Red Hat Enterprise MRG 3

メッセージングプログラミングリファレンス

Red Hat Enterprise Messaging でプログラミングガイド

Last Updated: 2020-09-01

Red Hat Enterprise MRG 3 メッセージングプログラミングリファレンス

Red Hat Enterprise Messaging でプログラミングガイド

Red Hat Customer Content Services

法律上の通知

概要

本ガイドは、Red Hat Enterprise Messaging Server を使用するアプリケーションを作成する開発者向けの情報を提供します。

目次

第1章 はじめに

1.1. RED HAT ENTERPRISE MRG MESSAGING

Red Hat Enterprise Messaging は、高度にスケーラブルな AMQP メッセージングブローカーで、Apache Qpid オープンソースプロジェクトをベースとしたクライアントライブラリーおよびツールのセットです。これは、Red Hat がエンタープライズユーザー向けに統合、テスト、およびサポートされます。

バグを報告します。

1.2. APACHE QPID

Apache Qpid は、Advanced Messaging Queue Protocol(AMQP)を実装するクロスプラットフォームのエンタープライズメッセージングシステムです。Apache Software Foundation オープンソースプロジェクトとして開発されています。

Apache Qpid を使用すると、AMQP モデルで直感的に使用しやすいメッセージング API をラップして、可能な限り複雑な処理を行えるようになりました（ただし、必要な場合は引き続き使用できます）。これにより、統合メッセージングを迅速かつ簡単に実行できるように、高性能でスケーラブルなアプリケーションを迅速に構築できます。

バグを報告します。

1.3. AMQP - ADVANCED MESSAGE QUEUING PROTOCOL

AMQP (Advanced Message Queuing Protocol)は、ワイヤプロトコルレベルで相互運用可能なメッセージング用のオープン標準です。AMQP を実装するメッセージブローカーは、アダプターまたはブリッジを必要とせずに相互に通信し、メッセージを交換できます。AMQP メッセージブローカーは、複数のプログラミング言語にファーストクラスのネイティブ言語バインディングを提供できるため、AMQP ベースのメッセージングは、Enterprise 全体でプラットフォーム間の互換性に適しています。

AMQP 標準は、ベンダーに依存しない OASIS 技術によって管理されます。

バグを報告します。

1.4. AMQP 0-10 と AMQP 1.0 の相違点

AMQP 1.0 は AMQP の最新規格です。AMQP 0-10 と AMQP 1.0 の最も重要な相違点は、この製品で使用される AMQP モデルのコンテキストを提供するために説明されています。

Broker Architecture

AMQP 0-10 は、オンラインプロトコルおよびブローカーアーキテクチャー（交換、バインディング、およびキュー）の仕様を提供します。一方、AMQP 1.0 はプロトコル仕様のみを提供し、ブローカーアーキテクチャーについては何も記述しません。AMQP 1.0 では、ブローカー、交換、またはバインディングは必要ありません。それらも除外しません。

「xchange」や「binding」の概念は 0-10 の概念です。

ブローカー管理

AMQP 0-10 は、ブローカーの管理に使用されるプロトコルコマンドを定義します。例として、「Queue Declare」、「Queue Delete」、「Queue Query」などがあります。AMQP 1.0 にはこのようなコマンドが含まれず、このような機能はより高いレイヤーで追加されることを前提としています。

MRG-M ブローカーにはレイヤー化された管理機能があります（Qpid Management Framework、QMF と呼ばれます）。

symmetry
AMQP 0-10 プロトコルは、各コネクションが「クライアント」エンドと「broker」エンドを持つように定義される非対称です。そのため、AMQP 0-10 はブローカー指向です。

AMQP 1.0 は対称であり、接続エンドポイントのロールにこのような制約を配置しません。1.0 はブローカーレスのポイントツーポイント通信を許可します。また、厳密な意味でブローカーではないサーバー/中間を作成することもできます。

バグを報告します。

1.5. MRG-M 3 での AMQP 1.0 サポート

1.5.1. C++ qpid::messaging API のサポート

MRG-M 3 では、**qpid::messaging** API で書かれた C++ および C# のアプリケーションが AMQP 1.0 について、特定のブローカーにその使用を結び付けないように、AMQP 1.0 と通信できます。

API 自体はシンプルです。アドレス構文、特に詳細なオプションには、マッピングの複雑性の多くが含まれます。

バグを報告します。

1.5.2. 応答先および一時的なキュー

API が 1.0 上で機能する仕組みに、マイナーな変更が 1 つあります。既存の 0-10 使用には影響しません。この変更には、リクエスト/応答パターンで応答を取得する場合など、一時キュー（またはトピック）の作成が必要になります。

0-10 を超えると、アドレスは '**#**' 文字で始まるノード名を、UUID を挿入して変換します。これは、名前がクライアントによって選択される 0-10 で適切に機能し、一意でなければなりません。名前のこの変換は、1 つのアドレス文字列からアドレスを作成する際に行われます（構成される部分ではなく）。その後、変更後の名前にアクセスできます **Address::getName()**。

ただし、1.0 より、このようなノードの名前はサーバーによって決定されます。この場合、アタッチに成功すると、割り当てられている名前をアプリケーションに戻す必要があります。新しいアクセプターを処理するには **Sender**、およびの両方に追加 **getAddress()** されてい **Receiver**ます。

0-10 の後方互換性を維持するには、Address コンストラクターは変換を **reply-to** 行いますが、1.0 に切り替えるアプリケーションでは、送信したリクエストメッセージで適切なアドレスを設定して正しいアドレスを取得してください。（この新しいアプローチは 0-10 と 1.0 の両方で機能します）。

バグを報告します。

1.5.3. 接続、セッション、およびリンク

使用されるプロトコルは、'**protocol**' 接続プロパティーからランタイム時に選択されます。認識できる値は '**amqp1.0**' および**amqp0-10**' です。AMQP 0-10 は依然としてデフォルトであり、1.0 のサポートは必要なモジュール（Apache Proton ライブラリー）がロードされている場合にのみ利用できます。

SASL ネゴシエーションは AMQP 1.0 で任意です。SASL レイヤーが必要ない場合は、に **sasl_mechanisms** connection オプションを設定でき **NONE**ます。

AMQP 1.0 は SSL 上で使用できますが、メッセージングクライアントは AMQP ネゴシエートされたセキュリティー層を使用しません。ピアは、使用されるポート（排他的または SSL ヘッダーを検出できる）で SSL を想定する必要があります。

トランザクションセッションはサポートされていません。

送信元または受信側の作成により、ピアへのリンクがアタッチされます。アタッチの詳細（ソースおよびターゲット）は、アドレス文字列で制御されます。

バグを報告します。

1.5.4. addresses

送信元またはレシーバーの作成時に指定されたアドレスで指定される名前は、ターゲットまたはソースのアドレスをそれぞれ設定するのに使用されます。

送信側にサブジェクトが指定されている場合は、明示的なサブジェクトなしで送信されるメッセージに使用されるデフォルトのサブジェクトになります。

レシーバーにサブジェクトが指定されている場合は、対象のメッセージセットのフィルターとして解釈されます。ワイルドカード（つまり ' '、または '**#**'）が含まれる場合 **legacy-amqp-topic-binding**、これは a として送信されなければ送信され **legacy-amqp-direct-binding**ます。

アドレスの名前が（または）'**#**' で始まる場合、動的フラグは対応するソースまたはターゲットに設定され、ノードのプロパティーに基づいて設定 **dynamic-node-properties** されます。動的フラグを設定すると、アドレスは指定しないでください。

バグを報告します。

1.5.5. オンデマンド作成レガシーアプリケーションの回避策

AMQP 1.0 では、クライアント指定の名前を持つオンデマンドのノードの作成は許可されません。ただし、MRG-M Qpid ブローカーには、0-10 でサポートされる動作と同様の**create**動作を可能にするエクステンション動作があります。つまり、存在しない場合は、クライアントによって指定された名前のノードが作成されます。これは、作成ポリシーに依存するアプリケーションの移行に役立ちます。ただし、これは標準以外の動作なので、新規アプリケーションはこれに依存しません。

アドレス指定されたノードが、名前として '**#**' を使用するか、または **create** ポリシーを介して作成される場合 - ノードのプロパティーがソースまたはターゲット **dynamic-node-properties** に送信されます。これらは、ノード内のネストされたマップで指定できます。また、ノードマップのすべての **durable** および **type** プロパティーが送信されます。また、ノードの 0-10 スタイルからの翻訳も **x-declare** あります。ノードに指定されたすべてのフィールドは、プロパティーに記載されているように含まれます。

バグを報告します。

1.5.6. リンクスコープの x-declare および x-subscribe

リンクスコープ **x-declare** および **x-subscribe** はサポートされていません。

代わりに、必要なサブスクリプションキュープロパティーと交換が必要なトピックノードを使用します。

バグを報告します。

1.5.7. ノードおよびリンクスコープの x-bindings

この **x-bindings** プロパティーは、ノードまたはリンクの AMQP 1.0 ではサポートされません。

リンクには、フィルターを使用します（これはレシーバーの作成時にのみ機能します）。

ノードの場合は、リンクにフィルターを使用します。ノードがキューである場合は、バインドする交換先に変更します。

バグを報告します。

1.5.8. ポリシーの削除

削除ポリシーは 1.0 上ではサポートされていません。削除ポリシーを使用する代わりに、作成時のノードのライフタイムポリシーを指定します。

バグを報告します。

1.5.9. ノードのライフタイムポリシー

1.0 仕様は **amqp:delete-on-close:list**、リンク確立に対応するように作成されたノードの以下のポリシーを定義します **amqp:delete-on-no-links:list amqp:delete-on-no-messages:list amqp:delete-on-no-links-or-messages:list**。

qpid::messaging API は **delete-on-close**、、、**delete-if-empty** またはのショートカット名を提供 **delete-if-unused**します **delete-if-unused-and-empty**。

ライフタイムポリシーは、ノードプロパティーで制御できます。例：

```
"my-queue;{create:always, node: {properties: {lifetime-policy: delete-if-empty}}}"
```

バグを報告します。

1.5.10. メッセージのタイムスタンプ

メッセージのタイムスタンプは AMQP 1.0 では利用できません。

バグを報告します。

1.5.11. AMQP メッセージプロパティーおよびヘッダーへのアクセス

1.0 メッセージ **properties** セクションの **message-id correlation-id user-id subject**、、、、**reply-to** および **content-type** フィールドはすべて、**Message** インスタンス上の同じ名前のアクセプターで設定または取得できます。これは、**header** セクションの **durable**、**priority** および **ttl** フィールドでも同様です。

AMQP 1.0 メッセージには **header** セクション内に **delivery-count** フィールドがあります。このフィールドには直接アクセプターがありません。ただし、値が 1 を超える場合、**Message::getRedelivered()** メソッドは true を返します。 の値で呼び出さ **Message::setRedelivered()** れた場合 **true**、配信数は 1 に設定され、それ以外の場合は 0 に設定されます。

受信した 1.0 メッセージの **application-properties** セクションは、**Message** クラスの **properties** マップから利用できます。**properties** マップは、メッセージを送信するときに **application-properties** セクションを設定するために使用されます。

Message クラスに直接アクセプターを持たない AMQP 1.0 メッセージ形式で定義された他のフィールドがあります。

キーの形式はです **x-amqp-*<field-name>***。使用中 **x-amqp-delivery-count** のキーは、**x-amqp-first-acquirer** および **header** セクション、および **x-amqp-to x-amqp-absolute-expiry-time x-amqp-creation-time x-amqp-group-id**、**x-amqp-qroup-sequence** および **properties** セクション **x-amqp-reply-to-group-id** のキーです。

delivery- および message- annotations セクションの他に、キーとキーを含むネストされたマップを介し **x-amqp-delivery-annotations** て利用でき **x-amqp-message-annotations** ます。

バグを報告します。

1.5.12. qpidd の AMQP サポート

1.0 サポートを有効にするには **qpidd**、**amqp** モジュールを読み込む必要があります。これにより、ブローカーは 0-10 1 と共に 1.0 プロトコルヘッダーを認識できます。

バグを報告します。

1.5.13. SASL(Simple Authentication and Security Layer)のサポート

デフォルトでは、ブローカーは 1.0 仕様で定義されている基礎となる SASL セキュリティーレイヤーを持つ接続を許可できます。ただし、認証が有効な場合は SASL セキュリティー層を使用する必要があります。

バグを報告します。

1.5.14. キューと交換

ブローカーは、両方の方向のキューまたは交換にリンクを割り当てることができます。ソースまたはターゲットのアドレスは、キューの名前または交換名と一致するかどうかを確認します。キュー と 同じ名前の交換がある場合は、キューが使用され、警告がログに記録されます。

ノードが交換の場合、ブローカーは一時的なリンクスコープのキューを作成し、交換にバインドします。このキューは送信リンクに使用されます。

ブローカーに接続された着信リンクおよび送信リンクは、以下の **qpid-config** ツールを使用して qpid 管理フレームワーク(QMF)で表示できます。

```
# qpid-config list incoming
```

または

```
# qpid-config list outgoing
```

動的フラグがソースまたはターゲットに設定されて **dynamic-node-properties** いると、は作成されたノードの特性を判断するために使用されます。プロパティーは QMF **create** メソッドのプロパティーと同じです。0-10 で定義されたオプション **durable auto-delete alternate-exchange**、**exchange-type** およびなどの qpidd 固有のオプションです **qpid.max-count**。

AMQP 1.0 **supported-dist-modes** プロパティーは、キューまたは交換が必要かどうかを判断します（**create** メソッドは '**type**' プロパティーを使用します）。を**move**指定すると、キューが作成されます。'**copy**' が指定されている場合、交換が作成されます。このプロパティーが設定されていない場合、キューが想定されます。

バグを報告します。

1.5.15. フィルター

発信リンクには、ソースにフィルターセットがある場合があります。ブローカーによってサポートされるフィルターは以下のとおりです。

- **legacy-amqp-direct-binding**
- **legacy-amqp-topic-binding**
- **legacy-amqp-headers-binding**
- **selector-filter**
- **xquery-filter**

フィルターは、アドレスに指定された **filter** プロパティーの **link** プロパティーで指定できます。この **filter** プロパティーの値はマップの一覧である必要があります。各マップは、名前、記述子（数値またはシンボリックとして指定できます）のキーと値のペアでフィルターを指定し、値を指定する必要があります。例：

```
my-xml-exchange; {link:{filter:{value:"declare variable $colour external;
colour='red'",name:x,descriptor:"apache.org:xquery-filter:string"}}}
```

表1.1 ノードタイプによるフィルターのサポート

	direct	トピック	ジャンクアウト	headers	xml	Queue
legacy-amqp-direct-binding	○	○	いいえ	いいえ	○	○
legacy-amqp-topic-binding	いいえ	○	いいえ	いいえ	いいえ	○
legacy-amqp-headers-binding	いいえ	いいえ	いいえ	○	いいえ	いいえ
xquery-filter	いいえ	いいえ	いいえ	いいえ	○	いいえ
selector-filter	○	○	○	○	○	○

バグを報告します。

1.5.16. AMQP 0-10 と AMQP 1.0 間のメッセージ変換

AMQP 0-10 で送信されたメッセージは AMQP 1.0 で送信するためにブローカーによって変換され、その逆も同様です。

1.0 のセクション **content-type** と 0-10 ヘッダー内の **properties** セクション間 **message-properties** で **message-id correlation-id userid**、、、および **content-encoding** マップします。ただし、0-10 は UUID で **message-id** なければなりません。1.0 メッセージを 0-10 に変換すると、このフィールドに有効な UUID が含まれていない場合は省略されます。

1.0 メッセージのヘッダーセクションの **priority** フィールドは 0-10 メッセージ **delivery-properties** の **priority** フィールドにマップします。1.0 メッセージの **durable** ヘッダーは 0-10 メッセージ **delivery-mode** において **delivery-properties** の値と同等で、**true** 以前の値は 2 の値で、後者のヘッダーと、**false** 前者は 1 と同等の値になります。

0-10 から 1.0 に変換する場合、はになり **reply-to routing-key**ます。交換が設定されていると、1.0 の **reply-to** アドレスは交換および任意のルーティングキーから構成されます（フォワードスラッシュで区切ります）。

型が指定されていない場合、クライアントは **reply-to** アドレスがキューであることを想定していることに注意してください。交換 **routing-key** 用の 0-10 が 1.0 に正しく変換されるようにするには **reply-to**、0-10 アドレスでノード種別（例：**amq.direct/rk; {node:{type:topic}}**'）を指定するか、アドレスインスタンスにタイプを設定します。

0-10 から 1.0 に変換する際、0-10 メッセージの空でない宛先がある場合は、1.0 メッセージ **properties** の **subject** フィールドが 0-10 メッセージの値に設定 **routing-key message-properties** されます。逆方向では、1.0 メッセージの **properties** セクションの **subject** フィールドが 0-10 メッセージ **routing-key** に設定 **message-properties** されます。255 文字で **routing-key** 切り捨てられることに注意してください。

0-10 メッセージの宛先は、1.0 に変換する際に **properties** セクションの '**to**' フィールドに入力されるために使用されますが、逆の変換は行われません（ブローカーから送信されたメッセージの宛先が 0-10 のサブスクリプションによって制御されるため）。

1.0 メッセージの **application-properties** セクションは 0-10 メッセージ **message-properties** の **application-headers** フィールドに変換され、その逆も同様です。

1.0 **reply-to** から 0-10 に変換する場合、アドレスにスラッシュが含まれている場合は、 *交換/ルーティングキー*の形式であることが仮定されます。スラッシュが含まれていない場合、これは単純なノード名になります。その名前が既存のキューと一致する場合、作成される 0-10 の交換は空に **reply-to** なり、ルーティングキーにキュー名が設定されます。名前が既存のキューに一致せず、名前が交換と一致する場合、交換に **reply-to** はノード名が設定され、ルーティングキーは空のままになります。ノードが既知のキューや交換を参照しない場合は、空に **reply-to** なります。

バグを報告します。

1.5.17. 機能

ブローカーはソースおよびターゲットの特定の機能を認識します。ブローカーのノードへのリンクをアタッチする場合、ターゲットまたはソースに対してそれぞれ機能を要求することができます。ブローカーが機能を認識し、問題のノードでそのケイパビリティーがサポートされる場合は、返信する attach 応答のケイパビリティーがエコーします。これにより、リンクを開始するピアが必要な性能を満たすかどうかを検証できます。

'**shared**' 機能は、交換からのサブスクリプションを複数のレシーバーで共有できるようにします。これを指定すると、作成されるサブスクリプションキューはリンクの名前になります（コンテナー ID は含まれません）。

ソースまたはターゲットが参照するキューまたは交換が永続化されている場合は、**durable**'' 機能が追加されます。ソースまたはターゲットがキューを参照すると、**queue**'' 機能が追加されます。ソースまたはターゲットが交換を参照すると、**topic**'' 機能が追加されます。ソースまたはターゲットがキューを参照する場合や、'**legacy-amqp-direct-binding**' が直接交換される場合は、追加されます。キューまたはトピックの交換を参照する場合は、'**legacy-amqp-topic-binding**' が追加されます。

'**create-on-demand**' 機能は、レガシーアプリケーションがメッセージングクライアントで '**create**' ポリシーを使用できるようにするエクステンションです。クライアントと名前付きノードが設定されていない場合、ノードは **dynamic-node-properties**、動的フラグが設定されているのと同じ方法で作成されます。

この拡張は、レガシーアプリケーションを移行するために提供されます。

バグを報告します。

1.5.18. 機能一致とアサート

assert オプションは、0-10 ベースのメカニズムと全く同じではありません。AMQP 1.0 より、クライアントは必要とする機能を設定し、ブローカーは提供可能な機能を設定し、**assert** オプションが有効になると、クライアントが要求するすべての機能がサポートされます。

クライアントが送信する機能は、ノードマップ内のネストされた一覧で制御できます。機能は単純な文字列（1.0 の *シンボル*）であり、名前と値のペアではありません。

ノードのプロパティーで設定 **durable** されている場合、'**durable**' のケイパビリティーが要求されます（揮発性メモリーが失われた場合、ノードはメッセージを失いません）。

が設定 **type** されている場合、これは要求される性能として渡されます。たとえば、'**queue**' は、ノードがキューのような特性に対応していることを意味します（コンシューマーが要求し、競合するコンシューマー間でメッセージを割り当てるまでメッセージを保存する）は、ノードが従来の pub-sub 特性に対応していることを**topic**意味します。

バグを報告します。

1.5.19. トピックを使用したサブスクリプションキューの設定

AMQP 1.0 でサブスクリプションキューを設定するには、タイプ 'topic' の新しいブローカーエンティティーが追加されました。

トピックは既存の交換を参照し、さらにそのトピックに割り当てられた受信側リンクのサブスクリプションキューを作成する際に使用するキューオプションを指定します。トピックは、異なる名前で存在できます。これは、各キューに異なるポリシーが適用される場所と同じ交換を指します。

トピックは、qpid-config ツールを使用して作成および削除できます（例：）。

```
# qpid-config add topic my-topic --argument exchange=amq.topic\
 --argument qpid.max_count=500 --argument qpid.policy_type=self-destruct
```

現在、アドレス '**my-topic/my-key**' を 1.0 経由で受信側が確立されると、サブスクリプションキューが 500 メッセージの制限で作成され、その制限を超えると（サブスクリプションが終了して）自体が削除され、キー ' ' で '**amq.topic**' にバインドされ**my-key**ます。

バグを報告します。

1.6. 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。

第2章 AMQP MODEL OVERVIEW

2.1. プロデューサー： コンシューマーモデル

AMQP Messaging は Producer: Consumer モデルを使用します。メッセージプロデューサーとメッセージコンシューマー間の通信は、交換とキューを提供するブローカーによって切り離されます。これにより、アプリケーションは異なる速度でデータを作成および使用できます。プロデューサーは、メッセージブローカーで交換するためにメッセージを送信します。コンシューマーは、対象のメッセージが含まれ、コンシューマーのメッセージをバッファーするサブスクリプションキューを作成することをサブスクライブしています。メッセージプロデューサーはサブスクリプションキューを作成し、消費するアプリケーション用に公開することもできます。

メッセージングブローカーは切り離されたレイヤーとして機能し、メッセージを分散する交換、コンシューマーおよびプロデューサーがパブリックキューおよびプライベートキューを作成し、それらを交換するためにサブスクライブし、プロデューサーアプリケーションが送信したメッセージをバッファーし、対象のコンシューマーがオンデマンドで配信します。

バグを報告します。

2.2. コンシューマー駆動型のメッセージング

AMQP は *コンシューマー駆動型* メッセージングを使用します。従来のポイントツーポイントメッセージングでは、メッセージプロデューサーがメッセージをキューにパブリッシュします。メッセージプロデューサーは、メッセージを受信するキューを把握します。このモデルのキューは、単一のコンシューマーのエンドポイントです。従来の publish-subscribe モデルでは、キューを複数のコンシューマーのエンドポイントとすることができます。キューには、キューに送信されたメッセージの個別コピーを受信するか、または独自のメッセージへのアクセスを共有してラウンドロビン方式で取得できます。
AMQP では、メッセージングのすべてのスタイルがサポートされます。単一のコンシューマーまたはコンシューマーに対して既知のキューに直接送信され、コンシューマーはキューでメッセージのコピーを閲覧したり、ラウンドロビン方式でメッセージ固有のインスタンスへのアクセスを共有するように強制したりできます。

AMQP は、メッセージを *交換* に送信する柔軟なアーキテクチャーを使用してこれらのパターンを実装します。交換によって、メッセージが交換にサブスクライブされているキューに分配されます。これにより、以前に説明したモデルをすべて許可し、メッセージ利用者が会話を駆動する機会も提供します。メッセージ生成アプリケーションは、オンラインになり、メッセージプロデューサーのメッセージに関心のある新規アプリケーションを認識する必要はありません。メッセージコンシューマーはキューを作成し、それらを交換にバインドできます。

AMQP には、さまざまな分散メカニズムをサポートする多くの交換タイプがあります。交換にサブスクライブすると、メッセージコンシューマーはキューをメッセージのフィルターとして機能するパラメーターにバインドできます。使用する交換タイプを選択し、バインディングキーを使用してその交換からメッセージをフィルターすることで、AMQP を使用して非常に柔軟で高速で拡張可能なメッセージングシステムを構築できます。

バグを報告します。

2.3. MESSAGE PRODUCER(SENDER)

メッセージブローカーで交換にメッセージを送信するアプリケーションは、メッセージブローカーに送信します。その後、交換によってメッセージを交換にサブスクライブされたキューに分配します。交換の種類やキューをサブスクライブするために使用されるパラメーターに応じて、メッセージはフィルタリングされるため、交換にサブスクライブしている各キューは、対象のメッセージのみを取得します。

メッセージプロデューサーは、コンシューマーに知識や関心のないメッセージを送信できます。メッセージの受信側から切り離されます。これにより、コンシューマーは受信するメッセージとそのメッセージを制御できます。プロデューサーは、キューを作成およびサブスクライブし、それらが送信したメッセージをそのキューにルーティングすることで、メッセージ消費方法を制御することもできます。これにより、幅広い設計が可能になります。

バグを報告します。

2.4. MESSAGE

アプリケーションは、他のアプリケーションに関連する情報を生成します。その情報を共有するには、情報をラップする移植可能なユニットを作成し、その情報を転送可能にする（メッセージ）。

メッセージは、*メッセージコンテンツ*（アプリケーションを受信するメッセージに関連する情報）、*メッセージヘッダー*、*ルーティングすべき場所*、 転送中にそれがどのように処理されるか、送信中に発生したメッセージ自体についての情報、メッセージ自体に関する情報で構成されます。

バグを報告します。

2.5. メッセージブローカー

メッセージは 2 つのアプリケーション間で直接送信できますが、作成するときに 2 つのアプリケーション間で相互に認識することが求められます。また、両方のアプリケーションが同時にオンラインになり、通信に同時にデータを生成する必要があることを意味します。アプリケーション間の通信が困難になるほど、アプリケーション間での通信は、共有される情報に関心を持ち始めるほどスケーリングされません。

*メッセージブローカーは切り離*されたレイヤーを提供します。サードパーティーにメッセージを送信することにより、メッセージブローカー（メッセージプロデューサーアプリケーション）は、その情報に関心のあるすべてのアプリケーションを認識する必要がなくなりました。メッセージブローカーは、メッセージを消費するアプリケーションのためにメッセージが含まれるキューを提供できます。メッセージブローカーは、異なるレートでデータの生成および消費に関連するアプリケーションが許可するバッファーも提供します。

Red Hat Enterprise Messaging は、Apache Qpid プロジェクトをベースとしたメッセージングブローカーを提供します。AMQP(Advanced Messaging Queue Protocol)メッセージングを実装します。

バグを報告します。

2.6. ルーティングキー

*ルーティングキー*はメッセージのブローカーによって配信用にメッセージをルーティングするために使用されるメッセージの文字列です。Red Hat Enterprise Messaging では、*メッセージサブジェクト*がルーティングに使用されます。

メッセージには内部 **x-ampq-0.10-routing-key** プロパティーがあります。ただし、これは Qpid Messaging API によって管理されるため、このプロパティーを手動でアクセスまたは設定する必要はありません。別の AMQP システムとメッセージを交換する場合に例外があります。この場合、Qpid Messaging API がメッセージおよび送信元のサブジェクトに基づいてこのプロパティーを管理する方法を理解する必要があります。

関連項目

- 「AMQP ルーティングキーおよびメッセージサブジェクト」

バグを報告します。

2.7. メッセージサブジェクト

メッセージにはサブジェクトプロパティーがあります。このサブジェクトはメッセージのルーティングに使用され、*ルーティングキー*と似ています。

メッセージサブジェクトはルーティングに使用されるため、メールサブジェクトには類似していません。電子メールメッセージでは、メールアドレスを使用してメールメッセージを受信側へルーティングするために使用されます。メールサブジェクトは、メッセージの内容を記述できます。Qpid メッセージのメッセージサブジェクトはメッセージのルーティングに使用されるため、1 つのアドレスを持つ 1 つ以上の受信者に電子メールを送信する機能など、メールアドレスのようになります。

メッセージのサブジェクトは空白にしたり、手動で設定したりすることもできます。または、メッセージがルーティングされる場所に基づいてメッセージが送信されるときに自動的に設定できます。

メッセージサブジェクトは送信時に自動的に設定されるため、メッセージサブジェクトとは処理されないアプリケーションを開発できるため、送信者によって設定できるようになります。また、より一般的な送信者を使用して、メッセージの件名を設定してルーティングに影響を与えることもできます。オプションは複数あります。

メッセージのサブジェクトが、送信側のオブジェクトで手動で設定されているか、または自動的に設定されたかに関係なく、メッセージの送信先を規定することを示します。

バグを報告します。

2.8. メッセージプロパティー

メッセージプロパティーは、*メッセージ*に設定できる **key:value** ペアの一覧です。一部の事前定義されたプロパティーは、メッセージブローカーによって、転送中のメッセージの処理方法を決定するために使用されます。これらのメッセージプロパティーを設定すると、サービスの品質と配信が保証されます。アプリケーション固有の機能には、他のユーザー定義メッセージプロパティーを設定できます。

バグを報告します。

2.9. CONNECTION

AMQP のコネクションは、メッセージブローカーとメッセージプロデューサーまたはメッセージコンシューマー間のネットワーク接続です。

バグを報告します。

2.10. SESSION

*セッション*は、クライアントアプリケーションとメッセージングブローカー間のスコープ付けの会話です。セッションは通信のために接続を使用し、リソースへの排他的アクセスや、セッションへのスコープ指定リソースの有効期間中の範囲を提供します。

複数の異なるセッションが同じ接続を使用できます。

バグを報告します。

2.11. 交換

AMQP では、*交換* は送信者からメッセージを受信するメッセージングブローカーの宛先です。メッセージの受信後、交換はメッセージのコピーを交換にバインドされたキューに分散します。消費アプリケーションは、これらのキューからメッセージを取得します。キューは、コンシューマーへの交換からのメッセージを指定するバインディングキーを使用して交換するバインドされます。キューバッファーメッセージ。これにより、多くのアプリケーションが単一の送信者から異なる速度でメッセージを受信できるようになります。

異なるディストリビューションアルゴリズムを提供するさまざまな種類の交換があります。キューを交換にバインドするために使用されるパラメーターは、交換の分散アルゴリズムと対話し、高性能な高度なルーティングスキーマを可能にします。

バグを報告します。

2.12. バインディング

メッセージキューは *バインディング* を使用して交換にバインドされます。バインディングは、交換からのどのメッセージがこのキューに関心があるかの説明です。交換タイプによって分散アルゴリズムが異なるため、交換にキューをサブスクライブするために使用するバインディングの内容は交換の種類とサブスクライバーの関心によって異なります。

バグを報告します。

2.13. トピック

AMQP 1.0 でサブスクリプションキューを設定するには、タイプ 'topic' の新しいブローカーエンティティーが追加されました。

トピックは既存の交換を参照し、さらにそのトピックに割り当てられた受信側リンクのサブスクリプションキューを作成する際に使用するキューオプションを指定します。異なるポリシーをキューに適用するのと同じ交換を参照する名前がそれぞれ異なるトピックが存在する可能性があります。

トピックは、qpid-config ツールを使用して作成および削除できます（例：）。

```
qpid-config add topic my-topic --argument exchange=amq.topic\
--argument qpid.max_count=500 --argument qpid.policy_type=self-destruct
```

現在、アドレス '**my-topic/my-key**' を 1.0 経由で受信側が確立されると、サブスクリプションキューが 500 メッセージの制限で作成され、その制限を超えると（サブスクリプションが終了して）自体が削除され、キー ' ' で '**amq.topic**' にバインドされ**my-key**ます。

バグを報告します。

2.14. DOMAIN

'domain' は別の AMQP 1.0 と互換性のあるプロセスを識別し、qpidd に接続するための十分な情報を提供します。ドメインには名前と URL があり、**sasl_mechanisms username**, も指定でき **password**ます。

ドメインは、のようにを使用して追加、削除 **qpid-config**、一覧表示できます。

```
qpid-config add domain my-domain --argument url=some.hostname.com:5672
```

ドメインが作成されたら、リンクオブジェクト「incoming」または「outgoing」リンクオブジェクトを作成して、その他のプロセス内のノードと qpidd 内のノード間のリンクを「incoming」または「outgoing」リンクのいずれかで確立できます。例：

```
qpid-config add incoming incoming-name --argument domain=my-domain --argument source=queue1
--argument target=queue2
```

このコマンドにより、コマンドが実行される qpidd インスタンス **queue1** で特定さ **my-domain** れたプロセス **queue2** でメッセージをプルし、そのメッセージをプルします。

送受信リンクは、何らかの理由で接続が失われた場合に自動的に再度確立されないことに注意してください。

バグを報告します。

2.15. 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に関する詳細は、『メッセージングインストール 『および設定ガイド』の「『キューのオプション」』 セクション』 を参照してください。

バグを報告します。

2.16. TRANSACTION

エディターが空のトピックコンテンツを初期化

- テキストの一部。

バグを報告します。

2.17. メッセージコンシューマー(RECEIVER)

メッセージ消費アプリケーションは、メッセージングブローカーからメッセージを受信します。これは、キューを作成し、バインディングキーを使用してメッセージングブローカーの交換にバインドすることで実行されます。

バグを報告します。

第3章 使ってみる

3.1. PYTHON を使い始める

3.1.1. Python メッセージングの開発

Python は、プロトタイピングに非常に簡単に使用できる、クロスプラットフォームの動的インタープリター言語です。解釈され、コンパイルされないため、コーディングからテストまでの時間が早くなります。これにより、テストおよび実験に非常に適しています。これはスクリプト言語のように使用でき、非常に大きなアプリケーションの開発にも使用できます。

本ガイドの多くの例は、Python code を使用して、Red Hat Enterprise Messaging を使用したプログラミングメッセージングアプリケーションの原則を説明します。これらのサンプルプログラムを実行するのは簡単な方法で実行し、コードをファイルに貼り付けてから、**python** インタープリターを呼び出してファイルを実行します。

軽量のプロトタイプ機能のほかに、メッセージング開発に最も有用な Python の機能が、Python インタープリターを対話的に実行できる機能です。オブジェクトの効果と状態をリアルタイムで試して検証することができます。

Apache Qpid の Python API は、Red Hat Enterprise Messaging でファーストクラスでサポートされる API です。

バグを報告します。

3.1.2. Python クライアントライブラリー

Python クライアント開発には、以下の 3 つのライブラリーがあります。

python-qpid

Apache Qpid Python クライアントライブラリー。

python-qpid-qmf

QMF(Queue Management Framework)Python クライアントライブラリー。

python-saslwrapper

saslwrapper ライブラリーの Python バインディング。

バグを報告します。

3.1.3. Python クライアントライブラリーのインストール(Red Hat Enterprise Linux 6)

Red Hat Enterprise Linux 6 用の Python クライアントライブラリーは、Red Hat カスタマーポータル から入手できます。

マシンが *Red Hat Network Classic* 管理を使用する場合には、**yum** コマンドで Python クライアントライブラリーをインストールできます。

Python クライアントライブラリーは、3 つのベースチャネルにあります。

- Red Hat Enterprise Linux Server 6

- Red Hat Enterprise Linux Workstation 6
- Red Hat Enterprise Linux Client 6

ベースチャネルのいずれかにシステムをサブスクライブします。

システムがベースチャネルにサブスクライブされると、root 権限で以下のコマンドを実行します。

```
yum install python-qpid python-qpid-qmf python-saslwrapper
```

バグを報告します。

3.2. .NET を使い始める

3.2.1. .NET メッセージング開発

すべての .NET 言語は、C++ Messaging API を使用してサポートされます。.NET 開発と他の言語の最も大きな相違点は、.NET 環境でブローカーは常にリモートサーバーで実行していることです。Python、C++、および Java の開発では、開発中にブローカーとクライアントを同じマシン上で実行でき、サンプルコードはこれを想定します。ただし、.NET クライアントとの接続はすべて、リモートで実行しているブローカーに接続します。

リモートサーバーに対して開発およびテストを行う間は、ファイアウォールを正しく設定することが重要です。このステップは、ブローカーがローカルで実行している場合にスキップできますが、ブローカーがリモートサーバーで稼働しているときに重要になります。

バグを報告します。

3.2.2. Windows SDK

MRG Messaging Windows SDK は、Windows 向けのネイティブ C++（非管理）および .NET（マネージド）クライアントを開発するために必要なファイルを含むダウンロードです。

バグを報告します。

3.2.3. Windows SDK コンテンツ

選択したバージョンに関係なく、Windows SDK には以下の共通のディレクトリーおよびファイルが含まれます。

\bin

- コンパイルされたバイナリー（.dll および .exe）ファイルと、関連するデバッグプログラムデータベース(.pdb)ファイル。
- ライブラリーファイルを強化します。
- Microsoft Visual Studio ランタイムライブラリーファイル。

\docs

Apache Qpid C++ API リファレンス

\dotnet_examples

Visual Studio ソリューションファイルと関連するプロジェクトファイル。C# の WinSDK の使用を示します。

\examples

管理されていない C++ で WinSDK を使用するための Visual Studio ソリューションファイルと関連するプロジェクトファイル

\include

.h ファイルのディレクトリーツリー

\lib

/bin 内のファイルに対応するリンカー .lib ファイル

バグを報告します。

3.2.4. Windows SDK のダウンロードおよびインストール方法

3.2.4.1. Windows SDK の取得

手順3.1 環境の Windows SDK の取得方法

1. Red Hat カスタマーポータル にログインします。
2. **A-Z** タブをクリックして製品一覧をアルファベット順で並べ替え、選択 **Red Hat Enterprise MRG Messaging** してダウンロード画面を表示します。
3. **Version** メニューから必要な製品バージョンを選択します。
4. **Architecture** メニューから必要なアーキテクチャーを選択します。
5. 環境に適した Windows SDK バイナリーを見つけ、をクリックし、ダウンロード **Download Now** を開始します。

「Windows SDK のダウンロードおよびインストール方法」の次のステップ

- 「Windows SDK のインストール」

バグを報告します。

3.2.4.2. Windows SDK のインストール

「Windows SDK のダウンロードおよびインストール方法」の以前の手順

- 「Windows SDK の取得」

1. ダウンロードした Windows SDK をファイルシステムに展開します。
2. ディレクトリーからエンビロメントの **/bin/Release/** ディレクトリーに **qpid***、すべての **boost*** ファイルをコピーし **/bin/Release/** ます。

バグを報告します。

3.3. C++ を使い始める

3.3.1. C++ メッセージング開発

Red Hat Enterprise Messaging がベースとなっているオープンソースの Apache Qpid ブローカーは、Java として利用でき、C++ ブローカーとして利用できます。Red Hat Enterprise Messaging のビルドに使用する C++ ブローカーです。

Python と C++ API には若干の違いがあります。ブローカー自体は C++ で書かれているため、C++ API が Python API と異なるいくつかのエリアでは、C++ API がより完全機能であり、ユーザーによってより広範囲に渡るルールとなります。

バグを報告します。

3.3.2. Linux 上の C++

3.3.2.1. C++ クライアントライブラリー

C++ クライアント開発用のパッケージは 5 つあります。

qpid-cpp-client
: Apache Qpid C++ クライアントライブラリー。

qpid-cpp-client-ssl
: クライアントの SSL サポート。

qpid-cpp-client-rdma
: Qpid クライアントの RDMA プロトコルのサポート（Infiniband を含む）。

qpid-cpp-client-devel
: Qpid C++ クライアントを開発するためのヘッダーファイルおよびツール。

qpidd-cpp-client-devel-docs
: AMQP クライアント開発ドキュメント。

バグを報告します。

3.3.2.2. Install C++ Client Libraries(Red Hat Enterprise Linux 6)

Red Hat Enterprise Linux 6 の C++ クライアントライブラリーは、Red Hat カスタマーポータル から入手できます。

マシンが *Red Hat Network Classic* 管理を使用する場合には、**yum** コマンドで C++ クライアントライブラリーをインストールできます。

システムを **Red Hat MRG Messaging v.2 (for RHEL-6 Server)** チャネルにサブスクライブします。

システムがこのチャンネルにサブスクライブしたら、root 権限で以下のコマンドを実行します。

```
yum install qpid-cpp-client qpid-cpp-client-rdma qpid-cpp-client-ssl qpid-cpp-client-devel
```

バグを報告します。

3.3.2.3. MRG 3 の C++ クライアントライブラリーのインストール

Red Hat Enterprise Linux 6 の C++ クライアントライブラリーは、Red Hat カスタマーポータル から入手できます。

マシンが *Red Hat Network Classic* 管理を使用する場合には、**yum** コマンドで C++ クライアントライブラリーをインストールできます。

システムを **Red Hat MRG Messaging v.3 (for RHEL-6 Server)** チャネルにサブスクライブします。

システムがこのチャンネルにサブスクライブしたら、root 権限で以下のコマンドを実行します。

```
yum install qpid-cpp-client qpid-cpp-client-rdma qpid-cpp-client-ssl qpid-cpp-client-devel
```

バグを報告します。

3.3.3. Windows 上の C++

3.3.3.1. Windows SDK

MRG Messaging Windows SDK は、Windows 向けのネイティブ C++（非管理）および .NET（マネージド）クライアントを開発するために必要なファイルを含むダウンロードです。

バグを報告します。

3.3.3.2. Windows SDK コンテンツ

選択したバージョンに関係なく、Windows SDK には以下の共通のディレクトリーおよびファイルが含まれます。

\bin

- コンパイルされたバイナリー（.dll および .exe）ファイルと、関連するデバッグプログラムデータベース(.pdb)ファイル。
- ライブラリーファイルを強化します。
- Microsoft Visual Studio ランタイムライブラリーファイル。

\docs

Apache Qpid C++ API リファレンス

\dotnet_examples

Visual Studio ソリューションファイルと関連するプロジェクトファイル。C# の WinSDK の使用を示します。

\examples

管理されていない C++ で WinSDK を使用するための Visual Studio ソリューションファイルと関連するプロジェクトファイル

\include

.h ファイルのディレクトリーツリー

\lib

/bin 内のファイルに対応するリンカー .lib ファイル

バグを報告します。

3.3.3.3. Windows SDK のダウンロードおよびインストール方法

3.3.3.3.1. Windows SDK の取得

手順3.2 環境の Windows SDK の取得方法

1. Red Hat カスタマーポータル にログインします。
2. **A-Z** タブをクリックして製品一覧をアルファベット順で並べ替え、選択 **Red Hat Enterprise MRG Messaging** してダウンロード画面を表示します。
3. **Version** メニューから必要な製品バージョンを選択します。
4. **Architecture** メニューから必要なアーキテクチャーを選択します。
5. 環境に適した Windows SDK バイナリーを見つけ、をクリックし、ダウンロード **Download Now** を開始します。

「Windows SDK のダウンロードおよびインストール方法」の次のステップ

- 「Windows SDK のインストール」

バグを報告します。

3.3.3.3.2. Windows SDK のインストール

「Windows SDK のダウンロードおよびインストール方法」の以前の手順

- 「Windows SDK の取得」

1. ダウンロードした Windows SDK をファイルシステムに展開します。
2. ディレクトリーからエンビロメントの **/bin/Release/** ディレクトリーに **qpid***、すべての **boost*** ファイルをコピーし **/bin/Release/** ます。

バグを報告します。

3.4. JAVA を始める

3.4.1. Java クライアントライブラリー

Java クライアント開発には、以下の 3 つのライブラリーがあります。

qpid-java-client

Qpid クライアントの Java 実装

qpid-java-common
Qpid Java クライアントの共通ファイル

qpid-java-example
プログラミングの例

関連項目

- 「Java JMS "Hello World" Program Listing」

バグを報告します。

3.4.2. Java クライアントライブラリーのインストール(Red Hat Enterprise Linux 6)

Red Hat Enterprise Linux 6 用の Java クライアント開発ライブラリーは、Red Hat Network から入手できます。

Java 開発パッケージをインストールするには、以下を実行します。

1. システムを **Additional Services Channels for Red Hat Enterprise Linux 6 / MRG Messaging v.2 (for RHEL-6 Server)** チャネルにサブスクライブします。
2. root 権限で以下の yum コマンドを実行します。

```
yum install qpid-java-client qpid-java-common qpid-java-example
```

バグを報告します。

3.5. RUBY を使い始める

3.5.1. Ruby メッセージングの開発

Ruby プログラミング言語には、他の言語と同じレベルのサポートはありません。Qpid Management Framework(QMF)にアクセスできるライブラリーがありますが、標準のメッセージング API でサポートされるクライアントライブラリーはありません。

