Sun Java System Message Queue 4.1 リリースノート

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

Part No: 820–3191
2007 年 9 月

070928@18741

目次

◆ ◆ ◆ 第 1 章

Sun Java System Message Queue 4.1 リリースノート

Version 4.1

Part No. 820-3191

このリリースノートには、Sun Java™ System Message Queue 4.1 のリリース時点で得られる重要な情報が含まれています。ここでは、新機能、拡張機能、既知の問題、制限事項などについて説明します。Message Queue を使用する前に、このドキュメントをよくお読みください。このリリースノートには、4.0 リリースの Message Queue に関する情報も含まれています。そのリリースで導入された機能については、17 ページの「Message Queue 4.0 について」を参照してください。

このリリースノートの最新版は、Sun Java System Message Queue ドキュメント Web サイトから入手できます。ソフトウェアをインストールおよび設定する前だけでなく、それ以降も定期的にこの Web サイトをチェックして、最新のリリースノートと製品マニュアルを確認してください。

このリリースノートは、次の節で構成されています。

このマニュアル内で参照している第三者の URL は、追加の関連情報を提供します。

このマニュアル内で引用する第三者の Web サイトの可用性について Sun は責任を負いません。こうしたサイトやリソース上の、またはこれらを通じて利用可能な、コンテンツ、広告、製品、その他の素材について、Sun は推奨しているわけではなく、Sun はいかなる責任も負いません。こうしたサイトやリソース上の、またはこれらを経由して利用可能な、コンテンツ、製品、サービスを利用または信頼したことに伴って発生した (あるいは発生したと主張される) いかなる損害や損失についても、Sun は一切の責任を負いません。

リリースノートの変更履歴

次の表は、Message Queue 製品のすべての 4.x リリースの日付と、各リリースに関連する主な変更内容を示しています。

表 1–1　変更履歴

日付	変更点
2006 年 5 月	Message Queue version 4.0 のこのマニュアルの初回リリース。
2007 年 1 月	Message Queue Beta versio 4.1 のこのマニュアルの初回リリース。JAAS サポートの説明が追加されています。
2007 年 4 月	Message Queue Beta versio 4.1 のこのマニュアルの 2 番目のリリース。高可用性の機能が追加されています。
2007 年 9 月	顧客の利便を考慮した、このマニュアルの 3 番目のリリース。Java Enterprise System Monitoring Framework のサポート、固定 C ポート、バグ修正、およびその他の機能に関する説明が追加されています。

Message Queue 4.1 について

Sun Java System Message Queue は、多くの機能を備えるメッセージサービスで、Java Messaging Specification (JMS) 1.1 に準拠する信頼性の高い非同期メッセージングを提供します。Message Queue では、JMS 仕様を超える機能も用意され大規模企業のシステム配備のニーズにも対応できるようになっています。

Message Queue の Version 4.1 では、高可用性、JASS (Java Authentication and Authorization Service)、固定 C ポートの使用、および Java Enterprise System Monitoring Framework のサポートが追加されています。また、いくつかの小さな機能拡張とバグ修正も追加されています。この節の構成は次のとおりです。

- 7 ページの「この 4.1 リリースでの新機能」
- 16 ページの「ハードウェアとソフトウェアの要件」

Message Queue 4.0 で導入された機能については、17 ページの「Message Queue 4.0 について」を参照してください。

この 4.1 リリースでの新機能

Message Queue 4.1 では、高可用性 (データおよびサービスの可用性) ブローカクラスタ、JAAS サポート、およびその他のさまざまな小機能が導入されています。この節では、これらの機能について説明し、参考資料も示します。

- 7 ページの「高可用性」
- 8 ページの「JAAS サポート」
- 14 ページの「持続ストアの形式の変更」
- 14 ページの「ブローカの設定」
- 15 ページの「JES Monitoring Framework のサポート」
- 16 ページの「トランザクションの管理」
- 16 ページの「C クライアント接続の固定ポート」

高可用性

Message Queue 4.1 では、データの可用性とサービスの可用性を提供する高可用性クラスタが導入されています。クライアントが高可用性ブローカへの接続を失った場合は、自動的にクラスタ内の別のブローカに再接続されます。新しい接続を提供するブローカは、障害が発生したブローカの持続データと持続状態を継承し、障害が発生したブローカのクライアントに中断することなくサービスの提供を続けます。高可用性ブローカは、セキュリティー保護された接続を介して実行できます。

高可用性ブローカでは、高可用性データベース (HADB) を使用する必要があります。そのようなデータベースがない場合、またはデータの可用性が重要でない場合は、自動再接続とサービスの可用性を提供する従来のクラスタを引き続き使用できます。

高可用性ブローカの設定は簡単です。クラスタ内のブローカごとに次の種類のブローカプロパティーを指定します。

- クラスタメンバーシッププロパティー。ブローカが高可用性クラスタの一部であることと、クラスタの ID およびブローカの ID を指定します。
- 高可用性データベース *(HADB)* プロパティー。持続メッセージ (JDBC) のモデル、HADB ベンダーの名前、およびベンダー固有のデータベースの設定プロパティーを指定します。
- 障害検出および継承プロパティー。ブローカ障害の検出と処理の方法を指定します。

この機能を使用するには、次の操作を実行する必要があります。

1. 高可用性データベースをインストールします。
2. JDBC ドライバの .jar ファイルをインストールします。
3. 高可用性持続ストア用のデータベーススキーマを作成します。
4. クラスタ内のブローカごとに、高可用性に関連したプロパティーを設定します。
5. クラスタ内の各ブローカを起動します。

高可用性の概念に関する説明、および従来のクラスタとの比較については、『Sun Java System Message Queue 4.1 Technical Overview』の第 4 章「Broker Clusters」を参照してください。高可用性の手続きおよび参照に関する情報については、『Sun Java System Message Queue 4.1 Administration Guide』の第 8 章「Broker Clusters」および『Sun Java System Message Queue 4.1 Administration Guide』の「Cluster Configuration Properties」を参照してください。

Message Queue version 4.0 で HADB データベースを使用していた場合、高可用性クラスタを使用するには、`dbmgr` ユーティリティーを使用して共有 HADB ストアにアップグレードできます。詳細は、46 ページの「ブローカクラスタ」を参照してください。

JAAS サポート

Message Queue では、ファイルベースおよび LDAP ベースの組み込み認証機構に加えて、JAAS (Java Authentication and Authorization Service) もサポートしています。JAAS を使用すると、さまざまなサービスをブローカに接続して Message Queue クライアントを認証できます。この節では、ブローカによって JAAS 準拠の認証サービスで利用可能になる情報について説明し、それらのサービスを使用するようにブローカを設定する方法についても説明します。

JAAS API についての説明は、このマニュアルの対象範囲ではありません。詳細を知る必要がある場合は、次の資料を参照してください。

- JAAS API の詳細は、『Java Authentication and Authorization Service (JAAS) Reference Guide』を参照してください。

  `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html`

- LoginModule の記述については、『Java Authentication and Authorization Service (JAAS) LoginModule Developer's Guide』を参照してください。

  `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASLMDevGuide.html`

JAAS API は J2SE のコア API であるため、Message Queue の実行環境の必須部分です。JAAS は、アプリケーションと認証機構の間の抽象化層を定義し、アプリケーションコードを変更せずに目的の機構をプラグインできるようにします。Message Queue サービスの場合、抽象化層はブローカ (アプリケーション) と認証プロバイダの間にあります。いくつかのブローカプロパティーを設定することで、JAAS 準拠の任意の認証サービスに接続して、ブローカコードを中断または変更せずにそのサービスをアップグレードできます。

JAAS 準拠の認証を使用している場合は、JMX クライアントを使用してブローカを管理できますが、ブローカを起動する前に手動で JAAS サポートを設定する必要があります (JAAS 関連のブローカプロパティーを設定することで行う)。JMX API を使用してこれらのプロパティーを変更することはできません。

JAAS の要素

図 1–1 に、JAAS の基本要素である JAAS クライアント、JAAS 準拠の認証サービス、および JAAS 設定ファイルを示します。

- JAAS クライアントは、JAAS 準拠の認証サービスを使用して認証を実行しようとするアプリケーションです。JAAS クライアントは、LoginModule を使用してこのサービスと通信し、LoginModule がユーザー名、パスワード、およびその他の関連情報を取得するために呼び出すことができるコールバックハンドラの提供を担当します。
- JAAS 準拠の認証サービスは、1 つ以上の LoginModule と、必要な認証を実行するロジックで構成されます。LoginModule には認証ロジックが含まれる場合があります。また、LoginModule は、プライベートプロトコルまたは API を使用して、認証ロジックを提供するモジュールと通信する場合があります。
- JAAS 設定ファイルは、JAAS 準拠のサービスとの通信に必要な LoginModule を検出するために JAAS クライアントが使用するテキストファイルです。

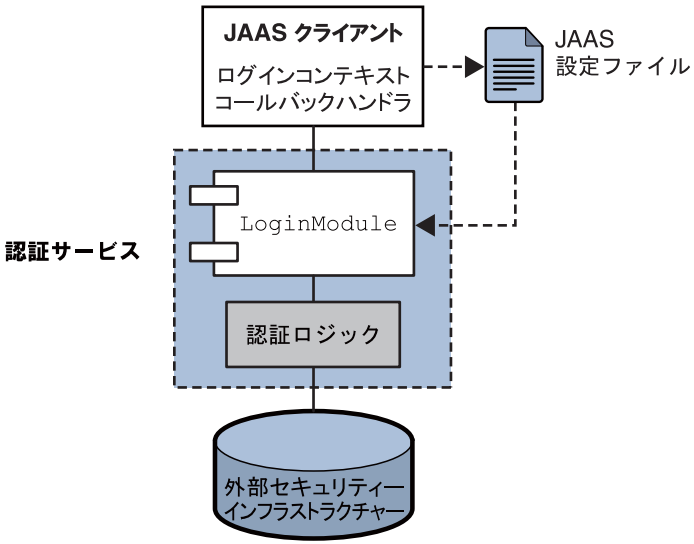

図 1–1　JAAS 要素

次の節では、Message Queue サービスがこれらの要素を使用して JAAS 準拠の認証を提供する方法について説明します。

JAAS と Message Queue

次の図に、Message Queue ブローカが JAAS を使用する方法を示します。この図は、前の図に示した JAAS モデルのより複雑な実装を示しています。

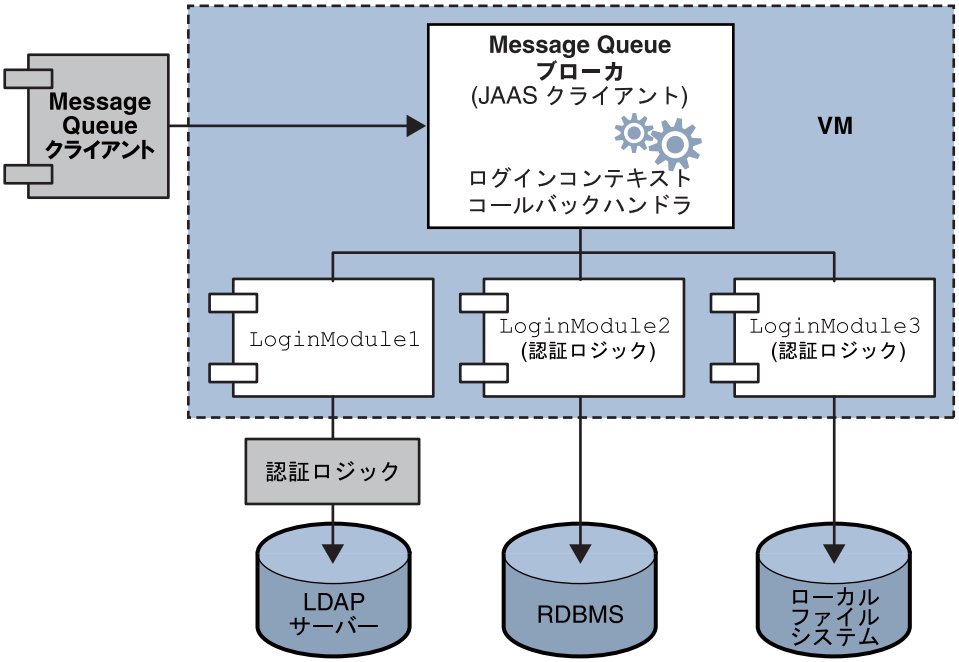

図 1–2　Message Queue が JAAS を使用する方法

簡単な方の事例に示されているように、認証サービス層はブローカとは別になっています。認証サービスは、1 つ以上のログインモジュール (LoginModule) と、必要に応じて追加する認証モジュールで構成されます。ログインモジュールは、ブローカと同じ Java 仮想マシンで実行されます。Message Queue ブローカは、ログインモジュールに対して `LogInContext` として表現され、ブローカ実行時コードの一部である `CallBackHandler` によってログインモジュールと通信します。

