Red Hat Enterprise MRG 3

メッセージングインストールおよび設定ガイド

Red Hat Enterprise MRG メッセージングサーバーのインストールおよび設定

Last Updated: 2020-09-01

Red Hat Enterprise MRG 3 メッセージングインストールおよび設定ガイド

Red Hat Enterprise MRG メッセージングサーバーのインストールおよび設定

Red Hat Customer Content Services

法律上の通知

概要

Red Hat Enterprise Messaging Server のインストールおよび設定

目次

MRG 3 の概要

1. MRG MESSAGING 2 と 3 の間の最も重要な違い

以下は、MRG 2 と MRG 3 の最も重要な違いです。

1. ブローカーおよび C++ メッセージングライブラリー(**qpid::messaging**)は、Apache Proton ライブラリー経由で amqp1.0 サポートを提供するようになりました（ただし、トランザクションは amqp1.0 ではまだ利用できないことに注意してください）。
2. クラスタリングは新しい高可用性実装に置き換えられました。
3. キューのしきい値アラートは、レベルによってトリガーされるのではなく、エッジによってトリガーされるようになりました。これにより、アラートのレート制限が改善されます。
4. メモリーをより効率的に使用するために、flow-to *-disk 実装がディスクページキュー* に変更されました。
5. **ring-strict** 制限ポリシーが削除されました。
6. メッセージングジャーナルは、新しい実装（動的に展開する *Linear Store*）に置き換えられました。

関連項目

- 「MRG 2 の機能 - Where Are Jane Now?」
- 「disk-paged キュー」
- 「Queue Threshold Alerts(Edge-triggered)」

バグを報告します。

第1章 MRG メッセージングを迅速にインストール

1.1. メッセージングサーバー

1.1.1. メッセージングサーバー

MRG Messaging は、Apache Qpid プロジェクトをベースとして、エンタープライズレベルのテスト済みのサポート対象のメッセージングサーバーです。MRG Messaging Server は、Apache Qpid ブローカーのエンタープライズグレードバージョンを使用して、メッセージングサービスを提供します。

関連項目

- 「メッセージングブローカーのメモリー要件」

バグを報告します。

1.1.2. メッセージングブローカー

メッセージングブローカー は、メッセージを保管、転送、および配布するサーバーです。Red Hat Enterprise Messaging は Apache Qpid C++ ブローカーを使用します。

バグを報告します。

1.1.3. Red Hat Enterprise Linux 6 への MRG-M 3 メッセージングサーバーのインストール

1. お使いのシステムに RHN Classic 管理 を使用している場合は、システムを Red Hat Enterprise Linux 6 のベースチャンネルにサブスクライブします。
2. また、お使いのインストールおよび要件に関連する利用可能な MRG Messaging ソフトウェアチャネルにサブスクライブします。

   MRG メッセージングソフトウェアチャンネル

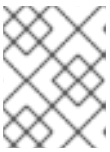
   ベースチャネル

   Additional Services Channels for Red Hat Enterprise Linux 6 / MRG Messaging v.3 (for RHEL-6 Server) チャネルにサブスクライブして、MRG Messaging Platform の完全インストールを有効にします。

   High Availability Channel

   Additional Services Channels for Red Hat Enterprise Linux 6 / RHEL Server High Availability チャネルにサブスクライブして、高可用性インストールを有効にします。

3. 以下のコマンドを使用して、MRG メッセージングサーバーおよびクライアントをインストールします。



注記

メッセージングクライアントサポートのみが必要な場合は、直接手順 4 に移動します。

MRG メッセージングサーバーおよびクライアント

root で以下の **yum** コマンドで「MRG Messaging」グループをインストールします。

```
yum groupinstall "MRG Messaging"
```

高可用性サポート

高可用性サポートが必要な場合は、以下の yum コマンドを使用してパッケージをインストールします（root で）。

```
yum install qpid-cpp-server-ha
```

4. **代替： インストールメッセージングクライアントサポートのみ**
メッセージングクライアントサポートのみが必要な場合は、以下の **yum** コマンドを実行して「Messaging Client Support」グループをインストールします（root で）。

```
yum groupinstall "Messaging Client Support"
```

「 MRG Messaging」グループがすでにインストールされている場合は、このグループをインストールする必要はありません。デフォルトでは含まれます。

注記

Qpid JMS AMQP 0.10 および 1.0 クライアントの両方を実行するには、Java 1.7 を実行する必要があります。システムにインストールされている Java バージョンが 1.7 以降であることを確認します。

バグを報告します。

1.1.4. MRG メッセージング 2 サーバーを MRG メッセージング 3 にアップグレード

MRG Messaging 2 から 3 にアップグレードするには、RHN チャネルの変更およびパッケージのインストールに加えて、複数の設定変更が必要になります。

1. お使いのシステムに RHN Classic 管理 を使用している場合は、システムを Red Hat Enterprise Linux 6 のベースチャンネルにサブスクライブします。
2. 互換性のないコンポーネントを削除します。root で以下のコマンドを実行します。

```
yum erase qpid-cpp-server-cluster sesame cumin cumin-messaging python-wallaby
```

3. MRG v2 チャネルからシステムのサブスクライブを解除します。
4. また、お使いのインストールおよび要件に関連する利用可能な MRG Messaging ソフトウェアチャネルにサブスクライブします。

MRG メッセージングソフトウェアチャンネル

ベースチャネル

Additional Services Channels for Red Hat Enterprise Linux 6 / MRG Messaging v.3 (for RHEL-6 Server) チャネルにサブスクライブして、MRG Messaging Platform の完全インストールを有効にします。

High Availability Channel

Additional Services Channels for Red Hat Enterprise Linux 6 / RHEL Server High Availability チャネルにサブスクライブして、高可用性インストールを有効にします。

5. 以下のコマンドを使用して MRG メッセージングサーバーおよびクライアントを更新します。

注記

メッセージングクライアントサポートのみが必要な場合は、直接ステップに進む必要があります。

MRG メッセージングサーバーおよびクライアント

root で以下の **yum** コマンドで「MRG Messaging」グループを更新します。

```
yum groupinstall "MRG Messaging"
```

高可用性サポート

高可用性サポートが必要な場合は、以下の yum コマンドを使用してパッケージを更新します（root で）。

```
yum install qpid-cpp-server-ha
```

6. メッセージングクライアントサポートのみが必要な場合は、以下の **yum** コマンドを実行して「Messaging Client Support」グループを更新します（root で）。

```
yum groupinstall "Messaging Client Support"
```

「 MRG Messaging」グループを更新している場合は、このグループを更新する必要はありません。デフォルトでは含まれます。

注記

Qpid JMS AMQP 0.10 および 1.0 クライアントの両方を実行するには、Java 1.7 を実行する必要があります。システムにインストールされている Java バージョンが 1.7 以降であることを確認します。

関連項目

- 「MRG 2 の機能 - Where Are Jane Now?」
- 「Application Migration」

バグを報告します。

1.1.5. Linearstore カスタムブローカー EFP パーティション

MRG-M 3.2 では Empty File Pool(EFP)ブローカーパーティションのアップグレードされたディレクトリー構造が導入され、固有の EFP パーティションとそのサイズを指定できます。

この機能により、異なるメディアで EFP を確立でき、キューがパフォーマンス要件に応じて使用する

パーティションを選択できるようになります。たとえば、スループットが高く、レイテンシーが低いキューは、より高価なソリッドステートメディアに確立できるようになりました。一方、低スループットな非クリティカルキューは、通常のローテーションメディアを使用するように転送できるようになりました。

新しいレイアウトでは、古いストアと新しいストアの両方が store ディレクトリーに相互排他的な場所に存在することが可能になります。これにより、必要に応じてアップグレードを元に戻すことができます。

関連項目

- 「永続オプション」
- 「MRG Messaging 3.1 サーバーを MRG Messaging 3.2 にアップグレード」

バグを報告します。

1.1.6. MRG Messaging 3.1 サーバーを MRG Messaging 3.2 にアップグレード

MRG-M 3.1 から 3.2 にアップグレードする際に、「Linearstore カスタムブローカー EFP パーティション」 特定の要件に記載されている EFP が変更されたためです。

手順1.1 MRG Messaging 3.1 から 3.2 へのアップグレード方法

1. 必要なソフトウェアチャンネルがすべてに正しくサブスクライブされていることを確認します。 「Red Hat Enterprise Linux 6 への MRG-M 3 メッセージングサーバーのインストール」
2. 以下のいずれかを実行してサーバーを停止します。
   a. コマンドラインから起動 **Ctrl**+**C** した場合は、次のコマンドを実行してサーバーを正しくシャットダウンします。
   b. **service qpidd stop** を実行して、サービスを正しく停止します。
3. **sudo yum update qpid-cpp-server-ha** を実行して、最新のパッケージにアップグレードします。
4. 

   重要

   カスタム EFP パーティションを設定する場合は、の手順を完了し 手順 1.2「Linearstore EFP を新規パーティション構造に手動でアップグレードする方法」 てください。

   実行 **qpidd** または要件に **service qpidd start** 応じてサーバーを再起動します。

アップグレード前に MRG-M ブローカーを正常にシャットダウンできない場合は、Linearstore EFP ファイルを手作業で新しい構造にアップグレードし、正しくリンクする必要があります。

Linearstore パーティションの変更の一環として、新しいディレクトリー構造が存在します。

ディレクトリーの変更

qls/dat
　これで、このディレクトリーはになり **qls/dat2**ます。ディレクトリー名以外に変更はありません。

qls/tpl

これで、このディレクトリーはになり **qls/tpl2**ます。

このディレクトリーに以前に保存されたジャーナルファイルは、ジャーナルファイルへのリンクになりました。実際のファイルが EFP の **qls/p*NNN*/efp/*[size]*k/in_use** ディレクトリーに置かれるようになりました。これにより、EFP が存在するパーティション内にファイルを含めることができます。

qls/jrnl

このディレクトリーには *[queue-name]* ディレクトリーが **qls/jrnl2**含まれるようになりました。

以前に保存した *[queue-name]* ディレクトリー **qls/jrnl** は、ジャーナルディレクトリーへのリンクになりました。実際のディレクトリーが EFP の **qls/p*NNN*/efp/*[size]*k/in_use** ディレクトリーに置かれるようになりました。これにより、EFP が存在するパーティション内にディレクトリーを含めることができます。

qls/p*NNN*/efp/*[size]*k

このタイプのディレクトリーには、空のファイルと、**/in_use** および **/returned** サブディレクトリーが含まれるようになりました。

p*NNN* --efp-partition パラメーターを使用してコマンドラインに設定されたブローカーパーティション ID に関連します。

***[size]*k** は、**--efp-file-size** パラメーターを使用してコマンドラインに設定されたブローカーパーティションの MiB 単位のサイズです。

ライブアップグレード時にデータの整合性を確保するために（ブローカーをシャットダウンできない）、新しいディレクトリー構造は以前のストアを回復しません。ストアコンテンツのバックアップを作成するために予防策を作成したら、ストアのコンテンツを手動でアップグレードする必要があります。

注記

必要に応じてクリーンなストアおよび再作成キューから開始することが推奨されます。以下の場合にのみアップグレードを実行します。

- 再作成できないキューがある。
- アップグレード前に期限切れにできないメッセージデータがあります。

例1.1 古いディレクトリー構造

```
qls
├── dat (contains Berkeley DB database files)
├── p001
│   └── efp
│       └── 2028k (contains empty/returned journal files)
├── jrnl
│   ├── queue_1 (contains in-use journal files belonging to queue_1)
│   ├── queue_2 (contains in-use journal files belonging to queue_2)
│   ├── queue_3 (contains in-use journal files belonging to queue_3)
│   ...
└── tpl (contains in-use journal files belonging to the TPL)
```

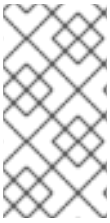
_

考えられる変化

- 任意の数の EFP ファイルサイズを使用でき、デフォルトの 2048k 以外の多くのディレクトリーが存在する可能性があります。
- パーティションディレクトリーは複数ありますが、古い Linearstore では EFP ファイルに個別のディレクトリーを提供する以外に便利な機能は実行されません。これらのディレクトリーには名前を付ける必要 **p*NNN*** があります。*NNN* は 3 桁の数字です。パーティション番号は連続する必要はありません。

例1.2 新しいディレクトリー構造

```
qls
   ├── dat2 (contains Berkeley DB database files)
   ├── p001
   │   └── efp
   │       └── 2028k (contains empty/returned journal files)
   │           ├── in_use (contains in-use journal files)
   │           └── returned (contains files recently returned from being in-use, but not yet processed
before being returned to the 2048k directory)
   │
   ├── jrnl2
   │   ├── queue_1 (contains in-use journal files belonging to queue_1)
   │   ├── queue_2 (contains in-use journal files belonging to queue_2)
   │   ├── queue_3 (contains in-use journal files belonging to queue_3)
   │   ...
   └── tpl2 (contains in-use journal files belonging to the TPL)
```



注記

データベースとジャーナルディレクトリーは相互排他的です。アップグレードが成功するまで、古い構造と journals/files が新しい構造とともに残しておくことが推奨されます。また、アップグレードが以前のバージョンにロールバックされる場合も、ストアは古いディレクトリー構造の使用を継続して動作します。

手順1.2 Linearstore EFP を新規パーティション構造に手動でアップグレードする方法

1. 新しいディレクトリーを作成し **qls/dat2** ます。

```
# mkdir dat2
```

2. **Berkeley DB** データベースの内容を新しい **qls/dat2** ディレクトリー **qls/dat** にコピーします。

```
# cp dat/* dat2/
```

3. の EFP ディレクトリーごとに **qls/p*NNN*/efp/*[size]*k**、2 つのサブディレクトリーを追加します。

    1. *in_use*

```
# mkdir p001/efp/2048k/in_use
```

2. *返済み*

```
# mkdir p001/efp/2048k/returned
```

デフォルトでは、パーティションは1つだけです。また **qls/p001**、EFP サイズは1つだけです **2048k**。

4. **jrnl2** ディレクトリーを作成します。

```
# mkdir jrnl2
```

古い **jrnl** ディレクトリー（それぞれ既存キューに名前が付けられている）に、新しいディレクトリーに同じ名前のディレクトリーを作成し **jrnl2** ます。

```
# mkdir jrnl2/[queue-name-1]
# mkdir jrnl2/[queue-name-2]
...
```

以下のコマンドを使用すると、ディレクトリーに存在する **jrnl2** ディレクトリーを一覧表示できます。

```
# dir jrnl
```

5. 各ジャーナルファイルは、正しい efp *サイズ* **in_use** ディレクトリーで、最初に正しいパーティションディレクトリーのディレクトリーにコピーする必要があります。次に、新しい **jrnl2/*[queue-name]*** ディレクトリーにあるこのジャーナルファイルへのリンクを作成する必要があります。

各ジャーナルファイルに 2 つの情報が必要です。

1. 発信元となるパーティション。
2. そのパーティション内のサイズ。

デフォルト設定は、1つのパーティション番号（ディレクトリー内 **qls/p001**）で、1つの EFP サイズ **2048k** （各ジャーナルファイルの概算サイズ）です。古いディレクトリー構造にこれらのデフォルトのみがある場合は、以下の手順を実行します。

a. のキューごとに **qls/jrnl**、ジャーナルファイルが存在する点に注意してください。移動すると、どのジャーナルファイルが他のジャーナルファイルとして他のジャーナルファイルとして存在するかを区別することは容易ではありません。

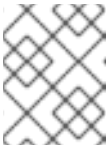
```
# ls -la jrnl/queue-name/*
```

b. すべてのジャーナルファイルを古いキューディレクトリーからパーティションのディレクトリーにコピーし **2048k in_use** ます。

```
# cp jrnl/queue-name/* p001/efp/2048k/in_use/
```

c. 最後に、上記の手順 3 で作成した新しいキューディレクトリーにこれらのファイルへのシンボリックリンクを作成します。このステップでは、上記のステップ b. でコピーしたファイルの名前が必要です。

```
# ln -s /abs_path_to/qls/p001/efp/2048k/in_use/journal_1_file_name.jrnl jrnl2/queue-name/
# ln -s /abs_path_to/qls/p001/efp/2048k/in_use/journal_2_file_name.jrnl jrnl2/queue-name/
...
```



注記

シンボリックリンクを作成する場合は、ソースファイルへの絶対パスを使用します。

d. キューの各ジャーナルファイルに対して直前の手順を繰り返します。

複数のパーティションが存在する場合は、どのジャーナルファイルがどのジャーナルファイルがどのパーティションに属するかを把握することが重要です。

各ジャーナルファイルのファイルヘッダーの 16 進ダンプを検査して、この情報を取得できます。オフセット 26(0x1a)の 2 バイトの値に注意してください。

```
# hexdump -Cn 4096 path/to/uuid.jrnl
00000000  51 4c 53 66 02 00 00 00  1c 62 0c f1 e2 4c 42 0d  |QLSf.....b...LB.|
00000010  5a 6b 00 00 00 00 00 00  01 00 01 00 00 00 00 00  |Zk..............|
00000020  00 02 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00000030  34 63 b9 54 00 00 00 00  8e 61 ef 2c 00 00 00 00  |4c.T.....a.,....|
00000040  2f 00 00 00 00 00 00 00  08 00 54 70 6c 53 74 6f  |/.........TplSto|
00000050  72 65 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |re..............|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

ディレクトリーのサイズディレクトリーが複数ある場合には **p*NNN*/efp/**、上記の手順 b. でコピーされるファイルのサイズを考慮し、それらが正しい efp サイズの **in_use** ディレクトリーにコピーされるようにする必要があります。

例1.3 パーティションで使用されている複数のサイズ

```
qls
└── jrnl
    ├── queue-1
    │   └──jrnl1_file.jrnl  (size 2101248)
    └── queue-2
        └──jrnl2_file.jrnl  (size 4198400)
```

これら両方のファイルがパーティションに属すると仮定して **p*NNN***、新しい **p*NNN*/efp/2048k/** ディレクトリーにコピー **jrnl1_file.jrnl** され、新しい pNN/efp/4096k/ ディレクトリーに **jrnl2_file.jrnl** コピーされます。

6. トランザクション準備済みリスト(TPL)は、トランザクションの準備およびコミット/中止の境界を記録する特別なキューです。新しいストアでは、という新しいディレクトリーに置かれ **tpl2**ます。

a. **tpl2** ディレクトリーを作成します。

```
# mkdir tpl2
```

b. ジャーナルファイルがディレクトリーにあり、シンボリックリンクが新しい **tpl tpl2** ディレクトリーに作成されている場合を除き、上記の手順 4 で説明するプロセスを繰り返します。

i. 現在のジャーナルファイルを一覧表示します。

```
# ls -la tpl
```

ii. ジャーナルファイルを **tpl** ディレクトリーから、上記の手順 4 の一部としてコピーした他のファイル **p*NNN*/efp/*[size]*k/in_use** と共に正しいディレクトリーにコピーします。

```
# cp tpl/* p001/efp/2048k/in_use/
```

iii. 新しい **tpl2** ディレクトリーにこれらのファイルへのシンボリックリンクを作成します。

```
# ln -s abs_path_to/qls/p001/efp/2048k/in_use/efp_journal_1_file_name.jrnl tpl2/
```

iv. コピーした各ファイルに対して上記の手順を繰り返し **tpl**ます。

複数のパーティションまたは複数の EFP サイズが使用されている場合は、上記の **note** 手順 4 を参照し、必要に応じて説明するように適切な調整を行います。

7. **qls** ディレクトリーの適切な所有権を復元します。

```
# chown -R qpidd:qpidd /absolute_path_to/qls
```

8. qls ディレクトリーの SELinux コンテキストの復元

```
# restorecon -FvvR /abs_path_to/qls
```

これでアップグレードが完了し、ブローカーを開始できるようになりました。これを確認するには、ブローカーが昇格したロギングで起動されることが推奨されます。これにより、Linearstore リカバリープロセスに関する追加のメッセージが出力されます。

ブローカーがコマンドラインで起動された場合は、**--log-enable info+** 最初の再起動にオプションを使用します。それ以外の場合は、ブローカーをサービスとして起動する前に、ブローカー設定ファイルを変更して、このログレベルを使用します。

すべてのキューが正常に復元され、予想されるメッセージがすべて復元されたことが確立されると、ブローカーが停止し、ログレベルが以前の設定またはデフォルト設定に返されます。

関連項目

- 「永続オプション」

バグを報告します。

1.1.7. メッセージブローカートラフィックのファイアウォールの設定

メッセージブローカーをインストールして設定する前に、使用するポートで着信接続を許可する必要があります。メッセージブローカー(AMQP)トラフィックのデフォルトポートはです **5672**。

これを許可するには、ファイアウォールを変更して、必要なポートでネットワークトラフィックを許可する必要があります。すべての手順は、**root** ユーザーとしてサーバーにログインしている間に実行する必要があります。

手順1.3 メッセージブローカートラフィックのファイアウォールの設定

1. テキストエディターで **/etc/sysconfig/iptables** ファイルを開きます。
2. ポートの着信接続を許可する **INPUT** ルールをファイル **5672** に追加します。新しいルールは、**REJECT** トラフィックの **INPUT** ルールの前に追加する必要があります。

```
-A INPUT -p tcp -m tcp --dport 5672  -j ACCEPT
```

3. 変更内容を **/etc/sysconfig/iptables** ファイルに保存します。
4. **iptables** サービスを再起動して、ファイアウォールの変更を有効にします。

```
# service iptables restart
```

これで、ポート上の MariaDB データベースサービスへの着信接続を許可するようにファイアウォールが設定され **5672**ます。

バグを報告します。

1.2. メモリー要件および制限事項

1.2.1. メモリー割り当て制限（32 ビット）

32 ビットオペレーティングシステムで実行しているブローカーには、3GB のメモリー割り当て制限があります。このようなシステムで 3GB 以上の容量を持つキューを作成できますが、キューがキューに 3GB に達すると、キューに多くのメッセージを送信しようとすると、メモリーの割り当てに失敗します。

バグを報告します。

1.2.2. ジャーナルでのトランザクションの影響

トランザクションが使用されると、トランザクション ID(XID)が各レコードに追加されます。XID のサイズは、ローカルトランザクションの 24 バイトです。分散トランザクションの場合、ユーザーはトランザクションモニターから取得される XID を提供し、任意のサイズになります。トランザクションでは、メッセージエンキューレコードに加えて、各メッセージデキュー、トランザクションアボートまたはコミットごとにジャーナルレコードが維持されます。

バグを報告します。

1.2.3. メッセージングブローカーのメモリー要件

Broker で必要となるメモリー量は、同時に処理するメッセージの数とサイズです。

メッセージのサイズは、メッセージヘッダーのサイズとメッセージボディーのサイズの組み合わせです。

メッセージサイズの計算
注記： トランザクションはメッセージのサイズを増やします。

手順1.4 推定メッセージサイズ

この方法では、理論的にメッセージサイズを計算することができます。

1. デフォルトのメッセージヘッダーコンテンツ（Java タイムスタンプや message-id など）: 55 バイト
2. ルーティングキー（例： "testQ" = 5 バイト）
3. Java クライアントは以下を追加します。
   - **content-type** （「text/plain」の場合は 10 バイト）
   - **user-id** （認証のために SASL に渡されるユーザー名、文字列サイズに等しいバイト数）
4. アプリケーションヘッダー：
   - アプリケーションヘッダーのオーバーヘッド - 8 バイト
   - テキストヘッダープロパティーの場合： **property_name_size + property_value_size + 4** bytes

たとえば、以下の **spout** ようなメッセージを送信します。

```
./run_example.sh org.apache.qpid.example.Spout -c=1 -b="guest:guest@localhost:5672" -
P=property1=value1 -P=property2=value2 "testQ; {create:always}" "123456789"
```

AMQP メッセージをボディーサイズ 9 で送信し、メッセージヘッダーのサイズは以下のようになります。

- デフォルトサイズの 55 バイト
- ルーティングキー「testQ」の 5 バイト
- content-type "text/plain" の 10 バイト
- user-id "guest" の 5 バイト
- アプリケーションヘッダーを使用するための 8 バイト
- 最初のプロパティーの 9+6+4 バイト
- 2 番目のプロパティーの場合は 9+6+4 バイト
- ヘッダーの合計サイズ： 121 バイト
- メッセージの合計サイズ： 130 バイト

手順1.5 ログからのメッセージサイズの決定

この方法では、稼働中のブローカーからメッセージサイズを測定できます。

1. 以下を追加してトレースロギングを有効に **/etc/qpid/qpidd.conf**します。

```
log-enable=trace+:qpid::SessionState::receiverRecord
log-enable=info+
log-to-file=/path/to/file.log
```

このロギングは大量のディスク領域を消費するため、テストの実行後にこの行を削除して、オフにする必要があります。

2. (re)Broker を起動します。

3. qpid クライアントからサンプルメッセージパターンを送信します。このサンプルメッセージパターンは、メッセージヘッダーとボディーの平均サイズが実際のユースケースに一致するように、通常の使用状況に対応する必要があります。

4. メッセージパターンの送信後、ログファイル内のログレコード用に grep を実行します。

```
2012-10-16 08:56:20 trace guest@QPID.2fa0df51-6131-463e-90cc-45895bea072c: recv cmd 2: header (121 bytes); properties={{MessageProperties: content-length=9; message-id=d096f253-56b9-33df-9673-61c55dcba4ae; content-type=text/plain; user-id=guest; application-headers={property1:V2:6:str16(value1),property2:V2:6:str16(value2)}; } {DeliveryProperties: priority=4; delivery-mode=2; timestamp=1350370580363; exchange=; routing-key=testQ; }}
```

このサンプルのログエントリーには、ヘッダーサイズ（121 バイト）とメッセージボディーサイズ（この場合は 9 バイト）の両方が content-length=9 として含まれます。

Broker のメッセージメモリーの使用状況
ブローカーでは、メモリーを使用してメッセージを保持します。さらに、以下が含まれます。

- メッセージヘッダーの 2 番目のインスタンスが保持されます。1 つは raw バイトとして、もう 1 つはマップとして保存されます。
- Message オブジェクトは 600 バイトを使用します。
- 各メッセージは、3 つのミューテックスおよびモニターによって保護されます。これには 208 バイトが必要です。

そのため、メッセージのメモリー使用量の合計は以下のようになります。

message_body +(message_header * 2)+ 808 バイト

メッセージボディーおよびヘッダーサイズの平均値を使用して、この図をキューの深さの合計で乗じると、ブローカーの飽和したメモリー負荷の数値が得られるようになります。

注記： 交換用のインメモリー最適化は、メッセージが交換に配信される際に、サブスクライブされたすべてのキューにメッセージのコピー 1 つを使用します。つまり、複数のキューへの交換のあるブローカーは、通常の操作では大幅に少ないメモリーを使用します。ただし、ブローカーが再起動され、キューからキューに格納されたメッセージをディスクに読み込むと、**キュー** に読み込まれ、メモリーのフル影響が発生します。

バグを報告します。

1.3. MRG 2 の機能 - WHERE ARE JANE NOW?

1.3.1. 設定ファイルの変更

MRG 2 設定ファイルがにあり **/etc/qpidd.conf**ます。MRG 3 ブローカーの設定ファイルはにあり **/etc/qpid/qpidd.conf**ます。

MRG 2 の設定は、手動で新しい MRG 3 設定ファイルにマージできます。

バグを報告します。

1.3.2. クラスター設定の変更

クラスタリングが変更され、MRG 2 クラスタリング設定は設定ファイルから削除する必要があります。以下のパラメーターを削除する必要があります。

- **cluster-name**
- **cluster-mechanism**
- **cluster-url**
- **cluster-username**
- **cluster-password**
- **cluster-cman**
- **cluster-size**
- **cluster-clock-interval**
- **cluster-read-max**

関連項目

- 「MRG 3 でのクラスタリングの変更」

バグを報告します。

1.3.3. フローからディスクへの置き換え

flow-to-disk はページングキューに置き換えられたため、設定ファイルから参照を削除 **flow_to_disk** する必要があります。

関連項目

- 「disk-paged キュー」

バグを報告します。

1.3.4. リニアストア

MRG 3 では、永続メッセージに新しい *リニアストア* が導入されました。Linear Store を使用すると、永続ジャーナルは動的に拡張できます。つまり、永続キューは削除および再作成せずに拡張できます。

MRG 2 ジャーナルからアップグレードする方法はありません。MRG 3 をクリーンストアから変更し、ジャーナルを使用するすべてのキューを再宣言する必要があります。

関連項目

- 「メッセージジャーナル」

バグを報告します。

1.3.5. アドレス文字列と接続オプション

新しい AMQP 1.0 qpid-jms クライアントは、古い AMQP 0.10 java クライアントによって使用される MRG 固有のアドレス文字列をサポートしません。

新しいクライアントでは、新しいクライアントに指定されたキュー名および「topic name」文字列は、消費するキューまたは Topic の名前のみを表します。

エンティティーの作成および削除に対するサブジェクトや操作などの古いクライアント構文の他の概念はサポートされません。

バグを報告します。

1.4. APPLICATION MIGRATION

1.4.1. MRG 3 での API サポート

qpid::messaging API のみ **qpid::types** が MRG 3 でサポートされます。

他の **qpid::** namespace API を使用するアプリケーション（例： **qpid::client**[1] および **qpid::types::Variant::fromString**[2]）をクリックして、サポートされている API のみを使用するように書き換える必要があります。

バグを報告します。

1.4.2. qpid::messaging Message::get/setContentObject()

構造化された AMQP 1.0 メッセージには、さまざまな方法でエンコードされたメッセージのボディーがあります。

Ruby および Python API は、構造化 AMQP 1.0 メッセージのボディーをデコードしません。AMQP 1.0 タイプとして送信されたメッセージはこれらのライブラリーによって受信できますが、ボディーはデコードされません。Ruby API および Python API を使用するアプリケーションは、ボディー自体をデコードする必要があります。

C++ API および C# API には新しいメソッドが追加され、構造化された AMQP 1.0 メッセージのセマンティックコンテンツ **Message::getContentObject() Message::setContentObject()** にアクセスします。これらのメソッドにより、メッセージのボディーにバリアントとしてアクセスまたは操作することができます。これらの方法を使用すると、プロトコルバージョンの両方で機能し、map-、list-、text-、または binary- メッセージの両方で機能するときに、最も広く適用可能なコードが生成されます。

content オブジェクトはバリアントで、型を判別できるバリアントで、コンテンツを自動的にデコードできるようにします。

以下の C++ 例は、新しいメソッドを示しています。

```
bool Formatter::isMapMsg(qpid::messaging::Message& msg) {
```

```
  return(msg.getContentObject().getType() == qpid::types::VAR_MAP);
}

bool Formatter::isListMsg(qpid::messaging::Message& msg) {
  return(msg.getContentObject().getType() == qpid::types::VAR_LIST);
}

qpid::types::Variant::Map Formatter::getMsgAsMap(qpid::messaging::Message& msg) {
  qpid::types::Variant::Map intMap;
  intMap = msg.getContentObject().asMap();
  return(intMap);
}

qpid::types::Variant::List Formatter::getMsgAsList(qpid::messaging::Message& msg) {
  qpid::types::Variant::List intList;
  intList = msg.getContentObject().asList();
  return(intList);
}
```

Message::getContent() また、コンテンツの raw バイトを **Message::setContent()** 引き続き参照します。API の **encode() decode()** およびメソッドは AMQP 0-10 形式の map- および list- メッセージをデコードし続けます。

バグを報告します。

1.4.3. AMQP 1.0 のあいまいなアドレス

クライアントが AMQP 1.0 であいまいなアドレスを使用する場合： キューと同じ名前の交換の両方が存在し、クライアントがノードタイプを明示的に指定しない場合 - **キュー** はデフォルトで使用されます。例外や警告は出力されず、クライアントは曖昧さのいずれの方法でも通知されません。ノードがキューと交換の両方に一致し、特定のタイプが指定されていない場合、警告はブローカーによってログに記録されます。

これとは対照的に AMQP 0-10 クライアントを使用する場合、以下の例外が「**Ambiguous address, please specify queue or topic as node type**」と報告されます。

バグを報告します。

---

[1] https://bugzilla.redhat.com/show_bug.cgi?id=995039

[2] https://bugzilla.redhat.com/show_bug.cgi?id=1141230

第2章 メッセージングブローカーの起動

2.1. コマンドラインとサービスとしての BROKER の起動

サービスとして起動すると、ブローカーは設定ファイルから起動オプションを読み取ります。コマンドラインから起動すると、ブローカーは起動オプションを設定ファイルまたはコマンドライン引数から読み取ることができます。

Broker をサービスとして起動すると、実稼働サーバーで便利です。Broker は、サーバーが再起動すると常に自動的に起動するように設定できます。開発の用途では、通常、異なる設定で Broker を起動して再起動すると、コマンドラインから開始する際に便利です。

バグを報告します。

2.2. コマンドラインでのブローカーの実行

2.2.1. コマンドラインでブローカーを起動します。

Broker の起動

1. デフォルトでは、ブローカーはにインストールされ /**usr**/**sbin**/ます。パスにない場合は、ブローカーを開始するためにパス全体を入力する必要があります。

   ```
   /usr/sbin/qpidd -t
   ```

   ブローカーの起動時に以下のような出力が表示されます。

   ```
   [date] [time] info Loaded Module: libbdbstore.so.0
   [date] [time] info Locked data directory: /var/lib/qpidd
   [date] [time] info Management enabled
   [date] [time] info Listening on port 5672
   ```

   -t or **--trace** オプションは、デバッグトレースを有効にし、ターミナルにメッセージを出力します。

   注記： ロックされたデータディレクトリー **/var/lib/qpidd** は、デフォルトで有効になっている永続性に使用されます。

バグを報告します。

2.2.2. コマンドラインで起動したブローカーを停止します。

1. ブローカーを停止するには、 **CTRL**+ **C** シェルプロンプトでを入力します。

   ```
   [date] [time] notice Shutting down.
   [date] [time] info Unlocked data directory: /var/lib/qpidd
   ```

ライブプレビュー。

バグを報告します。

2.3. BROKER をサービスとして実行

2.3.1. Broker をサービスとして実行

- Red Hat Enterprise Messaging は通常、実稼働環境のシナリオではサービスとして実行されます。root 権限でブローカーをサービスとして起動するには、以下のコマンドを実行します。

```
service qpidd start
```

メッセージブローカーは以下のメッセージで開始します。

```
Starting Qpid AMQP daemon:                 [  OK  ]
```

バグを報告します。

2.3.2. Broker サービスの停止

- サービスとして実行中のブローカーの状態を確認するには、**service status** コマンドを使用します。コマンドでブローカーを停止し **service stop**ます。

```
# service qpidd status
qpidd (pid PID) is running...