バグを報告します。

3.5.2. Ruby クライアントライブラリー

Ruby クライアント開発には、2 つのライブラリーがあります。

ruby-qpid-qmf
Ruby QMF バインディング

ruby-saslwrapper
saslwrapper ライブラリーの Ruby バインディング

バグを報告します。

3.5.3. Ruby クライアントライブラリーのインストール(Red Hat Enterprise Linux 6)

Ruby クライアント開発ライブラリーは、Red Hat カスタマーポータル から入手できます。

ruby-qpid-qmf パッケージはメインチャンネルにあります。この **ruby-saslwrapper** パッケージは Optional 子チャンネルにあります。

Ruby クライアント開発ライブラリーをインストールするには、以下を実行します。

1. 以下のチャンネルのいずれかにシステムをサブスクライブします。
   - **Red Hat Enterprise Linux Server 6**
   - **Red Hat Enterprise Linux Client 6**
   - **Red Hat Enterprise Linux Workstation 6**
2. root 権限で、以下のコマンドを実行します。

```
yum install ruby-qpid-qmf
```

3. 以下のチャンネルのいずれかをサブスクライブします。
   - **Red Hat Enterprise Linux Optional Server v 6**
   - **Red Hat Enterprise Linux Optional Client 6**
   - **Red Hat Enterprise Linux Optional Workstation 6**
4. root 権限で、以下のコマンドを実行します。

```
yum install ruby-saslwrapper
```

バグを報告します。

3.6. HELLO WORLD

3.6.1. Red Hat Enterprise Messaging "Hello World"

Qpid Messaging API を使用して Red Hat Enterprise Messaging でメッセージを送受信する方法は「Hello World」の例です。

python

```
import sys
from qpid.messaging import *

connection = Connection("localhost:5672")

try:
  connection.open()
  session = connection.session()

  sender = session.sender("amq.topic")
  receiver = session.receiver("amq.topic")
```

```
  message = Message("Hello World!")
  sender.send(message)

  fetchedmessage = receiver.fetch(timeout=1)
  print fetchedmessage.content
  session.acknowledge()

except MessagingError,m:
  print m

connection.close()
```

C#/.NET

```
using System;
using Org.Apache.Qpid.Messaging;

namespace Org.Apache.Qpid.Messaging {
  class Program {
    static void Main(string[] args) {
      String broker = args.Length > 0 ? args[0] : "localhost:5672";
      String address = args.Length > 1 ? args[1] : "amq.topic";

      Connection connection = null;
      try {
        connection = new Connection(broker);
        connection.Open();
        Session session = connection.CreateSession();

        Receiver receiver = session.CreateReceiver(address);
        Sender sender = session.CreateSender(address);

        sender.Send(new Message("Hello world!"));

        Message message = new Message();
        message = receiver.Fetch(DurationConstants.SECOND * 1);
        Console.WriteLine("{0}", message.GetContentObject());
        session.Acknowledge();

        connection.Close();
      } catch (Exception e) {
        Console.WriteLine("Exception {0}.", e);
        if (connection != null)
          connection.Close();
      }
    }
  }
}
```

C++

```
#include <qpid/messaging/Connection.h>
#include <qpid/messaging/Message.h>
#include <qpid/messaging/Receiver.h>
```

```
#include <qpid/messaging/Sender.h>
#include <qpid/messaging/Session.h>

#include <iostream>

using namespace qpid::messaging;

int main(int argc, char** argv) {
    std::string broker = argc > 1 ? argv[1] : "localhost:5672";
    std::string address = argc > 2 ? argv[2] : "amq.topic";
    Connection connection(broker);
    try {
        connection.open();
        Session session = connection.createSession();

        Receiver receiver = session.createReceiver(address);
        Sender sender = session.createSender(address);

        sender.send(Message("Hello world!"));

        Message message = receiver.fetch(Duration::SECOND * 1);
        std::cout << message.getContentObject() << std::endl;
        session.acknowledge();

        connection.close();
        return 0;
    } catch(const std::exception& error) {
        std::cerr << error.what() << std::endl;
        connection.close();
        return 1;
    }
}
```

バグを報告します。

3.6.2. Java JMS "Hello World" Program Listing

このプログラムは、**qpid-java-examples** パッケージ内の他の例とともに利用できます。

Java

```
package org.apache.qpid.example.jmsexample.hello;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.util.Properties;

public class Hello {

  public Hello() {
  }

  public static void main(String[] args) {
    Hello producer = new Hello();
```

```
    producer.runTest();
  }

  private void runTest() {
    try {
      Properties properties = new Properties();
      properties.load(this.getClass().getResourceAsStream("hello.properties"));
      Context context = new InitialContext(properties);

      ConnectionFactory connectionFactory
          = (ConnectionFactory) context.lookup("qpidConnectionfactory");
      Connection connection = connectionFactory.createConnection();
      connection.start();

      Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
      Destination destination = (Destination) context.lookup("topicExchange");

      MessageProducer messageProducer = session.createProducer(destination);
      MessageConsumer messageConsumer = session.createConsumer(destination);

      TextMessage message = session.createTextMessage("Hello world!");
      messageProducer.send(message);

      message = (TextMessage)messageConsumer.receive();
      System.out.println(message.getText());

      connection.close();
      context.close();
    }
    catch (Exception exp) {
      exp.printStackTrace();
    }
  }
}
```

Hello World サンプル JNDI プロパティーファイルの内容は次のとおりです **hello.properties**。

```
java.naming.factory.initial
  = org.apache.qpid.jndi.PropertiesFileInitialContextFactory

# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.qpidConnectionfactory
  = amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672'
# destination.[jndiname] = [address_string]
destination.topicExchange = amq.topic
```

バグを報告します。

3.6.3. "Hello World"——k-through

Qpid Messaging クライアント開発ライブラリーには、メッセージングブローカーと通信し、メッセージの作成および管理に必要な機能が含まれています。したがって、最初にメッセージをプログラムにインポートします。

python

```
from qpid.messaging import *
```

C++

```
#include <qpid/messaging/Connection.h>
#include <qpid/messaging/Message.h>
#include <qpid/messaging/Receiver.h>
#include <qpid/messaging/Sender.h>
#include <qpid/messaging/Session.h>

using namespace qpid::messaging;
```

C#/.NET

```
using Org.Apache.Qpid.Messaging;

namespace Org.Apache.Qpid.Messaging {
```

メッセージブローカーと通信するには、コネクションが必要です。**Connection** オブジェクトのインスタンスを作成して取得します。Connection オブジェクトコンストラクターは、ブローカーの URL をパラメーターとして取ります。

python

```
connection = Connection("localhost:5672")
```

C++

```
Connection connection(broker);
```

C#/.NET

```
Connection connection = null;
connection = new Connection(broker);
```

認証が必要なリモートサーバーに接続する場合は、フォームに接続 URL を指定でき**username/password@serverurl:port**ます。リモートサーバーでこれを試みる場合は、メッセージブローカーでファイアウォールを開いて、ブローカーポートの着信接続を許可するようにしてください。

AMQP 1.0 プロトコルを使用してコネクションを開くには、以下のように指定します。

C++

```
Connection connection(broker, "{protocol:amqp1.0}");
```

C#/.NET

```
connection = new Connection(broker, "{protocol:amqp1.0}");
```

これで、ブローカーに Connection インスタンスが設定され、次のステップでコネクションを開きます。Connection オブジェクトには、設定された接続を開く **open** メソッドがあります。

メッセージブローカーがオフラインの場合など、接続を開くと失敗する可能性があります。そのため、これをサポートする言語では try:except ブロックにラップし、例外をキャッチします。

Python はインデントを使用することに注意してください。したがって、間隔を十分に注意してください。

python

```
try:
  connection.open()
```

C++

```
try {
  connection.open();
```

C#/.NET

```
connection.Open();
```

サーバーへの接続が開かれているので、*セッション* を作成する必要があります。セッションは、アプリケーションとサーバー間のスコープ付けの会話です。サーバーはセッションの範囲を使用して、キューの排他的アクセスとセッションスコープのライフタイムを実施します。

Connection オブジェクトには、**Session** オブジェクトを返す **createSession** メソッド(**session** Python)があるため、以前に作成した接続からセッションを取得します。

python

```
session = connection.session()
```

C++

```
Session session = connection.createSession();
```

C#/.NET

```
Session session = connection.CreateSession();
```

Session オブジェクトには **sender**、**receiver** ターゲットアドレスまたはソースアドレスをパラメーターとして取り、それぞれと **Receiver** オブジェクト **Sender** を返すメソッドがあります。メッセージは送受信する必要があるオブジェクトであるため、セッションの各メソッドを呼び出して作成します。このデモに **amq.topic** 交換を使用します。これはブローカーで事前設定された交換であるため、作成は不要で、その存在に依存します。

python

```
sender = session.sender("amq.topic")
receiver = session.receiver("amq.topic")
```

C++

```
Receiver receiver = session.createReceiver(address);
Sender sender = session.createSender(address);
```

C#/.NET

```
Receiver receiver = session.CreateReceiver(address);
Sender sender = session.CreateSender(address);
```

送信者は **ルーター** として考えることができます。アプリケーションからブローカーにメッセージをルーティングします。送信者のコンストラクターに渡すパラメーターは、メッセージのルーティング先のブローカーの宛先です。この場合、送信者はブローカーの **amq.topic** 交換にメッセージをルーティングします。ルーティングターゲットは交換であるため、ブローカーによってさらにルーティングされます。

レシーバーは **サブスクライバー** として見なすことができます。レシーバーを作成すると、コンストラクターに渡すパラメーターはサーバーのオブジェクトに解決されます。オブジェクトがキューである場合、レシーバーはそのキューにサブスクライブされます。オブジェクトが交換であれば、この場合のようにキューがバックグラウンドに作成され、交換にサブスクライブされます。詳細は後で詳しく説明します。現時点では、送信者が **amq.topic** 交換にメッセージを送信し、受信側がキューで受信することを示すのに十分です。

送信元と受信側ができました。送信するメッセージを作成する時間です。**Message** オブジェクトは、**message.content**以下の文字列となる文字列のコンストラクターにパラメーターとして取ります。

python

```
message = Message("Hello World!")
```

コンストラクター **Message** を **message.content** 介して設定した **content-type** 場合、オブジェクトコンストラクターが正しい値を設定します。ただし、プロパティーに値を割り当てることで **Message** オブジェクトの作成後にこれを設定した場合は、**message.content** プロパティーを **message.content_type** 適切に設定する必要があります。

送信者の **send** メソッドを使用して、メッセージをブローカーに送信することができます。

python

```
sender.send(message)
```

C++

```
sender.send(Message("Hello world!"));
```

C#/.NET

```
sender.Send(new Message("Hello world!"));
```

メッセージブローカーの **amq.topic** 交換にメッセージが送信されます。

レシーバーを作成する際に、バックグラウンドでブローカーがプライベート一時キューを作成し、**amq.topic** 交換にサブスクライブしました。このメッセージは、このキューで待機します。

次の手順では、レシーバーの **fetch** 方法を使用して、ブローカーからメッセージを取得します。

python

```
fetchedmessage = receiver.fetch(timeout=1)
```

C++

```
Message message = receiver.fetch(Duration::SECOND * 1);
```

C#/.NET

```
Message message = new Message();
message = receiver.Fetch(DurationConstants.SECOND * 1);
```

timeout パラメーターは、メッセージを待つ **fetch** 時間を指定します。タイムアウトを設定しないと、受信側はキューにメッセージが表示されるまで無期限に待機します。タイムアウトを 0 に設定すると、受信側はキューを確認し、何もなければすぐに返します。メッセージをルーティングしてキューに表示するのに十分な時間を確保するために、これを 1 秒でタイムアウトに設定します。

これでメッセージが出力されるはずです。**Message** オブジェクトを **Fetch** 返すため、**content** プロパティーを出力します。

python

```
print fetchedmessage.content
```

C++

```
std::cout << message.getContent() << std::endl;
```

C#/.NET

```
Console.WriteLine("{0}", message.GetContent());
```

トランザクションを終了するには、確認応答します。これにより、メッセージブローカーはキューからクリアできます（メッセージのキューを解除します）。

python

```
session.acknowledge()
```

C++

```
session.acknowledge();
```

C#/.NET

```
session.Acknowledge();
```

最後に、例外処理をサポートする言語の例外をキャッチし、障害が発生した場合はコンソールに適したものを出力し、メッセージブローカーへの接続を閉じます。

python

```
except MessagingError,m:
    print m

connection.close()
```

C++

```
} catch(const std::exception& error) {
    std::cerr << error.what() << std::endl;
    connection.close();
    return 1;
}
```

C#/.NET

```
} catch (Exception e) {
    Console.WriteLine("Exception {0}.", e);
    if (connection != null)
      connection.Close();
}
```

プログラムを実行するには、としてファイルを保存し **helloworld.py**、コマンドを使用して実行し **python helloworld.py**ます。メッセージブローカーがローカルマシンで実行されている場合は、プログラムリストに「Hello World!」という単語が表示されるはずです。

バグを報告します。

第4章 "HELLO WORLD" を超える

4.1. サブスクリプション

"Hello World" の例で、*トピックの交換* にメッセージを送信しました。AMQP メッセージングは、交換を使用して、メッセージ送信者とメッセージプロデューサー間で柔軟な切り離されたルーティングを提供します。メッセージコンシューマーは、キューを作成して交換にバインドすることで、交換を *サブスクライブ*できます。交換とバインディングについては、独自のセクションでより深さで説明されています。ここでは、特にトピックの交換について簡単に連絡し、交換とキューの違いについて説明します。これは、メッセージコンシューマーがキューをバインドすることで交換に *サブスクライブ*する方法を知るためです。

交換は、さまざまな点でキューとは異なります。大きな違いの1つは、キューがメッセージをキューにキューに置くことで保存できる点です。交換ではそれらをキューに配布しますが、ローカルストレージはありません。メッセージコンシューマーはメッセージブローカーによってメッセージプロデューサーから切り離されます。キューは、2 つの間でメッセージをバッファーするメカニズムを提供し、異なる速度でデータを作成および消費できるようにします。メッセージコンシューマーは、メッセージを受信するためにメッセージをキューに公開する時点で接続する必要はありません。このメッセージは削除されるまでキューに残ります。

一方、交換は、異なるキューにメッセージをルーティングするメカニズムです。メッセージが交換に送信され、その交換にバインドされたキューがない場合、メッセージは失われます。これはいずれもリッスンしていないため、メッセージを保存する場所はありません。キューはサブスクリプションであり、プロデューサーが作成したキューに**「I（アプリケーション）が関心を持って**いること」、またはコンシューマーが作成したキューの場合、またはプロデューサー**が作成したキューの場合、「 I want these message to be here for interested applications are**being here」を指定します。対象のメッセージにサブスクライブするため、コンシューマーアプリケーションはキューを作成し、交換にバインドするか、既存のキュー（*サブスクライブ*）に接続します。アプリケーションに関連するメッセージを提供するために、アプリケーションはキューを作成し、交換(*publish*)にバインドします。その後、アプリケーションの使用により、そのキューを使用できます。

この「Hello World」のサンプルプログラムでは、**amq.topic** 交換をリッスンするレシーバーを作成しました。バックグラウンドでは、キューと **amq.topic** 交換を作成 **subscribes** します。Hello World プログラム送信者は **amq.topic** 交換に **パブリッシュ** します。**amq.topic** 交換はデモに使用するのが適切です。トピック交換では、交換に送信されるメッセージサブジェクトのフィルターとして機能する *バインディングキー* を使用して（交換と **バインド** ）キューをサブスクライブできます。バインディングキーがない交換にバインドするため、交換を介して送信されるすべてのメッセージに関心があることを通知します。

送信側がそのメッセージを **amq.topic** 交換すると、メッセージは受信側のサブスクリプションキューに送信されます。その後、受信側がサブスクリプションキューからメッセージ **fetch()** を取得するよう呼び出します。

これを実証するために、Hello World プログラムに 2 つの変更を加えます。

まず、メッセージを **amq.topic** 交換に送信します。送信 **後**、受信側を交換に登録します。

これらの操作の順序を変更する必要があります。

python

```
sender = session.sender("amq.topic")
receiver = session.receiver("amq.topic")
```

```
message = Message("Hello World!")
sender.send(message)
```

C++

```
Session session = connection.createSession();

Receiver receiver = session.createReceiver(address);
Sender sender = session.createSender(address);

sender.send(Message("Hello world!"));
```

C#/.NET

```
Session session = connection.CreateSession();

Receiver receiver = session.CreateReceiver(address);
Sender sender = session.CreateSender(address);

sender.Send(new Message("Hello world!"));
```

現在、メッセージを送信する **前に** 交換で受信側を登録します。代わりにメッセージを送信して、受信側を登録します。

python

```
sender = session.sender("amq.topic")

message = Message("Hello World!")
sender.send(message)

receiver = session.receiver("amq.topic")
```

C++

```
Session session = connection.createSession();

Sender sender = session.createSender(address);
sender.send(Message("Hello world!"));

Receiver receiver = session.createReceiver(address);
```

C#/.NET

```
Session session = connection.CreateSession();

Sender sender = session.CreateSender(address);
sender.Send(new Message("Hello world!"));

Receiver receiver = session.CreateReceiver(address);
```

変更した Hello World プログラムを実行すると、「Hello World!」メッセージが表示されることはありません。何が発生しましたか？送信側がメッセージを **amq.topic** 交換に公開しました。その後、交換によって、サブスクライブされたすべてのキューにメッセージが配信されましたが、これはありませんでした。レシーバーが交換にサブスクライブしている場合は、メッセージを受信するには時間がかかりません。元のバージョンのプログラムでは、受信側がメッセージを送信する前に交換にサブスクライブし、そのサブスクリプションキューでメッセージのコピーを受け取ります。

送信側および受信側の作成時に作成されたサブスクリプションキューを調べます。これは、**qpid-config** コマンドを使用して行います。ブローカーを再起動してすべてのキューをクリアします（ブローカーの再起動時に非**durable** キューはすべて破棄されます）。次に、以下のコマンドを実行します。

```
qpid-config queues
```

ブローカーのキューの一覧が表示されます。

次に、Hello World プログラムを元のフォームに戻し、メッセージを送信**する前にレシーバーが作成され（交換にサブスクライブ**）します。発生を確認するために、交換サブスクリプションの作成とメッセージを送信するまで、アプリケーションを一時停止します。Python でこの操作を行うには、ユーザーが Enter を押し、この **raw_input** 方法を使用してキーボード入力を取得します。

python

```
sender = session.sender("amq.topic")
receiver = session.receiver("amq.topic")

print "Press Enter to continue"
x= raw_input()

message = Message("Hello World!")
sender.send(message)
```

これでプログラムを実行し、一時停止されている間、ブローカーのキュー **qpid-config queues** を調べるのに使用します。

プログラムを実行し、一時停止している間に、以下のコマンドを実行します。

```
qpid-config queues
```

一意のランダム ID を持つ排他的キューが表示されます。これは作成されて **amq.topic** 交換にバインドされ、受信側が交換からメッセージを受信できるようにするキューです。また、その最後に同じ ID 番号を持つ他のキューも多数表示されます。これらは、**qpid-config** ユーティリティーがメッセージブローカーをクエリーし、コマンドを実行するキューの一覧を受信するために使用するキューです。このコマンドを再度実行すると、弊社のレシーバーキューは同じままで、他のキューには新しい ID が設定されています。**qpid-config** コマンドを実行するたびに、サーバーに応答を受け取るために独自のキューが作成されます。実行していない場合は、キューを表示するために実行する必要があるため **qpid-config**、これらのキューが存在していないと認識できませんが、この **qpid-config** キューの用語を取ることができます。

レシーバーのキューは、送信側がそのメッセージを交換に送信する際に交換（**サブスクライブ**）されるため、交換によって「Hello World!」メッセージがサブスクリプションキューに配信され、準備ができると受信側がフェッチできるようになります。

レシーバー用に作成されたキューは *排他的キュー* であるため、一度にアクセスできるのは 1 つのセッションのみです。

バージョン 2.2 以降

queue-exchange バインディングを表示するには、以下を実行します。

```
qpid-config queues -b
```

-b スイッチにはバインディングが表示されます。動的に作成された 2 つのキューが **amq.topic** 交換にバインドされていることを確認できます。

バージョン 2.3 以降

queue-exchange バインディングを表示するには、以下を実行します。

```
qpid-config queues -r
```

-r スイッチにはバインディングが表示されます。動的に作成された 2 つのキューが **amq.topic** 交換にバインドされていることを確認できます。

アプリケーションがウェイクして実行を完了すると、セッションを **connection.close()** 終了する呼び出しと、ブローカーの排他的キューが 2 つ削除されます。**qpid-config queues** 再度実行して確認します。

別の実験は、メッセージの送信前に受信側を作成し、メッセージが送信される後に別の受信側を作成することもできます。メッセージを受信するためにメッセージが送信される前に受信側が作成されていることと、メッセージが受信しないように送信された後に作成されることが予想されます。

この単純なアプリケーションは、動的に作成されたキューを使用して **amq.topic** 交換と対話します。このキューはプライベート（およびという名前 **exclusive**）であり、コンシューマーが切断されると削除されるため、パブリッシュには適していません。メッセージの送信時に交換に接続されている可能性のあるコンシューマーがメッセージを利用できるようにするには、メッセージ作成アプリケーションが公開されているキュー（*公開*）を作成する必要があります。消費アプリケーションは、この公開キューにサブスクライブし、切り離された方法でメッセージを受信できます。

当然ながら、メッセージを取り消すことは重要でない場合は、交換に単に公開する「Hello World」パターンを使用して、交換にサブスクライブして独自のキューを作成することができます。AMQP messaging は、メッセージングシステム設計に多くの柔軟性を提供します。

バグを報告します。

4.2. パブリッシュ

メッセージプロデューサーとして、AMQP メッセージングと使用できるパブリッシュストラテジーが多数あります。

メッセージを交換に公開でき、メッセージ消費アプリケーションは交換にサブスクライブし、独自のキューを作成できます。アプリケーションが生成する情報を配布する方法に応じて、使用できる交換タイプは複数あります。交換にパブリッシュする際の注意として、メッセージがウインストルに表示されないときにメッセージがリスニングされない場合は、メッセージ送信時にコンシューマーが交換にサブスクライブされていなければ、メッセージは ether に消えます。これは保存されません。コンシューマーがリッスンしているかどうかをメッセージを保存する必要がある場合は、キューに公開します。

キューを作成して交換にサブスクライブすることで、メッセージをキューに公開できます。その後、そのキューにルーティングされたメッセージを送信し、アプリケーションの使用により公開されたキューに接続し、メッセージを収集できます。このパブリッシュの方法は、ユーザーがリッスンしているかど

うかに関わらずメッセージをブローカーに保存する必要がある場合に使用します。パブリッシュのこの方法を使用すると、コンシューマーが交換に対して独自のサブスクリプションを作成したり、キューにのみ公開したりできます。

排他的キューにパブリッシュするには、直接交換にパブリッシュし、公開キューを排他バインディングキーによる交換にバインドします。つまり、メッセージをキューに直接ルーティングでき、no-one 以外のキューは、これらのメッセージを受信できる交換にバインドできます。

キューにパブリッシュし、コンシューマーがメッセージを受信する独自のキューを作成できるようにするには、ダウアウトまたは Topic の交換にパブリッシュし、適切なバインディングキーでキューを作成およびバインドしてメッセージを受信することができます。その後、コンシューマーはキューにサブスクライブし、独自のキューを作成し、交換にバインドすることもできます。

バグを報告します。

4.3. AMQP EXCHANGE タイプ

AMQP Exchange には 5 つのタイプがあります。異なる交換はメッセージのルーティング方法が異なるため、コンシューマーが対象となる特定の情報フローにサブスクライブできるようにします。

AMQP Exchange タイプは以下のとおりです。

direct

Direct Exchange を使用すると、コンシューマーはキューをキーにバインドできます。ダイレクトタイプの交換によってメッセージが受信された場合、メッセージはバインディングキーがメッセージの件名と一致するキューにルーティングされます。Direct Exchange は排他的バインディングもサポートし、キューが交換に送信されたメッセージを所有し、単純なダイレクトツーキューモデルを実装することができます。

トピック

Topic Exchange を使用すると、コンシューマーはワイルドカード一致を指定するキーを使用してキューをバインドできます。ワイルドカードは、交換に送信されたメッセージの件名と照合されます。これにより、トピックの交換と異なるバインディングキーを持つさまざまなキューを使用して、メッセージフィルタリングパターンを実装できます。

バグを報告します。

4.4. 事前設定された交換

初期状態の Red Hat Enterprise Messaging ブローカーには、メッセージングに使用できる事前設定された 5 つの交換があります。これらの交換はすべて永続的として設定されるため、ブローカーの起動時に常に利用可能になります。

デフォルトの交換

名前なしの直接交換。デフォルトではすべてのキューはこの交換にバインドされ、キュー名でアクセスできるようになります。

amq.direct

事前設定されたダイレクト交換。

amq.fanout

事前設定されたジャンクアウト交換。

amq.match

事前設定されたヘッダー交換。

amq.topic

事前設定されたトピック交換。

バグを報告します。

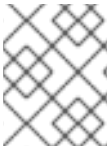
4.5. EXCHANGE サブスクリプションパターン

Exchange にサブスクライブする 3 つのパターンがあります。

1. メッセージのコピー
2. メッセージの移動
3. 排他的バインディング

メッセージのコピー

メッセージのコピーは、各コンシューマーがすべてのメッセージの独自のコピーを取得する場所です。



注記

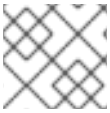
このアプローチは、パブリッシュ/サブスクライブパターン、一時 またはプライベートサブスクリプション としても知られて *います*。

この場合、アプリケーションが切断されると破棄される一時的なプライベートキューが作成され、バインドされます。このアプローチは、複数のコンシューマー間でメッセージの責任を共有する必要がない場合や、アプリケーションが稼働していない場合や切断された時に送信されるメッセージを重要としない場合に合理的です。

たとえば、サービスがメッセージに基づいてログのアクティビティーをログに記録する場合や、複数のコンシューマーにイベントの通知が必要な場合などに、この方法は理にかなっています。

メッセージの移動

メッセージの移行では、複数のコンシューマーが同じキューに接続し、ラウンドロビン方式でキューからメッセージを取得する場合です。



注記

このアプローチは *Shared Queue* としても知られています。

コンシューマー A とコンシューマー B が同じ共有キューにアクセスしている場合、コンシューマー A はコンシューマー B がキューから取得するメッセージを表示しません。たとえば、ワーカーノードがワークキューからジョブをディスパッチするケースでは、この方法が理にかなっています。1 つのノードでは、各メッセージのみを表示する必要があります。

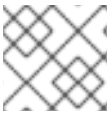
これにより、アプリケーションが切断されている場合はキューでメッセージをバッファーでき、キュー内のメッセージに対する責任を複数コンシューマーが共有できるようになります。

たとえば、ワーカーノードがワークキューからジョブをディスパッチするケースでは、この方法が理にかなっています。1 つのノードでは、各メッセージのみを表示する必要があります。

これらの 2 つのパターンは相互排他的ではありません。たとえば、3 つのワーカーノードはラウンドロビン方式でキューを共有でき、別のプロセスはキューに独自のメッセージのコピーを取得してアーカイブを作成できます。

排他的バインディング

3 つ目のパターンの exclusive バインディングは、コンシューマーのみがエンドポイントにルーティングされるメッセージにアクセスできることについて規定しています。



注記

排他的バインディングが AMQP 1.0 でサポートされない

バグを報告します。

4.6. デフォルトの交換

4.6.1. デフォルトの交換

Default Exchange は、事前設定された名前なしのダイレクト交換です。

デフォルトでは、すべてのキューは Default Exchange にバインドされます。つまり、交換の非修飾のキュー名が名前なしの交換に解決するため、キュー名をターゲットアドレスとして使用することでキューをターゲットにすることができます。

バグを報告します。

4.6.2. デフォルト交換を使用したキューへの公開

すべてのキューは、バインディングキーとしてキュー名を使用して、デフォルトの交換に自動的にバインドされます。そのため、デフォルトの交換にバインドされたキューに公開する必要があるのはキューを宣言することです。デフォルト交換へのバインディングは自動的に作成されます。Default Exchange は *直接交換* であるため、名前なしであるため、キューに到達するには、メッセージをキュー名に送信するだけで十分です。

以下を使用して、デフォルト交換にバインドされた「quick-publish」という名前のキューを作成するには、**qpid-config**以下を実行します。

```
qpid-config add queue quick-publish
```

アプリケーションでは、キューは送信側のオブジェクトを作成する副作用として作成できます。アドレスにパラメーターが含まれる **{create: always}** 場合、キューが存在しない場合は作成されます。さらに **always**、**create** コマンドは引数を取り、送信側がアドレスに接続 **sender** した場合に限りキューを作成するか **receiver**、受信側がそのアドレスに接続する場合にのみキューを作成するように指定することもできます。

以下は、「quick-publish」のサンプルキューの作成です。

python

```
sender = session.sender("quick-publish; {create: always}")
```

C++

```
Sender sender = session.createSender("quick-publish; {create: always}")
```

バグを報告します。

4.6.3. デフォルトの交換のサブスクライブ

デフォルト交換をサブスクライブするには、レシーバーを作成し、キューの名前をコンストラクターに渡します。たとえば、Python API を使用してキュー「**quick-publish**」にサブスクライブするには、以下を実行します。

C++

```
Receiver receiver = session.createReceiver('quick-publish');
```

python

```
receiver = session.receiver('quick-publish')
```

この受信側を使用して、**quick-publish** キューからメッセージを取得できるようになりました。

キューからメッセージを削除しない閲覧専用ビューを取得するには、以下を実行します。

C++

```
Receiver receiver = session.createReceiver('quick-publish; {mode: browse}');
```

python

```
receiver = session.receiver('quick-publish; {mode: browse}')
```

アプリケーションが独自のメッセージのコピーを要求する場合など、存在しないキューを作成してサブスクライブする場合は、**create** パラメーターを使用します。

C++

```
Receiver receiver = session.createReceiver("my-own-copies-please; {create: always, node: {type:
'queue'}}");
```

python

```
receiver = session.receiver("my-own-copies-please; {create: always, node: {type: 'queue'}}")
```

キュー**my-own-copies-please**"" がすでに存在する場合、受信側はそのキューに接続します。キューが存在しない場合は、作成されます（すべての例では、十分な権限が想定されます）。

「**my-own-copies-please**」という **交換** が存在する場合は、キューを作成する代わりに受信側がそのアドレスに警告して接続する点が 1 つあります。これは意図したものではなく、予測できない結果となります。これを回避するには、以下のように assert パラメーターを使用します。

C++

```
try {
  Receiver receiver = session.createReceiver("my-own-copies-please; {create: always, assert: always, node: {type: 'queue'}}");
} catch(const std::exception& error) {
  std::cerr << error.what() << std::endl;
}
```

python

```
try:
  receiver = session.receiver("my-own-copies-please; {create: always, assert: always, node: {type: 'queue'}}")
except MessagingError m:
  print m
```

「**my-own-copies-please**」がすでに存在していて交換されている場合は、レシーバーコンストラクターが例外「**expected queue, got topic**」を出力します。

直接交換のインスタンスですが、デフォルトの交換では同じキーを使用する複数のバインディングが許可されません。各キューは、Default Exchange に一意にバインドされます。つまり、他のダイレクト交換と同様に、キューに接続して送信されたメッセージを取得することしかできないため、別のキューを並行してメッセージのコピーを受信するために、別のキューをバインドすることはできません。

関連項目

- 「Exchange サブスクリプションパターン」

バグを報告します。

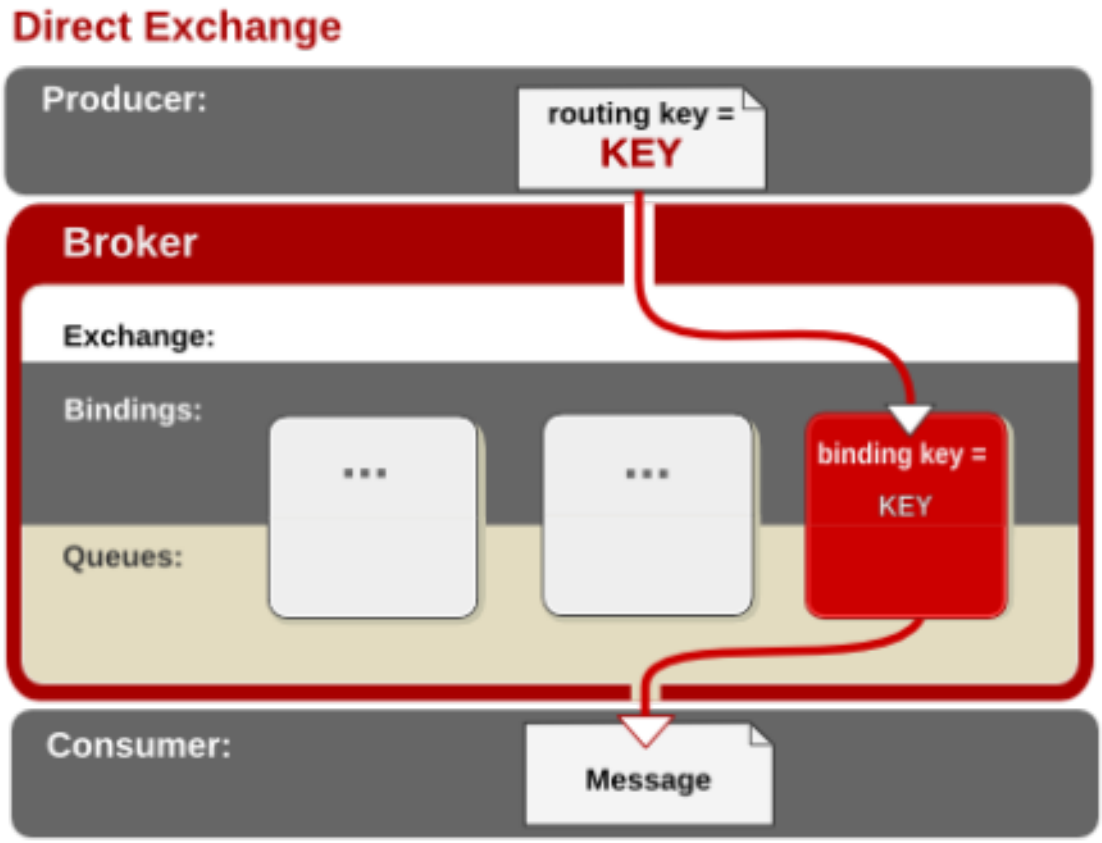
4.7. 直接交換

4.7.1. 直接交換

Direct Exchange はメッセージをルーティングし、キューのバインディングキーとメッセージの件名（*ルーティングキー*）と*完全に一致するキューにメッセージをルーティング*します。

この図では、**複数のキューが同じバインディングキーを使用して Direct Exchange にバインドできることに注意し** てください。図では、1 つのキューに 1 つのメッセージが表示されますが、その交換の他のキューに同じバインディングキーがある場合、それらのキューもメッセージを受け取ります。

図4.1 直接交換



直接交換は、トピック交換の特殊化です。実質的に、直接交換は、バインディングキーにワイルドカードが使用されていない Topic Exchange です。

Direct Exchange は *排他的バインディング*もサポートするため、キューはキューを交換専用にバインドするために使用されるルーティングキーを使用して、Direct Exchange に送信されたメッセージの受信側を保証できるようにします。

バグを報告します。

4.7.2. qpid-config を使用した直接交換の作成

このコマンドは、新しいダイレクト交換を **qpid-config add exchange direct *exchange name*** 作成します。

以下の **qpid-config** コマンド例は、という新しいダイレクト交換を作成し **engineering**ます。

```
qpid-config add exchange direct engineering
```

バグを報告します。

4.7.3. アプリケーションからの直接交換の作成

送信者または受信側を作成する副次的影響として、アプリケーションで直接交換を作成できます。たとえば、以下の例では、というダイレクト交換を作成し **engineering**ます。

python

```
sender = session.sender('engineering;{create: always, node:{type:topic, x-declare:{type:direct}}}')
```

engineering すでにという名前の交換が存在する場合は、送信者は新しいものを作成しようとはしませんが、既存のに接続します。ただし、名前のキューが **engineering** すでに存在する場合、送信者はその **キュー** にサイレントに接続するため、注意が必要です。

送信側が呼び出された新規交換または既存の交換に接続するようにするには **engineering**、以下の例の **assert**ように使用できます。

python

```
try:
  sender = session.sender('engineering;{create: always, node:{type:topic, x-declare:{type:direct}},
assert: always}')
except MessagingError, m:
  print m
```

使用すると、交換ではなくキューが **engineering** 存在しキューである **assert: always, node: {type: topic}**場合、送信元コンストラクターによって例外が「**expected topic, got queue**」という例外が発生します。

キューではなく交換であることを確認 **assert** するために使用できますが、その交換の ***タイプ*** を確認することはできません。

バグを報告します。

4.7.4. 直接交換への公開

直接交換にパブリッシュするには、2 つのオプションがあります。

特定のエンドポイントをターゲットとする送信者を作成します。

最初に、メッセージを公開先のエンドポイントに直接ルーティングする送信者を作成します。直接交換には完全一致が必要であるため、特定の送信先に送信する必要があります。同時に、複数のキューを交換にバインドして、同じ宛先にルーティングされたメッセージを受信できることに注意してください。そのため、複数のコンシューマーを持つ特定のエンドポイントになります。

まず、サーバーで以下のコマンドでエンドポイントを作成します。

```
qpid-config add exchange direct finance
```

または、以下のコードを使用します。

python

```
sender = session.sender('finance;{create:always, node: {type: topic, x-declare: {type: direct}}}')
```

この例では、**finance** 交換の **reports** エンドポイントにメッセージをルーティングする送信者を作成します。

python

```
sender = session.sender('finance/reports')
sender.send('Message to all consumers bound to finance with key reports')
```

送信したすべてのメッセージは、キーを使用して **finance** 直接交換にバインドされたキューに **sender** 移動 **reports**します。

この注意事項を説明するのに役立つので、直接交換に公開する 2 つ目のオプションを見てみましょう。

交換をターゲットとする送信者を作成します。

2 つ目のオプションは、メッセージを交換にルーティングする送信者を作成し、メッセージサブジェクトを使用して特定のエンドポイントへのルーティングを制御することです。これにより、たとえば、ランタイムで提供されるキーの名前（他のメッセージのボディーにある場合など）に基づいて、メッセージの移動先を動的に決定できます。

以下の例は、これを行う方法を示しています。

python

```
sender = session.sender('finance; {assert: always, node: {type: topic}}')
msg = Message('Message to all consumers bound to finance with key reports')
msg.subject = 'reports'
sender.send(msg)
```

交換をターゲットとする送信側で、メッセージを交換先に設定することで指定し **subject**ます。メッセージを送信する前に、サブジェクトを変更することで、その交換で異なるエンドポイントをターゲットにすることができます。たとえば、同じメッセージのコピーをおよびに送信するには、以下を **finance/reports finance/records**行います。

python

```
sender = session.sender('finance; {assert: always, node: {type: topic}}')
msg = Message('Message for reports and records')

msg.subject = 'reports'
sender.send(msg)

msg.subject = 'records'
sender.send(msg)
```

{assert: always, node: {type: topic}} デフォルトの交換に **finance** バインドされた名前のキューに誤って接続しないようにします。キューと交換には個別の namespace がありますが、デフォルトの交換は名前なしであることに注意してください。

注意事項

2 つ目のケースで確認できるように、メッセージのルーティング先に影響するサブジェクトを設定します。最初の方法（アドレス内のサブジェクトと送信者）を使用する場合は、メッセージサブジェクトを誤って設定しないように注意してください。メッセージサブジェクトが空白の場合に送信時に、送信元は正しいサブジェクトをメッセージに書き込みます。ただし、指定するメッセージサブジェクトは上書きされません。最初のメソッド - 送信側のアドレスで、メッセージのサブジェクトが設定されていないすべてのメッセージの「デフォルト宛先」を提供します。メッセージの送信前にサブジェクトを明示的に設定すると、交換上の他のエンドポイントをターゲットにすることができます。この場合、カスタムサブジェクトに基づいて追加のルーティングのために交換に送信されます。メッセージサブジェクトを設定するとルーティングが決定されることに注意してください。

バグを報告します。

4.7.5. 直接交換のサブスクライブ

メッセージのコピーを使用したデフォルト交換へのサブスクライブ

これは、実装する最も簡単な方法です。交換名とルーティングキーで構成されるアドレスを使用して受信側を作成します。たとえば、対象の**reports**キーを使用して、直接交換**finance**"" にレシーバーを作成します。

C++

```
Receiver receiver = session.createReceiver("finance/reports")
```

python

```
receiver = session.receiver('finance/reports')
```

共有キューを使用した直接交換のサブスクライブ

共有キューを使用したサブスクリプションは、サブスクリプションキューに命名して、特別でないものを定義することで作成できます。例：

C++

```
Receiver receiver = session.createReceiver("finance/quick-publish;{link:{name:my-subscription, x-declare:{exclusive:False}}}");
```

python

```
receiver = session.receiver('finance/quick-publish;{link:{name:my-subscription, x-declare:{exclusive:False}}}')
```

キューを作成し、ルーティングキーを使用してダイレクト交換にバインドできます。これには、を使用し **x-bindings**ます。例：

C++

```
Receiver receiver = session.createReceiver("my-subscription;{create: always, node:{x-bindings:[{exchange: 'finance', key: 'quick-publish'}]}}");
```

python

```
receiver = session.receiver('my-subscription;{create: always, node:{x-bindings: [{exchange: 'finance', key: 'quick-publish'}]}}')
```

「**my-subscription**」という名前の共有キューを作成し、キー "" で直接交換**finance**"" にバインドしました**quick-publish**。

AMQP 1.0

AMQP 1.0 では Link-scoped **x-declare** および Node-scoped **x-bindings** 句の両方に対応していないため、共有サブスクリプションのケイパビリティーをリクエストします。

C++

```
Receiver receiver = session.createReceiver("finance/quick-publish;{node: {capabilities:[shared]},
link: {name: 'my-subscription'}}");
```

関連項目

- 「Exchange サブスクリプションパターン」

バグを報告します。

4.7.6. 直接交換の排他的バインディング

直接交換で *排他的バインディング* を宣言すると、常に 1 つのコンシューマーが交換にバインドされるようになります。この鍵を使用して交換に新しいコンシューマーがサブスクライブされると、以前のコンシューマーのバインディングは同期的にドロップされます。これにより、同時バインド/バインド解除操作が行われる間にドロップメッセージや重複配信が発生する可能性なく、メッセージングルーティングが保証されたメッセージのアトミック性を持つコンシューマー間で切り換えることができます。

exchange-bind 引数 **qpid.exclusive-binding** は、排他的バインディングを宣言するために使用されます。

```
drain -f "amq.direct; {create:always, link: {name:one, x-bindings:[{key:unique, arguments:
{qpid.exclusive-binding:True}}]}}"
```

排他的バインディングは AMQP 1.0 では利用できません。

バグを報告します。

4.8. FANOUT EXCHANGE

4.8.1. 事前設定された交換

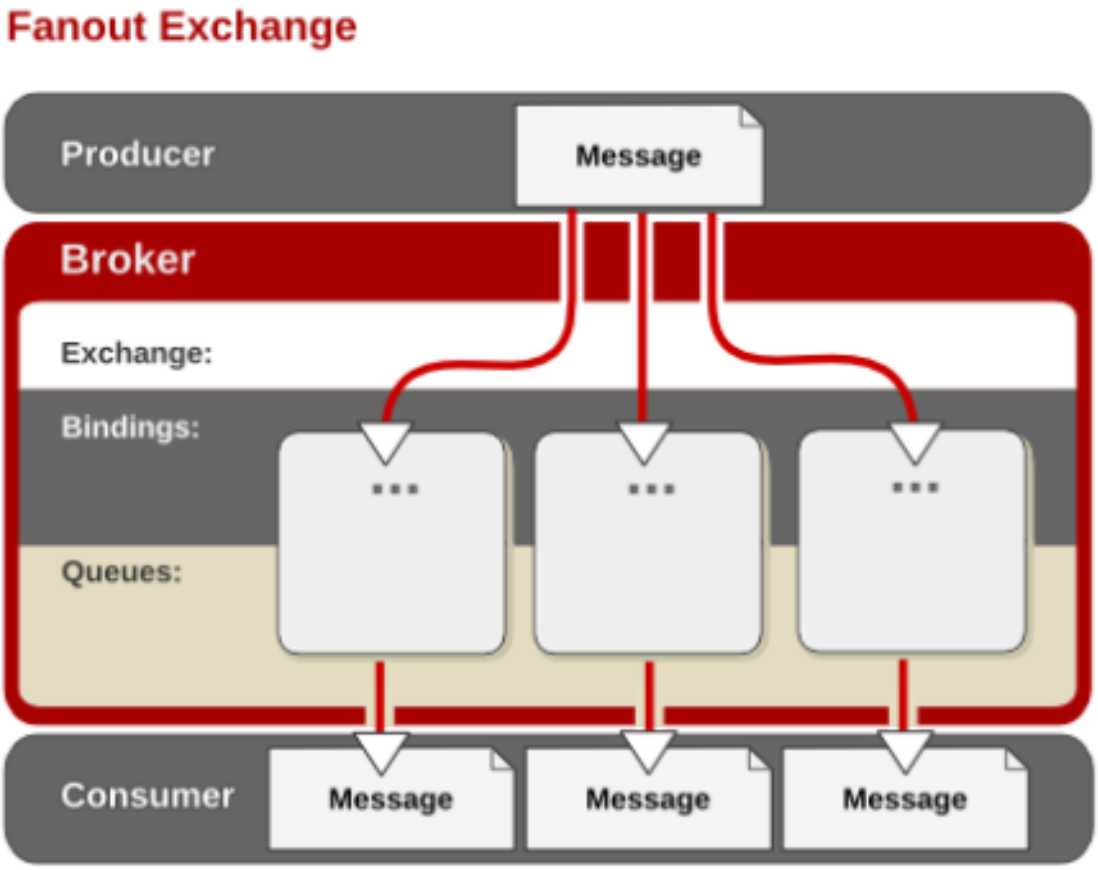
Red Hat Enterprise Messaging には、という事前設定済みの交換が同梱されてい **amq.fanout**ます。

