはサーバーの ID を検証できるということです。

セキュリティー出口の実行内容

このトピックでは、SSPI チャネル出口プログラムの実行内容について説明します。

提供されているチャネル出口プログラムは、セッションの確立時に、パートナー・システムの片方向認証か双方向 (相互) 認証のいずれかを提供します。 特定のチャネルに関して、各出口プログラムには関連したプリンシパル があります (ユーザー ID と同様。468 ページの『WebSphere MQ のアクセス制御と Windows プリンシパル』を参照してください)。 2 つの出口プログラム間の接続は、2 つのプリンシパル間の関連とも言うことができます。

基本となるセッションが確立された後、2 つのセキュリティー出口プログラム (1 つは MCA の送信用、もう 1 つは MCA の受信用) の間のセキュア接続が確立されます。 以下の順序で操作が行われます。

1. 各プログラムは、明示的なログイン操作などによって、特定のプリンシパルと関連付けられます。

2. コンテキスト・イニシエーターは、セキュリティー・パッケージのパートナー (Kerberos の場合は名前つきパートナー) とのセキュア接続を要求し、トークン (token1 と呼ばれる) を受け取ります。 既に確立されている基本セッションを使用して、トークンがパートナー・プログラムに送信されます。
3. パートナー・プログラム (コンテキスト・アクセプター) は token1 をセキュリティー・パッケージに渡します。これにより、コンテキスト・イニシエーターが本物かどうか検査されます。 NTLM の場合、接続はこれで確立されます。
4. Kerberos が提供するセキュリティー出口 (つまり、相互認証) の場合、セキュリティー・パッケージは 2 つ目のトークン (token2 と呼ばれる) を生成します。コンテキスト・アクセプターは基本セッションを使用することによってこのトークンをコンテキスト・イニシエーターに戻します。
5. コンテキスト・イニシエーターは token2 を使用して、コンテキスト・アクセプターが本物かどうか検査します。
6. この段階で両方のアプリケーションにおいてパートナーの真正が証明されると、セキュア (認証済み) 接続が確立されます。

WebSphere MQ のアクセス制御と Windows プリンシパル

WebSphere MQ が提供するアクセス制御は、ユーザーとグループを基にします。 Windows が提供する認証は、ユーザーや servicePrincipalName (SPN) などの、プリンシパルを基にします。 servicePrincipalName の場合、1 つのユーザーに関して多数が関連付けられる場合があります。

SSPI セキュリティー出口では、認証の際に、関連した Windows プリンシパルが使用されます。 Windows の認証が成功すると、出口は Windows プリンシパルに関連したユーザー ID を WebSphere MQ に渡し、アクセス制御が行えるようにします。

認証に関連する Windows プリンシパルは、使用される認証タイプによって異なります。

- NTLM 認証の場合、コンテキスト・イニシエーター用の Windows プリンシパルは、実行中のプロセスに関連したユーザー ID になります。 この認証は片方向なので、コンテキスト・アクセプターに関連したプリンシパルは関係ありません。
- Kerberos 認証の場合の CLNTCONN チャネルでは、Windows プリンシパルは、実行中のプロセスに関連したユーザー ID になります。 それ以外のチャネルでは、Windows プリンシパルは、QueueManagerName に次の接頭部を追加して形成される servicePrincipalName になります。

```
ibmMQSeries/
```

WebSphere MQ for Windows での LDAP サービスの使用

このトピックでは、ディレクトリー・サービスとはどのようなものかを説明し、ディレクトリー・アクセス・プロトコル (DAP) の役割について説明します。 また、WebSphere MQ アプリケーションで Lightweight Directory Access Protocol (LDAP) ディレクトリーを使用する方法についても、サンプル・プログラムをガイドとして使用しながら説明します。

注 : サンプル・プログラムは、既に LDAP をよく理解しているユーザー向けに設計されています。

以下のトピックには、ディレクトリー・サービス、LDAP、および WebSphere MQ での LDAP の使用について詳しい情報が記述されています。

- 468 ページの『ディレクトリー・サービス』
- 469 ページの『Lightweight Directory Access Protocol (LDAP)』
- 469 ページの『WebSphere MQ と LDAP の併用』

ディレクトリー・サービス

ディレクトリーとはオブジェクトに関する情報を格納する場所であり、特定のオブジェクトに関する情報を検索しやすいように編成されています。

一般的な例としては、人や会社に関する情報 (住所と電話番号) が格納されている電話帳があります。 もう 1 つの例は電子メール・システムの住所録で、ここにはさまざまな人の電子メール・アドレスや、場合によっては電話番号など、その他の情報も格納されています。

コンピューター・システムの場合は、ディレクトリーにプリンターや共用ディスクなど、コンピューター・リソースに関する情報を格納できます。 例えば、ディレクトリーを使用して、最も近くにあるカラー・プリンターの設置場所を知ることもできます。 WebSphere MQ アプリケーションでディレクトリーを使用すると、アプリケーション・サービス (例えば、売掛金の処理など) と、そのサービスを必要とするメッセージに使用されるキュー (キュー名とそのホストであるキュー・マネージャーの名前によって識別されます) を関連付けることができます。

ディレクトリーはクライアント / サーバー・システムとして実装され、ディレクトリー・サーバーは、すべての情報を保持してクライアントからの要求に答えます。 クライアントは、情報をユーザーに直接提供するユーザー・インターフェース・プログラムとするか、作業を実行するためにリソースを見付ける必要があるアプリケーション・プログラムとすることができます。 ディレクトリー・サービスは、ディレクトリー・サーバー、管理プログラム、およびディレクトリーの構成、更新、読み取りに必要なクライアント・ライブラリーとクライアント・プログラムからなっています。

Lightweight Directory Access Protocol (LDAP)

多数のディレクトリー・サービス (例えば、Novell Directory Services、DCE Cell Directory Service、Banyan StreetTalk、Windows Directory Services、X.500 など) と、電子メール製品に関連した住所録サービスが存在します。 X.500 は、国際標準化機構 (ISO) によってグローバル・ディレクトリー・サービスの標準として提案されたものです。 X.500 は通信に OSI プロトコル・スタックを必要とし、大部分はそれが原因で、大きな組織や学術機関に使用が限られています。 X.500 ディレクトリー・サーバーは、ディレクトリー・アクセス・プロトコル (DAP) を使用してクライアントと通信します。

LDAP (Lightweight Directory Access Protocol) は、DAP の簡易バージョンとして作成されました。 LDAP は実装が簡単で、あまり使用されない DAP のいくつかの機能が省略されており、TCP/IP 上で動作します。 これらの変更の結果として、LDAP は急速に多目的のディレクトリー・アクセス・プロトコルとして採用され始め、以前に使用されていた多数の専用プロトコルにとって代わりつつあります。 LDAP クライアントはゲートウェイを介して X.500 サーバーにアクセスする (X.500 には依然として OSI プロトコル・スタックが必要です) ことができますが、通常の X.500 の実装にも DAP アクセスだけでなく LDAP のネイティブ・サポートが組み込まれる場合が増えてきています。

LDAP ディレクトリーは、内容に効率よくアクセスできるよう分散することができ、複製を使用できます。

LDAP について詳しくは、「*Understanding LDAP*」(IBM Redbooks® 資料) を参照してください。

WebSphere MQ と LDAP の併用

WebSphere MQ 構成では、メッセージ・キューと伝送キューを定義した情報はローカルに保管されます。したがって、WebSphere MQ ネットワークにはさまざまな定義が分散されており、その情報をブラウズするために使用できる中央ディレクトリーは存在しません。 WebSphere MQ アプリケーション間のリモート・メッセージ交換は、通常、リモート・キューのローカル定義を使用して行われます。 アプリケーションは、最初にリモート・キューのローカル定義の中で指定されている名前を使用して MQOPEN 呼び出しを発行します。 次に、リモート・キューにメッセージを書き込むため、アプリケーションは MQOPEN 呼び出しから戻されたハンドルを指定して MQPUT を発行します。 リモート・キュー定義は、宛先キューの名前、宛先キュー・マネージャー、およびオプションとして伝送キューを提供します。 この手法では、アプリケーションはローカル・キュー定義の中で指定されている名前を実行時に知っている必要があります。

上記を変更すると、リモート・キューのローカル定義は使用されません。 アプリケーションは、MQOPEN の一部としてのリモート・キュー・マネージャー名も含め、宛先キューの完全な名前を指定できます。 したがって、アプリケーションはそれら 2 つの名前を実行時に知っている必要があります。 ローカル・キュー定義を使用して、また、適切な名前が付いた (またはデフォルトの) 伝送キューとそれに関連したターゲットへの配信用チャネルを使用して、ローカル・キュー・マネージャーが正しく構成されていなければなりません。

ソースとターゲットの両方のキュー・マネージャーが同じクラスターのメンバーとして定義されている場合は、上記の 2 つのシナリオにおける伝送キューとチャネルの局面は無視できます。 ターゲットの伝送キ

ューがクラスター・キューである場合は、リモート・キューのローカル定義も不要です。 ただし、前述の場合と同様に、アプリケーションはやはり宛先キューの名前を知っていなければなりません。

このようなアプリケーションのキュー名 (あるいはキューとキュー・マネージャー名の組み合わせ) への依存性は、ディレクトリー・サービスを使用して除去することができます。 アプリケーションの基準と WebSphere MQ オブジェクト名の間のマッピングは、ディレクトリーの中に保持することができ、アプリケーションに依存せずに動的に更新できます。 実行時に、メッセージを送信しようとする WebSphere MQ アプリケーションは、最初にアプリケーション・ベースの基準を使用してディレクトリーに照会します。 例えば、service_name = "accounts receivable" によって、関連する WebSphere MQ オブジェクト名を取り出し、次に、戻ってきた値を MQOPEN 呼び出しに使用します。

もう 1 つのディレクトリーの使用例は、多数の小さな倉庫やオフィスを持つ会社の場合で、WebSphere MQ MQI クライアントを使用すると、大きなオフィスに置かれている WebSphere MQ サーバーにメッセージを送信できます。 クライアントはホスト・マシンの名前、MQI チャネルのほか、メッセージの送信先である各サーバーのキュー名を知っている必要があります。 WebSphere MQ サーバーを別のマシンへ移動しなければならない場合もあり、そのような場合、そのサーバーを使用して通信しているすべてのクライアントは、その変更を知る必要があります。 LDAP ディレクトリー・サービスを使用すれば、ホスト・マシンの名前 (およびチャネル名とキュー名) を保管でき、クライアント・プログラムはサーバーにメッセージを送信したいときに、そのディレクトリーから情報を取り出すことができます。 その場合、ホスト名 (またはチャネル名かキュー名) が変更されても、ディレクトリーだけを更新すれば済みます。

あるアプリケーション・メッセージの複数の宛先を 1 つのディレクトリーに保管することもでき、どのディレクトリーを選択するかは、可用性または負荷の共有を考慮して決めることができます。

WebSphere MQ は、LDAP ディレクトリーを使って Secure Sockets Layer (SSL) で使用する認証情報を保管することもできます。 WebSphere MQ classes for Java は、LDAP ディレクトリーに情報を保管することもできます。

LDAP サンプル・プログラム

このサンプル・プログラムは、LDAP をよく理解していて、既に使用していると思われるユーザー向けに設計されています。 このプログラムは、どのようにして WebSphere MQ アプリケーションから LDAP ディレクトリーを使用できるかを示したサンプルです。

サンプル・プログラムの作成

このプログラムの作成とテストは、TCP/IP を使用した Windows でのみ行われました。 461 ページの『Windows での C プログラムの作成』に述べた一般的な考慮事項のほか、以下の点にも注意してください。

- このプログラムはクライアント・プログラムとして実行されるように設計されています。このため、MQIC.LIB ライブラリーを使用してリンクしてください。
- このプログラムは、WebSphere MQ のヘッダー・ファイルとライブラリーのほか、LDAP クライアントのヘッダー・ファイルとライブラリーを使用して作成する必要があります。

  例えば、IBM eNetwork クライアントを使用し、このプログラムを LIBLDAPSTATICE.LIB と LIBLBERSTATICSSL.LIB のライブラリーを使用してリンクしてください。

ディレクトリーの構成

サンプル・プログラムを実行するには、事前にサンプル・データを使用して LDAP ディレクトリー・サーバーを構成しておかなければなりません。

`tools\c\samples` ディレクトリー内の MQuser.ldif ファイルには、LDIF (LDAP データ交換形式) のサンプル・データが含まれています。 このファイルは、必要に応じて編集することができます。 このファイルには、3 つのオフィスからなる輸送部門を持つ MQuser という架空の会社に関するデータが入っています。 それぞれのオフィスには、WebSphere MQ サーバーを実行するマシンが 1 台ずつあります。

最低でも、ユーザーは WebSphere MQ サーバーを実行するマシンのホスト名が入っている 3 行を編集する必要があります。それらの行は 18、27、および 36 行目です。

```
host: LondonHost
  ...
host: SydneyHost
```

```
  ...
host: WashingtonHost
```

LondonHost、SydneyHost、および WashingtonHost を、WebSphere MQ サーバーを実行する 3 台のマシンの名前に変更してください。 また、チャネル名とキュー名を変更したければ、変更できます (サンプルでは、システム・デフォルトの名前を使用します)。 サンプル・データ内のオフィスの数を増減してもかまいません。

IBM Tivoli ディレクトリー・サーバーの構成

ディレクトリーのインストール方法について詳しくは、「IBM Tivoli Directory Server (ITDS) Administrator's Guide」を参照してください。 Installing and Configuring Server のトピックでは、Installing Server と Basic Server Configuration のセクションについて説明します。 必要に応じて、トピック『Administrator Interface』も参照して、インターフェースの機能についてもよく理解してください。

Configuring - How Do I のトピックでは、手順に従って管理者を開始し、セクション Configure Database に従ってデフォルト・データベースを作成します。 セクション Configure replica をスキップし、セクション Work with Suffixes を使用して、サフィックス o=MQuser を追加します。

データベースにエントリーを追加する前に、いくつかの属性定義とオブジェクト・クラス定義を追加することにより、ディレクトリー・スキーマを拡張しなければなりません。 これについては、 「IBM Tivoli Directory Server Administrator's Guide」の『Reference Information』の章のセクション『Directory Schema』に説明があります。 それに役立つ 2 つのサンプル・ファイルが添付されています。 mq.at.conf ファイル に、?etc?slapd.at.conf ファイルに追加しなければならない属性定義が入っています。 これを追加するには、slapd.at.conf を編集して次のような行を追加することにより、サンプル・ファイルをインクルードします。

```
include <pathname>/mq.at.conf
```

あるいは、slapd.at.conf ファイルを編集して、サンプル・ファイルの内容を直接追加してもかまいません。それには、次の行を追加します。

```
# MQ attribute definitions
attribute mqChannel           ces   mqChannel        1000   normal
attribute mqQueueManager      ces   mqQueueManager   1000   normal
attribute mqQueue             ces   mqQueue          1000   normal
attribute mqPort              cis   mqPort           64     normal
```

オブジェクト・クラス定義についても同様に、 etc?slapd.oc.conf を編集して次の行を追加し、サンプル・ファイルをインクルードできます。

```
include <pathname>/mq.oc.conf
```

または、サンプル・ファイルの内容を slapd.oc.conf, に直接追加することもできます。つまり、以下の行を追加します。

```
# MQ object classdefinition
objectclass mqApplication
      requires
              objectClass,
              cn,
              host,
              mqChannel,
              mqQueue
      allows
              mqQueueManager,
              mqPort,
              description,
              l,
              ou,
              seeAlso
```

この時点で、ディレクトリー・サーバーを始動し (「Administration (管理)」、「Server (サーバー)」、「Startup (始動)」)、サンプル・エントリーを追加することができます。 サンプル・エントリーを追加するには、「管理」の、管理者の「エントリーの追加」ページに進み、サンプル・ファイル `MQuser.ldif` の絶対パス名を入力し、「発信」をクリックします。

この時点でディレクトリー・サーバーが実行され、サンプル・プログラムを実行するのに適したデータがロードされます。

Netscape ディレクトリー・サーバーの構成

「Netscape サーバーの管理」ページを使用して、「**新規 Netscape ディレクトリー・サーバーの作成**」をクリックします。

構成情報が入ったフォームが表示されます。 ディレクトリー接尾部を **o=MQuser** に変更し、制限なしユーザーのパスワードを追加します。 その他の情報を、使用しているインストール・システムに合わせて変更してもかまいません。 「**OK**」をクリックすると、ディレクトリーが正常に作成されます。 「**サーバー管理に戻る**」をクリックし、ディレクトリー・サーバーを始動してください。 新しいディレクトリーについてディレクトリー・サーバー管理サーバーを始動するには、ディレクトリー名をクリックします。

データベースにエントリーを追加する前に、いくつかの属性定義とオブジェクト・クラス定義を追加することにより、ディレクトリー・スキーマを拡張してください。 「ディレクトリー・サーバー」ページの「**スキーマ**」タブをクリックしてください。 新しい属性を追加できるフォームが表示されます。 次の属性を追加してください (属性 OID は、すべての属性についてブランクのまま残してください)。

```
Attribute Name              Syntax
--------------              ------
mqChannel                   Case Exact String
mqQueueManager              Case Exact String
mqQueue                     Case Exact String
mqPort                      Integer
```

サイド・パネルで「**オブジェクト・クラスの作成**」をクリックすることにより、新しいオブジェクト・クラスを追加します。 オブジェクト・クラス名として **mqApplication** を入力し、親のオブジェクト・クラスとして **applicationProcess** を選択し、**オブジェクト・クラス OID** をブランクで残してください。 ここで、いくつかの属性をオブジェクト・クラスに追加します。 必須属性として **host**、 **mqChannel**、および **mqQueue** を選択し、許可された属性として **mqQueueManager** と **mqPort** を選択します。 「**新規オブジェクト・クラスの作成**」ボタンをクリックしてオブジェクト・クラスを作成します。

サンプル・データを追加するには、「**データベース管理**」タブをクリックし、サイド・パネルから「**エントリーの追加**」を選択します。 サンプル・データ・ファイル `<pathname>\MQuser.ldif` のパス名を入力し、パスワードを入力して、 「**OK**」をクリックします。

サンプル・プログラムは、未許可ユーザーとして実行され、デフォルトでは、Netscape Directory は未許可ユーザーにディレクトリーの検索を許可しません。 これを変更するには、「**アクセス制御**」タブをクリックしてください。 制限なしユーザーのパスワードを入力し、「**OK**」をクリックしてディレクトリーのアクセス制御エントリーにロードします。 それらのエントリーは、この時点では空になっています。 「**新規 ACI**」ボタンをクリックして新しいアクセス制御エントリーを作成します。 表示された入力ボックスで「**拒否**」 (これには下線が付いています) をクリックし、その結果として表示されるダイアログ・ボックスでそれを「**許可**」に変更します。 名前を、例えば **MQuser-access** として追加し、「**接尾部の選択**」をクリックして **o=MQuser** を選択します。 ターゲットとして **o=MQuser** を入力し、制限なしユーザーのパスワードを入力して、「**発信**」をクリックします。

この時点でディレクトリー・サーバーが実行され、サンプル・プログラムを実行するのに適したデータがロードされます。

サンプル・プログラムの実行

この時点では、LDAP ディレクトリー・サーバーが実行されており、それにはサンプル・データが入っています。 このデータは 3 台のホスト・マシンを指定しており、それらすべてのマシンは WebSphere MQ サーバーを実行していなければなりません。 それぞれのマシンでデフォルトのキュー・マネージャーが実行されていることを確認してください (別のキュー・マネージャーを指定するようサンプル・データを変更した場合は除きます)。

また、各マシン上で WebSphere MQ リスナー・プログラムも開始してください。サンプルは TCP/IP をデフォルトの WebSphere MQ ポート番号で使用しているので、リスナーを開始するには、次のコマンドを使用します。

```
runmqlsr -t tcp
```

サンプルをテストするため、何かのプログラムを実行して各 WebSphere MQ サーバーに着信したメッセージを読み取ってもかまいません。例えば、次のように amqstrg サンプル・プログラムを使用できます。

```
amqstrg SYSTEM.DEFAULT.LOCAL.QUEUE
```

このサンプル・プログラムは 3 つの環境変数を使用し、1 つは必須で 2 つはオプションの変数です。 必須変数は LDAP_BASEDN で、これはディレクトリー検索用のベース識別名を指定します。 サンプル・データを処理するには、これを ou=Transport, o=MQuser に設定してください。 例えば、Windows システムのコマンド・プロンプトに次のように入力します。

```
set LDAP_BASEDN=ou=Transport, o=MQuser
```

オプションの変数は LDAP_HOST と LDAP_VERSION です。 LDAP_HOST 変数は、LDAP サーバーを実行しているホストの名前を指定し、指定されていない場合にはデフォルトでローカル・ホストになります。 LDAP_VERSION 変数は、使用する LDAP プロトコルのバージョンを指定します。2 または 3 のいずれかです。 ほとんどの LDAP サーバーは、バージョン 3 のプロトコルをサポートするようになりましたが、すべての LDAP サーバーは、旧バージョン 2 もサポートしています。 このサンプルは、どちらのバージョンのプロトコルでも同様に機能します。指定しなければ、デフォルトでバージョン 2 になります。

この時点で、プログラム名とそのあとにメッセージの送信先にしたい WebSphere MQ アプリケーションの名前を入力することにより、サンプルを実行できます。このサンプル・データの場合、アプリケーション名は London、Sydney、および Washington です。 例えば、London アプリケーションにメッセージを送信するには、次のように入力します。

```
amqsldpc London
```

プログラムが WebSphere MQ サーバーに正しく接続できなかった場合は、該当するエラー・メッセージが表示されます。 接続に成功すると、メッセージの入力を開始できます。入力した各行 (< return> または < enter> で終了) は別個のメッセージとして送信され、空の行がプログラムを終了します。

プログラム設計

このプログラムには 2 つの部分があります。最初の部分は、環境変数とコマンド行の値を使用して LDAP ディレクトリー・サーバーに照会を行います。2 番目の部分は、ディレクトリーから戻された情報を使用して WebSphere MQ 接続を確立し、メッセージを送信します。

プログラムの最初の部分で使用される LDAP 呼び出しは、LDAP のバージョン 2 と 3 のどちらを使用するかによって少し異なり、それについては、LDAP クライアント・ライブラリーに添付されている文書に詳しい説明があります。 ここでは、要旨だけを説明します。

プログラムの最初の部分では、このプログラムが正しく呼び出されたかどうかを検査し、環境変数を読み取ります。 次に、指定されたホスト上の LDAP ディレクトリー・サーバーとの接続を確立します。

```
if (ldapVersion == LDAP_VERSION3)
{
  if ((ld = ldap_init(ldapHost, LDAP_PORT)) == NULL)
     ...
}
else
{
  if ((ld = ldap_open(ldapHost, LDAP_PORT)) == NULL )
     ...
}
```

接続が確立された時点で、プログラムは「ldap_set_option」呼び出しによってサーバー上のいくつかのオプションを設定し、そのあと、サーバーにバインドすることにより、サーバーに対してプログラム自体の認証を行います。

```
if (ldapVersion == LDAP_VERSION3)
{
  if (ldap_simple_bind_s(ld, bindDN, password) != LDAP_SUCCESS)
     ...
}
else
{
  if (ldap_bind_s(ld, bindDN, password, LDAP_AUTH_SIMPLE) !=
      LDAP_SUCCESS)
     ...
}
```

サンプル・プログラムでは、bindDN と password は NULL に設定されています。これは、プログラムがそれ自体を匿名ユーザーとして証明することを意味しています。つまり、プログラムは特別なアクセス権を持たず、パブリックに使用可能な情報にだけアクセスできます。 実際には、ほとんどの組織がディレクトリーに保管した情報へのアクセスを禁止し、許可を受けたユーザーだけがアクセスできるようにします。

バインド呼び出し ld への第 1 パラメーターはハンドルで、このハンドルは、この特定の LDAP セッションをプログラムの残りの部分全体で識別するために使用されます。 認証のあと、プログラムはディレクトリーからアプリケーション名に一致するエントリーを検索します。

```
rc = ldap_search_s(ld,                /* LDAP Handle             */
                   baseDN,            /* base distinguished name */
                   LDAP_SCOPE_ONELEVEL, /* one-level search       */
                   filterPattern,     /* filter search pattern   */
                   attrs,             /* attributes required     */
                   FALSE,             /* NOT attributes only     */
                   &ldapResult);      /* search result           */
```

これはサーバーへの単純な同期呼び出しであり、サーバーは結果を直接戻します。 これ以外にも、もっと複雑な照会の場合や多数の結果が予期される場合に適した検索のタイプがあります。 検索への第 1 パラメーターはハンドル ld で、これはセッションを識別します。 第 2 パラメーターはベース識別名で、これはディレクトリー内のどの場所から検索を開始するかを指定し、第 3 パラメーターは検索の範囲、つまり、開始点から見てどこまでのエントリーを検索するかを指定します。 これら 2 つのパラメーターが一体となって、ディレクトリー内のどのエントリーに対して検索を行うかを定義します。 次のパラメーター filterPattern は、検索対象を指定します。 attrs パラメーターは、検出したオブジェクトから取得する属性をリストします。 それに続く属性は、属性だけが必要なのか、それとも値も必要なのかを指定し、これを FALSE に設定することは、属性の値も必要であることを意味します。 最後のパラメーターは、結果を戻すために使用されます。

結果には、多数のディレクトリー・エントリーが含まれる場合があり、それぞれのエントリーは指定された属性とその値を伴っています。 したがって、結果から必要な値だけを抜き出す必要があります。 このサンプル・プログラムでは、エントリーを 1 つだけ検出するのが目的なので、結果の中から最初のエントリーだけに注目します。

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

この呼び出しは、最初のエントリーを表すハンドルを戻します。そのエントリーからすべての属性を抽出するため、次のような for ループをセットアップします。

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);
     attribute != NULL;
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))
{
```

これらの属性の 1 つ 1 つについて、それらに関連付けられている値を抽出します。 ここでも、1 つの属性に 1 つの値だけを予期しているので、最初の値だけを使用します。どの属性が存在するかがわかったら、値を適切なプログラム変数に格納します。

```
values = ldap_get_values(ld, ldapEntry, attribute);
if (values != NULL && values[0] != NULL)
{
  if (stricmp(attribute, MQ_HOST_ATTR) == 0)
  {
    mqHost = strdup(values[0]);
    ...
```

最後に、あとをきれいにするためにメモリーを解放し (ldap_value_free、ldap_memfree、ldap_msgfree)、サーバーからアンバインド することによってセッションをクローズします。

```
ldap_unbind(ld);
```

必要なすべての WebSphere MQ 値がディレクトリーから検出されたかどうかを検査し、検出済みならば、sendMessages() を呼び出して WebSphere MQ サーバーに接続して WebSphere MQ メッセージを送信します。

サンプル・プログラムの 2 番目の部分は sendMessages() ルーチンで、このルーチンにはすべての WebSphere MQ 呼び出しが含まれています。 これは amqsput0 サンプル・プログラムをモデルにしており、相違点はプログラムに渡すパラメーターが拡張されていることと、MQCONN 呼び出しの代わりに MQCONNX 呼び出しが使用されていることです。

IBM WebSphere MQ Telemetry 用のアプリケーションの開発

テレメトリー・アプリケーションは、センス装置や制御装置を、インターネット上および企業内の使用可能な他の情報源と統合します。

IBM WebSphere MQ Telemetry のアプリケーションの開発には、設計パターン、工夫された例、サンプル・プログラム、プログラミングの概念、参照情報を使用します。 IBM WebSphere MQ Telemetry デーモン (デバイス用) を使用して、IBM WebSphere MQ へのさまざまな小型デバイスの接続を単純化します。

関連概念

WebSphere MQ Telemetry

モニターと制御に関するテレメトリーの概念とシナリオ

関連タスク

WebSphere MQ Telemetry のインストール

WebSphere MQ Telemetry の管理

WebSphere MQ Telemetry のトラブルシューティング

関連資料

WebSphere MQ Telemetry リファレンス

IBM WebSphere MQ Telemetry サンプル・プログラム

MQ Telemetry Transport v3 クライアント・アプリケーションの基本的な使い方を示すサンプル・スクリプトが提供されています。 スクリプトは、メッセージをパブリッシュし、トピックにサブスクライブするために使用します。

始める前に

テレメトリー (MQXR) サービスを開始し、サンプル・プログラムを実行します。

ユーザー ID は、mqm ユーザー・グループのメンバーでなければなりません。

まず SampleMQM スクリプトを実行し、次に MQTTV3Sample スクリプトを実行して、パブリッシュとサブスクライブを実行します。 SampleMQM スクリプトによって作成されたキュー・マネージャーを削除するには、CleanupMQM サンプル・スクリプトを実行します。

SampleMQM スクリプトは QM1 という名前のキュー・マネージャーを作成して使用するので、QM1 キュー・マネージャーのあるシステムで無変更のまま実行しないでください。 行われた変更によって、既存のキュー・マネージャーの構成に影響を与える可能性があります。

このタスクについて

- SampleMQM アプリケーションは、QM1 という名前のテレメトリー対応のキュー・マネージャーを作成して開始します。 また、スクリプトは QM1 のデフォルトの伝送キューをセットアップし、ポート 1883 でのデフォルトのチャネル・リスニングを作成して開始します。 このチャネルは、それに接続されているクライアントの認証を実行しません。 チャネルのメッセージ・チャネル・エージェント・ユーザー ID (MCAUSER) 属性が「guest」(Windows システムの場合) または「nobody」(Linux システムの場合) に設定されている。 チャネルに接続されているクライアントは、それが実行されているシステムに応じて、「guest」ユーザーまたは「nobody」ユーザーのように扱われます。 このスクリプトは、 Windows システムでは「guest」、 Linux システムでは「nobody」に、 QM1 のすべてのトピックにパブリッシュおよびサブスクライブする権限を与えます。
- MQTTV3Sample アプリケーションは以下の場所にあります。
  - Windows *MQ_INSTALLATION_PATH*\mqxr\samples の場合

    ここで、*MQ_INSTALLATION_PATH* は IBM WebSphere MQ がインストールされている場所です。
  - Linux *MQ_INSTALLATION_PATH*/mqxr/samples の場合

  MQTTV3Sample アプリケーションはパブリッシャーとして機能し、サーバーのトピックに単一メッセージを送信します。 また、サブスクライバーとしても機能し、サーバーからのメッセージを listen します。
- CleanupMQM サンプル・スクリプトは、SampleMQM スクリプトによって作成された QM1 を終了して削除します。 SampleMQM スクリプトを再実行するか QM1 を削除する場合は、CleanupMQM サンプル・スクリプトを使用します。

手順

1. コマンド行に以下のコマンドを入力して SampleMQM スクリプトを実行します。
   - Windows で、SampleMQM スクリプトを実行するためのコマンドは、次のとおりです。

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\SampleMQM.bat
     ```
   - AIX および Linux で、SampleMQM スクリプトを実行するためのコマンドは、次のとおりです。

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

   ここで、*MQ_INSTALLATION_PATH* は IBM WebSphere MQ がインストールされている場所です。

   MQXR_SAMPLE_QM というキュー・マネージャーが作成されます。
2. 以下のコマンドを入力して、MQTTV3Sample スクリプトの最初の部分を実行します。
   - Windows では、1 つのコマンド行に次のコマンドを入力します。

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -a subscribe
     ```
   - AIX および Linux では、1 つのシェル・ウィンドウで以下のコマンドを入力します。

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscribe
     ```
3. 以下のコマンドを入力して、MQTTV3Sample スクリプトの 2 番目の部分を実行します。
   - Windows では、別のコマンド行に次のコマンドを入力します。

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -m "Hello from an MQTT v3
     application"
     ```
   - AIX および Linux では、別のシェル・ウィンドウで次のコマンドを入力します。

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -m "Hello from an MQTT v3 application"
     ```
4. SampleMQM スクリプトによって作成されたキュー・マネージャーを削除するには、以下のコマンドを使用して CleanupMQM スクリプトを実行します。

- Windows では、次のコマンドを入力します。

```
MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat
```

- AIX および Linux の別のシェル・ウィンドウで、以下のコマンドを入力します。

```
MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh
```

タスクの結果

2 番目のウィンドウに入力したメッセージ `Hello from an MQTT v3 application` がそのアプリケーションによってパブリッシュされ、最初のウィンドウのアプリケーションによって受信されます。 最初のウィンドウのアプリケーションは画面にそれを表示します。

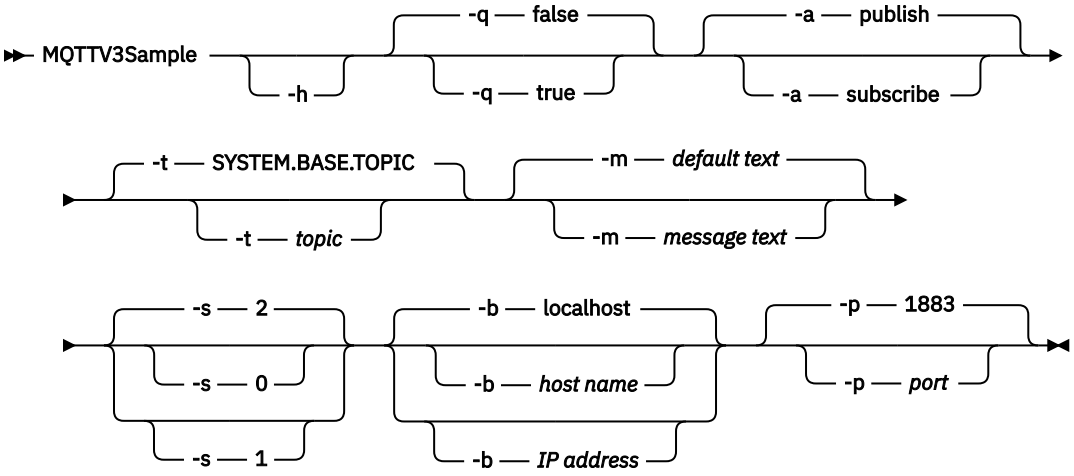
MQTTV3Sample プログラム

MQTTV3Sample プログラムのサンプル構文およびパラメーターに関する参照情報。

目的

MQTTV3Sample プログラムは、メッセージをパブリッシュし、トピックにサブスクライブするために使用できます。

MQTTV3Sample syntax



Parameters

-h
このヘルプ・テキストを出力して終了します。

-q
デフォルト・モードの false を使用する代わりに抑止モードを設定します。

-a
デフォルト・アクションのパブリッシュを想定する代わりにパブリッシュまたはサブスクライブを設定します。

-t
デフォルトのトピックにパブリッシュまたはサブスクライブする代わりに、トピックにパブリッシュまたはサブスクライブします。

-m
デフォルトのパブリケーション・テキスト「Hello from an MQTT v3 application」を送信する代わりに、メッセージ・テキストをパブリッシュします。

-s
デフォルトの QoS 2 を使用する代わりに QoS を設定します。

-b
デフォルトのホスト名の localhost に接続する代わりに、このホスト名または IP アドレスに接続します。

-p
デフォルトの 1883 を使用する代わりに、このポートを使用します。

MQTTV3Sample プログラムの実行

Windows 上で、トピックにサブスクライブするには、次のコマンドを使用します。

```
runMQTTV3Sample -a subscribe
```

Windows 上で、メッセージをパブリッシュするには、次のコマンドを使用します。

```
runMQTTV3Sample
```

提供されているサンプル・スクリプトの実行について詳しくは、475 ページの『IBM WebSphere MQ Telemetry サンプル・プログラム』を参照してください。

Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成

MQTT クライアント・アプリケーションを作成するためのステップを、チュートリアルの形で示します。コードの各行を説明します。 このタスクの終わりには、MQTT パブリッシャーを作成したことになります。WebSphere MQ エクスプローラーを使用して、パブリケーションをブラウズできます。

始める前に

WebSphere MQ Telemetry フィーチャーを、IBM WebSphere MQ Version 7.1 以降がインストールされているサーバーにインストールします。

クライアント・アプリケーションは、 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) の `com.ibm.mq.micro.client.mqttv3` パッケージを使用します。 この SDK は、IBM WebSphere MQ Telemetry インストールの一部です。 クライアントは、IBM WebSphere MQ Telemetry 機能に接続して、IBM WebSphere MQ とメッセージを交換します。

IBM WebSphere MQ Telemetry を管理するには、 IBM WebSphere MQ Explorer Version 7.1 のテレメトリー更新もインストールする必要があります。 更新は、IBM WebSphere MQ Telemetry インストールの一部です。

Java SE で稼働する MQTT クライアントには、Java SE のバージョン 6.0 以降が必要です。 IBM Java SE v6.0 は、 IBM WebSphere MQ Version 7.1 インストールの一部です。 これは *WebSphere MQ installation directory*`\java\jre` にあります。

このタスクについて

この例は、パブリッシュ・アプリケーション PubSync です。 PubSync は、トピック `MQTT Examples` に対して `Hello World` をパブリッシュし、パブリケーションがキュー・マネージャーに送達されたことの確認を待機します。

`MQTT Examples` への永続サブスクリプションを設定することにより、アプリケーションが機能することを確認できます。

この手順では、Eclipse を使用して、クライアントを開発、構築、および実行します。 Eclipse は、Eclipse プロジェクトの Web サイト (www.eclipse.org) からダウンロードできます。

アプリケーションを作成する場合は、Java ファイルを作成し、コマンド行を使用してそれらのファイルをコンパイルおよび実行できます。

新規ディレクトリーに、ディレクトリー・パス .\com\ibm\mq\id を作成します。 Example.java と PubSync.java の 2 つの Java ファイルを作成します。 482 ページの『コード例』 のコードをこれらの Java ファイルにコピーします。

次のコマンドを使用して、Java コードをコンパイルします。

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubSync.java com.ibm.mq.id.Example.java
```

次のコマンドを使用して、PubSync を実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubSync
```

手順

1. Eclipse で Java プロジェクトを作成します。
   a)「**ファイル**」>「**新規**」>「**Java プロジェクト**」 を選択し、プロジェクト名を入力します。 「**次へ**」をクリックします。

   JRE が正しいバージョンまたはそれ以降のバージョンであることを確認してください。 Java SE は 6.0 以降でなければなりません。

   b)「Java 設定」ページで、 「**ライブラリー**」>「**外部 JAR の追加 ...**」 をクリックします。

   c) WebSphere MQ Telemetry SDK フォルダーをインストールしたディレクトリーを参照します。 SDK\clients\java フォルダーを見つけ、すべての .jar「ファイル」>「」>「**開く**」>「**終了**」を選択します。
2. MQTT クライアント Javadoc をインストールします。

   MQTT クライアント Javadoc がインストールされると、Java エディターは MQTT v3 クラスによるアシスタンスを提供します。

   a) Java プロジェクトで、 「**パッケージ・エクスプローラー**」>「**参照されるライブラリー**」を開きます。 com.ibm.micro.client.mqttv3.jar を右クリックし、「**プロパティー**」をクリックします。

   b) プロパティー・ナビゲーターで、「**Javadoc ロケーション**」をクリックします。

   c)「Javadoc ロケーション」ページで、「**Javadoc URL**」>「**参照 ...**」をクリックします。 次に、*WMQ Installation directory*\mqxr\SDK\clients\java\doc\javadoc フォルダー >「**OK**」を見つけます。

   d)「**検証...**」>「**OK**」をクリックします。

   ドキュメンテーションを表示するためにブラウザーを開くよう促すプロンプトが出されます。
3. Java クラス・ウィザードを使用して、クラス PubSync を作成します。

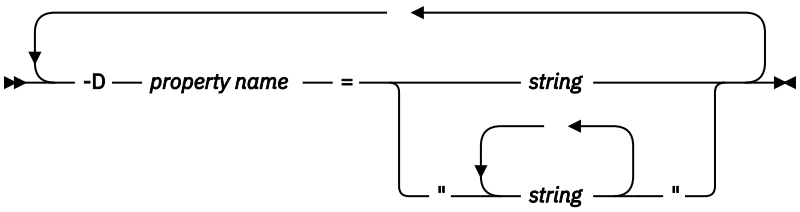
   a) 作成した Java プロジェクトを右クリックし、 「**新規**」 >「**クラス**」をクリックします。

   b) パッケージ名 com.ibm.mq.id を入力します。

   c) クラス名 PubSync

   d) メソッド・スタブ・ボックス、**public static void main(String [] args)**
4. ファイル、Example.java をパッケージ com.ibm.mq.id に作成します。 484 ページの図 89 のコードをこのファイルにコピーします。

   これらの例で使用されているパラメーターはすべて、プロパティーとして設定されます。 これらの値をオーバーライドするには、 Example.java でデフォルトを変更するか、 -D パラメーターを使用して Java コマンド行でオプションとしてプロパティーを指定します。



この例および 484 ページの『Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成』 の例で使用されるクライアント ID は、ランダム・ストリングが末尾に付いたユーザー名です。

5. 次の手順に従ってコードを作成するか、または 483 ページの図 88 からコードをコピーしてください。

   後の手順では、Pubsync.java のコードについて説明します。

6. try-catch ブロックを作成します。

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

   MQTT クライアントは、MqttException、MqttPersistenceException、または MqttSecurityException をスローします。 MqttPersistenceException および MqttSecurityException は、MqttException のサブクラスです。

   MqttException.getReasonCode メソッドを使用して、例外の理由を検索します。 MqttPersistenceException または MqttSecurityException がスローされた場合は、getCause メソッドを使用して、基礎となるスロー可能な例外を返します。

7. 新しい MqttClient インスタンスを作成します。

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

   後で WebSphere MQ に接続するために使用するサーバー・アドレスをクライアントに指定します。 クライアントを指定するためのクライアント ID を設定します。

   - オプションで、MqttClientPersistence インターフェースの実装を提供して、デフォルトの実装を置き換えることができます。 デフォルトの MqttPersistence 実装では、送達を待機している QoS 1 および 2 メッセージがファイルとして保管されます。 538 ページの『MQTT クライアントでのメッセージ持続性』 を参照してください。
   - MQTT のデフォルトの IBM WebSphere MQ TCP/IP ポートは 1883 です。 SSL の場合は 8883 になります。 この例では、デフォルト・アドレスは tcp://localhost:1883 に設定されます。
   - 通常は、クライアント ID を使用して特定の物理クライアントを識別できることが重要です。 クライアント ID は、サーバーに接続するすべてのクライアントで固有である必要があります。534 ページの『クライアント ID』 を参照してください。 前のインスタンスと同じクライアント ID の使用は、現在のインスタンスが、同じクライアントのインスタンスであることを示します。 実行中の 2 つのクライアントのクライアント ID が重複していると、両方のクライアントに例外がスローされ、一方のクライアントが強制終了します。
   - クライアント ID の長さは、23 バイトに制限されています。 この長さを超えると、例外がスローされます。 クライアント ID には、キュー・マネージャー名で許可されている文字のみを使用する必要があります。例えば、ハイフンやスペースなどは使用できません。
   - MqttClient.connect メソッドを呼び出すまで、メッセージ処理は実行されません。

   クライアント・オブジェクトを使用して、トピックをパブリッシュおよびサブスクライブし、まだ送達されていないパブリケーションに関する情報をリカバリーします。

8. パブリッシュするトピックを作成します。

```
MqttTopic topic = client.getTopic(Example.topicString);
```

トピック・ストリングは 64 K バイトに制限されています。これは、IBM WebSphere MQ トピック・ストリングの最大長を超えています。 他の点では、トピック・ストリングは WebSphere MQ トピック・ストリングと同じルールに従います。トピック・ストリングを参照してください。 この例では、トピック・ストリング MQTT Examples を設定します。

9. パブリケーション・メッセージを作成します。

```
MqttMessage message = new MqttMessage(Example.publication.getBytes());
```

ストリング "Hello World" はバイト配列に変換され、MqttMessage の作成に使用されます。

- MQTT メッセージ・ペイロードは常にバイト配列です。 getBytes メソッドは、ストリング・オブジェクトを UTF-8 に変換します。 MqttMessage には、メッセージ・ペイロードをストリングとして返すための便利な toString メソッドがあります。 これは、new string(message.getPayload) に相当します。
- パブリケーション・メッセージは、RFH2 ヘッダーを使用してキュー・マネージャーに送信され、メッセージ・データは jms-bytes メッセージとして送信されます。
- このメッセージ・オブジェクトには、サービスの品質および保持された属性が含まれます。 サービスの品質 (QoS) により、MQTT クライアントとキュー・マネージャー間でのメッセージの転送の信頼性が決定されます。543 ページの『MQTT クライアントによって提供されるサービス品質』を参照してください。 保持された属性により、キュー・マネージャーが今後のサブスクライバーのためにパブリケーションを保管するかどうかを制御します。 保持されない場合、パブリケーションは現在のサブスクライバーにのみ送信されます。545 ページの『保存パブリケーションおよび MQTT クライアント』を参照してください。 デフォルトの MqttMessage 設定は、"メッセージは少なくとも 1 回配信され、保持されません。"

10. サーバーに接続します。

```
client.connect();
```

この例では、デフォルトの接続オプションを使用してサーバーに接続します。 接続後、パブリッシュを開始できます。 デフォルトの接続オプションは、次のとおりです。

- TCP/IP 接続が閉じられるのを防ぐために、小さな「キープアライブ」メッセージが 15 秒ごとに送信されます。
- 前のパブリケーションの完了を確認せずに、セッションが開始されます。
- メッセージの送信を再び試行するまでの間隔は 15 秒です。
- 接続の遺言メッセージは作成されません。
- 標準の SocketFactory が接続の作成に使用されます。

ConnectionOptions オブジェクトを作成し、それを追加パラメーターとして client.connect に渡すことで、接続オプションを変更します。

11. パブリッシュ。

```
MqttDeliveryToken token = topic.publish(message);
```

この例では、トピック「MQTT Examples」への「Hello World」パブリケーションをキュー・マネージャーに送信します。

- publish メソッドが戻ると、メッセージは安全に MQTT クライアントに転送されますが、サーバーにはまだ転送されません。 メッセージに QoS 1 または QoS 2 が含まれる場合、送達が完了する前にクライアントに障害が発生すると、そのメッセージはローカルに保存されます。
- publish は、確認応答がサーバーから受信されているかどうかを確認するために使用する送達トークンを返します。

12. サーバーからの確認応答を待ちます。

```
token.waitForCompletion(Example.timeout);
```

PubSync 例では、メッセージが送達されたことを確認する、サーバーからの確認応答を待機します。

- タイムアウトが設定されていないと、クライアントは永久に待ち続けます。 タスク、484 ページの『Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成』では、コールバック・オブジェクトを使用して、待機せずに確認応答を受信する方法を示します。

13. クライアントをサーバーから切断します。

```
client.disconnect();
```

    クライアントはサーバーから切断され、実行中の MqttCallback メソッドが終了するのを待機します。 次に、残りの処理が完了するのを最大 30 秒間待機します。 静止タイムアウトを追加のパラメーターとして指定できます。

14. 変更を PubSync.java および Example.Java に保存します。

    Eclipse は自動的に Java をコンパイルします。 これで、プログラムを実行して結果を確認する準備が整いました。

タスクの結果

WebSphere MQ を使用してパブリケーションを表示するには、 482 ページの図 87 のスクリプトを使用して、トピック、キュー、および永続サブスクリプション (すべて "MQTTExampleTopic" と呼ばれる) を作成します。 クライアントを実行して MQTT Examples トピックにパブリッシュしてから、サンプル・プログラム **amqsbcg** を実行して、 MQTTExamples キューにあるパブリケーションを参照します。

1. キュー・マネージャーを開始し、そのテレメトリー (MQXR) サービスの実行を開始します。 テレメトリー・チャネルに設定された TCP/IP アドレスおよびポートが、MQTT アプリケーションで使用する値と一致するように構成します。
2. 永続サブスクリプションを構成するには、 mqttexamples.txt コマンド・スクリプトを作成し、 **runmqsc:** を使用してそれを実行します。

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

    図 *87. mqttExampleTopic.txt*

    Windows でこのスクリプトを実行するには、次のコマンドを入力します。

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Eclipse 内から、またはコマンド・ウィンドウで Java を実行して、クライアントを Java アプリケーションとして実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

    注: コマンド・ウィンドウは、パス com\ibm\mq\id を含むディレクトリーで開いている必要があります。

4. WebSphere MQ エクスプローラーを使用するか次のコマンドを実行して、結果を参照します。

```
amqsbcg MQTTExampleQueue queue manager name
```

コード例

PubSync.java は、この手順で説明しているコードをすべてリストしています。 484 ページの図 89 の Example クラスを変更して、PubSync.java で使用されているデフォルト・パラメーターをオーバーライドします。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      client.connect();
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
          + " seconds for publication of \"" + message.toString()
          + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName()
          + "\" for client instance: \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *88. PubSync.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

図 *89. Example.java*

関連概念

MQTT パブリッシュ/サブスクライブ・アプリケーション

Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成

このタスクでは、最初のパブリッシャー・アプリケーションを、チュートリアルに従って変更します。 この変更によってアプリケーションは、送達確認を待たずにパブリケーションを送信できるようになります。送達確認は、作成するコールバック・クラスによって受信されます。

始める前に

WebSphere MQ Telemetry フィーチャーを、IBM WebSphere MQ Version 7.1 以降がインストールされているサーバーにインストールします。

クライアント・アプリケーションは、 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) の `com.ibm.mq.micro.client.mqttv3` パッケージを使用します。 この SDK は、IBM WebSphere MQ Telemetry インストールの一部です。 クライアントは、IBM WebSphere MQ Telemetry 機能に接続して、IBM WebSphere MQ とメッセージを交換します。

IBM WebSphere MQ Telemetry を管理するには、 IBM WebSphere MQ Explorer Version 7.1 のテレメトリー更新もインストールする必要があります。 更新は、IBM WebSphere MQ Telemetry インストールの一部です。

Java SE で稼働する MQTT クライアントには、Java SE のバージョン 6.0 以降が必要です。 IBM Java SE v6.0 は、 IBM WebSphere MQ Version 7.1 インストールの一部です。 これは *WebSphere MQ installation directory*\java\jre にあります。

このタスクについて

この例は、パブリッシュ・アプリケーション PubAsync です。 PubAsync は、パブリケーションがキュー・マネージャーに送達されたことの確認を待たずに、トピック MQTT Examples に Hello World をパブリッシュします。 送達確認は、コールバック・クラス CallBack で受け取られます。

MQTT Examples への永続サブスクリプションを設定することにより、アプリケーションが機能することを確認できます。

この手順では、Eclipse を使用して、クライアントを開発、構築、および実行します。 Eclipse は、Eclipse プロジェクトの Web サイト (www.eclipse.org) からダウンロードできます。

手順に示したステップは、478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』 での PubSync.java アプリケーションを変更するためのステップです。

あるいは、コード 487 ページの『コード例』 を新しいディレクトリー .\com\ibm\mq\id にコピーすることもできます。 Example.java、 CallBack.java、および PubAsync.java の 3 つの Java ファイルを作成します。 次のコマンドを使用して、この例をコンパイルします。

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsync.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Example.java
```

次のコマンドを使用して、PubAsync を実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsync.class
```

手順

1. com.ibm.mq.id パッケージに、ファイル CallBack.java を作成します。 488 ページの図 92 のコードをこのファイルにコピーします。

   Callback.java は、MqttCallBack インターフェースを実装しています。 例では、追加コンストラクターがいくつかのインスタンス・データを使用してコールバックを初期化します。
2. com.ibm.mq.id パッケージで、PubSync.java を右クリックし、コピーします。 その名前を PubAsync に変更して、同じパッケージに貼り付けます。
3. コードの client.connect(); 行の直前で、クライアント ID を渡して CallBack クラスをインスタンス化します。

   ```
   CallBack callback = new CallBack(Example.clientId);
   client.setCallback(callback);
   ```

   - CallBack クラスは MqttCallBack を実装しています。 クライアント ID ごとにコールバック・インスタンスを 1 つ必要とします。 この例では、インスタンス・データとして保存するクライアント ID をコンストラクターが渡します。 これは、どのコールバック・インスタンスが開始されたかを識別するためにコールバックで使用されます。
   - コールバック・クラスに以下の 3 つのメソッドを実装する必要があります。

     public void messageArrived(MqttTopic topic, MqttMessage message)
     サブスクライブされているパブリケーションを受信します。

     public void connectionLost(Throwable cause)
     接続が失われると呼び出されます。

public void deliveryComplete(MqttDeliveryToken token))
パブリッシュされた QoS 1 または 2 メッセージの送達トークンが受信されると呼び出されます。

- コールバックは MqttClient.connect によってアクティブ化されます。

4. クライアントを切断します。
   a) token.waitForCompletion 式が含まれるステートメントを削除します。
   メインスレッドは、パブリケーションが送達されるのを待たずに続行します。
   b) クライアントが既に切断されているかどうかをテストします。
   MQTT クライアントは、MqttCallback の lostConnection メソッドに返されるエラーに続いて切断します。 あるいは、クライアント・アプリケーションが切断する場合もあります。 オープン接続があるかどうかをテストします。
   c) 定数 Example.quiesceTimeout を使用して、クライアントを静止するための最大時間を設定します。

```
if (client.isConnected())
    client.disconnect(Example.quiesceTimeout);
```

以下の 3 つの条件の組み合わせが真の場合は、クライアントが終了します。

a. このセッションで (セッションが再始動された場合は前のセッションで) パブリッシュされたすべてのメッセージについてコールバックが呼び出された。
b. メッセージが伝送途中なのに静止間隔の時間切れになった。 デフォルトでは、静止間隔は 30 秒です。 この静止タイムアウトは、待機ミリ秒数を client.disconnect のパラメーターとして渡すことによって変更できます。
c. いくつかのメッセージがクライアントによってパブリッシュされてキューに入れられた後、メッセージが送信される前に client.disconnect が呼び出された。 キューに入れられたメッセージは、まだ伝送途中ではありません。 セッションが再始動可能の場合は、セッションが再始動するとメッセージが再送されます。

タスクの結果

WebSphere MQ を使用してパブリケーションを表示するには、 486 ページの図 90 のスクリプトを使用して、トピック、キュー、および永続サブスクリプション (すべて "MQTTExampleTopic" と呼ばれる) を作成します。 クライアントを実行して MQTT Examples トピックにパブリッシュしてから、サンプル・プログラム **amqsbcg** を実行して、 MQTTExamples キューにあるパブリケーションを参照します。

1. キュー・マネージャーを開始し、そのテレメトリー (MQXR) サービスの実行を開始します。 テレメトリー・チャネルに設定された TCP/IP アドレスおよびポートが、MQTT アプリケーションで使用する値と一致するように構成します。
2. 永続サブスクリプションを構成するには、 mqttexamples.txt コマンド・スクリプトを作成し、 **runmqsc:** を使用してそれを実行します。

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

図 *90. mqttExampleTopic.txt*

Windows でこのスクリプトを実行するには、次のコマンドを入力します。

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Eclipse 内から、またはコマンド・ウィンドウで Java を実行して、クライアントを Java アプリケーションとして実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

注: コマンド・ウィンドウは、パス com\ibm\mq\id を含むディレクトリーで開いている必要があります。

4. WebSphere MQ エクスプローラーを使用するか次のコマンドを実行して、結果を参照します。

```
amqsbcg MQTTExampleQueue queue manager name
```

コード例

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubAsync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      client.connect();
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *91. PubAsync.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *92. CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

図 *93. Example.java*

Java を使用した MQ Telemetry Transport のリカバリー可能非同期パブリッシャーの作成

このタスクでは、チュートリアルに従って非同期パブリッシャー・アプリケーションを変更します。 この変更により、アプリケーションは、クライアントが前回実行されたときに認知されなかったパブリケーションの送達を完了できます。

始める前に

WebSphere MQ Telemetry フィーチャーを、IBM WebSphere MQ Version 7.1 以降がインストールされているサーバーにインストールします。

クライアント・アプリケーションは、 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) の `com.ibm.mq.micro.client.mqttv3` パッケージを使用します。 この SDK は、IBM WebSphere MQ Telemetry インストールの一部です。 クライアントは、IBM WebSphere MQ Telemetry 機能に接続して、IBM WebSphere MQ とメッセージを交換します。

IBM WebSphere MQ Telemetry を管理するには、 IBM WebSphere MQ Explorer Version 7.1 のテレメトリー更新もインストールする必要があります。 更新は、IBM WebSphere MQ Telemetry インストールの一部です。

Java SE で稼働する MQTT クライアントには、Java SE のバージョン 6.0 以降が必要です。 IBM Java SE v6.0 は、 IBM WebSphere MQ Version 7.1 インストールの一部です。 これは *WebSphere MQ installation directory*\java\jre にあります。

このタスクについて

この例は、パブリッシュ・アプリケーション PubAsyncRestartable です。 PubAsyncRestartable は、パブリケーションがキュー・マネージャーに送達されたことの確認を待たずに、「Hello World」をトピック MQTT Examples にパブリッシュします。 送達確認は、コールバック・クラス CallBack で受け取られます。 直前のインスタンスで完了していないパブリケーションのすべての送達トークンを調べることができます。 それらはまた、コールバック・クラスによって処理されます。

MQTT Examples への永続サブスクリプションをセットアップすることで、アプリケーションが機能することを確認できます。

この手順では、Eclipse を使用して、クライアントを開発、構築、および実行します。 Eclipse は、Eclipse プロジェクトの Web サイト (www.eclipse.org) からダウンロードできます。

「プロシージャー」 のステップでは、 484 ページの『Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成』内の PubAsync.java アプリケーションを変更します。

あるいは、コード 493 ページの『コード例』を新しいディレクトリー .\com\ibm\mq\id にコピーすることもできます。 Example.java、 CallBack.java、および PubAsyncRestartable.java の 3 つの Java ファイルを作成します。 次のコマンドを使用して、この例をコンパイルします。

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsyncRestartable.java com.ibm.mq.id.CallBack.java
com.ibm.mq.id.Example.java
```

次のコマンドを使用して、PubAsyncRestartable を実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsyncRestartable.class
```

手順

1. com.ibm.mq.id パッケージで、PubAsync.java を右クリックしてコピーします。 それを同じパッケージ内に貼り付けて、PubAsyncRestartable に名前変更します。
2. 再使用可能なクライアント ID を作成します。

   ```
   Example.clientId = String.format(
       "%-23.23s",
       (System.getProperty("user.name") + "_" + (System.getProperty(
           "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
   ```

   図 *94.* 再使用可能なクライアント *ID*

   478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』および 484 ページの『Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成』のアプリケーションは、クライアント接続ごとに新しいクライアント ID を使用しました。 再始動可能なパブリッシャーまたはサブスクライバーでは、クライアントが接続されるごとに同じクライアント ID を使用すること、および異なるクライアントには異なる ID を使用することが必要です。 534 ページの『クライアント ID』を参照してください。 再使用可能なクライアント ID は、ユーザー名およびクラスの名前から作成されます。 長さの限度は 23 バイトです。 キュー・マネージャー・オブジェクト名で有効な文字だけを使用する必要があります。 ハイフンが挿入されている場合、コードによって削除されます。
3. 重複メッセージを避けるために、メッセージの QoS はデフォルトの 1 ではなく 2 に設定されます。

```
message.setQos(Example.QoS);
```

Example.QoS の値を 2 に変更するか、Java コマンド行で -DQoS=2 オプションを使用して QoS プロパティーを引数として渡す必要があります。

4. MqttConnectOptions オブジェクトを作成し、その cleanSession 属性を false に設定します。

   a) MqttConnectOptions オブジェクトを作成します。

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   ```

   conOptions は、MqttClient コンストラクターのオプション・パラメーターです。

   b) clearSession 属性を設定します。

   ```
   conOptions.setCleanSession(Example.cleanSession);
   ```

   デフォルトで、パラメーター Example.cleanSession は、MqttConnectionOptions.cleanSession のデフォルト設定と一致するように true に設定されます。

   PubAsyncRestartable が再始動するとき、「クリーン・セッション」で始動して、QoS 1 または 2 の保留中のメッセージの送達トークンを消去することができます。

   すべての保留中の送達トークンを保持するためには、Example.cleanSession を false に設定します。 それらのトークンは、クライアントが再接続するときに、MqttCallBack クラスによって処理されます。

5. セッションが再始動している場合は、保留中の送達トークンをすべて取り出して、その内容をプリントします。

```
if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance "
            + client.getClientId());
```

6. conOptions パラメーターを MqttClient コンストラクターに渡します。

```
client.connect(conOptions);
```

7. 切断する際に、最大切断間隔を設定します。

```
client.disconnect(Example.timeout);
```

処理されている保留中の送達トークンを表示できるのは、直前のインスタンスが送達を完了しないで終了したときだけです。 この例を、PubAsyncRestartable が終了する前にパブリケーションが認識されない可能性があるときに実行するには、Example.timeout を 0 に設定します。

タスクの結果

WebSphere MQ を使用してパブリケーションを表示するには、 492 ページの図 95 のスクリプトを使用して、トピック、キュー、および永続サブスクリプション (すべて "MQTTExampleTopic" と呼ばれる) を作成します。 クライアントを実行して MQTT Examples トピックにパブリッシュしてから、サンプル・プログラム **amqsbcg** を実行して、 MQTTExamples キューにあるパブリケーションを参照します。

1. キュー・マネージャーを開始し、そのテレメトリー (MQXR) サービスの実行を開始します。 テレメトリー・チャネルに設定された TCP/IP アドレスおよびポートが、MQTT アプリケーションで使用する値と一致するように構成します。
2. 永続サブスクリプションを構成するには、 mqttexamples.txt コマンド・スクリプトを作成し、**runmqsc:** を使用してそれを実行します。

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

図 *95. mqttExampleTopic.txt*

Windows でこのスクリプトを実行するには、次のコマンドを入力します。

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Eclipse 内から、またはコマンド・ウィンドウで Java を実行して、クライアントを Java アプリケーションとして実行します。

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

注: コマンド・ウィンドウは、パス com\ibm\mq\id を含むディレクトリーで開いている必要があります。

4. WebSphere MQ エクスプローラーを使用するか次のコマンドを実行して、結果を参照します。

```
amqsbcg MQTTExampleQueue queue manager name
```

コード例

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
import com.ibm.micro.client.mqttv3.MqttDeliveryToken;
import com.ibm.micro.client.mqttv3.MqttMessage;
import com.ibm.micro.client.mqttv3.MqttTopic;
public class PubAsyncRestartable {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + (System.getProperty(
            "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance \""
            + client.getClientId() + "\"");
      client.connect(conOptions);
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *96. PubAsyncRestartable.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *97. CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

図 *98. Example.java*

Java を使用した MQ Telemetry Transport のサブスクライバーの作成

このタスクでは、チュートリアルに従ってサブスクライバー・アプリケーションを作成します。 サブスクライバーはトピックへのサブスクリプションを作成して、サブスクリプションのためのパブリケーションを受け取ります。

始める前に

WebSphere MQ Telemetry フィーチャーを、IBM WebSphere MQ Version 7.1 以降がインストールされているサーバーにインストールします。

クライアント・アプリケーションは、 IBM WebSphere MQ Telemetry Software Development Toolkit (SDK) の `com.ibm.mq.micro.client.mqttv3` パッケージを使用します。 この SDK は、IBM WebSphere MQ Telemetry インストールの一部です。 クライアントは、IBM WebSphere MQ Telemetry 機能に接続して、IBM WebSphere MQ とメッセージを交換します。

IBM WebSphere MQ Telemetry を管理するには、 IBM WebSphere MQ Explorer Version 7.1 のテレメトリー更新もインストールする必要があります。 更新は、IBM WebSphere MQ Telemetry インストールの一部です。

Java SE で稼働する MQTT クライアントには、Java SE のバージョン 6.0 以降が必要です。 IBM Java SE v6.0 は、 IBM WebSphere MQ Version 7.1 インストールの一部です。 これは *WebSphere MQ installation directory*\java\jre にあります。

このタスクについて

この例は、サブスクライバー・アプリケーション Subscribe です。 Subscribe はサブスクリプション・トピック MQTT Examples を作成して、サブスクリプションのパブリケーションを 30 秒間待機します。

サブスクライバーはサブスクリプションを作成して、パブリケーションを待機することができます。 また、以前に作成された同じクライアント ID のサブスクリプションに対して送信されたパブリケーションを受け取ることもできます。 MqttConnectionOptions.cleanSession ブール属性は、以前に送信されたパブリケーションを受け取るかどうかを制御します。 546 ページの『サブスクリプション』を参照してください。

パブリッシュのサンプル・プログラムを使用してパブリケーションを作成することも、 WebSphere MQ エクスプローラーを使用して MQTT Examples トピックのテスト・パブリケーションを作成することもできます。

この手順では、Eclipse を使用して、クライアントを開発、構築、および実行します。 Eclipse は、Eclipse プロジェクトの Web サイト (www.eclipse.org) からダウンロードできます。

手順に示された指示は、以前のいずれかのタスクで com.ibm.mq.id パッケージを既に作成していて、それを Example.java および Callback.java クラスにコピーしたことを想定しています。

手順

1. クラス Subscribe をパッケージ com.ibm.mq.id に作成します。
2. 再使用可能なクライアント ID を作成します。

```
Example.clientId = String.format(
    "%-23.23s",
    (System.getProperty("user.name") + "_" + (System.getProperty(
        "clientId", "Subscribe."))).trim()).replace('-', '_');
```

図 *99.* 再使用可能なクライアント *ID*

478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』および 484 ページの『Java を使用した MQ Telemetry Transport の非同期パブリッシャーの作成』のアプリケーションは、クライアント接続ごとに新しいクライアント ID を使用しました。 再始動可能なパブリッシャーまたはサブスクライバーでは、クライアントが接続されるごとに同じクライアント ID を使用すること、および異なるクライアントには異なる ID を使用することが必要です。 534 ページの『クライアント ID』を参照してください。 再使用可能なクライアント ID は、ユーザー名およびクラスの名前から作成されます。 長さの限度は 23 バイトです。 キュー・マネージャー・オブジェクト名で有効な文字だけを使用する必要があります。 ハイフンが挿入されている場合、コードによって削除されます。

3. try-catch ブロックを作成します。

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

MQTT クライアントは、MqttException、MqttPersistenceException、または MqttSecurityException をスローします。 MqttPersistenceException および MqttSecurityException は、MqttException のサブクラスです。

MqttException.getReasonCode メソッドを使用して、例外の理由を検索します。 MqttPersistenceException または MqttSecurityException がスローされた場合は、getCause メソッドを使用して、基礎となるスロー可能な例外を返します。

4. 新しい MqttClient インスタンスを作成します。

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

後で WebSphere MQ に接続するために使用するサーバー・アドレスをクライアントに指定します。 クライアントを指定するためのクライアント ID を設定します。

- オプションで、MqttClientPersistence インターフェースの実装を提供して、デフォルトの実装を置き換えることができます。 デフォルトの MqttPersistence 実装では、送達を待機している QoS 1 および 2 メッセージがファイルとして保管されます。 538 ページの『MQTT クライアントでのメッセージ持続性』を参照してください。
- MQTT のデフォルトの IBM WebSphere MQ TCP/IP ポートは 1883 です。 SSL の場合は 8883 になります。 この例では、デフォルト・アドレスは tcp://localhost:1883 に設定されます。
- 通常は、クライアント ID を使用して特定の物理クライアントを識別できることが重要です。 クライアント ID は、サーバーに接続するすべてのクライアントで固有である必要があります。534 ページの『クライアント ID』を参照してください。 前のインスタンスと同じクライアント ID の使用は、現在のインスタンスが、同じクライアントのインスタンスであることを示します。 実行中の 2 つのクライアントのクライアント ID が重複していると、両方のクライアントに例外がスローされ、一方のクライアントが強制終了します。
- クライアント ID の長さは、23 バイトに制限されています。 この長さを超えると、例外がスローされます。 クライアント ID には、キュー・マネージャー名で許可されている文字のみを使用する必要があります。例えば、ハイフンやスペースなどは使用できません。
- MqttClient.connect メソッドを呼び出すまで、メッセージ処理は実行されません。

クライアント・オブジェクトを使用して、トピックをパブリッシュおよびサブスクライブし、まだ送達されていないパブリケーションに関する情報をリカバリーします。

5. コードの client.connect(); 行の直前で、クライアント ID を渡して CallBack クラスをインスタンス化します。

```
CallBack callback = new CallBack(Example.clientId);
client.setCallback(callback);
```

- CallBack クラスは MqttCallBack を実装しています。 クライアント ID ごとにコールバック・インスタンスを 1 つ必要とします。 この例では、インスタンス・データとして保存するクライアント ID をコンストラクターが渡します。 これは、どのコールバック・インスタンスが開始されたかを識別するためにコールバックで使用されます。
- コールバック・クラスに以下の 3 つのメソッドを実装する必要があります。

  public void messageArrived(MqttTopic topic, MqttMessage message)
  サブスクライブされているパブリケーションを受信します。

  public void connectionLost(Throwable cause)
  接続が失われると呼び出されます。

  public void deliveryComplete(MqttDeliveryToken token))
  パブリッシュされた QoS 1 または 2 メッセージの送達トークンが受信されると呼び出されます。
- コールバックは MqttClient.connect によってアクティブ化されます。

6. MqttConnectOptions オブジェクトを作成して、その cleanSession 属性を設定します。

a) MqttConnectOptions オブジェクトを作成します。

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

conOptions は、MqttClient コンストラクターのオプション・パラメーターです。

b) clearSession 属性を設定します。

```
conOptions.setCleanSession(Example.cleanSession);
```

デフォルトで、パラメーター Example.cleanSession は、MqttConnectionOptions.cleanSession のデフォルト設定と一致するように true に設定されます。

デフォルトの MqttConnectOptions を使用するか、クライアントに接続する前に MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にそのクライアントの古いサブスクリプションがすべて削除されます。 セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続の前に MqttConnectOptions.cleanSession を false に設定すると、クライアントによって作成されるサブスクリプションはすべて、接続する前にクライアントに存在していたすべてのサブスクリプションに追加されます。 クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

cleanSession 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。 そのデフォルト・モード cleanSession=true では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。 それに代わる cleanSession=false モードでは、サブスクリプションは永続的になります。 クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。 クライアントは、再接続時にすべての未送達パブリケーションを受信します。 接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

cleanSession モードは接続する前に設定する必要があります。モードはセッション中ずっと続きます。 その設定を変更するには、クライアントを切断し、再接続する必要があります。 モードを cleanSession=false の使用から cleanSession=true の使用に変更すると、クライアントの前回のサブスクリプションすべてと、受信されていないパブリケーションがあればそれらすべてが廃棄されます。

7. conOptions パラメーターを MqttClient コンストラクターに渡します。

```
client.connect(conOptions);
```

8. サブスクリプションを作成します。

```
      client.subscribe(Example.topicString, Example.QoS);
```

この例は、1 つのトピック・フィルターを QoS オプションと共に渡す MqttClient.subscribe メソッドを使用します。 MqttClient.subscribe メソッドには 4 つのシグニチャーがあり、サブスクリプション・フィルターの配列を渡すことも単一のフィルターを渡すこともできます。

この例は、パブリッシュ例で使用されたトピック・ストリングをトピック・フィルターとして使用して、作成されるどのパブリケーションでも受け取るようにします。

例としての subscribe.java を実行するたびに、サブスクリプションが作成されます。 Example.topicString を変更しなければ、同じサブスクリプションが再作成されます。 サブスクリプションが再作成される場合、2 つの同じサブスクリプションとなることはありません。 クライアントは、同じサブスクリプションと一致する重複したパブリケーションのコピーを受け取りません。

サブスクリプションについては 546 ページの『サブスクリプション』、フィルターについては 548 ページの『MQTT クライアントのトピック・ストリングおよびトピック・フィルター』で説明されています。

9. いくつかのパブリケーションが届くまで待機してから、クライアントを切断します。

```
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
```

パブリケーションは、MqttCallback.messageArrived メソッドの実装によって受け取られます。

サブスクライブ・アプリケーションはメッセージをパブリッシュしていないので、送達トークンを待機しません。 client.disconnect は遅延することなく生じます。

コード例

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
public class Subscribe {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + System.getProperty("clientId",
            "Subscribe."))).trim();
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      client.connect(conOptions);
      System.out.println("Subscribing to topic \"" + Example.topicString
          + "\" for client instance \"" + client.getClientId()
          + "\" using QoS " + Example.QoS + ". Clean session is "
          + Example.cleanSession);
      client.subscribe(Example.topicString, Example.QoS);
      System.out.println("Going to sleep for " + Example.sleepTimeout / 1000
          + " seconds");
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
      System.out.println("Finished");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *100. Subscribe.java*

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

図 *101. CallBack.java*

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

図 *102. Example.java*

関連概念

MQTT パブリッシュ/サブスクライブ・アプリケーション

JAAS を使用する MQTT Java クライアントの認証

JAAS を使用してクライアントを認証する方法を学びます。 サンプル・プログラムの JAASLoginModule.java および Java プログラム例の PubSync.java を変更します。 テレメトリー・チャネルを JAAS 認証を必要とするように構成して、変更されたパブリッシャーを実行します。 その際に、JAAS を使用してユーザー名およびパスワードを検査します。

始める前に

このタスクを実行する前に、MQTT v3 クライアント jar ファイル、Javadoc、Eclipse がインストール済みであること、テレメトリー・チャネルが構成済みであること、および PubSync.java をコーディングして実行したことが想定されています。 PubSync.java の稼働中のバージョンを含む Eclipse ワークスペースが必要です。

このタスクは、Windows 用に記述されています。 Linux では、ディレクトリー・パスを変更してください。

このタスクについて

このタスクは、*WMQ Installation directory*\mqxr\samples\JAASLoginModule.java 内のサンプル JAASLoginModule クラスを変更して MyLogin.java を作成することに基づいています。 タスクでは、478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』にあるコード例の PubSync.java も変更して、ユーザー名およびパスワードを設定します。 テストの際に、MyLogin.java はユーザー名およびパスワードをランダムに受諾したり拒否したりします。

タスクにあるステップは、プログラミングの練習用に記述されています。 実稼働環境では、実際の認証を行うための手順に合わせる必要があります。

JAAS 認証のプログラム方法を示す典型的な説明として、ログイン・モジュールが JAAS をロードしたコンテキストを認証していると想定します。 テレメトリー (MQXR) サービスが JAAS を呼び出すとき、JAAS をロードしたコンテキストはテレメトリー (MQXR) サービスです。 テレメトリー (MQXR) サービスのコンテキストを認証するところはありません。それは常に mqm です。 代わりに、テレメトリー (MQXR) サービスは、クライアントのユーザー名およびパスワードがログイン・モジュール・クラスで使用可能となるようにセットアップします。 ユーザー名およびパスワードは、2 つのコールバックを使用してログイン・モジュールに渡されます。

```
javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
callbacks[0] =
      new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] =
      new javax.security.auth.callback.PasswordCallback("PasswordCallback", false);
callbackHandler.handle(callbacks);
String username =
      ((javax.security.auth.callback.NameCallback) callbacks[0]).getName();
char[] password =
      ((javax.security.auth.callback.PasswordCallback) callbacks[1]).getPassword();
```

クライアントの username および password だけが、ログイン・モジュールで使用可能なクライアントに関する情報です。

手順

1. PubSync.java と同じ Java プロジェクトに、2 つのパッケージ samples および security.jaas を作成します。

   samples パッケージは参照のためだけに使用されます。 security.jaas パッケージでコードを変更してください。
2. JAASLoginModule.java および JAASPrincipal.java を両方のパッケージにインポートします。

   必要であれば、Java ソースのパッケージ・ステートメントをリファクタリングしてコンパイル・エラーを除去します。
3. security.jaas パッケージ内のクラス名 JAASLoginModule を MyLogin にリファクタリングします。
4. MyLogin.java で、login メソッド内のいくつかのコードを置き換えて、モジュールが機能していることを示します。

   a) 以下のコードを置き換えます。

   ```
   // Accept everything.
   if (true)
     loggedIn = true;
   else
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   ```

   b) 以下のコードに置き換えます。

   ```
   // login half the users randomly
   PrintWriter pw = new PrintWriter(new FileWriter(System.getProperty("user.dir")
           + "\\MyLogin.log", true));
   pw.println("Called JAASLogin.login at "
           + System.getProperty("publication", "Hello World "
           + String.format("%tc", System.currentTimeMillis())));
   ```

```
if (Math.random() < 0.5)
  loggedIn = true;
pw.println("Username: \"" + username + "\", Password: \""
    + String.valueOf(password) + "\" loggedIn: " + loggedIn);
pw.close();
if (!loggedIn)
  throw new javax.security.auth.login.FailedLoginException("Login failed");
principal= new JAASPrincipal(username);
```

MyLogin.java の完全なソースは、504 ページの図 105 にあります。 パッケージ名が security.jaas にリファクタリングされた JAASPrincipal.java のソースは、505 ページの図 106 にあります。

5. service.env のクラスパスを、security/jaas/MyLogin.class および security/jaas/JAASPrincipal.class へのパスが含まれるディレクトリーを指すように設定します。

```
CLASSPATH=C:\WMQTelemetryApps\MQTTSecureExamples\bin
```

service.env を使用してクラスパスを WebSphere MQ サービスに渡す方法について詳しくは、テレメトリー・チャネルの JAAS 構成を参照してください。

6. jaas.config にログイン・モジュール・スタンザを追加します。

```
MyLoginExample {
  security.jaas.MyLogin required debug=true;
  };
```

jaas.config を使用して JAAS ログイン・モジュールを定義する方法について詳しくは、テレメトリー・チャネルの JAAS 構成を参照してください。

7. WebSphere MQ エクスプローラー内の「**新規テレメトリー・チャネル**」ウィザードを使用してテレメトリー・チャネルを追加します。 その際に、チャネルが JAAS 認証を必要とするように構成します。 それが MyLoginExample スタンザを参照するようにします。

例えば、ウィザードに入力する情報は mqxr_win.properties ファイル内のこのスタンザを改変したものとします。 Linux で作業をしている場合、ファイル名は mqxr_unix.properties です。 テレメトリー・プロパティー・ファイルは、直接編集せずに、ウィザードを使用してください。

```
com.ibm.mq.MQXR.channel/JAASMCAUser: \
com.ibm.mq.MQXR.Port=1884;\
com.ibm.mq.MQXR.JAASConfig=MyLoginExample;\
com.ibm.mq.MQXR.UserName=Admin;\
com.ibm.mq.MQXR.StartWithMQXRService=true
```

注: いずれかのテレメトリー・チャネル・パラメーターを変更するか、または security.jaas.Mylogin クラスを変更する場合、テレメトリー (MQXR) サービスを停止して再始動する必要があります。 サービスを再始動したときにのみ、変更内容が有効になります。

8. com.ibm.mq.id パッケージ内に PubSync.java のコピーを作成して、そのコピーの名前を PubSyncJAAS.java にします。

com.ibm.mq.id パッケージ内に PubSync.java を作成する手順については、478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』を参照してください。

9. PubSyncJAAS.java プログラム内の MqttConnectOptions.username および MqttConnectOptions.password を設定して、MqttConnectOptions を MqttClient.connect のパラメーターとして渡します。

```
MqttConnectOptions conOptions = new MqttConnectOptions();
conOptions.setUserName(Example.username);
conOptions.setPassword(Example.password);
client.connect(conOptions);
```

Example.java 内の定数セットを使用している、PubSyncJAAS.Java 内のイタリック体のコードを参照してください。

10. Example.TCPAddress を、JAAS 構成の MyLoginExample を使用するように構成したテレメトリー・チャネルのソケット・アドレスに設定します。 例えば、ポート番号として 1884 を使用します。
11. PubSyncJAAS を何回か実行して、クライアントがログインし、受諾されたり拒否されたりすることを確認します。

    ログインの試行が拒否されるたびに、例外がスローされます。

タスクの結果

503 ページの図 103 は、PubSyncJAAS.Java を 2 回実行したときの結果を示しています。 ログ・レコードが 503 ページの図 104 に示されています。

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:05 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_61c57a18_4bf7_40d" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Client exception caught
Client is not connected (32104)
    at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:88)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:105)
    at com.ibm.micro.client.mqttv3.MqttTopic.publish(MqttTopic.java:68)
    at com.ibm.mq.id.PubSync.main(PubSync.java:24)
```

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:40 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_1d1599a0_50f5_4ea" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Delivery token "1731749688" has been received: true
```

図 *103. PubSyncJAAS.java* からのコンソール出力

ログ・ファイル MyLogin.log は、 *WMQ Data directory* に保管されます。例えば、次のようになります。 C:\IBM\MQ\Data\MyLogin.log

```
Called JAASLogin.login at Hello World Fri Jun 04 08:31:05 BST 2010
Username: "Admin", Password: "Password" loggedIn: false
Called JAASLogin.login at Hello World Fri Jun 04 08:31:40 BST 2010
Username: "Admin", Password: "Password" loggedIn: true
```

図 *104. MyLogin.log*

例

504 ページの図 105 内のイタリック体のコードは、サンプルの JAASLoginModule.java への変更箇所です。

```
package security.jaas;
import java.io.FileWriter;
import java.io.PrintWriter;

public class JAASLogin implements javax.security.auth.spi.LoginModule {
  private javax.security.auth.Subject                  subject;
  private javax.security.auth.callback.CallbackHandler callbackHandler;
  JAASPrincipal                                        principal;
  boolean                                              loggedIn = false;
  public void initialize(javax.security.auth.Subject subject,
      javax.security.auth.callback.CallbackHandler callbackHandler,
      java.util.Map<String, ?> sharedState, java.util.Map<String, ?> options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
  }
  public boolean login() throws javax.security.auth.login.LoginException {
    try {
      javax.security.auth.callback.Callback[] callbacks = new
javax.security.auth.callback.Callback[2];
      callbacks[0] = new javax.security.auth.callback.NameCallback(
          "NameCallback");
      callbacks[1] = new javax.security.auth.callback.PasswordCallback(
          "PasswordCallback", false);

      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // login half the users randomly
      PrintWriter pw = new PrintWriter(new FileWriter(System
          .getProperty("user.dir")
          + "\\mylogin.log", true));
      pw.println("Called JAASLogin.login at "
          + System.getProperty("publication", "Hello World "
              + String.format("%tc", System.currentTimeMillis())));
      if (Math.random() < 0.5)
        loggedIn = true;
      pw.println("Username: \"" + username + "\", Password: \""
          + String.valueOf(password) + "\" loggedIn: " + loggedIn);
      pw.close();
      if (!loggedIn)
        throw new javax.security.auth.login.FailedLoginException("Login failed");
      principal = new JAASPrincipal(username);
    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }
    return loggedIn;
  }
  public boolean abort() throws javax.security.auth.login.LoginException {
    logout();
    return true;
  }
  public boolean commit() throws javax.security.auth.login.LoginException {
    if (loggedIn) {
      if (!subject.getPrincipals().contains(principal))
        subject.getPrincipals().add(principal);
    }
    return true;
  }
  public boolean logout() throws javax.security.auth.login.LoginException {
    subject.getPrincipals().remove(principal);
    principal = null;
    loggedIn = false;
    return true;
  }
}
```

図 *105. MyLogin.java*

505 ページの図 106 は、パッケージ security.jaas にコピーされたサンプル・コード JAASLoginPrincipal.java です。 JAASLoginPrincipal の目的は、java.security.Principal

インターフェースを実装して、MyLogin によって正常にログインしたユーザーの記録を保持することです。

```
package security.jaas;
public class JAASPrincipal implements java.security.Principal,
    java.io.Serializable {
  private static final long serialVersionUID = 1L;
  String                    name;
  public JAASPrincipal(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return (name);
  }
  public boolean equals(Object object) {
    if (object != null && object instanceof JAASPrincipal
        && name.equals(((JAASPrincipal) object).getName()))
      return true;
    else
      return false;
  }
  public int hashCode() {
    return name.hashCode();
  }
}
```

図 *106. JAASLoginPrincipal.java*

username および password を追加するために変更された PubSync.java 内のコードは、505 ページの図 107 内でイタリック体になっています。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setUserName(Example.username);
      conOptions.setPassword(Example.password);
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("With username \"" + conOptions.getUserName()
              + "\" and password \"" + String.valueOf(conOptions.getPassword()) + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

図 *107. PubSyncJAAS.Java*

Example.java 内の定数を、使用する構成と一致するように変更します。 この例では、SSL 設定は無視します。

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

図 *108. Example.java*

自己署名証明書を使用する SSL テレメトリー接続の認証

Keytool を使用して生成された自己署名証明書を使用して SSL 接続を認証します。 テレメトリー・チャネルを認証するか、またはテレメトリー・チャネルとそれに接続するクライアントとを認証するオプションがあります。 接続の上のメッセージ・フローは暗号化されます。

始める前に

開始する前にタスク 478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』を行い、PubSync.java が非セキュアな TCP/IP 接続での作業を開始するようにします。 このタスクでは、PubSync.java を変更して SSL 接続で機能するようにします。

このタスクについて

タスクにあるステップは、プログラミングの練習用に記述されています。 実稼働環境では、実際の認証を行うための手順に合わせる必要があります。

このタスクは、Windows 用に記述されています。 Linux では、ディレクトリー・パスを変更してください。

手順

1. タスク 507 ページの『SSL を使用するための PubSync.java の変更』を行って、SSL を使用するように PubSync.java を変更します。
2. テレメトリー・チャネルを構成して、SSL を使用する鍵ストアを作成します。

   テレメトリー・チャネルだけを認証するか、またはチャネルとそれに接続するクライアントとを認証します。

   - タスク 508 ページの『テレメトリー・チャネルの認証』 を実行して SSL に接続し、テレメトリー・チャネルを認証します。
   - タスク 509 ページの『テレメトリー・チャネルおよびクライアントの認証』 を実行して SSL に接続し、テレメトリー・チャネルとそれに接続するクライアントとを認証します。
3. テレメトリー (MQXR) サービスを停止して再始動することにより、テレメトリー・チャネルの構成に対する変更点を取得します。
4. クライアント・プログラムを実行して、構成が機能するかどうかを確認します。

SSL を使用するための PubSync.java の変更

最初のパブリッシャー・プログラム例を、SSL を使用してテレメトリー・チャネルに接続するように変更します。 変更されたプログラムで使用される SSL プロパティーを設定します。

始める前に

このタスクを実行する前に、MQTT v3 クライアント jar ファイル、Javadoc、Eclipse がインストール済みであること、テレメトリー・チャネルが構成済みであること、および PubSync.java をコーディングして実行したことが想定されています。 PubSync.java の稼働中のバージョンを含む Eclipse ワークスペースが必要です。

このタスクについて

このタスクは、478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』で作成されたパブリッシャー・クライアント PubSync.java をベースとして使用します。 SSL を使用するためには、少しだけ変更が必要です。 508 ページの図 109 および 508 ページの図 110 を参照してください。

手順

1. `com.ibm.mq.id` パッケージ内に `PubSync.java` のコピーを作成して、そのコピーの名前を `PubSyncSSL.java` にします。

   `com.ibm.mq.id` パッケージ内に PubSync.java を作成する手順については、478 ページの『Java を使用した最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』を参照してください。
2. `Example.SSLAddress` を、SSL 構成のために使用するように構成したテレメトリー・チャネルのソケット・アドレスに設定します。
3. クライアント・コンストラクターのソケット・アドレス・パラメーターを、`Example.SSLAddress` を使用するように変更します。

   ```
   MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
   ```
4. `PubSyncSSL.java` 内の `MqttConnectOptions.SSLProperties` を設定して、`MqttConnectOptions` を `MqttClient.connect` のパラメーターとして渡します。

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setSSLProperties(Example.getSSLSettings());
   client.connect(conOptions);
   ```

Example.java 内の定数セットを使用している、イタリック体のコード PubSyncSSL.Java を参照してください。

例

SSL を追加するための PubSync.java に対する変更は、508 ページの図 109 でイタリック体で示されています。

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setSSLProperties(Example.getSSLSettings());
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("SSL Properties" + conOptions.getSSLProperties());
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

図 *109. PubSyncSSL.Java*

Example.java に対する変更は、508 ページの図 110 に示されています。

```
 public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");

 public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore", "C:\\Certificates\\SSClientKey.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword", "password");
    properties.setProperty("com.ibm.ssl.trustStore", "C:\\Certificates\\SSClientTrust.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword", "password");
    return properties;
```

図 *110. Example.java* に対する変更

テレメトリー・チャネルの認証

クライアントは、チャネル上のメッセージ・フローの内容を暗号化するため、およびクライアントが正しいテレメトリー・チャネルに接続するようにするために、テレメトリー・チャネルを認証します。 サーバーはクライアントを認証しません。

このタスクについて

いくつかの異なる鍵ストア・エディターを使用して、自己署名証明書を作成および管理できます。 このタスクでは、JRE の一部であるコマンド・ラインの **keytool** コマンドを使用します。 WebSphere MQ に同

梱されている GUI ツール **iKeyman** を使用して、鍵ストアを参照し、鍵を生成することができます。 コマンド **strmqikm** を使用して **iKeyman** を起動します。

手順

1. 「**新規テレメトリー・チャネル**」ウィザードを使用して、SSL 接続を必要とするテレメトリー・チャネル SSLSSOptClients を作成します。 このチャネルは、匿名クライアントを受け入れます。

   以下の構成スタンザからチャネル構成を適合させます。 テレメトリー・プロパティー・ファイルは、直接編集せずに、ウィザードを使用してください。

   ```
   com.ibm.mq.MQXR.channel/SSLSSOptClients: \
   com.ibm.mq.MQXR.Port=8883;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. クライアントがテレメトリー・チャネルを認証するための鍵を生成します。

   a) 以下のようにして、新しい鍵ストア SSServerOptKey.jks 内にテレメトリー・チャネルのための自己署名鍵ペアを生成します。

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerOptKey.jks -storepass password -keypass password
   ```

   b) 以下のように -rfc オプションを使用して、そのパブリック証明書を ASCII ファイルとしてエクスポートします。

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerOptKey.jks -storepass password -rfc
   ```

   Windows でタスクを実行している場合は、SSServerPublic.cer をダブルクリックしてその内容を検査してください。

   c) 以下のようにして、パブリック証明書を新しいクライアント・トラストストア SSClientTrust.jks にインポートします。

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

   d) 空のクライアント鍵ストア SSClientKey.jks を作成します。

   Keytool には、空の鍵ストアを作成するコマンドがありません。 次のいずれかを選択できます。

   i) **strmqikm** を実行して鍵ストア SSClientKey.jks を作成するが、鍵は追加しない。

   ii) 509 ページの『テレメトリー・チャネルおよびクライアントの認証』のステップ 3a を実行するが、鍵はまだ使用しない。

テレメトリー・チャネルおよびクライアントの認証

クライアントはテレメトリー・チャネルを認証し、テレメトリー・チャネルはそれに接続するクライアントを認証します。 チャネルの上のメッセージ・フローは暗号化されます。

このタスクについて

いくつかの異なる鍵ストア・エディターを使用して、自己署名証明書を作成および管理できます。 このタスクでは、JRE の一部であるコマンド・ラインの **keytool** コマンドを使用します。 WebSphere MQ に同梱されている GUI ツール **iKeyman** を使用して、鍵ストアを参照し、鍵を生成することができます。 コマンド **strmqikm** を使用して **iKeyman** を起動します。

テレメトリー・チャネルは、508 ページの『テレメトリー・チャネルの認証』のタスクとは異なる鍵ストアによって構成されます。 同じ鍵ストアを使用することにより、鍵を鍵ストアに追加するステップ510 ページの『2』を省略できます。

手順

1. 「**新規テレメトリー・チャネル**」ウィザードを使用して、SSL 接続を必要とするテレメトリー・チャネル SSLSSReqClients を作成します。 このチャネルは、認証されたクライアントだけを受け入れます。

   以下の構成スタンザからチャネル構成を適合させます。

   ```
   com.ibm.mq.MQXR.channel/SSLSSReqClients: \
   com.ibm.mq.MQXR.Port=8884;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. クライアントがテレメトリー・チャネルを認証するための鍵を生成します。

   a) 以下のようにして、新しい鍵ストア SSServerReqKey.jks 内にテレメトリー・チャネルのための自己署名鍵ペアを生成します。

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerReqKey.jks -storepass password -keypass password
   ```

   b) 以下のように -rfc オプションを使用して、そのパブリック証明書を ASCII ファイルとしてエクスポートします。

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerReqKey.jks -storepass password -rfc
   ```

   Windows でタスクを実行している場合は、SSServerPublic.cer をダブルクリックしてその内容を検査してください。

   c) 以下のようにして、パブリック証明書を新しいクライアント・トラストストア SSClientTrust.jks にインポートします。

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

3. テレメトリー・チャネルがクライアントを認証するための鍵を生成します。

   a) 以下のようにして、新しい鍵ストア SSClientKey.jks 内にクライアントのための自己署名鍵ペアを生成します。

   ```
   Keytool -genkey -noprompt -alias SSClientPrivate
           -dname "CN=mqttclient.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSClientKey.jks -storepass password -keypass password
   ```

   b) 以下のように -rfc オプションを使用して、そのパブリック証明書を ASCII ファイルとしてエクスポートします。

   ```
   Keytool -export -noprompt -alias SSClientPrivate -file SSClientPublic.cer
           -keystore SSClientKey.jks -storepass password -rfc
   ```

   Windows でタスクを実行している場合は、SSClientPublic.cer をダブルクリックしてその内容を検査してください。

   c) 以下のようにして、パブリック証明書をサーバー鍵ストア SSServerReqKey.jks にインポートします。

   ```
   Keytool -import -noprompt -alias SSClientPublic -file SSClientPublic.cer
           -keystore SSServerReqKey.jks -storepass password
   ```

テレメトリー・チャネルは、秘密鍵および信頼される証明書の両方に同じストアを使用します。

証明書チェーンを使用する SSL テレメトリー接続の認証

認証局から取得したかまたは独自の認証手順を実装することによって取得した署名証明書を使用して、SSL 接続を認証します。 テレメトリー・チャネルを認証するか、またはテレメトリー・チャネルとそれに接続するクライアントとを認証するオプションがあります。 接続の上のメッセージ・フローは暗号化されます。

始める前に

開始する前にタスク 506 ページの『自己署名証明書を使用する SSL テレメトリー接続の認証』を行い、PubSyncSSL.Java が自己署名証明書を使用するセキュアな TCP/IP 接続での作業を開始するようにします。

このタスクについて

このタスクでは、506 ページの『自己署名証明書を使用する SSL テレメトリー接続の認証』内の 508 ページの『テレメトリー・チャネルの認証』および 509 ページの『テレメトリー・チャネルおよびクライアントの認証』のタスクを変更して、証明書チェーンによって認証された鍵の作業をします。

このタスクの証明書を認証局から取得すること、または http://www.openca.org/ などの Web サイトを使用して証明書を取得することができます。 民営の認証局は通常、短期間の試験的な証明書を無料で提供します。 このタスクは、民営の認証局から取得した証明書を使用してテストされました。

別のオプションは、https://www.openssl.org/などの Web サイトから入手したツールを使用して、独自の認証プロセスを構築し、それをコンピューターで実行することです。

JRE cacerts トラストストアは、このタスクでは使用されません。 511 ページの『テレメトリー・チャネルの認証』の作業では、指定されたトラストストアを使用する代わりに、クライアントの JRE cacerts トラストストアを使用できます。 証明書チェーンは、すでにクライアントの cacerts ストア内に独自のルート証明書を持つ、よく知られた認証局によって署名される場合があります。 この場合には、クライアントのトラストストアを指定しないでください。 クライアントに複数の JRE がインストールされている場合には、必ず適切な cacerts ストアを管理するようにしてください。

手順

1. まだそのようにしていなければ、タスク 507 ページの『SSL を使用するための PubSync.java の変更』を行って、SSL を使用するように PubSync.java を変更します。
2. テレメトリー・チャネルを構成して、SSL を使用する鍵ストアを作成します。

   テレメトリー・チャネルだけを認証するか、またはチャネルとそれに接続するクライアントとを認証します。

   - タスク 511 ページの『テレメトリー・チャネルの認証』 を実行して SSL に接続し、テレメトリー・チャネルを認証します。
   - タスク 513 ページの『テレメトリー・チャネルおよびクライアントの認証』 を実行して SSL に接続し、テレメトリー・チャネルとそれに接続するクライアントとを認証します。
3. テレメトリー (MQXR) サービスを停止して再始動することにより、テレメトリー・チャネルの構成に対する変更点を取得します。
4. クライアント・プログラムを実行して、構成が機能するかどうかを確認します。

テレメトリー・チャネルの認証

クライアントは、チャネル上のメッセージ・フローの内容を暗号化するため、およびクライアントが正しいテレメトリー・チャネルに接続するようにするために、テレメトリー・チャネルを認証します。 サーバーはクライアントを認証しません。

このタスクについて

いくつかの異なる鍵ストア・エディターを使用して、証明書を作成および管理できます。 このタスクでは、JRE の一部であるコマンド・ラインの **keytool** コマンドを使用します。 WebSphere MQ に同梱されている GUI ツール **iKeyman** を使用して、鍵ストアを参照し、鍵を生成することができます。 コマンド **strmqikm** を使用して **iKeyman** を起動します。

手順

1. 「**新規テレメトリー・チャネル**」ウィザードを使用して、SSL 接続を必要とするテレメトリー・チャネル SSLCAOptClients を作成します。 このチャネルは、匿名クライアントを受け入れます。

   以下の構成スタンザからチャネル構成を適合させます。 テレメトリー・プロパティー・ファイルは、直接編集せずに、ウィザードを使用してください。

   ```
   com.ibm.mq.MQXR.channel/SSLCAOptClients: \
   com.ibm.mq.MQXR.Port=8885;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. クライアントがテレメトリー・チャネルを認証するために CA が署名した鍵を生成します。

   a) 以下のようにして、新しい鍵ストア SSServerOptKey.jks 内にテレメトリー・チャネルのための自己署名鍵ペアを生成します。

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverOpt.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   一部の認証局で必要とされるので、鍵のアルゴリズムは RSA に設定されます。 証明書の共通名は固有でなければなりません。 一部の認証局は、同一の共通名キーを持つ複数の鍵を発行しません。

   b) 証明書署名要求 (CSR) を ASCII ファイルとして作成します。

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerOptKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   c) 認証局ソフトウェアを実行するか、またはその Web サイトにログオンします。 CSR ファイルを要求されるとき、CAServerOptKey.csr の内容を貼り付けます。

   d) 認証局は 1 つまたは 2 つの証明書、および署名応答ファイルを ASCII ファイルとして戻します。 以下のように、この内容を 2 つまたは 3 つのファイルに貼り付けます。

   ルート証明書
   CARoot.cer に貼り付けます。

   中間証明書 (Intermediate certificate)
   CAInter.cer に貼り付けます。

   サーバー署名の応答ファイル
   CAServerOpt.rsp に貼り付けます。

   JRE 証明書ストアはこのタスクで使用されません。 ルート証明書 1 つと CA からの署名された応答を受信した場合は、ルート証明書と署名された応答を以下の手順で使用します。 ルートおよび中間証明書を受信した場合は、中間証明書と署名された応答を使用します。

   e) 署名されたサーバー応答を、認証要求を発行したサーバーの鍵ストアで受け取ります。

   応答を受信すると、自己署名証明書は変更されて CA により署名されます。 応答の受信前と後で鍵ストアの証明書を見ると、署名者が変更されています。 そうでない場合は、鍵管理ツールによりエラーが報告されます。 証明書を使用する前に検査して、署名者が現在 CA であることを確認してください。

```
Keytool -import -noprompt -alias CAServer -file CAServerOpt.rsp
        -keystore CAServerOptKey.jks -storepass password
```

iKeyman など、鍵管理ソフトウェアによっては、 応答ファイルをインポートではなく受信します。

f) CA 証明書をクライアントのトラストストアにインポートします。

CA から証明書を 2 つ受信した場合は中間証明書、証明書を 1 つのみ受信した場合はルート証明書をインポートします。

次のいずれかの場合:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

または:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

テレメトリー・チャネルおよびクライアントの認証

クライアントはテレメトリー・チャネルを認証し、テレメトリー・チャネルはそれに接続するクライアントを認証します。 チャネルの上のメッセージ・フローは暗号化されます。

このタスクについて

いくつかの異なる鍵ストア・エディターを使用して、証明書を作成および管理できます。 このタスクでは、JRE の一部であるコマンド・ラインの **keytool** コマンドを使用します。 WebSphere MQ に同梱されている GUI ツール **iKeyman** を使用して、鍵ストアを参照し、鍵を生成することができます。 コマンド **strmqikm** を使用して **iKeyman** を起動します。

テレメトリー・チャネルは、511 ページの『テレメトリー・チャネルの認証』のタスク内のものとは異なる鍵ストアによって構成されます。 同じ鍵ストアを使用することにより、鍵を鍵ストアに追加するステップ 513 ページの『2』 を省略できます。

手順

1. 「**新規テレメトリー・チャネル**」ウィザードを使用して、SSL 接続を必要とするテレメトリー・チャネル `SSLCAReqClients` を作成します。 このチャネルは、認証されたクライアントだけを受け入れます。

   以下の構成スタンザからチャネル構成を適合させます。 テレメトリー・プロパティー・ファイルは、直接編集せずに、ウィザードを使用してください。

   ```
   com.ibm.mq.MQXR.channel/SSLCAReqClients: \
   com.ibm.mq.MQXR.Port=8886;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. クライアントがテレメトリー・チャネルを認証するために CA が署名した鍵を生成します。

   a) 以下のようにして、新しい鍵ストア `CAServerReqKey.jks` 内にテレメトリー・チャネルのための自己署名鍵ペアを生成します。

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerReqKey.jks -storepass password -keypass password
   ```

   一部の認証局で必要とされるので、鍵のアルゴリズムは RSA に設定されます。 証明書の共通名は固有でなければなりません。 一部の認証局は、同一の共通名キーを持つ複数の鍵を発行しません。

   b) 証明書署名要求 (CSR) を ASCII ファイルとして作成します。

```
Keytool -certreq -noprompt -alias CAServer -file CAServerReqKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAServerReqKey.jks -storepass password -keypass password
```

c) 認証局ソフトウェアを実行するか、またはその Web サイトにログオンします。 CSR ファイルを要求されるとき、CAServerReqKey.csr の内容を貼り付けます。

d) 認証局は 1 つまたは 2 つの証明書、および署名応答ファイルを ASCII ファイルとして戻します。 以下のように、この内容を 2 つまたは 3 つのファイルに貼り付けます。

ルート証明書
CARoot.cer に貼り付けます。

中間証明書 (Intermediate certificate)
CAInter.cer に貼り付けます。

サーバー署名の応答ファイル
CAServerReq.rsp に貼り付けます。

JRE 証明書ストアはこのタスクで使用されません。 ルート証明書 1 つと CA からの署名された応答を受信した場合は、ルート証明書と署名された応答を以下の手順で使用します。 ルートおよび中間証明書を受信した場合は、中間証明書と署名された応答を使用します。

e) 署名されたサーバー応答を、認証要求を発行したサーバーの鍵ストアで受け取ります。

応答を受信すると、自己署名証明書は変更されて CA により署名されます。 応答の受信前と後で鍵ストアの証明書を見ると、署名者が変更されています。 そうでない場合は、鍵管理ツールによりエラーが報告されます。 証明書を使用する前に検査して、署名者が現在 CA であることを確認してください。

```
Keytool -import -noprompt -alias CAServer -file CAServerReq.rsp
        -keystore CAServerReqKey.jks -storepass password
```

iKeyman など、鍵管理ソフトウェアによっては、 応答ファイルをインポートではなく受信します。

f) CA 証明書をクライアントのトラストストアにインポートします。

CA から証明書を 2 つ受信した場合は中間証明書、証明書を 1 つのみ受信した場合はルート証明書をインポートします。

次のいずれかの場合:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

または:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

3. テレメトリー・チャネルがクライアントを認証するために CA が署名した鍵を生成します。

a) 以下のようにして、新しい鍵ストア CAClientKey.jks 内にクライアントのための自己署名鍵ペアを生成します。

```
Keytool -genkey -noprompt -alias CAClientPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

一部の認証局で必要とされるので、鍵のアルゴリズムは RSA に設定されます。 証明書の共通名は固有でなければなりません。 一部の認証局は、同一の共通名キーを持つ複数の鍵を発行しません。

b) 証明書署名要求 (CSR) を ASCII ファイルとして作成します。

```
Keytool -certreq -noprompt -alias CAClient -file CAClientKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

c) 認証局ソフトウェアを実行するか、またはその Web サイトにログオンします。 CSR ファイルを要求されるとき、CAClientKey.csr の内容を貼り付けます。

d) 認証局は 1 つまたは 2 つの証明書、および署名応答ファイルを ASCII ファイルとして戻します。 以下のように、この内容を 2 つまたは 3 つのファイルに貼り付けます。

ルート証明書
CARoot.cer に貼り付けます。

中間証明書 (Intermediate certificate)
CAInter.cer に貼り付けます。

クライアント署名の応答ファイル
CAClient.rsp に貼り付けます。

JRE 証明書ストアはこのタスクで使用されません。 ルート証明書 1 つと CA からの署名された応答を受信した場合は、ルート証明書と署名された応答を以下の手順で使用します。 ルートおよび中間証明書を受信した場合は、中間証明書と署名された応答を使用します。

e) 署名されたクライアント応答を、認証要求を発行したクライアントの鍵ストアで受け取ります。

応答を受信すると、自己署名証明書は変更されて CA により署名されます。 応答の受信前と後で鍵ストアの証明書を見ると、署名者が変更されています。 そうでない場合は、鍵管理ツールによりエラーが報告されます。 証明書を使用する前に検査して、署名者が現在 CA であることを確認してください。

```
Keytool -import -noprompt -alias CAClient -file CAClient.rsp
        -keystore CAClientKey.jks -storepass password
```

iKeyman など、鍵管理ソフトウェアによっては、 応答ファイルをインポートではなく受信します。

f) CA 証明書をサーバーの鍵ストアにインポートします。

CA から証明書を 2 つ受信した場合は中間証明書、証明書を 1 つのみ受信した場合はルート証明書をインポートします。

次のいずれかの場合:

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAServerReqKey.jks -storepass password
```

または:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAServerReqKey.jks -storepass password
```

C を使った最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成

MQTT クライアントのパブリッシャー・アプリケーションを作成するためのステップを、チュートリアルの形で示します。 C コードの各行を説明します。 このタスクの終わりには、MQTT パブリッシャーを作成したことになります。

始める前に

開発されるクライアント・アプリケーションは、クライアント MQTT v3 の C クライアント・ライブラリーを使用します。 アプリケーションは WebSphere MQ Telemetry デーモン (デバイス用) に接続して、メッセージをパブリッシュします。 クライアントと WebSphere MQ Telemetry の通信の例については、最初のパブリッシャーの作成を参照してください。

このタスクについて

この例は、パブリッシュ・アプリケーション pubsync.c です。 プログラム pubsync.c は、ペイロード Hello World! を持つメッセージをトピック MQTT Example にパブリッシュし、パブリケーションがデーモンに送達されたことの確認を待機します。

簡単にするために、使用されているいくつかの関数からの戻りコードについては、正常な完了をテストされていません。 実動コードでは、戻りコードを調べてプログラムが期待どおり動作することを確認することができます。 予期しないエラーが発生した場合は、適切なアクションを取る必要があります。

MQTT Example のサブスクライバーをセットアップすることで、アプリケーションが機能することを確認できます。

選択した C 開発環境を使用して、クライアントの開発、ビルド、および実行を行ってください。 必要であれば、コードを例から直接コピーすることができます。

手順

1. 空のソース・ファイル pubsync.c を新規作成します。
2. ファイル settings.h を作成します。 図 2 のコードをファイルにコピーします。

   プログラムで使用されるすべてのパラメーターは、settings.h に定義されています。 ファイル内の値を変更することにより、値をオーバーライドできます。
3. 以下のステップでは、コードを説明します。 ステップに従うか、あるいは図 1 のコードを pubsync.c にコピーします。
4. 必要な標準ライブラリーと、MQTTClient.h および settings.h ファイルに関して、ヘッダー・ファイル include ステートメントを追加します。

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "string.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. main() 関数の定義を開始します。

   ```
   int main(int argc, char* argv[])
   {
   ```
6. プログラムで使用されるローカル変数を定義します。

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_message pubmsg;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   注: MQTTClient_connect 関数は接続オプションを必要とします。 MQTTClient_connectOptions_initializer には、デフォルト・オプションが含まれます。
7. クライアントを作成します。

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - & client は、新しく作成されたクライアントのハンドルを指すポインターです。 この関数が 0 の戻りコードで戻される場合、新しいクライアントへのハンドルが含まれています。 この例は、成功を想定しています。 実動コードで正常に完了するかどうか、エラー・コードをテストします。
   - ADDRESS は、デーモンが着信クライアント接続要求をモニターする MQTT ポートの URI です。
   - CLIENTID は、デーモンに対するクライアントの識別に使用される名前です。 アクティブ・クライアントにはそれぞれ固有の名前が必要です。 実行中の 2 つのクライアントのクライアント ID が重複していると、両方のクライアントに例外がスローされ、一方のクライアントが強制終了します。 この名前は、クライアント tjhat が切断後に再接続していることを認識するために、デーモンによって使用されます。 クライアント ID を参照してください。

- MQTTCLIENT_PERSISTENCE_NONE は、クライアントの状態がメモリー内に保持され、システム障害の発生時に失われることを指定します。 MQTTCLIENT_PERSISTENCE_DEFAULT は、ファイル・システム・ベースの持続性を指定します。 これにより、障害からいくらか保護されます。 より特殊化されたアプリケーションでは MQTTCLIENT_PERSISTENCE_USER を使用することができます。 これは、独自の持続性メカニズムを実装するためのインターフェースを提供します。 詳しくは、MQTTClientPersistence.h の API 資料を参照してください。 持続性が必要かどうかは、アプリケーション設計の問題です。 詳しくは、メッセージの持続性を参照してください。
- MQTT のデフォルト・デーモン TCP/IP ポートは 1883 です。 この例では、デフォルト・アドレスは tcp://localhost:1883 に設定されます。
- MQTTClient_connect 関数を呼び出すまで、メッセージ処理は行われません。

8. クライアントをデーモンに接続します。

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
     printf("Failed to connect, return code %d\n", rc);
     exit(-1);
}
```

- MQTTClient_connect 関数が呼び出され、クライアント・ハンドルとポインターが引数として接続オプションに渡されます。
- 接続要求を成功させるために、MQTTClient_connect 呼び出しからの戻りコードがテストされます。
- MQTTClient_connect が失敗した場合、プログラムはエラー・コード -1 で終了します。
- アプリケーションの接続後、パブリッシュとサブスクライブを開始することができます。
- TCP/IP 接続が閉じられないように、小さな「キープアライブ」メッセージが 20 秒おきに送信されます。 このオプションは conn_opts.keepAliveInterval によって設定されます。
- conn_opts.cleansession が true に設定されるので、前回の接続から残っている未完了メッセージの完了を確認せずにセッションが開始されます。 詳しくは、クリーン・セッションを参照してください。
- 接続の遺言メッセージは作成されません。 詳しくは、遺言を参照してください。

9. *MQTTClient_message* 構造体にデータを取り込んで、メッセージ・ペイロードとその属性を定義します。

```
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
```

- PAYLOAD はメッセージの内容です。
- 例ではストリングのペイロードが使用されていますが、MQTT ペイロードはバイト配列です。 ペイロードのサイズを指定するには、ストリングの長さが必要です。
- 例では QoS=1 メッセージがパブリッシュされるため、それに応じて値を設定します。
- メッセージはデーモンによって保存されないため、保存属性は false (0) に設定されます。 詳しくは、保存パブリケーションを参照してください。

10. メッセージをパブリッシュします。

```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
```

- パブリッシュ関数は、デーモンに送信されるクライアント、トピック、およびペイロードを指定します。
- TOPIC は、 settings.h で MQTT Example として定義されます。
- 関数には、*MQTTClient_deliveryToken* へのポインターも渡されます。 関数が戻されるときに、メッセージを表すトークンがこのポインターに取り込まれます。

- これでメッセージを安全に MQTT クライアントに転送できるようになりますが、デーモンにはまだ転送されません。 メッセージが QoS=1 または 2 であれば、送達が完了する前にクライアントに障害が起こった場合には、メッセージはローカル保管されます。
- この関数は、実動コードでの正常な完了をテストできるエラー・コードを戻します。

11. サーバーからの確認応答を待ちます。

```
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

- pubsync.c の例では、メッセージが送達されたことを確認するために、サーバーからの確認応答を待機します。
- クライアントおよびトークンの引数は、プログラムが完了を待機する特定のメッセージを識別します。
- TIMEOUT は、メッセージの送達の完了をプログラムが待機する時間の長さを制限します。 C を使った MQ Telemetry Transport の非同期パブリッシャーの作成タスクは、コールバック関数を使用して、待機なしで確認応答を受信する方法を示しています。
- この関数は、実動コードでの正常な完了をテストできるエラー・コードを戻します。

12. デーモンからクライアントを切断します。

```
MQTTClient_disconnect(client, 10000);
```

- クライアントはサーバーから切断し、未完了メッセージを完了するためにコールバック関数 (この例では使用されていない) を待ちます。
- 2 番目の引数は、静止タイムアウトをミリ秒で指定します。 この例では、切断前に行わなければならない他のすべての作業を完了するために最大 10 秒間待機します。
- この関数は、実動コードでの正常な完了のためにテストする必要のあるエラー・コードを戻します。

13. クライアントで使用されるメモリーを解放し、プログラムを終了します。

```
MQTTClient_destroy(&client);
}
```

タスクの結果

このクライアントによって送信されるパブリケーションを確認するには、MQTT Example トピックへのサブスクライバーを作成します。 詳しくは、C を使った MQ Telemetry Transport のサブスクライバーの作成を参照してください。

例

図 1 は、手順で説明されているコードの完全なリストです。 図 2 の settings.h ファイルを使用すると、pubsync.c で使用されるデフォルトのパラメーターを変更することができます。

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"

int pubsync_main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID,  MQTTCLIENT_PERSISTENCE_NONE, NULL);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
          printf("Failed to connect, return code %d\n", rc);
         exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for up to %d seconds for publication of %s\n"
          "on topic %s for client with ClientID: %s\n",
          TIMEOUT/1000, PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

図 *111. pubsync.c*

```
#define ADDRESS   "tcp://localhost:1883"
#define CLIENTID  "ExampleClientPub"
#define TOPIC     "MQTT Example"
#define PAYLOAD   "Hello World!"
#define QOS      1
#define TIMEOUT   10000L
```

図 *112. settings.h*

C を使った MQ Telemetry Transport の非同期パブリッシャーの作成

MQTT クライアントの非同期パブリッシャー・アプリケーションを作成するためのステップを、チュートリアルの形で示します。 C コードの各行を説明します。 このタスクの終わりには、MQTT 非同期パブリッシャーを作成したことになります。

このタスクでは、最初のパブリッシャー・アプリケーションを、チュートリアルに従って変更します。 この変更によってアプリケーションは、送達確認を待たずにパブリケーションを送信できるようになります。送達確認は、作成するコールバック関数によって受信されます。

始める前に

開発されるクライアント・アプリケーションは、クライアント MQTT v3 の C クライアント・ライブラリーを使用します。 アプリケーションは WebSphere MQ Telemetry デーモン (デバイス用) に接続して、メッセージをパブリッシュします。 クライアントと WebSphere MQ Telemetry の通信の例については、最初のパブリッシャーの作成を参照してください。

このタスクについて

この例は、pubasync.c というパブリッシュ・アプリケーションです。 プログラム pubasync.c は、パブリケーションがデーモンに送達されたことの確認を待たずに、ペイロード Hello World! を含むメッセージをトピック MQTT Example にパブリッシュします。 送達確認は、コールバック関数 MQTTClient_deliveryComplete で受け取られます。

簡単にするために、使用されているいくつかの関数からの戻りコードについては、正常な完了をテストされていません。 実動コードでは、戻りコードを調べてプログラムが期待どおり動作することを確認することができます。 予期しないエラーが発生した場合は、適切なアクションを取る必要があります。

MQTT Example へのサブスクライバーをセットアップすることにより、アプリケーションが機能することを確認できます。

選択した C 開発環境を使用して、クライアントの開発、ビルド、および実行を行ってください。

手順 のステップでは、 515 ページの『C を使った最初の MQ Telemetry Transport パブリッシャー・アプリケーションの作成』から pubsync.c アプリケーションを変更します。 必要であれば、コードを例から直接コピーすることができます。

手順

1. 空のソース・ファイル callback.h を新規作成します。
2. 図 2 のコードをファイルにコピーします。
   - callback.h は、非同期クライアントの操作に必要な 3 つのコールバック・メソッドを宣言します。
   - 変数 *deliveredtoken* も宣言されます。 これは、異なる実行のスレッドでメインプログラムとコールバックによってアクセスされます。 そのため、volatile として宣言されます。 コールバックを使用する際は、スレッド・セーフな方法で関連する変数にアクセスするように配慮してください。
3. 空のソース・ファイル callback.c を新規作成します。
4. 図 3 のコードをファイルにコピーします。
   - callback.c は、非同期操作のためにクライアントによって使用される delivered、msgarrvd、および connlost という 3 つのコールバック・メソッドを実装しています。
5. pubasync.c に他の include を追加した後、callback.h に以下の include ステートメントを追加します。

   ```
   #include "callback.h"
   ```

6. pubsync.c の内容を新規ファイル pubasync.c. にコピーします。
7. pubasync.c の MQTTClient_connect 関数呼び出しの直前に、クライアントのコールバック・メソッドを設定します。

   ```
   MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
   ```

   - 3 つのコールバック関数を指定する必要があります。 これらの関数は、callback.c に実装されています。
   - 一致するサブスクリプションによりメッセージがクライアントに送信されると、MQTTClient_messageArrived が呼び出されます。 受信メッセージがクライアント・アプリケーションによって正常に受信されたときに、これは true を戻す必要があります。 false を戻す場合、アプリケーションにメッセージ受信の問題があることをクライアントに示します。
   - クライアントがサーバーへの接続を失うと、MQTTClient_connectionLost が呼び出されます。
   - QoS1 または QoS2 メッセージが到着してサーバーによって確認されると、MQTTClient_deliveryComplete が呼び出されます。 QoS0 メッセージの場合は呼び出されません。 例の中のこの関数は、送達メッセージからのトークンを *deliveredtoken* に保存して、メッセージが到着したことを示します。
   - クライアントがサーバーから切断される間に MQTTClient_setCallbacks を呼び出す必要があります。
   - 2 番目の引数を使用すると、コンテキスト情報をコールバック関数に渡すことができます。 これはこの例では使用されないため、NULL に設定されます。
8. MQTTClient_publishMessage の呼び出しの直前に *deliveredtoken* をクリアします。 MQTTClient_deliveryComplete は、トークンを受信したときに *deliveredtoken* を設定します。

```
deliveredtoken = 0;
```

9. MQTTClient_waitForCompletion 呼び出しと、その後の printf ステートメントを削除し、元のトークンとコールバックで受信されるトークンとの一致を待機するループで置き換えます。

```
while(deliveredtoken != token);
```

これは 1 つの例であり、実動コード設計で対応しなければならないいくつかの状況は処理されません。例えば、以下のような状況があります。

- 送達が完了しない場合に、タイムアウトを実装できる
- 複数のメッセージが未完了の可能性がある。 (サンプル・プログラムで検査できる送達トークンは、一度に 1 つのみ。)

10. デーモンからクライアントを切断します。

```
MQTTClient_disconnect(client, 10000);
```

- クライアントはサーバーから切断し、未完了メッセージを完了するためにコールバック関数を待ちます。
- 2 番目の引数は、静止タイムアウトをミリ秒で指定します。 この例では、切断前に行わなければならない他のすべての作業を完了するために最大 10 秒間待機します。
- この関数は、実動コードでの正常な完了のためにテストする必要のあるエラー・コードを戻します。

11. クライアントで使用されるメモリーを解放し、プログラムを終了します。

```
MQTTClient_destroy(&client);
}
```

タスクの結果

このクライアントによって送信されるパブリケーションを確認するには、MQTT Example トピックへのサブスクライバーを作成します。 詳しくは、MQ Telemetry Transport のサブスクライバーの作成を参照してください。

例

pubasync.c、 callbacks.c 、および callbacks.h は、 手順で説明されているコードの完全なリストです。

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    deliveredtoken = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for publication of %s\n"
           "on topic %s for client with ClientID: %s\n", PAYLOAD, TOPIC, CLIENTID);
    while(deliveredtoken != token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

図 *113. pubasync.c*

```
MQTTClient_deliveryComplete delivered;
MQTTClient_messageArrived msgarrvd;
MQTTClient_connectionLost connlost;

extern volatile MQTTClient_deliveryToken deliveredtoken;
```

図 *114. callback.h*

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt)
{
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
    }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

図 *115. callback.c*

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientPub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

図 *116. settings.h*

C を使用した MQ Telemetry Transport のサブスクライバーの作成

MQTT クライアントのサブスクライバー・アプリケーションを作成するためのステップを、チュートリアルの形で説明します。 C コードの各行を説明します。 このタスクの終わりには、MQTT サブスクライバーを作成したことになります。

始める前に

開発されるクライアント・アプリケーションは、クライアント MQTT v3 の C クライアント・ライブラリーを使用します。 アプリケーションは WebSphere MQ Telemetry デーモン (デバイス用) に接続して、メッセージをパブリッシュします。 クライアントと WebSphere MQ Telemetry の通信の例については、最初のパブリッシャーの作成を参照してください。

このタスクについて

この例は、subscribe.c というサブスクライバー・アプリケーションです。 プログラム subscribe.c は、トピック MQTT Example にサブスクライブし、ユーザーがプログラムを終了するまでサブスクリプションに一致するパブリケーションを待機します。

サブスクライバーはトピックへのサブスクリプションを作成し、サブスクリプション・トピックに一致するメッセージを待ちます。 クライアントの切断中にパブリッシュされるメッセージで、前にクライアントによって作成されたサブスクリプションに一致するものを、クライアントの再接続時に受信することができます。 デバイス用の WebSphere MQ Telemetry (MQXR) サービスまたはデーモンは、前にクライアント

ID によって接続されていたクライアントを認識します。 詳しくは、クライアント ID を参照してください。 MQTTClient_connectOptions.cleansession ブール属性は、前に送信されたパブリケーションを受信するかどうかを制御します。 詳細については、532 ページの『クリーン・セッション』を参照してください。

簡単にするために、使用されているいくつかの関数からの戻りコードについては、正常な完了をテストされていません。 実動コードでは、戻りコードを調べてプログラムが期待どおり動作することを確認することができます。 予期しないエラーが発生した場合は、適切なアクションを取ることができます。

前述のパブリッシュのプログラム例を使用して、一致するパブリケーションをデバイス用 WebSphere MQ Telemetry デーモンに送信することができます。 あるいは、クライアントを WebSphere MQ Telemetry チャネルに接続する場合には、WebSphere MQ エクスプローラーを使って MQTT Example トピックにテスト・パブリケーションを作成します。

手順 の説明では、前のタスクのいずれかで callback.c, callback.h ファイルと settings.h ファイルが既に作成されていることを前提としています。

選択した C 開発環境を使用して、クライアントの開発、ビルド、および実行を行ってください。 必要であれば、コードを例から直接コピーすることができます。

手順

1. この例の settings.h のコピーを作成し、*CLIENTID* 定義ステートメントを次のようにして変更します。

   ```
   #define CLIENTID "ExampleClientSub"
   ```

   - 同じ ID を持つ 2 つのクライアントが 1 つのサーバーへの接続を試みる場合、そのうちの 1 つが強制的に切断されます。 通常、新しい接続試行が成功し、古い方の接続が切断されます
   - ClientID を変更すると、前に作成したパブリッシュの例を使ってこのサブスクライバーにメッセージを送信することができます。
2. 空のソース・ファイル subscribe.c を新規作成します。
3. 以下のステップでは、コードを説明します。 以下のステップに従うか、527 ページの図 117 のコードをファイル subscribe.c にコピーします。
4. 必要な標準ライブラリーと、MQTTClient.h および settings.h ファイルに関して、ヘッダー・ファイル include ステートメントを追加します。

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. main() 関数の定義を開始します。

   ```
   int main(int argc, char* argv[]) {
   ```
6. プログラムで使用されるローカル変数を定義します。

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   MQTTClient_connect 関数は接続オプションを必要とします。 MQTTClient_connectOptions_initializer には、デフォルト・オプションが含まれます。
7. クライアントを作成します。

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - & client は、新しく作成されたクライアントのハンドルを指すポインターです。 この関数が 0 の戻りコードで戻った場合、ポインターには新しいクライアントへのハンドルが含まれています。 こ

の例は、成功を想定しています。 実動コードで正常に完了するかどうか、エラー・コードをテストすることができます。

- ADDRESS は、デーモンが着信クライアント接続要求をモニターする MQTT ポートの URI です。
- CLIENTID は、デーモンに対するクライアントの識別に使用される名前です。 アクティブ・クライアントにはそれぞれ固有の名前が必要です。 実行中の 2 つのクライアントのクライアント ID が重複していると、両方のクライアントに例外がスローされ、一方のクライアントが強制終了します。 この名前は、クライアントが切断に続いて再接続を行っていることを認識するためにデーモンが使用します。 クライアント ID を参照してください。
- MQTTCLIENT_PERSISTENCE_NONE は、クライアントの状態がメモリー内に保持され、システム障害の発生時に失われることを指定します。 MQTTCLIENT_PERSISTENCE#_DEFAULT は、ファイル・システム・ベースの持続性を指定し、障害をある程度防いでくれます。 より特殊化されたアプリケーションでは MQTTCLIENT_PERSISTENCE_USER を使用することができます。 これは、独自の持続性メカニズムを実装するためのインターフェースを提供します。 持続性が必要かどうかは、アプリケーション設計の問題です。 詳しくは、メッセージの持続性を参照してください。
- MQTT のデフォルト・デーモン TCP/IP ポートは 1883 です。 この例では、デフォルト・アドレスは tcp://localhost:1883 に設定されます。
- MQTTClient_connect 関数を呼び出すまで、メッセージ処理は行われません。

8. クライアントをデーモンに接続します。

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
    printf("Failed to connect, return code %d\n", rc);
    exit(-1);
}
```

- MQTTClient_connect 関数が呼び出され、クライアント・ハンドルとポインターが引数として接続オプションに渡されます。
- 接続要求を成功させるために、MQTTClient_connect 呼び出しからの戻りコードがテストされます。
- 接続の呼び出しに失敗した場合、このプログラムはエラー・コード -1 で終了します。
- アプリケーションの接続後に、パブリッシュとサブスクライブを開始することができます。
- TCP/IP 接続が閉じられないように、小さな「キープアライブ」メッセージが 20 秒おきに送信されます。 このオプションは conn_opts.keepAliveInterval によって設定されます。
- *conn_opts.cleansession* が true に設定されるので、前回の接続から残っている未完了メッセージの完了を確認せずにセッションが開始されます。 詳しくは、クリーン・セッションを参照してください。
- 接続の遺言メッセージは作成されません。 詳しくは、遺言を参照してください。

9. トピックにサブスクライブします。

```
MQTTClient_subscribe(client, TOPIC, QOS);
```

- MQTTClient_subscribe 関数を使用して、選択されたトピックにクライアント・アプリケーションをサブスクライブします。 トピック名にはワイルドカード文字を含めることができます。 詳細については、548 ページの『MQTT クライアントのトピック・ストリングおよびトピック・フィルター』を参照してください。
- QoS 設定は、このサブスクライバーに送信されるメッセージに適用される最大サービス品質を決定します。 サーバーは、この設定と元のメッセージの QoS 設定のうちの低い方の値でメッセージを送信します。
- この関数は、実動コードでの正常な完了をテストできるエラー・コードを戻します。

10. ユーザーがキーボードから「Q」という文字を入力するまでループして待機します。

```
do {
    ch = getchar();
    } while(ch!='Q' && ch != 'q');
```

プログラムはメッセージが到着するのを待ちます。 この例では、すべてのメッセージ処理がコールバック関数 MQTTClient_messageArrived で行われます。 詳細については、526 ページの『メッセージの受信』を参照してください。

11. デーモンからクライアントを切断します。

```
MQTTClient_disconnect(client, 10000);
```

- クライアントはサーバーから切断し、未完了メッセージを完了するためにコールバック関数 (この例では使用されていない) を待ちます。
- 2 番目の引数は、静止タイムアウトをミリ秒で指定します。 この例では、切断前に実行しておかなければならない他のすべての作業を完了するために、最大 10 秒間待機します。
- この関数は、実動コードでの正常な完了をテストできるエラー・コードを戻します。

12. クライアントで使用されるメモリーを解放し、プログラムを終了します。

```
MQTTClient_destroy(&client);
}
```

メッセージの受信

このタスクについて

メッセージがサーバーから到着すると、MQTTClient_messageArrived 関数が開始されます。 以下のステップでは、コードを説明します。

手順

1. コールバック関数の定義を開始します。 この定義は、MQTTClient_messageArrived 関数のテンプレートと一致する必要があります。

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
```

- context は、MQTTClient_setCallbacks 関数が呼び出されたときにクライアント・ライブラリーに渡されるコンテキストへのアクセスを提供します。 この関数はこの例では使用されません。
- topicName は、受信メッセージのパブリッシュ先のトピックへのポインターです。 ワイルドカード文字を使ってサブスクライブを行った場合、このパラメーターは、メッセージに使用される特定のトピックを識別します。
- topicLen は、トピック・ストリングの長さです。 このオプションは、トピック・ストリングに NULL 文字を組み込まなければならないユーザーのために提供されています。
- message は、メッセージ・ペイロードおよび属性が含まれている MQTTClient_message 構造体へのポインターです。

2. 使用されるローカル変数を定義します。

```
int i;
char* payloadptr;
```

この例ではこれらの変数を使用して、繰り返しによってペイロードを出力します。

3. メッセージを出力して、トピックとメッセージのペイロードを表示します。

```
printf("Message arrived\n");
printf("     topic: %s\n",topicName);
printf("   message: ");
payloadptr = message->payload;
for(i=0; i<message->payloadlen; i++){
    putchar(*payloadptr++);
    }
putchar('\n');
```

- この例では、受信ペイロードが一連の印刷可能文字であることを前提としています。

- MQTT ペイロードは、バイトの配列です。 アプリケーションは、その意味の解釈を担当します。

4. メッセージの保管に使用されたメモリーを解放します。

```
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
```

- この例では、メッセージ処理はすべてコールバック関数で行われます。
- コールバック関数を短くして、可能な限り早くその呼び出しスレッドに制御を戻します。
- プログラムのメイン部分で処理されるようにメッセージ・ポインターが渡されます。
- メインプログラムは、処理の完了時に、メッセージによって使用されたメモリーを解放する必要があります。 `MQTTClient_freeMessage()` は、*MQTTClient_message* 構造体およびメッセージ・ペイロードを保持するために使用された 2 つのメモリー・ブロックをシステムに戻す便利な関数です。 `topicName` に割り振られたメモリーを、それぞれ示されているように解放する必要があります。

5. コールバックが正常にメッセージの処理を完了した場合には、値として true を返します。

```
    return 1;
}
```

- true 値を返すと、クライアント・ライブラリーは、メッセージを正常に送達されたものとして扱うことができます。
- コールバック関数がメッセージを適切に処理できない場合は、false 値が返されます。 例えば、コールバックが処理のためにメッセージをメインプログラムのキューに置こうとしたときに、そのキューが満杯の場合は、false を返すのが妥当です。
- QoS1 および QoS2 メッセージの場合、false 値を返すと、メッセージが送達されなかったためにさらに送達が試みられます。

コード例

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc;
    int ch;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
        }
    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

    MQTTClient_subscribe(client, TOPIC, QOS);
    do {
        ch = getchar();
        } while(ch!='Q' && ch != 'q');
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

図 *117. subscriber.c*

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt) {
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
        }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
void connlost(void *context, char *cause) {
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

図 *118. callback.h*

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientSub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

図 *119. settings.h*

MQTT クライアント・プログラミングの概念

このセクションで説明される概念は、MQTT protocol・バージョン 3.1 用の Java、JavaScript、および C クライアント・ライブラリーを理解するうえで役立ちます。 この概念は、クライアント・ライブラリーに付属している API ドキュメンテーションを補足するものです。

com.ibm.micro.client.mqttv3 には、 MQTT バージョン 3.1 プロトコル用のクライアント・ライブラリーの public メソッドを提供するクラスが含まれています。 com.ibm.micro.client.mqttv3 パッケージの 1 つのバージョンと、Java SE および ME のプロトコルを実装する付属のパッケージが、IBM WebSphere MQ Telemetry のインストールと共に提供されます。 最新バージョンの MQTT クライアント・ライブラリー (Java、 JavaScript) を入手し、API 資料を表示またはダウンロードするには、"MQTT クライアント・プログラミング・リファレンス"を参照してください。

MQTT クライアントを開発して実行するには、これらのパッケージをクライアント装置にコピーまたはインストールする必要があります。 別個のクライアント・ランタイムをインストールする必要はありません。

クライアントのライセンス条件は、クライアントの接続先のサーバーに関連付けられます。

MQTT クライアント・ライブラリーは、MQTT protocol・バージョン 3.1 の参照実装です。 ユーザー独自のクライアントを、さまざまな装置プラットフォームに適したさまざまな言語で実装できます。 MQ Telemetry Transport のフォーマットおよびプロトコルを参照してください。

API ドキュメンテーションでは、クライアントがどの MQTT サーバーに接続しているかについて想定していません。 クライアントの動作は、異なるサーバーに接続するときに若干の違いが出る可能性があります。 以下の説明は、IBM WebSphere MQ テレメトリー・サービスに接続している場合のクライアントの動作について示しています。

MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

コールバック

MqttCallback インターフェースには 3 つのコールバック・メソッドがあります。 実装の例については、Callback.java を参照してください。

connectionLost(java.lang.Throwable cause)

connectionLost は、通信エラーが原因で接続が切断されたときに呼び出されます。 また、接続が確立された後にサーバー上でエラーが発生したためにサーバーが接続を切断した場合にも呼び出されます。 サーバー・エラーは、キュー・マネージャーのエラー・ログに記録されます。 サーバーはクライアントとの接続を切断し、クライアントは MqttCallback.connectionLost を呼び出します。

クライアント・アプリケーションと同じスレッドで例外としてスローされる唯一のリモート・エラーは、MqttClient.connect からの例外です。 接続の確立後にサーバーによって検出されたエラーは、 MqttCallback.connectionLost コールバック・メソッドに throwables として報告されます。

connectionLost の原因となる代表的なサーバー・エラーとして、許可エラーがあります。 例えば、トピックに関してパブリッシュが許可されていないクライアントの代わりに、テレメトリー・サーバーがそのトピックに関してパブリッシュを行おうとすることがあります。 その場合、MQCC_FAIL 条件コードをテレメトリー・サーバーに戻す原因となるものはすべて、接続切断の原因となる可能性があります。

deliveryComplete(MqttDeliveryToken token)

deliveryComplete は、送達トークンをクライアント・アプリケーションに戻すために MQTT クライアントによって呼び出されます。535 ページの『送達トークン』を参照してください。 送達トークンの利用により、コールバックはメソッド token.getMessage を使ってパブリッシュ・メッセージにアクセスすることができます。

deliveryComplete メソッドによって呼び出された後でアプリケーション・コールバックが MQTT クライアントに制御を戻すと、送達が完了します。 送達が完了するまで、QoS が 1 または 2 に設定されたメッセージはパーシスタンス・クラスによって保存されます。

deliveryComplete の呼び出しは、アプリケーションとパーシスタンス・クラスの間の同期点になります。 deliveryComplete メソッドが同じメッセージに対して 2 度呼び出されることはありません。

アプリケーション・コールバックが deliveryComplete から MQTT クライアントに戻ると、クライアントは QoS 1 または 2 のメッセージに対して MqttClientPersistence.remove を呼び出します。 MqttClientPersistence.remove は、パブリッシュされたメッセージのローカルで保管されたコピーを削除します。

トランザクション処理の観点から見ると、deliveryComplete の呼び出しは、送達をコミットする単一フェーズのトランザクションです。 コールバック中に処理が失敗した場合、クライアントの再始動時に MqttClientPersistence.remove が再び呼び出され、パブリッシュ・メッセージのローカル・コピーを削除します。 コールバックは再び呼び出されません。 送達されたメッセージのログを保管するためにコールバックを使用する場合、ログと MQTT クライアントを同期することはできません。 ログを確実に保管するには、MqttClientPersistence クラスでログを更新してください。

送達トークンとメッセージは、メイン・アプリケーション・スレッドおよび MQTT クライアントによって参照されます。 MQTT クライアントは送達の完了時に MqttMessage オブジェクトを間接参照し、クライアントの切断時に送達トークン・オブジェクトを間接参照します。 送達が完了した後、MqttMessage オブジェクトに対してガーベッジ・コレクションを実行することができます (クライアント・アプリケーションがそれを間接参照する場合)。 セッションが切断された後、送達トークンに対してガーベッジ・コレクションを実行することができます。

メッセージがパブリッシュされた後、MqttDeliveryToken および MqttMessage 属性を取得することができます。 メッセージがパブリッシュされた後で MqttMessage 属性の設定を試みた場合、結果は未定義です。
MQTT クライアントは、同じ ClientIdentifier を使用して前のセッションに再接続した場合、送達確認の処理を続行します。 532 ページの『クリーン・セッション』を参照してください。
MQTT クライアント・アプリケーションは、前回のセッションに対しては MqttClient.CleanSession を false に設定し、新規セッションでは false に設定する必要があります。 保留中の送達に関して、MQTT クライアントは新規セッションで新しい送達トークンとメッセージ・オブジェクトを作成します。 MqttClientPersistence クラスを使用してオブジェクトをリカバリーします。 古い送達トークンとメッセージへの参照をアプリケーション・クライアントが依然として保持している場合は、それらを間接参照してください。 前回のセッションで開始されてこのセッションで完了する送達がある場合、アプリケーション・コールバックが新規セッションで呼び出されます。
保留中の送達が完了するとき、アプリケーション・クライアントの接続後にアプリケーション・コールバックが呼び出されます。 アプリケーション・クライアントは、接続前に MqttClient.getPendingDeliveryTokens メソッドを使って保留中の送達を取り出すことができます。
最初に、クライアント・アプリケーションは、パブリッシュされるメッセージ・オブジェクトとそのペイロード・バイト配列を作成しました。 MQTT クライアントはそれらのオブジェクトを参照します。 メソッド token.getMessage で送達トークンによって戻されるメッセージ・オブジェクトは、クライアントによって作成されたメッセージ・オブジェクトと必ずしも同じではありません。 新しい MQTT クライアント・インスタンスが送達トークンを再作成する場合、MqttClientPersistence クラスは MqttMessage オブジェクトを再作成します。 一貫性を保つために、token.isCompleted が true の場合、メッセージ・オブジェクトがアプリケーション・クライアントと MqttClientPersistence クラスのどちらによって作成されたかに関係なく、token.getMessage は null を返します。

messageArrived(MqttTopic topic, MqttMessage message)

messageArrived は、サブスクリプション・トピックに一致したクライアントのパブリケーションが到着したときに呼び出されます。 topic は、サブスクリプション・フィルターではなくパブリケーション・トピックです。 フィルターにワイルドカードが含まれる場合、これら 2 つは異なる可能性があります。
クライアントによって作成された複数のサブスクリプションにトピックが一致する場合、クライアントはパブリケーションの複数コピーを受信します。 サブスクライブ先でもあるトピックに対してクライアントがパブリッシュを行うと、クライアントは自分のパブリケーションのコピーを受信します。
QoS が 1 または 2 に設定された状態でメッセージが送信される場合、そのメッセージは MQTT クライアントが messageArrived を呼び出す前に MqttClientPersistence クラスによって保管されます。 messageArrived は deliveryComplete のように動作します。 これは 1 つのパブリケーションにつき一度だけ呼び出され、messageArrived が MQTT クライアントに戻ったときに MqttClientPersistence.remove によってパブリケーションのローカル・コピーが削除されます。 messageArrived が MQTT クライアントに戻ると、MQTT クライアントはトピックおよびメッセージへの参照を除去します。 アプリケーション・クライアントがオブジェクトへの参照を保持していない場合、トピックおよびメッセージ・オブジェクトに対してガーベッジ・コレクションが実行されます。

コールバック、スレッド、およびクライアント・アプリケーションの同期

MQTT クライアントはメイン・アプリケーション・スレッドとは別のスレッドでコールバック・メソッドを呼び出します。 コールバック用のスレッドは、クライアント・アプリケーションではなく、MQTT クライアントによって作成されます。

MQTT クライアントはコールバック・メソッドを同期します。 一度に実行されるコールバック・メソッドのインスタンスは 1 つだけです。 同期により、どのパブリケーションが送達されたかを記録するオブジェクトの更新が容易になります。 実行される MqttCallback.deliveryComplete のインスタンスは一度に 1 つなので、追加の同期を行うことなく安全に記録を更新することができます。 また、パブリケーショ

ンが一度に 1 つだけ到着する場合も同様です。 messageArrived メソッドのコードでは、同期せずにオブジェクトを更新することができます。 別のスレッドで、記録 (または更新されるオブジェクト) を参照している場合には、記録またはオブジェクトを同期します。

送達トークンは、メイン・アプリケーション・スレッドとパブリケーションの送達の間の同期メカニズムを提供します。 メソッド token.waitForCompletion は、特定のパブリケーションの送達が完了するか、オプションのタイムアウトが満了するまで待機します。 次のように、いくつかの簡単な方法で token.waitForCompletion を使用してパブリケーションを一度に 1 つずつ処理できます。

1. パブリケーションの送達が完了するまでアプリケーション・クライアントを一時停止する。詳しくは、483 ページの図 88 を参照してください。
2. MqttCallback.deliveryComplete メソッドと同期する。 MqttCallback.deliveryComplete が MQTT クライアントに戻るときにのみ、token.waitForCompletion が再開します。 このメカニズムを使用することで、メイン・アプリケーション・スレッドでコードが実行される前に、MqttCallback.deliveryComplete で実行されるコードを同期できます。

それぞれのパブリケーションが送達されるのを待たずにパブリッシュしながら、すべてのパブリケーションが送達されたときに確認するにはどうしたらよいでしょうか? 単一スレッドでパブリッシュを行う場合には、最後に送信されるパブリケーションが最後の送達になります。

サーバーに送信される要求の同期

531 ページの表 70 では、サーバーに要求を送信する MQTT Java クライアントのメソッドについて説明します。 アプリケーション・クライアントで無期限のタイムアウトが設定されない限り、クライアントがサーバーを無限に待つことはありません。 クライアントがハングする場合、それはアプリケーション・プログラミングの問題、または MQTT クライアントの障害です。

表 *70.* サーバーに要求を送信するメソッドの同期の振る舞い

メソッド	同期	タイムアウト間隔
MqttClient.Connect	サーバーとの接続が確立されるのを待ちます。	デフォルトの 30 秒 (またはパラメーターによる設定値) になると、例外をスローします。
MqttClient.Disconnect	MQTT クライアントが行わなければならない処理を完了して TCP/IP セッションが切断されるのを待ちます。	
MqttClient.Subscribe	Subscribe または UnSubscribe メソッドの完了を待ちます。	
MqttClient.UnSubscribe		
MqttClient.Publish	MQTT クライアントに要求を渡した後、直ちにアプリケーション・スレッドに戻ります。	なし。
MqttDeliveryToken.waitForCompletion	送達トークンが戻されるのを待ちます。	無期限。またはパラメーターで設定される値。

関連概念

クリーン・セッション

MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

クリーン・セッション

MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

MqttClient.connect メソッドを使用して MQTT クライアント・アプリケーションに接続する際、クライアントはクライアント ID とサーバーのアドレスを使って接続を識別します。 サーバーは、前回のサーバーへの接続のセッション情報が保存されているかどうかを調べます。 前回のセッションがまだ存在し、cleanSession=true の場合、クライアント側とサーバー側の前回のセッション情報は消去されます。 cleanSession=false の場合、前回のセッションが再開されます。 前回のセッションが存在しない場合、新規セッションが開始されます。

注 : WebSphere MQ Administrator は開かれているセッションを強制的に閉じ、すべてのセッション情報を削除します。 クライアントが cleanSession=false でセッションを再び開く場合、新規セッションが開始されます。

パブリケーション

デフォルトの MqttConnectOptions を使用するか、クライアントに接続する前に MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にそのクライアントで保留になっているすべてのパブリケーションの送達が削除されます。

クリーン・セッションの設定は、QoS=0 で送信されるパブリケーションには影響を与えません。 QoS=1 と QoS=2 の場合、cleanSession=true を使用するとパブリケーションが失われる可能性があります。

サブスクリプション

デフォルトの MqttConnectOptions を使用するか、クライアントに接続する前に MqttConnectOptions.cleanSession を true に設定すると、クライアントの接続時にそのクライアントの古いサブスクリプションがすべて削除されます。 セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続の前に MqttConnectOptions.cleanSession を false に設定すると、クライアントによって作成されるサブスクリプションはすべて、接続する前にクライアントに存在していたすべてのサブスクリプションに追加されます。 クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

cleanSession 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。 そのデフォルト・モード cleanSession=true では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。 それに代わる cleanSession=false モードでは、サブスクリプションは永続的になります。 クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。 クライアントは、再接続時にすべての未送達パブリケーションを受信します。 接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

cleanSession モードは接続する前に設定する必要があります。モードはセッション中ずっと続きます。 その設定を変更するには、クライアントを切断し、再接続する必要があります。 モードを cleanSession=false の使用から cleanSession=true の使用に変更すると、クライアントの前回のサブスクリプションすべてと、受信されていないパブリケーションがあればそれらすべてが廃棄されます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション

トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

クライアント ID

クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

クライアント ID は、MQTT システムの管理に使用されます。 管理対象のクライアントは幾十万にもなる可能性があるので、特定のクライアントを迅速に識別する必要があります。 例えば、ある装置に誤動作が生じて、カスタマーがヘルプ・デスクに電話してそれを通知したとします。 カスタマーはどのようにして装置を識別し、管理者はどのようにその識別をクライアントが通常接続しているサーバーに対応させるのでしょうか。 各装置をクライアント ID およびサーバーにマップするデータベースを調べる必要がありますか。 装置の名前から、それが接続するサーバーを識別できますか。 MQTT クライアント接続を参照するとき、各接続にはクライアント ID のラベルが付いています。 クライアント ID を物理装置にマップするテーブルを検索する必要がありますか。

クライアント ID は、特定の装置、ユーザー、またはそのクライアントで実行しているアプリケーションを識別しますか。 カスタマーが障害のある装置を新しいものと置き換えた場合、新しい装置の ID は古い装置のものと同じですか。 管理者は新しい ID を割り振りますか。 物理装置を変更しても同じ ID を使用する場合、未送信のパブリケーションおよびアクティブなサブスクリプションは新しい装置に自動的に転送されます。

クライアント ID が固有であることをどのように確認しますか。 固有の ID を生成するシステムとともに、クライアントに ID を設定するための信頼できるプロセスを持つ必要があります。 クライアント装置はユーザー・インターフェースのない「ブラック・ボックス」である可能性があります。 装置を製造するときに、MAC アドレスを使用するなどして、クライアント ID を設定しますか。 あるいは、装置がアクティブ化される前に装置を構成するソフトウェアのインストールおよび構成のプロセスがありますか。

クライアント ID を 48 ビットの装置 MAC アドレスから作成して、ID を短くしかも固有に保つことができます。 伝送サイズが重要な問題ではなければ、残りの 17 バイトを使用してアドレスの管理を容易にすることができます。

関連概念
MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

送達トークン
遺言パブリケーション

MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

送達トークン

クライアントがトピックにでパブリッシュするときに、新しい送達トークンが作成されます。 送達トークンは、パブリケーションの送達をモニターしたり、送達が完了するまでクライアント・アプリケーションをブロックしたりするために使用します。

トークンは MqttDeliveryToken オブジェクトです。 これは MqttTopic.publish() メソッドを呼び出すことによって作成されます。 クライアント・セッションが切断され、送達が完了するまで MQTT クライアントによって保存されます。

トークンは通常、送達が完了しているかどうかを調べるために使用します。 戻されるトークンを使用して token.waitForCompletion を呼び出すことにより、送達が完了するまでクライアント・アプリケーションをブロックします。 または、MqttCallBack ハンドラーを提供します。 パブリケーションの送達の一部として予期するすべての確認応答を MQTT クライアントが受信すると、クライアントは MqttCallBack.deliveryComplete を呼び出して送達トークンをパラメーターとして渡します。

送達が完了するまで、戻される送達トークンを使用して token.getMessage を呼び出すことにより、パブリケーションを調べることができます。

完了送達
送達の完了は非同期で、パブリケーションに関連付けられているサービス品質によって異なります。

最高 1 回

QoS=0
送達は、MqttTopic.publish からの戻り時に即時に完了します。
MqttCallback.deliveryComplete が即時に呼び出されます。

最低 1 回

QoS=1
送達は、パブリケーションに対する確認応答をキュー・マネージャーから受信したときに完了します。 確認応答を受信すると、MqttCallback.deliveryComplete が呼び出されます。 通信が遅かったり不安定であったりする場合は、MqttCallback.deliveryComplete が呼び出される前にメッセージが複数回送達される可能性があります。

正確に 1 回

QoS=2
パブリケーションがサブスクライバーにパブリッシュされたという完了メッセージをクライアントが受け取ると、送達が完了します。 パブリケーション・メッセージを受信した後すぐに MqttCallback.deliveryComplete が呼び出されます。 完了メッセージを待ちません。

まれに、MqttCallback.deliveryComplete からクライアント・アプリケーションが正常に MQTT クライアントに戻らないことがあります。 送達が完了したことは、MqttCallback.deliveryComplete が呼び出されたことで分かります。 クライアントが同じセッションを再始動する場合、MqttCallback.deliveryComplete は再び呼び出されません。

未完了送達

送達が完了せずにクライアント・セッションが切断された場合、クライアントを再接続して送達を完了することができます。 メッセージの送達は、MqttConnectionOptions 属性が false に設定されたセッションでメッセージがパブリッシュされた場合のみ完了できます。

同じクライアント ID およびサーバー・アドレスを使ってクライアントを作成してから接続します。 その際、cleanSession MqttConnectionOptions 属性は再び false に設定します。 cleanSession を true に設定すると、保留中の送達トークンが廃棄されてしまいます。

MqttClient.getPendingDeliveryTokens を呼び出すことにより、保留中の送達があるかどうかを調べることができます。 MqttClient.getPendingDeliveryTokens はクライアントに接続する前に呼び出すことができます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

遺言パブリケーション

MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

遺言用のトピックを作成します。 `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost` などのトピックを作成することができます。

`MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)` メソッドを使用して、「遺言」をセットアップします。

`lastWillPayload` メッセージにタイム・スタンプを作成することを検討します。 クライアントおよび接続環境の識別に役立つその他のクライアント情報を組み込みます。 `MqttClient` コンストラクターに `MqttConnectionOptions` オブジェクトを渡します。

`lastWillQos` を 1 または 2 に設定し、WebSphere MQ のメッセージを持続にして、確実に送信されるようにします。 最後に失われた接続情報を保存するには、`lastWillRetained` を `true` に設定します。

接続が予期せず終了した場合に、「遺言」パブリケーションがサブスクライバーに送信されます。 これは、クライアントが `MqttClient.disconnect` メソッドを呼び出さずに接続が終了した場合に送信されます。

接続をモニターするには、「遺言」パブリケーションを他のパブリケーションで補完して、接続およびプログラムされた切断を記録します。

関連概念
MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するか

を選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID
送達トークン
MQTT クライアントでのメッセージ持続性
パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントでのメッセージ持続性

パブリケーション・メッセージは、「最低 1 回」または「正確に 1 回」のサービス品質で送信される場合に、持続的になります。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

MQTT におけるメッセージ持続性には、メッセージが転送される方法と、それが持続メッセージとして IBM MessageSight および IBM WebSphere MQ のキューに入れられるかどうかの 2 つの面があります。

1. MQTT クライアントは、メッセージ持続性とサービス品質とを結合させます。 メッセージのために選択するサービス品質に応じて、メッセージは永続的になります。 メッセージ永続性は、必要なサービス品質を実装するために必要です。

   「最高 1 回」(QoS=0) を指定した場合、クライアントはパブリッシュの直後にメッセージを廃棄します。 メッセージのアップストリーム処理に障害が生じた場合、メッセージは再送信されません。 クライアントがアクティブのままであっても、メッセージは再送信されません。 QoS=0 のメッセージの動作は、IBM WebSphere MQ の高速非持続メッセージと同じです。
   メッセージは、QoS が 1 または 2 に指定されてクライアントによってパブリッシュされる場合、持続的になります。 そのメッセージはローカルに保管されて、「最低 1 回」(QoS=1) または「正確に 1 回」(QoS=2) の送達を保証する必要がなくなったときにのみ、クライアントから廃棄されます。

2. メッセージは、QoS が 1 または 2 にマーク付けされた場合、持続メッセージとして IBM MessageSight および IBM WebSphere MQ のキューに入れられます。 QoS=0 としてマーク付けされた場合は、非持続メッセージとして IBM MessageSight および IBM WebSphere MQ のキューに入れられます。 IBM WebSphere MQ では、メッセージ・チャネルの NPMSPEED 属性が FAST に設定されていない限り、非持続メッセージはキュー・マネージャー間で「正確に 1 回」転送されます。

永続パブリケーションはクライアント・アプリケーションによって受け取られるまでクライアント上に保管されます。 QoS=2 の場合、アプリケーションのコールバックから制御が戻ったときに、パブリケーションはクライアントから廃棄されます。 QoS=1 の場合、障害が生じたときは、アプリケーションがパブリケーションを再び受け取ることがあります。 QoS=0 の場合、コールバックがパブリケーションを受け取る回数は 1 回以内となります。 パブリケーション時に障害が生じている場合やクライアントが切断されている場合には、パブリケーションを受け取らないことがあります。

トピックにサブスクライブするときに、サブスクライバーがメッセージを受け取る際の QoS を、その持続性の能力に合うように低くすることができます。 より大きな QoS で作成されたパブリケーションは、サブスクライバーが要求したものの中で最高の QoS で送信されます。

メッセージの保管

小さな装置にデータ・ストレージを実装する方法は、さまざまに大きく異なります。 MQTT クライアントが管理するストレージに持続メッセージを一時保存するモデルは、遅すぎたりストレージ要求が大きすぎたりすることがあります。 モバイル・デバイスでは、モバイル・オペレーティング・システムによって MQTT メッセージに適したストレージ・サービスが提供される場合があります。

小型の装置の制約に適合するように柔軟性を提供するために、MQTT クライアントには 2 つの持続性インターフェースがあります。 インターフェースは永続メッセージの保管に関連した操作を定義します。 それらのインターフェースは、Java 用の MQTT クライアント の API ドキュメンテーションで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、 MQTT クライアント・プログラミング・リファレンスを参照してください。装置に適合するようにインターフェースを実装できます。 Java SE で実行される MQTT クライアントには、永続メッセージをファイル・システムに保管するインターフェースのデフォルト実装があります。 それは java.io パッケージを使用します。 クライアントには、Java ME のデフォルト実装 MqttDefaultMIDPPersistence もあります。

パーシスタンス (持続性) クラス

MqttClientPersistence
MqttClientPersistence の実装のインスタンスを、MqttClient コンストラクターのパラメーターとして MQTT クライアントに渡します。 MqttClientPersistence パラメーターを MqttClient コンストラクターから省略した場合、MQTT クライアントはクラス MqttDefaultFilePersistence または MqttDefaultMIDPPersistence を使用して持続メッセージを保管します。

MqttPersistable
MqttClientPersistence は、ストレージ・キーを使用して MqttPersistable オブジェクトを取得および配置します。 MqttDefaultFilePersistence も MqttDefaultMIDPPersistence も使用していない場合は、MqttPersistable の実装および MqttClientPersistence の実装を提供する必要があります。

MqttDefaultFilePersistence

MQTT クライアントは、MqttDefaultFilePersistence クラスを提供します。 クライアント・アプリケーションで MqttDefaultFilePersistence をインスタンス化する場合、永続メッセージを保管するディレクトリーを MqttDefaultFilePersistence コンストラクターのパラメーターとして提供できます。

あるいは、MQTT クライアントが MqttDefaultFilePersistence をインスタンス化して、ファイルをデフォルトのディレクトリーに入れることもできます。 ディレクトリーの名前は *client identifier*-tcp *hostname portnumber* です。 "\", "\\", "/", ":" および " " は、ディレクトリー名ストリングから削除されます。

ディレクトリーへのパスは、システム・プロパティー rcp.data の値です。 rcp.data が設定されていない場合、パスはシステム・プロパティー usr.data の値です。

rcp.data は、OSGi または Eclipse リッチ・クライアント・プラットフォーム (RCP) のインストールに関連付けられたプロパティーです。

usr.data は、アプリケーションを開始した Java コマンドが起動されたディレクトリーです。

MqttDefaultMIDPPersistence
MqttDefaultMIDPPersistence にはデフォルトのコンストラクターがあり、パラメーターはありません。 これは javax.microedition.rms.RecordStore パッケージを使用してメッセージを保管します。

関連概念
MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID
送達トークン
遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

パブリケーション

パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MqttMessage には、ペイロードとしてのバイト配列があります。 メッセージはできる限り小さくしてください。 MQTT プロトコルで許可されるメッセージの最大長は 250 MB です。

通常、MQTT クライアント・プログラムは java.lang.String または java.lang.StringBuffer を使用してメッセージ・コンテンツを扱います。 MqttMessage クラスには、ペイロードをストリングに変換するための便利な toString メソッドがあります。 java.lang.String または java.lang.StringBuffer からバイト配列のペイロードを作成するには、getBytes メソッドを使用します。

getBytes メソッドは、ストリングをプラットフォームのデフォルト文字セットに変換します。 デフォルト文字セットは、通常 UTF-8 です。 テキストだけを含む MQTT パブリケーションは、通常 UTF-8 でエンコードされています。 デフォルト文字セットをオーバーライドするには、メソッド getBytes("UTF8") を使用します。

IBM WebSphere MQ で、MQTT パブリケーションは jms-bytes メッセージとして受け取られます。 メッセージには、<mqtt> フォルダーおよび <mqps> フォルダーのある MQRFH2 フォルダーが含まれます。 <mqtt> フォルダーには clientId および qos が含まれますが、この内容は将来変更される可能性があります。

MqttMessage には 3 つの追加的な属性 (サービス品質、保持されるかどうか、および重複であるかどうか) があります。 重複フラグは、サービス品質が「最低 1 回」または「正確に 1 回」のときにだけ設定されます。 このメッセージが既に送信されたが、MQTT クライアントによって素早く認知されない場合、重複属性が true に設定された状態でそのメッセージが再送信されます。

パブリッシュ

MQTT クライアント・アプリケーションにパブリケーションを作成するには、MqttMessage を作成します。 そのペイロード、サービスの品質、および保持されるかどうかを設定して、MqttTopic.publish(MqttMessage message) メソッドを呼び出します。MqttDeliveryToken が返され、パブリケーションの完了は非同期です。

あるいは、 MQTT クライアントがパブリケーションを作成するときに、 MqttTopic.publish(byte [] payload, int qos, boolean retained) メソッドのパラメーターから一時メッセージ・オブジェクトを作成することもできます。

パブリケーションのサービス品質が「最低 1 回」または「正確に 1 回」、つまり QoS=1 または QoS=2 である場合、MQTT クライアントは MqttClientPersistence インターフェースを呼び出します。 送達トークンをアプリケーションに戻す前に、メッセージを保管するために MqttClientPersistence を呼び出します。

MqttDeliveryToken.waitForCompletion メソッドを使用することで、アプリケーションは、メッセージがサーバーに送達されるまでブロックするよう選択できます。 あるいは、アプリケーションはブロックなしで続行することもできます。 ブロックなしでパブリケーションが送達されたかどうかを確認するには、MQTT クライアントに MqttCallback を実装するコールバック・クラスのインスタンスを登録します。 MQTT クライアントは、パブリケーションが送達された直後に MqttCallback.deliveryComplete メソッドを呼び出します。 サービス品質に応じて、QoS=0 であれば送達はほぼ即時に行われ、QoS=2 であれば幾らかの時間がかかります。

送達が完了したかどうかポーリングするには、MqttDeliveryToken.isComplete メソッドを使用します。 MqttDeliveryToken.isComplete の値が false である間は、MqttDeliveryToken.getMessage を呼び出してメッセージの内容を取得できます。 MqttDeliveryToken.isComplete を呼び出した結果が true の場合、メッセージは既に廃棄されているので、MqttDeliveryToken.getMessage を呼び出すと NULL ポインター例外がスローされます。 MqttDeliveryToken.getMessage と MqttDeliveryToken.isComplete とが同期するような機能は組み込まれていません。

保留中の送達トークンをすべて受け取る前にクライアントが切断した場合、クライアントの新しいインスタンスは、接続の前に保留中の送達トークンを照会することができます。 クライアントが接続するまで新しい送達は完了しないので、MqttDeliveryToken.getMessage を安全に呼び出すことができます。 ど

のパブリケーションがまだ送達されていないかを検出するには、MqttDeliveryToken.getMessage メソッドを使用します。 MqttConnectOptions.cleanSession をデフォルト値 true に設定して接続すると、保留中の送達トークンは廃棄されます。

サブスクライブ

キュー・マネージャーまたは IBM MessageSight は、MQTT サブスクライバーに送るパブリケーションを作成します。 キュー・マネージャーは、MQTT クライアントによって作成されたサブスクリプション内のトピック・フィルターが、パブリケーション内のトピック・ストリングと一致するかどうかを確認します。 この一致は完全一致にすることも、または一致にワイルドカードを含めることもできます。 パブリケーションがキュー・マネージャーによってサブスクライバーに転送される前に、キュー・マネージャーは、パブリケーションに関連付けられたトピック属性を調べます。 ワイルドカード文字を含むトピック・ストリングを使用したサブスクライブ操作で説明されている検索手順に従い、管理トピック・オブジェクトによってサブスクライブ権限がユーザーに与えられているかどうかを識別します。

MQTT クライアントは、「最低 1 回」のサービス品質でパブリケーションを受け取ると、MqttCallback.messageArrived メソッドを呼び出してパブリケーションを処理します。 パブリケーションのサービス品質が「正確に 1 回」つまり QoS=2 である場合、MQTT クライアントは、メッセージを受け取ると MqttClientPersistence インターフェースを呼び出してそれを保管します。 その後、MqttCallback.messageArrived を呼び出します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られま

す。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントによって提供されるサービス品質

MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

パブリケーションのサービス品質は、`MqttMessage` の属性です。 これはメソッド `MqttMessage.setQos` によって設定されます。

メソッド `MqttClient.subscribe` は、トピックのクライアントに送信されるパブリケーションに適用されるサービス品質を下げることができます。 サブスクライバーに転送されるパブリケーションのサービス品質は、パブリケーションのサービス品質と異なる場合があります。 2 つのうちの低い方の値がパブリケーションの転送に使用されます。

最高 1 回
`QoS=0`

メッセージは最高 1 回送信されるか、まったく送信されません。 ネットワークでのメッセージの送信は確認応答されません。
メッセージは保管されません。 クライアントが切断されるか、サーバーで障害が発生すると、メッセージが失われる可能性があります。
`QoS=0` が最も速い転送モードです。 これは「応答不要送信」と呼ばれることがあります。
MQTT プロトコルは、`QoS=0` でパブリケーションをクライアントに転送するためにサーバーを必要としません。 サーバーがパブリケーションを受信したときにクライアントが切断される場合、サーバーによってはパブリケーションが廃棄される可能性があります。 テレメトリー (MQXR) サービスは、`QoS=0` で送信されるメッセージを廃棄しません。 そのメッセージは非持続メッセージとして保管され、キュー・マネージャーが停止する場合にのみ廃棄されます。

最低 1 回
`QoS=1`

`QoS=1` はデフォルトの転送モードです。
メッセージは常に、最低 1 回送信されます。 送信側が確認応答を受信しない場合、確認応答が受信されるまで、`DUP` フラグが設定されたメッセージが再送信されます。 結果として、受信側に同じメッセージが複数回送信されて、受信側がそれを複数回処理することになる可能性があります。
メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。
メッセージは、受信側によって処理された後、受信側から削除されます。 受信側がブローカーの場合、メッセージはそのサブスクライバーにパブリッシュされます。 受信側がクライアントの場合、メッセージはサブスクライバー・アプリケーションに送信されます。 メッセージを削除した後、受信側は送信側に確認応答を送信します。
送信側が受信側から確認応答を受信した後、そのメッセージは送信側から削除されます。

正確に 1 回
`QoS=2`

メッセージは常に、正確に 1 回送信されます。
メッセージは処理されるまで、送信側と受信側でローカルに保管しておく必要があります。
`QoS=2` は最も安全ですが、最も遅い転送モードです。 メッセージを送信側から削除する前に、送信側と受信側の間で少なくとも 2 組の伝送を取ります。 最初の伝送の後に受信側のメッセージを処理することができます。

1 組目の伝送で、送信側はメッセージを伝送し、メッセージが保管されたという確認応答を受信側から取得します。 送信側が確認応答を受信しない場合、確認応答が受信されるまで、DUP フラグが設定されたメッセージが再送信されます。
2 組目の伝送で、送信側はメッセージ "PUBREL" の処理を完了できることを受信側に伝えます。 送信側が "PUBREL" メッセージの確認応答を受信しない場合、確認応答が受信されるまで "PUBREL" メッセージが再送信されます。 送信側は、"PUBREL" メッセージに対する確認応答を受信すると、保存していたメッセージを削除します。
受信側は、メッセージを再処理しない場合には、最初または 2 番目のフェーズでメッセージを処理することができます。 受信側がブローカーの場合、サブスクライバーにメッセージをパブリッシュします。 受信側がクライアントの場合、サブスクライバー・アプリケーションにメッセージを送信します。 受信側は、メッセージの処理を完了したという完了メッセージを送信側に送り戻します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

保存パブリケーションおよび MQTT クライアント

保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

`MqttMessage.setRetained` メソッドを使用して、トピックのパブリケーションが保存されるかどうかを指定します。

IBM WebSphere MQ で保存パブリケーションを削除するには、 **CLEAR TOPICSTR** MQSC コマンドを実行します。

NULL ペイロードでパブリケーションを作成した場合、空のパブリケーションがサブスクライバーに転送されます。 その他の MQTT ブローカーは、空のパブリケーションをサブスクライバーに転送しない可能性があります。

保存パブリケーションを持つトピックに非保存パブリケーションをパブリッシュした場合、保存パブリケーションは影響を受けません。 現在のサブスクライバーは新しいパブリケーションを受信します。 新しいサブスクライバーは、まず保存パブリケーションを受信し、次に新しいパブリケーションがあればそれを受信します。

保存パブリケーションを作成または更新する場合、 QoS または 1 または 2 を指定してパブリケーションを送信します。 QoS を 0 の値で送信すると、IBM WebSphere MQ は非永続保存パブリケーションを作成します。 キュー・マネージャーが停止する場合、パブリケーションは保存されません。

保存パブリケーションを使用して最新の測定値を記録します。 保存トピックへの新規サブスクライバーは、即時に最新の測定値を受信します。 サブスクライバーが最後にパブリケーション・トピックにサブスクライブした後に新しい測定が行われていない場合、サブスクライバーが再度サブスクライブすると、そのトピックの最新の保存パブリケーションを再び受信します。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた `MqttMessage` のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライ

アントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

サブスクリプション

トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

`MqttClient.subscribe` メソッドを使用してサブスクリプションを作成します。 その際に、1 つ以上のトピック・フィルターおよびサービス品質パラメーターを渡します。 サービス品質パラメーターは、サブスクライバーがメッセージを受け取るために使用する用意のある最大のサービス品質を設定します。 このクライアントに送るメッセージを、それよりも高いサービス品質で送信することはできません。 サービス品質は、メッセージがパブリッシュされたときの元の値およびサブスクリプションに指定されたレベルよりも低く設定されます。 メッセージを受け取るためのデフォルトのサービス品質は、`QoS=1` つまり「最低 1 回」です。

サブスクリプション要求そのものは、`QoS=1` で送信されます。

MQTT クライアントが `MqttCallback.messageArrived` メソッドを呼び出すときに、パブリケーションはサブスクライバーによって受け取られます。 `messageArrived` メソッドは、メッセージをサブスクライバーにパブリッシュするときに使用されたトピック・ストリングも渡します。

`MqttClient.unsubscribe` メソッドを使用して、サブスクリプション (またはサブスクリプションのセット) を削除できます。

WebSphere MQ コマンドはサブスクリプションを削除できます。 WebSphere MQ エクスプローラーを使用するか、 **runmqsc** または PCF コマンドを使用して、サブスクリプションをリストします。 すべての MQTT クライアントのサブスクリプションの名前が示されます。 *ClientIdentifier*:*Topic name* という形式の名前が付けられます。

デフォルトの `MqttConnectOptions` を使用するか、クライアントに接続する前に `MqttConnectOptions.cleanSession` を `true` に設定すると、クライアントの接続時にそのクライアントの古いサブスクリプションがすべて削除されます。 セッション中にクライアントによって作成される新規サブスクリプションはすべて、切断時に削除されます。

接続の前に `MqttConnectOptions.cleanSession` を `false` に設定すると、クライアントによって作成されるサブスクリプションはすべて、接続する前にクライアントに存在していたすべてのサブスクリプションに追加されます。 クライアントが切断する際、サブスクリプションはすべてアクティブのままとなります。

`cleanSession` 属性がサブスクリプションに与える影響を知る別の方法は、それをモーダル属性と見なすことです。 そのデフォルト・モード `cleanSession=true` では、クライアントはセッションの有効範囲内でのみサブスクリプションを作成し、パブリケーションを受信します。 それに代わる `cleanSession=false` モードでは、サブスクリプションは永続的になります。 クライアントは接続および切断を行うことができ、そのサブスクリプションはアクティブのままとなります。 クライアントは、再

接続時にすべての未送達パブリケーションを受信します。 接続中、クライアントはアクティブになっているサブスクリプションのセットを代わりに変更することができます。

cleanSession モードは接続する前に設定する必要があります。モードはセッション中ずっと続きます。 その設定を変更するには、クライアントを切断し、再接続する必要があります。 モードを cleanSession=false の使用から cleanSession=true の使用に変更すると、クライアントの前回のサブスクリプションすべてと、受信されていないパブリケーションがあればそれらすべてが廃棄されます。

アクティブなサブスクリプションに一致するパブリケーションは、パブリッシュされるとすぐにクライアントに送信されます。 クライアントが切断されている場合は、そのクライアントが同じサーバーに同じクライアント ID を使用して再接続すれば、 そして MqttConnectOptions.cleanSession が false に設定されていれば送信されます。

特定のクライアントのサブスクリプションは、クライアント ID によって識別されます。 クライアントを別のクライアント装置から同じサーバーに再接続して、同じサブスクリプションの処理を続行し、未送達のパブリケーションを受け取ることもできます。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期
MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID

送達トークン

遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性

パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター
トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

MQTT クライアントのトピック・ストリングおよびトピック・フィルター

トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

トピック・ストリングは、パブリケーションをサブスクライバーに送信するために使われます。 メソッド `MqttClient.getTopic(java.lang.String topicString)` を使用して、トピック・ストリングを作成します。

トピック・フィルターは、トピックにサブスクライブし、パブリケーションを受信するために使われます。 トピック・フィルターにはワイルドカードを含めることができます。 ワイルドカードを使用すると、複数のトピックにサブスクライブできます。 サブスクリプション方式を使用してトピック・フィルターを作成します (例: `MqttClient.subscribe(java.lang.String topicFilter)`)。

トピック・ストリング

IBM WebSphere MQ トピック・ストリングの構文は、トピック・ストリングで説明されています。 MQTT トピック・ストリングの構文は、 Java 用の MQTT クライアントの API 資料の `MqttClient` クラスで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、MQTT クライアント・プログラミング・リファレンスを参照してください。

それぞれの種類のトピック・ストリングの構文は、ほとんど同じです。 ただし、小さな違いが 4 つあります。

1. MQTT クライアントによって IBM WebSphere MQ に送信されるトピック・ストリングは、キュー・マネージャー名の規則に従う必要があります。 特に、トピック・ストリングにハイフンを含めることはできません。
2. 最大長が異なります。 IBM WebSphere MQ トピック・ストリングは 10,240 文字に制限されます。 MQTT クライアントは、最大 65535 バイトのトピック・ストリングを作成することができます。
3. MQTT クライアントによって作成されるトピック・ストリングには、ヌル文字を含めることができません。
4. WebSphere Message Broker では、ヌルのトピック・レベル '`...//...`' が無効でした。 IBM WebSphere MQ ではヌルのトピック・レベルがサポートされています。

IBM WebSphere MQ パブリッシュ/サブスクライブとは異なり、`mqttv3` プロトコルには管理トピック・オブジェクトという概念がありません。 トピック・オブジェクトおよびトピック・ストリングからトピック・ストリングを構成することはできません。 ただし、トピック・ストリングは IBM WebSphere MQ の管理トピックにマップされます。 管理トピックに関連付けられているアクセス制御は、パブリケーションがトピックにパブリッシュされるか、廃棄されるかを決定します。 サブスクライバーへの転送時にパブリケーションに適用される属性は、管理トピックの属性の影響を受けます。

トピック・フィルター

IBM WebSphere MQ トピック・フィルターの構文は、 トピック・ベースのワイルドカード・スキームで説明されています。 MQTT クライアントで構成できるトピック・フィルターの構文は、Java 用の MQTT クライアントの API 資料の `MqttClient` クラスで説明されています。 MQTT クライアント・ライブラリーのクライアント API 資料へのリンクについては、 MQTT クライアント・プログラミング・リファレンスを参照してください。

それぞれの種類のトピック・フィルターの構文はほとんど同じです。 唯一の違いは、さまざまな MQTT ブローカーがトピック・フィルターを解釈する方法です。 WebSphere Message Broker V6 では、マルチレベル・ワイルドカードはトピック・フィルターの末尾でのみ使用可能でした。 IBM WebSphere MQ では、マルチレベル・ワイルドカードはトピック・ツリーのあらゆるレベルで使用可能です (例えば `USA/#/Dutchess County`)。

関連概念

MQTT クライアント・アプリケーションでのコールバックと同期

MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離

します。 パブリケーション、送達トークン、および接続消失イベントは、 MqttCallback を実装するコールバック・クラスのメソッドに送達されます。

クリーン・セッション
MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に MqttConnectOptions.cleanSession を設定して、クリーン・セッション・モードを変更します。

クライアント ID
送達トークン
遺言パブリケーション
MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。 「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

MQTT クライアントでのメッセージ持続性
パブリケーション
パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT クライアントは、IBM WebSphere MQ に送るパブリケーションを作成することができ、パブリケーションを受け取るために IBM WebSphere MQ MQ トピックにサブスクライブすることができます。

MQTT クライアントによって提供されるサービス品質
MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

保存パブリケーションおよび MQTT クライアント
保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

サブスクリプション
トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

C クライアント・プログラミングの概念

このトピックでは、MQ Telemetry Transport バージョン 3.1 の C クライアントと Java クライアントの違いについて説明します。 このトピックでは、クライアントの概念と C 参照情報について補足します。

このトピックは、528 ページの『MQTT クライアント・プログラミングの概念』と同様に編成されています。 見出しはそれぞれ『*WebSphere(r) MQ Telemetry Transport* クライアント・プログラミングの概念』内のトピックに対応しています。 各セクションでは、C クライアントと Java クライアントの違いについて説明します。 Java メソッドと C 関数のシグニチャーにおけるわずかな違いについては説明しません。

C クライアントが最も良く使用されるのは、テレメトリー・デバイスと WebSphere MQ Telemetry デーモン (デバイス用) の間に単純なアダプターを実装する場合です。 一般的にデーモンは、非常に単純なテレメトリー装置とテレメトリー (MQXR) サービスの間のネットワーク・コンセントレーターとして使用されます。

WebSphere MQ Telemetry デーモン (デバイス用) は C クライアントでもあります。 それとテレメトリー (MQXR) サービスとの振る舞いの違いについて説明します。 このデーモンは、それに接続しているクライアント用に JAAS または SSL を実装しません。

mqttclient.dll および mqttclient.lib は、MQ Telemetry Transport バージョン 3.1 プロトコルの C 実装環境用のクライアント関数が含まれる、32 ビットの Windows ライブラリーです。 32 ビットの Linux ライブラリーは、libmqttclient.so および libmqttclient.a です。 2 つのヘッダー・ファイル MQTTClient.h および MQTTClientPersistence.h には、クライアント・アプリケーションによって必要とされる関数およびその他の宣言が入っています。 これらのファイルは、WebSphere MQ Telemetry のインストールに含まれています。

MQ Telemetry Transport クライアントを開発して実行するには、これらのファイルをクライアント装置にコピーする必要があります。 WebSphere MQ クライアントとは異なり、別のクライアント・ランタイムをインストールする必要はありません。

WebSphere MQ および WebSphere MQ Telemetry デーモン (デバイス用) への MQ Telemetry Transport クライアントの接続を制御する、WebSphere MQ Telemetry 機能に関連付けられたライセンス条件を検討します。

C クライアントは、MQ Telemetry Transport バージョン 3.1 の参照実装です。 ユーザー独自のクライアントを、さまざまな装置プラットフォームに適したさまざまな言語で実装できます。 詳細については、MQ Telemetry Transport のフォーマットおよびプロトコルを参照してください。

MQTT クライアント ID

534 ページの『クライアント ID』	クライアント ID は、MQTT クライアントを識別する 23 バイトのストリングです。 各 ID は、一度に 1 つだけの接続先クライアントに対して固有のものでなければなりません。 ID に含まれる文字は、キュー・マネージャー名で有効な文字だけであることが必要です。 この制約内で、任意の ID ストリングを使用できます。 クライアント ID を割り振る手順、および選択した ID を使用してクライアントを構成する手段を持つことは大切です。

- 相違はありません。

パブリケーション

540 ページの『パブリケーション』	パブリケーションは、トピック・ストリングに関連付けられた MqttMessage のインスタンスです。 MQTT

- サービスの品質が「応答不要送信」(QoS=0) のパブリケーションについては、コールバック関数は呼び出されません。

送達トークン

535 ページの『送達トークン』	クライアントがトピックにでパブリッシュするときに、新しい送達トークンが作成されます。 送達トークンは、パブリケーションの送達をモニターしたり、送達が完了するまでクライアント・アプリケーションをブロックしたりするために使用します。

- 送達トークンは int です。 これには、MQTTClient_deliveryToken の typedef があります。
- サービスの品質が「応答不要送信」(QoS=0) のパブリケーションについては、コールバック関数は呼び出されません。

保存パブリケーション

545 ページの『保存パブリケーションおよび MQTT クライアント』	保存パブリケーションを持つトピックへのサブスクリプションを作成すると、トピックの最新の保存パブリケーションが即時に転送されます。

- パーシスタンスが構成されている場合にのみ、保存メッセージはデーモン内に保存されます。 保存メッセージおよびサブスクリプションの保存を参照してください。

  WebSphere MQ の場合、サービスの品質は、保存メッセージが永久に保存されるかどうかに影響します。 クライアントがテレメトリー・サービスに接続している場合、キュー・マネージャーがシャットダウンされると、サービスの品質が「応答不要送信」(QoS=0) の保存メッセージは廃棄されます。

サブスクリプション

546 ページの『サブスクリプション』	トピック・フィルターを使用して、パブリケーション・トピックにインタレストを登録するサブスクリプションを作成します。 クライアントは複数のサブスクリプションを作成するか、またはワイルドカードを使用するトピック・フィルターを含む 1 つのサブスクリプションを作成して、複数のトピックにインタレストを登録できます。 フィルターに一致するトピック上のパブリケーションは、クライアントに送られます。 クライアントが切断されている間もサブスクリプションはアクティブのままであることができます。 クライアントが再接続されるとき、パブリケーションがそこに送信されます。

- パーシスタンスが構成されている場合にのみ、永続サブスクリプションはデーモン内に保存されます。 保存メッセージおよびサブスクリプションの保存を参照してください。
- パブリケーションは同期を取って受信できます。 `MQTTClient_receive` 関数を呼び出します。

コールバックと同期

529 ページの『MQTT クライアント・アプリケーションでのコールバックと同期』	MQTT クライアントのプログラミング・モデルでは、スレッドが広く使用されます。 スレッドは、MQTT クライアント・アプリケーションを、サーバーとのメッセージ送受信における遅延からできるだけ切り離します。 パブリケーション、送達トークン、および接続消失イベントは、 `MqttCallback` を実装するコールバック・クラスのメソッドに送達されます。

- C クライアントにおける同期操作はモーダルです。 `MQTTClient_setCallback` を呼び出すと、クライアントは非同期モードになります。
- 同期モードでは、ネットワークを活動状態のまま維持するために、アプリケーション・クライアントが自発的に制御を明け渡すことで、MQTT クライアントが肯定応答を行い、MQTT ping を発行できるようにする必要があります。 制御を明け渡すには、`MQTTClient_receive` または `MQTTClient_yield` を呼び出します。

トピック・ストリングおよびフィルター

548 ページの『MQTT クライアントのトピック・ストリングおよびトピック・フィルター』	トピック・ストリングおよびトピック・フィルターは、パブリッシュおよびサブスクライブに使用されます。 MQTT クライアントのトピック・ストリングとトピック・フィルターの構文は、IBM WebSphere MQ のトピック・ストリングの構文とほぼ同じです。

- WebSphere MQ Telemetry デーモン (デバイス用) は、WebSphere MQ v7 とは異なる方法で、複数レベルのワイルドカード `#` を処理します。 `#` がワイルドカードとして振る舞うためには、フィルター・ストリングの最後の 2 文字が `/#` になっている必要があります。 WebSphere MQ v7 では、複数レベルのワイルドカードの有効な使用方法は、`../#/..` です。 WebSphere MQ Telemetry デーモン (デバイス用) は、WebSphere MQ Broker v6 と同様に、複数レベルのワイルドカードを処理します。

サービスの品質

543 ページの『MQTT クライアントによって提供されるサービス品質』	MQTT クライアントはパブリケーションを WebSphere MQ および MQTT クライアントに送信するために、「最高 1 回」、「最低 1 回」、および「正確に 1 回」という 3 つのサービス品質を提供します。 MQTT クライアントが WebSphere MQ にサブスクリプションの作成要求を送信する際、要求は「最低 1 回」のサービス品質で送信されます。

- 相違はありません。

メッセージの持続性

538 ページの『MQTT クライアントでのメッセージ持続性』	パブリケーション・メッセージは、「最低 1 回」または「正確に 1 回」のサービス品質で送信される場合に、持続的になります。 持続性のための独自の手段をクライアントに実装することも、クライアントで提供されるデフォルトの持続性の手段を使用することもできます。 永続性は、クライアントに送信するパブリケーションとクライアントから受信するパブリケーションの双方向に機能します。

- 言語バインディングに違いがあるため、C クライアントではメッセージの永続性のメカニズムを次のように設定します。 以下の 3 つのオプションのいずれか 1 つを `MQTTClient_create` の 4 つめのパラメーターとして設定して、MQTT C クライアントを呼び出します。

  MQTTCLIENT_PERSISTENCE_DEFAULT
  ファイル・ベースの永続性です。 詳細は、クライアント・プラットフォームによって異なります。

  MQTTCLIENT_PERSISTENCE_NONE
  データはメモリー内にのみ保持され、クライアントが停止すると失われます。 WebSphere MQ Telemetry デーモン (デバイス用) では、このオプションのみサポートしています。

  MQTTCLIENT_PERSISTENCE_USER
  独自の永続性メカニズムを実装する関数を開発することができます。 独自の関数を示すポインターが含まれた構造体 `MQTTClient_persistence` を、`MQTTClient_create` 呼び出しに渡します。 詳しくは、MQTT C クライアントの参照情報をお読みください。

クリーン・セッション

532 ページの『クリーン・セッション』	MQTT クライアントおよび遠隔測定 (MQXR) サービスは、セッション状態の情報を保持します。 状態情報は、「最高 1 回」、「最低 1 回」、および「正確に 1 回」のパブリケーションの受信を確実なものとするために使用されます。 セッション状態には MQTT クライアントによって作成されるサブスクリプションも含まれます。 セッション間の状態の情報を保持して MQTT クライアントを実行するか、保持せずに実行するかを選択することができます。 接続する前に `MqttConnectOptions.cleanSession` を設定して、クリーン・セッション・モードを変更します。

- 相違はありません。

遺言

537 ページの『遺言パブリケーション』	MQTT クライアント接続が予期せずに終了する場合、「遺言」パブリケーションを送信するように WebSphere MQ Telemetry を構成することができます。 パブリケーションの内容とその送信先のトピックを事前定義します。「遺言」は接続プロパティーです。 遺言はクライアントに接続する前に作成しておきます。

- 相違はありません。

プログラム・エラーの処理

ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

キュー・マネージャーは、可能なときはいつでも、MQI 呼び出しが行われるとすぐにどのようなエラーでも戻します。 これらはローカルで判別されたエラー です。

メッセージをリモート・キューに送信するときは、MQI 呼び出しが行われた時にエラーが明らかにならないことがあります。 この場合は、エラーを識別したキュー・マネージャーが、別のメッセージを発信元のプログラムに送ってエラーを報告します。 これらはリモートで判別されたエラー です。

ローカルで判別されたエラー

MQI 呼び出しの失敗、システムの中断、および間違ったデータを含むメッセージといった、ローカルで判別されるエラーについての情報を示します。

キュー・マネージャーが即時に報告できるエラーで、最も一般的な原因として次の 3 つが挙げられます。

- MQI 呼び出しが失敗した (例えば、キューが満ぱいのために)。
- 実行中のアプリケーションがキュー・マネージャーなどのシステム部分に依存している状態で、このようなシステム部分の運用の中断が発生した。
- メッセージに、正しく処理できないデータが含まれている。

非同期書き込み機能を使用している場合、エラーは即時に 報告されません。 MQSTAT 呼び出しを使用して、前の非同期書き込み操作に関する状況情報を取得 してください。

MQI 呼び出しの失敗

キュー・マネージャーは MQI 呼び出しのどのようなコーディング・エラーでもただちに報告することができます。 そのときは、事前定義された戻りコードのセットが使用されます。 戻りコードは、完了コードと理由コードに分けられます。

呼び出しが成功したかどうかを示すために、キュー・マネージャーは呼び出しが完了したときに完了コード を戻します。 完了コードには、成功、部分完了、および呼び出しの失敗を示す 3 種類のコードがあります。 また、キュー・マネージャーは、部分完了または呼び出しの失敗の理由を示す理由コード も戻します。

各呼び出しに対する完了コードおよび理由コードは、戻りコードにその呼び出しの説明と共に記載されています。 修正処置のアイデアを含む詳細については、以下を参照してください。

- 理由コード (その他すべての WebSphere MQ プラットフォームの場合)

各呼び出しから発生する可能性のあるすべての戻りコードを処理できるように、プログラムを設計してください。

システム割り込み

キュー・マネージャーがシステム障害から回復しなければならない場合に、それに接続するアプリケーションが割り込みを検知しないことがあります。 しかし、このような割り込みが起きてもユーザーのデータが失われないようにアプリケーションを設計しなければなりません。

データの一貫性を維持するために使用できる方法は、キュー・マネージャーが稼働しているプラットフォームによって次のように異なります。

UNIX、Linux および Windows システム
これらの環境では、MQPUT および MQGET 呼び出しを通常の方法で行うことができますが、MQCMIT および MQBACK 呼び出しを使用して同期点を宣言する必要があります (324 ページの『作業単位のコミットとバックアウト』を参照してください)。 CICS 環境では、CICS が管理する作業単位内で MQPUT および MQGET 呼び出しを行えるので、MQCMIT および MQBACK コマンドは使用不可になります。

失いたくないデータをすべて送達するには、持続メッセージを使用してください。 キュー・マネージャーが障害から回復する必要がある場合は持続メッセージはキューに再び戻されます。 UNIX、 Linux、および

Windows システム上の WebSphere MQ では、アプリケーション内の MQGET または MQPUT 呼び出しは、すべてのログ・ファイルが満杯になると失敗し、メッセージ MQRC_RESOURCE_PROBLEM が出されます。 AIX、 HP-UX、 Linux、Solaris、および Windows システムのログ・ファイルについて詳しくは、 管理を参照してください。

アプリケーションの実行中に、オペレーターがキュー・マネージャーを停止させる場合は、通常、休止オプションが使用されます。 キュー・マネージャーは静止状態となり、アプリケーションはその状態で作業を続行できますが、できるだけ早くすぐに終了させなければなりません。 小規模で高速のアプリケーションは、多くの場合、静止状態を無視して正常に終了するまで続行できます。 実行時間の長いアプリケーションや、メッセージの到着を待機しているアプリケーションでは、MQOPEN、MQPUT、MQPUT1、および MQGET 呼び出しを使用する場合には、静止状態のときは失敗 オプションを設定することが必要です。 これらのオプションは、キュー・マネージャーが静止しているときは呼び出しが失敗することを意味します。 ただし、アプリケーションには、静止状態を無視する呼び出しを出して正しく終了するだけの時間がある場合もあります。 このようなアプリケーションは、それ自体が行った変更をコミットまたはバックアウトしてから終了することもできます。

キュー・マネージャーが強制停止 (つまり、静止状態のない停止) された場合は、アプリケーションが MQI 呼び出しを 行うと MQRC_CONNECTION_BROKEN という理由コードが出されます。 アプリケーションを終了するか、UNIX、Linux、および Windows システムの場合は代わりに MQDISC 呼び出しを発行します。

誤りデータを含むメッセージ

使用中のアプリケーションで作業単位が使用されている場合に、プログラムがキューから取り出したメッセージを正常に処理できないときは、MQGET 呼び出しがバックアウトされます。

キュー・マネージャーはそれが起こった回数をカウントして、メッセージ記述子の *BackoutCount* フィールドに保持します。 このカウントは影響を受ける各メッセージの記述子に保持されます。 このカウントによりアプリケーションの効率に関する貴重な情報が得られます。 バックアウト・カウントが時間の経過に伴って増加するメッセージは、繰り返し拒否されます。 この現象が生じる理由を分析し、それに応じてそのようなメッセージを処理できるよう、アプリケーションを設計してください。

WebSphere MQ (Windows、UNIX、および Linux システム用) では、バックアウト・カウントは常にキュー・マネージャーの再始動時に保持されます。

問題判別用の報告メッセージの使用

リモート・キュー・マネージャーは、MQI 呼び出しを行ったときに、エラー (メッセージをキューに入れられない場合など) を報告することはできませんが、メッセージをどのように処理したかを知らせる報告メッセージを出力できます。

アプリケーション内では報告メッセージを作成 (MQPUT) することができ、オプションを選択してそれらを受信することもできます (この場合、別のアプリケーションかキュー・マネージャーによって送信されます)。

報告メッセージの作成

報告メッセージによって、アプリケーションは送られたメッセージを処理できないということを別のアプリケーションに通知できます。

ただし、*Report* フィールドを最初に分析して、メッセージを送信したアプリケーションが問題通知の対象となるかどうかを判断する必要があります。 報告メッセージが必要と判断した場合は、さらに次の事項を決定する必要があります。

- 元のメッセージをすべて含めるか、先頭の 100 バイトのデータだけを含めるか、それとも元のメッセージを一切含めないか。
- 元のメッセージをどのように処理するか。 廃棄することも、送達不能キューに入れることもできます。
- *MsgId* フィールドおよび *CorrelId* フィールドの内容は、同様に必要か。

作成される報告メッセージの理由を示すには、*Feedback* フィールドを使用してください。 報告メッセージは、アプリケーションの応答先キューに書き込みます。 詳細については、Feedback を参照してください。

報告メッセージの要求および受信 (MQGET)

メッセージを別のアプリケーションに送信するときは、*Report* フィールドを設定して必要なフィードバックを指示しない限り、問題が通知されることはありません。 使用可能なオプションについては、レポート・フィールドの構造を参照してください。

キュー・マネージャーは、報告メッセージをアプリケーションの応答先キューに必ず書き込みますが、使用する独自のアプリケーションにより同じ処理を実行することをお勧めします。 報告メッセージ機能を使用するときは、応答先キューの名前をメッセージのメッセージ記述子に指定してください。そうしないと、MQPUT 呼び出しは失敗します。

アプリケーションには、応答先キューをモニターし、そこに到着するメッセージを処理する手順が含まれている必要があります。 報告メッセージには、元のメッセージが全部または先頭の 100 バイトだけ入っているか、または元のメッセージはまったく入っていないことを念頭においてください。

キュー・マネージャーは、報告メッセージの *Feedback* フィールドを設定することにより、宛先キューが存在しないなどのエラーの原因を示します。 使用しているプログラムも同様の処理を行います。

報告メッセージの詳細については、11 ページの『レポート・メッセージ』を参照してください。

リモートで判別されたエラー

メッセージをリモート・キューに送信すると、ローカル・キュー・マネージャーが MQI 呼び出しを処理した際にエラーを検出しなかった場合でも、リモート・キュー・マネージャーによるメッセージの処理内容は、別の要因によって左右される可能性があります。

例えば、宛先に指定しているキューが満ぱいであったり、存在していない場合もあります。 メッセージが、宛先キューへの経路上にある他の中間キュー・マネージャーによって処理されなければならない場合は、これらのプログラムはエラーを検出する可能性があります。

メッセージ送達時の問題

MQPUT 呼び出しが失敗した場合、キューにメッセージを再度書き込む、メッセージを送信側に戻す、またはメッセージを送達不能キューに書き込むかのいずれかを行うことができます。

各オプションにはそれぞれの利点がありますが、宛先キューが満ぱいであったことが原因で MQPUT が失敗した場合には、必ずしもメッセージの書き込みを再試行する必要はありません。 この場合は、メッセージを送達不能キューに書き込むことにより、あとで正しい宛先キューにメッセージを送達できます。

メッセージ送達の再試行

MsgRetryCount および *MsgRetryInterval* という属性がそのチャネルに設定されていた場合、またはチャネルが使用する再試行出口プログラム (チャネルの属性である *MsgRetryExitId* フィールドに名前が保管されているプログラム) が存在する場合、リモート・キュー・マネージャーは、メッセージが送達不能キューに書き込まれる前に、キューにメッセージを再度書き込もうとします。

MsgRetryExitId フィールドがブランクの場合、*MsgRetryCount* および *MsgRetryInterval* という属性の値が使用されます。

MsgRetryExitId フィールドがブランクではない場合、この名前の出口プログラムが実行されます。ユーザー独自の出口プログラムの使用について詳しくは、398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

送信側への戻りメッセージ

送信側にメッセージを戻すには、元のメッセージをすべて取り込むように、生成される報告メッセージを要求します。

報告メッセージのオプションの詳細については、11 ページの『レポート・メッセージ』を参照してください。

送達不能 (未配布メッセージ) キューの使用

キュー・マネージャーは、メッセージを送達できないと、メッセージをその送達不能キューに書き込もうとします。 このキューは、キュー・マネージャーのインストール時に定義する必要があります。

ご使用のプログラムも送達不能キューを使用できますが、この使用方法はキュー・マネージャーによる使用方法と同様です。 送達不能キューの名前は、(MQOPEN 呼び出しを使用して) キュー・マネージャーのオブジェクトをオープンし、(MQINQ 呼び出しを使用して) *DeadLetterQName* 属性について問い合わせることによって確認できます。

キュー・マネージャーは、このキューにメッセージを書き込むときに、メッセージにヘッダーを追加します。その形式は、送達不能ヘッダー (MQDLH) 構造体によって記述されます。 MQDLH-送達不能ヘッダー を参照してください。 このヘッダーには、宛先キューの名前およびメッセージが送達不能キューに書き込まれた理由が入っています。 メッセージを目的のキューに書き込むときは、必ずヘッダーを除去し、問題を解決しておいてください。 さらに、キュー・マネージャーは、メッセージ記述子 (MQMD) の *Format* フィールドを変更することにより、メッセージに MQDLH 構造体が格納されていることを示します。

MQDLH 構造体

送達不能キューに書き込んだすべてのメッセージに、MQDLH 構造体を追加することをお勧めします。ただし、特定の WebSphere MQ 製品により提供された送達不能キュー・ハンドラーを使用する場合は、メッセージに MQDLH 構造体を追加する**必要があります**。

メッセージにヘッダーを追加すると、送達不能キューに対してメッセージが長くなりすぎることがあります。そのため、メッセージが、送達不能キューに許容される最大サイズよりも、少なくとも MQ_MSG_HEADER_LENGTH 定数の値だけ短いことを常に確認してください。 キューに書き込み可能なメッセージの最大サイズは、そのキューの *MaxMsgLength* 属性の値により決まります。 送達不能キューの場合、この属性をキュー・マネージャーによる最大許容値に設定してください。 使用中のアプリケーションがメッセージを送達できず、さらにメッセージが長すぎて送達不能キューに書き込むことができない場合は、MQDLH 構造体の説明に記述されている推奨事項に従ってください。

送達不能キューがモニターされ、そのキューに到着するすべてのメッセージが処理されることを確認してください。 送達不能キュー・ハンドラーはバッチ・ユーティリティーとして稼働し、送達不能キュー上の選択メッセージに関するさまざまな処理を実行するために使用できます。 詳細については、556 ページの『送達不能キュー処理』を参照してください。

データ変換が必要な場合に、MQGET 呼び出しの MQGMO_CONVERT オプションを使用すると、キュー・マネージャーはヘッダー情報を変換します。 メッセージを書き込むプロセスが MCA の場合、元のメッセージのすべてのテキストがヘッダーのあとに書き込まれます。

送達不能キューに書き込まれるメッセージがこのキューには長すぎる場合、メッセージが切り捨てられる場合があります。 こうした状況を示す一例としては、送達不能キュー上のメッセージがこのキューの *MaxMsgLength* 属性の値と同じ長さである場合が考えられます。

送達不能キュー処理

ここでは、送達不能キュー処理を使用する際の汎用プログラミング・インターフェース情報を示します。

送達不能キューの処理は、ローカル・システムの要件によって異なりますが、仕様を作成する際には次の点を考慮してください。

- メッセージは、その中に送達不能キューのヘッダーがあるので識別できます。これは、MQMD の形式フィールドの値が MQFMT_DEAD_LETTER_HEADER だからです。
- WebSphere MQ for z/OS で CICS を使用する場合、MCA がこのメッセージを送達不能キューに書き込むと、 *PutApplType* フィールドは MQAT_CICS になり、 *PutApplName* フィールドは MCA のトランザクション名が後に続く CICS システムの *ApplId* になります。
- このメッセージが送達不能キューに送られる理由は、送達不能キュー・ヘッダーの *Reason* フィールドに入ります。
- 送達不能キュー・ヘッダーには、宛先キュー名とキュー・マネージャー名の詳細が収められます。
- 送達不能キュー・ヘッダーには、メッセージが宛先キューに書き込まれる前にメッセージ記述子内に復元しなければならないフィールドが収められています。 次のとおりです。

1. *Encoding*
2. *CodedCharSetId*
3. *Format*

- メッセージ記述子は、元のアプリケーションによる PUT と同じです。 ただし、上記の 3 つのフィールド (Encoding、CodedCharSetId、および Format) を除きます。

送達不能キュー・アプリケーションでは、次の 1 つ以上の操作を実行する必要があります。

- *Reason* フィールドを調べます。 次の理由により、MCA がメッセージを書き込んだ場合があるからです。
  - メッセージがチャネルの最大メッセージ・サイズより長くなった。

    理由は MQRC_MSG_TOO_BIG_FOR_CHANNEL です。
  - メッセージがその宛先キューに書き込まれなかった。

    理由は、MQPUT 操作により戻る可能性があるすべての MQRC_* 理由コードとなります。
  - ユーザー出口で、このアクションが要求された。

    理由コードは、ユーザー出口により提供されるか、デフォルトの MQRC_SUPPRESSED_BY_EXIT となります。
- 可能な場合は、メッセージをその予定の宛先に転送することを試みる。
- メッセージが送達不能となった理由は明確になったが、すぐに修正できないとき、ある程度の間メッセージを保存してから廃棄する。
- 問題が特定されたら、問題の訂正を管理者に指示する。
- 破壊されているメッセージや処理できないメッセージを廃棄する。

送達不能キューから回復させたメッセージを処理するには、次の 2 つの方法があります。

1. メッセージの宛先がローカル・キューの場合は、次のようにします。
   - アプリケーション・データを抽出するために必要なコード変換を実行する。
   - そのデータがローカル関数の場合は、このデータのコード変換を実行する。
   - すべてのメッセージ記述子の詳細を復元して、結果として生成されたメッセージをローカル・キューに書き込む。
2. メッセージの宛先がリモート・キューの場合は、メッセージをそのキューに書き込む。

分散キューイング環境での未配布メッセージの処理方法については、メッセージを送達できない場合の処理を参照してください。

マルチキャスト・プログラミング

この情報は、キュー・マネージャーへの接続や例外報告などの、WebSphere MQ Multicast のプログラミング・タスクについて学習するために使用します。

WebSphere MQ Multicast は、ユーザーに対しては可能な限り透過的でありながら、既存のアプリケーションとは互換性があるように設計されています。 COMMINFO オブジェクトを定義し、TOPIC オブジェクトの **MCAST** および **COMMINFO** パラメーターを設定すると、マルチキャストを使用するために既存の WebSphere MQ アプリケーションを大幅に再作成する必要はなくなります。 ただし、考慮すべきいくつかの制約 (詳細は 557 ページの『マルチキャストと Message Queue Interface』を参照) やセキュリティー問題 (詳細はマルチキャストのセキュリティーを参照) が存在する場合があります。

マルチキャストと Message Queue Interface

ここでは、主な MQI 概念およびそれらと WebSphere MQ マルチキャストとの関連について説明します。

マルチキャスト・サブスクリプションは非永続です。物理キューが使用されず、永続サブスクリプションによって作成されるオフライン・メッセージがどこにも保管されないためです。

アプリケーションがマルチキャスト・トピックをサブスクライブすると、あたかもキューのハンドルであるかのように、取り込み (MQGET) に使用できるオブジェクト・ハンドルが返されます。 これは、管理マル

チキャスト・サブスクリプション (MQSO_MANAGED で作成されたサブスクリプション) のみがサポートされることを意味します。 つまり、サブスクリプションを作成してキューにあるメッセージを「指す」ことはできません。 そのため、サブスクリプション呼び出しで返されたオブジェクト・ハンドルからメッセージを取り込まなければならないことになります。 クライアントでは、メッセージはクライアントによって取り込まれるまでメッセージ・バッファーに保管されます。詳しくは、クライアント構成ファイルの MessageBuffer スタンザを参照してください。 クライアントがパブリッシュ速度に追いつかない場合は、必要に応じてメッセージが廃棄されます (最も古いメッセージを最初に廃棄)。

アプリケーションがマルチキャストを使用するかどうかは通常、管理上の決定です (トピック・オブジェクトの MCAST 属性を設定することによって指定)。 パブリッシュ・アプリケーションがマルチキャストが使用されないようにする必要がある場合は、MQOO_NO_MULTICAST オプションを使用できます。 同様に、サブスクライブ・アプリケーションは、MQSO_NO_MULTICAST オプションを使用してサブスクライブすることによって、マルチキャストが使用されないようにすることができます。

WebSphere MQ マルチキャストは、メッセージ・セレクターの使用をサポートします。 セレクターは、アプリケーションが、選択ストリングが表す SQL92 照会を満たすプロパティーが指定されたメッセージにのみ含まれているインタレストを登録するのに使用されます。 メッセージ・セレクターについて詳しくは、22 ページの『Selectors』を参照してください。

