MQSeries®

# Java™ の使用

SD88-7163-01
(英文原典：SC34-5456-06)

MQSeries®

# Java™ の使用

SD88-7163-01
(英文原典：SC34-5456-06)

**ご注意!**

本書、および本書がサポートする製品をご使用になる前に、 393ページの『付録F. 特記事項』にある一般的な情報を必ずお読みください。

本書は、IBM® MQSeries classes for Java バージョン 5.2.0 および MQSeries classes for Java Message Service バージョン 5.2 に適用されます。また、特に断りがない限り、それ以降のすべてのリリースおよび改訂レベルにも適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)


| | |
|---|---|
| **原 典:** | SC34-5456-06<br>MQSeries®<br>Using Java™ |
| **発 行:** | 日本アイ・ビー・エム株式会社 |
| **担 当:** | ナショナル・ランゲージ・サポート |

第1刷 2001.3


この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

# 目次

# 図

# 表

# 本書について

本書は、以下の内容を扱っています。

- MQSeries classes for Java (MQSeries システムへのアクセスに使用できる)
- MQSeries classes for Java Message Service (Java Message Service (JMS) へのアクセスと MQSeries アプリケーションへのアクセスの両方に使用できる)

**注:** 本書は、ソフトコピー (PDF および HTML) でのみ提供されており、製品の一部として、あるいは下記のアドレスにある MQSeries ファミリーの Web サイトから、入手できます。

`http://www.ibm.com/software/mqseries/`

本書を印刷された文書として注文することは**できません**。

## 本書で使用される省略語

本書全体で、次のような省略語を使用しています。

**MQ Java** MQSeries classes for Java と MQSeries classes for Java Message Service を合わせた呼び方

**MQ ベース Java** MQSeries classes for Java

**MQ JMS** MQSeries classes for Java Message Service

## 本書の対象読者

本書の内容は、 *MQSeries アプリケーション・プログラミング・ガイド* で説明されているような手続き型 MQSeries アプリケーション・プログラミングに精通したプログラマーを対象にしており、その知識をどのように MQ Java のプログラミング・インターフェースに生かせるかを扱っています。

## 本書を理解する上で必要な知識

本書を理解するためには、以下の知識や経験が必要です。

- Java プログラム言語に関する知識。
- メッセージ・キュー・インターフェース (MQI) の目的についての理解。これは、*MQSeries アプリケーション・プログラミング・ガイド* の、メッセージ・キュー・インターフェースに関する章、および *MQSeries アプリケーション・プログラミング・リファレンス* の、呼び出しの説明に関する章で扱われています。
- 全般的な MQSeries プログラムの経験、あるいは他の MQSeries 資料の内容に精通していること。

加えて、MQ ベース Java を CICS® Transaction Server for OS/390® で使用することを計画しているユーザーは、以下の点にも精通している必要があります。

- 顧客情報管理システム (CICS) の概念。

- CICS Java アプリケーション・プログラミング・インターフェース (API) の使用。
- CICS 内からの Java プログラムの実行。

VisualAge® for Java を使用した OS/390 UNIX® システム・サービス HPJ (High Performance Java) アプリケーションの開発を計画している場合には、エンタープライズ・ツールキット (OS/390 用) (VisualAge for Java Enterprise Edition for OS/390 バージョン 2 に同梱されている) にも精通している必要があります。

## 本書の使い方

第 1 部では、MQ ベース Java と MQ JMS の使い方を説明します。第 2 部では、MQ ベース Java を使用するプログラマーのための情報を扱います。第 3 部では、MQ JMS を使用するプログラマーのための情報を扱います。

はじめに、MQ ベース Java と MQ JMS について紹介した、第 1 部の章をお読みください。次に、希望される環境でクラスを使って MQSeries メッセージを送受信する方法を、第 2 部と第 3 部のプログラミングの手引きから確認できます。

なお、用語集と参考文献は、この本の最後にあります。

# 変更点の要約

このセクションでは、第 7 版の *MQSeries Java の使用* で加えられた変更を説明します。旧版の資料から変更された箇所には、左に縦線のマークが付いています。

## SD88-7163-01 での変更点

第 7 版では、 MQ Java V5.2 で追加された新しい機能に関して、変更が加えられました。次のような変更が加えられています。

- インストール手順の更新。 9ページの『第2章 インストール手順』を参照してください。
- 接続のプールのサポート。これにより、MQSeries キュー・マネージャーに対して複数の接続を使用するアプリケーションやミドルウェアのパフォーマンスが向上します。以下を参照してください。
  - 72ページの『接続プーリング』
  - 101ページの『MQEnvironment』
  - 138ページの『MQPoolServices』
  - 139ページの『MQPoolServicesEvent』
  - 141ページの『MQPoolToken』
  - 156ページの『MQQueueManager』
  - 167ページの『MQSimpleConnectionManager』
  - 172ページの『MQConnectionManager』
  - 171ページの『MQPoolServicesEventListener』
  - 179ページの『ManagedConnection』
  - 182ページの『ManagedConnectionFactory』
  - 184ページの『ManagedConnectionMetaData』
- 新しいサブスクライバー・キュー構成オプション。パブリッシュ / サブスクライブ・アプリケーションに対して、複数のキューと共用キューの両方からのアプローチを可能にします。以下を参照してください。
  - 45ページの『プロパティー』
  - 205ページの『基本サブスクライバー・キューの構成』
  - 345ページの『Topic』
  - 349ページの『TopicConnectionFactory』
- 新しいサブスクライバー終結処理ユーティリティー。サブスクライバー・オブジェクトのクローズが不完全であるがゆえに生じるあらゆる問題を防止します。209ページの『サブスクライバー・クリーンアップ・ユーティリティー』を参照してください。
- アプリケーション・サーバー機構のサポート、すなわちメッセージの並行処理。以下を参照してください。
  - 231ページの『第13章 MQ JMS アプリケーション・サーバー機構』

- 270ページの『ConnectionConsumer』
- 316ページの『QueueConnection』
- 329ページの『Session』
- 347ページの『TopicConnection』

- LDAP サーバーの構成情報の更新。 383ページの『付録C. Java オブジェクト用の LDAP サーバー構成』を参照してください。
- X/Open XA プロトコルを使用した分散トランザクションのサポート。 MQ JMS に XA クラスが組み込まれることにより、MQ JMS は、対応するトランザクション・マネージャーによって調整される 2 フェーズ・コミットに参加できます。以下を参照してください。
  - 387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』
  - 363ページの『XAConnection』
  - 364ページの『XAConnectionFactory』
  - 365ページの『XAQueueConnection』
  - 367ページの『XAQueueConnectionFactory』
  - 369ページの『XAQueueSession』
  - 370ページの『XASession』
  - 372ページの『XATopicConnection』
  - 374ページの『XATopicConnectionFactory』
  - 376ページの『XATopicSession』

## SC34-5456-05 での変更点

Linux のサポートが追加されました。

## SC34-5456-04 での変更点

### WebSphere™ と MQSeries Integrator V2 のサポート

プロダクト拡張機能として、 MQ base Java バージョン 5.1.2 が使用できるようになりました。これによって、以下のことが可能になります。

- パブリッシュ / サブスクライブのサポートを可能にする、 MQSeries Integrator for Windows NT® バージョン 2.0 への接続。詳細については、385ページの『付録D. MQSeries Integrator V2 への接続』を参照してください。
- WebSphere の CosNaming JNDI サービス・プロバイダーの使用。詳細については、38ページの『構成』を参照してください。

# 第1部 ユーザーのためのガイダンス

# 第1章 概説

この章では、MQSeries classes for Java および MQSeries classes for Java Message Service の概要と、その用途を紹介します。

## MQSeries classes for Java の概要

MQSeries classes for Java (MQ ベース Java) によって Java プログラム言語で書かれたプログラムでは、次のことが可能です。

- MQSeries クライアントとしての MQSeries への接続
- MQSeries サーバーへの直接接続

MQ ベース Java では、Java アプレット、アプリケーション、およびサーブレットから、MQSeries に呼び出しや照会を発行できます。これによって、クライアント・マシン上に他の MQSeries コードがなくても、メインフレームやレガシー・アプリケーションにアクセスすること (たいていはインターネットを経由して) が可能です。 MQ ベース Java を使用すると、インターネット端末のユーザーは、単なる情報の送信側と受信側ではなく、トランザクションの真の参加者となることができます。

## MQSeries classes for Java Message Service の概要

MQSeries classes for Java Message Service (MQ JMS) は、Sun 社の Java メッセージ・サービス (JMS) インターフェースを実装することによって、 JMS プログラムから MQSeries システムへのアクセスを可能にする、一連の Java クラスです。 JMS については、ポイント・ツー・ポイント方式とパブリッシュ / サブスクライブ方式の両方のモデルがサポートされています。

MQSeries アプリケーションを作成する際に MQ JMS を API として用いることには、いくつもの利点があります。その中には、JMS が複数のインプリメンテーションでオープン・スタンダードであるゆえの利点と、 MQ JMS にあって MQ ベース Java にはない、追加機能が持つ利点があります。

オープン・スタンダードを使用することの利点としては、次のような点があります。

- スキルとアプリケーション・コードの両面において、システム投資を無駄にしない。
- JMS アプリケーション・プログラミングのスキルを持つユーザーが使用できる。
- 様々な要件に合わせて、異なった JMS インプリメンテーションに接続できる。

JMS API の利点に関する詳細は、 Sun 社の Web サイト (`http://java.sun.com`) を参照してください。

MQ ベース Java にはない追加機能として、次のような機能があります。

- 非同期のメッセージ送達
- メッセージ・セレクター
- パブリッシュ / サブスクライブ・メッセージングのサポート

- 構造化されたメッセージ・クラス

## MQ Java の使用対象者

以下のシナリオのいずれかに該当する企業では、 MQSeries classes for Java や MQSeries classes for Java Message Service を使用することによって大きな益があります。

- イントラネット・ベースのクライアント / サーバー・ソリューションを導入しようとしている中規模または大規模の企業。この場合、インターネット・テクノロジーが低コストで容易なグローバル通信の利用を可能にする一方、 MQSeries 接続性は安全な送達と時間の独立性により高い保全性を提供します。
- パートナー企業との信頼できるビジネス間通信を必要としている中規模または大規模の企業。この場合も、インターネットが低コストで容易なグローバル通信の利用を可能にする一方、 MQSeries 接続性は安全な送達と時間の独立性により高い保全性を提供します。
- 共通インターネットから一部の企業アプリケーションへのアクセスを提供したい中規模または大規模の企業。この場合、インターネットが低コストなグローバル通信の利用を可能にする一方、 MQSeries 接続性はキューイング・パラダイムによって高い保全性を提供します。低コストに加え、1 日 24 時間使用可能であること、高速応答、および正確性の向上によって、顧客の要求を十分に満たすことができます。
- インターネット・サービス・プロバイダーまたは付加価値通信網プロバイダー。これらの企業は、インターネットによって提供される低コストで容易な通信を利用しつつ、 MQSeries 接続が持つ高い保全性によってさらに価値を付加できます。 MQSeries を利用するインターネット・サービス・プロバイダーは、 Web ブラウザーからの入力データの受信を即時に確認し、送達を保証し、さらにメッセージの状況をモニターする容易な方法を Web ブラウザーのユーザーに提供することができます。

MQSeries や MQSeries classes for Java Message Service は、企業アプリケーションにアクセスするため、また、複雑な Webアプリケーションを開発するために優れた基盤を提供します。 Web ブラウザーからのサービス要求はキューに入れられ、可能な時に処理されるので、システムの負荷に関係なく、エンド・ユーザーに適時応答を送信することができます。このキューを、ネットワークの観点でユーザーの「近く」に配置することによって、応答の時機はネットワークの負荷の影響を受けなくなります。さらに、MQSeries メッセージングのトランザクションの性質は、ブラウザーからの単純な要求をトランザクション手段で安全に一連の個別バックエンド処理に展開できるということを意味しています。

また、MQSeries classes for Java によって、アプリケーション開発者は、 Java プログラム言語の能力を用い、 Java 実行時環境がサポートされている任意のプラットフォームで実行可能なアプレットやアプリケーションを作成できるようになります。こうした要素が組み合わされると、マルチプラットフォームの MQSeries アプリケーションの開発にかかる時間は大幅に短縮されます。また、将来にアプレットの拡張を行う場合でも、これらは、アプレット・コードがダウンロードされる際にエンド・ユーザーによって自動的に選出されます。

## 接続オプション

プログラム可能オプションにより、MQ Java は、次のいずれかの方法で MQSeries に接続できます。

- 伝送制御プロトコル / インターネット・プロトコル (TCP/IP) を使用する MQSeries クライアントとして接続
- バインディング・モードで直接 MQSeries に接続

Windows NT 上の MQ ベース Java では、 VisiBroker for Java を使用して接続することも可能です。 表1 には、プラットフォームごとに使用できる接続モードが示されています。

*表 1. プラットフォームと接続モード*

| サーバー・プラットフォーム | 接続モード | | |
|---|---|---|---|
| | クライアント | | バインド |
| | 標準 | VisiBroker | |
| Windows NT | 可 | 可 | 可 |
| Windows® 2000 | 可 | 不可 | 可 |
| AIX® | 可 | 不可 | 可 |
| Sun OS (v4.1.4 以前) | 可 | 不可 | 不可 |
| Sun Solaris (v2.6、v2.8、V7、または SunOS v5.6、v5.7) | 可 | 不可 | 可 |
| OS/2® | 可 | 不可 | 可 |
| OS/400® | 可 | 不可 | 可 |
| HP-UX | 可 | 不可 | 可 |
| AT&T GIS UNIX | 可 | 不可 | 不可 |
| SINIX および DC/OSx | 可 | 不可 | 不可 |
| OS/390 | 不可 | 不可 | 可 |
| Linux | 可 | 不可 | 不可 |

**注:**

1. HP-UX の Java バインディング・サポートは、 POSIX draft10 pthread 版の MQSeries が稼働している HP-UXv11 でのみ使用できます。加えて、HP-UX Developer's Kit for Java 1.1.7 (JDK™), Release C.01.17.01 以上も必要です。
2. HP-UXv10.20、Linux、 Windows 95、および Windows 98 では、TCP/IP クライアント接続だけがサポートされています。

これらのオプションについては、この後のセクションでさらに詳しく説明します。

## クライアント接続

MQ Java を MQSeries クライアントとして使用する場合、インストール先のマシンは、 MQSeries サーバー・マシン (Web サーバーも含まれていることがある) であっても、別のマシンであっても構いません。ただし、MQ Java を Web サーバーと同じマシンにインストールすると、 MQ Java がローカルにインストールされていないマシンで MQSeries クライアント・アプリケーションをダウンロードおよび実行できる、という利点があります。

クライアントのインストールを選択した場合は常に、次の 3 つの異なるモードでインストールを実行できます。

**Java が使用可能な任意の Web ブラウザーから実行**
このモードでは、アクセスできる MQSeries キュー・マネージャーの位置を、使用するブラウザーのセキュリティー制限によって制約できます。

**アプレット・ビューアーを使用して実行**
この方式を使用するためには、クライアント・マシンに Java Developer Kit (JDK) か Java Runtime Environment (JRE) がインストールされていなければなりません。

**スタンドアロン Java プログラムとして実行、または Web アプリケーション・サーバーから実行**
この方式を使用するためには、クライアント・マシンに Java Developer Kit (JDK) か Java Runtime Environment (JRE) がインストールされていなければなりません。

## VisiBroker for Java の使用

Windows プラットフォームでは、通常の MQSeries クライアント・プロトコルを使用する接続の代替手段として、 VisiBroker を使用した接続がサポートされています。このサポートは、 VisiBroker for Java を Netscape Navigator と組み合わせて使用することによって有効になり、MQSeries サーバー・マシン上には、 VisiBroker for Java と MQSeries オブジェクト・サーバーが必要です。適切なオブジェクト・サーバーが MQ ベース Java とともに提供されます。

## バインド接続

バインディング・モードで使用すると、MQ Java は、ネットワーク経由で通信するのではなく、 Java Native Interface (JNI) を使用して、既存のキュー・マネージャー API を直接呼び出します。これで、MQSeries アプリケーションのパフォーマンスがネットワーク接続を使用する場合より良好になります。クライアント・モードとは異なり、バインディング・モードで作成されたアプリケーションは、アプレットとしてダウンロードすることができません。

バインド接続を使用するためには、 MQSeries サーバーに MQ Java がインストールされていなければなりません。

# 前提条件

MQ ベース Java の実行には、以下のソフトウェアが必要です。

- 使用するサーバー・プラットフォームに対応した MQSeries。
- 使用するサーバー・プラットフォームに対応した Java Developers Kit (JDK)。
- クライアント・プラットフォームに対応した Java Developers Kit、Java Runtime Environment (JRE)、または Java 使用可能 Web ブラウザー。 (5ページの『クライアント接続』を参照してください。)

  **注:** Web ブラウザー内で MQ ベース Java アプレット (たとえば、インストール検査プログラム) を実行するには、 Java 1.1.6 アプレットを実行できるブラ

ウザーが必要です。 Sun システムの HotJava™、 Netscape Navigator 4、および Microsoft® Internet Explorer 4 がこの要件を満たしているブラウザーです。

- VisiBroker for Java (Windows 上で稼働させ、VisiBroker 接続を使用する場合のみ)。
- OS/390 の場合は、 OS/390 バージョン 2.5 と UNIX システム・サービス。
- OS/400 の場合は、 AS/400® Developer Kit for Java (5769-JV1)、および OS/400 Qshell Interpreter (5769-SS1) オプション 30。

MQ JMS 管理ツールを使用する (37ページの『第5章 MQ JMS 管理ツールの使用』を参照してください) ためには、この他に以下のソフトウェアが必要です。

- 以下のサービス・プロバイダー・パッケージのうち、少なくとも 1 つ。
  - Lightweight Directory Access Protocol (LDAP) - `ldap.jar`、`providerutil.jar`。
  - ファイル・システム - `fscontext.jar`、`providerutil.jar`。
- Java Naming and Directory Service (JNDI) サービス・プロバイダー。これは、管理対象オブジェクトの物理表現を保管するリソースです。 MQ JMS では、おそらく LDAP サーバーがこの目的で使用されますが、このツールでは、ファイル・システム・コンテキスト・サービス・プロバイダーの使用もサポートされます。LDAP サーバーを使用する場合は、JMS オブジェクトの保管用に構成することが必要です。この構成についての説明は、383ページの『付録C. Java オブジェクト用の LDAP サーバー構成』を参照してください。

MQ JMS の XOpen/XA 機構を使用する場合は、MQSeries V5.2 が必要です。

# 第2章 インストール手順

この章では、MQSeries classes for Java および MQSeries classes for Java Message Service 製品のインストール方法を説明します。

## MQSeries classes for Java と MQSeries classes for Java Message Service のインストール

この製品は、AIX、 AS/400、HP-UX、Linux、 Sun Solaris、および Windows プラットフォームで使用できます。以下のコンポーネントが含まれています。

- MQSeries classes for Java (MQ ベース Java) バージョン 5.2.0
- MQSeries classes for Java Message Service (MQ JMS) バージョン 5.2 (AS/400 は除く)

それぞれのプラットフォームで使用可能な接続は、5ページの『接続オプション』でご確認ください。

この製品は、圧縮フォーマットのファイルで MQSeries の Web サイト (http://www.ibm.com/software/mqseries/) から入手できます。

**注:** OS/390 の場合、 MQ ベース Java は、MQSeries のサポートパック™ として提供されており、http://www.ibm.com/software/mqseries/ からダウンロードできます。

ただ単に MQ ベース Java クラスの最新バージョンを使用するのであれば、 MQ ベース Java バージョン 5.2.0 をインストールするだけでも構いません。しかし、MQ JMS アプリケーションを使用するのであれば、 MQ ベース Java と MQ JMS の両方をインストールする必要があります (これらは、2 つ合わせて MQ Java と呼ばれます)。

MQ ベース Java は、次の Java .jar ファイルにあります。

**com.ibm.mq.jar** このコードにはすべての接続オプションのサポートが含まれています。

**com.ibm.mq.iiop.jar** このコードは VisiBroker 接続しかサポートしません。これは、Windows プラットフォームでのみ提供されます。

**com.ibm.mqbind.jar** このコードは、バインド接続しかサポートしておらず、一部のプラットフォームでは提供またはサポートされていません。新規アプリケーションでは、このコードをご使用にならないようお勧めします。

MQ JMS は、次の Java .jar ファイルにあります。

**com.ibm.mqjms.jar**

### MQ ベース Java と MQ JMS のインストール

MQ JMS 製品には、 Sun Microsystems 社が提供する以下の Java ライブラリーが改めて配布されています。

| | |
|---|---|
| **connector.jar** | Version 1.0 Public Draft |
| **fscontext.jar** | Early Access 4 Release |
| **jms.jar** | バージョン 1.0.2 |
| **jndi.jar** | バージョン 1.1.2 |
| **ldap.jar** | バージョン 1.0.3 |
| **providerutil.jar** | バージョン 1.0 |

**注:** OS/390 では、 **com.ibm.mq.jar** ファイルしか提供されていません。このファイルは、UNIX システム・サービスと CICS Transaction Server for OS/390 の両方からの MQSeries へのバインド接続をサポートします。

インストールの指示は、プラットフォームごとに分かれています。ご使用になりたいプラットフォームに関係のあるセクションを参照してください。

| | |
|---|---|
| **AIX、 HP-UX、および Sun Solaris** | 『UNIX でのインストール』 |
| **AS/400** | 11ページの『AS/400 でのインストール』 |
| **Linux** | 12ページの『Linux でのインストール』 |
| **Windows** | 13ページの『Windows でのインストール』 |

インストールが完了すると、ファイルやサンプルは、 13ページの『インストール・ディレクトリー』 に示された場所にインストールされます。

また、インストールの終了後には、 13ページの『環境変数』 に示されているようにして、環境変数を更新してください。

**注:** 製品をインストールした後でベースの MQSeries をインストールまたは再インストールする場合には、注意が必要です。 MQSeries Java のサポートが前のレベルに戻ってしまいますので、 MQ ベース Java バージョン 5.1 をインストールすることがないようにしてください。

## UNIX でのインストール

このセクションでは、 MQ Java を AIX、 HP-UX、および Sun Solaris にインストールする方法について説明します。 Linux での MQ ベース Java のインストールについては、12ページの『Linux でのインストール』を参照してください。

**注:** クライアントのみのインストールの場合 (つまり、 MQSeries サーバーがインストールされていない場合) は、グループおよびユーザー ID mqm をセットアップする必要があります。詳細については、ご使用のプラットフォームに該当する MQSeries インストール・ガイドを参照してください。

1. root としてログオンします。
2. バイナリー・フォーマットでファイル `ma88_xxx.tar.Z` をコピーし、ディレクトリー /tmp に保管します。なお、ファイル名の *xxx* の部分には、次の中から適当なプラットフォーム ID が入ります。

- aix AIX
- hp10 HP-UXv10
- hp11 HP-UXv11
- sol Sun Solaris

3. 次のコマンドを入力します (ここで、 *xxx* には適当なプラットフォーム ID が入ります)。

```
uncompress -fv /tmp/ma88_xxx.tar.Z
tar -xvf /tmp/ma88_xxx.tar
rm /tmp/ma88_xxx.tar
```

これらのコマンドによって、必要なファイルとディレクトリーが作成されます。

4. それぞれのプラットフォームに合わせてインストール・ツールを使用します。
   - AIX では、 smitty を使用し、以下のタスクを実行します。
     a. mqm.java で始まるコンポーネントをすべてアンインストールします。
     b. コンポーネントを /tmp ディレクトリーからインストールします。
   - HP-UX では、sam を使用し、ファイル ma88_hp10 または ma88_hp11 の適当な方からインストールを実行します。

     **注:** Java は、コード・ページ 1051 (HP-UX のデフォルト) をサポートしていません。そのため、HP-UX でパブリッシュ / サブスクライブ・ブローカーを実行するためには、ブローカーのキュー・マネージャーの CCSID を代わりの値 (たとえば、 819 など) に変更する必要があるかもしれません。
   - Sun Solaris では、次のコマンドを入力して、必要なオプションを選択します。

```
pkgadd -d /tmp mqjava
```

次いで、次のコマンドを入力します。

```
rm -R /tmp/mqjava
```

## AS/400 でのインストール

このセクションでは、 MQ ベース Java を AS/400 にインストールする方法について説明します。

1. ファイル ma88_400.zip を PC 上の特定のディレクトリーにコピーします。
2. InfoZip の Unzip 機能を使用してファイルを圧縮解除します。
   すると、ma88_400.sav というファイルが作成されます。
3. AS/400 上の適切なライブラリー (たとえば、QGPL ライブラリーなど) に、MA88 という保管ファイルを作成します。

```
CRTSAVF FILE(QGPL/MA88)
```

4. ma88_400.sav を、バイナリー・イメージとしてこの保管ファイルに転送します。 FTP を使用してこれを行う場合、入力コマンドは次の例のようになります。

```
PUT C:¥TEMP¥MA88_400.SAV QGPL/MA88
```

5. RSTLICPGM を使用して MQSeries classes for Java (製品 ID 5648C60) をインストールします。

```
RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
```

6. 11ページの3 のステップで作成した保管ファイルを削除します。

```
DLTF FILE(QGPL/MA88)
```

## Linux でのインストール

このセクションでは、MQ Java を Linux にインストールする方法について説明します。

Linux の場合は、 ma88_linux.tgz と MQSeriesJava-5.2.0-1.noarch.rpm という 2 つのインストール・ファイルが使用できます。実行されるインストールは、どちらのファイルを使用しても同じです。

ターゲット・システムに対して root アクセス権を持っている場合や、 Red Hat Package Manager (RPM) データベースを使用してパッケージをインストールする場合は、 Java-5.2.0-1.noarch.rpm を使用してください。

ターゲット・システムに対して root アクセス権を持たない場合や、ターゲット・システムに RPM がインストールされていない場合は、 ma88_linux.tgz を使用してください。

ma88_linux.tgz を使用してインストールを実行する場合は、次のようにします。

1. 製品のインストール・ディレクトリー (たとえば、/opt) を選択します。

   ホーム・ディレクトリー外のディレクトリーを選択すると、root としてのログインが必要になる場合があります。
2. ファイル ma88_linux.tgz をホーム・ディレクトリーにコピーします。
3. 選択したインストール・ディレクトリーに移動します。たとえば、次のようにします。

   ```
   cd /opt
   ```
4. 次のコマンドを入力します。

   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

   すると、現行ディレクトリー (例では /opt) 内に mqm というディレクトリーが作成されます。

MQSeriesJava-5.2.0-1.noarch.rpm を使用してインストールを実行する場合は、次のようにします。

1. root としてログインします。
2. MQSeriesJava-5.2.0-1.noarch.rpm を作業ディレクトリーにコピーします。
3. 次のコマンドを入力します。

   ```
   rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
   ```

   すると、/opt/mqm/ に製品がインストールされます。インストール先のパスは、変更することも可能です (詳細については、お持ちの RPM 資料を参照してください)。

## Windows でのインストール

このセクションでは、 MQ Java を Windows にインストールする方法について説明します。

1. tmp という空のディレクトリーを作成し、それを現行ディレクトリーにします。
2. ファイル `ma88_win.zip` をこのディレクトリーにコピーします。
3. InfoZip の Unzip 機能を使用して `ma88_win.zip` を圧縮解除します。
4. このディレクトリーから `setup.exe` を実行し、ウィンドウに表示されるプロンプトの指示に従います。

   **注:** MQ ベース Java だけをインストールする場合は、関係するオプションをこの段階で選択してください。

## インストール・ディレクトリー

MQ Java V5.2 ファイルは、表2 に示されているディレクトリーにインストールされます。

*表 2. 製品のインストール・ディレクトリー*

| プラットフォーム | ディレクトリー |
|---|---|
| AIX | usr/mqm/java/ |
| AS/400 | /QIBM/ProdData/mqm/java/ |
| HP-UX および Sun Solaris | opt/mqm/java/ |
| Linux | *install_dir*/mqm/java/ |
| Windows 95、 98、2000、および NT | *install_dir*¥ |
| **注:** *install_dir* は、製品がインストールされたディレクトリーです。 Linux の場合は、おそらく /opt がこれに相当します。 | |

## 環境変数

インストールの後には、 MQ ベース Java コードとサンプル・ディレクトリーを組み込めるように CLASSPATH 環境変数を更新する必要があります。 表3 は、各種プラットフォームの代表的な CLASSPATH 設定を示したものです。

*表 3. 製品のためのサンプル CLASSPATH ステートメント*

| プラットフォーム | サンプル CLASSPATH |
|---|---|
| AIX | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>/usr/mqm/java/lib/com.ibm.mq.jar:<br>/usr/mqm/java/lib/connector.jar:<br>/usr/mqm/java/lib:<br>/usr/mqm/java/samples/base: |
| HP-UX および Sun Solaris | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>/opt/mqm/java/lib/com.ibm.mq.jar:<br>/opt/mqm/java/lib/connector.jar:<br>/opt/mqm/java/lib:<br>/opt/mqm/java/samples/base: |

*表 3. 製品のためのサンプル CLASSPATH ステートメント (続き)*

| プラットフォーム | サンプル CLASSPATH |
|---|---|
| Windows 95、98、2000、および NT | CLASSPATH=C:*jdk_dir*¥lib¥classes.zip;<br>*install_dir*¥lib¥com.ibm.mq.jar;<br>*install_dir*¥lib¥com.ibm.mq.iiop.jar;<br>*install_dir*¥lib¥connector.jar;<br>*install_dir*¥lib¥;<br>*install_dir*¥samples¥base¥; |
| AS/400 | CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar:<br>/QIBM/ProdData/mqm/java/lib/connector.jar:<br>/QIBM/ProdData/mqm/java/lib:<br>/QIBM/ProdData/mqm/java/samples/base: |
| Linux | CLASSPATH=*jdk_dir*/lib/classes.zip:<br>*install_dir*/mqm/java/lib/com.ibm.mq.jar:<br>*install_dir*/mqm/java/lib/connector.jar:<br>*install_dir*/mqm/java/lib:<br>*install_dir*/mqm/java/samples/base: |
| **注:**<br>1. *jdk_dir* は、 JDK がインストールされたディレクトリーです。<br>2. *install_dir* は、製品がインストールされたディレクトリーです。 | |

MQ JMS を使用する場合は、クラスパスに組み込まなければならない jar ファイルが他にもあります。これらは、25ページの『インストール後のセットアップ』 にリストされています。

使用すべきでないバインディング・パッケージ `com.ibm.mqbind` と依存関係にあるアプリケーションが存在する場合は、ファイル `com.ibm.mqbind.jar` もクラスパスに追加する必要があります。

表4 に示されているように、いくつかのプラットフォームでは、さらに他の環境変数も更新する必要があります。

*表 4. 製品のための環境変数*

| プラットフォーム | 環境変数 |
|---|---|
| AIX | LD_LIBRARY_PATH=/usr/mqm/java/lib |
| HP_UX | SHLIB_PATH=/opt/mqm/java/lib |
| Sun Solaris | LD_LIBRARY_PATH=/opt/mqm/java/lib |
| Windows 95、98、2000、および NT | PATH=*install_dir*¥lib |
| **注:** *install_dir* は、製品のインストール・ディレクトリーです。 | |

**注:**

1. MQSeries Java バインディング を OS/400 で使用する場合は、必ず、ライブラリー・リストの中にライブラリー QMQMJAVA を含めてください。

2. MQSeries 変数が付加され、既存のシステム環境変数が一切上書きされていないことを確認してください。既存のシステム環境変数が上書きされると、コンパイル時や実行時にアプリケーションの障害が発生する恐れがあります。

## Web サーバーの構成

MQSeries Java を Web サーバーにインストールすると、 MQSeries Java がローカルにインストールされていないマシンで MQSeries Java をダウンロードおよび実行することが可能になります。 MQSeries Java ファイルから Web サーバーにアクセスできるようにするには、クライアントがインストールされているディレクトリーを指示するように、 Web サーバーの構成をセットアップしなければなりません。この構成の方法に関する詳細は、Web サーバーの資料を参照してください。

**注:** OS/390 の場合、インストールされるクラスはクライアント接続をサポートしておらず、有効にクライアントにダウンロードされません。しかし、他のプラットフォームの jar ファイルを OS/390 に転送すれば、それをクライアントに対して使用することは可能です。

# 第3章 MQSeries classes for Java (MQ ベース Java) の使用

この章では、以下の内容を扱います。

- MQ ベース Java のインストールを検査するためのサンプル・アプレットやアプリケーション・プログラムを実行させるようにシステムを構成する方法
- 独自のプログラムを実行する場合の手順の修正方法

手順は、使用したい接続オプションによって異なります。自分の要件にあったセクションの指示に従ってください。

## TCP/IP クライアントを検査するためのサンプル・アプレットの使用

MQ ベース Java には、mqjavac.html というインストール検査アプレットが組み込まれています。このアプレットを使用して、TCP/IP 接続されている、MQ ベース Java のクライアント・モードを検査できます。 (19ページの『サンプル・アプリケーションを使用した検査』も参照してください。)

このアプレットは指定のキュー・マネージャーに接続され、 MQSeries 呼び出しをすべて実行し、なんらかの障害があればその診断メッセージを生成します。

アプレットは、 JDK と共に提供されているアプレット・ビューアーで実行できます。アプレット・ビューアーは、どのホストにあるキュー・マネージャーでもアクセスできます。

どの場合でも、アプレットが正常に完了しないときには、診断メッセージに示されている助言に従い、アプレットを実行し直してください。

### AS/400 でのサンプル・アプレットの使用

OS/400 オペレーティング・システムには、もともと装備されているグラフィカル・ユーザー・インターフェース (GUI) がありません。そのため、サンプル・アプレットを実行する場合には、グラフィックスが有効なハードウェアで、 Remote Abstract Window Toolkit for Java (AWT) か Class Broker for Java (CBJ) を使用する必要があります。コマンド行からクライアントを検査することも可能です (19ページの『サンプル・アプリケーションを使用した検査』を参照してください)。

## クライアント接続を受け入れるためのキュー・マネージャーの構成

クライアントからの着信接続要求を受け入れるようにキュー・マネージャーを構成するには、以下の手順に従います。

### TCP/IP クライアント

1. 以下の手順を使用してサーバー接続チャネルを定義します。

   **AS/400 以外のプラットフォームの場合:**

   a. strmqm コマンドを使用してキュー・マネージャーを開始します。

   b. 次のコマンドを入力して runmqsc プログラムを開始します。

```
runmqsc
```

c. 次のように入力することで、JAVA.CHANNEL と呼ばれるサンプル・チャネルを定義します。

```
DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for MQSeries Client for Java')
```

**AS/400 プラットフォームの場合:**

a. STRMQM コマンドを使用してキュー・マネージャーを開始します。

b. 次のように入力することで、JAVA.CHANNEL と呼ばれるサンプル・チャネルを定義します。

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
MCAUSERID(SOMEUSERID) TEXT('Sample channel for MQSeries Client for Java')
```

ここで、QMGRNAME はキュー・マネージャーの名前を表し、SOMEUSERID は MQSeries リソースに対して適当な権限を持つ AS/400 ユーザー ID を表します。

2. 以下のコマンドを使用してリスナー・プログラムを開始します。

**OS/2 および NT オペレーティング・システムの場合:**

次のコマンドを実行します。

```
runmqlsr -t tcp [-m QMNAME] -p 1414
```

**注:** デフォルトのキュー・マネージャーを使用する場合は、-m オプションを省略できます。

**Windows NT オペレーティング・システムで VisiBroker for Java を使用する場合:** 次のコマンドで IIOP (Internet Inter-ORB Protocol) サーバーを開始します。

```
java com.ibm.mq.iiop.Server
```

**注:** IIOP サーバーを停止するには、次のコマンドを発行します。

```
java com.ibm.mq.iiop.samples.AdministrationApplet shutdown
```

**UNIX オペレーティング・システムの場合:**

inetd が MQSeries チャネルを開始するように、inetd デーモンを構成します。この処理を行う方法については、*MQSeries クライアント* を参照してください。

**OS/400 オペレーティング・システムの場合:**

次のコマンドを実行します。

```
STRMQMLSR MQMNAME(QMGRNAME)
```

ここで、QMGRNAME はキュー・マネージャーの名前を表します。

## appletviewer からの実行

この方式を使用するためには、マシンに Java Developer's Kit (JDK) がインストールされていなければなりません。

**ローカル・インストール手順**

1. 使用する言語のサンプル・ディレクトリーに移動します。
2. 次のように入力します。

```
appletviewer mqjavac.html
```

**Web サーバー・インストール手順**

次のコマンドを入力します。

```
appletviewer http://Web.server.host/MQJavaclient/mqjavac.html
```

**注:**

1. 一部のプラットフォームの場合、このコマンドは「appletviewer」ではなく「applet」です。
2. 一部のプラットフォームでは、画面の左上の「アプレット」メニューから「プロパティー」を選択し、「ネットワーク・アクセス」を「制約なし」に設定することが必要な場合があります。

この手法を使用して、 TCP/IP アクセス権がある任意のホストで実行中の任意のキュー・マネージャーに接続できるようにする必要があります。

## 検査アプレットのカスタマイズ

mqjavac.html ファイルには、いくつかのオプション・パラメーターが組み込まれています。これらのパラメーターにより、ユーザーの要件に適合するようにアプレットを変更することができます。各パラメーターは、次のような HTML の行に定義されています。

```
<!PARAM name="xxx" value="yyy">
```

パラメーター値を指定するには、先頭の感嘆符を取り除き、必要に応じて値を編集します。以下のパラメーターを指定できます。

**hostname** ホスト名の編集ボックスに初めに表示する値。

**port** ポート番号の編集ボックスに初めに表示する値。

**channel** チャネルの編集ボックスに初めに表示する値。

**queueManager** キュー・マネージャーの編集ボックスに初めに表示する値。

**userID** キュー・マネージャーへの接続時に指定したユーザー ID。

**password** キュー・マネージャーへの接続時に指定したパスワード。

**trace** MQ ベース Java がトレース・ログを書き込むようにします。このオプションは、IBM サービスの指示があったときだけ使用してください。

# サンプル・アプリケーションを使用した検査

インストール検査プログラム MQIVP は MQ ベース Java に同梱されています。このアプリケーションを使用して、MQ ベース Java の接続モードをすべてテストできます。このプログラムでは、いくつかの選択項目とその他のデータについてプロンプトが表示され、検査対象の接続モードを判別します。インストールを検査するには、以下の手順に従ってください。

1. クライアント接続をテストする場合は、次のようにします。
   a. キュー・マネージャーを 17ページの『クライアント接続を受け入れるためのキュー・マネージャーの構成』の説明に従って構成します。

b. 残りの手順をクライアント・マシンで実行します。

バインド接続をテストする場合は、残りの手順を MQSeries サーバー・マシンで実行します。

2. サンプル・ディレクトリーに変更します。
3. 次のように入力します。

```
java MQIVP
```

このプログラムは、以下の処理を実行します。

a. 指定されたキュー・マネージャーへの接続、およびそのキュー・マネージャーからの切断。

b. システム・デフォルト・ローカル・キューのオープン、書き込み、読み取り、およびクローズ。

c. これらの処理が正常に終了したかどうかを示すメッセージの表示。

4. プロンプト (1) では、デフォルトのまま「MQSeries」にしておきます。
5. プロンプト (2) では、次のようにします。
   - TCP/IP 接続を使用する場合は、MQSeries サーバーのホスト名を入力します。
   - もともと装備されている接続 (バインディング・モード) を使用する場合は、フィールドをブランクのままにしておきます。(名前は入力しないでください。)

以下は、表示されるプロンプトと応答の例です。実際のプロンプトと応答は、使用している MQSeries ネットワークによって異なります。

```
Please enter the type of connection (MQSeries)              : (MQSeries)(1)
Please enter the IP address of the MQSeries server          : myhost(2)
Please enter the port to connect to                         : (1414)(3)
Please enter the server connection channel name              : JAVA.CHANNEL(3)
Please enter the queue manager name                         :
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager

Tests complete -
SUCCESS: This transport is functioning correctly.
Press Enter to continue...
```

**注:**

1. サーバー接続を選択すると、(3) のマークが付いたプロンプトは表示されません。
2. OS/390 の場合、プロンプト (1)、 (2)、または (3) は表示されません。
3. OS/400 の場合、コマンド `java MQIVP` は、 Qshell 対話式インターフェース (Qshell は、 OS/400 (5769-SS1) のオプション 30) からのみ実行できます。別の方法として、 CL コマンド `RUNJVA CLASS(MQIVP)` を使用してアプリケーションを実行することもできます。
4. Java の MQSeries バインディングを OS/400 で使用する場合は、必ず、ライブラリー・リストにライブラリー QMQMJAVA を含めてください。

### VisiBroker 接続の使用

VisiBroker を使用する場合は、17ページの『クライアント接続を受け入れるためのキュー・マネージャーの構成』の手順を必要としません。

VisiBroker を使用したインストールをテストする場合には、 19ページの『サンプル・アプリケーションを使用した検査』で説明されている手順を使用します。ただし、プロンプト (1) では、大文字小文字とも正確に `VisiBroker` と入力してください。

### CICS Transaction Server for OS/390 の使用

1. CICS にサンプル・アプリケーション・プログラムを定義します。
2. サンプル・アプリケーションを実行するトランザクションを定義します。
3. 標準出力に使用するファイルに、キュー・マネージャー名を書き込みます。
4. トランザクションを実行します。

プログラムの出力は、標準出力やエラー出力に使用されるファイルに入れられます。

Java プログラムの実行と、入力および出力ファイルの設定に関する詳細は、 CICS 資料を参照してください。

## ユーザー独自の MQ ベース Java プログラムの実行

ユーザー独自の Java アプレットまたはアプリケーションを実行する場合は、検査プログラムについて説明されている手順で、「mqjavac.html」または「MQIVP」の代わりに自分のアプリケーション名を使用します。

MQ ベース Java アプリケーションおよびアプレットの作成については、 51ページの『第2部 MQ ベース Java を使ったプログラミング』を参照してください。

## MQ ベース Java 問題の解決

プログラムが正常に完了しない場合は、インストール検査アプレットかインストール検査プログラムを実行し、診断メッセージに示されるアドバイスに従ってください。これらのプログラムについては、いずれも 17ページの『第3章 MQSeries classes for Java (MQ ベース Java) の使用』で説明されています。

問題が解決せず、IBM サービスに連絡する必要があるときは、トレース機能をオンにするようお願いする場合があります。トレースをオンにする方法は、クライアント・モードで実行している場合と、バインディング・モードで実行している場合とで異なります。以下のセクションでは、ご使用のシステムに合わせて手順を参照してください。

### サンプル・アプレットのトレース

サンプル・アプレットでトレースを実行する場合は、`mqjavac.html` ファイルを編集します。次の行を見つけてください。

```
<!PARAM name="trace" value="1">
```

この行で、感嘆符を取り除き、必要な詳細レベルに応じて 1 ～ 5 の数値に変更します。 (この数値が大きいほど、詳細な情報が収集されます。) この行は次のようになります。

```
<PARAM name="trace" value="n">
```

ここで、「n」は 1 ～ 5 の数値です。

トレース出力は Java コンソールまたは Web ブラウザーの Java ログ・ファイルに表示されます。

## サンプル・アプリケーションのトレース

MQIVP プログラムをトレースするには、次のように入力します。

```
java MQIVP -trace n
```

ここで、「n」は 1 ～ 5 の数値で、必要な詳細レベルに応じて変更します。 (この数値が大きいほど、詳細な情報が収集されます。)

トレースの使用法に関する詳細は、80ページの『MQ ベース Java プログラムのトレース』を参照してください。

### CICS Transaction Server for OS/390 でのトレース

CICS Transaction Server for OS/390 を使用する場合は、プログラムに直接、コマンド行の引き数を入力することができません。この場合は、その引き数で MQIVP.main() を呼び出す、小さなラッパー・プログラムを作成する必要があります。

## エラー・メッセージ

通常、よく表示されるエラー・メッセージは、次のとおりです。

**ローカル・ホスト IP アドレスを識別できません**

サーバーがネットワークに接続されていません。

*推奨処置*: サーバーをネットワークに接続してから再試行してください。

**ファイル gatekeeper.ior をロードできません**

この障害は、gatekeeper.ior ファイルが正しい場所に入っていない場合に、 VisiBroker アプレットが配置されている Web サーバーで発生する可能性があります。

*推奨処置*: そのアプレットが配置されているディレクトリーから VisiBroker Gatekeeper を再始動してください。 gatekeeper ファイルがこのディレクトリーに書き込まれます。

**障害: ソフトウェア (MQSeries) または VBROKER_ADM 変数が抜けていると考えられます**

この失敗は、Java ソフトウェア環境が不完全な場合に、MQIVP サンプル・プログラムで発生します。

*推奨処置*: クライアント側で、 VBROKER_ADM 環境変数が VisiBroker for Java 管理 (adm) ディレクトリーに設定されていることを確認してから再試行してください。

サーバーで、最新バージョンの MQ ベース Java がインストールされていることを確認してから再試行してください。

**NO_IMPLEMENT**

VisiBroker Smart エージェントが関係している通信上の問題があります。

*推奨処置*: VisiBroker の資料を調べてください。

**COMM_FAILURE**

VisiBroker Smart エージェントが関係している通信上の問題があります。

*推奨処置*: すべての VisiBroker Smart エージェントに同じポート番号を使用した状態で再試行してください。 VisiBroker の資料を調べてください。

**MQRC_ADAPTER_NOT_AVAILABLE**

VisiBroker の使用を試行してこのエラーが発生した場合は、 CLASSPATH 内に JAVA クラス org.omg.CORBA.ORB が存在していないことが考えられます。

*推奨処置*: CLASSPATH ステートメントに VisiBroker の `vbjorb.jar` および `vbjapp.jar` ファイルへのパスが含まれていることを確認してください。

**MQRC_ADAPTER_CONN_LOAD_ERROR**

OS/390 上で実行していてこのエラーが表示された場合は、 STEPLIB ステートメントに MQSeries SCSQANLE および SCSQAUTH データ・セットが含まれていることを確認してください。

# 第4章 MQSeries classes for Java Message Service (MQ JMS) の使用

この章では、以下のタスクについて説明します。

- テストおよびサンプル・プログラムを使用するためのシステムのセットアップ方法
- MQSeries classes for Java Message Service のインストールを検査するためのポイント・ツー・ポイントの Installation Verification Test (IVT) プログラムの実行方法
- パブリッシュ / サブスクライブのインストールを検査するためのパブリッシュ / サブスクライブの Installation Verification Test (IVT) プログラムの実行方法
- 独自のプログラムの実行方法

## インストール後のセットアップ

MQ JMS で、必要なリソースをすべて使用できるようにするには、以下のシステム変数を更新する必要があります。

### クラスパス

JMS プログラムを正常に動作させるためには、いくつかの Java パッケージを JVM で使用できるようにしなければなりません。そのためには、必要なパッケージを入手し、インストールした後、クラスパスにそれらを指定する必要があります。

以下の .jar ファイルをクラスパスに追加してください。

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar
- jms.jar
- jndi.jar
- jta.jar
- ldap.jar
- providerutil.jar

### 環境変数

MQ JMS のインストール・ディレクトリーにある、 `bin` サブディレクトリーには、いくつかのスクリプトがあります。これらは、いくつかの共通の処置を行うための便利なショートカットとして用意されています。これらのスクリプトの多くは、環境変数 `MQ_JAVA_INSTALL_PATH` が定義されていて、そこに MQ JMS のインストール・ディレクトリーが指示されていることを前提としています。この変数の設定は必須ではありませんが、もしこの変数を設定しないのであれば、それに応じて `bin` ディレクトリー内のスクリプトも編集する必要があります。

Windows NT では、**「システムのプロパティ」**の**「環境」**タブを使用して、クラスパスや新しい環境変数を設定することが可能です。また、UNIX の場合は、これらを通常のように各ユーザーのログオン・スクリプトから設定できます。プラットフォームに関係なく、各種のプロジェクトのための様々なクラスパスや他の環境変数を保守する場合に、スクリプトを使用することも可能です。

## パブリッシュ / サブスクライブ・モードのための付加的なセットアップ

JMS パブリッシュ / サブスクライブの MQ JMS インプリメンテーションを使用するためには、いくつかの付加的なセットアップが必要です。

### ブローカーの稼働の確認

MQSeries パブリッシュ / サブスクライブのインストールと稼働を検査する際は、次のコマンドを使用します。

```
dspmqbrk -m MY.QUEUE.MANAGER
```

ここで、MY.QUEUE.MANAGER は、ブローカーが稼働しているキュー・マネージャーの名前を表します。ブローカーが稼働していれば、次の例のようなメッセージが表示されます。

```
MQSeries message broker for queue manager MY.QUEUE.MANAGER running.
```

システムが、dspmqbrk コマンドを実行できないことを報告する場合は、MQSeries パブリッシュ / サブスクライブ・ブローカーが正しくインストールされているかどうかを確認してください。

ブローカーがアクティブになっていないことが報告された場合は、次のコマンドを使用してブローカーを開始します。

```
strmqbrk -m MY.QUEUE.MANAGER
```

### MQ JMS システム・キューの作成

MQ JMS パブリッシュ / サブスクライブを正しく機能させるためには、いくつかのシステム・キューを作成する必要があります。 MQ JMS のインストール・ディレクトリーにある、bin サブディレクトリーには、この作業を助けるためのスクリプトが提供されています。このスクリプトは、次のコマンドで使用できます。

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

エラーが発生する場合は、キュー・マネージャー名が正しく入力されているかどうか、そして、そのキュー・マネージャーが稼働しているかどうかを確認してください。

## 特権を持たないユーザーには権限が必要なキュー

JMS が使用するキューにアクセスする場合、特権を持たないユーザーは、権限の認可を受ける必要があります。 MQSeries のアクセス制御に関する詳細は、*MQSeries システム管理の手引き* で、 MQSeries オブジェクトの保護に関する章を参照してください。

JMS ポイント・ツー・ポイント・モードでは、以下のように、MQSeries classes for Java と同様のアクセス制御の発行が用いられます。

- QueueSender が使用するキューには書き込み権限が必要。
- QueueReceivers と QueueBrowsers が使用するキューには、読み取り、照会、およびブラウズ権限が必要。
- QueueSession.createTemporaryQueue 方式には、 QueueConnectionFactory temporaryModel フィールドに定義されたモデル・キュー (デフォルトでは、SYSTEM.DEFAULT.MODEL.QUEUE) へのアクセス権限が必要。

JMS パブリッシュ / サブスクライブ・モードでは、以下のシステム・キューが使用されます。

SYSTEM.JMS.ADMIN.QUEUE

SYSTEM.JMS.REPORT.QUEUE

SYSTEM.JMS.MODEL.QUEUE

SYSTEM.JMS.PS.STATUS.QUEUE

SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.SUBSCRIBER.QUEUE

SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE

SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

SYSTEM.BROKER.CONTROL.QUEUE

加えて、メッセージの発行を行うすべてのアプリケーションには、使用するトピック接続ファクトリーに指定された STREAM キューへのアクセス権限が必要です。デフォルトは、次のキューです。

SYSTEM.BROKER.DEFAULT.STREAM

## ポイント・ツー・ポイント IVT の実行

このセクションでは、MQ JMS と一緒に提供されている、ポイント・ツー・ポイントのインストール検査テスト・プログラム (IVT) について説明します。

IVT は、MQ JMS をバインディング・モードで使用し、ローカル・マシン上のデフォルト・キュー・マネージャーに接続することによって、インストールの検査を試行します。このとき、キュー・マネージャーは、 SYSTEM.DEFAULT.LOCAL.QUEUE キューにメッセージを送り、それから反対にそれをもう 1 度読み取る、という処理を行います。

このプログラムは、次の 2 つのモードで実行できます。

**管理対象オブジェクトの JNDI ルックアップを使用する**

JNDI モードでは、JMS クライアント・アプリケーションの操作に使用する管理対象オブジェクトを、強制的に JNDI ネーム・スペースから取得します (管理対象オブジェクトについての詳細は、 42ページの『JMS オブジェクトの管理』を参照してください)。この呼び出し方式には、管理ツールと同じ前提条件 (37ページの『第5章 MQ JMS 管理ツールの使用』を参照してください) が適用されます。

**管理対象オブジェクトの JNDI ルックアップを使用しない**

JNDI の使用を望まない場合は、非 JNDI モードで IVT を実行することにより、実行時に管理対象オブジェクトを作成します。 JNDI ベースのリポジ

トリーは、セットアップが比較的複雑であるため、最初は JNDI を使用しないで IVT を実行することが勧められています。

## JNDI を使用しないポイント・ツー・ポイント検査

IVT には、実行用のスクリプトとして、 UNIX では IVTRun、 Windows NT では IVTRun.bat というファイルが用意されています。このファイルは、インストール・ディレクトリーの bin サブディレクトリーに置かれています。

JNDI を使用せずにテストを行う場合は、次のコマンドを実行します。

```
IVTRun -nojndi
```

クライアント・モードで、JNDI を使用せずにテストを行う場合は、次のコマンドを実行します。

```
IVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
               [-channel <channel>]
```

ここで、それぞれは次のものを表します。

**qmgr** 接続したいキュー・マネージャーの名前

**hostname** キュー・マネージャーが稼働しているホスト

**port** キュー・マネージャーのリスナーが稼働している TCP/IP ポート (デフォルトは 1414)

**channel** クライアントの接続チャネル (デフォルトは SYSTEM.DEF.SVRCONN)

テストが正常に完了すると、次の例のような出力が表示されます。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d312020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR            JMS_IBM_PutApplType:11
JMSXGroupSeq:1                  JMSXDeliveryCount:0
JMS_IBM_MsgType:8               JMSXUserID:kingdon
JMSXApplID:D:¥jdk1.1.8¥bin¥java.exe
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
```

```
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

## JNDI を使用したポイント・ツー・ポイント検査

JNDI を使用して IVT を実行する場合は、LDAP サーバーが稼働していて、 Java オブジェクトを受け入れるように構成されている必要があります。次のメッセージが表示された場合は、LDAP サーバーには接続されているものの、そのサーバーが正しく構成されていないことを示します。

```
オブジェクトをバインドできません
```

このメッセージは、そのサーバーに Java オブジェクトが保管されないか、サフィックスが正しくないことを意味します。 383ページの『LDAP サーバー構成の検査』を参照してください。

加えて、 以下の管理対象オブジェクトを JNDI ネーム・スペースから取り出せなくてはなりません。

- MQQueueConnectionFactory
- MQQueue

これらのオブジェクトを自動的に作成するため、 UNIX では `IVTSetup`、 Windows NT では `IVTSetup.bat` というスクリプトが用意されています。次のコマンドを入力します。

```
IVTSetup
```

スクリプトは、MQ JMS 管理ツール (37ページの『第5章 MQ JMS 管理ツールの使用』を参照してください) を起動し、 JNDI ネーム・スペース内にオブジェクトを作成します。

MQQueueConnectionFactory は、 `ivtQCF` (LDAP では、`cn=ivtQCF`) という名前の下にバインドされます。プロパティーはすべてデフォルトの値になります。

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

MQQueue は、`ivtQ` (`cn=ivtQ`) という名前の下にバインドされます。 `QUEUE` プロパティーの値は `QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)` になります。それ以外のプロパティーはすべてデフォルトの値になります。

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

### ポイント・ツー・ポイント IVT

JNDI ネーム・スペースに管理対象オブジェクトが作成されたなら、次のコマンドを使用して IVTRun (Windows NT では IVTRun.bat) スクリプトを実行します。

```
IVTRun [ -t ] [ -url <"providerURL"> [ -icf <initCtxFact> ] ]
```

ここで、それぞれは次のものを表します。

**-t** トレースをオンにする (デフォルトでは、トレースがオフになっている)

**providerURL** 管理対象オブジェクトの JNDI 位置。デフォルトの初期コンテキスト・ファクトリーが使用されている場合、これは、以下の形式の LDAP URL になります。

```
ldap://hostname.company.com/contextName
```

ファイル・システム・サービス・プロバイダーが使用されている場合 (下記の initCtxFact を参照してください) は、 URL は次のような形式になります。

```
file://directorySpec
```

**注:** *providerURL* ストリングは、引用符 (″) で囲んでください。

**initCtxFact** 初期コンテキスト・ファクトリーのクラス名。デフォルトは、LDAP サービス・プロバイダーのクラス名で、次のような値です。

```
com.sun.jndi.ldap.LdapCtxFactory
```

ファイル・システム・サービス・プロバイダーが使用されている場合は、このパラメーターを次のように設定します。

```
com.sun.jndi.fscontext.RefFSContextFactory
```

テストが正常に完了すると、非 JNDI モードのときと同じような出力が表示されますが、このモードでは、QueueConnectionFactory と Queue の「作成」に関する行に、 JNDI からのオブジェクトの取り出しが示唆されます。次のコード断片は、その例です。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
 ...
 ...
```

必ずそうしなければならないわけではありませんが、 IVTSetup スクリプトによって作成されたオブジェクトを JNDI ネーム・スペースから除去する習慣を付けることが勧められています。 IVTTidy (Windows NT では IVTTidy.bat) というスクリプトは、この目的のために用意されています。

## IVT エラー・リカバリー

テストが正常に完了しなかった場合は、以下の点を考慮すると助けになるかもしれません。

- クラスパスに関係するエラー・メッセージが戻された場合は、25ページの『インストール後のセットアップ』を参照して、クラスパスが正しく設定されているかどうかを調べてください。
- IVT が失敗したときに、「MQQueueManager の作成に失敗しました」というメッセージと、 2059 という番号を含む別のメッセージが戻される場合があります。これは、MQSeries が、 IVT が実行されたマシンのデフォルト・ローカル・キュー・マネージャーに接続できなかったことを意味します。キュー・マネージャーが稼働していること、およびそれがデフォルト・キュー・マネージャーとしてマークされていることを確認してください。
- 「MQ キューのオープンに失敗しました」というメッセージは、 MQSeries が、デフォルト・キュー・マネージャーに接続はしたものの、SYSTEM.DEFAULT.LOCAL.QUEUE を開けなかったことを示します。これは、そのキューが、デフォルト・キュー・マネージャーに存在していないか、 PUT や GET の操作に対して使用可能になっていないことを意味する場合があります。そのキューを追加するか、テストが行われている間使用可能にしてください。

表5 は、 IVT でテストされるクラスと、そのクラスが提供されているパッケージのリストです。

*表 5. IVT でテストされるクラス*

| クラス | Jar ファイル |
|---|---|
| MQSeries JMS クラス | com.ibm.mqjms.jar |
| com.ibm.mq.MQMessage | com.ibm.mq.jar |
| javax.jms.Message | jms.jar |
| javax.naming.InitialContext | jndi.jar |
| javax.resource.cci.Connection | connector.jar |
| javax.transaction.xa.XAException | jta.jar |
| com/sun/jndi/toolkit/ComponentDirContext | providerutil.jar |
| com.sun.jndi.ldap.LdapCtxFactory | ldap.jar |

## パブリッシュ / サブスクライブのインストール検査テスト

パブリッシュ / サブスクライブのインストール検査テスト (PSIVT) プログラムは、すでにコンパイルされた形式でのみ提供されています。これは、`com.ibm.mq.jms` パッケージにあります。

PSIVT は、以下のタスクを試行します。

1. トピック `MQJMS/PSIVT/Information` でパブリッシュする、パブリッシャー `p` の作成
2. トピック `MQJMS/PSIVT/Information` でサブスクライブする、サブスクライバー `s` の作成
3. `p` を使用した、簡単なテキスト・メッセージのパブリッシュ
4. `s` を使用した、入力キューで待機しているメッセージの受信

PSIVT を実行すると、パブリッシャーはメッセージをパブリッシュし、サブスクライバーはメッセージを受信して表示します。パブリッシャーは、ブローカーのデフ

ォルト・ストリームにパブリッシュします。サブスクライバーは、非永続で、メッセージ選択を行わず、ローカル接続からのメッセージを受け入れます。サブスクライバーは同期的な受信を行い、1 つのメッセージに対して最長で 5 秒間の受信待ちをします。

PSIVT は、IVT と同様、JNDI モードとスタンドアロン・モードで実行できます。JNDI モードでは、JNDI を使用して、 JNDI ネーム・スペースから TopicConnectionFactory と Topic を受信します。 JNDI を使用しない場合は、実行時にこれらのオブジェクトが作成されます。

## JNDI を使用しないパブリッシュ / サブスクライブ検査

PSIVT の実行用に、 PSIVTRun (Windows NT では PSIVTRun.bat) という PSIVTRun スクリプトが提供されています。このファイルは、インストール・ディレクトリーの bin サブディレクトリーにあります。

JNDI を使用せずにテストを行う場合は、次のコマンドを実行します。

```
PSIVTRun -nojndi [-m <qmgr>] [-t]
```

クライアント・モードで、JNDI を使用せずにテストを行う場合は、次のコマンドを実行します。

```
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
                [-channel <channel>] [-t]
```

ここで、それぞれは次のものを表します。

**-nojndi** 管理対象オブジェクトの JNDI ルックアップを使用しない

**qmgr** 接続したいキュー・マネージャーの名前

**hostname** キュー・マネージャーが稼働しているホスト

**port** キュー・マネージャーのリスナーが稼働している TCP/IP ポート (デフォルトは 1414)

**channel** クライアントの接続チャネル (デフォルトは SYSTEM.DEF.SVRCONN)

**-t** トレースをオンにする (デフォルトはオフ)

テストが正常に完了すると、次の例のような出力が表示されます。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service
Publish/Subscribe Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding Text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive...

Got message:

JMS Message class: jms_text
  JMSType:         null
```

```
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d515800000000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT completed OK
PSIVT finished
```

## JNDI を使用したパブリッシュ / サブスクライブ検査

PSIVT を JNDI モードで実行する場合は、次の 2 つの管理対象オブジェクトを JNDI ネーム・スペースから取り出せなくてはなりません。

- TopicConnectionFactory は、`ivtTCF` という名前の下にバインドされます。
- Topic は、`ivtT` という名前の下にバインドされます。

これらのオブジェクトは、MQ JMS 管理ツール (37ページの『第5章 MQ JMS 管理ツールの使用』を参照してください) と、以下のコマンドを使用することによって定義できます。

```
DEFINE TCF(ivtTCF)
```

このコマンドは、TopicConnectionFactory を定義します。

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

このコマンドは、Topic を定義します。

これらの定義は、ブローカーがデフォルト・キュー・マネージャーで稼働していて、そのキュー・マネージャーが使用可能であることを前提とします。非デフォルト・キュー・マネージャーを使用する場合のこれらのオブジェクトの構成については、 42ページの『JMS オブジェクトの管理』を参照してください。これらのオブジェクトは、下記に説明されている `-url` コマンド行パラメーターで示されたコンテキストに常駐していなければなりません。

JNDI モードでテストを実行する場合は、次のコマンドを入力します。

```
PSIVTRun -url <purl> [-icf <initcf>] [-t]
```

ここで、それぞれは次のものを表します。

**-t** トレースをオンにする (デフォルトでは、トレースがオフになっている)

**-url <purl>** 管理対象オブジェクトが常駐している JNDI 位置の URL

**-icf <initcf>** JNDI [com.sun.jndi.ldap.LdapCtxFactory] のための initialContextFactory

テストが正常に完了すると、非 JNDI モードのときと同じような出力が表示されますが、このモードでは、QueueConnectionFactory と Queue の「作成」に関する行に、 JNDI からのオブジェクトの取り出しが示唆されます。

## PSIVT エラー・リカバリー

テストが正常に完了しなかった場合は、以下の点を考慮すると助けになるかもしれません。

- 次のようなメッセージが表示される場合があります。

  ```
  *** broker が実行されていません! 'strmqbrk' を使用して開始してください ***
  ```

  これは、ブローカーはターゲット・キュー・マネージャーにインストールされているものの、その制御キューに未解決のメッセージがいくつか含まれていることを意味します。この場合、ブローカーは稼働していないことになります。これを開始させるためには、strmqbrk コマンドを使用します。 (26ページの『パブリッシュ / サブスクライブ・モードのための付加的なセットアップ』を参照してください。)

- 次のようなメッセージが表示される場合もあります。

  ```
  キュー・マネージャーに接続できません: <default>
  ```

  この場合は、ご使用の MQSeries システムにデフォルト・キュー・マネージャーが構成されていることを確認してください。

- 次のようなメッセージが表示される場合もあります。

  ```
  キュー・マネージャーに接続できません: ...
  ```

  この場合は、PSIVT が使用する管理下の TopicConnectionFactory が、有効なキュー・マネージャー名で構成されていることを確認してください。あるいは、-nojndi オプションを使用しているのであれば、有効なキュー・マネージャーが指定されている (-m オプションを使用) ことを確認してください。

- 次のようなメッセージが表示される場合もあります。

  ```
  キュー・マネージャーの broker 制御キューにアクセスできません: ...
  broker がこのキュー・マネージャーにインストールされていることを確認してください
  ```

  この場合は、PSIVT が使用する管理下の TopicConnectionFactory が、ブローカーがインストールされているキュー・マネージャー名で構成されていることを確認してください。 -nojndi オプションを使用している場合は、キュー・マネージャー名が指定されている (-m オプションを使用) ことを確認してください。

# ユーザー独自の MQ JMS プログラムの実行

独自の MQ JMS プログラムの作成については、187ページの『第10章 MQ JMS プログラムの作成』を参照してください。

### MQ JMS プログラムの実行

MQ JMS には、提供されているプログラムやユーザーが作成したプログラムの実行を支援するためのユーティリティー・ファイル、 runjms(Windows NT では runjms.bat) が組み込まれています。

このユーティリティーは、トレースおよびログ・ファイルのためにデフォルトの位置を用意し、アプリケーションに必要なすべてのアプリケーション実行時パラメーターを追加できるようにします。提供されているスクリプトは、環境変数 MQ_JAVA_INSTALL_PATH に MQ JMS のインストール・ディレクトリーが設定されていることを前提としています。また、このスクリプトでは、トレースとログの出力先として、それぞれインストール・ディレクトリー内の trace および log サブディレクトリーを使用することも前提になっています。ただし、これらはあくまでも推奨されている出力先であって、ユーザーが選択した任意のディレクトリーを使用するようにスクリプトを編集することは可能です。

アプリケーションの実行には、次のコマンドを使用します。

```
runjms <classname of application> [application-specific arguments]
```

MQ JMS アプリケーションおよびアプレットの作成については、 185ページの『第3部 MQ JMS を使ったプログラミング』を参照してください。

## 問題の解決

プログラムが正常に完了しない場合は、 25ページの『第4章 MQSeries classes for Java Message Service (MQ JMS) の使用』で説明されているようにインストール検査プログラムを実行し、診断メッセージに示されるアドバイスに従ってください。

### プログラムのトレース

MQ JMS トレース機能は、IBM のスタッフがお客さまの問題を診断するための助けとして用意されています。

トレースは、デフォルトではオフになっています。これは、すぐに出力が大きくなってしまう上に、通常にはほとんど使用されることがないためです。

トレース出力を用意するように依頼された場合は、 Java プロパティー MQJMS_TRACE_LEVEL を次のいずれかの値に設定することによって、これを使用可能にできます。

**on** MQ JMS 呼び出しだけをトレースする

**base** MQ JMS 呼び出しと、その下で行われている MQ ベース Java 呼び出しの両方をトレースする

次に例を示します。

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

トレースをオフにするには、MQJMS_TRACE_LEVEL を **off** にします。

デフォルトでは、トレースは、現在の作業ディレクトリーに mqjms.trc という名前のファイルで出力されます。この出力先のディレクトリーは、 Java プロパティー MQJMS_TRACE_DIR を使用して別のディレクトリーにリダイレクトできます。

次に例を示します。

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

runjms ユーティリティー・スクリプトの場合は、次のように、環境変数 MQJMS_TRACE_LEVEL および MQ_JAVA_INSTALL_PATH を使用してこれらのプロパティーを設定します。

```
java -DMQJMS_LOG_DIR=%MQ_JAVA_INSTALL_PATH%¥log
-DMQJMS_TRACE_DIR=%MQ_JAVA_INSTALL_PATH%¥trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL% %1 %2 %3 %4 %5 %6 %7 %8 %9
```

ただし、これらはあくまでも推奨されているだけであって、必要に応じて変更することができます。

## ロギング

MQ JMS ログ機能は、深刻な問題、特にプログラミング上のエラーというよりも構成上のエラーを意味するような問題を報告するために用意されています。デフォルトでは、ログ出力は System.err ストリームに送られるようになっていて、これは、たいてい JVM が実行されているコンソールの stderr に表示されます。

とはいえ、次のように、新しい位置を指定する Java プロパティーを使用して、出力先をファイルにリダイレクトすることも可能です。

```
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

MQ JMS インストール・ディレクトリーの bin サブディレクトリーにあるユーティリティー・スクリプト runjms は、このプロパティーを次のように設定します。

```
<MQ_JAVA_INSTALL_PATH>/log
```

ここで、MQ_JAVA_INSTALL_PATH は、MQ JMS のインストール・ディレクトリーのパスを表します。ただし、これはあくまでも推奨されているだけであって、必要に応じて変更することができます。

ログの出力先がファイルにリダイレクトされた場合、出力はバイナリー形式になります。それで、ログを表示するためのユーティリティーとして、ファイルをプレーン・テキスト形式に変換する formatLog (Windows NT では formatLog.bat) が提供されています。このユーティリティーは、 MQ JMS のインストール・ディレクトリーの bin サブディレクトリーにあります。変換は、次のようにして実行します。

```
formatLog <inputfile> <outputfile>
```

# 第5章 MQ JMS 管理ツールの使用

管理ツールでは、8 つのタイプの MQ JMS オブジェクトのプロパティーを定義し、それを JNDI ネーム・スペースに保管することができます。 JMS クライアントは、こうして保管された管理対象オブジェクトを JNDI でネーム・スペースから取り出し、使用することができます。

このツールでは、以下の JMS オブジェクトを管理できます。

- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

これらのオブジェクトに関する詳細は、42ページの『JMS オブジェクトの管理』を参照してください。

**注:** JMSWrapXAQueueConnectionFactory と JMSWrapXATopicConnectionFactory は、WebSphere に固有のクラスです。これらのクラスは、パッケージ **com.ibm.ejs.jms.mq** に含まれています。

このツールでは、JNDI 内のディレクトリー・ネーム・スペースのサブコンテキストも操作できます。 41ページの『サブコンテキストの操作』を参照してください。

## 管理ツールの起動

管理ツールは、コマンド行インターフェースを備えています。このインターフェースは、対話的に使用することもできますし、このインターフェースを使用してバッチ・プロセスを開始させることも可能です。対話モードで使用する場合は、管理コマンドを入力するためのコマンド・プロンプトが表示されます。バッチ・モードを使用する場合は、ツールを開始するコマンドの中に、管理コマンド・スクリプトが入っているファイルの名前が組み込まれています。

管理ツールを対話モードで開始する場合は、次のコマンドを入力します。

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

ここで、それぞれは次のものを表します。

| | |
|---|---|
| **-t** | トレースをオンにする (デフォルトはオフ) |
| **-v** | 詳細な出力を生成する (デフォルトは簡潔な出力) |
| **-cfg config_filename** | 代替の構成ファイルの名前 (38ページの『構成』を参照) |

コマンド・プロンプトが表示されます。これは、管理ツールが管理コマンドを受け入れられる状態になったことを示します。このプロンプトは、初め、次のように表示されます。

```
InitCtx>
```

これは、現行コンテキスト (つまり、すべての名前指定およびディレクトリー操作が現在参照している JNDI コンテキスト) が、 PROVIDER_URL 構成パラメーターに定義された初期コンテキストであることを示します (『構成』を参照してください)。

ディレクトリーのネーム・スペースを移動すると、プロンプトにもそれが反映され、プロンプトには常に現行コンテキストが表示されます。

管理ツールをバッチ・モードで開始する場合は、次のコマンドを入力します。

```
JMSAdmin <test.scp
```

ここで、*test.scp* は、管理コマンドが入っているスクリプト・ファイルを表します (40ページの『管理コマンド』を参照してください)。ファイルの最後は、END コマンドでなければなりません。

## 構成

管理ツールは、次の 3 つのパラメーターの値で構成される必要があります。

**INITIAL_CONTEXT_FACTORY**

このパラメーターは、ツールで使用するサービス・プロバイダーを指示します。このプロパティーについては、現在、次の 3 つの値がサポートされています。

- com.sun.jndi.ldap.LdapCtxFactory (LDAP 用)
- com.sun.jndi.fscontext.RefFSContextFactory (ファイル・システム・コンテキスト用)
- com.ibm.ejs.ns.jndi.CNInitialContextFactory (WebSphere の CosNaming リポジトリーでの作業用)

**PROVIDER_URL**

このパラメーターは、セッションの初期コンテキストとする URL、つまり、ツールで実行するすべての JNDI 操作のルートとなる URL を指示します。このプロパティーは、現在、次の 3 つの形式でサポートされています。

- ldap://hostname/contextname (LDAP 用)
- file:[drive:]/pathname (ファイル・システム・コンテキスト用)
- iiop://hostname[:port] /[?TargetContext=ctx] (「基本」WebSphere CosNaming ネーム・スペースへのアクセス用)

**SECURITY_AUTHENTICATION**

このパラメーターは、 JNDI でサービス・プロバイダーにセキュリティー証明書を渡すかどうかを指示します。このパラメーターは、LDAP サービス・プロバイダーが使用されている場合にのみ使用されます。このプロパティーには、現在、次の 3 つの値のいずれかを指定できます。

- none (匿名認証)
- simple (単純認証)

- CRAM-MD5 (CRAM-MD5 認証メカニズム)