バグを報告します。

4.8.2. FANout Exchange

非同期交換は、すべてのメッセージを交換にバインドされたすべてのキューにルーティングします。

図4.2 FANout Exchange



交換は、トピック交換の特殊化です。効果的な交換は、交換にバインドされたすべてのキューがバインドされたトピック交換で、バインディングキー **#** としてワイルドカードを使用します。

バグを報告します。

4.8.3. qpid-config を使用した交換の作成

以下の例では、**qpid-config**以下を使用して新しい Productionout 交換を作成します。

```
qpid-config add exchange fanout my-fanout-exchange
```

durable （ブローカーを再起動する間）交換を行うには、**--durable** オプションを使用します。

```
qpid-config add exchange fanout my-fanout-exchange --durable
```

qpid-config exchanges コマンドは、ブローカーの交換を一覧表示します。

バグを報告します。

4.8.4. アプリケーションからの交換の作成

送信者またはレシーバーのアドレスに次のパラメーターを使用すると、きめ細かな交換をアプリケーションで宣言できます。

- **create: always**
- **node: {type: topic, x-declare: {exchange: *exchange-name*, type: fanout}}**

以下の例では、という名前の新しい FCout 交換を作成するアドレスを示してい **myfanout**ます。

python

```
tx = ssn.sender("myfanout; {create: always, node: {type: topic, x-declare: {exchange: myfanout, type: fanout}}}")
```

バグを報告します。

4.8.5. 送信交換を使用した複数キューへの公開

バルクアウト交換にバインドされたすべてのキューは、交換に送信されたすべてのメッセージのコピーを受け取ります。そのため、Fluentout の交換ですべてのコンシューマーに公開するには、メッセージを交換に送信します。

python

```
import sys
from qpid.messaging import *
con = Connection("localhost:5672")
con.open()
try:
  ssn = con.session()
  tx = ssn.sender("amq.fanout")
  tx.send("Hello to all consumers bound to the amq.fanout exchange")
finally:
  con.close()
```

バグを報告します。

4.8.6. 交換のサブスクライブ

フレームアウト交換にサブスクライブする場合は、以下の 2 つのオプションがあります。

1. 一時サブスクリプションを使用して交換にサブスクライブします。これにより、アプリケーションが切断されたときに破棄される一時的なプライベートキューが作成され、バインドされます。このアプローチは、複数のコンシューマー間でメッセージの責任を共有する必要がない場合や、アプリケーションが稼働していない場合や切断された時に送信されるメッセージを重要としない場合に合理的です。

2. 交換にバインドされたキューにサブスクライブします。これにより、アプリケーションが切断されている場合はキューでメッセージをバッファーでき、キュー内のメッセージに対する責任を複数コンシューマーが共有できるようになります。

プライベート、一時サブスクリプション

プライベートで一時サブスクリプションを実装するには、Fluentout 交換の名前を使用して受信側のアドレスとしてレシーバーを作成します。例：

python

```
rx = receiver("amq.fanout")
```

共有可能なサブスクリプション

コンシューマーアプリケーションを再起動しても維持される共有可能なサブスクリプションを実装するには、キューを作成し、そのキューにサブスクライブします。

qpid-config以下を使用してキューを作成し、バインドすることができます。

```
qpid-config add queue shared-q
qpid-config bind amq.fanout shared-q
```

注記： ブローカーを再起動してもキューを永続化するには、**--durable** オプションを使用します。

これらの **qpid-config** コマンドを実行した後、交換バインディングを表示する場合は、コマンドを使用します。MRG Messaging 2.2 以降では、コマンドを使用し **qpid-config exchanges -b**ます。MRG Messaging 2.3 以降では、コマンドを使用し **qpid-config exchanges -r**ます。

キューを作成してバインドしたら、アプリケーションにこのキューをリッスンするレシーバーを作成します。

python

```
rx = receiver("shared-q")
```

qpid-config以下を使用するのではなく、アプリケーションコードにキューを作成してバインドすることもできます。

AMQP 0-10

python

```
rx = receiver("shared-q;{create: always, link: {x-bindings: [{exchange: 'amq.fanout', queue: 'shared-q'}]}}")
```

AMQP 1.0

C++

```
Receiver receiver = session.createReceiver("amq.fanout;{node: {capabilities:[shared]}, link: {name: 'shared-q'}}");
```

関連項目

- 「Exchange サブスクリプションパターン」

バグを報告します。

4.9. トピック交換

4.9.1. 事前設定されたトピック交換

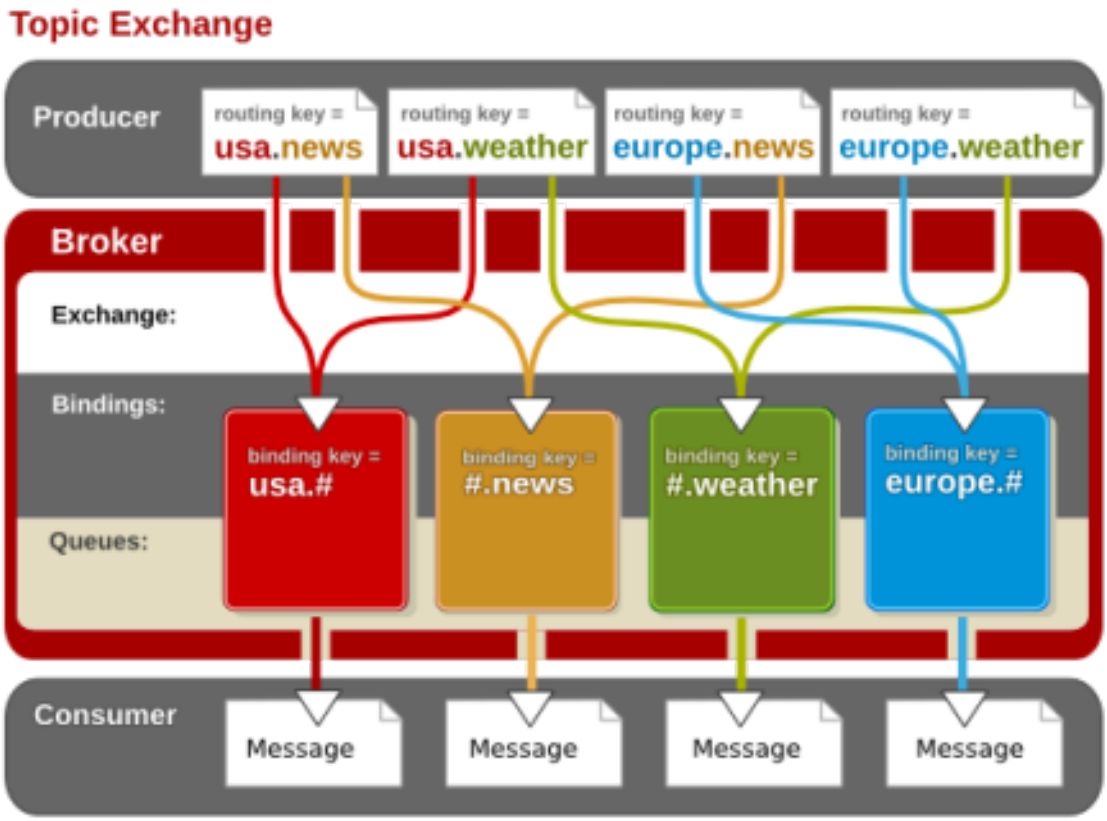
Red Hat Enterprise Messaging には、という事前設定済みのトピック交換が同梱されてい **amq.topic**ます。

バグを報告します。

4.9.2. トピック交換

Topic Exchange は、ダイレクト交換と同様に、メッセージのルーティングキー（サブジェクト）とサブスクリプションのバインディングキーに基づいてメッセージをルーティングします。違いは、トピック交換でバインディングキーでのワイルドカードの使用をサポートしており、柔軟なルーティングスキーマを実装できる点です。

図4.3 トピック交換



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**。

バグを報告します。

4.9.3. qpid-config を使用したトピック交換の作成

以下の **qpid-config** コマンドは、というトピック交換を作成し **news**ます。

```
qpid-config add exchange topic news
```

バグを報告します。

4.9.4. アプリケーションからのトピック交換の作成

以下の例では、というトピック交換を作成し **news**ます。

python

```
txtopic = ssn.sender("news; {create: always, node: {type: topic}}")
```

バグを報告します。

4.9.5. トピック交換への公開

トピック交換にパブリッシュするには、アドレスを交換する送信者を作成してから、メッセージの発行先をルーティングキーに設定します。

以下の例では、メッセージは、コンシューマーによる地理的なサブスクリプションを可能にするルーティングキーを使用して **news** トピックの交換に送信されます。

python

```
import sys
from qpid.messaging import *
conn = Connection("localhost:5672")
conn.open()
try:
  ssn = conn.session()
  txnews = ssn.sender("news; {create: always, node: {type: topic}}")
  msg = Message("News about Europe")
  msg.subject = "europe.news"
  txnews.send(msg)
  msg = Message("News about the US")
  msg.subject = "usa.news"
  txnews.send(msg)
finally:
  conn.close()
```

バグを報告します。

4.9.6. トピック交換のサブスクライブ

トピックの交換をサブスクライブするには、キューを作成し、必要なルーティングキーで交換にバインドします。

以下の例では、という名前のキュー **qpid-config** を作成 **news** し ***everything*.news**、一致するワイルドカードで **amq.topic** 交換にバインドします。ここでは、 すべてがピリオドの用語です。

```
qpid-config add queue news
qpid-config bind amq.topic news "#.news"
```

これで、ルーティングキーが **.news**以下で終わるすべてのメッセージの **news** キューをリッスンできます。

python

```
rxnews = ssn.receiver("news")
```

また、以下の例のようなアドレスを使用して、コード内で操作全体を行うこともできます。

AMQP 0-10

python

```
rxnews = ssn.receiver("news;{create: always, node: {type:queue}, link:{x-bindings:[{exchange:
'amq.topic', queue: 'news', key: '#.news'}]}}")
```

AMQP 1.0

C++

```
Receiver rxnews = ssn.createReceiver("amq.topic/#.news;{node:{capabilities:[shared]}, link:
{name: 'news'}}");
```

また、アプリケーションが切断されたり、メッセージの責任を共有したりする際にメッセージをキューに入れないように、アプリケーションの一時サブスクリプションを作成することもできます。このメソッドは、アプリケーションの一時的なプライベートキューを作成し、バインドします。

python

```
rxnews = ssn.receiver("amq.topic/#.news");
```

トピック交換キーのワイルドカード一致では、**#** シンボルは任意の数のピリオド用語に一致します。**#** は 1 つの用語に一致します。

関連項目

- 「Exchange サブスクリプションパターン」

バグを報告します。

4.10. ヘッダー交換

4.10.1. 事前設定されたヘッダー交換

Red Hat Enterprise Messaging には、という事前設定された永続ヘッダー交換が同梱されてい**amq.match**ます。

バグを報告します。

4.10.2. ヘッダー交換

Headers Exchange は、メッセージヘッダーのプロパティーとのマッチに基づいてルーティングを許可します。これにより、任意のドメイン固有のメッセージの属性に基づいた柔軟なルーティングスキーマが可能になります。

バグを報告します。

4.10.3. qpid-config を使用したヘッダー交換の作成

以下の **qpid-config** コマンド例は、というヘッダー交換を作成し **property-match**ます。

```
qpid-config add exchange headers property-match
```

バグを報告します。

4.10.4. アプリケーションからのヘッダー交換の作成

以下のコードは、というヘッダー交換を作成し **headers-match**ます。

python

```
txheaders = ssn.sender("headers-match;{create: always, node: {type: topic, x-declare: {exchange: headers-match, type: headers}}}")
```

バグを報告します。

4.10.5. ヘッダー交換への公開

ヘッダー交換にパブリッシュするには、エクスポーターコンストラクターに交換の名前を渡し、ヘッダーキーおよび値をメッセージに追加し **properties**ます。例：

python

```
import sys
from qpid.messaging import *
conn =  Connection("localhost:5672")
conn.open()
try:
  ssn = conn.session()
  txheaders = ssn.sender("amq.match")
  msg = Message("Headers Exchange message")
  msg.properties['header1'] = 'value1'
  txheaders.send(msg)
finally:
  ssn.close()
```

バグを報告します。

4.10.6. ヘッダー交換のサブスクライブ

変更

- 2013 年 4 月更新されました。
- 2013 年 7 月の更新

以下のコードはキューを作成し **match-q**、値がのヘッダーキーを **header1** 持つメッセージに一致するバインディングキーを使用して、**amq.match** 交換にサブスクライブし **value1**ます。

AMQP 0-10

python

```
rxheaders = ssn.receiver("match-q;{create: always, node: {type: queue}, link:{x-bindings:[{key: 'binding-name', exchange: 'amq.match', queue: 'match-q', arguments:{'x-match': 'any', 'header1': 'value1'}}]}}")
```

AMQP 1.0

C++

```
Receiver rxheaders = ssn.createReceiver("amq.match; {link: {name:match-q, filter:{value:{'x-match': 'any', 'header1': 'value1'}, name: headers, descriptor:'apache.org:legacy-amqp-headers-binding:map'}}}");
```

x-match 引数は **any**、バインディングのキーと値 **の** ペアを持つメッセージと一致する値を取るか **all**、またはヘッダーのバインディングキーから **すべて** のキーと値のペアを持つメッセージと一致する値を取ることができます。

同じヘッダーの複数の値に対して一致させることはできないことに注意してください。複数のヘッダーを異なる値で使用できますが、特定のヘッダーと照合できる値は 1 つだけです。

AMQP 1.0 はリンクスコープに対応していないため **x-binding**、フィルターが使用されます。

AMQP 0-10 はリンクスコープを使用します **x-binding**。**x-bindings** 引数に注意してください **key**。この引数はバインディングの名前付きハンドルを作成します。これは、実行時に表示され **qpid-config exchanges -r**ます。ハンドルがない場合、バインディングは名前で削除できません。**null** キーは有効ですが、名前で名前で削除できないだけでなく、ハンドルを使用してバインディングを作成すると、**null** その交換で **null** ハンドルでバインディングを作成しようとすると、新しいバインディングを作成するのではなく、既存のバインディングが更新されます。

バグを報告します。

4.11. XML 交換

4.11.1. カスタムの交換タイプ

AMQP Messaging はカスタムの交換タイプをサポートします。カスタムの交換では、どの基準に基づいてメッセージを操作したり、一致したりすることができます。

Red Hat Enterprise Messaging には、カスタムの交換タイプである *XML Exchange* が 1 つ含まれます。

バグを報告します。

4.11.2. 事前設定された XML 交換タイプ

Red Hat Enterprise Messaging には、カスタム XML Exchange **タイプ** が同梱されています。

XML Exchange は、ヘッダーまたはメッセージコンテンツに適用される XQuery に基づくメッセージと一致します。XML データを含むメッセージは、この交換に送信でき、メッセージの内容やメッセージヘッダーを基にフィルターされます。

バグを報告します。

4.11.3. XML 交換の作成

以下の **qpid-config** コマンド例は、という XML 交換を作成し **myxml**ます。

```
qpid-config add exchange xml myxml
```

以下のサンプルコードは、アプリケーションで同じ方法で実現する方法を示しています。

python

```
tx = ssn.sender("myxml; {create: always, node: {type: topic, x-declare: {exchange: myxml, type: xml}}}")
```

バグを報告します。

4.11.4. XML Exchange のサブスクライブ

以下のコードはキューを作成 **myxml** し、XQuery **xmlq** と交換にバインドすることで XML 交換にサブスクライブします。

AMQP 0-10

python

```
rxXML = ssn.receiver("myxmlq; {create:always, link: { x-bindings: [{exchange:myxml, key:weather, arguments:{xquery:'./weather'} }]}}")
```

AMQP 1.0

C++

```
Receiver rxXML = ssn.createReceiver("myxml/weather; {link: {name:myxmlq, filter:{name:myfilter, descriptor:'apache.org:query-filter:string', value:'./weather'}}}");
```

XQuery **./weather** は、ボディーコンテンツがルート XML 要素を持つすべてのメッセージと一致し **<weather>**ます。

の **key** 引数に注意してください **x-bindings**。これにより、バインディングに一意の名前があり、名前で削除および更新でき、交換の namespace で匿名であった場合のように、誤って更新されないようにします。

以下のコードは、より複雑な XQuery （AMQP 0-10 アドレス指定を使用）で XML 交換を使用する ことを示しています。

python

```
#!/usr/bin/python
import sys
from qpid.messaging import *

conn = Connection("localhost:5672")
conn.open()
try:
  ssn = conn.session()
  tx = ssn.sender("myxml/weather; {create: always, node: {type: topic, x-declare: {exchange:
myxml, type: xml}}}")

  xquerystr = 'let $w := ./weather '
  xquerystr += "return $w/station = 'Raleigh-Durham International Airport (KRDU)' "
  xquerystr += 'and $w/temperature_f > 50 '
  xquerystr += 'and $w/temperature_f - $w/dewpoint > 5 '
  xquerystr += 'and $w/wind_speed_mph > 7 '
  xquerystr += 'and $w/wind_speed_mph < 20'

  rxaddr = 'myxmlq; {create: always, '
  rxaddr += 'link: {x-bindings: [{exchange: myxml, '
  rxaddr += 'key: weather, '
  rxaddr += 'arguments: {xquery: "' + xquerystr + '"'
  rxaddr += '}}]}}'

  rx = ssn.receiver(rxaddr)

  msgstr = '<weather>'
  msgstr += '<station>Raleigh-Durham International Airport (KRDU)</station>'
  msgstr += '<wind_speed_mph>16</wind_speed_mph>'
  msgstr += '<temperature_f>70</temperature_f>'
  msgstr += '<dewpoint>35</dewpoint>'
  msgstr += '</weather>'

  msg = Message(msgstr)

  tx.send(msg)

  rxmsg = rx.fetch(timeout=1)
  print rxmsg

  ssn.acknowledge()

finally:
  conn.close()
```

バグを報告します。

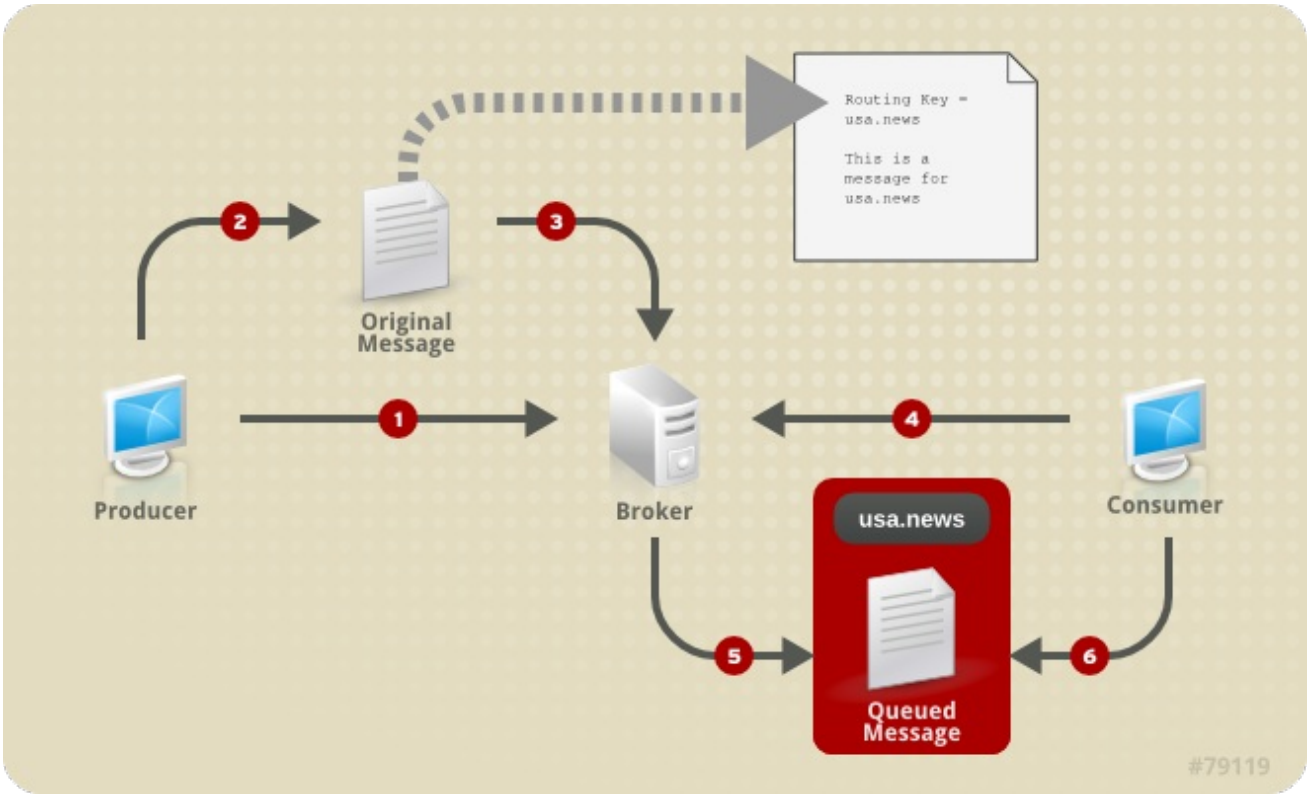
第5章 メッセージ配信と許可

5.1. メッセージライフサイクル

5.1.1. メッセージ配信の概要

以下の図は、メッセージ配信ライフサイクルを示しています。

図5.1 FANout Exchange



メッセージプロデューサーはメッセージを生成します。メッセージとは、コンテンツ、サブジェクト、およびヘッダーが含まれるオブジェクトです。少なくとも、メッセージプロデューサーはメッセージコンテンツが含まれるメッセージを生成します。

メッセージプロデューサーはメッセージをブローカーに送信し、メッセージのプロパティーまたはメッセージの送信に使用される送信側のオブジェクトのアドレスによってルーティングを処理させることができます。

または、メッセージプロデューサーが **message.subject**、ルーティングキー(2)として機能するを設定し、メッセージをブローカー(3)に送信します。

コンシューマーは、接続時にメッセージを交換（バックグラウンドで一時的なプライベートキューを使用する）をサブスクライブしています(4)。

メッセージは(5)を交換するためにサブスクライブされるキューでバッファーされます。コンシューマーは、コンシューマーが切断されたときにバッファーされたメッセージをキューにサブスクライブし、受信することができます。これらのキューは、コンシューマー間でメッセージの責任を共有するためにも使用できます。

バグを報告します。

5.1.2. メッセージの生成

Message オブジェクトは、メッセージを生成するために使用されます。

python

```
import sys
from qpid.messaging import *
...
msg = Message('This is the message content')
msg.content = 'Message content can be assigned like this'
msg.properties['header-key'] = 'value'

tx = ssn.sender('amq.topic')

# msg.subject set by sender for routing purposes
tx.send(msg)

msg.subject = 'Messaging Routing Key can also be manually set'
# beware that this will interfere with sender-object-based routing
```

バグを報告します。

5.1.3. 信頼性のあるリンク上でメッセージを送信する

信頼できるリンクで送信する場合は、以下を行います。

1. 送信側はメッセージをブローカーに渡します。
2. ブローカーは、メッセージの配信に責任があることを確認応答します。
3. 送信側はメッセージのローカルコピーを削除します。

同期操作では、スレッドはブロックされ、確認応答のラウンドトリップが発生します。非同期操作を使用して送信すると、確認と削除はバックグラウンドで実行され、未承認メッセージが受信元再生バッファーでバッファーされ、承認が承認されるまでは送信元再生バッファーでバッファーされます。

バグを報告します。

5.1.4. 信頼性のないリンク上でメッセージを送信する

信頼できないリンクで送信される場合：

1. 送信側はメッセージをブローカーに渡します。
2. 送信側はメッセージのローカルコピーを削除します。

このモードでは、送信元とブローカーの間でメッセージが失われる可能性があります。

バグを報告します。

5.1.5. Broker のメッセージ配布

ブローカーがメッセージを受信すると、メッセージと、それに関連するルーティング情報を検査し、配信方法を決定します。

メッセージを受信した交換のバインディングが調査され、メッセージとバインディングが一致すると、そのバインディングのあるキューにメッセージが送信されます。

バグを報告します。

5.1.6. 信頼性のあるリンクのメッセージ受信

信頼できるリンクでメッセージが受信された場合：

1. ブローカーはメッセージをレシーバーに渡します。

これ以降、受信側がコンシューマーを取得する際に、数多くの可能性が存在します。

1. 受信側はメッセージの *責任* を承認します。この場合、ブローカーはメッセージのサーバー側のコピーを削除します。
2. レシーバーはメッセージを *拒否* します。この場合、ブローカーはメッセージをキューに定義された **alternate-exchange** 場合にメッセージをルーティングするか、メッセージを破棄します。
3. レシーバーはメッセージを *解放* します。この場合、ブローカーはメッセージヘッダーのあるキューにメッセージを返し **redelivered:true**ます。
4. メッセージの承認または拒否なしで受信側が切断されます。この場合、ブローカーはメッセージヘッダーのあるキューにメッセージを返し **redelivered:true**ます。

バグを報告します。

5.1.7. 信頼性のないリンク上のメッセージ受信

信頼性のないリンクでメッセージが受信された場合：

1. ブローカーはメッセージをレシーバーに渡します。
2. ブローカーはメッセージのサーバー側のコピーを削除します。

受信側がメッセージを拒否する機会はありません。信頼できないリンクを使用する場合は、ブローカーがこれを再配信する機会はありません。

バグを報告します。

5.2. メッセージの閲覧とコンシューマー

5.2.1. メッセージ取得および許可

メッセージコンシューマーはキュー内のメッセージを *閲覧し* たり、それらを *消費* したりできます。

ブラウズ とは、消費するアプリケーションがメッセージを読み取りますが、メッセージが他のコンシューマーのためにキューに残ることを意味します。*消費する* アプリケーションは、キューからメッセージを削除することを意味します。これは、メッセージの *取得* とも呼ばれます。

まず、メッセージの閲覧と取得の相違点を 取得および承認 確認し、その後に、理解する必要のある 2 つのフェーズを持つ一連のプロセスについて詳しく説明します。

閲覧

含まれる **drain** プログラムは、browse モードまたは layers モードのいずれかで使用できます。

ドレイン（解放）のソースコードは、C++ および Python クライアントライブラリーパッケージに含まれます。C++ ソースコードをコンパイルしたり、Python インタープリターを使用して Python ソースをコンパイルすることもできます。

クライアントライブラリーパッケージがインストールされている場合は、以下を参照し **drain** てください。

```
/usr/share/doc/python-qpid-0.14/examples/api/drain
/usr/share/qpidc/examples/messaging/drain.cpp
```

閲覧と閲覧の違いを実証するには、以下を試行します。

ブローカーがインストールされ、実行中に、**qpid-config** コマンドでキューを作成します。

```
qpid-config add queue browse-acquire-demo
```

実行時に **browse-acquire-demo** キューが表示されるはずです **qpid-config queues**。

ここで、**browse-acquire-demo** 使用方法にメッセージを送信し **spout**ます **Spout**。はと同じパッケージに含まれ **drain**、同じディレクトリーにあります。spout を実行してメッセージをキューに送信します。

```
./spout browse-acquire-demo "Hello World"
```

この「Hello World」メッセージが **browse-acquire-demo** キューに送信されました。ドレイン（解放）を使用して、すべての最初の 1 つを *参照し* ます。

```
./drain -c 0 "browse-acquire-demo; {mode:browse}"
```

これで「Hello World」メッセージが表示されます。上記 **drain** を 2 度実行し、再度メッセージが表示されます。ドレイン（解放）プログラムを実行すると、キューにアクセスする 2 つの異なる閲覧コンシューマーをシミュレートします。メッセージは読み取り、閲覧中の他のアプリケーションで使用できる状態のままになります。

qpid-config以下を使用して、**browse-acquire-demo** キューの削除を試行します。

```
qpid-config del queue browse-acquire-demo
```

qpid-config メッセージがキューに残るため、エラーで応答します。

これで、この drain コマンドを実行します。

```
./drain -c 0 "browse-acquire-demo"
```

デフォルトのモードは burst *です*。モードが指定されていない状態でドレイン（解放）を実行すると、メッセージを *取得* します。「Hello World」メッセージが表示されます。ただし、今回はメッセージが削除されています。ドレイン（解放）を使用して再度参照を試みます。キューは空です。

qpid-config以下を使用して、now-empty キューを削除できます。

```
qpid-config del queue browse-acquire-demo
```

drain demo で表示されないことの 1 つは、ブラウザーに一度だけメッセージが表示されることです。実行するたびに異なるブラウザー **drain** が作成されるため、キューに毎回メッセージが出力されます。ただし、同じブラウザーは、表示される回数に関係なく、一度だけメッセージを確認します。

以下の Python コードは、閲覧および取得方法を示し、ブラウザーがメッセージを 1 度確認する方法を示しています。

python

```
import sys
from qpid.messaging import *

def msgfetch(rx):
  try:
    msg = rx.fetch(timeout=1)
  except MessagingError, m:
    msg = m
  return msg

connection = Connection("localhost:5672")
connection.open()
try:
  session = connection.session()
  tx = session.sender("browse-acquire-demo;{create:always}")
  rxbrowse1 = session.receiver("browse-acquire-demo;{mode:browse}")
  rxbrowse2 = session.receiver("browse-acquire-demo;{mode:browse}")
  rxbrowse3 = session.receiver("browse-acquire-demo;{mode:browse}")
  rxacquire = session.receiver("browse-acquire-demo")

  tx.send("Hello World")

  print "\nBrowser 1 saw message:"
  print msgfetch(rxbrowse1)

  print "Browser 1 then saw message:"
  print msgfetch(rxbrowse1)

  print "\nBrowser 2 saw message:"
  print msgfetch(rxbrowse2)

  print "Browser 2 then saw message:"
  print msgfetch(rxbrowse2)

  print "\nAcquired message:"
  print msgfetch(rxacquire)

  print "\nBrowser 3 saw message:"
  print msgfetch(rxbrowse3)

except MessagingError, m:
  print m
finally:
  connection.close()
```

ブラウザー 1 とブラウザー 2 の両方がメッセージを表示し、それぞれ 1 回だけ表示されます。Browser 3 がキューを確認する前にメッセージが取得されるため、ブラウザー 3 ではキューのメッセージは表示されません。

ただし、実行 **drain** してキューを調べます。

```
./drain -c 0 browse-acquire-demo
```

キューにメッセージが依然として表示される場合もあります（途中で削除します）。何が発生しましたか？

取得および承認

レシーバーがキューからメッセージを取得すると、ブローカーはメッセージをに設定し **acquired**ます。メッセージがある場合 **acquired**、ブローカーはメッセージを配信されたかのように処理しますが、キューから削除されません。メッセージを受信したコンシューマーがメッセージを承認するか、メッセージを *解放* するか、*メッセージを 拒否* するか、コンシューマーがネットワーク障害で切断する可能性があります。

この場合、メッセージは承認されずに、アプリケーションがブローカーから切断されています。アプリケーションが接続されている間、メッセージは接続され **acquired**、メッセージコンシューマーがキューからアクセスまたはフェッチしてもメッセージは表示されません。受信を許可せずにアプリケーションが切断されると、ブローカーはメッセージを **acquired** 状態から切り替え、ヘッダーを設定し **redelivered=True**ます。その後、アプリケーションを閉じた後に実行した **drain** ブラウザーなど、他のコンシューマーがメッセージを利用できるようになります。

「収集、確認応答」パターンのこの目的は、信頼できるメッセージの配信を提供することです。ノードのグループがメッセージによって駆動されるサービスを実行する状況を考えてみましょう。ワークを実行できる容量があれば、ワークグループの各ノードはキューから多数のメッセージを取得します。ノードはキューからのメッセージの一部を取得し、停電が生じる可能性があります。この場合、ブローカーに取得と確認という概念がない場合は、これらのメッセージが失われます。このパターンにより、ワーカーノードはメッセージを取得し、一部の作業を実行して所有者を確認し、メッセージが配信され、動作していることを安全に確認することができます。これにより、ウィンドウが制限されます。メッセージに対して動作し、受信を確認する前に重要な時点でノードが失敗した場合でも、他のノードはヘッダー '**redelivered=true**' のキューからメッセージを取得します。このアラートは、このメッセージがすでに動作している **可能性** がある他のノードに警告し、チェックを実行して、それを確認します。これにより、アプリケーションがこの機能を活用するように設計された場合でも、例外のウィンドウが制限されます。

アプリケーション内でメッセージを取得せずにコンシューマーが切断したときにキューに返すメッセージを表示するには、最終的な **connection.close()** 行の後に以下のコードをアプリケーションの最後に追加します。

python

```
connection.open()
try:
  session=connection.session()

  rxacquire2 = session.receiver("browse-acquire-demo")
  print "\nAcquirer 2 saw message:"
  print msgfetch(rxacquire2)
except MessagingError, m:
  print m
finally:
  session.acknowledge()
  connection.close()
```

-

アプリケーションはその接続を閉じ、メッセージを受け取らないようにブローカーからコンシューマーを切断します。その後、ブローカーへの新しい接続を開き、新しいコンシューマーとして効果的に表示されます。これで受信側は、ブローカーによってマークされたメッセージを認識し、別のコンシューマーがこのメッセージ **redelivered** を以前に取得したことを通知します。これでこのメッセージを取得し、他のコンシューマーがこのキューを閲覧したりフェッチしたりするために、このメッセージを再取得します。ただし、今回は接続を閉じる **session.acknowledge()** 前に呼び出します。このメソッドはメッセージの受信を認識します（すべてのメッセージはセッションに対して承認されていないものとして確認されます）。メッセージへの受信は確認済みなので、メッセージはキューから削除され **acquired**ています。

drain ここで実行すると、キューにメッセージがないことが確認できます。

メッセージの解放

コンシューマーはメッセージを明示的に **リリース** できます。これが発生すると、メッセージは再配信のキューに返されます。この効果は、コンシューマーがブローカーへの接続を失うかどうかと同じです。

Python API でメッセージを明示的に解放するには、メッセージおよび **Disposition(RELEASED)** パラメーターを指定して **acknowledge()** メソッドを呼び出します。

```
session.acknowledge(msg, Disposition(RELEASED))
```

C++ API でメッセージを明示的に解放するには、セッションの **release()** メソッドを呼び出します。

リンクの信頼性

これは、**信頼できる** リンク（通常は *at-least-once* リンク）上の動作です。これは、ブローカーへの受信側接続のデフォルトリンクです。*信頼でき* ないリンクを使用して受信側をキューに接続する場合、または交換に直接接続する場合、受信したメッセージは即座に取得され、確認する必要がありません。

デモキューのクリーンアップ

このデモに使用したキューを削除するには、ブローカーを再起動するか（ブローカーの再起動時に非**durable** キューがすべて削除される）か、**qpid-config**以下を使用できます。

```
qpid-config del queue browse-acquire-demo
```

キューに残っているメッセージがある場合、このコマンドは失敗します。キューが空でないことを示すメッセージが表示されます。**--force** スイッチを使用してこのチェックをオーバーライドし、そのメッセージのあるキューを削除するか、またはキューを空のまま **drain** にしてから、現在空のキューでコマンドを再発行することができます。

バグを報告します。

5.2.2. 信頼性のないリンクのメッセージ取得および許可

レシーバーとブローカー間のデフォルトのリンクは、信頼できるリンク（ *at-least-once* 信頼性のあるリンクとして知られています）です。このリンクは、2 フェーズ取得および確認応答の動作を使用して、ブローカーがキューからそれを削除する前に、メッセージの責任がコンシューマーによって明示的に受け入れられるようにします。

レシーバーとブローカー間の *信頼できない* リンクを要求することもできます。信頼性のないリンクでは、メッセージは確認応答とみなされ、コンシューマーがキューからメッセージを取得するとすぐに取得されます。レシーバーが明示的に認識しない場合には、メッセージがキューに戻る取得フェーズはあ

りません。ブローカーは、確認応答を待たずにコンシューマーが承認されたことを認識し、コンシューマーがそれを取得するときにメッセージを削除します。このリンクにより信頼性は低下しますが、スループットが向上する可能性があります。コンシューマーに障害が発生した場合にメッセージが失われる場合に役立ちます。

信頼できないリンクを要求するには、アドレス **link: {reliability: unreliable}** にを指定します。たとえば、「browse-acquire-demo」という名前のキューへの信頼できないリンクを持つレシーバーを作成するには、次のコマンドを実行します。

python

```
rxacquire = session.receiver("browse-acquire-demo; {link:{reliability: unreliable}")
```

以下のプログラムは、信頼性のないリンクを使用するレシーバーの使用および動作を示しています。

python

```
import sys
from qpid.messaging import *

def msgfetch(rx):
  try:
    msg = rx.fetch(timeout=1)
  except MessagingError, m:
    msg = m
  return msg

linktype=""
while linktype != "R" and linktype !="U":
  response = raw_input("Use (R)eliable or (U)nreliable link [R/U]?")
  linktype = response.upper()

connection = Connection("localhost:5672")
connection.open()
try:
  session = connection.session()
  tx = session.sender("browse-acquire-demo;{create: always}")
  rxbrowse1 = session.receiver("browse-acquire-demo;{mode:browse}")
  rxbrowse2 = session.receiver("browse-acquire-demo;{mode:browse}")
  rxbrowse3 = session.receiver("browse-acquire-demo;{mode:browse}")
  if linktype == "R":
    rxacquire = session.receiver("browse-acquire-demo")
  else:
    rxacquire = session.receiver("browse-acquire-demo; {link:{reliability:unreliable}}")

  tx.send("Hello World")

  print "\nBrowser 1 saw message:"
  print msgfetch(rxbrowse1)

  print "Browser 1 then saw message:"
  print msgfetch(rxbrowse1)

  print "\nBrowser 2 saw message:"
  print msgfetch(rxbrowse2)
```

```
  print "Browser 2 then saw message:"
  print msgfetch(rxbrowse2)

  print "\nAcquired message:"
  print msgfetch(rxacquire)

  rxacquire.close()

  print "\nBrowser 3 saw message:"
  print msgfetch(rxbrowse3)

except MessagingError, m:
  print m
finally:
  connection.close()

connection.open()
try:
  session=connection.session()

  rxacquire2 = session.receiver("browse-acquire-demo")
  print "\nAcquirer 2 saw message:"
  print msgfetch(rxacquire2)

except MessagingError, m:
  print m
finally:
  session.acknowledge()
  connection.close()
```

デモの信頼できるリンクを選択すると、Acquirer 2 は再配信メッセージを表示します。

```
Acquirer 2 saw message:
Message(redelivered=True, properties={'x-amqp-0-10.routing-key': u'browse-acquire-demo'},
content='Hello World')
```

最初の取得者は切断前にメッセージの送信を承認しなかったため、ブローカーは再配信のためにメッセージをキューに返しています。

デモ用に信頼できないリンクを選択すると、Acquirer 2 でメッセージは表示されません。

```
Acquirer 2 saw message:
None
```

信頼性のないリンクでは、最初のパッシブがストリーミングを承認することでメッセージの責任を明示的に受け入れていませんでしたが、ブローカーはメッセージをキューから削除しています。これは意味です **unreliable**。

信頼性のないリンクでのメッセージの解放および拒否

信頼できないリンクで取得したメッセージをリリースまたは拒否することはできません。信頼性のないリンクメッセージは、フェッチ時に暗示的に承認されます。

バグを報告します。

5.2.3. メッセージ拒否

信頼できるリンクでメッセージを取得すると、アプリケーションを *拒否する* ことができます。メッセージが拒否されると、ブローカーはそのメッセージをキューから削除します。キューがの状態で設定されている場合 **alternate exchange**、拒否されたメッセージがそこにルーティングされ、それ以外の場合は破棄されます。

Python API を使用してメッセージを拒否するには、セッションの **acknowledge()** メソッドを呼び出して、拒否するメッセージを渡し、**Disposition** パラメーター **REJECTED** として指定します。

python

```
msg = rx.fetch(timeout = 1)

if msg.content == "something we don't like":
  ssn.acknowledge(msg, Disposition(REJECTED))
else:
  ssn.acknowledge(msg)
```

これは、信頼できるリンクを使用する場合にのみ可能になることに注意してください。**unreliable** リンクを使用する場合、mesage はフェッチ時に暗黙的に承認されます。

バグを報告します。

5.2.4. 複数のソースからのメッセージの受信

前提条件

- 「Enable Receiver Prefetch」

Receiver オブジェクトは単一のサブスクリプションからメッセージを受信します。アプリケーションは多くのレシーバーを作成でき、これらの受信者からのメッセージを受信順に処理したい場合があります。**session** オブジェクトは、アプリケーション **nextReceiver** が複数のレシーバーからメッセージをフェデレーション順に読み込む方法を提供します。

注記： 次の受信 **prefetch** 側機能を使用するには、受信側に対して有効にし、受信側が同じセッションを使用している必要があります。

python

```
receiver1 = session.receiver(address1)
receiver1.capacity = 10
receiver2 = session.receiver(address)
receiver2.capacity = 10
message = session.next_receiver().fetch()
print message.content
session.acknowledge()
```

C++

```
Receiver receiver1 = session.createReceiver(address1);
receiver1.setCapacity(10);
Receiver receiver2 = session.createReceiver(address2);
receiver2.setCapacity(10);
```

```
Message message =  session.nextReceiver().fetch();
std::cout << message.getContent() << std::endl;
session.acknowledge(); // acknowledge message receipt
```

.NET/C#

```
Receiver receiver1 = session.CreateReceiver(address1);
receiver1.SetCapacity(10);
Receiver receiver2 = session.CreateReceiver(address2);
receiver2.SetCapacity(10);

Message message = new Message();
message =  session.NextReceiver().Fetch();
Console.WriteLine("{0}", message.GetContent());
session.Acknowledge();
```

バグを報告します。

5.2.5. 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。

5.2.6. 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。

第6章 高度なキューの機能

6.1. 参照専用キュー

「browse-only」と宣言されたキューを使用すると、サブスクライバーはメッセージにアクセスして通常どおりメッセージを取得できますが、メッセージは透過的に結果を閲覧するだけです。このメッセージはキューに残り、他のサブスクライバーからアクセスできます。

メッセージは、一部の収集メカニズムによってのみ閲覧専用キューから削除できます。たとえば、メッセージの TTL（有効期間）の期限が切れる場合などです。

spout および **drain** プログラムはクライアントライブラリーパッケージの一部であり、インストールされている場合は、次の場所にあります。

```
/usr/share/doc/python-qpid-${version}/examples/api/
```

以下は、スプリットおよびドレインクライアントによる参照のみキューの作成および使用例です。

```
./spout \
    -c 10 \
    --broker "localhost:${PORT}" \
    'q; {create: always, node:{type:queue , x-declare:{arguments:{"qpid.browse-only":1}}}}' \
    "All work and no play makes Mick a dull boy."

    ./drain --broker 'localhost:${PORT}' 'q'
```

関連項目

- 「メッセージの閲覧とコンシューマー」

バグを報告します。

6.2. 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。

6.3. 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。

6.4. サーバー側のセレクター

6.4.1. フィルターを使用したメッセージの選択

MRG 3 は、サーバー側のセレクターを使用してキューからメッセージを選択することをサポートします。これにより、SQL のような構文を使用してフィルターを指定できます。このフィルターは、サーバー上のメッセージのヘッダーおよびプロパティーに適用されます。フィルターに一致するメッセージはクライアントに配信されます。