次の表は、主な MQI 概念のすべておよびそれらとマルチキャストとの関連をリストしたものです。

表 *71. MQI* の概念と、マルチキャストとの関係

MQI 概念	**マルチキャストの使用を試みたときのアクション**	**理由コード**
ゼロ長メッセージの書き込み	拒否済み	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
グループ化	拒否済み	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	拒否済み	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
配布リスト	拒否済み	2154 (086A) (RC2154): MQRC_RECS_PRESENT_ERROR
MQINQ	トピック・ハンドルについては拒否。トピックの MQINQ および MQSET はサポートされていません。	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE
	管理対象ハンドルについては受け入れ。 照会できるのは Current Depth のみです。	• 値が Current Depth の場合、適用できる理由コードはありません。 • 値が Current Depth 以外の値である場合、理由コードは 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR です。
MQSET	すべてのハンドルについて拒否	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
トランザクション (XA または非 XA)	拒否済み	2072 (0818) (RC2072): MQRC_SYNCPOINT_NOT_AVAILABLE
メッセージ参照	拒否済み	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE

表 71. *MQI* の概念と、マルチキャストとの関係 *(続き)*		
MQI 概念	**マルチキャストの使用を試みたときのアクション**	**理由コード**
メッセージのロック	拒否済み	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
マークによる参照	拒否済み	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
コンテキストの受け渡し	拒否済み	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	拒否。 マルチキャストのみのトピックに MQPUT1 を試行することは無効です。	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
永続サブスクリプション	トピックに「マルチキャストのみ」のマークが付いている場合は拒否。それ以外の場合は非マルチキャスト・サブスクリプションが行われます。	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	拒否。 トピック・ストリングが 255 文字より大きい場合は、クライアントで拒否されます。	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
非管理サブスクリプション	トピックに「マルチキャストのみ」のマークが付いている場合は拒否。それ以外の場合は非マルチキャスト・サブスクリプションが行われます。	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	拒否済み	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

以下の項目は、この表の MQI 概念の一部をさらに詳しく説明したものであると同時に、表にない MQI 概念の一部の情報でもあります。

メッセージの持続性
非永続マルチキャスト・サブスクライバーの場合、パブリッシャーからの永続メッセージはリカバリー不能な形で配信されます。

メッセージ切り捨て
メッセージ切り捨てがサポートされます。つまり、アプリケーションは以下の処理を行えることになります。

1. MQGET を発行する。

2. MQRC_TRUNCATED_MSG_FAILED を取得する。
3. より大きいバッファーを割り振る。
4. MQGET を再発行してメッセージを取得する。

サブスクリプション有効期限
サブスクリプション期限はサポートされていません。 期限を設定しようとしても無視されます。

マルチキャストの高可用性

この情報は、WebSphere MQ Multicast の継続的な対等通信操作を理解するために使用します。WebSphere MQ は WebSphere MQ キュー・マネージャーに接続しますが、メッセージはそのキュー・マネージャー内を流れません。

マルチキャスト・トピック・オブジェクトの MQOPEN または MQSUB を実行するには、キュー・マネージャーへの接続を行う必要がありますが、メッセージ自体はキュー・マネージャー内を流れません。 したがって、マルチキャスト・トピック・オブジェクトに対する MQOPEN または MQSUB が完了した後は、キュー・マネージャーへの接続が失われている場合でも、マルチキャスト・メッセージの送信を続行することは可能です。 次の 2 つのモードの操作があります。

キュー・マネージャーに通常の接続を行う
キュー・マネージャーへの接続が存在している間は、マルチキャスト通信は可能です。 接続が失敗した場合は、通常の MQI 規則が適用されます。例えば、マルチキャスト・オブジェクト・ハンドルへの MQPUT は 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN を戻します。

キュー・マネージャーにクライアント接続の再接続を行う
再接続サイクル中にも、マルチキャスト通信は可能です。 これはつまり、キュー・マネージャーへの接続が切断された場合でも、マルチキャスト・メッセージの書き込みとコンシュームは影響を受けないということです。 クライアントはキュー・マネージャーへの再接続を試行し、再接続が失敗した場合には、接続ハンドルは破壊され、マルチキャストの呼び出しを含むすべての MQI 呼び出しは失敗します。 詳細については、クライアントの自動再接続を参照してください。

いずれかのアプリケーションが明示的に MQDISC を発行している場合、すべてのマルチキャスト・サブスクリプションとオブジェクト・ハンドルは閉じられます。

マルチキャストによる対等通信操作の続行

クライアント間の対等通信の利点の 1 つは、メッセージがキュー・マネージャー内を流れる必要がないということですしたがって、キュー・マネージャーへの接続が切断された場合でも、メッセージ転送は続行します。 以下の制限は、このモードの継続メッセージ要件に適用されます。

- 継続操作のために、MQCNO_RECONNECT_* オプションの 1 つを使用して接続を行う必要があります。 このプロセスは、通信セッションが切断されても、実際の接続ハンドルは破壊されておらず、代わりに再接続状態になることを意味します。 再接続が失敗する場合には、接続ハンドルは破壊され、それ以降のすべての MQI 呼び出しは実行できなくなります。
- このモードでは、MQPUT、MQGET、MQINQ、および非同期コンシュームのみがサポートされます。 すべての MQOPEN、MQCLOSE、または MQDISC 動詞は、完了するためにキュー・マネージャーへの再接続が必要です。
- キュー・マネージャーへの状況フローは停止します。したがって、キュー・マネージャーについて示される状態は、既に古くなっているかまたは失効している場合があります。 つまり、クライアントがメッセージを送受信するとしても、キュー・マネージャーの状況が分からないという意味です。 詳細については、マルチキャスト・アプリケーションのモニターを参照してください。

マルチキャスト・メッセージング用の MQI でのデータ変換

この情報は、WebSphere MQ Multicast メッセージングでデータ変換が機能する方法を理解するために使用します。

WebSphere MQ Multicast は、共有のコネクションレス・プロトコルです。このため各クライアントは、データ変換に対して固有の要求をすることができません。 同じマルチキャスト・ストリームをサブスクライ

ブするすべてのクライアントは、同じバイナリー・データを受け取ります。したがって、WebSphere MQ データ変換が必要な場合、変換は各クライアントでローカルに実行されます。

データはクライアント上で WebSphere MQ Multicast トラフィック用に変換されます。 **MQGMO_CONVERT** オプションが指定されている場合、データ変換は要求どおりに実行されます。 ユーザー定義のフォーマットでは、クライアントにデータ変換出口がインストールされている必要があります。クライアント・パッケージとサーバー・パッケージに現在入っているライブラリーの詳細については、416 ページの『データ変換出口の作成』を参照してください。

データ変換の管理については、Multicast メッセージングに関するデータ変換を使用可能にするを参照してください。

データ変換の詳細については、データ変換を参照してください。

データ変換出口および ClientExitPath の詳細については、クライアント構成ファイルの ClientExitPath スタンザを参照してください。

マルチキャスト例外報告

この情報は、WebSphere MQ Multicast イベント・ハンドラーと、WebSphere MQ Multicast 例外の報告について学習するために使用します。

WebSphere MQ Multicast は、イベント・ハンドラーを呼び出して、標準 WebSphere MQ イベント・ハンドラー・メカニズムを使用して報告されるマルチキャスト・イベントを報告することで、問題判別を支援します。

個々のマルチキャスト・イベントの結果として、複数の WebSphere MQ イベントが呼び出される可能性があります。これは同じマルチキャスト送信側または受信側を使用する複数の MQHCONN 接続ハンドルが存在することがあるためです。 ただし、それぞれのマルチキャスト例外では、WebSphere MQ 接続当たり 1 つのみのイベント・ハンドラーが呼び出されます。

WebSphere MQ MQCBDO_EVENT_CALL 定数により、アプリケーションは WebSphere MQ イベントのみを受け取るためにコールバックを登録することができます。さらに、MQCBDO_MC_EVENT_CALL により、アプリケーションはマルチキャスト・イベントのみを受け取るためにコールバックを登録することができます。 両方の定数を使用する場合、両方のタイプのイベントを受け取ります。

マルチキャスト・イベントの要求

WebSphere MQ Multicast イベントは、 cbd.Options フィールド内の MQCBDO_MC_EVENT_CALL 定数を使用します。 以下の例は、マルチキャスト・イベントの要求方法を示しています。

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_MC_EVENT_CALL;
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

MQCBDO_MC_EVENT_CALL オプションを cbd.Options フィールドに指定する場合、接続レベル・イベントではなく WebSphere MQ Multicast イベントのみがイベント・ハンドラーに送信されます。 どちらのタイプのイベントもイベント・ハンドラーに送信されるように要求するには、アプリケーションにより cbd.Options フィールドに MQCBDO_EVENT_CALL 定数を、MQCBDO_MC_EVENT_CALL 定数と共に指定する必要があります。以下に例を示します。

```
cbd.CallbackType     = MQCBT_EVENT_HANDLER;
cbd.Options          = MQCBDO_EVENT_CALL | MQCBDO_MC_EVENT_CALL
cbd.CallbackFunction = EventHandler;
MQCB(Hcon,MQOP_REGISTER,&cbd,MQHO_UNUSABLE_HOBJ,NULL,NULL,&CompCode,&Reason);
```

この定数のどちらも使用しない場合、接続レベル・イベントのみがイベント・ハンドラーに送信されます。

Options フィールドの値の詳細については、Options (MQLONG) を参照してください。

マルチキャスト・イベント・フォーマット

WebSphere MQ Multicast 例外には、コールバック関数の **Buffer** パラメーターで返されるいくつかのサポート情報が含まれます。 **Buffer** ポインターはポインターの配列を指し、MQCBC.DataLength フィールドは配列のサイズをバイト単位で指定します。 配列の最初の要素は、必ずイベントの短いテキスト記述を指します。 イベントのタイプに応じてさらにパラメーターを指定することもできます。 以下の表は、例外をリストしています。

表 72. マルチキャスト・イベント・コードの説明

イベント・コード	説明	追加データ
MQMCEV_PACKET_LOSS	リカバリー不能のパケット・ロス	失われたパケットの数
MQMCEV_HEARTBEAT_TIMEOUT	ハートビート制御パケットの長期間の欠落	なし
MQMCEV_VERSION_CONFLICT	より新しいプロトコル・バージョン・パケットの受信	なし
MQMCEV_RELIABILITY	送信側および受信側の異なる信頼性モード	なし
MQMCEV_CLOSED_TRANS	トピック送信は 1 ソースで閉じられる	なし
MQMCEV_STREAM_ERROR	ストリームでエラーが検出された	なし
MQMCEV_NEW_SOURCE	新しいソースがトピックの送信を開始する	ソース構造体
MQMCEV_RECEIVE_QUEUE_TRIMMED	時間切れまたはスペース切れが原因で PacketQ から削除されたパケットの数	トリムされたパケットの数
MQMCEV_PACKET_LOSS_NACK_EXPIRE	NACK 満了に起因するリカバリー不能パケット・ロス	失われたパケットの数
MQMCEV_ACK_RETRIES_EXCEEDED	**max_ack_retries** を超過した後に履歴から削除されたパケットの数	削除されたパケットの数
MQMCEV_STREAM_SUSPEND_NACK	このトピックで受け入れられたストリームで NACK が中断状態である	中断ストリーム ID
		ストリームが中断されるミリ秒単位の時間
MQMCEV_STREAM_RESUME_NACK	ストリームでの中断後に NACK は再開される	ストリーム ID
MQMCEV_STREAM_EXPELLED	このトピックで受け入れられたストリームが追放要求が原因で拒否された	ストリーム ID
MQMCEV_FIRST_MESSAGE	ソースからの最初のメッセージ	メッセージ番号
MQMCEV_LATE_JOIN_FAILURE	遅延結合セッションを開始できなかった	なし
MQMCEV_MESSAGE_LOSS	リカバリー不能のメッセージ損失	失われたメッセージの数

表 72. マルチキャスト・イベント・コードの説明 (続き)

イベント・コード	説明	追加データ
MQMCEV_SEND_PACKET_FAILURE	マルチキャスト送信側がマルチキャスト・パケットを送信できなかった	なし
MQMCEV_REPAIR_DELAY	マルチキャスト受信側が未解決の NAK の修復パケットを受け取らなかった	なし
MQMCEV_MEMORY_ALERT_ON	受信側の受信バッファーが満杯である	バッファー・プール使用率
MQMCEV_MEMORY_ALERT_OFF	受信側の受信バッファーが通常に戻っている	バッファー・プール使用率
MQMCEV_NACK_ALERT_ON	受信側の修復パケット要求率が最高水準点に達した	秒当たりのパケット数での、現在の修復要求率
MQMCEV_NACK_ALERT_OFF	受信側の修復パケット要求率が通常に戻っている	秒当たりのパケット数での、現在の修復要求率
MQMCEV_REPAIR_ALERT_ON	送信側の修復パケット送信率が最高水準点に達した	なし
MQMCEV_REPAIR_ALERT_OFF	送信側の修復パケット送信率が通常に戻っている	なし
MQMCEV_SHM_DEST_UNUSABLE	送信側トピック宛先により使用される共有メモリー領域が使用不可であることが検出された	なし
MQMCEV_SHM_PORT_UNUSABLE	受信側インスタンスにより使用される共有メモリー・ポートが使用不可であることが検出された	なし
MQMCEV_CCT_GETTIME_FAILED	調整クラスター時間からの取得時間が失敗した	なし
MQMCEV_DEST_INTERFACE_FAILURE	送信側トピック宛先により使用されるネットワーク・インターフェースに障害が発生し、バックアップのネットワーク・インターフェースが使用不可である	
MQMCEV_DEST_INTERFACE_FAILOVER	送信側トピック宛先により使用されるネットワーク・インターフェースに障害が発生し、別のインターフェースへのフェイルオーバーが正常に実行された	
MQMCEV_PORT_INTERFACE-FAILURE	受信側 rmmPort により使用されるネットワーク・インターフェースに障害が発生し、バックアップのネットワーク・インターフェースが使用不可である (または同じように障害が発生している)	RMM 構成

表 72. マルチキャスト・イベント・コードの説明 (続き)		
イベント・コード	**説明**	**追加データ**
MQMCEV_PORT_INTERFACE_FAILOVER	受信側 rmmPort により使用されるネットワーク・インターフェースに障害が発生し、別のインターフェースへのフェイルオーバーが正常に実行された	RMM 構成

.NET の使用

.NET プログラミング・フレームワークで作成されたプログラムは、WebSphere MQ classes for .NET によって、WebSphere MQ に WebSphere MQ MQI クライアントとして接続するか、または WebSphere MQ サーバーに直接接続することができます。

Microsoft の .NET Framework を使用するアプリケーションで WebSphere MQ の機能を 利用したい場合、WebSphere MQ classes for .NET を使用する必要があります。

オブジェクト指向型の WebSphere MQ .NET インターフェースは、MQI 動詞を使用するのではなく、オブジェクトのメソッドを使用するという点で、MQI インターフェースとは異なります。

手続き型の WebSphere MQ アプリケーション・プログラミング・インターフェースは、以下のリストのような動詞を中心に作成されます。

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

これらの動詞はすべて、操作する WebSphere MQ オブジェクトのハンドルをパラメーターとして 使用します。 .NET はオブジェクト指向なので、.NET プログラミング・インターフェースもオブジェクト指向になっています。 ユーザーのプログラムは、一連の WebSphere MQ オブジェクトで構成されています。これらのオブジェクトは、メソッドを呼び出すことによって操作します。 .NET でサポートされる任意の言語でプログラムを作成できます。

手続き型インターフェースを使用する場合は、呼び出し MQDISC(*Hconn*, CompCode, Reason) を使用して、キュー・マネージャーから切断します。 *Hconn* はキュー・マネージャーのハンドルです。

.NET インターフェースでは、キュー・マネージャーはクラス MQQueueManager のオブジェクトによって表されます。 このクラスの Disconnect() メソッドの呼び出しにより、 キュー・マネージャーから切断します。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.Disconnect();
```

WebSphere MQ classes for .NET は、.NET アプリケーションによる WebSphere MQ との対話を可能にする一連のクラスです。 アプリケーションで使用する、キュー・マネージャー、キュー、チャネル、メッセージといった WebSphere MQ の様々なコンポーネントを表します。 これらのクラスの詳細は、WebSphere MQ .NET のクラスとインターフェースを参照してください。

作成したアプリケーションをコンパイルするためには、.NET Framework がインストールされている必要があります。 WebSphere MQ classes for .NET および .NET Framework のインストール手順については、566 ページの『WebSphere MQ classes for .NET のインストール』を参照してください。

関連概念

技術概要

565 ページの『接続オプション』

WebSphere MQ classes for .NET のキュー・マネージャーへの接続には、3 つのモードが存在します。 どのタイプの接続が最も必要に適しているかを考慮してください。

577 ページの『WebSphere MQ classes for .NET の使用』
このトピックのコレクションでは、サンプル・プログラムを実行するようにシステムを構成して WebSphere MQ classes for .NET のインストールを検証する方法、および独自のプログラムを実行する方法について説明します。

579 ページの『WebSphere MQ .NET の問題の解決』
プログラムが正常に完了しない場合は、サンプル・アプリケーションの 1 つを実行して、 診断メッセージに表示される指示に従います。

580 ページの『WebSphere MQ .NET プログラムの作成およびデプロイ』
WebSphere MQ classes for .NET を使用して WebSphere MQ キューにアクセスするには、 WebSphere MQ キューにメッセージを書き込んだり、そこからメッセージを取得したりする呼び出しを含む、.NET によってサポートされる任意の言語でプログラムを作成します。

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

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

7 ページの『アプリケーションの開発』
IBM WebSphere MQ には、ビジネス・プロセスをサポートするために必要なメッセージを送受信するアプリケーションを開発するためのいくつかの手段が用意されています。 キュー・マネージャーや関連リソースを管理するためのアプリケーションを開発することもできます。

WebSphere MQ classes for .NET の概要

.NET プログラミング・フレームワークで作成されたプログラムは、WebSphere MQ classes for .NET によって、WebSphere MQ に WebSphere MQ MQI クライアントとして接続するか、または WebSphere MQ サーバーに直接接続することができます。

接続オプション

WebSphere MQ classes for .NET のキュー・マネージャーへの接続には、3 つのモードが存在します。 どのタイプの接続が最も必要に適しているかを考慮してください。

クライアント・バインディング接続

WebSphere MQ classes for .NET を WebSphere MQ MQI クライアントとして使用する には、WebSphere MQ サーバー・マシンまたは別のマシンに WebSphere MQ MQI クライアントとともにインストールします。クライアント・バインディング接続では、XA または非 XA トランザクションを使用できます。

サーバー・バインディング接続

WebSphere MQ classes for .NET は、サーバー・バインディング・モードで使用されている場合、ネットワーク経由で通信を行うのではなく、キュー・マネージャー API を使用します。 これにより、ネットワーク接続を使用した場合と比べて WebSphere MQ アプリケーションの パフォーマンスが向上します。

バインディング接続を使用するには、WebSphere MQ サーバーに WebSphere MQ classes for .NET をインストールする必要があります。

管理対象クライアント接続

このモードでの接続では、ローカル・マシンまたはリモート・マシンで実行されている WebSphere MQ サーバーに WebSphere MQ クライアントとして接続します。

このモードで接続する WebSphere MQ classes for .NET は、.NET 管理対象コードに残り、 ネイティブ・サービスに対する呼び出しを行いません。 管理対象コードについて詳しくは、Microsoft の資料を参照してください。

管理対象クライアントの使用には、いくつかの制限があります。 これらについて詳しくは、 580 ページの『管理対象クライアント接続』を参照してください。

WebSphere MQ classes for .NET のインストール

WebSphere MQ classes for .NET とそのサンプルは、WebSphere MQ と一緒にインストールされます。 前提条件として、Microsoft .NET Framework が必要です。

最新バージョンの WebSphere MQ classes for .NET は、デフォルトでは、WebSphere MQ 標準インストールの一部として、*Java and .NET Messaging and Web Services* フィーチャーにインストールされます。 インストール手順については、 Windows への IBM WebSphere MQ サーバーのインストール または Windows システムへの IBM WebSphere MQ クライアントのインストール を参照してください。

複数インストール環境で、WebSphere MQ classes for .NET をサポート・パックとして事前にインストールしてある場合は、そのサポート・パックをまずアンインストールしなければ WebSphere MQ をインストールできません。 WebSphere MQ と一緒にインストールされる WebSphere MQ classes for .NET フィーチャーに、サポート・パックと同じ機能が含まれています。

ソース・ファイルを含む、サンプル・アプリケーションも用意されています。577 ページの『サンプル・アプリケーション』を参照してください。

32 ビットまたは 64 ビット・プラットフォームで WebSphere MQ classes for .NET を実行するには、Microsoft .NET Framework V2.0 以降をインストールしておく必要があります。

注 : WebSphere MQ V7.0.1 をインストールする前に Microsoft .NET Framework v2.0 以降がインストールされていない場合、WebSphere MQ 製品のインストールはエラーなしで続行されますが、WebSphere MQ classes for .NET は使用できません。 WebSphere MQ 7.0.1 のインストール後に .NET Framework をインストールする場合は、 *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` スクリプトを実行して WebSphere .NET アセンブリーを登録する必要があります。ここで、 *WMQInstallDir* は、 WebSphere MQ 7.0.1 がインストールされているディレクトリーです。 このスクリプトにより、必要なアセンブリーがグローバル・アセンブリー・キャッシュ (GAC) にインストールされます。 実施されたアクションを記録する一連の `amqi*.log` ファイルが、 %TEMP% ディレクトリーに作成されます。

.NET 3 を使用した Microsoft WCF 用の WebSphere MQ カスタム・チャネルの使用については、 599 ページの『Microsoft Windows Communication Foundation (WCF) 用の IBM WebSphere MQ カスタム・チャネル』 を参照してください。

.NET での分散トランザクション

分散トランザクションやグローバル・トランザクションを使用すると、 クライアント・アプリケーションは複数のネットワーク・システムにある複数の異なるデータ・ソースを 1 つのトランザクションに 含めることができます。

分散トランザクションでは、トランザクション・マネージャーが複数のリソース・マネージャーの間で トランザクションの調整と管理を行います。

トランザクションは、単一フェーズ・コミット・プロセスまたは 2 フェーズ・コミット・プロセスにすることができます。 単一フェーズ・コミットは、1 つだけのリソース・マネージャーがトランザクションに参加するプロセスであり、 2 フェーズ・コミット・プロセスでは、複数のリソース・マネージャーがトランザクションに参加します。 2 フェーズ・コミット・プロセスでは、 トランザクション・マネージャーは準備呼び出しを送信して、 すべてのリソース・マネージャーがコミットする準備ができているかどうかをチェックします。 すべてのリソース・マネージャーから確認応答が受信されたら、 コミット呼び出しが発行されます。 そうでない場合、トランザクション全体のロールバックが実行されます。 詳しくは、「トランザクション管理およびサポート」を参照してください。 リソース・マネージャーは、自身がトランザクションに参加することをトランザクション・マネージャーに知らせる必要があります。 自身の参加をトランザクション・マネージャーに知らせた後、 そのリソース・マネージャーは、そのトランザクションがコミットまたはロールバックされるときに トランザクション・マネージャーからコールバックを受信します。

WebSphere MQ .NET のクラスは、既に非管理対象モードおよびサーバー・バインディング・モードの接続での分散トランザクションをサポートしています。 これらのモードでは、WebSphere MQ .NET クラスはすべての呼び出しを、.NET の代わりにトランザクション処理を管理する C 拡張トランザクション・クライアントに委任します。

現在、WebSphere MQ.NET のクラスは、管理対象モードでの分散トランザクションをサポートしています。管理対象モードでは、WebSphere MQ .NET のクラスは System.Transactions 名前空間を使用して分散トランザクションをサポートします。 System.Transactions インフラストラクチャーでは、 WebSphere MQ を含むすべてのリソース・マネージャーで開始されたトランザクションがサポートされ、トランザクション・プログラミングを単純かつ効率的に行えます。 WebSphere MQ .NET アプリケーションは、 .NET の暗黙的トランザクション・プログラミング・モデルまたは明示的トランザクション・プログラミング・モデルを使用して、メッセージの書き込みと読み取りを行うことができます。 暗黙的トランザクションでは、トランザクションをコミット、ロールバック (明示的トランザクションの場合)、または完了するタイミングを決定するアプリケーション・プログラム によってトランザクション境界が作成されます。 明示的トランザクションでは、 トランザクションのコミット、ロールバック、および完了を行うかどうかを明示的に指定する必要があります。

WebSphere MQ.NET は、Microsoft Distributed Transaction Coordinator (MS DTC) をトランザクション・マネージャーとして使用して、 複数のリソース・マネージャーの間でトランザクションの調整と管理を行います。 WebSphere MQ はリソース・マネージャーとして使用されます。

WebSphere MQ.NET は、X/Open Distributed Transaction Processing (DTP) モデルに従っています。 X/Open Distributed Transaction Processing モデルは、 ベンダー・コンソーシアムの Open Group によって提案された分散トランザクション処理モデルです。 このモデルはトランザクション処理とデータベースの領域を扱うほとんどの商用ベンダーの間で標準となっています。 ほとんどの商用のトランザクション管理製品は、X/DTP モデルをサポートしています。

トランザクションのモード

- 568 ページの『管理対象モードの分散トランザクション』
- 非管理対象モード用の分散トランザクション

さまざまなシナリオでのトランザクションの調整

- 1 つの接続が複数のトランザクションに参加している場合がありますが、どの時点においてもアクティブなトランザクションは 1 つだけです。
- トランザクション中は、MQQueueManager.Disconnect 呼び出しが優先されます。 この呼び出しの場合、トランザクションはロールバックするよう要求されます。
- トランザクション中は、MQQueue.Close または MQTopic.Close 呼び出しが優先されます。 この呼び出しの場合、トランザクションはロールバックするよう要求されます。
- トランザクション境界は、 トランザクションをコミット、ロールバック (明示的トランザクションの場合)、 または完了する (暗黙的トランザクションの場合) タイミングを決定するアプリケーション・プログラムによって作成されます。
- トランザクション中に、キュー呼び出しまたはトピック呼び出しに対して Put または Get 呼び出しを発行する前に、予期しないエラーのためにクライアント・アプリケーションが中断した場合、このトランザクションはロールバックされ、MQException がスローされます。
- キュー呼び出しまたはトピック呼び出しに対して Put または Get 呼び出し中に MQCC_FAILED 理由コードが返された場合は、理由コードとともに MQException がスローされ、トランザクションはロールバックされます。 準備呼び出しがトランザクション・マネージャーによってすでに発行済みの場合、WebSphere MQ .NET は、トランザクションを強制的にロールバックすることによって準備要求を返します。 次に、トランザクション・マネージャー DTC が発端になり、現在の周辺トランザクションに関わっているすべてのリソース・マネージャーとの現行作業がロールバックされます。
- 複数のリソース・マネージャーが関係するトランザクション中に、何らかの環境上の理由で Put または Get 呼び出しがいつまでもハングする場合、トランザクション・マネージャーは規定された期間待機します。 その期間が経過した後、すべてのリソース・マネージャーが現在の周辺トランザクションに入っているすべての作業がロールバックされます。 この無期限の待機が準備フェーズで発生する場合、トラン

ザクション・マネージャーはタイムアウトするか、リソースに対して未確定呼び出しを発行します。 この場合、トランザクションはロールバックされます。

- トランザクションを使用しているアプリケーションは、SYNC_POINT の下でメッセージを Put または Get する必要があります。 SYNC_POINT の下にないトランザクション・コンテキストでメッセージの Put 呼び出しまたは Get 呼び出しが発行された場合、その呼び出しは MQRC_UNIT_OF_WORK_NOT_STARTED 理由コードで失敗します。

Microsoft .NET System.Transactions 名前空間を使用した管理対象クライアント・トランザクション・サポートと非管理対象クライアント・トランザクション・サポートの間の動作上の違い

ネストされたトランザクションでは、TransactionScope が別の TransactionScope の中にあります。

- WebSphere MQ .NET が完全に管理するクライアントは、ネストされた TransactionScope をサポートします。
- WebSphere MQ .NET が管理しないクライアントは、ネストされた TransactionScope をサポートしません。

System.Transactions からの従属トランザクション

- WebSphere MQ .NET が完全に管理するクライアントは、System.Transactions が提供する従属トランザクション機能をサポートします。
- WebSphere MQ .NET が管理しないクライアントは、System.Transactions が提供する従属トランザクション機能をサポートしません。

製品サンプル

新しい製品サンプル `SimpleXAPut`、および `SimpleXAGet` は、 `WebSphere MQ\tools\dotnet\samples\cs\base` にあります。 これらのサンプルは C# アプリケーションです。 これは、SystemTransactions 名前空間を使用する分散トランザクションでの MQPUT および MQGET の使用を実演するものです。 これらのサンプルについて詳しくは、571 ページの『TransactionScope 内での単純なメッセージの書き込みおよび読み取りの作成』を参照してください。

管理対象モードの分散トランザクション

WebSphere MQ .NET クラスは、管理対象モードでの分散トランザクションのサポートに System.Transactions 名前空間を使用します。 管理対象モードでは、MS DTC は、トランザクションに参加しているすべてのサーバーにわたる分散トランザクションを調整および管理します。

WebSphere MQ .NET クラスは、System.Transactions.Transaction クラスに基づく明示的プログラミング・モデルと、トランザクションがインフラストラクチャーによって自動的に管理される、System.Transactions.TransactionScope クラスを使用する暗黙的プログラミング・モデルを提供します。

暗黙的トランザクション

次のコードの一部で、WebSphere MQ .NET アプリケーションが .NET の暗黙的トランザクション・プログラミングを使用してメッセージを書き込む方法を説明します。

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

暗黙的トランザクションのコード・フローの説明

コードは、*TransactionScope* を作成し、メッセージを有効範囲内に書き込みます。 その後、*Complete* を呼び出して、トランザクション・コーディネーターにトランザクションの完了を通知します。 トランザクション・コーディネーターは、*prepare* および *commit* を発行して、トランザクションを完了します。 発行が検出されると、*rollback* が呼び出されます。

明示的トランザクション

次のコードで、WebSphere MQ .NET アプリケーションが .NET の明示的トランザクション・プログラミング・モデルを使用してメッセージを書き込む方法を説明します。

```
MQQueueManager qMgr = new MQQueuemanager ("MQQM);
MQQueue Q = QMGR.AccessQueue("Q", MQC.MQOO_OUTPUT+MQC.MQOO_INPUT_SHARED);
MQPutMessageOptions pmo = new MQPutMessageOptions();
pmo.Options = MQC.MQPMO_SYNCPOINT;
MQMessage putMsg1 = new MQMessage();
Using(CommittableTransaction tx = new CommittableTransaction()){
Transaction.Current = tx;
    try
    {
    Q.Put(MSG,pmo);
    tx.commit();
    }
    catch(Exception)
    {tx.rollback();}
    }

Q.close();
qMgr.Disconnect();
}
```

明示的トランザクションのコード・フローの説明

コードの一部では *CommitableTransaction* クラスを使用してトランザクションを作成します。 これは、その有効範囲内にメッセージを書き込んでから、*commit* を明示的に呼び出してトランザクションを完了します。 問題が発生した場合には、*rollback* が呼び出されます。

非管理対象モードの分散トランザクション

WebSphere MQ.NET クラスは、暗黙的または明示的なトランザクション・プログラミング・モデルを使用して、拡張トランザクション・クライアントおよび COM+/MTS をトランザクション・コーディネーターとして使用する非管理対象接続 (クライアント) をサポートします。 非管理対象モードでは、WebSphere MQ .NET クラスは、そのすべての呼び出しを、.NET に代わってトランザクション処理を管理する C 拡張トランザクション・クライアントに委任します。

トランザクション処理は外部のトランザクション・マネージャーによって制御され、そのトランザクション・マネージャーの API の制御下でグローバルな作業単位が調整されます。 MQBEGIN、MQCMIT、および MQBACK の各 verb は使用できません。 WebSphere MQ .NET クラスは、その非管理対象トランスポート・モード (C クライアント) によってこのサポートを公開します。 XA 準拠トランザクション・マネージャーの構成を参照してください。

MTS は、CICS、Tuxedo、およびその他のプラットフォームで使用できるのと同じ機能を Windows NT 上で提供するためのトランザクション処理 (TP) システムとして進化したものです。 MTS がインストールされると、Microsoft 分散トランザクション・コーディネーター (MSDTC) と呼ばれる個別のサービスが Windows NT に追加されます。 MSDTC は、複数の別個のデータ・ストアまたはリソースにわたるトランザクションを調整します。 これが機能するには、各データ・ストアに専有のリソース・マネージャーが実装されている必要があります。

DTC XA 呼び出しを WebSphere MQ(X/Open) 呼び出しにマップするためのインターフェース (専有のリソース・マネージャー・インターフェース) を実装することにより、WebSphere MQ は MSDTC と互換性を持つようになります。 WebSphere MQ は、リソース・マネージャーの役割を果たします。

COM+ などのコンポーネントが WebSphere MQ へのアクセスを要求すると、COM は通常、適切な MTS コンテキスト・オブジェクトでトランザクションが必要かどうかを確認します。 トランザクションが必要な場合、COM は DTC に通知し、この操作に不可欠な WebSphere MQ トランザクションを自動的に開始します。 次に COM は、MQMTS ソフトウェアを使用してデータを操作し、必要に応じてメッセージの書き込みおよび読み取りを行います。 COM から取得されたオブジェクト・インスタンスは、データに対するすべてのアクションが完了した後で、SetComplete メソッドまたは SetAbort メソッドを呼び出します。 アプリケーションが SetComplete を呼び出すと、アプリケーションがトランザクションを完了したことが DTC に通知され、DTC は 2 フェーズ・コミット・プロセスに進むことができます。 次に DTC が MQMTS を呼び出し、今度は MQMTS が WebSphere MQ を呼び出して、トランザクションをコミットまたはロールバックします。

非管理対象クライアントを使用した WebSphere MQ .NET アプリケーションの作成

COM+ のコンテキスト内で実行するには、.NET クラスが System.EnterpriseServices.ServicedComponent を継承していなければなりません。 サービスを受けるコンポーネントを使用するアセンブリーを作成するための規則と推奨事項は、以下のとおりです。

注 : 以下のステップは、System.EnterpriseServices モードを使用する場合だけ該当します。

- COM+ で開始されるクラスとメソッドは、どちらも public でなければなりません (internal クラスでも、protected メソッドでも、static メソッドでもありません)。
- クラスとメソッドの属性: TransactionOption 属性は、クラスのトランザクション・レベル、すなわちトランザクションが使用不可か、サポートされているか、必須かどうかを指示します。 ExecuteUOW() メソッドの AutoComplete 属性は、COM+ に対して、スローされる未処理例外がない場合にトランザクションをコミットするよう指示します。
- アセンブリーの厳密な命名: アセンブリーには厳密な名前を付け、グローバル・アセンブリー・キャッシュ (GAC) に登録する必要があります。 アセンブリーは COM+ に明示的に登録されるか、GAC に登録された後に遅延登録によって登録されます。
- COM+ へのアセンブリーの登録: COM クライアントに公開されるアセンブリーを準備します。 次に、アセンブリー登録ツール regasm.exe を使用して、タイプ・ライブラリーを作成します。

```
        regasm UnmanagedToManagedXa.dll
```

- このアセンブリーを gacutil /i UnmanagedToManagedXa.dll を使用して GAC に登録します。
- .NET サービス・インストーラー・ツール regsvcs.exe を使用して、アセンブリーを COM+ に登録します。regasm.exe によって作成されたタイプ・ライブラリーを、次のようにして確認します。

```
Regsvcs /appname:UnmanagedToManagedXa  /tlb:UnmanagedToManagedXa.tlb
UnmanagedToManagedXa.dll
```

- アセンブリーが GAC にデプロイされ、遅延登録により後で COM+ に登録されます。 .NET Framework は、このコードが初めて実行された後に登録を行います。

以下のセクションでは、System.EnterpriseServices モデルと、COM+ での System.Transactions を使用したコード・フローの例について説明します。

System.EnterpriseServices モデルを使用したコード・フローの例

```
using System;
using IBM.WMQ;
using IBM.WMQ.Nmqi;
using System.Transactions;
using System.EnterpriseServices;

namespace UnmanagedToManagedXa
{

[ComVisible(true)]
[System.EnterpriseServices.Transaction(System.EnterpriseServices.TransactionOption.Required)
]
    public class MyXa : System.EnterpriseServices.ServicedComponent
    {

        public MQQueueManager QMGR = null;
        public MQQueueManager QMGR1 = null;
        public MQQueue QUEUE = null;
        public MQQueue QUEUE1 = null;
        public MQPutMessageOptions pmo = null;
        public MQMessage MSG = null;

        public MyXa()
        {
        }

        [System.EnterpriseServices.AutoComplete()]
        public void ExecuteUOW()
        {
            QMGR = new MQQueueManager("usemq");

            QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                     MQC.MQOO_INPUT_SHARED +
```

```
                                        MQC.MQOO_OUTPUT +
                                        MQC.MQOO_BROWSE);

            pmo = new MQPutMessageOptions();
            pmo.Options = MQC.MQPMO_SYNCPOINT;
            MSG = new MQMessage();
            QUEUE.Put(MSG, pmo);
            QMGR.Disconnect();
        }
    }
}


public void RunNow()
{
   MyXa xa = new MyXa();
   xa.ExecuteUOW();
}
```

COM+ との対話のための System.Transactions を使用したコード・フローの例

```
[STAThread]
public void ExecuteUOW()
{
Hashtable t1 = new Hashtable();
t1.Add(MQC.CHANNEL_PROPERTY, "SYSTEM.DEF.SVRCONN");
t1.Add(MQC.HOST_NAME_PROPERTY, "localhost");
t1.Add(MQC.PORT_PROPERTY, 1414);
t1.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_CLIENT);
TransactionOptions opts = new TransactionOptions();

using(TransactionScope scope = new  TransactionScope(TransactionScopeOption.RequiresNew,
                                opts,     EnterpriseServicesInteropOption.Full)
  {

        QMGR = new MQQueueManager("usemq", t1);
        QUEUE = QMGR.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                        MQC.MQOO_INPUT_SHARED +
                                        MQC.MQOO_OUTPUT +
                                        MQC.MQOO_BROWSE);

         pmo = new MQPutMessageOptions();
         pmo.Options = MQC.MQPMO_SYNCPOINT;
         MSG = new MQMessage();
         QUEUE.Put(MSG, pmo);
         scope.Complete();
  }
 QMGR.Disconnect();
}
```

TransactionScope 内での単純なメッセージの書き込みおよび読み取りの作成

製品のサンプル C# アプリケーションを WebSphere MQ 内で使用できます。 これらの単純なアプリケーションは、TransactionScope 内でのメッセージの書き込みおよび読み取りを実演するものです。 このタスクの終わりには、メッセージをキューまたはトピックに書き込んだり、キューまたはトピックから読み取ったりできるようになります。

始める前に

MSDTC サービスが実行されていて XA トランザクションに対して有効になっている必要があります。

このタスクについて

この例は、単純なアプリケーション SimpleXAPut および SimpleXAGet です。 プログラム SimpleXAPut および SimpleXAGet は、WebSphere MQ 内で使用できる C# アプリケーションです。 SimpleXAPut は、SystemTransactions 名前空間を使用する分散トランザクションでの MQPUT の使用を実演するものです。 SimpleXAGet は、SystemTransactions 名前空間を使用する分散トランザクションでの MQGET の使用を実演するものです。

SimpleXAPut は WebSphere MQ\tools\dotnet\samples\cs\base にあります。

手順

アプリケーションは、tools\dotnet\samples\cs\base\bin のコマンド行パラメーターを使用して実行できます。

```
SimpleXAPut.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

```
SimpleXAGet.exe -d destinationURI [-h host -p port -l channel -tx transaction -tm mode -n
numberOfMsgs]
```

パラメーターは、以下のとおりです。

-destinationURI

これはキューまたはトピックになります。 キューの場合は queue://queueName のように指定し、トピックの場合は topic://topicName のように指定します。

-host

これは localhost などのホスト名か IP アドレスになります。

-port

キュー・マネージャーが実行されているポートです。

-channel

使用する接続チャネルです。 デフォルトは SYSTEM.DEF.SVRCONN です。

-transaction

トランザクションの結果 (例えば commit または rollback など) です。

-mode

トランスポート・モード (例えば managed または unmanaged など) です。

-numberOfMsgs

メッセージの数です。 デフォルトは、1 です。

例

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

トランザクションの回復

このセクションでは、管理対象モードを使用している WebSphere MQ .NET XA におけるトランザクションの回復のプロセスについて説明します。

概要

分散トランザクション処理では、トランザクションを正常に完了することができます。 しかし、多くの理由から、トランザクションが失敗する可能性のあるシナリオが存在する可能性があります。 これらの理由には、システム障害、ハードウェア障害、ネットワーク・エラー、間違いがあるデータまたは無効データ、アプリケーション・エラー、自然災害、あるいは人災などがあります。 トランザクション障害を阻止することはできません。 分散トランザクション・システムは、これらの障害を処理できなければなりません。 エラーが発生したときに、それらのエラーを検出し、訂正できなければなりません。 このプロセスは、トランザクションの回復と呼ばれています。

分散トランザクション処理の重要な側面は、未完了または未確定のトランザクションを回復することです。 ある特定のトランザクションの作業単位部分は、それが回復されるまでロックされたままになるので、回復を実行することが重要になります。 Microsoft .NET の System.Transactions クラス・ライブラリーは、未

完了/未確定のトランザクションを回復するためのオプションを提供しています。 このリカバリー・サポートでは、リソース・マネージャーがトランザクション・ログを保守し、必要な場合には回復を実行することが期待されています。

回復モデル

Microsoft .NET のトランザクション回復モデルであるトランザクション・マネージャー (System.Transactions または Microsoft Distributed Transaction コーディネーター (MS DTC)、あるいはその両方) は、トランザクションの回復を開始、調整、および管理します。 OLE Tx プロトコル (Microsoft の XA プロトコル) を基礎とするリソース・マネージャーは、それらに代わって回復を実施、調整、および制御するよう DTC を構成するためのオプションを提供しています。 これを行うには、リソース・マネージャーがネイティブ・インターフェースを使用して XA_Switch を MS DTC に登録する必要があります。

XA_Switch は、リソース・マネージャーにおける xa_start、xa_end、および xa_recover などの XA 関数のエントリー・ポイントを分散トランザクション・コーディネーターに提供しています。

Microsoft 分散トランザクション・コーディネーター (DTC) を使用した回復:

Microsoft 分散トランザクション・コーディネーターは、2 種類の回復プロセスを提供しています。

コールド・リカバリー

コールド・リカバリーは、XA リソース・マネージャーとの接続が開いている間にトランザクション・マネージャー・プロセスが失敗すると実行されます。 トランザクション・マネージャーは、再始動すると、トランザクション・マネージャーのログを読み取り、XA リソース・マネージャーとの接続を再確立し、回復を開始します。

ホット・リカバリー

ホット・リカバリーは、XA リソース・マネージャーまたはネットワークが失敗したためにトランザクション・マネージャーと XA リソース・マネージャーとの間の接続が失敗しているのに、トランザクション・マネージャーが起動したままになっている場合に実行されます。 トランザクション・マネージャーは、失敗後に、定期的に XA リソース・マネージャーとの再接続を試みます。 接続が再確立すると、トランザクション・マネージャーは XA の回復を開始します。

System.Transactions 名前空間は、トランザクション・マネージャーとしての MS DTC に基づく分散トランザクションの管理された実装を提供します。 この実装は、MS DTC のネイティブ・インターフェースの機能に類似した機能を提供しますが、完全に管理された環境内にあります。 唯一の違いは、トランザクションの回復に関するものです。 System.Transactions は、リソース・マネージャーそれ自体が回復を実施し、トランザクション・マネージャー (MS DTC) と調整を行うことを期待しています。 リソース・マネージャーは、特定の未完了トランザクションの回復を要求しなければならず、トランザクション・マネージャーはそれを受け入れて、その特定のトランザクションの実際の結果に基づいて調整を行います。

WebSphere MQ .NET のトランザクション・リカバリー・プロセス

このセクションでは、WebSphere MQ .NET クラスを使用して分散トランザクションをリカバリーする方法について説明します。

概要

未完了トランザクションを回復するには、回復情報が必要です。 トランザクション回復情報は、リソース・マネージャーがストレージにログとして書き込まなければなりません。 WebSphere MQ .NET のクラスは、同様のパスをたどります。 トランザクション回復情報は、SYSTEM.DOTNET.XARECOVERY.QUEUE と呼ばれるシステム・キューにログとして書き込まれます。

WebSphere MQ .NET でのトランザクションの回復は、2 段階からなるプロセスです。

1. トランザクション回復情報のロギング。
   - トランザクションごとに、準備フェーズで、回復情報を含む持続メッセージが SYSTEM.DOTNET.XARECOVERY.QUEUE に追加されます。
   - このメッセージは、コミット呼び出しが成功すると削除されます。

2. モニター・アプリケーション WmqDotnetXAMonitor を使用したトランザクションの回復。

   - WmqDotnetXAMonitor は、SYSTEM.DOTNET.XARECOVERY.QUEUE 内のメッセージを処理し、未完了トランザクションを回復する .NET によって管理されるアプリケーションです。

   宛先キューにメッセージを書き込むことができない場合、MCA は、元のメッセージが入った例外レポートを生成し、伝送キューに書き込んで、元のメッセージで指定されている応答先キューに送信されるようにします。 (応答先キューが MCA と同じキュー・マネージャー上にある場合は、メッセージは伝送キューには送られず、その応答先キューに直接書き込まれます。)

SYSTEM.DOTNET.XARECOVERY.QUEUE

これは、未完了トランザクションのトランザクション回復情報を保持しているシステム・キューです。 このキューは、キュー・マネージャーの作成時に作成されます。

注 : SYSTEM.DOTNET.XARECOVERY.QUEUE キューは削除しないでください。

WMQDotnetXAMonitor アプリケーション

WebSphere MQ .NET XA Monitor アプリケーションは、特定のキュー・マネージャーをモニターし、未完了トランザクションがあればそれを回復します。 以下のトランザクションは未完了トランザクションとみなされ、回復されます。

未完了トランザクション

- トランザクションは作成されたけれども、タイムアウト期間中にコミットが完了しなかった場合。
- トランザクションは作成されたけれども、WebSphere MQ キュー・マネージャーが停止してしまった場合。
- トランザクションは作成されたけれども、トランザクション・マネージャーが停止してしまった場合。

モニター・アプリケーションは、WebSphere MQ .NET クライアント・アプリケーションが実行されているのと同じシステムから実行しなければなりません。 同じキュー・マネージャーに接続している複数のシステムでアプリケーションが実行されている場合、モニター・アプリケーションはすべてのシステムから実行しなければなりません。 各クライアント・マシンではアプリケーションを回復するためにモニター・アプリケーションが実行されていますが、各モニターは、トランザクションをもう一度参加させて完了できるよう現在のモニターのローカル MS DTC が調整していたトランザクションに対応するメッセージを特定できなければなりません。

WebSphere MQ .NET のトランザクション・リカバリーのユース・ケース

以下に、さまざまな使用法シナリオを示します。

- **1 つの DTC と 1 つのキュー・マネージャー・インスタンスを使用する WebSphere MQ アプリケーション:** このシナリオでは、キュー・マネージャーに接続してトランザクションの下で作業単位 (UoW) を実行しているときに、そのトランザクションが失敗して未完了になった場合は、モニター・アプリケーションがそのトランザクションを回復して、それを完了します。

  このシナリオでは、モニター・アプリケーションの 1 つのインスタンスが実行されます。 そのトランザクションには、1 つのキュー・マネージャーが関連付けられているからです。

- **1 つの DTC と 1 つのキュー・マネージャー・インスタンスを使用する複数の WebSphere MQ アプリケーション:** このシナリオでは、1 つの DTC の下で複数の WMQ アプリケーションが実行しており、すべてアプリケーションが同じキュー・マネージャーに接続して、トランザクションの下で UoW を実行しています。

  トランザクションが失敗し、未完了になると、モニター・アプリケーションはそれらのトランザクションを回復して、すべてのアプリケーションに関係するトランザクションを完了します。

  このシナリオでは、1 つのモニター・アプリケーションが実行されます。 トランザクションでは 1 つのキュー・マネージャーが使用されているからです。

- **複数の WebSphere MQ アプリケーション、複数の DTC、異なる複数のキュー・マネージャー・インスタンス:** このシナリオでは、異なる DTC の下に複数の WMQ アプリケーションが存在しており (すなわち、

各アプリケーションはそれぞれ異なるマシンで実行されています)、異なるキュー・マネージャーに接続しています。

障害が発生してトランザクションが未完了になると、モニター・アプリケーションがメッセージ内の TransactionManagerWhereabouts を検査して、DTC アドレスを判別します。 TransactionManagerWhereabouts の値が、モニターが実行されている DTC アドレスに一致すると、モニター・アプリケーションは回復を完了し、一致しなければ、DTC に対応するメッセージが見つかるまで検索を続けます。

このシナリオでは、実行されているモニター・アプリケーションのインスタンスは、クライアント (ユーザーまたはコンピューター) ごとに 1 つしかありません。 トランザクションでは、各クライアントそれ自体のキュー・マネージャーが使用されているからです。

- **複数の WebSphere MQ アプリケーション、複数の DTC、複数の同一キュー・マネージャー・インスタンス:** このシナリオでは、異なる DTC 下に複数の WMQ アプリケーションがあり (すなわち、各アプリケーションは、異なるマシン上で実行されています)、すべてのアプリケーションは同じキュー・マネージャーに接続しています。

  障害が発生してトランザクションが未完了になると、モニター・アプリケーションがメッセージ内の TransactionManagerWhereabouts を検査して、DTC アドレスと値が、モニターが実行されている DTC に一致するかどうかをチェックします。 両方の値が一致すると、モニター・アプリケーションは回復を完了し、一致しなければその DTC に対応するメッセージが見つかるまで検索を続けます。

  このシナリオでは、実行されているモニター・アプリケーションのインスタンスは、クライアント (ユーザーまたはコンピューター) ごとに 1 つしかありません。 トランザクションでは、各クライアントそれ自体のキュー・マネージャーの関連付けが使用されているからです。
- **複数の WebSphere MQ アプリケーション、1 つの DTC、異なる複数のキュー・マネージャー・インスタンス:** このシナリオでは、1 つの DTC の下に複数の WMQ アプリケーションがあり (すなわち、1 台のコンピューター上で複数の WMQ アプリケーションが実行されています)、それらは異なるキュー・マネージャーに接続しています。

  トランザクションが失敗して未完了になると、モニター・アプリケーションがそのトランザクションを回復します。

  このシナリオでは、実行されているモニター・アプリケーションのインスタンスの数と、それらのインスタンスに接続しているキュー・マネージャーの数は同じになります。 トランザクションでは、各アプリケーションそれ自体のキュー・マネージャーが使用されており、各トランザクションを回復する必要があるからです。

  注 : バックグラウンドでモニター・アプリケーションが実行されていない場合は、そのアプリケーションを始動できます。

WMQDotnetXAMonitor アプリケーションの使用

XA モニター・アプリケーションは手動で実行する必要があります。 このアプリケーションは、いつでも開始できます。 SYSTEM.DOTNET.XARECOVERY.QUEUE にあるメッセージを表示する場合は、このアプリケーションを始動できます。 あるいは、このアプリケーションをバックグラウンドで実行させたまま、WebSphere MQ .NET のクラスを使用して作成されたアプリケーションで任意のトランザクション作業を行うことができます。

モニター・アプリケーションを開始するためのコマンド

```
WmqDotnetXAMonitor.exe -m <QueueManagerName> -n <ConnectionName> -c <Channel> -i
```

どこ

- **n -** ホスト (ポート) フォーマットの接続名。 接続名には、複数の接続名を含めることができます。 複数の接続名は、コンマ区切りのリスト (例えば、「localhost (1414), localhost (1415), localhost (1416)」) で指定しなければなりません。 モニター・アプリケーションは、このコンマ区切りのリストで指定されている接続名ごとに回復を実行します。
- **c -** チャネル名。
- **m -** キュー・マネージャー名。 オプション
- **i -** ヒューリスティック・ブランチの完了。 オプション

モニター・アプリケーションは、以下のアクションを実行します。

1. 100 秒間隔で SYSTEM.DOTNET.XARECOVERY.QUEUE のキューの深さを確認します。
2. キューの深さがゼロよりも大きい場合、XA モニターは、メッセージがないかそのキューをブラウズし、メッセージが未完了トランザクションの基準を満たしているかどうかを確認します。
3. 未完了トランザクションの基準を満たしているメッセージがあった場合、モニター・アプリケーションはそのメッセージを取得して、トランザクション回復情報を取り出します。
4. 次にモニター・アプリケーションは、回復情報がローカルの MS DTC に関係しているかどうかを判別します。 関係している場合は、そのトランザクションの回復を進めます。 満たしていない場合は、次のメッセージのブラウズに戻ります。
5. 次にモニター・アプリケーションは、キュー・マネージャーを呼び出して、未完了トランザクションを回復します。

WmqDotNETXAMonitor アプリケーション構成ファイルの設定
アプリケーションをモニターするために、アプリケーション構成ファイルを使用して入力を提供することもできます。 アプリケーション構成ファイルのサンプルは、WebSphere MQ .NET に付属しています。 このファイルは、必要に応じて変更できます。

入力値を考慮する際には、アプリケーション構成ファイルが最優先されます。 コマンド・ラインとアプリケーション構成ファイルの両方で入力値が提供された場合は、アプリケーション構成ファイルの値が考慮されます。

アプリケーション構成ファイルのサンプル:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor アプリケーション・ログ
モニター・アプリケーションは、モニターの進行状況とトランザクションの回復状況をログに記録するためのログ・ファイルをアプリケーション・ディレクトリーに作成します。 ロギングは接続名と、回復が実行されている現在のキュー・マネージャーを表示するためのチャネル詳細で始まります。

回復が開始すると、トランザクション回復メッセージの MessageId、未完了トランザクションの TransactionId、およびトランザクションの実際の結果が、トランザクション・マネージャーの調整に従ってログに書き込まれます。

ログ・ファイルのサンプル:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Rollback
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Rollback
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

WebSphere MQ classes for .NET の使用

このトピックのコレクションでは、サンプル・プログラムを実行するようにシステムを構成して WebSphere MQ classes for .NET のインストールを検証する方法、および独自のプログラムを実行する方法について説明します。

キュー・マネージャーが *TCP/IP* クライアント接続を受け入れるように構成する

キュー・マネージャーがクライアントからの着信接続要求を受け入れるように構成するには、次の手順に従います。

1. サーバー接続チャネルを定義します。

   a. キュー・マネージャーを始動します。

   b. NET.CHANNEL[3]:

   ```
   DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
   DESCR('Sample channel for WebSphere MQ classes for .NET')
   ```

2. リスナーを開始します。

   ```
   runmqlsr -t tcp [-m qmnqme] [-p portnum]
   ```

   注 : 大括弧は、オプション・パラメーターを表します。 *qmname* は、デフォルトのキュー・マネージャーの場合は不要です。 ポート番号 *portnum* は、デフォルトの (1414) が使用される場合は不要です。

サンプル・アプリケーション

ユーザー独自の .NET アプリケーションを実行するには、サンプル・アプリケーションの代わりにユーザーのアプリケーションの名前を使用して、検証プログラムの指示に従います。

以下の 5 つのサンプル・アプリケーションが用意されています。

- メッセージ書き込みアプリケーション
- メッセージ読み取りアプリケーション
- 「Hello World」アプリケーション
- パブリッシュ/サブスクライブ・アプリケーション
- メッセージ・プロパティーを使用するアプリケーション

これらのサンプル・アプリケーションはすべて C# 言語で提供されていますが、一部は C++ および Visual Basic でも提供されています。 アプリケーションは、.NET によりサポートされる任意の言語で作成できます。

「メッセージ書き込み」プログラム SPUT (nmqsput.cs、mmqsput.cpp、vmqsput.vb)
このプログラムは、メッセージを指定のキューに書き込む方法を示します。 プログラムには 3 つのパラメーターがあります。

- キューの名前 (必須) (例: SYSTEM.DEFAULT.LOCAL.QUEUE)
- キュー・マネージャーの名前 (オプション)
- チャネルの定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。 チャネルが定義されている場合、そのフォーマットは MQSERVER 環境変数と同じです。

「メッセージ読み取り」プログラム SGET (nmqsget.cs、mmqsget.cpp、vmqsget.vb)
このプログラムは、メッセージを指定のキューから読み取る方法を示します。 プログラムには 3 つのパラメーターがあります。

[3] このサンプルでは、セキュリティーへの影響は考慮していません。 実動システムの場合は、SSL やセキュリティー出口の使用を考慮してください。 詳細については、セキュリティーを参照してください。

- キューの名前 (必須) (例: SYSTEM.DEFAULT.LOCAL.QUEUE)
- キュー・マネージャーの名前 (オプション)
- チャネルの定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。 チャネルが定義されている場合、そのフォーマットは MQSERVER 環境変数と同じです。

「Hello World」プログラム (nmqwrld.cs、mmqwrld.cpp、vmqwrld.vb)
このプログラムは、メッセージを書き込んだり読み取ったりする方法を示します。 プログラムには 3 つのパラメーターがあります。

- キューの名前 (オプション) (例: SYSTEM.DEFAULT.LOCAL.QUEUE、SYSTEM.DEFAULT.MODEL.QUEUE など)
- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション) (例: SYSTEM.DEF.SVRCONN/TCP/hostname(1414))

キュー名が指定されなかった場合、キュー名はデフォルトで SYSTEM.DEFAULT.LOCAL.QUEUE になります。 キュー・マネージャー名が指定されなかった場合、キュー・マネージャーとしてデフォルトのローカル・キュー・マネージャーが 設定されます。

「パブリッシュ/サブスクライブ」プログラム (MQPubSubSample.cs)
このプログラムは、WebSphere MQ のパブリッシュ/サブスクライブを使用する方法を示します。 このプログラムは C# でのみ提供されています。 プログラムには 2 つのパラメーターがあります。

- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション)

「メッセージ・プロパティー」プログラム (MQMessagePropertiesSample.cs)
このプログラムは、メッセージ・プロパティーを使用する方法を示します。 このプログラムは C# でのみ提供されています。 プログラムには 2 つのパラメーターがあります。

- キュー・マネージャーの名前 (オプション)
- チャネル定義 (オプション)

インストールを検証する場合は、これらのアプリケーションをコンパイルして実行します。

サンプル・アプリケーションは、作成に使用された言語に応じて、以下の場所にインストールされます。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされている上位ディレクトリーを表します。

C# (C)

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqswrld.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsput.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\nmqsget.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQPubSubSample.cs

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\MQMessagePropertiesSample.cs

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqswrld.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsput.cpp

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\mmqsget.cpp

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\vmqsget.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqswrld.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsput.vb

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\xmqsget.vb

サンプル・アプリケーションを作成するためのバッチ・ファイルが、言語ごとに用意されています。

C# (C)

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

bldcssamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib:MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

Managed C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

bldmcpsamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

これらのアプリケーションを Microsoft Visual Studio 2003/.NET SDKv1.1 でコンパイルする場合は、コンパイル・コマンド

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

を次のコマンドに置き換えてください:

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

bldvbsamp.bat ファイルには、1 行に 1 つのサンプルが 記述されています。 このサンプル・プログラムの作成に必要なことはこれだけです。

```
vbc /r:System.dll /r:MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

WebSphere MQ .NET の問題の解決

プログラムが正常に完了しない場合は、サンプル・アプリケーションの 1 つを実行して、 診断メッセージに表示される指示に従います。

サンプル・アプリケーションについては、577 ページの『WebSphere MQ classes for .NET の使用』を参照してください。

問題が解決せず、IBM サービスに連絡する必要があるときは、トレース機能をオンにするようお願いする場合があります。

サンプル・アプリケーションのトレース

トレース機能の使用については、599 ページの『WebSphere MQ .NET プログラムのトレース』を参照してください。

エラー・メッセージ

以下のような一般的なエラー・メッセージが表示されることがあります。

タイプ 'System.IO.FileNotFoundException' の未処理例外が、不明のモジュールで発生しました

このエラーが amqmdnet.dll または amqmdxcs.dll について発生した場合は、 その両方が「グローバル・アセンブリー・キャッシュ」に登録されるようにするか、 または amqmdnet.dll アセンブリーおよび amqmdxcs.dll アセンブリーを指す構成ファイルを作成します。 mscorcfg.msc を使用すれば、アセンブリー・キャッシュの内容を調べたり変更したりできます。 mscorcfg.msc は .NET Framework に付属しています。

WebSphere MQ のインストール時に .NET Framework が使用可能でなかった場合は、クラスがグローバル・アセンブリー・キャッシュに登録されていない可能性があります。 次のコマンドを使用すれば、登録プロセスを手動で再実行できます。

```
amqidnet -c MQ_INSTALLATION_PATH\bin\amqidotn.txt -l logfile.txt
```

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

このインストールに関する情報は、 指定のログ・ファイル (この例では、`logfile.txt`) に書き込まれます。

WebSphere MQ .NET プログラムの作成およびデプロイ

WebSphere MQ classes for .NET を使用して WebSphere MQ キューにアクセスするには、 WebSphere MQ キューにメッセージを書き込んだり、そこからメッセージを取得したりする呼び出しを含む、.NET によってサポートされる任意の言語でプログラムを作成します。

WebSphere MQ の資料には、C#、C++、および Visual Basic の言語に関する情報のみが含まれます。

この一連のトピックでは、WebSphere MQ システムと対話するアプリケーションの作成に役立つ情報を紹介します。 個別クラスの詳細については、WebSphere MQ .NET のクラスとインターフェースを参照してください。

接続の違い

WebSphere MQ .NET のプログラミング方法は、使用する接続モードによって多少異なります。

管理対象クライアント接続

WebSphere MQ classes for .NET が管理対象クライアントとして使用されている場合は、標準の WebSphere MQ MQI クライアントとは異なる点が数多くあります。

管理対象クライアントからは、以下の機能を使用できません。

- チャネル圧縮
- SSL サポート
- チャネル出口チェーニング

管理対象 クライアントでこれらの機能を使用しようとすると、MQException が返されます。 接続のクライアント側でエラーが検出されると、理由コード MQRC_ENVIRONMENT_ERROR が使用されます。 サーバー側でエラーが検出されると、サーバーから戻された理由コードが使用されます。

非管理対象クライアント用に作成されたチャネル出口は、機能しません。 管理対象クライアント専用の新しい出口を作成する必要があります。 ご使用のクライアント・チャネル定義テーブル (CCDT) に無効なチャネル出口が指定されていないか確認してください。

管理対象チャネル出口の名前の長さは最大 999 文字です。 ただし、CCDT を使用してチャネル出口名を指定する場合は、128 文字までに制限されます。

通信は、TCP/IP を使用した場合にのみサポートされます。

endmqm コマンドを使用してキュー・マネージャーを停止する場合、.NET 管理対象クライアントへのサーバー接続チャネルは、その他のクライアントへのサーバー接続チャネルに比べて、閉じるまでに時間がかかることがあります。

管理対象 WebSphere MQ の問題診断を使用するために *NMQ_MQ_LIB* を managed に設定している場合、**strmqtrc** コマンドのパラメーター -i、-p、-s、-b、および -c はいずれもサポートされません。

XA トランザクションを使用する管理対象 .NET アプリケーションは、z/OS キュー・マネージャーでは機能しません。 z/OS キュー・マネージャーに接続しようとする管理対象 .Net クライアントは、MQOPEN 呼び出し時にエラー MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) で失敗します。 ただし、XA トランザクションを使用する管理対象 .NET アプリケーションは、分散キュー・マネージャーと連動します。

使用する接続タイプの定義

接続タイプは、接続名、チャネル名、カスタマイズ値 NMQ_MQ_LIB、およびプロパティー MQC.TRANSPORT_PROPERTY の設定によって指定されます。

接続名は次のように指定できます。

- MQQueueManager コンストラクターで明示的に指定。

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーの、プロパティー MQC.HOST_NAME_PROPERTY とオプションの MQC.PORT_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 明示的な MQEnvironment 値として設定。

```
MQEnvironment.Hostname
```

  MQEnvironment.Port(オプション)。

- MQEnvironment.properties ハッシュ・テーブルの、プロパティー MQC.HOST_NAME_PROPERTY とオプションの MQC.PORT_PROPERTY で設定。

チャネル名は次のように指定できます。

- MQQueueManager コンストラクターで明示的に指定。

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーのプロパティー MQC.CHANNEL_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- 明示的な MQEnvironment 値として設定。

```
MQEnvironment.Channel
```

- MQEnvironment.properties ハッシュ・テーブルのプロパティー MQC.CHANNEL_PROPERTY で設定。

TRANSPORT プロパティーは次のように指定できます。

- MQQueueManager コンストラクターのハッシュ・テーブル・エントリーのプロパティー MQC.TRANSPORT_PROPERTY で設定。

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- MQEnvironment.properties ハッシュ・テーブルのプロパティー MQC.TRANSPORT_PROPERTY で設定。

以下のいずれかの値を使用して、必要な接続タイプを選択してください。

MQC.TRANSPORT_MQSERIES_BINDINGS - サーバーとして接続
MQC.TRANSPORT_MQSERIES_CLIENT - 非 XA クライアントとして接続
MQC.TRANSPORT_MQSERIES_XACLIENT - XA クライアントとして接続
MQC.TRANSPORT_MQSERIES_MANAGED - 非 XA 管理対象クライアントとして接続

カスタマイズ値 NMQ_MQ_LIB を設定して、次の表に示す接続タイプを明示的に選択できます。

NMQ_MQ_LIB 値	**接続タイプ**
mqic.dll	非 XA クライアントとして接続
mqicxa.dll	XA クライアントとして接続
mqm.dll	サーバーまたは非 XA クライアントとして接続
managed	非 XA 管理対象クライアントとして接続
注 : 旧リリースとの互換性を保つため、値 mqic32.dll および mqic32xa.dll は mqic.dll および mqicxa.dll の同義として受け入れられます。 ただし、mqm.dll および mqm.pdb は、バージョン 7.1 以降のクライアント・パッケージの一部でしかありません。	

お客様の環境で使用できない接続タイプを選択すると (例えば mqic32xa.dll を指定したが XA サポートがない場合)、WebSphere MQ .NET によって例外がスローされます。

NMQ_MQ_LIB を「managed」に設定すると、クライアントは管理対象の WebSphere MQ 問題診断テスト、.NET データ変換、およびその他の管理対象の低レベル WebSphere MQ 関数を使用するようになります。

NMQ_MQ_LIB にそれ以外のすべての値を設定した場合、.NET プロセスは非管理対象 WebSphere MQ の問題診断テストとデータ変換、および他の非管理対象の低レベルの WebSphere MQ 関数を使用するようになります (WebSphere MQ MQI クライアントまたはサーバーがシステムにインストールされていることを想定しています)。

WebSphere MQ .NET は次のように接続タイプを選択します。

1. MQC.TRANSPORT_PROPERTY が指定されている場合は、MQC.TRANSPORT_PROPERTY の値に従って接続します。

   ただし、MQC.TRANSPORT_PROPERTY を MQC.TRANSPORT_MQSERIES_MANAGED に設定しても、クライアント・プロセスが管理下で実行されることが保証されるわけではないので注意してください。 この設定を行っていても、以下のケースではクライアントは管理対象になりません。

   - プロセス内の他のスレッドが、MQC.TRANSPORT_MQSERIES_MANAGED 以外の値に設定された MQC.TRANSPORT_PROPERTY に接続している場合。
   - NMQ_MQ_LIB が「managed」に設定されていない場合、問題診断テスト、データ変換、および他の低レベル関数は、完全には管理対象になりません (WebSphere MQ MQI クライアントまたはサーバーがシステムにインストールされていることを想定しています)。
2. 接続名が指定され、チャネル名が指定されていない場合、またはチャネル名が指定され、接続名が指定されていない場合は、エラーがスローされます。
3. 接続名とチャネル名の両方が指定されている場合は、次のようになります。
   - NMQ_MQ_LIB が mqic32xa.dll に設定されている場合は、XA クライアントとして接続します。
   - NMQ_MQ_LIB が managed に設定されている場合は、管理対象クライアントとして接続します。
   - それ以外の場合は、非 XA クライアントとして接続します。
4. NMQ_MQ_LIB が指定されている場合は、NMQ_MQ_LIB の値に従って接続します。

5. WebSphere MQ サーバーがインストールされている場合は、サーバーとして接続します。
6. WebSphere MQ MQI クライアントがインストールされている場合は、非 XA クライアントとして接続します。
7. それ以外の場合は、管理対象クライアントとして接続します。

WebSphere MQ classes for .NET の構成ファイル

.NET クライアント・アプリケーションは、WebSphere MQ MQI クライアント構成ファイルと、管理対象接続タイプを使用している場合は .NET アプリケーション構成ファイルを使用することができます。 アプリケーション構成ファイルの設定が優先されます。

クライアント構成ファイル

WebSphere MQ classes for .NET クライアント・アプリケーションでは、他のすべての WebSphere MQ MQI クライアントと同じ方法でクライアント構成ファイルを使用できます。 このファイルには通常 mqclient.ini という名前が付けられますが、別のファイル名を指定することも可能です。 クライアント構成ファイルについて詳しくは、構成ファイルを使用したクライアントの構成 WebSphere MQ MQI クライアント構成ファイルを参照してください。

WebSphere MQ classes for .NET に関係する属性は、WebSphere MQ MQI クライアント構成ファイルの以下の属性のみです。 その他の属性を指定しても、効果はありません。

スタンザ	属性
CHANNELS	CCSID
CHANNELS	ChannelDefinitionDirectory
CHANNELS	ChannelDefinitionFile
CHANNELS	ServerConnectionParms
ClientExitPath	ExitsDefaultPath
ClientExitPath	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

これらの属性はすべて、該当する環境変数を使用して指定変更することができます。

アプリケーション構成ファイル

管理対象接続タイプを使用して実行している場合は、.NET アプリケーション構成ファイルを使用して、WebSphere MQ クライアント構成ファイル、および等価な環境変数をオーバーライドすることもできます。

.NET アプリケーション構成ファイルの設定は、管理対象接続タイプで実行されている場合にのみ有効で、その他の接続タイプでは無視されます。

.NET アプリケーション構成ファイルとそのフォーマットは、.NET Framework 内で一般使用できるよう Microsoft によって定義されていますが、この資料で言及している特定のセクション名、キー、および値は Websphere MQ に固有のものです。

.NET アプリケーション構成ファイルのフォーマットは、数多くのセクション で構成されます。 各セクションには、1 つ以上のキー が含まれ、各キーには、関連付けられた値 があります。 次の例に、TCP/IP KeepAlive プロパティーを制御するために .NET アプリケーション構成ファイルで使用されている、セクション、キー、および値を示します。

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

.NET アプリケーション構成ファイルのセクション名およびキーで使用されたキーワードは、クライアント構成ファイルで定義されたスタンザおよび属性のキーワードと完全に一致します。

詳しくは、Microsoft の資料を参照してください。

コード例

以下の C# コードの例は、次の 3 つのアクションを実行するアプリケーションを示しています。

1. キュー・マネージャーに接続します。
2. メッセージを SYSTEM.DEFAULT.LOCAL.QUEUE に書き込みます。
3. メッセージを返します。

また、接続タイプを変更する方法も示します。

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
  /// </summary>
  /// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
  static Hashtable init(String connectionType)
  {
    Hashtable connectionProperties = new Hashtable();

    // Add the connection type
    connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

    // Set up the rest of the connection properties, based on the
    // connection type requested
    switch(connectionType)
```

```
    {
      case MQC.TRANSPORT_MQSERIES_BINDINGS:
        break;
      case MQC.TRANSPORT_MQSERIES_CLIENT:
      case MQC.TRANSPORT_MQSERIES_XACLIENT:
      case MQC.TRANSPORT_MQSERIES_MANAGED:
        connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
        connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
        break;
    }

    return connectionProperties;
  }
  /// <summary>
  /// The main entry point for the application.
  /// </summary>
  [STAThread]
  static int Main(string[] args)
  {
    try
    {
      Hashtable connectionProperties = init(connectionType);

      // Create a connection to the queue manager using the connection
      // properties just defined
      MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

      // Set up the options on the queue we want to open
      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

      // Now specify the queue that we want to open,and the open options
      MQQueue system_default_local_queue =
        qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

      // Define a WebSphere MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                           // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define a WebSphere MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it a WebSphere MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("A WebSphere MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
```

```
    }

    return 0;
  }//end of start
}//end of sample
```

キュー・マネージャーに対する操作

このセクションでは、WebSphere MQ classes for .NET を使用したキュー・マネージャーへの 接続方法とキュー・マネージャーからの切断方法について説明します。

WebSphere MQ 環境のセットアップ

クライアント接続を使用してキュー・マネージャーに接続するには、まず、WebSphere MQ 環境をセットアップする必要があります。

注 : サーバー・バインディング・モードで WebSphere MQ classes for .NET を使用する場合、 この手順は不要です。

.NET プログラミング・インターフェースにより、NMQ_MQ_LIB カスタマイズ値を使用することができ、クラス MQEnvironment も組み込まれます。 このクラスでは、接続を試行する際に使用される、以下にリストしたような詳細情報を指定できます。

- チャネル名
- ホスト名
- ポート番号
- チャネル出口
- SSL
- ユーザー ID およびパスワード

MQEnvironment クラスについて詳しくは、 MQEnvironment .NET クラス を参照してください。

チャネル名とホスト名を指定するには、次のコードを使用します。

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

デフォルトでは、クライアントはポート 1414 で WebSphere MQ リスナーに接続しようとします。 別のポートを指定するには、次のコードを使用します。

```
MQEnvironment.Port = nnnn;
```

キュー・マネージャーへの接続

ここまでで、次のように MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続できる状態になっています。

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

キュー・マネージャーから切断するには、キュー・マネージャーで Disconnect メソッドを呼び出します。

```
queueManager.Disconnect();
```

キュー・マネージャーへの接続を試行する際には、キュー・マネージャー上での照会 (inq) 権限を持っている必要があります。 照会権限がない場合、接続試行は失敗します。

Disconnect メソッドを呼び出すと、そのキュー・マネージャーからアクセスした、オープンしているキューとプロセスがすべてクローズされます。 しかし、使用を終えた時点でこれらのリソースを明示的にク

ローズするプログラミングの習慣を付けておくことをお勧めします。 リソースをクローズするには、各リソースに関連付けられているオブジェクトで Close メソッドを使用します。

キュー・マネージャーの Commit メソッドと Backout メソッドは、手続き型インターフェースで使用される MQCMIT 呼び出しと MQBACK 呼び出しを置き換えるものです。

キューおよびトピックへのアクセス

MQQueueManager のメソッドまたは適切なコンストラクターを使用して、キューとトピックにアクセスできます。

キューにアクセスするには、MQQueueManager クラスのメソッドを使用します。 MQOD (オブジェクト記述子構造体) は、これらのメソッドのパラメーターに縮小されます。 例えば、MQQueueManager オブジェクトで表される queueManager というキュー・マネージャーのキューをオープンするには、次のコードを使用します。

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

options パラメーターは、MQOPEN 呼び出しの Options パラメーターと同じです。

AccessQueue メソッドは、クラス MQQueue の新しいオブジェクトを戻します。

キューの使用が終了したら、次の例に示すように、Close() メソッドを使用してそのキューをクローズします。

```
queue.Close();
```

WebSphere MQ .NET では、MQQueue コンストラクターを使用してキューを作成することもできます。 パラメーターは、accessQueue メソッドと同じパラメーター、および使用するインスタンス化済みの MQQueueManager オブジェクトを指定するキュー・マネージャー・パラメーターです。 以下に例を示します。

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

この方法でキュー・オブジェクトを構成すると、MQQueue の独自のサブクラスを作成できます。

同様に、MQQueueManager クラスのメソッドを使用してトピックにもアクセスすることができます。 トピックを開くには、AccessTopic() メソッドを使用します。 このメソッドは、クラス MQTopic の新規オブジェクトを返します。 トピックの使用が完了したなら、MQTopic の Close() メソッドを使用してトピックを閉じます。

MQTopic コンストラクターを使用してトピックを作成することもできます。 トピック用のコンストラクターはいくつもあります。詳しくは、MQTopic .NET クラスを参照してください。

メッセージの処理

メッセージは、キューまたはトピック・クラスのメソッドを使用して処理されます。 新しいメッセージを作成するには、新しい MQMessageobject を作成します。

キューまたはトピックへのメッセージの書き込みには、MQQueue または MQTopic クラスの Put() メソッドを使用します。 キューまたはトピックからのメッセージの読み取りには、MQQueue または MQTopic クラスの Get() メソッドを使用します。 MQPUT と MQGET によってバイトの配列の書き込みと読み取りを行う手続き型インターフェースと 異なり、WebSphere MQ classes for .NET は MQMessage クラスのインスタンスの書き込みと 読み取りを行います。 MQMessage クラスは、実際のメッセージ・データが含まれるデ

ータ・バッファーと、そのメッセージについて記述するすべての MQMD (メッセージ記述子) パラメーターをまとめてカプセル化します。

新しいメッセージを作成するには、MQMessage クラスの新しいインスタンスを作成し、WriteXXX メソッドを使用してメッセージ・バッファーにデータを書き込みます。

新規メッセージ・インスタンスが作成されると、MQMD の初期値および言語ごとの宣言 で定義されているように、すべての MQMD パラメーターが自動的にデフォルト値に設定されます。 MQQueue の Put() メソッドも、MQPutMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQPMO 構造体を表します。 次の例では、メッセージを作成して、キューに書き込んでいます。

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.WriteInt(25);

String name = "Charlie Jordan";
myMessage.WriteUTF(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message!
queue.Put(myMessage,pmo);
```

MQQueue の Get() メソッドは、キューから読み取ったメッセージを表す MQMessage の新しいインスタンスを返します。 また、このメソッドは MQGetMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQGMO 構造体を表します。

Get() メソッドは着信メッセージに合わせて内部バッファーのサイズを自動調整するため、最大メッセージ・サイズを指定する必要はありません。 返されたメッセージのデータにアクセスするには、MQMessage クラスの ReadXXX メソッドを使用します。

次の例は、メッセージをキューから読み取る方法を示しています。

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.Get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.ReadInt();
String name1 = theMessage.ReadUTF();
```

encoding メンバー変数を設定することにより、読み取りメソッドと書き込みメソッドで使用する数字形式を変更できます。

characterSet メンバー変数を設定することにより、読み取りと書き込みのストリングで使用する文字セットを変更できます。

詳しくは、MQMessage .NET クラスを参照してください。

注 : MQMessage の WriteUTF() メソッドは、ストリングに含まれる Unicode バイトだけでなく、ストリングの長さを自動的にエンコードします。 メッセージが 別の .NET プログラムによって (ReadUTF() を使用して) 読み取られる場合、これがストリング情報を送信する最も簡単な方法です。

メッセージ・プロパティーの処理

メッセージ・プロパティーでは、メッセージを選択したり、メッセージのヘッダーにアクセスすることなくメッセージについての情報を取得したりすることができます。 MQMessage クラスには、プロパティーの取得および設定を行うメソッドが含まれています。

メッセージ・プロパティーを使用して、アプリケーションに処理するメッセージを選択させたり、MQMD または MQRFH2 ヘッダーにアクセスすることなくメッセージについての情報を取得させたりすることができます。 また、メッセージ・プロパティーは、WebSphere と JMS アプリケーションの間の通信を行いやすくします。 WebSphere MQ のメッセージ・プロパティーについて詳しくは、メッセージ・プロパティーを参照してください。

MQMessage クラスには、プロパティーのデータ型に従ってプロパティーを取得および設定するいくつものメソッドが用意されています。 取得のメソッドには Get*Property という形式の名前が付いており、設定のメソッドには Set*Property という形式の名前が付いています。アスタリスク (*) の部分には、以下のいずれかのストリングが入ります。

- ブール値
- Byte
- Bytes
- 二重
- 浮動
- Int
- Int2
- Int4
- Int8
- Long
- オブジェクト
- 短
- ストリング

例えば、WebSphere MQ プロパティー myproperty (文字ストリング) を取得するには、呼び出し `message.GetStringProperty('myproperty')` を使用します。 オプションで、プロパティー記述子を渡すことができます。プロパティー記述子を追加する処理は WebSphere MQ によって行われます。

エラーの処理

WebSphere MQ classes for .NET で発生したエラーを、`try` ブロックと `catch` ブロックを使用して処理します。

.NET インターフェースのメソッドは、完了コードと理由コードを返しません。 代わりに、WebSphere MQ 呼び出しから返される完了コードと理由コードがどちらもゼロではないときは、必ず例外をスローします。これによってプログラム・ロジックが簡単になり、WebSphere MQ への呼び出しごとに戻りコードを検査する必要がなくなります。 プログラムのどの部分で障害に対処するかを決めることができます。 これらのポイントでは、次の例のように、コードを `try` と `catch` のブロックで囲むことができます。

```
try
{
  myQueue.Put(messageA,PutMessageOptionsA);
  myQueue.Put(messageB,PutMessageOptionsB);
}
catch (MQException ex)
{
  // This block of code is only executed if one of
  // the two put methods gave rise to a non-zero
  // completion code or reason code.
  Console.WriteLine("An error occurred during the put operation:" +
                    "CC = " + ex.CompletionCode +
                    "RC = " + ex.ReasonCode);
  Console.WriteLine("Cause exception:" + ex );
}
```

属性値の取得と設定

クラス MQManagedObject、MQQueue、および MQQueueManager には、属性値を取得または設定できるメソッドがあります。 MQQueue では、キューのオープン時に適切な照会フラグおよび設定フラグを指定している場合にのみメソッドが機能することに注意してください。

共通属性に関しては、MQQueueManager クラスと MQQueue クラスは MQManagedObject というクラスから継承します。 このクラスは、Inquire() インターフェースと Set() インターフェースを定義します。

new 演算子を使用して新規キュー・マネージャー・オブジェクトを作成すると、そのオブジェクトは自動的に inquire 用にオープンされます。 AccessQueue() メソッドを使用してキュー・オブジェクトにアクセスした場合、オブジェクトは照会操作用または 設定操作用に自動的にはオープンされません。 これにより、一部のタイプのリモート・キューでは問題が発生することがあります。 Inquire メソッドと Set メソッドを使用してキューのプロパティーを設定するには、AccessQueue() メソッドの openOptions パラメーターで適切な照会フラグと設定フラグを指定する必要があります。

inquire メソッドと set メソッドは、次の 3 つのパラメーターを取ります。

- selectors 配列
- intAttrs 配列
- charAttrs 配列

配列の長さは常に判明しているため、MQINQ で使用する SelectorCount、IntAttrCount、および CharAttrLength の各パラメーターは必要ありません。 次の例は、キューの照会を行う方法を示しています。

```
//inquire on a queue
int [ ] selectors = new int [2] ;
int [ ] intAttrs = new int [1] ;
byte [ ] charAttrs = new byte [MQC.MQ_Q_DESC_LENGTH];
selectors [0] = MQC.MQIA_DEF_PRIORITY;
selectors [1] = MQC.MQCA_Q_DESC;
queue.Inquire(selectors,intAttrs,charAttrs);
ASCIIEncoding enc = new ASCIIEncoding();
String s1 = "";
s1 = enc.GetString(charAttrs);
```

これらのオブジェクトの属性はすべて照会可能です。 属性のサブセットはオブジェクトのプロパティーとして公開されます。 オブジェクトの属性のリストについては、オブジェクトの属性を参照してください。 オブジェクトのプロパティーについては、該当するクラスの説明を参照してください。

マルチスレッド・プログラム

.NET 実行時環境は、本質的にマルチスレッドです。 WebSphere MQ classes for .NET では、キュー・マネージャー・オブジェクトを複数のスレッドで共用できますが、ターゲット・キュー・マネージャーへのすべてのアクセスが確実に同期するようになります。

始動時にキュー・マネージャーに接続して、キューをオープンする単純なプログラムを考えてみましょう。 このプログラムは、画面上に 1 つのボタンを表示します。 ユーザーがこのボタンをクリックすると、プログラムはキューからメッセージを取り出します。 この場合、アプリケーションの初期化は 1 つのスレッドで行われ、ボタンが押されたことに対する応答としてのコードの実行は別のスレッド (ユーザー・インターフェース・スレッド) で行われます。

WebSphere MQ .NET を実装すると、特定の接続 (MQQueueManager オブジェクト・インスタンス) に対しては、ターゲットの WebSphere MQ キュー・マネージャーへのすべてのアクセスは、必ず同期化されます。 デフォルトの動作では、キュー・マネージャーに呼び出しを行ったスレッドは、その接続に対して行われている他のすべての呼び出しが完了するまでブロックされます。 プログラム内の複数のスレッドから同じキュー・マネージャーに同時にアクセスする必要がある場合は、同時アクセスが必要なスレッドごとに新しい MQQueueManager オブジェクトを作成します。 (これは、スレッドごとに別の MQCONN 呼び出しを発行する ことと同じです。)

デフォルトの接続オプションが MQC.MQCNO_HANDLE_SHARE_NONE または MQC.MQCNO_SHARE_NO_BLOCK によって指定変更された場合、キュー・マネージャーは同期しなくなります。

.NET でのクライアント・チャネル定義テーブルの使用

WebSphere MQ 用の .NET クラスと一緒にクライアント・チャネル定義テーブル (CCDT) を使用することができます。 管理対象接続を使用しているか、非管理対象接続を使用しているかに応じて、CCDT の場所を指定する方法は異なります。

非 XA または XA 非管理対象クライアント接続タイプ

非管理対象接続タイプを使用して、次の 2 つの方法で CCDT の場所を指定できます。

- 環境変数 MQCHLLIB を使用して、テーブルが置かれているディレクトリーを指定し、MQCHLTAB を使用して、テーブルのファイル名を指定します。
- クライアント構成ファイルを使用します。 CHANNELS スタンザで、属性 ChannelDefinitionDirectory を使用して、テーブルが置かれているディレクトリーを指定し、ChannelDefinitionFile を使用して、ファイル名を指定します。

クライアント構成ファイルと環境変数の両方で場所が指定されている場合は、環境変数が優先します。 この機能を使用して、標準の位置をクライアント構成ファイルで指定し、必要な場合に環境変数を使用して標準の位置をオーバーライドすることができます。

管理対象クライアント接続タイプ

管理対象接続タイプを使用して、次の 3 つの方法で CCDT の場所を指定できます。

- .NET アプリケーション構成ファイルを使用します。 CHANNELS セクションで、キー ChannelDefinitionDirectory を使用して、テーブルが置かれているディレクトリーを指定し、ChannelDefinitionFile を使用して、ファイル名を指定します。
- 環境変数 MQCHLLIB を使用して、テーブルが置かれているディレクトリーを指定し、MQCHLTAB を使用して、テーブルのファイル名を指定します。
- クライアント構成ファイルを使用します。 CHANNELS スタンザで、属性 ChannelDefinitionDirectory を使用して、テーブルが置かれているディレクトリーを指定し、ChannelDefinitionFile を使用して、ファイル名を指定します。

複数の方法で場所が指定されている場合は、環境変数がクライアント構成ファイルに優先し、.NET アプリケーション構成ファイルがその他の 2 つの方法に優先します。 このフィーチャーを使用することで、クライアント構成ファイルで標準の場所を指定し、必要に応じて、環境変数またはアプリケーション構成ファイルを使用して標準の場所を指定変更できます。

どのチャネル定義を使用するかを .NET アプリケーションが判別する方法

WebSphere MQ .NET クライアント環境では、使用するチャネル定義をさまざまな方法で指定できます。 チャネル定義の指定が複数存在することもあります。 アプリケーションは、1 つ以上のソースからチャネル定義を取り出します。

複数のチャネル定義が存在する場合は、次の優先順位で、使用する 1 つのチャネル定義が選択されます。

1. MQQueueManager コンストラクターで、明示的に、またはハッシュ・テーブルに *MQC.CHANNEL_PROPERTY* を組み込むことによって、指定されるプロパティー。
2. MQEnvironment.properties ハッシュ・テーブル内のプロパティー *MQC.CHANNEL_PROPERTY*。
3. MQEnvironment 内のプロパティー *Channel*。
4. .NET アプリケーション構成ファイル、セクション名 CHANNELS、キー ServerConnectionParms (管理対象接続にのみ適用)
5. *MQSERVER* 環境変数。
6. クライアント構成ファイル、スタンザ CHANNELS、属性 ServerConnectionParms
7. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、.NET アプリケーション構成ファイルで指定されます (管理対象接続にのみ適用)。
8. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、環境変数 *MQCHLIB* および *MQCHLTAB* を使用して指定されます。
9. クライアント・チャネル定義テーブル (CCDT)。 CCDT の場所は、クライアント構成ファイルを使用して指定されます。

項目 1 から 3 まででは、チャネル定義は、アプリケーションによって提供された値から、フィールド単位で作成されます。 これらの値は、さまざまなインターフェースを使用して提供することができ、インター

フェースごとに複数の値が存在することもあります。 フィールド値は、次の優先順位に従って、チャネル定義に追加されます。

1. MQQueueManager コンストラクター上の *connName* の値。
2. MQQueueManager.properties ハッシュ・テーブルからのプロパティーの値。
3. MQEnvironment.properties ハッシュ・テーブルからのプロパティーの値。
4. MQEnvironment フィールドとして設定された値 (例えば、MQEnvironment.Hostname、MQEnvironment.Port)

項目 4 から 6 まででは、チャネル定義全体が値として提供されます。 チャネル定義で指定されていないフィールドは、システム・デフォルトを取ります。 チャネルとそのフィールドを定義するその他の方法からの値が、これらの指定にマージされることはありません。

項目 7 から 9 まででは、チャネル定義全体が CCDT から取得されます。 チャネルが定義されたときに明示的に指定されなかったフィールドは、システム・デフォルトを取ります。 チャネルとそのフィールドを定義するその他の方法からの値が、これらの指定にマージされることはありません。

IBM WebSphere MQ でのチャネル出口の使用。ネット

クライアント・バインディングを使用する場合は、他のクライアント接続の場合のようにチャネル出口を使用できます。 管理対象バインディングを使用する場合は、適切なインターフェースを実装する出口プログラムを作成する必要があります。

クライアント・バインディング

クライアント・バインディングを使用すると、チャネル出口で説明されている方法でチャネル出口を使用できます。 管理対象バインディング用に作成したチャネル出口は使用できません。

管理対象バインディング

管理対象接続を使用している場合は、出口を実行するために、適切なインターフェースを実装する新しい .NET クラスを定義します。 WebSphere MQ パッケージには次の 3 つの出口インターフェースが定義されています。

- MQSendExit
- MQReceiveExit
- MQSecurityExit

注 : このインターフェースを使用して作成されたユーザー出口は、非管理対象環境ではチャネル出口としてサポートされません。

次の例では、3 つすべてを実装するクラスを定義しています。

```
class MyMQExits : MQSendExit, MQReceiveExit, MQSecurityExit
{
   // This method comes from the send exit
  byte[] SendExit(MQChannelExit       channelExitParms,
                  MQChannelDefinition channelDefinition,
                  byte[]              dataBuffer
                  ref int             dataOffset
                  ref int             dataLength
                  ref int             dataMaxLength)
  {
     // complete the body of the send exit here
  }


  // This method comes from the receive exit
  byte[] ReceiveExit(MQChannelExit       channelExitParms,
                     MQChannelDefinition channelDefinition,
                     byte[]              dataBuffer
                     ref int             dataOffset
                     ref int             dataLength
                     ref int             dataMaxLength)
  {
     // complete the body of the receive exit here
```

```
  }


  // This method comes from the security exit
  byte[] SecurityExit(MQChannelExit       channelExitParms,
                      MQChannelDefinition channelDefParms,
                      byte[]              dataBuffer
                      ref int             dataOffset
                      ref int             dataLength
                      ref int             dataMaxLength)
  {
     // complete the body of the security exit here
  }

}
```

各出口は、MQChannelExit および MQChannelDefinition オブジェクトのインスタンスが渡されます。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

送信出口によって送信されるデータと、セキュリティー出口または受信出口によって受信されるデータは、出口のパラメーターを使用して指定されます。

オフセット *dataOffset* で長さ *dataLength* のバイト配列 *dataBuffer* のデータは、チャネル出口が送信出口であれば、送信出口で送信されようとしているデータです。チャネル出口がセキュリティー出口または受信出口であれば、セキュリティー出口または受信出口で受信されたデータです。 パラメーター *dataMaxLength* は、*dataBuffer* の出口で使用可能な (*dataOffset* からの) 最大長を示します。 注: セキュリティー出口では、 出口が初めて呼び出された場合、またはパートナー側がデータを送信しないことを初めて選択した場合は、dataBuffer がヌルになることがあります。

戻りでは、*dataOffset* と *dataLength* の値は、返されたバイト配列内のオフセットと 長さを示すように設定される必要があります。返されたバイト配列は、次に .NET クラスが使用します。 送信出口では、これは送信するデータを示します。 セキュリティー出口または受信出口では、解釈されるデータです。 通常、出口はバイト配列を返します (例外として、データを送信しないことを選択できるセキュリティー出口、および INIT または TERM という理由で呼び出されるすべての出口があります)。 したがって、作成できる最も単純な出口は、dataBuffer だけを戻す出口です。

次は、考えられる最も単純な出口の本体です。

```
{
  return dataBuffer;
}
```

MQChannelDefinition クラス

V7.5.0.6 Version 7.5.0, Fix Pack 6 以降、管理対象 .NET クライアント・アプリケーションで指定されたユーザー ID とパスワードは、クライアント・セキュリティー出口に渡される IBM WebSphere MQ .NET MQChannelDefinition クラスで設定されます。 セキュリティー出口はユーザー ID とパスワードを MQCD.RemoteUserIdentifier フィールドと MQCD.RemotePassword フィールドにコピーします (407 ページの『セキュリティー出口の作成』を参照してください)。

チャネル出口の指定 (管理対象クライアント)

MQQueueManager オブジェクトを (MQEnvironment または MQQueueManager コンストラクターで) 作成する際にチャネル名と接続名を指定する場合、チャネル出口を 2 つの方法で指定できます。

この 2 つの方法を優先順に示します。

1. MQQueueManager コンストラクターでのハッシュ・テーブル・プロパティー MQC.SECURITY_EXIT_PROPERTY、MQC.SEND_EXIT_PROPERTY、 または MQC.RECEIVE_EXIT_PROPERTY の引き渡し
2. MQEnvironment の SecurityExit、SendExit、または ReceiveExit プロパティーの設定

チャネル名と接続名を指定しない場合、チャネル出口は、クライアント・チャネル定義テーブル (CCDT) から選択されたチャネル定義から使用されます。 チャネル定義に保管されている値を指定変更することはで

きません。 チャネル定義テーブルについて詳しくは、クライアント・チャネル定義テーブルおよび 590 ページの『.NET でのクライアント・チャネル定義テーブルの使用』を参照してください。

どちらの場合も、次の形式のストリングで指定します。

```
Assembly_name(Class_name)
```

Class_name は、IBM.WMQ.MQSecurityExit、IBM.WMQ.MQSendExit、または IBM.WMQ.MQReceiveExit インターフェースを (必要に応じて) 実装する .NET クラスの、名前空間の指定を含んだ完全修飾名です。 *Assembly_name* は、このクラスを含むアセンブリーの、ファイル拡張子を含めて完全修飾された場所です。 MQEnvironment または MQQueueManager のプロパティーを使用する場合、ストリングの長さは最大 999 文字です。 ただし、チャネル出口名が CCDT に指定されている場合、128 文字までに制限されます。 必要に応じて、.NET クライアント・コードは 指定されたストリングを解析し、指定されたクラスのインスタンスをロードして作成します。

チャネル出口ユーザー・データの指定 (管理対象クライアント)

チャネル出口は、関連するユーザー・データを持つことができます。 MQQueueManager オブジェクトを (MQEnvironment または MQQueueManager コンストラクターで) 作成する際にチャネル名と接続名を指定する場合、ユーザー・データを 2 つの方法で指定できます。

この 2 つの方法を優先順に示します。

1. MQQueueManager コンストラクターでのハッシュ・テーブル・プロパティー MQC.SECURITY_USERDATA_PROPERTY、MQC.SEND_USERDATA_PROPERTY、 または MQC.RECEIVE_USERDATA_PROPERTY の引き渡し
2. MQEnvironment の SecurityUserData、SendUserData、または ReceiveUserData プロパティーの設定

チャネル名と接続名を指定しなかった場合は、クライアント・チャネル定義テーブル (CCDT) から選択されたチャネル定義からの出口ユーザー・データ値が使用されます。 チャネル定義に保管されている値を指定変更することはできません。 チャネル定義テーブルについて詳しくは、クライアント・チャネル定義テーブルおよび 590 ページの『.NET でのクライアント・チャネル定義テーブルの使用』を参照してください。

どちらの場合も、最大 32 文字のストリングで指定します。

.NET でのクライアントの自動再接続

予測しない接続中断時にクライアントが自動的にキュー・マネージャーに再接続するように設定できます。

クライアントは、キュー・マネージャーが停止したりネットワークやサーバーに障害が発生したりした場合に、 予期せずにキュー・マネージャーから切断されることがあります。

クライアント自動再接続を使用しない場合、接続に障害が発生したときにエラーが生成されます。 エラー・コードは、接続を再確立するのに役立ちます。

クライアント自動再接続機能を使用するクライアントは、再接続可能なクライアントと呼ばれます。 再接続可能なクライアントを作成するには、 キュー・マネージャーへの接続時に、再接続オプションと呼ばれる特定のオプションを指定します。

クライアント・アプリケーションが WebSphere MQ .NET クライアントである場合、 クライアント自動再接続をオンにするには、 MQQueueManager クラスを使用してキュー・マネージャーを作成する際に CONNECT_OPTIONS_PROPERTY に適切な値を指定します。 CONNECT_OPTIONS_PROPERTY の値の詳細については、再接続オプションを参照してください。

クライアント・アプリケーションが常に接続と再接続を行う相手として、 同じ名前のキュー・マネージャー、同じキュー・マネージャー、または クライアント接続テーブルで同じ QMNAME によって定義された 任意のセットのキュー・マネージャー (詳細については、 CCDT 内のキュー・マネージャー・グループを参照) から選択できます。

Secure Sockets Layer (SSL) サポート

以下のセクションは、管理対象クライアントには適用されません。

WebSphere MQ classes for .NET クライアント・アプリケーションでは、Secure Sockets Layer (SSL) 暗号化がサポートされます。 SSL は、通信の暗号化、認証、およびメッセージの整合性を提供します。 これは一般的に、インターネット上やイントラネット内で、任意の 2 つのピアの間の通信を保護するために使用されます。

SSL の有効化

SSL がサポートされているのは、クライアント接続の場合のみです。 SSL を有効にするには、キュー・マネージャーとの通信時に使用する CipherSpec を指定し、ターゲット・チャネルに設定された CipherSpec に一致させる必要があります。

SSL を有効にするには、MQEnvironment の SSLCipherSpec 静的メンバー変数を使用して CipherSpec を指定します。 次の例では、SECURE.SVRCONN.CHANNEL という 名前の SVRCONN チャネルに接続しています。このチャネルは、NULL_MD5 の CipherSpec で SSL を要求するように設定されています。

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "NULL_MD5";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

CipherSpec のリストについては、CipherSpec の指定を参照してください。

SSLCipherSpec プロパティーは、接続プロパティーのハッシュ・テーブルの MQC.SSL_CIPHER_SPEC_PROPERTY を使用して設定することもできます。

SSL を使用して正常に接続するには、キュー・マネージャーによって提示される証明書を認証できる認証局のルート証明書チェーンを使用して、クライアント鍵ストアをセットアップしておく必要があります。 同様に、SVRCONN チャネルの SSLClientAuth を MQSSL_CLIENT_AUTH_REQUIRED に設定してある場合は、キュー・マネージャーによって信頼されている識別個人証明書がクライアントの鍵ストアに含まれていなければなりません。

キュー・マネージャーの識別名の使用

キュー・マネージャーは、識別名 (DN) が含まれる SSL 証明書を使用して自身の ID を示します。

WebSphere MQ .NET クライアント・アプリケーションでこの DN を使用すると、確実に正しい キュー・マネージャーと通信することができます。 DN パターンを指定するときは、MQEnvironment の sslPeerName 変数を使用します。 例えば、以下のように設定すると、

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

キュー・マネージャーが QMGR で始まる共通名を持つ証明書を提示した場合にのみ、接続を成功させます。少なくとも 2 つの組織単位名があり、最初の組織単位名は IBM で、2 番目の組織単位名は WEBSPHERE でなければなりません。

SSLPeerName プロパティーは、接続プロパティーのハッシュ・テーブルの MQC.SSL_PEER_NAME_PROPERTY を使用して設定することもできます。 識別名、およびピア名の設定規則の詳細については、セキュリティーを参照してください。

SSLPeerName が設定されている場合、それが有効なパターンに設定されており、キュー・マネージャーが一致する証明書を提示した場合のみ接続が成功します。

SSL の使用時のエラー処理

SSL を使用してキュー・マネージャーに接続した場合に、WebSphere MQ classes for .NET から 発行される理由コードは次のとおりです。

MQRC_SSL_NOT_ALLOWED
SSLCipherSpec プロパティーが設定されましたが、バインディング接続が使用されました。 SSL をサポートしているのは、クライアント接続のみです。

MQRC_SSL_PEER_NAME_MISMATCH
SSLPeerName プロパティーで指定された DN パターンが、キュー・マネージャーによって提示された DN と一致しませんでした。

MQRC_SSL_PEER_NAME_ERROR
SSLPeerName プロパティーで指定された DN パターンが有効ではありませんでした。

.NET モニターの使用

重要な情報については、Windows 上のプライマリー・インストールでのみ使用できる機能 を参照してください。

.NET モニターは、WebSphere MQ トリガー・モニターに類似したアプリケーションです。 メッセージがモニター対象のキューで受信されると常にインスタンス化される .NET コンポーネントと、その後そのメッセージを処理する .NET コンポーネントを作成できます。 .NET モニターは、runmqdnm コマンドで開始され、endmqdnm コマンドで停止されます。 これらのコマンドについて詳しくは、runmqdnm および endmqdnm を参照してください。

.NET モニターを使用するには、amqmdnm.dll で定義される IMQObjectTrigger インターフェースを実装するコンポーネントを作成します。

コンポーネントはトランザクションであってもトランザクションでなくてもかまいません。 トランザクション・コンポーネントは、System.EnterpriseServices.ServicedComponent から継承され、RequiresTransaction または SupportsTransaction として登録される必要があります。 .NET モニターがトランザクションを既に開始しているため、RequiresNew として登録することはできません。

コンポーネントは、runmqdnm から MQQueueManager、MQQueue、および MQMessage オブジェクトを受け取ります。 また、 runmqdnm が開始されたときに、 *-u* コマンド行オプションを使用してユーザー・パラメーター・ストリングが指定されていれば、それを受け取ることもあります。 コンポーネントは、モニター対象のキューに到着したメッセージの内容を MQMessage オブジェクトで受け取ることに注意してください。 コンポーネントは、キュー・マネージャーに接続して、 キューをオープンしたり、メッセージそのものを取得する必要はありません。 コンポーネントは必要に応じてメッセージを処理し、.NET モニターに制御を返す必要があります。

コンポーネントがトランザクション・コンポーネントとして作成されている場合、そのコンポーネントは、System.EnterpriseServices.ServicedComponent で提供されている機能を使用してトランザクションをコミットするのかロールバックするのかを登録します。

コンポーネントがメッセージだけでなく MQQueueManager オブジェクトと MQQueue オブジェクトを受け取ると、 そのメッセージに対する完全なコンテキスト情報が得られるため、例えば WebSphere MQ に個別に接続することなく、同じ キュー・マネージャー上の別のキューをオープンすることができます。

コード例

このトピックには、.NET モニターからメッセージを取得して印刷するコンポーネントのコード例が 2 つあります。 1 つはトランザクション処理を使用し、もう 1 つは非トランザクション処理を使用します。 3 番目の例は、上の 2 つの例に適用できる共通ユーティリティー・ルーチンを示しています。 すべてのコード例は C# で作成されています。

例 1: トランザクション処理

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/
using System;
using System.EnterpriseServices;

using IBM.WMQ;
using IBM.WMQMonitor;

[assembly: ApplicationName("dnmsamp")]

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll TranAssembly.cs
//
// run (with dotnet monitor)
//
```

```
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c Tran

namespace dnmsamp
{
  [TransactionAttribute(TransactionOption.Required)]
  public class Tran : ServicedComponent, IMQObjectTrigger
  {
    Util util = null;

    [AutoComplete(true)]
    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("Tran");

      if (param != null)
        util.Print("PARAM: '" +param.ToString() + "'");

      util.PrintMessage(message);

      //System.Console.WriteLine("SETTING ABORT");
      //ContextUtil.MyTransactionVote = TransactionVote.Abort;

      System.Console.WriteLine("SETTING COMMIT");
      ContextUtil.SetComplete();
      //ContextUtil.MyTransactionVote = TransactionVote.Commit;
    }
  }
}
```

例 2: 非トランザクション処理

```
/*********************************************************************/
/* Licensed materials, property of IBM                               */
/* 63H9336                                                           */
/* (C) Copyright IBM Corp. 2005, 2024.                               */
/*********************************************************************/

using System;

using IBM.WMQ;
using IBM.WMQMonitor;

// build:
//
// csc -target:library -reference:amqmdnet.dll;amqmdnm.dll NonTranAssembly.cs
//
// run (with dotnet monitor)
//
// runmqdnm -m <QMNAME> -q <QNAME> -a dnmsamp.dll -c NonTran
namespace dnmsamp
{
  public class NonTran : IMQObjectTrigger
  {
    Util util = null;

    public void Execute(MQQueueManager qmgr, MQQueue queue,
        MQMessage message, string param)
    {
      util = new Util("NonTran");

      try
      {
        util.PrintMessage(message);
      }

      catch (Exception ex)
      {
        System.Console.WriteLine(">>> NonTran\n{0}", ex.ToString());
      }
    }
  }
}
```

例 3: 共通ルーチン

```
/**********************************************************************/
/* Licensed materials, property of IBM                                */
/* 63H9336                                                            */
/* (C) Copyright IBM Corp. 2005, 2024.                                */
/**********************************************************************/

using System;

using IBM.WMQ;

namespace dnmsamp
{
    /// <summary>
    /// Summary description for Util.
    /// </summary>
    public class Util
    {
    /* ------------------------------------------------------------------ */
    /* Default prefix string of the namespace.                             */
    /* ------------------------------------------------------------------ */
    private string prefixText = "dnmsamp";

    /* ------------------------------------------------------------------ */
    /* Constructor that takes the replacement prefix string to use.       */
    /* ------------------------------------------------------------------ */
    public Util(String text)
    {
      prefixText = text;
    }

    /* ------------------------------------------------------------------ */
    /* Display an arbitrary string to the console.                        */
    /* ------------------------------------------------------------------ */
    public void Print(String text)
    {
      System.Console.WriteLine("{0} {1}\n", prefixText, text);
    }



 /* ------------------------------------------------------------------ */
    /* Display the content of the message passed to the console.          */
    /* ------------------------------------------------------------------ */
    public void PrintMessage(MQMessage message)
    {
      if (message.Format.CompareTo(MQC.MQFMT_STRING) == 0)
      {
        try
        {
          string messageText = message.ReadString(message.MessageLength);

          Print(messageText);
        }

        catch(Exception ex)
        {
          Print(ex.ToString());
        }
      }
      else
      {
        Print("UNRECOGNISED FORMAT");
      }
    }

    /* ------------------------------------------------------------------ */
    /* Convert the byte array into a hex string.                          */
    /* ------------------------------------------------------------------ */
    static public string ToHexString(byte[] byteArray)
    {
      string hex = "0123456789ABCDEF";

      string retString = "";

      for(int i = 0; i < byteArray.Length; i++)
      {
        int h = (byteArray[i] & 0xF0)>>4;
        int l = (byteArray[i] & 0x0F);
```

```
        retString += hex.Substring(h,1) + hex.Substring(l,1);
      }

      return retString;
    }

    }
}
```

WebSphere MQ .NET プログラムのコンパイル

様々な言語で作成した .NET アプリケーションをコンパイルするためのコマンドの実例を示します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

WebSphere MQ classes for .NET を使用して C# アプリケーションをビルドするには、次のコマンドを 使用します。

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib:MQ_INSTALLATION_PATH\bin /out:MyProg.exe MyProg.cs
```

WebSphere MQ classes for .NET を使用して Visual Basic アプリケーションを作成するには、次の コマンドを使用します。

```
vbc /r:System.dll /r:MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

WebSphere MQ classes for .NET を使用して Managed C++ アプリケーションを作成するには、次の コマンドを使用します。

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

他の言語については、その言語のベンダーによって提供されている資料を参照してください。

WebSphere MQ .NET プログラムのトレース

WebSphere MQ .NET では、WebSphere MQ プログラムの場合と同じように、MQI を使用してトレース機能の開始と制御を行います。

ただし、strmqtrc コマンドの -i および -p パラメーター (プロセスおよびスレッドの ID、および名前付きプロセスをそれぞれ指定できるパラメーター) は効果がありません。

通常、トレース機能は、IBM サービスから要求された場合以外は使用する必要はありません。

トレース・コマンドについては、Windows でのトレースの使用を参照してください。

Microsoft Windows Communication Foundation (WCF) 用の IBM WebSphere MQ カスタム・チャネル

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

関連概念

600 ページの『.NET 3 を使用した WCF 用の WebSphere MQ カスタム・チャネルの紹介』
.NET 3 で Windows Communication Foundation (WCF) 用の WebSphere MQ カスタム・チャネルを使用するプログラマーが利用できる情報の概要です。

604 ページの『WCF 用の WebSphere MQ カスタム・チャネルの使用』
Windows Communication Foundation (WCF) 用の WebSphere MQ V7 カスタム・チャネルを使用するプログラマーが利用できる情報の概要です。

621 ページの『WCF サンプルの使用』

Windows Communication Foundation (WCF) サンプルでは、WebSphere MQ カスタム・チャネルの使用方法の簡単な例をいくつか紹介します。

627 ページの『WebSphere MQ 用の WCF カスタム・チャネルの問題判別』
WebSphere MQ トレースを使用すると、WebSphere MQ のコードのさまざまな部分で実行されている内容について詳細な情報を収集することができます。 Windows Communication Foundation (WCF) を使用する場合、Microsoft WCF インフラストラクチャー・トレースに統合された WCF カスタム・チャネル・トレースに対して、別個のトレース出力が生成されます。

.NET 3 を使用した WCF 用の WebSphere MQ カスタム・チャネルの紹介

.NET 3 で Windows Communication Foundation (WCF) 用の WebSphere MQ カスタム・チャネルを使用するプログラマーが利用できる情報の概要です。

WCF 用の WebSphere MQ カスタム・チャネルとは何か

WebSphere MQ カスタム・チャネルは、Microsoft Windows Communication Foundation (WCF) 統一プログラミング・モデルを使用するトランスポート・チャネルです。

Microsoft Windows Communication Foundation フレームワークは Microsoft .NET 3 で導入されたもので、.NET アプリケーションおよびサービスを、それらの接続に使用するトランスポートおよびプロトコルとは独立して開発できるため、 サービスまたはアプリケーションが配置される環境に応じた代替トランスポートまたは構成を使用できます。

接続は、以下の必要な組み合わせを含むチャネル・スタックを作成することで、 WCF によって実行時に管理されます。

- プロトコル要素: オプションの一連の要素。 WS-* 標準などのプロトコルをサポートするように 1 つ以上の要素を追加することも、 要素を追加しないこともできます。
- メッセージ・エンコーダー: ワイヤー形式でメッセージを直列化するように制御する、スタック内の必須要素。
- トランスポート・チャネル: 直列化したメッセージをそのエンドポイントにトランスポートさせる、スタック内の必須要素。

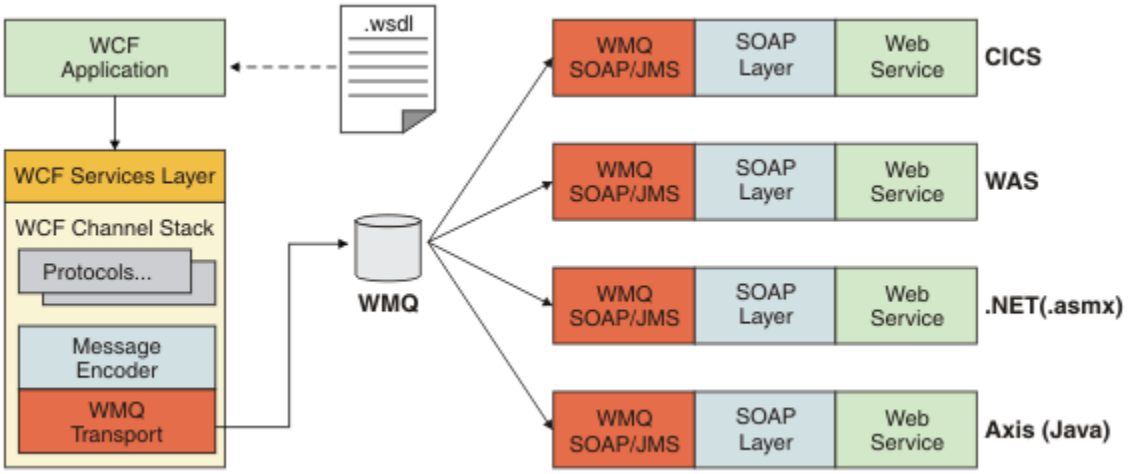
WebSphere MQ カスタム・チャネルは、トランスポート・チャネルであるため、WCF カスタム・バインディングを使用して、アプリケーションでの必要に応じて メッセージ・エンコーダーおよびオプションのプロトコルとペアにする必要があります。 このように、WCF を使用するように開発されたアプリケーションは、WebSphere MQ 用のカスタム・チャネルを使用して、Microsoft が提供する組み込みトランスポートを使用する場合と同じ方法でデータを送受信することができ、WebSphere MQ の非同期の、スケーラブルで信頼性の高いメッセージング機能を簡単に統合することができます。 サポートされる機能の完全なリストについては、604 ページの『WCF カスタム・チャネルのフィーチャーと機能』を参照してください。

WCF 用の WebSphere MQ カスタム・チャネルはいつ、どういう理由で使用するのか

WebSphere MQ カスタム・チャネルを使用して、Microsoft で提供される組み込みトランスポートと同じ方法で WCF クライアントとサービスの間でデータを送受信でき、 それによってアプリケーションは WCF 統一プログラミング・モデル内の WebSphere MQ のフィーチャーにアクセスできます。

WCF 用の WebSphere MQ カスタム・チャネルの標準的使用パターン・シナリオとしては、WebSphere MQ (SOAP/JMS) 上でホストされる Web サービスへのインターフェースがあります。

メッセージは、 WebSphere MQ の SOAP over JMS メッセージ形式を使用して伝送されます。これにより、WCF クライアントおよびサービスは、他の WebSphere MQ アプリケーション、またはこの形式と互換性のあるホスティング環境 (WebSphere Application Server、 CICS、Axis v1 で実行される Web サービスを含む) を呼び出したり呼び出したり呼び出したりすることができます。 および .asmx (.NET) (以下の図を参照)



SOAP over JMS について詳しくは、950 ページの『WebSphere MQ transport for SOAP』を参照してください。

の図の標準的なシナリオの例を以下に示します。

1. Web サービスは、WebSphere Application Server 内でホストされ、WebSphere Application Server の SOAP over JMS のサポートを使用して WebSphere MQ で公開されます。
2. そのサービスを記述する WSDL 文書を WCF ツールで使用して、クライアント・プロキシーおよび構成を生成することができます。このクライアント・プロキシーおよび構成は、カスタム・チャネルなどの適切な WCF チャネル・スタックを作成します。
3. 次に、クライアント・アプリケーションは、プロキシーを使用して、他の Web サービスと同じ方法でこの Web サービスを開始することができます。

このチャネルは、通常は WCF テキスト/SOAP メッセージ・エンコーダーで使用されますが、 必要であれば他の WCF メッセージ・エンコーダーとペアにすることができます。 代替エンコーダーを使用すると、SOAP over JMS をサポートしないネイティブの WebSphere MQ アプリケーションと制限付きで統合することもできます。ただしこれは、このチャネルの主要な役割ではありません。

WCF 環境でカスタム・チャネルを使用する主な利点は、以下のとおりです。

- 非同期呼び出し: 応答不要送信クライアント操作のサポート。 この操作では、クライアントは、応答の再ルーティングやマルチポップなどのサービスおよびフィーチャーの可用性を確保する必要がありません。
- 信頼性の高い拡張特性: キュー・ベースのメッセージングにより、 予想したとおりにシステムに能力を追加できます。
- サービス品質: メッセージは具体的でトレースしやすく、 管理が容易です。

WCF 用の WebSphere MQ カスタム・チャネルのソフトウェア要件およびインストール手順

このトピックでは、WCF 用の WebSphere MQ カスタム・チャネルのソフトウェア要件とインストール手順の概要について説明します。

WCF 用の WebSphere MQ カスタム・チャネルは、WebSphere MQ V7 以降のキュー・マネージャーにのみ接続可能です。

WebSphere MQ 用の WCF カスタム・チャネルのソフトウェア要件

ここでは、 WebSphere MQ の WCF カスタム・チャネルのソフトウェア要件をリストします。

ランタイム環境

- Microsoft .NET Framework v3.0 以降をホスト・マシンにインストールする必要があります。

- *Java and .NET Messaging and Web Services* が、WebSphere MQ 7.0.1 インストーラーによる作業の一環としてデフォルトでインストールされます。 カスタム・チャネルに必要な .NET アセンブリーをグローバル・アセンブリー・キャッシュにインストールします。

注 : WebSphere MQ V7.0.1 をインストールする前に Microsoft .NET Framework v2.0 以降がインストールされていない場合、WebSphere MQ 製品のインストールはエラーなしで続行されますが、WebSphere MQ カスタム・チャネルは使用できません。 WebSphere MQ 7.0.1 をインストールした後で .NET Framework をインストールする場合は、 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd スクリプトを実行して WebSphere MQ カスタム・チャネルをアクティブ化する必要があります。ここで、 *WMQInstallDir* は、WebSphere MQ 7.0.1 がインストールされているディレクトリーです。 このスクリプトにより、必要なアセンブリーがグローバル・アセンブリー・キャッシュ (GAC) にインストールされます。 実施されたアクションを記録する一連の amqi*.log ファイルが、%TEMP% ディレクトリーに作成されます。 .NET が以前のバージョンから v3.0 以上にアップグレードされた場合 (例えば、.NET v2.0 から)、amqiRegisterdotNet.cmd スクリプトを再実行する必要はありません。

開発環境

- Microsoft Visual Studio 2008 または Windows Software Development Kit for .NET 3.0 以降。
- サンプル・ソリューション・ファイルを作成するには、Microsoft .NET Framework V3.5 以降をホスト・マシンにインストールする必要があります。

注 : WebSphere MQ V7.0.1 をインストールする前に Microsoft .NET Framework v2.0 以降がインストールされていない場合、WebSphere MQ 製品のインストールはエラーなしで続行されますが、WebSphere MQ カスタム・チャネルは使用できません。 WebSphere MQ 7.0.1 をインストールした後で .NET Framework をインストールする場合は、 *WMQInstallDir*\bin\amqiRegisterdotNet.cmd スクリプトを実行して WebSphere MQ カスタム・チャネルをアクティブ化する必要があります。ここで、 *WMQInstallDir* は、WebSphere MQ 7.0.1 がインストールされているディレクトリーです。 このスクリプトにより、必要なアセンブリーがグローバル・アセンブリー・キャッシュ (GAC) にインストールされます。 実施されたアクションを記録する一連の amqi*.log ファイルが、%TEMP% ディレクトリーに作成されます。 .NET が以前のバージョンから v3.0 以上にアップグレードされた場合 (例えば、.NET v2.0 から)、amqiRegisterdotNet.cmd スクリプトを再実行する必要はありません。

WCF 用の WebSphere MQ カスタム・チャネル: 何がインストールされるのか

WebSphere MQ カスタム・チャネルは、Microsoft Windows Communication Foundation (WCF) 統一プログラミング・モデルを使用するトランスポート・チャネルです。 カスタム・チャネルは、 デフォルトで WebSphere MQ 7.0.1 インストールの一環としてインストールされます。

WCF 用の WebSphere MQ カスタム・チャネル

WCF 用の WebSphere MQ カスタム・チャネルは、デフォルトで WebSphere MQ 7.0.1 インストールの一部としてインストールされます。カスタム・チャネルとその依存関係は、デフォルトでインストールされる Java and .NET Messaging and Web Services コンポーネントに含まれます。 以前のバージョンから WebSphere MQ 7.0.1 にアップグレードする場合、 Java and .NET Messaging and Web Services コンポーネントが以前のインストール済み環境に既にインストールされている場合は、更新によってデフォルトで WCF 用の WebSphere MQ カスタム・チャネルがインストールされます。

Java and .NET Messaging and Web Services コンポーネントには IBM.XMS.WCF.dll ファイルが含まれ、 IBM.XMS.WCF.dll ファイルは WCF インターフェース・クラスを含むメイン・カスタム・チャネル・アセンブリーです。 このファイルは、グローバル・アセンブリー・キャッシュ (GAC) にインストールされ、ディレクトリー *MQ_INSTALLATION_PATH*\bin にもあります。ここで、 *MQ_INSTALLATION_PATH* は、 WebSphere MQ 7.0.1 がインストールされているディレクトリーです。

カスタム・チャネルを使用するために必要な主要クラスは *Namespace: IBM.XMS.WCF* にあり、以下のとおりです。

Transport Binding Name	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement
Transport Binding Importer:	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter

Transport Binding Config	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig

WebSphere MQ カスタム・チャネルのサンプル

これらのサンプルでは、WCF 用の WebSphere MQ カスタム・チャネルの使用方法についていくつかの簡単な例を紹介します。 サンプルとそれに関連するファイルは、*MQ_INSTALLATION_PATH*\tools\wcf\samples\ ディレクトリーにあります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。 WebSphere MQ カスタム・チャネルのサンプルについて詳しくは、621 ページの『WCF サンプルの使用』を参照してください。

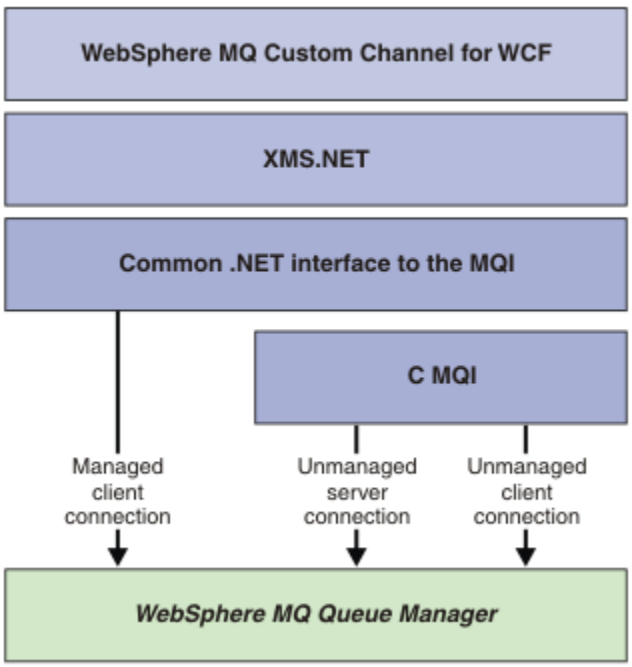
svcutil.exe.config

svcutil.exe.config は、 Microsoft WCF svcutil クライアント・プロキシー生成ツールで カスタム・チャネルを認識できるようにするために必要な構成設定の例です。 svcutil.exe.config ファイルは、*MQ_INSTALLATION_PATH*\tools\wcf\docs\examples\ ディレクトリーにあります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。 svcutil.exe.config の使用について詳しくは、 618 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』を参照してください。

WCF のアーキテクチャー

WCF 用の WebSphere MQ カスタム・チャネルは、 IBM Message Service Client for .NET (XMS .NET) API の上に統合されます。

WCF アーキテクチャーを以下の図に示します。



すべての必須コンポーネントは、WebSphere MQ V7.0.1 インストール済み環境にデフォルトでインストールされます。

管理対象クライアント接続、非管理対象サーバー接続、および非管理対象クライアント接続の 3 つの接続があります。 これらの接続について詳しくは、608 ページの『WCF 接続オプション』を参照してください。

WCF 用の WebSphere MQ カスタム・チャネルの使用

Windows Communication Foundation (WCF) 用の WebSphere MQ V7 カスタム・チャネルを使用するプログラマーが利用できる情報の概要です。

Microsoft Windows Communication Foundation は、 Microsoft .NET Framework 3 の Web サービスおよびメッセージング・サポートをサポートします。 WebSphere MQ V7 は、 Microsoft が提供する組み込みチャネルと同じ方法で、.NET Framework 3 の WCF 内でカスタム・チャネルとして使用できるようになりました。

カスタム・チャネルを通じて移送されるメッセージは、WebSphere MQ V7 の SOAP over JMS 実装に従ってフォーマットされます。 これにより、アプリケーションは WCF または WebSphere SOAP over JMS サービス・インフラストラクチャーによってホストされるサービスと通信できるようになります。 SOAP over JMS について詳しくは、950 ページの『WebSphere MQ transport for SOAP』を参照してください。

WCF カスタム・チャネルのフィーチャーと機能

WCF カスタム・チャネルのフィーチャーと機能については、以下のトピックを参照してください。

WCF カスタム・チャネルの形状

Microsoft Windows Communication Foundation (WCF) カスタム・チャネル内で WebSphere MQ を使用する際に可能なカスタム・チャネルの形状の概要です。

WCF 用の WebSphere MQ カスタム・チャネルは、次の 2 つのチャネル形状をサポートします。

- 片方向
- 要求/応答

WCF は、ホストされているサービス・コントラクトに応じて、 自動的にチャネル形状を選択します。

IsOneWay パラメーターのみを使用するメソッドを含むコントラクトは、 片方向チャネル形状でサービスが提供されます。例えば、次のようになります。

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

片方向と要求/応答のメソッドが混在するコントラクト、またはすべてが要求/応答のメソッドを含むコントラクトは、要求/応答チャネル形状でサービスが提供されます。 以下に例を示します。

```
[OperationContract]
int subtract(int a, int b);

[OperationContract(IsOneWay = true)]
void printString(string text);
```

注 : 片方向と要求/応答のメソッドが同じコントラクト内で混在するときには、 意図したとおりに動作するかどうかを確認する必要があります。 特に、混合環境内で作業する際に、 片方向メソッドはサービスからヌル応答を受信するまで待機するため、このことは重要です。

片方向チャネル

WCF 用の WebSphere MQ 片方向カスタム・チャネルは、例えば、片方向チャネル形状を使用して WCF クライアントからメッセージを送信する場合に使用されます。 チャネルは、例えば、クライアント・キュー・マネージャーから WCF サービス上のキューに送信するなど、 単一方向にのみメッセージを送信できます。

要求/応答チャネル

WCF 用の WebSphere MQ 要求/応答カスタム・チャネルは、 例えば、双方向に非同期でメッセージを送信する場合に使用されます。 非同期メッセージングには、同じクライアント・インスタンスを使用する必要があります。 チャネルは、例えば、クライアント・キュー・マネージャーから WCF サービス上のキューに

送信するなど、 単一方向にメッセージを送信し、次に、WCF からクライアント・キュー・マネージャー上のキューに応答メッセージを送信できます。

WCF URI パラメーターの名前と値

connectionFactory
connectionFactory パラメーターは必須です。 このパラメーターの構文については、 Web サービス・デプロイメントの URI 構文とパラメーター を参照してください。

initialContextFactory
initialContextFactory パラメーターは必須で、WebSphere Application Server およびその他の製品との互換性を保つために、その値を「com.ibm.mq.jms.Nojndi」に設定する必要があります (1010 ページの『サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する』を参照)。

WCF カスタム・チャネルの保証配信

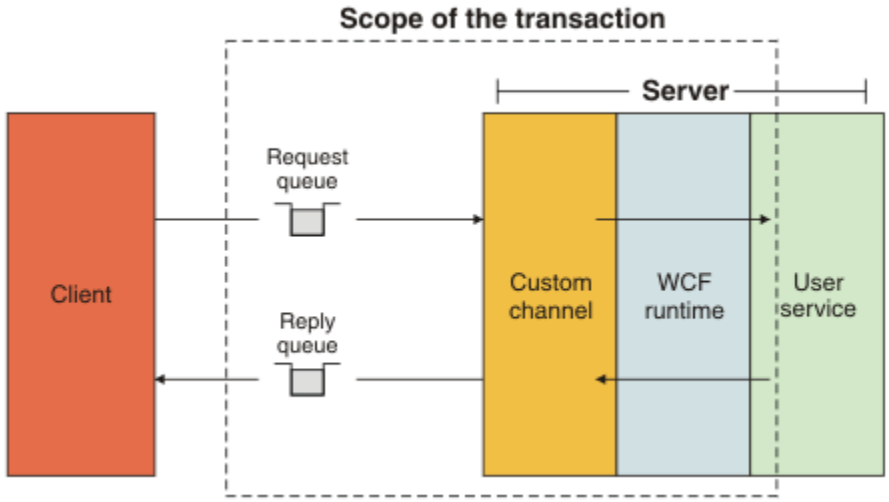
保証配信により、サービス要求または応答が確実に実施され、失われることはありません。

要求メッセージの受信および応答メッセージの送信は、ローカル・トランザクションの同期点下で行われ、ランタイム障害の場合には、これらのメッセージをロールバックすることができます。 ランタイム障害の例としては、サービスから未処理例外がスローされた、メッセージをサービスにディスパッチできなかった、応答メッセージを配信できなかった、などがあります。

`AssuredDelivery` は、サービスで受信された要求メッセージ、およびサービスから送信された応答メッセージを、ランタイム障害が発生しても失わないことを保証する、サービス契約で指定できる保証配信属性です。

システム障害や電源異常が発生した場合もメッセージを確実に保持するためには、メッセージを持続メッセージとして送信する必要があります。 持続メッセージを使用するには、クライアント・アプリケーションのエンドポイント URI でこのオプションを指定しておく必要があります。 URI プロパティーの設定について詳しくは、Web サービス・デプロイメントに関する URI の構文とパラメーターを参照してください。

分散トランザクションはサポートされず、トランザクションのスコープが、WebSphere MQ で実行される要求および応答メッセージ処理の枠を超えることはありません。 サービス内で実行された処理はすべて、メッセージの再受信の原因となった障害の結果として再実行できます。 次の図に、トランザクションのスコープを示します。



以下の例に示すように、`AssuredDelivery` 属性をサービス・クラスに適用することにより、確実性のある送達が有効になります。

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
```

```
    }
}
```

AssuredDelivery 属性を使用するときは、次の点に注意してください。

- メッセージがロールバックされて再受信された場合に障害が再発生する可能性があるとチャネルが判断すると、そのメッセージは有害メッセージとして扱われ、再処理用の要求キューには返されません。 例えば、受信メッセージが正しくフォーマットされていなかったり、サービスにディスパッチできなかったりする場合です。 サービス操作からスローされた未処理例外は、メッセージの再配信回数が、要求キューのバックアウトしきい値プロパティーで指定された最大回数に達するまで、必ず再送信されます。 詳しくは、606 ページの『WCF カスタム・チャネル有害メッセージ』を参照してください。
- チャネルは、トランザクションの整合性を確保するために単一実行スレッドを使用して、各要求メッセージの読み取り、処理、および応答をアトミック操作として実行します。 サービス操作を並行して実行できるように、チャネルは WCF がチャネルの複数のインスタンスを作成することを可能にします。 要求の処理で使用可能なチャネル・インスタンスの数は、バインディング・プロパティー MaxConcurrentCalls で制御されます。 詳しくは、614 ページの『WCF バインディング構成オプション』を参照してください。
- 保証配信機能は、IOperationInvoker および IErrorHandler の両方の WCF 拡張ポイントを使用します。 これらの拡張ポイントを外部で使用するアプリケーションは、事前に登録済みの拡張ポイントが確実に呼び出されるようにする必要があります。 IErrorHandler に対してこれを実行できないと、エラーが報告されない可能性が生じます。 IOperationInvoker でこれを実行できなかった場合は、WCF が応答を停止する可能性があります。

WCF カスタム・チャネル・セキュリティー

WCF 用の WebSphere MQ カスタム・チャネルは、キュー・マネージャーへの非管理対象クライアント接続の場合にのみ SSL の使用をサポートします。

SSL は、次の 2 つの方法のどちらかで指定できます。

- SOAP over JMS URI で SSL を直接指定します。 SSL のオプションについて詳しくは、SSL と WebSphere MQ transport for SOAP を参照してください。
- クライアント・チャネル定義テーブル (CCDT) 内の項目を使用して SSL を指定します。 CCDT について詳しくは、クライアント・チャネル定義テーブルを参照してください。

WCF クライアント・チャネル定義テーブル *(CCDT)*

WCF 用の WebSphere MQ カスタム・チャネルは、クライアント・チャネル定義テーブル (CCDT) の使用による、クライアント接続の接続情報の構成をサポートしています。

CCDT は、次の 2 つの環境変数によって制御します。

- *MQCHLLIB* で、テーブルが置かれているディレクトリーを指定します。
- *MQCHLTAB* で、テーブルのファイル名を指定します。

チャネル定義テーブルは、SOAP over JMS URI で直接指定することはできません。 これらの環境変数が定義されている場合、これらの環境変数は、URI で指定されているクライアント接続詳細に優先します。

クライアント・チャネル定義テーブルについて詳しくは、 クライアント・チャネル定義テーブル を参照してください。

関連概念

クライアント・チャネル定義テーブル

WCF カスタム・チャネル有害メッセージ

サービスが要求メッセージの処理に失敗したり、応答キューへの応答メッセージの配信に失敗したりすると、そのメッセージは有害メッセージとして扱われます。

有害要求メッセージ

要求メッセージを処理できないと、そのメッセージは有害メッセージとして扱われます。 このアクションにより、サービスは、処理不能の同じメッセージを二度と受信しなくなります。 処理不能の要求メッセージを有害メッセージとして扱うためには、次のいずれかの状況が当てはまる必要があります。

- メッセージのバックアウト・カウントが、要求キューで指定されたバックアウトしきい値を超えた場合。これが発生するのは、サービスに対して保証配信が指定されている場合だけです。 保証配信について詳しくは、605 ページの『WCF カスタム・チャネルの保証配信』を参照してください。
- メッセージが正しくフォーマットされていないため、SOAP over JMS メッセージとして解釈できなかった場合。

有害応答メッセージ

サービスが応答キューへの応答メッセージの配信に失敗すると、その応答メッセージは有害メッセージとして扱われます。 応答メッセージの場合は、このアクションにより、後で問題判別に役立てるために応答メッセージを取得することが可能になります。

有害メッセージの処理

有害メッセージに対するアクションは、キュー・マネージャーの構成、およびメッセージのレポート・オプションに設定されている値によって異なります。 SOAP over JMS では、次のレポート・オプションがデフォルトで要求メッセージに対し設定されています。これらのレポート・オプションは構成できません。

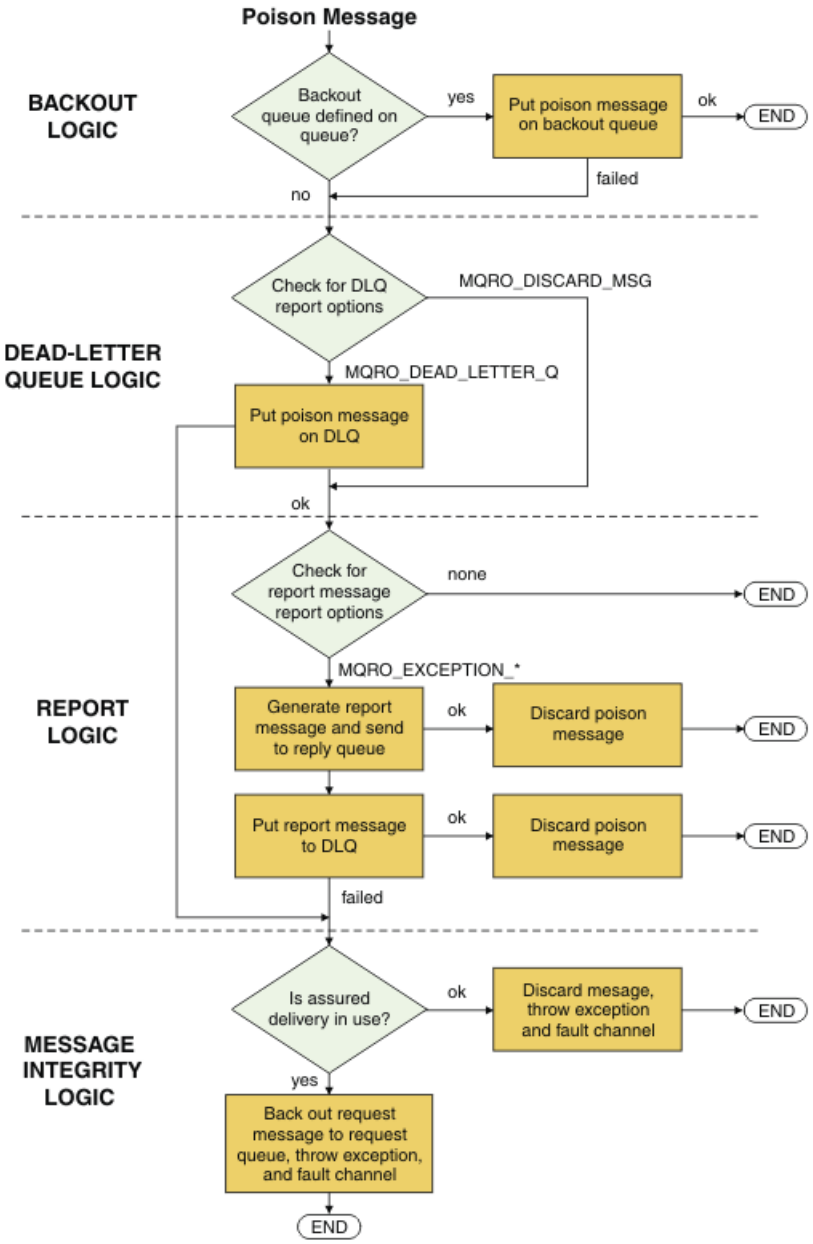
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

SOAP over JMS では、次のレポート・オプションがデフォルトで応答メッセージに対し設定されています。このレポート・オプションは構成できません。

- MQRO_DEAD_LETTER_Q

メッセージの送信元が WCF 以外の場合は、その送信元の資料を参照してください。

次の図に、有効なアクション、および有害メッセージ処理が失敗した場合に取られるステップを示します。



WCF 接続オプション

WCF 用の WebSphere MQ カスタム・チャネルをキュー・マネージャーに接続するモードは 3 つあります。どのタイプの接続が最も必要に適しているかを考慮してください。

接続オプションについて詳しくは、580 ページの『接続の違い』を参照してください。

WCF アーキテクチャーについて詳しくは、603 ページの『WCF のアーキテクチャー』を参照してください。

非管理対象クライアント接続

このモードの接続では、WebSphere MQ クライアントは、ローカル・マシンまたはリモート・マシンのいずれかで稼働している WebSphere MQ サーバーに接続されます。

WCF 用の WebSphere MQ カスタム・チャネルを WebSphere MQ クライアントとして使用するには、そのカスタム・チャネルを WebSphere MQ MQI クライアントと一緒に WebSphere MQ サーバーにインストールしても、または別個のマシンにインストールしても構いません。

非管理対象サーバー接続

サーバー・バインディング・モードで使用する場合、WCF 用の WebSphere MQ カスタム・チャネルは、ネットワーク経由で通信するのではなく、キュー・マネージャー API を使用します。 バインディング接続を使用すると、ネットワーク接続を使用した場合よりも、WebSphere MQ アプリケーションのパフォーマンスが向上します。

バインディング接続を使用するには、WCF 用の WebSphere MQ カスタム・チャネルを WebSphere MQ サーバーにインストールする必要があります。

管理対象クライアント接続

このモードの接続では、WebSphere MQ クライアントは、ローカル・マシンまたはリモート・マシンのいずれかで稼働している WebSphere MQ サーバーに接続されます。

このモードで接続する .NET 3 用の WebSphere MQ カスタム・チャネル・クラスは、.NET 管理対象コードに残り、ネイティブ・サービスに対する呼び出しを行いません。 管理対象コードについて詳しくは、Microsoft の資料を参照してください。

管理対象クライアントの使用には、いくつかの制限があります。 これらの制限事項について詳しくは、580 ページの『管理対象クライアント接続』を参照してください。

WCF 用の WebSphere MQ カスタム・チャネルの作成および構成

WCF 用の WebSphere MQ V7 カスタム・チャネルは、 Microsoft で提供されるトランスポート WCF チャネルと同じ方法で機能します。 WCF 用の WebSphere MQ カスタム・チャネルは、2 つの方法のどちらかで作成できます。

このタスクについて

WebSphere MQ カスタム・チャネルは、WCF トランスポート・チャネルとして WCF に統合されるため、メッセージ・エンコーダーおよびオプションのプロトコル・チャネルとペアにする必要があります。 それによって、アプリケーションで使用できる完全なチャネル・スタックを作成することができます。 完全なチャネル・スタックを正常に作成するためには、以下の 2 つの要素が必要になります。

1. バインディング定義: アプリケーション・チャネル・スタックの作成に必要な要素を指定します。 それらの要素には、トランスポート・チャネル、メッセージ・エンコーダー、および任意のプロトコル、 さらに一般的な構成設定が含まれます。 カスタム・チャネルの場合、バインディング定義は WCF カスタム・バインディングの形式で作成する必要があります。
2. エンドポイント定義: サービス・コントラクトをバインディング定義にリンクし、アプリケーションが接続可能な宛先を示す実際の接続 URI も提供します。 カスタム・チャネルの場合、この URI は SOAP over JMS URI の形式になります。

これらの定義は、次の 2 つの方法のいずれかで作成できます。

- 管理: 定義はアプリケーション構成ファイル (例えば、`app.config`) に詳細を提供することによって作成されます。
- プログラマチック: 定義はアプリケーション・コードから直接作成されます。

定義を作成する際にどちらの方法を使用するかについては、以下のように、アプリケーションの要件に基づいて決定する必要があります。

- 構成のための管理方法では、アプリケーションを再作成しなくても、サービスおよびクライアントのデプロイメント後の詳細を柔軟に変更することができます。
- 「プログラマチック」による構成方法では、構成エラーに対する保護が強化され、 また実行時に構成を動的に生成することができます。

アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、*WCF* カスタム・チャネルを管理的に作成する

WCF 用の WebSphere MQ カスタム・チャネルは、トランスポート・レベルの WCF チャネルです。 このカスタム・チャネルを使用するには、エンドポイントおよびバインディングを定義する必要があります。こ

れらの定義は、アプリケーション構成ファイルにバインディングおよびエンドポイント情報を指定することにより定義することができます。

WCF 用の WebSphere MQ カスタム・チャネル (トランスポート・レベルの WCF チャネル) を構成して使用するには、 バインディングおよびエンドポイント定義を定義する必要があります。 バインディング定義はチャネルの構成情報を保持し、エンドポイント定義は接続の詳細を保持します。 これらの定義は、以下の 2 つの方法で作成できます。

- アプリケーション・コードからプログラマチックに直接作成する。詳しくは、『612 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』』を参照してください。
- アプリケーション構成ファイルで詳細を指定することにより、管理的に作成する。この方法については、以下の手順で説明します。

一般的に、クライアントまたはサービスのアプリケーション構成ファイルには、*yourappname*.exe.config という名前が付けられています。ここで、*yourappname* はご使用のアプリケーションの名前です。 アプリケーション構成ファイルを最も簡単に変更する方法は、以下のようにして、SvcConfigEditor.exe という名前の Microsoft サービス構成エディター・ツールを使用する方法です。

- SvcConfigEditor.exe 構成エディター・ツールを開始します。 このツールのデフォルトのインストール・ロケーションは *Drive:*\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe です。ここで、ドライブ*:* はインストール・ドライブの名前です。

ステップ 1: バインディング要素拡張を追加して、WCF がカスタム・チャネルを検出できるようにする

1. **拡張 > 拡張 > バインディング・エレメント** を右クリックしてメニューを開き、「**新規**」 を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *73.* 新規バインディング要素フィールド

フィールド	**値**
名前	IBM.XMS.WCF.SoapJmsIbmTransportChannel
Type	グローバル・アセンブリー・キャッシュ (GAC) で IBM.XMS.WCF.dll にナビゲートし、IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig を選択します。

ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する

1.「**バインディング**」を右クリックしてメニューを開き、「**新規バインディング構成**」を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *74.* 新規バインディング構成フィールド

フィールド	**値**
名前	CustomBinding_WMQ
BindingElement 1	textMessageEncoding (MessageVersion: Soap11)
BindingElement 2	IBM.XMS.WCF.SoapJmsIbmTransportChannel

ステップ 3: バインディング・プロパティーを指定する

1. 『610 ページの『ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する』』で作成したバインディングから *IBM.XMS.WCF.SoapJmsIbmTransportChannel* トランスポート・バインディングを選択します。
2. 『614 ページの『WCF バインディング構成オプション』』の説明に従って、プロパティーのデフォルト値に必要な変更を加えます。

ステップ 4: エンドポイント定義を作成する

610 ページの『ステップ 2: カスタム・チャネルと WCF メッセージ・エンコーダーを結合するカスタム・バインディング定義を作成する』で作成したカスタム・バインディングを参照し、サービスの接続詳細を提供するエンドポイント定義を作成します。 この情報を指定する方法は、エンドポイント定義がクライアント・アプリケーション用の定義か、サービス・アプリケーション用の定義かによって異なります。

クライアント・アプリケーションの場合、以下のようにして、エンドポイント定義をクライアント・セクションに追加します。

1. 「**クライアント**」 > 「**エンドポイント**」と右クリックしてメニューを開き、「**新規クライアント・エンドポイント**」を選択します。
2. 以下の表に示されているように、フィールドに入力します。

表 *75.* 新規クライアント・エンドポイント・フィールド

フィールド	**値**
名前	`Endpoint_WMQ`
Address	サービスにアクセスするために必要な *WMQ* 接続の詳細を記述する *SOAP/JMS URI*。 詳細は、『613 ページの『WCF 用の WebSphere MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』』を参照してください。
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract	サービス・コントラクト・インターフェースの名前

サービス・アプリケーションの場合、以下のようにして、サービス定義をサービス・セクションに追加します。

1. 「**サービス**」を右クリックしてメニューを開き、「**新規サービス**」を選択してから、ホスト対象のサービス・クラスを選択します。
2. エンドポイント定義を新規サービスの「**エンドポイント**」セクションに追加し、以下の表に示されているようにフィールドに入力します。

表 *76.* 新規サービス・エンドポイント・フィールド

フィールド	**値**
名前	`Endpoint_WMQ`
Address	サービスにアクセスするために必要な *WMQ* 接続の詳細を記述する *SOAP/JMS URI*。 詳細は、『613 ページの『WCF 用の WebSphere MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』』を参照してください。
Binding	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`

表 76. 新規サービス・エンドポイント・フィールド (続き)	
フィールド	**値**
Contract	サービス実装クラスの名前

バインディング情報とエンドポイント情報をプログラマチックに指定することにより、*WCF* カスタム・チャネルを作成する

WCF 用の WebSphere MQ カスタム・チャネルは、トランスポート・レベルの WCF チャネルです。 カスタム・チャネルを使用するには、エンドポイントおよびバインディングを定義する必要があります。これらは、アプリケーション・コードから直接プログラマチックに定義することができます。

WCF 用の WebSphere MQ カスタム・チャネル (トランスポート・レベルの WCF チャネル) を構成して使用するには、 バインディングおよびエンドポイント定義を定義する必要があります。 バインディング定義はチャネルの構成情報を保持し、エンドポイント定義は接続の詳細を保持します。 詳しくは、621 ページの『WCF サンプルの使用』を参照してください。

これらの定義は、以下の 2 つの方法で作成できます。

- アプリケーション構成ファイルに詳細を指定することにより、管理的に作成する。詳しくは、『609 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』』を参照してください。
- アプリケーション・コードからプログラマチックに直接作成する。この方法については、以下の例で説明します。

ステップ 1: チャネルのトランスポート・バインディング要素のインスタンスを作成する

以下のコードをご使用のアプリケーションに追加します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

ステップ 2: バインディング・プロパティーを設定する

例えば、次のコードをアプリケーションに追加して ClientConnectionMode を設定することによって、必要なバインディング・プロパティーを設定します。

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

ステップ 3: トランスポート・チャネルをメッセージ・エンコーダーと結合するカスタム・バインディングを作成する

次のコードをアプリケーションに追加することによって、カスタム・バインディングを作成します。

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

ステップ 4: SOAP/JMS URI を作成する

サービスにアクセスするために必要な WebSphere MQ 接続の詳細を記述する SOAP/JMS URI は、エンドポイント・アドレスとして指定する必要があります。 この方法は、チャネルがサービス・アプリケーションに使用されているのか、クライアント・アプリケーションに使用されているのかによって異なります。

クライアント・アプリケーションの場合は、以下のように、SOAP/JMS URI を EndpointAddress として作成する必要があります。

```
EndpointAddress address = new EndpointAddress("jms:/queue?
destination=SampleQ@QM1&connectionFactory=connectQueueManager(QM1)&initialContextFactory=com.ibm
.mq.jms.Nojndi");
```

サービス・アプリケーションの場合は、以下のように、SOAP/JMS URI を URI として作成する必要があります。

```
Uri address = new Uri("jms:/queue?
destination=SampleQ@QM1&connectionFactory=connectQueueManager(QM1)&initialContextFactory=com.ibm
.mq.jms.Nojndi");
```

エンドポイント・アドレスについて詳しくは、『613 ページの『WCF 用の WebSphere MQ カスタム・チャネルのエンドポイント URI アドレス・フォーマット』』を参照してください。

WCF 用の *WebSphere MQ* カスタム・チャネルのエンドポイント *URI* アドレス・フォーマット

Universal Resource Identifier (URI) は、Web サービスを指定するための場所および接続詳細を提供します。この URI フォーマットでは、ターゲット・サービスにアクセスする際に、SOAP/ WebSphere MQ 固有のパラメーターおよびオプションを包括的に制御できます。

Web サービスは Universal Resource Identifier (URI) を使用して指定されます。 このセクションでは、WebSphere MQ Transport for SOAP でサポートされる URI フォーマットを指定します。 この URI フォーマットによって、ターゲット・サービスにアクセスする際の SOAP/WebSphere MQ 固有のパラメーターおよびオプションを包括的に制御できます。 この形式は、WebSphere Application Server (WAS) および CICS と互換性があり、 WebSphere MQ とこれらの両方の製品との統合を容易にします。

URI 構文は以下のとおりです。

```
jms:/queue?name=value&name=value...
```

ここで、 name はパラメーター名、 *value* は適切な値です。 name=*value* エレメントは、2 番目以降のオカレンスの前にアンパーサンド (&) を付けて、任意の回数繰り返すことができます。

URI プロパティーの設定について詳しくは、Web サービス・デプロイメントに関する URI の構文とパラメーターを参照してください。

パラメーター名は、WebSphere MQ オブジェクトの名前と同様、大文字小文字を区別します。 いずれかのパラメーターを複数回指定した場合は、最後に出現した該当パラメーターが有効になります。つまり、クライアント・アプリケーションは、URI にパラメーター値を付加することにより、パラメーター値を指定変更できます。 認識されない追加のパラメーターが含まれている場合、それらは無視されます。

URI を XML ストリングで保管する場合、 アンパーサンド文字を "&" と記述しなければなりません。同様に、スクリプト内に URI がコーディングされている場合は、**&** などのエスケープ文字に注意してください。そうでない場合は、シェルによって解釈されます。

次に、Axis サービス用の単純な URI の例を示します。

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

以下に、.NET サービス用の簡単な URI の例を示します。

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

必須パラメーターのみが提供され (targetService は .NET サービスでのみ必須)、connectionFactory にはオプションが指定されていません。

この軸の例では、 connectionFactory にいくつかのオプションが含まれています。

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

この Axis の例では、 connectionFactory の sslPeerName オプションも指定されています。 sslPeerName 自体の値には、 名前と値のペア、および意味のある組み込みブランクが含まれます。

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

WCF バインディング構成オプション

このトピックでは、構成オプションをカスタム・チャネルのバインディング情報に適用する方法について説明し、 使用可能なオプションをリストします。

バインディング構成オプションは、以下の 2 つの方法のいずれかで設定できます。

1. 管理: バインディング・プロパティー設定を、アプリケーション構成ファイル (例えば、app.config) のカスタム・バインディング定義のトランスポート・セクションに指定する必要があります。
2. プログラマチック: カスタム・バインディングの初期化中にプロパティーを指定するように、 アプリケーション・コードを変更する必要があります。

バインディング・プロパティーを管理的に設定する

バインディング・プロパティー設定は、 アプリケーション構成ファイル (例えば、app.config) に指定することもできます。 構成ファイルは、以下のように **svcutil** により生成されます。

```
<customBinding>

...

     <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>

...

</customBinding>
```

バインディング・プロパティーをプログラマチックに設定する

クライアント接続モードを指定するために WCF バインディング・プロパティーを追加するには、カスタム・バインディングの初期化時にプロパティーを指定するようにサービス・コードを変更する必要があります。

以下の例を使用して、非管理のクライアント接続モードを指定します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

WCF バインディング・プロパティー

プロパティー名	クライアント・アプリケーションまたはサービス・アプリケーション	管理値	プログラムによる値	説明
maxBufferPoolSize	両方	0 から 64 ビットの符号付き整数	0 から 64 ビットの符号付き整数	チャネル・インスタンスの WCF メッセージ・バッファーの保管のために使用可能なメモリーの最大サイズを指定します。

プロパティー名	クライアント・アプリケーションまたはサービス・アプリケーション	管理値	プログラムによる値	説明
maxMessageSize	両方	1 から 32 ビットの符号付き整数	1 から 32 ビットの符号付き整数	個別の WCF メッセージに使用可能な最大メモリーを指定します。
clientConnectionMode	両方	0 (デフォルト値) 1	AS_URI (デフォルト値) CLIENT_UNMANAGED	トランスポート・チャネルのクライアント接続モードを指定します。 0 は、クライアント接続モードが URI に指定されたとおりであることを意味します。 クライアント接続が使用されている場合にのみ使用します。 クライアント接続モードが、URI に指定されたとおりであることを指定します。 クライアント接続モードが設定されていない場合、0 がデフォルト値です。 1 は、クライアント接続モードが非管理クライアントであることを意味します。 クライアント接続が使用されている場合にのみ使用します。
MaxConcurrentCalls	クライアント	範囲は 0 から 2 147 483 647 です。 16 がデフォルト値です。	範囲は 0 から 2 147 483 647 です。 16 がデフォルト値です。	このプロパティーは、個々のクライアント・プロキシーに対して同時に実行できる並行操作の最大数を定義します。 それよりも多くの操作が開始された場合、進行中の操作が完了するかタイムアウトになるまで操作はキューに入れられます。 この設定を使用して、個々のプロキシーが取り込むことができる スレッドおよびリソースの最大数を制御できます。 0 はこの制限を取り外し、 すべての操作を並行して試行できます。

プロパティー名	クライアント・アプリケーションまたはサービス・アプリケーション	管理値	プログラムによる値	説明
MaxConcurrentCalls	サービス	範囲は 1 から 2 147 483 647 です。 16 がデフォルト値です。	範囲は 1 から 2 147 483 647 です。 16 がデフォルト値です。	このプロパティーが使用されるのは、確約済みデリバリー・フィーチャーが使用可能である場合のみです (確約済みデリバリーについて詳しくは、605 ページの『WCF カスタム・チャネルの保証配信』を参照してください)。 これは、所定のエンドポイントに同時に進行できる並行操作の最大数を指定します。 この設定を変更するときには、注意が必要です。 それぞれの並行操作には追加リソースが必要とされ、特に、要求を実行するための、カスタム・チャネルの新規インスタンスおよびスレッド・プールからの関連スレッドが必要です。 過剰な割り振りは、生産性を低下させ、 パフォーマンスに重大な影響を及ぼすことがあります。 このプロパティーをサポートするには、スレッド・プールを適切に構成する必要があります。

WCF 用サービスの作成とホスティング

Windows Communication Foundation (WCF) サービスの作成および構成方法を説明する Microsoft WCF サービスの概要です。

WCF およびそれを使用する WCF サービス用の IBM WebSphere MQ カスタム・チャネルは、以下の方法によってホストすることができます。

- セルフ・ホスティング
- Windows サービス

WCF 用の IBM WebSphere MQ カスタム・チャネルは、Windows Process Activation Service にホストできません。

以下の各トピックには、必要なステップが示された簡単なセルフ・ホスティングの例が記載されています。詳細情報と最新情報が記載された Microsoft WCF オンライン文書は、Microsoft MSDN の Web サイト (https://msdn.microsoft.com) にあります。

1 番目の方法を使用した *WCF* サービス*･*アプリケーションの作成*:* アプリケーション構成ファイルを使用して管理的にセルフ・ホスティングする

アプリケーション構成ファイルを作成したら、サービスのインスタンスを開き、指定されたコードをアプリケーションに追加します。

始める前に

『609 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』』の説明に従って、サービスのアプリケーション構成ファイルを作成または編集します。

このタスクについて

1. サービス・ホストでサービスのインスタンスを生成し、インスタンスを開きます。 サービス・タイプは、サービス構成ファイルに指定されているサービス・タイプと同じでなければなりません。
2. 以下のコードをご使用のアプリケーションに追加します。

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

2 番目の方法を使用した *WCF* サービス・アプリケーションの作成*:* アプリケーションから直接プログラマチックにセルフ・ホスティングする

バインディング・プロパティーを追加し、必要なサービス・クラスのインスタンスを使用してサービス・ホストを作成し、そのサービスを開きます。

始める前に

1. カスタム・チャネル IBM.XMS.WCF.dll ファイルへの参照をプロジェクトに追加します。IBM.XMS.WCF.dll は *WMQInstallDir*\bin にあります。ここで、 *WMQInstallDir* は、 WebSphere MQ 7 がインストールされているディレクトリーです。
2. *using* ステートメントを IBM.XMS.WCF 名前空間に追加します。例えば、次のとおりです。using IBM.XMS.WCF
3. 『612 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』』の説明に従って、チャネル・バインディング要素のインスタンスとエンドポイントを作成します。

このタスクについて

チャネルのバインディング・プロパティーへの変更が必要な場合は、以下のステップを実行します。

1. 以下の例に示すように、バインディング・プロパティーを transportBindingElement に追加します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 必要なサービス・クラスのインスタンスを使用してサービス・ホストを作成します。

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. サービスを開きます。

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
    ...
service.Close();
```

HTTP エンドポイントを使用したメタデータの公開

WCF 用の WebSphere MQ カスタム・チャネルを使用するよう構成されたサービスのメタデータを公開するための手順です。

このタスクについて
サービス・メタデータを公開する必要がある場合 (例えば、svcutil などのツールが、オフラインの WSDL ファイルからではなく、実行中のサービスから直接サービス・メタデータにアクセスできるようにするため)、HTTP エンドポイントを使用してサービス・メタデータを公開する必要があります。 以下のステップを使用して、このエンドポイントを追加できます。

1. メタデータを ServiceHost に公開する必要のある基底アドレスを追加します。例えば、次のとおりです。

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. サービスが開かれる前に、以下のコードを ServiceHost に追加します。

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

タスクの結果
メタデータが以下のアドレスで使用可能になりました。http://localhost:8000/MyService

WCF 用のクライアント・アプリケーションの作成

Microsoft Windows Communication Foundation (WCF) クライアント・アプリケーションの生成および作成の概要です。

WCF サービス用のクライアント・アプリケーションを作成できます。 通常、クライアント・アプリケーションの生成は、Microsoft ServiceModel メタデータ・ユーティリティー・ツール (Svcutil.exe) を使用して、アプリケーションで直接使用できる必要な構成ファイルとプロキシー・ファイルを作成することにより行います。

svcutil ツールを使用して、実行中のサービスのメタデータにより *WCF* クライアント・プロキシーおよびアプリケーション構成ファイルを生成する
Microsoft svcutil.exe ツールを使用して、WCF 用の WebSphere MQ カスタム・チャネルを使用するよう構成されたサービスのクライアントを生成する手順です。

始める前に
svcutil ツールを使用して、アプリケーションで直接使用できる必要な構成およびプロキシー・ファイルを作成するには、 以下の 3 つの前提条件があります。

- svcutil ツールを開始する前に、WCF サービスが実行されている必要があります。
- WCF サービスは、実行中のサービスからクライアントを直接生成するために、 WebSphere MQ カスタム・チャネル・エンドポイント参照に加えて、HTTP ポートを使用してそのメタデータを公開する必要があります。
- カスタム・チャネルが svcutil の構成データに登録されている必要があります。

このタスクについて
以下のステップでは、WebSphere MQ カスタム・チャネルを使用するように構成されているが、 別個の HTTP ポートを介して実行時にそのメタデータの公開も行う、サービスのクライアントを生成する方法について説明します。

1. WCF サービスを開始します (サービスは、svcutil ツールを開始する前に実行されている必要があります)。
2. インストールのルートにある svcutil.exe 構成ファイルの詳細をアクティブな svcutil 構成ファイル (通常は C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config) に追加して、svcutil が WebSphere MQ カスタム・チャネルを認識できるようにします。
3. コマンド・プロンプトから svcutil を実行します。例えば、次のようになります。

```
svcutil /language:C# /r:<installlocation>\bin\IBM.XMS.WCF.dll
                /config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. 生成された app.config および YourService.cs ファイルを、Microsoft Visual Studio クライアント・プロジェクトにコピーします。

次のタスク

サービス・メタデータを直接取得できない場合は、svcutil を使用して、代わりに wsdl からクライアント・ファイルを生成できます。 詳しくは、『619 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』を参照してください。

svcutil ツールを使用して、*WSDL* による *WCF* クライアント・プロキシーおよびアプリケーション構成ファイルを生成する

サービスのメタデータを使用できない場合に、WSDL から WCF クライアントを生成するための手順です。

サービスのメタデータを直接取り出して、実行中のサービスのメタデータからクライアントを生成することができない場合は、svcutil を使用して、代わりに WSDL からクライアント・ファイルを生成できます。以下の変更を WSDL に加えて、WebSphere MQ カスタム・チャネルが使用されるように指定する必要があります。

1. 以下の名前空間定義およびポリシー情報を追加します。

```
<wsdl:definitions
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">

        <wsp:Policy wsu:Id="CustomBinding_IWMQSampleContract_policy">
            <wsp:ExtactlyOne>
                <wsp:All>
                    <xms:xms xmlns:xms="http://sample.schemas.ibm.com/policy/xms" />
                </wsp:All>
            </wsp:ExactlyOne>
        </wsp:Policy>

...

</wsdl:definitions>
```

2. 新規ポリシー・セクションを参照するようバインディング・セクションを変更し、基礎となるバインディング要素から transport 定義をすべて除去します。

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. コマンド・プロンプトから svcutil を実行します。例えば、次のようになります。

```
svcutil /language:C# /r:MQ_INSTALLATION_PATH\bin\IBM.XMS.WCF.dll
        /config:app.config
MQ_INSTALLATION_PATH\src\samples\WMQAxis\default\service\soap.server.stockQuoteAxis_Wmq.wsdl
```

ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。

アプリケーション構成ファイルによる、クライアント・プロキシーを使用する *WCF* クライアント・アプリケーションの作成

始める前に

『609 ページの『アプリケーション構成ファイルにバインディング情報とエンドポイント情報を指定することにより、WCF カスタム・チャネルを管理的に作成する』』の説明に従って、クライアント用のアプリケーション構成ファイルを作成または編集します。

このタスクについて

クライアント・プロキシーのインスタンスを生成して開きます。 生成したプロキシーに渡されるパラメーターは、クライアント構成ファイルに指定されたエンドポイント名 (Endpoint_WMQ など) と同じにする必要があります。

```
MyClientProxy myClient = new MyClientProxy("Endpoint_WMQ");
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (Exception e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
```

プログラマチック構成による、クライアント・プロキシーを使用する *WCF* クライアント・アプリケーションの作成

始める前に

1. カスタム・チャネル IBM.XMS.WCF.dll ファイルへの参照をプロジェクトに追加します。 IBM.XMS.WCF.dll は *WMQInstallDir*\bin ディレクトリーにあります。ここで、*WMQInstallDir* は、WebSphere MQ 7 がインストールされているディレクトリーです。
2. *using* ステートメントを IBM.XMS.WCF 名前空間に追加します。例えば、次のとおりです。using IBM.XMS.WCF
3. 612 ページの『バインディング情報とエンドポイント情報をプログラマチックに指定することにより、WCF カスタム・チャネルを作成する』の説明に従って、チャネルのバインディング要素とエンドポイントのインスタンスを作成します。

このタスクについて

チャネルのバインディング・プロパティーへの変更が必要な場合は、以下のステップを実行します。

1. 以下の図に示すように、バインディング・プロパティーを transportBindingElement に追加します。

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
EndpointAddress address =
  new EndpointAddress("jms:/queue?destination=SampleQ@QM1&connectionFactory=

connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
```

2. 以下の図に示すようにクライアント・プロキシーを作成します。ここで、*binding* および *endpoint address* は、ステップ 620 ページの『1』 で構成されたバインディングとエンドポイント・アドレスで、次のように渡されます。

```
 MyClientProxy myClient = new MyClientProxy(binding, endpoint address);
            try {
                   myClient.myMethod("HelloWorld!");
                   myClient.Close();
            }
            catch (TimeoutException e) {
                Console.Out.WriteLine(e);
                myClient.Abort();
            }
            catch (CommunicationException e) {
                Console.Out.WriteLine(e);
```

```
                            myClient.Abort();
        }
        catch (Exception e) {
            Console.Out.WriteLine(e);
            myClient.Abort();
        }
```

WCF サンプルの使用

Windows Communication Foundation (WCF) サンプルでは、WebSphere MQ カスタム・チャネルの使用方法の簡単な例をいくつか紹介します。

サンプル・プロジェクトを作成するには、Microsoft .NET 3.5 SDK または Microsoft Visual Studio 2008 のいずれかが必要です。

単純な片方向クライアント/サーバー WCF サンプル

このサンプルは、片方向チャネル形状を使用して WCF クライアントから Windows Communication Foundation (WCF) サービスを開始するために使用される WebSphere MQ カスタム・チャネルを示します。

このタスクについて

このサービスは、ストリングをコンソールに出力する単一メソッドを実装します。 クライアントは、『618 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』で説明しているように、別個に公開された HTTP エンドポイントからサービス・メタデータを取得する `svcutil` ツールを使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` ファイル内のクライアント・アプリケーション、およびサービス・アプリケーションの *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` ファイル内の対応する値も変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は IBM WebSphere MQ のインストール・ディレクトリーです。 JMS エンドポイント URI のフォーマットについて詳しくは、 WebSphere MQ 製品資料の「 *WebSphere MQ Transport for SOAP* 」を参照してください。 サンプル・ソリューションおよびソースを変更する必要がある場合は、IDE (例えば、Microsoft Visual Studio 8 以降など) が必要になります。

手順

1. *QM1*
2. *SampleQ*
3. リスナーがメッセージを待機するようにサービスを開始します。 *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は IBM WebSphere MQ のインストール・ディレクトリーです。
4. クライアントを 1 回実行します。 *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は IBM WebSphere MQ のインストール・ディレクトリーです。

   クライアント・アプリケーションは 5 回ループして、5 つのメッセージを *SampleQ* に送信します。

タスクの結果

サービス・アプリケーションは *SampleQ* からメッセージを取得し、画面上に `Hello World` を 5 回表示します。

次のタスク

単純な応答-要求クライアント/サーバー WCF サンプル

このサンプルは、要求/応答チャネル形状を使用して Windows Communication Foundation (WCF) クライアントから WCF サービスを開始するために使用される WebSphere MQ カスタム・チャネルの例を示しています。

このタスクについて

このサービスは、2 つの数値を加算および減算するいくつかの単純な計算器メソッドを提供し、結果を返します。 クライアントは、『618 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』で説明しているように、別個に公開された HTTP エンドポイントからサービス・メタデータを取得する `svcutil` ツールを使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\client\app.config` ファイル内のクライアント・アプリケーション、および *MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` ファイル内のサービス・アプリケーションでも対応する値を変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。 JMS エンドポイント URI のフォーマットについて詳しくは、WebSphere MQ 製品資料の「*WebSphere MQ Transport for SOAP*」を参照してください。 サンプル・ソリューションおよびソースを変更する必要がある場合は、IDE (例えば、Microsoft Visual Studio 8 以降など) が必要になります。

手順

1. *QM1*
2. *SampleQ*
3. *SampleReplyQ*
4. リスナーがメッセージを待機するようにサービスを開始します。*MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe` ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。
5. クライアントを 1 回実行します。*MQ_INSTALLATION_PATH*`\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe` ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。

タスクの結果

クライアントが実行されると、以下のプロセスが開始され、4 回繰り返されて、合計で 5 つのメッセージが各方向に送信されます。

1. クライアントは、*SampleQ* に要求メッセージを書き込み、応答を待ちます。
2. サービスは *SampleQ* から要求メッセージを取得します。
3. サービスは、メッセージの内容を使用して値を加算および減算します。
4. サービスはその結果を *SampleReplyQ* 上のメッセージに書き込んで、クライアントが新規メッセージを書き込むのを待ちます。
5. クライアントは *SampleReplyQ* からメッセージを取得し、結果を画面上に表示します。

次のタスク

WCF クライアントから WebSphere MQ でホストされている .NET サービスへの接続サンプル

サンプル・クライアント・アプリケーションとサンプル・サービス・プロキシー・アプリケーションは、.NET と Java の両方で提供されています。 これらのサンプルは、株価の照会要求を受け取って、その株価を提供する Stock Quote サービスに基づいています。

始める前に

サンプルでは、.NET SOAP over JMS サービスのホスティング環境が WebSphere MQ に正しくインストールされ、構成されていて、ローカル・キュー・マネージャーからアクセス可能であることが必要です。 環境のインストールおよび構成については、『961 ページの『WebSphere MQ Web transport for SOAP のインストール』』を参照してください。

.NET SOAP over JMS サービスのホスティング環境が WebSphere MQ に正しくインストールおよび構成され、ローカル・キュー・マネージャーからアクセス可能になったら、追加の構成ステップを実行する必要があります。

1. WMQSOAP_HOME 環境変数を WebSphere MQ インストール・ディレクトリーに設定します (例: `C:\Program Files\IBM\WebSphere MQ`)。
2. Java コンパイラー `javac` が使用可能であり、PATH 上にあることを確認してください。
3. ファイル `axis.jar` を、 WebSphere インストール CD の `prereqs/axis` ディレクトリーから WebSphere MQ 実動ディレクトリーにコピーします。例: `C:\Program Files\IBM\WebSphere MQ\java\lib\soap`
4. PATH: *MQ_INSTALLATION_PATH*`\Java\lib` に追加します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ がインストールされているディレクトリーを表します。例: `C:\Program Files\IBM\WebSphere MQ`
5. .NET の場所が *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` で正しく指定されていることを確認します。ここで、 *MQ_INSTALLATION_PATH* は、 WebSphere MQ がインストールされているディレクトリーを表します (例: `C:\Program Files\IBM\WebSphere MQ`)。 .NET の場所は、例えば `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin` のように指定できます。

前述のステップが完了したら、サービスをテストして実行します。

1. SOAP over JMS 作業ディレクトリーにナビゲートします。
2. 以下のいずれかのコマンドを入力して、検証テストを実行し、サービス・リスナーを実行状態にします。
   - .NET の場合: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold`。ここで *MQ_INSTALLATION_PATH* は、 WebSphere MQ がインストールされているディレクトリーを表します。
   - AXIS の場合:*MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold`。 *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされているディレクトリーを表します。

`hold` 引数を指定すると、テストの完了後にリスナーが実行されたままになります。

この構成中にエラーが報告される場合は、以下の方法ですべての変更を除去すれば、手順を再開することができます。

1. 生成された SOAP over JMS ディレクトリーを削除します。
2. キュー・マネージャーを削除してください。

このタスクについて

このサンプルでは片方向チャネル形状を使用して、WCF クライアントから、WebSphere MQ で提供されている .NET SOAP over JMS サンプル・サービスへの接続を示します。 このサービスは、テキスト・ストリングをコンソールに出力する単純な StockQuote の例を実装します。

クライアントは、619 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』で説明しているように、 クライアント・ファイルを生成するために WSDL を使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQNET\default\client\app.config ファイル内のクライアント・アプリケーション、およびサービス・アプリケーションの *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl ファイル内の対応する値も変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーを表します。 JMS エンドポイント URI のフォーマットについて詳しくは、 WebSphere MQ 製品資料の「 *WebSphere MQ Transport for SOAP* 」を参照してください。

手順

クライアントを 1 回実行します。*MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーを表します。

クライアント・アプリケーションは 5 回ループして、5 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得して、画面上に Hello World を 5 回表示します。

WCF クライアントから WebSphere MQ サンプルによってホストされる Axis Java サービスへ

Java と .NET の両方に対して、サンプル・クライアント・アプリケーションとサンプル・サービス・プロキシー・アプリケーションが提供されています。 これらのサンプルは、株価の照会要求を受け取って、その株価を提供する Stock Quote サービスに基づいています。

始める前に

このサンプルでは、.NET SOAP over JMS サービスのホスティング環境が WebSphere MQ に正しくインストールされ、構成されていて、ローカル・キュー・マネージャーからアクセス可能であることが必要です。環境のインストールおよび構成については、『961 ページの『WebSphere MQ Web transport for SOAP のインストール』』を参照してください。

.NET SOAP over JMS サービスのホスティング環境が WebSphere MQ に正しくインストールおよび構成され、ローカル・キュー・マネージャーからアクセス可能になったら、追加の構成ステップを実行する必要があります。

1. WMQSOAP_HOME 環境変数を WebSphere MQ インストール・ディレクトリーに設定します (例: C:\Program Files\IBM\WebSphere MQ)。
2. Java コンパイラー javac が使用可能であり、PATH 上にあることを確認してください。
3. ファイル axis.jar を WebSphere のインストール CD の prereqs/axis ディレクトリーから WebSphere MQ のインストール・ディレクトリーへコピーします。
4. PATH: *MQ_INSTALLATION_PATH*\Java\lib に追加します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ がインストールされているディレクトリーを表します。例: C:\Program Files\IBM\WebSphere MQ
5. .NET の場所が *MQ_INSTALLATION_PATH*\bin\amqwcallWSDL.cmd で正しく指定されていることを確認します。ここで、 *MQ_INSTALLATION_PATH* は、 WebSphere MQ がインストールされているディレクトリーを表します (例: C:\Program Files\IBM\WebSphere MQ)。 .NET の場所は、例えば set

msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin のように指定できます。

前述のステップが完了したら、サービスをテストして実行します。

1. SOAP over JMS 作業ディレクトリーにナビゲートします。
2. 以下のいずれかのコマンドを入力して、検証テストを実行し、サービス・リスナーを実行状態にします。
   - .NET の場合: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold。ここで *MQ_INSTALLATION_PATH* は、 WebSphere MQ がインストールされているディレクトリーを表します。
   - AXIS の場合:*MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold。 *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされているディレクトリーを表します。

hold 引数を指定すると、テストの完了後にリスナーが実行されたままになります。

この構成中にエラーが報告される場合は、以下の方法ですべての変更を除去すれば、手順を再開することができます。

1. 生成された SOAP over JMS ディレクトリーを削除します。
2. キュー・マネージャーを削除してください。

このタスクについて

このサンプルは、片方向チャネル形状を使用した、 WebSphere MQ で提供される WCF クライアントから Axis Java SOAP over JMS サンプル・サービスへの接続を示しています。 このサービスは、 現行ディレクトリーに保存されるファイルにテキスト・ストリングを出力する、単純な StockQuote の例を実装します。

クライアントは、619 ページの『svcutil ツールを使用して、WSDL による WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』で説明しているように、 クライアント・ファイルを生成するために WSDL を使用して生成されています。

このサンプルは、このパラグラフで説明しているように、特定のリソース名を使用して構成されています。リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\client\app.config ファイル内のクライアント・アプリケーション、およびサービス・アプリケーションの *MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl ファイル内の対応する値も変更する必要があります。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーを表します。

手順

クライアントを 1 回実行します。*MQ_INSTALLATION_PATH*\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーを表します。

クライアント・アプリケーションは 5 回ループして、5 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得して、現行ディレクトリーのファイルに Hello World を 5 回追加します。

関連資料

634 ページの『異なる SOAP 応答要素名の処理』

WCF では、戻り値の名前はデフォルトで特定のフォーマットであることを前提としていますが、サービスは、前提となっているフォーマットで名前の要素を返さないことがあります。

WebSphere Application Server によってホストされる WCF クライアントから Java サービスへのサンプル

WebSphere Application Server (WAS) 6 には、 サンプル・クライアント・アプリケーションおよびサンプル・サービス・プロキシー・アプリケーションが用意されています。 要求/応答サービスも用意されています。

始める前に

このサンプルでは、以下の WebSphere MQ 構成を使用する必要があります。

表 *77. WebSphere MQ* に必要な構成

オブジェクト	**必須名**
キュー・マネージャー	QM1
ローカル・キュー	HelloWorld
ローカル・キュー	HelloWorldReply

このサンプルでは、 WebSphere Application Server V6 がホスティングする環境を正しくインストールし、構成している必要もあります。 WebSphere Application Server V6 は、デフォルトでバインディング・モード接続を使用して WebSphere MQ に接続します。 そのため、 WebSphere Application Server V6 は、キュー・マネージャーと同じマシンにインストールする必要があります。

WAS 環境を構成したら、以下の追加構成ステップを行う必要があります。

1. WebSphere Application Server JNDI リポジトリーに以下の JNDI オブジェクトを作成します。
   a. `HelloWorld` という JMS キュー宛先
      - JNDI 名を `jms/HelloWorld` に設定します。
      - キュー名を `HelloWorld` に設定します。
   b. `HelloWorldQCF` という JMS キュー接続ファクトリー
      - JNDI 名を `jms/HelloWorldQCF` に設定します。
      - キュー・マネージャー名を `QM1` に設定します。
   c. `WebServicesReplyQCF` という JMS キュー接続ファクトリー
      - JNDI 名を `jms/WebServicesReplyQCF` に設定します。
      - キュー・マネージャー名を `QM1` に設定します。
2. 次の構成で `HelloWorldPort` というメッセージ・リスナー・ポートを WebSphere Application Server に作成します。
   - 接続ファクトリーの JNDI 名を `jms/HelloWorldQCF` に設定します。
   - 宛先の JNDI 名を `jms/HelloWorld` に設定します。
3. 以下のようにして、ご使用の WebSphere Application Server に `HelloWorldEJBEAR.ear` Web サービス・アプリケーションをインストールします。
   a.「**アプリケーション**」 > 「**新規アプリケーション**」 > 「**新規エンタープライズ・アプリケーション**」をクリックします。
   b. *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` にナビゲートします。ここで、*MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。
   c. ウィザードのデフォルト・オプションを何も変更せずに、 アプリケーションをインストールした後でアプリケーション・サーバーを再始動してください。

WAS の構成が完了したら、以下のようにサービスを一度実行してテストします。

1. Soap over JMS 作業ディレクトリーにナビゲートします。
2. コマンド *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\TestClient.exe を入力してサンプルを実行します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。

このタスクについて

このサンプルでは、要求/応答チャネル形状を使用して、WCF クライアントから、(WebSphere MQ V7 に含まれる WCF サンプルで提供される) WebSphere Application Server SOAP over JMS サンプル・サービスへの接続を実演します。 メッセージ・フローは、WCF と WebSphere Application Server の間を、WebSphere MQ キューを使用して送信されます。 サービスは HelloWorld(...) メソッドを実装します。このメソッドはストリングを受け取り、クライアントにグリーティングを返します。

クライアントは、『618 ページの『svcutil ツールを使用して、実行中のサービスのメタデータにより WCF クライアント・プロキシーおよびアプリケーション構成ファイルを生成する』』で説明しているように、別個に公開された HTTP エンドポイントからサービス・メタデータを取得する svcutil ツールを使用して生成されています。

このサンプルは、以下の手順で説明するように、特定のリソース名を使用して構成されています。 リソース名を変更する必要がある場合は、*MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\default\client\app.config ファイル内のクライアント・アプリケーション、および *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear 内のサービス・アプリケーションで、対応する値も変更する必要があります。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。 JMS エンドポイント URI のフォーマットについて詳しくは、 Web サービス・デプロイメントの URI 構文とパラメーター を参照してください。

サービスとクライアントは、IBM Developer の記事「 *Building a JMS Web service using SOAP over JMS and WebSphere Studio*」に概説されているサービスとクライアントに基づいています。 WebSphere MQ WCF カスタム・チャネルと互換性のある SOAP over JMS Web サービスの開発について詳しくは、関連記事 (https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html) を参照してください。

手順

クライアントを 1 回実行します。*MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe ファイルを実行します。ここで、 *MQ_INSTALLATION_PATH* は WebSphere MQ のインストール・ディレクトリーです。

クライアント・アプリケーションは、両方のサービス・メソッドを同時に開始して、 2 つのメッセージをサンプル・キューに送信します。

タスクの結果

サービス・アプリケーションは、サンプル・キューからメッセージを取得して、 クライアント・アプリケーションがコンソールに出力する応答を HelloWorld(...) メソッド呼び出しに提供します。

WebSphere MQ 用の WCF カスタム・チャネルの問題判別

WebSphere MQ トレースを使用すると、WebSphere MQ のコードのさまざまな部分で実行されている内容について詳細な情報を収集することができます。 Windows Communication Foundation (WCF) を使用する場合、Microsoft WCF インフラストラクチャー・トレースに統合された WCF カスタム・チャネル・トレースに対して、別個のトレース出力が生成されます。

WCF カスタム・チャネルのトレースを完全に使用可能にすると、以下の 2 つの出力ファイルが生成されます。

1. Microsoft WCF インフラストラクチャー・トレースと統合された WCF カスタム・チャネル・トレース。
2. XMS .NET と統合された WCF カスタム・チャネル・トレース。

2 つのトレース出力を用意することにより、以下のように、各インターフェースで該当するツールを使用して問題を追跡できます。

- 適切な Microsoft ツールを使用した、WCF 問題の判別。
- XMS トレース・フォーマットを使用した、WebSphere MQ MQI クライアント問題の判別。

トレースの使用可能化を簡素化するために、.NET 3 TraceSource および XMS .NET トレース・スタックは両方とも、単一のインターフェースを使用して制御されます (『628 ページの『WCF トレース構成およびトレース・ファイル名』』を参照)

WCF カスタム・チャネルの例外階層

カスタム・チャネルからスローされる例外タイプは、WCF と整合性があり、一般的には TimeoutException または CommunicationException (または、CommunicationException のサブクラス) です。

エラー条件の詳細は、使用可能な場合は、リンク例外または内部例外を使用して入手できます。 次の例外は標準的な例外で、チャネルのアーキテクチャー内の層ごとに、追加のリンク例外が提供されます。例えば、CommunicationsException には、リンク例外 XMSException があり、このリンク例外には、リンク例外 MQException があります。

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

キー情報は、階層内最上位の CommunicationException のデータ収集で取り込まれ、提供されます。 このようにしてデータを取り込み、提供すると、アプリケーションは、リンクされている例外、およびリンクされている例外に含まれている可能性がある追加情報を照会するためにチャネルのアーキテクチャー内の各層にリンクする必要がなくなります。 以下のキー名が定義されています。

- IBM.XMS.WCF.ErrorCode: 現行のカスタム・チャネル例外のエラー・メッセージ・コード。
- IBM.XMS.ErrorCode: スタック内の最初の XMS 例外のエラー・メッセージ。
- IBM.WMQ.ReasonCode: 基礎になっている WebSphere MQ 理由コード。
- IBM.WMQ.CompletionCode: 基礎になっている WebSphere MQ 完了コード。

WCF トレース構成およびトレース・ファイル名

トレースが完全に使用可能になっている場合は、2 つの出力ファイルが生成されます。1 つは、WCF 問題の診断用で、もう 1 つは内部トレース診断資料の詳細ファイルです。 トレースの使用可能化を簡素化するために、.NET 3 TraceSource と XMS .NET の両方のトレース・スタックは単一のインターフェースを使用します。

WCF カスタム・チャネルでは、2 つの異なるトレース方式が使用可能です。この 2 つのトレース方式は、単独でアクティブ化することも、一緒にアクティブ化することもできます。 それぞれの方式で独自のトレース・ファイルが生成されるので、両方のトレース方式をアクティブ化すると、2 つのトレース出力ファイルが生成されます。

構成および使用可能化をできるだけ簡素にしておくために、両方のトレース方式の制御に同一のインターフェースが使用されています。 以下のセクションで説明されているように、`app.config` ファイルを編集して、関連するトレース構成を組み込む必要があります。 これによりユーザーは、自分自身の同等のセクションを追加して、出力を自分のアプリケーションからのトレースと組み合わせることができます。

WCF カスタム・チャネル・トレースは、デフォルトでは使用可能になっていません。 まず、トレース・リスナーを作成してから、次に、選択したトレース・ソースに必要なトレース・レベルを `app.config` ファイルに設定する必要があります。

WCF インフラストラクチャー・トレースを使用した WCF カスタム・チャネルの構成

以下のコードのセクションを、app.config ファイルの <system.diagnostics><sources> セクションに追加します。

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

前のコード断片では、.NET 3 の TraceSource を使用してチャネル・トレースが作成されます。 実行可能ファイルに関連付けられた構成ファイルの呼び出しはすべて、このコード断片により制御されます。

XMS .NET トレースを使用した WCF カスタム・チャネルの構成

XMS .NET トレースを構成するには、 app.config ファイルの <system.diagnostics><sources> セクションにコードのセクションを追加する必要があります。 ただし、コードの断片は、『WCF インフラストラクチャー・トレースを使用した WCF カスタム・チャネルの構成』セクションの拡張可能な <source> 要素に追加されます。 そのため、WCF インフラストラクチャー・トレース・コードは XMS .NET トレースが機能するために必要になりますが、WCF インフラストラクチャー・トレースは、必要ない場合には、使用不可に設定することができます (『WCF トレースの使用可能化』セクションを参照)。

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

WCF トレース構成変数

表 *78. WCF* トレース構成変数

変数	説明
名前	名前を IBM.XMS.WCF という形式で指定します。
switchValue	switchValue はトレース・レベルを制御します。 switchValue が Off に設定されている場合、WCF インフラストラクチャー TraceSource は生成されません。 他の値 (Verbose など) はすべて、TraceSource を生成します。 Microsoft からの詳細なトレース・レベル情報については、WCF 資料を参照するか、 Microsoft WCF Tracing Web ページ (https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx) にアクセスしてください。

表 *78. WCF* トレース構成変数 *(*続き*)*

変数	説明

xmsTraceSpecification=*ComponentName*=*type*=*state*	*ComponentName* は、トレースするクラスの名前です。この名前には、ワイルドカード文字 * を使用できます。以下に例を示します。
	`*=all=enabled`
	これは、すべてのクラスをトレースすることを指定します。
	`IBM.XMS.impl.*=all=enabled`
	これは、API トレースのみが必要であることを指定します。
	type は、以下のトレース・タイプのいずれかに指定できます。
	• all
	• debug
	• イベント
	• EntryExit
	state は、「enabled」または「disabled」のいずれかに指定できます。

xmsTraceFilePath="*filename*"	xmsTraceFilePath を指定しない場合、または xmsTraceFilePath は存在するが、空ストリングを含んでいる場合は、トレース・ファイルは現行ディレクトリーに配置されます。 指定されたディレクトリーにトレース・ファイルを保管するには、xmsTraceFilePath にディレクトリー名を指定します。例えば、次のようになります。 `xmsTraceFilePath="c:\somepath"`
xmsTraceFileSize="*size*"	トレース・ファイルの最大許容サイズ。 ファイルは、このサイズに達すると、アーカイブされて名前変更されます。 デフォルトの最大サイズは 20 KB で、以下のように指定されています。 `xmsTraceFileSize="20000000".`
xmsTraceFileNumber="*number*"	保存対象のトレース・ファイルの数。 デフォルトは 4 です (アクティブ・ファイルが 1 つと、アーカイブ・ファイルが 3 つ)。 最小許容数は 2 です。

表 78. WCF トレース構成変数 (続き)	
変数	**説明**

xmsTraceFormat="*format*"	xmsTraceFormat には、basic および advanced の 2 つのレベルがあります。 xmsTraceFormat を指定しない場合、または xmsTraceFormat は存在するが、空ストリングを含んでいる場合、デフォルトのトレース・フォーマットは「basic」です。 以下のように指定すると、トレース・ファイルはこのフォーマットで生成されます。
	`xmsTraceFormat="basic"`
	トレース・アナライザー・ツールと互換性のあるトレースが必要な場合は、以下のように指定する必要があります。
	`traceFormat="advanced"`

WCF トレースの使用可能化

この 2 種類のトレース方式を有効/無効にする操作には、4 つの組み合わせがあります。 これらの 4 つの組み合わせを使用するには、前のセクションで説明したコードの各セクションの値を編集する必要があります。

環境変数を設定することもできます。詳しくは、632 ページの『WCF_TRACE_ON 環境変数による WCF トレースの使用可能化』を参照してください。

次の表およびそれに示されている値は、app.config ファイルに既に追加されている、前に示したコード断片によって異なります。

表 79. WCF トレースの使用可能化の組み合わせ		
トレース・タイプ	**変更された値**	**例**
XMS トレース使用可能。 WCF TraceSource 使用可能。	switchValue が Off に設定されていない	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
XMS トレース使用可能。 WCF TraceSource 使用不可。	switchValue が Off に設定され、xmsTraceSpecification が指定されている	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

表 79. *WCF* トレースの使用可能化の組み合わせ (続き)

トレース・タイプ	変更された値	例
XMS トレース使用不可。WCF TraceSource 使用可能。	このような状態となるためには、以下の 2 つの方法があります。 • switchValue 変数が Off に設定されておらず、xmsTraceSpecification が追加されていない • switchValue 変数が Off に設定されておらず、xmsTraceSpecification が disabled に設定されている	<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>
XMS トレース使用不可。WCF TraceSource 使用不可。	このような状態となるためには、以下の 3 つの方法があります。 • app.config ファイルに <source> 要素を指定しない • switchValue 変数が Off に設定され、xmsTraceSpecification が追加されていない • switchValue 変数が Off に設定され、xmsTraceSpecification が disabled に設定されている	<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>

WCF_TRACE_ON 環境変数による WCF トレースの使用可能化

前述の WCF トレースを使用可能にする方法と同様に、XMS .NET トレースも、WCF_TRACE_ON 環境変数を使用して使用可能にすることができます。

WCF_TRACE_ON 環境変数をヌル以外の値に設定することは、xmstraceSpecification を *=all=enabled に設定することと等価です (例えば、「set WCF_TRACE_ON=true」となります)。

ただし、xmstraceSpecification が app.config ファイルで明示的に設定されている場合は、WCF_TRACE_ON 環境変数は指定変更されます。

WCF トレース出力ファイルおよびファイル名

XMS トレース・ファイルの名前には、従来より、ベース名とプロセス ID で構成される xms_trace_*pid*.log というフォーマットが使用されています (ここで、*pid* はプロセス ID です)。

XMS トレース・ファイルは、WCF カスタム・チャネル・トレース・ファイルと同時に生成することもできるため、XMS .NET トレース出力ファイルと統合された WCF カスタム・チャネル・トレース・ファイルには、混乱を避けるために、wcfxms_trace_*pid*.log (*pid* はプロセス ID) というフォーマットが使用されています。

トレース出力ファイルは、デフォルトでは現行作業ディレクトリーに作成されますが、この宛先は、必要に応じて再定義することができます。

WCF XMS First Failure Support Technology (FFST)

WebSphere MQ トレースを使用することにより、WebSphere MQ のコードのさまざまな部分で実行されている内容について詳細な情報を収集することができます。 XMS FFST には、WCF カスタム・チャネル用の独自の構成ファイルと出力ファイルがあります。

XMS FFST トレース・ファイルの名前には、以前からベース名とプロセス ID が使用され、xmsffdc*pid_date*.txt (*pid* はプロセス ID、*date* は日時) というフォーマットになっています。

XMS FFST トレース・ファイルは、WCF カスタム・チャネル XMS FFST ファイルと同時に生成することもできるため、WCF カスタム・チャネル XMS FFST 出力ファイルには、混乱を避けるために、以下のフォーマットが使用されます。wcfffdc*pid_date*.txt (*pid* はプロセス ID、*date* は日時)

このトレース出力ファイルは、デフォルトでは、現行作業ディレクトリーに作成されますが、この宛先は、必要に応じて再定義することができます。

XMS .NET トレース・ヘッダーを含む WCF カスタム・チャネルは、以下の例のようになります。

```
************* Start Display XMS WCF Environment *************
  Product Name  :- value
  WCF Version   :- value
  Level    :- value
*************  End Display XMS WCF Environment  *************
```

FFST トレース・ファイルは、標準的な方法でフォーマットされ、カスタム・チャネルに固有のフォーマットは使用されません。

WCF のバージョン情報

WCF のバージョン情報は問題判別に役立ち、カスタム・チャネルのアセンブリー・メタデータに含まれています。

WCF 用の WebSphere MQ カスタム・チャネルのバージョンのメタデータは、次の 3 つのいずれかの方法で取得できます。

- WebSphere MQ ユーティリティー dspmqver を使用します。 dspmqver の使用法については、dspmqver を参照してください。
- 「Windows」 エクスプローラー・プロパティー・ダイアログを使用する: Windows エクスプローラーで、**IBM.XMS.WCF.dll** > 「**プロパティー**」 > 「**バージョン**」。
- チャネルの FFST ファイルまたはトレース・ファイルのヘッダー情報を使用します。 FFST ヘッダー情報について詳しくは、632 ページの『WCF XMS First Failure Support Technology (FFST)』を参照してください。

WCF のヒント

次のヒントは、重要な順に並べられたものではなく、資料の新しいバージョンがリリースされたときに追加されたものと考えてください。 これらのヒントが、行っている作業に関連しているものである場合は、参照すると時間の節約に役立つことがあります。

WCF サービス・ホストからの例外の外部化

WCF サービス・ホストを使用してホストされているサービスの場合、サービス、WCF 内部、およびチャネル・スタックからスローされた未処理例外は、デフォルトでは外部化されません。 これらの例外に関する通知を受け取るには、エラー・ハンドラーを登録する必要があります。

次のコードに、サービスの属性として適用できるエラー・ハンドラー・サービス動作を定義する例を示します。

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, Collection<ServiceEndpoint> endpoints,
           BindingParameterCollection bindingParameters)
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
```

```
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        //
        // IErrorHandler Interface
        //
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
        }
    }
```

異なる *SOAP* 応答要素名の処理

WCF では、戻り値の名前はデフォルトで特定のフォーマットであることを前提としていますが、サービスは、前提となっているフォーマットで名前の要素を返さないことがあります。

WCF には、戻り値の名前は *methodName*Result というフォーマットであると見なす規則があります (ここで、*methodName* は、サービス操作の名前です)。 例えば、サービスの名前が getQuote である場合、WCF は、getQuoteResult という名前の応答が返されると見なします。

しかし、サービスは、このフォーマットに従わない名前の要素を返すことがあります。

scvutil ツールを実行してプロキシー・クライアントを生成するときに、WSDL が別の名前を指定すると、プロキシー・インターフェースはパラメーターを追加して、探す名前を WCF に指示します。 以下に例を示します。

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style =
System.ServiceModel.OperationFormatStyle.Rpc,
                                                  Use =
System.ServiceModel.OperationFormatUse.Encoded)]
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

独自のインターフェースを作成する場合 (例えば、既存のプロキシー・インターフェースの要求応答メソッドを追加する場合) は、サービスが別の名前を返したときにインターフェースに同じパラメーターを追加するようにしておく必要があります。 これを行わないと、サービス・メソッドの呼び出しで常にヌル値が返される、というよくある問題が発生します。オブジェクトが返されるとメソッドはヌルを返しますが、整数などの数値が返されると、サービス・メソッドは 0 を返します。

C++ の使用

WebSphere MQ では、WebSphere MQ オブジェクトと同等の C++ クラス、および配列データ型と同等のいくつかの追加クラスを提供します。 MQI を介して使用できない機能がいくつか提供されます。

WebSphere MQ バージョン 7.0 の時点では、WebSphere MQ プログラミング・インターフェースに対する機能拡張は、C++ クラスに適用されません。

WebSphere MQ C++ は、以下の機能を提供します。

- WebSphere MQ データ構造体の自動初期設定。
- ジャストインタイムのキュー・マネージャーの接続およびキューのオープン。

- 暗黙のキューの閉止およびキュー・マネージャーの切断。
- 送達不能ヘッダーの伝送と受信。
- IMS ブリッジ・ヘッダーの伝送と受信。
- 参照メッセージ・ヘッダーの伝送と受信。
- トリガー・メッセージの受信。
- CICS ブリッジ・ヘッダーの伝送と受信。
- 作業ヘッダーの伝送と受信。
- クライアント・チャネル定義。

次に示す Booch のクラス・ダイアグラムを見てください。 これらの図では、どのクラスも、ハンドルかデータ構造体のどちらかをもつ手続き型 MQI (例えば C の使用) の WebSphere MQ エンティティーとほぼ対応しています。 すべてのクラスは、ImqError (ImqError C++ クラスを参照) クラスの性質を継承しているため、エラー状態とオブジェクトを個別に関連付けることができます。

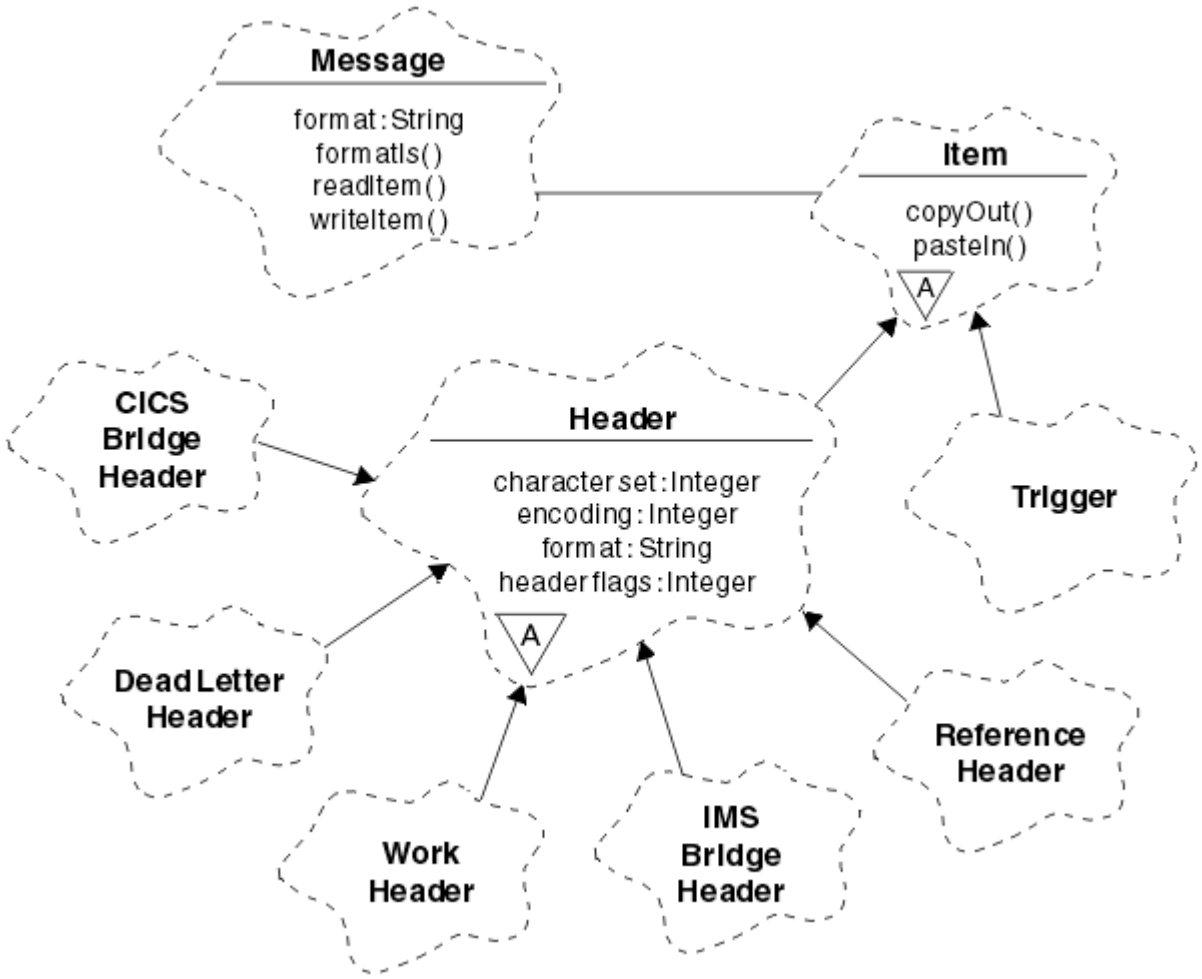


図 *120. WebSphere MQ C++* クラス *(*項目ハンドル*)*

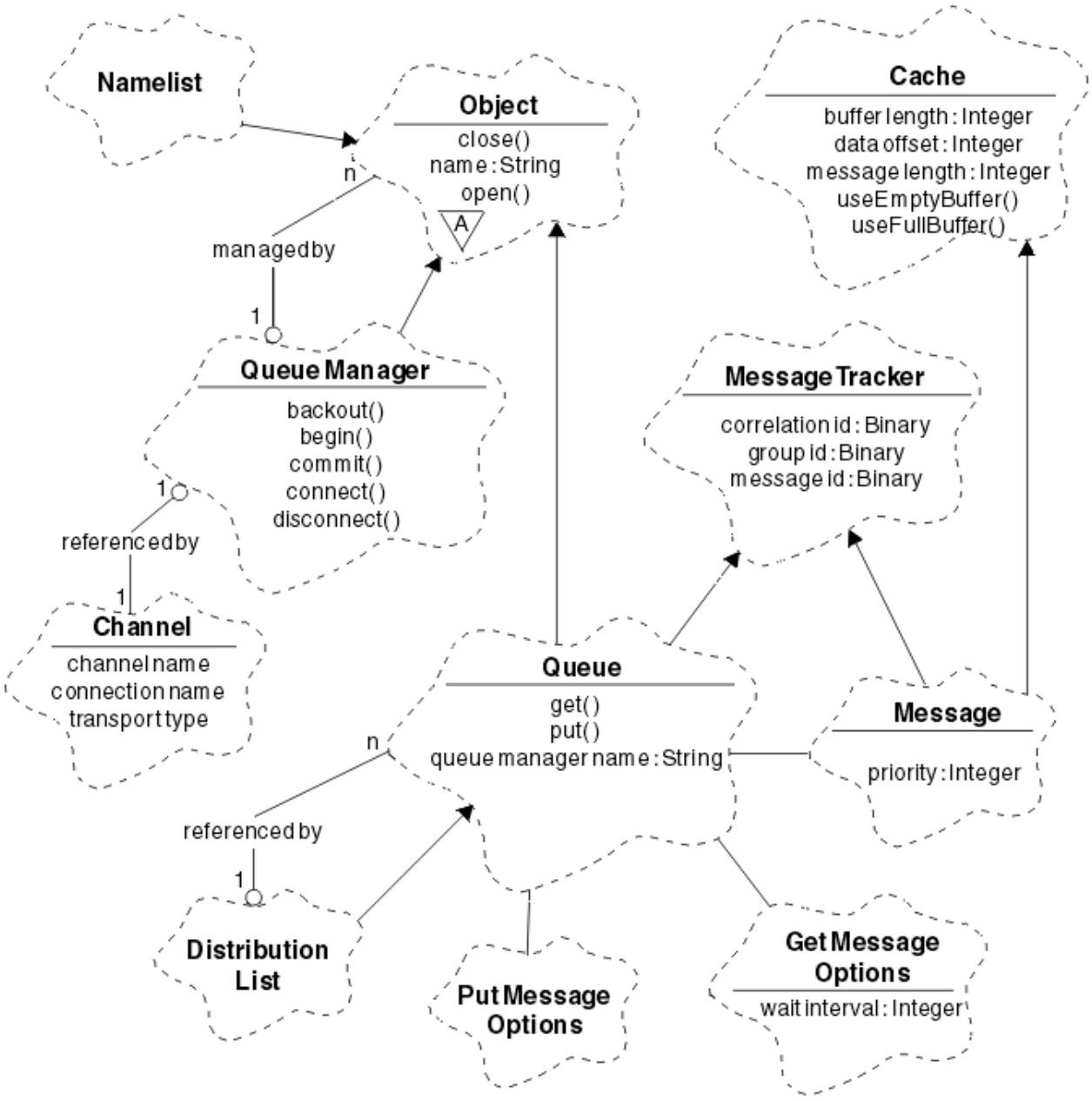


図 *121. WebSphere MQ C++* クラス *(*キュー管理*)*

Booch のクラス・ダイアグラムを正しく解釈するために、以下の表記規則に留意してください。

- クラス 名の下にはメソッドと重要な属性を示します。
- 抽象クラス は、雲型の囲みの中の小さな三角形で示します。
- 継承関係 は、親クラスの方向を指す矢印で示します。
- 2 つのクラスの間に協調関係 がある場合は、雲型の囲みの間を数字や文字の付いていない線でつないであります。
- 2 つのクラスの間に参照関係 がある場合は、雲型の囲みの間を数字付きの線でつないであります。 この数字は、特定の関係に同時に関与できるオブジェクトの数を表します。

以下のクラスおよびデータ・タイプは、 キュー管理クラス (636 ページの図 121 を参照) および 項目ハンドル・クラス (635 ページの図 120 を参照) の C++ メソッドシグニチャーで使用されます。

- MQBYTE24 などのバイト・アレイをカプセル化する ImqBinary クラス (ImqBinary C++ クラスを参照)。
- **typedef unsigned char ImqBoolean** として定義されている ImqBoolean データ・タイプ。

- MQCHAR64 などの文字アレイをカプセル化する ImqString クラス (ImqString C++ クラスを参照)。

データ構造体を持つエンティティーは、適切なオブジェクト・クラス内に包含されます。 個々のデータ構造体フィールド (C++ と MQI の相互参照を参照) は、メソッドでアクセスされます。

ハンドルを持つエンティティーは、ImqObject クラス階層 (ImqObject C++ クラスを参照) の下にあり、MQI へのカプセル化されたインターフェースを提供します。 これらのクラスのオブジェクトは、手続き型 MQI に関連して必要なメソッド呼び出し回数を減らすことのできる知的機能を持っています。 例えば、必要に応じてキュー・マネージャー接続を確立したり、廃棄したりでき、適切なオプションを使用してキューをオープンしてからクローズすることも可能です。

ImqMessage クラス (ImqMessage C++ クラスを参照) は、MQMD データ構造体をカプセル化します。また、キャッシュ式バッファー機能を提供することによって、ユーザー・データおよび項目 (646 ページの『C++ によるメッセージの読み取り』を参照) 用の保存場所としても利用できます。 ユーザー・データ用に固定長バッファーを提供し、そのバッファーを何回も使用することができます。 バッファー内に存在するデータの量は、使用するたびに変わる可能性があります。 別の方法として、システムが可変長のバッファーを提供して、管理することができます。 この場合には、バッファーのサイズ (メッセージの受信に使用できる量) と実際に使用される量 (伝送のためのバイト数または実際に受信されるバイト数のいずれか) の両方を慎重に考慮する必要があります。

関連概念

技術概要

637 ページの『C++ サンプル・プログラム』
メッセージの取得と書き込みのデモ用に 4 つのサンプル・プログラムが提供されています。

641 ページの『C++ 言語に関する考慮事項』
このトピック・コレクションでは、Message Queue Interface (MQI) を使用するアプリケーション・プログラムを作成する際に考慮しなければならない C++ 言語の使用法および規則について詳述します。

645 ページの『C++ によるメッセージ・データの作成』
メッセージ・データは、システムまたはアプリケーションが提供できるバッファー内に作成されます。 いずれの方法にも利点がいくつかあります。 バッファーの使用例がいくつか提供されています。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

7 ページの『アプリケーションの開発』
IBM WebSphere MQ には、ビジネス・プロセスをサポートするために必要なメッセージを送受信するアプリケーションを開発するためのいくつかの手段が用意されています。 キュー・マネージャーや関連リソースを管理するためのアプリケーションを開発することもできます。

関連資料

652 ページの『WebSphere MQ C++ プログラムの作成』
サポートされるコンパイラーの URL が、WebSphere MQ プラットフォーム上で C++ プログラムおよびサンプルをコンパイル、リンク、および実行するために使用するコマンドとともにリストされます。

C++ と MQI の相互参照

WebSphere MQ C++ クラス

C++ サンプル・プログラム

メッセージの取得と書き込みのデモ用に 4 つのサンプル・プログラムが提供されています。

サンプル・プログラムには次のものがあります。

- HELLO WORLD (imqwrld.cpp)
- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

サンプル・プログラムは、638 ページの表 80 に示されているディレクトリーの中にあります。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

表 *80.* サンプル・プログラムの位置

環境	**ソースが入ったディレクトリー**	**ビルドを含むディレクトリー プログラム**
AIX	*MQ_INSTALLATION_PATH*/サンプ	*MQ_INSTALLATION_PATH*/samp/bin/ia
HP-UX	*MQ_INSTALLATION_PATH*/サンプ	*MQ_INSTALLATION_PATH*/ samp/bin/ah (注 638 ページの『2』 を参照)
Solaris	*MQ_INSTALLATION_PATH*/サンプ	*MQ_INSTALLATION_PATH*/samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/サンプ	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATION_PATH*¥tools¥cplus¥samples	*MQ_INSTALLATION_PATH*¥tools¥cplus¥ samples¥bin¥vn (注 638 ページの『3』 を参照)

注 :

1. ILE C++ コンパイラー (IBM i 用) を 使用して作成されたプログラムは、ライブラリー QMQM に入っています。 組み込みファイルは、/QIBM/ProdData/mqm/inc 内にあります。
2. HP ANSI C++ コンパイラーを使用して構築されたプログラムは、 ディレクトリー *MQ_INSTALLATION_PATH*/samp/bin/ah 内にあります。 詳しくは、653 ページの『HP-UX における C++ プログラムの作成』を参照してください。
3. Microsoft Visual Studio を使用して作成されたプログラムは、*MQ_INSTALLATION_PATH*¥tools¥cplus¥samples¥bin¥vn 内にあります。 これらのコンパイラーの詳細については、658 ページの『Windows における C++ プログラムの作成』を参照してください。

サンプル・プログラム HELLO WORLD (imqwrld.cpp)

この C++ サンプル・プログラムは、ImqMessage クラスを使用して通常のデータグラム (C 構造体) を読み書きする方法を示します。

このプログラムは、ImqMessage クラスを使用して通常のデータグラム (C 構造体) を読み書きする方法を指示します。 このサンプルでは、メソッド呼び出しをほとんど採用していませんが、**open**、**close**、**disconnect** といった暗黙のメソッド呼び出しを利用します。

z/OS 以外のすべてのプラットフォームの場合

WebSphere MQ へのサーバー接続を使用している場合には、以下のいずれかの手順を実行します。

- 既存のデフォルト・キュー SYSTEM.DEFAULT.LOCAL.QUEUE を使用する場合は、パラメーターを引き渡さずに、プログラム **imqwrlds** を実行します。
- 動的に割り当てられた一時的なキューを使用する場合は、デフォルトのモデル・キューの名前 SYSTEM.DEFAULT.MODEL.QUEUE を引き渡して **imqwrlds** を実行します。

WebSphere MQ へのクライアント接続を使用している場合には、以下のいずれかの手順を実行します。

- MQSERVER 環境変数をセットアップし (詳しくは、MQSERVER を参照)、**imqwrldc** を実行します。

- **queue-name**、**queue-manager-name**、および **channel-definition** をパラメーターとして引き渡して、**imqwrldc** を実行します。 ここで、**channel-definition** は通常、SYSTEM.DEF.SVRCONN/TCP/*hostname*(1414) です。

サンプル・コード

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // WebSphere MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
```

```
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

サンプル・プログラム SPUT (imqsput.cpp) および SGET (imqsget.cpp)

これらの C++ プログラムは、指定されたキューにメッセージを書き込み、指定されたキューからメッセージを取り出します。

これらのサンプルは、以下のクラスの使用を示したものです。

- ImqError (ImqError C++ クラスを参照)
- ImqMessage (ImqMessage C++ クラス) を参照
- ImqObject (ImqObject C++ クラス) を参照
- ImqQueue (ImqQueue C++ クラス) を参照
- ImqQueueManager (ImqQueueManager C++ クラスを参照)

プログラムを実行するには、該当する指示に従います。

z/OS 以外のすべてのプラットフォームの場合

1. **imqsputs** *queue-name* を実行します。
2. コンソールでテキスト行を入力します。 これらの行は、指定されたキューにメッセージとして書き込まれます。
3. 入力データを終了するためにヌル行を 1 行入力します。
4. すべての行を取り出してコンソールで表示するために、**imqsgets** *queue-name* を実行します。

サンプル・プログラム DPUT (imqdput.cpp)

この C++ のサンプル・プログラムは、2 つのキューで構成された配布リストにメッセージを書き込みます。

DPUT は、ImqDistributionList クラス (ImqDistributionList C++ クラスを参照) の使用を示しています。 このサンプルは、z/OS ではサポートされていません。

1. 名前を指定された 2 つのキューにメッセージを配置するために、**imqdputs** *queue-name-1 queue-name-2* を実行します。
2. それらのキューからメッセージを取り出すために、**imqsgets** *queue-name-1* および **imqsgets** *queue-name-2* を実行します。

C++ 言語に関する考慮事項

このトピック・コレクションでは、Message Queue Interface (MQI) を使用するアプリケーション・プログラムを作成する際に考慮しなければならない C++ 言語の使用法および規則について詳述します。

C++ ヘッダー・ファイル

C++ 言語での WebSphere MQ アプリケーション・プログラムの 作成を支援するために、MQI の定義の一部としてヘッダー・ファイルが用意されています。

これらのヘッダー・ファイルを、以下の表に要約します。

表 *81. C/C++* ヘッダー・ファイル

ファイル名	目次
IMQI.HPP	C++ MQI クラス (CMQC.H および IMQTYPE.H を含む)
IMQTYPE.H	**ImqBoolean** データ・タイプを定義します
CMQC.H	MQI データ構造体およびマニフェスト定数

アプリケーションの移植性を向上させるために、次のように、ヘッダー・ファイルの名前を **#include** プリプロセッサー指示に小文字でコーディングしてください。

```
#include <imqi.hpp> // C++ classes
```

C++ のメソッドと属性

メソッド名は大/小文字混合です。 パラメーターと戻り値には、さまざまな考慮事項が適用されます。 属性には、必要に応じてゲット・メソッドおよびセット・メソッドを使用してアクセスします。

const となっているメソッドのパラメーターは入力専用です。 シグニチャーにポインター (*) または参照 (&) が含まれているパラメーターは、参照により渡されます。 ポインターや参照を含んでいない戻り値は、値により渡されます。返されたオブジェクトの場合、新規エンティティーであるこれらのオブジェクトは呼び出し側の責任となります。

一部のメソッド・シグニチャーには、指定がない場合にデフォルトをとる項目があります。 そのような項目は、必ずシグニチャーの終わりにあり、等号 (=) で示されています。この等号の後の値は、その項目が省略された場合に適用されるデフォルト値を示します。

これらのクラス内のメソッド名はすべて、小文字で始まり、残りの文字は大文字と小文字が混在しています。 メソッド名の最初のワードを除き、各ワードは、大文字で始まります。 意味が一般的に理解されない限り、省略語は使用されません。 使用される省略語には、*id* (ID の意) および *sync* (「同期」 の意) が含まれます。

オブジェクト属性には、「set」メソッドおよび「get」メソッドを使用してアクセスします。 set メソッドは ワード *set* で始まりますが、get メソッドには接頭部はありません。 属性が読み取り専用 であれば、「set」メソッドはありません。

属性はオブジェクトの作成中に有効な状態に初期設定されるため、オブジェクトの状態は常に一貫しています。

C++ のデータ・タイプ

データ・タイプはすべて、C **typedef** ステートメントによって定義されます。

タイプ **ImqBoolean** は IMQTYPE.H で **unsigned char** として定義されており、値 TRUE および FALSE を取ることができます。 **MQBYTE** 配列の代わりに **ImqBinary** クラス・オブジェクトを 使用したり、**char *** の代わりに **ImqString** クラス・オブジェクトを使用したりできます。 多くのメソッドで、ストレージ管理を容易にするために、**char** ポインターや **MQBYTE** ポインターの代わりにオブジェクトが返されます。 すべての戻り値は呼び出し元の責任となり、戻り値がオブジェクトの場合は、delete を使用してその記憶域を破棄できます。

C++ における 2 進ストリングの操作

2 進データのストリングは、**ImqBinary** クラスのオブジェクトとして宣言されます。 このクラスのオブジェクトは、一般的な C 演算子を使用して、コピー、比較、および設定することができます。 コード例が提供されています。

以下のコード例は、2 進ストリングに対する操作を示しています。

```
#include <imqi.hpp> // C++ classes

ImqMessage message ;
ImqBinary id, correlationId ;
MQBYTE24 byteId ;

correlationId.set( byteId, sizeof( byteId ) ); // Set.
id = message.id( );                            // Assign.
if ( correlationId == id ) {                   // Compare.
  ...
```

C++ における文字ストリングの操作

文字データは、変換演算子を使用して **char *** にキャストできる **ImqString** クラスのオブジェクトで返されることがよくあります。 ImqString クラスには、文字ストリングの処理を支援するメソッドが含まれています。

MQI C++ メソッドを使用して文字データが受け入れられたり返されたりするとき、その文字データは、必ずヌルで終わるため 任意の長さにできます。 ただし、WebSphere MQ には特定の制限があるため、情報が切り捨てられることがあります。 ストレージ管理を容易にするために、ほとんどの文字データが **ImqString** クラス・オブジェクトに入れて返されます。 このようなオブジェクトは、 所定の変換演算子を使用して **char *** にキャストでき、**char *** を必要とする多くの状況で読み取り専用 で使用できます。

注 : ImqString クラス・オブジェクトからの **char *** 変換結果は、 ヌルになることがあります。

C 関数は **char ***で使用できますが、 **ImqString** クラスには、望ましい特殊なメソッドがあります。 **operator length**() **strlen** および **storage**() と同等です。 文字データに割り振られるメモリーを示します。

C++ のオブジェクトの初期状態

すべてのオブジェクトには、それぞれの属性により表された一貫した初期状態があります。 初期値は、クラス記述に定義されます。

C++ からの C の使用

C++ プログラムからの C 関数を使用する場合は、適切なヘッダーをインクルードします。

次の例は、C++ プログラムにインクルードされた string.h を示しています。

```
extern "C" {
#include <string.h>
}
```

C++ の表記規則

この例は、メソッドの呼び出し方法とパラメーターの宣言方法を示しています。

このコード・サンプルでは、メソッドとパラメーター **ImqBoolean ImqQueue**::**get**(**ImqMessage &** *msg*) を使用します。

パラメーターを以下のように宣言し、使用します。

```
ImqQueueManager * pmanager ;    // Queue manager
ImqQueue * pqueue ;             // Message queue
ImqMessage msg ;                // Message
char szBuffer[ 100 ];          // Buffer for message data

pmanager = new ImqQueueManager ;
pqueue = new ImqQueue ;
pqueue -> setName( "myreplyq" );
pqueue -> setConnectionReference( pmanager );

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );

if ( pqueue -> get( msg ) ) {
  long lDataLength = msg.dataLength( );

  ...
}
```

C++ による暗黙の操作

メソッドが正常に実行されるためには前提条件を満たすことが必要ですが、そのための動作が適時に かつ暗黙的に実行されることがあります。 これらの暗黙命令とは、接続、オープン、再オープン、クローズ、および切断です。 接続とオープンの暗黙的な動作は、クラス属性によって制御できます。

接続

ImqQueueManager オブジェクトは、結果的に MQI に対するなんらかの呼び出し (C++ と MQI の相互参照を参照) が行われるメソッドの場合に自動的に接続されます。

オープン

ImqObject オブジェクトは、結果的に MQGET、MQINQ、MQPUT、または MQSET 呼び出しが発生するメソッドの場合に自動的にオープンされます。 **openFor** メソッドは、1 つまたは複数の関連する **open option** 値を指定するために使用します。

再オープン

ImqObject は、結果的に MQGET、MQINQ、MQPUT、または MQSET 呼び出しが発生するメソッドの場合に自動的に再オープンされます。これらの呼び出しでは、オブジェクトは既にオープンされていますが、既存の **open options** には、MQI 呼び出しが正常に行われるほど十分な機能はありません。 オブジェクトは、MQCO_NONE という一時的な **close options** 値を使用して一時的にクローズされます。 **openFor** メソッドを使用して、関連するものを追加**オープン・オプション**。

条件によっては再オープンで問題が発生する場合もあります。

- 一時動的キューは、クローズされたときに破棄され、再オープンすることはできません。

- 排他的入力用としてオープンされた (明示的に、あるいはデフォルトによる) キューは、クローズ時点と再オープン時点でウィンドウから他のプロセスによりアクセスされることがあります。
- ブラウズ・カーソル位置についての情報は、キューがクローズされた時点で失われます。 このような状態になっても、クローズや再オープンができなくなることはありませんが、この後、再度 MQGMO_BROWSE_FIRST が使用されるまでカーソルは使用できなくなります。
- 最後に取り出されたメッセージのコンテキストは、キューはクローズされた時点で失われます。

上記の状態のいずれかが発生したり、予測できる場合は、オブジェクトが (明示的または暗黙的に) オープンされる前に、適切な **open options** を明示的に設定して、再オープンを避けてください。

複雑なキューの取扱状態について明示的に **open options** を設定すると、結果的にパフォーマンスが向上し、再オープンの使用にかかわる問題が回避されます。

クローズ
ImqObject は、オブジェクト状態が実行可能でなくなった時点で (例えば、ImqObject の connection reference が切断された場合や、ImqObject オブジェクトが破棄された場合)、自動的にクローズされます。

切断
ImqQueueManager は、接続が実行可能でなくなった時点で (例えば、ImqObject の connection reference が切断された場合や、ImqQueueManager オブジェクトが破棄された場合)、自動的に切断されます。

C++ における 2 進ストリングと文字ストリング

ImqString クラスは、従来の *char * データ・フォーマットをカプセル化します。 ImqBinary クラスは、2 進のバイト配列をカプセル化します。 文字データを設定するメソッドには、データを切り捨てるものがあります。

文字 (**char ***) データを設定するメソッドでは、必ずデータのコピーが 作成されますが、WebSphere MQ には 特定の制限があるため、メソッドによってはそのコピーの上限を超えた部分は切り捨てられる場合があります。

ImqString クラス (ImqString C++ クラスを参照) は、従来の **char *** をカプセル化し、次の項目をサポートします。

- 比較
- 連結
- コピー
- 整数とテキスト間の変換
- トークン (ワード) 抽出
- 大文字変換

ImqBinary クラス (ImqBinary C++ クラスを参照) は任意のサイズの 2 進バイト・アレイをカプセル化します。 このクラスは、特に以下の属性を保持するために使用されます。

- **accounting token** (MQBYTE32)
- **connection tag** (MQBYTE128)
- **correlation id** (MQBYTE24)
- **facility token** (MQBYTE8)
- **group id** (MQBYTE24)
- **instance id** (MQBYTE24)
- **message id** (MQBYTE24)
- **message token** (MQBYTE16)
- **transaction instance id** (MQBYTE16)

これらの属性は次のクラスのオブジェクトに属しています。

- ImqCICSBridgeHeader (ImqCICSBridgeHeader C++ クラスを参照)
- ImqGetMessageOptions (ImqGetMessageOptions C++ クラスを参照)
- ImqIMSBridgeHeader (ImqIMSBridgeHeader C++ クラスを参照)
- ImqMessageTracker (ImqMessageTracker C++ クラスを参照)
- ImqQueueManager (ImqQueueManager C++ クラスを参照)
- ImqReferenceHeader (ImqReferenceHeader C++ クラスを参照)
- ImqWorkHeader (ImqWorkHeader C++ クラスを参照)

ImqBinary クラスは、比較とコピーもサポートします。

C++ ではサポートされない機能

WebSphere MQ C++ のクラスと メソッドは、WebSphere MQ プラットフォームに依存しないように設計されています。 したがって、備えている機能の一部が特定のプラットフォーム上でサポートされていない場合もあります。

ある機能を、 その機能がサポートされていないプラットフォーム上で使用しようとした場合、 その機能は WebSphere MQ によって 検出されますが、C++ 言語バインディングには検出されません。 WebSphere MQ は、他の MQI エラーと同様に、 プログラムにエラーを報告します。

C++ でのメッセージング

このトピック集では、C++ でメッセージングを準備して読み書きする方法について説明します。

C++ によるメッセージ・データの作成

メッセージ・データは、システムまたはアプリケーションが提供できるバッファー内に作成されます。 いずれの方法にも利点がいくつかあります。 バッファーの使用例がいくつか提供されています。

メッセージを送信する際に、まず最初に、ImqCache オブジェクト (ImqCache C++ クラスを参照) によって管理されるバッファーでメッセージ・データが作成されます。 バッファーは、(継承により) 各 ImqMessage オブジェクト (ImqMessage C++ クラスを参照) と関連付けられています。そのため、バッファーは (**useEmptyBuffer** または **useFullBuffer** メソッドを使用して)、アプリケーションにより提供されますが、システムにより自動的に提供することもできます。 メッセージ・バッファーを提供するアプリケーションの利点は、そのアプリケーションが作成されたデータ域を直接使用できるため、ほとんどの場合データのコピーが不要であることです。 欠点は、提供されたバッファーが固定長であることです。

バッファーは再利用できます。また、伝送バイト数は、各伝送時に必要に応じて変更できます。この変更には、送信前に **setMessageLength** メソッドを使用します。

システムにより自動的に提供される場合、使用可能なバイト数はシステムによって管理されるため、例えば、ImqCache の **write** メソッドまたは ImqMessage の **writeItem** メソッドを使用してデータをメッセージ・バッファーにコピーすることができます。 メッセージ・バッファーは、必要に応じて大きくなります。バッファーが大きくなる際に、以前に書き込まれたデータが失われることはありません。 大きなメッセージや複数の部分から成るメッセージは分割して、各部分を続けて書き込むことができます。

次の例は、単純化されたメッセージ送信を示しています。

1. ユーザーの提供するバッファーにある、準備済みデータを使用します。

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. ユーザーの提供するバッファーにある、準備済みデータを使用します。バッファー・サイズがデータ・サイズより大きくなっています。

```
char szBuffer[ 24 ] = "Hello world" ;
```

```
msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. ユーザー提供のバッファーにデータをコピーします。

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. システム提供のバッファーにデータをコピーします。

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. オブジェクトを使用して、システム提供のバッファーにデータをコピーします。(オブジェクトは内容だけでなくメッセージ・フォーマットを設定します。)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

C++ によるメッセージの読み取り

バッファーは、アプリケーションまたはシステムが提供できます。 データは、バッファーから直接アクセスするか、順次に読み取ることができます。 各メッセージ・タイプには、それと等価なクラスがあります。サンプル・コードが提供されています。

データを受信する際に、アプリケーションまたはシステムは、適切なメッセージ・バッファーを提供します。 特定の ImqMessage オブジェクトの複数の伝送および複数の受信の両方に、同一のバッファーを使用できます。 メッセージ・バッファーは、自動的に提供された場合、どのような長さのデータでも受信できるように大きくなります。 ただし、アプリケーションによって提供されたメッセージ・バッファーが、受信したデータを保持するのに十分な大きさではない場合があります。 その場合には、メッセージ受信に使用されるオプションに応じて、切り捨てが発生するか、受信が失敗します。

着信データは、メッセージ・バッファーから直接アクセスできますが、その場合、データ長は着信データの合計量を示します。 これとは別に、着信データをメッセージ・バッファーから順番に読み取ることができます。 この場合、データ・ポインターは着信データの次のバイトをアドレッシングし、データ・ポインターおよびデータ長はデータが読み取られるたびに更新されます。

項目 とはメッセージの各部分のことで、すべてがメッセージ・バッファーのユーザー域に入っています。項目は、順番に別個に処理する必要があります。 通常のユーザー・データと異なり、項目は送達不能ヘッダーまたはトリガー・メッセージであっても構いません。 項目は、必ずメッセージ形式と関連付けられています。ただし、メッセージ形式は必ずしも項目と**関連付けられてはいません**。

各項目ごとに、認識可能な WebSphere MQ メッセージ形式に対応するオブジェクトのクラスがあります。各送達不能ヘッダーおよび各トリガー・メッセージごとに 1 つずつあります。 ユーザー・データについてのオブジェクト・クラスはありません。 つまり、認識可能な形式が使い尽くされてしまうと、残りの部分の処理はアプリケーション・プログラムに任されます。 ユーザー・データのクラスは、ImqItem クラスを限定することによって作成できます。

次の例で示すメッセージ受信は、ユーザー・データに先行するいくつかの項目を想定し、そのような項目の処理を考慮しています。 項目に属さないユーザー・データは、項目が特定されたあとに受信されるデー

タとして定義されます。 任意の大きさのメッセージ・データを格納するには自動バッファー (デフォルト) を使用します。

```
ImqQueue queue ;
ImqMessage msg ;

if ( queue.get( msg ) ) {

  /* Process all items of data in the message buffer. */
  do while ( msg.dataLength( ) ) {
    ImqBoolean bFormatKnown = FALSE ;
    /* There remains unprocessed data in the message buffer. */

    /* Determine what kind of item is next. */

    if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
      ImqDeadLetterHeader header ;
      /* The next item is a dead-letter header.                */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( header ) ) {
        /* The dead-letter header has been extricated from the */
        /* buffer and transformed into a dead-letter object.   */
        /* The encoding and character set of the dead-letter   */
        /* object itself are MQENC_NATIVE and MQCCSI_Q_MGR.    */
        /* The encoding and character set from the dead-letter */
        /* header have been copied to the message attributes   */
        /* to reflect any remaining data in the buffer.        */

        /* Process the information in the dead-letter object.  */
        /* Note that the encoding and character set have       */
        /* already been processed.                             */
        ...
      }
      /* There might be another item after this, */
      /* or just the user data.                  */
    }
    if ( msg.formatIs( MQFMT_TRIGGER ) ) {
      ImqTrigger trigger ;
      /* The next item is a trigger message.                   */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;
      if ( msg.readItem( trigger ) ) {

        /* The trigger message has been extricated from the */
        /* buffer and transformed into a trigger object.    */
        /* Process the information in the trigger object. */
        ...
      }

      /* There is usually nothing after a trigger message. */
    }

    if ( msg.formatIs( FMT_USERCLASS ) ) {
      UserClass object ;
      /* The next item is an item of a user-defined class.     */
      /* For the next statement to work and return TRUE,       */
      /* the correct class of object pointer must be supplied. */
      bFormatKnown = TRUE ;

      if ( msg.readItem( object ) ) {
        /* The user-defined data has been extricated from the */
        /* buffer and transformed into a user-defined object. */

        /* Process the information in the user-defined object. */
        ...
      }

      /* Continue looking for further items. */
    }
    if ( ! bFormatKnown ) {
      /* There remains data that is not associated with a specific*/
      /* item class.                                              */
      char * pszDataPointer = msg.dataPointer( );       /* Address.*/
      int iDataLength = msg.dataLength( );              /* Length. */
```

```
      /* The encoding and character set for the remaining data are */
      /* reflected in the attributes of the message object, even   */
      /* if a dead-letter header was present.                      */
      ...