# service qpidd stop
Stopping Qpid AMQP daemon:                 [  OK  ]
```

バグを報告します。

2.3.3. サーバーの起動時に自動的に起動するように Broker サービスを設定します。

実稼働サーバーでは、通常、マシンを再起動するとメッセージブローカーが自動的に起動します。これには、**qpidd** サービスを有効にする必要があります。

- root で以下のコマンドを実行します。

```
chkconfig qpidd on
```

サーバーの起動時にメッセージブローカー qpidd が自動的に起動するようになりました。

バグを報告します。

2.4. 1 台のマシンでの複数のブローカーの実行

2.4.1. 複数のブローカーの実行

開発環境では、テストおよびプロトタイプ化のために、同じマシンで複数のブローカーを実行することができます。

バグを報告します。

2.4.2. 複数のブローカーの起動

1 台のマシンで複数のブローカーを実行するには、各ブローカーが異なるポートで実行し、ジャーナルに異なるディレクトリーを使用する必要があります。

1. 利用可能なポート（例： *5555* および *5556* ）を選択します。
2. **--data-dir** コマンドを使用して、新しいブローカーを起動し、それぞれに新しいデータディレクトリーを指定します。

```
$ qpidd -p 5555 --data-dir /tmp/qpid/store/1
```

```
$ qpidd -p 5556 --data-dir /tmp/qpid/store/2
```

バグを報告します。

第3章 付与(BROKER)オプション

3.1. コマンドラインでブローカーオプションの設定

単一インスタンスのオプションを設定するには、ブローカーの起動時にオプションをコマンドラインに追加します。

- この例では、コマンドラインオプションを使用 **-t** して、デバッグトレースでブローカーを起動します。

```
$ /usr/sbin/qpidd -t
```

コマンドラインからブローカーが呼び出されるたびにコマンドラインオプションを指定する必要があります。

バグを報告します。

3.2. 設定ファイルでのブローカーオプションの設定

Broker をサービスとして実行する際に Broker オプションを設定するには、コマンドラインから Broker の開始時に使用できるオプションのセットを作成しますが、コマンドラインで毎回指定しなくても設定ファイルを使用します。

1. root ユーザーになり、テキストエディターで **/etc/qpid/qpidd.conf** ファイルを開きます。
2. この例では、設定ファイルを使用してデバッグトレースを有効にします。変更は次回からブローカーを起動し、後続のセッションで使用されます。

```
# Configuration file for qpidd
trace=1
```

3. ブローカーをサービスとして実行している場合は、サービスを再起動して設定オプションを再読み込みする必要があります。

```
# service qpidd restart
Stopping qpidd daemon:            [  OK  ]
Starting qpidd daemon:            [  OK  ]
```

4. コマンドラインからブローカーを実行している場合は、設定ファイルを使用するコマンドラインオプションなしでブローカーを起動します。

```
# /usr/sbin/qpidd
[date] [time] info Locked data directory: /var/lib/qpidd
[date] [time] info Management enabled
[date] [time] info Listening on port 5672
```

バグを報告します。

3.3. BROKER オプション

3.3.1. Broker をデーモンとして実行するためのオプション

変更

- MRG 3 の新機能

ブローカーをデーモンとして実行するためのオプション	
-d	バックグラウンドでデーモンとして実行します。ブローカーからのログメッセージはデフォルトで **syslog** (**/var/log/messages**)に送信されます。
-q	現在実行中のブローカーをシャットダウンします。
-c	デーモンがすでに実行されているかどうかを確認します。実行中であれば、プロセス ID を返します。
--wait=<*seconds*>	初期化 およびシャットダウン 時に <*seconds*> 秒待ちます。この時間内に、デーモンが正常に初期化 またはシャットダウン を完了しなかった場合は、エラーが返されます。シャットダウン時に、デーモンはブローカーがシャットダウンしてからこの期間待機した後、成功または失敗を報告します。このオプションはオプションとともに使用する必要があり、この **-d** オプションは無視されます。

バグを報告します。

3.3.2. 一般的なブローカーオプション

ブローカーには、一般的なコマンドラインオプションが多数あります。に記載されている追加の高可用性オプションがあり 「ブローカー HA オプション」 ます。

一般的なブローカーコマンドラインオプション

-h

ヘルプメッセージを表示します。

--interface <*ipaddr*>

指定したネットワークインターフェースをリッスンします。複数のネットワークインターフェースに複数回使用できます。このオプションは、IPv4 アドレスおよび IPv6 アドレスをサポートします。明示的なアドレス、またはネットワークアダプターの名前（例： *eth0*、*em1*）を使用できます。ネットワークアダプター名を指定すると、ブローカーはそのアダプターにバインドされたすべてのアドレスにバインドされます。

--link-heartbeat-interval <*seconds*>

フェデレーションされたリンクの発信まで待機する秒数。デフォルトは 120 秒です。

--link-maintenance-interval <*seconds*>

バックアップブローカーがプライマリーへのリンクを確認し、必要な場合は再接続するまで待機する秒数。デフォルト値は 2 です。

-p *<Port_Number>*

ブローカーに指定されたポートを使用するように指示します。デフォルトはポート 5672 です。異なるポート番号を使用して、複数のブローカーを同時に実行できます。

--paging-dir *<directory>*

ディスクページキューに使用するディレクトリー。

--socket-fd *<fd>*

ファイル記述子によって指定された既存のソケットを使用します。複数のソケットに複数回使用できます。これは、たとえばテスト中にブローカーが親プロセスによって開始された場合に便利です。

-t

このオプションは、デバッグにのみ詳細なログメッセージを有効にします。

--tcp-nodelay *on|off*

TCP で ack を無効にします。これにより、特に同期操作のスループットが向上します。デフォルトはに設定 **on** されます。これは、を使用して設定ファイルで設定できます。
QPID_TCP_NODELAY=*on|off*

-v

インストールされているバージョンを表示します。

バグを報告します。

3.3.3. logging

デフォルトでは、ブローカーがコマンドラインで実行されている **stderr** 場合や、ブローカーがサービスとして実行される場合は **syslog** (**/var/log/messages**/)にログ出力が送信されます。

表3.1 ロギングオプション

ロギングのオプション syslog	
-t [--trace]	すべてのロギングを有効にします。
--log-disable *RULE*	選択したレベルおよびコンポーネントのロギングを無効にします（オプトアウト）。*ルール* はフォームです **LEVEL[+]:[:PATTERN]**。レベルは以下のいずれかになります **trace, debug, info, notice, warning, error, critical.**。これにより、デバッグ中にログメッセージを中断できます。これは複数回使用できます。

ロギングのオプション syslog	
--log-enable *RULE (notice+)*	選択したレベルおよびコンポーネントのロギングを有効にします。ルールはフォームです **LEVEL[+]:[:PATTERN]**。レベルは以下のいずれかになります **trace, debug, info, notice, warning, error, critical.**。たとえば、すべての警告、エラー、および重大なメッセージを **--log-enable warning+** ログに記録します。これは、フレーミング namespace からのデバッグメッセージを **--log-enable debug:framing** ログに記録します。これは複数回使用できます。
--log-time yes|no	ログメッセージへの include time in ログメッセージ
--log-level yes|no	ログメッセージに重大度を含める
--log-source	ログメッセージに source file:line を含める
--log-thread yes|no	メッセージにスレッド ID が含まれる
--log-function yes|no	ログメッセージに関数署名を含める
--log-hires-timestamp yes|no (0)	ログメッセージでの hi-resolution タイムスタンプの使用
--log-category yes|no (1)	ログメッセージにカテゴリーを含める
--log-prefix *STRING*	すべてのログメッセージに追加する接頭辞
--log-to-stderr yes|no	ロギング出力の送信先 **stderr**。コマンドラインから実行すると、デフォルトで有効になっています。
--log-to-stdout yes|no	ロギング出力の送信先 **stdout**。
--log-to-file *FILE*	指定されたファイル名にログ出力を送信します。*ファイル*。
--log-to-syslog yes|no	ロギング出力の送信先 **syslog**。サービスとして実行すると、デフォルトで有効になっています。
--syslog-name *NAME*	**syslog** メッセージで使用する名前を指定します。デフォルトはです **qpidd**。
--syslog-facility *LOG_XXX*	**syslog** メッセージで使用するファシリティーを指定します。デフォルトはです **LOG_DAEMON**。

関連項目

- *7章logging*

バグを報告します。

3.3.4. モジュール

表3.2 Broker でモジュールを使用するためのオプション

Broker でモジュールを使用するためのオプション	
--load-module *MODULENAME*	指定されたモジュールをプラグインとして使用します。
--module-dir <DIRECTORY>	別のモジュールディレクトリーを使用します。
--no-module-dir	モジュールディレクトリーを無視します。

モジュールを使用したヘルプ

モジュールのヘルプテキストを表示するには、以下の **--help** コマンドを使用します。

```
# /usr/sbin/qpidd --help
```

バグを報告します。

3.3.5. デフォルトモジュール

以下のモジュールがインストールされ、デフォルトでロードされます。

- XML の交換タイプ
- 永続性
- クラスタリング

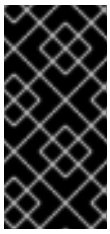
バグを報告します。

3.3.6. 永続オプション

表3.3 ジャーナルオプション

オプション	デフォルト	description
--store-dir *DIR*	詳細は、説明を参照してください。	永続ジャーナルのディレクトリーの場所を保存します。デフォルトは、デーモンとして実行 **/var/lib/qpidd** するか、コマンドラインから実行した **~/.qpidd** 場合です。このオプションを使用すると、デフォルトの場所やで指定された場所を上書きでき **--data-dir**ます。これは、使用する **--no-data-dir** 場合に必要です。

オプション	デフォルト	description
--truncate *yes*|*no*	いいえ	*yes*|*true*|*1* の場合、ストアは切り捨てられます（既存のレコードはすべて破棄）。| *false*|*0* がない場合、既存のストアファイルを復元のために保持します。
--wcache-page-size *N*	32	KiB の書き込みページキャッシュのページサイズ。許可される値 - 2 のべき乗。4:、4、8、16、32...値が小さいほど、スループットを犠牲にしてレイテンシーが短縮されます。
--wcache-num-pages *N*	16	書き込みページキャッシュのページ数。最小値： 4
--tpl-wcache-page-size *N*	4	トランザクションの準備済みリスト書き込みページキャッシュのページサイズ（KiB 単位）。許可される値 - 2 のべき乗。4:、4、8、16、32...値が小さいほど、スループットを犠牲にしてレイテンシーが短縮されます。
--tpl-wcache-num-pages *N*	16	トランザクション準備済みリスト書き込みページキャッシュのページ数。最小値： 4

オプション	デフォルト	description
--efp-partition *N*	1	空のジャーナルファイルを検索するために使用する空のファイルプールブローカーパーティション。このオプションを指定しないと、デフォルトのパーティション値 1 が使用されます。この値はブローカーパーティションに変換さ**p001**れます。 ブローカーのデフォルト以外のパーティションとジャーナルのファイルサイズを選択するには、**qpid-config --efp-partition** および **--efp-file-size** オプションを使用して、別のパーティションとサイズの組み合わせを選択します。例： **qpid-config add queue test-queue-5 --durable --efp-partition 5 --efp-file-size 8192**  **重要** パーティションは、ブローカーを起動する前に存在している必要があります。
--efp-file-size *N*	2048	空のファイルプールブローカーのジャーナルファイルサイズ（KiB 単位）4 KiB の倍数でなければなりません。このオプションを指定しないと、デフォルトのファイルサイズ 2048 KiB が使用されます。オプションを使用するには、のコマンド例を参照してください **--efp-partition**。

関連項目

- 「Linearstore カスタムブローカー EFP パーティション」

バグを報告します。

3.3.7. キューのオプション

表3.4 キューのオプション

オプション	デフォルト	description

オプション	デフォルト	description
--queue-purge-interval	600	ブローカーがすべてのキューを参照し、有効期限の切れた TTL(TTL)ですべてのメッセージをパージする時間（秒単位）を指定します。 このオプションは、メッセージ処理でコンシューマーが一貫してプロデューサーに遅れているキューに使用して、有効期限の切れたメッセージが TTL を経過しないようにします。

バグを報告します。

3.3.8. リソースクォータオプション

--max-connections Broker オプションで制限できる接続の最大数。

表3.5 リソースクォータオプション

オプション	description	デフォルト値
--max-connections *N*	ブローカーへの同時接続の合計。	500
--max-negotiate-time *N*	初期プロトコルネゴシエーションが成功する期間。これにより、クライアントが正しく動作していないことでリソース不足や、接続の完了を妨げる一時的なネットワーク問題の発生を防ぎます。	500
--session-max-unacked *N*	ブローカーは、承認をこの制限まで待たずにセッションにメッセージを送信します（または、セッションの集約リンクが低い場合）。この制限に達すると、ブローカーはクライアントから確認応答を待機した後、さらにメッセージを送信します。	5000（または約 625 KB / セッション）

注記

- **--max-connections** qpid コア制限で、ACL が有効かどうかに関わらず強制されます。
- **--max-connections** Broker ごとに強制されます。*N* ノードのクラスターでは、すべてのブローカーが最大接続を 20 に設定する場合、クラスターの許可されている接続の合計数は *N**20 になります。
- **--session-max-unacked** AMQP 1.0 で多数のセッションが使用される場合、メモリー使用を制御するのに役立ちます。AMQP 1.0 では、承認されていないメッセージ配信にセッションごとのバッファーが割り当てられます。

- **--session-max-unacked** ブローカーにセッション数が多く、メモリーのオーバーヘッドが問題である場合に各セッションのバッファーを小さくすることができます。

ACL ベースのクォータ

ACL ベースのクォータを有効にするには、ACL ファイルを読み込む必要があります。

表3.6 ACL コマンドラインオプション

オプション	description	デフォルト値
--acl-file *FILE (policy.acl)*	読み込むポリシーファイル（data dir から読み込まれる）。	

ACL ファイルが読み込まれると、以下の ACL オプションをコマンドラインで指定して、リソースクォータを強制できます。

表3.7 ACL ベースのリソースクォータオプション

オプション	description	デフォルト値
--connection-limit-per-user *N*	ユーザーごとに許容される最大接続数。0 は無制限を意味します。	0
--connection-limit-per-ip *N*	ホストの IP アドレスごとに許可される最大接続数。0 は無制限を意味します。	0
--max-queues-per-user *N*	個々のユーザーが作成した同時キューの合計	0

注記

- クラスターシステムでは、実際の接続数が、クラスター内のメンバーノードの数よりも **N** 1つ小さくなる可能性があります。たとえば、5 ノードクラスターで、20 個の接続が設定されている場合、制限が行われる前に実際の接続数が 24 に達する可能性があります。
- クラスター接続は、確立されると接続制限に対してチェックされます。空き接続が利用できない場合、クラスター接続は拒否されます。ただし、確立後、クラスター接続は接続を消費しません。
- 使用できる値 **N** は 0..65535 です。
- これらの制限は **クラスター** ごとに適用されます。
- 値がゼロ(0)の場合は、そのオプションの制限チェックが無効になります。
- ユーザーごとの接続は、認証されたユーザー名によって識別されます。
- ノードごとの接続は、管理接続インデックスである **<broker-ip><broker-port>-<client-ip><client-port>** タプルによって識別されます。
    - このスキームホストシステムは、IPv4、**localhost** IPv4、**127.0.0.1** IPv **::1** 6 などの複数の名前で識別でき、名前ごとに別の接続セットが許可されます。

- IP 接続ごとの接続は、接続で提供されたユーザーのクレデンシャルに関係なくカウントされます。個々のユーザーは 20 の接続が許可されますが、クライアントホストに 5 つの接続制限がある場合は、そのシステムから接続できるのは 5 回のみです。

バグを報告します。

3.3.9. セキュリティーオプション

変更

- MRG 3 の新機能。

表3.8 一般的なブローカーオプション

Broker を実行するためのセキュリティーオプション	
--ssl-use-export-policy	NSS エクスポートポリシーの使用
--ssl-cert-password-file <*PATH*>	必須。証明書データベースへのアクセスに使用するパスワードを含むプレーンテキストファイル。
--ssl-cert-name <*NAME*>	使用する証明書の名前。デフォルトはです **localhost.localdomain**。
--ssl-cert-db <*PATH*>	必須。証明書データベースが含まれるディレクトリーへのパス。
--ssl-port <*NUMBER*>	SSL 接続をリッスンするポート。ポートを指定しないと、ポート 5671 が使用されます。選択した SSL ポートが SSL 以外の接続のポートと同じ場合（--ssl-port と --port オプションが同じ場合）、SSL 暗号化接続と暗号化されていない接続の両方を同じポートに確立できます。ただし、この設定では IPv6 はサポートされません。
--ssl-require-client-authentication	SSL ハンドシェイク中に SSL クライアント認証（クライアント証明書の検証）が必要になります。これは SASL 認証の前に発生し、SASL から独立しています。 このオプションは、SSL 接続の EXTERNAL SASL メカニズムを有効にします。クライアントが EXTERNAL メカニズムを選択する場合、クライアントのアイデンティティーは検証された SSL 証明書から取得され、CN を使用してドメインを作成します。たとえば、証明書にプロパティーが含まれる場合 **CN=bob DC=acme DC=com**、クライアントのアイデンティティーはになり **bob@acme.com** ます。 クライアントが別の SASL メカニズムを選択する場合、クライアント証明書からアイデンティティーの取得は、SASL ハンドシェイク時にネゴシエートされたものに置き換えられます。

Broker を実行するためのセキュリティーオプション	
--ssl-sasl-no-dict	ディクショナリー攻撃によって攻撃される可能性のある SASL メカニズムを許可しないでください。これにより、EXTERNAL の代わりに弱いメカニズムが選択されないようにします。これは、辞書攻撃による影響を受けません。
--require-encryption	これにより、qpidd は暗号化された接続のみを受け入れるようになります。これは、SSL-port 上で EXTERNAL SASL を使用するクライアント、または TCP ポートに GSSAPI を使用するクライアントのみを意味します。
--listen-disable *PROTOCOL*	指定したプロトコルで接続を無効にします。たとえば、TCP 上の接続を **--listen-disable tcp** 無効にし、ブローカーが SSL-port の接続のみを受け入れるように強制します。

関連項目

- 「Broker での SSL の有効化」

バグを報告します。

3.3.10. トランザクションオプション

表3.9 トランザクションのオプション

オプション	description
--dtx-default-timeout *<seconds>*	デフォルトでは 60 秒です。 DTX トランザクションのジャーナルレコードは、指定した秒数後に削除されます。これは、外部トランザクションマネージャー(TM)が DTX トランザクションを準備し、コミットまたは中止しない場合に発生します。指定した秒数後、エントリーは孤立したと見なされ、消去されます。

バグを報告します。

第4章 QUEUES

4.1. メッセージキュー

メッセージキュー*は*、アプリケーションを消費して、対象のメッセージにサブスクライブするメカニズムです。

キューは、交換からメッセージを受信し、メッセージコンシューマーが消費するまでこれらのメッセージをバッファーします。これらのメッセージコンシューマーはキューを参照したり、キューからメッセージを取得したりできます。メッセージは再配信のキューに戻すことができ、コンシューマーによって拒否されます。

複数のコンシューマーがキューを共有でき、キューは単一のコンシューマーのみに限定される可能性があります。

メッセージプロデューサーは、キューを作成して交換に使用できるようにするか、または交換に送信してコンシューマーに残してキューを作成し、交換して対象のメッセージを受信することができます。

一時的なプライベートメッセージキューを作成して、応答チャネルとして使用できます。メッセージキューは、アプリケーションが切断されたときにブローカーによって削除されるように設定できます。メッセージをグループ化して、新たにメッセージのコピーでキュー内のメッセージを更新したり、特定のメッセージの優先順位を付けるように設定できます。

メッセージキューを管理する別の方法は、特にメッセージの Time To Live(TTL)の領域で使用され **--queue-purge-interval**ます。これは qpid-config オプションではありませんが、メッセージ TTL を設定でき、パージの試行に成功すると、その後メッセージが削除されます。

このブローカーオプション 「キューのオプション」 の詳細は、を参照してください。

バグを報告します。

4.2. QPID-CONFIG を使用したキューの作成と設定

qpid-config コマンドラインツールは、キューの作成および設定に使用できます。

--helpスイッチを指定してコマンドを実行して、のコマンド参照をすべて **qpid-config** 利用できます。

```
qpid-config --help
```

サーバーが指定されていない場合は、現在のマシンのメッセージブローカーに対して **qpid-config** 実行されます。別のマシンのメッセージブローカーと対話するには、**-a** または **--broker-addr** スイッチを使用します。例：

```
qpid-config -a server2.testing.domain.com
```

broker address オプションの引数は、ユーザー名、パスワード、およびポートを指定することもできます。

```
qpid-config -a user1/secretpassword@server2.testing.domain.com:5772
```

キューを作成するには、**qpid-config add queue** コマンドを使用します。このコマンドは、新しいキューの名前を引数、および [任意] キューオプションとして取ります。

簡単な例として、ローカルマシン **testqueue1** 上で実行されているメッセージブローカーに呼び出されるキューを作成します。

```
qpid-config add queue testqueue1
```

以下でキューを作成する際に指定できるさまざまなオプションを以下に示し **qpid-config**ます。

表4.1 qpid-config のキューの追加オプション

qpid-config の追加キューのオプション	
--alternate-exchange *queue name*	代替の交換の名前。キューが削除されると、このキューの残りのメッセージはすべてこの交換にルーティングされます。キューサブスクライバーによって拒否されたメッセージは、代替の交換にも送信されます。
--durable	新しいキューは永続化されます。このキューに送信された PERSISTENT とマークされている未配信メッセージとともに、サーバーが再起動すると再作成されます。
--file-count *integer*	キューの永続ジャーナルに含まれるファイルの数。最大 64 個までです。64 を超える指定を試みると、ジャーナルファイルが 64 個作成されます。
--file-size *integer*	ページ単位のファイルサイズ(64KiB/page)。
--max-queue-size *integer*	メモリー内キューの最大サイズ（バイト）。32 ビットのシステムキューでは、宣言されたサイズに関係なく、3 GB を超えることはありません。
--max-queue-count *integer*	メモリー内キューの最大サイズ（メッセージ数）。
--limit-policy *[none, reject, ring]*	キューの制限に達したときに実行するアクション。
--flow-stop-size *integer*	キューに格納されたバイト数がこの値を超えると、送信者フロー制御を有効にします。
--flow-resume-size *integer*	キューに格納されたバイト数がこの値を下回ると、送信者フロー制御をオフにします。
--flow-stop-count *integer*	キューに格納されたメッセージの数がこの値を超えると、送信者フロー制御を有効にします。
--flow-resume-count	キューに格納されたメッセージの数がこの値を下回ると、送信者フロー制御をオフにします。
--group-header	メッセージグループを有効にします。グループ識別子を保持するヘッダーの名前を指定します。

qpid-config の追加キューのオプション	
--shared-groups	複数のコンシューマーでメッセージグループを消費できるようにします。
--argument *name=value*	キュー引数に追加するキーと値のペアを指定します。これは、たとえば、自己生成メッセージのループバック配信 **no-local=true** を抑制するために使用できます。

排他的キューは作成されたセッションでのみ使用できるため **qpid-config**、このキューを使用して排他的キューを作成することはできません。

関連項目

- 「qpid-config の使用」
- 「最終値キューの宣言」
- 「ローカルに公開されるメッセージの無視」
- 付録A *交換およびキュー宣言引数*

バグを報告します。

4.3. メモリー割り当て制限（32 ビット）

32 ビットオペレーティングシステムで実行しているブローカーには、3GB のメモリー割り当て制限があります。このようなシステムで 3GB 以上の容量を持つキューを作成できますが、キューがキューに 3GB に達すると、キューに多くのメッセージを送信しようとすると、メモリーの割り当てに失敗します。

バグを報告します。

4.4. 排他的キュー

排他的キューは一度に 1 つのセッションでのみ使用できます。exclusive プロパティーセットでキューが宣言されると、キューが宣言されたセッションが閉じられるまで、そのキューは他のセッションで使用できなくなります。

サーバーが exclusive として宣言されたキューの宣言、バインド、削除、またはサブスクライブリクエストを受信すると、例外が発生し、要求セッションが終了します。

セッションのクローズはすぐに検出されないことに注意してください。クライアントがハートビートを有効にする場合、セッションクローズは保証された時間内に決定されます。特定の API でハートビートを設定する方法の詳細は、クライアント API を参照してください。

バグを報告します。

4.5. ローカルに公開されるメッセージの無視

キューを設定して、キューを所有するセッションと同じ接続を使用してパブリッシュされたすべてのメッセージを破棄できます。これにより、アプリケーションがメッセージの交換にサブスクライブしていることを示すメッセージループバックが抑制されます。

ローカルに公開されているメッセージを無視するようにキューを設定するには、queue 宣言 **no-local** のキーを key:value ペアとして使用します。キーの値は無視されます。キーが存在するだけで十分です。

たとえば、ローカルに公開されるメッセージを破棄するキューを作成するには、**qpid-config**以下を使用します。

```
qpid-config add queue noloopbackqueue1 --argument no-local=true
```

複数の異なるセッションが同じ接続を共有できることに注意してください。ローカルに公開されたメッセージを無視するキューは、キューを宣言した **接続** からのメッセージをすべて無視するため、その接続を使用するすべてのセッションはこのコンテキストでローカルになります。

バグを報告します。

4.6. LAST VALUE(LV)キュー

4.6.1. 最後の値キュー

Last Value Queues を使用すると、キュー内のメッセージが更新されたバージョンで上書きされます。Last Value Queue に送信されたメッセージは、ヘッダーキーを使用して、メッセージのバージョンとして自身を特定します。キューで一致するキー値を持つ新しいメッセージは、そのキーを持つ以前のメッセージが破棄されます。その結果、キューを閲覧するメッセージコンシューマーは、メッセージの最新バージョンのみを受け取ることになります。

バグを報告します。

4.6.2. 最終値キューの宣言

最後の値キューは、キューの作成 **qpid.last_value_queue_key** 時に指定されることで作成されます。

たとえば、以下を使用して、キー **stock-symbol** として **stock-ticker** 使用する最後の値キューを作成するには、次のコマンドを **qpid-config**実行します。

```
qpid-config add queue stock-ticker --argument qpid.last_value_queue_key=stock-symbol
```

アプリケーションで同じキューを作成するには、以下を実行します。

python

```
myLastValueQueue = mySession.sender("stock-ticker;{create:always, node:{type:queue, x-declare:{arguments:{'qpid.last_value_queue_key': 'stock-symbol'}}}}")
```

文字列と整数値の両方を最後の値として指定できます。上記で作成したサンプルキューを使用すると、「**RHT**」、"、および他の文字列値、および他の整数値**JAVA**、および他の整数値など **3 15**、Sockt - **symbol** キーの有効な値が含まれます。

バグを報告します。

4.7. メッセージグループ

4.7.1. メッセージグループ

*メッセージグループ*を使用すると、送信者はメッセージのグループすべてが同じコンシューマーによって処理されることを示します。送信者はメッセージのヘッダーを設定し、同じグループの一部として特定し、メッセージをグループ化が有効になっているキューに送信します。

ブローカーは、単一のコンシューマーがグループ内のメッセージに排他的にアクセスでき、グループのメッセージが受信順に配信され、再配信されるようにします。

メッセージグループ化は、Last Value Queue または Priority Queuing と併用できないことに注意してください。

メッセージグループの実装は、その機能要求にアタッチされた 仕様 で説明されています。QPID-3346: 複数のコンシューマー間で厳格なシーケンス消費をサポートするメッセージ グループをサポートします。

バグを報告します。

4.7.2. メッセージグループのコンシューマー要件

グループメッセージの適切な処理は、ブローカーとコンシューマーの両方の責任となります。コンシューマーがグループの一部であるメッセージを取得する場合、ブローカーはそのコンシューマーをそのメッセージグループの所有者にします。そのグループ内のメッセージはすべて、そのグループ **からフェッチしたすべてのメッセージがコンシューマーによって認識されるまで、そのコンシューマーにのみ表示されます。**コンシューマーがグループから取得したメッセージをすべて確認すると、ブローカーはグループの所有権を解放します。

コンシューマーは、グループ内のフェッチされたメッセージをすべて一度に確認する必要があります。メッセージのグループ化の目的は、グループ内のすべてのメッセージが、同じコンシューマーによって処理されるようにすることです。コンシューマーがキューからグループ化されたメッセージを受信し、障害のために切断すると、グループ内の未承認のメッセージが解放され、他のコンシューマーが利用できるようになります。ただし、グループの確認済みのメッセージはキューから削除されているため、ヘッダーのあるキューでグループの一部が利用可能になり **redelivered=True**、残りのグループは欠落しています。

このため、アプリケーションの使用は、すべてのグループ化されたメッセージを一度に確認するように注意する必要があります。

バグを報告します。

4.7.3. qpid-config を使用したメッセージグループのキューの設定

このサンプル **qpid-config** コマンドは「MyMsgQueue」というキューを作成し、メッセージのグループ化が有効になり、ヘッダーキー「GROUP_KEY」を使用してメッセージグループを特定します。

```
qpid-config add queue MyMsgQueue --group-header="GROUP_KEY" --shared-groups
```

バグを報告します。

4.7.4. デフォルトグループ

ヘッダーのグループ識別子がないメッセージグループを有効にしてキューに到達するすべてのメッセージは、同じ「デフォルト」グループに属すると見なされます。このグループはです **qpid.no-group**。他のグループにメッセージを割り当てることができない場合は、このグループに割り当てられます。

バグを報告します。

4.7.5. デフォルトのグループ名の上書き

キューにメッセージグループを有効にすると、メッセージはヘッダーフィールドとの一致に基づいてグループ化されます。グループのヘッダーに一致していないメッセージは、default グループに割り当てられます。デフォルトグループは、のように事前設定されてい **qpid.no-group**ます。このデフォルトのグループ名を変更するには、起動時に **default-message-group** configuration パラメーターの値を broker に指定します。たとえば、コマンドラインを使用します。

```
qpidd --default-message-group "EMPTY-GROUP"
```

バグを報告します。

4.8. 代替の交換

4.8.1. 拒否されたメッセージと順序付けされたメッセージ

メッセージはコンシューマーによって明示的に *拒否* できます。信頼できるリンクでメッセージを取得する場合、コンシューマーはブローカーがリリースするメッセージを認識する必要があります。メッセージを受け入れる代わりに、コンシューマーはメッセージを *拒否* できます。キューに代替の交換が指定されていない限り、ブローカーは拒否されたメッセージを破棄します。この場合、ブローカールートはメッセージを代替の交換に拒否します。

メッセージは、削除されるキューにある場合に孤立します。孤立したメッセージは、キューに別の交換が設定されていない限り破棄されます。その場合は、交換が代替の交換にルーティングされます。

バグを報告します。

4.8.2. 代替の交換

代替の交換 では、最初のルーティングを介して配信できないメッセージの配信代替が提供されます。

キューに指定された代替の交換では、2 種類のルーティング不可能なメッセージが代替の交換に送信されます。

1. メッセージコンシューマーによって取得されて拒否されるメッセージ（*拒否メッセージ*）。
2. 削除されるキューの未承認メッセージ（*孤立したメッセージ*）。

交換に指定された代替の交換では、ルーティング不可能なメッセージが代替の交換に送信されます。

1. 交換にマッチするバインディングがないルーティングキーを使用して、交換に送信されたメッセージ。

メッセージは、以前別の交換にルーティングされた後、孤立または拒否される場合に 2 番目の交換に再度ルーティングされません。これにより、再ルーティングの無限ループが発生するのを防ぎます。

ただし、一致するバインディングがないため、メッセージが別の交換にルーティングされ、その交換によって配信できない場合は、あるサーバーが設定された場合は、その交換の代替交換に **再度** ルーティングされます。これにより、デッドレターキューにフェイルオーバーできます。

バグを報告します。

4.9. キューのサイズ

4.9.1. キューのサイズの制御

キューのサイズを制御することは、メッセージングシステムのパフォーマンス管理において重要な部分です。

キューの作成時に、キューの最大キューサイズ(**qpid.max_size**)および最大メッセージ数(**qpid.max_count**)を指定できます。

qpid.max_size はバイト単位で指定されます **qpid.max_count**。これはメッセージの数として指定されます。

以下は、メモリーの最大サイズが 200 MB で最大 5000 メッセージを持つキューを **qpid-config** 作成します。

```
qpid-config add queue my-queue --max-queue-size=204800000 --max-queue-count 5000
```

アプリケーションでは、**qpid.max_count** および **qpid.max_size** ディレクティブは **arguments** の内にあり **x-declare node**ます。たとえば、以下のアドレスは上記の **qpid-config** コマンドとしてキューを作成します。

python

```
tx = ssn.sender("my-queue; {create: always, node: {x-declare: {'auto-delete': True, arguments: {'qpid.max_count': 5000, 'qpid.max_size': 204800000}}}}")
```

このコードの実行時にキューが存在しない場合のみ、**qpid.max_count** 属性が適用されることに注意してください。

制限に達する場合の動作： qpid.policy_type

キューがこれらの制限に到達した場合の動作は設定可能です。デフォルトでは、非**durable** キューでは動作はになり **reject**ます。さらにキューへの送信を試みると、送信元で **TargetCapacityExceeded** 例外が発生します。

設定可能な動作は、**qpid.policy_type** オプションを使用して設定されます。以下の値が使用できます。

reject
: メッセージパブリッシャーは例外をスローし **TargetCapacityExceeded**ます。これは、**durable** キュー以外のデフォルトの動作です。

リング
: 新しいメッセージのスペースを作成するために、最も古いメッセージは削除されます。

以下の **qpid-config** コマンド例は、制限ポリシーをに設定 **ring**します。

```
qpid-config add queue my-queue --max-queue-size=204800 --max-queue-count 5000 --limit-policy ring
```

同様に、アプリケーションでも同じことが実施されます。

python

```
tx = ssn.sender("my-queue; {create: always, node: {x-declare: {'auto-delete': True, arguments:
{'qpid.max_count': 5000, 'qpid.max_size': 204800, 'qpid.policy_type': 'ring'}}}}")
```

関連項目

- 「プロデューサーフロー制御」
- 「リングキューでの上書きされたメッセージの検出」
- 「disk-paged キュー」

バグを報告します。

4.9.2. disk-paged キュー

MRG 3 は MRG 2 **flow-to-disk** キューポリシーを、より高性能な *ページキュー*に 置き換えます。ページキューはファイルによってサポートされ、メッセージのページが設定可能な数のメモリーに保持されます。ページ化されたキューは、（キューが追加のストレージにファイルシステムを使用できるようにすることで）個別のメッセージではなくメッセージのメモリー内およびメッセージのページを書き込み（キューがファイルシステムを使用できるようにすることで）応答のパフォーマンスを分散します。

メッセージは *ページに保存されます*。メモリーページのサイズは設定可能で、ページ内にメッセージを適合できるように、予想される最大メッセージサイズよりも大きく設定する必要があります。ページサイズより大きいメッセージはブローカーによって拒否されます。

メモリーを保持するページ数は設定可能です。メッセージ内のメモリー内のページの最大数が設定されると、追加のメッセージによりページがディスクファイルへスワップアウトされ、空のインメモリーページで追加のメッセージを受け取ることができます。ページは、パフォーマンスを最適化するためにメモリーに保持されている間、ディスクにプロアクティブに書き込まれます。メモリー内ではないページからメッセージを要求すると、そのページはディスクから取得されます。メモリー内のページ制限に達すると、新しいページをロードできるようにページインメモリーがディスクに送信されます。

ブローカーがサポートするページの上限は、システムマッピングの制限によって決まります。これはカーネル属性であり **cat /proc/sys/vm/max_map_count**、によって検証され、以下のようにランタイム時に設定できます。

```
echo 100000 >/proc/sys/vm/max_map_count
```

この制限の設定方法は永続的ではなく、再起動後に消去されることに注意してください。永続的な方法で制限を設定するには、**/etc/sysctl.conf** ファイルを使用します。

ページ化されたキューのサイズは、通常のサイズおよびメッセージカウント制限で制御できます。

ページ化されたキューのディスクベースのストレージは本質的に永続的ではないことに注意してください。これは、キューおよびバランスメモリーの使用を管理するためにランタイム時に使用され、ブローカーの再起動後も永続化されません。ページ化されたキューは宣言できます。これは **durable**、要求するメッセージの永続性を提供します。

ページキューの制限

- ページ単位のキューはページサイズを超えるメッセージを処理できないため、キューは最大予想されるメッセージよりも大きいページで設定する必要があります。
- ページキューを LVQ または優先度キューにすることはできません。LVQ または Priority が指定されたページキューの作成を試みると例外が発生します。

ページ化されたキューの作成

キューをページ化されたキューとして設定するには、キューを宣言する **qpid.paging true** 際に引数を指定します。

その他の設定オプションは以下のとおりです。

qpid.max_pages_loaded

指定された時点でメモリーに保持できるページ数を制御します。デフォルト値は 4 です。

qpid.page_factor

ページのサイズを制御します。デフォルト値は 1 です。この値は、プラットフォームが定義するページサイズの倍数です。Linux では、プラットフォームで定義されたページサイズをコマンドを使用して確認でき **getconf PAGESIZE**ます。これは通常、CPU アーキテクチャーに応じて 4k です。

例

以下のコマンドラインの例は、ページ化されたキューの作成を示しています。

```
qpid-config add queue my-paged-queue --argument qpid.paging=True --argument
qpid.max_pages_loaded=100 --argument qpid.page_factor=1
```

同様に、以下の方法でコードで実行できます。

python