サーバー側のセレクターを使用するには、接続 URL **selector** のリンク部分にを指定します。

以下の例は、、、**green red**、またはをプロパティーの値 **blue** としてサーバーを選択して配信し **color** ます。

```
queue_name;{link:{selector:"color in ('green', 'red', 'blue')"}}
```

以下の例は、さまざまなヘッダープロパティーでメッセージをフィルターするセレクターを示しています。

```
queue_name;{link:{selector:"amqp.priority = 1"}}
queue_name;{link:{selector:"amqp.priority IS BETWEEN 3 AND 6"}}
queue_name;{link:{selector:"myflag AND amqp.redelivered"}}
queue_name;{link:{selector:"msg_title LIKE '%news%'"}
```

Python および一時構文

Python では、セレクターは一時構文で使用できます。たとえば、selector の C++ アドレスは以下のようになります。

```
queue_name;{link:{selector:"myproperty = 1"}}
```

Python では、以下のように一時的に使用されます。

```
queue_name;{link:{'x-subscribe': {'arguments': {'x-apache-selector': "myproperty = 1"}}}}
```

Java およびサーバー側のセレクター

Qpid Java クライアントは現在サーバー側のセレクターをサポートしておらず、JMS セレクターのみをサポートします。JMS セレクターはサーバー側のセレクターとは異なります。JMS slector の詳細については、JMS 仕様を参照してください。

関連項目

- 「Java Message Service with Filters」

バグを報告します。

6.4.2. サーバー側のセレクター構文

以下は、サーバー側のセレクターの非公式構文です。

```
SelectExpression ::= OrExpression? // Note 0

// Lexical Elements
Alpha ::= [a-zA-Z]
Digit ::= [0-9]

IdentifierInitial ::= Alpha | "_" | "$"
IdentifierPart ::= IdentifierInitial | Digit | "."
Identifier ::= IdentifierInitial IdentifierPart*
Constraint : Identifier NOT IN ("NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE",
"IN", "IS", "ESCAPE")  // Note 1


LiteralString ::= ("'" [^']* "'")+  // Note 2
LiteralExactNumeric ::= Digit+
Exponent ::= ("+"|"-")? LiteralExactNumeric
LiteralApproxNumeric ::=  Digit "." Digit* ( "E" Exponent )?  |
                          "." Digit+ ( "E" Exponent )?  |
                          Digit+ "E" Exponent   // Note 1
LiteralBool ::= "TRUE" | "FALSE"   // Note 1
Literal ::= LiteralBool | LiteralString | LiteralApproxNumeric | LiteralExactNumeric


EqOps ::= "=" | "<>"
ComparisonOps ::= EqOps | ">" | ">=" | "<" | "<="
AddOps ::= "+" | "-"
MultiplyOps ::= "*" | "/"

// Expression syntax
OrExpression ::= AndExpression  ( "OR" AndExpression )*
AndExpression ::= ComparisonExpression ( "AND" ComparisonExpression )*
ComparisonExpression ::= AddExpression "IS" "NOT"? "NULL" |
                         AddExpression "NOT"? "LIKE" LiteralString ( "ESCAPE" LiteralString )? |
                         AddExpression "NOT"? "BETWEEN" AddExpression "AND" AddExpression |
                         AddExpression "NOT"? "IN" "(" PrimaryExpression ("," PrimaryExpression)* ")" |
                         AddExpression ComparisonOps AddExpression |
                         "NOT" ComparisonExpression |
                         AddExpression   // Note 3


AddExpression ::= MultiplyExpression (  AddOps MultiplyExpression )*
MultiplyExpression ::= UnaryArithExpression ( MultiplyOps UnaryArithExpression )*
```

```
UnaryArithExpression ::= AddOps AddExpression |
                         "(" OrExpression ")" |
                         PrimaryExpression
PrimaryExpression ::= Identifier |
                      Literal
```

注記 0: 全体の式が空であれば true に評価され、空白は無視されます。空のセレクターは always true と解釈されます。

注記 1: 指数およびブール値の「**E**」など、予約されて **false**いるすべての単語は大文字 **true** と小文字を区別しません。

注記 2: このパターンの継続により、一重引用符が埋め込まれます。単一引用符は、 : become のように埋め込むこと **''** ができます **'**。

注記 3: ("ESCAPE" LiteralString) 節で **LiteralString** は、1 つの文字列に限定されます。文字 **%** および _ は使用できません。

バグを報告します。

6.5. 自動的に削除されたキュー

6.5.1. 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。

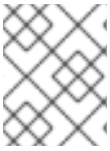
自動削除されたキューは通常、メッセージを **受信** するためにアプリケーションによって作成されます。たとえば、サービスから情報をリクエストするときにメッセージの「reply-to」プロパティーに指定する応答キューです。このシナリオでは、アプリケーションは独自の使用のためにキューを作成し、交換をサブスクライブします。コンシューマーアプリケーションが停止しると、キューは自動的に削除されます。メッセージブローカーから情報を受信する **qpid-config** ユーティリティーが作成したキューはこのパターンの例です。

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}}}')
```

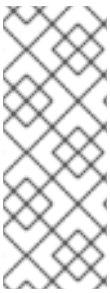


注記

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。

6.5.2. 自動的に削除されたキューの例

以下の Python コードは、自動削除キューの動作を示しています。自動削除キューは、アプリケーションが終了するとブローカーによってクリーンアップされます。通常、これらは交換にサブスクライブするために使用されます。典型的なユースケースは、メッセージの「reply-to」フィールドに指定する自動削除キューを作成して応答を戻します。

このデモでは、自動削除キューを使用して情報をサブスクライバーに公開します。これは、自動検出する理由により auto-delete キューは典型的な使用ではありません。

以下のコードをコピーして、として保存し **auto-delete-producer.py**ます。Python インタープリターを使用して実行できます。

python

```
import sys
from qpid.messaging import *

connection=Connection("localhost:5672")
connection.open()
try:
  session=connection.session()
  tx=session.sender("test-queue; {create:always, node:{x-declare:{auto-delete:True}}}")
  tx.send("test message!")
  x = raw_input("Press Enter to continue")
  tx.send("test message 2")
except MessagingError, m:
  print m
connection.close()
```

ローカルマシンでブローカーを再起動します。ブローカーを再起動すると、キュー以外のすべての**durable** キューが削除されます。これにより、クリーンな状態でこのテストを開始できます。

コマンドを実行します。

```
qpid-config queues
```

これにより、ブローカーのすべてのキューが一覧表示されます。およびが含まれるランダムな名前を持つ動的生成キューが **exclusive** あり **auto-del**ます。これは、キューのリストの取得に使用して **qpid-config** いるキューで、コマンドを実行するたびに変更されます。

次に、Python インタープリターを使用して **auto-delete-producer.py** プログラムを起動します。

```
python auto-delete-producer.py
```

プログラムが一時停止し、Enter を押すように求められます。Enter を押して続行します。

これで **qpid-config queues** 再度実行し、ブローカーのキューを一覧表示します。これで、このプログラムが作成した **test-queue** ことがわかります。プログラムは終了しましたが、キューは削除されていません。そのため、このプログラムはサブスクライブしていないためです。

amqp1.0 動作には違いがあることに注意してください。キュー **amqp0-10** の使用は、コンシューマーがある場合にのみ使用されない場合に削除されます。**amqp1.0** キューの使用は、コンシューマーがない場合でも使用されていないと削除されます。

これで、**drain** ユーティリティーを使用してキュー上のメッセージを調べます。この **drain** ユーティリティーは、C++ クライアントライブラリーパッケージおよび Python クライアントライブラリーパッケージに含まれます。

drain 実行時はキューにサブスクライブし、メッセージを取得し、サブスクライブを解除します。以下を実行します。

```
drain -c 0 test-queue
```

test-queue からのメッセージが画面に表示されます。**qpid-config queues** ここで実行すると、test-queue が削除されていることを確認できます。キューにサブスクライブし、サブスクライブ解除されたコンシューマー。

プロセスを再度試行し、今回はメッセージを取得 **drain** せずにキューを閲覧します。

```
drain -c 0 "test-queue;{mode:browse}"
```

キューが参照されている場合でも、キューが削除されていることを確認できます。サブスクリプションの閲覧数(acquiring)

これで非常に関心のあることを確認するには、キューにサブスクライブし、プログラムの実行 **中** にサブスクライブ解除します。

以下のコードをファイルにコピーし **auto-delete-subscribe.py**ます。

python

```
import sys
from qpid.messaging import *

connection=Connection("localhost:5672")
connection.open()
try:
  session=connection.session()
  rx=session.receiver("test-queue")
  print rx.fetch(timeout = 1)
  session.acknowledge()
except MessagingError,m:
  print m
connection.close()
```

ここで実行し **auto-delete-producer.py**ます。一時停止したら、を実行 **auto-delete-subscriber.py**します **qpid-config queues**。キューが削除されたことを確認します。

Enter を押して続行します。プログラムが終了したら、**drain** を使用して test-queue を参照します。これは存在しません。

レシーバーの作成および割り当てにより、コンシューマープログラムがキューにサブスクライブし **auto-delete-producer.py** た際に作成された test-queue は削除され、接続を閉じるとサブスクライブ解除されます。メッセージプロデューサーから送信された 2 番目のメッセージは配信され **ず、例外が発生しなくなりました**。

これは認識すべきものです。送信者はメッセージをメッセージブローカーにルーティングするローカルルーターへのハンドルです。送信側のコンストラクターパラメーターはルーティングキーです。このコンストラクターはキューの名前ですが、送信側は常にメッセージを交換にルーティングします。交換を指定しないと、デフォルトの交換が使用されます。これは、ブローカー上で名前なしの直接交換です。送信側のコンストラクターは、指定されたルーティングキーがメッセージブローカーの有効なターゲットを参照することをチェックし、デフォルトの交換で「test-queue」があることをチェックします。送信側が作成されると、このキューが存在することになります。その後、送信側の send メソッドは、ルーティングキーが「test-queue」に設定されたブローカーのデフォルト交換にメッセージをルーティングします。ターゲットの交換は存在するため、送信時に例外が発生しません。ルーティングキーに一致する交換にサブスクライブしているキューがないため、メッセージはブローカーでデフォルトの交換に到達します。

このシナリオを回避するには、パブリッシュに自動削除されていないキューを使用するか、送信元と共に受信側を作成してサブスクライブします。これにより、送信者の有効期間中キューが引き続き存在することが保証されます。プログラムでこれを行うには、送信側がキューを作成した後に直接レシーバーを作成してサブスクライブします。test-queue の存在と状態を確認できる 2 番目の一時停止を追加します。以下は、更新されたプログラムです。

python

```
import sys
from qpid.messaging import *

connection=Connection("localhost:5672")
connection.open()
try:
  session=connection.session()
  tx=session.sender("test-queue; {create:always, node:{x-declare:{auto-delete:True}}}")
  rx=session.receiver("test-queue")
  tx.send("test message!")
  x = raw_input("Press Enter to continue")
  tx.send("test message 2")
  x = raw_input("Press Enter to continue")
except MessagingError, m:
  print m
connection.close()
```

次に、**auto-delete-producer.py** プログラムを起動します。最初の一時停止 **auto-delete-subscriber.py** で実行します。以前は、これによりキューが削除され、2 番目のメッセージは何も表示されませんでした。この時点で、プロデューサーのサブスクリプションはキューを存続させ続けます。Enter を押して 2 番目のメッセージを **auto-delete-producer.py** 送信します。次に、**drain** またはのいずれかを使用してキューを確認し **auto-delete-subscriber.py**ます。この場合、キューが存在し、メッセージが予想通りに配信されていることを確認できます。

バグを報告します。

6.5.3. 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。

6.6. LAST VALUE(LV)キュー

6.6.1. 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。

6.6.2. 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。

6.6.3. 最後の値キューの例

この例では、Last Value Queue を作成および使用する方法を示しています。言語のバインディングとプログラミングの詳細は言語によって異なりますが、その原則は同じです。

通常の株価更新を提供するメッセージングキューを作成します。メッセージコンシューマーは現在の株価に関心を持ち、過去の情報が含まれるメッセージを希望したり、受け取りする必要がありません。このアプリケーションでは、最後の値キューが最適です。新たに受信されたメッセージは古いメッセージを更新し、置き換えることができます。

キュー「stock-ticker」を呼び出します。our shares-ticker キューは、「stock-symbol」を最後の値キューキーとして使用します。メッセージヘッダーのこのキーの値は、キューへの新しいメッセージ、またはキューにすでにあるメッセージへの更新としてメッセージを特定します。

まず、Qpid Messaging クライアントライブラリーをインポートします。

python

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

標準の AMQP ポート 5672 で実行しているブローカーへの Connection をローカルマシン上で作成します。

python

```
connection = Connection("localhost:5672")
connection.open()
```

次に、この接続を使用してセッションを作成します。

python

```
session = connection.session()
```

送信者を作成し、最後の値キューを同時に宣言します。「stock-ticker」というキューを作成し、最後の値キューキーに「stock-symbol」を使用します。このキューに送信されたメッセージは、ヘッダーに同じ「stock-symbol」を指定して、以前のメッセージへの更新として自身を特定します。

以下のステートメントは、1行のコードです。これは、表示中に改行する可能性がありますが、1行で入力する必要があります。

python

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

サイド注記： **qpid-config** コマンドラインツールを使用してキューを作成することもできます。

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

次に、一部のメッセージをキューに作成して送信します。ヘッダーで「stock-symbol」キーを使用して、メッセージに記述される発行先を特定します。最後の値キューはこのヘッダーキーを使用して、すでにキューにあるメッセージと適合します。

python

```
msg1 = Message("10")
msg1.properties = {'stock-symbol':'RHT'}

msg2 = Message("10")
```

```
msg2.properties = {'stock-symbol':'JAVA'}

msg3 = Message("10")
msg3.properties = {'stock-symbol':'MSFT'}

msg4 = Message("12")
msg4.properties = {'stock-symbol':'RHT'}
```

これらのメッセージを最後の値キューに送信すると、新しいコンシューマーにはキュー内に 3 つのメッセージが表示されるはずです。キューには、各チケットシンボルに 1 つずつ **msg4** 更新し **msg1**ます。標準の FIFO キューと最後の値キューの動作を比較するには、メッセージを control *-queue と呼ばれる制御* キューに同時に送信します。

python

```
controlSender = session.sender("control-queue;{create:always, node:{type:queue}}")
```

これでメッセージを 2 つのキューに送信します。

python

```
stockSender.send(msg1)
controlSender.send(msg1)

stockSender.send(msg2)
controlSender.send(msg2)

stockSender.send(msg3)
controlSender.send(msg3)

stockSender.send(msg4)
controlSender.send(msg4)
```

メッセージはキューにあります。これで、キューの内容を調べるために 2 つのレシーバーを作成します。

python

```
stockBrowser = session.receiver("stock-ticker; {mode:browse}")
controlBrowser = session.receiver("control-queue; {mode:browse}")
```

これらは閲覧用レシーバーであるため、メッセージを取得したりキューから削除したりしません。キューをクリアするには、以下のようにレシーバー宣言から browse プロパティーを削除 **session.receiver("stock-ticker")**し、demo を再度実行します。レシーバーを閲覧すると、キューをクリアせずに数回実行することで、最終値のキューの影響をより明確に確認できます。

レシーバーの事前フェッチ機能を使用してキュー上のメッセージを閲覧し、**available()** メソッドを使用してキューにあるメッセージ数をカウントできるようにします。これは、レシーバーのプリフェッチ **capacity** をデフォルトの 0 よりも高い値に設定します。

python

```
stockBrowser.capacity = 20
controlBrowser.capacity = 20
```

レシーバーのプリフェッチ容量が 20 に設定されている場合は、最大 20 個の利用可能なメッセージがキューから非同期的に取得されます。操作は非同期であるため、完了するまで待機する必要があります。事前にフェッチしたメッセージを調査する前に、アプリケーションを 10 秒間スリープ状態にします。

python

```
sleep 10
```

time ライブラリー **sleep** からインポートする必要があります。

python

```
from time import sleep
```

レシーバーの **available()** プロパティーを調べるために、これはキュー内のメッセージ数を表すことが確実に指定されていることに注意してください。非同期的に操作すると、ローカルで利用可能なメッセージの数が **available()** 報告されます。10 秒の遅延後には、キュー内のメッセージの合計数であることが合理的に考えられます。実際の非同期操作では、アプリケーションの実行をブロックしません。代わりに、以下のようなパターンを使用します。

python

```
while True:
  try:
    msg = stockBrowser.fetch(timeout = 10)
    print msg.properties["stock-symbol"] + ":" + msg.content
  except Empty:
    break
```

アプリケーションがスリープサイクルを終了すると、キュー内のメッセージ数を検査し、プリントアウトします。

python

```
print "Last Value Queue has " + str(stockBrowser.available()) + " messages"

print "\nLast Value Queue messages:"

for x in range(stockBrowser.available()):
  try:
    msg = stockBrowser.fetch(timeout = 1)
    print msg.properties["stock-symbol"] + ":" + msg.content
  except MessagingError, m:
    pass

print "Control Queue has " + str(controlBrowser.available()) + " messages"
```

```
print "\nControl Queue messages:"
for x in range(controlBrowser.available()):
  try:
    msg = controlBrowser.fetch(timeout = 1)
    print msg.properties["stock-symbol"] + ":" + msg.content
  except MessagingError, m:
    pass
```

最後に、セッションを確認し、接続を終了します。

python

```
session.acknowledge()
connection.close()
```

これでテストを実行する準備が整いました。以下は、プログラムの完全なリストです。

python

```
import sys
from qpid.messaging import *
from time import sleep

connection = Connection("localhost:5672")
try:
  connection.open()
  session = connection.session()

  stockSender = session.sender("stock-ticker;{create:always, node:{type:queue, x-declare:
{arguments:{'qpid.last_value_queue_key': 'stock-symbol'}}}}")
  controlSender = session.sender("control-queue;{create:always, node:{type:queue}}")

  stockBrowser = session.receiver("stock-ticker;{mode:browse}")
  controlBrowser = session.receiver("control-queue;{mode:browse}")
  controlBrowser = session.receiver("control-queue")

  msg1 = Message("10")
  msg1.properties = {'stock-symbol':'RHT'}

  msg2 = Message("10")
  msg2.properties = {'stock-symbol':'JAVA'}

  msg3 = Message("10")
  msg3.properties = {'stock-symbol':'MSFT'}

  msg4 = Message("12")
  msg4.properties = {'stock-symbol':'RHT'}

  stockSender.send(msg1)
  controlSender.send(msg1)

  stockSender.send(msg2)
  controlSender.send(msg2)
```

```
    stockSender.send(msg3)
    controlSender.send(msg3)

    stockSender.send(msg4)
    controlSender.send(msg4)

    stockBrowser.capacity = 20
    controlBrowser.capacity = 20

    sleep(10)

    print "\nLast Value Queue has " + str(stockBrowser.available()) + " messages"

    print "Last Value Queue messages:"

    for x in range(stockBrowser.available()):
      try:
        msg = stockBrowser.fetch(timeout = 1)
        print msg.properties["stock-symbol"] + ":" + msg.content
      except MessagingError, m:
        pass

    print "\nControl Queue has " + str(controlBrowser.available()) + " messages"

    print "Control Queue messages:"

      for x in range(controlBrowser.available()):
      try:
        msg = controlBrowser.fetch(timeout = 1)
        print msg.properties["stock-symbol"] + ":" + msg.content
      except MessagingError, m:
        pass

    session.acknowledge()

  except MessagingError,m:
    print m
  finally:
    connection.close()
```

バグを報告します。

6.6.4. 最後の値キューのコマンドラインの例

含まれるプログラムは **drain**、テスト目的でメッセージの送受信に使用 **spout** できます。2 つのユーティリティーのソースコードは、Python および C++ のクライアントライブラリーパッケージに含まれています。Python バージョンは、Python インタープリターを使用してコンパイル解除できます。

以下の **qpid-config** コマンドを実行して Last Value キューを作成します。

```
qpid-config add queue my-queue --argument qpid.last_value_queue_key=type
```

ヘッダーキー '**type**' は、キュー内のメッセージを照合するために使用されます。

次に、以下の **drain** コマンドを使用して 1 つ以上のブラウザーを起動します。

```
./drain -f -c 0 'my-queue; {mode: browse}'
```

これらのブラウザーは、キューに到達すると、リアルタイムですべてのメッセージが表示されます。

これで、**spout** を使用してキューにメッセージを送信し、キー '**type**' のヘッダー値を設定します。

```
./spout -P type=a my-queue a1
./spout -P type=a my-queue a2
./spout -P type=a my-queue a3
./spout -P type=b my-queue b1
./spout -P type=c my-queue c1
./spout -P type=c my-queue c2
./spout -P type=a my-queue a4
```

これらのメッセージが公開される前に起動されたブラウザーは、受信時にすべてのメッセージが出力されます。

次に、新しいブラウザーを起動します。

```
./drain -c 0 'my-queue; {mode: browse}'
```

このブラウザーには、固有の各 '**type**' 値に対する最後のメッセージのみが表示されます。

バグを報告します。

6.7. 優先順位キュー

6.7.1. 優先順位キュー

優先度キューは、優先度に基づいてメッセージを配信します。優先度が高いメッセージは、優先度の低いメッセージよりも前に送信されます。優先度レベルの合計は 10 個あります。

優先度キューは **qpid.priority** 属性で宣言されます。この属性は 1 から 10 の整数値で、キューの固有の優先度レベルの数を定義します。

たとえば、キューの **qpid.priority** 属性が 10 に設定されている場合、キューには 10 個の異なる優先度レベルがあります。この場合、優先度が 10 のメッセージは、優先度が 5 のメッセージの前に配信されます。これは、優先度が 1 のメッセージの前に送信されます。

キューの **qpid.priority** 属性が 2 に設定されている場合、キューに異なる優先度レベルが 2 つあります。この場合、メッセージの優先度 6-10 はキューの優先度レベル 1 で、メッセージの優先度 1-5 はキューの優先度 2 です。同じ優先度の帯域幅のメッセージは、その優先度と受信順序に基づいて配信されます。

バグを報告します。

6.7.2. 優先度キューの宣言

優先度キューを宣言するには、ノード宣言 **qpid.priorities** の **x-declare** 引数に値を指定します。例：

python

```
sender = session.sender('my-queue; {create: always, node:{x-declare:{arguments:
{qpid.priorities:10}}}}')
```

qpid-config以下を使用します。

```
qpid-config add queue 'my-queue; {create: always, node:{x-declare:{arguments:{qpid.priorities:10}}}}'
```

バグを報告します。

6.7.3. Priority Queues を使用する際の考慮事項

コンシューマーおよび優先度キューの閲覧

優先順位キューは、キューの通常の First-In-First-Out(FIFO)の順序ではなく、優先度順にメッセージを収集します。**コンシューマーを参照** するための配信順序は「未定義」です。書き込み時には、コンシューマーの閲覧は FIFO 順に優先度キューからメッセージを受信しますが、今後変更される可能性があるため、アプリケーションでこの動作に依存しないでください。

Certshare 機能

メッセージエンキュー速度が優先度キューのデキューレートを十分に超過する場合、優先度の低いメッセージがキューから削除されない可能性があります。この状況を回避するために 、 コンシューマーは各優先度レベルから指定されたメッセージのブロックを順番に取得することができます。

バグを報告します。

6.7.4. 優先順位キューのデモンストレーション

以下のプログラムは、優先度キューの使用および動作を示しています。

python

```
#!/usr/bin/python

import sys
from qpid.messaging import *

connection = Connection("localhost:5672")
connection.open()
try:
  ssn = connection.session()

  x = 0
  print "\n"
  while True:
    print "Create queue with 2 or 10 priority levels?"
    x = raw_input()
    if (x == "2") or (x == "10"):
      break

  tx = ssn.sender("nonpriority-demo-queue; {create: always, node: {type: 'queue'}}")
  print "Creating a priority queue with " + x + " priority levels:"
  address =  "priority-demo-queue; {create: always, "
  address = address + "node:{x-declare: {auto-delete:True, "
```

```
address = address + "arguments: {'qpid.priorities': "
address = address + x + "}}}}"
print address
txpriority = ssn.sender(address)

rx = ssn.receiver('nonpriority-demo-queue')
rxpriority = ssn.receiver("priority-demo-queue")
rxbrowse = ssn.receiver("priority-demo-queue; {mode: browse}")

print "\nPress Enter to continue\n"
x = raw_input()

print "First message sent:"
msg = Message("priority 1")
msg.priority = 1
tx.send(msg)
txpriority.send(msg)
print msg

print "Second message sent:"
msg = Message('priority 4')
msg.priority = 4
tx.send(msg)
txpriority.send(msg)
print msg

print "\nPress Enter to continue\n"
x = raw_input()
print "BROWSE PRIORITY QUEUE"
print "First browse in priority queue:"
print rxbrowse.fetch()

print "Second browse in priority queue:"
print rxbrowse.fetch()

print "\nPress Enter to continue\n"
x = raw_input()

print "ACQUIRE PRIORITY QUEUE"
print "First message in priority queue:"
print rxpriority.fetch()

print "Second message in priority queue:"
print rxpriority.fetch()

print "\nPress Enter to continue\n"
x = raw_input()

print "ACQUIRE NON-PRIORITY QUEUE"
print "First message in non-priority queue:"
print rx.fetch()

print "Second message in non-priority queue:"
print rx.fetch()
```

```
    ssn.acknowledge()
  finally:
    connection.close()
```

このプログラムを実行すると、優先度レベルは 2 または 10 の優先度キューを作成できます。次に、優先度 1 と 4 を使用して 2 つのメッセージをこのキューに送信します。次に、これにより優先順位キューから閲覧および取得する動作が実証され、これを非優先順位キューからの取得と対照的します。

以下は、プログラムの実行中に、10 個の優先度レベルの優先度キューが作成された場合の出力です。

```
Create queue with 2 or 10 priority levels?
10
Creating a priority queue with 10 priority levels:
priority-demo-queue; {create: always, node:{x-declare: {auto-delete:True, arguments: {'qpid.priorities':
10}}}}
```

キューは、異なる値でプログラムを複数回実行 **auto-delete: True** できるように宣言され **qpid.priorities**ます。送信側の作成時にキューがすでに存在する場合は、に指定された値 **qpid.priorities** は影響しません。この値は、キューの作成時にのみ有効です。

```
First message sent:
Message(priority=1, content='priority 1')
Second message sent:
Message(priority=4, content='priority 4')
```

優先度が 1（優先度が最も低い）と優先度 4（優先度が高い）の 2 つのメッセージが送信されます。

最初の調査は、閲覧側レシーバーです。優先度キューは、コンシューマーのみを取得してブラウザーには影響しません。したがって、メッセージの送信順序（FIFO *First In*、*First Out*） が表示されます。

```
BROWSE PRIORITY QUEUE
First browse in priority queue:
Message(priority=1, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
1')
Second browse in priority queue:
Message(priority=4, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
4')
```

ただし、優先度キューからメッセージを取得すると、そのメッセージは降順であることを確認できます。優先度 4 メッセージは、後で送信されても、優先度 1 メッセージの前に配信されます。

```
ACQUIRE PRIORITY QUEUE
First message in priority queue:
Message(priority=4, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
4')
Second message in priority queue:
Message(priority=1, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
1')
```

最後に、メッセージは優先順位以外のキューからキューに入れられ、ブローカーによって受信された順序で配信されます。

```
ACQUIRE NON-PRIORITY QUEUE
```

```
First message in non-priority queue:
Message(priority=1, properties={'x-amqp-0-10.routing-key': u'nonpriority-demo-queue'},
content='priority 1')
Second message in non-priority queue:
Message(priority=4, properties={'x-amqp-0-10.routing-key': u'nonpriority-demo-queue'},
content='priority 4')
```

デモが実行され、2 つのレベルのみが優先度キューが選択されている場合は、優先度キューが同じメッセージを配信順に配信します。

```
Create queue with 2 or 10 priority levels?
2
Creating a priority queue with 2 priority levels:
priority-demo-queue; {create: always, node:{x-declare: {auto-delete:True, arguments: {'qpid.priorities':
2}}}}

....

ACQUIRE PRIORITY QUEUE
First message in priority queue:
Message(priority=1, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
1')
Second message in priority queue:
Message(priority=4, properties={'x-amqp-0-10.routing-key': u'priority-demo-queue'}, content='priority
4')
```

キューに固有の優先度レベルが 2 つしかない場合、これらのレベルはメッセージの優先度が 1-5 と 6-10 になります。メッセージの優先度は両方とも 1 から 5 であるため、同じ優先度があり、ブローカーによって受信された順序に基づいて配信されます。

バグを報告します。

6.7.5. share Feature

優先度キューを使用する場合、メッセージプロデューサーとコンシューマーの間で速度の不一致が発生すると、キューに無期限に残される優先順位メッセージが少なくなります。すべての優先度のメッセージが処理されるようにするため、Cross *機能* を使用して各優先度レベルで事前定義されたメッセージ数を取得することができます。

の **x-qpid-fairshare** 引数は、共通の数のメッセージを、優先度別のレベルまたは priority レベルごとのカスタム数のメッセージ数に強制するために使用 **x-declare: argument** できます。

以下の例では、優先度が 10 のキューを作成し、順番に各優先度から 5 つのメッセージを取得します。

C++

```
Sender sender = session.createSender('my-queue; {create: always, node:{x-declare:{arguments:
{qpid.priorities:10, x-qpid-fairshare: 5}}}}')
```

以下の例では、優先度レベル 10 個で、priority-level ごとにカスタムのマーク共有量を持つキューを作成します。

C++

```
Sender sender = session.createSender('my-queue; {create: always, node:{x-declare:{arguments:
{qpid.priorities:10, x-qpid-fairshare-0: 3, x-qpid-fairshare-1: 5, x-qpid-fairshare-2: 3, x-qpid-
fairshare-3: 2, x-qpid-fairshare-4: 4, x-qpid-fairshare-5: 5, x-qpid-fairshare-6: 5, x-qpid-fairshare-7:
3, x-qpid-fairshare-8: 5, x-qpid-fairshare-9: 4, x-qpid-priorities: 10}}}}')
```

バグを報告します。

6.8. メッセージグループ

6.8.1. 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。

6.8.2. メッセージグループを有効にしてキューを作成する

メッセージグループを有効にしてキューを作成するには、キューの作成引数 **qpid.shared_msg_group** の値 **qpid.group_header_key** および値を指定します。

qpid.group_header_key は、メッセージと照合するために使用されるヘッダーキーです。ヘッダーのこのキーに同じ値を持つメッセージは同じグループに属します。

qpid.shared_msg_group をに設定する必要があり **1**ます。

以下の例では、ヘッダーフィールド「msgGroupID」を使用してメッセージをグループ化する自動削除キューを作成します。

python

```
groupedSender = session.sender("my-grouped-msg-queue; {create: always, node: {x-declare:
{auto-delete: True, arguments: {'qpid.group_header_key': 'msgGroupID',
'qpid.shared_msg_group': 1}}}}")
```

C++

```
Sender groupedSender = session.createSender("my-grouped-msg-queue; {create:always, node:
{x-declare: {auto-delete: True, arguments: {'qpid.group_header_key':'msgGroupID',
'qpid.shared_msg_group':1}}}}")
```

バグを報告します。

6.8.3. 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。

6.8.4. 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。

6.8.5. 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。

6.8.6. 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。

6.8.7. メッセージグループデモンストレーション

以下の Python プログラムは、メッセージグループの使用および動作を示しています。このプログラムを実行するには、コードをテキストファイルにコピーして貼り付け **message-groups.py**、そのまま保存してから、メッセージングブローカーが起動したマシンで Python を使用します。

プログラムは、メッセージングが有効または無効の自動削除キューを作成し、キューのグループヘッダーに一致するメッセージグループヘッダーを持つメッセージをキューに送信します。メッセージングを有効にすると、コンシューマーにブローカーによってメッセージグループの所有権が付与される方法と、キューで表示される内容や表示されない内容にどのように影響するかが実証されます。また、グループから取得したメッセージをすべて承認することで、コンシューマーがグループの所有権をリリースする方法と、フェッチされたメッセージの一部でグループの所有権がリリースされない方法を実証します。

プログラムは 2 つの異なる接続を使用して、2 つのコンシューマーをシミュレートします。コンシューマーは通常別のプロセスとして実行され、おそらく異なるマシンで実行されます。

python

```
import sys
from qpid.messaging import *

def sendmsg(group, num):
# send the message to the broker and add it to our in-memory representation of the broker queue
  global memoryqueue
  global tx

  msg = Message(group + num)
  msg.properties = {'ourGroupID': group}

  tx.send(msg)
  memoryqueue.append(group + num)

def pullmsg(consumer):
# fetch a message from the broker and print it to the console
  global counter
  global memoryqueue

  msg = consumers[consumer - 1].fetch(timeout = 1)

  print "\nQueued message: " + memoryqueue[counter]
  print "Consumer " + str(consumer) + " got: " + msg.content

  counter +=1
  return msg

# Two connections are used to simulate two distinct consumers
connection = Connection("localhost:5672")
connection2 = Connection("localhost:5672")
connection.open()
connection2.open()

try:
  session = connection.session()
  session2 = connection2.session()

  x = raw_input('Enable message grouping [Y/n]?')
```

```
  if x == 'N' or x == 'n':

    # Create the queue without message groups
    tx = session.sender("test-nogroup-queue; {create: always, node:{x-declare:{auto-
delete:True}}}")
    rx1 = session.receiver("test-nogroup-queue")
    rx2 = session2.receiver("test-nogroup-queue")

    print "\nMessage grouping is disabled"
    msggroup = False

  else:

    # Create the queue with message groups enabled
    tx = session.sender("test-group-queue; {create: always, node:{x-declare:{auto-delete: True,
arguments: {'qpid.group_header_key': 'ourGroupID', 'qpid.shared_msg_group' : 1}}}}")
    rx1 = session.receiver("test-group-queue")
    rx2 = session2.receiver("test-group-queue")

    print "\nMessage grouping is enabled"
    msggroup = True

# Put the receivers in an array so we can use a function to fetch messages
  consumers = []
  consumers.append(rx1)
  consumers.append(rx2)

  print "Sending interleaved messages from two different groups to the queue..."

# We create an in-memory picture of the queue, to see what order the messages are on the
broker
  memoryqueue = []

  sendmsg('A', '1')
  sendmsg('B', '1')
  sendmsg('B', '2')
  sendmsg('A', '2')
  sendmsg('B', '3')
  sendmsg('A', '3')

  counter = 0
  pullmsg(1)
  pullmsg(2)

  if msggroup:
    print "\nConsumer 1 now owns message group A. Consumer 2 now owns message group B."

  msgc1 = pullmsg(1)
  msgc2 = pullmsg(2)

  if msggroup:
    print "\nThe consumers will now acknowledge all the messages, or only the last one."
    resp = raw_input('Should they acknowlege all messages? [Y/n]')

    if resp == 'N' or resp == 'n':
      print "\nAcknowledging only part of the group. The consumers retain ownership of the group.
```

```
This is an anti-pattern! See the source code comments for details."

      session.acknowledge(msgc1)
      session2.acknowledge(msgc2)
      antipattern = True

      # Acknowledging only part of a group is an anti-pattern. Messages are grouped to ensure that
a single consumer can deal with the whole group. If this consumer now fails before completing the
rest of the group, the unacknowledged messages in the group will be released and redelivered by
the broker, but the acknowledged messages in the group are now missing in action!

    else:
      print "\nAcknowledging all fetched messages. The consumers will release ownership of the
groups."
      session.acknowledge()
      session2.acknowledge()
      antipattern = False

    print "\nPulling more messages from the queue:"

  pullmsg(1)
  pullmsg(2)
  if msggroup:
    if antipattern == False:
      print "\nConsumer 1 now owns message group B. Consumer 2 now owns message group A."
    print "\nSending some more messages to the queue..."

  sendmsg('B', '4')
  sendmsg('B', '5')
  sendmsg('A', '4')
  sendmsg('A', '5')

  pullmsg(1)
  pullmsg(2)
  pullmsg(1)
  pullmsg(2)

finally:
  connection.close()
  connection2.close()
```

プログラムの出力例

プログラムは、2 つの異なるグループ **A** と **B** - からキューにメッセージを送信します。メッセージグループが無効になっている場合の出力の例を以下に示します。

```
$ python message-groups.py
Enable message grouping [Y/n]?n

Message grouping is disabled
Sending interleaved messages from two different groups to the queue...

Queued message: A1
Consumer 1 got: A1
```

```
Queued message: B1
Consumer 2 got: B1

Queued message: B2
Consumer 1 got: B2

Queued message: A2
Consumer 2 got: A2

Queued message: B3
Consumer 1 got: B3

Queued message: A3
Consumer 2 got: A3

Queued message: B4
Consumer 1 got: B4

Queued message: B5
Consumer 2 got: B5

Queued message: A4
Consumer 1 got: A4

Queued message: A5
Consumer 2 got: A5
```

コンシューマーはキューからメッセージをラウンドロビン方式でプルし、キューにメッセージが送信される順序でメッセージが表示されます。

メッセージグループが有効になっているプログラムを実行すると、メッセージグループがキューでメッセージの表示方法に影響を与える方法が示されます。

```
$ python message-groups.py
Enable message grouping [Y/n]?y

Message grouping is enabled
Sending interleaved messages from two different groups to the queue...

Queued message: A1
Consumer 1 got: A1

Queued message: B1
Consumer 2 got: B1

Consumer 1 now owns message group A. Consumer 2 now owns message group B.

Queued message: B2
Consumer 1 got: A2

Queued message: A2
Consumer 2 got: B2
```

この段階では、取得したメッセージをすべて確認するか、一部のメッセージしか確認しないかを選択できます。これまでに取得したすべてのメッセージのグループの所有権が認識され、コンシューマーの次のメッセージがキューの次のメッセージになります。

```
The consumers will now acknowledge all the messages, or only the last one.
Should they acknowlege all messages? [Y/n]y

Acknowledging all fetched messages. The consumers will release ownership of the groups.

Pulling more messages from the queue:

Queued message: B3
Consumer 1 got: B3

Queued message: A3
Consumer 2 got: A3
```

その後、これらのメッセージのグループの所有権を取得します。

```
Consumer 1 now owns message group B. Consumer 2 now owns message group A.

Sending some more messages to the queue...

Queued message: B4
Consumer 1 got: B4

Queued message: B5
Consumer 2 got: A4

Queued message: A4
Consumer 1 got: B5

Queued message: A5
Consumer 2 got: A5
```

グループで取得したメッセージではなく、最後のメッセージのみの確認を選択すると、プログラムはアンチパターンであることを警告し、コンシューマーがグループの所有権を保持することを示します。

```
The consumers will now acknowledge all the messages, or only the last one.
Should they acknowlege all messages? [Y/n]n

Acknowledging only part of the group. The consumers retain ownership of the group. This is an anti-
pattern! See the source code comments for details.

Pulling more messages from the queue:

Queued message: B3
Consumer 1 got: A3

Queued message: A3
Consumer 2 got: B3

Sending some more messages to the queue...

