Red Hat AMQ Clients 2.11

AMQ .NET クライアントの使用

AMQ Clients 2.11 向け

Last Updated: 2023-07-14

Red Hat AMQ Clients 2.11 AMQ .NET クライアントの使用

AMQ Clients 2.11 向け

法律上の通知

概要

本ガイドでは、クライアントをインストールして設定する方法、実例を実行し、他の AMQ コンポーネントでクライアントを使用する方法を説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 概要

AMQ .NET は、.NET プラットフォームの軽量 AMQP 1.0 ライブラリーです。また、AMQP メッセージを送受信する .NET アプリケーションを作成できます。

AMQ .NET は AMQ Clients (複数の言語やプラットフォームをサポートするメッセージングライブラリースイート) に含まれています。クライアントの概要は、AMQ Clients の概要 を参照してください。本リリースに関する詳細は、AMQ Clients 2.11 リリースノート を参照してください。

AMQ .NET は、AMQP.Net Lite をベースにしています。詳細な API ドキュメントは、AMQ .NET API reference を参照してください。

1.1. 主な特長

- セキュアな通信用の SSL/TLS
- 柔軟な SASL 認証
- AMQP とネイティブのデータ型間のシームレスな変換
- AMQP 1.0 の全機能へのアクセス
- 完全な **IntelliSense** API ドキュメントを含む統合開発環境

1.2. サポートされる標準およびプロトコル

AMQ .NET は、以下の業界標準およびネットワークプロトコルをサポートします。

- Advanced Message Queueing Protocol (AMQP) のバージョン 1.0
- SSL の後継である TLS (Transport Layer Security) プロトコルのバージョン 1.0、1.1、1.2、および 1.3
- Simple Authentication and Security Layer (SASL) メカニズム ANONYMOUS、PLAIN、EXTERNAL
- IPv6 での最新の TCP

1.3. サポートされる構成

AMQ .NET でサポートされる構成に関する最新情報は、Red Hat Customer Portal で Red Hat AMQ でサポートされる設定 を参照してください。

1.4. 用語および概念

このセクションでは、コア API エンティティーを紹介し、コア API が連携する方法を説明します。

表1.1 API の用語

エンティティー	説明
接続	ネットワーク上の 2 つのピア間の通信チャネル

エンティティー	説明
セッション	メッセージの送受信を行うためのコンテキスト
送信者リンク	メッセージをターゲットに送信するためのチャネル
受信者リンク	ソースからメッセージを受信するためのチャネル
ソース	メッセージの名前付きの発信元
ターゲット	メッセージの名前付きの受信先
メッセージ	アプリケーションデータの変更可能なホルダー

AMQ .NET は **メッセージ** を送受信します。メッセージは、**リンク** を介して接続されたコンテナー間で転送されます。**セッション** 上でリンクは確立されます。セッションは**接続**上で確立されます。

送信ピアは、メッセージ送信用の **送信者リンク** を作成します。送信者リンクには、リモートピアでキューまたはトピックを識別する **ターゲット** があります。受信クライアントは、メッセージ受信用の**受信者リンク** を作成します。受信者リンクには、リモートピアでキューまたはトピックを識別する **ソース** があります。

1.5. このドキュメントの表記慣例

sudo コマンド
このドキュメントでは、root 権限を必要とするすべてのコマンドに対して **sudo** が使用されています。すべての変更がシステム全体に影響する可能性があるため、**sudo** を使用する場合は注意が必要です。**sudo** の詳細は、sudo コマンドの使用 を参照してください。

ファイルパス
このドキュメントでは、すべてのファイルパスが Linux、UNIX、および同様のオペレーティングシステムで有効です (例: **/home/andrea**)。Microsoft Windows では、同等の Windows パスを使用する必要があります (例: **C:\Users\andrea**)。

変数テキスト
このドキュメントでは、変数を含むコードブロックが紹介されていますが、これは、お客様の環境に固有の値に置き換える必要があります。可変テキストは矢印の中括弧で囲まれ、斜体の等幅フォントとしてスタイル設定されます。たとえば、以下のコマンドでは **<project-dir>** は実際の環境の値に置き換えます。

```
$ cd <project-dir>
```