    }

  }
}
```

この例の中の FMT_USERCLASS は、UserClass クラスのオブジェクトに対応する形式の名前 (8 文字) を表す定数です。この定数は、アプリケーションによって定義されます。

UserClass は、ImqItem クラス (ImqItem C++ クラスを参照) の派生クラスです。UserClass では ImqItem クラスの仮想メソッド **copyOut** および **pasteIn** を使用します。

次に ImqDeadLetterHeader クラス (ImqDeadLetterHeader C++ クラスを参照) のコード例を 2 つ示します。 1 つ目の例は、カプセル化されたメッセージ書き込み用 カスタム・コードを示します。

```
// Insert a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: copyOut ( ImqMessage & msg ) {
  ImqBoolean bSuccess ;
  if ( msg.moreBytes( sizeof( omqdlh ) ) ) {
    ImqCache cacheData( msg ); // Preserve original message content.
    // Note original message attributes in the dead-letter header.
    setEncoding( msg.encoding( ) );
    setCharacterSet( msg.characterSet( ) );
    setFormat( msg.format( ) );

    // Set the message attributes to reflect the dead-letter header.
    msg.setEncoding( MQENC_NATIVE );
    msg.setCharacterSet( MQCCSI_Q_MGR );
    msg.setFormat( MQFMT_DEAD_LETTER_HEADER );
    // Replace the existing data with the dead-letter header.
    msg.clearMessage( );
    if ( msg.write( sizeof( omqdlh ), (char *) & omqdlh ) ) {
      // Append the original message data.
      bSuccess = msg.write( cacheData.messageLength( ),
                            cacheData.bufferPointer( ) );
    } else {
      bSuccess = FALSE ;
    }
  } else {
    bSuccess = FALSE ;
  }
  // Reflect and cache error in this object.
  if ( ! bSuccess ) {
    setReasonCode( msg.reasonCode( ) );
    setCompletionCode( msg.completionCode( ) );
  }

  return bSuccess ;
}
```

2 つ目の例は、カプセル化されたメッセージ読み取り用 カスタム・コードを示します。

```
// Read a dead-letter header.
// Return TRUE if successful.
ImqBoolean ImqDeadLetterHeader :: pasteIn ( ImqMessage & msg ) {
  ImqBoolean bSuccess = FALSE ;

  // First check that the eye-catcher is correct.
  // This is also our guarantee that the "character set" is correct.
  if ( ImqItem::structureIdIs( MQDLH_STRUC_ID, msg ) ) {
    // Next check that the "encoding" is correct, as the MQDLH
    // contains numeric data.
    if ( msg.encoding( ) == MQENC_NATIVE ) {

      // Finally check that the "format" is correct.
      if ( msg.formatIs( MQFMT_DEAD_LETTER_HEADER ) ) {
        char * pszBuffer = (char *) & omqdlh ;
        // Transfer the MQDLH from the message and move pointer on.
        if ( bSuccess = msg.read( sizeof( omdlh ), pszBuffer ) ) {
```

```
          // Update the encoding, character set and format of the
          // message to reflect the remaining data.
          msg.setEncoding( encoding( ) );
          msg.setCharacterSet( characterSet( ) );
          msg.setFormat( format( ) );
        } else {

          // Reflect the cache error in this object.
          setReasonCode( msg.reasonCode( ) );
          setCompletionCode( msg.completionCode( ) );
        }
      } else {
        setReasonCode( MQRC_INCONSISTENT_FORMAT );
        setCompletionCode( MQCC_FAILED );
      }
    } else {
      setReasonCode( MQRC_ENCODING_ERROR );
      setCompletionCode( MQCC_FAILED );
    {
  } else {
    setReasonCode( MQRC_STRUC_ID_ERROR );
    setCompletionCode( MQCC_FAILED );
  }

  return bSuccess ;
}
```

自動バッファーでは、バッファー記憶が揮発性 です。 つまり、バッファー・データは、**get** メソッド呼び出しのたびに、物理位置が変わる可能性があります。 したがって、バッファー・データが参照されるたびに、**bufferPointer** メソッドまたは **dataPointer** メソッドを使用してメッセージ・データにアクセスしてください。

メッセージ・データの受信用に固定記憶域を確保しておくプログラムも書くことができます。 この場合、**get** メソッドを使用する前に **useEmptyBuffer** メソッドを呼び出すことができます。

固定の非自動領域を使用すると、メッセージは最大限のサイズに制限されるため、ImqGetMessageOptions オブジェクトの MQGMO_ACCEPT_TRUNCATED_MSG オプションを考慮に入れることが重要です。 このオプションを指定しない場合 (デフォルトです) は、MQRC_TRUNCATED_MSG_FAILED 理由コードが戻ると予想できます。 このオプションを指定した場合は、アプリケーションの設計により、MQRC_TRUNCATED_MSG_ACCEPTED という理由コードが戻る場合もあります。

次のコード例は、固定記憶域を使用してメッセージをどのように受信することができるかを示しています。

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

このコード例では、バッファーは、**bufferPointer** メソッドを使用した場合とは逆に、 常に *pszBuffer* メソッドで直接アドレッシングすることができます。 ただし、汎用アクセスの場合は、**dataPointer** メソッドを使用したほうがよいでしょう。 アプリケーション (ImqCache クラス・オブジェクトではない) では、ユーザー定義の (非自動) バッファーを廃棄する必要があります。

注意: useEmptyBuffer を使用してヌル・ポインターと、ゼロの長さを指定しても、 当然のことながら、ゼロの長さの固定長バッファーが指定されることはありません。 この組み合わせは、あらゆる直前のユーザー定義バッファーを無視し、代わりに元どおりに自動バッファーを使用するという要求として解釈されます。

C++ による送達不能キューへのメッセージの書き込み

送達不能キューにメッセージを書き込むためのプログラム・コードの例。

複数の部分から成るメッセージの代表的な例として、送達不能ヘッダーを持つメッセージがあります。 処理できないメッセージからのデータは、送達不能ヘッダーの最後に追加されます。

```
ImqQueueManager mgr ;        // The queue manager.
ImqQueue queueIn ;           // Incoming message queue.
ImqQueue queueDead ;         // Dead-letter message queue.
ImqMessage msg ;             // Incoming and outgoing message.
ImqDeadLetterHeader header ; // Dead-letter header information.

// Retrieve the message to be rerouted.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the dead-letter header information.
header.setDestinationQueueManagerName( mgr.name( ) );
header.setDestinationQueueName( queueIn.name( ) );
header.setPutApplicationName( /* ? */ );
header.setPutApplicationType( /* ? */ );
header.setPutDate( /* TODAY */ );
header.setPutTime( /* NOW */ );
header.setDeadLetterReasonCode( FB_APPL_ERROR_1234 );

// Insert the dead-letter header information. This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the dead-letter queue.
queueDead.setConnectionReference( mgr );
queueDead.setName( mgr.deadLetterQueueName( ) );
queueDead.put( msg );
```

C++ による IMS ブリッジへのメッセージの書き込み

IMS ブリッジにメッセージを書き込むためのプログラム・コードの例。

WebSphere MQ-IMS ブリッジに送信されるメッセージでは、特別なヘッダーが使用されることがあります。IMS ブリッジ・ヘッダーが、 通常のメッセージ・データの前に付けられます。

```
ImqQueueManager    mgr;         // The queue manager.
ImqQueue           queueBridge; // IMS bridge message queue.
ImqMessage         msg;         // Outgoing message.
ImqIMSBridgeHeader header;      // IMS bridge header.

// Set up the message.
//
// Here we are constructing a message with format
// MQFMT_IMS_VAR_STRING, and appropriate data.
//
msg.write( 2,       /* ? */ );         // Total message length.
msg.write( 2,       /* ? */ );         // IMS flags.
msg.write( 7,       /* ? */ );         // Transaction code.
msg.write( /* ? */, /* ? */ );         // String data.
msg.setFormat( MQFMT_IMS_VAR_STRING ); // The format attribute.

// Set up the IMS bridge header information.
//
// The reply-to-format is often specified.
// Other attributes can be specified, but all have default values.
//
header.setReplyToFormat( /* ? */ );

// Insert the IMS bridge header into the message.
//
// This will:
// 1) Insert the header into the message buffer, before the existing
//    data.
// 2) Copy attributes out of the message descriptor into the header,
//    for example the IMS bridge header format attribute will now
//    be set to MQFMT_IMS_VAR_STRING.
// 3) Set up the message attributes to describe the header, in
//    particular setting the message format to MQFMT_IMS.
//
msg.writeItem( header );

// Send the message to the IMS bridge queue.
//
```

```
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++ による CICS ブリッジへのメッセージの書き込み

CICS ブリッジにメッセージを書き込むためのプログラム・コードの例。

CICS ブリッジを使用して WebSphere MQ for z/OS に送信されるメッセージには、特別なヘッダーが必要です。 CICS ブリッジ・ヘッダーは、通常のメッセージ・データの前にきます。

```
ImqQueueManager mgr ;         // The queue manager.
ImqQueue queueIn ;            // Incoming message queue.
ImqQueue queueBridge ;        // CICS bridge message queue.
ImqMessage msg ;              // Incoming and outgoing message.
ImqCicsBridgeHeader header ;  // CICS bridge header information.

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Set up the CICS bridge header information.
// The reply-to format is often specified.
// Other attributes can be specified, but all have default values.
header.setReplyToFormat( /* ? */ );

// Insert the CICS bridge header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the CICS bridge queue.
queueBridge.setConnectionReference( mgr );
queueBridge.setName( /* ? */ );
queueBridge.put( msg );
```

C++ による作業ヘッダーでのメッセージの書き込み

z/OS ワークロード・マネージャーによって管理されるキュー宛のメッセージを書き込むためのプログラム・コードの例。

WebSphere MQ for z/OS に 送信されるメッセージの場合、z/OS Workload Manager に よって管理されているキューが宛先となります。 このようなメッセージには、特別なヘッダーが必要です。 作業ヘッダーは、通常のメッセージ・データの前に付けられます。

```
ImqQueueManager mgr ;             // The queue manager.
ImqQueue queueIn ;                // Incoming message queue.
ImqQueue queueWLM ;               // WLM managed queue.
ImqMessage msg ;                  // Incoming and outgoing message.
ImqWorkHeader header ;            // Work header information

// Retrieve the message to be forwarded.
queueIn.setConnectionReference( mgr );
queueIn.setName( MY_QUEUE );
queueIn.get( msg );

// Insert the Work header information.  This will vary
// the encoding, character set and format of the message.
// Message data is moved along, past the header.
msg.writeItem( header );

// Send the message to the WLM managed queue.
queueWLM.setConnectionReference( mgr );
queueWLM.setName( /* ? */ );
queueWLM.put( msg );
```

WebSphere MQ C++ プログラムの作成

サポートされるコンパイラーの URL が、WebSphere MQ プラットフォーム上で C++ プログラムおよびサンプルをコンパイル、リンク、および実行するために使用するコマンドとともにリストされます。

WebSphere MQ のサポートされる各プラットフォームおよび各バージョン用のコンパイラーは、 IBM WebSphere MQ の WebSphere MQ システム要件のページにリストされています。

WebSphere MQ C++ プログラムをコンパイルおよびリンクするために必要なコマンドは、ご使用のインストール済み環境および要件により異なります。 以下の例では、複数のプラットフォーム上における、WebSphere MQ のデフォルトのインストール済み環境を使用するコンパイラー用の標準的なコンパイル・コマンドおよびリンク・コマンドが示されています。

AIX における C++ プログラムの作成

AIX で WebSphere MQ C++ プログラムを作成するには、XL C Enterprise Edition コンパイラーを使用します。

クライアント

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

32 ビット非スレッド・アプリケーション

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

32 ビット・スレッド・アプリケーション

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

64 ビット非スレッド・アプリケーション

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

64 ビット・スレッド・アプリケーション

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

サーバー

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

32 ビット非スレッド・アプリケーション

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

32 ビット・スレッド・アプリケーション

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

64 ビット非スレッド・アプリケーション

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

64 ビット・スレッド・アプリケーション

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

HP-UX における C++ プログラムの作成

HP-UX で WebSphere MQ C++ プログラムを作成するには、aC++ コンパイラーまたは aCC コンパイラーを使用します。

HP-UX Itanium において、WebSphere MQ は Standard ランタイムのみをサポートします。 aCC コンパイラーを使用してください。

- libimqi23bh.sl は、Standard ランタイム用の WebSphere MQ C++ クラスを提供します。
- 以前のリリースとの互換性のために、libimqi23ah.sl から libimqi23bh.sl へのシンボリック・リンクが提供されています。

IA64 (IPF)

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: IA64 (IPF)

32 ビット非スレッド・アプリケーション

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

32 ビット・スレッド・アプリケーション

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqic
```

64 ビット・スレッド・アプリケーション

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

サーバー: IA64 (IPF)

32 ビット非スレッド・アプリケーション

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

32 ビット・スレッド・アプリケーション

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

64 ビット・スレッド・アプリケーション

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

Linux における C++ プログラムの作成

GNU g + + コンパイラーを使用して Linux 上で WebSphere MQ C++ プログラムをビルドします。

System p

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: System p

32 ビット非スレッド・アプリケーション

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

サーバー: System p

32 ビット非スレッド・アプリケーション

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
```

```
-limqs23gl
-limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

System z

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: System z

32 ビット非スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64  -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

サーバー: System z

32 ビット非スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

System x (32 ビット)

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: System x (32 ビット)

32 ビット非スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,
-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

サーバー: System x (32-bit)

32 ビット非スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

32 ビット・スレッド・アプリケーション

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATHlib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
```

```
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

64 ビット非スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

64 ビット・スレッド・アプリケーション

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Solaris における C++ プログラムの作成

Solaris で WebSphere MQ C++ プログラムを作成するには、Sun ONE コンパイラーを使用します。

SPARC

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: SPARC

32 ビット・アプリケーション

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64 ビット・アプリケーション

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

サーバー: SPARC

32 ビット・アプリケーション

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64 ビット・アプリケーション

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント: x86-64

32 ビット・アプリケーション

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

64 ビット・アプリケーション

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

サーバー: x86-64

32 ビット・アプリケーション

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

64 ビット・アプリケーション

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

Windows における C++ プログラムの作成

Windows で WebSphere MQ C++ プログラムを作成するには、Microsoft Visual Studio C++ コンパイラーを使用します。

32 ビット・アプリケーションで使用されるライブラリー (.lib) ファイルおよび dll ファイルは、*MQ_INSTALLATION_PATH*/Tools/Lib にインストールされ、64 ビット・アプリケーションで使用されるファイルは *MQ_INSTALLATION_PATH*/Tools/Lib64 にインストールされます。
MQ_INSTALLATION_PATH WebSphere MQ がインストールされている上位ディレクトリーを表します。

クライアント

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

サーバー

```
 cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

WebSphere MQ classes for Java の使用

WebSphere MQ classes for Java を使用すると、Java 環境で WebSphere MQ を使用できます。 Java アプリケーションは、 WebSphere MQ classes for Java または WebSphere MQ classes for JMS のいずれかを使用して、 WebSphere MQ リソースにアクセスできます。

WebSphere MQ classes for Java により、Java アプリケーションは以下を行うことができます。

- WebSphere MQ クライアントとしての WebSphere MQ への接続
- WebSphere MQ キュー・マネージャーへの直接接続

WebSphere MQ classes for Java は、 Message Queue Interface (MQI) (ネイティブ WebSphere MQ API) をカプセル化します。

WebSphere MQ classes for Java は、WebSphere MQ の C++ および .NET インターフェースと類似したオブジェクト・モデルを使用します。

WebSphere MQ classes for Java を使用する理由

ご使用のシステムで以下の点が重要な場合は、 WebSphere MQ classes for Java の使用を検討してください。

- WebSphere MQ classes for Java は、Message Queue Interface (MQI) (ネイティブ WebSphere MQ API) をカプセル化します。
  - プロシージャー型言語での MQI の使用に精通している場合は、この知識を Java 環境に移すことができます。
  - JMS を通して利用可能な範囲を超えて、WebSphere MQ 全範囲の機能を活用することができます。
- WebSphere MQ classes for Java は、WebSphere MQ の C++ および .NET インターフェースと類似したオブジェクト・モデルを使用します。 これらのインターフェースに精通している場合は、この知識を Java 環境に移すことができます。

注 : WebSphere MQ classes for Java では、自動クライアント再接続はサポートされていません。

WebSphere MQ classes for Java の概要

この一連のトピックでは、 WebSphere MQ classes for Java の概要とその用途について説明します。

WebSphere MQ classes for Java とは何ですか?

WebSphere MQ classes for Java を使用すると、Java 環境で WebSphere MQ を使用できます。

WebSphere MQ classes for Java により、Java アプリケーションは以下を行うことができます。

- WebSphere MQ クライアントとしての WebSphere MQ への接続
- WebSphere MQ キュー・マネージャーへの直接接続

WebSphere MQ classes for Java は、 Message Queue Interface (MQI) (ネイティブ WebSphere MQ API) をカプセル化します。

WebSphere MQ classes for Java は、WebSphere MQ の C++ および .NET インターフェースと類似したオブジェクト・モデルを使用します。

WebSphere MQ classes for Java を使用する理由

Java アプリケーションは、WebSphere MQ classes for Java または WebSphere MQ classes for JMS のいずれかを使用して、 WebSphere MQ リソースにアクセスできます。 WebSphere MQ classes for Java を使用することには、いくつかの利点があります。

ご使用のシステムで以下の点が重要な場合は、Websphere MQ classes for Java の使用を検討してください。

- WebSphere MQ classes for Java は、Message Queue Interface (MQI) (ネイティブ WebSphere MQ API) をカプセル化します。
  - プロシージャー型言語での MQI の使用に精通している場合は、この知識を Java 環境に移すことができます。
  - JMS を通して利用可能な範囲を超えて、WebSphere MQ 全範囲の機能を活用することができます。
- WebSphere MQ classes for Java は、WebSphere MQ の C++ および .NET インターフェースと類似したオブジェクト・モデルを使用します。 これらのインターフェースに精通している場合は、この知識を Java 環境に移すことができます。

WebSphere MQ classes for Java の接続オプション

WebSphere MQ classes for Java は、クライアント・モードまたはバインディング・モードで接続できます。

プログラマブル・オプションを使用すると、 WebSphere MQ classes for Java は、以下のいずれかの方法で WebSphere MQ に接続できます。

- 伝送制御プロトコル/インターネット・プロトコル (TCP/IP) を使用する WebSphere MQ MQI クライアントとして接続。
- バインディング・モードで、Java Native Interface (JNI) を使用して WebSphere MQ に直接接続する

クライアントを z/OS 上で実行できませんが、Client Attach Facility がインストールされている場合は、他のプラットフォーム上のクライアントを WebSphere MQ for z/OS キュー・マネージャーに接続できます。

以下のセクションでは、クライアント・モードとバインディング・モードの接続オプションについて詳しく説明します。

クライアント 接続

クライアント・モードでキュー・マネージャーに接続するために、WebSphere MQ classes for Java アプリケーションは、キュー・マネージャーが実行されているシステムと同じシステム上で実行することも、別のシステム上で実行することもできます。 いずれの場合も、WebSphere MQ classes for Java は TCP/IP を介してキュー・マネージャーに接続します。

WebSphere MQ classes for Java アプリケーションは、サポートされている任意のキュー・マネージャーにクライアント・モードを使用して接続できます。

クライアント・モード接続を使用するアプリケーションの作成方法について詳しくは、674 ページの『WebSphere MQ classes for Java の接続モード』を参照してください。

バインディング 接続

バインディング・モードで使用する場合、 WebSphere MQ classes for Java は、ネットワークを介して通信するのではなく、Java Native Interface (JNI) を使用して既存のキュー・マネージャー API を直接呼び出します。 ほとんどの環境では、バインディング・モードで接続すると、TCP/IP 通信のコストを回避することにより、クライアント・モードで接続するよりも WebSphere MQ classes for Java アプリケーションのパフォーマンスが向上します。

WebSphere MQ classes for Java を使用してバインディング・モードで接続するアプリケーションは、接続先のキュー・マネージャーと同じシステムで実行されなければなりません。

WebSphere MQ classes for Java アプリケーションを実行するために使用される Java ランタイム環境は、WebSphere MQ classes for Java ライブラリーをロードするように構成する必要があります。詳しくは、WebSphere MQ classes for Java ライブラリー を参照してください。

バインディング・モード接続を使用するアプリケーションの作成方法について詳しくは、674 ページの『WebSphere MQ classes for Java の接続モード』を参照してください。

WebSphere MQ classes for Java の前提条件

WebSphere MQ classes for Java を使用するには、他の特定のソフトウェア製品が必要です。

WebSphere MQ classes for Java の前提条件に関する最新情報については、 WebSphere MQ README ファイルを参照してください。

WebSphere MQ classes for Java アプリケーションを開発するには、Java Development Kit (JDK) が必要です。 ご使用のオペレーティング・システムでサポートされている JDK の詳細については、「WebSphere MQ system requirements」ページ (IBM WebSphere MQ) に記載されています。

WebSphere MQ classes for Java アプリケーションを実行するには、以下のソフトウェア・コンポーネントが必要です。

- キュー・マネージャーに接続するアプリケーションの場合は、WebSphere MQ キュー・マネージャー

- アプリケーションを実行するシステムごとに Java Runtime Environment (JRE) 適合する JRE が WebSphere MQ とともに提供されます。

FIPS 140-2 認定の暗号モジュールを使用するために SSL 接続が必要な場合は、 IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) が必要です。 IBM JDK および JRE のバージョン 1.4.2 またはそれ以降には、IBMJSSEFIPS が含まれています。

ご使用のオペレーティング・システム上の Java 仮想マシン (JVM) および TCP/IP 実装によってサポートされる WebSphere MQ classes for Java アプリケーション IPv6 の場合 で、 Internet Protocol バージョン 6 (IPv6) アドレスを使用できます。

WebSphere MQ classes for Java のインストールと構成

このセクションでは、WebSphere MQ classes for Java のインストール時に作成されるディレクトリーおよびファイルについて説明し、インストール後に WebSphere MQ classes for Java を構成する方法について説明します。

WebSphere MQ classes for Java のインストール内容

最新バージョンの WebSphere MQ classes for Java は、 WebSphere MQ と共にインストールされます。 そうするには、デフォルトのインストール・オプションをオーバーライドする必要があります。

WebSphere MQ のインストール方法については、以下を参照してください。

WebSphere MQ サーバーのインストール
IBM WebSphere MQ クライアントのインストール

WebSphere MQ classes for Java は、Java アーカイブ (JAR) ファイル、 com.ibm.mq.jar、および com.ibm.mq.jmqi.jar に含まれています。

プログラマブル・コマンド・フォーマット (PCF) などの、標準メッセージ・ヘッダーのサポートは、JAR ファイル com.ibm.mq.headers.jar にあります。

プログラマブル・コマンド・フォーマット (PCF) のサポートは、JAR ファイル com.ibm.mq.pcf.jar に含まれています。

WebSphere MQ classes for Java の JAR ファイルのインストールおよびアップグレード

WebSphere MQ classes for Java の JAR ファイルをシステムに取り込む方法としてサポートされているのは、WebSphere MQ 製品または WebSphere MQ MQI Client サポートパックのどちらかをインストールする方法、 または Apache Maven などのソフトウェア管理ツールを使用する方法のみです。 詳しくは、670 ページの『IBM WebSphere MQ classes for Java とソフトウェア管理ツール』を参照してください。

ソフトウェア管理ツールを使用している場合を除き、WebSphere MQ classes for Java の JAR ファイルを他のマシンから移動したりコピーしたりしないでください。

- フィックスパックは、JAR ファイルが別のマシンからコピーされた「インストール済み環境」に適用できません。 また、すべての JAR ファイルが相互に同期を保ち、互換性のあるレベルに保つようにすることが難しくなります。
- WebSphere MQ classes for JMS の JAR ファイルをマシン間でコピーすると、同じマシンにファイルの複数のコピーが存在するようになることがあり、これはコードの保守の問題やデバッグの問題の原因になることがあります。

アプリケーションのアーカイブ内に WebSphere MQ classes for Java の JAR ファイルを含めないでください。

- WebSphere MQ classes for Java に対するアップデートを、WebSphere MQ フィックスパックを使用して適用することができなくなります。
- アプリケーションが使用している WebSphere MQ classes for Java のバージョンを IBM サポートが判別することが困難になります。
- 同じ Java ランタイム環境内で実行されている複数のアプリケーションに異なるバージョンの WebSphere MQ classes for Java が組み込まれていると、同時に複数のバージョンの WebSphere MQ classes for Java が Java ランタイム環境にロードされるため、問題が起きることがあります。

- アプリケーションで BINDINGS トランスポートを使用してキュー・マネージャーに接続している場合は、キュー・マネージャーにメジャー・アップグレードが行われたときにも、それに対応するレベルの WebSphere MQ classes for Java を組み込むようにアプリケーションをアップデートする必要があります。

  例えば、キュー・マネージャーが WebSphere MQ Version 7.1 レベルにアップグレードされた場合であれば、BINDINGS トランスポートを使用してキュー・マネージャーに接続しているすべてのアプリケーションも、WebSphere MQ Version 7.1 classes for Java を組み込むようにアップデートする必要があります。

WebSphere MQ classes for Java には、以下の Java ライブラリーが同梱されています。

- connector.jar (バージョン 1.0)

Postcard というサンプル・アプリケーションが、JAR ファイル com.ibm.mq.postcard.jar にあります。

Javadoc ツールは、 WebSphere MQ classes for Java および WebSphere MQ classes for JMS API の仕様を含む HTML ページを生成するために使用されています。 HTML ページは、WebSphere MQ classes for JMS インストール・ディレクトリーの doc サブディレクトリー中にあります。 UNIX、Linux、および Windows の各システムでは、doc サブディレクトリーに 個別の HTML ページが格納されます。

インストールが完了すると、ファイルやサンプルは、662 ページの『 WebSphere MQ classes for Java のインストール・ディレクトリー』に示された場所にインストールされます。

Windows 以外のプラットフォームでは、インストールの後、環境変数を更新する必要があります (663 ページの『 WebSphere MQ classes for Java に関連する環境変数』を参照)。

WebSphere MQ classes for Java のインストール・ディレクトリー

WebSphere MQ classes for Java ファイルは、プラットフォームに応じて異なる場所にインストールされます。

662 ページの表 82 は、 WebSphere MQ classes for Java ファイルがインストールされる場所を示しています。

表 *82. WebSphere MQ classes for Java* インストール・ディレクトリー

プラットフォーム	**ディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/java/lib
HP-UX、 Linux、および Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*¥java¥lib
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

インストール検査プログラム (IVP) など、一部のサンプル・アプリケーションは WebSphere MQ に付属しています。662 ページの表 83 サンプル・アプリケーションがインストールされている場所を示しています。 WebSphere MQ classes for Java サンプルは、 `wmqjava` というサブディレクトリーにあります。 PCF サンプルは、`pcf` というサブディレクトリー内にあります。

表 *83.* サンプル・ディレクトリー

プラットフォーム	**ディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
HP-UX、 Linux、および Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*¥tools¥wmqjava¥
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

WebSphere MQ classes for Java に関連する環境変数

WebSphere MQ classes for Java アプリケーションを実行する場合は、そのクラスパスに WebSphere MQ classes for Java および samples ディレクトリーを含める必要があります。

WebSphere MQ classes for Java アプリケーションを実行するには、クラスパスに適切な WebSphere MQ classes for Java ディレクトリーが含まれている必要があります。 サンプル・アプリケーションを実行する場合は、クラス・パスに適切な samples ディレクトリーも組み込まなければなりません。 この情報は、Java 呼び出しコマンドまたは CLASSPATH 環境変数で提供できます。

663 ページの表 84 は、 WebSphere MQ classes for Java アプリケーション (サンプル・アプリケーションを含む) を実行するために各プラットフォームで使用する適切な CLASSPATH 設定を示しています。

表 *84. WebSphere MQ classes for Java* アプリケーション *(WebSphere MQ classes for Java* サンプル・アプリケーションを含む*)* を実行するための *CLASSPATH* 設定

プラットフォーム	**CLASSPATH 設定**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
HP-UX、 Linux、および Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*¥tools¥wmqjava¥samples;
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

-Xlint オプションを使用してコンパイルする場合に、com.ibm.mq.ese.jar が存在しないという警告メッセージが表示されることがあります。 この警告は無視して構いません。 このファイルは、IBM WebSphere MQ Advanced Message Security をインストールしている場合にのみ存在します。

WebSphere MQ classes for Java で提供されるスクリプトは、以下の環境変数を使用します。

MQ_JAVA_DATA_PATH
この環境変数は、ログおよびトレース出力のディレクトリーを指定します。

MQ_JAVA_INSTALL_PATH
この環境変数は、 WebSphere MQ classes for Java インストール・ディレクトリーに示すように、WebSphere MQ classes for Java がインストールされるディレクトリーを指定します。

MQ_JAVA_LIB_PATH
この環境変数は、各プラットフォームの WebSphere MQ classes for Java ライブラリーの場所に示すように、 WebSphere MQ classes for Java ライブラリーが保管されるディレクトリーを指定します。 WebSphere MQ classes for Java で提供される一部のスクリプト (IVTRun など) は、この環境変数を使用します。

Windows では、すべての環境変数がインストール時に自動的に設定されます。 その他のプラットフォームでは、ユーザーが自分でこれらを設定しなければなりません。UNIX システムでは、スクリプト **setjmsenv** (32 ビット JVM を使用している場合) または **setjmsenv64** (64 ビット JVM を使用している場合) を使用すれば、環境変数を設定できます。 AIX、 HP-UX、 Linux、および Solaris では、これらのスクリプトは *MQ_INSTALLATION_PATH*/java/bin ディレクトリーにあります。

IBM WebSphere MQ classes for Java ライブラリー

IBM WebSphere MQ classes for Java ライブラリーの場所は、プラットフォームによって異なります。 アプリケーションの開始時にこの場所を指定します。

Java Native Interface (JNI) ライブラリーの場所を指定するには、以下の形式で **java** コマンドを使用してアプリケーションを開始します。

```
java -Djava.library.path=library_path application_name
```

ここで、 *library_path* は、JNI ライブラリーを含む WebSphere MQ classes for Java ライブラリーへのパスです。 664 ページの表 85 は、各プラットフォームの WebSphere MQ classes for Java ライブラリーの場所を示しています。

表 *85.* 各プラットフォームの *WebSphere MQ classes for Java* ライブラリーの場所。

プラットフォーム	**WebSphere MQ classes for Java ライブラリーを含むディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
HP-UX Linux (POWER®、 x86-64 および zSeries s390x プラットフォーム) Solaris (x86-64 および SPARC プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Linux (x86 プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*¥Java¥lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*¥Java¥lib64 (64 ビット・ライブラリー)
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

注 :

1. AIX、 HP-UX、 Linux (Power プラットフォーム)、または Solaris では、32 ビット・ライブラリーまたは 64 ビット・ライブラリーのいずれかを使用します。 64 ビット・ライブラリーは、64 ビット・プラットフォーム上の 64 ビット Java 仮想マシン (JVM) でアプリケーションを実行する場合にのみ使用してください。 それ以外の場合は 32 ビット・ライブラリーを使用してください。
2. Windows では、PATH 環境変数を使用して、 **java** コマンドでロケーションを指定する代わりに、WebSphere MQ classes for Java ライブラリーのロケーションを指定できます。
3. IBM i で WebSphere MQ classes for Java をバインディング・モードで使用するには、ライブラリー QMQMJAVA がライブラリー・リストにあることを確認してください。

関連タスク

WebSphere MQ classes for Java の使用

IBM WebSphere MQ classes for Java 上での *OSGi* のサポート

OSGi は、バンドルの形によるアプリケーションのデプロイメントをサポートするフレームワークを提供します。 1 つの OSGi バンドルが、IBM WebSphere MQ classes for Java の一部として提供されます。

OSGi は、汎用で、安全で、管理された Java フレームワークを提供します。これは、バンドルの形態で付属するアプリケーションのデプロイメントをサポートします。 OSGi 対応デバイスはバンドルをダウンロ

ードしてインストールし、それらが不要になったら削除することができます。 フレームワークはバンドルのインストールおよび更新を動的かつ拡張が容易な仕方で管理します。

IBM WebSphere MQ classes for Java には、以下の OSGi バンドルが組み込まれています。

com.ibm.mq.osgi.java_< version number> .jar
アプリケーションが IBM WebSphere MQ classes for Java を使用できるようにする JAR ファイル。

ここで、< version number> は、インストールされている WebSphere MQ のバージョン番号です。

バンドルは、IBM WebSphere MQ インストール済み環境の `java/lib/OSGi` サブディレクトリー、または Windows の `java\lib\OSGi` フォルダーにインストールされます。

他の 9 つのバンドルも、IBM WebSphere MQ インストール済み環境の `java/lib/OSGi` サブディレクトリー、または Windows の `java\lib\OSGi` フォルダーにインストールされます。 これらのバンドルは IBM WebSphere MQ classes for JMS の一部であり、IBM WebSphere MQ classes for Java バンドルがロードされている OSGi ランタイム環境にロードしてはなりません。 IBM WebSphere MQ classes for Java OSGi バンドルを、既に IBM WebSphere MQ classes for JMS バンドルがロードされている OSGi ランタイム環境にロードすると、次のようなエラー

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

が、IBM WebSphere MQ classes for Java バンドルまたは IBM WebSphere MQ classes for JMS バンドルを使用するアプリケーションを実行したときに発生します。

IBM WebSphere MQ classes for Java 用の OSGi バンドルは、OSGi リリース 4 仕様に合わせて記述されているため、OSGi リリース 3 環境では動作しません。

OSGi ランタイム環境が必要な DLL ファイルまたは共用ライブラリーを検出できるように、システム・パスまたはライブラリー・パスを正しく設定する必要があります。

IBM WebSphere MQ classes for Java 用の OSGi バンドルを使用する場合、Java で記述されたチャネル出口クラスはサポートされません。OSGi などの複数クラス・ローダー環境にクラスをロードする際に固有の問題があるためです。 ユーザー・バンドルは IBM WebSphere MQ classes for Java バンドルを認識できますが、IBM WebSphere MQ classes for Java バンドルはユーザー・バンドルを認識できません。 結果として、IBM WebSphere MQ classes for Java バンドル内で使用されるクラス・ローダーは、ユーザー・バンドル内のチャネル出口クラスをロードできません。

OSGi の詳細については、OSGi Alliance Web サイトをご覧ください。

IBM WebSphere MQ classes for Java 構成ファイル

IBM WebSphere MQ classes for Java 構成ファイルは、 IBM WebSphere MQ classes for Java の構成に使用されるプロパティーを指定します。

IBM WebSphere MQ classes for Java 構成ファイルの形式は、標準の Java プロパティー・ファイルの形式です。

V 7.5.0.9 IBM WebSphere MQ Version 7.5.0、フィックスパック 9 以降、 `mqjava.config` という名前のサンプル構成ファイルが、 IBM WebSphere MQ classes for Java インストール・ディレクトリーの `bin` サブディレクトリーに用意されています。 このファイルは、サポートされているすべてのプロパティーとそのデフォルト値を文書化します。

注 : IBM WebSphere MQ インストール環境を将来のフィックスパックにアップグレードすると、このサンプル構成ファイルは上書きされます。 そのため、アプリケーションで使用できるようにサンプル構成ファイルをコピーしておくことをお勧めします。

IBM WebSphere MQ classes for Java 構成ファイルの名前と場所を選ぶことができます。 アプリケーションを開始するときに、以下のフォーマットで **java** コマンドを使用してください。

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

コマンド中、*config_file_url* は IBM WebSphere MQ classes for Java 構成ファイルの名前と場所を指定する Uniform Resource Locator (URL) です。 以下のタイプの URL がサポートされます。`http`、`file`、`ftp`、および `jar`。

以下の例は、**java** コマンドを示しています。

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

このコマンドは、 IBM WebSphere MQ classes for Java 構成ファイルを、ローカル Windows システム上のファイル `D:\mydir\mqjava.config` として識別します。

IBM WebSphere MQ classes for Java 構成ファイルは、アプリケーションとキュー・マネージャーまたはブローカーとの間のサポートされているいずれのトランスポートとも、共に使用することができます。

IBM WebSphere MQ classes for Java 構成ファイルで指定されたプロパティーの指定変更

IBM WebSphere MQ MQI client 構成ファイルは、IBM WebSphere MQ classes for Java を構成するために使用されるプロパティーを指定することもできます。 ただし、IBM WebSphere MQ MQI client 構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

必要に応じて、IBM WebSphere MQ classes for Java 構成ファイル内のプロパティーとして指定することにより、 IBM WebSphere MQ MQI client 構成ファイル内の任意の属性をオーバーライドできます。 IBM WebSphere MQ MQI client 構成ファイル中の属性を指定変更するには、IBM WebSphere MQ classes for Java 構成ファイルで次のフォーマットのエントリーを使用します。

```
com.ibm.mq.cfg.stanza.propName=propValue
```

エントリー中の変数には、以下の意味があります。

stanza
属性を含んでいる IBM WebSphere MQ MQI client 構成ファイル中のスタンザの名前。

propName
IBM WebSphere MQ MQI client 構成ファイルで指定されている属性の名前。

propValue
IBM WebSphere MQ MQI client 構成ファイルに指定されている属性の値をオーバーライドするプロパティーの値。

あるいは、 **java** コマンドでシステム・プロパティーとしてプロパティーを指定することにより、 IBM WebSphere MQ MQI client 構成ファイル内の属性をオーバーライドすることができます。 プロパティーをシステム・プロパティーとして指定するには、前述のフォーマットを使用してください。

IBM WebSphere MQ classes for Java に関連するのは、 IBM WebSphere MQ MQI client 構成ファイル内の以下の属性のみです。 他の属性を指定または指定変更しても、効果はありません。 特に、クライアント構成ファイルの CHANNELS スタンザ の `ChannelDefinitionFile` および `ChannelDefinitionDirectory` は使用されないことに注意してください。 IBM WebSphere MQ classes for Java で CCDT を使用する方法について詳しくは、 678 ページの『IBM WebSphere MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』 を参照してください。

表 *86.* クライアント構成ファイルのどのスタンザにどの属性が含まれているか

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64

表 86. クライアント構成ファイルのどのスタンザにどの属性が含まれているか (続き)	
スタンザ	**属性**
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

IBM WebSphere MQ MQI client 構成について詳しくは、構成ファイルを使用したクライアントの構成を参照してください。

関連タスク

IBM WebSphere MQ classes for Java アプリケーションのトレース

Java 標準環境トレース・スタンザ

Java 標準環境トレース設定スタンザを使用して、 IBM WebSphere MQ classes for Java トレース機能を構成できます。

com.ibm.msg.client.commonservices.trace.outputName = traceOutputName

traceOutputName はトレース出力の送信先のディレクトリーおよびファイル名です。

トレース・ファイルのデフォルト名は、アプリケーションによって使用されている IBM WebSphere MQ classes for Java のバージョンによって異なります。

- IBM WebSphere MQ classes for Java for Version 7.5.0, Fix Pack 8 以前の場合、 *traceOutputName* はデフォルトで、現行作業ディレクトリー内の `mqjms_%PID%.trc` という名前のファイルになります。
- V 7.5.0.9 Version 7.5.0, Fix Pack 9 の IBM WebSphere MQ classes for Java からは、 *traceOutputName* はデフォルトで、現行作業ディレクトリー内の `mqjava_%PID%.trc` という名前のファイルになります。

%PID% は現在のプロセス ID です。 プロセス ID が使用できない場合は、乱数が生成され、接頭部に `f` の文字が付けられます。 指定するファイル名にプロセス ID を含めるには、ストリング `%PID%` を使用します。

代替ディレクトリーを指定する場合、そのディレクトリーが存在していなければならず、また、そのディレクトリーへの書き込み権限が必要です。 書き込み権限がないと、トレース出力は `System.err` に書き込まれます。

com.ibm.msg.client.commonservices.trace.include = includeList

includeList は、トレースされるパッケージおよびクラスのリスト、または特殊値 `ALL` または `NONE` です。

パッケージ名またはクラス名はセミコロン (`;`) で区切ります。 **includeList** はデフォルトで `ALL` に設定され、 IBM WebSphere MQ classes for Java 内のすべてのパッケージとクラスをトレースします。

注 : パッケージを含めた後、そのパッケージのサブパッケージを除外することができます。 例えば、パッケージ `a.b` は含め、パッケージ `a.b.x` は除外する場合、`a.b.y` および `a.b.z` 内のものはすべてトレースに含まれますが、`a.b.x` 内のものと `a.b.x.1` 内のものは、どちらも除外されます。

com.ibm.msg.client.commonservices.trace.exclude = excludeList
excludeList は、トレースされないパッケージおよびクラスのリスト、または特殊値 ALL または NONE です。

パッケージ名またはクラス名はセミコロン (;) で区切ります。 **excludeList** はデフォルトで NONE に設定されるため、IBM WebSphere MQ classes for Java のパッケージおよびクラスはトレース対象から除外されません。

注: パッケージを除外した後、そのパッケージのサブパッケージを含めることができます。 例えば、パッケージ a.b は除外し、パッケージ a.b.x は含める場合、a.b.x および a.b.x.1 内のものはすべてトレースに含まれますが、a.b.y 内のものと a.b.z 内のものは、どちらも除外されます。

両方 (包含と除外) を指定した場合、同じレベルのパッケージまたはクラスはすべて含められます。

com.ibm.msg.client.commonservices.trace.maxBytes = maxArrayBytes
maxArrayBytes は、任意のバイト配列からトレースされる最大バイト数です。

maxArrayBytes が正整数に設定されると、トレース・ファイルに書き出されるバイト配列内のバイト数が制限されます。 *maxArrayBytes* の書き出し後に、バイト配列が切り捨てられます。
maxArrayBytes を設定すると、結果として生成されるトレース・ファイルのサイズが削減され、トレースがアプリケーションのパフォーマンスに与える影響が低減されます。

このプロパティーの値 0 は、どのバイト配列の内容もトレース・ファイルに送信されないことを意味します。

デフォルト値は -1 です。これは、トレース・ファイルに送信されるバイト配列内のバイト数の制限を除去します。

com.ibm.msg.client.commonservices.trace.limit = maxTraceBytes
maxTraceBytes は、トレース出力ファイルに書き込まれる最大バイト数です。

maxTraceBytes は **traceCycles** と連携して機能します。 書き込まれたトレースのバイト数が制限に近い場合、ファイルが閉じられ、新しいトレース出力ファイルが開始されます。

値 0 は、トレース出力ファイルの長さがゼロであることを意味します。 デフォルト値は -1 です。これは、トレース出力ファイルに書き込まれるデータの量が無制限であることを意味します。

com.ibm.msg.client.commonservices.trace.count = traceCycles
traceCycles は、循環するトレース出力ファイルの数です。

現行のトレース出力ファイルが **maxTraceBytes** で指定された制限に達すると、ファイルが閉じます。以降のトレース出力は、順序で次にあるトレース出力ファイルに書き込まれます。 各トレース出力ファイルは、ファイル名に付加される数値の接尾部によって区別されます。 現行または最新のトレース出力ファイルには接尾部 .trc.0 があり、次に新しいトレース出力ファイルは .trc.1 で終わり、以下同様となります。 それより古いトレース・ファイルは、制限に達するまで、同じ番号付けパターンに従います。

traceCycles のデフォルト値は 1 です。 **traceCycles** が 1 の場合、現行のトレース出力ファイルが最大サイズに達すると、ファイルはクローズされ、削除されます。 同じ名前の新しいトレース出力ファイルが開始されます。 したがって、トレース出力ファイルは一度に 1 つだけ存在することになります。

com.ibm.msg.client.commonservices.trace.parameter = traceParameters
traceParameters は、メソッド・パラメーターおよび戻り値をトレースに含めるかどうかを制御します。

traceParameters はデフォルトで TRUE に設定されます。 **traceParameters** が FALSE に設定されると、メソッド・シグニチャーのみがトレースされます。

com.ibm.msg.client.commonservices.trace.compress = compressedTrace
compressedTrace を TRUE に設定して、トレース出力を圧縮します。

compressedTrace のデフォルト値は FALSE です。

compressedTrace が TRUE に設定されると、トレース出力が圧縮されます。 デフォルトのトレース出力ファイル名には、`.trz` という拡張子が付いています。 圧縮をデフォルト値の FALSE に設定すると、ファイルの拡張子は `.trc` となります。これは、非圧縮であることを示します。 ただし、トレース出力のファイル名が **traceOutputName** で指定されている場合には、その名前が代わりに使用され、ファイルに接尾部は適用されません。

圧縮されたトレース出力は、圧縮されていないものよりサイズが小さくなります。 入出力が少なくなるため、圧縮されていないトレースよりも、圧縮されたトレースのほうが書き込み速度が速くなります。 圧縮されたトレースでは、圧縮されていないトレースよりも、IBM WebSphere MQ classes for Java のパフォーマンスに与える影響が少なくなります。

maxTraceBytes および **traceCycles** が設定されると、複数のフラット・ファイルの代わりに、複数の圧縮されたトレース・ファイルが作成されます。

制御されない形で IBM WebSphere MQ classes for Java が終了した場合、圧縮されたトレース・ファイルが無効である可能性があります。 このため、トレースの圧縮は、必ず IBM WebSphere MQ classes for Java が制御された形で終了する場合にのみ使用してください。 調査中の問題が原因で JVM 自体が予期せず停止することがない場合にのみ、トレース圧縮を使用してください。 `System.Halt()` シャットダウンまたは異常終了、無制御の JVM 終了につながる可能性のある問題を診断しているときは、トレースの圧縮を使用しないでください。

com.ibm.msg.client.commonservices.trace.level = traceLevel

traceLevel は、トレースのフィルター・レベルを指定します。 以下は、定義済みのトレース・レベルです。

表 *87.* 各トレース・レベルでのトレース対象

値	トレース対象
0	トレースはオフになります
1	例外
3	例外 警告
6	例外 警告 通知トレース・ポイント
8	例外 警告 通知トレース・ポイント メソッドの入り口と出口
9	例外 警告 通知トレース・ポイント メソッドの入り口と出口 IBM WebSphere MQ classes for Java とキュー・マネージャーの間で送信されるデータ。

注 : IBM サポートからの指示がない限り、常に値 9 を使用してください。

IBM WebSphere MQ classes for Java とソフトウェア管理ツール

Apache Maven などのソフトウェア管理ツールを IBM WebSphere MQ classes for Java で使用できます。

多くの大規模開発会社がこれらのツールを使用して、サード・パーティー・ライブラリーのリポジトリーを集中管理しています。

IBM WebSphere MQ classes for Java は、いくつかの JAR ファイルで構成されています。 この API を使用して Java 言語アプリケーションを開発する場合は、アプリケーションを開発するマシンに IBM WebSphere MQ Server、 IBM WebSphere MQ Client、または IBM WebSphere MQ Client SupportPac のいずれかをインストールする必要があります。

ソフトウェア管理ツールを使用し、IBM WebSphere MQ classes for Java を構成する JAR ファイルを集中管理リポジトリーに追加する場合は、以下の点を守る必要があります。

- リポジトリーまたはコンテナーは、社内の開発者だけが使用できるようにしなければなりません。 社外に分散させることは許可されません。
- リポジトリーには、単一の IBM WebSphere MQ リリースまたはフィックスパックからの整合した完全な JAR ファイル・セットを入れる必要があります。
- IBM サポートが提供するメンテナンスでリポジトリーを更新する必要があります。

IBM WebSphere MQ Version 7.5 の場合は、以下の JAR ファイルをリポジトリーにインストールする必要があります。

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- connector.jar

IBM WebSphere MQ アプリケーションのインストール後のセットアップ

IBM WebSphere MQ をインストールした後に、独自のアプリケーションを実行するためにインストール環境を構成できます。

ご使用の環境についてのより新しい情報または特定の情報を入手するために、IBM WebSphere MQ README ファイルを必ず確認してください。

IBM WebSphere MQ classes for Java アプリケーションをバインディング・モードで実行する前に、「構成」の説明に従って IBM WebSphere MQ を構成したことを確認してください。

WebSphere MQ classes for Java からのクライアント接続を受け入れるためのキュー・マネージャーの構成

クライアントからの着信接続要求を受け入れるようにキュー・マネージャーを構成するには、サーバー接続チャネルの使用を定義して許可し、リスナー・プログラムを開始します。

詳しくは、108 ページの『サンプル・プログラムの作成と実行』を参照してください。

Java セキュリティー・マネージャーでの *WebSphere MQ classes for Java* アプリケーションの実行

WebSphere MQ classes for Java は、Java セキュリティー・マネージャーを使用可能にして実行できます。Security Manager を使用可能にしてアプリケーションを正常に実行するには、適切なポリシー定義ファイルを使用して Java 仮想マシン (JVM) を構成する必要があります。

これを行う最も簡単な方法は、JRE に付属しているポリシー・ファイルを変更する方法です。 大部分のシステムでは、このファイルは、JRE ディレクトリーの下の相対パス `lib/security/java.policy` に格納されています。 ポリシー・ファイルは好みのエディターを使用して編集することも、JRE で提供されている `policytool` プログラムを使用して編集することもできます。

com.ibm.mq.jmqi.jar ファイルに権限を付与して、以下を実行できるようにする必要があります。

- (クライアント・モードで) ソケットを作成する。
- (バインディング・モードで) 固有のライブラリーをロードする。
- 環境から種々のプロパティーを読み取る。

Java セキュリティー・マネージャーの下で実行する場合は、WebSphere MQ classes for Java でシステム・プロパティー os.name が使用可能でなければなりません。

以下に、WebSphere MQ classes for Java をデフォルトのセキュリティー・マネージャーの下で正常に実行できるようにするポリシー・ファイル・エントリーの例を示します。

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase
"file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.commonservices.jar" {
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.mq.commonservices", "read";
  permission java.util.logging.LoggingPermission "control";
  //For access to the trace properties file.
  permission java.io.FilePermission "/tmp/trace.properties", "read";
  //For access to the trace output files.
  permission java.io.FilePermission "/tmp/*", "read,write";
};
```

注:

- *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。
- このポリシー・ファイルの例では、 WebSphere MQ classes for Java をセキュリティー・マネージャーの下で正しく動作させることができますが、アプリケーションが動作する前に、独自のコードを正しく実行できるようにする必要がある場合があります。
- WebSphere MQ classes for Java がアプリケーションの Java アーカイブ (JAR) ファイルにアクセスできるようにするには、最初の `grant` ステートメントに以下の許可を追加します。

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- ポリシー構成ファイルでこれらの `grant` ステートメントを使用するには、 WebSphere MQ classes for Java をインストールした場所、およびアプリケーションを保管する場所に応じて、パス名を変更する必要がある場合があります。
- WebSphere MQ classes for Java に付属のサンプル・コードは、特にセキュリティー・マネージャーで使用できるようになっていません。ただし、IVT テストは、このポリシー・ファイルとデフォルトのセキュリティー・マネージャーを適切に使用して実行されます。

IBM WebSphere MQ classes for Java インストール済み環境の検査

インストール検査プログラム MQIVP は、IBM WebSphere MQ classes for Java に付属しています。 このプログラムを使用して、IBM WebSphere MQ classes for Java の接続モードをすべてテストできます。

このプログラムでは、いくつかの選択項目とその他のデータについてプロンプトが表示され、検査対象の接続モードを判別します。 インストールを検査するには、以下の手順に従ってください。

1. クライアント・モードでプログラムを実行する場合は、108 ページの『サンプル・プログラムの作成と実行』の説明に従ってキュー・マネージャーを構成します。 使用するキューは SYSTEM.DEFAULT.LOCAL.QUEUE。
2. プログラムをクライアント・モードで実行する場合は、658 ページの『WebSphere MQ classes for Java の使用』も参照してください。

   プログラムを実行するシステムで、この手順の残りのステップを実行します。
3. 663 ページの『 WebSphere MQ classes for Java に関連する環境変数』の指示に従って CLASSPATH 環境変数を更新したことを確認します。
4. ディレクトリーを MQ_INSTALLATION_PATH/mqm/VRM/java/samples/wmqjava に変更します。ここで、MQ_INSTALLATION_PATH は IBM WebSphere MQ インストール済み環境へのパスであり、 VRM は製品のバージョン、リリース、およびモディフィケーション番号です。 その後、コマンド・プロンプトに次を入力します。

   ```
   java -Djava.library.path=library_path MQIVP
   ```

   ここで、 *library_path* は、 Java ライブラリーの IBM WebSphere MQ クラスへのパスです (WebSphere MQ classes for Java ライブラリー を参照)。

   (1) のマークが付いたプロンプトでは、次のようにします。

   - TCP/IP 接続を使用する場合は、IBM WebSphere MQ サーバーのホスト名を入力します。
   - ネイティブ接続 (バインディング・モード) を使用する場合は、このフィールドをブランクのままにします (名前は入力しないでください)。

   このプログラムは、以下の処理を実行します。

   - 1. キュー・マネージャーに接続します。
   - 2. キュー SYSTEM.DEFAULT.LOCAL.QUEUE を開き、キューにメッセージを書き込み、キューからメッセージを取得し、キューを閉じます。
   - 3. キュー・マネージャーへの接続を切断します。
   - 4. 操作が正常に実行された場合にメッセージを返します。

以下は、表示されるプロンプトと応答の例です。 実際のプロンプトと応答は、使用している IBM WebSphere MQ ネットワークによって異なります。

```
Please enter the IP address of the MQ server              : ipaddress(1)
Please enter the port to connect to                       : (1414)(2)
Please enter the server connection channel name            : channelname(2)
Please enter the queue manager name                       : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

注 :

1. サーバー接続を選択した場合は、(2) のマークが付いたプロンプトは表示されません。

IBM WebSphere MQ の問題の解決

最初に、インストール検査プログラムを実行します。 トレース機能も使用する必要がある場合があります。

プログラムが正常に完了しない場合は、インストール検査プログラムを実行し、診断メッセージに示されるアドバイスに従ってください。 このプログラムについては、672 ページの『IBM WebSphere MQ classes for Java インストール済み環境の検査』で説明されています。

問題が解決せず、IBM サービスに連絡する必要があるときは、トレース機能をオンにするようお願いする場合があります。 以下の例のようにこれを行ってください。

MQIVP プログラムをトレースするには、以下のようにします。

- *com.ibm.mq.commonservices* プロパティー・ファイルを作成します (com.ibm.mq.commonservices の使用 を参照してください)。
- 次のコマンドを入力します。

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path=library_path MQIVP -trace
```

ここで、

- *commonservices_properties_file* は、*com.ibm.mq.commonservices* プロパティー・ファイルへのパス (ファイル名を含む) です。
- *library_path* は、 WebSphere MQ classes for Java ライブラリーへのパスです (WebSphere MQ classes for Java ライブラリー を参照)。

トレースの使用法について詳しくは、IBM WebSphere MQ classes for Java アプリケーションのトレースを参照してください。

プログラマー向けの概要

このトピックのコレクションには、プログラマー向けの情報を記載します。

プログラムの作成方法についての詳細は、674 ページの『 WebSphere MQ classes for Java アプリケーションの作成』を参照してください。

WebSphere MQ classes for Java インターフェース

プロシージャー型 WebSphere MQ アプリケーション・プログラミング・インターフェースは、オブジェクトに対してアクションを実行する動詞を使用します。 Java プログラミング・インターフェースは、メソッドを呼び出すことによって操作するオブジェクトを使用します。

プロシージャー型 WebSphere MQ アプリケーション・プログラミング・インターフェースは、以下のような動詞によって構築されています。

```
       MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
       MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

これらの動詞はすべて、操作する WebSphere MQ オブジェクトのハンドルをパラメーターとして 使用します。 Java はオブジェクト指向であるため、Java プログラミング・インターフェースはこのラウンドを行います。 ユーザーのプログラムは、一連の WebSphere MQ オブジェクトで構成されています。 これらのオブジェクトは、メソッドを呼び出すことによって操作します。

手続き型のインターフェースを使用する場合は、MQDISC (Hconn, CompCode, Reason) の呼び出しを用いてキュー・マネージャーから切断します。*Hconn* はキュー・マネージャーに対するハンドルの 1 つです。

Java インターフェースでは、キュー・マネージャーはクラス MQQueueManager のオブジェクトによって表されます。 キュー・マネージャーからの切断は、そのクラスで disconnect() メソッドを呼び出すことによって行われます。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
```

```
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

WebSphere MQ classes for Java アプリケーションの作成

この一連のトピックでは、 WebSphere MQ システムと対話する Java アプリケーションの作成に役立つ情報を提供します。

WebSphere MQ classes for Java を使用して WebSphere MQ キューにアクセスするには、 WebSphere MQ キューにメッセージを書き込んだり、そこからメッセージを取得したりする呼び出しを含む Java アプリケーションを作成します。 個々のクラスについて詳しくは、 WebSphere MQ classes for Java を参照してください。

注 : WebSphere MQ classes for Java では、自動クライアント再接続はサポートされていません。

WebSphere MQ classes for Java の接続モード

WebSphere MQ classes for Java のプログラミング方法には、使用する接続モードに依存するものがあります。

クライアント接続を使用する場合は、IBM WebSphere MQ MQI client との相違がいくつかありますが、概念的には似ています。 バインディング・モードを使用する場合は、ファスト・パス・バインディングの使用、および MQBEGIN コマンドの発行が可能です。 MQEnvironment クラスで変数を設定することで、使用するモードを指定します。

WebSphere MQ classes for Java クライアント接続

WebSphere MQ classes for Java をクライアントとして使用する場合、これは IBM WebSphere MQ MQI client に似ていますが、いくつかの違いがあります。

WebSphere MQ classes for Java をクライアントとして使用するためにプログラミングする場合は、以下の相違点に注意してください。

- TCP/IP のみをサポートします。
- 始動時にどの WebSphere MQ 環境変数も読み取りません。
- チャネル定義および環境変数に格納される情報は、Environment と呼ばれるクラスに格納できます。 または、この情報を、接続の際にパラメーターとして渡すことができます。
- エラー条件と例外条件を `MQException` クラスに指定されたログに書き込みます。 デフォルトのエラー宛先は Java コンソールです。
- WebSphere MQ クライアント構成ファイル中の属性のうち、WebSphere MQ classes for Java に関係するのは以下のものだけです。 他の属性を指定しても、効果はありません。

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage

スタンザ	属性
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

- 文字データの変換が必要なキュー・マネージャーに接続したとき、キュー・マネージャーで変換を行えない場合には、V7 Java クライアントでその変換を実行できます。 クライアント JVM は、クライアントの CCSID とキュー・マネージャーの CCSID の間の変換をサポートしている必要があります。
- クライアントの自動再接続機能は、WebSphere MQ classes for Java ではサポートされていません。

クライアント・モードで使用する場合、 *WebSphere MQ classes for Java* は MQBEGIN 呼び出しをサポートしません。

サポートされる環境の詳細については、660 ページの『 WebSphere MQ classes for Java の接続オプション』を参照してください。

WebSphere MQ classes for Java バインディング・モード

WebSphere MQ classes for Java のバインディング・モードは、主な 3 つの点でクライアント・モードと異なります。

バインディング・モードで使用すると、WebSphere MQ classes for Java は、ネットワーク経由で通信するのではなく、Java Native Interface (JNI) を使用して、既存のキュー・マネージャー API を直接呼び出します。

デフォルトでは、WebSphere MQ classes for Java をバインディング・モードで使用するアプリケーションは、*ConnectOption* の MQCNO_STANDARD_BINDINGS を使用してキュー・マネージャーに接続します。

WebSphere MQ classes for Java は、以下の *ConnectOptions* をサポートしています。

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING

ConnectOptions の詳細については、208 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』 を参照してください。

バインディング・モードで、キュー・マネージャーによって調整されるグローバル作業単位を開始する MQBEGIN 呼び出しがサポートされるのは、WebSphere MQ for IBM i と WebSphere MQ for z/OS を除く、すべてのプラットフォームです。

`MQEnvironment` クラスによって提供されるパラメーターの大部分はバインディング・モードとは無関係で、無視されます。

サポートされる環境の詳細については、660 ページの『 WebSphere MQ classes for Java の接続オプション』を参照してください。

使用する *WebSphere MQ classes for Java* 接続の定義

使用する接続のタイプは、MQEnvironment クラスでの変数の設定によって決定されます。

以下の 2 つの変数が使用されます。

MQEnvironment.properties
接続タイプは、キー名 `CMQC.TRANSPORT_PROPERTY` に関連付けられた値によって決定されます。 次の値を指定できます。

CMQC.TRANSPORT_MQSERIES_BINDINGS
バインディング・モードで接続します

CMQC.TRANSPORT_MQSERIES_CLIENT
クライアント・モードで接続します

CMQC.TRANSPORT_MQSERIES
接続モードは *hostname* プロパティーの値によって決定されます

MQEnvironment.hostname
この変数の値を次のように設定します。

- クライアント接続の場合、接続先の IBM WebSphere MQ サーバーのホスト名にこの変数の値を設定します。
- バインディング・モードの場合、この変数を設定しないでおくか、または NULL に設定します。

キュー・マネージャーに対する操作

この一連のトピックでは、WebSphere MQ classes for Java を使用してキュー・マネージャーに接続する方法、およびキュー・マネージャーから切断する方法について説明します。

WebSphere MQ classes for Java 用の *WebSphere MQ* 環境のセットアップ

アプリケーションがクライアント・モードでキュー・マネージャーに接続するには、チャネル名、ホスト名、およびポート番号を指定する必要があります。

注 : このトピックの情報は、アプリケーションがクライアント・モードでキュー・マネージャーに接続する場合にのみ該当します。 バインディング・モードで接続する場合は、該当しません。 777 ページの『WebSphere MQ classes for JMS の接続モード』を参照してください。

チャネル名、ホスト名、およびポート番号を指定するには、2 つの方法 (MQEnvironment クラスのフィールドと MQQueueManager オブジェクトのプロパティー) のいずれかを使用することができます。

MQEnvironment クラスのフィールドを設定する場合、それらのフィールドはアプリケーション全体に適用されます (プロパティーのハッシュ・テーブルによりオーバーライドされる場所を除く)。 MQEnvironment でチャネル名およびホスト名を指定するには、次のコードを使用します。

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

これは、**MQSERVER** 環境変数を次のように設定することと同じです。

```
"java.client.channel/TCP/host.domain.com".
```

デフォルトでは、Java クライアントはポート 1414 で WebSphere MQ リスナーに接続しようとします。 別のポートを指定するには、次のコードを使用します。

```
MQEnvironment.port = nnnn;
```

nnnn は、必要なポート番号です。

キュー・マネージャー・オブジェクトの作成時にプロパティーを渡すと、その設定はそのキュー・マネージャーだけに適用されます。 **hostname**、**channel**、およびオプションで **port** というキーと適切な値を持ったエントリーを Hashtable オブジェクト内に作成します。 デフォルトのポート 1414 を使用する場合は、**port** エントリーを省略できます。 プロパティーのハッシュ・テーブルを受け付けるコンストラクターを使用して MQQueueManager オブジェクトを作成します。

アプリケーション名の設定によるキュー・マネージャーへの接続の識別

アプリケーションに名前を設定して、アプリケーションからキュー・マネージャーへの接続を識別することができます。 このアプリケーション名は、 **DISPLAY CONN MQSC/PCF** コマンド (フィールド名は **APPLTAG**) または WebSphere MQ エクスプローラーの「**アプリケーション接続**」画面 (フィールド名は **App name**) に表示されます。

アプリケーション名は 28 文字までに制限されており、それより長い名前は制限内に収まるように切り捨てられます。 アプリケーション名が指定されていない場合は、デフォルトが提供されます。 デフォルト名は起動 (メイン) クラスを基にしますが、この情報が利用できない場合は、テキスト `WebSphere MQ Client for Java` が使用されます。

起動クラスの名前が使用される場合は、必要に応じて、制限内に収まるように、それより前の位置にあるパッケージ名を削除して調整されます。 例えば、起動クラスが `com.example.MainApp` である場合は完全な名前が使用されますが、起動クラスが `com.example.dictionaryAndThesaurus.multilingual.mainApp` である場合は `multilingual.mainApp` が使用されます。これが、有効な長さに収まるクラス名と右端のパッケージ名との最長の組み合わせだからです。

クラス名自体が 28 文字より長い場合は、収まるように切り捨てられます。 例えば、`com.example.mainApplicationForSecondTestCase` は `mainApplicationForSecondTest` となります。

注 : z/OS プラットフォーム上で実行するキュー・マネージャーでは、アプリケーション名の設定をサポートしていません。

MQEnvironment クラスにアプリケーション名を設定するには、次のコードを使用して、**MQConstants.APPNAME_PROPERTY** キーを指定し、名前を MQEnvironment.properties ハッシュ・テーブルに追加します。

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

MQQueueManager コンストラクターに渡されるプロパティー・ハッシュ・テーブルにアプリケーション名を設定するには、**MQConstants.APPNAME_PROPERTY** キーを指定し、 プロパティー・ハッシュ・テーブルに名前を追加します。

WebSphere MQ クライアント構成ファイルで指定されたプロパティーの指定変更

WebSphere MQ クライアント構成ファイルは、 WebSphere MQ classes for Java の構成に使用されるプロパティーを指定することもできます。 ただし、WebSphere MQ MQI クライアント構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

必要に応じて、次のいずれかの方法で WebSphere MQ 構成ファイルの任意の属性を指定変更することができます。 オプションを優先順位の高いものから示します。

- 構成プロパティーの Java システム・プロパティーを設定します。
- MQEnvironment.properties マップのプロパティーを設定する。
- Java5 以降のリリースで、システム環境変数を設定する。

WebSphere MQ クライアント構成ファイル内の以下の属性のみが、 WebSphere MQ classes for Java に関連しています。 他の属性を指定または指定変更しても、効果はありません。

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime

スタンザ	属性
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

WebSphere MQ classes for Java でのキュー・マネージャーへの接続

MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続します。 disconnect() メソッドを呼び出して、キュー・マネージャーから切断します。

ここまでで、次のように MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続できる状態になっています。

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

キュー・マネージャーから切断するには、次のようにキュー・マネージャーで disconnect() メソッドを呼び出します。

```
queueManager.disconnect();
```

disconnect メソッドを呼び出すと、そのキュー・マネージャーからアクセスしたオープン・キューとプロセスはすべてクローズされます。 しかし、使用を終えた時点でこれらのリソースを明示的にクローズするプログラミングの習慣を付けておくことをお勧めします。 これは、関連するオブジェクトに close() メソッドを使って行います。

キュー・マネージャーの commit() メソッドと backout() メソッドは、プロシージャー型インターフェースで使用される MQCMIT 呼び出しと MQBACK 呼び出しに相当するものです。

IBM WebSphere MQ classes for Java を含むクライアント・チャネル定義テーブルの使用

IBM WebSphere MQ classes for Java クライアント・アプリケーションは、クライアント・チャネル定義テーブル (CCDT) に保管されたクライアント接続チャネル定義を使用できます。

`MQEnvironment` クラスで特定のフィールドおよび環境プロパティーを設定するか、プロパティー・ハッシュ・テーブルでそれらを `MQQueueManager` に渡すことによってクライアント接続チャネル定義を作成する代わりに、IBM WebSphere MQ classes for Java クライアント・アプリケーションは、クライアント・チャネル定義テーブルに保管されているクライアント接続チャネル定義を使用できます。 これらの定義は、IBM WebSphere MQ Script (MQSC) コマンドまたは IBM WebSphere MQ Programmable Command Format (PCF) コマンドによって、または IBM WebSphere MQ Explorer を使用して作成されます。

アプリケーションが `MQQueueManager` オブジェクトを作成すると、IBM WebSphere MQ classes for Java クライアントは、クライアント・チャネル定義テーブルで適切なクライアント接続チャネル定義を検索し、そのチャネル定義を使用して MQI チャネルを開始します。 クライアント・チャネル定義テーブルの詳細とその構成方法については、クライアント・チャネル定義テーブルを参照してください。

クライアント・チャネル定義テーブルを使用する場合、アプリケーションはまず URL オブジェクトを作成する必要があります。 URL オブジェクトは、クライアント・チャネル定義テーブルを格納するファイルの名前と場所を識別する URL (Uniform Resource Locator) をカプセル化し、そのファイルへのアクセス方法を指定します。

例えば、クライアント・チャネル定義テーブルがファイル ccdt1.tab に格納されていて、このファイルはアプリケーションが実行されるシステムと同じシステム上に保管されている場合、アプリケーションは以下の方法で URL オブジェクトを作成できます。

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

また、別の例として、ファイル ccdt2.tab にクライアント・チャネル定義テーブルが含まれており、このファイルがアプリケーションの実行されるシステムとは異なるシステムに保管されているとします。 FTP プロトコルを使用してこのファイルにアクセスできる場合、アプリケーションは以下の方法で URL オブジェクトを作成できます。

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

URL オブジェクトを作成したら、アプリケーションは、URL オブジェクトをパラメーターにとるコンストラクターの 1 つを使用して MQQueueManager オブジェクトを作成できます。 以下が例となります。

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

このステートメントにより、 IBM WebSphere MQ classes for Java クライアントは、URL オブジェクト chanTab2 によって識別されるクライアント・チャネル定義テーブルにアクセスし、そのテーブルで適切なクライアント接続チャネル定義を検索してから、そのチャネル定義を使用して MARS と呼ばれるキュー・マネージャーへの MQI チャネルを開始します。

アプリケーションがクライアント・チャネル定義テーブルを使用する場合は、以下の事項が適用されるので、注意してください。

- アプリケーションが、URL オブジェクトをパラメーターにとるコンストラクターを使用して MQQueueManager オブジェクトを作成する場合、MQEnvironment クラスにはフィールドとしても環境プロパティーとしてもチャネル名を設定しないでください。 チャネル名が設定されている場合、 IBM WebSphere MQ classes for Java クライアントは MQException をスローします。 チャネル名を指定するフィールドまたは環境プロパティーは、その値がヌル、空ストリング、ブランク文字のみのストリングのいずれでもない場合に設定されていると見なされます。
- MQQueueManager コンストラクター上の **queueManagerName** パラメーターは、次のいずれかの値をとることができます。
  - キュー・マネージャーの名前。
  - アスタリスク (*) とそれに続くキュー・マネージャー・グループ名。
  - アスタリスク (*)。
  - ヌル、空ストリング、またはブランク文字のみを含むストリング

  これらの値は、メッセージ・キュー・インターフェース (MQI) を使用しているクライアント・アプリケーションによって発行された MQCONN 呼び出しで、**QMgrName** パラメーターに使用できる値と同じです。 これらの値の意味について詳しくは、 194 ページの『Message Queue Interface の概要』を参照してください。

  アプリケーションが接続プーリングを使用する場合は、699 ページの『 WebSphere MQ classes for Java でのデフォルト接続プールの制御』を参照してください。
- IBM WebSphere MQ classes for Java クライアントは、クライアント・チャネル定義テーブルで適切なクライアント接続チャネル定義を検出すると、このチャネル定義から抽出された情報のみを使用して MQI チャネルを開始します。 アプリケーションが MQEnvironment クラスに設定したチャネル関連のフィールドまたは環境プロパティーは無視されます。

  特に、Secure Sockets Layer (SSL) を使用する場合は、以下の点に注意してください。
  - MQI チャネルで SSL が使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義で、 IBM WebSphere MQ classes for Java クライアントによってサポートされる CipherSpec の名前が指定されている場合のみです。

– クライアント・チャネル定義テーブルには、証明書取り消しリスト (CRL) を保持する LDAP (Lightweight Directory Access Protocol) サーバーの場所に関する情報も含まれます。 IBM WebSphere MQ classes for Java クライアントは、CRL を保持する LDAP サーバーへのアクセスにこの情報のみを使用します。
– クライアント・チャネル定義テーブルには、Online Certificate Status Protocol (OCSP) の応答側の場所を含めることもできます。 IBM WebSphere MQ classes for Java は、クライアント・チャネル定義テーブル・ファイル内の OCSP 情報を使用できません。 ただし、OCSP を構成することはできます (Online Certificate Protocol の使用セクションを参照)。

クライアント・チャネル定義テーブルでの SSL の使用の詳細については、MQI チャネルで SSL を使用するよう指定するを参照してください。

チャネル出口を使用する場合は、以下の点にも注意してください。

– MQI チャネルで使用されるのは、他のメソッドを使用して指定されたチャネル出口とデータに優先して、クライアント・チャネル定義テーブルから抽出されたチャネル定義によって指定されているチャネル出口とそれに関連するユーザー・データです。
– クライアント・チャネル定義テーブルから抽出されたチャネル定義は、 Java、C、または C++ で作成されたチャネル出口を指定できます。 Java でチャネル出口を作成する方法について詳しくは、 693 ページの『 WebSphere MQ classes for Java でのチャネル出口の作成』を参照してください。 チャネル出口を他の言語で作成する方法について詳しくは、696 ページの『 WebSphere MQ classes for Java での Java で作成されていないチャネル出口の使用』を参照してください。

IBM WebSphere MQ classes for Java クライアント接続を受け入れるためのポート範囲の指定

2 つの方法のいずれかでアプリケーションがバインドできるポート、またはポートの範囲を指定することができます。

IBM WebSphere MQ classes for Java アプリケーションがクライアント・モードで IBM WebSphere MQ キュー・マネージャーに接続しようとした場合、ファイアウォールは指定されたポートまたはポートの範囲から発生する接続のみを許可します。 この場合、アプリケーションがバインドできるポート、またはポートの範囲を指定することができます。 以下の方法でポートを指定できます。

- MQEnvironment クラスの localAddressSetting フィールドを設定できます。 以下が例となります。

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- 環境プロパティー CMQC.LOCAL_ADDRESS_PROPERTY を設定できます。 以下が例となります。

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- MQQueueManager オブジェクトを構成する際には、値が "192.0.2.0(2000,3000)" である LOCAL_ADDRESS_PROPERTY を含むプロパティーのハッシュ・テーブルをパスすることができます。

これらの例では、アプリケーションがキュー・マネージャーに後で接続すると、アプリケーションはローカル IP アドレスおよび 192.0.2.0(2000) から 192.0.2.0(3000) の範囲にあるポート番号にバインドされます。

複数のネットワーク・インターフェースを持つシステムでは、 localAddressSetting フィールドまたは環境プロパティー CMQC.LOCAL_ADDRESS_PROPERTY を 使用して、どのネットワーク・インターフェースを接続に使用するかを指定します。

ポートの範囲を制限すると、接続エラーが起きる可能性があります。 エラーが発生すると、IBM WebSphere MQ 理由コード MQRC_Q_MGR_NOT_AVAILABLE と次のメッセージを含む MQException がスローされます。

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

指定された範囲内のすべてのポートが使用中である場合、または指定された IP アドレス、ホスト名、ポート番号が正しくない場合 (負のポート番号など) は、エラーが発生する可能性があります。

WebSphere MQ classes for Java でのキュー、トピック、およびプロセスへのアクセス

キュー、トピック、およびプロセスへアクセスするには、MQQueueManager クラスのメソッドを使用します。 MQOD (オブジェクト記述子構造体) は、これらのメソッドのパラメーターに縮小されます。

キュー

キューをオープンするには、MQQueueManager クラスの accessQueue メソッドを使用できます。 例えば、queueManager というキュー・マネージャーの場合、以下のコードを使用します。

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

accessQueue メソッドは、クラス MQQueue の新規オブジェクトを戻します。

キューの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
queue.close();
```

MQQueue コンストラクターを使用してキューを作成することもできます。 各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、accessQueue メソッドの場合とまったく同じものです。 以下に例を示します。

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

キューの作成時には、いくつかのオプションを指定できます。 これらの詳細については、Class.com.ibm.mq.MQQueue を参照してください。 この方法でキュー・オブジェクトを構成すると、MQQueue の独自のサブクラスを作成できます。

トピック

同じように、MQQueueManager クラスの accessTopic メソッドを使用してトピックをオープンすることができます。 例えば、queueManager というキュー・マネージャーの場合は、以下のコードを使用してサブスクライバーおよびパブリッシャーを作成します。

```
MQTopic subscriber =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

```
MQTopic publisher =
      queueManager.accessTopic("TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_PUBLICATION, CMQC.MQOO_OUTPUT);
```

トピックの使用が完了したら、close() メソッドを使用してそのトピックをクローズしてください。

MQTopic コンストラクターを使用してトピックを作成することもできます。 キュー・マネージャー・パラメーターが追加されていることを除き、各パラメーターは accessTopic メソッドのものとまったく同じです。 以下に例を示します。

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

トピックの作成時には、いくつかのオプションを指定できます。 これらの詳細については、クラス com.ibm.mq.MQTopic を参照してください。 この方法でトピック・オブジェクトを構成すると、MQTopic のユーザー固有のサブクラスを作成することもできます。

トピックはパブリケーションかサブスクリプションのいずれかに対してオープンしなければなりません。 MQQueueManager クラスには 8 個の accessTopic メソッドがあり、 Topic クラスには 8 個のコンストラクターがあります。 それぞれ、4 個には **destination** パラメーターがあり、4 個には **subscriptionName** パラメーターがあります (内 2 個ずつには両方のパラメーターがあります)。 これらはトピックをサブスクリプションに対してオープンするためだけに使用できます。 残りの 2 個のメソッドには **openAs** パラメーターがあり、**openAs** パラメーターの値に応じてパブリケーションとサブスクリプションのいずれかに対してトピックをオープンできます。

永続サブスクライバーとしてトピックを作成するには、サブスクリプション名を受け付ける MQQueueManager クラスの accessTopic メソッドか MQTopic コンストラクターを使用し、どちらの場合も CMQC.MQSO_DURABLE オプションを設定します。

Processes

プロセスにアクセスするには、MQQueueManager の accessProcess メソッドを使用します。 例えば、queueManager というキュー・マネージャーの場合は、以下のコードを使用して MQProcess オブジェクトを作成します。

```
MQProcess process =
      queueManager.accessProcess("PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

プロセスにアクセスするには、MQQueueManager の accessProcess メソッドを使用します。

accessProcess メソッドは、クラス MQProcess の新規オブジェクトを戻します。

プロセス・オブジェクトの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
process.close();
```

MQProcess コンストラクターを使用してプロセスを作成することもできます。 各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、accessProcess メソッドの場合とまったく同じものです。 以下に例を示します。

```
 MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

この方法でプロセス・オブジェクトを構成すると、MQProcess のユーザー固有のサブクラスを作成することもできます。

WebSphere MQ classes for Java でのメッセージの処理

メッセージは MQMessage クラスで表されます。 メッセージの書き込みと取得は、MQQueue および MQTopic のサブクラスを持つ MQDestination クラスのメソッドを使用して行います。

MQDestination クラスの put() メソッドを使用して、メッセージをキューまたはトピックに書き込みます。 MQDestination クラスの get() メソッドを使用してキューまたはトピックからメッセージを取得します。 MQPUT および MQGET でバイトの配列の書き込みと取得を行うプロシージャー型インターフェースとは異なり、Java プログラミング言語では MQMessage クラスのインスタンスの書き込みと取得を行います。 MQMessage クラスは、実際のメッセージ・データが入っているデータ・バッファーを、そのメッセージを記述しているすべての MQMD (メッセージ記述子) パラメーターおよびメッセージ・プロパティーと共にカプセル化します。

新規メッセージを作成するには、MQMessage クラスの新規インスタンスを作成し、writeXXX メソッドを使用してデータをメッセージ・バッファーに書き込みます。

新規メッセージ・インスタンスが作成されると、MQMD の初期値および言語ごとの宣言 で定義されているように、すべての MQMD パラメーターが自動的にデフォルト値に設定されます。 また、MQDestination の

put() メソッドは、パラメーターとして MQPutMessageOptions クラスのインスタンスを取ります。 このクラスは MQPMO 構造体を表します。 次の例では、メッセージを作成して、キューに書き込んでいます。

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message!
queue.put(myMessage,pmo);
```

MQDestination の get() メソッドは、MQMessage の新しいインスタンスを戻します。 これはキューから取られた直後のメッセージを表します。 また、このメソッドは MQGetMessageOptions クラスのインスタンスをパラメーターとして使用します。 このクラスは MQGMO 構造体を表します。

get() メソッドは自動的にその内部バッファーのサイズを着信メッセージが収まるように調整するので、最大メッセージ・サイズの指定は不要です。 戻されたメッセージ中のデータにアクセスするには、MQMessage クラスの readXXX メソッドを使用します。

次の例は、メッセージをキューから読み取る方法を示しています。

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

encoding メンバー変数を設定することにより、読み取りメソッドと書き込みメソッドで使用する数字形式を変更できます。

characterSet メンバー変数を設定することにより、読み取りと書き込みのストリングで使用する文字セットを変更できます。

詳細については、1075 ページの『MQMessage クラス』を参照してください。

注 : MQMessage の writeUTF() メソッドでは、含まれている Unicode バイト数に加えてストリングの長さも自動的にエンコードされます。 メッセージが別の Java プログラムによって (readUTF() を使用して) 読み取られる場合、これがストリング情報を送信する最も簡単な方法です。

WebSphere MQ classes for Java での非永続メッセージのパフォーマンスの向上

メッセージをブラウズしたり、クライアント・アプリケーションから非永続メッセージをコンシュームしたりする際に、パフォーマンスの改善のために先読み を使用することができます。 MQGET または非同期コンシュームを使用するクライアント・アプリケーションは、メッセージをブラウズする際に、または非永続メッセージをコンシュームする際に、パフォーマンスの改善による益を得ます。

先読み機能の一般情報については、関連トピックを参照してください。

WebSphere MQ classes for Java において、MQQueue または MQTopic オブジェクトの CMQC.MQSO_READ_AHEAD プロパティーおよび CMQC.MQSO_NO_READ_AHEAD プロパティーを使用して、メッセージ・コンシューマーおよびキュー・ブラウザーがそのオブジェクトに対して先読みができるかどうかを決定します。

WebSphere MQ classes for Java を使用したメッセージの非同期書き込み

メッセージを非同期に書き込むには、MQPMO_ASYNC_RESPONSE を設定します。

MQDestination クラスの put() メソッドを使用して、メッセージをキューまたはトピックに書き込みます。メッセージを非同期に書き込む、つまりキュー・マネージャーからの応答を待たずに操作が完了できるようにするには、MQPutMessageOptions の options フィールドで MQPMO_ASYNC_RESPONSE を設定します。 非同期書き込みが成功したか失敗したかを判断するには、MQQueueManager.getAsyncStatus 呼び出しを使用します。

WebSphere MQ classes for Java におけるパブリッシュ/サブスクライブ

WebSphere MQ classes for Java では、トピックは MQTopic クラスによって表され、それに対するパブリッシュは MQTopic.put() メソッドを使用して行われます。

WebSphere MQ パブリッシュ/サブスクライブの一般情報については、 Introduction to WebSphere MQ パブリッシュ/サブスクライブ・メッセージング を参照してください。

WebSphere MQ classes for Java を使用した WebSphere MQ メッセージ・ヘッダーの処理

さまざまなタイプのメッセージ・ヘッダーを表す Java クラスが提供されています。 2 つのヘルパー・クラスも提供されています。

ヘッダー・オブジェクトは MQHeader インターフェースにより記述されます。これはヘッダー・フィールドへのアクセスおよびメッセージ内容の読み書きのための汎用メソッドを提供します。 各ヘッダー・タイプは MQHeader インターフェースを実装する固有のクラスを持ち、それぞれのフィールドの getter メソッドおよび setter メソッドを追加します。 例えば、MQRFH2 ヘッダー・タイプは MQRFH2 クラスにより表され、MQDLH ヘッダー・タイプは MQDLH クラスにより表される、という具合になります。 ヘッダー・クラスは必要なデータ変換を自動的に実行し、指定された任意の数値エンコードまたは文字セット (CCSID) でデータを読み書きできます。

MQHeaderIterator および MQHeaderList の 2 つのヘルパー・クラスは、メッセージ内のヘッダー内容の読み取りとデコード (構文解析) を支援します。

- MQHeaderIterator クラスは、java.util.Iterator のように機能します。 メッセージ内にさらに多くのヘッダーがあれば、next() メソッドは true を戻し、nextHeader() または next() メソッドは次のヘッダー・オブジェクトを戻します。
- MQHeaderList は、java.util.List のように機能します。 MQHeaderIterator のように、これはヘッダー内容を構文解析しますが、特定のヘッダーの検索、新規ヘッダーの追加、既存のヘッダーの除去、ヘッダー・フィールドの更新、およびヘッダー内容のメッセージへの書き戻しもできます。 別の方法として、空の MQHeaderList を作成して、それにヘッダー・インスタンスを取り込み、それをメッセージに一度または繰り返し書き込むことができます。

MQHeaderIterator および MQHeaderList クラスは、MQHeaderRegistry 内の情報を使用して、どの WebSphere MQ ヘッダー・クラスが特定のメッセージ・タイプおよびメッセージ形式と関連しているかを把握します。 MQHeaderRegistry は、現行のすべての WebSphere MQ フォーマットとヘッダー・タイプ、およびその実装クラスについての知識に従って構成され、ユーザー固有のヘッダー・タイプを登録することもできます。

以下の共通に使用される Websphere MQ ヘッダーのサポートも提供されています。

- MQRFH - 規則およびフォーマット・ヘッダー
- MQRFH2 – MQRFH のように、WebSphere Message Broker に属するメッセージ・ブローカーとのメッセージのやり取りに使用されます。 メッセージ・プロパティーを含めるためにも使用されます。
- MQCIH - CICS ブリッジ
- MQDLH - 送達不能ヘッダー
- MQIIH - IMS 情報ヘッダー
- MQRMH - 参照メッセージ・ヘッダー
- MQSAPH - SAP ヘッダー
- MQWIH - 作業情報ヘッダー

- MQXQH - 伝送キュー・ヘッダー
- MQDH - 配布ヘッダー
- MQEPH - カプセル化 PCF ヘッダー

ユーザー固有のヘッダーを表すクラスを定義することもできます。

MQHeaderIterator を使用して RFH2 ヘッダーを取得するには、GetMessageOptions の MQGMO_PROPERTIES_FORCE_MQRFH2 を設定するか、またはキュー・プロパティー PROPCTL を FORCE に設定します。

WebSphere MQ classes for Java を使用したメッセージ内のすべてのヘッダーの印刷

この例では、MQHeaderIterator のインスタンスは、キューから受け取った MQMessage 内のヘッダーを構文解析します。 nextHeader() メソッドから戻された MQHeader オブジェクトは、toString メソッドが呼び出されると、その構造と内容を表示します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeader;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

while (it.hasNext ())
{
    MQHeader header = it.nextHeader ();

    System.out.println ("Header type " + header.type () + ": " + header);
}
```

WebSphere MQ classes for Java を使用したメッセージのヘッダーのスキップオーバー

この例では、MQHeaderIterator の skipHeaders() メソッドは、メッセージ読み取りカーソルを最後のヘッダーの直後に配置します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderIterator;
...
MQMessage message = ... // Message received from a queue.
MQHeaderIterator it = new MQHeaderIterator (message);

it.skipHeaders ();
```

WebSphere MQ classes for Java を使用した送達不能メッセージでの理由コードの検索

この例では、read メソッドは、メッセージからの読み取りを行って、MQDLH オブジェクトへのデータの取り込みを行います。 読み取り操作の後に、メッセージ読み取りカーソルは MQDLH ヘッダー内容の直後に配置されます。

キュー・マネージャーの送達不能キューにあるメッセージには、送達不能ヘッダー (MQDLH) が接頭部に付きます。 これらのメッセージを処理する方法を決定するために (例えばそれらを再試行するか破棄するかを決定する)、送達不能処理アプリケーションは、MQDLH に含まれている理由コードを参照する必要があります。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH ();

dlh.read (message);

System.out.println ("Reason: " + dlh.getReason ());
```

すべてのヘッダー・クラスは、単一ステップでメッセージから直接に自分自身を初期化するための、便利なコンストラクターを備えています。 そのため、この例のコードは以下のように単純化することができます。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQDLH dlh = new MQDLH (message);

System.out.println ("Reason: " + dlh.getReason ());
```

WebSphere MQ classes for Java を使用した、送達不能メッセージからの *MQDLH* の読み取りおよび除去

この例では、送達不能メッセージからのヘッダーの除去に MQDLH が使用されています。

送達不能処理アプリケーションは、理由コードが一時エラーを示している場合は、通常、拒否されたメッセージを再処理依頼します。 メッセージを再処理依頼する前に、MQDLH ヘッダーを除去する必要があります。

この例は、以下のステップを実行します (サンプル・コード内のコメントを参照してください)。

1. MQHeaderList はメッセージ全体を読み取り、メッセージ内で検出された各ヘッダーはリスト内の項目になります。
2. 送達不能メッセージには最初のヘッダーとして MQDLH が含まれるため、これはヘッダー・リスト内の最初の項目で検出できます。 MQDLH は、MQHeaderList の作成時にメッセージから既に取り込まれているので、その読み取りメソッドを呼び出す必要はありません。
3. MQDLH クラスにより提供される getReason() メソッドを使用して、理由コードが抽出されます。
4. 理由コードが検査され、メッセージの再処理依頼が適切であることが示されます。 MQHeaderList remove() メソッドを使用して、MQDLH が除去されます。
5. MQHeaderList は、その残りの内容を新規メッセージ・オブジェクトに書き込みます。 新規メッセージには、MQDLH を除くオリジナル・メッセージのすべてが含まれ、キューに書き込めるようになります。コンストラクターおよび write メソッドへの **true** 引数は、メッセージ本体が MQHeaderList 内に保持され、再度書き出されることを示します。
6. 新規メッセージのメッセージ記述子の format フィールドには、以前には MQDLH フォーマット・フィールドにあった値が含まれることになります。 メッセージ・データは、メッセージ記述子内の数値エンコードおよび CCSID セットと一致します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQDLH;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from the dead-letter queue.
MQHeaderList list = new MQHeaderList (message, true); // Step 1.
MQDLH dlh = (MQDLH) list.get (0); // Step 2.
int reason = dlh.getReason (); // Step 3.
...
list.remove (dlh); // Step 4.

MQMessage newMessage = new MQMessage ();

list.write (newMessage, true); // Step 5.
newMessage.format = list.getFormat (); // Step 6.
```

WebSphere MQ classes for Java を使用したメッセージの内容の印刷

この例では、ヘッダーも含むメッセージの内容を印刷するために MQHeaderList を使用します。

出力には、メッセージの本体に加え、すべてのヘッダー内容のビューも含まれています。 MQHeaderList クラスはすべてのヘッダーを一度の実行でデコードしますが、MQHeaderIterator はそれらを、アプリケーション制御下で一度に 1 つずつステップスルーします。 この技法は、Websphere MQ アプリケーションの作成時の単純なデバッグ・ツールを備えるために使用できます。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ... // Message received from a queue.

System.out.println (new MQHeaderList (message, true));
```

この例は、MQMD クラスを使用して、メッセージ記述子フィールドも印刷します。
com.ibm.mq.headers.MQMD クラスの copyFrom() メソッドは、メッセージ本体の読み取りによってではなく、MQMessage のメッセージ記述子フィールドから、ヘッダー・オブジェクトに取り込みを行います。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQMD;
import com.ibm.mq.headers.MQHeaderList;
...
MQMessage message = ...
MQMD md = new MQMD ();
...
md.copyFrom (message);
System.out.println (md + "\n" + new MQHeaderList (message, true));
```

WebSphere MQ classes for Java を使用したメッセージ内の特定タイプのヘッダーの検索

この例では、MQHeaderList の indexOf(String) メソッドを使用して、メッセージ内の MQRFH2 ヘッダーを (もしあれば) 検索します。

```
import com.ibm.mq.MQMessage;
import com.ibm.mq.headers.MQHeaderList;
import com.ibm.mq.headers.MQRFH2;
...
MQMessage message = ...
MQHeaderList list = new MQHeaderList (message);
int index = list.indexOf ("MQRFH2");

if (index >= 0)
{
    MQRFH2 rfh = (MQRFH2) list.get (index);
    ...
}
```

WebSphere MQ classes for Java を使用した *MQRFH2* ヘッダーの分析

この例は、MQRFH2 クラスを使用して、指定されたフォルダー内の既知のフィールド値にアクセスする方法を示しています。

MQRFH2 クラスは、構造体の固定部分のフィールドだけでなく、NameValueData フィールドで伝送される XML エンコード・フォルダーの内容にもアクセスする多数の方法を提供します。 この例は、指定されたフォルダー内の既知のフィールド値にアクセスする方法を示しています。この例では、jms フォルダーの Rto フィールドにアクセスします。これは、MQ JMS メッセージの応答キュー名を表しています。

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

(特定のフィールドを直接要求する場合とは異なり) MQRFH2 の内容を調べるには、getFolders メソッドを使用して MQRFH2.Element のリストを戻すことができます。これはフォルダーの構造を表すものであり、これにはフィールドや他のフォルダーが含まれていることがあります。 フィールドまたはフォルダーをヌルに設定すると、それが MQRFH2 から除去されます。 この方法で NameValueData フォルダーの内容を操作すると、それに応じて StrucLength フィールドが自動的に更新されます。

WebSphere MQ classes for Java を使用した *MQMessage* オブジェクト以外のバイト・ストリームの読み取りおよび書き込み

以下の例では、データ・ソースが MQMessage オブジェクトではない場合に、ヘッダー・クラスを使用して、WebSphere MQ ヘッダー内容を構文解析および操作します。

データ・ソースが MQMessage オブジェクト以外のものである場合でも、WebSphere MQ ヘッダー内容を構文解析および操作するために、ヘッダー・クラスを使用できます。 すべてのヘッダー・クラスによって実装される MQHeader インターフェースは、メソッド `int read (java.io.DataInput message, int encoding, int characterSet)` および `int write (java.io.DataOutput message, int encoding, int characterSet)` を提供します。 com.ibm.mq.MQMessage クラスは、java.io.DataInput および java.io.DataOutput インターフェースを実装します。 これはつまり、2 つの MQHeader メソッドを使用して、MQMessage の内容を読み書きし、メッセージ記述子で指定されたエンコードと CCSID をオー

バーライドできることを意味します。 これは、エンコードの異なる一連のヘッダーを含むメッセージの場合に役立ちます。

DataInput および DataOutput オブジェクトは、例えばファイルやソケットのストリーム、または JMS メッセージで伝送されるバイト配列などの、他のデータ・ストリームから入手することもできます。 java.io.DataInputStream クラスは DataInput を実装し、java.io.DataOutputStream クラスは DataOutput を実装します。 この例では、WebSphere MQ ヘッダーの内容をバイト配列から読み取ります。

```
import java.io.*;
import com.ibm.mq.headers.*;
...
byte [] bytes = ...
DataInput in = new DataInputStream (new ByteArrayInputStream (bytes));
MQHeaderIterator it = new MQHeaderIterator (in, CMQC.MQENC_NATIVE,
  CMQC.MQCCSI_DEFAULT);
```

先頭が MQHeaderIterator の行は、以下で置き換えることができます。

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

この例は、DataOutputStream を使用してバイト配列に書き込みます。

```
MQHeader header = ... // Could be any header type
ByteArrayOutputStream out = new ByteArrayOutputStream ();

header.write (new DataOutputStream (out), CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
byte [] bytes = out.toByteArray ();
```

この方法でストリームを処理する場合は、encoding および characterSet 引数に正しい値を使用するように注意してください。 ヘッダーの読み取り時には、最初にバイト内容を書き込んだときに使用したエンコードおよび CCSID を指定します。 ヘッダーの書き込み時には、生成したいエンコードおよび CCSID を指定します。 データ変換は、ヘッダー・クラスにより自動的に実行されます。

WebSphere MQ classes for Java を使用した新規ヘッダー・タイプのクラスの作成

WebSphere MQ classes for Java では提供されていないヘッダー・タイプの Java クラスを作成できます。

WebSphere MQ classes for Java で提供されるヘッダー・クラスと同じ方法で使用できる新しいヘッダー・タイプを表す Java クラスを追加するには、MQHeader インターフェースを実装するクラスを作成します。 最も簡単な方法は、com.ibm.mq.headers.impl.Header クラスを拡張することです。 この例では、MQTM ヘッダー構造体を表す完全機能クラスを作成します。 それぞれのフィールドに個別の getter メソッドおよび setter メソッドを追加する必要はありませんが、これはヘッダー・クラスのユーザーには便利なものです。 フィールド名のストリングを取る汎用 getValue メソッドおよび汎用 setValue メソッドは、ヘッダー・タイプで定義されるすべてのフィールドで機能します。 継承された read メソッド、write メソッド、および size メソッドにより、新規ヘッダー・タイプのインスタンスの読み取りおよび書き込みが可能になり、そのフィールド定義に基づいてヘッダー・サイズが正確に計算されます。 タイプ定義は一度だけ作成されますが、このヘッダー・クラスのインスタンスは多数作成されます。 MQHeaderIterator クラスまたは MQHeaderList クラスを使用して新規ヘッダー定義をデコードできるようにするには、MQHeaderRegistry を使用してこれを登録します。 ただし、MQTM ヘッダー・クラスは実際にはこのパッケージで既に提供され、デフォルトのレジストリーで登録されていることに注意してください。

```
import com.ibm.mq.headers.impl.Header;
import com.ibm.mq.headers.impl.HeaderField;
import com.ibm.mq.headers.CMQC;

public class MQTM extends Header {
    final static HeaderType TYPE = new HeaderType ("MQTM");
    final static HeaderField StrucId = TYPE.addMQChar ("StrucId", CMQC.MQTM_STRUC_ID);
    final static HeaderField Version = TYPE.addMQLong ("Version", CMQC.MQTM_VERSION_1);
    final static HeaderField QName = TYPE.addMQChar ("QName", CMQC.MQ_Q_NAME_LENGTH);
    final static HeaderField ProcessName = TYPE.addMQChar ("ProcessName",
      CMQC.MQ_PROCESS_NAME_LENGTH);
    final static HeaderField TriggerData = TYPE.addMQChar ("TriggerData",
      CMQC.MQ_TRIGGER_DATA_LENGTH);
    final static HeaderField ApplType = TYPE.addMQLong ("ApplType");
    final static HeaderField ApplId = TYPE.addMQChar ("ApplId", 256);
    final static HeaderField EnvData = TYPE.addMQChar ("EnvData", 128);
    final static HeaderField UserData = TYPE.addMQChar ("UserData", 128);
```

```
    protected MQTM (HeaderType type){
        super (type);
    }
    public String getStrucId () {
        return getStringValue (StrucId);
    }
    public int getVersion () {
        return getIntValue (Version);
    }
    public String getQName () {
        return getStringValue (QName);
    }
    public void setQName (String value) {
        setStringValue (QName, value);
    }
    // ...Add convenience getters and setters for remaining fields in the same way.
}
```

WebSphere MQ classes for Java を使用した PCF メッセージの処理

PCF 構造化メッセージを作成および解析し、PCF 要求の送信と PCF 応答の収集を容易にするために、Java クラスが提供されています。

クラス PCFMessage および MQCFGR は、PCF パラメーター構造体の配列を表します。 これらは PCF パラメーターを追加および取得するための便利なメソッドを提供します。

PCF パラメーター構造体は、クラス MQCFH、MQCFIN、MQCFIN64、MQCFST、MQCFBS、MQCFIL、MQCFIL64、MQCFSL、および MQCFGR で表されます。 これらは以下の基本操作インターフェースを共有します。

- メッセージ内容の読み取りおよび書き込みのためのメソッド: read ()、write ()、および size ()
- パラメーターの操作のためのメソッド: getValue ()、setValue ()、getParameter ()、その他
- MQMessage 内の PCF の内容を構文解析する、列挙子メソッド .nextParameter ()

PCF フィルター・パラメーターは、フィルター機能を提供する inquire コマンドで使用されます。 これは以下のクラスでカプセル化されます。

- MQCFIF - 整数フィルター
- MQCFSF - ストリング・フィルター
- MQCFBF - バイト・フィルター

PCFAgent と PCFMessageAgent の 2 つのエージェント・クラスが、キュー・マネージャー、コマンド・サーバー・キュー、および関連応答キューへの接続を管理するために提供されています。 PCFMessageAgent は PCFAgent の拡張版であるため、通常はこちらを優先して使用すべきです。 PCFMessageAgent クラスは受け取った MQMessages を変換し、それらを PCFMessage 配列として呼び出し元に戻します。 PCFAgent は MQMessages の配列を戻しますが、これは使用する前に構文解析する必要があります。

WebSphere MQ classes for Java でのメッセージ・プロパティーの処理

WebSphere MQ classes for Java では、メッセージ・ハンドルを処理するための関数呼び出しに相当するものはありません。 メッセージ・ハンドルのプロパティーを設定する、戻す、または削除するには、MQMessage クラスのメソッドを使用します。

メッセージ・プロパティーに関する一般情報については、19 ページの『プロパティー名』を参照してください。

WebSphere MQ classes for Java では、メッセージへのアクセスは MQMessage クラスを介して行われます。 したがって、メッセージ・ハンドルは Java 環境では提供されず、 WebSphere MQ 関数呼び出し MQCRTMH、MQDLTMH、MQMHBUF、および MQBUFMH に相当するものはありません。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーを設定するには MQSETMP 呼び出しを使用します。 WebSphere MQ classes for Java で、MQMessage クラスの適切なメソッドを使用します。

- setBooleanProperty

- setByteProperty
- setBytesProperty
- setShortProperty
- setIntProperty
- setInt2Property
- setInt4Property
- setInt8Property
- setLongProperty
- setFloatProperty
- setDoubleProperty
- setStringProperty
- setObjectProperty

これらは、*set*property* メソッドと総称される場合があります。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーの値を戻すには MQINQMP 呼び出しを使用します。 WebSphere MQ classes for Java で、MQMessage クラスの適切なメソッドを使用します。

- getBooleanProperty
- getByteProperty
- getBytesProperty
- getShortProperty
- getIntProperty
- getInt2Property
- getInt4Property
- getInt8Property
- getLongProperty
- getFloatProperty
- getDoubleProperty
- getStringProperty
- getObjectProperty

これらは、*get*property* メソッドと総称される場合があります。

プロシージャー型インターフェースでメッセージ・ハンドルのプロパティーの値を削除するには MQDLTMP 呼び出しを使用します。 WebSphere MQ classes for Java では、MQMessage クラスの deleteProperty メソッドを使用します。

WebSphere MQ classes for Java でのエラーの処理

Java `try` および `catch` ブロックを使用して、WebSphere MQ classes for Java で発生したエラーを処理します。

Java インターフェースのメソッドは、完了コードと理由コードを返しません。 代わりに、WebSphere MQ 呼び出しから返される完了コードと理由コードがどちらもゼロではないときは、必ず例外をスローします。 これによってプログラム・ロジックが簡単になり、WebSphere MQ への呼び出しごとに戻りコードを検査する必要がなくなります。 プログラムのどの部分で障害に対処するかを決めることができます。 これらのポイントでは、次の例のように、コードを `try` と `catch` のブロックで囲むことができます。

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
```

```
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                       "CC = " + ex.completionCode +
                       "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

z/OS の Java 例外で報告される WebSphere MQ 呼び出し理由コードは、 z/OS およびその他のすべてのプラットフォームの 理由コード に記載されています。

WebSphere MQ classes for Java アプリケーションの実行中にスローされる例外も、ログに書き込まれます。 ただし、アプリケーションは MQException.logExclude() メソッドを呼び出して、指定の理由コードに関連した例外はログに記録されないようにすることができます。 この処理は、指定の理由コードに関連した例外が多数スローされると予想されるため、ログがこれらの例外で一杯にならないようにしたい、という場合に使用できます。 例えば、アプリケーションがループを反復するたびにキューからメッセージを取得しようとするときに、そのほとんどの試行で、適切なメッセージがキュー上にないと予想される場合は、理由コード MQRC_NO_MSG_AVAILABLE に関連した例外をログに記録しないようにできます。 アプリケーションは以前に指定の理由コードに関連した例外をログに記録しないようにした場合は、メソッド MQException.logInclude() を呼び出せば、それらの例外を再度ログに記録できます。

理由コードが、エラーに関連する詳細を必ずしも全部は伝えてはいない場合があります。 アプリケーションは、スローされる例外ごとにリンク付きの例外も検査する必要があります。 リンク付きの例外自体には、別のリンク付きの例外がある場合があるため、リンク付きの例外は元の根本の問題に戻るチェーンを形成します。 リンク付きの例外は、java.lang.Throwable クラスのチェーニングされた例外メカニズムを使用して実装され、アプリケーションは Throwable.getCause() メソッドを呼び出してリンク付き例外を取得します。 com.ibm.mq.jmqi.JmqiException の基礎となるインスタンスは MQException のインスタンスである例外から MQException.getCause() で取得され、このエラーを発生させた原因となる java.lang.Exception は、この例外から getCause によって取得されます。

デフォルトでは、MQException クラスは例外を自動的に System.err に送ります。これは通常、コンソールにダイレクトされます。 例外がコンソールに表示されないようにするには、アプリケーションに行を追加して MQException.log=null を設定します。

WebSphere MQ classes for Java での属性値の取得と設定

getXXX() および setXXX() メソッドが多数の共通属性に対して提供されています。 その他の属性は、汎用 inquire() および set() メソッドを使用してアクセスできます。

多くの共通属性では、MQManagedObject、MQDestination、MQQueue、MQTopic、MQProcess、および MQQueueManager クラスに getXXX() メソッドと setXXX() メソッドが含まれており、これらのメソッドによって、その属性値を取得および設定できます。 ただし、MQDestination、MQQueue、および MQTopic の場合、これらのメソッドは、オブジェクトのオープン時に適切な inquire フラグおよび set フラグを指定した場合にのみ機能します。

あまり使用されない属性については、MQQueueManager、MQDestination、MQQueue、MQTopic、および MQProcess クラスはすべて、MQManagedObject と呼ばれるクラスから継承します。 このクラスは inquire() および set() のインターフェースを定義します。

new 演算子を使用して新規キュー・マネージャー・オブジェクトを作成すると、そのオブジェクトは自動的に inquire 用にオープンされます。 また、accessProcess() メソッドを使用してプロセス・オブジェクトにアクセスすると、そのオブジェクトは自動的に inquire 用にオープンされます。 しかし、accessQueue() メソッドを使用してキュー・オブジェクトにアクセスした場合は、そのオブジェクトは自動的には inquire または set 操作用にはオープンされません。 これは、これらのオプションを自動的に追加すると、一部のタイプのリモート・キューで問題を生じさせる可能性があるためです。 キューに対して inquire、set、getXXX、および setXXX メソッドを使用するには、適切な「inquire」フラグと「set」フラグを accessQueue() メソッドの openOptions パラメーターに指定しなければなりません。 同じことが、宛先およびトピック・オブジェクトの場合にも言えます。

inquire メソッドと set メソッドは、次の 3 つのパラメーターを取ります。

- selectors 配列

- intAttrs 配列
- charAttrs 配列

Java の配列の長さは常に認識されているため、MQINQ にある SelectorCount、 IntAttrCount、および CharAttrLength パラメーターは必要ありません。 次の例は、キューの照会を行う方法を示しています。

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Java におけるマルチスレッド・プログラム

Java ランタイム環境は、本質的にマルチスレッド環境です。 WebSphere MQ classes for Java では、キュー・マネージャー・オブジェクトを複数のスレッドで共有することができますが、ターゲット・キュー・マネージャーへのすべてのアクセスが同期化されます。

マルチスレッド・プログラムは、Java では避けることが困難です。 始動時にキュー・マネージャーに接続して、キューをオープンする単純なプログラムを考えてみましょう。 このプログラムは、画面上に 1 つのボタンを表示します。 ユーザーがこのボタンをクリックすると、プログラムはキューからメッセージを取り出します。

Java ランタイム環境は、本質的にマルチスレッド環境です。 したがって、ユーザー・アプリケーションの初期化はある 1 つのスレッドで実行され、ボタンが押された応答で実行されるコードは別のスレッド (ユーザー・インターフェース・スレッド) で実行されます。

C ベースの WebSphere MQ MQI クライアントでは、複数のスレッド間でのハンドルの共用に制限があるため、この処理は問題となります。 WebSphere MQ classes for Java は、この制約を緩和して、キュー・マネージャー・オブジェクト (およびそれに関連するキュー、トピック、およびプロセス・オブジェクト) を複数のスレッドで共有できるようにします。

WebSphere MQ classes for Java の実装により、特定の接続 (MQQueueManager オブジェクト・インスタンス) について、ターゲットの WebSphere MQ キュー・マネージャーへのすべてのアクセスが同期されます。キュー・マネージャーに呼び出しを発行するスレッドは、その接続で進行中の他の呼び出しがすべて完了するまでブロックされます。 プログラム内の複数のスレッドから同じキュー・マネージャーに同時にアクセスする必要がある場合は、同時アクセスが必要なスレッドごとに新しい MQQueueManager オブジェクトを作成します。 (これは、スレッドごとに別の MQCONN 呼び出しを発行する ことと同じです。)

注: クラス `com.ibm.mq.MQGetMessageOptions` のインスタンスは、メッセージを同時に要求する複数のスレッド間で共用することはできません。 このクラスのインスタンスは、対応する MQGET 要求の間にデータが更新されます。そのため、このオブジェクトの同じインスタンスを複数のスレッドが同時に処理しようとすると、予期しない結果になることがあります。

WebSphere MQ classes for Java でのチャネル出口の使用

WebSphere MQ classes for Java を使用するアプリケーションでチャネル出口を使用する方法の概要。

以下のトピックでは、Java でチャネル出口を作成する方法、それを割り当てる方法、およびそれにデータを渡す方法について説明します。 その後、C で作成されたチャネル出口の使用方法および一連のチャネル出口の使用方法を説明します。

アプリケーションに、チャネル出口クラスをロードするための適切なセキュリティー権限が必要です。

WebSphere MQ classes for Java でのチャネル出口の作成

適切なインターフェースを実装する Java クラスを定義することにより、独自のチャネル出口を提供できます。

出口を実装するには、適切なインターフェースを実装する新しい Java クラスを定義します。 com.ibm.mq.exits パッケージには、次の 3 つの出口インターフェースが定義されています。

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

注 : チャネル出口は、クライアント接続でのみサポートされます。バインディング接続ではサポートされません。 例えば、C で作成されたクライアント・アプリケーションを使用している場合、 WebSphere MQ classes for Java の外部で Java チャネル出口を使用することはできません。

接続に定義されている SSL 暗号化はすべて、送信出口およびセキュリティー出口が呼び出された後で 実行されます。 同様に暗号化解除は、受信およびセキュリティー出口が呼び出される 前に 実行されます。

以下の例は、3 つのインターフェースすべてを実装するクラスを定義しています。

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                       MQCD channelDefinition,
                                       ByteBuffer agentBuffer)
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                        MQCD channelDefinition,
                                        ByteBuffer agentBuffer)
  {
    // Fill in the body of the security exit here
  }
}
```

出口ごとに、MQCXP オブジェクトおよび MQCD オブジェクトが渡されます。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

作成する出口クラスには、コンストラクターがなければなりません。 これはデフォルト・コンストラクターまたはストリング引数を持つコンストラクターのいずれかです。 それがストリングを取る場合、ユーザー・データは作成時に出口クラスに渡されます。 出口クラスにデフォルトのコンストラクターと単一引数コンストラクターの両方が含まれる場合、単一引数コンストラクターが優先されます。

送信出口およびセキュリティー出口の場合、出口コードは、サーバーに送信されるデータを戻す必要があります。 受信出口の場合、出口コードは、WebSphere MQ に解釈させる変更済みデータを戻す必要があります。

次は、考えられる最も単純な出口の本体です。

```
{ return agentBuffer; }
```

キュー・マネージャーはチャネル出口内からはクローズしないでください。

既存のチャネル出口クラスの使用

WebSphere MQ の 7.0 より前のバージョンでは、これらの出口を以下の例のようにインターフェース MQSendExit、MQReceiveExit、および MQSecurityExit を使用して実装します。 この方法は有効なままですが、機能性とパフォーマンスを向上させたい場合には、新しい方法をお勧めします。

```
public class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
    // Fill in the body of the send exit here
  }
    // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
    // Fill in the body of the receive exit here
  }
    // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
    // Fill in the body of the security exit here
  }
}
```

IBM WebSphere MQ classes for Java でのチャネル出口の割り当て

IBM WebSphere MQ classes for Java を使用してチャネル出口を割り当てることができます。

IBM WebSphere MQ classes for Java では IBM WebSphere MQ チャネルに直接相当するものはありません。 チャネル出口は MQQueueManager に割り当てられます。 例えば、WMQSecurityExit インターフェースを実装するクラスが定義されている場合、アプリケーションは次の 4 とおりの方法でセキュリティー出口を使用できます。

- MQQueueManager オブジェクトを作成する前に、クラスのインスタンスを MQEnvironment.channelSecurityExit フィールドに割り当てる
- MQQueueManager オブジェクトを作成する前に、MQEnvironment.channelSecurityExit フィールドをセキュリティー出口クラスを表すストリングに設定する
- MQQueueManager へ CMQC.SECURITY_EXIT_PROPERTY のキーで渡されるプロパティーのハッシュ・テーブル内にキー/値のペアを作成する
- クライアント・チャネル定義テーブル (CCDT) の使用

MQEnvironment.channelSecurityExit フィールドをストリングに設定することにより、プロパティーのハッシュ・テーブルにキー/値のペアを作成することにより、または CCDT を使用することによって割り当てられる出口は、デフォルト・コンストラクターで作成されていなければなりません。 アプリケーションによっては、クラスのインスタンスとして割り当てられた出口には、デフォルト・コンストラクターは必要ありません。

アプリケーションは、同様の方法で送信出口または受信出口を使用できます。 例えば、以下のコード・フラグメントは、クラス MyMQExits (MQEnvironment を使用して以前に定義済み) に実装されるセキュリティー出口、送信出口、および受信出口の使用方法を示しています。

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

複数のメソッドを使用することによってチャネル出口が割り当てられた場合、優先順位は次のとおりです。

1. CCDT の URL が MQQueueManager に渡された場合、CCDT の内容によって使用されるチャネル出口が決定され、MQEnvironment またはプロパティーのハッシュ・テーブル内の出口定義は無視されます。
2. CCDT URL が渡されていない場合、MQEnvironment およびハッシュ・テーブルからの出口定義がマージされます。
   - 同じ出口タイプが MQEnvironment とハッシュ・テーブルの両方で定義されている場合、ハッシュ・テーブル内の定義が使用されます。
   - 旧タイプと新タイプで等価の出口が指定されている場合 (例えば、IBM WebSphere MQ バージョン 7.0 より前のバージョンで使用されるタイプの出口にのみ使用できる sendExit フィールドと、どの送信出口にも使用できる channelSendExit フィールド)、旧出口ではなく新規の出口 (channelSendExit) が使用されます。

チャネル出口をストリングとして宣言した場合、IBM WebSphere MQ がチャネル出口プログラムの場所を探索できるようにする必要があります。 これは、アプリケーションが稼働している環境およびチャネル出口プログラムのパッケージ方法に応じて、さまざまな方法で実行できます。

- アプリケーション・サーバーで実行するアプリケーションの場合は、ファイルを 695 ページの表 88 で示されたディレクトリーに格納するか、または **exitClasspath** によって参照される JAR ファイルにパッケージする必要があります。
- アプリケーション・サーバーで実行しないアプリケーションの場合は、以下の規則が適用されます。
  - チャネル出口クラスが別の JAR ファイルにパッケージされている場合、これらの JAR ファイルは **exitClasspath** に含まれていなければなりません。
  - チャネル出口クラスが JAR ファイルにパッケージされていない場合、クラス・ファイルは 695 ページの表 88 で示されたディレクトリーまたは JVM システム・クラスパスまたは **exitClasspath** 内の任意のディレクトリーに保管できます。

exitClasspath プロパティーは 4 とおりの方法で指定できます。 これらの方法を優先度の順に以下に示します。

1. システム・プロパティー com.ibm.mq.exitClasspath (コマンド行で `-D` オプションを使用して定義される)
2. `mqclient.ini` ファイルの exitPath スタンザ
3. キー CMQC.EXIT_CLASSPATH_PROPERTY を使用したハッシュ・テーブル項目
4. MQEnvironment 変数 **exitClasspath**

java.io.File.pathSeparator 文字を使用した個別の複数のパス。

表 *88.* チャネル出口プログラムのディレクトリー

プラットフォーム	**ディレクトリー**
AIX、 HP-UX、 Linux、および Solaris	`/var/mqm/exits` (32 ビットのチャネル出口プログラム) `/var/mqm/exits64` (64 ビットのチャネル出口プログラム)
Windows	*install_data_dir*¥exits
注 : *install_data_dir* は、インストール中に IBM WebSphere MQ データ・ファイル用に選択したディレクトリーです。 デフォルト・ディレクトリーは `C:\Program Files\IBM\WebSphere MQ` です。	

WebSphere MQ classes for Java でのチャネル出口へのデータの引き渡し

チャネル出口へデータを渡したり、チャネル出口からアプリケーションへデータを戻したりすることができます。

agentBuffer パラメーター

送信出口の場合、*agentBuffer* パラメーターには、送信される直前のデータが入ります。 受信出口やセキュリティー出口の場合は、受信された直後のデータが *agentBuffer* パラメーターに入れられます。 配列の長さは式 agentBuffer.limit() で指示されるため、長さパラメーターは必要ありません。

送信出口およびセキュリティー出口の場合、出口コードは、サーバーに送信されるデータを戻す必要があります。 受信出口の場合、出口コードは、WebSphere MQ に解釈させる変更済みデータを戻す必要があります。

次は、考えられる最も単純な出口の本体です。

```
{ return agentBuffer; }
```

チャネル出口は、バッキング配列を持つバッファーとともに呼び出されます。 最良のパフォーマンスを得るためには、出口はバッキング配列を持つバッファーを戻す必要があります。

ユーザー・データ

channelSecurityExit、channelSendExit、または channelReceiveExit を設定することによってアプリケーションがキュー・マネージャーに接続する場合、channelSecurityExitUserData、channelSendExitUserData、または channelReceiveExitUserData フィールドを使用して、該当するチャネル出口クラスの呼び出し時に 32 バイトのユーザー・データをチャネル出口クラスに渡すことができます。 このユーザー・データはチャネル出口クラスに使用できますが、出口が呼び出されるたびにリフレッシュされます。 したがって、チャネル出口内でユーザー・データに加えられた変更はすべて失われます。 チャネル出口内のデータに永続的な変更を加える場合は、MQCXP exitUserArea を使用してください。 このフィールドのデータは、出口が呼び出される間も維持されます。

アプリケーションが securityExit、sendExit、または receiveExit を設定した場合、これらのチャネル出口クラスへはユーザー・データを渡すことはできません。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーと接続する場合、チャネル出口クラスが呼び出されるときに、クライアント接続チャネル定義で指定されたユーザー・データがチャネル出口クラスに渡されます。 クライアント・チャネル定義テーブルの使用については、678 ページの『IBM WebSphere MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』を参照してください。

WebSphere MQ classes for Java での *Java* で作成されていないチャネル出口の使用

C で作成されたチャネル出口プログラムを Java アプリケーションから使用する方法。

WebSphere MQ バージョン 7.0 では、C で作成されたチャネル出口プログラムの名前を、MQEnvironment オブジェクトまたはプロパティーのハッシュ・テーブル内の channelSecurityExit、channelSendExit、または channelReceiveExit フィールドへ渡すストリングとして指定することができます。 ただし、Java で記述されたチャネル出口プログラムを別の言語で記述されたアプリケーションで使用することはできません。

出口プログラム名を `library(function)` の形式で指定し、出口プログラムの場所がパス環境変数に含まれていることを確認します。

C でチャネル出口を作成する方法については、398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。

外部チャネル出口クラスの使用

WebSphere MQ バージョン 7.0 より前のバージョンでは、Java 以外の言語で作成されたチャネル出口を使用できるようにするために、以下の 3 つのクラスが提供されていました。

- MQExternalSecurityExit。MQSecurityExit インターフェースを実装します。
- MQExternalSendExit。MQSendExit インターフェースを実装します。
- MQExternalReceiveExit。MQReceiveExit インターフェースを実装します。

これらのクラスの使用は依然として有効ですが、新規のメソッドが推奨されます。

Java で作成されていないセキュリティー出口を使用するには、最初にアプリケーションで MQExternalSecurity 出口オブジェクトを作成する必要がありました。 アプリケーションは、MQExternalSecurityExit コンストラクターのパラメーターとして、セキュリティー出口が含まれるライブラリーの名前、セキュリティー出口のエントリー・ポイントの名前、およびセキュリティー出口が呼び出されたときに渡されるユーザー・データを指定しました。 Java で作成されないチャネル出口プログラムは、695 ページの表 88 で示されたディレクトリーに保管されていました。

WebSphere MQ classes for Java での一連のチャネル送信出口または受信出口の使用

WebSphere MQ classes for Java アプリケーションは、連続して実行される一連のチャネル送信出口または受信出口を使用できます。

一連の送信出口を使用するために、アプリケーションは、送信出口を含むリストまたはストリングを作成できます。 リストを使用する場合、リストの各エレメントは以下のいずれかにします。

- WMQSendExit インターフェースを実装するユーザー定義クラスのインスタンス
- MQSendExit インターフェースを実装するユーザー定義クラスのインスタンス (Java で作成された送信出口の場合)
- MQExternalSend 出口クラスのインスタンス (Java で作成されていない送信出口の場合)
- MQSendExitChain クラスのインスタンス
- ストリング・クラスのインスタンス

リストには別のリストを含めることはできません。

アプリケーションは、同様の方法で、一連の受信出口を使用できます。

ストリングを使用する場合は、1 つ以上のコンマ区切りの出口定義で構成する必要があります。それぞれの出口定義は、Java クラスの名前、または `library(function)` という形式の C プログラムにすることができます。

これによりアプリケーションは、MQQueueManager オブジェクトを作成する前に、リストまたはストリング・オブジェクトを MQEnvironment.channelSendExit フィールドに割り当てます。

出口に受け渡される情報のコンテキストは、出口のドメイン内だけです。 例えば、Java 出口と C 出口がチェーニングされている場合、Java 出口の存在は C 出口に影響しません。

出口チェーン・クラスの使用

WebSphere MQ バージョン 7.0 より前のバージョンでは、以下の 2 つのクラスが提供され、これらによって出口のシーケンスが許可されました。

- MQSendExitChain。MQSendExit インターフェースを実装します。
- MQReceiveExitChain。MQReceiveExit インターフェースを実装します。

これらのクラスの使用は依然として有効ですが、新規のメソッドが推奨されます。 WebSphere MQ Classes for Java インターフェースを使用することは、アプリケーションがまだ `com.ibm.mq.jar` に依存していることを意味します。 `com.ibm.mq.exits` パッケージ内の新しいインターフェース・セットが使用されている場合、 `com.ibm.mq.jar` に対する依存関係はありません。

一連の送信出口を使用するために、アプリケーションは、オブジェクトのリストを作成しました。リスト内の各オブジェクトは、次のいずれかでした。

- MQSendExit インターフェースを実装するユーザー定義クラスのインスタンス (Java で作成された送信出口の場合)
- MQExternalSend 出口クラスのインスタンス (Java で作成されていない送信出口の場合)
- MQSendExitChain クラスのインスタンス

アプリケーションは、このオブジェクト・リストをコンストラクターのパラメーターとして渡すことによって、MQSendExitChain オブジェクトを作成しました。 これによりアプリケーションは、MQQueueManager オブジェクトを作成する前に、MQSendExitChain オブジェクトを MQEnvironment.sendExit フィールドに割り当てることができました。

WebSphere MQ classes for Java でのチャネル圧縮

チャネルを流れるデータを圧縮すると、チャネルのパフォーマンスを改善し、ネットワーク・トラフィックを削減することができます。IBM WebSphere MQ classes for Java IBM WebSphere MQ に組み込まれている圧縮機能を使用します。

IBM WebSphere MQ で提供される機能を使用して、メッセージ・チャネルと MQI チャネルを流れるデータを圧縮できます。また、どちらのタイプのチャネルでも、ヘッダー・データとメッセージ・データを個別に圧縮できます。 デフォルトでは、チャネル上のデータは圧縮されません。 IBM WebSphere MQ への実装方法など、チャネル圧縮の 詳細については、データ圧縮 (COMPMSG) およびヘッダー圧縮 (COMPHDR) を参照してください。

IBM WebSphere MQ classes for Java アプリケーションは、クライアント接続のヘッダー・データまたはメッセージ・データの圧縮に使用できる手法を指定するために、java.util.Collection オブジェクトを作成します。 各圧縮手法は、このコレクション内の Integer オブジェクトであり、アプリケーションが圧縮手法をコレクションに追加する順序は、クライアント接続の開始時にキュー・マネージャーとの間で圧縮手法がネゴシエーションされる順序になります。 これによりアプリケーションは、MQEnvironment クラスの hdrCompList フィールド (ヘッダー・データの場合) または msgCompList フィールド (メッセージ・データの場合) にコレクションを割り当てることができます。 アプリケーションの準備ができたら、MQQueueManager オブジェクトを作成すればクライアント接続を開始できます。

以下のコード・フラグメントは、ここで説明した方法の例です。 最初のコード・フラグメントは、ヘッダー・データ圧縮の実装方法を示します。

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

2 番目のコード・フラグメントは、メッセージ・データ圧縮の実装方法を示します。

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

2 番目の例では、クライアント接続の開始時に、圧縮手法は RLE、ZLIBHIGH の順でネゴシエーションされます。 選択された圧縮手法は、MQQueueManager オブジェクトの存続期間中は変更できません。

クライアント接続のクライアントとキュー・マネージャーの両方でサポートされるヘッダー・データおよびメッセージ・データの圧縮手法は、MQChannelDefinition オブジェクトの hdrCompList フィールドおよび msgCompList フィールドのコレクションとして、チャネル出口に渡されます。 クライアント接続でヘッダー・データおよびメッセージ・データの圧縮に現在使用されている実際の手法は、MQChannelExit オブジェクトの CurHdrCompression フィールドおよび CurMsgCompression フィールドのチャネル出口に渡されます。

圧縮がクライアント接続で使用される場合、データは、チャネル送信出口が処理される前に圧縮され、チャネル受信出口が処理された後に抽出されます。 このため、送信出口および受信出口に渡されるデータは、圧縮された状態になります。

圧縮手法の指定および使用可能な圧縮手法の詳細については、クラス com.ibm.mq.MQEnvironment およびインターフェース com.ibm.mq.MQC を参照してください。

IBM WebSphere MQ classes for Java における TCP/IP 接続の共用

MQI チャネルの複数インスタンスが、単一の TCP/IP 接続を共用するようにできます。

IBM WebSphere MQ classes for Java では、単一の TCP/IP 接続を共用できる会話の数を、MQEnvironment.sharingConversations 変数を使用して制御します。

SHARECNV 属性は、接続共用へのベスト・エフォート・アプローチです。 したがって、IBM WebSphere MQ classes for Java で 0 より大きい SHARECNV 値を使用した場合は、新しい接続要求は既に確立されている接続を常に共用するという保証はありません。

WebSphere MQ classes for Java での接続プーリング

WebSphere MQ classes for Java では、予備の接続を再利用のためにプールすることができます。

WebSphere MQ classes for Java は、 WebSphere MQ キュー・マネージャーへの複数接続を扱うアプリケーションのための追加サポートを提供します。 そのため、ある接続が必要でなくなった際に、その接続を破棄しないでプールに入れておき、後で再利用することが可能になります。 これは、任意のキュー・マネージャーに連続して接続するアプリケーションやミドルウェアのパフォーマンスを大幅に向上させます。

WebSphere MQ には、デフォルトの接続プールがあります。 アプリケーションは、MQEnvironment クラスでトークンを登録したり登録から外したりすることで、この接続プールをアクティブにしたり非アクティブにしたりできます。 WebSphere MQ classes for Java が MQQueueManager オブジェクトを作成するときにプールがアクティブである場合、プールはこのデフォルト・プールを検索し、適切な接続を再使用します。 そして、MQQueueManager.disconnect() が呼び出されると、その下にある接続はプールに戻されます。

別の方法として、特定の用途のためにアプリケーションで MQSimpleConnectionManager 接続プールを構成することもできます。 こうして作成されたプールは、MQQueueManager オブジェクトの構成の際に指定することもできるほか、デフォルト接続プールとして使用するために MQEnvironment に渡すこともできます。

接続があまりに多くのリソースを使わないようにするために、MQSimpleConnectionManager オブジェクトが処理できる接続の総数を制限したり、接続プールの大きさを制限することができます。 JVM 内での接続に重複した需要がある場合、限度を設定することは便利です。

getMaxConnections() メソッドは、デフォルトで値ゼロを戻します。これは、MQSimpleConnectionManager オブジェクトが処理できる接続数に制限がないことを意味します。 setMaxConnections() メソッドを使うことにより、制限を設定することができます。 制限を設定し、その限界に達した場合、それ以降の接続要求は MQException をスローすることになり、理由コードは MQRC_MAX_CONNS_LIMIT_REACHED になります。

WebSphere MQ classes for Java でのデフォルト接続プールの制御

この例では、デフォルト接続プールの使用方法を示します。

次のサンプル・アプリケーション MQApp1 について検討します。

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 は、コマンド行からローカル・キュー・マネージャーのリストを入手し、リスト内の各キュー・マネージャーに順番に接続して、何らかの操作を実行します。 しかし、コマンド行に同じキュー・マネージャーが何回もリストされているような場合は、接続の確立を 1 回のみとして、その接続を何回も再利用した方がより効率的です。

WebSphere MQ classes for Java には、これを行うために使用できるデフォルト接続プールが用意されています。 このプールを使用可能にする場合は、いずれかの MQEnvironment.addConnectionPoolToken() メソッドを使用します。 プールを使用不可にする場合は、MQEnvironment.removeConnectionPoolToken() を使用します。

次のサンプル・アプリケーション MQApp2 は、機能的には MQApp1 と同じですが、それぞれのキュー・マネージャーに一度ずつしか接続しません。

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

このアプリケーションでは、1 つ目の太字になっている行で MQPoolToken オブジェクトを MQEnvironment に登録することにより、デフォルト接続プールが使用可能にされています。

MQQueueManager コンストラクターは、このプールに適当な接続がないかどうかを調べ、該当するキュー・マネージャーへの接続が存在していない場合にのみ接続を作成します。 使用された接続は、再利用された後、qmgr.disconnect() 呼び出しでプールに戻されます。 これらの API 呼び出しは、サンプル・アプリケーション MQApp1 と同じです。

2 つ目の強調表示されている行では、デフォルト接続プールが非アクティブにされています。これにより、そのプールに保管されているキュー・マネージャー接続はすべて破棄されます。 このように接続を破棄しないと、プール内のいくつかのキュー・マネージャー接続が使用されたままの状態でアプリケーションが終了されてしまうため、この処理は重要です。 接続が破棄されないままアプリケーションを終了すると、キュー・マネージャー・ログに記録されるようなエラーを引き起こす恐れがあります。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合、MQQueueManager コンストラクターは、まずテーブルを検索して、適切なクライアント接続チャネル定義を探します。 該当する定義が見つかった場合、コンストラクターは、チャネルに使用できる接続のデフォルト接続プールを検索します。 プール内で適切な接続が見つからなかった場合、コンストラクターは、クライアント・チャネル定義テーブルを検索して、次に適切なクライアント接続チャネル定義を探し、前述のとおりに処理します。 クライアント・チャネル定義テーブルの検索は完了したが、プール内で適切な接続が見つからなかった場合、コンストラクターはそのテーブルについて 2 回目の検索を開始します。 この検索では、コンストラクターは、適切な各クライアント接続チャネル定義について順番に新規接続の作成を試行して、最初に作成できた接続を使用します。

デフォルト接続プールでは、使用されていない接続を最大で 10 まで保管でき、使用されていない接続を最大で 5 分、アクティブに保つことができます。 この制限は、アプリケーションで変更できます (詳細は 701 ページの『 WebSphere MQ classes for Java での別の接続プールの提供』を参照してください)。

MQEnvironment を使用して MQPoolToken を用意する代わりに、次のように、アプリケーションで独自のものを構成することもできます。

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

一部のアプリケーション・ベンダーやミドルウェアのベンダーでは、カスタム接続プールに情報を渡すために MQPoolToken のサブクラスを用意しています。 この方法で構成して addConnectionPoolToken() に渡すと、その接続プールに追加情報も渡すことができます。

WebSphere MQ classes for Java におけるデフォルトの接続プールと複数のコンポーネント

この例では、登録された MQPoolToken オブジェクトの静的な集合の MQPoolToken を追加または削除する方法を示します。

MQEnvironment は、登録された MQPoolToken オブジェクトの静的な集合を保持します。 この集合での MQPoolTokens の追加と除去には、次のメソッドを使用します。

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

アプリケーションは、独立して存在し、キュー・マネージャーを使用して作業を行ういくつものコンポーネントによって構成されている場合があります。 そのようなアプリケーションでは、それぞれのコンポーネントが、その存続期間の間、MQEnvironment の集合に MQPoolToken を追加する必要があります。

例えば、サンプル・アプリケーション MQApp3 では、10 のスレッドを作成し、それぞれを開始します。 各スレッドはそれぞれの MQPoolToken に登録し、ある程度の時間待機して、それからキュー・マネージャーに接続します。 接続が切断されると、スレッドはそれぞれの MQPoolToken を除去します。

デフォルト接続プールは、MQPoolTokens の集合に 1 つでもトークンが残っている限り、つまりこのアプリケーションの存続期間の間はアクティブのままとなっています。 アプリケーションは、これらのスレッド全体の制御において、マスター・オブジェクトを保持する必要はありません。

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

WebSphere MQ classes for Java での別の接続プールの提供

この例は、クラス **com.ibm.mq.MQSimpleConnectionManager** を使用して別の接続プールを用意する方法を示しています。

このクラスには、接続プーリングのための基本的な機能が備わっており、アプリケーションはこのクラスを使用してプールの振る舞いをカスタマイズできます。

MQSimpleConnectionManager は、インスタンス化されると、MQQueueManager コンストラクターで指定できるようになります。 MQSimpleConnectionManager は、構成された MQQueueManager の下にある接続を管理するようになります。 MQSimpleConnectionManager にプールされた適切な接続が含まれる場合、その接続は再利用され、MQQueueManager.disconnect() 呼び出しが実行された後に MQSimpleConnectionManager に戻されます。

次のコード・フラグメントは、その動作を実例で示しています。

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

最初の MQQueueManager コンストラクターで作られた接続は、qmgr.disconnect() 呼び出しの後 myConnMan に保管されます。 この接続は、次に MQQueueManager コンストラクターが呼び出されたときに再利用されます。

2 番目の行では MQSimpleConnectionManager が使用可能にされます。 そして最後の行では MQSimpleConnectionManager が使用不可にされ、そのプールに保持されていた接続がすべて破棄されます。 なお、MQSimpleConnectionManager はデフォルトで MODE_AUTO になっていますが、これについてはこのセクションの後の方で説明します。

MQSimpleConnectionManager は、一番最近に使用された接続から順に割り振り、使用されてから最も時間が経過している接続から順に破棄していきます。 デフォルトでは、その接続が 5 分間使用されなかった場合と、プール内の使用されていない接続の数が 10 を超えた場合に、接続が破棄されます。 これらの値を変更するには、MQSimpleConnectionManager.setTimeout() を呼び出します。

加えて、MQQueueManager コンストラクターに Connection Manager が指定されなかった場合に使用するため、デフォルト接続プールとして使用する MQSimpleConnectionManager をセットアップすることもできます。

次のアプリケーションは、これを例示したものです。

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

太線の行は、MQSimpleConnectionManager オブジェクトを作成および構成します。 構成は以下を行います。

- 1 時間に渡って使用されなかった接続を破棄します。
- myConnMan によって管理される接続の数を 75 までに制限します。
- プール内の使用されていない接続を 50 までに制限します。
- MODE_AUTO を設定します。これはデフォルトです。 これを使用すると、これがデフォルト接続マネージャーであり、かつ MQEnvironment によって保持される MQPoolTokens の集合に少なくとも 1 つ以上のトークンが含まれている場合にのみ、このプールがアクティブになります。

新しい MQSimpleConnectionManager は、デフォルト接続マネージャーとして設定されます。

最後の行で、アプリケーションは MQApp3.main() を呼び出します。 これにより、いくつかのスレッドが実行されます。各スレッドは WebSphere MQ を個別に使用します。 これらのスレッドは、接続を作成する際に myConnMan を使用します。

WebSphere MQ classes for Java 用の独自の *ConnectionManager* の提供

WebSphere MQ classes for Java は、 Java EE コネクター・アーキテクチャーの部分的な実装を提供し、javax.resource.spi.ConnectionManager の実装を使用できるようにします。

アプリケーションやミドルウェアのプロバイダーが、代替の接続プールの実装を作成できます。 WebSphere MQ classes for Java は、 Java EE コネクター・アーキテクチャーの部分的な実装を提供します。 **javax.resource.spi.ConnectionManager** の実装は、デフォルトの Connection Manager として使用することも、また MQQueueManager コンストラクターで指定することもできます。

WebSphere MQ classes for Java は、 Java EE コネクター・アーキテクチャーの接続管理契約に準拠しています。 このセクションをお読みになる際には、Java EE Connector Architecture の Connection Management コントラクトに関する資料も併せてご覧ください (https://java.sun.com にある Sun の Java Web サイトを参照)。

ConnectionManager インターフェースでは、次のメソッドのみが定義されています。

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

MQQueueManager コンストラクターは、適切な ConnectionManager で allocateConnection を呼び出します。 必要な接続を記述するパラメーターとして、ManagedConnectionFactory と ConnectionRequestInfo の実装が渡されます。

ConnectionManager は、同じ ManagedConnectionFactory および ConnectionRequestInfo オブジェクトで作成された javax.resource.spi.ManagedConnection オブジェクトがないかどうかプールを調べます。 適切な ManagedConnection オブジェクトが見つかった場合、ConnectionManager はその候補の ManagedConnection を含む java.util.Set を作成します。 その後、ConnectionManager は以下を呼び出します。

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

ManagedConnectionFactory の WebSphere MQ 実装は、subject パラメーターを無視します。 このメソッドは、集合から適切な ManagedConnection を選択して戻します。適切な ManagedConnection が見つからない場合はヌルを戻します。 プール内に適当な ManagedConnection がない場合、ConnectionManager は以下のメソッドを使用してこれを作成できます。

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

この場合も、subject パラメーターは無視されます。 このメソッドは、WebSphere MQ キュー・マネージャーに接続し、新しく作られた接続を表す javax.resource.spi.ManagedConnection の実装を戻します。 ConnectionManager は、ManagedConnection を取得する (プールから、または新しく作成することによって) と、次のメソッドを使用して接続ハンドルを作成します。

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

この接続ハンドルは、allocateConnection() を呼び出すことによって戻されます。

ConnectionManager は、次のメソッドを使用して ManagedConnection にインタレストを登録します。

```
mc.addConnectionEventListener()
```

接続で重大エラーが発生した場合や、MQQueueManager.disconnect() が呼び出された際には、ConnectionEventListener にその旨が通知されます。 MQQueueManager.disconnect() が呼び出された場合、ConnectionEventListener は次のいずれかの処理を行います。

- mc.cleanup() 呼び出しを使用して ManagedConnection をリセットし、ManagedConnection をプールに戻します。
- mc.destroy() 呼び出しを使用して ManagedConnection を破棄します。

ConnectionManager は、デフォルトの ConnectionManager である場合に、インタレストを MQEnvironment によって管理される MQPoolTokens の集合の状態に登録することもできます。 これを行うためには、次のように、まず MQPoolServices オブジェクトを構成し、次いで MQPoolServices オブジェクトに MQPoolServicesEventListener オブジェクトを登録します。

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

MQPoolToken が集合に加えられたり集合から除去された場合や、デフォルトの ConnectionManager が変更された場合には、その旨がリスナーに通知されます。 MQPoolServices オブジェクトは、MQPoolTokens の集合の現在のサイズを照会するためにも使用できます。

WebSphere MQ classes for Java を使用した JTA/JDBC 調整

WebSphere MQ classes for Java は、 MQQueueManager.begin () メソッドをサポートします。これにより、WebSphere MQ は、 JDBC タイプ 2 または JDBC タイプ 4 準拠ドライバーを提供するデータベースのコーディネーターとして機能することができます。

このサポートは、一部のプラットフォームでは使用できません。 JDBC の調整をサポートしているプラットフォームを確認するには、https://www.ibm.com/software/integration/wmq/requirements/ を参照してください。

XA-JTA サポートを使用するには、特殊な JTA 切り替えライブラリーを使用する必要があります。 このライブラリーを使用するためのメソッドは、Windows を使用しているか、またはその他のプラットフォームのいずれかを使用しているかによって異なります。

Windows での JTA/JDBC 調整の構成

XA ライブラリーは、`jdbcxxx.dll` という形式の名前の DLL として提供されています。

V 7.5.0.7 付属の `jdbcora12.dll` は、IBM WebSphere MQ Windows サーバー・インストール環境で、Oracle 12C の使用を可能にします。

Windows システムでは、新しい XA ライブラリーは完全 DLL として提供されています。 この DLL の名前は `jdbcxxx.dll` です。`xxx` は、切り替えライブラリーがコンパイルされたデータベースを示します。 このライブラリーは、 IBM WebSphere MQ classes for Java インストール済み環境の `java\lib\jdbc` ディレクトリーまたは `java\lib64\jdbc` ディレクトリーにあります。 スイッチ・ロード・ファイルとしても説明されている XA ライブラリーをキュー・マネージャーに対して宣言する必要があります。 IBM WebSphere MQ Explorer を使用します。 XA リソース・マネージャーのキュー・マネージャー・プロパティー・パネルで、スイッチ・ロード・ファイルの詳細を指定します。 ライブラリーの名前のみを指定する必要があります。 以下に例を示します。

Db2 データベース・セットの場合は、SwitchFile フィールドを dbcdb2 に設定します。

Oracle データベース・セットの場合は、SwitchFile フィールドを `jdbcora` に設定します。

Windows 以外のプラットフォームでの JTA/JDBC 調整の構成

オブジェクト・ファイルは付属しています。 付属の makefile を使用して、適切なオブジェクト・ファイルをリンクし、構成ファイルを使用してそのファイルをキュー・マネージャーに宣言します。

データベース管理システムごとに、WebSphere MQ は 2 つのオブジェクト・ファイルを提供します。 一方のオブジェクト・ファイルは 32 ビット切り替えライブラリーを作成する場合にリンクし、もう一方のオブジェクト・ファイルは 64 ビット切り替えライブラリーを作成する場合にリンクする必要があります。 DB2 の場合、各オブジェクト・ファイルの名前は jdbcdb2.o で、Oracle の場合、各オブジェクト・ファイルの名前は jdbcora.o です。

各オブジェクト・ファイルにリンクするには、WebSphere MQ で提供される適切な makefile を使用する必要があります。 切り替えライブラリーが必要とするその他のライブラリーの中には、異なるシステムの異

なる場所に保管されるものもあります。 ただし、切り替えライブラリーは、ライブラリー・パス環境変数を使用してこれらのライブラリーを位置指定することはできません。切り替えライブラリーはキュー・マネージャーによってロードされますが、キュー・マネージャーは setuid 環境で稼働するからです。 したがって、提供された Make ファイルにより、スイッチ・ライブラリーにこれらのライブラリーの完全修飾パス名が含まれるようになります。

切り替えライブラリーを作成するには、**make** コマンドを次の形式で入力します。 32 ビット切り替えライブラリーを作成するには、WebSphere MQ インストールの /java/lib/jdbc ディレクトリーで、次のコマンドを入力します。 64 ビット切り替えライブラリーを作成するには、/java/lib64/jdbc ディレクトリーで、コマンドを入力します。

```
make DBMS
```

DBMS は、切り替えライブラリーの作成対象となるデータベース管理システムです。 有効な値は、 db2 (DB2 の場合) および `oracle` (Oracle の場合) です。

次に、**make** コマンドの例を示します。

```
make db2
```

以下の点に注意してください。

- 32 ビット・アプリケーションを実行するには、 使用するデータベース管理システムごとに 32 ビット切り替えライブラリーと 64 ビット切り替えライブラリーの両方を作成する必要があります。 64 ビット・アプリケーションを実行する場合、 作成する必要があるのは 64 ビット切り替えライブラリーだけです。 各切り替えライブラリーの名前は、DB2 の場合は jdbcdb2、 Oracle の場合は jdbcora です。 makefile を使用すると、32 ビット切り替えライブラリーと 64 ビット切り替えライブラリーが確実に異なる WebSphere MQ ディレクトリーに保管されます。 32 ビット切り替えライブラリーは /java/lib/jdbc ディレクトリーに保管され、64 ビット切り替えライブラリーは /java/lib64/jdbc ディレクトリーに保管されます。
- Oracle はシステム上のどこにでもインストールできるため、make ファイルでは ORACLE_HOME 環境変数を使用して、Oracle のインストール場所を位置指定します。

DB2、Oracle、または両方の切り替えライブラリーを作成したら、それらをキュー・マネージャーに宣言する必要があります。 キュー・マネージャー構成ファイル (qm.ini) にすでに DB2 または Oracle データベース用の XAResourceManager スタンザが含まれている場合は、各スタンザの SwitchFile 項目を次のいずれかに置き換える必要があります。

DB2 データベースの場合

```
SwitchFile=jdbcdb2
```

Oracle データベースの場合

```
SwitchFile=jdbcora
```

32 ビットまたは 64 ビットの切り替えライブラリーの完全修飾パス名は指定しないでください。 ライブラリーの名前のみを指定してください。

キュー・マネージャー構成ファイルに DB2 または Oracle データベース用の XAResourceManager スタンザがまだ含まれていない場合、あるいはさらに XAResourceManager スタンザを追加したい場合は、XAResourceManager スタンザの構成方法について、「管理」を参照してください。 ただし、新規 XAResourceManager スタンザの各 SwitchFile 項目は、DB2 データベースまたは Oracle データベースの場合について前述したとおりでなければなりません。 また、`ThreadOfControl=PROCESS` の項目も含めなければなりません。

キュー・マネージャー構成ファイルの更新が完了し、すべての適切なデータベース環境変数が設定されたことを確認したら、キュー・マネージャーを再始動できます。

JTA/JDBC 調整の使用

提供されている例のように API 呼び出しをコーディングします。

以下に、ユーザー・アプリケーション用の一連の基本 API 呼び出しを示します。

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

getJDBCConnection 呼び出しの xads は、接続先のデータベースの詳細を定義する、データベース固有の XADataSource インターフェースの実装です。 getJDBCConnection に渡す適切な XADataSource オブジェクトの作成方法を判別するには、ご使用のデータベースに関する資料を参照してください。

JDBC の機能を実行するために、該当するデータベース固有の jar ファイルでクラスパスを更新する必要もあります。

複数のデータベースに接続する必要がある場合は、getJDBCConnection を複数回呼び出して、異なるいくつかの接続に対してトランザクションを実行する必要があります。

XADataSource.getXAConnection の 2 種類の形式を反映して、getJDBCConnection には 2 種類の形式があります。

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

これらのメソッドは、throws 文節で Exception を宣言します。それは、JTA 機能を使用しないお客様が JVM ベリファイヤーに関する問題を回避できるようにするためです。 実際には javax.transaction.xa.XAException という例外がスローされ、以前は必要なかったプログラムのクラスパスに jta.jar ファイルを追加しなければならなくなります。

JTA/JDBC サポートを使用するには、アプリケーションに以下のステートメントを組み込む必要があります。

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

JTA/JDBC の調整に関する既知の問題と制限

JTA/JDBC サポートには特定の問題と制限があり、その一部は使用中のデータベース管理システムにより決まります。

このサポートは JDBC ドライバーを呼び出すため、この JDBC ドライバーの実装は、システムの動作に重大な影響を与える可能性があります。 特に、アプリケーションの稼働中にデータベースをシャットダウンしてしまうと、テスト済みの JDBC ドライバーは所定の動作をしません。 アプリケーションがまだデータベースに接続している間は、**常に**データベースを突然シャットダウンしないようにしてください。

複数の XAResourceManager スタンザ

キュー・マネージャー構成ファイル (qm.ini) 内で複数の XAResourceManager スタンザの使用はサポートされません。 最初のもの以外の XAResourceManager スタンザは無視されます。

DB2

DB2 が SQL0805N エラーを戻すことがあります。 この問題は、以下の CLP コマンドを使用して解決できます。

```
DB2 bind @db2cli.lst blocking all grant public
```

詳細については、DB2 の資料を参照してください。

XAResourceManager スタンザは、ThreadOfControl=PROCESS を使用するように構成しなければなりません。 DB2 バージョン 8.1 以降では、これは DB2 のコントロール設定のデフォルト・スレッドと一

致しないので、toc=p を XA Open String で指定しなければなりません。 JTA/JDBC と調整した DB2 用の XAResourceManager のスタンザの例は、以下の通りです。

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

これにより、JTA/JDBC 調整を使用する Java アプリケーションがマルチスレッド化されなくなることはありません。

Oracle
MQQueueManager.disconnect() の後に JDBC の Connection.close() メソッドを呼び出すと、SQLException が生成されます。 MQQueueManager.disconnect() の前に Connection.close() を呼び出すか、または Connection.close() への呼び出しを省略してください。

WebSphere MQ classes for Java での Secure Sockets Layer (SSL) サポート

WebSphere MQ classes for Java クライアント・アプリケーションは、Secure Sockets Layer (SSL) 暗号化をサポートします。 SSL 暗号化を使用するには、JSSE プロバイダーが必要です。

WebSphere MQ classes for Java クライアント・アプリケーション (TRANSPORT (CLIENT) を使用) は、Secure Sockets Layer (SSL) 暗号化をサポートします。 SSL は、通信の暗号化、認証、およびメッセージの整合性を提供します。 これは一般的に、インターネット上やイントラネット内で、任意の 2 つのピアの間の通信を保護するために使用されます。

WebSphere MQ classes for Java は、Java Secure Socket Extension (JSSE) を使用して SSL 暗号化を処理するため、JSSE プロバイダーを必要とします。 JSE v1.4 JVM には、JSSE プロバイダーが組み込まれています。 証明書を管理して保管する方法は、プロバイダーごとに違います。 詳細については、各 JSSE プロバイダーの資料を参照してください。

この節では、JSSE プロバイダーが正常にインストールおよび構成されていることと、適切な証明書がユーザーの JSSE プロバイダーにインストールされて使用可能であることを前提としています。

WebSphere MQ classes for Java クライアント・アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合は、678 ページの『IBM WebSphere MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』を参照してください。

IBM WebSphere MQ classes for Java での SSL の使用可能化

SSL を有効にするには、CipherSuite を指定します。 CipherSuite を指定する方法は 2 つあります。

SSL がサポートされているのは、クライアント接続の場合のみです。 SSL を使用可能にするには、キュー・マネージャーとの通信時に使用する CipherSuite を指定する必要があります。 また、この CipherSuite は、宛先チャネルに設定されている CipherSpec と 一致していなければなりません。 さらに、指定した CipherSuite は、使用している JSSE プロバイダーでサポートされていなければなりません。 ただし、CipherSuites は CipherSpec とは異なるため、異なる名前が付けられています。 712 ページの『「 WebSphere MQ classes for Java」の「SSL CipherSpecs および CipherSuites 」』 には、IBM WebSphere MQ にサポートされている CipherSpec を、JSSE に知られている同等な CipherSpec にマップする表を示します。

SSL を有効にするには、MQEnvironment の sslCipherSuite 静的メンバー変数を使用して、CipherSuite を指定します。 次の例は、RC4_MD5_EXPORT の CipherSpec を使った SSL が必要であるため、セットアップされている SECURE.SVRCONN.CHANNEL という名前の SVRCONN チャネルに付加するものです。

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

このチャネルの CipherSpec は RC4_MD5_EXPORT ですが、Java アプリケーションが指定する必要のある CipherSuite は SSL_RSA_EXPORT_WITH_RC4_40_MD5 です。 CipherSpec と CipherSuite の間のマッピ

ングのリストは、712 ページの『「 WebSphere MQ classes for Java」の「SSL CipherSpecs および CipherSuites 」』を参照してください。

アプリケーションは、環境プロパティー CMQC.SSL_CIPHER_SUITE_PROPERTY を設定して、CipherSuite を指定することもできます。

または、クライアント・チャネル定義テーブル (CCDT) を使用します。 詳しくは、678 ページの『IBM WebSphere MQ classes for Java を含むクライアント・チャネル定義テーブルの使用』 を参照してください。

クライアント接続で IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を使用する必要がある場合、 アプリケーションは MQEnvironment クラスの sslFipsRequired フィールドを `true` に設定できます。 あるいは、アプリケーションは環境プロパティー CMQC.SSL_FIPS_REQUIRED_PROPERTY を設定することもできます。 デフォルト値は `false` であり、IBM WebSphere MQ のサポートする任意の CipherSuite を クライアント接続に使用できます。

アプリケーションが複数のクライアント接続を使用する場合は、 アプリケーションが最初のクライアント接続を作成するときに使用される sslFipsRequired フィールドの値によって、後続のクライアント接続の作成時に使用される値が決まります。 このため、 アプリケーションが後続のクライアント接続を確立する時、 sslFipsRequired フィールドの値は無視されます。 sslFipsRequired フィールドに別の値を使用する場合は、アプリケーションを再始動する必要があります。

SSL を使用した接続を正常に行うには、キュー・マネージャーによって示されている証明書の認証元である、認証局のルート証明書を使用して、JSSE truststore をセットアップしなければなりません。 同様に、SVRCONN チャネルの SSLClientAuth が MQSSL_CLIENT_AUTH_REQUIRED に設定されている場合、キュー・マネージャーによって認められている識別証明書が、JSSE keystore に含まれていなければなりません。

関連資料

UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

IBM WebSphere MQ classes for Java でのキュー・マネージャーの識別名の使用

キュー・マネージャーはそれ自体を SSL 証明書を使用して識別します。この SSL 証明書には識別名 (DN) が含まれています。 IBM WebSphere MQ classes for Java クライアント・アプリケーションでこの DN を使用すると、確実に正しいキュー・マネージャーと通信することができます。

DN パターンを指定するときは、MQEnvironment の sslPeerName 変数を使用します。 例えば、以下のように設定すると、

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

キュー・マネージャーが QMGR で始まる共通名を持つ証明書を提示した場合にのみ、接続を成功させます。および少なくとも 2 つの組織単位名。最初の名前は IBM で、2 番目の名前は WebSphere でなければなりません。

sslPeerName が設定されている場合に接続が正常に行われるのは、有効なパターンが設定されていて、キュー・マネージャーが一致する証明書を示している場合のみです。

アプリケーションは、 環境プロパティー CMQC.SSL_PEER_NAME_PROPERTY を設定することによって、キュー・マネージャーの識別名を指定することもできます。 識別名の詳細については、識別名を参照してください。

IBM WebSphere MQ classes for Java での証明書取り消しリストの使用

java.security.cert.CertStore クラスを通して使用する証明書取り消しリスト (CRL) を指定します。IBM WebSphere MQ classes for Java 指定された CRL に照らして証明書を検査します。

証明書取り消しリスト (CRL) は、発行元の認証局かローカル組織のいずれかによって取り消された証明書のセットです。 多くの場合 CRL は、LDAP サーバーにホストされています。 Java 2 v1.4 では、CRL サーバーを接続時に指定でき、接続が許可される前に、キュー・マネージャーによって示されている証明書が、CRL と比較して検査されます。 証明書失効リストおよび IBM WebSphere MQ について詳しくは、「 証明書失効リストおよび権限失効リストの処理 」および「 WebSphere MQ classes for Java および WebSphere MQ classes for JMS へのアクセス」を参照してください。

注 : CertStore を LDAP サーバーでホストされている CRL と共に正常に使用するには、ご使用の Java Software Development Kit (SDK) が CRL に適合することを確認してください。 SDK によっては、CRL が LDAP v2 用のスキーマを定義する RFC 2587 に準拠している必要があります。 ほとんどの LDAP v3 サーバーは、代わりに RFC 2256 を使用しています。

使用する CRL は、java.security.cert.CertStore クラスによって指定されます。 CertStore のインスタンスの取得方法については、このクラスの資料を参照してください。 LDAP サーバーに基づいて CertStore を作成するには、まず LDAPCertStoreParameters インスタンス (使用するサーバーおよびポート設定によって初期設定される) を作成します。 以下に例を示します。

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

CertStoreParameters インスタンスを作成してから、CertStore で静的コンストラクターを使用し、タイプ LDAP の CertStore を作成します。

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

他の CertStore タイプ (Collection など) もサポートされています。 通常は、同じ CRL 情報で複数の CRL サーバーをセットアップすることにより、冗長性が提供されます。 CRL サーバーのそれぞれに CertStore オブジェクトを用意したら、適切な Collection 内にそれらを配置してください。 次の例では、ArrayList に置かれている CertStore オブジェクトが示されています。

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

この Collection は、接続して CRL 検査を使用可能にする前に、以下のように MQEnvironment 静的変数 sslCertStores に設定できます。

```
MQEnvironment.sslCertStores = crls;
```

接続を設定するときにキュー・マネージャーによって示される証明書は、以下のようにして検査されます。

1. sslCertStores によって識別される Collection にある最初の CertStore オブジェクトを使用し、CRL サーバーを識別します。
2. CRL サーバーへ接続してみます。
3. 接続が成功すれば、一致する証明書をサーバー内で検索します。
   a. 証明書が失効したことが分かった場合、検索プロセスは終了して接続要求は失敗します。理由コードは MQRC_SSL_CERTIFICATE_REVOKED です。
   b. 証明書が見つからない場合、検索プロセスは終了し、接続を先に進めることができます。
4. サーバーへの接続が失敗する場合、次の CertStore オブジェクトを使用して、CRL サーバーを識別します。プロセスはステップ 2 から繰り返されます。

   コレクションの最後の CertStore であった場合、 またはコレクションに CertStore オブジェクトが 1 つも入っていない場合、検索プロセスは失敗し、接続要求は理由コード MQRC_SSL_CERT_STORE_ERROR で失敗します。

Collection オブジェクトは、CertStores を使用する順序を決定します。

CMQC.SSL_CERT_STORE_PROPERTY を使用して、CertStore の Collection を設定することもできます。 便宜上、このプロパティーに単一の CertStore を (コレクションのメンバーにせずに) 指定することも可能です。

sslCertStores をヌルに設定した場合、CRL 検査は実行されません。 sslCipherSuite を設定しなければ、このプロパティーは無視されます。

WebSphere MQ classes for Java での秘密鍵の再ネゴシエーション

WebSphere MQ classes for Java クライアント・アプリケーションは、送受信される合計バイト数に関して、クライアント接続で暗号化に使用される秘密鍵が再ネゴシエーションされるタイミングを制御できます。

アプリケーションは、これを以下のいずれかの方法で実行できます。アプリケーションでこれらの方法を複数使用する場合、通常の優先順位ルールが適用されます。

- MQEnvironment クラスの sslResetCount フィールドを設定する方法。
- Hashtable オブジェクトの環境プロパティー MQC.SSL_RESET_COUNT_PROPERTY を設定する。 この方法では、アプリケーションによって、MQEnvironment クラスの `properties` フィールドにハッシュ・テーブルが割り当てられるか、またはそのコンストラクターで MQQueueManager オブジェクトにハッシュ・テーブルが受け渡されます。

sslReset カウント・フィールドまたは環境プロパティー MQC.SSL_RESET_COUNT_PROPERTY の値は、秘密鍵が再ネゴシエーションされる前に WebSphere MQ classes for Java クライアント・コードによって送受信される合計バイト数を表します。 送信バイト数は暗号化前の数であり、受信バイト数は暗号化解除された後の数です。 バイト数には、 WebSphere MQ classes for Java クライアントによって送受信される制御情報も含まれます。

リセット・カウントがゼロ (デフォルト値) の場合、秘密鍵は再ネゴシエーションされません。 CipherSuite が指定されていない場合、リセット・カウントは無視されます。

IBM WebSphere MQ classes for Java でのカスタマイズした *SSLSocketFactory* の提供

カスタマイズした JSSE ソケット・ファクトリーを使用する場合、MQEnvironment.sslSocketFactory を、そのカスタマイズしたファクトリー・オブジェクトに設定します。 詳細は、JSSE 実装によって異なります。

別々の JSSE 実装に、それぞれ別個のフィーチャーを提供できます。 例えば、特殊化した JSSE 実装で、暗号化ハードウェアの特定のモデルの構成が可能です。 さらに、いくつかの JSSE プロバイダーで、プログラムによって keystores および truststores をカスタマイズしたり、keystore からの識別証明書の選択を変更したりすることもできます。 JSSE では、それらのすべてのカスタマイズが、ファクトリー・クラス javax.net.ssl.SSLSocketFactory に抽象化されます。

カスタマイズした SSLSocketFactory 実装を作成する方法については、JSSE の資料を参照してください。 それらの詳細はプロバイダーごとに異なりますが、典型的な一連の手順を以下に示します。

1. SSLContext で静的メソッドを使用して、SSLContext オブジェクトを作成します。
2. 適切な KeyManager および TrustManager 実装 (それら自身のファクトリー・クラスから作成される) を使用して、この SSLContext を初期設定します。
3. SSLContext から SSLSocketFactory を作成します。

SSLSocketFactory オブジェクトがある場合、MQEnvironment.sslSocketFactory を、そのカスタマイズしたファクトリー・オブジェクトに設定します。 以下に例を示します。

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM WebSphere MQ classes for Java はこの SSLSocketFactory を使用して、IBM WebSphere MQ キュー・マネージャーに接続します。 このプロパティーは、CMQC.SSL_SOCKET_FACTORY_PROPERTY を使用して設定することもできます。 sslSocketFactory がヌルに設定されている場合、JVM のデフォルト SSLSocketFactory が使用されます。 sslCipherSuite を設定しなければ、このプロパティーは無視されます。

カスタムの SSLSocketFactories を使用する場合には、TCP/IP 接続を共有することによる影響を考慮してください。 接続の共用が可能である場合は、生成されたソケットが後続の接続要求のコンテキストにおいて何らかの形で異なっている場合でも、用意された SSLSocketFactory に新規ソケットは要求されません。 例えば、後続の接続で別のクライアント証明書が提示される場合は、接続の共用を許可するべきではありません。

WebSphere MQ classes for Java での *JSSE* 鍵ストアまたはトラストストアの変更

JSSE 鍵ストアまたはトラストストアを変更する場合、変更を有効にするためにいくつかのアクションを実行する必要があります。

JSSE 鍵ストアまたはトラストストアの内容を変更したり、鍵ストアまたはトラストストア・ファイルの場所を変更したりした場合、その時点で実行中の WebSphere MQ classes for Java アプリケーションは、その変更を自動的に認識しません。 変更を有効にするには、以下のアクションを行う必要があります。

- アプリケーションは、すべての接続をクローズし、接続プール内の未使用の接続を破棄する必要がある。
- JSSE プロバイダーが鍵ストアおよびトラストストアからの情報をキャッシュしている場合は、この情報をリフレッシュする必要がある。

これらのアクションが完了すると、アプリケーションは接続を再作成できます。

アプリケーションの設計方法や、JSSE プロバイダーが提供する機能によっては、アプリケーションを停止および再始動しなくても上記のアクションを実行できる場合があります。 ただし、アプリケーションを停止して再始動するのが最も簡単な解決方法です。

WebSphere MQ classes for Java で *SSL* を使用する場合のエラー処理

SSL を使用してキュー・マネージャーに接続するときに、WebSphere MQ classes for Java によっていくつかの理由コードが発行されることがあります。

これらは、以下のリストで説明します。

MQRC_SSL_NOT_ALLOWED
sslCipherSuite プロパティーが設定されましたが、バインディング接続が使用されました。 SSL をサポートしているのは、クライアント接続のみです。

MQRC_JSSE_ERROR
WebSphere MQ が処理できないエラーを、JSSE プロバイダーが報告しました。 この原因として、JSSE での構成の問題か、またはキュー・マネージャーによって示された証明書が、妥当性検査できなかったことが考えられます。 JSSE によって作成された例外は、MQException で getCause() メソッドを使用して取り出すことができます。

MQRC_SSL_INITIALIZATION_ERROR
SSL 構成オプションが指定された MQCONN または MQCONNX 呼び出しが発行されましたが、SSL 環境の初期化中にエラーが発生しました。

MQRC_SSL_PEER_NAME_MISMATCH
sslPeerName プロパティーで指定した DN パターンが、キュー・マネージャーによって示された DN と一致しませんでした。

MQRC_SSL_PEER_NAME_ERROR
sslPeerName プロパティーで指定した DN パターンが無効でした。

MQRC_UNSUPPORTED_CIPHER_SUITE
sslCipherSuite で指定された CipherSuite が、JSSE プロバイダーによって認識されませんでした。 JSSE プロバイダーがサポートしている CipherSuites の完全なリストは、SSLSocketFactory.getSupportedCipherSuites() メソッドを使用して、プログラムにより取得できます。 WebSphere MQ との通信に使用できる CipherSuite のリストについては、712 ページの『「 WebSphere MQ classes for Java」の「SSL CipherSpecs および CipherSuites 」』を参照してください。

MQRC_SSL_CERTIFICATE_REVOKED
キュー・マネージャーによって示された証明書が、sslCertStores プロパティーで指定された CRL にありました。 信頼されている証明書を使用するために、キュー・マネージャーを更新してください。

MQRC_SSL_CERT_STORE_ERROR
キュー・マネージャーで示された証明書を検索したものの、指定された CertStore がありませんでした。 MQException.getCause() メソッドが、試行された最初の CertStore の検索の間に発生したエラーを戻しました。 原因である例外が NoSuchElementException、ClassCastException、または NullPointerException である場合、sslCertStores プロパティーで指定されている Collection に、有効な CertStore オブジェクトが 1 つ以上含まれているか検査してください。

「*WebSphere MQ classes for Java*」の「*SSL CipherSpecs および CipherSuites* 」

IBM WebSphere MQ classes for Java アプリケーションがキュー・マネージャーへの接続を確立できるかどうかは、MQI チャネルのサーバー側で指定された CipherSpec およびクライアント側で指定された CipherSuite によって異なります。

CipherSpec と CipherSuite の各組み合わせで、IBM WebSphere MQ classes for Java アプリケーションがキュー・マネージャーに接続できるかどうかは 、MQEnvironment クラスの sslFipsRequired フィールドの値または環境プロパティー MQC.SSL_FIPS_REQUIRED_PROPERTY の値によって決まります。

MQI チャネルのサーバー側では、 CipherSpec の名前を DEFINE CHANNEL CHLTYPE(SVRCONN) コマンドの SSLCIPH パラメーター の値として指定できます。 MQI チャネルのクライアント側では、 IBM WebSphere MQ classes for Java アプリケーションは、MQEnvironment クラスの sslCipher スイート・フィールドを設定するか、環境プロパティー CMQC.SSL_CIPHER_SUITE_PROPERTY を設定することができます。

IBM Java または Oracle Java CipherSuite マッピングを使用するためのアプリケーションの構成

IBM WebSphere MQ Version 7.5.0、フィックスパック 5 以降、アプリケーションがデフォルトの IBM Java CipherSuite から WebSphere MQ CipherSpec へのマッピングを使用するか、 Oracle CipherSuite から WebSphere MQ CipherSpec へのマッピングを使用するかを構成できます。 そのため、IBM JRE と Oracle JRE のどちらをアプリケーションで使用するかに関係なく、TLS CipherSuites を使用できます。 Java システム・プロパティーの `com.ibm.mq.cfg.useIBMCipherMappings` が使用されるマッピングを制御します。 プロパティーは、次の値のうちのいずれかです。

true
IBM Java CipherSuite から WebSphere MQ CipherSpec へのマッピングを使用します。

この値がデフォルト値です。

false
Oracle CipherSuite から WebSphere MQ CipherSpec へのマッピングを使用します。

以下の表に、IBM WebSphere MQ によってサポートされる CipherSpec と、それらに相当する CipherSuite のリストを示します。 テーブルには、CipherSpec が MQI チャネルのサーバー側で指定され、それと同等の CipherSuite がクライアント側で指定されている場合に、IBM WebSphere MQ classes for Java アプリケーションがキュー・マネージャーへの接続を確立できるかどうかも示されています。

表 *89. WebSphere MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite*

CipherSpec	**同等の CipherSuite**	**SFIPS [1] が YES に設定されている場合、接続は可能か**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	No
NULL_SHA	SSL_RSA_WITH_NULL_SHA	No
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5 (IBM JRE) Oracle JRE に相当するものはありません。	No
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	No
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No

表 89. *WebSphere MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)

CipherSpec	**同等の CipherSuite**	**SFIPS [1] が YES に設定されている場合、接続は可能か**
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (IBM JRE) SSL_RSA_EXPORT_WITH_RC4_40_MD5 (Oracle JRE)	No
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256 (IBM JRE) TLS_RSA_WITH_NULL_SHA256 (Oracle JRE)	非該当 [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA (Oracle JRE)	はい [5] [7]
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA256 (Oracle JRE)	はい [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA (Oracle JRE)	はい [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA256 (Oracle JRE)	はい [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8]	SSL_RSA_WITH_DES_CBC_SHA	いいえ [3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8] [9]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No[4]

表 89. *WebSphere MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite* (続き)		
CipherSpec	**同等の CipherSuite**	**SFIPS[1] が YES に設定されている場合、接続は可能か**
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (IBM JRE) Oracle JRE に相当するものはありません。	No[6]

注 :

1. IBM WebSphere MQ classes for Java アプリケーションで、MQEnvironment クラスの sslFipsRequired フィールドを `true` に設定して FIPS 認証アルゴリズムのみを使用することを示し、sslFipsRequired フィールドを `false` に設定して非 FIPS 認証アルゴリズムも使用できることを示します。 あるいは、環境プロパティー CMQC.SSL_FIPS_REQUIRED_PROPERTY を設定することもできます。
2. この CipherSpec には、同等の CipherSuite はありません。
3. この CipherSpec は、2007 年 5 月 19 日より前は FIPS 140-2 で認証されていました。
4. この CipherSpec は、2007 年 5 月 19 日より前は FIPS 140-2 で認証されていました。 FIPS_WITH_DES_CBC_SHA という名前は歴史的な事情によるものであり、この CipherSpec がかつては FIPS 準拠であった (現在はそうではない) という事実を反映するものです。 この CipherSpec は非推奨となりました。使用することはお勧めしません。
5. WebSphere MQ エクスプローラーが使用する JRE に対して適切な無制限のポリシー・ファイルが適用されていない場合には、 これらの CipherSpec (TLS_RSA_WITH_AES_128_CBC_SHA、TLS_RSA_WITH_AES_128_CBC_SHA256、TLS_RSA_WITH_AES_256_CBC_SHA、TLS_RSA_WITH_AES_256_CBC_SHA256) を使用してエクスプローラーからキュー・マネージャーへの安全な接続を確立することはできません。

   ポリシー・ファイルの詳細については、Security information を参照してください。
6. FIPS_WITH_3DES_EDE_CBC_SHA という名前は歴史的な事情によるものであり、この CipherSpec がかつては FIPS 準拠であった (現在はそうではない) という事実を反映するものです。 この CipherSpec は非推奨となりました。使用することはお勧めしません。
7. これらの CipherSpec (TLS_RSA_WITH_NULL_SHA256、TLS_RSA_WITH_AES_128_CBC_SHA、TLS_RSA_WITH_AES_256_CBC_SHA256、TLS_RSA_WITH_AES_256_CBC_SHA、TLS_RSA_WITH_AES_256_CBC_SHA256) には、 IBM JRE 6.0 SR13 FP2、7.0 SR4 FP2 またはそれ以上が必要です。
8. これらの CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA、TLS_RSA_WITH_DES_CBC_SHA、TLS_RSA_WITH_RC4_128_SHA256) は、SSLv3 または TLS を使用できます。 デフォルトでは、FIPS が有効でなければ、SSLv3 が使用されます。 TLS を使用するには、Java System Property **com.ibm.mq.cfg.preferTLS** を `true` に設定します。
9. この CipherSpec の TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

関連情報

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

MQdev blog: MQ Java, TLS Ciphers, Non-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837

MQdev blog: The relationship between MQ CipherSpecs and Java Cipher Suites

WebSphere MQ classes for Java アプリケーションの実行

クライアント・モードまたはバインディング・モードのいずれかを使用してアプリケーション (main () メソッドを含むクラス) を作成する場合は、Java インタープリターを使用してプログラムを実行します。

次のコマンドを使用します。

```
java -Djava.library.path=library_path MyClass
```

ここで、 *library_path* は、 WebSphere MQ classes for Java ライブラリーへのパスです (WebSphere MQ classes for Java ライブラリーを参照)。

WebSphere MQ classes for Java 環境依存の動作

WebSphere MQ classes for Java を使用すると、さまざまなバージョンの WebSphere MQ に対して実行できるアプリケーションを作成できます。 このトピック集では、これらの異なるバージョンに依存する Java クラスの動作について説明します。

WebSphere MQ classes for Java は、すべての環境で一貫性のある機能と動作を提供するクラスのコアを提供します。 このコアの外部の機能は、アプリケーションの接続先のキュー・マネージャーの機能によって異なります。

ここで注記される点以外には、示される振る舞いは、それぞれのキュー・マネージャー用の「アプリケーション・プログラミング・リファレンス」で説明されているとおりです。

WebSphere MQ classes for Java のコア・クラス

WebSphere MQ classes for Java には、すべての環境で使用できるクラスのコア・セットが含まれています。

以下に示すクラスのコア・セットはコア・クラスと見なされ、716 ページの『 WebSphere MQ classes for Java のコア・クラスの制約事項とバリエーション』にリストされているわずかなバリエーションだけで、すべての環境で使用できます。

- MQEnvironment
- MQException
- MQGetMessageOptions

  以下は除外します。

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  以下は除外します。

  - inquire()
  - set()
- MQMessage

  以下は除外します。

  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength

- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  以下は除外します。

  – knownDestCount
  – unknownDestCount
  – invalidDestCount
  – recordFields
- MQProcess
- MQQueue
- MQQueueManager

  以下は除外します。

  – begin()
  – accessDistributionList()
- MQSimpleConnectionManager
- MQTopic
- MQC

注 :

1. 一部の定数はコアには含まれていないため (詳細については、716 ページの『 WebSphere MQ classes for Java のコア・クラスの制約事項とバリエーション』を参照)、完全に移植可能なプログラム内ではそれらの定数を使用しないでください。
2. プラットフォームによっては、一部の接続モードがサポートされていない場合があります。 このようなプラットフォームでは、サポートされているモードに関連するコア・クラスとオプションのみが使用できます。 (660 ページの『 WebSphere MQ classes for Java の接続オプション』を参照してください)。

