Sun Java™ System

Message Queue 3.6 SP3 技術の概要

2005Q4

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819-3564

目 次

図目次

表目次

はじめに

本書『Sun Java™ System Message Queue 3.6 SP3 2005Q4 技術の概要』では、Message Queue メッセージングサービスのテクノロジ、概念、アーキテクチャー、機能、および能力を紹介します。

したがって、『Message Queue 技術の概要』は、Message Queue マニュアルセット内のほかのマニュアルの基盤となります。Message Queue マニュアルセットのほかのマニュアルをお読みになる前に、本書をお読みください。

ここでは、次の節について説明します。

対象読者

このマニュアルは、Message Queue 製品を使用する予定の、または製品のテクノロジ、概念、アーキテクチャー、機能、および能力について理解することを望む、管理者やアプリケーション開発者その他を対象に作成されています。

管理者は、Message Queue メッセージングサービスの設定と管理を行います。このマニュアルを読むにあたって、メッセージングシステムの知識や理解は必要としません。

アプリケーション開発者は、Message Queue サービスを使用してほかのクライアントアプリケーションとメッセージを交換する、Message Queue クライアントアプリケーションを記述します。このマニュアルを読むにあたり、Message Queue サービスによって実装される Java Message Service (JMS) 仕様に関する知識は必要としません。

お読みになる前に

このマニュアルを読むための前提条件はありません。Message Queue の開発者ガイドおよび管理ガイドを読む前に、このマニュアルを読んで Message Queue の基本的な概念を理解してください。

内容の紹介

このマニュアルは、最初から順番に読み進むように構成されており、各章の内容は、それよりも前の章で説明される情報に基づいています。次の表に、各章の内容について簡単に説明します。

表 1　マニュアルの内容と構成

章	説明
第 1 章「メッセージングシステム : 概論」	メッセージングミドルウェアテクノロジ、JMS 標準について述べ、Message Queue サービスの標準の実装について説明します。
第 2 章「クライアントプログラミングモデル」	JMS プログラミングモデルについて述べ、Message Queue クライアントランタイムを使用して JMS クライアントを作成する方法について説明します。C++ クライアントと SOAP メッセージのトランスポートのランタイムサポートについて説明します。

表 1　マニュアルの内容と構成 (続き)

章	説明
第 3 章「Message Queue サービス」	管理タスクおよびツールについて述べ、コネクション、ルーティング、持続性、セキュリティー、および監視を設定するために使用されるブローカサービスについて説明します。
第 4 章「ブローカクラスタ」	Message Queue ブローカクラスタのアーキテクチャーと使用方法について説明します。
第 5 章「Message Queue と J2EE」	J2EE プラットフォーム環境で実装される派生 JMS について説明します。
付録 A「Message Queue オプションの JMS 機能の実装」	Message Queue 製品で JMS オプション項目を処理する方法を説明します。
付録 B「Message Queue の機能」	Message Queue 機能を一覧表示し、これらの実装に必要な手順をまとめ、追加情報の参照先を示します。
用語集	Message Queue の使用時に知っておくと便利な用語や概念について説明します。

本書で使用する表記規則

ここでは、このマニュアルで使用されている表記規則について説明します。

テキストの表記規則

表 2　マニュアルの表記規則

書式	説明
斜体	可変部分で使用されます。斜体で表記された項目や値は適宜置き換える必要があります。説明の対象となる語句や項目に対しても使用されます。
モノスペース	コード例、コマンド行に入力するコマンド、ディレクトリ、ファイルまたはパス名、エラーメッセージテキスト、クラス名、メソッド名 (シグネチャの全要素を含む)、パッケージ名、予約語、および URL を表します。
[]	コマンド行の構文ステートメントのオプションの値を示します。
すべて大文字	ファイルシステムタイプ (GIF、TXT、HTML など)、環境変数 (IMQ_HOME)、または頭文字 (JMS、JSP) を表します。

表 2　マニュアルの表記規則 (続き)

書式	説明
キー + キー	複数のキーストロークはプラス記号で結合します。Ctrl+A は、両方のキーを同時に押すことを表します。
キー - キー	連続するキーストロークはハイフンで結合します。Esc-S は、Esc キーを押してから離し、次に S キーを押すことを表します。

ディレクトリ変数の表記規則

Message Queue では 3 種類のディレクトリ変数が使用されますが、その設定方法は、プラットフォームによって異なります。表 3 では、これらの変数について説明し、Solaris™、Windows、および Linux の各プラットフォームでの使用方法についても説明します。

表 3　Message Queue ディレクトリ変数

変数	説明
IMQ_HOME	この変数は通常、Message Queue マニュアル内で Message Queue 基本ディレクトリ (ルートインストールディレクトリ) を参照するのに使用されます。 • Solaris の場合、ルート Message Queue インストールディレクトリは存在しません。そのため、IMQ_HOME は、Solaris 上のファイルの場所を参照するために Message Queue マニュアルで使用されることはありません。 • Solaris の Sun Java System Application Server の場合、ルート Message Queue インストールディレクトリは Application Server 基本ディレクトリの下の /imq です。 • Windows では、ルート Message Queue インストールディレクトリは、Message Queue インストーラによって設定されます。デフォルトでは、C:¥Program Files¥Sun¥MessageQueue3 です。 • Windows では、Sun Java System Application Server の場合、ルート Message Queue インストールディレクトリは Application Server 基本ディレクトリの下の /imq です。 • Linux の場合、ルート Message Queue インストールディレクトリは存在しません。そのため、IMQ_HOME は、Linux 上のファイルの場所を参照するために Message Queue マニュアルで使用されることはありません。

表 3　Message Queue ディレクトリ変数 (続き)

変数	説明
IMQ_VARHOME	Message Queue の一時的な、または動的に作成された設定ファイルやデータファイルが格納されている、/var ディレクトリです。任意のディレクトリを指す環境変数として設定されます。 • Solaris の場合、IMQ_VARHOME のデフォルト値は /var/imq ディレクトリです。 • Solaris の場合、Sun Java System Application Server の Evaluation Edition では、IMQ_VARHOME のデフォルト値は IMQ_HOME/var ディレクトリです。 • Windows の場合、IMQ_VARHOME のデフォルト値は IMQ_HOME/var ディレクトリです。 • Windows の場合、Sun Java System Application Server の IMQ_VARHOME のデフォルト値は IMQ_HOME/var ディレクトリです。 • Linux の場合、IMQ_VARHOME のデフォルト値は /var/opt/imq ディレクトリです。

表 3　Message Queue ディレクトリ変数 (続き)

変数	説明
IMQ_JAVAHOME	Message Queue 実行可能ファイルに必要な、Java™ ランタイム (JRE) の場所を指す環境変数です。 • Solaris では、IMQ_JAVAHOME が次の順序で Java ランタイムを検索しますが、ユーザーは必要な JRE の配置場所であればどこにでもオプションで値を設定できます。 Solaris 8 または 9: /usr/jdk/entsys-j2se /usr/jdk/jdk1.5.* /usr/jdk/j2sdk1.5.* /usr/j2se Solaris 10: /usr/jdk/entsys-j2se /usr/java /usr/j2se • Linux では、Message Queue が次の順序で Java ランタイムを検索しますが、ユーザーは、JRE の配置場所であればどこにでも IMQ_JAVAHOME の値をオプションで設定できます。 /usr/jdk/entsys-j2se /usr/java/jre1.5.* /usr/java/jdk1.5.* /usr/java/jre1.4.2* /usr/java/j2sdk1.4.2* • Windows では、IMQ_JAVAHOME がデフォルトで IMQ_HOME/jre に設定されますが、ユーザーは、必要な JRE の配置場所であればどこにでもその値をオプションで設定できます。

このマニュアルでは、IMQ_HOME、IMQ_VARHOME、および IMQ_JAVAHOME は、プラットフォーム固有の環境変数の表記法や構文 (たとえば、UNIX であれば $IMQ_HOME) なしで示されています。パス名には、通常、UNIX のディレクトリ区切り文字の表記法 (/) が使用されています。

関連資料

このガイド以外にも、Message Queue には追加のマニュアルが用意されています。

Message Queue マニュアルセット

Message Queue マニュアルセットは、次のマニュアルで構成されています。各マニュアルを通常使用する順番で、表 4 に一覧表示します。

表 4　Message Queue マニュアルセット

マニュアル	対象読者	説明
『Message Queue Installation Guide』	開発者および管理者	Message Queue ソフトウェアの Solaris、Linux、Windows の各プラットフォームへのインストール方法を説明しています。
『Message Queue リリースノート』	開発者および管理者	新機能、制限、既知のバグ、および技術的な注意点を収録しています。
『Message Queue 技術の概要』	開発者、管理者、および設計者	Message Queue 製品のテクノロジ、概念、アーキテクチャー、機能、および能力を紹介しています。
『Message Queue 管理ガイド』	管理者。開発者にも推奨	Message Queue 管理ツールを使用した管理タスクの実行に必要な基本情報を提供しています。
『Message Queue Developer's Guide for Java Clients』	開発者	JMS 仕様と SOAP/JAXM 仕様の Message Queue 実装を使用する Java クライアントプログラムの開発者向けにクイックスタートチュートリアルとプログラミング情報を提供しています。
『Message Queue Developer's Guide for C Clients』	開発者	Message Queue メッセージサービスへの C インタフェース (C-API) を使用する C クライアントプログラムの開発者向けにプログラミングマニュアルとリファレンスマニュアルを提供しています。

オンラインヘルプ

Message Queue には、Message Queue メッセージサービス管理タスクを実行するためのコマンド行ユーティリティーが含まれています。各ユーティリティーのオンラインヘルプにアクセスするには、『Message Queue 管理ガイド』を参照してください。

また、Message Queue には、グラフィカルユーザーインタフェース (GUI) 管理ツールである管理コンソール (Administration Console) (`imqadmin`) も含まれています。管理コンソールには、操作状況に合わせて表示できるオンラインヘルプが用意されています。

JavaDoc

JavaDoc 形式の Message Queue Java クライアント API (JMS API を含む) マニュアルは、次の場所にあります。

プラットフォーム	場所
Solaris	`/usr/share/javadoc/imq/index.html`
Linux	`/opt/imq/javadoc/index.html/`
Windows	`IMQ_HOME/javadoc/index.html`

このマニュアルは、Netscape または Internet Explorer などの HTML ブラウザで表示できます。このマニュアルには、標準の JMS API マニュアルおよび Message Queue 管理対象オブジェクト用の Message Queue 固有の API が含まれており (『Message Queue Developer's Guide for Java Clients』の第 3 章を参照)、メッセージングアプリケーションの開発者にとって有用です。

クライアントアプリケーションの例

クライアントアプリケーションのサンプルコードを示す多数のアプリケーション例が、オペレーティングシステムに応じて該当するディレクトリに含まれています (『Message Queue 管理ガイド』を参照)。

このディレクトリと各サブディレクトリにある README ファイルを参照してください。

Java Message Service (JMS) 仕様書

JMS 仕様書は、次のサイトにあります。

http://java.sun.com/products/jms/docs.html

この仕様書には、サンプルのクライアントコードも掲載されています。

関連するサードパーティーの Web サイトのリファレンス

このマニュアルでは、サードパーティーの URL が参考として示されているほか、追加の関連情報も提供されています。

注 サンマイクロシステムズ株式会社は、このマニュアルに記載されたサードパーティーの Web サイトの可用性については一切責任を負いません。また、このようなサイトまたはリソースで提供されているコンテンツ、広告、製品、そのほかのマテリアルを支持するわけではなく、それらに対する責任も一切負いません。サンマイクロシステムズ株式会社は、このようなサイトまたはリソースで提供されているコンテンツ、商品、またはサービスを使用もしくは信用したことで、実際に引き起こされた、または引き起こされたと考えられる損害や損失についても一切の責任は負いません。

コメントの送付先

Sun では、マニュアルの改善のために、皆様からのコメントおよび提案をお待ちしております。

コメントを送るには、http://docs.sun.com にアクセスして「コメントの送信」をクリックしてください。オンラインフォームにマニュアルのタイトルと Part No. をご記入ください。Part No. は、マニュアルのタイトルページか先頭に記述されている 7 桁または 9 桁の番号です。

Message Queue に固有の質問と問題については、
http://swforum.sun.com/jive/forum.jspa?forumID=24 も参照してください。

第 1 章

メッセージングシステム : 概論

Sun Java™ System Message Queue は、Java Message Service (JMS) 標準を実装し、拡張するメッセージングミドルウェア製品です。このことを十分に理解している場合は、31 ページの「Message Queue: 要素と機能」へ進んでください。そうでない場合は、最初からお読みください。

この章では、Message Queue などの製品の基盤をなすメッセージングテクノロジについて述べ、このテクノロジを標準化した JMS 仕様を、Message Queue がどのように実装し拡張するかについて説明します。次のトピックが含まれます。

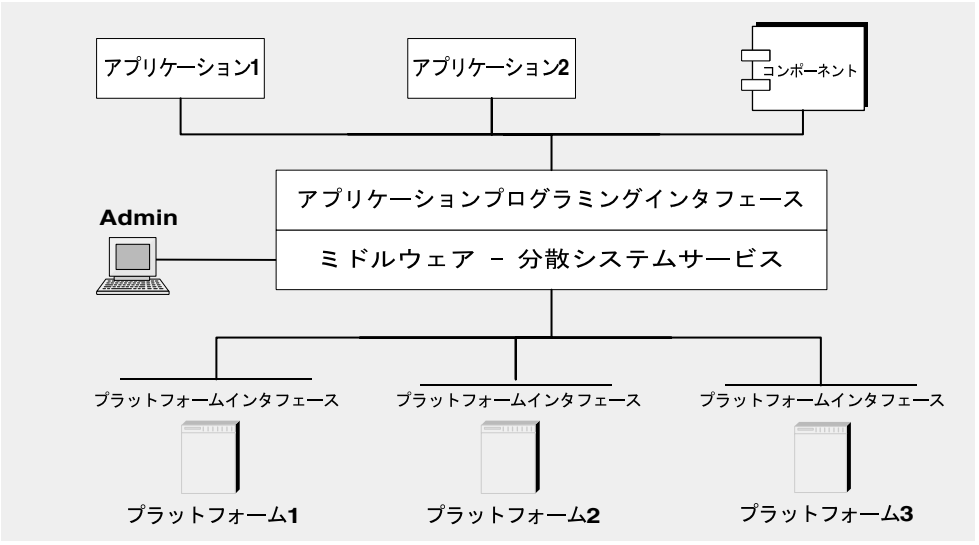
メッセージ指向ミドルウェア (MOM)

ビジネス、制度、およびテクノロジは絶えず変化しているため、そこで使用されるソフトウェアシステムには、このような変化に順応できる能力が要求されます。ビジネスで、合併、サービスの追加、または使用可能なサービスの拡張を行ったあとで、情報システムを作り直すだけの余裕はほとんど残っていません。この最も重要な時点でこそ、新しいコンポーネントの統合または既存のコンポーネントの拡張を、できるだけ効率的に行う必要があります。異種コンポーネントを最も簡単に統合する方法は、同種の要素として作り直すことではなく、異種であっても相互に通信できるようにする層を用意することです。ミドルウェアと呼ばれるこの層によって、個別に開発され、

異なるネットワークプラットフォームで稼働するソフトウェアコンポーネント (アプリケーション、EJB (Enterprise Java Bean)、サーブレットなどのコンポーネント) が、相互に対話できるようになります。この対話が可能になった時点から、ネットワークはコンピュータとして機能できます。

図 1-1 に示すように、ミドルウェアは、概念上、アプリケーション層とプラットフォーム層 (オペレーティングシステムと基礎となるネットワークサービス) の間に位置します。

図 1-1 ミドルウェア



異なるネットワークノードに分散したアプリケーションはアプリケーションインタフェースを使用して通信し、ほかのアプリケーションをホストしているオペレーティング環境の詳細を意識する必要も、これらのアプリケーションに接続するサービスを使用する必要もありません。さらに、相互接続したアプリケーションによるこの新しい仮想的なシステムは、管理インタフェースを備えることにより、信頼性と安全性を確保します。そのパフォーマンスは測定および調整可能であり、機能を損ねずにシステムを拡張できます。

ミドルウェアは、次のカテゴリに分類できます。

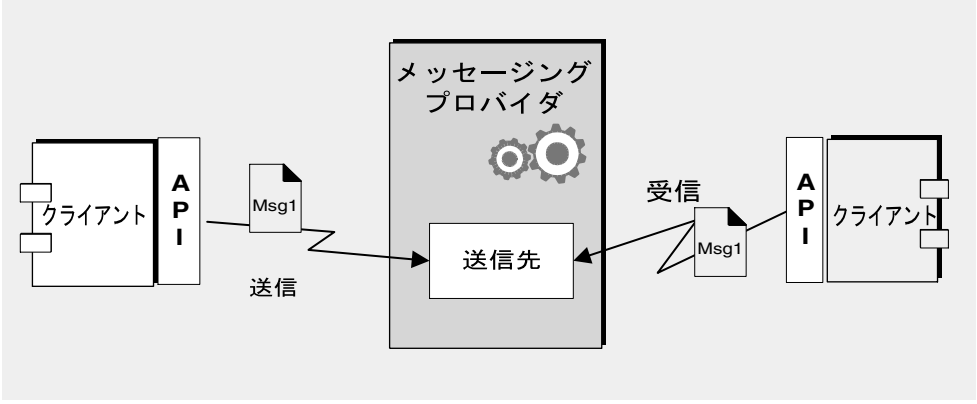
- リモートプロシージャー呼び出し (RPC) ベースのミドルウェア。あるアプリケーション内のプロシージャーが、ローカルでの呼び出しのように、リモートのアプリケーション内のプロシージャーを呼び出せるようにします。このミドルウェアにはリンクメカニズムが実装されており、リモートプロシージャーを検索して、呼び出し側から透過的にこれらのプロシージャーを利用できるようにしています。従来、この種のミドルウェアは、プロシージャーベースのプログラムを扱っていましたが、現在、オブジェクトベースのコンポーネントにも対応しています。

- オブジェクトリクエストブローカ (ORB) ベースのミドルウェア。アプリケーションのオブジェクトを、異種ネットワーク間で分散し共有できるようにします。
- メッセージ指向ミドルウェア (MOM) ベースのミドルウェア。分散型アプリケーションが、メッセージの送受信によって、データの通信および交換を行えるようにします。

これらのすべてのモデルで、あるソフトウェアコンポーネントが別のコンポーネントの動作にネットワークを介して影響を及ぼすことができます。RPC ベースと ORB ベースのミドルウェアが、密結合コンポーネントのシステムを作成する一方、MOM ベースのシステムが、疎結合コンポーネントに対応しているという点で、これらには違いがあります。RPC ベースまたは ORB ベースのシステムでは、あるプロシージャーが別のプロシージャーを呼び出すと、呼び出されたプロシージャーが返されるまで、別の処理を行えなくなります。前述のように、これらのモデルでは、ミドルウェアは部分的にスーパーリンカーとして機能し、呼び出されたプロシージャーをネットワーク上で検索し、ネットワークサービスを使用して、関数やメソッドのパラメータをプロシージャーに渡し、結果を返します。

MOM ベースのシステムは、図 1-2 に示すように、メッセージの非同期的交換を通じて、通信を行えるようにします。

図 1-2　MOM ベースのシステム



メッセージ指向ミドルウェアは、メッセージングプロバイダを利用して、メッセージング操作を仲介します。MOM システムの基本要素は、クライアント、メッセージ、および MOM プロバイダです。MOM プロバイダには API と管理ツールが含まれます。MOM プロバイダで、メッセージのルーティングおよび配信に使用されるアーキテクチャーには、集中メッセージサーバーを使用したものと、ルーティングおよび配信機能を各クライアントマシンに分散したものがあります。この 2 つの方式を併用した MOM 製品もあります。

MOM システムを使用した場合、クライアントは API 呼び出しを作成して、プロバイダが管理する送信先にメッセージを送信します。この呼び出しによって、プロバイダのサービスが呼び出され、メッセージのルーティングと配信を行います。クライアントはメッセージを送った後、受信側クライアントがメッセージを取得するまでメッセージがプロバイダに保持されるため、他の作業を続行することができます。メッセージベースのモデルとプロバイダの仲介とを組み合わせることにより、疎結合コンポーネントのシステムを作成できます。このようなシステムは、個々のコンポーネントまたはコネクションに障害が生じた場合でも確実に機能を続行でき、停止時間がありません。

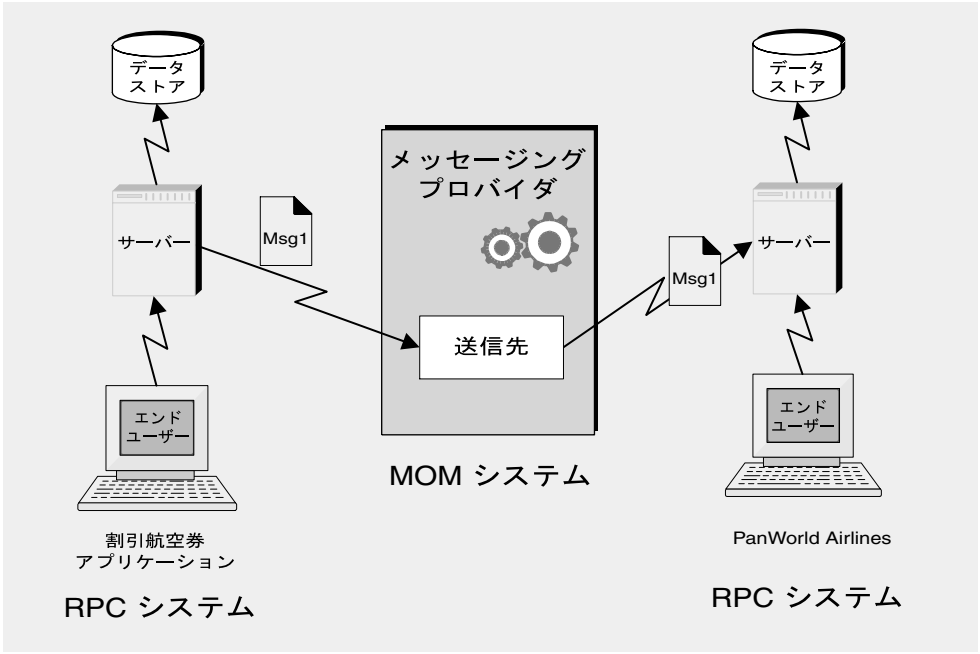
メッセージングプロバイダにクライアント間のメッセージングを仲介させた場合、管理インタフェースを追加することによって、パフォーマンスの監視と調整を行えるという別のメリットが得られます。したがって、クライアントアプリケーションは、メッセージの送信、受信、および処理の問題を除く実質的にすべての問題から解放されます。相互運用性、信頼性、セキュリティー、スケーラビリティー、パフォーマンスなどの問題の解決は、MOM システムを実装するコードと管理者に委ねられます。

これまで、メッセージ指向ミドルウェアを使用した分散コンポーネントの接続によるメリットについて述べてきました。同時にデメリットもあります。そのひとつは、疎結合自体から生じるものです。RPC システムでは、呼び出し側の関数は、呼び出された関数が実行中のタスクを終了するまで戻りません。非同期システムでは、呼び出し側のクライアントは、この作業の処理に必要なリソースが欠乏し、呼び出されたコンポーネントに障害が発生するまで、受信側に負荷をかけ続けることがあります。もちろん、パフォーマンスの監視とメッセージフローの調整によって、このような状況を最小限にとどめたり、回避したりすることができますが、RPC システムでは不要な作業になります。それぞれのシステムのメリットとデメリットを理解することが重要です。それぞれのシステムは、異なるタスクに適しています。場合によっては、必要とする動作を得るために、2 種類のシステムを組み合わせる必要が生じることもあります。

図 1-3 では、MOM システムが、2 つの RPC ベースのシステム間の通信を可能にしている様子を示しています。図の左側は、パフォーマンスの改善のために、クライアント、サーバー、およびデータストアのコンポーネントを別々のネットワークノードに分散しているアプリケーションを示しています。これは、割引航空券予約システムです。エンドユーザーがこのサービスの利用料を支払うことにより、このサービスが指定された行先と時間に対して利用可能な最低運賃を見つけます。データストアは、登録ユーザーに関する情報と、このプログラムに参加している航空会社に関する情報を格納しています。サーバーのロジックは、ユーザーの要求に基づいて、参加している航空会社の価格を照会し、情報を調べて、最も安い 3 社の料金をユーザーに提示します。図の右側には、参加している航空会社のいずれか 1 社の航空券 / 予約システムを表す、RPC ベースのシステムが示されています。図の右側は、割引システムに接続している航空会社の数だけ複製されます。データストアは、このような航空会社ごとに、利用可能なフライトに関する情報 (座席、フライト時間、価格など) を保存します。サーバーコンポーネントは、エンドユーザーが入力したデータに応じて情報を更新します。航空会社のサーバーも MOM サービスを利用して、割引予約システムから情報

の要求を受け取り、座席および価格情報を返します。顧客が PanWorld 社の割引航空券を購入することにした場合、このシステムのサーバーコンポーネントは、データストア内の情報を更新してから、請求者に対してチケットを発券するか、チケットを発券するように割引サービスへメッセージを送信します。