Queued message: B4
Consumer 1 got: A4

Queued message: B5
Consumer 2 got: B4
```

```
Queued message: A4
Consumer 1 got: A5

Queued message: A5
Consumer 2 got: B5
```

バグを報告します。

第7章 非同期メッセージング

7.1. 非同期操作

非同期操作により、ブローカーとの通信の一部がバックグラウンドで実行され、プログラムの実行が継続されます。送受信操作は同期的に実行され、クライアントとブローカーとの間で通信が行われる間はブロックされます。

非同期送信により、サーバーから確認応答を待たずに実行を継続できます。非同期受信を使用すると、受信側はバックグラウンドでメッセージを取得できるため、コード内の受信側を使用してメッセージを取得する場合は、メッセージがすでにフェッチされ、ローカルで利用可能になります。

非同期操作はスループットを大幅に向上しますが、非同期操作の動作を理解し、コード内で注意して管理する必要があります。

バグを報告します。

7.2. 非同期送信

7.2.1. 同期および非同期送信

送信者が信頼できるリンク上で同期的に送信されると、送信側がブローカーから確認応答を受け取るまで、送信者のスレッドの実行はブロックされます。これはテストおよびトラブルシューティングに役立ちますが、メッセージごとにラウンドトリップを導入することで、システムのスループットが低下します。

C++ API を使用する場合、デフォルトですべての呼び出しが *非同期* になります。ただし、Python API を使用する場合、その逆は true です。デフォルトでは、送信者はメッセージを同期的に送信します。

メッセージを非同期に送信できるため、ネットワーク帯域幅の使用量とスループットを最大化できます。非同期的に呼び出されると、ブローカーから受信を待つことなく、送信呼び出しが即座に返されます。

たとえば、以下の **send** オブジェクト **send()** メソッドへの呼び出しは非同期です。ブローカーから受信を待つことなく、即座に返します。

python

```
sender.send(message, sync = False)
```

C++

```
sender.send(message, false)
```

これは、C++ API のデフォルト動作であることに注意してください。

バグを報告します。

7.2.2. Sender Capacity

sender **capacity** は、送信者が許可するサーバーから非同期送信待ちの確認応答数を制御する送信元オブジェクトのプロパティーです。これらの未承認のメッセージは、リンクが失敗した場合の再送信のためにメモリーでバッファーされるため、送信側のリプレイバッファーサイズとも呼ばれ *ます*。

デフォルトでは、送信側の容量はに設定されています。**UNLIMITED**つまり、送信側は無制限の非同期呼び出しを許可し、システムのメモリー制限によってのみ制限されるメッセージをバッファーします。

送信側が UNLIMITED 以外の数字に **capacity** 設定されていると、送信者は多数の非同期送信操作を同時に処理できないことのみが許可されます。

たとえば、送信者の数が 10 に設定 **capacity** されている場合、最大 10 個の非同期送信操作が、送信側に対して同時に確認応答を待機できます。10 番目の非同期送信操作が呼び出され、その 10 つ目の操作がブローカーによって承認される前に 11 つ目の操作が試行される場合、送信者はブローカーによって非同期送信操作のいずれかが承認されるまでブロックされます。

送信側がサーバーを大幅に超過している場合は、バインドされていない送信側の容量がリソースに影響を及ぼす可能性がある点に留意してください。容量に達すると、送信者が非同期から同期送信動作に切り替わり、メッセージがブロックされます。これを考慮して送信側容量を調整し、送信側の容量と可用性を確認して、ブロッキングに問題がある場合に、送信側の容量と可用性をチェックします。

バグを報告します。

7.2.3. 送信元容量の設定

Python では、送信側の容量を設定するには、送信側の **capacity** プロパティーに値を割り当てます。C++ では、送信側の容量は **setCapacity** メソッドを使用して設定されます。

python

```
sender.capacity = 20
```

C++

```
sender.setCapacity(20)
```

バグを報告します。

7.2.4. クエリー送信容量

非同期メッセージの送信を使用すると、非同期呼び出しの状態を確認するために 3 つの送信者プロパティーを利用できます。以下のとおりです。

Sender Capacity

保留中の確認応答に常時指定できる非同期に送信されるメッセージの最大数。デフォルトではこれはですが **UNLIMITED**、未設定の非同期呼び出しの数を制限するように変更できます。送信側が容量にあるときにさらに非同期呼び出しを試みると、送信された別のメッセージがブローカーによって承認されるまでブロックされます。

C++

```
sender.getCapacity()
```

python

```
sender.capacity
```

送信元の未設定

ブローカーから保留中の確認応答を送信する非同期の数。

C++

```
sender.getUnsettled()
```

python

```
sender.unsettled()
```

送信元が利用可能

現在送信側が許可できる追加の非同期呼び出しの数。この値はプロパティーとして利用できますが、**sender.capacity** - から計算することもでき **sender.unsettled**ます。

C++

```
sender.getAvailable()
```

python

```
sender.available()
```

バグを報告します。

7.2.5. ブロックされた非同期送信の回避

非同期送信呼び出しはメッセージを送信バッファーに配置し、即座に実行されます。ただし、送信バッファーが満杯になると、領域が利用可能になるまで呼び出しがブロックされます。

非同期送信呼び出しがフルバッファーでブロックされないようにする必要がある場合は、呼び出しを行う前にバッファーの状態をクエリーする必要があります。たとえば、C++ の場合は以下のようになります。

C++

```
if (sender.getAvailable() > 0)
    sender.send(message, false)
// else drop the message
```

python

```
if sender.available() > 0:
  sender.send(message, sync=False)
else:
  # drop the message
```

送信者の再生バッファーサイズを増やして、満杯になる可能性を低減することもできます。

C++

```
sender.setCapacity(SOME_LARGE_NUMBER)
```

python

```
sender.capacity = SOME_LARGE_NUMBER
```

バグを報告します。

7.2.6. 非同期メッセージ送信の例

以下のコードは、送信者のプロパティーを使用して非同期送信操作を管理する方法を示しています。送信側が容量にある場合の同期ブロックを回避するオプションです。

C++

```
sender.setCapacity(MY_CAPACITY);

// Later
bool resend = true;
while (resend)
{
  if (sender.getAvailable()>0)
  {
    sender.send(message,false);
    resend = false;
  }
  else
  {
    // May wish to do nothing here
    // or send to log file
    std::cout << "Warning: Capacity \ full. Retry" << std::endl;
   }
}
// Later
if (sender.getUnsettled())
{
    session.sync();
}
```

python

```
snd.capacity = MY_CAPACITY

# Later

resend = True
while (resend):
  if (snd.available()>0):
    snd.send(msg, sync = False)
```

```
    resend = False
  else:
    print "Warning: Capacity full"

# Later
    if (snd.unsettled()):
      ssn.sync()
```

バグを報告します。

7.2.7. 非同期の送受信とリンクの信頼性

sender.capacity は、非同期送信時に送信者が許可する未確認応答の数です。2 フェーズの送信/承認の動作は、信頼できるリンクの特性です（技術的には *at-least-once* の信頼性のあるリンクとして知られています）。送信側はメッセージの確認応答にサーバーが応答するまでメッセージをローカルに送信し、そのメッセージをローカルにバッファーします。この未承認の送信メッセージのバッファーは、リンクが破棄されて再度確立された場合に、*送信者がメッセージを再送信（送信元再生）*できるようにします。信頼できるリンクが破棄されて透過的に再確立されると、サーバーが承認されていないメッセージが送信されても、送信元再生バッファーから再送信されます。

信頼できるリンクは、明示的にリンク信頼性が指定されていない送信者を作成する際に使用されるデフォルトのリンクです。送信側の作成時に **unreliable** リンクを明示的に要求できます。例：

python

```
sender = session.sender("amq.topic;{link: {'reliability': 'unreliable'}}")
```

unreliable リンクを使用する場合、送信側の容量には意味がありません。信頼性のないリンクでは、サーバーはメッセージの受信を認識しません。すべてのメッセージは、送信後確認応答した方が適切とみなされます。これは、送信側 **unreliable** の意味です。リンクがドロップされた場合、送信者はブローカーにどのメッセージがロードされたかを確認する方法がありません。また、容量が使用されないため、信頼できないリンク非同期送信者はブロックされないことを意味します。

Sender.capacity アプリケーションの公開をデータ損失に制限するために使用されます。また、送信側が再生バッファーで消費可能なメモリー量を制限します。プロデューサーのスロットルにも使用できます。送信元再生バッファーに必要なローカルメモリーの影響なしに、信頼できないリンクと、プロデューサーのスロットリングは発生しない状態で、非同期送信と最大スループットへの非同期送信を使用できます。ただし、信頼性が低下し、データ損失の可能性が重要ではない状況で、このパターンを使用することに注意してください。

以下のプログラムは、信頼できるリンクと信頼できないリンク上での非同期送信の違いを示しています。

python

```
import sys
from qpid.messaging import *

connection = Connection("localhost:5672")

try:
  connection.open()
  session = connection.session()
```

```
  linktype=""
  while linktype != "R" and linktype !="U":
    response = raw_input("Use (R)eliable or (U)nreliable link [R/U]? ")
    linktype = response.upper()

  if linktype == "U":
    sender = session.sender("amq.topic;{link: {'reliability': 'unreliable'}}")
  else:
    sender = session.sender("amq.topic")

  message = Message("Hello World:")
  print sender.capacity
  sender.capacity = 5
  for x in range (1000):
    if sender.available() == 0:
      print "Sender is blocking..."
    sender.send("Hello World: " + str(x), sync=False)
    print str(x) +" : " + str(sender.unsettled()) + " : " + str(sender.available())

except MessagingError,m:
    print m
finally:
  connection.close()
```

プログラムは、送信元を使用して、1000 個のメッセージを非同期的に送信し、容量が 5 個のメッセージを承認していないメッセージを送信します。出力は以下の形式になります。

```
message number : unacknowledged messages : further async send capacity
```

信頼できるリンク上で実行すると、未承認のメッセージ数と、非同期送信者がブロックする状況など、残りの async 送信容量が表示されます。

```
Use (R)eliable or (U)nreliable link [R/U]? R
...
918 : 1 : 4
919 : 2 : 3
920 : 3 : 2
921 : 4 : 1
922 : 5 : 0
Sender is blocking...
```

の値を試すと **sender.capacity** （プログラムコードで 5 に設定）、送信側ブロックへの影響を確認できます。

信頼できないリンク上で実行すると、送信者のパフォーマンスに影響 **sender.capacity** がないことが分かります。ただし、信頼できないことに注意してください。

```
Use (R)eliable or (U)nreliable link [R/U]? U
...
984 : 0 : 5
985 : 0 : 5
986 : 0 : 5
```

```
987 : 0 : 5
988 : 0 : 5
989 : 0 : 5
```

バグを報告します。

7.3. 非同期の受信

7.3.1. 非同期メッセージ取得(Prefetch)

デフォルトでは、受信側は **fetch()** 呼び出しの応答として単一のメッセージを同期的に取得します。*メッセージのプレフェッチを行うレシーバーの容量はデフォルトで 0 です。*

レシーバーの容量が 0 を超える値に設定されていると、受信側はキューからその数のメッセージに非同期的に取得します。この非同期の取得は *prefetch* と呼ばれ、レシーバーの **capacity** プロパティーを設定して有効および制御されます。

メッセージの事前フェッチには、以下の 2 つの利点があります。

- 事前にフェッチされたメッセージは、アプリケーションによって要求された場合にローカルで利用できます。ただし、ブローカーからメッセージを取得する同期呼び出しのオーバーヘッドはありません。
- prefetching が有効になっているレシーバーには、事前フェッチされたメッセージが利用可能数を判断するために呼び出される **available()** メソッドがあります。

available() メソッドには、以下の 2 つの点に注意してください。

prefetching は非同期です。つまり、キューの状態の絶対インジケータ **available()** として呼び出しによって返される数字に依存することはできません。たとえば、受信側の **available()** キャパシティーを 0 以外の値に設定したら、利用できる値が 0 のメッセージを返す可能性があります。これは、キューに必ずしもメッセージがないことを意味しますが、事前にフェッチされたメッセージはまだローカルで利用可能ではないことを意味します。

また、プレフェッチが有効なレシーバーの **available** メソッドによって報告される最大値は、**capacity** 受信側になります。この **available()** メソッドは、キュー内のメッセージの数ではなく、利用可能な事前フェッチされたメッセージの数を報告します。利用可能なメッセージの数がレシーバーの容量よりも小さい場合は、プレフェッチの非同期性質について、上記のメッセージと共にキュー内のメッセージ数であることを推測することができます。

バグを報告します。

7.3.2. Enable Receiver Prefetch

レシーバーがメッセージを事前フェッチできるようにするには、その容量を 0 よりも大きい値に設定します。

たとえば、以下のコードはレシーバーを作成し、最大 100 個のメッセージの事前フェッチを可能にします。

python

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

```
connection = Connection("localhost:5672")
connection.open()
ssn = connection.session()

prefetchingReceiver = ssn.receiver("testqueue; {create:always}");
prefetchingReceiver.capacity = 100
```

バグを報告します。

7.3.3. 受信メッセージについて非同期的に確認する

信頼できるリンク（技術的に *at-least-once* reinfrablity を持つリンク）は、リンクの信頼性を指定せずにレシーバーの作成時に使用されるデフォルトのリンクです。信頼性のないリンクのメッセージ確認応答は、を参照してください 信頼性のないリンク上でメッセージを受信したことを承認する。信頼できるリンクで受信されたメッセージは、コンシューマーによって承認されるまで **acquired** ブローカーに設定されます。メッセージが **acquired** モードの場合、キューには表示されません。応答を承認せずにコンシューマーが切断される **acquired** と、ヘッダーを使用してメッセージがコンシューマーから再度利用可能になり **redelivered=true** ます。

キューからメッセージを削除するには、コンシューマーはメッセージの受信を確認する必要があります。

Python では、**session** オブジェクトの **acknowledge()** メソッドを呼び出してこれを行います。

python

```
session.acknowledge()
```

引数のない **acknowledge()** メソッドを呼び出すと、そのセッションを使用してフェッチされたすべてのメッセージとして承認されます。特定のメッセージを承認するには、メッセージを引数として渡します。例：

python

```
msg = rx.fetch(timeout = 1)
session.acknowledge(msg)
```

このメソッドはデフォルトで同期的に実行され、ブローカーが応答するまで待機してから返信します。また、**sync = False** パラメーターを指定して非同期的に呼び出すこともできます。

python

```
session.acknowledge(msg, sync = False)
```

信頼性のないリンク上でメッセージを受信したことを承認する

レシーバーに **unreliable** リンクが要求されると、メッセージの取得時に確認応答が暗黙的になります。つまり、ブローカーは受信側が取得するとすぐにメッセージをマークします。確認を必要としず、メッセージのリリースや拒否はできません。

バグを報告します。

7.3.4. 非同期受信およびリンクの信頼性

非同期受信(prefetch)と **unreliable** リンクの組み合わせは、損失の可能性があることに留意してください。**unreliable** リンク上では、（キューの閲覧ではなく）アプリケーションが消費される場合に、事前にフェッチされると直ちにキューからメッセージが削除されます。コンシューマーからの承認待ちは行われません。事前にフェッチされたメッセージをディスパッチする前にコンシューマーが失敗すると、ブローカーは再配信されません。

この組み合わせを使用する場合には、非同期の受信(prefetch)と **unreliable** リンクの組み合わせは、影響に注意してください。

バグを報告します。

第8章 信頼性とサービス品質

8.1. リンクの信頼性

8.1.1. 信頼できるリンク

キューへの接続時に確立されたリンクはデフォルトで *信頼性* があります。技術的には、これは *at-least-once* の信頼性です。

信頼できるリンクでのメッセージの受信

メッセージコンシューマー（ *競合メッセージコンシューマーとも呼ばれます*）は、キューからメッセージを削除し、他のコンシューマーが使用不可にするメッセージコンシューマーです。メッセージコンシューマーが信頼できるリンクでブローカーからメッセージを取得すると、メッセージはに設定され **acquired**ます。取得した状態では、メッセージは他のコンシューマーには表示されません。これはコンシューマーが取得したすべての目的と目的を行いますが、ブローカーは、コンシューマーがストリープを *認識* するまでコピーを維持します。この時点で、ブローカーはメッセージを確実に配信し、そのコピーを削除します。

信頼できるリンクは、複数の動作を有効にします。メッセージを承認せずにコンシューマーがサーバーへの接続を閉じると、ブローカーはコンシューマーが失敗したことを仮定します。この場合、取得したメッセージはヘッダーとともにキューに返され **redelivered: true**ます。

さらに、コンシューマーはメッセージを明示的に *リリース* することを選択することもできます。この場合、ブローカーが同じアクションを実行する場合や、コンシューマーがメッセージの *拒否* を選択することもできます。メッセージが拒否されると、このキューまたは交換用にメッセージが設定されている場合 **alternate exchange**、ブローカーはこのメッセージをにルーティングします。設定されていない場合 **alternate exchange** は、メッセージは破棄されます。

信頼できるリンクでのメッセージの送信

信頼できるリンク経由でメッセージがブローカーに送信されると、送信者はブローカーが受信を受け入れるまでローカルコピーを維持します。その時点で、送信者はローカルコピーを削除します。同期的に送信すると、交換が行われるまでアプリケーションがブロックされます。非同期的にこれらの未承認の送信メッセージを送信するときは、*送信元再生バッファー* に保存されます。

信頼できるリンクがすぐに破棄され、再度確立されると、送信者はそのバッファーから未承認のメッセージを再送信し、データが失われないようにします。これにより、メッセージが複数回送信される可能性があり、**最低**でも -once という用語が送信されます。

信頼できるリンクの指定

キューへのすべてのリンクは、デフォルトで信頼性があります。キューへの接続時に信頼できるリンクを明示的に要求する必要はありません。

交換に接続する場合、リンクはデフォルトで信頼性がありません。交換への信頼できるリンクを指定するには、アドレス **link: {'reliability': 'at-least-once'}** に含めます。例：

```
sender = session.sender("amq.topic;{link: {'reliability': 'at-least-once'}}")
```

この場合、送信者は信頼できるリンク動作に従い、ブローカーによって確認されるまでメッセージをローカルでバッファーします。

バグを報告します。

8.1.2. 信頼性のないリンク

交換への接続時に確立されたリンクはデフォルトで *信頼* できません。また、キューへの接続を確立する際に、アプリケーションは **unreliable** リンクを明示的に要求できます。

信頼性のないリンクは、データを迅速かつ緩やかに送信します。サーバーまたはローカルクライアントのどちらにもバッファーがなく、接続が失われた状態から保護されます。クライアントが **unreliable** リンク経由でキューからメッセージを取得すると、ブローカーは、メッセージが受信され、正常に対処されたことをコンシューマーが承認するのを待たずに即座にこれを削除します。

シナリオによっては、**unreliable** リンクの使用時のスループットの増加が分かりますが、これは確実にとは限りません。信頼性のないリンクに対する最も明確な使用は、大量のデータを高速に送信し、データ損失が問題ではないことです。

ほとんどのアプリケーションは、信頼できるリンクで提供される保証を利用し、すべてのリンクのデフォルトとなります。

unreliable リンクのリクエスト

unreliable リンクを要求するには、受信側または送信者の **link: {'reliability': 'unreliable'}** アドレスを指定します。例：

python

```
sender = session.sender("amq.topic;{link: {'reliability': 'unreliable'}}")
```

バグを報告します。

8.2. キューのサイズ

8.2.1. 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」

バグを報告します。

8.2.2. キューのしきい値アラート

キューしきい値アラートは、キャパシティー制限セット（ **qpid.max_size** または **qpid.max_count**）が上限の 20% に達すると、ブローカーによって発行されます。図 50% は、broker オプションを使用してサーバー全体で設定でき **--default-event-threshold-ratio**ます。これをゼロに設定すると、デフォルトですべてのキューでアラートが無効になります。さらに、キューの作成 **qpid.alert_size** 時 **qpid.alert_count** および作成時に、デフォルトのアラートしきい値を上書きできます。

Alerts 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アラートメッセージにサブスクライブでき **qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdExceeded.#**ます。アラートはマップメッセージとして送信されます。

以下のコードは、アラートメッセージをサブスクライブして消費する方法を示しています。

python

```
conn = Connection.establish("localhost:5672")
session = conn.session()
rcv =
session.receiver("qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdExceeded.#")
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()
```

アラートリープ(Repeat Gap)

アラートメッセージがあふれるのを防ぐために、アラートメッセージには 60 秒の差があります。これにより、で異なる値を秒単位で指定 **qpid.alert_repeat_gap** すると、キューごとに上書きできます。

後方互換性エイリアス

以下のエイリアスは、以前のクライアントとの互換性を維持するために維持されます。

- **x-qpid-maximum-message-count** 以下と同等です。 **qpid.alert_count**
- **x-qpid-maximum-message-size** 以下と同等です。 **qpid.alert_size**
- **x-qpid-minimum-alert-repeat-gap** 以下と同等です。 **qpid.alert_repeat_gap**

バグを報告します。

8.3. プロデューサーフロー制御

8.3.1. 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。

8.3.2. 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。

8.3.3. 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。

8.3.4. 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。

8.3.5. 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。

8.4. クレジットベースのフロー制御

8.4.1. クレジットカードを使用したフロー制御

サブスクライバーは、特定の数のメッセージに対してクレジットをブローカーに割り当て、メッセージコンテンツの合計サイズを割り当てることで、サブスクライブされたキューからメッセージのフローを制御できます。ブローカーはメッセージを配信する際に、メッセージのクレジットを 1 つで減らし、メッセージの内容のサイズによってクレジットサイズをデクリメントすることで、このクレジットを消費します。ブローカーは、十分なクレジットがないサブスクリプションにメッセージを送信できません。

バグを報告します。

8.4.2. クレジット割り当てモード

AMQP 仕様で定義されたクレジット割り当てには、以下の 2 つのモードがあります。

- ブローカーがメッセージを送信する前に、*クレジットモード*はサブスクライバーによって明示的に再発行される必要があります。
- *ウィンドウモード*では、受信したメッセージに対して、クレジットが自動的に再発行されます。このモードでは、クライアントは転送の完了をブローカーに通知してメッセージが受信されたことを示します。完了は基本的には承認の形ですが、所有権の譲渡の確認である受け入れと混同しないようにしてください。

どちらのモードでも、メッセージ数および合計コンテンツサイズに無制限のクレジットを割り当てることができます。

バグを報告します。

8.5. 永続キュー

8.5.1. 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。

8.5.2. 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。

8.5.3. アプリケーション内で永続キューを作成する

以下のサンプルコードは、「important-messages」という永続キューを作成します。

C++

```
Sender sender = session.createSender("important-messages; {create:always, node:{durable: True}")
```

python

```
newqueue = session.sender("important-messages; {create:always, node:{durable: True}")
```

キューが宣言されても **durable 、 auto-delete**自動削除される **まで** 永続性があることに注意してください。これが必要な動作である場合は、注意して検討してください。

バグを報告します。

8.5.4. 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8.1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。

8.5.5. 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。

8.5.6. ジャーナルの説明

Red Hat Enterprise Messaging では、永続性に使用するファイルとキャッシュのサイズと数を設定できます。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。

各ジャーナルは、ディスクのキューとして実装され、読み取りキャッシュと書き込みキャッシュがメモリーに含まれています。ディスクでは、各キューはファイルのセットで構成されます。キャッシュはページ指向です。永続メッセージが永続キューに書き込まれると、関連するイベントは、ページが満杯またはタイムアウトになるまで書き込みキャッシュに累積され、ページは AIO を使用して永続キューに書き込まれます。書き込みキャッシュ内のメッセージはパブリッシャーに対して承認されていないため、ジャーナルに書き込まれるまでコンシューマーが読み取ることができません。ページサイズはパフォーマンスに影響します。ページサイズが小さくなると、レイテンシーが縮小され、ページサイズが大きくなると書き込み操作の数が減少し、スループットが向上します。

ジャーナルファイルは、関連付けられたキューが最初に宣言されたときに準備およびフォーマットされます。これにより、最初のパスで AIO を使用したスループットが 2 倍になり、必要な領域の割り当てが保証されます。ただし、永続キューが宣言されると、認識可能な遅延が発生する可能性があります。ファイルサイズが増大すると、遅延が大きくなります。

バグを報告します。

8.5.7. アプリケーションでのメッセージジャーナルの設定

キューのメッセージジャーナルのファイル数およびファイルサイズを設定するには、キューの作成に使用されるアドレスの **qpid.file_size** **qpid.file_count** **x-declare** 引数を指定し、指定します。

python

```
tx = ssn.sender("my-queue;{create: always, node: {durable: True, x-declare: {arguments:
{'qpid.file_size': 20, 'qpid.file_count': 12}}}}")
```

バグを報告します。

8.6. トランザクション

8.6.1. トランザクション

トランザクションセッションはメッセージトランザクションをサポートします。送信は成功または失敗しなければならないメッセージのグループです。トランザクションセッションで送信されたメッセージは、コミットのターゲットアドレスでのみ利用可能になります。同様に、受信、受信、確認応答されたメッセージは、コミット時にソースとしてのみ破棄されます。

トランザクション機能には信頼できるリンクが必要なことに注意してください。

バグを報告します。

8.6.2. トランザクションの例

以下のコードはトランザクションセッションを示しています。

.NET/C#

```
Connection connection = new Connection(broker);
Session session =  connection.createTransactionalSession();
...
if (smellsOk())
```

```
  session.Commit();
else
  session.Rollback();
```

C++

```
Connection connection(broker);
Session session =  connection.createTransactionalSession();
...
if (smellsOk())
  session.commit();
else
  session.rollback();
```

バグを報告します。

第9章 QPID 管理フレームワーク(QMF)

9.1. QMF - QPID 管理フレームワーク

Qpid 管理フレームワーク を使用すると、コマンドメッセージを使用してブローカーを管理できます。コマンドメッセージは、ルーティングキーまたはサブジェクトを使用して、交換を行うアドレスに送信されるマップメッセージ **qmf.default.direct/broker qmf.default.direct** です **broker**。メッセージには、送信者が応答を受信できる **reply-to** アドレスが含まれるはずです。

バグを報告します。

9.2. QMF バージョン

Red Hat Enterprise Messaging は、Qpid Management Framework バージョン 2 をサポートします。

QMFv2 は QMFv1 よりも多くの利点があります。これには、クラスターのノード間やフェデレーションされたリンク間で QMF メッセージを送信できます。

QMFv2 の詳細は Apache Qpid QMFv2 プロジェクトページを参照してください。

QMFv1 は、Red Hat Enterprise Messaging でサポートされなくなりました。

バグを報告します。

9.3. アプリケーションからの交換の作成

QMF メッセージを使用すると、アプリケーションから交換を作成できます。以下の QMF メッセージにより、Fluentout の交換が呼び出されます。 **test-fanout**

```
Message(subject='broker', reply_to='qmf.default.topic/direct.6da5bfc3-44fb-4441-b834-6c5897b9606a;{node:{type:topic}, link:{x-declare:{auto-delete:True,exclusive:True}}}', correlation_id='1', properties={'qmf.opcode': '_method_request', 'x-amqp-0-10.app-id': 'qmf2', 'method': 'request'}, content={'_object_id': {'_object_name': 'org.apache.qpid.broker:broker:amqp-broker'}, '_method_name': 'create', '_arguments': {'strict': True, 'type': 'exchange', 'name': u'test-fanout', 'properties': {'exchange-type': u'fanout'}}})
```

バグを報告します。

9.4. QMF によるブローカー交換およびキュー設定

qmf Command メッセージは、交換とキューの作成と設定に使用できます。**qpid-config** コマンドラインユーティリティーは QMF メッセージを使用して、その管理タスクの多くを実行します。

バグを報告します。

9.5. コマンドメッセージ

qmf コマンドメッセージは、ブローカーの QMF アドレスに送信される特別にフォーマットされたマップメッセージです **qmf.default.direct/broker**。

関連項目

- 13章 *マップおよびリスト*

バグを報告します。

9.6. QMF コマンドメッセージ構造

qmf コマンドメッセージの内容

qmf コマンドメッセージはマップメッセージです。QMF コマンドメッセージにはキーが含ま **_object_id** **_method_name** れ **_arguments**ます。

キー **_object_id** は必須です。この値は、コマンドのターゲットを特定するネストされたマップです。ブローカーとそのリソースを管理する QMF コマンドでは、**_object_id** マップには値が含ま **_object_name** れる単一の値が含まれ **org.apache.qpid.broker:broker:amqp-broker**ます。**_object_name** 値は '**package:class:id**' の構文です。必要な値は、を使用してスキーマから取得でき **qpid-tool**ます。

キーにはコマンド名を値として指定し、キーにはコマンド引数のネストされたマップ **_method_name** が **_arguments** 含まれます。

qmf コマンドメッセージプロパティー

2 つのメッセージプロパティー **x-amqp-0-10.app-id** とを設定 **qmf.opcode** する必要があります。プロパティーには常に値が含まれ、に値が **qmf.opcode** 含ま **qmf2** れている **x-amqp-0-10.app-id** はずです **_method_request**。

Qmf コマンド応答

サーバーからレスポンスを受け取るには、QMF コマンドメッセージの **reply-to** アドレスを、メッセージを受信できるアドレスに設定します。コマンドメッセージがブローカーの QMF アドレスに送信されると、応答は指定された **reply-to** アドレスから送信されます。 amqp0-10 使用 **qmf2**時 に応答メッセージに **x-amqp-0-10.app-id** プロパティーが設定されています。

コマンドメッセージが想定どおりに処理された場合、応答 **qmf.opcode** メッセージプロパティーはに設定され **_method_response**ます。エラーが発生した場合、**qmf.opcode** プロパティーに値が含まれ **_exception**ます。

応答メッセージの内容はマップです。有効な応答の場合、戻り値はキーに対してネストされたマップとして示され **_arguments**ます。例外の場合には、例外の詳細はキーに対するネストされたマップ内にあり **_values**ます。

バグを報告します。

9.7. コマンドの作成

QMF **create** コマンドは 5 つの引数を取ります。

type

作成するオブジェクトのタイプ。これはキュー、交換、またはバインディングになります。

Name

作成されるオブジェクトの名前。キューや交換の **name** 引数は単一の値です。たとえば、という名前のキューは name 引数をその値の文字列に **my-queue** 設定します。バインディングの名前では、パターンの *exchange/queue/key* を使用します。たとえば、 : は **amq.topic/my-queue/my-key** バインディングキー **my-queue** との交換 **amq.topic** と交換を指定します **my-key**。

properties

作成するオブジェクトの特定のプロパティー、値はネストされたマップです。

Strict

厳密な引数は、現在無視されるブール値を取ります。この値は、認識されないプロパティーが指定されている場合にコマンドが失敗するかどうかを示します。

auto_delete_timeout

オプション。最初に自動削除キューを宣言する際に指定する場合は、削除が実行される遅延を秒単位で指定します。注記： 削除の対象となり、遅延が期限切れになる前にキューが再宣言される場合、キューは削除されません。

以下のコード例では QMF を使用してという名前のキューを作成し **my-queue**ます。この例で **my-queue** は、10 秒後に自動削除するように設定されています。

python

```
conn = Connection(opts.broker)
try:
  conn.open()
  ssn = conn.session()
  snd = ssn.sender("qmf.default.direct/broker")
  reply_to = "reply-queue; {create:always, node:{x-declare:{auto-delete:true}}}"
  rcv = ssn.receiver(reply_to)

  content = {
            "_object_id": {"_object_name": "org.apache.qpid.broker:broker:amqp-broker"},
            "_method_name": "create",
            "_arguments": {"type":"queue", "name":"my-queue", "properties":{"auto-delete":True,
"qpid.auto_delete_timeout":10}}
          }
  request = Message(reply_to=reply_to, content=content)
  request.properties["x-amqp-0-10.app-id"] = "qmf2"
  request.properties["qmf.opcode"] = "_method_request"
  snd.send(request)

  try:
    response = rcv.fetch(timeout=opts.timeout)
    if response.properties['x-amqp-0-10.app-id'] == 'qmf2':
      if response.properties['qmf.opcode'] == '_method_response':
        return response.content['_arguments']
      elif response.properties['qmf.opcode'] == '_exception':
        raise Exception("Error: %s" % response.content['_values'])
      else: raise Exception("Invalid response received, unexpected opcode: %s" % m)
    else: raise Exception("Invalid response received, not a qmfv2 method: %s" % m)
  except Empty:
    print "No response received!"
  except Exception, e:
    print e
except ReceiverError, e:
  print e
except KeyboardInterrupt:
```

```
    pass

conn.close()
```

バグを報告します。

9.8. コマンドの削除

QMF **delete** コマンドは、3 つの引数を取ります。

type
削除するオブジェクトのタイプ。これはキュー、交換、またはバインディングになります。

Name
削除するオブジェクトの名前。キューや交換の **name** 引数は単一の値です **my-queue**。バインディングの名前では、パターンの *exchange/queue/key* を使用します。たとえば、 : は **amq.topic/my-queue/my-key** バインディングキー **my-queue** との交換 **amq.topic** と交換を指定します **my-key**。

オプション
キーを持つネストされたマップ **options**。これは、現在使用されません。

バグを報告します。

9.9. コマンドの一覧表示

以下の QMF メッセージがブローカーから交換のリストを要求する例です。

python

```
Message(subject='broker', reply_to='qmf.default.topic/direct.8b59a7ae-93f1-4450-9e43-1b0665bf622b;{node:{type:topic}, link:{x-declare:{auto-delete:True,exclusive:True}}}', correlation_id='1', properties={'qmf.opcode': '_query_request', 'x-amqp-0-10.app-id': 'qmf2', 'method': 'request'}, content={'_what': 'OBJECT', '_schema_id': {'_class_name': 'exchange'}})
```

以下の QMF メッセージがサーバーからキューの一覧を要求する例です。

python

```
Message(subject='broker', reply_to='qmf.default.topic/direct.7f703720-c815-4c79-986c-354b3963bc76;{node:{type:topic}, link:{x-declare:{auto-delete:True,exclusive:True}}}', correlation_id='1', properties={'qmf.opcode': '_query_request', 'x-amqp-0-10.app-id': 'qmf2', 'method': 'request'}, content={'_what': 'OBJECT', '_schema_id': {'_class_name': 'queue'}})
```

バグを報告します。

9.10. QMF を使用したキューおよび交換の作成

以下の QMF メッセージは、という名前の新しいキューを作成し **test**ます。

python

```
Message(subject='broker', reply_to='qmf.default.topic/direct.8702f596-b112-427d-b93e-7e0ae28f2ae8;{node:{type:topic}, link:{x-declare:{auto-delete:True,exclusive:True}}}', correlation_id='1', properties={'qmf.opcode': '_method_request', 'x-amqp-0-10.app-id': 'qmf2', 'method': 'request'}, content={'_object_id': {'_object_name': 'org.apache.qpid.broker:broker:amqp-broker'}, '_method_name': 'create', '_arguments': {'strict': True, 'type': 'queue', 'name': u'test', 'properties': {}}})
```

以下の QMF メッセージは、呼び出された新しいジャンクアウト交換を作成し **test-fanout**ます。

python

```
Message(subject='broker', reply_to='qmf.default.topic/direct.81915d0a-d2e1-4cf9-9369-921bac725aab;{node:{type:topic}, link:{x-declare:{auto-delete:True,exclusive:True}}}', correlation_id='1', properties={'qmf.opcode': '_method_request', 'x-amqp-0-10.app-id': 'qmf2', 'method': 'request'}, content={'_object_id': {'_object_name': 'org.apache.qpid.broker:broker:amqp-broker'}, '_method_name': 'create', '_arguments': {'strict': True, 'type': 'exchange', 'name': u'test-fanout', 'properties': {'exchange-type': u'fanout'}}})
```

バグを報告します。

9.11. QMF イベント

Qmf イベントは、ブローカーイベントの通知を提供するために QMF トピックに送信されるメッセージです。キューしきい値アラートは QMF イベントとして実装されます。

QMF トピックはです。ここで **qmf.default.topic/agent.ind.event.org_apache_qpid_broker.*$QMF_Event*.#**、*$QMF_Event* は以下の表で提供された QMF イベントの 1 つです。

表9.1 qmf イベント

qmf イベント	重大度	引数
clientConnect	inform	rhost、user、properties
clientConnectFail	WARN	rhost、user、reason、properties
clientDisconnect	inform	rhost、user、properties
brokerLinkUp	inform	rhost
brokerLinkDown	WARN	rhost
queueDeclare	inform	rhost, user, qName, durable, excl, autoDel, altEx, args, disp
queueDelete	inform	rhost、user、qName

qmf イベント	重大度	引数
exchangeDeclare	inform	rhost, user, exName, exType, altEx, durable, autoDel, args, disp
exchangeDelete	inform	rhost、user、exName
bind	inform	rhost, user, exName, qName, key, args
unbind	inform	rhost、user、exName、qName、key
subscribe	inform	rhost、user、qName、dest、excl、args
サブスクライブ解除	inform	rhost、user、dest
queueThresholdExceeded	WARN	qname、msgDepth、byteDepth

関連項目

- 「キューのしきい値アラート」

バグを報告します。

9.12. QMF クライアント接続イベント

クライアントがブローカーに接続または切断すると、QMF イベントメッセージが生成され、QMF トピックに送信されます。

これらのイベントの QMF トピックは以下のとおりです。

表9.2 qmf クライアント接続イベントに関するトピック

qmf キュー	目的
qmf.default.topic/agent.ind.event.org_apache_qpid_broker.clientConnect.#	クライアント接続
qmf.default.topic/agent.ind.event.org_apache_qpid_broker.clientConnectFail.#	接続試行の失敗
qmf.default.topic/agent.ind.event.org_apache_qpid_broker.clientDisconnect.#	クライアントの接続解除

QMF クライアント接続および Disconnection イベントメッセージの追加プロパティーは、特定のクライアントに接続と切断に一致して、監査とトラブルシューティングを有効にします。

- **client_ppid**[1]
- **client_pid**

- **client_process**

以下は、クライアント接続情報の集約である QMF クライアント接続イベントメッセージの例になります。

```
Fetched Message(
  properties={
    u'qmf.agent': u'apache.org:qpidd:a2ff61bc-19b2-4078-8a7e-9c007151c79c',
    'x-amqp-0-10.routing-key':
u'agent.ind.event.org_apache_qpid_broker.clientConnect.info.apache_org.qpidd.a2ff61bc-19b2-
4078-8a7e-9c007151c79c',
    'x-amqp-0-10.app-id': 'qmf2',
     u'qmf.content': u'_event',
     u'qmf.opcode': u'_data_indication',
     u'method': u'indication'},
     content=[{
       u'_schema_id': {
         u'_package_name': 'org.apache.qpid.broker',
         u'_class_name': 'clientConnect',
         u'_type': '_event',
         u'_hash': UUID('476930ed-01dd-9629-7f84-f42b4b0bc410')},
       u'_timestamp': 1347032560197086881,
       u'_values': {
         u'user': 'anonymous',
         u'properties': {
           u'qpid.session_flow': 1,
           u'qpid.client_ppid': 26139,
           u'qpid.client_pid': 26876,
           u'qpid.client_process': u'spout'},
        u'rhost': '127.0.0.1:5672-127.0.0.1:43276'},
     u'_severity': 6}])

Fri Sep  7 15:42:40 2012 org.apache.qpid.broker:clientConnect user=anonymous properties={
  u'qpid.session_flow': 1,
  u'qpid.client_ppid': 26139,
  u'qpid.client_pid': 26876,
  u'qpid.client_process': u'spout'}
rhost=127.0.0.1:5672-127.0.0.1:43276
```

バグを報告します。

9.13. ACL ルックアップクエリーメソッド

ACL 承認インターフェースをクエリーするには、qmf メソッドを使用できます。

Broker は、クエリーを行う ACL ファイルで起動する必要があります。その ACL ファイルには、ルックアップ操作を許可するのに十分なパーミッションが含まれる必要があります。

```
# Catch 22: allow anonymous to access the lookup debug functions
acl allow-log anonymous create  queue
acl allow-log anonymous all     exchange name=qmf.*
acl allow-log anonymous all     exchange name=amq.direct
acl allow-log anonymous all     exchange name=qpid.management
acl allow-log anonymous access  method   name=Lookup*
```

ACL 承認インターフェースをクエリーする QMF メソッドは **Lookup** およびです **LookupPublish**。

Lookup メソッドは、アクション、オブジェクト、およびプロパティーのセットに対する一般的なクエリーです。この **LookupPublish** 方法は、メッセージごとに最適化された fastpath クエリーです。

どちらの方法でも **allow**、、、**deny allow-log**、またはのいずれかになります **deny-log**。

method: Lookup

表9.3 method: Lookup

引数	type	方向
userId	long-string	I
action	long-string	I
オブジェクト	long-string	I
objectName	long-string	I
propertyMap	field-table	I
結果	long-string	O

メソッド： LookupPublish

表9.4 メソッド： LookupPublish

引数	type	方向
userId	long-string	I
exchangeName	long-string	I
routingkey	long-string	I
結果	long-string	O

管理プロパティーと統計

有効にするコマンドライン設定を反映するために、以下のプロパティーと統計が追加され、Acl クォータの拒否アクティビティーが反映されます。

表9.5 Broker Management Quota プロパティー

要素	type	access	description
maxConnections	uint16	ReadOnly	許可される最大接続

表9.6 ACL 管理インターフェース

要素	type	access	description
maxConnectionsPerIp	uint16	ReadOnly	許可される最大接続
maxConnectionsPerUser	uint16	ReadOnly	許可される最大接続
maxQueuesPerUser	uint16	ReadOnly	許可される最大キュー
connectionDenyCount	uint64		拒否された接続数
queueQuotaDenyCount	uint64		拒否されたキューの作成数

例

手順9.1 ACL ルックアップの例

実用的な例を確認するには、以下の手順に従います。

1. 以下で **acl-test-01-rules.acl** 再現した ACL ファイルのサンプルを使用して、ブローカーを起動し **QPID_LOG_ENABLE=debug+:acl**ます。
2. Python スクリプトを実行し **acl-test-01.py**ます。
3. Python プログラムの出力とブローカーログを確認します。

ACL ファイル **acl-test-01-rules.acl**

```
# acl-test-rules-00.acl
# 27-march-2012

group admins moe@COMPANY.COM \
          larry@COMPANY.COM \
     curly@COMPANY.COM \
     shemp@COMPANY.COM

group auditors aaudit@COMPANY.COM baudit@COMPANY.COM caudit@COMPANY.COM \
            daudit@COMPANY.COM eaduit@COMPANY.COM eaudit@COMPANY.COM

group tatunghosts tatung01@COMPANY.COM \
                tatung02/x86.build.company.com@COMPANY.COM \
                tatung03/x86.build.company.com@COMPANY.COM \
                tatung04/x86.build.company.com@COMPANY.COM \
                HTTP/tatung-test1.eng.company.com@COMPANY.COM

group publishusers publish@COMPANY.COM x-pubs@COMPANY.COM

# Admins: This should be the *only* group which ever gets "all" access
# to anything. Everything/everyone else must not be as permissive
acl allow-log admins all all

# Catch 22: allow anonymous to access the lookup debug functions
```

```
acl allow-log anonymous create  queue
acl allow-log anonymous all     exchange name=qmf.*
acl allow-log anonymous all     exchange name=amq.direct
acl allow-log anonymous all     exchange name=qpid.management
acl allow-log anonymous access  method   name=Lookup*

acl allow all publish exchange name=''

# Auditors
acl allow-log auditors all exchange name=company.topic routingkey=private.audit.*

# Tatung
acl allow-log tatunghosts  publish exchange name=company.topic  routingkey=tatung.*
acl allow-log tatunghosts  publish exchange name=company.direct routingkey=tatung-service-queue

# Publish
acl allow-log publishusers create queue
acl allow-log publishusers publish exchange name=qpid.management routingkey=broker
acl allow-log publishusers publish exchange name=qmf.default.topic routingkey=*
acl allow-log publishusers publish exchange name=qmf.default.direct routingkey=*

# Consumers - everyone
acl allow-log all bind exchange name=company.topic  routingkey=tatung.*
acl allow-log all bind exchange name=company.direct routingkey=tatung-service-queue

acl allow-log all consume queue

acl allow-log all access exchange
acl allow-log all access queue

acl allow-log all create queue name=tmp.* durable=false autodelete=true exclusive=true
policytype=ring

# All else is denied
acl deny-log all all
```

Python スクリプト **acl-test-01.py**

```
# acl-test-00.py
# test driver for QPID-3918 lookup hooks.
#
# The broker is to use acl-test-00-rules.acl.
#
import sys
import qpid
import qmf

totalLookups = 0
failLookups  = 0
exitOnError  = True

#
# Run a type 1 lookup
# This is the general lookup
#
def Lookup(acl, userName, action, aclObj, aclObjName, propMap, expectedResult = ''):
```

```
    global totalLookups
    global failLookups
    totalLookups += 1
    result = acl.Lookup(userName, action, aclObj, aclObjName, propMap)
    suffix = ''
    if (expectedResult != ''):
        if (result.result != expectedResult):
            failLookups += 1
            suffix = ', [ERROR: Expected ' + expectedResult + ']'
            if (result.result is None):
                suffix = suffix + ', [' + result.text + ']'
    print 'Lookup : [name:', userName, ", action: ", action, ", object: ", aclObj, \
        ", objName: '", aclObjName, "', properties: ", propMap, \
        "], [Result: ", result.result, "]", suffix
    if (exitOnError and failLookups > 0):
        sys.exit()

#
# Run a type 2 lookup
# This is a specific PUBLISH EXCHANGE ['user', 'exchangeName', 'routingKey'] lookup
#
def LookupPublish(acl, userName, exchName, keyName, expectedResult = ''):
    global totalLookups
    global failLookups
    totalLookups += 1
    result = acl.LookupPublish(userName, exchName, keyName)
    suffix = ''
    if (expectedResult != ''):
        if (result.result != expectedResult):
            failLookups += 1
            suffix = ', [ERROR: Expected ' + expectedResult + ']'
            if (result.result is None):
                suffix = suffix + ', [' + result.text + ']'
    print 'LookupPublish : [name:', userName, \
        ", exchName: '", exchName, "', key: ", keyName, \
        "], [Result: ", result.result, "]", suffix
    if (exitOnError and failLookups > 0):
        sys.exit()

#
# AllBut
#
# Given All names and some names we don't want,
# return the All list with the targets removed
#
def AllBut(allList, removeList):
    tmpList = allList[:]
    for item in removeList:
        try:
            tmpList.remove(item)
        except Exception, e:
            print "ERROR in AllBut() \nallList =  %s \nremoveList =  %s \nerror =  %s " \
                % (allList, removeList, e)
    return tmpList
```

```
#
# Main
#
# Fire up a session and get the acl methods
#

from qmf.console import Session
sess = Session()
broker = sess.addBroker()
acls = sess.getObjects(_class="acl", _package="org.apache.qpid.acl")
acl = acls[0]
# print acl.getMethods() # just to see the method names available

#
# define some group lists
#
g_admins = ['moe@COMPANY.COM', \
          'larry@COMPANY.COM', \
     'curly@COMPANY.COM', \
     'shemp@COMPANY.COM']

g_auditors = [ 'aaudit@COMPANY.COM','baudit@COMPANY.COM','caudit@COMPANY.COM', \
              'daudit@COMPANY.COM','eaduit@COMPANY.COM','eaudit@COMPANY.COM']

g_tatunghosts = ['tatung01@COMPANY.COM', \
                 'tatung02/x86.build.company.com@COMPANY.COM', \
                 'tatung03/x86.build.company.com@COMPANY.COM', \
                 'tatung04/x86.build.company.com@COMPANY.COM', \
                 'HTTP/tatung-test1.eng.company.com@COMPANY.COM']

g_publishusers = ['publish@COMPANY.COM', 'x-pubs@COMPANY.COM']

g_public = ['jpublic@COMPANY.COM', 'me@yahoo.com']

g_all = g_admins + g_auditors + g_tatunghosts + g_publishusers + g_public

action_all = ['consume','publish','create','access','bind','unbind','delete','purge','update']

#
# Run some tests
#
print '#'
print '# admin'
print '#'

for u in g_admins:
    Lookup(acl, u, "create", "queue", "anything", {"durable":"true"}, "allow-log")

print '#'
print '# auditors'
print '#'

uInTest = g_auditors + g_admins
uOutTest = AllBut(g_all, uInTest)

for u in uInTest:
```

```
    LookupPublish(acl, u, "company.topic", "private.audit.This", "allow-log")

for u in uInTest:
    for a in action_all:
        Lookup(acl, u, a, "exchange", "company.topic", {"routingkey":"private.audit.This"}, "allow-log")

for u in uOutTest:
    LookupPublish(acl, u, "company.topic", "private.audit.This", "deny-log")
    Lookup(acl, u, "bind", "exchange", "company.topic", {"routingkey":"private.audit.This"}, "deny-log")

print '#'
print '# tatungs'
print '#'

uInTest = g_admins + g_tatunghosts
uOutTest = AllBut(g_all, uInTest)

for u in uInTest:
    LookupPublish(acl, u, "company.topic",  "tatung.this2",          "allow-log")
    LookupPublish(acl, u, "company.direct", "tatung-service-queue", "allow-log")

for u in uOutTest:
    LookupPublish(acl, u, "company.topic",  "tatung.this2",          "deny-log")
    LookupPublish(acl, u, "company.direct", "tatung-service-queue", "deny-log")

for u in uOutTest:
    for a in ["bind", "access"]:
        Lookup(acl, u, a, "exchange", "company.topic",  {"routingkey":"tatung.this2"},          "allow-log")
        Lookup(acl, u, a, "exchange", "company.direct", {"routingkey":"tatung-service-queue"}, "allow-
log")

print '#'
print '# publishusers'
print '#'

uInTest = g_admins + g_publishusers
uOutTest = AllBut(g_all, uInTest)

for u in uInTest:
    LookupPublish(acl, u, "qpid.management",    "broker",   "allow-log")
    LookupPublish(acl, u, "qmf.default.topic",  "this3",    "allow-log")
    LookupPublish(acl, u, "qmf.default.direct", "this4",    "allow-log")

for u in uOutTest:
    LookupPublish(acl, u, "qpid.management",    "broker",   "deny-log")
    LookupPublish(acl, u, "qmf.default.topic",  "this3",    "deny-log")
    LookupPublish(acl, u, "qmf.default.direct", "this4",    "deny-log")

for u in uOutTest:
    for a in ["bind"]:
        Lookup(acl, u, a, "exchange", "qpid.management",    {"routingkey":"broker"}, "deny-log")
        Lookup(acl, u, a, "exchange", "qmf.default.topic",  {"routingkey":"this3"},  "deny-log")
        Lookup(acl, u, a, "exchange", "qmf.default.direct", {"routingkey":"this4"},  "deny-log")
    for a in ["access"]:
        Lookup(acl, u, a, "exchange", "qpid.management",    {"routingkey":"broker"}, "allow-log")
        Lookup(acl, u, a, "exchange", "qmf.default.topic",  {"routingkey":"this3"},  "allow-log")
```