WebSphere MQ classes for Java のコア・クラスの制約事項とバリエーション

通常、同等の MQI 呼び出しに環境の違いがある場合でも、コア・クラスはすべての環境で一貫した振る舞いをします。 その振る舞いは、以下の小さな制限とバリエーションを例外として、Windows、UNIX または Linux WebSphere MQ キュー・マネージャーが使用される場合と同様になります。

WebSphere MQ classes for Java での *MQGMO_ * * 値の制約事項

特定の MQGMO_* 値は、すべてのキュー・マネージャーでサポートされているわけではありません。

以下の MQGMO_* 値を使用すると、MQQueue.get() から MQException がスローされる可能性があります。

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP

MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

また、MQGMO_SET_SIGNAL は、Java から使用する場合はサポートされません。

WebSphere MQ classes for Java での *MQPMRF_ * * 値の制約事項
これらは、メッセージを配布リストに書き込むときにのみ使用されるため、配布リストをサポートするキュー・マネージャーのみにサポートされます。 例えば、z/OS キュー・マネージャーは配布リストをサポートしません。

WebSphere MQ classes for Java での *MQPMO_ * * 値の制約事項
一部の MQPMO_* 値は、すべてのキュー・マネージャーでサポートされているわけではありません。

以下の MQPMO_* 値を使用すると、MQQueue.put() または MQQueueManager.put() から MQException がスローされる可能性があります。

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

WebSphere MQ classes for Java における *MQCNO_ * * 値の制約事項とバリエーション
特定の MQCNO_* 値はサポートされていません。

- 自動クライアント再接続は、WebSphere MQ classes for Java ではサポートされていません。 MQCNO_RECONNECT_* をどのような値に設定しても、接続時の動作は MQCNO_RECONNECT_DISABLED と設定された場合と同じになります。
- MQCNO_FASTPATH は、MQCNO_FASTPATH をサポートしていないキュー・マネージャーでは無視されます。 クライアント接続でも、同様に無視されます。

WebSphere MQ classes for Java での *MQRO_ * * 値の制約事項
以下のレポート・オプションを設定できます。

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

詳しくは、Report を参照してください。

WebSphere MQ classes for Java のコア・クラス外のフィーチャー

WebSphere MQ classes for Java には、すべてのキュー・マネージャーでサポートされているわけではない API 拡張機能を使用するように特別に設計された特定の機能が含まれています。 このトピックのコレクションでは、それらの機能をサポートしない キュー・マネージャーを使用するときの、それらの機能の動作方法について説明します。

MQQueueManager コンストラクター・オプションでのバリエーション
MQQueueManager コンストラクターの中には、オプションの整数引数が含まれているものもあります。 この引数の値の中には、すべてのプラットフォームでは受け入れられないものがあります。

MQQueueManager コンストラクターにオプションの整数引数が含まれる場合、それは MQI の MQCNO オプション・フィールドにマップされ、標準接続とファスト・パス接続を切り替えるために使用されます。 コンストラクターのこの拡張形式は、使用されたオプションが MQCNO_STANDARD_BINDING または MQCNO_FASTPATH_BINDING の場合のみ、すべての環境で受け入れられます。 その他のオプションはすべて、コンストラクターが MQRC_OPTIONS_ERROR によって失敗する原因になります。 ファースト・パス・オプション CMQC.MQCNO_FASTPATH_BINDING は、このオプションをサポートするキュー・マネージャーへのバインディング接続で使用する場合にのみ受け付けられます。 他の環境では、無視されます。

MQQueueManager.begin() メソッドでの制限
このメソッドは、バインディング・モードの UNIX、Linux、または Windows システム上の WebSphere MQ キュー・マネージャーに対してのみ使用できます。 それ以外の場合は、失敗して MQRC_ENVIRONMENT_ERROR が出されます。

詳細については、704 ページの『 WebSphere MQ classes for Java を使用した JTA/JDBC 調整』を参照してください。

MQGetMessageOptions フィールドでのバリエーション
キュー・マネージャーによっては、バージョン 2 MQGMO 構造体をサポートしないものがあるため、いくつかのフィールドはデフォルト値に設定しなければなりません。

バージョン 2 MQGMO 構造体をサポートしないキュー・マネージャーを使用する場合、以下のフィールドはデフォルト値に設定したままにしておかなければなりません。

GroupStatus
SegmentStatus
Segmentation

また、MatchOptions フィールドは、MQMO_MATCH_MSG_ID および MQMO_MATCH_CORREL_ID のみをサポートします。 サポートされない値をこれらのフィールドに指定すると、それ以降の MQDestination.get() は失敗し、MQRC_GMO_ERROR が出されます。 キュー・マネージャーがバージョン 2 MQGMO 構造をサポートしない場合、これらのフィールドは正常な MQDestination.get() の後に更新されません。

WebSphere MQ classes for Java における配布リストの制約事項
すべてのキュー・マネージャーで MQDistributionList をオープンできるわけではありません。

以下のクラスは、配布リストを作成するために使用されます。

MQDistributionList
MQDistributionListItem
MQMessageTracker

どのような環境でも MQDistributionLists および MQDistributionListItems を作成して取り込めますが、すべてのキュー・マネージャーで MQDistributionList をオープンできるわけではありません。 特に、z/OS キュー・マネージャーは配布リストをサポートしません。 そのようなキュー・マネージャーを使用して MQDistributionList をオープンしようとすると、MQRC_OD_ERROR が出されます。

MQPutMessageOptions フィールドでのバリエーション
キュー・マネージャーが配布リストをサポートしない場合、特定の MQPMO フィールドの扱いが異なります。

MQPMO の 4 つのフィールドは、MQPutMessageOptions クラスの次の 4 つのメンバー変数として提供されます。

knownDestCount
unknownDestCount
invalidDestCount
recordFields

これらのフィールドは、主に配布リストでの使用を目的にしたものです。 しかし、配布リストをサポートするキュー・マネージャーは、単一キューへの MQPUT 後に DestCount フィールドも指定します。 例えば、キューがローカル・キューに解決されると、knownDestCount は 1 に設定され、その他の 2 つのカウント・フィールドは 0 に設定されます。

キュー・マネージャーが配布リストをサポートしない場合、これらの値は以下のようにシミュレートされます。

- put() が成功した場合には、unknownDestCount が 1 に設定され、その他は 0 に設定されます。
- put() が失敗した場合には、invalidDestCount が 1 に設定され、その他は 0 に設定されます。

recordFields 変数は配布リストで使用されます。 値は、環境とは関係なく、いつでも recordFields に書き込むことができます。 MQPutMessageOptions オブジェクトが、MQDistributionList.put() ではなく、後続の MQDestination.put() または MQQueueManager.put() で使用される場合、その値は無視されます。

WebSphere MQ classes for Java での MQMD フィールドの制約事項
セグメンテーションをサポートしないキュー・マネージャーを使用する場合には、メッセージのセグメンテーションに関連した特定の MQMD フィールドはデフォルト値のままにしておく必要があります。

以下の MQMD フィールドは、メッセージのセグメント化と大きく関係しています。

GroupId
MsgSeqNumber
オフセット
MsgFlags
OriginalLength

アプリケーションでこれらの MQMD フィールドのいずれかをデフォルト以外の値に設定した後、これらの値をサポートしないキュー・マネージャーに put() または get() を出す場合、put() または get() によって MQException が出され、MQRC_MD_ERROR になります。 そのようなキュー・マネージャーでの正常な put() または get() では、常に、MQMD フィールドがそのデフォルト値に設定されたままとなります。 グループ化またはセグメント化されたメッセージを、メッセージのグループ化およびセグメント化をサポートしないキュー・マネージャーに対して実行される Java アプリケーションに送信しないでください。

Java アプリケーションが、これらのフィールドをサポートしないキュー・マネージャーからメッセージを get () しようとし、取得される物理メッセージがセグメント化されたメッセージのグループの一部である (つまり、MQMD フィールドにデフォルト以外の値がある) 場合、そのメッセージはエラーなしで取得されます。 しかし、MQMessage の MQMD フィールドは更新されず、MQMessage の format プロパティーは MQFMT_MD_EXTENSION に設定され、実際のメッセージ・データには、新規フィールドの値が入っている MQMDE 構造体によって接頭部が付けられます。

CICS Transaction Server での WebSphere MQ classes for Java の制約事項
CICS Transaction Server for z/OS 環境では、メイン (最初) のスレッドしか CICS または WebSphere MQ 呼び出しを発行できません。

WebSphere MQ JMS クラスを CICS Java アプリケーション内で使用することはサポートされていないことに注意してください。

これは、この環境では、スレッド間で MQQueueManager オブジェクトや MQQueue オブジェクトを共用したり、子スレッドに新しい MQQueueManager を作成したりできないからです。

Java プラットフォーム Enterprise Edition 内での Java アプリケーションの IBM WebSphere MQ クラスの実行
Java EE で IBM WebSphere MQ classes for Java を使用する前に考慮しなければならない制約事項と設計上の考慮事項がいくつかあります。

IBM WebSphere MQ classes for Java には、 Java EE 環境内で使用する場合の制約事項があります。 また、Java EE 環境内で実行される IBM WebSphere MQ classes for Java アプリケーションを設計、実装、および管理する際には、考慮すべき追加の考慮事項があります。 これらの制限および考慮事項の概要は以下のセクションで説明します。

JTA トランザクションの制限

IBM WebSphere MQ classes for Java を使用するアプリケーション用にサポートされている唯一のトランザクション・マネージャーは、IBM WebSphere MQ 自体です。 JTA 制御下のアプリケーションは IBM WebSphere MQ classes for Java を利用できますが、これらのクラスを通して実行された作業は JTA 作業単位によっては制御されません。 代わりに、それらの作業は、JTA インターフェースを通してアプリケーション・サーバーにより管理される作業単位とは別のローカル作業単位を形成します。 特に、JTA トランザクションのロールバックによって、送受信されたメッセージのロールバックが生じることはありません。 この制限は、アプリケーションまたは Bean 管理トランザクション、コンテナー管理トランザクション、およびすべての Java EE コンテナーに適用されます。 メッセージング処理を直接 IBM WebSphere MQ で、ア

プリケーション・サーバーによって調整されるトランザクション内で実行する場合は、代わりに IBM WebSphere MQ classes for JMS を使用しなければなりません。

スレッドの作成

IBM WebSphere MQ classes for Java は、さまざまな操作のためにスレッドを内部で作成します。 例えば、BINDINGS モードで実行してローカル・キュー・マネージャーに直接呼び出しを行う場合、呼び出しは IBM WebSphere MQ classes for Java によって内部で作成される「ワーカー」スレッドで実行されます。 例えば、接続プールから使用されていない接続を消去したり、 終了したパブリッシュ/サブスクライブ・アプリケーションのサブスクリプションを除去したりする他のスレッドも内部で作成されることがあります。

一部の Java EE アプリケーション (例えば、EJB および Web コンテナーで実行されるアプリケーション) では、新規スレッドを作成してはなりません。 その場合、すべての作業はアプリケーション・サーバーによって管理されるメイン・アプリケーション・スレッドで実行されなければなりません。 アプリケーションが IBM WebSphere MQ classes for Java を使用する場合、アプリケーション・サーバーはアプリケーション・コードと IBM WebSphere MQ classes for Java コードを区別できない場合があり、上記で説明したスレッドによって、アプリケーションがコンテナー仕様に準拠しなくなります。 IBM WebSphere MQ classes for JMS は、これらの Java EE 仕様に違反しないため、代わりに使用できます。

セキュリティー制限

アプリケーション・サーバーによって実装されているセキュリティー・ポリシーによっては、新しい制御スレッドの作成や操作など、IBM WebSphere MQ classes for Java API によって実行される特定の操作が実行できなくなる場合があります (前出のセクションの説明を参照)。

例えば、アプリケーション・サーバーは通常、デフォルトで Java Security が無効にされた状態で実行され、アプリケーション・サーバー固有の構成で Java Security を有効にすることができます (アプリケーション・サーバーによっては、Java Security で使用されるポリシーをより詳細に構成することができるものもあります)。 Java セキュリティーが有効にされた場合、IBM WebSphere MQ classes for Java は、アプリケーション・サーバー用に定義された Java セキュリティー・ポリシーのスレッド化規則に違反する可能性があり、正常に機能するために必要なすべてのスレッドを API が作成できなくなる可能性があります。 スレッド管理の問題を予防するために、Java セキュリティーが有効にされている環境での IBM WebSphere MQ classes for Java の使用はサポートされていません。

アプリケーション独立の考慮事項

Java EE 環境内でアプリケーションを実行することの意図された利点は、アプリケーション分離です。 IBM WebSphere MQ classes for Java の設計と実装は、 Java EE 環境の前に行われます。 IBM WebSphere MQ Java のクラスは、アプリケーション分離の概念をサポートしない方法で使用できます。 この領域での考慮事項の具体的な例には次のようなものがあります。

- 以下のような、MQEnvironment クラス内での静的 (JVM プロセス全体) 設定の使用
  - 接続 ID および認証に使用されるユーザー ID およびパスワード
  - クライアント接続に使用されるホスト名、ポート、およびチャネル
  - 保護されたクライアント接続用の SSL 構成

  1 つのアプリケーションの利点のために MQEnvironment プロパティーを変更した場合、同じプロパティーを利用する他のアプリケーションも影響されます。 Java EE などの複数アプリケーション環境で実行する場合、各アプリケーションは、プロセス全体の MQEnvironment クラスで構成されたプロパティーをデフォルトにするのではなく、特定のプロパティー・セットを持つ MQQueueManager オブジェクトを作成することにより、独自の個別構成を使用する必要があります。
- MQEnvironment クラスは、同じ JVM プロセス内で IBM WebSphere MQ classes for Java を使用するアプリケーションすべてにグローバルに作用するいくつかの静的メソッドを導入します。特定のアプリケーションに対してこの動作をオーバーライドする方法はありません。 この例には、以下のものが含まれます。
  - 鍵ストアの場所などの、SSL プロパティーの構成
  - クライアント・チャネル出口の構成

– 診断トレースの使用可能化または使用不可化
– キュー・マネージャーへの接続の使用を最適化するために使用されるデフォルト接続プールの管理

このようなメソッドを呼び出すと、同じ Java EE 環境で実行されているすべてのアプリケーションに影響します。

- 接続プーリングは、同じキュー・マネージャーへの複数の接続を行うプロセスを最適化するために有効にされています。 デフォルト接続プール・マネージャーはプロセス規模であり、複数のアプリケーションによって共用されます。 したがって、接続プール構成の変更 (MQEnvironment.setDefaultConnectionManager() メソッドを使用して 1 つのアプリケーションのデフォルト接続マネージャーを置き換えるなど) は、同じ Java EE アプリケーション・サーバーで実行されている他のアプリケーションに影響を与えます。
- SSL は、IBM WebSphere MQ classes for Java を使用するアプリケーション用に、MQEnvironment クラスおよび MQQueueManager オブジェクト・プロパティーを使用して構成されます。 アプリケーション・サーバー自体の管理セキュリティー構成には統合されません。 IBM WebSphere MQ classes for Java を適切に構成することによって必要なレベルのセキュリティーが提供されるようにし、アプリケーション・サーバーの構成を使用しないようにしてください。

バインディング・モードの制約事項

IBM WebSphere MQ と WebSphere Application Server は、キュー・マネージャーのメジャー・バージョンと WebSphere Application Server に付属する IBM WebSphere MQ リソース・アダプター (RA) のメジャー・バージョンが異なるように、同じマシンにインストールできます。 例えば、IBM WebSphere MQ RA レベル 7.0.1 が付属する WebSphere Application Server Version 7.0 は、Version 6.0 キュー・マネージャーと同じマシン上にインストールできます。

キュー・マネージャーとリソース・アダプターの主要なバージョンが異なる場合、バインディング接続は使用できません。 リソース・アダプターを使用した WebSphere Application Server からキュー・マネージャーへの接続はすべて、クライアント・タイプ接続を使用する必要があります。 バージョンが同じ場合は、バインディング接続を使用できます。

WebSphere MQ classes for JMS の使用

WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) は、 WebSphere MQ に付属する JMS プロバイダーです。 javax.jms パッケージで定義されたインターフェースの実装に加えて、WebSphere MQ classes for JMS は JMS API への 2 セットの拡張機能を提供します。

JMS の仕様では、アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットが定義されています。 javax.jms パッケージは JMS インターフェースを定義しており、JMS プロバイダーはそのインターフェースを特定のメッセージング製品に実装します。 WebSphere MQ バージョン 7.5 は現在、JMS 1.1 仕様を使用しています。 WebSphere MQ classes for JMS は、WebSphere MQ 用の JMS インターフェースを実装する JMS プロバイダーです。

JMS の仕様では、ConnectionFactory オブジェクトと Destination オブジェクトが管理対象オブジェクトであることが想定されます。 管理者は、中央リポジトリーで管理対象オブジェクトを作成および保守します。 JMS アプリケーションは、Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。 WebSphere MQ classes for JMS は、管理対象オブジェクトの使用をサポートします。管理者は、 WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーのいずれかを使用して、管理対象オブジェクトを作成および保守できます。

WebSphere MQ classes for JMS は JMS API に対する 2 セットの拡張機能も提供します。 それらの拡張機能は主に、接続ファクトリーおよび宛先を実行時に動的に作成および構成することに重点が置かれていますが、メッセージングと直接的には関係のない機能 (例えば問題判別のための機能) も提供します。

WebSphere MQ JMS 拡張機能

以前のリリースの WebSphere MQ classes for JMS には、MQConnectionFactory、 MQQueue、および MQTopic などのオブジェクトに実装されている拡張機能が含まれています。 これらのオブジェクトには WebSphere MQ に固有のプロパティーやメソッドがあります。 オブジェクトは管理対象オブジェクトにすることができます。あるいは、アプリケーションはオブジェクトを実行時に動的に作成することができます。 このリリースの WebSphere MQ classes for JMS ではこれらの拡張機能が保持されてお

り、現在は、WebSphere MQ JMS 拡張機能と呼ばれています。 これらの拡張機能を使用するアプリケーションはすべて、引き続き変更せずに使用できます。

IBM JMS 拡張機能

このリリースの WebSphere MQ classes for JMS は、JMS API へのさらに汎用的な拡張機能のセットを提供しています。これはメッセージング・システムとしての WebSphere MQ に固有なものではありません。 これらの拡張は IBM JMS 拡張機能と呼ばれており、次のような幅広い目的があります。

- IBM JMS プロバイダー全体により高いレベルの整合性を持たせること
- 2 つの IBM メッセージング・システム間のブリッジ・アプリケーションをより簡単に作成できるようにすること
- 1 つの IBM JMS プロバイダーから別のプロバイダーにアプリケーションをより簡単に移植できるようにすること

拡張機能は、Message Service Client for C/C++ および Message Service Client for .NET で提供されるのと似た機能を提供します。

WebSphere MQ classes for JMS を使用する理由

WebSphere MQ classes for JMS を使用するメリットをまとめると、以下のようになります。

- JMS のスキルを再利用できます。

  WebSphere MQ classes for JMS は、WebSphere MQ 用の JMS インターフェースをメッセージング・システムとして実装する JMS プロバイダーです。 組織が WebSphere MQ を初めて使用するものの、JMS アプリケーション開発スキルは既に持っている場合、WebSphere MQ に付属する他の API のいずれかを使用するよりも、使い慣れた JMS API を使用して WebSphere MQ リソースにアクセスする方が容易であることに気付くでしょう。

- JMS は、 Java Platform, Enterprise Edition (Java EE) の不可欠な部分です。

  JMS は、 Java EE プラットフォームでメッセージングに使用する自然 API です。 Java EE に準拠するすべてのアプリケーション・サーバーには、JMS プロバイダーが組み込まれている必要があります。 JMS は、アプリケーション・クライアント、サーブレット、 JavaServer ページ (JSP)、Enterprise Java Bean (EJB)、およびメッセージ駆動型 Bean (MDB) で使用することができます。 特に、 Java EE アプリケーションは MDB を使用してメッセージを非同期に処理し、すべてのメッセージは JMS メッセージとして MDB に送信されることに注意してください。

- 管理者は、JMS 管理対象オブジェクトを中央リポジトリーで作成および保守できます。 WebSphere MQ classes for JMS アプリケーションは、Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得できます。

  JMS 接続ファクトリーおよび宛先は、キュー・マネージャー名、チャネル名、接続オプション、キュー名、およびトピック名などの WebSphere MQ に固有の情報をカプセル化します。 接続ファクトリーおよび宛先が管理対象オブジェクトとして保管される場合、この情報はアプリケーションにハードコーディングされません。 このため、この調整により、アプリケーションは基盤となる WebSphere MQ 構成からある程度の独立性を保持することができます。

- JMS は、アプリケーションの移植性を提供する業界標準の API です。

  JMS アプリケーションは、管理対象オブジェクトとして保管される接続ファクトリーおよび宛先を JNDI を使用して取得し、javax.jms パッケージで定義されたインターフェースだけを使用してメッセージング操作を実行します。 そうすると、アプリケーションは WebSphere MQ classes for JMS などの、どの JMS プロバイダーからも完全に独立し、アプリケーションを変更しなくても、ある JMS プロバイダーから別の JMS プロバイダーに移植することができます。

  JNDI が特定のアプリケーション環境で使用できない場合、WebSphere MQ classes for JMS アプリケーションは JMS API への拡張機能を使用して、実行時に接続ファクトリーおよび宛先を動的に作成および構成できます。 そうすると、アプリケーションは完全に自己完結型となりますが、WebSphere MQ classes for JMS に JMS プロバイダーとして結合されます。

- ブリッジ・アプリケーションは、JMS を使用して作成する方が簡単です。

  ブリッジ・アプリケーションとは、メッセージをあるメッセージング・システムから受信し、それを別のメッセージング・システムに送信するアプリケーションです。 プロダクト固有の API およびメッセージ

形式を使用してブリッジ・アプリケーションを作成することは、複雑になる可能性があります。 その代わりに、2 つの JMS プロバイダー (各メッセージング・システムに 1 つずつ) を使用して、ブリッジ・アプリケーションを作成できます。 そうすると、アプリケーションは 1 つの API、つまり JMS API のみを使用し、JMS メッセージのみを処理します。

WebSphere MQ classes for JMS の概要

このトピックでは、WebSphere MQ classes for JMS の概要、および WebSphere MQ classes for JMS を使用するために事前に必要な知識について説明します。

WebSphere MQ classes for JMS の前提条件

WebSphere MQ classes for JMS アプリケーションを開発し、実行するには、前提条件としていくつかのソフトウェア・コンポーネントが必要です。

WebSphere MQ classes for JMS の前提条件の最新情報については、WebSphere MQ の README ファイルを参照してください。

WebSphere MQ classes for JMS アプリケーションを開発するには、Java 2 Software Development Kit (SDK) が必要です。 ご使用のオペレーティング・システムでサポートされている JDK の詳細については、「WebSphere MQ System requirements」ページに記載されています。 WebSphere MQ の要件を参照してください。

WebSphere MQ classes for JMS アプリケーションを実行するには、以下のソフトウェア・コンポーネントが必要です。

- WebSphere MQ キュー・マネージャー
- Java ランタイム環境 (JRE) (アプリケーションを実行するシステムごとに)

FIPS 140-2 認定の暗号モジュールを使用するために SSL 接続が必要な場合は、 IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) が必要です。 バージョン 5 以降のすべての IBM Java 2 SDK および JRE には IBMJSSEFIPS が含まれています。

Internet Protocol バージョン 6 (IPv6) アドレスは、ご使用の WebSphere MQ classes for JMS アプリケーションで使用できます。 IPv6 アドレスは、ご使用の Java 仮想マシン (JVM) およびご使用のオペレーティング・システム上の TCP/IP 実装でサポートされます。 WebSphere MQ JMS 管理ツール (938 ページの『WebSphere MQ JMS 管理ツールの使用』を参照) でも IPv6 アドレスを使用できます。

WebSphere MQ JMS 管理ツールおよび WebSphere MQ エクスプローラーは、Java Naming and Directory Interface (JNDI) を使用して、管理対象オブジェクトを保管するディレクトリー・サービスにアクセスします。 WebSphere MQ classes for JMS アプリケーションは、JNDI を使用してディレクトリー・サービスから管理対象オブジェクトを取得することもできます。 サービス・プロバイダーは、JNDI 呼び出しをディレクトリー・サービスの呼び出しにマップすることによりディレクトリー・サービスへのアクセスを提供するコードです。 WebSphere MQ classes for JMS には、以下のサービス・プロバイダーが付属しています。

- Lightweight Directory Access Protocol (LDAP) サービス・プロバイダー。これはファイル ldap.jar および providerutil.jar にあります。 LDAP サービス・プロバイダーは、LDAP サーバーに基づくディレクトリー・サービスへのアクセスを提供します。
- ファイル・システム・サービス・プロバイダー。これはファイル fscontext.jar および providerutil.jar にあります。 ファイル・システム・サービス・プロバイダーは、ローカル・ファイル・システムに基づくディレクトリー・サービスへのアクセスを提供します。

LDAP サーバーに基づくディレクトリー・サービスを使用する場合は、LDAP サーバーをインストールおよび構成するか、または既存の LDAP サーバーへのアクセスを取得する必要があります。 特に、Java オブジェクトを保管するように LDAP サーバーを構成する必要があります。 LDAP サーバーをインストールおよび構成する方法について詳しくは、サーバーに付属する文書を参照してください。

IBM WebSphere MQ Client for HP Integrity NonStop Server 用の JMS プログラムの作成

このトピックでは、IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) のための JMS プログラムを開発および実行する前に理解しておくべき事柄を説明します。

JMS 用の IBM WebSphere MQ クラスは、IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) のインストール済み環境の一部としてインストールされます。 インストールの内容の要約について、詳しくはファイル・システムを参照してください。

一部のクライアント機能は、ホスト・オペレーティング・システムに固有です。 IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) でサポートされる機能について詳しくは、 IBM WebSphere MQ クライアント (HP Integrity NonStop Server 用) でサポートされる環境と機能を参照してください。

前提条件

JMS アプリケーションを作成して実行するには、*HP Integrity NonStop Server for Java* コンポーネントがインストールされ、使用可能になっている必要があります。

セットアップ

JMS 用の IBM WebSphere MQ クラスを使用できるアプリケーション実行/作成用の環境をセットアップする方法については、728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』を参照してください。

クライアント・アプリケーションからの接続を受け入れるようキュー・マネージャーを構成するのに必要な手順については、776 ページの『WebSphere MQ classes for JMS アプリケーションのインストール後のセットアップ』を参照してください。

JMS 環境用の IBM WebSphere MQ クラスを検証する方法については、779 ページの『WebSphere MQ classes for JMS の Point-to-Point インストール検査テスト』を参照してください。

アプリケーションの作成

JMS アプリケーションの作成の詳細については、809 ページの『WebSphere MQ classes for JMS アプリケーションの作成』を参照してください。

IBM WebSphere MQ JMS 管理ツールの使用について、詳しくは 938 ページの『WebSphere MQ JMS 管理ツールの使用』を参照してください。

サンプル

サンプル・アプリケーションは、インストール済み環境の `opt/mqm/samp/jms` サブディレクトリーに入っています。

サンプルの実行に必要な構成ステップについて、詳しくは 108 ページの『サンプル・プログラムの作成と実行』を参照してください。

問題解決

問題の解決については、800 ページの『IBM WebSphere MQ classes for JMS に関する問題の解決』を参照してください。

WebSphere MQ classes for JMS のインストールと構成

このセクションでは、WebSphere MQ classes for JMS のインストール時に作成されるディレクトリーとファイルについて説明し、インストール後に WebSphere MQ classes for JMS を構成する方法を示します。

関連概念

726 ページの『IBM WebSphere MQ classes for JMS のインストール内容』
IBM WebSphere MQ classes for JMS をインストールすると、いくつかのファイルとディレクトリーが作成されます。 Windows では、インストールの際に、自動的に環境変数を設定することによっていくつかの構成が実行されます。 その他のプラットフォーム、および特定の Windows 環境では、 IBM WebSphere MQ classes for JMS アプリケーションを実行する前に環境変数を設定する必要があります。

735 ページの『Java セキュリティー・マネージャーの下での WebSphere MQ classes for JMS アプリケーションの実行』

WebSphere MQ classes for JMS は、 Java セキュリティー・マネージャーを使用可能にして実行できます。セキュリティー・マネージャーを使用可能にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java 仮想マシン (JVM) を構成する必要があります。

739 ページの『IBM WebSphere MQ リソース・アダプター』
リソース・アダプターによって、アプリケーション・サーバーで実行されているアプリケーションが IBM WebSphere MQ リソースにアクセスできます。 インバウンド通信とアウトバウンド通信をサポートします。

776 ページの『WebSphere MQ classes for JMS アプリケーションのインストール後のセットアップ』
このトピックでは、WebSphere MQ classes for JMS アプリケーションがキュー・マネージャーのリソースにアクセスするために必要な権限について説明します。 また、接続モードの概要を示し、アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成する方法を説明します。

779 ページの『WebSphere MQ classes for JMS の Point-to-Point インストール検査テスト』
Point-to-Point インストール検査テスト (IVT) プログラムは、 WebSphere MQ classes for JMS に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、メッセージを SYSTEM.DEFAULT.LOCAL.QUEUE というキューに送信した後、メッセージをキューから受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

783 ページの『WebSphere MQ classes for JMS のパブリッシュ/サブスクライブ・インストール検査テスト』
パブリッシュ/サブスクライブ・インストール検査テスト (IVT) プログラムは、 WebSphere MQ classes for JMS に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、トピックにサブスクライブし、メッセージをトピックにパブリッシュした後、そのメッセージを受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』
IVT プログラムは EAR ファイルとして提供されます。 このプログラムを使用するには、このプログラムをデプロイし、JCA リソースとして一部のオブジェクトを定義する必要があります。

757 ページの『アウトバウンド通信のリソース・アダプターの構成』
アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

799 ページの『OSGi のサポート』
OSGi は、バンドルの形によるアプリケーションのデプロイメントをサポートするフレームワークを提供します。 9 つの OSGi バンドルが、IBM WebSphere MQ classes for JMS の一部として提供されます。

800 ページの『IBM WebSphere MQ classes for JMS に関する問題の解決』
インストール検査プログラムを実行し、トレースおよびログ機能を使用して、問題を調べることができます。

関連タスク

789 ページの『WAS CE での MQ リソース・アダプターのインストールとテスト』
WebSphere Application Server CE で IBM WebSphere MQ リソース・アダプターをインストールし、インストール検査テスト (IVT) アプリケーションを実行します。

791 ページの『カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ』
別のキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に、WebSphere Application Server CE で IVT アプリケーションおよび関連スクリプトを変更する必要があります。

794 ページの『カスタム IBM WebSphere MQ 環境を使用する JBoss での IVT アプリケーションのデプロイ』
JBoss に IBM WebSphere MQ リソース・アダプターをインストールするときに、別のキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に、まず JBoss で IVT アプリケーションおよび関連スクリプトを変更する必要があります。

IBM WebSphere MQ リソース・アダプターの問題判別

関連資料

798 ページの『WebSphere MQ classes for JMS に付属するスクリプト』
WebSphere MQ classes for JMS の使用時に実行する必要がある共通タスクを支援するために、多数のスクリプトが提供されています。

IBM WebSphere MQ classes for JMS のインストール内容

IBM WebSphere MQ classes for JMS をインストールすると、いくつかのファイルとディレクトリーが作成されます。 Windows では、インストールの際に、自動的に環境変数を設定することによっていくつかの構成が実行されます。 その他のプラットフォーム、および特定の Windows 環境では、 IBM WebSphere MQ classes for JMS アプリケーションを実行する前に環境変数を設定する必要があります。

ほとんどのオペレーティング・システムでは、 IBM WebSphere MQ classes for JMS は、 IBM WebSphere MQ のインストール時にオプションのコンポーネントとしてインストールされます。 IBM WebSphere MQ Client for HP Integrity NonStop Server の場合、 IBM WebSphere MQ classes for JMS はデフォルトでインストールされます。 IBM WebSphere MQ のインストールについて詳しくは、以下を参照してください。

WebSphere MQ サーバーのインストール
IBM WebSphere MQ クライアントのインストール

726 ページの表 90 は、各プラットフォームで IBM WebSphere MQ classes for JMS ファイルがインストールされる場所を示しています。

表 *90. IBM WebSphere MQ classes for JMS* のインストール・ディレクトリー

プラットフォーム	**ディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/java
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/java
HP-UX、 Linux、および Solaris	*MQ_INSTALLATION_PATH*/java
Windows	*MQ_INSTALLATION_PATH*¥java
MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。	

インストール・ディレクトリーには、以下のものが含まれます。

- IBM WebSphere MQ classes for JMS JAR ファイル。 *MQ_INSTALLATION_PATH*¥ java¥ lib ディレクトリーにあります。
- IBM WebSphere MQ ネイティブ・ライブラリー。Java ネイティブ・インターフェースを使用するアプリケーションによって使用されます。

  32 ビット・ネイティブ・ライブラリーは *MQ_INSTALLATION_PATH*¥ java¥ lib ディレクトリーにインストールされ、64 ビット・ネイティブ・ライブラリーは *MQ_INSTALLATION_PATH*\java\lib64 ディレクトリーにあります。

  IBM WebSphere MQ ネイティブ・ライブラリーの詳細については、730 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照してください。
- 追加スクリプト (798 ページの『WebSphere MQ classes for JMS に付属するスクリプト』を参照)。 これらのスクリプトは、 *MQ_INSTALLATION_PATH*¥ java¥ bin ディレクトリーにあります。
- JMS API 用の IBM WebSphere MQ クラスの仕様。 API の仕様を含む HTML ページの生成には Javadoc ツールが使用されています。

  HTML ページは、 *MQ_INSTALLATION_PATH*¥ java¥ doc¥ WMQJMSClasses ディレクトリーにあります。

  UNIX、 Linux、および Windows システムの場合、このサブディレクトリーには個々の HTML ページが含まれます。

- OGSi のサポート。 OSGi バンドルは java¥lib¥OSGi ディレクトリーにインストールされます (799 ページの『OSGi のサポート』 を参照)。
- IBM WebSphere MQ リソース・アダプター。これは、JCA 1.5 (またはそれ以降) 準拠の任意のアプリケーション・サーバーにデプロイすることができます。

  IBM WebSphere MQ リソース・アダプターは、 *MQ_INSTALLATION_PATH*¥ java¥ lib¥ jca ディレクトリーにあります。詳しくは、 739 ページの『IBM WebSphere MQ リソース・アダプター』 を参照してください。
- Windows では、デバッグに使用できるシンボルは、*MQ_INSTALLATION_PATH*¥ java¥ lib¥ symbols ディレクトリーにインストールされます。

インストール・ディレクトリーには、他の IBM WebSphere MQ コンポーネントに属するファイルも含まれています。 ディレクトリーは、以下のとおりです。

- SOAP 用の JMS トランスポートを提供する IBM WebSphere MQ transport for SOAP は、*MQ_INSTALLATION_PATH*¥ java¥ lib¥ soap ディレクトリーにインストールされます。 IBM WebSphere MQ transport for SOAP の詳細については、 950 ページの『WebSphere MQ transport for SOAP』 について説明しているインフォメーション・センターのセクションを参照してください。
- 分散プラットフォームでは、 IBM WebSphere MQ Bridge for HTTP は *MQ_INSTALLATION_PATH*¥ java¥ lib¥ http ディレクトリーにインストールされます。 IBM WebSphere MQ Bridge for HTTP について詳しくは、 1028 ページの『WebSphere MQ bridge for HTTP』 について説明しているインフォメーション・センターのセクションを参照してください。

IBM WebSphere MQ classes for JMS には、いくつかのサンプル・アプリケーションが用意されています。 727 ページの表 91 に、各プラットフォームでのサンプル・アプリケーションのインストール先を示します。

表 *91.* サンプル・ディレクトリー

プラットフォーム	**ディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/samp/jms
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/samp/jms
HP-UX、 Linux、および Solaris	*MQ_INSTALLATION_PATH*/samp/jms
Windows	*MQ_INSTALLATION_PATH*¥tools¥jms
MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。	

インストールの後、アプリケーションをコンパイルおよび実行するためには、いくつかの構成タスクを実行しなければならない場合があります。

728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』 では、単純な IBM WebSphere MQ classes for JMS アプリケーションを実行するために必要なクラスパスについて説明します。 このトピックでは、特殊な状況で参照する必要がある追加の JAR ファイル、および IBM WebSphere MQ classes for JMS で提供されるスクリプトを実行するために設定する必要がある環境変数についても説明します。

IBM WebSphere MQ classes for JMS アプリケーションを Java 以外の言語で作成されたコードにリンクする必要がある場合 (例えば、キュー・マネージャーへの接続時にバインディング・トランスポートを使用する場合)、 730 ページの『Java Native Interface (JNI) ライブラリーの構成』 で、Java コマンドのパラメーターとして指定する Java Native Interface (JNI) ライブラリーの場所を見つける方法を説明しています。

アプリケーションのトレースやロギングなどのプロパティーを制御するためには、構成プロパティー・ファイルを用意する必要があります。 IBM WebSphere MQ classes for JMS 構成プロパティー・ファイルについては、 732 ページの『IBM WebSphere MQ classes for JMS 構成ファイル』 で説明されています。

WebSphere MQ classes for JMS の JAR ファイルのインストールおよびアップグレード

IBM WebSphere MQ classes for JMS JAR ファイルをシステムにインストールするためにサポートされている唯一の方法は、IBM WebSphere MQ 製品または WebSphere MQ V7.5 クライアント SupportPac-

MQC75 のいずれかをインストールするか、 Apache Maven 734 ページの『IBM WebSphere MQ classes for JMS とソフトウェア管理ツール』などのソフトウェア管理ツールを使用することです。

ソフトウェア管理ツールを使用している場合を除き、 IBM WebSphere MQ classes for JMS JAR ファイルまたはネイティブ・ライブラリーを、他のマシンや、 IBM WebSphere MQ classes for JMS がインストールされているマシン上の別の場所に移動したり、コピーしたりしないでください。

- フィックスパックは、JAR ファイルが別のマシンからコピーされた「インストール済み環境」に適用できません。 これは、すべての JAR ファイルが相互に同期を保ち、互換性のあるレベルに保つようにすることが難しくなるためです。
- マシン間で IBM WebSphere MQ classes for JMS JAR ファイルをコピーすると、同じマシンに複数のファイル・コピーが存在することになり、コードの保守や問題のデバッグに問題が発生する可能性があります。
- dspmqver コマンドは、IBM WebSphere MQ インストール済み環境からのバージョン情報を表示するために使用され、 `\java\lib` ディレクトリーにインストールされている IBM WebSphere MQ classes for JMS のバージョン情報のみを表示します。

  ファイルの複数のコピーが同じマシン上にある場合、**dspmqver** を実行しても、アプリケーションによって使用されている IBM WebSphere MQ classes for JMS のバージョンに関する正確な情報が得られない場合があります。

IBM WebSphere MQ classes for JMS JAR ファイルは、アプリケーション・アーカイブ (エンタープライズ・アプリケーション・アーカイブや EAR ファイルなど) 内に含めないでください。

- IBM WebSphere MQ classes for JMS に対する更新は、 IBM WebSphere MQ フィックスパックを使用して適用することはできません。
- IBM サポートが、アプリケーションによって使用されている IBM WebSphere MQ classes for JMS のバージョンを簡単に判別することはできません。
- 複数のバージョンの IBM WebSphere MQ classes for JMS が Java ランタイム環境に同時にロードされるため、同じ Java ランタイム環境内で実行されている複数のアプリケーションに異なるバージョンの IBM WebSphere MQ classes for JMS が含まれていると、問題が発生する可能性があります。

  これらの問題の例には、以下のような例外があります。

```
java.lang.ClassCastException :
com.ibm.mq.jmqi.system.JmqiSystemEnvironment incompatible with
com.ibm.mq.jmqi.system.JmqiSystemEnvironment

java.lang.ClassCastException :
com.ibm.mq.jms.MQQueue incompatible with com.ibm.mq.jms.MQQueue
```

- アプリケーションが BINDINGS トランスポートを使用してキュー・マネージャーに接続する場合、キュー・マネージャーへのメジャー・アップグレードでは、対応するレベルの IBM WebSphere MQ classes for JMS を含むようにアプリケーションを更新する必要もあります。

  例えば、キュー・マネージャーが IBM WebSphere MQ バージョン 7.5 レベルにアップグレードされた場合、BINDINGS トランスポートを使用してキュー・マネージャーに接続するすべてのアプリケーションも、 IBM WebSphere MQ バージョン 7.5 classes for JMS を含むように更新する必要があります。

IBM WebSphere MQ classes for JMS で使用される環境変数

IBM WebSphere MQ classes for JMS アプリケーションをコンパイルして実行する前に、CLASSPATH 環境変数の設定に IBM WebSphere MQ classes for JMS Java アーカイブ (JAR) ファイルを含める必要があります。 要件によっては、他にも JAR ファイルをクラス・パスに追加しなければならない場合もあります。 IBM WebSphere MQ classes for JMS に付属するスクリプトを実行するには、他の環境変数を設定する必要があります。

IBM WebSphere MQ classes for JMS アプリケーションをコンパイルして実行するには、729 ページの表 92 に示されているように、ご使用のプラットフォームの CLASSPATH 設定を使用します。 設定にはサンプル・ディレクトリーを含めて、IBM WebSphere MQ classes for JMS サンプル・アプリケーションをコンパイルして実行できるようにします。 別の方法として、環境変数を使用する代わりに、**java** コマンドでクラス・パスを指定することもできます。

表 92. *IBM WebSphere MQ classes for JMS* アプリケーションをコンパイルして実行するための、サンプル・アプリケーションを含めた *CLASSPATH* 設定

プラットフォーム	**CLASSPATH 設定**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP Integrity NonStop Server	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP-UX、 Linux、および Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*¥tools¥jms;
z/OS	CLASSPATH=*MQ_INSTALLATION_PATH*/mqm/V7R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V6R0M0/java/samples/jms:
MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。	

JAR ファイル com.ibm.mqjms.jar のマニフェストには、IBM WebSphere MQ classes for JMS アプリケーションが必要とするその他のほとんどの JAR ファイルへの参照が含まれています。そのため、それらの JAR ファイルをクラス・パスに追加する必要はありません。 これらの JAR ファイルには、ディレクトリー・サービスから管理対象オブジェクトを取得するために Java Naming and Directory Interface (JNDI) を使用するアプリケーション、および Java Transaction API (JTA) を使用するアプリケーションが必要とする JAR ファイルが含まれます。

ただし、以下のような場合には、追加の JAR ファイルをクラス・パスに組み込む必要があります。

- com.ibm.mq.exits パッケージで定義されているチャネル出口インターフェースではなく、 com.ibm.mq パッケージで定義されているチャネル出口インターフェースを実装するチャネル出口クラスを使用している場合、 IBM WebSphere MQ classes for Java JAR ファイル com.ibm.mq.jar をクラス・パスに追加する必要があります。
- バージョン 1.4.2 の Java 2 Software Development Kit (SDK) を使用して Java コードをコンパイルする場合は、以下の JAR ファイルをクラスパスに追加する必要があります。
  - jms.jar
  - com.ibm.mq.jmqi.jar

  また、アプリケーションが JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出す場合は、以下の JAR ファイルもクラス・パスに追加する必要があります。
  - fscontext.jar
  - jndi.jar
  - ldap.jar
  - providerutil.jar

  さらに、アプリケーションが JTA を使用する場合は、jta.jar もクラス・パスに追加する必要があります。

これらの追加 JAR ファイルはアプリケーションのコンパイルでのみ必要になり、実行には必要ありません。

-Xlint オプションを使用してコンパイルする場合に、com.ibm.mq.ese.jar が存在しないという警告メッセージが表示されることがあります。 この警告は無視して構いません。 このファイルは、Extended Security Edition をインストールしている場合にのみ存在します。

IBM WebSphere MQ classes for JMS に付属しているスクリプトは、以下の環境変数を使用します。

MQ_JAVA_DATA_PATH
この環境変数は、ログおよびトレース出力のディレクトリーを指定します。

MQ_JAVA_INSTALL_PATH
この環境変数は、WebSphere MQ classes for JMS がインストールされるディレクトリーを指定します。

MQ_JAVA_LIB_PATH
この環境変数は、 731 ページの表 93 に示すように、 WebSphere MQ classes for JMS ライブラリーが保管されるディレクトリーを指定します。

Windows では、すべての環境変数がインストール時に自動的に設定されます。 その他のプラットフォームでは、ユーザーが自分でこれらを設定しなければなりません。

UNIX、HP Integrity NonStop Server、 または Linux の各システムで 32 ビットの JVM 使用しているときに環境変数を設定する場合は、スクリプト setjmsenv を使用できます。 UNIX システムまたは Linux システムで 64 ビットの JVM を使用しているときに環境変数を設定する場合は、スクリプト setjmsenv64 を使用できます。 これらのスクリプトは、 *MQ_INSTALLATION_PATH*/java/bin ディレクトリーにあります。 *MQ_INSTALLATION_PATH* は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。

setjmsenv スクリプトまたは setjmsenv64 スクリプトは、さまざまな方法で使用できます。表に示した必須環境変数を設定する際の基礎として使用したり、テキスト・エディターを使用して .profile に追加したりできます。 一般的ではない設定を使用している場合は、必要に応じてスクリプトの内容を編集してください。 または、JMS 始動スクリプトが実行されるセッションごとにこのスクリプトを実行することもできます。 このオプションを選択した場合は、JMS 検査プロセス中に . ./setjmsenv または . ./setjmsenv64 を入力して、開始するすべてのシェル・ウィンドウでスクリプトを実行する必要があります。

Java Native Interface (JNI) ライブラリーの構成

バインディング・トランスポートを使用してキュー・マネージャーに接続するか、クライアント・トランスポートを使用してキュー・マネージャーに接続し、Java 以外の言語で作成されたチャネル出口プログラムを使用する IBM WebSphere MQ classes for JMS アプリケーションは、Java Native Interface (JNI) ライブラリーにアクセスする環境で実行する必要があります。

このタスクについて

この環境をセットアップするには、IBM WebSphere MQ classes for JMS アプリケーションを開始する前に Java 仮想マシン (JVM) が mqjbnd ライブラリーをロードできるように、環境のライブラリー・パスを構成する必要があります。

IBM WebSphere MQ は、以下の 2 つの Java Native Interface (JNI) ライブラリーを提供します。

mqjbnd
このライブラリーは、バインディング・トランスポートを使用してキュー・マネージャーに接続するアプリケーションが使用します。 これは、 IBM WebSphere MQ classes for JMS とキュー・マネージャーの間のインターフェースを提供します。 IBM WebSphere MQ バージョン 7.5 と共にインストールされた mqjbnd ライブラリーは、任意の IBM WebSphere MQ バージョン 7.5 (またはそれ以前) のキュー・マネージャーに接続するために使用できます。

mqjexitstub02
mqjexitstub02 ライブラリーは、アプリケーションがクライアント・トランスポートを使用してキュー・マネージャーに接続し、Java 以外の言語で作成されたチャネル出口プログラムを使用するときに、IBM WebSphere MQ classes for JMS によってロードされます。

特定のプラットフォームでは、IBM WebSphere MQ は、これらの JNI ライブラリーの 32 ビット・バージョンと 64 ビット・バージョンをインストールします。 各プラットフォームでのライブラリーの場所を、表 1 に記載します。

表 *93.* 各プラットフォーム用の *IBM WebSphere MQ classes for JMS* ライブラリーの場所

プラットフォーム	**IBM WebSphere MQ classes for JMS ライブラリーを含むディレクトリー**
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
HP-UX	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Linux (パワー , x86-64 および zSeries s390x プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Linux (x86 プラットフォーム) Linux (zSeries プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib
Solaris (x86-64 および SPARC プラットフォーム)	*MQ_INSTALLATION_PATH*/java/lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*/java/lib64 (64 ビット・ライブラリー)
Windows	*MQ_INSTALLATION_PATH*¥ java¥ lib (32 ビット・ライブラリー) *MQ_INSTALLATION_PATH*¥java¥lib64 (64 ビット・ライブラリー)
MQ_INSTALLATION_PATH は、IBM WebSphere MQ がインストールされている上位ディレクトリーを表します。	

手順

1. JVM の **java.library.path** プロパティーを次の 2 つの方法のうちいずれかを実行して構成します。
   - 次の例に示されているように JVM 引数を指定します。

     ```
     -Djava.library.path=<path_to_library_directory>
     ```

     Linux 例えば、64 ビット JVM (Linux) のデフォルト場所におけるインストールの場合、次のように指定します。

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

- JVM が独自の java.library.path をセットアップするなどのように、シェル環境を構成します。このパスはプラットフォーム、および IBM WebSphere MQ のインストール場所によって異なります。例えば、64 ビットの JVM で、デフォルトの IBM WebSphere MQ インストール場所の場合、次の設定を使用できます。

  
  ```
  export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
  ```

  
  ```
  export LD_LIBRARY_PATH=/opt/mqm/java/
  lib64:$LD_LIBRARY_PATH
  ```

  
  ```
  set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
  ```

  環境が適切に構成されていない場合に表示される例外スタックの例を次に示します。

```
Caused by: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Failed to load the WebSphere MQ native JNI library: 'mqjbnd'.
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    at com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    at java.security.AccessController.doPrivileged(AccessController.java:400)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    at com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    at com.ibm.mq.jmqi.local.LocalMQ.<init>(LocalMQ.java:1350)
    at com.ibm.mq.jmqi.local.LocalServer.<init>(LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    at com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    at com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    at
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 more
Caused by: java.lang.UnsatisfiedLinkError: mqjbnd (Not found in java.library.path)
    at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    at java.lang.System.loadLibrary(System.java:534)
    at com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 more
```

2. 32 ビット環境または 64 ビット環境のいずれかをセットアップした後、以下のコマンドを使用して IBM WebSphere MQ classes for JMS アプリケーションを開始します。

   ```
   java application-name
   ```

   ここで、 *application-name* は、実行する IBM WebSphere MQ classes for JMS アプリケーションの名前です。

   IBM WebSphere MQ 理由コード 2495 (MQRC_MODULE_NOT_FOUND) を含む例外は、以下の場合に IBM WebSphere MQ classes for JMS によってスローされます。

   - 32 ビット Java ランタイム環境では 64 ビット Java ネイティブ・ライブラリーをロードできないため、 IBM WebSphere MQ classes for JMS アプリケーションは 32 ビット Java ランタイム環境で実行され、 IBM WebSphere MQ classes for JMS 用に 64 ビット環境がセットアップされています。
   - IBM WebSphere MQ classes for JMS アプリケーションは 64 ビット Java ランタイム環境で実行されます。また、64 ビット Java ランタイム環境は 32 ビット Java ネイティブ・ライブラリーをロードできないため、32 ビット環境は IBM WebSphere MQ classes for JMS 用にセットアップされています。

IBM WebSphere MQ classes for JMS 構成ファイル

WebSphere MQ classes for JMS 構成ファイルは、WebSphere MQ classes for JMS を構成するために使用されるプロパティーを指定します。

WebSphere MQ classes for JMS 構成ファイルのフォーマットは、標準 Java プロパティー・ファイルのフォーマットです。 jms.config という名前のサンプル構成ファイルが、WebSphere MQ classes for JMS インス

トール・ディレクトリーの bin サブディレクトリーに用意されています。 このファイル文書には、サポートされているすべてのプロパティーとそれらのデフォルト値が記録されています。

WebSphere MQ classes for JMS 構成ファイルの名前と場所を選ぶことができます。 アプリケーションを開始するときに、以下のフォーマットで **java** コマンドを使用してください。

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

コマンド中、*config_file_url* は WebSphere MQ classes for JMS 構成ファイルの名前と場所を指定する Uniform Resource Locator (URL) です。 次のタイプの URL (http、file、ftp、および jar) がサポートされています。

次に、**java** コマンドの例を示します。

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

このコマンドは、WebSphere MQ classes for JMS 構成ファイルを、ローカル Windows システムの D:¥mydir¥mjms.config ファイルとして識別します。

アプリケーションが開始すると、WebSphere MQ classes for JMS は構成ファイルの内容を読み取り、指定されたプロパティーを内部のプロパティー・ストアに保管します。 **java** コマンドが構成ファイルを識別しない場合、または構成ファイルが見つからない場合、WebSphere MQ classes for JMS はすべてのプロパティーについてデフォルト値を使用します。 **java** コマンドでシステム・プロパティーとして指定することにより、必要に応じて構成ファイル中のプロパティーを指定変更できます。

WebSphere MQ classes for JMS 構成ファイルは、アプリケーションとキュー・マネージャーまたはブローカーとの間のサポートされているいずれのトランスポートとも、共に使用することができます。

WebSphere MQ classes for JMS 構成ファイルにプロパティーを設定することによって、始動トレースを指定できないことに注意してください。 始動トレースは、次の例に示すようにして **java** コマンドでシステム・プロパティーを設定することによってのみ指定できます。

```
java -Dcom.ibm.msg.client.commonservices.trace.startup=true
     -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
      MyAppClass
```

WebSphere MQ MQI クライアント構成ファイルで指定されたプロパティーの指定変更

WebSphere MQ MQI クライアント構成ファイルは、WebSphere MQ classes for JMS を構成するために使用されるプロパティーを指定することもできます。 ただし、WebSphere MQ MQI クライアント構成ファイルで指定されたプロパティーは、アプリケーションがクライアント・モードでキュー・マネージャーに接続しているときにのみ適用されます。

WebSphere MQ classes for JMS 構成ファイルでプロパティーとして指定することにより、必要に応じて WebSphere MQ MQI クライアント構成ファイル中の属性を指定変更できます。 WebSphere MQ MQI クライアント構成ファイル中の属性を指定変更するには、WebSphere MQ classes for JMS 構成ファイルで次のフォーマットのエントリーを使用します。

```
com.ibm.mq.cfg.stanza.propName=propValue
```

エントリー中の変数には、以下の意味があります。

stanza
属性を含んでいる WebSphere MQ MQI クライアント構成ファイル中のスタンザの名前

propName
WebSphere MQ MQI クライアント構成ファイルで指定されている属性の名前

propValue
WebSphere MQ MQI クライアント構成ファイルで指定されている属性の値を指定変更するプロパティーの値

あるいは、**java** コマンドでシステム・プロパティーとしてプロパティーを指定することにより、WebSphere MQ MQI クライアント構成ファイル内の属性をオーバーライドすることができます。 プロパティーをシステム・プロパティーとして指定するには、前述のフォーマットを使用してください。

WebSphere MQ MQI クライアント構成ファイル中の属性のうち、WebSphere MQ classes for JMS に関係するのは以下のものだけです。 他の属性を指定または指定変更しても、効果はありません。

スタンザ	属性
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath
クライアント構成ファイルの ClientExitPath スタンザ	ExitsDefaultPath64
クライアント構成ファイルの ClientExitPath スタンザ	JavaExitsClasspath
クライアント構成ファイルの MessageBuffer スタンザ	MaximumSize
クライアント構成ファイルの MessageBuffer スタンザ	PurgeTime
クライアント構成ファイルの MessageBuffer スタンザ	UpdatePercentage
クライアント構成ファイルの TCP スタンザ	ClntRcvBufSize
クライアント構成ファイルの TCP スタンザ	ClntSndBufSize
クライアント構成ファイルの TCP スタンザ	Connect_Timeout
クライアント構成ファイルの TCP スタンザ	KeepAlive

IBM WebSphere MQ classes for JMS とソフトウェア管理ツール

Apache Maven などのソフトウェア管理ツールを IBM WebSphere MQ classes for JMS で使用できます。

多くの大規模開発会社がこれらのツールを使用して、サード・パーティー・ライブラリーのリポジトリーを集中管理しています。

IBM WebSphere MQ classes for JMS は、いくつかの JAR ファイルで構成されています。 この API を使用して Java 言語アプリケーションを開発する場合は、アプリケーションを開発するマシンに IBM WebSphere MQ Server、Client、または Client SupportPac をインストールする必要があります。

このようなツールを使用し、IBM WebSphere MQ classes for JMS を構成する JAR ファイルを集中管理リポジトリーに追加する場合は、以下の点を守る必要があります。

- リポジトリーまたはコンテナーは、社内の開発者だけが使用できるようにしなければなりません。 社外に分散させることは許可されません。
- リポジトリーには、単一の IBM WebSphere MQ リリースまたはフィックスパックからの整合した完全な JAR ファイル・セットを入れる必要があります。
- IBM サポートが提供するメンテナンスでリポジトリーを更新する必要があります。

IBM WebSphere MQ Version 7.5 の場合は、以下の JAR ファイルをリポジトリーにインストールする必要があります。

- com.ibm.mqjms.jar.
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- IBM WebSphere MQ classes for JMS を使用する場合は、CL3Export.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用する場合は、CL3Nonexport.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用する場合は、jndi.jar が必要です。

- IBM WebSphere MQ classes for JMS を使用する場合は、ldap.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用する場合は、rmm.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用する場合は、dhbcore.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用する場合は、jms.jar が必要です。
- IBM WebSphere MQ classes for JMS を使用し、ファイル・システムの JNDI コンテキストに保管されている JMS 管理対象オブジェクトにアクセスする場合は、fscontext.jar が必要です。
- providerutil.jar (IBM WebSphere MQ classes for JMS を使用し、ファイル・システムの JNDI コンテキストに保管されている JMS 管理対象オブジェクトにアクセスする場合)。

Java セキュリティー・マネージャーの下での WebSphere MQ classes for JMS アプリケーションの実行

WebSphere MQ classes for JMS は、 Java セキュリティー・マネージャーを使用可能にして実行できます。セキュリティー・マネージャーを使用可能にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java 仮想マシン (JVM) を構成する必要があります。

これを行う最も簡単な方法は、JRE に付属しているポリシー構成ファイルを変更する方法です。 大部分のシステムでは、このファイルは、JRE ディレクトリーの下の相対パス `lib/security/java.policy` に格納されています。 構成ファイルは好みのエディターを使用して編集することも、JRE で提供されている `policytool` プログラムを使用して編集することもできます。

重要: V 7.5.0.8 可能な限り、ホワイトリスト という用語は、許可リスト という用語に置き換えられました。 1 つの例外は、以下の Java システム・プロパティー名です。

アプリケーションで Java セキュリティー・マネージャー・メカニズムを使用する場合は、以下の権限を付与する必要があります。

- 使用する許可リスト・ファイルに対する FilePermission (ENFORCEMENT モードの場合は読み取り権限、DISCOVER モードの場合は書き込み権限を指定)
- com.ibm.mq.jms.whitelist、com.ibm.mq.jms.whitelist.discover、com.ibm.mq.jms.whitelist.mode の各プロパティーに対する PropertyPermission (読み取り)。

ClassName 許可リスティングは、 APAR IT14385 および IBM WebSphere MQ Version 7.5.0、フィックスパック 8 でサポートされます。 詳細については、736 ページの『ClassName JMS ObjectMessage』を参照してください。

以下は、デフォルトのセキュリティー・マネージャーでの WebSphere MQ classes for JMS の正常な実行を可能にするポリシー構成ファイルの 2 つのエントリーの例です。

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "console.encoding","read";
  permission java.lang.RuntimePermission "setContextClassLoader";
```

```
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.*","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
  permission java.util.logging.LoggingPermission "control";
  //Wherever trace output is expected
  permission java.io.FilePermission "/tmp/*","read,write";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
```

注 :

- *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされている上位ディレクトリーを表します。
- 最初の grant ステートメントには WebSphere MQ classes for JMS に必要なアクセス権が含まれており、2 番目の grant ステートメントには WebSphere MQ classes for JMS アプリケーションに必要なアクセス権が含まれています。
- WebSphere MQ classes for JMS がアプリケーションの Java アーカイブ (JAR) ファイルにアクセスできるようにするには、最初の grant ステートメントに以下のアクセス権を追加します。

```
permission java.io.FilePermission "/path_to_your_app/-", "read";
```

- ポリシー構成ファイルでこれらの grant ステートメントを使用するために、WebSphere MQ classes for JMS をインストールした場所とアプリケーションが保管されている場所に応じてパス名を変更する必要がある場合があります。
- WebSphere MQ classes for JMS に付属するサンプル・アプリケーションとそれを実行するスクリプトは、セキュリティー・マネージャーを使用可能にしません。

V 7.5.0.8 ClassName JMS ObjectMessage

V 7.5.0.8 WebSphere MQ classes for JMS では、JMS ObjectMessage インターフェースの実装におけるクラスの許可リスト登録のサポートにより、Java オブジェクト・シリアライゼーションおよびデシリアライゼーションのメカニズムに関連する可能性がある一部のセキュリティー・リスクに対する潜在的な緩和策が提供されます。

注 : 可能な限り、ホワイトリスト という用語は、許可リスト という用語に置き換えられました。 唯一の例外は、このトピックで説明されている Java システム・プロパティー名です。

デシリアライゼーションは任意の Java オブジェクトをインスタンス化するので、送信された悪意のあるデータによってさまざまな問題が発生する可能性があります。 このため、Java オブジェクトのシリアライゼーションおよびデシリアライゼーションのメカニズムは、潜在的なセキュリティー・リスクとして見なされています。 シリアライゼーションの注目すべきアプリケーションの 1 つとして、 Java Message Service (JMS) ObjectMessages では、シリアライゼーションを使用して任意のオブジェクトをカプセル化および転送します。

シリアライゼーションの許可リスティングを使用すると、シリアライゼーションがもたらすリスクの一部を軽減できる可能性があります。 許可リスティングは、ObjectMessages でカプセル化と抽出を実行できるクラスを明示的に指定することで、シリアライゼーションによるリスクから、ある程度保護することができます。

WebSphere MQ classes for JMS での許可リスティング

APAR IT14385 および IBM WebSphere MQ Version 7.5.0、フィックスパック 8 が適用されている場合、WebSphere MQ classes for JMS は、JMS ObjectMessage インターフェースの実装におけるクラスの許可リスト登録をサポートします。 許可リストには、ObjectMessage.setObject() でシリアライズできる Java クラスと、ObjectMessage.getObject() でデシリアライズできる Java クラスを定義します。

許可リストに含まれていないクラスのインスタンスを ObjectMessage でシリアライズ/デシリアライズしようとすると、 java.io.InvalidClassException が原因の javax.jms.MessageFormatException がスローされます。

許可リストの作成

重要 : WebSphere MQ classes for JMS を許可リストとともに配布することはできません。ObjectMessages を使用して転送するクラスの選択は、アプリケーション設計上の選択であるため、IBM WebSphere MQ が事前に設定できることではありません。

そのため、許可リスティングのメカニズムには以下の 2 つの動作モードが用意されています。

DISCOVERY
このモードでは、メカニズムは完全修飾クラス名のリストを生成します。ObjectMessages でのシリアライズ/デシリアライズが検出されたすべてのクラスが報告されます。

ENFORCEMENT
このモードでは、メカニズムは許可リスティングを適用します。許可リストに含まれていないクラスのシリアライズ/デシリアライズの試行は拒否されます。

このメカニズムを使用する場合には、最初に DISCOVERY モードで実行して現在シリアライズ/デシリアライズされているクラスのリストを収集し、 そのリストを確認してから、独自のホワイトリストを作成するための基礎として使用する必要があります。 そのリストを変更せずに使用して良い場合もありますが、そのように決定する前に、まずはリストを確認する必要があります。

許可リスティング・メカニズムの制御

許可リスティング・メカニズムを制御するために、以下の 3 つのシステム・プロパティーが用意されています。

com.ibm.mq.jms.whitelist
このプロパティーは、次のいずれかの方法で指定できます。

- 許可リストのファイルのパス名。ファイル URI 形式 (先頭は `file:`) で指定します。 DISCOVERY モードの場合は、このファイルは許可リスティング・メカニズムによって書き込まれます。 このファイルが存在していてはなりません。 ファイルが存在していると、メカニズムはそのファイルを上書きするのではなく、例外をスローします。 ENFORCEMENT モードの場合は、このファイルは許可リスティング・メカニズムによって読み取られます。
- 許可リストを構成する、完全修飾クラス名をコンマで区切ったもの。

このプロパティーが設定されていない場合、許可リスト・メカニズムは非アクティブになります。

Java セキュリティー・マネージャーを使用する場合は、 WebSphere MQ classes for JMS JAR ファイルに、このファイルに対する読み取り権限と書き込み権限があることを確認する必要があります。

com.ibm.mq.jms.whitelist.discover

- このプロパティーが設定されていないか false に設定されている場合、許可リスト・メカニズムは ENFORCEMENT モードで実行されます。
- このプロパティーが true に設定されていて、許可リストがファイル URI として指定されている場合、許可リスト・メカニズムは DISCOVERY モードで実行されます。
- このプロパティーが true に設定されていて、許可リストがクラス名のリストとして指定されている場合、許可リスト・メカニズムは該当する例外をスローします。
- このプロパティーが true に設定されていて、許可リストが com.ibm.mq.jms.whitelist プロパティーを使用して指定されていない場合、許可リスト・メカニズムは非アクティブになります。
- このプロパティーが true に設定されていて、許可リスト・ファイルが既に存在する場合、許可リスト・メカニズムは java.io.InvalidClassException をスローし、エントリーはファイルに追加されません。

com.ibm.mq.jms.whitelist.mode
このストリング・プロパティーは、以下の 3 つの方法のいずれでも指定できます。

- このプロパティーが SERIALIZE に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.setObject() メソッドでのみ許可リストの妥当性検査が行われます。
- このプロパティーが DESERIALIZE に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.getObject() メソッドでのみ許可リストの妥当性検査が行われます。

- このプロパティーが設定されていない場合、または他の値に設定されている場合、ENFORCEMENT モードでは、ObjectMessage.getObject() メソッドと ObjectMessage.setObject() メソッドの両方で許可リストの妥当性検査が行われます。

許可リスト・ファイルの形式

許可リスト・ファイルの形式の主な特徴を次に示します。

- 許可リスト・ファイルには、プラットフォームに適したデフォルトのファイル・エンコード方式と、プラットフォームの行終了方式が使用されます。

  注 : ファイルを異種システム間で移動する場合には、変換が必要になることがあります。
- 空でない各行には 1 つの完全修飾クラス名が含まれます。 空の行は無視されます。
- コメントを含めることができます。'#' 文字からその行末までの内容はすべて無視されます。
- 以下のような、ごく基本的なワイルドカード・メカニズムを使用できます。
  - '*' はクラス名の**最後の**要素として使用できます。
  - '*' はクラス名の**単一の**要素と一致します。つまりクラスとは一致しますがパッケージの部分とは一致しません。

  そのため、`com.ibm.mq.*` は `com.ibm.mq.MQMessage` と一致しますが `com.ibm.mq.jmqi.remote.api.RemoteFAP` とは一致しません。

  ワイルドカードは、デフォルト・パッケージ内のクラス、つまり明示的なパッケージ名のないクラスに対しては機能しないので、"*" のクラス名は拒否されます。
- 誤った形式の許可リスト・ファイル (例えばワイルドカードが最後の要素ではない `com.ibm.mq.*.Message` などの項目を含むファイル) を使用すると、java.lang.IllegalArgumentException がスローされます。
- 空の許可リスト・ファイルを使用すると、ObjectMessage の使用が完全に無効になります。

コンマ区切りリストとしての許可リストの形式

コンマ区切りリストとしての許可リストでも、同じワイルドカード・メカニズムが使用可能です。

- '*' をコマンド行、シェル・スクリプト、またはバッチ・ファイルに指定すると、オペレーティング・システムによって展開される場合があるので、特別な処理が必要になります。
- '#' コメント文字を使用できるのは、ファイルを指定する場合のみです。 許可リストがクラス名のコンマ区切りリストとして指定されている場合、多くの UNIX または Linux シェルではデフォルトのコメント文字であるため、オペレーティング・システムまたはシェルがそれを処理しないと想定すると、通常の文字として扱われます。

許可リスティングが行われるタイミング

許可リスティングは、アプリケーションが初めて ObjectMessage の setMessage() メソッドまたは getMessage() メソッドを実行したときに開始されます。

メカニズムが初期化されると、システム・プロパティーが評価され、許可リスト・ファイルが開かれ、ENFORCEMENT モードの場合は、許可リストに含まれているクラスのリストが読み込まれます。 この時点で、エントリーがアプリケーションの IBM WebSphere MQ JMS ログ・ファイルに書き込まれます。

メカニズムが初期化されたら、そのパラメーターを変更することはできません。 初期化のタイミングはアプリケーションの動作に依存するため、簡単には予測できません。 したがって、システム・プロパティーの設定値と許可リスト・ファイルの内容は、アプリケーションの開始時から固定であると見なす必要があります。 結果が保証されないため、アプリケーションの実行中にプロパティーや許可リスト・ファイルの内容を変更しないでください。

考慮事項

Java シリアライゼーション・メカニズムに固有のリスクを軽減するための最良のアプローチは、ObjectMessage の代わりに JSON を使用するなど、別のデータ転送方式を検討することです。 IBM WebSphere MQ Advanced Message Security (AMS) メカニズムを使用すると、メッセージの送信元が信頼できるソースであることが保証されるので、セキュリティーを高めることができます。

アプリケーションで Java セキュリティー・マネージャー・メカニズムを使用する場合は、以下の権限を付与する必要があります。

- 使用する許可リスト・ファイルに対する FilePermission (ENFORCEMENT モードの場合は読み取り権限、DISCOVER モードの場合は書き込み権限を指定)
- com.ibm.mq.jms.whitelist、com.ibm.mq.jms.whitelist.discover、com.ibm.mq.jms.whitelist.mode の各プロパティーに対する PropertyPermission (読み取り)。

関連概念

735 ページの『Java セキュリティー・マネージャーの下での WebSphere MQ classes for JMS アプリケーションの実行』
WebSphere MQ classes for JMS は、 Java セキュリティー・マネージャーを使用可能にして実行できます。セキュリティー・マネージャーを使用可能にした状態でアプリケーションを正常に実行するには、適切なポリシー構成ファイルを使用して Java 仮想マシン (JVM) を構成する必要があります。

IBM WebSphere MQ リソース・アダプター

リソース・アダプターによって、アプリケーション・サーバーで実行されているアプリケーションが IBM WebSphere MQ リソースにアクセスできます。 インバウンド通信とアウトバウンド通信をサポートします。

Java Platform, Enterprise Edition (Java EE) コネクター・アーキテクチャー (JCA) は、 Java EE 環境で実行されているアプリケーションを IBM WebSphere MQ または Db2 などのエンタープライズ情報システム (EIS) に接続する標準的な方法を提供します。 IBM WebSphere MQ リソース・アダプターには JCA 1.5 インターフェースが実装されており、IBM WebSphere MQ classes for JMS が含まれています。 このリソース・アダプターは、アプリケーション・サーバーで実行される JMS アプリケーションと Message Driven Beans (MDB) が、IBM WebSphere MQ キュー・マネージャーのリソースにアクセスできるようにするためのものです。 リソース・アダプターは Point-to-Point ドメインとパブリッシュ/サブスクライブ・ドメインの両方をサポートします。

IBM WebSphere MQ リソース・アダプターは、アプリケーションとキュー・マネージャーの間の 2 つのタイプの通信をサポートします。

アウトバウンド通信
: アプリケーションはキュー・マネージャーへの接続を開始し、同期的に JMS メッセージを JMS 宛先に送信し、JMS 宛先から JMS メッセージを受信します。

インバウンド通信
: JMS 宛先に届く JMS メッセージは MDB に送信され、MDB はメッセージを非同期的に処理します。

IBM WebSphere MQ classes for JMS の詳細については、721 ページの『WebSphere MQ classes for JMS の使用』を参照してください。

リソース・アダプターには、IBM WebSphere MQ classes for Java も含まれています。 これらのクラスは、リソース・アダプターがデプロイされているアプリケーション・サーバーで実行されているアプリケーションで自動的に使用可能になり、 IBM WebSphere MQ キュー・マネージャーのリソースにアクセスする際に、そのアプリケーション・サーバーで実行されているアプリケーションが IBM WebSphere MQ classes for Java API を使用できるようになります。 IBM WebSphere MQ classes for Java の詳細については、658 ページの『WebSphere MQ classes for Java の使用』を参照してください。

Java EE 環境内での IBM WebSphere MQ classes for Java の使用はサポートされていますが、制限があります。 これらの制限については、719 ページの『Java プラットフォーム Enterprise Edition 内での Java アプリケーションの IBM WebSphere MQ クラスの実行』を参照してください。

JCA リソース・アダプターをサポートするために必要なその他の資料

JCA リソース・アダプターの構成方法については、使用しているアプリケーション・サーバーの資料を参照してください。

すべてのアプリケーション・サーバーは、独自の管理インターフェースのセットを備えています。 JCA リソースを定義するためのグラフィカル・ユーザー・インターフェースを備えたアプリケーション・サーバーもあれば、XML デプロイメント計画を作成するためのアドミニストレーターを必要とするアプリケーション・サーバーもあります。 したがって、各アプリケーション・サーバー用に WebSphere MQ リソース・アダプターを構成する方法については、この資料では扱われていません。 この資料では、構成する必要がある事柄についてのみ焦点を当てています。 JCA リソース・アダプターの構成方法については、使用しているアプリケーション・サーバーで提供されている資料を参照してください。

この資料を理解するには、JMS および WebSphere MQ classes for JMS に精通している必要があります。 WebSphere MQ リソース・アダプターを構成するために使用されるプロパティーの多くは、WebSphere MQ classes for JMS オブジェクトのプロパティーと同等であり、同じ機能を持っています。

WebSphere MQ リソース・アダプターのインストール

WebSphere MQ リソース・アダプターは、リソース・アーカイブ (RAR) ファイルとして提供されます。 RAR ファイルをアプリケーション・サーバーにインストールします。 ディレクトリーをシステム・パスに追加する必要がある場合があります。

WebSphere MQ リソース・アダプターは、wmq.jmsra.rar というリソース・アーカイブ (RAR) ファイルとして提供されます。 このファイルは、WebSphere MQ classes for JMS とともに、740 ページの表 94 で示されるディレクトリーにインストールされます。

表 *94.* プラットフォームごとの、*wmq.jmsra.rar* を含むディレクトリー

プラットフォーム	**ディレクトリー**
AIX、 HP-UX、 Linux 、および Solaris	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH* ¥java¥lib¥jca
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。	

RAR ファイルは、WebSphere MQ classes for JMS および JCA インターフェースの WebSphere MQ インプリメンテーションを含んでいます。

WebSphere MQ リソース・アダプター RAR ファイルをアプリケーション・サーバーにインストールする必要がありますが、それを行う方法はアプリケーション・サーバーによって異なります。 リソース・アダプター RAR ファイルのインストール方法については、ご使用のアプリケーション・サーバーの資料を参照してください。

UNIX and Linux システム上のバインディング接続の場合、Java Native Interface (JNI) ライブラリーを含むディレクトリーがシステム・パスにあることを確認する必要があります。 このディレクトリーの場所 (そこには、WebSphere MQ classes for JMS ライブラリーも含まれている) については、『731 ページの表 93』を参照してください。 Windows の場合、このディレクトリーは WebSphere MQ classes for JMS をインストールする際に、自動的にシステム・パスに追加されます。

トランザクションは、クライアント・モードとバインディング・モードの両方でサポートされます。

WebSphere MQ リソース・アダプター、およびそのリソース・アダプターによって使用される WebSphere MQ classes for JMS のバージョンは、同じリリース・レベルでなければなりません。

WebSphere Application Server および *WebSphere MQ* リソース・アダプター

WebSphere Application Server バージョン 6 では、 WebSphere MQ リソース・アダプターを使用しないでください。 WebSphere Application Server V7 には、WebSphere MQ V7 リソース・アダプターのバージョンが含まれています。

WebSphere Application Server V6 内では、WebSphere MQ リソース・アダプターを使用しないでください。 WebSphere Application Server 内から WebSphere MQ キュー・マネージャーのリソースにアクセスするには、WebSphere MQ メッセージング・プロバイダーを JMS アプリケーションで使用する必要があります。 WebSphere MQ メッセージング・プロバイダーには、WebSphere MQ classes for JMS のバージョンが含まれています。

WebSphere Application Server V7 には、WebSphere MQ V7 リソース・アダプターのバージョンが含まれています。

詳しくは、技術情報『Which version of WebSphere MQ Resource Adapter (RA) is shipped with WebSphere Application Server ?』を参照してください。

WebSphere Application Server Liberty と *IBM WebSphere MQ* リソース・アダプター
IBM WebSphere MQ Version 7.5 リソース・アダプターは、wmqJmsClient-1.1 フィーチャーを使用して、WebSphere Application Server Liberty バージョン 8.5.5、フィックスパック 2 以降にインストールできます。 代わりに、いくつかの制約はありますが、汎用的な Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) サポートを使用してリソース・アダプターをインストールすることもできます。

リソース・アダプターを Liberty にインストールする場合の一般的な制約事項

wmqJmsClient-1.1 フィーチャーを使用する場合も、一般的な JCA サポートを使用する場合も、Version 7.5 リソース・アダプターには以下の制約事項が適用されます。

- IBM WebSphere MQ classes for Java は Liberty ではサポートされません。 これらは、IBM WebSphere MQ Liberty メッセージング・フィーチャーおよび汎用 JCA サポートのどちらとも、一緒に使用してはなりません。 詳細については、Using WebSphere MQ Java Interfaces in J2EE/JEE Environments を参照してください。
- IBM WebSphere MQ リソース・アダプターのトランスポート・タイプは BINDINGS_THEN_CLIENT です。このトランスポート・タイプは IBM WebSphere MQ Liberty メッセージング・フィーチャー内でサポートされていません。
- IBM WebSphere MQ Advanced Message Security (IBM WebSphere MQ AMS) フィーチャーは、IBM WebSphere MQ Liberty メッセージング・フィーチャーに含まれていません。

IBM WebSphere MQ Version 7.5 リソース・アダプターは、wmqJmsClient-2.0 フィーチャーと併用できません。

WebSphere MQ リソース・アダプターの構成

WebSphere MQ リソース・アダプターを構成するには、さまざまな JCA リソースおよびシステム・プロパティーを定義してください。

JCA リソースを以下のカテゴリーで定義してください。

- ResourceAdapter オブジェクトのプロパティー。リソース・アダプターのグローバル・プロパティー (診断トレースのレベルなど) を表します。 これらのプロパティーについては、742 ページの『ResourceAdapter オブジェクトの構成』を参照してください。
- ActivationSpec オブジェクトのプロパティー。MDB がインバウンド通信用にアクティブ化される方法を決定します。 これらのプロパティーについては、744 ページの『インバウンド通信のリソース・アダプターの構成』を参照してください。
- ConnectionFactory オブジェクトのプロパティー。アプリケーション・サーバーがアウトバウンド通信用の JMS ConnectionFactory オブジェクトを作成するために使用します。 これらのプロパティーについては、757 ページの『アウトバウンド通信のリソース・アダプターの構成』を参照してください。
- 管理対象宛先オブジェクトのプロパティー。アプリケーション・サーバーがアウトバウンド通信用の JMS Queue オブジェクトまたは JMS Topic オブジェクトを作成するために使用します。 これらのプロパティーについても、757 ページの『アウトバウンド通信のリソース・アダプターの構成』で説明します。

WebSphere MQ リソース・アダプター RAR ファイルには、META-INF/ra.xml というファイルが含まれています。これには、リソース・アダプターのデプロイメント記述子が入っています。 このデプロイメント記述子は、https://java.sun.com/xml/ns/j2ee/connector_1_5.xsd の XML スキーマによって定義され、リソース・アダプターおよびそれが提供するサービスに関する情報が含まれています。 また、アプリケーション・

サーバーもリソース・アダプターのデプロイメント計画を必要とすることがあります。 このデプロイメント計画はアプリケーション・サーバーに固有です。 例えば、WebSphere Application Server Community Edition では、geronimo-ra.xml というデプロイメント計画が必要です。

Secure Sockets Layer (SSL) を使用している場合は、以下の例のように、鍵ストア・ファイルおよびトラストストア・ファイルの場所を JVM システム・プロパティーとして指定してください。

```
java ... -Djavax.net.ssl.keyStore=
key_store_location
         -Djavax.net.ssl.trustStore=trust_store_location
         -Djavax.net.ssl.keyStorePassword=key_store_password
```

これらのプロパティーは、ActivationSpec オブジェクトと ConnectionFactory オブジェクトではプロパティーとして使用することができません。また、1 つのアプリケーション・サーバーに複数の鍵ストアを指定することもできません。 プロパティーは JVM 全体に適用されるため、アプリケーション・サーバーで実行している他のアプリケーションが SSL 接続を使用する場合には、そのアプリケーション・サーバーに影響を与えることがあります。 また、アプリケーション・サーバーはこれらのプロパティーを別の値にリセットすることもあります。 WebSphere MQ classes for JMS での SSL の使用の詳細については、910 ページの『WebSphere MQ classes for JMS での Secure Sockets Layer (SSL) の使用』を参照してください。

インストール検査テスト (IVT) プログラムが WebSphere MQ リソース・アダプターに付属していますが、このプログラムを実行する前に、リソース・アダプターを構成する必要があります。 IVT プログラムを実行するために構成する必要がある事柄については、787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』を参照してください。

リソース・アダプターのログ、警告、およびエラー・メッセージは、IBM WebSphere MQ classes for JMS と同じメカニズムを使用します。詳細については、800 ページの『ロギングおよび IBM WebSphere MQ classes for JMS』を参照してください。 WebSphere Application Server の場合、これらのメッセージはアプリケーション・サーバーの出力ログに自動的にリダイレクトされます。 WAS CE や JBoss などの他のアプリケーション・サーバーの場合、デフォルトでは、 mqjms.log というファイルに移動します。 リソース・アダプターを構成して、警告メッセージをアプリケーション・サーバーの標準出力ログに追加で記録するには、アプリケーション・サーバーの以下の JVM システム・プロパティーを設定します。

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

JVM システム・プロパティーを設定する方法について詳しくは、アプリケーション・サーバーの資料を参照してください。

ResourceAdapter オブジェクトの構成

ResourceAdapter オブジェクトは、WebSphere MQ リソース・アダプターのグローバル・プロパティーをカプセル化します。 リソース・アダプターの機能を使用して、これらのプロパティーを定義します。

ResourceAdapter オブジェクトには次の 2 つのプロパティー・セットがあります。

- 診断トレースに関連したプロパティー
- リソース・アダプターによって管理される接続プールに関連したプロパティー

これらのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。

診断トレースに関連したプロパティーの定義についての詳細は、IBM WebSphere MQ リソース・アダプターのトレースを参照してください。

リソース・アダプターは、MDB へのメッセージの送達に使用される、JMS 接続の内部接続プールを管理します。743 ページの表 95 接続プールに関連付けられている ResourceAdapter オブジェクトのプロパティーをリストします。

表 95. 接続プールに関連する *ResourceAdapter* オブジェクトのプロパティー

プロパティー名	タイプ	デフォルト値	説明
maxConnections	ストリング	50	WebSphere MQ キュー・マネージャーへの接続の最大数およびデプロイ済みの MDB の最大数
connectionConcurrency	ストリング	1	JMS 接続を共有する MDB の最大数。 接続を共有することはできません。このプロパティーの値は常に 1 です。
reconnectionRetryCount	ストリング	5	接続が失敗した場合に、リソース・アダプターが WebSphere MQ キュー・マネージャーに再接続しようとする最大数
reconnectionRetryInterval	ストリング	300 000	リソース・アダプターが WebSphere MQ キュー・マネージャーに再接続しようとする前に待機する時間 (ミリ秒)。
startupRetryCount	ストリング	0	開始時に MDB への接続を試行するデフォルトの回数 (アプリケーション・サーバーの始動時にキュー・マネージャーが実行されていない場合)。
startupRetryInterval	ストリング	30 000	次の接続開始を試行するまでのデフォルトのスリープ時間 (ミリ秒)。

MDB がアプリケーション・サーバーにデプロイされている場合、maxConnection プロパティーで指定された最大接続数を超えていなければ、新規 JMS 接続が作成され、キュー・マネージャーとの会話が開始されます。 このため、MDB の最大数は最大接続数と等しくなります。 デプロイされた MDB の数がこの最大数に達すると、別の MDB をデプロイしようとしても失敗します。 MDB が停止される場合、その接続は別の MDB が使用できます。

一般に、多数の MDB がデプロイされる場合、maxConnections プロパティーの値を大きくする必要があります。

WebSphere MQ キュー・マネージャーへの接続が失敗すると (例えば、ネットワーク障害のため)、reconnectionRetryCount および reconnectionRetryInterval プロパティーがリソース・アダプターの振る舞いを制御します。 接続が失敗すると、リソース・アダプターはその接続によって提供されるすべての MDB に対するメッセージの送達を、reconnectionRetryInterval プロパティーで指定されたインターバルの間、中断します。 その後、リソース・アダプターはキュー・マネージャーに再接続しようとします。 試行が失敗する場合、リソース・アダプターは、reconnectionRetryCount プロパティーによって課せられた制限に達するまで、reconnectionRetryInterval プロパティーで指定された間隔でさらに再接続しようとします。 すべての試行が失敗する場合、MDB が手動で再始動されるまで、送達は永久に停止されます。

一般に、ResourceAdapter オブジェクトは管理を必要としません。 ただし、例えば UNIX and Linux システムで診断トレースを使用可能にするには、以下のプロパティーを設定できます。

```
traceEnabled:       true
traceLevel:         10
```

これらのプロパティーは、リソース・アダプターが開始されていない場合は効果がありません。 例えば、WebSphere MQ リソースを使用するアプリケーションがクライアント・コンテナーでのみ実行している場合が、これに該当します。 この場合、診断トレースのプロパティーを Java 仮想マシン (JVM) システム・プロパティーとして設定できます。 以下の例のように、**java** コマンドで -D フラグを使用することによって、プロパティーを設定することができます。

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

ResourceAdapter オブジェクトのすべてのプロパティーを定義する必要はありません。 未指定のままになっているプロパティーはデフォルト値をとります。 管理された環境では、プロパティーの指定に 2 つの方

法を混用しないほうが良いでしょう。 混用する場合、JVM システム・プロパティーが ResourceAdapter オブジェクトのプロパティーに優先して指定されます。

インバウンド通信のリソース・アダプターの構成

インバウンド通信を構成するには、1 つ以上の ActivationSpec オブジェクトのプロパティーを定義します。

ActivationSpec オブジェクトのプロパティーは、メッセージ駆動型 Bean (MDB) が JMS メッセージを WebSphere MQ キューから受信する方法を決定します。 MDB のトランザクションの振る舞いはデプロイメント記述子で定義されています。

ActivationSpec オブジェクトには次の 2 つのプロパティー・セットがあります。

- WebSphere MQ キュー・マネージャーへの JMS 接続を作成するために使用するプロパティー。
- メッセージが指定されたキューに到着すると、それらを非同期で送達する JMS 接続コンシューマーを作成するために使用するプロパティー。

ActivationSpec オブジェクトのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。

744 ページの表 96 は、WebSphere MQ キュー・マネージャーへの JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティーを示します。

表 *96. JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
applicationName	ストリング	• 起動クラス名が使用可能である場合は、28 文字以内に調整されます。 使用可能でない場合は、ストリング `WebSphere MQ Client for Java` が使用されます。	アプリケーションをキュー・マネージャーに登録する際に使用した名前。 このアプリケーション名は、**DISPLAY CONN MQSC/PCF** コマンド (フィールド名は **APPLTAG**) または IBM WebSphere MQ Explorer の 「**アプリケーション接続**」画面 (フィールド名は **App name**) に表示されます。
brokerCCDurSubQueue[1]	ストリング	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが永続サブスクリプション・メッセージを受信するキューの名前
brokerCCSubQueue[1]	ストリング	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが非永続サブスクリプション・メッセージを受信するキューの名前
brokerControlQueue[1]	ストリング	• **SYSTEM.BROKER.CONTROL.QUEUE** • キュー名	ブローカー制御キューの名前
brokerQueueManager[1]	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	ブローカーが稼働しているキュー・マネージャーの名前
brokerSubQueue[1]	ストリング	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • キュー名	非永続メッセージ・コンシューマーがメッセージを受信するキューの名前

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
brokerVersion[1]	ストリング	• **指定なし** - ブローカーを V6 から V7 に移行した後、このプロパティーを設定し、RFH2 ヘッダーが使用されないようにします。 移行した後、このプロパティーは関係なくなります。 • **V1** - WebSphere MQ パブリッシュ/サブスクライブ・ブローカーを使用する場合。 または、WebSphere MQ Integrator、WebSphere Event Broker、WebSphere Business Integration Event Broker、または WebSphere Business Integration Message Broker のいずれかのブローカーを互換モードで使用する場合。 この値は、TRANSPORT が BIND または CLIENT に設定されている場合のデフォルト値です。 • **V2** - WebSphere MQ Integrator、WebSphere Event Broker、WebSphere Business Integration Event Broker、または WebSphere Business Integration Message Broker のいずれかのブローカーをネイティブ・モードで使用する場合。 この値は、TRANSPORT が DIRECT または DIRECTHTTP に設定されている場合のデフォルト値です。	使用されているブローカーのバージョン
ccdtURL	ストリング	• **null** • Uniform Resource Locator (URL)	クライアント・チャネル定義テーブル (CCDT) が格納されているファイルの名前と場所を識別し、このファイルのアクセス方法を指定する URL。
CCSID	ストリング	• **819** • Java 仮想マシン (JVM) によってサポートされるコード化文字セット ID	接続用のコード化文字セット ID
channel	ストリング	• **SYSTEM.DEF.SVRCONN** • MQI チャネルの名前	使用する MQI チャネルの名前
cleanupInterval[1]	int	• **3 600 000** • 正整数	パブリッシュ/サブスクライブ・クリーンアップ・ユーティリティーがバックグラウンドで実行する間隔 (ミリ秒)。
cleanupLevel[1]	ストリング	• **SAFE** • NONE • strong • FORCE • NONDUR	ブローカー・ベースのサブスクリプション・ストアのクリーンアップ・レベル

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
clientID	ストリング	• **null** • クライアント ID	接続のクライアント ID
cloneSupport	ストリング	• **DISABLED** - 一度に稼働できる永続トピック・サブスクライバーのインスタンスは 1 つだけです。 • ENABLED-同じ永続トピック・サブスクライバーの複数のインスタンスを同時に実行できますが、各インスタンスは別個の Java 仮想マシン (JVM) で実行する必要があります。	1 つの永続トピック・サブスクライバーの複数のインスタンスを同時に稼働できるかどうか
connectionNameList	ストリング	• **localhost(1414)** • コンマで区切られている項目で構成されるストリングです。各項目は以下の形式になります。 `HOSTNAME(PORT)` ここで、*HOSTNAME* は DNS 名または IP アドレスです。	インバウンド通信に使用される TCP/IP 接続名のリスト。 指定すると、**connectionNameList** は **hostname** プロパティーと **port** プロパティーを置き換えます。 このプロパティーを使用して、複数インスタンス・キュー・マネージャーに再接続します。 **connectionNameList** は **localAddress** の形式に類似していますが、混同しないでください。**localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
failIfQuiesce	Boolean	• **true** • false	キュー・マネージャーが静止状態の場合、特定のメソッドの呼び出しが失敗するかどうか
headerCompression	ストリング	• **NONE** • SYSTEM - RLE メッセージ・ヘッダーの圧縮が実行される	接続のヘッダー・データを圧縮するために使用できる技法のリスト

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
hostName	ストリング	• **localhost** • ホスト名 • IP アドレス	キュー・マネージャーが存在しているシステムのホスト名または IP アドレス。 **hostname** プロパティーと **port** プロパティーは、指定されている場合は **connectionNameList** プロパティーに置き換えられます。
localAddress	ストリング	• **null** • 以下の形式のストリング `[host_name][(low_port[,high_port])]` ここで、*host_name* はホスト名または IP アドレス、*low_port* および *high_port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	このプロパティーは、キュー・マネージャーへの接続に対して、以下のいずれかまたは両方を指定します。 • 使用されるローカル・ネットワーク・インターフェース • 使用されるローカル・ポート、またはローカル・ポートの範囲 **localAddress** は **connectionNameList** の形式に類似していますが、混同しないでください。**localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
messageCompression	ストリング	• **NONE** • ブランク文字で区切られた以下の値の 1 つ以上のリスト。 RLE ZLIBFAST ZLIBHIGH	接続のメッセージ・データを圧縮するために使用できる技法のリスト
messageRetention[1]	Boolean	• **true** - 不要なメッセージを入力キューに残す • false - 不要なメッセージは指定されている後処理オプションに従って扱う	入力キューにある不要なメッセージを接続のコンシューマーに保持させるかどうか

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
messageSelection[1]	ストリング	• **CLIENT** • BROKER	メッセージ選択を WebSphere MQ classes for JMS またはブローカーのどちらが行うかを決定します。 ブローカーによるメッセージ選択は、brokerVersion の値が 1 の場合、サポートされません。
パスワード	ストリング	• **null** • パスワード	キュー・マネージャーへの接続を作成する際に使用するデフォルト・パスワード
pollingInterval[1]	int	• **5000** • 任意の正整数	この値は、各セッション内のメッセージ・リスナーのキューに適切なメッセージがない場合に、各メッセージ・リスナーがキューからメッセージの取得を再度試みるまでの最大の時間間隔 (ミリ秒) です。 セッション内のいずれのメッセージ・リスナーでも適切なメッセージがない状態が頻繁に発生する場合は、このプロパティーの値を大きくすることを考えてください。 このプロパティーは、TRANSPORT の値が BIND または CLIENT の場合にのみ使用されます。
port	int	• **1414** • TCP ポート番号	キュー・マネージャーが listen を行うポート。 **hostname** プロパティーと **port** プロパティーは、指定されている場合は **connectionNameList** プロパティーに置き換えられます。
providerVersion	ストリング	• **指定なし** • 以下のいずれかの形式のストリング – V.R.M.F – V.R.M – V.R – V ここで、V、R、M、および F は、ゼロ以上の整数値です。	MDB が接続することになっているキュー・マネージャーのバージョン、リリース、修正レベルおよびフィックスパック。

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
queueManager	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	接続するキュー・マネージャーの名前
receiveExit [3]	ストリング	• **null** • 1 つ以上の項目をコンマで区切ったストリング。各項目は、 WebSphere MQ classes for Java インターフェース MQReceiveExit を実装するクラスの完全修飾名です。	チャネル受信出口プログラム、または連続して実行される一連の受信出口プログラムを識別します。
receiveExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル受信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ
rescanInterval[1]	int	• **5000** • 任意の正整数	Point-to-Point ドメインのメッセージ・コンシューマーがメッセージ・セレクターを使用して受信するメッセージを選択する場合、 WebSphere MQ classes for JMS は WebSphere MQ キューを検索して、このキューの *MsgDeliverySequence* 属性によって決定される順序で適切なメッセージを探します。 WebSphere MQ classes for JMS が適切なメッセージを検出してそれをコンシューマーに送信すると、 WebSphere MQ classes for JMS は、キュー内の現在位置から次の適切なメッセージの検索を再開します。 WebSphere MQ classes for JMS は、キューの終わりに達するまで、またはこのプロパティーの値によって決定される時間間隔 (ミリ秒) が経過するまで、この方法でキューの検索を続行します。 いずれの場合も、 WebSphere MQ classes for JMS はキューの先頭に戻って検索を続行し、これにより新しい時間間隔が開始します。
securityExit [3]	ストリング	• **null** • WebSphere MQ classes for Java インターフェース、 MQSecurityExit を実装するクラスの完全修飾名。	チャネル・セキュリティー出口プログラムを識別します。

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
securityExitInit	ストリング	• **null** • ユーザー・データのストリング	チャネル・セキュリティー出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ
sendExit [3]	ストリング	• **null** • 1 つ以上の項目をコンマで区切ったストリング。各項目は、 WebSphere MQ classes for Java インターフェース MQSendExit を実装するクラスの完全修飾名です。	チャネル送信出口プログラム、または連続して実行される一連の送信出口プログラムを識別します。
sendExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル送信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ
shareConvAllowed	Boolean	• NO-クライアント接続はそのソケットを共用できません。 • **YES** -クライアント接続はそのソケットを共用できます。	チャネル定義が一致する場合に、クライアント接続がそのソケットを同じプロセスから同じキュー・マネージャーへの他のトップレベル JMS 接続と共用できるかどうか
sparseSubscriptions[1]	Boolean	• **false** - サブスクリプションが頻繁なマッチング・メッセージを受信する。 • true - サブスクリプションが頻繁でないマッチング・メッセージを受信する。 この値は、サブスクリプション・キューがブラウズ用にオープンできることを必要とします。	TopicSubscriber オブジェクトのメッセージ検索ポリシーを制御する
sslCertStores	ストリング	• **null** • ブランクで区切られた 1 つ以上の LDAP URL のストリング。 各 LDAP URL の形式は次のとおりです。 `ldap://host_name[:port]` ここで、*host_name* はホスト名または IP アドレス、*port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	SSL 接続で使用する証明書取り消しリスト (CRL) を保持する Lightweight Directory Access Protocol (LDAP) サーバー
sslCipherSuite	ストリング	• **null** • CipherSuite の名前	SSL 接続で使用する CipherSuite
sslFipsRequired[2]	Boolean	• **false** • true	IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を SSL 接続で使用する必要があるかどうか

表 96. *JMS* 接続を作成するために使用される *ActivationSpec* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
sslPeerName	ストリング	• **null** • 識別名のテンプレート	SSL 接続で、キュー・マネージャーで提供されるデジタル証明書の識別名を確認するために使用するテンプレート
sslResetCount	int	• **0** • 0 から 999 999 999 の範囲の整数	SSL によって使用された秘密鍵の再ネゴシエーションの前に、SSL 接続で送信および受信したバイトの総数
sslSocketFactory	ストリング	javax.net.ssl.SSLSocketFactory インターフェースのインプリメンテーションを提供するクラスの、完全修飾クラス名を表すストリング。コンストラクター・メソッドに渡される引数を、括弧内に含めることもできます。	管理されたオブジェクトの有効範囲内で設定されたすべての接続は、SSLSocketFactory インターフェースのインプリメンテーションから得られたソケットを使用する。
statusRefreshInterval[1]	int	• **60000** • 任意の正整数	サブスクライバーがキュー・マネージャーとの接続を失ったときに検出する長期実行トランザクションのリフレッシュの間隔 (ミリ秒)。 このプロパティーは、subscriptionStore の値が QUEUE の場合にのみ使用されます。
subscriptionStore[1]	ストリング	• **ブローカー** • MIGRATE • QUEUE	WebSphere MQ classes for JMS がアクティブ・サブスクリプションに関する永続データを保管する場所を決定します。
transportType	ストリング	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	キュー・マネージャーへの接続で、クライアント・モードまたはバインディング・モードのどちらを使用するか。BINDINGS_THEN_CLIENT という値が指定されると、リソース・アダプターはまずバインディング・モードで接続を試みます。 この接続が失敗すると、リソース・アダプターはクライアント・モード接続を試行します。
ユーザー名	ストリング	• **null** • ユーザー名	キュー・マネージャーへの接続を作成する際に使用するデフォルト・ユーザー名

表 96. JMS 接続を作成するために使用される ActivationSpec オブジェクトのプロパティー (続き)			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
wildcardFormat	ストリング	• CHAR- ブローカー・バージョン 1 で使用された文字ワイルドカードのみ認識します。 • **TOPIC** - ブローカー・バージョン 2 で使用されたトピック・レベル・ワイルドカードのみ認識します。	使用されるワイルドカード構文のバージョン

注 :

1. このプロパティーは、WebSphere MQ classes for JMS のバージョン 7.0 で使用できます。providerVersion プロパティーのバージョン番号が 7 より小さく設定されていない限り、バージョン 7.0 キュー・マネージャーに接続されたアプリケーションに影響を及ぼしません。
2. sslFipsRequired プロパティーの使用の重要な詳細については、775 ページの『IBM WebSphere MQ リソース・アダプターの制限』を参照してください。
3. 出口を見つけることができるようにリソース・アダプターを構成する方法については、917 ページの『チャネル出口を使用するように IBM WebSphere MQ classes for JMS を構成する』を参照してください。

752 ページの表 97 は、JMS 接続コンシューマーを作成するために使用される ActivationSpec オブジェクトのプロパティーを示します。

表 97. JMS 接続コンシューマーを作成するために使用される ActivationSpec オブジェクトのプロパティー			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
destination	ストリング	宛先名	メッセージを受信する宛先。 useJNDI プロパティーは、このプロパティーの値がどのように解釈されるのかを判別します。
destinationType	ストリング	• javax.jms.Queue • javax.jms.Topic	宛先のタイプ。キュー、またはトピック。
maxMessages	int	• **1** • 正整数	サーバー・セッションに一度に割り当てることができるメッセージの最大数。アクティベーション・スペックが XA トランザクションで MDB にメッセージを送達する場合、このプロパティーの設定に関係なく、値 1 が使用されます。
maxPoolDepth	int	• **10** • 正整数	接続コンシューマーによって使用されるサーバー・セッション・プール内のサーバー・セッションの最大数
messageSelector	ストリング	• **null** • SQL92 メッセージ・セレクター式	送達されるメッセージを指定するメッセージ・セレクター式

表 97. *JMS* 接続コンシューマーを作成するために使用される *ActivationSpec* オブジェクトのプロパティー *(続き)*

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
nonASFTimeout	int	• **0** • 正整数	正の値は、非 ASF 送達が使用されることを示します。 この値は、読み取り要求で、まだ到着していない可能性のあるメッセージを待機する時間 (ミリ秒単位) です (待機を伴う読み取りの呼び出し)。デフォルト値 0 は ASF 送達が使用されることを示します。 このパラメーターは、アプリケーションが WebSphere Application Server バージョン 7 以降で実行している場合にのみ有効です。
nonASFRollbackEnabled	Boolean	• **false** - MDB で障害が起きても、メッセージはコンシュームされます。 • true - MDB 内で障害が起きたときは、メッセージがキューにロールバックされます。	MDB が非トランザクションである場合に、メッセージ送達が WebSphere MQ 同期点内で行われるかどうかを指定します。 MDB がトランザクションであるか、または nonASFTimeout が 0 に設定されている場合は、無視されます。
poolTimeout	int	• **300000** • 正整数	未使用のサーバー・セッションが、非アクティブ状態が原因でクローズされる前に、サーバー・セッション・プールでオープンしたままになっている時間 (ミリ秒)
readAheadAllowed	int	• **DESTINATION** - キュー定義またはトピック定義を参照することによって先読みが許可されるかどうかを判別します。 • DISABLED - 先読みは許可されない • ENABLED - 先読みは許可される • QUEUE - キュー定義を参照することによって先読みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって先読みが許可されるかどうかを判別します。	MDB が先読みを使用して、この宛先から内部バッファーへの非永続メッセージを受信する前に、それらを取得することを許可されるかどうか

表 97. *JMS* 接続コンシューマーを作成するために使用される *ActivationSpec* オブジェクトのプロパティー *(続き)*

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
readAheadClosePolicy	int	• **ALL** - 内部先読みバッファー内のすべてのメッセージは、停止する前に MDB に送信されます。 • CURRENT - 現行の MDB 呼び出しのみが完了します。内部先読みバッファー内にメッセージが残される可能性がありますが、それらは破棄されます。	管理者が MDB を停止すると、内部先読みバッファー内のメッセージに何が生じるか。
receiveCCSID	int	• **0** - JVM `Charset.defaultCharset` を使用します。 • 1208 - UTF-8 • サポートされるコード化文字セット ID	キュー・マネージャー・メッセージ変換のターゲット CCSID を 設定する宛先プロパティー。 **receiveConversion** が QMGR に設定されていない場合、値は無視されます。
receiveConversion	ストリング	• **CLIENT_MSG** • QMGR	キュー・マネージャーによりデータ変換を実行するかどうかを決定する宛先プロパティー。
startTimeout	int	• **10 000** • 正整数	MDB へのメッセージの送達処理がスケジュールされた後でその送達を開始しなければならない時間 (ミリ秒)。 この時間が経過すると、メッセージはキュー上にロールバックされます。
subscriptionDurability	ストリング	• **NonDurable** - 非永続サブスクリプションが、トピックにサブスクライブする MDB にメッセージを送達するために使用されます。 • Durable - 永続サブスクリプションが、トピックにサブスクライブする MDB にメッセージを送達するために使用されます。	トピックにサブスクライブする MDB にメッセージを送達するために、永続サブスクリプションが使用されるか、それとも非永続サブスクライブが使用されるか
subscriptionName	ストリング	• **"" (空ストリング)** • サブスクリプション名	永続サブスクリプションの名前

表 97. *JMS* 接続コンシューマーを作成するために使用される *ActivationSpec* オブジェクトのプロパティー *(続き)*

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
useJNDI	Boolean	• **false** - destination というプロパティーが、WebSphere MQ キューまたはトピックの名前として解釈されます。 • true - destination というプロパティーが、アプリケーション・サーバーの JNDI ネーム・スペース内で javax.jms.Queue オブジェクトまたは javax.jms.Topic オブジェクトの名前として解釈されます。	destination というプロパティーの値がどのように解釈されるのかを決定します。

destination および destinationType という ActivationSpec プロパティーは、明示的に定義される必要があります。 その他のプロパティーはすべてオプションです。

ActivationSpec オブジェクトは、競合するプロパティーを持つことができます。 例えば、バインディング・モードの接続に SSL プロパティーを指定できます。 この場合、振る舞いは移送タイプおよびメッセージング・ドメインによって決定されます。これは、destinationType プロパティーで判別された Point-to-Point かパブリッシュ/サブスクライブのいずれかです。 指定された移送タイプまたはメッセージング・ドメインに該当しないプロパティーは無視されます。

あるプロパティーを定義する際にその他のプロパティーを定義する必要があるが、それらの他のプロパティーを定義していない場合、ActivationSpec オブジェクトは、MDB のデプロイメント時にその validate() メソッドが呼び出される際に InvalidPropertyException 例外をスローします。 例外は、アプリケーション・サーバーに応じた仕方で、アプリケーション・サーバーの管理者に報告されます。 例えば、subscriptionDurability プロパティーを Durable (永続サブスクリプションを使用することを示す) に設定する場合、subscriptionName プロパティーも定義する必要があります。

ccdtURL および channel というプロパティーが両方とも定義されている場合、InvalidPropertyException 例外がスローされます。 ただし、ccdtURL プロパティーのみ定義し、channel というプロパティーはデフォルト値の SYSTEM.DEF.SVRCONN のままにする場合、例外はスローされず、ccdtURL プロパティーによって識別されるクライアント・チャネル定義テーブルが JMS 接続の開始に使用されます。

ActivationSpec オブジェクトのプロパティーのほとんどは、WebSphere MQ classes for JMS オブジェクトのプロパティーまたは WebSphere MQ classes for JMS メソッドのパラメーターと同等です。 ただし、以下の 3 つのチューニング・プロパティーと 1 つの使用可能度プロパティーについては、WebSphere MQ classes for JMS でそれに相当するものが存在しません。

startTimeout
リソース・アダプターがメッセージを MDB に送達するように Work オブジェクトをスケジュールした後で、アプリケーション・サーバーの作業マネージャーがリソースが使用可能になるのを待機する時間 (ミリ秒)。 メッセージの送達が開始する前にこの時間が経過する場合、Work オブジェクトはタイムアウトし、メッセージはキュー上にロールバックされ、リソース・アダプターはメッセージの送達をもう一度試行できます。 診断トレースが使用可能になっている場合、警告は診断トレースに書き込まれますが、使用可能になっていない場合、メッセージの送達処理に影響を与えません。 アプリケーション・サーバーの負荷が非常に高くなっているときに限り、こうした条件が発生することを予期できます。 この条件が定期的に発生する場合、このプロパティーの値を大きくして、作業マネージャーでメッセージ送達のスケジュール時間を長くすることを考慮してください。

maxPoolDepth
接続コンシューマーによって使用されるサーバー・セッション・プール内のサーバー・セッションの最大数。 サーバー・セッションが作成されると、キュー・マネージャーとの会話が開始されます。 接続コンシューマーはサーバー・セッションを使用して、メッセージを MDB に送達します。 プールの深さを深くすると、高容量の状態で同時に送達されるメッセージの数も増えますが、使用するアプリケーション・サーバーのリソースも増えることになります。 多数の MDB がデプロイされる場合、アプリケー

ション・サーバー上で管理可能なレベルで負荷を維持するには、プールの深さを浅くすることを考慮してください。 各接続コンシューマーはその独自のサーバー・セッション・プールを使用するため、このプロパティーはすべての接続コンシューマーに使用可能なサーバー・セッションの総数を定義するわけではないことに注意してください。

poolTimeout
未使用のサーバー・セッションが、非アクティブ状態が原因でクローズされる前に、サーバー・セッション・プールでオープンしたままになっている時間 (ミリ秒)。 メッセージ・ワークロードの一時的な増加により、負荷を分散させるために追加のサーバー・セッションが作成されます。しかし、メッセージ・ワークロードが標準に戻った後も、追加のサーバー・セッションはプール内に残り、使用されません。

サーバー・セッションが使用されるたびに、タイム・スタンプでマークが付けられます。 スカベンジャー・スレッドは定期的に、各サーバー・セッションがこのプロパティーで指定された期間内に使用されているか確認します。 サーバー・セッションが使用されていない場合、それはクローズされ、サーバー・セッション・プールから除去されます。 指定された期間が経過した直後にサーバー・セッションがクローズされないことがあります。 このプロパティーは、除去される前に非アクティブ状態になる最小期間を表すからです。

useJNDI
このプロパティーの説明については、752 ページの表 97 を参照してください。

MDB をデプロイするには、まず、MDB が必要とするプロパティーを指定して ActivationSpec オブジェクトのプロパティーを定義します。 以下の例は、明示的に定義される代表的なプロパティー・セットです。

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

アプリケーション・サーバーはプロパティーを使用して、ActivationSpec オブジェクトを作成し、それは MDB に関連付けられます。 ActivationSpec オブジェクトのプロパティーにより、メッセージが MDB に送達される方法が決定されます。 MDB が分散トランザクションを必要とするが、リソース・アダプターが分散トランザクションをサポートしていない場合、MDB のデプロイメントは失敗します。 分散トランザクションがサポートされるようにリソース・アダプターをインストールする方法については、740 ページの『WebSphere MQ リソース・アダプターのインストール』 を参照してください。

複数の MDB が同じ宛先からメッセージを受信する場合、Point-to-Point ドメインで送信されたメッセージは、他の MDB がメッセージを受信する資格があるとしても 1 つの MDB にのみ受信されます。 特に、2 つの MDB が異なるメッセージ・セレクターを使用しており、着信メッセージが両方のメッセージ・セレクターに一致する場合、1 つの MDB のみメッセージを受信します。 メッセージを受信するように選ばれた MDB は未定義で、特定の MDB がメッセージを受信するようには指定できません。 パブリッシュ/サブスクライブ・ドメインで送信されたメッセージは資格があるすべての MDB によって受信されます。

リソース・アダプターでのインバウンド有害メッセージの処理

ある環境では、MDB に送達されたメッセージが WebSphere MQ キューにロールバックされることがあります。 例えば、メッセージが後でロールバックされる作業単位内で送達される場合は、ロールバックが起こります。 ロールバックされるメッセージは再度送達されますが、不良フォーマットのメッセージは、MDB が繰り返し失敗する原因になるため、送達できません。 こうしたメッセージは有害メッセージと呼ばれます。 WebSphere MQ classes for JMS が将来の調査のために有害メッセージを別のキューに自動的に転送するか、そのメッセージを破棄するように、WebSphere MQ を構成することができます。

有害メッセージの処理方法の詳細については、893 ページの『IBM WebSphere MQ classes for JMS でのポイズン・メッセージの処理』 を参照してください。

関連タスク
MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する
関連資料
UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

アウトバウンド通信のリソース・アダプターの構成

アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

アウトバウンド通信を使用する場合、アプリケーション・サーバーで実行するアプリケーションはキュー・マネージャーへの接続を開始し、それから同期的な方法でメッセージをそのキューに送信し、そのキューからメッセージを受信します。 例えば、以下のサーブレット・メソッド、doGet() はアウトバウンド通信を使用します。

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

サーブレットが HTTP GET 要求を受信すると、ConnectionFactory オブジェクトおよび Queue オブジェクトを JNDI ネーム・スペースから検索し、それらのオブジェクトを使用してメッセージを WebSphere MQ キューに送信します。 次いで、サーブレットは送信したメッセージを受信します。

アウトバウンド通信を構成するには、JCA リソースを以下のカテゴリーで定義してください。

- ConnectionFactory オブジェクトのプロパティー。アプリケーション・サーバーが JMS ConnectionFactory オブジェクトを作成するために使用します。
- 管理対象宛先オブジェクトのプロパティー。アプリケーション・サーバーが JMS Queue オブジェクトまたは JMS Topic オブジェクトを作成するために使用します。

これらのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。 アプリケーション・サーバーによって作成された ConnectionFactory、Queue、および Topic オブジェクトは、JNDI ネーム・スペースにバインドされ、アプリケーションはそのネーム・スペースからオブジェクトを検索できます。

通常、アプリケーションが接続する必要があるキュー・マネージャーごとに 1 つの ConnectionFactory オブジェクトを定義します。 アプリケーションが Point-to-Point ドメインでアクセスする必要があるキューごとに 1 つの Queue オブジェクトを定義します。 そして、アプリケーションがパブリッシュまたはサブスクライブするトピックごとに 1 つの Topic オブジェクトを定義します。 ConnectionFactory オブジェクトはドメインに依存しないようにすることができます。 あるいは、ドメイン固有にすることもできます。 Point-to-Point ドメインの場合は QueueConnectionFactory オブジェクト、パブリッシュ/サブスクライブ・ドメインの場合は TopicConnectionFactory オブジェクトです。

758 ページの表 98 は、ConnectionFactory オブジェクトのプロパティーをリストしています。

表 98. *ConnectionFactory* オブジェクトのプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
applicationName	ストリング	• 起動クラス名が使用可能である場合は、28 文字以内に調整されます。 使用可能でない場合は、ストリング `WebSphere MQ Client for Java` が使用されます。	アプリケーションをキュー・マネージャーに登録する際に使用した名前。 このアプリケーション名は、**DISPLAY CONN MQSC/PCF** コマンド (フィールド名は **APPLTAG**) または IBM WebSphere MQ Explorer の「**アプリケーション接続**」 画面 (フィールド名は **App name**) に表示されます。
brokerCCSubQueue[1]	ストリング	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが非永続サブスクリプション・メッセージを受信するキューの名前。
brokerControlQueue[1]	ストリング	• **SYSTEM.BROKER.CONTROL.QUEUE** • キュー名	ブローカー制御キューの名前。
brokerPubQueue[1]	ストリング	• **SYSTEM.BROKER.DEFAULT.STREAM** • キュー名	パブリッシュ済みメッセージが送信されたキュー (ストリーム・キュー) の名前。
brokerQueueManager[1]	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	ブローカーが稼働しているキュー・マネージャーの名前。
brokerSubQueue[1]	ストリング	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • キュー名	非永続メッセージ・コンシューマーがメッセージを受信するキューの名前。 詳細については、BROKERSUBQ プロパティーを参照してください。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
brokerVersion[1]	ストリング	• **指定なし** - ブローカーを V6 から V7 に移行した後、このプロパティーを設定し、RFH2 ヘッダーが使用されないようにします。 移行した後、このプロパティーは関係なくなります。 • **V1** - IBM WebSphere MQ パブリッシュ/サブスクライブ・ブローカーを使用する場合。 または、IBM WebSphere MQ Integrator、WebSphere Event Broker、WebSphere Business Integration Event Broker、または WebSphere Business Integration Message Broker のいずれかのブローカーを互換モードで使用する場合。 この値は、TRANSPORT が BIND または CLIENT に設定されている場合のデフォルト値です。 • **V2** - IBM WebSphere MQ Integrator、WebSphere Event Broker、WebSphere Business Integration Event Broker、または WebSphere Business Integration Message Broker のいずれかのブローカーをネイティブ・モードで使用する場合。 この値は、TRANSPORT が DIRECT または DIRECTHTTP に設定されている場合のデフォルト値です。	使用されているブローカーのバージョン。
ccdtURL	ストリング	• **null** • Uniform Resource Locator (URL)	クライアント・チャネル定義テーブル (CCDT) が格納されているファイルの名前と場所を識別し、このファイルのアクセス方法を指定する URL。
CCSID	ストリング	• **819** • Java 仮想マシン (JVM) でサポートされるコード化文字セット ID	接続用のコード化文字セット ID。
channel	ストリング	• **SYSTEM.DEF.SVRCONN** • MQI チャネルの名前	使用する MQI チャネルの名前。
cleanupInterval[1]	int	• **3 600 000** • 正整数	パブリッシュ/サブスクライブ・クリーンアップ・ユーティリティーがバックグラウンドで実行する間隔 (ミリ秒)。
cleanupLevel[1]	ストリング	• **SAFE** • NONE • strong • FORCE • NONDUR	ブローカー・ベースのサブスクリプション・ストアのクリーンアップ・レベル。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
clientID	ストリング	• **null** • クライアント ID	接続のクライアント ID。
cloneSupport	ストリング	• **DISABLED** - 一度に稼働できる永続トピック・サブスクライバーのインスタンスは 1 つだけです。 • ENABLED - 1 つの永続トピック・サブスクライバーの 2 つ以上のインスタンスを同時に稼働できますが、各インスタンスを別々の Java 仮想マシン (JVM) で稼働させる必要があります。	1 つの永続トピック・サブスクライバーの複数のインスタンスを同時に稼働できるかどうか。
connectionNameList	ストリング	• **localhost(1414)** • コンマで区切られている項目で構成されるストリングです。各項目は以下の形式になります。 `HOSTNAME(PORT)` ここで、*HOSTNAME* は DNS 名または IP アドレスです。	アウトバウンド通信に使用される TCP/IP 接続名のリスト。 **connectionNameList** は、**hostname** プロパティーと **port** プロパティーを置き換えます。 このプロパティーを使用して、複数インスタンス・キュー・マネージャーに再接続します。 **connectionNameList** は **localAddress** の形式に類似していますが、混同しないでください。**localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。
failIfQuiesce	Boolean	• **true** • false	キュー・マネージャーが静止状態の場合、特定のメソッドの呼び出しが失敗するかどうか。
headerCompression	ストリング	• **NONE** • SYSTEM - RLE メッセージ・ヘッダーの圧縮が実行される	接続のヘッダー・データを圧縮するために使用できる技法のリスト。
hostName	ストリング	• **localhost** • ホスト名 • IP アドレス	キュー・マネージャーが存在しているシステムのホスト名または IP アドレス。 **hostname** プロパティーと **port** プロパティーは、指定されている場合は **connectionNameList** プロパティーに置き換えられます。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明

localAddress	ストリング	• **null**
		• 以下の形式のストリング
		`[host_name]`
		`[(low_port[,high_port])]`
		ここで、*host_name* はホスト名または IP アドレス、*low_port* および *high_port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。
			キュー・マネージャーとの接続場合、このプロパティーは以下のいずれか、または両方を指定します。
			• 使用されるローカル・ネットワーク・インターフェース
			• 使用されるローカル・ポート、またはローカル・ポートの範囲
			localAddress は **connectionNameList** の形式に類似していますが、混同しないでください。 **localAddress** はローカル通信の特性を指定しますが、**connectionNameList** はリモート・キュー・マネージャーに到達する方法を指定します。

messageCompression	ストリング	• **NONE** • ブランク文字で区切られた以下の値の 1 つ以上のリスト。 RLE ZLIBFAST ZLIBHIGH	接続のメッセージ・データを圧縮するために使用できる技法のリスト。
messageSelection[1]	ストリング	• **CLIENT** • BROKER	メッセージ選択を IBM WebSphere MQ classes for JMS またはブローカーのどちらが行うかを決定します。 ブローカーによるメッセージ選択は、brokerVersion の値が 1 の場合、サポートされません。
パスワード	ストリング	• **null** • パスワード	キュー・マネージャーへの接続を作成する際に使用するデフォルト・パスワード。
pollingInterval[1]	int	• **5000** • 任意の正整数	この値は、各セッション内のメッセージ・リスナーのキューに適切なメッセージがない場合に、各メッセージ・リスナーがキューからメッセージの取得を再度試みるまでの最大の時間間隔 (ミリ秒) です。 セッション内のいずれのメッセージ・リスナーでも適切なメッセージがない状態が頻繁に発生する場合は、このプロパティーの値を大きくすることを考えてください。 このプロパティーは、TRANSPORT の値が BIND または CLIENT の場合にのみ使用されます。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
port	int	• **1414** • TCP ポート番号	キュー・マネージャーが listen を行うポート。 **hostname** プロパティーと **port** プロパティーは、指定されている場合は **connectionNameList** プロパティーに置き換えられます。
providerVersion	ストリング	• **指定なし** • 以下のいずれかの形式のストリング – V.R.M.F – V.R.M – V.R – V ここで、V、R、M、および F は、ゼロ以上の整数値です。	アプリケーションが接続することになっているキュー・マネージャーのバージョン、リリース、修正レベル、およびフィックスパック。
pubAckInterval[1]	int	• **25** • 正整数	IBM WebSphere MQ classes for JMS がブローカーからの確認通知を要求するまでに、パブリッシャーによって公開されるメッセージの数。
queueManager	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	接続するキュー・マネージャーの名前。
receiveExit[3]	ストリング	• **null** • コンマで区切られた 1 つ以上の項目から成るストリング。各項目は、IBM WebSphere MQ classes for Java インターフェース、MQReceiveExit を実装するクラスの完全修飾名です。	チャネル受信出口プログラム、または連続して実行される一連の受信出口プログラムを識別します。
receiveExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル受信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
rescanInterval[1]	int	• **5000** • 任意の正整数	Point-to-Point ドメインのメッセージ・コンシューマーがメッセージ・セレクターを使用して受信するメッセージを選択する場合、WebSphere MQ classes for JMS は IBM WebSphere MQ MQ キューを検索して、このキューの *MsgDeliverySequence* 属性によって決定される順序で適切なメッセージを探します。 WebSphere MQ classes for JMS が適切なメッセージを検出してそれをコンシューマーに送信すると、 WebSphere MQ classes for JMS は、キュー内の現在位置から次の適切なメッセージの検索を再開します。WebSphere MQ classes for JMS は、キューの終わりに達するまで、またはこのプロパティーの値によって決定される時間間隔 (ミリ秒) が経過するまで、この方法でキューの検索を続行します。 いずれの場合も、 WebSphere MQ classes for JMS はキューの先頭に戻って検索を続行し、これにより新しい時間間隔が開始します。
securityExit[3]	ストリング	• **null** • WebSphere MQ classes for Java インターフェース、 MQSecurityExit を実装するクラスの完全修飾名。	チャネル・セキュリティー出口プログラムを識別します。
securityExitInit	ストリング	• **null** • ユーザー・データのストリング	チャネル・セキュリティー出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。
sendCheckCount	int	• **0** • 任意の正整数	単一の未処理 JMS セッション内で、非同期書き込みエラーの検査が行われてから次の検査が行われるまでに許可される送信呼び出しの数。
sendExit[3]	ストリング	• **null** • 1 つ以上の項目をコンマで区切ったストリング。各項目は、 WebSphere MQ classes for Java インターフェース MQSendExit を実装するクラスの完全修飾名です。	チャネル送信出口プログラム、または連続して実行される一連の送信出口プログラムを識別します。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
sendExitInit	ストリング	• **null** • コンマで区切られた 1 つ以上のユーザー・データ項目から成るストリング	チャネル送信出口プログラムが呼び出されたときに、その出口プログラムに渡されるユーザー・データ。
shareConvAllowed	Boolean	• NO-クライアント接続はそのソケットを共用できません。 • **YES** -クライアント接続はそのソケットを共用できます。	チャネル定義が一致する場合に、クライアント接続がそのソケットを同じプロセスから同じキュー・マネージャーへの他のトップレベル JMS 接続と共用できるかどうか
sparseSubscriptions[1]	Boolean	• **false** - サブスクリプションが頻繁なマッチング・メッセージを受信する。 • true - サブスクリプションが頻繁でないマッチング・メッセージを受信する。この値は、サブスクリプション・キューがブラウズ用にオープンできることを必要とします。	TopicSubscriber オブジェクトのメッセージ検索ポリシーを制御する。
sslCertStores	ストリング	• **null** • ブランクで区切られた 1 つ以上の LDAP URL のストリング。 各 LDAP URL の形式は次のとおりです。 `ldap://host_name[:port]` ここで、*host_name* はホスト名または IP アドレス、*port* は TCP ポート番号で、大括弧はオプション・コンポーネントを示します。	SSL 接続で使用する証明書取り消しリスト (CRL) を保持する Lightweight Directory Access Protocol (LDAP) サーバー。
sslCipherSuite	ストリング	• **null** • CipherSuite の名前	SSL 接続で使用する CipherSuite。
sslFipsRequired[2]	Boolean	• **false** • true	IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を SSL 接続で使用する必要があるかどうか。
sslPeerName	ストリング	• **null** • 識別名のテンプレート	SSL 接続で、キュー・マネージャーで提供されるデジタル証明書の識別名を確認するために使用するテンプレート。
sslResetCount	int	• **0** • 0 から 999 999 999 の範囲の整数	SSL によって使用された秘密鍵の再ネゴシエーションの前に、SSL 接続で送信および受信したバイトの総数。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
sslSocketFactory	ストリング	javax.net.ssl.SSLSocketFactory インターフェースのインプリメンテーションを提供するクラスの、完全修飾クラス名を表すストリング。 コンストラクター・メソッドに渡される引数を、括弧内に含めることもできます。	管理された宛先オブジェクトの有効範囲内で設定されたすべての接続は、SSLSocketFactory インターフェースのインプリメンテーションから得られたソケットを使用します。
statusRefreshInterval[1]	int	• **60000** • 任意の正整数	サブスクライバーがキュー・マネージャーとの接続を失ったときに検出する長期実行トランザクションのリフレッシュの間隔 (ミリ秒)。このプロパティーは、SUBSTORE の値が QUEUE の場合にのみ使用されます。
subscriptionStore[1]	ストリング	• **ブローカー** • MIGRATE • QUEUE	WebSphere MQ classes for JMS がアクティブ・サブスクリプションに関する永続データを保管する場所を決定します。
targetClientMatching	Boolean	• **true** • false	着信メッセージの JMSReplyTo ヘッダー・フィールドで識別されるキューに送信された応答メッセージに MQRFH2 ヘッダーがあるかどうか (着信メッセージに MQRFH2 ヘッダーがある場合のみ)。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
temporaryModel	ストリング	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • 任意のストリング	JMS 一時キューの作成に使用するモデル・キューの名前。 次の条件が両方とも該当する場合は、SYSTEM.DEFAULT.MODEL.QUEUE を使用します。 • アプリケーションが非永続メッセージを受け入れる一時キューを使用する。 • ConnectionFactory が指すキュー・マネージャーの一時キューは、一度に 1 つのアプリケーションによってのみ作成される。SYSTEM.DEFAULT.MODEL.QUEUE を開くことができるのは、一度に 1 つのアプリケーションによってのみであることに注意してください。 SYSTEM.JMS.TEMPQ.MODEL。 以下の状態の場合: • アプリケーションが永続メッセージを受け入れる一時キューを使用する場合。 • 複数のアプリケーションが ConnectionFactory が指すキュー・マネージャーに接続でき、それらのアプリケーションで同時に一時キューを作成する必要がある場合。 以下の状態では、DEFPSIST 属性を YES に設定し、DEFSOPT 属性を SHARED に設定して新規モデル・キューを定義します。 • アプリケーションが非永続メッセージを受け入れる一時キューを使用し、複数のアプリケーションが ConnectionFactory が指すキュー・マネージャーに接続し、それらのアプリケーションで同時に一時キューを作成する必要がある場合。 新規モデル・キューが作成されたら、temporaryModel プロパティーを新規モデル・キューの名前に設定します。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
tempQPrefix	ストリング	• **"" (空ストリング)** • IBM WebSphere MQ 動的キューの名前を形成するために使用できる接頭部。接頭部を形成するための規則は、 IBM WebSphere MQ オブジェクト記述子、構造体 MQOD 内の *DynamicQName* フィールドの内容を形成するための規則と同じですが、最後の非ブランク文字はアスタリスク (*) でなければなりません。プロパティーの値が空ストリングの場合、 WebSphere MQ classes for JMS は値 AMQ.* を使用します。 動的キューの作成時。	IBM WebSphere MQ 動的キューの名前を形成するために使用された接頭部。
tempTopicPrefix	ストリング	IBM WebSphere MQ トピック・ストリングに有効な文字だけで構成される非ヌル・ストリング	一時トピックの作成時に、JMS は "TEMP/*TEMPTOPICPREFIX*/*unique_id*" という形式のトピック・ストリングを生成します。またはこのプロパティーがデフォルト値のままになっている場合、"TEMP/*unique_id*" のみ生成します。 空でない TEMPTOPICPREFIX を指定すると、この接続の下で作成された一時トピックに対して、 サブスクライバー用の管理されたキューを作成するための特定のモデル・キューを定義できます。
transportType	ストリング	• **CLIENT** • BINDINGS • BINDINGS_THEN_CLIENT	キュー・マネージャーへの接続で、クライアント・モードまたはバインディング・モードのどちらを使用するか。 BINDINGS_THEN_CLIENT という値が指定されると、リソース・アダプターはまずバインディング・モードで接続を試みます。 この接続が失敗すると、リソース・アダプターはクライアント・モード接続の確立を試行します。
ユーザー名	ストリング	• **null** • ユーザー名	キュー・マネージャーへの接続を作成する際に使用するデフォルト・ユーザー名。
wildcardFormat	int	• CHAR- ブローカー・バージョン 1 で使用された文字ワイルドカードのみ認識します。 • TOPIC - ブローカー・バージョン 2 で使用されたトピック・レベル・ワイルドカードのみ認識します。	使用されるワイルドカード構文のバージョン。

表 98. *ConnectionFactory* オブジェクトのプロパティー (続き)			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
注 : 1. このプロパティーは IBM WebSphere MQ classes for JMS のバージョン 7.0 で使用できますが、providerVersion プロパティーのバージョン番号が 7 より小さく設定されていない限り、バージョン 7.0 キュー・マネージャーに接続されたアプリケーションに影響を及ぼしません。 2. sslFipsRequired プロパティーの使用の重要な詳細については、775 ページの『IBM WebSphere MQ リソース・アダプターの制限』を参照してください。 3. 出口を見つけることができるようにリソース・アダプターを構成する方法については、917 ページの『チャネル出口を使用するように IBM WebSphere MQ classes for JMS を構成する』を参照してください。			

以下の例は、ConnectionFactory オブジェクトの代表的なプロパティー・セットを示します。

```
channel:         SYSTEM.DEF.SVRCONN
hostName:        192.168.0.42
port:            1414
queueManager:    ExampleQM
transportType:   CLIENT
```

768 ページの表 99 は、Queue オブジェクトおよび Topic オブジェクトに共通するプロパティーを示します。

表 99. *Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
CCSID	ストリング	• **1208** • Java 仮想マシン (JVM) でサポートされるコード化文字セット ID	宛先用のコード化文字セット ID。
encoding	ストリング	• **NATIVE** • 以下の 3 文字のストリング – 先頭文字は 2 進整数の表記を示します。 - *N* は normal (標準) のエンコードを示します。 - *R* は reverse (逆) のエンコードを示します。 – 2 番目の文字はパック 10 進整数の表記を示します。 - *N* は normal (標準) のエンコードを示します。 - *R* は reverse (逆) のエンコードを示します。 – 3 番目の文字は浮動小数点数の表記を示します。 - *N* は標準の IEEE エンコードを示します。 - *R* は逆の IEEE エンコードを示します。 - *3* は zSeries エンコードを示します。 NATIVE はストリング NNN に相当します。	宛先の 2 進整数、パック 10 進整数、および浮動小数点数の表記

表 99. *Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー (続き)

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
expiry	ストリング	• **APP** - メッセージの有効期限時刻はメッセージ・プロデューサーによって決定される • UNLIM - メッセージに有効期限を設けない • 0 - メッセージに有効期限を設けない • メッセージの有効期限時刻を表す正整数 (ミリ秒)	宛先に送信されるメッセージの有効期限時刻。
failIfQuiesce	ストリング	• **true** • false	キュー・マネージャーが静止状態の場合、宛先へのアクセスが失敗するかどうか。
persistence	ストリング	• **APP** - メッセージの持続性はメッセージ・プロデューサーによって決定される • QDEF-メッセージの持続性は、 WebSphere MQ キューの *DefPersistence* 属性によって決定される。 • PERS - メッセージに持続性を与える • NON - メッセージに持続性を与えない • HIGH-メッセージの持続性は、 909 ページの『JMS 持続メッセージ』の説明に従って、WebSphere MQ キューの *NonPersistentMessageClass* 属性によって決定されます。	宛先に送信されるメッセージの持続性。
priority	ストリング	• **APP** - メッセージの優先順位はメッセージ・プロデューサーによって決定される • QDEF-メッセージの優先順位は、 IBM WebSphere MQ キューの *DefPriority* 属性によって決定される。 • 0 (最低優先順位) から 9 (最高優先順位) の範囲の整数	宛先に送信されるメッセージの優先順位。
putAsyncAllowed	ストリング	• QUEUE - キュー定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • DESTINATION - キュー定義またはトピック定義を参照することによって非同期書き込みが許可されるかどうかを判別します。 • DISABLED - 非同期書き込みは許可されない • ENABLED - 非同期書き込みは許可される	メッセージ・プロデューサーが非同期書き込みを使用してこの宛先にメッセージを送信することを許可されるかどうか。

表 *99. Queue* オブジェクトおよび *Topic* オブジェクトに共通するプロパティー *(続き)*			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
readAheadAllowed	int	• **DESTINATION** - キュー定義またはトピック定義を参照することによって先読みが許可されるかどうかを判別します。 • DISABLED - 先読みは許可されない • ENABLED - 先読みは許可される • QUEUE - キュー定義を参照することによって先読みが許可されるかどうかを判別します。 • TOPIC - トピック定義を参照することによって先読みが許可されるかどうかを判別します。	メッセージ・コンシューマーおよびキュー・ブラウザーが先読みを使用してこの宛先から内部バッファーへの非永続メッセージを受信する前に、それらを取得することを許可されるかどうか。
receiveCCSID	int	• **0** - JVM `Charset.defaultCharset` を使用します。 • 1208 - UTF-8 • サポートされるコード化文字セット ID	キュー・マネージャー・メッセージ変換のターゲット CCSID を 設定する宛先プロパティー。 **receiveConversion** が QMGR に設定されていない場合、値は無視されます。
receiveConversion	ストリング	• **CLIENT_MSG** • QMGR	キュー・マネージャーによりデータ変換を実行するかどうかを決定する宛先プロパティー。
targetClient	ストリング	• **JMS** - JMS アプリケーションをメッセージのターゲットにする • MQ -メッセージのターゲットは、JMS 以外の IBM WebSphere MQ アプリケーションです。	宛先に送信されるメッセージのターゲットが JMS アプリケーションかどうか。 ターゲットが JMS アプリケーションであるメッセージには MQRFH2 ヘッダーが含まれています。

770 ページの表 100 は、Queue オブジェクトに固有のプロパティーを示します。

表 *100. Queue* オブジェクトに固有のプロパティー			
プロパティー名	**タイプ**	**有効値 (太字はデフォルト値)**	**説明**
baseQueueManagerName	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	基礎となる IBM WebSphere MQ キューを所有するキュー・マネージャーの名前。
baseQueueName	ストリング	• **"" (空ストリング)** • キュー名	基礎となる IBM WebSphere MQ キューの名前。

771 ページの表 101 は、Topic オブジェクトに固有のプロパティーを示します。

表 101. *Topic* オブジェクトに固有のプロパティー

プロパティー名	タイプ	有効値 (太字はデフォルト値)	説明
baseTopicName	ストリング	• **"" (空ストリング)** • トピック名	基礎となるトピックの名前。
brokerCCDurSubQueue[1]	ストリング	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • キュー名	接続コンシューマーが永続サブスクリプション・メッセージを受信するキューの名前。
brokerDurSubQueue[1]	ストリング	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • キュー名	永続トピック・サブスクライバーがメッセージを受信するキューの名前。 詳細については、WebSphere MQ エクスプローラーの資料にある BROKEDURRSUBQ プロパティーの説明を参照してください。
brokerPubQueue[1]	ストリング	• **設定されない** • キュー名	パブリッシュ済みメッセージが送信されたキュー (ストリーム・キュー) の名前。 このプロパティーの値は、ConnectionFactory オブジェクトの brokerPubQueue プロパティーの値をオーバーライドします。 ただし、このプロパティーの値を設定しない場合、ConnectionFactory オブジェクトの brokerPubQueue プロパティーの値が代わりに使用されます。
brokerPubQueueManager[1]	ストリング	• **"" (空ストリング)** • キュー・マネージャー名	トピックに関して公開されたメッセージが送信されるキューを所有するキュー・マネージャーの名前。
brokerVersion[1]	ストリング	• **設定されない** • 1 • 2	使用されているブローカーのバージョン。 このプロパティーの値は、ConnectionFactory オブジェクトの brokerVersion プロパティーの値をオーバーライドします。 ただし、このプロパティーの値を設定しない場合、ConnectionFactory オブジェクトの brokerVersion プロパティーの値が代わりに使用されます。

注 :

1. このプロパティーは IBM WebSphere MQ classes for JMS のバージョン 7.0 で使用できますが、ConnectionFactory オブジェクトの providerVersion プロパティーのバージョン番号が 7 より小さく設定されていない限り、バージョン 7.0 キュー・マネージャーに接続されたアプリケーションに影響を及ぼしません。

以下の例は、Queue オブジェクトのプロパティー・セットを示します。

```
expiry:                 UNLIM
persistence:            QDEF
baseQueueManagerName:   ExampleQM
baseQueueName:          SYSTEM.JMS.TEMPQ.MODEL
```

以下の例は、Topic オブジェクトのプロパティー・セットを示します。

```
expiry:                 UNLIM
persistence:            NON
baseTopicName:          myTestTopic
```

関連タスク

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

関連資料

UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

V 7.5.0.9 アクティベーション・スペックの *targetClientMatching* プロパティーの構成

アクティベーション・スペックの **targetClientMatching** プロパティーを構成することで、要求メッセージに MQRFH2 ヘッダーが含まれないときに応答メッセージに MQRFH2 ヘッダーを含めることができます。 これは、応答メッセージでアプリケーションが定義するメッセージ・プロパティーがメッセージの送信時に組み込まれることを意味します。

このタスクについて

例えば、メッセージ駆動型 Bean (MDB) アプリケーションが、MQRFH2 ヘッダーを含まないメッセージを IBM WebSphere MQ JCA リソース・アダプターのアクティベーション・スペックを介してコンシュームし、その後、要求メッセージの JMSReplyTo フィールドから作成された JMS 宛先に応答メッセージを送信する場合は、要求メッセージに MQRFH2 ヘッダーが含まれていなくてもそれを応答メッセージに組み込む必要があります。そうしないと、アプリケーションが応答メッセージで定義したメッセージ・プロパティーが失われてしまいます。

targetClientMatching プロパティーでは、着信メッセージに MQRFH2 ヘッダーがある場合にのみ応答メッセージ (着信メッセージの JMSReplyTo ヘッダー・フィールドで識別されるキューに送信される) に MQRFH2 ヘッダーを組み込むようにするかを定義します。 アクティベーション・スペックのこのプロパティーは、WebSphere Application Server (従来型) でも WebSphere Application Server Liberty でも構成できます。

targetClientMatching プロパティーの値を `false` に設定すると、着信要求メッセージに MQRFH2 が含まれない場合に、その応答メッセージ (着信要求メッセージの JMSReplyTo ヘッダーから作成された JMS 宛先に送信される) に MQRFH2 ヘッダーが組み込まれます。 これは、JMS 宛先の **targetClient** プロパティーが値 `0` (メッセージに MQRFH2 ヘッダーが含まれることを意味する) に設定されるからです。 アウトバウンド・メッセージに MQRFH2 ヘッダーが存在すれば、IBM WebSphere MQ キューへの送信時にメッセージにユーザー定義のメッセージ・プロパティーを格納できます。

targetClientMatching プロパティーを `true` に設定した場合、要求メッセージに MQRFH2 ヘッダーが組み込まれていなければ、応答メッセージにも MQRFH2 ヘッダーは組み込まれません。

手順

- WebSphere Application Server (従来型) で、管理コンソールを使用して、**targetClientMatching** プロパティーを IBM WebSphere MQ アクティベーション・スペックのカスタム・プロパティーとして定義します。
  a) ナビゲーション・ペインで、「**リソース**」->「**JMS**」->「**アクティベーション・スペック**」をクリックします。
  b) 表示または変更するアクティベーション・スペックの名前を選択します。
  c)「**カスタム・プロパティー**」->「**新規**」をクリックした後、新規カスタム・プロパティーの詳細を入力します。
  プロパティーの名前を `targetClientMatching` に設定し、タイプを `java.lang.Boolean` に設定し、値を `false` に設定します。

- WebSphere Application Server Liberty で、 `server.xml` 内のアクティベーション・スペックの定義に **targetClientMatching** プロパティーを指定します。

  以下に例を示します。

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

関連概念

882 ページの『JMS アプリケーションでの宛先の作成』
JMS アプリケーションは、Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトとして宛先を取得する代わりに、セッションを使用して実行時に動的に宛先を作成することができます。アプリケーションは URI (Uniform Resource Identifier) を使用して WebSphere MQ キューまたはトピックを識別し、オプションで、Queue または Topic オブジェクトの 1 つ以上のプロパティーを指定することができます。

757 ページの『アウトバウンド通信のリソース・アダプターの構成』
アウトバウンド通信を構成するには、ConnectionFactory オブジェクトおよび管理対象宛先オブジェクトのプロパティーを定義してください。

ASF モードと非 *ASF* モード
アプリケーション・サーバー機構 (ASF) モードは、WebSphere Application Server のメッセージ・リスナー・サービスがメッセージを処理する場合に使用されるデフォルトの方法です。

メッセージ・リスナー・サービスには、以下のアプリケーション・サーバー機構 (ASF) および非アプリケーション・サーバー機構 (非 ASF) という 2 つの操作モードがあります。

- ASF モードは、アプリケーションに並行性およびトランザクションのサポートを提供します。 パブリッシュ/サブスクライブ・メッセージ駆動型 Bean の場合、ASF モードを使用することで、スループットと並行性が向上します。これは、非 ASF モードではリスナーが単一スレッドであるためです。
- 非 ASF モードは、JMS ASF (JMS 仕様のオプション拡張機能) をサポートしないサード・パーティー・メッセージング・プロバイダーで主に使用されます。 また、非 ASF モードはトランザクション・モードですが、パスの長さが ASF モードの場合より短いため、通常はパフォーマンスが向上します。

アプリケーション・サーバー上のすべてのメッセージ駆動型 Bean リスナーに対して非 ASF 操作モードを有効にするには、このプロパティーをゼロ以外の値に設定します。

注 :

z/OS システムでは非 ASF モードを選択できません。したがって、その場合はこのプロパティーにゼロ以外の値を設定しないでください。

ASF モードでのメッセージ処理

ASF モードでは、メッセージ駆動型 Bean (MDB) に適したメッセージが検出された場合にのみ、処理用にサーバー・セッションおよびスレッドが割り振られます。 MDB が同時に処理できるスレッドの数は、リスナー・ポートまたはアクティベーション・スペックの **Maximum Sessions** プロパティーの値によって決まります。

非 ASF モードでのメッセージ処理

非 ASF モードでは、リスナー・ポートまたはアクティベーション・スペックが開始された時点からスレッドがアクティブになります。 アクティブ・スレッドの数は、 **Maximum Sessions** プロパティーに指定された値によって決まります。 「**Maximum Sessions**」プロパティーで指定されたスレッドの数は、処理可能なメッセージの数に関係なくアクティブです。 アクティブ・スレッドはそれぞれ個々の物理ネットワーク接続です。

IBM WebSphere MQ バージョン 7.0 以降では、単一の物理ネットワーク接続を共用する最大 10 個のスレッドを使用できます。

関連概念

IBM WebSphere MQ classes for JMS アプリケーション・サーバー機構
このトピックでは、WebSphere MQ classes for JMS が Session クラス内の ConnectionConsumer クラスおよび拡張機能を実装する方法を説明します。 さらに、サーバー・セッション・プールの機能も要約しています。

関連タスク

非 ASF モード用アクティベーション・スペックの構成
アクティベーション・スペックは、WebSphere Application Server で実行するメッセージ駆動型 Bean (MDB) と IBM WebSphere MQ の宛先間の関係を管理および構成するための標準化された方法です。 このタスクでは、非 ASF モードを使用してメッセージを処理するように WebSphere Application Server を構成する方法を説明します。

関連情報

ASF モードおよび非 ASF モードでのメッセージ処理

非 *ASF* モード用アクティベーション・スペックの構成
アクティベーション・スペックは、WebSphere Application Server で実行するメッセージ駆動型 Bean (MDB) と IBM WebSphere MQ の宛先間の関係を管理および構成するための標準化された方法です。 このタスクでは、非 ASF モードを使用してメッセージを処理するように WebSphere Application Server を構成する方法を説明します。

始める前に

アクティベーション・スペックのプロパティーを定義する方法は、アプリケーション・サーバーで提供される管理インターフェースに応じて異なります。 このタスクは、WebSphere Application Server バージョン 7 以降をアプリケーション・サーバーとして使用し、IBM WebSphere MQ をメッセージング・プロバイダーとして使用することを前提としています。

注 :

z/OS システムでは、非 ASF モードを選択することはできません。

このタスクについて

アクティベーション・スペックのプロパティーにより、メッセージ駆動型 Bean (MDB) が JMS メッセージを IBM WebSphere MQ キューから受信する方法が決まります。 非 ASF モードを構成するには、1 つ以上のアクティベーション・スペックのプロパティーを定義します。

非 ASF モードで使用できる IBM WebSphere MQ 構成がいくつかあります。 以下の構成では、スレッドはそれぞれ別の物理ネットワーク接続を使用します。

- プロバイダー・バージョン・プロパティーが 6 に設定された接続ファクトリーを使用する IBM WebSphere MQ バージョン 7.x キュー・マネージャー。
- IBM WebSphere MQ バージョン 7.x キュー・マネージャーは、プロバイダー・バージョン・プロパティーが 7 に設定されているか、指定されていない接続ファクトリーを使用して、 **SHARECNV** (共有会話) パラメーターが 0 に設定されている IBM WebSphere MQ チャネルを介して接続します。

非 ASF を構成するには、ActivationSpec プロパティー **NON.ASF.RECEIVE.TIMEOUT** を正整数 (非 ASF 送達が使用されることを示す) に設定します。 この値は、読み取り要求で、まだ到着していない可能性のあるメッセージを待機する時間 (ミリ秒単位) です (待機を伴う読み取りの呼び出し)。 デフォルト値 0 は ASF 送達が使用されることを示します。 詳細については、**メッセージ・リスナー・サービスのカスタム・プロパティー**を参照してください。

このパラメーターは、アプリケーションが WebSphere Application Server バージョン 7 以降で実行している場合にのみ有効です。

手順

1. WebSphere Application Server 管理コンソールを開始します。
2. 次のように「listener service settings」ページを表示します。

a) ナビゲーション・ペインで、「**サーバー**」>「**サーバー・タイプ**」>「**WebSphere アプリケーション・サーバー**」を選択します。

b) コンテンツ・ペインで、アプリケーション・サーバーの名前をクリックします。

c)「**通信**」で、「**メッセージング**」>「**メッセージ・リスナー・サービス**」をクリックします。

3. カスタム・プロパティー **NON.ASF.RECEIVE.TIMEOUT** をメッセージ・リスナー・サービスのカスタム・プロパティーとして設定します。

a)「**カスタム・プロパティー**」をクリックします。

b)「**新規**」をクリックします。

c)「**名前**」フィールドに、プロパティー **NON.ASF.RECEIVE.TIMEOUT** の名前を入力します。

d)「**値**」フィールドに必要な値を入力します。

e)「**OK**」をクリックします。

4. 変更内容をマスター構成に保存します。
5. 変更した構成をアクティブにするには、アプリケーション・サーバーを停止してから再始動します。

タスクの結果

これで、非 ASF モードを使用するための WebSphere Application Server に対するメッセージ・リスナー・サービスのプロパティーが構成されました。

注 : 非 ASF モードを使用する場合は、不要なトランザクション・タイムアウトを避けるために、合計トランザクション存続時間タイムアウトに達する前に処理を完了するための十分な時間を取るようにしてください。 詳しくは、 WebSphere Application Server 製品資料の **NON.ASF.RECEIVE.TIMEOUT** を参照してください。

関連概念

773 ページの『ASF モードと非 ASF モード』
アプリケーション・サーバー機構 (ASF) モードは、WebSphere Application Server のメッセージ・リスナー・サービスがメッセージを処理する場合に使用されるデフォルトの方法です。

インバウンド通信のリソース・アダプターの構成
インバウンド通信を構成するには、1 つ以上の ActivationSpec オブジェクトのプロパティーを定義します。

関連情報

メッセージ駆動型 Bean

メッセージ・リスナー・サービス

ASF モードおよび非 ASF モードでのメッセージ処理

非 ASF モードでのメッセージの処理方法

IBM WebSphere MQ リソース・アダプターの制限

IBM WebSphere MQ リソース・アダプターを使用する際、IBM WebSphere MQ の機能のうちのあるものは、利用できないかまたは制限されます。

IBM WebSphere MQ リソース・アダプターには以下の制限があります。

- IBM WebSphere MQ リソース・アダプターは、 z/OS を除くすべての IBM WebSphere MQ プラットフォームでサポートされます。
- IBM WebSphere MQ リソース・アダプターは、ブローカーへのリアルタイム接続をサポートしません。クライアント・モードまたはバインディング・モードでの IBM WebSphere MQ キュー・マネージャーへの接続のみサポートします。
- IBM WebSphere MQ リソース・アダプターは、Java 以外の言語で作成されたチャネル出口プログラムをサポートしません。
- アプリケーション・サーバーが実行している間、sslFipsRequired プロパティーの値は、すべての JCA リソースについて true であるか、すべての JCA リソースについて false である必要があります。 JCA リソースが同時に使用されない場合でも、これが要件です。 sslFipsRequired プロパティーの値が JCA リソースごとに異なる場合、SSL 接続が使用されていなくても、IBM WebSphere MQ は理由コード MQRC_UNSUPPORTED_CIPHER_SUITE を出します。

- アプリケーション・サーバーに複数の鍵ストアを指定することはできません。 複数のキュー・マネージャーへの接続がある場合、すべての接続は同じ鍵ストアを使用する必要があります。 この制限は、WebSphere Application Server には適用されません。
- 適合するクライアント接続チャネル定義を複数指定してクライアント・チャネル定義テーブル (CCDT) を使用すると、リソース・アダプターが別のチャネル定義を選択し、それゆえに CCDT から異なるキュー・マネージャーを選択するという失敗をした場合に、トランザクション・リカバリーの問題を引き起こす可能性があります。 リソース・アダプターは、そのような構成が使用されないように防ぐアクションを実行しません。 トランザクション・リカバリーの問題を引き起こす可能性がある構成を避けるのは、ユーザーの責任となります。
- IBM WebSphere MQ Version 7.0.1 で導入された接続再試行機能は、JEE コンテナー (EJB/Servlet) で実行されている場合、アウトバウンド接続ではサポートされません。 接続再試行は、アダプターが JEE コンテナー・コンテキストで使用される場合、トランザクション構成や非トランザクション化での使用に関係なく、アウトバウンド JMS ではサポートされません。

関連タスク

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

関連資料

UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

WebSphere MQ classes for JMS アプリケーションのインストール後のセットアップ

このトピックでは、WebSphere MQ classes for JMS アプリケーションがキュー・マネージャーのリソースにアクセスするために必要な権限について説明します。 また、接続モードの概要を示し、アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成する方法を説明します。

WebSphere MQ の README ファイルを必ず確認してください。 このトピックの情報に優先する情報が含まれていることがあります。

特権を持たないユーザーには権限が必要な、*JMS* が使用するオブジェクト

JMS が使用するキューにアクセスする場合、特権を持たないユーザーは、権限の認可を受ける必要があります。 どの JMS アプリケーションでも、処理対象のキュー・マネージャーに対する権限が必要になります。

IBM WebSphere MQ でのアクセス制御について詳しくは、 Windows、 UNIX and Linux システムでのセキュリティーのセットアップ を参照してください。

WebSphere MQ classes for JMS アプリケーションには、キュー・マネージャーに対する `connect` および `inq` 権限が必要です。 **`setmqaut`** 制御コマンドを使用して、適切な許可を設定することができます。例えば、次のようにします。

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Point-to-Point ドメインについては、以下の権限が必要です。

- MessageProducer オブジェクトが使用するキューには、書き込み権限が必要です。
- MessageConsumer オブジェクトと QueueBrowser オブジェクトが使用するキューには、読み取り、照会、およびブラウズ権限が必要です。
- QueueSession.createTemporaryQueue() メソッドには、QueueConnectionFactory オブジェクトの TEMPMODEL プロパティーで指定されたモデル・キューへのアクセス許可が必要です。 デフォルトで、このモデル・キューは SYSTEM.TEMP.MODEL.QUEUE です。

これらのキューのいずれかが別名キューである場合、そのターゲット・キューには照会権限が必要です。ターゲット・キューがクラスター・キューの場合、ブラウズ権限も必要です。

パブリッシュ/サブスクライブ・ドメインでは、WebSphere MQ classes for JMS が IBM WebSphere MQ メッセージング・プロバイダー移行モードで IBM WebSphere MQ キュー・マネージャーに接続している場合、以下のキューが使用されます。

- SYSTEM.JMS.ADMIN.QUEUE

- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

IBM WebSphere MQ メッセージング・プロバイダー移行モードの詳細については、**PROVIDERVERSION** を使用する事例を参照してください。

また、WebSphere MQ classes for JMS がこのモードでキュー・マネージャーに接続している場合、メッセージをパブリッシュするすべてのアプリケーションには、TopicConnectionFactory またはトピック・オブジェクトによって指定されたストリーム・キューへのアクセス権限が必要です。 デフォルトで、キューは SYSTEM.BROKER.DEFAULT.STREAM です。

ConnectionConsumer、IBM WebSphere MQ リソース・アダプター、または WebSphere Application Server IBM WebSphere MQ メッセージング・プロバイダーを使用する場合は、さらに追加の権限が必要になることがあります。

ConnectionConsumer によって読み取られるキューには、`get`、`inq`、および `browse` 権限が必要です。 システム送達不能キュー、および ConnectionConsumer によって使用されるバックアウト・リキュー・キューまたはレポート・キューには、`put` および `passall` 権限が必要です。

パブリッシュ/サブスクライブ・メッセージングを実行するために WebSphere MQ メッセージング・プロバイダーの通常モードを使用するアプリケーションは、キュー・マネージャーによって提供される統合されたパブリッシュ/サブスクライブ機能を使用します。 使用されるトピックおよびキューの機密保護については、パブリッシュ/サブスクライブのセキュリティーを参照してください。

WebSphere MQ classes for JMS の接続モード

WebSphere MQ classes for JMS アプリケーションは、クライアント・モードかバインディング・モードのいずれかでキュー・マネージャーに接続できます。 クライアント・モードでは、WebSphere MQ classes for JMS は TCP/IP を介してキュー・マネージャーに接続します。 バインディング・モードでは、WebSphere MQ classes for JMS は、Java Native Interface (JNI) を使用してキュー・マネージャーに直接接続します。

z/OS 上の WebSphere Application Server で実行されるアプリケーションはバインディング・モードかクライアント・モードのいずれかでキュー・マネージャーに接続できますが、z/OS 上のそれ以外の環境で実行されるアプリケーションはバインディング・モードでのみキュー・マネージャーに接続できます。 それ以外のプラットフォームで実行されるアプリケーションはすべて、バインディング・モードかクライアント・モードのいずれかでキュー・マネージャーに接続できます。

現行のキュー・マネージャーで、WebSphere MQ classes for JMS のサポートされている現行バージョンまたは旧バージョンを使用できます。 また、WebSphere MQ classes for JMS の現行バージョンで、キュー・マネージャーのサポートされている現行バージョンまたは旧バージョンを使用できます。 異なるバージョンを混用する場合、機能は旧バージョンのレベルに制限されます。

以下のセクションでは、それぞれの接続モードについて詳しく説明します。

クライアント・モード

クライアント・モードでキュー・マネージャーに接続するには、キュー・マネージャーが実行されているのと同じシステムかまたは別のシステムで WebSphere MQ classes for JMS アプリケーションを実行できます。 いずれの場合も、WebSphere MQ classes for JMS は、TCP/IP を介してキュー・マネージャーに接続します。

バインディング・モード

バインディング・モードでキュー・マネージャーを接続するには、キュー・マネージャーが実行されているのと同じシステムで WebSphere MQ classes for JMS アプリケーションを実行する必要があります。

WebSphere MQ classes for JMS は、Java Native Interface (JNI) を使用してキュー・マネージャーに直接接続します。 バインディング・トランスポートを使用するには、WebSphere MQ classes for JMS を、WebSphere MQ Java Native Interface ライブラリーにアクセスできる環境で実行する必要があります。詳細については、730 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照してください。

WebSphere MQ classes for JMS は、*ConnectOption* について以下の値をサポートしています。

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

WebSphere MQ classes for JMS で使用される接続オプションを変更するには、接続ファクトリーのプロパティー CONNOPT を変更します。

接続オプションの詳細については、208 ページの『MQCONNX 呼び出しを使用したキュー・マネージャーへの接続』を参照してください。

バインディング・トランスポートを使用するには、使用する Java ランタイム環境が、 WebSphere MQ classes for JMS の接続先のキュー・マネージャーのコード化文字セット ID (CCSID) をサポートしている必要があります。

Java ランタイム環境でサポートされる CCSID を判別する方法について詳しくは、 WebSphere MQ V7 classes for Java または WebSphere MQ V7 classes for JMS を使用するときに生成される WebSphere MQ FDC (プローブ ID 21) を参照してください。

バインディング、その後にクライアント

これがデフォルトです。 このモードでキュー・マネージャーに接続する場合、 WebSphere MQ classes for JMS アプリケーションは、バインディング・モードで接続を試行します。この場合は、キュー・マネージャーがアプリケーションと同じマシン上にあることが必要です。 この接続が失敗すると、アプリケーションはクライアント・モードでの接続を試行します。この場合は、キュー・マネージャーの場所はローカルのアプリケーションと同じマシン上でも、またはリモートでも構いません。

WebSphere MQ classes for JMS アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成する

WebSphere MQ classes for JMS アプリケーションがクライアント・モードで接続できるようにキュー・マネージャーを構成するには、サーバー接続チャネル定義を作成し、リスナーを開始する必要があります。

z/OS では、クライアント接続機構をインストールしておく必要があります。

サーバー接続チャネル定義の作成

サーバー接続チャネル定義は、MQSC コマンド DEFINE CHANNEL を使用してすべてのプラットフォームで作成できます。 次のような例があります。

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i では、次の例のように、CL コマンド CRTMQMCHL を代わりに使用することができます。

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
                                TRPTYPE(*TCP)
                                MQMNAME(QMGRNAME)
```

このコマンドの *QMGRNAME* は、キュー・マネージャーの名前です。

Linux および Windows で実行される IBM WebSphere MQ Explorer、または z/OS の操作パネルおよび制御パネルを使用して、サーバー接続チャネル定義を作成することもできます。

チャネルの名前 (上記の例では JAVA.CHANNEL) は、アプリケーションがキュー・マネージャーとの接続で使用する接続ファクトリーの CHANNEL プロパティーで指定されたチャネル名と同じでなければなりません。 CHANNEL プロパティーのデフォルト値は SYSTEM.DEF.SVRCONN です。

リスナーの開始

キュー・マネージャーのリスナーがまだ開始されていない場合は、開始する必要があります。

リスナーは、z/OS を除くすべてのプラットフォームで MQSC コマンド START LISTENER を使用することによって開始できますが、最初に MQSC コマンド DEFINE LISTENER を使用してリスナー・オブジェクトを作成する必要があります。 次のような例があります。

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

UNIX、Linux、および Windows システムでは、次の例のように、制御コマンド **runmqlsr** を使用してリスナーを開始することもできます。

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

このコマンドの *QMgrName* は、キュー・マネージャーの名前です。

リスナーの開始は WebSphere MQ エクスプローラーからも行えます。これは Linux および Windows、または z/OS 上の操作パネルおよび制御パネルで実行されます。

リスナーが listen しているポートの番号は、アプリケーションがキュー・マネージャーとの接続で使用する接続ファクトリーの PORT プロパティーで指定されたポート番号と同じでなければなりません。 PORT プロパティーのデフォルト値は 1414 です。

WebSphere MQ classes for JMS の Point-to-Point インストール検査テスト

Point-to-Point インストール検査テスト (IVT) プログラムは、 WebSphere MQ classes for JMS に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、メッセージを SYSTEM.DEFAULT.LOCAL.QUEUE というキューに送信した後、メッセージをキューから受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

まず、インストール検査テストを JNDI を使わずに実行します。 このテストは自己完結型であり、ディレクトリー・サービスの使用を必要としないからです。 管理対象オブジェクトの説明については、943 ページの『JMS オブジェクト・タイプ』を参照してください。

JNDI を使用しない Point-to-Point インストール検査テスト

このテストで IVT プログラムは、必要なオブジェクトをすべて実行時に動的に作成および構成し、JNDI は使用しません。

IVT プログラムを実行するためのスクリプトが用意されています。 このスクリプトは、 UNIX and Linux システムでは IVTRun、 Windows では IVTRun.bat と呼ばれており、 WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

テストをバインディング・モードで実行するには、以下のコマンドを入力します。

```
IVTRun -nojndi [-m qmgr] [-v providerVersion] [-t]
```

クライアント・モードでテストを実行するには、108 ページの『サンプル・プログラムの作成と実行』の説明に従って、まずキュー・マネージャーをセットアップします。 使用されるチャネルのデフォルトは **SYSTEM.DEF.SVRCONN** であり、使用されるキューは **SYSTEM.DEFAULT.LOCAL.QUEUE** であることに注意してから、次のコマンドを入力します。

```
IVTRun -nojndi -client -m qmgr -host hostname [-port port] [-channel channel]
       [-v providerVersion] [-ccsid ccsid] [-t]
```

z/OS システムには、これに相当するスクリプトは用意されていませんが、以下のコマンドで Java クラスを直接呼び出して、IVT をバインディング・モードで実行できます。

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr] [-v providerVersion] [-t]
```

クラスパスには com.ibm.mqjms.jar を含める必要があります。

コマンドのパラメーターの意味は次のとおりです。

-m *qmgr*
IVT プログラムの接続先のキュー・マネージャーの名前。 テストをバインディング・モードで実行するときにこのパラメーターを省略すると、IVT プログラムはデフォルトのキュー・マネージャーに接続します。

-host *hostname*
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

-port *port*
キュー・マネージャーのリスナーが listen しているポートの番号。 デフォルト値は 1414 です。

-channel *channel*
IVT プログラムがキュー・マネージャーへの接続で使用する MQI チャネルの名前。 デフォルト値は SYSTEM.DEF.SVRCONN です。

-v *providerVersion*
IVT プログラムの接続先として想定されるキュー・マネージャーのリリース・レベル。

このパラメーターは、MQQueueConnectionFactory オブジェクトの PROVIDERVERSION プロパティーを設定するために使用され、PROVIDERVERSION プロパティーと同じ有効値を持ちます。 それで、このパラメーターに関する詳細 (その有効値を含む) は、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーにある PROVIDERVERSION プロパティーの説明を参照してください。

デフォルト値は unspecified です。

-ccsid *ccsid*
接続で使用されるコード化文字セットまたはコード・ページの ID (CCSID)。 デフォルト値は 819 です。

-t
トレースのスイッチをオンにします。 デフォルトでは、トレースはオフになっています。

テストが正常に行われると、次の出力例のような出力が生成されます。

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
```

```
   JMSMessage class: jms_text
   JMSType:          null
   JMSDeliveryMode:  2
   JMSExpiration:    0
   JMSPriority:      4
   JMSMessageID:     ID:414d5120514d5f6d627720202020202001edb14620005e03
   JMSTimestamp:     1187170264000
   JMSCorrelationID: null
   JMSDestination:   queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
   JMSReplyTo:       null
   JMSRedelivered:   false
     JMSXUserID: mwhite
     JMS_IBM_Encoding: 273
     JMS_IBM_PutApplType: 28
     JMSXAppID: WebSphere MQ Client for Java
     JMSXDeliveryCount: 1
     JMS_IBM_PutDate: 20070815
     JMS_IBM_PutTime: 09310400
     JMS_IBM_Format: MQSTR
     JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

JNDI を使用した Point-to-Point インストール検査テスト

このテストでは、IVT プログラムは JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出します。

テストを実行する前に、Lightweight Directory Access Protocol (LDAP) サーバーまたはローカル・ファイル・システムに基づくディレクトリー・サービスを構成しておく必要があります。 また、ディレクトリー・サービスを使用して管理対象オブジェクトを保管できるように WebSphere MQ JMS 管理ツールを構成しておく必要もあります。 これらの前提条件について詳しくは、723 ページの『WebSphere MQ classes for JMS の前提条件』を参照してください。 WebSphere MQ JMS 管理ツールを構成する方法については、939 ページの『JMS 管理ツールの構成』を参照してください。

IVT プログラムで JNDI を使用して MQQueueConnectionFactory オブジェクトおよび MQQueue オブジェクトをディレクトリー・サービスから取り出せるようにしておく必要があります。 これらの管理オブジェクトを作成するためのスクリプトが用意されています。 このスクリプトは、UNIX and Linux システムでは IVTSetup、Windows では IVTSetup.bat と呼ばれ、WebSphere MQ classes for JMS インストール・ディレクトリーの bin サブディレクトリーにあります。 スクリプトを実行するには、以下のコマンドを入力します。

```
IVTSetup
```

このスクリプトは WebSphere MQ JMS 管理ツールを呼び出し、管理対象オブジェクトを作成します。

MQQueueConnectionFactory オブジェクトは ivtQCF という名前でバインドされ、すべてそのプロパティーのデフォルト値で作成されます。これは、IVT プログラムがバインディング・モードで実行され、デフォルトのキュー・マネージャーに接続することを意味します。 IVT プログラムをクライアント・モードで実行する場合、またはデフォルト以外のキュー・マネージャーに接続する場合は、WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使用して、MQQueueConnectionFactory オブジェクトの該当するプロパティーを変更する必要があります。 WebSphere MQ JMS 管理ツールの使い方については、938 ページの『WebSphere MQ JMS 管理ツールの使用』を参照してください。 WebSphere MQ エクスプローラーの使い方については、WebSphere MQ エクスプローラーに付属するヘルプを参照してください。

MQQueue オブジェクトは ivtQ という名前でバインドされ、QUEUE プロパティー (SYSTEM.DEFAULT.LOCAL.QUEUE という値を持つ) を除き、すべてそのプロパティーのデフォルト値で作成されます。

管理対象オブジェクトを作成した後、IVT プログラムを実行できます。 JNDI を使用してテストを実行するには、以下のコマンドを入力します。

```
IVTRun -url "providerURL" [-icf initCtxFact] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-url "*providerURL*"
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://hostname/contextName`(LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/directoryPath`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

URL は引用符 (") で囲む必要があります。

-icf *initCtxFact*
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値でなければなりません。

- `com.sun.jndi.ldap.LdapCtxFactory`(LDAP サーバーに基づくディレクトリー・サービスの場合)。 これがデフォルト値です。
- `com.sun.jndi.fscontext.RefFSContextFactory`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)。

-t
トレースのスイッチをオンにします。 デフォルトでは、トレースはオフになっています。

テストが正常に行われると、JNDI を使用しない場合にテストが正常に行われたときに生成される出力と同様の出力が生成されます。 主な違いは、テストで JNDI を使用して MQQueueConnectionFactory オブジェクトおよび MQQueue オブジェクトが取り出されたことが出力で示される点です。

必須ではありませんが、IVTSetup スクリプトによって作成された管理対象オブジェクトを削除してテスト後にタイディアップすることをお勧めします。 これを行うためのスクリプトが用意されています。 このスクリプトは、 UNIX and Linux システムでは IVTTidy、 Windows では IVTTidy.bat と呼ばれており、WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

Point-to-Point インストール検査テストの問題判別

インストール検査テストは、次のような理由で失敗することがあります。

- クラスを検出できないことを示すメッセージが IVT プログラムによって書き込まれる場合は、728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』で説明されているとおりにクラス・パスが正しく設定されているか確認してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

  ```
  Failed to connect to queue manager 'qmgr' with connection mode 'connMode'
  and host name 'hostname'
  ```

  関連する理由コードは 2059 です。 メッセージの変数の意味は次のとおりです。

  qmgr
  IVT プログラムの接続試行先のキュー・マネージャーの名前。 IVT プログラムがバインディング・モードでデフォルトのキュー・マネージャーに接続を試行している場合、このメッセージ挿入はブランクになります。

  connMode
  接続モード。 `Bindings` または `Client` のいずれかです。

  hostname
  キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

  このメッセージは、IVT プログラムの接続試行先のキュー・マネージャーが使用不可であることを意味します。 キュー・マネージャーが稼働していること、および IVT プログラムがデフォルトのキュー・マネージャーに接続を試行している場合は、そのキュー・マネージャーがシステムのデフォルトのキュー・マネージャーとして定義されていることを確認してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
```

  このメッセージは、IVT プログラムが接続されているキュー・マネージャーにキュー SYSTEM.DEFAULT.LOCAL.QUEUE が存在しないことを意味します。 あるいは、キューが存在していてもメッセージの書き込みおよび取得が有効になっていないために IVT プログラムはそのキューを開くことができません。 キューが存在すること、およびメッセージの書き込みと取得が有効になっていることを確認してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
Unable to bind to object
```

  このメッセージは、LDAP サーバーとの接続は存在するものの、LDAP サーバーが正しく構成されていないことを意味します。 LDAP サーバーが Java オブジェクトを保管するように構成されていないか、オブジェクトまたはサフィックスに対する許可が正しくありません。 この状態でのヘルプ情報については、ご使用の LDAP サーバーの資料を参照してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

  このメッセージは、システムからのクライアント接続を受け入れるようにキュー・マネージャーが正しくセットアップされていないことを意味します。 詳細は 108 ページの『サンプル・プログラムの作成と実行』を参照してください。

WebSphere MQ classes for JMS のパブリッシュ/サブスクライブ・インストール検査テスト

パブリッシュ/サブスクライブ・インストール検査テスト (IVT) プログラムは、 WebSphere MQ classes for JMS に付属しています。 プログラムはキュー・マネージャーにバインディング・モードまたはクライアント・モードのいずれかで接続し、トピックにサブスクライブし、メッセージをトピックにパブリッシュした後、そのメッセージを受信します。 プログラムは必要なオブジェクトをすべて実行時に動的に作成および構成することができます。 また、JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出すこともできます。

まず、インストール検査テストを JNDI を使わずに実行します。 このテストは自己完結型であり、ディレクトリー・サービスの使用を必要としないからです。 管理対象オブジェクトの説明については、943 ページの『JMS オブジェクト・タイプ』を参照してください。

JNDI を使用しないパブリッシュ/サブスクライブ・インストール検査テスト

このテストで IVT プログラムは、必要なオブジェクトをすべて実行時に動的に作成および構成し、JNDI は使用しません。

IVT プログラムを実行するためのスクリプトが用意されています。 このスクリプトは、 UNIX and Linux システムでは PSIVTRun、 Windows では PSIVTRun.bat と呼ばれており、 WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

テストをバインディング・モードで実行するには、以下のコマンドを入力します。

```
PSIVTRun -nojndi [-m qmgr] [-bqm brokerQmgr] [-v providerVersion] [-t]
```

クライアント・モードでテストを実行する場合は、108 ページの『サンプル・プログラムの作成と実行』の説明に従ってまずキュー・マネージャーをセットアップします。その際には、使用されるチャネルがデフォルトでは SYSTEM.DEF.SVRCONN であることに注意してください。次いで、以下のコマンドを入力します。

```
PSIVTRun -nojndi -client -m qmgr -host hostname [-port port] [-channel channel]
         [-bqm brokerQmgr] [-v providerVersion] [-ccsid ccsid] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-m *qmgr*
IVT プログラムの接続先のキュー・マネージャーの名前。 テストをバインディング・モードで実行するときにこのパラメーターを省略すると、IVT プログラムはデフォルトのキュー・マネージャーに接続します。

-host *hostname*
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

-port *port*
キュー・マネージャーのリスナーが listen しているポートの番号。 デフォルト値は 1414 です。

-channel *channel*
IVT プログラムがキュー・マネージャーへの接続で使用する MQI チャネルの名前。 デフォルト値は SYSTEM.DEF.SVRCONN です。

-bqm *brokerQmgr*
ブローカーが稼働しているキュー・マネージャーの名前。 デフォルト値は、IVT プログラムの接続先のキュー・マネージャーの名前です。

このパラメーターは、-v パラメーターがキュー・マネージャーのバージョン番号として 7 未満を指定し、パブリッシュ/サブスクライブ・ブローカーとして WebSphere Event Broker または WebSphere Message Broker を使用している場合にのみ関係します。

-v *providerVersion*
IVT プログラムの接続先として想定されるキュー・マネージャーのリリース・レベル。

このパラメーターは、MQTopicConnectionFactory オブジェクトの PROVIDERVERSION プロパティーを設定するために使用され、PROVIDERVERSION プロパティーと同じ有効値を持ちます。 それで、このパラメーターに関する詳細 (その有効値を含む) は、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーにある PROVIDERVERSION プロパティーの説明を参照してください。

デフォルト値は unspecified です。

-ccsid *ccsid*
接続で使用されるコード化文字セットまたはコード・ページの ID (CCSID)。 デフォルト値は 819 です。

-t
トレースのスイッチをオンにします。 デフォルトでは、トレースはオフになっています。

テストが正常に行われると、次の出力例のような出力が生成されます。

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f6d627720202020202001edb14620006706
  JMSTimestamp:     1187182520203
  JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
  JMSDestination:   topic://MQJMS/PSIVT/Information
  JMSReplyTo:       null
```

```
   JMSRedelivered:   false
     JMSXUserID: mwhite
     JMS_IBM_Encoding: 273
     JMS_IBM_PutApplType: 26
     JMSXAppID: QM_mbw
     JMSXDeliveryCount: 1
     JMS_IBM_PutDate: 20070815
     JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
     JMS_IBM_PutTime: 12552020
     JMS_IBM_Format: MQSTR
     JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

JNDI を使用したパブリッシュ/サブスクライブ・インストール検査テスト

このテストでは、IVT プログラムは JNDI を使用して管理対象オブジェクトをディレクトリー・サービスから取り出します。

テストを実行する前に、Lightweight Directory Access Protocol (LDAP) サーバーまたはローカル・ファイル・システムに基づくディレクトリー・サービスを構成しておく必要があります。 また、ディレクトリー・サービスを使用して管理対象オブジェクトを保管できるように WebSphere MQ JMS 管理ツールを構成しておく必要もあります。 これらの前提条件について詳しくは、723 ページの『WebSphere MQ classes for JMS の前提条件』を参照してください。 WebSphere MQ JMS 管理ツールを構成する方法については、939 ページの『JMS 管理ツールの構成』を参照してください。

IVT プログラムで JNDI を使用して MQTopicConnectionFactory オブジェクトおよび MQTopic オブジェクトをディレクトリー・サービスから取り出せるようにしておく必要があります。 これらの管理オブジェクトを作成するためのスクリプトが用意されています。 このスクリプトは、 UNIX and Linux システムでは IVTSetup、 Windows では IVTSetup.bat と呼ばれ、 WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。 スクリプトを実行するには、以下のコマンドを入力します。

```
IVTSetup
```

このスクリプトは WebSphere MQ JMS 管理ツールを呼び出し、管理対象オブジェクトを作成します。

MQTopicConnectionFactory オブジェクトは ivtTCF という名前でバインドされ、すべてそのプロパティーのデフォルト値で作成されます。これは、IVT プログラムがバインディング・モードで実行され、デフォルトのキュー・マネージャーに接続し、組み込みパブリッシュ/サブスクライブ機能を使用することを意味します。 IVT プログラムをクライアント・モードで実行する場合、またはデフォルト以外のキュー・マネージャーに接続する場合、あるいは組み込みパブリッシュ/サブスクライブ機能の代わりに WebSphere Event Broker または WebSphere Message Broker を使用する場合には、WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使用して、MQTopicConnectionFactory オブジェクトの該当するプロパティーを変更する必要があります。 WebSphere MQ JMS 管理ツールの使い方については、938 ページの『WebSphere MQ JMS 管理ツールの使用』を参照してください。 WebSphere MQ エクスプローラーの使い方については、WebSphere MQ エクスプローラーに付属するヘルプを参照してください。

MQTopic オブジェクトは ivtT という名前でバインドされ、TOPIC プロパティー (MQJMS/PSIVT/Information という値を持つ) を除き、すべてそのプロパティーのデフォルト値で作成されます。

管理対象オブジェクトを作成した後、IVT プログラムを実行できます。 JNDI を使用してテストを実行するには、以下のコマンドを入力します。

```
PSIVTRun -url "providerURL" [-icf initCtxFact] [-t]
```

コマンドのパラメーターの意味は次のとおりです。

-url "*providerURL*"
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://hostname/contextName`(LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/directoryPath`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

URL は引用符 (") で囲む必要があります。

-icf *initCtxFact*
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値でなければなりません。

- `com.sun.jndi.ldap.LdapCtxFactory`(LDAP サーバーに基づくディレクトリー・サービスの場合)。 これがデフォルト値です。
- `com.sun.jndi.fscontext.RefFSContextFactory`(ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)。

-t
トレースのスイッチをオンにします。 デフォルトでは、トレースはオフになっています。

テストが正常に行われると、JNDI を使用しない場合にテストが正常に行われたときに生成される出力と同様の出力が生成されます。 主な違いは、テストで JNDI を使用して MQTopicConnectionFactory オブジェクトおよび MQTopic オブジェクトが取り出されたことが出力で示される点です。

必須ではありませんが、IVTSetup スクリプトによって作成された管理対象オブジェクトを削除してテスト後にタイディアップすることをお勧めします。 これを行うためのスクリプトが用意されています。 このスクリプトは、 UNIX and Linux システムでは IVTTidy、 Windows では IVTTidy.bat と呼ばれており、WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

パブリッシュ/サブスクライブ・インストール検査テストの問題判別

インストール検査テストは、次のような理由で失敗することがあります。

- クラスを検出できないことを示すメッセージが IVT プログラムによって書き込まれる場合は、728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』で説明されているとおりにクラス・パスが正しく設定されているか確認してください。
- 次のようなメッセージとともにテストが失敗する場合があります。

```
Failed to connect to queue manager 'qmgr' with
connection mode 'connMode' and host name 'hostname'
```

関連する理由コードは 2059 です。 メッセージの変数の意味は次のとおりです。

qmgr
IVT プログラムの接続試行先のキュー・マネージャーの名前。 IVT プログラムがバインディング・モードでデフォルトのキュー・マネージャーに接続を試行している場合、このメッセージ挿入はブランクになります。

connMode
接続モード。 `Bindings` または `Client` のいずれかです。

hostname
キュー・マネージャーが稼働しているシステムのホスト名または IP アドレス。

このメッセージは、IVT プログラムの接続試行先のキュー・マネージャーが使用不可であることを意味します。 キュー・マネージャーが稼働していること、および IVT プログラムがデフォルトのキュー・マネージャーに接続を試行している場合は、そのキュー・マネージャーがシステムのデフォルトのキュー・マネージャーとして定義されていることを確認してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
Unable to bind to object
```

このメッセージは、LDAP サーバーとの接続は存在するものの、LDAP サーバーが正しく構成されていないことを意味します。 LDAP サーバーが Java オブジェクトを保管するように構成されていないか、オブ

ジェクトまたはサフィックスに対する許可が正しくありません。 この状態でのヘルプ情報については、ご使用の LDAP サーバーの資料を参照してください。

- 次のようなメッセージとともにテストが失敗する場合があります。

```
The security authentication was not valid that was supplied for
QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
```

このメッセージは、システムからのクライアント接続を受け入れるようにキュー・マネージャーが正しくセットアップされていないことを意味します。 詳細については、108 ページの『サンプル・プログラムの作成と実行』を参照してください。

WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム

IVT プログラムは EAR ファイルとして提供されます。 このプログラムを使用するには、このプログラムをデプロイし、JCA リソースとして一部のオブジェクトを定義する必要があります。

インストール検査テスト (IVT) プログラムは、wmq.jmsra.ivt.ear という Enterprise Archive (EAR) ファイルで提供されます。 このファイルは、WebSphere MQ リソース・アダプター RAR ファイル、wmq.jmsra.rar と同じディレクトリーに、WebSphere MQ classes for JMS と共にインストールされます。 これらのファイルがインストールされる場所について詳しくは、740 ページの『WebSphere MQ リソース・アダプターのインストール』を参照してください。

IVT プログラムはアプリケーション・サーバー上にデプロイする必要があります。 IVT プログラムには、サーブレットと、WebSphere MQ キューとのメッセージの送受信が可能かどうかをテストする MDB が含まれています。 オプションで、IVT プログラムを使用して、WebSphere MQ リソース・アダプターが分散トランザクションをサポートするように正しく構成されていることを検証できます。

IVT プログラムを実行するには、その前に ConnectionFactory オブジェクト、Queue オブジェクト、および (場合によっては) Activation Specification オブジェクトを JCA リソースとして定義する必要があります。また、ご使用のアプリケーション・サーバーが、これらの定義から JMS オブジェクトを作成して、それらを JNDI 名前空間にバインドするようにする必要があります。 オブジェクトのプロパティーは選択できますが、以下にプロパティー・セットの単純な例を示します。

ConnectionFactory オブジェクト

```
channel:               SYSTEM.DEF.SVRCONN
hostName:              localhost
port:                  1414
queueManager:          ExampleQM
transportType:         CLIENT
```

Queue オブジェクト

```
baseQueueManagerName:  ExampleQM
baseQueueName:         TEST.QUEUE
```

デフォルトでは、IVT プログラムは、JNDI ネーム・スペース内で jms/ivt/IVTCF という名前の ConnectionFactory オブジェクト、および jms/ivt/IVTQueue という名前の Queue オブジェクトがバインドされることを予期します。 別の名前を使用することもできますが、その場合はオブジェクトの名前を IVT プログラムの最初のページに入力し、EAR ファイルを適切に変更する必要があります。

IVT プログラムをデプロイし、アプリケーション・サーバーが JMS オブジェクトを作成して、それらを JNDI ネーム・スペースにバインドした後、Web ブラウザーに以下のフォーマットで URL を入力することにより、IVT プログラムを開始できます。

```
http://app_server_host:port/WMQ_IVT/
```

ここで、*app_server_host* は、アプリケーション・サーバーが稼働しているシステムの IP アドレスまたはホスト名で、*port* は、アプリケーション・サーバーが listen している TCP ポートの番号です。 以下が例となります。

```
http://localhost:9080/WMQ_IVT/
```

788 ページの図 122 は、IVT プログラムの初期ページを示します。

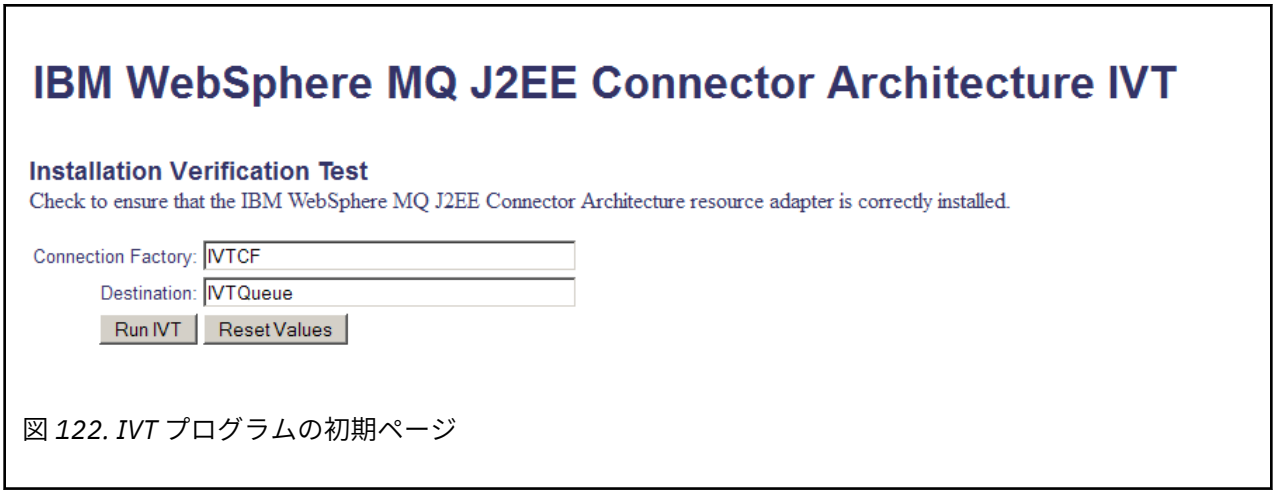


図 *122. IVT* プログラムの初期ページ

テストを実行するには、「**IVT の実行**」をクリックします。788 ページの図 123 IVT が成功した場合に表示されるページを示します。

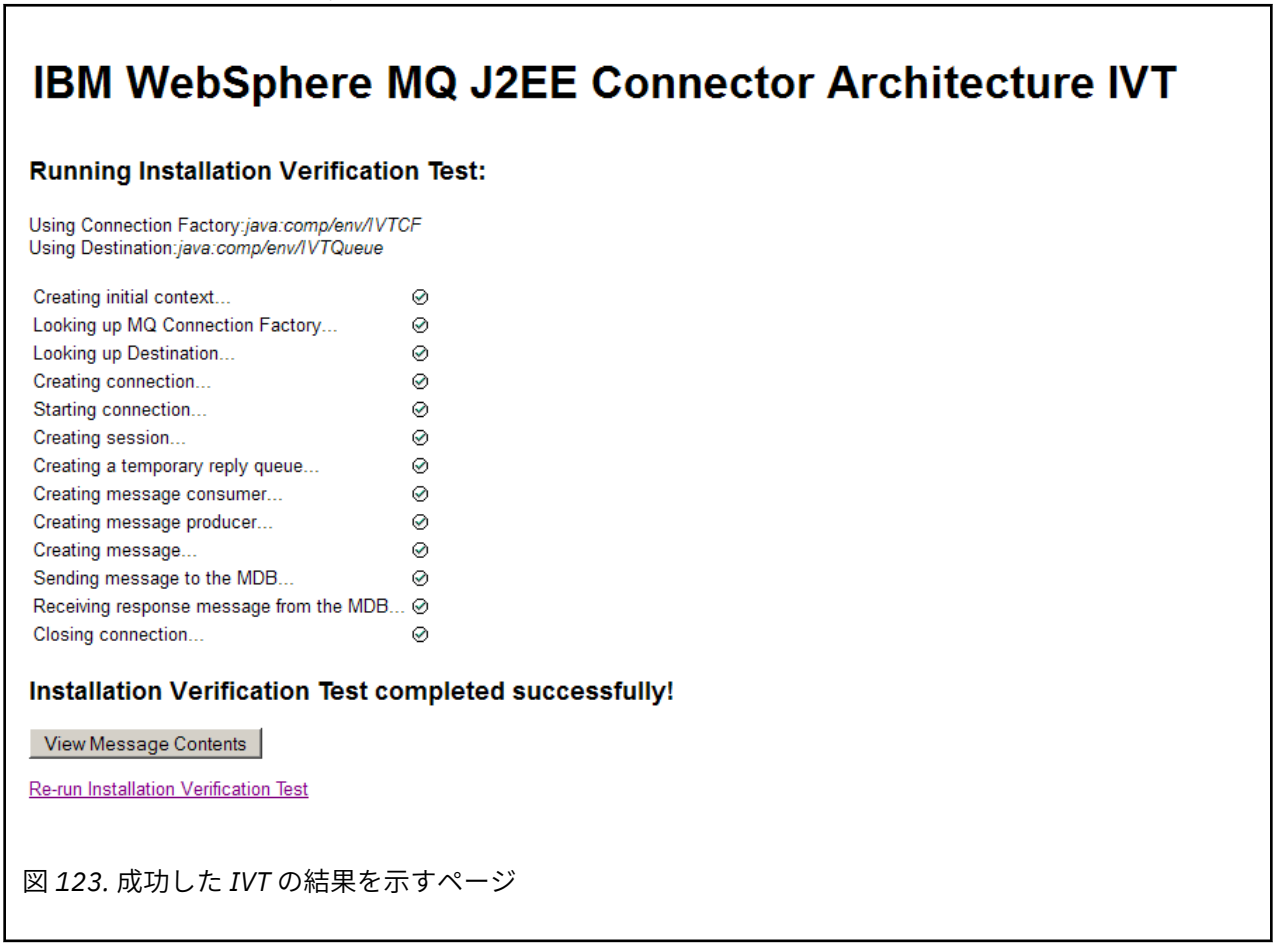


図 *123.* 成功した *IVT* の結果を示すページ

IVT が失敗すると、789 ページの図 124 に示すようなページが表示されます。 失敗の原因についてさらに詳しい情報を入手するには、「**スタック・トレースの表示**」をクリックします。

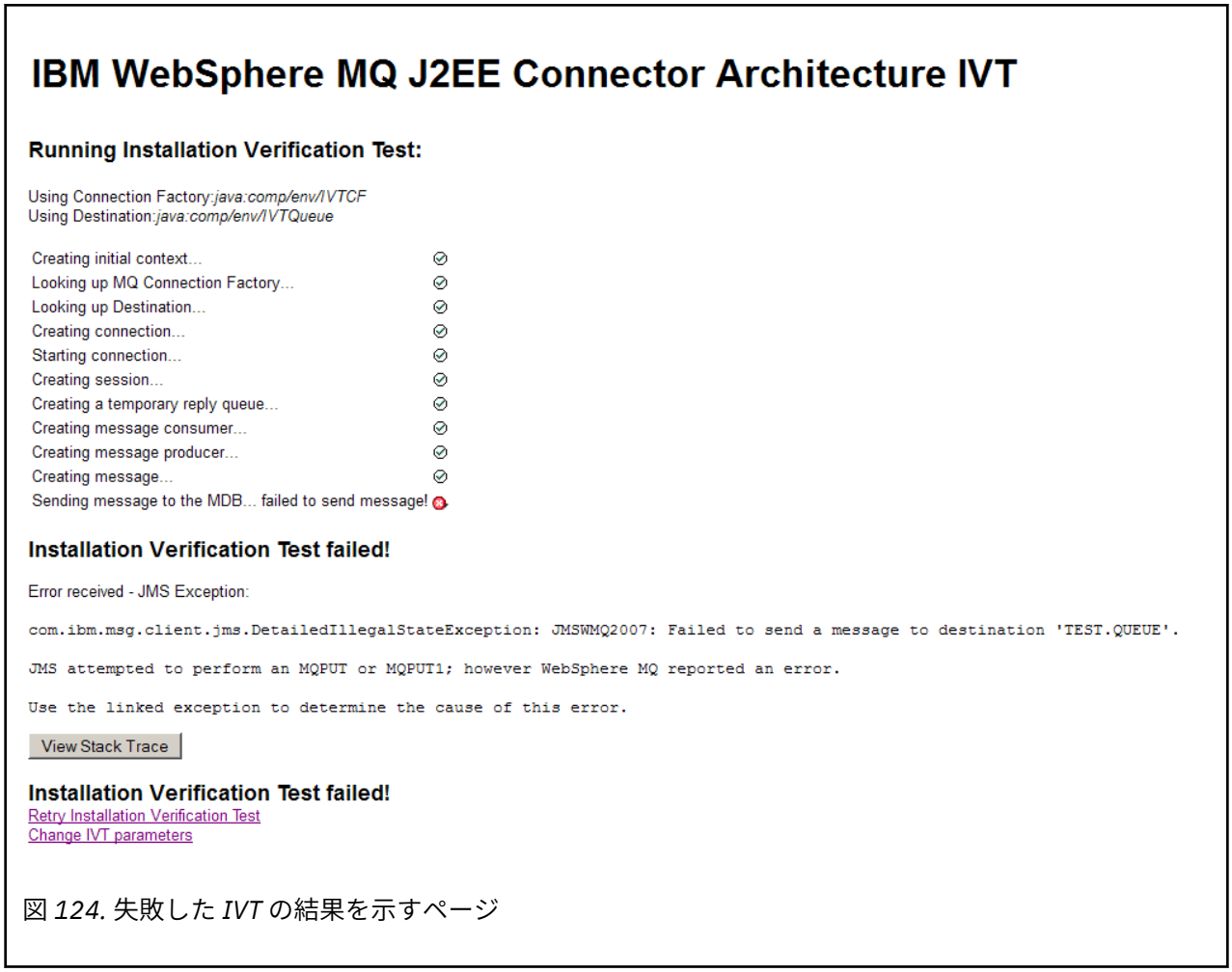


図 *124.* 失敗した *IVT* の結果を示すページ

IVT アプリケーションを JBoss および WAS CE アプリケーション・サーバーにデプロイするために提供されているユーティリティー・スクリプトについて詳しくは、以下を参照してください。

関連タスク

789 ページの『WAS CE での MQ リソース・アダプターのインストールとテスト』
WebSphere Application Server CE で IBM WebSphere MQ リソース・アダプターをインストールし、インストール検査テスト (IVT) アプリケーションを実行します。

792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』
IBM WebSphere MQ リソース・アダプターを JBoss AS 5.1 または 6 にインストールした後、インストール検査テスト (IVT) アプリケーションをインストールして実行することにより、リソース・アダプターのインストールをテストできます。

WAS CE での MQ リソース・アダプターのインストールとテスト

WebSphere Application Server CE で IBM WebSphere MQ リソース・アダプターをインストールし、インストール検査テスト (IVT) アプリケーションを実行します。

始める前に

このタスクは、 WebSphere Application Server CE サーバーが稼働していること、およびその標準的な管理タスクに精通していることを前提としています。 さらに、このタスクは IBM WebSphere MQ がローカル・システムにインストールされていることと、ユーザーが標準的な管理タスクに精通していることも前提としています。

リソース・アダプターを使用して IBM WebSphere MQ クライアントに接続していて、分散 XA トランザクションを実行する必要がある場合は、 「**クライアント XA のみ**」というマークが付いた追加のステップに従う必要があります。

1. ExampleQM というキュー・マネージャーを作成し、それを 108 ページの『サンプル・プログラムの作成と実行』 の説明に従ってセットアップします。その際には、リスナーがポート 1414 で開始し、使用されるチャネルが SYSTEM.DEF.SVRCONN であり、IVT アプリケーションにより使用されるキューが TEST.QUEUE であることに注意してください。 また、このアプリケーションが一時的な応答キューを作成できるよう、モデル・キュー SYSTEM.DEFAULT.MODEL.QUEUE に DSP および PUT 権限を付与する必要があります。 異なるキュー・マネージャー、接続の詳細、またはキューを使用する場合には、791 ページの『カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ』 を参照してください。
2. リソース・アダプター・ファイル (wmq.jmsra.rar)、IVT アプリケーション (wmq.jmsra.ivt.ear)、および WAS_CE_jmsra_deployment_plan.xml と WAS_CE_jmsra_ivt_deployment_plan.xml deployment plan files を取得します。 これらのファイルのロケーションについて詳しくは、740 ページの『WebSphere MQ リソース・アダプターのインストール』 を参照してください。

バインディング・モード接続およびクライアント・モード接続について詳しくは、777 ページの『WebSphere MQ classes for JMS の接続モード』 を参照してください。

異なるキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合には、791 ページの『カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ』 を参照してください。

手順

1. **クライアント XA のみ**: WAS_CE_jmsra_deployment_plan.xml ファイルのコピーを編集します。
   a) jms/ivt/IVTCF 接続定義を検索して、接続ファクトリーが XA トランザクションに対応するように変更します。
      i) NonXA セクションをコメント化します。

      ```
      <conn:xa-transaction>
      ```

      ii) XA 構成セクションのコメントを外します。

      ```
      <conn:xa-transaction>
          <conn:transaction-caching/>
      </conn:xa-transaction>
      ```

   b) 変更内容を保存します。
2. オプション: **クライアント XA のみ**: トランザクションを必要とする MDB のアセンブリー記述子を変更します。 これにより、IVT 内の MDB は XA トランザクションに参加することを強制されますが、IVT アプリケーションは、この変更を行わなくても引き続き機能します。
   a) wmq.jmsra.ivt.ear ファイルを開きます。
   b) そのファイル内で WMQ_IVT_MDB.jar を開きます。
   c) META-INF/ejb-jar.xml を編集します。
      i) アセンブリー記述子内の次の行をコメント化または削除します。

      ```
      <trans-attribute>NotSupported</trans-attribute>
      ```

      ii) アセンブリー記述子内の次の行のコメントを外します。

      ```
      <trans-attribute>Required</trans-attribute>
      ```

      iii) 変更を保存して、WMQ_IVT_MDB.jar ファイル内のファイルを更新します。
      iv) 変更された WMQ_IVT_MDB.jar ファイルで wmq.jmsra.ivt.ear ファイルを更新します。
3. 変更されたデプロイメント計画ファイルを使用して、リソース・アダプターをサーバーにデプロイします。
   a) これをコマンド行で行うには、以下の WAS CE コマンドを入力します。

   ```
   deploy -u system -p manager deploy wmq.jmsra.rar WAS_CE_jmsra_deployment_plan.xml
   ```

b) Web 管理インターフェースを使用して、「**アプリケーション**」>「**デプロイヤー**」 に移動します。

i) アーカイブを wmq.jmsra.rar ファイルに設定します。

ii) 計画を WAS_CE_jmsra_deployment_plan.xml ファイルに設定します。

iii)「インストール後にアプリケーションを開始」が選択されていることを確認します。

iv)「**インストール**」をクリックします。

4. 提供されたデプロイメント計画を使用して、IVT アプリケーションをサーバーにデプロイします。

a) これは、コマンド行で以下の WAS CE コマンドを使用して実行できます。

```
deploy -u system -p manager deploy wmq.jmsra.ivt.ear WAS_CE_jmsra_ivt_deployment_plan.xml
```

b) Web 管理インターフェースを使用して、「**アプリケーション**」>「**デプロイヤー**」 に移動します。

i) **アーカイブ**を `wmq.jmsra.ivt.ear` ファイルに設定します。

ii) **計画**を `WAS_CE_jmsra_ivt_deployment_plan.xml` ファイルに設定します。

iii)「**インストール後にアプリケーションを開始**」が選択されていることを確認します。

iv)「**インストール**」をクリックします。

5. IVT アプリケーションを実行します。 詳細については、787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』を参照してください。 WAS CE の場合、デフォルトの URL は `http://localhost:8080/WMQ_IVT/` です。

カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ

別のキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に、WebSphere Application Server CE で IVT アプリケーションおよび関連スクリプトを変更する必要があります。

このタスクについて

789 ページの『WAS CE での MQ リソース・アダプターのインストールとテスト』で指定した構成とは別の構成にデプロイする場合、つまり、異なるキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に以下の手順を実行してください。

手順

1. IVT アプリケーションに使用する別のキュー・マネージャーおよびキューを指定する場合は、`WAS_CE_jmsra_deployment_plan.xml` 内でそのキュー・マネージャーとキューに値を設定します。詳しくは、792 ページの『キュー・マネージャーおよびキューの値の設定』を参照してください。
2. メッセージ駆動型 Bean (MDB) の構成で別のキュー・マネージャーとキューを指定する場合は、`WAS_CE_jmsra_ivt_deployment_plan.xml` 内で、使用するキュー・マネージャーとキューの値を設定します。詳しくは、792 ページの『MDB 構成の値の設定』を参照してください。
3. IBM WebSphere MQ にバインディング・モードで接続するようにリソース・アダプターを構成する場合、JNI ライブラリーがシステム・パスまたは WAS CE のパスに存在することを確認します。 詳しくは、789 ページの『WAS CE での MQ リソース・アダプターのインストールとテスト』を参照してください。
4. リソース・アダプターを既にデプロイ済みの場合、以下のコマンドを使用して、変更されたデプロイメント計画を使用してそれを再デプロイし、設定を変更することができます。

```
deploy --user system --password manager redeploy wmq.jmsra.rar
WAS_CE_jmsra_deployment_plan.xml
```

次のタスク

789 ページの『WAS CE での MQ リソース・アダプターのインストールとテスト』での説明に従って、リソース・アダプターのデプロイを続行します。

キュー・マネージャーおよびキューの値の設定

`WAS_CE_jmsra_deployment_plan.xml` で使用するキュー・マネージャーおよびキューの値を設定する方法について説明します。

手順

`WAS_CE_jmsra_deployment_plan.xml` で、IVT アプリケーションで使用するキュー・マネージャーおよびキューの値を設定します。

jms/ivt/IVTCF 接続定義の場合、以下のようにします。

1. queueManager エレメントの値をキュー・マネージャーの名前に設定します。
2. クライアント接続を使用する場合、さまざまなクライアント接続エレメントの値をキュー・マネージャーへの接続に適した値に設定します。
3. バインディング接続を使用する場合、以下のようにします。
   a. transportType エレメントの値を BINDINGS に設定します。
   b. さまざまなクライアント接続エレメントをコメント化または削除します。
4. jms/ivt/IVTQueue メッセージ宛先の場合、baseQueueName エレメントの値を IVT アプリケーション用に作成したキューの名前に設定します。
5. 変更内容を保存します。

MDB 構成の値の設定

`WAS_CE_jmsra_deployment_plan.xml` で MDB 構成の値を設定する方法について説明します。

手順

`WAS_CE_jmsra_ivt_deployment_plan.xml` で、MDB の構成で使用するキュー・マネージャーおよびキューの値を設定します。

WMQ_IVT_MDB メッセージ・ドリブン Bean の場合、以下のようにします。

1. queueManager エレメントの値をキュー・マネージャーの名前に設定します。
2. クライアント接続を使用する場合、さまざまなクライアント接続エレメントの値をキュー・マネージャーへの接続に適した値に設定します。
3. バインディング接続を使用する場合、以下のようにします。
   a. transportType エレメントの値を BINDINGS に設定します。
   b. さまざまなクライアント接続エレメントをコメント化または削除します。
4. 変更内容を保存します。

JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト

IBM WebSphere MQ リソース・アダプターを JBoss AS 5.1 または 6 にインストールした後、インストール検査テスト (IVT) アプリケーションをインストールして実行することにより、リソース・アダプターのインストールをテストできます。

始める前に

重要 : これらの命令は JBoss AS 5.1 および 6 用であり、 JBoss AS 7 には無効です。

JBoss EAP 6.3 でのリソース・アダプターのインストールについては、 795 ページの『JBoss EAP 6.3 でのリソース・アダプターのインストールとテスト』を参照してください。

このタスクは、実行中の JBoss サーバーがあり、その標準的な管理タスクに精通していることを前提としています。 さらに、このタスクは IBM WebSphere MQ がローカル・システムにインストールされていることと、ユーザーが標準的な管理タスクに精通していることも前提としています。

リソース・アダプターを使用して IBM WebSphere MQ クライアントに接続し、分散 XA トランザクションを実行する必要がある場合は、「**クライアント XA のみ**」というマークが付いた追加のステップに従う必要があります。 バインディング・モード接続およびクライアント・モード接続について詳しくは、777 ページの『WebSphere MQ classes for JMS の接続モード』を参照してください。

手順

1. ExampleQM というキュー・マネージャーを作成し、それを 108 ページの『サンプル・プログラムの作成と実行』の説明に従ってセットアップします。

   キュー・マネージャーのセットアップ時に、次の点に注意してください。

   - リスナーはポート 1414 で開始する必要があります。
   - 使用するチャネルは SYSTEM.DEF.SVRCONN です。
   - IVT アプリケーションが使用するキューの名前は TEST.QUEUE です。

   また、モデル・キュー SYSTEM.DEFAULT.MODEL.QUEUE も DSP 権限と PUT 権限を付与され、このアプリケーションが一時的な応答キューを作成できるようにする必要があります。

   異なるキュー・マネージャー、接続の詳細、またはキューを使用する場合には、791 ページの『カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ』を参照してください。
2. リソース・アダプター・ファイル (`wmq.jmsra.rar`)、IVT アプリケーション (`wmq.jmsra.ivt.ear`)、および `jboss-jmsra-ds.xml` ファイルを入手します。

   これらのファイルのロケーションについては、740 ページの『WebSphere MQ リソース・アダプターのインストール』を参照してください。
3. **クライアント XA のみ**: `jboss-jmsra-ds.xml` ファイルを編集して、接続ファクトリーで XA トランザクションを使用可能にします。

   a) 接続ファクトリー定義 `<local-transaction/>`内の行をコメント化または削除します。

   b) 接続ファクトリー定義 `<xa-transaction/>`内の行のコメントを外します。

   c) 変更内容を保存します。
4. **クライアント XA のみ**: (オプション) トランザクションを必要とする MDB のアセンブリー記述子を変更します。 これにより、IVT 内の MDB は XA トランザクションに参加することを強制されますが、IVT アプリケーションは、この変更を行わなくても引き続き機能します。

   a) `wmq.jmsra.ivt.ear` ファイルを開きます。

   b) そのファイル内で `WMQ_IVT_MDB.jar` を開きます。

   c) `META-INF/ejb-jar.xml` を編集します。

      i) アセンブリー記述子内の次の行をコメント化または削除します。

      ```
      <trans-attribute>NotSupported</trans-attribute>
      ```

      ii) アセンブリー記述子内の次の行のコメントを外します。

      ```
      <trans-attribute>Required</trans-attribute>
      ```

      iii) 変更を保存して、`WMQ_IVT_MDB.jar` ファイル内のファイルを更新します。

      iv) 変更された `WMQ_IVT_MDB.jar` で `wmq.jmsra.ivt.ear` ファイルを更新します。
5. `wmq.jmsra.rar` ファイルをディレクトリー `jboss/server/default/deploy` にコピーすることによって、リソース・アダプターをサーバーにデプロイします。
6. `jboss-jmsra-ds.xml` ファイルをディレクトリー `jboss/server/default/deploy` にコピーすることによって、IVT アプリケーションに必要な JMS リソースを作成します。

7. wmq.jmsra.ivt.ear ファイルをディレクトリー jboss/server/default/deploy にコピーすることによって、IVT アプリケーションをデプロイします。
8. IVT アプリケーションを実行します。 詳しくは、787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』を参照してください。 JBoss の場合、デフォルトの URL は http://localhost:8080/WMQ_IVT/です。

カスタム IBM WebSphere MQ 環境を使用する JBoss での IVT アプリケーションのデプロイ

JBoss に IBM WebSphere MQ リソース・アダプターをインストールするときに、別のキュー、キュー・マネージャー、ポート、ホスト、チャネルを使用する場合、またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に、まず JBoss で IVT アプリケーションおよび関連スクリプトを変更する必要があります。

このタスクについて

重要 : 以下の説明は、Java EE バージョン 6 および 5 にのみ適用され、Java EE バージョン 7 には適用されません。 したがって、 JBoss バージョン 8 (WildFly) でのこれらの手順の使用はサポートされていません。

792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』で指定した構成とは別の構成にデプロイする場合、つまり、異なるキュー・マネージャー、キュー、ポート、ホスト、チャネルを使用する場合、 またはクライアント・モードの代わりにバインディング・モードを使用する場合は、リソース・アダプターまたは IVT アプリケーションをデプロイする前に以下の手順を実行してください。

手順

1. IVT アプリケーションで使用する別のキュー・マネージャーおよびキューを指定する場合は、そのキュー・マネージャーとキューに値を設定します。
   a) jms/ivt/IVTCF 接続定義の場合、以下のようにします。
      i) queueManager config-property の値をキュー・マネージャーの名前に設定します。
      ii) クライアント接続を使用する場合、さまざまなクライアント接続エレメントの値をキュー・マネージャーへの接続に適した値に設定します。
      iii) バインディング接続を使用する場合、transportType エレメントの値を BINDINGS に設定してから、さまざまなクライアント接続エレメントをコメント化または削除します。
   b) jms/ivt/IVTQueue mbean の場合、baseQueueName エレメントの値を IVT アプリケーション用に作成したキューの名前に設定します。
   c) 変更内容を保存します。
2. メッセージ駆動型 Bean (MDB) の構成に別のキュー・マネージャーとキューを指定する場合は、そのキュー・マネージャーとキューに接続するよう MDB の構成を変更します。
   a) wmq.jmsra.ivt.ear ファイルを開きます。
   b) そのファイル内で WMQ_IVT_MDB.jar を開きます。
   c) META-INF/ejb-jar.xml を編集します。
      i) queueManager activation-config-property の値をキュー・マネージャーの名前に設定します。
      ii) クライアント接続を使用する場合、さまざまなクライアント接続の activation-config-property の値をキュー・マネージャーへの接続に適した値に設定します。
      iii) バインディング接続を使用する場合、transportType activation-config-property の値を BINDINGS に設定してから、さまざまなクライアント接続エレメントをコメント化または削除します。
   d) 変更を保存して、WMQ_IVT_MDB.jar ファイル内のファイルを更新します。
   e) 変更された WMQ_IVT_MDB.jar で wmq.jmsra.ivt.ear ファイルを更新します。

3. バインディング・モードで IBM WebSphere MQ に接続するようにリソース・アダプターを構成する場合は、JNI ライブラリーがシステム・パスまたは JBoss のパスにあることを確認してください。 詳細については、730 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照してください。

次のタスク

792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』での説明に従って、リソース・アダプターのデプロイを続行します。

.