認証サービスは、ログインモジュールのエントリを含む JAAS 設定ファイルも提供します。設定ファイルは、モジュールを使用する順序、およびモジュールの使用に関する一部の条件を指定します。ブローカが起動すると、JAAS は Java システムプロパティー `java.security.auth.login.config` または Java セキュリティープロパティーファイルによって設定ファイルを検出します。次に、JAAS は、ブローカプロパティー `imq.user_repository.jaas.name` の値に応じて、JAAS 設定ファイル内のエントリを選択します。そのエントリは、認証に使用するログインモジュールを指定します。図に示すように、ブローカは複数のログインモジュールを使用できます。(設定ファイル、ログインモジュール、およびブローカの関係は、図 1–3 に示されています。)

ブローカが JAAS プラグイン認証サービスを使用するという事実は、Message Queue クライアントに対して完全に透明なままです。クライアントは引き続き以前と同じようにブローカに接続し、ユーザー名とパスワードを渡します。同様に、ブローカはコールバックハンドラを使用してこの情報を認証サービスに渡し、認証サービスはその情報を使用してユーザーを認証し、結果を返します。認証に成功した場合、ブローカは接続を許可します。認証に失敗した場合、クライアントランタイムはクライアントが処理する必要のある JMS セキュリティー例外を返します。

Message Queue クライアントの認証後に、さらに行うべき承認がある場合、ブローカは通常どおり続行されます。ブローカはアクセス制御ファイルを参照し、認証されたクライアントが引き受けるアクションを実行することを承認するかどうかを決定します。それらのアクションには、送信先へのアクセス、メッセージの消費、キューの表示などがあります。

JAAS 準拠の認証の設定

JAAS 準拠の認証の設定には、ブローカおよびシステムプロパティーを設定して、このタイプの認証を選択し、設定ファイルの場所を指定し、使用するログインモジュールのエントリを指定することが関係します。

この節では、JAAS クライアント、ログインモジュール、および JAAS 設定ファイルの関係を図で示し、次に JAAS 準拠の認証の設定に必要なプロセスについて説明します。次の図に、設定ファイル、ログインモジュール、およびブローカの関係を示します。

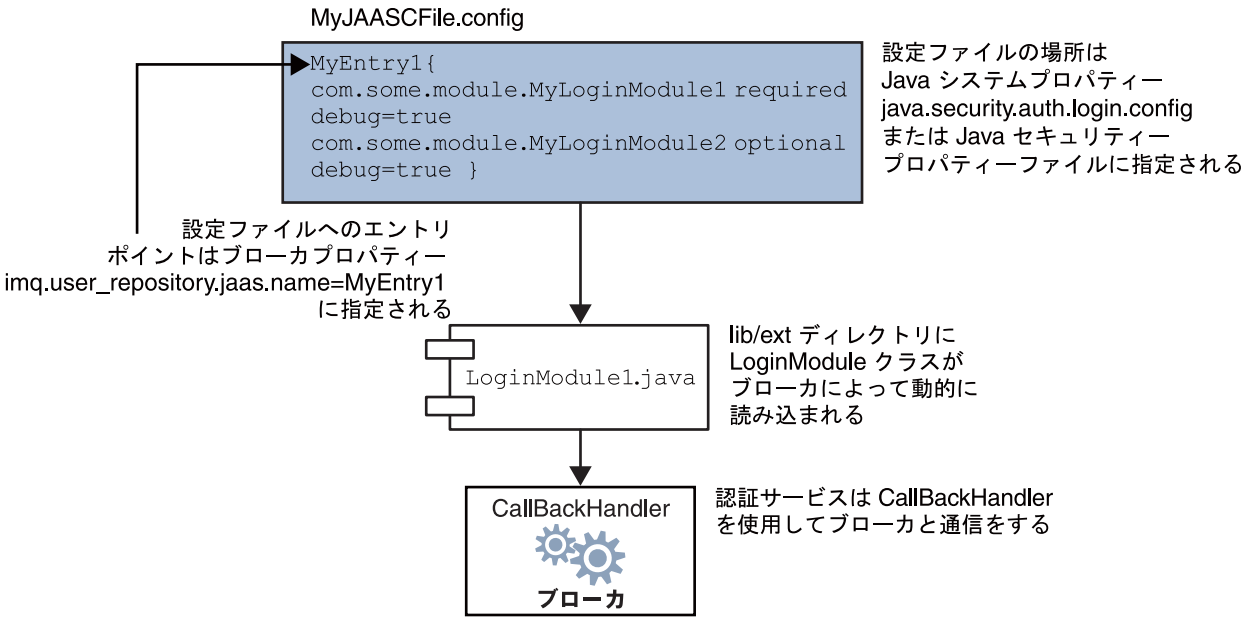

図 1–3　JAAS サポートの設定

図に示すように、JAAS 設定ファイル `MyJAASCFile.config` には、エントリポイントにグループ化されたいくつかのログインモジュールへの参照が含まれています。ブローカは、Java システムプロパティー `java.security.auth.login.config` または Java セキュリティープロパティーファイルを参照することで、設定ファイルを検出します。使用するログインモジュールは、ブローカプロパティー `imq.user_repository.jaas.name` を参照することで決定されます。このブローカプロパティーは、設定ファイル内の目的のエントリを指定します。これらのモジュールのためのクラスは、`lib/ext` ディレクトリにあります。

Message Queue 用の JAAS サポートを設定するには、次の手順を完了する必要があります。(開発環境では、これらの手順はすべて開発者が実行する場合があります。本稼働環境では、管理者がこれらのタスクの一部を引き継ぐことになります。)

1. 認証サービスを実装する 1 つ以上のログインモジュールクラスを作成します。ブローカがサポートする JAAS コールバックのタイプを下に一覧表示します。

   `javax.security.auth.callback.LanguageCallback`
   ブローカはこのコールバックを使用して、ブローカが動作しているロケールを認証サービスに渡します。この値は、ローカリゼーションに使用できます。

   `javax.security.auth.callback.NameCallback`
   ブローカはこのコールバックを使用して、接続の要求時に Message Queue クライアントによって指定されたユーザー名を認証サービスに渡します。

`javax.security.auth.callback.TextInputCallback`
ブローカはこのコールバックを使用して、`TextInputCallback.getPrompt()` が `imq.authentication.type` のときに、`imq.authentication.type` の値を認証サービスに指定します。現在、このフィールドで唯一可能な値は `basic` です。これは Base-64 パスワードエンコーディングを示しています。

`javax.security.auth.callback.PasswordCallback`
ブローカはこのコールバックを使用して、接続の要求時に Message Queue クライアントによって指定されたパスワードを認証サービスに渡します。

`javax.security.auth.callback.TextOutputCallback`
ブローカはこのコールバックを使用して、テキスト出力をブローカのログファイルに記録することで、認証サービスにログサービスを提供します。コールバックの MessageType `ERROR`、`INFORMATION`、`WARNING` はそれぞれ、ブローカログレベル `ERROR`、`INFO`、および `WARNING` にマップされます。

2. ログインモジュールクラスを参照するエントリを持つ JAAS 設定ファイルを作成し、このファイルの場所を Message Queue 管理者に指定します。(このファイルはリモートから検出でき、その場所を URL で指定できます。)
3. JAAS 設定ファイル内のエントリ (ログイン実装クラスを参照する) の名前を書き留めます。
4. ログインモジュールを実装するクラスを jar ファイルに保存し、その jar ファイルを Message Queue の `lib/ext` ディレクトリに配置します。
5. JAAS サポートに関連したブローカプロパティーを設定します。この点については、表 1–2 で説明しています。
6. 次のシステムプロパティーを設定して、JAAS 設定ファイルの場所を指定します。

   `java.security.auth.login.config=` *JAAS_Config_File_Location*

   たとえば、ブローカを起動するときに設定ファイルを指定できます。

   `imqbrokerd -Djava.security.auth.login.config=`*JAAS_Config_File_Location*

   JAAS 設定ファイルの場所を指定する方法はほかにもあります。追加の情報については、次を参照してください。

   `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/tutorials/LoginConfigFile.html`

次の表に、JAAS サポートの設定に必要なブローカプロパティーの一覧を示します。

表 1–2 JAAS サポートのためのブローカプロパティー

プロパティー	説明
`imq.authentication.type`	`basic` に設定して、Base-64 パスワードエンコーディングを指定します。これは、JAAS 認証用に許可される唯一の値です。

表 1–2 JAAS サポートのためのブローカプロパティー (続き)

プロパティー	説明
imq.authentication.basic.user_repository	jaas に設定して、JAAS 認証を指定します。
imq.accesscontrol.type	file に設定します。
imq.user_repository.jaas.name	認証機構として使用するログインモジュールを参照する目的のエントリ (JAAS 設定ファイル内) の名前に設定します。これは、手順 3 で書き留めた名前です。
imq.user_repository.jaas.userPrincipalClass	Message Queue のアクセス制御で使用されるこのプロパティーは、Message Queue のアクセス制御ファイルでユーザーエンティティーを表現する主体名の抽出にブローカが使用するログインモジュールで java.security.Principal 実装クラスを指定します。このプロパティーが指定されていない場合は、接続の要求をしたときに Message Queue クライアントから渡されたユーザー名が代わりに使用されます。
imq.user_repository.jaas.groupPrincipalClass	Message Queue のアクセス制御で使用されるこのプロパティーは、Message Queue のアクセス制御ファイルでグループエンティティーを表現する主体名の抽出にブローカが使用するログインモジュールで java.security.Principal 実装クラスを指定します。このプロパティーが指定されていない場合は、Message Queue のアクセス制御ファイル内のグループ規則 (存在する場合) は無視されます。

持続ストアの形式の変更

Message Queue の Version 4.1 では、JDBC ストアは高可用性をサポートするように変更されています。この理由で、JDBC ストアのバージョンは 410 に増えています。JDBC ストアのバージョン 350、370、および 400 は、自動的に 410 バージョンの形式に移行されます。

ファイルベースの持続ストアのバージョンは、何も変更がないので、370 のままです。

ブローカの設定

プロパティー IMQ_DEFAULT_EXT_JARS が imqenv.conf ファイルに追加されました。このプロパティーを設定して、ブローカが起動するときに CLASSPATH に含まれる外部 .jar ファイルのパス名を指定できます。このプロパティーを使用して外部 .jar ファイルの場所を指定した場合は、これらのファイルを lib/ext ディレクトリにコピーする必要はなくなります。外部 jar は、JDBC ドライバまたは JAAS ログインモジュールを参照できます。次のサンプルコマンドは、JDBC ドライバの場所を指定します。

```
IMQ_DEFAULT_EXT_JARS=/opt/SUNWhadb4/lib/hadbjdbc4.jar:/opt/SUNWjavadb/derby.jar
```