図 1-3　RPC システムと MOM システムとの組み合わせ



この例には、RPC システムと MOM システムとの相違点がいくつか示されています。分散コンポーネントの結合方法における相違点については、すでに述べました。そのほかにも、RPC システムは多くの場合、クライアントがエンドユーザーであるクライアントおよびサーバーコンポーネントの分散と結合に使用されますが、MOM システムでは、クライアントが、メッセージングによってのみ相互運用可能な異種ソフトウェアコンポーネントであるという点も異なります。

MOM システムでのより重要な問題は、MOM が有標製品として実装されるということから生じます。SuperMOM-X に依拠している企業が、SuperMOM-Y を使用している企業を合併すると、どうなるでしょうか。この問題を解決するために、標準的なメッセージングインタフェースが必要になります。SuperMOM-X と SuperMOM-Y の両方にこのインタフェースが実装されていれば、一方のシステムで稼働するように開発されたアプリケーションは、もう一方のシステムでも稼働できます。このようなインタフェースは、習得が容易であると同時に、高度なメッセージングアプリケーションをサポートできるだけの十分な機能を備えているべきです。1998 年に発表された

Java Message Service (JMS) 仕様は、まさしくこのことを目的としています。次の節では、JMS の基本機能について説明し、既存の有標 MOM 製品の共通要素を取り入れ、異種の受け入れと今後の成長に対応するために、この標準がどのように開発されたかについて説明します。

MOM 標準としての JMS

Java Messaging Service 仕様は、本来、Java アプリケーションから既存の MOM システムにアクセスできるようにするために開発されました。この仕様は発表以来、既存の多くの MOM ベンダーによって採用され、それ自体で非同期メッセージングシステムとして実装されてきました。

JMS 仕様の設計者は、その作成時に、既存のメッセージングシステムの本質的要素を取り入れようと考えました。この本質的要素は次のとおりです。

- メッセージのルーティングと配信を行うメッセージングプロバイダの概念
- ポイントツーポイントメッセージングやパブリッシュ / サブスクライブメッセージングなど、個別のメッセージングパターンまたはドメイン
- 同期および非同期メッセージ受信用の機構
- 信頼性の高いメッセージ配信のサポート
- ストリーム、テキスト、バイトなどの、共通のメッセージ形式

ベンダーは、JMS インタフェースを実装するライブラリ、メッセージのルーティングおよび配信の機能、およびメッセージングサービスの管理、監視、調整を行う管理ツールから構成される JMS プロバイダを用意して、JMS 仕様を実装しています。ルーティングおよび配信機能は、集中メッセージングサーバーまたはブローカが実行する場合も、各クライアントのランタイムに組み込んだ機能を通じて実装する場合もあります。

同時に、JMS プロバイダはさまざまな役割を果たすことができ、スタンドアロン製品として作成することも、より大規模な分散ランタイムシステム内の埋め込みコンポーネントとして作成することもできます。スタンドアロン製品の場合、エンタープライズアプリケーション統合システムの基幹を定めるために使用できます。アプリケーションサーバーに埋め込む場合は、内部コンポーネントメッセージングをサポートできます。たとえば J2EE は、JMS プロバイダを使用してメッセージ駆動型 Beans を実装し、EJB コンポーネントによるメッセージの送受信ができるようにしています。

既存のシステムのすべての機能を含んだ標準を作成していたなら、習得が難しく実装の困難なシステムになっていたでしょう。JMS ではそのようにせず、メッセージングの概念および機能の共通項を定義しました。この結果、習得が容易で、JMS プロバイダ間での JMS アプリケーションの移植性を最大限にした標準となりました。JMS が API 標準であり、プロトコル標準ではないことに留意することが重要です。あるベンダーから別のベンダーに JMS クライアントを移行するのは簡単です。ただし、JMS ベンダーが異なると、一般的に、直接互いに通信することはできません。

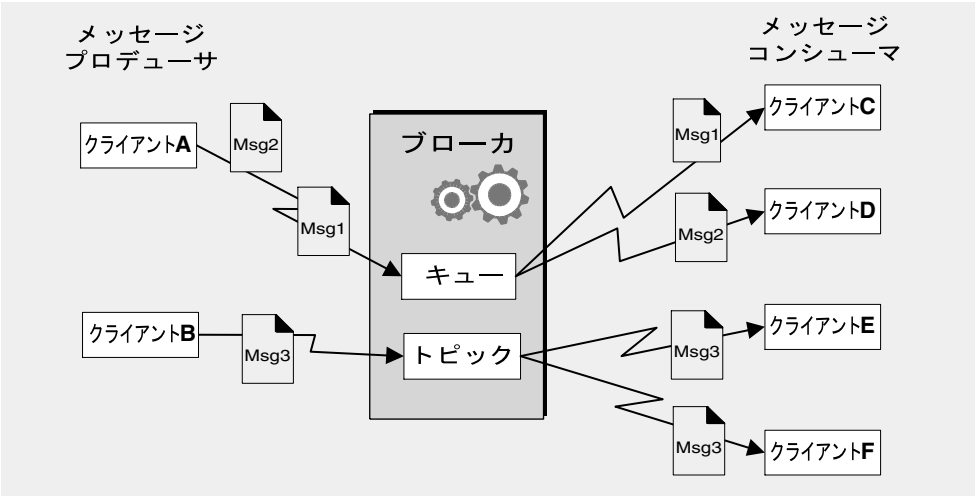
次の節では、JMS 仕様で規定された、基本オブジェクトとメッセージングパターンについて説明します。

JMS メッセージングオブジェクトおよびパターン

メッセージの送受信を行うために、JMS クライアントはまず、しばしばメッセージブローカとして実装される JMS プロバイダに接続する必要があります。接続すると、クライアントとブローカ間の通信チャネルが開かれます。次に、クライアントはメッセージの作成、プロデュース、およびコンシュームを行うためにセッションを設定する必要があります。このセッションは、クライアントとブローカ間の特定のやり取りを定めるメッセージのストリームと見なすことができます。クライアント自体は、メッセージプロデューサまたはメッセージコンシューマ、あるいはその両方になります。メッセージプロデューサは、ブローカが管理している送信先に、メッセージを送信します。メッセージコンシューマは、その送信先にアクセスして、メッセージをコンシュームします。メッセージには、ヘッダー、オプションのプロパティー、および本体が含まれます。本体にはデータが格納されます。ヘッダーには、メッセージのルーティングおよび管理にブローカで必要とされる情報が含まれます。プロパティーは、クライアントアプリケーションまたはプロバイダが、メッセージの処理でのそれぞれのニーズを満たすように定義できます。コネクション、セッション、送信先、メッセージ、プロデューサ、およびコンシューマは、JMS アプリケーションを構成する基本オブジェクトです。

これらの基本オブジェクトを使用して、クライアントアプリケーションは、2 つのメッセージングパターン (ドメイン) でメッセージを送受信できます。図 1-4 に、これらのパターンを示します。

図 1-4　JMS メッセージングパターン



クライアント A および B はメッセージプロデューサであり、2 種類の異なる送信先を経由して、クライアント C、D、および E にメッセージを送信しています。

- クライアント A、C、D 間のメッセージングは、ポイントツーポイントパターンを表しています。このパターンを使用した場合、クライアントがキュー送信先へメッセージを送信し、ただ 1 つの受信側がその送信先からメッセージを取得できます。この送信先にアクセスするほかの受信側は、そのメッセージを取得できません。
- クライアント B、E、F 間のメッセージングは、パブリッシャー / サブスクライブパターンを表しています。このブロードキャストパターンを使用した場合、クライアントがトピック送信先にメッセージを送信し、任意の数のコンシューミングサブスクライバがその送信先からメッセージを取得できます。各サブスクライバは、メッセージのコピーをそれぞれ取得します。

どちらのドメインのメッセージコンシューマも、メッセージの取得を同期または非同期にするかを選択できます。同期コンシューマは、メッセージを取得するための明示的な呼び出しを作成します。非同期コンシューマは、保留中のメッセージを渡すために呼び出されるコールバックメソッドを指定します。コンシューマは、受信メッセージに対して選択基準を指定することによって、メッセージをフィルタで除外することもできます。

管理対象オブジェクト

JMS 仕様は、あらゆる可能性を網羅しようとせずに、既存の MOM システムの多くの要素を結合した標準を作成しました。むしろ、異種および将来の成長に対応できる拡張可能な方式を確立しようとしました。JMS では、多くのメッセージング要素の定義と実装を、個々のプロバイダに任せています。これらの要素には、ロードバランス、標準エラーメッセージ、管理 API、セキュリティー、基本的なワイヤプロトコル、およびメッセージストアがあります。次の節の「Message Queue: 要素と機能」では、Message Queue で、これらの要素の多くがどのように実装され、JMS 仕様がどのように拡張されているかについて説明します。

JMS で完全には規定されていない 2 つのメッセージング要素が、コネクションファクトリと送信先です。これらは JMS プログラミングモデルで不可欠な要素ですが、プロバイダによるこれらのオブジェクトの定義および管理方法には、非常に多くの相違点および予想される相違点があったため、共通の定義の作成は不可能であり、また望ましいことでもありませんでした。したがって、これらの 2 つのオブジェクトは、プログラムによって作成されるのではなく、通常、管理ツールを使用して、作成および設定されます。次にこれらのオブジェクトはオブジェクトストアに保存され、JNDI 検索を通じて JMS クライアントからアクセスされます。

- コネクションファクトリ管理対象オブジェクトは、クライアントからブローカへのコネクションを生成するために使用されます。コネクション処理、クライアントの識別、メッセージヘッダーのオーバーライド、信頼性、フロー制御など、メッセージング動作のある側面を制御するプロバイダ固有の情報をカプセル化します。特定のコネクションファクトリから生成したすべてのコネクションは、そのファクトリに対して設定された動作を行います。
- 送信先管理対象オブジェクトは、ブローカ上の物理的な送信先を参照するために使用されます。プロバイダ固有の命名 (アドレス指定構文) 規則をカプセル化し、送信先を使用するメッセージングドメインが、キューかトピックかを指定します。

JMS クライアントは、管理対象オブジェクトの検索には必要ありません。これらのオブジェクトはプログラムによって作成された後、ブローカのメモリーに保存されます。プロトタイピングを迅速に行うには、これらのオブジェクトをプログラムによって作成する方法が最も簡単です。ただし、本稼働環境に配備する場合は、中央のリポジトリから管理対象オブジェクトを検索した方が、メッセージング動作の制御または管理をより簡単に行えます。

- コネクションファクトリオブジェクトに対して管理オブジェクトを使用することによって、管理者は、これらのオブジェクトを設定し直して、メッセージングパフォーマンスを調整できます。コーディングし直さずに、パフォーマンスを改善できます。
- 物理的な送信先に対して管理オブジェクトを使用した場合、管理者は、設定済みのオブジェクトにクライアントをアクセスさせることにより、ブローカ上で送信先が拡大しないよう制御できます。

- 管理対象オブジェクトにより、開発者は、プロバイダ固有の実装詳細を意識せずに済みます。あるプロバイダ用に開発するコードが、ほとんどあるいはまったく変更せずに、ほかのプロバイダへ移植できるようになります。

管理対象オブジェクトの使用により、図 1-5 に示す基本的な JMS アプリケーションの図式に最後の要素が加わります。

図 1-5 JMS アプリケーションの基本要素

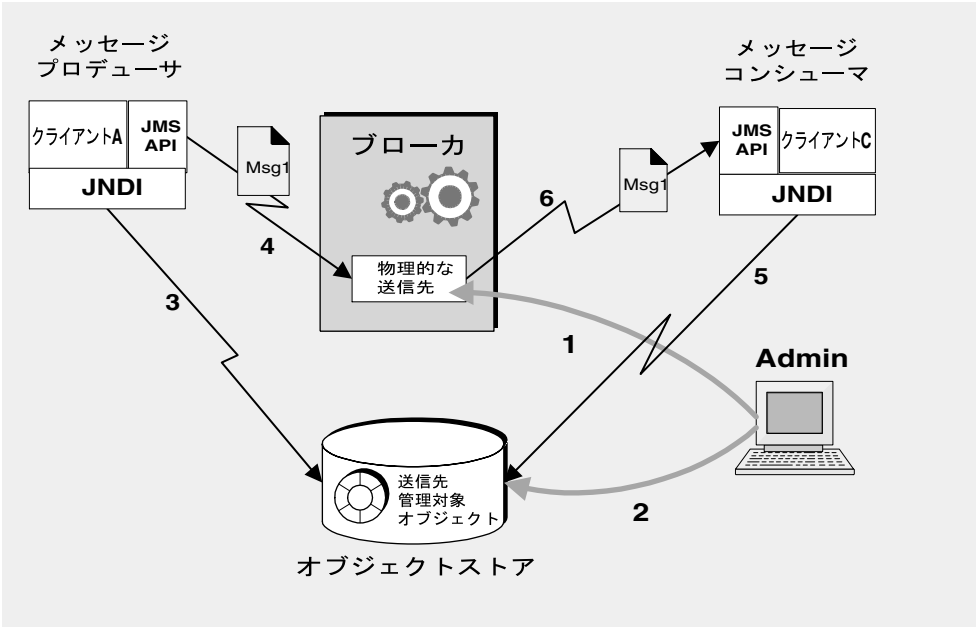


図 1-5 では、メッセージプロデューサとメッセージコンシューマが、送信先管理対象オブジェクトを使用して、それに対応する物理的な送信先にアクセスする様子を示しています。番号の付いた手順は、このメカニズムを使用してメッセージを送受信する場合に、管理者およびクライアントアプリケーションが行う必要のある操作を指しています。

1. 管理者が、ブローカ上に物理的な送信先を作成します。
2. 管理者が、送信先管理対象オブジェクトを作成し、そのオブジェクトが対応する物理的な送信先の名前とそのタイプ (キューまたはトピック) を指定して、オブジェクトを設定します。
3. メッセージプロデューサが、JNDI 検索呼び出しを使用して、送信先管理対象オブジェクトを検索します。
4. メッセージプロデューサが、送信先にメッセージを送信します。

5. メッセージコンシューマが、メッセージを取得できると予想する送信先管理対象オブジェクトを検索します。
6. メッセージコンシューマが、送信先からメッセージを取得します。

コネクションファクトリ管理対象オブジェクトを使用するプロセスも同様です。管理者は、管理ツールを使用して、コネクションファクトリ管理対象オブジェクトを作成および設定します。クライアントは、コネクションファクトリオブジェクトを検索し、そのオブジェクトを使用してコネクションを作成します。

管理対象オブジェクトを使用すると、メッセージングプロセスの手順が増えますが、同時に、メッセージングアプリケーションの堅牢性と移植性も高まります。

Message Queue: 要素と機能

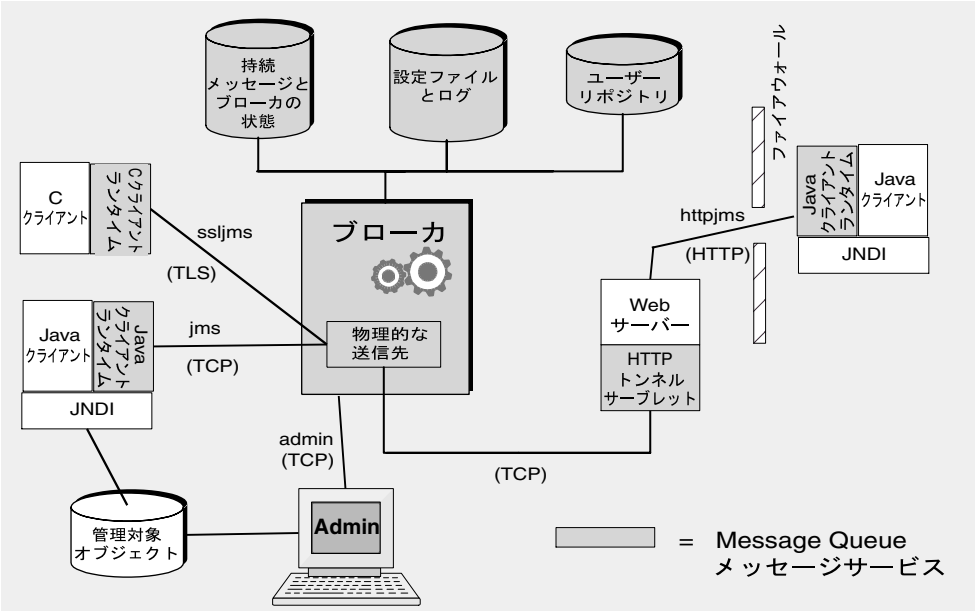
これまで、メッセージ指向ミドルウェアの要素について説明し、JMS を使用して MOM アプリケーションに移植性をもたらす方法について説明してきました。次に、Message Queue での JMS 仕様の実装について説明し、信頼性、安全性、拡張性の高いメッセージングサービスの実現に使用する機能およびツールを紹介します。

まず、多くの JMS プロバイダと同様に、Message Queue は、スタンドアロン製品として使用することも、J2EE アプリケーションサーバーに埋め込んで非同期メッセージングをもたらす、イネーブリングテクノロジとして使用することもできます。第 5 章「Message Queue と J2EE」では、J2EE で Message Queue が果たす役割について、さらに詳しく説明します。ほかの JMS 製品とは異なり、Message Queue は、JMS リファレンス実装と呼ばれています。この名称は、Message Queue が正確で完全な JMS 実装であるということを意味しています。また、将来、JMS に改訂や拡張が行われた場合でも、Message Queue 製品を使用できるということも保証しています。

Message Queue サービス

JMS プロバイダとして、Message Queue には、JMS インタフェースを実装し、管理サービスおよび制御を備えたメッセージングサービスが用意されています。これまでの JMS プロバイダの説明では、主に、メッセージの中継におけるブローカの役割に焦点を当ててきました。しかし、実際、JMS プロバイダには、信頼性、安全性、拡張性の高いメッセージングを提供するために、ブローカ以外にも多くの要素が必要です。図 1-6 では、Message Queue メッセージングサービスを構成する要素を示しています。これらの要素には、さまざまなコネクションサービス (異なるプロトコルに対応) や管理ツールのほか、メッセージング用、監視用、およびユーザー情報用のデータストアがあります。Message Queue サービスには、この図で灰色で示されたすべての要素が含まれます。

図 1-6　Message Queue サービス



見てわかるとおり、フル機能を備えた JMS プロバイダは、基本的な JMS モデルが疑わしく思われるほど複雑です。次の節では、上の図に示した Message Queue サービスの要素について説明します。これらの要素は、ブローカ、クライアントランタイムサポート、管理の 3 つのカテゴリに分類できます。

ブローカへの接続

図 1-6 に示すように、アプリケーションクライアントと管理クライアントの両方を、ブローカに接続できます。JMS 仕様では、プロバイダに実装する特定のワイヤプロトコルを規定していません。アプリケーションクライアントと管理クライアントがブローカへの接続に使用する Message Queue サービスは、現在、TCP、TLS、HTTP、または HTTPS プロトコルの上の層に位置しています。HTTP の上の層に位置しているため、メッセージはファイアウォールを通過できます。

- JMS サポートを実現し、クライアントによるブローカへの接続を可能にしているサービス (`jms`、`ssljms`、`http`、または `https`) は、サービスタイプが `NORMAL` であり、`TCP`、`TLS`、`HTTP`、または `HTTPS` プロトコルの上の層に位置しています。
- 管理者によるブローカへの接続を可能にしているサービス (`admin`、`ssladmin`) は、サービスタイプが `ADMIN` であり、`TCP` または `TLS` プロトコルの上の層に位置しています。

デフォルトでは、ブローカを起動すると、jms および admin サービスが稼働します。さらに、ブローカを設定して、これらのコネクションサービスのいずれかを実行することも、すべてを実行することもできます。各サービスは、特定の認証および承認 (アクセス制御) 機能をサポートしており、複数のコネクションをサポートするマルチスレッドです。

コネクションに障害が発生した場合、Message Queue サービスは、自動的にクライアントを同じブローカに再度接続したり、またこの機能を有効にしている場合は、別のブローカに接続したりすることができます。詳細は、付録 B 「Message Queue の機能」の自動再接続機能についての説明を参照してください。

クライアントは、コネクションを取得するコネクションファクトリを作成するときに、コネクションランタイムサポートを設定できます。オプションを使用すると、接続先のブローカ、再接続の処理方法、メッセージフロー制御などを指定できます。コネクションの設定方法については、50 ページの「コネクションファクトリとコネクション」を参照してください。

ブローカ

ブローカは、メッセージサービスの中心にあたり、信頼性の高いメッセージのルーティングと配信、ユーザーの認証、およびパフォーマンス監視用のデータ収集を行います。

- メッセージをルーティングおよび配信するために、ブローカは、受信メッセージをそれぞれの送信先に保管し、これらの送信先へのメッセージの出入りを管理します。
- 信頼性の高い配信を行うため、ブローカでは持続ストアを使用して、メッセージが受信されるまで状態情報と持続メッセージを保存します。ブローカまたはコネクションに障害が発生した場合、ブローカは、保存された情報に基づいて、ブローカの状態を復元し、操作を再試行できます。
- 交換するデータのセキュリティーを確保するために、ブローカでは認証されたコネクションを使用します。オプションで、SSL などのセキュアプロトコル上で実行することにより、データを暗号化できます。ブローカはまた、ユーザーに関する情報と、ユーザーがアクセス可能なデータまたは操作に関する情報を保持したリポジトリを使用し、管理します。ブローカは、このリポジトリで情報を検索することにより、サービスを要求しているユーザーを認証し、そのユーザーが実行しようとしている操作を承認します。
- システム監視のために、ブローカはメトリックスと診断情報を生成し、管理者はこれらの情報にアクセスして、パフォーマンスを測定しブローカを調整できます。メトリックス情報はプログラムでも利用でき、アプリケーションは、メッセージフローおよびパターンを調整して、パフォーマンスを向上できます。

Message Queue サービスには、管理者がブローカサポートの設定に使用できるさまざまな管理ツールが用意されています。詳細は、35 ページの「管理」を参照してください。

クライアントランタイムサポート

クライアントランタイムサポートは、Message Queue クライアントの作成時に、関連付けるライブラリで提供されます。クライアントランタイムは、Message Queue サービスと見なすことができ、クライアントに含まれます。たとえば、クライアントコードが API 呼び出しを作成してメッセージを送信する場合、これらのライブラリ内のコードが呼び出され、ブローカの物理的な送信先へメッセージを中継するために使用されるプロトコルに合わせて適切にメッセージをパッケージします。

Java および C クライアントサポート

Java クライアントのサポートには、JMS プロバイダのみを必要とします。ただし、図 1-6 に示すように、Message Queue クライアントでは、メッセージの送受信に、Java を使用することも、プロバイダ固有の C API を使用することもできます。これらのインタフェースは、Java または C ランタイムライブラリに実装されています。このランタイムライブラリが、ブローカへのコネクションを作成し、要求されたコネクションサービスに合わせて適切にパッケージするという実際の処理を行います。

- Java クライアントランタイムは、ブローカとの対話に必要なオブジェクトを Java クライアントに提供します。これらのオブジェクトには、コネクション、セッション、メッセージ、メッセージプロデューサ、メッセージコンシューマなどがあります。
- C クライアントランタイムは、C クライアントに、ブローカとの対話に必要な関数と構造を提供します。C クライアントランタイムは、JMS プログラミングモデルのプロシージャー版をサポートします。C クライアントは、管理対象オブジェクトへのアクセスに JNDI を使用できませんが、コネクションファクトリと送信先をプログラムによって作成できます。

Message Queue サービスには C API が用意されているため、旧バージョンの C アプリケーションと C++ アプリケーションを、JMS ベースのメッセージングに加えることができます。これら 2 つの API によって提供される機能には多数の相違点があります。これらについては、65 ページの「Java クライアントと C クライアント」で説明しています。

JMS 仕様は Java クライアントに限定した標準であることに注意してください。C サポートは、Message Queue プロバイダ固有の機能です。ほかのプロバイダに移植しようとしているクライアントアプリケーションでは使用しないでください。

Java クライアントでの SOAP サポート

Message Queue Java クライアントは、SOAP メッセージを JMS メッセージとしてラップして送受信することもできます。SOAP (Simple Object Access Protocol) を使用すると、分散環境にある 2 つのピア間で構造化データを交換できます。交換されるデータは、XML 方式で指定されます。

Sun の SOAP 処理は、現在、ポイントツーポイントモデルを使用した場合に制限されており、信頼性が保証されません。SOAP メッセージを JMS メッセージにラップし、ブローカを使用してルーティングすることにより、フル機能を備えた Message Queue メッセージングを活用することができます。これにより、信頼性の高い配信が保証され、ポイントツーポイントドメインのほかにトピックを使用できるようになります。Message Queue には、メッセージプロデューサが SOAP メッセージを JMS メッセージにラップする際に使用でき、メッセージコンシューマが SOAP メッセージを JMS メッセージから抽出する際に使用できるユーティリティールーティンが用意されています。

64 ページの「SOAP メッセージの処理」では、SOAP メッセージ処理について詳しく説明しています。

管理

Message Queue サービスには、次の作業に使用できるコマンド行ツールが用意されています。

- ブローカの起動および設定。
- 送信先の作成および管理、ブローカコネクションの管理、およびブローカリソースの管理。
- JNDI オブジェクトストア内の管理対象オブジェクトの追加、一覧表示、更新、および削除。
- ファイルベースのユーザーリポジトリの入力と管理。
- 持続ストレージ用の JDBC 準拠データベースの作成と管理。