JBoss EAP 6.3 でのリソース・アダプターのインストールとテスト

IBM WebSphere MQ リソース・アダプターを JBoss Enterprise Application Platform (EAP) 6.3 にインストールした後、スタンドアロン・サーバーまたは管理対象ドメインで実行されているサーバーのいずれかで、インストール検査テスト (IVT) アプリケーションをインストールして実行することにより、リソース・アダプターのインストールをテストできます。

このタスクについて

重要 : 以下の説明は、 JBoss EAP 6.3 専用です。 JBoss AS 5.1 および 6 でのリソース・アダプターのインストールについては、792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』を参照してください。

このタスクは、実行中の JBoss サーバーがあり、その標準的な管理タスクに精通していることを前提としています。 また、このタスクでは、ローカル・システムに IBM WebSphere MQ がインストールされていること、および標準管理に精通していることも前提としています。

手順

1. ExampleQM というキュー・マネージャーを作成し、それを108 ページの『サンプル・プログラムの作成と実行』の説明に従ってセットアップします。

   キュー・マネージャーのセットアップ時に、次の点に注意してください。

   - リスナーはポート 1414 で開始する必要があります。
   - 使用するチャネルは SYSTEM.DEF.SVRCONN です。
   - IVT アプリケーションが使用するキューの名前は TEST.QUEUE です。

   また、モデル・キュー SYSTEM.DEFAULT.MODEL.QUEUE も DSP 権限と PUT 権限を付与され、このアプリケーションが一時的な応答キューを作成できるようにする必要があります。

   異なるキュー・マネージャー、接続の詳細、またはキューを使用する場合には、791 ページの『カスタム MQ 環境の WAS CE での IVT アプリケーションのデプロイ』を参照してください。
2. リソース・アダプター・ファイル (`wmq.jmsra.rar`) および IVT アプリケーション (`wmq.jmsra.ivt.ear`) を入手します。

   これらのファイルのロケーションについては、740 ページの『WebSphere MQ リソース・アダプターのインストール』を参照してください。
3. リソース・アダプターをインストールしてから、インストール検査テスト (IVT) アプリケーションを実行することによってインストール済み環境をテストします。

   - スタンドアロン・サーバーにリソース・アダプターをインストールする場合は、795 ページの『スタンドアロン・サーバーでのインストールとテスト』を参照してください。
   - 管理対象ドメインで実行中のサーバーにリソース・アダプターをインストールする場合は、797 ページの『管理対象ドメインで実行中のサーバーでのインストールとテスト』を参照してください。

スタンドアロン・サーバーでのインストールとテスト

スタンドアロン・サーバー上の JBoss EAP 6.3 に IBM WebSphere MQ リソース・アダプターをインストールした後、インストール検査テスト (IVT) アプリケーションをインストールして実行することにより、リソース・アダプターのインストールをテストできます。

このタスクについて

このタスクの情報は、スタンドアロン・サーバーにリソース・アダプターをインストールしてテストする場合に当てはまります。 管理対象ドメインで実行中のサーバーにリソース・アダプターをインストールする場合は、797 ページの『管理対象ドメインで実行中のサーバーでのインストールとテスト』を参照してください。

重要 : 以下の説明は、 JBoss EAP 6.3 専用です。 JBoss AS 5.1 および 6 でのリソース・アダプターのインストールについては、 792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』を参照してください。

手順

1. wmq.jmsra.rar ファイルをディレクトリー <*EAP_HOME*>/standalone/deployments にコピーすることで、リソース・アダプターをサーバーにデプロイします。
2. 以下の項目を <*EAP_HOME*>/standalone/configuration/standalone-full.xml ファイルの <resource-adapters> セクションに追加して、IVT アプリケーションに必要な JMS リソースを作成します。

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
     <connection-definition
          class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
          jndi-name="java:jboss/jms/ivt/IVTCF"
          enabled="true"
          use-java-context="true"
          pool-name="IVTCF">
       <config-property name="port">
           1414
       </config-property>
       <config-property name="hostName">
           localhost
       </config-property>
       <config-property name="channel">
           SYSTEM.DEF.SVRCONN
       </config-property>
       <config-property name="transportType">
           CLIENT
       </config-property>
       <config-property name="queueManager">
           ExampleQM
       </config-property>
     </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
          TEST.QUEUE
      </config-property>
    </admin-object>
  </admin-objects>
</resource-adapter>
```

3. 次の情報をアプリケーション・サーバーの始動パラメーターに追加します。

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. wmq.jmsra.ivt.ear ファイルをディレクトリー <*EAP_HOME*>/standalone/deployments にコピーして、IVT アプリケーションをデプロイします。
5. アプリケーション・サーバーを始動します。
6. IVT アプリケーションを実行します。

詳しくは、787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』を参照してください。 JBoss の場合、デフォルトの URL は http://localhost:8080/WMQ_IVT/です。

注 : IVT アプリケーションの実行に必要な JMS リソースに使用される JNDI 名は次のとおりです。

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

上記で指定した URL を使用して IVT アプリケーションを起動する場合は、該当するフィールドにこれらのリソースの JNDI 名を入力し、「**IVT の実行**」をクリックしてアプリケーションを実行します。

管理対象ドメインで実行中のサーバーでのインストールとテスト

管理対象ドメインで稼働しているサーバー上の JBoss EAP 6.3 に IBM WebSphere MQ リソース・アダプターをインストールした後、インストール検査テスト (IVT) アプリケーションをインストールして実行することにより、リソース・アダプターのインストールをテストできます。

このタスクについて

このタスクの情報は、管理対象ドメインで実行中のサーバーにリソース・アダプターをインストールしてテストする場合に当てはまります。 スタンドアロン・サーバーにリソース・アダプターをインストールする場合は、795 ページの『スタンドアロン・サーバーでのインストールとテスト』を参照してください。

重要 : 以下の説明は、 JBoss EAP 6.3 専用です。 JBoss AS 5.1 および 6 でのリソース・アダプターのインストールについては、 792 ページの『JBoss AS 5.1 および 6 でのリソース・アダプターのインストールとテスト』を参照してください。

手順

1. JBoss 管理コンソールまたは管理 CLI を使用して、リソース・アダプターをサーバーにデプロイします。
2. 以下の項目を <EAP_HOME>/domain/configuration/domain.xml file の <resource-adapters> セクションに追加して、IVT アプリケーションに必要な JMS リソースを作成します。

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
    <connection-definition
         class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
         jndi-name="java:jboss/jms/ivt/IVTCF"
         enabled="true"
         use-java-context="true"
         pool-name="IVTCF">
      <config-property name="port">
          1414
      </config-property>
      <config-property name="hostName">
          localhost
      </config-property>
      <config-property name="channel">
          SYSTEM.DEF.SVRCONN
      </config-property>
      <config-property name="transportType">
          CLIENT
      </config-property>
      <config-property name="queueManager">
          ExampleQM
      </config-property>
    </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
```

```
            TEST.QUEUE
        </config-property>
      </admin-object>
    </admin-objects>
  </resource-adapter>
```

3. 次の情報をアプリケーション・サーバーの始動パラメーターに追加します。

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. アプリケーション・サーバーを停止してから再始動します。
5. JBoss 管理コンソールまたは管理 CLI を使用して IVT アプリケーションをデプロイします。
6. IVT アプリケーションを実行します。

   詳しくは、787 ページの『WebSphere MQ リソース・アダプターのインストール検査テスト・プログラム』を参照してください。 JBoss の場合、デフォルトの URL は `http://localhost:8080/WMQ_IVT/`です。

   注 : IVT アプリケーションの実行に必要な JMS リソースに使用される JNDI 名は次のとおりです。

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   上記の指定 URL を使用して IVT アプリケーションを起動するには、これらのリソースの JNDI 名をそれぞれのフィールドに入力してから、「**IVT の実行**」をクリックしてアプリケーションを実行します。

WebSphere MQ classes for JMS に付属するスクリプト

WebSphere MQ classes for JMS の使用時に実行する必要がある共通タスクを支援するために、多数のスクリプトが提供されています。

798 ページの表 102 では、すべてのスクリプトとその用法をリストしています。 これらのスクリプトは、WebSphere MQ classes for JMS インストール・ディレクトリーの `bin` サブディレクトリーにあります。

表 *102. WebSphere MQ classes for JMS* に付属するスクリプト

ユーティリティー	**以下を使用してください。**
Cleanup[1]	このスクリプトは前のリリースとの互換性のために維持されていますが、実行する機能はありません。 サブスクリプション情報の手動クリーンアップは必要なくなりました
DefaultConfiguration	Windows 以外のプラットフォーム上でデフォルト構成アプリケーションを実行します。
formatLog[1]	このスクリプトは前のリリースとの互換性のために維持されていますが、実行する機能はありません。 ログ出力は、読み取り可能なテキストで作成されるようになりました。
IVTRun[1] IVTSetup[1] IVTTidy[1]	point-to-point インストール検査テストで使用されます (779 ページの『WebSphere MQ classes for JMS の Point-to-Point インストール検査テスト』を参照)。
JMSAdmin[1]	WebSphere MQ JMS 管理ツールを実行します (938 ページの『IBM WebSphere MQ classes for JMS 管理ツールの起動』を参照)。
JMSAdmin.config	WebSphere MQ JMS 管理ツールの構成ファイル (939 ページの『JMS 管理ツールの構成』を参照)。
PSIVTRun[1]	パブリッシュ/サブスクライブのインストール検査テスト・プログラムを実行します (783 ページの『WebSphere MQ classes for JMS のパブリッシュ/サブスクライブ・インストール検査テスト』を参照)。

表 *102. WebSphere MQ classes for JMS* に付属するスクリプト *(続き)*	
ユーティリティー	**以下を使用してください。**
PSReportDump.class	このクラスは以前のリリースとの互換性のために維持されていますが、実行する機能はありません。
setjmsenv	728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』で説明されているように、 UNIX and Linux システム上の 32 ビット Java 仮想マシン (JVM) で WebSphere MQ classes for JMS アプリケーションを実行するための環境変数を設定します。
setjmsenv64	WebSphere MQ classes for JMS アプリケーションを、UNIX and Linux システム上の 64 ビット JVM で実行するように環境変数を設定します (728 ページの『IBM WebSphere MQ classes for JMS で使用される環境変数』を参照)。
注 : 1. Windows では、ファイル名の拡張子は .bat です。	

OSGi のサポート

OSGi は、バンドルの形によるアプリケーションのデプロイメントをサポートするフレームワークを提供します。 9 つの OSGi バンドルが、IBM WebSphere MQ classes for JMS の一部として提供されます。

OSGi は、汎用で、安全で、管理された Java フレームワークを提供します。これは、バンドルの形態で付属するアプリケーションのデプロイメントをサポートします。 OSGi 対応デバイスはバンドルをダウンロードしてインストールし、それらが不要になったら削除することができます。 フレームワークはバンドルのインストールおよび更新を動的かつ拡張が容易な仕方で管理します。

IBM WebSphere MQ classes for JMS には、以下の OSGi バンドルが組み込まれています。

com.ibm.msg.client.osgi.jms<version number>.jar
IBM WebSphere MQ classes for JMS 内のコードの共通レイヤー。 WebSphere MQ classes for JMS の階層化アーキテクチャーの詳細については、802 ページの『階層化アーキテクチャー』を参照してください。

com.ibm.msg.client.osgi.jms.prereq_<version number>.jar
共通レイヤーの前提条件 Java アーカイブ (JAR) ファイル。

com.ibm.msg.client.osgi.commonservices.j2se_<version number>.jar
Java プラットフォーム Standard Edition (Java SE) アプリケーションの共通サービス。

com.ibm.msg.client.osgi.nls_<version number>.jar
共通レイヤーのメッセージ。

com.ibm.msg.client.osgi.wmq_<version number>.jar
IBM WebSphere MQ classes for JMS の IBM WebSphere MQ メッセージング・プロバイダー。 IBM WebSphere MQ classes for JMS の階層化アーキテクチャーの形式については、802 ページの『階層化アーキテクチャー』を参照してください。

com.ibm.msg.client.osgi.wmq.prereq_<version number>.jar
IBM WebSphere MQ メッセージング・プロバイダーの前提条件 JAR ファイル。

com.ibm.msg.client.osgi.wmq.nls_<version number>.jar
IBM WebSphere MQ メッセージング・プロバイダーのメッセージ。

com.ibm.mq.osgi.directip_< バージョン番号> .jar
IBM WebSphere MQ メッセージング・プロバイダーによるブローカーへのリアルタイム接続の作成を可能にする JAR ファイル。

ここで、< version number> は、インストールされている WebSphere MQ のバージョン番号です。

バンドルは、 WebSphere MQ インストール済み環境の `java/lib/OSGi` サブディレクトリー、または Windows の `java\lib\OSGi` フォルダーにインストールされます。

バンドル com.ibm.mq.osgi.java < version number> .jar (これも WebSphere MQ インストール済み環境の `java/lib/OSGi` サブディレクトリー、または Windows 上の `java\lib\OSGi` フォルダーにインストールされます) は、WebSphere MQ classes for Java の一部です。 このバンドルは、既に WebSphere MQ classes for JMS がロードされている OSGi ランタイム環境にロードしてはなりません。

WebSphere MQ classes for JMS 用の OSGi バンドルは、OSGi リリース 4 仕様に合わせて記述されています。 OSGi リリース 3 環境では機能しません。

OSGi ランタイム環境が必要な DLL ファイルまたは共用ライブラリーを検出できるように、システム・パスまたはライブラリー・パスを正しく設定する必要があります。

IBM WebSphere MQ classes for JMS 用の OSGi バンドルを使用する場合、一時トピックは機能しません。 さらに、複数のクラス・ローダー環境 (OSGi など) でクラスをロードする際に固有の問題があるため、Java で作成されたチャネル出口クラスはサポートされません。 ユーザー・バンドルは IBM WebSphere MQ classes for JMS バンドルを認識できますが、IBM WebSphere MQ classes for JMS バンドルはユーザー・バンドルを認識できません。 結果として、IBM WebSphere MQ classes for JMS バンドル内で使用されるクラス・ローダーは、ユーザー・バンドル内のチャネル出口クラスをロードできません。

OSGi の詳細については、OSGi Alliance Web サイトをご覧ください。

IBM WebSphere MQ classes for JMS に関する問題の解決

インストール検査プログラムを実行し、トレースおよびログ機能を使用して、問題を調べることができます。

プログラムが正常に完了しない場合は、779 ページの『WebSphere MQ classes for JMS の Point-to-Point インストール検査テスト』および 783 ページの『WebSphere MQ classes for JMS のパブリッシュ/サブスクライブ・インストール検査テスト』の説明に従ってインストール検査プログラムの 1 つを実行し、診断メッセージに示されるアドバイスに従ってください。

ロギングおよび *IBM WebSphere MQ classes for JMS*

デフォルトでは、ログ出力は `mqjms.log` ファイルに送られます。 これは、特定のファイルまたはディレクトリーにリダイレクトできます。

IBM WebSphere MQ classes for JMS ログ機能は、プログラミング上のエラーではなく、特に構成上のエラーを示す深刻な問題を報告するために用意されています。 デフォルトでは、ログ出力は JVM 作業ディレクトリーの `mqjms.log` ファイルに送られます。

ログ出力は、プロパティー com.ibm.msg.client.commonservices.log.outputName を設定することにより、別のファイルにリダイレクトすることができます。 このプロパティーの値としては、以下のものが可能です。

- 単一のパス名。
- パス名のコンマ区切りリスト (すべてのデータがすべてのファイルにログ記録されます)。

それぞれのパス名については、以下のものが可能です。

- 絶対パスまたは相対パス。
- `stderr` または `System.err` (標準エラー出力ストリームを表します)。
- `sttdout` または `System.out` (標準出力ストリームを表します)。

プロパティーの値がディレクトリーを示す場合、ログ出力はそのディレクトリーの `mqjms.log` に書き込まれます。 プロパティーの値が特定のファイルを示す場合、ログ出力はそのファイルに書き込まれます。

このプロパティーは、IBM WebSphere MQ classes for JMS 構成ファイルで設定することもでき、**java** コマンドでシステム・プロパティーとして設定することもできます。 以下の例では、プロパティーはシステム・プロパティーとして設定されており、特定のファイルを示しています。

```
java -Djava.library.path=library_path
     -Dcom.ibm.msg.client.commonservices.log.outputName=/mydir/mylog.txt
      MyAppClass
```

ここで、*library_path* は IBM WebSphere MQ classes for JMS ライブラリーが入っているディレクトリーへのパスです (730 ページの『Java Native Interface (JNI) ライブラリーの構成』を参照)。

プロパティー com.ibm.msg.client.commonservices.log.status を OFF に設定すると、ログ出力を無効にすることができます。 このプロパティーのデフォルト値は ON です。

`System.err` および `System.out` の値を設定すると、ログ出力を `System.err` ストリームおよび `System.out` ストリームに送信できます。

プログラマー向けの WebSphere MQ classes for JMS の紹介

WebSphere MQ classes for JMS は、WebSphere MQ に付属する JMS プロバイダーです。 WebSphere MQ classes for JMS は、javax.jms パッケージで定義されたインターフェースの実装に加えて、JMS API への 2 セットの拡張機能を提供します。 Java Platform, Standard Edition (Java SE) アプリケーションと Java Platform, Enterprise Edition (Java EE) アプリケーションの両方で、WebSphere MQ classes for JMS を使用できます。

JMS の仕様では、アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットが定義されています。 最新バージョンの仕様は、バージョン 1.1 です。 javax.jms パッケージは JMS インターフェースの詳細を指定し、JMS プロバイダーはそれらのインターフェースを特定のメッセージング製品用に実装します。 WebSphere MQ classes for JMS は、WebSphere MQ 用の JMS インターフェースを実装する JMS プロバイダーです。

JMS アプリケーション内のロジックの流れは、ConnectionFactory および Destination オブジェクトで始まります。 アプリケーションは ConnectionFactory オブジェクトを使用して Connection オブジェクトを作成します。このオブジェクトは、メッセージング・サーバーへのアプリケーションのアクティブな接続を表します。 アプリケーションは Connection オブジェクトを使用して Session オブジェクトを作成します。このオブジェクトは、メッセージを作成およびコンシュームするための単一スレッド・コンテキストです。さらに、アプリケーションは Session オブジェクトおよび Destination オブジェクトを使用して MessageProducer オブジェクトを作成できます。アプリケーションはこのオブジェクトを使用して、メッセージを指定された宛先に送信します。 宛先はメッセージング・システム内のキューまたはトピックのいずれかであり、Destination オブジェクトによってカプセル化されます。 また、アプリケーションは Session オブジェクトおよび Destination オブジェクトを使用して MessageConsumer オブジェクトを作成できます。アプリケーションはこのオブジェクトを使用して、指定された宛先に送信されているメッセージを受信します。

JMS の仕様では、ConnectionFactory オブジェクトと Destination オブジェクトが管理対象オブジェクトであることが想定されます。 管理者は、中央リポジトリーで管理対象オブジェクトを作成および保守します。JMS アプリケーションは、Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。 管理オブジェクトのリポジトリーは、単純なファイルの場合もあれば、Lightweight Directory Access Protocol (LDAP) ディレクトリーの場合もあります。

WebSphere MQ classes for JMS は、管理対象オブジェクトの使用をサポートします。 アプリケーションは、WebSphere MQ 固有の情報をアプリケーション自体にハードコーディングしなくても、WebSphere MQ classes for JMS によって公開された WebSphereMQ のすべての機能を使用できます。 この調整により、アプリケーションは基盤となる WebSphere MQ 構成からある程度の独立性を保持することができます。 この独立を達成するために、アプリケーションは JNDI を使用して、管理オブジェクトとして保存される接続ファクトリーと宛先を取得でき、javax.jms パッケージで定義されるインターフェースのみを使用して、メッセージング操作を実行できます。 管理者は WebSphere MQ JMS 管理ツールまたは IBM WebSphere MQ エクスプローラーを使用して、中央リポジトリーの管理対象オブジェクトを作成および保守できます。 ただし、アプリケーション・サーバーは通常、管理オブジェクト用の独自のリポジトリーと、オブジェクトの作成と維持のための独自のツールを提供します。 したがって、 Java EE アプリケーションは JNDI を使用して、アプリケーション・サーバー・リポジトリーまたは中央リポジトリーから管理対象オブジェクトを取り出すことができます。

また、WebSphere MQ classes for JMS は JMS API への拡張機能も提供しています。 WebSphere MQ classes for JMS の以前のリリースには、MQConnectionFactory、MQQueue、および MQTopic オブジェクトで実装されている拡張機能が含まれています。 これらのオブジェクトには WebSphere MQ に固有のプロパティーやメソッドがあります。 オブジェクトは管理対象オブジェクトにすることができます。あるいは、アプリケーションはオブジェクトを実行時に動的に作成することができます。 WebSphere MQ classes for JMS の本リリースはこれらの拡張機能を保守しており、これらの拡張機能を使用するアプリケーション

は変更なしで引き続き使用することができます。 これらの拡張機能は *WebSphere MQ JMS* 拡張機能 として知られています。 この資料のセットでは、実行時にアプリケーションによって動的に作成されるオブジェクトは、管理オブジェクトと見なされない ことに注意してください。

WebSphere MQ JMS 拡張機能に加えて、WebSphere MQ classes for JMS の本リリースでは、より汎用的な JMS API への拡張機能のセットを提供しています。 これらの拡張機能は *IBM JMS* 拡張機能 として知られており、次のような幅広い目的があります。

- IBM JMS プロバイダー全体により高いレベルの整合性を持たせること
- 2 つの IBM メッセージング・システム間のブリッジ・アプリケーションをより簡単に作成できるようにすること
- 1 つの IBM JMS プロバイダーから別のプロバイダーにアプリケーションをより簡単に移植できるようにすること

それらの拡張機能は主に、接続ファクトリーおよび宛先を実行時に動的に作成および構成することに重点が置かれていますが、メッセージングと直接的には関係のない機能 (例えば問題判別のための機能) も提供します。

javax.jms インターフェースまたはいずれかの JMS 拡張機能セットを使用して作成された接続ファクトリー、キュー、またはトピック・オブジェクトは、これらの API のいずれかを使用して指定できます。つまり、どのインターフェースにもキャストできます。 最高レベルでアプリケーションの移植性を維持するには、要件に適した最も汎用的な API を使用してください。

Java SE と Java EE の両方のアプリケーションで、 WebSphere MQ classes for JMS を使用できます。 Java EE プラットフォームでは、 WebSphere MQ classes for JMS は、アプリケーションのコンポーネントと WebSphere MQ キュー・マネージャーの間の以下の 2 つのタイプの通信をサポートします。

アウトバウンド通信
JMS API を直接使用して、アプリケーション・コンポーネントはキュー・マネージャーへの接続を作成し、それからメッセージを送受信します。

例えば、アプリケーション・コンポーネントには、アプリケーション・クライアント、サーブレット、JavaServer ページ (JSP)、エンタープライズ Java Bean (EJB)、またはメッセージ駆動型 Bean (MDB) などがあります。 このタイプの通信では、アプリケーション・サーバー・コンテナーは、接続のプールやスレッド管理など、メッセージング操作をサポートする低レベルの機能のみを提供します。

インバウンド通信
宛先に届くメッセージは MDB に送信され、MDB はメッセージを処理します。

Java EE アプリケーションは、MDB を使用してメッセージを非同期に処理します。 MDB は JMS メッセージ・リスナーとしての役割を果たし、onMessage() メソッドによって実装されます。このメソッドはメッセージの処理方法を定義します。 MDB は、アプリケーション・サーバーの EJB コンテナーに実装されます。 MDB が構成される正確な方法は、使用しているアプリケーション・サーバーに依存しますが、構成情報で、接続先のキュー・マネージャー、キュー・マネージャーへの接続方法、メッセージをモニターする宛先、および MDB のトランザクションの動作を指定する必要があります。 この情報は EJB コンテナーによって使用されます。 MDB の選択基準を満たすメッセージが指定された宛先に届くと、EJB コンテナーは WebSphere MQ classes for JMS を使用してキュー・マネージャーからメッセージを取り出し、それから onMessage() メソッドを呼び出してメッセージを MDB に配信します。

IBM WebSphere MQ classes for JMS のアーキテクチャー

IBM WebSphere MQ バージョン 7.0、およびそれ以降のリリースで提供される IBM WebSphere MQ classes for JMS には、以前のリリースと比べて多数の機能拡張が含まれています。 これらの機能拡張のあるものは IBM WebSphere MQ classes for JMS の実装を変更した結果であり、またあるものは基礎となる IBM WebSphere MQ 機能への変更を IBM WebSphere MQ classes for JMS が活用した結果です。

以下のセクションでは、主な機能強化について要約します。

階層化アーキテクチャー

WebSphere MQ の以前のリリースでは、WebSphere MQ classes for JMS の実装は完全に WebSphere MQ に固有のものでした。 メッセージング・システムを提供するその他の IBM 製品にも JMS プロバイダーが

組み込まれていましたが、それらの JMS プロバイダーには WebSphere MQ classes for JMS の実装との共通点がほとんどまたはまったくありません。

WebSphere MQ V7.0 から、WebSphere MQ classes for JMS が階層化アーキテクチャーを持つようになりました。 コードの最上部レイヤーは、どの IBM JMS プロバイダーでも使用できる共通レイヤーです。 アプリケーションが JMS メソッドを呼び出すと、メッセージング・システムに固有でない呼び出しの処理はすべて共通レイヤーによって実行されます。この共通レイヤーは呼び出しへの一貫した応答も行います。 メッセージング・システムに固有の呼び出しの処理は、より下の層に委任されます。 803 ページの図 125 は、階層化アーキテクチャーを示します。

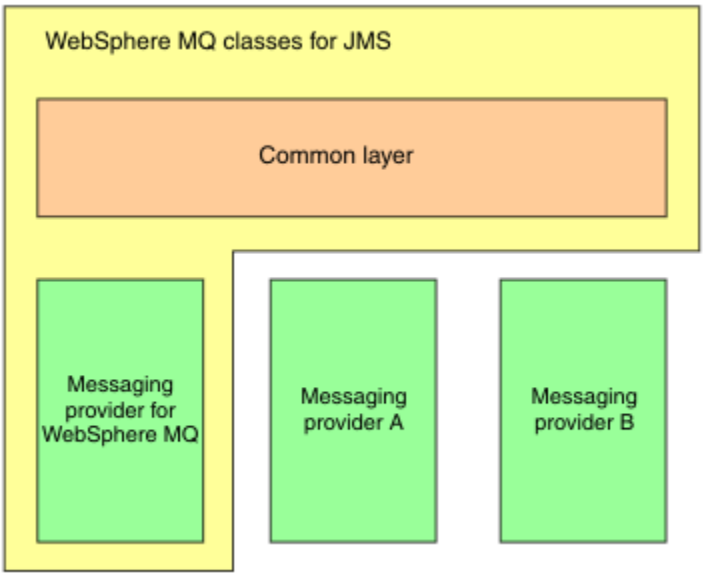


図 *125. IBM JMS* プロバイダーのための階層化アーキテクチャー

階層化アーキテクチャーに移行することには、以下のような目的があります。

- さまざまな IBM JMS プロバイダーの動作の一貫性を向上させること
- 2 つの IBM メッセージング・システム間のブリッジ・アプリケーションをより簡単に作成できるようにすること
- 1 つの IBM JMS プロバイダーから別のプロバイダーにアプリケーションをより簡単に移植できるようにすること

こうした WebSphere MQ classes for JMS の実装により、JMS API への新規の拡張機能のセットも導入されます。 こうした拡張機能は、*IBM JMS* 拡張機能として知られています。 これらの拡張機能が主に焦点を当てているのは、接続ファクトリーおよび宛先を実行時に動的に作成および構成することです。

IBM JMS 拡張機能を使用するアプリケーションは、JmsFactoryFactory オブジェクトを作成し、選択したメッセージング・システムを識別する定数をパラメーターとして指定して始動します。 アプリケーションは JmsFactoryFactory オブジェクトを使用して、選択したメッセージング・システム用に正しく特殊化されたクラスを持つ接続ファクトリーおよび宛先を作成します。

それから、アプリケーションはプロパティーを設定して接続ファクトリーおよび宛先を構成できます。 IBM JMS 拡張機能は、プロパティーを設定するための一連のメソッドを提供します。 これらのメソッドは、どのメッセージング・システムからも独立しています。 それぞれのデータ・タイプには独自の set メソッドがあり、それぞれのプロパティーは名前で識別されます。この名前は WMQConstants クラスの静的最終メンバーとして定義されます。 アプリケーションがこれらのメソッドのいずれかを呼び出す際に、呼び出し時のパラメーターの 1 つがプロパティーの名前で、他方のパラメーターはプロパティーの値です。

例えば、WebSphere MQ がメッセージング・システムである場合、接続ファクトリーのプロパティーの 1 つは接続先のキュー・マネージャーの名前です。 IBM JMS 拡張機能を使用して、アプリケーションは以下のメソッドを呼び出すことにより、キュー・マネージャーの名前を JUPITER に設定します。

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

それに対して、アプリケーションは以下のメソッドを呼び出すことにより、同じ機能を実行できます。

```
MQConnectionFactory myCF;
...
myCF.setQueueManager("JUPITER");
```

このメソッドは WebSphere MQ JMS 拡張機能であり、メッセージング・システムとして WebSphere MQ に固有のものです。 そのため、このメソッドを使用すると、アプリケーションは別の IBM JMS プロバイダーへの移植の容易性が低下する可能性があります。

WebSphere MQ classes for JMS と WebSphere MQ classes for Java との間の関係

WebSphere MQ のバージョン 7.0 より前のリリースでは、WebSphere MQ classes for JMS は、WebSphere MQ classes for Java の上にコードのレイヤーとしてほぼ完全に実装されていました。 この調整により、アプリケーション開発者の間でいくらかの混乱が生じました。なぜなら、MQEnvironment クラスでフィールドを設定したり、メソッドを呼び出すと、WebSphere MQ classes for JMS を使用して作成されるコードの実行時の動作に好ましくない影響および予期しない影響を与えることがあるからです。 さらに、WebSphere MQ classes for JMS の実装では、JMS API が WebSphere MQ classes for Java の上に自然に適合しない領域にいくつかの制約があり、これらの制約によってランタイム・パフォーマンスに関するいくつかの問題が生じています。

WebSphere MQ V7.0 以降、WebSphere MQ classes for JMS の実装は、WebSphere MQ classes for Java に依存しなくなりました。 WebSphere MQ classes for Java および WebSphere MQ classes for JMS は、MQI への共通の Java インターフェースを使用するピアになりました。 この調整によりパフォーマンスの最適化のための有効範囲が広がって、MQEnvironment クラスでフィールドを設定したり、メソッドを呼び出しても、WebSphere MQ classes for JMS を使用して作成されるコードの実行時の動作に影響を与えなくなります。804 ページの図 126 WebSphere MQ classes for JMS と WebSphere MQ classes for Java (WebSphere MQ の旧リリースおよび WebSphere MQ V7.0 以降のリリース) との関係を示しています。

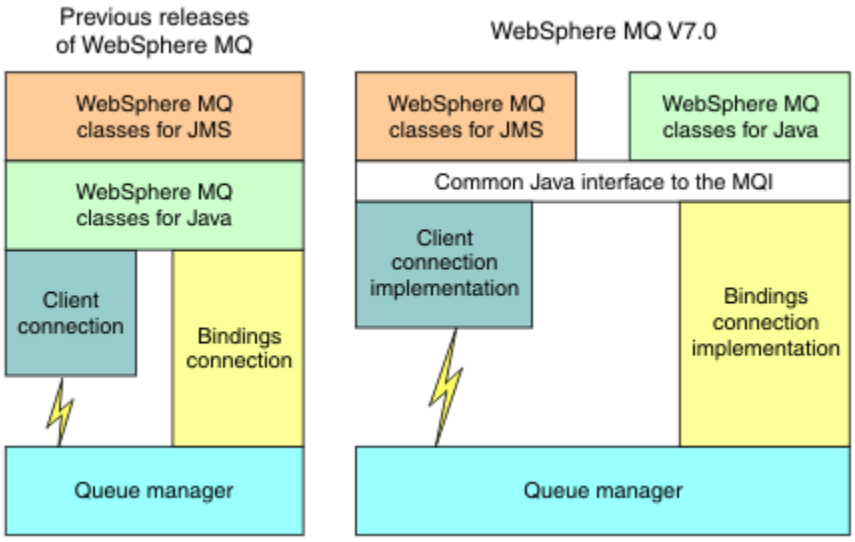


図 *126. WebSphere MQ classes for JMS と WebSphere MQ classes for Java との間の関係*

以前のリリースとの互換性を維持するために、Java で作成されたチャネル出口クラスは、チャネル出口クラスが WebSphere MQ classes for JMS から呼び出される場合でも、 WebSphere MQ classes for Java インターフェースを使用できます。 ただし、 WebSphere MQ classes for Java インターフェースを使用することは、アプリケーションが引き続き WebSphere MQ classes for Java JAR ファイル com.ibm.mq.jar に依存

していることを意味します。 com.ibm.mq.jar をクラス・パスで使用しない場合、代わりに com.ibm.mq.exits パッケージで新規のインターフェースのセットを使用できます。

現在、JMS 管理対象オブジェクトを WebSphere MQ エクスプローラーで作成し、構成できます。

パブリッシュ/サブスクライブ・メッセージング

WebSphere MQ V7.0 以降のリリースでは、パブリッシュ/サブスクライブ機能が組み込まれています。 この機能が、WebSphere MQ V6.0 で提供されていた WebSphere MQ パブリッシュ/サブスクライブの代わりとなります。

WebSphere MQ classes for JMS アプリケーションは組み込みパブリッシュ/サブスクライブ機能を使用できます。また、WebSphere MQ をトランスポートとするパブリッシュ/サブスクライブ・メッセージング用にこの機能を WebSphere Event Broker または WebSphere Message Broker の代わりに使用できます。 新機能を使用するように WebSphere MQ classes for JMS を構成することは、WebSphere パブリッシュ/サブスクライブ、WebSphere Event Broker、または WebSphere Message Broker を使用するように WebSphere MQ classes for JMS を構成するよりも単純です。 管理者およびアプリケーション開発者はパブリケーション・キュー、サブスクライバー・キュー、サブスクリプション・ストア、およびサブスクライバー・クリーンアップを管理する必要がなくなります。 さらに、ConnectionFactory および Topic オブジェクトには比較的少数のプロパティーがあります。

また、組み込みパブリッシュ/サブスクライブ機能では、保存パブリケーションや、アプリケーションのサブスクライブ先のトピック範囲を指定するために 2 つのワイルドカード方式から選択するといった追加機能も提供します。

アプリケーションは、パブリッシュ/サブスクライブ・メッセージング用に、WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続を依然として使用できます。 このサポートは変更されていません。

WebSphere MQ パブリッシュ/サブスクライブを使用するアプリケーションは、接続先のキュー・マネージャーがアップグレードされるときに変更を加えずに、組み込みパブリッシュ/サブスクライブ機能を使用できます。 アプリケーションによって設定されるものの、組み込みパブリッシュ/サブスクライブ機能では必要とされないプロパティーは無視されます。

WebSphere MQ メッセージング・プロバイダー

WebSphere MQ メッセージング・プロバイダーには次の 2 つの操作モードがあります。

- *WebSphere MQ* メッセージング・プロバイダー通常モード
- *WebSphere MQ* メッセージング・プロバイダー移行モード

WebSphere MQ メッセージング・プロバイダー通常モードは、WebSphere MQ バージョン 7.0 以降のリリースのキュー・マネージャーの全機能を使用して JMS を実装します。 このモードは、WebSphere MQ キュー・マネージャーに接続するためにのみ使用され、WebSphere MQ Version 7.0 以降のリリースのキュー・マネージャーにクライアント・モードとバインディング・モードのいずれかで接続できます。 このモードは、WebSphere MQ バージョン 7.0 以降のリリースの新機能を使用するために最適化されています。

WebSphere MQ メッセージング・プロバイダー移行モードは WebSphere MQ バージョン 6.0 機能に基づいており、WebSphere MQ バージョン 6.0 キュー・マネージャーで使用可能だった機能のみを使用して JMS を実装します。 WebSphere MQ メッセージング・プロバイダー移行モードを使用すると、WebSphere MQ バージョン 7.0 以降のリリースのキュー・マネージャーに接続できますが、 バージョン 7.0 での最適化は利用できません。 このモードにより、次のキュー・マネージャーのバージョンのいずれかへの接続が可能になります。

1. バインディング・モードまたはクライアント・モードの WebSphere MQ バージョン 7.0 以降のキュー・マネージャー。ただし、このモードでは WebSphere MQ バージョン 6.0 のキュー・マネージャーで使用可能であった機能のみを使用します。
2. クライアント・モードの WebSphere MQ バージョン 6.0 以前のキュー・マネージャー。

WebSphere Event Broker または WebSphere Message Broker に WebSphere MQ Enterprise Transport を使用して接続する場合は、WebSphere MQ メッセージング・プロバイダー移行モードを使用してください。 WebSphere MQ Real-Time Transport を使用する場合、接続ファクトリー・オブジェクトでプロパティーを

明示的に選択しているため、WebSphere MQ メッセージング・プロバイダー移行モードが自動的に選択されます。 WebSphere MQ Enterprise Transport を使用した WebSphere Event Broker または WebSphere Message Broker への接続は、『WebSphere MQ メッセージング・プロバイダー・モードの選択の規則』で説明されているモード選択規則に従います。

非同期メッセージの消費

WebSphere MQ V7.0 以降のリリースは、非同期メッセージ消費をサポートします。 アプリケーションは宛先に対するコールバック機能を登録できます。 適切なメッセージが宛先に送信されると、WebSphere MQ はその機能を呼び出し、メッセージをパラメーターとして渡します。 次いで、その機能はメッセージを非同期で処理します。 WebSphere MQ の以前のリリースでは、このフィーチャーは WebSphere MQ classes for JMS を使用する場合にのみ使用可能でした。

WebSphere MQ classes for JMS は、WebSphere MQ V7.0 以降のリリースのこの新機能を活用できるように変更されました。 現在、JMS メッセージ・リスナーの実装は WebSphere MQ にこれまで以上に適しており、WebSphere MQ classes for JMS は宛先をポーリングして、適切なメッセージが宛先に送信されたかどうかを確認する必要がなくなりました。 その結果として、JMS メッセージ・リスナーのパフォーマンスは改善されました。特に、アプリケーションが 1 つのセッションで複数のメッセージ・リスナーを使用して複数の宛先をモニターしている場合はそう言えます。 メッセージ・スループットは向上し、メッセージが宛先に届いた後でそのメッセージをメッセージ・リスナーに送達するために要する時間は減少します。

メッセージ・ドリブン Bean (MDB) には同様のパフォーマンスの向上が見られます。 さらに、WebSphere MQ 機能への別の機能拡張により、同じ宛先からのメッセージを消費している複数の MDB ではメッセージ上の競合の減少が見られています。

メッセージ選択

メッセージ ID または相関 ID によるメッセージ選択を除き、WebSphere MQ のバージョン 7.0 より前のリリースでは、メッセージ選択はすべて WebSphere MQ classes for JMS によって行われていました。 WebSphere MQ V7.0 以降のリリースでは、すべてのメッセージ選択がキュー・マネージャーによって実行されます。

結果として、メッセージ選択を使用するメッセージを消費するアプリケーションで、メッセージ・スループットが向上します。 クライアント・モードで接続するアプリケーションのほうが、パフォーマンスの向上率が高まります。これは、選択基準を満たすメッセージのみがネットワーク経由で移送され、WebSphere MQ classes for JMS はアプリケーションに送信するメッセージのみを処理するためです。

通信接続の共用

WebSphere MQ の以前のリリースでは、WebSphere MQ クライアント・アプリケーションが同じ MQI チャネルを使用して複数回キュー・マネージャーに接続された場合、MQI チャネルの各インスタンスは別個の TCP 接続を必要としました。 WebSphere MQ V7.0 以降のリリースでは、同じ MQI チャネルを使用するキュー・マネージャーへの各接続は単一の TCP 接続を共用できます。 この調整は、特に SSL を使用している場合には、必要なネットワーク・リソースが少なくなり、キュー・マネージャーへの複数の接続を作成するために要する合計時間が減少することを意味します。なぜなら、TCP 接続の開始時に SSL ハンドシェークが一回限り発生するからです。

WebSphere MQ classes for JMS はこの機能拡張を活用します。 クライアント・モードでキュー・マネージャーに接続するアプリケーションの場合、WebSphere MQ classes for JMS は、ConnectionFactory オブジェクトのプロパティーとして指定された名前を持つ MQI チャネルを使用して、キュー・マネージャーへの複数の接続を作成することがあります。 現在、キュー・マネージャーへのこうした接続はそれぞれ単一の TCP 接続を共用できます。

クライアント接続での先読み

アプリケーションがクライアント接続を使用して宛先から非永続メッセージを消費する場合、WebSphere MQ classes for JMS がバッファーを使用して対象となるメッセージをアプリケーションに送達する前に、それらを保管するようにその宛先を構成できます。 この最適化は先読みと呼ばれ、receive() メソッドを呼び出すことによってメッセージを同期に消費するアプリケーションや、メッセージ・リスナーおよび MDB

を呼び出すことによってメッセージを非同期に消費するアプリケーションが使用できます。 先読みは、迅速に消費する必要がある多数のメッセージを持つ宛先に特に有効です。

先読みは永続メッセージには適用されません。なぜなら、永続メッセージがバッファーに読み込まれた場合、キュー・マネージャーは障害発生後のメッセージをリカバリーできなくなるからです。 ただし、永続メッセージと非永続メッセージが混ざった宛先からメッセージを消費するアプリケーションは、これまでどおり先読みを使用できます。 メッセージの順序は保存されますが、先読みの実行時の利点は非永続メッセージにのみ適用されます。

先読みを使用するかどうか決定する場合、以下の点を考慮してください。

- アプリケーションが先読み用に構成されている宛先からメッセージを消費しており、何らかの理由でアプリケーションが終了する場合、現在バッファーに保管されている非永続メッセージは廃棄されます。
- 以下の条件がすべて当てはまる場合、セッション中にキューに送信されたメッセージは、送信順序どおりには受信されないことがあります。
  - アプリケーションが同じセッションの 2 つのメッセージ・コンシューマーを使用して、メッセージをキューから消費する。
  - それぞれのメッセージ・コンシューマーがキューに異なる Destination オブジェクトを使用する。
  - Destination オブジェクトのいずれかまたは両方が先読み用に構成されている。

メッセージの送信

アプリケーションがメッセージを宛先に送信する際に、アプリケーションが send() を呼び出すと WebSphere MQ classes for JMS がメッセージをキュー・マネージャーに転送し、キュー・マネージャーがメッセージを支障なく受け取ったかどうかを判別せずに制御をアプリケーションに戻すように、宛先を構成できます。 WebSphere MQ classes for JMS は、非持続メッセージと、トランザクション化されたセッションで送信された持続メッセージに関してのみ、この方法で機能することができます。

トランザクション化されたセッションで送信される持続メッセージの場合、アプリケーションは最終的に、キュー・マネージャーが commit () を呼び出すときにメッセージを安全に受信したかどうかを判別します。トランザクション化されていないセッションで送信されたメッセージの場合、 ConnectionFactory オブジェクトの SENDCHECKCOUNT プロパティーは、キュー・マネージャーがメッセージを安全に受信したことを WebSphere MQ classes for JMS が検査する前に送信するメッセージの数を指定します。

この最適化は、クライアント・モードでキュー・マネージャーに接続するアプリケーションに最も利点があり、一連のメッセージを連続して迅速に送信する必要がありますが、送信された各メッセージについてキュー・マネージャーから即時のフィードバックは必要としません。

チャネル出口

C または C++ で作成されたチャネル出口プログラムが WebSphere MQ classes for JMS から呼び出される場合、Websphere MQ MQI クライアントから呼び出されたときと同じように動作します。 Java で作成されたチャネル出口クラスのパフォーマンスが向上し、WebSphere MQ classes for Java のインターフェースを使用する代わりに、 com.ibm.mq.exits パッケージの新しいインターフェース・セットを使用してチャネル出口クラスを作成できるようになりました。

メッセージ・プロパティー

JMS メッセージは、ヘッダー・フィールドのセット、プロパティーのセット、およびアプリケーション・データを含む本文で構成されます。 最低でも、WebSphere MQ メッセージはメッセージ記述子とアプリケーション・データで構成されます。

WebSphere MQ classes for JMS アプリケーションが JMS メッセージを送信すると、WebSphere MQ classes for JMS はその JMS メッセージを WebSphere MQ メッセージにマップします。 JMS ヘッダー・フィールドおよびプロパティーの一部はメッセージ記述子内のフィールドにマップされ、一部は MQRFH2 ヘッダーと呼ばれる追加の WebSphere MQ ヘッダー内のフィールドにマップされます。 WebSphere MQ classes for JMS アプリケーションが JMS メッセージを受信すると、WebSphere MQ classes for JMS は逆マッピングを実行します。

そのため、MQI を使用して WebSphere MQ classes for JMS アプリケーションからメッセージを受信するアプリケーションは、MQRFH2 ヘッダーを処理できなければなりません。 アプリケーションが MQRFH2 ヘッダーを処理できない場合、MQRFH2 ヘッダーを WebSphere MQ メッセージに含めないように WebSphere MQ classes for JMS に通知するよう、Destination オブジェクトの TARGCLIENT プロパティーを設定できます。 ただし、MQRFH2 ヘッダーを除外することにより、一部の JMS ヘッダー・フィールドおよびプロパティーに保持された情報は失われます。

同様に、MQI を使用して WebSphere MQ classes for JMS アプリケーションにメッセージを送信するアプリケーションは、各メッセージに MQRFH2 ヘッダーを含める必要があります。 MQRFH2 ヘッダーが含まれていない場合、WebSphere MQ classes for JMS は、メッセージ記述子内のフィールドから派生できる JMS ヘッダー・フィールドおよびプロパティーのみ設定できます。

WebSphere MQ V7.0 は、MQI を使用して WebSphere MQ classes for JMS アプリケーションとの間でメッセージを送受信するアプリケーションのために追加サポートを提供しています。

アプリケーションが MQGET を呼び出して WebSphere MQ classes for JMS アプリケーションからメッセージを受信する場合、アプリケーションは以下の方法のいずれかでメッセージを受信することを選択できます。

1. メッセージはメッセージ記述子、JMS ヘッダー・フィールドおよびプロパティーから派生したデータを含む MQRFH2 ヘッダー、およびアプリケーション・データとともに送達されます。
2. メッセージはメッセージ記述子、アプリケーション・データ、およびメッセージ・プロパティーのセットとともに送達されます。

オプション 2 の場合、それぞれのメッセージ・プロパティーは、もともと WebSphere MQ classes for JMS によって MQRFH2 ヘッダー内のフィールドにマップされた JMS ヘッダー・フィールドまたはプロパティーを表します。 MQGET 呼び出しの後、アプリケーションは MQINQMP 呼び出しを使用して、メッセージ・プロパティーの値を取得します。 オプション 1 ではなくオプション 2 を使用してメッセージを受信すると、アプリケーション論理が以下のように単純化されます。

- アプリケーションは、MQRFH2 ヘッダーの変数部分を解析する必要がありません。この部分には XML のような形式でエンコードされた JMS ヘッダー・フィールドとプロパティー・データが含まれています。
- アプリケーションは MQRFH2 ヘッダーの変数部分の文字データを変換する必要がありません。

同様に、アプリケーションが MQPUT を呼び出してメッセージを WebSphere MQ classes for JMS アプリケーションに送信する前に、アプリケーションは、MQRFH2 ヘッダーを構成する代わりに、MQSETMP 呼び出しを使用してメッセージ・プロパティーの値を設定できます。

保守容易度

WebSphere MQ classes for JMS には、保守容易性に関連する以下のような多くの改善が含まれています。

- トレース。

  WebSphere MQ classes for JMS には、アプリケーションがトレースを制御するために使用できるクラスが含まれています。 アプリケーションはトレースを開始および停止し、トレース内の必要な詳細レベルを指定し、トレース出力をさまざまな方法でカスタマイズできます。

- ロギング。

  WebSphere MQ classes for JMS はログ・ファイルを保守します。ログ・ファイルには修正が必要なエラーに関するメッセージが含まれています。 メッセージはプレーン・テキストで書き込まれます。 WebSphere MQ classes for JMS には、アプリケーションがログ・ファイルのロケーションと最大サイズを指定するために使用できるクラスが含まれています。

- First Failure Support Technology (FFST)。

  重大な障害が発生した場合、WebSphere MQ classes for JMS は FFST レポートを FDC ファイルに生成します。 FFST レポートには、IBM サービスが問題を迅速に診断するために使用できる情報が含まれています。

- バージョン情報。

  WebSphere MQ classes for JMS には、アプリケーションが WebSphere MQ classes for JMS のバージョンを照会するために使用できるクラスが含まれています。

- 例外メッセージ。

  例外メッセージは拡張されており、エラーの原因およびエラーの修正に必要なアクションの詳細を提供します。
- アプリケーション・サーバー。

  WebSphere MQ classes for JMS の保守容易性機能と WebSphere Application Server の保守容易性機能の統合が改善されました。

MQC が MQConstants に置き換えられました

IBM WebSphere MQ バージョン 7.0 では、新しいパッケージ com.ibm.mq.constants が提供されるようになりました。 このパッケージには、多数のインターフェースを実装するクラス MQConstants が含まれています。 MQConstants には、MQC インターフェースに含まれていたすべての定数と、多数の新しい定数の定義が含まれます。 このパッケージ内のインターフェースは、IBM WebSphere MQ で使用される定数のヘッダー・ファイルの名前にほぼ従っています。

例えば、インターフェース CMQC には、定数 MQOO_INPUT_SHARED が含まれています。このインターフェースと定数は、ヘッダー・ファイル `cmqc.h` と定数 MQOO_INPUT_SHARED に対応します。

com.ibm.mq.constants は、 IBM WebSphere MQ classes for Java と IBM WebSphere MQ classes for JMS の両方で使用できます。

MQC は現在も存在し、ここには以前含まれていた定数が含まれています。 ただし、新しいアプリケーションでは、com.ibm.mq.constants パッケージを使用してください。

WebSphere MQ classes for JMS アプリケーションの作成

JMS モデルを概説した後、このトピックでは WebSphere MQ classes for JMS アプリケーションの作成方法について詳しく説明します。

JMS モデル

JMS モデルは、Java アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットを定義します。 WebSphere MQ classes for JMS は JMS プロバイダーとして、JMS オブジェクトと WebSphere MQ 概念の関連を定義します。 JMS の仕様では、特定の JMS オブジェクトが管理対象オブジェクトであることが想定されます。

JMS 仕様および javax.jms パッケージは、Java アプリケーションがメッセージング操作を実行するために使用できるインターフェースのセットを定義します。 以下のリストに、主な JMS インターフェースを要約します。

Destination
Destination は、アプリケーションがメッセージを送信する場所、またはアプリケーションが受信するメッセージの送信元、あるいはその両方です。

ConnectionFactory
ConnectionFactory オブジェクトは、接続の構成プロパティーのセットをカプセル化します。 アプリケーションは、接続ファクトリーを使用して接続を作成します。

接続
Connection オブジェクトは、メッセージング・サーバーに対するアプリケーションのアクティブな接続をカプセル化します。 アプリケーションは、接続を使用してセッションを作成します。

Session
Session は、メッセージを送受信する単一スレッド化されたコンテキストです。 アプリケーションは、Session を使用してメッセージ、メッセージ・プロデューサー、およびメッセージ・コンシューマーを作成します。 セッションは、トランザクション化しても、トランザクション化しなくてもかまいません。

メッセージ
Message オブジェクトは、アプリケーションが送信または受信するメッセージをカプセル化します。

MessageProducer
アプリケーションがメッセージ・プロデューサーを使用して宛先にメッセージを送信します。

MessageConsumer
アプリケーションがメッセージ・コンシューマーを使用して宛先に送信されたメッセージを受信します。

810 ページの図 127 は、これらのオブジェクトとその関係を示します。

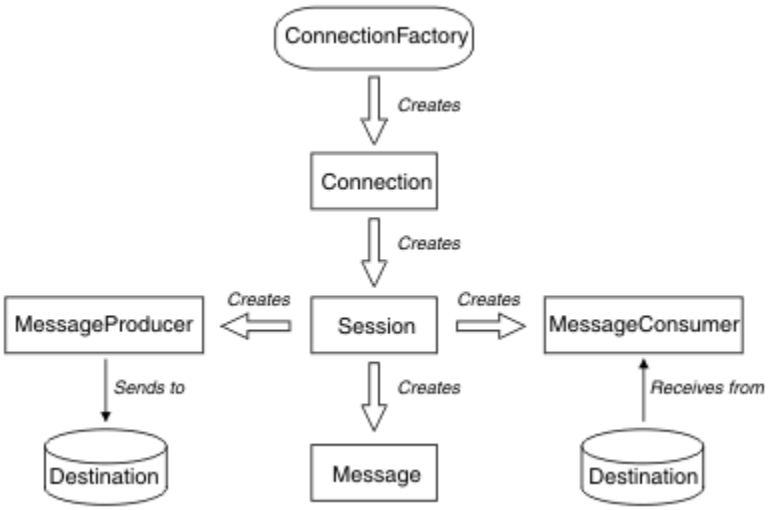


図 *127. JMS* オブジェクトとその関係

Destination、ConnectionFactory、または Connection オブジェクトは、マルチスレッド・アプリケーションの異なるスレッドによって並行して使用できますが、Session、MessageProducer、または MessageConsumer オブジェクトは、異なるスレッドによって並行して使用することはできません。 Session、MessageProducer、または MessageConsumer オブジェクトが並行して使用されないようにする最も簡単な方法は、スレッドごとに別の Session オブジェクトを作成することです。

JMS は 2 つのメッセージングのスタイルをサポートします。

- Point-to-Point メッセージング
- パブリッシュ/サブスクライブ・メッセージング

メッセージングのこれらのスタイルは、メッセージ・ドメイン とも呼ばれ、1 つのアプリケーションで両方のスタイルのメッセージングを結合することができます。 Point-to-Point ドメインの場合、宛先はキューであり、パブリッシュ/サブスクライブ・ドメインの場合、宛先はトピックです。

JMS 1.1 より前のバージョンの JMS では、Point-to-Point ドメイン用のプログラミングに 1 つのインターフェースおよびメソッドのセット、パブリッシュ/サブスクライブ・ドメイン用のプログラミングに別のセットを使用していました。 2 つのセットは似ていますが、別のものです。 JMS 1.1 では、両方のメッセージング・ドメインをサポートする共通のインターフェースおよびメソッドのセットを使うことができます。 共通のインターフェースは、メッセージ・ドメインごとにドメイン非依存のビューを提供します。 810 ページの表 103 は JMS ドメイン独立インターフェースおよび関連するドメイン固有インターフェースをリストしています。

表 *103. JMS* ドメイン独立インターフェースとドメイン固有インターフェース

ドメイン非依存インターフェース	**Point-to-Point ドメインのドメイン固有インターフェース**	**パブリッシュ/サブスクライブ・ドメインのドメイン固有インターフェース**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
接続	QueueConnection	TopicConnection
Destination	キュー	トピック
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher

表 *103. JMS* ドメイン独立インターフェースとドメイン固有インターフェース *(*続き*)*		
ドメイン非依存インターフェース	**Point-to-Point ドメインのドメイン固有インターフェース**	**パブリッシュ/サブスクライブ・ドメインのドメイン固有インターフェース**
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 1.1 では、ドメイン固有インターフェースすべてが保持されるため、既存のアプリケーションはこれらのインターフェースを使い続けることができます。 ただし、新規のアプリケーションでは、ドメイン独立インターフェースを使うことを考慮してください。

WebSphere MQ classes for JMS では、JMS オブジェクトは WebSphere MQ の概念と次のように関連します。

- Connection オブジェクトには、接続の作成に使用された接続ファクトリーのプロパティーから派生したプロパティーがあります。 これらのプロパティーは、アプリケーションがキュー・マネージャーに接続する方法を制御します。 これらのプロパティーの例はキュー・マネージャーの名前であり、クライアント・モードのキュー・マネージャーに接続するアプリケーションでは、キュー・マネージャーが動作しているシステムのホスト名または IP アドレスです。
- Session オブジェクトは WebSphere MQ 接続ハンドルをカプセル化します。これはセッションのトランザクションの有効範囲を定義します。
- MessageProducer オブジェクトと MessageConsumer オブジェクトはそれぞれ、WebSphere MQ オブジェクト・ハンドルをカプセル化します。

WebSphere MQ classes for JMS の使用時には WebSphere MQ の通常の全ルールが適用されます。 特に、アプリケーションはリモート・キューにメッセージを送信できますが、アプリケーションが接続しているキュー・マネージャーによって所有されるキューからしかメッセージを受信できないことに注意してください。

JMS の仕様では、ConnectionFactory オブジェクトと Destination オブジェクトが管理対象オブジェクトであることが想定されます。 管理者は、中央リポジトリーで管理対象オブジェクトを作成および保守します。 JMS アプリケーションは、Java Naming and Directory Interface (JNDI) を使用してこれらのオブジェクトを取得します。

WebSphere MQ classes for JMS では、Destination インターフェースの実装は Queue および Topic の抽象スーパークラスであるため、Destination のインスタンスは Queue オブジェクトか Topic オブジェクトのいずれかになります。 ドメイン非依存インターフェースは、キューまたはトピックを宛先として処理します。 MessageProducer オブジェクトまたは MessageConsumer オブジェクトのメッセージ・ドメインは、宛先がキューまたはトピックのいずれであるかによって決定されます。

それで、WebSphere MQ classes for JMS では、以下のタイプのオブジェクトが管理対象オブジェクトになります。

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- キュー
- トピック
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

JMS メッセージ

JMS メッセージは、ヘッダー、プロパティー、および本文で構成されます。 JMS では、5 つのタイプのメッセージ本体を定義します。

JMS メッセージは、以下の部分から構成されています。

ヘッダー
すべてのメッセージは、同じヘッダー・フィールドのセットをサポートします。 ヘッダー・フィールドには、メッセージを識別し、経路指定するために、クライアントとプロバイダーの両方によって使用される値が含まれています。

プロパティー
各メッセージには、アプリケーション定義のプロパティー値をサポートする組み込み機能が含まれています。 プロパティーは、アプリケーション定義のメッセージをフィルターに掛けるための効果的なメカニズムを提供します。

Body
JMS では、現在使用中の大部分のメッセージング・スタイルを包含するいくつかのタイプのメッセージ本体を定義します。

JMS では、以下の 5 つのタイプのメッセージ本体を定義します。

ストリーム
Java プリミティブ値のストリーム。 ストリームが満たされると、順次に読み取られます。

マップ
名前と値のペアのセット。ここで、名前はストリングであり、値は Java プリミティブ型です。 エントリーには、順次アクセスか、または名前によるランダムでのアクセスを実行できます。 エントリーの順序は定義されていません。

Text
java.lang.String を含むメッセージ。

オブジェクト
シリアライズ可能な Java オブジェクトが含まれているメッセージ

Bytes
非解釈バイトのストリーム。 このメッセージ・タイプは、既存のメッセージ形式と一致するように本体を事実上エンコードするためのものです。

JMSCorrelationID ヘッダー・フィールドは、1 つのメッセージを別のメッセージとリンクするために使用されます。 JMSCorrelationID ヘッダー・フィールドは、通常、応答メッセージを要求メッセージとリンクします。 JMSCorrelationID は、プロバイダー特定のメッセージ ID、アプリケーション固有のストリング、またはプロバイダー・ネイティブの byte[] 値を保持できます。

JMS のメッセージ・セレクター

メッセージには、アプリケーション定義のプロパティー値を含めることができます。 アプリケーションは、メッセージ・セレクターを使用して JMS プロバイダーでメッセージをフィルター操作できます。

メッセージには、アプリケーション定義のプロパティー値をサポートするための組み込み機能が含まれています。 これは事実上、メッセージにアプリケーション固有のヘッダー・フィールドを追加するためのメカニズムを提供します。 アプリケーションはプロパティーを指定することにより、メッセージ・セレクターおよびアプリケーション固有の基準を使用して、自身の代わりに JMS プロバイダーでメッセージを選択することも、メッセージをフィルター操作することもできます。 アプリケーション定義のプロパティーは、以下の規則に従わなければなりません。

- プロパティー名は、メッセージ・セレクター ID に関する規則に従わなければなりません。
- プロパティー値には、boolean、byte、short、int、long、float、double、および String を指定できます。
- JMSX および JMS_ name プレフィックスは予約されています。

プロパティー値は、メッセージを送信する前に設定されます。 クライアントがメッセージを受信すると、メッセージ・プロパティーは、読み取り専用になります。 この時点で、クライアントがプロパティーを設定しようとする場合、MessageNotWriteableException がスローされます。 clearProperties が呼び出される場合、プロパティーは読み取りにも書き込みにもなることができます。

プロパティー値は、メッセージの本体に値を複製する場合もあります。 JMS では、プロパティーに何が作成されるかについてポリシーを定義しません。 ただし、アプリケーション開発者は、JMS プロバイダーが通常、メッセージのプロパティー内のデータよりも効率的にメッセージ本体内のデータを処理できる点に留意してください。 最適なパフォーマンスを得るために、アプリケーションは、メッセージのヘッダーを

カスタマイズする必要が生じた場合にのみ、メッセージ・プロパティーを使用してください。 これを行う主な理由は、カスタマイズされたメッセージ選択をサポートするためです。

JMS メッセージ・セレクターにより、クライアントは、メッセージ・ヘッダーを使用して、関心のあるメッセージを指定できます。 ヘッダーがセレクターと一致するメッセージのみが送達されます。

メッセージ・セレクターは、メッセージ本体の値を参照することはできません。

メッセージのヘッダー・フィールドおよびプロパティー値がセレクター内の対応する ID に置換される際に、セレクターが true に評価された場合、メッセージ・セレクターはメッセージと一致します。

メッセージ・セレクターはストリングであり、その構文は SQL92 条件式構文のサブセットに基づいています。 メッセージ・セレクターが評価される順序は、優先順位内で左から右です。 括弧を使用してこの順序を変更できます。 定義済みのセレクター・リテラルおよび演算子名は、大文字でここに書き込まれます。 ただし、これらには大/小文字の区別がありません。

セレクターには、以下のものを含めることができます。

- リテラル
  - ストリング・リテラルは、引用符で囲まれます。 二重引用符は引用符を表します。 例は、'literal' および 'literal''s' です。 Java ストリング・リテラルと同様に、これらは Unicode 文字エンコードを使用します。
  - 正確な数字リテラルは、57、-957、+62 など、小数点なしの数値です。 Java long の範囲内の数値がサポートされます。
  - 近似数値リテラルは、7E3 または -57.9E2 などの浮動小数における数値、または 7.、-95.7、または +6.2 などの小数部を持つ数値です。 Java double の範囲内の数値がサポートされます。
  - ブール・リテラルの TRUE および FALSE。
- ID:
  - ID は、Java 文字と Java 数字の無制限の長さのシーケンスで、最初の文字は Java 文字でなければなりません。 英字は、メソッド Character.isJavaLetter が true を戻す任意の文字です。 これには _ と $が含まれます。 文字または数字は、メソッド Character.isJavaLetterOrDigit が true を返す任意の文字です。
  - ID は、名前 NULL、TRUE、または FALSE となることはできません。
  - ID は、NOT、AND、OR、BETWEEN、LIKE、IN、および IS となることはできません。
  - ID は、ヘッダー・フィールド参照またはプロパティー参照のいずれかです。
  - ID には、大/小文字の区別があります。
  - メッセージ・ヘッダー・フィールド参照は、以下のものに制限されます。
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID、および JMSType 値は、ヌルの場合があります。 その場合は、ヌル値として扱われます。
  - JMSX で始まる名前は、JMS 定義のプロパティー名です。
  - JMS_で始まる名前は、プロバイダー固有のプロパティー名です。
  - JMS で始まらない名前は、アプリケーション固有のプロパティー名です。 メッセージ内に存在しないプロパティーへの参照がある場合、その値はヌルです。 存在する場合には、その値は、対応するプロパティー値です。
- 空白文字は、Java 用に定義されているものと同じです (スペース、水平タブ、用紙送り、および行終了文字)。

- 式:
  - セレクターは、条件式です。 true に評価されるセレクターは一致し、false または unknown に評価されるセレクターは一致しません。
  - 演算式は、それ自体と、算術演算、ID (その値は数値リテラルとして扱われる)、および数値リテラルから構成されています。
  - 条件式は、それ自体と、比較演算、および論理演算から構成されています。
- 標準の括弧 () (式が評価される順序を設定する) がサポートされています。
- 論理演算子 (優先順位どおりに列挙): NOT、AND、OR。
- 比較演算子: =、>、>=、<、<=、<> (等しくない)。
  - 同じタイプの値のみを比較できます。 1 つの例外は、正確な数値と近似数値の比較が有効であることです。 (必要な型変換は、Java 数値プロモーションの規則によって定義されます。) 異なるタイプを比較する試みがある場合、セレクターは常に false です。
  - ストリングとブールの比較は、= および <> に制限されます。2 つのストリングは、同じ文字シーケンスを含んでいる場合にのみ等しくなります。
- 算術演算子 (優先順位どおりに列挙):
  - 単項 +、-。
  - *、/ (乗算および除算)。
  - +、- (加算および減算)。
  - ヌル値での算術演算はサポートされていません。 ヌル値での算術演算が試行される場合、完全セレクターは常に false です。
  - 算術演算では、Java 数値プロモーションを使用する必要があります。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比較演算子:
  - Age BETWEEN 15 and 19 は、age >= 15 AND age <= 19 と同じです。
  - Age NOT BETWEEN 15 and 19 は、age < 15 OR age> 19 に相当します。
  - BETWEEN 演算の式のいずれかがヌルである場合、演算の値は false です。 NOT BETWEEN 演算の式のいずれかがヌルである場合、演算の値は true です。
- identifier [NOT] IN (string-literal1, string-literal2, ...) ID がストリング値または NULL 値を持つ比較演算子。
  - Country IN (' UK', 'US', 'France') は 'UK' の場合には true であり、'Peru' の場合には false です。 これは、式 (Country = ' UK') OR (Country = ' US') OR (Country = ' France') と同じです。
  - Country NOT IN (' UK', 'US', 'France') は 'UK' の場合には false であり、'Peru' の場合には true です。 これは、式 NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France')) と同じです。
  - IN または NOT IN 演算の ID がヌルである場合、演算の値は不明です。
- ID がストリング値を持つ identifier [NOT] LIKE pattern-value [ESCAPE escape-character] 比較演算子。 pattern-value はストリング・リテラルです。ここで、_ は単一文字を表しており、% は文字シーケンス (空のシーケンスを含む) を表しています。 その他のすべての文字はそれ自体を表しています。 任意の escape-character は単一の文字ストリング・リテラルです。この文字は、pattern-value 内の _ および % の特殊な意味をエスケープするために使用されます。
  - phone LIKE '12%3' は、123 および 12993 の場合には true で、1234 の場合には false です。
  - word LIKE 'l_se' は、lose の場合には true で、loose の場合には false です。
  - underscored LIKE '¥_%' ESCAPE '¥' は _foo の場合には true で、bar の場合には false です。
  - phone NOT LIKE '12%3' は、123 および 12993 の場合には false で、1234 の場合には true です。
  - LIKE または NOT LIKE 演算の ID がヌルである場合には、演算の値は不明です。
- identifier IS NULL 比較演算子は、ヌルのヘッダー・フィールド値または欠落したプロパティー値をテストします。

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

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

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

上記でも指摘したとおり、プロパティー値はヌルである場合があります。 ヌル値を含むセレクター式の評価は、SQL 92 NULL セマンティクスによって定義されます。 以下は、これらのセマンティクスの要旨です。

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

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

SQL コメントは、サポートされていません。

JMS メッセージの WebSphere MQ メッセージへのマッピング

WebSphere MQ メッセージは、メッセージ記述子、オプションの MQRFH2 ヘッダー、および本体で構成されます。 JMS メッセージの内容は、WebSphere MQ メッセージに、その一部がマップされ、一部がコピーされます。

このトピックでは、このセクションの最初の部分で説明されている JMS メッセージ構造が WebSphere MQ メッセージにマップされる方法を説明します。 このセクションは、JMS と従来の WebSphere MQ アプリケーションとの間でメッセージを伝送したいプログラマーを対象としています。 また、2 つの JMS アプリケーション間で伝送されるメッセージを操作するユーザーも対象としています (WebSphere Message Broker の実装など)。

このセクションは、アプリケーションがブローカーとのリアルタイム接続を使用する場合には適用されません。 アプリケーションがリアルタイム接続を使用する場合は、通信はすべて TCP/IP を使用して直接行われます。WebSphere MQ キューやメッセージは使用されません。

WebSphere MQ メッセージは、以下の 3 つのコンポーネントから構成されています。

- WebSphere MQ メッセージ記述子 (MQMD)
- WebSphere MQ MQRFH2 ヘッダー
- メッセージ本体

MQRFH2 はオプションであり、出力メッセージに含まれる内容は、JMS Destination クラス内のフラグによって管理されます。 WebSphere MQ JMS 管理ツールを使用してこのフラグを設定できます。 MQRFH2 は JMS 特定の情報を伝送するため、受信宛先が JMS アプリケーションであることを送信側が認識している場合には、常にメッセージ内にその情報が含まれています。 通常、メッセージを直接、非 JMS アプリケーションに送信するときには、MQRFH2 を省略してください。 これは、そのようなアプリケーションが WebSphere MQ メッセージ内に MQRFH2 を予期していないためです。

着信メッセージに MQRFH2 ヘッダーが含まれていない場合、そのメッセージの JMSReplyTo ヘッダー・フィールドから取り出された Queue オブジェクトまたは Topic オブジェクトでは、キューまたはトピックに送信される応答メッセージにも MQRFH2 ヘッダーが含まれないようにするため、デフォルトでこのフラグが設定されます。 元のメッセージに MQRFH2 ヘッダーが含まれている場合に限り、応答メッセージに MQRFH2 ヘッダーを含める動作をオフに切り替えることができます。これを行うには、接続ファクトリーの TARGCLIENTMATCHING プロパティーを NO に設定します。

816 ページの図 128 に、JMS メッセージの構造が WebSphere MQ メッセージに変換されてから元に戻る様子を示します。

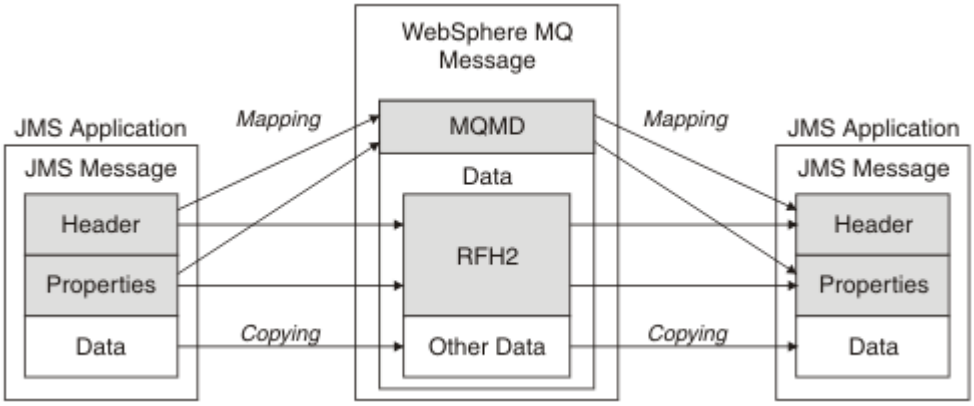


図 *128. MQRFH2* ヘッダーを使用して、メッセージが *JMS* と *WebSphere MQ* の間で変換される様子

これらの構造体は、以下の 2 つの方法で変換されます。

マッピング
MQMD が JMS フィールドと等しいフィールドを含んでいる場合、JMS フィールドは MQMD フィールドにマップされます。 追加の MQMD フィールドは、JMS プロパティーとして公開されます。これは、JMS アプリケーションが、非 JMS アプリケーションと通信する際にこれらのフィールドを取得または設定する必要があるためです。

コピー
等しい MQMD がない場合、JMS ヘッダー・フィールドまたはプロパティーが渡され、MQRFH2 内のフィールドとして変換されます。

MQRFH2 ヘッダーと *JMS*
この一連のトピックでは、MQRFH バージョン 2 ヘッダーについて説明します。これは、メッセージ内容と関連のある JMS 固有のデータを伝送します。 MQRFH2 バージョン 2 は、拡張可能なヘッダーであり、直接 JMS と関連のない追加情報も伝送できます。 ただし、このセクションでは、JMS による使用のみを扱います。すべての説明は、MQRFH2 – 規則および書式ヘッダー 2 を参照してください。

ヘッダーには 2 つの部分として、固定部分と変数部分があります。

固定部分
固定部分は、標準の WebSphere MQ ヘッダー・パターンにモデル化され、以下のフィールドから成り立っています。

StrucId (MQCHAR4)
構造体 ID

Must be MQRFH_STRUC_ID (値: "RFH ") (初期値)。

MQRFH_STRUC_ID_ARRAY (値: "R"、"F"、"H"、" ") も定義されます。

Version (MQLONG)
構造体のバージョン番号。

MQRFH_VERSION_2 (値: 2) (初期値) でなければなりません。

StrucLength (MQLONG)
NameValueData フィールドを含む、MQRFH2 の全長。

StrucLength に設定される値は、4 の倍数でなければなりません (NameValueData フィールド内のデータは、これをアーカイブするためにスペース文字で埋められる場合があります)。

Encoding (MQLONG)
データ・エンコード。

MQRFH2 の後のメッセージの部分にある数値データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) のエンコード。

CodedCharSetId (MQLONG)
コード化文字セット ID。

MQRFH2 の後のメッセージ部分にある文字データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) の表記。

Format (MQCHAR8)
フォーマット名。

MQRFH2 の後のメッセージの部分のフォーマット名。

フラグ (MQLONG)
フラグ。

MQRFH_NO_FLAGS =0。 フラグが何も設定されていません。

NameValueCCSID (MQLONG)
このヘッダーに含まれている NameValueData 文字ストリング用のコード化文字セット ID (CCSID)。 NameValueData は、ヘッダー (StrucID および Format) 内に含まれているその他の文字ストリングとは異なる文字セットでコード化される場合があります。

NameValueCCSID が 2 バイトの Unicode CCSID (1200、13488、または 17584) である場合、Unicode のバイト順は、MQRFH2 内の数値フィールドのバイト順と同じです。 (例えば、Version、StrucLength、NameValueCCSID 自体。)

表 *104. NameValueCCSID* フィールドに指定できる値

値	意味
1200	UCS2 開放型
1208	UTF8
13488	UCS2 2.0 サブセット
17584	UCS2 2.1 サブセット (ユーロ記号を含む)

変数部分
変数部分は、固定部分の後に続きます。 変数部分には、可変数の MQRFH2 フォルダーが含まれます。 それぞれのフォルダーには、可変数のエレメントまたはプロパティーが含まれます。 フォルダーは、関連のあるプロパティーをグループにまとめます。 JMS によって作成される MQRFH2 ヘッダーには、以下のフォルダーをいくつでも含めることができます。

<mcd> フォルダー

mcd には、メッセージの形式を記述するプロパティーが入ります。 例えば、メッセージ・サービス・ドメインの Msd プロパティーは、JMS メッセージを JMSTextMessage、JMSBytesMessage、JMSStreamMessage、JMSMapMessage、JMSObjectMessage、またはヌルとして識別します。

mcd フォルダーは常に、MQRFH2 が入っている JMS メッセージ内に存在します。

これは常に、 WebSphere Message Broker から送信された MQRFH2 を含むメッセージ内に存在します。 そして、メッセージのドメイン、形式、タイプ、およびメッセージ・セットを記述します。

表 *105. mcd* のプロパティー名、同義語、データ型、およびフォルダー

プロパティー同義語	プロパティー名	データ・タイプ	フォルダー
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>

表 *105. mcd* のプロパティー名、同義語、データ型、およびフォルダー *(続き)*			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

独自のプロパティーを mcd フォルダーに追加しないでください。

<jms> フォルダー

jms には、JMS ヘッダー・フィールドと、MQMD で完全には表現できない JMSX プロパティーが含まれています。 jms フォルダーは、常に JMS MQRFH2 の中に存在します。

<usr> フォルダー

usr には、メッセージに関連付けられているアプリケーション定義の JMS プロパティーが入ります。 usr フォルダーは、アプリケーションがアプリケーション定義プロパティーを設定した場合のみ存在します。

<mqext> フォルダー

mqext には、WebSphere Application Server によってのみ使用されるプロパティーが含まれます。 このフォルダーは、アプリケーションが IBM 定義のプロパティーを少なくとも 1 つ設定した場合にのみ存在します。

表 *106. mqext* のプロパティー名、同義語、データ型、およびフォルダー			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>

独自のプロパティーを mqext フォルダーに追加しないでください。

<mqps> フォルダー

mqps には、IBM WebSphere MQ パブリッシュ/サブスクライブによってのみ使用されるプロパティーが入ります。 このフォルダーは、統合されたパブリッシュ/サブスクライブ・プロパティーを少なくとも 1 つアプリケーションが設定した場合にのみ存在します。

表 *107. mqps* のプロパティー名、同義語、データ型、およびフォルダー			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData*</Sud></mqps>

表 *107. mqps* のプロパティー名、同義語、データ型、およびフォルダー *(続き)*			
プロパティー同義語	**プロパティー名**	**データ・タイプ**	**フォルダー**
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

独自のプロパティーを mqps フォルダーに追加しないでください。

819 ページの表 108 には、プロパティー名の完全なリストを示します。

表 *108. JMS* によって使用される *MQRFH2* フォルダーおよびプロパティー				
JMS フィールド名	**Java 型**	**MQRFH2 フォルダー名**	**プロパティー名**	**タイプ/値**
JMSDestination	Destination	jms	Dst	ストリング
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	ストリング	jms	Cid	ストリング
JMSReplyTo	Destination	jms	Rto	ストリング
JMSTimestamp	long	jms	Tms	i8
JMSType	ストリング	mcd	Type、Set、Fmt	ストリング
JMSXGroupID	ストリング	jms	Gid	ストリング
JMSXGroupSeq	int	jms	Seq	i4
xxx (ユーザー定義)	任意	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

NameValueLength (MQLONG)
この長さフィールドの直後にある NameValueData ストリングの長さ (バイト単位) (それ自体の長さは含まれません)。

NameValueData (MQCHARn)
単一文字ストリング。長さ (バイト単位) は、前の NameValueLength フィールドに示されています。 この文字ストリングには、プロパティーのシーケンスを保持しているフォルダーが含まれています。 それぞれのプロパティーは、名前/タイプ/値のセットで、名前がフォルダー名である XML エレメント内に含まれており、以下のとおりです。

```
 <foldername>
triplet1 triplet2 .....   tripletn </foldername>
```

終了 </foldername> タグの後には、埋め込み文字としてスペースを続けることができま。 それぞれのセットは、XML と同様の構文を使用してエンコードされます。

```
   <name dt='datatype'>value</name>
```

dt='datatype' エレメントはオプションであり、データ・タイプが事前定義されているため、多くのプロパティーでは省略されています。 これを含める場合は、 dt= タグの前に 1 つ以上のスペース文字を含める必要があります。

name
プロパティーの名前。819 ページの表 108 を参照してください。

datatype
省略後、820 ページの表 109 に示すデータ・タイプの 1 つと一致しなければなりません。

value
820 ページの表 109 の定義を使用して伝えられる値のストリング表記です。

ヌル値は、以下の構文を使用してエンコードされます。

```
<name dt='datatype' xsi:nil='true'></name>
```

xsi:nil='false'は使用しないでください。

表 *109.* プロパティーのデータ・タイプ

データ・タイプ	**定義**
ストリング	< および & を除く任意の文字シーケンス
boolean	文字 0 または 1 (0 = false, 1 = true)
bin.hex	オクテットを表す 16 進数字。
i1	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -128 から 127 (両端を含む) の範囲内になければなりません。
i2	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -32768 から 32767 (両端を含む) の範囲内になければなりません。
i4	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -2147483648 から 2147483647 (両端を含む) の範囲内になければなりません。
i8	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 -9223372036854775808 から 9223372036854775807 (両端を含む) の範囲内になければなりません。
int	オプションの符号 (小数部または指数なし) を持つ、数字 0..9 で表される数値。 i8 と同じ範囲内になければなりません。 送信側が特定の精度をプロパティーと関連付けない場合に、'i*' タイプのいずれかの代わりにこれを使用できます。

表 109. プロパティーのデータ・タイプ (続き)	
データ・タイプ	**定義**
r4	浮動小数点数、絶対値 < = 3.40282347E+38、> = 1.175E-37 (数字 0..9、オプションの符号、オプションの小数桁、オプションの指数を使用して表現)
r8	浮動小数点数、絶対値 < = 1.7976931348623E+308、> = 2.225E-307 数字 0..9、オプションの符号、オプションの小数桁、オプションの指数

ストリング値には、スペースを含めることができます。 ストリング値では以下のエスケープ・シーケンスを使用しなければなりません。

- & 文字の場合は &
- < 文字の場合は <

以下のエスケープ・シーケンスを使用できますが、必須ではありません。

- > 文字の場合は >
- ' 文字の場合は '
- " 文字の場合は "

対応する *MQMD* フィールドを持つ *JMS* フィールドおよびプロパティー
以下の表に、JMS ヘッダー・フィールド、JMS プロパティー、および JMS プロバイダー固有のプロパティーに対応する MQMD フィールドを示します。

821 ページの表 110 には、JMS ヘッダー・フィールドをリストし、821 ページの表 111 には、MQMD フィールドに直接マップされる JMS プロパティーをリストします。822 ページの表 112 プロバイダー固有のプロパティーと、それらがマップされる MQMD フィールドをリストします。

表 110. *MQMD* フィールドへの *JMS* ヘッダー・フィールドのマッピング			
JMS ヘッダー・フィールド	**Java 型**	**MQMD フィールド**	**C タイプ**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	ストリング	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	ストリング	CorrelId	MQBYTE24

表 111. *MQMD* フィールドへの *JMS* プロパティーのマッピング			
JMS プロパティー	**Java 型**	**MQMD フィールド**	**C タイプ**
JMSXUserID	ストリング	UserIdentifier	MQCHAR12
JMSXAppID	ストリング	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG

表 *111. MQMD* フィールドへの *JMS* プロパティーのマッピング *(*続き*)*

JMS プロパティー	**Java 型**	**MQMD フィールド**	**C タイプ**
JMSXGroupID	ストリング	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

表 *112. MQMD* フィールドへの *JMS* プロバイダー固有プロパティーのマッピング

JMS プロバイダー固有プロパティー	**Java 型**	**MQMD フィールド**	**C タイプ**
JMS_IBM_Report_Exception	int	レポート	MQLONG
JMS_IBM_Report_Expiration	int	レポート	MQLONG
JMS_IBM_Report_COA	int	レポート	MQLONG
JMS_IBM_Report_COD	int	レポート	MQLONG
JMS_IBM_Report_PAN	int	レポート	MQLONG
JMS_IBM_Report_NAN	int	レポート	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	レポート	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	レポート	MQLONG
JMS_IBM_Report_Discard_Msg	int	レポート	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	ストリング	Format[822 ページの『1』](#)	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	ストリング	CodedCharacterSetId[822 ページの『2』](#)	MQLONG
JMS_IBM_PutDate	ストリング	PutDate	MQCHAR8
JMS_IBM_PutTime	ストリング	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

注 :

1. JMS_IBM_Format は、メッセージ本体のフォーマットを表します。 これはメッセージの JMS_IBM_Format プロパティーのアプリケーション設定で定義できます (8 文字の長さ制限があることにご注意ください)。あるいは、JMS メッセージ・タイプに適したメッセージ本体の WebSphere MQ フォーマットである、デフォルトにすることもできます。 メッセージに RFH あるいは RFH2 セクションが含まれていない場合、JMS_IBM_Format は MQMD Format フィールドにのみマップします。 標準的なメッセージでは、メッセージ本体の直前に RFH2 の Format フィールドにマップします。
2. JMS_IBM_Character_Set プロパティーの値は数値の CodedCharacterSetId 値と同等の Java 文字セットを含むストリング値です。 MQMD フィールド CodedCharacterSetId は、JMS_IBM_Character_Set プロパティーで指定された Java 文字セットのストリングと同等のものを含む数値です。

JMS フィールドの *WebSphere MQ* フィールドへのマッピング *(*出力メッセージ*)*
以下の表に、send() または publish() 時に JMS ヘッダー・フィールドと JMS プロパティー・フィールドが MQMD および MQRFH2 フィールドにマップされる方法を示します。

823 ページの表 113 に、send() または publish() 時に JMS ヘッダー・フィールドが MQMD/RFH2 フィールドにマップされる方法を示します。823 ページの表 114 send () または publish () 時に JMS プロパティーが MQMD/RFH2 フィールドにマップされる方法を示しています。 824 ページの表 115 send () または publish () 時に JMS プロバイダー固有プロパティーが MQMD フィールドにマップされる方法を示します。

設定にメッセージ・オブジェクトというマークが付いているフィールドの場合、send() または publish() の実行の直前に、JMS メッセージ内に保持されている値が送信されます。 JMS メッセージ内の値は、この操作によって変更されることはありません。

設定に Send メソッドというマークが付いているフィールドの場合、send() または publish() の実行時に値が割り当てられます (JMS メッセージ内に保持されている値は無視されます)。 JMS メッセージ内の値は、使用されている値を示すように更新されます。

受信のみというマークが付いているフィールドは、伝送されず、send() または publish() によってメッセージ内で変更されることはありません。

表 *113.* 出力メッセージ・フィールドのマッピング

JMS ヘッダー・フィールド名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSDestination		MQRFH2	Send メソッド
JMSDeliveryMode	Persistence	MQRFH2	Send メソッド
JMSExpiration	Expiry	MQRFH2	Send メソッド
JMSPriority	Priority	MQRFH2	Send メソッド
JMSMessageID	MsgID		Send メソッド
JMSTimestamp	PutDate/PutTime		Send メソッド
JMSCorrelationID	CorrelId	MQRFH2	メッセージ・オブジェクト
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	メッセージ・オブジェクト
JMSType		MQRFH2	メッセージ・オブジェクト
JMSRedelivered			受信のみ

注 :

1. MQMD フィールド CodedCharacterSetId は、JMS_IBM_Character_Set プロパティーで指定された Java 文字セットのストリングと同等のものを含む数値です。

表 *114.* 出力メッセージ *JMS* プロパティーのマッピング

JMS プロパティー名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSXUserID	UserIdentifier		Send メソッド
JMSXAppID	PutApplName		Send メソッド
JMSXDeliveryCount			受信のみ
JMSXGroupID	GroupId	MQRFH2	メッセージ・オブジェクト

表 *114.* 出力メッセージ *JMS* プロパティーのマッピング *(*続き*)*

JMS プロパティー名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMSXGroupSeq	MsgSeqNumber	MQRFH2	メッセージ・オブジェクト

表 *115.* 出力メッセージの *JMS* プロバイダー固有プロパティーのマッピング

JMS プロバイダー固有プロパティー名	**伝送に使用される MQMD フィールド**	**ヘッダー**	**設定**
JMS_IBM_Report_Exception	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Expiration	レポート		メッセージ・オブジェクト
JMS_IBM_Report_COA/COD	レポート		メッセージ・オブジェクト
JMS_IBM_Report_NAN/PAN	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Pass_Msg_ID	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Pass_Correl_ID	レポート		メッセージ・オブジェクト
JMS_IBM_Report_Discard_Msg	レポート		メッセージ・オブジェクト
JMS_IBM_MsgType	MsgType		メッセージ・オブジェクト
JMS_IBM_Feedback	Feedback		メッセージ・オブジェクト
JMS_IBM_Format	Format		メッセージ・オブジェクト
JMS_IBM_PutApplType	PutApplType		Send メソッド
JMS_IBM_Encoding	Encoding		メッセージ・オブジェクト
JMS_IBM_Character_Set	CodedCharacterSetId		メッセージ・オブジェクト
JMS_IBM_PutDate	PutDate		Send メソッド
JMS_IBM_PutTime	PutTime		Send メソッド
JMS_IBM_Last_Msg_In_Group	MsgFlags		メッセージ・オブジェクト

send() または *publish()* の際の *JMS* ヘッダー・フィールドのマッピング
これらの注記は、send() または publish() の際の JMS フィールドのマッピングに関連しています。

JMSDestination → MQRFH2
これは、宛先オブジェクトの顕著な特性をシリアライズするストリングとして保管されるため、受信 JMS は等価の宛先オブジェクトを再構成することができます。 MQRFH2 フィールドは URI としてエ

ンコードされます (URI 表記の詳細については、884 ページの『Uniform Resource Identifier (URI)』を参照してください)。

JMSReplyTo → MQMD.ReplyToQ、ReplyToQMgr、MQRFH2
キュー名は MQMD.ReplyToQ フィールドにコピーされ、キュー・マネージャー名は ReplyToQMgr フィールドにコピーされます。 宛先拡張情報 (宛先オブジェクト内に保管されているその他の役立つ情報) は、MQRFH2 フィールドにコピーされます。 MQRFH2 フィールドは URI としてエンコードされます (URI 表記の詳細については、884 ページの『Uniform Resource Identifier (URI)』を参照してください)。

JMSDeliveryMode → MQMD.Persistence
Destination Object が指定変更しない限り、JMSDeliveryMode 値は send() または publish() メソッド、または MessageProducer によって設定されます。 JMSDeliveryMode 値は、以下のように MQMD.Persistence フィールドにマップされます。

- JMS 値 PERSISTENT は MQPER_PERSISTENT と等価です。
- JMS 値 NON_PERSISTENT は MQPER_NOT_PERSISTENT と等価です。

MQQueue 持続プロパティーが WMQConstants.WMQ_PER_QDEF に設定されないと、送達モード値も MQRFH2 でエンコードされます。

JMSExpiration ←/→ MQMD.Expiry、MQRFH2
JMSExpiration は、有効期限が切れる時刻 (現行時間と存続時間の合計) を保管しますが、MQMD は存続時間を保管します。 また、JMSExpiration はミリ秒単位ですが、MQMD.Expiry は 1/10 秒単位です。

- send() メソッドが無制限の存続時間を設定する場合、MQMD.Expiry は MQEI_UNLIMITED に設定され、JMSExpiration は MQRFH2 にエンコードされません。
- send() メソッドが 214748364.7 秒 (約 7 年) より短い存続時間を設定する場合、存続時間は MQMD.Expiry 内に保管され、有効期限が切れる時刻 (ミリ秒) は MQRFH2 内で i8 値としてエンコードされます。
- send() メソッドが 214748364.7 秒より長い存続時間を設定する場合、MQMD.Expiry は MQEI_UNLIMITED に設定されます。 正確な有効期限が切れる時刻 (ミリ秒単位) は、MQRFH2 内で i8 値としてエンコードされます。

JMSPriority → MQMD.Priority
JMSPriority 値 (0 から 9) を MQMD 優先順位値 (0 から 9) に直接マップします。 JMSPriority が非デフォルト値に設定される場合、優先順位も MQRFH2 内でエンコードされます。

JMSMessageID ← MQMD.MessageID
JMS から送信されるすべてのメッセージは、WebSphere MQ によって割り当てられる固有のメッセージ ID を持っています。 割り当てられた値は、MQPUT 呼び出し後に MQMD.MessageId フィールド内に戻され、JMSMessageID フィールド内のアプリケーションに渡されます。 WebSphere MQ messageId は 24 バイトのバイナリー値ですが、JMSMessageID はストリングです。 JMSMessageID は、文字 ID: という接頭部を持つ、48 個の 16 進文字のシーケンスに変換されたバイナリーの messageId 値から構成されています。 JMS は、メッセージ ID の生成を使用不可に設定できるヒントを提供します。 このヒントは無視され、すべての場合に固有 ID が割り当てられます。 send() が上書きされる前に JMSMessageId フィールドに任意の値が設定されます。

MQMD.MessageID これは、900 ページの『WebSphere MQ classes for JMS アプリケーションからのメッセージ記述子の読み取りおよび書き込み』で説明されているいずれかの WebSphere MQ JMS 拡張機能を使用して行うことができます。

JMSTimestamp → MQRFH2
送信の際に、JMSTimestamp フィールドは JVM クロックに従って設定されます。 この値は、MQRFH2 に設定されます。 send() の前に JMSTimestamp フィールドに設定される値は上書きされます。 JMS_IBM_PutDate および JMS_IBM_PutTime プロパティーも参照してください。

JMSType → MQRFH2
このストリングは MQRFH2 mcd.Type フィールドに設定されます。 URI フォーマットの場合、mcd.Set および mcd.Fmt フィールドにも影響する可能性があります。 927 ページの『WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続の使用』も参照してください。

JMSCorrelationID → MQMD.CorrelId、MQRFH2
JMSCorrelationID は、以下のうちの 1 つを保持することができます。

プロバイダー固有のメッセージ ID
これは、前に送信または受信されたメッセージのメッセージ ID であり、*ID:* という接頭部が付いた 48 桁の小文字の 16 進数のストリングである必要があります。接頭部は削除され、残りの文字はバイナリーに変換され、その後 MQMD.CorrelId フィールドに設定されます。 correlid 値は、MQRFH2 でエンコードされます。

プロバイダー・ネイティブの byte[] 値
値は、MQMD.CorrelId フィールドにコピーされ、必要であれば、24 バイトまでヌルで埋め込まれるか、または切り捨てられます。 correlid 値は、MQRFH2 でエンコードされます。

アプリケーション固有のストリング
値は、MQRFH2 にコピーされます。 ストリングの最初の 24 バイト (UTF8 形式) が、MQMD.CorrelID に書き込まれます。

JMS プロパティー・フィールドのマッピング
以下の注記は、JMS プロパティー・フィールドの WebSphere MQ メッセージへのマッピングを示しています。

JMSXUserID ← MQMD UserIdentifier
JMSXUserID は、送信呼び出しからの戻り時に設定されます。

JMSXAppID ← MQMD PutApplName
JSMXAppID は、送信呼び出しの戻り時に設定されます。

JMSXGroupID → MQRFH2 (Point-to-Point)
Point-to-Point メッセージの場合、JMSXGroupID は、MQMD GroupID フィールドにコピーされます。 JMSXGroupID が接頭部 ID: で始まる場合には、バイナリーに変換されます。 そうでない場合、UTF8 ストリングとしてエンコードされます。 必要であれば、値は 24 バイトの長さまで埋め込まれるか、または切り捨てられます。 MQMF_MSG_IN_GROUP フラグが立てられます。

JMSXGroupID → MQRFH2 (パブリッシュ/サブスクライブ)
パブリッシュ/サブスクライブ・メッセージの場合、JMSXGroupID はストリングとして MQRFH2 にコピーされます。

JMSXGroupSeq MQMD MsgSeqNumber (point-to-point)
Point-to-Point メッセージの場合、JMSXGroupSeq は、MQMD MsgSeqNumber フィールドにコピーされます。 MQMF_MSG_IN_GROUP フラグが立てられます。

JMSXGroupSeq MQMD MsgSeqNumber (パブリッシュ/サブスクライブ)
パブリッシュ/サブスクライブ・メッセージの場合、JMSXGroupSeq は i4 として MQRFH2 にコピーされます。

JMS プロバイダー特定のフィールドのマッピング
以下の注記は、JMS プロバイダー固有フィールドの IBM WebSphere MQ メッセージへのマッピングを示しています。

JMS_IBM_Report_<name> → MQMD Report
JMS アプリケーションは、以下の JMS_IBM_Report_XXX プロパティーを使用して、MQMD Report オプションを設定できます。 単一の MQMD は、いくつかの JMS_IBM_Report_XXX プロパティーにマップされます。 アプリケーションは、標準 IBM WebSphere MQ MQRO_ 定数 (com.ibm.mq.MQC に組み込まれている) にこれらのプロパティーの値を設定する必要があります。 したがって、例えば、十分なデータを持つ COD を要求するには、アプリケーションは、JMS_IBM_Report_COD を CMQC.MQRO_COD_WITH_FULL_DATA の値に設定しなければなりません。

JMS_IBM_Report_Exception

MQRO_EXCEPTION または
MQRO_EXCEPTION_WITH_DATA または
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION または
MQRO_EXPIRATION_WITH_DATA または
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA または
MQRO_COA_WITH_DATA または
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD または
MQRO_COD_WITH_DATA または
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

JMS_IBM_MsgType → MQMD MsgType
値は、直接 MQMD MsgType にマップします。 アプリケーションが JMS_IBM_MsgType の明示的な値を設定していない場合には、デフォルト値が使用されます。 このデフォルト値は、以下のように決定されます。

- JMSReplyTo が IBM WebSphere MQ キュー宛先に設定される場合、MSGType は値 MQMT_REQUEST に設定されます。
- JMSReplyTo が設定されていないか、または IBM WebSphere MQ キュー宛先以外の宛先に設定されている場合には、MsgType は値 MQMT_DATAGRAM に設定されます。

JMS_IBM_Feedback → MQMD Feedback
値は、直接 MQMD Feedback にマップします。

JMS_IBM_Format → MQMD Format
値は、直接 MQMD Format にマップします。

JMS_IBM_Encoding → MQMD Encoding
設定される場合、このプロパティーは Destination Queue または Topic の数値エンコードを指定変更します。

JMS_IBM_Character_Set → MQMD CodedCharacterSetId
設定される場合、このプロパティーは、Destination Queue または Topic のコード化文字セット・プロパティーを指定変更します。

JMS_IBM_PutDate ← MQMD PutDate
このプロパティーの値は、送信の際に、MQMD 内の PutDate フィールドから直接設定されます。 send の前に JMS_IBM_PutDate プロパティーに設定される値は上書きされます。 このフィールドは 8 文字のストリングで、IBM WebSphere MQ 日付形式が YYYYMMDD です。 このプロパティーは、JMS_IBM_PutTime プロパティーと使用して、キュー・マネージャーに従ってメッセージが書き込まれた時間を判別することができます。

JMS_IBM_PutTime ← MQMD PutTime
このプロパティーの値は、送信の際に、MQMD 内の PutTime フィールドから直接設定されます。 send の前に JMS_IBM_PutTime プロパティーに設定される値は上書きされます。 このフィールドは 8 文字

のストリングで、IBM WebSphere MQ 時刻形式が HHMMSSTH です。 このプロパティーは、JMS_IBM_PutDate プロパティーと使用して、キュー・マネージャーに従ってメッセージが書き込まれた時間を判別することができます。

JMS_IBM_Last_Msg_In_Group → MQMD MsgFlags
Point-to-Point メッセージングの場合、このブール値は MQMD MsgFlags フィールドの MQMF_LAST_MSG_IN_GROUP フラグにマップします。 通常、JMSXGroupID および JMSXGroupSeq プロパティーと共に使用され、レガシー IBM WebSphere MQ アプリケーションに対してこのメッセージがグループ内で最後であることを示します。 パブリッシュ/サブスクライブ・メッセージングの場合には、このプロパティーは無視されます。

WebSphere MQ フィールドの *JMS* フィールドへのマッピング *(*着信メッセージ*)*
以下の表に、get() または receive() 時に JMS ヘッダー・フィールドとプロパティー・フィールドが MQMD および MQRFH2 フィールドにマップされる方法を示します。

828 ページの表 116 に、get() または receive() 時に JMS ヘッダー・フィールドが MQMD/MQRFH2 フィールドにマップされる方法を示します。829 ページの表 117 は、get () または receive () 時に JMS プロパティー・フィールドが MQMD/MQRFH2 フィールドにどのようにマップされるかを示しています。 829 ページの表 118 JMS プロバイダー固有のプロパティーがどのようにマップされるかを示します。

表 *116.* 着信メッセージ *JMS* ヘッダー・フィールドのマッピング

JMS ヘッダー・フィールド名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMSDestination		jms.Dst または mqps.Top[828 ページの『1』]
JMSDeliveryMode	Persistence[828 ページの『2』]	jms.Dlv[828 ページの『2』]
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MsgID	
JMSTimestamp	PutDate[828 ページの『2』] PutTime[828 ページの『2』]	jms.Tms[828 ページの『2』]
JMSCorrelationID	CorrelId[828 ページの『2』]	jms.Cid[828 ページの『2』]
JMSReplyTo	ReplyToQ[828 ページの『2』] ReplyToQMgr[828 ページの『2』]	jms.Rto[828 ページの『2』]
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

注 :

1. jms.Dst と mqps.Top の両方が設定されている場合、jms.Dst の値が使用されます。
2. MQRFH2 または MQMD から値を取得することができるプロパティーに関して、両方が使用可能な場合には MQRFH2 の設定が使用されます。
3. JMS_IBM_Character_Set プロパティーの値は数値の CodedCharacterSetId 値と同等の Java 文字セットを含むストリング値です。

表 *117.* 着信メッセージ・プロパティーのマッピング

JMS プロパティー名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId[829 ページの『1』]	jms.Gid[829 ページの『1』]
JMSXGroupSeq	MsgSeqNumber[829 ページの『1』]	jms.Seq[829 ページの『1』]

注:

1. MQRFH2 または MQMD から値を取得することができるプロパティーに関して、両方が使用可能な場合には MQRFH2 の設定が使用されます。 MQMF_MSG_IN_GROUP または MQMF_LAST_MSG_IN_GROUP のメッセージ・フラグが設定されている場合、このプロパティーは MQMD 値からのみ設定されます。

表 *118.* 着信メッセージのプロバイダー固有 *JMS* プロパティーのマッピング

JMS プロパティー名	**MQMD フィールドの検索元**	**MQRFH2 フィールドの検索元**
JMS_IBM_Report_Exception	レポート	
JMS_IBM_Report_Expiration	レポート	
JMS_IBM_Report_COA	レポート	
JMS_IBM_Report_COD	レポート	
JMS_IBM_Report_PAN	レポート	
JMS_IBM_Report_NAN	レポート	
JMS_IBM_Report_ Pass_Msg_ID	レポート	
JMS_IBM_Report_Pass_Correl_ID	レポート	
JMS_IBM_Report_Discard_Msg	レポート	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding [829 ページの『1』]	Encoding	
JMS_IBM_Character_Set [829 ページの『1』]	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. 着信メッセージが Bytes Message である場合にのみ設定されます。

JMS アプリケーションと従来の *WebSphere MQ* アプリケーションとの間でのメッセージの交換
このトピックは、JMS アプリケーションが、MQRFH2 ヘッダーを処理できない従来の WebSphere MQ アプリケーションとメッセージを交換するときに起きる事柄を説明しています。

830 ページの図 129 は、マッピングを示しています。

アドミニストレーターは、JMS アプリケーションが従来の WebSphere MQ アプリケーションと、宛先の TARGCLIENT プロパティーを *MQ* に設定することで通信していることを示します。 これは、MQRFH2 ヘッダーが生成されないことを示しています。 これが行われない場合、受信側アプリケーションが MQRFH2 ヘッダーを扱えなければなりません。

JMS から、従来の WebSphere MQ アプリケーションで宛先となっている MQMD へのマッピングは、JMS から、JMS アプリケーションで宛先となっている MQMD へのマッピングと同じです。 WebSphere MQ classes for JMS が WebSphere MQ メッセージを受信し、MQMD *Format* フィールドが MQFMT_RFH2 以外に設定されている場合、データは非 JMS アプリケーションから受信されています。 フォーマットが MQFMT_STRING である場合、メッセージは JMS テキスト・メッセージとして受信されます。 それ以外の場合、メッセージは、JMS バイト・メッセージとして受信されます。 MQRFH2 が存在しないため、MQMD 内に伝送されるそれらの JMS プロパティーのみを復元できます。

WebSphere MQ classes for JMS が MQRFH2 ヘッダーを含まないメッセージを受信した場合、メッセージの JMSReplyTo ヘッダー・フィールドから取り出された Queue オブジェクトまたは Topic オブジェクトの TARGCLIENT プロパティーは、デフォルトで MQ に設定されます。 つまり、キューまたはトピックに送信される応答メッセージにも、MQRFH2 ヘッダーは含まれません。 元のメッセージに MQRFH2 ヘッダーが含まれている場合に限り、応答メッセージに MQRFH2 ヘッダーを含める動作をオフに切り替えることができます。 これを行うには、接続ファクトリーの TARGCLIENTMATCHING プロパティーを NO に設定します。

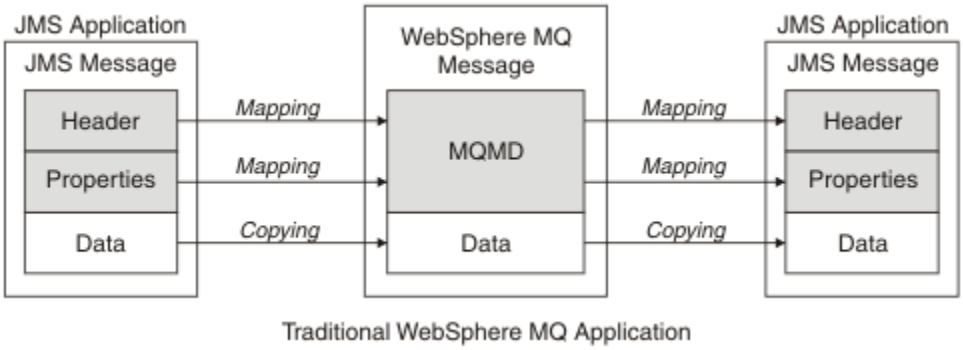


図 *129. JMS* メッセージが *WebSphere MQ* メッセージに変換される様子 *(MQRFH2* ヘッダーなし*)*

JMS メッセージ本体
このトピックには、メッセージ本体自体のエンコードに関する情報が記載されています。 エンコードは、JMS メッセージのタイプによって異なります。

ObjectMessage
ObjectMessage は、通常の方法で Java ランタイムによってシリアライズされるオブジェクトです。

TextMessage
TextMessage は、エンコードされたストリングです。 出力メッセージの場合、ストリングは、宛先オブジェクトによって提供されている文字セットにエンコードされます。 これは、デフォルトでは UTF8 エンコードになります (UTF8 エンコードは、メッセージの最初の文字で始まります。先頭に長さフィールドはありません)。 ただし、WebSphere MQ classes for JMS によってサポートされているその他の文字セットを指定できます。 こうした文字セットは、主に、非 JMS アプリケーションにメッセージを送信する際に使用されます。

文字セットが 2 バイト・セット (UTF16 を含む) である場合、宛先オブジェクトの整数エンコード仕様により、バイト順が決定されます。

着信メッセージは、メッセージ自体に指定されている文字セットおよびエンコードを使用して解釈されます。 これらの指定は、末尾の WebSphere MQ ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、末尾のヘッダーは、通常、MQRFH2 です。

BytesMessage

BytesMessage は、デフォルトでは、JMS 1.0.2 仕様および関連する Java 文書で定義された一連のバイトです。

アプリケーション自体によってアセンブルされた出力メッセージの場合、宛先オブジェクトのエンコード・プロパティーは、メッセージ内に含まれている整数および浮動小数点フィールドのエンコードを指定変更するために使用されることがあります。 例えば、浮動小数点値は IEEE 形式ではなく S/390® 形式で保管されるように要求することができます。

着信メッセージは、メッセージ自体に指定されている数値エンコードを使用して解釈されます。 この指定は、最後の WebSphere MQ ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、末尾のヘッダーは、通常、MQRFH2 です。

BytesMessage が受信され、変更されることなく再送信される場合、メッセージ本体は、受信されたとおりに、バイトごとに送信されます。 宛先オブジェクトのエンコード・プロパティーはメッセージ本体に影響を与えません。 BytesMessage 内で明示的に送信できる唯一のストリング系エンティティーは、UTF8 ストリングです。 これは Java UTF8 形式でエンコードされ、2 バイトの長さフィールドで始まります。 宛先オブジェクトの文字セット・プロパティーは、出力 BytesMessage のエンコードに影響を与えません。 着信 WebSphere MQ メッセージ内の文字セット値は、そのメッセージの JMS BytesMessage としての解釈に影響を与えません。

非 Java アプリケーションは、Java UTF8 エンコードを認識しない可能性があります。 したがって、テキスト・データを含む BytesMessage を送信する JMS アプリケーションの場合、アプリケーション自体は、そのストリングをバイト配列に変換し、これらのバイト配列を BytesMessage に書き込まなければなりません。

MapMessage

MapMessage は、XML の名前/タイプ/値のトリプレットを含むストリングです。次のようにエンコードされます。

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

ここで、datatype は、820 ページの表 109 に示したデータ・タイプの 1 つになります。 デフォルトのデータ・タイプは string です。そのため、ストリング・エレメントでは、属性 dt="string" は省略されます。

マップ・メッセージの本体を形成する XML ストリングのエンコードまたは解釈に使用される文字セットは、テキスト・メッセージに適用される規則に従って決定されます。

バージョン 5.3 より前のバージョンの WebSphere MQ classes for JMS では、次のようなフォーマットを使用して、マップ・メッセージの本体をエンコードしていました。

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

WebSphere MQ classes for JMS のバージョン 5.3 以降のバージョンはどちらのフォーマットでも解釈できますが、WebSphere MQ classes for JMS のバージョン 5.3 より前のバージョンは現行フォーマットを解釈できません。

アプリケーションが、バージョン 5.3 より前の WebSphere MQ classes for JMS を使用する別のアプリケーションにマップ・メッセージを送信する必要がある場合、送信側アプリケーションは接続ファクトリー・メソッド setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE) を呼び出して、マップ・メッセージが以前のフォーマットで送信されることを指定する必要があります。そうすれば、どの WebSphere MQ JMS クライアントもそれを解釈できます。 デフォルトでは、すべてのマップ・メッセージは現行フォーマットで送信されます。

StreamMessage

StreamMessage は、マップ・メッセージと同様のものですが、エレメント名は持ちません。

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

ここで、datatype は、820 ページの表 109 に示したデータ・タイプの 1 つになります。 デフォルトのデータ・タイプは string です。そのため、ストリング・エレメントでは、属性 dt="string" は省略されます。

StreamMessage 本体を構成する XML ストリングのエンコードまたは解釈に使用される文字セットは、TextMessage に適用される規則に従って決定されます。

MQRFH2.format フィールドは、以下のように設定されます。

MQFMT_NONE
ObjectMessage、BytesMessage、または本体がないメッセージの場合。

MQFMT_STRING
TextMessage、StreamMessage、または MapMessage の場合。

JMS メッセージ変換

JMS のメッセージ・データ変換は、メッセージの送受信時に実行されます。 WebSphere MQ では、ほとんどのデータ変換が自動的に実行されます。 テキストと数値データは、JMS アプリケーション間でメッセージを転送するときに変換されます。 テキストは、JMS アプリケーションと WebSphere MQ アプリケーション間で JMSTextMessage を交換するときに変換されます。

複雑なメッセージ交換を実行する場合は、以下の各トピックが参考になります。 複雑なメッセージ交換には次のようなものがあります。

- WebSphere MQ アプリケーションと JMS アプリケーション間における非テキスト・メッセージの転送
- バイト形式のテキスト・データの交換
- アプリケーションでのテキストの変換

JMS メッセージ・データ

データ変換は、アプリケーション間でテキストと数値データを交換するために必要です。 2 つの JMS アプリケーション間で交換する場合も同様です。 テキストと数値の内部表記は、メッセージで転送できるようにエンコードする必要があります。 エンコードにより、数値とテキストの表記方法が強制的に決定されます。 WebSphere MQ は、JMSObjectMessage を除く JMS メッセージ内のテキストと数値のエンコードを管理します。 839 ページの『JMSObjectMessage』を参照してください。 3 つのメッセージ属性を使用します。 3 つの属性は、CodedCharacterSetId、Encoding、および Format です。

これらの 3 つのメッセージ属性は、通常は JMS ヘッダー「MQRFH2」内、JMS メッセージの各フィールドに格納されます。 メッセージ・タイプが JMS ではなく MQ の場合には、属性はメッセージ記述子 MQMD に格納されます。 これらの属性は JMS メッセージ・データの変換に使用されます。 JMS メッセージ・データは、WebSphere MQ メッセージのメッセージ・データ部分で転送されます。

JMS メッセージのプロパティー

JMS_IBM_CHARACTER_SET などの JMS メッセージのプロパティーは、JMS メッセージの MQRFH2 ヘッダー部分で交換されます。 ただし、メッセージが MQRFH2 なしで送信された場合を除きます。 MQRFH2 なしで送信できるのは、JMSTextMessage と JMSBytesMessage のみです。 JMS プロパティーが WebSphere MQ メッセージ・プロパティーとしてメッセージ記述子 MQMD に格納されている場合は、MQMD 変換の一部として変換されます。 JMS プロパティーが MQRFH2 に格納されている場合は、MQRFH2.NameValueCCSID で指定された文字セットで格納されています。 メッセージを送受信する際に、メッセージ・プロパティーは JVM で内部表記との間で変換されます。 変換は、メッセージ記述子の文字セットまたは MQRFH2.NameValueCCSID との間で行われます。 数値データはテキストに変換されます。

JMS メッセージ変換

以下のトピックには、変換を必要とする複雑なメッセージを交換する場合に役立つ例とタスクが記載されています。

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、WebSphere MQ によって自動的に実行されます。

メッセージ変換のアプローチ方法に関するいくつかの質問を紹介します。

メッセージ変換について考慮する必要はありますか。
JMS 間のメッセージ転送で IBM WebSphere MQ プログラムを使用してテキスト・メッセージを交換する場合などでは、IBM WebSphere MQ によって必要な変換が自動的に実行されます。 しかし、パフォーマンス上の理由でデータ変換を自分で制御したい場合や、定義済みフォーマットを持つ複雑なメッセージを交換する場合もあります。 このような場合は、メッセージ変換を理解し、次のトピックを読む必要があります。

変換にはどのような種類がありますか。
変換には、主に 4 つの種類があります。それぞれについて次のセクションで説明します。

1. 833 ページの『JMS クライアント・データ変換』
2. 834 ページの『アプリケーション・データの変換』
3. 834 ページの『キュー・マネージャー・データ変換』
4. 835 ページの『メッセージ・チャネル・データ変換』

変換はどこで実行する必要がありますか。
836 ページの『メッセージ変換へのアプローチの選択: 「受信側で実行」』のセクションでは、"「受信側で実行」"の通常のアプローチについて説明します。 "「受信側が良好」" は、JMS データ変換にも適用されます。

JMS クライアント・データ変換

JMS クライアント [4] データ変換とは、Java プリミティブおよびオブジェクトが宛先に送信されるときに JMS メッセージ内のバイトに変換され、受信されるときに再び変換されることです。 JMS クライアント・データ変換は、JMSMessage クラスのメソッドを使用します。 メソッドは、837 ページの表 119 の JMSMessage クラス・タイプ別にリストされています。

数値とテキストの内部 JVM 表記との変換は、read、get、set、および write メソッドで実行されます。 変換が実行されるのは、メッセージの送信時、および受信したメッセージに対して read または get メソッドが呼び出されたときです。

メッセージの内容の書き込み (write) または設定 (set) で使用されるコード・ページと数値エンコードは、宛先の属性として定義されます。 宛先のコード・ページと数値エンコードは、管理上変更できます。 アプリケーションで、メッセージ内容の書き込みまたは設定を制御するメッセージ・プロパティーを設定して、宛先のコード・ページとエンコードを指定変更することもできます。

Native エンコードが定義されていない宛先への JMSBytesMessage メッセージの送信時に、数値エンコードを変換する場合は、メッセージの送信前にメッセージ・プロパティー JMS_IBM_ENCODING を設定する必要があります。 "受信側が適切な" パターンに従っている場合、または JMS アプリケーション間でメッセージを交換する場合は、アプリケーションで JMS_IBM_ENCODING を設定する必要はありません。 ほとんどの場合、Encoding プロパティーは Native のままで構いません。

JMSStreamMessage、JMSMapMessage、および JMSTextMessage の各メッセージでは、宛先の文字セット ID のプロパティーが使用されます。 数値がテキスト形式で書き出されると、エンコードは送信時に無

[4] "JMS クライアント" とは、クライアント・モードまたはバインディング・モードで実行される JMS インターフェースを実装する WebSphere MQ classes for JMS を指します。

視されます。 宛先の文字セットのプロパティーが適用される場合、メッセージの送信前に JMS クライアント・アプリケーション・プログラムで JMS_IBM_CHARACTER_SET を設定する必要はありません。

メッセージのデータを取得するために、アプリケーションは JMS メッセージの read メソッドまたは get メソッドを呼び出します。 これらのメソッドは、前のメッセージ・ヘッダーで定義されたコード・ページとエンコードを参照して、Java プリミティブとオブジェクトを正しく作成します。

JMS クライアント・データ変換は、JMS クライアント間でメッセージを交換するほとんどの JMS アプリケーションのニーズを満たすものです。 明示的なデータ変換をコーディングする必要はありません。 テキストをファイルに書き込むときに一般的に使用される java.nio.charset.Charset クラスは使用しないでください。 writeString メソッドと setString メソッドによって変換が実行されます。

JMS クライアント・データ変換の詳細については、846 ページの『JMS クライアント・メッセージ変換とエンコード』を参照してください。

アプリケーション・データの変換

JMS クライアント・アプリケーションは、 java.nio.charset.Charset クラスを使用して明示的な文字データ変換を実行できます。 838 ページの図 132 および 838 ページの図 133 の例を参照してください。 ストリング・データは、getBytes メソッドを使用してバイトに変換され、バイトとして送信されます。 このバイトは、バイト配列と Charset を取得する String コンストラクターを使用して、再びテキストに変換されます。 文字データは、encode メソッドと decode Charset メソッドを使用して変換されます。 通常、メッセージは JMSBytesMessage として送信または受信されます。これは、JMSBytesMessage のメッセージ部分に、アプリケーションによって書き込まれたデータ以外のものが含まれていないためです。[5]JMSStreamMessage、JMSMapMessage、または JMSObjectMessage を使用してバイトを送信および受信することもできます。

異なるエンコード形式で表される数値データを含むバイトをエンコードおよびデコードするための Java メソッドはありません。 数値データは、数値の JMSMessage の read メソッドおよび write メソッドを使用して自動的にエンコードおよびデコードされます。 read メソッドおよび write メソッドでは、メッセージ・データの JMS_IBM_ENCODING 属性の値を使用します。

アプリケーション・データ変換が一般的に使用されるのは、JMS クライアントが JMS 以外のアプリケーションからフォーマット済みのメッセージを送受信する場合です。 フォーマット済みメッセージには、データ・フィールドの長さで編成されたテキスト、数値、およびバイトのデータが含まれています。 非 JMS アプリケーションがメッセージ形式を "MQSTR "として指定していない限り、メッセージは JMSBytesMessage として構成されます。 JMSBytesMessage でフォーマット済みメッセージ・データを受信するには、一連のメソッドを呼び出す必要があります。 メソッドは、メッセージに書き込まれたフィールドと同じ順序で呼び出す必要があります。 フィールドが数値である場合、数値データのエンコードと長さを把握する必要があります。 いずれかのフィールドにバイト・データまたはテキスト・データが含まれている場合は、メッセージ内のバイト・データの長さを把握する必要があります。 フォーマット済みメッセージを使いやすい Java オブジェクトに変換するには、2 つの方法があります。

1. レコードに対応する Java クラスを構成して、メッセージの読み取りと書き込みをカプセル化します。 レコード内のデータへのアクセスには、クラスの get メソッドと set メソッドを使用します。
2. com.ibm.mq.headers クラスを拡張して、レコードに対応する Java クラスを構成します。 クラス内のデータへのアクセスは、 getStringValue(*fieldname*); という形式のタイプ固有のアクセサーを使用して行われます。

853 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照してください。

キュー・マネージャー・データ変換

WebSphere MQ V7.0 では、JMS クライアント・プログラムがメッセージを取得するときに、キュー・マネージャーによってコード・ページ変換を実行できます。 変換は C プログラムに対して実行される変換と同じです。 C プログラムは、MQGMO_CONVERT を MQGET GetMsgOpts パラメーター・オプションとして設定

[5] 1 つの例外: writeUTF を使用して書き込まれたデータは、2 バイトの長さフィールドから始まります

します。838 ページの図 131 を参照してください。 キュー・マネージャーは、WMQ_RECEIVE_CONVERSION 宛先プロパティーが WMQ_RECEIVE_CONVERSION_QMGR に設定されている場合に、メッセージを受信する JMS クライアント・プログラムに対して変換を実行します。JMS クライアント・プログラムも宛先プロパティーを設定できます。 835 ページの図 130 を参照してください。

V7.0 以前では、変換は常に JMS クライアントによって実行されていました。 JMS クライアント・データ変換は、JMS クライアントに認識されているタイプと長さの数値とテキストのシーケンスの変換に制限されます。 データ構造体は変換できません。 853 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照してください。 V7.0 のフィックスパック 7.0.1.5 より前のリリースでは、キュー・マネージャーで変換を実行できる場合、変換は常にキュー・マネージャーによって実行されます。7.0.1.5 以降、デフォルトの変換動作が V6.0 と同じ動作に戻り、すべての変換は JMS クライアントによって実行されます。 7.0.1.5 以降、または APAR IC72897 が適用された 7.0.1.4 では、新しい宛先オプション WMQ_RECEIVE_CONVERSION を設定して変換を実行する場所を制御したり、WMQ_RECEIVE_CCSID を設定して宛先のコード・ページを設定したりできます。835 ページの図 130 を参照してください。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

または、

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

図 *130.* キュー・マネージャー・データ変換の有効化

キュー・マネージャー変換の主な利点は、JMS 以外のアプリケーションとメッセージを交換するときに得られます。 メッセージの Format フィールドが定義され、宛先の文字セットまたはエンコードがメッセージと異なる場合は、ターゲット・アプリケーションからデータ変換が要求されると、キュー・マネージャーによって変換が実行されます。 キュー・マネージャーは、 CICS ブリッジ・ヘッダー (MQCIH) などの事前定義 WebSphere MQ メッセージ・タイプの 1 つに従ってフォーマット設定されたメッセージ・データを変換します。 Format フィールドがユーザー定義の場合、キュー・マネージャーは Format フィールドに指定された名前のデータ変換出口を探します。

キュー・マネージャーのデータ変換は、"「受信側で実行」"の設計パターンによる効果を最適化するために使用されます。 送信側の JMS クライアントで変換を実行する必要はありません。 JMS 以外の受信プログラムは、メッセージが必要なコード・ページとエンコードで確実に送信されるように、変換出口に依存します。 送信側が JMS クライアントで受信側が JMS 以外である場合、この例は IBM WebSphere MQ の V7.0 の前と後のリリースに適用されます。 IBM WebSphere MQ V7.0 では、変換出口は受信側の JMS プログラムでも呼び出すことができます。

データ変換出口ユーティリティー **crtmqcvx** を使用してデータ変換出口を作成し、キュー・マネージャーが独自のレコード・フォーマット済みデータを変換できるようにすることも可能です。 独自のレコード・フォーマットを作成し、com.ibm.mq.headers を使用してそれに Java クラスとしてアクセスし、独自の変換出口を使用して変換することができます。 ユーティリティーの名前は、z/OS では **CSQUCVX**、IBM i では **CVTMQMDTA** です。 853 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照してください。

メッセージ・チャネル・データ変換

WebSphere MQ の送信側、サーバー、クラスター受信側、およびクラスター送信側のチャネルには、メッセージ変換オプション CONVERT があります。 メッセージの内容は、メッセージの送信時にオプションで変換できます。 変換は、チャネルの送信側で行われます。 クラスター受信側の定義を使用して、対応するクラスター送信側チャネルが自動定義されます。

メッセージ・チャネルによるデータ変換は、通常、他の変換形式を使用できない場合に使用されます。

メッセージ変換へのアプローチの選択: "「受信側で実行」"

コード変換のための WebSphere MQ アプリケーション設計における通常のアプローチは、 "受信側"です。 "「受信側で実行」"は、メッセージ変換の数を減らします。 また、メッセージ転送時に一部の中間キュー・マネージャーでメッセージ変換が失敗した場合に、予期しないチャネル・エラーの問題が発生することを回避することもできます。 "「受信側で実行」"ルールは、受信側が実行できない理由がある場合にのみ中断されます。 例えば、受信側のプラットフォームに適切な文字セットがない場合があります。

"「受信側が良好」" は、JMS クライアント・アプリケーションの一般的なガイダンスでもあります。 ただし、特定のケースでは、送信元で正しい文字セットに変換したほうが効率的な場合があります。 JVM 内部表記からの変換は、テキスト・タイプや数値タイプを含んだメッセージの送信時に行う必要があります。 受信側が JMS クライアントでない場合、受信側に必要な文字セットに変換しておくことにより、JMS 以外の受信側で変換を実行しなくてもよくなる場合があります。 受信側が JMS クライアントの場合、メッセージ・データをデコードし、Java プリミティブとオブジェクトを作成するために、再度変換を行います。

JMS クライアント・アプリケーションと、C などの言語で作成されたアプリケーションとの違いは、Java がデータ変換を実行する必要があることです。 Java アプリケーションは、数値とテキストを内部表現からメッセージで使用されるエンコード形式に変換する必要があります。

宛先またはメッセージ・プロパティーを設定することにより、WebSphere MQ で使用される文字セットとエンコードを設定して、メッセージ内の数値とテキストをエンコードできます。 通常、文字セットは 1208、エンコードは Native のままにします。

WebSphere MQ では、バイト配列は変換されません。 ストリングと文字配列をバイト配列にエンコードするには、java.nio.charset パッケージを使用します。 Charset では、ストリングまたは文字配列をバイト配列に変換するために使用される文字セットを指定します。 Charset を使用してバイト配列をストリングまたは文字配列にデコードすることもできます。 ストリングと文字配列をエンコードするときに java.nio.charset.Charset.defaultCodePage に依存することはお勧めしません。 デフォルトの Charset は、通常、 Windows では windows-1252 、 UNIX では UTF-8 です。 windows-1252 は 1 バイト文字セット、UTF-8 はマルチバイト文字セットです。

通常、他の JMS アプリケーションとメッセージを交換する場合は、宛先の文字セットとエンコード・プロパティーをそれぞれデフォルト値の UTF-8 と Native のままにします。 数値またはテキストが含まれるメッセージを JMS アプリケーションと交換する場合は、メッセージ・タイプを JMSTextMessage、JMSStreamMessage、JMSMapMessage、または JMSObjectMessage の中から目的に合わせて選択します。 その他に実行する変換タスクはありません。

レコード・フォーマットを使用する JMS 以外のアプリケーションとメッセージを交換する場合は、さらに複雑になります。 レコード全体にテキストが含まれ、JMSTextMessage として転送できる場合を除き、アプリケーションでテキストをエンコードおよびデコードする必要があります。 宛先のメッセージ・タイプを MQ に設定し、JMSBytesMessage を使用して、IBM WebSphere MQ classes for JMS によって追加のヘッダーとタグ付け情報がメッセージ・データに追加されるのを防ぎます。 数値とバイトの書き込みには JMSBytesMessage メソッドを使用し、テキストをバイト配列に明示的に変換するには Charset クラスを使用します。 多くの要因が文字セットの選択に影響します。

- パフォーマンス: 最も多くのサーバーで使用されている文字セットにテキストを変換することによって変換の回数を減らすことができますか。
- 均一性: すべてのメッセージを同じ文字セットで転送します。
- 豊富な種類: アプリケーションで使用する必要があるすべてのコード・ポイントを含む文字セットはどれですか。
- 単純さ: 可変長およびマルチバイトの文字セットより、1 バイト文字セットのほうが簡単に使用できます。

853 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』を参照してください。 非 JMS アプリケーションと交換されるメッセージの変換の例を示します。

例

メッセージ・タイプと変換タイプの表

表 *119.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**Text**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

図 *131. amqsget0.c* からのコード・スニペット

JMSBytesMessage でのテキストの送受信

838 ページの図 132 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 タイプの混合を含むテキスト・ストリングをバイト・メッセージで受信するには、 838 ページの図 133 で *TEXT_LENGTH* と呼ばれるストリングの長さ (バイト単位) を知っている必要があります。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                  .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *132. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *133. JMSBytesMessage* からの *String* の受信

関連概念

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 V7.0 以降では、メッセージを受信する JMS クライアントもキュー・マネージャー・データ変換を使用します。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、キュー・マネージャー・データ変換はオプションです。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換

このタスクで示されている手順に従って、データ変換出口の設計と構築、および JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できる JMS クライアント・アプリケーションの設計と構築を行います。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換

メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMS メッセージ・タイプと変換

メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMSObjectMessage

JMSObjectMessage には、1 つのオブジェクトおよびそのオブジェクトが参照するすべてのオブジェクトが含まれ、これらのオブジェクトは JVM によってバイト・ストリームにシリアライズされています。 テキストは UTF-8 にシリアライズされ、65534 バイト以下のストリングまたは文字配列に制限されます。 JMSObjectMessage の利点は、アプリケーションがオブジェクトのメソッドと属性のみを使用している限り、データ変換の実行に関与しないという点です。 JMSObjectMessage には複雑なオブジェクト用のデータ変換が用意されているため、アプリケーション・プログラマーがメッセージのオブジェクトのエンコード方法を考慮する必要はありません。 JMSObjectMessage を使用した場合の欠点は、メッセージを交換できる相手が他の JMS アプリケーションのみであるという点です。 他の JMS メッセージ・タイプを選択することによって、JMS メッセージを JMS 以外のアプリケーションと交換できます。

842 ページの『JMSObjectMessage の送受信』に、メッセージで交換される String オブジェクトを示します。

JMS クライアント・アプリケーションは、JMS スタイルの本体を含むメッセージでのみ JMSObjectMessage を受信できます。 宛先で JMS スタイルの本体を指定する必要があります。

JMSTextMessage

JMSTextMessage には、1 つのテキスト・ストリングが格納されます。 テキスト・メッセージが送信されると、テキスト Format は "MQSTR　　"、 WMQConstants.MQFMT_STRING に設定されます。 テキストの CodedCharacterSetId は、宛先に定義されたコード化文字セット ID に設定されます。 テキストは、WebSphere MQ によって CodedCharacterSetId にエンコードされます。 CodedCharacterSetId フィールドと Format フィールドは、メッセージ記述子 MQMD で設定されるか、MQRFH2 の JMS フィールド内に設定されます。 メッセージの本体スタイルが WMQ_MESSAGE_BODY_MQ として定義されているか本体スタイルが指定されておらず、ターゲット宛先が WMQ_TARGET_DEST_MQ である場合、メッセージ記述子フィールドが設定されます。 それ以外の場合、メッセージには JMS RFH2 が含まれ、フィールドは MQRFH2 の固定部分に設定されます。

アプリケーションは、宛先に定義されているコード化文字セット ID を指定変更できます。 メッセージ・プロパティー JMS_IBM_CHARACTER_SET をコード化文字セット ID に設定する必要があります。842 ページの『JMSTextmessage の送受信』 の例を参照してください。

JMS クライアントが consumer.receive メソッドを呼び出す場合、キュー・マネージャー変換はオプションです。 キュー・マネージャー変換は、宛先プロパティー WMQ_RECEIVE_CONVERSION を WMQ_RECEIVE_CONVERSION_QMGR に設定すると有効になります。 キュー・マネージャーは、メッセージを JMS クライアントに転送する前にメッセージに対して指定されている JMS_IBM_CHARACTER_SET から、テキスト・メッセージを変換します。 宛先に別の WMQ_RECEIVE_CCSID が設定されているのでない限り、 変換後のメッセージの文字セットは 1208、UTF-8 です。 JMSTextMessage を参照するメッセー

ジ内の CodedCharacterSetId は、ターゲットの文字セット ID に更新されます。 テキストは、getText メソッドによってターゲットの文字セットから Unicode にデコードされます。842 ページの『JMSTextmessage の送受信』の例を参照してください。

JMSTextMessage は、MQ スタイルのメッセージ本体で JMS MQRFH2 ヘッダーなしで送信できます。 メッセージ本体のスタイルは、アプリケーションによって指定変更されない限り、宛先属性 WMQ_MESSAGE_BODY と WMQ_TARGET_DEST の値によって決まります。 アプリケーションは、destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) または destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) を呼び出すことにより、宛先に設定された値を指定変更できます。

MQ スタイルの本体で JMSTextMessage を宛先に送信するときに、WMQ_MESSAGE_BODY を WMQ_MESSAGE_BODY_MQ に設定した場合、メッセージを同じ宛先から JMSTextMessage として受信することはできません。 WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定された宛先から受信したすべてのメッセージは、JMSBytesMessage として受信されます。 メッセージを JMSTextMessage として受信しようとすると、例外 ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to javax.jms.TextMessage が発生します。

注 : JMSBytesMessage 内のテキストは JMS クライアントによって変換されません。 クライアントは、メッセージ内のテキストをバイト配列としてのみ受信できます。 キュー・マネージャー変換が有効になっている場合、テキストはキュー・マネージャーによって変換されますが、JMS クライアントはそのテキストを JMSBytesMessage でバイト配列として受信する必要があります。

通常は、WMQ_TARGET_DEST プロパティーを使用して、JMSTextMessage の本体スタイルを MQ と JMS のどちらで送信するかを制御したほうがよいでしょう。 その後、WMQ_TARGET_DEST が WMQ_TARGET_DEST_MQ または WMQ_TARGET_DEST_JMS に設定された宛先からメッセージを受信できます。 WMQ_TARGET_DEST は受信側に影響を及ぼしません。

JMSMapMessage および JMSStreamMessage

これらの 2 つの JMS メッセージ・タイプは似ています。 DataInputStream および DataOutputStream インターフェースに基づくメソッドを使用して、プリミティブ型をメッセージから読み取ったりメッセージに書き込んだりできます。844 ページの『メッセージ・タイプと変換タイプの表』を参照してください。 詳細については、846 ページの『JMS クライアント・メッセージ変換とエンコード』で説明します。 各プリミティブには、タグが付いています。830 ページの『JMS メッセージ本体』を参照してください。

数値データは、XML テキストとしてエンコードされたメッセージに対して読み書きされます。 宛先プロパティー JMS_IBM_ENCODING に対する参照は行われません。 テキスト・データは、JMSTextMessage 内のテキストと同じように扱われます。 843 ページの図 138 の例で作成されたメッセージ内容を調べると、すべてのメッセージ・データは、文字セット値 37 で送信されたために EBCDIC になっています。

JMSMapMessage または JMSStreamMessage では複数の項目を送信できます。

JMSMapMessage で指定された名前または JMSStreamMessage で指定された位置によってデータの個々の項目を取得できます。 各項目は、メッセージに格納されている CodedCharacterSetId 値を使用して get メソッドまたは read メソッドを呼び出すと、デコードされます。 項目の取得に使用されたメソッドによって、送信されたタイプとは異なるタイプが返された場合、タイプは変換されます。 タイプを変換できない場合、例外がスローされます。 詳細については、Class JMSStreamMessage を参照してください。 843 ページの『JMSStreamMessage および JMSMapMessage でのデータの送信』 の例は、型変換と、シーケンスからの JMSMapMessage の内容の取得を示しています。

JMSMapMessage および JMSStreamMessage の MQRFH2.format フィールドは "MQSTR "に設定されます。 宛先プロパティー WMQ_RECEIVE_CONVERSION が WMQ_RECEIVE_CONVERSION_QMGR に設定されている場合、メッセージ・データは JMS クライアントに送信される前にキュー・マネージャーによって変換されます。 メッセージの MQRFH2.CodedCharacterSetId は宛先の WMQ_RECEIVE_CCSID です。 MQRFH2.Encoding は Native です。 WMQ_RECEIVE_CONVERSION が WMQ_RECEIVE_CONVERSION_CLIENT_MSG である場合、MQRFH2 の CodedCharacterSetId と Encoding は送信側によって設定された値です。

JMS クライアント・アプリケーションは、JMS スタイルの本体を含むメッセージでのみ、MQ スタイルの本体が指定されていない宛先から JMSMapMessage または JMSStreamMessage を受信できます。

JMSBytesMessage

JMSBytesMessage には複数のプリミティブ型を含めることができます。 DataInputStream および DataOutputStream インターフェースに基づくメソッドを使用して、プリミティブ型をメッセージから読み取ったりメッセージに書き込んだりできます。844 ページの『メッセージ・タイプと変換タイプの表』を参照してください。 詳細については、839 ページの『JMS メッセージ・タイプと変換』で説明します。

メッセージ内の数値データのエンコードは、数値データを JMSBytesMessage に書き込む前に設定された JMS_IBM_ENCODING の値によって制御されます。 アプリケーションは、メッセージ・プロパティー JMS_IBM_ENCODING を設定して JMSBytesMessage に対して定義されたデフォルトの Native エンコードをオーバーライドできます。

テキスト・データは、readUTF および writeUTF を使用して UTF-8 で読み書きするか、readChar メソッドおよび writeChar メソッドを使用して Unicode ストリングで読み書きすることができます。 CodedCharacterSetId を使用するメソッドはありません。 または、JMS クライアントは、Charset クラスを使用してテキストをバイトにエンコードおよびデコードできます。 WebSphere MQ classes for JMS で変換を実行せずに JVM とメッセージの間でバイトを転送します。843 ページの『JMSBytesMessage でのテキストの送受信』を参照してください。

MQ アプリケーションに送信された JMSBytesMessage は、通常、MQ スタイルのメッセージ本体で JMS MQRFH2 ヘッダーなしで送信されます。 JMS アプリケーションに送信された場合、メッセージ本体のスタイルは通常 JMS です。 メッセージ本体のスタイルは、アプリケーションによって指定変更されない限り、宛先属性 WMQ_MESSAGE_BODY と WMQ_TARGET_DEST の値によって決まります。 アプリケーションは、destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) または destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ) を呼び出すことにより、宛先に設定された値を指定変更できます。

MQ スタイルの本体で JMSBytesMessage を送信した場合、MQ または JMS メッセージの本体スタイルを定義する宛先からメッセージを受信できます。 JMS スタイルの本体で JMSBytesMessage を送信した場合、JMS メッセージの本体スタイルを定義する宛先からメッセージを受信する必要があります。 そうしなければ、MQRFH2 はユーザー・メッセージ・データの一部として処理され、予期しているものとは異なる場合があります。

メッセージの本体スタイルが MQ と JMS のいずれであっても、その受信方法は WMQ_TARGET_DEST の設定の影響を受けません。

メッセージ・データに Format が指定されていて、キュー・マネージャーのデータ変換が有効である場合、メッセージは後でキュー・マネージャーによって変換されることがあります。 メッセージ・データのフォーマットを指定する以外の目的でフォーマット・フィールドを使用しないでください。または空白 (MQConstants.MQFMT_NONE) のままにしておくことは可能です。

JMSBytesMessage で複数の項目を送信できます。 メッセージに定義されているエンコードを使用してメッセージが送信されるときに、各数値項目が変換されます。

JMSBytesMessage からデータの個々の項目を取得できます。 メッセージを作成するために write メソッドが呼び出されたのと同じ順序で read メソッドを呼び出します。 各数値項目は、メッセージに格納されている Encoding 値を使用してメッセージが呼び出されるときに変換されます。

JMSMapMessage や JMSStreamMessage とは異なり、JMSBytesMessage には、アプリケーションによって書き込まれたデータのみが含まれます。 メッセージ・データには、JMSMapMessage および JMSStreamMessage の項目の定義に使用される XML タグなどの追加データは格納されません。 そのため、他のアプリケーション用にフォーマットされたメッセージを転送するには、JMSBytesMessage を使用します。

JMSBytesMessage、DataInputStream、および DataOutputStream の間の変換は、一部のアプリケーションで役立ちます。 JMS で com.ibm.mq.header パッケージを使用するには、843 ページの

『DataInputStream および DataOutputStream を使用したメッセージの読み書き』の例に基づくコードが必要です。

例

JMSObjectMessage の送受信

```
ObjectMessage omo = session.createObjectMessage();
omo.setObject(new String("A string"));
producer.send(omo);
...
ObjectMessage omi = (ObjectMessage)consumer.receive();
System.out.println((String)omi.getObject());
...
A string
```

図 *134. JMSObjectMessage* の送受信

JMSTextmessage の送受信

テキスト・メッセージに、異なる文字セットのテキストを含めることはできません。 例では、2 つの異なるメッセージに送信された、異なる文字セットのテキストを示します。

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

図 *135.* 宛先で定義された文字セットのテキスト・メッセージの送信

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

図 *136. ccsid 37* のテキスト・メッセージの送信

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

図 *137.* テキスト・メッセージの受信

JMSStreamMessage および JMSMapMessage でのデータの送信

```
StreamMessage smo = session.createStreamMessage();
smo.writeString("256");
smo.writeInt(512);
smo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(smo);
...
MapMessage mmo = session.createMapMessage();
mmo.setString("First", "256");
mmo.setInt("Second", 512);
mmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
producer.send(mmo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println("Stream: First as float "   + smi.readFloat() +
                           " Second as String " + smi.readString());
...
Stream: First as float: 256.0, Second as String: 512
...
MapMessage mmi = (MapMessage)consumer.receive();
System.out.println("Map: Second as String " + mmi.getString("Second") +
                       " First as double "  + mmi.getDouble("First"));
...
Map: Second as String: 512, First as double: 256.0
```

図 *138. JMSStreamMessage* および *JMSMapMessage* でのデータの送信

JMSBytesMessage でのテキストの送受信

843 ページの図 139 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 タイプの混合を含むテキスト・ストリングをバイト・メッセージで受信するには、843 ページの図 140 で *TEXT_LENGTH* と呼ばれるストリングの長さ (バイト単位) を知っている必要があります。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                  .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *139. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *140. JMSBytesMessage* からの *String* の受信

DataInputStream および DataOutputStream を使用したメッセージの読み書き

844 ページの図 141 のコードは、DataOutputStream を使用して JMSBytesMessage を作成します。

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
DataOutputStream dout = new DataOutputStream(bout);
BytesMessage messageOut = prod.session.createBytesMessage();
// messageOut.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
//                           ((MQDestination) (prod.destination)).getIntProperty
//                           (WMQConstants.WMQ_ENCODING));
int ccsidOut = (((MQDestination)prod.destination).getIntProperty(WMQConstants.WMQ_CCSID));
String codePageOut = CCSID.getCodepage(ccsidOut);
dout.writeInt(ccsidOut);
dout.write(codePageOut.getBytes(codePageOut));
messageOut.writeBytes(bout.toByteArray());
producer.send(messageOut);
```

図 *141. DataOutputStream* を使用した *JMSBytesMessage* の送信

JMS_IBM_ENCODING プロパティーを設定するステートメントがコメント化されています。 JMSBytesMessage に直接書き込む場合、ステートメントは有効ですが、DataOutputStream に書き込む場合は有効ではありません。 DataOutputStream に書き込まれる数値は、Native エンコードでエンコードされます。 JMS_IBM_ENCODING を設定しても効果はありません。

844 ページの図 142 のコードは、DataInputStream を使用して JMSBytesMessage を受信します。

```
static final int ccsidIn_SIZE = (Integer.SIZE)/8;
...
connection.start();
BytesMessage messageIn = (BytesMessage) consumer.receive();
int messageLength = new Long(messageIn.getBodyLength()).intValue();
byte [] bin = new byte[messageLength];
messageIn.readBytes(bin, messageLength);
DataInputStream din = new DataInputStream(new ByteArrayInputStream(bin));
int ccsidIn = din.readInt();
byte [] codePageByte = new byte[messageLength - ccsidIn_SIZE];
din.read(codePageByte, 0, codePageByte.length);
System.out.println("CCSID " + ccsidIn + " code page " + new String(codePageByte,
                    messageIn.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET)));
```

図 *142. DataInputStream* を使用した *JMSBytesMessage* の受信

コード・ページは、入力メッセージ・データのコード・ページ・プロパティー JMS_IBM_CHARACTER_SET を使用して印刷されます。 入力では、JMS_IBM_CHARACTER_SET は Java コード・ページであり、数字のコード化文字セット ID ではありません。

メッセージ・タイプと変換タイプの表

表 *120.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**Text**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

表 120. メッセージ・タイプと変換タイプ (続き)

	変換タイプ			
メッセージ・タイプ	**Text**	**数字**	**その他**	**なし**
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

関連概念

JMS メッセージ変換のアプローチ

JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、WebSphere MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード

JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換

キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 V7.0 以降では、メッセージを受信する JMS クライアントもキュー・マネージャー・データ変換を使用します。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、キュー・マネージャー・データ変換はオプションです。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
このタスクで示されている手順に従って、データ変換出口の設計と構築、および JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できる JMS クライアント・アプリケーションの設計と構築を行います。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

変換およびエンコードは、Java プリミティブまたはオブジェクトが JMS メッセージとの間で読み書きされるときに発生します。 この変換は JMS クライアント・データ変換と呼ばれ、キュー・マネージャー・データ変換やアプリケーション・データ変換とは区別されます。 変換は、JMS メッセージに対してデータが読み書きされる場合にのみ行われます。 テキストは、内部の 16 ビット Unicode 表記との間で変換されます。[6] メッセージ内のテキストに使用されている文字セットに変換されます。 数値データはメッセージに定義されたエンコードに変換され、Java プリミティブ数値型はメッセージに定義されたエンコードに変換されます。 変換を実行するかどうか、および実行する変換のタイプは、JMS メッセージ・タイプと読み取り操作や書き込み操作によって異なります。

846 ページの表 121 は、さまざまな JMS メッセージ・タイプの read メソッドと write メソッドを、実行される変換のタイプに分類したものです。 変換タイプについては、表の後で説明します。

表 *121.* メッセージ・タイプと変換タイプ

	変換タイプ			
メッセージ・タイプ	**Text**	**数字**	**その他**	**なし**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar

[6] 一部の Unicode 表記では、16 ビット以上が必要です。 Java SE のリファレンスを参照してください。

表 121. メッセージ・タイプと変換タイプ (続き)

	変換タイプ			
メッセージ・タイプ	**Text**	**数字**	**その他**	**なし**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Text

宛先のデフォルトの CodedCharacterSetId は 1208、UTF-8 です。 デフォルトでは、テキストは Unicode から変換され、UTF-8 テキスト・ストリングとして送信されます。 受信時に、テキストは、クライアントが受信したメッセージのコード化文字セットから Unicode に変換されます。

setText および writeString メソッドによって、テキストは Unicode から宛先に定義されている文字セットに変換されます。 アプリケーションは、メッセージ・プロパティー JMS_IBM_CHARACTER_SET を設定して、宛先の文字セットをオーバーライドできます。 メッセージを送信するとき、JMS_IBM_CHARACTER_SET は、数字で構成されるコード化文字セット ID である必要があります[7].

849 ページの『JMSTextmessage の送受信』のコード・スニペットは、2 つのメッセージを送信します。 1 つは宛先に定義されている文字セットで送信され、もう 1 つはアプリケーションによって定義された文字セット 37 で送信されます。

getText および readString メソッドは、メッセージのテキストをメッセージに定義されている文字セットから Unicode に変換します。 これらのメソッドは、メッセージ・プロパティー JMS_IBM_CHARACTER_SET に定義されているコード・ページを使用します。 コード・ページは、MQRFH2.CodedCharacterSetId からマップされますが、メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合を除きます。 メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合、コード・ページは MQMD.CodedCharacterSetId からマップされます。

850 ページの図 147 のコード・スニペットは、宛先に送信されたメッセージを受信します。 メッセージ内のテキストはコード・ページ IBM037 から Unicode に再び変換されます。

注 : テキストがコード化文字セット 37 に変換されたことを簡単に確認するには、WebSphere MQ エクスプローラーを使用します。 キューを参照し、受信前にメッセージのプロパティーを表示します。

[7] メッセージを受信する場合、 JMS_IBM_CHARACTER_SET は Java Charset コード・ページ名です。

850 ページの図 146 のコード・スニペットを 848 ページの図 143 の正しくないコード・スニペットと比較してください。 正しくないスニペットでは、テキスト・ストリングは 2 回変換されています。 1 回目はアプリケーションによって、2 回目は WebSphere MQ によって変換されています。

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

図 *143.* 正しくないコード・ページ変換

writeUTF メソッドは、テキストを Unicode から 1208、UTF-8 に変換します。 テキスト・ストリングの前には、2 バイトの長さが付加されます。 テキスト・ストリングの最大長は 65534 バイトです。 readUTF メソッドは、writeUTF メソッドによって書き込まれたメッセージ内の項目を読み取ります。 それは writeUTF メソッドによって書き込まれたバイトの数を正確に読み取ります。

数字

宛先のデフォルトの数値エンコードは Native です。 Java の Native エンコード定数の値は、273、x'00000111' であり、これはすべてのプラットフォームで同じです。 受信時に、メッセージ内の数値は数値 Java プリミティブに正しく変換されます。 変換では、メッセージに定義されているエンコードと read メソッドによって返されたタイプが使用されます。

send メソッドは、set と write によってメッセージに追加された数値を、宛先に定義されている数値エンコードに変換します。 メッセージの宛先のエンコードは、アプリケーション設定のメッセージ・プロパティー JMS_IBM_ENCODING によってオーバーライドできます。 例えば、次のとおりです。

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

数値メソッドの get と read は、メッセージ内の数値を、メッセージに定義されている数値エンコードから変換します。 これらの数値は、 read メソッドまたは get メソッドによって指定されたタイプに変換されます。 ENCODING プロパティー を参照してください。 メソッドでは、JMS_IBM_ENCODING で定義されているエンコードが使用されます。 エンコードは、MQRFH2.Encoding からマップされますが、メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合を除きます。 メッセージが MQ タイプのメッセージで MQRFH2 を含んでいない場合、メソッドでは、MQMD.Encoding で定義されているエンコードが使用されます。

850 ページの図 148 の例は、宛先の形式で数値をエンコードし、それを JMSStreamMessage で送信するアプリケーションを示しています。 850 ページの図 148 の例を 850 ページの図 149 の例と比較してください。 違いは、JMS_IBM_ENCODING を JMSBytesMessage で設定する必要があるという点です。

注 : 数値が正しくエンコードされたことを簡単に確認するには、WebSphere MQ エクスプローラーを使用します。 キューを参照し、コンシューム前にメッセージのプロパティーを表示します。

その他

boolean メソッドは、JMSByteMessage、JMSStreamMessage、および JMSMapMessage で true および false を x'01' および x'00' としてエンコードします。

UTF メソッドは Unicode を UTF-8 テキスト・ストリングにエンコードおよびデコードします。 ストリングは 65536 文字未満に制限され、2 バイトの長さフィールドがストリングの前に付加されます。

Object メソッドは、プリミティブ型をオブジェクトとしてラップします。 数値型とテキスト型は、プリミティブ型が数値メソッドとテキスト・メソッドを使用して読み書きされる場合と同じようにエンコードまたは変換されます。

なし

readByte、readBytes、readUnsignedByte、writeByte、および writeBytes メソッドは、1 バイトまたはバイト配列をアプリケーションとメッセージ間で変換なしに取得または書き込みます。 readChar および writeChar メソッドは、2 バイトの Unicode 文字をアプリケーションとメッセージ間で変換なしに取得および書き込みます。

readBytes および writeBytes メソッドを使用すると、アプリケーションは、850 ページの『JMSBytesMessage でのテキストの送受信』と同様に独自のコード・ポイント変換を実行できます。

WebSphere MQ は、メッセージが JMSBytesMessage で、readBytes および writeBytes メソッドが使用されるため、クライアントでコード・ページ変換を実行しません。 ただし、バイトがテキストを表す場合は、アプリケーションによって使用されるコード・ページが宛先のコード化文字セットに一致することを確認してください。 メッセージは、キュー・マネージャーの変換出口によって再び変換される場合があります。 別の可能性として、受信側の JMS クライアント・プログラムは、メッセージ内の JMS_IBM_CHARACTER_SET プロパティーを使用して、メッセージ内のテキストを表すバイト配列をストリングまたは文字に変換する規則に従うことがあります。

この例では、クライアントは宛先のコード化文字セットを変換に使用します。

```
bytes.writeBytes("In the destination code page".getBytes(
      CCSID.getCodepage(((MQDestination) destination)
           .getIntProperty(WMQConstants.WMQ_CCSID))));
```

あるいは、クライアントは、コード・ページを選択してから、対応するコード化文字セットをメッセージの JMS_IBM_CHARACTER_SET プロパティーに設定した可能性があります。 WebSphere MQ classes for Java では、 JMS_IBM_CHARACTER_SET を使用して、 MQRFH2 の JMS プロパティーまたはメッセージ記述子 MQMD の CodedCharacterSetId フィールドを設定します。

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);[8]
```

バイト配列が JMSStringMessage または JMSMapMessage に書き込まれると、WebSphere MQ classes for JMS はバイトが、JMSStringMessage および JMSMapMessage 内のテキストではなく 16 進データとして入力されるため、データ変換を実行しません。

アプリケーションでバイトが文字を表す場合は、メッセージに対して読み書きするコード・ポイントを考慮する必要があります。 849 ページの図 144 のコードは、宛先のコード化文字セットの使用規則に従っています。 JVM のデフォルトの文字セットを使用してストリングを作成する場合、バイト内容はプラットフォームによって異なります。 Windows 上の JVM では、通常、デフォルトの Charset は windows-1252、UNIX 上では UTF-8 です。 Windows と UNIX 間の交換では、テキストをバイトとして交換するために明示的なコード・ページを選択する必要があります。

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
           .getIntProperty(WMQConstants.WMQ_CCSID))));
```

図 *144. 宛先の文字セットを使用した、JMSStreamMessage へのストリングを表すバイトの書き込み*

例

JMSTextmessage の送受信

テキスト・メッセージに、異なる文字セットのテキストを含めることはできません。 例では、2 つの異なるメッセージに送信された、異なる文字セットのテキストを示します。

[8] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

図 *145.* 宛先で定義された文字セットのテキスト・メッセージの送信

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

図 *146. ccsid 37* のテキスト・メッセージの送信

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

図 *147.* テキスト・メッセージの受信

エンコードの例

以下の例では、宛先に定義されたエンコードでの数値の送信を示しています。 JMSBytesMessage の JMS_IBM_ENCODING プロパティーを、宛先に指定された値に設定する必要があります。

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

図 *148. JMSStreamMessage* での宛先のエンコードを使用した数値の送信

```
BytesMessage bmo = session.createBytesMessage();
bmo.writeInt(256);
int encoding = ((MQDestination) (destination)).getIntProperty
    (WMQConstants.WMQ_ENCODING)
bmo.setIntProperty(WMQConstants.JMS_IBM_ENCODING, encoding);
producer.send(bmo);
...
BytesMessage bmi = (BytesMessage)consumer.receive();
System.out.println(bmi.readInt());
...
256
```

図 *149. JMSBytesMessage* での宛先のエンコードを使用した数値の送信

JMSBytesMessage でのテキストの送受信

851 ページの図 150 のコードは、BytesMessage でストリングを送信します。 分かりやすくするために、例では JMSTextMessage に適した 1 つのストリングを送信しています。 タイプの混合を含むテキスト・ストリングをバイト・メッセージで受信するには、 851 ページの図 151 で *TEXT_LENGTH* と呼ばれるス

トリングの長さ (バイト単位) を知っている必要があります。 文字数が固定数のストリングでも、バイト表記のほうが長くなる可能性があります。

```
BytesMessage bytes = session.createBytesMessage();
String codePage = CCSID.getCodepage(((MQDestination) destination)
                  .getIntProperty(WMQConstants.WMQ_CCSID));
bytes.writeBytes("In the destination code page".getBytes(codePage));
producer.send(bytes);
```

図 *150. JMSBytesMessage* での *String* の送信

```
BytesMessage message = (BytesMessage)consumer.receive();
int TEXT_LENGTH = new Long(message.getBodyLength())).intValue();
byte[] textBytes = new byte[TEXT_LENGTH];
message.readBytes(textBytes, TEXT_LENGTH);
String codePage = message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET);
String textString = new String(textBytes, codePage);
```

図 *151. JMSBytesMessage* からの *String* の受信

関連概念

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、WebSphere MQ によって自動的に実行されます。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 V7.0 以降では、メッセージを受信する JMS クライアントもキュー・マネージャー・データ変換を使用します。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、キュー・マネージャー・データ変換はオプションです。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
このタスクで示されている手順に従って、データ変換出口の設計と構築、および JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できる JMS クライアント・アプリケーションの設計と構築を行います。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 V7.0 以降では、メッセージを受信する JMS クライアントもキュー・マネージャー・データ変換を使用します。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、キュー・マネージャー・データ変換はオプションです。

キュー・マネージャーは、メッセージ・データに設定されている CodedCharacterSetId、Encoding、および Format の値を使用して、メッセージ・データ内の文字および数値データを変換できます。 JMS 以外のアプリケーションでは、GetMessageOption と GMO_CONVERT を設定することによって変換機能を常に使用できます。 V7.0 までは、キュー・マネージャー変換機能は、メッセージを受信する JMS アプリケーションで使用できませんでした。

V7.0 までは、メッセージを送信する JMS クライアント・アプリケーションでキュー・マネージャー変換を使用できます。 JMS クライアントは、フォーマット済みレコードを構築し、メッセージ内のデータに対応する CodedCharacterSetId、Encoding、および Format 属性を設定します。 JMS 以外の受信側アプリケーションは、GMO_CONVERT を使用してメッセージを読み取り、ユーザー作成のデータ変換出口が呼び出されるようにします。 データ変換出口は、Format フィールドに名前が設定されている共有ライブラリーです。

V7.0 以降では、キュー・マネージャーで JMS クライアントに送信されたメッセージを変換できます。 7.0.0.0 から 7.0.1.4 までは、キュー・マネージャー変換は常に JMS クライアントに対して呼び出されます。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、宛先プロパティー WMQ_RECEIVE_CONVERSION を WMQ_RECEIVE_CONVERSION_QMGR または WMQ_RECEIVE_CONVERSION_CLIENT_MSG に設定してキュー・マネージャー変換を制御します。 WMQ_RECEIVE_CONVERSION_CLIENT_MSG は、JMS クライアントのキュー・マネージャー・データ変換がサポートされていなかった WebSphere MQ V6.0 の動作に一致するデフォルトの設定値です。 アプリケーションで宛先の設定を変更できます。

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

または、

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

図 *152.* キュー・マネージャー・データ変換の有効化

JMS クライアントのキュー・マネージャー・データ変換は、クライアントが consumer.receive メソッドを呼び出すときに行われます。 テキスト・データは、デフォルトでは UTF-8 (1208) に変換されます。 後続の read メソッドおよび get メソッドは、 UTF-8 から受信したデータのテキストをデコードし、Java テキスト・プリミティブを内部 Unicode エンコード方式で作成します。 UTF-8 は、キュー・マネージャー・データ変換からの唯一のターゲット文字セットではありません。 WMQ_RECEIVE_CCSID 宛先プロパティーを設定して、別の CCSID を選択できます。

アプリケーションでは、宛先設定を例えば 437 の DOS-US に設定して変更することもできます。

```
((MQDestination)destination).setIntProperty
             (WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

または、

```
((MQDestination)destination).setReceiveCCSID(437);
```

図 *153.* キュー・マネージャー変換のターゲットコード化文字セットの設定

WMQ_RECEIVE_CCSID を変更することには特殊な理由があります。 選択された CCSID によって、JVM で作成されたテキスト・オブジェクトに違いが生じることはありません。 ただし、一部の JVM では、プラットフォームによってメッセージ内のテキストの CCSID から Unicode への変換を処理できない場合があります。 オプションにより、メッセージでクライアントに送信されるテキストの CCSID を選択できます。 一部の JMS クライアント・プラットフォームには、UTF-8 で送信されるメッセージ・テキストに関する問題がありました。

JMS コードは、853 ページの図 154 の C コードの太字テキストと同じです。

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

図 *154. amqsget0.c* からのコード・スニペット

注 :

キュー・マネージャー変換は、 WebSphere MQ 形式が既知のメッセージ・データに対してのみ実行されます。 MQSTR、 または MQCIH は、事前定義されている既知の形式の例です。 データ変換出口を指定した場合に限り、既知のフォーマットにユーザー定義のフォーマットを使用することもできます。

JMSTextMessage、JMSMapMessage、および JMSStreamMessage として構成されたメッセージは、MQSTR フォーマットで、キュー・マネージャーによって変換できます。

関連概念

JMS メッセージ変換のアプローチ
JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、WebSphere MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード
JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

417 ページの『データ変換出口の起動』
データ変換出口とは、MQGET 呼び出しの処理中に制御を受け取るユーザー作成出口です。

関連タスク

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
このタスクで示されている手順に従って、データ変換出口の設計と構築、および JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できる JMS クライアント・アプリケーションの設計と構築を行います。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

JMS 以外のアプリケーションとのフォーマット済みレコードの交換
このタスクで示されている手順に従って、データ変換出口の設計と構築、および JMSBytesMessage を使用して JMS 以外のアプリケーションとメッセージを交換できる JMS クライアント・アプリケーションの設

計と構築を行います。 JMS 以外のアプリケーションとのフォーマット済みメッセージの交換は、データ変換出口を呼び出しても呼び出さなくても実行できます。

始める前に

JMSTextMessage を使用して、JMS 以外のアプリケーションとメッセージを交換するための簡単なソリューションを設計することもできますが、 このタスクの手順に従う前に、その可能性を排除します。

このタスクについて

JMS クライアントは、他の JMS クライアントと交換される JMS メッセージのフォーマットの詳細に関与しない場合は、簡単に作成できます。 メッセージ・タイプが JMSTextMessage、JMSMapMessage、JMSStreamMessage、または JMSObjectMessage である限り、WebSphere MQ はメッセージのフォーマットの詳細に注意を払います。 WebSphere MQ は、各種プラットフォームにおけるコード・ページと数値エンコードの相違を処理します。

これらのメッセージ・タイプを使用して、メッセージを JMS 以外のアプリケーションと交換できます。 そのためには、これらのメッセージが WebSphere MQ classes for JMS によってどのように構成されているかを理解する必要があります。 メッセージを解釈するように JMS 以外のアプリケーションを変更することもできます。 815 ページの『JMS メッセージの WebSphere MQ メッセージへのマッピング』を参照してください。

これらのいずれかのメッセージ・タイプを使用する利点は、JMS クライアント・プログラミングがメッセージを交換するアプリケーションのタイプに依存しないという点です。 欠点は、別のプログラムの変更が必要になる場合がありますが、他のプログラムを変更できないという点です。

その代わりに、既存のメッセージ・フォーマットを処理できる JMS クライアント・アプリケーションを作成する方法があります。 多くの場合、既存のメッセージは固定フォーマットであり、不定形式データ、テキスト、および数値が混在しています。 このタスクの手順と 857 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』の JMS クライアントの例を、フォーマット済みレコードを JMS 以外のアプリケーションと交換できる JMS クライアントを構築するための開始点として使用してください。

手順

1. レコード・レイアウトを定義するか、事前定義された WebSphere MQ ヘッダー・クラスのいずれかを使用します。

   事前定義された WebSphere MQ ヘッダーを処理するには、「WebSphere MQ メッセージ・ヘッダーの処理」を参照してください。

   855 ページの図 155 は、データ変換ユーティリティーで処理できるユーザー定義の固定長レコード・レイアウトの例です。
2. データ変換出口を作成します。

   「データ変換出口プログラムの作成」の指示に従ってデータ変換出口を作成します。

   857 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』の例を試すには、データ変換出口に MYRECORD という名前を付けます。
3. レコード・レイアウトをカプセル化し、レコードを送受信するための Java クラスを作成します。 次の 2 つの方法があります。

   - レコードが含まれる JMSBytesMessage を読み書きするクラスを作成します。 857 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』を参照してください。
   - レコードのデータ構造を定義するために com.ibm.mq.header.Header を拡張するクラスを書き込みます。「新規ヘッダー・タイプのクラスの作成」を参照してください。
4. 交換するメッセージのコード化文字セットを決定します。

   836 ページの『メッセージ変換へのアプローチの選択: 「受信側で実行」』を参照。
5. JMS MQRFH2 ヘッダーなしで MQ タイプのメッセージを交換する宛先を構成します。

送信側の宛先と受信側の宛先の両方を、MQ タイプのメッセージを交換するように構成する必要があります。 送信側と受信側の両方に同じ宛先を使用できます。

アプリケーションで宛先のメッセージ本体プロパティーをオーバーライドできます。

```
((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

857 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』 の例では、宛先のメッセージ本体プロパティーをオーバーライドして、MQ スタイルのメッセージが送信されるようにしています。

6. JMS と JMS 以外のアプリケーションでソリューションをテストします。

   データ変換出口のテストに便利なツールは、次のとおりです。

   - amqsgetc0.c サンプル・プログラムは、JMS クライアントによって送信されたメッセージの受信をテストするのに役立ちます。 856 ページの図 156 のヘッダー例 RECORD.h を使用するための推奨変更を参照してください。 変更により、amqsgetc0.c はサンプル JMS クライアント TryMyRecord.java から送信されたメッセージを受信します。 857 ページの『JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成』 を参照してください。
   - サンプル WebSphere MQ ブラウズ・プログラム amqsbcg0.c は、メッセージ・ヘッダーの内容、JMS ヘッダー、MQRFH2、およびメッセージ内容を調べるのに役立ちます。
   - **rfhutil** プログラム (以前は SupportPac IH03 で使用可能) を使用すると、テスト・メッセージをキャプチャーしてファイルに保管し、メッセージ・フローを駆動するために使用することができます。 出力メッセージをさまざまな形式で読み取ったり表示したりすることもできます。 この形式には、2 つのタイプの XML と、COBOL コピーブックに対する突き合わせも含まれます。 データは EBCDIC または ASCII のいずれかを使用します。 メッセージを送信する前に、RFH2 ヘッダーをメッセージに追加することができます。

   変更された amqsgetc0.c サンプル・プログラムを使用してメッセージを受信しようとして理由コード 2080 のエラーが発生した場合は、メッセージに MQRFH2 があるかどうかを確認します。 変更では、MQRFH2 が指定されていないメッセージが宛先に送信されたことを想定しています。

例

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                MQLONG StructLength;
                MQLONG Encoding;
                MQLONG CodeCharSetId;
                MQCHAR Format[8];
                MQLONG Flags;
                MQCHAR RecordData[32];
    };
```

図 *155. RECORD.h*

- RECORD.h データ構造の宣言

```
struct tagRECORD {
   MQCHAR4    StrucId;
   MQLONG     Version;
   MQLONG     StrucLength;
   MQLONG     Encoding;
   MQLONG     CCSID;
   MQCHAR8    Format;
   MQLONG     Flags;
   MQCHAR32   RecordData;
 };
typedef struct tagRECORD RECORD;
   typedef RECORD MQPOINTER PRECORD;
   RECORD record;
   PRECORD pRecord = &(record);
```

- RECORD, を使用するように MQGET 呼び出しを変更します。

  1. 変更前:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          buflen,          /* buffer length         */
          buffer,          /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

  2. 変更後:

```
    MQGET(Hcon,            /* connection handle     */
          Hobj,            /* object handle         */
          &md,             /* message descriptor    */
          &gmo,            /* get message options   */
          sizeof(RECORD),  /* buffer length         */
          pRecord,         /* message buffer        */
          &messlen,        /* message length        */
          &CompCode,       /* completion code       */
          &Reason);        /* reason code           */
```

- print ステートメントを変更します。

  1. 変更前:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. 終了:

```
/* buffer[messlen] = '\0';            add terminator          */
printf("ccsid <%d>, flags <%d>, message <%32.32s>\n \0",
        md.CodedCharSetId, record.Flags, record.RecordData);
```

図 *156. amqsget0.c* の変更

関連概念

JMS メッセージ変換のアプローチ

JMS アプリケーション設計者は、いくつかのデータ変換アプローチを利用できます。 これらのアプローチは排他的ではありません。これらのアプローチを組み合わせて使用するアプリケーションもあります。 アプリケーションが他の JMS アプリケーションとテキストのみを交換するか、メッセージのみを交換する場合、通常はデータ変換を考慮する必要はありません。 データ変換は、WebSphere MQ によって自動的に実行されます。

JMS クライアント・メッセージ変換とエンコード

JMS クライアント・メッセージ変換とエンコードに使用するメソッドのリスト、および各変換タイプのコード例を示します。

キュー・マネージャー・データ変換
キュー・マネージャー・データ変換は、JMS クライアントからメッセージを受信する JMS 以外のアプリケーションで常に使用できます。 V7.0 以降では、メッセージを受信する JMS クライアントもキュー・マネージャー・データ変換を使用します。 7.0.1.5、または APAR IC72897 が適用された 7.0.1.4 からは、キュー・マネージャー・データ変換はオプションです。

関連資料

JMS メッセージ・タイプと変換
メッセージ・タイプの選択は、メッセージ変換のアプローチに影響します。 ここでは、JMS メッセージ・タイプの JMSObjectMessage、JMSTextMessage、JMSMapMessage、JMSStreamMessage、および JMSBytesMessage のメッセージ変換とメッセージ・タイプの相互作用について説明します。

変換出口コードを作成するためのユーティリティー

JMSBytesMessage にレコード・レイアウトをカプセル化するクラスの作成

このタスクの目的は、JMSBytesMessage でデータ変換と固定レコード・レイアウトを結合する方法を、例を示して検討することです。 このタスクでは、いくつかの Java クラスを作成して、JMSBytesMessage 内のサンプル・レコード構造を交換します。 例を変更して、他のレコード構造を交換するクラスを作成できます。

JMSBytesMessage は、JMS 以外のプログラムと混合データ型レコードを交換するための最適な JMS メッセージ・タイプです。 JMS プロバイダーによってメッセージ本体に挿入される追加データはありません。 したがって、JMS クライアント・プログラムが既存の IBM WebSphere MQ プログラムと相互運用する場合は、これが最適なメッセージ・タイプです。 JMSBytesMessage を使用する場合の主な課題は、他のプログラムに必要なエンコードと文字セットのマッチングです。 解決策は、レコードをカプセル化するクラスの作成です。 特定のレコード・タイプについて、JMSBytesMessage の読み書きをカプセル化するクラスを使用すると、固定形式のレコードを JMS プログラムで送受信しやすくなります。 抽象クラスにインターフェースの汎用的な側面を取り込むことによって、解決策の多くをさまざまなレコード・フォーマットに再利用できます。 抽象汎用クラスを拡張するクラスにさまざまなレコード・フォーマットを実装できます。

代替手段として、com.ibm.mq.headers.Header クラスを拡張します。 Header クラスには、より宣言的な方法でレコード・フォーマットを作成する addMQLONG などのメソッドがあります。 Header クラスを使用した場合の欠点は、属性の取得と設定で複雑な解釈インターフェースが使用される点です。 どちらの方法でも、アプリケーション・コードの量は同じくらいになります。

JMSBytesMessage は、MQRFH2 に加えて 1 つのフォーマットだけを、1 つのメッセージにカプセル化します。 ただし、各レコードで同じフォーマット、コード化文字セット、およびエンコードが使用される場合を除きます。 JMSBytesMessage のフォーマット、エンコード、および文字セットは、MQRFH2 に続く、すべてのメッセージのプロパティーです。 例は、JMSBytesMessage にユーザー・レコードが 1 つのみ含まれていることを想定して作成されています。

始める前に

1. スキル・レベル: Java プログラミングおよび JMS に精通している必要があります。 Java 開発環境のセットアップに関する説明は提供されていません。 JMSTextMessage、JMSStreamMessage、または JMSMapMessage を交換するようにプログラムを作成しておくと良いでしょう。 そうすれば、JMSBytesMessage を使用したメッセージ交換の違いを確認できます。
2. 例では、IBM WebSphere MQ V7.0 が必要です。
3. この例は、 Eclipse ワークベンチの Java パースペクティブを使用して作成されました。 これには JRE 6.0 以上が必要です。 IBM WebSphere MQ エクスプローラーの Java パースペクティブを使用して、Java クラスを開発および実行できます。 あるいは、独自の Java 開発環境を使用します。
4. IBM WebSphere MQ エクスプローラーを使用すると、テスト環境の設定とデバッグをコマンド行ユーティリティーを使用する場合より簡単に実行できます。

このタスクについて

2 つのクラス、RECORD と MyRecord の作成方法についてガイドします。 これらの 2 つのクラスで固定形式のレコードがカプセル化されます。 これらのクラスには、属性を取得および設定するメソッドがありま

す。 get メソッドは JMSBytesMessage からレコードを読み取り、put メソッドはレコードを JMSBytesMessage に書き込みます。

このタスクの目的は、再利用できる実動品質のクラスを作成することではありません。 タスクの例を使用して、独自のクラスの作成に取り掛かることもできます。 このタスクの目的は、JMSBytesMessage を使用する際の、主に文字セット、フォーマット、およびエンコードの使用方法に関するガイダンスを示すことです。 クラス作成の各ステップについて説明し、見逃されることがある JMSBytesMessage の使用の側面について説明します。

RECORD クラスは抽象クラスであり、ユーザー・レコードの共通フィールドをいくつか定義します。 共通フィールドは、目印、バージョン、および長さフィールドを持つ標準の IBM WebSphere MQ ヘッダー・レイアウトでモデル化されます。 多くの IBM WebSphere MQ ヘッダーにあるエンコード、文字セット、およびフォーマット・フィールドは省略されます。 別のヘッダーは、ユーザー定義のフォーマットに従うことはできません。 RECORD クラスを拡張する MyRecord クラスは、追加のユーザー・フィールドでレコードを拡張してユーザー定義のフォーマットに従います。 クラスによって作成された JMSBytesMessage は、キュー・マネージャーのデータ変換出口によって処理できます。

864 ページの『例の実行に使用されるクラス』には、RECORD と MyRecord の完全なリストがあります。 RECORD と MyRecord をテストするための、追加の"足場"となるクラスのリストもあります。 追加のクラスは次のとおりです。

TryMyRecord
RECORD と MyRecord をテストするメインプログラムです。

EndPoint
JMS 接続、宛先、およびセッションを 1 つのクラスにカプセル化する抽象クラスです。 そのインターフェースは、RECORD と MyRecord クラスのテストのニーズを満たします。 それは JMS アプリケーションを作成するための確立された設計パターンではありません。

注: Endpoint クラスには、宛先の作成後に次のコード行が含まれます。

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

V7.0 では、V7.0.1.5 から、キュー・マネージャー変換を有効にする必要があります。 デフォルトでは無効になっています。 V7.0 の V7.0.1.4 までは、キュー・マネージャー変換はデフォルトで有効になっており、このコード行はエラーになります。

MyProducer と MyConsumer
EndPoint を拡張し、接続された、要求の受け入れ準備ができている MessageConsumer と MessageProducer を作成するクラスです。

すべてのクラスによって、JMSBytesMessage でのデータ変換の使用方法を理解するために構築および試すことのできる完全なアプリケーションが構成されます。

手順

1. デフォルトのコンストラクターで標準フィールドを IBM WebSphere MQ ヘッダーにカプセル化する抽象クラスを作成します。 後で、このクラスを拡張して、要件に合わせてヘッダーを調整できます。

```
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
```

```
        super();
    }
```

注:

a. 属性 structID から nextFormat までは、標準の IBM WebSphere MQ メッセージ・ヘッダーに配置されている順序でリストされています。

b. 属性 format、messageEncoding、および messageCharset は、ヘッダー自体を記述し、ヘッダーの一部ではありません。

c. レコードのコード化文字セット ID を格納するか、文字セットを格納するかを決定する必要があります。 Java は文字セットを使用し、 IBM WebSphere MQ メッセージはコード化文字セット ID を使用します。 例のコードでは文字セットが使用されています。

d. int は、 IBM WebSphere MQ によって MQLONG にシリアライズされます。 MQLONG は 4 バイトです。

2. 専用属性の getter と setter を作成します。

a) getter を作成または生成します。

```
    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }
```

b) setter を作成または生成します。

```
    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

3. JMSBytesMessage から RECORD インスタンスを作成するコンストラクターを作成します。

```
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }
```

注:

a. messageCharset と messageEncoding は、宛先に設定された値をオーバーライドするため、メッセージ・プロパティーから取り込まれます。 format は更新されません。 この例では、エラー・チェックは実行されません。 Record(BytesMessage) コンストラクターが呼び出される場合、JMSBytesMessage は RECORD タイプのメッセージと見なされます。 行 "setStructID(new String(structID, getMessageCharset()))" は、目印を設定します。

b. メソッドを完了するコード行は、メッセージ内のフィールドを順番にデシリアライズし、RECORD インスタンスに設定されたデフォルト値を更新します。

4. JMSBytesMessage にヘッダー・フィールドを書き込む put メソッドを作成します。

```
protected BytesMessage put(MyProducer myProducer) throws IOException,
        JMSException, UnsupportedEncodingException {
    setHeaderEncoding(myProducer.getEncoding());
    setHeaderCharset(myProducer.getCharset());
    myProducer.setMQClient(true);
    BytesMessage bytes = myProducer.session.createBytesMessage();
    bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
    bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
    bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                         myProducer.getCCSID());
    bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                    + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                    .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
    bytes.writeInt(getVersion());
    bytes.writeInt(getStructLength());
    return bytes;
}
```

注:

a. MyProducer は、JMS Connection、Destination、Session、および MessageProducer を 1 つのクラスにカプセル化します。 後で使用される MyConsumer は、JMS Connection、Destination、Session、および MessageConsumer を 1 つのクラスにカプセル化します。

b. JMSBytesMessage では、エンコードが Native 以外の場合、メッセージ内にエンコードを設定する必要があります。 宛先のエンコードがメッセージ・エンコード属性 JMS_IBM_CHARACTER_SET にコピーされ、RECORD クラスの属性として保存されます。

   i) "setMessageEncoding(myProducer.getEncoding());" は、 "(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));" を呼び出して宛先エンコードを取得します。

   ii) "Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());" は、メッセージのエンコードを設定します。

c. テキストをバイトに変換するために使用される文字セットは、宛先から取得され、RECORD クラスの属性として保存されます。 これは、 JMSBytesMessage の作成時に IBM WebSphere MQ classes for JMS によって使用されないため、メッセージ内で設定されません。

   "messageCharset = myProducer.getCharset();" 呼び出し

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

   コード化文字セット ID から Java 文字セットを取得します。

   " CCSID.getCodepage(ccsid)" は、パッケージ com.ibm.mq.headers にあります。 ccsid は、宛先を照会する MyProducer で別のメソッドから取得されます。

```
public int getCCSID() throws JMSException {
    return (((MQDestination) destination)
            .getIntProperty(WMQConstants.WMQ_CCSID));
}
```

d. "myProducer.setMQClient(true);" は、クライアント・タイプの宛先設定をオーバーライドして、強制的に IBM WebSphere MQ MQI クライアントにします。 管理構成エラーが覆い隠されるため、このコード行を省略することをお勧めします。

   "myProducer.setMQClient(true);" は以下を呼び出します。

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

   JMS の設定をオーバーライドする必要がある場合、このコードには、IBM WebSphere MQ 本体スタイルが未指定に設定されるという副作用があります。

   注:

IBM WebSphere MQ classes for JMS は、メッセージのフォーマット、エンコード、および文字セット ID をメッセージ記述子 MQMD または JMS ヘッダー MQRFH2 に書き込みます。 これは、メッセージに IBM WebSphere MQ スタイルの本体が含まれているかどうかによって異なります。 MQMD フィールドを手動で設定しないでください。

メッセージ記述子のプロパティーを手動で設定するメソッドが存在します。 このメソッドは JMS_IBM_MQMD_* プロパティーを使用します。 宛先プロパティー WMQ_MQMD_WRITE_ENABLED を設定して JMS_IBM_MQMD_* プロパティーを設定する必要があります。

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

宛先プロパティー WMQ_MQMD_READ_ENABLED を設定してプロパティーを読み取る必要があります。

メッセージ・ペイロード全体を完全に制御する場合にのみ JMS_IBM_MQMD_* を使用します。 JMS_IBM_* プロパティーとは異なり、JMS_IBM_MQMD_* プロパティーは IBM WebSphere MQ classes for JMS による JMS メッセージの構成方法を制御しません。 JMS メッセージのプロパティーと競合するメッセージ記述子プロパティーを作成する可能性があります。

e. メソッドを完了するコード行は、クラス内の属性をメッセージ内のフィールドとしてシリアライズします。

ストリング属性にはブランクが埋め込まれます。 ストリングは、レコードに定義されている文字セットを使用してバイトに変換され、メッセージ・フィールドの長さに切り捨てられます。

5. インポートを追加してクラスを完成させます。

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. RECORD クラスを拡張して追加フィールドを含めるクラスを作成します。 デフォルトのコンストラクターを含めます。

```
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
```

注:

a. RECORD サブクラスの MyRecord は、ヘッダーの目印、フォーマット、および長さをカスタマイズします。

7. getter および setter を作成または生成します。

a) getter を作成します。

```
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }Â .
```

b) setter を作成します。

```
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

8. JMSBytesMessage から MyRecord インスタンスを作成するコンストラクターを作成します。

```
    public MyRecord(BytesMessage message) throws JMSException, IOException,
           MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
```

注:

a. 標準のメッセージ・テンプレートを構成するフィールドは、まず RECORD クラスによって読み取られます。

b. recordData テキストは、メッセージの文字セット・プロパティーを使用して String に変換されます。

9. コンシューマーからメッセージを取得し、新しい MyRecord インスタンスを作成する静的メソッドを作成します。

```
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
           MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
```

注:

a. 例では、簡潔にするために、MyRecord(BytesMessage) コンストラクターを静的 get メソッドから呼び出しています。 通常は、メッセージの受信を新しい MyRecord インスタンスの作成と分離できます。

10. カスタマー・フィールドをメッセージ・ヘッダーが含まれる JMSBytesMessage に追加する put メソッドを作成します。

```
    public BytesMessage put(MyProducer myProducer) throws JMSException,
           IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                         + DATA_LENGTH + "s",getRecordData())
                         .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
```

注:

a. コード内のメソッド呼び出しは、MyRecord クラス内の属性をメッセージ内のフィールドとしてシリアライズします。

- recordData String 属性にはブランクが埋め込まれ、レコードに定義されている文字セットを使用してバイトに変換されて、RecordData フィールドの長さに切り捨てられます。

11. include ステートメントを追加してクラスを完成させます。

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

タスクの結果

結果:

- TryMyRecord クラスの実行結果を次に示します。
  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用します。

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In  flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 273 CCSID UTF-8
```

  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用しません。

```
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
Out flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID 37 MQ true
In  flags 1 text ABCDEFGHIJKLMNOPQRSTUVWXYZ012345 Encoding 546 CCSID IBM037
```

- メッセージを受信しないように TryMyRecord クラスを変更し、代わりに変更した amqsget0.c サンプルを使用してメッセージを受信した結果は、次のとおりです。 変更したサンプルは、フォーマット済みレコードを受け入れます。 853 ページの『JMS 以外のアプリケーションとのフォーマット済みレコードの交換』の 856 ページの図 156 を参照してください。
  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用します。

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
no more messages
Sample AMQSGET0 end
```

  - コード化文字セット 37 でメッセージを送信し、キュー・マネージャーの変換出口を使用しません。

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

例を別のコード・ページとデータ変換出口で試すには、次のようにします。 Java クラスを作成し、 IBM WebSphere MQ を構成し、メインプログラム TryMyRecord を実行します。 864 ページの図 157 を参照してください。

1. 例を実行するように IBM WebSphere MQ と JMS を構成します。 例を Windows で実行するための手順を示します。

   1. キュー・マネージャーを作成します。

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

   2. キューを作成します。

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

   3. JNDI ディレクトリーを作成します。

```
cd c:\
md JNDI-Directory
```

   4. JMS bin ディレクトリーに切り替えます。
   JMS 管理プログラムをここから実行する必要があります。 パスは *MQ_INSTALLATION_PATH*\java\bin です。

   5. 以下の JMS 定義を JMSQM1Q1.txt というファイルに作成します。

```
DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
VERSION(7)
END
```

6. **JMSAdmin プログラムを実行して JMS リソースを作成します。**

```
JMSAdmin < JMSQM1Q1.txt
```

2. IBM WebSphere MQ エクスプローラーを使用して作成した定義を、作成、変更、および参照できます。
3. TryMyRecord を実行します。

例の実行に使用されるクラス

図 864 ページの図 157 から 868 ページの図 162 にリストされているクラスは、圧縮ファイルでも使用できます。jm25529_.zip または jm25529_.tar.gz をダウンロードしてください。

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

図 *157. TryMyRecord*

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

図 *158. RECORD*

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }Â .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

図 *159. MyRecord*

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                  == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
       }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

図 *160. EndPoint*

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

図 *161. MyProducer*

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

図 *162. MyConsumer*

WebSphere MQ classes for JMS アプリケーションでの接続ファクトリーおよび宛先の作成と構成

WebSphere MQ classes for JMS アプリケーションは、Java Naming and Directory Interface (JNDI) 名前空間から、 IBM JMS 拡張機能を使用して、または WebSphere MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先を管理対象オブジェクトとして取得することにより、それらを作成できます。 また、アプリケーションは IBM JMS 拡張機能または WebSphere MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先のプロパティーを設定できます。

接続ファクトリーおよび宛先は、JMS アプリケーションのロジックの流れの開始点です。 アプリケーションは ConnectionFactory オブジェクトを使用してメッセージング・サーバーへの接続を作成し、Queue または Topic オブジェクトをターゲットとして使用してメッセージを送信したり、ソースとして使用してそこからメッセージを受信します。 そのため、アプリケーションは少なくとも 1 つの接続ファクトリーおよび 1 つ以上の宛先を作成する必要があります。 接続ファクトリーまたは宛先を作成したら、次にアプリケーションは 1 つ以上のプロパティーを設定してオブジェクトを構成する必要があるかもしれません。

要約すれば、アプリケーションは接続ファクトリーおよび宛先を以下の方法で作成し、構成できます。

JNDI を使用した管理対象オブジェクトの取得
管理者は WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使用して、接続ファクトリーおよび宛先を管理対象オブジェクトとして JNDI 名前空間に作成および構成できます。 そうすると、アプリケーションは JNDI 名前空間から管理対象オブジェクトを取得できます。 管理対象オブジェクトを取得したら、アプリケーションは、必要な場合にはそのオブジェクトの 1 つ以上のプロパティーを設定または変更できます。それには、IBM JMS 拡張機能または WebSphere MQ JMS 拡張機能のいずれかを使用します。

IBM JMS 拡張機能の使用
アプリケーションは IBM JMS 拡張機能を使用して、接続ファクトリーおよび宛先を実行時に動的に作成できます。 アプリケーションは最初に JmsFactoryFactory オブジェクトを作成し、次にこのオブジ

ェクトのメソッドを使用して、接続ファクトリーおよび宛先を作成します。 接続ファクトリーまたは宛先を作成したら、アプリケーションは JmsPropertyContext インターフェースから継承したメソッドを使用してそのプロパティーを設定できます。 あるいは、アプリケーションは Uniform Resource Identifier (URI) を使用して、宛先の作成時にその宛先の 1 つ以上のプロパティーを指定できます。

WebSphere MQ JMS 拡張機能の使用
また、アプリケーションは WebSphere MQ JMS 拡張機能を使用して、接続ファクトリーおよび宛先を実行時に動的に作成できます。 アプリケーションは提供されたコンストラクターを使用して、接続ファクトリーおよび宛先を作成します。 接続ファクトリーまたは宛先を作成したら、アプリケーションはオブジェクトのメソッドを使用してそのプロパティーを設定できます。 あるいは、アプリケーションは URI を使用して、宛先の作成時にその宛先の 1 つ以上のプロパティーを指定できます。

JNDI を使用して *JMS* アプリケーションで管理対象オブジェクトを取り出す

Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトを取得するには、JMS アプリケーションで初期コンテキストを作成してから、lookup () メソッドを使用してオブジェクトを取得する必要があります。

アプリケーションで管理対象オブジェクトを JNDI ネーム・スペースから取り出す前に、管理者はまず管理対象オブジェクトを作成する必要があります。 管理者は WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使って JNDI ネーム・スペースで管理対象オブジェクトを作成および保守できます。 WebSphere MQ JMS 管理ツールの使い方については、938 ページの『WebSphere MQ JMS 管理ツールの使用』を参照してください。 WebSphere MQ エクスプローラーの使い方については、WebSphere MQ エクスプローラーに付属するヘルプを参照してください。 ただし、アプリケーション・サーバーは通常、管理オブジェクト用の独自のリポジトリーと、オブジェクトの作成と維持のための独自のツールを提供します。

管理対象オブジェクトを JNDI ネーム・スペースから取り出すには、以下のコードに示されているように、アプリケーションでまず初期コンテキストを作成する必要があります。

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

このコードのストリング変数 `url` および `icf` の意味はそれぞれ次のとおりです。

url
ディレクトリー・サービスの Uniform Resource Locator (URL)。 URL は次のいずれかの形式になります。

- `ldap://`*`hostname`*`/`*`contextName`* (LDAP サーバーに基づくディレクトリー・サービスの場合)
- `file:/`*`directoryPath`* (ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

icf
初期コンテキスト・ファクトリーのクラス名。これは次のいずれかの値になります。

- `com.sun.jndi.ldap.LdapCtxFactory` (LDAP サーバーに基づくディレクトリー・サービスの場合)
- `com.sun.jndi.fscontext.RefFSContextFactory` (ローカル・ファイル・システムに基づくディレクトリー・サービスの場合)

JNDI パッケージと Lightweight Directory Access Protocol (LDAP) サービス・プロバイダーの組み合わせによっては、LDAP エラー 84 が発生する場合があります。 この問題を解決するには、InitialDirContext への呼び出しを行う前に、以下のコード行を挿入してください。

```
environment.put(Context.REFERRAL, "throw");
```

初期コンテキストが取得された後、アプリケーションは、以下の例で示されているように lookup() メソッドを使用して JNDI ネーム・スペースから管理対象オブジェクトを取り出すことができます。

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

このコードは LDAP に基づくネーム・スペースから以下のオブジェクトを取り出します。

- myCF という名前でバインドされる ConnectionFactory オブジェクト
- myQ という名前でバインドされる Queue オブジェクト
- myT という名前でバインドされる Topic オブジェクト

IBM JMS 拡張機能の使用

WebSphere MQ classes for JMS には、IBM JMS 拡張機能という、JMS API に対する機能拡張のセットが含まれています。 アプリケーションはこれらの拡張機能を使用して、実行時に接続ファクトリーおよび宛先を動的に作成し、WebSphere MQ classes for JMS オブジェクトのプロパティーを設定できます。 この拡張機能は、任意のメッセージング・プロバイダーと共に使用できます。

IBM JMS 拡張機能は、以下のパッケージ内の一連のインターフェースおよびクラスです。

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

パッケージは、<MQ_Install_Dir>/java/lib 内の com.ibm.mqjms.jar にあります。

これらの拡張機能は、以下の機能を提供します。

- Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトとして取得するのではなく、実行時に動的に接続ファクトリーおよび宛先を作成するためのファクトリー・ベースのメカニズム
- WebSphere MQ classes for JMS オブジェクトのプロパティーを設定するための一連のメソッド
- 問題に関する詳細情報を入手するためのメソッドがある一連の例外クラス
- トレースを制御するための一連のメソッド
- WebSphere MQ classes for JMS に関するバージョン情報を入手するための一連のメソッド

接続ファクトリーおよび宛先を実行時に動的に作成し、そのプロパティーを設定および取得することに関して、IBM JMS 拡張機能は、WebSphere MQ JMS 拡張機能に対する一連の代替セットのインターフェースを提供します。 ただし、WebSphere MQ JMS 拡張機能は WebSphere MQ メッセージング・プロバイダーに固有ですが、IBM JMS 拡張機能は WebSphere MQ に固有ではなく、802 ページの『階層化アーキテクチャー』で説明されている階層化アーキテクチャー内のどのメッセージング・プロバイダーとでも使用できます。

インターフェース com.ibm.msg.client.wmq.WMQConstants には定数の定義が含まれており、アプリケーションはそれを、IBM JMS 拡張機能を使用する WebSphere MQ classes for JMS オブジェクトのプロパティーの設定時に使用できます。 このインターフェースには、WebSphere MQ メッセージング・プロバイダーの定数と、どのメッセージング・プロバイダーにも依存していない JMS 定数が含まれています。

続くサンプル・コードは、以下のインポート・ステートメントが実行済みであると想定しています。

```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

接続ファクトリーと宛先の作成

アプリケーションが IBM JMS 拡張機能を使用して接続ファクトリーおよび宛先を作成する前に、まず JmsFactoryFactory オブジェクトを作成する必要があります。 以下の例が示すように、JmsFactoryFactory オブジェクトを作成するために、アプリケーションは JmsFactoryFactory クラスの getInstance() メソッドを呼び出します。

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

getInstance() 呼び出しのパラメーターは、WebSphere MQ メッセージング・プロバイダーを、選択したメッセージング・プロバイダーとして識別する定数です。 次いでアプリケーションは JmsFactoryFactory オブジェクトを使用して、接続ファクトリーと宛先を作成できます。

以下の例が示すように、接続ファクトリーを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createConnectionFactory() メソッドを呼び出します。

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsConnectionFactory オブジェクトを作成します。これはアプリケーションがバインディング・モードでデフォルトのキュー・マネージャーに接続することを意味します。 アプリケーションをクライアント・モードで接続したいか、またはデフォルトのキュー・マネージャー以外のキュー・マネージャーに接続したい場合、接続を作成する前に、アプリケーションは JmsConnectionFactory オブジェクトの適切なプロパティーを設定する必要があります。 これを行う方法については、872 ページの『WebSphere MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

JmsFactoryFactory クラスには、以下のタイプの接続ファクトリーを作成するためのメソッドも含まれています。

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- JmsXAConnectionFactory
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

以下の例が示すように、Queue オブジェクトを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createQueue() メソッドを呼び出します。

```
JmsQueue q1 = ff.createQueue("Q1");
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsQueue オブジェクトを作成します。 オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる WebSphere MQ キューを表します。 このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

createQueue() メソッドは、キュー Uniform Resource Identifier (URI) をパラメーターとして受け入れることもできます。 キュー URI は、WebSphere MQ キューの名前と、オプションで、キューを所有するキュー・マネージャーの名前、および JmsQueue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

このステートメントによって作成される JmsQueue オブジェクトは、キュー・マネージャー QM2 が所有する Q2 という WebSphere MQ キューを表します。この宛先に送信されるすべてのメッセージは永続的で、優先順位は 5 です。 キューの URI について詳しくは、884 ページの『Uniform Resource Identifier (URI)』を参照してください。 JmsQueue オブジェクトのプロパティーの別の設定方法については、872 ページの『WebSphere MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

以下の例が示すように、Topic オブジェクトを作成するために、アプリケーションは JmsFactoryFactory オブジェクトの createTopic() メソッドを使用できます。

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

このステートメントは、そのすべてのプロパティーにデフォルト値を指定して JmsTopic オブジェクトを作成します。 オブジェクトは Sport/Football/Results という名前のトピックを表します。

さらに、createTopic() メソッドは、トピック URI をパラメーターとして受け入れることもできます。 トピック URI は、トピックの名前と、オプションで JmsTopic オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントには、トピック URI の例が含まれています。

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

これらのステートメントによって作成される JmsTopic オブジェクトは、Sport/Tennis/Results というトピックを表し、この宛先に送信されるすべてのメッセージは非永続的なもので、優先順位 0 を持ちます。 トピック URI について詳しくは、884 ページの『Uniform Resource Identifier (URI)』を参照してください。 JmsTopic オブジェクトのプロパティーの別の設定方法については、872 ページの『WebSphere MQ classes for JMS オブジェクトのプロパティーの設定』を参照してください。

アプリケーションが接続ファクトリーまたは宛先を作成した後は、そのオブジェクトは選択されたメッセージング・プロバイダーだけでしか使用できません。

WebSphere MQ classes for JMS オブジェクトのプロパティーの設定

WebSphere MQ classes for JMS オブジェクトのプロパティーを IBM JMS 拡張機能を使用して設定するには、アプリケーションは com.ibm.msg.client.JmsPropertyContext インターフェースのメソッドを使用します。

Java データ型ごとに、 JmsPropertyContext インターフェースには、そのデータ型のプロパティーの値を設定するメソッドと、そのデータ型のプロパティーの値を取得するメソッドが含まれています。 例えば、アプリケーションは setIntProperty() メソッドを呼び出して、整数値でプロパティーを設定し、getIntProperty() メソッドを呼び出して、整数値でプロパティーを取得します。

さらに、com.ibm.mq.jms パッケージのクラスのインスタンスは、JmsPropertyContext インターフェースのメソッドを継承します。 したがってアプリケーションは、これらのメソッドを使用して、MQConnectionFactory、MQQueue、および MQTopic の各オブジェクトのプロパティーを設定できます。

アプリケーションが WebSphere MQ classes for JMS オブジェクトを作成するときに、プロパティーはデフォルト値で自動的に設定されます。 アプリケーションがプロパティーを設定するときは、新規の値がプロパティーの以前の値を置き換えます。 プロパティーが設定された後には、それを削除することはできませんが、その値を変更することはできます。

アプリケーションがプロパティーに無効な値を設定しようとした場合、WebSphere MQ classes for JMS は JMSException 例外をスローします。 アプリケーションが、設定されていないプロパティーを取得しようとした場合、その動作は JMS 仕様に記述されているとおりになります。 WebSphere MQ classes for JMS は、プリミティブ・データ・タイプに対して NumberFormatException 例外をスローし、参照されるデータ・タイプに対してヌルを戻します。