JES Monitoring Framework のサポート

Message Queue は、Sun Java Enterprise System (JES) Monitoring Framework をサポートしています。この Monitoring Framework は、一般的なグラフィカルインタフェースを使用して Java Enterprise System コンポーネントを監視できるようにします。このインタフェースは、Sun Java System Monitoring Console と呼ばれる Web ベースのコンソールによって実装されます。ほかの JES コンポーネントと一緒に Message Queue を実行している場合は、1 つのインタフェースを使用してこれらすべてのコンポーネントを管理するほうが便利なことがあります。

JES Monitoring Framework は、すべての JES コンポーネント製品で使用する共通データモデル (CMM) を定義します。このモデルにより、すべての JES コンポーネントを集中管理の単一形式で表示できます。Message Queue は、次のオブジェクトを JES Monitoring Framework に明示します。

- インストールされている製品
- ブローカのインスタンス名
- ブローカのポートマッパー
- 各接続サービス
- 物理的な各送信先
- 持続ストア
- ユーザーリポジトリ

これらの各オブジェクトは CMM オブジェクトにマップされ、その属性は JES 監視コンソールを使用して監視できます。管理者は、実行時にコンソールを使用してパフォーマンス統計を表示したり、自動的に監視する規則を作成したり、アラームを確認したりできます。Message Queue オブジェクトから CMM オブジェクトへのマッピングの詳細は、『Sun Java Enterprise System 監視ガイド』を参照してください。

JES 監視を有効にするには、次の操作を実行する必要があります。

1. 『Sun Java Enterprise System インストールガイド』に記載されている手順に従って、配備中のすべてのコンポーネント (Message Queue およびその他のコンポーネント) をインストールおよび設定します。
2. 『Sun Java Enterprise System 監視ガイド』の説明に従って、監視対象のすべてのコンポーネントに対して Monitoring Framework を有効にして設定します。
3. 『Sun Java Enterprise System 監視ガイド』の説明に従って、別のホストに Monitoring Console をインストールし、マスターエージェントを起動してから、Web サーバーを起動します。

JES Monitoring Framework を使用しても、ブローカのパフォーマンスには影響しません。メトリックスの収集作業はすべて Monitoring Framework によって行われ、データはブローカの既存の監視データインフラストラクチャーから取得されるためです。

トランザクションの管理

以前は、管理上、PREPARED 状態のトランザクションだけをロールバックすることができました。つまり、分散トランザクションの一部であるセッションが正常に終了しなかった場合、そのトランザクションは、ブローカ管理者がクリーンアップできない状態のままになりました。Message Queue 4.1 では、imqcmd ユーティリティーを使用して、STARTED、 FAILED、INCOMPLETE、COMPLETE、 PREPARED の状態にあるトランザクションをクリーンアップ (ロールバック) できます。

特定のトランザクションがロールバックできるかどうか (特に、PREPARED 状態でない場合) の判別に役立つように、imqcmd ユーティリティーは、追加のデータを imqcmd query txn 出力の一部として提供します。このデータは、そのトランザクションを開始した接続のコネクション ID を提供し、トランザクションが作成された時間を示します。管理者は、この情報を使用して、トランザクションをロールバックする必要があるかどうかを決定できます。一般に、管理者は早計にトランザクションをロールバックすることを避けるとよいでしょう。

C クライアント接続の固定ポート

C クライアントは、MQ_SERVICE_PORT_PROPERTY 接続プロパティーを使用して、接続先の固定ポートを指定できます。これは、ファイアウォールを通過しようとしている場合、またはブローカのポートマッパーサービス (動的にポートを割り当てる) をバイパスする必要がある場合に便利なことがあります。

ブローカ側でも JMS サービスポートを設定する必要があることを忘れないでください。たとえば、ssljms を介してクライアントをポート 1756 に接続する場合は、次のように操作します。

- クライアント側: MQ_SERVICE_PORT_PROPERTY を 1756 に設定し、MQ_CONNECTION_TYPE_PROPERTY を SSL に設定します。
- ブローカ側: 次のようにして、imq.*serviceNameType.protocol* .port プロパティーを 1756 に設定します。

```
imq.ssljms.ssl.port=1756
```

注 – MQ_SERVICE_PORT_PROPERTY 接続プロパティーは、Message Queue の version 3.7 Update 2 で導入されました。

ハードウェアとソフトウェアの要件

Version 4.1 のハードウェアとソフトウェアの要件については、『Sun Java System Message Queue 4.1 Installation Guide』を参照してください。

Message Queue 4.0 について

Message Queue 4.0 は、Application Server 9 PE のサポートに限定されたリリースです。マイナーリリースには、いくつかの新機能と機能拡張、およびバグ修正が含まれています。この節の構成は次のとおりです。

4.0 リリースでの新機能

Message Queue 4.0 に含まれる新機能は次のとおりです。

以降の節で、これらの機能について説明しています。

注意 – version 4.0 には、軽度とはいえ、状況によって十分な対応が必要になる変更が導入されています。その 1 つとして、パスワードを指定するコマンド行オプションが使用されなくなったことが挙げられます。今後は、43 ページの「使用されなくなったパスワードオプション」で説明するように、すべてのパスワードをファイルに保存する必要があります。

C API および C クライアントランタイムのインタフェースの変更

Message Queue の Version 4.0 では、デッドメッセージキューに配置されたすべてのメッセージで設定される 2 つの新しいプロパティーが追加されています。

- `JMS_SUN_DMQ_PRODUCING_BROKER` は、メッセージが生成されたブローカを指定します。
- `JMS_SUN_DMQ_DEAD_BROKER` は、メッセージをデッドとして指定したブローカを指定します。

Java API および Java クライアントランタイムのインタフェースの変更

Message Queue の Version 4.0 では、デッドメッセージキューに配置されたすべてのメッセージに設定される 2 つの新しいプロパティーが追加されています。

- JMS_SUN_DMQ_PRODUCING_BROKER は、メッセージが生成されたブローカを指定します。
- JMS_SUN_DMQ_DEAD_BROKER は、メッセージをデッドとして指定したブローカを指定します。

持続ストアに関する情報の表示

imqdbmgr コマンドに query サブコマンドが追加されました。このサブコマンドは、持続ストアの情報 (ストアバージョン、データベースユーザー、データベーステーブルが作成されたかどうかなど) を表示するために使用します。

次に、このコマンドによって表示される情報の例を示します。

```
imqdbmgr query

[04/Oct/2005:15:30:20 PDT] Using plugged-in persistent store:
        version=400
        brokerid=Mozart1756
        database connection url=jdbc:oracle:thin:@Xhome:1521:mqdb
        database user=scott
Running in standalone mode.
Database tables have already been created.
```

持続ストアの形式の変更

Message Queue の Version 3.7 UR1 では、パフォーマンスを向上するために持続ストアの形式に 2 つの変更が加えられました。1 つはファイルストアに対する変更で、もう 1 つは JDBC ストアに対する変更です。

- ファイルストア内で持続されるトランザクションデータの形式

  ディスク I/O を軽減し、JMS トランザクションのパフォーマンスを向上させるために、Message Queue ファイルベースの持続ストア内に格納されているトランザクション状態情報の形式が変更されました。

- Oracle JDBC ストア

  以前のバージョンの Message Queue では、Oracle のストアスキーマでは、メッセージデータを格納するために LONG RAW データ型が使用されていました。Oracle 8 では、Oracle によって BLOB データ型が導入され、LONG RAW 型は使用されなくなりました。Message Queue 3.7 UR1 では、パフォーマンスとサポート性を向上するために、データ型が BLOB に変更されました。

これらの変更がストアの互換性にも影響し、Message Queue version 3.7 UR1 では、ファイルストアと JDBC ストアの両方のストアバージョンが 350 から 370 に変更されました。

Message Queue version 4.0 では、最適化と将来の機能拡張をサポートするために、JDBC ストアが変更されました。このため、JDBC ストアのバージョンが 400 になりました。ただし Version 4.0 では、ファイルベースの持続ストアには変更が行われていないため、このファイルストアのバージョンは 370 のままです。

Message Queue 4.0 は、ファイルベースの持続ストアと JDBC の持続ストアの最新バージョンへの持続ストアの自動変換をサポートしています。`imqbrokerd` を最初に起動したときに、ユーティリティーが古いバージョンのストアを検出した場合、古いストアはそのままで、ストアを新しい形式に移行します。

- ファイルベースのストアのバージョン 200 および 350 は、バージョン 370 の形式に移行されます。
- JDBC ストアのバージョン 350 および 370 は、バージョン 400 の形式に移行されます。バージョン 200 ストアをアップグレードする必要がある場合は、いったん中間的な release 3.5 または 3. 6 にアップグレードする必要があります。

このアップグレードをロールバックする必要のある場合は、Message Queue 4.0 をいったんアンインストールして、以前実行していたバージョンを再インストールします。ストアの古いコピーはそのまま残っているので、ブローカはストアの古いコピーを実行できます。

ブローカ管理

コマンドユーティリティー (`imqcmd`) に、サブコマンドといくつかのオプションが追加されました。管理者はこれらを使用して、ブローカを休止したり、指定した間隔の後でブローカをシャットダウンしたり、接続を破棄したり、Java システムのプロパティー (たとえば、コネクション関連のプロパティー) を設定したりできます。

- ブローカを休止すると休止状態になり、ブローカをシャットダウンまたは再起動する前に、メッセージを排出してしまうことができます。休止状態にあるブローカに新しく接続が作成されることはありません。ブローカを休止するには、次のようなコマンドを入力します。

  ```
  imqcmd quiesce bkr -b Wolfgang:1756
  ```

- 指定した間隔の後でブローカをシャットダウンするには、次のようなコマンドを入力します。(時間間隔は、ブローカをシャットダウンするまでの待機を秒数で指定します。)

  ```
  imqcmd shutdown bkr -b Hastings:1066 -time 90
  ```

  時間間隔を指定した場合、ブローカはシャットダウンが発生するタイミングを示すメッセージを記録します。以下にその例を示します。

  ```
  Shutting down the broker in 29 seconds (29996 milliseconds)
  ```

ブローカがシャットダウンを待っている間、ブローカの動作は次のような影響を受けます。

- 管理 JMS 接続は引き続き受け付けられます。
- 新しい JMS 接続は受け付けられません。
- 既存の JMS 接続は引き続き機能します。
- ブローカが高可用性クラスタ内のほかのブローカを継承することはできません。
- imqcmd ユーティリティーはブロックはせず、シャットダウンの要求をブローカに送信してすぐに返します。

- 接続を破棄するには、次のようなコマンドを入力します。

```
imqcmd destroy cxn -n 2691475382197166336
```

コマンド `imqcmd list cxn` または `imqcmd query cxn` を使用してコネクション ID を取得します。

- `imqcmd` を使用してシステムプロパティーを設定するには、新しい –D オプションを使用します。これは、JMS コネクションファクトリのプロパティー、またはコネクション関連の Java システムのプロパティーの設定または上書きに便利です。次に例を示します。

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
imqcmd list svc -secure -Djavax.net.ssl.trustStore=/tmp/mytruststore
            -Djavax.net.ssl.trustStorePassword=mytrustword
```

`imqcmd` コマンドの構文の詳細は、『Sun Java System Message Queue 4.1 Administration Guide』の第 13 章「Command Line Reference」を参照してください。

JDBC 持続サポート

Apache Derby Version 10.1.1 が、JDBC 準拠の持続ストアプロバイダとしてサポートされるようになりました。

SSL サポート

release 4.0 から、クライアントコネクションファクトリのプロパティー `imqSSLIsHostTrusted` のデフォルト値が `false` になりました。使用しているアプリケーションが以前のデフォルト値の `true` に依存している場合は、再設定を行い、プロパティーを明示的に `true` に設定する必要があります。

ブローカが自己署名付き証明書を使用するように設定されているときは、ホストを信頼することもできます。この場合、接続で SSL ベースの接続サービスを使用するように指定する (imqConnectionType プロパティーを使用する) ことに加えて、`imqSSLIsHostTrusted` プロパティーを true に設定することをお勧めします。

たとえば、ブローカが自己署名付き証明書を使用するときに安全にクライアントアプリケーションを実行するには、次のようなコマンドを使用します。