```
tx = session.sender("my-paged-queue; {create: always, node: {x-declare: {'auto-delete': True,
arguments:{'qpid.page_factor': 1, 'qpid.max_pages_loaded': 100, 'qpid.paging': True}}}}")
```

バグを報告します。

4.9.3. リングキューでの上書きされたメッセージの検出

リングキューは、キュー容量に達すると、古いメッセージが受信メッセージで上書きされます。一部のアプリケーションは、メッセージが上書きされたときに注意する必要があります。これには、キューを **qpid.queue_msg_sequence** 引数で宣言します。

qpid.queue_msg_sequence 引数は、単一の文字列値をパラメーターとして受け入れます。この文字列の値は、リングキューを通過する各メッセージの message プロパティーとして broker によって追加され、プロパティーは連続的に整数値を増やします。

アプリケーションは各メッセージの **qpid.queue_msg_sequence** 指定されたプロパティーの値を確認して、リングキューで中間メッセージが上書きされたかどうかと、適切に応答されたかどうかを判断できます。

シーケンスで破損を検出するには、メッセージシーケンスをステートフルアプリケーションで調べる必要があることに注意してください。1 つのコンシューマーを持つ排他的キューは、これを実行できます。複数のコンシューマーがキューからメッセージを取得する場合、メッセージシーケンスはコン

シューマー間で分割され、メッセージが上書きされたかどうかを確認する方法はありません。

また、永続キューに送信された永続メッセージでもメッセージシーケンスが永続化されないため、ブローカーの再起動によってシーケンスが廃止されることに注意してください。

以下のコードは、以下の使用を示してい **qpid.queue_msg_sequence**ます。

python

```
import sys
from qpid.messaging import *
from qpid.datatypes import Serial

conn = Connection.establish("localhost:5672")
ssn = conn.session()

name="ring-sequence-queue"
key="my_sequence_key"
addr = "%s; {create:sender, delete:always, node: {x-declare: {arguments:
{'qpid.queue_msg_sequence':'%s', 'qpid.policy_type':'ring', 'qpid.max_count':4}}}}"  % (name, key)
sender = ssn.sender(addr)

msg = Message()
sender.send(msg)

receiver = ssn.receiver(name)
msg = receiver.fetch(1)

try:
    seqNo = Serial(long(msg.properties[key]))
    if seqNo != 1:
        print "Unexpected sequence number. Should be 1. Received (%s)" % seqNo
    else:
       print "Received message with sequence number 1"
except:
    print "Unable to get key (%s) from message properties" % key

"""
Test that sequence number for ring queues shows gaps when queue messages are overwritten
"""

msg = Message()
sender.send(msg)
msg = receiver.fetch(1)
seqNo = Serial(long(msg.properties[key]))

print "Received second message with sequence number %s" % seqNo
# send 5 more messages to overflow the queue
for i in range(5):
    sender.send(msg)

msg = receiver.fetch(1)
seqNo = msg.properties[key]
if seqNo != 3:
    print "Unexpected sequence number. Should be 3. Received (%s) - Message overwritten in ring
```

```
queue." % seqNo
receiver.close()
ssn.close()
```

メッセージシーケンス番号は、署名なしの 32 ビット整数として転送されるため、2^32 でラップします。Python で、「」の **Serial** クラスを使用 **qpid.datatype** してラッピングを処理します。

バグを報告します。

4.9.4. ACL を使用したキューのサイズ制限の実施

ACL で最大 キューサイズを適用できます。これにより、管理者は、ユーザーが多数のシステムリソースを消費できるキューを作成できないようにすることができます。

QUEUE ルールの作成には、メモリーキューとディスク上のキューサイズの両方の上限と下限を制限する ACL ルールがあります。

表4.2 キューサイズ ACL ルール

ユーザーオプション	ACL Limit プロパティー	units
qpid.max_size	queuemaxsizelowerlimit	bytes
	queuemaxsizeupperlimit	bytes
qpid.max_count	queuemaxcountlowerlimit	メッセージ
	queuemaxcountupperlimit	メッセージ
qpid.max_pages_loaded	pageslowerlimit	Page
	pagesupperlimit	Page
qpid.page_factor	pagefactorlowerlimit	整数（プラットフォーム定義のページサイズの倍数）
	pagefactorupperlimit	整数（プラットフォーム定義のページサイズの倍数）

ACL Limit Properties は、CREATE QUEUE 要求でユーザーがいずれかのオプションを表示すると評価されます。ユーザーのオプションが要求を許可する ACL ルールの制限プロパティー内にない場合、ルールは Deny の結果と一致します。

Deny ルールでは、制限プロパティーは無視されます。

例：

```
# Example of ACL specifying queue size constraints
# Note: for legibility this acl line has been split into multiple lines.
acl allow bob@QPID create queue name=q6 queuemaxsizelowerlimit=500000
```

```
queuemaxsizeupperlimit=1000000
queuemaxcountlowerlimit=200
queuemaxcountupperlimit=300
```

以下に示すようにキューが作成されると、これらの制限が再生されます。

C++

```
int main(int argc, char** argv) {
const char* url = argc>1 ? argv[1] : "amqp:tcp:127.0.0.1:5672";
const char* address = argc>2 ? argv[2] :
    "message_queue; “
    “ { create: always, “
    “   node: “
    “   { type: queue, “
    “     x-declare: ”
    “     { arguments: “
    “       { qpid.max_count:101,”
    “         qpid.max_size:1000000”
    “       }”
    “     }”
    “   }”
    “ }";
std::string connectionOptions = argc > 3 ? argv[3] : "";

Connection connection(url, connectionOptions);
try {
    connection.open();
    Session session = connection.createSession();
    Sender sender = session.createSender(address);
...
```

このキューは、**qpid-config** コマンドでも作成できます。

```
qpid-config add queue --max-queue-size=1000000 --max-queue-count=101
```

ACL ルールが処理されると、アクター、アクション、オブジェクト、およびオブジェクト名がすべて一致することを仮定すると、allow または deny のルールと一致します。ただし、ACL ルールはさらに制限され、500000 <= max_size <= 1000000 および 200 <= max_count <= 300 に制限されます。**queue_option max_count** は 101 であるため、サイズ制限は違反され（低すぎる）、許可ルールが拒否の決定と共に返されます。

上限 **と** 下限の **両方** を設定することは必須ではありません。低い制限のみを設定するか、または上限のみを設定できます。

バグを報告します。

4.9.5. Queue Threshold Alerts(Edge-triggered)

MRG 3 では、キューのしきい値アラートが *エッジによって引き起こされ* ます。しきい値はキューごとに設定され、以下になります。

- **qpid.alert_count_up** - 上限しきい値（メッセージ）

- **qpid.alert_size_up** - 上限（バイト）
- **qpid.alert_count_down** - 低いしきい値（メッセージ）
- **qpid.alert_size_down** - しきい値未満（バイト）

デフォルトでは、上限しきい値はキューの最大サイズ/カウントで乗算したグローバルしきい値比率に設定されます。グローバルしきい値比率は、**--default-event-threshold-ratio** コマンドラインオプションを使用して指定できます。指定しない場合は、デフォルトで 50% に設定されます。

デフォルトでは、下限しきい値は、デフォルトの上限しきい値の半分に設定されます。

注記： 上限と下限のしきい値には、イベントレートを制限するために、そのしきい値間に差があります。

イベント

2 つの異なるイベントがあります。

しきい値の上限の増加

キューの深さが（）から(***upper-threshold - 1***)に移動し、増加イベントフラグが設定さ ***upper-threshold*** れていない場合は、増加イベントが発生します。イベントが増加すると、増加イベントフラグが設定されます。増加するイベントが発生する前に、増加するイベントフラグをクリア（降格イベント）する必要があります。これにより、このイベントが複数回再トリガーされるのを防ぐことができます。これは、上位スレッシングの周りにキューの深さをフラッシュすることで、このイベントを複数回実行しないようにします。

しきい値の上限を下げる

増加するイベントフラグが設定され、キューの深さが(***lower-threshold + 1***)から順に移動すると、降順イベントが発生し ***lower-threshold***ます。イベントの減少により、増加するイベントフラグが消去され、増加するイベントがさらに発生し、下層にあるキューの深さをフラッシュすることで、このイベントを複数回繰り返し発生するのを防ぐことができます。

イベントは QMF フレームワークから送信されます。アドレスをリッスンしてイベントメッセージにサブスクライブできます。

- **qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdCrossedUpward.#**
- **qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdCrossedDownward.#**

イベントはマップメッセージとして送信されます。

qmf::org::apache::qpid::broker::EventQueueThresholdCrossedUpward(name, count, size)

qmf::org::apache::qpid::broker::EventQueueThresholdCrossedDownward(name, count, size)

以下のコードは、しきい値イベントメッセージへのサブスクライブおよび消費を示しています。

ウィンドウ 1

python

```
import sys
```

```
from qpid.messaging import *
conn = Connection.establish("localhost:5672")
session = conn.session()
rcv =
session.receiver("qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdCrossedUpward.#")
while True:
  event = rcv.fetch()
  print "Threshold exceeded on queue %s" % event.content[0]["_values"]["qName"]
  print "at a depth of %s messages, %s bytes" % (event.content[0]["_values"]["msgDepth"],
event.content[0]["_values"]["byteDepth"])
  session.acknowledge()
```

ウィンドウ 2

python

```
import sys
from qpid.messaging import *
connection = Connection.establish("localhost:5672")
session = connection.session()
rcv = session.receiver("threshold-queue; {create:always, node:{x-declare:{auto-delete:True,
arguments:{'qpid.alert_count_down':1,'qpid.alert_count_up':3}}}}")
snd = session.sender("threshold-queue")

snd.send("Message1")
snd.send("Message2")
snd.send("Message3")
rcv.fetch()
rcv.fetch()
rcv.fetch()
```

バグを報告します。

4.10. キューの削除

4.10.1. qpid-config でキューの削除

以下の **qpid-config** コマンドは、空のキューを削除します。

```
qpid-config del queue queue-name
```

このコマンドは、削除を実行する前にキューが空であることをチェックし、キューにメッセージが含まれる場合にエラーを報告し、キューにメッセージが含まれている場合は削除しません。

メッセージが含まれるキューを削除するには、**--force** スイッチを使用します。

```
qpid-config del queue queue-name --force
```

バグを報告します。

4.10.2. 自動的に削除されたキュー

キューは、*自動削除* するように設定できます。ブローカーは、宣言されたセッションの終了時にサブスクライバーがない場合は自動削除キューを削除 **し** ます。

アプリケーションはキュー自体を削除できますが、アプリケーションが失敗したり、その接続が失われても、キューをクリーンアップする機会が得られないことがあります。キューを自動削除として指定すると、キューが不要になったときにキューをクリーンアップする責任がブローカーに委譲されます。

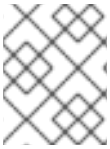
自動削除されたキューは通常、メッセージを **受信** するためにアプリケーションによって作成されます。たとえば、サービスから情報をリクエストするときにメッセージの「reply-to」プロパティーに指定する応答キューです。このシナリオでは、アプリケーションは独自の使用のためにキューを作成し、交換をサブスクライブします。コンシューマーアプリケーションが停止しると、キューは自動的に削除されます。メッセージブローカーから情報を受信する **qpid-config** ユーティリティーが作成したキューはこのパターンの例です。

最後のコンシューマーがサブスクリプションをキューに解放した後に、ブローカーによって設定されるキューが **auto-delete** 削除されます。**auto-delete** キューの作成後、コンシューマーがキューにサブスクライブするとすぐに削除の対象になります。キューにサブスクライブするコンシューマーの数がゼロになると、キューが削除されます。

以下は、Python API を使用して "my-response-queue" という名前の自動削除キューを作成する例です。

python

```
responsequeue = session.receiver('my-response-queue; {create:always, node:{x-declare:{auto-delete:True}}}')
```



注記

このキューの作成時にバインディングが指定されていないため、サーバーの交換（事前設定された名前なしのダイレクト **default** 交換）にバインドされます。

カスタムタイムアウト

削除が発生する前に猶予期間を提供するようにカスタムタイムアウトを設定できます。

注記

MRG-M 3.1.0 以降、C++ クライアントはデフォルト値の 120 秒をすべての永続的サブスクリプションに追加します。qpid python および Java クライアントにはデフォルトの設定がないため、手動で設定する必要があります。

指定 ***qpid.auto_delete_timeout:0*** されている場合には効果がありません。パラメーターを 0 に設定すると遅延自動削除機能が無効になります。

120 秒のタイムアウトを指定すると、最後のコンシューマーがキューから切断されてから削除されるまで、ブローカーは 120 秒間待機します。コンシューマーがその猶予期間内のキューにサブスクライブしている場合、キューは削除されません。これは、キューの情報を失うことなく、コンシューマーが接続をドロップし、再接続できるようにするのに役立ちます。

以下は、Python API を使用して「my-response-queue」の名前で自動削除キューを作成し、自動削除のタイムアウト（120 秒）を作成します。

python

```
responsequeue = session.receiver("my-response-queue; {create:always, node:{x-declare:{auto-delete:True, arguments:{'qpid.auto_delete_timeout':120}}}}")
```

アプリケーションが受信側で開いたままの場合は、公開の自動削除キューを削除できます。交換に送信しているのでエラーは発生しませんが、メッセージは現在存在しないキューには送信されません。

自己作成済みの自動削除キューにパブリッシュする場合は、自動削除されたキューの使用が適切な方法であるかどうかを慎重に検討してください。回答が「yes」の場合には（クリーンアップするテストに有用）、作成時にキューにサブスクライブします。その後、サブスクリプションはハンドルとして機能し、キューはリリースされるまで削除されません。

Python API の使用：

python

```
testqueue = session.sender("my-test-queue; {create:always, node:{x-declare:{auto-delete:True}}}")
testqueuehandle = session.receiver("my-test-queue")
    .....
connection.close()
# testqueuehandle is now released
```

キューを宣言したセッションは、サブスクライバーのみが可能であるため、コンシューマーがサブスクライブし、Auto-deletion の呼び出しをサブスクライブ解除することが例外で、キューが終了したセッションが終了する **exclusive** と、**auto-delete**これらのキューはブローカーによって削除されます。

バグを報告します。

4.10.3. キューの削除チェック

キューの削除が要求されると、以下のチェックが発生します。

- ACL が有効になっている場合、ブローカーは削除を開始したユーザーがこれを実行するパーミッションがあることを確認します。
- **ifEmpty** フラグが渡されると、キューが空でない場合は、ブローカーによって例外が発生します。
- **ifUnused** フラグが渡されると、キューにサブスクライバーがある場合にブローカーによって例外が発生します。
- キューが排他的である場合、ブローカーは削除を開始したユーザーがキューを所有することを確認します。

バグを報告します。

4.11. プロデューサーフロー制御

4.11.1. フロー制御

ブローカーは、制限が設定されているキューにプロデューサーフロー制御を実装します。これにより、宛先キューのオーバーフローが発生する可能性があるメッセージプロデューサーがブロックされます。十分なメッセージが配信され、確認応答されると、キューはブロック解除されます。

フロー制御は、送信元とブローカー間の信頼できるリンクに依存します。これは、送信されたメッセージの承認を停止して、メッセージプロデューサーが送信元再生バッファー容量に到達し、送信を停止することで機能します。

タイプが Limit Policy で設定されたキューには、キューフローのしきい値が有効になってい **ring ません**。これらのキューは、**ring** メカニズム を介してキャパシティーへのアクセスを処理します。制限のある他のすべてのキューには、キューの作成時にブローカーによって設定される 2 つのしきい値があります。

flow_stop_threshold

超過時にフロー制御を有効にするキューリソースの使用状況レベル。渡されたら、キューはオーバーフローが発生したと見なされ、プロデューサーフロー制御への送信メッセージの承認がブローカーによって承認されます。キューサイズまたはメッセージカウントの容量使用率の **いずれか** がこれをトリガーできることに注意してください。

flow_resume_threshold

以下にドロップされた場合にフロー制御を無効にするキューリソースの使用状況レベル。渡された後、キューはオーバーフローの発生で考慮されなくなり、ブローカーは送信されたメッセージを再度確認します。プロデューサーフロー制御が非アクティブになるまで、キューサイズとメッセージ数の **両方** が、このしきい値を下回る必要があることに注意してください。

これら 2 つのパラメーターの値は、容量制限の割合です。たとえば、キューに 204800 **qpid.max_size** (200MB)があり、ある場合 **80**、キューが 204800 または 163840 バイト **flow_stop_threshold** のエンキューメッセージである場合は、ブローカーがプロデューサーフロー制御を開始します。

キューのリソース使用率がを下回ると **flow_resume_threshold**、プロデューサーフロー制御が停止します。**flow_resume_threshold** 上記の設定で **flow_stop_threshold** は、プロデューサーフロー制御のロックが明確な結果となるため、実行しないでください。

バグを報告します。

4.11.2. キューフローの状態

キューのフロー制御の状態は、キューの QMF 管理オブジェクトの **flowState** ブール値によって決定できます。これが **true** フロー制御がアクティブである場合。

キューの管理オブジェクトには、フロー制御がキューに対してアクティブになるたびに **flowStoppedCount** 増分するカウンターも含まれます。

バグを報告します。

4.11.3. ブローカーのデフォルトフローのしきい値

以下の 2 つのブローカーオプションを使用して、デフォルトのフロー制御しきい値をブローカーに設定できます。

- **--default-flow-stop-threshold** = この容量の割合（サイズまたはカウント）でフロー制御がアクティベートされます。

- **--default-flow-resume-threshold** = この容量の割合（サイズまたはカウント）でフロー制御が非アクティブになる

たとえば、以下のコマンドは、フロー制御がキュー容量の 90 % でデフォルトでアクティブになるようにブローカーを起動し、キューが容量を 75% に戻したときに非アクティブにします。

```
qpidd --default-flow-stop-threshold=90 --default-flow-resume-threshold=75
```

バグを報告します。

4.11.4. ブローカー全体のデフォルトのフローしきい値の無効化

デフォルトでブローカーのすべてのキューのフロー制御をオフにするには、デフォルトのフロー制御パラメーターを 100% に設定してブローカーを起動します。

```
qpidd --default-flow-stop-threshold=100 --default-flow-resume-threshold=100
```

バグを報告します。

4.11.5. キューごとのフローのしきい値

以下の引数を使用して、キューの特定のフローしきい値を設定できます。

qpid.flow_stop_size
: **integer** フロー停止のしきい値（バイト単位）。

qpid.flow_resume_size
: **integer** フローは、バイト単位でしきい値を再開します。

qpid.flow_stop_count
: **integer** フローがメッセージ数としてしきい値を停止します。

qpid.flow_resume_count
: **integer** フローはしきい値をメッセージ数として再開します。

特定のキューのフロー制御を無効にするには、そのキューのフロー制御パラメーターをゼロに設定します。

バグを報告します。

第5章 永続メッセージによる確実に配信

5.1. 永続メッセージ

永続メッセージは、ブローカーが失敗する場合でも失われる必要のないメッセージです。

メッセージが永続的としてマークされ、永続キューに送信される と、ディスクに書き込まれ、ブローカーが失敗またはシャットダウンした場合に再起動時に再開します。

永続的としてマークされ、非永続キューに送信されるメッセージはブローカーによって永続化されません。

JMS API を使用して送信されたメッセージは、デフォルトで永続化されます。JMS API を使用して永続キューにメッセージを送信し、永続性のオーバーヘッドが発生したくない場合は、メッセージ永続化を false に設定します。

C++ API を使用して送信されるメッセージは、デフォルトでは永続化されません。C++ API の使用時にメッセージを永続的にマークするには、を使用 **Message.setDurable(true)** してメッセージを永続的としてマークします。

バグを報告します。

5.2. 永続性のあるキューと保証された配信

5.2.1. 永続ストアの設定

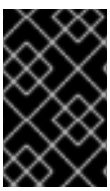
Red Hat Enterprise Messaging ブローカーは、デフォルトで永続性を有効にします。永続性がアクティブであることを確認するには、ログにジャーナルが作成され、ブローカーの起動時にストアモジュールが初期化されていることを確認します。ブローカーログには以下の行が含まれます。

```
notice Journal "TplStore": Created
```

重要

永続モジュールがロードされていない場合、キューが永続化され、メッセージが永続的にマークされている場合でも、メッセージとブローカーの状態はディスクに保存されません。

この **--store-dir** コマンドは、永続ストアに使用するディレクトリーと設定情報を指定します。デフォルトのディレクトリーは、サービスとして **/var/lib/qpidd qpidd** 実行するか、コマンドラインから実行する **~/.qpidd** タイミング **qpidd** です。指定がない場合 **--store-dir** は、によって特定されたディレクトリー内にサブディレクトリーが作成されます **--data-dir**。指定されて **--store-dir** いない場合 **--no-data-dir** は、エラーが発生します。



重要

実行中のブローカーは一度に 1 つのデータディレクトリーにアクセスできます。別のブローカーがデータディレクトリーにアクセスしようとすると、以下のエラーが表示されて失敗します。 ***Exception: Data directory is locked by another process.***

バグを報告します。

5.2.2. 永続キュー

デフォルトでは、キューの有効期間はサーバープロセスの実行にバインドされます。サーバーがシャットダウンするとキューが破棄され、ブローカーの再起動時に再作成する必要があります。*永続キュー*は、予定されているシャットダウンまたは予定外のシャットダウンによってブローカーが再起動した後に自動的に再確立されるキューです。

サーバーがシャットダウンしてキューが破棄されると、キュー内のメッセージはすべて失われます。サーバー再起動時の自動再作成に加え、永続キューは要求する *メッセージのメッセージ永続化* を提供します。永続キューとしてマークされ、永続キューに送信されるメッセージは、シャットダウン後に永続キューが再確立されると保存され、再配信されます。

永続キューに送信されたメッセージはすべて永続的ではなく、永続的とマークされたメッセージのみであることに注意してください。また、メッセージが永続としてマークしても、非永続キューにメッセージが送信されても効果はありません。永続化するには、メッセージが永続的としてマークされ、永続キューに送信される必要があります。

バグを報告します。

5.2.3. qpid-config を使用して永続キューを作成します。

で **--durable** オプションを使用 **qpid-config add queue** して永続キューを作成します。例：

```
qpid-config add queue --durable durablequeue
```

バグを報告します。

5.2.4. メッセージを永続的としてマークします。

*永続メッセージ*は、ブローカーが失敗する場合でも失われる必要のないメッセージです。メッセージを永続化するには、配信モードをに設定し **PERSISTENT**ます。たとえば、C++ では、以下のコードによりメッセージが永続化されます。

```
message.getDeliveryProperties().setDeliveryMode(PERSISTENT);
```

永続メッセージが永続キューに配信されると、キューに置かれるとディスクに書き込まれます。

メッセージプロデューサーが交換に永続メッセージを送信すると、ブローカーはメッセージを永続キューにルーティングし、メッセージが永続ストアに書き込まれるまで待機してから、メッセージプロデューサーに配信を承認します。この時点で、永続キューはメッセージの責任を担い、ブローカーが失敗しても失われないようにすることができます。キューが永続化できない場合、キューのメッセージはディスクに書き込まれません。メッセージが永続的としてマークされていない場合、永続キューにある場合でもディスクには書き込まれません。

表5.1 永続メッセージおよび永続性のあるキューディスクの状態

永続メッセージ AND 永続キュー	ディスクに書き込まれます。
永続メッセージと非永続キュー	ディスクに書き込まれない
非永続的なメッセージ AND 非永続的なキュー	ディスクに書き込まれない
非永続的なメッセージ AND 永続キュー	ディスクに書き込まれない

メッセージコンシューマーがキューからメッセージを読み取りると、コンシューマーがメッセージを承認するまでキューから削除されません（メッセージが永続化されているか、またはキューが永続化されているかどうか）。メッセージを受け入れることにより、コンシューマーはメッセージの責任を取り、キューの責任はなくなります。

バグを報告します。

5.2.5. 再起動後の永続メッセージ状態

ブローカーの再起動後に永続キューが再確立されると、永続とマークされたメッセージはすべて、ブローカーのシャットダウン前に確実に配信されませんでした。ブローカーには、これらのメッセージの配信ステータスに関する情報がありません。シャットダウンが発生する前に配信されても確認されていない可能性があります。これらのメッセージが以前に配信された可能性があることを警告するために、ブローカーは復元された **すべて** の永続メッセージに **redelivered** フラグを設定します。

消費アプリケーションは **redelivered** フラグを提案として扱う必要があります。

バグを報告します。

5.3. メッセージジャーナル

5.3.1. ジャーナルの説明

Messaging Journal (*journal*)という用語は、メッセージのオンディスクストレージを指します。

Red Hat Enterprise Messaging 3 は、ジャーナルの実装として、必要に応じて *動的に* 拡張するリニアストアを使用します。各キューには 1 つのジャーナルがあり、各エンキュー、デキュー、またはトランザクションイベントが順番に記録されます。

journal と **store** という用語は、いずれもオンディスクストレージを参照します。

固定サイズのディスクジャーナルを持つ MRG 2 の legacystore モジュールとは異なり、MRG 3 で使用されるリニアストアは、Empty File Pool(EFP)と呼ばれる空のファイルのプールから必要なファイルを追加して、レコードを EFP に含まないファイルを返します。これにより、サイズ制限なしで任意のサイズのジャーナルが可能になります。

ベストパフォーマンスは、EFP に先にジャーナルに使用できる空のファイルが含まれる場合に取得されます。ただし、EFP がない場合は、ストアは必要に応じてファイルを作成し、フォーマットします（パフォーマンス上の低下）。使用されるファイルは EFP に返されます。

古いストアジオメトリーパラメーターは有効ではなくなりました(**file-size**, **num-jfiles**)。EFP は、ファイルごとにデフォルトのファイルサイズ 2MB を使用します。

broker オプションは、ストアが置かれる場所を **--store-dir** 指定します。ブローカーは、指定のストア dir の下に「**qls**」（Qpid リニアストア）ディレクトリーを作成し、そこで Empty File Pool、db4 データベース、およびジャーナルを見つけます。特定の指定がない場合 **--store-dir** は、に指定されたディレクトリーが使用 **--data-dir** され、デフォルトの場所が使用されます。

バグを報告します。

5.3.2. ジャーナルの設定

ブローカーの永続オプションは、多くのジャーナル要素を制御します。

関連項目

- 「永続オプション」

バグを報告します。

第6章 パフォーマンスチューニングによるメッセージスループットの向上

6.1. リアルタイム JAVA を使用した JMS クライアントの実行

より確定的な動作を実現するには、JMS クライアントを Realtime Java 環境で実行できます。

1. クライアントはリアルタイムオペレーティングシステムで実行する必要があり、リアルタイムの Java ベンダーでサポートされる必要があります。Red Hat は、Sun および IBM の実装のみをサポートします。
2. ベンダーが提供するリアルタイムの .jar ファイルをクラスパスに置きます。
3. 以下の JVM 引数を設定します。

   ```
   -Dqpid.thread_factory="org.apache.qpid.thread.RealtimeThreadFactory"
   ```

   これにより、JMS クライアントが s ではなく **javax.realtime.RealtimeThread**s を使用するようになります **java.lang.Thread**。

   オプションで、スレッドの優先度は以下を使用して設定できます。デフォルト

   ```
   -Dqpid.rt_thread_priority=30
   ```

   では、優先度は 20 に設定されます。
4. ワークロードに基づいて、最適な結果を得るために JVM を調整する必要があります。詳細は、ベンダーの JVM チューニングガイドを参照してください。

バグを報告します。

6.2. QPID-LATENCY-TEST

qpid-latency-test は、レイテンシーを測定するためのコマンドラインユーティリティーです。これは、**qpid-cpp-client-devel** パッケージの一部として提供されます。

実行すると、メッセージングサーバーのパフォーマンスに関する統計が **qpid-latency-test** 提供されます。結果 **qpid-latency-test** とアプリケーションのパフォーマンスを比較して、アプリケーションまたはメッセージングサーバーがパフォーマンスのボトルネックであるかどうかを判断できます。

qpid-latency-test --help では、ユーティリティーの実行に関する詳細情報を提供します。

バグを報告します。

6.3. INFINIBAND

6.3.1. Infiniband の使用

MRG メッセージング接続は Infiniband を使用できます。これにより、高速でポイントツーポイントのシリアルリンクが高速になり、TCP 接続よりも大幅にレイテンシーが短縮されます。

バグを報告します。

6.3.2. Infiniband を使用するための前提条件

サーバーおよびクライアントを実行しているマシンは、それぞれ Infiniband が正しくインストールされている必要があります。特に以下が含まれます。

- Infiniband ハードウェア用のカーネルドライバーとユーザー空間ドライバーの両方をインストールする必要があります。
- Infiniband にロック可能なメモリーを割り当てます。

  デフォルトでは、オペレーティングシステムはすべてのユーザーメモリーをスワップアウトできます。InfiniBand にはロック可能なメモリーが必要ですが、これはスワップアウトできません。各接続には、ロック可能なメモリーが 8 メガバイト（8192 バイト）必要です。

  ロック可能なメモリーを割り当てるには、編集 **/etc/security/limits.conf** して制限を設定します。これは、指定のプロセスが割り当て可能なロック可能なメモリーの最大量です。
- Infiniband インターフェースは、IP over Infiniband を許可するように設定する必要があります。これは、RDMA 接続管理に使用されます。

バグを報告します。

6.3.3. メッセージングサーバーでの Infiniband の設定

前提条件

- Qpid が RDMA を使用するには、パッケージがインストールされている **qpid-cpp-server-rdma** 必要があります。
- RDMA プラグイン（）が **plugins** ディレクトリーに存在 **rdma.so**する必要があります。

手順6.1 メッセージングサーバーでの Infiniband の設定

- **Infiniband 用にロック可能なメモリーの割り当て**
  編集 **/etc/security/limits.conf** して、Infiniband のロック可能なメモリーを割り当てます。

  たとえば、サーバーを実行しているユーザーが qpidd で、64 接続(64*8192=524288)をサポートする場合は、以下のエントリーを追加します。

```
qpidd soft memlock 524288
qpidd hard memlock 524288
```

バグを報告します。

6.3.4. メッセージングクライアントでの Infiniband の設定

前提条件

- パッケージ **qpid-cpp-client-rdma** がインストールされている。

手順6.2 メッセージングクライアントでの Infiniband の設定

- **Infiniband 用にロック可能なメモリーの割り当て**
  ロック可能なメモリー **/etc/security/limits.conf** を割り当てるには編集します。

全ユーザーの制限（例： 16 接続をサポートする(16 *8192=32768)を設定するには、以下のエントリーを追加します。

```
* soft memlock 32768
```

特定ユーザーの制限を設定する場合は、制限を設定する際にそのユーザーの UID を使用します。

```
andrew soft memlock 32768
```

バグを報告します。

第7章 LOGGING

7.1. C++ でのロギング

Qpidd ブローカーおよび C++ クライアントはいずれも環境変数を使用してロギングを有効にできます。Linux システムおよび Windows システムは、同じ名前の環境変数および値を使用します。

1. を使用 **QPID_LOG_ENABLE** して、対象のロギングレベルを設定します（、***trace debug info***、***notice***、、***warning***、***error***、または ***critical***）。

```
export QPID_LOG_ENABLE="warning+"
```

2. Qpidd ブローカーおよび C++ クライアントは、ロギング出力の送信先 **QPID_LOG_OUTPUT** を判断するために使用されます。これは、ファイル名または特別な値 ***stderr stdout***、またはです ***syslog***。

```
export QPID_LOG_TO_FILE="/tmp/myclient.out"
```

3. Windows コマンドプロンプトから、以下のコマンド形式を使用して環境変数を設定します。

```
set QPID_LOG_ENABLE=warning+
set QPID_LOG_TO_FILE=D:\tmp\myclient.out
```

バグを報告します。

7.2. CHANGE BROKER LOGGING VERBOSITY

変更

- 新しいコンテンツ - 2013 年 2 月追加。

コマンドラインで qpidd を実行する場合は、構文で **--log-enable** オプションを使用します。

```
--log-enable LEVEL[+][:PATTERN]
```

設定ファイル（デフォルト/**etc/qpid/qpidd.conf** では）を使用する場合は、以下の行を使用します。

```
log-enable=LEVEL[+][:PATTERN]
```

注記

- **LEVEL** は以下のいずれかです **trace debug info notice warning error critical**。
- 「**+**」は、指定の重大度とより高い重大度をログに記録することを意味します（プラスなしでは、指定の重大度のログのみが有効になります）。
- **PATTERN** はロギング変更の範囲です。
- の文字列は、ロギングステートメントで C++ 関数の完全修飾名と照合 **PATTERN** されます。

- logging ステートメントで C++ 関数の完全修飾名を確認するには、ソースコードを確認するか、qpid 設定に追加するか、qpid ブローカーに強制的 **log-function=yes** にこのようなメッセージをログに記録します。
- したがって、たとえば **qpid::ha** モジュールのすべてと **--log-enable debug+:ha** 一致し、たとえば特定のメソッドのロギングは 1 つの特定のメソッドのみと **--log-enable debug+:broker::Queue::consumeNextMessage** なります（この例では特定の名前空間の **consumeNextMessage** メソッド）。
- **PATTERN** は多くの場合、、、、**acl amqp_0_10 brokerha management** またはのように、デバッグする必要があるモジュールに設定され **store**ます。
- オプションは複数回使用できます。
- **log-enable=debug+:ha**"" などのオプションの 1 つだけを使用すると、ha 情報のデバッグログは有効になりますが、他のログは生成されません。より詳細なロギングを追加するには、上記のオプションを追加して、デフォルト値も追加します。 **log-enable=info+**

バグを報告します。

7.3. BROKER のロギング解決の変更

ブローカーの実行中にロギングのタイムスタンプの解決は変更できます。

手順7.1稼働中のブローカーでの解決ロギングの変更

1. ファイルを編集し **/etc/qpid/qpidd.conf**、以下を追加します。

```
log-time=1
log-enable=info+
log-to-file=/var/lib/qpidd/771830.log
```

2. 起動し **qpid-tool**ます。
3. これで qpid-tool が実行されているので、以下のコマンドを発行します。

```
list broker
```

応答を受け取る前に、これを数回実行しなければならない場合があります。ブローカーからすべての情報を取得するには、しばらく時間がかかる場合があります。

4. 以下のような応答が表示されます。

```
114  14:03:39  -          amqp-broker
```

ブローカーを参照する番号（この例では「114」）。

5. 以下のコマンドを実行します。ブローカーに適した番号を置き換えます。

```
call 114 setLogHiresTimestamp 1
```

6. 次に、ログファイルを確認し **/var/lib/qpidd/771830.log**、高タイムスタンプを使用して開始されていることを確認します。たとえば、別の行を起動するなど、ブローカーがさらにログを記録できるように、何らかの作業を行う必要がある場合があり **qpid-tool**ます。

7. ロギングを通常の解像度に戻すには、で以下のコマンドを実行し **qpid-tool**ます。

```
call 114 setLogHiresTimestamp 0
```

8. 次に、ログファイルを再度確認し、highres を使用して停止したことを確認します。

バグを報告します。

7.4. オブジェクトライフサイクルの追跡

[Model] ログカテゴリーは、Connection オブジェクト、Session、および Subscription オブジェクトの作成、破棄、および主要な状態の変更、および Exchange、Queue、および Binding オブジェクトへの変更を追跡 します。

このログメッセージから、接続を作成したクライアントシステムアドレスのユーザー、その接続で作成されたセッション、およびそのセッションで作成されたサブスクリプションを判断できます。

同様に、exchange-binding-queue オブジェクトはログメッセージに十分な値を持ち、それらの間の対話を関連付けます。オブジェクトの破棄用のログメッセージには、そのオブジェクトに保持されるすべての管理統計の記録が含まれます。ログレコードを使用して作業すると、ブローカーの使用を特定のユーザーに戻すことができます。

debug ログレベルは、対応する管理イベントをミラーリングするログエントリーです。デバッグレベルのステートメントには、ユーザー名、リモートホスト情報、参照オブジェクトにユーザーが指定した名前を使用したその他の参照が含まれます。

trace ログレベルでは、管理リソースの構築と破棄を追跡するログエントリーです。TRACE レベルのステートメントは、内部管理キーを使用してオブジェクトを特定します。削除された各オブジェクトの trace ステートメントには、そのオブジェクトの管理統計が含まれます。

モデルログの有効化

- switch: を使用して、両方のログフレーバーを受信 **--log-enable trace+:Model** します。
- 詳細が低いログ **--log-enable debug+:Model** の場合は、switch: を使用します。

ログの管理対象オブジェクト

すべての管理オブジェクトはトレースログに含まれます。デバッグログには、以下の情報が含まれます **Connection, Queue, Exchange, Binding, Subscription**。

以下は、でキャプチャーされた対応する管理イベントとペアの、実際のログファイルデータの並べ替えです **qpid-printevents**。

1.connection

Create connection

```
event: Fri Jul 13 17:46:23 2012 org.apache.qpid.broker:clientConnect rhost=[::1]:5672-[::1]:34383
user=anonymous
debug: 2012-07-13 13:46:23 [Model] debug Create connection. user:anonymous rhost:[::1]:5672-
[::1]:34383
trace: 2012-07-13 13:46:23 [Model] trace Mgmt create connection. id:[::1]:5672-[::1]:34383
```

コネクションの削除