```
        Lookup(acl, u, a, "exchange", "qmf.default.direct", {"routingkey":"this4"},  "allow-log")

#
# Report statistics
#
print 'Total  Lookups: ', totalLookups
print 'Failed Lookups: ', failLookups

#
# Close the session
#
sess.close()
```

バグを報告します。

9.14. クラスターで QMF の使用

クラスターで QMF メッセージを使用するには、QMF バージョン 2 を使用します。クラスター内で qmf バージョン 1 メッセージは使用できません。

バグを報告します。

---

[1] Java クライアントでは使用できません。

第10章 QPID メッセージング API

10.1. 例外の処理

10.1.1. メッセージング例外リファレンス

メッセージングアプリケーションの非同期および切り離された環境では、ローカルのエラー状態とエラー状態またはリモートで発生する障害の両方について例外が発生します。堅牢なアプリケーションの開発では、さまざまな例外を予測し、処理することが必要になります。一部の例外は、メソッド自体のコンテキストからすぐには明確ではありません。

バグを報告します。

10.1.2. C++ メッセージング例外クラス階層

以下は、C++ API によって発生する例外と、それらが発生する状況です。例外のソースコードは Apache Qpid svn リポジトリー で確認できます。

MessagingException

メッセージング例外のベースクラス。

InvalidOptionString : public MessagingException

接続の設定に使用されるオプション文字列の構文が有効ではない場合に発生します。

KeyError : public MessagingException

一部のローカルオブジェクトの検索に失敗したことを示すためにスローされます。たとえば、名前別にセッション、送信者、またはレシーバーの取得を試行する場合などです。

LinkError : public MessagingException

ベースクラスは、一部のローカルオブジェクトの検索に失敗したことを示すために発生する例外です。

AddressError : public LinkError

一部のローカルオブジェクトの検索に失敗したことを示すためにスローされます。たとえば、名前別にセッション、送信者、またはレシーバーの取得を試行する場合などです。

ResolutionError : public AddressError

構文が正しいアドレスを解決できず、使用できなくなると発生します。

AssertionFailed : public ResolutionError

ノードのアサート済みプロパティーが一致しないアドレスの送信元またはレシーバーの作成時にスローされます。

NotFound : public ResolutionError

存在しないノードに送信者またはレシーバーの作成を試みる際に発生します。

MalformedAddress : public AddressError

無効な構文のアドレス文字列が使用されると発生します。

ReceiverError : public LinkError

FetchError : public ReceiverError

NoMessageAvailable : public FetchError

Receiver::fetch（）、Receiver::get（）、および Session::nextReceiver（）によりスローされ、タイムアウトが発生する前にメッセージがないことを示します。

SenderError : public LinkError

sendError : public SenderError

TargetCapacityExceeded : public SendError

送信側が、ピアのターゲットノードが事前設定された容量を超過することにつながるメッセージの送信を試行することを示しています。

SessionError : public MessagingException

TransactionError : パブリック SessionError

TransactionAborted : public TransactionError

再接続時に Session::commit（）にスローされ、トランザクションが自動的に中止されます。

TransactionUnknown : public TransactionError

ブローカーでのトランザクションの結果（コミットまたはロールバック）は不明です。これは、コミットの送信後、応答を受け取る前に接続が失敗すると発生します。

UnauthorizedAccess : public SessionError

アプリケーションはそのピアが承認されていないことを示すためにスローされました。

UnauthorizedAccess : public SessionError

ConnectionError : public MessagingException

TransportFailure : public MessagingException

根本的な接続の損失を示すためにスローされます。auto-reconnect を使用すると、ライブラリーによってキャプチャーされ、再接続の試行がトリガーされます。（設定がどのような設定によって）再接続に失敗すると、このクラスのインスタンスがスローされ、これを示します。

バグを報告します。

10.1.3. 接続例外

注記： 完全修飾なし、上記のすべての例外は **qpid::messaging** namespace にあります。

connection::Connection（const std::string&、const qpid::types:: Variant::Map&）

MessagingException 指定されたマップにあるオプションのいずれかが認識されない場合。

qpid::types::InvalidConversion オプションの値のタイプが間違っている場合。

connection::Connection（const std::string& url, const std::string& オプション）

MessagingException 指定されたマップにあるオプションのいずれかが認識されない場合。

qpid::types::InvalidConversion オプションの値のタイプが間違っている場合。

InvalidOptionString オプション文字列の形式が無効である場合。

Connection::setOption(const std::string& name, const qpid::types::Variant& value)

MessagingException named オプションが認識されない場合。

qpid::types::InvalidConversion オプションの値が正しくないタイプである場合。

Connection::open()

qpid::Url::Invalid url が有効でない場合は（構築時に提供される url、またはオプション経由で提供された reconnect_urls のいずれか）。

TransportFailure 接続が確立されていない場合。

ConnectionError AMQP 0-10 で定義された接続ハンドシェイクが完了するまでに、ブローカーが connection.close コントロールを送信する他の障害の場合。

qpid::types::InvalidConversion ブローカーが AMQP 0-10 仕様で定義されている '**known-host**' フィールドに誤ってエンコード **connection.open-ok control** された値を送信する場合。

Connection::isOpen()

例外は発生しません。

Connection::close()

TargetCapacityExceeded 接続に確立されたセッションのいずれかがメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess 接続に対して確立されたセッションのいずれかが、パーミッションが付与されていない操作の実行を試行する場合。

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが connection.close コントロールを送信する場合（たとえば、クライアントが行う直前にアクティブな接続を閉じる場合など）。

MessagingException クライアントが接続しているブローカーが session.detached コントロールを送信する場合（つまり、閉じるが進行中にアクティブなセッションを閉じる場合）。

TransportFailure ブローカーでクローズされた 'handshake' を実行しようとする際に接続が失われた場合。

Connection::createTransactionalSession(const std::string& name)

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信され、セッションのトランザクションを有効にする際に発生する可能性があります（例： 問題のブローカーがトランザクションをサポートしていない場合）。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが session.detached コントロールを送信する場合（たとえば、アクティブになる前にブローカーがセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

qpid::Url::Invalid 再接続が有効で、**reconnect_urls** オプション一覧の URL が無効である場合。

qpid::types::InvalidConversion AMQP 0-10 仕様で定義された **connection.open-ok** コントロールの '**known-host**' フィールドに対して、ブローカーが誤ってエンコードされた値を送信する場合。

Connection::createSession(const std::string&)

ConnectionError クライアントが接続しているブローカーが connection.close control を送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（たとえば、ブローカーがセッションを閉じるとアクティブになる前にセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

qpid::Url::Invalid 再接続が有効で、**reconnect_urls** オプション一覧の URL が無効である場合。

qpid::types::InvalidConversion AMQP 0-10 仕様で定義されている connection.open-ok コントロールの**known-host**' ' フィールドに誤ってエンコードされた値を送信する場合。

Connection::getSession(const std::string&)

KeyError 指定した名前のセッションが存在しない場合。

Connection::getAuthenticatedUsername()

例外は発生しません。

バグを報告します。

10.1.4. セッション例外

注記： 完全修飾なし、上記のすべての例外は **qpid::messaging** namespace にあります。

Session::close()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::commit()

TransactionAborted 元の AMQP 0-10 セッションが失われた場合（フェイルオーバーにより）、自動ロールバックが強制されます。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::rollback()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::acknowledge(bool)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::acknowledge(Message&, bool)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::acknowledgeUpTo(Message&, bool)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::reject(Message&)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

AMQP 0-10 で定義されているように execution.exception コマンドがクライアントの接続先のブローカーから受信された **SessionError** 場合に実行されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::release(Message&)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

接続が失われた場合に TransportFailure（自動再接続が有効である場合は再確立できません）。

Session::sync(bool)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::getReceivable()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::getUnsettledAcks()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

接続が失われた場合に TransportFailure（自動再接続が有効である場合は再確立できません）。

session::nextReceiver(Receiver&, Duration)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::nextReceiver(Duration)

Receiver::NoMessageAvailable 時間内にメッセージが利用可能になっていない場合。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

AMQP 0-10 で定義されているように execution.exception コマンドがクライアントの接続先のブローカーから受信された **SessionError** 場合に実行されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::createSender(const Address&)

ResolutionError アドレスの解決にエラーがある場合。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::createSender(const std::string&)

ResolutionError アドレスの解決にエラーがある場合。

MalformedAddress アドレス文字列の構文が有効ではない場合。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

session::createReceiver(const Address&)

ResolutionError アドレスの解決にエラーがある場合。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが connection.close control を送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::createReceiver(const std::string&)

ResolutionError アドレスの解決にエラーがある場合。

MalformedAddress アドレス文字列の構文が有効ではない場合。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Session::getSender(const std::string&)

KeyError 指定の名前の送信者がない場合。

Session::getReceiver(const std::string&)

指定の名前のレシーバーがない場合、KeyError

Session::checkError()

qpid::messaging::SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

qpid::messaging::ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

qpid::messaging::MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

Session::getConnection()

例外は発生しません。

Session::hasError()

例外は発生しません。

バグを報告します。

10.1.5. 送信者例外

注記： 完全修飾なし、上記のすべての例外は **qpid::messaging** namespace にあります。

sender::send(const Message& message, bool)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続するブローカーが session.detached コントロールを送信する場合（つまり、ブローカーがアクティブなセッションを閉じる場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Sender::close()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Sender::setCapacity(uint32_t)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Sender::getUnsettled()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Sender::getAvailable()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Sender::getCapacity()

例外は発生しません。

Sender::getName()

例外は発生しません。

Sender::getSession()

例外は発生しません。

バグを報告します。

10.1.6. receiver 例外

注記： 完全修飾なし、上記のすべての例外は **qpid::messaging** namespace にあります。

receiver::get(Message& message, Duration timeout=Duration::FOREVER)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

receiver::Message get(Duration timeout=Duration::FOREVER)

NoMessageAvailable 指定されたタイムアウトを待機した後に指定のタイムアウトを待機するメッセージがない場合、または Receiver が閉じられている場合は true に **isClose()** なります。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

receiver::fetch(Message& message, Duration timeout=Duration::FOREVER)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された execution.exception コマンドが、クライアントが接続先のブローカーから受信される場合。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

receiver::fetch(Duration timeout=Duration::FOREVER)

NoMessageAvailable 指定されたタイムアウトを待機した後に指定のタイムアウトを待機するメッセージがない場合、または Receiver が閉じられている場合は true に **isClose()** なります。

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Receiver::setCapacity(uint32_t)

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Receiver::getAvailable()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Receiver::getUnsettled()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Receiver::close()

TargetCapacityExceeded セッションが、設定された制限を超えるメッセージを送信しようとすると、キューが設定された制限を超えることになります。

UnauthorizedAccess セッションがパーミッションを付与されていない操作を実行しようとすると、

SessionError AMQP 0-10 で定義された **execution.exception** コマンドが、クライアントが接続しているブローカーから受信されます。

ConnectionError クライアントが接続しているブローカーが **connection.close** コントロールを送信する場合（つまり、ブローカーがアクティブな接続のクローズを開始する場合）。

MessagingException クライアントが接続しているブローカーが **session.detached** 制御を送信する場合（つまり、ブローカーがアクティブなセッションのクローズを開始する場合）。

TransportFailure 接続が失われた場合（自動再接続が有効になっていない場合）。

Receiver::isClosed()

例外は発生しません。

Receiver::getCapacity()

例外は発生しません。

Receiver::getName()

例外は発生しません。

Receiver::getSession()

例外は発生しません。

バグを報告します。

第11章 ADDRESSES

11.1. X 宣言パラメーター

以下のパラメーターは、アドレス文字列の **x-declare** 一部に指定できます。

表11.1

パラメーター	使用方法
auto-delete	**boolean** キュー/交換が自動的に削除されるべきかどうかの指定
exclusive	**boolean** キュー/交換の排他的性の指定
alternate-exchange	このキューが削除される際にメッセージをルーティングする代替の交換 / メッセージの一致するバインドを見つけることができない
arguments	キュー/交換に特に使用できる引数を含むネストされたマップ。詳細はを https://cwiki.apache.org/confluence/display/qpid/Qpid+extensions+to+AMQP 参照してください。

バグを報告します。

11.2. アドレス文字列オプションのリファレンス

表11.2

オプション	値	セマンティクス
assert	、**always**、**sender** またはのいずれ **never**か。 **receiver**	ノードオプションで指定されたプロパティーがアドレスを解決するすべてのプロパティーをアサートします。そうでない場合は解決に失敗し、例外が発生します。
create	、**always**、**sender** またはのいずれ **never**か。 **receiver**	アドレスが存在しないかどうかが参照するノードを作成します。ノードが存在する場合、エラーは発生しません。ノードの詳細は、node オプションで指定できます。
delete	、**always**、**sender** またはのいずれ **never**か。 **receiver**	送信者またはレシーバーが閉じられたときにノードを削除します。
node	**node** プロパティーを含むネストされたマップ。	アドレスが参照するノードのプロパティーを指定します。これらは **assert** または **create** オプションとともに使用されます。

オプション	値	セマンティクス
link	**link** プロパティーを含むネストされたマップ。	クライアントアプリケーションからターゲット/ソースアドレスへの、概念リンクの構築を制御するために使用されます。
mode	以下のいずれか **browse**になります。 **consume**	このオプションは、キューに解決するソースアドレスにのみ関連します。指定されている場合、受信側に配信されるメッセージは削除されず、キューに残されます。consume が指定されていると、通常の動作が適用されます。クライアントが受信を認識すると、メッセージはキューから削除されます。

バグを報告します。

11.3. ノードのプロパティー

表11.3

property	値	セマンティクス
type	以下のいずれか **topic**になります。 **queue**	ノードのタイプを示します。
durable	以下のいずれか **True**になります。 **False**	ブローカーが再起動した場合など、ノードが揮発性ストレージの損失後も存続するかどうかを示します。
x-declare	値が AMQP 0-10 **queue-declare** または **exchange-declare** コマンドの有効なフィールドに対応するネストされたマップ。	これらの値は、作成またはアサーションプロセスを微調整するために使用されます。ただし、これはプロトコル固有であることに注意してください。

property	値	セマンティクス

x-bindings	各バインディングがマップによって表されるネストされたリスト。バインディングのマップのエントリーには、AMQP 0-10 バインディングを記述するフィールドが含まれます。以下は x-bindings の形式です。
	
	[
	{
	exchange: <exchange>,
	queue: <queue>,
	key: <key>,
	arguments: {
	<key_1>: <value_1>,
	...,
	<key_n>: <value_n> }
	},
	...
	]
		create オプションと併せて、これらのバインディングはアドレスが解決されると、これらのバインディングが確立されます。assert オプションとともに、これらの各バインディングが存在するかどうかは解決中に検証されます。ここでも、これはプロトコル固有です。

properties	AMQP 1.0 プロパティーのネストされたマップ。	で指定したプロパティーのネストされたマップ **properties** は **x-declare**、の使用よりも推奨されます。これにより、プロパティーの使用時にネストされたプロパティーのマップが生成されます。
capabilities	AMQP 1.0 の機能を表す単一の文字列または文字列のリスト。	ソースまたはターゲットから要求される AMQP 1.0 機能が含まれるリスト。

バグを報告します。

11.4. リンクプロパティー

表11.4

オプション	値	セマンティクス

オプション	値	セマンティクス
reliability	以下のいずれか **unreliable**、**at-least-once**、 **at-most-once**[a], **exactly-once**[b]	現在 **unreliable**、およびのみサポートさ **at-least-once** れています。詳細は脚注を参照してください。 信頼性は、送信者またはレシーバーが要求するリンク信頼性のレベルを示し、**unreliable at-most-once** 現在シノニムとして扱われます。また、ブローカーがクラッシュしたり、ブローカーへの接続が失われたりした場合にメッセージが失われるようにします。メッセージが失われる可能性はなく、**at-least-once** メッセージが失われていないことを **exactly-once** 保証します。メッセージが失われていないことを保証し、正確に配信されます。
durable	: **True**、の 1 つ **False**。	ブローカーが再起動した場合など、リンクが揮発性ストレージの損失後も存続するかどうかを示します。
x-declare	値が AMQP 0-10 **queue-declare** コマンドの有効なフィールドに対応するネストされたマップ。	これらの値は、交換からの受信時にサブスクリプションキューをカスタマイズするために使用できます。ただし、これはプロトコル固有であることに注意してください。
x-subscribe	値が AMQP 0-10 **message-subscribe** コマンドの有効なフィールドに対応するネストされたマップ。	これらの値は、サブスクリプションのカスタマイズに使用できます。
x-bindings	AMQP 0-10 バインディングを記述するフィールド（、**queue exchange**、**key** および **arguments**）が含まれる可能性がある各エントリーのネストされたリスト。	これらのバインディングは、create オプションとは別に解決中に確立されます。それらは、ノードの作成ではなく、リンクプロセスの一部と見なされます。
filter	AMQP 1.0 フィルターが含まれる **name descriptor value**、、およびが含まれるマップ。	**name** はアプリケーションを選択する名前 **descriptor** です。フィルタータイプを特定する文字列記述子です。**value** はフィルターのタイプで決まります（例： **string legacy-amqp-direct-binding**および **map** for **legacy-amqp-headers-binding**）。

オプション	値	セマンティクス

[a] 要求されて **at-most-once** いる場合は、が使用 **unreliable** されます。

[b] 要求されて **exactly-once** いる場合は、が使用 **at-least-once** されます。

バグを報告します。

11.5. アドレス文字列グラフ

トークン

以下の正規表現は、アドレス文字列の解析に使用されるトークンを定義します。

```
LBRACE: \\{
RBRACE: \\}
LBRACK: \\[
RBRACK: \\]
COLON:  :
SEMI:   ;
SLASH:  /
COMMA:  ,
NUMBER: [+-]?[0-9]*\\.?[0-9]+
ID:     [a-zA-Z_](?:[a-zA-Z0-9_-]*[a-zA-Z0-9_])?
STRING: "(?:[^\\\\"]|\\\\.)*"|\'(?:[^\\\\']|\\\\.)*\'
ESC:    \\\\[^ux]|\\\\x[0-9a-fA-F][0-9a-fA-F]|\\\\u[0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]
SYM:    [.#*%@$^!+-]
WSPACE: [ \\n\\r\\t]+
```

文法

アドレスの正式な文法は以下のとおりです。

```
address := name [ SLASH subject ] [ ";" options ]
   name := ( part | quoted )+
subject := ( part | quoted | SLASH )*
 quoted := STRING / ESC
   part := LBRACE / RBRACE / COLON / COMMA / NUMBER / ID / SYM
options := map
    map := "{" ( keyval ( "," keyval )* )? "}"
 keyval "= ID ":" value
  value := NUMBER / STRING / ID / map / list
   list := "[" ( value ( "," value )* )? "]"
```

アドレス文字列のオプション

アドレス文字列オプションマップでは、以下のパラメーターがサポートされます。

AMQP 0-10

```
<name> [ / <subject> ] ; {
  create: always | sender | receiver | never,
  delete: always | sender | receiver | never,
```

```
  assert: always | sender | receiver | never,
  mode: browse | consume,
  node: {
    type: queue | topic,
    durable: True | False,
    x-declare: { ... <declare-overrides> ... },
    x-bindings: [<binding_1>, ... <binding_n>]
  },
  link: {
    name: <link-name>,
    durable: True | False,
    reliability: unreliable | at-most-once | at-least-once | exactly-once,
    x-declare: { ... <declare-overrides> ... },
    x-bindings: [<binding_1>, ... <binding_n>],
    x-subscribe: { ... <subscribe-overrides> ... }
  }
}
```

AMQP 1.0

```
<name> [ / <subject> ] ; {
  create: always | sender | receiver | never,
  assert: always | sender | receiver | never,
  mode: browse | consume,
  node: {
    type: queue | topic,
    durable: True | False,
    properties: { ... <nested-map> ... }[2],
    capabilities: [<capability_1>, ... <capability_n>]
  },
  link: {
    name: <link-name>,
    durable: True | False,
    reliability: unreliable | at-most-once | at-least-once | exactly-once,
    filter: { name: <name>, descriptor: <filter-descriptor>, value: <filter-value> }
  }
}
```

Create、Delete、および Assert ポリシー

（ **create delete** AMQP 0-10 のみ）、および **assert** ポリシーは、関連するアクションを実行するユーザーを指定します。

Always

アクションはすべてのメッセージングクライアントによって実行されます。

sender

アクションは送信元によってのみ実行されます。

receiver

アクションはレシーバーによってのみ実行されます。

Never

アクションは実行されない（デフォルト）。

node-Type

は以下の **node-type** いずれかになります。

トピック

AMQP 0-10 マッピングでは、トピックノードはデフォルトでトピック交換に、x-declare を使用して他の交換タイプを指定できます。

Queue

これがデフォルトです。 **node-type**

フィルター記述子

以下の AMQP 1.0 フィルターは MRG 3 に実装されます。

- **legacy-amqp-direct-binding**
- **legacy-amqp-topic-binding**
- **legacy-amqp-headers-binding**
- **selector-filter**
- **xquery-filter**

バグを報告します。

11.6. 接続オプション

接続の動作は、接続オプションを使用して制御できます。たとえば、ブローカーへの接続が失われた場合に、接続を自動的に再接続するように設定できます。

バグを報告します。

11.7. 接続オプションの設定

接続オプションを設定する方法は 2 つあります。1 つ目は、Connection コンストラクターで行います。

python

```
connection = Connection("localhost:5672", reconnect = True, reconnect_urls =
"amqp:tcp:127.0.0.1:5674", heartbeat = 1)
try:
  connection.open()
```

C++

```
Connection connection("localhost:5672", "{reconnect: true,
reconnect_urls:'amqp:tcp:127.0.0.1:5674', reconnect:true, heartbeat: 1}");
try {
    connection.open();
```

.NET/C#

```
Connection connection= new Connection("localhost:5672", "{reconnect: true,
reconnect_urls:'amqp:tcp:127.0.0.1:5674', reconnect:true, heartbeat: 1}");
try {
    connection.Open();
```

2 つ目は、Connection プロパティーを使用してこれを実行することです。

python

```
connection = Connection("localhost:5672")
connection.reconnect = True
try:
  connection.Open()
```

C++

```
Connection connection("localhost:5672");
connection.setOption("reconnect", true);
try {
    connection.open();
```

.NET/C#

```
Connection connection = new Connection("localhost:5672");
connection.SetOption("reconnect", true);
try {
    connection.Open();
```

バグを報告します。

11.8. 接続オプションのリファレンス

表11.5 接続オプション（一般）

オプション名	値のタイプ	セマンティクス
username	string	ブローカーへの認証時に使用するユーザー名。
password	string	ブローカーへの認証時に使用するパスワード。
heartbeat	整数	ハートビートが *N* 秒ごとに送信される要求。連続するハートビートが 2 つ不足すると、接続は失われたと見なされ、失敗するか、再接続プロセスを開始する（設定されている場合）。

オプション名	値のタイプ	セマンティクス
max-channels	整数	Messaging API のチューニングを支援するために、サポートされているチャネルの最大数を制限します。**AMPQ 1.0 ではサポートされません。**
max-frame-size	整数	メッセージング API のチューニングに役立つ最大フレームサイズを制限します。**AMPQ 1.0 ではサポートされません。** **最小値は 4096B 以上で、小さいと認証に失敗します。このプロダクトは、この制限を適用しません。**
protocol	string	使用する AMQP プロトコル。認識できる値は '**amqp1.0**' および**amqp0-10**' です。AMQP 0-10 がデフォルトです。**注記：Python クライアントではサポートされません。**
reconnect	ブール値	接続が失われた場合は透過的に再接続します。
reconnect_urls	ブローカーのアドレスリスト	接続の失敗時に通信を試行する 1 つ以上のブローカーの一覧。
reconnect_urls_replace	ブール値	**reconnect_urls** オプションの処理方法を制御します。true の場合、設定 **reconnect_urls** により古いリストが新しいリストに置き換えられます。false の場合、新しいリストが古いリストに追加されます。デフォルト値は false です。
reconnect_timeout	float	断念して例外を発生させるまでの再接続試行を継続する合計秒数。
reconnect_limit	整数	例外を断念して発生するまでの再接続試行の最大数。

オプション名	値のタイプ	セマンティクス
reconnect_interval_min	float	再接続試行までの最小秒数。最初の再接続試行は即座に行われます。失敗した場合は、最初の再接続遅延がの値に設定されます **reconnect_interval_min**。失敗した場合は、再接続の試行が成功または完了するまで、再接続間隔 **reconnect_interval_max** が指数関数的に増加します。この値は分数にすることができます。たとえば、0.001 は最大再接続間隔を 1 ミリ秒に設定します。
reconnect_interval_max	float	最大再接続間隔（秒単位）。この値は分数にすることができます。たとえば、0.001 は最大再接続間隔を 1 ミリ秒に設定します。
reconnect_interval	float	**reconnection_interval_min** およびと **reconnection_interval_max** を同じ秒数に設定します。
sasl_mechanisms	string	ブローカーに対して、スペースで区切られたリストとして認証する際に使用する特定の SASL メカニズム。
sasl_service	string	使用中の SASL メカニズムで必要な場合はサービス名。
sasl_min_ssf	整数	許容される最小のセキュリティー強度係数。
sasl_max_ssf	整数	許容できる最大セキュリティー強度係数。
ssl_cert_name	string	指定のクライアントに使用する証明書の名前。
ssl_ignore_hostname_verification_failure	ブール値	クライアントへのサーバーの認証を無効にします（最後の手段としてのみ使用してください）。true に設定すると、使用されるホスト名（または IP アドレス）がサーバー証明書にある内容と一致しない場合でも、クライアントはサーバー証明書に接続できます。

オプション名	値のタイプ	セマンティクス
tcp_nodelay	ブール値	セット（***tcp_no_delay***例：Nagle アルゴリズムを無効にします）。**注記： Python クライアントではサポートされません。**
transport	string	使用されるトランスポートプロトコルを設定します。デフォルトのオプションはです ***tcp***。ssl を有効にするには、に設定し ***ssl***ます。C++ クライアントは、追加でサポートし ***rdma***ます。

表11.6 接続オプション（Python Client のみ）

オプション名	値のタイプ	セマンティクス
address_ttl	float	キャッシュされたアドレス解決の有効期限が切れるまでの時間。
host	string	リモートホストの名前または ip アドレス（上書き **url**）。
port	整数	リモートホストのポート番号（上書き **url**）。
ssl_certfile	string	クライアントの公開鍵（PEM 形式）のあるファイル。
ssl_keyfile	string	クライアントの秘密鍵（PEM 形式）のあるファイル。
ssl_trustfile	string	サーバーを検証するための信頼された証明書が含まれるファイル。
url	string	[*<username>* [/ *<password>*] @] *<host>* [:*<port>*].

表11.7 接続オプション（AMPQ 1.0 のみ）

オプション名	値のタイプ	セマンティクス
container_id	string	接続に使用するコンテナー ID。

オプション名	値のタイプ	セマンティクス
nest_annotations	ブール値	true の場合、受信したメッセージのアノテーションはプロパティーとして示され、キー x-amqp-delivery-annotations または x-amqp-delivery-annotations が表示されます。値は、アノテーションを含むネストされたマップで構成されます。false の場合、アノテーションはプロパティーとマージされます。
set_to_on_send	ブール値	true の場合、送信されたすべてのメッセージでは、to フィールドに送信者のノード名が設定されます。
properties または **client_properties**	整数	送信された、開いているフレームに追加するプロパティー。

バグを報告します。

---

[2] The use of new **properties** nested map is recommended. The **x-declare** map is supported as a convenience and is automatically converted to a **properties** map before sending to the broker.

第12章 メッセージのタイムスタンプ

12.1. メッセージのタイムスタンプ

メッセージはブローカーに到達した日時でタイムスタンプを付けることができます。メッセージのタイムスタンプはデフォルトで無効になっています。

バグを報告します。

12.2. BROKER START-UP でのメッセージのタイムスタンプの有効化

Broker の起動時にメッセージのタイムスタンプを有効にするには、**--enable-timestamp yes** 引数を使用してブローカーを起動します。

```
./qpidd --enable-timestamp yes
```

バグを報告します。

12.3. アプリケーションからのメッセージのタイムスタンプの有効化

qmf コマンドメッセージを使用すると、ブローカーを再起動しなくても、アプリケーションからタイムスタンプを有効または無効にすることができます。

QMF メソッド **getTimestampConfig** およびタイムスタンプ設定の **setTimestampConfig** 取得および設定。

getTimestampConfig

受信したメッセージがタイムスタンプとなった **True** 場合に返されます。

setTimestampConfig

受信 **True** したメッセージのタイムスタンプを有効にする **False** には、タイムスタンプを無効にします。

バグを報告します。

12.4. PYTHON のメッセージのタイムスタンプへのアクセス

以下のコードは、受信したメッセージからメッセージのタイムスタンプを確認し、抽出します。

```
try:
    msg = receiver.fetch(timeout=1)
    if "x-amqp-0-10.timestamp" in msg.properties:
        print("Timestamp=%s" % str(msg.properties["x-amqp-0-10.timestamp"]))
except Empty:
    pass
```

バグを報告します。

12.5. C++ のメッセージのタイムスタンプへのアクセス

以下のコードは、受信したメッセージからメッセージのタイムスタンプを確認し、抽出します。

```
messaging::Message msg;
if (receiver.fetch(msg, messaging::Duration::SECOND*1)) {
    if (msg.getProperties().find("x-amqp-0-10.timestamp") !=
msg.getProperties().end()) {
        std::cout << "Timestamp=" <<
msg.getProperties()["x-amqp-0-10.timestamp"].asString() << std::endl;
    }
}
```

バグを報告します。

12.6. AMQ 0-10 メッセージプロパティーキーのタイムスタンプの使用

タイムスタンプ配信プロパティーが受信メッセージ(***delivery-properties.timestamp***)に設定されていると、***x-amqp-0-10.timestamp*** message プロパティーを使用してタイムスタンプの値にアクセスできます。

関連項目

- *19章AMQP 0-10 マッピング*

バグを報告します。

第13章 マップおよびリスト

13.1. メッセージコンテンツ内のマップおよびリスト

メッセージングアプリケーションは、多くの場合、言語およびプラットフォーム間でデータを交換する必要があります。メッセージにはマップとリストを含めることができます。

バグを報告します。

13.2. ネイティブデータタイプのマップおよび一覧表示

表13.1 サポートされている言語でのマップおよび一覧表示

言語	map	list
python	**dict**	**list**
C++	**Variant::Map**	**Variant::List**
Java	**MapMessage**	**ListMessage**
.NET	**Dictionary<string, object>**	**Collection<object>**

バグを報告します。

13.3. PYTHON での QPID マップおよびリスト

Python では、Qpid は、メッセージコンテンツで dict および list タイプを直接サポートします。以下のコードは、これらの構造をメッセージに送信する方法を示しています。

python

```
from qpid.messaging import *
# !!! SNIP !!!

content = {'Id' : 987654321, 'name' : 'Widget', 'percent' : 0.99}
content['colours'] = ['red', 'green', 'white']
content['dimensions'] = {'length' : 10.2, 'width' : 5.1,'depth' : 2.0};
content['parts'] = [ [1,2,5], [8,2,5] ]
content['specs'] = {'colors' : content['colours'],
                    'dimensions' : content['dimensions'],
                    'parts' : content['parts'] }
message = Message(content=content)
sender.send(message)
```

バグを報告します。

13.4. マップの PYTHON データタイプ

以下の表は、Python マップメッセージで送信できるデータタイプと、Java または C++ のクライアントが受信する対応するデータタイプを示しています。

表13.2 マップの Python データタイプ

Python データタイプ	→ C++	→ Java
bool	**bool**	**boolean**
int	**int64**	**long**
long	**int64**	**long**
float	**double**	**double**
unicode	**string**	**java.lang.String**
uuid	**qpid::types::Uuid**	**java.util.UUID**
dict	**Variant::Map**	**java.util.Map**
list	**Variant::List**	**java.util.List**

バグを報告します。

13.5. C++ の QPID マップおよびリスト

C++ では、Qpid は、メッセージコンテンツにエンコードできる Variant::Map および Variant::List タイプを定義します。以下のコードは、これらの構造をメッセージに送信する方法を示しています。

C++

```
using namespace qpid::types;

// !!! SNIP !!!

Message message;
Variant::Map content;
content["id"] = 987654321;
content["name"] = "Widget";
content["percent"] = 0.99;
Variant::List colours;
colours.push_back(Variant("red"));
colours.push_back(Variant("green"));
colours.push_back(Variant("white"));
content["colours"] = colours;

Variant::Map dimensions;
dimensions["length"] = 10.2;
dimensions["width"] = 5.1;
dimensions["depth"] = 2.0;
content["dimensions"]= dimensions;
```

```
Variant::List part1;
part1.push_back(Variant(1));
part1.push_back(Variant(2));
part1.push_back(Variant(5));

Variant::List part2;
part2.push_back(Variant(8));
part2.push_back(Variant(2));
part2.push_back(Variant(5));

Variant::List parts;
parts.push_back(part1);
parts.push_back(part2);
content["parts"]= parts;

Variant::Map specs;
specs["colours"] = colours;
specs["dimensions"] = dimensions;
specs["parts"] = parts;
content["specs"] = specs;

message.setContentObject(content);
sender.send(message, true);
```

バグを報告します。

13.6. マップの C++ データタイプ

以下の表は、C++ マップメッセージで送信できるデータタイプと、Java および Python でクライアントが受信する対応するデータタイプを示しています。

表13.3 マップの C++ データタイプ

C++ Data Type	→ Python	→ Java
bool	**bool**	**boolean**
uint16	**int | long**	**short**
uint32	**int | long**	**int**
uint64	**int | long**	**long**
int16	**int | long**	**short**
int32	**int | long**	**int**
int64	**int | long**	**long**
float	**float**	**float**

C++ Data Type	→ Python	→ Java
double	float	double
string	unicode	java.lang.String
qpid::types::Uuid	uuid	java.util.UUID
Variant::Map	dict	java.util.Map
Variant::List	list	java.util.List

バグを報告します。

13.7. .NET C# の QPID マップおよびリスト

Qpid Messaging API の .NET バインディングは、.NET 管理データタイプを C++ バリアントデータタイプにバインドします。以下のコードは、メッセージに Variant::Map および Variant::List 構造を送信する方法を示しています。

.NET/C#

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Org.Apache.Qpid.Messaging;

namespace Org.Apache.Qpid.Messaging.examples
{
    class MapSender
    {
        // csharp.map.sender example
        //
        // Send an amqp/map message
        // The map message contains simple types, a nested amqp/map,
        // an ampq/list, and specific instances of each supported type.
        //
        static int Main(string[] args)
        {
            string url = "amqp:tcp:localhost:5672";
            string address = "message_queue; {create: always}";
            string connectionOptions = "";

            if (args.Length > 0)
                url = args[0];
            if (args.Length > 1)
                address = args[1];
            if (args.Length > 2)
                connectionOptions = args[2];

            //
            // Create and open an AMQP connection to the broker URL
```

```
//
Connection connection = new Connection(url, connectionOptions);
connection.Open();

//
// Create a session and a sender
//
Session session = connection.CreateSession();
Sender sender = session.CreateSender(address);

//
// Create structured content for the message.  This example builds a
// map of items including a nested map and a list of values.
//
Dictionary<string, object> content = new Dictionary<string, object>();
Dictionary<string, object> subMap = new Dictionary<string, object>();
Collection<object> colors = new Collection<object>();

// add simple types
content["id"] = 987654321;
content["name"] = "Widget";
content["percent"] = 0.99;

// add nested amqp/map
subMap["name"] = "Smith";
subMap["number"] = 354;
content["nestedMap"] = subMap;

// add an amqp/list
colors.Add("red");
colors.Add("green");
colors.Add("white");
// list contains null value
colors.Add(null);
content["colorsList"] = colors;

// add one of each supported amqp data type
bool mybool = true;
content["mybool"] = mybool;

byte mybyte = 4;
content["mybyte"] = mybyte;

UInt16 myUInt16 = 5 ;
content["myUInt16"] = myUInt16;

UInt32 myUInt32 = 6;
content["myUInt32"] = myUInt32;

UInt64 myUInt64 = 7;
content["myUInt64"] = myUInt64;

char mychar = 'h';
content["mychar"] = mychar;

Int16 myInt16 = 9;
```

```
            content["myInt16"] = myInt16;

            Int32 myInt32 = 10;
            content["myInt32"] = myInt32;

            Int64 myInt64 = 11;
            content["myInt64"] = myInt64;

            Single mySingle = (Single)12.12;
            content["mySingle"] = mySingle;

            Double myDouble = 13.13;
            content["myDouble"] = myDouble;

            Guid myGuid = new Guid("000102030405060708090a0b0c0d0e0f");
            content["myGuid"] = myGuid;

            content["myNull"] = null;

            //
            // Construct a message with the map content and send it synchronously
            // via the sender.
            //
            Message message = new Message(content);
            sender.Send(message, true);

            //
            // Wait until broker receives all messages.
            //
            session.Sync();

            //
            // Close the connection.
            //
            connection.Close();
            return 0;
        }
    }
}
```

バグを報告します。

13.8. C# データタイプおよび.NET バインディング

以下の表は、.NET と C++ のデータタイプのマッピングを示しています。

表13.4 C++ と .NET バインディング間のデータタイプマッピング

C++ のデータタイプ	.NET バインディング
void	**nullptr**
bool	**bool**

C++ のデータタイプ	.NET バインディング
uint8	**byte**
uint16	**UInt16**
uint32	**UInt32**
uint64	**UInt64**
int16	**char**
int16	**Int16**
int32	**Int32**
int64	**Int64**
float	**Single**
double	**Double**
string	**string**
qpid::types::Uuid	**Guid**
Variant::Map	**Dictionary< string, object >**
Variant::List	**Collection< object >**

注記

.NET **string** オブジェクトは、UTF-8 エンコーディングのみを使用して C++ 文字列に変換されます。

バグを報告します。

第14章 要求/レスポンスパターン

14.1. 要求/レスポンスパターン

リクエスト/応答アプリケーションは message プロパティーを使用して、サーバーが **reply-to** メッセージを送信したクライアントに応答できるようにします。サーバーは、クライアントが認識する名前でサービスキューを設定します。クライアントはサーバーの応答のプライベートキューを作成し、リクエストのメッセージを作成し、リクエストの respond-to プロパティーをクライアントの応答キューのアドレスに設定し、リクエストをサービスキューに送信します。サーバーは、リクエストの **reply-to** プロパティーで指定されたアドレスに応答を送信します。

バグを報告します。

14.2. リクエスト/レスポンス C++ の例

この例では、リクエスト/応答パターンを使用するクライアントおよびサーバーです。サーバーはサービスキューを作成し、メッセージが到達するのを待機します。メッセージを受信すると、メッセージを送信元に送信します。

C++

```
Receiver receiver = session.createReceiver("service_queue; {create: always}");

Message request = receiver.fetch();
const Address& address = request.getReplyTo(); // Get "reply-to" from request ...
if (address) {
  Sender sender = session.createSender(address); // ... send response to "reply-to"
  Message response("pong!");
  sender.send(response);
  session.acknowledge();
}
```

クライアントはサービスキューの送信側を作成し、クライアントが応答キューの受信側を閉じたときに削除される応答キューも作成します。C++ クライアントでは、アドレスが文字で始まる場合は **#**、一意の名前が指定されます。

C++

```
Sender sender = session.createSender("service_queue");

Receiver receiver = session.createReceiver("#response-queue; {create:always}");
Address responseQueue = receiver.getAddress();

Message request;
request.setReplyTo(responseQueue);
request.setContent("ping");
sender.send(request);
Message response = receiver.fetch();
std::cout << request.getContent() << " -> " << response.getContent() << std::endl;
```

クライアントは文字列 ping をサーバーに送信します。サーバーは、replyTo プロパティーを使用して応答 pong を同じクライアントに送信します。

バグを報告します。

第15章 パフォーマンスのヒント

15.1. APACHE QPID プログラミングによるパフォーマンス

- レシーバーのメッセージを事前フェッチすることを検討してください。これにより、ラウンドトリップがなくなり、スループットが向上します。prefetch はデフォルトで無効になっており、受信したメッセージのスループットを改善する最も効果的な方法です。
- メッセージを非同期に送信します。ここでも、これによりラウンドトリップが解消され、スループットが向上します。C++ および .NET クライアントはデフォルトで非同期的に送信しますが、python クライアントはデフォルトで同期送信を行います。
- バッチで確認応答します。メッセージごとに個別に承認するのではなく、メッセージや特定の期間が経過した後に確認応答を発行することを検討してください。
- 送信側の容量を調整します。容量が低い場合、送信側はより多くの容量を解放する前に、ブローカーがメッセージの受信を確認するためにブロックされる可能性があります。
- c++ クライアントが送信するメッセージに応答アドレスを設定する場合は、アドレスタイプが必要に応じて queue または topic に設定されていることを確認します。これにより、AMQP 0-10 で応答の処理時に必要となるノードの種類をクライアントが判断する必要がなくなりました。
- レイテンシーに敏感なアプリケーションの場合、クライアント接続の ***tcp-nodelay*** オン **qpidd** および on を設定すると、レイテンシーを短縮できます。

バグを報告します。

第16章 クラスターフェイルオーバー

16.1. 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。

16.2. 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。

16.3. C++ でのクラスターフェイルオーバー

仮想 IP を使用するクラスターで MRG 3 C++ クライアントを使用するには、仮想 IP アドレスをブローカーアドレスとして指定します。フェイルオーバーは、クラスターマネージャーによって透過的に処理されます。

仮想 IP アドレスを使用しないクラスターがある場合には、複数のクラスターノードアドレスを単一の URL に指定し、true に connection オプション **reconnect** を指定します。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}");
```

バグを報告します。

16.4. PYTHON でのクラスターフェイルオーバー

仮想 IP を使用するクラスターで MRG 3 Python クライアントを使用するには、仮想 IP アドレスをブローカーアドレスとして指定します。フェイルオーバーは、クラスターマネージャーによって透過的に処理されます。

仮想 IP アドレスを使用しないクラスターがある場合には、**Connection.establish** またはの呼び出し **reconnect_urls** 時のように **host:port** アドレスの **reconnect=True** 一覧を指定し **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：

```
connection = qpid.messaging.Connection.establish("node1", reconnect=True, reconnect_urls=
["node1", "node2", "node3"], heartbeat=10)
```

バグを報告します。

16.5. JAVA JMS クライアントでのフェイルオーバーの動作

Java JMS クライアントでは、仮想 IP を使用するクラスターがある状態で、仮想 IP アドレスをブローカーアドレスとして指定します。フェイルオーバーは、クラスターマネージャーによって透過的に処理されます。

仮想 IP アドレスを使用しないクラスターがある場合、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 5 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。これは、後方互換性を確保するために提供されます。仮想 IP（および透過的なサーバー側のフェイルオーバー）を使用することが推奨されます。

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。

nofailover

すべての再試行およびフェイルオーバーロジックを無効にします。

<class>

他の値は、**org.apache.qpid.jms.failover.FailoverMethod** インターフェースを実装する必要があるクラス名として解釈されます。

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。

第17章 LOGGING

17.1. 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。

17.2. PYTHON でのロギング

Python クライアントライブラリーは、標準の Python ロギングモジュールを使用したロギングをサポートします。

basicConfig() ロギングメソッドは、すべての警告およびエラーを報告します。

```
from logging import basicConfig
basicConfig()
```

qpidd デーモンを使用すると、必要なロギングレベルを指定できます。たとえば、以下のコードは **DEBUG** レベルでのロギングを有効にします。

```
from qpid.log import enable, DEBUG
enable("qpid.messaging.io", DEBUG)
```

Python ロギングの詳細は、Python のドキュメントを参照してください。Qpid ロギングの詳細は、を実行し **$ pydoc qpid.log**ます。

バグを報告します。

17.3. ランタイム時のロギングレベルの変更

ブローカーのロギングレベルは、再起動せずにランタイム時に変更できます。これは、デバッグ中にロギングの詳細レベルを引き上げ、より低いレベルに戻すのに役立ちます。

Qpid Management Framework Broker オブジェクトには、ロギングレベルを制御する **setLogLevel** メソッドがあります。以下の C++ コードは、このメソッドを呼び出してロギングレベルを設定する方法を示しています。