第2章 インストールシステム

この章では、環境に AMQ .NET をインストールする手順を説明します。

2.1. 前提条件

- AMQ リリースファイルおよびリポジトリーにアクセスするには、サブスクリプション が必要です。
- Red Hat Enterprise Linux で AMQ .NET を使用するには、.NET Core 3.1 開発者ツールをインストールする必要があります。詳細は .NET Core 3.1 スタートガイド を参照してください。
- Microsoft Windows で AMQ .NET を使用してプログラムを構築するには、Visual Studio をインストールする必要があります。

2.2. RED HAT ENTERPRISE LINUX へのインストール

手順

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **INTEGRATION AND AUTOMATION** カテゴリーで **Red Hat AMQ Clients** エントリーを見つけます。
3. **Red Hat AMQ Clients** をクリックします。**Software Downloads** ページが開きます。
4. **AMQ Clients 2.11.0 .NET Core**.zip ファイルをダウンロードします。
5. **unzip** コマンドを使用して、ファイルの内容を、選択したディレクトリーに展開します。

   ```
   $ unzip amq-clients-2.11.0-dotnet-core.zip
   ```

   .zip ファイルの内容を抽出すると、**amq-clients-2.11.0-dotnet-core** という名前のディレクトリーが作成されます。これはインストールの最上位ディレクトリーであり、このドキュメントでは **<install-dir>** と呼びます。
6. テキストエディターを使用して **$HOME/.nuget/NuGet/NuGet.Config** ファイルを作成し、以下の内容を追加します。

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <configuration>
     <packageSources>
       <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3"/>
       <add key="amq-clients" value="<install-dir>/nupkg"/>
     </packageSources>
   </configuration>
   ```

   NuGet.Config ファイルがすでにある場合は、**amq-clients** 行を追加します。

   または、**<install-dir>/nupkg** ディレクトリーの .nupkg ファイルを既存のパッケージソースの場所に移動できます。

2.3. MICROSOFT WINDOWS へのインストール

手順

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **INTEGRATION AND AUTOMATION** カテゴリーで **Red Hat AMQ Clients** エントリーを見つけます。
3. **Red Hat AMQ Clients** をクリックします。**Software Downloads** ページが開きます。
4. **AMQ Clients 2.11.0 .NET** .zip ファイルをダウンロードします。
5. zip ファイルを右クリックし、**Extract All** を選択して、選択したディレクトリーにファイルの内容を展開します。

.zip ファイルの内容を抽出すると、**amq-clients-2.11.0-dotnet** という名前のディレクトリーが作成されます。これはインストールの最上位ディレクトリーであり、このドキュメントでは **<install-dir>** と呼びます。

第3章 スタートガイド

この章では、環境を設定して簡単なメッセージングプログラムを実行する手順を説明します。

3.1. 前提条件

- ご使用の環境の インストール 手順を完了する必要があります。
- インターフェイス **localhost** およびポート **5672** で接続をリッスンする AMQP 1.0 メッセージブローカーが必要です。匿名アクセスを有効にする必要があります。詳細は、ブローカーの開始 を参照してください。
- **amq.topic** という名前のキューが必要です。詳細は、キューの作成 を参照してください。

3.2. RED HAT ENTERPRISE LINUX で HELLOWORLD を実行する

Hello World の例では、ブローカーへの接続を作成し、グリーティングを含むメッセージを **amq.topic** キューに送信して、受信しなおします。成功すると、受信したメッセージがコンソールに出力されます。

<install-dir>/examples/netcoreapp3/HelloWorld-simple に変更し、**dotnet run** を使用してプログラムを構築し、実行します。

```
$ cd <install-dir>/examples/netcoreapp3/HelloWorld-simple
$ dotnet run
Hello World!
```

3.3. MICROSOFT WINDOWS で HELLO WORLD を実行する

Hello World の例では、ブローカーへの接続を作成し、グリーティングを含むメッセージを **amq.topic** キューに送信して、受信しなおします。成功すると、受信したメッセージがコンソールに出力されます。

手順

1. **<install-dir>** に移動し、Visual Studio で **amqp.sln** ソリューションファイルを開きます。
2. **Build** メニューから **Build Solution** を選択して、ソリューションをコンパイルします。
3. コマンドプロンプトウィンドウを開き、以下のコマンドを実行してメッセージを送受信します。