  無効な値が指定された場合、プロパティーはデフォルトで none になります。管理ツールでのセキュリティーに関する詳細は、『セキュリティー』を参照してください。

これらのパラメーターは、構成ファイル内に設定されます。ツールを起動する際には、37ページの『管理ツールの起動』で説明されているように、 -cfg コマンド行パラメーターを使用してこの構成を指定できます。構成ファイルの名前が指定されない場合、ツールは、デフォルトの構成ファイル (JMSAdmin.config) のロードを試行します。ツールは、まず現行ディレクトリーでこのファイルを探した後、<MQ_JAVA_INSTALL_PATH>/bin ディレクトリーでファイルを探します。 (なお、この <MQ_JAVA_INSTALL_PATH> は、 MQ JMS インストール・ディレクトリーのパスを表します。)

構成ファイルは 1 つのプレーン・テキスト・ファイルで、 '=' によって区切られた一連のキー / 値のペアから成っています。たとえば、次の例のようになります。

```
#サービス・プロバイダーの設定
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#初期コンテキストの設定
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#認証タイプの設定
    SECURITY_AUTHENTICATION=none
```

(行の先頭にある '#' の文字は、コメント、つまり実際には使用されない行を示します。)

インストール時には、<MQ_JAVA_INSTALL_PATH>/bin ディレクトリーにある、JMSAdmin.config というサンプル構成ファイルが指定されています。システムのセットアップに合うように、このファイルを編集してください。

## WebSphere のための構成

管理ツール (または続けてルックアップを実行する必要のあるすべてのクライアント・アプリケーション) を WebSphere の CosNaming リポジトリーで使用する場合は、以下の構成が必要です。

- CLASSPATH には、以下に示す WebSphere の JNDI 関連 jar ファイルを組み込む必要があります。
  - WebSphere V3.5 の場合:

    ```
    <WSAppserver>¥lib¥ejs.jar
    ```

- WebSphere V3.5 の PATH には、以下が組み込まれている必要があります。

  ```
  <WSAppserver>¥jdk¥jre¥bin
  ```

  ここで、<WSAppserver> は、WebSphere のインストール・パスを表しています。

## セキュリティー

管理者は、38ページの『構成』で説明されている SECURITY_AUTHENTICATION プロパティーの効果を知っている必要があります。

- このパラメーターが *none* に設定されている場合、JNDI は、セキュリティー認証を一切サービス・プロバイダーに渡さず、「匿名認証」が実行されます。

- パラメーターが simple や CRAM-MD5 に設定された場合は、セキュリティー認証が JNDI を通して下のサービス・プロバイダーに渡されます。これらのセキュリティー認証は、ユーザー識別名 (User DN) とパスワードの形式で与えられます。

セキュリティー認証が必要な場合は、ツールが初期化されるときにこれらの入力を求めるプロンプトが出されます。

**注:** 入力されたテキストは画面上に表示されますが、これにはパスワードも含まれています。それで、権限のないユーザーにパスワードを見られないように気を付けてください。

認証は、管理ツールそのものが行っているわけでなく、LDAP サーバーによって代行されています。それで、ディレクトリーの様々な部分に対するアクセス権限のセットアップと保守は、 LDAP サーバーの管理者が担当します。認証が失敗すると、ツールは、適当なエラー・メッセージを表示して終了します。

セキュリティーと JNDI についてのさらに詳しい情報は、 Sun 社の Java Web サイト (http://java.sun.com) を参照してください。

## 管理コマンド

コマンド・プロンプトが表示されていれば、ツールはコマンドを受け入れられる状態にあります。管理コマンドは、通常、次のような形式を取ります。

```
verb [param]*
```

ここで、*verb* の部分には、 表6にリストされているいずれかの管理動詞が入ります。有効なのは、コマンドの先頭に、標準の形式か短縮された形式の動詞が 1 つだけ含まれているコマンドのみです。

動詞に使用されるパラメーターは、その動詞によって異なります。たとえば、END 動詞はパラメーターを取りませんが、DEFINE 動詞は 1 ～ 20 ものパラメーターを取ります。 1 つ以上のパラメーターを取る動詞については、この章の後ろのセクションで詳しく扱います。

*表 6. 管理動詞*

| 動詞 | | 説明 |
|---|---|---|
| 標準 | 短縮形 | |
| ALTER | ALT | 指定された管理対象オブジェクトのプロパティーを少なくとも 1 つ変更する |
| DEFINE | DEF | 管理対象オブジェクトの作成、保管、または新しいサブコンテキストの作成を行う |
| DISPLAY | DIS | 保管されている 1 つ以上の管理対象オブジェクトのプロパティー、または現行コンテキストの内容を表示する |
| DELETE | DEL | 1 つ以上の管理対象オブジェクトをネーム・スペースから除去する、あるいは空のサブコンテキストを除去する |
| CHANGE | CHG | 初期コンテキストの下に属する任意のディレクトリー・ネーム・スペースに移動し、現行コンテキストを変える (セキュリティーの許可は保留) |
| COPY | CP | 保管されている管理対象オブジェクトのコピーを作成し、それを別名で保管する |

*表 6. 管理動詞 (続き)*

| 動詞 | | 説明 |
|---|---|---|
| 標準 | 短縮形 | |
| MOVE | MV | 管理対象オブジェクトの保管場所の名前を変える |
| END | | 管理ツールを閉じる |

動詞の名前の大文字小文字は区別されません。

通常、コマンドの終わりには改行キーを押します。ただしこれは、改行キーを押す前に、行の最後の文字として '+' 記号を入力することでオーバーライドできます。これによって、下の例に示されているように、複数行に渡ってコマンドを入力することが可能になります。

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

*、#、/ といった文字で始まる行は、コメント、つまり無視される行として扱われます。

## サブコンテキストの操作

ディレクトリー・ネーム・スペースのサブコンテキストの操作には、動詞 CHANGE、DEFINE、DISPLAY、および DELETE を使用できます。 表7 では、これらの使い方について説明しています。

*表 7. サブコンテキストの操作に使用されるコマンドの構文と説明*

| コマンド構文 | 説明 |
|---|---|
| DEFINE CTX(ctxName) | 現行コンテキストに ctxName という名前の新しい子サブコンテキストを作成する処理を試行します。セキュリティー違反があった場合、そのサブコンテキストがすでに存在している場合、無効な名前が指定された場合は、コマンドが失敗します。 |
| DISPLAY CTX | 現在のコンテキストの内容を表示します。管理対象オブジェクトには 'a'、サブコンテキストには '[D]' の注記が付きます。加えて、各オブジェクトの Java タイプも表示されます。 |
| DELETE CTX(ctxName) | 現行コンテキストから ctxName という名前の子コンテキストを削除する処理を試行します。そのコンテキストが見つからない場合、コンテキストが空でない場合、セキュリティー違反があった場合は、コマンドが失敗します。 |
| CHANGE CTX(ctxName) | 現行コンテキストを変えて、ctxName という子のコンテキストを参照するようにします。 ctxName には次の 2 つの特殊値があり、このいずれかを指定できます。<br>**=UP** 現在のコンテキストの親に移動する<br>**=INIT** 初期コンテキストに直接移動する<br><br>指定されたコンテキストが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。 |

# JMS オブジェクトの管理

このセクションでは、管理ツールで処理できる 8 つのタイプのオブジェクトについて説明します。また、各オブジェクトに構成できるプロパティーと、それらのプロパティーを操作するための動詞についても説明します。

## オブジェクト・タイプ

表8 では、管理対象オブジェクトの 8 つのタイプを紹介しています。「キーワード」という列は、43ページの表9 にあるコマンドの *TYPE* の部分と置き換えられるストリングを示します。

*表 8. 管理ツールで処理される JMS オブジェクト・タイプ*

| オブジェクト・タイプ | | 説明 |
|---|---|---|
| **Java** | **キーワード** | |
| MQQueueConnectionFactory | QCF | JMS QueueConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、JMS のポイント・ツー・ポイント・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |
| MQTopicConnectionFactory | TCF | JMS TopicConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、JMS のパブリッシュ / サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |
| MQQueue | Q | JMS Queue インターフェースの MQSeries インプリメンテーション。これは、JMS のポイント・ツー・ポイント・ドメインにおけるメッセージの宛先を表します。 |
| MQTopic | T | JMS Topic インターフェースの MQSeries インプリメンテーション。これは、JMS のパブリッシュ / サブスクライブ・ドメインにおけるメッセージの宛先を表します。 |
| MQXAQueueConnectionFactory[1] | XAQCF | JMS XAQueueConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、XA バージョンの JMS クラスを使用する JMS のポイント・ツー・ポイント・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |

*表 8. 管理ツールで処理される JMS オブジェクト・タイプ (続き)*

| オブジェクト・タイプ | | 説明 |
|---|---|---|
| **Java** | **キーワード** | |
| MQXATopicConnectionFactory[1] | XATCF | JMS XATopicConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、XA バージョンの JMS クラスを使用する JMS のパブリッシュ / サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |
| JMSWrapXAQueueConnectionFactory[2] | WSQCF | JMS QueueConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、WebSphere で XA バージョンの JMS クラスを使用する JMS のポイント・ツー・ポイント・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |
| JMSWrapXATopicConnectionFactory[2] | WSTCF | JMS TopicConnectionFactory インターフェースの MQSeries インプリメンテーション。これは、WebSphere で XA バージョンの JMS クラスを使用する JMS のパブリッシュ / サブスクライブ・ドメインに接続を作成するためのファクトリー・オブジェクトを表します。 |

1. これらのクラスは、アプリケーション・サーバーのベンダーが使用するために用意されたものです。アプリケーション・プログラマーにはあまり役に立たないかもしれません。
2. このスタイルの ConnectionFactory は、 JMS セッションを WebSphere によって調整されているグローバル・トランザクションに参加させる場合に使用してください。

## JMS オブジェクトに使用される動詞

ディレクトリー・ネーム・スペース内の管理対象オブジェクトの操作には、動詞 ALTER、DEFINE、DISPLAY、DELETE、 COPY、および MOVE を使用できます。 表9 は、これをまとめたものです。 *TYPE* の部分は、 42ページの表8 にリストされている、必要な管理対象オブジェクトを表すキーワードに置き換えてください。

*表 9. 管理対象オブジェクトの操作に使用されるコマンドの構文と説明*

| コマンド構文 | 説明 |
|---|---|
| ALTER *TYPE*(name) [property]* | 特定の管理対象オブジェクトのプロパティーを、指定された値に更新する処理を試行します。セキュリティー違反があった場合、指定されたオブジェクトが見つからない場合、または指定された新しいプロパティーが無効である場合は、コマンドが失敗します。 |

*表 9. 管理対象オブジェクトの操作に使用されるコマンドの構文と説明 (続き)*

| コマンド構文 | 説明 |
|---|---|
| DEFINE *TYPE*(name) [property]* | 指定されたプロパティーを持つタイプ *TYPE* のオブジェクトを作成し、それを name という名前で現行コンテキストに保管する処理を試行します。セキュリティー違反があった場合、指定された名前が無効であるかすでに存在している場合、または指定されたプロパティーが無効である場合は、コマンドが失敗します。 |
| DISPLAY *TYPE*(name) | name という名前で現行コンテキストにバインドされている、タイプ *TYPE* の管理対象オブジェクトのプロパティーを表示します。オブジェクトが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。 |
| DELETE *TYPE*(name) | name という名前を持つタイプ *TYPE* の管理対象オブジェクトを現行コンテキストから除去する処理を試行します。オブジェクトが存在しない場合やセキュリティー違反があった場合は、コマンドが失敗します。 |
| COPY *TYPE*(nameA) *TYPE*(nameB) | nameA という名前を持つタイプ *TYPE* の管理対象オブジェクトから、nameB 名前のコピーを作成します。これはすべて現行コンテキストの有効範囲内で行われます。コピー元のオブジェクトが存在しない場合、 nameB という名前のオブジェクトがすでに存在する場合、またはセキュリティー違反があった場合は、コマンドが失敗します。 |
| MOVE *TYPE*(nameA) *TYPE*(nameB) | nameA という名前を持つタイプ *TYPE* の管理対象オブジェクトを、 nameB という名前の管理対象オブジェクトに移動 (名前変更) します。これはすべて現行コンテキストの有効範囲内で行われます。移動するオブジェクトが存在しない場合、 nameB という名前のオブジェクトがすでに存在する場合、またはセキュリティー違反があった場合は、コマンドが失敗します。 |

## オブジェクトの作成

オブジェクトは、次のコマンド構文を使用して JNDI ネーム・スペース内に作成および保管されます。

```
DEFINE TYPE(name) [property]*
```

つまり、まず DEFINE 動詞を使い、その後に管理対象オブジェクトの参照 *TYPE*(name) を続けます。また、その後に 1 つ以上の*プロパティー* (45ページの『プロパティー』 を参照) が指定される場合もあります。

### LDAP に名前を付ける際の考慮事項

作成したオブジェクトを LDAP 環境に保管する場合、その名前は、一定の規則に準拠していなければなりません。その 1 つとして、オブジェクトやサブコンテキストの名前には、 cn= (共通名) や ou= (組織単位) といった接頭部を付けなければなりません。

この点、管理ツールでは、接頭部を付けなくてもオブジェクトやコンテキストの名前を参照できるようにすることで、 LDAP サービス・プロバイダーの使用を簡単にしています。ツールは、接頭部が入力されない場合に、自動的に、入力された名前にデフォルトの接頭部 (現在は cn=) を付けます。

たとえば、次の例のようになります。

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

入力されたオブジェクト名 (testQueue) には接頭部が付いていないのに、ツールが LDAP の命名規則に合うように自動的に接頭部を付けていることに注目してください。これと同様に、コマンド DISPLAY Q(testQueue) が出された場合でも、やはりこの接頭部が追加されます。

LDAP サーバーを、Java オブジェクトの保管用に構成する必要があります。この構成に役立つ情報は、383ページの『付録C. Java オブジェクト用の LDAP サーバー構成』で提供されています。

## プロパティー

プロパティーは、次の形式の名前 / 値のペアから成っています。

```
PROPERTY_NAME(property_value)
```

プロパティーの名前は、大文字小文字を区別しませんが、 表10 に示されている識別名に限定されます。この表は、各プロパティーに有効なプロパティー値も示しています。

*表 10. プロパティーの名前と有効値*

| プロパティー | | 有効値 (太字はデフォルト) |
|---|---|---|
| **標準** | **短縮形** | |
| DESCRIPTION | DESC | 任意のストリング |
| TRANSPORT | TRAN | • **BIND** - MQSeries バインディングを使用した接続<br>• CLIENT - クライアント接続 |
| CLIENTID | CID | 任意のストリング |
| QMANAGER | QMGR | 任意のストリング |
| HOSTNAME | HOST | 任意のストリング |
| PORT | | 任意の正整数 |
| CHANNEL | CHAN | 任意のストリング |
| CCSID | CCS | 任意の正整数 |
| RECEXIT | RCX | 任意のストリング |
| RECEXITINIT | RCXI | 任意のストリング |
| SECEXIT | SCX | 任意のストリング |

*表 10. プロパティーの名前と有効値 (続き)*

| プロパティー | | 有効値 (太字はデフォルト) |
|---|---|---|
| 標準 | 短縮形 | |
| SECEXITINIT | SCXI | 任意のストリング |
| SENDEXIT | SDX | 任意のストリング |
| SENDXITINIT | SDXI | 任意のストリング |
| TEMPMODEL | TM | 任意のストリング |
| MSGRETENTION | MRET | • **Yes** - 不要なメッセージを入力キューに残す<br>• No - 不要なメッセージは指定されている後処理オプションに従って扱う |
| BROKERVER | BVER | **V1** - 現在指定できる唯一の値 |
| BROKERPUBQ | BPUB | 任意のストリング (デフォルトは **SYSTEM.BROKER.DEFAULT.STREAM**) |
| BROKERSUBQ | BSUB | 任意のストリング (デフォルトは **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**) |
| BROKERDURSUBQ | BDSUB | 任意のストリング (デフォルトは **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**) |
| BROKERCCSUBQ | CCSUB | 任意のストリング (デフォルトは **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**) |
| BROKERCCDSUBQ | CCDSUB | 任意のストリング (デフォルトは **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**) |
| BROKERQMGR | BQM | 任意のストリング |
| BROKERCONQ | BCON | 任意のストリング |
| EXPIRY | EXP | • **APP** - JMS アプリケーションが有効期限を定義できる<br>• UNLIM - 有効期限を設けない<br>• 有効期限を表す正整数 (ミリ秒) |
| PRIORITY | PRI | • **APP** - JMS アプリケーションが優先順位を定義できる<br>• QDEF - キューのデフォルトの値を優先順位にする<br>• 0〜9 の範囲の任意の整数 |
| PERSISTENCE | PER | • **APP** - JMS アプリケーションが永続性を定義できる<br>• QDEF - キューのデフォルトの値から永続性を決定する<br>• PERS - メッセージに永続性を与える<br>• NON - メッセージに永続性を与えない |
| TARGCLIENT | TC | • **JMS** - JMS アプリケーションをメッセージのターゲットにする<br>• MQ - JMS ではない従来の MQSeries アプリケーションをメッセージのターゲットにする |
| ENCODING | ENC | 49ページの『ENCODING プロパティー』を参照 |
| QUEUE | QU | 任意のストリング |
| TOPIC | TOP | 任意のストリング |

プロパティーの多くは、特定のオブジェクト・タイプのサブセットにのみ関係するものです。それで表11 では、どのプロパティー / オブジェクト・タイプの組み合わせが有効であるかを示し、それぞれのプロパティーについて簡単に説明します。

*表 11. プロパティーとオブジェクト・タイプの有効な組み合わせ*

| プロパティー | 有効なオブジェクト・タイプ | | | | | | 説明 |
|---|---|---|---|---|---|---|---|
| | QCF | TCF | Q | T | WSQCF XAQCF | WSTCF XATCF | |
| DESCRIPTION | Y | Y | Y | Y | Y | Y | 保管オブジェクトの説明 |
| TRANSPORT | Y | Y | | | Y[1] | Y[1] | 接続に MQ バインディングを使用するか、クライアント接続を使用するか |
| CLIENTID | Y | Y | | | Y | Y | クライアントのストリング ID |
| QMANAGER | Y | Y | Y | | Y | Y | 接続するキュー・マネージャーの名前 |
| PORT | Y | Y | | | | | キュー・マネージャーが聴取を行うポート |
| HOSTNAME | Y | Y | | | | | キュー・マネージャーが常駐するホストの名前 |
| CHANNEL | Y | Y | | | | | 使用されているクライアント接続チャネルの名前 |
| CCSID | Y | Y | Y | Y | | | 接続に使用されるコード化文字セット ID |
| RECEXIT | Y | Y | | | | | 使用されている受信出口の完全修飾クラス名 |
| RECEXITINIT | Y | Y | | | | | 受信出口初期化ストリング |
| SECEXIT | Y | Y | | | | | 使用されているセキュリティー出口の完全修飾クラス名 |
| SECEXITINIT | Y | Y | | | | | セキュリティー出口の初期化ストリング |
| SENDEXIT | Y | Y | | | | | 使用されている送信出口の完全修飾クラス名 |
| SENDEXITINIT | Y | Y | | | | | 送信出口の初期化ストリング |
| TEMPMODEL | Y | | | | Y | | 一時キューの作成に使用するモデル・キューの名前 |
| MSGRETENTION | Y | | | | Y | | 入力キューにある不要なメッセージを接続のコンシューマーに保持させるかどうか |
| BROKERVER | | Y | | | | Y | 使用されているブローカーのバージョン |
| BROKERPUBQ | | Y | | | | Y | ブローカー入力キュー (ストリーム・キュー) の名前 |
| BROKERSUBQ | | Y | | | | Y | 非永続サブスクライブ・メッセージを取り出すキューの名前 |
| BROKERDURSUBQ | | | | Y | | | 永続サブスクライブ・メッセージを取り出すキューの名前 |

*表 11. プロパティーとオブジェクト・タイプの有効な組み合わせ (続き)*

| プロパティー | 有効なオブジェクト・タイプ | | | | | | 説明 |
|---|---|---|---|---|---|---|---|
| | QCF | TCF | Q | T | WSQCF XAQCF | WSTCF XATCF | |
| BROKERCCSUBQ | | Y | | | | Y | ConnectionConsumer のために非永続サブスクライブ・メッセージを取り出すキューの名前 |
| BROKERCCDSUBQ | | | | Y | | | ConnectionConsumer のために永続サブスクライブ・メッセージを取り出すキューの名前 |
| BROKERQMGR | | Y | | | | Y | ブローカーが稼働しているキュー・マネージャー |
| BROKERCONQ | | Y | | | | Y | ブローカーの制御キュー名 |
| EXPIRY | | | Y | Y | | | 宛先に送られたメッセージの有効期間 |
| PRIORITY | | | Y | Y | | | 宛先に送信されたメッセージの優先順位 |
| PERSISTENCE | | | Y | Y | | | 宛先に送信されたメッセージの永続性 |
| TARGCLIENT | | | Y | Y | | | ターゲット・アプリケーションでの情報の変換に MQSeries RFH2 形式を使用するかどうかを示すフィールド |
| ENCODING | | | Y | Y | | | この宛先に使用されるエンコード・スキーム |
| QUEUE | | | Y | | | | この宛先を表すキューの基礎名 |
| TOPIC | | | | Y | | | この宛先を表すトピックの基礎名 |

**注:**

1. WSTCF、WSQCF、XATCF、および XAQCF オブジェクトでは、 BIND トランスポート・タイプしか使用できません。
2. ツールで設定されたプロパティーとプログラム式のプロパティーとの関係は、379ページの『付録A. 管理ツールのプロパティーとプログラマブル・プロパティーの間のマッピング』で説明しています。
3. TARGCLIENT プロパティーは、ターゲット・アプリケーションでの情報の変換に MQSeries RFH2 形式を使用するかどうかを示します。

   MQJMS_CLIENT_JMS_COMPLIANT 定数は、情報の送信に RFH2 形式を使用することを示します。 RFH2 形式は、MQ JMS を使用するアプリケーションで認識されます。それで、MQJMS_CLIENT_JMS_COMPLIANT 定数は、ターゲット MQ JMS アプリケーションと情報を交換する場合に設定してください。

   MQJMS_CLIENT_NONJMS_MQ 定数は、情報の送信に RFH2 形式を使用しないことを示します。一般的に、この値は、既存の MQSeries アプリケーション (つまり、RFH2 を処理しないアプリケーション) に使用されます。

## プロパティーの依存関係

一部のプロパティーは、相互に依存関係を持っています。そのため、あるプロパティーを指定しても、それと依存関係にあるプロパティーに特定の値が設定されなけ

れば、その指定が意味を持たない場合があります。このような依存関係は、クライアント・プロパティーと出口初期化ストリングという、 2 つの特定のプロパティー・グループで見られます。

**クライアント・プロパティー**

TRANSPORT(CLIENT) プロパティーが接続ファクトリーに明示的に設定されていないと、ファクトリーが提供する接続で使用されるトランスポートは MQ バインディングになります。そのため、この接続ファクトリーのクライアント・プロパティーはまったく構成できません。クライアント・プロパティーには、次のプロパティーが含まれます。

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

TRANSPORT プロパティーが CLIENT に設定されていない状態でこれらのプロパティーのいずれかを設定しようとすると、エラーになります。

**出口初期化ストリング**

該当する出口名が指定されていない場合、出口初期化ストリングの設定はすべて無効になります。出口初期化プロパティーには、以下のプロパティーが含まれます。

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

たとえば、RECEXIT(some.exit.classname) を指定せずに RECEXITINIT(myString) を指定すると、エラーになります。

## ENCODING プロパティー

ENCODING プロパティーが取ることのできる有効値は、その他のプロパティーよりも複雑です。エンコード・プロパティーは、3 つのサブプロパティーから構成されます。

| | |
|---|---|
| **整数エンコード** | これには、NORMAL と REVERSED があります。 |
| **10 進数エンコード** | これには、NORMAL と REVERSED があります。 |
| **浮動小数点エンコード** | これには、IEEE_NORMAL、IEEE_REVERSED、および S390 (System/390®) があります。 |

ENCODING は、次の構文で、3 文字のストリングとして表されます。

```
{N|R}{N|R}{N|R|3}
```

このストリングで、それぞれは以下を表しています。

- N は NORMAL
- R は REVERSED
- 3 は S390 (System/390)
- 最初の文字は*整数エンコード*
- 2 番目の文字は *10 進数エンコード*
- 3 番目の文字は*浮動小数点エンコード*

これらの文字の組み合わせで、ENCODING プロパティーには 12 の値を設定できます。

これらに加えて、NATIVE というストリングもあります。この値を指定すると、Java プラットフォームに合ったエンコード値が設定されます。

有効な ENCODING の組み合わせの例を以下に示します。

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

## エラー状態の例

このセクションでは、オブジェクトの作成時に発生する可能性のあるエラー状態の例を示します。

### プロパティーが不明

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
不明なプロパティー: PIZZA
```

### オブジェクトのプロパティーが無効

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
QCF の無効なプロパティー: PRI
```

### プロパティー値のタイプが無効

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
CCS プロパティーの無効な値: English
```

### プロパティー値が有効範囲外

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
PRI プロパティーの無効な値: 12
```

### プロパティーの不調和 - クライアント / バインディング

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
このコンテキストに無効なプロパティーがあります: クライアント・バインド属性
が壊れています
```

### プロパティーの不調和 - 出口の初期化

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
有効なオブジェクトを作成できません。指定したパラメーターを確かめてください
このコンテキストに無効なプロパティーがあります: ExitInit が Exit ストリング
なしで与えられました
```

# 第2部 MQ ベース Java を使ったプログラミング

# 第6章 プログラマー向けの概要

この章には、プログラマー向けの情報が記載されています。プログラムの作成方法について詳しくは、57ページの『第7章 MQ ベース Java プログラムの作成』を参照してください。

## Java インターフェースを使用する理由

MQSeries classes for Java プログラミング・インターフェースにより、 MQSeries アプリケーションの開発者は、 Java の利点を多く利用できるようになります。

- Java プログラミング言語は**使いやすい**言語です。

  ヘッダー・ファイル、ポインター、構造体、共用体、および演算子を多重定義する必要がありません。 Java で書かれたプログラムは、C および C++ で書かれた同等のプログラムより開発とデバッグが容易です。

- Java は**オブジェクト指向**です。

  Java のオブジェクト指向機能は C++ に匹敵しますが、多重継承はありません。その代りに、Java ではインターフェースの概念が使用されます。

- Java は本質的に**分散型**です。

  Java クラス・ライブラリーには、 HTTP および FTP などの TCP/IP プロトコルを用いて処理するためのルーチンのライブラリーが含まれています。 Java プログラムはファイル・システムへのアクセスと同様に容易に URL にアクセスできます。

- Java は**堅固な**言語です。

  Java では、考えられる問題の早期検査、動的 (実行時) 検査、およびエラーが発生しやすい状態の除去に重点が置かれています。 Java では、メモリーの上書きとデータの破壊の可能性がなくなる参照の概念が使用されます。

- Java は**安全な**言語です。

  Java はネットワーク環境や分散環境で実行されることを意図していて、セキュリティーに多くの重点が置かれています。 Java プログラムは自分の実行時スタックをオーバーランしたり、自分のプロセス・スペース外のメモリーを壊したりすることはありません。また、Java プログラムは、インターネットからダウンロードされる際に、ローカル・ファイルの読み取りや書き込みを行うことはありません。

- Java は**移植可能**です。

  Java 仕様には「インプリメンテーションに依存している」面は 1 つもありません。 Java コンパイラーは、特定のアーキテクチャーに偏らないオブジェクト・ファイル形式を生成します。コンパイル済みのコードは、Java 実行時システムが存在している限り、多くのプロセッサーで実行可能です。

MQSeries classes for Java を使用してアプリケーションを作成すると、ユーザーはプログラムの Java バイト・コード (アプレット と呼ばれる) をインターネットからダウンロードします。そして、ユーザーはこれらのアプレットを自分のマシンで実

行できます。これは、Web サーバーへのアクセス権のあるユーザーは、前もって自分のマシンにインストールしなくてもアプリケーションをロードし実行できるという意味です。

プログラムへの更新が必要な場合には、Web サーバー上のコピーを更新します。すると、すべてのユーザーが、次にアプレットにアクセスした時に自動的に最新バージョンを受信します。これで、膨大な数のデスクトップが関係している従来のクライアント・アプリケーションのインストールと更新に関するコストを著しく削減できます。

企業のファイアウォールの外側からアクセス可能な Web サーバー上にアプレットを入れておくと、インターネット上の誰もがそのアプリケーションをダウンロードし使用することができます。これは、インターネット上の任意の場所からメッセージを読み取って、使用している MQSeries システムに取り込むことができるということを示しています。このことは、インターネット・アクセス可能サービス、サポート、および電子商取引アプリケーションを 1 つにまとめて構築するという目的への第一歩となります。

## MQSeries classes for Java インターフェース

手続型 MQSeries アプリケーション・プログラミング・インターフェースは、以下の動詞によって構築されています。

```
MQBACK、MQBEGIN、MQCLOSE、MQCMIT、MQCONN、MQCONNX、
MQDISC、MQGET、MQINQ、MQOPEN、MQPUT、MQPUT1、MQSET
```

これらの動詞はすべて、操作対象の MQSeries オブジェクトへのハンドルをパラメーターとして取ります。 Java はオブジェクト指向なので、 Java プログラミング・インターフェースではこのハンドルがよく使われます。ユーザーのプログラムは、以下の例のように、一連の MQSeries オブジェクトで構成されています。これらのオブジェクトは、メソッドを呼び出すことによって操作します。

手続き型のインターフェースを使用する場合は、 MQDISC (Hconn, CompCode, Reason) の呼び出しを用いてキュー・マネージャーから切断します。 *Hconn* はキュー・マネージャーに対するハンドルの 1 つです。

Java インターフェースでは、キュー・マネージャーはクラス MQQueueManager のオブジェクトで表されます。そして、そのキュー・マネージャーからの切断は、そのクラスで disconnect() メソッドを呼び出すことによって行われます。

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

## Java 開発キット

自分で作成したアプレットまたはアプリケーションをコンパイルする前に、使用している開発プラットフォーム用の Java 開発キット (JDK) にアクセスしなければなりません。 JDK には、 MQSeries classes for Java クラスが依存している標準の Java クラス、変数、コンストラクター、およびインターフェースすべてが含まれているほか、サポートされる各プラットフォームでアプレットおよびプログラムをコンパイルし実行するために必要なツールも含まれています。

JDK は、以下の WWW アドレスにある IBM Software Download Catalog から入手できます。

```
http://www.ibm.com/software/download
```

また、アプリケーションは、IBM Visual Age for Java の統合開発環境に組み込まれている JDK でも開発できます。

AS/400 プラットフォームで Java アプリケーションをコンパイルする場合は、まず、以下の製品をインストールする必要があります。

- AS/400 Developer Kit for Java (5769-JV1)
- OS/400 Qshell Interpreter (5769-SS1) オプション 30

## MQSeries classes for Java クラス・ライブラリー

MQSeries classes for Java は、 Java アプレットとアプリケーションが MQSeries と対話できるようにする、一連の Java クラスです。

以下のクラスが提供されます。

- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

また、次の Java インターフェースも提供されます。

- MQC
- MQPoolServicesEventListener
- MQReceiveExit

- MQSecurityExit
- MQSendExit

加えて、次の Java インターフェースのインプリメンテーションも提供されます。ただし、これらのインターフェースは、アプリケーションから直接使用することを意図したものではありません。

- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

Java において、パッケージ とは、関連したクラスのセットをまとめてグループ化するためのメカニズムのことをいいます。 MQSeries クラスとインターフェースは、com.ibm.mq と呼ばれる Java パッケージとして出荷されます。 MQSeries classes for Java パッケージを自分のプログラムに組み込むには、次の行をソース・ファイルの先頭に追加します。

```
import com.ibm.mq.*;
```

# 第7章 MQ ベース Java プログラムの作成

MQSeries classes for Java を使用して MQSeries キューにアクセスする場合には、MQSeries キューでのメッセージ書き込みとメッセージ取得の呼び出しが含まれた、Java プログラムを作成します。このプログラムには、Java アプレット、 Java サーブレット、あるいは Java アプリケーション といった形態のものがあります。

この章では、MQSeries システムと対話する Java アプレット、サーブレット、およびアプリケーションの作成に役立つ情報を扱います。個々のクラスについての詳細は、91ページの『第9章 MQ ベース Java クラスおよびインターフェース』を参照してください。

## アプレットを作成するかアプリケーションを作成するか

アプレット、サーブレット、またはアプリケーションのどれを作成するかは、使用する接続とプログラムをどこから実行するかによって異なります。

アプレットとアプリケーションには、主に次のような違いがあります。

- アプレットはアプレット・ビューアーを用いて実行されるか、あるいは Webブラウザー内で実行され、サーブレットは Web アプリケーション・サーバー内で実行され、アプリケーションはスタンドアロンで実行されます。
- アプレットは、Web サーバーから Web ブラウザー・マシンにダウンロードできますが、アプリケーションとサーブレットはダウンロードできません。

以下の一般規則が適用されます。

- MQSeries classes for Java がローカルにインストールされていないマシンからユーザー・プログラムを実行したい場合には、アプレットを作成する必要があります。
- MQSeries classes for Java 固有のバインディング・モードはアプレットをサポートしていません。したがって、プログラムをすべての接続モード (固有のバインディング・モードを含む) で使用したい場合には、サーブレットかアプリケーションを作成しなければなりません。

## 接続の違い

MQSeries classes for Java 用にプログラムする方法には、使用したい接続モードと何らかの依存関係があります。

### クライアント接続

MQSeries classes for Java がクライアントとして使用される場合は、 MQSeries C クライアントと類似していますが、次の違いがあります。

- TCP/IP のみをサポートします。
- 接続テーブルをサポートしません。
- 始動時にどの MQSeries 環境変数も読み取りません。

- チャネル定義および環境変数に格納される情報は、MQEnvironment と呼ばれるクラスに格納されます。あるいはこの情報は、接続の際にパラメーターとして渡すことができます。
- エラー条件と例外条件を MQException クラスに指定されたログに書き込みます。デフォルトのエラー宛先は Java コンソールです。

MQSeries classes for Java クライアントは MQBEGIN 動詞および高速バインドをサポートしません。

MQSeries クライアントの一般的な情報については、*MQSeries クライアント* を参照してください。

**注:** VisiBroker 接続を使用している場合には、 MQEnvironment に設定されているユーザー ID およびパスワードは MQSeries サーバーに転送されません。有効なユーザー ID は IIOP サーバーに適用されるユーザー ID です。

## バインディング・モード

MQSeries classes for Java のバインディング・モードは、次の点でクライアント・モードとは異なります。

- MQEnvironment クラスによって提供されるパラメーターの大部分が無視されます。
- バインディングでは、MQBEGIN 動詞および MQSeries キュー・マネージャーへの高速バインディングがサポートされます。

**注:** MQSeries for AS/400 では、キュー・マネージャーによって調整されたグローバル作業単位を開始するための MQBEGIN の使用はサポートされていません。

## 使用する接続の定義

接続は、MQEnvironment クラスでの変数の設定によって判別されます。

**MQEnvironment.properties**

これには、次のキー / 値のペアを入れることができます。

- クライアント接続およびバインド接続の場合。

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES
```

- VisiBroker 接続の場合。

```
MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_VISIBROKER
MQC.ORB_PROPERTY, orb
```

**MQEnvironment.hostname**

この変数の値を次のように設定します。

- クライアント接続の場合、接続先の MQSeries サーバーのホスト名にします。
- バインディング・モードの場合、ヌルにします。

# サンプル・コード

このセクションでは、60ページの図1 と 63ページの図2 で 2 つのサンプル・コードを扱います。それぞれのサンプル・コードは、特定の接続を使用するために作成されるものであり、代替接続を使用するのに必要な変更について説明する注釈が付記されています。

## サンプル・アプレット・コード

以下のサンプル・コードは、TCP/IP 接続を使用して次のことを実行するアプレットを示しています。

1. キュー・マネージャーへの接続
2. SYSTEM.DEFAULT.LOCAL.QUEUE へのメッセージの書き込み
3. メッセージの再取得

## サンプル・コード

```
// ============================================================================
//
// Licensed Materials - Property of IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995,1999
//
// ============================================================================
// MQSeries Client for Java サンプル・アプレット
//
// このサンプルは、アプレット・ビューアーと HTML ファイルを使用するアプレットとして、
// 次のコマンドで実行されます。-
//            appletviewer MQSample.html
// 出力は、コマンド行に表示され、アプレット・ビューアーのウィンドウには表示されません。
//
// 注: MQSeries と TCP/IP が間違いなく正しくセットアップされているのに
// MQSeries エラー 2 理由 2059 が戻される場合は、「アプレット・ビューアー (Applet viewer)」
// ウィンドウのプロパティーの選択で「アプレット (Applet)」をクリックし、
// 「ネットワーク・アクセス (Network access)」を無制限に変更します。
import com.ibm.mq.*;            // MQSeries classes for Java パッケージをインクルードします

public class MQSample extends java.applet.Applet
{

  private String hostname = "your_hostname";      // 接続するホストの
                                                  // 名前を定義します
  private String channel  = "server_channel";     // 使用するクライアントのチ
                                                  // ャネルの名前を定義します
                                                  // 注: MQSeries サーバーがデフォ
                                                  // ルトの TCP/IP ポート 1414
                                                  // で聴取を行うと想定します
  private String qManager = "your_Q_manager";     // 接続するキュー・マネー
                                                  // ジャー・オブジェクトの
                                                  // 名前を定義します

  private MQQueueManager qMgr;                    // キュー・マネージャー・
                                                  // オブジェクトを定義します
  // クラスが呼び出されると、最初に初期化が実行されます。

  public void init()
  {
     // MQSeries 環境をセットアップします
     MQEnvironment.hostname = hostname;           // ホスト名とチャネルのスト
                                                  // リングは直接ここに書き込
     MQEnvironment.channel  = channel;            // むこともできたのですが!

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY, // TCP/IP またはサーバー
                                  MQC.TRANSPORT_MQSERIES);// 接続を設定します

  } // init の終わり
```

*図 1. MQSeries classes for Java サンプル・アプレット (1/3)*

```
public void start()
{

  try {
    // キュー・マネージャーへの接続を作成します
    qMgr = new MQQueueManager(qManager);

    // オープンしたいキューにオプションをセットアップします...
    // 注: Java では、すべての MQSeries オプションに MQC の接頭部を付けます。
     int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                       MQC.MQOO_OUTPUT ;
    // ここで、オープンしたいキューとオープンのオプションを指定します...

    MQQueue system_default_local_queue =
            qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                             openOptions);

    // 簡潔な MQSeries メッセージを定義し、UTF 形式で何らかのテキストを書き込みます..

    MQMessage hello_world = new MQMessage();
    hello_world.writeUTF("Hello World!");

    // メッセージ・オプションを指定します...

    MQPutMessageOptions pmo = new MQPutMessageOptions();  // デフォルトを受け入れます
                                                          // MQPMO_DEFAULT
                                                          // 定数と同じ結果
                                                          // になります
    // キューにメッセージを書き込みます

    system_default_local_queue.put(hello_world,pmo);

    // メッセージを再取得します...
    // まず、メッセージを受け取る MQSeries メッセージ・バッファーを定義します..

    MQMessage retrievedMessage = new MQMessage();
    retrievedMessage.messageId = hello_world.messageId;
    // メッセージ取得のオプションを設定します..

    MQGetMessageOptions gmo = new MQGetMessageOptions();  // デフォルトを受け入れます
                                                          // MQGMO_DEFAULT と同じ
                                                          // 結果になります
    // キューからメッセージを取得します..

    system_default_local_queue.get(retrievedMessage, gmo);
    // UTF メッセージ・テキストを表示してメッセージの取得を確認します

    String msgText = retrievedMessage.readUTF();
    System.out.println("The message is: " + msgText);

    // キューをクローズします

    system_default_local_queue.close();

    // キュー・マネージャーへの接続を切断します

    qMgr.disconnect();

  }
  // 上記のタスクでエラーが発生した場合は、何がうまくいかなかったのかを
  // 識別するように努めてください。それは MQSeries エラーでしたか?
```

*図 1. MQSeries classes for Java サンプル・アプレット (2/3)*

## サンプル・コード

```
    catch (MQException ex)
    {
      System.out.println("An MQSeries error occurred : Completion code " +
                         ex.completionCode +
                         " Reason code " + ex.reasonCode);
    }
    // Java バッファー・スペースのエラーでしたか?
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the
      message buffer: " + ex);
    }

  } // start の終わり

} // サンプルの終わり
```

*図 1. MQSeries classes for Java サンプル・アプレット (3/3)*

### VisiBroker for Java を使用するための接続の変更

次の行

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);
```

を次の行

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_VISIBROKER);
```

に変更し、さらに、次の行を追加して ORB (オブジェクト要求ブローカー) を初期化します。

```
ORB orb=ORB.init(this,null);
MQEnvironment.properties.put(MQC.ORB_PROPERTY,orb);
```

また、次の import ステートメントをファイルの先頭に追加することも必要です。

```
import org.omg.CORBA.ORB;
```

VisiBroker を使用している場合には、ポート番号またはチャネルの指定は不要です。

## サンプル・アプリケーション・コード

以下のサンプル・コードは、バインディング・モードを使用して次のことを実行する単純なアプリケーションを示しています。

1. キュー・マネージャーへの接続
2. SYSTEM.DEFAULT.LOCAL.QUEUE へのメッセージの書き込み
3. メッセージの再取得

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 1999
// ============================================================================
// MQSeries classes for Java サンプル・アプリケーション
//
// このサンプルは、Java アプリケーションとして、次のコマンドで実行されます:- java MQSample

import com.ibm.mq.*;            // MQSeries classes for Java パッケージをインクルードします

public class MQSample
{
  private String qManager = "your_Q_manager";  // 接続するキュー・マネー
                                               // ジャーを定義します
  private MQQueueManager qMgr;                 // キュー・マネージャー・
                                               // オブジェクトを定義します
  public static void main(String args[]) {
     new MQSample();
  }
  public MQSample() {
   try {

      // キュー・マネージャーへの接続を作成します
      qMgr = new MQQueueManager(qManager);

      // オープンしたいキューにオプションをセットアップします...
      //注: Java では、すべての MQSeries オプションに MQC の接頭部を付けます。

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;
      // ここで、オープンしたいキューとオープンの
      // オプションを指定します...
      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);

      // 簡潔な MQSeries メッセージを定義し、UTF 形式で何らかのテキストを書き込みます..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");
      // メッセージ・オプションを指定します...

      MQPutMessageOptions pmo = new MQPutMessageOptions(); // デフォルトを受け入れます
                                                           // MQPMO_DEFAULT と同じ結果になります
```

*図 2. MQSeries classes for Java サンプル・アプリケーション (1/2)*

## サンプル・コード

```
     // キューにメッセージを書き込みます
      system_default_local_queue.put(hello_world,pmo);

      // メッセージを再取得します...
      // まず、メッセージを受け取る MQSeries メッセージ・バッファーを定義します..

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // メッセージ取得のオプションを設定します...

      MQGetMessageOptions gmo = new MQGetMessageOptions(); // デフォルトを受け入れます
                                                           // MQGMO_DEFAULT と同じ結果になります
      // キューからメッセージを取得します...

      system_default_local_queue.get(retrievedMessage, gmo);

      // UTF メッセージ・テキストを表示してメッセージの取得を確認します

      String msgText = retrievedMessage.readUTF();
      System.out.println("The message is: " + msgText);
      // キューをクローズします...
      system_default_local_queue.close();
      // キュー・マネージャーへの接続を切断します

      qMgr.disconnect();
    }
      // 上記のタスクでエラーが発生した場合は、何がうまくいかなかったのかを
      // 識別するように努めてください。それは MQSeries エラーでしたか?
    catch (MQException ex)
    {
      System.out.println("An MQSeries error occurred : Completion code " +
                         ex.completionCode + " Reason code " + ex.reasonCode);
    }
      // Java バッファー・スペースのエラーでしたか?
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the message buffer: " + ex);
    }
  }
} // サンプルの終わり
```

*図 2. MQSeries classes for Java サンプル・アプリケーション (2/2)*

# キュー・マネージャーに対する操作

このセクションでは、MQSeries classes for Java を使用してキュー・マネージャーに接続する方法と、そのキュー・マネージャーから接続を切断する方法について説明します。

## MQSeries 環境のセットアップ

**注:** このステップは MQSeries classes for Java をバインディング・モードで使用する場合は不要です。その場合には、『キュー・マネージャーへの接続』 へ進んでください。クライアント接続を使用してキュー・マネージャーに接続する前に、 MQEnvironment をセットアップしてください。

「C」ベースの MQSeries クライアントは、環境変数に従って MQCONN 呼び出しの動作を制御します。 Java アプレットには環境変数へのアクセス権がないため、Java プログラミング・インターフェースにはクラス MQEnvironment が組み込まれています。このクラスにより、接続の試行時に使用する以下の詳細を指定することができます。

- チャネル名
- ホスト名
- ポート番号
- ユーザー ID
- パスワード

チャネル名およびホスト名を指定するには、次のコードを使用します。

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

これは、MQSERVER 環境変数を次のように設定することと同じです。

```
"java.client.channel/TCP/host.domain.com".
```

デフォルトでは、Java クライアントはポート 1414 で MQSeries リスナーに接続しようとします。別のポートを指定するには、次のコードを使用します。

```
MQEnvironment.port = nnnn;
```

ユーザー ID およびパスワードは、デフォルトではブランクになっています。ブランクでないのユーザー ID とパスワードを指定するには、次のコードを使用します。

```
MQEnvironment.userID   = "uid";  // env var MQ_USER_ID と同等のもの
MQEnvironment.password = "pwd";  // env var MQ_PASSWORD と同等のもの
```

**注:** VisiBroker for Java を使用して接続をセットアップしている場合は、62ページの『VisiBroker for Java を使用するための接続の変更』を参照してください。

## キュー・マネージャーへの接続

ここまでで、次のように MQQueueManager クラスの新規インスタンスを作成することによって、キュー・マネージャーに接続できる状態になっています。

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

### キュー・マネージャーの操作

キュー・マネージャーから切断するには、次のようにキュー・マネージャーで disconnect() メソッドを呼び出します。

```
queueManager.disconnect();
```

disconnect メソッドを呼び出すと、そのキュー・マネージャーからアクセスしたオープン・キューとプロセスはすべてクローズされます。しかし、使用を終えた時点でこれらのリソースを明示的にクローズするプログラミングの習慣を付けておくとよいでしょう。これは、close() メソッドを使って行います。

キュー・マネージャーの commit() メソッドと backout() メソッドは、手続型インターフェースで使用される MQCMIT 呼び出しと MQBACK 呼び出しに代わるものです。

## キューおよびプロセスへのアクセス

キューとプロセスへのアクセスには、MQQueueManager クラスを使用します。MQOD (オブジェクト記述子構造体) は、これらのメソッドのパラメーターに縮小されます。たとえば、キュー・マネージャー ″queueManager″ のキューをオープンするには、以下のコードを使用します。

```
MQQueue queue = queueManager.accessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

*options* パラメーターは、MQOPEN 呼び出しの Options パラメーターと同じものです。

accessQueue メソッドは、クラス MQQueue の新規オブジェクトを戻します。

キューの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
queue.close();
```

MQSeries classes for Java では、MQQueue コンストラクターを使用してキューを作成することもできます。各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、 accessQueue メソッドの場合と全く同じものです。たとえば、以下のとおりです。

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

この方法でキュー・オブジェクトを構成すると、MQQueue のユーザー固有のサブクラスを作成することもできます。

プロセスにアクセスするには、accessQueue の代りに accessProcess を使用します。*動的キュー名パラメーター* はプロセスに適用されないので、このメソッドには*動的キュー名パラメーター* がありません。

accessProcess メソッドは、クラス MQProcess の新規オブジェクトを戻します。

プロセス・オブジェクトの使用が完了したら、次の例のように close() メソッドを使用してそのキューをクローズしてください。

```
process.close();
```

MQSeries classes for Java では、MQProcess コンストラクターを使用してプロセスを作成することもできます。各パラメーターは、キュー・マネージャー・パラメーターが追加されている以外は、 accessProcess メソッドの場合と全く同じものです。この方法でプロセス・オブジェクトを構成すると、 MQProcess のユーザー固有のサブクラスを作成することもできます。

## メッセージの処理

メッセージは、MQQueue クラスの put() メソッドでキューに書き込まれ、MQQueue クラスの get() メソッドでキューから読み取られます。 MQPUT および MQGET でバイトの配列の書き込みと読み取りを行う手続型インターフェースとは異なり、 Java プログラム言語では MQMessage クラスのインスタンスの書き込みと読み取りが行われます。 MQMessage クラスは、実際のメッセージ・データが入っているデータ・バッファーを、そのメッセージを記述しているすべての MQMD (メッセージ記述子) パラメーターとともにカプセル化します。

新規メッセージを作成するには、MQMessage クラスの新規インスタンスを作成し、writeXXXメソッドを使用してデータをメッセージ・バッファーに書き込みます。

新規メッセージ・インスタンスが作成されると、*MQSeries アプリケーション・プログラミング・リファレンス* 説明されているように、 MQMD パラメーターがすべて自動的にそのデフォルト値に設定されます。また、MQQueue の put() メソッドは、パラメーターとして MQPutMessageOptions クラスのインスタンスを取ります。このクラスは MQPMO 構造体を表します。次の例では、メッセージを作成し、それをキューに書き込みます。

```
// 自分の年齢と自分の名前を含む新しいメッセージを作成します
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Wendy Ling";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// デフォルトのメッセージ書き込みのオプションを使用します...
MQPutMessageOptions pmo = new MQPutMessageOptions();

//メッセージを書き込みます
queue.put(myMessage,pmo);
```

MQQueue の get() メソッドは、MQMessage の新規インスタンスを戻します。これはキューから取られたばかりのメッセージを表します。また、パラメーターとして MQGetMessageOptions クラスのインスタンスを取ります。このクラスは MQGMO 構造体を表します。

get() メソッドは自動的にその内部バッファーのサイズを着信メッセージが収まるように調整するので、最大メッセージ・サイズの指定は不要です。戻されたメッセージ中のデータにアクセスするには、MQMessage クラスの readXXX メソッドを使用します。

### メッセージの処理

以下の例は、キューからメッセージを読み取る方法を示しています。

```
// キューからメッセージを取得します
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // デフォルトの値を使用します

// メッセージのデータを抽出します
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

read メソッドと write メソッドが使用する数値形式は、 *encoding* メンバー変数を設定することで変更できます。

ストリングの読み取りと書き込みに使用する文字セットは、 *characterSet* メンバー変数を設定することで変更できます。

詳細は 116ページの『MQMessage』を参照してください。

**注:** MQMessage の writeUTF() メソッドでは、含まれている Unicode バイト数に加えてストリングの長さも自動的にエンコードされます。メッセージが別の Java プログラムによって (readUTF() を使用して) 読み取られる場合には、これがストリング情報を送信する最も簡単な方法です。

## エラーの処理

Java インターフェースのメソッドは完了コードと理由コードを戻しません。その代わりに、MQSeries の呼び出しによる完了コードと理由コードが両方ともゼロでない場合は必ず例外を投げます。これで、MQSeries への各呼び出し後に戻りコードを調べる必要がないように、プログラム・ロジックが単純化されます。プログラム内のどのポイントで障害の可能性を処理したいかを決定できます。これらのポイントでは、次の例のように、コードを「try」と「catch」のブロックで囲むことができます。

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
// このコードのブロックは、指定された 2 つの put
// メソッドのうちのいずれかがゼロ以外の完了コード
// か理由コードを戻した場合にのみ実行されます。
System.out.println("An error occurred during the put operation:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
}
```

## 属性値の取得と設定

大半の共通属性では、MQManagedObject、MQQueue、MQProcess、および MQQueueManager クラスに getXXX() メソッドと setXXX() メソッドが含まれており、これらのメソッドによって、その属性値を取得および設定することができます。ただし、MQQueue の場合、これらのメソッドは、キューのオープン時に適切な「inquire」フラグと「set」フラグを指定した場合しか機能しません。

一部の共通属性の場合、MQQueueManager、MQQueue、および MQProcess クラスはすべて、 MQManagedObject と呼ばれるクラスから継承します。このクラスは inquire() および set() のインターフェースを定義します。

*new* 演算子を使用して新規キュー・マネージャー・オブジェクトを作成すると、そのオブジェクトは自動的に「inquiry」用にオープンされます。また、accessProcess() メソッドを使用してプロセス・オブジェクトにアクセスすると、そのオブジェクトは自動的に「inquiry」用にオープンされます。しかし、accessQueue() メソッドを使用してキュー・オブジェクトにアクセスした場合は、そのオブジェクトは自動的には「inquire」または「set」操作用にオープンされません。これは、これらのオプションを自動的に追加すると、一部のタイプのリモート・キューで問題を生じさせる可能性があるためです。キューに対して inquire、set、getXXX、および setXXX メソッドを使用するには、適切な「inquire」フラグと「set」フラグを accessQueue() メソッドの openOptions パラメーターに指定しなければなりません。

inquire メソッドと set メソッドは、次の 3 つのパラメーターを取ります。

- selectors 配列
- intAttrs 配列
- charAttrs 配列

Java では、配列の長さが常に認識されているので、 MQINQ にある SelectorCount、 IntAttrCount、および CharAttrLength パラメーターは不要です。以下の例は、キューに対して照会を行う方法を示しています。

```
// キューに対して照会を実行します
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

## マルチスレッド・プログラム

Java では、マルチスレッド・プログラムは避け難いものです。キュー・マネージャーに接続し、始動時にキューをオープンする単純なプログラムの場合を考えてみます。このプログラムは画面にボタンを 1 つだけ表示します。そのボタンが押されると、プログラムはキューからメッセージを取り出します。

Java 実行時環境は本質的にマルチスレッド環境です。したがって、ユーザー・アプリケーションの初期化はある 1 つのスレッドで実行され、ボタンが押された応答で実行されるコードは別のスレッド (ユーザー・インターフェース・スレッド) で実行されます。

″C″ ベースの MQSeries クライアントでは、ハンドルは複数のスレッド間で共用できないので、この処理は問題となります。 MQSeries classes for Java では、この制

約が緩和されているため、キュー・マネージャー・オブジェクト (さらに、それと関連したキュー・オブジェクトおよびプロセス・オブジェクト) が複数のスレッド間で共用できます。

MQSeries classes for Java を導入すると、指定の接続 (MQQueueManager オブジェクト・インスタンス) の場合、宛先 MQSeries キュー・マネージャーへの全アクセスが確実に同期するようになります。そのため、キュー・マネージャーに呼び出しを発行したいスレッドは、その接続で進行中の他の呼び出しがすべて完了するまでブロックされます。 ユーザー・プログラム内の複数のスレッドから同じキュー・マネージャーに同時にアクセスする必要がある場合は、同時アクセスを必要としているスレッドごとに新規キュー・マネージャー・オブジェクトを作成してください。(これは、スレッドごとに個別の MQCONN 呼び出しを発行することと同等です。)

**注:** CICS Transaction Server for OS/390 環境では、メイン (最初) のスレッドしか CICS または MQSeries 呼び出しを発行できません。これは、この環境では、スレッド間で MQQueueManager オブジェクトや MQQueue オブジェクトを共用したり、子スレッドに新しい MQQueueManager を作成したりすることができないからです。

# ユーザー出口の作成

MQSeries classes for Java では、ユーザー独自の送信出口、受信出口、およびセキュリティー出口を提供することができます。

出口をインプリメントするには、適切なインターフェースをインプリメントする新規 Java クラスを定義します。 MQSeries パッケージには、次の 3 つの出口インターフェースが定義されています。

- MQSendExit
- MQReceiveExit
- MQSecurityExit

以下の例は、3 つの出口インターフェースすべてをインプリメントするクラスを定義しています。

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // このメソッドは、送信出口にあります
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // 送信出口の本体をここに入れます
  }


  // このメソッドは、受信出口にあります
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
     // 受信出口の本体をここに入れます
  }


  // このメソッドはセキュリティー出口にあります
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
     // セキュリティー出口の本体をここに入れます
  }

}
```

出口ごとに、MQChannelExit および MQChannelDefinition オブジェクト・インスタンスが渡されます。これらのオブジェクトは、手続型インターフェースに定義された MQCXP 構造体および MQCD 構造体を表します。

送信出口の場合、*agentBuffer* パラメーターには、いままさに送信されようとしているデータが入ります。そして、受信出口やセキュリティー出口の場合は、受信されたばかりのデータが *agentBuffer* パラメーターに入れられます。配列の長さは式 agentBuffer.length で指示されるため、長さパラメーターは必要ありません。

送信出口およびセキュリティー出口の場合、出口コードは、サーバーに送信されるバイト配列を戻します。受信出口の場合、出口コードは、MQSeries classes for Java に解釈させる変更済みデータを戻します。

次は、考えられる最も単純な出口の本体です。

```
{
  return agentBuffer;
}
```

ダウンロードされた Java アプレットとしてプログラムを実行する場合は、適用されるセキュリティーの制限に従い、どのローカル・ファイルの読み取りや書き込みも行えません。出口に構成ファイルが必要な場合は、そのファイルを Web 上に設定し、 java.net.URL クラスを使用してそのファイルをダウンロードしてから、内容を調べることができます。

## 接続プーリング

MQSeries classes for Java バージョン 5.2 では、 MQSeries キュー・マネージャーへの複数接続を扱うアプリケーションのサポートが追加されました。そのため、ある接続が必要でなくなったときに、それを破棄しないでプールに入れておき、後で再利用することが可能になります。これは、任意のキュー・マネージャーに連続して接続するアプリケーションやミドルウェアのパフォーマンスをかなり向上させます。

MQSeries には、デフォルトの接続プールがあります。アプリケーションは、MQEnvironment クラスでトークンを登録したり登録から外したりすることで、この接続プールをアクティブにしたり非アクティブにしたりすることができます。プールがアクティブになっていると、 MQ ベース Java が MQQueueManager オブジェクトを作成するときにこのデフォルトのプールが調べられ、適当な接続があればそれが再利用されます。そして、MQQueueManager.disconnect() が呼び出されると、その下にある接続はプールに戻されます。

別の方法として、特定の用途のためにアプリケーションで MQSimpleConnectionManager 接続プールを構成することもできます。こうして作られたプールは、MQQueueManager オブジェクトの構成の際に指定することもできますし、デフォルト接続プールとして使用するために MQEnvironment に渡すこともできます。

加えて、MQ ベース Java は、Java 2 Platform Enterprise Edition (J2EE) Connector Architecture の部分的なインプリメンテーションも備えています。 JAAS 1.0 (Java Authentication and Authorization Service) がインストールされた Java 2 v1.3 JVM で稼働するアプリケーションでは、**javax.resource.spi.ConnectionManager** をインプリメントすることによって独自の接続プールが使用できます。また、このインターフェースを MQQueueManager コンストラクターで指定したり、デフォルト接続プールとして指定することも可能です。

### デフォルト接続プールの制御

次のサンプル・アプリケーション MQApp1 について考慮します。

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (qmgr で何らかの操作を実行します)
```

```
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 は、コマンド行からローカル・キュー・マネージャーのリストを入手し、リスト内の各キュー・マネージャーに順番に接続して、何らかの操作を実行します。しかし、コマンド行に同じキュー・マネージャーが何回もリストされているような場合は、接続の確立を 1 回だけにして、その接続を何回も再利用した方がずっと効率的です。

MQ ベース Java のデフォルト接続プールは、これを行うために用意されているのです。このプールを使用可能にする場合は、いずれかの MQEnvironment.addConnectionPoolToken() メソッドを使用します。プールを使用不可にする場合は、MQEnvironment.removeConnectionPoolToken() を使用します。

次のサンプル・アプリケーション MQApp2 は、機能的には MQApp1 と同じですが、それぞれのキュー・マネージャーに 1 度ずつしか接続しません。

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (qmgr で何らかの操作を実行します)
            :
            qmgr.disconnect();
         }
         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

このアプリケーションでは、 1 つ目の太字になっている行で MQPoolToken オブジェクトを MQEnvironment に登録することにより、デフォルト接続プールが使用可能にされています。

それで、MQQueueManager コンストラクターは、このプールに適当な接続がないかどうかを調べ、該当するキュー・マネージャーへの接続が存在していない場合にだけ接続を作成することになります。使用された接続は、再利用された後、qmgr.disconnect() 呼び出しでプールに戻されます。これらの API 呼び出しは、サンプル・アプリケーション MQApp1 と同じです。

2 つ目の強調表示されている行では、デフォルト接続プールが非アクティブにされています。これにより、そのプールに保管されているキュー・マネージャー接続はすべて破棄されます。このように接続を破棄しないと、プール内のいくつかのキュー・マネージャー接続が使用されたままの状態でアプリケーションが終了されてしまうため、この処理は重要です。接続が破棄されないままアプリケーションを終了すると、キュー・マネージャー・ログに記録されるようなエラーを引き起こす恐れがあります。

デフォルト接続プールでは、使用されていない接続を最大で 10 まで保管でき、使用されていない接続を最大で 5 分、アクティブに保つことができます。この制限は、アプリケーションで変更できます (詳細は 76ページの『別の接続プールの使用』を参照してください)。

MQEnvironment を使用して MQPoolToken を用意する代わりに、次のように、アプリケーションで独自のものを構成することもできます。

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

アプリケーションやミドルウェアのベンダーの中には、カスタム接続プールに情報を渡すために MQPoolToken のサブクラスを用意しているところもあります。この方法で構成を行い、addConnectionPoolToken() に渡せば、補助的な情報もその接続プールに渡すことができます。

## デフォルト接続プールと複数のコンポーネント

MQEnvironment は、登録された MQPoolToken オブジェクトの静的な集合を保持します。この集合での MQPoolTokens の追加と除去には、次のメソッドを使用します。

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

アプリケーションは、独立して存在し、キュー・マネージャーを使用して作業を行ういくつものコンポーネントによって構成されている場合があります。そのようなアプリケーションでは、それぞれのコンポーネントが、その存続期間の間、MQEnvironment の集合に MQPoolToken を追加する必要があります。

たとえば、サンプル・アプリケーション MQApp3 では、10 のスレッドを作成し、それぞれを開始します。各スレッドはそれぞれの MQPoolToken に登録し、いくらかの時間待機して、それからキュー・マネージャーに接続します。接続が切断されると、スレッドはそれぞれの MQPoolToken を除去します。

デフォルト接続プールは、MQPoolTokens の集合に 1 つでもトークンが残っている限り、つまりこのアプリケーションの存続期間の間ずっとアクティブになっています。アプリケーションは、これらのスレッド全体の制御において、マスター・オブジェクトを保持する必要はありません。

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}
class MQApp3_Thread extends Thread
{
      long time;
      public MQApp3_Thread(long time)
      {
         this.time=time;
      }
```

```
                public synchronized void run()
                {
                   MQPoolToken token=MQEnvironment.addConnectionPoolToken();
                   try {
                      wait(time);
                      MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
                      :
                      : (qmgr で何らかの操作を実行します)
                      :
                      qmgr.disconnect();
                   }
                   catch (MQException mqe) {System.err.println("Error occurred!");}
                   catch (InterruptedException ie) {}
                   MQEnvironment.removeConnectionPoolToken(token);
                }
             }
```

## 別の接続プールの使用

このセクションでは、クラス **com.ibm.mq.MQSimpleConnectionManager** を使用して別の接続プールを用意する方法を説明します。このクラスには、接続プーリングのための基本的な機能が備わっており、アプリケーションはこのクラスを使用してプールの動作をカスタマイズすることができます。

MQSimpleConnectionManager は、インスタンス化されると、 MQQueueManager コンストラクターで指定できるようになります。すると、MQSimpleConnectionManager は、構成された MQQueueManager の下にある接続を管理するようになります。MQSimpleConnectionManager に適切な接続がプーリングされていれば、その接続は再利用され、 MQQueueManager.disconnect() 呼び出しが実行された後に MQSimpleConnectionManager に戻されます。

次のコード・フラグメントは、その動作を実例で示しています。

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (qmgr で何らかの操作を実行します)
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (qmgr2 で何らかの操作を実行します)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

最初の MQQueueManager コンストラクターで作られた接続は、 qmgr.disconnect() 呼び出しの後 myConnMan に保管されます。この接続は、次に MQQueueManager コンストラクターが呼び出されたときに再利用されます。

2 番目の行では MQSimpleConnectionManager が使用可能にされます。そして最後の行では MQSimpleConnectionManager が使用不可にされ、そのプールに保持されていた接続がすべて破棄されます。なお、MQSimpleConnectionManager はデフォルトで MODE_AUTO になっていますが、これについてはこのセクションの後の方で説明します。

MQSimpleConnectionManager は、一番最近に使用された接続から順に割り振り、使用されてから最も時間がたっている接続から順に破棄していきます。デフォルトでは、その接続が 5 分間使用されなかった場合と、プール内の使用されていない接続の数が 10 を超えた場合に、接続が破棄されます。これらの値はいずれも、以下を使用して変更できます。

- MQSimpleConnectionManager.setTimeout()
- MQSimpleConnectionManager.setHighThreshold()

加えて、MQQueueManager コンストラクターに Connection Manager が指定されなかった場合に使用するため、デフォルト接続プールとして使用する MQSimpleConnectionManager をセットアップすることもできます。

次のアプリケーションは、これを実例で示したものです。

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String[] args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

MQSimpleConnectionManager は、太字の部分の行でセットアップされます。ここでは、次のように設定されています。

- 1 時間使用されなかった接続を破棄する。
- プールに保持できる、使用されていない接続の数の限界は 50 とする。
- MODE_AUTO (実際のデフォルト) を使用する。これを使用すると、これがデフォルト接続マネージャーであり、かつ MQEnvironment によって保持される MQPoolTokens の集合に少なくとも 1 つ以上のトークンが含まれている場合にのみ、このプールがアクティブになります。

新しい MQSimpleConnectionManager は、デフォルト接続マネージャーとして設定されます。

最後の行で、アプリケーションは MQApp3.main() を呼び出しています。これにより、それぞれが独立して MQSeries を使用する、いくつかのスレッドが実行されます。これらのスレッドは、今後接続を作成する際には myConnMan を使用します。

## 独自の ConnectionManager の使用

JAAS 1.0 がインストールされている Java 2 v1.3 では、アプリケーションやミドルウェアの提供者が、代替の接続プールのインプリメンテーションを作成できます。MQ ベース Java は、J2EE Connector Architecture の部分的なインプリメンテーションを備えています。 **javax.resource.spi.ConnectionManager** のインプリメンテーションは、デフォルトの Connection Manager として使用することもできますし、MQQueueManager コンストラクターで指定することもできます。

MQ ベース Java は、 J2EE Connector Architecture の Connection Management コントラクトを使用してコンパイルを実行します。それで、このセクションをお読みになる際には、 Sun 社の Web サイト `http://java.sun.com` にある、 J2EE Connector Architecture の Connection Management コントラクトに関する資料も併せてご覧ください。

ConnectionManager インターフェースでは、次のメソッドだけが定義されています。

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

MQQueueManager コンストラクターは、適当な ConnectionManager で allocateConnection を呼び出します。必要な接続を記述するパラメーターとして、ManagedConnectionFactory と ConnectionRequestInfo のインプリメンテーションが渡されます。

ConnectionManager は、同じ ManagedConnectionFactory および ConnectionRequestInfo オブジェクトで作成された javax.resource.spi.ManagedConnection オブジェクトがないかプールを調べます。適当な ManagedConnection オブジェクトが見つかった場合、 ConnectionManager はその候補の ManagedConnection を含む java.util.Set を作成します。その後、ConnectionManager は以下を呼び出します。

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

ManagedConnectionFactory の MQSeries インプリメンテーションは、 subject パラメーターを無視します。このメソッドは、集合から適当な ManagedConnection を選択して戻します。そして、適当な ManagedConnection が見つからない場合はヌルを戻します。プール内に適当な ManagedConnection がない場合、 ConnectionManager は以下のメソッドを使用してこれを作成できます。

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

この場合もやはり、subject パラメーターは無視されます。このメソッドは、MQSeries キュー・マネージャーに接続し、新しく作られた接続を表す javax.resource.spi.ManagedConnection のインプリメンテーションを戻します。ConnectionManager は、ManagedConnection を取得する (プールから、または新しく作成することによって) と、次のメソッドを使用して接続ハンドルを作成します。

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

この接続ハンドルは、allocateConnection() を呼び出すことによって戻されます。

ConnectionManager は、次のメソッドを使用して ManagedConnection にインタレストを登録します。

```
mc.addConnectionEventListener()
```

接続で重大エラーが発生した場合や、MQQueueManager.disconnect() が呼び出されたときには、 ConnectionEventListener にそのことが通知されます。MQQueueManager.disconnect() が呼び出された場合、 ConnectionEventListener は次のいずれかの処理を行います。

- mc.cleanup() 呼び出しを使用して ManagedConnection をリセットし、ManagedConnection をプールに戻す
- mc.destroy() 呼び出しを使用して ManagedConnection を破棄する

デフォルトの ConnectionManager にする ConnectionManager では、インタレストを MQEnvironment によって管理される MQPoolTokens の集合の状態に登録することもできます。これを行うためには、次のように、まず MQPoolServices オブジェクトを構成し、次いで MQPoolServices オブジェクトに MQPoolServicesEventListener オブジェクトを登録します。

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

MQPoolToken が集合に加えられたり集合から除去された場合や、デフォルトの ConnectionManager が変更された場合には、そのことがリスナーに通知されます。MQPoolServices オブジェクトは、MQPoolTokens の集合の現在のサイズを照会するためにも使用できます。

## MQ ベース Java プログラムのコンパイルとテスト

MQ ベース Java プログラムをコンパイルする前に、9ページの『第2章 インストール手順』で説明したように、 MQSeries classes for Java インストール・ディレクトリーが CLASSPATH 環境変数に入っていることを確認してください。

クラス ″MyClass.java″ をコンパイルするには、次のコマンドを使用します。

```
javac MyClass.java
```

### MQ ベース Java アプレットの実行

アプレット (java.applet.Applet のサブクラス) を作成する場合は、その実行前に、ユーザーのクラスを参照する HTML ファイルを作成しなければなりません。サンプル HTML ファイルは次のようになります。

```
<html>
<body>
<applet code="MyClass.class" width=200 height=400>
</applet>
</body>
</html>
```

この HTML ファイルを Java が使用可能な Web ブラウザーにロードするか、 Java 開発キット (JDK) のアプレット・ビューアーを使用して、このアプレットを実行します。

アプレット・ビューアーを使用するには、次のコマンドを入力します。

```
appletviewer myclass.html
```

### MQ ベース Java アプリケーションの実行

クライアント・モードまたはバインディング・モードのいずれかを使用してアプリケーション (main() メソッドが含まれているクラス) を作成する場合は、 Java インタープリターでプログラムを実行してください。次のコマンドを使用します。

```
java MyClass
```

**注:** 「.class」拡張子はクラス名から省略されています。

### CICS Transaction Server for OS/390 での MQ ベース Java アプリケーションの実行

CICS 下のトランザクションとして Java アプリケーションを実行する場合には、以下のようにしてください。

1. 提供されている CEDA トランザクションを使用して、アプリケーションとトランザクションを CICS に定義する。

2. CICS システムに MQSeries CICS アダプターがインストールされていることを確認する。 (詳細については、 *MQSeries (OS/390 版) システム管理の手引き* を参照してください。)
3. CICS 始動 JCL (ジョブ制御言語) の DHFJVM パラメーターで指定した JVM 環境に、適切な CLASSPATH および LIBPATH エントリーが組み込まれていることを確認する。
4. 何らかの通常のプロセスを使用してトランザクションを開始する。

CICS Java トランザクションの実行に関する詳細は、お手持ちの CICS システムの資料を参照してください。

## MQ ベース Java プログラムのトレース

MQ ベース Java にはトレース機能が含まれています。コードに問題がありそうな場合には、この機能を使って診断メッセージを生成することができます。 (通常、この機能を使用する必要があるのは、 IBM サービスから要請された場合だけです。)

トレースは、MQEnvironment クラスの enableTracing メソッドと disableTracingメソッドによって制御されます。たとえば、以下のとおりです。

```
MQEnvironment.enableTracing(2);    // レベル 2 でトレースします
 ...                               // トレースするコマンドをここに指定します
MQEnvironment.disableTracing();    // トレースを再度オフにします
```

トレースは Java コンソール (System.err) に書き込まれます。

ユーザー・プログラムがアプリケーションである場合や、 appletviewer コマンドを使用してローカル・ディスクからプログラムを実行する場合は、トレース出力をユーザー選択のファイルにリダイレクトすることもできます。以下のサンプル・コードは、 myapp.trc というファイルにトレース出力をリダイレクトする方法の例を示しています。

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
  // ファイルがオープンできなかったため、
  // 代わりに System.err にトレースします
  MQEnvironment.enableTracing(2);
}
```

次の 5 つの異なるトレースのレベルがあります。

1. 入り口、出口、および例外をトレースする
2. 1 の他にパラメーター情報もトレースする
3. 2 の他に、送信および受信した MQSeries ヘッダーとデータ・ブロックもトレースする
4. 3 の他に、送信および受信したユーザー・メッセージ・データもトレースする
5. 4 の他に、Java 仮想マシン内のメソッドもトレースする

## MQ ベース Java プログラムのトレース

Java 仮想マシン内のメソッドをレベル 5 でトレースする場合は、次のようにします。

- アプリケーションの場合は、コマンド `java_g` (`java` の代わり) を実行してこれを実行する
- アプレットの場合は、コマンド `appletviewer_g` (`appletviewer` の代わり) を使用して実行する

**注:**

1. `java_g` は、OS/390 上の High Performance Java (HPJ) アプリケーションではサポートされていません。
2. `java_g` は、 OS/400 ではサポートされていませんが、これに似た機能は、RUNJVA コマンドに `OPTION(*VERBOSE)` を使用することによって得られます。

## MQ ベース Java プログラムのトレース

# 第8章 環境による動作の違い

この章では、Java クラスが使用できる各種環境における Java クラスの動作について説明します。 MQSeries classes for Java クラスにより、以下の環境で使用できるアプリケーションを作成することができます。