```
java -DimqConnectionType=TLS
      -DimqSSLIsHostTrusted=true <ClientAppName>
```

ブローカが自己署名付き証明書を使用するときに安全に管理ツール `imqcmd` を実行するには、次のようなコマンドを使用します。

```
imqcmd list svc -secure -DimqSSLIsHostTrusted=true
```

JMX サポート

Java Management Extensions (JMX) 仕様に準拠して、Message Queue ブローカを設定および監視するための新しい API が追加されました。この API を使用すると、Message Queue クライアントアプリケーション内部からプログラムによってブローカ関数を設定および監視することができます。以前のバージョンの Message Queue では、これらの関数にはコマンド行または管理コンソールからしかアクセスできませんでした。

この API は、一連の JMX *Managed Bean (MBean)* によって設定されており、次の Message Queue 関連のリソースを管理します。

- メッセージブローカ
- 接続サービス
- 接続
- 宛先
- メッセージプロデューサ
- メッセージコンシューマ
- トランザクション
- ブローカクラスタ
- ロギング
- Java Virtual Machine (JVM)

これらの MBean によって、基礎となるリソースの状態を、同期をとりながらポーリングおよび操作するための属性と操作が設定されます。また、状態が変更されたときに、クライアントアプリケーションが非同期で待機して応答できるようにする通知も設定されます。JMX API を使用して、クライアントアプリケーションは次のようなタスクを設定および監視することができます。

- ブローカのポート番号を設定する
- ブローカのメッセージの最大サイズを設定する
- 接続サービスを一時停止する
- 接続サービスの最大スレッド数を設定する
- サービス上の現在の接続数を取得する
- 接続を破棄する
- 送信先を作成する
- 送信先を破棄する
- 送信先の自動作成を有効または無効にする
- 送信先からすべてのメッセージを削除する

- ブローカの起動後に送信先の受信したメッセージの累積数を取得する
- キューの現在の状態 (実行中または一時停止) を取得する
- トピックのメッセージプロデューサの現在数を取得する
- 永続サブスクライバからすべてのメッセージを削除する
- 現在の JVM ヒープサイズを取得する

JMX API の紹介と詳細情報については、『Sun Java System Message Queue 4.1 Developer's Guide for JMX Clients』を参照してください。

ブローカサポート: JMX 関連のプロパティー

JMX API をサポートするために、いくつかの新しいブローカプロパティーが追加されました (表 1–3 を参照)。これらのプロパティーはいずれも Message Queue コマンドユーティリティー (`imqcmd`) によるコマンド行からは設定できません。その代わりに、ブローカユーティリティー (`imqbrokerd`) の `-D` オプションを使用するか、ブローカのインスタンス設定ファイル (`config.properties`) を手作業で編集することで設定できます。さらに、これらのプロパティーのいくつか (`imq.jmx.rmiregistry.start`、`imq.jmx.rmiregistry.use`、 `imq.jmx.rmiregistry.port`) は、表 1–4 に示されている新しいブローカユーティリティーオプションを使用して設定できます。次の表は、各オプションとその型、それぞれの使用方法について示しています。

表 1–3　JMX サポートのための新しいブローカプロパティー

プロパティー	型	説明
`imq.jmx.rmiregistry.start`	ブール型	ブローカの起動時に RMI レジストリを起動するかどうかを指定します。 `true` の場合、ブローカは `imq.jmx.rmiregistry.port` によって指定されたポートで RMI レジストリを起動し、これを使用して JMX コネクタ用の RMI スタブを格納します。この場合、`imq.jmx.rmiregistry.use` の値は無視されます。 デフォルト値 : `false`
`imq.jmx.rmiregistry.use`	ブール型	外部の RMI レジストリを使用するかどうかを指定します。 `imq.jmx.rmiregistry.start` が `false` の場合のみ適用されます。 `true` の場合、ブローカは `imq.jmx.rmiregistry.port` によって指定されたポートで外部の RMI レジストリを使用し、JMX コネクタ用の RMI スタブを格納します。この外部の RMI レジストリは、ブローカの起動時にすでに実行されている必要があります。 デフォルト値 : `false`

表 1–3 JMX サポートのための新しいブローカプロパティー *(続き)*

プロパティー	型	説明
imq.jmx.rmiregistry.port	整数	RMI レジストリのポート番号 imq.jmx.rmiregistry.start または imq.jmx.rmiregistry.use が true の場合のみ適用されます。このポート番号を JMX サービス URL の URL パスに追加することによって、JMX コネクタが RMI レジストリを使用するように設定できます。 デフォルト値: 1099
imq.jmx.connector.list	文字列	事前設定された JMX コネクタの名前。コンマで区切られます。 デフォルト値: jmxrmi,ssljmxrmi
imq.jmx.connector.activelist	文字列	ブローカの起動時にアクティブになる JMX コネクタの名前。コンマで区切られます。 デフォルト値: jmxrmi
imq.jmx.connector.*connectorName* .urlpath	文字列	コネクタ *connectorName* の、JMX サービス URL の *urlPath* コンポーネント JMX サービス URL パスが明示的に指定されている場合 (共有の外部 RMI レジストリが使用されている場合など) に有効です。 デフォルト値: JMX コネクタ用の RMI スタブを格納するために RMI レジストリが使用されている場合 (つまり imq.jmx.registry.start または imq.jmx.registry.use が true の場合) /jndi/rmi://*brokerHost*:*rmiPort* /*brokerHost*/*brokerPort*/*connectorName* RMI レジストリが使用されていない場合 (デフォルトの場合。つまり、 imq.jmx.registry.start と imq.jmx.registry.use の両方が false の場合): /stub/*rmiStub* ここで *rmiStub* は、RMI スタブそのものをエンコードおよび直列化した表現です。
imq.jmx.connector.*connectorName* .useSSL	ブール型	コネクタ *connectorName* に Secure Socket Layer (SSL) を使用するかどうかを指定します。 デフォルト値: false

表 1–3　JMX サポートのための新しいブローカプロパティー　(続き)

プロパティー	型	説明
imq.jmx.connector.*connectorName* .brokerHostTrusted	ブール型	コネクタ *connectorName* のブローカによって提示された任意の証明書を信頼するかどうかを指定します。 imq.jmx.connector. *connectorName*.useSSL が true の場合のみ適用されます。 false の場合、Message Queue クライアントランタイムは提示されたすべての証明書を検証します。証明書の署名者がクライアントのトラストストア内に存在しない場合、検証は失敗します。 true の場合、証明書の検証はスキップされます。これは、ソフトウェアのテスト中で自己署名した証明書が使用されている場合などに便利です。 デフォルト値: false

imq.jmx.connector.list プロパティーは、ブローカの起動時に作成される一連の名前付き JMX コネクタを定義します。imq.jmx.connector.activelist では、これらの中でアクティブにするものを指定します。名前付きのコネクタには、それぞれ独自のプロパティーセットがあります。

imq.jmx.connector.*connectorName* .urlpath
imq.jmx.connector.*connectorName* .useSSL
imq.jmx.connector.*connectorName* .brokerHostTrusted

デフォルトでは、 jmxrmi および ssljmxrmi という名前の 2 つの JMX コネクタが作成されます。前者は SSL 暗号化を使用しないように設定 (imq.jmx.connector.jmxrmi.useSSL = false) され、後者は使用するように設定 (imq.jmx.connector.ssljmxrmi.useSSL = true) されています。デフォルトでは、ブローカの起動時には jmxrmi コネクタのみがアクティブになります。通信のセキュリティー保護のために ssljmxrmi コネクタをアクティブにする方法については、25 ページの「JMX クライアント用の SSL サポート」を参照してください。

使いやすくするために、コマンド行のブローカユーティリティー (imqbrokerd) にも、RMI レジストリの使用方法、起動、ポートを制御するための新しいオプション (表 1–4) が追加されています。これらのオプションの使用方法と効果は、表 1–3 に示されているブローカプロパティーの中の対応するものと同じです。次の表では、各オプションとともに対応するブローカプロパティーを示し、その使用方法についても説明します。

表 1–4 JMX サポートのための新しいブローカユーティリティーオプション

オプション	対応するブローカプロパティー	説明
`-startRmiRegistry`	`imq.jmx.rmiregistry.start`	ブローカの起動時に RMI レジストリを起動するかどうかを指定します。
`-useRmiRegistry`	`imq.jmx.rmiregistry.use`	外部の RMI レジストリを使用するかどうかを指定します。
`-rmiRegistryPort`	`imq.jmx.rmiregistry.port`	RMI レジストリのポート番号

ブローカの起動時に作成および起動される JMX コネクタの JMX サービス URL を表示するために、コマンド行のコマンドユーティリティー (`imqcmd`) に新しいサブコマンド (表 1–5) が追加されました。この情報は、JMX コネクタを取得するために Message Queue の簡易クラス `AdminConnectionFactory` を使用しない JMX クライアントにとって必要になります。また、Java Monitoring and Management Console (`jconsole`) などの汎用の JMX ブラウザを介して Message Queue を管理または監視するために使用することもできます。

表 1–5 新しいコマンドユーティリティーのサブコマンド

サブコマンド	説明
`list jmx`	JMX コネクタの JMX サービス URL を表示します

JMX クライアント用の SSL サポート

これまでに説明したように、Message Queue のメッセージブローカは、デフォルトでは、事前に設定済みの JMX コネクタ `jmxrmi` を使用して、セキュリティー保護されていない通信用に設定されています。通信のセキュリティー保護のために SSL (Secure Socket Layer) を使用する必要のあるアプリケーションでは、代わりにセキュリティー保護された JMX コネクタである `ssljmxrmi` をアクティブにする必要があります。このためには次の手順を実行する必要があります。

1. 『Message Queue 管理ガイド』に示されている `ssljms`、`ssladmin`、または `cluster` 接続サービスと同じ方法で、署名済みの証明書を取得およびインストールします。
2. 必要に応じて、ルート CA 証明書をトラストストア内にインストールします。
3. ブローカの起動時にアクティブになるように、JMX コネクタのリストに `ssljmxrmi` コネクタを追加します。

   ```
   imq.jmx.connector.activelist=jmxrmi,ssljmxrmi
   ```

4. Message Queue ブローカユーティリティー (`imqbrokerd`) をパスワードファイル内のキーストアパスワードに渡すか、プロンプト表示されるコマンド行から入力して、ブローカを起動します。

5. デフォルトでは、ssljmxrmi コネクタ、またはその他の SSL ベースのコネクタは、提示されるすべてのブローカ SSL 証明書を検証するように設定されています。この検証を回避する場合、ソフトウェアのテスト中で自己署名の証明書を使用している場合などは、ブローカプロパティー imq.jmx.connector.ssljmxrmi.brokerHostTrusted を true に設定します。

クライアント側では、ssljmxrmi を優先コネクタとして指定した URL を使用して管理者のコネクションファクトリ (AdminConnectionFactory) を設定する必要があります。