```
event: Fri Jul 13 17:46:23 2012 org.apache.qpid.broker:clientDisconnect rhost=[::1]:5672-[::1]:34383
```

```
user=anonymous
debug: 2012-07-13 13:46:23 [Model] debug Delete connection. user:anonymous rhost:[::1]:5672-[::1]:34383
trace: 2012-07-13 13:46:29 [Model] trace Mgmt delete connection. id:[::1]:5672-[::1]:34383
Statistics: {bytesFromClient:1451, bytesToClient:892, closing:False, framesFromClient:25, framesToClient:21, msgsFromClient:1, msgsToClient:1}
```

2.session

セッションの作成

```
event: TBD
debug: TBD
trace: 2012-07-13 13:46:09 [Model] trace Mgmt create session. id:18f52c22-efc5-4c2f-bd09-902d2a02b948:0
```

セッションの削除

```
event: TBD
debug: TBD
trace: 2012-07-13 13:47:13 [Model] trace Mgmt delete session. id:18f52c22-efc5-4c2f-bd09-902d2a02b948:0
Statistics: {TxnCommits:0, TxnCount:0, TxnRejects:0, TxnStarts:0, clientCredit:0, unackedMessages:0}
```

3.交換

交換の作成

```
event: Fri Jul 13 17:46:34 2012 org.apache.qpid.broker:exchangeDeclare disp=created exName=myE exType=topic durable=False args={} autoDel=False rhost=[::1]:5672-[::1]:34384 altEx= user=anonymous
debug: 2012-07-13 13:46:34 [Model] debug Create exchange. name:myE user:anonymous rhost:[::1]:5672-[::1]:34384 type:topic alternateExchange: durable:F
trace: 2012-07-13 13:46:34 [Model] trace Mgmt create exchange. id:myE
```

交換の削除

```
event: Fri Jul 13 18:19:33 2012 org.apache.qpid.broker:exchangeDelete exName=myE rhost=[::1]:5672-[::1]:37199 user=anonymous
debug: 2012-07-13 14:19:33 [Model] debug Delete exchange. name:myE user:anonymous rhost:[::1]:5672-[::1]:37199
trace: 2012-07-13 14:19:42 [Model] trace Mgmt delete exchange. id:myE
Statistics: {bindingCount:0, bindingCountHigh:0, bindingCountLow:0, byteDrops:0, byteReceives:0, byteRoutes:0, msgDrops:0, msgReceives:0, msgRoutes:0, producerCount:0, producerCountHigh:0, producerCountLow:0}
```

4.Queue

キューの作成

```
event: Fri Jul 13 18:19:35 2012 org.apache.qpid.broker:queueDeclare disp=created durable=False args={} qName=myQ autoDel=False rhost=[::1]:5672-[::1]:37200 altEx= excl=False user=anonymous
debug: 2012-07-13 14:19:35 [Model] debug Create queue. name:myQ user:anonymous rhost:[::1]:5672-[::1]:37200 durable:F owner:0 autodelete:F alternateExchange:
trace: 2012-07-13 14:19:35 [Model] trace Mgmt create queue. id:myQ
```

キューの削除

```
event: Fri Jul 13 18:19:37 2012 org.apache.qpid.broker:queueDelete user=anonymous qName=myQ
rhost=[::1]:5672-[::1]:37201
debug: 2012-07-13 14:19:37 [Model] debug Delete queue. name:myQ user:anonymous rhost:
[::1]:5672-[::1]:37201
trace: 2012-07-13 14:19:42 [Model] trace Mgmt delete queue. id:myQ
Statistics: {acquires:0, bindingCount:0, bindingCountHigh:0, bindingCountLow:0, byteDepth:0,
byteFtdDepth:0, byteFtdDequeues:0, byteFtdEnqueues:0, bytePersistDequeues:0,
bytePersistEnqueues:0, byteTotalDequeues:0, byteTotalEnqueues:0, byteTxnDequeues:0,
byteTxnEnqueues:0, consumerCount:0, consumerCountHigh:0, consumerCountLow:0,
discardsLvq:0, discardsOverflow:0, discardsPurge:0, discardsRing:0, discardsSubscriber:0,
discardsTtl:0, flowStopped:False, flowStoppedCount:0, messageLatencyAvg:0,
messageLatencyCount:0, messageLatencyMax:0, messageLatencyMin:0, msgDepth:0,
msgFtdDepth:0, msgFtdDequeues:0, msgFtdEnqueues:0, msgPersistDequeues:0,
msgPersistEnqueues:0, msgTotalDequeues:0, msgTotalEnqueues:0, msgTxnDequeues:0,
msgTxnEnqueues:0, releases:0, reroutes:0, unackedMessages:0, unackedMessagesHigh:0,
unackedMessagesLow:0}
```

5.バインディング

バインディングの作成

```
event: Fri Jul 13 17:46:45 2012 org.apache.qpid.broker:bind exName=myE args={} qName=myQ
user=anonymous key=myKey rhost=[::1]:5672-[::1]:34385
debug: 2012-07-13 13:46:45 [Model] debug Create binding. exchange:myE queue:myQ key:myKey
user:anonymous rhost:[::1]:5672-[::1]:34385
trace: 2012-07-13 13:46:23 [Model] trace Mgmt create binding.
id:org.apache.qpid.broker:exchange:,org.apache.qpid.broker:queue:myQ,myQ
```

バインディングの削除

```
event: Fri Jul 13 17:47:06 2012 org.apache.qpid.broker:unbind user=anonymous exName=myE
qName=myQ key=myKey rhost=[::1]:5672-[::1]:34386
debug: 2012-07-13 13:47:06 [Model] debug Delete binding. exchange:myE queue:myQ key:myKey
user:anonymous rhost:[::1]:5672-[::1]:34386
trace: 2012-07-13 13:47:09 [Model] trace Mgmt delete binding.
id:org.apache.qpid.broker:exchange:myE,org.apache.qpid.broker:queue:myQ,myKey
Statistics: {msgMatched:0}
```

6.Subscription

サブスクリプションの作成

```
event: Fri Jul 13 18:19:28 2012 org.apache.qpid.broker:subscribe dest=0 args={} qName=b78b1818-
7a20-4341-a253-76216b40ab4a:0.0 user=anonymous excl=False rhost=[::1]:5672-[::1]:37198
debug: 2012-07-13 14:19:28 [Model] debug Create subscription. queue:b78b1818-7a20-4341-a253-
76216b40ab4a:0.0 destination:0 user:anonymous rhost:[::1]:5672-[::1]:37198 exclusive:F
trace: 2012-07-13 14:19:28 [Model] trace Mgmt create subscription.
id:org.apache.qpid.broker:session:b78b1818-7a20-4341-a253-
76216b40ab4a:0,org.apache.qpid.broker:queue:b78b1818-7a20-4341-a253-76216b40ab4a:0.0,0
```

サブスクリプションの削除

```
event: Fri Jul 13 18:19:28 2012 org.apache.qpid.broker:unsubscribe dest=0 rhost=[::1]:5672-
[::1]:37198 user=anonymous
```

> debug: 2012-07-13 14:19:28 [Model] debug Delete subscription. destination:0 user:anonymous rhost:
> [::1]:5672-[::1]:37198
> trace: 2012-07-13 14:19:32 [Model] trace Mgmt delete subscription.
> id:org.apache.qpid.broker:session:b78b1818-7a20-4341-a253-
> 76216b40ab4a:0,org.apache.qpid.broker:queue:b78b1818-7a20-4341-a253-76216b40ab4a:0.0,0
> Statistics: {delivered:1}

バグを報告します。

第8章 接続およびリソースの保護

8.1. 簡易認証およびセキュリティー層 - SASL

8.1.1. SASL - 簡易認証およびセキュリティーレイヤー

MRG Messaging は Simple Authentication and Security Layer(SASL)を使用して、AMQP 仕様に記載されているようにブローカーへの着信接続を特定および承認します。SASL はさまざまな認証方法を提供します。メッセージングブローカーは、利用可能な SASL メカニズムの任意の組み合わせを許可するように設定できます。クライアントはメッセージングブローカーとネゴシエートして、両方が使用できる SASL メカニズムを見つけることができます。

MRG メッセージングクライアント（JMS クライアントを除く）およびブローカーは Cyrus SASL ライブラリーを使用して完全な SASL 実装を可能にします。

バグを報告します。

8.1.2. Windows クライアントでの SASL サポート

Windows Qpid C++ および C# クライアントは、**ANONYMOUS PLAIN** および **EXTERNAL** 認証メカニズムのみをサポートします。

現時点では、Windows ではその他の SASL メカニズムはサポートされません。

sasl-mechanism が指定されていない場合、選択したデフォルトのメカニズムは通常 Windows と Linux で異なります。

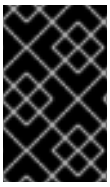
バグを報告します。

8.1.3. SASL メカニズム

変更

- 2013 年 4 月更新されました。

ブローカーによって許可される SASL 認証メカニズムは、ブローカーのファイルによって制御され **/etc/sasl2/qpidd.conf** ます。許可されるメカニズムを小さなサブセットに絞り込むには、このファイルを編集してメカニズムを削除します。



重要

PLAIN 認証メカニズムは、パスワードをクリアテキストで送信します。このメカニズムを使用する場合は、完全なセキュリティーサービスライブラリー(SSL)を使用することが推奨されます。

SASL メカニズム

ANONYMOUS

クライアントは匿名で接続できます。

ブローカーとの起動時に **auth=no**、認証は無効になり **PLAIN**、**ANONYMOUS** 認証メカニズムは **識別メカニズム** として利用できますが、認証値はありません。

PLAIN

パスワードは、クライアントとブローカーの間でプレーンテキストで渡されます。これはセキュアなメカニズムではなく、開発環境でのみ使用してください。PLAIN が実稼働環境で使用されている場合は、トランスポートの SSL 暗号化がパスワードを保護する SSL 接続でのみ使用してください。

ブローカーの起動時に **auth=no**、認証が無効になることに注意してください。**PLAIN** および **ANONYMOUS** 認証メカニズムは **識別メカニズム** として利用できますが、認証値はありません。

DIGEST-MD5

HTTP ヘッダーを使用した MD5 ハッシュ化されたパスワード交換。これは、中程度のセキュリティープロトコルです。

CRAM-MD5

MD5 暗号化を使用するチャレンジ応答プロトコル。

KERBEROS/GSSAPI

Generic Security Service Application Program Interface(GSSAPI)は、異なるセキュリティープロバイダーの接続を可能にするフレームワークです。最も頻繁に使用されるのは Kerberos です。GSSAPI セキュリティーは、シングルサインオン、不透明なトークン交換、トランスポートセキュリティーなどのセキュリティーを一元管理します。

EXTERNAL

外部 SASL 認証は、クライアントとサーバーとの間の SSL 暗号化接続を使用します。クライアントは、接続を暗号化する証明書を提示します。この証明書には、接続の暗号化キーとクライアントのアイデンティティーの両方が含まれます。

バグを報告します。

8.1.4. SASL メカニズムおよびパッケージ

以下の表は、認証メカニズムごとにサーバーにインストールする必要がある **cyrus-sasl-*** パッケージを表しています。

表8.1

method	パッケージ	/etc/sasl2/qpidd.conf entry
ANONYMOUS	-	-
PLAIN	**cyrus-sasl-plain**	**mech_list: PLAIN**
DIGEST-MD5	**cyrus-sasl-md5**	**mech_list: DIGEST-MD5**
CRAM-MD5	**cyrus-sasl-md5**	**mech_list: CRAM-MD5**
KERBEROS/GSSAPI	**cyrus-sasl-gssapi**	**mech_list: GSSAPI**
EXTERNAL	-	**mech_list: EXTERNAL**

バグを報告します。

8.1.5. ローカルパスワードファイルを使用した SASL の設定

ローカル SASL データベースは、PLAIN、DIGEST-MD5、または CRAM-MD5 SASL 認証メカニズムによって使用されます。GSSAPI/KERBEROS メカニズムはクライアントを Kerberos ユーザーデータベースに対して認証し、EXTERNAL メカニズムはクライアントによって使用される SSL 証明書内のクライアント ID を使用して接続を暗号化します。

手順8.1 ローカルパスワードファイルを使用した SASL の設定

1. **saslpasswd2** コマンドを使用して、新しいユーザーをデータベースに追加します。認証および ACL 承認のユーザー ID は、フォームを使用し ***user-id@domain***ます。

   ブローカーに正しいレルムが設定されていることを確認します。これは、設定ファイルを編集するか、**-u** オプションを使用して実行できます。ブローカーのデフォルトレルムはです ***QPID***。

   ```
   # saslpasswd2 -f /var/lib/qpidd/qpidd.sasldb -u QPID new_user_name
   ```

2. 既存のユーザーアカウントは、以下の **-f** オプションを使用して一覧表示できます。

   ```
   # sasldblistusers2 -f /var/lib/qpidd/qpidd.sasldb
   ```

   注記

   のユーザーデータベース **/var/lib/qpidd/qpidd.sasldb** は、 **qpidd** ユーザーがのみ読み取り可能です。 **qpidd** ユーザー以外のユーザーからブローカーを起動する場合は、設定ファイルを変更するか、認証をオフにする必要があります。

   このファイルは、**qpidd** ユーザーが読み取る必要があることに注意してください。このファイルを削除して再作成すると、qpidd ユーザーに読み取り権限があることを確認してください。認証試行は失敗します。

3. 認証をオンまたはオフにするには、ブローカーの起動時に **auth yes|no** オプションを使用します。

   ```
   # /usr/sbin/qpidd --auth yes

   # /usr/sbin/qpidd --auth no
   ```

   また、**/etc/qpid/qpidd.conf** 設定ファイルに適切な行を追加して、認証をオンまたはオフにすることもできます。

   ```
   auth=no

   auth=yes
   ```

   SASL 設定ファイルは Red Hat Enterprise Linux **/etc/sasl2/qpidd.conf** のにあります。

バグを報告します。

8.1.6. ACL を使用した SASL の設定

1. ACL の使用を開始するには、**--acl-file** コマンドを使用してパスとファイル名を指定します。ファイル名には以下の **.acl** 拡張子が必要です。

```
$ qpidd --acl-file ./aclfilename.acl
```

2. 必要に応じて、**--connection-limit-per-user** およびコマンドを使用して、ユーザーごとにアクティブな接続の数を制限することが **--connection-limit-per-ip** できます。この制限は、**--acl-file** コマンドが指定されている場合にのみ適用できます。

3. **cat** コマンドでファイルを表示し、テキストエディターで編集できるようになりました。パスとファイル名が見つからない場合は、起動に **qpidd** 失敗します。

バグを報告します。

8.1.7. Kerberos 5 の設定

Kerberos はブローカーの GSSAPI(Generic Security Services Application Program Interface)認証メカニズムを使用して Kerberos サーバーとの認証を行います。

MRG メッセージングブローカーとユーザーの両方が Kerberos サーバーのプリンシパルであるため、両方のクライアントが Kerberos 認証サービスのクライアントになります。

注記

以下の手順では、ドメイン名および Kerberos レルムの例を使用します。以下の手順に従うには、Kerberos サーバーを設定し、ネットワーク環境に適切なドメイン名と Kerberos レルムを使用する必要があります。

Kerberos を使用するには、ブローカーと各ユーザーの両方を Kerberos サーバーで認証する必要があります。

1. qpidd ブローカーまたは qpidd メッセージングクライアントを実行する各マシンに、Kerberos ワークステーションソフトウェアおよび Cyrus SASL GSSAPI をインストールします。

```
$ sudo yum install cyrus-sasl-gssapi krb5-workstation
```

2. の mech_list 行 **/etc/sasl2/qpidd.conf** を以下のように変更します。

```
mech_list: GSSAPI
```

3. 次の行を追加 **/etc/qpid/qpidd.conf**します。

```
auth=yes
realm=QPID
```

4. Kerberos データベースに Qpid ブローカーを登録します。

   従来は、Kerberos プリンシパルはプライマリー、インスタンス、レルムの 3 つの部分に分けられます。一般的な Kerberos V5 形式は以下のとおりです **primary/instance@REALM**。ブローカーの場合、プライマリーは **qpidd**インスタンスは完全修飾ドメイン名で、REALM は Kerberos ドメインレルムです。デフォルトでは、このレルムはですが **QPID**、以下の例に従って qpid.conf で異なるレルムを指定できます。

```
realm=EXAMPLE.COM
```

たとえば、完全修飾ドメイン名で Kerberos ドメイン名が **dublduck.example.com** の場合 **EXAMPLE.COM**、プリンシパル名はになり **qpidd/dublduck.example.com@EXAMPLE.COM**ます。

```
FDQN=`hostname --fqdn`
REALM="EXAMPLE.COM"
kadmin -r $REALM  -q "addprinc -randkey -clearpolicy qpidd/$FQDN"
```

これで、ブローカーの Kerberos キータブファイルを作成します。ブローカーはキータブファイルへの読み取りアクセスが必要になります。以下のスクリプトはキータブファイルを作成し、ブローカーの読み取りアクセスを許可します。

```
QPIDD_GROUP="qpidd"
kadmin -r $REALM  -q "ktadd -k /etc/qpidd.keytab qpidd/$FQDN@$REALM"
chmod g+r /etc/qpidd.keytab
chgrp $QPIDD_GROUP /etc/qpidd.keytab
```

キータブファイルのデフォルトの場所はです **/etc/krb5.keytab**。別のキータブファイルを使用する場合は、以下のように KRB5_KTNAME 環境変数にファイルの名前を含める必要があります。

```
export KRB5_KTNAME=/etc/qpidd.keytab
```

正しく設定されている場合、に設定して Kerberos サポートをブローカーで有効にできるようになりました。 **AUTH** および **realm** にあるオプション **/etc/qpid/qpidd.conf**:

```
CDATA[# /etc/qpid/qpidd.conf
auth=yes
realm=EXAMPLE.COM
```

ブローカーを再起動して、これらの設定をアクティベートします。

5. 各 Qpid ユーザーが Kerberos データベースに登録され、クライアントマシンで Kerberos が正しく設定されていることを確認してください。Qpid ユーザーは、Qpid メッセージングクライアントを実行するアカウントです。正しく設定されている場合、以下のコマンドは成功します。

```
$ kinit user@REALM.COM
```

6. **Java JMS クライアントの追加設定**
Java JMS クライアントには追加の手順が必要です。

   a. Java JVM は以下の引数で実行する必要があります。

      -Djavax.security.auth.useSubjectCredsOnly=false

      現在のスレッドを所有する「subject」から取得する代わりに、SASL GSSAPI クライアントが Kerberos 認証情報を明示的に取得するよう強制します。

      -Djava.security.auth.login.config=myjas.conf

      jass 設定ファイルを指定します。以下は、JASS 設定ファイルのサンプルです。

      -

```
com.sun.security.jgss.initiate {
    com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
};
```

-Dsun.security.krb5.debug=true

トラブルシューティングの詳細なデバッグ情報を有効にします。

b. クライアント接続 URL は、以下の Kerberos 固有のブローカープロパティーを指定する必要があります。

- **sasl_mechs**をに設定する必要があり **GSSAPI**ます。
- **sasl_protocol**qpidd ブローカーのプリンシパルに設定する必要があります（例：/）**qpidd**。
- **sasl_server**SASL サーバーのホストに設定する必要があります（例：） **sasl.com**。

Kerberos 接続の接続 URL の例を以下に示します。

```
amqp://guest@clientid/testpath?brokerlist='tcp://localhost:5672?
sasl_mechs='GSSAPI'&sasl_protocol='qpidd'&sasl_server='<server-host-name>"
```

バグを報告します。

8.2. TLS/SSL の設定

8.2.1. SSL を使用した暗号化

暗号化と証明書管理 **qpidd** は、Mozilla の Network Security Services Library(NSS)により提供されます。

関連項目

- 付録B *OpenSSL Certificate Reference*

バグを報告します。

8.2.2. クライアント証明書のインストールに関する注意事項

6.4 よりも前の Red Hat Enterprise Linux のバージョンでは、証明書データベースに追加される新しいクライアント証明書をロードするために、ブローカーを再起動する必要があります。

Red Hat Enterprise Linux 6.5 には、クライアント証明書をデータベースに追加し、再起動せずにブローカーが読み込むことができる変更が含まれています。この変更は、Red Hat Enterprise Linux 6.4 でホットフィックスとして利用できます。

関連項目

- 付録B *OpenSSL Certificate Reference*

バグを報告します。

8.2.3. Broker での SSL の有効化

変更

- 2013 年 4 月更新されました。

1. 認証局(CA)により署名された証明書が必要です。この証明書はクライアントによって信頼される必要もあります。サーバー認証に加えてクライアント認証が必要な場合は、クライアント証明書も CA で署名し、ブローカーによって信頼される必要があります。

   証明書データベースは、Mozilla Network Security Services(NSS) **certutil** ツールにより作成および管理されます。このユーティリティーの詳細は、Mozilla Web サイト （SSL 接続の設定およびテストのチュートリアルなど）を参照してください。証明書データベースは通常、パスワードで保護されます。パスワードを指定する最も安全な方法は、データベースの作成時にパスワードファイルを使用し、ブローカーの起動時に **ssl-cert-password-file** オプションでパスワードファイルを指定することです。

   以下のスクリプトは、certutil を使用して証明書データベースを作成する方法を示しています。

   ```
   mkdir ${CERT_DIR}
   certutil -N -d ${CERT_DIR} -f ${CERT_PW_FILE}
   certutil -S -d ${CERT_DIR} -n ${NICKNAME} -s "CN=${NICKNAME}" -t "CT,," -x -f
   ${CERT_PW_FILE} -z /usr/bin/certutil
   ```

   ブローカーの起動時 **ssl-cert-password-file** に *${CERT_PW_FILE}* の値を設定 **ssl-cert-db** し、*${CERT_DIR}* の値に設定し、*${NICKNAME}* の値 ***ssl-cert-name*** に設定し ます。

2. ブローカーの起動時に以下の SSL オプションを使用できます。

   --ssl-use-export-policy

   NSS エクスポートポリシーを使用します。このオプションを指定すると、サーバーは NSS エクスポートポリシーを使用した暗号化のUS エクスポート制限に準拠します。指定しない場合、サーバーはそのポリシーを使用します。詳細は、Mozilla SSL Export Policy Function のドキュメント を参照してください。

   --ssl-cert-password-file *PATH*

   必須。証明書データベースへのアクセスに使用するパスワードを含むプレーンテキストファイル。

   --ssl-cert-db *PATH*

   必須。証明書データベースが含まれるディレクトリーへのパス。

   --ssl-cert-name *NAME*

   使用する証明書の名前。デフォルトはです **localhost.localdomain**。

   --ssl-port *NUMBER*

   SSL 接続をリッスンするポート。ポートを指定しないと、ポート 5671 が使用されます。

   選択した SSL ポートが SSL 以外の接続のポートと同じ場合（例： **--ssl-port** と **--port** オプションが同じ場合）、SSL 暗号化および暗号化されていない接続の両方を同じポートに確立できます。ただし、この設定では IPv6 はサポートされません。

--ssl-require-client-authentication

SSL ハンドシェイク中に SSL クライアント認証（クライアント証明書の検証）が必要になります。これは SASL 認証の前に発生し、SASL から独立しています。

このオプションは、SSL 接続の **EXTERNAL** SASL メカニズムを有効にします。クライアントが **EXTERNAL** メカニズムを選択する場合、クライアントのアイデンティティーは検証された SSL 証明書から取得され **CN**、を使用して、ドメイン **DC**を作成します。たとえば、証明書にプロパティーが含まれる場合 **CN=bob DC=acme DC=com**、クライアントのアイデンティティーはになり **bob@acme.com**ます。

クライアントが別の SASL メカニズムを選択する場合、クライアント証明書からアイデンティティーの取得は、SASL ハンドシェイク時にネゴシエートされたものに置き換えられます。

--ssl-sasl-no-dict

ディクショナリー攻撃によって攻撃される可能性のある SASL メカニズムを許可しないでください。これにより、よりも弱いメカニズムが選択されないようにします。これは **EXTERNAL**、ディクショナリー攻撃による影響を受けません。

--require-encryption

これにより、暗号化された接続 **qpidd** のみが許可されます。これは、SSL-port 上で EXTERNAL SASL を使用するクライアント、または TCP ポートに GSSAPI を使用するクライアントのみを意味します。

バグを報告します。

8.2.4. クライアント用の SSL 証明書のエクスポート

サーバーで SSL が有効になっている場合、クライアントはセキュアな接続を確立するために SSL 証明書のコピーを必要とします。

以下のコマンド例は、ブローカーの NSS データベースからクライアント証明書と秘密鍵をエクスポートするために使用できます。

```
pk12util -o <p12exportfile> -n <certname> -d <certdir> -w <p12filepwfile>

openssl pkcs12 -in <p12exportfile> -out <clcertname> -nodes -clcerts -passin pass:<p12pw>
```

SSL コマンドおよびオプションの詳細は、OpenSSL ドキュメントを参照してください。Red Hat Enterprise Linux のタイプ： **man openssl**

バグを報告します。

8.2.5. Windows での SSL の有効化

以下の手順では、ブローカーから SSL 証明書をエクスポートし、Windows マシンにインストールし、Windows で実行しているクライアントとブローカー間の SSL 接続を有効にします。

手順8.2 ブローカーでの SSL 証明書の作成

1. ブローカーで以下のコマンドを実行し、証明書をエクスポートします。

```
# cd /var/lib/qpidd

# mkdir qpid_nss_db
# cd qpid_nss_db
# ls
# echo password > ssl_pw_file
# cat ssl_pw_file
password

# certutil -S -d . -n qrootCA -s "CN=qrootCA" -t "CT,," -x -m 1000 -v 120 -f ssl_pw_file
# certutil -S -n "fully-qualified-server-name.com" -s "CN="fully-qualified-server-name.com -c
qrootCA -t ",," -m 1001 -v 120 -d . -f ssl_pw_file
# certutil -S -n client -s "CN=client" -t ",," -m 1005 -v 120 -c qrootCA -d . -f ssl_pw_file
# pk12util -d . -o client.p12 -n client
Enter Password or Pin for "NSS Certificate DB":
Enter Password or Pin for "NSS Certificate DB":
Enter password for PKCS12 file:
Re-enter password:
pk12util: PKCS12 EXPORT SUCCESSFUL
# openssl pkcs12 -in client.p12 -out client.pem -nodes -clcerts
Enter Import Password:
MAC verified OK
```

2. ファイルが存在することを確認します。

```
# ls
cert8.db  client.p12  client.pem  key3.db  secmod.db  ssl_pw_file
```

手順8.3 qpid_nss_db フォルダーを他のブローカーマシンにコピーし、その所有者 qpidd として設定します。

1. 他のブローカーで以下のコマンドを実行し、最初のブローカーからファイルをコピーします。

```
# scp -r qpid_nss_db root@other-broker.com:/var/lib/qpidd
# chown -R qpidd:qpidd qpid_nss_db
```

2. ファイルとそのパーミッションを確認します。

```
# ll
total 89896
-rw-r-----. 1 qpidd qpidd        0 Jul 16 06:27 lock
-rw-r--r--. 1 qpidd qpidd 91989014 Nov  1 06:52 qpidd.log
-rw-------. 1 qpidd qpidd    12288 Oct  7 05:32 qpidd.sasldb
drwxr-xr-x. 2 qpidd qpidd     4096 Nov  6 04:32 qpid_nss_db
-rw-r-----. 1 qpidd qpidd       37 Jul 16 06:27 systemId
```

手順8.4 ブローカー設定ファイルの変更

- ブローカー設定ファイルを編集し **/etc/qpid/qpidd.conf**ます。

```
ssl-require-client-authentication=no
log-to-file=/var/lib/qpidd/qpidd.log
ssl-port=5671
```

```
log-enable=info+
ssl-cert-password-file=/var/lib/qpidd/qpid_nss_db/ssl_pw_file
ssl-cert-name=fully-qualified-server-name.com
auth=no
ssl-cert-db=/var/lib/qpidd/qpid_nss_db
```

手順8.5 ブローカーの起動

- ブローカーを起動し、SSL ポートでリッスンしていることを確認します。

```
# service qpidd restart
Stopping Qpid AMQP daemon:          [  OK  ]
Starting Qpid AMQP daemon:          [  OK  ]

# netstat -nap | grep qpidd
tcp        0      0 0.0.0.0:5671            0.0.0.0:*               LISTEN      25184/qpidd
tcp        0      0 0.0.0.0:5672            0.0.0.0:*               LISTEN      25184/qpidd
tcp        0      0 :::5671                 :::*                    LISTEN      25184/qpidd
tcp        0      0 :::5672                 :::*                    LISTEN      25184/qpidd
```

手順8.6 Windows マシンにエクスポートするフォルダーの作成

1. 以下の手順を実行します。
   - Windows マシンにエクスポートするフォルダーの作成
   - .txt 形式で新しいパスワードファイルを作成します。
   - 認証局の証明書を .cer 形式にエクスポート
   - クライアント証明書を .pfx 形式にエクスポートする

```
# mkdir windir
# echo password2 > windir/win_pw_file.txt
# cat windir/win_pw_file.txt
password2
# certutil -L -d qpid_nss_db -n qrootCA -f ssl_pw_file -a > windir/qrootCA.cer
# pk12util -d qpid_nss_db -n client  -k qpid_nss_db/ssl_pw_file -w windir/win_pw_file.txt -o
windir/client.pfx
        pk12util: PKCS12 EXPORT SUCCESSFUL
```

2. ファイルが存在することを確認します。

```
# ls windir
client.pfx  qrootCA.cer  win_pw_file.txt
```

手順8.7 Windows マシンへのファイルのコピー

- Windows マシンに **windir** フォルダーをコピーします。

オプションのパス - GUI またはコマンドライン

Windows マシンに証明書をインストールするには、GUI の使用またはコマンドラインを使用した 2 つのオプションがあります。

手順8.8 認証局のインストール - GUI

1. Windows マシンでの実行 **mmc**
2. **File** / をクリックします。 **Add/Remove Snap-in...**
3. **Certificates** -> -> **Add** -> -> **Computer account** -> **Local computer Finish** -> を選択します。 **OK**
4. コンソールのアンパック証明書（ローカルコンピューター）
5. Trusted Root Certification Authorities を右クリックし、選択します。 **All Tasks/Import...**
6. パスを **qrootCA.cer** ファイルに設定し、Trusted Root Certification Authorities 証明書ストアを選択してアクションを確認し、コンソール設定を保存します。

手順8.9 認証局のインストール - コマンドライン

- 以下のコマンドを実行して、コマンドラインで証明書をインポートします。

```
certmgr.exe -add -c C:\windir\qrootca.cer -s -r localMachine root
```

手順8.10 接続のテスト

- コマンドラインで以下のコマンドを実行し、接続をテストします（環境変数を設定する必要はありません）。

```
C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671 --connection-options {transport:ssl} "amq.topic"
```

オプションのパスウェイ - 環境経由でインストールまたは指定する証明書

Windows マシン証明書ストアに証明書をインストールするか、環境変数で証明書を指定できます。

手順8.11 Windows 証明書ストアへの証明書のインストール

以下の手順に従って、現在のユーザー/Personal 証明書ストア **client.pfx** にクライアント証明書をインストールします。

1. run **mmc**
2. **File** / をクリックします。 **Add/Remove Snap-in...**
3. **Certificates** -> -> **Add>** -> **My user account Finish** -> を選択します。 **OK**
4. コンソールの展開 **Certificates - Current User**
5. 右クリックし **Personal**ます。
6. を選択し **All Tasks / Import**ます。
7. **client.pfx** ファイルへのパスの割り当て
8. をクリックし **Next**ます。
9. からのパスワードを入力します **win_pw_file.txt** （この場合は*password2* ）。

10. コンソール設定 **Certificate Store Personal** を選択して保存します。
11. ブローカー設定を変更してクライアント認証を必要とするようにし、再起動します。
12. 環境変数を設定します。

```
>set QPID_SSL_CERT_STORE=My
>set QPID_SSL_CERT_NAME=client
```

13. メッセージを送信してテストします。

```
>C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671 --connection-
options {transport:ssl,sasl-mechanisms:EXTERNAL} amq.topic
```

手順8.12 環境経由で証明書を指定する

1. Windows マシンに環境変数を設定します。

```
>set QPID_SSL_CERT_FILENAME=<path_to_the_client.pfx>
>set QPID_SSL_CERT_PASSWORD_FILE=<path_to_the_win_pw_file.txt>
>set QPID_SSL_CERT_NAME=client
```

例：

```
>C:\qpid_VS2008\bin\Release>set QPID_SSL_CERT_FILENAME=C:\windir\client.pfx

>C:\qpid_VS2008\bin\Release>set
QPID_SSL_CERT_PASSWORD_FILE=C:\windir\win_pw_fil
e.txt

>C:\qpid_VS2008\bin\Release>set QPID_SSL_CERT_NAME=client
```

2. メッセージを送信してテストします。

```
C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671 --connection-
options {transport:ssl,sasl-mechanisms:EXTERNAL} amq.topic
```

バグを報告します。

8.2.6. C++ クライアントでの SSL の有効化

環境変数を使用して、C++ クライアントに以下のオプションを指定できます。

表8.2 C++ クライアント用の SSL クライアント環境変数

C++ クライアントの SSL クライアントオプション	
QPID_SSL_USE_EXPORT_POLICY	NSS エクスポートポリシーの使用
QPID_SSL_CERT_PASSWORD_FILE *PATH*	証明書データベースへのアクセスに使用するパスワードを含むファイル

C++ クライアントの SSL クライアントオプション	
QPID_SSL_CERT_DB *PATH*	証明書データベースが含まれるディレクトリーへのパス
QPID_SSL_CERT_NAME *NAME*	使用する証明書の名前。SSL クライアント認証を有効にすると、通常、証明書名を指定する必要があります。

SSL 接続を使用する場合は、クライアントが証明書データベースの場所、クライアントの証明書が含まれるディレクトリー、および認証局の公開鍵を指定する必要があります。これを行うには、環境変数 **QPID_SSL_CERT_DB** をディレクトリーの完全パス名に設定します。コネクションが SSL クライアント認証を使用する場合は、クライアントのパスワードも必要になります。パスワードは保護されているファイルに配置する必要があります。また、**QPID_SSL_CERT_PASSWORD_FILE** 変数はこのパスワードが含まれる場所に設定する必要があります。

Qpid Messaging API で SSL が有効になっている接続を開くには、***transport*** connection オプションを に設定し ***ssl***ます。

関連項目

- 「クライアント用の SSL 証明書のエクスポート」
- 付録B *OpenSSL Certificate Reference*

バグを報告します。

8.2.7. Java クライアントでの SSL の有効化

1. サーバーおよびクライアント認証の両方で、信頼された CA をトラストストアとキーストアにインポートし、それらのキーを生成します。生成された鍵を使用して証明書要求を作成し、要求を使用して証明書を作成します。その後、署名済み証明書をキーストアにインポートできます。クライアントの起動時に以下の引数を Java JVM に渡します。

```
-Djavax.net.ssl.keyStore=/home/bob/ssl_test/keystore.jks
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=/home/bob/ssl_test/certstore.jks
-Djavax.net.ssl.trustStorePassword=password
```

2. サーバー側の認証のみの場合、信頼された CA をトラストストアにインポートし、クライアントの起動時に以下の引数を Java JVM に渡します。

```
-Djavax.net.ssl.trustStore=/home/bob/ssl_test/certstore.jks
-Djavax.net.ssl.trustStorePassword=password
```

3. 以下の例に従って、Java クライアントは接続 URL の SSL オプションを使用して SSL 暗号化を有効にする必要があります。