```
#include <qpid/messaging/Connection.h>
#include <qpid/messaging/Session.h>
#include <qpid/messaging/Sender.h>
#include <qpid/messaging/Receiver.h>
#include <qpid/messaging/Message.h>
#include <qpid/messaging/Address.h>

#include <iostream>

using namespace std;
using namespace qpid::messaging;
using namespace qpid::types;

int main(int argc, char** argv) {
  if (argc < 2) {
    cerr << "Invalid number of parameters, expecting log level (info, trace, warning or so)" << endl;
    return 1;
  }
  string log_level = argv[1];

  Connection connection(argc>2?argv[2]:"localhost:5672");
  connection.open();
  Session session = connection.createSession();
  Sender sender = session.createSender("qmf.default.direct/broker");
  Receiver receiver = session.createReceiver("#reply-queue; {create:always, node:{x-declare:{auto-
delete:true}}}");
  Address responseQueue = receiver.getAddress();

  Message message;
  Variant::Map content;
  Variant::Map OID;
  Variant::Map arguments;

  OID["_object_name"] = "org.apache.qpid.broker:broker:amqp-broker";
  arguments["level"] = log_level;

  content["_object_id"] = OID;
  content["_method_name"] = "setLogLevel";
  content["_arguments"] = arguments;

  message.setContentObject(content);
  message.setReplyTo(responseQueue);
  message.setProperty("x-amqp-0-10.app-id", "qmf2");
  message.setProperty("qmf.opcode", "_method_request");
  message.setContentType("amqp/map");

  sender.send(message, true);

  /* receive a response from the broker & check our request was successfully processed */
  Message response;
  if (receiver.fetch(response,qpid::messaging::Duration(30000)) == true) {
    qpid::types::Variant::Map recv_props = response.getProperties();
```

```
  if (recv_props["qmf.opcode"] == "_method_response")
    std::cout << "Response: OK" << std::endl;
  else if (recv_props["qmf.opcode"] == "_exception")
    std::cerr << "Error: " << response.getContent() << std::endl;
  else
    std::cerr << "Invalid response received!" << std::endl;
}
else
  std::cout << "Timeout: No response received within 30 seconds!" << std::endl;

receiver.close();
sender.close();
session.close();
connection.close();
return 0;
    }
```

1. サンプルコードをファイルに保存し **set_log_level.cpp**ます。
2. コードの Connection URL を変更して、ブローカーに解決します。この時点で、ローカルマシンの 5672 ポートで実行しているブローカーに接続するように設定されます。
3. サンプルコードをコンパイルします。

```
g++ -Wall -lqpidclient -lqpidcommon -lqpidmessaging -lqpidtypes -o set_log_level
set_log_level.cpp
```

4. 補完されたプログラムを使用して、ブローカーのログレベルを変更します。

```
./set_log_level "trace+"
```

5. ログレベルでの変更を確認するには、プログラムを実行するときにサーバーログを監視します。

バグを報告します。

第18章 SECURITY

18.1. QPID が提供するセキュリティー機能

QPID は、認証、ルールベースの承認、暗号化、およびデジタル署名を提供します。

バグを報告します。

18.2. 認証

Qpid は Simple Authentication and Security Layer(SASL)を使用して、ブローカーへのクライアント接続を認証します。SASL は、さまざまな認証方法をサポートするフレームワークです。セキュアなアプリケーションには、CRAM-MD5、DIGEST-MD5、または GSSAPI(Kerberos)メカニズムを使用します。ANONYMOUS メカニズムはセキュアではありません。PLAIN メカニズムは、SSL と併用した場合に限りセキュアになります。

バグを報告します。

18.3. 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。

18.4. KERBEROS 認証の有効化

Kerberos 認証では、プログラムを実行しているユーザーがすでに認証されている場合（たとえば、使用している場合） **kinit**、ユーザー名とパスワードを指定する必要はありません。別の認証形式を使用している場合や、Kerberos で認証されていない場合は、接続オプションとして指定できます。

```
connection.setOption("username", "mick");
connection.setOption("password", "pa$$word");
```

バグを報告します。

18.5. SSL の有効化

暗号化および署名は SSL を使用して行われます（SASL を使用しても可能）。SSL を有効にするには、以下を設定します。 **transport** 接続オプション ***ssl***:

```
connection.setOption("transport", "ssl");
```

バグを報告します。

18.6. C++ クライアント用の SSL クライアント環境変数

表18.1 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
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。

バグを報告します。

第19章 AMQP 0-10 マッピング

19.1. AMQP 0-10 マッピング

送信者またはレシーバーを作成してトリガーしたブローカーとの対話は、指定されたアドレスが解決する内容によって異なります。アドレスでノードタイプが指定されていない場合、クライアントはブローカーにクエリーを実行し、キューを参照するか交換を行います。

キューに送信すると、キュー名はルーティングキーとして設定され、メッセージはデフォルト（または名前なし）の交換に転送されます。交換に送信すると、メッセージがその交換に転送され、ルーティングキーが指定されている場合はメッセージサブジェクトに設定されます。デフォルトのサブジェクトはターゲットアドレスで指定できます。必要に応じて、各メッセージに対してサブジェクトを個別にオーバーライドすることもできます。指定のサブジェクトも、の ***application-headers*** フィールドの **qpid.subject** エントリーとして追加され ***message-properties***ます。

キューから受信する場合、ソースアドレスのサブジェクトは無視されます。クライアントは、問題のキューに対して ***message-subscribe*** リクエストを送信します。これ ***accept-mode*** は、リンクプロパティーの信頼性オプションにより決まります。信頼できないリンクの場合は、***accept-mode*** は none で、信頼できるリンクは明示的なリンクです。キューのデフォルトは信頼性があります。***acquire-mode*** は、mode オプションの値によって決定されます。モードが obtain モードの閲覧に設定されていると ***not-acquired***、そのモードはに設定され ***pre-acquired***ます。***message-subscribe*** コマンドの exclusive および arguments フィールドは、***x-subscribe*** マップを使用して制御できます。

交換から受信を行うと、クライアントはサブスクリプションキューを作成し、交換にバインドします。サブスクリプションキューの引数は、リンクプロパティー内の ***x-declare*** マップを使用して指定できます。信頼性オプションは、他のほとんどのパラメーターを決定します。信頼性が ***unreliable*** 自動削除されると、排他的キューが使用されます。つまり、クライアントまたは接続でメッセージが失われる可能性があります。***exactly-once*** キューは自動削除されるよう設定されていません。サブスクリプションキューの耐久性は、リンクプロパティーの永続オプションにより決まります。バインディングプロセスは、ソースアドレスが解決する交換のタイプによって異なります。

- トピック交換では、サブジェクトが指定されておらず、リンクに定義 ***x-bindings*** されていない場合、サブスクリプションキューは任意のルーティングキーに一致するワイルドカードを使用してバインドされます（そのアドレスに送信されたメッセージはすべて受信されることが予想されます）。ソースアドレスでサブジェクトが指定されている場合には、バインディングキーに使用されます（つまり、ソースアドレスのサブジェクトがワイルドカードを含むバインディングパターンである可能性があることを意味します）。

- ジャンクアウト交換の場合、バインディングキーはマッチングとは関係ありません。エラーアウト交換に対して解決するソースアドレスから作成された受信側は、送信元アドレスに含まれるサブジェクトに関係なく、その交換に送信されたすべてのメッセージを受け取ります。引数をバインドに設定する必要がある場合は、リンクプロパティーの ***x-bindings*** 要素を使用する必要があります。

- 直接交換では、サブジェクトがバインディングキーとして使用されます。サブジェクトを指定しないと、空の文字列がバインディングキーとして使用されます。

- ヘッダー交換では、サブジェクトを指定しない ***x-match*** と、バインディング引数にエントリーが含まれ、他のエントリーも含まないため、すべてのメッセージが一致します。サブジェクトが指定されている場合、バインディング引数には all に ***x-match*** 設定されたエントリーと、ソースアドレスのサブジェクトの値を **qpid.subject** 持つエントリーが含まれます（これは、ソースアドレスのサブジェクトがメッセージサブジェクトと一致する必要があることを意味します）。さらに制御するには、リンクプロパティーの ***x-bindings*** 要素を使用する必要があります。

- XML の交換では、サブジェクトが指定されている場合はバインディングキーとして使用され、XQuery は、その値を持つメッセージに一致するものを定義し **qpid.subject**ます。この場合も、ソースアドレスで指定されたサブジェクトが正確に一致するメッセージのみが受信されます。サブジェクトが指定されていない場合、空の文字列は任意のメッセージに一致する xquery とともにバインディングキーとして使用されます（これは、ルーティングキーの受信として空の文字列を持つメッセージのみを受け取ることを意味します）。さらに制御するには、リンクプロパティーの x-bindings 要素を使用する必要があります。XML 交換に対して解決するソースアドレスには、ルーティングキーに関係なくメッセージを受信する方法がないため、リンクプロパティーの subject または x-bindings 要素が含まれている必要があります。

リンクオプションに x-bindings 一覧がある場合、バインディングはそのリスト内の各要素に対して作成されます。各要素は ***queue***、、、、***exchange key***またはという名前を持つネストされたマップです ***arguments***。キューの値がない場合は、キュー名にアドレスが解決され、暗黙的に解決されます。exchange の値がない場合は、アドレスが暗黙的に解決されます。

以下の表は、Qpid Messaging API メッセージプロパティーが AMQP 0-10 メッセージプロパティーおよび配信プロパティーにマップされる方法を示しています。この表では、 **msg** Qpid Messaging API で定義された Message クラスを参照します。**mp** AMQP 0-10 を参照します。 **message-properties** struct、および **dp** AMQP 0-10 を参照します。 **delivery-properties** 構造。

表19.1 AMQP 0-10 メッセージプロパティーへのマッピング

Python API	C++ API [a]	AMQP 0-10 プロパティー [b]
msg.id	**msg.{get,set}MessageId()**	**mp.message_id**
msg.subject	**msg.{get,set}Subject()**	**mp.application_headers ["qpid.subject"]**
msg.user_id	**msg.{get,set}UserId()**	**mp.user_id**
msg.reply_to	**msg.{get,set}ReplyTo()**	**mp.reply_to**[c]
msg.correlation_id	**msg.{get,set}CorrelationId()**	**mp.correlation_id**
msg.durable	**msg.{get,set}Durable()**	**dp.delivery_mode == delivery_mode.persistent**[d]
msg.priority	**msg.{get,set}Priority()**	**dp.priority**
msg.ttl	**msg.{get,set}Ttl()**	**dp.ttl**
msg.redelivered	**msg.{get,set}Redelivered()**	**dp.redelivered**
msg.properties	**msg.{get,set}Properties()**	**mp.application_headers**
msg.content_type	**msg.{get,set}ContentType()**	**mp.content_type**

Python API	C++ API [a]	AMQP 0-10 プロパティー [b]

[a] C++ Messaging の .NET バインディングは、C++ API で説明されているメッセージおよび配信プロパティーをすべて提供します。

[b] これらのエントリーで ***mp*** は、AMQP message プロパティーを ***dp*** 参照し、AMQP 配信プロパティーを参照します。

[c] reply_to は、プロトコル表現からアドレスに変換されます。

[d] msg.durable は列挙ではなくブール値であることに注意してください。

バグを報告します。

19.2. AMQ 0-10 メッセージプロパティーキー

Qpid Messaging API は特別なメッセージプロパティープロパティーキーを認識し、対応する AMQP 0-10 定義へのマッピングを自動的に提供します。

たとえば、メッセージを送信する場合、プロパティーにエントリーが含まれる場合 ***x-amqp-0-10.app-id***、その値を使用して送信メッセージに ***message-properties.app-id*** プロパティーを設定します。同様に、受信メッセージが ***message-properties.app-id*** 設定されている場合は、メッセージプロパティーキーを使用してその値にアクセスでき ***x-amqp-0-10.app-id*** ます。

同様に、メッセージを送信する場合は、プロパティーにエントリーが含まれる場合 ***x-amqp-0-10.content-encoding***、その値を使用して送信メッセージに ***message-properties.content-encoding*** プロパティーを設定します。同様に、受信メッセージが ***message-properties.content-encoding*** 設定されている場合は、メッセージプロパティーキーを使用してその値にアクセスでき ***x-amqp-0-10.content-encoding*** ます。

受信メッセージのルーティングキー(***delivery-properties.routing-key***)は、***x-amqp-0-10.routing-key*** message プロパティーを介してアクセスできます。

バグを報告します。

19.3. AMQP ルーティングキーおよびメッセージサブジェクト

Red Hat Enterprise Messaging で Qpid Messaging API を使用してメッセージを送信すると、**x-amqp-0-10.routing-key** プロパティーはメッセージサブジェクトの値に設定されますが、1 つの例外があります。

サブジェクトが明示的に設定されているメッセージには、サブジェクトが保持され、AMQP ルーティングキーが送信時にメッセージサブジェクトに設定されます。

メッセージに手動でサブジェクトが設定されていない場合、送信者の宛先アドレスにサブジェクトが含まれる場合、そのサブジェクトは送信者によって設定されます。

たとえば、以下の送信者を使用します。

```
sender = session.sender('amq.topic/SubjectX')
```

以下の 2 つのメッセージがあるとします。

```
msg1 = Message('A message with no subject')
```

```
msg2 = Message('A message with a subject')
msg2.subject = 'SubjectY'
```

msg1 AMQP ルーティングキーは ' ' に設定されており、そのサブジェクト **SubjectX**'**SubjectY**' を **msg2** 保持し、AMQP ルーティングキーが '**SubjectY**' に設定されています。

他の 2 つのケースのみがあります。

1 つ目は、送信先アドレスのサブジェクトがない送信者を介して、サブジェクトのないメッセージが送信される場合です。たとえば、Python の場合は以下のようになります。

```
sender = session('amq.topic')
msg = Message('No subject, and none assigned by the sender')
sender.send(msg)
```

この場合、メッセージは空のサブジェクトと空の AMQP ルーティングキーと共に送信されます。

2 つ目のケースと唯一の例外ケースは、空白のサブジェクトがあり、手動で割り当てられた AMQP ルーティングキーがその宛先アドレスのサブジェクトがない送信元を介して送信されます。たとえば、Python の場合は以下のようになります。

```
sender = session('amq.topic')
msg = Message('No subject, but a manually assigned AMQP routing key')
msg.properties['x-amqp-0-10.routing-key'] = 'amqp-SubjectX'
sender.send(msg)
```

この場合、メッセージは空のサブジェクトと、任意の AMQP ルーティングキーが割り当てられます。

この場合、メッセージは Red Hat Enterprise Messaging トピックの交換でルーティングされないことに注意してください。相互運用性に関するシナリオでは役に立つ **amqp-0-10.routing-key** 場合がありますが、Red Hat Enterprise Messaging では、メッセージのルーティングに使用 **subject** されます。

以下の Python プログラムは、メッセージサブジェクト、送信先アドレスサブジェクト、およびメッセージルーティングキーとメッセージルーティングキーの間の対話に関する様々な変更を示しています。

```
import sys
from qpid.messaging import *

# This program demonstrates that the x-amqp-0-10.routing-key
# (1) is (re)set to the message subject when the message has a subject or
#   is sent via a sender that has a subject
# (2) is not a valid basis for routing in a topic exchange
#   -  the topic exchange will not route a message to a queue

def sendmsg(msg, note = ''):
  global rxplain, rxsubject, txplain, txsubject, ssn, testcount

  msg.properties['sender'] = 'Plain Sender'
  txplain.send(msg)

  msg.properties['sender'] = 'SubjectX Sender'
  txsubject.send(msg)
```

```
if testcount > 0:
  x = raw_input('\nPress Enter for the next test message')
  print '\n==============================================\n'

testcount = testcount + 1
print '\nScenario ' + str(testcount)
print '\nSent message:\n'
subject = 'Blank'
if msg.subject:
  subject = msg.subject
print 'Subject:\t' + subject
routekey = 'Blank'
if 'x-amqp-0-10.routing-key' in msg.properties:
  routekey = msg.properties['x-amqp-0-10.routing-key']
print 'Routing Key:\t' + routekey

msgcount = 0

print '\nThe queue listening for all messages received:'
try:
  while True:
    rxmsg = rxplain.fetch(timeout = 1)
    subject ='Blank'
    if rxmsg.subject:
      subject = rxmsg.subject
    routekey = 'Blank'
    if 'x-amqp-0-10.routing-key' in rxmsg.properties:
      routekey = rxmsg.properties['x-amqp-0-10.routing-key']
    print '\nSubject:\t' + subject
    print 'Routing Key:\t' + routekey
    print 'Sent via:\t' + rxmsg.properties['sender']
    msgcount = 1
    ssn.acknowledge(rxmsg)
except:
  pass

if msgcount == 0:
  print 'Nothing\n'
else:
  msgcount = 0

print '\nThe queue listening for SubjectX messages received:'
try:
  while True:
    rxmsg = rxsubject.fetch(timeout = 1)
    subject ='Blank'
    if rxmsg.subject:
      subject = rxmsg.subject
    routekey = 'Blank'
    if 'x-amqp-0-10.routing-key' in rxmsg.properties:
      routekey = rxmsg.properties['x-amqp-0-10.routing-key']
    print '\nSubject:\t' + subject
    print 'Routing Key:\t' + routekey
    print 'Sent via:\t' + rxmsg.properties['sender']
    msgcount = 1
    ssn.acknowledge(rxmsg)
```

```
  except:
    pass

  if msgcount == 0:
    print 'Nothing\n'

  if note != '':
    print '\nNote: ' + note + "\n"

connection = Connection("localhost:5672")
connection.open()

try:
  ssn = connection.session()

  # we create our receivers here so that queues are created to hold the messages sent
  rxplain = ssn.receiver("amq.topic")
  rxsubject = ssn.receiver("amq.topic/SubjectX")

  txplain = ssn.sender("amq.topic")
  txsubject = ssn.sender("amq.topic/SubjectX")

  testcount = 0

  msg = Message("Plain message, no subject")
  sendmsg(msg, "a subject sender writes the subject and routing key when a message has no
subject, a plain sender does not")

  msg = Message("Message with subject")
  msg.subject = "SubjectX"
  sendmsg(msg, "a plain sender writes the routing key if the message has a subject")

  msg = Message("Message with a different subject")
  msg.subject = "SubjectY"
  sendmsg(msg, "a subject sender does not rewrite a subject, both senders use the message subject
to write routing key")

  msg = Message("Message with routing key")
  msg.properties["x-amqp-0-10.routing-key"] = "SubjectX"
  sendmsg(msg, "a routing key is not sufficient to route to a queue - the match is on subject")

  msg = Message("Message with different routing key")
  msg.properties["x-amqp-0-10.routing-key"] = "SubjectY"
  sendmsg(msg, "the only case where you can manually set a non-blank routing key is a message
with a blank subject, sent via a plain sender")

  msg = Message("Message with different routing key and subject")
  msg.properties["x-amqp-0-10.routing-key"] = "SubjectY"
  msg.subject = "SubjectZ"
  sendmsg(msg, "all messages with subjects and all messages sent via a subject sender have their
routing key rewritten")

finally:
  connection.close()
```

バグを報告します。

19.4. AMQ 0-10 メッセージプロパティーキーのタイムスタンプの使用

タイムスタンプ配信プロパティーが受信メッセージ(***delivery-properties.timestamp***)に設定されていると、***x-amqp-0-10.timestamp*** message プロパティーを使用してタイムスタンプの値にアクセスできます。

関連項目

- 12章*メッセージのタイムスタンプ*

バグを報告します。

第20章 QPID-JAVA AMQP 0-10 クライアントの使用

20.1. JAVA JMS における簡単なメッセージングプログラム

以下のプログラムは、**qpid-java** クライアントを使用してメッセージを送受信する方法を示しています。JMS プログラムは通常 JNDI を使用して、アプリケーションが必要とする接続ファクトリーおよび宛先オブジェクトを取得します。これにより、設定はアプリケーションコード自体から分離されます。

この例では、プロパティーファイルを使用して JNDI コンテキストを作成し、コンテキストを使用して接続ファクトリーの検索、接続の作成および開始、セッションの作成、JNDI コンテキストから宛先の検索を行います。プロデューサーとコンシューマーを作成し、プロデューサーのあるメッセージを送信し、コンシューマーでメッセージを受信します。

```
package org.apache.qpid.example.jmsexample.hello;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.util.Properties;

public class Hello {

    public Hello() {
    }

    public static void main(String[] args) {
     Hello producer = new Hello();
        producer.runTest();
    }

    private void runTest() {
     try {
    Properties properties = new Properties();
        properties.load(this.getClass().getResourceAsStream("hello.properties"));
        Context context = new InitialContext(properties);

        ConnectionFactory connectionFactory
          = (ConnectionFactory) context.lookup("qpidConnectionfactory");
     Connection connection = connectionFactory.createConnection();
     connection.start();

    Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
    Destination destination = (Destination) context.lookup("topicExchange");

    MessageProducer messageProducer = session.createProducer(destination);
    MessageConsumer messageConsumer = session.createConsumer(destination);

     TextMessage message = session.createTextMessage("Hello world!");
   messageProducer.send(message);

   message = (TextMessage)messageConsumer.receive();
   System.out.println(message.getText());

   connection.close();
   context.close();
```

```
    }
    catch (Exception exp) {
     exp.printStackTrace();
    }
  }
 }
```

説明

以下は、プログラムコードの説明です。

```
properties.load(this.getClass().getResourceAsStream("hello.properties"));
```

接続プロパティー、キュー、トピック、アドレス指定オプションを指定する JNDI プロパティーファイルをロードします。

```
Context context = new InitialContext(properties);
```

JNDI 初期コンテキストを作成します。

```
ConnectionFactory connectionFactory
        = (ConnectionFactory) context.lookup("qpidConnectionfactory");
```

Qpid の JMS 接続ファクトリーを作成します。

```
Connection connection = connectionFactory.createConnection();
```

JMS 接続を作成します。

```
connection.start();
```

接続をアクティベートします。

```
Session session=connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
```

セッションを作成します。このセッションはトランザクション(transactions='false')ではなく、メッセージは自動的に承認されます。

```
Destination destination = (Destination) context.lookup("topicExchange");
```

トピック交換の宛先を作成するため、送信元と受信側で使用することができます。

```
MessageProducer messageProducer = session.createProducer(destination);
```

トピックの交換にメッセージを送信するプロデューサーを作成します。

```
MessageConsumer messageConsumer = session.createConsumer(destination);
```

トピックの交換からメッセージを読み取るコンシューマーを作成します。

```
message = (TextMessage)messageConsumer.receive();
```

次の利用可能なメッセージを読み取ります。

```
connection.close();
```

接続、接続によって管理されるすべてのセッション、および各セッションで管理されるすべての送信元と受信側を閉じます。

```
context.close();
```

JNDI コンテキストを閉じます。

hello.properties file

hello.properties ファイルの内容は以下のとおりです。

```
 java.naming.factory.initial
   = org.apache.qpid.jndi.PropertiesFileInitialContextFactory

# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.qpidConnectionfactory
   = amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672'
# destination.[jndiname] = [address_string]
destination.topicExchange = amq.topic
```

バグを報告します。

20.2. AMQP MESSAGING の APACHE QPID JNDI プロパティー

qpid-jms AMQP 1.0 クライアントは以下の JNDI プロパティーをサポートします。

connectionfactory.*<jndiname>*
接続ファクトリーが接続を実行するために使用する Connection URL。

queue.*<jndiname>*
JMS キュー。Apache Qpid で **amq.direct** 交換として実装されます。

topic.*<jndiname>*
JMS トピック。Apache Qpid で **amq.topic** 交換として実装されます。

destination.*<jndiname>*
アドレス文字列（または、以前の実装と後方互換性のためにバインディング URL）を使用して、amq 宛先、キュー、トピック、およびヘッダーの照合をすべて定義するために使用できます。

バグを報告します。

20.3. APACHE QPID の JNDI プロパティー

Apache Qpid は、JMS 接続および宛先の指定に使用できる JNDI プロパティーを定義します。以下は JNDI プロパティーファイルの例です。

```
java.naming.factory.initial
```

```
  = org.apache.qpid.jndi.PropertiesFileInitialContextFactory

# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.qpidConnectionfactory
  = amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672'
# destination.[jndiname] = [address_string]
destination.topicExchange = amq.topic
```

バグを報告します。

20.4. MRG 3 での永続的サブスクリプションキュー

MRG 3 では、**qpid-java** クライアントに永続サブスクリプションキューの名前を指定する必要があります。

これは、MRG 2 で機能する以下のコマンドが MRG 3 で例外を報告することを意味します。

```
# java -cp ${CLASSPATH} org.apache.qpid.example.Drain  "amq.topic/some_subject;{ link: { durable: true } }"

javax.jms.JMSException: Error registering consumer: org.apache.qpid.AMQException: You cannot mark a subscription queue as durable without providing a name for the link.
```

クライアント例外を回避するには、リンクに名前を付けます。例：

```
# java -cp ${CLASSPATH} org.apache.qpid.example.Drain  "amq.topic/some_subject;{ link: { name: some_name, durable: true } }"
```

バグを報告します。

20.5. 接続 URL

JNDI プロパティーでは、Connection URL は接続のプロパティーを指定します。Connection URL の形式は以下のとおりです。

```
amqp://[<user>:<pass>@][<clientid>]<virtualhost>[?<option>='<value>'[&<option>='<value>']]
```

たとえば、以下の接続 URL はユーザー名、パスワード、クライアント ID、仮想ホスト("test")、単一のブローカーを持つブローカーリスト、およびポート 5672 を使用してホスト名 localhost を持つ TCP ホストを指定します。

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

Apache Qpid は、接続 URL の以下のプロパティーに対応します。

表20.1 接続 URL プロパティー

オプション	type	description
brokerlist	「ブローカーリスト URL」	このコネクションに使用するブローカー。現在のリリースでは、正確に 1 つのブローカーを指定する必要があります。

オプション	type	description
max_prefetch	整数	宛先ごとの事前にフェッチされたメッセージの最大数。
sync_publish	{'persistent' | 'transient' | 'all' | ''}	sync コマンドは、永続メッセージまたは一時的なメッセージが毎回送信され、受信が確実に行われます。 **persistent** 永続メッセージに対してこの動作を設定します。 **transient** この動作を一時的なメッセージのみに設定します。 **all** 両方のタイプのメッセージを同期しますが、デフォルトの動作 **''** も同じ効果を持ちます。
sync_ack	ブール値	sync コマンドは、受信を確認するために毎回承認後に送信されます。
use_legacy_map_msg_format	ブール値	JMS Map メッセージを使用し、0.7 リリースよりも古い JMS クライアントで新しいクライアントをデプロイする場合、これを設定して古いクライアントがマップメッセージエンコーディング **true** を把握できるようにする必要があります。

オプション	type	description
failover	**{'roundrobin' | 'failover_exchange' | 'singlebroker' | 'nofailover' | '<class>'}**	• **roundrobin** ブローカーリストで指定される各ブローカーを試行します。 • **failover_exchange** ブローカー URL に指定された初期ブローカーに接続し、フェイルオーバーの交換を介してメンバーシップの更新を受け取ります。 • **singlebroker** 初期ブローカーのみに接続し、フェイルオーバーをサポートしません。 • **nofailover** すべての再試行およびフェイルオーバーロジックを無効にします。 • その他の値は、org.apache.qpid.jms.failover.FailoverMethod インターフェースを実装する必要があるクラス名として解釈されます。
ssl	ブール値	すべてのブローカー接続に SSL **ssl='true'**を使用します。brokerlist エントリーのブローカーごとの設定をオーバーライドします。指定されていない場合、指定した各ブローカーの brokerlist エントリーを使用して、SSL が使用されるかどうかを判断します。

ブローカーリスト URL

ブローカーリストは、この形式の URL を使用して指定されます。

```
brokerlist=<transport>://<host>[:<port>](?<param>=<value>)?(&<param>=<value>)*
```

たとえば、典型的なブローカー一覧 URL は以下のようになります。

```
brokerlist='tcp://localhost:5672'
```

ブローカーリストには複数のブローカーアドレスを含めることができます。存在する場合は、利用可能なリストの最初のブローカーにコネクションが作成されます。通常、ブローカーがダウンした場合にアプリケーションがフェールオーバーできるため、複数のブローカーを使用する場合はフェイルオーバーの交換を使用することが推奨されます。

例20.1 ブローカーリスト

ブローカーリストは、セキュリティーオプションなどのブローカーへの接続時に使用されるプロパティーを指定できます。このブローカーリストは、GSSAPI を使用した Kerberos 接続のオプションを指定します。

```
amqp://guest:guest@test/test?sync_ack='true'
  &brokerlist='tcp://ip1:5672?sasl_mechs='GSSAPI''
```

このブローカーリストは SSL オプションを指定します。

```
amqp://guest:guest@test/test?sync_ack='true'
  &brokerlist='tcp://ip1:5672?ssl='true'&ssl_cert_alias='cert1''
```

このブローカーリストは、connectdelay を使用して 2 つのブローカーを指定し、ブローカーオプションを再試行します。また、フェイルオーバー接続 URL プロパティーも示します。

```
amqp://guest:guest@/test?failover='roundrobin?cyclecount='2''
  &brokerlist='tcp://ip1:5672?retries='5'&connectdelay='2000';tcp://ip2:5672?
retries='5'&connectdelay='2000''
```

以下のブローカー一覧の URL オプションがサポートされます。

表20.2 Broker List URL オプション

オプション	type	description
idle_timeout	整数	idle_timeout メッセージの頻度（秒単位）
sasl_mechs	--	セキュアなアプリケーションの場合、提案 **CRAM-MD5 DIGEST-MD5**、またはです **GSSAPI**。この **ANONYMOUS** メソッドは安全ではありません。この **PLAIN** 方法は、SSL と併用した場合に限り安全です。Kerberos を設定 **sasl_mechs** するに **GSSAPI sasl_protocol** は、qpidd ブローカーのプリンシパル（例：）を設定し **qpidd/**、SASL サーバーのホスト（例：）に設定 **sasl_server** する必要があります **sasl.com**。SASL External は SSL 認定を使用してサポートされます（例：）。**ssl='true'&sasl_mechs='EXTERNAL'**

オプション	type	description
sasl_encryption	ブール値	if **sasl_encryption='true'**、JMS クライアントは、GSSAPI を使用して接続を暗号化するためにブローカーでセキュリティー層をネゴシエートしようとします。これを行うには、GSSAPI がとして選択される必要があることに注意してください **sasl_mech**。
ssl	ブール値	if **ssl**='**true**' の場合、JMS クライアントは SSL を使用して接続を暗号化します。
tcp_nodelay	ブール値	if **tcp_nodelay**='**true**' の場合、TCP パケットのバッチ処理が無効になります。
sasl_protocol	--	Kerberos にのみ使用されます。たとえば、qpidd ブローカーのプリンシパルに設定 **sasl_protocol** する必要があります。 **qpidd/**
sasl_server	--	Kerberos を設定 **sasl_mechs** するには **GSSAPI**、に SASL サーバーのホストを設定 **sasl_server** する必要があります（例：）**sasl.com**。
trust_store	string	Kerberos トラストストアへのパス
trust_store_password	string	Kerberos トラストストアのパスワード
key_store	string	Kerberos キーストアへのパス
key_store_password	string	Kerberos キーストアのパスワード
ssl_verify_hostname	ブール値	SSL を使用する場合、ブローカー URL の "=" を使用してホスト名の検証を有効**ssl_verify_hostnametrue**にできます。
ssl_cert_alias	string	複数の証明書がキーストアにある場合、エイリアスを使用して正しい証明書を抽出します。
retries	整数	Broker リストの各ブローカーへの接続を再試行する回数。デフォルトは 1 です。
connectdelay	整数	再接続を試みるまで待機する時間（ミリ秒単位）。デフォルトは 0 です。

オプション	type	description
connecttimeout	整数	ソケット接続が正常に実行されるまで待機する時間（ミリ秒単位）。値が 0 の場合は無限のタイムアウトを表します。つまり、接続の試行は確立されるまでブロックするか、エラーが発生します。デフォルトは 30000 です。
tcp_nodelay	ブール値	TCP パケット **tcp_nodelay='true'**のバッチ処理が無効になっている場合。Qpid 0.14 以降、デフォルトで true に設定されます。

バグを報告します。

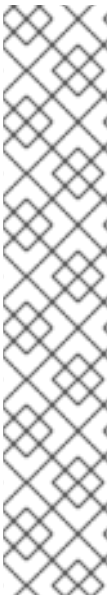
20.6. JAVA JMS メッセージプロパティー

以下の表は、Qpid Messaging API メッセージプロパティーが AMQP 0.10 および 1.0 メッセージプロパティーおよび配信プロパティーにマップされる方法を示しています。

表20.3 JMS ヘッダーの AMQP フィールドへのマッピング

JMS ヘッダー名	AMQP Identifier	AMQP フィールド	AMQP Section	注記
JMSCorrelationID	**correlation_id**	**correlation-id**	**properties**	
JMSDeliveryMode	**delivery_mode**	**durable**	**header**	計算された値：**[durable ? 'PERSISTENT' : 'NON_PERSISTENT']**
JMSDestination	**to**	**to**	**properties**	
JMSExpiration	**absolute_expiry_time**	**absolute-expiry-time**	**properties**	
JMSMessageID	**message_id**	**message-id**	**properties**	
JMSPriority	**priority**	**priority**	**header**	
JMSRedelivered	**redelivered**	**delivery-count**	**header**	計算値：**delivery-count**> 0
JMSReplyTo	**reply_to**	**reply-to**	**properties**	

JMS ヘッダー名	AMQP Identifier	AMQP フィールド	AMQP Section	注記
JMSTimestamp	**creation_time**	**creation-time**	**properties**	
JMSType	**subject**	**subject**	**properties**	



注記

JMS 仕様に従って、メッセージヘッダーフィールド参照は以下に制限されます。

- ***JMSDeliveryMode***
- ***JMSPriority***
- ***JMSMessageID***
- ***JMSTimestamp***
- ***JMSCorrelationID***
- ***JMSType***

JMS のみを使用する場合、これらのフィールドのみはセレクターで厳密に有効です。

バグを報告します。

20.7. JMS MAPMESSAGE タイプ

Qpid-java は、メッセージ内のマップのサポートを提供する Java JMS **MapMessage** インターフェースをサポートします。以下のコードは、Java JMS **MapMessage** で送信方法を示しています。

例20.2 Java JMS MapMessage の送信

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.MapMessage;
import javax.jms.MessageProducer;
import javax.jms.Session;

import org.apache.qpid.client.AMQAnyDestination;
import org.apache.qpid.client.AMQConnection;

import edu.emory.mathcs.backport.java.util.Arrays;

// !!! SNIP !!!

MessageProducer producer = session.createProducer(queue);
```

```
MapMessage m = session.createMapMessage();
m.setIntProperty("Id", 987654321);
m.setStringProperty("name", "Widget");
m.setDoubleProperty("price", 0.99);

List<String> colors = new ArrayList<String>();
colors.add("red");
colors.add("green");
colors.add("white");
m.setObject("colours", colors);

Map<String,Double> dimensions = new HashMap<String,Double>();
dimensions.put("length",10.2);
dimensions.put("width",5.1);
dimensions.put("depth",2.0);
m.setObject("dimensions",dimensions);

List<List<Integer>> parts = new ArrayList<List<Integer>>();
parts.add(Arrays.asList(new Integer[] {1,2,5}));
parts.add(Arrays.asList(new Integer[] {8,2,5}));
m.setObject("parts", parts);

Map<String,Object> specs = new HashMap<String,Object>();
specs.put("colours", colors);
specs.put("dimensions", dimensions);
specs.put("parts", parts);
m.setObject("specs",specs);

producer.send(m);
```

以下の表は、で送信できるデータタイプを示しています。 **MapMessage**および、Python または C++ でクライアントが受信する対応するデータタイプ。

表20.4 マップの Java データタイプ

Java Data Type	? Python	? C++
boolean	**bool**	**bool**
short	**int | long**	**int16**
int	**int | long**	**int32**
long	**int | long**	**int64**
float	**float**	**float**
double	**float**	**double**
java.lang.String	**unicode**	**std::string**

Java Data Type	? Python	? C++
java.util.UUID	**uuid**	**qpid::types::Uuid**
java.util.Map[a]	**dict**	**Variant::Map**
java.util.List	**list**	**Variant::List**

[a] Qpid では、マップはネストできます。これは、JMS 仕様で必要な機能を超えています。

バグを報告します。

20.8. JMS LISTMESSAGE

JMS **ListMessage** タイプはリストの送信に利用できます。

レシーバー側では、List メッセージは 3 つのインターフェースを介して公開されます。

1. **javax.jms.StreamMessage**
2. **javax.jms.MapMessage**
3. **org.apache.qpid.jms.ListMessage**

送信側では、List メッセージを送信する方法は 2 つあります。

1. **org.apache.qpid.jms.ListMessage** - でこれを作成 **createListMessage()** し **org.apache.qpid.jms.Session**ます。

   例：

   ```
   ListMessage msg =  ((org.apache.qpid.jms.Session)ssn).createListMessage();
   ```

2. 作成 **-Dqpid.use_legacy_stream_message=false** するストリームメッセージを設定すると、リストメッセージとしてエンコードされます。

   例：

   ```
   StreamMessage msg = jmsSession.createStreamMessage();
   ```

コードの例については、このサンプルコード を参照してください。

バグを報告します。

20.9. JMS クライアントロギング

qpid-java クライアントロギングは、Simple Logging Facade for Java(SLF4J)を使用して処理されます。Slf4J は、log4j や JDK 1.4 ロギングなどの他のロギングシステムに委譲するファケードです。

log4j バインディングを使用する場合は、のログレベルを設定し **org.apache.qpid**ます。それ以外の場合は log4j はデフォルト ***DEBUG*** で、ログ過剰によりパフォーマンスが大幅に低下します。実稼働環境で推奨されるロギングレベルはです ***WARN***。

以下の例は、log4j バインディングを使用して SLF4J のクライアントロギングを設定するために使用されるロギングプロパティーを示しています。これらのプロパティーは **log4j.properties** ファイルに配置して配置することも **CLASSPATH**、***-Dlog4j.configuration*** プロパティーを使用して明示的に設定できます。

例20.3 Log4j ロギングプロパティー

```
log4j.logger.org.apache.qpid=WARN, console
log4j.additivity.org.apache.qpid=false

log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.Threshold=all
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%t %d %p [%c{4}] %m%n
```

バグを報告します。

20.10. AMQP 0-10 JMS クライアントの設定

20.10.1. 設定方法および粒度

qpid-java クライアントでは、複数の設定オプションを使用して、さまざまなレベルで動作をカスタマイズできます。

- JVM 引数を使用した JVM レベル： JVM 内で作成されたすべての接続、セッション、コンシューマー、およびプロデューサーに影響します。

  例： **-dmax_prefetch=1000** プロパティーは、使用するメッセージのクレジットを指定します。

- 接続またはブローカープロパティーを使用した接続レベル： その接続によって作成された各接続およびセッション、コンシューマー、プロデューサーに影響します。

  例： **amqp://guest:guest@test/test?max_prefetch='1000' &brokerlist='tcp://localhost:5672'** プロパティーは、使用するメッセージのクレジットを指定します。これにより、JVM 引数で指定したすべての値が上書きされ **max_prefetch**ます。

- アドレス指定オプションを使用した宛先レベル： 各宛先を使用して作成されたプロデューサーとコンシューマーに影響します。

  例： capacity オプション **my-queue; {create: always, link:{capacity: 10}}** は、使用するメッセージのクレジットを指定します。これにより、接続レベルの設定が上書きされます。

バグを報告します。

20.10.2. qpid-java JVM 引数

表20.5 接続動作の設定オプション

プロパティー名	type	デフォルト値	description
qpid.amqp.version	string	0-10	使用される AMQP バージョンを設定します。現在 0-8、0-9、0-91、および 0-10 をサポートします。クライアントは指定されたバージョンでネゴシエーションを開始し、ブローカーが指定のバージョンに対応していない場合のみ、ダウンスケールをネゴシエートします。
qpid.heartbeat	int	120（秒）	ハートビートの間隔（秒単位）。連続しない 2 つのハートビートにより、接続がタイムアウトします。これは、接続ごとに設定することもできます。
ignore_setclientID	ブール値	false	クライアント ID が接続 URL で指定された場合、そのクライアント ID が使用され、使用されない場合は ID が生成されます。ID が生成された Qpid 後に指定された場合は、例外が発生します。このプロパティーを 'true' に設定するとチェックを無効にし、クライアント ID をいつでも設定できます。

表20.6 セッション動作の設定オプション

プロパティー名	type	デフォルト値	description
qpid.session.command_limit	int	65536	承認されていないコマンドの数を制限します。
qpid.session.byte_limit	int	1048576	承認されていないコマンドの数をバイト単位で制限します。

プロパティー名	type	デフォルト値	description
qpid.use_legacy_map_message	ブール値	false	古いマップメッセージエンコーディングを使用します。デフォルトでは、マップメッセージは 0-10 マップエンコーディングを使用してエンコードされます。これは、接続ごとに設定することもできます。
qpid.jms.daemon.dispatcher	ブール値	false	セッションディスパッチャースレッドがデーモンスレッドかどうかを制御します。このシステムプロパティーを true に設定すると、セッションディスパッチャースレッドがデーモンスレッドとして作成されます。この設定はバージョン 0.16 で導入されています。

表20.7 コンシューマー動作の設定オプション

プロパティー名	type	デフォルト値	description
max_prefetch	int	500	クレジットに対するメッセージの最大数。接続ごとや送信先ごとに設定することもできます。
qpid.session.max_ack_delay	long	1000 (ms)	***AUTO_ACK*** およびを使用する際にバッファーでメッセージをフラッシュするタイマー間隔 ***DUPS_OK***。
sync_ack	ブール値	false	設定した場合、各メッセージは同期的に承認されます。***AUTO_ACK*** mode を使用する場合は、これを「true」に設定します。接続ごとに設定することもできます。

表20.8 プロデューサー動作のための設定オプション

プロパティー名	type	デフォルト値	description
sync_publish	string	-	メッセージを同期的に送信します。有効な値は ***persistent***、、***transient***、です ***all***。接続ごとに設定することもできます。

表20.9 スレッド用の設定オプション

プロパティー名	type	デフォルト値	description
qpid.thread_factory	string	**org.apache.qpid.thread.DefaultThreadFactory**	使用するスレッドファクトリーを指定します。リアルタイム JVM を使用する場合は、に設定し**org.apache.qpid.thread.RealtimeThreadFactory**ます。
qpid.rt_thread_priority	int	20	リアルタイムスレッドファクトリーによって作成されたリアルタイムスレッドの優先度(1-99)を指定します。

表20.10 I/O の設定オプション

プロパティー名	type	デフォルト値	description
qpid.transport	string	**org.apache.qpid.transport.network.io.IoNetworkTransport**	使用するトランスポートの実装。**org.apache.qpid.transport.network.NetworkTransport** トランスポートメカニズムを指定することもできます。
qpid.sync_op_timeout	long	60000（ミリ秒）	同期操作が完了するまで待機する時間。古いクライアントとの互換性を維持するには、を使用し**amqj.default_syncwrite_timeout**ます。

プロパティー名	type	デフォルト値	description
qpid.tcp_nodelay	ブール値	true	基盤のソケットの TCP_NODELAY プロパティーを設定します。 これは、Connection URL オプションを使用して接続ごとに設定することもできます。 古いクライアントとの互換性のために、シノニム **amqj.tcp_nodelay** がサポートされます。
qpid.send_buffer_size	整数	65535	基盤のソケットの SO_SNDBUF プロパティーを設定します。 古いクライアントとの互換性のために、シノニム **amqj.sendBufferSize** がサポートされます。
qpid.receive_buffer_size	整数	65535	基盤のソケットの SO_RCVBUF プロパティーを設定します。 古いクライアントとの互換性のために、シノニム **amqj.receiveBufferSize** がサポートされます。
qpid.failover_method_timeout	long	60000	フェイルオーバー中は、接続の再確立を試みる各試行のタイムアウトとなります。再接続の試行がタイムアウトを超えると、フェイルオーバープロセス全体が中止されます。 AMQP 0-8/0-9/0-9-1 クライアントにのみ適用されます。

表20.11 セキュリティーの設定オプション

プロパティー名	type	デフォルト値	description

プロパティー名	type	デフォルト値	description
qpid.sasl_mechs	string	PLAIN	使用される SASL メカニズム。複数の場合はコンマで区切って指定できます。サポートされる値は PLAIN、GSSAPI、および EXTERNAL です。
qpid.sasl_protocol	string	AMQP	GSSAPI を SASL メカニズムとして使用する場合は、qpidd ブローカーのプリンシパルに設定する **sasl_protocol** 必要があります。
qpid.sasl_server_name	string	localhost	GSSAPI を SASL メカニズムとして使用する場合は、SASL サーバーのホストに設定する **sasl_server** 必要があります。