WebSphere MQ classes for JMS オブジェクトの定義済みプロパティーに加え、アプリケーションはその固有のプロパティーを設定できます。 これらのアプリケーションで定義されたプロパティーは、WebSphere MQ classes for JMS により無視されます。

WebSphere MQ classes for JMS オブジェクトのプロパティーの詳細については、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーを参照してください。

以下のコードは、IBM JMS 拡張機能を使用してプロパティーを設定する方法の例です。 このコードは、接続ファクトリーの 5 つのプロパティーを設定します。

```
factory.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE,
                       WMQConstants.WMQ_CM_CLIENT);
factory.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
factory.setStringProperty(WMQConstants.WMQ_HOST_NAME, "HOST1");
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
factory.setStringProperty(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setStringProperty(WMQConstants.WMQ_APPLICATIONNAME, "My Application");
```

それらのプロパティーを設定すると、アプリケーションは QM1.SVR という名前の MQI チャネルを使用して、クライアント・モードでキュー・マネージャー QM1 に接続します。 キュー・マネージャーは HOST1 というホスト名を持つシステムで稼働し、キュー・マネージャーのリスナーはポート番号 1415 で listen します。 この接続およびその下にあるセッションに関連付けられた他のキュー・マネージャー接続には、アプリケーション名「My Application」が関連付けられます。

注 : z/OS プラットフォーム上で実行するキュー・マネージャーでは、アプリケーション名の設定をサポートしていないため、この設定は無視されます。

JmsPropertyContext インターフェースには setObjectProperty() メソッドも含まれており、アプリケーションはそれをプロパティーの設定に使用できます。 メソッドの 2 番目のパラメーターは、プロパティーの値をカプセル化するオブジェクトです。 例えば、以下のコードは整数 1415 をカプセル化する整数オブジェクトを作成し、次いで setObjectProperty() を呼び出して、接続ファクトリーの PORT プロパティーを値 1415 に設定します。

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

したがってこのコードは、以下のステートメントと同等です。

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

それとは反対に、getObjectProperty() メソッドは、プロパティーの値をカプセル化するオブジェクトを戻します。

プロパティー値のデータ型の暗黙的な変換

アプリケーションが JmsPropertyContext インターフェースのメソッドを使用して、WebSphere MQ classes for JMS オブジェクトのプロパティーを設定または取得する場合、プロパティーの値は、あるデータ・タイプから別のデータ・タイプに暗黙に変換できます。

例えば、以下のステートメントは、JmsQueue オブジェクト q1 の PRIORITY プロパティーを設定します。

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

PRIORITY プロパティーには整数値が指定されているため、setStringProperty() 呼び出しは、ストリング "5" (ソース値) を整数 5 (ターゲット値) に暗黙的に変換し、それが PRIORITY プロパティーの値になります。

これとは逆に、以下のステートメントは、JmsQueue オブジェクト q1 の PRIORITY プロパティーを取得します。

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

整数 5 (ソース値) は、PRIORITY プロパティーの値です。これは、getStringProperty() 呼び出しによって暗黙的にストリング "5" (ターゲット値) に変換されます。

WebSphere MQ classes for JMS によりサポートされる変換は、873 ページの表 122 に示されています。

表 *122.* あるデータ型から別のデータ型への、サポートされる変換

ソースのデータ型	**サポートされているターゲット・データ・タイプ**
boolean	ストリング
byte	int、long、short、String
文字	ストリング
double	ストリング
float	double、String
int	long、String

表 122. あるデータ型から別のデータ型への、サポートされる変換 (続き)	
ソースのデータ型	**サポートされているターゲット・データ・タイプ**
long	ストリング
不足	int、long、String
ストリング	boolean、byte、double、float、int、long、short

サポートされる変換を制御する一般規則は、以下のとおりです。

- 変換中に失われるデータがない場合には、数値をあるデータ型から別のデータ型に変換できます。 例えば、データ型 `int` の値は、データ型 `long` の値に変換できますが、データ型 `short` の値には変換できません。
- 任意のデータ型の値を、ストリングに変換できます。
- 変換に適正な形式であれば、ストリングは任意の他のデータ・タイプ (`char` を除く) の値に変換できます。 アプリケーションが適正な形式でないストリングを変換しようとした場合、WebSphere MQ classes for JMS は NumberFormatException 例外をスローします。
- アプリケーションが、サポートされていない変換をしようとした場合、WebSphere MQ classes for JMS は MessageFormatException 例外をスローします。

あるデータ型から別のデータ型に値を変換するための特定の規則は、以下のとおりです。

- ブール値をストリングに変換する場合、値 `true` はストリング "true" に変換され、値 `false` はストリング "false" に変換されます。
- ストリングをブール値に変換する場合、ストリング "true" (大/小文字の区別をしない) は `true` に変換され、ストリング "false" (大/小文字の区別をしない) は `false` に変換されます。 それ以外のストリングは `false` に変換されます。
- ストリングをデータ型 `byte`、`int`、`long`、または `short` の値に変換する場合、ストリングは以下の形式でなければなりません。

  [*blanks*][*sign*]*digits*

  ストリングの構成要素の意味は以下のとおりです。

  blanks
  オプションの先行ブランク文字。

  sign
  オプションの正符号 (+) または負符号 (-)。

  digits
  数字 (0 から 9) の連続シーケンス。 少なくとも 1 つの数字がなければなりません。

  数字のシーケンスの後に、ストリングには数字ではない他の文字を含めることができますが、これらの文字の先頭に達するとすぐに変換は停止します。 ストリングは 10 進整数を表すと想定されます。

  ストリングが適正な形式でない場合、WebSphere MQ classes for JMS は NumberFormatException 例外をスローします。
- ストリングをデータ型 `double` または `float` の値に変換する場合、ストリングは以下の形式でなければなりません。

  [*blanks*][*sign*]*digits*[*e_char*[*e_sign*]*e_digits*]

  ストリングの構成要素の意味は以下のとおりです。

  blanks
  オプションの先行ブランク文字。

  sign
  オプションの正符号 (+) または負符号 (-)。

digits
数字 (0 から 9) の連続シーケンス。 少なくとも 1 つの数字がなければなりません。

e_char
指数文字。*E* または *e* のいずれかです。

e_sign
指数用のオプションの正符号 (+) または負符号 (-)。

e_digits
指数用の数字 (0 から 9) の連続シーケンス。 ストリングに指数文字が含まれている場合は、少なくとも 1 つの数字がなければなりません。

数字のシーケンス、または指数を表すオプション文字の後に、ストリングには数字ではない他の文字を含めることができますが、これらの文字の先頭に達するとすぐに変換は停止します。 ストリングは、10 の累乗の指数を持つ 10 進浮動小数点数を表すと想定されます。

ストリングが適正な形式でない場合、WebSphere MQ classes for JMS は NumberFormatException 例外をスローします。

- 数値 (データ・タイプが byte の値を含む) をストリングに変換する場合、値は 10 進数としての値のストリング表現に変換され、その値の ASCII 文字を含むストリングに変換されるのではありません。 例えば、整数 65 はストリング "65" に変換され、ストリング "A" に変換されるのではありません。

単一呼び出しでの複数のプロパティーの設定

JmsPropertyContext インターフェースには setBatchProperties() メソッドも含まれており、アプリケーションはそれを単一呼び出しでの複数のプロパティーの設定に使用できます。 メソッドのパラメーターは、一連のプロパティーの名前と値のペアをカプセル化する Map オブジェクトです。

例えば、以下のコードは setBatchProperties() メソッドを使用して、872 ページの『WebSphere MQ classes for JMS オブジェクトのプロパティーの設定』に示す接続ファクトリーの同じ 5 つのプロパティーを設定します。 このコードは、Map インターフェースを実装する HashMap クラスのインスタンスを作成します。

```
HashMap batchProperties = new HashMap();
batchProperties.put(WMQConstants.WMQ_CONNECTION_MODE,
                    new Integer(WMQConstants.WMQ_CM_CLIENT));
batchProperties.put(WMQConstants.WMQ_QUEUE_MANAGER, "QM1");
batchProperties.put(WMQConstants.WMQ_WMQ_HOST_NAME, "HOST1");
batchProperties.put(WMQConstants.WMQ_PORT, "1414");
batchProperties.put(WMQConstants.WMQ_CHANNEL, "QM1.SVR");
factory.setBatchProperties(batchProperties);
```

Map.put() メソッドの 2 番目のパラメーターはオブジェクトでなければならないことに注意してください。 したがって、例で示すとおり、プリミティブ・データ型のプロパティー値は、オブジェクト内でカプセル化するか、またはストリングによって表現されなければなりません。

setBatchProperties() メソッドは、各プロパティーを妥当性検査します。 setBatchProperties() メソッドが、例えばその値が無効であるなどの理由でプロパティーを設定できない場合、指定されたどのプロパティーも設定されません。

プロパティーの名前と値

アプリケーションが、JmsPropertyContext インターフェースのメソッドを使用して WebSphere MQ classes for JMS オブジェクトのプロパティーを設定および取得する場合、アプリケーションは以下のいずれかの方法でプロパティーの名前と値を指定できます。 記載されている各例は、JmsQueue オブジェクト q1 の PRIORITY プロパティーを設定し、キューに送信されるメッセージが、send() 呼び出しに指定された優先順位を持つようにする方法を示しています。

com.ibm.msg.client.wmq.WMQConstants インターフェースに定数として定義されたプロパティーの名前と値の使用

以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

キューおよびトピック Uniform Resource Identifier (URI) で使用できるプロパティーの名前と値の使用
以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setIntProperty("priority", -2);
```

宛先のプロパティーの名前と値だけがこの方法で指定できます。

WebSphere MQ JMS 管理ツールにより認識されるプロパティーの名前と値の使用
以下のステートメントは、この方法でプロパティーの名前と値を指定する方法の例です。

```
q1.setStringProperty("PRIORITY", "APP");
```

以下のステートメントで示すとおり、簡易形式のプロパティー名も受け入れることができます。

```
q1.setStringProperty("PRI", "APP");
```

アプリケーションがプロパティーを取得するときに戻される値は、アプリケーションがプロパティーの名前を指定する方法に応じて異なります。 例えば、アプリケーションが定数 WMQConstants.WMQ_PRIORITY をプロパティー名として指定する場合、戻される値は整数 -2 です。

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

アプリケーションがストリング "priority" をプロパティー名として指定すると、同じ値が戻されます。

```
int n2 = getIntProperty("priority");
```

ただし、アプリケーションがストリング "PRIORITY" または "PRI" をプロパティー名として指定する場合、戻される値はストリング "APP" です。

```
String s1 = getStringProperty("PRI");
```

内部的に、WebSphere MQ classes for JMS は、プロパティーの名前と値を、com.ibm.msg.client.wmq.WMQConstants インターフェースで定義されるリテラル値として保管します。 これはプロパティーの名前と値の定義された正規形式です。 一般規則として、アプリケーションがプロパティーの名前と値を指定する他の 2 つの方法のいずれかを使用してプロパティーを設定する場合、WebSphere MQ classes for JMS は、指定された入力形式からの名前と値を、この正規形式に変換する必要があります。 同様に、アプリケーションがプロパティーの名前と値を指定する他の 2 つの方法のいずれかを使用してプロパティーを取得する場合、WebSphere MQ classes for JMS は、指定された入力形式からの名前を正規形式に変換し、正規形式からの値を必要な出力形式に変換する必要があります。 これらの変換を実行する必要があることには、パフォーマンス上の含意があります。

トレース・ファイル内、または WebSphere MQ classes for JMS ログ内の、例外により戻されるプロパティーの名前と値は、必ず正規形式になります。

Map インターフェースの使用

JmsPropertyContext インターフェースは java.util.Map インターフェースを拡張します。 したがって、アプリケーションは Map インターフェースのメソッドを使用して、WebSphere MQ classes for JMS オブジェクトのプロパティーにアクセスできます。

例えば、以下のコードは、接続ファクトリーのすべてのプロパティーの名前と値を印刷します。 このコードは、Map インターフェースのメソッドだけを使用して、プロパティーの名前と値を取得します。

```
// Get the names of all the properties
Set propNames = factory.keySet();

// Loop round all the property names and get the property values
Iterator iterator = propNames.iterator();
while (iterator.hasNext()){
    String pName = (String)iterator.next();
    System.out.println(pName+"="+factory.get(pName));
}
```

Map インターフェースのメソッドを使用しても、プロパティーの妥当性検査または変換はバイパスされません。

WebSphere MQ JMS 拡張機能の使用

WebSphere MQ classes for JMS には、WebSphere MQ JMS 拡張機能と呼ばれる JMS API に対する拡張機能のセットが含まれています。 アプリケーションはこれらの拡張機能を使用して接続ファクトリーや宛先を実行時に動的に作成したり、接続ファクトリーや宛先のプロパティーを設定したりすることができます。

WebSphere MQ classes for JMS のパッケージ com.ibm.jms および com.ibm.mq.jms にはクラスのセットが入っています。 それらのクラスは JMS インターフェースを実装し、WebSphere MQ JMS 拡張機能を格納しています。 以下のコード例は、次のステートメントによってそれらのパッケージがインポート済みであることを前提としています。

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
```

アプリケーションは WebSphere MQ JMS 拡張機能を使用して、以下の機能を実行することができます。

- Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトとして取得する代わりに、実行時に接続ファクトリーおよび宛先を動的に作成する
- 接続ファクトリーおよび宛先のプロパティーを設定する

接続ファクトリーの作成

接続ファクトリーを作成するために、アプリケーションは以下の例で示されているように MQConnectionFactory コンストラクターを使用できます。

```
MQConnectionFactory factory = new MQConnectionFactory();
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQConnectionFactory オブジェクトを作成します。これは、アプリケーションがバインディング・モードでデフォルトのキュー・マネージャーに接続することを意味します。 アプリケーションをクライアント・モードで接続する場合、またはデフォルト以外のキュー・マネージャーに接続する場合、アプリケーションは接続を作成する前に MQConnectionFactory オブジェクトの適切なプロパティーを設定する必要があります。 これを行う方法については、877 ページの『接続ファクトリーのプロパティーの設定』を参照してください。

アプリケーションは、同様の方法で以下のタイプの接続ファクトリーを作成することができます。

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

接続ファクトリーのプロパティーの設定

アプリケーションは、接続ファクトリーの適切なメソッドを呼び出すことにより、接続ファクトリーのプロパティーを設定することができます。 接続ファクトリーは、管理対象オブジェクトまたは実行時に動的に作成されたオブジェクトのいずれかです。

例えば、次のようなコードがあるとします。

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

このコードは MQConnectionFactory オブジェクトを作成した後、オブジェクトの 5 つのプロパティーを設定します。 それらのプロパティーを設定すると、アプリケーションは QM1.SVR という名前の MQI チャネ

ルを使用してクライアント・モードでキュー・マネージャー QM1 に接続します。 キュー・マネージャーは HOST1 というホスト名を持つシステムで稼働し、キュー・マネージャーのリスナーはポート番号 1415 で listen します。

ブローカーとのリアルタイム接続の場合、アプリケーションは次のコードを使用できます。

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_DIRECT);
factory.setHostName("HOST2");
factory.setPort(1507);
```

このコードは、ブローカーが HOST2 というホスト名を持つシステムで稼働し、ポート番号 1507 で listen していることを前提としています。

ブローカーとのリアルタイム接続を使用するアプリケーションは、パブリッシュ/サブスクライブ・スタイルのメッセージングのみ使用できます。 Point-to-Point スタイルのメッセージングは使用できません。

特定の接続ファクトリーのプロパティーの組み合わせのみ有効です。 有効な組み合わせについては、WebSphere MQ classes for JMS オブジェクトのプロパティー間の依存関係 を参照してください。

接続ファクトリーのプロパティー、およびそのプロパティーの設定に使用するメソッドについて詳しくは、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーを参照してください。

宛先の作成

Queue オブジェクトを作成するために、アプリケーションは以下の例で示されているように MQQueue コンストラクターを使用できます。

```
MQQueue q1 = new MQQueue("Q1");
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQQueue オブジェクトを作成します。オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる WebSphere MQ キューを表します。 このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

MQQueue コンストラクターの代替形式は、以下の例に示されているように、2 つのパラメーターを持ちます。

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

このステートメントによって作成される MQQueue オブジェクトは、キュー・マネージャー QM2 が所有する Q2 という名前の WebSphere MQ キューを表します。 このようにして識別されるキュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合、アプリケーションがメッセージをこの宛先に送信するときに、WebSphere MQ がメッセージをローカル・キュー・マネージャーからリモート・キュー・マネージャーに経路指定できるように構成する必要があります。

MQQueue コンストラクターは単一パラメーターとしてキュー Uniform Resource Identifier (URI) を受け入れることもできます。 キュー URI は、WebSphere MQ キューの名前と、オプションで、キューを所有するキュー・マネージャーの名前、および MQQueue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

このステートメントで作成される MQQueue オブジェクトは、キュー・マネージャー QM3 が所有する Q3 という WebSphere MQ キューを表します。この宛先に送信されるすべてのメッセージは永続的で、優先順位は 5 になります。 キュー URI について詳しくは、884 ページの『Uniform Resource Identifier (URI)』を参照してください。 MQQueue オブジェクトのプロパティーを設定する別の方法については、879 ページの『宛先のプロパティーの設定』を参照してください。

アプリケーションは、以下の例で示されているように MQTopic コンストラクターを使用して Topic オブジェクトを作成することができます。

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

このステートメントは、すべてそのプロパティーのデフォルト値で MQTopic オブジェクトを作成します。オブジェクトは Sport/Football/Results という名前のトピックを表します。

MQTopic コンストラクターはパラメーターとしてトピック URI を受け入れることもできます。 トピック URI は、トピックの名前と、オプションで MQTopic オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはトピック URI の例が含まれています。

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

このステートメントによって作成される MQTopic オブジェクトは、Sport/Tennis/Results というトピックを表し、この宛先に送信されるすべてのメッセージは非永続で、優先順位 0 を持ちます。 トピックの URI について詳しくは、884 ページの『Uniform Resource Identifier (URI)』を参照してください。 MQTopic オブジェクトのプロパティーを設定する別の方法については、879 ページの『宛先のプロパティーの設定』を参照してください。

宛先のプロパティーの設定

アプリケーションは、宛先の適切なメソッドを呼び出すことにより、宛先のプロパティーを設定することができます。 宛先は、管理対象オブジェクトまたは実行時に動的に作成されたオブジェクトのいずれかです。

例えば、次のようなコードがあるとします。

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

このコードは MQQueue オブジェクトを作成した後、オブジェクトの 2 つのプロパティーを設定します。それらのプロパティーを設定すると、その宛先に送信されるメッセージはすべて永続メッセージとなり、優先順位は 5 になります。

アプリケーションは、以下の例で示されているとおり、同様の方法で MQTopic オブジェクトのプロパティーを設定することができます。

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

このコードは MQTopic オブジェクトを作成した後、オブジェクトの 2 つのプロパティーを設定します。 それらのプロパティーを設定すると、その宛先に送信されるメッセージはすべて非永続メッセージとなり、優先順位は 0 になります。

宛先のプロパティー、およびそのプロパティーの設定に使用するメソッドについて詳しくは、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーを参照してください。

JMS アプリケーションでの接続の構築

接続を構築するために、JMS アプリケーションは ConnectionFactory オブジェクトを使用して Connection オブジェクトを作成した後、接続を開始します。

アプリケーションは、以下の例で示されているように ConnectionFactory オブジェクトの createConnection() メソッドを使用して Connection オブジェクトを作成します。

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

JMS 接続が作成されると、 IBM WebSphere MQ classes for JMS は接続ハンドル (Hconn) を作成し、キュー・マネージャーとの会話を開始します。

QueueConnectionFactory インターフェースと TopicConnectionFactory インターフェースはそれぞれ、ConnectionFactory インターフェースから createConnection() メソッドを継承しています。 このため、以下の例で示されているように createConnection() メソッドを使用してドメイン特定のオブジェクトを作成することができます。

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

このコード断片は、QueueConnection オブジェクトを作成します。 アプリケーションは、このオブジェクト上でドメイン独立の操作、または Point-to-Point ドメインにのみ適用される操作を実行できます。 ただし、アプリケーションがパブリッシュ/サブスクライブ・ドメインに対してのみ適用される操作を実行しようとすると、IllegalStateException 例外が以下のメッセージと一緒にスローされます。

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

その理由は、接続がドメイン特定の接続ファクトリーにより作成されているためです。

注 : アプリケーション・プロセス ID は、キュー・マネージャーに渡されるデフォルト・ユーザー ID として使用されることに注意してください。 アプリケーションがクライアント・トランスポート・モードで実行される場合は、このプロセス ID がサーバー上に存在し、関係する許可がプロセス ID に与えられている必要があります。 別の ID を使用する必要がある場合は、createConnection(username, password) メソッドを使用してください。

JMS 仕様では、接続は stopped 状態で作成されることが規定されています。 接続が開始するまで、接続に関連付けられたメッセージ・コンシューマーはメッセージを受信できません。 アプリケーションは、以下の例で示されているように Connection オブジェクトの start() メソッドを使用して接続を開始します。

```
connection.start();
```

JMS アプリケーションでのセッションの作成

セッションを作成するために、JMS アプリケーションは Connection オブジェクトの createSession() メソッドを使用します。

createSession() メソッドは 2 つのパラメーターを持ちます。

1. セッションがトランザクション化されているかどうかを指定するパラメーター
2. セッションの確認応答モードを指定するパラメーター

例えば、以下のコードは、トランザクション化されていないセッションを作成し、確認応答モードは AUTO_ACKNOWLEDGE です。

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

JMS セッションが作成されると、 IBM WebSphere MQ classes for JMS は接続ハンドル (Hconn) を作成し、キュー・マネージャーとの会話を開始します。

Session オブジェクト、およびそこから作成された MessageProducer または MessageConsumer オブジェクトはすべて、マルチスレッド・アプリケーションの別のスレッドで並行して使用することができません。それらのオブジェクトが並行して使用されないようにするための最も簡単な方法は、各スレッドごとに別個の Session オブジェクトを作成することです。

JMS アプリケーションでのトランザクション化されたセッション

JMS アプリケーションは、最初にトランザクション化されたセッションを作成することによって、ローカル・トランザクションを実行できます。 アプリケーションでは、トランザクションのコミットとロールバックが可能です。

JMS アプリケーションは、ローカル・トランザクションを実行できます。 ローカル・トランザクションとは、アプリケーションが接続されているキュー・マネージャーのリソースにのみに対して行われた変更が含まれるトランザクションのことです。 ローカル・トランザクションを実行するには、まずアプリケーションは、Connection オブジェクトの createSession() メソッドを呼び出すことでトランザクション化セッションを作成し、セッションがトランザクション化されるパラメーターとして指定する必要があります。 その後、そのセッション内で送受信されたすべてのメッセージは、トランザクションの順序でグループ化されます。 トランザクションは、トランザクションが開始されてから送受信したメッセージがアプリケーションでコミットまたはロールバックされると終了します。

トランザクションをコミットするには、アプリケーションで Session オブジェクトの commit() メソッドを呼び出します。 トランザクションがコミットされると、そのトランザクション内に送信されたすべてのメッセージは、他のアプリケーションに配信できるようになります。また、そのトランザクション内に受信したすべてのメッセージが認知されるので、メッセージング・サーバーはそれらのメッセージをアプリケーションへ再配信しなくなります。 また、Point-to-Point ドメインでは、受信したメッセージがメッセージング・サーバーのキューからも除去されます。

トランザクションをロールバックするには、アプリケーションで Session オブジェクトの rollback() メソッドを呼び出します。 トランザクションがロールバックされると、そのトランザクション内に送信されたすべてのメッセージはメッセージング・サーバーによって破棄されます。また、そのトランザクション内に受信したすべてのメッセージは再配信できるようになります。 Point-to-Point ドメインでは、受信されたメッセージはキューに書き戻され、再び他のアプリケーションから見えるようになります。

アプリケーションがトランザクション化されたセッションを作成するか、commit() または rollback() メソッドを呼び出すと、自動的に新しいトランザクションが開始されます。 したがって、トランザクション化されたセッションには常にアクティブなトランザクションが含まれます。

アプリケーションがトランザクション化されたセッションを閉じると、暗黙的なロールバックが行われます。 アプリケーションが接続を閉じると、その接続のトランザクション化されたセッションすべてで暗黙的なロールバックが行われます。

アプリケーションが接続を閉じずに終了した場合も、その接続のトランザクション化されたセッションすべてで暗黙のロールバックが行われます。

トランザクションは、トランザクション化されたセッションに完全に包含されています。 トランザクションがセッションをまたぐことはできません。 つまり、アプリケーションは、トランザクション化された複数のセッションの中でメッセージを送受信したり、これらのすべてのアクションを単一のトランザクションとしてコミットまたはロールバックしたりすることはできません。

JMS セッションの確認応答モード

トランザクション化されないすべてのセッションには、アプリケーションによって受信されたメッセージをどのように確認するかを決定する、確認応答モードが存在します。 使用可能な確認応答モードは 3 つあり、どの確認応答モードを選択するかはアプリケーションの設計に影響を与えます。

セッションがトランザクション化されない場合、アプリケーションが受信するメッセージを確認する方法は、セッションの確認応答モードによって決定されます。 以下の部分では、3 つの確認応答モードについて説明します。

AUTO_ACKNOWLEDGE

セッションは、アプリケーションが受信した各メッセージを自動的に確認します。

メッセージがアプリケーションに同期化して配信されると、セッションは、Receive 呼び出しが正常に完了するたびにメッセージの受信を確認します。 メッセージが非同期的に送達された場合は、セッションは、メッセージ・リスナーの onMessage() メソッドの呼び出しが正常に完了するたびにメッセージの受信を確認します。

アプリケーションがメッセージを正常に受信しても、障害によって確認が行えない場合は、そのメッセージは再び送達可能になります。 このため、アプリケーションは、再送達されたメッセージを扱うことができなければなりません。

DUPS_OK_ACKNOWLEDGE

セッションは、メッセージ選択時にアプリケーションが受信したメッセージを確認します。

この確認応答モードを使用すると、セッションで行わなければならない作業の量を減らすことができますが、障害によってメッセージの確認ができなかったときは、複数のメッセージが再び送達可能になる可能性があります。 このため、アプリケーションは、再送達されたメッセージを扱うことができなければなりません。

制約事項 : AUTO_ACKNOWLEDGE および DUPS_OK_ACKNOWLEDGE モードでは、JMS は、メッセージ・リスナーで処理できない例外のアプリケーションによるスローをサポートしません。 これは、メッセージ・リスナーの処理が正常に行われたかどうかに関係なく (ただし、障害がいずれも致命的なものではなく、アプリケーションの続行を妨げるものでない限り)、メッセージ・リスナーから処理が戻ると、必ずメッセージが確認されることを意味します。 メッセージの確認をより細かく制御する必要がある場合は、CLIENT_ACKNOWLEDGE またはトランザクション化モードを使用します。これらのモードを使用すれば、確認の機能をアプリケーションから完全に制御できます。

CLIENT_ACKNOWLEDGE

Message クラスの Acknowledge メソッドを呼び出すことにより、受信したメッセージをアプリケーションが確認します。

アプリケーションは各メッセージの受信を個々に確認するか、または複数のメッセージを一括して受信し、受信した最後のメッセージに対してのみ Acknowledge メソッドを呼び出すことができます。 Acknowledge メソッドが呼び出されると、このメソッドの前回の呼び出し以降に受信したすべてのメッセージが確認されます。

これらの確認応答モードのいずれかと組み合わせることにより、アプリケーションは Session クラスの Recover メソッドを呼び出してセッションでメッセージの送達を停止したり、再開させたりすることができます。 受信されたが前回は未確認だったメッセージについては、再送達されます。 ただし、前回送達されたときと同じシーケンスで送達されるとは限りません。 これらのメッセージが再送達されるまでの間に、より優先順位の高いメッセージが届いている可能性もありますし、オリジナルのメッセージの一部が有効期限切れになっている場合もあります。 Point-to-Point ドメインの場合は、オリジナルのメッセージの一部が別のアプリケーションによって消費されている可能性もあります。

アプリケーションでは、メッセージの JMSRedelivered ヘッダー・フィールドの内容を調べることによって、メッセージが再送達中かどうかを確認できます。 アプリケーションでこれを行うには、Message クラスの getJMSRedelivered() メソッドを呼び出します。

JMS アプリケーションでの宛先の作成

JMS アプリケーションは、Java Naming and Directory Interface (JNDI) 名前空間から管理対象オブジェクトとして宛先を取得する代わりに、セッションを使用して実行時に動的に宛先を作成することができます。 アプリケーションは URI (Uniform Resource Identifier) を使用して WebSphere MQ キューまたはトピックを識別し、オプションで、Queue または Topic オブジェクトの 1 つ以上のプロパティーを指定することができます。

セッションを使用した Queue オブジェクトの作成

以下の例が示すように、Queue オブジェクトを作成するために、アプリケーションは Session オブジェクトの createQueue() メソッドを使用できます。

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

このコードは、すべてのプロパティーにデフォルト値が指定された Queue オブジェクトを作成します。 オブジェクトは、ローカル・キュー・マネージャーに属する Q1 と呼ばれる WebSphere MQ キューを表します。 このキューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

createQueue() メソッドは、キュー URI もパラメーターとして受け入れます。 キュー URI は、WebSphere MQ キューの名前を指定し、オプションで、キューを所有するキュー・マネージャーの名前、および Queue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 以下のステートメントにはキュー URI の例が含まれています。

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

このステートメントによって作成される Queue オブジェクトは、 QM2 というキュー・マネージャーが所有する Q2 という WebSphere MQ キューを表します。この宛先に送信されるすべてのメッセージは永続的であり、優先順位は 5 になります。 このようにして識別されるキュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合、WebSphere MQ は、アプリケーションがメッセージをこの宛先に送信するときに Websphere MQ がメッセージをローカル・キュー・マネージャーからキュー・マネージャー QM2 に経路指定できるように構成する必要があります。 URI の詳細については、884 ページの『Uniform Resource Identifier (URI)』を参照してください。

createQueue() メソッドのパラメーターには、プロバイダー固有の情報が含まれていることに注意してください。 したがって、Queue オブジェクトを管理対象オブジェクトとして JNDI ネーム・スペースから取り出す代わりに、createQueue() メソッドを使用して Queue オブジェクトを作成すると、アプリケーションの移植性は低くなる場合があります。

以下の例が示すように、アプリケーションは Session オブジェクトの createTemporaryQueue() メソッドを使用して TemporaryQueue オブジェクトを作成できます。

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

セッションは一時キューを作成するために使用されますが、一時キューの有効範囲は、セッションを作成するために使用された接続です。 接続のどのセッションでも、一時キューのためのメッセージ・プロデューサーおよびメッセージ・コンシューマーを作成できます。 一時キューは、接続が終了するまで、またはアプリケーションが TemporaryQueue.delete() メソッドを使用して一時キューを明示的に削除するまで、そのどちらかが起きるまで存在します。

アプリケーションが一時キューを作成すると、WebSphere MQ classes for JMS は、アプリケーションの接続先になるキュー・マネージャーに動的キューを作成します。 接続ファクトリーの TEMPMODEL プロパティーは、動的キューを作成するために使用するモデル・キューの名前を指定し、接続ファクトリーの TEMPQPREFIX プロパティーは、動的キューの名前を形成するために使用する接頭部を指定します。

セッションを使用した Topic オブジェクトの作成

以下の例が示すように、Topic オブジェクトを作成するために、アプリケーションは Session オブジェクトの createTopic() メソッドを使用できます。

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

このコードは、すべてのプロパティーにデフォルト値が指定された Topic オブジェクトを作成します。 オブジェクトは Sport/Football/Results という名前のトピックを表します。

さらに、createTopic() メソッドはトピック URI をパラメーターとして受け入れます。 トピック URI は、トピックの名前と、オプションで 1 つ以上の Topic オブジェクトのプロパティーを指定するストリングです。以下のコードには、トピック URI の例が含まれています。

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

このコードによって作成されるトピック・オブジェクトは、「Sport/Tennis/Results」というトピックを表し、この宛先に送信されるすべてのメッセージは非永続で、優先順位 0 を持ちます。 トピックの URI について詳しくは、884 ページの『Uniform Resource Identifier (URI)』を参照してください。

createTopic() メソッドのパラメーターには、プロバイダー固有の情報が含まれていることに注意してください。 したがって、Topic オブジェクトを管理対象オブジェクトとして JNDI ネーム・スペースから取り出す代わりに、createTopic() メソッドを使用して Topic オブジェクトを作成すると、アプリケーションの移植性は低くなる場合があります。

以下の例が示すように、アプリケーションは Session オブジェクトの createTemporaryTopic() メソッドを使用して TemporaryTopic オブジェクトを作成できます。

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

セッションは一時トピックを作成するために使用されますが、一時トピックの有効範囲は、セッションを作成するために使用された接続です。 接続のどのセッションでも、一時トピックのためのメッセージ・プロデューサーおよびメッセージ・コンシューマーを作成できます。 一時トピックは、接続が終了するまで、またはアプリケーションが TemporaryTopic.delete() メソッドを使用して一時トピックを明示的に削除するまで、そのどちらかが起きるまで存在します。

アプリケーションが一時トピックを作成すると、 WebSphere MQ classes for JMS は、TEMP/*tempTopicPrefix* という文字で始まる名前のトピックを作成します。ここで、*tempTopicPrefix* は、接続ファクトリーの TEMPTOPICPREFIX プロパティーの値です。

Uniform Resource Identifier (URI)

キュー URI は、WebSphere MQ キューの名前を指定し、オプションで、キューを所有するキュー・マネージャーの名前、およびアプリケーションにより作成される Queue オブジェクトの 1 つ以上のプロパティーを指定するストリングです。 トピック URI は、トピックの名前と、オプションでアプリケーションにより作成される 1 つ以上の Topic オブジェクトのプロパティーを指定するストリングです。

キュー URI の形式は以下のとおりです。

```
queue://[qMgrName]/qName[?propertyName1=propertyValue1
                         &propertyName2=propertyValue2
                         &...]
```

トピック URI の形式は以下のとおりです。

```
topic://topicName[?propertyName1=propertyValue1
                  &propertyName2=propertyValue2
                  &...]
```

これらの形式の変数には、以下の意味があります。

qMgrName
URI により識別されるキューを所有するキュー・マネージャーの名前。

キュー・マネージャーは、ローカル・キュー・マネージャーまたはリモート・キュー・マネージャーのいずれかです。 リモート・キュー・マネージャーの場合、WebSphere MQ は、アプリケーションがメッセージをキューに送信するときに Websphere MQ がメッセージをローカル・キュー・マネージャーからリモート・キュー・マネージャーに経路指定できるように構成する必要があります。

名前が指定されない場合、ローカル・キュー・マネージャーが想定されます。

qName
WebSphere MQ キューの名前。

キューは、ローカル・キュー、別名キュー、またはリモート・キュー定義のいずれかです。

キュー名の作成規則については、IBM WebSphere MQ オブジェクトの命名規則を参照してください。

topicName
トピックの名前。

トピック名の作成規則については、IBM WebSphere MQ オブジェクトの命名規則を参照してください。ワイルドカード文字 +、#、*、および? は使用しないでください。 トピック名で使用できます。 これらの文字を含むトピック名は、サブスクライブするときに予期しない結果になる場合があります。 トピック・ストリングの使用を参照してください。

***propertyName1*, *propertyName2*, ...**
アプリケーションにより作成される Queue または Topic オブジェクトのプロパティー名。 885 ページの表 123 は、URI で使用できる有効なプロパティー名をリストしています。

プロパティーを指定しない場合、Queue または Topic オブジェクトは、そのすべてのプロパティーでデフォルト値を取ります。

***propertyValue1*, *propertyValue2*, ...**
アプリケーションにより作成される Queue または Topic オブジェクトのプロパティーの値。 885 ページの表 123 は、URI で使用できる有効なプロパティーの値をリストしています。

大括弧 ([]) はオプション・コンポーネントを示し、省略符号 (...) は、プロパティーの名前と値の対のリストがあれば、それに 1 つ以上の名前と値の対を含められることを意味します。

885 ページの表 123 は、キューおよびトピック URI で使用できる有効なプロパティー名および有効値をリストしています。 WebSphere MQ JMS 管理ツールがプロパティーの値にシンボリック定数を使用するとしても、URI にはシンボリック定数を含めることはできません。

表 *123.* キューおよびトピック *URI* で使用するプロパティー名と有効値

プロパティー名	**説明**	**有効値**
CCSID	WebSphere MQ classes for JMS がメッセージを宛先に転送するときに、メッセージの本体の文字データが表記される方法	• WebSphere MQ によりサポートされる任意のコード化文字セット ID。
encoding	WebSphere MQ classes for JMS がメッセージを宛先に転送するときに、メッセージの本体に数値データが表記される方法	• WebSphere MQ メッセージ記述子の「エンコード」フィールドの任意の有効な値。
expiry	宛先に送信されるメッセージの存続時間	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの存続時間。 • 0 - 宛先に送信されるメッセージは有効期限が切れることはありません。 • ミリ秒で存続時間を指定する正整数。
multicast	ブローカーへのリアルタイム接続を使用するときのトピックに対するマルチキャストの設定	以下のリストには有効値が含まれます。各値と関連付けられるのは、WebSphere MQ JMS 管理ツールで使用される、MULTICAST プロパティーの対応する値です。 MULTICAST プロパティーおよびその有効値の説明については、Properties of IBM WebSphere MQ classes for JMS オブジェクトのプロパティーを参照してください。 • -1 - ASCF • 0 - DISABLED • 3 - NOTR • 5 - RELIABLE • 7 - ENABLED

表 123. キューおよびトピック *URI* で使用するプロパティー名と有効値 (続き)

プロパティー名	説明	有効値
persistence	宛先に送信されるメッセージの持続性	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの持続性。 • -1- WebSphere MQ キューまたはトピックの *DefPersistence* 属性で指定されたとおり。 • 1 - 非持続。 • 2 - 持続。 • 3 - WebSphere MQ JMS 管理ツールで使用される PERSISTENCE プロパティーの値 HIGH と等価。 この値の説明については、909 ページの『JMS 持続メッセージ』を参照してください。
priority	宛先に送信されるメッセージの優先順位	• -2 - send() 呼び出しに指定されたとおり。または send() 呼び出しに指定されていない場合は、メッセージ・プロデューサーのデフォルトの優先順位。 • -1- WebSphere MQ キューまたはトピックの *DefPriority* 属性によって指定されたとおり。 • 宛先に送信されるメッセージの優先順位を指定する、0 から 9 の範囲内の整数。
targetClient	宛先に送信されるメッセージに MQRFH2 ヘッダーが含まれるかどうか。	• 0 - メッセージには MQRFH2 ヘッダーが含まれます。 • 1 - メッセージには MQRFH2 ヘッダーは含まれません。

例えば、以下の URI は、ローカル・キュー・マネージャーにより所有される、Q1 という WebSphere MQ キューを識別します。 この URI を使用して作成された Queue オブジェクトには、そのすべてのプロパティーのデフォルト値があります。

```
queue:///Q1
```

以下の URI は、QM2 というキュー・マネージャーによって所有される、Q2 という WebSphere MQ キューを識別します。 この宛先に送信されるすべてのメッセージの優先順位は 6 です。 この URI を使用して作成されたキュー・オブジェクトの残りのプロパティーには、デフォルト値が含まれます。

```
queue://QM2/Q2?priority=6
```

以下の URI は、Sport/Athletics/Results というトピックを識別します。 この宛先に送信されるすべてのメッセージは非永続であり、優先順位は 0 です。 この URI を使用して作成されたトピック・オブジェクトの残りのプロパティーには、デフォルト値が含まれます。

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

JMS アプリケーションでのメッセージの送信

JMS アプリケーションがメッセージを宛先に送信する前に、まず宛先用の MessageProducer オブジェクトを作成する必要があります。 メッセージを宛先に送信するために、アプリケーションは Message オブジェクトを作成し、MessageProducer オブジェクトの send() メソッドを呼び出します。

アプリケーションは MessageProducer オブジェクトを使用してメッセージを送信します。 アプリケーションは通常、MessageProducer オブジェクトを特定の宛先用 (キューまたはトピックが可能) に作成するため、同じメッセージ・プロデューサーを使用して送信されたメッセージは、すべて同じ宛先に送られます。したがって、アプリケーションが MessageProducer オブジェクトを作成する前に、まず Queue または Topic オブジェクトを作成する必要があります。 Queue または Topic オブジェクトの作成方法について詳しくは、以下のトピックを参照してください。

- 869 ページの『JNDI を使用して JMS アプリケーションで管理対象オブジェクトを取り出す』
- 870 ページの『IBM JMS 拡張機能の使用』
- 877 ページの『WebSphere MQ JMS 拡張機能の使用』
- 882 ページの『JMS アプリケーションでの宛先の作成』

以下の例が示すように、MessageProducer オブジェクトを作成するために、アプリケーションは Session オブジェクトの createProducer() メソッドを使用できます。

```
MessageProducer producer = session.createProducer(destination);
```

パラメーター destination は、アプリケーションによって前もって作成された Queue または Topic オブジェクトです。

アプリケーションがメッセージを送信する前に、Message オブジェクトを作成する必要があります。 メッセージ本体にはアプリケーション・データが含まれており、JMS は以下の 5 タイプのメッセージ本体を定義しています。

- Bytes
- マップ
- オブジェクト
- ストリーム
- Text

メッセージ本体の各タイプにはその固有の JMS インターフェースがあります。これは Message インターフェースのサブインターフェースであり、その本体のタイプでメッセージを作成するための Session インターフェースのメソッドです。 以下のステートメントが示すように、例えばテキスト・メッセージのインターフェースは TextMessage であり、アプリケーションが Session オブジェクトの createTextMessage() メソッドを使用してテキスト・メッセージを作成します。

```
TextMessage outMessage = session.createTextMessage(outString);
```

メッセージおよびメッセージ本体の詳細については、811 ページの『JMS メッセージ』を参照してください。

以下の例が示すように、メッセージを送信するために、アプリケーションは MessageProducer オブジェクトの send() メソッドを使用します。

```
producer.send(outMessage);
```

アプリケーションは send() メソッドを使用して、どちらのメッセージング・ドメインでもメッセージを送信できます。 宛先の種類によって、どのメッセージング・ドメインが使用されるかが決定されます。 ただし、パブリッシュ/サブスクライブ・ドメイン特定である MessageProducer のサブインターフェースである TopicPublisher には、send() メソッドの代わりに使用できる publish() メソッドもあります。 これら 2 つのメソッドは、機能的には同じです。

アプリケーションは、宛先を指定しない MessageProducer オブジェクトを作成できます。 この場合、アプリケーションは、send() メソッドを呼び出すときに宛先を指定する必要があります。

アプリケーションがトランザクション内でメッセージを送信する場合、トランザクションがコミットされるまで、そのメッセージは宛先に送信されません。 これは、アプリケーションがメッセージを送信できず、同じトランザクション内でメッセージに対する応答を受信できないことを意味します。

アプリケーションがメッセージを送信したときに WebSphere MQ classes for JMS がメッセージを転送し、キュー・マネージャーがメッセージを支障なく受信したかどうかを判別せずに制御をアプリケーションに戻すように、宛先を構成することが可能です。 これは、非同期書き込み と呼ばれることもあります。 詳細については、924 ページの『IBM WebSphere MQ classes for JMS でのメッセージの非同期書き込み』を参照してください。

JMS アプリケーションでのメッセージの受信

アプリケーションはメッセージ・コンシューマーを使用してメッセージを受信します。 永続トピック・サブスクライバーは、コンシューマーが非アクティブの間に送信されたメッセージも含め、宛先に送信されたすべてのメッセージを受信するメッセージ・コンシューマーです。 アプリケーションは、メッセージ・セレクターを使用することによって、受信するメッセージを選択することができ、メッセージ・リスナーを使用することによって、メッセージを非同期で受信することができます。

アプリケーションは MessageConsumer オブジェクトを使用してメッセージを受信します。 アプリケーションは、特定の宛先 (キューまたはトピック) の MessageConsumer オブジェクトを作成し、メッセージ・コンシューマーを使って受信したすべてのメッセージが同じ宛先から受信されるようにします。 そのため、アプリケーションは MessageConsumer オブジェクトを作成する前に、Queue または Topic オブジェクトを最初に作成しておく必要があります。 Queue または Topic オブジェクトの作成方法について詳しくは、以下のトピックを参照してください。

- 869 ページの『JNDI を使用して JMS アプリケーションで管理対象オブジェクトを取り出す』
- 870 ページの『IBM JMS 拡張機能の使用』
- 877 ページの『WebSphere MQ JMS 拡張機能の使用』
- 882 ページの『JMS アプリケーションでの宛先の作成』

アプリケーションは、以下の例で示されているように Session オブジェクトの createConsumer() メソッドを使用して MessageConsumer オブジェクトを作成します。

```
MessageConsumer consumer = session.createConsumer(destination);
```

パラメーター `destination` は、アプリケーションによって前もって作成された Queue または Topic オブジェクトです。

その後、アプリケーションは、以下の例で示されているように MessageConsumer オブジェクトの receive() メソッドを使用して宛先からメッセージを受信します。

```
Message inMessage = consumer.receive(1000);
```

receive() 呼び出しのパラメーターは、メッセージを即時に入手できない場合に適切なメッセージが到着するまでメソッドが待機する時間 (ミリ秒単位) を指定します。 このパラメーターを省略すると、呼び出しは適切なメッセージが到着するまで無期限にブロックされます。 アプリケーションがメッセージを待機しないようにするには、代わりに receiveNoWait() メソッドを使用します。

receive() メソッドは、特定のタイプのメッセージを戻します。 例えば、アプリケーションがテキスト・メッセージを受信した場合に、receive() 呼び出しで戻されるオブジェクトは TextMessage オブジェクトです。

しかし、receive() 呼び出しで戻される、宣言されたタイプのオブジェクトは、Message オブジェクトです。 そのため、受信したばかりのメッセージの本文からデータを抽出するには、アプリケーションは Message クラスから、より特定的なサブクラス (TextMessage など) にキャストする必要があります。 メッセージのタイプが分からない場合、アプリケーションは `instanceof` 演算子を使用してタイプを判別することができます。 エラーをスムーズに処理できるように、常に、キャストする前にアプリケーションでメッセージのタイプを判別しておくことをお勧めします。

以下のコードは、instanceof 演算子を使用し、テキスト・メッセージの本文からデータを抽出する方法を示します。

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

アプリケーションがトランザクション内でメッセージを送信する場合、トランザクションがコミットされるまで、そのメッセージは宛先に送信されません。 これは、アプリケーションがメッセージを送信できず、同じトランザクション内でメッセージに対する応答を受信できないことを意味します。

先読み用に構成されている宛先からメッセージ・コンシューマーがメッセージを受信する場合、アプリケーションが終了したときに先読みバッファーにある非永続メッセージはすべて廃棄されます。

パブリッシュ/サブスクライブ・ドメインで JMS は、非永続トピック・サブスクライバーと永続トピック・サブスクライバーの 2 種類のメッセージ・コンシューマーを識別します。これらについては、以下の 2 つのセクションで説明されます。

非永続トピック・サブスクライバー

非永続トピック・サブスクライバーは、サブスクライバーがアクティブになっている間にパブリッシュされたメッセージのみを受信します。 非永続サブスクリプションは、アプリケーションが非永続トピック・サブスクライバーを作成したときに開始し、アプリケーションがサブスクライバーを閉じるかまたはサブスクライバーが有効範囲から外れたときに終了します。 非永続トピック・サブスクライバーは、WebSphere MQ classes for JMS の拡張として、保存パブリケーションも受信しますが、ブローカーとのリアルタイム接続の使用時には受信しません。

非永続トピック・サブスクライバーを作成するために、アプリケーションは、宛先として Topic オブジェクトを指定して、ドメイン独立 createConsumer() メソッドを使用できます。 あるいは、以下の例で示されているように、ドメイン特定 createSubscriber() メソッドをアプリケーションで使用することもできます。

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

パラメーター topic は、アプリケーションによって前もって作成された Topic オブジェクトです。

永続トピック・サブスクライバー

制約事項 : アプリケーションは、ブローカーとのリアルタイム接続を使用しているときには永続トピック・サブスクライバーを作成できません。

永続トピック・サブスクライバーは、永続サブスクリプションの存続中にパブリッシュされるすべてのメッセージを受信します。 これには、サブスクライバーがアクティブになっていない間にパブリッシュされるすべてのメッセージも含まれます。 永続トピック・サブスクライバーは、WebSphere MQ classes for JMS の拡張として、保存パブリケーションも受信します。

アプリケーションは、以下の例で示されているように Session オブジェクトの createDurableSubscriber() メソッドを使用して永続トピック・オブジェクトを作成します。

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

createDurableSubscriber() 呼び出しの最初のパラメーターは、アプリケーションによって前もって作成された Topic オブジェクトで、2 番目のパラメーターは、永続サブスクリプションを識別するために使用される名前です。

永続トピック・サブスクライバーを作成するためのセッションには、それに関連付けられたクライアント ID が必要です。 セッションに関連付けられているクライアント ID は、そのセッションの作成時に使用された接続のクライアント ID と同じです。 クライアント ID は、 ConnectionFactory オブジェクトの CLIENTID プロパティーを設定することによって指定できます。 あるいは、アプリケーションから、

Connection オブジェクトの setClientID() メソッドを呼び出すことによって、クライアント ID を指定できます。

永続サブスクリプションの識別で使用される名前は、クライアント ID 内でのみ固有であることが必要です。したがって、クライアント ID は、永続サブスクリプションの完全な固有 ID の一部を形成します。 既に作成済みの永続サブスクリプションを使用し続けるには、アプリケーションがその永続サブスクリプションに関連付けられているものと同じクライアント ID を持つセッション、および同じサブスクリプション名を使用して、永続トピック・サブスクライバーを作成する必要があります。

永続サブスクリプションは、永続サブスクリプションが現時点で存在していないクライアント ID とサブスクリプション名を使用してアプリケーションが永続トピック・サブスクライバーを作成するときに開始します。 ただし、永続サブスクリプションは、アプリケーションが永続トピック・サブスクライバーを閉じるときには終了しません。 永続サブスクリプションを終了するには、アプリケーションがその永続サブスクリプションに関連付けられているものと同じクライアント ID を持つ Session オブジェクトの unsubscribe() メソッドを呼び出す必要があります。 以下の例で示されているように、unsubscribe() 呼び出しのパラメーターはサブスクリプション名です。

```
session.unsubscribe("D_SUB_000001");
```

永続サブスクリプションの有効範囲はキュー・マネージャーです。 永続サブスクリプションが 1 つのキュー・マネージャーに存在し、別のキュー・マネージャーに接続されているアプリケーションが同じクライアント ID とサブスクリプション名を持つ永続サブスクリプションを作成する場合、2 つの永続サブスクリプションは完全に独立したものとなります。

メッセージ・セレクター

後続の receive() 呼び出しで特定の条件を満たすメッセージのみを戻すように、アプリケーションで指定することができます。 MessageConsumer オブジェクトを作成する際、どのメッセージを取り出すかを決定する構造化照会言語 (SQL) 式をアプリケーションで指定することができます。 この SQL 式は、メッセージ・セレクター と呼ばれます。 メッセージ・セレクターには JMS メッセージ・ヘッダー・フィールドとメッセージ・プロパティーの名前を含めることができます。 メッセージ・セレクターを構成する方法について詳しくは、812 ページの『JMS のメッセージ・セレクター』を参照してください。

以下の例は、myProp という名前のユーザー定義プロパティーに基づいてアプリケーションでメッセージを選択する方法を示しています。

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

JMS の仕様では、アプリケーションでメッセージ・コンシューマーのメッセージ・セレクターを変更することができません。 アプリケーションがメッセージ・セレクターとともにメッセージ・コンシューマーを作成した後、そのコンシューマーが存続する間、メッセージ・セレクターも存続します。 アプリケーションで複数のメッセージ・セレクターが必要な場合、アプリケーションは各メッセージ・セレクターごとにメッセージ・コンシューマーを作成する必要があります。

アプリケーションがバージョン 7 のキュー・マネージャーと接続されている場合、接続ファクトリーの MSGSELECTION プロパティーには効果がありません。 パフォーマンスを最適化するために、キュー・マネージャーがすべてのメッセージ選択を行います。

ローカル・パブリケーションの抑制

アプリケーションは、コンシューマー独自の接続でパブリッシュされたパブリケーションを無視するメッセージ・コンシューマーを作成することができます。 アプリケーションは、以下の例で示されているように、createConsumer() 呼び出しの 3 番目のパラメーターを `true` に設定することによってこれを行います。

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

createDurableSubscriber() 呼び出しでは、アプリケーションは、以下の例で示されているように 4 番目のパラメーターを true に設定することによってこれを行います。

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

メッセージの非同期送達

アプリケーションは、メッセージ・リスナーをメッセージ・コンシューマーに登録することによって、メッセージを非同期で受信することができます。 メッセージ・リスナーは onMessage という名前のメソッドを持ちます。このメソッドは、適切なメッセージが入手可能になると非同期で呼び出されます。メッセージを処理することがこのメソッドの目的です。 以下のコードはそのメカニズムを示しています。

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

アプリケーションは、receive() 呼び出しを使用してメッセージを同期的に受信するか、またはメッセージ・リスナーを使用してメッセージを非同期で受信するために、セッションを使用することができます。セッションがその両方を兼ねることはできません。 アプリケーションが同期および非同期でメッセージを受信する必要がある場合、別個にセッションを作成する必要があります。

セッションが非同期でメッセージを受信するようにセットアップされると、以下のメソッドをそのセッションまたはそのセッションから作成されたオブジェクトで呼び出すことはできません。

- MessageConsumer.receive()
- MessageConsumer.receive(long)
- MessageConsumer.receiveNoWait()
- Session.acknowledge()
- MessageProducer.send(Destination, Message)
- MessageProducer.send(Destination, Message, int, int, long)
- MessageProducer.send(Message)
- MessageProducer.send(Message, int, int, long)
- Session.commit()
- Session.createBrowser(Queue)
- Session.createBrowser(Queue, String)
- Session.createBytesMessage()
- Session.createConsumer(Destination)

- Session.createConsumer(Destination, String, boolean)
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Destination)
- Session.createQueue(String)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String)

これらのメソッドのいずれかを呼び出すと、以下のメッセージを含む JMSException がスローされます。

```
JMSCC0033: セッションが非同期で使用されている場合、同期メソッド呼び出しは許可されません: 'method name'
```

はスローされます。

有害メッセージの受信

アプリケーションは、処理できないメッセージを受信することがあります。 メッセージを処理できない理由はさまざまです。例えば、メッセージの形式が誤っている場合などが考えられます。 このようなメッセージを有害メッセージと呼び、そのメッセージが繰り返し処理されるのを防ぐには、特別な処理が必要になります。

有害メッセージの処理方法の詳細については、893 ページの『IBM WebSphere MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

V 7.5.0.8 サブスクリプション・ユーザー・データの取得

IBM WebSphere MQ classes for JMS アプリケーションがキューから取り込むメッセージが、管理上定義された永続サブスクリプションによって書き込まれたものである場合、アプリケーションはそのサブスクリプションに関連付けられたユーザー・データ情報にアクセスする必要があります。 その情報はプロパティーとしてメッセージに追加されています。

Version 7.5.0, Fix Pack 8 以降、キューから取り込まれたメッセージに MQPS フォルダーが指定された RFH2 ヘッダーが含まれている場合は、Sud キーに関連付けられている値 (存在する場合) が、IBM WebSphere MQ classes for JMS アプリケーションに返される JMS メッセージ・オブジェクトに、String プロパティーとして追加されます。 このプロパティーをメッセージから取得するには、JmsConstants インターフェースの定数 JMS_IBM_SUBSCRIPTION_USER_DATA をメソッド javax.jms.Message.getStringProperty(java.lang.String) で使用して、サブスクリプション・ユーザー・データを取得します。

以下の例では、MQSC コマンド **DEFINE SUB** を使用して、管理永続サブスクリプションを定義します。

```
DEFINE SUB('MY.SUBCRIPTION') TOPICSTR('PUBLIC') DEST('MY.SUBSCRIPTION.Q')
USERDATA('Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q')
```

トピック・ストリング PUBLIC にパブリッシュされたメッセージのコピーが、キュー MY.SUBSCRIPTION.Q に書き込まれます。 そして、そのメッセージには、この永続サブスクリプションに関連付けたユーザー・データがプロパティーとして追加されます。プロパティーは、キー Sud を使用して RFH2 ヘッダーの MQPS フォルダーに保管されます。

IBM WebSphere MQ classes for JMS アプリケーションで以下の呼び出しを実行できます。

```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

すると、次のストリングが返されます。

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

.

関連概念

816 ページの『MQRFH2 ヘッダーと JMS』

関連タスク

管理サブスクリプションの定義

関連資料

DEFINE SUB

インターフェース JmsConstants

WebSphere MQ classes for JMS アプリケーションのクローズ

WebSphere MQ classes for JMS アプリケーションが、停止する前に特定の JMS オブジェクトを明示的に閉じることは重要です。 ファイナライザーは呼び出せない場合があるので、リソースを解放するためにファイナライザーをあてにしないでください。 圧縮されたトレースをアクティブにしてアプリケーションを終了させないようにしてください。

ガーベッジ・コレクションのみでは、すべての WebSphere MQ classes for JMS および WebSphere MQ リソースをタイムリーに解放することはできません。特に、アプリケーションがセッション・レベル以下に多数の一時 JMS オブジェクトを作成する必要がある場合にそう言えます。 このため、必要でなくなった時に、アプリケーションが Connection、Session、MessageConsumer、または MessageProducer オブジェクトを閉じることが重要です。

アプリケーションが接続を閉じないで終了すると、その接続のトランザクション化されたセッションすべてで暗黙的なロールバックが行われます。 アプリケーションによる変更が確実にコミットされるようにするには、アプリケーションを閉じる前に接続を明示的に閉じてください。

JMS オブジェクトを閉じるためにアプリケーション内でファイナライザーを使用しないでください。 ファイナライザーは呼び出せない場合があるので、リソースを解放できないことがあります。 Connection を閉じると、それから作成されたすべての Session も閉じられます。 同様に、Session から作成された MessageConsumer および MessageProducer も、Session を閉じる時に閉じられます。 ただし、リソースがタイムリーな方法で解放されるようにするために、Session、MessageConsumer、および MessageProducer は明示的に閉じることを考慮してください。

トレース圧縮がアクティブ化されている場合、System.Halt() のシャットダウンと、異常で制御されていない JVM の終了は、トレース・ファイルの破損という結果になる可能性があります。 可能であれば、必要とするトレース情報を収集した後、トレース機能をオフにしてください。 アプリケーションを異常終了までトレースする場合は、圧縮されていないトレース出力を使用してください。

IBM WebSphere MQ classes for JMS でのポイズン・メッセージの処理

有害メッセージとは、受信側の MDB アプリケーションによって処理できないメッセージです。 ポイズン・メッセージが発生した場合、JMS MessageConsumer および ConnectionConsumer オブジェクトは、

BOQNAME および BOTHRESH の 2 つのキュー・プロパティーに従ってそのメッセージをリキューできます。

時折、不適切なフォーマットのメッセージがキューに到着する場合があります。 ここで言う「不適切なフォーマット」とは、受信側のアプリケーションがメッセージを正しく処理できないことを意味します。 このようなメッセージがあると、受信側のアプリケーションに障害が発生したり、この不適切なフォーマットのメッセージがアプリケーションによってバックアウトされたりすることがあります。 そして、メッセージは繰り返し入力キューに送達され、アプリケーションによって繰り返しバックアウトされる可能性があります。 これらのメッセージを、ポイズン・メッセージ と呼びます。 JMS MessageConsumer オブジェクトは、ポイズン・メッセージを検出し、それを代わりの宛先に転送します。

IBM WebSphere MQ キュー・マネージャーは、それぞれのメッセージがバックアウトされた回数のレコードを保持します。 この回数が、構成可能なしきい値に達すると、メッセージ・コンシューマーはそのメッセージを名前付きのバックアウト・キューに再キューイングします。 この再キューイングが何らかの理由により失敗すると、メッセージは入力キューから除去され、送達不能キューに再キューイングされるか、または廃棄されます。 詳細については、933 ページの『ASF でのキューからのメッセージの除去』を参照してください。

MessageConsumer と ConnectionConsumer が有害メッセージをリキューする方法には違いがあります。 ConnectionConsumer は、メッセージ送達に影響を与えずに有害メッセージをリキューできます。 このリキュー処理は、アプリケーション・コードへの実際のメッセージ送達に関連付けられたすべての作業単位の外側で実行されます。 これが可能なのは、ConnectionConsumer 操作がマルチスレッド化されているためです。

しかし、MessageConsumer はセッション・レベル下では単一スレッドであるため、すべての有害メッセージのリキューは現行の作業単位内で実行されます。 このことがアプリケーションの操作に影響することはありませんが、有害メッセージがトランザクション化セッションまたは Client_acknowledge セッションでリキューされた場合、そのリキュー・アクション自体がコミットされるのは、アプリケーション・コード (または該当する場合には、アプリケーション・コンテナー・コード) が現行の作業単位をコミットした後です。

JMS ConnectionConsumer オブジェクトは、同じキュー・プロパティーを使用して、同様の方法でポイズン・メッセージを処理します。 複数の接続コンシューマーが同じキューをモニターしている場合は、リキューが行われるしきい値の回数を超えても有害メッセージがアプリケーションに送達されることがあります。 この動作の原因は、接続コンシューマーが個別にキューをモニターして有害メッセージを再キューイングする方法にあります。

しきい値およびバックアウト・キューの名前は、IBM WebSphere MQ キューの属性です。 これらの属性の名前は、BackoutThreshold および BackoutRequeueQName です。 これらの属性が適用されるキューは、以下のとおりです。

- Point-to-Point メッセージングの場合、これは基礎ローカル・キューです。 これは、メッセージ・コンシューマーおよび接続コンシューマーがキューの別名を使用する場合に重要です。
- IBM WebSphere MQ メッセージング・プロバイダーの通常モードでのパブリッシュ/サブスクライブ・メッセージングの場合、トピックの管理対象キューは、このモデル・キューから作成されています。
- IBM WebSphere MQ メッセージング・プロバイダーのマイグレーション・モードにおけるパブリッシュ/サブスクライブ・メッセージングの場合、これは、TopicConnectionFactory オブジェクトで定義された CCSUB キュー、または Topic オブジェクトで定義された CCDSUB キューです。

IBM WebSphere MQ classes for JMS は、キューの BackoutThreshold および BackoutRequeueQName を照会します。 このため、アプリケーションを実行するユーザーに、キューでの問い合わせ権限を付与する必要があります。

V 7.5.0.9 ターゲット・キューがクラスター・キューである場合、必要な権限は、使用されている IBM WebSphere MQ classes for JMS のバージョンによって異なります。

- IBM WebSphere MQ classes for JMS for Version 7.5.0, Fix Pack 9 と、APAR IT26482 の暫定修正を使用する場合は、照会アクセス権限が必要です。
- その他のすべてのバージョンについては、inquire、browse、および get アクセス権限を付与します。

BackoutThreshold 属性および BackoutRequeueQName 属性を設定するには、次の MQSC コマンドを実行します。

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

BackoutThreshold 属性がゼロ以外の値に設定されている場合、予期しない動作を回避するために、BackoutRequeueQName 属性を有効なキュー名に設定します。

パブリッシュ/サブスクライブ・メッセージングの場合、システムがそれぞれのサブスクリプションごとに動的キューを作成すると、これらの属性値は IBM WebSphere MQ classes for JMS のモデル・キュー SYSTEM.JMS.MODEL.QUEUE から取得されます。 これらの設定を変更するには、以下を使用できます。

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

バックアウトのしきい値がゼロの場合、ポイズン・メッセージの処理は不可となり、ポイズン・メッセージは入力キュー上に残ります。 それ以外の場合は、バックアウト・カウントがしきい値に達すると、メッセージは指定されたバックアウト・キューに送信されます。 バックアウト・カウントがしきい値に達してもメッセージをバックアウト・キューに入れることができないときは、メッセージは送達不能キューに送信されるか、廃棄されます。 この状況は、バックアウト・キューが定義されていない場合、または MessageConsumer オブジェクトがメッセージをバックアウト・キューに送信できない場合に発生します。詳細については、933 ページの『ASF でのキューからのメッセージの除去』 を参照してください。

メッセージがバックアウト・リキュー・キューにリキューされた場合、メッセージ記述子 (MQMD) の一部のフィールドの値が変化します。 MQMD のフォーマットの詳細については、MQMD - メッセージ記述子を参照してください。

メッセージがバックアウト・キューに入れられると、次の MQMD のフィールドの値が変化します。

- PutDate が、メッセージがバックアウト・リキュー・キューに入れられた日付に更新されます。
- PutTime が、メッセージがバックアウト・リキュー・キューに入れられた時刻に更新されます。
- バックアウト・カウントがゼロにリセットされます。
- メッセージの存続時間が、元のメッセージを JMS アプリケーションが受信したときに残っていた存続時間を表すように更新されます。

以下のフィールドの値は、メッセージがバックアウト・キューに入れられても同じままです。

- StructId
- バージョン
- レポート
- MessageType
- Feedback
- Encoding
- CodedCharSetId
- MsgId
- CorrelId
- ReplyToQ
- ReplyToQMgr
- Format
- Persistence
- Priority

IBM WebSphere MQ classes for JMS の例外

IBM WebSphere MQ classes for JMS アプリケーションは、JMS API 呼び出しによりスローされたか、または例外ハンドラーに送達された例外を処理できなければなりません。

IBM WebSphere MQ classes for JMS は、例外をスローすることで実行時の問題をレポートします。JMSException は、JMS メソッドによりスローされる例外のルート・クラスであり、JMSException 例外のキャッチにより、すべての JMS 関連例外を処理する汎用の方法が提供されます。

各 JMSException 例外は、以下の情報をカプセル化します。

- Throwable.getMessage() メソッドの呼び出しによりアプリケーションが入手する、プロバイダー固有の例外メッセージ。
- JMSException.getErrorCode() メソッドの呼び出しによりアプリケーションが入手する、プロバイダー固有のエラー・コード。
- リンクの例外。 JMS API 呼び出しによりスローされる例外の多くは下位レベルの問題の結果であり、この例外にリンクされた別の例外により報告されます。 アプリケーションは、JMSException.getLinkedException() または Throwable.getCause() メソッドを呼び出すことでリンクされた例外を入手します。

IBM WebSphere MQ classes for JMS によってスローされる例外の大半は、JMSException のサブクラスのインスタンスです。 これらのサブクラスは com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装しますが、これは以下の追加情報を提供します。

- 例外メッセージの説明。これはアプリケーションが JmsExceptionDetail.getExplanation() メソッドを呼び出して入手します。
- 例外に対する推奨されるユーザー応答。これはアプリケーションが JmsExceptionDetail.getUserAction() メソッドを呼び出して入手します。
- 例外メッセージ内のメッセージ挿入のためのキー。 アプリケーションは、JmsExceptionDetail.getKeys() メソッドを呼び出すことで、すべてのキーのイテレーターを入手します。
- 例外メッセージ内のメッセージ挿入。 例えば、メッセージ挿入が例外を引き起こしたキューの名前であり、アプリケーションがその名前にアクセスできれば役立つ場合があります。 アプリケーションは、JmsExceptionDetail.getValue() メソッドを呼び出すことで、指定されたキーに対応するメッセージ挿入を入手します。

JmsExceptionDetail インターフェース内のすべてのメソッドは、詳細が入手できない場合にはヌルを戻すことがあります。

例えば、存在しない IBM WebSphere MQ キュー用のメッセージ・プロデューサーをアプリケーションが 作成しようとすると、以下の情報を持つ例外がスローされます。

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but WebSphere MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

スローされた例外 com.ibm.msg.client.jms.DetailedInvalidDestinationException は、javax.jms.InvalidDestinationException のサブクラスであり、com.ibm.msg.client.jms.JmsExceptionDetail インターフェースを実装します。

リンクされた例外

リンクされた例外は、実行時の問題に関する詳細な情報を提供します。 したがって、スローされる各 JMSException 例外に対して、アプリケーションはリンクされた例外を検査する必要があります。 リンク付きの例外自体には、別のリンク付きの例外がある場合があるため、リンク付きの例外は元の根本の問題に戻るチェーンを形成します。 リンク付きの例外は、java.lang.Throwable クラスのチェーニングされた例外メカニズムを使用して実装され、アプリケーションは Throwable.getCause() メソッドを呼び出してリンク付き例外を取得します。 JMSException 例外の場合、getLinkedException() メソッドは、実際には Throwable.getCause() メソッドによって代行されます。

例えば、キュー・マネージャーへの接続時にアプリケーションが誤ったポート番号を指定する場合、例外は以下のチェーンを形成します。

```
com.ibm.msg.client.jms.DetailIllegalStateException
   |
   +--->com.ibm.mq.MQException
           |
           +--->com.ibm.mq.jmqi.JmqiException
```

```
                    |
                    +--->java.net.ConnectionException
```

一般に、チェーン内の各例外は、コード内の異なるレイヤーからスローされます。 例えば、先行するチェーン内の例外は、以下のレイヤーによりスローされます。

- 最初の例外である、JMSException のサブクラスのインスタンスは、IBM WebSphere MQ classes for JMS の共通レイヤーによりスローされます。
- 次の例外である、com.ibm.mq.MQException のインスタンスは、IBM WebSphere MQ メッセージング・プロバイダーによりスローされます。
- 次の例外 (com.ibm.mq.jmqi.JmqiException のインスタンス) は、MQI への共通 Java インターフェースによってスローされます。
- 最後の例外である java.net.ConnectionException のインスタンスは、Java クラス・ライブラリーによってスローされます。

IBM WebSphere MQ classes for JMS の階層化アーキテクチャーの形式について詳しくは、802 ページの『IBM WebSphere MQ classes for JMS のアーキテクチャー』を参照してください。

以下のコードと類似のコードを使用して、アプリケーションは適切なすべての情報を取り出すために、このチェーンを繰り返すことができます。

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
    System.err.println("Caught JMSException");

    // Check for linked exceptions in JMSException
    Throwable t = je;
    while (t != null) {
        // Write out the message that is applicable to all exceptions
        System.err.println("Exception Msg: " + t.getMessage());
        // Write out the exception stack trace
        t.printStackTrace(System.err);

        // Add on specific information depending on the type of exception
        if (t instanceof JMSException) {
            JMSException je1 = (JMSException) t;
            System.err.println("JMS Error code: " + je1.getErrorCode());

            if (t instanceof JmsExceptionDetail){
                JmsExceptionDetail jed = (JmsExceptionDetail)je1;
                System.err.println("JMS Explanation: " + jed.getExplanation());
                System.err.println("JMS Explanation: " + jed.getUserAction());
            }
        } else if (t instanceof MQException) {
            MQException mqe = (MQException) t;
            System.err.println("WMQ Completion code: " + mqe.getCompCode());
            System.err.println("WMQ Reason code: " + mqe.getReason());
        } else if (t instanceof JmqiException){
            JmqiException jmqie = (JmqiException)t;
            System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
            System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
            System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
            System.err.println("WMQ Msg User Response: "
                               + jmqie.getWmqMsgUserResponse());
            System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
        }

        // Get the next cause
        t = t.getCause();
    }
}
```

例外のタイプは異なる場合があり、さまざまなタイプの例外がさまざまな情報をカプセル化するため、アプリケーションはチェーン内の各例外のタイプを常時確認する必要があることに注意してください。

問題に関する IBM WebSphere MQ 固有情報の入手

com.ibm.mq.MQException および com.ibm.mq.jmqi.JmqiException のインスタンスは、問題に関する IBM WebSphere MQ 固有の情報をカプセル化します。

MQException 例外は、以下の情報をカプセル化します。

- getCompCode() メソッドの呼び出しによりアプリケーションが入手する、完了コード。
- getReason() メソッドの呼び出しによりアプリケーションが入手する、理由コード。

JmqiException 例外はさらに、完了コードと理由コードをカプセル化します。 しかしさらに、JmqiException 例外は、AMQ*nnnn* または CSQ*nnnn* メッセージ内の情報も、それが例外と関連付けられていればカプセル化します。 例外の適切なメソッドを呼び出すことによって、アプリケーションはこのメッセージのさまざまな構成要素 (重大度、説明、およびユーザー応答など) を入手できます。

このセクションで言及されているメソッドの使用方法の例については、896 ページの『リンクされた例外』のサンプル・コードを参照してください。

IBM WebSphere MQ classes for JMS の以前のバージョンからのアップグレード

IBM WebSphere MQ classes for JMS の以前のバージョンと比較すると、ほとんどのエラー・コードと例外メッセージはバージョン 7 で変更されました。 これらの変更の理由は、 IBM WebSphere MQ classes for JMS のアーキテクチャーが階層化され、コード内のさまざまなレイヤーから例外がスローされるようになったためです。

例えば、存在しないキュー・マネージャーにアプリケーションが接続しようとした場合、以前のバージョンの IBM WebSphere MQ classes for JMS は、以下の情報と共に JMSException 例外をスローしました。

```
MQJMS2005: Failed to create MQQueueManager for 'localhost:QM_test'.
```

この例外には、以下の情報があるリンクされた MQException 例外が含まれていました。

```
MQJE001: Completion Code 2, Reason 2058
```

同じ状況で比較することにより、バージョン 7 の IBM WebSphere MQ classes for JMS は、JMSException 例外を以下の情報と共にスローします。

```
Message : JMSWMQ0018: Failed to connect to queue manager 'QM_test' with
          connection mode 'Client' and host name 'localhost'.
Class : class com.ibm.msg.client.jms.DetailedJMSException
Error Code : JMSWMQ0018
Explanation : null
User Action : Check the queue manager is started and if running in client mode,
              check there is a listener running. Please see the linked exception
              for more information.
```

この例外には、以下の情報があるリンクされた MQException 例外が含まれます。

```
Message : JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED')
          reason '2058' ('MQRC_Q_MGR_NAME_ERROR').
Class : class com.ibm.mq.MQException
Completion Code : 2
Reason Code : 2058
```

Throwable.getMessage() メソッドによって返された例外メッセージ、または JMSException.getErrorCode() メソッドによって返されたエラー・コードをアプリケーションで構文解析またはテストする場合、バージョン 7 より前のリリースからアップグレードすると、バージョン 7 の IBM WebSphere MQ classes for JMS を使用するためにアプリケーションの変更が必要になることがあります。

例外リスナー

アプリケーションは Connection オブジェクトで例外リスナーを登録できます。 その後、問題が発生して接続が使用不可になった場合、IBM WebSphere MQ classes for JMS は、onException() メソッドの呼び出しによって 例外を例外リスナーに送ります。 次いで、アプリケーションには、接続を再確立する機会があります。

V 7.5.0.8 IBM WebSphere MQ Version 7.5.0、フィックスパック 8 から組み込まれた APAR IT14820 は、アプリケーションで使用される JMS 接続ファクトリーの ASYNC_EXCEPTIONS プロパティーが C_EXCEPTIONALL に設定されていても、アプリケーションの JMS ExceptionListener が非接続切断例外 (MQRC_GET_INHIBITED など) に対して呼び出されないという問題を修正しました。 これは、Version 7.5.0, Fix Pack 8 より前のデフォルト値でした。

V 7.5.0.8 JMS MessageListener と JMS ExceptionListener を構成する現行の JMS アプリケーションの動作を維持し、IBM WebSphere MQ classes for JMS が JMS の仕様と整合性を持つようにするために、IBM WebSphere MQ classes for JMS で ASYNC_EXCEPTIONS JMS ConnectionFactory プロパティーのデフォルト値が ASYNC_EXCEPTIONS_CONNECTIONBROKEN に変更されています。 結果として、デフォルトでは、接続切断のエラー・コードに対応する例外だけがアプリケーションの JMS ExceptionListener に送信されます。

V 7.5.0.8 さらに、Version 7.5.0, Fix Pack 8 以降で行われた IBM WebSphere MQ classes for JMS の更新によって、アプリケーションで使用される JMS ConnectionFactory の ASYNC_EXCEPTIONS プロパティーが値 ASYNC_EXCEPTIONS_ALL に設定されている場合、非同期メッセージ・コンシューマーへのメッセージ送信時に発生する接続切断以外のエラーに関係する JMSExceptions が、引き続き登録済み ExceptionListener に送信されるようになりました。

V 7.5.0.8 Version 7.5.0, Fix Pack 8 の例外リスナーの変更内容、および以前のリリースから変更が加えられた理由について詳しくは、 JMS: バージョン 7.5 を参照してください。

他のあらゆるタイプの問題の場合、JMSException 例外は現行の JMS API 呼び出しによってスローされます。

アプリケーションが例外リスナーを Connection オブジェクトで登録していない場合、例外リスナーに本来送達されるはずのすべての例外は、IBM WebSphere MQ classes for JMS ログに書き込まれます。

関連資料

ASYNCEXCEPTION

WebSphere MQ classes for JMS のエラーのロギング

ユーザーによる訂正処置が必要かもしれない実行時問題に関する情報は、WebSphere MQ classes for JMS ログに書き込まれます。

例えば、アプリケーションが接続ファクトリーのプロパティーを設定しようとしたのにプロパティーの名前が認識されない場合、WebSphere MQ classes for JMS はその問題に関する情報をログに書き込みます。

デフォルトでは、ログを含むファイルは mqjms.log という名前で、現行作業ディレクトリーに含まれます。ただし、WebSphere MQ classes for JMS 構成ファイルの com.ibm.msg.client.commonservices.log.outputName プロパティーを設定することによって、ログ・ファイルの名前と場所を変更できます。 WebSphere MQ classes for JMS 構成ファイルの詳細については、732 ページの『IBM WebSphere MQ classes for JMS 構成ファイル』を参照してください。
com.ibm.msg.client.commonservices.log.outputName プロパティーの有効な値の詳細については、800 ページの『ロギングおよび IBM WebSphere MQ classes for JMS』を参照してください。

WebSphere MQ classes for JMS の First Failure Support Technology (FFST)

WebSphere MQ classes for JMS で重大な内部エラーが発生すると、First Failure Support Technology (FFST) 情報が生成されます。

FFST 情報は、JMSC*nnnn*.FDC という名前のファイルに書き込まれます (*nnnn* は 4 桁の数字)。 このファイルは、トレース出力が書き込まれるディレクトリーのサブディレクトリーである、FFDC というディレクトリーにあります。 デフォルトでは、トレース出力は現行作業ディレクトリーに書き込まれますが、WebSphere MQ classes for JMS 構成ファイルの **com.ibm.msg.client.commonservices.trace.outputName** プロパティーを設定することにより、トレース出力を別のディレクトリーにリダイレクトすることができます。 WebSphere MQ classes for JMS 構成ファイルの詳細については、732 ページの『IBM WebSphere MQ classes for JMS 構成ファイル』を参照してください。

FFST 情報が生成されるときにトレースが使用可能になっているならば、FFST 情報はトレース・ファイルにも書き込まれます。 JMS プログラムのトレース方法について詳しくは、IBM WebSphere MQ classes for JMS アプリケーションのトレースを参照してください。

FFDC ファイルの生成を抑止するには、プロパティー **com.ibm.msg.client.commonservices.ffst.suppress** を次のように設定します。

0
すべての FFDC ファイルを出力します (デフォルト)。

-1
特定のタイプの最初の FFDC ファイルのみを出力します。

integer
この数値の倍数のファイルを除き、すべての FFDC ファイルを抑制します。

WebSphere MQ classes for JMS アプリケーションから WebSphere MQ 機能へのアクセス

WebSphere MQ classes for JMS は、WebSphere MQ のいくつかのフィーチャーを活用する機能を提供します。

重要 : これらの機能は JMS 仕様の枠外であるか、またはある場合 JMS 仕様に違反するものです。 それらを用いると、お使いのアプリケーションが他の JMS プロバイダーとの互換性をなくす恐れがあります。 それら JMS 仕様に準拠しない機能には「重要」通知が付されます。

WebSphere MQ classes for JMS アプリケーションからのメッセージ記述子の読み取りおよび書き込み

宛先やメッセージのプロパティーを設定することによって、メッセージ記述子 (MQMD) へのアクセス権限を制御します。

WebSphere MQ アプリケーションの中には、送られてくるメッセージの MQMD に特定の値が設定されていることが必要なものもあります。 WebSphere MQ classes for JMS は、JMS アプリケーションが MQMD フィールドを設定することにより、JMS アプリケーションが WebSphere MQ アプリケーションを「駆動」できるようにするためのメッセージ属性を提供します。

MQMD プロパティーの設定値が有効になるように、宛先オブジェクト・プロパティー WMQ_MQMD_WRITE_ENABLED を true に設定する必要があります。 そうすれば、MQMD フィールドに値を割り当てるために、メッセージのプロパティー設定方式 (例えば、setStringProperty) を使用することができます。 StrucId と Version を除くすべての MQMD フィールドが公開されます。BackoutCount は読み取り可能ですが、書き込むことはできません。

この例では、MQMD.UserIdentifier が「JoeBloggs」に設定されたキューまたはトピックにメッセージが書き込まれます。

```
// Create a ConnectionFactory, connection, session, producer, message
// ...

// Create a destination
// ...

// Enable MQMD write
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_WRITE_ENABLED, true);

// Optionally, set a message context if applicable for this MD field
dest.setIntProperty(WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT,
  WMQConstants.WMQ_MDCTX_SET_IDENTITY_CONTEXT);

// On the message, set property to provide custom UserId
msg.setStringProperty("JMS_IBM_MQMD_UserIdentifier", "JoeBloggs");

// Send the message
// ...
```

JMS_IBM_MQMD_UserIdentifier を設定する前に WMQ_MQMD_MESSAGE_CONTEXT を設定する必要があります。 WMQ_MQMD_MESSAGE_CONTEXT の使用に関する詳細については、903 ページの『JMS メッセージ・オブジェクトのプロパティー』を参照してください。

同様に、メッセージを受信する前に WMQ_MQMD_READ_ENABLED を true に設定することによって MQMD フィールドの内容を抽出し、次いで getStringProperty のようなメッセージのゲット・メソッドを使用できます。 受信するプロパティーはすべて読み取り専用です。

次の例では、メッセージの MQMD.ApplIdentityData フィールドの値を保持している *value* フィールドが、キューまたはトピックから取得される結果となります。

```
// Create a ConnectionFactory, connection, session, consumer
// ...

// Create a destination
// ...

// Enable MQMD read
dest.setBooleanProperty(WMQConstants.WMQ_MQMD_READ_ENABLED, true);

// Receive a message
// ...

// Get MQMD field value using a property
String value = rcvMsg.getStringProperty("JMS_IBM_MQMD_ApplIdentityData");
```

JMS 宛先オブジェクトのプロパティー
宛先オブジェクトの 2 つのプロパティーは、JMS からの MQMD へのアクセスを制御し、3 番目のものはメッセージのコンテキストを制御します。

表 *124.* プロパティーの名前および説明

プロパティー	**短縮形**	**説明**
WMQ_MQMD_WRITE_ENABLED	MDW	JMS アプリケーションが MQMD フィールドの値を設定できるかどうか
WMQ_MQMD_READ_ENABLED	MDR	JMS アプリケーションが MQMD フィールドの値を抽出できるかどうか
WMQ_MQMD_MESSAGE_ CONTEXT	MDCTX	JMS アプリケーションによって、メッセージ・コンテキストのどのレベルが設定されるか。 このプロパティーが有効となるためには、アプリケーションが適切なコンテキスト権限を持って実行されていなければなりません。

表 *125.* プロパティーの名前、値、および設定方式

プロパティー	**管理ツール内での有効値 (太字はデフォルト)**	**プログラム内での有効値**	**Set メソッド**
WMQ_MQMD_WRITE_ENABLED	• **NO** JMS_IBM_MQMD* プロパティーはすべて無視され、その値は基礎となる MQMD 構造にコピーされません。 • YES JMS_IBM_MQMD* プロパティーは処理されます。 それらの値は基礎となる MQMD 構造にコピーされます。	• **False** • True	setMQMDWriteEnabled

表 125. プロパティーの名前、値、および設定方式 (続き)

プロパティー	管理ツール内での有効値 (太字はデフォルト)	プログラム内での有効値	Set メソッド
WMQ_MQMD_READ_ENABLED	• **NO** メッセージの送信時に、送信されたメッセージの JMS_IBM_MQMD* プロパティーは、MQMD での更新済みのフィールド値を反映するように更新されません。 メッセージを受信するときに、送信側が JMS_IBM_MQMD* プロパティーの一部またはすべてを設定した場合でも、受信されたメッセージでそのプロパティーを使用できません。 • YES メッセージ送信の際に、送られるメッセージ上のすべての JMS_IBM_MQMD* プロパティーは、送信者が明示的に設定しないかったものも含め、MQMD 内の更新されたフィールド値を反映して更新されます。 メッセージを受信するときに、受信されたメッセージですべての JMS_IBM_MQMD* プロパティー (送信側が明示的に設定しなかったものを含む) を使用できます。	• **False** • True	setMQMDReadEnabled
WMQ_MQMD_MESSAGE_CONTEXT (WMQ_MQMD_MESSAGE_CONTEXT)	• **DEFAULT** MQOPEN API 呼び出しおよび MQPMO 構造体は、メッセージ・コンテキスト・オプションを明示的に指定しません。 • SET_IDENTITY_CONTEXT MQOPEN API 呼び出しは メッセージ・コンテキスト・ オプション MQOO_SET_ IDENTITY_CONTEXT を指定し、MQPMO 構造体は MQPMO_SET_IDENTITY_ CONTEXT を指定します。 • SET_ALL_CONTEXT MQOPEN API 呼び出しは メッセージ・コンテキスト・ オプション MQOO_SET_ALL_ CONTEXT を指定し、MQPMO 構造体は MQPMO_SET_ALL_ CONTEXT を指定します。	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	setMQMDMessageContext

JMS メッセージ・オブジェクトのプロパティー
JMS_IBM_MQMD という接頭部を持つメッセージ・オブジェクトのプロパティーは、対応する MQMD フィールドを設定したり読み取ったりすることを可能にします。

メッセージの送信

StrucId と Version を除くすべての MQMD フィールドが表されます。 これらのプロパティーは MQMD フィールドのみを参照しています。このフィールドでは、MQMD ヘッダーと MQRFH2 ヘッダーの両方でプロパティーが発生し、MQRFH2 のバージョンは設定も抽出もされません。

JMS_IBM_MQMD_BackoutCount を除き、これらのすべてのプロパティーを設定できます。
JMS_IBM_MQMD_BackoutCount に設定された値はすべて無視されます。

プロパティーが最大長を持っていて、長過ぎる値が提供された場合、その値は切り捨てられます。

特定のプロパティーについては、宛先オブジェクト上の WMQ_MQMD_MESSAGE_CONTEXT プロパティーも設定する必要があります。 このプロパティーが有効となるためには、アプリケーションが適切なコンテキスト権限を持って実行されていなければなりません。 WMQ_MQMD_MESSAGE_CONTEXT に適切な値を設定しない場合、プロパティーの値は無視されます。 WMQ_MQMD_MESSAGE_CONTEXT を適切な値に設定したものの、キュー・マネージャーに対する十分なコンテキスト権限がない場合、JMSException が発行されます。 WMQ_MQMD_MESSAGE_CONTEXT の特定の値を必要とするプロパティーは、以下のものです。

次のプロパティーでは、WMQ_MQMD_MESSAGE_CONTEXT が WMQ_MDCTX_SET_IDENTITY_CONTEXT または WMQ_MDCTX_SET_ALL_CONTEXT に設定される必要があります。

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

次のプロパティーでは、WMQ_MQMD_MESSAGE_CONTEXT が WMQ_MDCTX_SET_ALL_CONTEXT に設定される必要があります。

- JMS_IBM_MQMD_PutApplType
- JMS_IBM_MQMD_PutApplName
- JMS_IBM_MQMD_PutDate
- JMS_IBM_MQMD_PutTime
- JMS_IBM_MQMD_ApplOriginData

メッセージの受信

受信メッセージ上のこれらすべてのプロパティーは、WMQ_MQMD_READ_ENABLED プロパティーが true に設定されていれば、メッセージを作成したアプリケーションで設定した実際のプロパティーと関係なく利用できます。 JMS 仕様によれば、最初にプロパティーをすべてクリアしない限り、アプリケーションでは、受信したメッセージのプロパティーを変更できません。 プロパティーを変更せずに、受信メッセージを転送することができます。

重要 : ご使用のアプリケーションが、WMQ_MQMD_READ_ENABLED プロパティーが true に設定されている宛先からメッセージを受信し、そのメッセージを WMQ_MQMD_WRITE_ENABLED が true に設定されている宛先に転送すると、受信されたメッセージのすべての MQMD フィールド値が転送メッセージにコピーされるという結果になります。

プロパティーの表

この表は、MQMD フィールドを表すメッセージ・オブジェクトのプロパティーのリストを示します。 フィールドと、許容されている値についての完全な説明については、リンク先を参照してください。

表 *126.* プロパティーの名前、説明、およびタイプ

プロパティー	説明	Java 型	完全な説明のリンク先
JMS_IBM_MQMD_Report	レポート・メッセージのオプション	整数	Report

表 126. プロパティーの名前、説明、およびタイプ (続き)

プロパティー	説明	Java 型	完全な説明のリンク先
JMS_IBM_MQMD_MsgType	メッセージ・タイプ	整数	MsgType
JMS_IBM_MQMD_Expiry	メッセージの存続時間。	整数	Expiry
JMS_IBM_MQMD_Feedback	フィードバックまたは理由コード	整数	Feedback
JMS_IBM_MQMD_Encoding	メッセージ・データの数値エンコード	整数	Encoding
JMS_IBM_MQMD_CodedCharSetId	メッセージ・データの文字セット ID	整数	CodedCharSetId
JMS_IBM_MQMD_Format	メッセージ・データの形式名。	ストリング	Format
JMS_IBM_MQMD_Priority [1]	メッセージ優先順位	整数	Priority
JMS_IBM_MQMD_Persistence	メッセージの持続性	整数	Persistence
JMS_IBM_MQMD_MsgId [2]	メッセージ ID	Object (byte[]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	相関 ID	Object (byte[]) [4]	CorrelId
JMS_IBM_MQMD_BackoutCount	バックアウトのカウンター	整数	BackoutCount
JMS_IBM_MQMD_ReplyToQ	応答キューの名前	ストリング	ReplyToQ
JMS_IBM_MQMD_ReplyToQMgr	応答キュー・マネージャーの名前	ストリング	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	ユーザー ID	ストリング	UserIdentifier
JMS_IBM_MQMD_AccountingToken	アカウンティング・トークン	Object (byte[]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	ID に関連するアプリケーション・データ	ストリング	ApplIdentityData
JMS_IBM_MQMD_PutApplType	メッセージを書き込んだアプリケーションのタイプ	整数	PutApplType
JMS_IBM_MQMD_PutApplName	メッセージを書き込んだアプリケーションの名前	ストリング	PutApplName
JMS_IBM_MQMD_PutDate	メッセージを書き込んだ日付	ストリング	PutDate
JMS_IBM_MQMD_PutTime	メッセージを書き込んだ時刻	ストリング	PutTime
JMS_IBM_MQMD_ApplOriginData	発生元に関係するアプリケーション・データ	ストリング	ApplOriginData
JMS_IBM_MQMD_GroupId	グループ ID	Object (byte[]) [4]	GroupId
JMS_IBM_MQMD_MsgSeqNumber	グループ中の論理メッセージの順序番号	整数	MsgSeqNumber

表 126. プロパティーの名前、説明、およびタイプ (続き)

プロパティー	説明	Java 型	完全な説明のリンク先
JMS_IBM_MQMD_Offset	論理メッセージの先頭を起点とする、物理メッセージ中のデータのオフセット	整数	Offset
JMS_IBM_MQMD_MsgFlags	メッセージ・フラグ	整数	MsgFlags
JMS_IBM_MQMD_OriginalLength	元のメッセージの長さ	整数	OriginalLength

1. **重要 :** JMS_IBM_MQMD_Priority に 0 から 9 までの範囲に含まれない値を割り当てると、JMS 仕様違反になります。
2. **重要 :** JMS 仕様には、メッセージ ID は JMS プロバイダーによって設定されている必要があり、かつ固有であるかヌルである必要があると記されています。 JMS_IBM_MQMD_MsgId に値を割り当てると、この値は JMSMessageID にコピーされます。 このようにして、メッセージ ID が JMS プロバイダーによって設定されず、固有のものとならない場合もあります。これは JMS 仕様に違反します。
3. **重要 :** 'ID:' というストリングから始まる値を JMS_IBM_MQMD_CorrelId に割り当てると、これは JMS 仕様に違反します。
4. **重要 :** メッセージでバイト配列プロパティーを使用すると、JMS 仕様に違反します。

WebSphere MQ classes for JMS を使用したアプリケーションからの *IBM WebSphere MQ* メッセージ・データへのアクセス

IBM WebSphere MQ classes for JMS を使用して、アプリケーション内の完全な WebSphere MQ メッセージ・データにアクセスできます。 すべてのデータにアクセスするには、メッセージが JMSBytesMessage である必要があります。 JMSBytesMessage の本体には、MQRFH2 ヘッダー、その他の IBM WebSphere MQ ヘッダー、および以下のメッセージ・データが含まれています。

宛先の WMQ_MESSAGE_BODY プロパティーを WMQ_MESSAGE_BODY_MQ に設定して、JMSBytesMessage 内のすべてのメッセージ本体データを受信します。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS または WMQ_MESSAGE_BODY_UNSPECIFIED に設定されていると、メッセージ本体は JMS MQRFH2 ヘッダーなしで返され、JMSBytesMessage のプロパティーには RFH2 で設定されているプロパティーが反映されます。

一部のアプリケーションは、このトピックで説明している機能を使用できません。 アプリケーションが WebSphere MQ V6 キュー・マネージャーに接続されている場合、または PROVIDERVERSION が 6 に設定されている場合、これらの機能を使用できません。

メッセージの送信

メッセージの送信の際、宛先プロパティー WMQ_MESSAGE_BODY は WMQ_TARGET_CLIENT よりも優先されます。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS に設定されている場合、 WebSphere MQ classes for JMS は、 JMSMessage プロパティーおよびヘッダー・フィールドの設定に基づいて、自動的に MQRFH2 ヘッダーを生成します。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定されている場合、メッセージ本体に追加のヘッダーが加えられることはありません。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_UNSPECIFIED に設定されている場合、 WMQ_TARGET_CLIENT が WMQ_TARGET_DEST_MQ に設定されていない限り、 WebSphere MQ classes for JMS は MQRFH2 ヘッダーを送信します。 受信の際、WMQ_TARGET_CLIENT が WMQ_TARGET_DEST_MQ に設定されていると、あらゆる MQRFH2 がメッセージ本体から削除されます。

注 : JMSBytesMessage および JMSTextMessage では MQRFH2 は不要です。一方、JMSStreamMessage、JMSMapMessage、および JMSObjectMessage では必要となります。

WMQ_MESSAGE_BODY_UNSPECIFIED は WMQ_MESSAGE_BODY のデフォルト設定であり、WMQ_TARGET_DEST_JMS は WMQ_TARGET_CLIENT のデフォルト設定です。

JMSBytesMessage を送信する場合、WebSphere MQ メッセージの作成時に JMS メッセージ本体のデフォルト設定を指定変更することができます。 以下のプロパティーを使用します。

- JMS_IBM_Format または JMS_IBM_MQMD_Format: このプロパティーは、先行する Websphere MQ ヘッダーがない場合、WebSphere MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのフォーマットを指定します。
- JMS_IBM_Character_Set または JMS_IBM_MQMD_CodedCharSetId: このプロパティーは、先行する Websphere MQ ヘッダーがない場合に JMS メッセージ本体を開始する WebSphere MQ ヘッダーまたはアプリケーション・ペイロードの CCSID を指定します。
- JMS_IBM_Encoding または JMS_IBM_MQMD_Encoding: このプロパティーは、先行する Websphere MQ ヘッダーがない場合、WebSphere MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのエンコード方式を指定します。

両方のタイプのプロパティーを指定した場合、宛先プロパティー WMQ_MQMD_WRITE_ENABLED が true に設定されている限り、JMS_IBM_MQMD_* プロパティーによって、対応する JMS_IBM_* プロパティーが指定変更されます。

JMS_IBM_MQMD_* と JMS_IBM_* を使用したメッセージ・プロパティーの設定には、実際には大きな相違があります。

1. JMS_IBM_MQMD_* プロパティーは IBM WebSphere MQ JMS プロバイダーに固有のプロパティーです。
2. JMS_IBM_MQMD_* プロパティーは、MQMD でのみ設定されます。 JMS_IBM_* プロパティーは、メッセージに MQRFH2 JMS ヘッダーが含まれていない場合にのみ、MQMD で設定されます。 それ以外の場合は、JMS RFH2 ヘッダーで設定されます。
3. JMS_IBM_MQMD_* プロパティーは JMSMessage に書き込まれたテキストおよび数値のエンコード方式には影響しません。

   受信側のアプリケーションは、MQMD.Encoding と MQMD.CodedCharSetId の値が、メッセージ本体内の数値およびテキストのエンコード方式と文字セットに対応していると見なす場合があります。 JMS_IBM_MQMD_* プロパティーを使用する場合、送信側アプリケーションでそのように設定する必要があります。 メッセージ本体内の数値およびテキストのエンコード方式と文字セットは、JMS_IBM_* プロパティーで設定します。

   907 ページの図 163 の不適切にコード化されたスニペットは、文字セット 1208 でエンコードされ、MQMD.CodedCharSetId が 37 に設定されたメッセージを送信します。

a. 誤ってエンコードされたメッセージの送信

```
TextMessage tmo = session.createTextMessage();
((MQDestination) destination).setMessageBodyStyle
                 (WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQDestination)destination).setMQMDWriteEnabled(true);
tmo.setIntProperty(WMQConstants.JMS_IBM_MQMD_CODEDCHARSETID, 37);
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 1208);
tmo.setText("String one");
producer.send(tmo);
```

b. MQMD.CodedCharSetId の値によって設定された JMS_IBM_CHARACTER_SET の値に依存した、メッセージの受信

```
TextMessage tmi = (TextMessage) cons.receive();
System.out.println("Message is \"" + tmi.getText() + "\"");
```

c. 結果としての出力:

```
Message is "ëÈÊ'>...??>?"
```

図 *163.* 一貫性なくコード化された *MQMD* とメッセージ・データ

907 ページの図 164 のコードのいずれかのスニペットは、自動的に生成される MQRFH2 ヘッダーが追加されることなく、メッセージ本体にアプリケーション・ペイロードが含まれている状態で、メッセージがキューまたはトピックに置かれる結果となります。

1. WMQ_MESSAGE_BODY_MQ の設定:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. WMQ_TARGET_DEST_MQ の設定:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

図 *164. MQ* メッセージ本体が含まれたメッセージの送信

メッセージの受信

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_JMS に設定されている場合、インバウンド JMS メッセージ・タイプおよび本体は、受信される Websphere MQ メッセージの内容によって判別されます。 このメッセージ・タイプおよび本体は、MQRFH2 ヘッダー内のフィールドによって決定されます。または、MQRFH2 がない場合は MQMD 内のフィールドによって決定されます。

WMQ_MESSAGE_BODY が WMQ_MESSAGE_BODY_MQ に設定されている場合、インバウンド JMS メッセージ・タイプは JMSBytesMessage です。 JMS メッセージ本体は、基礎となる MQGET API 呼び出しによって戻されるメッセージ・データです。 メッセージ本体の長さは、MQGET 呼び出しによって戻された長さです。メッセージ本体のデータの文字セットおよびエンコード方式は、MQMD の CodedCharSetId および Encoding フィールドによって判別されます。 メッセージ本体内のデータのフォーマットは、MQMD の Format フィールドによって判別されます。

WMQ_MESSAGE_BODY が WMQ_WMQ_MESSAGE_BODY_UNSPECIFIED に設定されている場合 (デフォルト値)、IBM WebSphere MQ classes for JMS がそれを WMQ_MESSAGE_BODY_JMS に設定します。

JMSBytesMessage を受信したならば、以下のプロパティーを参照してデコードすることができます。

- JMS_IBM_Format または JMS_IBM_MQMD_Format: このプロパティーは、先行する Websphere MQ ヘッダーがない場合、WebSphere MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのフォーマットを指定します。
- JMS_IBM_Character_Set または JMS_IBM_MQMD_CodedCharSetId: このプロパティーは、先行する Websphere MQ ヘッダーがない場合に JMS メッセージ本体を開始する WebSphere MQ ヘッダーまたはアプリケーション・ペイロードの CCSID を指定します。
- JMS_IBM_Encoding または JMS_IBM_MQMD_Encoding: このプロパティーは、先行する Websphere MQ ヘッダーがない場合、WebSphere MQ ヘッダーまたは JMS メッセージ本体を開始するアプリケーション・ペイロードのエンコード方式を指定します。

以下のコード・スニペットの結果は、受信されたメッセージ (JMSBytesMessage) となります。 受信されたメッセージ、および受信された MQMD のフォーマット・フィールドの内容に関係なく、メッセージは JMSBytesMessage となります。

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

宛先プロパティー *WMQ_MESSAGE_BODY*
WMQ_MESSAGE_BODY は、JMS アプリケーションが WebSphere MQ メッセージの MQRFH2 を、メッセージ・ペイロードの一部として (すなわち、JMS メッセージ本体の一部として) 処理するかどうかを決定します。

表 *127.* プロパティーの名前および説明

プロパティー	短縮形	説明
WMQ_MESSAGE_BODY	MBODY	JMS アプリケーションが WebSphere MQ メッセージの MQRFH2 を、メッセージ・ペイロードの一部として (すなわち、JMS メッセージ本体の一部として) 処理するかどうか。

表 128. プロパティーの名前、値、および設定方式

プロパティー	管理ツール内での有効値 (太字はデフォルト)	プログラム内での有効値	Set メソッド
WMQ_ メッセージ _BODY	• **UNSPECIFIED** 送信の際に、WebSphere MQ classes for JMS が MQRFH2 ヘッダーを生成して組み込むかどうかは、WMQ_TARGET_CLIENT の値によって異なります。 受信の際には、JMS の値に従って作動します。 • JMS 送信の際に、WebSphere MQ classes for JMS は自動的に MQRFH2 ヘッダーを生成して WebSphere MQ メッセージに組み込みます。 受信の際に、WebSphere MQ classes for JMS は、MQRFH2 (存在する場合) の値に従って JMS メッセージ・プロパティーを設定します。MQRFH2 を JMS メッセージ本体の一部として提示することはしません。 • MQ 送信の際に、WebSphere MQ classes for JMS は MQRFH2 を生成しません。 受信の際に、WebSphere MQ classes for JMS は MQRFH2 を JMS メッセージ本体の一部として表示します。	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS 持続メッセージ

WebSphere MQ classes for JMS アプリケーションでは、**NonPersistentMessageClass** キュー属性を使用して JMS 持続メッセージのパフォーマンスを向上させることができます。 ただし、信頼性は多少低下します。

WebSphere MQ キューには、**NonPersistentMessageClass** という名前の属性があります。 キュー・マネージャーが再始動するときにキュー上の非持続メッセージが破棄されるかどうかは、この属性の値によって決まります。

ローカル・キューに対してこの属性を設定するには、WebSphere MQ スクリプト・コマンド (MQSC) である DEFINE QLOCAL を、以下のいずれかのパラメーターと共に使用します。

NPMCLASS(NORMAL)
キュー・マネージャーが再始動するとき、キュー上の非持続メッセージは破棄されます。 これはデフォルト値です。

NPMCLASS(HIGH)
キュー・マネージャーが静止シャットダウンまたは即時シャットダウンの後で再始動するとき、キュー上の非持続メッセージは破棄されません。 ただし、優先シャットダウンや障害の後には、非持続メッセージが破棄される可能性があります。

このトピックでは、WebSphere MQ classes for JMS アプリケーションでこのキュー属性を使用することによって JMS 持続メッセージのパフォーマンスを向上させる方法について説明します。

Queue オブジェクトまたは Topic オブジェクトの PERSISTENCE プロパティーには、HIGH という値を使用できます。 この値は WebSphere MQ JMS 管理ツールを使用して設定できます。また、アプリケーションで Destination.setPersistence() メソッドを呼び出して、値 WMQConstants.WMQ_PER_NPHIGH をパラメーターとして渡すこともできます。

PERSISTENCE プロパティーの値が HIGH の宛先にアプリケーションから JMS 持続メッセージまたは JMS 非持続メッセージが送信されたときに、基礎 WebSphere MQ キューが NPMCLASS(HIGH) に設定されていると、送信されたメッセージは、WebSphere MQ 非持続メッセージとしてそのキューに入れられます。 宛先の PERSISTENCE プロパティーの値が HIGH ではない場合、または基礎キューが NPMCLASS(NORMAL) に設定されている場合には、JMS 持続メッセージは WebSphere MQ 持続メッセージとしてキューに入れられ、JMS 非持続メッセージは WebSphere MQ 非持続メッセージとしてキューに入れられます。

JMS 持続メッセージが WebSphere MQ 非持続メッセージとしてキューに入れられる場合にキュー・マネージャーの静止シャットダウンや即時シャットダウンの後にそのようなメッセージが破棄されないようにするには、メッセージが経路指定されるすべてのキューを NPMCLASS(HIGH) に設定する必要があります。 パブリッシュ/サブスクライブ・ドメインでは、これらのキューにサブスクライバー・キューが含まれます。 アプリケーションで、PERSISTENCE プロパティーの値が HIGH の宛先に対するメッセージ・コンシューマーの作成が試行され、基礎となる WebSphere MQ キューが NPMCLASS(NORMAL) に設定されると、この構成の実行を補助するために WebSphere MQ classes for JMS から InvalidDestinationException がスローされます。

宛先の PERSISTENCE プロパティーを HIGH に設定しても、その宛先でメッセージが受信される方法には影響しません。 JMS 持続メッセージとして送信されたメッセージは JMS 持続メッセージとして受信され、JMS 非持続メッセージとして送信されたメッセージは JMS 非持続メッセージとして受信されます。

アプリケーションで、PERSISTENCE プロパティーの値が HIGH の宛先に最初のメッセージが送信される、または PERSISTENCE プロパティーの値が HIGH の宛先に対して最初のメッセージ・コンシューマーが作成されると、WebSphere MQ classes for JMS は MQINQ 呼び出しを発行して、基礎となる WebSphere MQ キュー上で NPMCLASS(HIGH) が設定されているかどうかを判断します。 したがって、アプリケーションは、キューに対して照会を行う権限を持っていなければなりません。 また、WebSphere MQ classes for JMS は、該当の宛先が削除されるまで MQINQ 呼び出しの結果を保持します。追加の MQINQ 呼び出しを実行することはありません。 したがって、アプリケーションでその宛先がまだ使用されているにもかかわらず基礎キューの NPMCLASS 設定を変更した場合、WebSphere MQ classes for JMS ではその新しい設定を認識できません。

JMS 持続メッセージを WebSphere MQ 非持続メッセージとして WebSphere MQ キューに入れることができるようにすると、パフォーマンスが向上します。ただし、信頼性は多少低下します。 JMS 持続メッセージの信頼性を最大にする必要がある場合は、PERSISTENCE プロパティーの値が HIGH である宛先にはメッセージを送信しないでください。

JMS レイヤーでは、SYSTEM.DEFAULT.MODEL.QUEUE の代わりに SYSTEM.JMS.TEMPQ.MODEL を使用できます。 SYSTEM.DEFAULT.MODEL.QUEUE では持続メッセージを受け入れることができないため、SYSTEM.JMS.TEMPQ.MODEL は持続メッセージを受け入れる永続動的キューを作成します。 したがって、一時キューを使用して持続メッセージを受け入れる場合は、SYSTEM.JMS.TEMPQ.MODEL を使用するか、またはモデル・キューを、自分で選択した代わりのキューに変更する必要があります。

WebSphere MQ classes for JMS での Secure Sockets Layer (SSL) の使用

WebSphere MQ classes for JMS アプリケーションは、SSL 暗号化を使用できます。 これを実行するには、JSSE プロバイダーが必要です。

TRANSPORT(CLIENT) を使用した WebSphere MQ classes for JMS 接続では、Secure Sockets Layer (SSL) 暗号化がサポートされます。 SSL は、通信の暗号化、認証、およびメッセージの整合性を提供します。 これは一般的に、インターネット上やイントラネット内で、任意の 2 つのピアの間の通信を保護するために使用されます。

WebSphere MQ classes for JMS は、Java Secure Socket Extension (JSSE) を使用して SSL 暗号化を処理するため、JSSE プロバイダーを必要とします。 JSE v1.4 JVM には、JSSE プロバイダーが組み込まれています。 証明書を管理して保管する方法は、プロバイダーごとに違います。 詳細については、各 JSSE プロバイダーの資料を参照してください。

この節では、JSSE プロバイダーが正常にインストールおよび構成されていることと、適切な証明書がユーザーの JSSE プロバイダーにインストールされて使用可能であることを前提としています。 つまり、JMSAdmin を使用して いくつかの管理プロパティーを設定する準備が完了しています。

WebSphere MQ classes for JMS アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合は、919 ページの『IBM WebSphere MQ classes for JMS でのクライアント・チャネル定義テーブルの使用』を参照してください。

SSLCIPHERSUITE オブジェクト・プロパティー
SSLCIPHERSUITE を設定して、ConnectionFactory オブジェクトに対する SSL 暗号化を使用可能にします。

ConnectionFactory オブジェクトで SSL 暗号化を使用可能にするには、JMSAdmin を使用して、SSLCIPHERSUITE プロパティーを JSSE プロバイダーによってサポートされている CipherSuite に設定します。 これは、宛先チャネルで設定された CipherSpec と一致していなければなりません。 ただし、CipherSuites は CipherSpec とは異なるため、異なる名前が付けられています。 914 ページの『JMS での SSL CipherSpecs と CipherSuites』には、WebSphere MQ にサポートされている CipherSpecs を、JSSE に知られている同等な CipherSuites にマップする表を示します。 WebSphere MQ での CipherSpecs および CipherSuites の詳細については、セキュリティーを参照してください。

例えば、RC4_MD5_EXPORT の CipherSpec を使用して SSL 対応の MQI チャネルに接続するよう ConnectionFactory オブジェクトを設定するには、JMSAdmin に対して次のコマンドを発行します。

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

これは、MQConnectionFactory オブジェクトで setSSLCipherSuite() メソッドを使用することにより、プログラムから設定することも可能です。

CipherSpec が SSLCIPHERSUITE プロパティーで指定されている場合、JMSAdmin は便宜上 CipherSpec を適切な CipherSuite に割り当てようとして警告を発行します。 アプリケーションがプロパティーを指定している場合は、このようなマッピングは試みられません。

または、クライアント・チャネル定義テーブル (CCDT) を使用します。 詳細については、919 ページの『IBM WebSphere MQ classes for JMS でのクライアント・チャネル定義テーブルの使用』を参照してください。

SSLFIPSREQUIRED オブジェクト・プロパティー
IBM Java JSSE FIPS プロバイダー (IBMJSSEFIPS) によってサポートされる CipherSuite を使用するために接続が必要な場合は、接続ファクトリーの SSLFIPSREQUIRED プロパティーを YES に設定します。

このプロパティーのデフォルト値は NO です。つまり、接続で WebSphere MQ によってサポートされる任意の CipherSuite を使用することができます。

アプリケーションが複数の接続を使用する場合、アプリケーションが最初の接続を作成するときに使用される SSLFIPSREQUIRED の値によって、アプリケーションが以降のすべての接続を作成するときに使用される値が決まります。 これは、以降の接続の作成時に使用される接続ファクトリーの SSLFIPSREQUIRED プロパティーの値が無視されることを意味します。 別の SSLFIPSREQUIRED 値を使用する場合は、アプリケーションを再始動する必要があります。

アプリケーションでは、ConnectionFactory オブジェクトの setSSLFipsRequired() メソッドを呼び出すことで、このプロパティーを設定できます。 CipherSuite が設定されていない場合、このプロパティーは無視されます。

関連タスク
MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する
関連資料
UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

SSLPEERNAME オブジェクト・プロパティー
SSLPEERNAME を使用して識別名パターンを指定し、JMS アプリケーションが確実に正しいキュー・マネージャーに接続するようにします。

JMS アプリケーションでは、識別名 (DN) パターンを指定することにより、正しいキュー・マネージャーに接続していることが確認できます。 キュー・マネージャーがパターンと一致する DN を提示する場合にのみ、接続は成功します。 このパターンの形式の詳細については、関連トピックを参照してください。

DN は、ConnectionFactory オブジェクトの SSLPEERNAME プロパティーを使用して設定されます。 例えば、次の JMSAdmin コマンドは、ConnectionFactory オブジェクトが QMGR. で始まる共通名、および少なくとも 2 つの組織単位名、最初は IBM、次は WEBSPHERE を使用して、キュー・マネージャーを識別するように設定します。

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

検査は大文字と小文字を区別しないで行われます。またコンマの代わりにセミコロンを使用できます。 SSLPEERNAME は、MQConnectionFactory オブジェクトで setSSLPeerName() メソッドを使用することにより、アプリケーションから設定することも可能です。 このプロパティーが設定されない場合、キュー・マネージャーで提供される識別名 (DN) に対して検査は実行されません。 CipherSuite が設定されなければ、このプロパティーは無視されます。

SSLCERTSTORES オブジェクト・プロパティー
SSLCERTSTORES を使用して、証明書取り消しリスト (CRL) の検査に使用する LDAP サーバーのリストを指定します。

非トラステッドになった証明書を識別するには、証明書取り消しリスト (CRL) が一般的に使用されます。 多くの場合 CRL は、LDAP サーバーにホストされています。 JMS では、Java 2 v1.4 以降での CRL 検査のために LDAP サーバーを指定できます。 次の JMSAdmin の例では、crl1.ibm.com という LDAP サーバーにホストされている CRL を使用するよう JMS に指示しています。

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

注 : LDAP サーバーでホストされている CRL を使用して CertStore を正常に使用するには、ご使用の Java Software Development Kit (SDK) が CRL と互換性があることを確認してください。 SDK によっては、CRL が LDAP v2 用のスキーマを定義する RFC 2587 に準拠している必要があります。 ほとんどの LDAP v3 サーバーは、代わりに RFC 2256 を使用しています。

使用している LDAP サーバーがデフォルト・ポートの 389 で稼働していない場合、ホスト名にコロン (:) とポート番号を追加して、そのポートを指定できます。 キュー・マネージャーによって提示される証明書が、crl1.ibm.com にホストされている CRL に提示される場合、接続は完了しません。 Single Point of Failure を避けるため、JMS では、スペース文字で区切った LDAP サーバーのリストを提供することにより、複数の LDAP サーバーを指定できます。 以下が例となります。

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

複数の LDAP サーバーを指定すると、JMS は、キュー・マネージャーの証明書を正常に検査できるサーバーを見つけるまで、各サーバーを順番に試行します。 各サーバーには、同一の情報が含まれている必要があります。

この形式の文字列は、MQConnectionFactory.setSSLCertStores() メソッドで、アプリケーションにより提供することができます。 別の方法として、アプリケーションで 1 つ以上の java.security.cert.CertStore オブジェクトを作成し、適切な Collection オブジェクトに配置し、この Collection オブジェクトを setSSLCertStores() メソッドに提供することができます。 この方法で、アプリケーションは CRL 検査をカスタマイズすることができます。 CertStore オブジェクトの組み立て方法と使用方法の詳細については、ご使用の JSSE の資料を参照してください。

接続を設定するときにキュー・マネージャーによって示される証明書は、以下のようにして検査されます。

1. sslCertStores によって識別される Collection にある最初の CertStore オブジェクトを使用し、CRL サーバーを識別します。
2. CRL サーバーへ接続してみます。
3. 接続が成功すれば、一致する証明書をサーバー内で検索します。

a. 証明書が失効したことが分かった場合、検索プロセスは終了して接続要求は失敗します。理由コードは MQRC_SSL_CERTIFICATE_REVOKED です。

b. 証明書が見つからない場合、検索プロセスは終了し、接続を先に進めることができます。

4. サーバーへの接続が失敗する場合、次の CertStore オブジェクトを使用して、CRL サーバーを識別します。プロセスはステップ 2 から繰り返されます。

   これが Collection の最後の CertStore であるか、Collection に CertStore オブジェクトが含まれない場合、検索プロセスは失敗し、接続要求は失敗します。理由コードは MQRC_SSL_CERT_STORE_ERROR です。

Collection オブジェクトは、CertStores を使用する順序を決定します。

アプリケーションが setSSLCertStores() を使用して CertStore オブジェクトの Collection を設定する場合、MQConnectionFactory を JNDI 名前空間にバインドできなくなります。 バインドしようとすると、例外が発生します。 sslCertStores プロパティーが設定されない場合、キュー・マネージャーで提供される証明書に対する失効の検査は実行されません。 CipherSuite が設定されなければ、このプロパティーは無視されます。

SSLRESETCOUNT オブジェクト・プロパティー
このプロパティーは、暗号化に使用された秘密鍵が再ネゴシエーションされる前に、接続で送信および受信したバイトの総数を表します。

送信バイト数は暗号化前の数であり、受信バイト数は暗号化解除された後の数です。 このバイト数には、WebSphere MQ classes for JMS で送受信された制御情報も含まれます。

例えば、SSL 対応の MQI チャネル (このチャネルの秘密鍵は、4 MB のデータが流れた後再ネゴシエーションされる) を介した接続の作成に使用できる ConnectionFactory オブジェクトを構成するには、JMSAdmin に対して次のコマンドを発行します。

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

アプリケーションでは、ConnectionFactory オブジェクトの setSSLResetCount() メソッドを呼び出すことで、このプロパティーを設定できます。

このプロパティーの値がゼロ (デフォルト値) の場合、秘密鍵の再ネゴシエーションは行われません。CipherSuite が設定されていない場合、このプロパティーは無視されます。

SSLSocketFactory オブジェクト・プロパティー
アプリケーション用に SSL 接続の他の面をカスタマイズするには、SSLSocketFactory を作成し、JMS がそれを使用するように構成します。

特定アプリケーションのために、SSL 接続の他の面をカスタマイズできます。 例えば、暗号ハードウェアを初期設定したり、使用中の鍵ストアおよびトラストストアを変更したりしたい場合があります。 そのためには、アプリケーションで、それに従ってカスタマイズされた javax.net.ssl.SSLSocketFactory オブジェクトをまず作成する必要があります。 カスタマイズできる機能はプロバイダーによって異なるため、インスタンスを作成する方法についてはご使用の JSSE の資料を参照してください。 適切な SSLSocketFactory オブジェクトを入手したら、MQConnectionFactory.setSSLSocketFactory() メソッドを使用して、カスタマイズした SSLSocketFactory オブジェクトを使用するよう JMS を構成します。

アプリケーションが setSSLSocketFactory() メソッドを使用してカスタマイズした SSLSocketFactory オブジェクトを設定する場合、MQConnectionFactory オブジェクトは JNDI ネーム・スペースにバインドできなくなります。 バインドしようとすると、例外が発生します。 このプロパティーが設定されていなければ、デフォルトの SSLSocketFactory オブジェクトが使用されます。 デフォルトの SSLSocketFactory オブジェクトの振る舞いの詳細については、ご使用の JSSE の資料を参照してください。 CipherSuite が設定されなければ、このプロパティーは無視されます。

重要 : それ自体が保護されていない JNDI ネーム・スペースから ConnectionFactory オブジェクトを取り出す場合、SSL プロパティーを使用してもセキュリティーが保証されるとは考えないでください。 特に、JNDI を標準的に LDAP 実装しても保護にはなりません。 アタッカーは LDAP サーバーをだませるため、JMS アプリケーションはだまされたと気付かずに間違ったサーバーに接続する可能性があります。 適切なセキュリティー配置がなされていれば、JNDI の他の実装 (fscontext 実装など) は保護されます。

JSSE 鍵ストアまたは *JSSE* トラストストアの変更
鍵ストアまたはトラストストアに変更を加える場合、選出する変更に対して特定のアクションを取る必要があります。

JSSE 鍵ストアや JSSE トラストストアの内容を変更したり、鍵ストア・ファイルやトラストストア・ファイルの位置を変更したりした場合、その時点で実行中の WebSphere MQ classes for JMS アプリケーションはその変更を自動的には認識しません。 変更を有効にするには、以下のアクションを行う必要があります。

- アプリケーションは、すべての接続をクローズし、接続プール内の未使用の接続を破棄する必要がある。
- JSSE プロバイダーが鍵ストアおよびトラストストアからの情報をキャッシュしている場合は、この情報をリフレッシュする必要がある。

これらのアクションが完了すると、アプリケーションは接続を再作成できます。

アプリケーションの設計方法や、JSSE プロバイダーが提供する機能によっては、アプリケーションを停止および再始動しなくても上記のアクションを実行できる場合があります。 ただし、アプリケーションを停止して再始動するのが最も簡単な解決方法です。

JMS での *SSL CipherSpecs* と *CipherSuites*
WebSphere MQ によってサポートされる CipherSpec およびそれらと同等の CipherSuite

914 ページの表 129 に、WebSphere MQ によってサポートされる CipherSpecs と、それらと同等の CipherSuites を示します。 ConnectionFactory プロパティー SSLFIPSREQUIRED が NO に設定されている場合、WebSphere MQ classes for JMS アプリケーションは、サポートされている CipherSpec が MQI チャネルのサーバー側で指定され、それと同等の CipherSuite がクライアント側で指定されている場合、キュー・マネージャーに接続できます。 SSLFIPSREQUIRED が YES に設定されている場合、CipherSpec と CipherSuite の組み合わせによって、アプリケーションがキュー・マネージャーに接続できるかどうかが決まります。

MQI チャネルのサーバー側では、 CipherSpec の名前を DEFINE CHANNEL CHLTYPE(SVRCONN) コマンドの SSLCIPH パラメーター の値として指定できます。 MQI チャネルのクライアント側では、CipherSuite の名前を次のように指定できます。

- アプリケーションでは、ConnectionFactory オブジェクトの setSSLCipherSuite() メソッドを呼び出すことができます。
- WebSphere MQ JMS 管理ツールを使用すると、ConnectionFactory オブジェクトの SSLCIPHERSUITE プロパティーを設定できます。

表 *129. WebSphere MQ* によってサポートされる *CipherSpec* およびそれらと同等の *CipherSuite*

CipherSpec	**同等の CipherSuite**	**SFIPS [1] が YES に設定されている場合、接続は可能か**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	No
NULL_SHA	SSL_RSA_WITH_NULL_SHA	No
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	No
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	No
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	No
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	No
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	No
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	No
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	No
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	No

表 129. WebSphere MQ によってサポートされる CipherSpec およびそれらと同等の CipherSuite (続き)		
CipherSpec	**同等の CipherSuite**	**SFIPS [1] が YES に設定されている場合、接続は可能か**
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	非該当 [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	はい [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	はい [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	はい [5] [7]
AES_SHA_US [2]		
TLS_RSA_WITH_DES_CBC_SHA [8] [9]	SSL_RSA_WITH_DES_CBC_SHA	いいえ [3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA [8]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	いいえ [4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	いいえ [6]

注 :

1. WebSphere MQ JMS 管理ツールを使用している場合、SFIPS は ConnectionFactory プロパティー SSLFIPSREQUIRED の短縮名です。
2. この CipherSpec には、同等の CipherSuite はありません。
3. この CipherSpec は、2007 年 5 月 19 日より前は FIPS 140-2 で認証されていました。
4. この CipherSpec は、2007 年 5 月 19 日より前は FIPS 140-2 で認証されていました。 FIPS_WITH_DES_CBC_SHA という名前は歴史的な事情によるものであり、この CipherSpec がかつては FIPS 準拠であった (現在はそうではない) という事実を反映するものです。 この CipherSpec は非推奨となりました。使用することはお勧めしません。
5. WebSphere MQ エクスプローラーが使用する JRE に対して適切な無制限のポリシー・ファイルが適用されていない場合には、 これらの CipherSpec (TLS_RSA_WITH_AES_128_CBC_SHA、 TLS_RSA_WITH_AES_256_CBC_SHA、TLS_RSA_WITH_AES_256_CBC_SHA256) を使用してエクスプローラーからキュー・マネージャーへの安全な接続を確立することはできません。

   ポリシー・ファイルの詳細については、Security information を参照してください。
6. FIPS_WITH_3DES_EDE_CBC_SHA という名前は歴史的な事情によるものであり、この CipherSpec がかつては FIPS 準拠であった (現在はそうではない) という事実を反映するものです。 この CipherSpec は非推奨となりました。使用することはお勧めしません。
7. これらの CipherSpec (TLS_RSA_WITH_NULL_SHA256、TLS_RSA_WITH_AES_128_CBC_SHA、 TLS_RSA_WITH_AES_256_CBC_SHA、TLS_RSA_WITH_AES_256_CBC_SHA256) には、 IBM JRE 6.0 SR13 FP2、7.0 SR4 FP2 またはそれ以上が必要です。
8. これらの CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA、TLS_RSA_WITH_DES_CBC_SHA、 TLS_RSA_WITH_RC4_128_SHA256) は、SSLv3 または TLS を使用できます。 デフォルトでは、FIPS が有効でなければ、SSLv3 が使用されます。 TLS を使用するには、Java System Property **com.ibm.mq.cfg.preferTLS** を `true` に設定します。
9. この CipherSpec の TLS_RSA_WITH_3DES_EDE_CBC_SHA は推奨されません。 ただし、32 GB 以下のデータの転送にはまだ使用できますが、これを超えるとエラー AMQ9288 を出して接続が終了します。 このエラーを回避するために、Triple-DES を使用しないか、またはこの CipherSpec を使用する際に秘密鍵リセットを有効にする必要があります。

関連情報

MQI クライアントでの実行時に FIPS 認定の CipherSpec のみを使用するように指定する

UNIX、 Linux 、および Windows の連邦情報処理標準 (FIPS)

Java for WebSphere MQ classes for JMS でのチャネル出口の作成

チャネル出口は、指定されたインターフェースを実装する Java クラスを定義することによって作成します。

com.ibm.mq.exits パッケージには、次の 3 つのインターフェースが定義されています。

- WMQSendExit (送信出口用)
- WMQReceiveExit (受信出口用)
- WMQSecurityExit (セキュリティー出口用)

以下のサンプル・コードは、3 つのインターフェースすべてを実装するクラスを定義しています。

```
public class MyMQExits implements
WMQSendExit, WMQReceiveExit, WMQSecurityExit {
    // Default constructor
  public MyMQExits(){
  }
    // This method implements the send exit interface
  public ByteBuffer channelSendExit(
MQCXP channelExitParms,
                                  MQCD channelDefinition,
                                  ByteBuffer agentBuffer)
  {
    // Complete the body of the send exit here
  }
    // This method implements the receive exit interface
  public ByteBuffer channelReceiveExit(
MQCXP channelExitParms,
                                    MQCD channelDefinition,
                                    ByteBuffer agentBuffer)
  {
    // Complete the body of the receive exit here
  }
    // This method implements the security exit interface
  public ByteBuffer channelSecurityExit(
MQCXP channelExitParms,
                                     MQCD channelDefinition,
                                     ByteBuffer agentBuffer)
  {
    // Complete the body of the security exit here
  }
}
```

各出口は、MQCXP オブジェクトおよび MQCD オブジェクトをパラメーターとして受け取ります。 これらのオブジェクトは、プロシージャー型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

送信出口が呼び出される場合、agentBuffer パラメーターには、サーバー・キュー・マネージャーに送信される直前のデータが入ります。 データの長さは式 agentBuffer.limit() で指定されるため、長さパラメーターは不要です。 送信出口は、サーバー・キュー・マネージャーに送信するデータを値として返します。 しかし、この送信出口が一連の送信出口の最後の送信出口でない場合は、返されたデータは一連の送信出口の次のものに渡されます。 送信出口は、agentBuffer パラメーターに受け取るデータを変更して返すこともできますし、変更せずに返すこともできます。 それで、考えられる最も単純な出口の本体は次のとおりです。

```
{ return agentBuffer; }
```

受信出口が呼び出される場合、agentBuffer パラメーターには、サーバー・キュー・マネージャーから受け取ったデータが入ります。 受信出口は、WebSphere MQ classes for JMS がアプリケーションに渡すデータを値として返します。 しかし、この受信出口が一連の受信出口の最後の受信出口でない場合は、返されたデータは一連の受信出口の次のものに渡されます。

セキュリティー出口が呼び出される場合、agentBuffer パラメーターには、接続のサーバー側のセキュリティー出口からのセキュリティー・フローで受け取ったデータが入ります。 セキュリティー出口は、サーバーのセキュリティー出口へのセキュリティー・フローで送信するデータを値として返します。

チャネル出口は、バッキング配列を持つバッファーとともに呼び出されます。 最良のパフォーマンスを得るためには、出口はバッキング配列を持つバッファーを戻す必要があります。

チャネル出口を呼び出すときには、32 文字までのユーザー・データを渡すことができます。 出口はこのユーザー・データに MQCXP オブジェクトの getExitData() メソッドを呼び出すことによってアクセスします。出口は setExitData() メソッドを呼び出してユーザー・データを変更できますが、ユーザー・データは出口が呼び出されるごとにリフレッシュされます。 したがって、ユーザー・データに加えられた変更はすべて失われます。 しかし、出口は MQCXP オブジェクトの出口ユーザー域を使用して、ある呼び出しから次の呼び出しにデータを渡すことができます。 出口は getExitUserArea() メソッドを呼び出して、出口ユーザー域に参照でアクセスします。

すべての出口クラスには、コンストラクターがなければなりません。 コンストラクターは、前の例に示したようにデフォルト・コンストラクターにすることも、ストリング・パラメーターのあるコンストラクターにすることもできます。 コンストラクターは呼び出されて、クラス内に定義されている出口ごとに出口クラスのインスタンスを作成します。 したがって、前出の例の場合、送信出口のために MyMQExits クラスの 1 つのインスタンスが作成され、受信出口のためにもう 1 つのインスタンスが作成され、セキュリティー出口のために 3 番目のインスタンスが作成されます。 ストリング・パラメーターのあるコンストラクターが呼び出される場合、インスタンスが作成されるチャネル出口に渡されるのと同じユーザー・データがそのパラメーターに入ります。 出口クラスにデフォルト・コンストラクターと単一パラメーターを持つコンストラクターの両方がある場合、単一パラメーターを持つコンストラクターが優先されます。

接続はチャネル出口内から閉じないでください。

接続のサーバー側にデータが送信される際には、SSL 暗号化はチャネル出口が呼び出された後で 実行されます。 同様に、接続のサーバー側からデータを受信する際には、SSL 暗号化解除はチャネル出口が呼び出される前に 実行されます。

WebSphere MQ classes for JMS のバージョン 7.0 より前のバージョンでは、チャネル出口はインターフェース MQSendExit、MQReceiveExit、および MQSecurityExit を使用して実装していました。 これらのインターフェースは今でも使用できますが、機能とパフォーマンスが向上しているので、新しいインターフェースの方が望ましいです。

チャネル出口を使用するように *IBM WebSphere MQ classes for JMS* を構成する

IBM WebSphere MQ classes for JMS のアプリケーションは、キュー・マネージャーに接続したときに開始する MQI チャネル上のチャネル・セキュリティー出口、送信出口、受信出口を使用することができます。アプリケーションは、Java、C、または C++ で作成された出口を使用することができます。また、アプリケーションは、連続して実行される一連の送信出口または受信出口を使用することもできます。

以下のプロパティーを使用して、JMS 接続で使用する 1 つの送信出口か一連の送信出口を指定します。

- MQConnectionFactory オブジェクトの **SENDEXIT** プロパティー。
- IBM WebSphere MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **sendexit** プロパティー。
- 出力通信用に IBM WebSphere MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **sendexit** プロパティー。

このプロパティーの値は、コンマで区切った 1 つ以上の項目からなるストリングです。 各項目には、以下の方法のいずれかで 1 つの送信出口を指定します。

- Java で作成された送信出口の `WMQSendExit` インターフェースを実装するクラスの名前。
- C または C++ で作成された送信出口用の *libraryName*(*entryPointName*) という形式のストリング。

同様に、以下のプロパティーでは、接続で使用する 1 つの受信出口か一連の受信出口を指定します。

- MQConnectionFactory オブジェクトの **RECEXIT** プロパティー。
- IBM WebSphere MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **receiveexit** プロパティー。
- 出力通信用に IBM WebSphere MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **receiveexit** プロパティー。

以下のプロパティーでは、接続で使用するセキュリティー出口を指定します。

- MQConnectionFactory オブジェクトの **SECEXIT** プロパティー。

- IBM WebSphere MQ リソース・アダプターがインバウンド通信に使用するアクティベーション・スペックの **securityexit** プロパティー。
- 出力通信用に IBM WebSphere MQ リソース・アダプターによって使用される ConnectionFactory オブジェクトの **securityexit** プロパティー。

MQConnectionFactories の場合、**SENDEXIT**、**RECEXIT**、および **SECEXIT** プロパティーは、IBM WebSphere MQ JMS 管理ツールまたは IBM WebSphere MQ Explorer を使うことによって設定できます。 あるいは、アプリケーションから、`setSendExit()` メソッド、`setReceiveExit()` メソッド、および `setSecurityExit()` メソッドを呼び出すことによって、これらのプロパティーを設定できます。

チャネル出口はその独自のクラス・ローダーによってロードされます。 チャネル出口を見つけるために、クラス・ローダーは指定された順序で下記の場所を検索します。

1. プロパティー **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** によって指定されたクラスパス、または IBM WebSphere MQ クライアント構成ファイルの Channels スタンザの **JavaExitsClassPath** 属性によって指定されたクラスパス。
2. Java システム・プロパティー **com.ibm.mq.exitClasspath** で指定されたクラス・パス。 このプロパティーは現在、推奨されていません。
3. IBM WebSphere MQ 出口ディレクトリー (918 ページの表 130 を参照)。 最初に、クラス・ローダーは、Java アーカイブ (JAR) ファイルにパッケージされていないクラス・ファイルを求めてディレクトリーを検索します。 チャネル出口が見つからなかった場合、クラス・ローダーは次にディレクトリー内の JAR ファイルを検索します。

表 *130. IBM WebSphere MQ* 出口ディレクトリー

プラットフォーム	**ディレクトリー**
UNIX and Linux	`/var/mqm/exits` (32 ビット・チャネル出口) `/var/mqm/exits64` (64 ビット・チャネル出口)
Windows	*install_data_dir*¥exits *install_data_dir* は、インストール中に IBM WebSphere MQ データ・ファイル用に選択したディレクトリーです。 デフォルト・ディレクトリーは `C:\Program Files\IBM\WebSphere MQ` です。

注 : チャネル出口が複数の場所に存在する場合、IBM WebSphere MQ classes for JMS は最初に検出したインスタンスをロードします。

クラス・ローダーの親は、IBM WebSphere MQ classes for JMS のロードで使用されるクラス・ローダーです。 そのため、親クラス・ローダーは、チャネル出口が上記のいずれの場所でも見つからなかった場合にそれをロードすることができます。 ただし、 JEE アプリケーション・サーバーなどの環境で IBM WebSphere MQ classes for JMS を使用する場合は、親クラス・ローダーの選択に影響を与える可能性が低いため、アプリケーション・サーバーで Java システム・プロパティー **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** を設定してクラス・ローダーを構成する必要があります。

Java セキュリティー・マネージャーを有効にしてアプリケーションを実行している場合は、そのアプリケーションが実行されている Java ランタイム環境で使用されるポリシー構成ファイルには、チャネル出口クラスをロードする権限が必要です。 その方法について詳しくは、Java セキュリティー・マネージャーの下での IBM MQ classes for JMS アプリケーションの実行を参照してください。

Version 7.0 より前のバージョンの IBM WebSphere MQ で提供されている `MQSendExit`、`MQReceiveExit`、および `MQSecurityExit` インターフェースは引き続きサポートされます。 これらのインターフェースを実装するチャネル出口を使用する場合は、`com.ibm.mq.jar` がクラス・パスになければなりません。

C でチャネル出口を作成する方法については、398 ページの『メッセージング・チャネルのためのチャネル出口プログラム』を参照してください。 C または C++ で書かれたチャネル出口プログラムは、918 ページの表 130 に示されるディレクトリーに保管しなければなりません。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーに接続する場合は、919 ページの『IBM WebSphere MQ classes for JMS でのクライアント・チャネル定義テーブルの使用』を参照してください。

WebSphere MQ classes for JMS を使用する際のチャネル出口に渡すユーザー・データの指定

チャネル出口を呼び出すときには、32 文字までのユーザー・データを渡すことができます。

MQConnectionFactory オブジェクトの SENDEXITINIT プロパティーでは、各送信出口が呼び出されたときにその出口に渡されるユーザー・データを指定します。 このプロパティーの値は、コンマで区切ったユーザー・データの 1 つ以上の項目からなるストリングです。 ストリング内のユーザー・データの各項目の位置は、そのユーザー・データが一連の送信出口の内のどの送信出口に渡されるかを決定します。 例えば、ストリング内のユーザー・データの最初の項目は、一連の送信出口の内の最初の送信出口に渡されます。

SENDEXITINIT プロパティーは、WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使うことによって設定できます。 あるいは、アプリケーションから、setSendExitInit() メソッドを呼び出すことによって、プロパティーを設定できます。

同様に、ConnectionFactory オブジェクトの RECEXITINIT プロパティーでは各受信出口に渡されるユーザー・データを指定し、SECEXITINIT プロパティーではセキュリティー出口に渡されるユーザー・データを指定します。 これらのプロパティーは、WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使うことによって設定できます。 あるいは、アプリケーションから、setReceiveExitInit() メソッドと setSecurityExitInit() メソッドを呼び出すことによって、これらのプロパティーを設定できます。

チャネル出口に渡されるユーザー・データを指定する際には、以下の規則に注意してください。

- ストリング内のユーザー・データの項目数が一連の出口の数より多い場合、ユーザー・データの余分の項目は無視されます。
- ストリング内のユーザー・データの項目数が一連の出口の数より少ない場合、指定されていないユーザー・データの項目はそれぞれ空ストリングに設定されます。 ストリング内の 2 つの連続したコンマやストリングの始まりのコンマも、指定されていないユーザー・データの項目を表します。

アプリケーションがクライアント・チャネル定義テーブル (CCDT) を使用してキュー・マネージャーと接続する場合、チャネル出口が呼び出されるときに、クライアント接続チャネル定義で指定されたユーザー・データがチャネル出口に渡されます。 クライアント・チャネル定義テーブルの使用については、919 ページの『IBM WebSphere MQ classes for JMS でのクライアント・チャネル定義テーブルの使用』を参照してください。

IBM WebSphere MQ classes for JMS でのクライアント・チャネル定義テーブルの使用

IBM WebSphere MQ classes for JMS アプリケーションでは、クライアント・チャネル定義テーブル (CCDT) に格納されたクライアント接続チャネル定義を使用できます。 CCDT を使用するには、ConnectionFactory オブジェクトを構成します。 このコマンドの使用には、いくつかの制限があります。

ConnectionFactory オブジェクトの特定のプロパティーを設定してクライアント接続チャネル定義を作成する代わりに、 IBM WebSphere MQ classes for JMS アプリケーションは、クライアント・チャネル定義テーブルに保管されているクライアント接続チャネル定義を使用できます。 このような定義の作成には、IBM WebSphere MQ スクリプト・コマンド (MQSC) または IBM WebSphere MQ プログラマブル・コマンド・フォーマット (PCF) のコマンドを使用します。 アプリケーションが Connection オブジェクトを作成すると、 IBM WebSphere MQ classes for JMS は、クライアント・チャネル定義テーブルで適切なクライアント接続チャネル定義を検索し、そのチャネル定義を使用して MQI チャネルを開始します。 クライアント・チャネル定義テーブルの詳細とその構成方法については、クライアント・チャネル定義テーブルを参照してください。

クライアント・チャネル定義テーブルを使用するには、ConnectionFactory オブジェクトの CCDTURL プロパティーに URL オブジェクトを設定する必要があります。 URL オブジェクトは、クライアント・チャネル定義テーブルを格納するファイルの名前と場所を識別する URL (Uniform Resource Locator) をカプセル化し、そのファイルへのアクセス方法を指定します。 CCDTURL プロパティーは、IBM WebSphere MQ JMS

管理ツールを使用して設定できます。また、アプリケーションで、URL オブジェクトを作成してから ConnectionFactory オブジェクトの setCCDTURL() メソッドを呼び出すことによってこのプロパティーを設定できます。

例えば、ファイル ccdt1.tab にクライアント・チャネル定義テーブルが含まれており、アプリケーションが実行されるシステムと同じシステムにこのファイルが保管されている場合、アプリケーションで CCDTURL プロパティーを設定するには、以下のようにします。

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

もう 1 つの例として、ファイル ccdt2.tab にクライアント・チャネル定義テーブルが入っており、 アプリケーションの実行システムとは異なるシステムにこのファイルが格納されている場合を想定します。 FTP プロトコルを使用してこのファイルにアクセスできる場合、アプリケーションでは以下の方法で CCDTURL プロパティーを設定できます。

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

ConnectionFactory オブジェクトの CCDTURL プロパティーを設定する必要があるほかに、同じオブジェクトの QMANAGER プロパティーに以下のいずれかの値を設定する必要もあります。

- キュー・マネージャーの名前。
- アスタリスク (*) とそれに続くキュー・マネージャー・グループ名。
- アスタリスク (*)。
- 空ストリング、またはブランク文字のみを含むストリング。

これらの値は、メッセージ・キュー・インターフェース (MQI) を使用しているクライアント・アプリケーションによって発行された MQCONN 呼び出しで、*QMgrName* パラメーターに使用できる値と同じです。したがって、これらの値の意味の詳細については、MQCONN を参照してください。 QMANAGER プロパティーは、WebSphere MQ JMS 管理ツールまたは IBM WebSphere MQ エクスプローラーを使うことによって設定できます。 あるいは、アプリケーションから、ConnectionFactory オブジェクトの setQueueManager() メソッドを呼び出すことによって、プロパティーを設定できます。

その後、アプリケーションが ConnectionFactory オブジェクトから Connection オブジェクトを作成すると、 IBM WebSphere MQ classes for JMS は、CCDTURL プロパティーによって識別されるクライアント・チャネル定義テーブルにアクセスし、QMANAGER プロパティーを使用して適切なクライアント接続チャネル定義をテーブルで検索し、そのチャネル定義を使用してキュー・マネージャーへの MQI チャネルを開始します。

アプリケーションが createConnection() メソッドを呼び出すときに、ConnectionFactory オブジェクトの CCDTURL プロパティーと CHANNEL プロパティーの両方を設定することはできないことに注意してください。 両方のプロパティーが設定されると、このメソッドは例外をスローします。 CCDTURL プロパティーまたは CHANNEL プロパティーは、その値がヌル、空ストリング、ブランク文字のみのストリングのいずれでもない場合に設定されていると見なされます。

IBM WebSphere MQ classes for JMS は、クライアント・チャネル定義テーブルで適切なクライアント接続チャネル定義を検出すると、テーブルから抽出された情報のみを使用して MQI チャネルを開始します。ConnectionFactory オブジェクトのチャネル関連プロパティーはすべて無視されます。

特に、Secure Sockets Layer (SSL) を使用する場合は、以下の点に注意してください。

- MQI チャネルで SSL が使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義で、 IBM WebSphere MQ classes for JMS によってサポートされる CipherSpec の名前が指定されている場合のみです。
- クライアント・チャネル定義テーブルには、証明書取り消しリスト (CRL) を保持する LDAP (Lightweight Directory Access Protocol) サーバーの場所に関する情報も含まれます。 IBM WebSphere MQ classes for JMS は、CRL を保持する LDAP サーバーにアクセスするためにこの情報のみを使用します。
- クライアント・チャネル定義テーブルには、Online Certificate Status Protocol (OCSP) の応答側の場所を含めることもできます。 IBM WebSphere MQ classes for JMS は、クライアント・チャネル定義テーブ

ル・ファイル内の OCSP 情報を使用できません。 ただし、OCSP を構成することはできます (Online Certificate Protocol の使用セクションを参照)。

クライアント・チャネル定義テーブルでの SSL の使用の詳細については、SSL チャネルを持つ拡張トランザクション・クライアントの使用を参照してください。

チャネル出口を使用する場合は、以下の点にも注意してください。

- MQI チャネルで使用されるのは、クライアント・チャネル定義テーブルから抽出されたチャネル定義によって指定されているチャネル出口と関連ユーザー・データに限られます。
- クライアント・チャネル定義テーブルから抽出されたチャネル定義は、Java で作成されたチャネル出口を指定できます。 つまり、例えばクライアント接続チャネル定義を作成する場合、DEFINE CHANNEL コマンドの SCYEXIT パラメーターには、WMQSecurityExit インターフェースを実装するクラスの名前を指定できます。 同様に、SENDEXIT パラメーターには、WMQSendExit インターフェースを実装するクラスの名前を指定できます。また、RCVEXIT パラメーターには、WMQReceiveExit インターフェースを実装するクラスの名前を指定できます。 Java でチャネル出口を作成する方法について詳しくは、916 ページの『Java for WebSphere MQ classes for JMS でのチャネル出口の作成』を参照してください。

  Java 以外の言語で作成されたチャネル出口の使用もサポートされます。 別の言語で作成されたチャネル出口を DEFINE CHANNEL コマンドの SCYEXIT、SENDEXIT、および RCVEXIT パラメーターに指定する方法については、DEFINE CHANNEL を参照してください。

JMS クライアントの自動再接続

ネットワーク、キュー・マネージャー、またはサーバーで障害が起こった後に自動的に再接続が行われるよう、JMS クライアントを構成します。

`MQConnectionFactory` クラスの `CONNECTIONNAMELIST` プロパティーおよび `CLIENTRECONNECTOPTIONS` プロパティーを使用して、接続で障害が起こった後に再接続されるようクライアント接続を構成したり、キュー・マネージャーが停止した後にクライアント・アプリケーションに再接続するよう管理要求を構成したりすることができます。

connectionNameList の接続名の完全なリストには、接続名のリストを処理できる set/getconnectionNameList メソッドでのみアクセスできます。 名前のリストを処理しない get/setHostname などのメソッドは、リスト内の最初の名前にアクセスします。

自動的に再接続可能なクライアント接続は、接続が一度確立された場合にのみ再接続可能になります。

再接続が自動的に行われた後に、アプリケーションが引き続き正常に稼働できるかどうかは、再接続の設計にかかっています。 関連トピックを参照し、再接続可能なクライアントを設計する方法について理解してください。 既存のクライアントの中には、自動的に再接続された後の変更を行わなくても正常に稼働するものがあります。

クライアントの自動再接続機能は、WebSphere MQ classes for Java ではサポートされていません。

障害の起きたキュー・マネージャーに接続しているすべてのクライアントが同時に再接続するという状況を回避するため、一部は固定、一部はランダムに決まる遅延間隔を空けて再接続が試行されます。

デフォルトでは、再接続は以下の間隔で試行されます。

1. 最初の試行は、1 秒に 250 ミリ秒までのランダム要素を加算した初期遅延の後、実行されます。
2. 2 回目の試行は、最初の試行が失敗した後、2 秒に 500 ミリ秒までのランダム間隔を加算した遅延を空けて、実行されます。
3. 3 回目の試行は、2 回目の試行が失敗した後、4 秒に 1 秒までのランダム間隔を加算した遅延を空けて、実行されます。
4. 4 回目の試行は、3 回目の試行が失敗した後、8 秒に 2 秒までのランダム間隔を加算した遅延を空けて、実行されます。
5. 5 回目の試行は、4 回目の試行が失敗した後、16 秒に 4 秒までのランダム間隔を加算した遅延を空けて、実行されます。
6. 6 回目の試行、およびそれ以後のすべての試行は、前回の試行が失敗した後、25 秒に 6.25 秒までのランダム間隔を加算した遅延を空けて、実行されます。

この再接続のプロセスが、クライアントが正常にキュー・マネージャーに再接続されるまで、または再接続の最大間隔が経過するまで、続行されます。

キュー・マネージャーの復旧に必要な時間、またはスタンバイ・キュー・マネージャーがアクティブ化されるまでの時間をより正確に反映させるために、デフォルト値を大きくする必要がある場合は、MQCLIENT.INI ファイルの **ReconDelay** 属性を使用して遅延の値を変更します。

関連概念

クライアントの自動再接続

関連タスク

構成ファイルを使用したクライアントの構成

IBM WebSphere MQ classes for JMS での TCP/IP 接続の共有

MQI チャネルの複数インスタンスが、単一の TCP/IP 接続を共用するようにできます。

同じ Java ランタイム環境内で実行され、 IBM WebSphere MQ classes for JMS または IBM WebSphere MQ リソース・アダプターを使用して CLIENT トランスポートを使用してキュー・マネージャーに接続するアプリケーションは、同じチャネル・インスタンスを共有するようにすることができます。

チャネル・インスタンスと TCP/IP 接続間には 1 対 1 の関係があります。 チャネル・インスタンスごとに 1 つの TCP/IP 接続が作成されます。

SHARECNV パラメーターが 1 よりも大きい値に設定されてチャネルが定義されている場合、その数の会話で 1 つのチャネル・インスタンスを共用できます。 この機能を使用するために接続ファクトリーまたはアクティベーション・スペックを有効にするには、**SHARECONVALLOWED** プロパティーを YES に設定します。

JMS アプリケーションで作成されたすべての JMS 接続および JMS セッションは、キュー・マネージャーとの独自の会話を作成します。

アクティベーション・スペックが開始されると、IBM WebSphere MQ classes for JMS リソース・アダプターは、アクティベーション・スペックで使用するキュー・マネージャーとの会話を開始します。 アクティベーション・スペックに関連付けられているサーバー・セッション・プール内の各サーバー・セッションでも、キュー・マネージャーとの会話が開始されます。

SHARECNV 属性は、接続共用へのベスト・エフォート・アプローチです。 したがって、 IBM WebSphere MQ classes for JMS で 0 より大きい SHARECNV 値を使用した場合、新しい接続要求は、既に確立されている接続を常に共用するという保証はありません。

チャネル・インスタンスの数の計算

アプリケーションで作成されるチャネル・インスタンスの最大数を決定するには、以下の公式を使用します。

アクティベーション・スペック

チャネル・インスタンスの数 = (<**maxPoolDepth**> + 1)/<**SHARECNV**>

ここで、<**maxPoolDepth**> は **maxPoolDepth** プロパティーの値、<**SHARECNV**> はアクティベーション・スペックによって使用されるチャネルの **SHARECNV** プロパティーの値です。

他の JMS アプリケーション

チャネル・インスタンスの数 = (< JMS 接続> + < JMS セッション>)/<**SHARECNV**>

ここで、< JMS 接続> は、アプリケーションによって作成された接続の数です。ここで、< JMS セッション> は、アプリケーションによって作成された JMS セッションの数であり、<**SHARECNV**> は、アクティベーション・スペックによって使用されるチャネルの **SHARECNV** プロパティーの値です。

例

以下の例は、アプリケーションが IBM WebSphere MQ classes for JMS または IBM WebSphere MQ classes for JMS リソース・アダプターを使用して、キュー・マネージャーに作成するチャネル・インスタンスの数を計算するために公式を使用する方法を示しています。

JMS アプリケーションの例

JMS アプリケーション接続では、CLIENT トランスポートを使用してキュー・マネージャーに接続し、1 つの JMS 接続と 3 つの JMS セッションを作成します。 アプリケーションがキュー・マネージャーへの接続に使用するチャネルの **SHARECNV** プロパティーは 10 の値に設定されています。 アプリケーションの実行時には、アプリケーションとキュー・マネージャーの間に存在する会話は 4 つあり、チャネル・インスタンスは 1 つあります。 4 つの会話はすべてチャネル・インスタンスを共用します。

アクティベーション・スペックの例

アクティベーション・スペックでは、CLIENT トランスポートを使用してキュー・マネージャーに接続します。 アクティベーション・スペックは **maxPoolDepth** プロパティーを 10 に設定して構成されています。 アクティベーション・スペックが使用するように構成されているチャネルの **SHARECNV** プロパティーは 10 に設定されています。 アクティベーション・スペックが実行され、同時に 10 件のメッセージが処理されている場合、アクティベーション・スペックとキュー・マネージャーの間の会話の数は 11 (10 個の会話はサーバー・セッション用で、1 個はアクティベーション・スペック用です) となります。 アクティベーション・スペックで使用されるチャネル・インスタンスの数は 2 です。

アクティベーション・スペックの例

アクティベーション・スペックでは、CLIENT トランスポートを使用してキュー・マネージャーに接続します。 アクティベーション・スペックは、**maxPoolDepth** プロパティーを 5 に設定して構成されます。 アクティベーション・スペックが使用されるように構成されているチャネルでは、**SHARECNV** プロパティーが 0 に設定されています。 アクティベーション・スペックが実行されていて、5 つのメッセージを同時に処理している場合、アクティベーション・スペックとキュー・マネージャーの間の会話の数は 6 (サーバー・セッションの場合は 5、アクティベーション・スペックの場合は 1) になります。 アクティベーション・スペックで使用されるチャネル・インスタンスの数は 6 です。チャネルの **SHARECNV** プロパティーは 0 に設定されているため、各会話では独自のチャネル・インスタンスが使用されます。

WebSphere MQ classes for JMS でのクライアント接続を受け入れるためのポート範囲の指定

LOCALADDRESS プロパティーを使用して、アプリケーションがバインドできるポート範囲を指定します。

WebSphere MQ classes for JMS アプリケーションがクライアント・モードで WebSphere MQ キュー・マネージャーに接続しようとした場合、ファイアウォールは指定されたポートまたはポートの範囲から発生する接続のみを許可します。 この場合、ConnectionFactory、QueueConnectionFactory または TopicConnectionFactory オブジェクトの LOCALADDRESS プロパティーを使用して、アプリケーションがバインドできるポート、またはポートの範囲を指定することができます。

LOCALADDRESS プロパティーは、WebSphere MQ JMS 管理ツールを使うか、または JMS アプリケーションで setLocalAddress() メソッドを呼び出すことによって設定できます。 アプリケーション内からプロパティーを設定する例を以下に示します。

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

アプリケーションがキュー・マネージャーに接続すると、アプリケーションはローカル IP アドレスおよび 192.0.2.0(2000) から 192.0.2.0(3000) の範囲にあるポート番号にバインドされます。

複数のネットワーク・インターフェースが存在するシステムでは、LOCALADDRESS プロパティーを使用して、1 つの接続に対してどのネットワーク・インターフェースを使用すべきかを指定することもできます。

ブローカーにリアルタイムで接続する場合、マルチキャストが使用される場合にのみ、LOCALADDRESS プロパティーが使用されます。 この場合、このプロパティーを使用して、1 つの接続に対してどのローカル・ネットワーク・インターフェースを使用すべきかを指定できますが、プロパティーの値にポート番号、またはポート番号の範囲を指定することはできません。

ポートの範囲を制限すると、接続エラーが起きる可能性があります。 エラーが起きた場合、WebSphere MQ 理由コード MQRC_Q_MGR_NOT_AVAILABLE および以下のメッセージを含む、MQException が組み込まれた JMSException がスローされます。

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

指定された範囲内のすべてのポートが使用中である場合、または指定された IP アドレス、ホスト名、ポート番号が正しくない場合 (負のポート番号など) は、エラーが発生する可能性があります。

WebSphere MQ classes for JMS はアプリケーションが必要とする接続以外の接続を作成する可能性があるため、常にポートの範囲を指定するようにしてください。 一般的に、アプリケーションが作成する各セッションごとに 1 つのポートが必要で、WebSphere MQ classes for JMS はさらに 3 つまたは 4 つの追加ポートが必要になる場合があります。 接続エラーが起きた場合は、ポートの範囲を増やしてください。

WebSphere MQ classes for JMS でデフォルトで使用される接続プーリングは、ポートの再使用が可能になる速度に影響を与える場合があります。 このため、ポートの解放中に接続エラーが起きる可能性があります。

WebSphere MQ classes for JMS でのチャネル圧縮

WebSphere MQ classes for JMS アプリケーションは、WebSphere MQ 機能を使用してメッセージ・ヘッダーまたはデータを圧縮できます。

WebSphere MQ チャネルを流れるデータを圧縮すると、チャネルのパフォーマンスを改善し、ネットワーク・トラフィックを削減することができます。 WebSphere MQ で提供される機能を使用して、メッセージ・チャネルと MQI チャネルを流れるデータを圧縮できます。 また、どちらのタイプのチャネルでも、ヘッダー・データとメッセージ・データを個別に圧縮できます。 デフォルトでは、チャネル上のデータは圧縮されません。

WebSphere MQ classes for JMS アプリケーションでは、java.util.Collection オブジェクトを作成することによって、接続上のヘッダー・データまたはメッセージ・データの圧縮に使用可能な手法を指定します。 各圧縮手法は、このコレクション内の Integer オブジェクトです。アプリケーションが圧縮手法をコレクションに追加する順序は、アプリケーションの接続作成時に圧縮手法がキュー・マネージャーにネゴシエーションされた順序になります。 その後、アプリケーションで、このコレクションを ConnectionFactory オブジェクトに渡します。このとき、対象データがヘッダー・データであれば setHdrCompList() メソッドを呼び出し、メッセージ・データであれば setMsgCompList() メソッドを呼び出します。 アプリケーション側の準備が完了すれば、接続を作成できます。

以下のコード・フラグメントは、ここで説明した方法の例です。 最初のコード・フラグメントは、ヘッダー・データ圧縮の実装方法を示します。

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

2 番目のコード・フラグメントは、メッセージ・データ圧縮の実装方法を示します。

```
Collection msgComp = new Vector();
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_RLE));
msgComp.add(new Integer(WMQConstants.WMQ_COMPMSG_ZLIBHIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

2 番目の例では、圧縮技法は、接続の作成時に RLE、次に ZLIBHIGH の順にネゴシエーションされます。 選択された圧縮手法は、Connection オブジェクトが存続している間は変更できません。 接続上で圧縮を使用するには、Connection オブジェクトを作成する前に、setHdrCompList() メソッドと setMsgCompList() メソッドを呼び出す必要があります。

IBM WebSphere MQ classes for JMS でのメッセージの非同期書き込み

通常、アプリケーションが宛先にメッセージを送信すると、アプリケーションはキュー・マネージャーが要求を処理したことを確認するまで待機しなければなりません。 ある環境では、そうする代わりにメッセージを非同期的に書き込ませることで、メッセージングのパフォーマンスを向上させることができます。

アプリケーションにメッセージを非同期的に書き込ませる場合、キュー・マネージャーは、呼び出しのたびに成功や失敗を返しません。その代わりに、周期的にエラーの確認を行うことができます。

キュー・マネージャーがメッセージを支障なく受信したかどうかを判別することなく、宛先がアプリケーションに制御を 戻すかどうかは、以下のプロパティーによって異なります。

- JMS 宛先プロパティー PUTSYNCALLOWED (短縮名 - PAALD)。

  PUTASYNCALLOWED は、JMS アプリケーションが非同期でメッセージを書き込むことができるかどうかを制御します (JMS 宛先が表している基礎となるキューまたはトピックでこのオプションが許可されている場合)。
- IBM WebSphere MQ キュー またはトピック・プロパティー DEFPRESP (デフォルトの書き込み応答タイプ)。

  DEFPRESP では、キューにメッセージを書き込むアプリケーションまたはトピックにメッセージをパブリッシュするアプリケーションで、非同期書き込み機能を使用できるかどうかを指定します。

以下の表に、PUTASYNCALLOWED プロパティーと DEFPRESP プロパティーに指定できる値、 および非同期書き込み機能を有効にするのに必要となる値を示します。

表 *131.* メッセージを非同期で書き込むかどうかを判別する *PUTASYNCALLOWED* プロパティーと *DEFPRESP* プロパティー

WebSphere MQ キュー・プロパティー	**PUTASYNCALLOWED = NO**	**PUTASYNCALLOWED = YES**	**PUTASYNCALLOWED = AS_DEST または AS_Q_DEF または AS_T_DEF**
DEFPRESP=SYNC	非同期書き込み機能が無効	非同期書き込み機能が有効	非同期書き込み機能が無効
DEFPRESP=ASYNC	非同期書き込み機能が無効	非同期書き込み機能が有効	非同期書き込み機能が有効

トランザクション化されたセッションで送信されるメッセージの場合、アプリケーションは最終的に、`commit()` を呼び出すときにキュー・マネージャーがメッセージを支障なく受け取ったかどうかを判別します。

アプリケーションがトランザクション化されたセッションの中で持続メッセージを送信し、1 つ以上のメッセージの受信に支障があった場合、トランザクションはコミットに失敗し、例外が生成されます。 しかし、アプリケーションがトランザクション化されたセッションの中で非持続メッセージを送信し、1 つ以上のメッセージの受信に支障があった場合は、トランザクションは正常にコミットされます。 このアプリケーションは、非持続メッセージが無事に到着しなかったというフィードバックを受信しません。

トランザクション化されていないセッションで送信された非持続メッセージの場合、*ConnectionFactory* オブジェクトの SENDCHECKCOUNT プロパティーで、キュー・マネージャーがメッセージを支障なく受け取ったことを IBM WebSphere MQ classes for JMS が検査するまでにいくつのメッセージを送信するかを指定します。

検査により 1 つ以上のメッセージの受信に支障があったことが分かり、アプリケーションが例外リスナーをその接続で登録している場合、IBM WebSphere MQ classes for JMS は例外リスナーの `onException()` メソッドを呼び出し、JMS 例外をアプリケーションに渡します。

JMS 例外はエラー・コード JMSWMQ0028 を持ち、このコードにより以下のメッセージが表示されます。

```
At least one asynchronous put message failed or gave a warning.
```

さらに JMS 例外には、詳細を提供するリンクされた例外があります。 SENDCHECKCOUNT プロパティーのデフォルト値はゼロで、これはそのような検査が行われないことを意味します。

この最適化は、クライアント・モードでキュー・マネージャーに接続するアプリケーションに最も利点があり、一連のメッセージを連続して迅速に送信する必要がありますが、送信された各メッセージについてキュー・マネージャーから即時のフィードバックは必要としません。 ただし、バインディング・モードでキュー・マネージャーに接続する場合でも、アプリケーションは引き続きこの最適化を使用することができますが、予期されるパフォーマンスの利点はそれほど大きくありません。

WebSphere MQ classes for JMS での先読みの使用

WebSphere MQ で提供される先読み機能を使用することで、トランザクション外で受信された非永続メッセージを、アプリケーションから要求される前に IBM WebSphere MQ classes for JMS に送信することができます。 IBM WebSphere MQ classes for JMS は内部バッファーにメッセージを保管し、アプリケーションから要求された場合はそのメッセージをアプリケーションに渡します。

`MessageConsumers` または `MessageListeners` を使用してトランザクション外の宛先からメッセージを受信する IBM WebSphere MQ classes for JMS アプリケーションは、先読み機能を使用できます。 先読みを使用することで、オブジェクトを使用するアプリケーションは、メッセージ受信時のパフォーマンスを改善できます。

`MessageConsumers` または `MessageListeners` を使用するアプリケーションが先読みを使用できるかどうかは、以下のプロパティーによって決まります。

- JMS 宛先プロパティー READAHEADALLOWED (短縮名 - RAALD)。 READAHEADALLOWED は、JMS 宛先が表している基礎となるキューまたはトピックでこのオプションが許可されている場合に、JMS アプリケーションがトランザクション外の非永続メッセージの取得時または参照時に先読みを使用できるかどうかを制御します。
- IBM WebSphere MQ キューまたはトピック・プロパティー DEFREADA (デフォルトの先読み)。 DEFREADA は、トランザクション外の非永続メッセージを受信または参照しているアプリケーションが先読みを使用できるかどうかを指定します。

以下の表に、READAHEADALLOWED プロパティーおよび DEFREADA プロパティーに指定できる値と、先読み機能を有効にするために必要な値を示します。

表 *132.* トランザクション外の非永続メッセージを受信または参照する際に先読みを使用するかどうかを決定する *READAHEADALLOWED* プロパティーと *DEFREADA* プロパティー

WebSphere MQ 宛先プロパティー	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NO**	**AS_DEST または AS_Q_DEF または AS_T_DEF**
WebSphere MQ キュー・プロパティー			
DEFREADA = NO	先読み機能が有効	先読み機能が無効	先読み機能が無効
DEFREADA = YES	先読み機能が有効	先読み機能が無効	先読み機能が有効
DEFREADA = DISABLED	先読み機能が無効	先読み機能が無効	先読み機能が無効

先読み機能が有効な場合に、アプリケーションで `MessageConsumer` または `MessageListener` が作成されると、IBM WebSphere MQ classes for JMS は、`MessageConsumer` または `MessageListener` がモニターする宛先の内部バッファーを作成します。 `MessageConsumer` または `MessageListener` にはそれぞれ 1 つの内部バッファーがあります。 アプリケーションが以下のいずれかのメソッドを呼び出すと、キュー・マネージャーは IBM WebSphere MQ classes for JMS への非永続メッセージの送信を開始します。

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

IBM WebSphere MQ classes for JMS は、アプリケーションによるメソッド呼び出しで、最初のメッセージをアプリケーションに自動的に返します。 その他の非永続メッセージは、IBM WebSphere MQ classes for JMS によって、宛先用に作成された内部バッファーに保管されます。 アプリケーションが次のメッセージの処理を要求すると、IBM WebSphere MQ classes for JMS は内部バッファーの次のメッセージを返します。

内部バッファーが空の場合、IBM WebSphere MQ classes for JMS はキュー・マネージャーからの非永続メッセージをさらに要求します。

IBM WebSphere MQ classes for JMS によって使用される内部バッファーは、アプリケーションが MessageConsumer を閉じるとき、または MessageListener が関連付けられている JMS セッションを閉じるときに削除されます。

MessageConsumers の場合、内部バッファー内の未処理メッセージはすべて失われます。

MessageListeners を使用する場合、内部バッファー内のメッセージに何が起こるかは、JMS 宛先プロパティー READAHEADCLOSEPOLICY (短縮名-RACP) によって異なります。 プロパティーのデフォルト値は DELIVER_ALL です。これは、MessageListener の作成時に使用された JMS セッションが、内部バッファー内のすべてのメッセージがアプリケーションに配信されるまでクローズされないことを意味します。 プロパティーを DELIVER_CURRENT に設定すると、現行メッセージがアプリケーションで処理され、内部バッファー内の残りのメッセージがすべて破棄されてから、JMS セッションがクローズされます。

WebSphere MQ classes for JMS での保存パブリケーション

WebSphere MQ classes for JMS クライアントは、保存パブリケーションを使用するように構成することができます。

パブリッシャーは、将来このトピックに関心を持つサブスクライバーが現れた際にパブリケーションを送信できるよう、パブリケーションのコピーを保存しておくことを指定できます。 これは、 WebSphere MQ classes for JMS で、整数プロパティー JMS_IBM_RETAIN を値 1 に設定することによって行います。 これらの値に対して、com.ibm.msg.client.jms.JmsConstants インターフェースで定数が定義されています。 例えば、*msg* というメッセージを作成した際に、これを保存パブリケーションとして設定するには、以下のコードを使用します。

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

これで、このメッセージを通常のものとして送信できるようになります。 JMS_IBM_RETAIN を受信メッセージ内で照会することも可能です。 つまり、受信メッセージが保存パブリケーションであるかどうかを照会できるわけです。

WebSphere MQ classes for JMS での XA のサポート

JMS は、サポートしているトランザクション・マネージャーで、バインディング・モードとクライアント・モードの XA 準拠のトランザクションをサポートしています。

アプリケーション・サーバー環境で XA 機能が必要な場合は、アプリケーションを適切に構成する必要があります。 分散トランザクションを使用するようにアプリケーションを構成する方法については、アプリケーション・サーバー独自の資料を参照してください。

WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続の使用

WebSphere MQ classes for JMS アプリケーションは、パブリッシュ/サブスクライブ・メッセージング用に、WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続を使用できます。 ブローカーおよび WebSphere MQ classes for JMS の両方は、リアルタイム接続を使用可能にするように構成する必要があります。

アプリケーションが WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続を使用する場合、アプリケーションとブローカーは、WebSphere MQ Real-Time Transport を使用してメッセージを交換します。 構成に応じて、メッセージを WebSphere MQ Multicast Transport を使用してアプリケーションに送信することもできます。

アプリケーションがどのように WebSphere MQ キュー・マネージャーに接続し、WebSphere MQ Enterprise Transport を使用して WebSphere Event Broker または WebSphere Message Broker のブローカーとメッセージを交換するかについては、前のリリースの WebSphere MQ classes for JMS の資料を参照してください。 WebSphere MQ Enterprise Transport を使用するには、アプリケーションは WebSphere MQ メッセージング・プロバイダーのマイグレーション・モードで実行する接続ファクトリーを使用して、キュー・マネージャーに接続する必要があることに注意してください。

WebSphere Event Broker または *WebSphere Message Broker* のブローカーのリアルタイム接続用の構成

WebSphere MQ classes for JMS application で WebSphere Event Broker または WebSphere Message Broker のブローカーのリアルタイム接続を使用するには、ブローカーがメッセージを listen およびパブリッシュする TCP/IP ポートからメッセージを読み取るようにメッセージ・フローを作成およびデプロイすることで、ブローカーを構成する必要があります。 要件に応じて、追加的な方法でブローカーを構成することが必要になる場合があります。

ブローカーを構成するには、以下のメッセージ・フローの 1 つを作成およびデプロイする必要があります。

- Real-timeOptimizedFlow メッセージ処理ノードを含むメッセージ・フロー
- Real-timeInput メッセージ処理ノードおよび Publication メッセージ処理ノードを含むメッセージ・フロー

Real-timeOptimizedFlow または Real-timeInput ノードを、リアルタイム接続に使用する TCP/IP ポート上で listen するように構成する必要があります。 デフォルトでは、リアルタイム接続のポート番号は 1506 です。

以下の要件のいずれかがある場合も、ブローカーを構成する必要があります。

- アプリケーションを Secure Sockets Layer (SSL) 認証を使用してブローカーに接続させる場合
- アプリケーションを HTTP トンネルを使用してブローカーに接続させる場合
- メッセージをマルチキャストを使用してメッセージ・コンシューマーに送達したい場合

ブローカーの構成方法については、*WebSphere Event Broker* の製品資料 や *WebSphere Message Broker* の製品資料 を参照してください。

WebSphere Event Broker または *WebSphere Message Broker* のブローカーへのリアルタイム接続用の *WebSphere MQ classes for JMS* の構成

WebSphere MQ classes for JMS アプリケーションに WebSphere Event Broker または WebSphere Message Broker のブローカーへのリアルタイム接続を使用させるには、WebSphere MQ classes for JMS を、接続ファクトリーの特定のプロパティーを設定することで構成する必要があります。 要件に応じて、WebSphere MQ classes for JMS は追加的な方法で構成することが必要な場合があります。

WebSphere MQ classes for JMS を構成するには、接続ファクトリーの以下のプロパティーを設定する必要があります。

- TRANSPORT プロパティーは DIRECT に設定する必要があります。

  ただし、アプリケーションが HTTP トンネルを使用して接続する場合は、TRANSPORT プロパティーを代わりに DIRECTHTTP に設定する必要があります。 929 ページの『HTTP トンネルの使用』 を参照してください。
- HOSTNAME プロパティーは、ブローカーが稼働しているシステムのホスト名または IP アドレスに設定する必要があります。
- PORT プロパティーは、ブローカーがリアルタイム接続を listen するポートの番号に設定する必要があります。

アプリケーションは、IBM JMS 拡張機能または WebSphere MQ JMS 拡張機能を使用して、それらのプロパティーを実行時に動的に設定することができます。 別の方法として、接続ファクトリーが管理対象オブジェクトである場合、管理者はそれらのプロパティーを WebSphere MQ JMS 管理ツールまたは WebSphere MQ エクスプローラーを使用して設定できます。

プロパティー、およびこれらの値を設定するためにアプリケーションが使用するメソッドについては、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーを参照してください。 WebSphere MQ JMS 管理ツールの使い方については、938 ページの『WebSphere MQ JMS 管理ツールの使用』 を参照してください。 WebSphere MQ エクスプローラーの使い方については、WebSphere MQ エクスプローラーに付属するヘルプを参照してください。

以下の要件のいずれかがある場合、WebSphere MQ classes for JMS は追加の構成が必要です。

- アプリケーションを Secure Sockets Layer (SSL) 認証を使用してブローカーに接続させる場合

- アプリケーションを HTTP トンネルを使用してブローカーに接続させる場合
- アプリケーションをプロキシー・サーバーからブローカーに接続させる場合
- メッセージをマルチキャストを使用してメッセージ・コンシューマーに送達したい場合

以下のセクションでは、それぞれの要件に応じて WebSphere MQ classes for JMS を構成する方法を説明しています。

Secure Sockets Layer (SSL) 認証の使用

SSL 認証は、ブローカーへのリアルタイム接続で使用できます。 このタイプの接続では、認証のみがサポートされます。 アプリケーションとブローカーとの間を流れるメッセージ・データの暗号化と復号、およびデータの改ざんの検出には、SSL は使用できません。

この状態と、アプリケーションがクライアント・モードでキュー・マネージャーに接続する場合の違いに注意してください。 後者の場合、WebSphere MQ SSL サポートを使用して、アプリケーションとキュー・マネージャーとの間を流れるメッセージ・データの暗号化または復号、およびデータの改ざんの検出を行い、認証を提供することができます。

ブローカーに対するリアルタイム接続でメッセージ・データを保護したい場合、代わりにブローカーが提供する機能を使用することができます。 保護品質 (QoP) の値を、メッセージを保護したい各トピックに割り当てることができます。 これにより、トピック毎に違うレベルのメッセージ保護を選択することができます。 ブローカーにより提供されるメッセージ保護の詳細については、*WebSphere Event Broker* の製品資料 や *WebSphere Message Broker* の製品資料 を参照してください。

ブローカーへのリアルタイム接続で SSL 認証を使用するには、接続ファクトリーの DIRECTAUTH プロパティーを CERTIFICATE に設定する必要があります。

相互認証に SSL を使用する場合、ブローカーの「認証プロトコル・タイプ」プロパティーでは、対称 SSL のオプション R を指定する必要があります。 ブローカーの認証だけに SSL を使用する場合、ブローカーの「認証プロトコル・タイプ」プロパティーでは、非対称 SSL のオプション S を指定する必要があります。 しかしこの場合、アプリケーションは、以下の例で示すとおり、ユーザー ID とパスワードをパラメーターとして使用して、createConnection() を呼び出すことでブローカーに接続する必要があります。

```
factory.createConnection("user1", "user1pw");
```

次いでブローカーは、SSL の代わりにユーザー ID とパスワードとを使用して、アプリケーションを認証します。 SSL 認証用にブローカーを構成する方法の詳細については、*WebSphere Event Broker* の製品資料 や *WebSphere Message Broker* の製品資料 を参照してください。

注 :

1. DIRECTAUTH プロパティーの値が、SSL 認証がブローカーへのリアルタイム接続で使用されるかどうかを決定します。SSLCIPHERSUITE プロパティーの値ではありません。
2. SSL 認証がブローカーへのリアルタイム接続で使用される場合、SSLPEERNAME および SSLCRL プロパティーは、アプリケーションがキュー・マネージャーにクライアント・モードで接続するときに実行されるのと同じ検査を実行するために使用されます。
3. WebSphere MQ classes for JMS は、同じ Java Secure Socket Extension (JSSE) 鍵ストアおよびトラストストア構成を使用して、以下のいずれかの状況で SSL サポートを提供できます。
   - アプリケーションがブローカーへのリアルタイム接続を使用する場合
   - アプリケーションがキュー・マネージャーにクライアント・モードで接続する場合

HTTP トンネルの使用

WebSphere MQ classes for JMS アプリケーションは、HTTP トンネルを使用してブローカーに接続できます。これはつまり、Web サイトに接続しているかのように、アプリケーションが HTTP プロトコルを使用してブローカーに接続するという意味です。

ブローカーへのリアルタイム接続で HTTP トンネルを使用するには、接続ファクトリーの TRANSPORT プロパティーを DIRECTHTTP に設定する必要があります。

HTTP トンネルは SSL 認証と共に使用したり、プロキシー・サーバーを介して接続したり、マルチキャストを使用してメッセージを送達したりすることはできません。 サポートされている HTTP プロトコルのバージョンは、1.0 です。 HTTP バージョン 1.1 はサポートされていません。

プロキシー・サーバーを介した接続

WebSphere MQ classes for JMS アプリケーションは、プロキシー・サーバーを介して接続することにより、ブローカーへのリアルタイム接続を使用できます。 WebSphere MQ classes for JMS は、プロキシー・サーバーに直接接続し、RFC 2817 で定義されているインターネット・プロトコルを使用して、プロキシー・サーバーに接続要求をブローカーに転送するように要求します。

プロキシー・サーバーを介してブローカーに接続するには、接続ファクトリーの以下のプロパティーを設定する必要があります。

- PROXYHOSTNAME プロパティーは、プロキシー・サーバーが稼働しているシステムのホスト名または IP アドレスに設定する必要があります。
- PROXYPORT プロパティーは、プロキシー・サーバーが listen するポートの番号に設定する必要があります。

PROXYHOSTNAME プロパティーが設定されていない場合、または空ストリングに設定されている場合、WebSphere MQ classes for JMS は、HOSTNAME および PORT プロパティーだけを使用してブローカーに直接接続しようとし、プロキシー・サーバーを介しての接続は試行しません。

マルチキャストを使用したメッセージの送達

ブローカーへのリアルタイム接続を使用していれば、マルチキャストを使用してメッセージ・コンシューマーにメッセージを送達できます。

マルチキャストを使用可能にするには、Topic オブジェクトの MULTICAST プロパティーを、必要なマルチキャスト・オプションに設定する必要があります。 別の方法として、Topic オブジェクトの MULTICAST プロパティーが ASCF に設定されている場合、接続ファクトリーの MULTICAST プロパティーは、必要なマルチキャスト・オプションに設定する必要があります。

WebSphere MQ classes for JMS は、Packet Transfer Layer (PTL) および Pragmatic General Multicast (PGM) マルチキャスト・プロトコルの両方をサポートしており、どちらの PGM プロトコル (カプセル化された PGM/IP および PGM UDP) の実装のサポートも組み込まれています。 ただし、PGM/IP サポートは以下のプラットフォームでのみ使用可能です。

- AIX (32 ビットのみ)
- Linux (x86 プラットフォーム)
- Linux (zSeries プラットフォーム、32 ビットのみ)
- Solaris SPARC (32 ビットのみ)
- Windows (32 ビットのみ)
- z/OS

WebSphere MQ classes for JMS アプリケーション・サーバー機構

このトピックでは、WebSphere MQ classes for JMS が Session クラス内の ConnectionConsumer クラスおよび拡張機能を実装する方法を説明します。 さらに、サーバー・セッション・プールの機能も要約しています。

WebSphere MQ classes for JMS は、 *Java Message Service Specification*、バージョン *1.1* で指定されている Application Server Facilities (ASF) をサポートします (Sun の Java Web サイト (https://java.sun.com) を参照してください)。 この仕様では、このプログラミング・モデル内で以下の 3 つの役割を定めています。

- **JMS プロバイダー**は、ConnectionConsumer および拡張セッション機能を提供します。
- **アプリケーション・サーバー**は、ServerSessionPool および ServerSession 機能を提供します。

- **クライアント・アプリケーション**は、JMS プロバイダーおよびアプリケーション・サーバーが提供する機能を使用します。

このトピックの情報は、アプリケーションがブローカーとのリアルタイム接続を使用する場合には適用されません。

JMS ConnectionConsumer

ConnectionConsumer インターフェースは、スレッドのプールに同時にメッセージを送達するための高性能のメソッドを提供します。

JMS 仕様では、アプリケーション・サーバーは、`ConnectionConsumer` インターフェースを使用して、JMS 実装と密接に統合できます。 この機能は、メッセージの並行処理を提供します。 通常、アプリケーション・サーバーがスレッドのプールを作成し、JMS 実装がこれらのスレッドに使用可能なメッセージを作成します。 JMS 対応のアプリケーション・サーバー (WebSphere Application Server など) は、この機能を使用して、メッセージ駆動型 Bean など高水準のメッセージング機能を提供することができます。

通常のアプリケーションは ConnectionConsumer を使用しないが、エキスパート JMS クライアントは ConnectionConsumer を使用することがあります。 そのようなクライアントに対して、ConnectionConsumer は、スレッドのプールに同時にメッセージを送達するための高性能のメソッドを提供します。 メッセージがキューまたはトピックに到達した際、JMS はプールからスレッドを選択し、そのスレッドにメッセージのバッチを送達します。 これを行うために、JMS は、関連する MessageListener の `onMessage()` メソッドを実行します。

複数の Session および MessageConsumer オブジェクト (それぞれ登録された MessageListener を持つ) を構成することによって、同じ効果が得られます。 ただし、ConnectionConsumer の方が、パフォーマンスが良く、リソースの使用量が少なく、より大きな柔軟性があります。 特に、必要となる Session オブジェクトが少数で済みます。

ASF によるアプリケーションの計画

このセクションでは、以下のようなアプリケーション計画の方法について説明します。

- 931 ページの『ASF を使用した Point-to-Point メッセージングの一般原則』
- 932 ページの『ASF を使用したパブリッシュ/サブスクライブ・メッセージングの一般原則』
- 933 ページの『ASF でのキューからのメッセージの除去』
- ASF での有害メッセージの処理。 893 ページの『IBM WebSphere MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

ASF を使用した *Point-to-Point* メッセージングの一般原則

このトピックでは、ASF を使用した Point-to-Point メッセージングの一般情報について説明します。

アプリケーションは、QueueConnection オブジェクトから ConnectionConsumer を作成する際、JMS キュー・オブジェクトおよびセレクター・ストリングを指定します。 その後、ConnectionConsumer は、関連した ServerSessionPool 内のセッションにメッセージを供給し始めます。 メッセージがキューに到着し、セレクターと一致する場合、メッセージは、関連付けられている ServerSessionPool 内のセッションに送達されます。

WebSphere MQ 用語では、キュー・オブジェクトは、ローカル・キュー・マネージャー上にある QLOCAL または QALIAS のいずれかを表します。 QALIAS である場合には、その QALIAS は QLOCAL を参照しなければなりません。 完全に解決された WebSphere MQ QLOCAL を、基礎 *QLOCAL* と言います。 ConnectionConsumer がクローズされておらず、その親 QueueConnection が開始されている場合には、ConnectionConsumer はアクティブ であると言います。

複数の ConnectionConsumer (それぞれ異なるセレクターを持つ) を、同じ基礎 QLOCAL に対して実行することが可能です。 パフォーマンスを保つために、不必要なメッセージをキュー上に累積させないようにしてください。 不必要なメッセージとは、アクティブな ConnectionConsumer が一致するセレクターを持たないメッセージを指します。 これらの不必要なメッセージがキューから除去されるように QueueConnectionFactory を設定できます (詳細については、933 ページの『ASF でのキューからのメッセージの除去』を参照してください)。 以下の 2 つの方法のいずれかで、この動作を設定できます。

- JMS 管理ツールを使用して、QueueConnectionFactory を MRET(NO) に設定する。
- プログラムで、以下を使用する。

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

この設定を変更しない場合、デフォルトでは、キュー上にこうした不必要なメッセージを保存します。

WebSphere MQ キュー・マネージャーをセットアップする際は、以下の点を考慮してください。

- 基礎 QLOCAL は、共用入力のために使用可能でなければなりません。 これを行うには、以下の MQSC コマンドを使用します。

```
ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)
```

- キュー・マネージャーは、使用可能な送達不能キューを所有していなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れる際に問題が発生する場合、基礎 QLOCAL からのメッセージ送達は停止します。 送達不能キューを定義するには、以下のものを使用します。

```
ALTER QMGR DEADQ(your.dead.letter.queue.name)
```

- ConnectionConsumer を実行するユーザーは、MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を所有していなければなりません。 詳細については、ご使用のプラットフォーム用の WebSphere MQ 資料を参照してください。
- 不必要なメッセージがキュー上に残されている場合、システム・パフォーマンスが低下します。 したがって、メッセージ・セレクター間で、ConnectionConsumer がキューからすべてのメッセージを除去するように、メッセージ・セレクターを計画してください。

MQSC コマンドの詳細については、MQSC リファレンスを参照してください。

ASF を使用したパブリッシュ/サブスクライブ・メッセージングの一般原則

ConnectionConsumers は、指定した Topic のメッセージを受信します。 ConnectionConsumer は、永続にも非永続にもできます。 ConnectionConsumer が使用するキュー (複数可) を指定する必要があります。

アプリケーションが TopicConnection オブジェクトから ConnectionConsumer を作成する際、アプリケーションは、Topic オブジェクトおよびセレクター・ストリングを指定します。 その後、ConnectionConsumer は、サブスクライブ対象のトピックに関する保存されているパブリケーションを含め、Topic オブジェクト上のセレクターと一致するメッセージを受信し始めます。

また、アプリケーションは、特定の名前に関連付けられた永続 ConnectionConsumer の作成もできます。 この ConnectionConsumer は、永続 ConnectionConsumer が最後にアクティブであったとき以降に、Topic についてパブリッシュされているメッセージを受信します。 この ConnectionConsumer は、Topic 上のセレクターと一致するすべてのメッセージを受信します。 しかし、ConnectionConsumer が先読みを使用する場合、クローズする際に、クライアント・バッファーにある非持続メッセージが失われることがあります。

WebSphere MQ classes for JMS が WebSphere MQ メッセージング・プロバイダーのマイグレーション・モードである場合、別個のキューが非永続 ConnectionConsumer サブスクリプションに使用されます。 TopicConnectionFactory 上の CCSUB 構成可能オプションは、使用するキューを指定します。 通常、CCSUB は、同じ TopicConnectionFactory を使用するすべての ConnectionConsumer が使用するための単一キューを指定する必要があります。 ただし、キュー名の接頭部とその後にアスタリスク (*) を指定することによって、各 ConnectionConsumer に一時キューを生成させることができます。

WebSphere MQ classes for JMS が WebSphere MQ メッセージング・プロバイダーのマイグレーション・モードである場合、Topic の CCDSUB プロパティーは、永続サブスクリプションに使用するキューを指定します。 ここでもまた、既に存在するキュー、またはキュー名の接頭部の後にアスタリスク (*) が付いたものになり得ます。既に存在するキューを指定すると、トピックをサブスクライブするすべての永続 ConnectionConsumer がこのキューを使用します。 キュー名の接頭部とその後にアスタリスク (*) を指定した場合、永続 ConnectionConsumer が指定の名前で初めて作成されるときにキューが生成されます。 このキューは、後に永続 ConnectionConsumer が同じ名前で作成される際に再使用されます。

WebSphere MQ キュー・マネージャーをセットアップする際は、以下の点を考慮してください。

- キュー・マネージャーは、使用可能な送達不能キューを所有していなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れる際に問題が発生する場合、基礎 QLOCAL からのメッセージ送達は停止します。 送達不能キューを定義するには、以下のものを使用します。

```
ALTER QMGR DEADQ(your.dead.letter.queue.name)
```

- ConnectionConsumer を実行するユーザーは、MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を所有していなければなりません。 詳細については、ご使用のプラットフォーム用の WebSphere MQ 資料を参照してください。
- 個々の ConnectionConsumer 用に、別個の専用キューを作成することによって、パフォーマンスを最適化することができます。 この場合、リソースが余分に使用されます。

ASF でのキューからのメッセージの除去

アプリケーションが ConnectionConsumer を使用する際、いくつかの状況では、JMS は、キューからメッセージを除去する必要があります。

これは、次のような状況です。

不適切なフォーマットのメッセージ
JMS が解析不能なメッセージが到着することがあります。

有害メッセージ
メッセージはバックアウトしきい値に達しますが、ConnectionConsumer は、バックアウト・キュー上へのメッセージのリキューに失敗します。

関係のない ConnectionConsumer
Point-to-Point メッセージングの場合、QueueConnectionFactory が不必要なメッセージを保存しないように設定されている場合、どの ConnectionConsumer にも不必要なメッセージが到着します。

これらの状態では、ConnectionConsumer は、キューからメッセージを除去しようとします。 メッセージの MQMD のレポート・フィールド内の後処理オプションは、正確な振る舞いを設定します。 そうしたオプションは、以下のとおりです。

MQRO_DEAD_LETTER_Q
メッセージは、キュー・マネージャーの送達不能キューにリキューされます。 これがデフォルトです。

MQRO_DISCARD_MSG
メッセージは破棄されます。

また、ConnectionConsumer はレポート・メッセージも生成しますが、これもメッセージの MQMD のレポート・フィールドに依存します。 このメッセージは、ReplyToQmgr 上のメッセージの ReplyToQ に送信されます。 レポート・メッセージが送信されている間にエラーがある場合、メッセージは、代わりに、送達不能キューに送信されます。 メッセージの MQMD のレポート・フィールド内の例外レポート・オプションは、レポート・メッセージの詳細を設定します。 そうしたオプションは、以下のとおりです。

MQRO_EXCEPTION
オリジナル・メッセージの MQMD を含むレポート・メッセージが生成されます。 このメッセージには、いかなるメッセージ本体データも含まれていません。

MQRO_EXCEPTION_WITH_DATA
MQMD、任意の MQ ヘッダー、および 100 バイトの本体データを含むレポート・メッセージが生成されます。

MQRO_EXCEPTION_WITH_FULL_DATA
オリジナル・メッセージからのすべてのデータを含むレポート・メッセージが生成されます。

default
レポート・メッセージは生成されません。

レポート・メッセージが生成される際、以下のオプションが有効です。

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID

- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

ポイズン・メッセージがリキューできない場合、たいていは送達不能キューがいっぱいであるか、許可が正しく指定されていないことが原因ですが、どのような動作になるかはメッセージが持続メッセージであるかどうかによって異なります。 メッセージが非持続メッセージである場合、メッセージは破棄され、レポート・メッセージは生成されません。 メッセージが持続メッセージである場合は、その宛先で listen しているすべての接続コンシューマーへのメッセージの送達が停止します。 これらの接続コンシューマーを閉じ、問題を解決する必要があります。その後、接続コンシューマーを再作成し、メッセージの送達を再開することができます。

送達不能キューを定義し、定期的にチェックして、問題が発生していないかを確認することは重要です。 特に、送達不能キューがその最大の深さに達していないこと、および、その最大メッセージ・サイズがすべてのメッセージに対して十分な大きさであることを確認してください。

メッセージが送達不能キューにリキューされるとき、メッセージの前に WebSphere MQ 送達不能ヘッダー (MQDLH) が付けられます。 MQDLH のフォーマットの詳細については、MQDLH - 送達不能ヘッダーを参照してください。 以下のフィールドによって、ConnectionConsumer が送達不能キューに入れたメッセージを識別したり、ConnectionConsumer が生成したメッセージを報告したりできます。

- PutApplType は、`MQAT_JAVA (0x1C)` です。
- PutApplName は、`"MQ JMS ConnectionConsumer"` です。

これらのフィールドは、送達キュー上のメッセージの MQDLH 内およびレポート・メッセージの MQMD 内にあります。 MQMD のフィードバック・フィールド、および MQDLH の Reason フィールドには、エラーを説明しているコードが含まれています。 これらのコードの詳細については、935 ページの『ASF での理由コードおよびフィードバック・コード』を参照してください。 他のフィールドについては、MQDLH - 送達不能ヘッダーを参照してください。

ASF での有害メッセージの処理

Application Server Facilities (ASF) では、ポイズン・メッセージの処理の仕方が WebSphere MQ classes for JMS 内の他の場所とは少し異なっています。

WebSphere MQ classes for JMS でのポイズン・メッセージの処理については、893 ページの『IBM WebSphere MQ classes for JMS でのポイズン・メッセージの処理』を参照してください。

Application Server Facilities (ASF) を使用する場合は、MessageConsumer の代わりに ConnectionConsumer で有害メッセージを処理します。 ConnectionConsumer は、キューの BackoutThreshold および BackoutRequeueQName プロパティーに従ってメッセージをリキューします。

アプリケーションが ConnectionConsumers を使用している場合、メッセージがバックアウトされる状況は、アプリケーション・サーバーが提供するセッションによって異なります。

- セッションが非トランザクション化セッションで、AUTO_ACKNOWLEDGE または DUPS_OK_ACKNOWLEDGE が指定されている場合、メッセージがバックアウトされるのは、システム・エラーの後、またはアプリケーションが予期せずに終了した場合のみです。
- セッションが非トランザクションで CLIENT_ACKNOWLEDGE を伴う場合、認められていないメッセージは、Session.recover() を呼び出すアプリケーション・サーバーによってバックアウトできます。

  通常、MessageListener またはアプリケーション・サーバーのクライアント実装は、Message.acknowledge() を呼び出します。Message.acknowledge() は、これまでにセッションに送達されたすべてのメッセージを認識します。
- セッションがトランザクション化セッションの場合、無応答メッセージは、アプリケーション・サーバーが Session.rollback() を呼び出すことによってバックアウトすることができます。
- アプリケーション・サーバーが XASession を提供する場合、メッセージは分散トランザクションに応じてコミットまたはバックアウトされます。 アプリケーション・サーバーは、トランザクションを完了させる責任があります。

WebSphere Application Server バージョン 5.0 およびバージョン 5.1 の組み込み JMS プロバイダーは、WebSphere MQ classes for JMS について説明された方法とは異なる方法で有害メッセージを処理します。

組み込み JMS プロバイダーが有害メッセージを処理する方法について詳しくは、関係する WebSphere Application Server 製品資料を参照してください。

エラーの処理

このセクションでは、935 ページの『ASF でのエラー状態からの回復』や935 ページの『ASF での理由コードおよびフィードバック・コード』などを含めて、エラー処理のさまざまな側面を取り上げます。

ASF でのエラー状態からの回復

ConnectionConsumer が重大エラーに遭遇した場合、同じ QLOCAL 内にインタレストを持つすべての ConnectionConsumer へのメッセージ送達は停止します。 このエラーの発生時には、影響を受けた Connection とともに登録されている ExceptionListener が通知されます。 アプリケーションがこれらのエラー状態から回復できる 2 つの方法があります。

通常、ConnectionConsumer が送達不能キューにメッセージをリキューできない場合にこのような重大なエラーが発生するか、または QLOCAL からメッセージを読み取るときにエラーに遭遇します。

影響を受けた Connection とともに登録されている ExceptionListener が通知されるため、これらを使用して問題の原因を特定できます。 問題解決のためにシステム管理者が介入しなければならない場合があります。

これらのエラー状態から回復するには、以下のいずれかの手法を使用してください。

- 影響を受けたすべての ConnectionConsumer で `close()` を呼び出します。 アプリケーションは、影響を受けたすべての ConnectionConsumer がクローズされて、システム問題が解決された後にのみ、新しい ConnectionConsumer を作成できます。
- 影響を受けたすべての Connection で `stop()` を呼び出します。 すべての Connection が停止され、システム問題が解決されると、アプリケーションは、その Connection を正常に `start()` できます。

ASF での理由コードおよびフィードバック・コード

理由コードおよびフィードバック・コードを使用して、エラーの原因を判別します。 ConnectionConsumer によって生成される一般的な理由コードをここで示します。

エラーの原因を判別するには、以下の情報を使用します。

- レポート・メッセージ内のフィードバック・コード
- 送達不能キュー内のメッセージの MQDLH 内にある理由コード

ConnectionConsumer は、以下の理由コードを生成します。

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

原因

メッセージは、QLOCAL 上に定義されている Backout Threshold に達したが、Backout Queue が定義されていません。

Backout Queue を定義できないプラットフォームでは、メッセージは、JMS 定義のバックアウトしきい値 20 に達します。

アクション

これが必要なければ、関係のある QLOCAL に Backout Queue を定義してください。 また、複数のバックアウトの原因も調べてください。

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

原因

Point-to-Point メッセージングで、キューをモニターしている ConnectionConsumer 用のセレクターのいずれかと一致しないメッセージが存在します。 パフォーマンスを保つために、メッセージが送達不能キューにリキューされます。

アクション

この状態を回避するには、キューを使用している ConnectionConsumer がすべてのメッセージを処理する一連のセレクターを提供するか、または QueueConnectionFactory を設定してメッセージを保存してください。

または、メッセージの送信元を調べてください。

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

原因
JMS が、キュー上のメッセージを解釈できません。

アクション
メッセージの発信元を調べてください。 通常、JMS は、BytesMessage または TextMessage として予期しないフォーマットのメッセージを送達します。 時折、メッセージが極めて不適切なフォーマットである場合には、このアクションは失敗します。

他のコードがこれらのフィールドに表示されるのは、Backout Queue へのメッセージのリキューが失敗したことが原因です。 この状態では、コードは、リキューが失敗した理由を説明しています。 これらのエラーの原因を診断するには、「API 理由コード」を参照してください。

レポート・メッセージを ReplyToQ に入れることができない場合、レポート・メッセージは送達不能キューに入れられます。 この状態では、MQMD のフィードバック・フィールドは、このトピックで説明されているとおりに埋められます。 MQDLH 内の理由フィールドは、レポート・メッセージを ReplyToQ に入れることができなかった理由を説明します。

AFS でのサーバー・セッション・プールの機能

このトピックでは、サーバー・セッション・プールの機能を要約しています。

937 ページの図 165 に、ServerSessionPool および ServerSession 機能の基本を要約します。

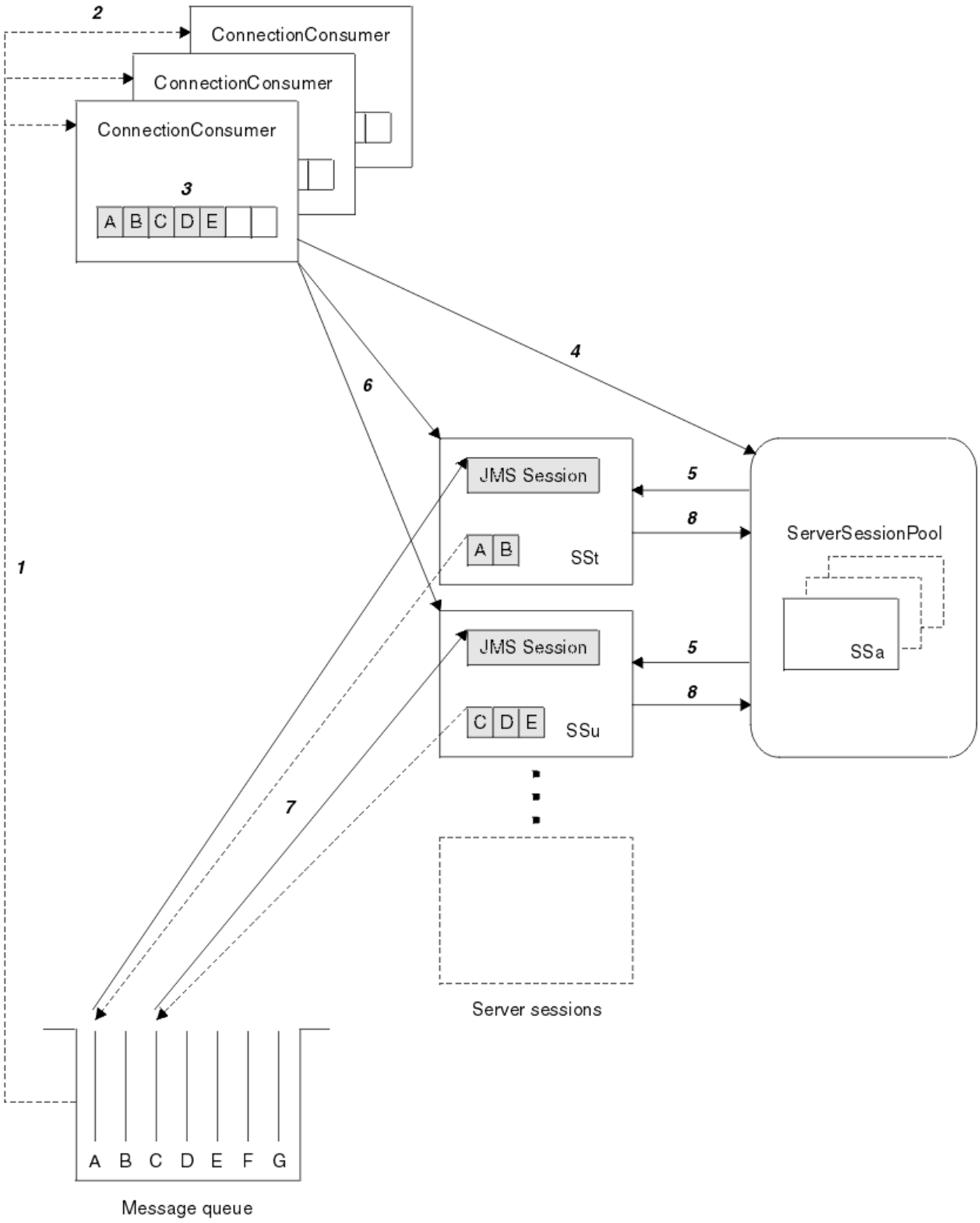


図 *165. ServerSessionPool* および *ServerSession* 機能

1. ConnectionConsumer は、キューからメッセージ参照を取得します。
2. それぞれの ConnectionConsumer は特定のメッセージ参照を選択します。
3. ConnectionConsumer バッファーは、選択されたメッセージ参照を保持します。
4. ConnectionConsumer は、ServerSessionPool から 1 つ以上の ServerSession を要求します。
5. ServerSession は ServerSessionPool から割り振られます。

6. ConnectionConsumer は、ServerSession にメッセージ参照を割り当て、ServerSession スレッドの実行を開始します。
7. 各 ServerSession は、キューからその参照されたメッセージを取得し、 JMS Session と関連のある MessageListener から onMessage メソッドにメッセージを渡します。
8. その処理を完了した後、ServerSession はプールに戻されます。

アプリケーション・サーバーは、ServerSessionPool および ServerSession 機能を提供します。

WebSphere MQ JMS 管理ツールの使用

管理ツールを使用して、8 つのタイプの WebSphere MQ classes for JMS オブジェクトのプロパティーを定義し、JNDI 名前空間に格納できます。 アプリケーションは、JNDI を使用して、これらの管理対象オブジェクトをネーム・スペースから取り出すことができます。

このツールでは、以下の WebSphere MQ classes JMS オブジェクトを管理できます。

- MQConnectionFactory
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

これらのオブジェクトについて詳しくは、942 ページの『JMS オブジェクトの管理』を参照してください。

このツールを使用するために必要なプロパティー・タイプおよび値については、IBM WebSphere MQ classes for JMS オブジェクトのプロパティーにリストされています。

このツールを使用して、管理者は、JNDI 内のディレクトリー・ネーム・スペースのサブコンテキストを操作することもできます。 942 ページの『WebSphere MQ JMS 管理ツールでのサブコンテキストの操作』を参照してください。

WebSphere MQ エクスプローラーを使って JMS 管理対象オブジェクトを作成および構成することもできます。

IBM WebSphere MQ classes for JMS 管理ツールの起動

管理ツールは、コマンド行インターフェースを備えています。 このインターフェースは、対話的に使用することも可能であり、また このインターフェースを使用してバッチ・プロセスを開始させることも可能です。

対話モードで使用する場合は、管理コマンドを入力するためのコマンド・プロンプトが表示されます。 バッチ・モードを使用する場合は、ツールを開始するコマンドの中に、管理コマンド・スクリプトが入っているファイルの名前が組み込まれています。