```
AdminConnectionFactory  acf = new AdminConnectionFactory();
acf.setProperty(AdminConnectionConfiguration.imqAddress, "mq://myhost:7676/ssljmxrmi");
```

必要に応じて、システムプロパティー javax.net.ssl.trustStore および javax.net.ssl.trustStorePassword を使用して、JMX クライアントをトラストストアにポイントします。

クライアントランタイムログ

この節では、Message Queue 4.0 による接続のクライアントランタイムログのサポートと、セッション関連のイベントについて説明します。

JDK 1.4 以上には、java.util.logging ライブラリが含まれています。このライブラリは、アプリケーション固有のログに使用される標準のロガーインタフェースを実装しています。

Message Queue のクライアントランタイムは Java Logging API を使用してログ機能を実装します。ログ活動を設定するために、すべての J2SE 1.4 ロギング機能を使用できます。たとえば、Message Queue クライアントランタイムがログ情報を出力する方法を設定するために、アプリケーションは次の Java ロギング機能を使用することができます。

- ログハンドラ
- ログフィルタ
- ログフォーマッタ
- ログレベル

Java Logging API の詳細は、http://java.sun.com/javase/ja/6/docs/ja/technotes/guides/logging/overview.html にある Java ロギングの概要を参照してください。

ログの名前空間、レベル、活動

Message Queue プロバイダは、ログレベルやログ活動に関連付けて一連のログの名前空間を定義します。Message Queue クライアントは、ログ設定が適切であれば、この名前空間を使用して接続やセッションイベントをログに記録することができます。

Message Queue クライアントランタイムのルートのログ名前空間は、javax.jms と定義されます。Message Queue クライアントランタイム内のすべてのロガーが、この名前を親の名前空間として使用します。

Message Queue クライアントランタイムに使用されるログレベルは、java.util.logging.Level クラス内で定義されるものと同じです。このクラスでは、7 つの標準ログレベルと、ログのオン/オフに使用できる 2 つの追加設定が定義されます。

OFF	ログをオフにします。
SEVERE	優先順位が最高である最高値。アプリケーションで定義します。
WARNING	アプリケーションで定義します。
INFO	アプリケーションで定義します。
CONFIG	アプリケーションで定義します。
FINE	アプリケーションで定義します。
FINER	アプリケーションで定義します。
FINEST	優先順位が最低である最低値。アプリケーションで定義します。
ALL	すべてのメッセージのログを有効にします。

一般に、Message Queue クライアントランタイム内で発生した例外やエラーは、名前空間 javax.jms を使用するロガーによってログ記録されます。

- JVM からスローされクライアントランタイムによってキャッチされる例外 (IOException など) は、ログ名前空間 javax.jms を使用するロガーによって WARNING レベルでログ記録されます。
- クライアントランタイムからスローされる JMS 例外 (IllegalStateException など) は、ログ名前空間 javax.jms を使用するロガーによって FINER レベルでログ記録されます。
- JVM からスローされクライアントランタイムによってキャッチされるエラー (OutOfMemoryError など) は、ログ名前空間 javax.jms を使用するロガーによって SEVERE レベルでログ記録されます。

以下の表は、JMS コネクションとセッションについて、ログ記録できるイベントとログイベントに対して設定する必要のあるログレベルを示しています。

次の表は、コネクションのログレベルとイベントについて示したものです。

表 1–6　名前空間 javax.jms.connection のログレベルとイベント

ログレベル	イベント
FINE	接続が作成されました
FINE	接続が起動しました
FINE	接続が閉じました
FINE	接続が破棄されました
FINE	再接続されました
FINER	その他の接続アクティビティー (setClientID など)
FINEST	メッセージ、通知、Message Queue アクション、制御メッセージ (トランザクションのコミットなど)

セッションの場合、次の情報がログレコードに記録されます。

- コンシューマに配信されるメッセージの各ログレコードには、ConnectionID、SessionID、ConsumerID が含まれています。
- プロデューサによって送信されるメッセージの各ログレコードには、ConnectionID、SessionID、ProducerID、送信先名が含まれています。

次の表は、セッションのログレベルとイベントについて示したものです。

表 1–7　名前空間 javax.jms.session のログレベルとイベント

ログレベル	イベント
FINE	セッションが作成されました
FINE	セッションが閉じました
FINE	プロデューサが作成されました
FINE	コンシューマが作成されました
FINE	送信先が作成されました
FINER	その他のセッションアクティビティー (セッションのコミットなど)。
FINEST	メッセージがプロデュースされ、消費されました。(メッセージのプロパティーと本文はログレコードに記録されません。)

デフォルトでは、出力ログレベルはアプリケーションの実行されている JRE から継承されます。JRE_DIRECTORY/lib/logging.properties ファイルを参照してレベルを確認してください。

ログはプログラムや設定ファイルを使用して設定できます。また、ログの発生する範囲を制御することもできます。次の節では、これらの機能について説明します。

JRE ロギング設定ファイルの使用

次の例は、JRE_DIRECTORY/lib/logging.properties ファイル内でログの名前空間とログレベルを設定する方法を示しています。この方法は Java ランタイム環境のログレベルの設定に使用されます。JRE を使用するすべてのアプリケーションが同じログ設定になります。次に示すサンプル設定では、名前空間 javax.jms.connection のログレベルを INFO に設定し、その出力が java.util.logging.ConsoleHandler に書き込まれるように指定しています。

```
#logging.properties file.
# "handlers" specifies a comma separated list of log Handler
# classes. These handlers will be installed during VM startup.
# Note that these classes must be on the system classpath.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.

    handlers= java.util.logging.ConsoleHandler

# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers. For any given facility this global level
# can be overriden by a facility-specific level.
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.

    .level= INFO

# Limit the messages that are printed on the console to INFO and above.

    java.util.logging.ConsoleHandler.level = INFO
    java.util.logging.ConsoleHandler.formatter =
                                    java.util.logging.SimpleFormatter

# The logger with javax.jms.connection name space will write
# Level.INFO messages to its output handler(s). In this configuration
# the ouput handler is set to java.util.logging.ConsoleHandler.

    javax.jms.connection.level = INFO
```

特定のアプリケーションに対するログ設定ファイルの使用

アプリケーションの実行に使用する Java コマンド行からログ設定ファイルを定義することもできます。このアプリケーションは、指定のログファイル内で定義された

設定を使用します。次の例では、configFile は JRE_DIRECTORY/lib/logging.properties ファイル内で定義されているものと同じ形式を使用します。

```
java -Djava.util.logging.config.file=configFile MQApplication
```

プログラムによるログ設定

次のコードでは、java.util.logging API を使用して、javax.jms.connection 名前空間のログレベルを FINE に変更することで、接続イベントをログ記録しています。このようなコードをアプリケーションに追加して、プログラムによるログ設定を行うことができます。

```
import java.util.logging.*;
//construct a file handler and output to the mq.log file
//in the system's temp directory.

    Handler fh = new FileHandler("%t/mq.log");
    fh.setLevel (Level.FINE);

//Get Logger for "javax.jms.connection" domain.

    Logger logger = Logger.getLogger("javax.jms.connection");
    logger.addHandler (fh);

//javax.jms.connection logger would log activities
//with level FINE and above.

    logger.setLevel (Level.FINE);
```

接続イベント通知

接続イベント通知を使用すると、Message Queue クライアントは接続のクローズおよび再接続イベントを待機して、通知タイプと接続状態に基づいて適切なアクションを起こすことができます。たとえば、フェイルオーバーが発生してクライアントが別のブローカに再接続された場合、アプリケーションはそのトランザクションの状態をクリーンアップしてから新しいトランザクションに進む必要があるかもしれません。

Message Queue プロバイダは、接続について深刻な問題を検出した場合、接続オブジェクトの登録済みの例外リスナーを呼び出します。このプロバイダは、リスナーの onException メソッドを呼び出し、問題を記述する JMSException 引数にこれを渡します。Message Queue プロバイダはイベント通知 API も提供し、これを使用してクライアントランタイムは、接続状態の変更をアプリケーションに通知できます。通知 API は次の要素によって定義されます。

- com.sun.messaging.jms.notification パッケージ。イベントリスナーと通知イベントオブジェクトを定義します。
- com.sun.messaging.jms.Connection インタフェース。javax.jms.Connection インタフェースへのエクステンションを定義します。

次の節では、通知をトリガーできるイベントと、イベントリスナーの作成方法について説明します。

接続イベント

次の表は、イベントリスナーによって返されるイベントについて説明したものです。

接続イベントの発生時に JMS 例外リスナーは呼び出されません。例外リスナーが呼び出されるのは、クライアントランタイムの再接続の試行回数が上限に達してしまった場合のみです。クライアントランタイムは、常に例外リスナーの前にイベントリスナーを呼び出します。

表 1–8 通知イベント

イベントタイプ	意味
ConnectionClosingEvent	Message Queue クライアントランタイムは、管理者のシャットダウン要求によって接続がクローズされようとしているという通知をブローカから受信したときに、このイベントを生成します。
ConnectionClosedEvent	Message Queue クライアントランタイムは、ブローカのエラーによって接続がクローズされたか、管理者のシャットダウン要求または再起動要求によって接続がクローズされたときに、このイベントを生成します。 イベントリスナーが ConnectionClosedEvent を受信すると、アプリケーションは受信したイベントの getEventCode() メソッドを使用して、クローズの原因を指定するイベントコードを取得します。

表 1–8 通知イベント (続き)

イベントタイプ	意味
ConnectionReconnectedEvent	Message Queue クライアントランタイムがブローカに再接続されました。このブローカは、このクライアントが以前接続していたものと同じ場合もありますが、別のブローカの場合もあります。 アプリケーションは、受信したイベントの getBrokerAddress メソッドを使用して、再接続先のブローカのアドレスを取得することができます。
ConnectionReconnectFailedEvent	Message Queue クライアントランタイムがブローカへの再接続に失敗しました。再接続の試みに失敗するたびに、ランタイムは新しいイベントを生成し、それをイベントリスナーに配信します。 接続イベントの発生時に JMS 例外リスナーは呼び出されません。呼び出されるのは、クライアントランタイムの再接続の試行回数が上限に達してしまった場合のみです。クライアントランタイムは、常に例外リスナーの前にイベントリスナーを呼び出します。

イベントリスナーの作成

次のコード例は、接続イベントリスナーの設定方法を示したものです。接続イベントが発生するたびに、イベントリスナーの onEvent メソッドがクライアントランタイムによって呼び出されます。

```
//create an MQ connection factory.

com.sun.messaging.ConnectionFactory factory =
        new com.sun.messaging.ConnectionFactory();

//create an MQ connection.

com.sun.messaging.jms.Connection connection =
       (com.sun.messaging.jms.Connection )factory.createConnection();

//construct an MQ event listener.  The listener implements
//com.sun.messaging.jms.notification.EventListener interface.

com.sun.messaging.jms.notification.EventListener eListener =
       new ApplicationEventListener();

//set event listener to the MQ connection.
```

```
connection.setEventListener ( eListener );
```

イベントリスナーの例

この例では、アプリケーションが、そのイベントリスナーが接続イベントをアプリケーションのロギングシステムに記録するように選択します。