また、GUI ベースの管理コンソールを使用して、次のコマンド行機能を実行できます。

- ブローカへの接続および管理。
- 物理的な送信先の作成および管理。
- オブジェクトストアへの接続、ストアへのオブジェクトの追加、およびそれらの管理。

Message Queue サービスの拡張

クライアント数またはコネクション数が増加するにつれ、ボトルネックを解消し、パフォーマンスを向上させるために、メッセージサービスを拡張する必要が生じる場合があります。Message Queue メッセージサービスでは、ユーザーのニーズに応じた拡張オプションを多数用意しています。これらは、便宜上、次のカテゴリに分類されます。

- 垂直方向の拡張は、より高い処理能力を追加し、利用可能なリソースを拡張することによって実現できます。この拡張を行うには、プロセッサまたはメモリーを追加するか、共有スレッドモデルへ切り替えるか、または Java VM を 64bit モードで実行します。

  ポイントツーポイントドメインを使用している場合は、複数のコンシューマからのキューへのアクセスを有効にすることにより、コンシューマ側を拡張できます。この方式を使用すると、アクティブおよびバックアップコンシューマの最大数を指定できます。ロードバランスメカニズムでも、コンシューマの現在の容量とメッセージ処理速度を考慮します。これは、Message Queue の機能です。JMS 仕様で規定しているメッセージング動作は、1 つのコンシューマのみがキューにアクセスしている場合のものです。複数のコンシューマを許可するキューの動作は、プロバイダ固有の機能です。Message Queue 開発者ガイドで、この拡張オプションの詳細について説明しています。

- ステートレスな水平方向の拡張は、追加ブローカを使用して、これらのブローカに既存のクライアントを分散することにより実現できます。この方式は実装が簡単ですが、独立した作業グループにメッセージング操作を分割できる場合にのみ適しています。

- ステートフルな水平方向の拡張は、複数のブローカを接続してクラスタを構成することにより実現されます。ブローカクラスタでは、各ブローカは、クラスタ内のほかのすべてのブローカ、およびローカルアプリケーションクライアントにも接続します。ブローカは、同一ホスト上に置くことも、ネットワーク上に分散させることもできます。送信先とコンシューマに関する情報は、クラスタ内のすべてのブローカで複製されます。送信先またはサブスクライバに対する更新も伝えられます。これにより、各ブローカは、直接接続しているプロデューサから、クラスタ内のほかのブローカに接続しているコンシューマへ、メッセージをルーティングできます。バックアップコンシューマが使用されている状況では、1 つのブローカまたはコネクションに障害が発生した場合、アクセスできないコンシューマへ送信されたメッセージを、別のブローカ上のバックアップコンシューマに転送できます。

  ブローカまたはコネクションに障害が発生した場合、持続エンティティー (送信先および永続サブスクリプション) に関する状態情報が同期されなくなる可能性があります。たとえば、クラスタ化したブローカが停止し、送信先がクラスタ内の別のブローカ上に作成された場合、最初のブローカが再起動されると新しい送信先が不明になります。この問題を防ぐために、クラスタ内の 1 つのブローカを

マスターブローカに指定することができます。このブローカは、マスター設定ファイルでの送信先と永続サブスクリプションに対するすべての変更を追跡し、一時的にオフラインになっているクラスタ内のブローカを更新します。詳細は、第 4 章「ブローカクラスタ」を参照してください。

マスターブローカを使用した場合でも、Message Queue ではサービスの可用性しか向上しません。ブローカまたはコネクションに障害が発生した場合、データの可用性は確保されません。たとえば、クラスタ化したブローカが利用できなくなった場合、そのブローカが保持しているすべての持続メッセージは、そのブローカが復元されるまで利用できません。現在のところ、データの可用性を確保する唯一の手段は、SunCluster Message Queue エージェントを使用するというものです。この場合、持続ストアが共有ファイルシステム上に保持されます。ブローカに障害が発生した場合、2 番目のノード上の Message Queue エージェントがブローカを起動して、このブローカが共有ストアを引き継ぎます。クライアントはそのブローカに接続し直されます。この結果、サービスの継続と持続データへのアクセスの両方が確保されます。

イネーブリングテクノロジとしての Message Queue

Java 2 Platform, Enterprise Edition (J2EE プラットフォーム) は、Java プログラミング環境における分散コンポーネントモデルについて規定する仕様です。J2EE プラットフォームの要件の 1 つは、分散コンポーネントが、信頼性の高い非同期メッセージ交換機能により相互に対話できるようにすることです。この機能は JMS プロバイダが提供するものであり、サービスを提供するという役割と、JMS メッセージをコンシュームできる特殊な種類の EJB (Enterprise Java Bean) コンポーネントである MDB (メッセージ駆動型 Beans) をサポートするという 2 つの役割を果たします。

J2EE 準拠のアプリケーションサーバーは、指定された JMS プロバイダの機能を使用する場合、そのプロバイダによって用意されたリソースアダプタを使用する必要があります。Message Queue は、このようなリソースアダプタを用意しています。アプリケーションサーバー環境に配備され稼働する MDB などの J2EE コンポーネントは、プラグインの JMS プロバイダのサポートを使用すると、J2EE コンポーネント、および外部の JMS コンポーネントと、JMS メッセージを交換できます。これにより、分散コンポーネントを緊密に統合することができます。

Message Queue リソースアダプタについては、第 5 章「Message Queue と J2EE」を参照してください。

製品エディション

Message Queue 製品には、Enterprise および Platform の 2 つのエディションが用意されています。どちらのエディションも JMS 仕様を完全に実装していますが、それぞれ異なる機能セットとライセンス数に対応しています。

Enterprise Edition は、Platform Edition に次の機能を追加したものです。

- HTTP および HTTPS サポート
- C クライアントサポート
- スケーラブルなコネクション機能
- ブローカクラスタ
- キューあたり無制限のメッセージコンシューマへのキューの配信。Platform Edition では、キューあたり 3 つのコンシューマに配信を制限しています。
- クラスタ内の別のブローカへのクライアントコネクションのフェイルオーバー
- メッセージベースの監視 API

Message Queue Enterprise Edition には、使用する CPU 数に基づいた無制限の永続的ライセンスが用意されています。このライセンスでは、複数ブローカメッセージサービスでのブローカ数に制限がありません。

Message Queue Platform Edition では、ブローカでサポートされるクライアントコネクションの数に制限がありません。製品には、基本ライセンスまたは 90 日間トライアルライセンスが付いています。

- **基本ライセンス**には有効期限はありませんが、Enterprise Edition 機能は含まれません。
- **90 日間の企業向けトライアルライセンス**では、すべての Enterprise Edition 機能を使用および評価できますが、90 日間の有効期限が設けられています。このライセンスの使用方法については、『Message Queue 管理ガイド』の起動オプションの説明を参照してください。

ライセンス対象の機能とライセンス数、再配布権、および Platform Edition から Enterprise Edition へのアップグレードについては、『Message Queue Installation Guide』を参照してください。

Message Queue 機能の要約

Message Queue は、JMS 仕様の要件よりはるかに多くの機能や特長を備えています。Message Queue は、これらの機能により、24 時間稼働の基幹業務で大量のメッセージを交換する多くの分散コンポーネントで構成されるシステムを統合できます。これらの機能についての概要は、付録 B 「Message Queue の機能」を参照してください。

第 2 章

クライアントプログラミングモデル

この章では、Message Queue クライアントプログラミングの基本について説明します。次のトピックが含まれます。

この章では、Java クライアントの設計と実装を中心に説明します。概して、C クライアントの設計は、Java クライアントの設計とほぼ同じです。この章の最終節では、Java クライアントと C クライアントの相違点をまとめています。Message Queue クライアントのプログラミングの詳細は、『Message Queue Developer's Guide for Java Clients』および『Message Queue Developer's Guide for C Clients』を参照してください。

第 3 章「Message Queue サービス」では、Message Queue サービスを使用して、メッセージングパフォーマンスのサポート、管理、および調整を行う方法について説明します。

設計とパフォーマンス

Message Queue アプリケーションの動作は、クライアント設計、コネクション設定、ブローカ設定、ブローカの調整、リソース管理など、多くの要因に左右されます。一部の要因は開発者の扱う範囲ですが、ほかの要因は管理者の問題になります。ただし、理想的には、開発者は Message Queue サービスでのアプリケーション設計のサポートと拡張について意識し、管理者はアプリケーションを調整するときに設計目標を意識することが望まれます。メッセージングの動作は、再設計によっても、注意深い監視および調整によっても最適化できます。したがって、優れた Message Queue アプリケーションの作成における重要な点は、開発者と管理者が、アプリケーションのライフサイクルの各段階で実現できることを理解し、所期の動作と実際の動作に関する情報を共有することです。

メッセージングドメイン

メッセージングミドルウェアは、メッセージのプロデュースとコンシュームを行うことにより、コンポーネントおよびアプリケーションの通信を実現します。JMS API では、この通信を制御するパターン、すなわちメッセージングドメインとして、ポイントツーポイントメッセージングとパブリッシュ / サブスクライブメッセージングの 2 つを規定しています。JMS API は、これらのパターンをサポートするように構成されています。コネクション、セッション、プロデューサ、コンシューマ、送信先、メッセージという基本的な JMS オブジェクトが、両方のドメインでメッセージング動作を指定するために使用されます。

ポイントツーポイントメッセージング

ポイントツーポイントドメインでは、メッセージプロデューサは送信側と呼ばれ、コンシューマは受信側と呼ばれます。これらは、キューと呼ばれる送信先を使用してメッセージを交換します。送信側は、キューへメッセージをプロデュースし、受信側はキューからメッセージをコンシュームします。

図 2-1 に、ポイントツーポイントドメインでの最も単純なメッセージング操作を示します。`MyQueueSender` は、`Msg1` をキュー送信先 `MyQueue1` に送信します。続いて、`MyQueueReceiver` は、`MyQueue1` からメッセージを取得します。

図 2-1　単純なポイントツーポイントメッセージング

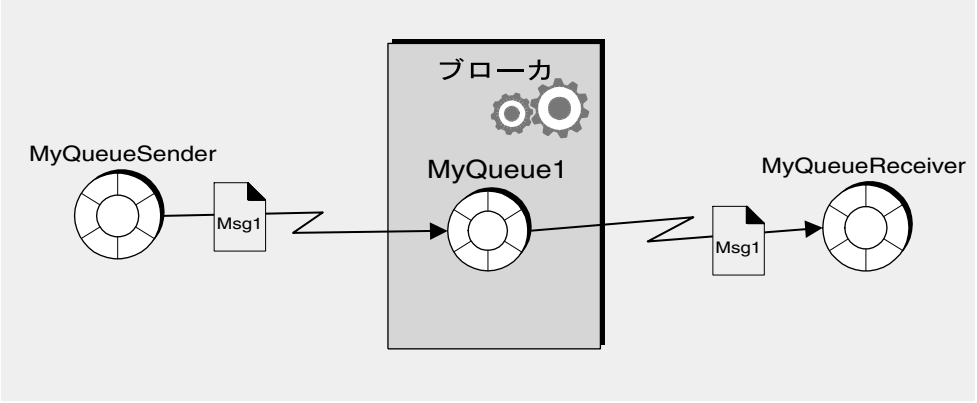
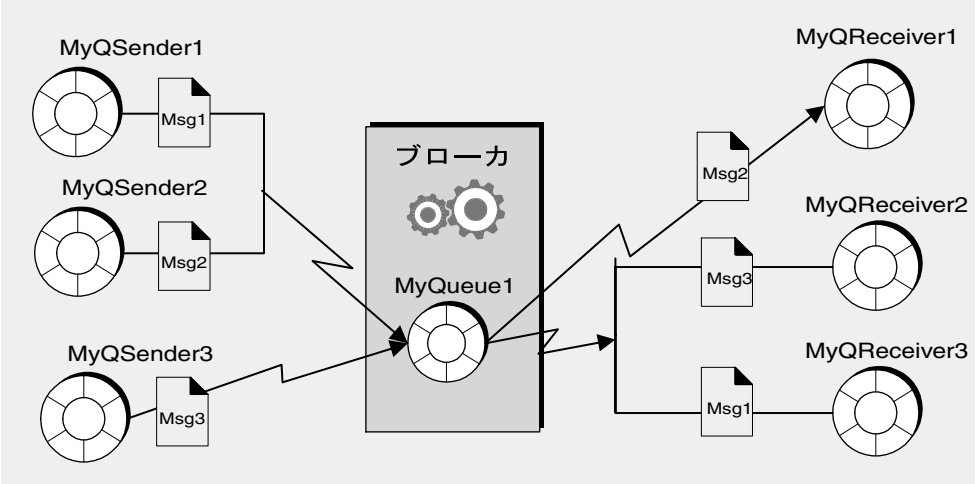


図 2-2 に、このドメインでの可能性を表すために、より複雑なポイントツーポイントメッセージングのイメージを示します。2 つの送信側、MyQSender1 と MyQSender2 は、同じコネクションを使用して、メッセージを MyQueue1 に送信します。MyQSender3 は、別のコネクションを使用して、メッセージを MyQueue1 に送信します。受信側では、MyQReceiver1 が MyQueue1 からメッセージをコンシュームし、MyQReceiver2 と MyQReceiver3 は、コネクションを共有して、MyQueue1 からメッセージをコンシュームします。

図 2-2　複雑なポイントツーポイントメッセージング



この複雑な図では、ポイントツーポイントメッセージングに関するその他の要点が多数示されています。

- 複数のプロデューサから 1 つのキューにメッセージを送信できる。プロデューサは、1 つのコネクションを共有することも、別々のコネクションを使用することもできますが、すべてが同じキューにアクセスできます。
- 複数の受信側がキューからメッセージをコンシュームできるが、それぞれのメッセージは、単一の受信側でしかコンシュームできない。このため、`Msg1`、`Msg2`、および `Msg3` は、別々の受信側にコンシュームされます。これは、Message Queue の拡張機能です。
- 受信側は、1 つのコネクションを共有することも、別々のコネクションを使用することもできるが、すべてが同じキューにアクセスできる。これは、Message Queue の拡張機能です。
- 送信側と受信側はタイミング依存性がない。受信側は、クライアントがメッセージを送信したときに稼働していてもしていなくても、メッセージを取り出せます。
- 送信側と受信側は、実行時に、動的に追加および削除できるので、必要に応じて、メッセージングシステムを拡張したり縮小したりできる。
- メッセージは、送信順にキューに入れられるが、コンシュームされる順番は、メッセージの有効期限、メッセージの優先度、メッセージのコンシュームでセレクタを使用するかどうかなどの要因によって決まる。

ポイントツーポイントモデルには、次のような多くのメリットがあります。

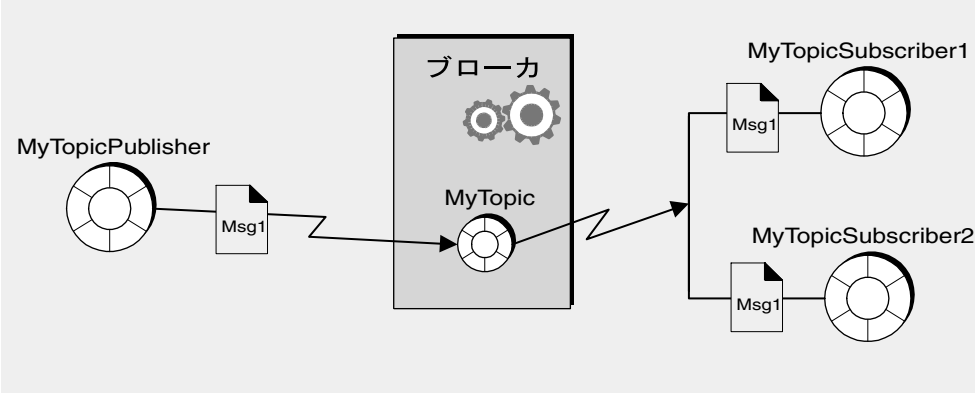
- 複数の受信側が同じキューからメッセージをコンシュームできることにより、メッセージを受信する順序が重要でない場合は、メッセージコンシュームをロードバランスできる。これは、Message Queue の拡張機能です。
- 受信側がない場合でも、キューに送られるメッセージは常に保持される。
- Java クライアントは、キューブラウザオブジェクトを使用して、キューの内容を調べることができる。続いてクライアントは、この調査から得られた情報に基づいて、メッセージをコンシュームできます。つまり、コンシュームモデルは通常 FIFO (先入れ先出し) ですが、メッセージセレクタを使用することにより、どのメッセージが必要であるかがわかれば、コンシューマはキューの先頭にはないメッセージでもコンシュームできます。管理クライアントも、キューブラウザを使用して、キューの内容を監視できます。

パブリッシュ / サブスクライブメッセージング

パブリッシュ / サブスクライブドメインでは、メッセージプロデューサは、パブリッシャーと呼ばれ、メッセージコンシューマは、サブスクライバと呼ばれます。これらは、トピックと呼ばれる送信先を使用してメッセージを交換します。パブリッシャーは、トピックへメッセージをプロデュースし、サブスクライバはトピックへサブスクライブして、トピックからメッセージをコンシュームします。

図 2-3 に、パブリッシュ / サブスクライブドメインの単純なメッセージング操作を示します。MyTopicPublisher は、Msg1 を送信先 MyTopic へパブリッシュします。続いて、MyTopicSubscriber1 と MyTopicSubscriber2 はそれぞれ、MyTopic から Msg1 のコピーを受信します。

図 2-3　単純なパブリッシュ / サブスクライブメッセージング

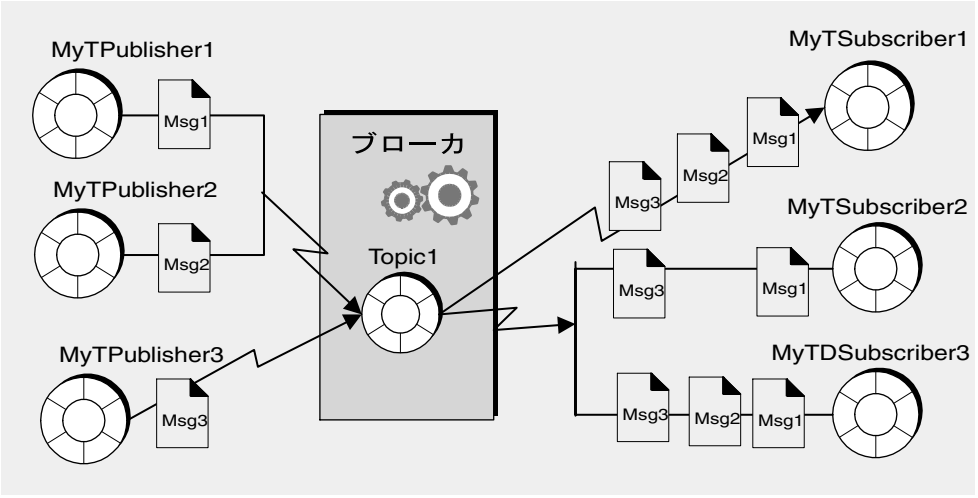


パブリッシュ / サブスクライブモデルでは、複数のサブスクライバが必要になるわけではありませんが、このドメインが複数のメッセージをブロードキャストできることを強調するために、図では 2 つのサブスクライバを示しています。トピックに対応するすべてのサブスクライバが、このトピックへパブリッシュされた任意のメッセージのコピーを取得します。

サブスクライバは、非永続でも永続でもかまいません。ブローカは、すべてのアクティブなサブスクライバのメッセージを保持しますが、非アクティブなサブスクライバのメッセージは、これらのサブスクライバが永続である場合にのみ保持します。

図 2-4 に、このパターンによって実現する可能性を表すために、より複雑なパブリッシュ / サブスクライブメッセージングのイメージを示します。複数のプロデューサがメッセージを Topic1 送信先にパブリッシュします。複数のサブスクライバが、Topic1 送信先からメッセージをコンシュームします。サブスクライバがセレクタを使用してメッセージをフィルタ処理しないかぎり、各サブスクライバは、選択したトピックへパブリッシュされたすべてのメッセージを取得します。図 2-4 では、MyTSubscriber2 は、Msg2 をフィルタで除外しています。

図 2-4　複雑なパブリッシュ / サブスクライブメッセージング



この複雑な図では、パブリッシュ / サブスクライブメッセージングに関するその他の要点が多数示されています。

- 複数のプロデューサから 1 つのトピックにメッセージをパブリッシュできる。プロデューサは、1 つのコネクションを共有することも、別々のコネクションを使用することもできますが、すべてが同じトピックにアクセスできます。
- 複数のサブスクライバが 1 つのトピックからメッセージをコンシュームできる。サブスクライバは、セレクタを使用してメッセージをフィルタで除外したり、コンシューム前にメッセージの有効期限が切れないかぎり、トピックへパブリッシュされたすべてのメッセージを取得します。
- サブスクライバは、1 つのコネクションを共有することも、別々のコネクションを使用することもできるが、すべてが同じトピックにアクセスできる。
- 永続サブスクライバは、アクティブでも非アクティブでもかまわない。非アクティブの間、サブスクライバへのメッセージはブローカが保持します。
- パブリッシャーとサブスクライバは、実行時に、動的に追加および削除できるので、必要に応じて、メッセージングシステムを拡張したり縮小したりできる。
- メッセージは、送信順にトピックへパブリッシュされるが、コンシュームされる順番は、メッセージの有効期限、メッセージの優先度、メッセージのコンシュームでセレクタを使用するかどうかなどの要因によって決まる。
- パブリッシャーとサブスクライバはタイミング依存性がある。トピックサブスクライバは、サブスクリプションの作成後にパブリッシュされたメッセージだけをコンシュームできます。

パブリッシュ / サブスクライブモデルの主要なメリットは、複数のサブスクライバへメッセージをブロードキャストできるという点です。

ドメイン固有 API と統合 API

JMS API では、ポイントツーポイントドメインまたはパブリッシュ / サブスクライブドメインのいずれかの実装に使用できるインタフェースとクラスを規定しています。これらは、表 2-1 の 2 列目と 3 列目に示したドメイン固有 API です。JMS API では、そのほかに統合ドメインを規定しており、これを使用すれば、汎用メッセージングクライアントをプログラムできます。このようなクライアントの動作は、メッセージのプロデュース先でありメッセージのコンシューム元である送信先のタイプによって決定されます。送信先がキューの場合、メッセージングはポイントツーポイントのパターンに従って動作します。送信先がトピックの場合、メッセージングはパブリッシュ / サブスクライブパターンに従って動作します。

表 2-1 JMS プログラミングドメインとオブジェクト

基本タイプ (統合ドメイン)	ポイントツーポイントドメイン	パブリッシュ / サブスクライブドメイン
Destination (Queue または Topic)*	Queue	Topic
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

* プログラミングの手法によっては、特定の送信先タイプを指定する必要があります。

統合ドメインは、JMS バージョン 1.1 から導入されています。以前の 1.02b 仕様に適合する必要がある場合は、ドメイン固有 API を使用できます。ドメイン固有 API を使用すると、たとえばキュー送信先に対する永続サブスクライバの作成など、特定のプログラミングエラーを防止するクリーンなプログラムインタフェースが得られます。ただしドメイン固有 API には、同じトランザクションまたは同じセッションにおいて、ポイントツーポイント操作とパブリッシュ / サブスクライブ操作を組み合わせることができないというマイナス面もあります。両方の操作を組み合わせる必要がある場合は、統合ドメイン API を選択してください。2 つのドメインの組み合わせ例については、57 ページの「要求 / 応答パターン」を参照してください。

プログラミングオブジェクト

JMS メッセージングの実装に使用されるオブジェクトは、基本的にプログラミングドメイン全体で同じです (コネクションファクトリ、コネクション、セッション、プロデューサ、コンシューマ、メッセージ、および送信先)。図 2-5 に、これらのオブジェクトを示します。図の上から下に向かい、コネクションファクトリオブジェクトからどのようにオブジェクトが生成されるかを示しています。

これらのオブジェクトのうち、コネクションファクトリと送信先の 2 つは、1 つのオブジェクトストア内にあるように示されています。これは、これらのオブジェクトが通常、管理対象オブジェクトとして作成、設定、管理されるという点を強調するためです。この章を通じて、コネクションファクトリと送信先は、プログラムではなく管理操作によって作成されると想定しています。

図 2-5　JMS プログラミングオブジェクト

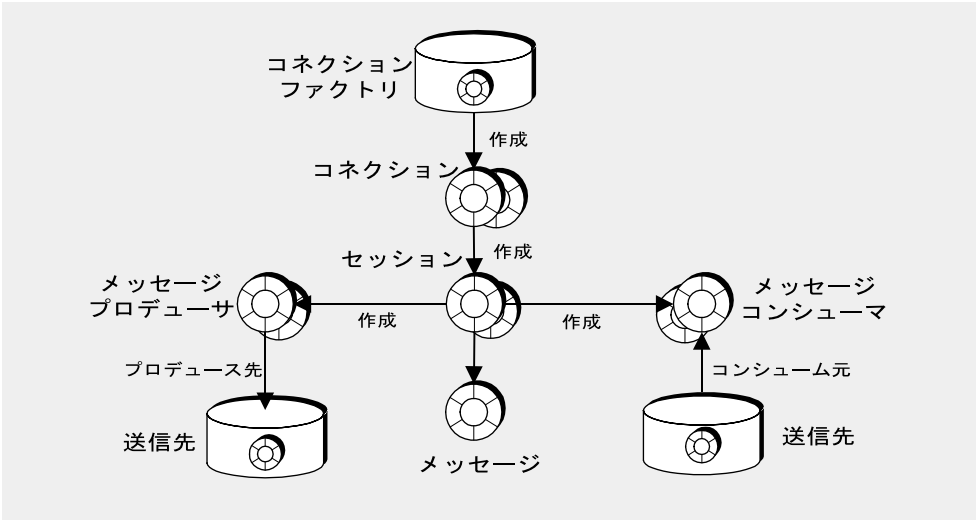


表 2-2 は、メッセージの送受信に必要な手順をまとめています。手順 1、および 3 から 6 は、送信側と受信側で同じであることに注意してください。

表 2-2　メッセージのプロデュースとコンシューム

メッセージのプロデュース	メッセージのコンシューム
1. 管理者がコネクションファクトリ管理対象オブジェクトを作成します。	
2. 管理者が、物理的な送信先と、それを参照する管理対象オブジェクトを作成します。	
3. クライアントが、JNDI 検索を使用して、コネクションファクトリオブジェクトを取得します。	
4. クライアントが、JNDI 検索を使用して、送信先オブジェクトを取得します。	
5. クライアントが、コネクションを作成し、このコネクションに固有の任意のプロパティーを設定します。	
6. クライアントが、セッションを作成し、メッセージングの信頼性を決定するプロパティーを設定します。	
7. クライアントがメッセージプロデューサを作成します。	クライアントがメッセージコンシューマを作成します。
8. クライアントがメッセージを作成します。	クライアントがコネクションを開始します。
9. クライアントがメッセージを送信します。	クライアントがメッセージを受信します。