1. UNIX または Windows プラットフォーム上の MQSeries V2.x サーバーに接続された MQSeries Client for Java
2. UNIX または Windows プラットフォーム上の MQSeries V5 サーバーに接続された MQSeries Client for Java
3. UNIX または Windows プラットフォーム上の MQSeries V5 サーバーで実行中の MQSeries Java バインディング
4. MQSeries for MVS/ESA™ サーバーで実行中の MQSeries Java バインディング
5. CICS Transaction Server for OS/390 バージョン 1.3 が稼働する MQSeries for MVS/ESA サーバーで実行中の MQSeries Java バインディング

どの場合でも、MQSeries classes for Java コードは、土台となる MQSeries サーバーが提供するサービスを使用します。これには、機能のレベルによる違いがあります (たとえば、MQSeries V5 は V2 の機能のスーパーセットを提供します)。また、一部の API 呼び出しやオプションの動作にも違いがあります。こうした動作の違いの多くは小さなものであり、しかもほとんどは、 OS/390 (MQSeries for MVS/ESA) サーバーとその他のプラットフォームで使用されるサーバーの間の違いです。

MQSeries classes for Java は、すべての環境で一貫した機能と動作を提供する、クラスの「コア」を提供します。そしてそれに加えて、 2 および 3 の環境専用に設計された、「V5 拡張機能」も提供しています。続くセクションでは、この「コア」と「拡張機能」について説明します。

## コアの詳細

MQSeries classes for Java には、以下に示すクラスのコアが含まれており、 84ページの『コア・クラスの制限とバリエーション』にリストされたわずかなバリエーションのみを用いてすべての環境で使用することができます。

- MQEnvironment
- MQException
- MQGetMessageOptions

  以下は除外します。
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  以下は除外します。
  - inquire()
  - set()

- MQMessage

  以下は除外します。
  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  以下は除外します。
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  以下は除外します。
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

**注:**

1. 一部の定数はコアには含まれていないので (詳細については、『コア・クラスの制限とバリエーション』を参照)、完全にポータブルなプログラム内ではそれらの定数を使用しないでください。
2. プラットフォームによっては、一部の接続モードがサポートされていない場合があります。このようなプラットフォームでは、サポートされているモードに関連するコア・クラスとオプションだけが使用できます。 (5ページの表1 を参照してください。)

## コア・クラスの制限とバリエーション

一般に、コア・クラスはすべての環境で一貫して動作しますが、 85ページの表12 に記載されている若干の制限とバリエーションがあります。

ここに記載されているバリエーションとは別に、通常、同等の MQSeries クラスに環境の違いがある場合でも、コア・クラスはすべての環境で一貫した動作をします。一般に、この動作は、環境 2 および 3 の場合と同じようになります。

*表 12. コア・クラスの制限とバリエーション*

| クラスまたはエレメント | 制限とバリエーション |
|---|---|
| MQGMO_LOCK<br>MQGMO_UNLOCK<br>MQGMO_BROWSE_MSG_UNDER_CURSOR | 4 または 5 の環境で使用すると、 MQRC_OPTIONS_ERROR の原因になります。 |
| MQPMO_NEW_MSG_ID<br>MQPMO_NEW_CORREL_ID<br>MQPMO_LOGICAL_ORDER | 2 および 3 の環境以外で使用するとエラーになります。 (V5 の拡張機能に関する箇所を参照してください。) |
| MQGMO_LOGICAL_ORDER<br>MQGMO_COMPLETE_MESSAGE<br>MQGMO_ALL_MSGS_AVAILABLE<br>MQGMO_ALL_SEGMENTS_AVAILABLE | 2 および 3 の環境以外で使用するとエラーになります。 (V5 の拡張機能に関する箇所を参照してください。) |
| MQGMO_SYNCPOINT_IF_PERSISTENT | 1 の環境で使用するとエラーになります。 (V5 の拡張機能に関する箇所を参照してください。) |
| MQGMO_MARK_SKIP_BACKOUT | 4 および 5 の環境で使用すると、 MQRC_OPTIONS_ERROR の原因になります。 |
| MQCNO_FASTPATH_BINDING | 3 の環境でのみサポートされています。 (V5 の拡張機能に関する箇所を参照してください。) |
| MQPMRF_* フィールド | 2 および 3 の環境でのみサポートされています。 |
| MQQueue.priority > MaxPriority の場合のメッセージの書き込み | 4 および 5 の環境では、 MQCC_FAILED と MQRC_PRIORITY_ERROR により拒否されます。<br><br>その他の環境では、警告 MQCC_WARNING および MQRC_PRIORITY_EXCEEDS_MAXIMUM 付きで受け入れられ、MaxPriority を指定して書き込んだかのようにメッセージが処理されます。 |
| BackoutCount | 4 および 5 の環境では、メッセージが 255 回を超えてバックアウトされていても、最大バックアウト・カウント 255 が戻されます。 |
| デフォルト動的キュー名 | 4 および 5 の環境では、 CSQ.* という名前になります。その他のシステムでは、AMQ.* という名前になります。 |
| 以下の MQMessage.report オプション。<br>MQRO_EXCEPTION_WITH_FULL_DATA<br>MQRO_EXPIRATION_WITH_FULL_DATA<br>MQRO_COA_WITH_FULL_DATA<br>MQRO_COD_WITH_FULL_DATA<br>MQRO_DISCARD_MSG | すべての環境で設定できますが、レポート・メッセージが OS/390 キュー・マネージャーによって生成されている場合はサポートされません。 OS/390 キュー・マネージャーは Java アプリケーションにアクセスできない可能性があるため、この問題はすべての Java 環境に影響を与えます。 OS/390 キュー・マネージャーが関連している可能性がある場合でも、これらのオプションをあてにしないでください。 |
| MQQueueManager.commit() および MQQueueManager.backout() | 5 の環境でこれらのメソッドを使用すると、 MQRC_ENVIRONMENT_ERROR が戻されます。この環境では、アプリケーションは JCICS タスク同期化メソッド、com.ibm.cics.server.Task.commit() および com.ibm.cics.server.Task.rollback() を使用する必要があります。 |

## 制限

*表 12. コア・クラスの制限とバリエーション (続き)*

| クラスまたはエレメント | 制限とバリエーション |
| --- | --- |
| MQQueueManager コンストラクター | 4 および 5 の環境では、 MQEnvironment (およびオプショナル・プロパティーの引き数) にあるオプションがクライアント接続を暗黙指定すると、コンストラクターが MQRC_ENVIRONMENT_ERROR で失敗します。<br><br>4 および 5 の環境では、コンストラクターから MQRC_CHAR_CONVERSION_ERROR が戻される場合もあります。 OS/390 言語環境プログラム® の National Language Resources コンポーネントがインストールされていることを確認してください。特に、IBM-1047 コード・ページと ISO8859-1 コード・ページとの間で変換が可能であることを確認してください。<br><br>4 および 5 の環境では、コンストラクターから MQRC_UCS2_CONVERSION_ERROR が戻される場合もあります。 MQSeries classes for Java は、Unicode からキュー・マネージャーのコード・ページへの変換を試行し、特定のコード・ページが使用できない場合は、デフォルトで IBM-500 を使用します。 OS/390 C/C++ オプショナル機能の一部としてインストールされる、 Unicode に合った変換テーブルがあること、および言語環境プログラムがそのテーブルの位置を認識していることを確認してください。 UCS-2 変換の使用可能化に関する詳細は、*OS/390 C/C++ プログラミング・ガイド* (SC88-7720) を参照してください。 |

# 他の環境で操作されるバージョン 5 拡張機能

MQSeries classes for Java には、MQSeries V5 に導入された API 拡張機能を使用するために特別に設計された以下の機能が含まれています。これらの機能は、2 および 3 の環境でのみ設計通りに動作します。このトピックでは、これらの機能が他の環境でどのように動作するかを説明します。

### MQQueueManager コンストラクター・オプション

MQQueueManager コンストラクターには、オプションの整数引き数が組み込まれています。これは MQI の MQCNO.options フィールドにマップされ、標準接続と高速パス接続を切り換えるために使用されます。コンストラクターのこの拡張形式は、使用されたオプションが MQCNO_STANDARD_BINDING または MQCNO_FASTPATH_BINDING の場合のみ、すべての環境で受け入れられます。その他のオプションはすべて、コンストラクターが MQRC_OPTIONS_ERROR によって失敗する原因になります。高速パス・オプション MQC.MQCNO_FASTPATH_BINDING は、 MQSeries V5 バインド (環境 3) で使用される場合のみ受け入れられます。その他の環境で使用される場合には受け入れられません。

### MQQueueManager.begin() メソッド

このメソッドは、3 の環境でのみ使用できます。これ以外の環境で使用すると、MQRC_ENVIRONMENT_ERROR で失敗します。 MQSeries for AS/400 では、キュー・マネージャーによって調整されたグローバル作業単位を開始するための `begin()` メソッドの使用はサポートされていません。

### MQPutMessageOptions オプション

次のフラグは、すべての環境で MQPutMessageOptions オプション・フィールドに設定できます。ただし、2 または 3 以外の環境では、後続に MQQueue.put() を使用すると put() は MQRC_OPTIONS_ERROR により失敗します。

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

### MQGetMessageOptions オプション

次のフラグは、すべての環境で MQGetMessageOptions オプション・フィールドに設定できます。ただし、2 または 3 以外の環境では、後に MQQueue.get() を使用すると、get() は MQRC_OPTIONS_ERROR により失敗します。

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

次のフラグは、すべての環境で MQGetMessageOptions オプション・フィールドに設定できます。ただし、1 の環境では、後に MQQueue.get() を使用すると、get() は MQRC_OPTIONS_ERROR により失敗します。

- MQGMO_SYNCPOINT_IF_PERSISTENT

### MQGetMessageOptions フィールド

値は、環境に関係なく以下のフィールドに設定できます。ただし、2 または 3 以外の環境では、後続に MQQueue.get() が使用されている MQGetMessageOptions にデフォルト以外の値が入っていると、 get() は MQRC_GMO_ERROR により失敗します。これは、2 または 3 以外の環境では、これらのフィールドが常に、すべての get() の正常終了後に初期値に設定されるということを意味します。

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

### 配布リスト

以下のクラスは、配布リストを作成するために使用されます。

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

MQDistributionList および MQDistributionListItems はすべての環境で作成移植できますが、 MQDistributionList を正常に作成しオープンできるのは、 2 および 3 の環境だけです。それ以外の環境で作成しオープンしようとすると、MQRC_OD_ERROR により拒否されます。

### MQPutMessageOptions フィールド

MQPMO の 4 つのフィールドは、MQPutMessageOptions クラスの次の 4 つのメンバー変数として再現されます。

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

基本的には、配布リストで使用するためのものですが、MQSeries V5 サーバーは、単一キューへの MQPUT 後に DestCount フィールドも埋め込みます。たとえば、キューがローカル・キューに解決されると、knownDestCount は 1 に設定され、その他の 2 つのフィールドは 0 に設定されます。 2 および 3 の環境では、 V5 サーバーによって設定される値は MQPutMessageOptions クラスに戻されます。その他の環境では、戻り値は次のようにシミュレートされます。

- put() が成功した場合には、unknownDestCount が 1 に設定され、その他は 0 に設定されます。
- put() が失敗した場合には、invalidDestCount が 1 に設定され、その他は 0 に設定されます。

recordFields は配布リストで使用されます。値は、環境とは無関係にいつでも recordFields に書き込むことができます。ただし、MQPutMessage オプションの後に MQDistributionList.put() ではなく MQQueue.put() が使用されていると、その値は無視されます。

### MQMD フィールド

以下の MQMD フィールドは、大部分がメッセージのセグメント化と関係しています。

- GroupId
- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

アプリケーションでこれらの MQMD フィールドのどれかをデフォルト以外の値に設定してから、 2 または 3 以外の環境で put() または get() を呼び出すと、その put() または get() で例外 (MQRC_MD_ERROR) が発生します。 2 または 3 以外の環境で put() または get() が正常終了すると、常に、新規 MQMD フィールドがそのデフォルト値に設定されたままになります。通常、グループ化されたメッセージまたはセグメント化されたメッセージは、 MQSeries V5 またはそれ以降でないキュー・マネージャーに対して実行中の Java アプリケーションには送信しないでください。このようなアプリケーションで get を発行し、取り出す物理メッセージがグループ化されたメッセージまたはセグメント化されたメッセージの一部として存在している (MQMD フィールドがデフォルト以外の値に設定されている) と、そのメッセージはエラーなしで取り出されます。ただし、MQMessage の MQMD フィールドは更新されません。 MQMessage の format プロパティーは MQFMT_MD_EXTENSION に設定され、真のメッセージ・データには、新規フィールドの値が入っている MQMDE 構造体によって接頭部が付けられます。

## V5 拡張機能

# 第9章 MQ ベース Java クラスおよびインターフェース

この章では、すべての MQSeries classes for Java クラスおよびインターフェースについて説明します。ここには、クラスおよびインターフェースごとの変数、コンストラクター、およびメソッドに関する詳しい説明が含まれています。

以下のクラスについて説明します。

- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

以下のインターフェースについて説明します。

- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

# MQChannelDefinition

```
java.lang.Object
  |
  +-- com.ibm.mq.MQChannelDefinition
```

public class **MQChannelDefinition**
extends **Object**

MQChannelDefinition クラスは、キュー・マネージャーへの接続に関する情報を送信、受信、およびセキュリティー出口に渡すために使用されます。

**注:** バインディング・モードで直接 MQSeries に接続する場合、このクラスは適用されません。

## 変数

### channelName

```
public String channelName
```

接続を確立するチャネルの名前。

### queueManagerName

```
public String queueManagerName
```

接続先のキュー・マネージャーの名前。

### maxMessageLength

```
public int maxMessageLength
```

キュー・マネージャーに送信できるメッセージの最大長。

### securityUserData

```
public String securityUserData
```

セキュリティー出口で使用されるストレージ域。ここに入れられる情報は、セキュリティー出口の呼び出しにまたがって保存され、送信出口と受信出口でも使用可能です。

### sendUserData

```
public String sendUserData
```

送信出口で使用されるストレージ域。ここに入れられる情報は、送信出口の呼び出しにまたがって保存され、セキュリティー出口と受信出口でも使用可能です。

### receiveUserData

```
public String receiveUserData
```

受信出口で使用されるストレージ域。ここに入れられる情報は、受信出口の呼び出しにまたがって保存され、送信出口とセキュリティー出口でも使用可能です。

### connectionName

```
public String connectionName
```

キュー・マネージャーが存在しているマシンの TCP/IP ホスト名。

### remoteUserId

```
public String remoteUserId
```

接続を確立するために使用されるユーザー ID。

### remotePassword

```
public String remotePassword
```

接続を確立するために使用されるパスワード。

## コンストラクター

### MQChannelDefinition

```
public MQChannelDefinition()
```

# MQChannelExit

```
java.lang.Object
   |
   +-- com.ibm.mq.MQChannelExit
```

public class **MQChannelExit**
extends **Object**

このクラスは、送信出口、受信出口、およびセキュリティー出口の呼び出し時に該当する出口に渡すコンテキスト情報を定義します。 exitResponse メンバー変数は、MQSeries Client for Java が次に実行するアクションを指示するため、出口ごとに設定する必要があります。

**注:** バインディング・モードで直接 MQSeries に接続する場合、このクラスは適用されません。

## 変数

**MQXT_CHANNEL_SEC_EXIT**
```
public final static int MQXT_CHANNEL_SEC_EXIT
```

**MQXT_CHANNEL_SEND_EXIT**
```
public final static int MQXT_CHANNEL_SEND_EXIT
```

**MQXT_CHANNEL_RCV_EXIT**
```
public final static int MQXT_CHANNEL_RCV_EXIT
```

**MQXR_INIT**
```
public final static int MQXR_INIT
```

**MQXR_TERM**
```
public final static int MQXR_TERM
```

**MQXR_XMIT**
```
public final static int MQXR_XMIT
```

**MQXR_SEC_MSG**
```
public final static int MQXR_SEC_MSG
```

**MQXR_INIT_SEC**
```
public final static int MQXR_INIT_SEC
```

**MQXCC_OK**
```
public final static int MQXCC_OK
```

**MQXCC_SUPPRESS_FUNCTION**
```
public final static int MQXCC_SUPPRESS_FUNCTION
```

**MQXCC_SEND_AND_REQUEST_SEC_MSG**
```
public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
```

**MQXCC_SEND_SEC_MSG**
```
public final static int MQXCC_SEND_SEC_MSG
```

**MQXCC_SUPPRESS_EXIT**
```
public final static int MQXCC_SUPPRESS_EXIT
```

**MQXCC_CLOSE_CHANNEL**

```
public final static int MQXCC_CLOSE_CHANNEL
```

**exitID** `public int exitID`

呼び出された出口のタイプ。 MQSecurityExit の場合、これは常に MQXT_CHANNEL_SEC_EXIT となります。 MQSendExit の場合、これは常に MQXT_CHANNEL_SEND_EXIT となり、MQReceiveExit の場合、これは常に MQXT_CHANNEL_RCV_EXIT となります。

**exitReason**

```
public int exitReason
```

出口を呼び出す理由。考えられる値は、次のとおりです。

**MQXR_INIT**
出口の開始。チャネル接続条件の折衝後で、何らかのセキュリティー・フローの送信前に呼び出されます。

**MQXR_TERM**
出口の終了。切断フローの送信後で、ソケット接続の破棄前に呼び出されます。

**MQXR_XMIT**
送信出口の場合、データをキュー・マネージャーに送信することを指示します。

受信出口の場合、データがキュー・マネージャーから受信済みであることを指示します。

**MQXR_SEC_MSG**
セキュリティー・メッセージがキュー・マネージャーから受信済みであることをセキュリティー出口に指示します。

**MQXR_INIT_SEC**
出口がキュー・マネージャーでセキュリティー・ダイアログを開始することを指示します。

**exitResponse**

```
public int exitResponse
```

MQSeries classes for Java が次に実行するアクションを指示するために、出口によって設定されます。有効な値は、次のとおりです。

**MQXCC_OK**
セキュリティーの交換が完了していることを指示するために、セキュリティー出口によって設定されます。

また、戻されたデータをキュー・マネージャーに送信することを指示するために、送信出口によって設定されます。

あるいは、戻されたデータが MQSeries Client for Java による処理のために使用可能であることを指示するために、受信出口によって設定されます。

**MQXCC_SUPPRESS_FUNCTION**
キュー・マネージャーとの通信をシャットダウンすることを指示するために、セキュリティー出口によって設定されます。

**MQXCC_SEND_AND_REQUEST_SEC_MSG**
戻されたデータをキュー・マネージャーに送信すること、およびキュー・マネージャーからの応答が必要であることを指示するために、セキュリティー出口によって設定されます。

**MQXCC_SEND_SEC_MSG**
戻されたデータをキュー・マネージャーに送信すること、およびキュー・マネージャーからの応答が不要であることを指示するために、セキュリティー出口によって設定されます。

**MQXCC_SUPPRESS_EXIT**
以後は呼び出されないことを指示するために、任意の出口によって設定されます。

**MQXCC_CLOSE_CHANNEL**
キュー・マネージャーへの接続がクローズされることを指示するために、任意の出口によって設定されます。

### maxSegmentLength

```
public int maxSegmentLength
```

キュー・マネージャーへの 1 つの伝送の最大長。

キュー・マネージャーに送信するデータを出口が戻す場合は、戻されるデータの長さがこの値を超えてはなりません。

### exitUserArea

```
public byte exitUserArea[]
```

出口で使用可能なストレージ域。

exitUserArea に入れられたデータはすべて、MQSeries Client for Java によって、 exitID が同じ出口呼び出しにまたがって保存されます。 (すなわち、送信出口、受信出口、およびセキュリティー出口にはそれぞれ、固有の独立したユーザー域があります。)

### capabilityFlags

```
public static final int capabilityFlags
```

キュー・マネージャーの能力を指示します。

MQC.MQCF_DIST_LISTS フラグしかサポートされていません。

### fapLevel

```
public static final int fapLevel
```

折衝済みのフォーマットおよびプロトコル (FAP) レベル。

## コンストラクター

### MQChannelExit

```
public MQChannelExit()
```

# MQDistributionList

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQDistributionList
```

public class **MQDistributionList**
extends **MQManagedObject** (113ページを参照してください。)

**注:** このクラスを使用できるのは、 MQSeries V5 (またはそれ以降) のキュー・マネージャーに接続している場合だけです。

MQDistributionList は、MQDistributionList コンストラクターを使用することによって、あるいは MQQueueManager に accessDistributionList メソッドを使用することによって作成されます。

配布リストは、put() メソッドへの単一の呼び出しを使用してメッセージを送信できる一連のオープン・キューを表します。 (*MQSeries アプリケーション・プログラミング・ガイド* の『Distribution lists』も参照してください。)

## コンストラクター

### MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
                          MQDistributionListItem[] litems,
                          int openOptions,
                          String alternateUserId )
                          throws MQException
```

`qMgr` はリストがオープンされるキュー・マネージャーです。

`litems` は配布リストに組み込む項目です。

残りのパラメーターの詳細については、165ページの『accessDistributionList』を参照してください。

## メソッド

### put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions )
                             throws MQException
```

メッセージを配布リスト上のキューに書き込みます。

**パラメーター**

*message*
メッセージ記述子情報および戻されたメッセージ・データが入る入出力パラメーター。

*putMessageOptions*

MQPUT のアクションを制御するオプション。 (詳細については、144ページの『MQPutMessageOptions』を参照してください。)

put が失敗した場合は、MQException が投げられます。

### getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem()
```

配布リスト中の先頭項目を戻すか、またはリストが空の場合は *null* を戻します。

### getValidDestinationCount

```
public int getValidDestinationCount()
```

正常にオープンされた配布リスト中の項目数を戻します。

### getInvalidDestinationCount

```
public int getInvalidDestinationCount()
```

正常にオープンされなかった配布リスト中の項目数を戻します。

# MQDistributionListItem

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
          |
          +-- com.ibm.mq.MQDistributionListItem
```

public class **MQDistributionListItem**
extends **MQMessageTracker** (136 ページを参照してください。)

**注:** このクラスを使用できるのは、 MQSeries V5 (またはそれ以降) のキュー・マネージャーに接続している場合だけです。

MQDistributionListItem は、配布リスト中の単一項目 (キュー) を表します。

## 変数

### completionCode

```
public int completionCode
```

この項目に対する最後の操作から取得される完了コード。このコードが、MQDistributionListの構造体である場合は、完了コードはキューのオープンに関連付けられています。 PUT 操作である場合、完了コードはこのキューへのメッセージの書き込みに関連付けられています。

初期値は ″0″ です。

### queueName

```
public String queueName
```

配布リストで使用したいキューの名前。モデル・キューの名前にすることはできません。

初期値は ″″ です。

### queueManagerName

```
public String queueManagerName
```

キューが定義されているキュー・マネージャーの名前。

初期値は ″″ です。

### reasonCode

```
public int reasonCode
```

この項目に対する最後の操作から取得される理由コード。このコードが、MQDistributionListの構造体である場合は、理由コードはキューのオープンに関連付けられています。 PUT 操作である場合は、理由コードはこのキューへのメッセージの書き込みに関連付けられています。

初期値は ″0″ です。

## コンストラクター

### MQDistributionListItem

```
public MQDistributionListItem()
```

## MQDistributionListItem

新規 MQDistributionListItem オブジェクトを構成します。

# MQEnvironment

```
java.lang.Object
   |
   +-- com.ibm.mq.MQEnvironment
```

public class **MQEnvironment**
extends **Object**

**注:** このクラスのメソッドと属性はすべて MQSeries classes for Java クライアント接続に適用されますが、 enableTracing、disableTracing、properties、および version_notice はバインド接続にしか適用されません。

MQEnvironment には、 MQQueueManager オブジェクト (およびそれと対応している MQSeries への接続) が構成される環境を制御する静的メンバー変数が入ります。

MQEnvironment クラスに設定される値は、 MQQueueManager コンストラクターが呼び出されると有効になるので、 MQEnvironment クラスの値は MQQueueManager インスタンスの構成前に設定する必要があります。

## 変数

**注:** * のマークが付いている変数は、バインディング・モードで直接 MQSeries に接続する場合には適用されません。

### version_notice

```
public final static String version_notice
```

MQSeries classes for Java の現行バージョン。

### securityExit*

```
public static MQSecurityExit securityExit
```

セキュリティー出口により、キュー・マネージャーに接続しようとする時に発生するセキュリティー・フローをカスタマイズできます。

ユーザー独自のセキュリティー出口を提供するには、 MQSecurityExit インターフェースをインプリメントするクラスを定義し、 securityExit をそのクラスのインスタンスに割り当てます。これ以外の場合には、 securityExit を null に設定したままにしておくことができます。その場合、セキュリティー出口が呼び出されることはありません。

175ページの『MQSecurityExit』も参照してください。

### sendExit*

```
public static MQSendExit sendExit
```

送信出口により、キュー・マネージャーに送信するデータを調べることができ、場合によっては更新することもできます。通常、これはキュー・マネージャーで対応している受信出口と組み合わせて使用されます。

ユーザー独自の送信出口を提供するには、 MQSendExit インターフェースをインプリメントするクラスを定義し、 sendExit をそのクラスのイ

ンスタンスに割り当てます。これ以外の場合には、sendExit を null に設定したままにしておくことができます。その場合、送信出口が呼び出されることはありません。

177ページの『MQSendExit』も参照してください。

**receiveExit***

```
public static MQReceiveExit receiveExit
```

受信出口により、キュー・マネージャーから受信するデータを調べることができ、場合によっては更新することもできます。通常、これはキュー・マネージャーで対応している送信出口と組み合わせて使用されます。

ユーザー独自の受信出口を提供するには、 MQReceiveExit インターフェースをインプリメントするクラスを定義し、 receiveExit をそのクラスのインスタンスに割り当てます。これ以外の場合には、receiveExit を null に設定したままにしておくことができます。その場合、受信出口が呼び出されることはありません。

173ページの『MQReceiveExit』も参照してください。

**hostname***

```
public static String hostname
```

MQSeries サーバーが存在しているマシンの TCP/IP ホスト名。hostname もプロパティーの指定変更もまだ設定されていない場合は、ローカル・キュー・マネージャーへの接続にバインディング・モードが使用されます。

**port*** `public static int port`

接続先のポート。これは、MQSeries サーバーが接続要求の着信を listen しているポートです。デフォルトの値は 1414 です。

**channel***

```
public static String channel
```

宛先キュー・マネージャー上の接続先のチャネルの名前。クライアント・モードで使用する MQQueueManager インスタンスを構成する前に、このメンバー変数またはこれに対応する当該のプロパティーを設定しなければなりません 。

**userID***

```
public static String userID
```

MQSeries 環境変数 MQ_USER_ID と同等。

このクライアントにセキュリティー出口が定義されていないと、 userID の値がサーバーに送信され、サーバー・セキュリティー出口が呼び出された時に、その出口で使用可能になります。この値は、MQSeries クライアントの識別を検証するために使用することができます。

デフォルト値は "" です。

**password***

```
public static String password
```

MQSeries 環境変数 MQ_PASSWORD と同等。

このクライアントにセキュリティー出口が定義されていないと、password の値がサーバーに送信され、サーバー・セキュリティー出口が呼び出された時に、その出口で使用可能になります。この値は、MQSeries クライアントの識別を検証するために使用することができます。

デフォルト値は "" です。

### properties

```
public static java.util.Hashtable properties
```

MQSeries 環境を定義している一連のキー / 値のペア。

このハッシュ・テーブルにより、環境プロパティーを個別の変数としてではなく、キー / 値のペアとして設定することができます。

また、これらのプロパティーは、 MQQueueManager コンストラクターのパラメーターにハッシュ・テーブルとして渡すこともできます。コンストラクターで渡されるプロパティーは、このプロパティー変数で設定される値より優先されますが、別の方法で交換可能です。定義されているプロパティーの優先順位は、次のとおりです。

1. MQQueueManager コンストラクターの `properties` パラメーター
2. MQEnvironment.properties
3. その他の MQEnvironment 変数
4. 定数のデフォルト値

使用できるキー / 値のペアは、以下の表に示してあります。

| キー | 値 |
|---|---|
| MQC.CCSID_PROPERTY | 整数 (MQEnvironment.CCSID を指定変更します) |
| MQC.CHANNEL_PROPERTY | ストリング (MQEnvironment.channel を指定変更します) |
| MQC.CONNECT_OPTIONS_PROPERTY | 整数 (デフォルトとして MQC.MQCNO_NONE を使用します) |
| MQC.HOST_NAME_PROPERTY | ストリング (MQEnvironment.hostname を指定変更します) |
| MQC.ORB_PROPERTY | org.omg.CORBA.ORB (オプショナル) |
| MQC.PASSWORD_PROPERTY | ストリング (MQEnvironment.password を指定変更します) |
| MQC.PORT_PROPERTY | 整数 (MQEnvironment.port を指定変更します) |
| MQC.RECEIVE_EXIT_PROPERTY | MQReceiveExit (MQEnvironment.receiveExit を指定変更します) |
| MQC.SECURITY_EXIT_PROPERTY | MQSecurityExit (MQEnvironment.securityExit を指定変更します) |
| MQC.SEND_EXIT_PROPERTY | MQSendExit (MQEnvironment.sendExit を指定変更します) |

| キー | 値 |
|---|---|
| MQC.TRANSPORT_PROPERTY | MQC.TRANSPORT_MQSERIES_BINDINGS<br>または<br>MQC.TRANSPORT_MQSERIES_CLIENT<br>または<br>MQC.TRANSPORT_VISIBROKER<br>または<br>MQC.TRANSPORT_MQSERIES (″hostname″ の値を基にしてバインドかクライアントかを選択するデフォルト) |
| MQC.USER_ID_PROPERTY | ストリング (MQEnvironment.userID を指定変更します) |

### CCSID*

```
public static int CCSID
```

クライアントで使用される CCSID。

この値を変更すると、接続先のキュー・マネージャーが MQSeries ヘッダーの情報を変換する方法に影響します。 MQSeries ヘッダーのデータはすべて、 MQMessage クラスの applicationIdData および putApplicationName フィールド中のデータを除き、 ASCII コード・セットの不変部分から作成されます。 (116ページの『MQMessage』を参照してください。)

これら 2 つのフィールドに ASCII コード・セットの可変部分からの文字が使用されないようにするには、 CCSIDを 819 からその他の任意の ASCII コード・セットに変更した方が安全です。

クライアントの CCSID を、接続しているキュー・マネージャーの CCSIDと同じにすると、キュー・マネージャーはメッセージ・ヘッダーの変換を実行しなくなるので、パフォーマンス上の利点が得られます。

デフォルト値は 819 です。

## コンストラクター

### MQEnvironment

```
public MQEnvironment()
```

## メソッド

### disableTracing

```
public static void disableTracing()
```

MQSeries Client for Javaのトレース機能をオフにします。

### enableTracing

```
public static void enableTracing(int level)
```

MQSeries Client for Javaのトレース機能をオンにします。

**パラメーター:**

*level* 必要なトレースの 1 〜 5 (5 が最も詳細) の範囲のレベル。

### enableTracing

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

MQSeries Client for Javaのトレース機能をオンにします。

**パラメーター:**

*level* 必要なトレースの 1 ～ 5 (5 が最も詳細) の範囲のレベル。

*stream* トレースが書き込まれるストリーム。

### setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

指定された MQConnectionManager をデフォルトの ConnectionManager に設定します。デフォルトの ConnectionManager は、 MQQueueManager コンストラクターに何も ConnectionManager が指定されない場合に使用されます。加えて、このメソッドを使用すると MQPoolTokens の集合は空になります。

**パラメーター:**

*cxManager*
デフォルトの ConnectionManager にする MQConnectionManager。

### setDefaultConnectionManager

```
public static void setDefaultConnectionManager
                    (javax.resource.spi.ConnectionManager cxManager)
```

デフォルトの ConnectionManager を設定し、MQPoolTokens の集合を空にします。デフォルトの ConnectionManager は、 MQQueueManager コンストラクターに何も ConnectionManager が指定されない場合に使用されます。

このメソッドには、JAAS 1.0 以降がインストールされた Java 2 v1.3 以降の JVM が必要です。

**パラメーター:**

*cxManager*
デフォルトの ConnectionManager
(javax.resource.spi.ConnectionManager インターフェースをインプリメントする)。

### getDefaultConnectionManager

```
public static javax.resource.spi.ConnectionManager
                                  getDefaultConnectionManager()
```

デフォルトの ConnectionManager を戻します。デフォルトの ConnectionManager が実際に MQConnectionManager である場合は、ヌルを戻します。

### addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token)
```

指定された MQPoolToken をトークンの集合に追加します。デフォルトの ConnectionManager にとって、これは暗示になる場合があります。というのも、通常、デフォルトの ConnectionManager は、集合の中にトークンが存在しているときにだけ使用可能になるからです。

**パラメーター:**

*token* トークンの集合に追加する MQPoolToken。

### addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken()
```

MQPoolToken を構成し、それをトークンの集合に追加します。MQPoolToken は、アプリケーションに戻され、後で removeConnectionPoolToken() に渡されます。

### removeConnectionPoolToken

```
public static void removeConnectionPoolToken(MQPoolToken token)
```

指定された MQPoolToken をトークンの集合から除去します。指定された MQPoolToken が集合内に存在しない場合は、何も行われません。

**パラメーター:**

*token* トークンの集合から除去する MQPoolToken。

# MQException

```
java.lang.Object
   |
   +-- java.lang.Throwable
          |
          +-- java.lang.Exception
                 |
                 +-- com.ibm.mq.MQException
```

public class **MQException**
extends **Exception**

MQSeries エラーが発生した場合には、必ず MQException が投げられます。ログに記録される例外の出力ストリームを変更するには、MQException.log の値を設定します。デフォルト値は System.err です。このクラスには、完了コードとエラー・コードの定数の定義が含まれています。 MQCC_ で始まる定数は MQSeries 完了コードで、MQRC_ で始まる定数は MQSeries 理由コードです。 *MQSeries アプリケーション・プログラミング・リファレンス* には、これらのエラーと考えられる原因が詳しく記載されています。

## 変数

**log** `public static` **`java.io.outputStreamWriter`** `log`

例外がログに記録されるストリーム。 (デフォルトは System.err です。) これを null に設定すると、ログ記録は行われません。

**completionCode**

`public int completionCode`

エラーの発生を通知する MQSeries 完了コード。考えられる値は、次のとおりです。

- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

**reasonCode**

`public int reasonCode`

エラーを説明する MQSeries 理由コード。理由コードについて詳しくは、*MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

**exceptionSource**

`public Object exceptionSource`

例外を投げたオブジェクト・インスタンス。エラーの原因を判別する場合には、これを診断の一部として使用することができます。

## コンストラクター

**MQException**

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

## MQException

新規 MQException オブジェクトを構成します。

### Parameters

*completionCode*
MQSeries 完了コード。

*reasonCode*
MQSeries 理由コード。

*source* エラーが発生したオブジェクト。

# MQGetMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQGetMessageOptions
```

public class **MQGetMessageOptions**
extends **Object**

このクラスには、MQQueue.get() の動作を制御するオプションが含まれています。

**注:** このクラスで使用可能な一部のオプションの動作は、使用される環境によって異なります。これらのエレメントには * のマークが付けられています。詳細は、83ページの『第8章 環境による動作の違い』を参照してください。

## 変数

### options

```
public int options
```

MQQueue.get のアクションを制御するオプション。次の値のどれかを指定しても、どれも指定しなくてもかまいません。複数のオプションが必要な場合は、これらの値をまとめて追加するか、ビット単位の OR 演算子を使用して結合することができます。

**MQC.MQGMO_NONE**

**MQC.MQGMO_WAIT**
メッセージの到着を待ちます。

**MQC.MQGMO_NO_WAIT**
適切なメッセージがない場合には、即時に戻ります。

**MQC.MQGMO_SYNCPOINT**
メッセージを同期点制御のもとで読み取ります。メッセージは他のアプリケーションが使用できないようにマークされていますが、キューから削除されるのは、作業単位がコミットされる時だけで。作業単位がバックアウトされると、このメッセージは再び使用可能になります。

**MQC.MQGMO_NO_SYNCPOINT**
メッセージを同期点制御なしに読み取ります。

**MQC.MQGMO_BROWSE_FIRST**
キューの先頭からブラウズします。

**MQC.MQGMO_BROWSE_NEXT**
キューの現在位置からブラウズします。

**MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR***
ブラウズ・カーソル以下のメッセージをブラウズします。

**MQC.MQGMO_MSG_UNDER_CURSOR**
ブラウズ・カーソル以下のメッセージを読み取ります。

**MQC.MQGMO_LOCK***
ブラウズするメッセージをロックします。

**MQC.MQGMO_UNLOCK***
前にロックしたメッセージをアンロックします。

**MQC.MQGMO_ACCEPT_TRUNCATED_MSG**
メッセージ・データの切り捨てを許可します。

**MQC.MQGMO_FAIL_IF_QUIESCING**
キュー・マネージャーが静止している場合には、失敗します。

**MQC.MQGMO_CONVERT**
アプリケーション・データがメッセージ・バッファーにコピーされる前に、 MQMessage の characterSet および encoding 属性に準拠するようにそのデータを変換することを要求します。データ変換は、データがメッセージ・バッファーから検索されるときにも適用されるので、アプリケーションでは通常このオプションを設定しません。

**MQC.MQGMO_SYNCPOINT_IF_PERSISTENT***
メッセージが持続している場合には、同期点制御付きでメッセージを読み取ります。

**MQC.MQGMO_MARK_SKIP_BACKOUT***
キューのメッセージを復元しないで作業単位をバックアウトすることができます。

**セグメント化およびグループ化** MQSeries メッセージは、単一のエンティティーとして送受信することも、送受信のために幾つかのセグメントに分割することも、さらにグループ内の他のメッセージにリンクすることもできます。

送信されるデータの各断片は、完全な*論理* メッセージ、または長い論理メッセージの 1 つのセグメントとなる 1 つの*物理* メッセージとして認識されます。

通常、物理メッセージごとに異なる `MsgId` があります。単一論理メッセージのすべてのセグメントの `groupId` 値と `MsgSeqNumber` 値は同じですが、`Offset` 値はセグメントごとに異なります。 `Offset` フィールドには、論理メッセージの先頭からの物理メッセージ中のデータのオフセットが指定されます。セグメントは個々の物理メッセージであるため、その `MsgId` 値はセグメントごとに異なっているのが普通です。

1 つのグループを形成している各論理メッセージの `groupId` 値は同じですが、 `MsgSeqNumber` 値はグループ内のメッセージごとに異なります。グループ内の各メッセージをセグメント化することもできます。

以下のオプションは、セグメント化またはグループ化されたメッセージを処理するために使用することができます。

**MQC.MQGMO_LOGICAL_ORDER***
グループ内のメッセージおよび論理メッセージのセグメントを論理順序に戻します。

**MQC.MQGMO_COMPLETE_MSG***
完全な論理メッセージのみ検索します。

**MQC.MQGMO_ALL_MSGS_AVAILABLE***
グループ内のメッセージがすべて使用可能な場合のみ、そのグループからメッセージを検索します。

**MQC.MQGMO_ALL_SEGMENTS_AVAILABLE***
グループ内のセグメントがすべて使用可能な場合のみ、その論理メッセージのセグメントを検索します。

**waitInterval**

```
public int waitInterval
```

MQQueue.get 呼び出しが適切なメッセージの到着を待つ最大時間 (ミリ秒数)。 MQC.MQGMO_WAIT と組み合わせて使用されます。
MQC.MQWI_UNLIMITED の値は、無制限の待機が必要であることを指示します。

**resolvedQueueName**

```
public String resolvedQueueName
```

キュー・マネージャーが、メッセージが取り出されたキューのローカル名を入れる出力フィールド。別名キューまたはモデル・キューをオープンした場合、これはキューのオープンに使用した名前とは異なります。

**matchOptions***

```
public int matchOptions
```

検索するメッセージを判別するための選択基準。以下のマッチング・オプションを設定することができます。

**MQC.MQMO_MATCH_MSG_ID**
マッチングするメッセージ ID。

**MQC.MQMO_MATCH_CORREL_ID**
マッチングする相関 ID。

**MQC.MQMO_MATCH_GROUP_ID**
マッチングするグループ ID。

**MQC.MQMO_MATCH_MSG_SEQ_NUMBER**
メッセージ順序番号でマッチング。

**MQC.MQMO_NONE**
マッチングは不要。

**groupStatus***

```
public char groupStatus
```

検索するメッセージがグループ内にあるかどうか、そしてグループ内にある場合は、それがグループ内の最後のメッセージであるかどうかを示す出力フィールド。考えられる値は、次のとおりです。

**MQC.MQGS_NOT_IN_GROUP**
メッセージはグループ内にありません。

**MQC.MQGS_MSG_IN_GROUP**
メッセージはグループ内にありますが、そのグループ内の最後のメッセージではありません。

**MQC.MQGS_LAST_MSG_IN_GROUP**
メッセージはグループ内の最後のメッセージです。この値は、グループ内にメッセージが 1 つしかない場合でも戻されます。

**segmentStatus***

```
public char segmentStatus
```

検索するメッセージが論理メッセージのセグメントであるかどうかを指示する出力フィールドです。メッセージがセグメントである場合、このフラグは、そのセグメントが最後のセグメントかどうかを指示します。考えられる値は、次のとおりです。

**MQC.MQSS_NOT_A_SEGMENT**
メッセージはセグメントではありません。

**MQC.MQSS_SEGMENT**
メッセージはセグメントですが、論理メッセージの最後のセグメントではありません。

**MQC.MQSS_LAST_SEGMENT**
メッセージは論理メッセージの最後のセグメントです。論理メッセージにセグメントが 1 つしかない場合もこの値が戻されます。

**segmentation***

```
public char segmentation
```

論理メッセージのセグメントである、取り出されたメッセージに対して、セグメント化が許可されているかどうかを示す出力フィールド。考えられる値は、次のとおりです。

**MQC.MQSEG_INHIBITED**
セグメント化は許可されていません。

**MQC.MQSEG_ALLOWED**
セグメント化は許可されています。

## コンストラクター

**MQGetMessageOptions**

```
public MQGetMessageOptions()
```

MQC.MQGMO_NO_WAIT (ゼロの待機インターバル、およびブランクで解決されるキュー名) に設定されたオプションを用いて、新規 MQGetMessageOptions オブジェクトを構成します。

# MQManagedObject

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
```

public class **MQManagedObject**
extends **Object**

MQManagedObject は MQQueueManager、MQQueue、および MQProcess のスーパークラスです。これは、これらのリソースの属性を照会および設定する機能を提供します。

## 変数

### alternateUserId

```
public String alternateUserId
```

このリソースをオープンする時に指定された代替ユーザー ID (ある場合)。この属性を設定しても効果はありません。

### name

```
public String name
```

このリソースの名前 (アクセス・メソッドに指定された名前かキュー・マネージャーによって動的キューに割り振られた名前のいずれか)。この属性を設定しても効果はありません。

### openOptions

```
public int openOptions
```

このリソースをオープンする時に指定されたオプション。この属性を設定しても効果はありません。

### isOpen

```
public boolean isOpen
```

このリソースが現在オープンされているかどうかを示す属性。この属性は*使用しないでください*。この属性を設定しても効果はありません。

### connectionReference

```
public  MQQueueManager connectionReference
```

このリソースが属しているキュー・マネージャー。この属性を設定しても効果はありません。

### closeOptions

```
public int closeOptions
```

リソースのクローズ方法を制御するために設定される属性。デフォルト値は MQC.MQCO_NONE で、永続動的キューと、キューを作成したオブジェクトによって現在アクセスされている一時動的キュー以外のすべてのリソースにとっては、これが唯一の暗黙的な値となります。このようなキューには、以下の追加値が許可されています。

#### MQC.MQCO_DELETE

メッセージが 1 つもない場合には、キューを削除します。

**MQC.MQCO_DELETE_PURGE**
キューのメッセージをすべて除去して、そのキューを削除します。

## コンストラクター

### MQManagedObject

```
protected MQManagedObject()
```

コンストラクター・メソッド。

## メソッド

### getDescription

```
public String getDescription()
```

MQException を投げます。

キュー・マネージャーに保持されているときは、このリソースの記述を戻します。

リソースのクローズ後にこのメソッドが呼び出された場合は、MQException が投げられます。

### inquire

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

MQException を投げます。

整数の配列およびオブジェクト (キュー、プロセス、またはキュー・マネージャー) の属性が入っている一連の文字ストリングを戻します。

照会する属性はセレクター配列に指定します。暗黙的セレクターおよびそれと対応する整数値の詳細については、*MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

MQManagedObject、MQQueue、MQQueueManager、および MQProcess に定義されている getXXX() メソッドを使用すると、より共通性の高い属性が大量に照会される可能性があるので注意してください。

**パラメーター**

*selectors*
照会する値をもつ属性を識別する整数配列。

*intAttrs* 整数属性値が戻される配列。整数属性値が、セレクター配列中の整数属性セレクターと同じ順序で戻されます。

*charAttrs*
文字属性が連結されて戻されるバッファー。文字属性が、セレクター配列中の文字属性セレクターと同じ順序で戻されます。各属性ストリングの長さは属性ごとに固定されています。

照会が失敗した場合は、MQException が投げられます。

### isOpen

```
public boolean isOpen()
```

isOpen 変数の値を戻します。

### set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

MQException を投げます。

セレクターのベクトルに定義された属性を設定します。

設定される属性はセレクター配列に指定します。暗黙的セレクターおよびそれと対応する整数値の詳細については、*MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

MQQueue に定義されている setXXX() メソッドを使用すると、一部のキュー属性が設定される可能性があるので注意してください。

#### Parameters

*selectors*
設定する値をもつ属性を識別する整数配列。

*intAttrs* 設定する整数属性値の配列。これらの値は、セレクター配列中の整数属性セレクターと同じ順序になっていなければなりません。

*charAttrs*
設定する文字属性が連結されるバッファー。これらの値は、セレクター配列中の文字属性セレクターと同じ順序になっていなければなりません。各文字属性の長さは固定されています。

set が失敗した場合は、MQException が投げられます。

### close

```
public synchronized void close()
```

MQException を投げます。

オブジェクトをクローズします。このメソッドの呼び出し後には、このリソースに対するどの操作も許可されません。 close メソッドの動作は、closeOptions 属性を設定することによって更新できる場合があります。

MQSeries 呼び出しが失敗した場合は、MQException が投げられます。

# MQMessage

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessage
```

public class **MQMessage**
implements **DataInput**, **DataOutput**

MQMessage は、メッセージ記述子と MQSeries メッセージのデータの両方を表します。データをメッセージから読み取るための readXXX メソッドのグループと、データをメッセージに書き込むための writeXXX メソッドのグループがあります。これらの read メソッドと write メソッドで使用される数値とストリングの形式は、encoding メンバー変数と characterSet メンバー変数によって制御できます。残りのメンバー変数には、メッセージが送信アプリケーションと受信アプリケーションの間で伝送される時にアプリケーション・メッセージ・データと同時に発生する制御情報が入ります。アプリケーションは、メッセージをキューに書き込む前に値をメンバー変数に設定することができ、メッセージをキューから取り出した後に値を読み取ることができます。

## 変数

**report** `public int report`

report は別のメッセージに関するメッセージです。このメンバー変数により、元のメッセージを送信するアプリケーションで、メッセージが必要か、アプリケーション・メッセージ・データをれレポート・メッセージに組み込むかどうか、およびレポートまたは応答のメッセージ ID と相関 ID を設定する方法も指定することができます。次のレポート・タイプを要求しても、どれも要求しなくてもかまいません。

- 例外
- 有効期限
- 到達確認
- 送達確認

タイプごとに、アプリケーション・メッセージ・データをレポート・メッセージに組み込むかどうかによって、下記の対応する 3 つの値のうちの 1 つだけを指定する必要があります。

**注:** 次のリスト中で ****** マーク付きの値は、 MVS キュー・マネージャーではサポートされないので、ユーザー・アプリケーションが MVS キュー・マネージャーにアクセスすることがよくある場合には、アプリケーションが実行中のプラットフォームに関係なく使用しないでください。

有効な値は、次のとおりです。

- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA**

- MQC.MQRO_COA
- MQC.MQRO_COA_WITH_DATA
- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

次のどちらかを指定して、レポートや応答メッセージのメッセージ ID を生成する方法を制御できます。

- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

次のどちらかを指定して、レポートや応答メッセージの相関 ID を設定する方法を制御できます。

- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

次のどちらかを指定して、元のメッセージを宛先キューに送達できない場合のメッセージの後処理を制御できます。

- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG **

report オプションが指定されていないと、デフォルトは次のようになります。

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

以下の一方または両方を指定して、受信アプリケーションが肯定アクションまたは否定アクション・レポート・メッセージあるいはその両方を送信するように要求することができます。

- MQRO_PAN
- MQRO_NAN

### messageType

```
public int messageType
```

メッセージのタイプを指示します。現在、システムによって以下の値が定義されています。

- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

アプリケーションで定義済みの値を使用することもできます。これらの値は、 MQC.MQMT_APPL_FIRST ～ MQC.MQMT_APPL_LAST の範囲内とする必要があります。

このフィールドのデフォルト値は MQC.MQMT_DATAGRAM です。

### expiry

```
public int expiry
```

メッセージを書き込むアプリケーションによって設定され、 10 分の 1 秒単位で表現された満了時間。メッセージの満了時間が経過した後、そのメッセージはキュー・マネージャーが廃棄できるメッセージになります。メッセ

ージに MQC.MQRO_EXPIRATION フラグの 1 つが指定されている場合には、レポートがメッセージの廃棄時に生成されます。

デフォルト値は MQC.MQEI_UNLIMITED (メッセージの有効期限が切れることはないという意味) です。

### feedback

```
public int feedback
```

レポートの性質を指示するために、タイプ MQC.MQMT_REPORT のメッセージで使用される変数。システムによって、以下のフィードバック・コードが定義されています。

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

MQC.MQFB_APPL_FIRST ～ MQC.MQFB_APPL_LAST の範囲内のアプリケーションで定義済みフィードバック値を使用することもできます。

このフィールドのデフォルト値は MQC.MQFB_NONE (フィードバックが提供されないことを指示する) です。

### encoding

```
public int encoding
```

このメンバー変数は、アプリケーション・メッセージ・データ中の数値に使用される表現を指定します。これは、2 進数、パック 10 進数、および浮動小数点データに適用されます。 read メソッドと write メソッドのこれらの数値形式への動作は、適宜更新されます。

以下のエンコードが 2 進整数に定義されています。

**MQC.MQENC_INTEGER_NORMAL**
Java と同じビッグ・エンディアン整数

**MQC.MQENC_INTEGER_REVERSED**
PC で使用される場合と同じリトル・エンディアン整数

以下のエンコードがパック 10 進数に定義されています。

**MQC.MQENC_DECIMAL_NORMAL**
システム/390 で使用される場合と同じビッグ・エンディアン・パック 10 進数

**MQC.MQENC_DECIMAL_REVERSED**
リトル・エンディアン・パック 10 進数

以下のエンコードが浮動小数点数に定義されています。

**MQC.MQENC_FLOAT_IEEE_NORMAL**
Java と同じビッグ・エンディアン IEEE 浮動小数点

**MQC.MQENC_FLOAT_IEEE_REVERSED**
PC で使用される場合と同じリトル・エンディアン IEEE 浮動小数点

**MQC.MQENC_FLOAT_S390**
システム/390 形式の浮動小数点

encoding フィールドの値は、これら 3 つのセクションからの各値を 1 つにまとめて追加する (またはビット単位の OR 演算子を使用する) ことによって構成する必要があります。デフォルト値は、次のとおりです。

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

便宜上、この値は MQC.MQENC_NATIVE とも表現されます。この設定により、 writeInt() はビッグ・エンディアン整数を書き込み、 readInt() はビッグ・エンディアン整数を読み取ることができます。その代りに、フラグ MQC.MQENC_INTEGER_REVERSED を設定すると、 writeInt()はリトル・エンディアン整数を書き込み、 readInt() はリトル・エンディアン整数を読み取ることになります。

IEEE 形式の浮動小数点からシステム/390 形式の浮動小数点に変換すると、精度が低下する可能性があるので注意してください。

### characterSet

```
public int characterSet
```

これは、アプリケーション・メッセージ・データ中の文字データのコード化文字セット ID を指定します。これに従って、readString、readLine、および writeString メソッドの動作が更新されます。

このフィールドのデフォルト値は MQC.MQCCSI_Q_MGR で、これはアプリケーション・メッセージ・データの文字データがキュー・マネージャーの文字セットになっていることを指定します。 表13 に示されている追加の文字セット値がサポートされています。

*表 13. 文字セット ID*

| characterSet | 説明 |
|---|---|
| 819 | iso-8859-1 / latin1 / ibm819 |
| 912 | iso-8859-2 / latin2 / ibm912 |
| 913 | iso-8859-3 / latin3 / ibm913 |
| 914 | iso-8859-4 / latin4 / ibm914 |
| 915 | iso-8859-5 / cyrillic / ibm915 |
| 1089 | iso-8859-6 / arabic / ibm1089 |
| 813 | iso-8859-7 / greek / ibm813 |
| 916 | iso-8859-8 / hebrew / ibm916 |
| 920 | iso-8859-9 / latin5 / ibm920 |
| 37 | ibm037 |
| 273 | ibm273 |
| 277 | ibm277 |

*表 13. 文字セット ID (続き)*

| characterSet | 説明 |
|---|---|
| 278 | ibm278 |
| 280 | ibm280 |
| 284 | ibm284 |
| 285 | ibm285 |
| 297 | ibm297 |
| 420 | ibm420 |
| 424 | ibm424 |
| 437 | ibm437 / PC オリジナル |
| 500 | ibm500 |
| 737 | ibm737 / PC ギリシャ語 |
| 775 | ibm775 / PC バルト語 |
| 838 | ibm838 |
| 850 | ibm850 / PC Latin 1 |
| 852 | ibm852 / PC Latin 2 |
| 855 | ibm855 / PC キリル文字 |
| 856 | ibm856 |
| 857 | ibm857 / PC トルコ語 |
| 860 | ibm860 / PC ポルトガル語 |
| 861 | ibm861 / PC アイスランド語 |
| 862 | ibm862 / PC ヘブライ語 |
| 863 | ibm863 / PC カナダ・フランス語 |
| 864 | ibm864 / PC アラビア語 |
| 865 | ibm865 / PC 北欧ゲルマン系言語 |
| 866 | ibm866 / PC ロシア語 |
| 868 | ibm868 |
| 869 | ibm869 / PC 現代ギリシャ語 |
| 870 | ibm870 |
| 871 | ibm871 |
| 874 | ibm874 |
| 875 | ibm875 |
| 918 | ibm918 |
| 921 | ibm921 |
| 922 | ibm922 |
| 930 | ibm930 |
| 933 | ibm933 |
| 935 | ibm935 |
| 937 | ibm937 |
| 939 | ibm939 |
| 942 | ibm942 |
| 948 | ibm948 |
| 949 | ibm949 |
| 950 | ibm950 / Big 5 中国語 (繁体字) |
| 964 | ibm964 / CNS 11643 中国語 (繁体字) |
| 970 | ibm970 |
| 1006 | ibm1006 |
| 1025 | ibm1025 |
| 1026 | ibm1026 |
| 1097 | ibm1097 |
| 1098 | ibm1098 |
| 1112 | ibm1112 |
| 1122 | ibm1122 |
| 1123 | ibm1123 |
| 1124 | ibm1124 |

*表 13. 文字セット ID (続き)*

| characterSet | 説明 |
|---|---|
| 1381 | ibm1381 |
| 1383 | ibm1383 |
| 2022 | JIS |
| 932 | PC 日本語 |
| 954 | EUCJIS |
| 1250 | Windows Latin 2 |
| 1251 | Windows キリル文字 |
| 1252 | Windows Latin 1 |
| 1253 | Windows ギリシャ語 |
| 1254 | Windows トルコ語 |
| 1255 | Windows ヘブライ語 |
| 1256 | Windows アラビア語 |
| 1257 | Windows バルト語 |
| 1258 | Windows ベトナム語 |
| 33722 | ibm33722 |
| 5601 | ksc-5601 韓国語 |
| 1200 | Unicode |
| 1208 | UTF-8 |

### format

```
public String format
```

メッセージの送信側が、メッセージ中のデータの性質を受信側に指示するために使用する形式名。ユーザー独自の形式名を使用することができますが、文字「MQ」で始まる名前は、キュー・マネージャーによって定義されているという意味があります。キュー・マネージャーに組み込まれた形式は、次のとおりです。

**MQC.MQFMT_NONE**
形式名なし

**MQC.MQFMT_ADMIN**
コマンド・サーバー要求 / 応答メッセージ

**MQC.MQFMT_COMMAND_1**
タイプ 1 コマンド応答メッセージ

**MQC.MQFMT_COMMAND_2**
タイプ 2 コマンド応答メッセージ

**MQC.MQFMT_DEAD_LETTER_HEADER**
送達不能ヘッダー

**MQC.MQFMT_EVENT**
イベント・メッセージ

**MQC.MQFMT_PCF**
プログラム式コマンド形式のユーザー定義メッセージ

**MQC.MQFMT_STRING**
全体が文字で構成されているメッセージ

**MQC.MQFMT_TRIGGER**
トリガー・メッセージ

**MQC.MQFMT_XMIT_Q_HEADER**
伝送キュー・ヘッダー

デフォルト値は MQC.MQFMT_NONE です。

### priority

```
public int priority
```

メッセージ優先順位。また、特殊値 MQC.MQPRI_PRIORITY_AS_Q_DEF もアウトバウンド・メッセージに設定できますが、この場合には、そのメッセージの優先順位は宛先キューのデフォルト priority 属性から取られます。

デフォルト値は MQC.MQPRI_PRIORITY_AS_Q_DEF です。

### persistence

```
public int persistence
```

メッセージ持続性。以下の値が定義されています。
- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

デフォルト値は MQC.MQPER_PERSISTENCE_AS_Q_DEF で、メッセージの持続性が宛先キューのデフォルト persistence 属性から取られることを指示します。

### messageId

```
public byte messageId[]
```

MQQueue.get() 呼び出しの場合、このフィールドには、検索するメッセージのメッセージ ID を指定します。通常、キュー・マネージャーは、メッセージ ID および相関 ID が指定されたものと一致する最初のメッセージを戻します。特殊値 MQC.MQMI_NONE により、*任意の* メッセージ ID とマッチングすることができます。

MQQueue.put() 呼び出しの場合、このフィールドには、使用するメッセージ ID を指定します。 MQC.MQMI_NONE が指定されていると、キュー・マネージャーはメッセージの書き込み時に一意のメッセージ ID を生成します。このメンバー変数の値は、使用したメッセージ ID を指示するために、書き込み後に更新されます。

デフォルト値は MQC.MQMI_NONE です。

### correlationId

```
public byte correlationId[]
```

MQQueue.get() 呼び出しの場合、このフィールドには、検索するメッセージの相関 ID を指定します。通常、キュー・マネージャーは、メッセージ ID および相関 ID が指定されたものと一致する最初のメッセージを戻します。特殊値 MQC.MQCI_NONEにより、*任意の* 相関 ID とマッチングすることができます。

MQQueue.put() 呼び出しの場合、このフィールドには、使用する相関 ID を指定します。

デフォルト値は MQC.MQCI_NONE です。

### backoutCount

```
public int backoutCount
```

前に、メッセージが作業単位の一環として MQQueue.get() 呼び出しによって戻され、その後にバックアウトされた回数のカウント。

デフォルト値はゼロです。

### replyToQueueName

```
public String replyToQueueName
```

メッセージの get 要求を発行したアプリケーションが MQC.MQMT_REPLY および MQC.MQMT_REPORT メッセージを送信する先のメッセージ・キューの名前。

デフォルト値は "" です。

### replyToQueueManagerName

```
public String replyToQueueManagerName
```

応答メッセージまたはレポート・メッセージの送信先のキュー・マネージャーの名前。

デフォルト値は "" です。

この値が MQQueue.put() 呼び出しで "" になっていると、QueueManager がこの値を書き込みます。

**userId** `public String userId`

メッセージの識別コンテキストの一部で、このメッセージの発信元のユーザーを識別します。

デフォルト値は "" です。

**accountingToken**

`public byte accountingToken[]`

メッセージの識別コンテキストの一部。これにより、アプリケーションは、メッセージの結果として実行された処理に適切に課金することができます。

デフォルト値は "MQC.MQACT_NONE" です。

**applicationIdData**

`public String applicationIdData`

メッセージの識別コンテキストの一部。これは、アプリケーション・スイートによって定義される情報で、メッセージまたはそのメッセージの送信元に関する追加情報を提供するために使用することができます。

デフォルト値は "" です。

**putApplicationType**

`public int putApplicationType`

メッセージを書き込むアプリケーションのタイプ。これはシステム定義またはユーザー定義の値とすることができます。システムによって以下の値が定義されています。

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

デフォルト値は特殊値 MQC.MQAT_NO_CONTEXT で、これはコンテキスト情報がメッセージ中に存在していないことを指示します。

**putApplicationName**

`public String putApplicationName`

メッセージを書き込むアプリケーションの名前。デフォルト値は "" です。

**putDateTime**

`public GregorianCalendar putDateTime`

メッセージが書き込まれた時刻と日付。

**applicationOriginData**

```
public String applicationOriginData
```

アプリケーションによって定義されている情報で、メッセージの送信元に関する追加情報を提供するために使用することができます。

デフォルト値は "" です。

**groupId**

```
public byte[] groupId
```

物理メッセージが属しているメッセージ・グループを識別するバイト列。

デフォルト値は "MQC.MQGI_NONE" です。

**messageSequenceNumber**

```
public int messageSequenceNumber
```

グループ内の論理メッセージの順序番号。

**offset** `public int offset`

セグメント化されたメッセージでは、物理メッセージの先頭からの物理メッセージ中のデータのオフセット。

**messageFlags**

```
public int messageFlags
```

メッセージのセグメント化と状況を制御するフラグ。

**originalLength**

```
public int originalLength
```

セグメント化されたメッセージの元の長さ。

## コンストラクター

**MQMessage**

```
public MQMessage()
```

デフォルトのメッセージ記述子情報と空のメッセージ・バッファーを含む新規メッセージを作成します。

## メソッド

**getTotalMessageLength**

```
public int getTotalMessageLength()
```

このメッセージを検索した (検索しようとした) メッセージ・キューに保管されているときのメッセージの合計バイト数。 MQQueue.get() メソッドがメッセージ切り捨てエラー・コードで失敗すると、このメソッドはキュー上のメッセージの合計サイズを通知します。

148ページの『MQQueue.get』も参照してください。

**getMessageLength**

```
public int getMessageLength
```

IOException を投げます。

この MQMessage オブジェクト内のメッセージ・データのバイト数。

### getDataLength

```
public int getDataLength()
```

MQException を投げます。

読み取る残りのメッセージ・データのバイト数。

### seek

```
public void seek(int pos)
```

IOException を投げます。

*pos* を指定することによってカーソルをメッセージ・バッファー内の絶対位置に移動します。これ以降の読み取りおよび書き込みはバッファー内のこの位置から行われます。

pos がメッセージ・データ長の外側に指定されていると、EOFException が投げられます。

### setDataOffset

```
public void setDataOffset(int offset)
```

IOException を投げます。

カーソルをメッセージ・バッファー内の絶対位置に移動します。このメソッドは seek() の同義語で、他の MQSeries API との言語間の互換性のために提供されています。

### getDataOffset

```
public int getDataOffset()
```

IOException を投げます。

メッセージ・データ内の現行カーソル位置 (読み取り操作および書き込み操作が有効になるポイント) を戻します。

### clearMessage

```
public void clearMessage()
```

IOException を投げます。

メッセージ・バッファー内のデータをすべて廃棄し、データ・オフセットをゼロにリセットします。

### getVersion

```
public int getVersion()
```

使用中の構造体のバージョンを戻します。

### resizeBuffer

```
public void resizeBuffer(int size)
```

IOException を投げます。

以後の get 操作に必要となる可能性があるバッファー内のサイズに関する、MQMessage オブジェクトへのヒント。現在、メッセージにメッセージ・データが入っていて、新規サイズが現行サイズより小さくなっていると、そのメッセージ・データは切り捨てられます。

### readBoolean

```
public boolean readBoolean()
```

IOException を投げます。

メッセージ・バッファー内の現在位置から (符号付きの) 1 バイトを読み取ります。

### readChar

```
public char readChar()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から Unicode 文字を 1 文字読み取ります。

### readDouble

```
public double readDouble()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から倍精度を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_FLOAT_IEEE_NORMAL および MQC.MQENC_FLOAT_IEEE_REVERSED の値は、それぞれビッグ・エンディアン形式およびリトル・エンディアン形式の IEEE 標準倍精度を読み取ります。

MQC.MQENC_FLOAT_S390 の値は、システム/390 形式の浮動小数点数を読み取ります。

### readFloat

```
public float readFloat()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から浮動小数点を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_FLOAT_IEEE_NORMAL および MQC.MQENC_FLOAT_IEEE_REVERSED の値は、それぞれビッグ・エンディアン形式およびリトル・エンディアン形式の IEEE 標準浮動小数点を読み取ります。

MQC.MQENC_FLOAT_S390 の値は、システム/390 形式の浮動小数点数を読み取ります。

### readFully

```
public void readFully(byte b[])
```

Exception、EOFException を投げます。

バイト配列 b をメッセージ・バッファーからのデータで埋めます。

### readFully

```
public void readFully(byte b[],
                      int off,
                      int len)
```

IOException、EOFException を投げます。

バイト配列 b の *len* エレメントを、メッセージ・バッファー内のオフセット *off* から始まるデータで埋めます。

### readInt

```
public int readInt()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から整数を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン整数を読み取り、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン整数を読み取ります。

### readInt4

```
public int readInt4()
```

IOException、EOFException を投げます。

readInt() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### readLine

```
public String readLine()
```

IOException を投げます。

characterSet メンバー変数で識別されたコード・セットから Unicode に変換してから、 ¥n、¥r、¥r¥n、または EOF で区切られた行を読み取ります。

### readLong

```
public long readLong()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から long を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン long を読み取り、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン long を読み取ります。

### readInt8

```
public long readInt8()
```

IOException、EOFException を投げます。

readLong() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### readObject

```
public Object readObject()
```

OptionalDataException、ClassNotFoundException、IOException を投げます。

メッセージ・バッファーからオブジェクトを 1 つ読み取ります。オブジェクトのクラス、クラスのシグニチャー、およびクラスの非一時フィールドと非静的フィールドの値がすべて読み取られます。

### readShort

```
public short readShort()
```

IOException、EOFException を投げます。

### readInt2

```
public short readInt2()
```

IOException、EOFException を投げます。

readShort() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### readUTF

```
public String readUTF()
```

IOException を投げます。

メッセージ・バッファー内の現在位置から、 2 バイトの長さフィールドを接頭部として持つ UTF ストリングを読み取ります。

### readUnsignedByte

```
public int readUnsignedByte()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から符号なしのバイトを 1 バイト読み取ります。

### readUnsignedShort

```
public int readUnsignedShort()
```

IOException、EOFException を投げます。

メッセージ・バッファー内の現在位置から符号なしの short を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン符号なし short を読み取り、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン符号なし short を読み取ります。

### readUInt2

```
public int readUInt2()
```

IOException、EOFException を投げます。

readUnsignedShort() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### readString

```
public String readString(int length)
```

IOException、EOFException を投げます。

characterSet メンバー変数で識別されるコード・セットのストリングを読み取り、そのストリングを Unicode に変換します。

#### パラメーター

*length* 読み取る文字数 (文字当りに複数バイトを使用するコード・セットもあるので、コード・セットに従うバイト数とは異なる場合があります) 。

### readDecimal2

```
public short readDecimal2()
```

IOException、EOFException を投げます。

2 バイトのパック 10 進数 (-999..999) を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。
MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を読み取り、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を読み取ります。

### readDecimal4

```
public int readDecimal4()
```

IOException、EOFException を投げます。

4 バイトのパック 10 進数 (-9999999..9999999) を読み取ります。このメソッドの動作は、encoding メンバー変数の値によって決まります。
MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を読み取り、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を読み取ります。

### readDecimal8

```
public long readDecimal8()
```

IOException、EOFException を投げます。

8 バイトのパック 10 進数 (-999999999999999 ～ 999999999999999) を読み取ります。このメソッドの動作は、encoding メンバー変数によって決まります。 MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を読み取り、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を読み取ります。

### setVersion

```
public void setVersion(int version)
```

使用する構造体のバージョンを指定します。考えられる値は、次のとおりです。

- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

通常、バージョン 2 の構造体を処理できるキュー・マネージャーに接続している時に、クライアントで強制的にバージョン 1 の構造体を使用したい場合を除き、このメソッドを呼び出す必要はありません。これ以外のすべての状況では、クライアントは、キュー・マネージャーの能力を照会することによって、使用する構造体の正しいバージョンを判別します。

### skipBytes

```
public int skipBytes(int n)
```

IOException、EOFException を投げます。

メッセージ・バッファー内で前方に n バイト移動します。

このメソッドは、以下のいずれかが起きるまで呼び出しをブロックします。

- 全バイトがスキップされた
- メッセージ・バッファーの終わりが検出された
- 例外が投げられた

スキップされたバイト数を戻します。この値は常に n と一致します。

### write

```
public void write(int b)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に 1 バイトを書き込みます。

### write

```
public void write(byte b[])
```

IOException を投げます。

メッセージ・バッファー内の現在位置にバイトの配列を書き込みます。

### write

```
public void write(byte b[],
                  int off,
                  int len)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に一連のバイトを書き込みます。配列 b のオフセット *off* から *len* バイトが取られ、書き込まれます。

### writeBoolean

```
public void writeBoolean(boolean v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置にブール値を書き込みます。

### writeByte

```
public void writeByte(int v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に 1 バイトを書き込みます。

### writeBytes

```
public void writeBytes(String s)
```

IOException を投げます。

メッセージ・バッファーに一連のバイトとしてストリングを書き込みます。ストリング内のそれぞれの文字は、高位 8 ビットを廃棄することによって順番に書き込まれます。

### writeChar

```
public void writeChar(int v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に Unicode 文字を 1 文字書き込みます。

### writeChars

```
public void writeChars(String s)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に一連の Unicode 文字としてストリングを書き込みます。

### writeDouble

```
public void writeDouble(double v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に倍精度を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_FLOAT_IEEE_NORMAL および MQC.MQENC_FLOAT_IEEE_REVERSED の値は、それぞれビッグ・エンディアン形式およびリトル・エンディアン形式の IEEE 標準浮動小数点を書き込みます。

MQC.MQENC_FLOAT_S390 の値は、システム/390 形式の浮動小数点数を書き込みます。 IEEE 倍精度の値域は S/390® 倍精度浮動小数点数の値域より大きいので、非常に大きい数値は変換できないことに注意してください。

### writeFloat

```
public void writeFloat(float v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に浮動小数点を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_FLOAT_IEEE_NORMAL および MQC.MQENC_FLOAT_IEEE_REVERSED の値は、それぞれビッグ・エンディアン形式およびリトル・エンディアン形式の IEEE 標準浮動小数点を書き込みます。

MQC.MQENC_FLOAT_S390 の値は、システム/390 形式の浮動小数点数を書き込みます。

### writeInt

```
public void writeInt(int v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に整数を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン整数を書き込み、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン整数を書き込みます。

### writeInt4

```
public void writeInt4(int v)
```

IOException を投げます。

writeInt() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### writeLong

```
public void writeLong(long v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に long を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン long を書き込み、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン long を書き込みます。

### writeInt8

```
public void writeInt8(long v)
```

IOException を投げます。

writeLong() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### writeObject

```
public void writeObject(Object obj)
```

IOException を投げます。

指定のオブジェクトをメッセージ・バッファーに書き込みます。オブジェクトのクラス、クラスのシグニチャー、クラスおよびそのクラスの全スーパータイプの非一時フィールドと非静的フィールドの値がすべて書き込まれます。

### writeShort

```
public void writeShort(int v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に short を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_INTEGER_NORMAL の値はビッグ・エンディアン short を書き込み、 MQC.MQENC_INTEGER_REVERSED の値はリトル・エンディアン short を書き込みます。

### writeInt2

```
public void writeInt2(int v)
```

IOException を投げます。

writeShort() の同義語で、言語間の MQSeries API 互換性のために提供されています。

### writeDecimal2

```
public void writeDecimal2(short v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に 2 バイトのパック 10 進形式の数値を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を書き込み、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を書き込みます。

#### パラメーター

*v* -999 ～ 999 の範囲内とすることができます。

### writeDecimal4

```
public void writeDecimal4(int v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に 4 バイトのパック 10 進形式の数値を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を書き込み、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を書き込みます。

**パラメーター**

*v* -9999999 ～ 9999999 の範囲内とすることができます。

### writeDecimal8

```
public void writeDecimal8(long v)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に 8 バイトのパック 10 進形式の数値を書き込みます。このメソッドの動作は、encoding メンバー変数の値によって決まります。

MQC.MQENC_DECIMAL_NORMAL の値はビッグ・エンディアン・パック 10 進数を書き込み、 MQC.MQENC_DECIMAL_REVERSED の値はリトル・エンディアン・パック 10 進数を書き込みます。

**パラメーター**

*v* -999999999999999 ～ 999999999999999 の範囲内とすることができます。

### writeUTF

```
public void writeUTF(String str)
```

IOException を投げます。

メッセージ・バッファー内の現在位置に、 2 バイトの長さフィールドを接頭部として持つ UTF ストリングを書き込みます。

### writeString

```
public void writeString(String str)
```

IOException を投げます。

ストリングを characterSet メンバー変数で識別されるコード・セットに変換し、そのストリングをメッセージ・バッファー内の現在位置に書き込みます。

# MQMessageTracker

```
java.lang.Object
   |
   +-- com.ibm.mq.MQMessageTracker
```

public abstract class **MQMessageTracker**
extends **Object**

**注:** このクラスを使用できるのは、 MQSeries V5 (またはそれ以降) のキュー・マネージャーに接続している場合だけです。

このクラスは、MQDistributionListItem (99 ページ) に継承され、メッセージ・パラメーターを配布リスト内に指定された宛先用に調整するために使用されます。