```
public class ApplicationEventListener implements
               com.sun.messaging.jms.notification.EventListener {

public void onEvent ( com.sun.messaging.jms.notification.Event connEvent ) {
          log (connEvent);
}
private void log ( com.sun.messaging.jms.notification.Event connEvent ) {
          String eventCode = connEvent.getEventCode();
          String eventMessage = connEvent.getEventMessage();
          //write event information to the output stream.
         }
}
```

ハードウェアとソフトウェアの要件

Version 4.0 のハードウェアとソフトウェアの要件については、Sun Java System Application Server Platform Edition 9 の『リリースノート』を参照してください。

今回のリリースで修正されたバグ

次の表に、Message Queue の 4.1 バージョンで修正されたバグを示します。

表 1–9　Message Queue 4.1 で修正されたバグ

バグ	説明
6381703	処理されたリモートメッセージが、そのメッセージの発信元のブローカが再起動された場合に、2 回コミットされることがあります。
6388049	未完了の分散トランザクションをクリーンアップできません。
6401169	imqcmd のコミットおよびロールバックオプションで、確認のプロンプトが表示されません。
6473052	自動作成されたキューのデフォルトはラウンドロビンであるべきです。(`MaxNumberConsumers` = -1)。

表 1–9 Message Queue 4.1 で修正されたバグ (続き)

バグ	説明
6474990	ブローカのログに、`imqcmd list dst` コマンドに対して `ConcurrentModificationException` が示されます。
6487413	LimitBehavior が `REMOVE_OLDEST` または `REMOVE_LOWER_PRIORITY` のときに、メモリーがリークします。
6488340	ブローカがスピンし、クライアントは確認に対する応答を待機します。
6502744	ブローカが、デッドメッセージキューのデフォルト制限 (1000 のメッセージ) を尊重しません。
6517341	クライアントが高可用性クラスタへ接続している際の、再接続ロジックを、*imqReconnectEnabled* プロパティーの値に関係なく再接続を可能にするように改善すべきです。
6528736	起動時に Windows 自動起動サービス (*imqbrokersvc*) に障害が発生します。
6561494	両者が 1 つのセッションを共有していると、メッセージが間違ったコンシューマに配信されます。
6567439	`PREPARED` トランザクションの生成メッセージがブローカの再起動後にコミットされた場合、それらのメッセージは順序どおりに配信されません。

次の表に、Message Queue 4.0 で修正されたバグを示します。

表 1–10 Message Queue 4.0 で修正されたバグ

バグ番号	説明
4986481	Message Queue 3.5 では、自動再接続モードでの `Session.recover` の呼び出しがハングすることがありました。
4987325	`Session.recover` の呼び出し後に、再配信されたフラグは再配信されたメッセージに対して `false` に設定されました。
6157073	新しく接続メッセージが変更され、接続の合計数のほかにサービス上の接続数も表示されるようになりました。
6193884	Message Queue は、メッセージに非アスキー文字を使用するロケールで文字化けメッセージを syslog に出力します。
6196233	`JMSMessageID` を使用したメッセージ選択が機能しません。
6251450	クラスタのシャットダウン時、 `connectList` に `ConcurrentModificationException` が発生します。
6252763	`java.nio.HeapByteBuffer.putLong/Int` に `java.nio.BufferOverflowException` が発生します。

表 1–10　Message Queue 4.0 で修正されたバグ　　(続き)

バグ番号	説明
6260076	Oracle ストレージを使用すると、起動後に発行される最初のメッセージが遅くなります。
6260814	JMSXUserID 上のセレクタ処理が、常に false と評価されてしまいます。
6264003	キューブラウザにコミットされていないメッセージが表示されます。
6271876	消費されていないメッセージを含むコンシューマを閉じようとすると、接続フロー制御が正しく動作しなくなります。
6279833	Message Queue では、2 つのブローカが同じ JDBC テーブルを使用することはできません。
6293053	システム IP アドレスが変更された場合、–reset store を使用してストアをクリアしていないと、マスターブローカが正しく起動しません。
6294767	Message Queue ブローカがネットワークソケットを開く場合、そのネットワークソケット上に SO_REUSEADDR を設定する必要があります。
6304949	TopicConnectionFactory に ClientID プロパティーを設定することができません。
6307056	txn ログがパフォーマンス上のボトルネックになっています。
6320138	Message Queue C API では reply-to ヘッダーからキューの名前を決定することができません。
6320325	Solaris 上に JDK 1.4 と JDK 1.5 の両方がインストールされている場合、ブローカが JDK 1.5 より前に JDK 1.4 を検出してしまうことがあります。
6321117	マルチブローカクラスタの初期化で java.lang.NullPointerException が発生します。
6330053	サブスクライバからトランザクションをコミットしている場合、JMS クライアントで java.lang.NoClassDefFoundError が発生します。
6340250	C-API で MESSAGE タイプをサポート。
6351293	Apache Derby データベースへのサポートを追加。

重要情報

この節には、製品の主要マニュアルには含まれていない最新の情報が含まれています。この節は、次のトピックで構成されています。

インストール時の注意事項

Solaris、Linux、および Windows プラットフォーム上に Message Queue Platform Edition をインストールする場合のインストール前の指示、アップグレード手順、その他の関連情報については、『Sun Java System Message Queue 4.1 Installation Guide 』を参照してください。

Message Queue Enterprise Edition を Solaris、Linux、および HPUX の各プラットフォームにインストールする際の、インストール前の手順、および他の関連情報については、『 Sun Java Enterprise System インストールガイド』を参照してください。

Solaris、Linux、HPUX および Windows プラットフォーム上の Message Queue Enterprise Edition へのアップグレードに関連するアップグレードおよび移行の指示については、『Sun Java Enterprise System Upgrade and Migration Guide』を参照してください。

互換性の問題

この節では、Message Queue 4.1 の互換性の問題を説明しています。

インタフェースの安定性

Sun Java System Message Queue で使用される多くのインタフェースは、時間の経過につれて変更される可能性があります。『Sun Java System Message Queue 4.1 Administration Guide』の付録 B「Stability of Message Queue Interfaces」では、インタフェースをそれらの安定性に従って分類しています。安定性に優れるインタフェースほど、製品の後継バージョンで変更される可能性は低くなります。

Message Queue の次回のメジャーリリースに関係する問題

Message Queue の次回のメジャーリリースでは、クライアントがこのリリースとの互換性がなくなるような変更が導入される可能性があります。この情報は、このような変更に備えていただく目的で今回提供しています。

- Sun Java System Message Queue の一部としてインストールされる各ファイルの場所が変更される可能性があります。これによって、特定の Message Queue ファイルの現在の場所に依存する既存のアプリケーションの動作が中断する可能性があります。
- 3.5 以前のブローカは、これより新しいブローカのクラスタ内では動作できなくなる可能性があります。
- 今後のリリースでは、Message Queue クライアントは 1.5 より前のバージョンの JDK を使用できなくなる可能性があります。

Message Queue 4.1 に関するマニュアルの更新

この『リリースノート』のほかに、Message Queue 4.1 にはもう 1 つ新しいマニュアルが追加されています。それは、『Sun Java System Message Queue 4.1 Developer's Guide for JMX Clients 』です。このマニュアルは、Message Queue の 4.0 リリースで導入されました。4.1 バージョンでは、JMX モデルを紹介する概念上の情報が追加されました。

Message Queue 3.6 SP3, 2005Q4 用に発行された Message Queue のマニュアルが、Application Server 9 PE クライアントの要件に関連して更新されました。このマニュアルセットは次の場所で入手できます。

`http://docs.sun.com/app/docs/coll/1307.1`

また、一部のマニュアルは、翻訳されており、`http://docs.sun.com/app/docs/coll/1374.1` から入手できます。

インストールとアップグレードに関する情報

『Sun Java System Message Queue 4.1 Installation Guide 』は、プラットフォーム固有の情報を反映するように更新されました。このマニュアルには、Message Queue 4.1 に関連したインストールおよびアップグレードの情報が含まれるようになりました。

『管理ガイド』

『管理ガイド』が、高可用性クラスタ、JAAS サポート、および JMX サポートの情報を提供するように更新されました。

『Developer's Guide for Java Clients』

『Developer's Guide for Java Clients』は、クライアントランタイムのログサポート、および接続イベント通知の追加情報を反映するように更新されました。

『Developer's Guide for C Clients』

『Developer's Guide for C Clients』は、`MQGetDestinationName` 関数、`MQ_Message` メッセージタイプ、および固定ポートの追加情報を反映するように更新されました。

既知の問題点と制限事項

この節には、Message Queue 4.1 の既知の問題についてのリストが含まれています。次の内容について説明します。

現時点のバグ、その状態、および回避策の一覧については、Java Developer Connection™ メンバーは、Java Developer Connection Web サイトの「Bug Parade」ページを参照してください。新しいバグを報告する前に、このページをチェックしてください。すべての Message Queue バグがリストされているわけではありませんが、このページはある問題が報告済みかどうかを知りたい場合に活用できます。

`http://bugs.sun.com/bugdatabase/index.jsp`

注 – Java Developer Connection のメンバーになるのは無料ですが、登録が必要です。Java Developer Connection のメンバーになる方法についての詳細は、Sun の「For Developers」Web ページを参照してください。

新しいバグの報告や機能に関する要求を行うには、`imq-feedback@sun.com` 宛てにメールを送信してください。

インストールに関する情報

この節では、Message Queue version 4.1 のインストールに関連した問題について説明します。

製品レジストリと JES

Message Queue の Version 4.1 は、新しいインストーラでインストールされます。このインストーラは、JDK、NSS ライブラリ、JavaHelp など、Message Queue が必要とする共有コンポーネントもインストールおよびアップグレードします。このインストーラと Java Enterprise System (JES) インストーラは、同じ製品レジストリを共有しません。JES でインストールされたあるバージョンの Message Queue が削除され、Message Queue インストーラで Message Queue 4.1 にアップグレードされた場合、JES 製品レジストリは矛盾する状態になることがあります。その結果、JES アンインストーラを実

行すると、JES でインストールしなかった Message Queue 4.1 とそれが依存する共有コンポーネントが意図せずに削除されることがあります。

JES インストーラでインストールしたソフトウェアをアップグレードする最善の方法は、次のとおりです。

1. JES アンインストーラを使用して、Message Queue とその共有コンポーネントを削除します。
2. Message Queue インストーラを使用して、Message Queue 4.1 をインストールします。

適切な JRE の選択

Message Queue 4. 1 Installer JDK Selection Screen を使用すると、Message Queue で使用するシステムで既存の JDK/JRE を選択できます。残念ながら、示されているリストには、インストーラアプリケーションの実行に使用する JRE も含まれています。この JRE は、インストーラバンドルの一部であり、実際にシステムにインストールされることはありません。*(バグ 6585911)*

インストーラが使用する JRE はそのパスによって判別可能です。このパスは、解凍されたインストーラディレクトリ内に存在し、サブディレクトリ `mq4_1–installer` が含まれているはずです。次に例を示します。

some_directory`/mq4_1–installer/usr/jdk/instances/jdk1.5.0/jre`

この JRE は Message Queue での使用には選択しないでください。代わりに、システム上の別の JDK を選択してください。別の JDK が存在しない場合は、プラットフォームに適した処置を講じてください。

- Solaris または Linux の場合:「Java(TM) SDK のデフォルトバージョンをインストールして使用します。」を選択します。
- Windows の場合: Message Queue 4.1 インストーラを実行する前に、JDK をダウンロードしてインストールします。

Windows でのインストール

Message Queue を Windows にインストールするときは、次の制限事項に注意してください。

- インストーラは、Message Queue 用のエントリを「スタート」>「すべてのプログラム」メニューに追加しません *(バグ 6567258)*。管理コンソールを起動するには、『Sun Java System Message Queue 4.1 Administration Guide』の「Starting the Administration Console」に示されているように、コマンド行を使用します。
- インストーラは、`IMQ_HOME\mq\bin` ディレクトリを `PATH` 環境変数に追加しません。*(バグ 6567197)*。ユーザーは、Message Queue ユーティリティー (`IMQ_HOME\mq\bin\`*command*) を起動するときに、このエントリを `PATH` 環境変数に追加するか、フルパス名を指定する必要があります。

- インストーラは、Message Queue がインストールされたことを示すために Windows レジストリにエントリを追加しません。
- サイレントモードで実行すると、インストーラはすぐに返されます。インストールは行われますが、ユーザーにはサイレントインストールがいつ実際に実行されたかを知る方法がありません。 *(バグ 6586560)*
- テキストモード (`installer –t`) は Windows ではサポートされていません。Windows でテキストモードでインストーラを実行すると、エラーメッセージが表示されます。このメッセージは、インストーラが英語以外のロケールで実行されていても、英語で表示されます。*(バグ 6594142)*
- インストーラの「Install Home」画面に表示される文字列「Install Home」は、インストーラが英語以外のロケールで実行されていても、英語で表示されます。*(バグ 6592491)*

Solaris でのインストール

エラーメッセージおよび「未完了」の概要状態は、`installer-n` コマンドを使用してインストールしようとしているユーザーの誤解を招きます。コマンドは実際には正常に動作しています。*(バグ 6594351)*

Linux でのインストール

次に示す問題は、Linux プラットフォームでのインストールに影響します。

- 「JDK 選択」パネルで、スクロールリストに項目が 1 つしか表示されません。このため、リスト内のほかの JDK を選択することが難しくなります。*(バグ 6584735)*
- JDK が現在のもので、ユーザーが「JDK 選択」画面で「Java(TM) SDK のデフォルトバージョンをインストールして使用します。」を選択した場合、インストーラはその JDK をインストールしようとし、パッケージをインストールできないというレポートを返します。この問題があっても、インストールは正常に完了します。*(バグ 6581310)*
- インストーラを予行モード (`installer –n`) で実行すると、「概要」画面にいくつかのエラーメッセージが表示され、「未完了」のインストール状態も表示されます。これは誤りで、誤解を招きます。予行モードでは、何もシステムにインストールされません。後でインストールに使用できる回答ファイルが作成されるだけです。*(バグ 6594351)*
- 古いバージョンの Message Queue のローカライズ版 RPM がシステムに存在する場合、Message Queue 4.1 のローカライズ版 RPM のインストール (「多言語パッケージ」の画面で「Message Queue 多言語パッケージをインストールする」を選択した場合) は失敗します。インストールは、前の 3.7 UR1 インストールの Il8 パッケージとの衝突のために失敗します。 *(バグ 6594381)*

回避策: 4.1 Installer を実行する前に、`rpm –e` コマンドを使用して手動でローカライズ版 RPM を削除します。ここで関係のある RPM を判別するには、『Sun Java System Message Queue 4.1 Installation Guide』の「Message Queue Packages (RPMs)」を参照してください。

すべてのプラットフォームでのインストール

次に示す問題は、すべてのプラットフォームでのインストールに影響します。

- インストーラが Message Queue 4.1 のインストールを処理中で、「進行状況」画面が表示されているときは、「取消し」ボタンがアクティブになります。ここで「取消し」ボタンを選択すると、インストールが完了しないか、破棄されます。 *(バグ 6595578)*
- インストーラの「概要」画面には、クリックするとログまたは概要ページビューアが開くリンクがいくつか含まれています。「閉じる」というラベルの付いたボタンの代わりに、ウィンドウの閉じるボタン「X」を使用してこのビューアウィンドウを取り消した場合は、このビューアウィンドウを再度表示することはできません。*(バグ 6587138)*

  回避策:「閉じる」というラベルの付いたボタンを使用してウィンドウを閉じます。
- システムに古いバージョンの Message Queue および NSS/NSPR があると、インストーラの「アップグレード」で、アップグレードが必要な Message Queue のリストだけが表示され、NSS/NSPR のアップグレードが必要であることは触れられません。これは、すべての関連ソフトウェアがインストールプロセスの一部としてアップグレードされる (正しい情報が表示される「インストール準備完了」画面に示される) ことに関係した、「更新」画面の問題にすぎません。*(バグ 6580696)*

  回避策: NSS/NSPR ファイルが最新版でない場合は最新版がインストールされ、古いバージョンがアンインストールされるため、回避の必要はありません。
- インストーラまたはアンインストーラがテキストモード (`installer –t`) で実行されているときは、「概要」画面にログファイルまたは概要ファイルを含むディレクトリが表示されますが、これらのファイルの名前のリストは表示されません。*(バグ 6581592)*
- 存在しないファイルの名前を指定すると、矛盾した不明瞭なエラーメッセージが生成されます。 *(バグ 6587127)*

バージョン情報

インストーラには、Message Queue のバージョン情報が隠された形式で表示されます。*(バグ 6586507)*

Solaris プラットフォームの場合は、下の表を参照して、インストールされているバージョンを判別してください。

表 1–11 バージョンの形式

インストーラで表示されるバージョン	Message Queue のリリース
4.1.0.0	4.1
3.7.0.1	3.7 UR1
3.7.0.2	3.7 UR2
3.7.0.3	3.7 UR3
3.6.0.0	3.6
3.6.0.1	3.6 SP1
3.6.0.2	3.6 SP2
3.6.0.3	3.6 SP3
3.6.0.4	3.6 SP4

注 – 3.6 SP4 のパッチリリースの場合 (たとえば、3.6 SP4 Patch 1)、インストーラで表示されるリリース文字列は同じままです。厳密なバージョンを判別するには、コマンド `imqbrokerd –version` を実行する必要があります。

Linux プラットフォームの場合は、簡単に形式を変換できません。Linux 上のインストーラで表示されるバージョン番号は、次の形式になります。