```
> cd <install-dir>\bin\Debug
> HelloWorld-simple
Hello World!
```

第4章 例

この章では、サンプルプログラムで AMQ .NET を使用する方法について説明します。

その他の例は、AMQ .NET サンプルスイート および AMQP.Net Lite サンプル を参照してください。

4.1. メッセージの送信

このクライアントプログラムは **<connection-url>** を使用してサーバーに接続し、ターゲット **<address>** の送信者を作成し、**<message-body>** を含むメッセージを送信し、接続を切断して終了します。

例: メッセージの送信

```
namespace SimpleSend
{
    using System;
    using Amqp;                                                    (1)

    class SimpleSend
    {
        static void Main(string[] args)
        {
            string    url = (args.Length > 0) ? args[0] :          (2)
                "amqp://guest:guest@127.0.0.1:5672";
            string target = (args.Length > 1) ? args[1] : "examples";   (3)
            int     count = (args.Length > 2) ? Convert.ToInt32(args[2]) : 10;   (4)

            Address      peerAddr = new Address(url);              (5)
            Connection connection = new Connection(peerAddr);      (6)
            Session       session = new Session(connection);
            SenderLink     sender = new SenderLink(session, "send-1", target);   (7)

            for (int i = 0; i < count; i++)
            {
                Message msg = new Message("simple " + i);          (8)
                sender.Send(msg);                                  (9)
                Console.WriteLine("Sent: " + msg.Body.ToString());
            }

            sender.Close();                                        (10)
            session.Close();
            connection.Close();
        }
    }
}
```

(1) **using Amqp;** Amqp ネームスペースで定義されたタイプをインポートします。Amqp は、ライブラリーファイル **Amqp.Net.dll** へのプロジェクト参照によって定義され、AMQ .NET に関連するすべてのクラス、インターフェイス、および値タイプを提供します。

(2)

コマンドライン arg[0] **url** は、AMQP 接続のホストまたは仮想ホストのネットワークアドレスです。この文字列は、接続トランスポート、ユーザーとパスワードの認証情報、およびリモートホス

3 コマンドライン arg[1] **target** は、リモートホストのメッセージ先エンドポイントまたはリソースの名前です。

4 コマンドライン arg[2] **count** は送信するメッセージ数です。

5 **peerAddr** は AMQP コネクションの作成に必要な構造です。

6 AMQP コネクションを作成します。

7 **sender** は、メッセージを送信するクライアント SenderLink です。リンクは、任意に send-1 と命名されます。リンク名は、使用している環境において合理的で、混雑したシステム上でもトラフィックを識別しやすい名前を使用してください。リンク名に制限はありませんが、同じセッション内で一意である必要があります。

8 メッセージ送信ループでは、新しいメッセージが作成されます。

9 メッセージは AMQP ピアに送信されます。

10 すべてのメッセージが送信されると、プロトコルオブジェクトは正しい順序でシャットダウンされます。

サンプルの実行
サンプルプログラムを実行するには、コマンドラインでコンパイルして実行します。詳細は、3章*スタートガイド*を参照してください。

```
<install-dir>\bin\Debug>simple_send "amqp://guest:guest@localhost" service_queue
```

4.2. メッセージの受信

このクライアントプログラムは **<connection-url>** を使用してサーバーに接続し、ソース **<address>** の受信側を作成し、終了するか、**<count>** メッセージに到達するまでメッセージを受信します。

例: メッセージの受信

```
namespace SimpleRecv
{
   using System;
   using Amqp;                                                    1

   class SimpleRecv
   {
      static void Main(string[] args)
      {
         string    url = (args.Length > 0) ? args[0] :                    2
            "amqp://guest:guest@127.0.0.1:5672";
         string source = (args.Length > 1) ? args[1] : "examples";        3
         int     count = (args.Length > 2) ? Convert.ToInt32(args[2]) : 10; 4