## 変数

### feedback

```
public int feedback
```

レポートの性質を指示するために、タイプ MQC.MQMT_REPORT のメッセージで使用される変数。システムによって、以下のフィードバック・コードが定義されています。

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

MQC.MQFB_APPL_FIRST ～ MQC.MQFB_APPL_LAST の範囲内のアプリケーションで定義済みフィードバック値を使用することもできます。

このフィールドのデフォルト値は MQC.MQFB_NONE (フィードバックが提供されないことを指示する) です。

### messageId

```
public byte messageId[]
```

メッセージの書き込み時に使用するメッセージ ID。 MQC.MQMI_NONE が指定されていると、キュー・マネージャーはメッセージの書き込み時に一意のメッセージ ID を生成します。このメンバー変数の値は、使用したメッセージ ID を指示するために、書き込み後に更新されます。

デフォルト値は MQC.MQMI_NONE です。

### correlationId

```
public byte correlationId[]
```

メッセージの書き込み時に使用する相関 ID。

デフォルト値は MQC.MQCI_NONE です。

### accountingToken

```
public byte accountingToken[]
```

メッセージの識別コンテキストの一部。これにより、アプリケーションは、メッセージの結果として実行された処理に適切に課金することができます。

デフォルト値は MQC.MQACT_NONE です。

### groupId

```
public byte[] groupId
```

物理メッセージが属しているメッセージ・グループを識別するバイト列。

デフォルト値は MQC.MQGI_NONE です。

# MQPoolServices

```
java.lang.Object
   |
   └ com.ibm.mq.MQPoolServices
```

public class **MQPoolServices**
extends **Object**

**注:** 通常、アプリケーションはこのクラスを使用しません。

MQPoolServices クラスは、 MQSeries 接続のデフォルト ConnectionManager として使用される ConnectionManager のインプリメンテーションに使用できます。

ConnectionManager は、MQPoolServices オブジェクトを構成し、それを用いてリスナーを登録することができます。このリスナーは、MQEnvironment が管理する MQPoolTokens の集合に関連するイベントを受信します。この情報は、何らかの必要な始動や終結処理の作業を実行する際に、ConnectionManager で活用できます。

139ページの『MQPoolServicesEvent』と 171ページの『MQPoolServicesEventListener』も参照してください。

## コンストラクター

### MQPoolServices

```
public MQPoolServices()
```

新規 MQPoolServices オブジェクトを構成します。

## メソッド

### addMQPoolServicesEventListener

```
public void addMQPoolServicesEventListener
                                (MQPoolServicesEventListener listener)
```

MQPoolServicesEventListener を追加します。このリスナーは、MQEnvironment によって制御される MQPoolToken の集合でトークンの追加や除去が行われた場合や、デフォルトの ConnectionManager が変更された場合にイベントを受信します。

### removeMQPoolServicesEventListener

```
public void removeMQPoolServicesEventListener
                                (MQPoolServicesEventListener listener)
```

MQPoolServicesEventListener を除去します。

### getTokenCount

```
public int getTokenCount()
```

MQEnvironment に現在登録されている MQPoolToken の数を戻します。

# MQPoolServicesEvent

```
java.lang.Object
   |
   +-- java.util.EventObject
          |
          +-- com.ibm.mq.MQPoolServicesEvent
```

**注:** 通常、アプリケーションはこのクラスを使用しません。

MQPoolServicesEvent は、 MQEnvironment によって制御されている MQPoolToken の集合でトークンの追加や除去が行われた場合に生成されます。また、デフォルトの ConnectionManager が変更された場合にも、イベントは生成されます。

138ページの『MQPoolServices』と 171ページの『MQPoolServicesEventListener』も参照してください。

## 変数

**TOKEN_ADDED**

```
public static final int TOKEN_ADDED
```

MQPoolToken が集合に追加されるときに使用されるイベント ID。

**TOKEN_REMOVED**

```
public static final int TOKEN_REMOVED
```

MQPoolToken が集合から除去されるときに使用されるイベント ID。

**DEFAULT_POOL_CHANGED**

```
public static final int DEFAULT_POOL_CHANGED
```

デフォルトの ConnectionManager が変更されるときに使用されるイベント ID。

**ID** `protected int ID`

イベント ID。有効な値は、次のとおりです。

- TOKEN_ADDED
- TOKEN_REMOVED
- DEFAULT_POOL_CHANGED

**token** `protected MQPoolToken token`

トークン。イベント ID が DEFAULT_POOL_CHANGED のときは、ヌルになります。

## コンストラクター

**MQPoolServicesEvent**

```
public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)
```

イベント ID とトークンに基づいて MQPoolServicesEvent を構成します。

**MQPoolServicesEvent**

```
public MQPoolServicesEvent(Object source, int eid)
```

### MQPoolServicesEvent

イベント ID に基づいて MQPoolServicesEvent を構成します。

## メソッド

**getId** `public int getId()`

イベント ID を取得します。

**戻り**

以下のいずれかの値のイベント ID。

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

**getToken**

`public MQPoolToken getToken()`

集合で追加または削除されたトークンを戻します。イベント ID が DEFAULT_POOL_CHANGED の場合は、ヌルになります。

# MQPoolToken

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPoolToken
```

public class **MQPoolToken**
extends **Object**

MQPoolToken は、デフォルト接続プールの使用可能化に使用できます。この MQPoolToken は、アプリケーション・コンポーネントが MQSeries に接続する前に MQEnvironment クラスに登録され、コンポーネントが MQSeries の使用を終えると登録解除されます。通常、デフォルトの ConnectionManager は、登録された MQPoolToken の集合が空になるまでアクティブになっています。

MQPoolToken には、メソッドや変数がありません。 ConnectionManager のプロバイダーは、MQPoolToken を拡張することによって、 ConnectionManager にヒントを渡せるようにすることもできます。

106ページの『MQEnvironment.addConnectionPoolToken』および 106ページの『MQEnvironment.removeConnectionPoolToken』を参照してください。

## コンストラクター

### MQPoolToken

```
public MQPoolToken()
```

新規 MQPoolToken オブジェクトを構成します。

# MQProcess

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQProcess
```

public class **MQProcess**
extends **MQManagedObject** (113 ページを参照してください。)

MQProcess は MQSeries プロセスの照会操作を提供します。

## コンストラクター

### MQProcess

```
public MQProcess(MQQueueManager qMgr,
                 String processName,
                 int openOptions,
                 String queueManagerName,
                 String alternateUserId)
          throws MQException
```

キュー・マネージャー qMgr のプロセスにアクセスします。残りのパラメーターの詳細については、156ページの『MQQueueManager』の accessProcess を参照してください。

## メソッド

### getApplicationId

```
public String getApplicationId()
```

開始するアプリケーションを識別する文字ストリング。この情報は、開始キューのメッセージを処理するトリガー・モニター・アプリケーションが使用するためのものです。また、この情報はトリガー・メッセージの一部として開始キューに送信されます。

プロセスのクローズ後にこのメソッドを呼び出すと、MQException が投げられます。

### getApplicationType

```
public int getApplicationType()
```

例外を投げます (107 ページを参照してください)。

これは、トリガー・メッセージの受信に対する応答で開始されるプログラムの性質を識別します。アプリケーション・タイプには、任意の値を取ることができますが、以下の値を標準タイプとしてお勧めします。

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400

- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT
- MQC.MWQAT_USER_FIRST (ユーザー定義アプリケーション・タイプの最低値)
- MQC.MQAT_USER_LAST (ユーザー定義アプリケーション・タイプの最高値)

### getEnvironmentData

```
public String getEnvironmentData()
```

MQException を投げます。

開始するアプリケーションに関係のある環境情報が入っているストリング。

### getUserData

```
public String getUserData()
```

MQException を投げます。

開始するアプリケーションに関係のあるユーザー情報が入っているストリング。

### close

```
public synchronized void close()
```

MQException を投げます。

115ページの『MQManagedObject.close』 の指定変更。

# MQPutMessageOptions

```
java.lang.Object
   |
   +-- com.ibm.mq.MQPutMessageOptions
```

public class **MQPutMessageOptions**
extends **Object**

このクラスには、MQQueue.put() の動作を制御するオプションが含まれています。

**注:** このクラスで使用可能な一部のオプションの動作は、使用される環境によって異なります。これらのエレメントには * のマークが付けられています。詳細については、87ページの『他の環境で操作されるバージョン 5 拡張機能』を参照してください。

## 変数

### options

```
public int options
```

MQQueue.put のアクションを制御するオプション。次の値のどれかを指定しても、どれも指定しなくてもかまいません。複数のオプションが必要な場合は、これらの値をまとめて追加するか、ビット単位の OR 演算子を使用して結合することができます。

**MQC.MQPMO_SYNCPOINT**
メッセージを同期点制御を用いて書き込みます。このメッセージは、作業単位がコミットされるまで、その作業単位の外側では表示されません。作業単位がバックアウトされると、このメッセージは削除されます。

**MQC.MQPMO_NO_SYNCPOINT**
メッセージを同期点制御なしに書き込みます。なお、同期点制御オプションが指定されない場合は、デフォルトの 'no syncpoint' が想定されます。これは、OS/390 を含め、すべてのプラットフォームに適用されます。

**MQC.MQPMO_NO_CONTEXT**
コンテキストをメッセージと関連付けません。

**MQC.MQPMO_DEFAULT_CONTEXT**
デフォルト・コンテキストをメッセージと関連付けます。

**MQC.MQPMO_SET_IDENTITY_CONTEXT**
識別コンテキストをアプリケーションから設定します。

**MQC.MQPMO_SET_ALL_CONTEXT**
すべてのコンテキストをアプリケーションから設定します。

**MQC.MQPMO_FAIL_IF_QUIESCING**
キュー・マネージャーが静止している場合には、失敗します。

**MQC.MQPMO_NEW_MSG_ID***
送信メッセージごとに新規メッセージ ID を生成します。

**MQC.MQPMO_NEW_CORREL_ID***
送信メッセージごとに新規相関 ID を生成します。

**MQC.MQPMO_LOGICAL_ORDER***
メッセージ・グループ内の論理メッセージおよびセグメントを論理順序にします。

**MQC.MQPMO_NONE**
指定するオプションがありません。他のオプションと組み合わせて使用しないでください。

**MQC.MQPMO_PASS_IDENTITY_CONTEXT**
識別コンテキストを入力キュー・ハンドルから渡します。

**MQC.MQPMO_PASS_ALL_CONTEXT**
すべてのコンテキストを入力キュー・ハンドルから渡します。

### contextReference

```
public MQQueue ContextReference
```

これはコンテキスト情報のソースを指示する入力フィールドです。

options フィールドに MQC.MQPMO_PASS_IDENTITY_CONTEXT または MQC.MQPMO_PASS_ALL_CONTEXT が含まれている場合には、このフィールドを設定して、コンテキスト情報のソースとなる MQQueue を参照してください。

このフィールドの初期値は null です。

### recordFields *

```
public int recordFields
```

メッセージを配布リストに書き込む時に、キューごとにカスタマイズするフィールドを指示するフラグ。以下の 1 つまたは複数のフラグを指定することができます。

**MQC.MQPMRF_MSG_ID**
MQDistributionListItem の messageId 属性を使用します。

**MQC.MQPMRF_CORREL_ID**
MQDistributionListItem の correlationId 属性を使用します。

**MQC.MQPMRF_GROUP_ID**
MQDistributionListItem の groupId 属性を使用します。

**MQC.MQPMRF_FEEDBACK**
MQDistributionListItem の feedback 属性を使用します。

**MQC.MQPMRF_ACCOUNTING_TOKEN**
MQDistributionListItem の accountingToken 属性を使用します。

特殊値 MQC.MQPMRF_NONE は、フィールドをカスタマイズしないことを指示します。

### resolvedQueueName

```
public String resolvedQueueName
```

これは、キュー・マネージャーによって、メッセージが入っているキューの名前に設定される出力フィールドです。これは、オープンされているキューが別名キューまたはモデル・キューの場合には、キューのオープンに使用した名前とは異なる場合があります。

### resolvedQueueManagerName

```
public String resolvedQueueManagerName
```

これは、キュー・マネージャーによって、リモート・キュー名で指定されたキューを所有しているキュー・マネージャーの名前に設定される出力フィールドです。これは、キューがリモート・キューの場合には、アクセス元のキュー・マネージャーの名前とは異なる場合があります。

### knownDestCount *

```
public int knownDestCount
```

これは、キュー・マネージャーによって、現行呼び出しがローカル・キューに解決されるキューに対して正常に送信を完了したメッセージ数に設定される出力フィールドです。このフィールドは、配布リストの一部ではない単一キューのオープン時にも設定されます。

### unknownDestCount *

```
public int unknownDestCount
```

これは、キュー・マネージャーによって、現行呼び出しがリモート・キューに解決されるキューに対して正常に送信を完了したメッセージ数に設定される出力フィールドです。このフィールドは、配布リストの一部ではない単一キューのオープン時にも設定されます。

### invalidDestCount *

```
public int invalidDestCount
```

これは、キュー・マネージャーによって、配布リスト内のキューに送信できなかったメッセージ数に設定される出力フィールドです。このカウントには、正常にオープンされたが put 操作が失敗したキューの他に、オープンが失敗したキューも含まれています。このフィールドは、配布リストの一部ではない単一キューのオープン時にも設定されます。

## コンストラクター

### MQPutMessageOptions

```
public MQPutMessageOptions()
```

オプションが設定されていない新規 MQPutMessageOptions オブジェクトと、ブランクの resolvedQueueName および resolvedQueueManagerName を構成します。

# MQQueue

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueue
```

public class **MQQueue**
extends **MQManagedObject**
(113 ページを参照してください。)

MQQueue は、MQSeries キューについての inquire、set、put、および get 操作を提供します。 inquire および set 機能は、MQ.MQManagedObject から継承します。

161ページの『MQQueueManager.accessQueue』も参照してください。

## コンストラクター

### MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
               String alternateUserId )
               throws MQException
```

キュー・マネージャー qMgr のキューにアクセスします。

残りのパラメーターの詳細については、161ページの『MQQueueManager.accessQueue』を参照してください。

## メソッド

### get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

MQException を投げます。

指定の最大メッセージ・サイズまでのメッセージをキューから検索します。

このメソッドはパラメーターとして MQMessage オブジェクトを取ります。これは、入力パラメーターとしてオブジェクト内の一部のフィールド (特に、 messageId および correlationId) を使用するので、これらが要求通りに確実に設定されていることが重要になります。 (289ページの『Message』を参照してください。)

get が失敗すると、MQMessage オブジェクトは未変更のままです。成功した場合には、メッセージ記述子 (メンバー変数) および MQMessage のメッセージ・データ部分が、着信メッセージからのメッセージ記述子およびメッセージ・データによって完全に置き換えられます。

指定の MQQueueManager から MQSeries への呼び出しはすべて同期していることに注意してください。したがって、待機ありの get を実行すると、同じ MQQueueManager を使用している他のスレッドはすべて、その get が完了するまで、それ以降の MQSeries 呼び出しを実行できないようにブロックされます。同時に MQSeries にアクセスする複数のスレッドが必要な場合には、スレッドごとに固有の MQQueueManager オブジェクトを作成しなければなりません。

### パラメーター

*message*
メッセージ記述子情報および戻されたメッセージ・データが入る入出力パラメーター。

*getMessageOptions*
get のアクションを制御するオプション。 (109ページの『MQGetMessageOptions』を参照してください。)

*MaxMsgSize*
この呼び出しで受信できる最大メッセージ。キューのメッセージがこのサイズより大きいと、次のうちの 1 つが発生する可能性があります。

1. MQC.MQGMO_ACCEPT_TRUNCATED_MSG フラグが、MQGetMessageOptions オブジェクトの options メンバー変数に設定されていると、メッセージは指定のバッファー・サイズに収まるだけメッセージ・データで埋められ、完了コード MQException.MQCC_WARNING および理由コード MQException.MQRC_TRUNCATED_MSG_ACCEPTED の例外が投げられます。
2. MQC.MQGMO_ACCEPT_TRUNCATED_MSG フラグが設定されていないと、メッセージはキュー上に残り、完了コード MQException.MQCC_WARNING および理由コード MQException.MQRC_TRUNCATED_MSG_FAILED の MQException が発生します。

get が失敗した場合は、MQException が投げられます。

## get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

MQException を投げます。

メッセージのサイズとは無関係に、キューからメッセージを検索します。大きいメッセージの場合、ユーザーに代り get メソッドが MQSeries への 2 つの呼び出し (1 つは必要なバッファー・サイズを設定するための呼び出し、もう 1 つはメッセージ・データそのものを読み取るための呼び出し) を発行することが必要な場合もあります。

このメソッドはパラメーターとして MQMessage オブジェクトを取ります。これは、入力パラメーターとしてオブジェクト内の一部のフィールド (特

に、 messageId および correlationId) を使用するので、これらが要求通りに確実に設定されていることが重要になります。 (289ページの『Message』を参照してください。)

get が失敗すると、MQMessage オブジェクトは未変更のままです。成功した場合には、メッセージ記述子 (メンバー変数) および MQMessage のメッセージ・データ部分が、着信メッセージからのメッセージ記述子およびメッセージ・データによって完全に置き換えられます。

指定の MQQueueManager から MQSeries への呼び出しはすべて同期していることに注意してください。したがって、待機ありの get を実行すると、同じ MQQueueManager を使用している他のスレッドはすべて、その get が完了するまで、それ以降の MQSeries 呼び出しを実行できないようにブロックされます。同時に MQSeries にアクセスする複数のスレッドが必要な場合には、スレッドごとに固有の MQQueueManager オブジェクトを作成しなければなりません。

**パラメーター**

*message*
メッセージ記述子情報および戻されたメッセージ・データが入る入出力パラメーター。

*getMessageOptions*
get のアクションを制御するオプション。 (詳細については、109ページの『MQGetMessageOptions』を参照してください。)

get が失敗した場合は、MQException が投げられます。

### get

```
public synchronized void get(MQMessage message)
```

これは、前述の get メソッドの単純化されたバージョンです。

**パラメーター**

*MQMessage*
メッセージ記述子情報および戻されたメッセージ・データが入る入出力パラメーター。

このメソッドは、get を実行するために MQGetMessageOptions のデフォルト・インスタンスを使用します。使用されるメッセージ・オプションは MQGMO_NOWAITです。

### put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions)
```

MQException を投げます。

メッセージをキューに入れます。

このメソッドはパラメーターとして MQMessage オブジェクトを取ります。このオブジェクトのメッセージ記述子プロパティーはこのメソッドの結果として更新される場合があります。このメソッドの完了直後の値が MQSeries キューに書き込まれた値になります。

put が完了した後の MQMessage オブジェクトへの変更は、MQSeries キューの実際のメッセージには影響しません。

put を実行すると、messageId と correlationId が更新されます。それで、これ以降に同じ MQMessage オブジェクトを使用して put/get への呼び出しを行う場合には、必ずこのことを考慮してください。また、put を呼び出してもメッセージ・データは消去されないので、

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

このようにすると、2 つのメッセージが書き込まれます。最初のメッセージには ″a″ が入り、2 番目には ″ab″ が入ります。

**パラメーター**

*message*
メッセージ記述子データおよび送信されるメッセージが入るメッセージ・バッファー。

*putMessageOptions*
put のアクションを制御するオプション。 (144ページの『MQPutMessageOptions』を参照してください。)

put が失敗した場合は、MQException が投げられます。

### put

```
public synchronized void put(MQMessage message)
```

これは、前述の put メソッドの単純化されたバージョンです。

**パラメーター**

*MQMessage*
メッセージ記述子データおよび送信されるメッセージが入るメッセージ・バッファー。

このメソッドは、put を実行するために MQPutMessageOptions のデフォルト・インスタンスを使用します。

**注:** キューをクローズした後にこのメソッドを呼び出すと、以降のメソッドはすべて MQException を投げます。

### getCreationDateTime

```
public GregorianCalendar getCreationDateTime()
```

MQException を投げます。

このキューが作成された日時。

### getQueueType

```
public int getQueueType()
```

MQException を投げます。

**戻り** 以下の値の 1 つを含む、このキューのタイプ。

- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

### getCurrentDepth

```
public int getCurrentDepth()
```

MQException を投げます。

現在キュー上にあるメッセージの数を取得します。この値は、put 呼び出し時および get 呼び出しのバックアウト時に増加します。また、非ブラウズ get 時および put 呼び出しのバックアウト時に減少します。

### getDefinitionType

```
public int getDefinitionType()
```

MQException を投げます。

キューの定義方法を指示します。

**戻り** 以下のいずれか。

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

### getMaximumDepth

```
public int getMaximumDepth()
```

MQException を投げます。

任意のある時点でキュー上に存在できるメッセージの最大数。すでにこの数のメッセージが入っているキューにメッセージを書き込もうとすると、理由コード MQException.MQRC_Q_FULL で失敗します。

### getMaximumMessageLength

```
public int getMaximumMessageLength()
```

MQException を投げます。

このキューの各メッセージ内に存在できるアプリケーション・データの最大長。この値より大きいメッセージを書き込もうとすると、理由コード MQException.MQRC_MSG_TOO_BIG_FOR_Q で失敗します。

### getOpenInputCount

```
public int getOpenInputCount()
```

MQException を投げます。

メッセージをキューから除去するために現在有効になっているハンドルの数。これは、MQSeries classes for Java によって (accessQueue を使用して) 作成したばかりのハンドルではなく、ローカル・キュー・マネージャーに認識されているハンドルの*合計* 数です。

### getOpenOutputCount

```
public int getOpenOutputCount()
```

MQException を投げます。

メッセージをキューに追加するために現在有効になっているハンドルの数。これは、MQSeries classes for Java によって (accessQueue を使用して) 作成したばかりのハンドルではなく、ローカル・キュー・マネージャーに認識されているハンドルの*合計* 数です。

### getShareability

```
public int getShareability()
```

MQException を投げます。

キューが入力用に何回もオープンできるかどうかを指示します。

**戻り** 以下のいずれか。

- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

### getInhibitPut

```
public int getInhibitPut()
```

MQException を投げます。

put 操作をこのキューに許可するかどうかを指示します。

**戻り** 以下のいずれか。

- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

### setInhibitPut

```
public void setInhibitPut(int inhibit)
```

MQException を投げます。

put 操作をこのキューに許可するかどうかを制御します。暗黙的値は次のとおりです。

- MQC.MQQA_PUT_INHIBITED
- MQC.MQQA_PUT_ALLOWED

### getInhibitGet

```
public int getInhibitGet()
```

MQException を投げます。

get 操作をこのキューに許可するかどうかを指示します。

**戻り** 考えられる値は、次のとおりです。

- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

### setInhibitGet

```
public void setInhibitGet(int inhibit)
```

MQException を投げます。

get 操作をこのキューに許可するかどうかを制御します。暗黙的値は次のとおりです。

- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

### getTriggerControl

```
public int getTriggerControl()
```

MQException を投げます。

アプリケーションがキューのサービスを開始するように、トリガー・メッセージを開始キューに書き込むかどうかを指示します。

**戻り** 考えられる値は、次のとおりです。

- MQC.MQTC_OFF
- MQC.MQTC_ON

### setTriggerControl

```
public void setTriggerControl(int trigger)
```

MQException を投げます。

アプリケーションがキューのサービスを開始するように、トリガー・メッセージを開始キューに書き込むかどうかを制御します。暗黙的値は次のとおりです。

- MQC.MQTC_OFF
- MQC.MQTC_ON

### getTriggerData

```
public String getTriggerData()
```

MQException を投げます。

このキューへのメッセージの到着によって、トリガー・メッセージが開始キューに書き込まれる時に、キュー・マネージャーがそのトリガー・メッセージに挿入する自由形式のデータ。

### setTriggerData

```
public void setTriggerData(String data)
```

MQException を投げます。

このキューへのメッセージの到着によって、トリガー・メッセージが開始キューに書き込まれる時に、キュー・マネージャーがそのトリガー・メッセージに挿入する自由形式のデータを設定します。ストリングの許容最大長は MQC.MQ_TRIGGER_DATA_LENGTH に指定されています。

### getTriggerDepth

```
public int getTriggerDepth()
```

MQException を投げます。

トリガー・タイプが MQC.MQTT_DEPTH に設定されている時に、トリガー・メッセージが書き込まれる前に、キューに存在していなければならないメッセージの数。

### setTriggerDepth

```
public void setTriggerDepth(int depth)
```

MQException を投げます。

トリガー・タイプが MQC.MQTT_DEPTH に設定されている時に、トリガー・メッセージが書き込まれる前に、キューに存在していなければならないメッセージの数を設定します。

### getTriggerMessagePriority

```
public int getTriggerMessagePriority()
```

MQException を投げます。

メッセージがトリガー・メッセージの生成に寄与しないメッセージ優先順位 (すなわち、キュー・マネージャーは、トリガー・メッセージを生成するかどうかを決める時にこれらのメッセージを無視します)。値をゼロにすると、すべてのメッセージはトリガー・メッセージの生成に寄与することになります。

### setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority)
```

MQException を投げます。

メッセージがトリガー・メッセージの生成に寄与しないメッセージ優先順位を設定します (すなわち、キュー・マネージャーは、トリガー・メッセージを生成するかどうかを決める時にこれらのメッセージを無視します)。値をゼロにすると、すべてのメッセージはトリガー・メッセージの生成に寄与することになります。

### getTriggerType

```
public int getTriggerType()
```

MQException を投げます。

このキューにメッセージが到着した結果としてトリガー・メッセージが書き込まれる条件。

**戻り** 考えられる値は、次のとおりです。

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

### setTriggerType

```
public void setTriggerType(int type)
```

MQException を投げます。

このキューにメッセージが到着した結果としてトリガー・メッセージが書き込まれる条件を設定します。考えられる値は、次のとおりです。

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

### close

```
public synchronized void close()
```

MQException を投げます。

115ページの『MQManagedObject.close』 の指定変更。

# MQQueueManager

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueueManager
```

public class **MQQueueManager**
extends **MQManagedObject**
(113 ページを参照してください。)

**注:** このクラスで使用可能な一部のオプションの動作は、使用される環境によって異なります。これらのエレメントには * のマークが付けられています。詳細は、83ページの『第8章 環境による動作の違い』を参照してください。

## 変数

### isConnected

```
public boolean isConnected
```

キュー・マネージャーへの接続がまだオープンされている場合は真。

## コンストラクター

### MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

MQException を投げます。

指定のキュー・マネージャーへの接続を作成します。

**注:** MQSeries classes for Java を使用している場合は、接続要求時に使用するホスト名、チャネル名、およびポートを MQEnvironment クラスに指定します。これは、このコンストラクターを呼び出す*前に* 実行しなければなりません。

次の例は、hostname fred.mq.com というマシンで実行中のキュー・マネージャー ″MYQM″ への接続を示しています。

```
MQEnvironment.hostname = "fred.mq.com";  // 接続先のホスト
MQEnvironment.port     = 1414;           // 接続先のポート
                                         // これが設定されない場合は、デ
                                         // フォルトで 1414 (デフォルトの
                                         // MQSeries ポート) が使用されます
MQEnvironment.channel  = "channel.name"; // キュー・マネージャー上の
                                         // SVR CONN チャネルの名前
                                         // (大文字と小文字を区別して
                                         // ください)
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

キュー・マネージャー名がブランク (null または ″″) になっていると、接続はデフォルト・キュー・マネージャーに対して行われます。

101ページの『MQEnvironment』も参照してください。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

MQException を投げます。

このコンストラクターは、MQEnvironment にあるプロパティーを使用して、指定されたキュー・マネージャーに接続します。接続は、指定された MQConnectionManager によって管理されます。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

MQException を投げます。

このコンストラクターは、MQEnvironment にあるプロパティーを使用して、指定されたキュー・マネージャーに接続します。接続は、指定された ConnectionManager によって管理されます。

このメソッドには、JAAS 1.0 以降がインストールされた、 Java 2 v1.3 以降の JVM が必要です。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

MQException を投げます。

このバージョンのコンストラクターは、バインディング・モード専用で、キュー・マネージャーに接続する場合に拡張接続 API (MQCONNX) を使用します。 *options* パラメーターによって、高速バインドまたは通常バインドのどちらかを選択することができます。以下の値を指定できます。

- MQC.MQCNO_FASTPATH_BINDING (高速バインドの場合) *****
- MQC.MQCNO_STANDARD_BINDING (通常バインドの場合)

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

MQException を投げます。

このコンストラクターは、指定されたオプションを渡して MQCONNX を実行します。接続は、指定された MQConnectionManager によって管理されます。

### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

MQException を投げます。

### MQQueueManager

このコンストラクターは、指定されたオプションを渡して MQCONNX を実行します。接続は、指定された ConnectionManager によって管理されます。

このメソッドには、JAAS 1.0 以降がインストールされた、 Java 2 v1.3 以降の JVM が必要です。

#### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      java.util.Hashtable properties)
```

properties パラメーターは、この特定キュー・マネージャーの MQSeries 環境を記述している一連のキー / 値のペアを取ります。これらのプロパティーが指定されると、MQEnvironment クラスによって設定された値が指定変更され、個々のプロパティーをキュー・マネージャーごとに設定できるようにします。 103ページの『″MQEnvironment.properties″』を参照してください。

#### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

MQException を投げます。

このコンストラクターは、指定されたプロパティーの Hashtable で MQEnvironment の設定を指定変更して、指定されたキュー・マネージャーに接続します。接続は、指定された MQConnectionManager によって管理されます。

#### MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

MQException を投げます。

このコンストラクターは、指定されたプロパティーの Hashtable で MQEnvironment の設定を指定変更して、指定されたキュー・マネージャーに接続します。接続は、指定された ConnectionManager によって管理されます。

このメソッドには、JAAS 1.0 以降がインストールされた、 Java 2 v1.3 以降の JVM が必要です。

## メソッド

#### getCharacterSet

```
public int getCharacterSet()
```

MQException を投げます。

キュー・マネージャーのコード・セットの CCSID (コード化文字セット ID) を戻します。これは、キュー・マネージャーがアプリケーション・プログラミング・インターフェースの文字ストリング・フィールドすべてに使用する文字セットを定義します。

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

### getMaximumMessageLength

```
public int getMaximumMessageLength()
```

MQException を投げます。

キュー・マネージャーが処理できるメッセージの最大長 (バイト) を戻します。最大メッセージ長がこれより大きいキューは定義することができません。

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

### getCommandLevel

```
public int getCommandLevel()
```

MQException を投げます。

キュー・マネージャーがサポートするシステム制御コマンドのレベルを指示します。特定のコマンド・レベルと対応しているシステム制御コマンドのセットは、キュー・マネージャーが実行中のプラットフォームのアーキテクチャーによって異なります。詳細については、ご使用のプラットフォームの MQSeries 資料を参照してください。

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

**戻り** MQC.MQCMDL_LEVEL_xxx 定数のいずれか。

### getCommandInputQueueName

```
public String getCommandInputQueueName()
```

MQException を投げます。

キュー・マネージャーで定義されているコマンド入力キューの名前を戻します。これは、アプリケーションが、許可されている場合に、コマンドを送信できる宛先のキューです。

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

### getMaximumPriority

```
public int getMaximumPriority()
```

MQException を投げます。

キュー・マネージャーがサポートする最高メッセージ優先順位を戻します。優先順位の範囲はゼロ最低値からこの値までです。

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

### getSyncpointAvailability

```
public int getSyncpointAvailability()
```

MQException を投げます。

キュー・マネージャーが作業単位と、 MQQueue.get および MQQueue.put メソッドとの同期点処理をサポートするかどうかを指示します。

**戻り**

- MQC.MQSP_AVAILABLE (同期点処理が使用可能な場合)
- MQC.MQSP_NOT_AVAILABLE (同期点処理が使用不能な場合)

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

### getDistributionListCapable

```
public boolean getDistributionListCapable()
```

キュー・マネージャーが配布リストをサポートするかどうかを指示します。

### disconnect

```
public synchronized void disconnect()
```

MQException を投げます。

キュー・マネージャーへの接続を終了します。このキュー・マネージャーからアクセスしたオープン・キューやプロセスはすべてクローズされ、それ以後は使用不能になります。キュー・マネージャーから切断した場合、再接続する唯一の方法は新しい MQQueueManager オブジェクトを作成することです。

通常、作業単位の一環として実行された作業は、すべてコミットされます。しかし、接続が MQConnectionManager ではなく ConnectionManager によって管理されている場合は、作業単位がロールバックされることがあります。

### commit

```
public synchronized void commit()
```

MQException を投げます。

このメソッドを呼び出すと、アプリケーションが同期点に達していること、および最後の同期点が永続化されてから発生したすべてのメッセージの読み取りと書き込みがキュー・マネージャーに指示されます。作業単位の一環として (MQC.MQPMO_SYNCPOINT フラグを MQPutMessageOptions の options フィールドに設定することで) 書き込まれたメッセージは、他のアプリケーションで使用可能になります。作業単位の一環として (MQC.MQGMO_SYNCPOINT フラグを MQGetMessageOptions の options フィールドに設定することで) 検索されたメッセージは、削除されます。

後述の ″backout″ の説明も参照してください。

### backout

```
public synchronized void backout()
```

MQException を投げます。

このメソッドを呼び出すと、最後の同期点がバックアウトされてから発生したすべてのメッセージの読み取りと書き込みがキュー・マネージャーに指示されます。作業単位の一環として (MQC.MQPMO_SYNCPOINT フラグを MQPutMessageOptions の options フィールドに設定することで) 書き込まれたメッセージは削除され、作業単位の一環として (MQC.MQGMO_SYNCPOINT フラグを MQGetMessageOptions の options フィールドに設定することで) 検索されたメッセージはキュー上に復元されます。

前述の「commit」の説明も参照してください。

### accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName, int openOptions,
                        String queueManagerName,
                        String dynamicQueueName,
                        String alternateUserId
                        )
```

MQException を投げます。

メッセージの読み取り/ブラウズ、メッセージの書き込み、キューの属性についての照会、あるいはキューの属性の設定のための、このキュー・マネージャーでの MQSeries キューへのアクセス権を設定します。

指定されたキューがモデル・キューの場合は、動的ローカル・キューが作成されます。作成済みのキューの名前は、戻された MQQueue オブジェクトの name 属性を調べることによって判別できます。

#### パラメーター

*queueName*
: オープンするキューの名前。

*openOptions*
: キューのオープンを制御するオプション。有効なオプションは、次のとおりです。

**MQC.MQOO_BROWSE**
: メッセージをブラウズするためにオープンします。

**MQC.MQOO_INPUT_AS_Q_DEF**
: キュー定義のデフォルトを使用してメッセージを読み取るためにオープンします。

**MQC.MQOO_INPUT_SHARED**
: 共用アクセスによってメッセージを読み取るためにオープンします。

**MQC.MQOO_INPUT_EXCLUSIVE**
排他アクセスによってメッセージを読み取るためにオープンします。

**MQC.MQOO_OUTPUT**
メッセージを書き込むためにオープンします。

**MQC.MQOO_INQUIRE**
照会用にオープンします - プロパティーを照会したい場合に必要です。

**MQC.MQOO_SET**
属性を設定するためにオープンします。

**MQC.MQOO_SAVE_ALL_CONTEXT**
メッセージを検索した時にコンテキストを保管します*。

**MQC.MQOO_SET_IDENTITY_CONTEXT**
識別コンテキストの設定を許可します。

**MQC.MQOO_SET_ALL_CONTEXT**
すべてのコンテキストの設定を許可します。

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**
指定のユーザー ID を用いて妥当性検査します。

**MQC.MQOO_FAIL_IF_QUIESCING**
キュー・マネージャーが静止している場合には、失敗します。

**MQC.MQOO_BIND_AS_QDEF**
デフォルトのバインドをキューに使用します。

**MQC.MQOO_BIND_ON_OPEN**
キューのオープン時にハンドルを宛先にバインドします。

**MQC.MQOO_BIND_NOT_FIXED**
指定の宛先にバインドしません。

**MQC.MQOO_PASS_ALL_CONTEXT**
すべてのコンテキストを渡すことができます。

**MQC.MQOO_PASS_IDENTITY_CONTEXT**
識別コンテキストを渡すことができます。

複数のオプションが必要な場合は、これらの値をまとめて追加するか、ビット単位の OR 演算子を使用して結合することができます。これらのオプションについて詳しくは、MQSeries *MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

*queueManagerName*
キューが定義されているキュー・マネージャーの名前。全体にブランクになっている名前や、null になっている名前は、MQQueueManager オブジェクトの接続先のキュー・マネージャーを指示します。

*dynamicQueueName*
queueName でモデル・キューの名前が指定されていない限り、この

パラメーターは無視されます。モデル・キューの名前が指定されている場合は、このパラメーターに作成する動的キューの名前を指定します。 queueName でモデル・キューの名前が指定されていても、ブランク名や空名は無効になります。名前の最後のブランクでない文字がアスタリスク (*) になっていると、キュー・マネージャーは、そのアスタリスクを、キュー用に生成された名前がこのキュー・マネージャーで一意となることを保証する文字のストリングで置き換えます。

*alternateUserId*

MQOO_ALTERNATE_USER_AUTHORITY が openOptions パラメーターに指定されている場合は、このパラメーターに、オープンの許可を調べるために使用する代替ユーザー ID を指定します。MQOO_ALTERNATE_USER_AUTHORITY が指定されていない場合、このパラメーターはブランク (または null) にしてもかまいません。

**戻り** 正常にオープンされた MQQueue

オープンが失敗した場合は、MQException が投げられます。

『″accessProcess″』も参照してください。

### accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName,
                        int openOptions
                        )
```

キュー・マネージャーからの切断後にこのメソッドを呼び出すと、MQException が投げられます。

**パラメーター**

*queueName*

オープンするキューの名前

*openOptions*

キューのオープンを制御するオプション

パラメーターの詳細については、161ページの『MQQueueManager.accessQueue』を参照してください。

*queueManagerName*、*dynamicQueueName*、および *alternateUserId* は "" に設定されます。

### accessProcess

```
public synchronized MQProcess accessProcess
                         (
                        String processName,
                        int openOptions,
                        String queueManagerName,
                        String alternateUserId
                        )
```

MQException を投げます。

process 属性についての照会のための、このキュー・マネージャーでの MQSeries プロセスへのアクセス権を設定します。

**パラメーター**

*processName*
オープンするプロセスの名前。

*openOptions*
プロセスのオープンを制御するオプション。照会は自動的に指定のオプションに追加されるので、明示的に指定する必要はありません。

有効なオプションは、次のとおりです。

**MQC.MQOO_ALTERNATE_USER_AUTHORITY**
指定のユーザー ID を用いて妥当性検査します

**MQC.MQOO_FAIL_IF_QUIESCING**
キュー・マネージャーが静止している場合には失敗します

複数のオプションが必要な場合は、これらの値をまとめて追加するか、ビット単位の OR 演算子を使用して結合することができます。これらのオプションについて詳しくは、*MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

*queueManagerName*
プロセスが定義されているキュー・マネージャーの名前。アプリケーションでは、このパラメーターをブランクまたは null のままにしておく必要があります。

*alternateUserId*
MQOO_ALTERNATE_USER_AUTHORITY が openOptions パラメーターに指定されている場合は、このパラメーターに、オープンの許可を調べるために使用する代替ユーザー ID を指定します。MQOO_ALTERNATE_USER_AUTHORITY が指定されていない場合、このパラメーターはブランク (または null) にしてもかまいません。

**戻り** 正常にオープンされた MQProcess。

オープンが失敗した場合は、MQException が投げられます。

161ページの『MQQueueManager.accessQueue』も参照してください。

### accessProcess

これは、前述の AccessProcess メソッドの単純化されたバージョンです。

```
public synchronized MQProcess accessProcess
                      (
                     String processName,
                     int openOptions
                     )
```

これは、前述の AccessQueue メソッドの単純化されたバージョンです。

**パラメーター**

*processName*
オープンするプロセスの名前。

*openOptions*
プロセスのオープンを制御するオプション。

オプションの詳細については、163ページの『″accessProcess″』を参照してください。

*queueManagerName* および *alternateUserId* は "" に設定されます。

### accessDistributionList

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

MQException を投げます。

**パラメーター**

*litems* 配布リストに組み込む項目。

*openOptions*
配布リストのオープンを制御するオプション。

*alternateUserId*
MQOO_ALTERNATE_USER_AUTHORITY が openOptions パラメーターに指定されている場合は、このパラメーターに、オープンの許可を調べるために使用する代替ユーザー ID を指定します。MQOO_ALTERNATE_USER_AUTHORITY が指定されていない場合、このパラメーターはブランク (または null) にしてもかまいません。

**戻り** オープンされ、PUT 操作が実行できる状態になっている新規作成の MQDistributionList。

オープンが失敗した場合は、MQException が投げられます。

161ページの『MQQueueManager.accessQueue』も参照してください。

### accessDistributionList

これは、前述の AccessDistributionList メソッドの単純化されたバージョンです。

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

**パラメーター**

*litems* 配布リストに組み込む項目。

*openOptions*
配布リストのオープンを制御するオプション。

パラメーターの詳細については、165ページの『accessDistributionList』を参照してください。

*alternateUserId* は "" に設定されます。

### begin* (bindings connection only)

```
public synchronized void begin()
```

MQException を投げます。

このメソッドは、バインディング・モードの MQSeries classes for Java でのみサポートされ、新しい作業単位が開始されたことを示すシグナルをキュー・マネージャーに送ります。

このメソッドは、ローカルの 1 フェーズ・トランザクションを使用するアプリケーションには使用しないでください。

### isConnected

```
public boolean isConnected()
```

isConnected 変数の値を戻します。

# MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   |                          |
   +- com.ibm.mq.MQSimpleConnectionManager
```

public class **MQSimpleConnectionManager**
implements **MQConnectionManager** (172 ページを参照してください。)

MQSimpleConnectionManager は、基本接続プール機能を提供します。デフォルトの接続マネージャーとして、または MQQueueManager コンストラクターへのパラメーターとして、 MQSimpleConnectionManager を使用することができます。MQQueueManager が構成されると、プール内の最新の接続が使用されます。

接続が指定された期間未使用であるとき、またはプール内に、指定された数を超える未使用の接続があるとき、接続は (別のスレッドによって) 破棄されます。未使用接続のタイムアウト期間および最大数を指定することができます。

## 変数

**MODE_AUTO**
`public static final int MODE_AUTO`。 『setActive』を参照してください。

**MODE_ACTIVE**
`public static final int MODE_ACTIVE`。 『setActive』を参照してください。

**MODE_INACTIVE**
`public static final int MODE_INACTIVE`。 『setActive』を参照してください。

## コンストラクター

**MQSimpleConnectionManager**
`public MQSimpleConnectionManager()`

MQSimpleConnectionManager を構成します。

## メソッド

**setActive**
`public void setActive(int mode)`

接続プールのアクティブ・モードを設定します。

**パラメーター**

*mode* 接続プールの必須アクティブ・モード。有効な値は、次のとおりです。

**MODE_AUTO**
接続プールは、Connection Manager がデフォルトの接続マネージャーであり、 MQEnvironment によって保持されてい

る MQPoolTokens のセット内に少なくとも 1 つのトークンがある間は、アクティブです。これは、デフォルトのモードです。

**MODE_ACTIVE**
接続プールは、常にアクティブです。MQQueueManager.disconnect() が呼び出されると、基礎接続はプールされ、次回 MQQueueManager オブジェクトが構成されるときに再使用される可能性があります。接続がタイムアウト期間よりも長く未使用であるか、プールのサイズが高しきい値を超える場合、接続は別のスレッドによって破棄されます。

**MODE_INACTIVE**
接続プールは、常に非アクティブです。このモードに入ると、MQSeries への接続のプールはクリアされます。MQQueueManager.disconnect() が呼び出されると、アクティブな MQQueueManager オブジェクトを基礎とする接続は破棄されます。

### getActive

```
public int getActive()
```

接続プールのモードを取得します。

**戻り**

以下の値のいずれかを持つ現行アクティブ・モード (167ページの『setActive』を参照):

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

### setTimeout

```
public void setTimeout(long timeout)
```

タイムアウト値を設定します。ここで、この時間内に未使用のままの接続は、別のスレッドによって破棄されます。

**パラメーター**

*timeout* タイムアウトの値 (ミリ秒単位)。

### getTimeout

```
public long getTimeout()
```

タイムアウト値を戻します。

### setHighThreshold

```
public void setHighThreshold(int threshold)
```

高しきい値を設定します。プール内の未使用接続の数がこの値を超える場合、プール内の 1 番古い未使用接続が破棄されます。

**パラメーター**

*threshold*
プール内の未使用接続の最大数。

### getHighThreshold

```
public int getHighThreshold ()
```

高しきい値を戻します。

# MQC

public interface **MQC**
extends **Object**

MQC インターフェースは、MQ Java プログラミング・インターフェースが使用する定数 (完了コード定数およびエラー・コード定数を除く) をすべて定義しています。これらの定数の 1 つをユーザー・プログラム内から参照するには、定数名に「MQC.」という接頭部を付けてください。 たとえば、キューのクローズ・オプションは次のように設定することができます。

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // クローズされると、
                                      // キューを
                                      // 削除します。
 ...
```

これらの定数の詳しい説明は、*MQSeries アプリケーション・プログラミング・リファレンス*にあります。

完了コード定数およびエラー・コード定数は、MQException クラス内に定義されています。 107ページの『MQException』を参照してください。

# MQPoolServicesEventListener

public interface **MQPoolServicesEventListener**
extends **Object**

**注:** 通常、アプリケーションは、このインターフェースを使用しません。

MQPoolServicesEventListener は、デフォルトの ConnectionManager の提供者によるインプリメンテーションのためのものです。 MQPoolServicesEventListener が MQPoolServices オブジェクトで登録されると、 MQEnvironment が管理する MQPoolTokens のセットに MQPoolToken が追加されたり、またはそのセットから除去されるときにはいつでも、イベント・リスナーはイベントを受け取ります。また、イベント・リスナーは、デフォルトの ConnectionManager が変更されるときはいつでもイベントを受け取ります。

138ページの『MQPoolServices』および139ページの『MQPoolServicesEvent』を参照してください。

## メソッド

### tokenAdded

```
public void tokenAdded(MQPoolServicesEvent event)
```

MQPoolToken がセットに追加されるときに呼び出されます。

### tokenRemoved

```
public void tokenRemoved(MQPoolServicesEvent event)
```

MQPoolToken がセットから除去されるときに呼び出されます。

### defaultConnectionManagerChanged

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

デフォルトの ConnectionManager が設定されるときに呼び出されます。 MQPoolTokens のセットは消去されます。

# MQConnectionManager

これは、アプリケーションがインプリメントできない私用インターフェースです。MQSeries classes for Java は、このインターフェース (MQSimpleConnectionManager) のインプリメンテーションを提供します。これは、MQQueueManager コンストラクター上、または MQEnvironment.setDefaultConnectionManager を介して指定することができます。

167ページの『MQSimpleConnectionManager』を参照してください。

独自の ConnectionManager を提供したいアプリケーションまたはミドルウェアは、javax.resource.spi.ConnectionManager をインプリメントする必要があります。これには、JAAS 1.0 がインストールされている Java 2 v1.3 が必要です。

# MQReceiveExit

public interface **MQReceiveExit**
extends **Object**

受信出口インターフェースにより、 MQSeries classes for Java によってキュー・マネージャーから受信したデータを調べることができ、場合によっては更新することもできます。

**注:** バインディング・モードで直接 MQSeries に接続する場合は、このインターフェースは適用されません。

ユーザー独自の受信出口を提供するには、このインターフェースを導入するクラスを定義します。また、MQQueueManager オブジェクトの構成前に、そのクラスの新規インスタンスを作成し、 MQEnvironment.receiveExit 変数をそのインスタンスに割り当てます。 たとえば、次のようにします。

```
// MyReceiveExit.java 内
class MyReceiveExit implements MQReceiveExit {
  // receiveExit メソッドの
  // インプリメンテーションを提供しなければなりません。
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
   // 出口コードはここで実行されます...
  }
}
// メインプログラム...
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // その他の初期設定
MQQueueManager qMgr        = new MQQueueManager("");
```

## メソッド

### receiveExit

```
public abstract byte[] receiveExit(MQChannelExit channelExitParms,
                                   MQChannelDefinition channelDefinition,
                                   byte agentBuffer[])
```

クラスで提供しなければならない受信出口メソッド。 このメソッドは、MQSeries classes for Java がキュー・マネージャーから何らかのデータを受信すると常に呼び出されます。

#### パラメーター

*channelExitParms*

出口が呼び出されるコンテキストに関する情報が入ります。 exitResponseメンバー変数は、次に実行するアクションを MQSeries classes for Java に指示するために使用する出力パラメーターです。 詳細については、94ページの『MQChannelExit』を参照してください。

*channelDefinition*
キュー・マネージャーとのすべての通信が通過するチャネルの詳細が入ります。

*agentBuffer*
channelExitParms.exitReason が MQChannelExit.MQXR_XMIT の場合は、 agentBuffer にはキュー・マネージャーから受信したデータが入り、これ以外の場合は、agentBuffer は null になります。

### 戻り

MQSeries classes for Java が直ちにデータを処理できる (MQXCC_OK) ように出口応答コードが (channelExitParms で) 設定されている場合は、受信出口メソッドは処理するデータを戻さなければなりません。したがって、最も単純な受信出口は単一行 ″return agentBuffer;″ で構成されます。

以下も参照してください。
- 170ページの『MQC』
- 92ページの『MQChannelDefinition』

# MQSecurityExit

public interface **MQSecurityExit**
extends **Object**

セキュリティー出口インターフェースにより、キュー・マネージャーに接続しようとする時に発生するセキュリティー・フローをカスタマイズできます。

**注:** バインディング・モードで直接 MQSeries に接続する場合は、このインターフェースは適用されません。

ユーザー独自のセキュリティー出口を提供するには、このインターフェースを導入するクラスを定義します。 また、MQQueueManager オブジェクトの構成前に、そのクラスの新規インスタンスを作成し、 MQEnvironment.securityExit 変数をそのインスタンスに割り当てます。 たとえば、次のようにします。

```
// MySecurityExit.java 内
class MySecurityExit implements MQSecurityExit {
  // securityExit メソッドの
  // インプリメンテーションを提供しなければなりません。
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
   // 出口コードはここで実行されます...
  }
}
// メインプログラム...
MQEnvironment.securityExit = new MySecurityExit();
 ...    // その他の初期設定
MQQueueManager qMgr        = new MQQueueManager("");
```

## メソッド

### securityExit

```
public abstract byte[] securityExit(MQChannelExit channelExitParms,
                                    MQChannelDefinition channelDefinition,
                                    byte agentBuffer[])
```

クラスで提供しなければならないセキュリティー出口メソッド。

#### パラメーター

*channelExitParms*
出口が呼び出されるコンテキストに関する情報が入ります。 exitResponseメンバー変数は、次に実行するアクションを MQSeries Client for Java に指示するために使用する出力パラメーターです。 詳細については、94ページの『MQChannelExit』を参照してください。

*channelDefinition*
キュー・マネージャーとのすべての通信が通過するチャネルの詳細が入ります。

*agentBuffer*
channelExitParms.exitReason が MQChannelExit.MQXR_SEC_MSG の

場合は、 agentBuffer にはキュー・マネージャーから受信したセキュリティー・メッセージが入り、これ以外の場合は、agentBuffer は null になります。

### 戻り

メッセージをキュー・マネージャーに送信するように出口応答コードが (channelExitParms で) 設定されている場合は、セキュリティー出口メソッドは送信するデータを戻さなければなりません。

以下も参照してください。
- 170ページの『MQC』
- 92ページの『MQChannelDefinition』

# MQSendExit

public interface **MQSendExit**
extends **Object**

送信出口インターフェースにより、 MQSeries Client for Java によってキュー・マネージャーに送信するデータを調べることができ、場合によっては更新することもできます。

**注:** バインディング・モードで直接 MQSeries に接続する場合は、このインターフェースは適用されません。

ユーザー独自の送信出口を提供するには、このインターフェースを導入するクラスを定義します。また、MQQueueManager オブジェクトの構成前に、そのクラスの新規インスタンスを作成し、 MQEnvironment.sendExit 変数をそのインスタンスに割り当てます。 たとえば、次のようにします。

```
// MySendExit.java 内
class MySendExit implements MQSendExit {
  // sendExit メソッドのインプリメンテーションを提供しなければなりません。
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
   // 出口コードはここで実行されます...
  }
}
// メインプログラム...
MQEnvironment.sendExit = new MySendExit();
 ...    // その他の初期設定
MQQueueManager qMgr        = new MQQueueManager("");
```

## メソッド

### sendExit

```
public abstract byte[] sendExit(MQChannelExit channelExitParms,
                                MQChannelDefinition channelDefinition,
                                byte agentBuffer[])
```

クラスで提供しなければならない送信出口メソッド。 このメソッドは、MQSeries classes for Java がキュー・マネージャーに何らかのデータを送信すると常に呼び出されます。

#### パラメーター

*channelExitParms*
出口が呼び出されるコンテキストに関する情報が入ります。 exitResponseメンバー変数は、次に実行するアクションを MQSeries classes for Java に指示するために使用する出力パラメーターです。 詳細については、94ページの『MQChannelExit』を参照してください。

*channelDefinition*
キュー・マネージャーとのすべての通信が通過するチャネルの詳細が入ります。

*agentBuffer*

channelExitParms.exitReason が MQChannelExit.MQXR_XMIT の場合は、 agentBuffer にはキュー・マネージャーに送信するデータが入り、これ以外の場合は、agentBuffer は null になります。

### 戻り

メッセージをキュー・マネージャーに送信する (MQXCC_OK) ように出口応答コードが (channelExitParms で) 設定されている場合は、送信出口メソッドは送信するデータを戻さなければなりません。 したがって、最も単純な送信出口は単一行 ″return agentBuffer;″ で構成されます。

以下も参照してください。

- 170ページの『MQC』
- 92ページの『MQChannelDefinition』

# ManagedConnection

public interface **javax.resource.spi.ManagedConnection**

**注:** 通常、アプリケーションはこのクラスを使用しません。 ConnectionManager のインプリメンテーションによって使用されることになっています。

MQSeries classes for Java は、 ManagedConnectionFactory.createManagedConnection から戻される ManagedConnection のインプリメンテーションを提供します。このオブジェクトは、MQSeries キュー・マネージャーへの接続を表しています。

## メソッド

### getConnection

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

ResourceException を投げます。

ManagedConnection オブジェクトによって表される物理接続用の新規接続ハンドルを作成します。 MQSeries classes for Java の場合、これは、MQQueueManager オブジェクトを戻します。 ConnectionManager は、通常、allocateConnection からこのオブジェクトを戻します。

サブジェクト・パラメーターは無視されます。 cxRequestInfo パラメーターが適切でない場合、ResourceException が投入されます。それぞれの単一 ManagedConnection ごとに、複数の接続ハンドルを同時に使用することができます。

### destroy

```
public void destroy()
```

ResourceException を投げます。

MQSeries キュー・マネージャーへの物理接続を破棄します。保留ローカル・トランザクションは、コミットされます。詳細については、180ページの『getLocalTransaction』を参照してください。

### cleanup

```
public void cleanup()
```

ResourceException を投げます。

すべてのオープン接続ハンドルをクローズし、物理接続を、プール準備完了になっている初期状態にリセットします。保留ローカル・トランザクションは、ロールバックされます。詳細については、180ページの『getLocalTransaction』を参照してください。

### associateConnection

```
public void associateConnection(Object connection)
```

ResourceException を投げます。

## ManagedConnection

MQSeries classes for Java は、現在、このメソッドをサポートしていません。 javax.resource.NotSupportedException が投入されます。

### addConnectionEventListener

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

ConnectionEventListener を ManagedConnection インスタンスに追加します。

重大エラーが ManagedConnection 上で起こる場合、または MQQueueManager.disconnect() がこの ManagedConnection に関連した接続ハンドル上で呼び出されるとき、リスナーは通知を受けます。リスナーは、ローカル・トランザクション・イベントについては通知されません (『getLocalTransaction』を参照してください)。

### removeConnectionEventListener

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

登録された ConnectionEventListener を除去します。

### getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

ResourceException を投げます。

MQSeries classes for Java は、現在、このメソッドをサポートしていません。 javax.resource.NotSupportedException が投入されます。

### getLocalTransaction

```
public LocalTransaction getLocalTransaction()
```

MQSeries classes for Java は、現在、このメソッドをサポートしていません。 javax.resource.NotSupportedException が投入されます。

現在のところ、ConnectionManager は MQSeries ローカル・トランザクションを管理することができず、登録された ConnectionEventListeners は、ローカル・トランザクションに関連したイベントについて通知されません。 cleanup() が起こるとき、進行中の作業単位はロールバックされます。 destroy() が起こるとき、進行中の作業単位はコミットされます。

既存の API の振る舞いは、進行中の作業単位が MQQueueManager.disconnect() でコミットされるというものです。この既存の振る舞いは、MQConnectionManager (ConnectionManager ではなく) が接続を管理するときにのみ保持されます。

### getMetaData

```
public ManagedConnectionMetaData getMetaData()
```

ResourceException を投げます。

基礎となっているキュー・マネージャーに関するメタ・データ情報を取得します。 184ページの『ManagedConnectionMetaData』を参照してください。

### setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

ResourceException を投げます。

この ManagedConnection 用のログ書き込み機能を設定します。 ManagedConnection が作成されると、 ManagedConnection は、その ManagedConnectionFactory からログ書き込み機能を継承します。

MQSeries classes for Java は、現在、ログ書き込み機能を使用していません。ロギングについての詳細は、107ページの『MQException.log』を参照してください。

### getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

ResourceException を投げます。

この ManagedConnection 用のログ書き込み機能を戻します。

MQSeries classes for Java は、現在、ログ書き込み機能を使用していません。 ロギングについての詳細は、107ページの『MQException.log』を参照してください。

# ManagedConnectionFactory

public interface **javax.resource.spi.ManagedConnectionFactory**

**注:** 通常、アプリケーションはこのクラスを使用しません。

MQSeries classes for Java は、この ConnectionManagers へのインターフェースのインプリメンテーションを提供します。 ManagedConnectionFactory は、ManagedConnection を構成し、候補セットから適切な ManagedConnection を選択するために使用されます。このインターフェースの詳細については、 J2EE Connector Architecture 仕様書を参照してください (http://java.sun.com にある Sun 社の Web サイトを参照してください)。

## メソッド

### createConnectionFactory

```
public Object createConnectionFactory()
```

ResourceException を投げます。

MQSeries classes for Java は、現在、createConnectionFactory メソッドをサポートしていません。このメソッドは、javax.resource.NotSupportedException を投げます。

### createConnectionFactory

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

ResourceException を投げます。

MQSeries classes for Java は、現在、createConnectionFactory メソッドをサポートしていません。このメソッドは、javax.resource.NotSupportedException を投げます。

### createManagedConnection

```
public ManagedConnection createManagedConnection
                                (javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

ResourceException を投げます。

MQSeries キュー・マネージャーに新規物理接続を作成し、この接続を表す ManagedConnection オブジェクトを戻します。 MQSeries は、サブジェクト・パラメーターを無視します。

### matchManagedConnection

```
public ManagedConnection matchManagedConnection
                                (java.util.Set connectionSet,
                                 javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

ResourceException を投げます。

提供されている候補セット ManagedConnection から、該当する ManagedConnection を検索します。ヌルか、または接続の基準にかなうセットから適切な ManagedConnection を戻します。

### setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

ResourceException を投げます。

この ManagedConnectionFactory 用のログ書き込み機能を設定します。ManagedConnection が作成されると、 ManagedConnection は、その ManagedConnectionFactory からログ書き込み機能を継承します。

MQSeries classes for Java は、現在、ログ書き込み機能を使用していません。ロギングについての詳細は、107ページの『MQException.log』を参照してください。

### getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

ResourceException を投げます。

この ManagedConnectionFactory 用のログ書き込み機能を戻します。

MQSeries classes for Java は、現在、ログ書き込み機能を使用していません。 ロギングについての詳細は、107ページの『MQException.log』を参照してください。

### hashCode

```
public int hashCode()
```

この ManagedConnectionFactory 用の hashCode を戻します。

### equals

```
public boolean equals(Object other)
```

この ManagedConnectionFactory が別の ManagedConnectionFactory に等しいかどうかを検査します。両方の ManagedConnectionFactories が同じターゲットのキュー・マネージャーを記述している場合には、真を戻します。

# ManagedConnectionMetaData

public interface **javax.resource.spi.ManagedConnectionMetaData**

**注:** 通常、アプリケーションはこのクラスを使用しません。 ConnectionManager のインプリメンテーションによって使用されることになっています。

ConnectionManager は、キュー・マネージャーへの基礎物理接続に関連のあるメタ・データを検索するために、このクラスを使用することができます。このクラスのインプリメンテーションは、ManagedConnection.getMetaData() から戻されます。

## メソッド

### getEISProductName

```
public String getEISProductName()
```

ResourceException を投げます。

“IBM MQSeries” を戻します。

### getProductVersion

```
public String getProductVersion()
```

ResourceException を投げます。

ManagedConnection の接続先となる MQSeries キュー・マネージャーのコマンド・レベルを記述しているストリングを戻します。

### getMaxConnections

```
public int getMaxConnections()
```

ResourceException を投げます。

0 を戻します。

### getUserName

```
public String getUserName()
```

ResourceException を投げます。

ManagedConnection がキュー・マネージャーへのクライアント接続を表す場合、これは、接続に使用されるユーザー ID を戻します。そうでない場合、空ストリングを戻します。

# 第3部 MQ JMS を使ったプログラミング

# 第10章 MQ JMS プログラムの作成

この章では、MQ JMS アプリケーションの作成を援助するための情報を提供します。また、この章では、JMS モデルに関する簡単な紹介、およびアプリケーション・プログラムが実行する必要がありそうないくつかの共通タスクのプログラミングに関する詳細情報を提供します。

## JMS モデル

JMS は、メッセージ受け渡しサービスの汎用ビューを定義します。このビューについて理解すること、およびこのビューが、基礎となっている MQSeries トランスポートにマップする方法を理解することは重要です。

汎用 JMS モデルは、 Sun 社の `javax.jms` パッケージで定義されている以下のインターフェースに基づいています。

**Connection**
基礎トランスポートへのアクセスを提供し、 **Session** を作成するために使用されます。

**Session**
**MessageProducer** および **MessageConsumer** を作成するために使用されるメソッドを含む、メッセージの生成および使用のためのコンテキストを提供します。

**MessageProducer**
メッセージを送信するために使用されます。

**MessageConsumer**
メッセージを受信するために使用されます。

**Connection** はスレッド・セーフですが、 **Session**、**MessageProducer**、および **MessageConsumer** はスレッド・セーフでないことに注意してください。推奨される戦略は、アプリケーション・スレッドごとに 1 つの Session を使用することです。

MQSeries 用語の場合:

**Connection**
一時キューの有効範囲を提供します。また、MQSeries に接続する方法を制御するパラメーターを保持するための場所を提供します。 MQSeries Java クライアント接続を使用する場合、これらのパラメーターの例としては、キュー・マネージャーの名前、およびリモート・ホストの名前があります。

**Session**
`HCONN` が含まれているので、トランザクションの有効範囲を定義します。

**MessageProducer および MessageConsumer**
書き込みまたは読み取りのための特定のキューを定義する `HOBJ` が含まれています。

通常の MQSeries 規則が適用されることに注意してください。

- どの時点においても、HCONN につき 1 つのオペレーションのみが進行中になることができます。したがって、Session と関連のある MessageProducer または MessageConsumer を同時に呼び出すことはできません。これは、Session につき 1 つのスレッドという JMS 制限と一貫性があります。
- PUT はリモート・キューを使用することができますが、 GET はローカル・キュー・マネージャー上のキューにのみ適用できます。

汎用 JMS インターフェースは、「ポイント・ツー・ポイント方式」および「パブリッシュ / サブスクライブ方式」の振る舞いに関する、さらに特化されたバージョンにサブクラス化されます。

「ポイント・ツー・ポイント」バージョンは、次のとおりです。

- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

JMS のかぎとなる概念は、 `javax.jms` 内のインターフェースへの参照のみを使用するアプリケーション・プログラムを書くことが可能であり、そうするよう強く勧められていることです。すべてのベンダー特定の情報は、以下のインプリメンテーションにカプセル化されます。

- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

これらは「管理対象オブジェクト」(つまり、ベンダー提供の管理ツールを使用して構築でき、 JNDI ネーム・スペース内に保管できるオブジェクト) として知られています。 JMS アプリケーションは、どのベンダーがインプリメンテーションを提供したかを知らなくても、ネーム・スペースからこれらのオブジェクトを検索し、使用することができます。

## 接続の構築

接続は、直接作成されるのではなく、接続ファクトリーを使用して構築されます。ファクトリー・オブジェクトは、JNDI ネーム・スペース内に保管できます。したがって、プロバイダー特定の情報から JMS アプリケーションを隔離することができます。ファクトリー・オブジェクトを作成および保管する方法についての詳細は、37ページの『第5章 MQ JMS 管理ツールの使用』に記載されています。

使用可能な JNDI ネーム・スペースがない場合には、189ページの『実行時のファクトリーの作成』を参照してください。

### JNDI からのファクトリーの検索

JNDI ネーム・スペースからオブジェクトを検索するには、 `IVTRun` サンプル・ファイルから取られている以下のフラグメントに示されているように、初期コンテキストがセットアップされなければなりません。

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

ここで、それぞれは次のものを表します。

**icf** 初期コンテキスト用のファクトリー・クラスを定義します。

**url** コンテキスト特定の URL を定義します。

JNDI の使用法の詳細については、Sun 社の JNDI 資料を参照してください。

**注:** JNDI パッケージと LDAP サービス・プロバイダーの一部の組み合わせの結果、 LDAP エラー 84 が生じることがあります。問題を解決するには、`InitialDirContext` に呼び出しを行う前に、以下の行を挿入してください。

```
environment.put(Context.REFERRAL, "throw");
```

いったん初期テキストを入手すると、オブジェクトは、`lookup()` メソッドを使用してネーム・スペースから検索されます。以下のコードは、LDAP ベースのネーム・スペースから `ivtQCF` という名前の QueueConnectionFactory を検索します。

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

## 接続を作成するためのファクトリーの使用

ファクトリー・オブジェクト上の `createQueueConnection()` メソッドは、次のコードで示されているように、'Connection' を作成するために使用されます。

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

## 実行時のファクトリーの作成

JNDI ネーム・スペースが使用可能でない場合、実行時にファクトリー・オブジェクトを作成することができます。ただし、このメソッドを使用すると、MQSeries 特定クラスへの参照が必要であるため、 JMS アプリケーションの移植性が減少します。

以下のコードは、すべてのデフォルト設定を使って QueueConnectionFactory を作成します。

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(代わりに、`com.ibm.mq.jms` パッケージをインポートする場合、 `com.ibm.mq.jms.` 接頭部を省略することができます。)

上記のファクトリーから作成される接続は、ローカル・マシン上のデフォルトのキュー・マネージャーに接続するために Java バインディングを使用します。 190ページの表14 に示されている `set` メソッドは、 MQSeries 特定情報を持つファクトリーをカスタマイズするために使用できます。

### 接続の開始

JMS 仕様によると、接続が「停止」状態で作成されることが必要です。接続が開始するまで、接続と関連のある MessageConsumer は、メッセージを受信できません。接続を開始するには、以下のコマンドを発行してください。

```
connection.start();
```

*表 14. MQQueueConnectionFactory 上の set メソッド*

| メソッド | 説明 |
|---|---|
| setCCSID(int) | MQEnvironment.CCSID プロパティーを設定するために使用されます。 |
| setChannel(String) | クライアント接続用のチャネルの名前 |
| setHostName(String) | クライアント接続用のホストの名前 |
| setPort(int) | クライアント接続用のポート |
| setQueueManager(String) | キュー・マネージャーの名前 |
| setTemporaryModel(String) | QueueSession.createTemporaryQueue() への呼び出しの結果として一時宛先を生成するために使用されるモデル・キューの名前。これは、永続動的キューではなく、一時動的キューの名前にすることをお勧めします。 |
| setTransportType(int) | MQSeries に接続する方法を指定します。現在使用可能なオプションは、以下のとおりです。<br>• JMSC.MQJMS_TP_BINDINGS_MQ (デフォルト)<br>• JMSC.MQJMS_TP_CLIENT_MQ_TCPIP<br>JMSC は、パッケージ com.ibm.mq.jms 内にあります。 |
| setReceiveExit(String)<br>setSecurityExit(String)<br>setSendExit(String)<br>setReceiveExitInit(String)<br>setSecutityExitInit(String)<br>setSendExitInit(String) | これらのメソッドは、基礎となっている MQSeries Classes for Java によって提供される送信、受信、セキュリティー出口の使用を可能にするためのものです。 set*Exit メソッドは、関係のある終了メソッドをインプリメントするクラスの名前をとります。 (詳細については、MQSeries 5.1 の製品資料を参照してください。)<br>また、クラスは、単一の String パラメーターを持つコンストラクターをインプリメントしなければなりません。このストリングは、出口が必要とする初期化データを提供し、対応する set*ExitInit メソッドで提供される値に設定されます。 |

## クライアントまたはバインディング・トランスポートの選択

MQ JMS は、クライアントまたはバインディング・トランスポートのいずれかを使用して MQSeries と通信することができます。 Java バインディングを使用する場合、 JMS アプリケーションと MQSeries キュー・マネージャーは、同じマシン上になければなりません。クライアントを使用する場合には、キュー・マネージャーは、アプリケーションと異なるマシン上にあってもかまいません。

接続ファクトリー・オブジェクトの内容は、どのトランスポートを使用するかを決定します。 37ページの『第5章 MQ JMS 管理ツールの使用』 では、クライアントまたはバインディング・トランスポートとともに使用するためのファクトリー・オブジェクトを定義する方法を説明しています。

以下のコード・フラグメントは、アプリケーション内にトランスポートをどのように定義できるかを示しています。

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

## セッションの取得

いったん接続が行われたら、QueueConnection 上の createQueueSession メソッドを使用して、セッションを取得してください。

メソッドは、2 つのパラメーターをとります。

1. セッションが 'transacted' または 'non-transacted' のどちらかを判別するブール値。
2. 'acknowledge' モードを判別するパラメーター。

最も単純なケースは、以下のコード・フラグメントに示されているように、AUTO_ACKNOWLEDGE との 'non-transacted' セッションの場合です。

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

**注:** 接続はスレッド・セーフですが、セッション (およびそれらのセッションから作成されるオブジェクト) はスレッド・セーフではありません。マルチスレッド・アプリケーションに関して推奨されることは、それぞれのスレッドごとに別々のセッションを使用することです。

## メッセージの送信

メッセージは、MessageProducer を使用して送信されます。ポイント・ツー・ポイント方式の場合、これは、QueueSession 上の createSender メソッドを使用して作成される QueueSender です。 QueueSender は、通常、その送信元を使用して送信されるすべてのメッセージが同じ宛先に送信されるように、特定のキューのために作成されます。宛先は Queue オブジェクトを使用して指定されます。キュー・オブジェクトは、実行時に作成されるか、 JNDI ネーム・スペース内に構築され保管されるかのいずれかです。

キュー・オブジェクトは、以下の方法で JNDI から検索されます。

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS は、com.ibm.mq.jms.MQQueue 内に Queue のインプリメンテーションを提供します。これには、MQSeries 特有の振る舞いの詳細を制御するプロパティーが含まれていますが、多くの場合、デフォルト値を使用することができます。 JMS は

宛先を指定する標準の方法を定義して、アプリケーション内の MQSeries 特定コードを最小化にとどめます。このメカニズムは、QueueSession.createQueue メソッドを使用し、宛先を記述している string パラメーターをとります。ストリング自体はベンダー特定の形式のままですが、これは、直接ベンダー・クラスを参照するよりももっと柔軟性のある方法です。

MQ JMS は、createQueue() の string パラメーター用の 2 つの形式を受け入れます。

- 最初の形式は、samples ディレクトリー内の IVTRun プログラムから取られている以下のフラグメントに示されているように、MQSeries キューの名前です。

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- 2 番目の形式は、さらに強力で、一様リソース ID (URI) に基づいています。この形式では、リモート・キュー (接続しているキュー・マネージャー以外のキュー・マネージャー上のキュー) を指定することができます。また、この形式では、 com.ibm.mq.jms.MQQueue オブジェクトに含まれているその他のプロパティーを設定することもできます。

  キュー用の URI は、シーケンス queue:// で始まり、キューが入っているキュー・マネージャーの名前が続きます。この後に、さらに、残りの Queue プロパティーを設定する、'/'、キューの名前、そして任意に、名前値の対のリストが続きます。たとえば、前の例と等価の URI は、以下のとおりです。

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  キュー・マネージャーの名前は省略されることに注意してください。これは、Queue オブジェクトが使用されるときに、所有する QueueConnection の接続先となるキュー・マネージャーとして解釈されます。

以下の例は、キュー・マネージャー 'HOST1.QM1' 上のキュー 'Q1' に接続し、すべてのメッセージを非持続性および優先順位 5 として送信します。

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

表15 では、URI の名前値の部分で使用できる名前をリストしています。この形式の欠点は、それらの値用の記号名がサポートされていないことです。したがって、適切な箇所で、表は特殊値を示しています。これらの特殊値は変わることがあることに注意してください。 (プロパティーの別の設定方法については、193ページの『'set' メソッドを使用したプロパティーの設定』を参照してください。)

*表 15. キュー URI 用のプロパティー名*

| プロパティー | 説明 | 値 |
|---|---|---|
| expiry | メッセージの存続時間 (ミリ秒単位) | 無限の場合には 0、タイムアウトの場合には正整数 (ms) |
| priority | メッセージの優先順位 | 0 ～ 9、-1=QDEF、-2=APP |
| persistence | メッセージがディスクに「固定保管」される必要があるかどうか | 1 = 非持続性、 2 = 持続性、-1=QDEF、-2=APP |
| CCSID | 宛先の文字セット | 整数 - 基本 MQSeries 資料内にリストされている有効な値 |

*表 15. キュー URI 用のプロパティー名 (続き)*

| プロパティー | 説明 | 値 |
|---|---|---|
| targetClient | 受信アプリケーションが JMS 準拠かどうか | 0=JMS、1=MQ |
| encoding | 数値フィールドの表示方法 | 基本 MQSeries 資料内で説明されている整数値 |
| **QDEF** - プロパティーが MQSeries キューの構成によって決定される必要があることを意味する特殊値。<br>**APP** - JMS アプリケーションがこのプロパティーを制御できることを意味する特殊値。 | | |

いったん Queue オブジェクトが取得されたら (上記のように createQueue を使用して、または JNDI から)、 QueueSender を作成するために createSender メソッドに渡さなければなりません。