```
<majorReleaseNumber>.<minorReleaseNumber>-<someNumber>
```

たとえば、3.7–22 のようになります。これにより、3.7 リリースの 1 つであることがわかりますが、どの特定のバージョンかはわかりません。それを判別するには、コマンド `imqbrokerd –version` を実行します。

ローカリゼーションの問題

次に示す問題は、ローカリゼーションの問題に関係しています。

- インストーラを英語以外のロケールでテキストモード (`installer –t`) で実行すると、マルチバイト文字が文字化けします。 *(バグ 6586923)*
- ユーザーは、インストーラの「概要」画面を使用して「概要」レポートを表示できます。残念ながら、マルチバイトのロケールでインストーラを実行すると、このレポート (HTML ページ) は文字化けします。*(バグ 6587112)*

  回避策: HTML ファイルを編集して、その中に指定されている文字セットを訂正します。HTML ファイルには、次のような部分が含まれているはずです。

  ```
  meta http-equiv=” Content-Type” content=” text/html; charset=UTF-8
  ```

「UTF-8」を「*locale_name* .UTF-8」に置き換えます。たとえば、Solaris の場合は ja_JA.UTF-8 または ko.UTF-8 に、Linux の場合は ja_JA.utf8 または ko_KO.utf8 にします。

- インストーラの「進行状況」画面で、進行状況のバーに見慣れない文字が表示されます。ツールヒントは、英語以外のロケールではハードコードされています。*(バグ 6591632)*
- テキストモード (installer –t) は Windows ではサポートされていません。Windows でテキストモードでインストーラを実行すると、エラーメッセージが表示されます。このメッセージは、インストーラが英語以外のロケールで実行されていても、ローカライズされません。*(バグ 6594142)*
- インストーラの「ライセンス」画面には、インストーラが実行されるロケールに関係なく、英語のライセンステキストが表示されます。*(バグ 6592399)*

  回避策: ローカライズされたライセンスファイルにアクセスするには、LICENSE_MULTILANGUAGE.pdf ファイルを検索してください。
- インストーラの使用方法に関するヘルプテキストがローカライズされていません。*(バグ 6592493)*
- インストーラの概要 HTML ページに表示される文字列「None」は、英語でハードコードされています。*(バグ 6593089)*
- 著作権のページは、フランス語以外のロケールにはローカライズされていません。*(バグ 6590992)*
- インストーラをドイツ語のロケールで実行すると、ほかのロケールでは表示される開始画面のテキストが正しく表示されません。*(バグ 6592666)*
- インストーラの「Install Home」画面に表示される文字列「Install Home」がローカライズされていません。その文字列は、英語以外のロケールでインストーラを実行しても、英語で表示されます。*(バグ 6592491)*
- テキストモード (installer –t) でインストーラを実行すると、どのロケールでインストーラを実行するかに関係なく、英語の応答選択肢「Yes」および「No」が使用されます。*(バグ 6593230)*
- インストーラの「JDK 選択」画面の参照ボタンのツールヒントは、英語でハードコードされています。*(バグ 6593085)*

使用されなくなったパスワードオプション

以前のバージョンの Message Queue では、–p または –password オプションを使用して、次のようなコマンドのパスワードを対話形式で指定することができました。imqcmd、imqbrokerd、imdbmgr version 4.0 から、これらのオプションは使用できなくなりました。パスワードを指定するには次の手順に従います。

1. パスワードのみを格納するために使用するファイルで、パスワードプロパティーを必要な値に設定します。

パスワードファイル内で、次の構文を使用してパスワードを指定します。

PasswordPropertyName= MyPassword

2. –passfile オプションを使用してパスワードファイルの名前を渡します。

パスワードファイルには、次に示すパスワードを 1 つ以上格納することができます。

- SSL キーストアを開くために使用するキーストアパスワード。このパスワードを指定するには imq.keystore.password プロパティーを使用します。
- 接続が匿名でない場合に LDAP ディレクトリとの接続のセキュリティーを確保するために使用する LDAP リポジトリパスワード。このパスワードを指定するには imq.user_repository.ldap.password プロパティーを使用します。
- JDBC 準拠のデータベースとの接続に使用する JDBC データベースパスワード。このパスワードを指定するには imq.persist.jdbc.vendorName.password プロパティーを使用します。プロパティー名の *vendorName* コンポーネントは、データベースベンダーを指定する変数です。hadb、derby、pointbase、oracle、mysql から選択できます。
- imqcmd コマンド (ブローカ管理タスクを実行する) に対するパスワード。このパスワードを指定するには imq.imqcmd.password プロパティーを使用します。

次の例では、JDBC データベースに対するパスワードに abracadabra を設定しています。

```
imq.persist.jdbc.mysql.password=abracadabra
```

ブローカがユーザー作成のパスワードファイルを使用するように設定するには、次のいずれかの方法を実行します。

- ブローカの config.properties ファイルに次のプロパティーを設定する。

  ```
  imq.passfile.enabled=true
  imq.passfile.dirpath=MyFileDirectory
  imq.passfile.name=MyPassfileName
  ```

- imqbrokerd コマンドの –passfile オプションを使用する。