         Address      peerAddr = new Address(url);                        5
         Connection connection = new Connection(peerAddr);                6
         Session       session = new Session(connection);
```

```
            ReceiverLink receiver = new ReceiverLink(session, "recv-1", source);
            for (int i = 0; i < count; i++)
            {
                Message msg = receiver.Receive();
                receiver.Accept(msg);
                Console.WriteLine("Received: " + msg.Body.ToString());
            }
            receiver.Close();
            session.Close();
            connection.Close();
        }
    }
}
```

1. **using Amqp;** Amqp ネームスペースで定義されたタイプをインポートします。Amqp は、ライブラリーファイル **Amqp.Net.dll** へのプロジェクト参照によって定義され、AMQ .NET に関連するすべてのクラス、インターフェイス、および値タイプを提供します。
2. コマンドライン arg[0] **url** は、AMQP 接続のホストまたは仮想ホストのネットワークアドレスです。この文字列は、接続トランスポート、ユーザーとパスワードの認証情報、およびリモートホストの接続のポート番号を説明します。**url** はブローカー、スタンドアロンピア、またはルーターネットワークの Ingress ポイントに対応できます。
3. コマンドライン arg[1] **source** は、リモートホストのメッセージソースエンドポイントまたはリソースの名前です。
4. コマンドライン arg[2] **count** は送信するメッセージ数です。
5. **peerAddr** は AMQP コネクションの作成に必要な構造です。
6. AMQP コネクションを作成します。
7. **receiver** は、メッセージを受信するクライアント **ReceiverLink** です。リンクは、任意に **recv-1** という名前です。リンク名は、使用している環境において合理的で、混雑したシステム上でもトラフィックを識別しやすい名前を使用してください。リンク名に制限はありませんが、同じセッション内で一意である必要があります。
8. メッセージが受信されます。
9. メッセージが許可されます。これにより、メッセージの所有権がピアから受信側に譲渡されます。
10. すべてのメッセージが受信されると、プロトコルオブジェクトは正しい順序でシャットダウンされます。

サンプルの実行

サンプルプログラムを実行するには、コマンドラインでコンパイルして実行します。詳細は、*3章スタートガイド* を参照してください。

```
<install-dir>\bin\Debug>simple_recv "amqp://guest:guest@localhost" service_queue
```

第5章 ネットワーク接続

5.1. 接続 URI

このセクションでは、AMQP リモートピアへの接続に使用される Connection URI 文字列の標準形式を説明します。

```
scheme = ( "amqp" | "amqps" )
host = ( <fully qualified domain name> | <hostname> | <numeric IP address> )

URI = scheme "://" [user ":" [password] "@"] host [":" port]
```

- **scheme amqp** - コネクションは TCP トランスポートを使用し、デフォルトのポートを 5672 に設定します。
- **scheme amqp** - コネクションは SSL/TLS トランスポートを使用し、デフォルトのポートを 5671 に設定します。
- **user** - オプションの接続認証ユーザー名。**ユーザー** 名が存在する場合、クライアントはコネクション起動時に AMQP SASL ユーザー認証情報の交換を開始します。
- **password** - オプションの接続認証パスワード。
- **host** - 接続が転送されるネットワークホスト。
- **port** - 接続が転送されるネットワークポート。**ポート** のデフォルト値は AMQP トランスポートスキームにより決定されます。

接続 URI サンプル

```
amqp://127.0.0.1
amqp://amqpserver.example.com:5672
amqps://joe:somepassword@bigbank.com
amqps://sue:secret@test.example.com:21000
```

5.2. 再接続およびフェイルオーバー

AMQ .NET は再接続とフェイルオーバーを提供しませんが、接続エラーを傍受して再接続することで、アプリケーションに実装できます。コードの例は、ReconnectSender.cs の例 を参照してください。

第6章 セキュリティー

6.1. ユーザーとパスワードを使用した接続

AMQ .NET は、ユーザーとパスワードによる接続を認証できます。

認証に使用する認証情報を指定するには、接続 URL に **user** および **password** フィールドを設定します。

例: ユーザーとパスワードを使用した接続