```
QueueSender queueSender = session.createSender(ioQueue);
```

結果として生じる queueSender オブジェクトは、 send メソッドを使用してメッセージを送信するために使用されます。

```
queueSender.send(outMessage);
```

## 'set' メソッドを使用したプロパティーの設定

まず、デフォルトのコンストラクターを使用して com.ibm.mq.jms.MQQueue のインスタンスを作成することによって、 Queue のプロパティーを設定することができます。次に、パブリック set メソッドを使用して、必要な値を入れることができます。このメソッドは、プロパティー値の記号名を使用できることを意味しています。ただし、これらの値はベンダー特定であり、コード内に組み込まれるので、アプリケーションは移植性が減少します。

以下のコード・フラグメントは、 set メソッドを使用したキュー・プロパティーの設定を示しています。

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

表16 は、 set メソッドで使用するために MQ JMS で提供される記号プロパティー値を示しています。

*表 16. キュー・プロパティーの記号値*

| プロパティー | 管理ツール・キーワード | 値 |
|---|---|---|
| expiry | UNLIM<br>APP | JMSC.MQJMS_EXP_UNLIMITED<br>JMSC.MQJMS_EXP_APP |
| priority | APP<br>QDEF | JMSC.MQJMS_PRI_APP<br>JMSC.MQJMS_PRI_QDEF |

*表 16. キュー・プロパティーの記号値 (続き)*

| プロパティー | 管理ツール・キーワード | 値 |
|---|---|---|
| persistence | APP<br>QDEF<br>PERS<br>NON | JMSC.MQJMS_PER_APP<br>JMSC.MQJMS_PER_QDEF<br>JMSC.MQJMS_PER_PER<br>JMSC.MQJMS_PER_NON |
| targetClient | JMS<br>MQ | JMSC.MQJMS_CLIENT_JMS_COMPLIANT<br>JMSC.MQJMS_CLIENT_NONJMS_MQ |
| encoding | Integer(N)<br>Integer(R)<br>Decimal(N)<br>Decimal(R)<br>Float(N)<br>Float(R)<br>Native | JMSC.MQJMS_ENCODING_INTEGER_NORMAL<br>JMSC.MQJMS_ENCODING_INTEGER_REVERSED<br>JMSC.MQJMS_ENCODING_DECIMAL_NORMAL<br>JMSC.MQJMS_ENCODING_DECIMAL_REVERSED<br>JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL<br>JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED<br>JMSC.MQJMS_ENCODING_NATIVE |

エンコードの詳細については、49ページの『ENCODING プロパティー』を参照してください。

## メッセージ・タイプ

JMS は、いくつかのメッセージ・タイプを提供します。それぞれのメッセージ・タイプには、その内容に関するいくつかの知識が盛り込まれています。メッセージ・タイプ用のベンダー特定のクラス名を参照することを避けるために、メッセージ作成のためのメソッドが Session オブジェクトに提供されています。

サンプル・プログラムでは、以下の方法で、テキスト・メッセージが作成されます。

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

使用できるメッセージ・タイプは、以下のとおりです。

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

これらのタイプの詳細は、253ページの『第14章 JMS インターフェースおよびクラス』に記載されています。

# メッセージの受信

メッセージは、QueueReceiver を使用して受信されます。これは、createReceiver() メソッドを使用して Session から作成されます。このメソッドは、メッセージが受信される場所を定義する Queue パラメーターをとります。 Queue オブジェクトを作成する方法の詳細については、191ページの『メッセージの送信』を参照してください。

サンプル・プログラムは、受信側を作成し、以下のコードでテスト・メッセージを読み返します。

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

受信呼び出しにおけるパラメーターはタイムアウトです (ミリ秒単位)。このパラメーターは、即時に使用可能なメッセージがない場合のメソッドの待機時間を定義します。このパラメーターを省略することができます。この場合、呼び出しは無期限にブロックします。遅延したくない場合には、receiveNoWait() メソッドを使用してください。

受信メソッドは、適切なタイプのメッセージを戻します。たとえば、TextMessage がキュー上に置かれる場合、メッセージを受信するときに、戻されるオブジェクトは、TextMessage のインスタンスです。

メッセージ本文から内容を抽出するには、汎用メッセージ・クラス (受信メソッドの宣言された戻りタイプ) から TextMessage などのさらに特定のサブクラスにキャストする必要があります。受信されたメッセージ・タイプが分からない場合には、'instanceof' 演算子を使用して、メッセージ・タイプを判別することができます。予期しないエラーをスムーズに処理できるように、常に、キャストする前にメッセージ・クラスをテストすることを習慣にしてください。

以下のコードは、'instanceof' の使用、および TextMessage からの内容の抽出を示しています。

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

## メッセージ・セレクター

JMS は、このサブセットが受信呼び出しによって戻されるように、キュー上にメッセージのサブセットを選択するためのメカニズムを提供します。 QueueReceiver を作成するとき、どのメッセージを検索するかを判別するための SQL (構造化照会言語) 式を含むストリングを提供することができます。セレクターは、JMS メッセージ・ヘッダー内のフィールドだけでなく、メッセージ・プロパティー内のフィールド (これらは実際にはアプリケーション定義のヘッダー・フィールドである) も参照することができます。ヘッダー・フィールド名と SQL セレクターの構文の詳細は、211ページの『第12章 JMS メッセージ』に記載されています。

### メッセージの受信

以下の例は、myProp という名前のユーザー定義のプロパティーを選択する方法を示しています。

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

**注:** JMS 仕様は、受信側と関連のあるセレクターの変更を許可していません。いったん受信側が作成されると、セレクターは、その受信側の存続期間にわたって固定されます。これは、異なるセレクターが必要な場合には、新しい受信側を作成しなければならないことを意味しています。

## 非同期送達

QueueReceiver.receive() への呼び出しを行うもう 1 つの方法は、適切なメッセージが使用可能であるときに自動的に呼び出されるメソッドを登録することです。以下のフラグメントはそのメカニズムを示しています。

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // メッセージが使用可能であるとき、JMS によって呼び出されるメソッド。
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // アプリケーション特定の処理
    .
    .
    .
  }
}

 .
 .
 .
  // メイン・プログラム (他の何らかのクラスのもの)
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // メイン・プログラムは、他のアプリケーション特定の振る舞いで続行できます。
```

**注:** QueueReceiver とともに非同期送達を使用すると、Session 全体が非同期としてマークされます。非同期送達を使用している Session と関連のある QueueReceiver の receive メソッドに明示的呼び出しを行うと、エラーになります。

# クローズ

ガーベッジ・コレクションだけは、タイミング良くすべての MQSeries リソースを解放することはできません。これは、特に、アプリケーションが Session レベル以下で数多くの一時 JMS オブジェクトを作成する必要がある場合に当てはまります。したがって、リソースが必要でなくなったときに様々なクラス (QueueConnection、QueueSession、 QueueSender、および QueueReceiver) の close() メソッドを呼び出すことは重要です。

### シャットダウン時の Java 仮想マシンのハング

MQ JMS アプリケーションが Connection.close() を呼び出さないで終了すると、一部の JVM はハングします。この問題が起こる場合には、 Connection.close() への呼び出しを組み込むようにアプリケーションを編集するか、または Ctrl-C キーを使用して JVM を終了してください。

## エラーの処理

JMS アプリケーション内の実行時エラーは、例外によって報告されます。 JMS 内の大多数のメソッドは、エラーを示すために JMSExceptions を投じます。これらの例外をキャッチし、適切な出力にそれらの例外を表示することは、良いプログラミングの習慣です。

通常の Java 例外とは異なり、 JMSException には、その中に組み込まれるさらに多くの例外が含まれることがあります。 JMS では、このことは、基礎となっているトランスポートから重要な詳細情報を渡すための有効な方法となります。 MQ JMS の場合、MQSeries が MQException を出すとき、この例外は、通常、JMSException 内の組み込み例外として含められます。

JMSException のインプリメンテーションは、その toString() メソッドの出力に組み込み例外を含んでいません。したがって、以下のフラグメントに示されているように、組み込み例外を明示的にチェックし、プリントすることが必要です。

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

### 例外リスナー

非同期メッセージ送達では、アプリケーション・コードは、メッセージを受信する際の障害によって持ち上がる例外をキャッチできません。これは、アプリケーション・コードが receive() メソッドに明示呼び出しを行わないためです。この状態に対処するには、ExceptionListener (onException() メソッドをインプリメントするクラスのインスタンス) を登録することができます。重大エラーが起こると、このメソッドは、その唯一のパラメーターとして渡される JMSException とともに呼び出されます。さらに詳細な情報は、Sun 社の JMS 資料に記載されています。

# 第11章 パブリッシュ / サブスクライブ・アプリケーションのプログラミング

このセクションでは、MQSeries Classes for Java Message Service を使用するパブリッシュ / サブスクライブ・アプリケーションを作成するために使用されるプログラミング・モデルを紹介します。

## 単純なパブリッシュ / サブスクライブ・アプリケーションの作成

このセクションでは、単純な MQ JMS アプリケーションの ‘作成シナリオ’ を提供します。

### 必要なパッケージをインポートする

MQSeries classes for Java Message Service アプリケーションは、少なくとも以下のものを含むいくつかのインポート・ステートメントで始まります。

```
import javax.jms.*;                 // 管理対象オブジェクトの
import javax.naming.*;              // JNDI ルックアップに使用される
import javax.naming.directory.*;    // JMS インターフェース
```

### JMS オブジェクトを取得または作成する

次のステップは、いくつかの JMS オブジェクトを取得または作成することです。

1. TopicConnectionFactory を取得する
2. TopicConnection を作成する
3. TopicSession を作成する
4. JNDI からトピックを取得する
5. TopicPublisher および TopicSubscriber を作成する

以下に示されているように、これらのプロセスの多くは、ポイント・ツー・ポイント方式に使用されているプロセスに類似しています。

#### TopicConnectionFactory を取得する

TopicConnectionFactory を取得するための良い方法は、アプリケーション・コードの移植性が保たれるように、JNDI ルックアップを使用することです。以下のコードは、JNDI コンテキストを初期設定します。

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // JNDI コンテキストへの接続不能を処理するためのコードを追加します。
}
```

**注:** CTX_FACTORY および INIT_URL 変数は、インストール・システムおよび JNDI サービス・プロバイダーに適するようにカスタマイズする必要があります。

JNDI 初期設定に必要なプロパティーは、ハッシュ・テーブル内にあり、InitialDirContext コンストラクターに渡されます。この接続が失敗する場合、アプリケーション内で後で必要となる管理対象オブジェクトが使用不能であることを示すために例外が出されます。

ここで、アドミニストレーターが定義したルックアップ・キーを使用して、TopicConnectionFactory を取得してください。

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

JNDI ネーム・スペースが使用可能でない場合、実行時に TopicConnectionFactory を作成することができます。 189ページの『実行時のファクトリーの作成』内の QueueConnectionFactory で説明されているメソッドと類似した方法で、新しい com.ibm.mq.jms.MQTopicConnectionFactory を作成します。

### TopicConnection を作成する

これは TopicConnectionFactory オブジェクトから作成されます。接続は、常に、stop 状態で初期設定され、以下のコードで開始されなければなりません。

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

### TopicSession を作成する

これは TopicConnection を使用して作成されます。このメソッドは 2 つのパラメーターをとります。 1 つはセッションがトランザクションされるかどうかを示し、もう 1 つは確認通知モードを指定します。

```
TopicSession session = conn.createTopicSession( false,
                                                 Session.AUTO_ACKNOWLEDGE );
```

### トピックを取得する

このオブジェクトは、後で作成される TopicPublisher および TopicSubscriber で使用するために JNDI から取得できます。以下のコードは、トピックを検索します。

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
    // JNDI からのトピックの検索不能を処理するためのコードを追加します。
 }
```

JNDI ネーム・スペースが使用可能でない場合、 203ページの『実行時のトピックの作成』で説明されているように、実行時にトピックを作成することができます。

### パブリケーションのコンシューマーとプロデューサーを作成する

作成する JMS クライアント・アプリケーションの性質によって、サブスク

ライバーまたはパブリッシャー (あるいはその両方) を作成しなければなりません。以下のように、createPublisher および createSubscriber メソッドを使用してください。

```
// 指定されたトピックについてパブリッシュして、パブリッシャーを作成します。
 TopicPublisher pub = session.createPublisher( topic );
// 指定されたトピックについてサブスクライブして、サブスクライバーを作成します。
 TopicSubscriber sub = session.createSubscriber( topic );
```

## メッセージをパブリッシュする

TopicPublisher オブジェクト pub は、メッセージをパブリッシュするために使用されます。これは、QueueSender がポイント・ツー・ポイント・ドメイン内で使用されるのと同様です。以下のフラグメントは、セッションを使用して TextMessage を作成してから、メッセージをパブリッシュします。

```
// TextMessage を作成し、いくつかのデータを TextMessage に入れます。
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "This is a short test string!" );

// パブリッシャーを使用して、メッセージをパブリッシュします。
pub.publish( outMsg );
```

## サブスクリプションを受信する

以下のコードにあるように、サブスクライバーは、送達されるサブスクリプションを読み取ることができなければなりません。

```
// 次の待機中のサブスクリプションを検索します。
TextMessage inMsg = (TextMessage)sub.receive();

// メッセージの内容を取得します。
String payload = inMsg.getText();
```

このコード・フラグメントは、'get-with-wait' を実行します。これは、メッセージが使用可能になるまで、受信呼び出しがブロックされることを意味しています。受信呼び出しの代替バージョンが使用可能です ('receiveNoWait' など)。詳細については、362ページの『TopicSubscriber』を参照してください。

## 不必要なリソースをクローズする

終了時に、パブリッシュ / サブスクライブ・アプリケーションによって使用されるすべてのリソースを解放することは重要です。クローズできるオブジェクト (パブリッシャー、サブスクライバー、セッション、および接続) 上で close() メソッドを使用してください。

```
// パブリッシャーおよびサブスクライバーをクローズします。
pub.close();
sub.close();

// セッションおよび接続をクローズします。
session.close();
conn.close();
```

# トピックの使用

このセクションでは、MQSeries classes for Java Message Service アプリケーション内での JMS Topic オブジェクトの使用法について説明します。

## トピック名

このセクションでは、MQSeries classes for Java Message Service 内でのトピック名の使用法について説明します。

**注:** JMS 仕様は、トピック階層の使用および保守に関する厳密な詳細を指定していません。したがって、この領域は、プロバイダーによって異なることがあります。

MQ JMS 内のトピック名は、ツリー階層で配置されます。 図3 にそれらの例が示されています。

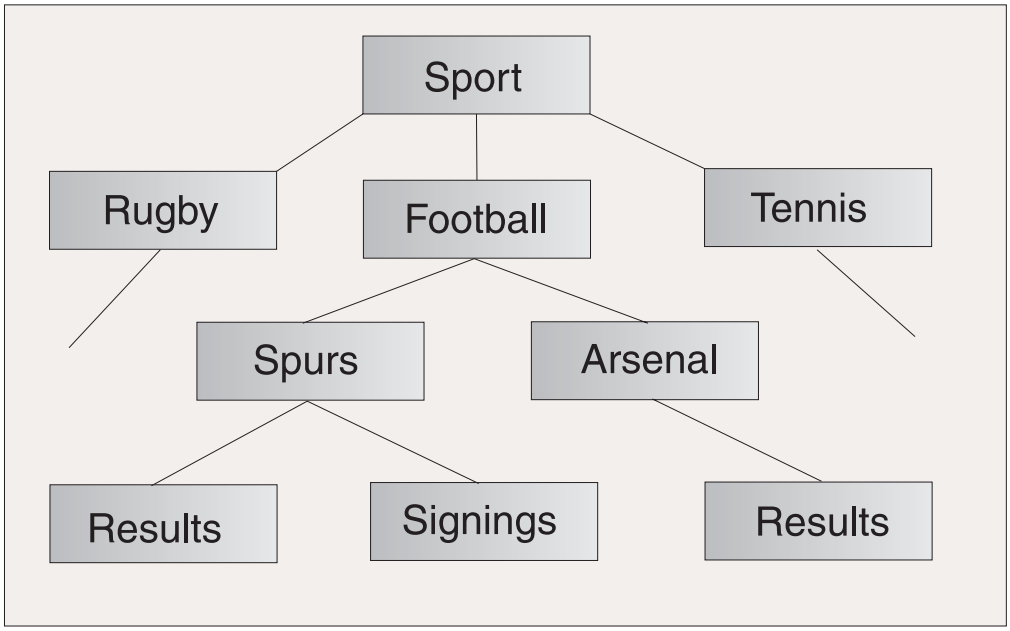


*図3. トピック名の階層*

トピック名では、ツリー内のレベルは、'/' 文字で区切られます。これは、'Signings' ノードがトピック名によって表されることを意味しています。

```
Sport/Football/Spurs/Signings
```

MQSeries classes for Java Message Service におけるトピック・システムの強力な機能は、ワイルドカードの使用です。それによって、サブスクライバーが 1 度に複数のトピックにサブスクライブすることが可能になります。 '*' ワイルドカードはゼロ以上の文字と一致しますが、'?' ワイルドカードは単一文字と一致します。

サブスクライバーが、以下のトピック名で示されているトピックにサブスクライブする場合、

```
Sport/Football/*/Results
```

サブスクライバーは、以下のものを含むトピックについてのパブリケーションを受信します。

- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

サブスクリプション・トピックが以下のものである場合、

```
Sport/Football/Spurs/*
```

サブスクライバーは、以下のものを含むトピックについてのパブリケーションを受信します。

- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

明示的に、システムのブローカー側で使用するトピック階層を管理する必要はありません。指定されたトピックについての最初のパブリッシャーまたはサブスクライバーが存在するようになると、ブローカーは自動的に、現在パブリッシュされサブスクライブされているトピックの状態を作成します。

**注:** パブリッシャーは、名前にワイルドカードが含まれているトピックについてはパブリッシュできません。

## 実行時のトピックの作成

実行時に Topic オブジェクトを作成する 4 つの方法があります。

1. 1 つの引き数 MQTopic コンストラクターを使用してトピックを構成します。
2. デフォルトの MQTopic コンストラクターを使用してトピックを構成してから、setBaseTopicName(..) メソッドを呼び出します。
3. セッションの createTopic(..) メソッドを使用します。
4. セッションの createTemporaryTopic() メソッドを使用します。

**方式 1: MQTopic(..) の使用**

この方式では、JMS Topic インターフェースの MQSeries インプリメンテーションへの参照が必要なので、コードは移植不能になります。

コンストラクターは、1 つの引き数をとり、一様リソース ID (URI) でなければなりません。 MQSeries classes for Java Message Service Topic の場合、これは、以下の形式でなければなりません。

```
topic://TopicName[?property=value[&property=value]*]
```

URI および許可される名前値の対の詳細については、191ページの『メッセージの送信』を参照してください。

以下のコードは、非持続性のトピック、優先順位 5 のメッセージを作成します。

```
// 1 つの引き数 MQTopic コンストラクターを使用して Topic を作成します。
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

**方式 2: MQTopic()、そして setBaseTopicName(..) の使用**

この方式は、デフォルトの MQTopic コンストラクターを使用します。したがって、コードは移植不能になります。

オブジェクトが作成された後、setBaseTopicName メソッドを使用し、必要なトピック名に入れて渡して、baseTopicName プロパティーを設定してください。

**注:** ここで使用されているトピック名は、非 URI 形式であり、名前値の対を含めることはできません。 193ページの『'set' メソッドを使用したプ

ロパティーの設定』で説明されているように、 'set' メソッドを使用してこれらの値を設定してください。以下のコードは、このメソッドを使用して、トピックを作成します。

```
// デフォルトの MQTopic コンストラクターを使用して Topic を作成します。
Topic rtTopic = new MQTopic();

// setter メソッドを使用してオブジェクト・プロパティーを設定します。
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

**方式 3: session.createTopic(..) の使用**

Topic オブジェクトは、 TopicSession の createTopic メソッドを使用して作成されることもあります。これは、以下のように、トピック URI をとります。

```
// session ファクトリー・メソッドを使用して、Topic を作成します。
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

**方式 4: session.createTemporaryTopic() の使用**

TemporaryTopic は、同じ TopicConnection によって作成されるサブスクライバーによってのみ使用される Topic です。 TemporaryTopic は、以下のように作成されます。

```
// session ファクトリー・メソッドを使用して TemporaryTopic を作成します。
Topic rtTopic = session.createTemporaryTopic();
```

# サブスクライバー・オプション

JMS サブスクライバーを使用するためのいくつかの異なる方法があります。このセクションでは、JMS サブスクライバーを使用するいくつかの例について説明します。

JMS は、2 つのタイプのサブスクライバーを提供します。

**非永続サブスクライバー**

これらのサブスクライバーは、サブスクライバーがアクティブなときにメッセージがパブリッシュされる場合にのみ、選択されたトピックについてのメッセージを受信します。

**永続サブスクライバー**

これらのサブスクライバーは、サブスクライバーが活動状態にないときにパブリッシュされるものを含め、トピックについてパブリッシュされるすべてのメッセージを受信します。

## 非永続サブスクライバーの作成

200ページの『パブリケーションのコンシューマーとプロデューサーを作成する』で作成されたサブスクライバーは非永続であり、以下のコードで作成されます。

```
// 指定されたトピックについてサブスクライブして、サブスクライバーを作成します。
 TopicSubscriber sub = session.createSubscriber( topic );
```

## 永続サブスクライバーの作成

永続サブスクライバーの作成は、非永続サブスクライバーの作成と非常に類似していますが、サブスクライバーを固有に識別する名前も提供しなければなりません。

```
// 固有に識別する名前を提供して、永続サブスクライバーを作成します。
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

close() メソッドが呼び出されるとき (またはそれらのサブスクライバーが有効範囲外になるとき)、非永続サブスクライバー自体は自動的に登録解除します。ただし、永続サブスクリプションを終了したい場合には、明示的にシステムに通知しなければなりません。これを行うには、セッションの unsubscribe() メソッドを使用して、サブスクライバーを作成した固有名で渡してください。

```
// 上記で作成した永続サブスクライバーをサブスクライブ解除します
session.unsubscribe( "D_SUB_000001" );
```

永続サブスクライバーは、 MQTopicConnectionFactory キュー・マネージャー・パラメーター内に指定されているキュー・マネージャーで作成されます。異なるキュー・マネージャーで、同じ名前を持つ永続サブスクライバーを作成する試みが続けて行われた場合、新しい、完全に独立した永続サブスクライバーが戻されます。

## メッセージ・セレクターの使用

メッセージ・セレクターを使用して、与えられている規準を満たしていないメッセージをフィルターに掛けて除外することができます。メッセージ・セレクターの詳細については、195ページの『メッセージ・セレクター』を参照してください。メッセージ・セレクターは、以下のように、サブスクライバーと関連があります。

```
// メッセージ・セレクターを、非永続サブスクライバーと関連付けます。
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

## ローカル・パブリケーションの抑制

サブスクライバー自体の接続でパブリッシュされるパブリケーションを無視するサブスクライバーを作成することは可能です。以下のように、createSubscriber 呼び出しの 3 番目のパラメーターを true に設定します。

```
// noLocal オプションを設定して、非永続サブスクライバーを作成します。
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

## サブスクライバー・オプションの結合

必要であれば、セレクターを適用してローカル・パブリケーションを無視する永続サブスクライバーを作成することができるように、サブスクライバーのバリエーションを結合することができます。以下のコード・フラグメントは、結合されたオプションの使用法を示しています。

```
// セレクターが適用された永続 noLocal サブスクライバーを作成します。
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                        selector, true );
```

## 基本サブスクライバー・キューの構成

MQ JMS V5.2 では、サブスクライバーを構成できる 2 つの方法があります。

- 複数キュー・アプローチ

  それぞれのサブスクライバーは、そのサブスクライバーに割り当てられる排他キューを持っており、そこから、そのメッセージ全体を検索します。 JMS は、サブスクライバーごとに新しいキューを作成します。これは、MQ JMS V1.1 で使用可能な唯一のアプローチです。

- 共用キュー・アプローチ

  あるサブスクライバーが共用キューを使用し、そこから、その他のサブスクライバーはそれらのメッセージを検索します。このアプローチでは、複数のサブスクライバーを提供するために 1 つのキューのみを必要とします。これは、MQ JMS V5.2 で使用されるデフォルトのアプローチです。

MQ JMS V5.2 では、どのアプローチを使用するかを選択したり、どのキューを使用するかを構成することができます。

一般的に、共用キュー・アプローチのほうが、パフォーマンス上の利点があります。高スループットを持つシステムの場合、必要なキュー数を意図的に削減するため、大きなアーキテクチャー上および管理上の利点もあります。

以下のように、状況によっては、複数キュー・アプローチを使用するほうが良い場合もあります。

- メッセージ・ストレージの理論物理容量がより大きいです。

  MQSeries キューは、640000 を超えるメッセージを保持することはできません。共用キュー・アプローチでは、キューを共用するすべてのサブスクライバー間でこれを分けなければなりません。 この問題は、永続サブスクライバーの場合にはさらに重大です。永続サブスクライバーの存続時間は、通常、非永続サブスクライバーの存続時間よりもさらに長いためです。したがって、永続サブスクライバーの場合、さらに多くのメッセージが累積される場合があります。

- サブスクリプション・キューの外部管理が、より容易です。

  ある特定のアプリケーション・タイプの場合、アドミニストレーターは、特定のサブスクライバー・キューの状態および深さをモニターしたい場合もあるでしょう。このタスクは、サブスクライバーとキューとの間で 1 対 1 のマッピングがあるとき、かなり単純になります。

## デフォルトの構成

デフォルトの構成では、以下の共用サブスクリプション・キューを使用します。

- SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE (非永続サブスクリプションの場合)
- SYSTEM.JMS.D.SUBSCRIPTION.QUEUE (永続サブスクリプションの場合)

これらは、MQJMS_PSQ.MQSC スクリプトを実行するときに作成されます。

必要であれば、代替物理キューを指定することができます。複数キュー・アプローチを使用するために構成を変更することもできます。

## 非永続サブスクライバーの構成

以下のいずれかの方法で、非永続サブスクライバーのキュー名プロパティーを設定することができます。

- MQ JMS 管理ツール (JNDI 検索済みオブジェクト用) を使用して、BROKERSUBQ プロパティーを設定します。
- ユーザーのプログラムで setBrokerSubQueue() メソッドを使用します。

非永続サブスクリプションの場合、ユーザーが指定するキュー名は、以下の文字で始まらなければなりません。

SYSTEM.JMS.ND.

共用キュー・アプローチを選択するには、明示キュー名を指定してください。ここで、名前付きキューは、共用キューに使用するためのキューです。指定するキューは、サブスクリプションを作成する前にすでに物理的に存在していなければなりません。

複数キュー・アプローチを選択するには、* 文字で終わるキュー名を指定してください。その後、このキュー名で作成されるそれぞれのサブスクライバーは、その特定のサブスクライバー専用の、適切な動的キューを作成します。 MQ JMS は、そのようなキューを作成するために独自の内部モデル・キューを使用します。 したがって、複数キュー・アプローチでは、必要なすべてのキューが動的に作成されます。

複数キュー・アプローチを使用するとき、明示キュー名を指定することはできません。ただし、キュー接頭部を指定することができます。これにより、異なるサブスクライバー・キュー・ドメインを作成することができます。たとえば、以下のようにすることができます。

SYSTEM.JMS.ND.MYDOMAIN.*

このサブスクリプションと関連のあるすべての動的キューが SYSTEM.JMS.ND.MYDOMAIN で始まるキュー名を持つようにするため、 * 文字より前の文字が接頭部として使用されます。

## 永続サブスクライバーの構成

以前に説明したように、永続サブスクリプションに複数キュー・アプローチを使用する十分の理由があります。永続サブスクリプションは、より長いライフ・スパンを持っているのが普通です。したがって、大量の未検索のメッセージがキュー上に累積していることが考えられます。

したがって、永続サブスクライバーのキュー名プロパティーは、 Topic オブジェクト (つまり、TopicConnectionFactory よりもさらに管理しやすいレベル) 内に設定されます。これにより、TopicConnectionFactory から始まる複数のオブジェクトを再作成しなくても、いくつかの異なるサブスクライバー・キュー名を指定することができます。

以下のいずれかの方法で、永続サブスクライバー・キュー名を設定することができます。

- MQ JMS 管理ツール (JNDI 検索済みオブジェクト用) を使用して、BROKERDURSUBQ プロパティーを設定します。
- ユーザーのプログラムで setBrokerDurSubQueue() メソッドを使用します。

```
// 複数キュー・アプローチを使用して、
// MQTopic 永続サブスクライバー・キュー名を設定します。
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

いったん Topic オブジェクトが初期設定されると、指定されたサブスクリプションを作成するために TopicSession createDurableSubscriber() メソッドに渡されます。

```
// 初期の Topic を使用して永続サブスクライバーを作成します。
TopicSubscriber sub = new session.createDurableSubscriber
                                          (sportsTopic, "D_SUB_SPORT_001");
```

永続サブスクリプションの場合、提供するキュー名は、以下の文字で始まらなければなりません。

SYSTEM.JMS.D.

共用キュー・アプローチを選択するには、明示キュー名を指定してください。ここで、名前付きキューは、共用キューに使用するためのキューです。指定するキューは、サブスクリプションを作成する前にすでに物理的に存在していなければなりません。

複数キュー・アプローチを選択するには、* 文字で終わるキュー名を指定してください。その後、このキュー名で作成されるそれぞれのサブスクライバーは、その特定のサブスクライバー専用の、適切な動的キューを作成します。 MQ JMS は、そのようなキューを作成するために独自の内部モデル・キューを使用します。 したがって、複数キュー・アプローチでは、必要なすべてのキューが動的に作成されます。

複数キュー・アプローチを使用するとき、明示キュー名を指定することはできません。ただし、キュー接頭部を指定することができます。これにより、異なるサブスクライバー・キュー・ドメインを作成することができます。たとえば、以下のようにすることができます。

SYSTEM.JMS.D.MYDOMAIN.*

このサブスクリプションと関連のあるすべての動的キューが SYSTEM.JMS.D.MYDOMAIN で始まるキュー名を持つようにするため、 * 文字より前の文字が接頭部として使用されます。

### 永続サブスクライバーの再作成およびマイグレーションの問題

永続サブスクライバーの場合、サブスクライバーが削除されるまで、サブスクライバー・キュー名を再構成しないでください。つまり、unsubscribe() を実行した後、新たに再びキューを作成してください (古いサブスクライバー・メッセージは削除されることに留意してください)。

ただし、MQ JMS V1.1 を使用してサブスクライバーを作成した場合、そのサブスクライバーは、現行レベルにマイグレーションするときに認識されます。サブスクリプションを削除する必要はありません。サブスクリプションは、複数キュー・アプローチを使用して操作を続けます。

## パブリッシュ / サブスクライブの問題の解決

このセクションでは、パブリッシュ / サブスクライブ・ドメインを使用する JMS クライアント・アプリケーションを開発するときに発生する可能性のあるいくつかの問題について説明します。このセクションでは、パブリッシュ / サブスクライブ・ドメインに特有の問題について説明していることに注意してください。一般的なトラブルシューティングのガイダンスの詳細については、 197ページの『エラーの処理』および 35ページの『問題の解決』を参照してください。

### 不完全なパブリッシュ / サブスクライブのクローズ

JMS クライアント・アプリケーションが、終了時に、すべての外部リソースを引き渡すことは重要です。これを行うには、オブジェクトが必要でなくなったら、クロ

ーズできるすべてのオブジェクト上で close() メソッドを呼び出してください。パブリッシュ / サブスクライブ・ドメインの場合、これらのオブジェクトは、以下のとおりです。

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

MQSeries classes for Java Message Service インプリメンテーションでは、「カスケード・クローズ」を使用してこのタスクを容易にします。このプロセスでは、TopicConnection 上での 'close' への呼び出しは、作成した TopicSession のそれぞれに 'close' への呼び出しを行うことになります。これにより、セッションが作成したすべての TopicSubscriber および TopicPublisher 上に、 'close' への呼び出しを行うことになります。

したがって、外部リソースの適切な解放を確認するために、アプリケーションが作成するそれぞれの接続ごとに connection.close() を呼び出すことは重要です。

状況によっては、'close' プロシージャーが完了しない場合があります。これらの状況には、以下のものが含まれます。

- MQSeries クライアントの接続切れ
- 予期しないアプリケーションの終了

これらの状況では、close() は呼び出されず、外部リソースは、終了したアプリケーションのためにオープンしたままです。この主な結果は、以下のとおりです。

**ブローカー状態の不整合**

MQSeries メッセージ・ブローカーには、もはや存在しないサブスクライバーおよびパブリッシャーに関する登録情報が含まれている可能性があります。これは、ブローカーが、メッセージを受信することのないサブスクライバーにメッセージを転送し続ける場合があることを意味しています。

**サブスクライバーおよびキューの残り**

サブスクライバーの登録解除プロシージャーの一部分として、サブスクライバー・メッセージの除去があります。適切であれば、サブスクリプションを受信するために使用された、基礎となっている MQSeries キューも除去されます。通常の終了が行われなかった場合、これらのメッセージおよびキューは残ったままです。ブローカー状態の不整合がある場合、キューは、読み取られることのないメッセージで引き続き埋め尽くされます。

## サブスクライバー・クリーンアップ・ユーティリティー

サブスクライバー・オブジェクトのスムーズでないクローズと関連した問題を避けるために、 MQ JMS には、サブスクライバー・クリーンアップ・ユーティリティーが含まれています。 このユーティリティーは、その物理キュー・マネージャーを使用するための最初の TopicConnection が初期設定されるときに、キュー・マネージャー上で実行します。指定されたキュー・マネージャー上のすべての TopicConnection がクローズされる場合、そのキュー・マネージャーに対して、次の TopicConnection が初期設定されるとき、ユーティリティーは再び実行します。

クリーンアップ・ユーティリティーは、その他のアプリケーションから発生した初期の MQ JMS パブリッシュ / サブスクライブ問題を検出しようとします。問題が検出された場合、以下のようにして、関連したリソースをクリーンアップします。

- MQSeries メッセージ・ブローカーに対して登録解除する
- サブスクリプションと関連した未検索のメッセージおよびキューをクリーンアップする

クリーンアップ・ユーティリティーは、バックグラウンドで透過的に実行され、短時間だけ持続します。その他の MQ JMS 操作には影響を与えません。指定されたキュー・マネージャーに対して大量の問題が検出された場合、リソースがクリーンアップされる間、初期設定時間に少しの遅れが生じることがあります。

**注:** 可能ならいつでも、サブスクライバー問題の発生を避けるために、すべてのサブスクライバー・オブジェクトを正しくクローズすることを強くお勧めします。

## ブローカー・レポートの処理

MQ JMS インプリメンテーションは、コマンドの登録および登録解除を確認するために、ブローカーからのレポート・メッセージを使用します。これらのレポート・メッセージは、通常、MQSeries classes for Java Message Service インプリメンテーションによって使い果たされますが、いくつかのエラー状態の下では、レポート・メッセージがキュー上に残っている場合があります。これらのメッセージは、ローカル・キュー・マネージャー上の `SYSTEM.JMS.REPORT.QUEUE` キューに送信されます。

Java アプリケーションの `PSReportDump` は、MQSeries classes for Java Message Service とともに提供され、プレーン・テキスト形式でこのキューの内容をダンプします。その後、これらの情報は、ユーザーまたは IBM サポート担当員のいずれかによって解析することができます。問題が診断または修正された後、メッセージのキューを消去するために、このアプリケーションを使用することもできます。

ツールのコンパイル形式は、 `<MQ_JAVA_INSTALL_PATH>/bin` ディレクトリーにインストールされます。ツールを呼び出すには、このディレクトリーに変更してから、以下のコマンドを使用してください。

```
java PSReportDump [-m queueManager] [-clear]
```

ここで、それぞれは次のものを表します。

**-m queueManager**
= 使用するキュー・マネージャーの名前を指定します。

**-clear** = その内容をダンプした後、メッセージのキューを消去します。

出力は画面に送信されるか、またはその出力をファイルにリダイレクトすることができます。

# 第12章 JMS メッセージ

JMS メッセージは、以下の部分から成り立っています。

**ヘッダー** すべてのメッセージは、同じヘッダー・フィールドのセットをサポートします。ヘッダー・フィールドには、メッセージを識別し、経路指定するために、クライアントとプロバイダーの両方によって使用される値が含まれています。

**プロパティー** それぞれのメッセージには、アプリケーション定義のプロパティー値をサポートするための組み込み機能が含まれています。プロパティーは、アプリケーション定義のメッセージをフィルターに掛けるための効果的なメカニズムを提供します。

**本体** JMS は、現在使用されているメッセージング・スタイルの大部分を包含するいくつかのタイプのメッセージ本体を定義しています。

JMS は、以下の 5 つのタイプのメッセージ本体を定義しています。

**ストリーム** Java プリミティブ値のストリーム。ストリームに満たされたメッセージは、順次に読み取られます。

**マップ** 一連の名前値の対。ここで、名前はストリングであり、値は Java プリミティブ・タイプです。エントリーには、順次アクセスか、または名前によるランダムでのアクセスを行えます。エントリーの順序は定義されていません。

**テキスト** java.util.String を含むメッセージ。

**オブジェクト** 直列化可能 Java オブジェクトを含むメッセージ。

**バイト** 非解釈バイトのストリーム。このメッセージ・タイプは、事実上、既存のメッセージ形式と一致するように本体をエンコードするためのものです。

JMSCorrelationID ヘッダー・フィールドは、 1 つのメッセージを別のメッセージとリンクするために使用されます。 JMSCorrelationID ヘッダー・フィールドは、通常、応答メッセージをその要求メッセージとリンクします。 JMSCorrelationID は、プロバイダー特定のメッセージ ID、アプリケーション特定のストリング、またはプロバイダー・ネイティブの byte[] 値を保持することができます。

## メッセージ・セレクター

メッセージには、アプリケーション定義のプロパティー値をサポートするための組み込み機能が含まれています。実際これは、メッセージにアプリケーション特定のヘッダー・フィールドを追加するためのメカニズムを提供します。プロパティーにより、アプリケーションは、メッセージ・セレクターを介して、アプリケーション特定の基準を使用して、JMS プロバイダーがメッセージを選択したり、またはメッ

セージをフィルターに掛けることができるようにします。アプリケーション定義のプロパティーは、以下の規則に従わなければなりません。

- プロパティー名は、メッセージ・セレクター ID に関する規則に従わなければなりません。
- プロパティー値は、boolean、byte、short、int、long、float、double、および string とすることができます。
- 以下の名前の接頭部は予約済みです。JMSX、JMS_。

プロパティー値は、メッセージを送信する前に設定されます。クライアントがメッセージを受信すると、メッセージ・プロパティーは、読み取り専用になります。この時点で、クライアントがプロパティーを設定しようとする場合、MessageNotWriteableException が投げられます。 clearProperties が呼び出される場合、プロパティーは読み取りにも書き込みにもなることができます。

プロパティー値は、メッセージの本体に値を複製する場合もあれば、複製しない場合もあります。 JMS は、プロパティーに何が作成され、何が作成されないかについてポリシーを定義していません。ただし、アプリケーション開発者は、 JMS プロバイダーがたいてい、メッセージのプロパティー内のデータよりもより効率的にメッセージ本体内のデータを処理できることに留意すべきです。より良いパフォーマンスを得るために、アプリケーションは、メッセージのヘッダーをカスタマイズする必要があるときに、メッセージ・プロパティーのみを使用すべきです。これを行う主な理由は、カスタマイズされたメッセージ選択をサポートすることです。

JMS メッセージ・セレクターにより、クライアントは、メッセージ・ヘッダーを使用して、関心のあるメッセージを指定することができます。ヘッダーがセレクターと一致するメッセージのみが送達されます。

メッセージ・セレクターは、メッセージ本体の値を参照することはできません。

メッセージのヘッダー・フィールドおよびプロパティー値がセレクター内の対応する ID に置換される際に、セレクターが true に評価されるとき、メッセージ・セレクターはメッセージと一致します。

メッセージ・セレクターはストリングであり、その構文は SQL92 条件式構文のサブセットに基づいています。メッセージ・セレクターが評価される順序は、優先順位内で左から右です。この順序を変更するために括弧を使用することができます。定義済みのセレクター・リテラルおよび演算子名は、大文字でここに書き込まれます。ただし、これらは大文字小文字の区別がありません。

セレクターには、以下のものを含めることができます。

- リテラル
  - ストリング・リテラルは、単一引用符で囲まれます。二重の単一引用符は、単一引用符を表しています。例は、'literal' および 'literal''s' です。 Java ストリング・リテラルのように、これらは Unicode 文字エンコードを使用します。
  - 正確な数字リテラルは、57、-957、+62 など、小数点なしの数値です。 Java long の範囲内の数値がサポートされています。
  - 近似数値リテラルは、7E3 または -57.9E2 などの浮動小数における数値、または 7.、-95.7、または +6.2 などの小数部を持つ数値です。 Java double の範囲内の数値がサポートされています。

  - ブール・リテラルの TRUE および FALSE。
- ID:
  - ID は、Java 英字および Java 数字の無制限の長さシーケンスであり、そのうちの最初のものは、 Java 英字です。英字は、メソッド Character.isJavaLetter が true を戻す任意の文字です。これには、'_' および '$' が含まれます。英字または数字は、メソッド Character.isJavaLetterOrDigit が true を戻す任意の文字です。
  - ID は、名前 NULL、TRUE、または FALSE になることはできません。
  - ID は、NOT、AND、OR、BETWEEN、LIKE、IN、および IS になることはできません。
  - ID は、ヘッダー・フィールド参照またはプロパティー参照のいずれかです。
  - ID には、大文字小文字の区別があります。
  - メッセージ・ヘッダー・フィールド参照は、以下のものに制限されます。
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    JMSMessageID、JMSTimestamp、JMSCorrelationID、および JMSType 値は、ヌルの場合があります。そうであれば、NULL 値として扱われます。
  - 'JMSX' で始まる名前は、JMS 定義のプロパティー名です。
  - 'JMS_' で始まる名前は、プロバイダー特定のプロパティー名です。
  - 'JMS' で始まっていない名前は、アプリケーション特定のプロパティー名です。メッセージ内に存在しないプロパティーへの参照がある場合、その値は NULL です。存在しない場合には、その値は、対応するプロパティー値です。
- 空白は、Java に定義されているのと同じで、スペース、水平タブ、改ページ、および行終了文字です。
- 式:
  - セレクターは、条件式です。 true に評価されるセレクターは一致し、false または unknown に評価されるセレクターは一致しません。
  - 演算式は、それ自体と、算術演算、 ID (その値は数値リテラルとして扱われる) および数値リテラルから成っています。
  - 条件式は、それ自体と、比較演算、および論理演算から成り立っています。
- 標準の括弧 () (式が評価される順序を設定する) がサポートされています。
- 論理演算子 (優先順位どおりに列挙): NOT、AND、OR。
- 比較演算子: =、>、>=、<、<=、<> (等しくない)。
  - 同じタイプの値のみを比較することができます。 1 つの例外は、正確な数値と近似数値の比較が有効であることです。 (必要な型変換は、Java 数値プロモーションによって定義されます。) 異なるタイプを比較する試みがある場合、セレクターは常に false です。

  - ストリングとブールの比較は、= および <> に制限されます。 2 つのストリングは、それらのストリングに含まれている文字シーケンスが全く同じ場合にのみ等しいです。
- 算術演算子 (優先順位どおりに列挙):
  - 単項 +、-。
  - *、/ (乗算および除算)。
  - +、- (加算および減算)。
  - NULL 値での算術演算はサポートされていません。 NULL 値での算術演算が試行される場合、完全セレクターは常に false です。
  - 算術演算は、Java 数値プロモーションを使用しなければなりません。
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 比較演算子:
  - age BETWEEN 15 and 19 は、age >= 15 AND age <= 19 と同じです。
  - age NOT BETWEEN 15 and 19 は、age < 15 OR age > 19 と同じです。
  - BETWEEN 演算の expr のいずれかが NULL である場合、演算の値は false です。 NOT BETWEEN 演算の expr のいずれかが NULL である場合、演算の値は true です。
- ID がストリングまたは NULL 値を持つ identifier [NOT] IN (string-literal1, string-literal2,...) 比較演算子。
  - Country IN (' UK', 'US', 'France') は 'UK' の場合には true であり、 'Peru' の場合には false です。これは、式 (Country = ' UK') OR (Country = ' US') OR (Country = ' France') と同じです。
  - Country NOT IN (' UK', 'US', 'France') は 'UK' の場合には false であり、 'Peru' の場合には true です。これは、式 NOT ((Country = ' UK') OR (Country = ' US') OR (Country = ' France')) と同じです。
  - IN または NOT IN 演算の ID が NULL である場合、演算の値は不明です。
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] 比較演算子。ここで、identifier はストリング値を持っています。 pattern-value はストリング・リテラルです。ここで、'_' は単一文字を表しており、 '%' は文字シーケンス (空のシーケンスを含む) を表しています。その他のすべての文字はそれ自体を表しています。任意の escape-character は単一の文字ストリング・リテラルです。この文字は、パターン値内の '_' および '%' の特殊な意味をエスケープするために使用されます。
  - phone LIKE '12%3' は '123' '12993' の場合には true で、'1234' の場合には false です。
  - word LIKE 'l_se' は 'lose' の場合には true で、'loose' の場合には false です。
  - underscored LIKE '¥_%' ESCAPE '¥' は '_foo' の場合には true で、 'bar' の場合には false です。
  - phone NOT LIKE '12%3' は '123' '12993' の場合には false であり、 '1234' の場合には true です。
  - LIKE または NOT LIKE 演算の ID が NULL である場合には、演算の値は不明です。

- identifier IS NULL 比較演算子は、ヌルのヘッダー・フィールド値または欠落したプロパティー値をテストします。
  - prop_name IS NULL
- identifier IS NOT NULL 比較演算子は、ヌルでないヘッダー・フィールド値またはプロパティー値の存在をテストします。
  - prop_name IS NOT NULL

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

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

上記で注釈されたように、プロパティー値は NULL である場合があります。
NULL 値を含むセレクター式の評価は、SQL 92 NULL セマンティクスによって定義されます。以下は、これらのセマンティクスの要旨です。

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

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

SQL コメントは、サポートされていません。

# JMS メッセージの MQSeries メッセージへのマッピング

このセクションでは、この章の最初の部分で説明されている JMS メッセージ構造が MQSeries メッセージにマップされる方法について説明します。このセクションは、JMS と従来の MQSeries アプリケーションとの間でメッセージを伝送したいプログラマーを対象としています。また、2 つの JMS アプリケーション間で伝送されるメッセージを操作したいユーザーも対象としています (たとえば、メッセージ・ブローカーのインプリメンテーション)。

MQSeries メッセージは、以下の 3 つのコンポーネントから成り立っています。
- MQSeries メッセージ記述子 (MQMD)
- MQSeries MQRFH2 ヘッダー
- メッセージ本体

MQRFH2 はオプションであり、出力メッセージ内の包含物は、JMS Destination クラス内のフラグによって管理されます。 MQSeries JMS 管理ツールを使用してこのフラグを設定することができます。 MQRFH2 は JMS 特定の情報を搬送するため、受信宛先が JMS アプリケーションであることが送信側に分かっているときには、常にメッセージ内にその情報が含まれています。通常、メッセージを直接非 JMS アプリケーション (MQSeries ネイティブ・アプリケーション) に送信するときには、 MQRFH2 を省略してください。これは、そのようなアプリケーションがその MQSeries メッセージ内に MQRFH2 を予期していないためです。 図4 は、構造の変換を示しています。

JMS メッセージ・モデルの MQSeries へのマッピング

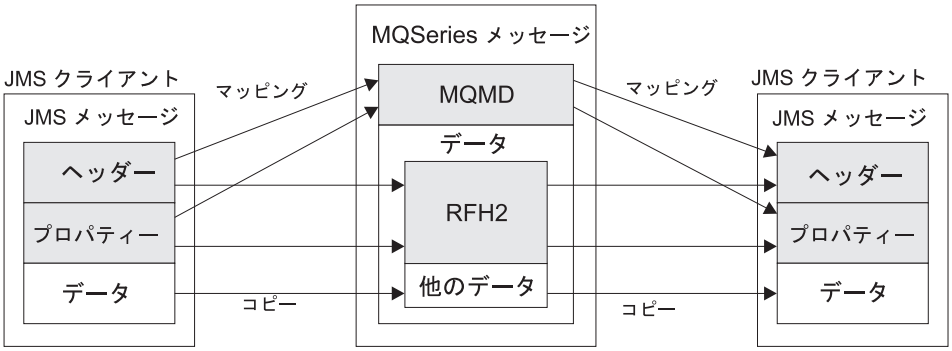


*図 4. JMS から MQSeries へのマッピング・モデル*

これらの構造は、以下の 2 つの方法で変換されます。

**マッピング**
MQMD が JMS フィールドと等しいフィールドを含んでいる場合、 JMS フィールドは MQMD フィールドにマップされます。追加の MQMD フィールドは、JMS プロパティーとして公開されます。JMS アプリケーションが、非 JMS アプリケーションと通信するときにこれらのフィールドを取得または設定する必要があるためです。

**コピー** 等しい MQMD がない場合、JMS ヘッダー・フィールドまたはプロパティーが渡され、 MQRFH2 内のフィールドとして変換されます。

## MQRFH2 ヘッダー

このセクションでは、MQRFH バージョン 2 ヘッダーについて説明します。これは、メッセージ内容と関連のある JMS 特定のデータを搬送します。 MQRFH2 バージョン 2 は、拡張可能なヘッダーであり、直接 JMS と関連のない追加情報も搬送することができます。ただし、このセクションでは、JMS による使用のみを扱います。

ヘッダーには 2 つの部分として、固定部分と変数部分があります。

### 固定部分

固定部分は、標準の MQSeries ヘッダー・パターンにモデル化され、以下のフィールドから成り立っています。

**StrucId (MQCHAR4)**
構造 ID。

MQRFH_STRUC_ID (値: ″RFH ″) (初期値) でなければなりません。

MQRFH_STRUC_ID_ARRAY (値: 'R','F','H',' ') は、通常の方法でも定義されます。

**Version (MQLONG)**
構造のバージョン番号。

MQRFH_VERSION_2 (値: 2) (初期値) でなければなりません。

**StrucLength (MQLONG)**
NameValueData フィールドを含む、MQRFH2 の全長。

StrucLength に設定される値は、 4 の倍数でなければなりません (NameValueData フィールド内のデータは、これをアーカイブするためにスペース文字で埋め込まれる場合があります)。

**Encoding (MQLONG)**
データ・エンコード。

MQRFH2 の後のメッセージの部分にある数値データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) のエンコード。

**CodedCharSetId (MQLONG)**
コード化文字セット ID。

MQRFH2 の後のメッセージの部分にある文字データ (次のヘッダー、またはこのヘッダーの後のメッセージ・データ) の表記。

**Format (MQCHAR8)**
フォーマット名。

MQRFH2 の後のメッセージの部分のフォーマット名。

**Flags (MQLONG)**
フラグ。

MQRFH_NO_FLAGS =0。セットされたフラグはありません。

**NameValueCCSID (MQLONG)**
このヘッダーに含まれている NameValueData 文字ストリング用の

コード化文字セット ID (CCSID)。 NameValueData は、ヘッダー (StrucID および Format) 内に含まれているその他の文字ストリングとは異なる文字セットでコード化される場合があります。

NameValueCCSID が 2 バイトの Unicode CCSID (1200、13488、または 17584) である場合、 Unicode のバイト順は、MQRFH2 内の数値フィールドのバイト順と同じです。 (たとえば、Version、StrucLength、NameValueCCSID 自体。)

NameValueCCSID は、以下のリストにある値だけを取ります。

| | |
|---|---|
| 1200 | UCS2 開放型 |
| 1208 | UTF8 |
| 13488 | UCS2 2.0 サブセット |
| 17584 | UCS2 2.1 サブセット (ユーロ・シンボルを含む) |

### 変数部分

変数部分は、固定部分の後に続きます。変数部分には、可変数の MQRFH2 フォルダーが含まれます。それぞれのフォルダーには、可変数のエレメントまたはプロパティーが含まれます。フォルダーは、関連のあるプロパティーをグループにまとめます。 JMS によって作成される MQRFH2 ヘッダーには、最大 3 つのフォルダーを含めることができます。

**<mcd> フォルダー**

このフォルダーには、メッセージの形状またはフォーマットについて記述しているプロパティーが含まれます。たとえば、msd プロパティーは、メッセージを Text、Bytes、Stream. Map、Object、または 'Null' として識別します。このフォルダーは、常に、JMS MQRFH2 内に存在します。

**<jms> フォルダー**

これは、JMS ヘッダー・フィールドと、 MQMD では完全に表現できない JMSX プロパティーを移送するために使用されます。このフォルダーは、常に、JMS MQRFH2 内に存在します。

**<usr> フォルダー**

このフォルダーは、メッセージと関連のあるアプリケーション定義のプロパティーを移送するために使用されます。このフォルダーは、アプリケーションが何らかのアプリケーション定義のプロパティーを設定した場合にのみ存在します。

表17 は、プロパティー名の完全なリストを示しています。

*表 17. JMS によって使用される MQRFH2 フォルダーおよびプロパティー*

| JMS フィールド | | MQRFH2 フィールド | | |
|---|---|---|---|---|
| 名前 | Java タイプ | フォルダー名 | プロパティー名 | タイプ/値 |
| JMSDestination | Destination | jms | Dst | ストリング |
| JMSExpiration | long | jms | Exp | i8 |
| JMSPriority | int | jms | Pri | i4 |
| JMSDeliveryMode | int | jms | Dlv | i4 |
| JMSCorrelationID | String | jms | Cid | ストリング |

*表 17. JMS によって使用される MQRFH2 フォルダーおよびプロパティー (続き)*

| JMS フィールド | | MQRFH2 フィールド | | |
|---|---|---|---|---|
| 名前 | Java タイプ | フォルダー名 | プロパティー名 | タイプ/値 |
| JMSReplyTo | Destination | jms | Rto | ストリング |
| JMSType | String | mcd | Type | ストリング |
| JMSXGroupID | String | jms | Gid | ストリング |
| JMSXGroupSeq | int | jms | Seq | i4 |
| xxx (User Defined) | Any | usr | xxx | 任意 |
| | | mcd | Msd | jms_none<br>jms_text<br>jms_bytes<br>jms_map<br>jms_stream<br>jms_object |

変数部分でプロパティーを表現するために使用される構文は、以下のとおりです。

**NameValueLength (MQLONG)**

この長さフィールドの直後にある NameValueData ストリングの長さ (バイト単位) (そのストリング自体の長さは含まない)。NameValueLength に設定される値は、4 の倍数でなければなりません (NameValueData フィールドは、これをアーカイブするためにスペース文字で埋め込まれます)。

**NameValueData (MQCHARn)**

単一文字ストリング。長さ (バイト単位) は、前の NameValueLength フィールドに示されています。この文字ストリングには、プロパティーのシーケンスを保持しているフォルダーが含まれています。それぞれのプロパティーは、'name/type/value' トリプレットで、名前がフォルダー名である XML エレメント内に含まれており、以下のとおりです。

```
<foldername> triplet1 triplet2 .....   tripletn </foldername>
```

`</foldername>` という終了タグの後には、埋め込み文字としてスペースを入れることができます。それぞれのトリプレットは、XML のような構文を使用してエンコードされます。

```
<name dt='datatype'>value</name>
```

`dt='datatype'` エレメントのデータ・タイプは事前定義であるため、このエレメントはオプションであり、多くのプロパティーの場合省略されます。このエレメントが組み込まれている場合には、1 つまたは複数のスペース文字を、 `dt=` タグの前に入れなければなりません。

`name` はプロパティーの名前です。 218ページの表17 を参照してください。

`datatype` は、大文字への変換後、 220ページの表18 内のリテラル値の 1 つと一致していなければなりません。

表18 に示されているように、 value は、伝送される値のストリング表記です。

ヌル値は、以下の構文を使用してエンコードされます。

```
<name/>
```

*表 18. プロパティーのデータ・タイプと値*

| データ・タイプ | 値 |
|---|---|
| string | < および & を除外する文字のシーケンス |
| boolean | 文字 0 または 1 (1 = ″true″) |
| bin.hex | オクテットを表す 16 進数字 |
| i1 | 0〜9 の数字とオプションの符号を使って表現される整数 (分数や指数ではない)。-128 〜 127 (両端を含む) の範囲内になければなりません。 |
| i2 | 0〜9 の数字とオプションの符号を使って表現される整数 (分数や指数ではない)。-32768 〜 32767 (両端を含む) の範囲内になければなりません。 |
| i4 | 0〜9 の数字とオプションの符号を使って表現される整数 (分数や指数ではない)。-2147483648 〜 2147483647 (両端を含む) の範囲内になければなりません。 |
| i8 | 0〜9 の数字とオプションの符号を使って表現される整数 (分数や指数ではない)。-9223372036854775808 〜 9223372036854775807 (両端を含む) の範囲内になければなりません。 |
| int | 0〜9 の数字とオプションの符号を使って表現される整数 (分数や指数ではない)。'i8' と同じ範囲内になければなりません。送信側が特定の精度をプロパティーと関連付けたくない場合に、 'i*' タイプのいずれかの代わりにこれを使用することができます。 |
| r4 | 0〜9 の数字、オプションの符号、オプションの小数桁、オプションの指数を使用して表現される浮動小数点数。絶対値 <= 3.40282347E+38、>= 1.175E-37。 |
| r8 | 0〜9 桁、オプションの符号、オプションの小数桁、オプションの指数を使用して表現される浮動小数点数。絶対値 <= 1.7976931348623E+308、>= 2.225E-307。 |

ストリング値には、スペースを含めることができます。ストリング値では以下のエスケープ・シーケンスを使用しなければなりません。

& 文字には &
< 文字には <

以下のエスケープ・シーケンスを使用することができますが、必須ではありません。

> 文字には >
' 文字には '
" 文字には "

## 対応する MQMD フィールドを持つ JMS フィールドおよびプロパティー

221ページの表19 は、MQMD フィールドに直接マップされるプロパティーをリストしています。

*表 19. MQMD フィールドへの JMS プロパティーのマッピング*

| JMS フィールド | | MQMD フィールド | |
|---|---|---|---|
| ヘッダー | Java タイプ | フィールド | C タイプ |
| JMSDeliveryMode | int | Persistence | MQLONG |
| JMSExpiration | long | Expiry | MQLONG |
| JMSPriority | int | Priority | MQLONG |
| JMSMessageID | String | MessageID | MQBYTE24 |
| JMSTimestamp | long | PutDate<br>PutTime | MQCHAR8<br>MQCHAR8 |
| JMSCorrelationID | String | CorrelId | MQBYTE24 |
| **プロパティー** | | | |
| JMSXUserID | String | UserIdentifier | MQCHAR12 |
| JMSXAppID | String | PutApplName | MQCHAR28 |
| JMSXDeliveryCount | int | BackoutCount | MQLONG |
| JMSXGroupID | String | GroupId | MQBYTE24 |
| JMSXGroupSeq | int | MsgSeqNumber | MQLONG |
| **プロバイダー特定** | | | |
| JMS_IBM_Report_Exception | int | Report | MQLONG |
| JMS_IBM_Report_Expiration | int | Report | MQLONG |
| JMS_IBM_Report_COA | int | Report | MQLONG |
| JMS_IBM_Report_COD | int | Report | MQLONG |
| JMS_IBM_Report_PAN | int | Report | MQLONG |
| JMS_IBM_Report_NAN | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Msg_ID | int | Report | MQLONG |
| JMS_IBM_Report_Pass_Correl_ID | int | Report | MQLONG |
| JMS_IBM_Report_Discard_Msg | int | Report | MQLONG |
| JMS_IBM_MsgType | int | MsgType | MQLONG |
| JMS_IBM_Feedback | int | Feedback | MQLONG |
| JMS_IBM_Format | String | Format | MQCHAR8 |
| JMS_IBM_PutApplType | int | PutApplType | MQLONG |
| JMS_IBM_Encoding | int | Encoding | MQLONG |
| JMS_IBM_Character_Set | String | CodedCharacterSetId | MQLONG |

## JMS フィールドの MQSeries フィールドへのマッピング (出力メッセージ)

222ページの表20 は、send() または publish() の際にヘッダー / プロパティー・フィールドが MQMD/RFH2 フィールドにマップされる方法を示しています。

'Set by Message Object' というマークが付いているフィールドの場合、伝送される値は、send/publish() の直前に JMS メッセージ内に保持されている値です。 JMS メッセージ内の値は、send/publish() によって未変更のままにされます。

'Set by Send Method' というマークが付いているフィールドの場合、 send/publish() の実行時に値が割り当てられます (JMS メッセージ内に保持されている値は無視されます)。 JMS メッセージ内の値は、使用されている値を示すように更新されます。

'Receive-only' というマークが付いているフィールドは、伝送されず、 send() または publish() によってメッセージ内で未変更のままです。

*表 20. 出力メッセージ・フィールドのマッピング*

| JMS フィールド | 伝送手段 | | Set by |
|---|---|---|---|
| 名前 | MQMD フィールドに入れる | ヘッダーに入れる | |
| JMSDestination | | MQRFH2 | Send Method |
| JMSDeliveryMode | Persistence | MQRFH2 | Send Method |
| JMSExpiration | Expiry | MQRFH2 | Send Method |
| JMSPriority | Priority | MQRFH2 | Send Method |
| JMSMessageID | MessageID | | Send Method |
| JMSTimestamp | PutDate/PutTime | | Send Method |
| JMSCorrelationID | CorrelId | MQRFH2 | Message Object |
| JMSReplyTo | ReplyToQ/ReplyToQMgr | MQRFH2 | Message Object |
| JMSType | | MQRFH2 | Message Object |
| JMSRedelivered | | | Receive-only |
| プロパティー | | | |
| JMSXUserID | UserIdentifier | | Send Method |
| JMSXAppID | PutApplName | | Send Method |
| JMSXDeliveryCount | | | Receive-only |
| JMSXGroupID | GroupId | MQRFH2 | Message Object |
| JMSXGroupSeq | MsgSeqNumber | MQRFH2 | Message Object |
| プロバイダー特定 | | | |
| JMS_IBM_Report_Exception | Report | | Message Object |
| JMS_IBM_Report_Expiration | Report | | Message Object |
| JMS_IBM_Report_COA/COD | Report | | Message Object |
| JMS_IBM_Report_NAN/PAN | Report | | Message Object |
| JMS_IBM_Report_Pass_Msg_ID | Report | | Message Object |
| JMS_IBM_Report_Pass_Correl_ID | Report | | Message Object |
| JMS_IBM_Report_Discard_Msg | Report | | Message Object |
| JMS_IBM_MsgType | MsgType | | Message Object |
| JMS_IBM_Feedback | Feedback | | Message Object |
| JMS_IBM_Format | Format | | Message Object |
| JMS_IBM_PutApplType | PutApplType | | Send Method |
| JMS_IBM_Encoding | Encoding | | Message Object |
| JMS_IBM_Character_Set | CodedCharacterSetId | | Message Object |

## send()/publish() の際の JMS ヘッダー・フィールドのマッピング

以下の注記は、send()/publish() の際の JMS フィールドのマッピングと関連があります。 (なお、矢印の向きはマッピングの方向を示します。)

- **JMS Destination → MQRFH2:** これは、宛先オブジェクトの顕著な特性を直列化するストリングとして保管されるので、受信 JMS は等価の宛先オブジェクトを再構成することができます。 MQRFH2 フィールドは、URI としてエンコードされます (URI 表記の詳細については、 192ページの一様リソース ID を参照してください)。
- **JMSReplyTo → MQMD ReplyToQ, ReplyToQMgr, MQRFH2:** Queue および QueueManager 名は、それぞれ、MQMD ReplyToQ および ReplyToQMgr フィールドにコピーされます。宛先拡張情報 (Destination Object 内に保管されているその他の役立つ情報) は、 MQRFH2 フィールドにコピーされます。 MQRFH2 フィールドは、URI としてエンコードされます (URI 表記の詳細については、 192ページの一様リソース ID を参照してください)。
- **JMSDeliveryMode → MQMD Persistence:** Destination Object がオーバーライドしない限り、 JMSDeliveryMode 値は、send/publish() メソッドまたは MessageProducer によって設定されます。 JMSDeliveryMode 値は、以下のように MQMD Persistence フィールドにマップされます。
  - JMS 値 PERSISTENT は MQPER_PERSISTENT と等価です。
  - JMS 値 NON_PERSISTENT は MQPER_NOT_PERSISTENT と等価です。

  JMSDeliveryMode が非デフォルト値に設定される場合、送達モード値は、MQRFH2 内にもエンコードされます。
- **JMSExpiration ←/→ MQMD Expiry, MQRFH2:** JMSExpiration は、満了時間 (現行時間と存続時間の合計) を保管しますが、MQMD は存続時間を保管します。また、JMSExpiration はミリ秒単位ですが、MQMD.expiry は 1/100 秒単位です。
  - send() メソッドが無制限の存続時間を設定する場合、 MQMD Expiry は MQEI_UNLIMITED に設定され、JMSExpiration は MQRFH2 にエンコードされません。
  - send() メソッドが 214748364.7 秒 (約 7 年) より少ない存続時間を設定する場合、存続時間は MQMD 内に保管されます。 Expiry および満了時間 (ミリ秒単位) は、MQRFH2 内で i8 としてエンコードされます。
  - send() メソッドが 214748364.7 秒より長い存続時間を設定する場合、MQMD.Expiry は MQEI_UNLIMITED に設定されます。正確な満了時間 (ミリ秒単位) は、MQRFH2 内で i8 値としてエンコードされます。
- **JMSPriority → MQMD Priority:** JMSPriority 値 (0 ～ 9) を MQMD 優先順位値 (0 ～ 9) に直接マップします。 JMSPriority が非デフォルト値に設定される場合、優先順位も MQRFH2 内でエンコードされます。
- **JMSMessageID ← MQMD MessageID:** JMS から送信されるすべてのメッセージは、 MQSeries によって割り当てられる固有のメッセージ ID を持っています。割り当てられた値は、MQPUT 呼び出し後に MQMD messageId フィールド内に戻され、 JMSMessageID フィールド内のアプリケーションに渡されます。 MQSeries messageId は 24 バイトのバイナリー値ですが、JMSMessageID はストリングです。 JMSMessageID は、文字 'ID:' という接頭部を持つ、 48 個の 16 進文字のシーケンスに変換されたバイナリーの messageId 値から成り立ていま

す。 JMS は、メッセージ ID の生成を不可に設定できるヒントを提供します。このヒントは無視され、すべての場合に固有 ID が割り当てられます。 send() が上書きされる前に JMSMessageId フィールドに任意の値が設定されます。

- **JMSTimestamp ← MQMD PutDate, PutTime:** 送信後、 JMSTimestamp フィールドには、MQMD PutDate および PutTime フィールドに指定されている日付 / 時刻値に等しい値が設定されます。 send() が上書きされる前に JMSMessageId フィールドに任意の値が設定されます。
- **JMSType → MQRFH2:** このストリングは MQRFH2 に設定されます。
- **JMSCorrelationID → MQMD CorrelId, MQRFH2:** JMSCorrelationID は、以下のうちの 1 つを保持することができます。
  - **プロバイダー特定のメッセージ ID:** これは、前もって送信されたか受信されたメッセージからのメッセージ ID なので、 'ID:' という接頭部を持つ 48 個の 16 進数字のストリングでなければなりません。接頭部は除去され、残りの文字はバイナリーに変換されてから、MQMD CorrelId フィールドに設定されます。 correlid 値は、MQRFH2 でエンコードされます。
  - **プロバイダー・ネイティブの byte[] 値:** 値は、 MQMD CorrelId フィールドにコピーされ、必要であれば、 24 バイトまでヌルで埋め込まれるか、または切り捨てられます。 correlid 値は、MQRFH2 でエンコードされます。
  - **アプリケーション特定のストリング:** 値は、MQRFH2 にコピーされます。ストリングの最初の 24 バイト (UTF8 形式) が、MQMD CorrelID に書き込まれます。

## JMS プロパティー・フィールドのマッピング

以下の注記は、JMS プロパティー・フィールドの MQSeries メッセージへのマッピングを示しています。 (なお、矢印の向きはマッピングの方向を示します。)

- **JMSXUserID ← MQMD UserIdentifier:** JMSXUserID は、送信呼び出しからの戻り時に設定されます。
- **JMSXAppID ← MQMD PutApplName:** JSMXAppID は、送信呼び出しの戻り時に設定されます。
- **JMSXGroupID → MQRFH2 (ポイント・ツー・ポイント):** ポイント・ツー・ポイント・メッセージの場合、 JMSXGroupID は、MQMD GroupID フィールドにコピーされます。 JMSXGroupID が接頭部 'ID:' で始まる場合には、バイナリーに変換されます。そうでない場合、UTF8 ストリングとしてエンコードされます。必要であれば、値は 24 バイトの長さに埋め込まれるか、または切り捨てられます。 MQF_MSG_IN_GROUP フラグが立てられます。
- **JMSXGroupID → MQRFH2 (パブリッシュ / サブスクライブ):** パブリッシュ / サブスクライブ・メッセージの場合、 JMSXGroupID はストリングとして MQRFH2 にコピーされます。
- **JMSXGroupSeq → MQMD MsgSeqNumber (ポイント・ツー・ポイント):** ポイント・ツー・ポイント・メッセージの場合、 JMSXGroupSeq は MQMD MsgSeqNumber フィールドにコピーされます。 MQF_MSG_IN_GROUP フラグが立てられます。
- **JMSXGroupSeq → MQMD MsgSeqNumber (パブリッシュ / サブスクライブ):** パブリッシュ / サブスクライブ・メッセージの場合、 JMSXGroupSeq は i4 として MQRFH2 にコピーされます。

## JMS プロバイダー特定のフィールドのマッピング

以下の注記は、JMS Provider フィールドの MQSeries メッセージへのマッピングを示しています。

- **JMS_IBM_Report_<name> → MQMD Report:** JMS アプリケーションは、以下の JMS_IBM_Report_XXX プロパティーを使用して、MQMD Report オプションを設定することができます。単一の MQMD は、いくつかの JMS_IBM_Report_XXX プロパティーにマップされます。アプリケーションは、標準 MQSeries MQRO_ 定数 (com.ibm.mq.MQC に組み込まれている) にこれらのプロパティーの値を設定する必要があります。したがって、たとえば、十分なデータを持つ COD を要求するには、アプリケーションは、 JMS_IBM_Report_COD を MQC.MQRO_COD_WITH_FULL_DATA の値に設定しなければなりません。

  **JMS_IBM_Report_Exception**

  MQRO_EXCEPTION または
  MQRO_EXCEPTION_WITH_DATA または
  MQRO_EXCEPTION_WITH_FULL_DATA

  **JMS_IBM_Report_Expiration**

  MQRO_EXPIRATION または
  MQRO_EXPIRATION_WITH_DATA または
  MQRO_EXPIRATION_WITH_FULL_DATA

  **JMS_IBM_Report_COA**

  MQRO_COA または
  MQRO_COA_WITH_DATA または
  MQRO_COA_WITH_FULL_DATA

  **JMS_IBM_Report_COD**

  MQRO_COD または
  MQRO_COD_WITH_DATA または
  MQRO_COD_WITH_FULL_DATA

  **JMS_IBM_Report_PAN**
  MQRO_PAN

  **JMS_IBM_Report_NAN**
  MQRO_NAN

  **JMS_IBM_Report_Pass_Msg_ID**
  MQRO_PASS_MSG_ID

  **JMS_IBM_Report_Pass_Correl_ID**
  MQRO_PASS_CORREL_ID

  **JMS_IBM_Report_Discard_Msg**
  MQRO_DISCARD_MSG

- **JMS_IBM_MsgType → MQMD MsgType:** 値は、直接 MQMD MsgType にマップします。アプリケーションが JMS_IBM_MsgType の正確な値を設定していない場合には、デフォルト値が使用されます。このデフォルト値は、以下のように決定されます。

  - JMSReplyTo が MQSeries キュー宛先に設定される場合、 MSGType は値 MQMT_REQUEST に設定されます。
  - JMSReplyTo が設定されていないか、MQSeries キュー宛先以外の宛先に設定されている場合には、 MsgType は値 MQMT_DATAGRAM に設定されます。
- **JMS_IBM_Feedback → MQMD Feedback:** 値は、直接 MQMD Feedback にマップします。
- **JMS_IBM_Format → MQMD Format:** 値は、直接 MQMD Format にマップします。
- **JMS_IBM_Encoding → MQMD Encoding:** 設定される場合、このプロパティーは Destination Queue または Topic の数値エンコードをオーバーライドします。
- **JMS_IBM_Character_Set → MQMD CodedCharacterSetId:** 設定される場合、このプロパティーは、 Destination Queue または Topic のコード化文字セット・プロパティーをオーバーライドします。

## MQSeries フィールドの JMS フィールドへのマッピング (着信メッセージ)

表21 は、ヘッダー / プロパティー・フィールドが send() または publish() 時に MQMD/MQRFH2 フィールドにマップされる方法を示しています。

*表 21. 着信メッセージ・フィールドのマッピング*

| **JMS フィールド** | **検索元** | |
|---|---|---|
| **名前** | **MQMD フィールド** | **MQRFH2** |
| **JMS ヘッダー** | | |
| JMSDestination | | jms.Dst |
| JMSDeliveryMode | Persistence | |
| JMSExpiration | | jms.Exp |
| JMSPriority | Priority | |
| JMSMessageID | MessageID | |
| JMSTimestamp | PutDate PutTime | |
| JMSCorrelationID | CorrelId | jms.Cid |
| JMSReplyTo | ReplyToQ ReplyToQMgr | jms.Rto |
| JMSType | | mcd.Type |
| JMSRedelivered | BackoutCount | |
| **JMS プロパティー** | | |
| JMSXUserID | UserIdentifier | |
| JMSXAppID | PutApplName | |
| JMSXDeliveryCount | BackoutCount | |
| JMSXGroupID | GroupId | jms.Gid |
| JMSXGroupSeq | MsgSeqNumber | jms.Seq |
| **JMS プロバイダー特定** | | |
| JMS_IBM_Report_Exception | Report | |
| JMS_IBM_Report_Expiration | Report | |
| JMS_IBM_Report_COA | Report | |

*表 21. 着信メッセージ・フィールドのマッピング (続き)*

| JMS フィールド | 検索元 | |
|---|---|---|
| 名前 | MQMD フィールド | MQRFH2 |
| JMS_IBM_Report_COD | Report | |
| JMS_IBM_Report_PAN | Report | |
| JMS_IBM_Report_NAN | Report | |
| JMS_IBM_Report_ Pass_Msg_ID | Report | |
| JMS_IBM_Report_Pass_Correl_ID | Report | |
| JMS_IBM_Report_Discard_Msg | Report | |
| JMS_IBM_MsgType | MsgType | |
| JMS_IBM_Feedback | Feedback | |
| JMS_IBM_Format | Format | |
| JMS_IBM_PutApplType | PutApplType | |
| JMS_IBM_Encoding[1] | Encoding | |
| JMS_IBM_Character_Set[1] | CodedCharacterSetId | |
| 1. 着信メッセージが Bytes Message である場合にのみ設定されます。 | | |

## JMS からネイティブ MQSeries アプリケーションへのマッピング

このセクションでは、JMS Client アプリケーションから、 MQRFH2 ヘッダーを認識しない従来の MQSeries アプリケーションにメッセージを送信する場合に起こる事柄について説明します。 228ページの図5 は、マッピングのダイアグラムです。

アドミニストレーターは、 MQSeries Destination の TargetClient 値を JMSC.MQJMS_CLIENT_NONJMS_MQ に設定することによって、 JMS Client がそのようなアプリケーションと通信していることを示します。これは、MQRFH2 フィールドが生成されないことを示しています。

JMS から、ネイティブ MQSeries アプリケーションで宛先となっている MQMD へのマッピングは、 JMS から、実際の JMS クライアントで宛先となっている MQMD へのマッピングと同じです。 JMS が MQFMT_RFH2 以外に設定されている MQMD Format フィールドを含む MQSeries メッセージを受信する場合には、非 JMS アプリケーションからデータを受信しているということです。 Format が MQFMT_STRING である場合、メッセージは JMS Text Message として受信されます。そうでない場合、メッセージは、JMS Bytes Message として受信されます。 MQRFH2 がないため、MQMD 内に伝送されるそれらの JMS プロパティーのみを復元することができます。

JMS メッセージ・モデルから従来の MQSeries アプリケーションへのマッピング

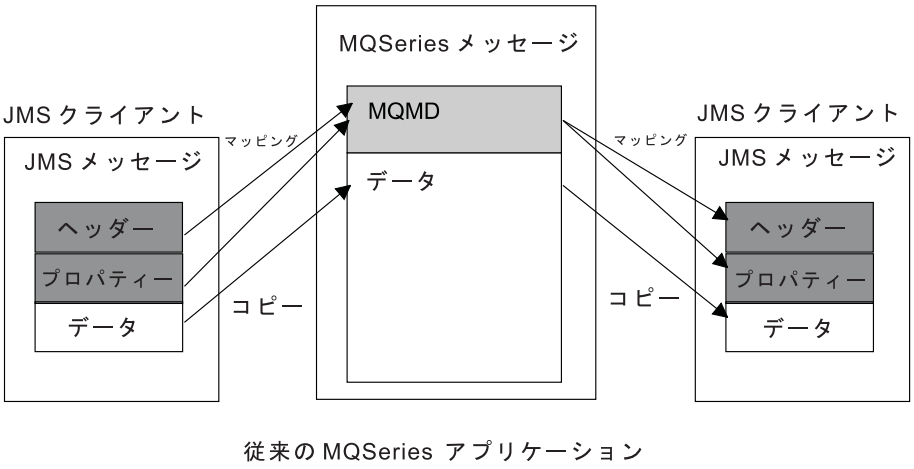


*図 5. JMS から MQSeries へのマッピング・モデル*

## メッセージ本体

このセクションでは、メッセージ本体自体のエンコードについて説明します。エンコードは、JMS メッセージのタイプによって異なります。

**ObjectMessage**

通常の方法で、 Java Runtime によって直列化されるオブジェクトです。

**TextMessage**

エンコードされたストリングです。出力メッセージの場合、ストリングは、Destination オブジェクトによって提供されている文字セットにエンコードされます。これは、デフォルトで UTF8 エンコードになります (UTF8 エンコードは、メッセージの最初の文字で始まります。先頭に長さフィールドはありません)。ただし、MQ Java によってサポートされているその他の文字セットを指定することができます。そのような文字セットは、主に、非 JMS アプリケーションにメッセージを送信するときに使用されます。

文字セットが 2 バイト・セット (UTF16 を含む) である場合、 Destination オブジェクトの整数エンコード仕様により、バイト順が決定されます。

着信メッセージは、メッセージ自体に指定されている文字セットおよびエンコードを使用して解釈されます。これらの指定は、右端の MQSeries ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、右端のヘッダーは、通常、MQRFH2 になります。

**BytesMessage**

デフォルトでは、 JMS 1.0.2 仕様および関連のある Java 資料で定義されているように、バイトのシーケンスです。

アプリケーション自体によってアセンブルされた出力メッセージの場合、Destination オブジェクトのエンコード・プロパティーは、メッセージ内に含まれている整数および浮動小数点フィールドのエンコードをオーバーライド

するために使用されることがあります。たとえば、浮動小数点値は IEEE 形式ではなく S/390 形式で保管されることを要求することができます。

着信メッセージは、メッセージ自体に指定されている数値エンコードを使用して解釈されます。この指定は、右端の MQSeries ヘッダー (またはヘッダーがない場合には MQMD) 内にあります。 JMS メッセージの場合、右端のヘッダーは、通常、MQRFH2 になります。

BytesMessage が受信され、変更を加えることなく再送信される場合、メッセージ本体は、受信されたとおりに、バイトごとに送信されます。Destination オブジェクトのエンコード・プロパティーはメッセージ本体に影響を与えません。 BytesMessage 内で明示的に送信できる唯一のストリング系エンティティーは、UTF8 ストリングです。これは、Java UTF8 形式でエンコードされ、2 バイトの長さフィールドで始まります。 Destination オブジェクトの文字セット・プロパティーは、出力 BytesMessage のエンコードに影響を与えません。着信 MQSeries メッセージ内の文字セット値は、JMS BytesMessage としてそのメッセージの解釈に影響を与えません。

非 Java アプリケーションは、普通、 Java UTF8 エンコードを認識しません。したがって、テキスト・データを含む BytesMessage を送信する JMS アプリケーションの場合、アプリケーション自体は、そのストリングをバイト配列に変換し、これらのバイト配列を BytesMessage に書き込まなければなりません。

### MapMessage

一連の XML name/type/value トリプレットを含むストリングで、以下のようにエンコードされます。