表20.12 SASL メカニズムとしての GSSAPI の JVM プロパティー

プロパティー名	type	デフォルト値	description
javax.security.auth.useSubjectCredsOnly	ブール値	true	'false' に設定すると、SASL GASSPI 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。
java.security.auth.login.config	string	-	JASS 設定ファイルを指定します。

表20.13 SSL 接続の設定オプション

プロパティー名	type	デフォルト値	description
qpid.ssl_timeout	long	60000	操作の待機時に Java SSL エンジンによって使用されるタイムアウト値。

プロパティー名	type	デフォルト値	description
qpid.ssl.KeyManagerFactory.algorithm	string	-	キーマネージャーファクトリーアルゴリズム名。設定されていない場合、デフォルトで Java ランタイム呼び出し KeyManagerFactory.getDefaultAlgorithm（）から返された値に設定されます。 古いクライアントとの互換性を維持するために、同じ名前 qpid.ssl.keyStoreCertType がサポートされます。
qpid.ssl.TrustManagerFactory.algorithm	string	-	トラストマネージャーファクトリーアルゴリズム名。設定されていない場合、デフォルト値は Java ランタイム呼び出し TrustManagerFactory.getDefaultAlgorithm（）から返された値に設定されます。 古いクライアントとの互換性を維持するために、同じ名前 qpid.ssl.trustStoreCertType がサポートされます。

表20.14 SSL 接続の JVM プロパティー

プロパティー名	type	デフォルト値	description
javax.net.ssl.keyStore	string	JVM のデフォルト	キーストアパスを指定します。
javax.net.ssl.keyStorePassword	string	JVM のデフォルト	キーストアのパスワードを指定します。
javax.net.ssl.trustStore	string	JVM のデフォルト	トラストストアパスを指定します。
javax.net.ssl.trustStorePassword	string	JVM のデフォルト	トラストストアのパスワードを指定します。

バグを報告します。

20.11. JAVA MESSAGE SERVICE WITH FILTERS

20.11.1. No Local filter（ローカルフィルターなし）

```
<type name="no-local-filter" class="composite" source="list" provides="filter">
  <descriptor name="apache.org:no-local-filter:list" code="0x0000468C:0x00000003"/>
</type>
```

メッセージは、元のメッセージがソースから現在受信されている別の接続でソースのコンテナーに送信された場合に限り、simple-no-local-filter によって受け入れられます。

バグを報告します。

20.11.2. セレクターフィルター

```
<type name="selector-filter" class="restricted" source="string" provides="filter">
  <descriptor name="apache.org:selector-filter:string" code="0x0000468C:0x00000004"/>
</type>
```

qpid-java JMS "selector" は、メッセージをフィルタリングするための SQL 構文を定義します。「ヘッダー」および「プロパティー」の値に基づいてセレクターフィルターが行われます。selector-filter は JMS で定義されたセレクターを使用しますが、JMS ヘッダーの名前が同等の AMQP ヘッダーに変換されます。定義された JMS ヘッダーは AMQP message セクション内の同等のフィールドにマップできます。

ヘッダーの全一覧はに含まれてい 「Java JMS メッセージプロパティー」 ます。

ネットワーク上のセレクター文字列をエンコードする場合、これらの JMS ヘッダー名 ***field_name*** は上記のテーブルで名前 **amqp.*field_name*** が付けられた AMQP 1.0 フィールドで、ハイフンがアンダースコアに置き換えられます。たとえば、セレクター： は以下のようにネットワーク経由で **JMSCorrelationID = 'abc' AND color = 'blue' AND weight > 2500** 転送されます。
amqp.correlation_id = 'abc' AND color = 'blue' AND weight > 2500

JMS メッセージの「プロパティー」は AMQP application-properties セクションと同じです。そのため、メッセージセレクターのプロパティー Foo への参照は、application-properties セクションのキー「Foo」（存在する場合）に関連付けられた値として評価されます。

JMS セレクターのオペランドは、JMS 内で使用可能なタイプの観点で定義されます。アプリケーションプロパティーセクションに対して評価されると、そのセクション内の値は以下のタイプマッピングに従って評価されます。

表20.15 AMQP タイプと JMS タイプへのマッピング

AMQP タイプ	JMS セレクタータイプ
null	null
ブール値	ブール値
ubyte	short
ushort	int

AMQP タイプ	JMS セレクタータイプ
uint	long
ulong	long
byte	byte
short	short
int	int
long	long
float	float
double	double
decimal32	double
decimal64	double
decimal128	double
char	char
timestamp	long
uuid	byte[16]
バイナリー	byte[]
string	string
記号	string

バグを報告します。

第21章 QPID-JMS AMQP 1.0 クライアントの使用

21.1. QPID AMQP 1.0 JMS クライアント設定

本章では、JNDI の設定および作成方法、関連する設定の **InitialContext**構文、定義時に設定できるさまざまな URI オプションなど、**qpid-jms** クライアントのさまざまな設定オプションについて説明し **ConnectionFactory**ます。

アプリケーションは **InitialContext**、JNDI から取得した自体を使用して **InitialContextFactory**、のような JMS オブジェクトを検索し **ConnectionFactory**ます。Qpid JMS クライアントは、**org.apache.qpid.jms.jndi.JmsInitialContextFactory InitialContextFactory クラスの実装を提供します**。これは、3 つの主な方法で設定および使用できます。

Java Classpath 上の **jndi.properties ファイルを介して行います。**

Classpath で名前が付けられたファイル **jndi.properties** を追加して、***java.naming.factory.initial*** プロパティーを value に設定します。**org.apache.qpid.jms.jndi.JmsInitialContextFactory**InitialContext オブジェクトをインスタンス化すると、Qpid **InitialContextFactory** 実装が検出されます。

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

コンテキストに含める特定の Queue オブジェクトおよび Topic オブジェクトは **ConnectionFactory、jndi.properties** ファイル内で直接プロパティー（以下の構文）または ***java.naming.provider.url*** プロパティーを **jndi.properties** 使用して参照される別のファイルで設定されます。

システムプロパティーを使用した場合。

java.naming.factory.initial システムプロパティーを value に設定 **org.apache.qpid.jms.jndi.JmsInitialContextFactory**InitialContext オブジェクトをインスタンス化すると、Qpid **InitialContextFactory** 実装が検出されます。

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

含めるコンテキストの特定の **ConnectionFactory**Queue オブジェクトおよび Topic オブジェクトは、***java.naming.provider.url*** システムプロパティーを使用して渡されるファイルのプロパティーとして設定されます。これらのプロパティーの構文は以下のとおりです。

環境の Hashtable をプログラムで使用。

InitialContext は、作成中に環境を渡すことで直接設定できます。

```
Hashtable<Object, Object> env = new Hashtable<Object, Object>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.apache.qpid.jms.jndi.JmsInitialContextFactory");
javax.naming.Context context = new javax.naming.InitialContext(env);
```

コンテキストに含める特定の Queue オブジェクトおよび Topic オブジェクトは **ConnectionFactory**、環境内で直接 Hashtable または Hashtable 環境内で参照される別のファイルで、プロパティー（以下の構文）として設定され ***java.naming.provider.url*** ます。

プロパティーファイルまたは環境 Hashtable で使用されるプロパティー構文は以下のとおりです。

表21.1 プロパティーの構文

property	構文
ConnectionFactory	**connectionfactory.lookupName = URI**
Queue	**queue.lookupName = queueName**
トピック	**topic.lookupName = topicName**

たとえば、ConnectionFactoryキュー、および Topic の定義に使用される以下のプロパティーについて考えてみましょう。

```
connectionfactory.myFactoryLookup = amqp://localhost:5672
queue.myQueueLookup = queueA
topic.myTopicLookup = topicA
```

これらのオブジェクトは、以下のようにコンテキストから検索できます。

```
ConnectionFactory factory = (ConnectionFactory) context.lookup("myFactoryLookup");
Queue queue = (Queue) context.lookup("myQueueLookup");
Topic topic = (Topic) context.lookup("myTopicLookup");
```

バグを報告します。

21.2. QPID AMQP 1.0 JMS クライアント接続 URL

qpid-jms クライアントの接続 URI の基本的な形式は以下のとおりです。

```
amqp://hostname:port[?option=value[&option2=value...]]
```

クライアントは、URI を使用して多数の異なる設定で設定することができます。これらは ConnectionFactory、以下の表で説明されています。

以下のオプションは Connection、、、Session MessageConsumer およびなどの JMS オブジェクトの動作に適用され MessageProducerます。

表21.2 JMS 設定オプション

オプション	description
jms.username	接続の認証に使用されるユーザー名の値
jms.password	接続の認証に使用されるパスワードの値
jms.clientID	コネクションに適用される ClientID 値。
jms.forceAsyncSend	からすべてのメッセージが送信されたかどうかを設定します。 **MessageProducer** トランザクション内のメッセージや非永続的なメッセージなどの認定されるメッセージのみを非同期に送信します。

オプション	description
jms.alwaysSyncSend	すべての非同期送信条件をオーバーライドし、常にすべてのメッセージを送信する **MessageProducer** 同期的に。
jms.sendAcksAsync	すべてのメッセージの確認応答が非同期的に送信されるようにします。
jms.localMessageExpiry	かどうかを制御します。 **MessageConsumer** インスタンスは有効期限の切れたメッセージをローカルでフィルターするか、配信します。デフォルトでは、この値は true に設定され、期限切れのメッセージはフィルターされます。
jms.localMessagePriority	事前フェッチメッセージを有効にした場合には、指定の Message の優先度の値に基づいてローカルで順番が変更されます。デフォルトは false です。
jms.validatePropertyNames	メッセージプロパティー名が有効な Java 識別子として検証される必要がある場合。デフォルトはです **true**。
jms.queuePrefix	JMS セッションから作成されたキューの名前に追加される任意の接頭辞値。
jms.topicPrefix	JMS セッションから作成された Topic の名前に追加される任意の接頭辞の値です。
jms.closeTimeout	接続が閉じられてから返されるまでの待機時間を制御するタイムアウト値。デフォルトでは、クライアントは通常のクローズ完了イベントまで 15 秒待機します。
jms.connectTimeout	クライアントが接続確立で待機してからエラーを返す時間を制御するタイムアウト値。デフォルトでは、クライアントは接続が確立されるまで 15 秒間待機した後、失敗します。
jms.clientIDPrefix	新しい Connection が JMS 用に作成されるときに生成されるクライアント ID 値に使用されるオプションの接頭辞値 **ConnectionFactory**。デフォルトのプレフィックスはです **ID:**。
jms.connectionIDPrefix	JMS に新しい Connection が作成されたときに生成された Connection ID の値に使用されるオプションの接頭辞値 **ConnectionFactory**。この接続 ID は、JMS Connection オブジェクトから一部の情報をログに記録する際に使用されます。これにより、設定可能な接頭辞によりログの階層化が容易になります。デフォルトのプレフィックスはです **ID:**。

以下の値は、リモートピアがクライアントに送信し、各コンシューマーインスタンスの事前フェッチバッファーに保持できるメッセージ数を制御します。

表21.3 事前フェッチオプション

オプション	description
jms.prefetchPolicy.queuePrefetch	デフォルトはです。 **1000**
jms.prefetchPolicy.topicPrefetch	デフォルトはです。 **1000**
jms.prefetchPolicy.queueBrowserPrefetch	デフォルトはです。 **1000**
jms.prefetchPolicy.durableTopicPrefetch	デフォルトはです。 **1000**
jms.prefetchPolicy.all	すべての事前フェッチ値を 1 度に設定するために使用されます。

RedeliveryPolicy パラメーターは、再配信されたメッセージをクライアント上でどのように処理するかを制御します。

表21.4 再配信オプション

オプション	description
jms.redeliveryPolicy.maxRedeliveries	再配信される回数に基づいて受信メッセージを拒否するタイミングを制御します。デフォルト値は無効(**-1**)です。値がゼロ(**0**)の場合は、メッセージの再配信が許可されないことが示されます。5(**5**)を指定すると、メッセージを 5 回再配信できます。

プレーン TCP を使用してリモートに接続する場合、オプションは基盤のソケットの動作を設定します。これらのオプションは、たとえば以下のような他の設定オプションと共に接続 URI に追加されます。

```
amqp://localhost:5672?jms.clientID=foo&transport.connectTimeout=30000
```

表21.5 TCP トランスポートオプション

オプション	description
transport.sendBufferSize	デフォルトはです。 **64k**
transport.receiveBufferSize	デフォルトはです。 **64k**
transport.trafficClass	デフォルトはです。 **0**
transport.connectTimeout	デフォルトは **60** seconds です。
transport.soTimeout	デフォルトはです。 **-1**
transport.soLinger	デフォルトはです。 **-1**
transport.tcpKeepAlive	デフォルトはです。 **false**

オプション	description
transport.tcpNoDelay	デフォルトはです。 **true**

SSL Transport は TCP トランスポートを拡張し、amqps URI スキームを使用して有効化されます。SSL Transport は TCP ベースの Transport の機能を拡張するため、すべての TCP トランスポートオプションは SSL トランスポート URI で有効です。

簡単な SSL ベースのクライアント URI を以下に示します。

```
amqps://localhost:5673
```

表21.6 SSL トランスポートオプション

オプション	description
transport.keyStoreLocation	デフォルトはシステムプロパティーから読み取られます。 ***javax.net.ssl.keyStore***
transport.keyStorePassword	デフォルトはシステムプロパティーから読み取られます。 ***javax.net.ssl.keyStorePassword***
transport.trustStoreLocation	デフォルトはシステムプロパティーから読み取られます。 ***javax.net.ssl.trustStore***
transport.trustStorePassword	デフォルトはシステムプロパティーから読み取られます。 ***javax.net.ssl.keyStorePassword***
transport.storeType	使用されるトラストストアのタイプ。デフォルトはです **JKS**。
transport.contextProtocol	SSLContext の取得時に使用されるプロトコル引数。デフォルトはです **TLS**。
transport.enabledCipherSuites	有効にする暗号スイート。コンマ区切り。デフォルトなし。コンテキストのデフォルト暗号が使用されます。無効にした暗号は、すべてこれより削除されます。
transport.disabledCipherSuites	無効にする暗号スイート（コンマ区切り）。ここに一覧表示されている暗号は、有効な暗号から削除されます。デフォルトは指定しません。
transport.enabledProtocols	有効にするプロトコル（コンマ区切り）。デフォルトなし。コンテキストのデフォルトプロトコルが使用されます。無効にしたプロトコルは、すべてこれから削除されます。
transport.disabledProtocols	無効にするプロトコル（コンマ区切り）。ここに一覧表示されているプロトコルは、有効なプロトコルから削除されます。デフォルトはです **SSLv2Hello,SSLv3**。

オプション	description
transport.trustAll	設定された信頼ストアに関係なく、提供されるサーバー証明書を暗黙的に信頼するかどうか。デフォルトはです **false**。
transport.verifyHost	接続しているホスト名が提供されたサーバー証明書と一致していることを検証するかどうか。デフォルトはです **true**。
transport.keyAlias	クライアント証明書をサーバーに送信する必要がある場合にキーストアからキーペアを選択する際に使用するエイリアス。デフォルトは指定しません。

表21.7 AMQP オプション

オプション	description
amqp.idleTimeout	ピアが AMQP フレームを送信しない場合に接続が失敗するアイドルタイムアウト（ミリ秒単位）。デフォルトはです **60000**。
amqp.vhost	***vhost*** に接続します。SASL および Open hostname フィールドに値を設定するために使用されます。デフォルトは Connection URI の主要なホスト名です。
amqp.saslLayer	接続が SASL レイヤーを使用するかどうかを制御します。デフォルトはです **true**。
amqp.saslMechanisms	クライアントによって選択でき、サーバーが提供し、設定された認証情報で使用できる SASL メカニズム。1 つ以上のメカニズムを指定する場合はコンマで区切ります。デフォルトは、、、、およびであるすべてのクライアントがサポートするメカニズムから選択できるように **EXTERNAL CRAM-MD5 PLAIN** し **ANONYMOUS** ます。
amqp.maxFrameSize	max-frame-size の値は、ピアにアドバタイズされるバイト単位の値です。デフォルトはです **1048576**。

フェイルオーバーを有効にすると、現在の接続への接続が何らかの理由で失われたときに、クライアントが別のブローカーに自動的に再接続できます。フェイルオーバー URI は常に failover 接頭辞で開始され、ブローカーの URI の一覧は括弧内に含まれます。

jms. オプションは、括弧外にあるフェイルオーバー URI 全体に適用され、そのライフタイムの JMS Connection オブジェクトに影響します。

フェイルオーバーの URI は以下のようになります。

```
failover:(amqp://host1:5672,amqp://host2:5672)?
jms.clientID=foo&failover.maxReconnectAttempts=20
```

括弧内の個別のブローカーの詳細情報は、以前に定義した transport. または amqp. オプションを使用し、各ホストが接続されたときにこれらを適用することができます。

```
failover:(amqp://host1:5672?amqp.option=value,amqp://host2:5672?transport.option=value)?jms.clientID=foo
```

表21.8 フェイルオーバーオプション

オプション	description
failover.initialReconnectDelay	クライアントがリモートピアへの再接続を試みるまで待機する時間。デフォルト値はゼロ(**0**)で、最初の試行は即時に行われます。
failover.reconnectDelay	連続する再接続試行間の遅延を制御します（デフォルトは **10** ミリ秒）。backoff オプションが有効になっていないと、この値は一定のままになります。
failover.maxReconnectDelay	再接続を試行するまでクライアントが待機する最大時間。この値は、遅延が大きくなり過ぎないようにバックオフ機能が有効になっている場合にのみ使用されます。接続試行の最大時間として、デフォルトで **30** 秒数になります。
failover.useReconnectBackOff	設定された乗数に基づいて再接続試行の間隔が増大するかどうかを制御します。このオプションのデフォルトはです **true**。
failover.reconnectBackOffMultiplier	再接続の遅延値を拡張するために使用される乗数のデフォルトはです **2.0d**。
failover.maxReconnectAttempts	接続を失敗したとクライアントに報告する前の再接続試行回数。デフォルトは制限なしまたは(**-1**)です。
failover.startupMaxReconnectAttempts	このオプションの前にリモートピアに接続されていないクライアントでは、接続に失敗したと報告するまでの接続試行回数を制御します。デフォルトは、の値を使用し **maxReconnectAttempts**ます。
failover.warnAfterReconnectAttempts	フェイルオーバーの再接続が試行されたことを示すメッセージをクライアントがログに記録する頻度を制御します。デフォルトでは、すべての **10** 接続試行をログに記録します。

また、フェイルオーバー URI は、「ネストされた」オプションの定義をサポートします。AMQP および transport オプションの値は、すべてのネストされたブローカー URI に適用できるので、繰り返しを避けるのに役立ちます。

これは、フェイルオーバーでないブローカー URI についてすでに概説された同じ transport. および amqp. URI オプションを使用して実行されますが、でプレフィックスが付けられ*failover.nested.*ます。たとえば、*amqp.vhost* オプションに同じ値を接続しているすべてのブローカーに適用するには、以下のような URI がある場合があります。

```
failover:(amqp://host1:5672,amqp://host2:5672)?
jms.clientID=foo&failover.nested.amqp.vhost=myhost
```

クライアントにはオプションの Discovery モジュールがあります。これは、接続するブローカー URI が初期 URI では提供されず、関連する検出エージェントでクライアントが動作する際に検出される、カスタマイズされたフェイルオーバー層を提供します。現在、2 つの検出エージェントの実装があり、ファイルから URI を読み込むファイル監視と、リッスンするクライアントに対してブローカーアドレスをブロードキャストするように設定されている ActiveMQ 5 ブローカーで動作するマルチキャストリスナーがあります。

検出の使用時の一般的なフェイルオーバー関連のオプションは、先に詳細に示されているものと同じで、メインの接頭辞が failover. から discovery. に更新され、検出されたすべてのブローカー URI に共通する URI オプションを指定するのに使用される「ネストされた」オプション接頭辞で、先に詳細が更新され *failover.nested.* discovery.discoveredます。たとえば、エージェント URI の詳細がない場合、一般的な検出 URI は以下のようになります。

```
discovery:(<agent-uri>)?
discovery.maxReconnectAttempts=20&discovery.discovered.jms.clientID=foo
```

ファイル監視検出エージェントを使用するには、フォームのエージェント URI を使用します。

```
discovery:(file:///path/to/monitored-file?updateInterval=60000)
```

ファイル監視エージェントの URI オプションはです **updateInterval**。これは、ファイルを検査する頻度（ミリ秒単位）を制御します。デフォルト値はです **30000**。

ActiveMQ 5 ブローカーでマルチキャスト検出エージェントを使用するには、フォームのエージェント URI を使用します。

```
discovery:(multicast://default?group=default)
```

上記のマルチキャストエージェント URI でホストとして default を使用することに注意してください（これは、エージェントがデフォルトに置き換えられます **239.255.2.3:6155**）。マルチキャスト設定で使用する実際の IP およびポートを指定するには、これを変更することができます。

マルチキャスト検出エージェントの URI オプションはです **group**。これは、どのマルチキャストグループメッセージがリッスンされるかを制御します。デフォルト値はです **default**。

バグを報告します。

21.3. QPID AMQP 1.0 JMS クライアントロギング

クライアントは SLF4J API を利用し、ユーザーは Log4J を使用するために slf4j-log4j などの SLF4J 'binding' を提供することで、ニーズに応じて特定のロギング実装を選択できます。SLF4J の詳細は、を参照してください http://www.slf4j.org/。

クライアントは、にあるロガー名を使用します。 **org.apache.qpid.jms** 階層： 必要に応じてロギング実装を設定できます。

問題をデバッグする際に、Qpid Proton AMQP 1.0 ライブラリーから追加のプロトコルトレースロギングを有効にすることが役に立つ場合があります。これを行うには、2 つのオプションがあります。

- 環境変数を（Java システムプロパティーではなく） **PN_TRACE_FRM** に設定します。これにより **true**、Proton はフレームログを標準出力(stdout)に記録します。

- クライアントがプロトコルトレーサーを Proton **amqp.traceFrames=true** に追加して、接続 URI にオプションを追加し、以下を設定します。
  org.apache.qpid.jms.provider.amqp.FRAMES ログ **TRACE** に出力が含まれるロガー。

バグを報告します。

第22章 QPID C++ MESSAGING の .NET バインディング

22.1. C++ メッセージングクライアントの例の.NET バインディング

表22.1 クライアントとサーバーの例

名前の例	説明の例
csharp.example.server	レシーバーを作成し、メッセージをリッスンします。受信時に、メッセージのコンテンツは大文字に変換され、受信したメッセージの ReplyTo アドレスに転送されます。
csharp.example.client	サーバーに一連のメッセージを送信し、元のメッセージの内容と受信したメッセージコンテンツを出力します。

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.2. C++ メッセージング API の『.NET バインディングクラスマッピング』

表22.2 マップ送信および受信側の例

名前の例	説明の例
csharp.map.receiver	レシーバーを作成し、マップメッセージをリッスンします。受信時にメッセージがデコードされ、コンソールに表示されます。
csharp.map.sender	マップメッセージを作成し、そのメッセージをに送信し **map.receiver**ます。マップメッセージには、サポートされるすべての .NET メッセージングバインディングデータタイプの値が含まれます。

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.3. .NET バインディング（C++ MESSAGING API クラス： ADDRESS）

表22.3 .NET バインディング（C++ Messaging API クラス： Address）

.NET バインディングクラス： Address	
言語	**構文**
C++	***class Address***
.NET	***public ref class Address***
コンストラクター	
C++	***Address();***
.NET	***public Address();***
コンストラクター	
C++	***Address(const std::string& address);***
.NET	***public Address(string address);***
コンストラクター	
C++	***Address(const std::string& name, const std::string& subject, const qpid::types::Variant::Map& options, const std::string& type = "");***
.NET	***public Address(string name, string subject, Dictionary<string, object> options);***
.NET	***public Address(string name, string subject, Dictionary<string, object> options, string type);***
コンストラクターのコピー	
C++	***Address(const Address& address);***
.NET	***public Address(Address address);***
デストラクター	
C++	***~Address();***
.NET	***~Address();***
finalizer	
C++	該当なし
.NET	***!Address();***

.NET バインディングクラス： Address	
言語	**構文**
割り当て演算子をコピーする	
C++	***Address& operator=(const Address&);***
.NET	***public Address op_Assign(Address rhs);***
property: 名前	
C++	***const std::string& getName() const;***
C++	***void setName(const std::string&);***
.NET	***public string Name { get; set; }***
property: Subject	
C++	***const std::string& getSubject() const;***
C++	***void setSubject(const std::string&);***
.NET	***public string Subject { get; set; }***
プロパティー： オプション	
C++	***const qpid::types::Variant::Map& getOptions() const;***
C++	***qpid::types::Variant::Map& getOptions();***
C++	***void setOptions(const qpid::types::Variant::Map&);***
.NET	***public Dictionary<string, object> Options { get; set; }***
プロパティー： タイプ	
C++	***std::string getType() const;***
C++	***void setType(const std::string&);***
.NET	***public string Type { get; set; }***
その他の部分	
C++	***std::string str() const;***

.NET バインディングクラス： Address	
言語	**構文**
.NET	***public string ToStr();***
その他の部分	
C++	***operator bool() const;***
.NET	該当なし
その他の部分	
C++	***bool operator !() const;***
.NET	該当なし

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.4. .NET BINDING FOR THE C++ MESSAGING API CLASS: CONNECTION

表22.4 .NET Binding for the C++ Messaging API Class: Connection

.NET Binding Class: Connection	
言語	**構文**
C++	***class Connection : public qpid::messaging::Handle<ConnectionImpl>***
.NET	***public ref class Connection***
コンストラクター	
C++	***Connection(ConnectionImpl* impl);***
.NET	該当なし
コンストラクター	
C++	***Connection();***

.NET Binding Class: Connection	
言語	**構文**
.NET	該当なし
コンストラクター	
C++	***Connection(const std::string& url, const qpid::types::Variant::Map& options = qpid::types::Variant::Map());***
.NET	***public Connection(string url);***
.NET	***public Connection(string url, Dictionary<string, object> options);***
コンストラクター	
C++	***Connection(const std::string& url, const std::string& options);***
.NET	***public Connection(string url, string options);***
Copy Constructor	
C++	***Connection(const Connection&);***
.NET	***public Connection(Connection connection);***
デストラクター	
C++	***~Connection();***
.NET	***~Connection();***
finalizer	
C++	該当なし
.NET	***!Connection();***
割り当て演算子をコピーする	
C++	***Connection& operator=(const Connection&);***
.NET	***public Connection op_Assign(Connection rhs);***
method: setOption	

.NET Binding Class: Connection	
言語	**構文**
C++	***void setOption(const std::string& name, const qpid::types::Variant& value);***
.NET	***public void SetOption(string name, object value);***
メソッド： オープン	
C++	***void open();***
.NET	***public void Open();***
プロパティー： isOpen	
C++	***bool isOpen();***
.NET	***public bool IsOpen { get; }***
method: close	
C++	***void close();***
.NET	***public void Close();***
メソッド： createTransactionalSession	
C++	***Session createTransactionalSession(const std::string& name = std::string());***
.NET	***public Session CreateTransactionalSession();***
.NET	***public Session CreateTransactionalSession(string name);***
メソッド： createSession	
C++	***Session createSession(const std::string& name = std::string());***
.NET	***public Session CreateSession();***
.NET	***public Session CreateSession(string name);***
メソッド： getSession	
C++	***Session getSession(const std::string& name) const;***
.NET	***public Session GetSession(string name);***

.NET Binding Class: Connection	
言語	**構文**
Property: AuthenticatedUsername	
C++	***std::string getAuthenticatedUsername();***
.NET	***public string GetAuthenticatedUsername();***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.5. .NET BINDING FOR THE C++ MESSAGING API CLASS: DURATION

表22.5 .NET Binding for the C++ Messaging API Class: Duration

.NET バインディングクラス： Duration	
言語	**構文**
C++	***class Duration***
.NET	***public ref class Duration***
コンストラクター	
C++	***explicit Duration(uint64_t milliseconds);***
.NET	***public Duration(ulong mS);***
コンストラクターのコピー	
C++	該当なし
.NET	***public Duration(Duration rhs);***
デストラクター	
C++	デフォルト
.NET	デフォルト
finalizer	

.NET バインディングクラス： Duration	
言語	**構文**
C++	該当なし
.NET	デフォルト
property: intervaliseconds	
C++	***uint64_t getMilliseconds() const;***
.NET	***public ulong Milliseconds { get; }***
operator: *	
C++	***Duration operator*(const Duration& duration, uint64_t multiplier);***
.NET	***public static Duration operator *(Duration dur, ulong multiplier);***
.NET	***public static Duration Multiply(Duration dur, ulong multiplier);***
C++	***Duration operator*(uint64_t multiplier, const Duration& duration);***
.NET	***public static Duration operator *(ulong multiplier, Duration dur);***
.NET	***public static Duration Multiply(ulong multiplier, Duration dur);***
定数	
C++	***static const Duration FOREVER;***
C++	***static const Duration IMMEDIATE;***
C++	***static const Duration SECOND;***
C++	***static const Duration MINUTE;***
.NET	***public sealed class DurationConstants***
.NET	***public static Duration FORVER;***
.NET	***public static Duration IMMEDIATE;***
.NET	***public static Duration MINUTE;***
.NET	***public static Duration SECOND;***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.6. .NET BINDING FOR THE C++ MESSAGING API CLASS: FAILOVERUPDATES

表22.6 .NET Binding for the C++ Messaging API Class: failoverUpdates

.NET Binding Class: failoverUpdates	
言語	**構文**
C++	***class FailoverUpdates***
.NET	***public ref class FailoverUpdates***
コンストラクター	
C++	***FailoverUpdates(Connection& connection);***
.NET	***public FailoverUpdates(Connection connection);***
デストラクター	
C++	***~FailoverUpdates();***
.NET	***~FailoverUpdates();***
finalizer	
C++	該当なし
.NET	***!FailoverUpdates();***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.7. .NET バインディング（C++ MESSAGING API クラス用）: MESSAGE

表22.7 .NET バインディング（C++ Messaging API クラス用）: Message

.NET バインディングクラス： Message	
言語	**構文**
C++	***class Message***
.NET	***public ref class Message***
コンストラクター	
C++	***Message(const std::string& bytes = std::string());***
.NET	***Message();***
.NET	***Message(System::String ^ theStr);***
.NET	***Message(System::Object ^ theValue);***
.NET	***Message(array<System::Byte> ^ bytes);***
コンストラクター	
C++	***Message(const char*, size_t);***
.NET	***public Message(byte[] bytes, int offset, int size);***
Copy Constructor	
C++	***Message(const Message&);***
.NET	***public Message(Message message);***
割り当て演算子をコピーする	
C++	***Message& operator=(const Message&);***
.NET	***public Message op_Assign(Message rhs);***
デストラクター	
C++	***~Message();***
.NET	***~Message();***
finalizer	
C++	該当なし

.NET バインディングクラス： Message	
言語	**構文**
.NET	***!Message()***
property: ReplyTo	
C++	***void setReplyTo(const Address&);***
C++	***const Address& getReplyTo() const;***
.NET	***public Address ReplyTo { get; set; }***
property: Subject	
C++	***void setSubject(const std::string&);***
C++	***const std::string& getSubject() const;***
.NET	***public string Subject { get; set; }***
Property: ContentType	
C++	***void setContentType(const std::string&);***
C++	***const std::string& getContentType() const;***
.NET	***public string ContentType { get; set; }***
property: MessageId	
C++	***void setMessageId(const std::string&);***
C++	***const std::string& getMessageId() const;***
.NET	***public string MessageId { get; set; }***
property: UserId	
C++	***void setUserId(const std::string&);***
C++	***const std::string& getUserId() const;***
.NET	***public string UserId { get; set; }***
プロパティー： CorrelationId	

.NET バインディングクラス： Message	
言語	**構文**
C++	***void setCorrelationId(const std::string&);***
C++	***const std::string& getCorrelationId() const;***
.NET	***public string CorrelationId { get; set; }***
プロパティー： Priority	
C++	***void setPriority(uint8_t);***
C++	***uint8_t getPriority() const;***
.NET	***public byte Priority { get; set; }***
プロパティー： Ttl	
C++	***void setTtl(Duration ttl);***
C++	***Duration getTtl() const;***
.NET	***public Duration Ttl { get; set; }***
プロパティー： Durable	
C++	***void setDurable(bool durable);***
C++	***bool getDurable() const;***
.NET	***public bool Durable { get; set; }***
プロパティー： Redelivered	
C++	***bool getRedelivered() const;***
C++	***void setRedelivered(bool);***
.NET	***public bool Redelivered { get; set; }***
メソッド： setProperty	
C++	***void setProperty(const std::string&, const qpid::types::Variant&);***
.NET	***public void SetProperty(string name, object value);***

.NET バインディングクラス： Message	
言語	**構文**
プロパティー： プロパティー	
C++	***const qpid::types::Variant::Map& getProperties() const;***
C++	***qpid::types::Variant::Map& getProperties();***
.NET	***public Dictionary<string, object> Properties { get; set; }***
Method: SetContent	
C++	***void setContent(const std::string&);***
C++	***void setContent(const char* chars, size_t count);***
.NET	***public void SetContent(byte[] bytes);***
.NET	***public void SetContent(string content);***
.NET	***public void SetContent(byte[] bytes, int offset, int size);***
メソッド： GetContent	
C++	***std::string getContent() const;***
.NET	***public string GetContent();***
.NET	***public void GetContent(byte[] arr);***
.NET	***public void GetContent(Collection<object> __p1);***
.NET	***public void GetContent(Dictionary<string, object> dict);***
Method: GetContentPtr	
C++	***const char* getContentPtr() const;***
.NET	該当なし
プロパティー： ContentSize	
C++	***size_t getContentSize() const;***
.NET	***public ulong ContentSize { get; }***

.NET バインディングクラス： Message	
言語	**構文**
struct: EncodingException	
C++	***struct EncodingException : qpid::types::Exception***
.NET	該当なし
method: デコード	
C++	***void decode(const Message& message, qpid::types::Variant::Map& map, const std::string& encoding = std::string());***
C++	***void decode(const Message& message, qpid::types::Variant::List& list, const std::string& encoding = std::string());***
.NET	該当なし
メソッド： encode	
C++	***void encode(const qpid::types::Variant::Map& map, Message& message, const std::string& encoding = std::string());***
C++	***void encode(const qpid::types::Variant::List& list, Message& message, const std::string& encoding = std::string());***
.NET	該当なし
メソッド： AsString	
C++	該当なし
.NET	***public string AsString(object obj);***
.NET	***public string ListAsString(Collection<object> list);***
.NET	***public string MapAsString(Dictionary<string, object> dict);***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.8. .NET バインディング（C++ MESSAGING API クラス： RECEIVER）

表22.8. .NET バインディング（C++ Messaging API クラス： Receiver）

表22.8 .NETバインディング（C++ Messaging APIクラス： Receiver）

.NET Binding Class: Receiver	
言語	**構文**
C++	***class Receiver***
.NET	***public ref class Receiver***
コンストラクター	
.NET	***Constructed object is returned by Session.CreateReceiver***
コンストラクターのコピー	
C++	***Receiver(const Receiver&);***
.NET	***public Receiver(Receiver receiver);***
デストラクター	
C++	***~Receiver();***
.NET	***~Receiver();***
finalizer	
C++	該当なし
.NET	***!Receiver()***
割り当て演算子をコピーする	
C++	***Receiver& operator=(const Receiver&);***
.NET	***public Receiver op_Assign(Receiver rhs);***
メソッド： Get	
C++	***bool get(Message& message, Duration timeout=Duration::FOREVER);***
.NET	***public bool Get(Message mmsgp);***
.NET	***public bool Get(Message mmsgp, Duration durationp);***
メソッド： Get	

.NET Binding Class: Receiver	
言語	**構文**
C++	***Message get(Duration timeout=Duration::FOREVER);***
.NET	***public Message Get();***
.NET	***public Message Get(Duration durationp);***
メソッド： Fetch	
C++	***bool fetch(Message& message, Duration timeout=Duration::FOREVER);***
.NET	***public bool Fetch(Message mmsgp);***
.NET	***public bool Fetch(Message mmsgp, Duration duration);***
メソッド： Fetch	
C++	***Message fetch(Duration timeout=Duration::FOREVER);***
.NET	***public Message Fetch();***
.NET	***public Message Fetch(Duration durationp);***
property: Capacity	
C++	***void setCapacity(uint32_t);***
C++	***uint32_t getCapacity();***
.NET	***public uint Capacity { get; set; }***
プロパティー： Available	
C++	***uint32_t getAvailable();***
.NET	***public uint Available { get; }***
property: Unsettled	
C++	***uint32_t getUnsettled();***
.NET	***public uint Unsettled { get; }***
メソッド： 閉じる	

.NET Binding Class: Receiver	
言語	**構文**
C++	***void close();***
.NET	***public void Close();***
property: IsClosed	
C++	***bool isClosed() const;***
.NET	***public bool IsClosed { get; }***
property: 名前	
C++	***const std::string& getName() const;***
.NET	***public string Name { get; }***
プロパティー： Session	
C++	***Session getSession() const;***
.NET	***public Session Session { get; }***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.9. .NET BINDING FOR THE C++ MESSAGING API CLASS: SENDER

表22.9 .NET Binding for the C++ Messaging API Class: Sender

.NET Binding Class: Sender	
言語	**構文**
C++	***class Sender***
.NET	***public ref class Sender***
コンストラクター	
.NET	***Constructed object is returned by session.createSender***

.NET Binding Class: Sender	
言語	**構文**
コンストラクターのコピー	
C++	***Sender(const Sender&);***
.NET	***public Sender(Sender sender);***
デストラクター	
C++	***~Sender();***
.NET	***~Sender();***
finalizer	
C++	該当なし
.NET	***!Sender()***
割り当て演算子をコピーする	
C++	***Sender& operator=(const Sender&);***
.NET	***public Sender op_Assign(Sender rhs);***
メソッド： Send	
C++	***void send(const Message& message, bool sync=false);***
.NET	***public void Send(Message mmsgp);***
.NET	***public void Send(Message mmsgp, bool sync);***
メソッド： 閉じる	
C++	***void close();***
.NET	***public void Close();***
property: Capacity	
C++	***void setCapacity(uint32_t);***
C++	***uint32_t getCapacity();***

.NET Binding Class: Sender	
言語	**構文**
.NET	***public uint Capacity { get; set; }***
プロパティー： Available	
C++	***uint32_t getAvailable();***
.NET	***public uint Available { get; }***
property: Unsettled	
C++	***uint32_t getUnsettled();***
.NET	***public uint Unsettled { get; }***
property: 名前	
C++	***const std::string& getName() const;***
.NET	***public string Name { get; }***
プロパティー： Session	
C++	***Session getSession() const;***
.NET	***public Session Session { get; }***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.10. .NET BINDING FOR THE C++ MESSAGING API CLASS: SESSION

表22.10 .NET Binding for the C++ Messaging API Class: Session

言語	構文
C++	***class Session***
.NET	***public ref class Session***
コンストラクター	

言語	構文
.NET	構築されたオブジェクトがによって返される ***Connection.CreateSession***
コンストラクターのコピー	
C++	***Session(const Session&);***
.NET	***public Session(Session session);***
デストラクター	
C++	***~Session();***
.NET	***~Session();***
finalizer	
C++	該当なし
.NET	***!Session()***
割り当て演算子をコピーする	
C++	***Session& operator=(const Session&);***
.NET	***public Session op_Assign(Session rhs);***
メソッド： 閉じる	
C++	***void close();***
.NET	***public void Close();***
メソッド： コミット	
C++	***void commit();***
.NET	***public void Commit();***
メソッド： ロールバック	
C++	***void rollback();***
.NET	***public void Rollback();***
method: Acknowledge	

言語	構文
C++	***void acknowledge(bool sync=false);***
C++	***void acknowledge(Message&, bool sync=false);***
.NET	***public void Acknowledge();***
.NET	***public void Acknowledge(bool sync);***
.NET	***public void Acknowledge(Message __p1);***
.NET	***public void Acknowledge(Message __p1, bool __p2);***
メソッド： 拒否	
C++	***void reject(Message&);***
.NET	***public void Reject(Message __p1);***
方法： リリース	
C++	***void release(Message&);***
.NET	***public void Release(Message __p1);***
メソッド： 同期	
C++	***void sync(bool block=true);***
.NET	***public void Sync();***
.NET	***public void Sync(bool block);***
property: Receivable	
C++	***uint32_t getReceivable();***
.NET	***public uint Receivable { get; }***
Property: UnsettledAcks	
C++	***uint32_t getUnsettledAcks();***
.NET	***public uint UnsettledAcks { get; }***
メソッド： NextReceiver	

言語	構文
C++	***bool nextReceiver(Receiver&, Duration timeout=Duration::FOREVER);***
.NET	***public bool NextReceiver(Receiver rcvr);***
.NET	***public bool NextReceiver(Receiver rcvr, Duration timeout);***
メソッド： NextReceiver	
C++	***Receiver nextReceiver(Duration timeout=Duration::FOREVER);***
.NET	***public Receiver NextReceiver();***
.NET	***public Receiver NextReceiver(Duration timeout);***
メソッド： CreateSender	
C++	***Sender createSender(const Address& address);***
.NET	***public Sender CreateSender(Address address);***
メソッド： CreateSender	
C++	***Sender createSender(const std::string& address);***
.NET	***public Sender CreateSender(string address);***
Method: CreateReceiver	
C++	***Receiver createReceiver(const Address& address);***
.NET	***public Receiver CreateReceiver(Address address);***
Method: CreateReceiver	
C++	***Receiver createReceiver(const std::string& address);***
.NET	***public Receiver CreateReceiver(string address);***
メソッド： GetSender	
C++	***Sender getSender(const std::string& name) const;***
.NET	***public Sender GetSender(string name);***
メソッド： GetReceiver	

言語	構文
C++	***Receiver getReceiver(const std::string& name) const;***
.NET	***public Receiver GetReceiver(string name);***
プロパティー： Connection	
C++	***Connection getConnection() const;***
.NET	***public Connection Connection { get; }***
プロパティー： HasError	
C++	***bool hasError();***
.NET	***public bool HasError { get; }***
メソッド： CheckError	
C++	***void checkError();***
.NET	***public void CheckError();***

関連項目

- 「Windows SDK コンテンツ」

バグを報告します。

22.11. .NET クラス： SESSIONRECEIVER

SessionReceiver クラスは、特定のセッションですべてのレシーバーが受信したメッセージに便利なコールバックメカニズムを提供します。

```
using Org.Apache.Qpid.Messaging;
using System;

namespace Org.Apache.Qpid.Messaging.SessionReceiver
{
    public interface ISessionReceiver
    {
        void SessionReceiver(Receiver receiver, Message message);
    }

    public class CallbackServer
    {
        public CallbackServer(Session session, ISessionReceiver callback);
```

```
        public void Close();
    }
}
```

このクラスを使用するには、クライアントプログラムには Org.Apache.Qpid.Messaging およびへの参照が含まれ Org.Apache.Qpid.Messaging.SessionReceiverます。呼び出しプログラムは、ISessionReceiver インターフェースを実装する関数を作成します。この関数は、セッションがメッセージを受信すると常に呼び出されます。コールバックプロセスは作成によって開始され CallbackServer、クライアントプログラムがを呼び出すまで実行され CallbackServer.Close functionます。

SessionReceiver コールバックを使用した完全な操作例は、に含まれてい cpp/bindings/qpid/dotnet/examples/csharp.map.callback.receiverます。

関連項目

- 「Windows SDK コンテンツ」

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 改訂履歴

改訂 3.2.0-6 Fri Oct 16 2015 **Conscot ムムー**
Post 3.2 GA update: 『qpid-jms AMQP 1.0 client Chapter を使用』 した追加。

改訂 3.2.0-5 Thu Oct 8 2015 **Conscot ムムー**
MRG-M 3.2 GA

改訂 3.2.0-3 Tue Sep 29 2015 **Conscot ムムー**
MRG-M 3.2 GA 用準備

改訂 3.2.0-1 Tue Jul 14 2015 **Jared イタリア**
MRG-M 3.2 GA 用準備

改訂 3.1.0-5 Wed Apr 01 2015 **Jared イタリア**
MRG-M 3.1 GA 用準備

改訂 3.0.0-4 Tue Sep 23 2014 **Jared イタリア**
MRG-M 3.0 GA 用準備