```
Address addr = new Address("amqp://<user>:<password>@example.com");
Connection conn = new Connection(addr);
```

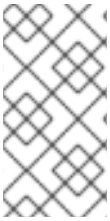
6.2. SASL 認証の設定

リモートピアへのクライアント接続は、SASL ユーザー名とパスワードの認証情報を交換できます。接続 URI に **user** フィールドが存在すると、この交換を制御します。**user** を指定すると、SASL 認証情報が交換されます。**user** を指定しないと、SASL 認証情報は交換されません。

デフォルトでは、クライアントは **EXTERNAL**、**PLAIN**、および **ANONYMOUS** SASL メカニズムをサポートします。

6.3. SSL/TLS 接続の設定

SSL/TLS を使用してサーバーとの通信のセキュリティーを確保します。クライアントは、SSL/TLS ハンドシェイクのみ、または SSL/TLS ハンドシェイクおよびクライアント証明書認証用に設定できます。詳細は、証明書の管理 セクションを参照してください。



注記

TLS Server Name Indication (SNI) はクライアントライブラリーによって自動的に処理されます。ただし、SNI は、ホストが完全修飾ドメイン名またはホスト名である **amqps** トランスポートスキームを使用するアドレスに対してのみシグナル化されます。SNI は、ホストが数値の IP アドレスである場合にシグナル送信されません。

第7章 送信者と受信者

クライアントは、送信者と受信者のリンクを使用して、メッセージ配信のチャネルを表現します。送信者と受信者は一方向であり、送信元はメッセージの発信元に、ターゲットはメッセージの宛先になります。

ソースとターゲットは、多くの場合、メッセージブローカーのキューまたはトピックを参照します。ソースは、サブスクリプションを表すためにも使用されます。

7.1. オンデマンドでのキューとトピックの作成

メッセージサーバーによっては、キューとトピックのオンデマンド作成をサポートします。送信側または受信側が割り当てられている場合、サーバーは送信側ターゲットアドレスまたは受信側ソースアドレスを使用して、アドレスに一致する名前でキューまたはトピックを作成します。

メッセージサーバーは通常、キュー (1 対 1 のメッセージ配信用) またはトピック (1 対多のメッセージ配信用) を作成します。クライアントは、ソースまたはターゲットに **queue** または **topic** 機能を設定してどちらを優先するかを示すことができます。

キューまたはトピックセマンティクスを選択するには、以下の手順に従います。

1. キューとトピックを自動的に作成するようにメッセージサーバーを設定します。多くの場合、これがデフォルト設定になります。

2. 以下の例のように、送信者ターゲットまたは受信者ソースに **queue** または **topic** 機能を設定します。

例: オンデマンドで作成されたキューへの送信

```
Target target = new Target() {
    Address = "jobs",
    Capabilities = new Symbol[] {"queue"},
};

SenderLink sender = new SenderLink(session, "s1", target, null);
```

例: オンデマンドで作成されたトピックからの受信

```
Source source = new Source() {
    Address = "notifications",
    Capabilities = new Symbol[] {"topic"},
};

ReceiverLink receiver = new ReceiverLink(session, "r1", source, null);
```

詳細は、以下の例を参照してください。

- QueueSend.cs
- QueueReceive.cs
- TopicSend.cs
- TopicReceive.cs

7.2. 永続サブスクリプションの作成

永続サブスクリプションは、メッセージの受信側を表すリモートサーバーの状態です。通常、メッセージ受信者は、クライアントが終了すると、破棄されます。ただし、永続サブスクリプションは永続的であるため、クライアントはそれらのサブスクリプションの割り当てを解除してから、後で再度アタッチできます。デタッチ時に受信したすべてのメッセージは、クライアントの再割り当て時に利用できます。

永続サブスクリプションは、クライアントコンテナー ID とレシーバー名を組み合わせてサブスクリプション ID を形成することで一意に識別されます。これらには、サブスクリプションを回復できるように、安定した値が必要です。

永続サブスクリプションを作成するには、以下の手順に従います。

1. 接続コンテナー ID を **client-1** などの安定した値に設定します。