```
<map><elementName1 dt='datatype'>value</elementName1>
<elementName2 dt='datatype'>value</elementName2>.....
</map>
```

ここで、それぞれは次のものを表します。

`datatype` は、220ページの表18 で説明されている値のいずれかをとることができます。

`string` はデフォルトのデータ型なので、`dt='string'` は省略されます。

MapMessage 本体を構成する XML ストリングをエンコードまたは解釈するために使用される文字セットは、 TextMessage に適用される規則に従って決定されます。

### StreamMessage

マップのようなものですが、エレメント名を持っていません。

```
<stream><elt dt='datatype'>value</elt>
<elt dt='datatype'>value</elt>.....</stream>
```

すべてのエレメントは、同じタグ名 (elt) を使用して送信されます。デフォルトのタイプはストリングなので、ストリング・エレメントの場合、`dt='string'` は省略されます。

StreamMessage 本体を構成する XML ストリングをエンコードまたは解釈するために使用される文字セットは、 TextMessage に適用される規則に従って決定されます。

MQRFH2.format フィールドは、以下のように設定されます。

**MQFMT_NONE**

ObjectMessage、BytesMessage、または本体がないメッセージの場合。

**MQFMT_STRING**

TextMessage、StreamMessage、または MapMessage の場合。

# 第13章 MQ JMS アプリケーション・サーバー機構

MQ JMS V5.2 は、Java Message Service 1.0.2 仕様で指定されているアプリケーション・サーバー機構 (ASF) をサポートします (`http://java.sun.com` にある Sun 社の Java Web サイトを参照してください)。この仕様では、このプログラミング・モデル内で以下の 3 つの役割を識別します。

- **JMS プロバイダー**は、 ConnectionConsumer および拡張セッション機能を提供します。
- **アプリケーション・サーバー**は、 ServerSessionPool および ServerSession 機能を提供します。
- **クライアント・アプリケーション**は、 JMS プロバイダーおよびアプリケーション・サーバーが提供する機能を使用します。

以下のセクションには、MQ JMS が ASF をインプリメントする方法に関する詳細が記載されています。

- 『ASF クラスおよび関数』 では、 MQ JMS が Session クラス内の ConnectionConsumer クラスおよび拡張機能をインプリメントする方法を説明します。
- 240ページの『アプリケーション・サーバーのサンプル・コード』 は、 MQ JMS で提供されているサンプル ServerSessionPool および ServerSession コードについて説明します。
- 243ページの『ASF の使用の例』 では、クライアント・アプリケーションの観点から、提供されている ASF サンプルおよび ASF の使用例について説明します。

**注:** ASF 用の Java Message Service 1.0.2 仕様では、 X/Open XA プロトコルを使用する分散トランザクションのための JMS サポートについても述べています。MQ JMS が提供する XA サポートの詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

## ASF クラスおよび関数

MQ JMS は、Session クラス内の ConnectionConsumer クラスおよび拡張機能をインプリメントします。詳細については、以下を参照してください。

- 138ページの『MQPoolServices』
- 139ページの『MQPoolServicesEvent』
- 141ページの『MQPoolToken』
- 171ページの『MQPoolServicesEventListener』
- 270ページの『ConnectionConsumer』
- 316ページの『QueueConnection』
- 329ページの『Session』
- 347ページの『TopicConnection』

## ConnectionConsumer

JMS 仕様では、アプリケーション・サーバーは、 ConnectionConsumer インターフェースを使用することによって、 JMS インプリメンテーションと密接に統合することができます。この機能は、メッセージの並行処理を提供します。通常、アプリケーション・サーバーがスレッドのプールを作成し、 JMS インプリメンテーションがこれらのスレッドに使用可能なメッセージを作成します。 JMS 対応のアプリケーション・サーバーは、メッセージ処理 bean など、高水準メッセージング機能を提供するためにこの機能を使用することができます。

通常のアプリケーションは ConnectionConsumer を使用しませんが、熟練した JMS クライアントは ConnectionConsumer を使用することがあります。そのようなクライアントにとって、ConnectionConsumer は、スレッドのプールに同時にメッセージを送達するための高性能のメソッドを提供します。メッセージがキューまたはトピックに到着するとき、 JMS はプールからスレッドを選択し、そのスレッドにメッセージのバッチを送達します。これを行うために JMS は、関連した MessageListener の `onMessage()` メソッドを実行します。

複数の Session および MessageConsumer オブジェクト (それぞれ登録された MessageListener を持つ) を構成することによって、同じ効果を上げることができます。ただし、ConnectionConsumer の方が、パフォーマンスが良く、リソースの使用量が少なく、より大きな柔軟性があります。特に、Session オブジェクトが少なくて済みます。

ConnectionConsumer を使用するアプリケーションの開発を助けるために、 MQ JMS は、完全に機能する、プールのサンプル・インプリメンテーションを提供します。何の変更も行わないでこのインプリメンテーションを使用することもでき、アプリケーションの特定の必要を満たすように調整することもできます。

# アプリケーションの計画

### ポイント・ツー・ポイント・メッセージングの一般原則

アプリケーションが QueueConnection オブジェクトから ConnectionConsumer を作成するとき、 JMS Queue オブジェクトおよびセレクター・ストリングを指定します。その後、ConnectionConsumer は、メッセージの受信を開始します (または、より正確に言えば、関連した ServerSessionPool 内の Session にメッセージの提供を開始します)。メッセージがキューに到着し、セレクターと一致する場合、メッセージは、関連のある ServerSessionPool 内の Session に送達されます。

MQSeries 用語では、Queue オブジェクトは、ローカル・キュー・マネージャー上にある QLOCAL または QALIAS のいずれかを表します。 QALIAS である場合には、その QALIAS は QLOCAL を参照しなければなりません。完全に解決された MQSeries QLOCAL を、*基礎 QLOCAL* と言います。 ConnectionConsumer は、クローズされておらずその親 QueueConnection が開始されている場合には、アクティブであると言います。

複数の ConnectionConsumer (それぞれ異なるセレクターを持つ) を、同じ基礎 QLOCAL に対して実行することは可能です。パフォーマンスを保つために、不必要なメッセージをキュー上に累積しないようにしてください。不必要なメッセージとは、アクティブな ConnectionConsumer が一致するセレクターを持たないメッセージ

のことです。これらの不必要なメッセージがキューから除去されるように QueueConnectionFactory を設定することができます (詳細については、 236ページの『キューからのメッセージの除去』を参照してください)。以下の 2 つの方法のどちらかで、この動作を設定することができます。

- JMS 管理ツールを使用して、QueueConnectionFactory を MRET(NO) に設定します。
- ご使用のプログラムで、以下のものを使用します。

  ```
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

この設定を変更しない場合、デフォルトでは、キュー上にそのような不必要なメッセージを保存することになっています。

同じ基礎 QLOCAL を宛先とする ConnectionConsumer を、複数の QueueConnection オブジェクトから作成することは可能です。ただし、パフォーマンスのために、複数の JVM が同じ基礎 QLOCAL に対して ConnectionConsumer を作成しないことをお勧めします。

MQSeries キュー・マネージャーをセットアップするとき、以下の点を考慮してください。

- 基礎 QLOCAL は、共用入力のために使用可能でなければなりません。これを行うには、以下の MQSC コマンドを使用します。

  ```
  ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)
  ```

- キュー・マネージャーは、使用可能な送達不能キューを持っていなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れるときに問題に遭遇する場合、基礎 QLOCAL からのメッセージ送達は停止します。送達不能キューを定義するには、以下のものを使用します。

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- ConnectionConsumer を実行するユーザーは、 MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を持っていなければなりません。詳細については、特定のプラットフォーム用の MQSeries 資料を参照してください。
- 不必要なメッセージがキュー上に残されている場合、システム・パフォーマンスが低下します。したがって、メッセージ・セレクター間で、 ConnectionConsumer がキューからすべてのメッセージを除去するように、メッセージ・セレクターを計画してください。

MQSC コマンドの詳細については、*MQSeries MQSC コマンド・リファレンス*を参照してください。

## パブリッシュ / サブスクライブ・メッセージングの一般原則

アプリケーションが TopicConnection オブジェクトから ConnectionConsumer を作成するとき、アプリケーションは、Topic オブジェクトおよびセレクター・ストリングを指定します。その後、ConnectionConsumer は、セレクターと一致するその Topic についてのメッセージを受信し始めます。

あるいは、アプリケーションは、特定の名前と関連のある永続 ConnectionConsumer を作成することもできます。この ConnectionConsumer は、永続 ConnectionConsumer が最後にアクティブだったとき以降に、 Topic についてパブリ

ッシュされているメッセージを受信します。この ConnectionConsumer は、セレクターと一致する Topic に関するすべてのメッセージを受信します。

非永続サブスクリプションの場合、 ConnectionConsumer サブスクリプションに対して別々のキューが使用されます。 TopicConnectionFactory 上の CCSUB 構成可能オプションは、使用するキューを指定します。通常、CCSUB は、同じ TopicConnectionFactory を使用するすべての ConnectionConsumer が使用するための単一キューを指定する必要があります。ただし、それぞれの ConnectionConsumer が、キュー名の接頭部とそれに続く '*' を指定することによって一時キューを生成することは可能です。

永続サブスクリプションの場合、Topic の CCDSUB プロパティーは、使用するキューを指定します。前述のとおり、これはすでに存在するキューか、またはキュー名の接頭部とそれに続く '*' です。すでに存在するキューを指定する場合、 Topic にサブスクライブするすべての永続 ConnectionConsumer は、このキューを使用します。キュー名の接頭部とそれに続く '*' を指定する場合、はじめて永続 ConnectionConsumer が指定の名前で作成されるときにキューが生成されます。このキューは、後に永続 ConnectionConsumer が同じ名前で作成されるときに再使用されます。

MQSeries キュー・マネージャーをセットアップするとき、以下の点を考慮してください。

- キュー・マネージャーは、使用可能な送達不能キューを持っていなければなりません。 ConnectionConsumer が、送達不能キューにメッセージを入れるときに問題に遭遇する場合、基礎 QLOCAL からのメッセージ送達は停止します。送達不能キューを定義するには、以下のものを使用します。

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- ConnectionConsumer を実行するユーザーは、 MQOO_SAVE_ALL_CONTEXT および MQOO_PASS_ALL_CONTEXT を伴う MQOPEN を実行するための権限を持っていなければなりません。詳細については、特定のプラットフォーム用の MQSeries 資料を参照してください。
- 個々の ConnectionConsumer 用に、別々の専用キューを作成することによって、パフォーマンスを最適化することができます。この場合、余分のリソースが使用されることになります。

## ポイズン・メッセージの処理

時折、ひどいフォーマットのメッセージがキューに到着することがあります。そのようなメッセージによって、受信アプリケーションに障害が起き、メッセージの受信がバックアウトすることがあります。この状況では、そのようなメッセージを受信した後、繰り返しキューに戻されることがあります。これらのメッセージを、ポイズン・メッセージ と言います。 ConnectionConsumer は、ポイズン・メッセージを検出し、それらのメッセージを代替宛先に転送できなければなりません。

アプリケーションが ConnectionConsumer を使用するとき、メッセージがバックアウトされる状況は、アプリケーション・サーバーが提供する Session によって異なります。

- Session が非トランザクションで AUTO_ACKNOWLEDGE または DUPS_OK_ACKNOWLEDGE を伴う場合、メッセージは、システム・エラーの後、またはアプリケーションが予期せずに終了する場合にのみバックアウトされます。
- Session が非トランザクションで CLIENT_ACKNOWLEDGE を伴う場合、確認されないメッセージは、 `Session.recover()` を呼び出すアプリケーション・サーバーによってバックアウトできます。

  通常、MessageListener またはアプリケーション・サーバーのクライアント・インプリメンテーションは、 `Message.acknowledge()` を呼び出します。`Message.acknowledge()` は、これまでセッション上に送達されたすべてのメッセージを確認します。
- 通常、Session がトランザクションの場合、アプリケーション・サーバーは Session をコミットします。アプリケーション・サーバーがエラーを検出する場合、アプリケーション・サーバーは、 1 つまたは複数のメッセージをバックアウトすることを選択することがあります。
- アプリケーション・サーバーが XASession を提供する場合、メッセージは分散トランザクション次第でコミットまたはバックアウトされます。アプリケーション・サーバーは、トランザクションを完了させる責任があります。

MQSeries キュー・マネージャーは、それぞれのメッセージがバックアウトされた回数のレコードを保持します。この数が構成可能なしきい値に達するとき、ConnectionConsumer は、指名された Backout Queue 上にメッセージをリキューします。このリキューが何らかの理由で失敗する場合、メッセージはキューから除去され、送達不能キューにリキューされるか、または破棄されます。詳細については、236ページの『キューからのメッセージの除去』を参照してください。

ほとんどのプラットフォームで、しきい値およびリキュー・キューは、MQSeries QLOCAL のプロパティーです。ポイント・ツー・ポイント・メッセージングの場合、これは基礎 QLOCAL のはずです。パブリッシュ / サブスクライブ・メッセージングの場合、これは、TopicConnectionFactory 上に定義されている CCSUB キュー、または Topic 上に定義されている CCDSUB キューです。しきい値およびリキュー Queue プロパティーを設定するには、以下の MQSC コマンドを発行してください。

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

パブリッシュ / サブスクライブ・メッセージングの場合、システムがそれぞれのサブスクリプションごとに動的キューを作成すると、これらの設定値は MQ JMS モデル・キューから入手されます。これらの設定値を更新するには、以下のものを使用できます。

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

しきい値がゼロである場合、ポイズン・メッセージの処理はできなくなり、ポイズン・メッセージは入力キュー上に残ります。そうでない場合、バックアウト・カウントがしきい値に達したとき、メッセージは、指名されたリキュー・キューに送信されます。バックアウト・カウントがしきい値に達しても、メッセージがリキュー・キューに入ることができない場合、メッセージは送達不能キューに送信されるか、または破棄されます。この状態は、リキュー・キューが定義されていない場合、または ConnectionConsumer がリキュー・キューにメッセージを送信できない場

合に発生します。一部のプラットフォームでは、しきい値およびリキュー・キュー・プロパティーを指定することができません。これらのプラットフォームでは、バックアウト・カウントが 20 に達するときに、メッセージは送達不能キューに送信されるか、または破棄されます。詳細については、『キューからのメッセージの除去』を参照してください。

## キューからのメッセージの除去

アプリケーションが ConnectionConsumer を使用するとき、いくつかの状況では、JMS は、キューからメッセージを除去する必要があります。

**ひどいフォーマットのメッセージ**
JMS が解析できないメッセージが到着することがあります。

**ポイズン・メッセージ**
メッセージがバックアウトしきい値に達したのに、 ConnectionConsumer は、バックアウト・キュー上にメッセージをリキューすることに失敗します。

**関心のない ConnectionConsumer**
ポイント・ツー・ポイント・メッセージングの場合、QueueConnectionFactory が不必要なメッセージを保存しないように設定されているとき、どの ConnectionConsumer にも不必要なメッセージが到着します。

これらの状態では、ConnectionConsumer は、キューからメッセージを除去しようとします。メッセージの MQMD のレポート・フィールド内の後処理オプションは、正確な動作を設定します。これらのオプションは、以下のとおりです。

**MQRO_DEAD_LETTER_Q**
メッセージは、キュー・マネージャーの送達不能キューにリキューされます。これはデフォルトです。

**MQRO_DISCARD_MSG**
メッセージは破棄されます。

また、ConnectionConsumer はレポート・メッセージも生成しますが、これはメッセージの MQMD のレポート・フィールドにも依存しています。このメッセージは、ReplyToQmgr 上のメッセージの ReplyToQ に送信されます。レポート・メッセージが送信されている間にエラーがある場合、メッセージは、代わりに、送達不能キューに送信されます。メッセージの MQMD のレポート・フィールド内の例外レポート・オプションは、レポート・メッセージの詳細を設定します。これらのオプションは、以下のとおりです。

**MQRO_EXCEPTION**
オリジナル・メッセージの MQMD を含むレポート・メッセージが生成されます。このメッセージには、いかなるメッセージ本体データも含まれていません。

**MQRO_EXCEPTION_WITH_DATA**
MQMD、任意の MQ ヘッダー、および 100 バイトの本体データを含むレポート・メッセージが生成されます。

**MQRO_EXCEPTION_WITH_FULL_DATA**
オリジナル・メッセージからのすべてのデータを含むレポート・メッセージが生成されます。

**default**
レポート・メッセージは生成されません。

レポート・メッセージが生成されるとき、以下のオプションが有効です。

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

ConnectionConsumer が後処理オプションまたは例外レポート・オプションに従うことができない場合、メッセージの MQMD では、そのアクションは、メッセージの永続性によって異なります。メッセージが非永続である場合、メッセージは破棄され、レポート・メッセージは生成されません。メッセージが永続である場合、QLOCAL からのすべてのメッセージの送達は停止します。

したがって、送達不能キューを定義し、定期的にチェックして、問題が発生していないか確認することは重要です。特に、送達不能キューがその最大の深さに達していないことや、その最大メッセージ・サイズがすべてのメッセージに対して十分な大きさであることを確認してください。

メッセージが送達不能キューにリキューされるとき、メッセージの前に MQSeries 送達不能ヘッダー (MQDLH) が付けられます。 MQDLH のフォーマットの詳細については、*MQSeries アプリケーション・プログラミング・リファレンス*を参照してください。以下のフィールドによって、ConnectionConsumer が送達不能キューに入れたメッセージを識別したり、 ConnectionConsumer が生成したメッセージを報告することができます。

- PutApplType は、`MQAT_JAVA (0x1C)` です。
- PutApplName は、“`MQ JMS ConnectionConsumer`” です。

これらのフィールドは、送達キュー上のメッセージの MQDLH 内およびレポート・メッセージの MQMD 内にあります。 MQMD のフィードバック・フィールド、およびMQDLH の Reason フィールドには、エラーを説明しているコードが含まれています。これらのコードの詳細については、『『エラー処理』』を参照してください。その他のフィールドは、*MQSeries アプリケーション・プログラミング・リファレンス* で説明されているとおりです。

# エラー処理

## エラー状態からの回復

ConnectionConsumer が重大エラーに遭遇する場合、同じ QLOCAL 内にインタレストを持つすべての ConnectionConsumer へのメッセージ送達は停止します。通常、これは、ConnectionConsumer が送達不能キューにメッセージをリキューできない場合に発生するか、または QLOCAL からメッセージを読み取るときにエラーに遭遇します。

これが発生すると、アプリケーションおよびアプリケーション・サーバーは、以下の方法で通知されます。

- 影響を受けた Connection とともに登録されている ExceptionListener が通知されます。

問題の原因を識別するために、これらを使用することができます。場合によっては、問題解決のためにシステム管理者が介入しなければなりません。

アプリケーションがこれらのエラー状態から回復できる 2 つの方法があります。

- 影響を受けたすべての ConnectionConsumer で `close()` を呼び出します。アプリケーションは、影響を受けたすべての ConnectionConsumer がクローズされ、システム問題が解決された後にのみ、新しい ConnectionConsumer を作成することができます。
- 影響を受けたすべての Connection で `stop()` を呼び出します。いったんすべての Connection が停止され、システム問題が解決されると、アプリケーションは、正常にすべての Connection を `start()` できます。

## 理由コードおよびフィードバック・コード

エラーの原因を判別するには、以下のものを使用することができます。

- レポート・メッセージ内のフィードバック・コード
- 送達不能キュー内のメッセージの MQDLH 内にある理由コード

ConnectionConsumer は、以下の理由コードを生成します。

**MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)**

**原因** メッセージは、QLOCAL 上に定義されている Backout Threshold に達しましたが、 Backout Queue は定義されていません。

Backout Queue を定義できないプラットフォームでは、メッセージは、20 という JMS 定義のバックアウトしきい値に達します。

**処置** この状態を回避するには、キューを使用している ConnectionConsumer がすべてのメッセージを処理する一連のセレクターを提供するようにするか、または QueueConnectionFactory を設定してメッセージを保存するようにしてください。

または、メッセージの送信元を調べてください。

**MQRC_MSG_NOT_MATCHED (0x93B; 2363)**

**原因** ポイント・ツー・ポイント・メッセージングで、キューをモニターしている ConnectionConsumer 用のセレクターのいずれかと一致しないメッセージがあります。パフォーマンスを保つために、メッセージが送達不能キューにリキューされます。

**処置** この状態を回避するには、キューを使用している ConnectionConsumer がすべてのメッセージを処理する一連

のセレクターを提供するようにするか、または QueueConnectionFactory を設定してメッセージを保存するようにしてください。

または、メッセージの送信元を調べてください。

**MQRC_JMS_FORMAT_ERROR (0x93C; 2364)**

**原因** JMS は、キュー上のメッセージを解釈できません。

**処置** メッセージの発信源を調べてください。通常、JMS は、BytesMessage または TextMessage として予期しないフォーマットのメッセージを送達します。時折、メッセージが非常にひどいフォーマットである場合には、このアクションは失敗します。

これらのフィールドに表示されるその他のコードは、 Backout Queue にメッセージをリキューする試みが失敗したことが原因です。この状態では、コードは、リキューが失敗した理由を説明しています。これらのエラーの原因を診断するには、*MQSeries アプリケーション・プログラミング・リファレンス* を参照してください。

レポート・メッセージを ReplyToQ に入れることができない場合には、それは送達不能キューに入れられます。この状態では、MQMD のフィードバック・フィールドは、上記で説明されているとおりに埋められます。 MQDLH 内の理由フィールドは、レポート・メッセージを ReplyToQ に入れられなかった理由を説明しています。

# アプリケーション・サーバーのサンプル・コード

図6 は、ServerSessionPool および ServerSession 機能の基本を要約しています。

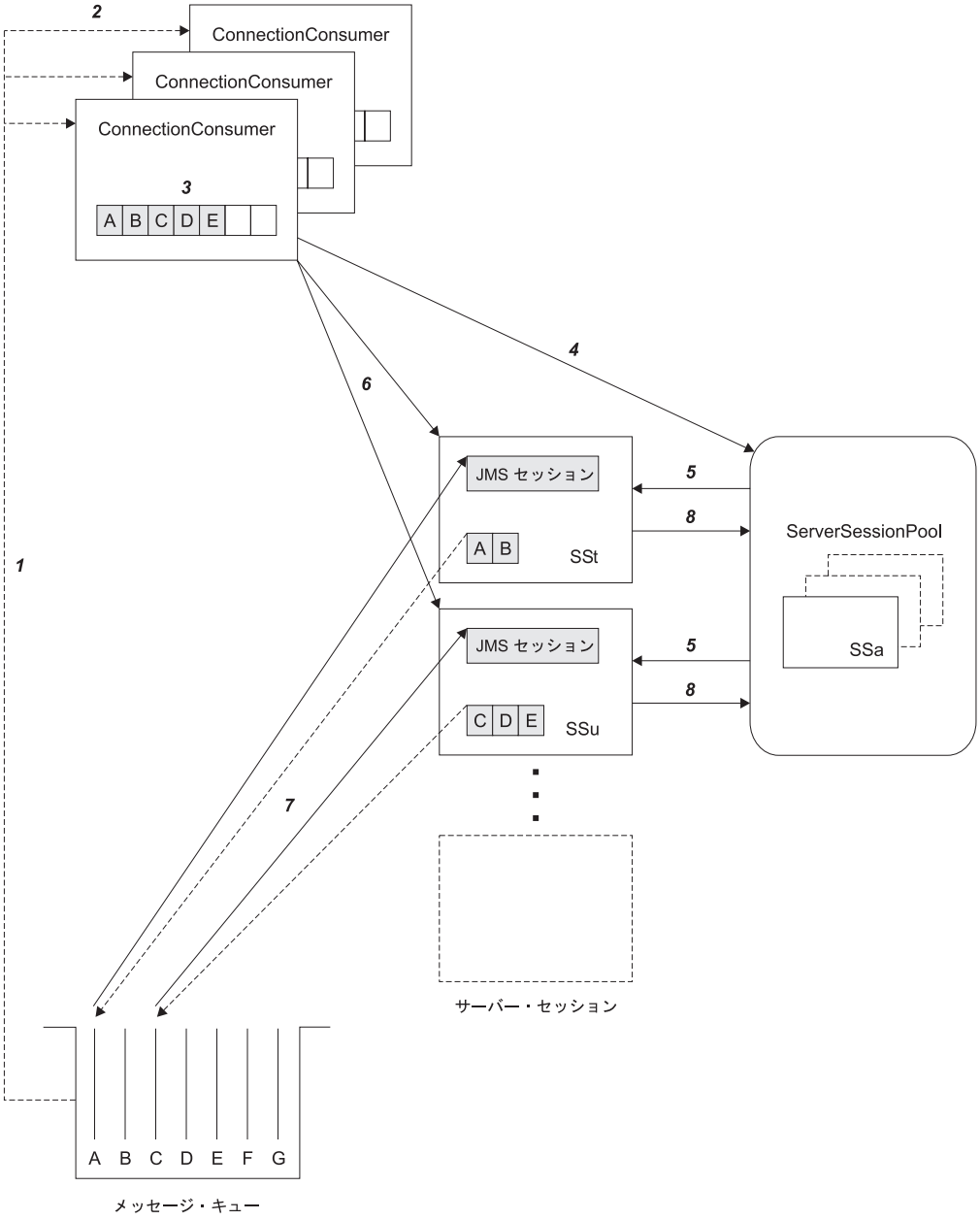


*図 6. ServerSessionPool および ServerSession 機能*

1. ConnectionConsumer は、キューからメッセージ参照を取得します。
2. それぞれの ConnectionConsumer は特定のメッセージ参照を選択します。
3. ConnectionConsumer バッファーは、選択されたメッセージ参照を保持します。
4. ConnectionConsumer は、ServerSessionPool から 1 つまたは複数の ServerSession を要求します。
5. ServerSession は ServerSessionPool から割り振られます。

6. ConnectionConsumer は、ServerSession にメッセージ参照を割り当て、ServerSession スレッドの実行を開始します。
7. それぞれの ServerSession は、キューからその参照されたメッセージを取得します。これは、JMS Session と関連のある MessageListener から onMessage メソッドにメッセージを渡します。
8. その処理を完了した後、ServerSession はプールに戻されます。

通常、アプリケーション・サーバーは、ServerSessionPool および ServerSession の機能を提供します。ただし、MQ JMS は、プログラム・ソースとともに、これらのインターフェースの単純インプリメンテーションで提供されます。これらのサンプルは、以下のディレクトリー内にあります。ここで、<install_dir> は、MQ JMS 用のインストール・ディレクトリーです。

```
<install_dir>/samples/jms/asf
```

これらのサンプルでは、スタンドアロン環境で MQ JMS ASF を使用することができます (つまり、それに適したアプリケーション・サーバーを必要とするわけではありません)。また、これらのサンプルでは、これらのインターフェースをインプリメントし、 MQ JMS ASF を使用する方法の例を提供します。これらの例は、MQ JMS ユーザーと、他のアプリケーション・サーバーのベンダーの両方を援助することを目的としています。

## MyServerSession.java

このクラスは、javax.jms.ServerSession インターフェースをインプリメントします。その基本機能は、JMS Session とスレッドを関連付けることです。このクラスのインスタンスは、 ServerSessionPool によってプールされます (242ページの『MyServerSessionPool.java』を参照してください)。 ServerSession として、以下の 2 つのメソッドをインプリメントしなければなりません。

- getSession()。この ServerSession と関連のある JMS Session を戻します。
- start()。この ServerSession のスレッドを開始し、その結果、JMS Session の run() メソッドが呼び出されます。

MyServerSession は、Runnable インターフェースもインプリメントします。したがって、ServerSession のスレッドの作成はこのクラスに基づいており、別々のクラスを必要としません。

クラスは、2 つのブール・フラグの値 (ready および quit) に基づいている wait() - notify() メカニズムを使用します。このメカニズムは、ServerSession がその構成中に、その関連したスレッドを作成し開始することを意味しています。ただし、run() メソッドの本体を自動的には実行しません。 run() メソッドの本体は、ready フラグが start() メソッドによって true に設定されるときにのみ実行されます。関連した JMS Session にメッセージを送達する必要があるとき、 ASF は start() メソッドを呼び出します。

送達の場合、JMS Session の run() メソッドが呼び出されます。 MQ JMS ASF は、メッセージとともに run() メソッドをすでにロードしています。

送達が完了した後、ready フラグは false にリセットされ、所有している ServerSessionPool は、送達が完了したことを通知されます。その後、ServerSession

は、start() メソッドが再び呼び出されるか、または close() メソッドが呼び出されるまで待ち状態のままになり、この ServerSession のスレッドを終了します。

## MyServerSessionPool.java

このクラスは、javax.jms.ServerSessionPool インターフェースをインプリメントし、 ServerSession のプールへのアクセスを作成し、制御するために存在します。

この単純インプリメンテーションでは、プールは、プールの構成中に作成される ServerSession オブジェクトの静的配列から成り立っています。以下の 4 つのパラメーターがコンストラクターに渡されます。

- javax.jms.Connection *connection*

  JMS Session を作成するために使用される接続。

- int *capacity*

  MyServerSession オブジェクトの配列のサイズ。

- int *ackMode*

  JMS Session の必須確認モード。

- MessageListenerFactory *mlf*

  JMS Session に提供されるメッセージ・リスナーを作成する MesssageListenerFactory。 243ページの『MessageListenerFactory.java』を参照してください。

プールのコンストラクターは、 MyServerSession オブジェクトの配列を作成するためにこれらのパラメーターを使用します。提供されている接続は、指定された確認モードの JMS Session を作成し、ドメイン (ポイント・ツー・ポイントの場合には QueueSession、パブリッシュ / サブスクライブの場合には TopicSession) を訂正するために使用されます。 Session は、メッセージ・リスナーとともに提供されます。最後に、JMS Session に基づく ServerSession オブジェクトが作成されます。

このサンプル・インプリメンテーションは、静的モデルです。つまり、プール内のすべての ServerSession は、プールが作成されるときに作成され、その後、プールは拡大も縮小もできません。このアプローチの目的は単純さです。必要であれば、ServerSessionPool が ServerSession を動的に作成するための洗練されたアルゴリズムを使用することは可能です。

MyServerSessionPool は、inUse と呼ばれるブール値の配列を保守することによって、 ServerSession が現在使用されているレコードを保持します。これらのブール値は、すべて false に初期設定されています。 getServerSession メソッドが呼び出され、プールから ServerSession を要求するとき、inUse 配列は、最初の false 値を検索されます。検出されると、ブール値は true に設定され、対応する ServerSession が戻されます。 inUse 配列内に False 値がない場合、 getServerSession メソッドは、通知が行われるまで、 wait() しなければなりません。

通知は、以下のいずれかの状況で行われます。

- プールがシャットダウンされる必要があることを示す、プールの close() メソッドが呼び出されたとき。
- 現在使用されている ServerSession は、そのワークロードを完了し、serverSessionFinished メソッドを呼び出します。 serverSessionFinished メソ

ッドは、プールに ServerSession を戻し、対応する inUse フラグを false に設定します。その後、ServerSession は再使用可能になります。

### MessageListenerFactory.java

このサンプルでは、メッセージ・リスナー・ファクトリー・オブジェクトは、それぞれの ServerSessionPool インスタンスと関連があります。 MessageListenerFactory クラスは、 `javax.jms.MessageListener` インターフェースをインプリメントするクラスのインスタンスを入手するために使用される、非常に単純なインターフェースを表しています。クラスには、単一のメソッドが含まれています。

```
javax.jms.MessageListener createMessageListener();
```

このインターフェースのインプリメンテーションは、 ServerSessionPool が構成されるときに提供されます。このオブジェクトは、プール内の ServerSession をバックアップする個々の JMS Session 用のメッセージ・リスナーを作成するために使用されます。このアーキテクチャーは、 `MessageListenerFactory` インターフェースの別々のインプリメンテーションが、それぞれの ServerSessionPool を持っていなければならないことを意味しています。

MQ JMS には、MessageListenerFactory インプリメンテーションの例 (245ページの『CountingMessageListenerFactory.java』で説明されている) が含まれています。

## ASF の使用の例

ディレクトリー `<install_dir>/samples/jms/asf` 内にソースを持つ一連のクラスがあります (ここで、 <install_dir> は、MQ JMS 用のインストール・ディレクトリーです)。これらのクラスは、240ページの『アプリケーション・サーバーのサンプル・コード』で説明されているサンプルのスタンドアロン・アプリケーション・サーバー環境内で、 231ページの『ASF クラスおよび関数』で説明されている MQ JMS アプリケーション・サーバー機構を使用します。

これらのサンプルは、クライアント・アプリケーションの観点から ASF の 3 つの使用例を提供しています。

- 単純なポイント・ツー・ポイントの例では、以下のものを使用します。
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- より複雑なポイント・ツー・ポイントの例では、以下のものを使用します。
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- 単純なパブリッシュ / サブスクライブの例では、以下のものを使用します。
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java

- より複雑なパブリッシュ / サブスクライブの例では、以下のものを使用します。
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

以下のセクションでは、それぞれのクラスについて順番に説明します。

## Load1.java

このクラスは、いくつかのメッセージを含む指定されたキューをロードしてから終了する、単純な汎用 JMS アプリケーションです。このクラスは、これらのインターフェースをインプリメントする MQ JMS クラスを使用して、 JNDI ネーム・スペースから必要な管理対象オブジェクトを検索するか、またはそれらのオブジェクトを明示的に作成することができます。 必要な管理対象オブジェクトは、QueueConnectionFactory および Queue です。キューをロードするメッセージ数および個々のメッセージ・プット間のスリープ時間を設定するために、コマンド行オプションを使用することができます。

このアプリケーションには、2 つのバージョンのコマンド行構文があります。

JNDI を指定して使用する場合、構文は以下のとおりです。

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

JNDI を指定しないで使用する場合、構文は以下のとおりです。

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

表22 では、パラメーターについて説明し、それぞれのデフォルトを示します。

*表 22. Load1 のパラメーターおよびデフォルト*

| パラメーター | 意味 | デフォルト |
|---|---|---|
| jndiICF | JNDI に使用される初期コンテキスト・ファクトリー・クラス | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI に使用されるプロバイダー URL | ldap://localhost/o=ibm,c=us |
| qcfLookup | QueueConnectionFactory に使用される JNDI ルックアップ・キー | cn=qcf |
| qLookup | Queue に使用される JNDI ルックアップ・キー | cn=q |
| qMgrName | 接続先のキュー・マネージャーの名前 | "" (デフォルトのキュー・マネージャーを使用する) |
| qName | ロードするキューの名前 | SYSTEM.DEFAULT.LOCAL.QUEUE |
| sleepTime | メッセージ・プット間で休止する時間 (ミリ秒単位) | 0 (休止なし) |
| numMsgs | 書き込むメッセージの数 | 1000 |

エラーがある場合、エラー・メッセージが表示され、アプリケーションは終了します。

MQSeries キュー上のメッセージ・ロードをシミュレートするためにこのアプリケーションを使用することができます。そして、このメッセージ・ロードは、以下のセクションで説明されている ASF 使用可能アプリケーションを起動することができます。キューに書き込まれるメッセージは、単純な JMS TextMessage オブジェクトです。これらのオブジェクトには、ユーザー定義のメッセージ・プロパティーは含まれておらず、異なるメッセージ・リスナーを利用するのに役立ちます。ソース・コードは、必要であれば、このロード・アプリケーションを変更できるように提供されています。

## CountingMessageListenerFactory.java

このファイルには、以下の 2 つのクラスの定義が含まれています。

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener は、 `javax.jms.MessageListener` インターフェースの非常に単純なインプリメンテーションです。 CountingMessageListener は、その `onMessage` メソッドが呼び出された回数のレコードを保持しますが、渡されるメッセージについては何も行いません。

CountingMessageListenerFactory は、CountingMessageListener のファクトリー・クラスです。 CountingMessageListenerFactory は、 243ページの『MessageListenerFactory.java』で説明されている `MessageListenerFactory` インターフェースのインプリメンテーションです。このファクトリーは、生成するすべてのメッセージ・リスナーのレコードを保持します。このファクトリーには、メソッド `printStats()` も含まれており、これらのリスナーごとに使用統計を表示します。

## ASFClient1.java

このアプリケーションは、MQ JMS ASF のクライアントとしての機能を果たします。このアプリケーションは、単一の MQSeries キュー内でメッセージを使用するために、単一の ConnectionConsumer をセットアップします。このアプリケーションでは、使用されるメッセージ・リスナーごとにスループット統計を表示し、 1 分後に終了します。

アプリケーションは、これらのインターフェースをインプリメントする MQ JMS クラスを使用して、 JNDI ネーム・スペースから必要な管理対象オブジェクトを検索するか、またはそれらのオブジェクトを明示的に作成することができます。必要な管理対象オブジェクトは、QueueConnectionFactory および Queue です。

このアプリケーションには、2 つのバージョンのコマンド行構文があります。

JNDI を指定して使用する場合、構文は以下のとおりです。

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

JNDI を指定しないで使用する場合、構文は以下のとおりです。

## ASF の使用の例

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

表23 では、パラメーターについて説明し、それぞれのデフォルトを示します。

*表 23. ASFClient1 のパラメーターおよびデフォルト*

| パラメーター | 意味 | デフォルト |
|---|---|---|
| jndiICF | JNDI に使用される初期コンテキスト・ファクトリー・クラス | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI に使用されるプロバイダー URL | ldap://localhost/o=ibm,c=us |
| qcfLookup | QueueConnectionFactory に使用される JNDI ルックアップ・キー | cn=qcf |
| qLookup | Queue に使用される JNDI ルックアップ・キー | cn=q |
| qMgrName | 接続先のキュー・マネージャーの名前 | "" (デフォルトのキュー・マネージャーを使用する) |
| qName | 使用するキューの名前 | SYSTEM.DEFAULT.LOCAL.QUEUE |
| poolSize | 使用されている ServerSessionPool 内に作成される ServerSession の数 | 5 |
| batchSize | 一度に ServerSession に割り当てることのできるメッセージの最大数。 | 10 |

アプリケーションは QueueConnectionFactory から QueueConnection を入手します。

ServerSessionPool (MyServerSessionPool 形式) は、以下のものを使用して構成されます。

- 前もって作成された QueueConnection
- 必要な poolSize
- 確認モード AUTO_ACKNOWLEDGE
- 245ページの『CountingMessageListenerFactory.java』で説明されているように、CountingMessageListenerFactory のインスタンス。

次に、接続の createConnectionConsumer メソッドが呼び出され、以下の場所に渡されます。

- 以前に入手された Queue
- ヌルのメッセージ・セレクター (すべてのメッセージが受信されることを示す)
- 作成されたばかりの ServerSessionPool
- 必要な batchSize

その後、接続の start() メソッドを呼び出して、メッセージの使用が開始されます。

クライアント・アプリケーションは、10 秒ごとに統計を表示して、使用されるそれぞれのメッセージ・リスナーごとにスループット統計を表示します。 1 分後、接続はクローズされ、サーバー・セッション・プールは停止され、アプリケーションは終了します。

## Load2.java

このクラスは、Load1.java と類似の方法で、いくつかのメッセージを含む指定されたキューをロードしてから終了する JMS アプリケーションです。コマンド行構文も、Load1.java の構文と類似しています (構文内の Load1 を Load2 に置き換えてください)。詳細については、244ページの『Load1.java』を参照してください。

相違点は、それぞれのメッセージに、値というユーザー・プロパティーが含まれており、 0 と 100 との間の、ランダムに選択された整数値をとります。このプロパティーは、メッセージにメッセージ・セレクターを適用できることを意味しています。したがって、メッセージは、 248ページの『ASFClient2.java』で説明されているクライアント・アプリケーション内で作成される 2 つのコンシューマー間で共用できます。

## LoggingMessageListenerFactory.java

このファイルには、以下の 2 つのクラスの定義が含まれています。

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener は、 javax.jms.MessageListener インターフェースのインプリメンテーションです。 LoggingMessageListener は、それに渡されるメッセージをとり、ログ・ファイルにエントリーを書きこみます。デフォルトのログ・ファイルは ./ASFClient2.log です。このファイルを検査し、このメッセージ・リスナーを使用している接続コンシューマーに送信されるメッセージをチェックすることができます。

LoggingMessageListenerFactory は、LoggingMessageListener 用のファクトリー・クラスです。 CountingMessageListenerFactory は、 243ページの『MessageListenerFactory.java』で説明されている MessageListenerFactory インターフェースのインプリメンテーションです。

## ASFClient2.java

ASFClient2.java は、ASFClient1.java よりもやや複雑なクライアント・アプリケーションです。 ASFClient2.java は、同じキューから送られる 2 つの ConnectionConsumer を作成しますが、異なるメッセージ・セレクターを適用します。アプリケーションは、一方のコンシューマーに対して CountingMessageListenerFactory を使用しますが、他方のコンシューマーに対しては LoggingMessageListenerFactory を使用します。 2 つの異なるメッセージ・リスナー・ファクトリーを使用することは、各コンシューマーがそれぞれのサーバー・セッション・プールを持っていなければならないことを意味しています。

アプリケーションは、画面上に一方の ConnectionConsumer と関連のある統計を表示し、ログ・ファイルに他方の ConnectionConsumer に関連のある統計を書き込みます。

コマンド行構文は、 245ページの『ASFClient1.java』 の構文と類似しています (構文内の ASFClient1 を ASFClient2 に置き換えてください)。 2 つのサーバー・セッション・プールのそれぞれには、 poolSize パラメーターによって設定される数の ServerSession が含まれています。

メッセージの配布数は均等ではありません。 Load2 によってソース・キューにロードされるメッセージには、ユーザー・プロパティーが含まれています。ここで、値は 0 と 100 の間であり、均等かつランダムに分散されなければなりません。メッセージ・セレクター `value>75` は highConnectionConsumer に適用され、メッセージ・セレクター `value≤75` は normalConnectionConsumer に適用されます。 highConnectionConsumer のメッセージ (合計ロードの約 25%) が LoggingMessageListener に送信されます。 normalConnectionConsumer のメッセージ (合計ロードの約 75%) は CountingMessageListener に送信されます。

クライアント・アプリケーションが実行するとき、 normalConnectionConsumer と関連のある統計およびそれと関連した CountingMessageListenerFactories は、 10 秒ごとに画面に表示されます。 highConnectionConsumer と関連のある統計およびそれと関連した LoggingMessageListenerFactories は、ログ・ファイルに書き込まれます。

メッセージの実際の宛先を調べるために、画面およびログ・ファイルを調べることができます。それぞれの CountingMessageListeners ごとに合計を追加してください。すべてのメッセージが使用される前にクライアント・アプリケーションが終了しない限り、これはロードの約 75% を計上しているはずです。ログ・ファイル・エントリーの数は、ロードの残りを計上しているはずです。 (すべてのメッセージが使用される前にクライアント・アプリケーションが終了する場合、アプリケーション・タイムアウトを増やすことができます。)

## TopicLoad.java

このクラスは、247ページの『Load2.java』 で説明されている Load2 キュー・ローダーの、パブリッシュ / サブスクライブ・バージョンである JMS アプリケーションです。このクラスは、指定されたトピックの下で必要な数のメッセージをパブリッシュしてから、終了します。それぞれのメッセージには、値というユーザー・プロパティーが含まれており、 0 と 100 との間のランダムに選択された整数値をとります。

このアプリケーションを使用するには、ブローカーが実行しており、必要なセットアップが完了していることを確認してください。詳細については、26ページの『パブリッシュ / サブスクライブ・モードのための付加的なセットアップ』を参照してください。

このアプリケーションには、2 つのバージョンのコマンド行構文があります。

JNDI を指定して使用する場合、構文は以下のとおりです。

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

JNDI を指定しないで使用する場合、構文は以下のとおりです。

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

表24 では、パラメーターについて説明し、それぞれのデフォルトを示します。

*表 24. TopicLoad のパラメーターおよびデフォルト*

| パラメーター | 意味 | デフォルト |
|---|---|---|
| jndiICF | JNDI に使用される初期コンテキスト・ファクトリー・クラス | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI に使用されるプロバイダー URL | ldap://localhost/o=ibm,c=us |
| tcfLookup | TopicConnectionFactory に使用される JNDI ルックアップ・キー | cn=tcf |
| tLookup | Topic に使用される JNDI ルックアップ・キー | cn=t |
| qMgrName | 接続先のキュー・マネージャーの名前、およびメッセージのパブリッシュ先となるブローカー・キュー・マネージャー | "" (デフォルトのキュー・マネージャーを使用する) |
| tName | パブリッシュ先となるトピックの名前 | MQJMS/ASF/TopicLoad |
| sleepTime | メッセージ・プット間で休止する時間 (ミリ秒単位) | 0 (休止なし) |
| numMsgs | 書き込むメッセージの数 | 200 |

エラーがある場合、エラー・メッセージが表示され、アプリケーションは終了します。

## ASFClient3.java

ASFClient3.java は、245ページの『ASFClient1.java』のパブリッシュ / サブスクライブ・バージョンであるクライアント・アプリケーションです。 ASFClient3.java は、単一の Topic 上にパブリッシュされるメッセージを使用するために、単一の ConnectionConsumer をセットアップします。このアプリケーションでは、使用されるメッセージ・リスナーごとにスループット統計を表示し、 1 分後に終了します。

このアプリケーションには、2 つのバージョンのコマンド行構文があります。

### ASF の使用の例

JNDI を指定して使用する場合、構文は以下のとおりです。

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

JNDI を指定しないで使用する場合、構文は以下のとおりです。

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

表25 では、パラメーターについて説明し、それぞれのデフォルトを示します。

*表 25. ASFClient3 のパラメーターおよびデフォルト*

| パラメーター | 意味 | デフォルト |
|---|---|---|
| jndiICF | JNDI に使用される初期コンテキスト・ファクトリー・クラス | com.sun.jndi.ldap.LdapCtxFactory |
| jndiURL | JNDI に使用されるプロバイダー URL | ldap://localhost/o=ibm,c=us |
| tcfLookup | TopicConnectionFactory に使用される JNDI ルックアップ・キー | cn=tcf |
| tLookup | Topic に使用される JNDI ルックアップ・キー | cn=t |
| qMgrName | 接続先のキュー・マネージャーの名前、およびメッセージのパブリッシュ先となるブローカー・キュー・マネージャー | "" (デフォルトのキュー・マネージャーを使用する) |
| tName | 使用するトピックの名前 | MQJMS/ASF/TopicLoad |
| poolSize | 使用されている ServerSessionPool 内に作成される ServerSession の数 | 5 |
| batchSize | 一度に ServerSession に割り当てることのできるメッセージの最大数。 | 10 |

ASFClient1 と同様に、クライアント・アプリケーションは、 10 秒ごとに統計を表示して、使用されるメッセージ・リスナーごとにスループット統計を表示します。 1 分後、接続はクローズされ、サーバー・セッション・プールは停止され、アプリケーションは終了します。

## ASFClient4.java

ASFClient4.java は、より複雑なパブリッシュ / サブスクライブ・クライアント・アプリケーションです。 ASFClient4.java は、すべて同じトピックから送られる 3 つの ConnectionConsumer を作成しますが、それぞれは異なるメッセージ・セレクターを提供します。

最初の 2 つのコンシューマーは、アプリケーション 248ページの『ASFClient2.java』 と同じ方法で、高水準および標準のメッセージ・セレクターを使用します。 3 番目のコンシューマーは、いかなるメッセージ・セレクターも使用しません。アプリケーションは、最初の 2 つのセレクター・ベースのコンシューマーに対して 2 つの CountingMessageListenerFactories を使用し、 3 番目のコンシューマーに対しては LoggingMessageListenerFactory を使用します。アプリケーション

は異なるメッセージ・リスナー・ファクトリーを使用するので、各コンシューマーはそれぞれのサーバー・セッション・プールを持っていなければなりません。

アプリケーションは、画面上に 2 つのセレクター・ベースのコンシューマーと関連のある統計を表示します。アプリケーションは、ログ・ファイルに 3 番目の ConnectionConsumer と関連のある統計を書き込みます。

コマンド行構文は、 249ページの『ASFClient3.java』 の構文と類似しています (構文内の ASFClient3 を ASFClient4 に置き換えてください)。 3 つのサーバー・セッション・プールのそれぞれには、 poolSize パラメーターによって設定される数の ServerSession が含まれています。

クライアント・アプリケーションが実行するとき、 normalConnectionConsumer および highConnectionConsumer と関連のある統計、さらにそれと関連した CountingMessageListenerFactories が、10 秒ごとに画面に表示されます。 3 つ目の ConnectionConsumer と関連のある統計およびそれと関連した LoggingMessageListenerFactories は、ログ・ファイルに書き込まれます。

メッセージの実際の宛先を調べるために、画面およびログ・ファイルを調べることができます。それぞれの CountingMessageListeners ごとに合計を追加し、ログ・ファイル・エントリーの数を調べてください。

メッセージの配布は、同じアプリケーションのポイント・ツー・ポイント・バージョンによって入手される配布とは異なっているはずです (ASFClient2.java)。これは、パブリッシュ / サブスクライブ・ドメインでは、トピックの各コンシューマーは、そのトピック上でパブリッシュされる各メッセージごとにそれぞれのコピーを入手するためです。このアプリケーションでは、指定されたトピック・ロードの場合、高水準および標準のコンシューマーは、それぞれ、ロードの約 25% および 75% を受信します。 3 番目のコンシューマーは、ロードの 100% を受信します。したがって、受信されるメッセージの合計数は、トピック上に最初にパブリッシュされたロードの 100% を超えています。

# 第14章 JMS インターフェースおよびクラス

MQSeries classes for Java Message Service は、インターフェースおよびクラスの Sun javax.jms パッケージに基づくいくつかの java クラスおよびインターフェースで構成されています。クライアントは、下記にリストされ、以下のセクションで詳細に説明されている、 Sun インターフェースおよびクラスを使用して作成する必要があります。 Sun インターフェースおよびクラスをインプリメントする MQSeries オブジェクトの名前は、'MQ' という接頭部を持っています (オブジェクト記述で述べられていない限り)。オブジェクト記述には、標準 JMS 定義からの MQSeries オブジェクトの逸脱に関する詳細が含まれています。これらの逸脱事項には、'*' のマークが付けられています。

## Sun Java Message Service クラスおよびインターフェース

以下の表は、パッケージ **javax.jms** に含まれている JMS オブジェクトをリストしています。 **'*'** のマークが付けられているインターフェースは、アプリケーションによりインプリメントされます。 **'**'** のマークが付けられているインターフェースは、アプリケーション・サーバーによってインプリメントされます。

*表 26. インターフェースの要約*

| インターフェース | 説明 |
|---|---|
| **BytesMessage** | BytesMessage は、解釈されていないバイトのストリームを含むメッセージを送信するのに使用します。 |
| **Connection** | JMS Connection は JMS プロバイダーへのクライアントのアクティブな接続です。 |
| **ConnectionConsumer** | アプリケーション・サーバーに対して、 Connection は ConnectionConsumer を作成する特別な機能を提供します。 |
| **ConnectionFactory** | ConnectionFactory は、アドミニストレーターが定義した一連の接続構成パラメーターをカプセル化します。 |
| **ConnectionMetaData** | ConnectionMetaData は Connection を記述する情報を提供します。 |
| **DeliveryMode** | JMS によってサポートされている送達モード。 |
| **Destination** | Queue および Topic 用の親インターフェース。 |
| **ExceptionListener*** | 例外リスナーは、Connection 非同期送達スレッドによって出される例外を受信するために使用されます。 |
| **MapMessage** | MapMessage は、名前と値のペア (名前は String で、値は Java プリミティブ・タイプ) のセットを送信するのに使用します。 |
| **Message** | Message インターフェースは、すべての JMS メッセージのルート・インターフェースです。 |
| **MessageConsumer** | すべてのメッセージ・コンシューマーの親インターフェース。 |
| **MessageListener*** | MessageListener は非同期に送達されたメッセージを受信するのに使用します。 |

*表 26. インターフェースの要約 (続き)*

| インターフェース | 説明 |
|---|---|
| **MessageProducer** | クライアントは、Destination にメッセージを送信するために MessageProducer を使用します。 |
| **ObjectMessage** | ObjectMessage は、直列化可能 Java オブジェクトを含むメッセージを送信するのに使用します。 |
| **Queue** | Queue オブジェクトは、プロバイダー固有のキュー名をカプセル化します。 |
| **QueueBrowser** | クライアントは QueueBrowser を使用して、キュー上のメッセージを削除せずに参照します。 |
| **QueueConnection** | QueueConnection は、JMS ポイント・ツー・ポイント・プロバイダーへのアクティブな接続です。 |
| **QueueConnectionFactory** | QueueConnectionFactory は、 JMS ポイント・ツー・ポイント・プロバイダーとの QueueConnection を作成するためにクライアントが使用します。 |
| **QueueReceiver** | QueueReceiver は、キューに送達されたメッセージを受信するためにクライアントが使用します。 |
| **QueueSender** | QueueSender は、クライアントがキューにメッセージを送信するのに使用します。 |
| **QueueSession** | QueueSession は、QueueReceiver、 QueueSender、QueueBrowser、および TemporaryQueue を作成するメソッドを提供します。 |
| **ServerSession **** | ServerSession は、アプリケーション・サーバーによってインプリメントされるオブジェクトです。 |
| **ServerSessionPool **** | ServerSessionPool は、 ConnectionConsumer のメッセージを処理するための ServerSession のプールを提供するためにアプリケーション・サーバーによってインプリメントされるオブジェクトです。 |
| **Session** | JMS Session は、メッセージを作成および消費するための単一スレッドのコンテキストです。 |
| **StreamMessage** | StreamMessage は、 Java プリミティブのストリームを送信するのに使用します。 |
| **TemporaryQueue** | TemporaryQueue は、QueueConnection の継続期間中に作成される固有の Queue オブジェクトです。 |
| **TemporaryTopic** | TemporaryTopic は、TopicConnection の継続期間中に作成される固有の Topic オブジェクトです。 |
| **TextMessage** | TextMessage は、`java.lang.String` を含むメッセージを送信するために使用されます。 |
| **Topic** | Topic オブジェクトは、プロバイダー固有のトピック名をカプセル化します。 |
| **TopicConnection** | TopicConnection は、JMS Pub/Sub プロバイダーへのアクティブな接続です。 |
| **TopicConnectionFactory** | TopicConnectionFactory は、 JMS パブリッシュ / サブスクライブ・プロバイダーとの TopicConnection を作成するためにクライアントが使用します。 |

*表 26. インターフェースの要約 (続き)*

| インターフェース | 説明 |
|---|---|
| **TopicPublisher** | TopicPublisher は、クライアントがトピックについてのメッセージをパブリッシュするのに使用します。 |
| **TopicSession** | TopicSession は、TopicPublisher、TopicSubscriber、および TemporaryTopic を作成するためのメソッドを提供します。 |
| **TopicSubscriber** | TopicSubscriber は、トピックにパブリッシュされたメッセージを受信するためにクライアントが使用します。 |
| **XAConnection** | XAConnection は、XASession を提供することによって Connection の機能を拡張したものです。 |
| **XAConnectionFactory** | いくつかのアプリケーション・サーバーは、 Java Transaction Service (JTS) 対応のリソース使用を分散トランザクションにグループ分けするためのサポートを提供します。 |
| **XAQueueConnection** | XAQueueConnection は QueueConnection と同じ作成オプションを提供します。 |
| **XAQueueConnectionFactory** | XAQueueConnectionFactory は QueueConnectionFactory と同じ作成オプションを提供します。 |
| **XAQueueSession** | XAQueueSession は、QueueReceiver、QueueSender、および QueueBrowser を作成するのに使用できる正規の QueueSession を提供します。 |
| **XASession** | XASession は、Java Transaction API (JTA) の JMS プロバイダーのサポートへのアクセスを追加することによって、 Session の機能を拡張します。 |
| **XATopicConnection** | XATopicConnection は TopicConnection と同じ作成オプションを提供します。 |
| **XATopicConnectionFactory** | XATopicConnectionFactory は TopicConnectionFactory と同じ作成オプションを提供します。 |
| **XATopicSession** | XATopicSession は、TopicSubscriber および TopicPublisher を作成するために使用できる正規の TopicSession を提供します。 |

*表 27. クラスの要約*

| クラス | 説明 |
|---|---|
| **QueueRequestor** | サービス要求の作成を簡単にするために、JMS は QueueRequestor ヘルパー・クラスを提供しています。 |
| **TopicRequestor** | サービス要求の作成を簡単にするために、JMS は TopicRequestor ヘルパー・クラスを提供しています。 |

# MQSeries JMS クラス

以下の表は、Sun インターフェースをインプリメントする MQSeries クラスを含む **com.ibm.mq.jms** および **com.ibm.jms** パッケージをリストしています。

*表 28. パッケージ 'com.ibm.mq.jms' クラスの要約*

| クラス | インプリメント |
|---|---|
| MQConnection | Connection |
| MQConnectionConsumer | ConnectionConsumer |
| MQConnectionFactory | ConnectionFactory |
| MQConnectionMetaData | ConnectionMetaData |
| MQDestination | Destination |
| MQMessageConsumer | MessageConsumer |
| MQMessageProducer | MessageProducer |
| MQQueue | Queue |
| MQQueueBrowser | QueueBrowser |
| MQQueueConnection | QueueConnection |
| MQQueueConnectionFactory | QueueConnectionFactory |
| MQQueueEnumeration | QueueBrowser からの java.util.Enumeration |
| MQQueueReceiver | QueueReceiver |
| MQQueueSender | QueueSender |
| MQQueueSession | QueueSession |
| MQSession | Session |
| MQTemporaryQueue | TemporaryQueue |
| MQTemporaryTopic | TemporaryTopic |
| MQTopic | Topic |
| MQTopicConnection | TopicConnection |
| MQTopicConnectionFactory | TopicConnectionFactory |
| MQTopicPublisher | TopicPublisher |
| MQTopicSession | TopicSession |
| MQTopicSubscriber | TopicSubscriber |
| MQXAConnection | XAConnection |
| MQXAConnectionFactory | XAConnectionFactory |
| MQXAQueueConnection | XAQueueConnection |
| MQXAQueueConnectionFactory | XAQueueConnectionFactory |
| MQXAQueueSession | XAQueueSession |
| MQXASession | XASession |
| MQXATopicConnection | XATopicConnection |
| MQXATopicConnectionFactory | XATopicConnectionFactory |
| MQXATopicSession | XATopicSession |

*表 29. パッケージ 'com.ibm.jms' クラスの要約*

| クラス | インプリメント |
|---|---|
| JMSBytesMessage | BytesMessage |
| JMSMapMessage | MapMessage |
| JMSMessage | Message |
| JMSObjectMessage | ObjectMessage |
| JMSStreamMessage | StreamMessage |
| JMSTextMessage | TextMessage |

以下の JMS インターフェースのサンプル・インプリメンテーションが、本リリースの MQSeries classes for Java Message Service で提供されます。

- ServerSession
- ServerSessionPool

240ページの『アプリケーション・サーバーのサンプル・コード』を参照してください。

# BytesMessage

public interface **BytesMessage**
extends **Message**

MQSeries クラス: **JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

BytesMessage は、解釈されていないバイトのストリームを含むメッセージを送信するのに使用します。これは **Message** を継承し、バイト・メッセージの本文を追加します。メッセージの受信側がバイトの解釈を提供します。

**注:** このメッセージ・タイプは、既存のメッセージ・フォーマットをエンコードするクライアントが使用します。可能であれば、このメッセージ・タイプではなく、他の自己定義メッセージ・タイプを使用してください。

参照: **MapMessage**、**Message**、 **ObjectMessage**、**StreamMessage**、および **TextMessage**。

## メソッド

### readBoolean

```
public boolean readBoolean() throws JMSException
```

バイト・メッセージから boolean 値を読み取ります。

**戻り:** 読み取られた boolean 値。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・バイトの最後である場合。

### readByte

```
public byte readByte() throws JMSException
```

バイト・メッセージから符号付きの 8 ビット値を読み取ります。

**戻り:** 符号付きの 8 ビット byte として戻される、バイト・メッセージ内の次のバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readUnsignedByte

```
public int readUnsignedByte() throws JMSException
```

バイト・メッセージから符号なしの 8 ビット数値を読み取ります。

**戻り:** 符号なしの 8 ビット数値として解釈される、バイト・メッセージ内の次のバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readShort

```
public short readShort() throws JMSException
```

バイト・メッセージから符号付きの 16 ビット数値を読み取ります。

**戻り:** 符号付きの 16 ビット数値として解釈される、バイト・メッセージ内の次の 2 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readUnsignedShort

```
public int readUnsignedShort() throws JMSException
```

バイト・メッセージから符号なしの 16 ビット数値を読み取ります。

**戻り:** 符号なしの 16 ビット整数として解釈される、バイト・メッセージ内の次の 2 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

## BytesMessage

### readChar

```
public char readChar() throws JMSException
```

バイト・メッセージから Unicode 文字値を読み取ります。

**戻り:** Unicode 文字として解釈される、バイト・メッセージ内の次の 2 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readInt

```
public int readInt() throws JMSException
```

バイト・メッセージから符号付きの 32 ビット整数を読み取ります。

**戻り:** `int` として解釈される、バイト・メッセージ内の次の 4 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readLong

```
public long readLong() throws JMSException
```

バイト・メッセージから符号付きの 64 ビット整数を読み取ります。

**戻り:** `long` として解釈される、バイト・メッセージ内の次の 8 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readFloat

```
public float readFloat() throws JMSException
```

バイト・メッセージから `float` を読み取ります。

**戻り:** `float` として解釈される、バイト・メッセージ内の次の 4 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readDouble

```
public double readDouble() throws JMSException
```

バイト・メッセージから double を読み取ります。

**戻り:** double として解釈される、バイト・メッセージ内の次の 8 つのバイト。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readUTF

```
public java.lang.String readUTF() throws JMSException
```

修正 UTF-8 フォーマットを使用してエンコードしたストリングをバイト・メッセージから読み取ります。最初の 2 つのバイトは 2 バイトの長さフィールドとして解釈されます。

**戻り:** バイト・メッセージからの Unicode ストリング。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- MessageEOFException - メッセージ・バイトの最後である場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readBytes

```
public int readBytes(byte[] value) throws JMSException
```

バイト・メッセージからバイト配列を読み取ります。ストリーム内に十分なバイトがあれば、バッファー全体が埋められます。もしなければ、バッファーの一部が埋められます。

**パラメーター:**

value - データの読み込み先のバッファー。

**戻り:** バッファーに読み込まれるバイトの合計数か、または -1 (バイトの最後に達したためにデータがもうない場合)。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### readBytes

```
public int readBytes(byte[] value, int length)
                                    throws JMSException
```

バイト・メッセージの一部を読み取ります。

**パラメーター:**

- value - データの読み込み先のバッファー。
- length - 読み取るバイト数。

**戻り:** バッファーに読み込まれるバイトの合計数か、または -1 (バイトの最後に達したためにデータがもうない場合)。

**投入:**

- MessageNotReadableException - メッセージが書き込み専用モードの場合。
- IndexOutOfBoundsException - length が負の場合、または配列 value の長さよりも小さい場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

boolean を 1 バイト値としてバイト・メッセージに書き込みます。値 true は値 (byte)1 として書き出され、値 false は値 (byte)0 として書き出されます。

**パラメーター:**

value - 書き込まれる boolean 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeByte

```
public void writeByte(byte value) throws JMSException
```

byte を 1 バイト値としてバイト・メッセージに書き出します。

**パラメーター:**

value - 書き込まれる byte 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeShort

```
public void writeShort(short value) throws JMSException
```

short を 2 バイトとしてバイト・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる short。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeChar

```
public void writeChar(char value) throws JMSException
```

char を 2 バイト値としてバイト・メッセージに書き込みます (高位バイトが最初です)。

**パラメーター:**

value - 書き込まれる char 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeInt

```
public void writeInt(int value) throws JMSException
```

int を 4 バイトとしてバイト・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる int。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeLong

```
public void writeLong(long value) throws JMSException
```

long を 8 バイトとしてバイト・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる long。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeFloat

```
public void writeFloat(float value) throws JMSException
```

クラス Float 内の floatToIntBits メソッドを使用して float 引き数を int に変換してから、 int 値を 4 バイトの数量としてバイト・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる float 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeDouble

```
public void writeDouble(double value) throws JMSException
```

クラス Double 内の doubleToLongBits メソッドを使用して double 引き数を long に変換してから、その long 値を 8 バイトの数量としてバイト・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる double 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeUTF

```
public void writeUTF(java.lang.String value)
                              throws JMSException
```

マシンから独立した様式の UTF-8 エンコードを使用してストリングをバイト・メッセージに書き込みます。バッファーに書き込まれる UTF-8 ストリングの先頭は 2 バイトの長さフィールドです。

**パラメーター:**

value - 書き込まれる String 値。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

バイト配列をバイト・メッセージに書き込みます。

**パラメーター:**

`value` - 書き込まれるバイト配列。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeBytes

```
public void writeBytes(byte[] value,
                       int length) throws JMSException
```

バイト配列の一部をバイト・メッセージに書き込みます。

**パラメーター:**

- `value` - 書き込まれるバイト配列の値。
- `offset` - バイト配列内の初期オフセット。
- `length` - 使用するバイト数。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### writeObject

```
public void writeObject(java.lang.Object value)
                                      throws JMSException
```

Java オブジェクトをバイト・メッセージに書き込みます。

**注:** このメソッドは、プリミティブ・オブジェクト・タイプ (Integer、Double、 Long など)、ストリング、およびバイト配列の場合にのみ機能します。

**パラメーター:**

`value` - 書き込まれる Java オブジェクト。

**投入:**

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- MessageFormatException - オブジェクトが無効なタイプの場合。
- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

### reset

```
public void reset() throws JMSException
```

## BytesMessage

メッセージの本文を読み取り専用モードにし、バイト (バイト単位) を先頭に再配置します。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージをリセットできない場合。
- MessageFormatException - メッセージのフォーマットが無効な場合。

# Connection

public interface **Connection**
サブインターフェース: **QueueConnection**, **TopicConnection**, **XAQueueConnection**, および **XATopicConnection**

MQSeries クラス: **MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

JMS Connection は JMS プロバイダーへのクライアントのアクティブな接続です。

参照: **QueueConnection**、**TopicConnection**、 **XAQueueConnection**、および **XATopicConnection**