以降の節では、プロデューサとコンシューマが使用するオブジェクト (コネクション、セッション、メッセージ、および送信先) について説明します。続いて、メッセージのプロデュースとコンシュームについて説明し、JMS オブジェクトの説明を締めくくります。

コネクションファクトリとコネクション

クライアントは、コネクションファクトリ (connection factory) オブジェクト (ConnectionFactory) を使用して、コネクション (connection) を作成します。コネクションオブジェクト (Connection) は、ブローカへのクライアントのアクティブなコネクションを表します。デフォルトで起動されるか、このクライアントの管理者が明示的に起動する基礎となるコネクションサービスを使用します。

通信リソースの割り当てとクライアントの認証は、コネクションが作成されるときに行われます。このオブジェクトは比較的重いため、ほとんどのクライアントはメッセージングのすべてを 1 つのコネクションだけで行います。コネクションは同時使用をサポートするため、複数のプロデューサとコンシューマが 1 つのコネクションを共有できます。

コネクションファクトリを作成するときに、そのプロパティーを設定しておくことによって、コネクションファクトリから生成されるすべてのコネクションの動作を設定できます。Message Queue の場合、次の情報を指定します。

- ブローカが存在するホストの名前、必要なコネクションサービス、およびそのサービスへアクセスするためにクライアントが使用するポート。
- コネクションに障害が生じた場合に、ブローカへの自動再接続を処理する方法。この機能は、コネクションが失われた場合に、クライアントを同じ (または別の) ブローカに接続し直します。データのフェイルオーバーは保証されません。別のブローカへ再接続した場合、持続メッセージと他の状態情報が失われる可能性があります。
- 永続サブスクリプションを追跡するためにブローカが必要とするクライアントの ID。
- 接続を試行するユーザーのデフォルト名とパスワード。接続時にパスワードが指定されない場合、この情報がユーザーの認証と操作の承認に使用されます。
- 信頼性に関係のないクライアントに対して、ブローカ通知を抑制すべきかどうか。
- ブローカとクライアントランタイムとの間で、コントロールメッセージおよびペイロードメッセージのフローを制御する方法。
- キューのブラウズの処理方法。(Java クライアントのみ)
- 特定のメッセージヘッダーフィールドをオーバーライドすべきかどうか。

クライアントアプリケーションの起動に使用するコマンド行から、コネクションファクトリのプロパティーをオーバーライドできます。また、どのコネクションのプロパティーも、そのコネクションのプロパティーを設定することによってオーバーライドできます。

コネクションオブジェクトを使用して、セッション (session) オブジェクトの作成、例外リスナーの設定、または JMS バージョンおよびプロバイダ情報の取得を行えます。

セッション

コネクションがクライアントとブローカ間の通信チャネルである場合、セッションは、クライアントとブローカ間の単一のやり取りをマークします。セッションオブジェクトは、主にメッセージ、メッセージプロデューサ、およびメッセージコンシューマの作成に使用します。セッションを作成するときには、多数の 通知 (acknowledgement) オプションまたはトランザクションを使用して、信頼性の高い配信を設定します。詳細は、59 ページの「信頼性の高いメッセージング」を参照してください。

JMS 仕様によれば、セッションは、シングルスレッドコンテキストで、メッセージのプロデュースとコンシュームを実行します。単一のセッションに対して複数のメッセージプロデューサとコンシューマを作成できますが、順番に使用するという制限があります。スレッド処理の実装は、Java クライアントと C クライアントでは少し異なります。スレッド処理の実装および制限の詳細は、該当する開発者ガイドを参照してください。

また、セッションオブジェクトを使用して、次の処理も行えます。

- 管理対象オブジェクトを使用せずに送信先を定義するクライアントの送信先の作成および設定。
- 一時トピックおよびキューの作成および設定。これらは、要求 / 応答パターンの一部として使用されます。57 ページの「要求 / 応答パターン」を参照してください。
- トランザクション処理のサポート。
- メッセージのプロデュースまたはコンシュームの順番の定義。
- 非同期コンシューマに対するメッセージリスナーの実行のシリアライズ。
- キューブラウザの作成。(Java クライアントのみ)

メッセージ

メッセージは、ヘッダー、プロパティー、および本体の 3 つの部分で構成されています。メッセージを正しく作成し、特定のメッセージング動作を設定するために、この構造を理解する必要があります。

メッセージヘッダー

すべての JMS メッセージにはヘッダーが必要です。ヘッダーには、あらかじめ定義された 10 のフィールドがあります。これらについて、表 2-3 で説明します。

表 2-3　JMS 規定のメッセージヘッダー

ヘッダーフィールド	説明
JMSDestination	メッセージを送信する送信先オブジェクトの名前を指定します。(プロバイダが設定する)
JMSDeliveryMode	メッセージが持続かどうかを指定します。(デフォルトでプロバイダが設定するか、プロデューサまたは個々のメッセージごとにクライアントが明示的に設定する)
JMSExpiration	メッセージの有効期限を指定します。(デフォルトでプロバイダが設定するか、プロデューサまたは個々のメッセージにごとにクライアントが設定する)
JMSPriority	0 (低) から 9 (高) の範囲内で、メッセージの優先度を指定します。(デフォルトでプロバイダが設定するか、プロデューサまたは個々のメッセージごとにクライアントが明示的に設定する)
JMSMessageID	プロバイダのインストールのコンテキスト内で、メッセージに対する一意の ID を指定します。(プロバイダが設定する)
JMSTimestamp	プロバイダがメッセージを受信した時間を指定します。(プロバイダが設定する)
JMSCorrelationID	クライアントが 2 つのメッセージのやりとりを定義できるようにするための値です。(必要に応じてクライアントが設定する)
JMSReplyTo	コンシューマが応答を送信すべき送信先を指定します。(必要に応じてクライアントが設定する)
JMSType	メッセージセレクタが評価できる値です。(必要に応じてクライアントが設定する)
JMSRedelivered	メッセージがすでに配信されたが通知されていないかどうかを指定します。(プロバイダが設定する)

この表からわかるように、メッセージヘッダーフィールドは、メッセージの識別、メッセージのルーティングの設定、メッセージ処理に関する情報の提供など、さまざまな目的に使用されます。

最も重要なフィールドの 1 つである JMSDeliveryMode によって、メッセージ配信の信頼性が決まります。このフィールドは、メッセージが持続かどうかを示します。

- 持続メッセージは、必ず 1 回だけ配信されてコンシュームされることが保証されています。持続メッセージは、メッセージサービスで障害が発生しても消失しません。

- 非持続メッセージは、1 回は配信されることが保証されています。非持続メッセージは、メッセージサービスで障害が発生した場合に消失する可能性があります。

メッセージヘッダーフィールドにはプロバイダ (ブローカまたはクライアントランタイム) が設定するものも、クライアントが設定するものもあります。メッセージプロデューサは、特定のメッセージング動作を得るために、ヘッダー値を設定する必要が生じる場合があります。メッセージコンシューマは、メッセージがどのようにルーティングされ、どのような追加処理が必要かを認識するために、ヘッダー値を読み込む必要が生じる場合があります。

ヘッダーフィールド (`JMSDeliveryMode`、`JMSExpiration`、および `JMSPriority`) は、次の 3 つの異なるレベルに設定できます。

- コネクションファクトリから生成されたすべてのコネクションから出されるメッセージ用。
- プロデュースされた各メッセージ用。
- 特定のメッセージプロデューサから出されるすべてのメッセージ用。

これらのフィールドを複数のレベルに設定した場合、コネクションファクトリに対して設定された値は、個々のメッセージに対して設定された値をオーバーライドします。特定のメッセージに対して設定された値は、メッセージのプロデューサに対して設定された値をオーバーライドします。

メッセージヘッダーフィールドに使用する定数名は、実装されている言語によって異なります。詳細は、『Message Queue Developer's Guide for Java Clients』または『Message Queue Developer's Guide for C Clients』を参照してください。

メッセージプロパティー

メッセージは、プロパティーと呼ばれるオプションのヘッダーフィールドも含めることができます。このフィールドは、プロパティー名とプロパティー値のペアとして指定されます。クライアントとプロバイダはプロパティーを使用して、メッセージヘッダーを拡張し、メッセージの識別と処理に役立つすべての情報を含めることができます。受信側クライアントは、メッセージプロパティーを使用して、指定された基準に適合したメッセージだけを配信するように要求できます。たとえば、コンシューミングクライアントは、ニュージャージーで働くパートタイム従業員の給与に関するメッセージが必要であることを示すことができます。プロバイダは、指定された基準に適合しないメッセージを配信しません。

JMS 仕様では、9 つの標準プロパティーが規定されています。クライアントが設定するプロパティーもあれば、プロバイダが設定するプロパティーもあります。その名前は予約文字「JMSX」で始まります。クライアントまたはプロバイダは、これらのプロパティーを使用して、メッセージの送信者、メッセージの状態、メッセージが配信された回数および時間を判断できます。これらのプロパティーは、プロバイダがメッセージをルーティングしたり、診断情報を提供したりする際に役立ちます。

Message Queue でもメッセージプロパティーが定義されています。これらのプロパティーは、圧縮されたメッセージを識別し、そのメッセージを配信できない場合の処理方法を識別するために使用されます。詳細は、『Message Queue Developer's Guide for Java Clients』を参照してください。

メッセージ本体

メッセージ本体には、クライアントが交換しようとするデータが含まれます。

表 2-4 に示すように、JMS メッセージのタイプによって、本体に含まれる内容と、コンシューマによる処理方法が決まります。セッションオブジェクトには、メッセージ本体の各タイプに応じた作成メソッドが含まれます。

表 2-4　メッセージ本体のタイプ

タイプ	説明
StreamMessage	本体に Java プリミティブ値のストリームを含むメッセージです。順番に入力され、読み取られます。
MapMessage	本体に名前値のペアを含むメッセージです。エントリの順番は定義されていません。
TextMessage	本体に、XML メッセージなどの Java 文字列を含むメッセージです。
ObjectMessage	本体にシリアライズされた Java オブジェクトを含むメッセージです。
BytesMessage	本体に未解釈バイトのストリームを含むメッセージです。

Java クライアントは、送信するメッセージの本体をクライアントランタイムで圧縮するように、プロパティーを設定することができます。コンシューマ側の Message Queue ランタイムは、メッセージを解凍してから配信します。

メッセージのプロデュース

メッセージは、コネクションおよびセッションのコンテキスト内でメッセージプロデューサによって送信またはパブリッシュされます。メッセージのプロデュースは、とても簡単です。クライアントは、メッセージプロデューサオブジェクト (MessageProducer) を使用して、送信先オブジェクトにより API 内に示される物理的な送信先 (destination) にメッセージを送信します。

プロデューサを作成するときに、すべてのプロデューサのメッセージが送信されるデフォルトの送信先を指定できます。また、持続性、優先度、および有効期間を制御するメッセージヘッダーフィールドのデフォルト値も指定できます。したがって、メッセージの送信時に代わりの送信先を指定したり、指定されたメッセージのヘッダーフィールドに代わりの値を設定したりすることによってオーバーライドしないかぎり、これらのデフォルト値は、そのプロデューサから出されるすべてのメッセージによって使用されます。

メッセージプロデューサは、JMSReplyTo メッセージヘッダーフィールドを設定することにより、要求 / 応答パターンも実装できます。詳細は、57 ページの「要求 / 応答パターン」を参照してください。

メッセージのコンシューム

メッセージは、コネクションおよびセッションのコンテキスト内でメッセージコンシューマによって受信されます。クライアントは、メッセージコンシューマオブジェクト (MessageConsumer) を使用して、送信先オブジェクトとして API 内に示される指定された物理的な送信先からメッセージを受信します。

次の 3 つの要因は、ブローカがメッセージをコンシューマに配信する方法に影響します。

- コンシュームが同期か非同期か
- 受信メッセージのフィルタ処理にセレクタを使用するかどうか
- メッセージがトピック送信先からコンシュームされる場合、サブスクライバが永続かどうか

メッセージの配信およびクライアント設計に影響するその他の主要な要因は、コンシューマで必要となる信頼性の度合いです。59 ページの「信頼性の高いメッセージング」を参照してください。

同期コンシューマと非同期コンシューマ

メッセージコンシューマは、同期または非同期のどちらかのメッセージのコンシュームをサポートしています。

- 同期コンシュームとは、メッセージを配信してコンシュームするよう、コンシューマが明示的に要求することを意味します。

  メッセージの要求で使用するメソッドに応じて、同期コンシューマは、メッセージが届くまで無期限に待機するか、指定された時間だけメッセージを待機するか、またはコンシュームされる準備ができているメッセージがない場合にすぐに応答するかを選択できます。「コンシュームされる」とは、オブジェクトがすぐにクライアントで利用できることを意味します。正常に送信されてもブローカが処理を完了していないメッセージは、コンシュームされる準備ができていません。

- 非同期コンシュームでは、コンシューマとして登録されているメッセージリスナーオブジェクト (`MessageListener`) にメッセージが自動的に配信されます。セッションスレッドがメッセージリスナーオブジェクトの `onMessage()` メソッドを呼び出すと、クライアントはメッセージをコンシュームします。

セレクタを使用したメッセージのフィルタ処理

メッセージコンシューマは、メッセージセレクタを使用して、プロパティが特定の選択条件に一致するメッセージだけをメッセージサービスによって配信させることができます。コンシューマを作成するときに、この基準を指定します。

セレクタは SQL に似た構文を使用してメッセージプロパティーを照合します。たとえば次のようになります。

```
color = 'red'

size > 10
```

Java クライアントも、キューをブラウズしているときに、セレクタを指定できます。これにより、選択したメッセージのうちコンシュームされるのを待機しているメッセージを確認できます。

永続サブスクライバの使用

セッションオブジェクトを使用して、トピックへの永続サブスクライバを作成できます。ブローカは、このようなサブスクライバに対するメッセージを、このサブスクライバが非アクティブになった場合でも保持します。

ブローカは、サブスクライバの状態を保持し、サブスクライバが再度アクティブになったときにメッセージの配信を再開する必要があるので、サブスクライバの送受信全体から指定されたサブスクライバを識別できる必要があります。サブスクライバの識別情報は、サブスクライバを作成したコネクションの `ClientID` プロパティーと、サブスクライバを作成するときに指定したサブスクライバ名から構成されています。

要求 / 応答パターン

同じコネクション内で、または統合 API を使用しているときはセッション内でも、プロデューサとコンシューマを組み合わせることができます。さらに、JMS API では、一時送信先を使用することにより、メッセージング操作用の要求 / 応答パターンを実装できます。

要求 / 応答パターンを設定するために、メッセージプロデューサは次のタスクを実行する必要があります。

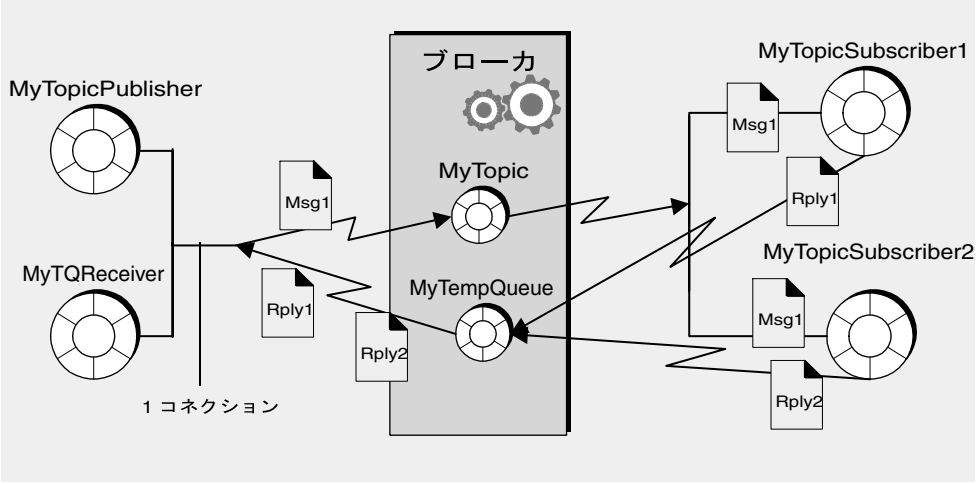
1. コンシューマが応答を送信できる一時送信先を作成する。
2. 送信するメッセージで、メッセージヘッダーの `JMSReplyTo` フィールドをその一時送信先に設定する。

メッセージコンシューマは、メッセージを処理するときに、メッセージの `JMSReplyTo` フィールドを調べ、応答が要求されているかどうかを判断し、指定された送信先へ応答を送信します。

要求応答メカニズムにより、プロデューサは、応答送信先用の管理対象オブジェクトを設定する必要がなくなり、コンシューマは、要求に対して簡単に応答できるようになります。このパターンは、プロデューサが、続行する前に要求が処理されていることを確認する必要があるときに役立ちます。

図 2-6 に、トピックへメッセージを送信し、一時キュー内の応答を受信する要求 / 応答パターンを示します。

図 2-6　要求 / 応答パターン



図に示すように、MyTopicPublisher は、Msg1 を送信先 MyTopic へプロデュースします。MyTopicSubsriber1 と MyTopicSubscriber2 はメッセージを受信し、MyTempQueue へ応答を送信します。そこから MyTQReceiver が応答を取り出します。このパターンは、たとえば、多数のクライアントへ価格情報をパブリッシュし、順次処理を行うためにその (応答) 命令をキューに入れるアプリケーションに役立ちます。

一時送信先は、一時送信先を作成したコネクションが存在する間だけしか存続しません。どのプロデューサも一時送信先に送信できますが、一時送信先にアクセスできる唯一のコンシューマは、送信先を作成したコネクションと同じコネクションによって作成されたコンシューマです。

要求 / 応答パターンは一時送信先の作成によって決まるので、次の場合にはこのパターンを使用しないでください。

- 応答が送信される前に、一時送信先を作成したコネクションが終了すると予想される場合。
- 一時送信先へ持続メッセージを送信する必要がある場合。

信頼性の高いメッセージング

メッセージング配信は 2 つのステップで行われます。最初のステップでは、プロデューサからブローカ上の物理的な送信先へメッセージが送られます。次のステップでは、その送信先からコンシューマへメッセージが送られます。このように、メッセージは、ブローカへの移動、ブローカに突然障害が発生した場合にブローカのメモリーに格納されている間、ブローカからコンシューマへの移動という 3 つのいずれかの段階で消失する可能性があります。信頼性の高い配信は、これらのどの段階でも配信が失敗しないように保証します。ブローカに障害が発生した場合、非持続メッセージは常に消失する可能性があるので、信頼性の高い配信は持続メッセージにのみ該当します。

信頼性の高い配信を確保するために、次の 2 つのメカニズムが使用されます。

- クライアントは、通知 (acknowledgement) またはトランザクション (transaction) を使用して、メッセージのプロデュースおよびコンシュームが成功したことを確認できる。
- メッセージがコンシュームされる前にブローカに障害が発生した場合に、保存されたメッセージのコピーを取得して操作を再試行できるように、ブローカは持続ストアにメッセージを保存できる。

次の節では、信頼性を確保する場合の 2 つの側面について説明します。

通知

通知は、信頼性の高いメッセージの配信を確実に行うために、クライアントとメッセージサービスの間で送信されるメッセージです。通知は、プロデューサとコンシューマとでは使用方法が異なります。

メッセージがプロデュースされる場合、ブローカは、メッセージを受信し、送信先に保管し、持続的に保存したことを通知します。プロデューサの `send()` メソッドは、通知を受信するまでブロックします。持続メッセージが送信されるとき、これらの通知はクライアントからは見えません。

メッセージがコンシュームされる場合、クライアントは、ブローカが送信先からメッセージを削除する前に、送信先から配信されたメッセージを受信し、コンシュームしたことを通知します。JMS では、違うレベルの信頼性を表す別の通知モードを規定しています。

- `AUTO_ACKNOWLEDGE` モードでは、クライアントによってコンシュームされる各メッセージについてセッションが自動的に通知します。セッションスレッドはブロックし、コンシュームされたメッセージそれぞれのクライアント通知をセッションが処理したことをブローカが確認するまで待ちます。

- CLIENT_ACKNOWLEDGE モードでは、クライアントは、1 つ以上のメッセージがコンシュームされたあとに、メッセージオブジェクトの acknowledge() メソッドを呼び出すことによって明示的に通知します。このため、セッションは、このメソッドを最後に呼び出してからセッションによってコンシュームされたすべてのメッセージに対して通知します。セッションスレッドはブロックし、クライアント通知をセッションが処理したことをブローカが確認するまで待ちます。

  Message Queue では、クライアントが 1 つのメッセージの受信しか通知できないようにするメソッドを用意することにより、このモードを拡張しています。

- DUPS_OK_ACKNOWLEDGE モードでは、10 個のメッセージがコンシュームされてからセッションが通知します。このモードではブローカ通知が不要なため、セッションスレッドはブロックしてブローカ通知を待つことはありません。このモードは、メッセージが消失されないように保証しますが、重複したメッセージが受信されないことを保証するものではありません。このため、DUPS_OK という名前が付けられています。

信頼性よりもパフォーマンスを重視するクライアントの場合、Message Queue サービスでは、NO_ACKNOWLEDGE モードを用意することにより、JMS API を拡張しています。このモードでは、ブローカはクライアント通知を追跡しないので、メッセージがコンシューミングクライアントによって正常に処理されたかどうか保証されません。このモードを選択した場合、非永続サブスクライバへ送信される非持続メッセージのパフォーマンスが向上します。

トランザクション

トランザクション とは、1 つ以上のメッセージのプロデュースまたはコンシューム、あるいはその両方を 1 つの極小の単位にグループ化する方法です。前述のクライアントとブローカの通知プロセスは、トランザクションにも適用されます。この場合、クライアントランタイムおよびブローカ通知は、トランザクションのレベルで自動的に処理されます。トランザクションがコミットされると、ブローカの通知が自動的に送信されます。

セッションは処理済みとして設定でき、JMS API には、トランザクションの開始、コミット、またはロールバックを行うメソッドが用意されています。

メッセージがトランザクション内でプロデュースまたはコンシュームされるに従って、メッセージサービスがさまざまな送受信を追跡し、JMS クライアントが呼び出しを実行してトランザクションを確定したときにだけ、送受信の操作を完了させます。トランザクション内での特定の送信や受信の操作が失敗すると、例外が発生します。クライアントコードは、これを無視するか、操作を試行し直すか、またはトランザクション全体をロールバックして、例外を処理できます。トランザクションがコミットされると、すべての操作が完了します。トランザクションがロールバックされると、正常に行われたすべての操作が取り消されます。

トランザクションの範囲は、常に単一セッションです。つまり、単一セッションのコンテキストで実行された、1 つ以上のプロデューサまたはコンシューマの操作は、単一のトランザクションにグループ化されます。トランザクションは 1 つのセッション内で行われるので、1 つの終端間トランザクションでメッセージのプロデュースとコンシュームの両方を行うことはできません。

JMS 仕様では、分散トランザクションもサポートしています。つまり、メッセージのプロデュースとコンシュームは、データベースシステムなど、ほかのリソースマネージャーに関連した操作を含む大容量の分散トランザクションの一部となります。分散トランザクションをサポートするには、Java Systems Application Server が提供するトランザクションマネージャーなどを入手する必要があります。

分散トランザクションでは、Java Transaction API (JTA)、XA Resource API の仕様で定義された 2 フェーズコミットプロトコルを使用して、メッセージサービスやデータベースマネージャーといった複数のリソースマネージャーによって実行される操作を、分散トランザクションマネージャーが追跡および管理します。Java の世界では、リソースマネージャーと分散トランザクションマネージャー間の対話は、JTA の仕様で記述されます。

分散トランザクションのサポートとは、メッセージングクライアントが、JTA で定義される `XAResource` インタフェースを介して、分散トランザクションに加わることができるということです。このインタフェースでは、2 フェーズコミットを実装するための、数多くのメソッドが定義されます。API の呼び出しがクライアント側で行われている間、JMS メッセージサービスは分散トランザクション内のさまざまな送受信操作やトランザクションの状態を追跡し、Java Transaction Service (JTS) で提供される分散トランザクションマネージャーと一致したときにだけ、メッセージング操作を完了します。