```
amqp://username:password@clientid/test?brokerlist='tcp://localhost:5672?ssl='true''
```

4. SSL 接続で問題をデバッグする必要がある場合は、クライアントの起動時に引数を Java JVM **-Djavax.net.debug=ssl** に渡して、Java の SSL デバッグを有効にします。

関連項目

- 「クライアント用の SSL 証明書のエクスポート」
- 付録B *OpenSSL Certificate Reference*

バグを報告します。

8.2.8. Python クライアントでの SSL の有効化

Python クライアントで SSL を使用するには、以下の **いずれか** を行います。

1. フォームの URL を使用します。*host* はブローカーのホスト名で **amqps://<host>:<port>**、*port* は SSL ポート（通常は 5671）になります。
2. *接続の '**transport**' 属性を '' に設定し **ssl**ます。*

Python クライアントには、SSL 機能にいくつかの制限があります。

*サーバー認証は要求され、認証に **EXTERNAL** SASL メカニズムを使用する場合は、明示的にクライアント名を指定する必要があります。*

- *Python クライアントにはオプションのパラメーターがあります（を **ssl_trustfile** 参照 Python SSL パラメーター）。このパラメーターを指定すると、証明書のトラストストアの検証が実行されます。*
- *オプションのパラメーターが指定されて **ssl_trustfile** いる場合、Python クライアントはサーバーの SSL 証明書と接続ホスト名と照合します。*
- *認証に EXTERNAL SASL メカニズムを使用する場合は、接続文字列にクライアント名を指定する必要があります。接続文字列で提供されるこのクライアント名は、SSL 証明書のアイデンティティーと一致する必要があります。この 2 つがない場合、接続に失敗するか、接続文字列にクライアント名を指定しないか、または SSL 証明書のアイデンティティーに一致するクライアント名を指定します。*

Python SSL パラメーター

QPID Python クライアントは、以下の SSL 関連の設定パラメーターを受け入れます。

- ***ssl_certfile** - 接続のローカル側（クライアント）を識別するために使用される PEM 形式の証明書が含まれるファイルへのパス。これは、サーバーがクライアント側の認証が必要な場合に必要です。*
- ***ssl_keyfile** - クライアントの秘密鍵は、証明書（ssl_certfile など）と同じファイルに保存されます。クライアントの秘密鍵が含まれて **ssl_certfile** いない場合には、このパラメーターを PEM ファイル形式の秘密鍵を含むファイルへのパスに設定する必要があります。*
- ***ssl_skip_hostname_check** - true に設定すると、サーバー証明書に対する接続ホスト名の検証はスキップされます。*
- ***ssl_trustfile** このパラメーターには、信頼できる認証局(CA)証明書のチェーンが含まれる PEM 形式のファイルへのパスが含まれます。これらの証明書は、リモートサーバーの認証に使用されます。*
- *これらのパラメーターは、作成時に **qpid.Connection()** オブジェクトへの引数として渡されます。例：*

```
Connection("amqps://client@127.0.0.1:5671", ssl_certfile="/path/to/certfile",
```

```
ssl_keyfile="/path/to/keyfile")
```

関連項目

- 「クライアント用の SSL 証明書のエクスポート」
- 付録B OpenSSL Certificate Reference

バグを報告します。

8.3. 認可

8.3.1. アクセス制御リスト(ACL)

MRG Messaging では、Authorization は、アクセス制御リスト(ACL)を使用して認証された各ユーザーが実行できるアクションを指定します。

バグを報告します。

8.3.2. デフォルト ACL ファイル

バージョン2.2 までのバージョンでは、デフォルトの ACL ファイルの場所はになり **/etc/qpidd.acl** ます。

バージョン2.3 から、デフォルトの ACL ファイルがに移動し **/etc/qpid/qpidd.acl** ます。変更されていない既存のインストールは引き続き以前の ACL ファイルと場所を使用し、新しいインストールでは新しいデフォルトの場所およびファイルが使用されます。

バグを報告します。

8.3.3. アクセス制御リスト(ACL)の読み込み

--acl-file コマンドを使用して、アクセス制御リストを読み込みます。ファイル名には以下の **.acl** 拡張子が必要です。

```
$ qpidd --acl-file ./aclfilename.acl
```

バグを報告します。

8.3.4. ACL のリロード

QMF メソッドを使用して、**qpid-tool** またはプログラムコードから使用して、ブローカーを再起動せずに ACL をリロードできます。

を使用して ACL をリロードします。 qpid-tool

ACL をリロードするには、十分な権限を持つ **qpid-tool** アカウントで使用する必要があります。

1. **qpid-tool** 以下を開始します。

```
$ qpid-tool admin/mysecretpassword@mybroker:5672
Management Tool for QPID
qpid:
```

2. ACL リストをチェックして、オブジェクト ID を取得します。

```
qpid: list acl
Object Summary:
   ID   Created   Destroyed  Index
   ================================
   103  12:57:41  -          116
```

3. 必要に応じて、ACL を確認できます。

```
qpid: show 103
Object of type: org.apache.qpid.acl:acl:_data(23510fc1-dc51-a952-39c2-e18475c1677e)
   Attribute             103
   ================================================
   brokerRef              116
   policyFile             /tmp/reload.acl
   enforcingAcl           True
   transferAcl            False
   lastAclLoad            Tue Oct 30 12:57:41 2012
   maxConnectionsPerIp    0
   maxConnectionsPerUser  0
   maxQueuesPerUser       0
   aclDenyCount           0
   connectionDenyCount    0
   queueQuotaDenyCount    0
```

4. ACL を再読み込みするには、ACL オブジェクトの reload メソッドを呼び出します。

```
qpid: call 103 reloadACLFile
qpid: OK (0) - {}
```

プログラムコードから ACL を再読み込み

QMF メソッドを呼び出すと、ブローカー ACL をランタイム時にリロードできます。

以下のコードは、適切な QMF メソッドを呼び出して ACL をリロードします。

python

```
import qmf.console
qmf = qmf.console.Session()
qmf_broker = qmf.addBroker('localhost:5672')
acl = qmf.getObjects(_class="acl")[0]
result = acl.reloadACLFile()
print result
```

リロード操作が正常に実行されるには、サーバーが ACL を有効にして起動する必要があります。

バグを報告します。

8.3.5. アクセス制御リストの作成

1. ACL ファイルのユーザー ID は<user-id>@<domain> の形式になります。ドメインはブローカーの SASL 設定を使用して設定され、qpidd の domain/realm は使用され、**--realm** デフォルトは「QPID」に設定されています。

2. ACL ファイル内の各行は、ユーザーに特定の権限を付与または拒否します。

   a. ACL ファイルの最後の行がの場合 **acl deny all all**、ACL は 拒否モード を使用し、明示的に許可される権限のみが付与されます。

   ```
   acl allow user@QPID all all
   acl deny all all
   ```

   このサーバーでは、拒否モードがデフォルトです。アクションは実行 **user@QPID** できますが、nobody は使用できません。

   b. ACL ファイルの最後の行がの場合 **acl allow all all**、ACL は 許可モード を使用し、明示的に拒否されるもの以外のすべての権限が付与されます。

   ```
   acl deny user@QPID all all
   acl allow all all
   ```

   このサーバーでは、allow mode がデフォルトになります。ACL は他のすべてのユーザーがアクションの実行を許可しますが、**user@QPID** すべてのパーミッションを拒否します。

3. ACL 処理は、以下の行のいずれかが発生すると終了します。

   ```
   acl allow all all
   ```

   ```
   acl deny all all
   ```

   このステートメントの1 つの行は無視されます。

   ```
   acl allow all all
   acl deny user@QPID all all # This line is ignored !!!
   ```

4. ACL 構文は、特定のアクションに対して詳細なアクセス権限を許可します。

   ```
   acl allow carlt@QPID create exchange name=carl.*
   acl allow fred@QPID create all
   acl allow all consume queue
   acl allow all bind exchange
   acl deny all all
   ```

5. ACL ファイルはユーザーグループを定義して、そのファイルにパーミッションを割り当てることができます。

   ```
   group admin ted@QPID martin@QPID
   acl allow admin create all
   acl deny all all
   ```

バグを報告します。

8.3.6. ACL 構文

ACL ルールは 1 行に指定して、以下の構文に従う必要があります。

```
acl permission {<group-name>|<user-name>|"all"} {action|"all"} [object|"all"] [property=<property-value>]
```

ACL ファイルで、以下の構文規則が適用されます。

- デフォルト（匿名）の交換はを使用して識別され **name=amq.default** ます。
- **#** 文字で始まる行はコメントとみなされ、無視されます。
- 空白のみを含む空の行と行が無視される
- すべてのトークン **name1** は大文字 **Name1** と小文字を **create** 区別します。 **CREATE**
- グループリストは、\ 文字で行を終了することで、以下の行に拡張できます。
- 追加の空白文字 - である。空白文字が複数あり、トークン間および後の空白文字は無視されます。グループおよび ACL 定義は、**group** または、前の空白文字なしのいずれか **acl** で開始する必要があります。
- すべての ACL ルールが 1 行に限定されます。
- ルールは、名前の一致が取得されるまでファイルの上部から解釈されます。ポイント処理が停止します。
- キーワードは、個人、グループ、およびアクションをすべて **all** 一致させます。
- ファイルの最後の行（present の有無に関わらず）は仮定され **acl deny all all** ます。ファイルに存在する場合、その下の行はすべて無視されます。
- 名前とグループ名に **a-z** は、、、**A-Z 0-9**、**-** およびのみを含めることができます。 _
- 使用可能なグループ定義の前に、ルールの前に付ける必要があります。グループとして定義されていない名前は、個々の名前になります。
- ACL ファイルが有効でないと QPID が起動しない
- QMF メソッドを呼び出すと、ランタイム時に ACL ルールをリロードできます。

関連項目

- 「ACL のリロード」

バグを報告します。

8.3.7. ACL 定義リファレンス

以下の表は **permission**、、**action object**、および ACL ルールファイルの使用可能な値 **property** を示しています。

表8.3 ACL ルール： パーミッション

allow	アクションを許可する

allow-log	アクションを許可し、イベントログに記録します。
deny	アクションを拒否します。
deny-log	アクションを拒否し、イベントログでアクションのログを記録します。

表8.4 ACL ルール： アクション

consume	サブスクリプションの作成時に適用
publish	メッセージ転送のメッセージごとに適用され、このルールは最も多くのリソースを消費します。
create	バインディング、キュー、交換、リンクなど、オブジェクトの作成時に適用されます。
access	オブジェクトの読み取りまたはアクセス時に適用されます。
bind	オブジェクトが結合されている場合に適用されます。
unbind	オブジェクトがバインドされていない場合に適用されます。
delete	オブジェクトの削除時に適用
purge	delete と同様に、アクションは複数のオブジェクトに対して実行されます。
update	オブジェクトの更新時に適用されます。

表8.5 ACL ルール： オブジェクト

queue	キュー
exchange	交換
broker	ブローカー
link	フェデレーションまたはブローカー間のリンク
method	管理またはエージェントまたはブローカーメソッド

表8.6 ACL ルール： プロパティー

name	文字列。キュー名や交換名などのオブジェクト名。

durable	ブール値。オブジェクトが永続化されていることを示します。
routingkey	文字列。ルーティングキーを指定します。
autodelete	ブール値。接続が閉じられるとオブジェクトが削除されるかどうかを示します。
exclusive	ブール値。***exclusive*** フラグの存在を示します。
type	文字列。トピック、stickout、xml などのオブジェクトのタイプ
alternate	文字列。交換の名前
queuename	文字列。キューの名前（オブジェクトが他のものである場合のみ使用） ***queue***
schemapackage	文字列。qmf スキーマパッケージ名
schemaclass	文字列。qmf スキーマクラス名
policytype	文字列。キューの制限ポリシー。キュー作成のルールでのみ使用されます。
maxqueuesize	整数。キューの作成が許可されている最大キューサイズ（バイト単位）の最大値。キュー作成のルールでのみ使用されます。
maxqueuecount	整数。キューの作成が許可される最大キュー深さ（メッセージ単位）の最大値。キュー作成のルールでのみ使用されます。

バグを報告します。

8.3.8. ACL を使用したキューのサイズ制限の実施

ACL で最大 キューサイズを適用できます。これにより、管理者は、ユーザーが多数のシステムリソースを消費できるキューを作成できないようにすることができます。

QUEUE ルールの作成には、メモリーキューとディスク上のキューサイズの両方の上限と下限を制限する ACL ルールがあります。

表8.7 キューサイズ ACL ルール

ユーザーオプション	ACL Limit プロパティー	units
qpid.max_size	queuemaxsizelowerlimit	bytes
	queuemaxsizeupperlimit	bytes

ユーザーオプション	ACL Limit プロパティー	units
qpid.max_count	queuemaxcountlowerlimit	メッセージ
	queuemaxcountupperlimit	メッセージ
qpid.max_pages_loaded	pageslowerlimit	Page
	pagesupperlimit	Page
qpid.page_factor	pagefactorlowerlimit	整数（プラットフォーム定義のページサイズの倍数）
	pagefactorupperlimit	整数（プラットフォーム定義のページサイズの倍数）

ACL Limit Properties は、CREATE QUEUE 要求でユーザーがいずれかのオプションを表示すると評価されます。ユーザーのオプションが要求を許可する ACL ルールの制限プロパティー内にない場合、ルールは Deny の結果と一致します。

Deny ルールでは、制限プロパティーは無視されます。

例：

```
# Example of ACL specifying queue size constraints
# Note: for legibility this acl line has been split into multiple lines.
acl allow bob@QPID create queue name=q6 queuemaxsizelowerlimit=500000
                                        queuemaxsizeupperlimit=1000000
                                        queuemaxcountlowerlimit=200
                                        queuemaxcountupperlimit=300
```

以下に示すようにキューが作成されると、これらの制限が再生されます。

C++

```
int main(int argc, char** argv) {
const char* url = argc>1 ? argv[1] : "amqp:tcp:127.0.0.1:5672";
const char* address = argc>2 ? argv[2] :
    "message_queue; “
    “ { create: always, “
    “   node: “
    “   { type: queue, “
    “     x-declare: ”
    “     { arguments: “
    “       { qpid.max_count:101,”
    “         qpid.max_size:1000000”
    “       }”
    “     }”
    “   }”
    “ }";
std::string connectionOptions = argc > 3 ? argv[3] : "";

Connection connection(url, connectionOptions);
```

```
try {
    connection.open();
    Session session = connection.createSession();
    Sender sender = session.createSender(address);
...
```

このキューは、**qpid-config** コマンドでも作成できます。

```
qpid-config add queue --max-queue-size=1000000 --max-queue-count=101
```

ACL ルールが処理されると、アクター、アクション、オブジェクト、およびオブジェクト名がすべて一致することを仮定すると、allow または deny のルールと一致します。ただし、ACL ルールはさらに制限され、500000 <= max_size <= 1000000 および 200 <= max_count <= 300 に制限されます。**queue_option max_count** は 101 であるため、サイズ制限は違反され（低すぎる）、許可ルールが拒否の決定と共に返されます。

上限 と 下限の **両方** を設定することは必須ではありません。低い制限のみを設定するか、または上限のみを設定できます。

バグを報告します。

8.3.9. リソースクォータオプション

--max-connections Broker オプションで制限できる接続の最大数。

表8.8 リソースクォータオプション

オプション	description	デフォルト値
--max-connections *N*	ブローカーへの同時接続の合計。	500
--max-negotiate-time *N*	初期プロトコルネゴシエーションが成功する期間。これにより、クライアントが正しく動作していないことでリソース不足や、接続の完了を妨げる一時的なネットワーク問題の発生を防ぎます。	500

注記

- **--max-connections** qpid コア制限で、ACL が有効かどうかに関わらず強制されます。
- **--max-connections** Broker ごとに強制されます。N ノードのクラスターでは、すべてのブローカーが最大接続を 20 に設定する場合、クラスターの許可されている接続の合計数は N*20 になります。

ACL ベースのクォータ
ACL ベースのクォータを有効にするには、ACL ファイルを読み込む必要があります。

表8.9 ACL コマンドラインオプション

オプション	description	デフォルト値
--acl-file *FILE (policy.acl)*	読み込むポリシーファイル（data dir から読み込まれる）。	

ACL ファイルが読み込まれると、以下の ACL オプションをコマンドラインで指定して、リソースクォータを強制できます。

表8.10 ACL ベースのリソースクォータオプション

オプション	description	デフォルト値
--connection-limit-per-user *N*	ユーザーごとに許容される最大接続数。0 は無制限を意味します。	0
--connection-limit-per-ip *N*	ホストの IP アドレスごとに許可される最大接続数。0 は無制限を意味します。	0
--max-queues-per-user *N*	個々のユーザーが作成した同時キューの合計	0

注記

- *クラスターシステムでは、実際の接続数が、クラスター内のメンバーノードの数よりも **N** 1 つ小さくなる可能性があります。たとえば、5 ノードクラスターで、20 個の接続が設定されている場合、制限が行われる前に実際の接続数が 24 に達する可能性があります。*
- *クラスター接続は、確立されると接続制限に対してチェックされます。空き接続が利用できない場合、クラスター接続は拒否されます。ただし、確立後、クラスター接続は接続を消費しません。*
- *使用できる値 **N** は 0..65535 です。*
- *これらの制限は **クラスター** ごとに適用されます。*
- *値がゼロ(0)の場合は、そのオプションの制限チェックが無効になります。*
- *ユーザーごとの接続は、認証されたユーザー名によって識別されます。*
- *ノードごとの接続は、管理接続インデックスである **<broker-ip><broker-port>-<client-ip><client-port>** タプルによって識別されます。*
    - *このスキームホストシステムは、IPv4、**localhost** IPv4、**127.0.0.1** IPv **::1** 6 などの複数の名前で識別でき、名前ごとに別の接続セットが許可されます。*
    - *IP 接続ごとの接続は、接続で提供されたユーザーのクレデンシャルに関係なくカウントされます。個々のユーザーは 20 の接続が許可されますが、クライアントホストに 5 つの接続制限がある場合は、そのシステムから接続できるのは 5 回のみです。*

バグを報告します。

8.3.10. ユーザーごとのリソースクォータ

リソースクォータは、詳細な制御のために ACL を使用してユーザーごとに設定できます。

ルール構文

ユーザーごとの ACL ルール構文は以下のとおりです。

```
quota connections|queues value <group-name-list>|<user-name-list> [ <group-name-list>|<user-name-list>]
```

接続クォータ

接続クォータはコマンドラインスイッチ '**--connection-limit-per-user N**' と併用して機能し、ユーザーを一定数の同時接続に制限します。

- コマンドラインスイッチ '**--connection-limit-per-user**' がなく、ACL ファイルに「quota connections」ルールがない場合、接続制限は強制されません。
- コマンドラインスイッチ '**--connection-limit-per-user**' が存在する場合は、擬似ユーザー 'all' の初期値を割り当てます。
- ACL ファイルに擬似ユーザーの「all」のクォータを指定すると、ACL ファイルで名前が付けられていないすべてのユーザーに適用されます。
- ルールファイルが処理されるため、ユーザーの接続クォータは順番に登録されます。ユーザーは任意の数の接続クォータ値を割り当てることはできますが、最終的な値のみが保持され、適用されます。
- グループの接続クォータは、「quota connections」行が処理される際に、グループ内の各ユーザーに対して接続クォータとして適用されます。
- クォータの値は 0 から 65530 までになります。値がゼロ(0)の場合は接続を拒否します。

キュークォータ

キュークォータはコマンドラインスイッチ '**--max-queues-per-user N**' と併用して機能し、ユーザーを一定数の同時キューに制限します。

- コマンドラインスイッチ '**--max-queues-per-user**' がなく、ACL ファイルに「quota queues」ルールがない場合、キューの制限は強制されません。
- コマンドラインスイッチ '**--max-queues-per-user**' が存在する場合は、擬似ユーザー 'all' の初期値を割り当てます。
- ACL ファイルに擬似ユーザーの「all」のクォータを指定すると、ACL ファイルで名前が付けられていないすべてのユーザーに適用されます。
- ルールファイルが処理されるため、ユーザーのキュークォータは順番に登録されます。ユーザーは任意の数のキュークォータ値を割り当てることができますが、最後の値のみが保持され、適用されます。
- グループのキュークォータは、'quota queues' 行が処理される際に、グループ内の各ユーザーのキュークォータとして適用されます。
- クォータの値は 0 から 65530 までになります。値がゼロ(0)の場合は、キューの作成アクションを拒否します。

バグを報告します。

8.3.11. ホスト名による接続制限

0.30 C++ Broker ACL モジュールは、ユーザーが接続できる TCP/IP ホストの許可および拒否リストを作成する機能を追加します。ルールは、以下の形式を受け入れます。

```
acl allow user create connection host=host1
acl allow user create connection host=host1,host2
acl deny  user create connection host=all
```

フォームの使用は、単一のホストを **host=host1** 指定します。単一のホストでは、複数の TCP/IP アドレスに名前が解決される可能性があります。たとえば、localhost は 127.0.0.1 と ::1 の両方に解決され、その他の多くのアドレスにも解決されます。このホストに関連付けられたアドレスからの接続がルールと一致し、それに応じて接続が許可されまたは拒否されます。

フォームを使用すると、TCP/IP アドレスの範囲が **host=host1,host2** 指定されます。ホスト範囲では、各ホストが単一の TCP/IP アドレスに解決し、2 番目のアドレスは 1 番目のアドレスよりも数値が大きい必要があります。host >= host1 および host <= host2 がルールに一致し、接続が許可されるか、拒否されるホストからの接続。

フォームを使用すると、すべての TCP/IP アドレスが **host=all** 指定されます。任意のホストからの接続がルールと一致し、それに応じて接続が許可されるか、または拒否されます。

接続の拒否は、着信 TCP/IP 接続にのみ適用されます。その他のソケットタイプは、範囲チェックでの対象または拒否されません。

接続作成ルールは、3 つのカテゴリーに分類されます。

1. user = all、host != all

   これらはグローバルルールを定義し、特定のユーザールールの前に適用されます。これらのルールを使用して、AMPQ プロトコルが実行され、ユーザー名がネゴシエートされる前に接続を拒否することができます。

2. user != all、host = any legal host or 'all'

   これらは、ユーザールールを定義します。これらのルールはグローバルルールの後に適用され、AMQP プロトコルでユーザー ID がネゴシエートされた後に適用されます。

3. user = all、host = all

   このルールは、他のルールがマッチしない場合に何を実行するかを定義します。デフォルト値は「ALLOW」です。このタイプのルールは 1 つだけ定義できます。

以下の例は、この機能の使用方法を示しています。

例8.1 ホスト名による接続制限

```
group admins alice bob chuck
group Company1 c1_usera c1_userb
group Company2 c2_userx c2_usery c2_userz
acl allow admins   create connection host=localhost
acl allow admins   create connection host=10.0.0.0,10.255.255.255
acl allow admins   create connection host=192.168.0.0,192.168.255.255
acl allow admins   create connection host=[fc00::],[fc00::ff]
```

```
acl allow Company1 create connection host=company1.com
acl deny  Company1 create connection host=all
acl allow Company2 create connection host=company2.com
acl deny  Company2 create connection host=all
```

この例では、管理者は、ローカルホストまたは 10.0.0.0/24、192.168.0.0/16、および fc00::/7 サブネット上のシステムから接続することができます。Company1 ユーザーは company1.com からのみ接続でき、Company2 ユーザーは company2.com からしか接続できません。ただし、この例では不具合があります。admins グループには、接続の確立が許可されている特定のホストがありますが、どこでも接続はブロックされません。Company1 グループと Company2 グループが適切にブロックされます。この ACL ファイルは以下のように書き換えることができます。

```
group admins alice bob chuck
group Company1 c1_usera c1_userb
group Company2 c2_userx c2_usery c2_userz
acl allow admins   create connection host=localhost
acl allow admins   create connection host=10.0.0.0,10.255.255.255
acl allow admins   create connection host=192.168.0.0,192.168.255.255
acl allow admins   create connection host=[fc00::],[fc00::ff]
acl allow Company1 create connection host=company1.com
acl allow Company2 create connection host=company2.com
acl deny  all      create connection host=all
```

一覧表示された管理者は、許可されたホスト以外から接続がブロックされるようになりました。

バグを報告します。

8.3.12. ルーティングキーワイルドカード

Topic Exchange 一致ロジック は、**routingkey** プロパティーを含む ACL ルールに使用されます。これらのルールには、以下が含まれます。

- bind exchange <name> routingkey=X
- unbind exchange <name> routingkey=X
- publish exchange <name> routingkey=X

この **routingkey** プロパティーは、Topic Exchange に一致するものと同じロジックを使用して一致するようになりました。これにより、管理者は、**routingkey** 値が使用される namespace にマップする柔軟な用語でユーザー制限を表すことができます。

ワイルドカード一致とトピック交換
バインディングキーでは、任意の数のピリオド用語に **#** 一致し、1 つの用語に * 一致します。

そのため、のバインディングキー **#.news** は **usa.news** およびなどのサブジェクトとメッセージに一致し **germany.europe.news**、のバインディングキーはサブジェクトとメッセージに ***.news** 一致しますが **usa.news**、バインディングキーはメッセージとサブジェクトに一致しません **germany.europe.news**。

例：
次の ACL ルール。

```
acl allow-log uHash1@COMPANY publish exchange name=X routingkey=a.#.b
acl deny all all
```

*ユーザーが X を交換するために **uHash1@COMPANY** パブリッシュすると、以下の結果を生成します。*

表8.11

X を交換する publish の Routingkey	結果
a.b	allow-log
a.x.b	allow-log
a..x.y.zz.b	allow-log
a.b.	deny
q.x.b	deny

バグを報告します。

8.3.13. ルーティングキーワイルドカードの例

以下の表は、ACL ルールとメッセージヘッダーのルーティングキーワイルドカード間の対話を示しています。テーブルの上部には、ワイルドカードを使用したルーティングキーがあります。左側の下はメッセージヘッダーです。

*この文字は、指定したルーティングキーを持つメッセージを許可する ACL ルールが指定されたヘッダーを持つメッセージがルーティングされ **ない** ことを **X** 示します。「Routed」は、指定 さ れたルーティングキーを持つメッセージを許可する ACL ルールによって指定のヘッダーを持つメッセージがルーティングされることを示します。*

表8.12 ルーティングキー、メッセージヘッダー、および結果ルーティング。

ルーティングキー ->	a.#	#.e	a.#.e	a.#.c.#.e	#.c.#	#
メッセージヘッダー：						
ax	X	X	X	X	X	routed
a.x	routed	X	X	X	X	routed
ex	X	X	X	X	X	routed
e.x	X	X	X	X	X	routed
ae	X	X	X	X	X	routed

ルーティングキー ->	a.#	#.e	a.#.e	a.#.c.#.e	#.c.#	#
a.e	routed	routed	routed	X	X	routed
a..e	routed	routed	routed	X	X	routed
a.x.e	routed	routed	routed	X	X	routed
a.c.e	routed	routed	routed	routed	routed	routed
a..c..e	routed	routed	routed	routed	routed	routed
a.b.c.d.e	routed	routed	routed	routed	routed	routed
a.b.x.c.d.y.e	routed	routed	routed	routed	routed	routed
a.#	routed	X	X	X	X	routed
#.e	X	routed	X	X	X	routed
a.#.e	routed	routed	routed	X	X	routed
a.#.c.#.e	routed	routed	routed	routed	routed	routed
#.c.#	X	X	X	X	routed	routed
#	X	X	X	X	X	routed

バグを報告します。

8.3.14. ユーザー名およびドメイン名シンボルの置換

MRG 3 では、ユーザー名とドメイン名置換変数の単純なセットを使用できます。これにより、管理者はプライベートリソースや共有リソースを簡単に定義できます。

Acl ファイルで、シンボルの置換は、プロパティー値にテキストが提供される場所でも許可されます。

*以下の表では、認証されたユーザーに置換キーワード **bob.user@QPID.COM** が拡張されています。*

表8.13

keyword	拡張
${userdomain}	bob_user_QPID_COM
${user}	bob_user
${domain}	QPID_COM

元の名前にはピリオド「.」があり、シンボル「@」文字はアンダースコア「_」に変換されます。これにより、ACL ファイルの Routingkey で置換キーワードが使用されたときに置換が機能します。

ルーティングキーでのシンボル置換およびワイルドカードの使用

* シンボルは、ルーティングキーの単一のフィールド内の任意の数の文字に対してワイルドカード一致を使用できます。例：

```
acl allow user_group publish exchange name=users routingkey=${user}-delivery-*
```

ルーティングキーの仕様で使用されると、「#」シンボルは、任意の数のドット付きサブジェクト名フィールドに置き換えられます。ユーザーおよびドメインシンボルの置換は、ルーティングキーのワイルドカードシンボルと組み合わせることもできます。以下 **#** に例を示します。

```
acl allow user_group bind exchange name=${user}-work2 routingkey=news.#.${user}
```

ルーティングキーのワイルドカードの ACL マッチング

${user}** またはのいずれか **${userdomain} に一致する前に ACL 処理が一致し **${domain}**ます。多くの場合、ACL 処理は同等の処理 **${userdomain}** として使用され **${user}_${domain}**、この 2 つの形式は交換可能なものとして使用できます。例外は、ルーティングキー内にワイルドカードを指定するルールです。この場合、組み合わせ **${user}_${domain}** は一致しないので、フォーム **${userdomain}** を使用してください。

たとえば、以下のルールはマッチしません。

```
acl allow all publish exchange name=X routingkey=${user}_${domain}.c
```

たとえば、ACL プロセッサーが routingkey を検索するため、ルールはマッチしません **${userdomain}.c**。

ACL 記号置換の例

管理者は、すべてのユーザーがプライベート交換、プライベートキュー、およびそれらの間のプライベートバインディングを作成できるようにする ACL ルールファイルをセットアップできます。この例では、ユーザーはプライベートのバックアップ交換、キュー、およびバインディングを作成することもできます。これにより、ユーザーの交換、キュー、バインディングの作成に制限が提供され、各ユーザーがこれらのリソースに排他的にアクセスできるようになります。

```
#
# Create primary queue and exchange:
acl allow all create  queue    name=${user}-work alternate=${user}-work2
acl deny  all create  queue    name=${user}-work alternate=*
acl allow all create  queue    name=${user}-work
acl allow all create  exchange name=${user}-work alternate=${user}-work2
acl deny  all create  exchange name=${user}-work alternate=*
acl allow all create  exchange name=${user}-work
#
# Create backup queue and exchange
#
acl deny  all create  queue    name=${user}-work2 alternate=*
acl allow all create  queue    name=${user}-work2
acl deny  all create  exchange name=${user}-work2 alternate=*
acl allow all create  exchange name=${user}-work2
#
# Bind/unbind primary exchange
#
acl allow all bind   exchange name=${user}-work routingkey=${user} queuename=${user}-work
```

```
acl allow all unbind exchange name=${user}-work routingkey=${user} queuename=${user}-work
#
# Bind/unbind backup exchange
#
acl allow all bind   exchange name=${user}-work2 routingkey=${user} queuename=${user}-work2
acl allow all unbind exchange name=${user}-work2 routingkey=${user} queuename=${user}-work2
#

# deny mode
#
acl deny all all
```

バグを報告します。

8.3.15. ACL 定義の例

ほとんどの ACL ファイルは、グループを定義することで始まります。

```
group admin ted@QPID martin@QPID
group user-consume martin@QPID ted@QPID
group group2 kim@QPID user-consume rob@QPID
group publisher group2 \
tom@QPID andrew@QPID debbie@QPID
```

ACL ファイルのルールにより、ユーザーまたはグループに特定のパーミッションを付与または拒否します。

```
acl allow carlt@QPID create exchange name=carl.*
acl allow rob@QPID create queue
acl allow guest@QPID bind exchange name=amq.topic routingkey=stocks.rht.#
acl allow user-consume create queue name=tmp.*

acl allow publisher publish all durable=false
acl allow publisher create queue name=RequestQueue
acl allow consumer consume queue durable=true
acl allow fred@QPID create all
acl allow bob@QPID all queue
acl allow admin all
acl allow all consume queue
acl allow all bind exchange
acl deny all all
```

上記の例では、最後の行は許可されていないすべての承認を **acl deny all all**拒否します。デフォルトはデフォルトですが、明確化するために最後の行に明示的に含めると便利です。デフォルトですべての権限を付与する場合は、最後の行 **acl allow all all** で指定できます。

セキュリティー違反 **guest** を引き起こす可能性のある QMF 管理方法にアクセスしてログに記録することはできません。

```
group allUsers guest@QPID
....
acl deny-log allUsers create link
acl deny-log allUsers access method name=connect
acl deny-log allUsers access method name=echo
acl allow all all
```

- *バグを報告します。*

第9章 高可用性

9.1. クラスタリング（高可用性）

9.1.1. MRG 3 でのクラスタリングの変更

MRG 3 は **cluster** モジュールを新しい **ha** モジュールに置き換えます。このモジュールは、高可用性のためのアクティブ/パッシブクラスタリング機能を提供します。

MRG 2 の **cluster** モジュールは active-active: クライアントはクラスターの任意のブローカーに接続できます。新しい **ha** モジュールはアクティブ/パッシブです。1 つのブローカーが プライマリー として動作し、他のブローカーは バックアップ として機能します。プライマリーのみがクライアント接続を許可します。クライアントがバックアップブローカーへの接続を試みると、接続は中止され、プライマリーに接続するまでクライアントは失敗します。

新しい **ha** モジュールは、仮想 IP アドレス もサポートします。クライアントは、自動的にプライマリーブローカーにルーティングされる単一の IP アドレスで設定できます。これは推奨される設定です。

フェイルオーバーの交換は、後方互換性を確保するために提供されます。新しい実装では、代わりに仮想 IP アドレスを使用する必要があります。

マルチスレッドパフォーマンスの向上

MRG 2 では、クラスター化されたブローカーは単一の CPU スレッドのみを使用します。一部のユーザーは、1 台のマシンで複数のクラスター化されたブローカーを実行して、複数のコアを利用することでこれを回避します。

MRG 3 では、クラスター化されたブローカーが複数のスレッドを使用し、マルチコア CPU を活用できるようになりました。

バグを報告します。

9.1.2. アクティブ/パッシブメッセージングクラスター

High Availability(HA) モジュールは、フォールトトレランスメッセージ配信を提供するために、アクティブ/パッシブ で ホットスタンバイ のメッセージングクラスターを提供します。

アクティブ/パッシブのクラスターでは、プライマリーと呼ばれるブローカーが1 つだけアクティブで、一度にクライアントを提供します。他のブローカーはバックアップとして構築されます。プライマリーの変更はすべてのバックアップに複製されるため、常に最新または「ホット」になります。バックアップブローカーは、クライアントがプライマリーのみに接続する要件を実施するために、クライアント接続の試行を拒否します。

プライマリーに失敗すると、バックアップのいずれかが新しいプライマリーとして引き継ぐようにプロモートされます。クライアントは、新しいプライマリーに自動的にフェイルオーバーします。複数のバックアップがある場合には、他のバックアップもフェイルオーバーして、新しいプライマリーのバックアップになります。

このアプローチは、障害を検出 **rgmanager** するために外部のクラスターリソースマネージャーに依存し、新しいプライマリーパーティションを選択し、ネットワークパーティションを処理します。

バグを報告します。

9.1.3. メッセージ損失の回避

メッセージが失われないように、プライマリーブローカーは、メッセージが複製され、すべてのバックアップブローカーによって承認されたり、プライマリーキューから消費された場合に、クライアントから受信したメッセージの確認応答を送信します。

これにより、確認応答されたメッセージはすべて安全になります。これらのメッセージは、すべてのバックアップブローカーに消費またはバックアップされています。レプリケート前に消費されるメッセージはレプリケートする必要はありません。これにより、アクティブなコンシューマーでキューを複製する場合にワークロードが削減されます。

クライアントは、プライマリーによって承認されるまで、クライアントの再生バッファーで未承認メッセージを保持します。プライマリーに失敗すると、クライアントは新しいプライマリーにフェイルオーバーし、承認されていないメッセージをすべて再送信します。

プライマリーがクラッシュすると、新しいプライマリーとして引き継がれるバックアップで確認済みのメッセージがすべて利用できます。未承認のメッセージはクライアントによって再送信されます。そのため、メッセージは失われません。

これは、メッセージが複製される可能性があることを意味します。障害が発生すると、バックアップによってメッセージが受信され、クライアントによって新しいプライマリーになり、再送信されます。アプリケーションは、重複を特定して削除するために手順を実行する必要があります。

フェイルオーバー後に新しいプライマリーがプロモートされると、最初に「リカバリー」モードで実行されます。このモードでは、前のプライマリーに接続されたすべてのバックアップの代わりにメッセージの承認が遅延します。これにより、バックアップが接続およびキャッチするまで、新しいプライマリーの障害からメッセージを保護します。

すべてのメッセージをバックアップブローカーにレプリケートする必要があるわけではありません。メッセージがバックアップに複製される前に通常のクライアントによって消費され、確認応答された場合は、複製する必要がありません。

バグを報告します。

9.1.4. ha Broker のステータス

join

プライマリーに接続されていない新規ブローカーの初期ステータス。

catch-up

プライマリーに接続され、キューとメッセージでキャッチされるバックアップブローカー。

Ready

完全にキャッチアップでき、プライマリーとして引き継ぐ準備が整っているバックアップブローカー。

recovering

新規に昇格したプライマリーは、バックアップの接続とキャッチを待機します。

Active

すべてのバックアップが接続され、キャッチアップされたアクティブなプライマリーブローカー。

バグを報告します。

9.1.5. MRG 3 の HA の制限

MRG 3 では、HA サポートにいくつかの制限があります。

- HA レプリケーションは 65434 キューに制限されます。
- **qpidd-primary** サービスを手動で再配置して、qpid ブローカーが ready 状態ではないノードには実行できません（停止またはキャッチアップまたは参加状態のいずれか）。
- クラスターの順序付けされた failover-domains('**ordered=1**' in **cluster.conf**) によるフェイルバックにより、特定の条件下で無限なフェイルオーバーループが発生する可能性があります。これを回避するには、で **nofailback=1** 指定されている failover-domains を使用し **cluster.conf**ます。
- ローカルトランザクションの変更はアトミックにレプリケートされます。ローカルトランザクション中にプライマリーがクラッシュした場合、データが失われることはありません。分散トランザクションは、HA クラスターによってまだサポートされていません。
- 設定変更（キューの作成または削除、交換、バインディング）は非同期的にレプリケートされます。変更に使用される管理ツールは、プライマリーで完了すると変更が完全に考慮されますが、まだすべてのバックアップに複製されないことがあります。
- プライマリーへのフェデレーションリンクが正しくフェイルオーバーされません。プライマリーからのフェデレーションされたリンクはフェールオーバーされ、それらは新しいプライマリーに再度接続されません。qpidd-primary start up スクリプトを、プライマリーのプロモート時にフェデレーションリンクを再作成するスクリプトに置き換えて、この問題を回避できます。

バグを報告します。

9.1.6. ブローカー HA オプション

qpid-ha Broker ユーティリティーのオプション

ha-cluster yes|no

ブローカーがクラスターに参加できるようにするには、「yes」に設定します。

ha-queue-replication yes|no

クラスターに参加せずに、特定のキューのレプリケーションを有効にします。

ha-brokers-url URL

相互に接続するためにクラスターブローカーによって使用される URL。URL には、仮想 IP アドレスではなく、ブローカーアドレスのコンマ区切りリストを含める必要があります。

URL の完全な形式は、この文法で指定されます。