## メソッド

### getClientID

```
public java.lang.String getClientID()
                                throws JMSException
```

この接続用のクライアント ID を取得します。クライアント ID は、ConnectionFactory でアドミニストレーターが事前構成するか、または setClientId を呼び出して割り当てることができます。

**戻り:** 固有のクライアント ID。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションでこの Connection のクライアント ID を戻せない場合。

### setClientID

```
public void setClientID(java.lang.String clientID)
                                          throws JMSException
```

この接続用のクライアント ID を設定します。

**注:** Point-to-Point 接続の場合、クライアント ID は無視されます。

**パラメーター:**

clientID - 固有のクライアント ID。

**投入:**

- JMSException - 内部エラーのため、 JMS のインプリメンテーションでこの Connection 用のクライアント ID を設定できない場合。
- InvalidClientIDException - JMS クライアントが無効な、または重複している クライアント ID を指定した場合。
- IllegalStateException - 接続のクライアント ID の設定を適切でないときに試行した場合、または接続のクライアント ID が管理上構成されている場合。

## Connection

### getMetaData

```
public ConnectionMetaData getMetaData() throws JMSException
```

この接続のメタデータを取得します。

**戻り:** 接続のメタデータ

**投入:** JMSException - この Connection の Connection メタデータを、JMS インプリメンテーションで取得できなかった場合の一般例外。

**参照:** **ConnectionMetaData**

### getExceptionListener

```
public ExceptionListener getExceptionListener()
                                          throws JMSException
```

この Connection の ExceptionListener を取得します。

**戻り:** この Connection の ExceptionListener。

**投入:** JMSException - この Connection の Exception リスナーを、 JMS インプリメンテーションで取得できなかった場合の一般例外。

### setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
                                                    throws JMSException
```

この接続の例外リスナーを設定します。

**パラメーター:**
handler - 例外リスナー。

**投入:** JMSException - この Connection の Exception リスナーを、 JMS インプリメンテーションで設定できなかった場合の一般例外。

### start

```
public void start() throws JMSException
```

着信メッセージの Connection の送達を開始 (または再開) します。すでに開始されているセッションを開始しようとした場合、無視されます。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションでメッセージの送達を開始できない場合。

**参照:** stop

### stop

```
public void stop() throws JMSException
```

着信メッセージの Connection の送達を一時的に停止するのに使用します。start メソッドを使用すれば、送達を再開できます。停止されている場合、その Connection のメッセージの全コンシューマーへの送達が禁止されます。同期受信が妨げられるので、メッセージはメッセージ・リスナーへ送達されません。

セッションを停止しても、メッセージを送信する能力には影響しません。すでに停止されているセッションを停止しようとした場合、無視されます。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションでメッセージの送達を停止できない場合。

**参照:** start

### close

```
public void close() throws JMSException
```

プロバイダーが Connection のために JVM の外側でいくつかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。クローズされる接続のセッション、プロデューサー、およびコンシューマーをクローズする必要はありません。

接続をクローズすると、そのセッションの処理中の何らかのトランザクションがロールバックされます。セッションの作業が外部のトランザクション・マネージャーによって調整されている場合、 XASession を使用すると、セッションのコミットおよびロールバック・メソッドは使用されず、クローズされるセッションの作業の結果は、後でトランザクション・マネージャーによって判別されます。接続をクローズしても、クライアントが確認しているセッションの強制的な確認は行われません。

MQ JMS はセッションが使用できる MQSeries hConns のプールを保持します。状況によっては、Connection.close() がこのプールをクリアします。アプリケーションが複数の Connection を連続して使用する場合、強制的にプールを JMS Connection 間でアクティブにしておくことが可能です。これを行うには、ご使用の JMS アプリケーションが存続する間、 MQPoolToken を com.ibm.mq.MQEnvironment に登録しておきます。詳細については、72ページの『接続プーリング』および 101ページの『MQEnvironment』を参照してください。

**投入:** JMSException - 内部エラーのため、JMS のインプリメンテーションで接続をクローズできない場合。たとえば、リソースを解放できない場合やソケット接続をクローズできない場合などです。

# ConnectionConsumer

public interface **ConnectionConsumer**

MQSeries クラス: **MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

アプリケーション・サーバーに対して、 Connection は ConnectionConsumer を作成する特別な機能を提供します。消費するメッセージは Destination および Property Selector で指定します。また、そのメッセージを処理するために、使用する ServerSessionPool を ConnectionConsumer に指定しなければなりません。

参照: **QueueConnection** および **TopicConnection**。

## メソッド

### close()

```
public void close() throws JMSException
```

プロバイダーが ConnectionConsumer のために JVM の外側でいくつかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**投入:** JMSException - JMS のインプリメンテーションで ConnectionConsumer のためのリソースの解放が行えない場合、または接続コンシューマーをクローズできない場合。

### getServerSessionPool()

```
public ServerSessionPool getServerSessionPool()
                                        throws JMSException
```

その接続コンシューマーに関連したサーバー・セッションを取得します。

**戻り:** この接続コンシューマーが使用するサーバー・セッション・プール。

**投入:** JMSException - 内部エラーのため、JMS のインプリメンテーションで、その接続コンシューマーに関連したサーバー・セッション・プールを取得できない場合。

# ConnectionFactory

public interface **ConnectionFactory**
サブインターフェース: **QueueConnectionFactory**、**TopicConnectionFactory**、**XAQueueConnectionFactory**、および **XATopicConnectionFactory**

MQSeries クラス: **MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

ConnectionFactory は、アドミニストレーターによって定義されている接続構成パラメーターのセットをカプセル化します。これは JMS プロバイダーとの接続を作成するために、クライアントが使用します。

参照: **QueueConnectionFactory**、 **TopicConnectionFactory**、**XAQueueConnectionFactory**、および **XATopicConnectionFactory**

## MQSeries コンストラクター

### MQConnectionFactory

```
public MQConnectionFactory()
```

## メソッド

### setDescription *

```
public void setDescription(String x)
```

オブジェクトの簡略説明。

### getDescription *

```
public String getDescription()
```

オブジェクト記述の取得します。

### setTransportType *

```
public void setTransportType(int x) throws JMSException
```

使用するトランスポート・タイプを設定します。
JMSC.MQJMS_TP_BINDINGS_MQ か
JMSC.MQJMS_TP_CLIENT_MQ_TCPIP のいずれかにできます。

### getTransportType *

```
public int getTransportType()
```

トランスポート・タイプを取得します。

### setClientId *

```
public void setClientId(String x)
```

この Connection を使って作成するすべての接続に使用されるクライアント ID を設定します。

## ConnectionFactory

### getClientId *

```
public String getClientId()
```

この ConnectionFactory を使って作成するすべての接続に使用されるクライアント ID を取得します。

### setQueueManager *

```
public void setQueueManager(String x) throws JMSException
```

接続先のキュー・マネージャーの名前を設定します。

### getQueueManager *

```
public String getQueueManager()
```

キュー・マネージャーの名前を取得します。

### setHostName *

```
public void setHostName(String hostname)
```

接続先のホストの名前 (クライアント専用)。

### getHostName *

```
public String getHostName()
```

ホストの名前を取得します。

### setPort *

```
public void setPort(int port) throws JMSException
```

クライアント接続のポートを設定します。

**パラメーター:**

port - 使用する新しい値。

**投入:** JMSException (ポートが負の場合)

### getPort *

```
public int getPort()
```

ポート番号を取得します (クライアント接続専用)。

### setChannel *

```
public void setChannel(String x) throws JMSException
```

使用するチャネルを設定します (クライアント専用)。

### getChannel *

```
public String getChannel()
```

使用されたチャネルを取得します (クライアント専用)。

### setCCSID *

```
public void setCCSID(int x) throws JMSException
```

キュー・マネージャーへの接続時に使用する文字セットを設定します。許可されている値のリストについては、119ページの表13 を参照してください。ほとんどの状況では、デフォルト値 (819) を使用するようにお勧めします。

**getCCSID ***

```
public int getCCSID()
```

キュー・マネージャーの文字セットを取得します。

**setReceiveExit ***

```
public void setReceiveExit(String receiveExit)
```

受信出口をインプリメントするクラスの名前。

**getReceiveExit ***

```
public String getReceiveExit()
```

受信出口クラスの名前を取得します。

**setReceiveExitInit ***

```
public void setReceiveExitInit(String x)
```

受信出口クラスのコンストラクターに渡される初期化ストリング。

**getReceiveExitInit ***

```
public String getReceiveExitInit()
```

受信出口クラスに渡された初期化ストリングを取得します。

**setSecurityExit ***

```
public void setSecurityExit(String securityExit)
```

セキュリティー出口をインプリメントするクラスの名前。

**getSecurityExit ***

```
public String getSecurityExit()
```

セキュリティー出口クラスの名前を取得します。

**setSecurityExitInit ***

```
public void setSecurityExitInit(String x)
```

セキュリティー出口のコンストラクターに渡される初期化ストリング。

**getSecurityExitInit ***

```
public String getSecurityExitInit()
```

セキュリティー出口の初期化ストリングを取得します。

**setSendExit ***

```
public void setSendExit(String sendExit)
```

送信出口をインプリメントするクラスの名前。

**getSendExit ***

```
public String getSendExit()
```

送信出口クラスの名前を取得します。

## ConnectionFactory

### setSendExitInit *

```
public void setSendExitInit(String x)
```

送信出口のコンストラクターに渡される初期化ストリング。

### getSendExitInit *

```
public String getSendExitInit()
```

送信出口の初期化ストリングを取得します。

# ConnectionMetaData

public interface **ConnectionMetaData**

MQSeries クラス: **MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData は接続を記述している情報を提供します。

## MQSeries コンストラクター

### MQConnectionMetaData

```
public MQConnectionMetaData()
```

## メソッド

### getJMSVersion

```
public java.lang.String getJMSVersion() throws JMSException
```

JMS のバージョンを取得します。

**戻り:** JMS のバージョン。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getJMSMajorVersion

```
public int getJMSMajorVersion() throws JMSException
```

JMS のメジャー・バージョン番号を取得します。

**戻り:** JMS のメジャー・バージョン番号。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getJMSMinorVersion

```
public int getJMSMinorVersion() throws JMSException
```

JMS のマイナー・バージョン番号を取得します。

**戻り:** JMS のマイナー・バージョン番号。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getJMSXPropertyNames

```
public java.util.Enumeration getJMSXPropertyNames()
                                                throws JMSException
```

この接続がサポートしている JMSX プロパティーの名前の列挙を取得します。

**戻り:** JMSX PropertyNames の列挙。

**投入:** JMSException - プロパティー名の取得中に、 JMS のインプリメンテーションで内部エラーが発生した場合。

### getJMSProviderName

```
public java.lang.String getJMSProviderName()
                                        throws JMSException
```

JMS プロバイダーの名前を取得します。

**戻り:** JMS プロバイダーの名前。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getProviderVersion

```
public java.lang.String getProviderVersion()
                                         throws JMSException
```

JMS プロバイダーのバージョンを取得します。

**戻り:** JMS プロバイダーのバージョン。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getProviderMajorVersion

```
public int getProviderMajorVersion() throws JMSException
```

JMS プロバイダーのメジャー・バージョン番号を取得します。

**戻り:** JMS プロバイダーのメジャー・バージョン番号。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### getProviderMinorVersion

```
public int getProviderMinorVersion() throws JMSException
```

JMS プロバイダーのマイナー・バージョン番号を取得します。

**戻り:** JMS プロバイダーのマイナー・バージョン番号。

**投入:** JMSException - メタデータの取得中に、JMS のインプリメンテーションで内部エラーが発生した場合。

### toString *

```
public String toString()
```

**指定変更:**

クラス Object 内の toString。

# DeliveryMode

public interface **DeliveryMode**

JMS によってサポートされている送達モード。

## フィールド

### NON_PERSISTENT

```
public static final int NON_PERSISTENT
```

これは、メッセージを継続的ストレージに記録する必要がないので、最もオーバーヘッドの小さい送達モードです。

### PERSISTENT

```
public static final int PERSISTENT
```

このモードでは、クライアントの送信操作の一部としてメッセージを継続的ストレージに記録するように、 JMS プロバイダーに指示します。

# Destination

public interface **Destination**
サブインターフェース: **Queue**、**TemporaryQueue**、**TemporaryTopic**、および **Topic**

MQSeries クラス: **MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

Destination オブジェクトは、プロバイダー固有のアドレスをカプセル化します。

参照: **Queue**、**TemporaryQueue**、 **TemporaryTopic**、および **Topic**

## MQSeries コンストラクター

### MQDestination

```
public MQDestination()
```

## メソッド

### setDescription *

```
public void setDescription(String x)
```

オブジェクトの簡略説明。

### getDescription *

```
public String getDescription()
```

オブジェクトの説明を取得します。

### setPriority *

```
public void setPriority(int priority) throws JMSException
```

この宛先に送信されるすべてのメッセージの優先順位を指定変更するのに使用します。

### getPriority *

```
public int getPriority()
```

指定変更する優先順位の値を取得します。

### setExpiry *

```
public void setExpiry(int expiry) throws JMSException
```

この宛先に送信されるすべてのメッセージの有効期限を指定変更するのに使用します。

### getExpiry *

```
public int getExpiry()
```

この宛先の有効期限の値を取得します。

### setPersistence *

```
public void setPersistence(int persistence)
                                    throws JMSException
```

この宛先に送信されるすべてのメッセージの永続性を指定変更するのに使用します。

### getPersistence *

```
public int getPersistence()
```

この宛先の永続性の値を取得します。

### setTargetClient *

```
public void setTargetClient(int targetClient)
                                    throws JMSException
```

リモート・アプリケーションが JMS 準拠かどうかを示すフラグを立てます。

### getTargetClient *

```
public int getTargetClient()
```

JMS 準拠の標識フラグを取得します。

### setCCSID *

```
public void setCCSID(int x) throws JMSException
```

この宛先に送信されるメッセージ内のテキスト・ストリングをエンコードするのに使用される文字セット。許可されている値のリストについては、119 ページの表13 を参照してください。 デフォルト値は 1208 (UTF8) です。

### getCCSID *

```
public int getCCSID()
```

この宛先で使用される文字セットの名前を取得します。

### setEncoding *

```
public void setEncoding(int x) throws JMSException
```

この宛先に送信されるメッセージ内の数値フィールドに使用される、エンコードを指定します。許可されている値のリストについては、119ページの表 13 を参照してください。

### getEncoding *

```
public int getEncoding()
```

この宛先で使用されるエンコードを取得します。

# ExceptionListener

public interface **ExceptionListener**

JMS プロバイダーが Connection に関する重大な問題を検出した場合、その Connection の ExceptionListener に知らせます (ExceptionListener が登録されている場合)。これは、その問題を記述している JMSException を渡す、リスナーの onException() メソッドを呼び出すことによって行われます。

これにより、クライアントに問題を非同期で通知できます。メッセージを消費するだけの Connection もあるので、それらの Connection が失敗したことを知る方法は他にはありません。

例外は以下の場合に送達されます。
- 非同期メッセージの受信に失敗した
- メッセージが実行時例外を投じた

## メソッド

### onException

```
public void onException(JMSException exception)
```

ユーザーに JMS 例外を通知します。

**パラメーター:**

exception - JMS 例外。非同期メッセージ送達による例外があります。多くの場合、それらはキュー・マネージャーからのメッセージの受信に関する問題を示しています。あるいは、JMS インプリメンテーションでの内部エラーの場合もあります。

# MapMessage

public interface **MapMessage**
extends **Message**

MQSeries クラス: **JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

MapMessage は、名前と値のペア (名前は String で、値は Java プリミティブ・タイプ) のセットを送信するのに使用します。エントリーには、順次アクセスか、または名前によるランダムでのアクセスを行えます。エントリーの順序は定義されていません。

参照: **BytesMessage**、**Message**、 **ObjectMessage**、**StreamMessage**、および **TextMessage**

## メソッド

### getBoolean

```
public boolean getBoolean(java.lang.String name)
                                                throws JMSException
```

指定した名前のブール値を戻します。

**パラメーター:**

name - ブール値の名前。

**戻り:** 指定した名前のブール値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getByte

```
public byte getByte(java.lang.String name)
                                        throws JMSException
```

指定した名前のバイト値を戻します。

**パラメーター:**

name - バイトの名前。

**戻り:** 指定した名前のバイト値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

## MapMessage

### getShort

```
public short getShort(java.lang.String name) throws JMSException
```

指定した名前の short 値を戻します。

**パラメーター:**

name - short の名前。

**戻り:** 指定した名前の short 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getChar

```
public char getChar(java.lang.String name)
                                        throws JMSException
```

指定した名前の Unicode 文字値を戻します。

**パラメーター:**

name - Unicode 文字の名前。

**戻り:** 指定した名前の Unicode 文字値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getInt

```
public int getInt(java.lang.String name)
                                          throws JMSException
```

指定した名前の整数値を戻します。

**パラメーター:**

name - 整数の名前。

**戻り:** 指定した名前の整数値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getLong

```
public long getLong(java.lang.String name)
                                        throws JMSException
```

指定した名前の long 値を戻します。

**パラメーター:**

name - long の名前。

**戻り:** 指定した名前の long 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getFloat

```
public float getFloat(java.lang.String name) throws JMSException
```

指定した名前の float 値を戻します。

**パラメーター:**

name - float の名前。

**戻り:** 指定した名前の float 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getDouble

```
public double getDouble(java.lang.String name) throws JMSException
```

指定した名前の double 値を戻します。

**パラメーター:**

name - double の名前。

**戻り:** 指定した名前の double 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getString

```
public java.lang.String getString(java.lang.String name)
                                                  throws JMSException
```

指定した名前のストリング値を戻します。

**パラメーター:**

name - ストリングの名前。

**戻り:** 指定した名前のストリング値。この名前の項目が何もない場合、ヌル値が戻されます。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getBytes

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

指定した名前のバイト配列値を戻します。

**パラメーター:**

name - バイト配列の名前。

**戻り:** 指定した名前のバイト配列値のコピー。この名前の項目が何もない場合、ヌル値が戻されます。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageFormatException - この型変換が無効な場合。

### getObject

```
public java.lang.Object getObject(java.lang.String name)
                                  throws JMSException
```

指定した名前の Java オブジェクト値を戻します。このメソッドは、setObject メソッド呼び出しか、それと同等のプリミティブ・セット・メソッドのいずれかを使用して、 Map 内に保管されている値をオブジェクト様式で戻します。

**パラメーター:**

name - Java オブジェクトの名前。

**戻り:** 指定した名前の Java オブジェクト値のコピー (int として設定されている場合、Integer が戻されます)。この名前の項目が何もない場合、ヌル値が戻されます。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### getMapNames

```
public java.util.Enumeration getMapNames() throws JMSException
```

Map メッセージのすべての名前の列挙を戻します。

**戻り:** この Map メッセージ内のすべての名前の列挙。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

### setBoolean

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

指定した名前のブール値を Map に設定します。

**パラメーター:**

- name - ブール値の名前。
- value - Map に設定するブール値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setByte

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

指定した名前のバイト値を Map に設定します。

**パラメーター:**

- name - バイトの名前。
- value - Map に設定するバイト値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setShort

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

指定した名前の short 値を Map に設定します。

**パラメーター:**

- name - short の名前。
- value - Map に設定する short 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setChar

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

指定した名前の Unicode 文字値を Map に設定します。

**パラメーター:**

- name - Unicode 文字の名前。
- value - Map に設定する Unicode 文字値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setInt

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

指定した名前の整数値を Map に設定します。

**パラメーター:**

- name - 整数の名前。
- value - Map に設定する整数値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setLong

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

指定した名前の long 値を Map に設定します。

**パラメーター:**

- name - long の名前。
- value - Map に設定する long 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setFloat

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

指定した名前の float 値を Map に設定します。

**パラメーター:**

- name - float の名前。
- value - Map に設定する float 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setDouble

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

指定した名前の double 値を Map に設定します。

**パラメーター:**

- name - double の名前。
- value - Map に設定する double 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

指定した名前のストリング値を Map に設定します。

**パラメーター:**

- name - ストリングの名前。
- value - Map に設定するストリング値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

指定した名前のバイト配列値を Map に設定します。

**パラメーター:**

- name - バイト配列の名前。
- value - Map に設定するバイト配列値。

  配列への以後の変更によってマップ内の値が更新されないように、配列のコピーが行われます。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

指定した名前のバイト配列値の一部を Map に設定します。

配列への以後の変更によってマップ内の値が更新されないように、配列のコピーが行われます。

**パラメーター:**

- name - バイト配列の名前。

- value - Map に設定するバイト配列値。
- offset - バイト配列内の初期オフセット。
- length - コピーするバイト数。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### setObject

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

指定した名前の Java オブジェクト値を Map に設定します。このメソッドは、オブジェクト・プリミティブ・タイプ (Integer、Double、Long など)、ストリング、およびバイト配列の場合にのみ機能します。

**パラメーター:**

- name - Java オブジェクトの名前。
- value - Map に設定される Java オブジェクト値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageFormatException - オブジェクトが無効な場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### itemExists

```
public boolean itemExists(java.lang.String name)
                                    throws JMSException
```

この MapMessage に項目が存在するかどうか検査します。

**パラメーター:**

name - テストする項目の名前。

**戻り:** 項目が存在する場合は真。

**投入:** JMSException - JMS エラーが発生した場合。

# Message

public interface **Message**
サブインターフェース: **BytesMessage**、**MapMessage**、**ObjectMessage**、**StreamMessage**、および **TextMessage**

MQSeries クラス: **JMSMessage**

```
java.lang.Object
  |
  +----com.ibm.jms.MQJMSMessage
```

Message インターフェースは、すべての JMS メッセージのルート・インターフェースです。これは、すべてのメッセージで使用される JMS ヘッダーおよび確認メソッドを定義します。

## フィールド

### DEFAULT_DELIVERY_MODE

```
public static final int DEFAULT_DELIVERY_MODE
```

デフォルトの送達モード値。

### DEFAULT_PRIORITY

```
public static final int DEFAULT_PRIORITY
```

デフォルトの優先順位の値。

### DEFAULT_TIME_TO_LIVE

```
public static final long DEFAULT_TIME_TO_LIVE
```

デフォルトの存続時間値。

## メソッド

### getJMSMessageID

```
public java.lang.String getJMSMessageID()
                                    throws JMSException
```

メッセージ ID を取得します。

**戻り:** メッセージ ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がメッセージ ID を取得できない場合。

**参照:** setJMSMessageID()

### setJMSMessageID

```
public void setJMSMessageID(java.lang.String id)
                                          throws JMSException
```

メッセージ ID を設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

**パラメーター:**
id - メッセージの ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がメッセージ ID を設定できない場合。

**参照:** getJMSMessageID()

### getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

メッセージのタイム・スタンプを取得します。

**戻り:** メッセージのタイム・スタンプ。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がタイム・スタンプを取得できない場合。

**参照:** setJMSTimestamp()

### setJMSTimestamp

```
public void setJMSTimestamp(long timestamp)
                                    throws JMSException
```

メッセージのタイム・スタンプを設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

**パラメーター:**
timestamp - このメッセージのタイム・スタンプ。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がタイム・スタンプを設定できない場合。

**参照:** getJMSTimestamp()

### getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                                throws JMSException
```

メッセージの相関 ID を、バイトの配列として取得します。

**戻り:** バイトの配列としての、メッセージの相関 ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が相関 ID を取得できない場合。

**参照:** setJMSCorrelationID()、getJMSCorrelationID()、setJMSCorrelationIDAsBytes()

### setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[]
                                       correlationID)
                                              throws JMSException
```

メッセージの相関 ID を、バイトの配列として設定します。クライアントはこの呼び出しを使用して、直前のメッセージからのメッセージ ID と同じか、またはアプリケーション固有のストリングと同じ相関 ID を設定できます。アプリケーション固有のストリングは、先頭を文字 ID にすることはできません。

**パラメーター:**

correlationID - ストリングとしての相関 ID、または参照されるメッセージのメッセージ ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が相関 ID を設定できない場合。

**参照:** setJMSCorrelationID()、getJMSCorrelationID()、getJMSCorrelationIDAsBytes()

### getJMSCorrelationID

```
public java.lang.String getJMSCorrelationID()
                                        throws JMSException
```

メッセージの相関 ID を取得します。

**戻り:** ストリングとしての、メッセージの相関 ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が相関 ID を取得できない場合。

**参照:** setJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

### setJMSCorrelationID

```
public void setJMSCorrelationID
                (java.lang.String correlationID)
                                        throws JMSException
```

メッセージの相関 ID を設定します。

クライアントは JMSCorrelationID ヘッダー・フィールドを使用して、あるメッセージを別のメッセージとリンクさせることができます。多くの場合、応答メッセージをその要求メッセージとリンクさせるために使用します。

**注:** JMSCorrelationID で使用される byte[] 値は、移植可能ではありません。

**パラメーター:**

correlationID - 参照されるメッセージのメッセージ ID。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が相関 ID を設定できない場合。

**参照:** getJMSCorrelationID()、getJMSCorrelationIDAsBytes()、setJMSCorrelationIDAsBytes()

## Message

### getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException
```

このメッセージの応答の送信先を取得します。

**戻り:** このメッセージの応答の送信先

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が ReplyTo Destination を取得できない場合。

**参照:** setJMSReplyTo()

### setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo)
                                          throws JMSException
```

このメッセージの応答の送信先を設定します。

**パラメーター:**

replyTo - このメッセージの応答の送信先。ヌル値は、予期される応答がないことを示しています。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が ReplyTo Destination を設定できない場合。

**参照:** getJMSReplyTo()

### getJMSDestination

```
public Destination getJMSDestination() throws JMSException
```

このメッセージの宛先を取得します。

**戻り:** このメッセージの宛先。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS Destination を取得できない場合。

**参照:** setJMSDestination()

### setJMSDestination

```
public void setJMSDestination(Destination destination)
                                                 throws JMSexception
```

このメッセージの宛先を設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

**パラメーター:**

destination - このメッセージの宛先。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS Destination を設定できない場合。

**参照:** getJMSDestination()

### getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException
```

このメッセージの送達モードを取得します。

**戻り:** このメッセージの送達モード。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS DeliveryMode を取得できない場合。

**参照:** setJMSDeliveryMode()、DeliveryMode

### setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode)
                                              throws JMSException
```

このメッセージの送達モードを設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

メッセージの送信時に送達モードを更新するには、 QueueSender または TopicPublisher で setDeliveryMode メソッドを使用します (このメソッドは MessageProducer から継承されます)。

**パラメーター:**

deliveryMode - このメッセージの送達モード。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS DeliveryMode を設定できない場合。

**参照:** getJMSDeliveryMode()、DeliveryMode

### getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException
```

このメッセージが再送達されるかどうかの指示を取得します。

クライアントが受信したメッセージに、再送達されたことを示す標識がセットされている場合、おそらく (絶対ではありませんが) そのメッセージは以前にクライアントに送達されたものの、その時点ではクライアントはその受信を確認しなかったということです。

**戻り:** このメッセージが再送達される場合は真。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS Redelivered フラグを取得できない場合。

**参照:** setJMSRedelivered()

### setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered)
                                                throws JMSException
```

このメッセージが再送達されるかどうかの指示を設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

**パラメーター:**

redelivered - このメッセージが再送達されるかどうかの指示。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMSRedelivered フラグを設定できない場合。

**参照:** getJMSRedelivered()

### getJMSType

```
public java.lang.String getJMSType() throws JMSException
```

メッセージ・タイプを取得します。

**戻り:** メッセージ・タイプ。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージ・タイプを取得できない場合。

**参照:** setJMSType()

### setJMSType

```
public void setJMSType(java.lang.String type)
                                        throws JMSException
```

メッセージ・タイプを設定します。

JMS クライアントは、そのタイプをアプリケーションが使用するかどうかに関係なく、値をタイプに割り当てなければなりません。これにより、そのタイプを必要とするプロバイダーが使用できるように、確実にそのタイプが正しく設定されます。

**パラメーター:**

type - メッセージのクラス。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージ・タイプを設定できない場合。

**参照:** getJMSType()

### getJMSExpiration

```
public long getJMSExpiration() throws JMSException
```

メッセージの有効期限の値を取得します。

**戻り:** メッセージの有効期限が切れる時刻。送信時の GMT に、クライアントが指定した存続時間の値を足したものです。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージの有効期限を取得できない場合。

**参照:** setJMSExpiration()

### setJMSExpiration

```
public void setJMSExpiration(long expiration)
                                          throws JMSException
```

メッセージの有効期限の値を設定します。

メッセージの送信時には、このメソッドを使用して設定された値は無視されますが、受信したメッセージ内の値を変更するのに、このメソッドを使用できます。

**パラメーター:**

expiration - メッセージの有効期限の時刻。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージの有効期限を設定できない場合。

**参照:** getJMSExpiration()

### getJMSPriority

```
public int getJMSPriority() throws JMSException
```

メッセージ優先順位を取得します。

**戻り:** メッセージ優先順位

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージの優先順位を取得できない場合。

**参照:** setJMSPriority() (優先順位の等級について)

### setJMSPriority

```
public void setJMSPriority(int priority)
                                    throws JMSException
```

このメッセージの優先順位を設定します。

JMS は 10 段階の優先順位の値を定義します。最も低い優先順位が 0 で、最も高い優先順位が 9 です。さらに、クライアントは優先順位 0 〜 4 を通常優先順位の等級と見なし、優先順位 5 〜 9 を急送優先順位の等級と見なさなければなりません。

**パラメーター:**

priority - このメッセージの優先順位。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が JMS メッセージの優先順位を設定できない場合。

**参照:** getJMSPriority()

### clearProperties

```
public void clearProperties() throws JMSException
```

メッセージのプロパティーをクリアします。ヘッダー・フィールドおよびメッセージ本文はクリアされません。

**投入:** JMSException - 内部 JMS エラーが原因で、 JMS が JMS メッセージのプロパティーをクリアできない場合。

### propertyExists

```
public boolean propertyExists(java.lang.String name)
                                              throws JMSException
```

プロパティー値が存在するかどうか検査します。

**パラメーター:**

name - テストするプロパティーの名前。

**戻り:** プロパティーが存在する場合は真。

**投入:** JMSException - 内部 JMS エラーが原因で、プロパティーが存在するかどうか検査できない場合。

### getBooleanProperty

```
public boolean getBooleanProperty(java.lang.String name)
                                                    throws JMSException
```

指定した名前の boolean プロパティーの値を戻します。

**パラメーター:**

name - boolean プロパティーの名前。

**戻り:** 指定した名前の boolean プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getByteProperty

```
public byte getByteProperty(java.lang.String name)
                                                throws JMSException
```

指定した名前の byte プロパティーの値を戻します。

**パラメーター:**

name - byte プロパティーの名前。

**戻り:** 指定した名前の byte プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getShortProperty

```
public short getShortProperty(java.lang.String name)
                                                 throws JMSException
```

指定した名前の short プロパティーの値を戻します。

**パラメーター:**

name - short プロパティーの名前。

**戻り:** 指定した名前の short プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getIntProperty

```
public int getIntProperty(java.lang.String name)
                                            throws JMSException
```

指定した名前の integer プロパティーの値を戻します。

**パラメーター:**

name - integer プロパティーの名前。

**戻り:** 指定した名前の integer プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getLongProperty

```
public long getLongProperty(java.lang.String name)
                                             throws JMSException
```

指定した名前の long プロパティーの値を戻します。

**パラメーター:**

name - long プロパティーの名前。

**戻り:** 指定した名前の long プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getFloatProperty

```
public float getFloatProperty(java.lang.String name)
                                              throws JMSException
```

指定した名前の float プロパティーの値を戻します。

**パラメーター:**

name - float プロパティーの名前。

**戻り:** 指定した名前の float プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getDoubleProperty

```
public double getDoubleProperty(java.lang.String name)
                                               throws JMSException
```

指定した名前の double プロパティーの値を戻します。

**パラメーター:**

name - double プロパティーの名前。

**戻り:** 指定した名前の double プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getStringProperty

```
public java.lang.String getStringProperty (java.lang.String name)
                                                    throws JMSException
```

指定した名前の String プロパティーの値を戻します。

**パラメーター:**

name - String プロパティーの名前。

**戻り:** 指定した名前の String プロパティーの値。この名前のプロパティーが何もない場合、ヌル値が戻されます。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。
- MessageFormatException - この型変換が無効な場合。

### getObjectProperty

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                                    throws JMSException
```

指定した名前の Java オブジェクトのプロパティー値を戻します。

**パラメーター:**

name - Java オブジェクトのプロパティーの名前。

**戻り:** オブジェクト様式での、指定した名前の Java オブジェクトのプロパティー値 (たとえば int として設定されている場合、Integer が戻されます)。この名前のプロパティーが何もない場合、ヌル値が戻されます。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを取得できない場合。

### getPropertyNames

```
public java.util.Enumeration getPropertyNames()
                                          throws JMSException
```

すべてのプロパティー名の列挙を戻します。

**戻り:** プロパティー値のすべての名前の列挙。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がプロパティー名を取得できない場合。

### setBooleanProperty

```
public void setBooleanProperty(java.lang.String name,
                               boolean value) throws JMSException
```

指定した名前の boolean プロパティーの値を Message に設定します。

**パラメーター:**

- name - boolean プロパティーの名前。
- value - Message に設定する boolean プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setByteProperty

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

指定した名前の byte プロパティーの値を Message に設定します。

**パラメーター:**

- name - byte プロパティーの名前。
- value - Message に設定する byte プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setShortProperty

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

指定した名前の short プロパティーの値を Message に設定します。

**パラメーター:**

- name - short プロパティーの名前。
- value - Message に設定する short プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setIntProperty

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

指定した名前の integer プロパティーの値を Message に設定します。

**パラメーター:**

- name - integer プロパティーの名前。
- value - Message に設定する integer プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setLongProperty

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

指定した名前の long プロパティーの値を Message に設定します。

**パラメーター:**

- name - long プロパティーの名前。
- value - Message に設定する long プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setFloatProperty

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

指定した名前の float プロパティーの値を Message に設定します。

**パラメーター:**

- name - float プロパティーの名前。
- value - Message に設定する float プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setDoubleProperty

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

指定した名前の double プロパティーの値を Message に設定します。

**パラメーター:**

- name - double プロパティーの名前。
- value - Message に設定する double プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setStringProperty

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

指定した名前の String プロパティーの値を Message に設定します。

**パラメーター:**

- name - String プロパティーの名前。
- value - Message に設定する String プロパティーの値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がプロパティーを設定できない場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### setObjectProperty

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

指定した名前のプロパティーの値を Message に設定します。

**パラメーター:**

- name - Java オブジェクトのプロパティーの名前。
- value - Message に設定される Java オブジェクトのプロパティー値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS が Property を設定できない場合。
- MessageFormatException - オブジェクトが無効な場合。
- MessageNotWriteableException - プロパティーが読み取り専用の場合。

### acknowledge

```
public void acknowledge() throws JMSException
```

このメッセージとそれ以前にセッションから受信したすべてのメッセージを確認します。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が確認を行えなかった場合。

### clearBody

```
public void clearBody() throws JMSException
```

メッセージ本文を消去します。メッセージの他の部分はすべて、そのまま残されます。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS が消去を行えなかった場合。

# MessageConsumer

public interface **MessageConsumer**
サブインターフェース: **QueueReceiver** および **TopicSubscriber**

MQSeries クラス: **MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

すべてのメッセージ・コンシューマーの親インターフェース。 クライアントは、Destination からメッセージを受信するのにメッセージ・コンシューマーを使用します。

## メソッド

### getMessageSelector

```
public java.lang.String getMessageSelector()
                                throws JMSException
```

このメッセージ・コンシューマーのメッセージ・セレクター式を取得します。

**戻り:** このメッセージ・コンシューマーのメッセージ・セレクター。

**投入:** JMSException - JMS エラーが原因で、JMS がメッセージ・セレクターを取得できない場合。

### getMessageListener

```
public MessageListener getMessageListener()
                                     throws JMSException
```

メッセージ・コンシューマーの MessageListener を取得します。

**戻り:** メッセージ・コンシューマーのリスナー、またはリスナーが設定されていない場合はヌル。

**投入:** JMSException - JMS エラーが原因で、JMS がメッセージ・リスナーを取得できない場合。

**参照:** setMessageListener

### setMessageListener

```
public void setMessageListener(MessageListener listener)
                                           throws JMSException
```

メッセージ・コンシューマーの MessageListener を設定します。

**パラメーター:**

messageListener - このリスナーにメッセージが送達されます。

**投入:** JMSException - JMS エラーが原因で、JMS がメッセージ・リスナーを設定できない場合。

**参照:** getMessageListener

### receive

```
public Message receive() throws JMSException
```

このメッセージ・コンシューマー用に作成された次のメッセージを受信します。

**戻り:** このメッセージ・コンシューマー用に作成された次のメッセージ。

**投入:** JMSException - エラーが原因で、JMS が次のメッセージを受信できない場合。

### receive

```
public Message receive(long timeOut) throws JMSException
```

指定したタイムアウト間隔内に到着した次のメッセージを受信します。タイムアウト間隔がゼロの場合、呼び出しはメッセージが到着するまで無期限に待機します。

**パラメーター:**

timeout - タイムアウト値 (ミリ秒単位)。

**戻り:** このメッセージ・コンシューマー用に作成された次のメッセージ、または入手できない場合はヌル。

**投入:** JMSException - エラーが原因で、JMS が次のメッセージを受信できない場合。

### receiveNoWait

```
public Message receiveNoWait() throws JMSException
```

次のメッセージを即時に入手できる場合、そのメッセージを受信します。

**戻り:** このメッセージ・コンシューマー用に作成された次のメッセージ、または入手できない場合はヌル。

**投入:** JMSException - エラーが原因で、JMS が次のメッセージを受信できない場合。

### close

```
public void close() throws JMSException
```

プロバイダーが MessageConsumer のために JVM の外側でいくつかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

処理中の受信またはメッセージ・リスナーが完了するまで、この呼び出しはブロックします。

**投入:** JMSException - エラーが原因で、JMS がコンシューマーをクローズできない場合。

# MessageListener

public interface **MessageListener**

MessageListener は非同期に送達されたメッセージを受信するのに使用します。

## メソッド

### onMessage

```
public void onMessage(Message message)
```

メッセージをリスナーに渡します。

**パラメーター:**

message - リスナーに渡すメッセージ。

**参照** Session.setMessageListener

# MessageProducer

public interface **MessageProducer**
サブインターフェース: **QueueSender** および **TopicPublisher**

MQSeries クラス: **MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

クライアントは、メッセージ・プロデューサーを使用して、メッセージを宛先に送信します。

## MQSeries コンストラクター

### MQMessageProducer

```
public MQMessageProducer()
```

## メソッド

### setDisableMessageID

```
public void setDisableMessageID(boolean value)
                                        throws JMSException
```

メッセージ ID が使用不可かどうか設定します。

デフォルトでは、メッセージ ID が使用可能になっています。

**注:** このメソッドは、MQSeries classes for Java Message Service のインプリメンテーションでは無視されます。

**パラメーター:**
value - メッセージ ID が使用不可かどうかを示します。

**投入:** JMSException - 内部エラーが原因で、JMS が使用不可のメッセージ ID を設定できない場合。

### getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException
```

メッセージ ID が使用不可かどうかの指示を取得します。

**戻り:** メッセージ ID が使用不可かどうかの指示。

**投入:** JMSException - 内部エラーが原因で、JMS が使用不可のメッセージ ID を取得できない場合。

### setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                               throws JMSException
```

メッセージ・タイム・スタンプが使用不可かどうか設定します。

デフォルトでは、メッセージ・タイム・スタンプが使用可能になっています。

**注:** このメソッドは、MQSeries classes for Java Message Service のインプリメンテーションでは無視されます。

**パラメーター:**

value - メッセージ・タイム・スタンプが使用不可かどうかを示します。

**投入:** JMSException - 内部エラーが原因で、 JMS が使用不可のメッセージ・タイム・スタンプを設定できない場合。

### getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                        throws JMSException
```

メッセージ・タイム・スタンプが使用不可かどうかの指示を取得します。

**戻り:** メッセージ ID が使用不可かどうかの指示。

**投入:** JMSException - 内部エラーが原因で、 JMS が使用不可のメッセージ・タイム・スタンプを取得できない場合。

### setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                                         throws JMSException
```

プロデューサーのデフォルト送達モードを設定します。

デフォルトでは、送達モードが PERSISTENT に設定されています。

**パラメーター:**

deliveryMode - このメッセージ・プロデューサーのメッセージ送達モード。

**投入:** JMSException - 内部エラーが原因で、JMS が送達モードを設定できない場合。

**参照:** getDeliveryMode

### getDeliveryMode

```
public int getDeliveryMode() throws JMSException
```

プロデューサーのデフォルト送達モードを取得します。

**戻り:** このメッセージ・プロデューサーのメッセージ送達モード。

**投入:** JMSException - 内部エラーが原因で、JMS が送達モードを取得できない場合。

**参照:** setDeliveryMode

### setPriority

```
public void setPriority(int priority) throws JMSException
```

プロデューサーのデフォルトの優先順位を設定します。

デフォルトでは、優先順位が 4 に設定されています。

**パラメーター:**

priority - このメッセージ・プロデューサーのメッセージ優先順位。

**投入:** JMSException - 内部エラーが原因で、JMS が優先順位を設定できない場合。

**参照:** getPriority

### getPriority

```
public int getPriority() throws JMSException
```

プロデューサーのデフォルトの優先順位を取得します。

**戻り:** このメッセージ・プロデューサーのメッセージ優先順位。

**投入:** JMSException - 内部エラーが原因で、JMS が優先順位を取得できない場合。

**参照:** setPriority

### setTimeToLive

```
public void setTimeToLive(long timeToLive)
                                    throws JMSException
```

ディスパッチ時からの時間のデフォルトの長さ (ミリ秒単位) を設定します。これは、作成されたメッセージをメッセージ・システムが保存する時間です。

デフォルトでは、存続時間は 0 に設定されています。

**パラメーター:**

timeToLive - メッセージの存続時間 (ミリ秒単位)。ゼロを指定すると無制限になります。

**投入:** JMSException - 内部エラーが原因で、JMS が存続時間を設定できない場合。

**参照:** getTimeToLive

### getTimeToLive

```
public long getTimeToLive() throws JMSException
```

ディスパッチ時からの時間のデフォルトの長さ (ミリ秒単位) を取得します。これは、作成されたメッセージをメッセージ・システムが保存する時間です。

**戻り:** メッセージの存続時間 (ミリ秒単位)。ゼロは無制限になります。

**投入:** JMSException - 内部エラーが原因で、JMS が存続時間を取得できない場合。

**参照:** setTimeToLive

## MessageProducer

### close

```
public void close() throws JMSException
```

プロバイダーが MessageProducer のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**投入:** JMSException - エラーが原因で、JMS がプロデューサーをクローズできない場合。

# MQQueueEnumeration *

public class **MQQueueEnumeration**
extends **Object**
implements **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

キュー上のメッセージの列挙。このクラスは JMS 仕様では定義されず、MQQueueBrowser の getEnumeration メソッドを呼び出すことによって作成されます。このクラスには、ブラウズ・カーソルを保持するための基本 MQQueue インスタンスが含まれます。カーソルがキューの最後を離れると、そのキューはクローズします。

このクラスのインスタンスをリセットする方法はありません。これは「1 度限りの」メカニズムとして機能します。

参照: **MQQueueBrowser**

## メソッド

### hasMoreElements

```
public boolean hasMoreElements()
```

別のメッセージを戻せるかどうか指示します。

### nextElement

```
public Object nextElement() throws NoSuchElementException
```

現在のメッセージを戻します。

hasMoreElements() が「true」を戻すと、 nextElement() は必ずメッセージを戻します。戻されるメッセージは、hasMoreElements() と nextElement 呼び出しの間で有効期限日付を渡すことができます。

# ObjectMessage

public interface **ObjectMessage**
extends **Message**

MQSeries クラス: **JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

ObjectMessage は、直列化可能 Java オブジェクトを含むメッセージを送信するのに使用します。これは Message から継承し、 Java 参照を 1 つ含む本文を追加します。使用できるのは、直列化可能 Java オブジェクトだけです。

参照: **BytesMessage**、**MapMessage**、 **Message**、**StreamMessage**、および **TextMessage**

## メソッド

### setObject

```
public void setObject(java.io.Serializable object)
                                      throws JMSException
```

このメッセージ・データを含む直列化可能オブジェクトを設定します。ObjectMessage には、setObject() が呼び出された時点でのオブジェクトのスナップショットが含まれます。オブジェクトの以降の変更は ObjectMessage 本文には影響を与えません。

**パラメーター:**

object - メッセージのデータ。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がオブジェクトを設定できない場合。
- MessageFormatException - オブジェクトの直列化が失敗した場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### getObject

```
public java.io.Serializable getObject()
                            throws JMSException
```

このメッセージ・データを含む直列化可能オブジェクトを取得します。デフォルト値はヌルです。

**戻り:** このメッセージ・データを含む直列化可能オブジェクト。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がオブジェクトを取得できない場合。
- MessageFormatException - オブジェクトの非直列化が失敗した場合。

# Queue

public interface **Queue**
extends **Destination**
サブインターフェース: **TemporaryQueue**

MQSeries クラス: **MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Queue オブジェクトは、プロバイダー固有のキュー名をカプセル化します。このようにして、クライアントはキューの識別を JMS メソッドに指定します。

## MQSeries コンストラクター

### MQQueue *

```
public MQQueue()
```

管理ツールが使用するデフォルトのコンストラクター。

### MQQueue *

```
public MQQueue(String URIqueue)
```

新しい MQQueue インスタンスを作成します。 192ページで説明されているように、このストリングは URI 形式をとります。

### MQQueue *

```
public MQQueue(String queueManagerName,
               String queueName)
```

## メソッド

### getQueueName

```
public java.lang.String getQueueName()
                                  throws JMSException
```

このキューの名前を取得します。

名前に依存しているクライアントは移植可能ではありません。

**戻り:** キュー名

**投入:** JMSException - 内部エラーが原因で、 Queue に使用される JMS のインプリメンテーションがキュー名を戻せない場合。

### toString

```
public java.lang.String toString()
```

キュー名を見栄えの良い印刷バージョンで戻します。

**戻り:** このキューのプロバイダー固有の識別値。

**指定変更:**

クラス java.lang.Object 内の toString

### getReference *

```
public Reference getReference() throws NamingException
```

このキューのための参照を作成します。

**戻り:** このキューのための参照

**投入:** NamingException

### setBaseQueueName *

```
public void setBaseQueueName(String x) throws JMSException
```

MQSeries キュー名の値を設定します。

**注:** このメソッドは管理ツール専用です。 queue:qmgr:queue 形式のストリングのデコードは試行されません。

### getBaseQueueName *

```
public String getBaseQueueName()
```

**戻り:** MQSeries キュー名の値。

### setBaseQueueManagerName *

```
public void setBaseQueueManagerName(String x) throws JMSException
```

MQSeries キュー・マネージャー名の値を設定します。

**注:** このメソッドは管理ツール専用です。

### getBaseQueueManagerName *

```
public String getBaseQueueManagerName()
```

**戻り:** MQSeries キュー・マネージャー名の値。

# QueueBrowser

public interface **QueueBrowser**

MQSeries クラス: **MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

クライアントは QueueBrowser を使用して、キュー上のメッセージを削除せずに参照します。

**注:** ブラウズ・カーソルを保持するには、 MQSeries クラス **MQQueueEnumeration** を使用します。

参照: **QueueReceiver**

## メソッド

### getQueue

```
public Queue getQueue() throws JMSException
```

このキュー・ブラウザーに関連付けられているキューを取得します。

**戻り:** キュー。

**投入:** JMSException - JMS エラーが原因で、このブラウザーに関連付けられているキューを JMS が取得できない場合。

### getMessageSelector

```
public java.lang.String getMessageSelector() throws JMSException
```

このキュー・ブラウザーのメッセージ・セレクター式を取得します。

**戻り:** このキュー・ブラウザーのメッセージ・セレクター。

**投入:** JMSException - JMS エラーが原因で、このブラウザーのメッセージ・セレクターを JMS が取得できない場合。

### getEnumeration

```
public java.util.Enumeration getEnumeration() throws JMSException
```

現在のキュー・メッセージをブラウズするための列挙を、受信の順序で取得します。

**戻り:** メッセージをブラウズするための列挙。

**投入:** JMSException - JMS エラーが原因で、このブラウザーの列挙を JMS が取得できない場合。

**注:** 存在しないキュー用にブラウザーが作成されている場合、getEnumeration への最初の呼び出しが行われるまで、これは検出されません。

## close

```
public void close() throws JMSException
```

プロバイダーが QueueBrowser のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**投入:** JMSException - JMS エラーが原因で、JMS がこのブラウザーをクローズできない場合。

# QueueConnection

public interface **QueueConnection**
extends **Connection**
サブインターフェース: **XAQueueConnection**

MQSeries クラス: **MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

QueueConnection は、JMS ポイント・ツー・ポイント・プロバイダーへのアクティブな接続です。クライアントは、 1 つまたは複数の QueueSession (メッセージの生成および消費に使用する) を作成するために、 QueueConnection を使用します。

参照: **Connection**、 **QueueConnectionFactory**、および **XAQueueConnection**

## メソッド

### createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

QueueSession を作成します。

**パラメーター:**

- transacted - 真であれば、セッションがトランザクション化されます。
- acknowledgeMode - コンシューマーまたはクライアントが、受け取ったあらゆるメッセージを確認するかどうかを指示します。以下の値を指定できます。

  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  セッションがトランザクション化されている場合、このパラメーターは無視されます。

**戻り:** 新たに作成されたキュー・セッション。

**投入:** JMSException - 内部エラーか、特定のトランザクションおよび確認通知モードのサポート不足が原因で、 JMS Connection がセッションを作成できない場合。

### createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                                      (Queue queue,
                                       java.lang.String messageSelector,
                                       ServerSessionPool sessionPool,
                                       int maxMessages)
                                                  throws JMSException
```

この接続用の接続コンシューマーを作成します。これは専門的な機能なので、通常の JMS クライアントは使用しません。

**パラメーター:**

- queue - アクセスするキュー。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。
- sessionPool - この接続コンシューマーに関連付けられるサーバー・セッション・プール。
- maxMessages - サーバー・セッションに一度に割り当てられるメッセージの最大数。

**戻り:** 接続コンシューマー。

**投入:**

- JMSException - 内部エラーか、sessionPool および messageSelector の無効な引き数が原因で、 JMS Connection が接続コンシューマーを作成できない場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

**参照:** ConnectionConsumer

### close *

```
public void close() throws JMSException
```

**指定変更:**

クラス MQConnection 内の close。

# QueueConnectionFactory

public interface **QueueConnectionFactory**
extends **ConnectionFactory**
サブインターフェース: **XAQueueConnectionFactory**

MQSeries クラス: **MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

QueueConnectionFactory は、 JMS ポイント・ツー・ポイント・プロバイダーとの QueueConnection を作成するためにクライアントが使用します。

参照: **ConnectionFactory** および **XAQueueConnectionFactory**

## MQSeries コンストラクター

### MQQueueConnectionFactory

```
public MQQueueConnectionFactory()
```

## メソッド

### createQueueConnection

```
public QueueConnection createQueueConnection()
                                        throws JMSException
```

デフォルトのユーザー ID でキュー接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**戻り:** 新たに作成されたキュー接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーがキュー接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### createQueueConnection

```
public QueueConnection createQueueConnection
                                (java.lang.String userName,
                                 java.lang.String password)
                                        throws JMSException
```

特定のユーザー ID でキュー接続を作成します。

**注:** このメソッドを使用できるのは、トランスポート・タイプ JMSC.MQJMS_TP_CLIENT_MQ_TCPIP だけです (ConnectionFactory を

参照してください)。接続は停止モードで作成されます。
Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**パラメーター:**

- userName - 呼び出し側のユーザー名。
- password - 呼び出し側のパスワード。

**戻り:** 新たに作成されたキュー接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーがキュー接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### setTemporaryModel *

```
public void setTemporaryModel(String x) throws JMSException
```

### getTemporaryModel *

```
public String getTemporaryModel()
```

### getReference *

```
public Reference getReference() throws NamingException
```

このキュー接続ファクトリーで使用される参照を作成します。

**戻り:** このオブジェクトで使用される参照

**投入:** NamingException

### setMessage Retention*

```
public void setMessageRetention(int x) throws JMSException
```

messageRetention 属性のメソッドを設定します。

**パラメーター:**

有効な値は、次のとおりです。

- JMSC.MQJMS_MRET_YES - 不要なメッセージを入力キューに置いたままにします。
- JMSC.MQJMS_MRET_NO - 不要なメッセージを後処理オプションに従って扱います。

### getMessage Retention*

```
public void getMessageRetention()
```

messageRetention 属性のメソッドを取得します。

**戻り:**

- JMSC.MQJMS_MRET_YES - 不要なメッセージを入力キューに置いたままにします。
- JMSC.MQJMS_MRET_NO - 不要なメッセージを後処理オプションに従って扱います。

# QueueReceiver

public interface **QueueReceiver**
extends **MessageConsumer**

MQSeries クラス: **MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

QueueReceiver は、キューに送達されたメッセージを受信するためにクライアントが使用します。

参照: **MessageConsumer**

このクラスは **MQMessageConsumer** から以下のメソッドを継承します。

- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

## メソッド

### getQueue

```
public Queue getQueue() throws JMSException
```

このキューの受信側に関連付けられているキューを取得します。

**戻り:** キュー。

**投入:** JMSException - 内部エラーが原因で、このキューの受信側のためのキューを JMS が取得できない場合。

# QueueRequestor

public class **QueueRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

サービス要求の作成を簡単にするために、 JMS はこの QueueRequestor ヘルパー・クラスを提供しています。 QueueRequestor コンストラクターは、トランザクション化されていない QueueSession および宛先の Queue に対して提供されています。これは応答に使用される TemporaryQueue を作成し、 request() メソッド (要求メッセージを送信し、その応答を待つ) を提供します。ユーザーはより洗練されたバージョンを自由に作成できます。

参照: **TopicRequestor**

## コンストラクター

### QueueRequestor

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                              throws JMSException
```

このインプリメンテーションでは、セッション・パラメーターがトランザクション化されておらず、 AUTO_ACKNOWLEDGE か DUPS_OK_ACKNOWLEDGE のいずれかであることを前提としています。

**パラメーター:**

- session - キューが属するキュー・セッション。
- queue - 要求 / 応答の呼び出しを実行するキュー。

**投入:** JMSException - JMS エラーが発生した場合。

## メソッド

### request

```
public Message request(Message message)
                                    throws JMSException
```

要求を送信し、応答を待ちます。 replyTo には一時キューが使用され、予期される応答は要求ごとに 1 つだけです。

**パラメーター:**

message - 送信するメッセージ。

**戻り:** 応答メッセージ。

**投入:** JMSException - JMS エラーが発生した場合。

## QueueRequestor

### close

```
public void close() throws JMSException
```

プロバイダーが QueueRequestor のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**注:** このメソッドは、 QueueRequestor コンストラクターに渡される Session オブジェクトをクローズします。

**投入:** JMSException - JMS エラーが発生した場合。

# QueueSender

public interface **QueueSender**
extends **MessageProducer**

MQSeries クラス: **MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

QueueSender は、クライアントがキューにメッセージを送信するのに使用します。

通常、QueueSender は特定の Queue に関連付けられます。ただし、何らかの特定の Queue と関連付けられていない、不特定 QueueSender を作成することもできます。

参照: **MessageProducer**

## メソッド

### getQueue

```
public Queue getQueue() throws JMSException
```

このキューの送信側に関連付けられているキューを取得します。

**戻り:** キュー。

**投入:** JMSException - 内部エラーが原因で、このキューの送信側のためのキューを JMS が取得できない場合。

### send

```
public void send(Message message) throws JMSException
```

メッセージをキューに送信します。 QueueSender のデフォルトの送達モード、存続時間、および優先順位を使用します。

**パラメーター:**

message - 送信されるメッセージ。

**投入:**

- JMSException - エラーが原因で、JMS がメッセージを送信できない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - Queue 送信側に無効なキューがある状況で、クライアントがこのメソッドを使用した場合。

### send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

## QueueSender

送達モード、優先順位、および存続時間を指定して、メッセージをキューに送信します。

**パラメーター:**

- message - 送信されるメッセージ。
- deliveryMode - 使用する送達モード。
- priority - このメッセージの優先順位。
- timeToLive - メッセージの存続時間 (ミリ秒単位)。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージを送信できない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - Queue 送信側に無効なキューがある状況で、クライアントがこのメソッドを使用した場合。

### send

```
public void send(Queue queue,
                 Message message) throws JMSException
```

QueueSender のデフォルトの送達モード、存続時間、および優先順位を使用して、指定したキューにメッセージを送信します。

**注:** このメソッドを使用できるのは、不特定 QueueSender だけです。

**パラメーター:**

- queue - このメッセージの送信先のキュー。
- message - 送信されるメッセージ。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージを送信できない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - 無効なキューを指定して、クライアントがこのメソッドを使用した場合。

### send

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

送達モード、優先順位、および存続時間を使用して、指定したキューにメッセージを送信します。

**注:** このメソッドを使用できるのは、不特定 QueueSender だけです。

**パラメーター:**

- queue - このメッセージの送信先のキュー。
- message - 送信されるメッセージ。

- deliveryMode - 使用する送達モード。
- priority - このメッセージの優先順位。
- timeToLive - メッセージの存続時間 (ミリ秒単位)。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージを送信できない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - 無効なキューを指定して、クライアントがこのメソッドを使用した場合。

### close *

```
public void close() throws JMSException
```

プロバイダーが QueueSender のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**投入:** JMSException (何らかのエラーが原因で、JMS がプロデューサーをクローズできない場合)。

**指定変更:**

クラス MQMessageProducer 内の close。

# QueueSession

public interface **QueueSession**
extends **Session**

MQSeries クラス: **MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

QueueSession は、QueueReceivers、QueueSenders、QueueBrowsers、および TemporaryQueues を作成するメソッドを提供します。

参照: **Session**

以下のメソッドは **MQSession** から継承されます。

- close
- commit
- rollback
- recover

## メソッド

### createQueue

```
public Queue createQueue(java.lang.String queueName)
                                          throws JMSException
```

Queue 名を指定して Queue を作成します。これを使用すれば、プロバイダー固有の名前が付いたキューを作成できます。 192ページに説明されているように、このストリングは URI 形式をとります。

**注:** この機能に依存しているクライアントは移植可能ではありません。

**パラメーター:**

queueName - このキューの名前。

**戻り:** 指定した名前の Queue。

**投入:** JMSException - JMS エラーが原因で、セッションがキューを作成できない場合。

### createReceiver

```
public QueueReceiver createReceiver(Queue queue)
                                          throws JMSException
```

指定したキューからメッセージを受信するための QueueReceiver を作成します。

**パラメーター:**

queue - アクセスするキュー。

**投入:**

- JMSException - JMS エラーが原因で、セッションが受信側を作成できない場合。
- InvalidDestinationException - 無効な Queue が指定された場合。

### createReceiver

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

指定したキューからメッセージを受信するための QueueReceiver を作成します。

**パラメーター:**

- queue - アクセスするキュー。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。

**投入:**

- JMSException - JMS エラーが原因で、セッションが受信側を作成できない場合。
- InvalidDestinationException - 無効な Queue が指定された場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

### createSender

```
public QueueSender createSender(Queue queue)
                                      throws JMSException
```

指定したキューにメッセージを送信するための QueueSender を作成します。

**パラメーター:**

queue - アクセスするキュー、または不特定プロデューサーの場合はヌル。

**投入:**

- JMSException - JMS エラーが原因で、セッションが送信側を作成できない場合。
- InvalidDestinationException - 無効な Queue が指定された場合。

### createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                                        throws JMSException
```

指定したキュー上のメッセージを見るための QueueBrowser を作成します。

**パラメーター:**

queue - アクセスするキュー。

**投入:**

- JMSException - JMS エラーが原因で、セッションがブラウザーを作成できない場合。
- InvalidDestinationException - 無効な Queue が指定された場合。

## QueueSession

### createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

指定したキュー上のメッセージを見るための QueueBrowser を作成します。

**パラメーター:**

- queue - アクセスするキュー。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。

**投入:**

- JMSException - JMS エラーが原因で、セッションがブラウザーを作成できない場合。
- InvalidDestinationException - 無効な Queue が指定された場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

### createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                    throws JMSException
```

一時キューを作成します。この存続時間は QueueConnection の存続期間と同じです (ただし、それより前に削除された場合は除く)。

**戻り:** 一時キュー。

**投入:** JMSException - JMS エラーが原因で、セッションが一時キューを作成できない場合。

# Session

public interface **Session**
extends **java.lang.Runnable**
サブインターフェース: **QueueSession**、**TopicSession**、**XAQueueSession**、**XASession**、および **XATopicSession**

MQSeries クラス: **MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

JMS Session は、メッセージを作成および消費するための単一スレッドのコンテキストです。

参照: **QueueSession**、**TopicSession**、 **XAQueueSession**、**XASession**、および **XATopicSession**

## フィールド

### AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

この確認通知モードを使用すると、受信のために呼び出しから正常に戻された時、またはメッセージを処理するために呼び出したメッセージ・リスナーが正常に戻された時に、セッションが自動的にメッセージを確認します。

### CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

この確認通知モードを使用すると、メッセージの確認通知メソッドを呼び出して、クライアントがメッセージを確認します。

### DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

この確認通知モードでは、メッセージの送達をセッションが確認しないように指示します。

## メソッド

### createBytesMessage

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

BytesMessage を作成します。 BytesMessage は、解釈されていないバイトのストリームを含むメッセージを送信するのに使用します。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

## Session

### createMapMessage

```
public MapMessage createMapMessage() throws JMSException
```

MapMessage を作成します。 MapMessage は、名前と値のペア (名前は String で、値は Java プリミティブ・タイプ) の自己定義のセットを送信するのに使用します。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### createMessage

```
public Message createMessage() throws JMSException
```

Message を作成します。 Message インターフェースは、すべての JMS メッセージのルート・インターフェースです。これはすべての標準メッセージ・ヘッダー情報を保持します。これはヘッダー情報だけを含んでいるメッセージで十分な場合に送信できます。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### createObjectMessage

```
public ObjectMessage createObjectMessage()
                              throws JMSException
```

ObjectMessage を作成します。 ObjectMessage は、直列化可能 Java オブジェクトを含むメッセージを送信するのに使用します。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### createObjectMessage

```
public ObjectMessage createObjectMessage
                             (java.io.Serializable object)
                              throws JMSException
```

初期化された ObjectMessage を作成します。 ObjectMessage は、直列化可能 Java オブジェクトを含むメッセージを送信するのに使用します。

**パラメーター:**

object - このメッセージを初期化するのに使用するオブジェクト。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### createStreamMessage

```
public StreamMessage createStreamMessage()
                              throws JMSException
```

StreamMessage を作成します。 StreamMessage は、 Java プリミティブの自己定義ストリームを送信するのに使用します。

**投入:** JMSException (内部エラーが原因で、JMS がこのメッセージを作成できない場合)。

### createTextMessage

```
public TextMessage createTextMessage() throws JMSException
```

TextMessage を作成します。 TextMessage は、String を含むメッセージを送信するのに使用します。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### createTextMessage

```
public TextMessage createTextMessage
                       (java.lang.String string)
                       throws JMSException
```

初期化された TextMessage を作成します。 TextMessage は、String を含むメッセージを送信するのに使用します。

**パラメーター:**

string - このメッセージを初期化するのに使用するストリング。

**投入:** JMSException - 内部エラーが原因で、JMS がこのメッセージを作成できない場合。

### getTransacted

```
public boolean getTransacted() throws JMSException
```

セッションがトランザクション化されたモードかどうか ?