対話モード

管理ツールを対話モードで開始する場合は、次のコマンドを入力します。

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

ここで、

-t
トレースをオンにする (デフォルトはオフ)

このトレース・ファイルは "%MQ_JAVA_DATA_PATH%"\errors (Windows) または /var/mqm/trace (UNIX) に生成されます。 トレース・ファイルの名前は以下の形式になります。

mqjms_*PID*.trc

ここで、*PID* は JVM のプロセス ID です。

-v
詳細な出力を生成する (デフォルトは簡潔な出力)

-cfg config_filename
代替の構成ファイルの名前を指定する。 このパラメーターを省略した場合は、デフォルトの構成ファイル JMSAdmin.config が使用されます。 (939 ページの『JMS 管理ツールの構成』を参照してください。)

コマンド・プロンプトが表示されます。これは、管理ツールが管理コマンドを受け入れられる状態になったことを示します。 このプロンプトは、最初に、次のように表示されます。

```
InitCtx>
```

これは、現行コンテキスト (つまり、すべての名前指定およびディレクトリー操作が現在参照している JNDI コンテキスト) が、PROVIDER_URL 構成パラメーターに定義された初期コンテキストであることを示します (939 ページの『JMS 管理ツールの構成』を参照してください)。

ディレクトリーのネーム・スペースをトラバースすると、プロンプトにもそれが反映され、プロンプトには常に現行コンテキストが表示されます。

バッチ・モード

管理ツールをバッチ・モードで開始する場合は、次のコマンドを入力します。

```
JMSAdmin <test.scp
```

ここで、*test.scp* は、管理コマンドが入っているスクリプト・ファイルを表します (941 ページの『WebSphere MQ JMS 管理ツールの管理コマンド』を参照してください)。 ファイルの最後は、END コマンドでなければなりません。

JMS 管理ツールの構成

WebSphere MQ JMS 管理ツールでは、構成ファイルを使用して、特定のプロパティーの値を設定します。システムに合わせて調整できるサンプル・ファイルが提供されています。

構成ファイルは、等号 (=) で区切られたキーと値のペアのセットで構成されるプレーン・テキスト・ファイルです。 これは、以下の例に示されています。

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(行の最初の列にある「#」の文字は、コメントまたは使用されない行を表します)

WebSphere MQ には、サンプル構成ファイルが用意されています。 このファイルは JMSAdmin.config という名前で、 <MQ_JAVA_INSTALL_PATH>/bin ディレクトリーにあります。 システムのセットアップに合わせて、このファイルを編集してください。

管理ツールは、次のプロパティーの値を指定して構成します。

INITIAL_CONTEXT_FACTORY
ツールで使用するサービス・プロバイダーです。 このプロパティーでサポートされる値は、以下のとおりです。

- com.sun.jndi.ldap.LdapCtxFactory (LDAP 用)
- com.sun.jndi.fscontext.RefFSContextFactory (ファイル・システム・コンテキスト用)

上記のリストにはない InitialContextFactory を使用することもできます。 詳細については、940 ページの『リストにない InitialContextFactory を WebSphere MQ JMS 管理ツールで使用する』を参照してください。

PROVIDER_URL
セッションの初期コンテキストとなる URL、つまり、ツールで実行されるすべての JNDI 操作のルートとなる URL です。 このプロパティーは、次の 2 つの形式でサポートされています。

- ldap://hostname/contextname
- file:[drive:]/pathname

LDAP URL の形式は、LDAP プロバイダーに応じて変わる場合があります。 詳細は、LDAP の資料を参照してください。

SECURITY_AUTHENTICATION
JNDI でサービス・プロバイダーにセキュリティー信任状を渡すかどうかを指定します。 このプロパティーは、LDAP サービス・プロバイダーが使用されている場合にのみ使用されます。 このプロパティーには、次の 3 つの値のいずれかを指定できます。

- none (匿名認証)
- simple (単純認証)
- CRAM-MD5 (CRAM-MD5 認証メカニズム)

無効な値が指定された場合、プロパティーはデフォルトで none になります。 管理ツールでのセキュリティーに関する詳細は、940 ページの『JMS 管理ツールのセキュリティーの構成』を参照してください。

これらのプロパティーは、構成ファイル内に設定されます。 ツールを起動する際には、938 ページの『IBM WebSphere MQ classes for JMS 管理ツールの起動』で説明されているように、`-cfg` コマンド行パラメーターを使用してこの構成を指定できます。 構成ファイル名を指定しない場合、ツールはデフォルトの構成ファイル (`JMSAdmin.config`) をロードしようとします。 このファイルはまず現行ディレクトリーで検索され、次に `<MQ_JAVA_INSTALL_PATH>/bin` ディレクトリーで検索されます。ここで、`<MQ_JAVA_INSTALL_PATH>` は WebSphere MQ classes for JMS インストール済み環境へのパスです。

リストにない *InitialContextFactory* を *WebSphere MQ JMS* 管理ツールで使用する

2 つの InitialContextFactory 値がサポートされています。 JMS 管理構成ファイルでパラメーターを設定して、他の JNDI コンテキストを使用できます。

管理ツールを使用して、939 ページの『JMS 管理ツールの構成』でリストされている以外の JNDI コンテキストに接続できます。その際、JMSAdmin 構成ファイルで定義されている 3 つのパラメーターを使用します。

別の InitialContextFactory を使用するには、次のようにします。

1. INITIAL_CONTEXT_FACTORY プロパティーを、必要とするクラス名に設定します。
2. USE_INITIAL_DIR_CONTEXT、NAME_PREFIX、および NAME_READABILITY_MARKER プロパティーを使用して、InitialContextFactory の振る舞いを定義します。

これらのプロパティーの設定については、サンプル構成ファイルのコメントで説明されています。

サポートされている INITIAL_CONTEXT_FACTORY 値のうちのいずれかを使用する場合は、ここでリストされている 3 つのプロパティーを定義する必要はありません。 ただし、それらに値を指定してシステム・デフォルトをオーバーライドすることは可能です。 3 つの InitialContextFactory プロパティーのうちの 1 つ以上を省略した場合、管理ツールはその他のプロパティーの値に基づいて適切なデフォルトを提供します。

JMS 管理ツールのセキュリティーの構成

セキュリティー資格認定をサービス・プロバイダーに渡すかどうかを決定するには、SECURITY_AUTHENTICATION プロパティーを使用します。

SECURITY_AUTHENTICATION プロパティーについては、939 ページの『JMS 管理ツールの構成』 で説明されています。 このプロパティーの効果は次のとおりです。

- このパラメーターが none に設定されている場合、JNDI は、セキュリティー信任状を一切サービス・プロバイダーに渡さず、匿名認証 が実行されます。
- パラメーターが simple や CRAM-MD5 に設定された場合は、セキュリティー信任状が JNDI を介して下のサービス・プロバイダーに渡されます。 これらのセキュリティー信任状は、ユーザー識別名 (User DN) とパスワードの形式で与えられます。

セキュリティー資格認定が必要な場合は、ツールの初期設定時にこれらの入力を要求するプロンプトが出されます。 これは、JMSAdmin 構成ファイル内で PROVIDER_USERDN および PROVIDER_PASSWORD プロパティーを設定することにより、避けることができます。

注 : これらのプロパティーを使用しない場合、入力されたテキスト (パスワードを含む) が画面にもう一度表示されます。 これには、セキュリティー上の事柄が含まれている場合があります。

認証は、管理ツール自体が行うわけでなく、LDAP サーバーによって代行されています。 ディレクトリーの様々な部分に対するアクセス権限のセットアップと保守は、LDAP サーバーの管理者が行う必要があります。 詳細は、LDAP の資料を参照してください。 認証が失敗した場合、ツールは、適切なエラー・メッセージを表示して終了します。

セキュリティーおよび JNDI について詳しくは、Sun の Java Web サイト (https://java.sun.com) の資料を参照してください。

WebSphere MQ JMS 管理ツールの管理コマンド

管理ツールは、管理動詞とその適切なパラメーターで構成されたコマンドを受け入れます。

コマンド・プロンプトが表示されていれば、ツールはコマンドを受け入れられる状態にあります。 管理コマンドは、通常、次のような形式を取ります。

```
verb [param]*
```

ここで、**verb** の部分には、941 ページの表 133 にリストされているいずれかの管理動詞が入ります。 すべての有効なコマンドには、コマンドの先頭に、標準の形式か短縮された形式の動詞が 1 つ含まれています。

動詞に使用されるパラメーターは、その動詞によって異なります。 例えば、END 動詞はパラメーターを取ることができませんが、DEFINE 動詞は任意の個数のパラメーターを取ることができます。 1 つ以上のパラメーターを取る動詞については、関連トピックで詳しく説明します。

表 *133.* 管理動詞

動詞	短縮形	説明
ALTER	ALT (T)	管理対象オブジェクトのプロパティーを少なくとも 1 つ変更する
DEFINE	DEF	管理対象オブジェクトの作成、保管、またはサブコンテキストの作成を行う
DISPLAY	DIS	保管されている 1 つ以上の管理対象オブジェクトのプロパティー、または現行コンテキストの内容を表示する
削除	DEL	1 つ以上の管理対象オブジェクトをネーム・スペースから除去する、あるいは空のサブコンテキストを除去する
CHANGE	CHG	初期コンテキストの下に属する任意のディレクトリー・ネーム・スペースをトラバースし、現行コンテキストを変える (セキュリティーの許可は保留)
COPY	CP	保管されている管理対象オブジェクトのコピーを作成し、それを別名で保管する
MOVE	MV	管理対象オブジェクトの保管場所の名前を変える

表 133. 管理動詞 (続き)		
動詞	**短縮形**	**説明**
END		管理ツールを閉じる

動詞の名前は大/小文字を区別しません。

通常、コマンドの終わりには復帰キーを押します。 ただし、復帰キーを押す直前に正符号 (+) を入力すると、これをオーバーライドできます。 これによって、下の例に示されているように、複数行に渡ってコマンドを入力することが可能になります。

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

* # / のいずれかの文字で始まる行は、コメントとして処理され、無視されます。

WebSphere MQ JMS 管理ツールでのサブコンテキストの操作

ディレクトリー・ネーム・スペースのサブコンテキストの操作には、動詞 **CHANGE**、**DEFINE**、**DISPLAY**、および **DELETE** を使用します。

これらの動詞の使い方を 942 ページの表 134 に示します。

表 134. サブコンテキストの操作に使用されるコマンドの構文と説明	
コマンド構文	**説明**
DEFINE CTX(ctxName)	現行コンテキストに ctxName という名前の子サブコンテキストを作成する処理を試行します。 セキュリティー違反があった場合、そのサブコンテキストがすでに存在している場合、または無効な名前が指定された場合は、コマンドが失敗します。
DISPLAY CTX	現在のコンテキストの内容を表示します。 管理対象オブジェクトには a、サブコンテキストには [D] の注釈が付けられます。 各オブジェクトの Java タイプも表示されます。
DELETE CTX(ctxName)	現行コンテキストから ctxName という名前の子コンテキストを削除する処理を試行します。 そのコンテキストが見つからない場合、コンテキストが空でない場合、またはセキュリティー違反があった場合は、コマンドが失敗します。
CHANGE CTX(ctxName)	現行コンテキストを変更して、ctxName という子のコンテキストを参照するようにします。 ctxName には次の 2 つの特殊値があり、このいずれかを指定できます。 **=UP** 現在のコンテキストの親に移動する **=INIT** 初期コンテキストに直接移動する 指定されたコンテキストが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。

JMS オブジェクトの管理

このセクションでは、管理ツールで処理できる 8 つのタイプのオブジェクトについて説明します。 また、各オブジェクトに構成可能なプロパティーと、それらのプロパティーを操作するための動詞についても説明します。

WebSphere MQ エクスプローラーを使って JMS 管理対象オブジェクトを作成および構成することもできます。

JMS オブジェクト・タイプ

次の表に、管理対象オブジェクトの 8 つのタイプを示します。

「キーワード」という列は、944 ページの表 136 にあるコマンドの *TYPE* の部分と置き換えられるストリングを示します。

表 *135.* 管理ツールで処理される *JMS* オブジェクト・タイプ

オブジェクト・タイプ	キーワード	説明
MQConnectionFactory	CF	JMS ConnectionFactory インターフェースの WebSphere MQ 実装。 これは、Point-to-Point ドメインおよびパブリッシュ/サブスクライブ・ドメインの両方に接続を作成するためのファクトリー・オブジェクトを表します。
MQQueueConnectionFactory	QCF	JMS QueueConnectionFactory インターフェースの WebSphere MQ 実装。 これは、Point-to-Point ドメインに接続を作成するためのファクトリー・オブジェクトを表します。
MQTopicConnectionFactory	TCF	JMS TopicConnectionFactory インターフェースの WebSphere MQ 実装。 これは、パブリッシュ/サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。
MQQueue	Q	JMS Queue インターフェースの WebSphere MQ 実装。 これは、Point-to-Point ドメインにおけるメッセージの宛先を表します。
MQTopic	T	JMS Topic インターフェースの WebSphere MQ 実装。 これは、パブリッシュ/サブスクライブ・ドメインにおけるメッセージの宛先を表します。
MQXAConnectionFactory[944 ページの『1』]	XACF	JMS XAConnectionFactory インターフェースの WebSphere MQ 実装。 これは、XA バージョンの JMS クラスを使用する Point-to-Point ドメインおよびパブリッシュ/サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。
MQXAQueueConnectionFactory[944 ページの『1』]	XAQCF	JMS XAQueueConnectionFactory インターフェースの WebSphere MQ 実装。 これは、XA バージョンの JMS クラスを使用する Point-to-Point ドメインに接続を作成するためのファクトリー・オブジェクトを表します。
MQXATopicConnectionFactory[944 ページの『1』]	XATCF	JMS XATopicConnectionFactory インターフェースの WebSphere MQ 実装。 これは、XA バージョンの JMS クラスを使用するパブリッシュ/サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。

表 135. 管理ツールで処理される JMS オブジェクト・タイプ (続き)		
オブジェクト・タイプ	**キーワード**	**説明**
注: 1. これらのクラスは、アプリケーション・サーバーのベンダーが使用するために提供されたものです。 アプリケーション・プログラマーには直接的には役立たないと考えられます。		

JMS オブジェクトに使用される動詞

ディレクトリー・ネーム・スペース内の管理対象オブジェクトの操作には、動詞 ALTER、DEFINE、DISPLAY、DELETE、COPY、および MOVE を使用できます。

944 ページの表 136 に、これらの動詞の使い方を要約します。 *TYPE* の部分は、943 ページの表 135 にリストされている、必要な管理対象オブジェクトを表すキーワードに置き換えてください。

表 136. 管理対象オブジェクトの操作に使用されるコマンドの構文と説明	
コマンド構文	**説明**
ALTER *TYPE*(name) [property]*	管理対象オブジェクトのプロパティーを、指定された値に更新する処理を試行します。 セキュリティー違反があった場合、指定されたオブジェクトが見つからない場合、または指定された新しいプロパティーが無効である場合は、コマンドが失敗します。
DEFINE *TYPE*(name) [property]*	指定されたプロパティーを持つタイプ *TYPE* のオブジェクトを作成し、それを name という名前で現行コンテキストに保管する処理を試行します。 セキュリティー違反があった場合、指定された名前が無効であるかその名前のオブジェクトが存在している場合、または指定されたプロパティーが無効である場合は、コマンドが失敗します。
DISPLAY *TYPE*(name)	name という名前で現行コンテキストにバインドされている、タイプ *TYPE* の管理対象オブジェクトのプロパティーを表示します。 オブジェクトが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。
DELETE *TYPE*(name)	name という名前を持つタイプ *TYPE* の管理対象オブジェクトを現行コンテキストから除去する処理を試行します。 オブジェクトが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。
COPY *TYPE*(nameA) *TYPE*(nameB)	nameA という名前を持つタイプ *TYPE* の管理対象オブジェクトから、nameB という名前のコピーを作成します。 これはすべて現行コンテキストの有効範囲内で行われます。 コピー元のオブジェクトが存在しない場合、nameB という名前のオブジェクトが存在する場合、またはセキュリティー違反があった場合は、コマンドが失敗します。
MOVE *TYPE*(nameA) *TYPE*(nameB)	nameA という名前を持つタイプ *TYPE* の管理対象オブジェクトを、nameB という名前の管理対象オブジェクトに移動 (名前変更) します。 これはすべて現行コンテキストの有効範囲内で行われます。 移動するオブジェクトが存在しない場合、nameB という名前のオブジェクトが存在する場合、またはセキュリティー違反があった場合は、コマンドが失敗します。

WebSphere MQ JMS 管理ツールでのオブジェクトの作成

DEFINE コマンドを使用してオブジェクトを作成し、JNDI 名前空間に格納します。

次のコマンド構文を使用します。

```
DEFINE TYPE(name) [property]*
```

つまり、DEFINE 動詞の後に *TYPE*(name) 管理対象オブジェクト参照が続き、その後にゼロ個以上の プロパティー が続きます (IBM WebSphere MQ classes for JMS オブジェクトのプロパティー を参照)。

JMS オブジェクトに関する *LDAP* の命名の考慮事項
作成したオブジェクトを LDAP 環境に保管する場合、その名前は、一定の規則に準拠していなければなりません。 管理ツールを使用すると、デフォルトの接頭部を追加することによって、命名規則に準拠できます。

命名規則の 1 つとして、オブジェクトやサブコンテキストの名前には、cn= (共通名) や ou= (組織単位) といった接頭部を付けなければなりません。

この点、管理ツールでは、接頭部を付けなくてもオブジェクトやコンテキストの名前を参照できるようにすることで、LDAP サービス・プロバイダーの使用を簡単にしています。 ツールは、接頭部が入力されない場合に、自動的に、入力された名前にデフォルトの接頭部を付けます。 LDAP の場合、これは cn= です。

デフォルトの接頭部を変更するには、940 ページの『リストにない InitialContextFactory を WebSphere MQ JMS 管理ツールで使用する』の説明に従って JMSAdmin 構成ファイル内の NAME_PREFIX プロパティーを設定します。

例えば、次のようになります。

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

入力されたオブジェクト名 (testQueue) には接頭部が付いていませんが、ツールが LDAP の命名規則に合うように自動的に接頭部を付けています。 これと同様に、コマンド DISPLAY Q(testQueue) が出された場合でも、やはりこの接頭部が追加されます。

Java オブジェクトを保管するように LDAP サーバーを構成することが必要な場合があります。 この構成についての説明は、ご使用の LDAP サーバーの資料を参照してください。

JMS オブジェクトを作成する時のエラー状態の例

オブジェクトの作成時に、いくつかの一般的なエラー状態が発生する場合があります。

このようなエラー状態の例は、次のとおりです。

CipherSuite にマップされる CipherSpec

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

オブジェクトのプロパティーが無効

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

プロパティー値のタイプが無効

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

プロパティーの不調和 - クライアント/バインディング

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

プロパティーの不調和 - 出口の初期化

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

プロパティー値が有効範囲外

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

プロパティーが不明

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

以下は、JMS アプリケーションから JNDI 管理オブジェクトを探す時に Windows で起きる可能性のあるエラー状態の例です。

1. WebSphere JNDI プロバイダーの com.ibm.websphere.naming.WsnInitialContextFactory を使用している場合、サブコンテキストで定義されている管理オブジェクトにアクセスするには、スラッシュ (/) を使う必要があります。例えば、jms/MyQueueName です。 円記号 (¥) を使うと、InvalidNameException がスローされます。
2. Sun JNDI プロバイダーの com.ibm.websphere.naming.WsnInitialContextFactory を使用している場合、サブコンテキストで定義されている管理オブジェクトにアクセスするには、円記号 (¥) を使う必要があります。例えば、ctx1¥¥fred です。 スラッシュ (/) を使うと、NameNotFoundException がスローされます。

WebSphere MQ Explorer for JMS 構成の使用

IBM WebSphere MQ エクスプローラー・グラフィカル・ユーザー・インターフェースを使用して、JMS オブジェクトを WebSphere MQ オブジェクトから、WebSphere MQ オブジェクトを JMS オブジェクトから作成し、他の WebSphere MQ オブジェクトを管理およびモニターします。

始める前に

JMS 管理対象オブジェクトを WebSphere MQ エクスプローラーで作成して構成する前に、名前指定およびディレクトリー・サービスで JMS オブジェクトが保管される JNDI 名前空間のルートを定義する初期コンテキストを追加してください。 詳細については、JMS 管理対象オブジェクトについての IBM WebSphere MQ エクスプローラーのユーザー支援を参照してください。

このタスクについて

以下のタスクは、IBM WebSphere MQ エクスプローラーを使用して、IBM WebSphere MQ エクスプローラー内の既存のオブジェクトからコンテキストにより、または新規オブジェクト作成ウィザード内から実行できます。 いくつかの代表的なタスクでの WebSphere MQ エクスプローラーのユーザー支援の例については、WebSphere MQ Explorer ヘルプを参照してください。

手順

- 以下のいずれかの WebSphere MQ オブジェクトから JMS 接続ファクトリーを作成します。
  a) WebSphere MQ キュー・マネージャー (ローカル・コンピューターまたはリモート・システムのどちらでも)。
  b) WebSphere MQ チャネル

c) WebSphere MQ リスナー

- JMS 接続ファクトリーを使用した WebSphere MQ キュー・マネージャーの WebSphere MQ エクスプローラーへの追加
- WebSphere MQ キューからの JMS キューの作成
- JMS キューからの WebSphere MQ キューの作成
- WebSphere MQ トピックからの JMS トピックの作成。これは WebSphere MQ オブジェクトまたは動的トピックにすることができます。
- JMS トピックからの WebSphere MQ トピックの作成

WebSphere MQ Headers パッケージの使用

WebSphere MQ Headers パッケージは、メッセージの WebSphere MQ ヘッダーを操作するために使用できる、ヘルパー・インターフェースとクラスのセットを提供します。 通常、WebSphere MQ Headers パッケージを使用するのは、(プログラマブル・コマンド・フォーマット (PCF) メッセージを使用して) コマンド・サーバーで管理サービスを実行するためです。

このタスクについて

WebSphere MQ Headers パッケージは、`com.ibm.mq.headers` パッケージおよび `com.ibm.mq.pcf` パッケージに含まれています。 この機能は、 WebSphere MQ が Java アプリケーションで使用するために提供する以下の 2 つの代替 API の両方に使用できます。

- WebSphere MQ classes for Java (WebSphere MQ Headers Base Java とも呼ばれます)。
- WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS (WebSphere MQ JMS とも呼ばれる)。

WebSphere MQ Base Java アプリケーションは通常、MQMessage オブジェクトを操作します。ヘッダー・サポート・クラスは、 WebSphere MQ Base Java インターフェースをネイティブに理解するため、これらのオブジェクトと直接対話できます。

WebSphere MQ JMS では、メッセージのペイロードは、通常は、DataInput ストリームと DataOutput ストリームを使用して操作できるストリングまたはバイト配列オブジェクトです。 WebSphere MQ Headers パッケージは、これらのデータ・ストリームと対話するために使用することが可能で、WebSphere MQ JMS アプリケーションで送受信する MQ メッセージを操作するのに適しています。

したがって、 WebSphere MQ Headers パッケージには、 WebSphere MQ Base Java パッケージへの参照が含まれていますが、これも WebSphere MQ JMS アプリケーション内での使用を目的としており、 Java Platform, Enterprise Edition (Java EE) 環境内での使用に適しています。

WebSphere MQ Headers パッケージの一般的な使用方法は、例えば以下のような理由で、プログラマブル・コマンド・フォーマット (PCF) の管理メッセージを操作することです。

- WebSphere MQ リソースに関する詳細にアクセスするため。
- キューの深さをモニターするため。
- キューへのアクセスを禁止するため。

WebSphere MQ JMS API で PCF メッセージを使用することにより、この種のアプリケーション中心リソースの管理は、 WebSphere MQ Base Java API を使用することなく、 Java EE アプリケーション内から実行できます。

手順

- WebSphere MQ Headers パッケージを使用して、 WebSphere MQ classes for Java のメッセージ・ヘッダーを操作するには、 948 ページの『 WebSphere MQ classes for Java での使用』を参照してください。
- WebSphere MQ Headers パッケージを使用して JMS のメッセージ・ヘッダーを操作するには、948 ページの『WebSphere MQ classes for JMS との併用』を参照してください。

WebSphere MQ classes for Java での使用

WebSphere MQ classes for Java アプリケーションは通常、MQMessage オブジェクトを操作します。ヘッダー・サポート・クラスは、 WebSphere MQ classes for Java インターフェースをネイティブに理解するため、これらのオブジェクトと直接対話できます。

このタスクについて

WebSphere MQ には、WebSphere MQ Base Java API (WebSphere MQ classes for Java) で WebSphere MQ Headers パッケージを使用する方法を示すサンプル・アプリケーションがいくつか用意されています。

サンプルは、以下の 2 つの事柄を示しています。

- PCF メッセージを作成して管理アクションを実行し、応答メッセージを解析する方法。
- WebSphere MQ classes for Java を使用してこの PCF メッセージを送信する方法。

これらのサンプルは、使用しているプラットフォームに応じて、 WebSphere MQ インストール済み環境の `samples` ディレクトリーまたは `tools` ディレクトリーの `pcf` ディレクトリーの下にインストールされます (662 ページの『 WebSphere MQ classes for Java のインストール・ディレクトリー』 を参照)。

手順

1. 管理アクションを実行し、応答メッセージを解析するために、PCF メッセージを作成します。
2. WebSphere MQ classes for Java を使用して、この PCF メッセージを送信します。

関連概念

684 ページの『 WebSphere MQ classes for Java を使用した WebSphere MQ メッセージ・ヘッダーの処理』
さまざまなタイプのメッセージ・ヘッダーを表す Java クラスが提供されています。 2 つのヘルパー・クラスも提供されています。

689 ページの『 WebSphere MQ classes for Java を使用した PCF メッセージの処理』
PCF 構造化メッセージを作成および解析し、PCF 要求の送信と PCF 応答の収集を容易にするために、Java クラスが提供されています。

WebSphere MQ classes for JMS との併用

WebSphere MQ Headers を WebSphere MQ classes for JMS と共に使用するには、 WebSphere MQ classes for Java の場合と同じ必須ステップを実行します。 PCF メッセージは、 WebSphere MQ Headers パッケージと、 WebSphere MQ classes for Java の場合と同じサンプル・コードを使用して、まったく同じ方法で作成し、応答を解析することができます。

このタスクについて

WebSphere MQ API を使用して PCF メッセージを送信するには、メッセージ・ペイロードを JMS Bytes Message に書き込み、標準の JMS API を使用して送信する必要があります。 唯一の考慮事項として、メッセージには JMS RFH2 ヘッダーや MQMD に特定の値を指定した他のヘッダーを含めてはなりません。

PCF メッセージを送信するには、以下のステップを実行します。 PCF メッセージが作成され、応答メッセージから情報が抽出される方法は、 WebSphere MQ classes for Java の場合と同じです (948 ページの『 WebSphere MQ classes for Java での使用』 を参照)。

手順

1. SYSTEM.ADMIN.COMMAND.QUEUE を表す JMS のキュー宛先を作成します。

   WebSphere MQ JMS アプリケーションは、PCF メッセージを SYSTEM.ADMIN.COMMAND.QUEUE。このキューを表す JMS 宛先オブジェクトにアクセスする必要があります。 宛先には以下のプロパティーを設定する必要があります。

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

WebSphere Application Server を使用する場合は、これらのプロパティーを宛先のカスタム・プロパティーとして定義する必要があります。

アプリケーション内から宛先をプログラムで作成するには、以下のコードを使用します。

```
Queue q1 = session.createQueue("SYSTEM.ADMIN.COMMAND.QUEUE");
((MQQueue) q1).setIntProperty(WMQConstants.WMQ_MESSAGE_BODY,
    WMQConstants.WMQ_MESSAGE_BODY_MQ);
((MQQueue) q1).setMQMDWriteEnabled(true);
```

2. PCF メッセージを、適切な MQMD 値を含む JMS Bytes メッセージに変換します。

   JMS Bytes メッセージを作成し、そこに PCF メッセージを書き込む必要があります。 応答キューを作成する必要がありますが、固有の設定は必要ありません。

   以下のサンプル・コード・スニペットは、JMS Bytes メッセージを作成して、それに com.ibm.mq.headers,pcf.PCFMessage オブジェクトを書き込む方法を示しています。 PCFMessage オブジェクト (pcfCmd) は、WebSphere MQ Headers パッケージを使用して、事前に作成されています。 (PCFMessage をロードするパッケージが com.ibm.mq.headers.pcf.PCFMessage であることに注目してください)。

```
 // create the JMS Bytes Message
final BytesMessage msg = session.createBytesMessage();

// Create the wrapping streams to put the bytes into the message payload
ByteArrayOutputStream baos = new ByteArrayOutputStream();
DataOutput dataOutput = new DataOutputStream(baos);

// Set the JMSReplyTo so the answer comes back
msg.setJMSReplyTo(new MQQueue("adminResp"));

// write the pcf into the stream
pcfCmd.write(dataOutput);
baos.flush();
msg.writeBytes(baos.toByteArray());

// we have taken control of the MD, so need to set all
// flags in the MD that we require - main one is the format
msg.setJMSPriority(4);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_PERSISTENCE,
  CMQC.MQPER_NOT_PERSISTENT);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_EXPIRY, 300);
msg.setIntProperty(WMQConstants.JMS_IBM_MQMD_REPORT,
    CMQC.MQRO_PASS_CORREL_ID);
msg.setStringProperty(WMQConstants.JMS_IBM_MQMD_FORMAT, "MQADMIN");

// and send the message
sender.send(msg);
```

3. メッセージを送信し、標準の JMS API を使用して応答を受信します。
4. 応答メッセージを処理するために PCF メッセージに変換します。

   応答メッセージを取り出して、PCF メッセージとして処理するには、以下のコードを使用します。

```
 // Get the message back
BytesMessage msg = (BytesMessage) consumer.receive();

// get the size of the bytes message & read into an array
int bodySize = (int) msg.getBodyLength();
byte[] data = new byte[bodySize];
msg.readBytes(data);

// Read into Stream and DataInput Stream
ByteArrayInputStream bais = new ByteArrayInputStream(data);
DataInput dataInput = new DataInputStream(bais);

// Pass to PCF Message to process
PCFMessage response = new PCFMessage(dataInput);
```

関連概念

811 ページの『JMS メッセージ』
JMS メッセージは、ヘッダー、プロパティー、および本文で構成されます。 JMS では、5 つのタイプのメッセージ本体を定義します。

WebSphere MQ での Web サービスの使用

IBM WebSphere MQ transport for SOAP または IBM WebSphere MQ bridge for HTTP を使用して、Web サービス用の IBM WebSphere MQ アプリケーションを開発できます。

IBM WebSphere MQ transport for SOAP は、SOAP のための JMS トランスポートを提供します。 IBM WebSphere MQ transport for SOAP は、 Microsoft Windows Communication Foundation、 WebSphere Application Server、および CICS Transaction Server などの他の環境にも統合されます。

IBM WebSphere MQ transport for SOAP の詳細については、 950 ページの『WebSphere MQ transport for SOAP』を参照してください。

IBM WebSphere MQ Bridge for HTTP を使用すると、クライアント・アプリケーションは、WebSphere MQ MQI クライアントをインストールしなくても、 IBM WebSphere MQ とメッセージを交換できます。 HTTP 機能を使って任意のプラットフォームまたは言語から WebSphere MQ を呼び出すことができます。

IBM WebSphere MQ Bridge for HTTP について詳しくは、 1028 ページの『WebSphere MQ bridge for HTTP』を参照してください。

関連概念

8 ページの『アプリケーション開発の概念』
選択した手続き型言語またはオブジェクト指向言語を使用して、 IBM WebSphere MQ アプリケーションを作成することができます。 アプリケーション開発者に役立つ IBM WebSphere MQ の概念については、このトピックのリンクを使用してください。

78 ページの『使用するプログラミング言語の決定』
この情報を使用して、IBM WebSphere MQ によってサポートされているプログラミング言語およびフレームワークについて調べ、それらを使用するための考慮事項を検討してください。

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

96 ページの『WebSphere MQ プログラムのサンプル』
この一連のトピックでは、さまざまなプラットフォーム上での WebSphere MQ プログラムのサンプルを紹介しています。

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

353 ページの『クライアント・アプリケーションの作成』
WebSphere MQ でクライアント・アプリケーションを作成するために知っておくべき内容

277 ページの『パブリッシュ/サブスクライブ・アプリケーションの作成』
パブリッシュ/サブスクライブ WebSphere MQ アプリケーションの作成を開始します。

429 ページの『IBM WebSphere MQ アプリケーションの構築』
この情報を使用して、各種のプラットフォームでの IBM WebSphere MQ アプリケーションの構築について学習します。

553 ページの『プログラム・エラーの処理』
ここでは、ご使用のアプリケーションの MQI 呼び出しで、呼び出しを行うときや、メッセージを最終宛先に送信するときに発生するエラーについて説明します。

WebSphere MQ transport for SOAP

WebSphere MQ transport for SOAP は、SOAP のための JMS トランスポートを提供します。 WebSphere MQ transport for SOAP は、Microsoft Windows Communication Foundation、WebSphere Application Server、CICS Transaction Server などの他の環境に統合することもできます。

IBM WebSphere MQ transport for SOAP の概要

IBM WebSphere MQ transport for SOAP は、SOAP のための JMS トランスポートを提供します。 WebSphere MQ SOAP の送信側およびリスナーは、Web サービスを呼び出す手段を提供します。

WebSphere MQ SOAP リスナーは、.NET Framework 1、.NET Framework 2、および Axis 1.4 でホストされているサービスをサポートします。 WebSphere MQ SOAP 送信側は、.NET Framework 1、.NET Framework 2、Axis 1.4、および Axis2 で実行される Web サービス・クライアントをサポートします。 クライアントは、WebSphere MQ サーバー・アプリケーションまたはクライアント・アプリケーションのどちらかにすることができます。 IBM WebSphere MQ transport for SOAP は、Microsoft Windows Communication Foundation、WebSphere Application Server、CICS Transaction Server などの他の環境に統合することもできます。

Microsoft Windows Communication Foundation への統合は、IBM WebSphere MQ support for .NET Framework 3 の一部となっています。

IBM WebSphere MQ transport for SOAP は、IBM WebSphere MQ で JMS を使用して SOAP メッセージをトランスポートするためのプロトコルとツールのセットです。 これには、951 ページの表 137 に示すさまざまなアプリケーション環境用にさまざまなパッケージがあります。

表 *137. IBM WebSphere MQ transport for SOAP* アプリケーション環境

	追加の WebSphere MQ コンポーネントとの統合	**フレームワークへの統合**
WebSphere MQ インストールの 一部として提供	.NET Framework 1 。NET Framework 2 軸 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (クライアントのみ)
別のソフトウェア・パッケージで 提供		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

IBM WebSphere MQ transport for SOAP をアプリケーション・フレームワークと統合することにより、IBM WebSphere MQ への Web サービスの開発とデプロイメントが簡単になります。

追加の IBM WebSphere MQ SOAP コンポーネントを使用すれば、 WebSphere MQ SOAP コンポーネントと直接対話して、サービスを開発およびデプロイできます。 Web サービスと Web サービス・クライアントを構成して、IBM WebSphere MQ にデプロイするには、IBM WebSphere MQ SOAP ツールを使用します。

統合環境では、開発とデプロイメントはより簡単になります。 SOAP HTTP Web サービスの開発とデプロイには、今まで開発とデプロイに使ってきたのと同じツールを使用します。 WebSphere MQ ツールを使用して、必要な IBM WebSphere MQ キュー、チャネル、およびキュー・マネージャーを構成する必要があります。

これらのあらゆる環境から IBM WebSphere MQ SOAP クライアントとサーバーをミックス・アンド・マッチすることができます。

利点

WebSphere MQ transport for SOAP を使用する場合、既存の IBM WebSphere MQ ユーザーには以下の主な利点があります。

IBM WebSphere MQ ネットワークを使用して既存の Web サービスに接続する。

この場合のサービスは、自分で書いたサービスか、あるいはデプロイした他のパッケージ・ソフトウェア・アプリケーションへのインターフェースとして提供しているサービスである可能性があります。

この利点は、既存の WebSphere MQ ネットワークを使用して Web サービスに接続できることにあります。 IBM WebSphere MQ transport には、管理された信頼性のあるキュー型メッセージング・サービスであることの利点があります。

新規のアプリケーションを記述するか、または既存のアプリケーションを変換して、IBM WebSphere MQ インターフェースではなく SOAP を使用するようにする。

通常、アプリケーションを他のアプリケーションと統合するには、特定の WebSphere MQ アダプターを開発することが必要です。 アダプターには 2 つのパーツがあります。コネクター部分がトランスポートに対してメッセージの送受信をします。アダプター部分がデータをアプリケーション固有のフォーマットに変換および逆変換します。 アプリケーションの各ペアを統合するのは、新たな挑戦になります。

SOAP の利点は、アプリケーション・インターフェース定義の SOAP による標準化、そして、トランスポートの選択ができることにあります。 アプリケーション固有のアダプターを書く必要はありません。そして、コネクターに IBM WebSphere MQ と HTTP のどちらを使用するかを選択できます。 どのトランスポートを選択するかは、必要なサービス品質と接続性品質によって異なります。

既存の SOAP over HTTP ユーザーにとっては、WebSphere MQ transport for SOAP の利点は、管理され、信頼性のある非同期トランスポートが使用できることです。 利点は 2 つあります。

可用性とパフォーマンスに優れた真の非同期プログラミング・モデル。
非同期クライアント・インターフェースを使用すれば、クライアント・アプリケーションとサービス・アプリケーションを同時に有効にする必要がなくなります。 クライアントによって送信される要求は、サービスが要求を処理するために使用可能になるまで保管されます。

信頼性と可用性に優れた設計の、すぐに作動可能な管理対象ネットワーク。

IBM WebSphere MQ をトランスポートとして選択すると、高信頼性メッセージングを提供する管理対象ネットワークを使用することによる利点が得られます。

これとは対照的に、TCP/IP 上の HTTP および FTP などのトランスポートは管理されていません。 管理されていないネットワークは、接続が予測不能である場合には理想的です。管理タスクがほとんどないからです。

要約

IBM WebSphere MQ transport for SOAP は、以下のコンポーネントを提供します。

- SOAP/JMS トランスポート・バインディングが WSDL 文書で使用されていますが、これは SOAP サービスを JMS トランスポートにバインドするためのものです。 WebSphere MQ による SOAP/JMS バインディングの実装は、次の 2 つの形式のどちらかの URI を使います。

  WebSphere MQ transport for SOAP

  ```
  jms:/queue?&Name=Value&Name=Value...
  ```

  W3C Candidate Recommendation 用の WebSphere MQ ワイヤー・フォーマット

  ```
  jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
  ```

- SOAP メッセージの WebSphere MQ メッセージへのマッピング。
- SOAP 要求を受信する 2 つの IBM WebSphere MQ SOAP リスナー。1 つは Java 用、もう 1 つは .NET Framework 1 または .NET Framework 2 用です。 リスナーは、.NET または Axis 1.4 を使用して SOAP 要求を処理します。
- IBM WebSphere MQ SOAP 要求を作成する 2 つの IBM WebSphere MQ SOAP 送信側。 Web サービス・クライアントは、`jms:` SOAP 要求を処理するため、送信側に登録します。
- Windows Communication Foundation (WCF) (.NET 3 とも呼ばれる) との統合。これを使用して、WebSphere MQ transport for SOAP のメッセージ送受信を行います。
- クライアントと Axis2 (JAX-WS とも呼ばれる) との統合。WebSphere MQ transport for SOAP のメッセージまたは W3C SOAP JMS メッセージの送信を行います。
- コマンド **amqwdeployWMQService**。これは、IBM WebSphere MQ transport for SOAP を使用して、Web サービスをデプロイするための開発用とランタイム用のコンポーネントおよびスクリプトを作成します。
- サンプルの Java および .NET クライアントとサービス・コード。
- クラスパスを設定するスクリプト、およびその他のユーティリティー・スクリプト。

統合環境では、送信側とリスナーは各環境に統合されます。これらは開発ツールおよびデプロイメント・ツールの拡張となります。

SOAP と WebSphere MQ の統合

WebSphere MQ transport for SOAP は SOAP、および Web サービスのツールとランタイムを拡張します。これは、HTTP の代わりに WebSphere MQ を使って SOAP トランスポートを行います。 トランスポートとして WebSphere MQ transport for SOAP を使用するために既存の Web サービスを変更する必要はありません。 トランスポートでは、SOAP/JMS 用にカスタム URI 形式を使用します。 SOAP/JMS 用の W3C URI 形式は、制限はありますが Axis2 クライアントでサポートされています。

.NET Framework 1、.NET Framework 2、および Axis 1.4 環境のクライアントでは、コードに追加の行を加える必要があります。 Axis 2 および Windows Communication Foundation (WCF) クライアントでは、追加のコードは必要ありません。 .NET Framework 1、.NET Framework 2、および Axis 1.4 環境では、WebSphere MQ SOAP リスナーはサービスを実行します。 WebSphere MQ transport for SOAP は、WCF、CICS、および WebSphere Application Server など、他のいくつかのアプリケーション・サーバー環境にも統合できます。

SOAP とは

SOAP[9] アプリケーションが要求、応答、およびデータグラムを交換するために使用するメッセージおよび対話プロトコルの標準化された形式について説明します。 SOAP はメッセージ転送のために使用されるトランスポートに依存しません。また、メッセージの送受信を行うアプリケーション環境にも依存しません。W3C は SOAP バージョン 1.2 を次のように簡潔に定義しています。

SOAP バージョン 1.2 は、非集中型の分散環境においてピア間で構造化された型付き情報を交換するために使用できる *XML* ベースの情報の定義を提供します。[10].

SOAP を使用するには、HTTP、E メール、あるいは WebSphere MQ などのトランスポートに SOAP をバインドしておかなければなりません。

SOAP プロトコル・バインディング・フレームワークは、HTTP など他のプロトコルの上で SOAP メッセージを運ぶための規則をセットにしたものです。 SOAP Version 1.2 Part 2: Adjuncts (Second Edition) には、SOAP HTTP バインディングが記述されています。

W3C Candidate Recommendation, 4 June 2009, SOAP over Java Message Service 1.0 には、SOAP JMS バインディングの推奨事項が記述されています。 JMS は API 仕様であって、トランスポート・プロトコルではないので、JMS SOAP 勧告では SOAP JMS メッセージのワイヤー・フォーマットについては記述していません。 記述されているのは、SOAP 対話プロトコルと JMS API バインディングです。 結果として、JMS SOAP 勧告を利用する場合であってもやはり、SOAP クライアントと SOAP サーバーで同じ JMS 実装を使用しなければなりません。 これは、JMS の任意の実装環境で SOAP JMS アプリケーションを実行できるようにします。 J2EE アプリケーション・サーバーと JMS 実装環境の両方が JCA 仕様に準拠していれば、その JMS 実装環境をそのサーバーにプラグインできます。 WebSphere MQ JMS は JCA 仕様に準拠しており、同様に準拠したアプリケーション・サーバーにプラグインできます。

WebSphere MQ transport for SOAP バインディングは、W3C 標準候補に似ていますが、同じではありません。 この使用法については、トピック MQRFH2 SOAP 設定値に説明があります。 W3C Candidate Recommendation とは異なり、SOAP バインディングは正式に指定されてはいません。 これは事実上、HTTP バインディングであり、サービス・アドレスは `http://authority/path?query#fragment` ではなく、`jms:/queue?name=value&name=value...` の形式を取ります。 `jms:` は、公式に登録された IANA URI スキームではありません。

Web サービスとは

SOAP は、異なる言語で書かれた、異なるプラットフォーム上で実行されるプログラムが、さまざまなトランスポート・プロトコルを使用して通信できるようにします。 SOAP はプロトコル仕様です。 Web サービスは、インターネット・プロトコルを使用してアクセス可能な SOAP インターフェースを介してサービスを提供するアプリケーションです。

---

[9] 歴史的に、頭字語は Simple Object Access Protocol (Simple Object Access Protocol) の略だった。
[10] W3C: SOAP バージョン 1.2 パート 0

SOAP の重要な目的は、クライアントにとって使いやすいサービスを提供することです。 いったん、サービスを使用するクライアントを設計したら、外部資料を参照せずにそのサービスへの呼び出しをプログラムできます。 サービス・インターフェースは、XML で、WSDL 文書として記述します。 `http://authority/path?wsdl` に照会すると、SOAP サービスの WSDL 記述が返されます。

ヒント: WebSphere MQ を使用する Web サービスをデプロイするときは、標準 WSDL 照会が動作するように、HTTP へのサービスもデプロイしてください。

Web サービスの開発

Web サービスにはクライアント部分とサービス部分があります。 サービスを最初に記述します。これは、WSDL に書かれたインターフェース記述から始めるか、または、サービス・クラスの記述規則に従って行います。 Web サービス・ツールキットには、クラスのインターフェース定義から WSDL を生成するユーティリティーが含まれています。例えば、**java2wsdl** や **disco** です。 また、WSDL インターフェース記述からスケルトンを生成またはクラス化するためのツールもあります。例えば、**wsdl2java**、**wsimport**、または **wsdl** などです。 前者はボトムアップ開発、後者はトップダウン開発と呼ばれます。

WebSphere MQ transport for SOAP の **amqwdeployWMQService** コマンドでは、これらのツールを使用して、WSDL、クライアント・スタブ、およびクライアント・プロキシーを生成します。

一般に Web サービスは、以下のような特定のアプリケーション・サーバー環境をターゲットとした統合開発環境を使用して記述されます。

Eclipse IDE for Java EE 開発者
Axis 2 用の Web サービスを作成します。 JAX-RPC および JAX-WS のサポート

Rational® Application Developer V7.5
WebSphere Application Server V7 およびそれ以前のバージョン用の Web サービスを作成します。 Axis 用も作成できます。 JAX-RPC と JAX-WS をサポートします。

WebSphere Integration Developer V6.2
WebSphere Process Server および WebSphere ESB 用 Web サービスを作成します。 JAX-RPC と JAX-WS をサポートします。

Visual Studio 2008 (バージョン 9)
.NET Framework 3.5 以前 (Windows Communication Foundation) 用の Web サービスを作成します。

Visual Studio 2005 (バージョン 8)
.NET Framework 2 およびそれ以前用の Web サービスを作成します。

これらのツールのどれでも、WebSphere MQ transport for SOAP と組み合わせて使用できます。 HTTP で使用するようにサービスを開発したら、**amqwdeployWMQService** ツールを使用して、WebSphere MQ をトランスポートとして使用するようにサービスをデプロイします。 このツールの出力を使用して新規のクライアントを記述することも、WebSphere MQ transport for SOAP を使用するように既存のクライアントを変更することもできます。

WebSphere MQ transport for SOAP をアプリケーション環境に統合する場合は、**amqwdeployWMQService** ツールを使ったり、クライアント・コードを変更したりする必要はありません。 クライアント SOAP レイヤーは、プレフィックス `jms:` が付いた URI を持つクライアント要求を、WebSphere MQ transport for SOAP に送信します。 サーバー SOAP レイヤーは、WebSphere MQ transport for SOAP を呼び出して `jms:` SOAP 要求を待機し、WebSphere MQ transport for SOAP に応答を返します。

通常、.NET サービスはコード内で Web サービス・アノテーションを使用してボトムアップで開発され、Java サービスは WSDL インターフェース定義を使用してトップダウンで開発されています。 Java Standard Edition バージョン 6 は JAX-WS 2.0 をサポートし、アノテーションを使用してサービス・インターフェースの定義を限定するため、アプローチの違いは狭まります。 Java サービスをトップダウンでボトムアップで簡単に開発できるようになりました。 どちらのアプローチを選択するかは、開発方式の問題です。

Web サービス・クライアントは、サービスを記述した後、WSDL サービス定義と生成されたクライアント・スタブおよびプロキシーを使用して記述します。 アプリケーションによっては、クライアントを記述するときにサービス定義がわからない場合があります。 その場合のクライアントは、サービス WSDL を取得してサービス要求をダイナミックに作成します。 より一般的には、サービス定義はわかっているが、サービ

スのデプロイ先のアドレスがわかっていない場合があります。 Web サービス・ツールキットはそのようなクライアントのために、サービス要求を作成するために使用するインターフェースを生成します。 そのクライアントは、必要になったときにサービス・アドレスを提供します。 3 番目のケースは、クライアントが必要とする情報すべてが WSDL に含まれている場合です。 その場合の WSDL には、インターフェースとサービスのアドレスの両方が含まれています。 Web サービス・ツールキットが生成するコードには、クライアントがサービスに要求をするために必要な情報すべてが含まれています。

WebSphere MQ transport for SOAP では、上記 3 種類のスタイルのどれでも使用できます。

Web サービス・アプリケーション環境

Web サービス・ツールキットは、サービスの WSDL 定義から、SOAP 要求/応答で転送されるバイト・ストリームへのマッピングを必要とします。 このバイト・ストリームは SOAP 仕様で定義され、SOAP エンベロープに格納されます。 SOAP エンベロープを 955 ページの図 166 に示します。

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
<!-- headers... -->
</soap:Header>
<soap:Body>
<!-- payload or fault message -->
</soap:Body>
</soap:Envelope>
```

図 *166. SOAP* エンベロープ

SOAP エンベロープから言語バインディングへのマッピングおよび逆方向のマッピングは、一部は標準化されていますが、一部はプロプラエタリーです。 .NET アーキテクチャーではマッピングは基本的な事柄であり、共通言語ランタイム (CLR) の一部として提供されています。 このマッピングは、JAX 仕様によって Java で標準化されています。 Java マッピングは標準化されているため、Java Web サービス・クライアントおよびサービスは、異なる Java ベース・アプリケーション環境間で移植可能です。 JAX-RPC (JAX-WS 1.0 とも呼ばれる) は、現在最も多く使用されているマッピングです。 これは、Axis 1.4 でサポートされています。 JAX-WS (JAX-WS 2.0 とも呼ばれる) は大きく改善された標準であり、JAX-RPC を急速に置き換えようとしています。 JAX-WS は Axis 2.0 でサポートされています。 WebSphere MQ 7.0.1 は JAX-WS と Axis 2 をサポートしていません。

WebSphere MQ transport for SOAP は SOAP エンベロープのコンテンツを変更しませんし、コンテンツはトランスポートに影響を与えません。 一方、言語バインディングは WebSphere MQ transport for SOAP に影響を与えます。 WebSphere MQ 7.0.1 は、WebSphere MQ transport for SOAP に添付されたコードとユーティリティーを使用して、.NET Framework 1、.NET Framework 2、および Axis 1.4 をサポートします。 .NET Framework 3 および 3.5 における WebSphere transport for SOAP のサポートは、Windows Communication Foundation 用 WebSphere MQ カスタム・チャネルを使用して実装されています。

他の SOAP 開発環境およびランタイム環境には、WebSphere MQ transport for SOAP のサポート、およびさまざまな言語のサポートが添付されている場合があります。 例えば、CICS 上で実行される Web サービスでは、COBOL や PL/1 などの言語をサポートします。

注 : 使用されるマッピングによって、Web サービスの相互運用性に違いが生じることはありません。 .NET、JAX-RPC、および JAX-WS それぞれのマッピングを使用して書かれたクライアントとサービスをミックス・アンド・マッチできます。

WebSphere MQ transport for SOAP とは

WebSphere MQ transport for SOAP は、SOAP バインディングと Web サービス・ツールキットです。 これらを共に使用することにより、アプリケーションで、HTTP ではなく WebSphere MQ を使用した SOAP メッセージ交換ができるようになります。956 ページの図 167 は、SOAP トランスポートとして HTTP の代わりに WebSphere MQ を示しています。

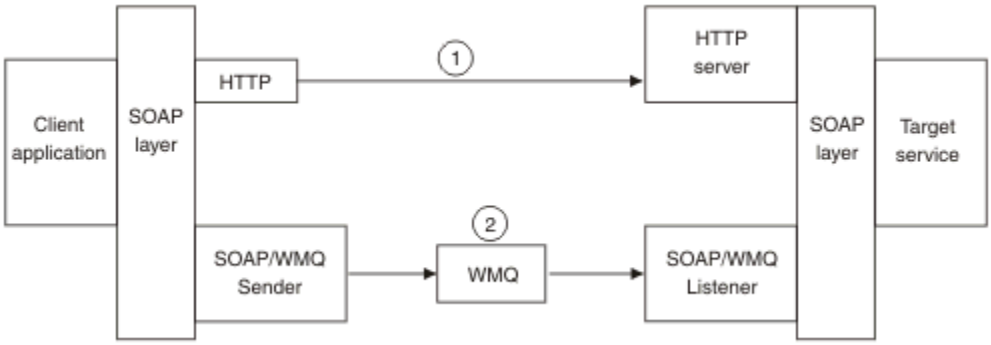


図 *167. WebSphere MQ transport for SOAP* の概要

図の (1) に示したのが SOAP over HTTP です。 クライアント SOAP レイヤーは要求を SOAP メッセージに変換し、HTTP コンポーネントが TCP/IP 上に送信します。 HTTP サーバー・コンポーネントは HTTP 要求を (一般には、TCP/IP port 80 を) listen します。 要求が SOAP サービスに対するものである場合、HTTP サーバー・コンポーネントは SOAP レイヤーを呼び出して SOAP 要求をメソッド呼び出しに変換します。 それから、応答が返されます。

SOAP over WebSphere MQ を (2) に示します。 クライアント・アプリケーションは WebSphere MQ SOAP 送信側コンポーネントを `jms:` プロトコルと SOAP レイヤーのハンドラーとして登録します。 SOAP レイヤーは `jms:` にアドレス指定された SOAP メッセージを WebSphere MQ SOAP 送信側に渡します。 送信側はメッセージ内の URI を使用して、メッセージを必要なサービス品質を持つ要求キューに配置します。 対応する WebSphere MQ SOAP リスナーは要求キュー上にメッセージが来るのを待機し、SOAP レイヤーを呼び出して要求を処理して応答を返します。

SOAP 送信側およびリスナーは、通常の WebSphere MQ プログラムです。 これらは、957 ページの図 168 に示すように同一のキュー・マネージャーに接続することができます。あるいは、異なるキュー・マネージャーに接続することもできます。958 ページの図 169 を参照してください。 クライアントはクライアント接続によって接続されます。

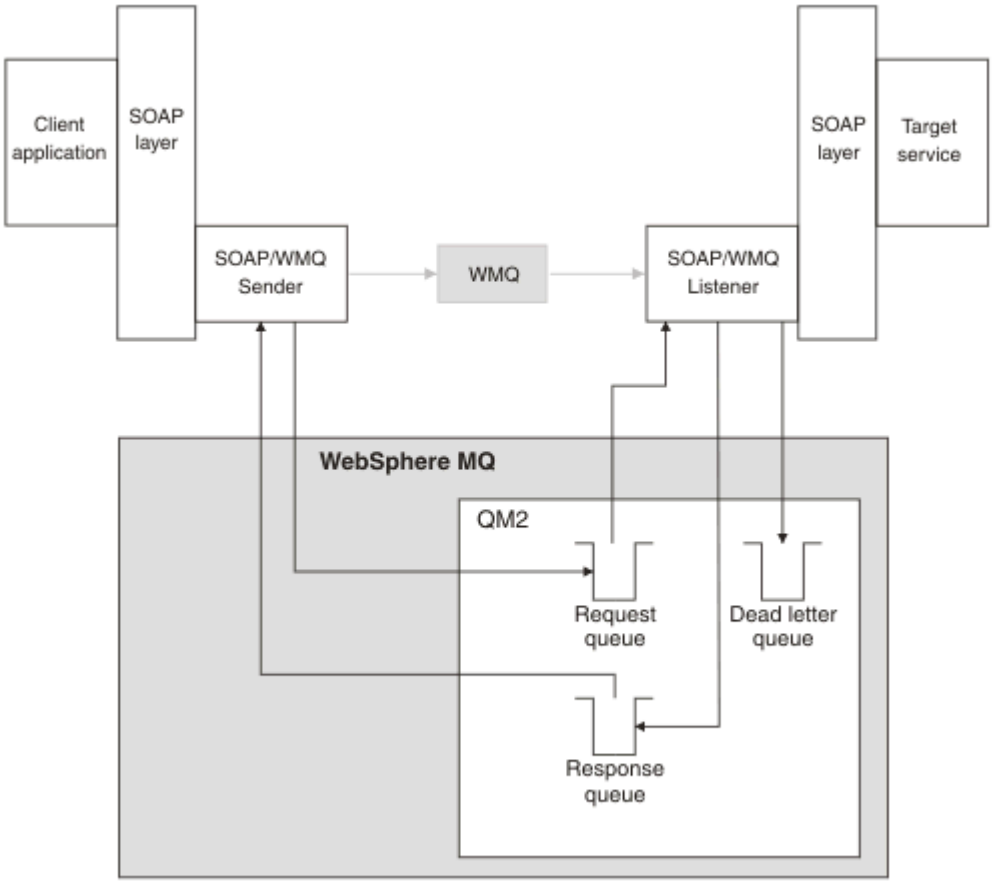


図 *168. SOAP/WebSphere MQ* で使用するキュー *(*単一キュー・マネージャー*)*

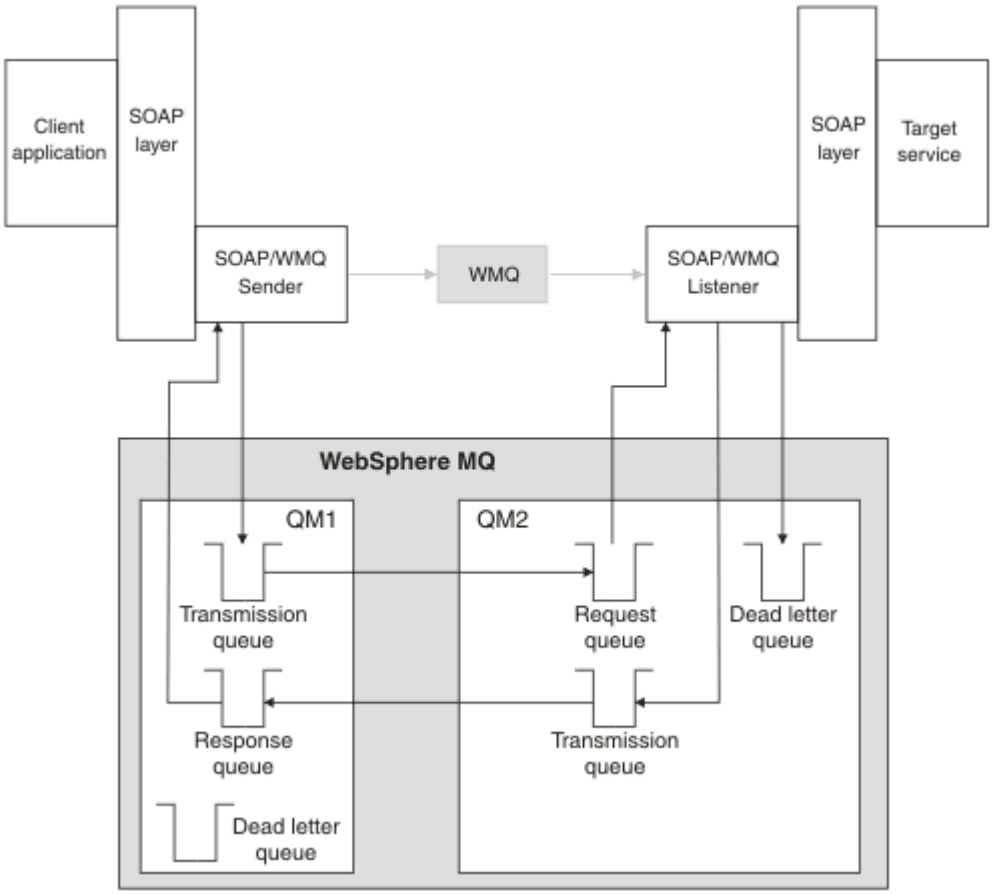


図 *169. SOAP/WebSphere MQ* で使用するキュー *(*個別のキュー・マネージャー*)*

W3C Candidate Recommendation による SOAP から JMS へのバインディング

W3C 勧告候補は、SOAP over JMS バインディングの SOAP over Java Message Service 1.0 を定義します。また、以下の例にも役立ちます (Java(tm) メッセージ・サービス 1.0 の URI スキーム)[11].

アプリケーション・フレームワークによっては、WebSphere Application Server v7 のように、W3C Candidate Recommendation をサポートしているものがあります。 W3C Candidate Recommendation と互換性のある URI 付きでフォーマットされた SOAP 要求を Axis2 クライアントを使用して送信します。 WebSphere MQ Axis 2 クライアント用の W3C SOAP over JMS URI を参照してください。 Axis2 クライアントは、SOAP 要求内の URI に基づき、W3C フォーマットまたは WebSphere MQ transport for SOAP フォーマットの SOAP 要求を送信します。

Axis2 クライアントによる W3C Recommendation のサポートは、7.0.1.3 フィックスパックで導入されました。 他のクライアントのサポートおよび WebSphere MQ が提供する SOAP リスナーのサポートは提供されません。

関連概念

.NET Framework 1、.NET Framework 2、および Axis 1.4 上での、WebSphere transport for SOAP の実装
独自の WebSphere MQ SOAP 送信側およびリスナーを記述する計画をする場合があります。 .NET Framework 1、.NET Framework 2、および Axis 1.4 上での WebSphere MQ transport for SOAP の実装をガイドとして使用してください。

WebSphere MQ transport for SOAP および Web サービス高信頼性メッセージング
Web サービス高信頼性メッセージングは、信頼性の低い接続上で信頼性の高い Web サービス要求/応答を交換するためのプロトコルです。 これは、短時間の接続中断問題を解決するのに最適です。

11 最新のドラフトについて、*W3C* 仕様参照内の JMS の URI スキームを探します。

.NET Framework 1、.NET Framework 2、および Axis 1.4 上での、WebSphere transport for SOAP の実装

独自の WebSphere MQ SOAP 送信側およびリスナーを記述する計画をする場合があります。 .NET Framework 1、.NET Framework 2、および Axis 1.4 上での WebSphere MQ transport for SOAP の実装をガイドとして使用してください。

1. クライアント・プログラムでは、HTTP トランスポートについて行うのと同様に、適切な Web サービス・フレームワークを使用します。 `jms:` プレフィックスを登録することも必要です。 接頭部は、`com.ibm.mq.soap.Register.extension()` Java メソッドまたは `IBM.WMQSOAP.Register.Extension()` CLR メソッドのいずれかを使用して登録されます。
2. Axis 1.4、.NET Framework 1 または 2 のフレームワークは、SOAP/HTTP の場合と全く同じように、呼び出しを SOAP 要求メッセージにマーシャルします。
3. WebSphere MQ サービスは、`jms:` プレフィックスの付いた URI によって識別されます。 フレームワークは `jms:` URI を識別すると、WebSphere MQ transport の送信側コードである `com.ibm.mq.soap.transport.jms.WMQSender` (Axis 1.4 の場合) または `IBM.WMQSOAP.MQWebRequest` (.NET1 および 2 の場合) を呼び出します。 フレームワークは `http:` プレフィックス付きの URI を検出すると、標準の SOAP over HTTP 送信側を呼び出します。
4. SOAP メッセージは WebSphere MQ SOAP 送信側が要求キューを使用してトランスポートします。 **`SimpleJavaListener`** (Java の場合) または **`amqwSOAPNETListener`** (.NET の場合) が要求メッセージを受け取ります。

   WebSphere MQ SOAP リスナーはスタンドアロン・プロセスであり、スレッド数をカスタマイズ可能なマルチスレッドを持ちます。
5. WebSphere MQ SOAP リスナーは着信 SOAP 要求を読み取り、それを該当する Web サービス・インフラストラクチャーに渡します。
6. Web サービス・インフラストラクチャーは、SOAP 要求メッセージを解析してサービスを呼び出しします。 この手順は HTTP トランスポートに到着したメッセージの場合と同じです。
7. インフラストラクチャーはその応答を SOAP 応答メッセージにフォーマットし、WebSphere MQ SOAP リスナーに返します。
8. リスナーはメッセージを応答キューに置き、メッセージは WebSphere MQ SOAP 送信側に転送されます。 送信側はこれをクライアント Web サービス・インフラストラクチャーに渡します。
9. クライアント・インフラストラクチャーは、応答 SOAP メッセージを解析し、その結果をクライアント・アプリケーションに返します。

各アプリケーション・コンテキストには、個別の WebSphere MQ 要求キューからサービスが提供されます。

Axis 1.4 の場合、アプリケーション・コンテキストは、WebSphere MQ SOAP リスナーおよびサービスが適切なディレクトリーにおいて確実に実行されるように制御されます。 Axis 1.4 は、そのディレクトリーに正しい CLASSPATH を設定します。

.NET の場合、アプリケーション・コンテキストは、WebSphere MQ SOAP リスナーが、`ApplicationHost.CreateApplicationHost` の呼び出しで作成されたコンテキストでサービスを実行することによって制御されます。 この呼び出しによって、ターゲット実行ディレクトリーが指定されます。 この後、各サービスが、デプロイされたディレクトリーで作動します。

`amqwdeployWMQService` は、要求キューと応答キューを生成します。 また、キューの処理と Axis 1.4 へのサービスのデプロイに必要なインフラストラクチャーも生成します。

関連概念

SOAP と WebSphere MQ の統合

WebSphere MQ transport for SOAP および Web サービス高信頼性メッセージング

Web サービス高信頼性メッセージングは、信頼性の低い接続上で信頼性の高い Web サービス要求/応答を交換するためのプロトコルです。 これは、短時間の接続中断問題を解決するのに最適です。

WebSphere MQ transport for SOAP および *Web* サービス高信頼性メッセージング

Web サービス高信頼性メッセージングは、信頼性の低い接続上で信頼性の高い Web サービス要求/応答を交換するためのプロトコルです。 これは、短時間の接続中断問題を解決するのに最適です。

WebSphere MQ for SOAP は、SOAP メッセージの引き渡しに WebSphere MQ の管理対象高信頼性ネットワークを利用します。 HTTP および FTP などのトランスポートは管理されていません。 管理されていないネットワークは、接続が予測不能である場合には理想的です。要求/応答が失われないことによる利点よりも、接続管理の難しさとコストのほうが上回ってしまうからです。

管理されていないネットワークにおいて接続が切断した場合にファイルが失われる問題を克服するため、管理対象 FTP などのサービスによって、FTP の上部に管理レイヤーを構築します。 管理レイヤーはユーザーからのファイル転送が成功したかを検査する責任を担い、必要に応じて欠落ファイルを再送します。 管理対象 FTP を使用するには、接続の両端に管理ソフトウェアをインストールしておかなければなりません。

Web サービス高信頼性メッセージング (WSRM) では、これとは異なったアプローチによって信頼性の低い接続における問題を解決します。 目的は、接続の両端で同じソフトウェアを使わなくても、Web サービスの要求/応答を高信頼性転送にすることです。 どのソフトウェアでも、Web サービス高信頼性メッセージング・プロトコルを実装すると、他のソフトウェアと信頼性の高いメッセージ交換ができます。

接続に障害が起こると、送信側と受信側は、生成された URI をキーとして使用して、WSRM メッセージ転送のコンテキストを保存する必要があります。 送信側と受信側は、新しい接続の確立を試行し続けます。 新しい接続の確立に成功したら、転送が完了します。 WSRM 仕様では、コンテキストの保存方法や新しい接続を試行するタイミングについては、定めていません。

短時間の障害のみが重要だと判断される場合もあります。 より長時間の障害の場合、ある程度の時間がたっても転送を再開できなければ、その転送を破棄することにするかもしれません。 同様に、クライアントとサービスのどちらかに障害が起きた場合も、その転送を破棄することにするかもしれません。 転送の保証をユーザー責任にしておくのなら、クライアントとサービス間の調整を管理する必要はほとんどありません。

ネットワーク障害が 30 分を超えるような長時間に及ぶ場合、あるいはクライアントかサーバーに障害が起こった場合、接続のいくつかはもはや再確立できない可能性が高くなります。 管理されない方法での WSRM によるメッセージ転送の自動復元にはもはや依存できなくなります。 WSRM 接続の障害の管理を考慮する必要があります。このことは、クライアントとサービスのネットワークを管理するためのソフトウェアを開発することを意味します。

短時間の障害を克服するために WSRM を使用すれば、モバイル・ネットワークにおいてメッセージが失われる問題に対する処理を大幅に削減できます。 メッセージの送達を自分で保証する必要がなくなるのなら、メッセージ損失を削減されるという利点は、WSRM 実装の開発に追加コストを払う根拠になります。

SOAP over JMS では、メッセージの送達を保証し、長時間に及ぶクライアント、サーバーおよびネットワークの障害を処理します。 SOAP のサービス品質に HTTP より高い信頼性を求めているのなら、WebSphere MQ transport for SOAP と WSRM のどちらのソリューションを選択すべきでしょうか。 その答えは多くの要因に依存します。 考慮すべき要因には、次にリストするものが挙げられます。

1. 信頼性低下が接続障害によるものかどうか。
2. 接続障害がどれほど長時間に及ぶか。
3. 接続のクライアント側とサーバー側の両方を管理できるかどうか。
4. ユーザーまたは管理者にメッセージ送達の最終的な責任があるかどうか。

関連概念

SOAP と WebSphere MQ の統合

.NET Framework 1、.NET Framework 2、および Axis 1.4 上での、WebSphere transport for SOAP の実装

独自の WebSphere MQ SOAP 送信側およびリスナーを記述する計画をする場合があります。 .NET Framework 1、.NET Framework 2、および Axis 1.4 上での WebSphere MQ transport for SOAP の実装をガイドとして使用してください。

WebSphere MQ Web サービスのインストールおよび検証

これらのトピックにある指示を使用して、WebSphere MQ transport for SOAP をインストールおよび検証します。

WebSphere MQ Web transport for SOAP のインストール

WebSphere MQ Web transport for SOAP をインストールするには、この手順で行います。 このインストールにより、WebSphere MQ を SOAP トランスポートとして使用して Web サービス・クライアントまたはサービスを実行するためのツールが作成されます。 これらのツールは、.NET Framework 1、.NET 2、Axis 1.4、または Axis2 SOAP 環境で使用されます。

始める前に

IBM WebSphere MQ のシステム要件を参照して、前提条件となる製品を確認します。 インストール・プロセスでは、前提ソフトウェアが存在するかどうかおよび使用可能かどうかの検査は行われません。 前提条件ソフトウェアがインストールされていることを検証する必要があります。

WebSphere MQ は、Axis 1.4 ランタイムのコピーを提供します。 これ以外のバージョンがインストールされている場合でも、それではなく、このバージョンを WebSphere MQ と一緒に使用してください。 IBM は、Apache Axis に関する技術サポートを提供していません。 Apache Axis に技術的な問題がある場合は、The Apache Software Foundation にお問い合わせください。

.NET Framework 3 SOAP 環境で Web サービスを実行するためには、WebSphere MQ は Windows Communication Foundation を使用します。 Windows Communication Foundation 用の WebSphere MQ カスタム・チャネルは、WebSphere MQ を SOAP メッセージのトランスポートとして使用して Web サービス・クライアントおよびサービスを実行します。

このタスクについて

WebSphere MQ Web transport for SOAP は、WebSphere MQ MQI のクライアント・アプリケーションかサーバー・アプリケーションのどちらかとしてインストールできます。 WebSphere MQ がクライアントまたはサーバーとして既にコンピューターにインストールされている場合は、リストされているコンポーネントがインストール済みであることを確認してください。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

以下のインストール・ステップを実行します。

手順

1. インストールする "「Java and. Net Messaging and Web services」" コンポーネントを選択します。
2. Solaris および HP-UX の場合、インストールする "「Java ランタイム環境」" コンポーネントを選択します。
3. インストールの対象として「開発ツールキット」を選択します。
4. ご使用のプラットフォーム向けの「スタートアップ・ガイド」に記載されているとおりに WebSphere MQ をインストールして検査します。
5. WebSphere MQ インストール・メディアの `prereqs/axis` ディレクトリーにある Apache Axis 1.4 ランタイム `axis.jar` をコピーします。 コピー先は、962 ページの表 138、962 ページの表 139、または 962 ページの表 140 に示したインストール・ディレクトリーです。

   Windows

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

   AIX

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

   HP-UX、Solaris、および Linux (すべてのプラットフォーム) のインストール・ディレクトリー

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

6. Windows 2003 では、**Aspnet_regiis.exe** を実行して、使用する共通言語ランタイムのバージョンを示すようにスクリプト・マップを更新します。

   %SystemRoot%Microsoft.NET/Framework/*version-number* で **Aspnet_regiis.exe** ユーティリティーを探します。
7. 環境変数 WMQSOAP_HOME を、WebSphere MQ インストール・ディレクトリーを示すように設定します。

タスクの結果

表 *138. Windows* インストール・ディレクトリー	
ロケーション	**目次**
MQ_INSTALLATION_PATH\programs\bin	バイナリー、コマンド、DLL、および実行可能ファイル
MQ_INSTALLATION_PATH\programs\java\lib	.jar files
MQ_INSTALLATION_PATH\programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH\programs\soap\samples	サンプルおよび IVT

表 *139. AIX* インストール・ディレクトリー	
ロケーション	**目次**
MQ_INSTALLATION_PATH/bin	シェル・スクリプト
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar およびその他の JAX-RPC .jar ファイル
MQ_INSTALLATION_PATH/samp/soap	サンプルおよび IVT

表 *140. HP-UX*、*Solaris*、および *Linux (*すべてのプラットフォーム*)* のインストール・ディレクトリー	
ロケーション	**目次**
MQ_INSTALLATION_PATH/bin	シェル・スクリプト
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar およびその他の JAX-RPC .jar ファイル
MQ_INSTALLATION_PATH/samp/soap	サンプルおよび IVT

次のタスク

1. .NET の場合にのみ、WebSphere MQ transport for SOAP ファイルをグローバル・アセンブリー・キャッシュに登録する必要があります。 WebSphere MQ のインストール時に .NET が既にインストールされている場合、登録はインストール時に自動的に行われます。 .NET を WebSphere MQ の後にインストールした場合は、IVT の最初の実行時に登録が自動的に行われます。

   amqiregisterdotnet.cmd を実行すると、 .NET アセンブリーの登録を行うことができます。 また、どの段階であっても、**amqiregisterdotnet.cmd** を実行して強制的に再登録することができます。 一度登録すると、システム再始動を行った後もその登録は有効であるため、以降の登録は通常は必要ありません。
2. 963 ページの『IBM WebSphere MQ transport for SOAP の検査』 に示されているように、インストール検査テストを実行します。

3. Axis2 クライアントを開発する場合は、Apache から Axis2 1.4.1 をダウンロードする必要があります (986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』を参照)。

IBM WebSphere MQ transport for SOAP の検査

runivt コマンドを使用して、 IBM WebSphere MQ transport for SOAP を検証します。 このコマンドは、いくつかのデモンストレーション・アプリケーションを実行し、インストール後の環境が正しくセットアップされていることを確認します。

始める前に

runivt コマンドを実行する前に、以下のランタイム環境があることを確認します。

- Axis でのみ実行する場合: システムで Java SDK (SOE 内) が使用可能になっている必要があります。 また、システムの **PATH** 環境変数に `java.exe` および `javac.exe` コマンドの場所を組み込む必要もあります。
- .NET でのみテストを実行するには (Windows でのみサポート)、ご使用のシステムに Java SDK と .NET コンパイラーおよびツールの両方が必要です。 そのようにするには、Visual Studio コマンド・プロンプトまたは Microsoft Windows SDK コマンド・プロンプトにアクセスしてから、`java.exe` ファイルと `javac.exe` ファイルの場所を **PATH** 環境変数に追加します。
- 使用可能なすべてのテストを実行する場合: Windows プラットフォームでは、.NET でのテスト実行の説明に従って環境を構成する必要があります。 UNIX and Linux プラットフォームでは、Axis でのみのテスト実行の説明に従って環境を構成する必要があります。

このタスクについて

.NET と Axis の両方に対して検証テストを実行するのではなく、Axis のみまたは .NET のみに対してテストを実行した方がよい場合があります。

テストで問題が発生したために再度開始する場合は、以下のようにします。

1. `immediate` オプションを使用してキュー・マネージャー `WMQSOAP.DEMO.QM` を停止します。
2. 別ウィンドウで開始されたリスナーを停止します。
3. キュー・マネージャーを削除してください。
4. 作成した一時 samples ディレクトリーを削除し、再度開始します。

UNIX and Linux プラットフォームでは、X Windows System セッションを使用してコマンドを実行する必要があります。

runivt コマンドは、`soap/samples` ディレクトリーの内容を変更します。 インストール・イメージを変更しないでおくには、samples ディレクトリーを一時ロケーションにコピーし、その一時ロケーションから検証テストを実行します。

必要に応じて何回でもインストール検査を実行できます。

.NET Framework 1、.NET Framework 2、および Axis 1.4 上の IBM WebSphere MQ transport for SOAP のインストールを検査するには、以下のステップを実行します。

手順

1. `./tools/soap/samples` ディレクトリー・ツリーを一時ロケーションにコピーします。
2. 一時ディレクトリーを現行ディレクトリーとしてコマンド・ウィンドウを開始します。
3. **runivt** コマンドを使用してインストール・テストを開始します。 `runivt` スクリプトは、テスト・クラス、サンプル・クライアント、およびサービスをデプロイして実行する前にコンパイルします。 テスト・クラス、サンプル・クライアント、および実行するサービスについては、「WebSphere(r) MQ Web transport for SOAP のインストール」で概説されているインストール手順を実行し、 `runivt` コマンドの実行に使用するコマンド・プロンプトに必要なランタイム環境が設定されていることを確認します。 **runivt** コマンドを実行するには、以下のいずれかの方法を使用します。

- Axis でのみテストを実行: `runivt Axis`
- .NET でのみテストを実行 (Windows でのみサポートされます): `runivt DotNet`
- 使用可能なすべてのテストを実行: `runivt`

`runivt` コマンド構文およびパラメーターについては、 **runivt**: IBM WebSphere MQ transport for SOAP インストール検査テストを参照してください。 実行できるテストは、ファイル `ivttests.txt` (Windows の場合) および `ivttests_unix.txt` (UNIX and Linux プラットフォームの場合) にリストされています。

関連資料
runivt: WebSphere MQ transport for SOAP インストール検査テスト

WebSphere MQ transport for SOAP 対応 Web サービスの作成

通常の Web サービス開発環境を使用して、WebSphere MQ transport for SOAP に対応したサービスを作成します。

始める前に

1. WebSphere MQ transport for SOAP に付属のコマンド行ツールを使用する場合は、以下が前提条件となります。
   a. サービスのデプロイメント・ディレクトリーを作成します。
   b. そのディレクトリーでコマンド・ウィンドウを開始します。
   c. .NET の場合は、csc.exe と wsdl.exe がパスになければならず、かつ同じバージョンの .NET Framework からのものでなければなりません。
   d. Java の場合:
      i) **amqwsetcp** コマンドを実行してクラスパスをセットアップします。
      ii) 同じバージョン・レベルの IBM JRE および JDK が現行パスになければなりません。 バージョン・レベルは、少なくとも 5.0 である必要があります。
      iii) 追加の `.jar` ライブラリーの場所と、作成するサービス用を含む .java パッケージを含むディレクトリーが含まれるように、クラスパスをカスタマイズします。 クラスパスに現行ディレクトリー `"."` を挿入します。
      iv) 作成するサービスのパッケージ名に対応したディレクトリーを、コマンド・ウィンドウの現行ディレクトリーと相対的な場所に作成します。
2. あるいは、Web サービス作成をサポートするワークベンチ・ツールを使用します。 開発タスクの例では、 Microsoft Visual Studio 2008、 Eclipse IDE for Java EE Developers および WebSphere Application Server Community Edition を使用します。

このタスクについて

既存の Web サービスは、WebSphere transport for SOAP と連動させるための変更を必要としません。 WebSphere MQ transport for SOAP に付属のツールは、Web サービスをデプロイし、WebSphere MQ SOAP リスナーを使用してそれを実行します。 また、これらのツールは、WSDL、.NET クライアント・スタブ、および `.java` プロキシー・クラスを生成して、WebSphere MQ transport for SOAP クライアントを作成します。

サービスを作成し、サービスのデプロイメントとクライアントの生成の準備をするには、以下のステップを実行します。 Eclipse または Microsoft Visual Studio 2008 を使用してサービスを作成するには、関連タスクで示すステップに従ってください。

手順

1. 通常の開発環境を使用してサービスを作成します。
2. HTTP Web サービス・クライアントを使用してサービスをテストします。

3. デプロイメント・ディレクトリーを準備するには、以下のステップを実行します。
   - Java の場合
     a. サービス・インターフェースを定義した .java ファイルをデプロイメント・ディレクトリーにコピーします。
     b. サービスの .class ファイルを、パッケージ名に対応したディレクトリーにコピーします。
     c. クラスパスが必要なすべてのクラスを見つけることができることを確認します。 **javac** を使用して、サービスの .java ファイルをコンパイルします。
   - .NET の場合
     a. サービスを定義した .asmx ファイルをデプロイメント・ディレクトリーにコピーします。
     b. 分離コード・モデルを使用している場合は、.dll ファイルを *deployment directory*\bin ディレクトリーにコピーします。

WebSphere MQ transport for SOAP 対応 *JAX-RPC* サービスを *Eclipse* を使用して作成する

WebSphere MQ をサービス・プロバイダーとして使用して実行する Axis 1.4 Web サービスを作成します。通常の Web サービス開発環境を使用して、Axis 1.4 へのデプロイメント用のサービスを作成します。

始める前に

Axis 1.4 用 WebSphere MQ SOAP リスナーに Web サーバーをデプロイするための要件を考慮に入れてください。

- Axis 1.4 用 WebSphere MQ SOAP リスナーは、バージョン 5.0 以上の IBM JRE を必要とします。 作成に使用される JRE と JDK は、同じバージョン・レベルでなければなりません。
- Axis 1.4 用 WebSphere MQ SOAP リスナーは、WebSphere と共にインストールされる axis.jar を必要とします。 開発環境にインストールされた axis.jar ファイルではなく、WebSphere MQ でインストールされた axis.jar ファイルを参照するように、開発環境のビルド・パスを変更します。
- WebSphere MQ V7.0.1 までは、デプロイされるサービス用に生成される WSDL は、RPC/encoded です。V7.1 から、RPC/literal スタイルの WSDL も要求できるようになりました。 生成された WSDL は、デプロイメントにのみ使用されます。 WS-I 準拠の WSDL を使用してサービスを定義できます。

このタスクについて

通常の Web サービス開発環境を使用してサービスを作成します。

このタスクでは、自由に使用できるオープン・ソースの Eclipse Java EE IDE for Web Developers (Galileo と呼ばれます) を使用します。 アプリケーション・サーバーには、Geronimo に基づいた WebSphere Application Server Community Edition v2.1 (Community Edition) を使用します。 IDE とサーバーを入手、インストール、および構成する方法については、関連タスクを参照してください。

以前 IDE で提供されていた Web Services Explorer を使用して HTTP をトランスポートとして使用するサービスをテストします。 あるいは、独自のクライアント・コードを使用して HTTP クライアント・プロキシーを生成し、サービスをテストします。

以下のステップに従って、ボトムアップ Web サービスを作成することができます。 サンプル・プログラム StockQuoteAxis.java を例として使用してください。

手順

1. 新しいワークスペースを使用して Eclipse IDE for Java EE Developers を開始します。
2. Java50 を使用するようにワークスペースを構成します。
   WebSphere Application Server Community Edition 2.1.4 は、Java60 とは連動しません。
   a) **ウィンドウ** > **設定** > **Java** > **インストール済み JRE** > **追加 ...** > **標準 VM** > **次へ** > **ディレクトリー ...**
   b) **Java50** > **OK** > **Finish** のインストール・ディレクトリーを参照します。

c) **Java50** JRE> OK を確認します。

3. Community Edition ランタイム環境を追加し、Community Edition を開始します。
   a)「**ウィンドウ**」 >「**設定**」 >「**サーバー**」 >「**ランタイム環境**」 >「**追加 ...**」
   b)「**新規サーバー・ランタイム環境**」 リストから **IBM WASCE v2.1**」 を選択し、 「**新規ローカル・サーバーの作成**」 >「**次へ**」 にチェック・マークを付けます。
   「**IBM WASCE 2.1**」がリストにない場合は、そのほかに以下の 2 つのタスクを完了する必要があります。
      i) WebSphere Application Server Community Edition をインストールします。
      ii) Community Edition 用 Eclipse アップデートをインストールします。
   詳しくは、WebSphere Application Server Community Edition を参照してください。
   c) アプリケーション・サーバーのインストール・ディレクトリー>「**OK**」 >「**終了**」 >「**OK**」を参照します。
   d) サーバー・ビューで 「**IBM WASCE v2.1 サーバー**」 を右クリックして、 「**開始**」 を選択します。
   ヒント : Eclipse で WASCE を管理できます。 **IBM WASCE v2.1 サーバー** >「**WASCE コンソールの起動**」 を右クリックします。 デフォルトの **Username** および **Password** は system および manager です。

4. Web サービスのサーバーとランタイムをセットアップします。
   a)「**ウィンドウ**」 >「**設定**」 >「**Web サービス**」 >「**サーバーおよびランタイム**」
   b) サーバーとして「**IBM WASCE v2.1 サーバー**」を選択します。
   c) Web サービス・ランタイムを「**Apache Axis**」のままにしておきます。

5. 動的 Web プロジェクトを作成します。
   a)「**ファイル**」 >「**新規**」 >「**動的 Web プロジェクト**」。
   プロジェクトに StockQuoteAxis という名前を付けます。
   b)「**EAR にプロジェクトを追加**」 >「**新規 ...**」 にチェック・マークを付けます。
   c)「**EAR アプリケーション・プロジェクト**」ページで、**Project name** StockQuoteAxisEAR >「**終了**」 を入力します。
   Java EE パースペクティブに切り替えることを示すダイアログ・ボックスに応答して **OK** と応答するか、Java パースペクティブをそのまま使用してこれらの指示に正確に従います。
   d) ターゲット・ランタイムとして「**IBM WASCE 2.1 サーバー**」が選択されています。 これを受け入れて、その他のデフォルトの 「**終了**」を選択します。
   Java EE パースペクティブに切り替えることを示すダイアログ・ボックスに応答して **OK** と応答するか、Java パースペクティブをそのまま使用してこれらの指示に正確に従います。

6. StockQuoteAxis.java サンプル・プログラムをインポートします。
   a) **StockQuoteAxis** Web プロジェクトを開きます。> **src** フォルダー>「**インポート ...**」 を右クリックします。
   b)「**一般**」 >「**ファイル・システム**」 >「**次へ**」 を選択します。
   c) *MQ_INSTALLATION_PATH*\tools\soap\samples\java\server > チェック **StockQuoteAxis.java** >「**終了**」 を参照します。
   MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。 サーバー・ディレクトリーを強調表示しないと、そこに含まれているファイルを確認できません。

7. StockQuoteAxis.java をその正しいパッケージに移動することにより、コンパイル・エラーを修正します。
   a) StockQuoteAxis.java を開き、問題>「**クイック・フィックス**」 を右クリックします。
   b)「**Move 'StockQuoteAxis.java' to package 'soap.server'** > **Save**」をダブルクリックします。

8. StockQuoteAxis.java から Web サービスを作成します。

a) StockQuoteAxis.java > 「**Web サービス**」 > 「**Web サービスの作成**」 > 「**次へ**」を右クリックします。

サービスに関する以下のデフォルト構成を受け入れます。

Web サービス・タイプ
ボトムアップ Java beanWeb サービス

サービス実装
soap.server.StockQuoteAxis

サーバー
IBM WASCE v2.1 server

Web サービス・ランタイム
Apache Axis

サービス・プロジェクト
StockQuoteAxis

サービス EAR プロジェクト
StockQuoteAxisEAR

構成
クライアント生成なし

9. アクセスする方法と Web サービスのスタイルを選択して、 「**次へ**」をクリックします。

プロンプトが出された場合はサーバーを開始します。

a) すべてのメソッドが選択されたままにしておきます。

b) 文書/リテラル (折り返し) スタイルを選択します。

10. **完了**

サービスがデプロイされたら、StockQuoteAxis Web プロジェクトの WebContent\wsdl フォルダーで、生成された StockQuoteAxis.wsdl ファイルを見つけます。

11. Web Services Explorer で HTTP を使用してサービスをテストします。

a) StockQuoteAxis.wsdl > 「**Web サービス・エクスプローラーでのテスト**」を右クリックします。

b) 「**Web Services Explorer**」ウィンドウで、「**StockQuoteAxisSoapBinding**」アクションのオペレーション「**getQuote**」をクリックします。

c) **symbol** 入力フィールド> **Go** に ibm を入力します。

12. WebSphere MQ transport for SOAP を使用してサービスをテストします。

サービスをデプロイするには、 com.ibm.mq.soap.jar にある **SimpleJavaListener** を使用します。 WebSphere MQ Java および SOAP ライブラリーをビルド・パスに追加する必要があります。

a) **StockQuoteAxis** Web プロジェクト> **ビルド・パス > ビルド・パスの構成 ...** を右クリックします。

b) 「**ライブラリー**」 タブ> 「**外部 JAR の追加 ...**」をクリックします。 *MQ_INSTALLATION_PATH*\java\lib を参照します。 すべての .jar ファイルを選択し、「**開く**」 > 「**外部 JAR の追加 ...**」を選択します。 *WMQ Install directory*\java\lib\soap を参照して、すべての .jar ファイルを選択し、> 「**開く**」 > 「**OK**」を選択します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされている上位ディレクトリーを表します。

c) プロジェクト・エクスプローラーで、 StockQuoteAxis\Java Resources\Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class\ SimpleJavaListener > 「**実行 ...**」 > 「**構成の実行 ...**」 を右クリックします。

ヒント :

SimpleJavaListener の構成が存在しない場合は、「**構成の実行**」ウィザードの「**構成の作成、管理、および実行**」ページにある「**新規構成**」アイコンをクリックします。

SimpleJavaListener を停止するコマンドはありません。 **SimpleJavaListener** をモニターまたは停止するには、Eclipse で「**デバッグ・パースペクティブ**」を開きます。

d)「**(x)= 引数**」タブを開きます。 「**プログラム実引数**」入力域に、**SimpleJavaListener** のパラメーターを入力します。

この例の場合は、次のように入力します。

```
-u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
```

注 : ターゲット・サービスは StockQuoteAxis で、サービス・デプロイメント記述子 StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd で作成されたターゲット・サービス名と一致します。 amqwdeployWMQService は、 soap.server.StockQuoteAxis という名前のターゲット・サービスを作成します。 この例では、HTTP サーバーと同じ StockQuoteAxis.class と service-config.wsdd を使用します。

e) 同じタブで、「**作業ディレクトリー**」を、server-config.wsdd ファイルを参照するように、次のように構成します。
${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}

a) **実行**

エラーはコンソールに書き込まれます。 コンソールがブランクのままであれば、**SimpleJavaListener** は正常に開始しました。

b) デプロイメントをテストするには、タスク 978 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-RPC クライアントを開発する』 で作成した StockQuoteAxis クライアントを実行します。

例: StockQuoteAxis サンプル・プログラム

サンプル Java Web サービス StockQuoteAxis.java は、
WMQ install directory\tools\soap\samples\java\server にインストールされています。
StockQuoteAxis.java、 969 ページの図 170 には、以下の 4 つのメソッドがあります。

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteTran(String symbol)

```
package soap.server;
import java.lang.Thread;
import java.io.FileWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }

    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,
                exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

図 *170. StockQuoteAxis*

次のタスク

コマンド **amqwdeployWMQService** を使用した HTTP の代わりに、WebSphere MQ Transport for SOAP を使用してサービスをデプロイします。

このコマンドには、Apache Axis 1.4 デプロイメント記述子を作成することによってサービスをデプロイするオプション axisDeploy があります。 WebSphere MQ SOAP リスナーがサービスを実行します。 SOAP リスナーは SimpleJavaListener という名前で、WebSphere MQ transport for SOAP に付属しています。

関連タスク

Microsoft Visual Studio 2008 を使用して WebSphere MQ transport for SOAP 用の .NET 1 サービスまたは .NET 2 サービスを開発する
Microsoft Visual Studio 2008 を使用して .NET 1 または .NET 2 用の SampleStockQuote Web サービスを開発します。

W3C SOAP over JMS 用の JAX-WS EJB Web サービスの開発
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、JEE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 2 番目のステップです。

Microsoft Visual Studio 2008 を使用して *WebSphere MQ transport for SOAP* 用の *.NET 1* サービスまたは *.NET 2* サービスを開発する

Microsoft Visual Studio 2008 を使用して .NET 1 または .NET 2 用の SampleStockQuote Web サービスを開発します。

このタスクについて

Visual Studio 2008 を使用して、分離コード実装の StockQuote サービスを作成します。

手順

1. サービスのテンプレートを作成し、HTTP 上で実行していることを確認します。
   a) Visual Studio 2008 を開始し、「**ファイル**」>「**新規**」>「**プロジェクト...**」をクリックします。「**C#**」プロジェクト・タイプ、「**.NET Framework 2**」、および「**ASP.NET Web サービス・アプリケーション**」を選択します。「**名前:**」および「**ソリューション名:**」に StockQuoteDotNet と入力し、「**OK**」をクリックします。
   b) **Solution Explorer** 内で「**Service1.asmx**」を右クリックして、「**名前変更**」>StockQuote.asmx をクリックします。
   c) コード・フラグメント public class Service1 を public class StockQuote に変更します。
   d) **Solution Explorer** 内で「**StockQuote.asmx**」を右クリックして、「**アプリケーションから開く...**」>「**XML エディター**」をクリックします。 Class="StockQuoteDotNet.Service1" を Class="StockQuoteDotNet.StockQuote" に変更します。
   e) コード・フラグメント [WebService(Namespace = "http://tempuri.org/")] を [WebService(Namespace = "http://stock.samples/")]に変更します。
   f) コードの行 [ToolboxItem(false)] を除去します。
   g)「**デバッグ**」>「**デバッグの開始 (F5)**」をクリックして、これまでの作業がすべて正しいことを確認します。 Explorer で出力を検査します。
2. サンプル SQDNNonInline.asmx.cs からメソッドを追加し、HTTP 上でサービスをテストします。
   a) *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs を開き、HelloWorld メソッドを 4 つの Quote メソッドに置き換えます。 971 ページの図 171 を参照してください。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーを表します。
   b)「**ビルド**」> ソリューションの「**再ビルド**」を選択し、エラー状態の「**スレッド**」 の 1 つを右クリックし、「**解決**」>「**System.Threading**」の使用を選択します。
   c) F5 を押してデバッグを開始します。
   サービスは WS-I Basic Profile v1.1 には準拠していません。 WebMethod 注釈を [SoapRpcMethod] から [SoapDocumentMethod] に変更するか、または注釈 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)] を除去するかを選択できます。
   d) F5 を押して、HTTP を使用する実装を検査します。
3. WSDL とクライアントを生成し、WebSphere MQ transport for SOAP を使用するサービスを実行します。
   a) プロジェクト・ディレクトリー・ツリー内の、StockQuote.asmx が格納されている場所でコマンド・ウィンドウを開きます。
   b) (オプション) amqwdeployWMQService を使用して成果物を生成します。 以下のように、キュー・マネージャーを開始しなければなりません。

```
amqswdeployWMQService -f StockQuote.asmx
-u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

   すべての成果物は ./generated ディレクトリー・ツリー内に作成されます。
   c) (オプション) WebSphere MQ transport for SOAP を使用するサービスを呼び出す WSDL のみを生成します。

```
amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
```

```
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

a) .NET リスナーを実行します。 .\generated\server\startWMQNListener.cmd を使用するか、次のコマンドを入力します。

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. WSDL から生成されたクライアントか、または **amqwdeployWMQService** によって生成されたクライアントを使用してサービスをテストします。

サンプル・コード

サンプル .NET Web サービス StockQuoteDotNet は、*MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet にインストールされています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。 公開されているサンプルの Web サービス・バインディングは、このタスクで使用されているバインディングとは多少異なります。 このタスクでは、Visual Studio 2008 で使用されているデフォルトを使用します。

.NET Framework 1 と .NET Framework 2 の Web サービスには、2 つの例があります。 StockQuoteDotNet.asmx、 インライン・サービスです。 SQDNNoninline.asmx、 SQDNNoninline.asmx.cs によって実装される分離コード Web サービスです。

StockQuoteDotNet には以下の 4 つのメソッドがあります。

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteDOC(String symbol)

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

図 *171.* インライン・サービス*: StockQuoteDotNet.asmx*

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

図 *172.* 分離コード*:* 設計*: SQDNNonInline.asmx*

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

図 *173.* 分離コード*:* 実装*: SQDNNonInline.asmx.cs*

関連タスク

WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する
WebSphere MQ をサービス・プロバイダーとして使用して実行する Axis 1.4 Web サービスを作成します。通常の Web サービス開発環境を使用して、Axis 1.4 へのデプロイメント用のサービスを作成します。

W3C SOAP over JMS 用の JAX-WS EJB Web サービスの開発
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、JEE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 2 番目のステップです。

W3C SOAP over JMS 用の *JAX-WS EJB Web* サービスの開発

SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、JEE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 2 番目のステップです。

始める前に

Rational Application Developer を使用して EJB Web サービスを作成します。 Rational Application Developer の Web サービス・ウィザードでは、オプションで、W3C 勧告候補 SOAP over JMS バインディングを使って Web サービスを作成できます。 Rational Application Developer 7.54 が必要です。 演習では、Rational Software Architect for WebSphere Software v7.5.5.1 に含まれる Rational Application Developer が使用されました。

EJB は、このタスクの一部として Rational Application Developer から WebSphere Application Server にデプロイされます。 1011 ページの『W3C SOAP over JMS を使用するための WebSphere Application Server の構成』を完了する必要があります。

タスクで実際に使われる WSDL を作成するには、965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』のタスクをまず完了する必要があります。 その後、Eclipse Galileo ワークスペースの動的 Web プロジェクトから、または WASCE にデプロイされた実行中の HTTP Web サービスから、WSDL をインポートできます。

1012 ページの『WebSphere Application Server リソースの構成』の結果として、WebSphere Application Server がまだ実行中である可能性があります。 そうでない場合、RAD のサーバー・ビューから開始できます。

このタスクについて

このタスクでは、WebSphere MQ transport for SOAP を使って **SimpleJavaListener** によって実行される JAX-RPC Axis サービスから、W3C SOAP over JMS プロトコルを使って WebSphere Application Server で実行される JAX-WS サービスに、`StockQuoteAxis` サービスを再デプロイします。

SimpleJavaListener から WebSphere Application Server へのサービスの移行は、以下の 2 つの部分から成ります。

1. Rational Application Developer でトップダウン方式の EJB Web サービス・ウィザードを使用して、サービスに関する WSDL から Web サービス・インターフェースを生成します。
2. WebSphere MQ SOAP サンプル `StockQuoteAxis.java` をインポートすることにより、サービスを実装します。

代わりの手法として、`StockQuoteAxis.java` からボトムアップ方式でサービスを生成することもできます。 ただし、移行後のサービスのインターフェースを確実に同じにするには、同じ WSDL を使用するトップダウン方式がより適切です。

EJB コンテナーには JMS サポートが含まれるため、Web コンテナーではなく EJB コンテナー用に Web サービスが開発されます。

手順

1. Rational Application Developer を開始して、WebSphere Application Server が実行中であることを確認します。

   a) 新しいワークスペースで Rational Application Developer を開始します。

   b) Java EE パースペクティブを開きます。

   c)「**サーバー**」タブを開いて、WebSphere Application Server が稼働していることを確認します。

   - ビューに WebSphere Application Server v7.0 がない場合は、ビュー>「**新規**」 >「**サーバー**」を右クリックします。 ウィザードの選択項目に従って、WebSphere Application Server v7.0 インスタンスを作成します。
   - サーバーが表示されていても開始されていない場合は、矢印をクリックしてサーバーを開始します。
   - プロパティーを確認し、サーバー・ログに素早くアクセスするには、 **WebSphere Application Server v7.0 at localhost** >「**プロパティー**」> **WebSphere Application Server** を右クリックします。

- サーバーを管理するには、外部ブラウザーを使用して URL `http://localhost:9061/ibm/console/unsecureLogon.jsp` を開くか、 **WebSphere Application Server v7.0 at localhost** > 「**管理コンソールの実行**」を右クリックします。
- デフォルト設定は自動公開です。 多くの場合、サーバーの更新を手動でデプロイする操作が好まれます。 「**WebSphere Application Server v7.0 (localhost)**」をダブルクリックして、「**概要**」ウィンドウの「**公開**」三角アイコンを展開します。 「**自動公開しない**」をクリックします。
- 変更する必要のある可能性のあるもう 1 つのデフォルトは、「**概要**」ウィンドウの「**ワークベンチのシャットダウン時にサーバーを終了する**」チェック・ボックスです。これをクリアします。

2. JEE プロジェクトを作成します。

   エンタープライズ・アプリケーション・プロジェクト (EAR) およびエンタープライズ Java Bean (EJB) プロジェクトを作成する必要があります。

   a)「**ファイル**」>「**新規**」>「**エンタープライズ・アプリケーション・プロジェクト**」を選択します。 プロジェクトに `W3CJMSEAR` >「**終了**」という名前を付けます。

   デフォルトでは、ターゲット・ランタイムとして WebSphere Application Server v7.0、および EAR バージョン 5.0 を指定する必要があります。 デフォルト構成を選択する必要があります。

   b)「**ファイル**」>「**新規**」>「**EJB プロジェクト**」を選択します。 プロジェクトに `W3CJMSEJB` という名前を付けます。 「**EAR プロジェクト名**」 >「**次へ**」として `W3CEARJMS` を選択します。

   デフォルト EJB モジュール・バージョンは 3.0 で、デフォルト構成が再び使用されます。

   c)「**EJB クライアント JAR モジュールの作成**」 チェック・ボックスをクリアし、>「**終了**」をクリアします。

3. `StockQuoteAxis` WSDL から EJB Web サービスを生成してデプロイします。

   a)「**実行**」>「**Web サービス・エクスプローラーの起動**」を選択します。

   b)「**Web サービス・エクスプローラー**」 ウィンドウのアイコンを使用して WSDL ページを選択し、Navigator で 「**WSDL メイン**」 をクリックします。

   c)「**アクション**」ウィンドウで、`StockQuoteAxis.wsdl` の WSDL URL を入力または参照します。

   HTTP サービスとしてデプロイされた `StockQuoteAxis` が実行される WASCE の場合、URL は次のとおりです。

   ```
   http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
   ```

   ファイル・システム内の WSDL の場合、URL は例えば次のようになります。

   ```
   File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
   ```

   d) ナビゲーター・ツリーで、インポートされた URL を含む行をクリックします。

   これが Web サービス・エクスプローラーにインポートされた最初の WSDL である場合、「**WSDL メイン**」の直後の行がこれに該当します。

   e)「**アクション**」ウィンドウで、「**Web サービス起動ウィザード**」>「**Web サービス・スケルトン**」>「**実行**」をクリックします。

   f) Web サービス・ウィザードで、「**トップダウン EJB Web サービス**」を選択します。

   974 ページの表 141 の情報を使って構成を選択または確認します。「**警告なしにファイルを上書き**」にチェック・マークを付けて「**次へ**」をクリックします。

表 *141.* トップダウン *EJB Web* サービスの構成

フィールド	**値**
サーバー	WebSphere Application Server v7.0
Web サービス・ランタイム	IBM WebSphere JAX-WS
サービス・プロジェクト	W3CJMSEJB

表 *141.* トップダウン *EJB Web* サービスの構成 *(*続き*)*	
フィールド	**値**
サービス EAR プロジェクト	W3CJMSEAR
構成:	No client generation

g)「**WebSphere JAX-WS EJB トップダウン Web サービス作成のオプションを指定してください**」というサブタイトルが付いたページで、「**JMS バインディングへの切り替え**」ボックスにチェック・マークを付けます。 また、「**ラッパー・スタイルを使用可能にする**」、「**WSDL をプロジェクトにコピー**」、および「**Web サービス・デプロイメント記述子の生成**」 > 「**次へ**」にチェック・マークを付けます。

h)「**WebSphere JAX-WS JMS バインディング構成**」というタイトルのページで、「**SOAP/JMS インターオペラビリティー・プロトコルの使用**」 にチェック・マークを付け、 975 ページの表 142 からの値を指定します。その他のフィールドはブランクのまま 「**次へ**」にします。

表 *142. WebSphere JAX-WS JMS* バインディング構成	
フィールド	**値**
JMS 宛先	queue
宛先 JNDI 名:	requestaxis
JMS 接続ファクトリー	qm1
応答先の名前	W3CJMSEAR
構成:	replyaxis

a)「**WebSphere JAX-WS ルーター・プロジェクト構成**」というタイトルのページで、「**ActivationSpec JNDI 名**」フィールドに qm1as と入力して、「**次へ**」をクリックします。

RAD がプロジェクトを生成してデプロイするまでに約 30 秒から 1 分ほどかかります。

b)「**Web サービスの公開**」 ページのオプションを無視して、 「**終了**」をクリックします。

4. 生成された WSDL を確認します。

サービス固有の WSDL を生成してプロジェクト内に保存するよう要求しました。

a) エンタープライズ・エクスプローラー・ナビゲーターで、フォルダー **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl** を開きます。 StockQuoteAxis.wsdl をダブルクリックして、WSDL エディターでこれを開きます。

バインディングを検査します。次のような JMS url です。

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. オプションのステップ: JAX-WS を使用した SOAP over HTTP への EJB のバインドを行います。

EJB へ 2 つのバインディングを提供することで、クライアントは Web サービスを呼び出すために SOAP バインディングを選択できます。 また、これによりクライアントは HTTP を使用して、WSDL を獲得するために Web サーバーに照会することができます。

EJB を SOAP over HTTP にバインドするステップは、このタスクには含まれていません。

6. サンプル StockQuoteAxis.java を使用して StockQuoteAxis を実装し、再デプロイします。

a) エンタープライズ・エクスプローラー・ナビゲーターで、フォルダー **W3CJMSEJB** > 「**サービス**」「ダブルクリック」 StockQuoteAxisService を開き、Java エディターで実装クラスを開きます。

b) *WebSphere MQ Installation directory*\tools\soap\samples\java\server フォルダー内の StockQuoteAxis.java サンプル・プログラムを開き、すべてのメソッドを選択します。ただし、クラス名> **Copy** は選択しません。

c) StockQuoteAxisSoapBindingImpl.java で、すべてのメソッドを選択します (ただしクラス名を除く)。StockQuoteAxis.java からメソッドに貼り付けます。

d) サービスが呼び出されたときに WebSphere Application Server コンソールに出力するための print ステートメントを追加します。
getQuote(String symbol) メソッドを次のように変更します。

```
public float getQuote(String symbol) {
    System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
                       + symbol);
    return ((float) 55.25);
```

e) インポートを修正します (「**ソース**」>「**インポートの編成**」>「**保存**」をクリックします)。

f) 実装がインターフェースにマッチしないことが原因の 3 つのエラーを修正します。

エラーの原因は、StockQuoteAxis.java のメソッドのうち 3 つが例外をスローし、サービスの WSDL に障害メッセージが含まれないことです。 この問題はメソッド・シグニチャーとメソッド Web サービス・アノテーションの間の不一致と診断されます。

@WebFault を使ってメソッドにアノテーションを付けて WSDL を再生成するか、あるいはインターフェースを変更せずに、例外を除去してください。

同じインターフェースを保持するには、3 つの throws exception をメソッド・シグニチャーから除去して「**保存**」をクリックします。

次のタスク

1022 ページの『W3C SOAP over JMS を使用した Axis2 クライアントへのデプロイ』

関連タスク

WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する
WebSphere MQ をサービス・プロバイダーとして使用して実行する Axis 1.4 Web サービスを作成します。通常の Web サービス開発環境を使用して、Axis 1.4 へのデプロイメント用のサービスを作成します。

Microsoft Visual Studio 2008 を使用して WebSphere MQ transport for SOAP 用の .NET 1 サービスまたは .NET 2 サービスを開発する
Microsoft Visual Studio 2008 を使用して .NET 1 または .NET 2 用の SampleStockQuote Web サービスを開発します。

WebSphere MQ transport for SOAP 対応 WebSphere MQ Web サービス・クライアントの作成

通常の開発環境を使用して、WebSphere MQ transport for SOAP に対応した Web サービス・クライアントを作成します。

始める前に

サービスを作成します。 964 ページの『WebSphere MQ transport for SOAP 対応 Web サービスの作成』で示す例の中の 1 つを使用できます。

クライアントをどのように作成、デプロイ、使用するか、およびクライアント生成のための WSDL をどこで取得するかについて、選択を行います。

WebSphere MQ transport for SOAP 対応のクライアントおよびサービスの作成アプローチを決定します。

2 とおりのアプローチがあります。

1. 標準的な開発ツールを使用して HTTP サービスおよびクライアントを作成し、WebSphere MQ transport for SOAP の URL を使用する。
2. WebSphere MQ transport for SOAP に付属のツールとサンプルを使用する。

HTTP のアプローチを取る場合は、サービスを HTTP サーバー上で実行できるほか、WebSphere MQ transport for SOAP を使用してサービスを実行できます。 WebSphere MQ transport for SOAP を使用してサービスを実行するには、適切な SOAP 用 WebSphere MQ リスナーを構成し、サービスを実行するためのパスとデプロイメント記述子をセットアップします。 WebSphere MQ transport for SOAP が

提供するツールは、ユーザーに代わって構成を行います。 あるいは、ユーザーがリスナーの実行環境を構成することもできます。

WebSphere MQ transport for SOAP に付属のツールは、入門用として、またトランスポートのデプロイ方法を習得するのに役立ちます。 実動作業上は、標準的なツールを使用することと、さまざまな SOAP トランスポートがアクセス可能な同一サービスをデプロイすることに利点があります。

作成するクライアントのタイプを決定します。

どのようなタイプの Web サービス・クライアントを作成するかを決定する必要があります。 この選択は、サービス・インターフェースとサービスのアドレスが分かっているかどうかによって決まります。

インターフェースが分かっている場合は、Axis または .NET のツールを使用して、サービス・インターフェースからプロキシー・クライアント・クラスを生成します。 プロキシー・クライアント・クラスを使用することで、サービスを呼び出すクライアントが作成しやすくなります。 クライアントを作成する時点でサービスの場所が分かっている場合は、静的プロキシー・インターフェースを使用します。 サービスの場所が変更される場合、例えば実動サーバーにサービスが再デプロイされる場合は、動的プロキシー・インターフェースを使用します。

Axis の場合、クライアントを作成する時点でサービス・インターフェースが分かっていなければ、Axis 1.4 用の動的起動インターフェース (DII) クライアントを作成できます。 DII クライアントは、どのサービスを呼び出すのにも総称インターフェースを使用します。 特定のサービスにパラメーターを正しく渡すには、その具体的なサービス・インターフェースをプログラムで作成する必要があります。 インターフェースは、クライアントでプログラムで作成するか、サービスの WSDL をクライアントにロードすることによって作成します。 Axis2 の場合は、ディスパッチ・クライアントを作成できます。 DII クライアントが呼び出しモデルを使用するのに対し、ディスパッチ・クライアントは、文書モデルを使用してクライアント要求を記述します。 両方とも、要求を動的に作成するのに効果があります。

サービスの WSDL を取得します。

プログラムで作成されるサービス・インターフェースの場合以外は、Web サービス・クライアントを作成するためには、まずサービス WSDL を取得する必要があります。 サービス WSDL は、以下の 3 つの異なるソースから取得できます。

1. **java2wsdl** (Axis) や **disco** (.NET) などのツールを使用して、Web サービス実装から直接取得する。
2. URL *Web service http url*?wsdl を使用して Web サービスを照会する。
3. ファイル・システム上のファイルか、または UDDI や WebSphere Service Registry and Repository などのレジストリーにあるファイルから取得する。

注 : HTTP を使用してサービスにアクセスできなければ、WSDL 照会は機能しません。 サービス自体は、WebSphere MQ transport for SOAP を使用する場合のみ使用できる可能性があります。

amqwdeployWMQService によって生成される WSDL は、**java2wsdl** または **disco** を使用して生成される WSDL と同じではありません。 生成される WSDL は、サービス「Top Down」を作成するために WSDL から始めた場合でも、その WSDL とも異なります。 Axis の場合、`server-config.wsdd` デプロイメント記述子は、クライアントによって作成された SOAP メッセージを操作とサービスにマップします。 **amqwdeployWMQService** は、Eclipse から異なるデプロイメント記述子を生成します。

クライアントを作成するために使用する WSDL は、サービスがどのようにデプロイされるかによって、次のように異なります。

amqwdeployWMQService を使用してデプロイ
amqwdeployWMQService によって生成される WSDL を使用します。 -w フラグを指定し、`rpcLiteral` WSDL を選択します。 互換性を維持するために、`rpcEncoded` WSDL を選択することもできます。 `rpcEncoded` WSDL は、.NET および Axis 1.4 クライアントのみと連係します。

SimpleJavaListener を使用した手動デプロイメント
以下の WSDL ファイルのいずれかを使用します。

1. サービスを定義するために使用される、つまりリポジトリーに保管される WSDL。
2. **java2wsdl** によってサービスから生成される WSDL。

3. URL *Web service http url* ?wsdl を使用して照会された WSDL (HTTP サーバーから使用可能な場合)。 Web Services Explorer などのツールを実行して、サービス定義を Eclipse に直接インポートすることもできます。

サービスの URI を変更しなければならない場合もあります。 HTTP サービスのアドレスから、WebSphere MQ transport for SOAP の URI に変更します。

amqSOAPNETListener を使用した手動デプロイメント
以下の WSDL ファイルのいずれかを使用します。

1. サービスを定義するために使用される、つまりリポジトリーに保管される WSDL。
2. .NET サービス・クラス (.asmx) から取得される WSDL。 **disco** を使用する。
3. URL *Web service http url* ?wsdl を使用して照会された WSDL (使用可能な場合)。 Web Services Explorer などのツールを実行して、サービス定義を Eclipse に直接インポートすることもできます。
4. .NET サービス・クラス (.asmx) に対して **amqswsdl** を実行することによって取得した WSDL。

サービスの URI を変更しなければならない場合もあります。 HTTP サービスのアドレスから、WebSphere MQ transport for SOAP の URI に変更します。

Windows Communication Foundation にデプロイ
URL *Web service http url*?wsdl を使用してサービス WSDL を取得します。 サービスには、サービス定義の一部として serviceMetaData 動作構成が定義されていなければなりません。

異なるサーバー・プラットフォームへのデプロイメント
正しいサービス WSDL を取得する方法について、プラットフォームに付属のガイダンスに従ってください。

このタスクについて
標準的な開発ツールを使用してクライアントを作成します。 以下のタスクでは、.NET 1 および 2、Axis 1.4 (JAX-RPC) および Axis2 (JAX-WS) 用のクライアントを作成する方法を分かりやすく説明しています。 Windows Communication Foundation については、関連タスクのリンクを参照してください。

Eclipse を使用して *WebSphere transport for SOAP* 用の *JAX-RPC* クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように Axis 1.4 Web サービス・クライアントを開発します。

始める前に
サービスを使用可能にしなければなりません。 練習問題としてこのタスクに従う場合は、965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』のタスクで作成したワークスペースとサービスを使用してください。 Axis 1.4 Web サービスをサポートする Eclipse でアプリケーション・サーバーを実行する必要があります。 このタスクでは、無料で入手できる WebSphere Application Server Community Edition バージョン 2.1.4 を使用します。 この製品は、965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』のタスクの一部として構成されています。 Tomcat 6 を使用することもできます。このオープン・ソース・アプリケーション・サーバーの方が小規模です。

このタスクについて

このタスクは、Windows 上で稼働する Eclipse を使用した、サンプルの StockQuoteAxis サービスに関する 3 種類のクライアントの開発を示します。 該当するクライアントは、クライアント・プロキシーを使用して開発する静的クライアントと動的クライアント、および DII クライアントです。

WSDL からクライアント・プロキシーを生成する以下の 2 つの方法を示します。このどちらかを使用します。

1. **amqwdeployWMQService** を使用してクライアント・プロキシーを生成します。

2. WSDL を Eclipse 内にインポートし、Web サービス・ウィザードを使用してクライアント・プロキシーを生成します。

手順

1. Java EE 開発者用の Eclipse IDE を開始します。
2. StockQuoteAxisClient という名前の Java プロジェクトを作成します。
   a) Java パースペクティブ>「**ファイル**」>「**新規**」>「**Java プロジェクト**」に切り替えます。「**Java プロジェクトの作成**」**ページ** の **Project name** フィールドに、StockQuoteAxisEclipseClient と入力します。 実行環境が **J2SE1-1.4** または **J2SE-1.5** >「**次へ**」 のいずれかであることを確認してください。
   b)「**Java 設定**」ページで、「**ライブラリー**」タブ >「**外部 JAR の追加...**」
   c) *MQ_INSTALLATION_PATH*/java/lib を参照して、すべての .jar ファイルを選択し、「**開く**」をクリックします。
   MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされているディレクトリーです。
   d) *MQ_INSTALLATION_PATH*/java/lib/soap を参照して、すべての .jar ファイルを選択し、「**開く**」をクリックします。 WebSphere MQ インストール・メディアからこのディレクトリー内に axis.jar をインストールしていなければなりません。
   MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされているディレクトリーです。
   e)「**ライブラリー**」 タブは、クライアントのビルドに必要なすべての .jar ファイルを参照し、「**終了**」をクリックします。
3. Eclipse でサンプル StockQuoteAxis Web サービスに関するプロキシーを作成する以下の 2 つの方法のうち 1 つに従います。
   - **amqwdeployWMQService** を使用してクライアント・プロキシーを生成します。
     a. キュー・マネージャーを作成します。 このタスクの場合、デフォルトのキュー・マネージャーとして QM1 を作成します。
     b. 作業ディレクトリー samples を作成します。 StockQuoteAxis.java サンプル・プログラムを samples/soap/server にコピーします。
     c. *MQ_INSTALLATION_PATH*/bin の amqwsetcp.cmd を変更して、現行ディレクトリーをクラスパスに含めます。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。
     d. samples 内でコマンド・ウィンドウを開き、変更した **amqwsetcp** コマンドを実行します。
     e. 以下のコマンドを実行して、StockQuoteAxis サービスに関する WSDL を作成します。

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

     要確認 : Java コマンドを使用する場合は、 "." または "\" ではなく、 "/"を使用してください。

     ヒント : 生成済みのプロキシーを Eclipse 内にインポートする代わりに、.samples/generated から生成済みの WSDL をインポートできます。 生成結果のプロキシーには、以下の 2 点の違いがあります。

     i) パッケージ名が違います (リファクタリングできます)。
     ii) Eclipse で生成されたプロキシーには、追加のヘルパー・クラス StockQuoteAxisProxy.java が含まれています。

     f. 以下のコマンドを実行して、StockQuoteAxis サービスに関するクライアント・プロキシーを作成します。

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
-u "jms:/queue?destination=REQUESTAXIS
```

```
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

g. 以下のように、クライアント・プロキシーを StockQuoteAxisClient 内にインポートします。

i) **StockQuoteAxisClient¥ src** > 「**ファイル・システム**」 > 「**次へ**」 > 「**参照 ...**」 > フォルダー .\samples\generated\client\remote\soap\server > 「**OK**」を右クリックします。

ii)「**インポート**」 ページで 「**サーバー**」 にチェック・マークを付け、 「**終了**」をクリックします。

h. パッケージ名を soap.server にリファクタリングします。

i) クライアント・プロキシーを含むパッケージを右クリックして、「**リファクタリング**」>「**名前変更**」をクリックします。 **New name**: soap.server > と入力します。その他の選択項目については、選択したデフォルトのままにします> 「**OK**」 。 すべてのエラーが修正されます。

- Eclipse を使用してクライアント・プロキシーを生成します。

サービスに関する WSDL を入手する方法は、選択できます。 この例では、サービスは WebSphere Application Server Community Edition にデプロイされており、Web サーバーから WSDL を入手します。 このデプロイメントについては、965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』のタスクで説明します。

a. Eclipse で、Web パースペクティブに切り替え、WebSphere Application Server Community Edition v2.1 Server が実行中で StockQuoteAxis がデプロイ済みで同期済みであることを確認します。

b. 以下のように、WSDL を Web Services Explorer 内にインポートします。

i) アクション・バーで、「**Web Services Explorer**」アイコンをクリックするか、「**実行**」>「**Web Services Explorer の起動**」をクリックします。

ii) Web Services Explorer の WSDL ページ・アイコンをクリックして、WSDL ページに切り替えます。

iii) Web Services Explorer の「ナビゲーター」ウィンドウで「**WSDL のメイン**」をクリックします。

iv) Web サービスの URL の後に ?WSDL を付けて入力します。 965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』のタスクでデプロイした StockQuoteAxis の URL は以下のとおりです。

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

c. 以下のように、クライアント・プロキシーを生成します。

i) Web Services Explorer のナビゲーターで、「**http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl**」をクリックします。

ii)「**アクション**」ウィンドウで、「**Web サービス起動ウィザード**」をクリックし、「**Web サービス・クライアント**」を選択した状態のまま、「**進む**」をクリックします。

iii) ウィザードの最初のページで、構成内の 「**クライアント**」 プロジェクト・リンクをクリックし、 **StockQuoteAxisClient** クライアント・プロジェクト> 「**OK**」を選択します。

ヒント: このウィザードのウィンドウからフォーカスがなくなる場合があります。 手動でフォーカスを戻す必要があります。

iv) JAX-RPC クライアントを生成するには、Web サービス・ランタイムが Apache Axis でなければなりません。

v)「**完了**」をクリックします。

vi) サービスの静的 URL を、StockQuoteAxis サービスに関する WebSphere MQ transport for SOAP アドレスを指すように変更します。 HTTP サーバーを使用してクライアントをテストし終えるまでは、このステップのスキップを選択することもできます。

a) StockQuoteAxisServiceLocator.java を開き、StockQuoteAxis_address に関する宣言を見つけます。

b) URL を以下のように変更します。

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

ヒント：Eclipse では、自動的に & が & に変換され、.java コード内にストリングをコピー・アンド・ペーストする際にはその逆に変換されます。

d. それぞれが main メソッドを持つ 3 つの Java クライアント・クラスを作成します。

i) パッケージを作成します。「**StockQuoteAxisClient/src**」を右クリックして、「**新規パッケージ**」をクリックします。 soap.client という名前を付けて、「**終了**」をクリックします。

ii)「**soap.client**」>「**新規**」>「**クラス**」を選択します。 クラスに SQAStaticClient という名前を付け、「**public static void main(string [] args)**」にチェック・マークを付けて、「終了」をクリックします。

iii) 手順を繰り返して、SQADynamicClient.java と SQADIIClient.java を作成します。

e. クライアント・コードを作成します。

985 ページの図 177 から 986 ページの図 181 には、クライアント・コードの 3 つのスタイルの例が示されています。 これらの例は、HTTP URL を使用し、HTTP サーバーにデプロイされた StockQuoteAxis サービスを使ってクライアントをテストします。 WebSphere MQ transport for SOAP を使用してデプロイされた StockQuoteAxis サービスに対してクライアントを実行するには、URL を以下のように変更します。

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

– 985 ページの図 177 と 985 ページの図 179 では、Eclipse によって生成されたプロキシーを使用します。Eclipse には、コーディングをいくらか簡単にする特別な StockQuoteAxisproxy ヘルパー・クラスがあります。

– 985 ページの図 178 と 986 ページの図 180 では、**amqwdeployWMQService** によって生成されたプロキシーを使用します。

– 986 ページの図 181 では、プロキシー・クラスを使用しません。

各クライアントが com.ibm.mq.soap.Register.extension() を呼び出して、WebSphere MQ transport for SOAP にリンクします。 拡張は、クライアント・デプロイメント記述子に登録されます。 Axis 1.4 へのクライアント・デプロイメントについては、 1017 ページの『IBM WebSphere MQ transport for SOAP を使用した Web サービス・クライアントの Axis 1.4 へのデプロイ』で説明されています。

f. ワークスペース内で構成された WebSphere Application Server Community Edition サーバーによってホストされる StockQuoteAxis に SOAP 要求を送信して、クライアントを実行します。

i) サーバーが実行中で、StockQuoteAxis がデプロイ済みで同期済みであることを確認します。

ii) テストするクライアントを選択するか開き、アクション・バーで「**実行**」をクリックします。あるいは、緑色の「実行」アイコンをクリックするか、ナビゲーターでクライアントを 8 つクリックして、>「**実行**」 >「**実行構成 ...**」をクリックします。 クライアントの実行に必要なパラメーターを構成します。

g. WebSphere MQ transport for SOAP を使用してクライアントを実行します。

この手順では、 **amqwdeployWMQService** を使用してサービスをデプロイします。この手順は、**amqwdeployWMQService** によって作成された WSDL またはプロキシーを使用するクライアントでのみ機能します。 元の WSDL か Eclipse によってビルドされたプロキシーを使用してクライアントを実行するには、Eclipse によってビルドされたデプロイメント記述子を使用してサービスをデプロイします。 targetServiceName としてサービス・ポート・バインディング名を使用して、**SimpleJavaListener** を手動で開始します。

i) 1005 ページの『サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する』 の指示に従って、サービスを WebSphere MQ

Simple Java SOAP リスナーにデプロイします。 サービスのデプロイメントは、WSDL を使用するクライアントか、**amqwdeployWMQService** によってビルドされたクライアント・プロキシーの場合のみ実行できます。

ii) コマンド・ウィンドウで、**amqwclientconfig** を実行して、クライアント・デプロイメント記述子ファイル client-deploy.wsdd を作成します。

iii) WebSphere MQ transport for SOAP を使用して、テストする Java プロジェクトのルートに client-deploy.wsdd をインポートします。

a) Java プロジェクト **StockQuoteAxisEclipseClient** > 「**インポート**」 > 「**ファイル・システム**」 > 「**次へ**」 > 「**参照 ...**」 を右クリックします。

b) 右側のペインで、 client-deploy.wsdd が含まれているディレクトリーを参照して、「**開く**」 > 「**インポート**」 ウィザード・ページでディレクトリーを選択し、 client-deploy.wsdd にチェック・マークを付けます。

c)「**宛先フォルダー:**」に StockQuoteAxisEclipseClient が入力されていることを確認し、「**終了**」をクリックします。

iv) このプロジェクトで Java アプリケーションを実行するための作業ディレクトリーが StockQuoteAxisEclipseClient ディレクトリーであることを確認します。

Java プロジェクト **StockQuoteAxisEclipseClient** > **Run as ....** > 「**実行構成 ...**」 > 「**(x) = 引数**」 タブを選択し、「作業ディレクトリー」で 「**デフォルト**」 ラジオ・ボタンにチェック・マークが付いており、パスが StockQuoteAxisEclipseClient であることを確認します。 別の方法として、以下のいずれかを選択し、クライアント構成を含む別の場所またはファイルを選択します。

– 「**その他:**」にチェック・マークを付け、選択したディレクトリー・パスを入力します。

– 「**VM 引数**」 ウィンドウで、 -Daxis.ClientConfigFile=*full path to client deployment descriptor file* と入力します。

v) WebSphere MQ transport for SOAP を使用してデプロイされたサービスを指すように URL が構成されていることを確認します。 ステップ ii で説明されているようにクライアントを実行します。

ヒント : 通常、以下のいずかのエラーが発生することがあります。

i) `Exception: No client transport named 'jms' found!.`

ii) JMS 接続エラーです。

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

説明:

i) client-config.wsdd が見つからないか、client-config.wsdd に行 <transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/> が組み込まれています。

ii) ビルド・パスに問題がある可能性があります。.jar ファイルが *MQ_INSTALLATION_PATH*/java/lib に含まれていません。 *MQ_INSTALLATION_PATH* は、WebSphere MQ がインストールされているディレクトリーです。

iii) server-config-wsdd か、**SimpleSoapListener** に渡されるパラメーターに関する、サービス・デプロイメントの問題です。

iv) デプロイメント記述子とサービスの実装の間に不一致があります。

Eclipse 内でクライアントを実行することが難しい場合は、コマンド・ウィンドウを使用して以下のように試行します。

i) ワークスペース・ディレクトリー・ツリー内の StockQuoteAxisEclipseClient\bin ディレクトリーに切り替えます。

ii) **amqwsetcp** と **amqwclientconfig** を実行します。

iii) java soap/client/SQAStaticClient を実行します。

サンプル JAX-RPC Web サービス・クライアント

WebSphere MQ に付属のサンプル Java Web サービス・クライアントは、 *MQ_INSTALLATION_PATH* \tools\soap\samples\java\clients にインストールされています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。

SQAxis2Axis.java

SQAxis2Axis.java (983 ページの図 174) は、StockQuoteAxis サービスを呼び出す動的プロキシー・クライアントです。 動的プロキシーにコンパイルされるサービスの URL は、コマンド行に別の URL を指定することによって指定変更できます。

SQAxis2DotNet.java

SQAxis2DotNet.java (984 ページの図 175) は、StockQuoteDotNet サービスを呼び出す動的プロキシー・クライアントです。 動的プロキシーにコンパイルされるサービスの URL は、コマンド行に別の URL を指定することによって指定変更できます。

WsdlClient.java

WsdlClient.java、 984 ページの図 176 は、 StockQuoteDotNet サービスまたは StockQuoteAxis サービスのいずれかを呼び出すための動的呼び出しクライアントです。 クライアントは、デフォルトで StockQuoteAxis サービスを呼び出します。 コマンド行オプション -D を追加して StockQuoteDotNet サービスを呼び出し、-w を追加して .\generated\StockQuoteDotNet_Wmq.wsdl のポートとは異なるポートを指定します。

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                            e.printStackTrace();
         System.exit(2);
      }
   }
```

図 *174. SQAxis2Axis.java*

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
}
```

図 *175. SQAxis2DotNet.java*

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("  '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

図 *176. WsdlClient.java*

このタスクで使用されているサンプル・クライアントは以下のとおりです。

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

図 *177. Eclipse* 生成プロキシーを使用する静的クライアント

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

図 *178. amqwdeployWMQService* 生成プロキシーを使用する静的クライアント

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

図 *179. Eclipse* 生成プロキシーを使用する動的クライアント

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

図 *180. amqwdeployWMQService* 生成プロキシーを使用する動的クライアント

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

図 *181. DII* クライアント *(*プロキシーなし*)*

関連タスク

Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように Axis2 Web サービス・クライアントを開発します。 WebSphere MQ transport for SOAP で提供されるサンプル Axis2 クライアントをリストし、**wsimport** コマンドを使用してプロキシーを生成します。

Microsoft Visual Studio 2008 を使用して WebSphere transport for SOAP 用の .NET 1 または .NET 2 クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように .NET 1 または .NET 2 Web サービス・クライアントを開発します。

Eclipse を使用して *WebSphere transport for SOAP* 用の *JAX-WS* クライアントを開発する

WebSphere MQ transport for SOAP を使用して実行するように Axis2 Web サービス・クライアントを開発します。 WebSphere MQ transport for SOAP で提供されるサンプル Axis2 クライアントをリストし、**wsimport** コマンドを使用してプロキシーを生成します。

始める前に

Axis2 ライブラリーを入手し、クライアントを実行するように開発およびテスト環境を構成してください。

注 : Axis で使用されるバージョンとリリースの名前は混乱を招きます。 通常、Axis 1.4 は JAX-RPC の実装を指し、Axis2 は JAX-WS の実装を指します。

Axis 1.4 はバージョン・レベルです。 インターネットで Axis 1.4 を検索すると、http://ws.apache.org/axis/ が検出されます。 このページには、旧バージョンの Axis (1.2、1.3) と、2006 年 4 月 22 日付けの最終リリースの Axis 1.4 のリストが含まれています。 その後のリリースの Axis 1.4 があり、バグが修正されていますが、 これらはすべて Axis 1.4 と呼ばれています。 WebSphere MQ に付属しているのは、これらのバグ修正リリースの 1 つです。 Axis 1.4 の場合、http://ws.apache.org/axis/から取得できるバージョンではなく、WebSphere MQ に付属する `axis.jar` のバージョンを使用します。

Axis の Web サイトでは Axis 1.1 にも言及していますが、これは通常 Axis 1.4 と呼ばれることの方が多いすべてのバージョンを指しています。 Axis 1.2 は、通常 Axis2 と呼ばれるバージョンを指して使用されています。

Axis 1.5 は Axis 1.4 の後のリリースではなく、Axis2 リリースの 1 つです。 Axis 1.5 を検索すると、http://ws.apache.org/axis2/ が検出されます。https://ws.apache.org/axis2/download.cgi 0.9 から 1.5.1 までのラベルが付けられた Axis2 のリリース・バージョン (紛らわしいバージョン 1.4 を含む) のリストが入ります。 WebSphere MQ transport for SOAP と併用する Axis2 のリリース・バージョンは 1.4.1 です。 http://ws.apache.org/axis2/download/1_4_1/download.cgi から Axis2 1.4.1 をダウンロードしてください。

wsimport か、または IDE で提供されているツールのどちらかを使用して、WebSphere MQ transport for SOAP 用の Web サービス・クライアントのプロキシーを生成することを選択できます。 Eclipse IDE for Java EE Developer 3.5 SR1 は **wsdl2java** を使用します。**wsimport** は Java 6 で提供されています。 Java 5 を使用して、 **wsimport** または **wsdl2java** で生成されたクライアント・プロキシーを実行できます。

WebSphere MQ transport for SOAP で提供されるサンプル Web サービス Axis2 クライアントは、**wsimport** を使用して開発されました。992 ページの『サンプル Axis2 クライアント』を参照してください。

以下のタスクは、 Eclipse IDE for Java EE Developers にパッケージされている Web サービス・ウィザードによって生成されるプロキシーを生成して使用する方法を示しています。 サンプル・クライアントは、**wsimport** によって作成されるプロキシーの使用法を示します。

Web サービス・ウィザードを使用するには、Axis2 をサポートするアプリケーション・サーバーをワークベンチに追加しなければなりません。 以下のステップは、ワークベンチを使用して Axis2 をサポートするように WASCE を構成する方法を示しています。

1. Eclipse IDE for Java EE Developers で使用されるアプリケーション・サーバーを、 Axis2 をサポートするように構成します。 この例では、WASCE 2.1.4 アプリケーション・サーバーを構成します。これは、965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』で作成したワークスペースの一部です。
   a. サーバーを構成するワークスペース設定を開きます。「**ウィンドウ**」を開いて、「**設定**」をクリックします。
   b. インストール済みの JRE が `Java50` であることを確認します。「**インストール済み JRE**」をクリックします。
   c. サーバーとして WASCE を追加します。 「**サーバー**」 > 「**ランタイム環境**」 > 「**追加 ...**」 > **IBM** > **WASCE v2.1** >「次へ」をクリックします。 JRE は Java50 > WASCE インストール・ディレクトリー > **OK** > 「**終了**」でなければなりません。 Eclipse Java EE IDE for Web Developers の WASCE プラグインをインストールしておく必要があります。
   d. 次のように Axis2 を追加します。「**Web サービス**」>「**Axis2 の設定**」をクリックします。 **Axis2**「**ランタイム**」 タブ> 「**参照 ...**」 多数の `Axis2` jar ファイルが含まれているディレクトリーを開き、「**適用**」をクリックします。
   e. WASCE を Axis2: 「**Web サービス**」 > 「**サーバーおよびランタイム**」をクリックします。 **Server** で「 **IBM WASCE v2.1 Server**」を選択し、 **Web service runtime** の下で「 **Apache Axis2** > **適用** > **OK** 」を選択します。
   f. 次のようにサーバーを開始します。Web パースペクティブを開き、「サーバー」ビューを開きます。「サーバー」ビューを右クリックして、「**新規**」>「**サーバー**」をクリックします。 **IBM WASCE v2.1 Server** が選択され、構成されます> 「**終了**」。 サーバーを始動します。

2. Web サービス・ウィザードを実行するために、StockQuoteAxis サービスを WASCE にデプロイしていることを確認します。
3. WebSphere MQ transport for SOAP サービスを使用してサービスをテストするには、このサービスを Axis 1.4 用 WebSphere MQ transport for SOAP リスナーにデプロイします。965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』を参照してください。

このタスクについて

Eclipse IDE for Java EE 開発者は、 Java50 および Web サービス・ウィザードを使用して、サービスのプロキシー・クラスを生成します。 プロキシー・クラスは、Java 6 で提供される **wsimport** ツールによって作成されるクラスとは異なります。 別の方法として、 **wsimport** を使用してプロキシー・クラスを生成し、作成したパッケージを Eclipse Java EE IDE for Web Developers にインポートすることもできます。

Eclipse IDE for Java EE 開発者の Web サービス・ウィザードは、Web プロジェクト内に Web サービス・クライアントを作成します。 クライアントは、単純な Java アプリケーションとして実行できます。アプリケーション・サーバーは必要ありません。 コードを Java プロジェクトに転送し、 Axis2 JAR ファイルを組み込むようにビルド・パスを構成することもできます。

手順

1. 以下のように、新しいエンタープライズ・プロジェクト内に Web プロジェクトを作成します。
   a) Project Explorer 内で何も選択されていない状態で、空白文字を右クリックし、「**新規**」>「**エンタープライズ・アプリケーション・プロジェクト**」をクリックして、StockQuoteAxis2EAR という名前を付け、「**終了**」をクリックします。 ウィンドウに No と応答して、 Java EE パースペクティブを開くオプションを表示します。

   デフォルトは、WASCE を使用するように設定されています。
   b) StockQuoteAxis2EAR を右クリックして、「**新規**」>「**動的 Web プロジェクト**」をクリックします。 プロジェクトに StockQuoteAxis2WebClient という名前を付け、EAR メンバーシップ・ボックスにチェック・マークを付けて、このプロジェクトを「**StockQuoteAxis2EAR**」に追加します。 WASCE 2.1 がターゲット・ランタイムとして選択されます。
   c)「**新規動的 Web プロジェクト**」ページの「構成」セクションで、「**変更...**」をクリックし、Axis2 Web サービス・プロジェクト・ファセットにチェック・マークを付けます。 **動的 Web モジュール 2.5**、**Java 6.0**、および **WASCE デプロイメント 1.2** は既に検査済みです。 > **OK** > 「**終了**」。 ウィンドウに No と応答して、 Java EE パースペクティブを開くオプションを表示します。
2. 以下のように、サービス用の WSDL をワークスペース内にインポートし、クライアント・プロキシーを生成します。

   この例では、WSDL 文書は HTTP サービス・バインディングを含み、静的 Web クライアント・プロキシーのターゲットになります。 クライアント・プロキシーを生成する前に、Web サービス・バインディング内の URL を、WebSphere MQ transport for SOAP の URL を指すように変更できます。 その後、静的 Web クライアント・プロキシーが、WebSphere MQ transport for SOAP にデプロイされるサービスになります。

   a) 次のように Web Services Explorer を起動します。アクション・バーのアイコンを使用するか、「**実行**」>「**Web Services Explorer の起動**」を使用します。
   b)「**Web Services Explorer**」ウィンドウの WSDL アイコンをクリックして WSDL エクスプローラーを選択し、ナビゲーター・ウィンドウで「**WSDL のメイン**」をクリックし、StockQuoteAxis WSDL ファイルの URL を入力し、「**進む**」をクリックします。
   この例では、次の HTTP サービスから直接 WSDL を入手します。http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
   c) ナビゲーターで、Web サービスの URL がある行をクリックします。 「**アクション**」ウィンドウで、「**ワークベンチへの WSDL のインポート**」をクリックし、「**ワークベンチ・プロジェクト**」として「**StockQuoteAxis2WebClient**」を選択し、「**WSDL ファイル名**」に StockQuoteAxisHTTP.wsdl と入力して「**進む**」をクリックします。
   d)「**StockQuoteAxisHTTP.wsdl**」を右クリックして、「**Web サービス**」>「**クライアントの生成**」をクリックします。 次のように、ウィザードの Web サービス・ページに関する構成情報を確認します。

Server: IBM WASCE v2.1 Server、Web service runtime: Apache Axis2、Client project: StockQuoteAxis2WebClient、 Client EAR project: StockQuoteAxisEAR。 構成を訂正するには、誤っている行をクリックします。

e)「**次へ**」をクリックし、コード生成の設定を確認し、「**終了**」をクリックします。

新しいパッケージ soap.server が作成されており、必要なプロキシーが含まれていることに注意してください。

3. WebSphere MQ transport for SOAP を JMS トランスポートとして実行するようにプロジェクトを構成します。

WebSphere MQ transport for SOAP には、transportSender はありますが、transportReceiver はありません。 つまり、WebSphere MQ transport for SOAP は Axis2 クライアントをサポートします。 現在 Axis2 サービスはサポートしていません。

a) **StockQuoteAxis2WebClient** プロジェクトで、 WebContent\WEB-INF\conf\axis2.xml > 「**開く ...**」 > 「**XML エディター**」を右クリックします。

b) 最後の transportSender (ファイルの末尾近く) を検索し、コメント化されている JMS transportSender > 右クリックして、行> **Add before ...** > **transportSender** を見つけます。

c) **transportSender** を右クリックして「**属性の追加**」 > 「**名前**」を選択し、**transportSender** を右クリックして「**属性の追加**」 > 「**クラス**」を選択します。

d)「**名前**」を右クリックして「**属性の編集**」をクリックし、「**値:**」に jms と入力します。

e)「**クラス**」 > 「**属性の編集**」 > 「 **値の入力:** com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender」>「保存」を右クリックします。

f) 次のように com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender をビルド・パスに追加します。「**StockQuoteAxis2WebClient**」を右クリックして、「**ビルド・パス**」>「**ビルド・パスの構成...**」をクリックし、「**ライブラリー**」タブ >「**外部 JAR の追加...**」をクリックします。 *MQ_INSTALLATION_PATH*\java\lib > 「**OK**」ですべての JAR を選択します。

MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされているディレクトリーです。

4. 同期静的クライアントを作成し、HTTP を使用してテストしてから、WebSphere MQ transport for SOAP を使用して静的クライアントを実行するようプロキシーを変換します。

a)「**Java リソース: src**」 > 「**新規**」 > 「**パッケージ**」 >「パッケージに名前を付ける」 soap.client >「終了」を右クリックします。

b) **soap.client** を右クリックして「**新規**」 > 「**クラス**」を選択し、そのクラスに SQA2StaticClient と名前を指定して、「**完了**」をクリックします。

c) クラスを以下のコードに置き換えて、 「**保存**」をクリックします。

図 *182. SQA2DynamicClient.java*

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. WASCE にデプロイされた StockQuoteAxis サービスと WebSphere MQ transport for SOAP を使用してクライアントをテストします。

a) プロジェクト・エクスプローラーで、 **SQA2StaticClient** > 「**実行 ...**」 > 「**Java アプリケーション**」を右クリックします。
結果の Response is 55.25 がコンソール・ビューに表示されます。 「コンソール」ビューで WASCE コンソール・ウィンドウを選択して、WASCE サーバー StockQuoteAxis called with parameter: ibm の出力を表示することもできます。

b) プロキシーはサービスのアドレス http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis を使用してビルドされたので、静的クライアントは HTTP 上で実行するサービスを呼び出します。 WebSphere MQ transport for SOAP を使用するサービスを呼び出すように静的クライアントを変更できます。 以下の指示では、プロキシーを再ビルドせずに StockQuoteAxisServiceStub.java 内のサービス・アドレスを変更し、axis2.xml をロードするように SQA2StaticClient ランタイム・パラメーターを構成します。 axis2.xml を構成すると、 WebSphere MQ transport for SOAP を使用するように Axis2 が構成されます。

c) StockQuoteAxisServiceStub.java を開きます。
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis の 2 つのオカレンスを以下に置き換えます。

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

d) ここで SQA2StaticClient を実行すると、JMS 用に構成された transportSender が見つからないため、例外がスローされます。
例外は、以下のとおりです。

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) Project Explorer で、「**SQA2StaticClient**」を右クリックして「**次を実行...**」>「**構成の実行...**」をクリックします。 「**(x) = 引数**」 タブに切り替え、 「**VM 引数**」 入力域で、 axis2.conf ファイル> **Apply** > 「**実行**」へのパスを入力します。
VM 引数は、-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml になります。 または、Axis2 構成ファイルへの標準パスを提供できます。

f) 再度 SQA2StaticClient を実行します。 今回の実行では、WebSphere MQ transport for SOAP を使用します。 WASCE コンソールで新しい出力がないことを調べて、そのことを確認します。
SimpleJava リスナーに関連付けられているコンソールまたはコマンド・ウィンドウを開きます。そこに StockQuoteAxis called with parameter: ibm という出力が表示されます。

6. HTTP および WebSphere MQ transport for SOAP 用の動的クライアントを作成してテストします。

a) **soap.client** を右クリックして「**新規**」 > 「**クラス**」を選択し、そのクラスに SQA2DynamicClient と名前を指定して、「**完了**」をクリックします。

b) クラスを以下のコードに置き換えて、 「**保存**」をクリックします。

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
```

```
          + (stub.getQuote(request)).getGetQuoteReturn());
      } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
      }
    }
  }
```

c) 以下のように、SQA2DynamicClient.java に関する実行構成を作成し、パスを axis2.xml に追加します。
-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml

d) SQA2DynamicClient を実行します。 SQA2DynamicClient、WASCE、および **SimpleJavaListener** に関するコンソール出力を確認します。

7. 非同期クライアントを作成し、コールバック・ハンドラーとメインプログラム・スレッド内の結果にアクセスします。

Eclipse Java EE IDE for Web Developers の Web サービス・ウィザードによって作成される非同期クライアント・プロキシーは、 **wsimport** によって作成されるプロキシーとは異なります。 **wsimport** プロキシーは Future、Response、および AsyncHandler 総称タイプを使用します。

Eclipse Java EE IDE for Web Developers の Web サービス・ウィザードは、StockQuoteAxisServiceCallbackHandler 抽象クラスを作成します。StockQuoteAxisServiceCallbackHandler を拡張して、コールバック・ハンドラーを作成しなければなりません。

a) **soap.client** を右クリックして「**新規**」>「**クラス**」を選択し、そのクラスに SQA2CallbackHandler と名前を指定して、「**完了**」をクリックします。

b) クラスを以下のコードに置き換えます。

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) **soap.client** を右クリックして「**新規**」>「**クラス**」を選択し、そのクラスに SQA2AsyncClient と名前を指定して、「**完了**」をクリックします。

d) クラスを以下のコードに置き換えます。

図 *183. SQA2AsyncClient.java*

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
```

```
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                  .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
     }
   }
 }
```

コンソール出力は以下のとおりです。

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

サンプル Axis2 クライアント

サンプル・プロキシーは、Java 6 にパッケージされている **wsimport** ツールを使用して生成されます。 6 つのサンプルが提供されています。

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

サンプル StockQuoteAxis サーバー用のクライアント・サンプルが生成されます。 **amqwdepoyWMQServer** コマンドを使用して WSDL を生成します。-w スイッチを指定して rpcLiteral スタイルを選択します。 以下のコマンドを使用して、サンプルのプロキシーを生成します。

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

図 *184. DynamicProxyClientSync.java*

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

図 *185. DynamicProxyClientAsyncPolling.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by
polling...");
      Response<Float> response = service.getQuoteAsync("49");
```

```
      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

図 *186. DynamicProxyClientAsyncCallback.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
```

```
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

図 *187. DispatchClientSync.java*

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";
```

```
      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
      System.out.println(" > getQuoteOneWay call has returned");

      /*****************************************
       * Create Request Reply SOAPMessage request.
       *****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
```

```
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

図 *188. DispatchClientAsyncPolling.java*

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
```

```
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by
polling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

図 *189. DispatchClientAsyncCallback.java*

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
```

```
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
```

```
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

関連タスク

Eclipse を使用して WebSphere transport for SOAP 用の JAX-RPC クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように Axis 1.4 Web サービス・クライアントを開発します。

Microsoft Visual Studio 2008 を使用して WebSphere transport for SOAP 用の .NET 1 または .NET 2 クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように .NET 1 または .NET 2 Web サービス・クライアントを開発します。

Microsoft Visual Studio 2008 を使用して WebSphere transport for SOAP 用の .NET 1 または .NET 2 クライアントを開発する

WebSphere MQ transport for SOAP を使用して実行するように .NET 1 または .NET 2 Web サービス・クライアントを開発します。

始める前に

以下のように、.NET 1 または .NET 2 クライアントの開発を開始する方法は多数あります。

1. **amqwdeployWMQService** を使用し、Web サービスからクライアント・スタブを生成して Visual Studio 内にインポートします。
2. **java2wsdl** を使用して Web サービスの Java 実装から WSDL を生成してから、.NET に付属の wsdl.exe を使用してクライアント・スタブを生成します。
3. **amqswsdl** を使用し、サービスの .NET .asmx 実装から WSDL を生成してから、wsdl.exe を使用します。
4. HTTP 用のサービスを開発してデプロイした場合は、「**Web 参照の追加 ...**」 を使用します。 Visual Studio のウィザードを使用して、HTTP サービスにアクセスするようにクライアントを構成します。 WebSphere MQ transport for SOAP にデプロイされたサービスを参照する URL を変更します。

このタスクでは、969 ページの『Microsoft Visual Studio 2008 を使用して WebSphere MQ transport for SOAP 用の .NET 1 サービスまたは .NET 2 サービスを開発する』で開発したサービスを使用します。

このタスクについて

以下のステップに従って、HTTP および WebSphere MQ transport for SOAP 用の .NET 1 または .NET 2 クライアントを作成します。

手順

1. クライアント・コンソール・アプリケーションを作成し、StockQuote HTTP Web サービスを呼び出すように変更します。
   a) **Solution Explorer** 内で**ソリューション**「**StockQuoteDotNet**」を右クリックして、「追加...」>「新規プロジェクト」をクリックします。 「**C#**」プロジェクト・タイプ、「**.NET Framework 2.0**」、および「**コンソール・アプリケーション**」を選択します。 プロジェクトに StockQuoteClientDotNet という名前を付け、「**OK**」をクリックします。
   b) **Solution Explorer** 内で**ソリューション**「**StockQuoteDotNet**」を右クリックして、「追加...」>「新規プロジェクト」をクリックします。 「**C#**」プロジェクト・タイプ、「**.NET Framework 2.0**」、および「**コンソール・アプリケーション**」を選択します。 プロジェクトに StockQuoteClientDotNet という名前を付け、「**OK**」をクリックします。
   c)「**StockQuoteClientDotNet**」を右クリックして「**始動プロジェクトとして設定**」をクリックします。
   d)「**StockQuoteClientDotNet**」を右クリックして「**Web 参照の追加...**」をクリックし、このソリューション内の Web サービスを参照し、「**StockQuote**」>「**参照の追加**」を選択します。 ローカル・ホストと新しい構成ファイル app.config への Web 参照を追加したことに注意してください。
   e) Solution Explorer で、コンソール・アプリケーションの名前を Program.cs から StockQuoteClientDotNet.cs に変更し、「**OK**」をクリックして、Program.cs が使用されているすべての場所を StockQuoteClientDotNet.cs に変更します。
   f) StockQuoteClientDotNet.cs の内容を 1001 ページの図 190 のコードに置き換えます。

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

図 *190. HTTP StockQuoteClientDotNet* プログラム

g) StockQuoteClientDotNet を起動し、StockQuote.asmx サービスに対してテストします。

i) **F5** を押し、アクション・バー内の緑色の矢印をクリックするか、「**デバッグ**」>「**デバッグの開始 (F5)**」をクリックします。

StockQuoteDOtNet プロジェクトが同じソリューション内にある場合は、自動的に開始されます。ない場合は、最初にサービスを始動する必要があります。

ワークスペースの背後にコマンド・ウィンドウと結果が開きます。 Console.ReadLine(); ステートメントにより、**Enter** を押さないと閉じません。

ヒント: StockQuote.asmx が StockQuoteDotNet プロジェクトの先頭ページであることを確認してください。

2. WebSphere MQ transport for SOAP を使用する StockQuote.asmx サービスを呼び出すように StockQuoteClientDotNet を変更します。

a) 太字で示されている行をクライアントに追加します。

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

図 *191.* 変更された *StockQuoteClientDotNet* プログラム

別の方法として、デフォルトの URL を変更します。 **StockQuoteClientDotNet** > **プロパティー** > **Settings.settings** を開き、 StockQuoteClientDotNet_localhost_StockQuote プロパティーの値を WebSphere MQ transport for SOAP URL に変更します。

b) amqsoap.dll への参照を追加します。

i) **ソリューションエクスプローラ**の **StockQuoteClientDotNet** プロジェクトで、「**参照**」>「**参照の追加 ...**」>「**参照**」タブを右クリックし、 *MQ_INSTALLATION_PATH*\bin を選択して **amqsoap.dll** >「**OK**」を選択します。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。

3. WebSphere MQ transport for SOAP を使用する StockQuote.asmx サービスを使用してクライアントをテストします。

a) StockQuoteDotNet プロジェクト・ディレクトリーでコマンド・ウィンドウを開きます。 .\StockQuoteDotNet\StockQuoteDotNet > .bin\StockQuoteDotNet.dll が存在することを確認します。 ない場合は、ソリューションを再ビルドします。

b) コマンド **amqwRegisterdotNet** を入力します。

amqwRegisterdotNet は、インストール済み環境につき一度のみ実行する必要があります。

c) genAsmxWMQBits を指定して **amqwdeployWMQServer** を実行した場合は、以下のように .NET SOAP リスナーを実行します。

generated\server\startWMQNListener

d) 別の方法として、以下のように直接リスナーを実行します。

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. Visual Studio 2008 で、**F5** を押して StockQuoteClientDotNet を実行します。

.NET Framework 1 および .NET Framework 2 Web サービス・クライアント

WebSphere MQ transport for SOAP で提供されるサンプル .NET クライアントは、生成済みのスタブを使用してサンプルの Axis および .NET サービスを呼び出します。

.NET Framework 1 および .NET Framework 2 クライアントの場合、WebSphere MQ は .NET クライアントを使用して Web サービスへのアクセスを提供します。 **amqwdeployWMQService** コマンドにはオプション genProxiestoDotNet があり、このオプションは Web サービスに関する .NET Framework 1 または .NET Framework 2 クライアント・スタブを生成します。 .NET **wsdl** ツールか Microsoft Visual Studio 2005 または 2008 によって生成されたクライアント・スタブを使用することもできます。

サンプルの .NET Framework 1 および .NET Web サービス・クライアントは、*MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet にインストールされています。 *MQ_INSTALLATION_PATH* WebSphere MQ がインストールされているディレクトリーです。

SQVB2Axis.vb
: 1003 ページの図 192 の SQVB2Axis.vb は、**StockQuoteAxisService** サービスを呼び出す Visual Basic クライアントです。

SQVB2DotNet.vb
: 1004 ページの図 193 の QVB2DotNet.vb は、**StockQuoteDotNet** サービスを呼び出す Visual Basic クライアントです。

SQCS2Axis.cs
: 1004 ページの図 194 の SQCS2Axis.cs は、**StockQuoteAxisService** サービスを呼び出す C# クライアントです。 サービスの URL は、コマンド行に別の URL を指定することによって指定変更できます。

SQCS2DotNet.cs
: 1004 ページの図 195 の SQCS2DotNet.cs は、**StockQuoteDotNet** サービスを呼び出す C# クライアントです。 サービスの URL は、コマンド行に別の URL を指定することによって指定変更できます。

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

図 *192. SQVB2Axis*

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

図 *193. SQVB2DotNet*

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

図 *194. SQCS2Axis*

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

図 *195. SQCS2DotNet*

関連タスク

Eclipse を使用して WebSphere transport for SOAP 用の JAX-RPC クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように Axis 1.4 Web サービス・クライアントを開発します。

Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する
WebSphere MQ transport for SOAP を使用して実行するように Axis2 Web サービス・クライアントを開発します。 WebSphere MQ transport for SOAP で提供されるサンプル Axis2 クライアントをリストし、**wsimport** コマンドを使用してプロキシーを生成します。

WebSphere MQ transport for SOAP を使用した Web サービスのデプロイ

さまざまなサーバー環境の 1 つに Web サービスをデプロイし、WebSphere MQ transport for SOAP を使用してそれに接続します。

始める前に

Web サービスを作成し、ターゲット環境で SOAP over HTTP を使用してそれをテストします。

このタスクについて

Web サービスをデプロイして、さまざまな SOAP ランタイム環境で WebSphere MQ transport for SOAP と連動させることができます。 サービスを Axis 1.4 にデプロイする場合は、WebSphere MQ と共にインストールされるソフトウェアのみを使用して行えます。 他のランタイム環境の場合は、追加ソフトウェアをインストールする必要があります。

デプロイメント手順が示されているサーバーに対してのみ WebSphere MQ transport for SOAP を実行できるという制限があるわけではありません。 リストされた環境のいずれかにサービスをデプロイするには、この手順を使用します。

注 : 統合環境の中には、W3C 勧告の JMS SOAP バインディングを使用した SOAP over JMS だけでなく、WebSphere MQ transport for SOAP バインディングを使用した SOAP over JMS を提供しているものもあります。 WebSphere MQ の 7.0.1.2 までのリリースは、WebSphere MQ transport for SOAP バインディングのみサポートします。 7.0.1.3 以降では、SOAP over JMS に関する W3C 勧告候補に準拠した URI を使用して Axis2 クライアントをデプロイできます。 チュートリアル『Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5』を参照してください。

サービスを *Axis 1.4* にデプロイし、*amqwdeployWMQService* を使用して *WebSphere transport for SOAP* に使用する

デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

始める前に

1. WebSphere MQ transport for SOAP のインストール手順に従います。
2. **runivt** コマンドを使用して、インストールと環境を検査します。
3. サービスを再デプロイするには、以下のようにします。
   a. `./generated` サブディレクトリーとそのすべてのサブディレクトリーを削除します。
   b. 宛先キューから要求を除去し、宛先キューを削除します。
   c. ステップ 1005 ページの『2』 から手順を進めます。

このタスクについて

この手順は、Axis 1.4 サービスを初めてデプロイするためのものです。 Axis 1.4 サービスを再始動するには、Axis 1.4 SOAP リスナーを再実行します (ステップ 1006 ページの『11』)。

新規の Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイするには、以下の手順を使用します。

手順

1. デプロイメント・ファイルを保持するためのディレクトリー *deployDir* を作成します。
   デプロイメント・ユーティリティーでは、サービスはそれぞれ別個のディレクトリーからデプロイされなければなりません。
2. *deployDir* で、コマンド・ウィンドウ (Windows の場合) または X Window System を使用してコマンド・シェル (UNIX and Linux の場合) を開き、**amqwdeployWMQService** を実行します。

3. **amqwsetcp** を実行してクラスパスを設定します。

   バージョン 5.0 以降の同じバージョン・レベルの JRE と JDK が、クラスパスになければなりません。
4. クラス・ソース *className*.java を *deployDir* にコピーします。
5. *className* と同じパッケージ内のすべての Java ソース・ファイルを *deployDir*/*packageName* にコピーします。ここで、 *packageName* はパッケージ名に対応するディレクトリー・ツリーです。
6. 実行 **javac** *packageName*.*className*.

   javac が他のクラスを検出するためには、現行ディレクトリー "." または *packageName* ディレクトリーにパスを追加しなければならない場合もあります。
7. サービスの Axis WSDL を次のように作成します。

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. サービスの WebSphere MQ リソースを次のように作成します。

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

   ヒント:

   新規キュー・マネージャーとそれが必要とするリソースをセットアップする場合、作成とテストを行うには、**setupWMQSOAP** を実行します。

   新しいキュー・マネージャーをデフォルトとしてセットアップする場合は、*WMQ install directory*\tools\soap\samples ディレクトリーから **setupWMQSOAP** のコピーを作成し、その行に -q パラメーターを追加します。

```
call :try -q crtmqm %QMGR%
```

9. Axis リスナーを作成し、次のようにサービスをデプロイします。

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. サービスの WSDL を生成する必要がある場合は、クライアント・スタブ (つまりクライアント・プロキシー) を生成し、以下のパラメーターのいずれかを指定して **amqwdeployWMQService** を実行します。

    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - genProxiestoAxis

    注: WSDL を生成するのは、プロキシーを生成する前でなければなりません。 *className*.java をコンパイルするためにインポートされるすべてのクラスを検出できるように CLASSPATH がセットアップされていなければ、AllAxis オプションは失敗します。 *className*.java を含むパッケージに複数の Java ファイルがある場合は、最初に **javac** を使用してコンパイルする必要があります。 **amqwdeployWMQService** -f *packageName*.*className*.java -c CompileJava は *className*.java のみをコンパイルします。
11. 生成された Axis リスナーを開始します。

```
.\generated\server\startWMQJListener.cmd
```

関連タスク

サービスを .NET Framework 1 または 2 サービスにデプロイして WebSphere MQ transport for SOAP を使用する
.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

サービスを CICS Transaction Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

W3C SOAP over JMS を使用するための WebSphere Application Server の構成
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

サービスを WebSphere ESB および Process Server サービス・エンドポイントにデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

サービスを *.NET Framework 1* または *2* サービスにデプロイして *WebSphere MQ transport for SOAP* を使用する

.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

始める前に

1. WebSphere MQ transport for SOAP のインストール手順に従います。
2. **runivt** コマンドを使用して、インストールと環境を検査します。
3. .NET Framework ファイルの `wsdl.exe` と `csc.exe` のパスが設定されていなければなりません。 PATH 変数で示される `wsdl.exe` と `csc.exe` のコピーは、.NET Framework と同じレベルでなければなりません。 複数の .NET Framework がインストールされている場合や Visual Studio を使用している場合は、PATH 変数を注意深く確認してください。
4. サービスを再デプロイするには、以下のようにします。
   a. `./generated` サブディレクトリーとそのすべてのサブディレクトリーを削除します。
   b. 宛先キューから要求を除去し、宛先キューを削除します。
   c. ステップ 1008 ページの『2』 から手順を進めます。

このタスクについて

この手順は、.NET サービスを初めてデプロイするためのものです。 .NET サービスを再始動するには、.NET SOAP リスナーを再実行します (ステップ 1008 ページの『9』)。

新規の .NET Framework 1 または .NET Framework 2 サービスを WebSphere MQ transport for SOAP にデプロイするには、以下の手順を使用します。

手順

1. デプロイメント・ファイルを保持するためのディレクトリー *deployDir* を作成します。

   デプロイメント・ユーティリティーでは、サービスはそれぞれ別個のディレクトリーからデプロイされなければなりません。
2. *deployDir* でコマンド・ウィンドウを開き、**amqwdeployWMQService** を実行します。

```
C:\IBM\ID\QuoteClient>
```

3. **amqwsetcp** を実行してクラスパスを設定します。

   クラスパスは Axis クライアントにのみ必要です。
4. .NET サービス *className*.asmx を *deployDir* にコピーします。
5. サービス実装をライブラリー (.dll) にビルドします。

   インライン・サービス実装は、*className*.asmx に含まれます。 分離コード・サービス実装は、*className*.asmx.cs である可能性があります。

   1008 ページの図 196 は、.NET Framework V2 サービスをライブラリーとしてビルドするためのコマンドの例を示しています。

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

   図 *196. .NET Framework V2* サービスのビルド・コマンド

6. *className*.dll を *deployDir**bin* にコピーします。
7. WebSphere MQ リソースをセットアップし、サービスに必要なリスナーを次のように作成します。

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. サービスの WSDL を生成する必要がある場合は、クライアント・スタブ (つまりクライアント・プロキシー) を生成し、以下のパラメーターのいずれかを指定して **amqwdeployWMQService** を実行します。

   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - genProxiestoAxis

   注 : WSDL を生成するのは、プロキシーを生成する前でなければなりません。
9. 生成された .NET リスナーを開始します。

```
.\generated\server\startWMQNListener.cmd
```

関連タスク

サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する

デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

サービスを CICS Transaction Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

W3C SOAP over JMS を使用するための WebSphere Application Server の構成
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

サービスを WebSphere ESB および Process Server サービス・エンドポイントにデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

サービスを *CICS Transaction Server* にデプロイして *WebSphere transport for SOAP* を使用する

WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

始める前に

WebSphere MQ 用のクライアントまたはサービスを、HTTP 用に作成する場合と同じツールを使用して作成します。 CICS には、**Java2wsdl** と **wsdl2Java** に対応する以下のツールがあります。

- **DFHWS2LS** は、Web サービス記述を開始点とします。 メッセージの記述とそれらのメッセージで使用されるデータ型を使用して、高水準言語のデータ構造体を構成します。 さまざまな言語で書かれたアプリケーション・プログラム内の構造体で使用できます。
- **DFHLS2WS** は、高水準言語データ構造体を開始点とします。 この構造体を使用して、メッセージの記述を含んだ Web サービス記述を構成します。 言語データ構造体からメッセージのスキーマも作成します。

CICS 製品資料の Web サービスの作成 の説明に従って、Web サービスを作成します。

このタスクについて

「CICS」 製品資料の WebSphere MQ トランスポートを使用するための 「CICS」 の構成 の説明に従ってください。 手順に従えば、Web サービスを WebSphere MQ transport for SOAP にデプロイできます。

関連タスク

サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する
デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

サービスを .NET Framework 1 または 2 サービスにデプロイして WebSphere MQ transport for SOAP を使用する
.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

W3C SOAP over JMS を使用するための WebSphere Application Server の構成
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

サービスを WebSphere ESB および Process Server サービス・エンドポイントにデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

サービスを *WebSphere Application Server* にデプロイして *WebSphere transport for SOAP* を使用する

WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

始める前に

Rational Application Developer、WebSphere Integration Developer、または Web サービス・ツールキットを使用して、Web サービスを作成します。

このタスクについて

WebSphere MQ transport for SOAP を WebSphere Application Server の SOAP トランスポートとして使用してサービスをデプロイするには、以下の手順を使用します。

手順

1. WebSphere MQ を、WebSphere Application Server のサービス統合バスの JMS メッセージング・プロバイダーとして構成します。
2. サービスが必要とする WebSphere MQ リソースを構成します。
3. WebSphere Application Server Network Deployment 製品資料の「同期 SOAP over JMS エンドポイント・リスナーの JMS リソースの構成」の説明に従ってください。
   他の WebSphere Application Server プラットフォーム用の対応手順があります。
4. サービス URI を WebSphere MQ transport for SOAP URI に準拠するように変更します。
5. サービスを WebSphere Application Server にデプロイします。

次のタスク

クライアントがサービスに照会してその応答で WSDL を受信できるように、HTTP をトランスポートとするサービスをデプロイします。

関連タスク

サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する
デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

サービスを .NET Framework 1 または 2 サービスにデプロイして WebSphere MQ transport for SOAP を使用する
.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

サービスを CICS Transaction Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

W3C SOAP over JMS を使用するための WebSphere Application Server の構成
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

サービスを WebSphere ESB および Process Server サービス・エンドポイントにデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

W3C SOAP over JMS を使用するための *WebSphere Application Server* の構成

SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

始める前に

このタスクでは WebSphere Application Server v7.0.0.9 および WebSphere MQ v7.0.1.3 が必要です。

このタスクについて

このタスクには以下の 2 つのステップがあります。

手順

1. 1012 ページの『WebSphere MQ リソースの構成』
2. 1012 ページの『WebSphere Application Server リソースの構成』

次のタスク

1012 ページの『WebSphere MQ リソースの構成』

関連タスク

サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する
デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

サービスを .NET Framework 1 または 2 サービスにデプロイして WebSphere MQ transport for SOAP を使用する
.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

サービスを CICS Transaction Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

サービスを WebSphere ESB および Process Server サービス・エンドポイントにデプロイして WebSphere transport for SOAP を使用する
WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

WebSphere MQ リソースの構成

始める前に

Axis2 をサポートするには WebSphere MQ 7.0.1.3 以降が必要です。

このタスクについて

タスクの説明を単純にするために、他のソフトウェアと同じワークステーション上に WebSphere MQ がインストールされていて、バインディング接続を使用すると想定します。 WebSphere Application Server および Axis2 クライアントの構成はクライアント接続を扱います。 クライアント接続を使用してタスクと共に実行するには、Axis2 クライアント・コンピューターと WebSphere Application Server コンピューターの両方において、要求/応答キューとの間でメッセージを put および get できることを確認してください。

ここでも説明を単純化するために、セキュリティー構成を使用しません。 ユーザー ID には完全な `mqm` 権限があります。

手順

1. デフォルト・キュー・マネージャー `QM1` を作成します。

   WebSphere MQ エクスプローラーを使用して、`QM1` をデフォルト・キュー・マネージャーとして作成します。 自動的に開始するよう構成して、リスナーを作成するオプションを選択します。 代わりの方法として、以下のコマンドを使用できます。

   ```
   crtmqm -q -sa QM1
   strmqm
   echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
              control(qmgr) replace | runmqsc
   echo start listener(LISTENER.TCP) | runmqsc
   ```

2. 要求キュー `REQUESTAXIS` および応答キュー `REPLYAXIS` を定義します。

   エクスプローラーまたは以下のコマンドを使用します。

   ```
   echo define ql(REQUESTAXIS) replace | runmqsc
   echo define ql(REPLYAXIS) replace | runmqsc
   ```

次のタスク

1012 ページの『WebSphere Application Server リソースの構成』

WebSphere Application Server リソースの構成

始める前に

W3C SOAP over JMS をサポートするには WebSphere Application Server v7 が必要です。 この構成は、WebSphere Application Server バージョン 7.0 Test Environment v7.0.0.9 Update 1 で実行されました。 WebSphere Application Server は、Rational Software Architect for WebSphere Software 7.5.4 に付属のものを使用しました。 入手可能な最新アップデートを適用して Rational Software Architect を v7.5.5.1 に更新しました。

インストール・プロセスの一部として、WebSphere Application Server 用のプロファイルを作成します。 このタスクでは、管理セキュリティーは有効にしません。 デフォルト・プロファイル名は `was70profile1`、サーバーは `server1` です。

このタスクについて

WebSphere Application Server を構成します。 Rational Application Developer からサーバーを開始して、サーバー・ビューから管理コンソールを開始できます。または、コマンド・ファイルを使ってサーバーを開始して、ブラウザーを使ってサーバーを管理することもできます。 このタスクでは 2 番目の方法を使用します。

サーバー・コマンド・ファイルは、 *Rational Installation Root*\SDP\runtimes\base_v7\profiles\was70profile1\bin フォルダーにあります。 検査するログ・ファイルは、 *Rational Installation Root*\SDP\runtimes\base_v7\profiles\was70profile1\logs\server1 にあります。

規則として、すべての WebSphere MQ オブジェクト名は大文字、WebSphere MQ オブジェクトを参照するすべての JNDI 名は小文字です。

手順

1. サーバーを始動します。

   ```
   startServer server1
   ```

2. ブラウザーを開始し、管理コンソールを開いてログインします。

   ```
   http://localhost:9061/ibm/console/unsecureLogon.jsp
   ```

   ユーザー ID フィールドに任意のストリングを入力します。

3. 接続ファクトリー qm1 を作成します

   a) ナビゲーターで、 「**リソース**」 > 「**JMS**」 > 「**接続ファクトリー**」を開きます。

   b) 接続ファクトリー・ウィンドウで、有効範囲「**Node=*nodename***」を選択して「**新規**」をクリックします。

   c)「 **WebSphere MQ メッセージング・プロバイダー** 」>「 **OK**」を選択します。

   d) 1013 ページの表 143 >「**次へ**」からキュー・マネージャー接続情報を指定します。

   表 *143.* キュー・マネージャー接続情報

フィールド名	**値**
名前	qm1
JNDI 名	qm1

   e) 接続方式として 「**このウィザードに必要なすべての情報を入力してください**」 を選択し、 「**次へ**」を選択します。

   f) キュー接続の詳細として QM1 と入力し、 「**次へ**」をクリックします。

   g) 1013 ページの表 144 >「**次へ**」から接続の詳細を入力します。

   表 *144.* 接続詳細

フィールド名	**値**
トランスポート	Bindings, then client
Hostname	localhost
ポート	1414
サーバー接続チャネル	SYSTEM.DEF.SVRCONN

   h)「**接続のテスト**」 >「**次へ**」 >「**終了**」 >「**保存**」。

4. JMS 要求キュー requestaxis を作成します。

   a) Navigator で、 「**リソース**」 > 「**JMS**」 > 「**キュー**」を開きます。

   b) 接続ファクトリー・ウィンドウで、有効範囲「**Node=*nodename***」を選択して「**新規**」をクリックします。

   c)「 **WebSphere MQ メッセージング・プロバイダー** 」>「 **OK**」を選択します。

   d) 1014 ページの表 145 >「**OK**」 >「**保存**」からキューの詳細を入力します。

表 *145.* キューの詳細	
フィールド名	**値**
名前	`requestaxis`
JNDI 名	`requestaxis`
キュー名	`REQUESTAXIS`
キュー・マネージャー名	`QM1`

5. ステップ 1013 ページの『4』を繰り返して JMS 応答キュー `replyaxis` を作成します。
6. アクティベーション・スペック `qm1as` を作成します。

   アクティベーション・スペックは、要求キューでのメッセージ到着時に Web サービス・ルーター・メッセージ駆動型 Bean (MDB) を起動します。 MDB は、Rational Application Developer Web サービス・ウィザードによって作成される Web サービスのデプロイメント記述子で定義されます。

   a) Navigator で、 「**リソース**」 > 「**JMS**」 > 「**アクティベーション・スペック**」を開きます。

   b) 接続ファクトリー・ウィンドウで、有効範囲「**Node=*nodename***」を選択して「**新規**」をクリックします。

   c)「 **WebSphere MQ メッセージング・プロバイダー** 」>「 **OK**」を選択します。

   d) 1014 ページの表 146 >「**次へ**」からアクティベーション・スペックの基本属性を入力します。

表 *146.* アクティベーション・スペックの名前	
フィールド名	**値**
名前	`qm1as`
JNDI 名	`qm1as`

   e) 1014 ページの表 147 >「**次へ**」から MDB 情報を指定します。

表 *147. MDB* 情報	
フィールド名	**値**
宛先 JNDI 名	`requestaxis`
メッセージ・セレクター	ブランクのまま
宛先タイプ	`Queue`

   f) 接続方式として 「**このウィザードに必要なすべての情報を入力してください**」 を選択し、 「**次へ**」を選択します。

   g) キュー接続の詳細として `QM1` と入力し、 「**次へ**」をクリックします。

   h) 1013 ページの表 144 >「**次へ**」から接続の詳細を入力します。

表 *148.* 接続詳細	
フィールド名	**値**
トランスポート	`Bindings, then client`
Hostname	`localhost`
ポート	`1414`
サーバー接続チャネル	`SYSTEM.DEF.SVRCONN`

   i)「**接続のテスト**」 >「**次へ**」 >「**終了**」 >「**保存**」。
7. 応答キュー用のキュー接続ファクトリー `jms/WebServicesReplyQCF` を作成します。

Web サービス・ルーターは、キュー接続ファクトリーを使って応答キューにアクセスします。 Web サービスのデプロイメント記述子では、キュー接続ファクトリーに jms/WebServicesReplyQCF デフォルト JNDI 名が付けられます。 デプロイメント記述子の中でこの名前を変更できます。 このタスクでは、デフォルト名を JMS リソース定義に追加します。

a) Navigator で、 「**リソース**」 > 「**JMS**」 > 「**キュー接続ファクトリー**」を開きます。

b) 接続ファクトリー・ウィンドウで、有効範囲「**Node=*nodename***」を選択して「**新規**」をクリックします。

c)「 **WebSphere MQ メッセージング・プロバイダー** 」>「 **OK**」を選択します。

d) 1015 ページの表 149 >「**次へ**」から、キュー接続ファクトリーの基本属性を入力します。

表 *149.* キュー接続ファクトリーの名前

フィールド名	**値**
名前	WebServicesReplyQCF
JNDI 名	jms/WebServicesReplyQCF

e) 接続方式として 「**このウィザードに必要なすべての情報を入力してください**」 を選択し、 「**次へ**」を選択します。

f) キュー接続の詳細として QM1 と入力し、 「**次へ**」をクリックします。

g) 1013 ページの表 144 >「**次へ**」から接続の詳細を入力します。

表 *150.* 接続詳細

フィールド名	**値**
トランスポート	Bindings, then client
Hostname	localhost
ポート	1414
サーバー接続チャネル	SYSTEM.DEF.SVRCONN

h)「**接続のテスト**」 >「**次へ**」 >「**終了**」 >「**保存**」。

次のタスク

972 ページの『W3C SOAP over JMS 用の JAX-WS EJB Web サービスの開発』

サービスを *WebSphere ESB* および *Process Server* サービス・エンドポイントにデプロイして *WebSphere transport for SOAP* を使用する

WebSphere MQ transport for SOAP は、WebSphere ESB および Process Server によって直接サポートされるわけではありません。 カスタム Export を構成する必要があります。

このタスクについて

WebSphere Integration Developer で SOAP データ形式変更を行え、これを WebSphere MQ JMS Export にバインドすることにより、カスタム WebSphere MQ JMS SOAP Export を作成できます。

カスタマイズ Export を作成して WebSphere MQ transport for SOAP を介して SOAP 要求を受信するようにするには、以下の手順に従います。

手順

1.「 WebSphere Process Server for Multiplatforms V6.2 」製品資料の「 インポートおよびエクスポートの概要 」および「 WebSphere MQ への接続方法」を参照してください。
2. IBM Business Process Manager バージョン 8.6 製品資料の 「 MQ JMS エクスポート・バインディングの生成」 のタスクに従います。

『製品付属の JMS データ・フォーマット変換』に記載されている SOAP データ・バインディングを使用して SOAP メッセージをフォーマット設定します。

関連タスク

サービスを Axis 1.4 にデプロイし、amqwdeployWMQService を使用して WebSphere transport for SOAP に使用する

デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行し、Axis 1.4 リスナーを開始することによって、Axis 1.4 サービスを WebSphere MQ transport for SOAP にデプロイします。

サービスを .NET Framework 1 または 2 サービスにデプロイして WebSphere MQ transport for SOAP を使用する

.NET Framework 1 または 2 サービスを WebSphere MQ transport for SOAP にデプロイします。 デプロイメント・ディレクトリーを作成し、**amqwdeployWMQService** コマンドを実行した後、.NET リスナーを開始します。

サービスを CICS Transaction Server にデプロイして WebSphere transport for SOAP を使用する

WebSphere MQ transport for SOAP が CICS Transaction Server 4.1 Web サービス・サポートに組み込まれます。

サービスを WebSphere Application Server にデプロイして WebSphere transport for SOAP を使用する

WebSphere MQ transport for SOAP が、WebSphere Application Server のサービス統合バスに組み込まれます。

W3C SOAP over JMS を使用するための WebSphere Application Server の構成

SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続する 1 番目のステップです。 トランスポートとして W3C SOAP over JMS にバインドされた Web サービスを開発してデプロイするために、WebSphere MQ リソースおよび WebSphere Application Server リソースを構成します。

Web サービス・クライアントをデプロイして WebSphere MQ transport for SOAP を使用する

Web サービス・クライアントをさまざまなクライアント環境の 1 つにデプロイし、WebSphere MQ transport for SOAP を使用してサービスに接続します。

始める前に

Web サービスを作成し、それをデプロイして WebSphere MQ transport for SOAP を使用するようにします。

このタスクについて

Web サービス・クライアントをデプロイして、さまざまなクライアント環境で WebSphere MQ transport for SOAP と連動させることができます。 WebSphere MQ とともにインストールされたソフトウェアのみを使用して、Java クライアントを Axis 1.4 にデプロイできます。 他のクライアント環境の場合は、追加ソフトウェアをインストールする必要があります。

デプロイメント手順が示されているクライアント環境でのみ WebSphere transport for SOAP を実行できるという制限があるわけではありません。 サポートされる環境のいずれかにクライアントをデプロイするには、この手順を使用します。

注 : 統合環境の中には、W3C 勧告の JMS SOAP バインディングを使用した SOAP over JMS だけでなく、WebSphere MQ transport for SOAP バインディングを使用した SOAP over JMS を提供しているものもあります。 WebSphere MQ の 7.0.1.2 までのリリースは、WebSphere MQ transport for SOAP バインディングのみサポートします。 7.0.1.3 以降では、SOAP over JMS に関する W3C 勧告候補に準拠した URI を使用して Axis2 クライアントをデプロイできます。 チュートリアル『Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5』を参照してください。

IBM WebSphere MQ transport for SOAP を使用した *Web* サービス・クライアントの *Axis 1.4* へのデプロイ

クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 IBM WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

始める前に

ヒント : HTTP にサービスをデプロイし、HTTP 用クライアントの開発とテストを行い、IBM WebSphere MQ transport for SOAP 用のクライアントを変更します。

1. クライアントに Register.extension() 呼び出しを追加します。
2. クライアント・プロキシー・ロケーター・クラスで、静的な Web サービス・アドレスを IBM WebSphere MQ transport for SOAP 用 URI を使用するように変更します。

このタスクについて

Axis 1.4 クライアントで IBM WebSphere MQ transport for SOAP を使用するようにデプロイする場合、HTTP クライアントの場合と比較してデプロイメントに追加のステップが 1 つ必要になります。 クライアント・デプロイメント記述子 `client-config.wsdd` を作成し、それによって `jms:` トランスポートを送信側クラス `com.ibm.mq.soap.transport.jms.WMQSender` にマップする必要があります。

コマンド **amqwdeployWMQService** を使用してクライアント・プロキシーを生成する場合、このコマンドが生成したディレクトリーを使ってクライアントをデプロイできます。

手順

1. クライアントのデプロイメント・ファイルを保持するためのディレクトリー *deployDir* を作成します。
2. *deployDir* で、Windows システムの場合はコマンド・ウィンドウを、UNIX and Linux システムの場合は X Window システムを使用してコマンド・シェルを開きます。
3. **amqwsetcp.cmd** コマンドを実行して CLASSPATH
4. **amqwclientconfig.cmd** コマンドを実行して、Axis 1.4 クライアントのデプロイメント記述子 `client-config.wsdd` を *deployDir* に作成します。
5. クライアント・パッケージのクラス、クライアント・プロキシー・クラス、およびクライアントが使用するライブラリーが、CLASSPATH に存在することを確認します。

   amqwdeployWMQService は、.NET クライアント・プロキシーを `./generated/server/soap/client/remote/dotnetService` に、Axis 1.4 プロキシーを `./generated/server/soap/client/remote/`*`client package`* に配置します。

例

Axis 1.4 Java クライアントの構成と出力のサンプルを 1018 ページの図 199 に示します。 クライアント (1018 ページの図 198) は、入力パラメーターをエコー出力する Web サービスを呼び出します。 サービス定義 1018 ページの図 197 には、サービスの WSDL に書かれた URI を示します。

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                      name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

図 *197.* サービス定義

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                   + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

図 *198. Axis 1.4 Java* クライアント

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

図 *199.* クライアント構成および出力

次のタスク

1. クライアントを IBM WebSphere MQ クライアントとしてデプロイする場合は、クライアントとサーバーの接続チャネルを構成します。
2. クライアントをサービスへの別のキュー・マネージャーにデプロイする場合は、宛先キューをクライアントで使用可能にする必要があります。 サービス・キュー・マネージャー上の宛先キューをクラスター・キューとして構成するか、クライアント・キューマネージャー上でリモート・キュー定義として構成します。

関連タスク

WebSphere MQ transport for SOAP を使用するための Axis2 への Web サービス・クライアントのデプロイ
デプロイメント・ディレクトリーと Axis2 構成ファイルをクライアント用に準備します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

W3C SOAP over JMS を使用した Axis2 クライアントへのデプロイ

SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 4 番目のステップです。 WebSphere MQ transport for SOAP 用に開発された Axis2 クライアントの URL を変更して、SOAP over JMS の W3C 勧告候補を使用するようにします。

WebSphere MQ transport for SOAP を使用するための .NET Framework 1 および 2 への Web サービス・クライアントのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供します。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

WebSphere MQ transport for SOAP を使用するための *Axis2* への *Web* サービス・クライアントのデプロイ

デプロイメント・ディレクトリーと Axis2 構成ファイルをクライアント用に準備します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

始める前に

ヒント : HTTP にサービスをデプロイします。 HTTP 用のクライアントを開発してテストした後、WebSphere MQ transport for SOAP を使用してサービスを参照するように URL を変更します。

このタスクでは、管理対象外の Axis2 クライアントを Java Standard Edition にデプロイする方法を示します。 Web コンテナーに Axis2 クライアントをデプロイすることが可能です。 986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』では、Web コンテナー内にクライアントを開発して、WebSphere Application Server Community Edition にデプロイしました。 サーバー構成の一部として、Axis2 ファセットを有効にして、そのファセットを Web コンテナーの構成に含めました。 他のアプリケーション・サーバー上での Web コンテナーの構成については、Axis2 資料 (http://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_container)、または Web サーバー付属の資料を参照してください。

注 : Axis2 では「サーブレット・コンテナー」という用語が使われます。 サーブレット・コンテナーは Web コンテナーと同じです。

このタスクについて

WebSphere MQ transport for SOAP を使用する Axis2 クライアントのデプロイは、HTTP を使用する Axis2 クライアントのデプロイに似ています。 WebSphere MQ JAR ファイルのクラスパスを提供し、Axis2 構成ファイルを変更するための追加の手順が必要です。 JMS 用の追加のエントリーを Axis2 構成ファイルに含める必要があります。 このエントリーは、JMS `transportSender` を実装する WebSphere MQ transport for SOAP JAR ファイルを参照します。

Axis2 には、クライアント・デプロイメントを簡単にするスクリプト `axis2.bat` または `axis2.sh` が含まれています。1021 ページの図 203 および 1022 ページの図 204 の例を参照してください。

注 :

1. `axis2.bat` には、修正の必要なバグがあります。 ストリング `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` を `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`に変更する必要があります。
2. `axis2.bat` および `axis2.sh` では、すべての Axis2 JAR ファイルを個別にクラスパスに追加する代わりに、素早く一括参照する方法として `-Djava.ext.dirs` が使われています。 ただし、この手法には欠陥があり、一部の JRE でしか機能しません。 IBM JRE ではこれが機能しません。

   JVM パラメーター `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`は、Axis JAR ファイルを JVM で使用できるようにします。 JVM では、いくつかの Axis JAR ファイルのインスタンス化が試行されて、エラーが発生します (その詳細は JVM によって異なります)。 通常は、スタック・トレースに以下のいずれかの行が表示されます。

   ```
   org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
   ```

または org.apache.axis2.deployment.DeploymentException:
java.security.NoSuchAlgorithmException: MD5 MessageDigest not available

管理されない Axis2 クライアントを実行する正しい方法は、Axis2 JAR ファイルをクラスパスに追加することです。 クラスパスは JVM ではなく、クライアント・アプリケーションでのみ使用可能です。

axis2 スクリプトを使用せずに、管理されない Axis2 クライアントを実行するための一般的な手順を示します。 1021 ページの図 201 および 1021 ページの図 202 の例は、Windows および Linux 用のスクリプトです。

手順

1. Axis2 1.4.1 を http://ws.apache.org/axis2/download/1_4_1/download.cgi からダウンロードして、Axis2-1.4.1 フォルダーの中に解凍します。
2. Axis2-1.4.1\conf で axis2.xml を更新します。
   a) Axis2-1.4.1\conf で axis2.xml を更新します。 次のように、transportSender として WebSphere MQ transport for SOAP を追加します。

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) 必要に応じて、接続プールのサイズをデフォルトの 10 から変更します。