```
url = ["amqp:"][ user ["/" password] "@" ] addr ("," addr)*
addr = tcp_addr / rmda_addr / ssl_addr / ...
tcp_addr = ["tcp:"] host [":" port]
rdma_addr = "rdma:" host [":" port]
ssl_addr = "ssl:" host [":" port]'>
```

ha-public-url URL

このオプションは、**amq.failover** 交換を使用している場合に限り後方互換性を確保するために必要です。この交換は廃止され、代わりに仮想 IP アドレスを使用することが推奨されます。

設定されている場合、この URL は **amq.failover** exchange によってアドバタイズされ、broker オプションを上書きし **known-hosts-url**ます。

ha-replicate VALUE

キューと交換がデフォルトで複製されるかどうかを指定します。value は **none**、、**configuration**、のいずれかになります **all**。

ha-username USER, ha-password PASS, Ha-mechanism MECHANISM

HA ブローカーが相互に接続するために使用される認証設定。承認を使用している場合は、このユーザーにすべてのパーミッションが必要です。

ha-backup-timeout SECONDS

リカバリーするプライマリーが、予想されるバックアップが接続され、準備状態になるまで待機する最大時間。

SECONDS で指定した値は、秒数（例：）となります。10分の1 分の10 分の "0.1"。明示的な単位（例：10 秒）、10ms（ミリ秒）、10us（マイクロ秒）、10ns（ナノ秒）もあります。

link-maintenance-interval SECONDS

ha はフェデレーションリンクを使用して、バックアップからプライマリーに接続します。バックアップブローカーはこの間隔でプライマリーへのリンクを確認し、必要な場合は再度接続します。デフォルトは 2 秒です。高速フェイルオーバーの場合（例：0.1 秒）以下を設定できます。設定が低すぎると、バックアップでリンクチェックが過剰になります。

link-heartbeat-interval SECONDS

フェデレーションされたリンクのストリームを待機する秒数、またはブローカーステータスチェックのタイムアウトを待つ秒数。

デフォルトは 120 秒です。HA シナリオで高速なフェイルオーバー検出を可能にするため、小さい値（10 秒など）を指定します。値が低すぎると、低速なブローカーは失敗とみなされ、強制終了されます。

この間隔でハートビートを受信しないと、プライマリーはそのバックアップの有効期限（たとえば、バックアップがハングまたはパーティションに分割される場合）を考慮します。

rgmanager がハングまたはパーティション指定されたブローカーを検出するには、この間隔までかかる場合があります。プライマリーはこの間隔を最大 2 回実行して、ハングアップまたはパーティション化されたバックアップを検出できます。この時間帯は、メッセージを送信するクライアントも遅延する可能性があります。

HA クラスターを設定するには **ha-cluster**、およびを設定する必要があり **ha-brokers-url**ます。

関連項目

- *「キューと交換のレプリケーションの制御」*

バグを報告します。

9.1.7. クラスタリングのファイアウォール設定

以下のポートはクラスター化されたシステムで使用され、ファイアウォールで開いている必要があります。

表9.1 クラスター化されたシステムが使用するポート

ポート	protocol	コンポーネント
5404	UDP	CMAN
5405	UDP	CMAN
5405	TCP	luci
8084	TCP	luci
11111	TCP	Ricci
14567	TCP	gnbd
16851	TCP	modclusterd
21064	TCP	dlm
50006	TCP	ccsd
50007	UDP	ccsd
50008	TCP	ccsd
50009	TCP	ccsd

*以下の **iptables** コマンドは、root 権限で実行すると、これらのポートでの通信を許可するようにシステムを設定します。*

```
iptables -I INPUT -p udp -m udp --dport 5405  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 5405  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 8084  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 11111  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 14567  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 16851  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 21064  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50006  -j ACCEPT
iptables -I INPUT -p udp -m udp --dport 50007  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50008  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50009  -j ACCEPT
service iptables save
service iptables restart
```

バグを報告します。

9.1.8. クラスタリングの ACL 要件

クラスタリングには、ブローカー間のフェデレーションリンクが必要です。ブローカーがある場合 **auth=yes**、すべてのフェデレーションリンクはデフォルトで許可されません。HA クラスタリングで使用されるフェデレーションを許可するには、以下の ACL ルールが必要です。

```
acl allow <ha-username> all all
```

バグを報告します。

9.1.9. Cluster Resource Manager(rgmanager)

Broker のフェールオーバーはリソースマネージャーによって管理され **rgmanager** ます。

リソースマネージャーは、クラスターの各ノードで **qpidd** ブローカーを起動します。その後、リソースマネージャーはブローカーの 1 つをプライマリーにプロモートします。他のブローカーは、**ha-brokers-url** 設定オプションで提供される URL を使用して、プライマリーにバックアップとして接続します。

接続されると、バックアップブローカーの状態がプライマリーと同期されます。バックアップが同期される場合（「hot」）は、プライマリーで障害が発生した場合に引き継げる準備が整います。バックアップブローカーは同期を維持するために、プライマリーから更新を継続的に受信します。

プライマリーに失敗すると、バックアップブローカーはフェールオーバーモードになります。リソースマネージャーは障害を検出し、新しいプライマリーとなるバックアップのいずれかをプロモートします。他のバックアップは新しいプライマリーに接続し、状態と同期します。

また、リソースマネージャーは、ネットワークパーティションから生じるスプリットブレイン状態からクラスターを保護します。ネットワークパーティションは、クラスターを相互に認識できない 2 つのサブグループに分割します。クォーラムの投票アルゴリズムは、クォーラムサブグループのノードを無効にします。

バグを報告します。

9.1.10. HA クラスターコンポーネントのインストール

手順9.1 qpidd HA コンポーネントのインストール手順

1. 端末を開き、スーパーユーザーアカウントに切り替えます。
2. **yum install qpid-cpp-server-ha** を実行して、必要なすべてのコンポーネントをインストールします。

手順9.2 Red Hat Linux HA クラスターコンポーネントのインストール手順

1. 「RHEL Server High Availability」チャンネルにシステムをサブスクライブします。
2. 端末を開き、スーパーユーザーアカウントに切り替えます。
3. **yum install -y rgmanager ccs** を実行して、必要なすべてのコンポーネントをインストールします。
4. HA クラスタリングを開始する前に、Network Manager を無効にします。ha クラスタリングは、Network Manager の開始または有効化で正常に機能しません。

```
# chkconfig NetworkManager off
```

5. rgmanager、cman、および ricci サービスをアクティベートします。

```
# chkconfig rgmanager on
# chkconfig cman on
# chkconfig ricci on
```

6. qpidd サービスを非アクティブにします。

```
# chkconfig qpidd off
```

rgmanager **chkconfig** が qpidd を起動し、停止するため、**qpidd** サービスは **オフになっ** ている必要があります。通常のシステム init プロセスも qpidd の起動および停止を試みると、rgmanager が qpidd プロセスの追跡を失う可能性があります。

qpidd をオフにしないと、qpidd プロセスが実際に実行中の場合に停止される qpidd サービスが **clustat** 表示されます。この場合、qpidd ログには以下のようなエラーが表示されます。

```
critical Unexpected error: Daemon startup failed: Cannot lock /var/lib/qpidd/lock: Resource temporarily unavailable
```

バグを報告します。

9.1.11. 仮想 IP アドレス

Qpid HA クラスタリングは仮想 IP アドレスをサポートします。仮想 IP アドレスは、プロモーションが発生するたびにクラスター内のプライマリーノードに移動される IP アドレスです。リソースマネージャーは、このアドレスをクラスターのプライマリーノードに関連付け、障害時に新しいプライマリーに再配置します。これは、リストではなく単一の IP アドレスを公開することができるため、設定が簡素化されます。

仮想 IP アドレスは各ノードで正しく設定する必要があります。クラスターマネージャーの唯一のジョブは、仮想 IP アドレスを起動および停止する必要があります。ノードの物理アダプターと同じネットワーク上の仮想 IP アドレスを使用する場合、ノードのプライマリーにプロモートされる際に、クラスターマネージャーはアドレスをアダプターにバインドできます。また、それ以上の設定は必要ありません。仮想 IP アドレスが物理アダプターとは別のネットワークにある場合は、2 番目の物理アダプターまたは仮想ネットワークインターフェースを仮想 IP に設定する必要があります。

関連項目

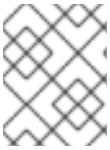
- 「HA クラスターの設定」

バグを報告します。

9.1.12. HA クラスターの設定

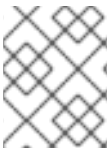
MRG メッセージングブローカーは、を使用してクラスター化でき、アクティブ/パッシブでホットスタンバイの qpiddby qpidd HA クラスターを **cman rgmanager** 作成できます。基盤 **cman** となるクラスタリング技術に関する詳細は **rgmanager**、『『Red Hat Enterprise Linux クラスター管理ガイド』を参照してください』。

ha クラスタリングは、**/etc/cluster/cluster.conf** ファイルを使用して **cman** およびを設定し **rgmanager**ます。



注記

*HA が操作するには、ブローカー管理が必要です。これは、デフォルトで有効になっています。オプションを「no」に **mgmt-enable** することはできません。*



注記

誤ったセキュリティー設定は、開始時の問題の一般的な原因です。を参照してください「security」。

*このツールは、**cluster.conf** ファイルを設定する高レベルのユーザーフレンドリーメカニズムを **ccs** 提供します。これは、クラスターを設定するのに推奨される方法です。この **ccs** ツールの使用方法の詳細は『『Red Hat Enterprise Linux クラスター管理ガイド』』を参照してください。*

*以下の手順では **node1**、という名前の 3 つのノードで構成されるクラスターの例 **ccs** を作成 **node2** し **node3**ます。**root** ユーザーとして以下のコマンドを実行します。*

1. ***ricci** サービスを起動します。*

```
service ricci start
```

2. *以前に **ricci** パスワードを設定していない場合は、ここで設定します。*

```
passwd ricci
```

3. *新規クラスターを作成します。*

```
ccs -h localhost --createcluster qpid-test
```

4. *3 つのノードを追加します。*

```
ccs -h localhost --addnode node1.example.com
ccs -h localhost --addnode node2.example.com
ccs -h localhost --addnode node3.example.com
```

5. *をそれぞれに追加 **failoverdomain** します。*

```
ccs -h localhost --addfailoverdomain node1-domain restricted
ccs -h localhost --addfailoverdomain node2-domain restricted
ccs -h localhost --addfailoverdomain node3-domain restricted
```

6. *をそれぞれに追加 **failoverdomainnode** します。*

```
ccs -h localhost --addfailoverdomainnode node1-domain node1.example.com
ccs -h localhost --addfailoverdomainnode node2-domain node2.example.com
ccs -h localhost --addfailoverdomainnode node3-domain node3.example.com
```

7. *スクリプトを追加します。*

```
ccs -h localhost --addresource script name=qpidd file=/etc/init.d/qpidd
ccs -h localhost --addresource script name=qpidd-primary file=/etc/init.d/qpidd-primary
```

8. *仮想 IP アドレスを追加します。*

```
ccs -h localhost --addresource ip address=20.0.20.200 monitor_link=1
```

9. *各ノードに **qpidd** サービスを追加します。失敗した場合は再起動する必要があります。*

```
ccs -h host --addservice node1-qpidd-service domain=node1-domain recovery=restart
ccs -h localhost --addsubservice node1-qpidd-service script ref=qpidd
ccs -h localhost --addservice node2-qpidd-service domain=node2-domain recovery=restart
ccs -h localhost --addsubservice node2-qpidd-service script ref=qpidd
ccs -h localhost --addservice node3-qpidd-service domain=node3-domain recovery=restart
ccs -h localhost --addsubservice node3-qpidd-service script ref=qpidd
```

10. *プライマリーサービスを追加し **qpidd** ます。これは一度に1 つのノードでのみ実行され、任意のノードで実行できます。*

```
ccs --host localhost --addservice qpidd-primary-service recovery=relocate autostart=1
exclusive=0
ccs -h localhost --addsubservice qpidd-primary-service script ref=qpidd-primary
ccs -h localhost --addsubservice qpidd-primary-service ip ref=20.0.20.200
```

*以下は、直前の手順で作成されたコメント化されたバージョンの **/etc/cluster/cluster.conf** ファイルです。*

```
<?xml version="1.0"?>
<!--
This is an example of a cluster.conf file to run qpidd HA under rgmanager.
This example configures a 3 node cluster, with nodes named node1, node2 and node3.

NOTE: fencing is not shown, you must configure fencing appropriately for your cluster.
-->

<cluster name="qpid-test" config_version="18">
   <!-- The cluster has 3 nodes. Each has a unique nodeid and one vote
        for quorum. -->
   <clusternodes>
      <clusternode name="node1.example.com" nodeid="1"/>
      <clusternode name="node2.example.com" nodeid="2"/>
      <clusternode name="node3.example.com" nodeid="3"/>
   </clusternodes>

   <!-- Resouce Manager configuration. -->
   <rm>
      <!--
    There is a failoverdomain for each node containing just that node.
    This specifies that the qpidd service should always run on each node.
        -->

      <failoverdomains>
         <failoverdomain name="node1-domain" restricted="1">
            <failoverdomainnode name="node1.example.com"/>
         </failoverdomain>
         <failoverdomain name="node2-domain" restricted="1">
            <failoverdomainnode name="node2.example.com"/>
```

```
        </failoverdomain>
        <failoverdomain name="node3-domain" restricted="1">
            <failoverdomainnode name="node3.example.com"/>
        </failoverdomain>
    </failoverdomains>

    <resources>
        <!-- This script starts a qpidd broker acting as a backup. -->
        <script file="/etc/init.d/qpidd" name="qpidd"/>

        <!-- This script promotes the qpidd broker on this node to primary. -->
        <script file="/etc/init.d/qpidd-primary" name="qpidd-primary"/>

        <!-- This is a virtual IP address on a seprate network for client traffic. -->
        <ip address="20.0.20.200" monitor_link="1"/>
    </resources>

    <!-- There is a qpidd service on each node,
         it should be restarted if it fails. -->

    <service name="node1-qpidd-service" domain="node1-domain" recovery="restart">
        <script ref="qpidd"/>
    </service>
    <service name="node2-qpidd-service" domain="node2-domain" recovery="restart">
        <script ref="qpidd"/>
    </service>
    <service name="node3-qpidd-service" domain="node3-domain"  recovery="restart">
        <script ref="qpidd"/>
    </service>

    <!-- There should always be a single qpidd-primary service,
         it can run on any node. -->

    <service name="qpidd-primary-service" autostart="1" exclusive="0" recovery="relocate">

        <script ref="qpidd-primary"/>
        <!-- The primary has the IP addresses for brokers and clients to connect. -->
        <ip ref="20.0.20.200"/>
    </service>
  </rm>
</cluster>
```

*ノードには、1 つ **failoverdomain** のノードのみが含まれるノードがあります。これにより、qpidd サービスがすべてのノードで常に実行されるように指定します。*

*この **resources** セクションは、**qpidd** サービスの起動に使用する **qpidd** スクリプトを定義します。また、既存の **qpidd** ブローカーをプライマリーステータスにプロモートするのではなく、新規サービスを起動する **qpidd-primary** スクリプトも定義します。この **qpidd-primary** スクリプトは、**qpid-cpp-server-ha** パッケージによりインストールされます。*

resources セクションは、クライアントとブローカー間の通信の仮想 IP アドレスを定義します。

*仮想 IP アドレスを活用するには、以下の行が含まれている **qpidd.conf** 必要があります。*

```
ha-cluster = yes
ha-public-url = 20.0.20.200
ha-brokers-url = 20.0.10.1, 20.0.10.2, 20.0.10.3
```

この設定では、3 つのノード(**ha-brokers-url**)の実際のネットワークアドレスと、クライアントが以下に接続する必要のあるクラスターの仮想 IP アドレスを指定します **20.0.10.200**。

この **service** セクションでは、各ノードごとに 3 つの **qpidd** サービスを定義します。各サービスは、そのノードのみを含む制限されたフェールオーバードメインにあり、リカバリーポリシーが **restart** あります。つまり、rgmanager は各ノード **qpidd** で実行され、失敗した場合には再起動します。

qpidd-primary スクリプト **qpidd-primary-service** を使用した 1 つは 1 つです。これはドメインに制限されず、リカバリーポリシーが **relocate** あります。これは、クラスターの **rgmanager** 起動時 **qpidd-primary** にノードの 1 つで起動し、元のノードが失敗した場合は別のノードに再度移動することを意味します。**qpidd-primary** スクリプトを実行しても、新しいブローカープロセスは起動せず、既存のブローカーをプロモートしてプライマリーになります。

バグを報告します。

9.1.13. HA ノードでの qpidd のシャットダウン

ノードごとの **qpidd** サービスと再配置可能な **qpidd-primary** サービスはいずれも同じ **qpidd** デーモンにより実装されます。

その結果、**qpidd** サービスを停止しても、プライマリーとして動作する **qpidd** デーモンは停止せず、**qpidd-primary** サービスを停止しても、バックアップとして動作する **qpidd** プロセスは停止されません。

プライマリーとして機能しているノードをシャットダウンするには、**qpidd** サービスをシャットダウンして、プライマリーの場所を **再配置** する必要があります。

```
clusvcadm -d somenode-qpidd-service
clusvcadm -r qpidd-primary-service
```

これを実行すると、そのノードの **qpidd** デーモンがシャットダウンします。また、このサービス **qpidd** は、その場所で実行されなくなったため、プライマリーサービスがノードに戻さないようにします。

バグを報告します。

9.1.14. HA クラスターの起動および停止

クラスターを起動すると、システムの起動時にクラスターサービスの起動が有効になります。クラスターを停止すると、再起動時にクラスターサービスの起動が無効になります。

ノード上での HA クラスターの起動および停止

ノード上で HA クラスターを起動するには、以下を実行します。

```
ccs [-h host] --start
```

ノード上の HA クラスターを停止するには、以下を実行します。

```
ccs [-h host] --stop
```

すべてのノードでHA クラスターを起動および停止

設定されているすべてのノードでHA クラスターを起動するには、以下を実行します。

```
ccs [-h host] --startall
```

これにより、システムの起動時にクラスターサービスの起動も可能になることに注意してください。

設定されているすべてのノードでHA クラスターを停止するには、以下を実行します。

```
ccs [-h host] --stopall
```

バグを報告します。

9.1.15. root 以外のユーザー（root 以外の）ユーザーを使用するようにクラスタリングを設定する

qpidd を root 以外のユーザーとして実行する場合は、ユーザーが読み取り可能かつ書き込み可能な場所に設定ファイルを保存する必要があります。これには、以下のように qpidd の start-up スクリプトを変更します。

```
# diff -u /etc/rc.d/init.d/qpidd /etc/rc.d/init.d/qpidd-mod
--- /etc/rc.d/init.d/qpidd.orig 2014-01-15 19:06:19.000000000 +0100
+++ /etc/rc.d/init.d/qpidd      2014-02-07 16:02:47.136001472 +0100
@@ -38,6 +38,9 @@
 prog=qpidd
 lockfile=/var/lock/subsys/$prog
 pidfile=/var/run/qpidd.pid
+
+CFG_DIR=/var/lib/qpidd
+QPIDD_OPTIONS="--config ${CFG_DIR}/qpidd.conf --client-config ${CFG_DIR}/qpidc.conf"

 # Source configuration
 if [ -f /etc/sysconfig/$prog ] ; then
```

上記のパッチが適用されると **/etc/rc.d/init.d/qpidd**、ブローカーの設定ファイルは、デフォルトでは **/etc/qpid** そのままではなく、**/var/lib/qpidd** ディレクトリーから読み取られます。

バグを報告します。

9.1.16. Broker Administration Tools and HA

通常、クライアントはバックアップブローカーに接続できません。ただし、管理ツールはバックアップブローカーに接続できます。これらのツールを使用する場合は、レプリケートされたキューからメッセージを追加または削除したり、複製されたキューまたは交換を作成または削除したりしないでください。複製されたキューが中断するため、メッセージが失われる可能性があります。

qpid-ha HA 構成設定を表示および変更できます。

このツール **qpid-stat** は **qpid-config qpid-route**、コマンドラインでフラグを渡すと、バックアップに接続 **--ha-admin** します。

バグを報告します。

9.1.17. キューと交換のレプリケーションの制御

デフォルトでは、キューと交換は自動的にレプリケートされません。デフォルトの動作を変更するには、ha-replicate 設定オプションを設定します。これには、以下のいずれかの値があります。

all

キュー、交換、バインディング、メッセージのすべてを自動的に複製します。

configuration

キュー、交換、バインディングの存在を複製しますが、メッセージを複製しません。

none

何も複製しません。これはデフォルトです。

キューまたは交換 **qpid.replicate** 時に引数を渡すと、特定のキューのデフォルトを超過したり、交換したりできます。これはと同じ値を取り **ha-replicate** ます。

バインディングは、キューがバインドされどちらの交換にもレプリケーションすべてまたは設定がある場合に自動的にレプリケートされます。そうでない場合は、バインディングはレプリケートされません。

レプリケートされたキューを作成し、以下のような **qpid-config** 管理ツールで交換できます。

```
qpid-config add queue myqueue --replicate all
```

レプリケートされたキューを作成し、クライアント API 経由で交換するには、以下のようにノードエントリーをアドレスに追加します。

```
"myqueue;{create:always,node:{x-declare:{arguments:{'qpid.replicate':all}}}}"
```

ブローカーによって自動的に作成される組み込み交換がありますが、これらの交換はレプリケートされません。組み込み交換は、デフォルト（名前なし）交換、AMQP 標準交換（、、**amq.fanout** および **amq.match**）**amq.direct amq.topic**、および管理交換（、**qpid.management qmf.default.direct** および **qmf.default.topic**）です。

複製されたキューをこれらの交換のいずれかにバインドすると、バインディングはレプリケートされないため、キューにはフェイルオーバー後にバインディングは含まれません。

バグを報告します。

9.1.18. クライアント接続と失敗

クライアントはプライマリーブローカーのみに接続できます。バックアップブローカーは、クライアントによる接続の試行を拒否します。バックアップブローカーによって拒否されたクライアントは、プライマリーに接続するまで自動的にフェイルオーバーします。複数のアドレスが **ha-public-url** 含まれる場合、クライアントはすべてのアドレスをローテーションで試行します。仮想 IP アドレスの場合は、再接続するまでクライアントが同じアドレスで再試行します。

クライアントはクラスターの URL で設定されます（クライアントのタイプごとに以下を参照してください）。以下の 2 つの可能性が考えられます。

1. URL には、リソースマネージャーによってプライマリーブローカーに割り当てられる単一の仮想 IP アドレスが含まれます。**これは推奨される設定です。**

2. URL には複数のアドレスが含まれ、クラスターのブローカーごとに1 つずつアドレスが含まれます。

最初の例では、リソースマネージャーは仮想 IP アドレスをプライマリーブローカーに割り当てます。そのため、クライアントは1 つのアドレスでのみ再試行する必要があります。2 つ目のケースでは、クライアントはプライマリーに正常に接続されるまで、URL の各アドレスを繰り返し再試行します。

プライマリーブローカーが失敗すると、クライアントは新しいプライマリーに接続するまですべての既知のクラスターアドレスを再試行します。クライアントは、障害発生時にブローカーによって以前に送信されても確認されなかったメッセージをすべて再送信します。同様に、ブローカーによって送信されても、クライアントによって確認されていないメッセージは再度キューに入れられます。

TCP は接続障害の検出に時間がかかる場合があります。クライアントは、ハートビートを使用して接続障害を検出できるように接続を設定し、ハートビートの間隔を指定できます。ハートビートが使用されると、ハートビート間隔の 2 倍未満の障害が検出されます。以下のセクションでは、各クライアントでハートビートを有効にする方法を説明します。

注記： クラスターで仮想 IP アドレスを使用している場合は、サーバーでフェールオーバーが行われ、仮想 IP アドレスを使用してクラスターを単一のブローカーとして扱います。以下のセクションでは、クラスターが仮想 IP を使用しない場合に、複数のアドレスでクライアントを設定する方法を説明します。

クラスターに 3 つのノードがある **node2** と仮定し **node1**、**node3** すべてデフォルトの AMQP ポートを使用しており、仮想 IP アドレスを使用していない場合。クライアントに接続するには、address(es) を指定し、reconnect プロパティーを true に設定します。以下のサブセクションでは、クライアントの各種別に接続する方法を説明します。

C++ クライアント

C++ クライアントでは、複数のクラスターアドレスを単一の URL に指定します。また、true に connection オプションを指定 **reconnect** する必要があります。例：

```
qpid::messaging::Connection c("node1,node2,node3","{reconnect:true}");
```

ハートビートはデフォルトで無効になっています。**heartbeat** オプションを使用して接続のハートビート間隔（秒単位）を指定して有効にできます。例：

```
qpid::messaging::Connection c("node1,node2,node3","{reconnect:true,heartbeat:10}");
```

Python クライアント

Python クライアントでは、**Connection.establish** またはの呼び出し **reconnect_urls** 時として **host:port** アドレスのリスト **reconnect=True** と、アドレスのリストを指定し **Connection.open**ます。

```
connection = qpid.messaging.Connection.establish("node1", reconnect=True, reconnect_urls=
["node1", "node2", "node3"])
```

ハートビートはデフォルトで無効になっています。'**heartbeat**' オプションで接続のハートビート間隔（秒単位）を指定して有効にできます。例：

```
connection = qpid.messaging.Connection.establish("node1", reconnect=True, reconnect_urls=
["node1", "node2", "node3"], heartbeat=10)
```

Java JMS クライアント

*Java JMS クライアントでは、接続で有効になっていると、クライアントのフェールオーバーは自動的に処理されます。**failover** プロパティーを使用して fail-over を使用するようにコネクションを設定できます。*

```
connectionfactory.qpidConnectionfactory = amqp://guest:guest@clientid/test?
brokerlist='tcp://localhost:5672'&failover='failover_exchange'
```

このプロパティーには 3 つの値を使用できます。

フェイルオーバーモード

failover_exchange

接続に失敗した場合は、クラスター内の他のブローカーにフェイルオーバーします。

roundrobin

接続に失敗した場合は、brokerlist に指定されたブローカーの1 つにフェイルオーバーします。

singlebroker

フェイルオーバーはサポート対象外であり、接続は単一のブローカーのみに限定されます。

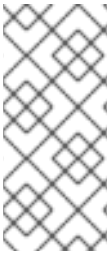
*Connection URL では、ハートビートは **idle_timeout** プロパティーを使用して設定されます。これはハートビート期間に対応する整数（秒単位）です。たとえば、JNDI プロパティーファイルから以下の行はハートビートタイムアウトを 3 秒に設定します。*

```
connectionfactory.qpidConnectionfactory = amqp://guest:guest@clientid/test?
brokerlist='tcp://localhost:5672',idle_timeout=3
```

バグを報告します。

9.1.19. security

本項では、セキュリティー設定の HA 固有の側面について説明します。認証 の有効化およびアクセス制御リストの設定に関する詳細は、 を参照してください。



注記

*設定を使用した認証を無効 **auth=no** にしない限り、以下のオプションを **設定** し、少なくとも以下のエントリーを持つACL ファイルが **必要** です。*

*セキュリティー設定が正しくない場合、**バックアップはプライマリーに接続できません**。併せて参照してください。 「クラスター設定のトラブルシューティング」*

認証が有効な場合、以下のオプションを使用してHA ブローカーによって使用されるクレデンシャルを設定する必要があります。

表9.2 HA のセキュリティーオプション

HA のセキュリティーオプション	
ha-username *USER*	HA ブローカーのユーザー名。これには **@QPID** 接尾辞を含めることはでき **ません**。

HA のセキュリティーオプション	
ha-password *PASS*	HA ブローカーのパスワード。
ha-mechanism *MECHANISM*	HA ブローカーのメカニズム。Broker-to-broker 通信に対して有効にするメカニズムは、クライアントでも使用できるため、セキュアな環境 **ANONYMOUS** で使用しないでください。

このアイデンティティーは、バックアップからプライマリーへのフェデレーションリンクを承認するために使用されます。また、バックアップでアクションを承認して、キューの作成や交換など、プライマリー状態を複製するために使用されます。

承認が有効な場合、HA レプリケーションを機能させるには、以下のルールを持つアクセス制御リストが必要です。suppose ***ha-username****=USER*

```
acl allow USER@QPID all all
```

関連項目

- *「クラスタリングの ACL 要件」*

バグを報告します。

9.1.20. HA クラスタリングと永続性

メッセージに永続ストアを使用する場合、クラスター内の各ブローカーには独自のストアがあります。クラスター全体に失敗し、再起動される場合、プライマリーとなる **最初** *のブローカーはそのストアから復元します。その他のブローカーはすべてストアをクリアし、プライマリーから更新を取得し、一貫性を確保します。*

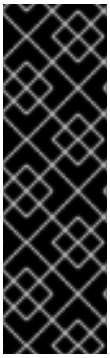
バグを報告します。

9.1.21. キューレプリケーションと HA

アクティブ/パッシブクラスターのサポートに加えて、ブローカーがクラスター環境に含まれていない場合でも、 ***HA*** *モジュールは個別のキューレプリケーションをサポートします。元のキューは通常どおりに使用されますが、元のキューにメッセージが追加または削除されると、レプリカキューは自動的に更新されます。*

レプリカキューを作成するには、 ***HA*** *モジュールを元のブローカーとレプリカブローカーの両方でロードする必要があります。これはデフォルトで自動的に行われます。*

スタンドアロンブローカーの場合は、 ***ha-queue-replication=yes*** *設定オプションを指定する必要があります。このオプションは自動的に読み込まれるため、クラスター環境の一部であるブローカーには不要です。*



重要

キューの変更は、元のキューからの自動更新でのみ受け取る必要があります。

レプリカキューでメッセージを手動で追加または削除すると、レプリケーションに一貫性がなくなり、メッセージが失われる可能性があります。

***HA** モジュールは、レプリカキューへのアクセスが制限されません（クラスターの場合と同様）。アプリケーションは、レプリカが元のレプリカから切断されるまで使用されないようにする必要があります。*

例9.1 ノード間のキューの複製

***myqueue** これはのキューであると仮定し **node1**ます。*

***myqueue** on のレプリカを作成するに **node2**は、以下のコマンドを実行します。*

```
qpid-config --broker=node2 add queue --start-replica node1 myqueue
```

*レプリカブローカーに **myqueue** すでに存在する場合は、以下のコマンドを実行して元のキューからレプリケーションを開始します。*

```
qpid-ha replicate -b node2 node1 myqueue
```

バグを報告します。

9.2. クラスター管理

9.2.1. を使用したクラスター管理 qpid-ha

***qpid-ha** は、クラスターとそのブローカーに関する情報の表示、クライアント接続の接続の接続解除、クラスター内のブローカーのシャットダウン、またはクラスター全体をシャットダウンできるコマンドラインユーティリティーです。これは、コマンドとオプションを受け入れます。*

***qpid-ha** には、以下のコマンドおよびパラメーターがあります。*

コマンド

status

*HA ステータスを出力します。指定されたブローカーがプライマリー（アクティブ）またはバックアップとして機能するかを返します。**--all** オプションを指定すると、クラスター全体が一覧表示されます。*

例：

```
# qpid-ha status
ready
# qpid-ha status --all
192.168.6.60:5672 ready
192.168.6.61:5672 active
192.168.6.62:5672 ready
```

PING

ブローカーが実行中で応答しているかどうかを確認します。

クエリー

HA 設定とステータスを出力します。以下の情報が返されます。

- *ブローカーステータス：primary（アクティブ）またはバックアップ(ready)。*
- *HA ブローカー URL のリスト*
- *パブリック（仮想）HA URL*
- *レプリケーションのステータス*

例：

```
# qpid-ha query
Status:       ready
Brokers URL:   amqp:tcp:192.168.6.60:5672,tcp:192.168.6.61:5672,tcp:192.168.6.62:5672
Public URL:    amqp:tcp:192.168.6.251:5672
Replicate:     all
```

複製

現在のブローカーで <queue> から <remote-broker> から <queue> へ の レプリケーションを設定します。

parameters

--broker=BROKER

qpidd ブローカーのアドレス。構文を以下に示します。

```
[username/password@] hostname | ip-address [:port]
```

--sasl-mechanism=SASL_MECH

認証用の SASL メカニズム（EXTERNAL、ANONYMOUS、PLAIN、CRAM-MD5、DIGEST-MD5、GSSAPI など）。SASL は、利用可能な最も安全なメカニズムを自動的に選択し、このオプションを使用して上書きします。

--ssl-certificate=SSL_CERT

クライアント SSL 証明書（PEM 形式）。

--config=CONFIG

*設定ファイル（など）を読み込んで/**etc/qpid/qpidd.conf**、ローカルの qpidd に接続します。*

--timeout=SECONDS

*ブローカーがタイムアウト内で応答しない場合は断念します。**0** は永久に待機します。デフォルトはです **10.0**。*

--ssl-key=KEY

クライアント SSL 秘密鍵（PEM 形式）

--help-all

上記のコマンドおよびパラメーターをすべて出力します。

*各コマンドは、さまざまなオプションを受け入れます。オプションは、**--help-all** 以下のオプションを使用して確認できます。*

```
$ qpid-ha --help-all
```

関連項目

- *「キューレプリケーションと HA」*

バグを報告します。

9.3. クラスターのトラブルシューティング

9.3.1. クラスター設定のトラブルシューティング

*HA クラスターを最初に起動すると、すべてのブローカーは結合モードになります。ブローカーはプライマリーを自動的に選択せず、クラスターマネージャー **rgmanager** に依存します。**rgmanager** が実行していない場合、または正しく設定されていない場合、ブローカーは参加状態のままになります。詳細はを 「Cluster Resource Manager(rgmanager)」 参照してください。*

ブローカーがクラスターの別のブローカーへの接続を確立できない場合、ログには以下のような SASL エラーが含まれます。

```
info SASL: Authentication failed: SASL(-13): user not found: Password verification failed
```

```
warning Client closed connection with 320: User anonymous@QPID federation connection denied.
Systems with authentication enabled must specify ACL create link rules.
```