**戻り:** セッションがトランザクション化されたモードの場合は true。

**投入:** JMSException - JMS Provider 内での内部エラーが原因で、トランザクション化されたモードを JMS が戻せない場合。

### commit

```
public void commit() throws JMSException
```

このトランザクションで扱われたすべてのメッセージをコミットし、現在保持されているあらゆるロックを解放します。

**投入:**

- JMSException - 内部エラーのため、JMS がトランザクションをコミットできない場合。
- TransactionRolledBackException - コミット中の内部エラーが原因で、トランザクションがロールバックする場合。

### rollback

```
public void rollback() throws JMSException
```

このトランザクションで扱われたあらゆるメッセージをロールバックし、現在保持されているあらゆるロックを解放します。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションでトランザクションをロールバックできない場合。

## Session

### close

```
public void close() throws JMSException
```

プロバイダーが Session のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

トランザクション化されたセッションをクローズすると、進行中のあらゆるトランザクションがロールバックされます。セッションをクローズすると、そのメッセージのプロデューサーおよびコンシューマーが自動的にクローズされるので、それらを個別にクローズする必要はありません。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションで Session をクローズできない場合。

### recover

```
public void recover() throws JMSException
```

このセッションでのメッセージの送達を停止し、最も古い未確認メッセージからメッセージの送達を再開します。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションでメッセージ送達の停止およびメッセージ送信の再開ができない場合。

### getMessageListener

```
public MessageListener getMessageListener()
                                  throws JMSException
```

セッション用に区別されているメッセージ・リスナーを戻します。

**戻り:** このセッションに関連付けられているメッセージ・リスナー。

**投入:** JMSException - JMS Provider 内の内部エラーが原因で、 JMS がメッセージ・リスナーを取得できない場合。

**参照:** setMessageListener

### setMessageListener

```
public void setMessageListener(MessageListener listener)
                                           throws JMSException
```

セッション用に区別されているメッセージ・リスナーを設定します。これが設定されている場合、そのセッションでメッセージ受信の他の形式は使用できません。それでも、メッセージ送信のすべての形式が引き続きサポートされています。

これは専門的な機能なので、通常の JMS クライアントは使用しません。

**パラメーター:**

listener - このセッションと関連付けるメッセージ・リスナー。

**投入:** JMSException - JMS Provider 内の内部エラーが原因で、 JMS がメッセージ・リスナーを設定できない場合。

**参照:** getMessageListener、ServerSessionPool、ServerSession

### run

```
public void run()
```

このメソッドは、アプリケーション・サーバーのみによる使用を目的としています。

**指定方法:**

インターフェース java.lang.Runnable 内の `run` による。

**参照:** ServerSession

# StreamMessage

public interface **StreamMessage**
extends **Message**

MQSeries クラス: **JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

StreamMessage は、 Java プリミティブのストリームを送信するのに使用します。

参照: **BytesMessage**、**MapMessage**、 **Message**、**ObjectMessage**、および **TextMessage**

## メソッド

### readBoolean

```
public boolean readBoolean() throws JMSException
```

ストリーム・メッセージからブール値を読み取ります。

**戻り:** 読み取られたブール値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readByte

```
public byte readByte() throws JMSException
```

ストリーム・メッセージからバイト値を読み取ります。

**戻り:** 8 ビットのバイトとして戻される、ストリーム・メッセージ内の次のバイト。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readShort

```
public short readShort() throws JMSException
```

ストリーム・メッセージから 16 ビットの数値を読み取ります。

**戻り:** ストリーム・メッセージからの 16 ビットの数値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readChar

```
public char readChar() throws JMSException
```

ストリーム・メッセージから Unicode 文字値を読み取ります。

**戻り:** ストリーム・メッセージからの Unicode 文字。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readInt

```
public int readInt() throws JMSException
```

ストリーム・メッセージから 32 ビットの整数を読み取ります。

**戻り:** int として解釈される、ストリーム・メッセージからの 32 ビットの整数値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readLong

```
public long readLong() throws JMSException
```

ストリーム・メッセージから 64 ビットの整数を読み取ります。

**戻り:** long として解釈される、ストリーム・メッセージからの 64 ビットの整数値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後に達した場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readFloat

```
public float readFloat() throws JMSException
```

ストリーム・メッセージから float を読み取ります。

**戻り:** ストリーム・メッセージからの float 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後に達した場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readDouble

```
public double readDouble() throws JMSException
```

ストリーム・メッセージから double を読み取ります。

**戻り:** ストリーム・メッセージからの double 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readString

```
public java.lang.String readString() throws JMSException
```

ストリーム・メッセージからストリングを読み込みます。

**戻り:** ストリーム・メッセージからの Unicode ストリング。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readBytes

```
public int readBytes(byte[] value)
             throws JMSExceptioneam message.
```

指定した byte[] オブジェクトへ、ストリーム・メッセージからバイト配列フィールドを読み取ります (バッファー読み取り)。バッファー・サイズがメッセージ・フィールド内のデータのサイズ以下の場合、データの残りを取り出すために、アプリケーションはこのメソッドへの呼び出しをさらに行わなければなりません。いったん byte[] フィールド値への最初の readBytes 呼び出しが行われると、そのフィールドの完全な値が読み取られてからでなければ、次のフィールドの読み取りは有効にはなりません。そのフィールドの完全な値が読み取られる前に次のフィールドの読み取りが試行されると、MessageFormatException が投げられます。

**パラメーター:**

value - データの読み込み先のバッファー。

**戻り:** バッファーに読み込まれるバイトの合計数か、または -1 (バイト・フィールドの最後になったためにデータがもうない場合)。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- MessageFormatException - この型変換が無効な場合。
- MessageNotReadableException - メッセージが書き込み専用モードの場合。

### readObject

```
public java.lang.Object readObject() throws JMSException
```

ストリーム・メッセージから Java オブジェクトを読み取ります。

**戻り:** オブジェクト・フォーマットでの、ストリーム・メッセージからの Java オブジェクト (たとえば int として設定されている場合、Integer が戻されます)。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。

- MessageEOFException - メッセージ・ストリームの最後が受信された場合。
- NotReadableException - メッセージが書き込み専用モードの場合。

### writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

ブール値をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれるブール値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを読み取れない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeByte

```
public void writeByte(byte value) throws JMSException
```

バイトをストリーム・メッセージに書き出します。

**パラメーター:**

value - 書き込まれるバイト値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeShort

```
public void writeShort(short value) throws JMSException
```

short をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる short。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeChar

```
public void writeChar(char value) throws JMSException
```

char をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる char 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeInt

```
public void writeInt(int value) throws JMSException
```

int をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる int。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeLong

```
public void writeLong(long value) throws JMSException
```

long をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる long。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeFloat

```
public void writeFloat(float value) throws JMSException
```

float をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる float 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeDouble

```
public void writeDouble(double value) throws JMSException
```

double をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる double 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeString

```
public void writeString(java.lang.String value)
                                          throws JMSException
```

ストリングをストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれる String 値。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

バイト配列をストリーム・メッセージに書き込みます。

**パラメーター:**

value - 書き込まれるバイト配列。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeBytes

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

バイト配列の一部をストリーム・メッセージに書き込みます。

**パラメーター:**

- value - 書き込まれるバイト配列の値。
- offset - バイト配列内の初期オフセット。
- length - 使用するバイト数。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。

- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### writeObject

```
public void writeObject(java.lang.Object value)
                                        throws JMSException
```

Java オブジェクトをストリーム・メッセージに書き込みます。このメソッドは、オブジェクト・プリミティブ・タイプ (Integer、Double、Long など)、ストリング、およびバイト配列の場合にのみ機能します。

**パラメーター:**

value - 書き込まれる Java オブジェクト。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージを書き込めない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。
- MessageFormatException - オブジェクトが無効な場合。

### reset

```
public void reset() throws JMSException
```

メッセージを読み取り専用モードにし、ストリームを先頭に再配置します。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がメッセージをリセットできない場合。
- MessageFormatException - メッセージのフォーマットが無効な場合。

# TemporaryQueue

public interface **TemporaryQueue**
extends **Queue**

MQSeries クラス: **MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

TemporaryQueue は、QueueConnection の継続期間中に作成される固有の Queue オブジェクトです。

## メソッド

### delete

```
public void delete() throws JMSException
```

この一時キューを削除します。それを既存の送信側または受信側がまだ使用している場合、JMSException が投げられます。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションで TemporaryQueue を削除できない場合。

# TemporaryTopic

public interface **TemporaryTopic**
extends **Topic**

MQSeries クラス: **MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

TemporaryTopic は、TopicConnection の継続期間中に作成される固有の Topic オブジェクトで、その接続のコンシューマーだけが消費できます。

## MQSeries コンストラクター

### MQTemporaryTopic

```
MQTemporaryTopic() throws JMSException
```

## メソッド

### delete

```
public void delete() throws JMSException
```

この一時トピックを削除します。それを既存のパブリッシャーまたはサブスクライバーがまだ使用している場合、 JMSException が投げられます。

**投入:** JMSException - 内部エラーのため、 JMS のインプリメンテーションで TemporaryTopic を削除できない場合。

# TextMessage

public interface **TextMessage**
extends **Message**

MQSeries クラス: **JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

TextMessage は、java.lang.String を含むメッセージを送信するのに使用します。これは Message を継承し、テキスト・メッセージの本文を追加します。

参照: **BytesMessage**、**MapMessage**、 **Message**、**ObjectMessage**、および **StreamMessage**

## メソッド

### setText

```
public void setText(java.lang.String string)
                                        throws JMSException
```

このメッセージのデータを含むストリングを設定します。

**パラメーター:**

string - メッセージのデータを含むストリング。

**投入:**

- JMSException - 内部 JMS エラーが原因で、JMS がテキストを設定できない場合。
- MessageNotWriteableException - メッセージが読み取り専用モードの場合。

### getText

```
public java.lang.String getText() throws JMSException
```

このメッセージのデータを含むストリングを取得します。デフォルト値はヌルです。

**戻り:** メッセージのデータを含むストリング。

**投入:** JMSException - 内部 JMS エラーが原因で、JMS がテキストを取得できない場合。

# Topic

public interface **Topic**
extends **Destination**
サブインターフェース: **TemporaryTopic**

MQSeries クラス: **MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Topic オブジェクトは、プロバイダー固有のトピック名をカプセル化します。このようにして、クライアントはトピックの識別を JMS メソッドに指定します。

参照: **Destination**

## MQSeries コンストラクター

### MQTopic

```
public MQTopic()
public MQTopic(string URItopic)
```

**TopicSession.createTopic** を参照してください。

## メソッド

### getTopicName

```
public java.lang.String getTopicName() throws JMSException
```

このトピックの名前を URI フォーマットで取得します。 (URI フォーマットについては、203ページの『実行時のトピックの作成』で説明されています。)

**注:** 名前に依存しているクライアントは移植可能ではありません。

**戻り:** トピック名。

**投入:** JMSException - 内部エラーが原因で、 Topic で使用される JMS のインプリメンテーションでトピックを戻せない場合。

### toString

```
public String toString()
```

トピック名を見栄えの良い印刷バージョンで戻します。

**戻り:** この Topic のプロバイダー固有の識別値。

**指定変更:**
クラス Object 内の toString。

### getReference *

```
public Reference getReference()
```

## Topic

このトピックで使用される参照を作成します。

**戻り:** このオブジェクトで使用される参照

**投入:** NamingException

### setBaseTopicName *

```
public void setBaseTopicName(String x)
```

基礎となる MQSeries トピック名で使用されるメソッドを設定します。

### getBaseTopicName *

```
public String getBaseTopicName()
```

基礎となる MQSeries トピック名で使用されるメソッドを取得します。

### setBrokerDurSubQueue *

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

brokerDurSubQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerDurSubQueue - 使用する永続サブスクリプション・キューの名前。

### getBrokerDurSubQueue *

```
public String getBrokerDurSubQueue()
```

brokerDurSubQueue 属性で使用されるメソッドを取得します。

**戻り:** 使用する永続サブスクリプション・キュー (brokerDurSubQueue) の名前。

### setBrokerCCDurSubQueue *

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

brokerCCDurSubQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerCCDurSubQueue - ConnectionConsumer で使用する永続サブスクリプション・キューの名前。

### getBrokerCCDurSubQueue *

```
public String getBrokerCCDurSubQueue()
```

brokerCCDurSubQueue 属性で使用されるメソッドを取得します。

**戻り:** ConnectionConsumer で使用する永続サブスクリプション・キュー (brokerCCDurSubQueue) の名前。

# TopicConnection

public interface **TopicConnection**
extends **Connection**
サブインターフェース: **XATopicConnection**

MQSeries クラス: **MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

TopicConnection は、 JMS パブリッシュ / サブスクライブ・プロバイダーへのアクティブな接続です。

参照: **Connection**、 **TopicConnectionFactory**、および **XATopicConnection**

## メソッド

### createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

TopicSession を作成します。

**パラメーター:**

- transacted - 真であれば、セッションがトランザクション化されます。
- acknowledgeMode - 以下のいずれかです。
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  コンシューマーまたはクライアントが、受け取ったあらゆるメッセージを確認するかどうかを指示します。セッションがトランザクション化されている場合、このパラメーターは無視されます。

**戻り:** 新たに作成されたトピック・セッション。

**投入:** JMSException - 内部エラーか、または特定のトランザクションおよびまたは確認通知モードのサポート不足が原因で、 JMS Connection がセッションを作成できない場合。

### createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                              (Topic topic,
                               java.lang.String messageSelector,
                               ServerSessionPool sessionPool,
                               int maxMessages)
                                        throws JMSException
```

### TopicConnection

この接続の接続コンシューマーを作成します。これは専門的な機能なので、通常の JMS クライアントは使用しません。

**パラメーター:**

- topic - アクセスするトピック。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。
- sessionPool - この接続コンシューマーに関連付けられるサーバー・セッション・プール。
- maxMessages - サーバー・セッションに一度に割り当てられるメッセージの最大数。

**戻り:** 接続コンシューマー。

**投入:**

- JMSException - 内部エラーか、または sessionPool の無効な引き数が原因で、 JMS Connection が接続コンシューマーを作成できない場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

**参照:** ConnectionConsumer

### createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                              (Topic topic,
                               java.lang.String subscriptionName
                               java.lang.String messageSelector,
                               ServerSessionPool sessionPool,
                               int maxMessages)
                                        throws JMSException
```

この接続の永続接続コンシューマーを作成します。これは専門的な機能なので、通常の JMS クライアントは使用しません。

**パラメーター:**

- topic - アクセスするトピック。
- subscriptionName - 永続サブスクリプションの名前。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。
- sessionPool - この永続接続コンシューマーに関連付けられるサーバー・セッション・プール。
- maxMessages - サーバー・セッションに一度に割り当てられるメッセージの最大数。

**戻り:** 永続接続コンシューマー。

**投入:**

- JMSException - 内部エラーか、または sessionPool および messageSelector の無効な引き数が原因で、 JMS Connection が接続コンシューマーを作成できない場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

**参照:** ConnectionConsumer

# TopicConnectionFactory

public interface **TopicConnectionFactory**
extends **ConnectionFactory**
サブインターフェース: **XATopicConnectionFactory**

MQSeries クラス: **MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

TopicConnectionFactory は、 JMS パブリッシュ / サブスクライブ・プロバイダーとの TopicConnection を作成するためにクライアントが使用します。

参照: **ConnectionFactory** および **XATopicConnectionFactory**

## MQSeries コンストラクター

### MQTopicConnectionFactory

```
public MQTopicConnectionFactory()
```

## メソッド

### createTopicConnection

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

デフォルトのユーザー ID が設定されている、トピック接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**戻り:** 新たに作成されたトピック接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーがトピック接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### createTopicConnection

```
public TopicConnection createTopicConnection
                                        (java.lang.String userName,
                                        java.lang.String password)
                                        throws JMSException
```

特定のユーザー ID が設定されている、トピック接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**注:** このメソッドが有効なのは、トランスポート・タイプが IBM_JMS_TP_CLIENT_MQ_TCPIP の場合だけです。 ConnectionFactory を参照してください。

**パラメーター:**

- userName - 呼び出し側のユーザー名。
- password - 呼び出し側のパスワード。

**戻り:** 新たに作成されたトピック接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーがトピック接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### setBrokerControlQueue *

```
public void setBrokerControlQueue(String x) throws JMSException
```

brokerControlQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerControlQueue - ブローカー制御キューの名前。

### getBrokerControlQueue *

```
public String getBrokerControlQueue()
```

brokerControlQueue 属性で使用されるメソッドを取得します。

**戻り:** ブローカーの制御キュー名。

### setBrokerQueueManager *

```
public void setBrokerQueueManager(String x) throws JMSException
```

brokerQueueManager 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerQueueManager - ブローカーのキュー・マネージャーの名前。

### getBrokerQueueManager *

```
public String getBrokerQueueManager()
```

brokerQueueManager 属性で使用されるメソッドを取得します。

**戻り:** ブローカーのキュー・マネージャー名。

### setBrokerPubQueue *

```
public void setBrokerPubQueue(String x) throws JMSException
```

brokerPubQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerPubQueue - ブローカーのパブリッシュ・キューの名前。

### getBrokerPubQueue *

```
public String getBrokerPubQueue()
```

brokerPubQueue 属性で使用されるメソッドを取得します。

**戻り:** ブローカーのパブリッシュ・キュー名。

**setBrokerSubQueue ***

```
public void setBrokerSubQueue(String x) throws JMSException
```

brokerSubQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerSubQueue - 使用する非永続サブスクリプション・キューの名前。

**getBrokerSubQueue ***

```
public String getBrokerSubQueue()
```

brokerSubQueue 属性で使用されるメソッドを取得します。

**戻り:** 使用する非永続サブスクリプション・キューの名前。

**setBrokerCCSubQueue ***

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

brokerCCSubQueue 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerSubQueue - ConnectionConsumer で使用する非永続サブスクリプション・キューの名前。

**getBrokerCCSubQueue ***

```
public String getBrokerCCSubQueue()
```

brokerCCSubQueue 属性で使用されるメソッドを取得します。

**戻り:** ConnectionConsumer で使用する非永続サブスクリプション・キューの名前。

**setBrokerVersion ***

```
public void setBrokerVersion(int x) throws JMSException
```

brokerVersion 属性で使用されるメソッドを設定します。

**パラメーター:**

brokerVersion - ブローカーのバージョン番号。

**getBrokerVersion ***

```
public int getBrokerVersion()
```

brokerVersion 属性で使用されるメソッドを取得します。

**戻り:** ブローカーのバージョン番号。

## TopicConnectionFactory

### getReference *

```
public Reference getReference()
```

このトピック接続ファクトリーで使用される参照を戻します。

**戻り:** このトピック接続ファクトリーで使用される参照。

**投入:** NamingException

# TopicPublisher

public interface **TopicPublisher**
extends **MessageProducer**

MQSeries クラス: **MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

TopicPublisher は、クライアントがトピックについてのメッセージをパブリッシュするのに使用します。 TopicPublisher は、JMS メッセージ・プロデューサーのパブリッシュ / サブスクライブのバリアントです。

## メソッド

### getTopic

```
public Topic getTopic() throws JMSException
```

このパブリッシャーと関連付けられているトピックを取得します。

**戻り:** このパブリッシャーのトピック

**投入:** JMSException - 内部エラーが原因で、このトピック・パブリッシャーのトピックを JMS が取得できない場合。

### publish

```
public void publish(Message message) throws JMSException
```

トピックにメッセージをパブリッシュします。トピックのデフォルトの送達モード、存続時間、および優先順位を使用します。

**パラメーター:**

message - パブリッシュするメッセージ。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージをパブリッシュできない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - Topic Publisher が無効なトピックを指定している状態で、クライアントがこのメソッドを使用した場合。

### publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

トピックに送達モード、優先順位、および存続時間を指定して、トピックにメッセージをパブリッシュします。

**パラメーター:**

- message - パブリッシュするメッセージ。
- deliveryMode - 使用する送達モード。
- priority - このメッセージの優先順位。
- timeToLive - メッセージの存続時間 (ミリ秒単位)。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージをパブリッシュできない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - Topic Publisher が無効なトピックを指定している状態で、クライアントがこのメソッドを使用した場合。

### publish

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

不特定メッセージ・プロデューサーの場合に、メッセージをトピックにパブリッシュします。トピックのデフォルトの送達モード、存続時間、および優先順位を使用します。

**パラメーター:**

- topic - このメッセージのパブリッシュ先のトピック。
- message - 送信するメッセージ。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージをパブリッシュできない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - 無効なトピックを指定している状態で、クライアントがこのメソッドを使用した場合。

### publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

不特定メッセージ・プロデューサーの場合に、送達モード、優先順位、および存続時間を指定して、トピックにメッセージをパブリッシュします。

**パラメーター:**

- topic - このメッセージのパブリッシュ先のトピック。
- message - 送信するメッセージ。
- deliveryMode - 使用する送達モード。
- priority - このメッセージの優先順位。
- timeToLive - メッセージの存続時間 (ミリ秒単位)。

**投入:**

- JMSException - 内部エラーが原因で、JMS がメッセージをパブリッシュできない場合。
- MessageFormatException - 無効なメッセージが指定された場合。
- InvalidDestinationException - 無効なトピックを指定している状態で、クライアントがこのメソッドを使用した場合。

### close *

```
public void close() throws JMSException
```

プロバイダーが TopicPublisher のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**投入:** JMSException (エラーが原因で、JMS がプロデューサーをクローズできない場合)。

**指定変更:**

クラス MQMessageProducer 内のクローズ。

# TopicRequestor

public class **TopicRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

この TopicRequestor クラスは、サービス要求の作成を助けるために JMS が提供しているものです。

TopicRequestor コンストラクターは、トランザクション化されていない TopicSession および宛先の Topic に対して提供されています。これは応答に使用される TemporaryTopic を作成し、 request() メソッド (要求メッセージを送信し、その応答を待つ) を提供します。ユーザーはより洗練されたバージョンを自由に作成できます。

## コンストラクター

### TopicRequestor

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

TopicRequestor クラスのコンストラクター。このインプリメンテーションでは、セッション・パラメーターがトランザクション化されておらず、AUTO_ACKNOWLEDGE か DUPS_OK_ACKNOWLEDGE のいずれかであることを前提としています。

**パラメーター:**

- session - トピックが属するトピック・セッション。
- topic - 要求 / 応答の呼び出しを実行するトピック。

**投入:** JMSException - JMS エラーが発生した場合。

## メソッド

### request

```
public Message request(Message message) throws JMSException
```

要求を送信し、応答を待ちます。

**パラメーター:**

message - 送信するメッセージ。

**戻り:** 応答メッセージ。

**投入:** JMSException - JMS エラーが発生した場合。

### close

```
public void close() throws JMSException
```

プロバイダーが TopicRequestor のために JVM の外側でいくらかのリソースを割り振る場合があるため、それらのリソースが必要でなければ、クライアントはそれらをクローズしなければなりません。これらのリソースを後で再利用するために、ガーベッジ・コレクションを当てにすることはできません。これは、当てにできるほどガーベッジ・コレクションはすぐには行われないためです。

**注:** このメソッドは、 TopicRequestor コンストラクターに渡される Session オブジェクトをクローズします。

**投入:** JMSException - JMS エラーが発生した場合。

# TopicSession

public interface **TopicSession**
extends **Session**

MQSeries クラス: **MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

TopicSession は、TopicPublisher、TopicSubscriber、および TemporaryTopic を作成するためのメソッドを提供します。

参照: **Session**

## MQSeries コンストラクター

### MQTopicSession

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

**TopicConnection.createTopicSession** を参照してください。

## メソッド

### createTopic

```
public Topic createTopic(java.lang.String topicName)
                                              throws JMSException
```

URI フォーマットの Topic 名を指定して Topic を作成します。 (URI フォーマットについては、203ページの『実行時のトピックの作成』で説明されています。) これを使用すれば、プロバイダー固有の名前が付いたトピックを作成できます。

**注:** この機能に依存しているクライアントは移植可能ではありません。

**パラメーター:**

topicName - このトピックの名前。

**戻り:** 指定した名前のトピック。

**投入:** JMSException - JMS エラーが原因で、セッションがトピックを作成できない場合。

### createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                     throws JMSException
```

指定したトピックに非永続サブスクライバーを作成します。

**パラメーター:**

topic - サブスクライブ先のトピック

**投入:**

- JMSException - JMS エラーが原因で、セッションがサブスクライバーを作成できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。

### createSubscriber

```
public TopicSubscriber createSubscriber
                            (Topic topic,
                             java.lang.String messageSelector,
                             boolean noLocal) throws JMSException
```

指定したトピックに非永続サブスクライバーを作成します。

**パラメーター:**

- topic - サブスクライブ先のトピック。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。この値はヌルでも構いません。
- noLocal - 設定すると、独自の接続によってパブリッシュされるメッセージの送達を禁止します。

**投入:**

- JMSException - JMS エラーまたは無効なセレクターが原因で、セッションがサブスクライバーを作成できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

### createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name) throws JMSException
```

指定したトピックに永続サブスクライバーを作成します。同じ名前の永続サブスクライバーと、新しいトピックかメッセージ・セレクター (あるいはその両方) を作成することにより、クライアントは既存の永続サブスクリプションを変更できます。

**パラメーター:**

- topic - サブスクライブ先のトピック。
- name - このサブスクリプションを識別するのに使用する名前。

**投入:**

- JMSException - JMS エラーが原因で、セッションがサブスクライバーを作成できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。

**TopicSession.unsubscribe** を参照してください。

### createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name,
                                     java.lang.String messageSelector,
                                     boolean noLocal)  throws JMSException
```

指定したトピックに永続サブスクライバーを作成します。

**パラメーター:**

- topic - サブスクライブ先のトピック。
- name - このサブスクリプションを識別するのに使用する名前。
- messageSelector - プロパティーがメッセージ・セレクター式と一致するメッセージだけが送達されます。この値はヌルでも構いません。
- noLocal - 設定すると、独自の接続によってパブリッシュされるメッセージの送達を禁止します。

**投入:**

- JMSException - JMS エラーまたは無効なセレクターが原因で、セッションがサブスクライバーを作成できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。
- InvalidSelectorException - メッセージ・セレクターが無効な場合。

### createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                                    throws JMSException
```

指定したトピックにパブリッシャーを作成します。

**パラメーター:**

topic - パブリッシュ先のトピック、または不特定プロデューサーの場合はヌル。

**投入:**

- JMSException - JMS エラーが原因で、セッションがパブリッシャーを作成できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。

### createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                     throws JMSException
```

一時トピックを作成します。この存続時間は TopicConnection の存続期間と同じです (ただし、それより前に削除された場合は除く)。

**戻り:** 一時トピック。

**投入:** JMSException - JMS エラーが原因で、セッションが一時トピックを作成できない場合。

### unsubscribe

```
public void unsubscribe(java.lang.String name)
                                      throws JMSException
```

クライアントによって作成された永続サブスクリプションをサブスクライブ解除します。

**注:** アクティブなサブスクリプションが存在する間はこのメソッドは使用しないでください。まず、サブスクライバーに対して close() を使用しなければなりません。

**パラメーター:**

name - このサブスクリプションを識別するのに使用する名前。

**投入:**

- JMSException - JMS エラーが原因で、 JMS が永続サブスクリプションをサブスクライブ解除できない場合。
- InvalidDestinationException - 無効なトピックが指定された場合。

# TopicSubscriber

public interface **TopicSubscriber**
extends **MessageConsumer**

MQSeries クラス: **MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

TopicSubscriber は、トピックにパブリッシュされたメッセージを受信するためにクライアントが使用します。 TopicSubscriber は、JMS メッセージ・コンシューマーのパブリッシュ / サブスクライブのバリアントです。

参照: **MessageConsumer** および **TopicSession.createSubscriber**

MQTopicSubscriber は MQMessageConsumer から以下のメソッドを継承します。

close
getMessageListener
receive
receiveNoWait
setMessageListener

## メソッド

### getTopic

```
public Topic getTopic() throws JMSException
```

このサブスクライバーと関連付けられているトピックを取得します。

**戻り:** このサブスクライバーのトピック。

**投入:** JMSException - 内部エラーが原因で、このトピック・サブスクライバーのトピックを JMS が取得できない場合。

### getNoLocal

```
public boolean getNoLocal() throws JMSException
```

この TopicSubscriber の NoLocal 属性を取得します。この属性のデフォルト値は false です。

**戻り:** ローカルでパブリッシュされるメッセージを継承する場合、true に設定されます。

**投入:** JMSException - 内部エラーが原因で、このトピック・サブスクライバーの NoLocal 属性を JMS が取得できない場合。

# XAConnection

public interface **XAConnection**
サブインターフェース: **XAQueueConnection** および **XATopicConnection**

MQSeries クラス: **MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection は、XASession を提供することによって Connection の機能を拡張したものです。 MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **XAQueueConnection** および **XATopicConnection**

# XAConnectionFactory

public interface **XAConnectionFactory**
サブインターフェース: **XAQueueConnectionFactory**
および **XATopicConnectionFactory**

MQSeries クラス: **MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

アプリケーション・サーバーの中には、 JTS 可能リソースの使用を分散トランザクションにグループ化するサポートを提供しているものもあります。 JMS トランザクションを JTS トランザクションに組み込むには、アプリケーション・サーバーは JTS 対応の JMS プロバイダーを必要とします。 JMS プロバイダーは JMS XAConnectionFactory を使用して、JTS サポートを公開します。この JMS XAConnectionFactory は、アプリケーション・サーバーが XASession を作成するのに使用します。 XAConnectionFactory は、ConnectionFactory と全く同様に JMS 管理対象オブジェクトです。それらを見つけるために、アプリケーション・サーバーは JNDI を使用することが求められます。

MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **XAQueueConnectionFactory** および **XATopicConnectionFactory**

# XAQueueConnection

public interface **XAQueueConnection**
extends **QueueConnection** および **XAConnection**

MQSeries クラス: **MQXAQueueConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQQueueConnection
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection は QueueConnection と同じ作成オプションを提供します。定義上の相違点は、XAConnection がトランザクション化されているという点だけです。MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **XAConnection** および **QueueConnection**

## メソッド

### createXAQueueSession

```
public XAQueueSession createXAQueueSession()
```

XAQueueSession を作成します。

**投入:** JMSException - 内部エラーが原因で、 JMS Connection が XA キュー・セッションを作成できない場合。

### createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

QueueSession を作成します。

**パラメーター:**

- transacted - 真であれば、セッションがトランザクション化されます。
- acknowledgeMode - コンシューマーまたはクライアントが、受け取ったあらゆるメッセージを確認するかどうかを指示します。以下の値を指定できます。
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  セッションがトランザクション化されている場合、このパラメーターは無視されます。

**戻り:** 新たに作成されたキュー・セッション (これは XA キュー・セッションではないので注意してください)。

## XAQueueConnection

**投入:** JMSException - 内部エラーが原因で、JMS Connection がキュー・セッションを作成できない場合。

# XAQueueConnectionFactory

public interface **XAQueueConnectionFactory**
extends **QueueConnectionFactory** および **XAConnectionFactory**

MQSeries クラス: **MQXAQueueConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

XAQueueConnectionFactory は QueueConnectionFactory と同じ作成オプションを提供します。 MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **QueueConnectionFactory** および **XAConnectionFactory**

## メソッド

### createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection()
                                              throws JMSException
```

デフォルトのユーザー ID を使用して XAQueueConnection を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**戻り:** 新たに作成された XA キュー接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーが XA キュー接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection
                                    (java.lang.String userName,
                                     java.lang.String password)
                                                    throws JMSException
```

特定のユーザー ID を使用して XA キュー接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**パラメーター:**

- userName - 呼び出し側のユーザー名。
- password - 呼び出し側のパスワード。

**戻り:** 新たに作成された XA キュー接続。

## XAQueueConnectionFactory

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーが XA キュー接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

# XAQueueSession

public interface **XAQueueSession**
extends **XASession**

MQSeries クラス: **MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

XAQueueSession は、QueueReceiver、QueueSender、および QueueBrowser を作成するのに使用できる正規の QueueSession を提供します。 MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

QueueSession に対応する XAResource は、 getXAResource メソッド (XASession から継承される) を呼び出すことにより取得できます。

参照: **XASession**

## メソッド

### getQueueSession

```
public QueueSession  getQueueSession()
                                 throws JMSException
```

この XAQueueSession に関連付けられているキュー・セッションを取得します。

**戻り:** キュー・セッション・オブジェクト。

**投入:** JMSException - JMS エラーが発生した場合。

# XASession

public interface **XASession**
extends **Session**
サブインターフェース: **XAQueueSession** および **XATopicSession**

MQSeries クラス: **MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession は、JTA 用に JMS プロバイダーのサポートへのアクセスを追加することにより、 Session の機能を拡張したものです。このサポートは javax.transaction.xa.XAResource オブジェクトの形式をとります。このオブジェクトの機能は、標準の X/Open XA Resource インターフェースで定義されているものとよく似ています。

アプリケーション・サーバーは、XASession のトランザクションの割り当ての制御を、その XAResource を取得することによって制御します。それは XAResource を使用して、トランザクションへのセッションの割り当てや、トランザクションでの作業の準備およびコミットなどを行います。

XAResource はいくつかの非常に高度な機能 (複数のトランザクションに対する作業のインターリーブや、進行中のトランザクションのリストの回復など) を提供します。

JTA 対応のJMS プロバイダーは、この機能性を完全に実装しなければなりません。これを行うために、JMS プロバイダーは XA をサポートするデータベースのサービスを使用するか、あるいはこの機能性を最初から実装します。

アプリケーション・サーバーのクライアントには、正規の JMS Session に相当するものが提供されます。バックグラウンドで、アプリケーション・サーバーは基礎となる XASession のトランザクション管理を制御します。

MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **XAQueueSession** および **XATopicSession**

## メソッド

### getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

呼び出し側に XA リソースを戻します。

**戻り:** 呼び出し側への XA リソース。

### getTransacted

```
public boolean getTransacted()
                       throws JMSException
```

必ず true が戻されます。

**指定方法:**

Session インターフェースでの getTransacted。

**戻り:** true - セッションがトランザクション化されたモードの場合。

**投入:** JMSException - JMS Provider 内での内部エラーが原因で、トランザクション化されたモードを JMS が戻せない場合。

### commit

```
public void commit()
            throws JMSException
```

XASession オブジェクトには、このメソッドは呼び出さないでください。呼び出すと、TransactionInProgressException が投げられます。

**指定方法:**

Session インターフェースでのコミット。

**投入:** TransactionInProgressException - XASession でこのメソッドが呼び出された場合。

### rollback

```
public void rollback()
              throws JMSException
```

XASession オブジェクトには、このメソッドは呼び出さないでください。呼び出すと、TransactionInProgressException が投げられます。

**指定方法:**

Session インターフェースでのロールバック。

**投入:** TransactionInProgressException - XASession でこのメソッドが呼び出された場合。

# XATopicConnection

public interface **XATopicConnection**
extends **TopicConnection** および **XAConnection**

MQSeries クラス: **MQXATopicConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

XATopicConnection は TopicConnection と同じ作成オプションを提供します。定義上の相違点は、XAConnection がトランザクション化されているという点だけです。MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **TopicConnection** および **XAConnection**

## メソッド

### createXATopicSession

```
public XATopicSession createXATopicSession()
                                      throws JMSException
```

XATopicSession を作成します。

**投入:** JMSException - 内部エラーが原因で、 JMS Connection が XA トピック・セッションを作成できない場合。

### createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                throws JMSException
```

TopicSession を作成します。

**指定方法:**
インターフェース TopicConnection での createTopicSession。

**パラメーター:**

- transacted - 真であれば、セッションがトランザクション化されます。
- acknowledgeMode - 以下のいずれかです。
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  コンシューマーまたはクライアントが、受け取ったあらゆるメッセージを確認するかどうかを指示します。セッションがトランザクション化されている場合、このパラメーターは無視されます。

**戻り:** 新たに作成されたトピック・セッション (これは XA トピック・セッションではないので注意してください)。

**投入:** JMSException - 内部エラーが原因で、 JMS Connection がトピック・セッションを作成できない場合。

# XATopicConnectionFactory

public interface **XATopicConnectionFactory**
extends **TopicConnectionFactory** および **XAConnectionFactory**

MQSeries クラス: **MQXATopicConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQTopicConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

XATopicConnectionFactory は TopicConnectionFactory と同じ作成オプションを提供します。 MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

参照: **TopicConnectionFactory** および **XAConnectionFactory**

## メソッド

### createXATopicConnection

```
public XATopicConnection createXATopicConnection()
                                        throws JMSException
```

デフォルトのユーザー ID を使用して XA トピック接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**戻り:** 新たに作成された XA トピック接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーが XA トピック接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

### createXATopicConnection

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                        throws JMSException
```

指定したユーザー ID を使用して XA トピック接続を作成します。接続は停止モードで作成されます。 Connection.start メソッドが明示的に呼び出されるまで、メッセージは送達されません。

**パラメーター:**

- userName - 呼び出し側のユーザー名
- password - 呼び出し側のパスワード

**戻り:** 新たに作成された XA トピック接続。

**投入:**

- JMSException - 内部エラーが原因で、JMS プロバイダーが XA トピック接続を作成できない場合。
- JMSSecurityException - ユーザー名またはパスワードが無効なため、クライアント認証が失敗した場合。

# XATopicSession

public interface **XATopicSession**
extends **XASession**

MQSeries クラス: **MQXATopicSession**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQXASession
            |
            +----com.ibm.mq.jms.MQXATopicSession
```

XATopicSession は、TopicSubscriber および TopicPublisher を作成するのに使用できる、 TopicSession を提供します。 MQ JMS が XA クラスを使用する方法の詳細については、387ページの『付録E. WebSphere での JMS JTA/XA インターフェース』を参照してください。

TopicSession に対応する XAResource は、 getXAResource メソッド (XASession から継承される) を呼び出すことにより取得できます。

参照: **TopicSession** および **XASession**

## メソッド

### getTopicSession

```
public TopicSession getTopicSession()
                            throws JMSException
```

この XATopicSession に関連付けられているトピック・セッションを取得します。

**戻り:** トピック・セッション・オブジェクト。

**投入:**

- JMSException - JMS エラーが発生した場合。

# 第4部 付録

# 付録A. 管理ツールのプロパティーとプログラマブル・プロパティーの間のマッピング

MQSeries Classes for Java Message Service は、MQ JMS 管理ツールを使用するかアプリケーション・プログラムで、管理対象オブジェクトのプロパティーを設定および照会するための機能を提供します。 表30 は、管理ツールで使用される各プロパティー名と、それが参照する対応するメンバー変数との間のマッピングを示しています。また、そのツールで使用される記号プロパティー値と、それに相当するプログラマブル・プロパティー値との間のマッピングについても示しています。

*表 30. 管理ツール内のプロパティーの表記とそれに相当するプログラマブル・プロパティーとの比較*

| プロパティー | メンバー変数名 | プロパティー値のマッピング | |
|---|---|---|---|
| | | ツール | プログラム |
| DESCRIPTION | description | | |
| TRANSPORT | transportType | • BIND<br>• CLIENT | JMSC.MQJMS_TP_BINDINGS_MQ<br>JMSC.MQJMS_TP_CLIENT_MQ_TCPIP |
| CLIENTID | clientId | | |
| QMANAGER | queueManager* | | |
| HOSTNAME | hostName | | |
| PORT | port | | |
| CHANNEL | channel | | |
| CCSID | CCSID | | |
| RECEXIT | receiveExit | | |
| RECEXITINIT | receiveExitInit | | |
| SECEXIT | securityExit | | |
| SECEXITINIT | securityExitInit | | |
| SENDEXIT | sendExit | | |
| SENDEXITINIT | sendExitInit | | |
| TEMPMODEL | temporaryModel | | |
| MSGRETENTION | messageRetention | • YES<br>• NO | JMSC.MQJMS_MRET_YES<br>JMSC.MQJMS_MRET_NO |
| BROKERVER | brokerVersion | • V1 | JMSC.MQJMS_BROKER_V1 |
| BROKERPUBQ | brokerPubQueue | | |
| BROKERSUBQ | brokerSubQueue | | |
| BROKERDURSUBQ | brokerDurSubQueue | | |
| BROKERCCSUBQ | brokerCCSubQueue | | |
| BROKERCCDSUBQ | brokerCCDurSubQueue | | |
| BROKERQMGR | brokerQueueManager | | |
| BROKERCONQ | brokerControlQueue | | |
| EXPIRY | expiry | • APP<br>• UNLIM | JMSC.MQJMS_EXP_APP<br>JMSC.MQJMS_EXP_UNLIMITED |

*表 30. 管理ツール内のプロパティーの表記とそれに相当するプログラマブル・プロパティーとの比較 (続き)*

| プロパティー | メンバー変数名 | プロパティー値のマッピング | |
|---|---|---|---|
| | | ツール | プログラム |
| PRIORITY | priority | • APP<br>• QDEF | JMSC.MQJMS_PRI_APP<br>JMSC.MQJMS_PRI_QDEF |
| PERSISTENCE | persistence | • APP<br>• QDEF<br>• PERS<br>• NON | JMSC.MQJMS_PER_APP<br>JMSC.MQJMS_PER_QDEF<br>JMSC.MQJMS_PER_PER<br>JMSC.MQJMS_PER_NON |
| TARGCLIENT | targetClient | • JMS<br>• MQ | JMSC.MQJMS_CLIENT_JMS_COMPLIANT<br>JMSC.MQJMS_CLIENT_NONJMS_MQ |
| ENCODING | encoding | | |
| QUEUE | baseQueueName | | |
| TOPIC | baseTopicName | | |
| **注:** * MQQueue オブジェクトの場合、メンバー変数名は baseQueueManagerName です。 | | | |

# 付録B. MQSeries classes for Java Message Service で提供されているスクリプト

以下のファイルは、MQ JMS インストール・システムの bin ディレクトリーにあります。これらのスクリプトは、 MQ JMS のインストールまたは使用中に実行する必要がある共通のタスクを支援するために提供されています。 表31 はスクリプトとその用途をリストしています。

*表 31. MQSeries classes for Java Message Service で提供されているユーティリティー*

| ユーティリティー | 用途 |
|---|---|
| IVTRun.bat<br>IVTTidy.bat<br>IVTSetup.bat | ポイント・ツー・ポイントのインストール検査テスト・プログラムを実行するために使用します (27ページの『ポイント・ツー・ポイント IVT の実行』を参照)。 |
| PSIVTRun.bat | パブリッシュ / サブスクライブのインストール検査テスト・プログラムを実行するために使用します (31ページの『パブリッシュ / サブスクライブのインストール検査テスト』を参照)。 |
| formatLog.bat | バイナリー・ログ・ファイルをプレーン・テキストに変換するのに使用します (36ページの『ロギング』を参照)。 |
| JMSAdmin.bat | 管理ツールを実行するのに使用します (37ページの『第5章 MQ JMS 管理ツールの使用』を参照)。 |
| JMSAdmin.config | 管理ツール用の構成ファイル (38ページの『構成』を参照)。 |
| runjms.bat | JMS アプリケーションを実行する際に役立つユーティリティー・スクリプト (34ページの『ユーザー独自の MQ JMS プログラムの実行』を参照)。 |
| PSReportDump.class | ブローカー・レポート・メッセージを表示するのに使用します (210ページの『ブローカー・レポートの処理』を参照)。 |
| **注:** UNIX システムでは、拡張子 '.bat' はファイル名から省略されます。 | |

## スクリプト

# 付録C. Java オブジェクト用の LDAP サーバー構成

MQ JMS によって管理されるオブジェクトを保管するのに JNDI を使用し、 JNDI サービス・プロバイダーとして LDAP を使用する場合、サーバーは LDAP v3 (たとえば、 SecureWay® eNetwork Directory v3.1) でなければならず、 Java オブジェクトを保管するように構成されていなければなりません。

## LDAP サーバー構成の検査

Java オブジェクトを受け入れられるように、LDAP サーバーがすでに構成されているかどうかを検査するには、 LDAP モードで MQ JMS 管理ツールを実行します (37ページの『管理ツールの起動』 を参照してください)。

以下のコマンドを使用して、テスト・オブジェクトの作成および表示を試行します。

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

例外が発生しなければ、サーバーは正しく構成されており、JMS オブジェクトの保管に進むことができます。

'SchemaViolationException' が戻された場合、またはメッセージ「オブジェクトをバインドできません」が表示された場合、サーバーは正しく構成されていません。 Java オブジェクトを保管するようにサーバーが構成されていないか、オブジェクトに対する許可またはサフィックスが正しくないかのどちらかです。以下の手順は、構成タスクを行う際に役立ちます。

## 構成手順

サーバーの管理を可能にするツールを多くの LDAP サーバーが提供しています。これらのツールの使用に関する詳細は、ご使用のサーバーの資料を参照してください。これらのツールを使用することにより、 '属性' および 'オブジェクト・クラス' 定義を含むスキーマを表示および更新することができます。

必要であれば追加して、スキーマに以下のオブジェクト・クラス定義が必ず含まれるようにしてください。

```
( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Container for a Java object'
  SUP top
  STRUCTURAL
  MUST ( cn )
)

( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Java object representation'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $
```

```
        javaCodebase $
        javaDoc $
        description )
)

( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Java serialized object'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)

( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'JNDI reference'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $
        javaFactory )
)
```

また、必要であればスキーマを更新して、スキーマに以下の属性定義が必ず含まれるようにしてください。

```
( 1.3.6.1.4.1.42.2.27.4.1.11
        NAME 'javaReferenceAddress'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.10
        NAME 'javaFactory'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.7
        NAME 'javaCodebase'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

更新が完了したら、LDAP サーバーを停止させてから再始動し、それから 383ページの『LDAP サーバー構成の検査』 で説明されている構成検査手順を繰り返してください。

# 付録D. MQSeries Integrator V2 への接続

MQSeries Integrator V2 は以下の用途で使用できます。

- MQ JMS 用のパブリッシュ / サブスクライブ・ブローカーとして
- JMS クライアント・アプリケーションによって作成されたメッセージを経路指定または変換するため、および JMS クライアントにメッセージを送信またはパブリッシュするため。

## パブリッシュ / サブスクライブ

MQSeries Integrator V2 は、 MQ JMS 用のパブリッシュ / サブスクライブ・ブローカーとして使用できます。そのようにするには、以下のセットアップ活動が必要です。

- 基本となる MQSeries

  まず、ブローカー・パブリケーション・キューを作成しなければなりません。これはブローカー・キュー・マネージャー上の MQSeries キューで、パブリケーションをブローカーに実行依頼するのに使用されます。このキューには独自の名前を選べますが、 TopicConnectionFactory の BROKERPUBQ プロパティーのキュー名と一致していなければなりません。デフォルトでは TopicConnectionFactory の BROKERPUBQ プロパティーは値 SYSTEM.BROKER.DEFAULT.STREAM に設定されているので、 TopicConnectionFactory 内で別の名前を構成するのでない限り、キューの名前には SYSTEM.BROKER.DEFAULT.STREAM を使用してください。

- MQSeries Integrator V2

  次のステップは、ブローカー用に実行グループ内のメッセージ・フロー をセットアップすることです。メッセージ・フローの目的は、ブローカー・パブリケーション・キューからメッセージを読み取ることです。 (必要であれば、複数のパブリケーション・キューをセットアップすることもできます。そのような場合、各キューごとに独自の TopicConnectionFactory とメッセージ・フローが必要です。)

  基本メッセージ・フローは、出力が Publication (または MQOutput) ノードの入力に接続されている、 MQInput ノード (SYSTEM.BROKER.DEFAULT.STREAM キューから読み取るように構成されている) から成り立っています。

  そのため、メッセージ・フローのダイアグラムは以下のようになっています。

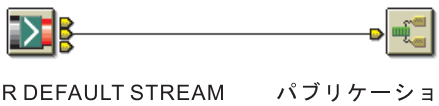


*図 7. MQSeries Integrator のメッセージ・フロー*

このメッセージ・フローを展開し、ブローカーを開始する場合、JMS アプリケーションの全体像から見ると、 MQSeries Integrator V2 ブローカーは MQSeries パ

ブリッシュ / サブスクライブ・ブローカーのように動作します。現在のサブスクリプションの状態は、MQSeries Integrator コントロール・センターを使用して表示できます。

**注:**

1. Java Message Service 用に MQSeries クラスに対して変更を行う必要はありません。
2. MQSeries パブリッシュ / サブスクライブ・ブローカーと MQSeries Integrator V2 ブローカーは、同一のキュー・マネージャーで共存することはできません。
3. MQSeries Integrator V2 のインストールおよびセットアップ手順については、*MQSeries Integrator for Windows NT バージョン 2.0 インストール・ガイド* で説明されています。

## 変換および経路指定

JMS クライアント・アプリケーションによって作成されたメッセージを経路指定または変換するため、および JMS クライアントにメッセージを送信またはパブリッシュするために、 MQSeries Integrator V2 を使用できます。

217ページの『MQRFH2 ヘッダー』で説明されているように、MQSeries JMS のインプリメンテーションでは、メッセージに関する情報を伝達するために MQRFH2 の mcd フォルダーが使用されます。デフォルトでは、メッセージがテキスト、バイト、ストリーム、マップ、またはオブジェクト・メッセージであるかどうかを識別するために、「メッセージ・ドメイン (Msd)」プロパティーが使用されます。

JMS アプリケーションがテキストまたはバイト・メッセージを作成する場合、アプリケーションはこの Msd プロパティーを指定変更でき、他の mcd フォルダー・フィールドを設定できます。これは、以下のような特別な URI フォーマットを持つ「JMS タイプ (JMS Type)」プロパティーを設定することによって行います。

```
mcd://domain/set/type[?format=fmt]
```

*domain*、*set*、*type*、および *fmt* フィールド (*fmt* はオプションです) 内の値は、出力 MQRFH2 にコピーされます。つまり、アプリケーションは、これらのフィールドを MQSeries Integrator V2 メッセージ・フローによって認識される値に設定できるということです。

# 付録E. WebSphere での JMS JTA/XA インターフェース

MQSeries classes for Java Message Service には JMS XA インターフェースが組み込まれています。これらを使用することにより、トランザクション・マネージャー (Java Transaction API (JTA) に準拠している) によって調整される 2 フェーズ・コミットに、 MQ JMS が参加できます。

このセクションでは、WebSphere Application Server アドバンスド版でこれらの機能を使用する方法について説明し、 WebSphere がグローバル・トランザクションで JMS の送信および受信操作、およびデータベースの更新を調整できるようにします。

WebSphere で MQ JMS および XA クラスを使用する前に、追加のインストールまたは構成ステップを行う場合があります。最新の情報については、MQSeries Java の使用 SupportPac Web ページ (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`) の `Readme.txt` ファイルを参照してください。

## WebSphere での JMS インターフェースの使用

このセクションでは、WebSphere Application Server アドバンスド版での JMS インターフェースの使用について説明します。

JMS プログラム、MQSeries、および EJB bean の基礎について理解していなければなりません。これらの詳細については、JMS の仕様書、EJB V2 の仕様書 (両方とも Sun から入手できます)、本書、MQ JMS で提供されているサンプル、および MQSeries と WebSphere に関する他のマニュアルで扱われています。

### 管理対象オブジェクト

JMS はベンダー固有の情報をカプセル化するのに管理対象オブジェクトを使用します。これにより、エンド・ユーザー・アプリケーションに対するベンダー固有の詳細の影響が最小限に抑えられます。管理対象オブジェクトは JNDI ネーム・スペースに保管され、ベンダー固有の内容を知らなくても、移植可能な方法で検索したり、使用したりすることができます。

スタンドアロンで使用する場合、MQ JMS は以下のクラスを提供します。

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere は MQ JMS が WebSphere と統合できるように、管理対象オブジェクトの以下のペアをさらに提供しています。

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

これらのオブジェクトは、 MQQueueConnectionFactory および MQTopicConnectionFactory と全く同じ方法で使用できます。ただし、それらは内部で JMS クラスの XA バージョンを使用し、 MQ XAResource を WebSphere トランザクションに含めます。

## コンテナー管理トランザクションと bean 管理トランザクションとの比較

コンテナー管理トランザクションは、 EJB コンテナーによって自動的に境界が定められている EJB bean 内のトランザクションです。 bean 管理トランザクションは、プログラムによって (UserTransaction インターフェースを介して) 境界が定められている、 EJB bean 内のトランザクションです。

## 2 フェーズ・コミットと 1 フェーズ最適化の比較

特定のトランザクションで複数の XAResource が使用される場合、 WebSphere コーディネーターは純粋な 2 フェーズ・コミットを呼び出すだけです。リソースを 1 つだけ呼び出すトランザクションは、1 フェーズ最適化を使用してコミットされます。これにより、分散および非分散トランザクションでさまざまな ConnectionFactory を使用する必要がほとんどなくなります。

## 管理対象オブジェクトの定義

WebSphere 固有の接続ファクトリーを定義して、それらを JNDI ネーム・スペースに保管するために、 MQ JMS 管理ツールを使用できます。 *MQ_install_dir*/bin 内の admin.config ファイルに、以下の行を含めてください。

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://hostname/
```

*MQ_install_dir* は MQ JMS のインストール・ディレクトリーで、 *hostname* は WebSphere を実行するマシンの名前または IP アドレスです。

com.ibm.ejs.ns.jndi.CNInitialContextFactory にアクセスするには、 WebSphere lib ディレクトリーから CLASSPATH にファイル ejs.jar を追加しなければなりません。

新しいファクトリーを作成するには、以下の 2 つの新しいタイプを設定して、定義動詞を使用します。

```
def WSQCF(name) [properties]
def WSTCF(name) [properties]
```

これらの新しいタイプは、同等の QCF または TCF タイプと同じプロパティーを使用します。ただし、許可されているのが BIND トランスポート・タイプだけであるという点が異なります (そのため、クライアント・プロパティーは構成できません)。詳細については、42ページの『JMS オブジェクトの管理』を参照してください。

## 管理オブジェクトの検索

EJB bean では、以下のように InitialContext.lookup() メソッドを使用して、 JMS 管理オブジェクトを検索します。

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

オブジェクトは汎用 JMS インターフェースにキャストして、汎用 JMS インターフェースとして使用できます。通常、MQSeries 特有のクラスをアプリケーション・コードにプログラミングする必要はありません。

## サンプル

WebSphere Application Server アドバンスド版で MQ JMS を使用する際の基本を説明するサンプルが 3 つあります。これらは *MQ_install_dir*/samples/ws (*MQ_install_dir* は MQ JMS のインストール・ディレクトリー) のサブディレクトリー内にあります。

- Sample1 は、コンテナー管理トランザクションの使用による、キュー内のメッセージに関する単純な put および get を示しています。
- Sample2 は、bean 管理トランザクションの使用による、キュー内のメッセージに関する単純な put および get を示しています。
- Sample3 はパブリッシュ / サブスクライブ API の使用を示しています。

EJB bean の作成および配置に関する詳細については、 WebSphere Application Server の資料を参照してください。

各サンプル・ディレクトリー内の readme.txt ファイルには、各 EJB bean からの出力例が含まれています。提供されているスクリプトは、ローカル・マシンでデフォルトのキュー・マネージャーが使用可能であることを前提としています。インストール・システムがデフォルトとは異なる場合、必要に応じてそれらのスクリプトを編集できます。

### Sample1

Sample1EJB.java (sample1 ディレクトリー内にある) は、 JMS を使用する以下の 2 つのメソッドを定義しています。

- putMessage() は、TextMessage をキューに送信し、送信メッセージの MessageID を戻します。
- getMessage() は、指定した MessageID を持つメッセージをキューから読み取ります。

このサンプルを使用する前に、以下の 2 つの管理対象オブジェクトを WebSphere JNDI ネーム・スペースに保管しなければなりません。

**QCF1** WebSphere 特有のキュー接続ファクトリー

**Q1** キュー

両方のオブジェクトとも、jms/Samples サブコンテキストにバインドしなければなりません。

管理対象オブジェクトをセットアップするには、 MQ JMS 管理ツールを使用して手動でそれらをセットアップするか、あるいは提供されているスクリプトを使用することができます。

WebSphere ネーム・スペースにアクセスするには、MQ JMS 管理ツールを構成しなければなりません。管理ツールの構成方法についての詳細は、39ページの『WebSphere のための構成』を参照してください。

典型的なデフォルト設定を使用して管理対象オブジェクトをセットアップするには、以下のコマンドを入力して、スクリプト admin.scp を実行できます。

```
JMSAdmin < admin.scp
```

bean は、getMessage および putMessage メソッドを TX_REQUIRED としてマークして配置しなければなりません。これにより、確実にコンテナーは各メソッドが入力される前にトランザクションを開始し、メソッドの完了時にトランザクションをコミットするようになります。メソッド内では、トランザクション状態に関連するアプリケーション・コードは何も必要ありません。ただし、putMessage から送信されるメッセージは同期点で生じるので、トランザクションがコミットされるまで使用可能にはならないことを覚えておいてください。

sample1 ディレクトリーには、 EJB bean を呼び出すための単純なクライアント・プログラム Sample1Client.java があります。また、このプログラムの実行を簡単にするためのスクリプト runClient もあります。

クライアント・プログラム (またはスクリプト) はパラメーターを 1 つとります。このパラメーターは、EJB bean putMessage メソッドによって送信される TextMessage の本文として使用されます。それから getMessage が呼び出されて、キューからメッセージを読み取り、本文をクライアントに戻して表示させます。 EJB bean はアプリケーション・サーバーの標準出力 (stdout) に進行メッセージを送信するので、実行中にその出力をモニターすることもできます。

クライアントからリモートであるマシンにアプリケーション・サーバーがある時、Sample1Client.java を編集しなければならない場合があります。デフォルトを使用しない時、ローカル・インストール・パスおよび配置される jar ファイルの名前に一致するように、 runClient スクリプトを編集しなければならない場合があります。

## Sample2

Sample2EJB.java (sample2 ディレクトリーにある) は、sample1 と同じタスクを実行し、同じ管理対象オブジェクトを必要とします。 sample1 との相違点は、 sample2 はトランザクション境界を制御するのに bean 管理トランザクションを使用するという点です。

まだ sample1 を実行していない場合、389ページの『Sample1』で説明されているように、必ず管理対象オブジェクト QCF1 および Q1 をセットアップしてください。

putMessage メソッドおよび getMessage メソッドは、 UserTransaction のインスタンスを取得することにより開始します。それらはこのインスタンスを使用して、UserTransaction.begin() メソッドを介してトランザクションを作成します。その後は、各メソッドの最後までコードの主な本体は sample1 と同じです。各メソッドの最後で、トランザクションは UserTransaction.commit() 呼び出しによって完了されます。

sample2 ディレクトリーには、 EJB bean を呼び出すための単純なクライアント・プログラム Sample2Client.java があります。また、このプログラムの実行を簡単にするためのスクリプト runClient もあります。これらは、389ページの『Sample1』で説明されているのと同じ方法で使用できます。

## Sample3

Sample3EJB.java (sample3 ディレクトリーにある) は、 WebSphere でのパブリッシュ / サブスクライブ API の使用を示しています。メッセージのパブリッシュは、ポイント・ツー・ポイントの場合とよく似ています。ただし、TopicSubscriber を使用してメッセージを受信する際に相違点があります。

パブリッシュ / サブスクライブ・プログラムは、普通、非永続サブスクライバーを使用します。これらの非永続サブスクライバーは、それらが所有しているセッションの存続時間の間だけ存在します (サブスクライバーが明示的にクローズされた場合はもっと短くなります)。またそれらがブローカーからメッセージを受信するのは、その存続時間の間だけです。

sample1 をパブリッシュ / サブスクライブに変換するために、 putMessage の QueueSender を TopicPublisher に、 getMessage の QueueReceiver を非永続 TopicSubscriber に置換するかもしれません。しかしこれは、メッセージの送信時に、ブローカーがトピックへの何らかのサブスクライバーを認識しないので失敗します。そのため、そのメッセージは廃棄されます。

解決方法として、メッセージがパブリッシュされる前に永続サブスクライバーを作成することができます。永続サブスクライバーはセッションの存続時間を超えて、送達可能エンドポイントとして存続します。そのため、メッセージは getMessage() への呼び出し中の検索で入手可能になります。

EJB bean には、以下の 2 つの追加メソッドが組み込まれています。

- createSubscription (永続サブスクリプションを作成する)
- destroySubscription (永続サブスクリプションを削除する)

これらのメソッドは (putMessage および getMessage とともに)、 TX_REQUIRED 属性を指定して配置しなければなりません。

sample3 を使用する前に、以下の 2 つの管理対象オブジェクトを WebSphere JNDI ネーム・スペースに保管しなければなりません。

TCF1

T1

両方のオブジェクトとも、jms/Samples サブコンテキストにバインドしなければなりません。

管理対象オブジェクトをセットアップするには、 MQ JMS 管理ツールを使用して手動でそれらをセットアップするか、あるいはスクリプトを使用することができます。 sample3 ディレクトリーには、スクリプト `admin.scp` があります。

WebSphere ネーム・スペースにアクセスするには、MQ JMS 管理ツールを構成しなければなりません。管理ツールの構成方法についての詳細は、39ページの『WebSphere のための構成』を参照してください。

典型的なデフォルト設定を使用して管理対象オブジェクトをセットアップするには、以下のコマンドを入力して、スクリプト `admin.scp` を実行できます。

```
JMSAdmin < admin.scp
```

すでに admin.scp を実行して sample1 または sample2 用にオブジェクトをセットアップしてある場合、 sample3 で admin.scp を実行するとエラー・メッセージが表示されます。 (これらは jms および Samples サブコンテキストの作成を試行した時に発生します。) これらのエラー・メッセージは無視して構いません。

また、sample3 を実行する前に、 MQSeries パブリッシュ / サブスクライブ・ブローカー (SupportPac MA0C) が必ずインストールされ、実行されているようにしてください。

sample3 ディレクトリーには、 EJB bean を呼び出すための単純なクライアント・プログラム Sample3Client.java があります。また、このプログラムの実行を簡単にするためのスクリプト runClient もあります。これらは、389ページの『Sample1』で説明されているのと同じ方法で使用できます。

# 付録F. 特記事項

本書はアメリカ合衆国で提供されている製品およびサービス用に作成されたものであり、本書に記載の製品、サービス、またはフィーチャーが日本においては提供されていない場合があります。日本で利用可能な製品、サービス、およびフィーチャーについては、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品 、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の 知的所有権を侵害することのない、機能的に同等な製品、プログラム、またはサービスを使用することができます。ただし、IBM 製以外の製品と組み合わせた場合、その操作の評価と検証については、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

**以下の保証は、国または地域の法律に沿わない場合は、適用されません。** IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適確な表現や誤植など) は、本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手してください。 IBM は、それらの製品のテストは行っておりません。また、IBM 以外の製品に関するパフォーマンスの正確性、互換性、またはその他の要求は確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

## 商標

次のものは、IBM Corporation の米国およびその他の国における商標です。

| | | |
|---|---|---|
| AIX | AS/400 | BookManager |
| CICS | IBM | IBMLink |
| Language Environment | MQSeries | MVS/ESA |
| OS/2 | OS/390 | OS/400 |
| SecureWay | SupportPac | System/390 |
| S/390 | VisualAge | VSE/ESA |
| WebSphere | | |

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

Microsoft、Windows、および Windows NT は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

# 用語集および略語集

この用語集では、本書で使用されている用語と、日常とは異なる意味で使用されている語を定義しています。定義は、1 つの用語に適用するだけでなく、本書で使用される語に特定の意味付けをしている場合があります。

探している用語が見つからない場合には、本書の索引または *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994 を参照してください。

## ［ア行］

**アプリケーション・プログラミング・インターフェース (Application Programming Interface (API)).** プログラマーが自分のアプリケーションで使用できる関数と変数をまとめたもの。

**アプレット (applet).** Web ページでのみ稼働するように設計されている Java プログラム。

**インスタンス (instance).** インスタンスはオブジェクトのことを指す。オブジェクトを生成する目的でクラスがインスタンス化された場合、オブジェクトはそのクラスのインスタンスとなる。

**インターネット (Internet).** インターネットは、共用情報の協調公衆ネットワークを指す。物理的には、インターネットは、現存のすべての公衆通信ネットワーク・リソース全体のサブセットを使用する。技術的には、インターネットが協調公衆ネットワークとして区別される点は、 TCP/IP (伝送制御プロトコル / インターネット・プロトコル) と呼ばれる一連のプロトコルを使用する点である。

**インターネット ORB 間プロトコル (Internet Inter-ORB Protocol (IIOP)).** 異なるベンダーからの ORB 間の TCP/IP 通信の規格。

**インターフェース (interface).** インターフェースは、抽象メソッドだけが含まれ、インスタンス変数は 1 つも含まれないクラスを指す。インターフェースは幾つかの異なるクラスから成るサブクラスによって導入できる一連の共用メソッドを提供する。

**オブジェクト (object).** (1) Java では、オブジェクトはクラスのインスタンスを指す。クラスは事柄のグループをモデル化し、オブジェクトはそのグループの特定メンバーをモデル化する。 (2) MQSeries では、オブジェクトはキュー・マネージャー、キュー、またはチャネルを指す。

**オブジェクト管理グループ (Object Management Group (OMG)).** オブジェクト指向プログラミングでの規格を定める団体。

**オブジェクト・リクエスト・ブローカー (Object Request Broker (ORB)).** 異なる言語で作成され、異機種の分散環境の異なるマシンで実行されるオブジェクト間にインターオペラビリティーを提供するアプリケーション・フレームワーク。

## ［カ行］

**カプセル化 (encapsulation).** カプセル化は、オブジェクトのデータを専用またはプロテクト状態にし、プログラマーがメソッド呼び出しのみによってデータにアクセスおよびデータを操作できるオブジェクト指向プログラミング手法である。

**キャスト (casting).** オブジェクトまたはプリミティブ・タイプの値の別のタイプへの明示的な変換を説明するために、 Java で使用される用語。

**キュー (queue).** キューは MQSeries オブジェクトの 1 つである。メッセージ・キューイング・アプリケーションはメッセージをキューに書き込み、メッセージをキューから読み取ることができる。

**キュー・マネージャー (queue manager).** キュー・マネージャーは、メッセージ・キューイング・サービスをアプリケーションに提供するシステム・プログラムである。

**共用 (public).** 共用クラスまたは共用インターフェースはすべての場所で表示される。共用メソッドまたは共用変数は、そのクラスが表示されるすべての場所で表示される。

**クライアント (client).** MQSeries では、クライアントは、サーバー上でキューイング・サービスへのアクセスをローカル・ユーザー・アプリケーションに提供する実行時コンポーネントを指す。

**クラス (class).** データおよびそのデータに対して操作するメソッドを 1 つにカプセル化した集合。クラス

は、そのクラスのインスタンスとなるオブジェクトを生成する目的でインスタンス化される場合がある。

## ［サ行］

**サーバー (server).** (1) MQSeries サーバーは、メッセージ・キューイング・サービスをリモート・ワークステーションで実行中のクライアント・アプリケーションに提供するキュー・マネージャーを指す。 (2) より一般的には、サーバーは、クライアント /サーバーの特定の 2 つのプログラムの情報フロー・モデルで情報の要求に応答するプログラムを指す。 (3) サーバー・プログラムが実行されるコンピューター。

**サーブレット (servlet).** Web サーバーでのみ稼働するように設計されている Java プログラム。

**サブクラス (subclass).** サブクラスは他のクラスを拡張するクラスである。サブクラスはそのスーパークラスの共用メソッドと変数およびプロテクト・メソッドと変数を継承する。

**スーパークラス (superclass).** スーパークラスは、他のクラスによって拡張されるクラスである。スーパークラスの共用メソッド変数およびプロテクト・メソッドと変数はサブクラスで使用可能である。

**専用 (private).** 専用フィールドは、それを所有するクラスの外側では表示されない。

## ［タ行］

**多重定義 (overloading).** 同一の有効範囲内で 1 つの ID が複数の項目を参照している状態。 Java では、メソッドは多重定義できるが、変数や演算子はできない。

**チャネル (channel).** MQI チャネル (MQI Channel) を参照。

**伝送制御プロトコル / インターネット・プロトコル (Transmission Control Protocol/Internet Protocol (TCP/IP)).** ローカル・エリア・ネットワークと広域ネットワークの両方に対等通信接続機能をサポートする一連の通信プロトコル。

## ［ハ行］

**ハイパーテキスト・マークアップ言語 (Hypertext Markup Language (HTML)).** ワールド・ワイド・ウェブ (WWW) に表示する情報を定義するために使用する言語。

**パッケージ (package).** Java におけるパッケージは、一片の Java コードに特定のクラス・セットへのアクセス権を提供する方法を指す。ある特定パッケージの一部である Java コードには、そのパッケージ内のすべてのクラスへのアクセス権と、それらのクラス内のすべての非専用メソッドおよびフィールドへのアクセス権がある。

**プロテクト (protected).** プロテクト・フィールドは、それを所有するクラス内、サブクラス内、またはそのクラスが構成されているパッケージ内でのみ表示される。

## ［マ行］

**メソッド (method).** メソッドは、関数またはプロシージャーに対応するオブジェクト指向プログラミング用語である。

**メッセージ (message).** メッセージ・キューイング・アプリケーションでは、メッセージはプログラム間で送信される通信を指す。

**メッセージ・キュー (message queue).** キュー (queue) を参照。

**メッセージ・キューイング (message queuing).** アプリケーション内の各プログラムが、キューにメッセージを書き込むことによって他のプログラムと連絡するプログラミング手法。

## ［ワ行］

**ワールド・ワイド・ウェブ (World Wide Web (WWW)).** WWW は一連の共通プロトコルに基づいたインターネット・サービス。これにより、特別に構成済みのサーバー・コンピューターが、通常の方法でインターネットを通じて文書を配布できるようになる。

## A

**Abstract Window Toolkit for Java (AWT).** ネイティブ・プラットフォーム版のコンポーネントを使用してインプリメントされる、グラフィカル・ユーザー・インターフェース (GUI) コンポーネントの集合。

**API.** アプリケーション・プログラミング・インターフェース。

**AWT.** Abstract Window Toolkit for Java。

## E

**EJB.** Enterprise JavaBeans

**Enterprise JavaBeans (EJB).** 再利用可能なビジネス・ロジックおよび移植可能なエンタープライズ・アプリケーションを作成するために Sun Microsystems によって配布されている、サーバー側のコンポーネント・アーキテクチャー。 Enterprise JavaBean コンポーネントは完全に Java で作成され、あらゆる EJB 準拠サーバーで実行される。

## H

**HTML.** ハイパーテキスト・マークアップ言語。

## I

**IEEE.** 米国電気電子学会。

**IIOP.** インターネット ORB 間プロトコル。

## J

**J2EE.** Java 2 Platform, Enterprise Edition

**JAAS.** Java Authentication and Authorization Service

**Java 2 Platform, Enterprise Edition (J2EE).** 複数層の Web ベース・アプリケーションを開発する機能を提供する、サービス、API、およびプロトコルのセット。

**Java Authentication and Authorization Service (JAAS).** エンティティー認証およびアクセス制御を提供する Java サービス。

**Java Developer Kit (JDK).** Sun Microsystems 社が Java 開発者向けに配布しているパッケージ。このパッケージには、Java インタープリター、Java クラス、および Java 開発ツール (コンパイラー、デバッガー、逆アセンブラー、アプレット・ビューアー、スタブ・ファイル生成プログラム、および文書生成プログラム) が含まれている。

**Java Naming and Directory Interface (JNDI).** Java プログラム言語で指定されている API。 Java プログラム言語で作成されたアプリケーションに、命名およびディレクトリー機能を提供する。

**Java Runtime Environment (JRE).** コア実行可能ファイル、および標準の Java プラットフォームを構成するファイルを含んでいる、 Java 開発キット (JDK) のサブセット。 JRE には Java 仮想マシン、コア・クラス、およびサポートするファイルが含まれている。

**Java Transaction API (JTA).** アプリケーションおよび J2EE サーバーがトランザクションにアクセスできるようにする API。

**Java 仮想マシン (Java Virtual Machine (JVM)).** コンパイルされた Java コード (アプレットおよびアプリケーション) を実行する、中央演算処理装置 (CPU) のソフトウェア・インプリメンテーション。

**Java トランザクション・サービス (Java Transaction Service (JTS)).** JTA をサポートし、 API のレベルより低い OMG オブジェクト・トランザクション・サービス 1.1 仕様の Java マッピングをインプリメントするトランザクション・マネージャー。

**Java メッセージ・サービス (Java Message Service (JMS)).** Java プログラムからエンタープライズ・メッセージング・システムにアクセスするための Sun Microsystem の API。

**JDK.** Java Developer Kit

**JMS.** Java メッセージ・サービス。

**JNDI.** Java Naming and Directory Interface

**JRE.** Java Runtime Environment

**JTA.** Java Transaction API

**JTS.** Java トランザクション・サービス。

**JVM.** Java 仮想マシン。

## L

**LDAP.** Lightweight Directory Access Protocol

**Lightweight Directory Access Protocol (LDAP).** ディレクトリー・サービスにアクセスするためのクライアント / サーバー・プロトコル。

## M

**MQDLH.** MQSeries 送達不能ヘッダー。 *MQSeries アプリケーション・プログラミング・リファレンス* を参照。

**MQI チャネル (MQI channel).** MQI チャネルは、MQSeries クライアントをサーバー・システム上のキュー・マネージャーに接続し、 MQI 呼び出しと応答を両方向に転送する。

**MQMD.** MQSeries メッセージ記述子。

**MQSC.** MQSeries コマンド。

**MQSeries.** MQSeries はメッセージ・キューイング・サービスを提供する IBM ライセンス・プログラム・ファミリー。

**MQSeries コマンド (MQSeries commands (MQSC)).** MQSeries オブジェクトを操作するのに使用するコマンド。人間が見て理解でき、すべてのプラットフォームで同じである。

**MQSeries メッセージ記述子 (MQSeries Message Descriptor (MQMD)).** MQSeries メッセージの一部として送られる、メッセージ・フォーマットおよびプロパティーを記述する制御情報。

# O

**OMG.** オブジェクト管理グループ。

**ORB.** オブジェクト・リクエスト・ブローカー。

# R

**Red Hat Package Manager (RPM).** Red Hat Linux プラットフォームおよび他の Linux および UNIX プラットフォームで使用する、ソフトウェア・パッケージング・システム。

**RPM.** Red Hat Package Manager

# T

**TCP/IP.** 伝送制御プロトコル / インターネット・プロトコル (Transmission Control Protocol/Internet Protocol)。

# U

**Uniform Resource Locator (URL).** コンピューター上またはネットワーク (インターネットなど) 内の情報リソースを表す一連の文字。

**URL.** Uniform Resource Locator

# V

**VisiBroker for Java.** Java で書かれているオブジェクト・リクエスト・ブローカー (ORB)。

# W

**Web.** ワールド・ワイド・ウェブ (WWW) を参照。

**Web ブラウザー (Web browser).** ワールド・ワイド・ウェブ (WWW) 上で配布される情報を形式設定し、表示するプログラム。

# 参照文献

この節で記載している資料は、現在の MQSeries 全製品で利用できます。

## プラットフォーム共通の MQSeries 資料

これらの資料の多くは、MQSeries「ファミリー」資料とも呼ばれ、MQSeries レベル 2 の全製品に適用されます。最新の MQSeries レベル 2 製品は、次のとおりです。

- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries (AT&T GIS UNIX 版) V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS V2.2.1.1
- MQSeries for (Compaq Tru64 UNIX 版) V5.1
- MQSeries for HP-UX V5.2
- MQSeries for Linux V5.2
- MQSeries (OS/2 Warp 版) V5.1
- MQSeries for OS/390 V5.2
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Sun Solaris V5.2
- MQSeries for Sun Solaris, Intel Platform Edition V5.1
- MQSeries for Tandem NonStop Kernel V2.2.0.1
- MQSeries for VSE/ESA V2.1.1
- MQSeries (Windows 拡張版) V2.0
- MQSeries (Windows 拡張版) V2.1
- MQSeries for Windows NT and Windows 2000 V5.2

プラットフォーム共通の MQSeries 資料は、次のとおりです。

- *MQSeries Brochure* (G511-1908)
- *An Introduction to Messaging and Queuing* (GC33-0805)
- *MQSeries 相互通信* (SC88-7775)
- *MQSeries* キュー・マネージャー・クラスター (SD88-7165)
- *MQSeries クライアント* (GC88-7495)
- *MQSeries システム管理の手引き* (SC88-7776)
- *MQSeries MQSC コマンド・リファレンス* (SC88-7315)
- *MQSeries イベント・モニター* (SC88-8705)
- *MQSeries プログラム式システム管理* (SC88-7562)
- *MQSeries 管理インターフェースのプログラミングの手引きおよび解説書* (SD88-7145)
- *MQSeries メッセージ* (GC88-7777)
- *MQSeries アプリケーション・プログラミング・ガイド* (SC88-7253)
- *MQSeries アプリケーション・プログラミング・リファレンス* (SC88-7354)
- *MQSeries Programming Interfaces Reference Summary* (SX33-6095)
- *MQSeries C++ の使用* (SC88-7778)
- *MQSeries Java の使用* (SD88-7163)
- *MQSeries アプリケーション・メッセージング・インターフェース* (SC88-8704)

## プラットフォーム固有の MQSeries 資料

各 MQSeries 製品には、MQSeries ファミリーの資料に加えて、 1 つ以上のプラットフォーム固有の資料があります。

**MQSeries for AIX V5.2**

*MQSeries for AIX インストール・ガイド*, GC88-7770

**MQSeries for AS/400 V5.2**

*MQSeries for AS/400 インストール・ガイド* (GD88-7309)

*MQSeries (AS/400 版) システム管理の手引き* (SD88-7310)

*MQSeries (AS/400 版) アプリケーション・プログラミング解説書 (ILE RPG)* (SD88-7311)

**MQSeries (AT&T GIS UNIX 版) V2.2**

*MQSeries (AT&T GIS UNIX 版) システム管理の手引き* (SC88-7646)

**MQSeries for Compaq (DIGITAL) OpenVMS V2.2.1.1**

*MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide* (GC33-1791)

**MQSeries for (Compaq Tru64 UNIX 版) V5.1**

*MQSeries (Compaq Tru64 UNIX 版) インストールの手引き* (GC88-8625)

**MQSeries for HP-UX V5.2**

*MQSeries for HP-UX インストール・ガイド* (GC88-7772)

**MQSeries for Linux V5.2**

*MQSeries for Linux インストール・ガイド* (GD88-7354)

**MQSeries (OS/2 Warp 版) V5.1**

*MQSeries (OS/2 Warp 版) インストールの手引き* (GC88-7771)

**MQSeries for OS/390 V5.2**

*MQSeries for OS/390 概説および計画ガイド* (GC88-8615)

*MQSeries for OS/390 システム・セットアップ・ガイド* (SC88-8616)

*MQSeries for OS/390 システム管理ガイド* (SC88-8617)

*MQSeries for OS/390 問題判別ガイド* (GC88-8846)

*MQSeries for OS/390 メッセージおよびコード* (GC88-8845)

*MQSeries for OS/390 Licensed Program Specifications* (GC34-5893)

*MQSeries for OS/390 Program Directory*

**MQSeries link for R/3 V1.2 ユーザーズ・ガイド**

*MQSeries link for R/3 ユーザーズ・ガイド* (GD88-7041)

**MQSeries for SINIX and DC/OSx V2.2**

*MQSeries for SINIX and DC/OSx System Management Guide* (GC33-1768)

**MQSeries for Sun Solaris V5.2**

*MQSeries for Sun Solaris インストール・ガイド* (GC88-7773)

**MQSeries for Sun Solaris, Intel Platform Edition V5.1**

*MQSeries for Sun Solaris, Intel Platform Edition インストール・ガイド* (GC88-8749)

**MQSeries for Tandem NonStop Kernel V2.2.0.1**

*MQSeries for Tandem NonStop Kernel System Management Guide* (GC33-1893)

**MQSeries for VSE/ESA V2.1.1**

*MQSeries for VSE/ESA™ Licensed Program Specifications* (GC34-5365)

*MQSeries for VSE/ESA System Management Guide* (GC34-5364)

**MQSeries (Windows 拡張版) V2.0**

*MQSeries (Windows 拡張版) 使用者の手引き* (GC88-7649)

**MQSeries (Windows 拡張版) V2.1**

*MQSeries (Windows 拡張版) 使用者の手引き* (GC88-7208)

**MQSeries for Windows NT and Windows 2000 V5.2**

*MQSeries for Windows NT and Windows 2000 インストール・ガイド* (GD88-7162)

*MQSeries (Windows NT 版) コンポーネント・オブジェクト・モデル・インターフェースの使用* (SD88-7161)

*MQSeries LotusScript Extension* (SD88-7146)

## ソフトコピー・ブック

MQSeries 資料のほとんどは、ハードコピーとソフトコピーの両方の形式で提供されます。

## HTML 形式

以下の MQSeries 製品には HTML 形式の MQSeries 資料が付属しています。

- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries for (Compaq Tru64 UNIX 版) V5.1
- MQSeries for HP-UX V5.2
- MQSeries for Linux V5.2

- MQSeries (OS/2 Warp 版) V5.1
- MQSeries for OS/390 V5.2
- MQSeries for Sun Solaris V5.2
- MQSeries for Sun Solaris, Intel Platform Edition V5.1
- MQSeries for Windows NT and Windows 2000 V5.2 (コンパイル済み HTML)
- MQSeries link for R/3 V1.2

MQSeries に関する HTML 形式の資料は、以下に示す MQSeries 製品ファミリーのホーム・ページにもあります。

```
http://www.ibm.com/software/mqseries/
```

## PDF

PDF ファイルは、Adobe Acrobat Reader を使用して、表示または印刷することができます。

Adobe Acrobat Reader を入手する必要がある場合、または Acrobat Reader がサポートされるプラットフォームに関する最新情報が必要な場合は、以下の Adobe Systems Inc. の Web サイトにアクセスしてください。

```
http://www.adobe.com/
```

以下の MQSeries 製品には、PDF 版の MQSeries 資料が付属しています。

- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries for (Compaq Tru64 UNIX 版) V5.1
- MQSeries for HP-UX V5.2
- MQSeries for Linux V5.2
- MQSeries (OS/2 Warp 版) V5.1
- MQSeries for OS/390 V5.2
- MQSeries for Sun Solaris V5.2
- MQSeries for Sun Solaris, Intel Platform Edition V5.1
- MQSeries for Windows NT and Windows 2000 V5.2
- MQSeries link for R/3 V1.2

MQSeries に関する PDF 版の資料は、以下に示す MQSeries 製品ファミリーのホーム・ページにもあります。

```
http://www.ibm.com/software/mqseries/
```

## BookManager® 形式

MQSeries ライブラリーは、*Transaction Processing and Data* コレクション・キット (SK2T-0730) を含むさまざまなオンライン・ライブラリー・コレクション・キットで、 IBM BookManager 形式で提供されています。以下の IBM ライセンス・プログラムを使用して、ソフトコピー資料を表示できます。

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

## PostScript 形式

MQSeries ライブラリーは、多くの MQSeries バージョン 2 製品で PostScript (.PS) 形式で提供されています。 PostScript 形式の資料は、PostScript プリンターで印刷したり、適切なビューアーを使用して表示できます。

## Windows ヘルプ形式

*MQSeries (Windows 拡張版) 使用者の手引き* が、MQSeries (Windows 拡張版) V2.0 および MQSeries (Windows 拡張版) V2.1 で、 Windows ヘルプ形式で提供されています。

---

# インターネットで利用できる MQSeries 情報

MQSeries 製品ファミリーのホーム・ページの URL は以下のとおりです。

```
http://www.ibm.com/software/mqseries/
```

この URL からのリンクによって、次のようなサービスを利用できます。

- MQSeries 製品ファミリーについての最新情報を入手する。
- HTML 形式の MQSeries 資料 (英文のみ) にアクセスする。
- MQSeries SupportPac をダウンロードする。

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]



## [カ行]

# [サ行]

## ［タ行］



## ［ナ行］



## ［ハ行］



## ［マ行］



## ［ヤ行］

## ［ラ行］



## ［ワ行］



## ［数字］



## A



## B



## C



## D



## E



## F



## H



## I



## J

# L



# M

## N



## O



## P



## Q



## R



## S



## T



## U



## V



## W



## X

Printed in Japan

SD88-7163-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12