ローカルトランザクションに関しては、無視したり、操作を試行し直したり、分散トランザクション全体をロールバックしたりして、クライアントは例外を処理できます。

持続ストレージ

信頼性のもう 1 つの側面は、ブローカが持続メッセージをコンシューマに配信する前に、その持続メッセージを失うことがないようにすることです。つまり、メッセージが物理的な送信先に到達したら、ブローカはそのメッセージを持続データストア (data store) に保管する必要があります。何かの理由でブローカが停止した場合、持続データストアは後からメッセージを復元し、適切なコンシューマに配信します。

ブローカは、永続サブスクリプションも持続的に保存する必要があります。持続的に保存しないと、障害が発生した場合、ブローカは、メッセージがトピック送信先に届いたあとにアクティブになった永続サブスクライバにメッセージを配信できなくなります。

メッセージ配信を保証するメッセージングアプリケーションは、メッセージを持続的として指定し、永続サブスクリプション状態のトピック送信先またはキュー送信先のいずれかにメッセージを配信する必要があります。

第 3 章「Message Queue サービス」では、Message Queue サービスで用意されているデフォルトのメッセージストアと、管理者による代替ストアのセットアップおよび設定方法について説明します。

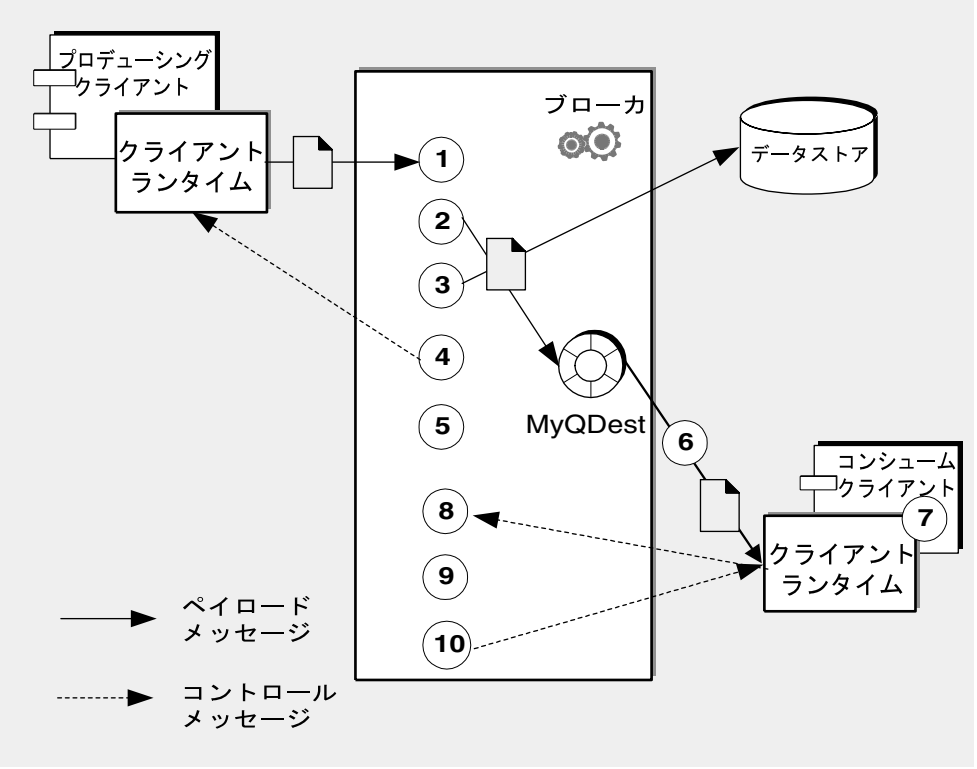
システム全体でのメッセージの流れ

これまで示してきた説明のまとめとして、ここでは、Message Queue サービスで、どのようにメッセージがプロデューサからコンシューマへ配信されているかついて説明します。全体像を理解するため、さらに説明する必要があります。配信の過程でシステムが処理するメッセージは、次の 2 つのカテゴリに分類されます。

- **ペイロードメッセージ**。これは、プロデューサがコンシューマへ送信するメッセージです。
- **コントロールメッセージ**。これは、ブローカとクライアントランタイム間で送られるプライベートメッセージであり、ペイロードメッセージの正常な配信と、コネクション全体のメッセージのフロー制御が確実に行われるようにします。

図 2-7 に、メッセージ配信を示します。

図 2-7　メッセージ配信手順



信頼性の高い方法で、持続的に配信されるメッセージのメッセージ配信手順は次のとおりです。

メッセージのプロデュース

1. クライアントランタイムが、コネクションを使用して、メッセージプロデューサからブローカにメッセージを配信します。

メッセージの処理とルーティング

2. ブローカが、コネクションからメッセージを読み込み、適切な送信先に保管します。

3. ブローカが (持続) メッセージをデータストアに保管します。

4. ブローカが、メッセージを受信したことについて、メッセージプロデューサのクライアントランタイムに通知します。

5. ブローカが、メッセージのルーティングを決定します。

6. ブローカは、コンシューマの一意の識別子をメッセージに付けて、メッセージをその送信先から適切なコネクションへ書き出します。

メッセージのコンシューム

7. メッセージコンシューマのクライアントランタイムが、コネクションからメッセージコンシューマにメッセージを配信します。

8. メッセージコンシューマのクライアントランタイムが、メッセージをコンシュームしたことについて、ブローカに通知します。

メッセージの存続終了

9. ブローカはクライアント通知を処理し、すべての通知を受信したときに、(持続) メッセージを削除します。

10. ブローカは、コンシューマのクライアントランタイムに対して、クライアント通知が処理されたかどうかを確認します。

管理者がメッセージを送信先から削除した場合、または管理者が永続サブスクリプションを削除または再定義し、その結果、トピック送信先内のメッセージが配信されずに削除された場合、ブローカはメッセージがコンシュームされる前にメッセージを破棄できます。その他の状況では、ブローカにメッセージを破棄させるのではなく、デッドメッセージキューと呼ばれる特殊な送信先に保存させることができます。メッセージの有効期限が切れたとき、メモリーの制限のために削除されたとき、またはクライアントが例外をスローしたために配信が失敗したとき、メッセージはデッドメッセージキューに保管されます。メッセージをデッドメッセージキューに保存しておくと、システムの問題を解決し、特定の状況でメッセージを復元することができます。

SOAP メッセージの処理

SOAP (35 ページの「Java クライアントでの SOAP サポート」を参照) を使用すると、分散環境にある 2 つのピア間で構造化データ (XML 方式で指定) を交換できます。Sun の SOAP の実装では、現在、信頼性の高い SOAP メッセージングをサポートしておらず、SOAP メッセージのパブリッシングもサポートしていません。ただし、Message Queue サービスを使用して、信頼性の高い SOAP メッセージングを取得し、必要な場合は、SOAP メッセージをパブリッシュできます。Message Queue サービスは、直接 SOAP メッセージを配信しませんが、SOAP メッセージを JMS メッセージにラップし、通常の JMS メッセージのようにこれらのメッセージをプロデュースおよびコンシュームして、JMS メッセージから SOAP メッセージを抽出できます。

Message Queue では、`javax.xml.messaging` と `com.sun.messaging.xml` の 2 つのパッケージを使用して、SOAP サポートを提供します。これらのライブラリに実装されたクラスを使用して、SOAP メッセージを受信し、SOAP メッセージを JMS メッセージにラップし、JMS メッセージから SOAP メッセージを抽出できます。J2EE プラットフォームには、パッケージ `java.xml.soap` が用意されており、このパッケージを使用して、SOAP メッセージをアセンブルまたは逆アセンブルできます。

信頼性の高い SOAP メッセージングを取得するには、次の手順を行う必要があります。

1. `java.xml.soap` パッケージで定義されたオブジェクトを使用して SOAP メッセージを作成するか、`javax.xml.messaging` パッケージで定義されたサーブレットを使用して SOAP メッセージを受信するか、または JAX-RPC などの Web サービスを使用して SOAP メッセージを受信します。
2. MessageTransformer ユーティリティーを使用して、SOAP メッセージを JMS メッセージに変換します。
3. 目的の送信先に JMS メッセージを送信します。
4. JMS メッセージを非同期的または同期的にコンシュームします。
5. JMS メッセージをコンシュームしたあと、MessageTransformer ユーティリティーを使用して、SOAP メッセージに変換します。
6. SAAJ API (`java.xml.soap` パッケージで定義) を使用して、SOAP メッセージを逆アセンブルします。

SOAP メッセージとその処理に関する詳細な情報については、『Message Queue Developer's Guide for Java Clients』を参照してください。

Java クライアントと C クライアント

Message Queue では、そのメッセージングサービス用の C API を備えているため、旧バージョンの C アプリケーションと C++ アプリケーションを、JMS ベースのメッセージングに加えることができます。

JMS プログラミングモデルは、Message Queue C クライアントの設計の基盤です。『Message Queue Developer's Guide for C Clients』では、このモデルが C データタイプや関数によってどのように実装されているかについて説明しています。

Java インタフェースと同様に、C インタフェースは次の機能をサポートします。

- パブリッシュ / サブスクライブコネクションとポイントツーポイントコネクション
- 同期コンシューマと非同期コンシューマ
- `CLIENT`、`AUTO`、および `DUPS_OK` 通知モード
- ローカルトランザクション
- セッションの復元
- 一時トピックおよびキュー
- メッセージセレクタ

ただし、Java Message Service 仕様は、*Java* クライアントに限定した標準であることに注意する必要があります。したがって、C Message Queue API は、Message Queue プロバイダ固有の機能であり、他の JMS プロバイダでは使用できません。C クライアントを含んだメッセージングアプリケーションは、別の JMS プロバイダでは処理できません。

C インタフェースでは、次の機能がサポートされません。

- 管理対象オブジェクトの使用
- マップ、ストリーム、またはオブジェクトメッセージのタイプ
- コンシューマベースのフロー制御
- キューブラウザ
- JMS アプリケーションサーバー機能 (コネクションコンシューマ、分散トランザクション)
- SOAP メッセージの受信または送信
- 圧縮された JMS メッセージの受信または送信
- 自動再接続またはフェイルオーバー。コネクションに障害が発生した場合、クライアントランタイムが自動的にブローカへ再接続できるようにします
- `NO_ACKNOWLEDGE` モード

第 3 章

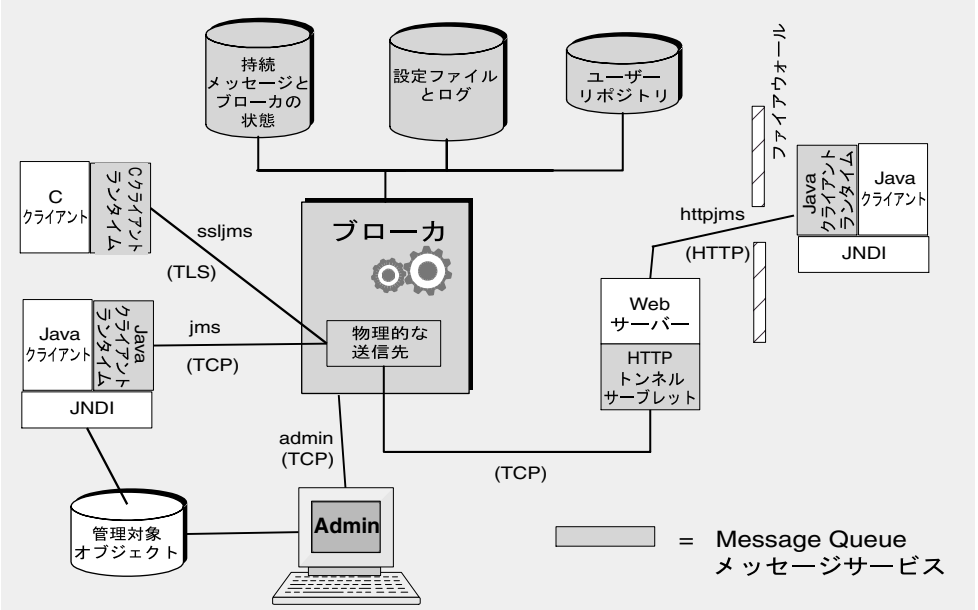
Message Queue サービス

Message Queue クライアントのパフォーマンスは、クライアントの設計と、Message Queue サービスの設定および管理方法によって異なります。この章では、第 1 章で紹介した Message Queue サービスについて詳しく説明します。サービスのコンポーネント、これらのコンポーネントを設定するために使用するツールの概要、さまざまな環境でメッセージサービスを管理するために必要なタスクの概要について説明します。この章は以下の節で構成されています。

コンポーネントサービス

図 3-1 に Message Queue サービスを示します。第 2 章では、プログラミングモデルの概要と、クライアントが Java および C API を使用して、クライアントアプリケーションからアクセス可能なメッセージサービスの一部であるクライアントランタイムと対話する方法について説明します。この章では、管理者がアクセス可能なメッセージサービスのコンポーネントおよびサービスを中心に説明します。

図 3-1　Message Queue サービス



Message Queue サービスは、ブローカのプロパティーを設定することによって管理します。これらのプロパティーは、特定のプロパティーによって影響を受けるサービスまたはブローカコンポーネントに応じていくつかのカテゴリに分類されます。ブローカサービスには次のものがあります。

- **コネクションサービス**: ブローカとクライアント間の物理的な接続を管理し、送受信メッセージの転送を行う。
- **ルーティングサービス**: JMS メッセージ、およびメッセージサービスによって使用されるコントロールメッセージのルーティングと配信を行い、信頼性の高い配信をサポートする。
- **持続サービス**: 持続ストレージへのデータの書き込みおよび接続ストレージからのデータの取得を管理する。
- **セキュリティーサービス**: ブローカに接続するユーザーを認証し、ユーザーの操作を承認する。
- **監視サービス**: メトリックスと診断情報を生成し、この情報を指定された出力チャネルに書き込む。

以降の節では、これらの各サービスと、特定のニーズに合わせてサービスをカスタマイズするために使用するプロパティーの概要について説明します。

ブローカのプロパティーは、それぞれの設定ファイル内で定義されますが、ブローカを起動するために使用するコマンド行で定義することもできます。『Message Queue 管理ガイド』では、これらの設定ファイルと、1 ファイル内のプロパティー値を使用して別のファイル内で設定された値をオーバーライドするための優先順位について説明しています。起動コマンドを使用して設定されたプロパティーはほかのすべての設定をオーバーライドします。

コネクションサービス

コネクション関連のプロパティーを使用して、ブローカとそのクライアントの間の物理的な接続を設定および管理します。Message Queue クライアントで使用可能なコネクションサービスについては、32 ページの「ブローカへの接続」で説明しています。この項では、使用可能なコネクションサービス、サービス名、サービスタイプ、および基礎となるプロトコルについて説明しています。コネクションサービスは、マルチスレッド化されており、専用のポート経由で使用できます。このポートは、ブローカのポートマッパーによって動的に割り当てることも、管理者が静的に割り当てることもできます。デフォルトでは、ブローカを起動すると、`jms` サービスと `admin` サービスが稼働します。

すべてのコネクションには 2 つの端があるため、両側でコネクションの設定を行い、調整する必要があります。

- クライアントは、コネクションファクトリオブジェクトの特定の属性を設定して、デフォルト以外のコネクションサービス、ホスト、およびポートの要求、異なるブローカへのコネクションが必要な場合に再接続するためのブローカのリストの指定、および再接続動作の設定を行う必要があります。クライアントはまた、失敗したコネクションをテストするための ping 間隔を指定することもできます。
- 管理者は、ブローカのプロパティーを使用して、デフォルト以外のコネクションサービスの有効化、必要に応じた静的なポートの割り当て、スレッドの設定、および複数のネットワークカードが使用された場合の接続先のホストの指定を行います。管理者はまた、クライアントがアクセス可能かどうかをテストするための ping 間隔を指定することもできます。これはリソースの管理に役立ちます。

クライアントは、ファイアウォール経由で Message Queue サービスに接続できます。これは、ファイアウォール管理者に特定のポートを開いてもらってからその (静的) ポートに接続するか、109 ページの「HTTP コネクション」で説明しているように、HTTP サービスまたは HTTPS サービスを使用して実現できます。

各コネクションサービスは、特定の認証および承認機能もサポートします。詳細は、75 ページの「セキュリティーサービス」を参照してください。

ポートマッパー

コネクションサービスには、ブローカのメインポート 7676 に常駐する共通ポートマッパーによりポートが割り当てられます。Message Queue クライアントランタイムは、ブローカとの間でコネクションを設定する場合、最初にポートマッパーに接続し、選択したコネクションサービスのポート番号を要求します。

ポートマッパーは、`jms`、`ssljms`、`admin`、および `ssladmin` の各サービスの設定時に、静的なポート番号を割り当てることによってオーバーライドできます。ただし、静的ポートは通常、ファイアウォールを通してコネクションを確立する場合のように特殊な状況でのみ使用され、一般的にはお勧めしません。

スレッドプール管理

各コネクションサービスは、複数のコネクションをサポートするマルチスレッドです。これらのコネクションに必要なスレッドは、プール内のブローカによって保持されます。スレッドの割り当て方法は、スレッドの最小数および最大数に指定した値、および選択したスレッドモデルによって決まります。

ブローカのプロパティーを設定してスレッドの最小数および最大数を指定できます。コネクションでスレッドが必要になると、コネクションをサポートするサービスのスレッドプールにスレッドが追加されます。最小数では、割り当てに使用できるスレッドの数を指定します。使用可能なスレッドの数がこの最小しきい値を超えると、システムはスレッドをシャットダウンします。スレッドは最小値に再び達するまで解放されるため、メモリーリソースが節約されます。負荷が大きい場合、プールの最大数に達するまでスレッドの数が増加する可能性があります。最大数に達すると、スレッドが使用可能になるまで新しい接続は拒否されます。

選択したスレッドモデルにより、スレッドが 1 つのコネクション専用か複数のコネクションで共有されるかが指定されます。

- 専用モデルでは、ブローカへのコネクションごとに 2 つのスレッドが必要になります。1 つは受信メッセージ用で、1 つは送信メッセージ用です。これにより、使用可能なコネクション数が制限されますがパフォーマンスが向上します。
- 共有モデルでは、メッセージを送受信するときに共有スレッドによってコネクションが処理されます。各コネクションに専用のスレッドが必要ないので、このモデルでは使用可能なコネクションの数が増加しますが、スレッド管理のオーバーヘッドが増えるためにパフォーマンスに影響します。

送信先とルーティングサービス

いったんクライアントがブローカに接続されると、メッセージのルーティングと配信を進めることができます。この段階では、メッセージの円滑な流れを確保し、リソースを効率的に使用するために、ブローカがさまざまなタイプの物理的な送信先を作成および管理します。ブローカは、ルーティングと配信に関連するブローカのプロパティーを使用して、アプリケーションのニーズに合った方法でこれらのタスクを管理します。

ブローカ上の物理的な送信先、つまりメッセージコンシューマに配信される前にメッセージが保存されるメモリーの場所の概念についてはすでに説明しました。次の 4 つのタイプの物理的な送信先があります。

- **管理者作成の送信先**は、GUI (`imqadmin`) または `imqcmd` ユーティリティーを使用して管理者が作成します。この送信先は、プログラムによって作成される論理的な送信先か、管理者が作成しクライアントによって検索される送信先管理対象オブジェクトのどちらかに対応します。管理者作成の各送信先のプロパティーを設定または更新するには、`imqcmd` ユーティリティーを使用します。
- **自動作成の送信先**は、メッセージのコンシューマまたはプロデューサが、存在しない送信先にアクセスしようとするたびにブローカによって自動的に作成されます。これらは一般的に開発中に使用されます。このような送信先の作成を許可しないようにブローカのプロパティーを設定できます。特定のブローカ上のすべての自動作成の送信先を設定するには、ブローカのプロパティーを設定します。

  自動作成された送信先は、使用されなくなると、つまり、コンシューマクライアントを保持せず、メッセージを一切含まない状態になると、ブローカによって自動的に破棄されます。ブローカを再起動して、持続メッセージが含まれている場合にだけ、自動作成の送信先が再作成されます。
- **一時送信先**は、メッセージに対する応答を受信するために送信先が必要な場合に、クライアントがプログラムによって明示的に作成および破棄します。名前でわかるように、これらの送信先は、一時的であり、送信先が作成されたコネクションの存続期間中にだけ、ブローカによって保持されます。

  一時送信先は、持続的には保持されず、ブローカの再起動時に作成し直されることもありません。ただし、管理ツールからは認識できます。
- **デッドメッセージキュー**は、ブローカの起動時に自動的に作成される特殊な送信先で、診断用のデッドメッセージを保管するために使用します。デッドメッセージキューのプロパティーを設定するには、`imqcmd` ユーティリティーを使用します。

送信先の管理

送信先を管理するには、`imqcmd` ユーティリティーを使用します。送信先の管理では、次の 1 つ以上のタスクを実行します。

- 送信先の作成、一時停止、再開、または破棄。

- ブローカ上のすべての送信先の一覧表示。
- 送信先の状態とプロパティーに関する情報の表示。
- 送信先のメトリックス情報の表示。
- 送信先用のメッセージを保持するために使用されるディスクスペースの圧縮。
- 物理的な送信先のプロパティーの更新。

管理タスクは、管理される送信先のタイプ (管理者作成、自動作成、一時、デッドメッセージキュー) によって異なります。たとえば、一時送信先は明示的に破棄する必要がありません。また自動作成されるプロパティーは、ブローカの設定プロパティーを使用して設定され、そのブローカ上のすべての自動作成される送信先に適用されます。

物理的な送信先の設定

最適なパフォーマンスを得るために、物理的な送信先を作成または更新するときにプロパティーを設定することができます。次のようなプロパティーを設定することができます。

- 送信先のタイプと名前。
- 送信先の個別の制限および全体の制限 (メッセージの最大数、最大合計バイト数、メッセージあたりの最大バイト数、プロデューサの最大数)。
- 個別の制限または全体の制限を超えた場合にブローカが実行する処理。
- 1 回のバッチで配信されるメッセージの最大数。
- 送信先用のデッドメッセージをデッドメッセージキューに送信するかどうか。
- クラスタ化されたブローカで送信先をクラスタ内の他のブローカに複製するかどうか。

また、キュー送信先に対してバックアップコンシューマの最大数を設定したり、クラスタ化されたブローカに対してローカルキューへの送信を優先するかどうかを指定したりできます。

デッドメッセージキューの制限および動作を設定することもできます。ただし、このキューのデフォルトのプロパティーは、標準のキューのプロパティーとは異なっています。

メモリーの管理

送信先は、送信先が処理するメッセージの数とサイズ、および登録するコンシューマの数と永続性によっては、リソースを著しく消費する可能性があるので、メッセージサービスのパフォーマンスと信頼性を確保するために、送信先を綿密に管理する必要があります。

プロパティーを設定することで、ブローカが受信メッセージのために過負荷状態になったり、ブローカのメモリーが不足したりすることを防ぐことができます。ブローカは、送信先の制限、システム全体の制限、およびシステムメモリーのしきい値という 3 つのメモリー保護レベルを使用して、リソースが少なくなったときにもメッセージサービスの動作を維持します。送信先の制限とシステム全体の制限が適切に設定されている場合、重要なシステムメモリーのしきい値を超えないようにするのが理想的です。

送信先のメッセージ制限

送信先の属性を設定して、送信先ごとにメモリーとメッセージフローを管理することができます。たとえば、送信先で許容されるプロデューサの最大数、送信先で許容されるメッセージの最大数 (または、サイズ)、および任意のメッセージの最大サイズを指定できます。

また、これらの制限に達した場合のブローカの対応方法 (プロデューサの動作を遅くする、最も古いメッセージを破棄する、優先度がもっとも低いメッセージを破棄する、最新のメッセージを拒否する) を指定できます。

システム全体のメッセージ制限

プロパティーを使用してブローカ上のすべての送信先に適用される制限を設定することもできます。メッセージの総数とすべてのメッセージによって消費される総メモリー量を指定できます。どちらかのシステム全体のメッセージ制限に達した場合、ブローカは新しいメッセージを拒否します。

システムメモリーのしきい値

最後に、プロパティーを使用して、ブローカが段階的に深刻なアクションを実行するしきい値を設定し、メモリーの過負荷を避けるようにすることができます。実行するアクションは、`green` (使用可能なメモリーが十分にある)、`yellow` (ブローカのメモリーが減っている)、`orange` (ブローカのメモリーが不十分である)、`red` (ブローカのメモリーが不足している) といったメモリーリソースの状態によって異なります。ブローカのメモリーの状態が `green` から `red` へと進むにつれ、ブローカは次のタイプの深刻なアクションを段階的に実行します。

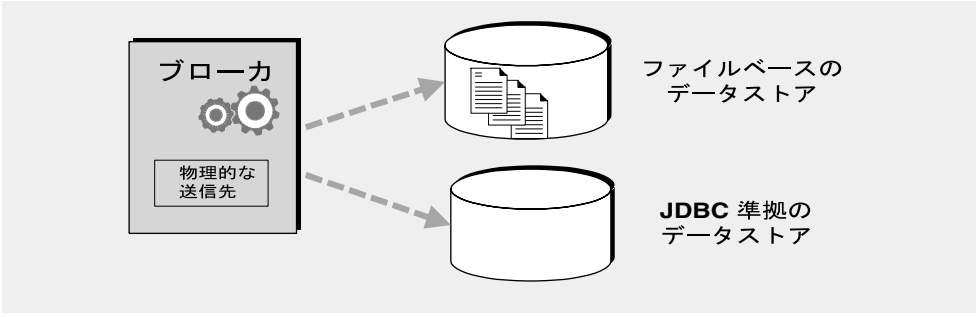
- データストアの持続メッセージのメモリー内コピーを廃棄する。
- 非持続メッセージのプロデューサを減らし、最終的にブローカへのメッセージのフローを停止する。持続メッセージのフローは、各メッセージがブローカによって通知される要件によって自動的に制限される。

持続サービス

障害が発生したブローカを復元するには、メッセージの配信処理の状態を作成し直す必要があります。そのためには、状態情報をデータストアに保存する必要があります。ブローカが再起動されると、保存されているデータを使用して送信先および永続サブスクリプションを再作成し、持続メッセージを復元し、開いているトランザクションをロールバックし、未配信メッセージのルーティングテーブルを再作成します。その後ブローカは、メッセージの配信を再開します。

Message Queue サービスは、ファイルベースの持続モジュールと JDBC 準拠の持続モジュールの両方をサポートしていますが (図 3-2 を参照)、デフォルトではファイルベースの持続を使用します。

図 3-2　持続サポート



ファイルベースの持続

ファイルベースの持続は、個々のファイルを使用して持続データを保存するメカニズムです。ファイルベースの持続を使用する場合は、次のタスクを実行するようにブローカのプロパティーを設定できます。

- メッセージが追加および削除されたときの断片化を減らすために、データストアを圧縮する。
- 書き込みのたびにメモリー内の状態と物理的なストレージデバイスとを同期する。この同期化により、システム破壊によるデータの損失をなくすことができます。
- データストアファイルへのメッセージの割り当てを管理し、ファイルの管理および保存に必要なリソースを管理する。

通常、ファイルベースの持続の方が、JDBC ベースの持続より処理速度が速くなりますが、JDBC 準拠のストアによる冗長性および管理制御を好むユーザーもいます。

JDBC ベースの持続

JDBC ベースの持続は、JDBC™ (Java Database Connectivity) インタフェースを使用して、ブローカと JDBC 準拠のデータストアを接続します。ブローカが JDBC ドライバを介してデータストアにアクセスできるようにするには、次のことを実行する必要があります。

- JDBC 関連ブローカ設定プロパティーを設定します。これらのプロパティーを使用して、使用する JDBC ドライバの指定、JDBC ユーザーとしてのブローカの認証、必要なテーブルの作成などを行います。
- `imqdbmgr` ユーティリティーを使用して適切なスキーマでデータストアを作成します。

これらのタスクの実行手順および関連する設定プロパティーについては、『Message Queue 管理ガイド』に記載されています。

セキュリティーサービス

Message Queue サービスは、各ブローカインスタンス用の認証および承認 (アクセス制御) をサポートし、暗号化もサポートしています。

- 認証により、検証されたユーザーだけがブローカとのコネクションを確立できるようにします。
- 承認により、リソースにアクセスし、特定の操作を実行する権限を持つユーザーまたはグループを指定します。
- 暗号化により、コネクションを通して配信されるときに改ざんされないよう、メッセージを保護します。

認証機能と承認機能は、メッセージングシステムのユーザーに関する情報 (名前、パスワード、グループ (group) メンバーシップなど) を含むリポジトリによって異なります。また、ユーザーまたはグループによる特定の操作を承認するには、ユーザーまたはグループが実行できる操作を指定するアクセス制御プロパティーファイルをブローカがチェックする必要があります。ブローカがユーザーを認証してユーザーの操作を承認するために必要な情報は、管理者が設定する必要があります。

図 3-3 は、ブローカが認証と承認を与えるために必要とするコンポーネントを示しています。

図 3-3　セキュリティーマネージャーのサポート

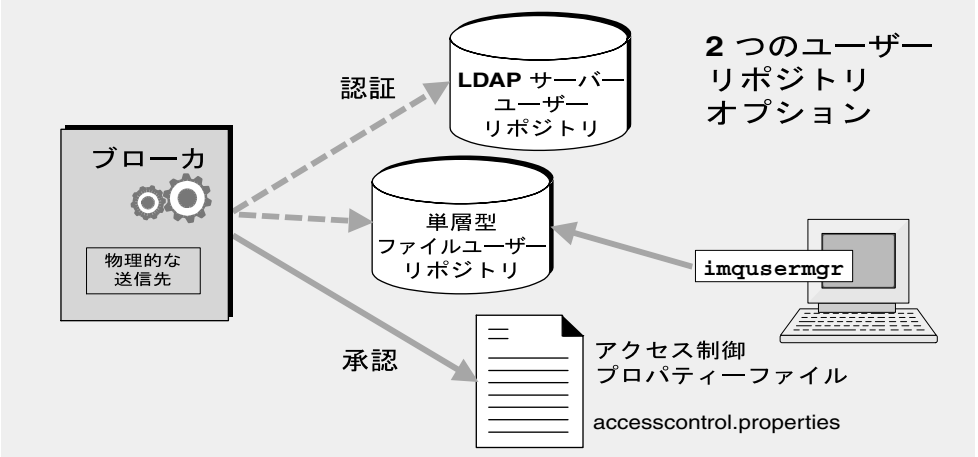


図 3-3 に示すように、Message Queue サービスに付属している単層型ファイルユーザーリポジトリにユーザーデータを保存するか、既存の LDAP リポジトリにプラグインすることができます。ブローカのプロパティーを設定して、自分の選択を指定します。

- 単層型ファイルリポジトリを選択する場合は、`imqusermgr` ユーティリティーを使用してリポジトリを管理する必要があります。これは、使いやすい組み込みユーティリティーです。
- 既存の LDAP サーバーを使用する場合は、LDAP ベンダーによって提供されているツールを使用して、ユーザーリポジトリの値の入力と管理を行います。また、ブローカが LDAP サーバーに対してユーザーおよびグループに関する情報を照会できるように、ブローカインスタンス設定ファイル内でプロパティーを設定する必要があります。

  スケーラビリティーが重要な場合やさまざまなブローカでリポジトリを共有する必要がある場合は、LDAP オプションの方が適しています。ブローカクラスタを使用する場合などが、これに該当します。

認証と承認

クライアントがコネクションを要求する場合、ユーザー名とパスワードを入力する必要があります。ブローカは、指定されたユーザー名とパスワードをユーザーリポジトリ内に格納されているものと比較します。クライアントからブローカにパスワードが送信される場合、パスワードは、Base64 か、メッセージダイジェスト (MD5) ハッシュのどちらかを使用して暗号化されます。単層型ファイルリポジトリでは MD5 が

使用され、LDAP リポジトリでは Base64 が必要です。LDAP を使用する場合は、セキュリティー保護された TLS プロトコルを使用する方がよいでしょう。ブローカのプロパティーを設定して各コネクションサービスで使用する暗号化のタイプを 1 つずつ設定するか、あるいはブローカ単位で暗号化を設定することができます。

ユーザーが何らかの操作を実行しようとすると、ブローカは、ユーザーリポジトリ内のユーザー名とグループのメンバーシップを、アクセス制御プロパティーファイル内にある、その操作へのアクセス権が与えられているユーザー名とグループのメンバーシップと照らし合わせます。アクセス制御プロパティーファイルでは、次の操作に対するユーザーまたはグループのアクセス権を指定します。

- ブローカへの接続
- 送信先へのアクセス : 特定の送信先、またはすべての送信先に対してのコンシューマ、プロデューサ、またはキューブラウザの作成
- 送信先の自動作成

ブローカのプロパティーを設定して、次の情報を指定します。

- アクセス制御が有効かどうか
- アクセス制御ファイルの名前
- パスワードの暗号化方法
- ブローカからの認証要求に対するクライアントの応答をシステムが待機する時間
- セキュリティー保護された接続に必要な情報

暗号化

クライアントとブローカ間で送信されるメッセージを暗号化するには、SSL (Secure Socket Layer) 標準に基づいたコネクションサービスを使用する必要があります。SSL は、SSL 対応のブローカとクライアント間で暗号化されたコネクションを確立して、コネクションレベルのセキュリティーを提供します。

ブローカのプロパティーを設定して、使用する SSL キーストアのセキュリティープロパティーおよびパスワードファイルの名前と場所を指定できます。

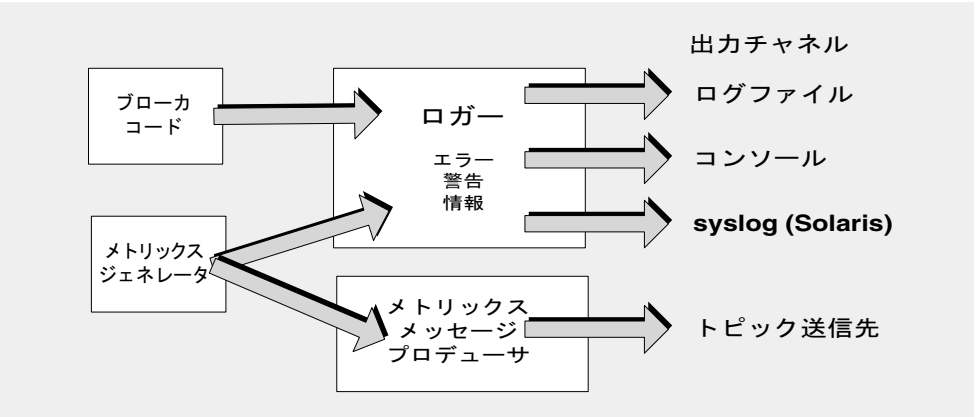
監視サービス

ブローカにはアプリケーションとブローカのパフォーマンスを監視および診断するコンポーネントが含まれています。たとえば、次のようなコンポーネントが含まれています。

- データを生成するコンポーネント (イベントを記録するメトリックスジェネレータとブローカコード)。
- 多数の出力チャネルに対して情報を書き込むロガーコンポーネント。
- メトリックス情報を含む JMS メッセージを、JMS 監視クライアントによってコンシュームさせるためにトピック送信先へ送るメッセージプロデューサ。

この仕組みの概略を、図 3-4 に示します。

図 3-4　監視サービスのサポート



メトリックスジェネレータ

メトリックスジェネレータは、ブローカとの間で入出力されるメッセージフロー、ブローカメモリー内のメッセージ数とそれらが消費するメモリー量、開かれているコネクションの数、使用中のスレッドの数など、ブローカの動作に関する情報を提供します。

ブローカのプロパティーを設定して、メトリックスデータの生成をオン、またはオフにすることも、メトリックスレポートを生成する頻度を指定することもできます。

ロガー

Message Queue のロガーは、ブローカコードとメトリックスジェネレータによって生成された情報を取得し、標準出力 (コンソール)、ログファイル、および Solaris™ プラットフォームではエラーの場合に `syslog` デーモンプロセスなどにそれらの情報を書き込みます。

ブローカのプロパティーを設定して、ロガーが収集する情報のタイプと、各出力チャネルに書き込む情報のタイプを指定できます。ログファイルに出力する場合、ログファイルを閉じて新しいファイルに出力がロールオーバーされる時点を指定できます。ログファイルが指定したサイズや有効期間に達すると、そのファイルは保存されて、新しいログファイルが作成されます。

ロガーの設定方法およびロガーによるパフォーマンス情報の入手方法についての詳細は、『Message Queue 管理ガイド』を参照してください。

メトリックスメッセージプロデューサ (Enterprise Edition)

図 3-4 に示すメトリックスメッセージプロデューサは、定期的にメトリックスジェネレータから情報を受け取り、その情報をメッセージに書き込みます。その後、そのメッセージは、メッセージに含まれるメトリックス情報のタイプに応じて、多数あるメトリックストピック送信先の 1 つに送信されます。

これらのメトリックストピック送信先にサブスクライブされた Message Queue クライアントは、メッセージをコンシュームし、メッセージに含まれるメトリックスデータを処理できます。これにより開発者は、カスタム監視ツールを作成してメッセージングアプリケーションをサポートできます。各タイプのメトリックスメッセージで報告されるメトリックスの数量についての詳細は、『Message Queue Developer's Guide for Java Clients』を参照してください。メトリックスメッセージのプロデュースの設定方法に関する詳細は、『Message Queue 管理ガイド』を参照してください。

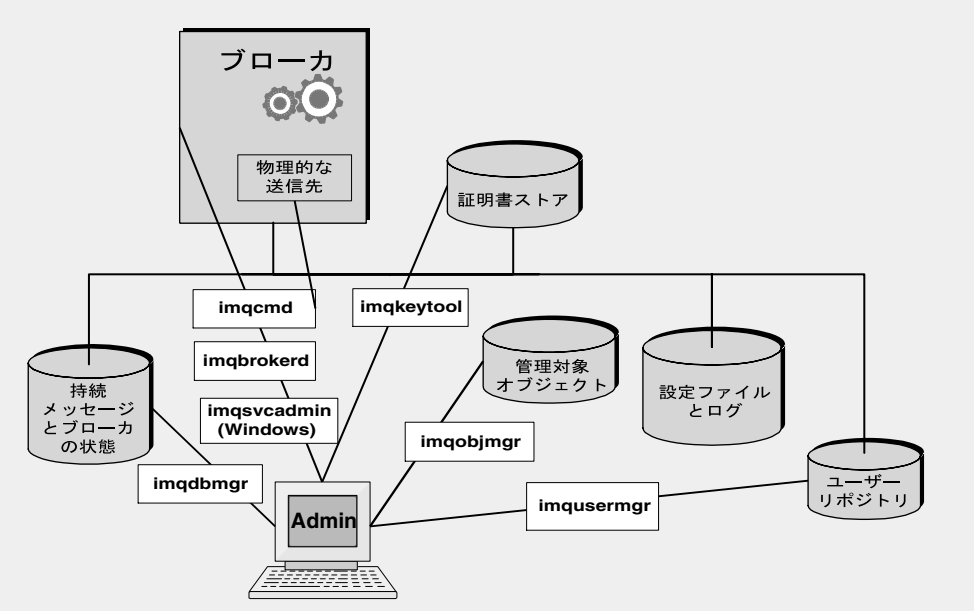
管理ツールとタスク

ここでは、Message Queue サービスを設定するために使用するツール、および開発環境または本稼働環境をサポートするために実行する必要があるタスクについて説明します。

管理ツール

図 3-5 に、クライアントコネクションを除外したメッセージサービスの概要を示します。ここでは、ブローカコンポーネントとそれらを管理するために使用されるツールに焦点を当てています。

図 3-5　管理ツール



次のコマンド行ツールを使用して、Message Queue サービスを設定および管理することができます。

- `imqbrokerd` ユーティリティーを使用して、ブローカを起動します。`imqbrokerd` コマンドのオプションを使用して、クラスタ内でブローカを接続するかどうか、および追加の起動設定情報を指定することができます。
- ブローカを起動したあとで、`imqcmd` ユーティリティーを使用して、物理的な送信先を作成、更新、および削除します。そのようにして、ブローカとそのコネクションサービスを制御したり、ブローカのリソースを管理したりします。

- imqobjmgr ユーティリティーを使用して、JNDI オブジェクトストア内の管理対象オブジェクトの追加、一覧表示、更新、および削除を行います。
- imqusermgr ユーティリティーを使用して、ユーザーの認証および承認のためのファイルベースのユーザーリポジトリに値を入力します。
- imqdbmgr ユーティリティーを使用して、持続ストレージで使用される JDBC 準拠のデータベースを作成および管理します。組み込みファイルストアは外部管理を必要としません。
- imqkeytool ユーティリティーを使用して、SSL 認証で使用される自己署名付き証明書を作成します。
- imqsvcadmin ユーティリティーを使用して、ブローカを Windows サービスとしてインストール、照会、および削除します。

GUI ベースの管理コンソールは、imqcmd ユーティリティーと imqobjmgr ユーティリティーの一部の機能を組み合わせたものです。この管理コンソールを使用して次のことができます。

- ブローカへの接続および管理。
- 物理的な送信先の作成および管理。
- オブジェクトストアへの接続、ストアへのオブジェクトの追加、およびそれらの管理。

開発環境のサポート

クライアントコンポーネントの開発では、管理作業を最小限に抑えるのが最適です。Message Queue 製品は、この目的のために設計され、出荷状態で使用できます。ブローカを起動するだけで十分です。次の手法を使用して、開発に集中することができます。

- データストア (組み込みのファイル持続)、ユーザーリポジトリ (ファイルベース)、アクセス制御プロパティーファイルのデフォルトの実装を使用します。開発テストを行うには、これらで十分です。デフォルトのユーザーリポジトリは、デフォルトエントリと一緒に作成され、インストール後すぐにブローカを使用できるようになります。デフォルトのユーザー名 (guest) とパスワード (guest) を使用して、クライアントを認証できます。
- 目的に合ったディレクトリを作成することにより、単純なファイルシステムオブジェクトストアを使用してそこに管理対象オブジェクトを保管します。また、ストアを一切作成しない場合は、管理対象オブジェクトをコードで直接インスタンス化することができます。
- 物理的な送信先をブローカ上で明示的に作成せずに、自動作成された物理的な送信先を使用します。詳細は、適切な開発者ガイドを参照してください。

本稼働環境のサポート

本稼働環境では、アプリケーションのパフォーマンスを向上させ、スケーラビリティー、可用性、およびセキュリティーに関する企業の要件を達成するためにメッセージサービス管理が重要な役割を果たします。この環境では、管理者が実行するタスクが数多くあります。これらのタスクはセットアップ操作とメンテナンス操作に大きく分類できます。

セットアップ操作

一般的に、次のセットアップ操作を行う必要があります。

- セキュリティー保護された管理アクセス

  ファイルベースまたはLDAPのどちらのユーザーリポジトリを使用する場合でも、管理者が`admin`グループに所属し、セキュリティー保護されたパスワードを持っていることを確認します。必要に応じて、管理者用にセキュリティー保護されたブローカへのコネクションを作成します。

- セキュリティー保護されたクライアントアクセス

  ファイルベースまたはLDAPのどちらのユーザーリポジトリを使用する場合でも、メッセージサービスにアクセスできるユーザーの名前をユーザーリポジトリに入力し、アクセス制御プロパティーファイルを編集して、それらのユーザーに適切な承認を与えます。必要に応じて、SSLベースのコネクションサービスを設定します。認証されていないコネクションを防止するために、「guest」ユーザーのパスワードを変更します。

- 物理的な送信先の作成および管理

  メッセージ数とメッセージに割り当てられるメモリー量をブローカリソースでサポートできるように送信先属性を設定します。

- 管理対象オブジェクトの作成および設定

  LDAPオブジェクトストアを使用する場合は、ストアを設定します。コネクションファクトリオブジェクトと送信先管理対象オブジェクトの作成および設定

- ステートフルな水平方向の拡張が必要な場合のブローカクラスタの作成

  中央設定ファイルを作成し、マスターブローカを指定します。

メンテナンス操作

ブローカリソースを監視および制御し、アプリケーションのパフォーマンスを調整するには、アプリケーションを配備したあとで次のタスクを実行する必要があります。

- アプリケーションクライアントのサポートと管理
  - 送信先、永続サブスクリプション、およびトランザクションを監視および管理する。
  - 自動作成機能を無効にする
  - デッドメッセージキューを監視および管理する
- ブローカの監視と調整
  - 障害の発生したブローカを復元する
  - ブローカを監視、調整、および再設定する。
  - ブローカのメモリーリソースを管理する。
  - 必要な場合にクラスタを拡大する。
- 管理対象オブジェクトの管理

  必要に応じて追加の管理対象オブジェクトを作成し、パフォーマンスとスループットを向上させるようにコネクションファクトリーの属性を調整します。

Message Queue サービスの拡張

ブローカを接続し、状態情報をそれらで共有できるようにすることで、Message Queue サービスを水平方向に拡張できます。これにより、任意のブローカがリモートの送信先にアクセスし、より多くのクライアントをサポートできるようになります。詳細は、第 4 章「ブローカクラスタ」を参照してください。

第 4 章

ブローカクラスタ

Message Queue Enterprise Edition は、連携動作してクライアントへのメッセージ配信サービスを提供する、ブローカクラスタをサポートしています。クラスタにより、管理者は、複数のブローカ間でクライアントコネクションを分散させ、メッセージトラフィックの量に応じてメッセージングの動作を拡張または縮小することができます。

この章では、ブローカクラスタのアーキテクチャーと内部の仕組みについて説明します。次のトピックが含まれます。

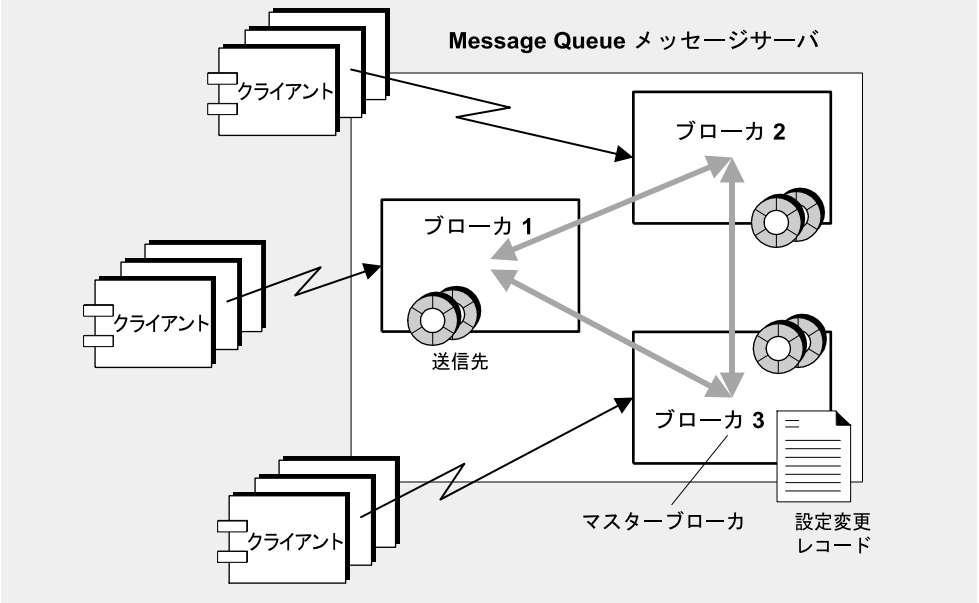
ブローカクラスタを使用すると、サービスの可用性は向上しますが、データの可用性は向上しません。クラスタ内の 1 つのブローカに障害が発生した場合、そのブローカに接続している複数のクライアントはクラスタ内の別のブローカに再接続できますが、代わりのブローカに再接続する間に一部のデータが失われることがあります。

クラスタのアーキテクチャー

図 4-1 に、ブローカクラスタの Message Queue のアーキテクチャーを示します。1 つのクラスタ内のブローカそれぞれは、他のすべてのブローカと直接接続されています。各クライアント (メッセージプロデューサまたはコンシューマ) には 1 つのホームブローカがあります。クライアントは、ホームブローカに直接通信し、そのブローカがクラスタ内の唯一のブローカであるかのようにメッセージを送受信します。背後では、そのホームブローカが他のブローカと協調動作し、接続されたすべてのクライアントに配信サービスを提供します。

クラスタ内では、サービスの可用性は送信先と永続サブスクライバに関する情報を共有できるブローカによって異なります。クラスタ化されたブローカに障害が発生した場合、この状態情報が同期されなくなる可能性があります。この危険性に対処するために、クラスタ内の 1 つのブローカをマスターブローカとして指定することができます。マスターブローカは、設定変更レコードを保持し、クラスタの持続エンティティー (送信先と永続サブスクリプション) への変更点を追跡します。このレコードを使用して、変更が加えられたときにオフラインだったブローカに変更情報を伝えます。

図 4-1　クラスタのアーキテクチャー



以降の節では、クラスタ内でメッセージ配信が実行される仕組み、および 1 つまたは複数のブローカがオフラインになっていてもブローカを設定および同期する方法について説明します。

メッセージ配信

クラスタ構成では、ブローカが送信先とメッセージコンシューマに関する情報を共有します。各ブローカは次の情報を認識しています。

- クラスタ内のすべての物理的な送信先の名前、タイプ、および属性
- 各メッセージコンシューマの名前、場所、および配信対象メッセージ
- 上の情報に対する更新 (削除、追加、または再設定)

これにより、各ブローカは、直接接続されたメッセージプロデューサからのメッセージをリモートのメッセージコンシューマにルーティングすることができるようになります。プロデューサのホームブローカの役割は、コンシューマのホームブローカとは異なります。

- プロデューサのホームブローカは、そのプロデューサから送信されたメッセージの保持とルーティング、ログ作成、トランザクションの管理、コンシューミングクライアントからの通知の処理を行います。
- コンシューマのホームブローカは、コンシューマに関する情報の保持、メッセージのコンシューマへの転送、コンシューマが引き続き利用可能かどうかおよびメッセージが正常にコンシュームされたかどうかのプロデューサのブローカへの通知を行います。

クラスタ化されたブローカは、連係動作して、クラスタ内のメッセージトラフィックを最小限に抑えます。たとえば、リモートのブローカに同じトピック送信先への同一のサブスクリプションが 2 つある場合、メッセージは 1 回だけ送信されます。ローカルコンシューマへの配信がリモートコンシューマへの配信よりも優先されるように指定する送信先のプロパティーを設定することによってトラフィックをさらに減らすことができます。

クライアントとブローカの間で、暗号化によるセキュリティー保護されたメッセージ配信が必要な場合は、クラスタを設定して、ブローカ間でセキュリティー保護されたメッセージ配信を行うこともできます。

送信先の属性

クラスタ化されたブローカ上にある物理的な送信先の属性のセットは、クラスタ内にあるその送信先のすべてのインスタンスに適用されます。ただしこれらの属性によって指定された制限には、クラスタ全体に適用されるものもあれば、個々の送信先インスタンスに適用されるものもあります。表 4-1 に物理的な送信先に設定可能な属性の一覧とそれらの範囲を示します。

表 4-1　クラスタ化されたブローカ上の物理的な送信先のプロパティー

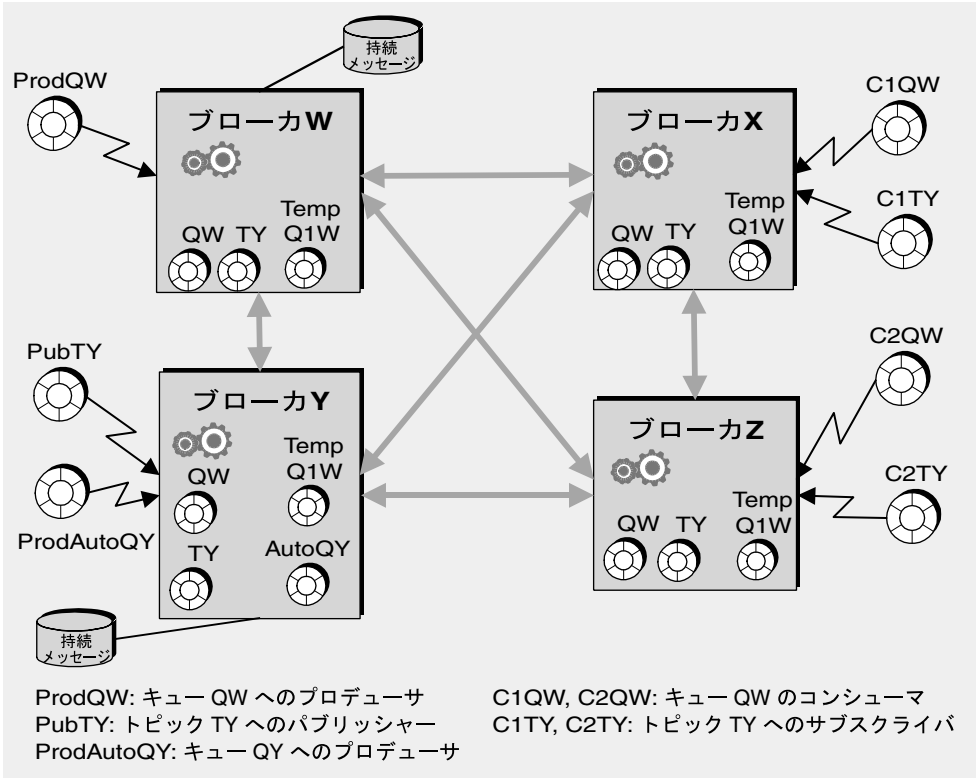
プロパティー名	範囲
maxNumMsgs	ブローカごと。これにより、クラスタ全体にプロデューサを配信し、コンシュームされていないメッセージの総数に制限を設定することができます。
maxTotalMsgBytes	ブローカごと。これにより、クラスタ全体にプロデューサを配信し、コンシュームされていないメッセージ用に予約されているメモリーの総容量に制限を設定することができます。
lmitBehavior	グローバル。
maxBytesPerMsg	ブローカごと。
maxNumProducers	ブローカごと。
maxNumActiveConsumers	グローバル。
maxNumBackupConsumers	グローバル。
consumerFlowLimit	グローバル。
localDeliveryPreferred	グローバル。
isLocalOnly	グローバル。
useDMQ	ブローカごと。

クラスタ化と送信先

送信先が管理者作成、自動作成、一時のいずれかによって、クラスタ内で送信先が伝播される方法、およびコネクションまたはブローカに障害が発生した場合の送信先の処理方法が変わります。

図 4-2 に 4 つのクラスタ化されたブローカを示します。図は、ブローカ間の直接 (プライベート) 接続、およびクライアントとクライアントが接続されるブローカの間のコネクションを示しています。図に示したいくつかの可能な処理について、以下の節で説明します。

図 4-2　クラスタの例



応答先モデルを使用したキューへのプロデュース

前の図は次のことを示しています。

1. 管理者は、物理的な送信先 `QW` を作成します。キューは、作成時にクラスタ全体に複製されます。
2. プロデューサ `ProdQW` がメッセージをキュー `QW` に送信し、応答先モデルを使用して、応答を一時キュー `TempQ1W` に送信します。一時キューは、アプリケーションが一時送信先を作成し、コンシューマを追加するときに作成および複製されます。
3. ホームブローカ `BrokerW` は、`QW` に送信されたメッセージを保持し、このメッセージの選択条件を満たす最初のアクティブなコンシューマにメッセージをルーティングします。メッセージを受信する準備ができたコンシューマに応じて、メッセージは、コンシューマ `C1QW` (`BrokerX` 上) またはコンシューマ `C2QW` (`BrokerY` 上) に配信されます。メッセージを受信したコンシューマは、送信先 `TempQ1W` に応答を送信します。

自動作成の送信先へのプロデュース

前の図は次のことを示しています。

1. プロデューサ ProdAutoQY がブローカ上に存在していない送信先 AutoQY にメッセージを送信します。
2. ブローカは、メッセージを保持し、送信先 AutoQY を作成します。

   自動作成の送信先は、クラスタ全体に自動的に複製されません。コンシューマがキュー AutoQY からのメッセージの受信を選択したときにのみ、コンシューマのホームブローカが送信先 AutoQY を作成して、コンシューマにメッセージを送信します。1 つのコンシューマが自動作成の送信先を作成した時点で、送信先がクラスタ全体に複製されます。

トピック送信先へのパブリッシュ

前の図は次のことを示しています。

1. 管理者は、物理的なトピック送信先 TY を作成します。管理者作成の送信先 TY は、送信先が使用される前にブローカクラスタ全体に複製されます。
2. パブリッシャー PubTY がメッセージを TY に送信します。
3. ホームブローカ BrokerY は、TY にパブリッシュされたすべてのメッセージを保持し、それらのメッセージをこのメッセージの選択条件に一致するすべてのトピックサブスクライバにルーティングします。

コネクションまたはブローカに障害が発生した場合の送信先の処理

表 4-2 は、クラスタ内のさまざまな種類の送信先が複製および削除される方法を説明しています。

表 4-2　クラスタ内の送信先の処理

送信先	伝播と削除
管理者作成	送信先は、作成時にクラスタ内に伝播され、各ブローカが送信先に関する情報を持続的に保存します。 送信先は、管理者が明示的に送信先を削除すると破棄されます。 マスターブローカがある場合は、クラスタ内のブローカが状態情報を同期できるように、作成と削除のレコードがマスターブローカ内に保存されます。

表 4-2 クラスタ内の送信先の処理 (続き)

送信先	伝播と削除
一時	送信先は、作成時にクラスタ全体に伝播されます。 一時送信先に関連付けられているコンシューマが再接続を許可されている場合、送信先はコンシューマのホームブローカ上に持続的に保存されます。再接続できない場合、送信先は保存されません。 コンシューマがコネクションを失った場合、送信先はすべてのブローカ上で削除されます。 コンシューマのホームブローカがクラッシュし、コンシューマが再接続を許可されている場合、このコンシューマに関連付けられている一時送信先が監視されます。コンシューミングクライアントが一定の時間内に再接続しない場合、クライアントに障害が発生したと見なされ、送信先が削除されます。
自動作成	プロデューサが作成され、送信先が存在しない場合、プロデューサのホームブローカ上に送信先が作成されます。 存在しない送信先用のコンシューマが作成された場合、コンシューマと送信先に関する情報がクラスタ全体に伝播されます。 自動作成の送信先は、管理者が明示的に削除することも、自動的に削除することもできます。 • 一定の時間にわたりコンシューマまたはメッセージがない場合は、ブローカごとに削除される。 • ブローカの再起動時、その送信先用のメッセージがない場合は、ブローカごとに削除される。

クラスタ設定

起動時にクラスタ内のブローカどうしでコネクションを確立するには、マスターブローカが存在する場合はそれも含め、それぞれのブローカに他のすべてのブローカのホスト名とポート番号を渡す必要があります。この情報は、クラスタ設定プロパティーのセットによって指定されます。このプロパティーセットは、クラスタ内のすべてのブローカで同じになっているべきです。それぞれのブローカの設定プロパティーは個別に指定できますが、この方法だと誤りが発生しやすく、クラスタ設定の一貫性が失われる可能性が高くなります。代わりに、設定プロパティーのすべてを、起動時に各ブローカが参照する中央の 1 つのクラスタ設定ファイルに記述する方法をお勧めします。これにより、すべてのブローカで確実に同じ設定情報を共有できます。

クラスタ設定プロパティーについての詳細は、『Message Queue 管理ガイド』を参照してください。

注 クラスタ設定ファイルは、クラスタを設定するために使用するのが本来の目的ですが、クラスタ内の他のすべてのブローカと共有する他のプロパティーを保管するためにも便利な場所です。

クラスタの同期化

クラスタの設定が変更されると、どんな場合でも変更に関する情報がクラスタ内のすべてのブローカに自動的に伝播されます。次のいずれかのイベントが発生した場合、クラスタの設定が変更されます。

- クラスタのブローカの 1 つで、送信先が作成されたか破棄された。
- 送信先のプロパティーが変更された。
- メッセージコンシューマがそのホームブローカを登録した。
- 明示的に、またはクライアント、ブローカ、あるいはネットワークの障害により、メッセージコンシューマがホームブローカから切断された。
- メッセージコンシューマがトピックの永続サブスクリプションを確立した。

こうした変更の情報は、変更が行われたときにオンラインになっている、クラスタ内のすべてのブローカに即座に伝播されます。ただし、クラッシュなどのためにオフラインになっているブローカは、変更が生じたときに変更の通知を受け取りません。オフラインブローカに対処するため、Message Queue はクラスタの設定変更記録を保持し、作成または破棄されたすべての持続エントリ (送信先および永続サブスクリプション) を記録します。オフラインからオンラインに戻ったブローカや、クラスタに新しく追加されたブローカは、この記録を参照して送信先および永続サブスクライバに関する情報を調べ、現在アクティブなメッセージコンシューマに関して他のブローカと情報を交換します。

マスターブローカに指定されたクラスタ内の 1 つのブローカは、設定変更記録を保守する役割を担います。他のブローカはマスターブローカなしで初期化を完了できないので、マスターブローカはクラスタ内で必ず最初に起動してください。マスターブローカがオフラインになると、他のブローカが設定変更記録にアクセスできないので、設定情報をクラスタ内全体に伝播できなくなります。このような状況では、送信先または永続サブスクリプションを作成、再設定、または破棄しようとしたり、永続サブスクリプションの再有効化のような関連性のある操作を実行しようとしたりすると、例外が生じます。ただし、非管理メッセージ配信は、通常どおり動作を続けます。マスターブローカと設定変更記録の使用はオプションです。これらが必要になるのは、クラスタの設定を変更したあとまたはブローカの障害が発生したあとにクラスタの同期化を行う場合のみです。

第 5 章

Message Queue と J2EE

Java 2 Platform、Enterprise Edition (J2EE プラットフォーム) は、多層アプリケーションとシンクライアントエンタープライズアプリケーションをホストする、標準サーバープラットフォームの仕様です。J2EE プラットフォームの要件の 1 つは、分散コンポーネントが、信頼性の高い非同期メッセージングにより対話できるようにすることです。この対話動作は、JMS プロバイダを使用することによって可能になります。事実上、Message Queue は J2EE プラットフォームの基準 JMS 実装製品です。

この章では、J2EE プラットフォーム環境で実装される派生 JMS について説明します。章には次のトピックが含まれます。

Message Queue を J2EE 準拠アプリケーションサーバー用の JMS プロバイダとして使用する方法については、『Message Queue 管理ガイド』を参照してください。

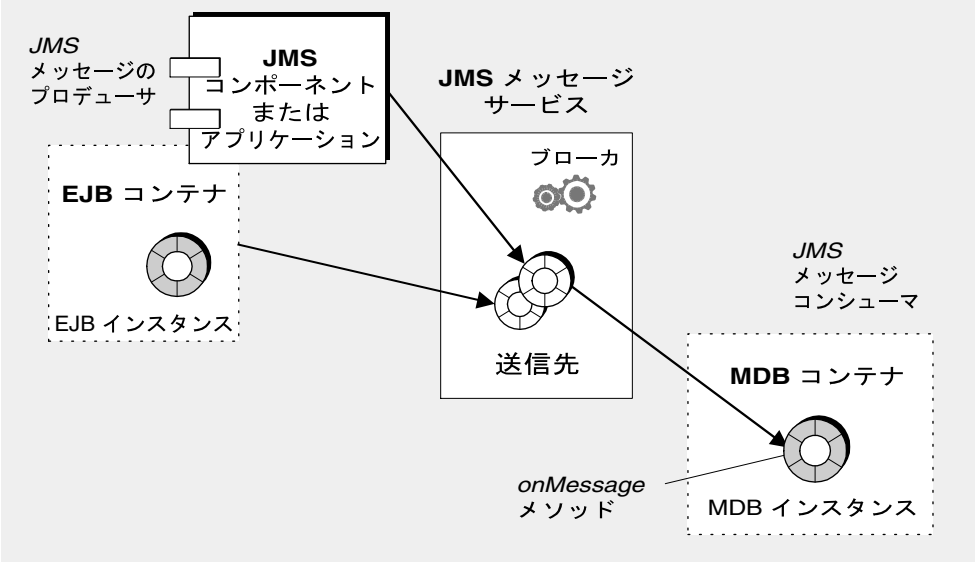
JMS/J2EE プログラミング : メッセージ駆動型 Beans

第 2 章で説明した一般的な JMS クライアントプログラミングモデルのほかに、さらに JMS クライアントに特化したプログラミングモデルがあり、J2EE プラットフォームアプリケーションのコンテキストで使用されます。この特殊なクライアントは、メッセージ駆動型 Beans と呼ばれ、EJB 2.0 以降の仕様 (http://java.sun.com/products/ejb/docs.html) で指定されている Enterprise JavaBeans (EJB) コンポーネントのシリーズの 1 つです。

その他の EJB コンポーネント (セッション Beans とエンティティー Beans) は同時に呼び出す必要があるため、メッセージ駆動型 Beans が必要となります。ただし、多くのエンタープライズアプリケーションは非同期メッセージングを必要とします。これらのアプリケーションの多くは、サーバーサイドコンポーネントがサーバーリソースを結びつけずに、相互に通信できることが要件となっています。このため、メッセージのプロデューサにしっかり結合していなくても、メッセージを受信してコンシュームできる EJB コンポーネントが必要になります。この機能は、サーバーサイドコンポーネントがアプリケーションイベントに応答する必要のある、すべてのアプリケーションで必要となります。エンタープライズアプリケーションでは、負荷が増加する場合、この機能を拡張する必要があります。

メッセージ駆動型 Beans (MDB) は、サポートするコンポーネントに分散サービスを提供する特殊な EJB コンテナによってサポートされる EJB コンポーネントです。

図 5-1　MDB を使用したメッセージング



- JMS メッセージ駆動型 Beans は、JMS `MessageListener` インタフェースを実装する EJB です。MDB 開発者がプログラミングした `onMessage` メソッドは、MDB コンテナがメッセージを受信したときに呼び出されます。`onMessage()` メソッドは、標準的な `MessageListener` オブジェクトの `onMessage()` メソッドがコンシュームするのと同じように、メッセージをコンシュームします。ほかの EJB コンポーネントの場合とは異なり、メソッドを MDB でリモートに呼び出さないため、これと関連付けられているホームまたはリモートのインタフェースはありません。MDB は単一の送信先からのメッセージをコンシュームできます。図 5-1 にあるように、スタンドアロン JMS アプリケーション、JMS コンポーネント、EJB コンポーネント、および Web コンポーネントにより、メッセージをプロデュースできます。

- 特殊な EJB コンテナが MDB をサポートします。このコンテナは、MDB のインスタンスを作成し、メッセージの非同期コンシュームを行うようにインスタンスを設定します。コンテナは、メッセージサービスを使用したコネクションの設定 (認証を含む)、特定の送信先に関するセッションのプールの作成、プールされたセッション間のメッセージ分散の管理を行います。コンテナは MDB インスタンスのライフサイクルを制御するため、受信メッセージの読み込みに対応できるように、MDB インスタンスのプールを管理します。

  メッセージコンシュームを設定するときにコンテナによって使用されるコネクションファクトリと送信先の属性を指定する配置記述子は、MDB に関連付けられています。また、配置記述子には、コンテナを設定するために配置ツールによって必要とされるほかの情報も含まれます。各コンテナでは、1 つの MDB のインスタンスをサポートします。

J2EE アプリケーションサーバーのサポート

J2EE アーキテクチャーでは、EJB コンテナが J2EE アプリケーションサーバーにホストされます。アプリケーションサーバーは、トランザクションマネージャー、持続マネージャー、ネームサービス、およびメッセージングや MDB の場合には JMS プロバイダなど、さまざまなコンテナで必要とされるリソースを提供します。

Sun Java System Application Server では、Sun Java System Message Queue によって JMS メッセージングのリソースが提供されます。

- Sun Java System Application Server 7.0 の場合、Message Queue メッセージングシステムはネイティブな JMS プロバイダとしてアプリケーションサーバーに統合されます。
- Sun J2EE 1.4 アプリケーションサーバーの場合、Message Queue は埋め込みの JMS リソースアダプタとしてアプリケーションサーバーにプラグインされます。

アプリケーションサーバーの将来リリースでは、Message Queue はリソースアダプタの標準的な配備方法と設定方法を使用してアプリケーションサーバーにプラグインされる予定です。

J2EE アーキテクチャーの詳細は、
http://java.sun.com/j2ee/download.html#platformspec にある「J2EE Platform Specification」を参照してください。

JMS リソースアダプタ

リソースアダプタは、J2EE 1.4 に準拠するアプリケーションサーバーに追加機能をプラグインする際の標準的な方法です。J2EE Connector Architecture (J2EECA) 1.5 仕様によって定義されている標準を使用して、アプリケーションサーバーは標準的な手法で外部システムとやり取りできます。これらの外部システムには、企業情報システム (EIS)、および JMS プロバイダなどのメッセージングシステムが含まれます。Message Queue には、アプリケーションサーバーで Message Queue を JMS プロバイダとして使用できるようにする JMS リソースアダプタが組み込まれています。

JMS リソースアダプタをアプリケーションサーバーへ接続することにより、アプリケーションサーバーに配備されて稼働している J2EE コンポーネントで、JMS メッセージを交換することができます。これらのコンポーネントに必要となる JMS コネクションファクトリと送信先管理対象オブジェクトは、J2EE アプリケーションサーバー管理ツールを使用して作成し設定されます。

ただし、ブローカと物理的な送信先の管理など、そのほかの管理操作は J2EECA 仕様には含まれていません。プロバイダ固有のツールを使用しなければ実行できません。

Message Queue リソースアダプタは、Sun J2EE 1.4 アプリケーションサーバーに組み込まれています。ただし、ほかの J2EE 1.4 アプリケーションサーバーではまだ承認されていません。

Message Queue リソースアダプタは、オペレーティングシステムに依存するディレクトリに格納された単一ファイル (`imqjmsra.rar`) です (『Message Queue 管理ガイド』を参照)。`imqjmsra.rar` ファイルには、リソースアダプタの配備ディスクリプタ (`ra.xml`) とともに、アダプタを使用するためにアプリケーションサーバーで必要とされる JAR ファイルが記述されています。

アプリケーションサーバーに添付の説明書に従ってリソースアダプタを配置し設定すれば、J2EE 1.4 準拠のアプリケーションサーバーで Message Queue リソースアダプタを使用できます。商用の J2EE 1.4 アプリケーションサーバーが市販されるようになり、Message Queue リソースアダプタもそれらのアプリケーションサーバー用として承認されるようになれば、Message Queue のマニュアルには関連する配備手順と設定手順についての情報が掲載されるようになります。

Message Queue オプションの JMS 機能の実装

JMS 仕様は、いくつかの項目をオプションとしています。各 JMS プロバイダ (ベンダー) は、それらのオプションを実装するかどうかを選択します。この付録では、Message Queue 製品で JMS オプション項目を処理する方法について説明します。

表 A-1 に Message Queue サービスが JMS オプション項目を処理する方法を示します。

表 A-1 オプションの JMS 機能

JMS 仕様の節	説明と Message Queue における実装
3.4.3 `JMSMessageID`	「メッセージ ID に対しては、メッセージの作成とサイズの拡大に力を注いできたことから、一部には、メッセージ ID がアプリケーションで使用されないというヒントが与えられれば、メッセージオーバーヘッドを最適化できる JMS プロバイダも存在します。JMS メッセージプロデューサは、メッセージ ID を無効にするためのヒントを与えてくれます。」 **Message Queue 実装** : 製品は、メッセージ ID の生成を無効にしません。`MessageProducer` での `setDisableMessageID()` の呼び出しは、すべて無視されます。すべてのメッセージは、有効な `MessageID` 値を持ちます。
3.4.12 メッセージのヘッダーフィールドのオーバーライド	「JMS では、管理者がこれらのヘッダーフィールド値をオーバーライドする方法を、具体的に規定してはいません。JMS プロバイダは、この管理オプションをサポートする必要はありません。」 **Message Queue 実装** : Message Queue 製品は、クライアントランタイムの設定により、メッセージヘッダーフィールドの値の、管理上のオーバーライドをサポートしています (51 ページの「メッセージヘッダー」を参照)。

表 A-1　　オプションの JMS 機能 (続き)

JMS 仕様の節	説明と Message Queue における実装
3.5.9 JMS の定義済みプロパティー	「JMS では、JMS の定義済みプロパティー用として、JMSX プロパティー名のプレフィックスを予約しています。」「特に記述がないかぎり、これらのプロパティーのサポートはオプションです。」 **Message Queue 実装** : JMS 1.1 仕様で定義済みの JMSX プロパティーは、Message Queue 製品でサポートされています (『Message Queue 管理ガイド』を参照)。
3.5.10 プロバイダ固有のプロパティー	「JMS では、プロバイダ固有のプロパティー用として、'JMS_*<vendor_name>*' というプロパティー名のプレフィックスを予約しています。」 **Message Queue 実装** : プロバイダ固有のプロパティーの目的は、プロバイダにネイティブなクライアントで JMS をサポートするのに必要な、特殊な機能を提供することです。これらは、JMS 対 JMS のメッセージングには使用できません。
4.4.8 分散トランザクション	「JMS では、プロバイダが分散トランザクションをサポートすることを必要としていません。」 **Message Queue 実装** : 分散トランザクションは、このリリースの Message Queue 製品でサポートされています (60 ページの「トランザクション」を参照)。
4.4.9 複数のセッション	「JMS では、PTP < ポイントツーポイント分散モデル > について、同じキューに同時にアクセスする QueueReceivers に対するセマンティクスを指定していません。しかし、JMS がこの機能のサポートを禁止しているわけではありません。」詳細は、JMS 仕様の 5.8 節を参照してください。 **Message Queue 実装** : Message Queue 実装は、複数のコンシューマへのキュー配信をサポートしています。詳細は、42 ページの「ポイントツーポイントメッセージング」を参照してください。

Message Queue の機能

Message Queue サービスは、JMS 1.1 仕様を完全に実装し、柔軟で信頼性の高い非同期のメッセージ配信を実現します。JMS への準拠に関連する問題については、付録 A「Message Queue オプションの JMS 機能の実装」を参照してください。ただし、Message Queue は JMS の要件より多くの機能や特長を備えています。これらの機能を使用して、24 時間稼働の基幹業務で大量のメッセージを交換する多くの分散コンポーネントで構成されるシステムを統合および監視できます。

このマニュアルでは、Message Queue サービスについて説明する過程でこれらの機能について説明してきました。参考までに、この付録では、Message Queue の機能の要約を記載します。各機能について簡単に説明し、機能を使用するための操作の要約を示し、さらに、このマニュアル内でこれらの機能について説明している節、および Message Queue マニュアルセットに含まれるこれらの機能を詳細に説明する特定のマニュアルの参照先を示します。

表 B-1 に Message Queue の機能をアルファベット順に示します。これらの機能は次に示すカテゴリに大きく分類することができます。

- 統合サポート
  - HTTP コネクション
  - セキュリティー保護コネクション
  - C クライアントサポート
  - SOAP のサポート
  - J2EE リソースアダプタ
- セキュリティー
  - 認証
  - 承認
  - 暗号化 (「セキュリティー保護コネクション」を参照)

- スケーラビリティー
  - スレッド管理
  - スレッド管理
  - スレッド管理
- 可用性
  - メモリーリソース管理
  - クライアントに対するメッセージフローの制御
  - 自動再接続
  - 信頼性の高いデータ持続
  - 接続の ping
- 管理機能
  - 管理ツール
  - メッセージベースの監視 API
  - 調整可能なパフォーマンス
  - 設定可能な物理的な送信先
  - ブローカの設定
  - デッドメッセージキュー
- 柔軟なサーバー設定
  - 設定可能な持続性
  - LDAP サーバーのサポート
  - JNDI サービスプロバイダのサポート
- パフォーマンス
  - メッセージ圧縮
  - 調整可能なパフォーマンス
  - 設定可能な物理的な送信先

表 B-1　Message Queue の機能

機能	説明と参照情報
管理ツール	Message Queue サービスは、GUI ツールとコマンド行ツールを備えており、これらのツールによって、送信先、トランザクション、永続サブスクリプション、管理対象オブジェクトストア、ユーザーリポジトリ、JDBC 準拠のデータストア、およびサーバー証明書を管理することができます。 **参照情報** 80 ページの「管理ツール」 『Message Queue 管理ガイド』の「管理タスクと管理ツール」
認証	ブローカへの接続を要求するユーザーを認証します。 Message Queue サービスを使用して、ユーザーの名前とパスワードをユーザーリポジトリに格納されている値と比較して検証することにより、ユーザーがブローカに接続できるようにします。リポジトリとして Message Queue に付属している単層型ファイルリポジトリを使用することも、LDAP リポジトリ (LDAP v2 または v3 プロトコル) を使用することもできます。 **使用手順** 1. ユーザーリポジトリを作成するかデフォルトのインスタンスを使用します。 2. `imqusermgr` ツールを使用してリポジトリに値を入力します。 **参照情報** 76 ページの「認証と承認」 『Message Queue 管理ガイド』の「セキュリティーの管理」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
承認	ユーザーによる特定の操作の実行を承認します。 Message Queue サービスを使用して、ユーザーおよびユーザーのグループが実行できる操作を指定するアクセス制御プロパティーファイルを作成することができます。ブローカは、クライアントによって、コネクションの作成、プロデューサの作成、コンシューマの作成、またはキューの参照を要求されたときにこのファイルを確認します。 **使用手順** ブローカインスタンス用に自動的に作成されたアクセス制御プロパティーファイルを編集します。 **参照情報** 76 ページの「認証と承認」 『Message Queue 管理ガイド』の「セキュリティーの管理」
自動再接続	管理者は、コネクションファクトリ管理対象オブジェクトに対してコネクションの属性を設定し、コネクションに障害が発生した場合に自動再接続できるようにします。再接続は、同じブローカに対して行うか、クラスタが使用されている場合はクラスタ内の別のブローカに対して行うことができます。再接続の試行回数と試行の間隔を指定することができます。クラスタ化されたブローカの場合は、ブローカのリストを反復する回数および特定の順序でリストを反復するかどうかを指定することもできます。 **参照情報** 69 ページの「コネクションサービス」 『Message Queue 管理ガイド』の「管理対象オブジェクトの管理」と「管理対象オブジェクト属性のリファレンス」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
ブローカクラスタ	管理者は、ブローカインスタンスをブローカクラスタにグループ化することで、数多くのブローカインスタンスの間でクライアントコネクションとメッセージ配信のバランスを取ることができます。 **使用手順** 1. クラスタ内の各ブローカのクラスタ設定プロパティーを指定します。この操作は、設定ファイルを使用するか、各ブローカのプロパティーを設定することで実行できます。 2. マスターブローカがある場合は、マスターブローカを起動します。 3. クラスタ内のほかのブローカを起動します。 **参照情報** 第 4 章「ブローカクラスタ」 『Message Queue 管理ガイド』の「クラスタを使用した作業」
ブローカの設定	管理者は、ブローカのプロパティーを設定して、Message Queue サービスのパフォーマンスを調整できます。調整するプロパティーには、ルーティングサービス、持続サービス、セキュリティー、監視、管理対象オブジェクトの管理が含まれます。 **参照情報** 第 3 章「Message Queue サービス」 『Message Queue 管理ガイド』の「ブローカの設定」と「ブローカのプロパティーのリファレンス」
C クライアントサポート	C クライアントは Message Queue メッセージングサービスを使用して、メッセージを送受信することができます。C API により、旧バージョンの C アプリケーションと C++ アプリケーションを JMS ベースのメッセージに加えることができます。 Message Queue の C API は、標準 JMS 機能のほとんどに対応する C クライアントランタイムによってサポートされています。例外は、管理対象オブジェクト、マップ / ストリーム / メッセージ本体のタイプ、分散トランザクション、およびキューブラウザを使用する場合です。C クライアントランタイムも、Message Queue のエンタープライズ機能のほとんどをサポートしません。 **参照情報** 65 ページの「Java クライアントと C クライアント」 『Message Queue Developer's Guide for C Clients』

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
圧縮されたメッセージ	Java クライアントは、送信するメッセージをクライアントランタイムで圧縮するように、メッセージプロパティーを設定することができます。コンシューマ側のランタイムは、メッセージを解凍してからコンシューマに配信します。メッセージの圧縮によって実際にパフォーマンスが向上するかどうかを判断するために使用できる追加のプロパティーが用意されています。 **参照情報** 54 ページの「メッセージ本体」 『Message Queue Developer's Guide for Java Clients』の「Message Queue Clients: Design and Features」
設定可能な持続性	管理者は、Message Queue に付属しているファイルベースの持続ストアを使用するか、Oracle 8i などの JDBC 準拠のデータベースを使用するようにブローカを設定することができます。 **使用手順** ファイルシステム持続ストレージまたは JDBC 準拠ストレージに関連するブローカのプロパティーを設定します。 **参照情報** 74 ページの「持続サービス」 『Message Queue 管理ガイド』の「ブローカの設定」
設定可能な物理的な送信先	管理者は、送信先を作成するときに物理的な送信先のプロパティーを設定することによって、いくつかのメッセージング動作を定義することができます。すべての送信先に対して、次の動作を設定できます (コンシュームされていないメッセージの最大数またはそのようなメッセージで使用可能な最大メモリー量、メモリーの制限に達した場合にブローカで拒否するメッセージ、プロデューサとコンシューマの最大数、最大メッセージサイズ、1 回のバッチで配信されるメッセージの最大数、送信先がローカルのコンシューマにのみ配信できるかどうか、送信先上のデッドメッセージをデッドメッセージキューに移動できるかどうか)。 **参照情報** 71 ページの「送信先とルーティングサービス」 『Message Queue 管理ガイド』の「物理的送信先を管理する」と「物理的送信先のプロパティーのリファレンス」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
接続の ping	管理者は、コネクションファクトリーの属性を設定して、クライアントランタイムからブローカへの ping 操作の頻度を指定することができます。これにより、クライアントは、失敗したコネクションを前もって検出することができます。 **参照情報** 69 ページの「コネクションサービス」 『Message Queue 管理ガイド』の「コネクションファクトリの属性」
デッドメッセージキュー	Message Queue メッセージサービスは、期限切れになったかまたはブローカが処理できなかったメッセージを保持するためのデッドメッセージキューを作成します。キューの内容を調べて、システムのパフォーマンスを監視、調整、または問題を解決することができます。 **参照情報** 71 ページの「送信先とルーティングサービス」 『Message Queue 管理ガイド』の「物理的送信先を管理する」
HTTP コネクション	Java クライアントは、ブローカへの HTTP コネクションを作成することができます。 HTTP トランスポートにより、ファイアウォールを通過してメッセージを配信できます。Message Queue は、Web サーバー環境で実行される HTTP トンネルサーブレットを使用して、HTTP の仕組みを実装しています。クライアントによってプロデュースされるメッセージは、クライアントランタイムによって HTTP 要求としてラップされ、HTTP を介し、ファイアウォールを通過してトンネルサーブレットに配信されます。トンネルサーブレットは HTTP 要求から JMS メッセージを抽出し、そのメッセージを TCP/IP 経由でブローカに配信します。 **使用手順** 1. HTTP トンネルサーブレットを Web サーバー上に配備します。 2. ブローカの `httpjms` コネクションサービスを設定し、ブローカを起動します。 3. HTTP コネクションを設定します。 4. ブローカへの HTTP コネクションを取得します (Java クライアントのみ) **参照情報** 32 ページの「ブローカへの接続」 『Message Queue 管理ガイド』の付録 C 「HTTP サポートの有効化」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
対話型の監視	管理者は、`imqcmd metrics` コマンドを使用して、ブローカをリモートから監視することができます。監視するデータには、JVM メトリックス、ブローカメッセージのフロー、コネクション、コネクションリソース、メッセージ、送信先メッセージのフロー、送信先リソースの使用状況が含まれます。 **参照情報** 78 ページの「監視サービス」 『Message Queue 管理ガイド』の「メッセージサーバーの監視」
J2EE リソースアダプタ	Message Queue によって提供されるリソースアダプタを、J2EE 準拠のアプリケーションサーバーにプラグインすることができます。J2EE にはアプリケーションサーバー内で実行される分散コンポーネントが信頼性の高い非同期メッセージを使用して対話できるという要件があり、Message Queue を JMS プロバイダとして使用することによって、アプリケーションサーバーはこの要件を満たすことができます。 **使用手順** アダプタの属性を設定することによってアダプタを設定します。 **参照情報** 98 ページの「J2EE アプリケーションサーバーのサポート」 『Message Queue 管理ガイド』の「JMS リソースアダプタ属性リファレンス」
JNDI サービスプロバイダのサポート	クライアントは、JNDI API を使用して管理対象オブジェクトを検索することができます。 管理者は、`imqobjmgr` ユーティリティーを使用して、JNDI を使用してアクセス可能なオブジェクトストア内の管理対象オブジェクトの追加、一覧表示、更新、および削除を行うことができます。 **参照情報** 80 ページの「管理ツール」 『Message Queue 管理ガイド』の「コマンドのリファレンス」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
LDAP サーバーのサポート	管理者は、LDAP サーバーを使用して、管理対象オブジェクトの保存および認証と承認に必要なユーザー情報を保存することができます。デフォルトでは、Message Queue では、このデータ用のファイルベースストレージを用意しています。 **管理対象オブジェクト用に使用するには** 1. ベンダーによって提供されているツールを使用して、ユーザーリポジトリの値の入力と管理を行います。 2. LDAP 関連のブローカプロパティーを設定します。 3. 管理ユーザーのアクセス制御を設定します。 **参照情報** 『Message Queue 管理ガイド』の「セキュリティーの管理」 **ユーザーリポジトリ用に使用するには** 1. ベンダーによって提供されているツールを使用して、LDAP サーバーを設定します。 2. LDAP 関連のブローカプロパティーを設定して、初期コンテキストとストアの場所を定義します。 3. LDAP サーバーの動作のセキュリティー保護に関連する LDAP 関連のブローカプロパティーを設定します。 **参照情報** 75 ページの「セキュリティーサービス」 『Message Queue 管理ガイド』の「管理対象オブジェクトの管理」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
メモリーリソース管理	管理者は、次の動作を設定することができます。 1. 送信先のプロパティーを設定して、プロデューサの最大数、メッセージの最大サイズ、および任意のメッセージの最大サイズを指定できます。 2. 送信先のプロパティーを設定してメッセージフローを制御します。 3. 送信先のプロパティーを設定して各送信先のメッセージフローを管理します。 4. ブローカのプロパティーを設定して、そのブローカのすべての送信先のメッセージ制限を指定します。 5. ブローカのプロパティーを設定して、ブローカがさらに深刻な状況に陥ったときに、メモリーの過負荷を避けるためのアクションを実行できるように、使用可能なシステムメモリーのしきい値を指定します。実行されるアクションは、メモリーリソースの状態によって異なります。 **参照情報** 71 ページの「送信先とルーティングサービス」 『Message Queue 管理ガイド』の「ブローカの設定」、「ブローカのプロパティーのリファレンス」、および「物理的送信先のプロパティーのリファレンス」
メッセージ圧縮	開発者は、送信する前にクライアントランタイムでメッセージを圧縮するようにメッセージヘッダーのプロパティーを設定することができます。クライアントランタイムは、メッセージを解凍してからコンシューマに配信します。 **参照情報** 53 ページの「メッセージプロパティー」 『Message Queue Developer's Guide for Java Clients』の「Message Compression」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
クライアントに対するメッセージフローの制御	管理者または開発者は、接続を設定して、さまざまなフロー制限と測定手段を指定してペイロードメッセージとコントロールメッセージの衝突を最小限に抑え、これによりメッセージのスループットを最大限に高めることができます。 **使用手順** コネクションファクトリ管理対象オブジェクトのフロー制御属性を設定するか (管理者)、コネクションファクトリのフロー制御プロパティーを設定します (開発者)。 **参照情報** 50 ページの「コネクションファクトリとコネクション」 『Message Queue 管理ガイド』の「管理対象オブジェクトの管理」と「管理対象オブジェクト属性のリファレンス」
メッセージベースの監視 API	Java クライアントは監視 API を使用してカスタム監視アプリケーションを作成することができます。監視アプリケーションは、特殊なトピック送信先からメトリックスメッセージを取得するコンシューマです。 **使用手順** 1. メトリックス監視クライアントを作成します。 2. ブローカのプロパティーを設定して、ブローカのメトリックスメッセージプロデューサを設定します。 3. メトリックストピック送信先のアクセス制御を設定します。 4. 監視クライアントを起動します。 **参照情報** 78 ページの「監視サービス」 『Message Queue Developer's Guide for Java Clients』の「Using the Metrics Monitoring API」 『Message Queue 管理ガイド』の「メッセージサーバーの監視」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
複数のコンシューマへのキューの配信	クライアントは特定のキューに対して複数のコンシューマを登録することができます。 管理者は、アクティブコンシューマの最大数、キューのバックアップコンシューマの最大数を指定することができます。ブローカは、メッセージを登録コンシューマに分配することができるので、コンシューマ間で負荷が分散され、システムの拡張性が維持されます。 **使用手順** 物理的な送信先のプロパティー `maxNumActiveConsumers` と `maxNumBackupConsumers` を設定します。 **参照情報** 42 ページの「ポイントツーポイントメッセージング」 『Message Queue 管理ガイド』の「物理的送信先のプロパティー」と「Multiple Consumer Queue Performance」
信頼性の高いデータ持続	絶対的な信頼性を確保するために、`imq.persist.file.sync.enabled` プロパティーを true に設定することで、オペレーティングシステムがデータを必ず同期的に書き込むようにすることができます。この同期化により、システム障害によるデータの損失をなくすことができますが、パフォーマンスは低下します。これで、データは失われなくなりますが、この時点でデータがクラスタ化されたブローカで共有されていないため、クラスタ内のほかのブローカはデータを利用できません。システムが再び稼動したときに、ブローカは確実に操作を再開することができます。 **参照情報** 74 ページの「持続サービス」 『Message Queue 管理ガイド』の「ブローカのプロパティーのリファレンス」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
セキュリティー保護コネクション	クライアントは、TCP/IP および HTTP トランスポートでの SSL (Secure Socket Layer) 標準を使用してメッセージ送信をセキュリティー保護することができます。これらの SSL ベースのコネクションサービスでは、クライアントとブローカ間で送信されるメッセージを暗号化することができます。 SSL のサポートは、自己署名サーバー証明書に基づいています。Message Queue は、非公開 / 公開キーを生成するユーティリティーを備え、自己署名証明書に公開キーを埋め込みます。証明書はブローカへのコネクションを要求しているクライアントに渡され、クライアントはこの証明書を使用して暗号化されたコネクションを設定します。 **使用手順** 1. 自己署名付き証明書または署名済み証明書を生成します。 2. セキュリティー保護サービスを有効にします。 3. ブローカを起動します。 4. クライアントのセキュリティーコネクションのプロパティーを設定し、クライアントを実行します。 **参照情報** 32 ページの「ブローカへの接続」 75 ページの「セキュリティーサービス」 『Message Queue 管理ガイド』の「セキュリティーの管理」 『Message Queue Developer's Guide for Java Clients』 『Message Queue Developer's Guide for C Clients』
SOAP のサポート	クライアントは、SOAP (XML) メッセージを受信し、それらのメッセージを JMS メッセージとしてラップし、ＪＭＳ メッセージと同じように Message Queue を使用して交換することができます。 クライアントは、特殊なサーブレットを使用して SOAP メッセージを受信し、ユーティリティークラスを使用して SOAP メッセージを JMS メッセージとしてラップすることができます。さらに別のユーティリティークラスを使用して SOAP メッセージを JMS メッセージから抽出することができます。クライアントは標準の SAAJ ライブラリを使用して、SOAP メッセージをアセンブルまたは逆アセンブルすることができます。 **参照情報** 64 ページの「SOAP メッセージの処理」 『Message Queue Developer's Guide for Java Clients』の「Working With SOAP Message」

表 B-1　Message Queue の機能 (続き)

機能	説明と参照情報
スレッド管理	管理者は、特定のコネクションサービスに割り当てられるスレッドの最大数と最小数を指定することができます。管理者はまた、共有スレッドモデルを使用して、コネクションサービスのスループットを増加させることができるかどうかを決定できます。これにより、アイドル状態のコネクションに割り当てられているスレッドをほかの接続で使用できます。 **使用手順** コネクションサービスのスレッド関連プロパティーを設定します。 **参照情報** 70 ページの「スレッドプール管理」 『Message Queue 管理ガイド』の「ブローカの設定」
調整可能なパフォーマンス	管理者は、ブローカのプロパティーを設定して、メモリーの使用状況、スレッド処理リソース、メッセージのフロー、コネクションサービス、信頼性パラメータ、およびメッセージのスループットとシステムのパフォーマンスに影響を与えるほかの要素を調整できます。 **参照情報** 78 ページの「監視サービス」 『Message Queue 管理ガイド』の「メッセージサーバーの監視」と「メッセージサービスの分析と調整」

用語集

この用語集では、Message Queue の使用時に知っておくと便利な用語や概念について説明します。Sun Java System で使用されるすべての用語が記載された用語集については、http://docs.sun.com/app/docs/doc/819-4629 を参照してください。

JMS プロバイダ (JMS provider)　メッセージングシステムの JMS インタフェースを実装し、システムの設定と管理に必要な管理と制御機能を追加する製品。

暗号化 (encryption)　コネクションを通して配信されるときに、改ざんされないようにメッセージを保護するメカニズム。

管理対象オブジェクト (administered objects)　コネクションファクトリや送信先などのあらかじめ設定されたオブジェクト。プロバイダ固有の実装細部をカプセル化し、1 つ以上の JMS クライアントで使用するために管理者が作成します。管理対象オブジェクトを使用することにより、JMS クライアントはプロバイダから独立できます。管理対象オブジェクトは、JNDI 検索を使用して、JMS クライアントによって JNDI ネームスペースに配置され、JMS クライアントからアクセスされます。

キュー (queue)　ポイントツーポイント配信モデルを実装するために、管理者が作成するオブジェクト。メッセージをコンシュームするクライアントがアクティブでない場合でも、メッセージを保持するためにキューは常に使用可能です。キューは、プロデューサとコンシューマの中間段階の待機場所として使用されます。

クライアント (client)　メッセージを交換するメッセージサービスを使用して、ほかのクライアントと対話するアプリケーション (またはソフトウェアコンポーネント)。クライアントは、プロデューシングクライアント、コンシューミングクライアント、またはその両方のいずれかになります。

クライアント識別子 (client identifier)　コネクションおよびコネクションのオブジェクトを、クライアントの代わりに Message Queue ブローカが管理する状態と関連付ける識別子。

クライアントランタイム (client runtime)　メッセージングクライアントに、Message Queue メッセージサービスとのインタフェースを提供する Message Queue ソフトウェア。クライアントランタイムは、クライアントが送信先にメッセージを送信し、送信先からメッセージを受信するために必要なすべての操作をサポートします。

クラスタ (cluster)　スケーラブルなメッセージングサービスを提供するために、並行して処理を行う連結された複数のブローカ。

グループ (group)　コネクション、送信先、および特定の操作の利用を承認する目的で、Message Queue クライアントのユーザーが属するグループ。

コネクション (connection)　ペイロードメッセージとコントロールメッセージの両方を渡すために使用する、クライアントとブローカの間の通信チャネル。

コネクションファクトリ (connection factory)　ブローカへのコネクションを作成するために、クライアントが使用する管理対象オブジェクト。コネクションファクトリは、`ConnectionFactory` オブジェクト、`QueueConnectionFactory` オブジェクト、または `TopicConnectionFactory` オブジェクトのいずれかです。

コンシューマ (consumer)　送信先から送信されたメッセージを受信する場合に使用するセッションによって作成されるオブジェクト (`MessageConsumer`)。ポイントツーポイント配信モデルの場合、コンシューマは受信側、またはブラウザ (`QueueReceiver` または `QueueBrowser`) のどちらかであり、パブリッシュ / サブスクライブ配信モデルの場合、コンシューマはサブスクライバ (`TopicSubscriber`) です。

承認 (authorization)　コネクションサービスや送信先などのメッセージサービスのリソースに、ユーザーが、メッセージサービスによってサポートされる特定の操作を実行するためにアクセスできるかどうかをメッセージサービスが判断するプロセス。

セッション (session)　メッセージを送受信するためのシングルスレッドのコンテキスト。キューセッション、またはトピックセッションのどちらかになります。

セレクタ (selector)　メッセージのソートおよびルーティングで使用するメッセージヘッダーのプロパティー。メッセージサービスは、メッセージセレクタに設定された条件に基づいて、メッセージのフィルタリングやルーティングを行います。

送信先 (destination)　コンシューマへのルーティングおよび後続の配信のために、プロデュースされたメッセージが配信される Message Queue ブローカの物理的な送信先。この物理的な送信先は、管理対象オブジェクトにより識別されカプセル化されます。管理対象オブジェクトを使ってクライアントはメッセージをプロデュースする送信先、メッセージをコンシュームする送信先を指定します。

通知 (acknowledgement)　クライアントとブローカの間で交換されるメッセージを制御して、信頼性の高い方法で配信できるようにします。通知には、クライアント通知とブローカ通知という 2 つの一般的なタイプがあります。

データストア (data store)　ブローカに必要な情報 (永続サブスクリプション、送信先のデータ、持続メッセージ、および監査データ) が恒久的に格納されるデータベース。

デッドメッセージ (dead message)　通常の処理または明示的な管理者による操作以外の理由でシステムから削除されるメッセージ。メッセージは、有効期限が切れたり、メモリー限界を上回るために送信先から削除されたり、配信処理に失敗したりしたときにデッドメッセージと見なされます。デッドメッセージは、デッドメッセージキューに保管することができます。

デッドメッセージキュー (dead message queue)　ブローカの起動時に自動的に作成される特殊な送信先。診断用のデッドメッセージを保管するために使用します。

トピック (topic)　パブリッシュ / サブスクライブ配信モデルを実装するために、管理者が作成するオブジェクト。トピックは、アドレス指定されたメッセージの収集および配信を担うコンテンツ階層のノードとして表示できます。中間段階としてトピックを使用することにより、メッセージパブリッシャーがメッセージサブスクライバから分離されます。

ドメイン (domain)　JMS メッセージング処理をプログラミングするために、JMS クライアントが使用するオブジェクトの集まり。ポイントツーポイント配信モデル用とパブリッシュ / サブスクライブ配信モデル用の 2 つのプログラミングドメインがあります。

トランザクション (transaction)　不可分な単位の作業。この作業は完了されるか、あるいは完全にロールバックされる必要があります。

認証 (authentication)　検証されたユーザーだけがブローカへのコネクションをセットアップすることができるようにするプロセス。

配信モード (delivery mode)　メッセージングの信頼性のインジケータ。必ず 1 回 (1 回に限って) 配信され確実にコンシュームされることを保証する (持続配信モード)、あるいはメッセージが 1 回は配信されることを保証する (非持続配信モード) のどちらかを示します。

配信モデル (delivery model)　メッセージが配信されるモデル。ポイントツーポイント、またはパブリック / サブスクライブのどちらかになります。JMS には、それぞれ特定のクライアントランタイムオブジェクトと特定の送信先タイプを使用する個別のプログラミングドメイン、並びにユニファイドプログラミングドメインがあります。

非同期メッセージング (asynchronous messaging)　メッセージ交換の仕組み。メッセージは、メッセージを受信するコンシューマの準備状態に関係なく送信されます。言い換えると、メッセージの送信側は、送信メソッドが返されるのを待たずにほかの作業を続行することができます。メッセージコンシューマが使用中またはオフラインになっている場合でもメッセージは送信されますが、コンシューマがメッセージを受信するのは、コンシューマの準備が整っている場合です。

ブローカ (broker)　メッセージのルーティング、配信、持続性、セキュリティー、およびログ作成を管理し、パフォーマンスおよびリソース使用状況の監視および調整を実行する場合のインタフェースとして機能する Message Queue のエンティティー。

プロデューサ (producer)　送信先にメッセージを送信するために使用するセッションによって作成されるオブジェクト (MessageProducer)。ポイントツーポイント配信モデルの場合、プロデューサは送信側 (`QueueSender`) になり、パブリッシュ / サブスクライブ配信モデルの場合、プロデューサはパブリッシャー (`TopicPublisher`) になります。

メッセージ (messages)　メッセージングクライアントがコンシュームする非同期要求、レポート、またはイベント。メッセージは、ヘッダーと本体から構成されており、ヘッダーにはフィールドを追加できます。メッセージのヘッダーでは、標準フィールドとオプションのプロパティーを指定します。送信中のデータはメッセージ本体に含まれます。

メッセージサービス (message service)　分散されたコンポーネントまたはアプリケーションの間で、信頼性の高い非同期のメッセージ交換機能を提供するミドルウェアサービス。ブローカがその機能を実行するために必要なブローカ、クライアントランタイム、いくつかのデータストア、およびブローカを設定および監視し、パフォーマンスを調整するために必要な管理ツールが含まれています。

メッセージング (messaging)　エンタープライズアプリケーションで使用される非同期要求、レポート、またはイベントのシステム。これにより、弱い連結のアプリケーションが情報を確実かつ安全に送信できます。

索引

L

M

O

P

S

T

さ

し

す

せ

そ

た

て

と

に

は

ふ

へ

ほ

ま

み

め

ゆ

よ

り

る

ろ