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   <parameter name="ResourcePoolCapacity">20</parameter>
   </transportSender>
   ```

   ResourcePoolCapacity は、キャッシュに保持されるサービス・エンドポイント・エントリーの数を定義します。 値は少なくとも 1 でなければなりません。 サービス・エンドポイント・エントリーの数がキャッシュ・サイズを超えると、新規エントリー用のスペースを確保するためにエントリーが削除されます。 エンドポイント・エントリーのサイズはさまざまに異なります。 キャッシュ・スラッシングを防ぐために、十分に大きな数を設定してください。

   986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』のステップ 3 を参照してください。
3. ディレクトリー *deployDir* を作成します。 このディレクトリーの下に、クライアントおよびクライアント・プロキシーを含むフォルダー構造をコピーします。 *deployDir* は、 Eclipse Java プロジェクトの *project*\bin フォルダーに相当します。
4. *deployDir* で、Windows の場合はコマンド・ウィンドウを、UNIX and Linux システムの場合は X Window システムを使用してコマンド・シェルを開きます。
5. クラスパスを更新して、現行ディレクトリーと Axis2 JAR ファイル (com.ibm.mqjms.jar および com.ibm.mq.axis2.jar) を含めます。

   com.ibm.mqjms.jar は、必要な他のすべての WebSphere MQ JAR ファイルを参照します。
6. **Java** コマンドを使ってクライアント・プログラムを開始します。

例

Axis2 クライアントを実行する 4 つの例が、1021 ページの図 202 から 1022 ページの図 204 に示されています。 1021 ページの図 200 は、991 ページの図 183 でリストされた非同期クライアントを実行した出力を示しています。

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

図 *200. SQA2AsyncClient* 実行の出力

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

図 *201. runpojo.bat: Windows*、クラスパスを使用

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

図 *202. runpojo.sh: Linux*、クラスパスを使用

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

図 *203. runaxis2.bat: Windows*、*axis2.bat* を使用

注

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

図 *204. runaxis2.sh: Linux*、*axis2.sh* を使用

注

関連タスク

IBM WebSphere MQ transport for SOAP を使用した Web サービス・クライアントの Axis 1.4 へのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 IBM WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

W3C SOAP over JMS を使用した Axis2 クライアントへのデプロイ
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 4 番目のステップです。 WebSphere MQ transport for SOAP 用に開発された Axis2 クライアントの URL を変更して、SOAP over JMS の W3C 勧告候補を使用するようにします。

WebSphere MQ transport for SOAP を使用するための .NET Framework 1 および 2 への Web サービス・クライアントのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供します。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

W3C SOAP over JMS を使用した *Axis2* クライアントへのデプロイ

SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 4 番目のステップです。 WebSphere MQ transport for SOAP 用に開発された Axis2 クライアントの URL を変更して、SOAP over JMS の W3C 勧告候補を使用するようにします。

始める前に

Axis2 クライアントおよび WebSphere MQ transport for SOAP プロトコルを使用して **SimpleJavaListener** を呼び出すために、986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』のタスクをまず完了する必要があります。

さらに、Web サービスを既に作成し、以下に示すこれまでのタスクで WebSphere MQ および WebSphere Application Server が構成済みでなければなりません。

1. 1012 ページの『WebSphere MQ リソースの構成』.
2. 1012 ページの『WebSphere Application Server リソースの構成』.
3. 972 ページの『W3C SOAP over JMS 用の JAX-WS EJB Web サービスの開発』.

このタスクでは、クライアントは Eclipse Galileo で実行されます。 Axis2 に付属の `Axis2.bat` ファイルを変更することにより、コマンド行からクライアントを実行できます。

このタスクについて

WebSphere Application Server によって提供される StockQuoteAxis サービスを呼び出すために既存の Axis2 `StockQuoteAxis` 静的クライアントで必要な変更点は、クライアントに渡される URL を変更することだけです。 WSDL は変更されていないため、`soap.server` パッケージ内で同じプロキシー・クラスを使用できます。

クライアントに渡される URL を定義するには、2 つの方法があります。 生成された `StockQuoteAxis.wsdl` と同じ URL を使用できます。 WebSphere Application Server JNDI ディレクト

リーにアクセスするための jndiInitialContextFactory および jndiURL パラメーターを追加する必要があります。 別の方法は、URL を変更して、JNDI 検索を使わずに QM1 上の REQUESTAXIS および REPLYAXIS キューへの直接アクセスをクライアントに与えることです。

Axis2 クライアントに渡される URL で定義される接続パラメーターは、SOAP メッセージの送受信に必要な WebSphere MQ キュー・マネージャーおよびキューへの接続に使われます。 Axis2 クライアントに渡される接続パラメーターは、必ずしもサービスによって使われるとは限りません。 WebSphere MQ の分散キューイング機能を使用すると、同じキュー・マネージャー (または同じネーム・サーバー) を使用しないようにクライアントとサービスを分離することができます。

手順

1. 生成された StockQuoteAxis.wsdl からの URL を保存して、メモリー節約のために Rational Application Developer を閉じます。

   サーバー構成を変更しなかった場合、Rational Application Developer を閉じるとアプリケーション・サーバーが停止します。 その場合は、以下のコマンドを使ってサーバーを開始してください。

   ```
   startserver server1
   ```

2. Eclipse Galileo を Axis2 クライアント・プロジェクトと共にワークスペースで開きます。
3. SQA2StaticClient.java を開きます。

   SQA2StaticClient.java を参照してください。
4. URI の queue バリアントを使ってサービスを呼び出します。

   a) URL を変更します。

   新しい URI は次のとおりです。

   ```
   jms:queue:REQUESTAXIS
             ?replyToName=REPLYAXIS
             &connectionFactory=connectQueueManager(QM1)Bind(Server)
             &targetService=StockQuoteAxis;
   ```

   これを、StockQuoteAxis.wsdl からの URL と比較してください。

   ```
   jms:jndi:requestaxis
            ?jndiConnectionFactoryName=qm1
            &targetService=StockQuoteAxis
   ```

   図 *205. StockQuoteAxis.wsdl からの URL*

   - ここでは REQUESTAXIS は JNDI 名ではなくキュー名であるため、大文字です。
   - QM1 への接続は直接接続です。
   - URI には、応答先の宛先名が含まれていません。 クライアントは、想定する応答キューを定義する必要があります。

   b) 同じ 「**実行 ...**」 を使用して SQA2StaticClient.java を実行します。 構成については、 986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』を参照してください。
5. WebSphere Application Server をネーミング・サーバーとして使用して、URI の jndi バリアントを使ってサービスを呼び出します。

   a) StockQuoteAxis.wsdl からの URL を使用します (1023 ページの図 205)。 WebSphere Application Server でネーミング・サービスを使用するために欠落しているパラメーターを指定します。

   指定する必要のある欠落パラメーターとその値は、次のとおりです。

表 151. 追加の *JNDI* パラメーター

パラメーター	この例で使用される値	説明
&jndiURL	iiop://localhost:2810 または corbaname:iiop:localhost:2810	ネーミング・プロバイダーの URI。 WebSphere Application Server の場合、値のデフォルトは 2809 です。 これは RMI コネクターのポート番号、およびブートストラップ・ポートとも呼ばれます。 値は SystemOut.log にリストされます。 `00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810`
&jndiInitialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory	WebSphere Application Server によって使用される初期コンテキスト・ファクトリーの名前。
&replyToName	replyaxis	REPLYAXIS キューの JNDI 名。

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) JNDI 検索で必要な JAR ファイルを追加します。

この構成では、JMS url の jndi バリアントを使ってタスクを実行するために、ビルド・パスに以下の JAR ファイルが追加されました。

- *Rational install directory*\SDP\runtimes\base_v7\runtimes の com.ibm.jaxws.thinclient_7.0.0.jar。
- *Rational install directory*\SDP\runtimes\base_v7\plugins からの com.ibm.ws.runtime.jar

別の JNDI プロバイダーの場合、異なる JAR ファイルが必要です。

ビルド・パス内のその他の JAR ファイルは次のとおりです。

i) *WebSphere MQ Install directory*\java\lib 内のすべての JAR ファイル。

ii) *Axis2-1.5.1*\lib 内のすべての JAR ファイル。

iii) Java 6.0 JRE。

c) 同じ「**実行 ...**」を使用して SQA2StaticClient.java を実行します。 構成については、986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』を参照してください。

タスクの結果

どちらの場合も、サービスからの応答がクライアント・コンソール・ビューに表示されます。

関連タスク

IBM WebSphere MQ transport for SOAP を使用した Web サービス・クライアントの Axis 1.4 へのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 IBM WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

WebSphere MQ transport for SOAP を使用するための Axis2 への Web サービス・クライアントのデプロイ
デプロイメント・ディレクトリーと Axis2 構成ファイルをクライアント用に準備します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

WebSphere MQ transport for SOAP を使用するための .NET Framework 1 および 2 への Web サービス・クライアントのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供します。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

WebSphere MQ transport for SOAP を使用するための *.NET Framework 1* および *2* への *Web* サービス・クライアントのデプロイ

クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供します。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

始める前に

ヒント: Visual Studio を使用してサービスとクライアントを開発し、テストします。 その後、WebSphere MQ transport for SOAP 用にクライアントを変更します。

1. .NET Framework 1 または 2 を使用してサービスをデプロイする場合は、ライブラリー (`.dll`) としてサービスをビルドします。 WebSphere MQ transport for SOAP を使用してデプロイします。
2. クライアントに Register.Extension() 呼び出しを追加します。
3. *`MQ_Install`*`\bin` にある amqsoap.dll への参照を追加します。
4. WebSphere MQ transport for SOAP 用に、クライアント・プロキシー・クラス・コンストラクターの静的 `Url` プロパティーを `jms:/` URI に変更します。

このタスクについて

WebSphere MQ transport for SOAP を使用するために .NET Framework 1 または 2 用の Web サービス・クライアントをデプロイするには、追加のデプロイメント手順が必要です。 `amqsoap.dll` を .NET Framework に登録する必要があります。 `amqsoap.dll` は、 WebSphere MQ transport for SOAP のインストールの一部として自動的に登録されますが、再度登録する必要がある場合があります。

コマンド **amqwdeployWMQService** を使用してクライアント・プロキシーを生成する場合、このコマンドが生成したディレクトリーを使ってクライアントをデプロイできます。

手順

1. クライアントのデプロイメント・ファイルを保持するためのディレクトリー *deployDir* を作成します。
2. *deployDir* でコマンド・ウィンドウを開きます。
3. **amqwsetcp** を実行して CLASSPATH を設定します (Axis 1.4 でサービスが実行される場合)。
4. 必要に応じて **amqwRegisterDotNet** を実行して、`amqsoap.dll` を .NET Framework に登録します。

例

この例は、.NET Framework V2 クライアントからの構成と出力 (1026 ページの図 208) を示しています。 クライアント (1026 ページの図 207) は、入力パラメーターをエコー出力する Web サービスを呼び出します。 静的 Url 定義 (1026 ページの図 206) は、クライアント・プロキシーのコンストラクターを示しています。

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
               &connectionFactory=(connectQueueManager(QM1)binding(server))
               &initialContextFactory=com.ibm.mq.jms.Nojndi
               &targetService=Quote.asmx
               &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

図 *206.* 静的クライアント・プロキシー・コンストラクター

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

図 *207.* クライアント・プログラム

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

図 *208.* 構成および出力

次のタスク

1. クライアントを WebSphere MQ MQI クライアントとしてデプロイする場合は、クライアントとサーバーの接続チャネルを構成します。
2. クライアントをサービスへの別のキュー・マネージャーにデプロイする場合は、宛先キューをクライアントで使用可能にする必要があります。 サービス・キュー・マネージャー上の宛先キューをクラスター・キューとして構成するか、クライアント・キューマネージャー上でリモート・キュー定義として構成します。

関連タスク

IBM WebSphere MQ transport for SOAP を使用した Web サービス・クライアントの Axis 1.4 へのデプロイ
クライアント用のデプロイメント・ディレクトリーとデプロイメント記述子を作成します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 IBM WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

WebSphere MQ transport for SOAP を使用するための Axis2 への Web サービス・クライアントのデプロイ
デプロイメント・ディレクトリーと Axis2 構成ファイルをクライアント用に準備します。 クライアント・プロキシーとクライアント・クラスを提供し、CLASSPATH をセットアップします。 WebSphere MQ キューおよびチャネルを構成し、サービスを開始してクライアントをテストします。

W3C SOAP over JMS を使用した Axis2 クライアントへのデプロイ
SOAP over JMS の W3C 勧告候補にバインドされた Web サービスは、Java EE アプリケーション・サーバーの EJB コンテナー内で稼働する必要があります。 このタスクは、W3C SOAP over JMS プロトコルを使用して、WebSphere Application Server にデプロイされた Web サービスと Axis2 Web サービス・クライアントを接続するタスクの 4 番目のステップです。 WebSphere MQ transport for SOAP 用に開発された Axis2 クライアントの URL を変更して、SOAP over JMS の W3C 勧告候補を使用するようにします。

W3C SOAP over JMS および WebSphere Application Server を使用した JAX-WS サービスへの Axis2 クライアントの接続

このタスクを完了すると、WebSphere Application Server で実行される JAX-WS Web サービスを Axis2 クライアントから呼び出した状態になります。 Axis2 クライアントおよび WebSphere Application Server は、WebSphere MQ で稼働する SOAP over JMS プロトコルの W3C 勧告候補を使用します。 Eclipse Galileo および Rational Application Developer を使用して、Web サービス・クライアントと Web サービスをそれぞれビルドします。

始める前に

このタスクでは、バージョン 7 の Rational Software Development Environment および WebSphere Application Server が必要です。 このタスクは、 Rational Software Architect for WebSphere Software v7.5.5.1、および WebSphere Application Server バージョン 7.0 Test Environment v7.0.0.9 Update 1 に同梱されている Rational Application Developer を使用して作成されました。 WebSphere MQ v7.0.1.3 も必要です。

このタスクは、他の 2 つのタスク (965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』、および 986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』) の上に作成されています。 これらのタスクを完了するために、開発環境には既に Eclipse Galileo、WASCE、WASCE 用 Eclipse プラグイン、および Axis2 1.4.1 がインストール済みです。 このタスクでは WASCE は必要ありません。

一部のステップは複雑です。 このステップでは、 Rational Application Developer を使用する WebSphere Application Server 用の Web サービス・アプリケーションの開発について、ある程度の知識があることを前提としています。 このタスクでは、高性能のプロセッサーと大容量のメモリーが必要とされます。 このタスクは VMWare Windows XP SP3 仮想マシン、メモリー 1.8GB で実行されました。

タスクを始める前に、すべてのソフトウェアをインストールしてください。 処理能力に応じて、ソフトウェアのダウンロードに約 1 日、さらにインストールに約 1 日かかります。 タスクの実行には少なくとも半日かかります。

このタスクについて

このタスクのシナリオは次のとおりです。 オープン・ソースのツール Eclipse Galileo を使って株価情報の Web サービス `StockQuoteAxis` を開発しました。 `StockQuoteAxis` は、オープン・ソース・サーバーの WASCE で実行している SOAP over HTTP を使用してデプロイされます。

デプロイされる Web サービスを、SOAP over HTTP に加えて、標準に基づくメッセージング・トランスポート (例えば SOAP over JMS)、または Web サービス高信頼性メッセージングにバインドします。 クライアントとサービスの両方で標準に基づくインターフェースを使用します。 このため、将来のプロジェクトの開発チームが WebSphere MQ transport for SOAP を使用してソリューションを実装しましたが、まだ実動には至っていません。

WebSphere MQ transport for SOAP 用の SOAP クライアントを HTTP クライアントから変更する必要があるという問題は、Axis2 クライアントによって取り除かれました。 IBM WebSphere MQ transport for SOAP によって接続されるサービスのホストが、WebSphere MQ で提供される特殊なリスナー `SimpleJavaListener` であるという問題がまだ残っています。

W3C の SOAP over JMS 標準のステータスが勧告候補になり、一部のベンダーは W3C SOAP over JMS のサポートを提供しています。 このサポートを使用すると、Web サービスをアプリケーション・サーバーにデプロイし、さまざまな接続プロトコルを使用して同じサービスに接続できます。 WebSphere Application Server v7 で提供されるサポートによって、メッセージ・ベースの SOAP トランスポートを使用するために別個の Web サービスをホストする必要があるという問題が取り除かれます。 標準に基づくメッセージ・トランスポート・インターフェース JMS を使用する場合、さまざまなベンダーのツールを使ってソリューションを開発できます。 将来的には、Eclipse の Web サービス・ツールに SOAP over JMS バインディングが含まれることが期待されます。

ほとんどのステップは、Eclipse または WebSphere 製品に付属の管理ツールを使って実行されます。 ステップは、Windows 環境を想定して説明されています。 一部のコマンドをわずかに変更するだけで、他のプラットフォームでもステップを実行できます。

準備ステップとして、HTTP Web サービスの作成、および Axis2 を使用した接続がリストされています。これらのステップのクライアントと WSDL を使用して、ソリューションが作成されます。

手順

1. Axis2 クライアントおよび IBM WebSphere MQ transport for SOAP を使用して StockQuoteAxis Web サービスに接続します
   a) 965 ページの『WebSphere MQ transport for SOAP 対応 JAX-RPC サービスを Eclipse を使用して作成する』
   b) 986 ページの『Eclipse を使用して WebSphere transport for SOAP 用の JAX-WS クライアントを開発する』
   c) 1019 ページの『WebSphere MQ transport for SOAP を使用するための Axis2 への Web サービス・クライアントのデプロイ』
2. Axis2 クライアントおよび W3C 勧告候補 SOAP over JMS を使用して StockQuoteAxis Web サービスに接続します。
   a) 1012 ページの『WebSphere MQ リソースの構成』
   b) 1012 ページの『WebSphere Application Server リソースの構成』
   c) 972 ページの『W3C SOAP over JMS 用の JAX-WS EJB Web サービスの開発』
   d) 1022 ページの『W3C SOAP over JMS を使用した Axis2 クライアントへのデプロイ』

WebSphere MQ bridge for HTTP

WebSphere MQ bridge for HTTP を使用すると、クライアント・アプリケーションは WebSphere MQ MQI クライアントをインストールしなくても WebSphere MQ とメッセージを交換できます。 HTTP 機能を使って任意のプラットフォームまたは言語から WebSphere MQ を呼び出すことができます。

WebSphere MQ bridge for HTTP の概要

WebSphere MQ Bridge for HTTP は、Java、Enterprise Environment (JEE) Web アプリケーションです。HTTP クライアントはこのアプリケーションに **POST**、**GET**、および **DELETE** 要求を送信することにより、WebSphere MQ キューに対してメッセージの書き込み、参照、および削除を行えます。 保証配信が必要な場合、WebSphere MQ bridge for HTTP は、メッセージとともに使用するのに適しません。

利点

WebSphere MQ bridge for HTTP を使用すると、HTTP を使用して、広範な種類の環境との間で WebSphere MQ メッセージの送受信ができます。

- HTTP をサポートするが、WebSphere MQ をサポートしていない環境。
- WebSphere MQ MQI クライアントをインストールするための十分なストレージ・スペースがない環境。
- WebSphere MQ へのアクセスを必要とする各システムに WebSphere MQ MQI クライアントをインストールするには、環境の数が膨大である場合。
- Web ベース・アプリケーションで、WebSphere MQ への独自のブリッジをコーディングせずにメッセージの送受信を行いたい場合。
- Ajax などの非同期技法を使用して Web ベース・アプリケーションを拡張しようとしている場合。WebSphere MQ bridge for HTTP は、Representation State Transfer (REST) over HTTP を使用して、WebSphere MQ のキューとトピックを使用可能にします。

HTTP サポートは、Point-to-Point および パブリッシュ/サブスクライブ・メッセージング・トポロジーのどちらでも使用できます。

HTTP サポートの動作方法

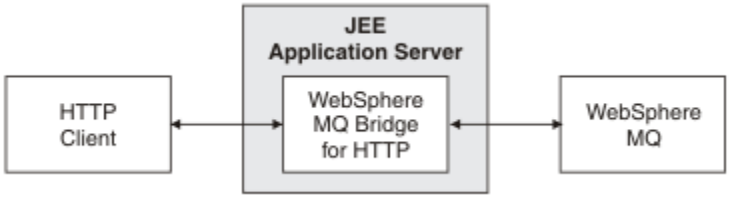


図 *209. WebSphere MQ bridge for HTTP*

WebSphere MQ bridge for HTTP Web アプリケーションは、1 つ以上のクライアントから http 要求を受信します。 そして、WebSphere MQ との対話を代行し、クライアントへ HTTP 応答を返します。

WebSphere MQ bridge for HTTP は、リソース・アダプターを使用して WebSphere MQ に接続する JEE サーブレットです。 この HTTP サーブレットは 3 つの異なるタイプの HTTP 要求 **POST**、**GET**、および **DELETE** を処理します。

表 *152. WebSphere MQ bridge for HTTP の verb*

HTTP リクエスト	**結果**
POST	メッセージをキューまたはトピックに書き込みます。
GET	キュー上の最初のメッセージをブラウズします。 HTTP プロトコルの仕様どおり、**GET** ではキューからメッセージを削除しません。 パブリッシュ/サブスクライブ・メッセージングでは、**GET** を使用しないでください。
削除	キューまたはトピックからメッセージを取得して削除します。

HTTP POST の例

HTTP **POST** はキューにメッセージを書き込むか、またはトピックにパブリケーションを書き込みます。 **HTTPPOST** Java サンプルは、キューへのメッセージを HTTP **POST** 要求で送るサンプルです。 Java を使用する代わりに、ブラウザーのフォームや AJAX ツールキットを使用して HTTP **POST** 要求を作成することもできます。

1029 ページの図 210 に、myQueue というキューにメッセージを書き込むための HTTP 要求を示します。 この要求には HTTP ヘッダー `x-msg-correlId` が含まれていて、これにより、WebSphere MQ メッセージの相関 ID が設定されます。

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

図 *210.* キューに対する *HTTP **POST*** 要求の例

1030 ページの図 211 に、クライアントに返信される応答を示します。 応答のコンテンツはありません。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

図 *211. HTTP POST* 応答の例

HTTP DELETE の例

HTTP **DELETE** では、キューからメッセージを取得してそのメッセージを削除するか、またはパブリケーションを取得して削除します。 **HTTPDELETE** Java サンプルは、HTTP **DELETE** 要求で、キューからのメッセージを読み取るサンプルです。 Java を使用する代わりに、ブラウザーのフォームや AJAX ツールキットを使用して HTTP **DELETE** 要求を作成することもできます。

1030 ページの図 212 に、myQueue というキュー上の次のメッセージを削除するための HTTP 要求を示します。 応答では、メッセージ本文がクライアントに返されます。 WebSphere MQ の用語では、HTTP **DELETE** は破壊読み取りです。

この要求には、HTTP 要求ヘッダー `x-msg-wait` が含まれていて、これが WebSphere MQ bridge for HTTP に対して、メッセージがキューに到着するのを待機する時間の長さを指示します。 この要求には `x-msg-require-headers` 要求ヘッダーも含まれていて、これはクライアントが応答でメッセージ相関 ID を受け取ることを指定しています。

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

図 *212. HTTP **DELETE*** 要求の例

1030 ページの図 213 は、クライアントに返される応答です。 相関 ID は、要求の `x-msg-require-headers` で要求されると、クライアントに返されます。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

図 *213. HTTP **DELETE*** 応答の例

HTTP GET の例

HTTP **GET** はキューからメッセージを取得します。 メッセージはキューに置かれたままになります。 WebSphere MQ の用語では、HTTP **GET** は表示要求です。 HTTP **GET** 要求は、Java クライアント、ブラウザーのフォーム、または AJAX ツールキットを使用して作成できます。

1031 ページの図 214 に、myQueue というキュー上の次のメッセージを表示するための HTTP 要求を示します。

この要求には、HTTP 要求ヘッダー `x-msg-wait` が含まれていて、これが WebSphere MQ bridge for HTTP に対して、メッセージがキューに到着するのを待機する時間の長さを指示します。 この要求には `x-msg-`

require-headers 要求ヘッダーも含まれていて、これはクライアントが応答でメッセージ相関 ID を受け取ることを指定しています。

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

図 *214. HTTP **GET*** 要求の例

1031 ページの図 215 は、クライアントに返される応答です。 相関 ID は、要求の x-msg-require-headers で要求されると、クライアントに返されます。

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

図 *215. HTTP **GET*** 応答の例

WebSphere MQ Bridge for HTTP のインストール、構成、および検証

WebSphere MQ bridge for HTTP を入手するには、 WebSphere MQ MQI クライアントまたはサーバーのインストール資料から "Java メッセージングおよび Web サービス" をインストールします。 WebSphere MQ Bridge for HTTP を適切なアプリケーション・サーバーにデプロイします。

始める前に

IBM WebSphere MQ のシステム要件を参照して、前提条件となる製品を確認します。 インストール・プロセスでは、WebSphere MQ Bridge for HTTP を実行するための前提ソフトウェアの有無および使用可能であるかどうかについての検査は行われません。 前提条件ソフトウェアがインストールされていることを検証する必要があります。

WebSphere MQ Bridge for HTTP は、WebSphere MQ リソース・アダプターをインストールすることにより、任意の Java EE 1.4 準拠アプリケーション・サーバー上で稼働します。 WebSphere MQ Bridge for HTTP は、WebSphere Application Server のバージョン 6.0.2.1 より前のリリースでも稼働させることができます。 WebSphere MQ を JMS プロバイダーとして組み込むには、WebSphere Application Server Message Listener Port (MLP) を使用してください。

WebSphere MQ bridge for HTTP のサポートは、以下のアプリケーション・サーバーに対してのみ提供されます。

- WebSphere Application Server 6.0.2.1 以降。
- WebSphere Application Server Community Edition バージョン 1.1 以降。

このタスクについて

WebSphere MQ Bridge for HTTP は、 .war ファイル WMQHTTP.war として提供されます。

- UNIX プラットフォームおよび Linux の場合:
  - WMQHTTP.war は、 "「Java メッセージングおよび Web サービス」" インストール・オプションの一部として組み込まれています。 このオプションは、クライアントとサーバー両方のインストール・マテリアルで選択可能です。

- WMQHTTP.war は *<mqmtop>*/java/http/WMQHTTP.war にインストールされます。 *<mqmtop>* は、 WebSphere MQ がインストールされているディレクトリーです。
- WMQHTTP.samples は *<mqmtop>*/java/http/samples にインストールされます。 *<mqmtop>* は、 WebSphere MQ がインストールされているディレクトリーです。

WebSphere MQ Bridge for HTTP のインストール、デプロイ、構成、および検証を行うには、以下のインストール・ステップを実行します。 構成ステップの詳細は、アプリケーション・サーバーによって異なります。 実際のアプリケーション・サーバーで必要なステップのためのテンプレートとして、1032 ページの『WebSphere MQ Bridge for HTTP の WebSphere Application Server V6.1.0.9 へのデプロイと検証』を使用してください。

手順

1. WebSphere MQ MQI クライアントまたはサーバーをインストールして、WMQHTTP.war を取得します。
2. WMQHTTP.war を、アプリケーション・サーバーへのデプロイが可能なサーバーにコピーします。
3. WMQHTTP.war をアプリケーション・サーバーにデプロイします。
4. 必要に応じて、WebSphere MQ をリソース・アダプターとしてアプリケーション・サーバーにインストールします。

   使用しているアプリケーション・サーバーで、WebSphere MQ が既にメッセージング・プロバイダーとして構成されているかどうかを調べます。 使用しているアプリケーション・サーバーで提供されている管理ツールを使用して、WebSphere MQ を検索します。 WebSphere MQ は、パス「**リソース**」>「**JMS**」>「**メッセージング・プロバイダー**」の下にあります。
5. WebSphere MQ MQI クライアント・トランスポートを使用するキュー・マネージャーに接続するように、アプリケーション・サーバー上の接続ファクトリーを構成します。[12].
6. アプリケーション・サーバーで、WMQHTTP.war Web アプリケーションが接続ファクトリーを使用するように構成します。
7. 設定を確認してください。
   a) 接続ファクトリー内の名前付きキュー・マネージャーとローカル・キューをセットアップします。
   b) ローカル・キュー上にメッセージを配置します。
   c) 接続ファクトリー内で名前付きサーバー接続チャネルを作成し、ローカル・キューへの読み取り/書き込み権限を与えます。
   d) キュー・マネージャーとリスナーを始動します。
   e) 始動していない場合は、アプリケーション・サーバーと WMQHTTP.war を始動します。
   f) ブラウザーを開いて、次のように入力します。http://*hostname*:*web port*/*Context root*/msg/queue/*local queue*

タスクの結果

ブラウザーのウィンドウに、ローカル・キューに配置したメッセージが表示されます。

次のタスク

1. 1032 ページの『WebSphere MQ Bridge for HTTP の WebSphere Application Server V6.1.0.9 へのデプロイと検証』にあるサンプルを試行します。
2. サンプル HTTP Java アプリケーションを実行します。

WebSphere MQ Bridge for HTTP の WebSphere Application Server V6.1.0.9 へのデプロイと検証

以下の例を使用して、サンプル HTTP Java プログラムを実行するために、WebSphere MQ Bridge for HTTP のデプロイメントを準備します。 ここでは、WebSphere Application Server V6.1.0.9 にデプロイします。

[12] 最初は、少なくともクライアント・トランスポートを構成します。 一部のアプリケーション・サーバーは、直接接続またはバインディング・モード接続を使用して WebSphere MQ に接続できます。

始める前に

1. 1031 ページの『WebSphere MQ Bridge for HTTP のインストール、構成、および検証』に書かれた説明に従って、WMQHTTP.war を WebSphere Application Server のインストール済み環境にアクセス可能なサーバーにコピーします。
2. キュー・マネージャーとキューを使用できるように構成して、構成をテストします。
   - サンプルでは、1033 ページの表 153 に記載した値を使用してキュー・マネージャーを構成しています。

表 *153.* キュー・マネージャー構成	
オブジェクト	**値**
ホスト名	itso-01
キュー・マネージャー	QM1
ローカル・キュー	HTTPTESTQ
サーバー接続チャネル	MYSVRCON HTTPTESTQ に対して読み取りと書き込みを行うための十分な権限を持つ MCA ユーザー ID を構成します。
リスナー・ポート	1414

3. キュー・マネージャーとリスナーを始動します。
4. HTTPTESTQ 上にテスト・メッセージを書き込みます。 以下に例を示します。
   a. WebSphere MQ エクスプローラーを始動します。
   b. QM1 のローカル・キューのリストで「**HTTPTESTQ**」を右クリック >「**テスト・メッセージの書き込み**」>「**First Message**」と入力 >「**メッセージの書き込み**」>「**クローズ**」と操作します。
5. アプリケーション・サーバーを始動して、Integrated Solutions Console にサインオンします。

このタスクについて

このサンプルでは、アプリケーション・サーバーとして WebSphere Application Server V6.1.0.9 を実行している場合に実行するステップを示しました。 別のバージョンの WebSphere Application Server を実行している場合、および別のアプリケーション・サーバーを実行している場合、手順は異なります。 WebSphere Application Server V6.1.0.9 では、WebSphere MQ MQI クライアント・ライブラリーを使用して、メッセージング・プロバイダーとして WebSphere MQ がインストールされて、事前構成されています。 WebSphere MQ がメッセージング・プロバイダーとして事前構成されていない場合、または WebSphere MQ サーバー・バインディングを使用する場合は、アプリケーション・サーバーに JEE 用 WebSphere MQ リソース・アダプターをインストールして構成する必要があります。

以下の説明に従って、WebSphere MQ Bridge for HTTP を WebSphere Application Server V6.1.0.9 にデプロイし、ブラウザーを使用してデプロイの検証を行います。

手順

1. ナビゲーション・ペインで、 「**リソース**」>「**JMS プロバイダー**」>「 **WebSphere MQ メッセージング・プロバイダー**」をクリックします。

   WebSphere Application Server のデプロイメントに応じて、ノード、セル、またはサーバーのいずれかのレベルに構成できます。 このサンプルでは、サーバー・レベルのデプロイメントを使用します。
2. 「**追加プロパティー**」の下で、「**接続ファクトリー**」>「**新規**」とクリックします。
3. JMS プロバイダー・フォームで、 1034 ページの表 154 の情報、または選択した代替手段を指定し、「**適用**」>「**保存**」をクリックします。

表 *154.* 設定または変更するフィールド	
フィールド	**値**
名前	WMQHTTPBridge
JNDI 名	jms/WMQHTTPJCAConnectionFactory
キュー・マネージャー	QM1
ホスト	itso-01
ポート	1414
チャネル	MYSVRCON
トランスポート・タイプ	CLIENT

4. ナビゲーション・ペインで、「**アプリケーション**」>「**新規アプリケーションのインストール**」をクリックします。
5. フォームに WMQHTTP.war のパス、およびコンテキスト・ルートを入力し、「**次へ**」をクリックします。
   a) コンテキスト・ルートはオプションです。 サンプル HTTP アプリケーションでは、mq がデフォルトのコンテキスト・ルートです。
   b) コンテキスト・ルートは、WebSphere MQ Bridge for HTTP を識別するための URI の一部となります。 コンテキスト・ルートは省略することができ、また、後で変更することもできます。
6. インストール・ウィザードの「**インストール・オプションの選択**」ページでは、デフォルトのまま何も変更する必要はありません。「**次へ**」をクリックします。
7. 「**モジュールをサーバーにマップ**」 ページで、クラスターまたはサーバーを選択し、「選択」ボックスにチェック・マークを付けて、 「**適用**」>「**次へ**」をクリックします。
8. 「**リソース参照をリソースにマップ**」 ページの **javax.jms.ConnectionFactory** フォームで、「**参照 ...**」をクリックします。 IBM WebSphere MQ Bridge for HTTP の行に追加できます。
9. 「**エンタープライズ・アプリケーション > 使用可能リソース**」ページで、「**WMQHTTPBridge**」を選択して、「**適用**」をクリックします。
10. 「**javax.jms.ConnectionFactory**」フォームに戻って、認証方式を選択します。
    a) 例えば、「**なし**」を選択して、「**適用**」をクリックします。 その他のオプションは追加の構成をする場合に必要になります。
11. IBM WebSphere MQ Bridge for HTTP の「**選択**」チェック・ボックスにチェック・マークを付け、「**次へ**」>「**次へ**」>「**終了**」>「**保存**」 をクリックします。
12. ナビゲーション・ペインで、「**アプリケーション**」>「**エンタープライズ・アプリケーション**」をクリックします。
13. WMQHTTP.war の選択ボックスにチェック・マークを付けて、「**始動**」をクリックします。
14. ブラウザー・ウィンドウを開きます。 http://itso-01:9080/mq/msg/queue/HTTPTESTQ と入力します。ここで、ホスト名とポートには該当するものを使用してください。

タスクの結果

正しく構成されていれば、ブラウザー・ウィンドウに「First Message」と表示されます。

次のタスク

サンプル HTTP Java アプリケーションを実行します。

WebSphere MQ Bridge for HTTP を使用したパブリッシュ/サブスクライブ

WebSphere MQ Bridge for HTTP は、WebSphere MQ classes for JMS パブリッシュ/サブスクライブ・インターフェースを使用します。 HTTP **POST** でパブリケーションを作成します。 HTTP **DELETE** で非永続管理

サブスクリプションを作成します。 トピック URI を使用する前に、JMS 用パブリッシュ/サブスクライブを構成しなければなりません。

パブリッシュ/サブスクライブは、バージョン 7 で WebSphere MQ に完全に統合されています。 バージョン 7 より前は、別のパブリッシュ/サブスクライブ・ブローカーがパブリケーションとサブスクリプションを処理していました。 これは、バージョン 7 で完全に統合されたパブリッシュ/サブスクライブと区別するために、 "キュー型" パブリッシュ/サブスクライブと呼ばれます。 バージョン 7 は、統合パブリッシュ/サブスクライブを使用して、キューに入れられたパブリッシュ・サブスクライブをエミュレートします。 このエミュレーションにより、既存のキュー型パブリッシュ/サブスクライブを利用するアプリケーションと、統合型を利用するアプリケーションとを、同じキュー・マネージャー上に共存させて実行することができます。 キュー型パブリッシュ/サブスクライブ・アプリケーションは、統合型アプリケーションと相互運用して、同じトピックを共有することもできます。 バージョン 6 ではブローカーは WebSphere MQ に添付されていました。バージョン 6 より前はサポート・パックにより利用可能でした。

構成

WebSphere MQ Bridge for HTTP は、パブリッシュとサブスクライブに JMS インターフェースを使用します。 バージョン 7 では、PROVIDERVERSION JMS プロパティーを使用することにより、WebSphere MQ classes for JMS でキュー型と統合型のどちらのパブリッシュ/サブスクライブを使用するかを制御できます。

さらに考慮できるのは、WebSphere MQ MQI クライアント・ライブラリーと共に WebSphere MQ Bridge for HTTP を使用するか、あるいはサーバー・ライブラリーと共に使用するかという点です。 バージョン 6 クライアント・ライブラリーでは、キュー型パブリッシュ/サブスクライブのみをサポートしていましたが、バージョン 7 ライブラリーではキュー型と統合型の両方のパブリッシュ/サブスクライブをサポートしています。 WebSphere MQ をメッセージング・プロバイダーとして使用する多くの Web サーバーまたはアプリケーション・サーバーでは、クライアント・ライブラリーを使用してそれを行います。 統合パブリッシュ/サブスクライブを使用するには、 WebSphere MQ MQI クライアント・ライブラリーとサーバー・ライブラリーの両方が少なくともバージョン 7 でなければなりません。 いずれかが 7 より前のバージョンの WebSphere を実行している場合は、キューに入れられたパブリッシュ/サブスクライブを構成する必要があります。 1035 ページの表 155 を参照してください。 使用している Web サーバーまたはアプリケーション・サーバーでインストールまたは構成してあるライブラリーがどちらであるかを確認します。

表 *155.* パブリッシュ/サブスクライブ構成モード

	クライアント V6 以前	**クライアント V7 以降**
サーバー V6 以前	1. \java\bin\MQJMS_PSQ.mqsc スクリプトを実行します。	サポート対象外
サーバー V7 以降	1. \java\bin\MQJMS_PSQ.mqsc スクリプトを実行します。 2. キュー・マネージャーを PSMODE=ENABLED に設定します。	1. PROVIDERVERSION = 7 の場合 a. キュー・マネージャーを PSMODE=ENABLED または PSMODE=COMPAT に設定します。 2. PROVIDERVERSION = 6 の場合 a. キュー・マネージャーを PSMODE=ENABLED に設定します。

パブリッシュ

次の URI に HTTP **POST** 要求を送信します。

```
http://hostname:port/context_root/msg/topic/topicString
```

メッセージ・コンテンツはトピック・ストリング *topicString* を使用してパブリッシュされます。

サブスクライブ

次の URI に HTTP **DELETE** 要求を送信します。

```
http://hostname:port/context_root/msg/topic/topicString
```

WebSphere MQ Bridge for HTTP は、トピック・ストリング *topicString* への管理非永続サブスクリプションを作成します。 パブリケーションが返ってきたらすぐにサブスクリプションは削除されます。返されなくても、カスタム・エンティティー・ヘッダー `x-msg-wait` で設定された待機インターバルが満了するとすぐに削除されます。

WebSphere MQ bridge for HTTP のサンプルの実行

WebSphere MQ bridge for HTTP サンプルは、Windows オペレーティング・システムのみで使用可能です。このサンプルは、Java プログラムから HTTP **POST** および HTTP **DELETE** コマンドを WebSphere MQ Bridge for HTTP に実行依頼する方法を示しています。

始める前に

1031 ページの『WebSphere MQ Bridge for HTTP のインストール、構成、および検証』の 1032 ページの『7』に書かれた手順を実行して、WebSphere MQ bridge for HTTP のインストール済み環境を検証しておきます。

HTTP サンプルは、1036 ページの表 156 に示すディレクトリーにインストールされています。 各サンプルのソース・コードは `/src` サブディレクトリーにインストールされています。

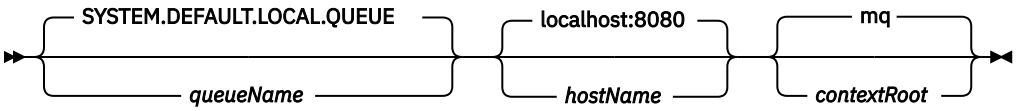
表 *156. HTTP* サンプルの場所	
プラットフォーム	**ロケーション**
Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
その他のプラットフォーム	*MQ_INSTALLATION_PATH*/samp/http
MQ_INSTALLATION_PATH は、WebSphere MQ がインストールされているディレクトリーを表します。	

このタスクについて

サンプルでは、WebSphere MQ AMQSPUT および AMQSGET サンプル・アプリケーションをシミュレートしています。 これらは、Point-to-Point メッセージング環境における以下の関数を例証しています。

- **HTTPPOST** - WebSphere MQ bridge for HTTP を使用して、 WebSphere MQ キューにメッセージを書き込むために Java アプリケーションで HTTP **POST** 要求を送信し、応答を処理します。
- **HTTPDELETE** - WebSphere MQ Bridge for HTTP を使用して WebSphere MQ キューからメッセージを取得するために Java アプリケーションで HTTP **DELETE** 要求を送信し、WebSphere MQ メッセージを含む応答を処理します。

HTTPPOST と HTTPDELETE のパラメーター



HTTPPOST サンプルを実行するには、以下の手順すべてを行います。

手順

1. コマンド・ウィンドウで、HTTP サンプル・ディレクトリーに移動します。

2. **HTTPPOST** サンプルを実行します。

```
java -classpath . HTTPPOST [parameters]
```

HTTPPOST サンプルを開始すると、次の出力が表示されます。

```
HTTP POST Sample start
Target server is 'hostName'
Target queue is 'queueName'
Target context-root is 'contextRoot'
```

3. コマンド・プロンプトに、メッセージ本文にするテキストを入力します。
4. Enter キーを押して、メッセージを WebSphere MQ キューに送ります。
   a) 別のメッセージを送信する場合は、テキストをさらに入力します。
      このテキストは 2 番目の WebSphere MQ メッセージの本体になります。
   b) Enter キーを押して、メッセージを WebSphere MQ キューに送ります。
5. Enter キーを 2 回押すと、**HTTPPOST** が終了します。
   以下の出力が表示されます。

```
HTTP POST Sample end
```

次のタスク

HTTPDELETE サンプルは、WebSphere MQ キューに書きこんだすべてのメッセージに対して破壊読み取りを実行します。

HTTPDELETE サンプルを実行するには、以下の手順すべてを行います。

1. コマンド・ウィンドウで、*MQ_INSTALLATION_PATH*/tools/samples に移動します。
   MQ_INSTALLATION_PATH WebSphere MQ がインストールされているディレクトリーを表します。
2. **HTTPDELETE** サンプルを実行します。

```
java -classpath . HTTPPOST [parameters]
```

HTTPDELETE サンプルを開始すると、次の出力が表示されます。

```
HTTP DELETE Sample start
Target server is 'host:port'
Target queue is 'your queue name'
Target context-root is 'your context-root'
message
message
...
```

WebSphere Bridge for HTTP のセキュリティーに関する考慮事項

標準的な Web セキュリティーに関する考慮事項は、Web ブラウザー・クライアントの認証に適用されます。 WebSphere MQ リソースに対する権限は、WebSphere Bridge for HTTP サーブレットを実行するユーザーのレベルであり、個々の Web ブラウザー・クライアントを実行するユーザーのレベルではありません。 標準的な WebSphere MQ セキュリティーの考慮事項が WebSphere MQ に適用されます。

WebSphere Bridge for HTTP を使用した、Web ブラウザーから WebSphere MQ アプリケーションへのデータ・フローとその戻りのデータ・フローは、以下の 3 つのステップからなります。

クライアント 接続
ブラウザーから WebSphere Bridge for HTTP への HTTP を使用した TCP/IP 接続。

WebSphere MQ へのリソース・アダプター接続
WebSphere Bridge for HTTP から WebSphere MQ キュー・マネージャーへの接続。 この接続は、TCP/IP 上のクライアント接続か、またはローカル WebSphere MQ バインディング接続のどちらかにな

ります。 いったん接続されると、http 要求は標準ローカル・キュー上、または伝送キュー上に置かれます。

1 つ以上のチャネル上の WebSphere MQ ローカル・キューからターゲット・キューへ。
キュー、トピック、キュー・マネージャー、およびチャネルをセキュアにするための標準的な技法が適用されます。

応答では、これらのステップは逆になります。

クライアント 接続

Web コンテナーを使用して、HTTP クライアントとアプリケーション・サーバーの間の接続をセキュアにします。 HTTPS を使用するなど、標準的な HTTP サーバーの技法を使用します。 詳細については、使用しているアプリケーション・サーバーの資料を参照してください。

WebSphere MQ へのリソース・アダプター接続

リソース・アダプターとキュー・マネージャーとの間の接続は、単一のユーザー ID のみを使用して許可されます。 WebSphere Bridge for HTTP からの要求を識別するために、単一のユーザー ID を割り当てます。 このユーザー ID には、外部ユーザーがアクセスする必要のあるリソースのみに対する制限付き WebSphere MQ 権限を与えなければなりません。 Web セキュリティーの標準的技法を使用して、実際のクライアントを別々に認証し、後続するクライアントとの対話のための信頼関係を確立することが必要です。

単一のユーザー ID を使用したリソース・アダプターとキュー・マネージャーとの間の接続をセキュアにします。 このユーザー ID が持つ権限は、キューとトピックに対するメッセージの読み取りと書き込みに必要な権限を超えないように制限します。 WebSphere Bridge for HTTP は、インターネットと使用しているイントラネットとの間のアタック・ポイントになります。

リソース・アダプターと WebSphere MQ との間の接続を保護する方法は、具体的な リソース・アダプターにより異なります。 リソース・アダプターについては、 の資料を参照してください。

Component Object Model インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用

WebSphere MQ Automation Classes for ActiveX (MQAX) は ActiveX コンポーネント群で、 WebSphere MQ にアクセスするためにアプリケーションで使用できるクラスを提供します。

MQAX を使用するには、WebSphere MQ 環境と、その通信先になる WebSphere MQ アプリケーションが必要です。

MQAX を使用すると、WebSphere MQ を介してアクセスできる企業システムであればどんな システムでも、ActiveX アプリケーションでトランザク ションを実行したり、データにアクセスしたりできます。

WebSphere MQ Automation Classes for ActiveX には、次の機能があります。

- WebSphere MQ API の機能にアクセスでき、他の WebSphere MQ プラットフォームとの完全な相互接続性を実現できる。
- ActiveX コンポーネントに求められる標準の規則を順守する。
- WebSphere MQ オブジェクト・モデル (.NET、C++、Java、および LotusScript®でも使用可能) に準拠します。

MQAX には、スターター・サンプルが付属しています。 最初は、これらのサンプルを使用して、MQAX のインストールが成功したこと、WebSphere MQ の基本的な環境が整ったことを確認できます。 このサンプルでは、MQAX の使用例も示しています。

COM と ActiveX のスクリプト記述

コンポーネント・オブジェクト・モデル (COM) は、 Microsoft によって定義されたオブジェクト・ベースのプログラミング・モデルです。 COM は、いくつかのソフトウェア・コンポーネントを、記述 言語や保存場所に関係なく、相互に通信したり、位置を見つけることがで きるような形態で提供する方法を指定します。

ActiveX は、アプリケーションの開発、再使用可能なコンポーネン ト、および Microsoft Windows プラットフォーム上でのインターネット・テ クノロジーを統合した、COM ベースのテクノロジーの集合です。 ActiveX コンポーネントは、アプリケーションが動的にアク セスできるインターフェースを提供します。 ActiveX のスクリプト記述クライアントは、コンパイラーのようなアプリケーションの 1 つで、ActiveX (または COM) コンポーネントが提供するインターフェースを使用するプログラムやス クリプトを作成したり実行したりすることができます。

WebSphere MQ 環境のサポート

WebSphere MQ Automation Classes for ActiveX が動作するには、 **32 ビット**の ActiveX スクリプト記述クライアントが必要です。

COM コンポーネントは、 **32 ビット**のアプリケーションでのみ使用できます。 64 ビットの COM アプリケーションを書き込みたい場合は、.NET インターフェースを使用できます。

WebSphere MQ のサーバー環境で MQAX を実行するには、システムに Windows 2000 以降をインストールする必要があります。

WebSphere MQ MQI のクライアント環境で MQAX を実行するには、Windows 2000 以降をインストールしたシステムで WebSphere MQ MQI クライアントを使用する必要があります。

WebSphere MQ MQI クライアントは、1 つ以上の WebSphere MQ サーバーにアクセスできなければなりません。 WebSphere MQ MQI クライアントおよび WebSphere MQ サーバーの両方がシステムにインストールされている場合、MQAX アプリケーションは常にサーバーに対して実行されます。 MQAI との ActiveX インターフェースは、WebSphere MQ サーバー環境でのみ使用できます。

WebSphere MQ Automation Classes for ActiveX を使用した設計とプログラミング

ActiveX 以外のアプリケーションにアクセスする MQAX アプリケーションの設計

WebSphere MQ Automation Classes は、WebSphere MQ API の機能へのアクセスを提供します。 したがって、WebSphere MQ を使用することにより Windows アプリケーションにもたらされるすべての機能を活用することができます。

アプリケーションの全体的な設計はどの WebSphere MQ アプリケーションでも同じであるため、7 ページの『アプリケーションの開発』セクションで説明している設計上の側面をすべて考慮してください。

WebSphere MQ Automation Classes を使用するには、COM オブジェクトの作成と使用をサポートしている言語を使用して、アプリケーション内に Windows プログラムをコーディングします。 例えば、Visual Basic、Java、およびその他の ActiveX スクリプト・クライアントです。 これによって、必要な WebSphere MQ オブジェクトを使用言語固有の構文でコーディングできるため、 アプリケーションにクラスを簡単に組み入れることができます。

WebSphere MQ Automation Classes for ActiveX の使用法

WebSphere MQ Automation Classes for ActiveX を使用する ActiveX アプリケーションを設計する場合に最も重要な情報は、リモートの WebSphere MQ システムで送受信されるメッセージです。 したがって、メッセージに挿入されている項目の形式を認識する必要があります。 1 つの作業に対する MQAX スクリプトの場合、そのスクリプトとメ ッセージを選択したり送信したりする WebSphere MQ アプリケーションの両方が そのメッセージの構造を認識する必要があります。

MQAX アプリケーションでメッセージを送信する場合、送信先の MQAX 側 でデータ変換を行うときは、さらに次の情報を認識していなければなりませ ん。

- リモート・システムが使用するコード・ページ
- リモート・システムが使用するコード化

コードの移植性を維持するために、コード・ページとエンコードを設定することをお勧めします (送信側システムと受信側システムの両方で現在それらが同じである場合でも)。

設計するシステムの構成方法を考えるとき、MQAX スクリプトは WebSphere MQ のキュー・マネージャーか WebSphere MQ クライアントのどちらかがインストールされているマシンと同じマシン上で実行することを忘れないでください。

プログラミングのヒント

次に示すヒントは、記載順には意味がありません。 行おうとしている作業に該当するヒントに従うと、作業時間を短縮することができます。

メッセージ記述子のプロパティー

プログラム内でメッセージ記述子のプロパティーを操作する場合は、それと等価な 16 進フィールドを使用することをお勧めします。

このセクションでは、以下のプロパティーについて説明しています。

- AccountingToken
- CorrelationId
- GroupId
- MessageId

WebSphere MQ アプリケーションがメッセージの発信元で、WebSphere MQ がこれらのプロパティーを生成する場合、これらのプロパティーの値を調べたり WebSphere MQ へのメッセージ内にそれらのプロパティーを渡したりするなど、何らかの方法でそれらのプロパティーを操作する場合は、AccountingTokenHex、CorrelationIdHex、GroupIdHex、および MessageIdHex の各プロパティーを使用することをお勧めします。 これは、WebSphere MQ が生成する値は、0 から 255 (0 と 255 を含む) までのいずれかの値をとるバイト列であり、 印刷可能文字ストリングではないからです。

MQAX スクリプトがメッセージの発信元である場合は、AccountingToken、CorrelationId、GroupId、および MessageId の各プロパティーか、またはそれらと等価な 16 進プロパティーのいずれかを使用することができます。

WebSphere MQ 定数

WebSphere MQ 定数は、ライブラリー MQAX200 内の enum WebSphere MQ のメンバーとして提供されています。

WebSphere MQ ストリング定数

WebSphere MQ ストリング定数およびそれらに対応する文字ストリング。

WebSphere MQ のストリング定数は、WebSphere MQ Automation Classes for ActiveX の使用時は利用できません。 次のリストに示す明示的な文字ストリングと、その他の必要な文字ストリングを使用してください。 コマンドはスペースを使用して埋めて 8 文字にすることが必要です。

MQFMT_NONE	" "
MQFMT_ADMIN	"MQADMIN "
MQFMT_CHANNEL_COMPLETED	"MQCHCOM "
MQFMT_CICS	"MQCICS "
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD "
MQFMT_DIST_HEADER	"MQHDIST "
MQFMT_EVENT	"MQEVENT "
MQFMT_IMS	"MQIMS "

MQFMT_IMS_VAR_STRIsNG	"MQIMSVS "
MQFMT_MD_EXTENSION	"MQHMDE "
MQFMT_PCF	"MQPCF "
MQFMT_REF_MSG_HEADER	"MQHREF "
MQFMT_RF_HEADER	"MQHRF "
MQFMT_STRING	"MQSTR "
MQFMT_TRIGGER	"MQTRIG "
MQFMT_WORK_INFO_HEADER	"MQHWIH "
MQFMT_XMIT_Q_HEADER	「MQXMIT」

ヌル・ストリング定数

MQMessage の MQMI_NONE (24 の NULL 文字)、MQCI_NONE (24 の NULL 文字)、MQGI_NONE (24 の NULL 文字)、および MQACT_NONE (32 の NULL 文字) の 4 つのプロパティーの初期設定に使用される WebSphere MQ の定数は、 WebSphere MQ Automation Classes for ActiveX ではサポートされません。 これらのプロパティーに空ストリングを設定すると、同じ効果があります。

例えば、MQMessage の各種 ID を空ストリングに設定する場合、次 のように指定します。
mymessage.**MessageId** = "" *mymessage*.**CorrelationId** = "" *mymessage*.**AccountingToken** = ""

WebSphere MQ からのメッセージの受信

WebSphere MQ からメッセージを受け取るには、いくつかの方法があります。

- Visual Basic の TIMER 機能を使用して、GET に続けて Wait を発行してポーリングする方法。
- GET を Wait オプション付きで発行する方法。 WaitInterval プロパティーを設定して待ち時間を指定します。 マルチスレッド環境で稼働するようシステムをセットアップしている場合でも、その時点で実行中のソフトウェアが単一スレッド方式でだけ実行する可能性がある場合には、このことを検討してください。 この設定では、システムが無期限にロックアップすることが なくなります。

  他のスレッドは影響を受けずに動作します。 ただし、他のスレッドで WebSphere MQ にアクセスする必要がある場合は、 別の MQAX キュー・マネージャーとキュー・オブジェクトを 使用して、他のスレッドと WebSphere MQ の間に 2 つ目の接続を確立する必要があります。

Wait オプションを指定して GET を実行し、WaitInterval を MQWI_UNLIMITED に設定すると、プロセスが単一スレッド方式の場合、GET 呼び出しが完了するまでシステムがロックされます。

データ変換の使用

WebSphere MQ Automation Classes for ActiveX は、数値エンコードと文字セット変換という 2 つのデータ変換形式をサポートしています。

数値のコード化

MQMessage の Encoding プロパティーを設定すると、各種の 数値コード化システム間の変換が次のメソッドで行われます。

- ReadDecimal2 メソッド
- ReadDecimal4 メソッド
- ReadDouble メソッド
- ReadDouble4 メソッド
- ReadFloat メソッド
- ReadInt2 メソッド
- ReadInt4 メソッド

- ReadLong メソッド
- ReadShort メソッド
- ReadUInt2 メソッド
- WriteDecimal2 メソッド
- WriteDecimal4 メソッド
- WriteDouble メソッド
- WriteDouble4 メソッド
- WriteFloat メソッド
- WriteInt2 メソッド
- WriteInt4 メソッド
- WriteLong メソッド
- WriteShort メソッド
- WriteUInt2 メソッド

コード化プロパティーの設定と解釈は、システムに提供された WebSphere MQ 定数を使用して行うことができます。1042 ページの図 216 は、以下の例を示しています。

```
/* Encodings for Binary Integers */
        MQENC_INTEGER_UNDEFINED
        MQENC_INTEGER_NORMAL
        MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
        MQENC_DECIMAL_UNDEFINED
        MQENC_DECIMAL_NORMAL
        MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
        MQENC_FLOAT_UNDEFINED
        MQENC_FLOAT_IEEE_NORMAL
        MQENC_FLOAT_IEEE_REVERSED
        MQENC_FLOAT_S390
```

図 *216.* コード化のためにシステムに提供された *WebSphere MQ* 定数

例えば、整数を Intel システムから System/390® オペレーティング・システムに System/390 エンコードで送信するには、次のようにします。

```
Dim msg As New MQMessage   'Define a WebSphere MQ message for our use..
Print msg.Encoding    'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg.Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg.Encoding    'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg.WriteLong(local_num)         'Write the number into the message
```

文字セットの変換

文字セットの変換は、コード・ページの異なるシステム間でメッセ ージを送信する場合に必要です。 コード・ページの変換は、次のメソッドやプロパティーで使用します。

- ReadString メソッド
- ReadNullTerminatedString メソッド
- WriteString メソッド
- WriteNullTerminatedString メソッド
- MessageData プロパティー

MQMessage の CharacterSet プロパティーは、サポートされる文字セットの値 (CCSID) に設定する必要があります。

WebSphere MQ Automation Classes for ActiveX は、変換テーブルを使用して文字セットの変換を行います。

例えば、ストリングを自動的にコード・ページ 437 に変換 する場合は、次のように指定します。

```
Dim msg As New MQMessage                 'Define a WebSphere MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

WriteString メソッドで、ストリング・データ (例の "A character string") を Unicode ストリングとし て受け取ります。 その後、変換テーブル 34B001B5.TBL を使用して このデータの Unicode からコード・ページ 437 への変換を行います。

Unicode ストリングの中でコード・ページ 437 でサポート されない文字には 、コード・ページ 437 の標準の置換文字が与えられます。

同様に、ReadString メソッドを使用すると、WebSphere MQ メッセージ記述子 (MQMD) の値によって設定された文字セットが着信メッセージに含まれ、 このコード・ページから Unicode への変換が行われた後、変換後の文字セットがスクリプト記述言語に渡されます。

スレッド

WebSphere MQ Automation Classes for ActiveX は、オブジェクトをスレッド間で使用できる自由スレッド化モデルを導入します。

MQAX では、MQQueue オブジェクトおよび MQQueueManager オブジェクトが使用できますが、 現在、WebSphere MQ では異なるスレッド間でのハンドルの共用はできません。

別のスレッド上でこれらのハンドルを使用しようとするとエラーになり、 WebSphere MQ が戻りコード MQRC_HCONN_ERROR を戻します。

注 : 各プロセスごとに MQSession オブジェクトは 1 つしか存在しません。 マルチスレッド環境では、MQSession CompletionCode と ReasonCode の使用はお勧めできません。 MQSession のエラー値は、最初のスレッド上でのエラーの発生とチェックの間に 2 番目のスレッドによって上書きされる可能性があります。 スレッドは、各メソッドの呼び出し時またはプロパティーへのアクセス時に直列 化されます。 したがって、Wait オプションを指定して Get を発行すると、MQAX オブジェクトにアクセスする他のスレッドはその操作が完了するまで中断状態にされてしまいます。

エラーの処理

ここでは、MQAX オブジェクトのプロパティー、エラー処理の動作、発生した例外の処理方法を記述する規則、およびプロパティーの取得について説明します。

各 MQAX オブジェクトは、エラー情報を保持するプロパティーと これらのプロパティーを初期化または消去するメソッドを持っています。 エラー情報の保持に使用するプロパティーは、次のとおりです。

- CompletionCode
- ReasonCode
- ReasonName

メソッドは次のとおりです。

- ClearErrorCodes

エラー処理の動作

MQAX スクリプトまたはアプリケーションが、 MQAX オブジェク トのメソッドを呼び出したり、MQAX オブジェクトのプロパティーにアクセ スしたり、更新したりすると、次の処理が発生します。

1. 当該オブジェクトの ReasonCode と CompletionCode が更新されま す。

2. MQSession オブジェクトの ReasonCode と CompletionCode も同じ 情報で更新されます。

   注 : スレッド・アプリケーションにおける MQSession エラー・コードの使用に関する制約事項については、1043 ページの『スレッド』を参照してください。

   CompletionCode が MQSession の ExceptionThreshold プロパティーの 値以上の場合、MQAX は例外 (No. 32000) をスローします。 この 例外は、スクリプトの On Error 文 (または同等文) で処理します。
3. Error 関数を使用して、次の形式の関連するエラー・ストリングを検索します。

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   On Error 文の使用法の詳細については、ActiveX のスクリプト記述言語についての資料を参照してください。

   簡単なエラー・ハンドラーでは、MQSession オブジェクト に CompletionCode および ReasonCode を使用することが便利です。

   ReasonName プロパティーは、 ReasonCode の現行値の WebSphere MQ 記号名を戻します。

例外の発生

例外発生の処理方法についての規則を以下に示します。

- プロパティーまたはメソッドが完了コードを例外のしきい値 (通常は 2 に設定) 以上の値に設定すると、例外が発生します。
- すべてのメソッドの呼び出しとプロパティーの設定で、完了コードが設定さ れます。

プロパティーの読み取り

CompletionCode と ReasonCode は必ず更新されるとは限らないため、ここに挙げるのは特殊なケースです。

- プロパティーの読み取りに成功した場合、そのオブジェクトと MQSession オブジェクトの ReasonCode と CompletionCode は未変更のまま です。
- プロパティーの読み取りに失敗して警告の CompletionCode が設定さ れた場合は、ReasonCode と CompletionCode は未変更のままです。
- プロパティーの読み取りに失敗してエラーの CompletionCode が設定 された場合は、ReasonCode と CompletionCode が真の値を反映するように更新され、上記のエラー処理が行われます。

MQSession クラスは、メソッド *ReasonCodeName* を持っています。 このメソッドは、WebSphere MQ の理由コードを記号名で置き換えるときに使用されます。 このメソッドは、プログラム開発中に予期しないエラーが発生する恐れのある ときに特に役に立ちます。 ただし、名前は、ユーザーに提示するのに適切とはいえません。

クラスはそれぞれ、そのクラスの現行の理由コードの記号名を戻すプロパティー *ReasonName* も持っています。

WebSphere MQ Automation Classes for ActiveX リファレンス

このセクションでは、ActiveX のために開発された WebSphere MQ Automation Classes for ActiveX (MQAX) のクラスについて説明しています。 このクラスにより、WebSphere MQ を使用して ActiveX 以外の環境で稼働している他のアプリケーションにアクセスできる ActiveX アプリケーションを作成することができます。

WebSphere MQ Automation Classes for ActiveX インターフェース

WebSphere MQ Automation Classes for ActiveX は、クラスの使用に必要な事前定義された ActiveX の数値定数 (MQMT_REQUEST など) を提供しています。

ActiveX 自動化クラスは以下のもので構成されています。

- 1046 ページの『MQSession クラス』
- 1049 ページの『MQQueueManager クラス』
- 1060 ページの『MQQueue クラス』
- 1075 ページの『MQMessage クラス』
- 1096 ページの『MQPutMessageOptions クラス』
- 1098 ページの『MQGetMessageOptions クラス』
- 1101 ページの『MQDistributionList クラス』
- 1105 ページの『MQDistributionListItem クラス』

さらに、WebSphere MQ Automation Classes for ActiveX は、 クラスを使用するために必要である事前定義の ActiveX 数値定数 (MQMT_REQUEST など) を提供します。 この定数は、ライブラリー MQAX200 の enum MQ にあります。 この定数は、WebSphere MQ C ヘッダー・ファイル (cmqc*.h) に定義された定数のサブセットで、 ActiveX 理由コードのための追加 WebSphere MQ 自動化クラスが付随しています。

WebSphere MQ Automation Classes for ActiveX クラスについて

この情報は、アプリケーションの開発に関する参照情報の参照トピックと併せてお読みください。

重要な情報については、Windows 上のプライマリー・インストールでのみ使用できる機能 を参照してください。

MQSession クラスは、MQAX オブジェクトのどれかで最後に実行された処理の状況を含むルート・オブジェクトを提供します。 詳しくは、1043 ページの『エラーの処理』 を参照してください。

MQQueueManager および MQQueue クラスは、基礎となる WebSphere MQ オブジェクトにアクセスできます。 一般に、これらのクラスのためのメソッドまたはプロパティーにアクセスすると、 結果として WebSphere MQ MQI 全体で呼び出しが実行されます。

MQMessage、MQPutMessageOptions、および MQGetMessageOptions クラスは、MQMD、MQPMO、および MQGMO データ構造体をカプセル化します。 キューにメッセージを送信したり、キューからメッセージを検索したりする場合に、これらのクラスを使用すると役立ちます。

MQDistributionList クラスは、キューの集合 (出力のためのローカル、リモートまたは別名) をカプセル化します。 MQDistributionListItem クラスは、MQOR、MQRR、および MQPMR 構造体をカプセル化し、それを所有している配布リストと関連付けます。

パラメーターの受け渡し

メソッド呼び出しでのパラメーターは、パラメーターがオブジェクトの場合は除き、すべて値を無視します。その場合、パラメーターは渡される参照になります。

提供されるクラス定義は、パラメーターまたはプロパティーごとにデータ型をリストします。 Visual Basic など多くの ActiveX クライアントでは、使用される変数が必須タイプでない場合、その値は自動的に必須タイプに変換されたり、必須タイプから変換されます。このような変換が可能になります。 クライアントの標準規則を以下に示します。MQAX ではこのような変換はできません。

多くのメソッドは固定長ストリング・パラメーターを取るか、または固定長文字ストリングを戻します。変換規則は次のとおりです。

- ユーザーが入力パラメーターまたは戻り値として誤った長さの固定長ストリングを指定する場合、値が切り捨てられるか、または必要に応じて末尾に空間が埋められます。
- ユーザーが入力パラメーターとして誤った長さの可変長ストリングを指定する場合、値が切り捨てられるか、末尾に空間が埋められます。
- ユーザーが戻り値として誤った長さの可変長ストリングを指定する場合、そのストリングは必要な長さに調整されます (値が戻されると、ストリングの直前の値が破棄されるためです)。
- 入力パラメーターとして提供されるストリングにヌルが組み込まれていることがあります。

これらのクラスは MQAX200 ライブラリーにあります。

オブジェクト・アクセス方式

これらの方式は、単一の WebSphere MQ 呼び出しのどれにも直接関係しません。 これらの方式はそれぞれ、参照情報を保管するオブジェクトを作成し、 その後に WebSphere MQ オブジェクトを接続したり、または MQSeries オブジェクトをオープンします。

キュー・マネージャーに接続が行われると、WebSphere MQ が生成する「接続ハンドル」属性が保持されます。

キューをオープンすると、WebSphere MQ が生成する「オブジェクト・ハンドル」属性が保持されます。

これらの WebSphere MQ 属性は MQAX プログラムで直接使用できません。

エラー

パラメーターの受け渡しにおける構文エラーが ActiveX クライアントによりコンパイル時および実行時に検出されることがあります。 Visual Basic の On Error を使用して、エラーをトラップすることができます。

WebSphere MQ ActiveX クラスにはすべて特殊な読み取り専用プロパティー、 つまり ReasonCode および CompletionCode が含まれています。 いつでもこれらのプロパティーを読み取ることができます。

他のプロパティーにアクセスしようとしたり、メソッド呼び出しを実行しようとしたりすると、WebSphere MQ からエラーが生成される可能性があります。

プロパティーの設定またはメソッド呼び出しが正常に終了した場合、所有オブジェクトの ReasonCode は MQRC_NONE に設定され、CompletionCode は MQCC_OK に設定されます。

プロパティーへのアクセスまたはメソッド呼び出しが正常に終了しない場合、理由コードおよび完了コードがこれらのフィールドに設定されます。

MQSession クラス

これは、WebSphere MQ Automation Classes for ActiveX のためのルート・クラスです。

ActiveX クライアント・プロセスあたり MQSession オブジェクトは必ず 1 つしかありません。 2 番目のオブジェクトを作成しようとすると、元のオブジェクトに対する 2 番目の参照が作成されます。

作成

新規作成は、新規の MQSession オブジェクトを作成します。

構文

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

プロパティー

- 1047 ページの『CompletionCode プロパティー』.
- 1047 ページの『ExceptionThreshold プロパティー』.
- 1047 ページの『ReasonCode プロパティー』.
- 1048 ページの『ReasonName プロパティー』.

メソッド

- 1048 ページの『AccessGetMessageOptions メソッド』.
- 1048 ページの『AccessMessage メソッド』.
- 1048 ページの『AccessPutMessageOptions メソッド』.
- 1048 ページの『AccessQueueManager メソッド』.
- 1049 ページの『ClearErrorCodes メソッド』.

- 1049 ページの『ReasonCodeName メソッド』.

CompletionCode プロパティー

読み取り専用。 WebSphere MQ オブジェクトに対し発行された最新のメソッドまたはプロパティー・セットによって設定された WebSphere MQ 完了コードを戻します。

メソッドまたはプロパティー・セットが MQAX オブジェクトに対して正常に呼び出される場合、MQCC_OK にリセットされます。

エラー事象ハンドラーは、どのオブジェクトが呼び出されたかを認識しなくても、このプロパティーを検索し、エラーを診断します。

簡単なエラー・ハンドラーでは、MQSession オブジェクト に CompletionCode および ReasonCode を使用することが非常に便利です。

注 : スレッド・アプリケーションにおける MQSession エラー・コードの使用に関する制約事項については、1043 ページの『スレッド』を参照してください。

定義先: MQSession クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文:

取得する場合: *completioncode & = MQSession*.CompletionCode

ExceptionThreshold プロパティー

読み取り/書き込み可能。 MQAX が例外を処理するための WebSphere MQ エラーのレベルを定義します。 デフォルトは MQCC_FAILED です。 実質的に MQCC_FAILED より大きい値は、例外処理を避けるので、プログラマーに CompletionCode および ReasonCode での検査の実行を任せます。

定義先: MQSession クラス

データ型: 長整数 (Long)

値:

- 任意。ただし、MQCC_WARNING、MQCC_FAILED、またはそれ以上を検討してください。

構文:

取得する場合: *ExceptionThreshold& = MQSession*.**ExceptionThreshold**

設定する場合: *MQSession*.**ExceptionThreshold** = *ExceptionThreshold$*

ReasonCode プロパティー

読み取り専用。 WebSphere MQ オブジェクトに対し発行された最新のメソッドまたはプロパティー・セットによって設定された理由コードを戻します。

エラー事象ハンドラーは、どのオブジェクトが呼び出されたかを認識しなくても、このプロパティーを検索し、エラーを診断します。

簡単なエラー・ハンドラーでは、MQSession オブジェクト に CompletionCode および ReasonCode を使用することが非常に便利です。

注 : スレッド・アプリケーションにおける MQSession エラー・コードの使用に関する制約事項については、1043 ページの『スレッド』を参照してください。

定義先: MQSession クラス

データ型: 長整数 (Long)

値:

- Reason (MQLONG) と 1112 ページの『理由コード』に列挙されている追加 MQAX 値を参照してください。

構文: 取得する場合: *reasoncode & = MQSession*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

注 : スレッド・アプリケーションにおける MQSession エラー・コードの使用に関する制約事項については、1043 ページの『スレッド』を参照してください。

定義先: MQSession クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQSession*.**ReasonName**

AccessGetMessageOptions メソッド

新規 MQGetMessageOptions オブジェクトを作成します。

定義先: MQSession クラス

構文: *gmo = MQSession*.**AccessGetMessageOptions()**

AccessMessage メソッド

新規 MQMessage オブジェクトを作成します。

定義先: MQSession クラス

構文: *msg = MQSession*.**AccessMessage()**

AccessPutMessageOptions メソッド

新規 MQPutMessageOptions オブジェクトを作成します。

定義先: MQSession クラス

構文: *pmo = MQSession*.**AccessPutMessageOptions()**

AccessQueueManager メソッド

新規 MQQueueManager オブジェクトを作成し、WebSphere MQ MQI クライアントまたは WebSphere MQ サーバーを使って実キュー・マネージャーにそのオブジェクトを接続します。 このメソッドは、接続を実行する上に、キュー・マネージャーのオブジェクトのオープンも実行します。

WebSphere MQ MQI クライアントおよび WebSphere MQ サーバーを両方ともご使用のシステムにインストールすると、 MQAX アプリケーションはサーバーに対してデフォルトで実行されるようになります。 クライアントに対して MQAX を実行するには、GMQ_MQ_LIB 環境変数でクライアント・バインディング・ライブラリーを指定する必要があります。例えば GMQ_MQ_LIB=mqic.dll と設定します。

クライアントのみのインストールの場合、GMQ_MQ_LIB 環境変数を設定する必要はありません。 この変数が設定されていない場合、WebSphere MQ は amqzst.dll のロードを試みます。 この DLL が存在しない場合 (クライアントのみのインストールで見られます)、WebSphere MQ は mqic.dll のロードを試みます。

正常に終了した場合は、MQQueueManager の ConnectionStatus を TRUE に設定します。

キュー・マネージャーを、ActiveX インスタンスあたり 1 つの MQQueueManager オブジェクトにだけ接続できます。

キュー・マネージャーへの接続が失敗すると、エラー事象が発生し、MQSession オブジェクトの ReasonCode および CompletionCode が設定されます。

定義先: MQSession クラス

構文: *set qm = MQSession*.**AccessQueueManager** (*Name$*)

パラメーター:*Name$* はストリング (String)。 接続されるキュー・マネージャーの名前。

ClearErrorCodes メソッド

CompletionCode を MQCC_OK にリセットし、ReasonCode を MQRC_NONE にリセットします。

定義先: MQSession クラス

構文: Call *MQSession*.**ClearErrorCodes()**

ReasonCodeName メソッド

所定の数値と共に理由コードの名前を戻します。 ユーザーにエラー条件の明確な指示を提示することが役立ちます。 名前は、多少不明確 (例えば ReasonCodeName(2059) が **MQRC_Q_MGR_NOT_AVAILABLE**) なので、考えられるエラーを把握し、 アプリケーションに適する説明と置き換えるべきです。

定義先: MQSession クラス

構文: *errname $= MQSession*.**ReasonCodeName**(*reasonCode&*)

パラメーター:*reasoncode &* は長整数 (Long)。 記号名が必要とされる理由コード。

MQQueueManager クラス

このクラスは、キュー・マネージャーへの接続を表します。 WebSphere MQ クライアントが提供するアクセス権を使って、キュー・マネージャーをローカル (WebSphere MQ サーバー) またはリモートで実行できます。 アプリケーションは、このクラスのオブジェクトを作成し、そのオブジェクトをキュー・マネージャーに接続する必要があります。 このクラスのオブジェクトが破棄される場合、キュー・マネージャーから自動的に切断されます。

包含

MQQueue クラス・オブジェクトはこのクラスと関連付けられています。

新規作成は、新規 MQQueueManager オブジェクトを作成し、すべてのプロパティーを初期値に設定します。 あるいは、MQSession クラスの AccessQueueManager メソッドを使用します。

作成

新規作成は、**新規** MQQueueManager オブジェクトを作成し、すべてのプロパティーを初期値に設定します。 あるいは、MQSession クラスの AccessQueueManager メソッドを使用します。

構文

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

プロパティー

- 1051 ページの『AlternateUserId プロパティー』.
- 1051 ページの『AuthorityEvent プロパティー』.
- 1051 ページの『BeginOptions プロパティー』.
- 1052 ページの『ChannelAutoDefinition プロパティー』.
- 1052 ページの『ChannelAutoDefinitionEvent プロパティー』.
- 1052 ページの『ChannelAutoDefinitionExit プロパティー』.

- 1052 ページの『CharacterSet プロパティー』.
- 1052 ページの『CloseOptions プロパティー』.
- 1053 ページの『CommandInputQueueName プロパティー』.
- 1053 ページの『CommandLevel プロパティー』.
- 1053 ページの『CompletionCode プロパティー』.
- 1053 ページの『ConnectionHandle プロパティー』.
- 1053 ページの『ConnectionStatus プロパティー』.
- 1054 ページの『ConnectOptions プロパティー』.
- 1054 ページの『DeadLetterQueueName プロパティー』.
- 1054 ページの『DefaultTransmissionQueueName プロパティー』.
- 1054 ページの『Description プロパティー』.
- 1054 ページの『DistributionLists プロパティー』.
- 1055 ページの『InhibitEvent プロパティー』.
- 1055 ページの『IsConnected プロパティー』.
- 1055 ページの『IsOpen プロパティー』.
- 1055 ページの『LocalEvent プロパティー』.
- 1056 ページの『MaximumHandles プロパティー』.
- 1056 ページの『MaximumMessageLength プロパティー』.
- 1056 ページの『MaximumPriority プロパティー』.
- 1056 ページの『MaximumUncommittedMessages プロパティー』.
- 1056 ページの『Name プロパティー』.
- 1056 ページの『ObjectHandle プロパティー』.
- 1056 ページの『PerformanceEvent プロパティー』.
- 1057 ページの『Platform プロパティー』.
- 1057 ページの『ReasonCode プロパティー』.
- 1057 ページの『ReasonName プロパティー』.
- 1057 ページの『RemoteEvent プロパティー』.
- 1058 ページの『StartStopEvent プロパティー』.
- 1058 ページの『SyncPointAvailability プロパティー』.
- 1058 ページの『TriggerInterval プロパティー』.

方法

- 1058 ページの『AccessQueue メソッド』.
- 1059 ページの『AddDistributionList メソッド』.
- 1059 ページの『Backout メソッド』.
- 1059 ページの『Begin メソッド』.
- 1059 ページの『ClearErrorCodes メソッド』.
- 1059 ページの『Commit メソッド』.
- 1060 ページの『Connect メソッド』.
- 1060 ページの『Disconnect メソッド』.

プロパティー・アクセス

次のプロパティーには、いつでもアクセスできます。

- 1051 ページの『AlternateUserId プロパティー』.
- 1053 ページの『CompletionCode プロパティー』.
- 1053 ページの『ConnectionStatus プロパティー』.
- 1057 ページの『ReasonCode プロパティー』.

上記以外のプロパティーには、オブジェクトがキュー・マネージャーに接続されており、ユーザー ID がそのキュー・マネージャーに対して照会する許可が与えられている場合にだけ、アクセスできます。 代替ユーザー ID が設定され、現在のユーザー ID に代替ユーザー ID を使用する許可が与えられている場合、代替ユーザー ID に代わりの照会権限があるかどうかチェックされます。

これらの条件が適用されない場合、WebSphere MQ Automation Classes for ActiveX は、 キュー・マネージャーに接続しようと試み、照会のため自動的にオープンします。 正常に動作しない場合は、MQCC_FAILED という CompletionCode および次のいずれかの ReasonCodes を設定します。

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

AlternateUserId プロパティー

読み取り/書き込み可能。 キュー・マネージャー属性へのアクセスを検査するために使われる代替ユーザー ID です。

このプロパティーは、IsConnected が TRUE の場合には設定しないでください。

このプロパティーは、オブジェクトのオープン中には設定できません。

Defined in: MQQueueManager クラス

Data Type: 12 文字のストリング (String)

Syntax: 取得する場合: *altuser $= MQQueueManager*.**AlternateUserId** 設定する場合: *MQQueueManager*.**AlternateUserId** = *altuser $*

AuthorityEvent プロパティー

読み取り専用。 MQI AuthorityEvent 属性。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *authevent = MQQueueManager*.**AuthorityEvent**

BeginOptions プロパティー

読み取り/書き込み可能。 これらは Begin メソッドに適用するオプションです。 初期値は MQBO_NONE です。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQBO_NONE

構文: 取得する場合: *beginoptions & =MQQueueManager*.**BeginOptions**

設定する場合: *MQQueueManager*.**BeginOptions**=*beginoptions &*

ChannelAutoDefinition プロパティー

読み取り専用。 これは自動チャネル定義が許可されているかどうかを制御します。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQCHAD_DISABLED
- MQCHAD_ENABLED

構文: 取得する場合: *channelautodef & = MQQueueManager*.**ChannelAuto 定義**

ChannelAutoDefinitionEvent プロパティー

読み取り専用。 これは自動チャネル定義事象が生成されるかどうかを制御します。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *channelautodefevent & =MQQueueManager*.**ChannelAutoDefinitionEvent**

ChannelAutoDefinitionExit プロパティー

読み取り専用。 自動チャネル定義に使用されるユーザー出口の名前。

定義先:
MQQueueManager クラス

データ型:
ストリング

構文: 取得する場合: *channelautodefexit$= MQQueueManager*.**ChannelAutoDefinitionExit**

CharacterSet プロパティー

読み取り専用。 MQI CodedCharSetId 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *characterset & = MQQueueManager*.**CharacterSet**

CloseOptions プロパティー

読み取り/書き込み可能。 キュー・マネージャーをクローズするときに発生することを制御するために使用されるオプション。 初期値は MQCO_NONE です。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQCO_NONE

構文: 取得する場合: *closeopt & = MQQueueManager*.**CloseOptions**

設定する場合: *MQQueueManager*.**CloseOptions** =*closeopt &*

CommandInputQueueName プロパティー

読み取り専用。 MQI CommandInputQName 属性。

定義先: MQQueueManager クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *commandinputqname$ = MQQueueManager*.**CommandInputQueueName**

CommandLevel プロパティー

読み取り専用。 WebSphere MQ キュー・マネージャーの実装のレベルおよびバージョン (MQI CommandLevel 属性) を戻します。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *level & = MQQueueManager*.**CommandLevel**

CompletionCode プロパティー

読み取り専用。 オブジェクトに対して発行された最新のメソッドまたはプロパティー・アクセスによって設定された完了コードを戻します。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = MQQueueManager*.**CompletionCode**

ConnectionHandle プロパティー

読み取り専用。 WebSphere MQ キュー・マネージャー・オブジェクトの接続ハンドル。

定義先:
MQQueueManager クラス

データ型:
Long

構文: 取得する場合: *hconn & = MQQueueManager.***ConnectionHandle**

ConnectionStatus プロパティー

読み取り専用。 オブジェクトがキュー・マネージャーに接続されているかどうかを示します。

定義先: MQQueueManager クラス

データ型: ブール (Boolean)

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *status = MQQueueManager*.**ConnectionStatus**

ConnectOptions プロパティー

読み取り / 書き込み可能。 これらのオプションは Connect メソッドに適用します。 初期値は MQCNO_NONE です。

定義先:
MQQueueManager クラス

データ型:
Long

値:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

構文: 取得する場合: *connectoptions & =MQQueueManager*.**ConnectOptions**

設定する場合: *MQQueueManager*.**ConnectOptions**=*connectoptions &*

DeadLetterQueueName プロパティー

読み取り専用。 MQI DeadLetterQName 属性。

定義先: MQQueueManager クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *dlqname$ = MQQueueManager*.**DeadLetterQueueName**

DefaultTransmissionQueueName プロパティー

読み取り専用。 MQI DefXmitQName 属性。

定義先: MQQueueManager クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *defxmitqname$ = MQQueueManager*.**DefaultTransmissionQueueName**

Description プロパティー

読み取り専用。 MQI QMgrDesc 属性。

定義先: MQQueueManager クラス

データ型: 64 文字のストリング (String)

構文: 取得する場合: *description$ = MQQueueManager*.**Description**

DistributionLists プロパティー

読み取り専用。 これは、配布リストをサポートするキュー・マネージャーの機能です。

定義先:
MQQueueManager クラス

データ型:
ブール値

値:

- TRUE (-1)

- FALSE (0)

構文: 取得する場合: *distributionlists= MQQueueManager*.**DistributionLists**

InhibitEvent プロパティー

読み取り専用。 MQI InhibitEvent 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *inhibevent & = MQQueueManager*.**InhibitEvent**

IsConnected プロパティー

キュー・マネージャーが現在接続されているかどうかを示す値。

読み取り専用。

定義先: MQQueueManager クラス

データ型: ブール (Boolean)

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *isconnected = MQQueueManager*.**IsConnected**

IsOpen プロパティー

キュー・マネージャーが照会のために現在オープンしているかどうかを示す値。

読み取り専用。

定義先:
MQQueueManager クラス

データ型:
ブール値

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *IsOpen = MQQueueManager*.**IsOpen**

LocalEvent プロパティー

読み取り専用。 MQI LocalEvent 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *localevent & = MQQueueManager*.**LocalEvent**

MaximumHandles プロパティー

読み取り専用。 MQI MaxHandles 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxhandle & = MQQueueManager*.**MaximumHandles**

MaximumMessageLength プロパティー

読み取り専用。 MQI MaxMsgLength キュー・マネージャー属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxmessagelength & = MQQueueManager*.**MaximumMessageLength**

MaximumPriority プロパティー

読み取り専用。 MQI MaxPriority 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxpriority & = MQQueueManager*.**MaximumPriority**

MaximumUncommittedMessages プロパティー

読み取り専用。 MQI MaxUncommittedMsgs 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxuncommitted & = MQQueueManager*.**MaximumUncommitted メッセージ**

Name プロパティー

読み取り/書き込み可能。 MQI QMgrName 属性。 MQQueueManager を接続すると、このプロパティーを書き込むことができません。

定義先: MQQueueManager クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *name$ = MQQueueManager*.**name**

設定する場合 *MQQueueManager*.**name** = *name$*

注 : Visual Basic は、ビジュアル・インターフェースで使用するために "Name" プロパティーを予約しています。 したがって、Visual Basic 内で使用する場合は、小文字、つまり "name" を使用します。

ObjectHandle プロパティー

読み取り専用。 WebSphere MQ キュー・マネージャー・オブジェクトのオブジェクト・ハンドル。

定義先:
MQQueueManager クラス

データ・タイプ
Long

構文: 取得する場合: *hobj & = MQQueueManager.***ObjectHandle**

PerformanceEvent プロパティー

読み取り専用。 MQI PerformanceEvent 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *perfevent & = MQQueueManager*.PerformanceEvent

Platform プロパティー

読み取り専用。 MQI Platform 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

構文: 取得する場合: *platform & = MQQueueManager*.**プラットフォーム**

ReasonCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コードを戻します。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = MQQueueManager*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

定義先: MQQueueManager クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQQueueManager*.**ReasonName**

RemoteEvent プロパティー

読み取り専用。 MQI RemoteEvent 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *remoteevent & = MQQueueManager*.**RemoteEvent**

StartStopEvent プロパティー

読み取り専用。 MQI StartStopEvent 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *strstpevent & = MQQueueManager*.**StartStop イベント**

SyncPointAvailability プロパティー

読み取り専用。 MQI SyncPoint 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

値:

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

構文: 取得する場合: *syncpointavailability & = MQQueueManager*.**SyncPointAvailability**

TriggerInterval プロパティー

読み取り専用。 MQI TriggerInterval 属性。

定義先: MQQueueManager クラス

データ型: 長整数 (Long)

構文: 取得する場合: *trigint & = MQQueueManager*.**TriggerInterval**

AccessQueue メソッド

MQQueue オブジェクトを作成し、そのキューの接続参照プロパティーを設定することにより、そのオブジェクトをこの MQQueueManager に関連付けます。 このメソッドは、MQQueue オブジェクトの Name、OpenOptions、DynamicQueueName、および AlternateUserId プロパティーを提供される値に設定し、オープンしようと試みます。

オープンが正常に終了しない場合、呼び出しは失敗します。 このオブジェクトに対してエラー・イベントが発生します。 このオブジェクトの ReasonCode と CompletionCode、および MQSession ReasonCode と CompletionCode が設定されます。

DynamicQueueName、QueueManagerName、および AlternateUserId パラメーターは任意選択で、デフォルト値は "" です。

キュー・プロパティーが読み取られる場合、その他のオプションに加えて、 OpenOption MQOO_INQUIRE を指定する必要があります。

オープンされるキューがローカルの場合、QueueManagerName を設定しないか、または "" に設定します。そうでない場合は、そのキューを所有するリモートのキュー・マネージャーの名前に設定します。これにより、そのリモート・キューのローカル定義を開く試みが行われます。 リモート・キュー名の解決およびキュー・マネージャーの別名割り当てについて詳しくは、 別名とは何ですか?

Name プロパティーがモデル・キュー名に設定される場合、作成される動的キューの名前を DynamicQueueName$ パラメーターに指定します。 DynamicQueueName$ パラメーターに提供される値が "" の場合、キュー・オブジェクトに設定され、オープン呼び出しで使用される値は、"AMQ.*" になります。 動的キューの命名について詳しくは、223 ページの『動的キューの作成』を参照してください。

定義

定義先: MQQueueManager クラス。

構文

構文: set queue = MQQueueManager.**AccessQueue**(Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parameters

Name$ はストリング (String)。 WebSphere MQ キューの名前を表す。

OpenOptions: は長整数 (Long)。 キューがオープンされるときに使用されるオプション。 OpenOptions (MQLONG) を参照してください。

QueueManagerName$ はストリング (String)。 オープンされるキューを所有するキュー・マネージャーの名前。 "" の値は、キュー・マネージャーがローカルであることを暗黙指定します。

*DynamicQueueName$* はストリング (String)。 Name$ パラメーターがモデル・キューを指定する場合に、キューがオープンされる時点で動的キューに割り当てられる名前。

AlternateUserId$ はストリング (String)。 キューのオープン時にアクセスの妥当性検査をするために使用される代替ユーザー ID。

AddDistributionList メソッド

新規 MQDistributionList オブジェクトを作成し、所有しているキュー・マネージャーへの接続参照を設定します。

定義先:
MQQueueManager クラス

構文: *set distributionlist* = **MQQueueManager.***AddDistributionList*

Backout メソッド

最新の同期点以降作業単位の一部として発生したコミットされていないメッセージの書き込みおよび読み取りをバックアウトします。

定義先: MQQueueManager クラス

構文: Call *MQQueueManager.***Backout()**

Begin メソッド

キュー・マネージャーで調整される作業単位を開始します。 開始オプションは、このメソッドの動作に影響します。

定義先:
MQQueueManager クラス

構文: Call *MQQueueManager*.**Begin()**

ClearErrorCodes メソッド

MQQueueManager クラスおよび MQSession クラスの両方について、 CompletionCode を MQCC_OK に、ReasonCode を MQRC_NONE にリセットします。

定義先: MQQueueManager クラス

構文: Call *MQQueueManager*.**ClearErrorCodes()**

Commit メソッド

最新の同期点以降作業単位の一部として発生したメッセージの書き込みおよび読み取りをコミットします。

定義先: MQQueueManager クラス

構文: Call *MQQueueManager*.**Commit()**

Connect メソッド

MQQueueManager オブジェクトを WebSphere MQ MQI クライアントまたはサーバーにより実キュー・マネージャーに接続します。 このメソッドは、接続を作成するだけでなく、キュー・マネージャー・オブジェクトに照会を出せるようにするために、キュー・マネージャーを開くこともします。

IsConnected を TRUE に設定します。

ActiveX インスタンスあたり最大 1 つの MQQueueManager オブジェクトをキュー・マネージャーに接続することを許可されています。

定義先: MQQueueManager クラス

構文: Call *MQQueueManager*.**Connect()**

Disconnect メソッド

MQQueueManager オブジェクトをキュー・マネージャーから切断します。

IsConnected を FALSE に設定します。

MQQueueManager オブジェクトと関連付けられるすべてのキュー・オブジェクトは使用できなくなり、再オープンできません。

コミットされていない変更 (メッセージの書き込みおよび読み取り) はすべてコミットされます。

定義先: MQQueueManager クラス

構文: Call *MQQueueManager*.**Disconnect()**

MQQueue クラス

このクラスは、WebSphere MQ キューへのアクセスを表します。 この接続は関連付けられた MQQueueManager オブジェクトにより提供されます。 このクラスのオブジェクトが破棄される場合、自動的にクローズされます。

包含

MQQueue クラスは MQQueueManager クラスに含まれます。

作成

New は、新しい MQQueue オブジェクトを作成し、すべてのプロパティーを初期値に設定します。 あるいは、MQQueueManager クラスの AccessQueue メソッドを使用します。

構文

```
Dim que As New MQQueue Set que  = New MQQueue
```

プロパティー

- 1063 ページの『AlternateUserId プロパティー』.
- 1063 ページの『BackoutRequeueName プロパティー』.
- 1063 ページの『BackoutThreshold プロパティー』.
- 1063 ページの『BaseQueueName プロパティー』.

- 1064 ページの『CloseOptions プロパティー』.
- 1064 ページの『CompletionCode プロパティー』.
- 1064 ページの『ConnectionReference プロパティー』.
- 1064 ページの『CreationDateTime プロパティー』.
- 1065 ページの『CurrentDepth プロパティー』.
- 1065 ページの『DefaultInputOpenOption プロパティー』.
- 1065 ページの『DefaultPersistence プロパティー』.
- 1065 ページの『DefaultPriority プロパティー』.
- 1065 ページの『DefinitionType プロパティー』.
- 1065 ページの『DepthHighEvent プロパティー』.
- 1066 ページの『DepthHighLimit プロパティー』.
- 1066 ページの『DepthLowEvent プロパティー』.
- 1066 ページの『DepthLowLimit プロパティー』.
- 1066 ページの『DepthMaximumEvent プロパティー』.
- 1065 ページの『DepthHighEvent プロパティー』.
- 1066 ページの『DepthHighLimit プロパティー』.
- 1066 ページの『DepthLowEvent プロパティー』.
- 1066 ページの『DepthLowLimit プロパティー』.
- 1066 ページの『DepthMaximumEvent プロパティー』.
- 1066 ページの『Description プロパティー』.
- 1066 ページの『DynamicQueueName プロパティー』.
- 1067 ページの『HardenGetBackout プロパティー』.
- 1067 ページの『InhibitGet プロパティー』.
- 1067 ページの『InhibitPut プロパティー』.
- 1067 ページの『InitiationQueueName プロパティー』.
- 1068 ページの『IsOpen プロパティー』.
- 1068 ページの『MaximumDepth プロパティー』.
- 1068 ページの『MaximumMessageLength プロパティー』.
- 1068 ページの『MessageDeliverySequence プロパティー』.
- 1069 ページの『ObjectHandle プロパティー』.
- 1069 ページの『OpenInputCount プロパティー』.
- 1069 ページの『OpenOptions プロパティー』.
- 1069 ページの『OpenOutputCount プロパティー』.
- 1069 ページの『OpenStatus プロパティー』.
- 1069 ページの『ProcessName プロパティー』.
- 1070 ページの『QueueManagerName プロパティー』.
- 1070 ページの『QueueType プロパティー』.
- 1070 ページの『ReasonCode プロパティー』.
- 1070 ページの『ReasonName プロパティー』.
- 1070 ページの『RemoteQueueManagerName プロパティー』.
- 1071 ページの『RemoteQueueName プロパティー』.
- 1071 ページの『ResolvedQueueManagerName プロパティー』.
- 1071 ページの『ResolvedQueueName プロパティー』.

- 1071 ページの『RetentionInterval プロパティー』.
- 1071 ページの『Scope プロパティー』.
- 1071 ページの『ServiceInterval プロパティー』.
- 1072 ページの『ServiceIntervalEvent プロパティー』.
- 1072 ページの『Shareability プロパティー』.
- 1072 ページの『TransmissionQueueName プロパティー』.
- 1072 ページの『TriggerControl プロパティー』.
- 1072 ページの『TriggerData プロパティー』.
- 1073 ページの『TriggerDepth プロパティー』.
- 1073 ページの『TriggerMessagePriority プロパティー』.
- 1073 ページの『TriggerType プロパティー』.
- 1073 ページの『Usage プロパティー』.

方法

- 1073 ページの『ClearErrorCodes メソッド』
- 1073 ページの『Close メソッド』
- 1074 ページの『Get メソッド』
- 1074 ページの『Open メソッド』
- 1075 ページの『Put メソッド』

プロパティー・アクセス

キュー・オブジェクトがキュー・マネージャーに接続していない場合、次のプロパティーを読み取ることができます。

- 1064 ページの『CompletionCode プロパティー』
- 1069 ページの『OpenStatus プロパティー』
- 1070 ページの『ReasonCode プロパティー』

また、次のプロパティーに対して読み取りおよび書き込みを行うことができます。

- 1063 ページの『AlternateUserId プロパティー』
- 1064 ページの『CloseOptions プロパティー』
- 1064 ページの『ConnectionReference プロパティー』
- 1068 ページの『Name プロパティー』
- 1069 ページの『OpenOptions プロパティー』

キュー・オブジェクトがキュー・マネージャーに接続している場合、次のプロパティーをすべて読み取ることができます。

Queue Attribute プロパティー

前のセクションでリストされていないプロパティーは、基礎となる WebSphere MQ キューのすべての属性です。 アクセスできるのは、オブジェクトがキュー・マネージャーに接続されており、ユーザーのユーザー ID がそのキューに対して照会または設定する許可が与えられている場合に限ります。 代替ユーザー ID が設定され、現在のユーザー ID に代替ユーザー ID を使用する許可が与えられている場合、代替ユーザー ID に代わりの権限があるかどうかチェックされます。

プロパティーは所定の QueueType に適するプロパティーでなければなりません。 詳しくは、キューの属性を参照してください。

これらの条件が適用されていない場合、プロパティー・アクセスは、MQCC_FAILED という CompletionCode および次の ReasonCodes のいずれかを設定します。

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode は MQCC_WARNING)

キューのオープン

MQQueue オブジェクトを作成する唯一の方法は、 MQQueueManager AccessQueue メソッドまたは New メソッドを使用することです。 MQQueue オブジェクトをクローズするか削除するまで、あるいは作成中のキュー・マネージャー・オブジェクトを削除するか、またはキュー・マネージャーへの接続が失われるまで、オープンしている MQQueue オブジェクトは オープン (OpenStatus=TRUE) のままになります。 MQQueue CloseOptions プロパティーの値が、MQQueue オブジェクトを削除する場合に起こるクローズ操作の動作を制御します。

MQQueueManager AccessQueue メソッドは、OpenOptions パラメーターを使用してキューをオープンします。 MQQueue.Open メソッドは、OpenOptions プロパティーを使用してキューをオープンします。 WebSphere MQ では、オープン・キュー・プロセスの一部として、ユーザー権限に対し OpenOptions の妥当性検査を行います。

AlternateUserId プロパティー

読み取り/書き込み可能。 このプロパティーのオープン時にキューへのアクセスを検査するために使われる代替ユーザー ID。

このプロパティーはオブジェクトのオープン中 (つまり、 IsOpen が TRUE の場合) には設定できません。

定義先: MQQueue クラス

データ型: 12 文字のストリング (String)

構文: 取得する場合: *altuser$ = MQQueue*.**AlternateUserId**

設定する場合: *MQQueue.***AlternateUserId** = *altuser$*

BackoutRequeueName プロパティー

読み取り専用。 MQI BackOutRequeueQName 属性

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *backoutrequeuename$ = MQQueue*.**BackoutRequeueName**

BackoutThreshold プロパティー

読み取り専用。 MQI BackoutThreshold 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- BackoutThreshold (MQLONG) を参照してください。

構文: 取得する場合: *backoutthreshold & = MQQueue.***BackoutThreshold**

BaseQueueName プロパティー

読み取り専用。 別名が変換されるキュー名。

別名キューにのみ有効。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *baseqname$ = MQQueue*.**BaseQueueName**

CloseOptions プロパティー

読み取り / 書き込み可能。 キューをクローズするときに発生することを制御するために使用されるオプション。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE および MQCO_DELETE_PURGE は、動的キューにのみ有効です。

構文: 取得する場合: *closeopt & = MQQueue*.**CloseOptions**

設定する場合: *MQQueue*.**CloseOptions** = *closeopt &*

CompletionCode プロパティー

読み取り専用。 オブジェクトに対して発行された最新のメソッドまたはプロパティー・アクセスによって設定された完了コードを戻します。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = MQQueue*.**CompletionCode**

ConnectionReference プロパティー

読み取り/書き込み可能。 オブジェクトが属するキュー・マネージャー・オブジェクトを定義します。 キューのオープン時に接続参照を書き込みできません。

定義先: MQQueue クラス

データ型: MQQueueManager

値:

- 活動状態の WebSphere MQ キュー・マネージャー・オブジェクトへの参照。

構文: 設定する場合: *set MQQueue*.**ConnectionReference** = *ConnectionReference*

取得する場合 *set ConnectionReference = MQQueue*.**ConnectionReference**

CreationDateTime プロパティー

読み取り専用。 このキューが作成された日時。

定義先: MQQueue クラス

データ型: タイプ 7 (日時) のバリアントまたは EMPTY

構文: 取得する場合: *datetime = MQQueue*.**CreationDateTime**

CurrentDepth プロパティー

読み取り専用。 現在キューに入っているメッセージの数。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *currentdepth & = MQQueue*.**CurrentDepth**

DefaultInputOpenOption プロパティー

読み取り専用。 OpenOptions が MQOO_INPUT_AS_Q_DEF を指定する場合、キューをオープンする方法を制御します。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

構文: 取得する場合: *defaultinop & = MQQueue*.**DefaultInputOpenOption**

DefaultPersistence プロパティー

読み取り専用。 キュー上のメッセージについてのデフォルト持続性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *defpersistence & = MQQueue*.**DefaultPersistence**

DefaultPriority プロパティー

読み取り専用。 キュー上のメッセージについてのデフォルトの優先順位。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *defpriority & = MQQueue*.**DefaultPriority**

DefinitionType プロパティー

読み取り専用。 キュー定義タイプ。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQDT_PREDEFINED
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

構文: 取得する場合: *deftype & = MQQueue*.**DefinitionType**

DepthHighEvent プロパティー

読み取り専用。 MQI QDepthHighEvent 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *depthhighevent & = MQQueue.***DepthHigh イベント**

DepthHighLimit プロパティー

読み取り専用。 MQI QDepthHighLimit 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *depthhighlimit & = MQQueue.***DepthHighLimit**

DepthLowEvent プロパティー

読み取り専用。 MQI QDepthLowEvent 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *depthlowevent & = MQQueue.***DepthLow イベント**

DepthLowLimit プロパティー

読み取り専用。 MQI QDepthLowLimit 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *depthlowlimit & = MQQueue.***DepthLowLimit**

DepthMaximumEvent プロパティー

読み取り専用。 MQI QDepthMaxEvent 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQEVR_DISABLED
- MQEVR_ENABLED

構文: 取得する場合: *depthmaximumevent & = MQQueue.***DepthMaximum イベント**

Description プロパティー

読み取り専用。 キューの記述。

定義先: MQQueue クラス

データ型: 64 文字のストリング (String)

構文: 取得する場合: *description$ = MQQueue***.Description**

DynamicQueueName プロパティー

読み取り / 書き込み可能。キューがオープンしているときは、読み取り専用。

このプロパティーは、モデル・キューがオープンしているときに使用される動的キュー名を制御します。プロパティー・セット (キューがクローズしているときだけ) として、または MQQueueManager.AccessQueue() へのパラメーターとして、ユーザーがワイルドカードを使って設定することができます。

動的キューの実際の名前は、QueueName を照会すると、検索できます。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

値:

- 有効な WebSphere MQ キュー名。

構文: 設定する場合: *MQQueue***.DynamicQueueName** = *dynamicqueuename$*

取得する場合: *dynamicqueuename$ = MQQueue***.DynamicQueueName**

HardenGetBackout プロパティー

読み取り専用。 正確なバックアウト・カウントを保持するかどうか。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

構文: 取得する場合: *hardengetback & = MQQueue***.HardenGetBackout**

InhibitGet プロパティー

読み取り/書き込み可能。 MQI InhibitGet 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

構文: 取得する場合: *getstatus & = MQQueue***.InhibitGet**

設定する場合: *MQQueue***.InhibitGet** = *getstatus &*

InhibitPut プロパティー

読み取り/書き込み可能。 MQI InhibitPut 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

構文: 取得する場合: *putstatus & = MQQueue***.InhibitPut**

設定する場合: *MQQueue***.InhibitPut** = *putstatus &*

InitiationQueueName プロパティー

読み取り専用。 開始キューの名前。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *initqname$ = MQQueue*.**InitiationQueueName**

IsOpen プロパティー

キューがオープンしているかどうかを戻します。

読み取り専用。

定義先: MQQueue クラス

データ型: ブール (Boolean)

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *open = MQQueue*.**IsOpen**

MaximumDepth プロパティー

読み取り専用。 キューの最大長。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxdepth & = MQQueue*.**MaximumDepth**

MaximumMessageLength プロパティー

読み取り専用。 このキューに許可された最大メッセージ長 (バイト単位)。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *maxmlength & = MQQueue*.**MaximumMessageLength**

MessageDeliverySequence プロパティー

読み取り専用。 メッセージ・デリバリー・シーケンス。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQMDS_PRIORITY
- MQMDS_FIFO

構文: 取得する場合: *messdelseq & = MQQueue*.**MessageDeliverySequence**

Name プロパティー

読み取り/書き込み可能。 MQI Queue 属性。 MQQueue をオープンした後で、このプロパティーを書き込むことはできません。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *name$ = MQQueue*.**name**

設定する場合: *MQQueue*.**name** = *name$*

注 : Visual Basic は、ビジュアル・インターフェースで使用するために "Name" プロパティーを予約しています。 したがって、Visual Basic 内で使用する場合は、小文字、つまり "name" を使用します。

ObjectHandle プロパティー

読み取り専用。 WebSphere MQ キュー・オブジェクトのオブジェクト・ハンドル。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *hobj & = MQQueue.***ObjectHandle**

OpenInputCount プロパティー

読み取り専用。 入力のためのオープン数。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *openincount & = MQQueue***.OpenInputCount**

OpenOptions プロパティー

読み取り/書き込み可能。 キューのオープンに使用されるオプション。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- OpenOptions (MQLONG) を参照してください。

構文: 取得する場合: *openopt & = MQQueue***.OpenOptions**

設定する場合: *MQQueue.***OpenOptions** = *openopt &*

OpenOutputCount プロパティー

読み取り専用。 出力のためのオープン数。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *openoutcount & = MQQueue***.OpenOutputCount**

OpenStatus プロパティー

読み取り専用。 キューがオープンしているかどうかを示します。 初期値は、AccessQueue メソッドの後 TRUE、 または New メソッドの後 FALSE です。

定義先: MQQueue クラス

データ型: ブール (Boolean)

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *status & = MQQueue***.OpenStatus**

ProcessName プロパティー

読み取り専用。 MQI ProcessName 属性。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *procname$ =MQQueue*.**ProcessName**

QueueManagerName プロパティー

読み取り/書き込み可能。 WebSphere MQ キュー・マネージャー名。

定義先: MQQueue クラス

データ型: ストリング (String)

構文: 取得する場合: *QueueManagerName$ = MQQueue*.**QueueManagerName**

設定する場合: *MQQueue*.**QueueManagerName** = *QueueManagerName$*

QueueType プロパティー

読み取り専用。 MQI QType 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQT_ALIAS
- MQQT_LOCAL
- MQQT_MODEL
- MQQT_REMOTE

構文: 取得する場合: *queuetype & = MQQueue*.**QueueType**

ReasonCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コードを戻します。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = MQQueue*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

定義先: MQQueue クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQQueue*.**ReasonName**

RemoteQueueManagerName プロパティー

読み取り専用。 リモート・キュー・マネージャーの名前。 リモート・キューにのみ有効。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *remqmanname$ = MQQueue*.**RemoteQueueManagerName**

RemoteQueueName プロパティー

読み取り専用。 リモート・キュー・マネージャーで認識されているようなキューの名前。 リモート・キューにのみ有効。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *remqname$ = MQQueue*.**RemoteQueueName**

ResolvedQueueManagerName プロパティー

読み取り専用。 ローカル・キュー・マネージャーにとって最終的な宛先として認識されているキュー・マネージャーの名前。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *resqmanname$ = MQQueue*.**ResolvedQueueManagerName**

ResolvedQueueName プロパティー

読み取り専用。 ローカル・キュー・マネージャーにとって最終的な宛先として認識されているキューの名前。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *resqname$ = MQQueue*.**ResolvedQueueName**

RetentionInterval プロパティー

読み取り専用。 キューを保存すべき時期。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *retinterval & = MQQueue*.**RetentionInterval**

Scope プロパティー

読み取り専用。 このキューに対するエントリーがセル・ディレクトリーでも存在するかどうかを制御します。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQSCO_Q_MGR
- MQSCO_CELL

構文: 取得する場合: *scope & = MQQueue*.**Scope**

ServiceInterval プロパティー

読み取り専用。 MQI QServiceInterval 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *serviceinterval & = MQQueue.***ServiceInterval**

ServiceIntervalEvent プロパティー

読み取り専用。 MQI QServiceIntervalEvent 属性。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

構文: 取得する場合: *serviceintervalevent & = MQQueue.***ServiceInterval イベント**

Shareability プロパティー

読み取り専用。 キューの共用性

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

構文: 取得する場合: *shareability & = MQQueue***.Shareability**

TransmissionQueueName プロパティー

読み取り専用。 伝送キュー名。 リモート・キューにのみ有効。

定義先: MQQueue クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *transqname$ = MQQueue***.TransmissionQueueName**

TriggerControl プロパティー

読み取り/書き込み可能。 トリガー制御。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQTC_OFF
- MQTC_ON

構文: 取得する場合: *trigcontrol & = MQQueue***.TriggerControl**

設定する場合: *MQQueue***.TriggerControl** = *trigcontrol &*

TriggerData プロパティー

読み取り/書き込み可能。 トリガー・データです。

定義先: MQQueue クラス

データ型: 64 文字のストリング (String)

構文: 取得する場合: *trigdata$ = MQQueue***.TriggerData**

設定する場合: *MQQueue***.TriggerData** = *trigdata$*

TriggerDepth プロパティー

読み取り/書き込み可能。 トリガー・メッセージが書き込まれる前に、キュー上に存在する必要があるメッセージ数。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *trigdepth & = MQQueue*.**TriggerDepth**

設定する場合: *MQQueue*.**TriggerDepth** = *trigdepth &*

TriggerMessagePriority プロパティー

読み取り/書き込み可能。 トリガーのしきい値メッセージ優先順位。

定義先: MQQueue クラス

データ型: 長整数 (Long)

構文: 取得する場合: *trigmesspriority & = MQQueue*.**TriggerMessagePriority**

設定する場合: *MQQueue*.**TriggerMessagePriority** = *trigmesspriority &*

TriggerType プロパティー

読み取り/書き込み可能。 トリガー・タイプ。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

構文: 取得する場合: *trigtype & = MQQueue*.**TriggerType**

設定する場合: *MQQueue*.**TriggerType** = *Trigtype &*

Usage プロパティー

読み取り専用。 キューの使用対象を示します。

定義先: MQQueue クラス

データ型: 長整数 (Long)

値:

- MQUS_NORMAL
- MQUS_TRANSMISSION

構文: 取得する場合: *usage & = MQQueue*。**使用法**

ClearErrorCodes メソッド

MQQueue クラスおよび MQSession クラスの両方について、 CompletionCode を MQCC_OK に、ReasonCode を MQRC_NONE にリセットします。

定義先: MQQueue クラス

構文: Call *MQQueue*.**ClearErrorCodes()**

Close メソッド

CloseOptions の現在の値を使用して、キューをクローズします。

定義先: MQQueue クラス

構文: Call *MQQueue*.**Close()**

Get メソッド

メッセージをキューから検索します。

このメソッドは、オブジェクトの MQMD 内のフィールドの一部を入力パラメーターとして使用して、MQMessage オブジェクトをパラメーターとして取ります。 特に、MessageId フィールドと CorrelId フィールドを使用するので、必要に応じて必ずこれらのフィールドを設定しておくことが重要です。 これらのフィールドについて詳しくは、 MsgId (MQBYTE24) および CorrelId (MQBYTE24) を参照してください。

このメソッドが失敗した場合、MQMessage オブジェクトは未変更になります。 このメソッドが正常に終了した場合、MQMessage オブジェクトの MQMD およびメッセージ・データ部分は、着信メッセージの MQMD およびメッセージ・データ部分で置き換えられます。 MQMessage 制御プロパティーは、次のように設定されます。

- **MessageLength** は、WebSphere MQ メッセージ長に設定されます。
- **DataLength** は WebSphere MQ メッセージ長に設定されます。
- **DataOffset** はゼロに設定されます。

定義先:
MQQueue クラス

構文: Call *MQQueue*.**Get**(*Message, GetMsgOptions, GetMsgLength*)

Parameters
メッセージ:

検索されるメッセージを表す MQMessage オブジェクト。

GetMsgOptions:

読み込み操作を制御するための任意選択の MQGetMessageOptions オブジェクト。 このパラメーターが指定されていない場合は、デフォルトの MQGetMessageOptions が使用されます。

GetMsgLength:

キューから取得される WebSphere MQ メッセージの最大長を制御するための、オプションの 2 バイト長または 4 バイト長の値。

MQGMO_ACCEPT_TRUNCATED_MSG オプションが指定される 場合、メッセージのサイズが指定された長さを超えると、GET は完了コード MQCC_WARNING および理由コード MQRC_TRUNCATED_MSG_ACCEPTED を出して、正常に終了します。

MessageData はデータの最初の GetMsgLength バイトを保持します。

MQGMO_ACCEPT_TRUNCATED_MSG が指定**されておらず** 、メッセージのサイズが指定される長さを超える場合、 完了コード MQCC_FAILED と共に理由コード MQRC_TRUNCATED_MESSAGE_FAILED が戻されます。

メッセージ・バッファーのコンテンツが未定義の場合、総メッセージ長は、検索されたメッセージの全長に設定されます。

メッセージ長パラメーターが指定されていない場合、メッセージ・バッファーの長さは着信メッセージの最低サイズに自動的に調整されます。

Open メソッド

以下の現行値を使用してキューをオープンします。

1. QueueName
2. QueueManagerName
3. AlternateUserId

4. DynamicQueueName

定義先:
MQQueue クラス

構文: Call *MQQueue*.**Open()**

Put メソッド

メッセージをキューに入れます。

このメソッドは、パラメーターとして MQMessage オブジェクトを使用します。 このオブジェクトのメッセージ記述子 (MQMD) プロパティーは、このメソッドの結果として更新されることがあります。 このメソッドの実行後直ちに指定される値は、WebSphere MQ キューに書き込まれた値です。

Put を完了した後の MQMessage オブジェクトへの変更は、 WebSphere MQ キューにおける実際のメッセージに影響しません。

定義先:
MQQueue クラス

構文: Call *MQQueue*.**Put**(*Message, PutMsgOptions)*

Parameters
メッセージ

書き込まれるメッセージを表す MQMessage オブジェクト

PutMsgOptions

PUT 操作を制御するためのオプションが 含まれる MQPutMessageOptions オブジェクト。 指定されていない場合、デフォルトの PutMessageOptions が使用されます。

MQMessage クラス

このクラスは、WebSphere MQ メッセージを表します。 WebSphere MQ メッセージ記述子 (MQMD) をカプセル化するためのプロパティーが含まれ、アプリケーション定義のメッセージ・データを保持するためのバッファーが提供されます。

このクラスには、ActiveX アプリケーションから MQMessage オブジェクトにデータをコピーする Write メソッドが組み込まれています。 同様に、このクラスには、MQMessage オブジェクトから ActiveX アプリケーションにデータをコピーする Read メソッドが組み込まれています。 このクラスは、バッファーのメモリーの自動割り振り、および自動割り振り解除を管理します。 バッファーは書き込まれるデータのサイズに合わせて拡張するので、 MQMessage オブジェクトを作成する場合に、 アプリケーションがバッファーのサイズを宣言する必要はありません。

バッファー・サイズがそのキューの MaximumMessageLength プロパティーを超える場合、メッセージを WebSphere MQ キューに置くことはできません。

バッファーを構成した後、MQMessage オブジェクトは、MQQueue.Put メソッドを使用して WebSphere MQ キューに書き込みを行うことができます。 このメソッドは、このオブジェクトの MQMD およびメッセージ・データの部分のコピーを受け取り、そのコピーをキューに配置します。 したがって、このアプリケーションは、PUT を実行した後に WebSphere MQ キューにあるメッセージに影響を与えずに、MQMessage オブジェクトを変更または削除することができます。 WebSphere MQ キュー上のメッセージをコピーする場合、 キュー・マネージャーが MQMD のいくつかのフィールドを調整することができます。

MQQueue.Get メソッドを使用して、着信メッセージを MQMessage オブジェクトに読み取ることができます。 これにより、既に MQMessage オブジェクト内に存在している可能性のある MQMD またはメッセージ・データは、着信メッセージからの値で置き換えられます。 このメソッドは、MQMessage オブジェクトのデータ・バッファーのサイズを、着信メッセージ・データのサイズに一致するよう調整します。

包含

メッセージは、MQSession クラスにより含まれます。

作成

New は、MQMessage オブジェクトを作成します。 メッセージ記述子プロパティーは最初デフォルト値に設定され、メッセージ・データのバッファーは空になります。

構文

```
Dim msg As New MQMessage or Set msg = New MQMessage
```

プロパティー

制御プロパティーは次のとおりです。

- 1078 ページの『CompletionCode プロパティー』
- 1078 ページの『DataLength プロパティー』
- 1078 ページの『DataOffset プロパティー』
- 1079 ページの『MessageLength プロパティー』
- 1079 ページの『ReasonCode プロパティー』
- 1079 ページの『ReasonName プロパティー』

メッセージ記述子プロパティーは次のとおりです。

- 1079 ページの『AccountingToken プロパティー』
- 1080 ページの『AccountingTokenHex プロパティー』
- 1080 ページの『ApplicationIdData プロパティー』
- 1080 ページの『ApplicationOriginData プロパティー』
- 1080 ページの『BackoutCount プロパティー』
- 1081 ページの『CharacterSet プロパティー』
- 1081 ページの『CorrelationId プロパティー』
- 1081 ページの『CorrelationIdHex プロパティー』
- 1082 ページの『Encoding プロパティー』
- 1082 ページの『Expiry プロパティー』
- 1083 ページの『Feedback プロパティー』
- 1083 ページの『Format プロパティー』
- 1083 ページの『GroupId プロパティー』
- 1083 ページの『GroupIdHex プロパティー』
- 1084 ページの『MessageData プロパティー』
- 1084 ページの『MessageFlags プロパティー』
- 1084 ページの『MessageId プロパティー』
- 1084 ページの『MessageIdHex プロパティー』
- 1085 ページの『MessageSequenceNumber プロパティー』
- 1085 ページの『MessageType プロパティー』
- 1085 ページの『Offset プロパティー』
- 1085 ページの『OriginalLength プロパティー』
- 1086 ページの『Persistence プロパティー』
- 1086 ページの『Priority プロパティー』
- 1086 ページの『PutApplicationName プロパティー』
- 1086 ページの『PutApplicationType プロパティー』

- 1087 ページの『PutDateTime プロパティー』
- 1087 ページの『ReplyToQueueManagerName プロパティー』
- 1087 ページの『ReplyToQueueName プロパティー』
- 1087 ページの『Report プロパティー』
- 1087 ページの『TotalMessageLength プロパティー』
- 1088 ページの『UserId プロパティー』

方法

- 1088 ページの『ClearErrorCodes メソッド』
- 1088 ページの『ClearMessage メソッド』
- 1088 ページの『Read メソッド』
- 1088 ページの『ReadBoolean メソッド』
- 1088 ページの『ReadByte メソッド』
- 1089 ページの『ReadDecimal2 メソッド』
- 1089 ページの『ReadDecimal4 メソッド』
- 1089 ページの『ReadDouble メソッド』
- 1089 ページの『ReadDouble4 メソッド』
- 1090 ページの『ReadFloat メソッド』
- 1090 ページの『ReadInt2 メソッド』
- 1090 ページの『ReadInt4 メソッド』
- 1090 ページの『ReadLong メソッド』
- 1090 ページの『ReadNullTerminatedString メソッド』
- 1091 ページの『ReadShort メソッド』
- 1091 ページの『ReadString メソッド』
- 1091 ページの『ReadUInt2 メソッド』
- 1091 ページの『ReadUnsignedByte メソッド』
- 1091 ページの『ReadUTF メソッド』
- 1092 ページの『ResizeBuffer メソッド』
- 1092 ページの『Write メソッド』
- 1092 ページの『WriteBoolean メソッド』
- 1093 ページの『WriteByte メソッド』
- 1093 ページの『WriteDecimal2 メソッド』
- 1093 ページの『WriteDecimal4 メソッド』
- 1093 ページの『WriteDouble メソッド』
- 1094 ページの『WriteDouble4 メソッド』
- 1094 ページの『WriteFloat メソッド』
- 1094 ページの『WriteInt2 メソッド』
- 1094 ページの『WriteInt4 メソッド』
- 1094 ページの『WriteLong メソッド』
- 1095 ページの『WriteNullTerminatedString メソッド』
- 1095 ページの『WriteShort メソッド』
- 1095 ページの『WriteString メソッド』
- 1095 ページの『WriteUInt2 メソッド』

- 1096 ページの『WriteUnsignedByte メソッド』
- 1096 ページの『WriteUTF メソッド』

プロパティー・アクセス

いつでもすべてのプロパティーを読み取ることができます。

読み取り / 書き込み可能である DataOffset を除き、制御プロパティーは読み取り専用です。 Message Descriptor プロパティーは、読み取り専用である BackoutCount および TotalMessageLength を除き、すべて読み取り / 書き込み可能です。

しかし、メッセージが WebSphere MQ キューに書き込まれたときに、 一部の MQMD プロパティーはキュー・マネージャーによって変更されることがあります。 それらのプロパティーの変更方法について詳しくは、MQMD のフィールドを参照してください。

データ変換

CharacterSet プロパティーをキュー・マネージャーのコード化文字セット識別子 (MQCCSI_Q_MGR) に設定し、それをストリングに渡すと、 2 進データを WebSphere MQ メッセージに渡すことができます。 chr$ 関数を使用して、文字以外のデータをストリングに設定することができます。

Read メソッドおよび Write メソッドはデータ変換を実行します。 これらのメソッドは、メッセージ記述子から Encoding プロパティーおよび CharacterSet プロパティーにより定義されたように、ActiveX 内部形式と WebSphere MQ メッセージ形式とを変換します。 メッセージを書き込む場合は、Write メソッドを実行する前に、Encoding および CharacterSet に、メッセージの受信側の特性に一致する値を設定します。 メッセージを読み取る場合、これらの値は着信 MQMD の値から設定されるので、通常このステップは不要です。

これは、MQQueue.Get メソッドにより実行される変換の後で起こる追加のデータ変換ステップです。

CompletionCode プロパティー

読み取り専用。 このオブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された WebSphere MQ 完了コードを戻します。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = MQMessage*.**CompletionCode**

DataLength プロパティー

読み取り専用。 このプロパティーは値を戻します。

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Read メソッドの前に使用して、予測文字数がバッファーに実際存在することをチェックできます。

初期値はゼロです。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *bytesleft & = MQMessage*.**DataLength**

DataOffset プロパティー

読み取り/書き込み可能。 メッセージ・オブジェクトのメッセージ・データ部分内の現在位置。

この値は、メッセージ・データ・バッファーの最初からのバイト・オフセットとして表されます。バッファーの先頭文字はゼロという DataOffset 値に対応します。

Read メソッドまたは Write メソッドは、DataOffset が参照する文字で、操作を開始します。 これらのメソッドは、この位置から順にバッファーのデータを処理し、DataOffset を更新して、処理される最終バイトの直後にあるバイト (ある場合) を指します。

DataOffset は、ゼロから MessageLength までの範囲の値だけを取ることができます。 DataOffset = MessageLength の場合、DataOffset は最後を指すので、これがバッファーの最初の無効文字になります。 Write メソッドは、この状況で許可されます。つまり、バッファー内のデータを拡張し、追加されるバイト数に合わせて MessageLength を増やします。 バッファーの最後を超えた読み取りは無効です。

初期値はゼロです。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *currpos & = MQMessage*.**DataOffset**

設定する場合: *MQMessage*.**DataOffset** = *currpos &*

MessageLength プロパティー

読み取り専用。 DataOffset の値に関係なく、メッセージ・オブジェクトのメッセージ・データ部分の全長を文字単位で戻します。

初期値はゼロです。 このメッセージ・オブジェクトを参照 した Get メソッドの呼び出し後に、着信メッセージ長に設定されます。 アプリケーションが Write メソッドを使用してデータをオブジェクトに追加する場合、メッセージ長は増分されます。 Read メソッドにより影響を受けません。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *msglength & = MQMessage*.**MessageLength**

ReasonCode プロパティー

読み取り専用。 このオブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コードを戻します。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = MQMessage*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。
定義先: MQMessage クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQMessage*.**ReasonName**

AccountingToken プロパティー

読み取り/書き込み可能。 MQMD AccountingToken - メッセージ識別子文脈の一部。

初期値はすべてヌルです。

定義先: MQMessage クラス

データ型: 32 文字のストリング (String)

構文: 取得する場合: *actoken$ = MQMessage*.**AccountingToken**

設定する場合: *MQMessage.***AccountingToken** = *actoken$*

どのような場合に、AccountingToken プロパティーの代わりに AccountingTokenHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

AccountingTokenHex プロパティー

読み取り/書き込み可能。 MQMD AccountingToken - メッセージ識別子文脈の一部。

それぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、"6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 64 個の 16 進文字を指定する必要があります。

初期値は "0...0" です。

定義先: MQMessage クラス

データ型: 32 個の ASCII 文字を表す 64 個の 16 進文字のストリング (String)

構文: 取得する場合: *actokenh$ = MQMessage*.**AccountingTokenHex**

設定する場合: *MQMessage.***AccountingTokenHex** = *actokenh$*

どのような場合に、AccountingToken プロパティーの代わりに AccountingTokenHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

ApplicationIdData プロパティー

読み取り/書き込み可能。 MQMD ApplIdentityData - メッセージ識別子文脈の一部。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 32 文字のストリング (String)

構文: 取得する場合: *applid$ = MQMessage*.**ApplicationIdData**

設定する場合: *MQMessage*.**ApplicationIdData** = *applid$*

ApplicationOriginData プロパティー

読み取り/書き込み可能。 MQMD ApplOriginData - メッセージの起点文脈の一部。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 4 文字のストリング (String)

構文: 取得する場合: *applor$ = MQMessage*.**ApplicationOriginData**

設定する場合: *MQMessage*.**ApplicationOriginData** = *applor$*

BackoutCount プロパティー

読み取り専用。 MQMD BackoutCount。

初期値は 0 です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *backoutct & = MQMessage*.**BackoutCount**

CharacterSet プロパティー

読み取り/書き込み可能。 MQMD CodedCharSetId。

初期値は特殊値 **MQCCSI_Q_MGR** です。

CharacterSet が **MQCCSI_Q_MGR** に設定されている場合、 WriteString メソッドの文字変換には、現行ロケールのコード・ページが使用されます。 サーバー・アプリケーションの場合、使用されるコード・ページは、キュー・マネージャーのコード・ページです。 クライアント・アプリケーションの場合は、デフォルトの現行ロケールのコード・ページです。

以下に例を示します。

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

'n' が 0 以上で 255 以下の場合、結果はバッファーに書き込まれる 'n' という単一のバイトの値になります。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *:30ccid& = MQMessage* **.CharacterSet**

設定する場合: *MQMessage*.**CharacterSet**= *ccid &*

例

コード・ページ 437 にストリングを書き込みたい場合は、次を発行します。

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

WriteString 呼び出しを発行する前に、CharacterSet に希望の値を設定します。

CorrelationId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、メッセージの MQMD に組み込まれる CorrelationId。 これはまた、キューからメッセージを取得するときに突き合わされる ID でもあります。

初期値はヌルです。

定義先: MQMessage クラス

データ型: 24 文字のストリング (String)

構文: 取得する場合: *correlid$ = MQMessage*.**CorrelationId** 設定する場合: *MQMessage*.**CorrelationId** = *correlid$*

どのような場合に、CorrelationId プロパティーの代わりに CorrelationIdHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

CorrelationIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、メッセージの MQMD に組み込まれる CorrelationId。 これはまた、キューからメッセージを取得するときに突き合わされる CorrelationId でもあります。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、"6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0...0" です。

定義先: MQMessage クラス

データ型: 24 個の ASCII 文字を表す 48 個の 16 進文字のストリング (String)

構文: 取得する場合: *correlidh$ = MQMessage*.**CorrelationIdHex**

設定する場合: *MQMessage*.**CorrelationIdHex** = *correlidh$*

どのような場合に、CorrelationId プロパティーの 代わりに CorrelationIdHex を使用する必要があるかについては、 1040 ページの『メッセージ記述子のプロパティー』を参照してください。

Encoding プロパティー

読み取り/書き込み可能。 アプリケーション・メッセージ・データの数値に使用される表示を識別する MQMD フィールド。

初期値は特殊値 MQENC_NATIVE で、プラットフォームによって変わります。

このプロパティーは、次のメソッドにより使用されます。

- ReadDecimal2 メソッド
- ReadDecimal4 メソッド
- ReadDouble メソッド
- ReadDouble4 メソッド
- ReadFloat メソッド
- ReadInt2 メソッド
- ReadInt4 メソッド
- ReadLong メソッド
- ReadShort メソッド
- ReadUInt2 メソッド
- WriteDecimal2 メソッド
- WriteDecimal4 メソッド
- WriteDouble メソッド
- WriteDouble4 メソッド
- WriteFloat メソッド
- WriteInt2 メソッド
- WriteInt4 メソッド
- WriteLong メソッド
- WriteShort メソッド
- WriteUInt2 メソッド

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *encoding & = MQMessage*。**エンコード** 設定: *MQMessage*。**エンコード** = *encoding &*

メッセージ・バッファーにデータを書き込む準備が整っている場合、受信側キュー・マネージャーが独自のデータ変換を実行できなければ、このフィールドを受信側キュー・マネージャーのプラットフォームの特性に一致するよう設定する必要があります。

Expiry プロパティー

読み取り/書き込み可能。 MQMD 満了時間フィールドで、1/10 秒で予測されます。

初期値は特殊値 MQEI_UNLIMITED です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *expiry & = MQMessage*。**有効期限**

設定する場合: *MQMessage*。**有効期限** = *expiry &*

Feedback プロパティー

読み取り/書き込み可能。 MQMD feedback フィールド。

初期値は特殊値 MQFB_NONE です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- Feedback を参照してください。

構文: 取得する場合: *feedback & = MQMessage*。**フィードバック**

設定する場合: *MQMessage*。**フィードバック** = *feedback &*

Format プロパティー

読み取り/書き込み可能。 MQMD format フィールド。 メッセージ・データの性質を記述する組み込み形式またはユーザー定義の形式の名前を指定します。

初期値は特殊値 MQFMT_NONE です。

定義先: MQMessage クラス

データ型: 8 文字のストリング (String)

構文: 取得する場合: *format$ = MQMessage*.**Format**

設定する場合: *MQMessage*.**Format** = *format$*

GroupId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる GroupId。 これはまた、キューからメッセージを取得するときに突き合わされる ID でもあります。 初期値はすべてヌルです。

定義先:
MQMessage クラス

データ型:
24 文字のストリング (String)

構文: 取得する場合: *groupid$ = MQMessage*.**GroupId**

設定する場合: *MQMessage*.**GroupId** = *groupid$*

どのような場合に、GroupId プロパティーの代わりに GroupIdHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

GroupIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる GroupId。 これはまた、キューからメッセージを取得するときに突き合わされる ID でもあります。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、 "6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0...0" です。

定義先:
MQMessage クラス

データ型:
24 ASCII 文字を表す 48 の 16 進文字のストリング (String)

構文: 取得する場合: *groupidh$ = MQMessage*.**GroupIdHex**

設定する場合: *MQMessage*.**GroupIdHex** = *groupidh$*

どのような場合に、GroupId プロパティーの代わりに GroupIdHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

MessageData プロパティー

読み取り/書き込み可能。 メッセージの内容全体を文字ストリングとして検索または設定します。

定義先: MQMessage クラス

データ型: バリアント (Variant)

注 : このプロパティーにより使用されるデータ型はバリアントですが、MQAX ではストリングのバリアント型になるよう予測されます。 この型以外のバリアントで渡す場合、エラー MQRC_OBJECT_TYPE_ERROR が戻されます。

構文: 取得する場合: *String$ = MQMessage*.**MessageData**

設定する場合: *MQMessage*.**MessageData** = *String$*

MessageFlags プロパティー

読み取り / 書き込み可能。 セグメント化制御情報を指定するメッセージ・フラグ。 初期値は 0 です。

定義先:
MQMessage クラス

データ型:
Long

値:
MsgFlags (MQLONG) を参照してください。

構文: 取得する場合: *messageflags & = MQMessage*.**MessageFlags**

設定する場合: *MQMessage*.**MessageFlags** = *messageflags &*

MessageId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、メッセージの MQMD に組み込まれる MessageId。 これはまた、キューからメッセージを取得するときに突き合わされる ID でもあります。

初期値はすべてヌルです。

定義先: MQMessage クラス

データ型: 24 文字のストリング (String)

構文: 取得する場合: *messageid$ = MQMessage*.**MessageId**

設定する場合: *MQMessage*.**MessageId** = *messageid$*

どのような場合に、MessageId プロパティーの代わりに MessageIdHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

MessageIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、メッセージの MQMD に組み込まれる MessageId。 これはまた、キューからメッセージを取得するときに突き合わされる MessageId でもあります。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、"6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0...0" です。

定義先: MQMessage クラス

データ型: 24 個の ASCII 文字を表す 48 個の 16 進文字のストリング (String)

構文: 取得する場合: *messageidh$ = MQMessage*.**MessageIdHex**

設定する場合: *MQMessage*.**MessageIdHex** = *messageidh$*

どのような場合に、MessageId プロパティーの代わりに MessageIdHex を使用する必要があるのかについては、1040 ページの『メッセージ記述子のプロパティー』を参照してください。

MessageSequenceNumber プロパティー

読み取り / 書き込み可能。 グループ内のメッセージを識別するシーケンス情報。 初期値は 1 です。

定義先:
MQMessage クラス

データ型:
Long

値:
MsgSeqNumber (MQLONG) を参照してください。

構文: 取得する場合: *sequencenumber & = MQMessage*.**SequenceNumber**

設定する場合: *MQMessage*.**SequenceNumber** = *sequencenumber &*

MessageType プロパティー

読み取り/書き込み可能。 MQMD MsgType フィールド。

初期値は MQMT_DATAGRAM です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- MsgType (MQLONG) を参照してください。

構文: 取得する場合: *msgtype & = MQMessage*.**MessageType**

設定する場合: *MQMessage*.**MessageType** = *msgtype &*

Offset プロパティー

読み取り / 書き込み可能。 セグメント化したメッセージのオフセット。 初期値は 0 です。

定義先:
MQMessage クラス

データ型:
Long

値:
Offset (MQLONG) を参照してください。

構文: 取得する場合: *offset & = MQMessage*.**Offset**

設定する場合: *MQMessage*.**Offset** = *offset &*

OriginalLength プロパティー

読み取り / 書き込み可能。 分割されたメッセージの元の長さ。 初期値は MQOL_UNDEFINED です。

定義先:
MQMessage クラス

データ型:
Long

値:
OriginalLength (MQLONG) を参照してください。

構文: 取得する場合: *originallength & = MQMessage*.**OriginalLength**

設定する場合: *MQMessage*.**OriginalLength** = *originallength &*

Persistence プロパティー

読み取り/書き込み可能。 メッセージの永続設定。

初期値は MQPER_PERSISTENCE_AS_Q_DEF です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *persist & = MQMessage*。**パーシスタンス**

設定する場合: *MQMessage*。**パーシスタンス** = *persist &*

Priority プロパティー

読み取り/書き込み可能。 メッセージの優先順位。

初期値は特殊値 MQPRI_PRIORITY_AS_Q_DEF です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *priority & = MQMessage*。**優先順位**

設定する場合: *MQMessage*。**優先順位** = *priority &*

PutApplicationName プロパティー

読み取り/書き込み可能。 MQMD PutApplName - メッセージの起点文脈の一部。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 28 文字のストリング (String)

構文: 取得する場合: *putapplnm$ = MQMessage*.**PutApplicationName**

設定する場合: *MQMessage*.**PutApplicationName** = *putapplnm$*

PutApplicationType プロパティー

読み取り/書き込み可能。 MQMD PutApplType - メッセージの起点文脈の一部。

初期値は MQAT_NO_CONTEXT です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- PutApplType (MQLONG) を参照してください。

構文: 取得する場合: *putappltp & = MQMessage*.**PutApplicationType**

設定する場合: *MQMessage*.**PutApplicationType** = *putappltp &*

PutDateTime プロパティー

読み取り/書き込み。 このプロパティーは、MQMD PutDate および PutTime フィールドを結合しています。これは、メッセージを書き込んだ時期を示すメッセージの起点文脈の一部です。

ActiveX 拡張機能は、ActiveX 日時形式と WebSphere MQ MQMD で使用される日時形式とを変換します。無効な PutDate または PutTime を持つメッセージを受け取る場合、Get メソッドの後の PutDateTime プロパティーは EMPTY に設定されます。

初期値は EMPTY です。

定義先: MQMessage クラス

データ型: タイプ 7 (日時) のバリアントまたは EMPTY

構文: 取得する場合: *datetime = MQMessage*.**PutDateTime**

設定する場合: *MQMessage*.**PutDateTime** = *datetime*

ReplyToQueueManagerName プロパティー

読み取り/書き込み可能。 MQMD ReplyToQMgr フィールド。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *replytoqmgr$ = MQMessage*.**ReplyToQueueManagerName**

設定する場合: *MQMessage*.**ReplyToQueueManagerName** = *replytoqmgr$*

ReplyToQueueName プロパティー

読み取り/書き込み可能。 MQMD ReplyToQ フィールド。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *replytoq$ = MQMessage*.**ReplyToQueueName**

設定する場合: *MQMessage*.**ReplyToQueueName** = *replytoq$*

Report プロパティー

読み取り/書き込み可能。 メッセージのレポート・オプション。

初期値は MQRO_NONE です。

定義先: MQMessage クラス

データ型: 長整数 (Long)

値:

- Report を参照してください。

構文: 取得する場合: *report & = MQMessage*.**レポート**

設定する場合: *MQMessage*。**レポート** = *report &*

TotalMessageLength プロパティー

読み取り専用。 MQGET が受け取る最後のメッセージの長さを検索します。 メッセージが切り捨てられない場合には、値は MessageLength プロパティーの値と同じです。

定義先: MQMessage クラス

データ型: 長整数 (Long)

構文: 取得する場合: *totalmessagelength & = MQMessage*.**TotalMessageLength**

UserId プロパティー

読み取り/書き込み可能。 MQMD UserIdentifier - メッセージ識別子文脈の一部。

初期値はすべてブランクです。

定義先: MQMessage クラス

データ型: 12 文字のストリング (String)

構文: 取得する場合: *userid$ = MQMessage*.**UserId**

設定する場合: *MQMessage*.**UserId** = *userid$*

ClearErrorCodes メソッド

MQMessage クラスおよび MQSession クラスの両方について、 CompletionCode を MQCC_OK に、 ReasonCode を MQRC_NONE にリセットします。

定義先: MQMessage クラス

構文: Call *MQMessage*.**ClearErrorCodes()**

ClearMessage メソッド

このメソッドは、MQMessage オブジェクトのデータ・バッファー部分をクリアします。 データ・バッファーのメッセージ・データは、MessageLength、 DataLength、および DataOffset がすべてゼロに設定されているため、どれも失われます。

メッセージ記述子 (MQMD) 部分は影響を受けません。アプリケーションは、MQMessage オブジェクトを再使用する前に、一部の MQMD フィールドを変更しなければならないことがあります。 MQMD フィールドの設定を元に戻したい場合、「新規」を使用して、オブジェクトを新しいインスタンスで置き換える必要があります。

定義先: MQMessage クラス

構文: Call *MQMessage*.**ClearMessage()**

Read メソッド

メッセージ・バッファーからバイト配列に一連のバイトを読み込みます。 読み込んだバイト数分だけ DataOffset が増分され、Data が減分されます。

定義先:
MQMessage クラス

構文: Data = MQMessage.**Read**(len &)

パラメーター:
len &: Long。 読み取られるバイト単位のデータの長さを表す。

ReadBoolean メソッド

メッセージ・バッファーの現在位置から 1 バイトのブール値を読み取り、 2 バイトの TRUE(-1)/FALSE(0) ブール値を戻します。 DataOffset は 1 バイトずつ増分され、Data Length は 1 バイトずつ減分されます。

定義先:
MQMessage クラス

構文: *value = MQMessage*.**ReadBoolean**

ReadByte メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 1 バイトを読み取り、 その値を -128 から 127 までの範囲の Integer 型 (符号付き 2 バイト) 整数値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 1 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 1 ずつ増分され、DataLength は 1 ずつ減分されます。

メッセージ・データのバイトは、符号付き 2 進整数であると想定されます。

定義先: MQMessage クラス

構文: *integerv% = MQMessage*.**ReadByte**

ReadDecimal2 メソッド

2 バイトのパック 10 進数を読み取り、その値を符号付き 2 バイト整数値として戻します。 DataOffset は 2 バイトずつ増分され、Data Length は 2 バイトずつ減分されます。

定義先:
MQMessage クラス

構文: *value% = MQMessage*.**ReadDecimal2**

ReadDecimal4 メソッド

4 バイトのパック 10 進数を読み取り、その値を符号付き 4 バイトの整数値として戻します。 DataOffset は 4 バイトずつ増分され、Data Length は 4 バイトずつ減分されます。

定義先:
MQMessage クラス

構文: Call *value & = MQMessage*.**ReadDecimal4**

ReadDouble メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 8 バイトを読み取り、 その値を Double 型 (符号付き 8 バイト) 浮動小数点値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 8 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 8 ずつ増分され、DataLength は 8 ずつ減分されます。

メッセージ・データの 8 文字は、2 進浮動小数点数であると想定されます。 エンコード方式は MQMessage.Encoding プロパティーで指定されます。 System/360 形式からの変換はサポートされていません。

定義先: MQMessage クラス

構文: *doublev# = MQMessage*.**ReadDouble**

ReadDouble4 メソッド

ReadDouble4 および WriteDouble4 メソッドは、 ReadFloat および WriteFloat の代替です。 これは、4 バイトの IEEE 浮動小数点形式に変換するには大きすぎる 4 バイトの System/390 浮動小数点のメッセージ値をサポートしているためです。

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 4 バイトを読み取り、 その値を Double 型 (符号付き 8 バイト) 浮動小数点値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 4 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分され、DataLength は 4 ずつ減分されます。

メッセージ・データの 4 文字は、2 進浮動小数点数であると想定されます。 エンコード方式は MQMessage.Encoding プロパティーで指定されます。 System/360 形式からの変換はサポートされていません。

定義先: MQMessage クラス

構文: *doublev# = MQMessage.***ReadDouble4**

ReadFloat メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 4 バイトを読み取り、 その値を Single 型 (符号付き 4 バイト) 浮動小数点値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 4 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分され、DataLength は 4 ずつ減分されます。

メッセージ・データの 4 文字は、浮動小数点数であると想定されます。 エンコード方式は MQMessage.Encoding プロパティーで指定されます。 System/360 形式からの変換はサポートされていません。

定義先: MQMessage クラス

構文: *singlev! = MQMessage***.ReadFloat**

ReadInt2 メソッド

このメソッドは、ReadShort メソッドと同じです。

構文: *integerv% = MQMessage***.ReadInt2**

ReadInt4 メソッド

このメソッドは、ReadLong メソッドと同じです。

構文: *bigint & = MQMessage*。**読み取り間隔 4**

ReadLong メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 4 バイトを読み取り、 その値を Long 型 (符号付き 4 バイト) 整数値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 4 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分され、DataLength は 4 ずつ減分されます。

メッセージ・データの 4 文字は、2 進整数であると想定されます。エンコード方式は MQMessage.Encoding プロパティーで指定されます。

定義先: MQMessage クラス

構文: *bigint & = MQMessage***.ReadLong**

ReadNullTerminatedString メソッド

このメソッドは、ストリングにヌル文字が組み込まれている場合に ReadString の代わりに使用するためのものです。

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる指定のバイト数分を読み取り、 その値を ActiveX ストリングとして戻します。 ストリングの最後の前にヌル値が組み込まれている場合、戻されるストリングの長 さは減らされ、ヌル値の前の文字だけを反映します。

ストリングにヌル文字が組み込まれているかどうかに関係なく、指定された値の分だけ DataOffset が増分され、DataLength が減分されます。

メッセージ・データ内の文字は、MQMessage.CharacterSet プロパティーによって指定されているコード・ページのストリングであると想定されます。 アプリケーションでは、ActiveX 表示への変換が実行されます。

定義先:
MQMessage クラス

構文: *string $ = MQMessage*.**ReadNullTerminatedString(length &)**

パラメーター:
長さ & *Long*。 バイト単位のストリング・フィールドの長さを表す。

ReadShort メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 2 バイトを読み取り、 その値を Integer 型 (符号付き 2 バイト) 値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 2 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 2 ずつ増分され、DataLength は 2 ずつ減分されます。

メッセージ・データの 2 文字は、2 進整数であると想定されます。 エンコード方式は MQMessage.Encoding プロパティーで指定されます。

定義先: MQMessage クラス

構文: *integerv% = MQMessage*.**ReadShort**

ReadString メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる n バイトを読み取り、 その値を ActiveX ストリングとして戻します。

このメソッドを発行したときに MQMessage.DataLength が n 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は n ずつ増分され、DataLength は n ずつ減分されます。

メッセージ・データの n 文字は、MQMessage.CharacterSet プロパティーによって指定されているコード・ページのストリングであると想定されます。 アプリケーションでは、ActiveX 表示への変換が実行されます。

定義先: MQMessage クラス

構文: *stringv $= MQMessage*.**ReadString**(長さ *&*)

パラメーター
length & Long。 バイト単位のストリング・フィールドの長さを表す。

ReadUInt2 メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照する文字で始まる 2 バイトを読み取り、 その値を Long 型 (符号付き 4 バイト) 整数値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 2 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 2 ずつ増分され、DataLength は 2 ずつ減分されます。

メッセージ・データの 2 バイトは、符号なし 2 進整数であると想定されます。 エンコード方式は MQMessage.Encoding プロパティーで指定されます。

定義先: MQMessage クラス

構文: *bigint & = MQMessage*。**ReadUInt2**

ReadUnsignedByte メソッド

このメソッドは、 メッセージ・データ・バッファーから、 DataOffset が参照するバイトで始まる 1 バイトを読み取り、 その値を 0 から 255 までの範囲の Integer 型 (符号付き 2 バイト) 整数値として戻します。

このメソッドを発行したときに MQMessage.DataLength が 1 未満であれば、このメソッドは失敗します。

このメソッドが正常に終了した場合、DataOffset は 1 ずつ増分され、DataLength は 1 ずつ減分されます。

メッセージ・データの 1 文字は、符号なし 2 進整数であると見なされます。

定義先: MQMessage クラス

構文: *integerv% = MQMessage*.**ReadUnsignedByte**

ReadUTF メソッド

このメソッドは、メッセージから、DataOffset が参照するバイトで始まる UTF 書式制御ストリングを読み取り、 その値を ActiveX ストリングとして戻します。 このメッセージ内のストリングは、2 バイトの長さからなり、その後に文字データが続きます。

このメソッドを発行したときに MQMessage.DataLength がストリングの長さ未満であ れば、このメソッドは失敗します。

このメソッドが正常に終了した場合、そのストリングの長さ分だけ DataOffset は増分され、DataLength は減分されます。

定義先:
MQMessage クラス

構文: *value$ = MQMessage*.**ReadUTF**

ResizeBuffer メソッド

このメソッドは、メッセージ・データ・バッファーを保持するために現在内部的に割り振られている記憶域の容量を更新します。 アプリケーションが大量のメッセージを処理しようとしていることを認識している場合は、十分なバッファーが割り振られていることを確認できるという点で、アプリケーションに自動バッファー管理の制御権が与えられます。 アプリケーションはこの呼び出しを使用する必要はありません。つまり、呼び出しを使用しない場合、自動バッファー管理コードにより、バッファーのサイズが適するサイズに拡張されます。

バッファーを現在の MessageLength より小さくなるようサイズ変更する場合、データが消失するリスクを負うことになります。 データが消失する場合、このメソッドは MQCC_WARNING の CompletionCode および MQRC_DATA_TRUNCATED の ReasonCode を戻します。

バッファーを **DataOffset** プロパティーの値より小さくなるようサイズ変更する場合、次のようになります。

- **DataOffset** プロパティーは、新規バッファーの最後を指すよう変更されます。
- **DataLength** プロパティーはゼロに設定されます。
- **MessageLength** プロパティーは、新規バッファー・サイズに変更されます。

定義先:
MQMessage クラス

構文: *MQMessage*.**ResizeBuffer**(長さ *&*)

パラメーター:
Length& は長整数 (Long)。 文字に必要なサイズを表す。

Write メソッド

バイト配列からメッセージ・バッファー内の Data Offset が参照する位置に一連のバイトを書き込みます。必要なら、バッファーの長さ (MQMessage.MQMessageLength) が、 バイト配列を完全に入れることができるように拡張されます。 このメソッドが正常に終了した場合、DataOffset は書き込まれたバイト数分だけ増分されます。

定義先:
MQMessage クラス

構文: Call *MQMessage*.**Write**(value)

パラメーター:
data: はバイト配列またはバイト配列に対するバリアント参照

WriteBoolean メソッド

2 バイトのブール値からメッセージ・バッファー内の現在位置に 1 バイトのブール値を書き込みます。DataOffset は 1 バイトずつ増分されます。

定義先:
MQMessage クラス

構文: Call *MQMessage*.**WriteBoolean**(value)

パラメーター:
value: はブール (2 バイト)。 書き込まれる値を表す。

WriteByte メソッド

このメソッドは、符号付き 2 バイトの整数値を取り、 メッセージ・データ・バッファー内の DataOffset が参照する位置に、 その値を 1 バイトの 2 進数として書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 1 バイトずつ増分されます。

指定される値は、-128 から 127 の範囲でなければなりません。 値がこの範囲でない場合、メソッドは、CompletionCode MQCC_FAILED および ReasonCode MQRC_WRITE_VALUE_ERROR を戻します。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteByte**(*value%*)

パラメーター: *value%* は整数 (Integer)。 書き込まれる値を表す。

WriteDecimal2 メソッド

2 バイトのパック 10 進数として符号付き 2 バイト整数を書き込みます。 DataOffset は 2 バイトずつ増分されます。

定義先:
MQMessage クラス

構文: Call *MQMessage*.**WriteDecimal2**(value%)

パラメーター:
value% は整数 *(Integer)*。 書き込まれる値を表す。

WriteDecimal4 メソッド

4 バイトのパック 10 進数として符号付き 4 バイト整数を書き込みます。 DataOffset は 4 バイトずつ増分されます。

定義先:
MQMessage クラス

構文: *MQMessage*.**WritedDecimal4**(値 &)

パラメーター:
value & Long。 書き込まれる値を表す。

WriteDouble メソッド

このメソッドは、符号付き 8 バイトの浮動小数点値を取り、メッセージ・データ・バッファー内の DataOffset が参照する位置を起点として、8 バイトの浮動小数点数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 8 ずつ増分されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する浮動小数点表示に変換します。 *System/360* 形式への変換はサポートされていません。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteDouble**(*value#*)

パラメーター:
value# は Double。 書き込まれる値を表す。

WriteDouble4 メソッド

どのような場合に、ReadFloat および WriteFloat の 代わりに ReadDouble4 および WriteDouble4 を使用する必要があるかについては、 1089 ページの『ReadDouble4 メソッド』を参照してください。

このメソッドは、符号付き 8 バイトの浮動小数点値を取り、メッセージ・データ・バッファー内の DataOffset が参照する位置を起点として、 4 バイトの浮動小数点数としてその値を書き込みます。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分されます。

この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する浮動小数点表示に変換します。 *System/360* 形式への変換はサポートされていません。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteDouble4(value#)**

パラメーター: *value#* は Double。 書き込まれる値を表す。

WriteFloat メソッド

このメソッドは、符号付き 4 バイトの浮動小数点値を取り、メッセージ・データ・バッファー内の DataOffset が参照する文字を起点として、4 バイトの浮動小数点数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する 2 進数表示に変換します。 *System/360* 形式への変換はサポートされていません。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteFloat**(*value!*)

パラメーター 値*!* Float。 書き込まれる値を表す。

WriteInt2 メソッド

このメソッドは、WriteShort メソッドと同じです。

構文: Call *MQMessage*.**WriteInt2**(*value%*)

パラメーター *value%* は整数 (Integer)。 書き込まれる値を表す。

WriteInt4 メソッド

このメソッドは、WriteLong メソッドと同じです。

構文: *MQMessage* **を呼び出します。WriteInt4**(値 *&*)

パラメーター *value &* Long。 書き込まれる値を表す。

WriteLong メソッド

このメソッドは、符号付き 4 バイトの整数値を取り、 メッセージ・データ・バッファー内の DataOffset が参照するバイトを起点として、 4 バイトの 2 進数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 4 ずつ増分されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する 2 進数表示に変換します。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteLong**(値 *&*)

パラメーター *value &* Long。 書き込まれる値を表す。

WriteNullTerminatedString メソッド

このメソッドは、通常の WriteString を実行し、指定される長さまで残りのバイトにヌルを組み込みます。 初期書き込みストリングが書き込むバイト数が指定される長さと同じ場合、ヌルは書き込まれません。 バイト数が指定される長さを超える場合、エラー (理由コード MQRC_WRITE_VALUE_ERROR) が設定されます。

このメソッドが正常に終了した場合、DataOffset は指定された長さ分だけ増分されます。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteNullTerminatedString***(value$, length &)*

パラメーター:
value$ はストリング (String)。 書き込まれる値を表す。

length& は長整数 (Long)。 バイト単位のストリング・フィールドの長さを表す。

WriteShort メソッド

このメソッドは、符号付き 2 バイトの整数値を取り、 メッセージ・データ・バッファー内の DataOffset が参照するバイトを起点として、 2 バイトの 2 進数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 2 ずつ増分されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する 2 進数表示に変換します。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteShort**(*value%*)

パラメーター *value%* は整数 (Integer)。 書き込まれる値を表す。

WriteString メソッド

このメソッドは、ActiveX ストリングを取り、 メッセージ・データ・バッファー内の DataOffset が参照するバイトを起点として、 その値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset はストリングの長さのバイト分だけ増分されます。

このメソッドは、MQMessage.CharacterSet プロパティーが指定するコード・ページに文字を変換します。

定義先: MQMessage クラス

構文: Call *MQMessage*.**WriteString**(*value$*)

パラメーター: *value$* はストリング (String)。 書き込まれる値を表す。

WriteUInt2 メソッド

このメソッドは、符号付き 4 バイトの整数値を取り、 メッセージ・データ・バッファー内の DataOffset が参照するバイトを起点として、 2 バイトの符号なし 2 進数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 2 ずつ増分されます。

このメソッドは、MQMessage.Encoding プロパティーが指定する 2 進数表示に変換します。 指定される値は、0 から 2**16-1 の範囲でなければなりません。 値がこの範囲でない場合メソッドは、CompletionCode MQCC_FAILED および ReasonCode MQRC_WRITE_VALUE_ERROR を戻します。

定義先: MQMessage クラス

構文: *MQMessage* **を呼び出します。** **WriteUInt2**(値 *&*)

パラメーター *value &* Long。 書き込まれる値を表す。

WriteUnsignedByte メソッド

このメソッドは、符号付き 2 バイトの整数値を取り、 メッセージ・データ・バッファー内の DataOffset が参照する文字を起点として、1 バイトの符号なし 2 進数としてその値を書き込みます。 この操作によって、バッファー内のその位置にあるデータが書き換えられ、 必要であればバッファーの長さ (MQMessage.MessageLength) が拡張されます。

このメソッドが正常に終了した場合、DataOffset は 1 ずつ増分されます。

指定される値は、0 から 255 の範囲でなければなりません。 値がこの範囲でない場合メソッドは、CompletionCode MQCC_FAILED および ReasonCode MQRC_WRITE_VALUE_ERROR を戻します。

定義先:
MQMessage クラス

構文: Call *MQMessage*.**WriteUnsignedByte**(*value%*)

パラメーター *value%* は整数 (Integer)。 書き込まれる値を表す。

WriteUTF メソッド

このメソッドは、ActiveX ストリングを取り、 メッセージ・データ・バッファーの現在位置に UTF 形式でその値を書き込みます。 書き込まれたデータは、2 バイトの長さからなり、その後に文字データが続きます。 このメソッドが正常に終了した場合、DataOffset はストリングの長さ分増分されます。

定義先:
MQMessage クラス

構文: Call *MQMessage*.**WriteUTF**(value$)

パラメーター:
value$ はストリング *(String)*。 書き込まれる値を表す。

MQPutMessageOptions クラス

このクラスは、 WebSphere MQ キューにメッセージを書き込むアクションを制御するさまざまなオプションをカプセル化します。

包含

MQPutMessageOptions クラスは MQSession クラスに含まれます。

作成

新規作成は、新規 MQPutMessageOptions オブジェクトを作成し、すべてのプロパティーを初期値に設定します。

あるいは、MQSession クラスの AccessPutMessageOptions メソッドを使用します。

構文

Dim *pmo* **As New MQPutMessageOptions** または

Set *pmo* =**New MQPutMessageOptions**

プロパティー

- 1097 ページの『CompletionCode プロパティー』.
- 1097 ページの『Options プロパティー』.
- 1097 ページの『ReasonCode プロパティー』.
- 1097 ページの『ReasonName プロパティー』.

- 1098 ページの『RecordFields プロパティー』.
- 1098 ページの『ResolvedQueueManagerName プロパティー』.
- 1098 ページの『ResolvedQueueName プロパティー』.

方法

- 1098 ページの『ClearErrorCodes メソッド』.

CompletionCode プロパティー

読み取り専用。 オブジェクトに対して発行された最新のメソッドまたはプロパティー・アクセスによって設定された完了コードを戻します。

定義先: MQPutMessageOptions クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = PutOpts*.**CompletionCode**

Options プロパティー

読み取り/書き込み可能。 MQPMO Options フィールド。 このフィールドの初期値は MQPMO_NONE です。 詳しくは、MQPMO オプションを参照してください。

定義先: MQPutMessageOptions クラス

データ型: 長整数 (Long)

構文: 取得する場合: *options & = PutOpts*.**オプション**

設定する場合: *PutOpts*。**オプション** = *options &*

MQPMO_PASS_IDENTITY_CONTEXT および MQPMO_PASS_ALL_CONTEXT オプションはサポートされていません。

ReasonCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コードを戻します。

定義先: MQPutMessageOptions クラス

データ型: 長整数 (Long)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = PutOpts*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

定義先: MQPutMessageOptions クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = PutOpts*.**ReasonName**

RecordFields プロパティー

読み取り/書き込み可能。 配布リストにメッセージを書き込む場合に、キューあたりベースでカスタマイズされるフィールドを示すフラグ。 初期値はゼロです。

このプロパティーは、MQI MQPMO 構造体の PutMsgRecFields フラグに対応します。 MQI では、これらのフラグが (MQPMR 構造体の) どのフィールドが存在するか、どのフィールドを MQPUT が使用するか制御します。 MQPutMessageOptions オブジェクトでは、これらのフィールドは常に存在するので、フラグは Put が使用するフィールドにしか影響を与えません。 詳細については、 *WebSphere MQ* アプリケーション・プログラミング・リファレンス を参照してください。

定義先:
MQPutMessageOptions クラス

データ型:
Long

構文: 取得する場合: *recordfields & = PutOpts*.**RecordFields**

設定する場合: *PutOpts*.**RecordFields** = *recordfields &*

ResolvedQueueManagerName プロパティー

読み取り専用。 MQPMO ResolvedQMgrName フィールド。 詳細については、ResolvedQMgrName (MQCHAR48) を参照してください。 初期値はすべてブランクです。

定義先: MQPutMessageOptions クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *qmgr$ = PutOpts*.**ResolvedQueueManagerName**

ResolvedQueueName プロパティー

読み取り専用。 MQPMO ResolvedQName フィールド。 詳細については、ResolvedQName (MQCHAR48) を参照してください。 初期値はすべてブランクです。

定義先: MQPutMessageOptions クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *qname$ = PutOpts*.**ResolvedQueueName**

ClearErrorCodes メソッド

MQPutMessageOptions クラスおよび MQSession クラスの両方について、 CompletionCode を MQCC_OK に、ReasonCode を MQRC_NONE にリセットします。

定義先: MQPutMessageOptions クラス

構文: Call *PutOpts*.**ClearErrorCodes()**

MQGetMessageOptions クラス

このクラスは、 WebSphere MQ キューからメッセージを読み取るアクションを制御するさまざまなオプションをカプセル化します。

包含

MQGetMessageOptions クラスは MQSession クラスに含まれます。

作成

新規作成は、新規 MQGetMessageOptions オブジェクトを作成し、すべてのプロパティーを初期値に設定します。

あるいは、MQSession クラスの AccessGetMessageOptions メソッドを使用します。

プロパティー

- 1099 ページの『CompletionCode プロパティー』
- 1099 ページの『MatchOptions プロパティー』
- 1100 ページの『Options プロパティー』
- 1100 ページの『ReasonCode プロパティー』
- 1100 ページの『ReasonName プロパティー』
- 1100 ページの『ResolvedQueueName プロパティー』
- 1100 ページの『WaitInterval プロパティー』

方法

- 1100 ページの『ClearErrorCodes メソッド』

構文

Dim *gmo* **As New MQGetMessageOptions** または

Set *gmo* = **New MQGetMessageOptions**

CompletionCode プロパティー

読み取り専用。 オブジェクトに対して発行された最新のメソッドまたはプロパティー・アクセスによって設定された完了コードを戻します。

定義先: MQGetMessageOptions クラス

データ型: 長整数 (Long)

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = GetOpts*.**CompletionCode**

MatchOptions プロパティー

読み取り/書き込み可能。 MQGET に使用される選択基準を制御するオプション。 初期値は、MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID です。

定義先:
MQGetMessageOptions クラス

データ型:
Long

値:
MatchOptions (MQLONG) を参照してください。

構文: 取得する場合: *matchoptions & = GetOpts*.**MatchOptions**

設定する場合: *GetOpts*.**MatchOptions** =*matchoptions &*

Options プロパティー

読み取り/書き込み可能。 MQGMO Options フィールド。 詳しくは、Options を参照してください。 初期値は MQGMO_NO_WAIT です。

定義先: MQGetMessageOptions クラス

データ型: 長整数 (Long)

構文: 取得する場合: *options & = GetOpts*. 設定するオプション: *GetOpts*。 **オプション** = *options &*

ReasonCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コードを戻します。

定義先: MQGetMessageOptions クラス

データ型: 長整数 (Long)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = GetOpts*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 最新の理由コードの記号名を戻します。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。
定義先: MQGetMessageOptions クラス

データ型: ストリング (String)

値:

- API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQGetMessageOptions*.**ReasonName**

ResolvedQueueName プロパティー

読み取り専用。 MQGMO ResolvedQName フィールド。 詳細については、ResolvedQName (MQCHAR48) を参照してください。 初期値はすべてブランクです。

定義先: MQGetMessageOptions クラス

データ型: 48 文字のストリング (String)

構文: 取得する場合: *qname$ = GetOpts*.**ResolvedQueueName**

WaitInterval プロパティー

読み取り/書き込み。 MQGMO WaitInterval フィールド。 待機アクションが Options プロパティーにより要求された場合に、Get が適切なメッセージの到着を待つミリ秒単位の最大時間。 このフィールドの初期値は 0 です。 MQGMO オプションの詳細については、 MQGMO を参照してください。

定義先: MQGetMessageOptions クラス

データ型: 長整数 (Long)

構文: 取得する場合: *wait & = GetOpts*.**WaitInterval**

設定する場合: *GetOpts*.**WaitInterval** = wait &

ClearErrorCodes メソッド

MQGetMessageOptions クラスおよび MQSession クラスの両方について、 CompletionCode を MQCC_OK に、ReasonCode を MQRC_NONE にリセットします。

定義先: MQGetMessageOptions クラス

構文: Call *GetOpts*.**ClearErrorCodes()**

MQDistributionList クラス

このクラスは、キューの集合 (出力のためのローカル、リモートまたは別名) をカプセル化します。

作成

新規作成は、新規の MQDistributionList オブジェクトを作成します。

あるいは、MQQueueManager クラスの AddDistributionList メソッドを使用します。

プロパティー

- 1101 ページの『AlternateUserId プロパティー』
- 1101 ページの『CloseOptions プロパティー』
- 1102 ページの『CompletionCode プロパティー』
- 1102 ページの『ConnectionReference プロパティー』
- 1102 ページの『FirstDistributionListItem プロパティー』
- 1102 ページの『IsOpen プロパティー』
- 1103 ページの『OpenOptions プロパティー』
- 1103 ページの『ReasonCode プロパティー』
- 1103 ページの『ReasonName プロパティー』

メソッド

- 1103 ページの『AddDistributionListItem メソッド』
- 1104 ページの『ClearErrorCodes メソッド』
- 1104 ページの『Close メソッド』
- 1104 ページの『Open メソッド』
- 1104 ページの『Put メソッド』

構文

Dim *distlist*.**As New MQDistributionList** または **Set** *distlist* = **New MQDistributionList**

AlternateUserId プロパティー

読み取り/書き込み可能。 このキューのオープン時にキューのリストへのアクセスを検査するために使われる代替ユーザー ID。

定義先:
MQDistributionList クラス

データ型:
12 文字のストリング (String)

構文: 取得する場合: *altuser$ = MQDistributionList*.**AlternateUserId**

設定する場合: *MQDistributionList*.**AlternateUserId** = *altuser$*

CloseOptions プロパティー

読み取り/書き込み可能。 配布リストをクローズするときに発生することを制御するために使用されるオプション。 初期値は MQCO_NONE です。

定義先:
MQDistributionList クラス

データ型:
Long

値:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

構文: 取得する場合: *closeopt & = MQDistributionList*.**CloseOptions**

設定する場合: *MQDistributionList*.**CloseOptions** = *closeopt &*

CompletionCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された完了コード。

定義先:
MQDistributionList クラス

データ型:
Long

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode & = MQDistributionList*.**CompletionCode**

ConnectionReference プロパティー

読み取り/書き込み可能。 配布リストが属するキュー・マネージャー。

定義先:
MQDistributionList クラス

データ型:
MQQueueManager

構文: 取得する場合: *set queuemanager = MQDistributionList*.**ConnectionReference**

設定する場合: *set MQDistributionList*. **ConnectionReference** = *queuemanager*

FirstDistributionListItem プロパティー

読み取り専用。 配布リストと関連付けられる最初の配布リストの項目オブジェクト。

定義先:
MQDistributionList クラス

データ型:
MQDistributionListItem

値:

構文: 取得する場合: *set distributionlistitem = MQDistributionList*.**FirstDistributionListItem**

IsOpen プロパティー

読み取り専用。

定義先:
MQDistributionList クラス

データ型:
ブール値

値:

- TRUE (-1)
- FALSE (0)

構文: 取得する場合: *IsOpen = MQDistributionList*.**IsOpen**

OpenOptions プロパティー

読み取り/書き込み可能。 配布リストがオープンされるときに使用されるオプション。

定義先:
MQDistributionList クラス

データ型:
Long

値:
MQPMO オプションを参照してください。

構文: 取得する場合: *openopt & = MQDistributionList*.**OpenOptions**

設定する場合: *MQDistributionList*.**OpenOptions** = openopt &

ReasonCode プロパティー

読み取り専用。 オブジェクトに対し発行された最新のメソッドまたはプロパティー・アクセスによって設定された理由コード。

定義先:
MQDistributionList クラス

データ型:
Long

値:
API 理由コードを参照してください。

構文: 取得する場合: *reasoncode & = MQDistributionList*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 理由コードの記号名。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

定義先:
MQDistributionList クラス

データ型:
ストリング

値:
API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQDistributionList*.**ReasonName**

AddDistributionListItem メソッド

新規 MQDistributionListItem オブジェクトを作成し、そのオブジェクトを配布リスト・オブジェクトと関連付けます。 キュー名はパラメーターは必須です。

配布リスト項目の DistributionList プロパティーは、所有している配布リストに設定され、配布リストの FirstDistributionListItem プロパティーは、新規の配布リスト項目を参照するよう設定されます。

新規の配布リスト項目の場合、PreviousDistributionListItem プロパティーには何も設定されず、NextDistributionListItem プロパティーは、 前に最初であった配布リスト項目を参照するよう設定されます。前に何もない (つまり 新規の項目はすでにある項目の前に挿入される) 場合は、NextDistributionListItem プロパティーには 何も設定されません。

配布リストがオープンしている場合、エラーが戻されます。

定義先:
MQDistributionList クラス

構文: set distributionlistitem = *MQDistributionList*.**AddDistributionListItem** (QName$, QMgrName$)

パラメーター:

QName$ はストリング (String)。 WebSphere MQ キューの名前を表す。

QMgrName$ はストリング (String)。 WebSphere MQ キュー・マネージャー名を表す。

ClearErrorCodes メソッド

MQDistributionList クラスおよび MQSession class クラスの両方について、完了コードを MQCC_OK にリセットし、理由コードを MQRC_NONE にリセットします。

定義先:
MQDistributionList クラス

構文: Call *MQDistributionList*.**ClearErrorCodes**()

Close メソッド

Close オプションの現在値を使用して、配布リストをクローズします。

定義先:
MQDistributionList クラス

構文: Call *MQDistributionList*.**Close**()

Open メソッド

AlternateUserId の現在値を使用して、現在のオブジェクトと関連付けられる配布リスト項目の QueueName および (適切な場合) QueueManagerName プロパティーが指定したそれぞれのキューをオープンします。

定義先:
MQDistributionList クラス

構文: Call *MQDistributionList*.**Open**()

Put メソッド

配布リストと関連づけられる配布リスト項目が識別したそれぞれのキューにメッセージを書き込みます。

定義先:

MQDistributionList クラス

構文

MQDistributionList.**Put**(メッセージ、 PutMsg オプション &)

Parameters

Message。書き込まれるメッセージを表す MQMessage オブジェクト。

PutMsgOptions。PUT 操作を制御するためのオプションが含まれる MQPutMessageOptions オブジェクト。 指定されていない場合、デフォルトの PutMessageOptions が使用されます。

このメソッドは、パラメーターとして MQMessage オブジェクトを使用します。 次の配布リスト項目プロパティーは、このメソッドの結果として更新されることがあります。

- CompletionCode
- ReasonCode
- ReasonName

- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Feedback
- AccountingToken
- AccountingTokenHex

MQDistributionListItem クラス

このクラスは、MQOR、MQRR、および MQPMR 構造体をカプセル化し、それを所有している配布リストと関連付けます。

作成

MQDistributionList クラスの AddDistributionListItem メソッドを使用します。

プロパティー

方法

- 1106 ページの『AccountingToken プロパティー』.
- 1106 ページの『AccountingTokenHex プロパティー』.
- 1106 ページの『CompletionCode プロパティー』.
- 1107 ページの『CorrelationId プロパティー』.
- 1107 ページの『CorrelationIdHex プロパティー』.
- 1107 ページの『DistributionList プロパティー』.
- 1107 ページの『Feedback プロパティー』.
- 1108 ページの『GroupId プロパティー』.
- 1108 ページの『GroupIdHex プロパティー』.
- 1108 ページの『MessageId プロパティー』.
- 1108 ページの『MessageIdHex プロパティー』.
- 1109 ページの『NextDistributionListItem プロパティー』.
- 1109 ページの『PreviousDistributionListItem プロパティー』.
- 1109 ページの『QueueManagerName プロパティー』.
- 1109 ページの『QueueName プロパティー』.
- 1109 ページの『ReasonCode プロパティー』.
- 1110 ページの『ReasonName プロパティー』.
- 1110 ページの『ClearErrorCodes メソッド』.

プロパティー:

- AccountingToken プロパティー
- AccountingTokenHex プロパティー
- CompletionCode プロパティー
- CorrelationId プロパティー

- CorrelationIdHex プロパティー
- DistributionList プロパティー
- Feedback プロパティー
- GroupId プロパティー
- GroupIdHex プロパティー
- MessageId プロパティー
- MessageIdHex プロパティー
- NextDistributionListItem プロパティー
- PreviousDistributionListItem プロパティー
- QueueManagerName プロパティー
- QueueName プロパティー
- ReasonCode プロパティー
- ReasonName プロパティー

メソッド*:*

- ClearErrorCodes メソッド

作成*:*

MQDistributionList クラスの AddDistributionListItem メソッドを使用します。

AccountingToken プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる AccountingToken。 初期値はすべてヌルです。

定義先:
MQDistributionListItem クラス

データ型:
32 文字のストリング (String)

構文: 取得する場合: *accountingtoken$ = MQDistributionListItem*.**AccountingToken**

設定する場合: *MQDistributionListItem*.**AccountingToken** = *accountingtoken$*

AccountingTokenHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる AccountingToken。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、 "6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 64 個の 16 進文字を指定する必要があります。

初期値は "0...0" です。

定義先:
MQDistributionListItem クラス

データ型:
32 個 ASCII 文字を表す 64 個の 16 進文字のストリング (String)

構文: 取得する場合: *accountingtokenh$ = MQDistributionListItem*.**AccountingTokenHex**

設定する場合: *MQDistributionListItem*.**AccountingTokenHex** = *accountingtokenh$*

CompletionCode プロパティー

読み取り専用。 所有している配布リスト・オブジェクトに対し発行された最新の OPEN または PUT 要求によって設定された完了コード。

定義先:
MQDistributionListItem クラス

データ型:
Long

値:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *completioncode$ = MQDistributionListItem*.**CompletionCode**

CorrelationId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる CorrelId。 初期値はすべてヌルです。

定義先:
MQDistributionListItem クラス

データ型:
24 文字のストリング (String)

構文: 取得する場合: *correlid$ = MQDistributionListItem*.**CorrelationId**

設定する場合: *MQDistributionListItem*.**CorrelationId** = *correlid$*

CorrelationIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる CorrelId。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、 "6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0..0" です。

定義先:
MQDistributionListItem クラス

データ型:
24 ASCII 文字を表す 48 の 16 進文字のストリング (String)

構文: 取得する場合: *correlidh$ = MQDistributionListItem*.**CorrelationIdHex**

設定する場合: *MQDistributionListItem*.**CorrelationIdHex** = *correlidh$*

DistributionList プロパティー

読み取り専用。 この配布リスト項目が関連付けられる配布リスト。

定義先:
MQDistributionListItem クラス

データ型:
MQDistributionList

構文: 取得する場合: *set distributionlist = MQDistributionListItem*.**DistributionList**

Feedback プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる Feedback 値。

定義先:
MQDistributionListItem クラス

データ型:
Long

値:
Feedback (MQLONG) を参照してください。

構文: 取得する場合: *feedback & = MQDistributionListItem*.**Feedback**

設定する場合: *MQDistributionListItem*.**Feedback** = *feedback &*

GroupId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる GroupId。 初期値はすべてヌルです。

定義先:
MQDistributionListItem クラス

データ型:
24 文字のストリング (String)

構文: 取得する場合: *groupid$ = MQDistributionListItem*.**GroupId**

設定する場合: *MQDistributionListItem*.**GroupId** = *groupid$*

GroupIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる GroupId。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、 "6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0..0" です。

定義先:
MQDistributionListItem クラス

データ型:
24 個 ASCII 文字を表す 48 個の 16 進文字のストリング (String)

構文: 取得する場合: *groupidh$ = MQDistributionListItem*.**GroupIdHex**

設定する場合: *MQDistributionListItem*.**GroupIdHex** = *groupidh$*

MessageId プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる MessageId。 初期値はすべてヌルです。

定義先:
MQDistributionListItem クラス

データ型:
24 文字のストリング (String)

構文: 取得する場合: *messageid$ = MQDistributionListItem*.**MessageId**

設定する場合: *MQDistributionListItem*.**MessageId** = *messageid$*

MessageIdHex プロパティー

読み取り/書き込み可能。 キューにメッセージを書き込む場合、 メッセージの MQPMR に組み込まれる MessageId。

ストリングのそれぞれの 2 文字は、単一の ASCII 文字に相応する 16 進数を表します。 例えば、"6" と "1" の対の文字は単一文字 "A" を表し、 "6" と "2" の対の文字は単一文字 "B" を表すというようになります。

有効な 48 個の 16 進文字を指定する必要があります。

初期値は "0..0" です。

定義先:
MQDistributionListItem クラス

データ型:
24 ASCII 文字を表す 48 の 16 進文字のストリング (String)

構文: 取得する場合: *messageidh$ = MQDistributionListItem*.**MessageIdHex**

設定する場合: *MQDistributionListItem*.**MessageIdHex** = *messageidh$*

NextDistributionListItem プロパティー

読み取り専用。 同じ配布リストと関連付けられる次の配布リストの項目オブジェクト。

定義先:
MQDistributionListItem クラス

データ型:
MQDistributionListItem

構文: 取得する場合: *set distributionlistitem = MQDistributionListItem*.**NextDistributionListItem**

PreviousDistributionListItem プロパティー

読み取り専用。 同じ配布リストと関連付けられる前の配布リストの項目オブジェクト。

定義先:
MQDistributionListItem クラス

データ型:
MQDistributionListItem

構文: 取得する場合: *set distributionlistitem = MQDistributionListItem*.**PreviousDistributionListItem**

QueueManagerName プロパティー

読み取り/書き込み可能。 WebSphere MQ キュー・マネージャー名。

定義先:
MQDistributionListItem クラス

データ型:
48 文字のストリング (String)

構文: 取得する場合: *qmname$ = MQDistributionListItem*.**QueueManagerName**

設定する場合: *MQDistributionListItem*.**QueueManagerName** = *qmname$*

QueueName プロパティー

読み取り/書き込み可能。 WebSphere MQ キュー名。

定義先:
MQDistributionListItem クラス

データ型:
48 文字のストリング (String)

構文: 取得する場合: *qname$ = MQDistributionListItem*.**QueueName**

設定する場合: *MQDistributionListItem*.**QueueName** = *qname$*

ReasonCode プロパティー

読み取り専用。 所有している配布リスト・オブジェクトに対し発行された最新の OPEN または PUT によって設定された完了コード。

定義先:
MQDistributionListItem クラス

データ型:
Long

値:
API 理由コードを参照してください。

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

構文: 取得する場合: *reasoncode & = MQDistributionListItem*.**ReasonCode**

ReasonName プロパティー

読み取り専用。 理由コードの記号名。 例えば、"MQRC_QMGR_NOT_AVAILABLE" です。

定義先:
MQDistributionListItem クラス

データ型:
ストリング

値:
API 理由コードを参照してください。

構文: 取得する場合: *reasonname$ = MQDistributionListItem*.**ReasonName**

ClearErrorCodes メソッド

MQDistributionListItem クラスおよび MQSession クラスの両方について、 完了コードを MQCC_OK にリセットし、理由コードを MQRC_NONE にリセットします。

定義先:
MQDistributionListItem クラス

構文: Call *MQDistributionListItem*.**ClearErrorCodes**

トラブルシューティング

提供されているトレース機能、思いがけない危険、およびそれらを回避する方法に関するヘルプについての情報。

このトラブルシューティングの以下のセクションでは、提供されているトレース機能と、思いがけない危険およびそれらを回避するための方法について説明します。

- 1110 ページの『トレースの使用法』
- 1112 ページの『WebSphere MQ Automation Classes for ActiveX スクリプトにエラーが発生した場合』
- 1112 ページの『理由コード』
- 1114 ページの『コード・レベル・ツール』

トレースの使用法

MQAX は、ユーザー・サイトで問題が発生したときに、保守部門が発生した 問題の状況を突き止めるためのトレース機能を備えています。 トレース機能は、MQAX スクリプトを実行 したときに使用したパスを示します。 問題があるとき以外は、システム・リソースの浪費を避けるために、トレースをオフに設定して実行してください。

次の 3 つの環境変数を設定してトレースを制御します。

- OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

OMQ_TRACE に どの値 を指定しても、トレース機能がオンになることに注意してください。 OMQ_TRACE を OFF に設定しても、トレースは引き続きアクティブです。

トレースをオフに切り替える場合は、OMQ_TRACE に値を指定しないでく ださい。

1.「**スタート**」をクリックします。
2.「**コントロール パネル**」をクリックします。
3.「**システム**」をダブルクリックします。
4.「**詳細設定**」をクリックします。
5.「**環境変数**」をクリックします。
6.「(ユーザー名) のユーザー環境変数」というタイトルのセクションで、「**新規**」をクリックします。
7. 該当するフィールドに変数名と変数値を入力して、「**OK**」をクリックします。
8.「**OK**」をクリックして、「環境変数」ウィンドウを閉じます。
9.「**OK**」をクリックして、「システムのプロパティ」ウィンドウを閉じます。
10.「コントロール パネル」ウィンドウをクローズします。

トレース・ファイルを書き込む場所を決定したら、ディスクからの読み取 り権限だけではなく、ディスクへの十分な書き込み権限があることを確認します。

トレース機能をオンに切り替えると、MQAX の稼働速度が低下しますが、 ActiveX 環境や WebSphere MQ 環境のパフォーマンスには影響はありません。 トレース・ファイルが必要なくなったら、削除してもかまいませ ん。

OMQ_TRACE 変数のステータスを変更する場合は、MQAX を停止す る必要があります。

トレース・ファイルのファイル名とディレクトリー

トレース・ファイルのファイル名は、OMQnnnnn.trc の形式になります。この 場合の nnnnn は、そのとき実行中の ActiveX プロセスの ID です。

表 *157.* コマンドとその効果

コマンド	機能
SET OMQ_TRACE_PATH = drive:¥directory	トレース・ファイルを書き込むトレース・ディレクトリーを設定 します。
SET OMQ_TRACE_PATH =	OMQ_PATH 環境変数を削除して、ActiveX の始動時の現行作業ディレク トリーが使用されます。
ECHO %OMQ_TRACE_PATH%	Windows のトレース・ディレクトリーの現行の設定を表示します。
SET OMQ_TRACE = xxxxxxxx	トレース機能をオンに設定します。 '=' 符号の後に 1 つ以上の文字を指定すると、トレース機能が オンに切り替わります。 例: SET OMQ_TRACE=yes SET OMQ_TRACE = no。 どちらの例でも、トレースは ON に設定されます。 この設定は、1 つのウィンドウまたはセッションでのみ有効です。
SET OMQ_TRACE=	トレース機能をオフに設定します。
ECHO %OMQ_TRACE%	Windows の環境変数の内容を表示します。
SET	Windows のすべての環境変数の内容を表示します。
SET OMQ_TRACE_LEVEL = 9	トレース・レベルを 9 に設定します。 9 より大きい値を指定しても、トレース・ファイルに追加情報は生成されません。

WebSphere MQ Automation Classes for ActiveX スクリプトにエラーが発生した場合

WebSphere MQ Automation Classes for ActiveX スクリプトにエラーが 発生した場合、いくつかの情報ソースがあります。

基本障害症状レポート

トレース機能とは別に、予期しない内部エラーが発生した場合、基本障害症状レポートが生成されることがあります。

このレポートは、OMQnnnnn.fdc という名前のファイルに生成されます。 この場合の nnnnn は、そのときに実行中の ActiveX プロセスの番号です。 このファイルは、ActiveX を始動した作業ディレクトリー、または OMQ_PATH 環境変数で指定したパスの中にあります。

その他情報ソース

WebSphere MQ は、使用されているプラットフォームに応じてさまざまなエラー・ログとトレース情報を提供します。 ご使用の Windows NT アプリケーションの事象ログを参照してくだ さい。

理由コード

WebSphere MQ MQI 用の資料で説明されている理由コードの他に、次の理由 コードが発生することがあります。 この他のコードについては、 WebSphere MQ アプリケーションの事象ログを参照してください。

表 *158.* 理由コードとその意味

理由コード	説明
MQRC_LIBRARY_LOAD_ERROR (6000)	1 つ以上の WebSphere MQ ライブラリーがロードできませんでした。 すべての WebSphere MQ ライブラリーが使用中のシステム上の正しい 検索パスの中にあることを確認してください。 例えば、WebSphere MQ ライブラリーが格納されているディレクトリーが PATH の中にあることを確認してください。
MQRC_CLASS_LIBRARY_ERROR (6001)	WebSphere MQ クラス・ライブラリー呼び出しのいずれかが、 予期しない ReasonCode または CompletionCode を戻しました。 基本障害症状レポートで詳細を確認してください。 最新のメソッドまたはプロパティーと、使用しているクラスを書き 留めて、IBM サポートにその問題を連絡してください。
MQRC_STRING_LENGTH_TOO_BIG (6002)	メッセージ・バッファーに対して 65,535 バイトを超える長さの UTF 書式制御ストリングを作成しようとしました。
MQRC_WRITE_VALUE_ERROR (6003)	msg.WriteByte (240) などの範囲外の値が使用されています。
MQRC_PACKED_DECIMAL_ERROR (6004)	メッセージ・バッファーからパック 10 進数を読み込もうとしましたが、データ・ポインターのデータが有効なパック・データ形式ではありません。
MQRC_FLOAT_CONVERSION_ERROR (6005)	メッセージ・バッファーから単精度または倍精度浮動小数点数を読み込もうとしましたが、データ・ポインターのデータが適切な浮動小数点形式ではありません。
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	オープンしているオブジェクトの **OpenOptions** 値 が正しくないか、オブジェクトに 1 つ以上の追加オプションが必要です。 暗黙的な再オープンが必要ですが、閉止が防止されています。 起こりうるすべての事象に対応するため、**OpenOptions** を明示的に設定してください。この設定によって、暗黙的に再度オープンする必要がなくなります。 排他入力の目的でキューがオープンしているため、閉止が防止されて います。閉止すると、他のユーザーにそのキューのアクセス権を与えるウィンドウが表示されます。

表 *158.* 理由コードとその意味 (続き)

理由コード	説明
MQRC_REOPEN_INQUIRE_ERROR (6101)	オープンしているオブジェクトの OpenOptions 値 が正しくないか、オブジェクトに 1 つ以上の追加オプションが 必要です。 暗黙的な再オープンが必要ですが、閉止が防止されています。 OpenOptions を明示的に設定して、MQOO_INQUIRE を組み込みます。 オブジェクトの 1 つまたは複数の特性を動的にチェックする必要がある ため、閉止が防止されています。また、OpenOptions が MQOO_INQUIRE をまだ組 み込んでいません。
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	オープンしているオブジェクトの OpenOptions 値 が正しくないか、オブジェクトに 1 つ以上の追加オプションが 必要です。 暗黙的な再オープンが必要ですが、閉止が防止されています。 起こりうるすべての事象に対応するため、OpenOptions を明 示的に設定してください。この設定によって、暗黙的に再度オープンする必 要がなくなります。 キューが MQOO_SAVE_ALL_CONTEXT によりオープンされているため、閉止 が防止されています。また、復元できない Get が前に実行されています。 このため、保存されている状態情報がオープン・キューに関連付けられ、この情報がクローズによって破壊される可能性が生じました。
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	オープンしているオブジェクトの **OpenOptions** 値 が正しくないか、オブジェクトに 1 つ以上の追加オプションが必要です。 暗黙的な再オープンが必要ですが、閉止が防止されています。 起こりうるすべての事象に対応するため、**OpenOptions** を明示的に設定してください。この設定によって、暗黙的に再度オープンする必要がなくなります。 キューが定義タイプ MQQDT_TEMPORARY_DYNAMIC (閉止によって破棄され る) を持つローカル・キューであるため、閉止が防止されています。
MQRC_ATTRIBUTE_LOCKED (6104)	オブジェクトがオープンしているときに、そのオブジェクトの値または属性 を変更しようとしました。 **AlternateUserId** などの一定の属性は、オブジェクトがオープン しているときは変更できません。
MQRC_CURSOR_NOT_VALID (6105)	暗黙の再オープンで最後に使用されたため、オープンされているキューのブ ラウズ・カーソルが無効にされました。 起こりうるすべての事象に対応するため、OpenOptions を明 示的に設定してください。この設定によって、暗黙的に再度オープンする必 要がなくなります。
MQRC_ENCODING_ERROR (6106)	読み込みのため、次のメッセージ項目の符号化は、MQENC_NATIVE である必要があります。
MQRC_STRUCID_ERROR (6107)	データ・ポインターで始まる 4 文字から派生した次のメッセージ項目の ID の構造が、欠落しているか、あるいはその項目が読み込まれる変数のタイプと一致していません。
MQRC_NULL_POINTER (6108)	非ヌル・ポインターの指定が必要であるか、暗黙指定さ れる場合に、ヌル・ポインターが指定されました。 このエラーは、VBA から呼び出しのパラメーターとして使用される WebSphere MQ オブジェクトの明示宣言を使用したことによって発生した可能性 があります (例えば、dim msg as Object が ok である場合、 dim msg as MqMessage を指定すると問題が発生する場合があります)。 例えば、Excel で q を定義し、 dim msg as MqMessageq を設定している場合、put msg を指定すると、reasonCode MQRC_NULL_POINTER が与えられます。これは、VisualBasic から正しく動作します。
MQRC_NO_CONNECTION_REFERENCE (6109)	**MQQueue** オブジェクトが **MQQueueManager** との接続を失っています。 このエラーは、 **MQQueueManager** が切断されている場合に発生します。 **MQQueue** オブジェクトを削除してください。
MQRC_NO_BUFFER (6110)	使用可能なバッファーがありません。 **MQMessage** オブジェクトの場合、発生するはずのない オブジェクトの状態の内部矛盾を示して、バッファーの割り振りができません。
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	2 進データの長さが宛先の属性の長さと矛盾しています。 ゼロは、すべての属性の長さとして正しい値です。 24 は、**CorrelationId** の正しい長さであり、 **MessageId** 32 は **AccountingToken** の正しい長さです。
MQRC_BUFFER_NOT_AUTOMATIC (6112)	ユーザー定義および管理対象バッファーのサイズを変更することはできません。 メッセージ・バッファーはシステムで管理されているため、これは内部的な不整合を示します。

表 *158.* 理由コードとその意味 (続き)

理由コード	説明
MQRC_INSUFFICIENT_BUFFER (6113)	要求を収容するデータ・ポインターより後は、使用可能なバッファー・スペースが不足しています。 このエラーは、バッファーのサイズを変更できないために起こった可能 性があります。
MQRC_INSUFFICIENT_DATA (6114)	読み取り要求を収容するデータ・ポインターの後に、十分なデータ がありません。 バッファーのサイズを縮小して、適切なサ イズに変更してから、もう一度データを読み取ってください。
MQRC_DATA_TRUNCATED (6115)	バッファーから別のバッファーにコピーされたとき、データが切り捨てられました。 このエラーは、宛先バッファーのサイズを変更できないため、一 方または他方のバッファーのアドレス指定に問題があるため、あるいは、バ ッファーのサイズが縮小され、サイズの小さいバッファーに置き換えられ ているために起こった可能性があります。
MQRC_ZERO_LENGTH (6116)	正の値の長さの指定が必要であるか、暗黙指定される場合に、ゼロ長が指定されています。
MQRC_NEGATIVE_LENGTH (6117)	ゼロまたは正の値の長さを指定する必要があるところに、負の値の長さが指定されています。
MQRC_NEGATIVE_OFFSET (6118)	ゼロまたは正の値のオフセットを指定する必要があるところに、負の値のオフセットが指定されています。
MQRC_INCONSISTENT_FORMAT (6119)	次のメッセージ項目の形式と、項目が読み込まれる変数のタイプとが一致していません。
MQRC_INCONSISTENT_OBJECT_STATE (6120)	オープンしている当該オブジェクトと、接続されていない 参照先 MQQueueManager オブジェクトの間に矛盾があります。
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	MQPutMessageOptions のコンテキスト参照で、有効な MQQueue オブジェクトが参照されません。 オブジェクトはすでに破 棄されています。
MQRC_CONTEXT_OPEN_ERROR (6122)	MQPutMessageOptions のコンテキスト参照で、コンテキストを設定するためにオープンできなかった MQQueue オブジェクトが参照されます。 このエラーは、MQQueue オブジェクトのオープン・オプションが不適切で あるため 、発生した可能性があります。 参照先オブジェクトの理由コードを検査して、原因を明らかに してください。
MQRC_STRUC_LENGTH_ERROR (6123)	内部データ構造の長さとデータの内容が矛盾しています。 MQRMH では、固定フィールドとすべてのオフセット・データを 格納するには長さが不十分です。
MQRC_NOT_CONNECTED (6124)	キュー管理プログラムとの必要な接続が使用できず、接続が暗黙 的に確立できないため、メソッドが失敗しました。
MQRC_NOT_OPEN (6125)	WebSphere MQ オブジェクトがオープンしておらず、オープンを暗黙的に実行できないため、 メソッドが失敗しました。
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	配布先リストに MQDistributionList Item オブジェクトが ないため、MQDistributionList がオープンに失敗しました。 修正処置: 配布先リストに 1 つ以上の MQDistributionListItem オブジェクトを追加してください。
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	オブジェクトはオープンしているが、オープン・オプションが必 要な操作と矛盾しているため、メソッドが失敗しました。 修正処置: 適切なオープン・オプションを指定してオブジェクトを オープンしてから再試行してください。
MQRC_WRONG_VERSION (6128)	指定されたバージョン番号か検出されたバージョン番号が間違っているかまたはサポートされていないため、メソッドが失敗しました。

コード・レベル・ツール

ユーザーは、IBM の保守サービス・チームから インストールしたコードのレベルを尋ねられることがあります。

コードのレベルを見つけるには、'MQAXLEV' ユーティリティー・プ ログラムを実行します。

コマンド・プロンプトで、ディレクトリーを MQAX200.dll が格納されているディレクトリーに変更するか、全パス 長さを追加して次のように入力します。

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

この場合の MQAXLEV.OUT は、出力ファイルの名前です。

出力ファイルを指定しないと、詳細は画面に表示されます。

以下の例では、コード・レベル・ツールから出力されるファイルの例について詳述します。

例、コード・レベル・ツールからの出力ファイル

```
5639-B43 (C) Copyright IBM Corp. 1996, 2024. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 ***** lib/mqole/mqole.cpp, mqole, p000, p000 L981119 1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216             1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212               1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212             1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212           1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212           1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212           1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219             1.11 99/02/18 12:12:59
```

MQAI との間の ActiveX インターフェース

各 COM インターフェースの簡単な概説と、MQAI での使用法については、1038 ページの『Component Object Model インターフェース (WebSphere MQ Automation Classes for ActiveX) の使用』を参照してください。

MQAI を使用すると、アプリケーションでプログラム式コマンド形式 (PCF) のコマンドを作成および送信することができ、その際に PCF に必要 な可変長バッファーを直接入手したり、フォーマットしたりする必要がありません。 MQAI の詳細については、WebSphere MQ 管理インターフェース (MQAI) の紹介を参照してください。 MQAI ActiveX MQBag クラスは、MQAI によってサポートされるデータ・バッグを、COM オブジェクトの作成をサポートする任意の言語 (Visual Basic、C++、Java、およびその他の ActiveX スクリプト・クライアントなど) で使用できる方法でカプセル化します。

MQAI の ActiveX インターフェースは、MQI に対する COM インターフェースを 提供する MQAX クラスと共に使用するように設計されています。 MQAX クラスの詳細については、1039 ページの『ActiveX 以外のアプリケーションにアクセスする MQAX アプリケーションの設計』を参照してください。

ActiveX インターフェースは、MQBag と呼ばれる 1 つのクラスを提供し ます。 このクラスを使用して、MQAI データ・バッグとそのプロパティーが作成 されます。各バッグ内でデータ項目を作成するとき、およびデータ 項目を使った作業を行うときにメソッドが使用されます。 MQBag の Execute メソッドは、バッグのデータを PCF メッセー ジとして WebSphere MQ キュー・マネージャーに送信し、その応答を収集します。

MQBag クラスおよびそのプロパティーとメソッドについて詳しくは、1115 ページの『MQBag クラス』を参照してください。

PCF メッセージは、指定したキュー・マネージャー・オブジェクトに、 オプションで要求キューと応答キューを使用して送信されます。 応答は、新しい MQBag オブジェクトの中に戻されます。 すべてのコマンドと応答のセットについては、プログラマブル・コマンド・フォーマットの定義で説明しています。 適切な要求キューと応答キューを選択すると、 WebSphere MQ ネットワーク内のいずれのキュー・マネージャーにもコマンドを送信できます。

MQBag クラス

MQBag クラスは、MQBag オブジェクトを必要に応じて作成するときに使用されます。 MQBag クラスは、インスタンス化されると、新しい MQBag オブジェクト参照を戻します。

Visual Basic で MQBag オブジェクトを作成するには、次のように指定します。

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

MQBag プロパティー

MQBag オブジェクトのプロパティーを以下のリストで説明します。

- 1116 ページの『Item プロパティー』.
- 1118 ページの『Count プロパティー』.
- 1118 ページの『Options プロパティー』.

MQBag メソッド

MQBag オブジェクトのメソッドを以下のリストで説明します。

- 1119 ページの『Add メソッド』.
- 1119 ページの『AddInquiry メソッド』.
- 1120 ページの『Clear メソッド』.
- 1120 ページの『Execute メソッド』.
- 1121 ページの『FromMessage メソッド』.
- 1121 ページの『ItemType メソッド』.
- 1122 ページの『Remove メソッド』.
- 1122 ページの『Selector メソッド』.
- 1123 ページの『ToMessage メソッド』.
- 1123 ページの『Truncate メソッド』.

エラーの処理

MQBag オブジェクトの処理中にエラーが検出された場合 (基礎となる MQAX オブジェクトまたは MQAI オブジェクトによってバッグにエラーが 戻されるものを含む)、エラー例外が発生します。 MQBag クラスは COM ISupportErrorInfo インターフェースをサポ ートしているため、エラー処理ルーチンで次の情報が使用できます。

- エラー番号: 検出されたエラーの WebSphere MQ の理由コードと COM の機能コードからなる番号です。 COM の標準仕様である機能フィールドは、エラーの 責任がある領域を示します。 WebSphere MQ によって検出されたエラーのエラー番号は、必ず FACILITY_ITF になります。
- エラーの発生元: エラーが検出されたオブジェクトのタイプとバージョンを識別します。 MQBag の操作中に検出されるエラーの場合、エラーの発生元は常に MQBag.MQBag1 です。
- エラー記述: WebSphere MQ の理由コードの記号名を示す ストリングです。

エラー情報へのアクセス方法は、スクリプト記述言語によって異なります。 例えば、Visual Basic では、情報は Err オブジェクトに戻され、エラー番号から定数 vbObjectError を減算することによって、WebSphere MQ 理由コードが得られます。

ReasonCode = Err.Number - vbObjectError

MQBag の Execute メッセージが PCF メッセージを送信し、応答を 受け取った場合、送信したコマンドが正常に実行されなかった可能性があっても 、処理は正常終了したものと見なされます。 この場合、応答バッグ自体には、「プログラマブル・コマンド・フォーマットの定義」で説明されている完了コードとエラー理由コードが含まれています。

Item プロパティー

目的

Item プロパティーは、バッグの中の項目を表します。 Item プロパティーは、1 つの項目の値を設定したり、値について問い合 わせるときに使用されます。 このプロパティーは、以下の MQAI 呼び出しに対応して使用されます。

- "mqSetString"
- "mqSetInteger"
- "mqInquireInteger"
- "mqInquireString"
- "mqInquireBag"

これらの MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Item (Selector, ItemIndex, Value)

Parameters

Selector **(VARIANT) - 入力**
設定または問い合わせ対象の項目のセレクター。

項目について問い合わせる場合は、 MQSEL_ANY_USER_SELECTOR がデフォルトです。 項目を設定する場合のデフォルトは、MQIA_LIST または MQCA_LIST です。

Selector が long 型でない場合、 結果として MQRC_SELECTOR_TYPE_ERROR が生成されます。

このパラメーターはオプションです。

ItemIndex **(LONG) - 入力**
この値は、設定または問い合わせの対象として指定したセレクター を持つ項目のオカレンスを識別します。 デフォルトは、MQIND_NONE です。

このパラメーターはオプションです。

Value **(VARIANT) - 入出力**
戻り値または設定する値。 項目を問い合わせる場合、戻り値の型は long 型、ストリング型、 MQBag 型のいずれかです。 ただし、項目を設定する場合の値は、 long 型かストリング型のいずれかでなければなりません。これ以外の値を指定すると、結果として MQRC_ITEM_VALUE_ERROR が生成されます。

Visual Basic 言語の呼び出し

バッグ内の項目の値について問い合わせる場合は、次のように指定しま す。

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

MQBag の参照の場合は、次のように指定します。

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

バッグ内の項目の値を設定する場合は、次のように指定します。

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Count プロパティー

目的

Count プロパティーは、バッグ内のデータ項目の数を表します。 このプロパティーは、MQAI 呼び出し "mqCountItems" と対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Count (*Selector, Value*)

Parameters

***Selector* (VARIANT) - 入力**
カウントに入れるデータ項目のセレクター。

デフォルトは MQSEL_ALL_USER_SELECTORS です。

Selector が long 型でない場合、MQRC_SELECTOR_TYPE_ERROR が戻されます。

***Value* (LONG) - 出力**
Selector によってカウントに入れられたバッグ内の項目の 数。

Visual Basic 言語の呼び出し

バッグ内の項目の数を戻す場合は、次のように指定します。

```
ItemCount = mqbag.Count([Selector])
```

Options プロパティー

目的

Options プロパティーは、バッグを使用するためのオプションを設定し ます。 このプロパティーは、MQAI 呼び出し "mqCreateBag" の Options パラメーターと対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Options (*Options*)

Parameters

***Options* (LONG) - 入出力**
バッグのオプション。

注 : バッグのオプションは、各データ項目をバッグに追加したり、バッグ内 で設定する**前** に設定する必要があります。 バッグが空でないときにオプションを変更すると、 結果として MQRC_OPTIONS_ERROR が生成されます。 これは、バッグが後で消去される場合でも適用されます。

Visual Basic 言語の呼び出し

バッグ内の項目のオプションについて問い合わせる場合は、次のように 指定します。

```
Options = mqbag.Options
```

バッグ内の項目のオプションを設定する場合は、次のように指定しま す。

```
mqbag.Options = Options
```

MQBag メソッド

以降の節では、MQBag オブジェクトのメソッドについて説明します。

Add メソッド

目的

Add メソッドは、バッグにデータ項目を追加します。 このメソッドは、MQAI 呼び出し "mqAddInteger" と "mqAddString" に対応します。 これらの MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Add (*Value, Selector*)

Parameters

Value **(VARIANT) -** 入力
データ項目の整数値またはストリング値。

Selector **(VARIANT) -** 入力
追加する項目を識別するセレクター。

デフォルトは、Value の型によって MQIA_LIST または MQCA_LIST となります。 Selector パラメーターが long 型でない場合、 結果として MQRC_SELECTOR_TYPE_ERROR が生成されます。

Visual Basic 言語の呼び出し

バッグに項目を追加する場合は、次のように指定します。

```
mqbag.Add(Value,[Selector])
```

AddInquiry メソッド

目的

AddInquiry メソッドは、INQUIRE コマンドを実行する目的で管理バッグ が送信されたときに戻される属性を指定するセレクターを追加します。 このメソッドは、MQAI 呼び出し "mqAddInquiry" と対応します。この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

AddInquiry *(Inquiry)*

Parameters

Inquiry **(LONG) -** 入力
INQUIRE 管理コマンドによって戻される WebSphere MQ 属性の選択子。

Visual Basic 言語の呼び出し

AddInquiry メソッドを使用する場合は、次のように指定します。

```
mqbag.AddInquiry(Inquiry)
```

Clear メソッド

目的

Clear メソッドは、バッグのすべてのデータ項目を削除します。 このメソッドは、MQAI 呼び出し "mqClearBag" と対応します。この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

消去

Visual Basic 言語の呼び出し

バッグのすべてのデータ項目を削除する場合は、次のように指定します。

```
mqbag.Clear
```

Execute メソッド

目的

Execute メソッドは、管理コマンド・メッセージをコマンド・サーバーに送信し、何らかの応答メッセージを待ちます。 このメソッドは、MQAI 呼び出し "mqExecute" と対応します。この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

***(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag* を実行します。)**

Parameters

***QueueManager* (MQQueueManager) - 入力**
アプリケーションが接続されているキュー・マネージャー。

***Command* (LONG) - 入力**
実行されるコマンド。

***OptionsBag* (MQBag) - 入力**
呼び出しの処理に影響を与えるオプションが入っているバッグ。

***RequestQ* (MQQueue) - 入力**
管理コマンド・メッセージを入れるキュー。

***ReplyQ* (MQQueue) - 入力**
応答メッセージを受信するキュー。

***ReplyBag* (MQBag) - 出力**
応答メッセージのデータが入っているバッグの参照。

Visual Basic 言語の呼び出し

管理コマンド・メッセージを送信して応答メッセージを待つ場合は、次の ように指定します。

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

FromMessage メソッド

目的

FromMessage メソッドは、メッセージのデータをバッグにロードします。 このメソッドは、MQAI 呼び出し "mqBufferToBag" と対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

FromMessage (*Message*, *OptionsBag*)

Parameters

Message **(MQMessage) - 入力**
変換するデータが入っているメッセージ。

OptionsBag **(MQBag) - 入力**
呼び出しの処理を制御するオプション。

Visual Basic 言語の呼び出し

メッセージのデータをバッグにロードする場合は、次のように指定しま す。

```
mqbag.FromMessage(Message,[OptionsBag])
```

ItemType メソッド

目的

ItemType メソッドは、バッグ内の指定した項目に値の型を戻します。 このメソッドは、MQAI 呼び出し "mqInquireItemInfo" と対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

ItemType (*Selector*, *ItemIndex*, *ItemType*)

Parameters

Selector **(VARIANT) - 入力**
問い合わせする項目を識別する選択子。

デフォルトは MQSEL_ANY_USER_SELECTOR です。 `Selector` パラメーターが long 型でない場合、結果として MQRC_SELECTOR_TYPE_ERROR が生成されます。

ItemIndex **(LONG) - 入力**
問い合わせの対象の項目のインデックス。

デフォルトは、MQIND_NONE です。

ItemType **(LONG) - 出力**
指定した項目のデータ型。

注: Selector パラメーターと ItemIndex パラメー ターのどちらか一方、または両方を指定する必要があります。 どちらのパラメーターも指定しないと、結果として MQRC_PARAMETER_MISSING が生成されます。

Visual Basic 言語の呼び出し

値の型を戻す場合は、次のように指定します。

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

Remove メソッド

目的

Remove メソッドはバッグから項目を削除します。 このメソッドは、MQAI 呼び出し "mqDeleteItem" と対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Remove (*Selector*, *ItemIndex*)

Parameters

***Selector* (VARIANT) - 入力**
削除する項目を識別するセレクター。

デフォルトは MQSEL_ANY_USER_SELECTOR です。 Selector パラメーターが long 型でない場合、結果として MQRC_SELECTOR_TYPE_ERROR が生成されます。

***ItemIndex* (LONG) - 入力**
削除する項目のインデックス。

デフォルトは、MQIND_NONE です。

注: Selector パラメーターと ItemIndex パラメー ターのどちらか一方、または両方を指定する必要があります。 どちらのパラメーターも指定しないと、結果として MQRC_PARAMETER_MISSING が生成されます。

Visual Basic 言語の呼び出し

バッグから項目を削除する場合は、次のように指定します。

```
mqbag.Remove([Selector],[ItemIndex])
```

Selector メソッド

目的

Selector メソッドは、バッグの中の指定した項目のセレクターを戻します。 このメソッドは、MQAI 呼び出し "mqInquireItemInfo" と対応します。 この MQAI 呼び出しについては、プログラマブル・コマンド・フォーマット・リファレンスに説明があります。

Format

Selector (*Selector*, *ItemIndex*, *OutSelector*)

Parameters

***Selector* (VARIANT) - 入力**
問い合わせする項目を識別する選択子。

デフォルトは MQSEL_ANY_USER_SELECTOR です。 Selector パラメーターが long 型でない場合、結果として MQRC_SELECTOR_TYPE_ERROR が生成されます。

***ItemIndex* (LONG) - 入力**
問い合わせする項目のインデックス。

デフォルトは、MQIND_NONE です。

***OutSelector* (VARIANT) - 出力**
指定した項目のセレクター。

注 : Selector パラメーターと ItemIndex パラメー ターのどちらか一方、または両方を指定する必要があります。 どちらのパラメーターも指定しないと、結果として MQRC_PARAMETER_MISSING が生成されます。

Visual Basic 言語の呼び出し

項目のセレクターを戻す場合は、次のように指定します。

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

ToMessage メソッド

目的

ToMessage メソッドは、MQMessage オブジェクトに対する参照を戻します。 参照には、バッグ内のデータが入っています。 このメソッドは、プログラマブル・コマンド・フォーマット・リファレンスの MQAI 呼び出し "mqBagToBuffer" に対応します。

Format

ToMessage (*OptionsBag, Message*)

Parameters

***OptionsBag* (MQBag) - 入力**
メソッドの処理を制御するオプションが入っているバッグ。

***Message* (MQMessage)-出力**
バッグ内のデータが入っている MQMessage オブジェクト参照子。

Visual Basic 言語の呼び出し

ToMessage メソッドを使用する場合は、次のように指定します。

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Truncate メソッド

目的

Truncate メソッドは、バッグ内のユーザー項目の数を減らします。 このメソッドは、プログラマブル・コマンド・フォーマット・リファレンスの MQAI 呼び出し "mqTruncateBag" に対応します。

Format

Truncate (*ItemCount*)

Parameters

***ItemCount* (LONG) - 入力**
切り捨ての後、バッグに残すユーザー項目の数。

Visual Basic 言語の呼び出し

バッグ内のユーザー項目の数を減らす場合は、次のように指定します。

```
mqbag.Truncate(ItemCount)
```

WebSphere MQ Automation Classes for ActiveX スターター・サンプルについて

この付録では、WebSphere MQ Automation Classes for ActiveX スターター・サンプルと、 その使用法について説明します。

WebSphere MQ (Windows 版) には、以下の Visual Basic サンプル・ プログラムが付属しています。

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

これらのサンプルは、Visual Basic 4 または Visual Basic 5 で実行されます。 これらは、ディレクトリー ... ¥ tools¥ mqax¥ samples¥ vb にあります。

また、このディレクトリーの中には、Microsoft Excel および html のサンプルもあります。 次のとおりです。

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

注 : Visual Basic 5 をお使いの場合は、**必ず** Visual Basic コンポーネ ント grid32.ocx を選択してインストールしてください。

サンプルのデモ

各サンプルは、WebSphere MQ Automation Classes for ActiveX を使用して、 次の操作を行う方法を示します。

- キュー・マネージャーに接続します。
- キューへのアクセス
- キューへのメッセージの書き込み
- キューからのメッセージの読み取り

Visual Basic のサンプルの中心部を以降の節に記載します。

1125 ページの『サンプルを実行する準備』および

1125 ページの『サンプルでのエラー処理』

実行、ActiveX スターター・サンプル

WebSphere MQ Automation Classes for ActiveX スターター・サンプルを実行する前に、デフォルトのキュー・マネージャーが稼働しており、 必要なキュー定義が作成されていることを確認してください。 キュー・マネージャーの作成と稼働の方法、およびキューの作成方法の詳細については、管理を参照してください。 このサンプルでは、キュー SYSTEM.DEFAULT.LOCAL.QUEUE これは、通常はセットアップされる WebSphere MQ サーバーで定義する必要があります。

以下のリストに、データ・バッグのさまざまな使用方法を示します。

- キュー・マネージャーに接続します。
- キューへのアクセス
- キューへのメッセージの書き込み
- キューからのメッセージの読み取り

Microsoft Basic Version 4 以降用の MQAX スターター・サンプルについては、1125 ページの『MQAXTRIV サンプルの実行』 を参照してください。

キュー・マネージャーおよびキュー・オブジェクトのプロパティーとメソッドをブラウズできるサンプルについては、1127 ページの『MQAXCLSS サンプルの始動』を参照してください。

MQAXDLST サンプルについては、1127 ページの『MQAXDLST サンプル』を参照してください

Microsoft Excel 95 以降用の MQAX スターター・サンプルの実行方法については、1127 ページの『MQAXTRIV.XLS サンプルの実行』を参照してください。

MQAX.XLS を用いた Bank のデモを実行する方法については、1128 ページの『MQAX.XLS による Bank デモの実行』を参照してください。

ActiveX 互換 WWW ブラウザーを使用したスターター・サンプルについては、1128 ページの『ActiveX 互換 WWW ブラウザーを使ったスターター・サンプル』を参照してください。

サンプルを実行する準備

サンプルを実行するには、どのサンプルを実行するかによって、次のいずれかが必要となります。

- Microsoft Visual Basic Version 4 以降
- Microsoft Excel 95 以降
- Web ブラウザー

また、以下のことも必要です。

- 稼働している WebSphere MQ キュー・マネージャー
- 定義済みの WebSphere MQ キュー

サンプルでのエラー処理

WebSphere MQ Automation Classes for ActiveX パッケージに付属しているほとんどのサンプルでは、エラー処理についてはほとんど、あるいはまったく示されていません。 エラー処理の詳細については、1043 ページの『エラーの処理』を参照してください。

MQAXTRIV サンプルの実行

1. キュー・マネージャーを始動します。
2. Windows のエクスプローラまたはファイル・マネージャで、 サンプル MQAXTRIV.VBP (Visual Basic プロジェクト・ファイル) のアイコンを選 択して、ファイルをオープンします。

   Visual Basic プログラムが始動し、フ ァイル MQAXTRIV.VBP がオープンされます。
3. Visual Basic で、ファンクション・キー 5 (F5) を押してサンプル を実行します。
4. ウィンドウ・フォーム「MQAX trivial tester」の中の任意の場所をクリックします。

正しく動作したら、ウィンドウのバックグラウンドが緑に変わります。 セットアップに問題があった場合、ウィンドウのバックグラウンドは赤になり、エラー情報が表示されます。

以下の図は、Visual Basic のサンプルの中核部分を示しています。

```
Option Explicit

Private Sub Form_Click()

'*************************************************************************
'* This simple example illustrates how to put and get a WebSphere MQ message to
'* and from a WebSphere MQ message queue. The data from the message returned by the
'*get is read and compared with that from the original message.
'*************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* get message option

Dim GetOptions As MQGetMessageOptions   '* put message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*************************************************************************
'* Handle errors
'*************************************************************************
On Error GoTo HandleError

'*************************************************************************
'* Initialize the current position for the form
'*************************************************************************
CurrentX = 0
CurrentY = 0

'*************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                      MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127)  '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
```

```
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*****************************************************************************
'* Handle errors
'*****************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
    Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error:  " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

MQAXCLSS サンプルの始動

このサンプルでは、キュー・マネージャーおよび キュー・オブジェクトのプロパティーおよびメソッドをブラウズすることができます。

1. キュー・マネージャーを始動します。
2. Windows エクスプローラーでドキュメント・アイコンをダブルクリックするか、Visual Basic のファイル・メニューから「ファイル」>「開く」をクリックして、ファイル MQAXCLSS.VBP を開きます。
3. サンプルを始動します。
4. 適切なキュー・マネージャー名およびキュー名を入力し、対応するボタンをクリックします。

MQAXDLST サンプル

Visual Basic MQAXDLST サンプルでは、1 回の書き込みで同じメッセージを 2 つのキューに送る配布リストの使用法を示しています。 このサンプルを実行するには、MQAXCLSS サンプルの場合と同様の操作を行います。

Microsoft Excel 95 以降用の MQAX スターター・サンプル

この節では、Microsoft Excel 95 以降用の MQAX スターター・サンプル (MQAXTRIV.XLS) を実行する方法について説明します。

MQAXTRIV.XLS サンプルの実行

1. キュー・マネージャーを始動します。

2. エクスプローラまたはファイル・マネージャで、MQAX サンプル MQAXTRIV.XLS のアイコンを選択します。
3. スプレッドシート内のボタンをクリックします。
4. 成功または失敗のメッセージが表示され、画面が更新されます。

MQAX.XLS による *Bank* デモの実行

Bank デモを実行するには、以下の手順に従います。

1. キュー・マネージャーを始動します。
2. IBM WebSphere MQ の MQSC コマンド・ファイル BANK.TST を実行します。 これにより、必要な IBM WebSphere MQ キュー定義が設定されます。

   MQSC コマンド・ファイルの使用方法については、スクリプト (MQSC) コマンドを参照してください。
3. MQAXBSRV.VBP を実行します。 このサンプル・プログラムは、バックエンド・アプリケーションをシミュレートするサーバーですが、このプログラムは Microsoft Excel を用いて実行する必要があります。
4. MQAX.XLS を実行します。 このサンプルは、クライアント IBM WebSphere MQ のデモです。
5. リストからお客様を選択します。
6. 「**送信**」をクリックします。

短い休止 (約 3 秒) の後、フィールドに値が取り込まれ、棒グラフが表示されます。

ActiveX 互換 WWW ブラウザーを使ったスターター・サンプル

注 : このサンプルを実行するには、ActiveX と互換性のある Web ブラウザ ーを稼働していなければなりません。 Microsoft Internet Explorer は、互換性のある Web ブラウザーで すが、Netscape Navigator は互換性がありません。

HTML サンプルの実行

このサンプルでは、VBScript および JavaScript から MQAX を呼び出す方法を示しています。

1. キュー・マネージャーを始動します。
2. ActiveX と互換性のある Web ブラウザーで、ファイル "MQAXTRIV.HTM" をオープンします。

   Windows のエクスプローラでファイル・アイコンをダブルクリッ クするか、ActiveX と互換のある Web ブラウザーの「ファイ ル」メニューから「ファイル」 -「開く」を選択します。これにより、 ファイルがオープンします。
3. 画面に表示される指示に従って操作します。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
日本アイ・ビー・エム株式会社
法務・知的財産
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
〒 103-8510
103-8510
東京 103-8510、日本

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 INTERNATIONAL BUSINESS MACHINES CORPORATION は、法律上の瑕疵担保責任、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 "" 国または地域によっては、法律の強行規定により、保証責任の制限が 禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものでは ありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部では ありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用すること ができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他の ライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で 決定されたものです。 そのため、他の操作環境で得られた結果は、異なる可能性があります。 一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。 実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、 もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。 したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、 予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、 あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、名前や住所が類似する個人や企業が実在して いるとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

プログラミング・インターフェース情報 (提供されている場合) は、このプログラムで使用するアプリケーション・ソフトウェアの作成を支援することを目的としています。

本書には、プログラムを作成するユーザーが IBM WebSphere MQ のサービスを使用するためのプログラミング・インターフェースに関する情報が記載されています。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。 診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

重要 : この診断、修正、およびチューニング情報は、変更される可能性があるため、プログラミング・インターフェースとして使用しないでください。

商標

IBM、 IBM ロゴ、 ibm.com®は、世界の多くの国で登録された IBM Corporation の商標です。 現時点での IBM の商標リストについては、 "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml をご覧ください。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

この製品には、 Eclipse Project (http://www.eclipse.org/) により開発されたソフトウェアが含まれています。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

部品番号:

(1P) P/N: