Sun Java™ System

Message Queue 3
技術の概要

2005Q1

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

Part No: 819-2221

目 次

図目次

表目次

はじめに

本書『Sun Java™ System Message Queue 3 2005Q1 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、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他のマニュアルの基礎を成します。Message Queue マニュアルセットの他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。

本書の対象読者

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、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。

Message Queue 管理者とは、Message Queue メッセージングシステム、特にシステムの中枢となる Message Queue メッセージサーバーの設定および管理の担当者です。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されません。

アプリケーション開発者は、Message Queue サービスを使用して他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する責任を担います。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されません。

本書を読む前に

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 の基本的な概念に関する理解を得てください。

本書の構成

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。次の表は、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「基本概念」	Message Queue の概念的な背景について述べ、エンタープライズメッセージングシステムについて解説し、Java Message Service の概念とテクノロジを紹介します
第 2 章「Message Queue の紹介」	アーキテクチャについて解説し、エンタープライズ用途の機能と能力について説明することにより、Message Queue サービスを紹介します
第 3 章「信頼性の高いメッセージ配信」	メッセージングアプリケーションにおいて、Message Queue サービスが信頼性の高いメッセージ配信を実現する仕組みについて説明します
第 4 章「メッセージサーバー」	ブローカの内部構造を検討し、さまざまなブローカのコンポーネントとその機能を説明します。開発環境と運用環境で Message Queue を使用する場合の異なる手法を紹介します。
第 5 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と内部の仕組み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 6 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
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 (Message Queue、JSP) を表します。
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 インストールガイド』	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 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が含まれています。各ユーティリティ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	ロケーション
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。

コメントの送付先

第 1 章

基本概念

Sun Java™ System Message Queue (Message Queue) は、企業全体に分散されたアプリケーションとコンポーネントを統合可能な、信頼性の高い非同期メッセージングサービスを提供します。異なるプラットフォームやオペレーティングシステムで実行されるプロセスであっても、このサービスに接続して相互に対話することができます。

Message Queue は、Java™ Message Service (JMS) オープン標準を実装する標準ベースのメッセージング (messaging) ソリューションです。また Message Queue は、相互運用性、セキュリティ、スケーラビリティ、可用性、管理機能、および企業での大規模配備に必要なその他の特長を備えています。

この章では、Message Queue の基本概念を説明します。次のトピックが含まれます。

JMS の概念や用語についてすでに理解している場合は、第 2 章「Message Queue の紹介」にお進みください。

エンタープライズメッセージングシステム

エンタープライズメッセージングシステムにより、独立した分散アプリケーションやアプリケーションコンポーネントは、メッセージを介して対話することができます。これらのコンポーネントは、同一ホストや同一ネットワークにあっても、またはインターネットを介して疎結合されていても、メッセージングによってデータを渡し、それぞれの機能を調整します。

多数のコンポーネントがメッセージを同時に交換し、高密度スループットをサポートできるようにするには、メッセージを受信するコンシューマの準備が整ってからメッセージを送信するという方法では対応しきれません。メッセージコンシューマが使用中またはオフラインの場合には、準備が整ってからコンシューマ側でメッセージを受け取ることができるような仕組みを、システムに備える必要があります。このように、メッセージの送信と受信を切り離す方法を非同期メッセージ配信と呼びます。

非同期メッセージングモデルは、複雑なシステムを統合するタスクに対して優れた適性を示します。複雑なシステムでは、1 つのコンポーネントが処理を実行する過程で別のコンポーネントの動作を抑制することは難しく、望ましくもありません。非同期メッセージングでは、同期システムで実行可能な制御機能の一部が犠牲になりますが、コンポーネントの相互動作の柔軟性を大幅に向上させることができます。また、1 つのコンポーネントで障害が発生した場合でもシステム全体に波及しないので、堅牢性も高められます。

エンタープライズメッセージングシステムの要件

一般的なエンタープライズアプリケーションシステムは、多くの分散コンポーネントで構成されており、24 時間稼働の基幹業務で大量のメッセージを交換します。非同期メッセージングをサポートすることに加えて、このような基幹システムをサポートするには、エンタープライズメッセージングシステムが次の要件を満たしている必要があります。

信頼性の高い配信能力：あるコンポーネントから別のコンポーネントに送信されるメッセージが、ネットワークやシステムの障害で失われないようにする必要があります。つまりシステムには、メッセージ配信を保証できるだけの信頼性が必要です。

セキュリティ：メッセージングシステムは、ユーザーの認証、メッセージとリソースへの承認されたアクセス、ネットワーク上の暗号化など、基本的なセキュリティ機能をサポートしている必要があります。

スケーラビリティ：メッセージングシステムは、パフォーマンスやメッセージスループットを実質的に低下させることなく、ユーザーやメッセージ数の増加などによる負荷の増大に対応できるようにする必要があります。ビジネスやアプリケーションが拡大するにつれ、これは重要な要件となります。

可用性：メッセージングシステムは、ほとんど停止することなく稼働する必要があります。そのためシステムは、障害が発生した場合でもメッセージングサービスを継続して提供できるよう、十分な冗長性を備えています。

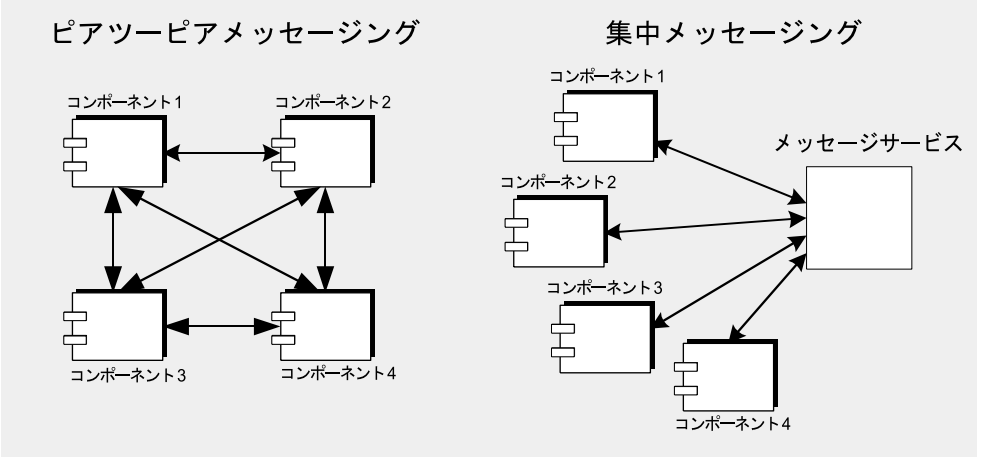
管理機能：メッセージングシステムは、メッセージの配信状態を監視および管理するツールを装備している必要があります。管理者には、システムリソースを最適化し、システムパフォーマンスを調整する能力が必要です。

集中 (MOM) メッセージング

Message Queue では、図 1-1 に示すような集中メッセージングシステムを採用しています。このようなシステムでは、各メッセージングコンポーネントが 1 つの中央メッセージサービスとのコネクションを維持します。各コンポーネントは、きめ細かに定義されたインタフェースを介してメッセージサービスと対話します。

一方、図の左側に示すようなピアツーピアシステムでは、すべてのメッセージングコンポーネントそれぞれが他のすべてのコンポーネントとのコネクションを維持します。ピアツーピアシステムでは、高速かつ安全に信頼性の高い方法で配信できますが、コンポーネントごとに信頼性やセキュリティを確保するためのコードを備える必要があります。メッセージの送信と受信が緊密に結び付いているので、非同期で配信することも困難です。システムにコンポーネントが追加されると、コネクション数は急激に増加することになるので、システムの拡張性はすぐに尽きてしまいます。集中管理する方法でも、ピアツーピアシステムの場合には問題が残ります。

図 1-1　集中メッセージングとピアツーピアメッセージング



エンタープライズメッセージングの実現手段に適した集中システムでは、メッセージサービスによってコンポーネント間のメッセージのルーティングと配信を行います。メッセージサービスには、信頼性の高い配信処理を実行し、セキュリティを確保する役割があります。このシステムのコンポーネントは疎結合なので、非同期メッセージングを実現しやすい形態になっています。

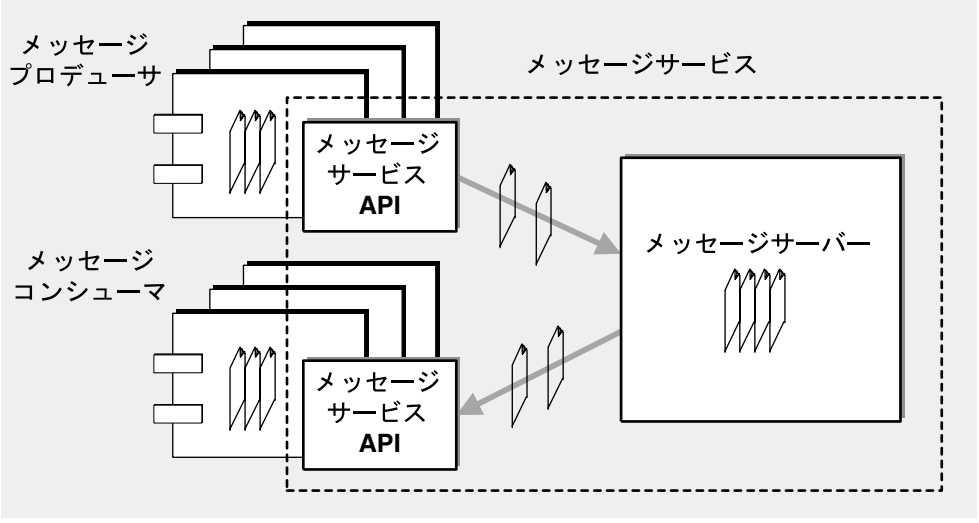
メッセージコンポーネントがシステムに追加されても、コネクション数は一定の割合で増加するだけなので、メッセージサービスを拡張すればシステムも容易に拡張できます。集中メッセージサービスは、メッセージングクライアントを接続することのほかに管理インタフェースとしての機能も提供し、これにより、動作の設定、パフォーマンスの監視、およびサービスの調整を行って各メッセージクライアントの必要を満たすことができます。

メッセージサービスの基本アーキテクチャ

図 1-2 に、集中メッセージングシステムの基本アーキテクチャを示します。この基本アーキテクチャは、共通メッセージサービス経由でメッセージを交換する、メッセージプロデューサ (producer) とメッセージコンシューマ (consumer) で構成されています。メッセージプロデューサとメッセージコンシューマは、同一メッセージングコンポーネント (またはアプリケーション) 内に複数存在できます。

メッセージプロデューサは、メッセージサービスプログラミング API を使用して、メッセージをメッセージサーバー (message server) に送信します。メッセージサーバーは、メッセージルーティングと配信コンポーネントを利用して、メッセージの配信対象として登録されている 1 つ以上のメッセージコンシューマにメッセージを配信します。コンシューマは、メッセージサービスプログラミング API を使用してメッセージを受信します。メッセージサービスは、すべての適切なコンシューマへのメッセージの配信を保証する役割を担います。

図 1-2　メッセージサービスのアーキテクチャ



このプロセスは、郵便物のやり取りを例にして説明できます。郵便物の宛先には最終的な受取人の住所を記入しますが、郵便物は郵便局を通して配送され、受取人がその郵便物をポストから取り出すまでには、その中間地点で何度か保管されることになります。

Java Message Service (JMS) の基本

Message Queue は、Java Message Service (JMS) オープン標準を実装するエンタープライズメッセージングシステム、JMS プロバイダ (JMS provider) です。そのため、JMS の概念は Message Queue サービスの仕組みを理解する上で欠かすことができません。

JMS 仕様には、信頼性の高い非同期メッセージングの動作を制御する、規則とセマンティクスのセットが規定されています。この仕様では、メッセージ構造、プログラミングモデル、および API を定義しています。

この節では、このマニュアルを理解するのに必要な JMS の概念や用語を説明します。次のトピックが含まれます。

- 25 ページの「JMS メッセージ構造」
- 27 ページの「JMS プログラミングモデル」
- 30 ページの「信頼性の高いメッセージング」
- 33 ページの「JMS 管理対象オブジェクト」

JMS メッセージ構造

Message Queue のデータは、JMS メッセージを使用して交換されます。JMS 仕様に準じて、プロデューシングクライアントによって作成される JMS メッセージは、ヘッダー、プロパティ、および本体の 3 つの部分で構成されています。

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。ヘッダーフィールドには、メッセージのルーティングと識別に使用する値が入力されています。

ヘッダーの値は次の複数の方法で設定できます。

- メッセージの作成または配信中に、JMS プロバイダによって自動的に設定する
- メッセージプロデューサが作成されたときに指定された設定値を使用して、プロデューシングクライアントが設定する
- メッセージごとにプロデューシングクライアントが設定する

JMS によって定義されるヘッダーフィールドについての詳細は、『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。これらのヘッダーフィールドにより、メッセージの送信先、有効期限、優先度などを定義できます。

プロパティ

メッセージには、プロパティというヘッダーフィールドをオプションで追加できます。プロパティフィールドは、プロパティ名とプロパティ値のペアで指定されます。プロパティは、メッセージヘッダーの延長と見なすことができ、データを作成したプロセス名、作成された時刻、および各データの構造に関する情報が入力されています。JMS プロバイダは、圧縮の有無や存続終了時の廃棄方法など、メッセージの処理方法を判別するためのプロパティを追加することもあります。

JMS プロバイダは、セレクタとしてメッセージプロパティを使用し、メッセージのソートとルーティングを実行できます。プロデューシングクライアントは、アプリケーション固有のプロパティをメッセージに設定することができ、コンシューミングクライアントは、プロパティに特定の値が含まれるメッセージだけを受け取る方法を選択できます。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のみが必要であることを指示することが可能です。指定された選択条件に一致しないメッセージは、クライアントに配信されません。

セレクタは、コンシューミングクライアントの動作を単純化し、メッセージが不要なクライアントへの配信メッセージによるオーバーヘッドを解消します。ただし、メッセージサービスでは選択条件を処理する必要があるので、オーバーヘッドが若干増加します。メッセージセレクタの構文とセマンティクスについては、JMS の仕様書で解説されています。

メッセージ本体のタイプ

JMS メッセージのタイプにより、表 1-1 の規定に従ってメッセージ本体の内容が決まります。

表 1-1 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。

JMS プログラミングモデル

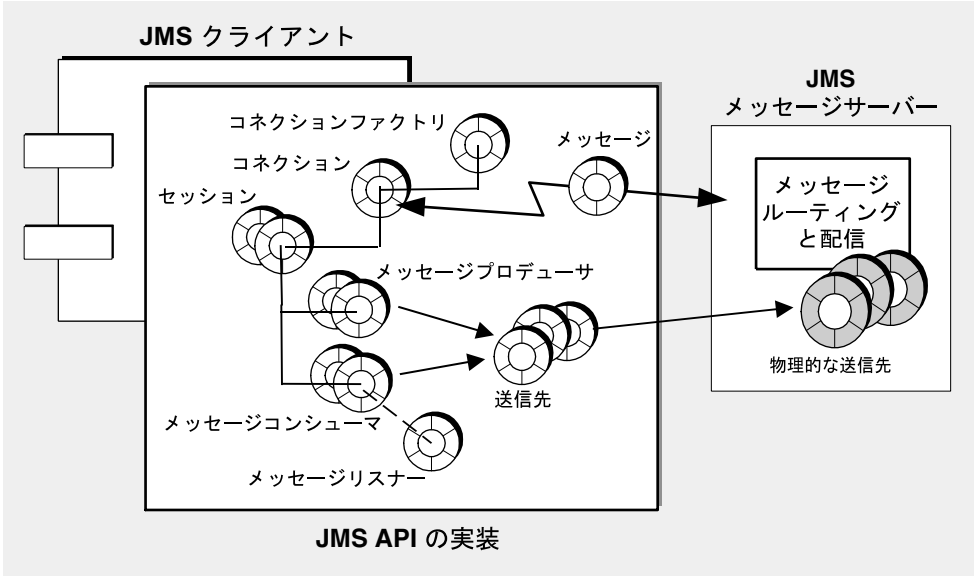
JMS プログラミングモデルは、非同期メッセージングサービスのアーキテクチャをサポートしており、JMS クライアントは、JMS メッセージサービスによってメッセージを交換します。JMS プロバイダは、JMS メッセージングを実行するために必要なオブジェクトを用意します。これらのオブジェクトは、JMS アプリケーションプログラミングインタフェース (API) を実装します。

この節では、JMS メッセージングに必要なプログラミングオブジェクトについて説明し、メッセージの送受信に使用する配信モデル (ポイントツーポイントおよびパブリッシュ / サブスクライブ) を紹介します。

プログラミングオブジェクト

図 1-3 に、JMS クライアントをセットアップしてメッセージを配信するために使用するオブジェクトを示します。

図 1-3　JMS プログラミングオブジェクト



JMS プログラミングモデルでの JMS 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、JMS メッセージサーバーとの間でメッセージを送受信するときに使用するコネクション (connection) を確立します。コネクションオブジェクト (`Connection`) は、メッセージサーバーに対するクライアントのアクティブなコネクションです。

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、セッション (session) オブジェクト (Session) の作成に使用されます。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。このコンテキストは、メッセージの作成と、メッセージの送受信を行う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を作成するために使用され、配信するメッセージの順番を定義します。セッションは、多数の通知オプションまたはトランザクションを使って、信頼性の高い配信処理をサポートします。

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、API の中の送信先オブジェクトによって示される特定の物理的な送信先 (destination) にメッセージを送信します。メッセージプロデューサは、配信モード (持続と非持続)、優先度、生存期間など、プロデューサが物理的な送信先に送信する全メッセージを制御するデフォルトのメッセージヘッダー値を指定できます。

同様にクライアントは、メッセージコンシューマオブジェクト (MessageConsumer) を使用して、API の中の送信先オブジェクトに示される特定の物理的な送信先からメッセージを受信します。送信先のタイプは、キュー (queue) とトピック (topic) の 2 種類あり、メッセージ配信モデルによって決まります。

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。

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。
- 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。

プログラミングドメイン : メッセージ配信モデル

JMS は、ポイントツーポイントとパブリッシュ / サブスクライブという 2 つの異なるメッセージ配信モデル (delivery model) をサポートしています。

ポイントツーポイント (キュー送信先): メッセージは、1 つのプロデューサから 1 つのコンシューマに配信されます。この配信モデルの送信先タイプはキューです。メッセージは、最初にキュー送信先に配信され、次にそのキューから、1 回に 1 つずつ、そのキューに対して登録されたコンシューマのいずれかに配信されます。キュー送信先にメッセージを送

信できるプロデューサの数に制限はありませんが、個々のメッセージは、1 つのコンシューマだけに向けて配信され、コンシュームされることが保証されています。キュー送信先のコンシューマが 1 つも登録されていない場合、キューは受信したメッセージを保持し、キュー送信先に対してコンシューマが登録されたときにそのメッセージを配信します。

パブリッシュ / サブスクライブ (トピック送信先): メッセージは、1 つのプロデューサから複数のコンシューマに配信されます。この配信モデルの送信先タイプはトピックです。メッセージは最初にトピック送信先に配信され、次にトピックに加入した、すべてのアクティブなコンシューマに配信されます。任意の数のプロデューサが 1 つのトピック送信先にメッセージを送信することができ、各メッセージは任意の数の加入済みコンシューマに配信できます。

トピック送信先は、永続サブスクリプションもサポートします。永続サブスクリプションは、トピック送信先に登録されていても、メッセージが配信されたときに非アクティブになっている可能性があるコンシューマを表しています。コンシューマは、もう一度アクティブになったときにメッセージを受け取ります。トピック送信先として登録されたコンシューマがない場合、トピックは、永続サブスクリプション状態の非アクティブコンシューマ宛のメッセージを保持するだけです。

この 2 つのメッセージ配信モデルは、セマンティクスが若干異なる API オブジェクトの 3 つの組み合わせを使用して処理され、表 1-2 に示すように異なるプログラミングドメイン (domain) を表しています。

表 1-2 JMS プログラミングドメインとオブジェクト

基本タイプ (統一ドメイン)	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。

統一ドメインは、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 を使用することには、たとえばキュー送信先の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という利点もあり

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、統一ドメイン API を選択してください。

Message Queue 製品に付属するアプリケーション例や、Message Queue のマニュアルに掲載されるコードの多くでは、別々のプログラミングドメインを使用しています。

信頼性の高いメッセージング

メッセージの配信モード (delivery mode) は、持続モードか非持続モードのいずれかに設定できます。配信モードにより、メッセージ配信の信頼性が決まります。

- 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。持続メッセージでは、信頼性が最優先されます。
- 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すると消失します。非持続メッセージの場合は、信頼性はそれほど重要ではありません。

持続メッセージの場合、信頼性の保証には 2 つの側面があります。1 つは、通知とトランザクションを使用することにより、メッセージの作成とコンシュームが正常に実行されるようにすることです。もう 1 つは、持続ストアにメッセージを格納することにより、コンシューマに配信される前にメッセージサービスが持続メッセージを失うことがないようにすることです。

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 / トランザクション

信頼性の高いメッセージングは、メッセージプロデューサからメッセージサーバーの物理的な送信先への、および物理的な送信先からメッセージコンシューマへの持続メッセージの正常配信の保証に依存しています。この信頼性は、JMS セッションによってサポートされる 2 つの一般的なメカニズムである、通知およびトランザクション (transaction) のいずれかを使用することによって確保されます。トランザクションの場合、分散トランザクションマネージャに制御されている状態では、ローカルまたは分散のどちらかになる可能性があります。

通知

通知は、信頼性の高い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。

メッセージが作成される場合、メッセージサービスは、配信されたメッセージを受信して送信先に格納し、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、通知が返されるまでブロックします。

メッセージがコンシュームされる場合、クライアントは、メッセージサービス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て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。それらのモードの一部では、クライアントがブロックし、メッセージを削除したためにクライアントがそのメッセージを再配信できないことをメッセージサーバーが確認するまで待機します。

ローカルトランザクション

セッションを処理済みとして設定することもできます。この場合は、1 つ以上のメッセージのプロデュースおよびコンシュームが、トランザクションという極小の単位にグループ化されます。JMS API には、トランザクションを起動、確定、およびロールバックする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。

ローカル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、単一のローカル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。言い換えると、送信先へのメッセージの配信とそれに続くクライアントへのメッセージの配信は、同じトランザクション内に入れることができません。

分散トランザクション

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といった複数のリソースマネージャ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と分散トランザクションマネージャ間の対話は、JTA の仕様で記述されます。

分散トランザクションをサポートするということは、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 つの面は、メッセージサービス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、持続メッセージが物理的な送信先に到達したら、メッセージサーバーはそのメッセージを持続データストアに保管する必要があります。何かの理由でメッセージサーバーが停止した場合、持続データストアはメッセージを復元し、適切なコンシューマに配信します。

メッセージサーバーは、永続サブスクリプションも持続的に格納する必要があります。持続的に格納しないと、障害が発生した場合、メッセージサーバーはメッセージがトピック送信先に到達したあとにアクティブになる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。

JMS 管理対象オブジェクト

JMS プログラミングモデルで使用されるオブジェクトの中の 2 つであるコネクションファクトリと送信先は、プロバイダの JMS 仕様の実装方法によって異なります。

- コネクションファクトリオブジェクトは、プロバイダがメッセージを配信する際に使用するプロトコルとメカニズムによって動作が決まるコネクションを作成するために使用します。
- 送信先オブジェクトは、ブローカの物理的な送信先の名前を指定するために使用し、メッセージサーバーの物理的な送信先に固有の命名規則と機能によって異なります。

クライアントの移植性を維持しながら、プロバイダがこれらのオブジェクトを定義する場合の柔軟性を最大限に引き出せるようにするため、JMS 仕様では、プロバイダ固有情報をカプセル化する管理対象オブジェクトをコネクションファクトリと送信先用に定義しています。これらのオブジェクトは、管理者によって設定され、JNDI ネームスペース (オブジェクトストア) に格納されます。クライアントは、標準 JNDI 検索コードを介して管理対象オブジェクトにアクセスします。

管理対象オブジェクトにより、JMS クライアントはプロバイダ固有オブジェクトを検索および参照するための論理名を使用できます。この方法では、クライアントコードで、特定の命名構文やアドレス指定構文、またはプロバイダによって使用される設定可能なプロパティについて意識する必要はありません。これにより、クライアントコードはプロバイダに依存しなくなります。

42 ページの「管理対象オブジェクト」には、Message Queue で使用される管理対象オブジェクトに関する補足情報があります。

注　JMS 仕様では、JNDI 検索を使用して管理対象オブジェクトにアクセスする必要はありません。クライアントコードにより、コネクションファクトリオブジェクトと送信先オブジェクトをインスタンス化し、その属性の値を設定できます。ただし、クライアントコードは他のプロバイダに移植できません。

第 2 章

Message Queue の紹介

Message Queue は、JMS 1.1 仕様に準拠した信頼性の高い非同期メッセージングサービスです。また Message Queue には、大規模なエンタープライズ配備の必要を満たすために、JMS 仕様で定められた要件以上の機能も多数用意されています。

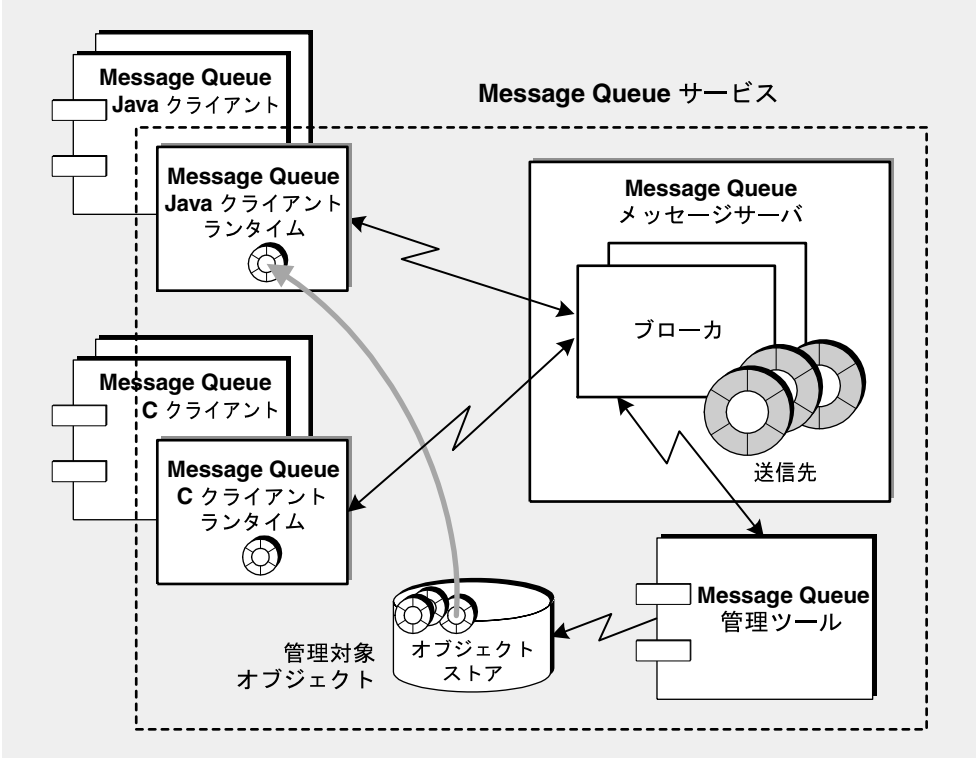
この章では、Message Queue サービスのアーキテクチャについて説明し、企業向けのさまざまな特長や機能を紹介します。章には次のトピックが含まれます。

メッセージサービスのアーキテクチャ

Message Queue サービスは、以下の要素で構成されています。

図 2-1 に、これらのエレメントの連携動作の仕組みを示します。

図 2-1　Message Queue サービスのアーキテクチャ



図に示されるとおり、Message Queue クライアントは、Java または C API を使用してメッセージを送受信します。これらの API は、Java または C クライアントのランタイムライブラリに実装されます。ランタイムライブラリは、ブローカへのコネクションを作成する実際の動作を行い、要求されるコネクションサービスに合わせて適切にパッケージします。アプリケーションが管理対象オブジェクトを使用すると、クライアントランタイムはオブジェクトストアの中からその管理対象オブジェクトを見つけ、管理対象オブジェクトを使用してコネクションを設定し、物理的な送信先を特定します。ブローカは、メッセージのルーティングと配信を行います。管理者は、Message Queue 管理ツールを使用してブローカを管理し、管理対象オブジェクトをオブジェクトストアに追加します。

これらの各要素については、以下の節で簡単に説明します。

メッセージサーバー

メッセージサーバーは、1 つ以上のブローカで構成され、メッセージのルーティングと配信を実行します。Message Queue サービスの中核となる構成要素です。

メッセージサーバーは、メッセージのルーティングサービスと配信サービスを実行する、1 つのブローカまたはブローカクラスタとして連携動作するブローカのセットで構成されています。ブローカは、以下のタスクを実行するプロセスです。

- ユーザーの認証およびユーザーが実行しようとする操作の承認
- クライアントとの通信チャネルの設定
- プロデューシングクライアントからメッセージを受信して、それぞれの物理的な送信先に配置
- 1 つ以上のコンシューミングクライアントへのメッセージのルーティングと配信
- 信頼性の高い配信の保証
- システムパフォーマンスを監視する場合のデータの提供

メッセージサーバー、メッセージサーバーの内部コンポーネント、および実行可能な機能についての詳細は、69 ページの第 4 章「メッセージサーバー」を参照してください。

Message Queue Enterprise Edition は、連結された複数のブローカインスタンスによって構成されるブローカクラスタをサポートしており、これによりメッセージサーバーは、メッセージトラフィックの量に応じて拡張または縮小することができます。アーキテクチャとクラスタ構成の問題の詳細は、第 5 章「ブローカクラスタ」を参照してください。

クライアントランタイム

Message Queue クライアントランタイム (client runtime) は、Message Queue サービスへのインタフェースを備えるクライアントアプリケーションです。クライアントランタイムは、Message Queue クライアントがメッセージをプロデュース (送信先にメッセージを送信) し、メッセージをコンシューム (送信先からメッセージを受信) する場合に必要なすべての操作をサポートします。

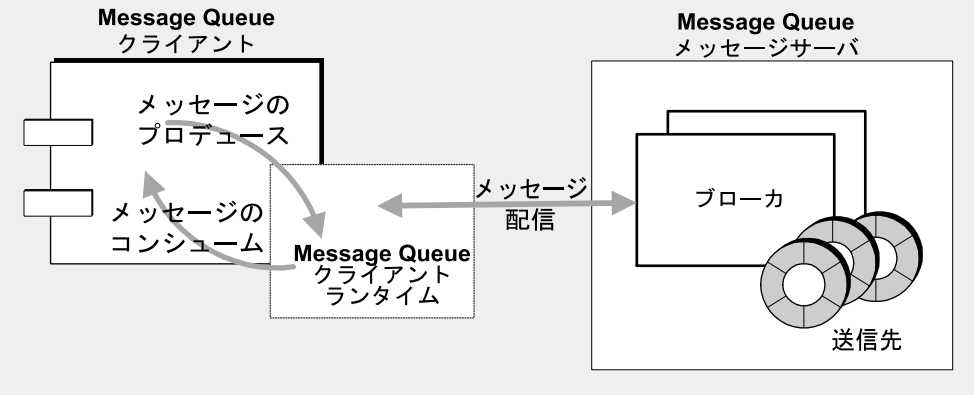
36 ページの図 2-1 に示すように、Message Queue クライアントランタイムは次の 2 種類の言語で実装されています。

- **Java クライアントランタイム** : JMS API を実装し、Message Queue メッセージサーバーと対話するために必要なすべてのオブジェクトとともに、Java クライアントアプリケーションとコンポーネントを提供します。これらのインタフェースオブジェクトには、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** : Message Queue サーバーと対話するために必要な C プログラミングインタフェースとともに、C クライアントアプリケーションとコンポーネントを提供します。C クライアントランタイムは、JMS API メッセージングモデルのプロシージャ版をサポートします。

Message Queue クライアントとメッセージサーバーの間でクライアントランタイムが担う中心的役割を、図 2-2 に示します。メッセージの配信がクライアントランタイムとメッセージサーバーの間の対話動作であるのに対し、メッセージのプロデュースおよびコンシュームは、クライアントとクライアントランタイムの間の対話動作です。

図 2-2　クライアントランタイムとメッセージング操作



クライアントランタイムは以下の機能を実行します。

- メッセージサーバーへのメッセージ配信を管理する。
- コネクションを設定する。
- クライアントの識別情報を確立する。
- クライアント通知を実装する。
- コネクション全体でメッセージのフローを制御する。
- プロデューシングクライアントが設定したメッセージヘッダの値をオーバーライドする。

以下の項では、クライアントランタイムの機能について簡単に説明します。クライアントランタイムの何種類かの動作は、コネクションファクトリオブジェクトのプロパティを設定することによりカスタマイズできます。

コネクション処理

コネクション処理動作を設定するには、クライアントの接続先にするブローカのホスト名とポート、および使用するコネクションサービスのタイプを指定する必要があります。クラスタを構成するブローカとの間でコネクションを確立する場合には、コネクションの確立先となるアドレスの一覧を指定する必要があります。ブローカがオフラインの場合は、クライアントランタイムによってクラスタ内の別のブローカに接続されます。

Enterprise Edition では、コネクションに失敗すると、ブローカへの再接続がクライアントランタイムによって自動的に実行されます。クラスタを構成するブローカにクライアントが接続されている場合は、同じブローカに再接続するか、最初のコネクションとは異なるブローカに接続できます。

ブローカインスタンスが、Message Queue と Sun Cluster を統合することで実現できる高可用性共有持続ストアを使用しない場合は、異なるブローカインスタンスに再接続すると、障害が発生したり切断されたりしたブローカが保持している持続メッセージや他の状態情報は消失する可能性があります。つまり、再接続によってコネクションはフェイルオーバーしますが、データは失われます。

クライアントの識別

アプリケーションで使用すると有効な場合には、どのコネクションにでもクライアント ID を設定できます。クライアント ID は、永続サブスクライバを識別できるように設定する必要があります。

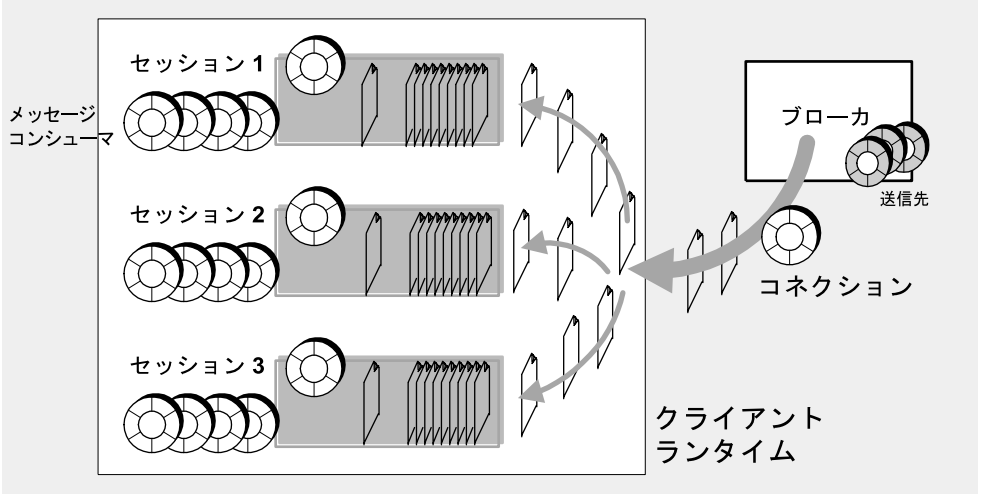
永続サブスクリプションを追跡し続けるため、ブローカは一意のクライアント識別情報を使用します。クライアント ID を使用して、メッセージがトピック送信先に配信されたときに非アクティブ状態になっている永続サブスクライバを識別します。ブローカは、非アクティブのサブスクライバ宛のメッセージを保持し、そのサブスクライバがアクティブになったら配信できるようにします。

したがって、配置されたアプリケーションで永続サブスクリプションを使用する場合はいつでも、クライアント ID を設定する必要があります。Message Queue 機能により、クライアント ID を指定するときに特殊な変数名構文を使用できます。これにより、管理者またはプログラムのどちらがオブジェクトを作成した場合でも、コネクションファクトリオブジェクトから入手したコネクションごとに、異なるクライアント ID を入手することが可能です。詳細は、『Message Queue 管理ガイド』を参照してください。

コンシューマへのメッセージ分散

コネクションを介してブローカが配信したメッセージは、クライアントランタイムが受信し、適切な Message Queue セッションに分散されます。メッセージは、40 ページの図 2-3 に示すようにキューに入れられ、それぞれのメッセージコンシューマによってコンシュームされるのを待ちます。

図 2-3　Message Queue クライアントランタイムへのメッセージの配信



メッセージは、各セッションキューから 1 つずつフェッチされ、receive() メソッドを呼び出すクライアントスレッドにより同期的に、またはメッセージリスナオブジェクトの onMessage() メソッドを呼び出すセッションスレッドにより非同期的にコンシュームされます (セッションはシングルスレッド)。

クライアントランタイムに配信されるメッセージのフローは、コンシューマレベルで測定されます。コネクションファクトリのプロパティを適切に調整することにより、メッセージのフローのバランスをとり、1 つのセッションに配信されたメッセージが、同じコネクションの他のセッションへのメッセージ配信に悪影響を与えることのないようにします。

信頼性の高い確実なメッセージ配信

クライアントランタイムには、信頼性の高い方法で確実にメッセージを配信するという重要な役割があります。クライアントランタイムは、JMS 仕様のクライアント通知機能とトランザクションモードをサポートし、信頼性の高い配信を保証するためのさまざまなブローカ通知動作を制御します。

JMS 仕様では、さまざまな信頼性レベルを提供する多くのクライアント通知モードについて規定しています。これらの通知モード、および Message Queue によって実装される追加モードについては、メッセージのコンシュームに関する節で説明します (63 ページの「クライアント通知」を参照)。

持続メッセージおよび信頼性の高い配信の場合、ブローカは通常、一度だけかつ確実にメッセージをコンシュームするために使用した操作が完了すると、クライアントランタイムに通知します。コネクションファクトリのプロパティを使用してそのようなブローカ通知を抑制し、それによってネットワーク帯域幅と処理を節約することが可能です。当然のことですが、ブローカ通知を抑制すると信頼性の高い配信は保証されなくなります。

メッセージフロー制御

クライアントランタイムは、コネクション全体のメッセージのフローを見張るゲートキーパーです。Message Queue は、コネクション全体を流れる正規の JMS ペイロードメッセージ以外にもさまざまなコントロールメッセージを送信して、信頼性の高い配信を保証し、コネクション全体でのメッセージのフローを管理し、他の制御機能を実行します。

ペイロードメッセージとコントロールメッセージは同じコネクションで競合するので、衝突して停滞を引き起こす可能性があります。クライアントランタイムは、さまざまな設定可能フロー制限と測定手段を強制的に適用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ます。

メッセージのヘッダー値のオーバーライド

クライアントランタイムは、メッセージの持続性、生存期間、および優先度を指定する JMS メッセージヘッダーフィールドをオーバーライドできます。

Message Queue は、コネクションレベルでメッセージヘッダーのオーバーライドを許可します。オーバーライドは、所定のコネクションのコンテキストでプロデュースされるすべてのメッセージに適用されます。

クライアントランタイムがメッセージヘッダー値をオーバーライドする機能を備えているので、Message Queue 管理者は、メッセージサーバーのリソースをより細かく制御できます。ただし、これらのフィールドをオーバーライドすることには、メッセージの持続性などのアプリケーション固有の要件を阻害する危険性が伴います。したがってこの機能は、適当なアプリケーションユーザーまたは設計者に相談した上でのみ使用してください。

その他の機能

クライアントランタイムは、ほかに以下の機能を実行します。

- **キューのブラウズ特性** : クライアントランタイムには、キュー送信先の内容をブラウズする場合に一度に取得するメッセージの数、およびメッセージ送信までの待ち時間を設定できます。

- **メッセージ圧縮** : Java クライアントランタイムは、メッセージのプロデュース時にメッセージを圧縮し、メッセージのコンシューム時にメッセージを解凍することができます。圧縮または解凍処理を実行するかどうかは、クライアントがメッセージを作成するときにメッセージヘッダーに設定される、Message Queue 固有のメッセージプロパティによって決まります。

管理対象オブジェクト

管理対象オブジェクトは、プロバイダ固有の実装情報と、コネクションおよび送信先に関する設定情報をカプセル化します。管理対象オブジェクトは、プログラムによって作成するか、管理者ツールを使用して作成および設定し、オブジェクトストアに保管して、標準 JNDI 検索コードによりクライアントアプリケーションからアクセスできます。

Message Queue には、以下の表に示す管理対象オブジェクトタイプが用意されています。

表 2-1 Message Queue 管理対象オブジェクトタイプ

タイプ	説明
`Destination`	ブローカ内の物理的な送信先を表します。ブローカ内の物理的な送信先のプロバイダ固有名が収められています。メッセージコンシューマまたはメッセージプロデューサのオブジェクトは、送信先管理対象オブジェクトを使用して対応する物理的な送信先にアクセスします。
`Connection Factory`	クライアントアプリケーションと Message Queue メッセージサーバーの間で物理的な接続を確立します。また、物理的な接続の動作を制御する Message Queue クライアントランタイムも設定します。コネクションファクトリ管理対象オブジェクトの属性値を設定する場合は、そのオブジェクトが確立するすべてのコネクションに適用されるプロパティを指定します。
`XA Connection Factory`	分散トランザクションをサポートする物理的な接続を確立するために使用します (31 ページの「分散トランザクション」を参照)。XA コネクションファクトリオブジェクトは、正規のコネクションファクトリオブジェクトと同じ属性セットを共有しますが、分散トランザクションに対応するために必要な補足メカニズムを使用可能にします。

表 2-1　Message Queue 管理対象オブジェクトタイプ (続き)

タイプ	説明
SOAP Endpoint	SOAP メッセージの最終送信先を特定します。これは、SOAP メッセージを受信可能なサーブレットの URL です。SOAP 終端管理対象オブジェクトは、複数の URL を指定して設定可能です。また、オブジェクトに関連付けられた検索名とオブジェクトストア属性を指定します。

JNDI を介しての管理対象オブジェクトの使用

JMS 仕様では、JMS クライアントが JNDI ネームスペースの管理対象オブジェクトを検索するよう規定していませんが、管理対象オブジェクトを検索することにははっきりとした利点があります。たとえば、一元管理を行い、レコードがなくてもコネクション (クライアントランタイムの動作) を設定および再設定し、クライアントを他の JMS プロバイダに移植可能にすることができます。

管理対象オブジェクトにより、次のように Message Queue サービスを容易に制御および管理できます。

- 管理者は、クライアントアプリケーションをあらかじめ設定済みのコネクションファクトリオブジェクトにアクセスさせることにより、クライアントランタイムの動作を指定できます。
- 管理者は、クライアントアプリケーションを既存の物理的な送信先に対応する設定済みの送信先管理対象オブジェクトにアクセスさせることにより、物理的な送信先の拡散を制御できます。

言い換えると、管理対象オブジェクトを使用することにより、Message Queue の管理者は、メッセージサービスの詳細設定を制御しながら、同時にクライアントアプリケーションをプロバイダ非依存にすることができます。

管理対象オブジェクトを使用することにより、クライアントプログラマは、プロバイダ固有の構文とオブジェクト命名規則、またはプロバイダ固有の設定プロパティを把握している必要がなくなります。実際、管理対象オブジェクトを読み取り専用に指定することにより、管理者は、管理対象オブジェクトが最初に作成されたときに設定した管理対象オブジェクトの属性値を、クライアントアプリケーションが変更できないようにすることができます。

クライアントアプリケーションでは、所有する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両方をインスタンス化することができますが、この手法は管理対象オブジェクトの基本目的を損ないます。Message Queue 管理者は、アプリケーションによって要求されるブローカリソースを制御し、メッセージングのパフォーマンスを調整する必要があります。また、管理対象オブジェクトを直接インスタンス化すると、クライアントアプリケーションがプロバイダに依存したものになります。

管理対象オブジェクトについては上述のとおりですが、管理制御が問題にならない開発環境では、多くの場合アプリケーションによって管理対象オブジェクトがインスタンス化されます。

オブジェクトストア

Message Queue 管理対象オブジェクトはオブジェクトストアに保管され (36 ページの図 2-1 を参照)、クライアントアプリケーションは JNDI 検索を行なってオブジェクトストアに保管された管理対象オブジェクトにアクセスできます。Message Queue は、標準 LDAP ディレクトリサーバーとファイルシステムオブジェクトストアの 2 つのオブジェクトストアタイプをサポートします。

LDAP サーバーオブジェクトストア : LDAP サーバーは、運用メッセージングシステム用のオブジェクトストアとしてお勧めします。LDAP 実装は、多数のベンダーでサポートされており、分散システムでの使用を考慮した設計になっています。LDAP サーバーは、運用環境で役立つセキュリティ機能も備えています。

ファイルシステムオブジェクトストア : ファイルシステムのオブジェクトストアは、運用システムでの使用はお勧めしませんが、開発環境で使いやすいという利点があり、Message Queue でもサポートされています。LDAP サーバーをセットアップする必要はなく、ローカルのファイルシステム上にディレクトリを作成するだけで利用できます。ただし、複数のコンピュータノードにまたがって配備されたクライアントの集中オブジェクトストアとして、ファイルシステムのオブジェクトストアを使用できるのは、これらのクライアントが、オブジェクトストアの常駐するディレクトリに対してアクセス権を持っている場合に限られます。

管理ツール

Message Queue 管理ツールは、一連のコマンド行ユーティリティとグラフィカルユーザーインタフェース (GUI) 管理コンソールで構成されています。

コマンド行ユーティリティ : Message Queue は、ブローカの起動と管理、物理的な送信先の作成と管理、管理対象オブジェクトの管理、より特殊なその他の管理タスクの実行など、すべての Message Queue 管理タスクを実行するコマンド行ユーティリティを装備しています。すべてのコマンド行ユーティリティは、共通の形式、構文規則、およびオプションを共有します。コマンド行ユーティリティの使用法の詳細は、『Message Queue 管理ガイド』を参照してください。

管理コンソール : 管理コンソールは、Message Queue コマンド行ユーティリティの機能の一部を提供します。管理コンソールを使用して、ブローカの管理、物理的な送信先の作成と管理、および管理対象オブジェクトの管理を行うことができます。ただし、コマンド行ユーティリティが提供するいくつかの特殊なタスクは実行できません。たとえば、管理コンソールを使用して、ブローカを起動したり、ブローカクラスタを作成したり、ユーザーリポジトリを管理したりすることはできません。これらのタスクを実行するには、Message Queue のコマンド行ユーティリティを使用する必要があります。

『Message Queue 管理ガイド』には、管理コンソールについて解説し、基本的なタスクを実行する場合の管理コンソールの使用法を示した、簡単で実践的なチュートリアルが用意されています。

管理コンソールおよびコマンド行ユーティリティの一部を使用することにより、ブローカおよび物理的な送信先をリモート管理することができます。

製品の機能

Message Queue サービスおよび前の節で説明したアーキテクチャでは、柔軟で信頼性の高い非同期メッセージ配信に関する JMS 1.1 仕様を全面的に実装しています。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 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。

以下で説明する Message Queue のエンタープライズ機能は、次のカテゴリに分類されています。

統合サポート機能

Message Queue を使用して、数種類のトランスポートプロトコルのサポート、Message Queue サービスへの C クライアントインタフェース、SOAP (XML) メッセージのサポート、および接続可能 J2EE リソースアダプタを組み込むことにより、エンタープライズ全体で異なるアプリケーションとコンポーネントを統合することができます。

複数トランスポートのサポート

Message Queue は、TCP や HTTP などの異なる多くのトランスポートプロトコルを介し、セキュリティ保護コネクションを使用して、クライアントが Message Queue メッセージサーバーと対話する機能をサポートしています。

HTTP コネクション: 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、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。同様に Message Queue は、HTTPS トンネルサーブレットを使用してセキュリティ保護された HTTP コネクションをサポートします。HTTP コネクションのアーキテクチャについての詳細は、73 ページの「HTTP/HTTPS サポート」を参照してください。HTTP/HTTPS コネクションの設定と構成の詳細は、『Message Queue 管理ガイド』を参照してください。

セキュリティ保護コネクション: Message Queue は、TCP/IP および HTTP トランスポートでの SSL (Secure Socket Layer) 標準に基づく、セキュリティ保護されたメッセージ送信機能を備えています。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。自己署名証明書を作成して SSL ベースのコネクションサービスを有効にする方法の詳細は、『Message Queue 管理ガイド』を参照してください。

C クライアントインタフェース

Message Queue は、Java 言語メッセージングクライアントをサポートする以外に、Message Queue サービスへの C 言語インタフェースとしても機能します。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。ただし、Message Queue の C API を使用するクライアントは、他の JMS プロバイダに移植できません。

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。

C API の機能、および C API が C データタイプと関数を使用して JMS プログラミングモデルを実装する方法の詳細は、『Message Queue Developer's Guide for C Clients』を参照してください。

SOAP (XML) メッセージングサポート

Message Queue は、シンプルオブジェクトアクセスプロトコル (Simple Object Access Protocol、SOAP) 仕様に準拠したメッセージの作成と配信をサポートします。SOAP では、非集中の分散環境にあるピア間で、構造化 XML データ、つまり SOAP メッセージを交換できます。SOAP メッセージは、XML 形式にする必要のない添付ファイルも含めることが可能な XML ドキュメントです。

SOAP メッセージは XML でエンコードされているので、SOAP のメッセージをプラットフォーム非依存にすることができます。SOAP メッセージは、旧式のシステムからデータにアクセスし、エンタープライズ間でデータを共有するために使用できます。XML に基づくデータ統合テクノロジなので、Web サービスなどの Web ベースコンピューティングとの相性も良好です。ファイアウォールは、SOAP パケットを認識し、SOAP メッセージヘッダーで開示される情報に基づいてメッセージをフィルタできます。

Message Queue は、SAAJ (Attachments API for Java) 仕様による SOAP を実装します。SAAJ は、アプリケーションプログラミングインタフェースの 1 つで、実装することにより、SOAP メッセージングのプログラミングモデルをサポートし、SOAP メッセージを作成、送信、受信、および検査するために使用可能な Java オブジェクトを提供できます。SAAJ は次の 2 つのパッケージを定義します。

- `javax.xml.soap`: このパッケージのオブジェクトを使用して、SOAP メッセージの各部を定義し、SOAP メッセージをアセンブルまたは逆アセンブルします。このパッケージを使用すれば、プロバイダのサポートなしで SOAP メッセージを送信することもできます。
- `javax.xml.messaging`: このパッケージのオブジェクトにより、プロバイダを使用して SOAP メッセージを送信し、SOAP メッセージを受信します。

Message Queue は、SOAP メッセージを JMS メッセージに変換し、またその逆の変換も実行するユーティリティを備えています。これらのユーティリティにより、SOAP メッセージがサーブレットよって受信され、JMS メッセージに変換され、Message Queue によって JMS コンシューマに配信されるようにすることができます。また、も

う一度 SOAP メッセージに変換し、SOAP 終端に配信することも可能です。言い換えると、Message Queue により、SOAP 終端の間で、信頼性の高い非同期の方法で SOAP メッセージを交換する、つまり SOAP メッセージを Message Queue サブスクライバにパブリッシュすることができます。

詳細は、『Message Queue Developer's Guide for Java Clients』を参照してください。

J2EE リソースアダプタ

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。つまり、J2EE プラットフォームでは JMS のサポートが必要となります。

JMS のサポートは、JMS メッセージをコンシューム可能な特殊な種類の EJB (Enterprise Java Bean) である MDB (メッセージ駆動型 Bean) を使用して、J2EE プログラミングモデルで提供されます。J2EE 準拠のアプリケーションサーバーは、JMS メッセージングをサポートする MDB コンテナを用意する必要があります。MDB コンテナは、JMS リソースアダプタをアプリケーションサーバーに接続して準備します。Message Queue は、そのようなリソースアダプタを装備しています。

Message Queue リソースアダプタをアプリケーションサーバーに接続することにより、アプリケーション環境内に配備されて稼働している MDB などの J2EE コンポーネントは、それらのコンポーネントどうしおよび外部 JMS コンポーネントとの間で、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の詳細は、第 6 章「Message Queue と J2EE」を参照してください。

セキュリティ機能

保管されたメッセージデータや移送中のメッセージデータを保護する機能は、ほとんどのエンタープライズアプリケーションにとって必要不可欠です。Message Queue は、ユーザー認証、リソースへのアクセス制御、メッセージの暗号化など、さまざまなレベルでのセキュリティ機能を提供します。

認証 : Message Queue は、パスワードベースのユーザー認証をサポートしています。メッセージサーバーへの接続は、単層型ファイルユーザーリポジトリまたは LDAP ユーザーリポジトリに格納されているパスワードに基づいて許可されます。すべての接続試行に関する情報 (ユーザーおよびホストコンピュータ) はログに記録され、追跡できます。

承認 : アクセス制御リスト (Access control list、ACL) により、ブローカのコネクションと物理的な送信先へのアクセスを、設定可能な方法できめ細かく制御することができます。ユーザーとグループアクセスの両方がサポートされています。承認処理はブローカ単位で実行され、各ブローカは別々のアクセス制御ファイルを保有できます。

暗号化 : SSL サポートにより、最大長の SSL 実装を使用して、メッセージサーバーとそのクライアントの間 (TCP/IP コネクションと HTTP コネクションのどちらでも) のすべてのメッセージトラフィックを暗号化できます。

ユーザーリポジトリへの入力方法、アクセス制御リストの管理方法、および SSL サポートの設定方法については、『Message Queue 管理ガイド』を参照してください。

スケーラビリティ機能

Message Queue では、ユーザー、クライアントコネクション、およびメッセージ負荷が大きくなるにつれて、アプリケーションを拡張することができます。

スケーラブルなコネクション機能

Message Queue ブローカは、数千の同時コネクションを処理できます。デフォルトでは、それぞれのコネクションが専用のブローカスレッドによって処理されます。この方式では、コネクションがアイドル状態のときでもスレッドが結合されるので、コネクションサービスを設定して、複数のコネクションが同じスレッドを共有できるようにすることが可能です。この共有スレッドプールモデルは、ブローカがサポート可能なコネクション数を大幅に増大させます。詳細は、73 ページの「スレッドプールマネージャ」を参照してください。

ブローカクラスタ

コネクション数およびブローカによって配信されるメッセージ数が増大するにつれ、余分な負荷は補足的なブローカインスタンスを Message Queue サーバーに追加することによって管理できます。ブローカクラスタにより、クライアントコネクションどうしの平衡をとり、数多くのブローカインスタンス全体における配信処理を管理して、メッセージサーバーのスケーラビリティを大幅に高めます。ブローカインスタンスは、同じホストに、またはネットワーク全体に分散して配置することができます。クラスタリングは、メッセージのスループットを向上させ、ビジネスの必要が大きくなるのに伴ってメッセージング帯域幅を拡大する場合に理想的な方法です。ブローカクラスタについては、89 ページの第 5 章「ブローカクラスタ」で説明されています。詳しくは『Message Queue 管理ガイド』を参照してください。

複数のコンシューマへのキューの配信

JMS 仕様に従って、1 つのキュー送信先にある 1 つのメッセージは、1 つのコンシューマにだけ配信することができます。Message Queue では、複数のコンシューマを 1 つのキューに登録できます。これによりブローカは、メッセージを異なる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。

複数コンシューマへのキューの配信の実装では、設定可能なロードバランス方式を使用します。この方式を使用することにより、アクティブコンシューマの最大数と、何らかの障害が発生した場合にアクティブコンシューマに入れ替えられる準備のできたバックアップコンシューマの最大数を指定できます。また、ロードバランスのメカニズムでは、コンシューマの現在の容量とメッセージの処理速度を考慮します。

ロードバランスされたキューの配信についての詳細は、61 ページの「複数のコンシューマへのキューの配信」を参照してください。

可用性機能

Message Queue は、サービスの停止時間を最小限に抑える数多くの機能を備えています。たとえば、障害を防止するメカニズムから、Sun Cluster と統合して高可用性を実現する機能まで、広範囲に及びます。

メッセージサービスの安定性

メッセージサービスの可用性を保証するもっとも有力な方法の 1 つとして、高いパフォーマンスを発揮し、障害を最小限に抑えるサービスを提供することが挙げられます。Message Queue は、メモリーのオーバーヘッドやパフォーマンスの停滞を防ぐメカニズムを備えています。このメカニズムは、メッセージサーバーとクライアントランタイムの両方で作動します。

メッセージサーバーのリソース管理：メッセージサーバーは、メモリーと CPU リソースの点で限界があるので、応答がなくなったり、動作が不安定になるほどに過負荷がかかる可能性があります。一般にこの状態は、メッセージのプロデュース速度がコンシューム速度をはるかに上回るときに発生します。そのような状況を避けるため、個々の物理的な送信先レベルとシステム全域レベルでブローカを設定し、メモリーのオーバーランを回避することができます。詳細は、77 ページの「メモリーリソース管理」を参照してください。

クライアントランタイムのメッセージフロー制御：さらに Message Queue は、クライアントランタイムへのメッセージ配信を制御するメカニズムも備えています。フロー制御メカニズムを使用することにより、クライアントランタイムへのメッセージ配信を最適化すると同時に、クライアントがメモリーを使い尽くしてしまう事態を避けることができます。詳細は、41 ページの「メッセージフロー制御」を参照してください。

メッセージサーバーへの自動再接続

Message Queue には、自動再接続機能があります。メッセージサーバーとクライアントの間のコネクションで障害が発生すると、Message Queue は、コネクションの再確立を試みながらクライアント状態を保持します。ほとんどの場合、コネクションが再確立されると、メッセージのプロデュースとコンシュームは透過的に再開されます。詳細は、『Message Queue 管理ガイド』を参照してください。

Sun Cluster による高可用性

Message Queue のブローカクラスタにより、非常にスケーラブルなメッセージサーバーを構築できますが、クラスタ内の 1 つのブローカインスタンスから別のブローカインスタンスへのフェイルオーバーは現在のところサポートされていません。ただし、Message Queue を Sun Cluster ソフトウェアと統合すれば、高可用性を備えたメッセージサーバーを構築できます。Message Queue 用に開発された Sun Cluster エージェントを使用すれば、Sun Cluster により、ブローカで障害が発生した場合でも状態データを失わずに、実質的な停止時間なしでメッセージサーバーを透過的かつ即座に復元することができます。

管理機能

Message Queue は、メッセージサービスの監視と管理を行い、メッセージサービスのパフォーマンスを調整するための機能を数多く備えています。

堅牢な管理ツール

Message Queue は、コマンド行ツールと GUI ツールの両方を備えており、これらのツールによって Message Queue メッセージサーバーを運用し、送信先、トランザクション、永続サブスクリプション、およびセキュリティを管理することができます (44 ページの「管理ツール」を参照)。

また Message Queue は、メッセージサーバーのリモート監視と管理をサポートするとともに、JMS 管理対象オブジェクト、ユーザーリポジトリ、プラグイン JDBC 準拠データストア、および自己署名サーバー証明書を管理するツールをサポートしています。管理ツールの使用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。

メッセージベースの監視 API

Message Queue では、カスタム監視アプリケーションを作成する際に使用可能な、単純な JMS ベース監視 API を使用できます。これらの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。メトリックスメッセージには、Message Queue ブローカによって提供される監視データが含まれています (83 ページの「メトリックスメッセージプロデューサ (Enterprise Edition)」を参照)。

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 クライアントを開発してメトリックスメッセージをコンシュームする方法について説明した、『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。

調整可能なパフォーマンス

Message Queue では、多くの方法でメッセージサーバーとクライアントランタイムの両方を調整し、最適なパフォーマンスを発揮することができます。主要なリソースを監視し、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他の要素を調整できます。メッセージサービスのパフォーマンスを調整する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。

柔軟なサーバー設定機能

Message Queue では、持続オブジェクト、ユーザー情報、および管理対象オブジェクトの保管方法を選択できます。

設定可能な持続性

メッセージの配信を保証するため、Message Queue は、メッセージがコンシュームされるまでメッセージと他の持続オブジェクトを保管します。優れたパフォーマンスのファイルベース持続ストアを用意するとともに、Message Queue は、設定可能な持続性もサポートします。これにより、Oracle 8i などの埋め込みまたは外部の JDBC 準拠データベースに持続メッセージを保管できます。詳細は、78 ページの「持続マネージャ」を参照してください。

LDAP サーバーのサポート

Message Queue では、認証および承認処理で必要になる管理対象オブジェクトとユーザー情報の両方を保管する、ファイルベースストレージを使用できます。また Message Queue は、管理対象オブジェクトストアとユーザーリポジトリで LDAP サーバーを使用する方法もサポートしています。LDAP サーバーでは、そのような情報を保存および取得するためのより安全で標準的な方法を提供しているので、運用システムで使用することをお勧めします。管理対象オブジェクトおよびユーザーリポジトリで LDAP サーバーを使用する場合についての詳細は、『Message Queue 管理ガイド』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。機能セットについては、次の表に比較を示します。機能についての詳細は、45 ページの「製品の機能」を参照してください。

表 2-2　機能比較 : Enterprise Edition と Platform Edition

Enterprise Edition	Platform Edition
高度統合サポート機能	
HTTP サポート、TCP サポート	TCP サポート
SSL (Secure Socket Layer) 標準に基づくセキュリティ保護コネクション	SSL (Secure Socket Layer) 標準に基づくセキュリティ保護コネクション
C 言語 API、Java API	Java API (C API はトライアルライセンスでのみ仕様可能。)
SOAP (XML) メッセージングサポート	SOAP (XML) メッセージングサポート
J2EE リソースアダプタ	J2EE リソースアダプタ
セキュリティ機能	
単層型ファイルまたは LDAP ユーザーリポジトリのいずれかからの認証、アクセス制御ファイルを使用した承認、および SSL 暗号化。	Enterprise Edition と同じ
スケーラビリティ機能	
スケーラブルなコネクション機能	固定されたコネクション機能

表 2-2　機能比較 : Enterprise Edition と Platform Edition (続き)

Enterprise Edition	Platform Edition
メッセージサーバーをブローカクラスタとして実装可能	シングルブローカメッセージサーバー
無制限のメッセージコンシューマへのキューの配信 (キューあたり)	最大 3 つのメッセージコンシューマへのキューの配信 (キューあたり)
可用性機能	
メモリーリソース管理とメッセージフロー制御によるメッセージサービスの安定性	Enterprise Edition と同じ
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、または同じブローカへの自動再接続。	クライアントコネクションフェイルオーバーなし。同じブローカへの自動再接続を許可。
Sun Cluster による高可用性	Sun Cluster による高可用性
管理機能	
堅牢な管理ツール	堅牢な管理ツール
管理ツールとログ作成に加えてのメッセージベース監視 API	管理ツールとログ作成、メッセージベース監視 API なし
パフォーマンスの調整可能	パフォーマンスの調整可能
柔軟なサーバー設定機能	
接続可能持続性	接続可能持続性
LDAP サーバーのサポート	LDAP サーバーのサポート

Platform Edition と Enterprise Edition のライセンス数については、以下の節で説明します。

Enterprise Edition

Message Queue Enterprise Edition により、企業の運用環境にメッセージングアプリケーションを配備して稼働させることができます。また Enterprise Edition を使用して、メッセージングアプリケーションやコンポーネントの開発、デバッグ、および負荷テストも実施できます。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。

Platform Edition

Message Queue Platform Edition では、メッセージサーバー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。基本ライセンスの Platform Edtion は、要件の緩い運用環境の JMS プロバイダとして使用できます。このライセンスには、Enterprise Edition の機能は含まれていません。
- **90 日間の企業向けトライアルライセンス**には、基本ライセンスには含まれていない Enterprise Edition の全機能が含まれています。ただし、ライセンスの有効期間は 90 日間であるため、Enterprise Edition の製品で提供される機能を評価するのに適しています。90 日間の企業向けトライアルライセンスの使用方法についての詳細は、『Message Queue 管理ガイド』で説明されている起動オプションを参照してください。

Platform Edition は、Sun の Web サイトから無料でダウンロードできます。また、Sun Java System Application Server プラットフォームにもバンドルされています。Message Queue を Platform Edition から Enterprise Edition にアップグレードする方法については、『Message Queue インストールガイド』を参照してください。

注 Message Queue のすべてのエディションでは、製品の一部である Message Queue クライアントランタイムが、商用として無償で再配布できます。製品の他のファイルはすべて再配布できません。一部のファイルは無償で再配布可能なので、1 つのライセンスで Message Queue サービスに接続可能な Message Queue クライアントアプリケーションを開発し、追加のライセンス料なしでサードパーティに販売することができます。サードパーティは、Message Queue を購入して Message Queue メッセージサーバーにアクセスするか、Message Queue メッセージサーバーをインストールして稼働させているさらに別のサードパーティとの間でコネクションを確立する必要があります。

Sun 製品群の中の Message Queue

Message Queue は、アプリケーションによって直接使用されるミドルウェアであるだけでなく、他のミドルウェアによっても使用され、Sun が提供する他のサーバーやアプリケーションでも使用されます。このため、Message Queue は、Solaris や Java Enterprise System で配布されると同時に、Sun Java System Application Server にも同梱されています。

Application Server に同梱されている Message Queue は、J2EE プラットフォームが JMS プロバイダに提供する JMS 要件を満たしています。この場合の Message Queue は、Application Server でホストされるアプリケーションによって直接使用されます。詳細は、95 ページの第 6 章「Message Queue と J2EE」を参照してください。

第 3 章

信頼性の高いメッセージ配信

この章では、Message Queue サービスによって実現される信頼性の高いメッセージ配信の仕組みについて説明します。説明では、システム内のメッセージの送信経路を追跡し、適切なコンシューマまでのメッセージのルーティングと配信、およびメッセージの配信を保証するために利用されているさまざまなメカニズムを取り上げます。

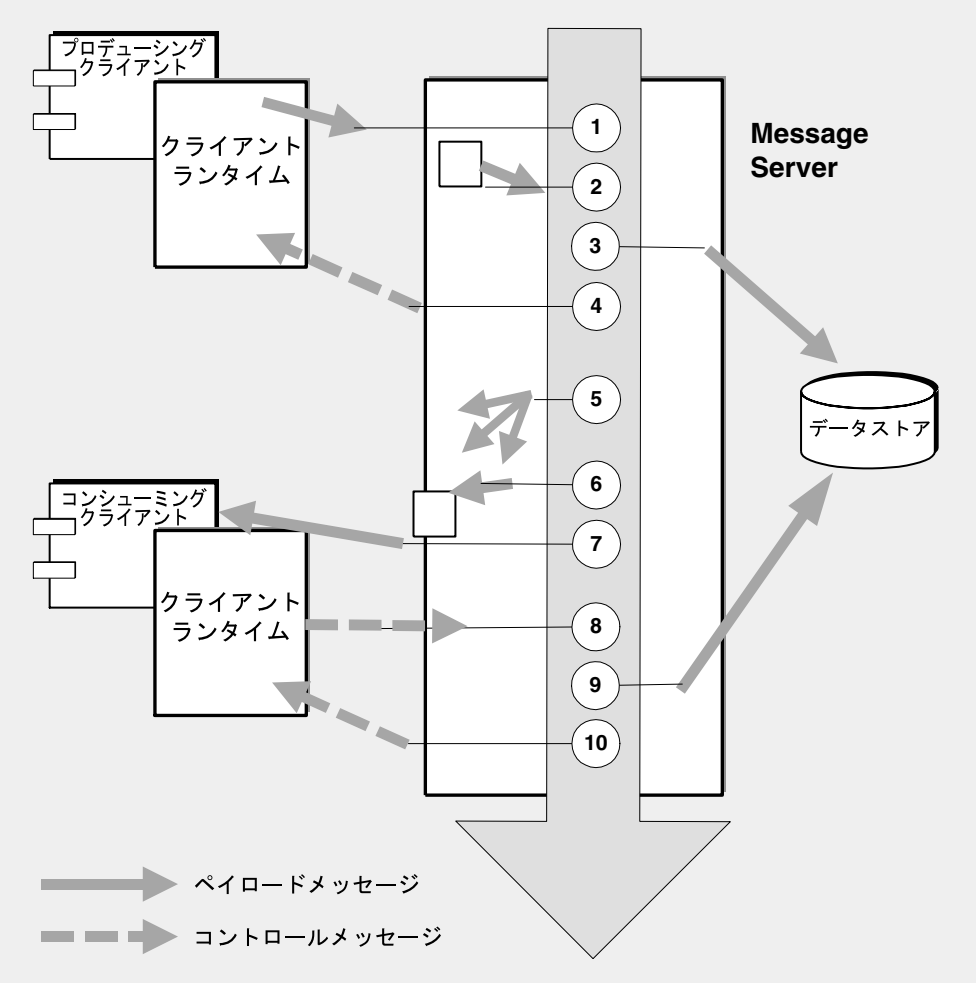
章には次のトピックが含まれます。

この章には、開発者と管理者にとって有用な資料が含まれており、第 2 章「Message Queue の紹介」の内容を補完するものとなっています。

システム全体でのメッセージの流れ

メッセージプロデューサから始まり、メッセージコンシューマで終わる、Message Queue メッセージサービスによるメッセージ配信の流れを図 3-1 に示します。以下の項では、配信プロセスの各段階について詳しく説明します。

図 3-1　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メッセージサーバー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. メッセージサーバーが、コネクションからメッセージの内容を読み込み、適切な送信先に保管します。

3. メッセージサーバーが (持続) メッセージをデータストアに保管します。

4. メッセージサーバーが、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. メッセージサーバーが、メッセージのルーティングを決定します。

6. メッセージサーバーは、送信先から適切なコネクションに、メッセージを書き出します。

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、メッセージサーバーに通知します。

メッセージの存続終了

9. メッセージサーバーはクライアント通知を処理し、送信先とデータストアの両方から (持続) メッセージを削除します。

10. メッセージサーバーは、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のでメッセージを再び配信することはできないことを確認します。

これらの一連の配信手順でシステムによって処理されるメッセージは、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** : プロデューシングクライアントにより、コンシューミングクライアントに送信される JMS メッセージ。
- **コントロールメッセージ** : メッセージサーバーとクライアントランタイムの間で送受信される通知およびその他の非ペイロードメッセージ。ペイロードメッセージが確実に配信されることを保証し、コネクション全体でのメッセージのフローを制御します。

メッセージ配信処理

Message Queue サービスによるメッセージの処理は、図 3-1 に続く手順の説明にあるように、プロデューサからコンシューマまでの配信の流れを数段階に分けて実行されます。

配信の段階は次のとおりです。

- 60 ページの「メッセージのプロデュース」
- 60 ページの「メッセージの処理とルーティング」
- 63 ページの「メッセージのコンシューム」
- 66 ページの「メッセージの存続終了」

次の項で、これらの段階について説明します。

メッセージのプロデュース

メッセージのプロデュースでは、メッセージはクライアントによって作成され、クライアントランタイムによりブローカの送信先へのコネクションを介して送信されます。

メッセージの配信モードが持続 (ブローカで障害が発生しても、必ず 1 回の配信を保証) に設定されている場合、ブローカはデフォルトでコントロールメッセージとしてブローカ通知をクライアントランタイムに返信します。このブローカ通知により、ブローカが送信先にメッセージを配信し、ブローカのデータストアに保管したことを示します。クライアントスレッドは、ブローカ通知を受け取るまでブロックします。

メッセージの配信モードが非持続に設定されている場合、ブローカはデフォルトでクライアントランタイムにブローカ通知を返信せず、クライアントスレッドはブロックしません。ただし、ブローカが非持続メッセージを受信するかどうかを知る必要がある場合は、ブローカ通知を有効にできます。実際、送信先メモリーが限界に達した場合は、ブローカ通知を有効にしてメッセージのプロデュース速度を落とす必要があります (77 ページの「送信先のメッセージ制限」を参照)。

メッセージの処理とルーティング

ブローカは、着信 JMS ペイロードメッセージを受信すると、その送信先に格納し、適切な 1 つまたは複数のコンシューマにルーティングします。

一般に、すべてのメッセージは、配信されるか期限切れになるまで物理的な送信先 (メモリー内) に残ります。ブローカで障害が発生すると、これらのメッセージは失われます。メッセージが持続的である場合、ブローカはメッセージをデータベースかファイルシステムに保管し、障害のあとに復元します。

メッセージの処理方法は、次の節で説明されるとおり、送信先のタイプ (キューまたはトピック) によって決まります。また、管理者が物理的な送信先を作成したときに、その送信先に設定された送信先プロパティによっても異なります。

キューの送信先

キューの送信先は、メッセージが配信され、1 つだけのコンシューマによってコンシュームされることになっている、ポイントツーポイントメッセージングで使用されます。

1 つのキュー送信先内のメッセージは 1 つのコンシューマにのみ配信されますが、Message Queue では、複数のコンシューマを 1 つのキューに登録できます。それからブローカは、登録された異なるコンシューマにメッセージを分配し、コンシューマ間で負荷を分散させることができます。

基本的なルーティングメカニズム

メッセージは、プロデューサから到着するとキューに入れられます。それぞれのメッセージは、キューの先頭になると、キューに登録された 1 つのコンシューマに向けてルーティングされます。メッセージがキューの先頭に来る順番は、メッセージの到着順序と優先度によって決まります。

セレクタのプロパティ値がメッセージに設定されていると、ブローカは、登録済みコンシューマによって指定されているいずれかのセレクタ値とそのプロパティ値を比較し、メッセージをコンシューマに向けてルーティングする前にセレクタ値が一致することを確認します。

複数のコンシューマへのキューの配信

複数のコンシューマへのキュー配信の実装では、多くのキュー送信先プロパティに基づいて設定可能なロードバランス方式を使用します。

- ロードバランスされたキューの配信では、アクティブにするコンシューマの最大数を設定できます。
- また、アクティブコンシューマで障害が発生した場合に置き換えることが可能な、バックアップコンシューマの最大数も設定できます。

コンシューマの数がこの 2 つのプロパティの合計値を超えた場合、新しいコンシューマは拒否されます。Message Queue Platform Edition はキューあたり最大 3 つのコンシューマ (2 つはアクティブ、1 つはバックアップ) をサポートします。Message Queue Enterprise Edition がサポートするコンシューマの数は無制限です。

ロードバランスメカニズムでは、さまざまなコンシューマがメッセージをコンシュームする度合いを考慮します。キュー送信先内のメッセージは、設定可能なサイズ (キュー送信先のコンシューマフロー制限プロパティ) にまとめられ、新たに使用可能になったアクティブコンシューマにキューに登録された順序でルートされます。これらのメッセージが配信されると、キューに到着する追加メッセージは、コンシューマが使用可能

になると同時にバッチ処理によってコンシューマにルーティングされます。コンシューマは、前に配信された設定可能な割合のメッセージをコンシュームすると使用可能になります。言い換えると、各コンシューマへのディスパッチ速度は、コンシューマの現在の容量とメッセージの処理速度によって異なります。

アクティブコンシューマに障害が生じると、1 番目のバックアップコンシューマがアクティブになり、障害の生じたコンシューマの動作を引き継ぎます。こうしたメカニズムであるため、キュー送信先に複数のアクティブコンシューマがある場合は、メッセージがコンシュームされる順序は保証されません。

メッセージの生成速度が遅い場合は、ブローカがアクティブコンシューマ間で不均等にメッセージをディスパッチしている可能性があります。必要な数以上のアクティブコンシューマが存在すると、一部のコンシューマはメッセージを受信しないことがあります。

ブローカクラスタ環境では、複数のコンシューマへの配信を設定し、ローカルコンシューマの優先度を設定できます。キュー送信先のプロパティを使用すると、プロデューサのホームブローカ、つまりプロデューサのメッセージ送信先となるブローカ (ローカルブローカ) にコンシューマが存在しない場合にのみ、メッセージがリモートコンシューマへ配信されるように指定できます。これにより、リモートコンシューマへ向けてルーティング (コンシューマのホームブローカを経由) するとスループットが低下する可能性がある状況で、パフォーマンスを向上させることができます。

トピックの送信先

トピックの送信先は、パブリッシュ / サブスクライバメッセージングで使用されます。メッセージは、送信先で配信対象を登録しているすべてのコンシューマに配信されるようになっています。

基本的なルーティングメカニズム

メッセージは、プロデューサから送信されてくると、トピックをサブスクライブするすべてのコンシューマに向けてルーティングされます。永続サブスクリプションをトピックに登録している場合、コンシューマは、トピックにメッセージが配信されたとき、アクティブ状態になってそのメッセージを受信する必要はありません。これは、コンシューマが再びアクティブになるまでブローカがメッセージを保存し、それから配信するからです。

セレクタのプロパティ値がメッセージに設定されていると、ブローカは、登録済みコンシューマによって指定されているいずれかのセレクタ値とそのプロパティ値を比較し、メッセージをコンシューマに向けてルーティングする前にセレクタ値が一致することを確認します。

永続サブスクリプションとクライアント ID

トピックへの永続サブスクリプションを持つことが可能なのは 1 人のユーザーだけです。そのユーザーがメッセージサーバーへのコネクションをオープンおよびクローズするときには、ユーザー ID が同じままになっている必要があります。クライアント識別子 (client identifier) は、それぞれの永続サブスクリプションが 1 人のユーザーにのみ対応していることを確認するために使用します。

クライアント ID により、メッセージサーバーへのクライアントのコネクションと、クライアントに代わってメッセージサーバーが保持する状態情報を関連付けます。定義により、クライアント ID は一意の値になります。

永続サブスクリプションを作成するには、クライアントがプログラムで JMS API メソッド呼び出しを使用して、またはクライアントが使用するコネクションファクトリオブジェクトでの管理操作によって、クライアント ID を設定する必要があります。

メッセージのコンシューム

ルーティングが済んだメッセージは、それぞれのコンシューマに配信されます。コンシューマがペイロードメッセージを受信すると、コンシューミングクライアントランタイムは、クライアントがメッセージを受信して処理したことを伝える通知をブローカに送信します。ブローカは、その送信先からメッセージを削除する前に、このクライアント通知が送信されてくるのを待ちます。クライアント通知は、個々のメッセージ、メッセージグループ、またはトランザクションに適用可能です。

クライアント通知

JMS 仕様に従い、クライアントでは、セッションを作成する場合に 3 つの基本通知モードのいずれかを指定できます。選択するモードは、必要なメッセージ配信の信頼性によって決まります。

Message Queue は、`NO_ACKNOWLDEGE` モードを追加してクライアント通知モードのセットを拡張します。以下の項では、基本モードと拡張モードについて説明します。

AUTO_ACKNOWLEDGE モード

`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。また、セッションスレッドがブロックし、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。この確認動作は、ブローカ通知と呼ばれます。

- メッセージリスナによる非同期メッセージコンシュームでは、`onmessage()` メソッドが返されたあとにメッセージが通知されます。

- 同期メッセージコンシュームでは、receive() メソッドが返される直前にメッセージが通知されます。この場合は、メッセージがコンシュームされる前にシステムで障害が発生すると、メッセージが部分的に処理されても消失する可能性があります。信頼性を高めるには、CLIENT_ACKNOWLEDGE モードまたは処理済みセッションを使用して、システムで障害が発生してもメッセージが消失しないことを保証します。

CLIENT_ACKNOWLEDGE モード

CLIENT_ACKNOWLEDGE モードでは、クライアントがほとんどの制御を行うことができます。このモードでは、1 つ以上のメッセージがコンシュームされたあとで、クライアントが明示的に通知します。通知は、クライアントがメッセージオブジェクトの acknowledge() メソッドを呼び出したときに発生します。ここでセッションは、最後に acknowledge() メソッドが呼び出されたあとにそのセッションによってコンシュームされるすべてのメッセージに対して通知します。これには、コンシュームされた順序に依存せずに、セッション内の多くのメッセージリスナーによって非同期にコンシュームされたメッセージが含まれます。

また、セッションスレッドがブロックして、コンシュームされたメッセージのバッチに対する戻りブローカ通知を待ちます。戻りブローカ通知は、ブローカがクライアント通知を処理したことを確認します。

一般に、クライアント通知とブローカ通知は 1 つずつ送信されずにバッチ処理されるので、通常 CLIENT_ACKNOWLEDGE モードでは、AUTO_ACKNOWLEDGE モードに比べてコネクション帯域幅が節約され、ブローカ通知のオーバーヘッドが低減されます。当然ながら、このモードではクライアントが各メッセージを通知し、バッチ処理は行われません。また、通知は 1 つずつ送信されます。

注 Message Queue には、CLIENT_ACKNOWLEDGE モードで使用できる固有のメソッドも用意されています。これにより、標準的な動作ではなく、そのメソッドを呼び出す個々のメッセージのみを通知することができます。この仕組みは、Message Queue Developer's Guide for Java Clients で説明されているプログラミング技法を使用して実現されています。

DUPS_OK_ACKNOWLEDGE モード

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。この値は、現在設定できません。AUTO_ACKNOWLEDGE モードや CLIENT_ACKNOWLEDGE モードとは異なり、DUPS_OK_ACKNOWLEDGE モードではブローカ通知が要求されないので、ブローカ通知を待ってセッションスレッドがブロックされることはありません。

つまり、メッセージが配信されて 1 回だけコンシュームされるという保証はありません。一般に、メッセージが頻繁に配信されることはなく、配信したメッセージに対するクライアント通知をブローカが受信しておらず、障害が発生した場合にのみ再配信されます。重複して配信しても構わない場合、クライアントでは DUPS_OK_ACKNOWLEDGE モードを使用してください。

クライアント通知はバッチ処理され、クライアントスレッドはブロックしないので、一般にメッセージのスループットは他のモードの場合よりも一段と高くなります。

NO_ACKNOWLEDGE モード

NO_ACKNOWLEDGE モードでは、ブローカがクライアントに代わってクライアント通知を実行するので、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という保証はありません。

メッセージのスループットが重要で、信頼性の高い配信が重要ではない場合は、このモードを使用してください。このモードを使用する場合としては、たとえば、短い間隔でメッセージが定期的に送信されるので、メッセージ負荷が高く、メッセージが消失してもそれほど問題にならない場合などがあります。

このモードは、JMS 仕様を拡張したものであり、他の JMS プロバイダと連携する必要のないクライアントでのみ使用してください。

トランザクション

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分類される JMS メッセージの配信にも適用されます。そのような場合、クライアント通知とブローカ通知は、トランザクションに関係するすべてのメッセージを含むトランザクションレベルで作動します。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。

ブローカはトランザクションを追跡し、トランザクションがコミットされるか、あるいは障害が発生した場合にロールバックされるようにします。このトランザクション管理は、大規模な分散トランザクションの一部であるローカルトランザクションもサポートします (31 ページの「分散トランザクション」を参照)。ブローカは、これらのトランザクションがコミットされるまで、トランザクションの状態を追跡します。ブローカは、起動するとコミットされていないすべてのトランザクションを調べ、手動で解決する必要のある PREPARED 状態のトランザクションを除いてデフォルトではトランザクションをすべてロールバックします。

Message Queue では、XA コネクションファクトリを介した分散トランザクションのサポートが実装されています。この XA コネクションファクトリでは、次に XA セッションを作成する XA コネクションを確立できます。さらに、分散トランザクションへのサポートには、サードパーティの JTS (Java Transaction Service)、または JTS を提供する J2EE 準拠の Application Server のいずれかが必要です。

メッセージの存続終了

ブローカは、正常に配信されたあとで送信先メモリーからメッセージを削除します。一方、正常に配信されていないのにメッセージが廃棄される場合もあります。次の項では、メッセージが廃棄されるときの条件について説明します。

通常のメッセージ削除

通常の条件では、メッセージが正常に配信され、クライアント通知によって確認されたときに、ブローカが送信先メモリーからメッセージを削除します。

ブローカは、コンシューマにメッセージを配信する場合、配信済みのマークをメッセージに付けますが、メッセージが受信されてコンシュームされたかどうかを実際には把握していません。そのためブローカは、その物理的な送信先と持続ストアからメッセージを削除する前に、クライアント通知が送信されてくるのを待ちます。

メッセージがトピックに送信された場合、ブローカは、メッセージを送信した先の各メッセージコンシューマからクライアント通知を受信するまで、メッセージを削除しません。トピックのサブスクリプションが永続的な場合、ブローカは、各メッセージをその送信先で保持し、各永続サブスクライバがアクティブなコンシューマになるとそのメッセージを配信します。ブローカは、クライアントの通知を受信するたびにそれを記録し、メッセージの有効期限が切れていない場合に限り、すべての通知を受信したあとにのみメッセージを削除します。ブローカは、クライアント通知モードに応じて、ブローカ通知をクライアントに返信することにより、クライアント通知を受信したことを確認する場合があります。

ブローカまたはコネクションで障害が発生すると、ブローカクライアント通知を受信せず、以前に配信しても通知されなかったメッセージすべてを、`Redelivered` フラグでマークを付けて再配信します。たとえば、メッセージを受信したことを通知する前にキューコンシューマがオフラインになり、別のコンシューマ (場合によっては同じコンシューマ) がその後キューに登録された場合、ブローカは `Redelivered` フラグを付けて通知されていないメッセージを新しいコンシューマに再配信します。このような条件下でのメッセージの再配信が関係するクライアントアプリケーションでは、このフラグが設定されていないかどうかメッセージを確認する必要があります。

注　受信されてもクライアントによって通知されなかったメッセージを、クライアントが明示的に再配信を要求することができる、JMS API (`recover Session`) があります。そのようなメッセージを再配信する場合、ブローカは `Redelivered` フラグによってメッセージにマークを付けます。

異常なメッセージ削除

メッセージは、配信することができない場合、配信を阻んだ条件に応じて、破棄されるかデッドメッセージキューに送られます。

以下の条件の下では、正常に配信およびコンシュームされる前に、メッセージはブローカによって廃棄されます。

- 管理ツールを使用して、1 つ以上のメッセージの送信先をパージする
- 持続サブスクリプションを削除または再定義し、配信の可能性がないのにトピック送信先にメッセージを残す。

ただし、上述の条件では、メッセージはデッドメッセージと見なされ、設定された動作に応じて、廃棄されるかデッドメッセージキューに送られます。

- メッセージの有効期限が切れ、メッセージヘッダーで設定されている `JMSExpiration` の値を超える。
- 送信先がメモリー限界のしきい値を超えたために、送信先の「削除」制限動作が呼び出される。
- メッセージをコンシュームできない (クライアントが例外をスローした) ために、メッセージの配信に失敗する。

上述のようなメッセージについては、保持するように選択し、デッドメッセージキューに入れることができます。メッセージをデッドメッセージキューに入れると、ブローカは Message Queue 固有のプロパティ値をそのメッセージに書き込み、キューに入れた時刻と理由を指定します。

その後は、デッドメッセージキューからメッセージを取り出し、診断することができます。詳細は、76 ページの「デッドメッセージキュー」を参照してください。

パフォーマンス問題

メッセージ配信の信頼性が高くなればなるほど、その信頼性を実現するために、より多くのオーバーヘッドや帯域幅が必要となります。したがって、信頼性とパフォーマンスの兼ね合いは、設計上の重要な考慮事項です。パフォーマンスは、非持続メッセージをプロデュースおよびコンシュームするように設定すれば最大限に高めることができます。一方信頼性は、持続メッセージをプロデュースおよびコンシュームし、処理済みのセッションを使用することで最大限に高めることができます。この両極の間に、各アプリケーションの必要に応じて、多くのオプションが存在しています。

たとえば、メッセージを処理可能な速度は、メッセージングアプリケーションの設計、メッセージサーバーの設定、クライアントランタイムの設定などの数多くの要因が組み合わされて決まる数値です。これらの要因は非常にはっきりしていますが、それぞれの要因の相互作用がパフォーマンスを最大化するタスクを複雑にしている場合があります。

この節では、信頼性とパフォーマンスの兼ね合いを決める際に関与する要因のいくつかについて検討します。

配信モード: 配信モードは、メッセージを多くても 1 回 (非持続) または 1 回だけ (持続) 配信するかどうかを指定します。持続メッセージを管理するには、コネクション全体を流れるブローカ通知メッセージや、ブローカ通知の受信を待ってブロックするクライアント通知モードを使用する必要があります。スループットを増大させるには、クライアントランタイムがブローカ通知を抑制するように設定できますが、この設定にすると、持続メッセージが必ず 1 回だけ配信されるという保証が取り消されます。

クライアント通知モード: 4 つの各クライアント通知モードは、異なる処理レベルと帯域幅のオーバーヘッドを必要とします。AUTO_ACKNOWLEDGE モードはほとんどのオーバーヘッドを消費し、メッセージ単位での信頼性を保証します。CLIENT_ACKNOWLEDGE モードは、通知をバッチ処理するので必要とする帯域幅が少なくて済み、DUPS_OK_ACKNOWLEDGE モードでは消費するオーバーヘッドは少ないものの、メッセージを重複して配信する可能性があります。NO_ACKNOWLEDGE モードでは最善のパフォーマンスが得られますが、メッセージが消失する可能性があります。

クライアントアプリケーションの設計: 1 つのセッションでキューに入るメッセージの数は、そのセッションを使用するメッセージコンシューマの数と、各コンシューマのメッセージ負荷によって決まります。クライアント側のメッセージのプロデュースまたはメッセージのコンシュームに時間がかかる場合は、通常は、アプリケーションを再設計し、より多くのセッションにメッセージプロデューサとメッセージコンシューマを分散し、またはより多くのコネクションにセッションを分散してパフォーマンスを改善できます。パフォーマンスに影響を与える設計上の問題については、『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。

メッセージフロー測定: コネクションの帯域幅をめぐるコントロールメッセージとペイロードメッセージの競合は、クライアントランタイムによって管理できます。クライアントランタイムを適切に設定すれば、ブローカ通知の配信速度を上げ、ブロックされたセッションスレッドを解放し、メッセージのコンシューム速度を高めることができます。詳細は、『Message Queue 管理ガイド』を参照してください。

メッセージフロー制限: メッセージのコンシュームは、クライアントランタイムのリソース限度に到達すると速度が低下する可能性があります。1 つ以上のコンシューマによってコンシュームされるのを待ってクライアントランタイムに保持されるメッセージの数を制限することにより、これらのリソース制限を回避できます。詳細は、『Message Queue 管理ガイド』を参照してください。

第 4 章

メッセージサーバー

37 ページの「メッセージサーバー」で紹介されている Message Queue メッセージサーバーは、メッセージのルーティングと配信を実行する、1 つのブローカまたは協調動作するブローカセット (ブローカクラスタ) で構成されています。

この章では、ブローカの内部構造について解説し、ブローカのさまざまなコンポーネントについて述べ、開発環境と運用環境でブローカを設定および管理するために必要なステップについて概要を示します。この章は以下の節で構成されています。

- 69 ページの「ブローカのアーキテクチャ」
- 71 ページの「ブローカのコンポーネント」
- 84 ページの「開発環境と運用環境」

ブローカの機能要素について理解すれば、必要なブローカの動作を設定し、動作を拡張し、パフォーマンスを最適化しやすくなります。そのためこの章は、アプリケーション開発者よりも管理者向けの内容になっています。

ブローカのアーキテクチャ

メッセージ配信を実行するため、ブローカは、クライアントとの通信チャネルの設定、認証と承認、適切なメッセージルーティング、信頼性の高い配信の保証、およびシステムパフォーマンスを監視するためのデータの提供を行う必要があります。

この複合的な機能の組み合わせを実行するために、ブローカはさまざまな内部コンポーネントを使用します。各コンポーネントには、配信プロセスにおける特定のロールが割り当てられています。図 4-1 にブローカのコンポーネントを、表 4-1 にはそれらのコンポーネントの概要を示します。メッセージルーターコンポーネントがメッセージルーティングと配信サービスを主に実行し、それ以外のコンポーネントは、メッセージルーターが依存する重要なサポートサービスを提供しています。

図 4-1　ブローカのコンポーネント

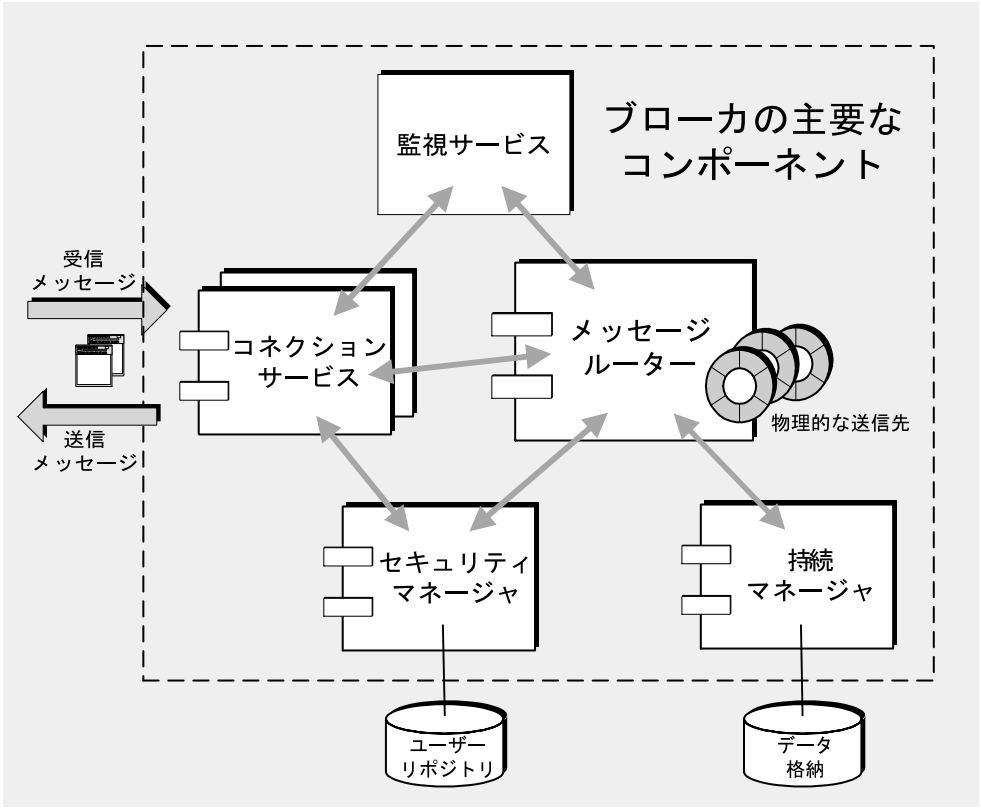


表 4-1　主要なブローカサービスコンポーネントと機能

コンポーネント	説明 / 機能
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います。
メッセージルーター	メッセージのルーティングおよび配信を管理します。JMS メッセージと、JMS メッセージの配信をサポートするために Message Queue メッセージングシステムが使用するコントロールメッセージを含みます。
持続マネージャ	システム障害が発生しても、JMS メッセージが配信されるように、持続ストレージへのデータの書き込みを管理します。
セキュリティマネージャ	ブローカへのコネクションを要求するユーザーに認証サービスを提供し、認証されたユーザーに承認サービス (アクセス制御) を提供します。

表 4-1　主要なブローカサービスコンポーネントと機能 (続き)

コンポーネント	説明 / 機能
監視サービス	さまざまな出力チャネルに書き込み可能なメトリックスと診断情報を生成します。この情報を使用してブローカを監視および管理できます。

ブローカを設定する場合は、負荷状態やアプリケーションの複雑さなどに応じて、実際にはこれらのサービスを設定してブローカのパフォーマンスを最適化していることになります。

ブローカのコンポーネント

以下の節では、図 4-1 に示すブローカの各コンポーネント、コンポーネントの機能、およびコンポーネントの動作について説明します。それぞれのプロパティと設定手順については、『Message Queue 管理ガイド』を参照してください。

コネクションサービス

Message Queue ブローカは、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との通信をサポートしています。各コネクションサービスは、サービスタイプとプロトコルタイプで指定されます。

サービスタイプ: サービスが、JMS メッセージ配信サービス (NORMAL) を提供するのか、管理ツールをサポートする Message Queue 管理サービス (ADMIN) を提供するのかについて指定します。

プロトコルタイプ: サービスをサポートする基礎となるトランスポートプロトコルレイヤーを指定します。

Message Queue ブローカで現在使用できるコネクションサービスを、表 4-2 に示します。

表 4-2　ブローカがサポートするコネクションサービス

サービス名	サービスタイプ	プロトコルタイプ
jms	NORMAL	TCP
ssljms	NORMAL	TLS (SSL ベースセキュリティ)
httpjms (Enterprise Edition)	NORMAL	HTTP

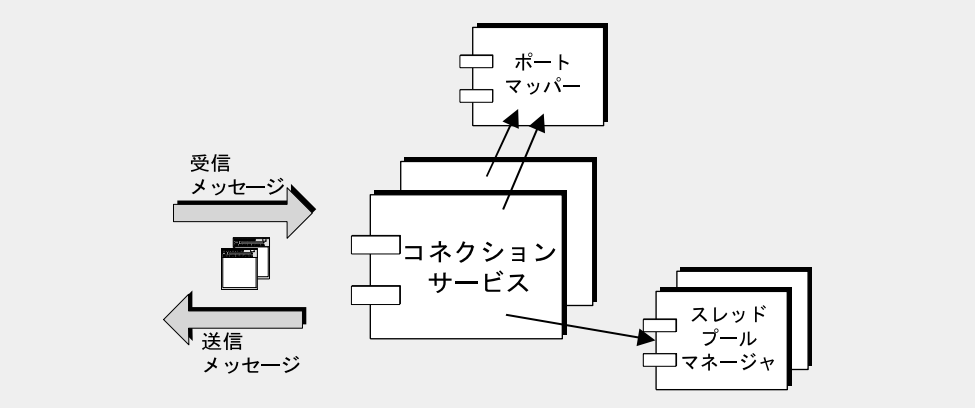
表 4-2　ブローカがサポートするコネクションサービス (続き)

サービス名	サービスタイプ	プロトコルタイプ
httpsjms (Enterprise Edition)	NORMAL	HTTPS (SSL ベースセキュリティ)
admin	ADMIN	TCP
ssladmin	ADMIN	TLS (SSL ベースセキュリティ)

ブローカを設定して、これらのコネクションサービスの一部、またはすべてを実行することができます。各コネクションサービスは、特定の認証および承認 (アクセス制御) 機能をサポートします (80 ページの「セキュリティマネージャ」を参照)。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。

各コネクションサービスは、ブローカのホスト名とポート番号で指定した特定のポートで使用できます。ポートを動的に割り当てることもできれば、コネクションサービスが使用可能なポートを指定することもできます。この仕組みの概略を、図 4-2 に示します。

図 4-2　コネクションサービスのサポート



ポートマッパー

コネクションサービスには、共通ポートマッパーによりポートが割り当てられます。ポートマッパー自体は、標準のポート番号 7676 に常駐します。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、必要とする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 の各コネクション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 (73 ページの「HTTP/HTTPS サポート」を参照)、一般的にはお勧めしません。

スレッドプールマネージャ

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、スレッドプールマネージャのコンポーネントが管理するスレッドプールに保存されます。スレッドプールマネージャを設定して、スレッドプールに保持されるスレッドの最小数と最大数を指定することができます。コネクションでスレッドが必要になると、スレッドプールにスレッドが追加されます。スレッドの最小数より少なくなると、システムは最小数のしきい値になるまでスレッドをシャットダウンして、スレッドを解放します。これによってメモリーのリソースが節約されます。スレッドプール内のスレッドは、1 つのコネクションだけに割り当てるか、必要に応じて、複数のコネクションに割り当てることができます。

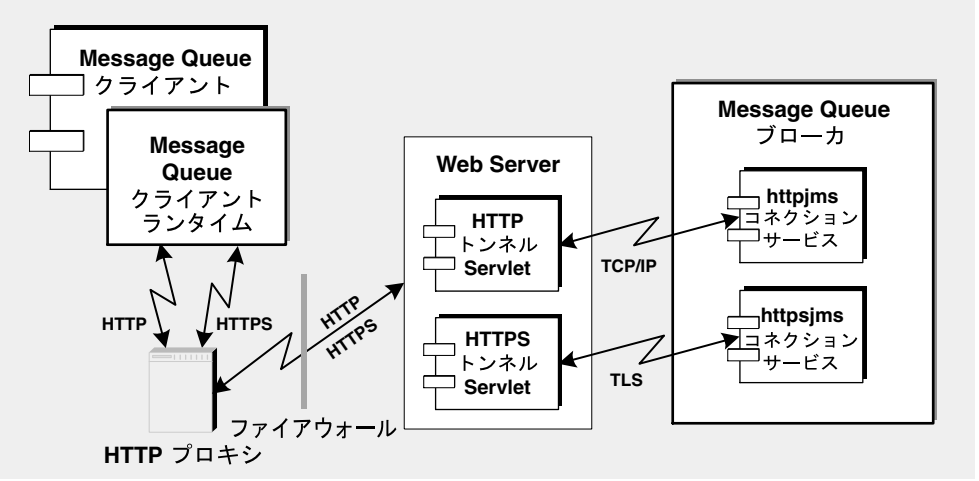
HTTP/HTTPS サポート

HTTP/HTTPS サポートにより、Message Queue クライアントは、直接 TCP コネクションを使用する代わりに HTTP プロトコルを使用してブローカと通信できます。クライアントをファイアウォールによってブローカと分離する必要がある場合、ファイアウォールを通しての通信が可能なので HTTP/HTTPS サービスを使用できます。

注 HTTP/HTTPS サポートは、Java クライアントに対応しており、C クライアントには対応していません。

図 4-3 に、HTTP/HTTPS サポートの提供に関連する主なコンポーネントを示します。

図 4-3　HTTP/HTTPS サポートのアーキテクチャ



- Message Queue クライアント側では、HTTP または HTTPS の転送ドライバが MJS メッセージを HTTP 要求にカプセル化し、これらの要求が正しい順序で Web サーバーに確実に送信されるようにします。
- クライアントは、HTTP プロキシサーバーをオプションで使用して、必要に応じて Web サーバーとやり取りできます。
- HTTP または HTTPS トンネルサーブレット (どちらも Message Queue にバンドルされている) は、Web サーバーに読み込まれ、JMS メッセージがブローカに転送される前に、その JMS メッセージをクライアント HTTP 要求から取り出します。HTTP/HTTPS トンネルサーブレットも、ブローカ応答をクライアントに返信します。1 つの HTTP/HTTPS トンネルサーブレットを使用して、複数のブローカにアクセスできます。
- ブローカ側では、ブローカの起動時に、トンネルサーブレットへのブローカコネクションが確立されます。メッセージが HTTP または HTTPS トンネルサーブレットから送信されるとき、`httpjms` または `httpsjms` コネクションサービスは、それぞれメッセージのラップを解除し、ブローカのメッセージルーターコンポーネントにサブミットします。

図 4-3 に示す HTTP と HTTPS のアーキテクチャは、非常によく似ています。主な相違点は、HTTPS (`httpsjms` コネクションサービス) の場合、トンネルサーブレットにクライアントとブローカの両方へのセキュリティ保護されたコネクションがあることです。

Message Queue の HTTPS トンネルサーブレットは、自己署名証明書を、コネクションを要求している任意のブローカに渡します。ブローカは証明書を使用して、HTTPS トンネルサーブレットへの暗号化されたコネクションを設定します。このコネクションが確立されれば、Message Queue クライアントとトンネルの間の安全なコネクションをネゴシエーションできます。

`httpjms` サービスと `httpsjms` サービスは、『Message Queue 管理ガイド』で説明されるプロパティを使用して設定されます。

メッセージルーター

サポートされているコネクションサービスを使用して、クライアントとブローカ間でコネクションが確立されると、メッセージのルーティングおよび配信が処理できます。

Message Queue のメッセージングは、メッセージの 2 段階配信を前提としています。まず、プロデューシングクライアントからブローカの物理的送信先に、次にブローカの物理的送信先から 1 つ以上のコンシューミングクライアントに送られます。メッセージルーターは、適切な送信先に到着したメッセージを収容し、その後適切なコンシューマにメッセージをルーティングして配信するプロセスを管理します。

この節では、さまざまな種類の送信先およびそれらの送信先のメモリーリソース管理について、個別に、および総合的に説明します。メッセージのルーティングと配信のメカニズムについては、第 3 章「信頼性の高いメッセージ配信」を参照してください。

物理的な送信先

物理的な送信先は、受信メッセージがコンシューミングクライアントにルーティングされる前に、着信メッセージを保管するブローカの物理メモリーの場所を表しています。

送信先は、作成方法と作成目的に応じて、管理者作成、自動作成、一時、デッドメッセージキューなどの多くのカテゴリに分類されます。

管理者作成の送信先

管理者作成の送信先は、Message Queue 管理ツール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、クライアントが検索す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。

Message Queue メッセージサーバーは、メッセージングシステムの中心的なハブなので、そのパフォーマンスと信頼性はエンタープライズアプリケーションの成功にとって重要です。送信先は、送信先が処理するメッセージの数とサイズ、および登録するメッセージコンシューマの数と永続性によっては、リソースを著しくコンシュームす

る可能性があるので、メッセージサーバー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。したがって、アプリケーションのための送信先の作成、送信先の監視、必要に応じたリソース要件の再構成が、管理者の基本的な業務になります。

自動作成の送信先

自動作成の送信先は、管理者の介入なしに、必要に応じてブローカが自動的に作成します。特に、自動作成の送信先は、メッセージコンシューマまたはメッセージプロデューサが、存在しない送信先へのアクセスを試みる場合にはいつでも作成されます。自動作成の送信先は、一般に開発とテストサイクルの期間に、送信先を動的に作成する必要がある場合に使用されます。自動作成機能を有効または無効にするように、ブローカを設定することができます。

送信先が自動的に作成されると、同じ送信先名を使用する異なるクライアントアプリケーションどうしの間でクラッシュが発生したり、送信先をサポートするために必要なリソースによってシステムのパフォーマンスが低下したりする可能性があります。このため、自動作成された送信先は、それ以降使用されない状態、つまり、それ以降メッセージ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。ブローカが再起動すると、持続メッセージが含まれている場合にだけ、自動作成された送信先が再び作成されます。

一時送信先

一時送信先は、ほかのクライアントに送信した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 JMS API を使用して明示的に作成および破棄します。これらの送信先は、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。管理者が一時送信先を破棄することはできません。また、一時送信先が使用されている間、つまりアクティブなメッセージコンシューマの場合は、クライアントアプリケーションによって破棄されることもありません。管理者が作成した送信先や自動作成された送信先 (持続メッセージを保持する) と違って、一時送信先は、継続的には格納されず、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、Message Queue 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

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。デッドメッセージは、通常の処理または明示的な管理操作以外の理由でシステムから削除されるメッセージです。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。

メッセージは、次の 2 つの方法でデッドメッセージキューに入れることができます。

- メッセージを廃棄するのではなく、デッドメッセージキューに入れるように送信先を設定することができます。

- クライアント開発者は、メッセージをデッドメッセージキューに入れてデッドメッセージにするかどうかを判断するプロパティ値を、メッセージを作成するときに設定できます。

メッセージをデッドメッセージキューに入れると、追加のプロパティ情報がそのメッセージに書き込まれ、デッドメッセージになった原因に関する情報が残されます。

メモリーリソース管理

メッセージサーバーには、メモリーや CPU サイクルなどのリソースにおいて制限があります。そのためメッセージサーバーは、ブローカがサポートするメッセージングアプリケーションの使用方法に応じて、応答しなくなったり安定性を失ったりするほどの過負荷状態に陥る可能性があります。これは特に、送信先へのメッセージのプロデュース速度が、コンシューム速度よりはるかに速い場合に問題になります。

メッセージルーターには、メモリーリソースを管理し、ブローカのメモリー不足を防ぐためのメカニズムが組み込まれています。この場合ルーターは、送信先のメッセージ制限、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システムの動作を維持します。

送信先のメッセージ制限

メッセージは長期間送信先に残ることがあるので、メモリーリソースが問題になる可能性があります。送信先に割り当てるメモリーが多すぎると、システムメモリーのサイズが制限され、少なすぎると、メッセージが拒否されます。

各送信先の負荷要求に基づいて柔軟な対応ができるようにするため、各送信先のメモリーリソースとメッセージフローを管理するプロパティを設定できます。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、これらの制限値に達したときにメッセージルーターが示す応答を、4 つの応答方法から指定することもできます。4 種類の制限の動作は次のとおりです。

- メッセージプロデューサの動作を遅くする
- もっとも古いメッセージを廃棄する
- 有効期間に従い、優先度がもっとも低いメッセージを廃棄する
- 最新のメッセージを拒否する

システム全体のメッセージ制限

システム全体のメッセージ制限は、第 2 の保護手段です。システム全体の制限を指定すると、ブローカのすべての送信先に対して一括して適用され、メッセージの総数とすべてのメッセージによって使用される総メモリー量が制限されます。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、メッセージルーター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

システムメモリーのしきい値は、第 3 の保護手段です。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 yellow、orange、red へと進むにつれ、ブローカは次のタイプの本格的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します。
- 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します。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ます。

これらのアクションはどちらもパフォーマンスを低下させます。

システムメモリーのしきい値に達した場合は、送信先ごとのメッセージ制限とシステム全体のメッセージ制限が適切に設定されていません。場合によっては、潜在するメモリーの過負荷をしきい値がタイミングよく指摘できないことがあります。したがって、この機能に依存せず、代わりに送信先を個別および全体的に設定して、メモリーリソースを最適化してください。

注意深い監視と調整により、送信先ベースの制限と動作を使用して、システムが過負荷にならないようにメッセージの受信と送信のバランスを取ることができます。これらのメカニズムは、オーバーヘッドを増加させ、メッセージのスループットを制限することがありますが、それでも動作の完全性を維持します。

持続マネージャ

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。この場合、ブローカは、すべての持続メッセージを重要な転送情報および配信情報とともにデータストアに保存する必要があります。復元するには、ブローカが次のタスクも実行する必要があります。

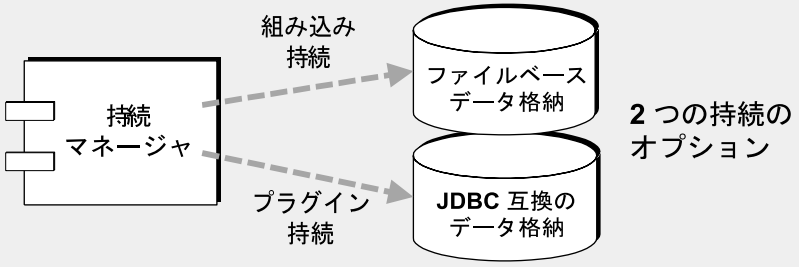
- 送信先の再作成
- 各トピックの永続サブスクリプションのリストの復元
- 各メッセージの通知リストの復元
- コミットされたすべてのトランザクションの状態の復元

持続マネージャは、このすべての状態情報の格納および検索を管理します。

ブローカは、再起動すると、持続マネージャによって管理されるデータを使用して送信先と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 は、組み込み持続モジュールとプラグイン持続モジュールの両方をサポートしています (図 4-4 を参照)。組み込み持続は、ファイルベースのデータストアです。プラグイン持続は、JDBC (Java Database Connectivity) インタフェースを使用し、JDBC 互換のデータストアを必要とします。通常、組み込み持続の方が、プラグイン持続より処理速度が速くなりますが、JDBC 互換のデータベースシステムによる冗長性および管理制御を好むユーザーもいます。

図 4-4　持続マネージャのサポート



組み込み持続性 : デフォルトの Message Queue 持続ストレージソリューションは、ファイルベースのデータストアで、個々のファイルを使用して持続データを保存します。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、ファイルベースのデータストアを圧縮できます。信頼性を最大にするため、持続操作によりメモリー内の状態と物理的なストレージとを同期するように指定できます。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が、パフォーマンスが低下します。データストアには機密事項を扱うメッセージや財産的価値のあるメッセージが格納されることがあるため、権限のないアクセスからデータストアのファイルを保護してください。

プラグイン持続 : JDBC ドライバを介してアクセスが可能な任意のデータストアにアクセスするように、ブローカを設定することができます。この作業では、さまざまな JDBC 関連ブローカ設定プロパティを設定し、Message Queue Database Manager ユーティリティを使用し、適切なスキーマでデータストアを作成します。手順および関連する設定プロパティについての詳細は、『Message Queue 管理ガイド』を参照してください。

セキュリティマネージャ

Message Queue では、認証および承認 (アクセス制御) 機能が用意されており、暗号化機能もサポートされています。

- 認証により、検証されたユーザーだけがメッセージサーバー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、コネクションサービスや送信先などのリソースにアクセスし、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権限を持つユーザーを指定します。
- 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ユーザーリポジトリによって異なります (81 ページの図 4-5 を参照)。ユーザーリポジトリとしては、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、グループメンバーシップなど) を含むファイル、ディレクトリ、またはデータベースがあります。ユーザー名とパスワードはブローカへのコネクションが要求されたときに、ユーザーを認証するために使用されます。ユーザー名とグループのメンバーシップは、送信先のプロデューシングメッセージ、またはコンシューミングメッセージなどの操作を承認するために、アクセス制御ファイルと一緒に使用されます。

Message Queue 提供のユーザーリポジトリに値を入力するか、前から存在する LDAP ユーザーリポジトリをブローカにプラグインすることができます。単層型ファイルのユーザーリポジトリは容易に使用できますが、セキュリティ攻撃に対して脆弱なため、評価および開発を目的とする場合にだけ使用してください。LDAP ユーザーリポジトリはセキュリティ保護されているので、プロデュース目的に最適です。

以下の項では、認証、承認、および暗号化について説明します。詳細は、『Message Queue 管理ガイド』を参照してください。

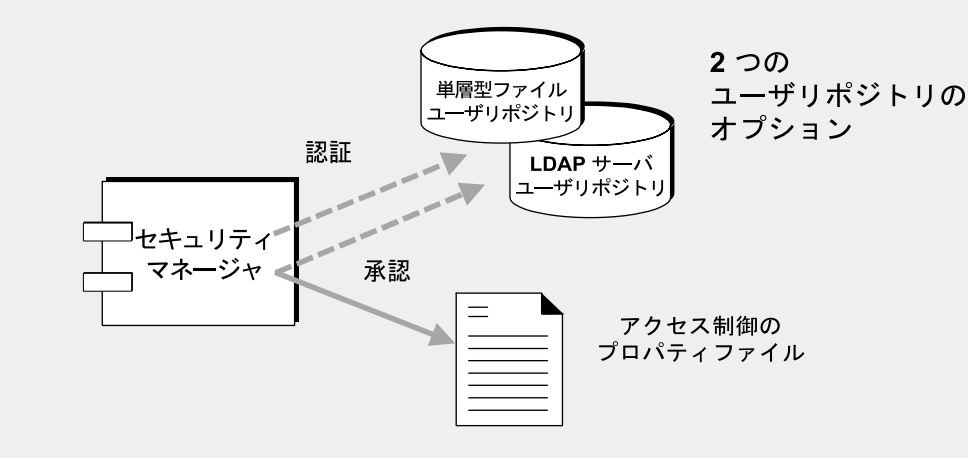
認証

Message Queue のセキュリティは、パスワードベースの認証がサポートしています。クライアントがブローカへの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。セキュリティマネージャは、クライアントから提示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、Base-64 か、メッセージダイジェスト (MD) のどちらかを使用して暗号化されます。セキュリティ保護された送信については、82 ページの「暗号化」を参照してください。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。

承認

クライアントアプリケーションのユーザーが認証されると、ユーザーはさまざまな Message Queue 関連のアクティビティを実行することを承認されます。セキュリティマネージャは、ユーザーベースとグループベースの両方のアクセス制御をサポートしています。ユーザー名、またはユーザーリポジトリでユーザーに割り当てられているグループに応じて、ユーザーは特定の Message Queue 操作を実行するためのアクセス権を付与されます。これらのアクセス制御は、アクセス制御プロパティファイルで指定します (図 4-5 を参照)。

図 4-5　セキュリティマネージャのサポート



ユーザーが何らかの操作を実行しようとすると、セキュリティマネージャは、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ファイルでは、次の操作に対する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

グループを定義し、ユーザーリポジトリ内のそれらのグループとユーザーを関連付けることができます。ただし、グループは単層型ファイルのユーザーリポジトリで完全にはサポートされません。次に、アクセス制御プロパティファイルを編集することにより、ユーザーまたはグループがアクセスしてプロデュース、コンシューム、またはブラウズできる送信先を指定できます。個々の送信先、またはすべての送信先を、特定のユーザーまたはグループだけがアクセスできるようにすることも可能です。

さらに、ブローカを設定して、送信先の自動作成 (76 ページの「自動作成の送信先」を参照) を許可する場合、ブローカが送信先を自動作成するユーザーを、アクセス制御プロパティファイルを編集して管理することができます。

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を提供します。

監視サービス

ブローカには、ブローカの動作を監視および診断するためのさまざまなコンポーネントが用意されています。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。

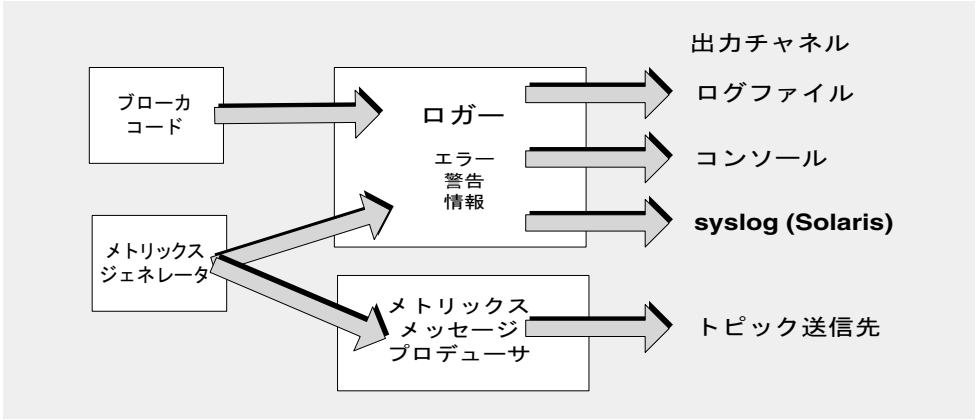
この仕組みの概略を、83 ページの図 4-6 に示します。

メトリックスジェネレータ

図 4-6 に示す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。

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。

図 4-6　監視サービスのサポート



ロガー

図 4-6 に示す 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 `syslog` デーモンプロセスなどの多くの出力チャネルにそれらの情報を書き込みます。

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 4-6 に示すメトリックスメッセージプロデューサのコンポーネントは、定期的にメトリックスジェネレータのコンポーネント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、送信先内の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情報を処理できます。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 クライアント

を開発してメトリックスメッセージをコンシュームする方法について説明した、『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。

開発環境と運用環境

メッセージングアプリケーションを開発およびテストし、運用環境にそれらのアプリケーションを配備して管理するには、Message Queue サービスによって提供されるメッセージングインフラストラクチャが必要です。

この節では、開発環境と運用環境で Message Queue を使用する場合の異なる手法を紹介します。次のトピックが含まれます。

- 84 ページの「開発環境とタスク」
- 85 ページの「運用環境とタスク」

運用環境とタスクを設定および管理する責任が委ねられているのは管理者ですが、運用環境の各部を設定および構成し、クライアントが期待通りに動作するかどうかを判断するために必要な情報を管理者に提供するために、運用環境について理解することは開発者にとっても重要です。

開発環境とタスク

開発環境では、作業は Message Queue のクライアントアプリケーションのプログラミングに重点が置かれます。Message Queue サービスは、主にテストのために必要となります。

出荷状態の設定

Message Queue 製品は、出荷状態で使用できるように設計されています。デフォルト値でブローカを起動すれば、デフォルトのデータストア、ユーザーリポジトリ、およびアクセス制御プロパティファイルを使用することができます。

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、管理者による介入なしでインストール後すぐに Message Queue ブローカを使用できます。言い換えれば、ユーザーやパスワードの初期設定を行わなくても、ブローカを使用することができます。デフォルトのユーザー名 (`guest`) とパスワード (`guest`) を使用して、クライアントユーザーを認証できます。

サンプルアプリケーションも数多く用意されているので、これを利用して新規アプリケーションを開発できます。

開発手法

開発環境では、柔軟性が重視され、通常は次の手法を取り入れます。

- 開発者がテストで使用するブローカを起動する程度に、管理の手間を最小限に抑えます。
- データストア (ファイルベースの組み込み持続)、ユーザーリポジトリ (ファイルベースのユーザーリポジトリ)、アクセス制御プロパティファイルのデフォルトの環境を使用します。通常の開発テストでは、これらのデフォルトの実装で十分です。
- 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するか、クライアントコードの管理対象オブジェクトをインスタンス化してオブジェクトストアをまったく使用しないようにします。
- 複数のブローカのテストを実行する場合は、マスターブローカを使用しません (91 ページの「クラスタの同期化」を参照)。
- 一般に、明示的に作成した送信先ではなく、自動作成された送信先を使用します。

運用環境とタスク

運用環境では、アプリケーションを配備、管理、および調整して、最大限のパフォーマンスを引き出す必要があります。そのような状況では、メッセージングインフラストラクチャの管理と調整は、必要不可欠ではないとしても、重要な要件となります。

また、導入段階では、スケーラビリティと可用性の両方における企業の要件に対応する必要があります。これには、メッセージサービスのカスタマイズされた設定とセットアップ、パフォーマンスの調整とシステムの拡張による増大する負荷への対処、およびメッセージサービスリソースとアプリケーション固有リソースの両方に対する毎日の監視と管理の実行が必要です。したがって管理タスクは、メッセージングシステムの複雑さとメッセージングシステムがサポートするアプリケーションの複雑さによって異なります。以下の節では、管理タスクをセットアップ操作とメンテナンス操作に分類しています。これらのタスクを実行するために要求される手順については、『Message Queue 管理ガイド』を参照してください。

セットアップ操作

運用環境では、セキュリティ保護アクセスのセットアップ、コネクションファクトリと送信先オブジェクトの設定、クラスタのセットアップ、持続ストアの設定、およびメモリー管理を行う必要があります。

➤ 運用環境を設定するには

通常、次のセットアップ操作のうち少なくとも一部を実行する必要があります。

- **セキュリティ保護された管理者アクセス** (保護された管理ツールの使用):
  - admin コネクションサービスがアクティブであることを確認する。
  - 承認 : 特定の個人または admin グループに対して admin コネクションサービスへのアクセスを許可する。
  - グループに対して承認を実行する場合は、管理者が admin グループに属していることを確認する。
    - ファイルベースのユーザーリポジトリ : デフォルトの admin グループを持つ。管理者が admin グループに属していることを確認するか、デフォルトの admin ユーザーを使用する場合は、admin パスワードを変更する。
    - LDAP ユーザーリポジトリ : 管理者が admin グループに属していることを確認する。
- **セキュリティ保護されたクライアントアクセス** :
  - 認証 : ファイルベースのユーザーリポジトリにエントリを作成するか、あるいはブローカを設定して、既存の LDAP ユーザーリポジトリを使用する。

    少なくとも、管理機能をパスワードで保護する必要がある。
  - 承認 : アクセス制御プロパティファイルのアクセス設定を変更する。
  - 暗号化 : SSL ベースのコネクションサービスを設定する。
- **物理的な送信先の作成**
- **管理対象オブジェクトの作成** :
  - LDAP オブジェクトストアを設定する。
  - 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を作成する。
- **必要な場合はブローカクラスタの作成** :
  - 中央設定ファイルを作成する。
  - マスターブローカを指定する。
- **ブローカを設定してプラグイン持続を使用する。**
- **メモリー管理の設定**

  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が使用可能なブローカのメモリーリソース内に収まるように送信先属性を設定する。

メンテナンス操作

運用環境では、Message Queue メッセージサーバーのリソースを監視および制御する必要があります。アプリケーションのパフォーマンス、信頼性、およびセキュリティが重視されるため、次に示すさまざまな運用タスクを Message Queue の管理ツールを使用して実行する必要があります。

➤ 運用環境を設定するには

- **アプリケーション動作の管理**
  - ブローカの自動作成機能を無効にする。
  - アプリケーションに代わって物理的な送信先を作成する。
  - 送信先へのユーザーアクセスを設定する。
  - 送信先を監視および管理する。
  - 永続サブスクリプションを監視および管理する。
  - トランザクションを監視および管理する。
- **ブローカの監視と調整**
  - ブローカのメトリックスを使用して、ブローカの調整および再設定を行う。
  - ブローカのメモリーリソースを管理する。
  - クラスタにブローカを追加して、ロードバランスを実行する。
  - 障害の発生したブローカを復元する。
- **管理対象オブジェクトの管理**:
  - 必要に応じて、追加の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を作成する。
  - コネクションファクトリの属性値を調整して、パフォーマンスとスループットを改善する。

第 5 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、メッセージサーバーは、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そ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。

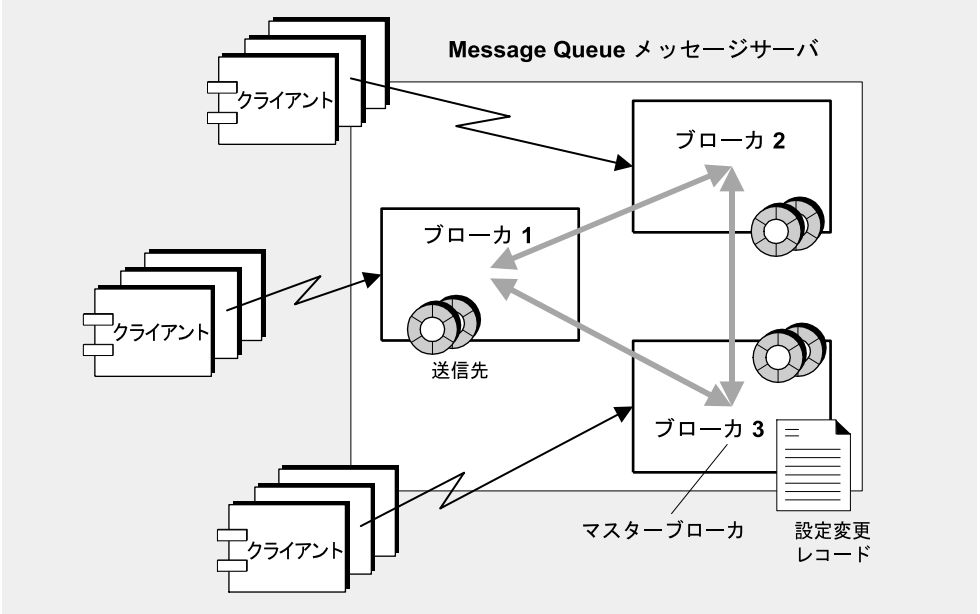
ブローカクラスタの設定と管理を担当する管理者、またはクラスタを使用してメッセージングアプリケーションをテストする必要のある開発者は、この章の内容を必ずお読みください。

クラスタのアーキテクチャ

図 5-1 に、ブローカクラスタ構成の Message Queue のアーキテクチャを示します。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、ホームブローカがサーバー内の唯一のブローカであるかのようにしてメッセージを送受信し、直接通信します。背後では、そのホームブローカがクラスタ内の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するときの負荷を分散しています。

クラスタ内では、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。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 (送信先と永続サブスクリプション) への変更点はそこに記録されます。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。詳細については、次の「クラスタの同期化」を参照してください。

図 5-1　クラスタのアーキテクチャ



次の節では、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、それぞれの送信先がクラスタ内のすべてのブローカで複製されます。各ブローカは、ほかのすべてのブローカの送信先に対して登録されたメッセージコンシューマを認識しています。これにより各ブローカは、ブローカに直接接続しているメッセージプロデューサから、リモートのメッセージコンシューマにメッセージをルーティングし、リモートのプロデューサから、ブローカに直接接続しているコンシューマにメッセージを配信することができます。

メッセージプロデューサが所有するホームブローカは、そのプロデューサによって作成されたメッセージの場合に、すべてのストレージとルーティングを担当し、すべてのクライアント通知を処理します。クラスタ内のメッセージトラフィックを最小限に抑えるため、メッセージは、ターゲットブローカに接続されたコンシューマにメッセージが配信されることになっている場合にのみ、1 つのブローカから別のブローカに送信されます。複数のコンシューマへのキュー配信などの場合には、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して、トラフィックをさらに減らすことができます。クライアントとメッセージサーバーの間で、暗号化による安全なメッセージ配信が必要になる状況では、クラスタを設定して、ブローカ間でセキュリティ保護されたメッセージ配信を実行することも可能です。

注 いくつかの例外がありますが、クラスタ内の送信先プロパティは、個々の送信先インスタンスよりも、全体としてのクラスタに集合的に適用されます。特定の送信先プロパティについての詳細は、『Message Queue 管理ガイド』を参照してください。

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のセットによって指定されます。このプロパティセットは、クラスタ内のすべてのブローカで統一する必要があります。それぞれのブローカの設定プロパティ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、設定の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。伝播される設定変更には、次の情報が含まれます。

- 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。
- 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。

こうした設定変更情報は、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。

開発環境

ブローカクラスタの使用方法は、開発環境と運用環境のどちらの環境に配備されているかによって異なります。

開発環境

クラスタをテストに使用し、スケーラビリティやブローカの復元が重視されない開発環境では、マスターブローカの必要性はほとんどありません。多くのテスト環境では、送信先が自動作成 (76 ページの「自動作成の送信先」を参照) され、これらの送信先の永続サブスクリプションは、テスト対象のアプリケーションによって作成および破棄されます。マスターブローカが存在しない場合、Message Queue はほかのブローカを起動するために、マスターブローカを実行する要件を緩和し、送信先や永続サブスクリプションが変更されて実行中のすべてのブローカに伝播されるのを許可します。ただし、一度オフラインになってから復元された場合、ブローカはオフライン中に行われた変更情報によって同期されません。マスターブローカを使用するように環境を再設定すると、Message Queue は通常の要件を再適用します。

運用環境

スケーラビリティとブローカの復元が重要視される運用環境では、マスターブローカを使用して設定変更記録を保守することが欠かせません。これにより、ブローカが一度オフラインになってから復元された場合でも、オフライン中に行われた変更情報によってブローカを同期します。

実際、設定変更記録の偶発的な破損やマスターブローカで発生する障害から保護するため、設定変更記録を定期的にバックアップすることをお勧めします。Message Queue には、設定変更記録をバックアップして復元する場合のコマンド行オプションが用意されています。必要に応じて、マスターブローカとしてサービスを提供するブローカを変更することもできます。詳細は、『Message Queue 管理ガイド』を参照してください。

第 6 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。

この章では、J2EE コンポーネントのプログラミングと配備の両方について説明しているため、アプリケーション開発者と管理者いずれにも有用です。

JMS/J2EE プログラミング : メッセージ駆動型 Beans

27 ページの「JMS プログラミングモデル」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 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 JMS クライアントは、メッセージ駆動型 Beans と呼ばれ、EJB 2.0 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。

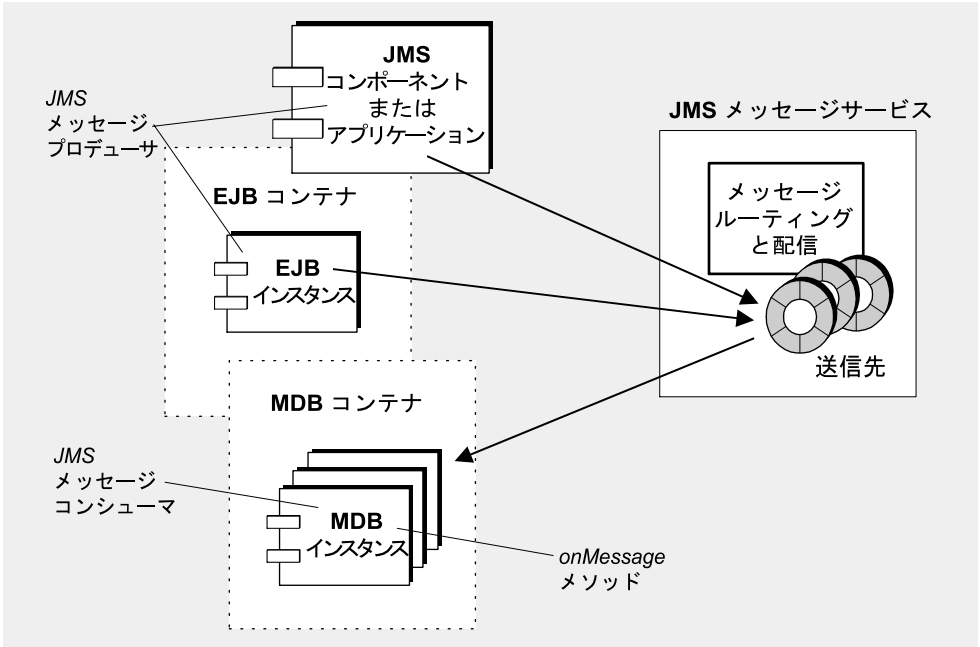
ほかの EJB コンポーネント (セッション Beans とエンティティ Beans) は同時に呼び出す必要があるため、メッセージ駆動型 Beans が必要となります。これらの EJB コンポーネントは、標準的な EJB インタフェースを介してしかアクセスできないため、非同期メッセージ受信のメカニズムは備わっていません。

ただし、非同期メッセージングは多くのエンタープライズアプリケーションの要件となっています。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、相互に通信および応答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、特殊な EJB コンテナ (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ソフトウェア環境) のサポート対象となる特殊な EJB コンポーネントです。

メッセージ駆動型 Beans: MDB は、JMS `MessageListener` インタフェースを実装する JMS メッセージコンシューマです。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 6-1 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。

図 6-1　MDB を使用したメッセージング



MDB コンテナ : MDB は、専用の EJB コンテナによってサポートされます。この EJB 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する役割を担います。これには、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、セッションのプールや関連する MDB インスタンスでメッセージが受信されるときのメッセージ分散の管理が含まれます。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。

コネクションファクトリと送信先のメッセージコンシュームを設定するときに、コンテナに使用される管理対象オブジェクトの JNDI 検索名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 のインスタンスだけをサポートします。

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 (http://java.sun.com/j2ee/download.html#platformspec にある「J2EE Platform Specification」を参照) 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、持続マネージャ、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 アプリケーションサーバーでは、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 アプリケーションサーバー 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。

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 1.4 は、J2EE Connector Architecture (J2EECA) 1.5 仕様によって定義されています。このアーキテクチャを使用することにより、J2EE 1.4 準拠の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、さまざまな企業情報システム (EIS)、および JMS プロバイダなどのさまざまな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。

J2EECA 1.5 によって実装が容易になった標準的なやり取りには、コネクションプーリング、スレッドプーリング、トランザクションとセキュリティのコンテキスト伝達、各種のメッセージ駆動型 Beans コンテナのサポートがあります。仕様には、コネクションファクトリとそのほかの管理対象オブジェクトを作成する標準的な方法も含まれています。

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、メッセージサーバー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を説明します。

この資料は、アプリケーション開発者向けです。

各 JMS オプション項目の処理方法を、表 A-1 に示します。

表 A-1 オプションの JMS 機能

JMS 仕様の節	説明と Message Queue における取り扱い
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。

表 A-1　　オプションの JMS 機能 (続き)

JMS 仕様の節	説明と Message Queue における取り扱い
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 (41 ページの「メッセージのヘッダー値のオーバーライド」を参照)。
3.5.9 JMS の定義済みプロパティ	「JMS では、JMS の定義済みプロパティ用として、JMSX プロパティ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	「JMS では、プロバイダ固有のプロパティ用として、'JMS_<*vendor_name*>' というプロパティ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。Message Queue 3 では、プロバイダ固有のプロパティを使用していません。
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 (65 ページの「トランザクション」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 における取り扱い
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、61 ページの「複数のコンシューマへのキューの配信」を参照してください。

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。

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　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　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　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 メッセージサーバー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 メッセージサーバー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。クライアントランタイムは、`ConnectionFactory` プロパティを設定することによって設定されます。

クラスタ　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　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、クライアントとメッセージサーバーの間の通信チャネル。

コネクションファクトリ (connection factory)　メッセージサーバー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　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　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。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 メッセージサーバーの物理的な送信先。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　クライアントとメッセージサーバー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　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　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　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　検証されたユーザーだけがメッセージサーバー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　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他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　メッセージのルーティング、配信、持続性、セキュリティ、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。

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 (`TopicPublisher`) になります。

メッセージ　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。

メッセージサーバー (message server)　Message Queue サービスに集中配信サービスを提供する 1 つ以上のブローカ。クライアントへのコネクション、メッセージ処理とルーティング、持続性、セキュリティ、監視などのサービスを提供します。メッセージサーバーは、プロデューシングクライアントのメッセージ送信先であり、コンシューミングクライアントへのメッセージ配信元である、物理的な送信先を管理します。

メッセージサービス　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。メッセージサーバーがその機能を実行するために必要とする、メッセージサーバー、クライアントランタイム、および数個のデータストアが組み込まれています。

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

あ

い

え

お

か

き

く

こ

さ

し

す

せ

そ

つ

て

と

に

は

ひ

ふ

へ

ほ

ま

め

ゆ

ら

り

る

ろ