```
Connection conn = new Connection(new Address(connUrl),
                                 SaslProfile.Anonymous,
                                 new Open() { ContainerId = "client-1" },
                                 null);
```

2. **Durable** プロパティーおよび **ExpiryPolicy** プロパティーを設定して、受信側ソースで持続性を設定します。

```
Source source = new Source()
{
    Address = "notifications",
    Durable = 2,
    ExpiryPolicy = new Symbol("never"),
};
```

3. **sub-1** などの安定した名前で受信側を作成し、ソースプロパティーを適用します。

```
ReceiverLink receiver = new ReceiverLink(session, "sub-1", source, null);
```

サブスクリプションからデタッチするには、受信側を明示的に閉じることなく接続を閉じます。サブスクリプションを終了するには、受信側を直接閉じます。

詳細は、DurableSubscribe.cs の例 を参照してください。

7.3. 共有サブスクリプションの作成

共有サブスクリプションとは、1 つ以上のメッセージレシーバーを表すリモートサーバーの状態のことです。このサブスクリプションは共有されているため、複数のクライアントが同じメッセージのストリームから消費できます。

クライアントは、受信者のソースに**shared** 機能を設定して、共有サブスクリプションを設定します。

共有サブスクリプションは、クライアントコンテナー ID とレシーバー名を組み合わせてサブスクリプション ID を形成することで一意に識別されます。複数のクライアントプロセスで同じサブスクリプションを特定できるように、これらに安定した値を指定する必要があります。**shared** に加えて **global** 機能が設定されている場合、サブスクリプション識別に受信者名だけが使用されます。

共有サブスクリプションを作成するには、以下の手順に従います。

1. 接続コンテナー ID を **client-1** などの安定した値に設定します。

```
Connection conn = new Connection(new Address(connUrl),
                                 SaslProfile.Anonymous,
                                 new Open() { ContainerId = "client-1" },
                                 null);
```

2. **shared** 機能を設定して、共有用の受信側ソースを設定します。

```
Source source = new Source()
{
   Address = "notifications",
   Capabilities = new Symbol[] {"shared"},
};
```

3. **sub-1** などの安定した名前で受信側を作成し、ソースプロパティーを適用します。

```
ReceiverLink receiver = new ReceiverLink(session, "sub-1", source, null);
```

詳細は、SharedSubscribe.cs の例 を参照してください。

第8章 メッセージ配信

8.1. メッセージの送信

メッセージを送信するには、接続、セッション、および送信者のリンクを作成し、**Message** オブジェクトで **Sender.Send()** メソッドを呼び出します。

例: メッセージの送信

```
Connection connection = new Connection(new Address("amqp://example.com"));
Session session = new Session(connection);
SenderLink sender = new SenderLink(session, "sender-1", "jobs");

Message message = new Message("job-content");
sender.Send(message);
```

詳細は、Send.cs の例 を参照してください。

8.2. メッセージの受信

メッセージを受信するには、接続、セッション、および受信側リンクを作成し、**Receiver.Receive()** メソッドを呼び出して返された **Message** オブジェクトを使用します。

例: メッセージの受信

```
Connection connection = new Connection(new Address("amqp://example.com"));
Session session = new Session(connection);
ReceiverLink receiver = new ReceiverLink(session, "receiver-1", "jobs");

Message message = receiver.Receive();
receiver.Accept(message);
```

Receiver.Accept() 呼び出しは、メッセージを受信および処理したリモートピアに指示します。

詳細は、Receive.cs の例 を参照してください。

第9章 ロギング

ロギングはトラブルシューティングおよびデバッグで重要です。デフォルトでは、ロギングがオフになっています。ロギングを有効にするには、ログレベルを設定し、ログメッセージを受信する委譲関数を提供する必要があります。

9.1. ログ出力レベルの設定

ライブラリーはさまざまなレベルでログトレースを出力します。

- Error
- Warning
- Information
- Verbose

最も低いログレベルである **Error** は、エラーイベントのみを追跡し、最少限のログメッセージを生成します。それより高いログレベルには、下位のログレベルがすべて含まれ、大量のログメッセージを生成します。