  ```
  imqbrokerd –passfile MyPassfileName
  ```

一般的な問題

この節では Message Queue 4.1 の一般的な問題を説明しています。中には以前のバージョンの Message Queue で発生した問題もあります。

- HTTP トランスポートを使用している JMS クライアントが、Ctrl-C の使用などにより突然終了した場合、ブローカがクライアント接続や関連するすべてのリソースを解放するまでに、およそ 1 分かかります。

この 1 分の間にクライアントのほかのインスタンスが起動し、同じ ClientID、永続サブスクリプション、またはキューを使おうとした場合、そのインスタンスは「クライアント ID はすでに使用されています」の例外を受け取ります。このことは実際の問題ではなく、上記の終了処理の結果にすぎません。およそ 1 分経過後にクライアントが起動すると、すべて問題なく動作します。

- SOAP クライアント。これまでは、`mail.jar` および `mail.jar` を参照するために使用する SAAJ 1.2 実装 JAR が `CLASSPATH` 内に存在する必要はありませんでした。SAAJ 1.3 では、この参照が削除されたため、Message Queue クライアントが `mail.jar` を明示的に `CLASSPATH` 内に配置する必要があります。

管理および設定上の問題

次に示す問題は Message Queue の管理および設定に関係するものです。

- Windows コンピュータで `CLASSPATH` に二重引用符が含まれている場合、`imqadmin` および `imqobjmgr` ユーティリティーはエラーをスローします (バグ *ID 5060769*)。

  回避策: エラーメッセージは無視できます。ブローカはコンシューマへのエラーの通知を正しく処理します。このエラーは、システムの信頼性には影響を与えません。

- すべての Solaris および Windows スクリプトで、`-javahome` オプションの値に空白文字が含まれると動作しない (バグ *ID 4683029*)。

  `javahome` オプションは Message Queue コマンドおよびユーティリティーで使用し、使用する代替の Java 2 互換のランタイムを指定します。ただし、代替の Java 2 互換のランタイムへのパスには、空白文字を含めることはできません。空白文字を含むパスの例は次のとおりです。

  Windows の場合: `C:/jdk 1.4`

  Solaris の場合: `/work/java 1.4`

  回避策: Java ランタイムを、空白文字が含まれない場所またはパスにインストールします。

- `imqQueueBrowserMaxMessagesPerRetrieve` 属性は、クライアントランタイムがキューの内容を検索するときに一度に取得することのできるメッセージの最大数を指定します。ただし、クライアントアプリケーションは常にキューにあるすべてのメッセージを取得します。このため `imqQueueBrowserMaxMessagesPerRetrieve` 属性は、キューに入れられたメッセージがチャンクされ、クライアントランタイムに配信される方法 (より少ない大型チャンク、またはより多い小型チャンク) には影響しますが、参照されるメッセージの総数には影響しません。この属性の値を変更するとパフォーマンスに影響が出る可能性はありますが、クライアントアプリケーションの取得するデータ量が変動することはありません (バグ *ID 6387631*)。

ブローカの問題

次に示す問題は Message Queue ブローカに影響します。

- ブローカをラウンドロビン配信用に設定する方法に関していくらか混乱がありました。解決策は簡単で、設定可能なものです。

  1. 送信先属性 `maxNumActiveConsumers` を -1 に設定します。これにより、ラウンドロビン配信がオンになります。
  2. 送信先属性 `consumerFlowLimit` を 1 に設定します。これにより、配信が次のコンシューマに進む前に、1 つのコンシューマに配信されるメッセージの数が指定されます。別のチャンクの場合は、この属性を必要な値に設定します。デフォルトでは、100 のメッセージがコンシューマごとに配信されます。

- 持続ストアがあまりにも多くの送信先を開く場合、ブローカがアクセス不可能になる (バグ *ID 4953354*)。

  回避策: この状態はブローカがシステムのオープンファイル記述子の制限に達したことが原因です。Solaris や Linux では、`ulimit` コマンドを使って、ファイル記述子の制限を増やします。

- 送信先が破棄された場合、コンシューマが孤立する (バグ *ID 5060787*)。

  送信先が破棄された場合、アクティブコンシューマが孤立します。いったんコンシューマが孤立すると、送信先が再作成された場合でもメッセージを受信しなくなります。

  回避策: この問題には、回避策がありません。

ブローカクラスタ

次に示す問題は、クラスタ化したブローカに影響します。

- このリリースでは、フル接続のブローカクラスタのみサポートされています。つまり、クラスタ内のすべてのブローカは、そのクラスタ内のほかのブローカと相互に直接やり取りする必要があります。`imqbrokerd -cluster` コマンド行引数を使用してブローカを接続する場合は、そのクラスタ内のすべてのブローカが含まれていることを確認してください。
- HADB を使用するブローカは、10M バイトより大きなメッセージを処理できません。*(バグ 6531734)*
- クライアントが高可用性ブローカに接続されている場合、クライアントランタイムは、`imqAddressListIterations` 値が何に設定されているかに関係なく、成功するまで再接続を試みます。
- クラスタの一部であるブローカに接続されているクライアントは、現在 `QueueBrowser` を使用して該当するクラスタ内のリモートブローカにあるキューを検索することはできません。クライアントが検索できるのは、直接接続されているブローカにあるキューの内容のみです。この場合でも、クライアントは、クラ

スタ内の任意のブローカに対してキューにメッセージを送信したりキューからのメッセージを消費したりできます。制約を受けるのは検索のみです。

- 従来のクラスタでは、4.1 ブローカを 3.x ブローカと一緒にクラスタ化する場合は、プロパティー `imq.autocreate.queue.maxNumActiveConsumers=1` を 4.1 ブローカ用に設定する必要があります。そうしないと、ブローカはクラスタ接続を確立できません。
- 高可用性クラスタに変換するときに、Message Queue Manager ユーティリティー (imqdbmgr) を使用して、既存のスタンドアロン HADB 持続データストアを共有 HADB ストアに変換できます。コマンドは次のとおりです。

  ```
  imqdbmgr upgrade hastore
  ```

  このユーティリティーは次の場合に使用できます。

  - 4.0 スタンドアロン HADB ストアから 4.1 共有 HADB ストアに移行する。この場合、ブローカは自動的にストアをアップグレードします。次に、`imqdbmgr` コマンドを実行して、アップグレードしたデータストアを共有で使用するために変換できます。
  - スタンドアロン 4.1 HADB ストアから共有 HADB ストアに移行する。この場合は、上記の `imqdbmgr` コマンドを実行して、データストアを共有で使用するために変換する必要があるだけです。

  このコマンドは HADB ストアの変換をサポートするだけなので、このコマンドを使用してファイルベースのストアまたはほかの JDBC ストアを共有 HADB ストアに変換することはできません。以前に 3.x バージョンの Message Queue を実行していた場合は、高可用性機能を使用するためには、HADB ストアを作成してから、データを手動でそのストアに移行する必要があります。
- コマンド `imqdbmgr upgrade hastore` を使用した HADB ストアへの変換は、ストアに 10,000 を超えるメッセージが保持されている場合、「設定されているロックの数が多すぎます」というメッセージが表示されて失敗することがあります。(バグ *ID 6588856)*)。

  (回避策:) 次のコマンドを使用して、ロックの数を増やします。

  ```
  hadbm set NumberOfLocks=<desiredNumber>
  ```

  詳細は、『Sun Java System Application Server 9.1 Enterprise Edition トラブルシューティングガイド』の「HADB の問題」を参照してください。
- 1 つのトランザクションに 500 を超えるリモートメッセージがコミットされている場合、ブローカはエラー「HADB-E-12815: Table memory space exhausted」を返すことがあります。 *(バグ ID 6550483)*

  詳細は、『Sun Java System Application Server 9.1 Enterprise Edition トラブルシューティングガイド』の「HADB の問題」を参照してください。
- ブローカクラスタで、開始していないリモート接続へのメッセージをブローカがキューに入れます (バグ *ID 4951010*)。

回避策: いったんその接続が開始すると、メッセージはコンシューマによって受信されます。コンシューマの接続が閉じている場合、メッセージは別のコンシューマへ再配信されます。

- 1 つのトランザクションでリモートブローカから複数のメッセージを消費しているときは、次のエラーメッセージがブローカに記録される可能性があります。このメッセージは無視しても問題ありません。

```
[26/Jul/2007:13:18:27 PDT] WARNING [B2117]:
Message acknowledgement failed from
mq://129.145.130.95:7677/?instName=a&brokerSessionUID=3209681167602264320:
    ackStatus = NOT_FOUND(404)\
    Reason = Update remote transaction state to COMMITED(6):
transaction 3534784765719091968 not found, the transaction
may have already been committed.
    AckType = MSG_CONSUMED
    MessageBrokerSession = 3209681167602264320
    TransactionID = 3534784765719091968
        SysMessageID = 8-129.145.130.95(95:fd:93:91:ec:a0)-33220-1185481094690
        ConsumerUID = 3534784765719133952\par

[26/Jul/2007:13:18:27 PDT] WARNING Notify commit transaction
[8-129.145.130.95(95:fd:93:91:ec:a0)-33220-1185481094690,
[consumer:3534784765719133952, type=NONE]]
TUID=3534784765719091968 got response:
com.sun.messaging.jmq.jmsserver.util.BrokerException:
 Update remote transaction state to COMMITED(6):
  transaction 3534784765719091968 not found, the transaction may have already
  been committed.:
com.sun.messaging.jmq.jmsserver.util.BrokerException: Update remote transaction
state to COMMITED(6): transaction 3534784765719091968 not found, the transaction
 may have already been committed.r
```

このメッセージは、`imq.txn.reapLimit` プロパティーが 1 つのトランザクション内のリモートメッセージの数と比較して少ない場合に、トランザクション内の後のメッセージのためにコミットをメッセージホームブローカに通知するときに記録されます。*(バグ 6585449)*

回避策: このメッセージを回避するには、`imq.txn.reapLimit` プロパティーの値を増やします。

JMX の問題

Windows プラットフォームでは、トランザクションマネージャーの監視 MBean の `getTransactionInfo` メソッドが不正なトランザクションの作成時間を含むトランザクション情報を返します (バグ *ID 6393359*)。

回避策: 代わりにトランザクションマネージャーの監視 MBean の `getTransactionInfoByID` メソッドを使用します。

SOAP のサポート

SOAP サポートに関連した 2 つの問題に注意する必要があります。

- Message Queue の version 4.0 のリリースから、SOAP 管理によるオブジェクトのサポートは中止されています。
- SOAP 開発は、いくつかのファイルに依存しています。`SUNWjaf`、`SUNWjmail`、`SUNWxsrt`、および `SUNWjaxp` です。Message Queue の version 4.1 では、これらのファイルを利用できるのは JDK version 1.6.0 以降で Message Queue を実行している場合のみです。

再配布可能なファイル

Sun Java System Message Queue 4.1 には、バイナリ形式で使用でき自由に配布可能な次のファイルのセットが含まれています。

`fscontext.jar`
`imq.jar`
`imqjmx.jar`
`imqxm.jar`
`jaas.jar`
`jms.jar`
`libmqcrt.so` (HPUX)
`libmqcrt.so` (UNIX)
`mqcrt1.dll` (Windows)

さらに、`LICENSE` ファイルおよび `COPYRIGHT` ファイルも再配布できます。

障害を持つユーザー向けのアクセシビリティー機能

このメディアの出版後にリリースされたアクセシビリティーを入手する場合は、請求に応じて Sun から提供される 508 条に関する製品評価資料を参照し、使いやすいソリューションの配備にもっとも適したバージョンを調べてください。最新バージョンのアプリケーション
は、`http://sun.com/software/javaenterprisesystem/get.html` にあります。

アクセシビリティーに対する Sun の取り組みについては、`http://sun.com/access` を参照してください。

問題の報告とフィードバックの方法

Sun Java System Message Queue に問題が発生した場合は、次のいずれかの方法で Sun のカスタマサポートにお問い合わせください。

- Sun ソフトウェアサポートサービス http://www.sun.com/service/sunone/software

  このサイトには、ナレッジベース、オンラインサポートセンター、ProductTracker へのリンクと保守プログラムおよびサポートの連絡先電話番号へのリンクがあります。
- 保守契約を結んでいるお客様の場合は、専用ダイヤルをご利用ください。

最善の問題解決のため、サポートに連絡する際には次の情報をご用意ください。

- 問題が発生する状況と、実行処理への影響を含む問題の説明。
- マシンのタイプ、OS のバージョン、および製品のバージョン (問題に関連している可能性のあるパッチやその他のソフトウェアを含む)
- 問題を再現するために用いた詳細な手順。
- すべてのエラーログまたはコアダンプ。

Sun Java System Software Forum

次のサイトでは、Sun Java System Message Queue フォーラムが利用できます。

http://swforum.sun.com/jive/forum.jspa?forumID=24

ご参加を歓迎いたします。

Java Technology Forum

Java Technology Forums には、関連する JMS のフォーラムがあります。

http://forum.java.sun.com

このマニュアルに関するコメント

弊社では、マニュアルの改善に努めており、お客様からのコメントおよびご忠告をお受けしております。

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。このオンラインフォームでは、マニュアルのタイトルと Part No. もご記入ください。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。このマニュアルのタイトルは『Sun Java System Message Queue 4.1 リリースノート』で、Part No. は 820-3191 です。

その他の情報

次のインターネットのサイトで、Sun Java System の情報を参照できます。

- 関連文書

  http://docs.sun.com/prod/java.sys
- プロフェッショナルサービス

  http://www.sun.com/service/sunps/sunone
- ソフトウェア製品およびサービス

  http://www.sun.com/software
- Software Support Services

  http://www.sun.com/service/sunone/software
- サポートおよびナレッジベース

  http://www.sun.com/service/support/software
- Sun サポートおよびトレーニングサービス

  http://training.sun.com
- コンサルティングおよびプロフェッショナルサービス

  http://www.sun.com/service/sunps/sunone
- 開発者向けの情報

  http://developers.sun.com
- Sun 開発者サポートサービス

  http://www.sun.com/developers/support
- ソフトウェアトレーニング

  http://www.sun.com/software/training