```
warning Client closed connection with 320: ACL denied anonymous@QPID creating a federation link.
```

手順9.3 クラスター SASL 設定のトラブルシューティング

1. *の説明に従って、HA セキュリティー設定および ACL ファイルを設定します。 「security」*
2. *クラスターが実行され、プライマリーがプロモートされたら、 : を実行してブローカーが 1 つのクラスターとして実行されていることを*

   ```
   qpid-ha status --all
   ```

   確認します。
3. *クラスターが実行されたら、ブローカーが 1 つのクラスターとして実行されていること **qpid-ha** を確認します。*

バグを報告します。

9.3.2. 低速なリカバリーの時間

以下の構成設定は、復旧時間に影響します。使用される値の例により、ライトロードされたシステムで高速リカバリーが可能になります。テストを実行して、システムに適切な値と負荷条件があるかどうかを判断します。

cluster.conf

<rm status_poll_interval=1>

status_poll_interval は、リソースマネージャーが管理サービスのステータスをチェックする間隔（秒単位）です。これは、マネージャーが失敗したサービスを検出する速度に影響します。

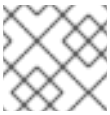
<ip address="20.0.20.200" monitor_link="yes" sleeptime="0"/>

これは、クライアントトラフィックの仮想 IP アドレスです。VIP に使用される NIC の正常性を **monitor_link="yes"** 監視することを **sleeptime="0"** 意味します。つまり、仮想 IP を新規アドレスにフェイルオーバーしたときに遅延しないようにします。

qpidd.conf

link-maintenance-interval=0.1

必要な場合は、バックアップブローカーがプライマリー再接続へのリンクをチェックするまで待機する秒数。デフォルト値はです **2**。この値は、高速なフェイルオーバー（例： **0.1**）よりも低い値に設定することができます。



注記

値を低く設定すると、ブローカーのリンクチェックが過剰になります。

link-heartbeat-interval=5

フェデレーションリンクのハートビート間隔。HA クラスターは、プライマリーと各バックアップ間のフェデレーションリンクを使用します。プライマリーは、障害が発生したバックアップを検出するためにハートビート間隔を最大 2 倍の時間がかかる場合があります。送信者がメッセージを送信すると、プライマリーはすべてのバックアップが確認されるのを待機してから、送信元に承認します。非接続バックアップでは、ハートビートで検出されるまで、プライマリーが送信側をブロックする可能性があります。

この間隔は、rgmanager によるブローカーステータスチェックのタイムアウトとしても使用されます。ハングしたブローカーを検出するには、この間隔 **rgmanager** にかかる場合があります。

デフォルトは 120 秒です。これは、可用性と応答時間が重要な多くの実稼働環境では、高すぎる可能性があります。ただし、設定が低すぎると、ネットワークの輻輳下で、または負荷が大きい場合、応答が遅いブローカーを再起動することができ **rgmanager**ます。

バグを報告します。

9.3.3. クラスター障害の合計

クラスターは、アクティブなプライマリーブローカーまたはバックアップブローカーが稼働状態のままである限り、可用性を保証します。すべてのブローカーが同時に失敗すると、クラスターが失敗し、永続化されないデータが失われます。

ブローカーの状態は 6 つです。

1. ***standalone***: *HA クラスターの一部ではない*
2. ***joining***: *新規起動バックアップ（まだクラスターに参加していない）*
3. ***catch-up***: *バックアップはプライマリーに接続され、キューやメッセージをダウンロードしている。*
4. ***ready***: *バックアップはプライマリーから接続されアクティブに複製され、引き継ぐ準備が整っています。*
5. ***recovering***: *新たにプライマリーにプロモートされ、クライアントを提供する前にバックアップがキャッチされるのを待機します。1 つのプライマリーブローカーのみが一度にリカバリーできます。*
6. ***active***: *クライアントを提供すると、一度にアクティブにできる単一のプライマリーブローカーのみを使用できます。*

アクティブなプライマリーブローカーがありますが、クライアントはサービスを取得できます。アクティブなプライマリーに失敗すると、「ready」バックアップブローカーの 1 つが引き継ぎ、リカバリーされ、アクティブになります。バックアップは、「ready」状態にある場合にのみプライマリーに昇格できます（すべてのブローカーが「参加」状態にある新規クラスターの最初のプライマリーを除く）。

1 つのアクティブなプライマリーおよび N-1 バックアップが準備できている N ブローカーの安定したクラスターがある場合、システムは N-1 の障害を迅速かつ維持できます。ブローカーはアクティブにプロモートされ、サービスの提供を継続します。

ただし、この時点では、他のブローカーの少なくとも 1 つが復旧し、バックアップの準備が整うまで、システムは停止ブローカーの障害に対応できません。クラスターが 2 つのモードのいずれかで失敗する前に、ブローカーが失敗する場合（正確な障害のタイミングにより異なる）。

1. クラスターがハングする

*すべてのブローカーは参加中またはキャッチアップモードです。新しいプライマリーをプロモート **rgmanager** しようとしますが、新しいプライマリーは検索できず、その候補は破棄されません。clustat は **qpidd** サービスが実行中であることを示しますが、**qpidd-primary** サービスが停止したことを示します。*

表9.3

サービス名	所有者(——)	状態
service:mrg33-qpidd-service	20.0.10.33	Started
service:mrg34-qpidd-service	20.0.10.34	Started
service:mrg35-qpidd-service	20.0.10.35	Started
service:qpidd-primary-service	(20.0.10.33)	Stopped

*最終的に、に示されるように、すべてのブローカーは「結合」モードで停止し **qpid-ha status --all** ます。*

この時点で、以下のいずれかの方法でクラスターを再起動する必要があります。

クラスター全体を再起動します。

- *luci:<your-cluster>:Nodes で再起動をクリックしてクラスター全体を再起動します。*
- **または、** *でクラスターを停止して再起動します。* ***ccs --stopall ccs --startall***

Qpid サービスのみを再起動します。

- *luci:<your-cluster>:Service Groups で以下を行います。*
  - *すべての qpidd（プライマリーではない）サービスを選択し、再起動をクリックします。*
  - *qpidd-primary サービスを選択し、再起動をクリックします。*
- *clusvcadm でプライマリーサービスおよび qpidd サービスを* **停止** *してから再起動します（プライマリー最終）。*

2.クラスターが再起動します。

新しいプライマリーがプロモートされ、クラスターが機能するようになりました。障害が失われる前に永続的なデータすべて。

バグを報告します。

9.3.4. フェンシングとネットワークのパーティション

ネットワークパーティションは、クラスターを 2 つ以上のサブクラスターに分割するネットワーク障害です。この場合、各ブローカーは独自のサブクラスターのブローカーと通信できますが、他のサブクラスターのブローカーとは通信できません。この状態は、「スプリットブレイン」とも呼ばれます。

1 つのサブクラスターのノードは、他のサブクラスターのノードが期限切れであるか、または実行中であるかどうかを判別できませんが、切断されています。クラスターの整合性がなくなるため、各サブクラスターが独立して独自の qpidd プライマリーを宣言し、提供するクライアントを開始させることはできません。1 つのサブクラスターのみがサービスを提供するようにする必要があります。

クォーラム は、動作を継続するサブクラスターを決定し、電源フェンシング により、非クォーラムのサブクラスターのノードがサービスに一貫性のない状態で提供を試みないようにします。詳細は、「『Red Hat Enterprise Linux 6 High Availability Add-on Overview 』」の章 2 を参照してください。クォーラム および 第 4 章フェンシング。

バグを報告します。

第10章 ブローカーフェデレーション

10.1. ブローカーフェデレーション

ブローカーフェデレーションにより、メッセージングネットワークは、あるブローカー（ ソースブローカー）のメッセージが別のブローカー（ 宛先ブローカー）に自動的にルーティングされるメッセージ ルート を作成して定義できます。これらのルートは、2 つのブローカー（ ソース交換と 宛先交換）の交換、またはソースブローカー（ソースキュー）の キューから宛先ブローカーの交換まで定義できます。

メッセージルートは一方向です。双方向フローが必要な場合は、各方向に1 つのルートが作成されます。ルートは永続的または一時的なものにすることができます。永続的なルートはブローカーの再起動後も維持され、ソースブローカーと宛先の両方が利用可能になり次第、ルートを復元します。宛先への接続が失われると、永続ルートに関連付けられたメッセージがソース上で蓄積され、接続の再確立時にそれらが取得されます。

ブローカーフェデレーションは、一度に1 つのルートを持つ、多くのブローカーを持つ大規模なメッセージングネットワークの構築に使用できます。ネットワーク接続が許可されている場合、分散メッセージングネットワーク全体は単一の場所から設定できます。ルーティングに使用されるルールは、サーバーや責任が日ごとに変化する場合や、その他の変化状態を反映するために動的に変更できます。

バグを報告します。

10.2. ブローカーフェデレーションのユースケース

ブローカーフェデレーションは、さまざまなシナリオで役に立ちます。これらの一部は、機能的な組織で行う必要があります。たとえば、ブローカーは、地理的、サービスタイプ、優先度別に編成できます。フェデレーションのユースケースを以下に示します。

- geography: お客様のリクエストは顧客に近い処理場所にルーティングできます。
- サービスタイプ： 高値お客様は、より応答性の高いサーバーにルーティングできます。
- Load balancing: ブローカー間のルーティングは、実際の負荷または予想される負荷の変更を考慮するために動的に変更できます。
- 高可用性： 既存のブローカーが利用できなくなると、新しいブローカーにルーティングを変更できます。
- WAN Connectivity: フェデレーションされたルートは、幅広いエリアネットワーク全体で異種の場所を接続でき、クライアントは独自のローカルエリアネットワーク上のブローカーに接続できます。各ブローカーは、WAN 接続にギャップがある場合でもメッセージを保持できる永続キューを提供できます。
- 機能組織： ソフトウェアサブシステム間のメッセージのフローは、分散アプリケーションの論理構造をミラーリングするように設定できます。
- レプリケートされた交換： XML の交換などの高機能交換を複製して、パフォーマンスをスケーリングできます。
- Interdepartmental Workflow: ブローカー間のメッセージのフローは、組織で相互ワークフローをミラーリングするように設定できます。

バグを報告します。

10.3. ブローカーフェデレーションの概要

10.3.1. メッセージルート

ブローカーフェデレーションは、メッセージルートを作成します。ルートの宛先は常に宛先ブローカーで交換されます。

デフォルトでは、宛先ブローカーを設定してメッセージルートが作成され、ソースブローカーに連絡してソースキューにサブスクライブします。これはプルルートと呼ば れます。

ソースブローカーを設定してから宛先ブローカーに接続してメッセージを送信することで、ルートを作成することもできます。これは プッシュルート と呼ばれ、メッセージングルートの設定時に宛先ブローカーが利用できない場合や、同じ宛先交換で多数のルートが作成される場合に特に便利です。

ルートのソースは、ソースブローカーの交換またはキューのいずれかになります。ルートが2 つの交換の間にある場合、ルーティング基準を明示的に指定することも、宛先交換のバインディングを使用してルーティング基準を指定できます。この機能をサポートするには、以下の3 つの種類のメッセージルートがあります。

- キュールート
- 交換ルート
- 動的交換ルート

バグを報告します。

10.3.2. キュールート

キュールートは、送信元キューから宛先交換にすべてのメッセージをルーティングします。メッセージの承認が有効な場合、メッセージは宛先の交換によって受信されたときにキューから削除されます。メッセージ確認応答がオフである場合、メッセージは送信されるとキューから削除されます。

AMQP キューに複数のサブスクリプションがある場合、メッセージはサブスクライバー間で分散されます。たとえば、同じキューから送信される2 つのキュールートはそれぞれ負荷分散された数のメッセージを受け取ります。

負荷分散の代わりに facutout 動作が必要な場合は、交換ルート を使用します。

バグを報告します。

10.3.3. 交換ルート

Exchange は、バインディングキーを使用して、ソースの交換から宛先の交換にルーティングをルーティングします（FANout の交換では任意）。

内部的に、交換ルートを作成すると、ソースブローカーにプライベートキュー（auto-delete、exclusive）が作成され、宛先ブローカーにルーティングされるメッセージを保持し、このプライベートキューをソースブローカーの交換にバインドし、宛先ブローカーをキューにサブスクライブします。

バグを報告します。

10.3.4. 動的交換ルート

動的交換ルート は、あるブローカー（ソース）上の交換を別のブローカー（宛先）の交換に接続し、クライアントが宛先ブローカー上の交換をサブスクライブできるようにし、交換に送信されたサブスクリプションに一致するメッセージを受信します。

動的交換ルートは 方向です。ブローカーA からブローカーB への動的な交換ルートを作成すると、ブローカーB はクライアントの代わりにブローカーA の交換に対するサブスクリプションを動的に作成し、削除します。このようにして、ブローカーB はクライアントのプロキシーとして機能します。クライアントがブローカーB の宛先交換をサブスクライブすると、ブローカーB は、クライアントが作成したサブスクリプションとブローカーA のソース交換をサブスクライブします。ブローカーB にサブスクライブするクライアントは、ブローカーB の交換と、ブローカーA の動的なルーティング交換の両方を効果的にサブスクライブします。

消費ブローカーは、クライアントをサブスクライブし、使用ブローカーの宛先交換にサブスクライブし、サブスクライブ解除するソースブローカーの動的交換に対してサブスクリプションを作成し、削除します。動的交換ルートは競合するサブスクリプションであるため、宛先ブローカーは他のサブスクライバーと同様にメッセージコンシューマーになります。

動的交換ルートでは、送信元および宛先の交換が同じ型で、同じ名前を持つ必要があります。たとえば、ソースの交換が直接交換の場合、宛先の交換も直接交換で、名前が一致する必要があります。

内部的には、動的交換ルートは交換ルートと同じ方法で実装されます。ただし、動的交換ルートの実装に使用されるバインディングは、宛先の交換変更のバインディングが変更される点が異なります。

動的交換ルートは常にプルルートです。プッシュルートになることはできません。

関連項目

- *「動的交換ルートの作成および削除」*

バグを報告します。

10.3.5. フェデレーショントポロジー

フェデレーションされたネットワークは通常、2 つのブローカー間で双方向リンク（一方向リンクのペアとして実装）を使用するツリー、星、または線です。一方向リンクのみが使用されている場合は、リングトポロジーも可能です。

すべてのメッセージの転送には時間がかかります。パフォーマンスを向上させるには、メッセージの送信元と最終宛先間のブローカーの数を最小限に抑えます。ほとんどの場合、ツリーまたは星トポロジーはこの最適です。

A と B の間に複数のパスがある場合、ノードA のペアのB では、すべてのメッセージがA と B の間でサイクルできないことを確認します。メッセージをループさせると、フェデレーションされたネットワークがいっぱいになる可能性があります。ツリー、星、およびライントポロジーにはメッセージループがありません。双方向リンクを持つリングトポロジーは、この問題を引き起こすトポロジーの例です。これは、指定のブローカーが 2 つの異なるブローカーから同じメッセージを受信できるためです。サービスメッシュトポロジーもこの問題を引き起こす可能性があります。

バグを報告します。

10.3.6. フェデレーション高可用性クラスター

フェデレーションは通常、高可用性メッセージクラスターとともに使用され、クラスターを使用して各 LAN で高可用性を提供し、フェデレーションを使用してクラスター間でメッセージをルーティングします。

2 つのクラスター間のメッセージルートを作成するには、最初のクラスターの1 つのブローカーと 2 つ目のクラスターの任意の1 つのブローカー間のルートを作成します。指定のクラスターの各ブローカーは、同じクラスターの別のブローカーに定義されたメッセージルートを使用できます。メッセージルートが定義されるブローカーが失敗する場合、同じクラスターの別のブローカーはメッセージルートを復元できます。

バグを報告します。

10.4. ブローカーフェデレーションの設定

10.4.1. qpid-route ユーティリティー

qpid-route は、ブローカーのフェデレーションネットワークを設定し、ネットワークのステータスとトポロジーを表示するために使用されるコマンドラインユーティリティーです。接続可能なブローカー間のルートを設定するために使用 **qpid-route** できます。

バグを報告します。

10.4.2. qpid-route 構文

の構文は以下のとおり **qpid-route** です。

```
qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange>
qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>

qpid-route [OPTIONS] route add <dest-broker> <src-broker> <exchange> <routing-key>
qpid-route [OPTIONS] route del <dest-broker> <src-broker> <exchange> <routing-key>

qpid-route [OPTIONS] queue add <dest-broker> <src-broker> <dest-exchange>  <src-queue>
qpid-route [OPTIONS] queue del <dest-broker> <src-broker> <dest-exchange>  <src-queue>

qpid-route [OPTIONS] route list  [<broker>]
qpid-route [OPTIONS] route flush [<broker>]
qpid-route [OPTIONS] route map   [<broker>]

qpid-route [OPTIONS] link add  <dest-broker> <src-broker>
qpid-route [OPTIONS] link del  <dest-broker> <src-broker>
qpid-route [OPTIONS] link list [<dest-broker>]
```

、**dest-broker**および **broker**の構文は以下のとおり **src-broker** です。

```
[username/password@] hostname | ip-address [:<port>]
```

上記の構文の有効な例は次 **localhost 10.1.1.7:10000**のとおりです **broker-host:10000 guest/guest@localhost**。

バグを報告します。

10.4.3. qpid-route オプション

変更

- MRG-M 3 の新機能

表10.1 qpid-route フェデレーションを管理するオプション

オプション	description
-v	詳細出力。
-q	サイレントの出力は、重複した警告を出力しません。
-d	ルートを永続化します。
-e	リンクの最後のルートを削除した後にリンクを削除します。
--timeout *N*	qpid-route がブローカーに接続する場合（秒単位）を待機する最大時間。デフォルトは 10 秒です。
--ack *N*	N. バッチでのルーティングメッセージ転送の確認応答。デフォルトは 0（確認応答なし）です。1 以上を設定すると、確認応答を有効にします。確認応答を使用すると、N よりも大きい N の値が 1 よりも大きいと、特に 2 つのブローカー間にネットワークレイテンシーが大きくなるとパフォーマンスが大幅に向上します。
--credit *N*	確認応答で使用する有限のクレジットを指定します。デフォルトでは、クレジットカードは 0 で、クレジットカードフロー制御は無効になっています。Backpressure は、明示的な **--credit** 引数を使用して調整できます。
-s [--src-local]	ソースブローカーでルートを設定します（プッシュルートを作成します）。
-t <transport> [--transport <transport>]	ルートに使用するトランスポートプロトコル。 ● TCP（デフォルト） ● ssl ● rdma
--client-sasl-mechanism <mech>	クライアントが宛先ブローカーに接続する際に認証を行うための SASL メカニズム（例： EXTERNAL、ANONYMOUS、PLAIN、CRAM-MD、DIGEST-MD5、GSSAPI）。

バグを報告します。

10.4.4. キュールートの作成および削除

1. キュールートを作成および削除するには、以下の構文を使用します。

```
qpid-route [OPTIONS] queue add <dest-broker> <src-broker> <dest-exchange> <src-queue>
qpid-route [OPTIONS] queue del <dest-broker> <src-broker> <dest-exchange> <src-queue>
```

2. たとえば、以下のコマンドを使用して、ソースブローカーで名前が付けられたキューから、宛先ブローカーの **amq.fanout** 交換 **public** にすべてのメッセージをルーティング **localhost:10002** するキュールートを作成し **localhost:10001**ます。

```
$ qpid-route queue add localhost:10001 localhost:10002 amq.fanout public
```

3. オプションで、キュールートを保持する **-d** オプションを指定します。ブローカーのいずれかまたは両方が再起動されると、キュールートが復元されます。

```
$ qpid-route -d queue add localhost:10001 localhost:10002 amq.fanout public
```

4. **del** コマンドは、コマンドと同じ引数を取り **add** ます。以下のコマンドを使用して、上記のキュールートを削除します。

```
$ qpid-route queue del localhost:10001 localhost:10002 amq.fanout public
```

バグを報告します。

10.4.5. 交換ルートの作成および削除

1. 交換ルートを作成および削除するには、以下の構文を使用します。

```
qpid-route [OPTIONS] route add <dest-broker> <src-broker> <exchange> <routing-key>
qpid-route [OPTIONS] route del <dest-broker> <src-broker> <exchange> <routing-key>
qpid-route [OPTIONS] flush [<broker>]
```

2. たとえば、以下のコマンドを使用して、ソースブローカーの交換 **global.#** から、宛先ブローカーの **amq.topic** 交換にバインディングキーと一致するメッセージをルーティング **localhost:10002** する **amq.topic** 交換ルートを作成し **localhost:10001**ます。

```
$ qpid-route route add localhost:10001 localhost:10002 amq.topic global.#
```

3. 多くのアプリケーションでは、宛先の交換に公開されるメッセージもソース交換にルーティングする必要があります。2 つ目の交換ルートを作成し、2 つの交換のロールを取り消します。

```
$ qpid-route route add localhost:10002 localhost:10001 amq.topic global.#
```

4. 交換ルートを保持する **-d** オプションを指定します。ブローカーのいずれかまたは両方が再起動されると、交換ルートが復元されます。

```
$ qpid-route -d route add localhost:10001 localhost:10002 amq.fanout public
```

5. **del** コマンドは、コマンドと同じ引数を取り **add** ます。以下のコマンドを使用して、上記の最初の交換ルートを削除します。

```
$ qpid-route route del localhost:10001 localhost:10002 amq.topic global.#
```

バグを報告します。

10.4.6. Broker のすべてのルートの削除

- **flush** コマンドを使用して、指定されたブローカーのすべてのルートを削除します。

```
qpid-route [OPTIONS] route flush [<broker>]
```

たとえば、以下のコマンドを使用してブローカーのすべてのルートを削除し **localhost:10001**ます。

```
$ qpid-route route flush localhost:10001
```

バグを報告します。

10.4.7. 動的交換ルートの作成および削除

1. 動的交換ルートを作成および削除するには、以下の構文を使用します。

```
qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange>
qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>
```

2. 2 つのブローカーごとに新しいトピック交換を作成します。

```
$ qpid-config -a localhost:10003 add exchange topic fed.topic
$ qpid-config -a localhost:10004 add exchange topic fed.topic
```

3. ソースブローカーの交換から宛先ブローカーの **fed.topic** 交換にメッセージをルーティング **localhost:10004** する動的 **fed.topic** 交換ルートを作成し **localhost:10003**ます。

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
```

内部的には、これによりソースブローカーのプライベート自動削除キューが作成され、そのキューは宛先ブローカーの **fed.topic** 交換に関連する各バインディングを使用してソースブローカーの **fed.topic** 交換にバインドされます。

4. 多くのアプリケーションでは、宛先の交換に公開されるメッセージもソース交換にルーティングする必要があります。2 番目の動的交換ルートを作成し、2 つの交換のロールを取り消します。

```
$ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
```

5. 交換ルートを保持する **-d** オプションを指定します。ブローカーのいずれかまたは両方が再起動されると、交換ルートが復元されます。

```
$ qpid-route -d dynamic add localhost:10004 localhost:10003 fed.topic
```

交換ルートが永続化される場合、ソースの交換でルートのメッセージを保管するために使用されるプライベートキューも永続化されます。ブローカー間の接続が失われると、宛先の交換のメッセージが復元されるまで累積されます。

6. **del** コマンドは、コマンドと同じ引数を取り **add** ます。上記の最初の交換ルートを削除します。

```
$ qpid-route dynamic del localhost:10004 localhost:10003 fed.topic
```

内部的には、メッセージルートに関連付けられたプライベートキューのソース交換のバインディングが削除されます。

バグを報告します。

10.4.8. ルートの表示

手順10.1 route list コマンドの使用

1. 以下の 2 つのルートを作成します。

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
```

2. **route list** コマンドを使用して、ブローカーに関連付けられたルートを表示します。

```
$ qpid-route route list localhost:10003
localhost:10003 localhost:10004 fed.topic <dynamic>
```

これ **localhost:10003** は、作成される 2 つのルート（宛先であるルート）の1 つのルートのみを示すことに注意してください。

3. 宛先であるルートを表示するに **localhost:10004** は、**localhost:10004**以下を実行 **route list** します。

```
$ qpid-route route list localhost:10004
localhost:10004 localhost:10003 fed.topic <dynamic>
```

手順10.2 route map コマンドの使用

1. この **route map** コマンドは、ブローカーに関連付けられたすべてのルートを表示し、指定のブローカーとフェデレーションするブローカーのすべてのルートを再帰的に表示します。たとえば、上記の 2 つのブローカーに対して **route map** コマンドを実行します。

```
$ qpid-route route map localhost:10003

Finding Linked Brokers:
    localhost:10003... Ok
    localhost:10004... Ok

Dynamic Routes:

  Exchange fed.topic:
    localhost:10004 <=> localhost:10003
```

```
Static Routes:
  none found
```

2 つの動的交換リンクは、双方向リンクであるかのように表示されます。この **route map** コマンドは、大規模な、より複雑なネットワークに役に立ちます。

2. 16 の動的交換ルートでネットワークを設定します。

```
qpid-route dynamic add localhost:10001 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10001 fed.topic

qpid-route dynamic add localhost:10003 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10003 fed.topic

qpid-route dynamic add localhost:10004 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10004 fed.topic

qpid-route dynamic add localhost:10002 localhost:10005 fed.topic
qpid-route dynamic add localhost:10005 localhost:10002 fed.topic

qpid-route dynamic add localhost:10005 localhost:10006 fed.topic
qpid-route dynamic add localhost:10006 localhost:10005 fed.topic

qpid-route dynamic add localhost:10006 localhost:10007 fed.topic
qpid-route dynamic add localhost:10007 localhost:10006 fed.topic

qpid-route dynamic add localhost:10006 localhost:10008 fed.topic
qpid-route dynamic add localhost:10008 localhost:10006 fed.topic
```

3. 1 つのブローカーで **route map** コマンドを使用して、ネットワーク全体を表示します。

```
$ qpid-route route map localhost:10001

Finding Linked Brokers:
    localhost:10001... Ok
    localhost:10002... Ok
    localhost:10003... Ok
    localhost:10004... Ok
    localhost:10005... Ok
    localhost:10006... Ok
    localhost:10007... Ok
    localhost:10008... Ok

Dynamic Routes:

  Exchange fed.topic:
    localhost:10002 <=> localhost:10001
    localhost:10003 <=> localhost:10002
    localhost:10004 <=> localhost:10002
    localhost:10005 <=> localhost:10002
    localhost:10006 <=> localhost:10005
    localhost:10007 <=> localhost:10006
    localhost:10008 <=> localhost:10006
```

```
Static Routes:
  none found
```

バグを報告します。

10.4.9. 回復性接続

ブローカールートが作成された場合や、ブローカーの再起動後に永続ブローカールートが復元されると、ソースブローカーと宛先ブローカーの間で 回復性のある接続 が作成されます。通信エラーが原因で接続が失敗すると、再接続が試行されます。リトライ間隔は 2 秒で始まり、試行回数がさらに行われると、64 秒まで増えます。64 秒ごとに再試行し続けます。認証の問題により接続が失敗すると、再接続は試行されません。

バグを報告します。

10.4.10. 回復接続の表示

このコマンドは、ブローカーの回復的な接続を表示するために使用 **link list** できます。

```
$ qpid-route link list localhost:10001

Host          Port    Transport Durable  State          Last Error
===============================================================================

localhost     10002   tcp       N    Operational
localhost     10003   tcp       N    Operational
localhost     10009   tcp       N    Waiting        Connection refused
```

上記の出力には、接続に対して受信した最後の接続エラーの文字列表現が **Last Error** 含まれます。は接続の状態を **State** 表し、以下の値のいずれかになります。

表10.2 状態値 $ qpid-route link list

オプション	description
待機中	再接続を試みる前に待機します。
接続	接続を確立しようとします。
運用性	接続が確立され、使用できる。
Failed	接続に失敗し、再試行されません（通常は認証に失敗したためです）。
closed	接続が閉じられ、すぐに削除されます。
パッシブ	クラスターが別のクラスターにフェデレーションされている場合は、いずれかのノードのみがリモートノードへの実際の接続を持ちます。クラスター内の他のノードには、パッシブな接続があります。

バグを報告します。

10.4.11. 2.x と 3.x 間のブローカーフェデレーションの制限

バージョン 2.x からバージョン 3.x ブローカーフェデレーションのサポートは、引き続き長期的な 0-10 フェデレーション機能によって提供されます。2.x および 3.x ブローカーの両方をフェデレーション用に管理するには、バージョン 3.x をプロビジョニングして 2.x ブローカーとの後方互換性を確保する **qpid-route** 必要があります。これは、2.x ブローカーの引数カウントの不一致処理の相違点により生じます。

ソースブローカーが新しいバージョンを使用するリンクを作成 **qpid-route** しようとすると、2.x バージョンの qpid-route に改善された引数カウント処理が含まれていないため、互換性が破損します。

バグを報告します。

第11章 QPID JCA ADAPTER

11.1. JCA ADAPTER

JCA Adapter は、エンタープライズ情報システム（メインフレームトランザクション処理やデータベースシステムなど）、アプリケーションサーバー、およびエンタープライズアプリケーション間のアウトバウンド接続とインバウンド接続を提供します。これは、Message-Driven Bean(MDB)へのメッセージのインフローと、他の Java EE コンポーネントから送信されるメッセージの送信を制御します。また、メッセージングアプリケーションを微調整するためのさまざまなオプションも提供します。

バグを報告します。

11.2. QPID JCA ADAPTER

Qpid Resource Adapter は Java Connector Architecture(JCA)1.5 準拠のリソースアダプターで、EE アプリケーションと AMQP 0.10 メッセージブローカー間の Java EE 統合を許可します。現在、アダプターは C++ ベースのブローカーのみをサポートし、Apache Qpid C++ ブローカーでのみテストされています。

バグを報告します。

11.3. QPID JCA ADAPTER のインストール

Red Hat Enterprise Linux 5 または 6 を実行しているシステムの場合は、Qpid JCA Adapter は **qpid-jca** および **qpid-jca-xarecovery** パッケージで提供されます。これらの RPM パッケージは、デフォルトの MRG Messaging インストールに含まれています。

その他のオペレーティングシステムでは、Qpid JCA Adapter が **JCA Adapter <JCA-VERSION>** および **JCA Adapter <JCA-VERSION> detached signature** パッケージで提供されます。これらの ZIP ファイルは、Red Hat Network の **MRG Messaging v. 2 (for non-Linux platforms)** チャネル **Downloads** セクションから取得できます。

バグを報告します。

11.4. QPID JCA ADAPTER 設定

11.4.1. アプリケーションごとのサーバー設定情報

リソースアダプターのコンポーネントのデプロイは、アプリケーションサーバーによって異なります。本ガイドでは、アダプターの機能の概要と、JBoss Enterprise Application Platform 5 および 6 のインストール手順を説明します。ほとんどのアプリケーションサーバープラットフォームでは、アダプターを設定するためのプラットフォーム固有の詳細は、通常という名前の README ファイルで提供され **README-<server-platform>.txt** ます。

関連項目

- 「JBoss EAP 5 での Qpid JCA Adapter のデプロイ」
- 「JBoss EAP 6 での Qpid JCA Adapter のデプロイ」

バグを報告します。

11.4.2. JCA Adapter ra.xml 設定

ra.xml** ファイルには JCA Adapter の設定パラメーターが含まれます。一部のアプリケーションサーバー環境では、このファイルは設定を直接変更するために直接編集されます（JBoss EAP 5 など）、-ds.xml** ファイルの設定によって上書きされます。*

*以下は、設定または上書きが可能な **ra.xml** ファイルに設定されたプロパティーです。*

表11.1 ResourceAdapter プロパティー

パラメーター	description	デフォルト値
ClientId	接続のクライアント ID	client_id
SetupAttempts	失敗前の設定試行回数	5
SetupInterval	設定試行の間隔（ミリ秒単位）	5000
UseLocalTx	XA ではなくローカル翻訳を使用する	false
Host	ブローカーホスト	localhost
Port	ブローカーポート	5672
Path	接続ファクトリーの仮想パス	テスト
ConnectionURL	接続 URL	amqp://anonymous:passwd@client/test?brokerlist='tcp://localhost?sasl_mechs='PLAIN''
UseConnectionPerHandler	MessageHandler ごとの JMS 接続の使用	true

表11.2 アウトバウンド ResourceAdapter プロパティー

パラメーター	description	デフォルト値
SessionDefaultType	デフォルトのセッションタイプ	javax.jms.Queue
UseTryLock	ロックタイムアウトを秒単位で指定する	0
UseLocalTx	XA ではなくローカルトランザクションを使用	false
ClientId	接続のクライアント ID	client_id
ConnectionURL	接続 URL	
Host	ブローカーホスト	localhost

パラメーター	description	デフォルト値
Port	ブローカーポート	5672
Path	接続ファクトリーの仮想パス	テスト

Inbound ResourceAdapter と AdminObjects

*また、**ra.xml** ファイルには、インバウンドリソースアダプターと管理されたオブジェクト (**AdminObject**) の設定が含まれます。これらの 2 つの設定は、リソースアダプターおよびアウトバウンドリソースアダプターの設定とは異なります。リソースアダプターおよびアウトバウンドリソースアダプター設定は、**ra.xml** ファイルの設定パラメーターの値を設定します。Inbound Resource Adapter と Admin Object は設定パラメーターを **定義** しますが、設定は設定しません。管理されたオブジェクト mbean 定義の責任で、に定義されたプロパティーを設定し **ra.xml** ます。*

表11.3 管理オブジェクトプロパティー

管理オブジェクトクラス	property
org.apache.qpid.ra.admin.QpidQueue	**DestinationAddress**
org.apache.qpid.ra.admin.QpidTopic	**DestinationAddress**
org.apache.qpid.ra.admin.QpidConnectionFactoryProxy	**ConnectionURL**

関連項目

- *「JBoss EAP 5 での Qpid JCA Adapter のデプロイ」*
- *「JBoss EAP 6 での Qpid JCA Adapter のデプロイ」*

バグを報告します。

11.4.3. トランザクションサポート

*Qpid JCA Adapter は、3 レベルのトランザクションサポート（および）を提供 **XA LocalTransactions** し **NoTransaction** ます。*

通常、Qpid JCA Adapter を使用すると XA トランザクションを使用することになりますが、これが常に当てはまるわけではありません。トランザクションサポート設定は、各アプリケーションサーバーに固有のものです。サポートされるアプリケーションサーバーごとに、ドキュメントを参照してください。

バグを報告します。

11.4.4. トランザクションの制限

Qpid JCA Adapter には以下のような制限があります。

1. *Qpid C++ ブローカーは、クラスター化されたブローカー内の XA の使用をサポートしません。クラスターを実行している場合は、アダプターが LocalTransactions を使用するように設定する必要があります。*

2. *xaRecovery は現在実装されていません。システムに障害が発生した場合は、管理者またはその他の認定済みの担当者が、不完全な（または **疑わしい**）トランザクションを手動で解決する必要があります。*

バグを報告します。

11.5. JBOSS EAP 5 での QPID JCA ADAPTER のデプロイ

11.5.1. JBoss EAP 5 での Qpid JCA アダプターのデプロイ

JBoss EAP 5 で使用する Qpid JCA Adapter をインストールするには、設定ファイルを JBoss デプロイディレクトリーに転送します。

手順11.1 JBoss EAP の Qpid JCA アダプターのデプロイ

1. ***qpid-ra-<version>.rar** ファイルを見つけます。リソースアダプター、Qpid Java クライアントファイル、および **.jar META-INF** ディレクトリーが含まれる zip アーカイブデータファイルです。*

2. ***qpid-ra-<version>.rar** ファイルを JBoss デプロイディレクトリーにコピーします。JBoss デプロイディレクトリーはで **JBOSS_ROOT/server/<server-name>/deploy**、JBOSS_ROOT は JBoss インストールのルートディレクトリーを示し、<server-name> はデプロイメントサーバーの名前を表します。*

3. *アダプターのインストールに成功すると、次のメッセージが表示されます。*

   ```
   INFO  [QpidResourceAdapter] Qpid resource adaptor started
   ```

   この時点で、アダプターはデプロイされ、設定が可能です。

バグを報告します。

11.5.2. JBoss EAP 5 での JCA 設定

11.5.2.1. JCA アダプター設定ファイル

変更

- *2013 年 4 月更新されました。*

*EAP 5.x 環境の JCA アダプターの標準設定メカニズムは ***-ds.xml** ファイルです。Qpid JCA アダプターには JCA 仕様ごとにグローバル **ra.xml** ファイルがありますが、このファイル内のデフォルトの値セットは常にファイル設定 ***-ds.xml** ファイルを介して上書きされます。*

ResourceAdapter** 設定は、インバウンドおよびアウトバウンド接続の汎用プロパティーを提供します。ただし、標準の JBoss 設定アーティファクト、-ds.xml** ファイル、**ManagedConnectionFactories** および MDB ActivationSpec を使用してデプロイおよびインバウンドアクティベーションのデプロイおよびインバウンドアクティベーションを行う際に、これらのプロパティーを上書きできます。サンプル ***-ds.xml** ファイル **qpid-jca-ds.xml**はディレクトリーにあり **/usr/share/doc/qpid-jca-<VERSION>/example/conf**/ます。*

Qpid JCA Adapter ディレクトリーには一般的なファイルが **/usr/share/qpid-jca** 含まれています。この **README.txt** ファイルには、Qpid JCA Adapter に関連付けられたすべてのプロパティーが詳細に説明されています。

バグを報告します。

11.5.2.2. ConnectionFactory の設定

11.5.2.2.1. ConnectionFactory

JCA 仕様に準拠する ConnectionFactory コンポーネントは、標準アウトバウンド接続のプロパティーを定義します。

バグを報告します。

11.5.2.2.2. EAP 5 の ConnectionFactory 設定

JBoss EAP 5 では、ConnectionFactories は ***-ds.xml** ファイルを使用して設定されます。ディストリビューションには、サンプルファイル(**qpid-jca-ds.xml**)が提供されます。このファイルは、開発やデプロイメントのニーズに合わせて変更できます。

バグを報告します。

11.5.2.2.3. XAConnectionFactory の例

以下の例は、XA トランザクションのサンプルファイルの ConnectionFactory の部分を示しています。

```
<tx-connection-factory>
  <jndi-name>QpidJMSXA</jndi-name>
  <xa-transaction/>
  <rar-name>qpid-ra-<ra-version>.rar</rar-name>
  <connection-definition>org.apache.qpid.ra.QpidRAConnectionFactory</connection-definition>
  <config-property name="ConnectionURL">amqp://guest:guest@/test?
brokerlist='tcp://localhost:5672?sasl_mechs='ANONYMOUS''</config-property>
  <max-pool-size>20</max-pool-size>
</tx-connection-factory>
```

- **QpidJMSXA** 接続ファクトリーは XA 対応 ManagedConnectionFactory を定義します。
- プロパティーに特定の ra バージョンを挿入する必要があり **rar-name** ます。
- **jndi-name** および **ConnectionURL** プロパティーは、お使いの環境に応じて変更できます。

デプロイ後、ConnectionFactory は以下の構文で Java Naming and Directory Interface(JNDI)にバインドされます。

```
java:<jndi-name>
```

この例では、以下のように解決します。

```
java:QpidJMSXA
```

バグを報告します。

11.5.2.2.4. ローカル接続ファクトリーの例

以下の例は、ローカルトランザクションのサンプルファイルの **ConnectionFactory** 一部を示しています。

```
<tx-connection-factory>
  <jndi-name>QpidJMS</jndi-name>
  <rar-name>
    qpid-ra-<ra-version>.rar
  </rar-name>
  <local-transaction/>
  <config-property name="useLocalTx" type="java.lang.Boolean">
    true
  </config-property>
  <config-property name="ConnectionURL">
amqp://anonymous:@client/test?brokerlist='tcp://localhost:5672?sasl_mechs='ANONYMOUS''
  </config-property>
  <connection-definition>
    org.apache.qpid.ra.QpidRAConnectionFactory
  </connection-definition>
  <max-pool-size>20<max-pool-size>
</tx-connection-factory>
```

- **QpidJMS** 接続ファクトリーは、非 XA を定義し **ConnectionFactory**ます。通常、これは XA が望ましくない **ConnectionFactory** 場所や、現在 XA に対応していないクラスター化された Qpid Broker 設定で実行している場合に使用します。
- プロパティーに特定の ra バージョンを挿入する必要があり **rar-name** ます。
- **jndi-name** および **ConnectionURL** プロパティーは、お使いの環境に応じて変更できます。

デプロイメント後、以下の構文 **ConnectionFactory** で Java Naming and Directory Interface(JNDI) にバインドされます。

```
java:<jndi-name>
```

この例では、以下のように解決します。

```
java:QpidJMS
```

バグを報告します。

11.5.2.3. 管理されたオブジェクトの設定

11.5.2.3.1. EAP 5 で管理されたオブジェクト

宛先（キュー、トピック）は JCA の標準管理オブジェクト(AdminObject) を使用して JBoss EAP で設定されます。これらのオブジェクトは、ConnectionFactory 設定と共に ***-ds.xml** ファイル内に置かれます。サンプル **qpid-jca-ds.xml** ファイルは、JMS Queue/トピックと接続ファクトリーの 2 つのオブジェクトを提供します。

バグを報告します。

11.5.2.3.2. JMS キューの管理オブジェクトの例

変更

- *2013 年 4 月更新されました。*

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=HelloQueue">
  <attribute name="JNDIName">HelloQueue</attribute>
  <depends optional-attribute-name="RARName">
    jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
  </depends>
  <attribute name="Type">
    org.apache.qpid.ra.admin.QpidQueue
  </attribute>
  <attribute name="Properties">
     DestinationAddress=amq.direct
  </attribute>
</mbean>
```

上記の XML は、JNDI にバインドされる JMS キューを以下のように定義します。

```
HelloQueue
```

*この宛先は JNDI から取得でき、メッセージの消費または実稼働に使用されます。**DestinationAddress** プロパティーは環境に合わせてカスタマイズできます。特定の設定オプションについては、Qpid Java Client のドキュメントを参照してください。*

バグを報告します。

11.5.2.3.3. JMS トピック管理オブジェクトの例

変更

- *2013 年 4 月更新されました。*

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=HelloTopic">
  <attribute name="JNDIName">
    HelloTopic
  </attribute>
  <depends optional-attribute-name="RARName">
    jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
  </depends>
  <attribute name="Type">
    org.apache.qpid.ra.admin.QpidTopic
  </attribute>
  <attribute name="Properties">
     DestinationAddress=amq.topic
  </attribute>
</mbean>
```

上記の XML は、JNDI にバインドされる JMS トピックを以下のように定義します。

```
HelloTopic
```

この宛先は JNDI から取得でき、メッセージの消費または実稼働に使用されます。**DestinationAddress** プロパティーは環境に合わせてカスタマイズできます。特定の設定オプションについては、Qpid Java Client のドキュメントを参照してください。

バグを報告します。

11.5.2.3.4. ConnectionFactory 管理オブジェクトの例

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=QpidConnectionFactory">
  <attribute name="JNDIName">
    QpidConnectionFactory
  </attribute>
  <depends optional-attribute-name="RARName">
    jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
  </depends>
  <attribute name="Type">
    javax.jms.ConnectionFactory
  </attribute>
  <attribute name="Properties">
    ConnectionURL=amqp://anonymous:@client/test?brokerlist='tcp://localhost:5672?
sasl_mechs='ANONYMOUS''
  </attribute>
</mbean>
```

上記の XML は、JBoss EAP 5 およびその他の外部クライアントに使用できる ConnectionFactory を定義します。通常、この接続ファクトリーは、アプリケーションサーバーを必要としないスタンドアロンまたは「シン」クライアントによって使用されます。このオブジェクトは、以下のように JBoss EAP 5 の JNDI ツリーにバインドされます。

```
QpidConnectionFactory
```

バグを報告します。

11.6. JBOSS EAP 6 での QPID JCA ADAPTER のデプロイ

11.6.1. JBoss EAP 6 での Qpid JCA Adapter のデプロイ

JBoss EAP 6 で使用する Qpid JCA Adapter をインストールするには、Qpid JCA Adapter 設定ファイルを JBoss デプロイディレクトリーにコピーします。

手順11.2 JBoss EAP の Qpid JCA アダプターのデプロイ

1. **qpid-ra-<version>.rar** ファイルを見つけます。リソースアダプター、Qpid Java クライアントファイル、および **.jar META-INF** ディレクトリーが含まれる zip アーカイブデータファイルです。

2. **qpid-ra-<version>.rar** ファイルを JBoss デプロイディレクトリーにコピーします。JBoss デプロイディレクトリーは **JBOSS_ROOT/<server-config>/deployments**、JBOSS_ROOT が JBoss インストールのルートディレクトリーで、<server-config> はデプロイメントサーバー設定の名前になります。

JCA Adapter のデプロイ時に、JCA アダプターを使用する前に設定する必要があります。

バグを報告します。

11.6.2. JBoss EAP 6 での JCA 設定

11.6.2.1. JBoss EAP 6 の JCA Adapter 設定ファイル

JBoss EAP 6.x では、これまでの EAP バージョンとは異なる設定スキームが使用されます。

各サーバー設定タイプには、ディレクトリーに以下の設定ファイルが含まれ **JBOSS_ROOT/<server-config>/configuration** ます。

- <server-config>-full.xml
- <server-config>-full-ha.xml
- <server-config>.xml

各ファイルはケイパビリティーセットに対応し、そのプロファイルのサブシステムの設定が含まれます。

バグを報告します。

11.6.2.2. デフォルトのメッセージングプロバイダーを Qpid JCA Adapter に置き換えます。

サーバー設定ファイルからの以下の XML フラグメントは、デフォルトの EAP 6 メッセージングプロバイダーに代わるものです。

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
  <session-bean>
    <stateless>
      <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
    </stateless>
    <stateful default-access-timeout="5000" cache-ref="simple"/>
    <singleton default-access-timeout="5000"/>
  </session-bean>
  <mdb>
    <resource-adapter-ref resource-adapter-name="qpid-ra-<rar-version>.rar"/>
    <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
  </mdb>
  <pools>
    <bean-instance-pools>
      <strict-max-pool name="slsb-strict-max-pool" max-pool-size="20" instance-acquisition-
timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
      <strict-max-pool name="mdb-strict-max-pool" max-pool-size="20" instance-acquisition-
timeout="5" instance-acquisition-timeout-unit="MINUTES"/>
    </bean-instance-pools>
  </pools>
  <caches>
    <cache name="simple" aliases="NoPassivationCache"/>
    <cache name="passivating" passivation-store-ref="file" aliases="SimpleStatefulCache"/>
  </caches>
  <passivation-stores>
    <file-passivation-store name="file"/>
  </passivation-stores>
  <async thread-pool-name="default"/>
```

```
    <timer-service thread-pool-name="default">
      <data-store path="timer-service-data" relative-to="jboss.server.data.dir"/>
    </timer-service>
    <remote connector-ref="remoting-connector" thread-pool-name="default"/>
    <thread-pools>
      <thread-pool name="default">
        <max-threads count="10"/>
        <keepalive-time time="100" unit="milliseconds"/>
      </thread-pool>
    </thread-pools>
  </subsystem>
```

このサブシステム設定の関連行は以下のとおりです。

```
  <mdb>
    <resource-adapter-ref resource-adapter-name="qpid-ra-<rar-version>.rar"/>
    <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
  </mdb>
```

バグを報告します。

11.6.2.3. 設定方法

以下の 2 つのオプションがあります。

1. 既存の設定ファイルを直接編集します。
2. 既存の設定ファイルをコピーしてコピーし、コマンドで新しい設定ファイルを使用してサーバーを起動します。

```
JBOSS_HOME/bin/standalone.sh -c your-modified-config.xml
```

バグを報告します。

11.6.2.4. 最小 EAP 6 の設定例

JBoss EAP 6 サーバー設定ファイルからの以下の XML フラグメントは最小限の設定例で、XA 対応 ManagedConnectionFactory と 2 つの JMS 宛先（キューとトピック）を設定します。

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.0">
  <resource-adapters>
    <resource-adapter>
      <archive>
        qpid-ra-<rar-version>.rar
      </archive>
      <transaction-support>
        XATransaction
      </transaction-support>
      <config-property name="connectionURL">
        amqp://anonymous:passwd@client/test?brokerlist='tcp://localhost?sasl_mechs='PLAIN''
      </config-property>
      <config-property name="TransactionManagerLocatorClass">
        org.apache.qpid.ra.tm.JBoss7TransactionManagerLocator
      </config-property>
```

```
            <config-property name="TransactionManagerLocatorMethod">
                getTm
            </config-property>
            <connection-definitions>
                <connection-definition class-
name="org.apache.qpid.ra.QpidRAManagedConnectionFactory" jndi-name="QpidJMSXA" pool-
name="QpidJMSXA">
                    <config-property name="connectionURL">
                        amqp://anonymous:passwd@client/test?brokerlist='tcp://localhost?
sasl_mechs='PLAIN''
                    </config-property>
                    <config-property name="SessionDefaultType">
                        javax.jms.Queue
                    </config-property>
                </connection-definition>
            </connection-definitions>
            <admin-objects>
                <admin-object class-name="org.apache.qpid.ra.admin.QpidTopicImpl" jndi-
name="java:jboss/exported/GoodByeTopic" use-java-context="false" pool-name="GoodByeTopic">
                    <config-property name="DestinationAddress">
                        amq.topic/hello.Topic
                    </config-property>
                </admin-object>
                <admin-object class-name="org.apache.qpid.ra.admin.QpidQueueImpl" jndi-
name="java:jboss/exported/HelloQueue" use-java-context="false" pool-name="HelloQueue">
                    <config-property name="DestinationAddress">
                        hello.Queue;{create:always, node:{type:queue, x-declare:{auto-delete:true}}}
                    </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

バグを報告します。

11.6.2.5. その他のリソース

詳細は、JBoss Enterprise Application Platform ドキュメントおよび Qpid JCA Adapter に含まれる README ファイルを参照してください。

バグを報告します。

第12章 管理ツールおよびコンソール

12.1. コマンドラインユーティリティー

12.1.1. コマンドライン管理ユーティリティー

コマンドラインツールは、シェルプロンプトで使用する管理および診断ツールです。

表12.1 コマンドライン管理ユーティリティー

ユーティリティー	description
qpid-config	ブローカーでの交換、キュー、バインディングの表示と設定
qpid-tool	ブローカー内のアクセス設定、統計、および制御
qpid-queue-stats	ブローカーのキューのサイズおよびキュー/デキューレートの監視
qpid-ha	クラスターの設定および表示
qpid-route	ブローカー間のフェデレーションルートの設定
qpid-stat	さまざまなブローカーオブジェクトの詳細および統計の表示
qpid-printevents	ブローカーからイベントにサブスクライブし、コンソールウィンドウに発生したイベントの詳細を表示します。

バグを報告します。

12.1.2. qpid-config の使用

1. シェルプロンプト **qpid-config --help** からコマンドを実行し、コマンドの詳細の一覧を表示します。

```
$ qpid-config --help

Usage:  qpid-config [OPTIONS]
qpid-config [OPTIONS] exchanges [filter-string]
qpid-config [OPTIONS] queues    [filter-string]
qpid-config [OPTIONS] add exchange <type> <name> [AddExchangeOptions]
qpid-config [OPTIONS] del exchange <name>
..[output truncated]...
```

2. オプション **qpid-config** なしを使用して、すべての交換およびキューの概要を表示します。

```
$ qpid-config
```

```
Total Exchanges: 6
  topic: 2
headers: 1
 fanout: 1
 direct: 2
  Total Queues: 7
durable: 0
   non-durable: 7
```

3. 以下の **queues** コマンドを使用して、既存のすべてのキューの情報を一覧表示します。

```
$ qpid-config queues
Queue Name                                Attributes
================================================================
my-queue                                  --durable
qmfc-v2-hb-localhost.localdomain.20293.1  auto-del excl --limit-policy=ring
qmfc-v2-localhost.localdomain.20293.1     auto-del excl
qmfc-v2-ui-localhost.localdomain.20293.1  auto-del excl --limit-policy=ring
reply-localhost.localdomain.20293.1       auto-del excl
topic-localhost.localdomain.20293.1       auto-del excl --limit-policy=ring
```

4. **add queue** コマンドと、作成するキューの名前を使用して、新しいキューを追加します。

```
$ qpid-config add queue queue_name
```

5. キューを削除するには、削除するキューの名前を付けて **del queue** コマンドを実行します。

```
$ qpid-config del queue queue_name
```

6. **exchanges** コマンドで、既存のすべての交換の情報を一覧表示します。**-r** オプション ("recursive") を追加して、バインディング情報も表示します。

```
$ qpid-config -r exchanges

Exchange '' (direct)
 bind pub_start => pub_start
 bind pub_done => pub_done
 bind sub_ready => sub_ready
 bind sub_done => sub_done
 bind perftest0 => perftest0
 bind mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => mgmt-3206ff16-fb29-4a30-82ea-
e76f50dd7d15
 bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-4a30-82ea-
e76f50dd7d15
Exchange 'amq.direct' (direct)
 bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-4a30-82ea-
e76f50dd7d15
 bind repl-df06c7a6-4ce7-426a-9f66-da91a2a6a837 => repl-df06c7a6-4ce7-426a-9f66-
da91a2a6a837
 bind repl-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae => repl-c55915c2-2fda-43ee-9410-
b1c1cbb3e4ae
Exchange 'amq.topic' (topic)
Exchange 'amq.fanout' (fanout)
```

```
Exchange 'amq.match' (headers)
Exchange 'qpid.management' (topic)
 bind mgmt.# => mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15
```

7. **add exchange** コマンドで新しい交換を追加します。作成する交換の名前とともに、タイプ（ダイレクト、トピック、またはジャンクアウト）を指定します。交換を永続的にするには、**--durable** オプションを追加することもできます。

```
$ qpid-config add exchange direct exchange_name --durable
```

8. 交換を削除するには、削除する交換の名前を指定して **del exchange** コマンドを実行します。

```
$ qpid-config del exchange exchange_name
```

バグを報告します。

12.1.3. qpid-tool の使用

1. ブローカーへの接続が **qpid-tool** 作成され、コマンドはシェルプロンプト自体ではなく、ツール内で実行されます。接続を作成するには、表示するブローカーを実行しているマシンの名前または IP アドレスを指定して、シェルプロンプト **qpid-tool** で実行します。TCP ポート番号を **:** 文字で追加することもできます。

```
$ qpid-tool localhost

Management Tool for QPID
qpid:
```

2. 接続に成功すると、qpid-tool に **qpid:** プロンプトが表示されます。このプロンプト **help** で入力して、コマンドの詳細の一覧を表示します。

```
qpid: help
Management Tool for QPID

Commands:
list                    - Print summary of existing objects by class
list <className>        - Print list of objects of the specified class
list <className> all    - Print contents of all objects of specified class
...[output truncated}...
```

3. **qpid-tool** オブジェクト という用語を使用して、キュー、交換、ブローカー、およびその他のデバイスを参照します。既存のオブジェクトの一覧を表示するには、プロンプト **list** で以下を入力します。

```
# qpid-tool
Management Tool for QPID
qpid: list
Summary of Objects by Type:
   Package                Class         Active  Deleted
   ======================================================
   org.apache.qpid.broker  exchange      8       0
   org.apache.qpid.broker  broker        1       0
   org.apache.qpid.broker  binding       16      12
   org.apache.qpid.broker  session       2       1
```

```
org.apache.qpid.broker  connection    2       1
org.apache.qpid.broker  vhost         1       0
org.apache.qpid.broker  queue         6       5
org.apache.qpid.broker  system        1       0
org.apache.qpid.broker  subscription  6       5
```

4. クラスを指定することで、リストするオブジェクトを選択できます。

```
qpid: list system
Object Summary:
  ID   Created   Destroyed  Index

=====================================================================

  167  07:34:13  -          UUID('b3e2610e-5420-49ca-8306-dca812db647f')
```

5. オブジェクトクラスの詳細を表示するには、**schema** コマンドを使用してクラスを指定します。

```
qpid: schema queue
Schema for class 'qpid.queue':
Element              Type         Unit         Access      Notes   Description
=====================================================================
=======================================
vhostRef             reference                 ReadCreate  index
name                 short-string              ReadCreate  index
durable              boolean                   ReadCreate
autoDelete           boolean                   ReadCreate
exclusive            boolean                   ReadCreate
arguments            field-table               ReadOnly          Arguments supplied in
queue.declare
storeRef             reference                 ReadOnly          Reference to persistent queue (if
durable)
msgTotalEnqueues     uint64       message                        Total messages enqueued
msgTotalDequeues     uint64       message                        Total messages dequeued
msgTxnEnqueues       uint64       message                        Transactional messages
enqueued
msgTxnDequeues       uint64       message                        Transactional messages
dequeued
msgPersistEnqueues   uint64       message                        Persistent messages enqueued
msgPersistDequeues   uint64       message                        Persistent messages dequeued
msgDepth             uint32       message                        Current size of queue in messages
msgDepthHigh         uint32       message                        Current size of queue in
messages (High)
msgDepthLow          uint32       message                        Current size of queue in
messages (Low)
byteTotalEnqueues    uint64       octet                          Total messages enqueued
byteTotalDequeues    uint64       octet                          Total messages dequeued
byteTxnEnqueues      uint64       octet                          Transactional messages enqueued
byteTxnDequeues      uint64       octet                          Transactional messages dequeued
bytePersistEnqueues  uint64       octet                          Persistent messages enqueued
bytePersistDequeues  uint64       octet                          Persistent messages dequeued
byteDepth            uint32       octet                          Current size of queue in bytes
byteDepthHigh        uint32       octet                          Current size of queue in bytes (High)
byteDepthLow         uint32       octet                          Current size of queue in bytes (Low)
```

```
enqueueTxnStarts      uint64      transaction          Total enqueue transactions
started
enqueueTxnCommits     uint64      transaction          Total enqueue transactions
committed
enqueueTxnRejects     uint64      transaction          Total enqueue transactions
rejected
enqueueTxnCount       uint32      transaction          Current pending enqueue
transactions
enqueueTxnCountHigh   uint32      transaction          Current pending enqueue
transactions (High)
enqueueTxnCountLow    uint32      transaction          Current pending enqueue
transactions (Low)
dequeueTxnStarts      uint64      transaction          Total dequeue transactions
started
dequeueTxnCommits     uint64      transaction          Total dequeue transactions
committed
dequeueTxnRejects     uint64      transaction          Total dequeue transactions
rejected
dequeueTxnCount       uint32      transaction          Current pending dequeue
transactions
dequeueTxnCountHigh   uint32      transaction          Current pending dequeue
transactions (High)
dequeueTxnCountLow    uint32      transaction          Current pending dequeue
transactions (Low)
consumers             uint32      consumer             Current consumers on queue
consumersHigh         uint32      consumer             Current consumers on queue
(High)
consumersLow          uint32      consumer             Current consumers on queue
(Low)
bindings              uint32      binding              Current bindings
bindingsHigh          uint32      binding              Current bindings (High)
bindingsLow           uint32      binding              Current bindings (Low)
unackedMessages       uint32      message              Messages consumed but not yet
acked
unackedMessagesHigh   uint32      message              Messages consumed but not
yet acked (High)
unackedMessagesLow    uint32      message              Messages consumed but not
yet acked (Low)
messageLatencySamples delta-time  nanosecond           Broker latency through this
queue (Samples)
messageLatencyMin     delta-time  nanosecond           Broker latency through this
queue (Min)
messageLatencyMax     delta-time  nanosecond           Broker latency through this
queue (Max)
messageLatencyAverage delta-time  nanosecond           Broker latency through this
queue (Average)
```

6. ツールを終了してシェルに戻るには、プロンプト **quit** で以下を入力します。

```
qpid: quit
Exiting...
```

バグを報告します。

12.1.4. qpid-queue-stats の使用

*コマンドを実行し **qpid-queue-stats** てツールを起動します。*

このツールは、以下の形式で現在のマシンでブローカーのstats を報告します。

```
Queue Name                           Sec      Depth     Enq Rate     Deq Rate
================================================================================
======
message_queue                       10.00     11224       0.00       54.01
qmfc-v2-ui-radhe.26001.1              10.00         0       0.10       0.10
topic-radhe.26001.1                 10.00         0       0.20       0.20
message_queue                       10.01      9430       0.00      179.29
qmfc-v2-ui-radhe.26001.1              10.01         0       0.10       0.10
topic-radhe.26001.1                 10.01         0       0.20       0.20
```

ブローカーのキューは左側に一覧表示されます。Sec 列はサンプルレートです。このツールは10 秒間隔でブローカーをポーリングするようにハードコーディングされます。Depth 列はキュー内のメッセージ数を報告します。enq Rate は、最後のサンプルからキューに追加されたメッセージの数です。Deq Rate は、最後のサンプルからキューから削除されたメッセージの数です。

*別のサーバーのキューを表示するには、**-a** スイッチを使用してリモートサーバーアドレスを指定します。また、任意でリモートポートと認証情報を指定します。*

*たとえば、アドレスを使用してリモートサーバーのキューを確認するには **192.168.1.145**、以下のコマンドを実行します。*

```
qpid-queue-stats -a 192.168.1.145
```

*サーバーのキューを確認するには、次のコマンドを **broker1.mydomain.com**実行します。*

```
qpid-queue-stats -a broker1.mydomain.com
```

*ブローカーがポート 8888 で実行されているサーバーのキューを確認するには **broker1.mydomain.com**、次のコマンドを実行します。*

```
qpid-queue-stats -a broker1.mydomain.com:8888
```

*認証が必要なサーバーのキューを確認するには **192.168.1.145**、次のコマンドを実行します。*

```
qpid-queue-stats -a username/password@192.168.1.145
```

バグを報告します。

付録A 交換およびキュー宣言引数

A.1. EXCHANGE およびキュー引数のリファレンス

変更

- ***qpid.last_value_queue qpid.last_value_queue_no_browse*** *非推奨となり、削除されます。*
- ***qpid.msg_sequence*** *置き換えられ* ***qpid.queue_msg_sequence****たキュー引数。*
- ***ring_strict*** *および* ***flow_to_disk*** *が有効な* ***qpid.policy_type*** *値でなくなりました。*
- ***qpid.persist_last_node*** *非推奨となり、削除されています。*

以下は、キューと交換を宣言するための引数の完全なリストです。

交換オプション

qpid.exclusive-binding *(bool)*

指定のバインディングキーが1 つのキューのみに関連付けられるようにします。

qpid.ive *(bool)*

if がに設定します。 「true」交換は 初期値交換で、他の交換 と異なります。交換に送信された最後のメッセージがキャッシュされ、新しいキューが交換にバインドされている場合は、メッセージがバインディング基準に適合する場合、このメッセージをキューにルーティングしようとします。これにより、新しいキューが最後に受信したメッセージを初期値として使用することができます。

qpid.msg_sequence *(bool)*

if がに設定します。 「true」交換によって、という名前のシーケンス番号が挿入されます。 「qpid.msg_sequence」 各メッセージのメッセージヘッダーに切り替えます。このシーケンス番号のタイプは int64 です。交換から最初のメッセージのシーケンス番号は1 で、後続のメッセージごとに順番に増分されます。qpid ブローカーの再起動時にシーケンス番号が1 にリセットされます。

qpid.sequence_counter *(int64)*

指定の番号で ***qpid.msg_sequence*** *カウントを開始します。*

キューオプション

no-local *(bool)*

キューがキューを宣言するのと同じ接続のセッションによってキューに格納されたメッセージを破棄することを指定します。

qpid.alert_count *(uint32_t)*

キューメッセージ数がこのサイズを超える場合は、アラートを送信する必要があります。

qpid.alert_repeat_gap *(int64_t)*

イベント間の最小間隔を秒単位で制御します。デフォルト値は 60 秒です。

qpid.alert_size *(int64_t)*

キューサイズ（バイト単位）がこの値を超えると、アラートが送信されます。

***qpid.auto_delete_timeout** (bool)*

キューが自動的に削除されるように設定されている場合、キューはここで指定されている秒数後に削除されます。

***qpid.browse-only** (bool)*

キューのすべてのユーザーは、参照を強制されます。リング、LVQ、またはTTL でキューのサイズを制限します。この引数名はアンダースコアではなくハイフンを使用することに注意してください。

***qpid.file_count** (int)*

キューの永続ジャーナルにファイル数を設定します。デフォルト値は8 です。

***qpid.file_size** (int64)*

ファイルのページ数を設定します（各ページは64 KB です）。デフォルト値は24 です。

***qpid.flow_resume_count** (uint32_t)*

フローはしきい値をメッセージ数として再開します。

***qpid.flow_resume_size** (uint64_t)*

フローは、バイト単位でしきい値を再開します。

***qpid.flow_stop_count** (uint32_t)*

フローがメッセージ数としてしきい値を停止します。

***qpid.flow_stop_size** (uint64_t)*

フロー停止のしきい値（バイト単位）。

qpid.last_value_queue_key （文字列）

最後の値キューに使用するキーを定義します。

***qpid.max_count** (uint32_t)*

***policy_type** が判断されるアクションの前にキューに格納できるメッセージデータの最大バイトサイズ。*

***qpid.max_size** (uint64_t)*

*が決定するアクションの前にキューに含まれること **policy_type** ができるメッセージの最大数。*

qpid.policy_type （文字列）

*キューサイズを制御するデフォルトの動作を設定します。有効な値は **reject** およびです **ring**。*

***qpid.priorities** (size_t)*

キューが認識する固有の優先度レベルの数（最大10）。デフォルト値は1 レベルです。

qpid.queue_msg_sequence （文字列）

指定された名前のカスタムヘッダーをキューに格納されたメッセージに追加します。このヘッダーには、シーケンス番号が自動的に設定されます。

qpid.trace.exclude （文字列）

指定の（コンマ区切り）トレース ID の 1 つを含むメッセージを送信しません。

qpid.trace.id （文字列）

*指定されたトレース ID を、キューから送信されたメッセージのアプリケーションヘッダー**x-qpid.trace**"" に追加します。*

x-qpid-maximum-message-count

*これはのエイリアスです **qpid.alert_count**。*

x-qpid-maximum-message-size

*これはのエイリアスです **qpid.alert_size**。*

x-qpid-minimum-alert-repeat-gap

*これはのエイリアスです **qpid.alert_repeat_gap**。*

x-qpid-priorities

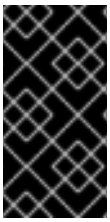
*これはのエイリアスです **qpid.priorities**。*

バグを報告します。

付録B OPENSSL CERTIFICATE REFERENCE

B.1. 証明書の参照

openssl コマンドで証明書を作成および管理するためのこの参照は、SSL に精通していることを前提としています。SSL に関する背景情報は、OpenSSL ドキュメント(www.openssl.org)を参照してください。



重要

認証局(CA)が署名した証明書のみがセキュアなシステムに使用することが推奨されます。本セクションの自己署名証明書を生成する手順は、認証された CA から証明書を待つ一方で、テストおよび開発のアクティビティーまたはソフトウェアの評価を容易に行うことが意図されています。

証明書の生成

手順B.1 秘密鍵の作成

- このコマンドを使用して、ファイル暗号化で1024 ビット RSA 秘密鍵を生成します。キーファイルが暗号化されている場合は、アプリケーションが秘密鍵にアクセスするたびにパスワードが必要になります。

```
# openssl genrsa -des3 -out mykey.pem 1024
```

このコマンドを使用して、ファイルの暗号化なしでキーを生成します。

```
# openssl genrsa  -out mykey.pem 1024
```

手順B.2 自己署名証明書の作成

以下のコマンドはそれぞれ、新しい秘密鍵と 自己署名 証明書を生成します。これは、独自の CA として機能し、追加の署名は必要ありません。この証明書は、生成時から 1 週間の期限が切れます。

1. **nodes** オプションを使用すると、キーは暗号化せずに保存されます。OpenSSL により、証明書の作成に必要な値の入力が求められます。

```
# openssl req -x509 -nodes -days 7 -newkey rsa:1024 -keyout mykey.pem -out mycert.pem
```

2. **subj** オプションを使用すると、値を指定したり、対話式プロンプトを回避したりできます。以下に例を示します。

```
# openssl req -x509 -nodes -days 7 -subj '/C=US/ST=NC/L=Raleigh/CN=www.redhat.com' -
newkey rsa:1024 -keyout mykey.pem -out mycert.pem
```

3. **new** および **key** オプションは、新規キーを生成する代わりに、既存の鍵を使用して証明書を生成します。

```
# openssl req -x509 -nodes -days 7 -new -key mykey.pem -out mycert.pem
```

証明書署名要求の作成

証明書を生成し、認証局(CA)で署名できるようにするには、CSR（証明書署名要求）を生成する必要があります。

```
# openssl req -new -key mykey.pem -out myreq.pem
```

証明書署名要求は、署名のために認証認証局に送信でき、有効な署名済み証明書が返されます。CSR を送信し、署名済み証明書を受信する手順は、使用する特定の認証局によって異なります。

独自の認証局の作成

独自の認証局を作成し、これを使用して証明書要求に署名することができます。認証局がシステムで信頼できる認証局として追加されると、認証局が署名した証明書がそのシステムで有効になります。このオプションは、証明書の数が一時的に必要になる場合に便利です。

1. の説明に従って、CA 用の自己署名証明書を作成し 手順B.2「自己署名証明書の作成」 ます。
2. OpenSSL では、CA が証明書に署名できるように以下のファイルを設定する必要があります。デフォルト設定を使用して、新しい OpenSSL インストールのある Red Hat Enterprise Linux システムで、以下のファイルを設定します。
   a. CA 証明書ファイルのパスをとして設定し **/etc/pki/CA/cacert.pem** ます。
   b. CA 秘密鍵ファイルのパスをとして設定し **/etc/pki/CA/private/cakey.pem** ます。
   c. で長さなしのインデックスファイルを作成し **/etc/pki/CA/index.txt** ます。
   d. で、初期シリアル番号（01 など）を含むファイルを作成し **/etc/pki/CA/serial** ます。
   e. RHEL 5 で以下の手順を実行する必要があります。
      i. 新しい証明書を保存するディレクトリーを作成します **/etc/pki/CA/newcerts**。
      ii. 証明書ディレクトリーに移動します **cd /etc/pki/tls/certs**。
3. 以下のコマンドは、CA を使用して CSR に署名します。

```
# openssl ca -notext -out mynewcert.pem -infiles myreq.pem
```

証明書のインストール

1. OpenSSL が証明書を認識するには、**certs** ディレクトリーにハッシュベースのシンボリックリンクを生成する必要があります。**/etc/pki/tls** は、Red Hat Enterprise Linux のバージョンの OpenSSL にある **certs** ディレクトリーの親です。**version** コマンドを使用して、親ディレクトリーを確認します。

```
# openssl version -d
OPENSSLDIR: "/etc/pki/tls"
```

2. 以下のコマンドを使用して、証明書に必要なシンボリックリンクを作成します。

```
# ln -s certfile `openssl x509 -noout -hash -in certfile`.0
```

複数の証明書が同じハッシュ値を持つことができます。この場合は、リンク名の接尾辞をより高い番号に変更します。例：

```
# ln -s certfile `openssl x509 -noout -hash -in certfile`.4
```

証明書の値の確認

証明書の内容は、以下のコマンドでプレーンテキストで確認できます。

```
# openssl x509 -text -in mycert.pem
```

NSS から PEM 形式への証明書のエクスポート

NSS 証明書データベースに保存されている証明書は、複数の方法でエクスポートして PEM 形式に変換できます。

- このコマンドにより、NSS データベースから指定されたニックネームの証明書をエクスポートします。

  ```
  # certutil -d . -L -n "Some Cert" -a > somecert.pem
  ```

- これらのコマンドは一緒に使用して、NSS データベースから証明書および秘密鍵をエクスポートし、それを PEM 形式に変換できます。クライアント証明書、CA の証明書、および秘密鍵を含むファイルを生成します。

  ```
  # pk12util -d . -n "Some Cert" -o somecert.pk12
  # openssl pkcs12 -in somecert.pk12 -out tmckay.pem
  ```

  PEM 出力ファイルの内容を制限するオプションについては、**openssl pkcs12** コマンドのドキュメントを参照してください。

バグを報告します。

付録C 改訂履歴

改訂 3.2.0-7 GA 後の更新。	June 2017	Susan イタリア
改訂 3.2.0-6 GA 後の更新。	Fri Oct 16 2015	Conscot ムムー
改訂 3.2.0-5 MRG-M 3.2 GA	Thu Oct 8 2015	Conscot ムムー
改訂 3.2.0-3 MRG-M 3.2 GA 用準備	Tue Sep 29 2015	Conscot ムムー
改訂 3.2.0-1 MRG-M 3.2 GA 用準備	Tue Jul 14 2015	Jared イタリア
改訂 3.1.0-5 MRG-M 3.1 GA 用準備	Wed Apr 01 2015	Jared イタリア
改訂 3.0.0-1 MRG-M 3.0 GA 用準備	Tue Sep 23 2014	Jared イタリア