```
// Enable Error logs only.
Trace.TraceLevel = TraceLevel.Error
```

```
// Enable Verbose logs. This includes logs at all log levels.
Trace.TraceLevel = TraceLevel.Verbose
```

9.2. プロトコルロギングの有効化

ログレベルが **Frame** の場合、処理は異なります。トレースレベルを **Frame** に設定すると、AMQP プロトコルヘッダーおよびフレームの出力を追跡できます。

他のログレベルのいずれかでトレースする場合、通常のトレース出力と AMQP フレームトレースを同時に取得するためには、論理的に **Frame** との ORed である必要があります。以下に例を示します。

```
// Enable just AMQP frame tracing
Trace.TraceLevel = TraceLevel.Frame;
```

```
// Enable AMQP Frame logs, and Warning and Error logs
Trace.TraceLevel = TraceLevel.Frame | TraceLevel.Warning;
```

以下のコードは、AMQP フレームをコンソールに書き込みます。

例: ロギング委譲

```
Trace.TraceLevel = TraceLevel.Frame;
Trace.TraceListener = (f, a) => Console.WriteLine(
      DateTime.Now.ToString("[hh:mm:ss.fff]") + " " + string.Format(f, a));
```

第10章 相互運用性

この章では、AMQ .NET を他の AMQ コンポーネントと組み合わせて使用する方法を説明します。AMQ コンポーネントの互換性の概要は、製品の概要 を参照してください。

10.1. 他の AMQP クライアントとの相互運用

AMQP メッセージは AMQP タイプシステム を使用して設定されます。このような一般的な形式は、異なる言語の AMQP クライアントが相互に対話できる理由の 1 つです。

メッセージを送信する場合、AMQ .NET は自動的に言語ネイティブの型を AMQP でエンコードされたデータに変換します。メッセージの受信時に、リバース変換が行われます。

注記

AMQP タイプの詳細は、Apache Qpid プロジェクトによって維持される インタラクティブタイプのリファレンス を参照してください。

表10.1 AMQP 型

AMQP 型	説明
null	空の値
boolean	true または false の値
char	単一の Unicode 文字
string	Unicode 文字のシーケンス
binary	バイトのシーケンス
byte	署名済み 8 ビット整数
short	署名済み 16 ビット整数
int	署名済み 32 ビット整数
long	署名済み 64 ビット整数
ubyte	署名なしの 8 ビット整数
ushort	署名なしの 16 ビット整数
uint	署名なしの 32 ビット整数
ulong	署名なしの 64 ビット整数
float	32 ビット浮動小数点数

AMQP 型	説明
double	64 ビット浮動小数点数
array	単一型の値シーケンス
list	変数型の値シーケンス
map	異なるキーから値へのマッピング
uuid	ユニバーサルー意識別子
symbol	制限されたドメインからの 7 ビットの ASCII 文字列
timestamp	絶対的な時点

表10.2 エンコード前およびデコード後における AMQ .NET タイプ

AMQP 型	エンコード前の AMQ .NET タイプ	デコード後の AMQ .NET タイプ
null	**null**	**null**
boolean	**System.Boolean**	**System.Boolean**
char	**System.Char**	**System.Char**
string	**system.String**	**system.String**
binary	**System.Byte[]**	**System.Byte[]**
byte	**system.SByte**	**system.SByte**
short	**System.Int16**	**System.Int16**
int	**System.Int32**	**System.Int32**
long	**System.Int64**	**System.Int64**
ubyte	**System.Byte**	**System.Byte**
ushort	**System.UInt16**	**System.UInt16**
uint	**System.UInt32**	**System.UInt32**
ulong	**System.UInt64**	**System.UInt64**

AMQP 型	エンコード前の AMQ .NET タイプ	デコード後の AMQ .NET タイプ
float	System.Single	System.Single
double	system.Double	system.Double
list	Amqp.List	Amqp.List
map	Amqp.Map	Amqp.Map
uuid	System.Guid	System.Guid
symbol	Amqp.Symbol	Amqp.Symbol
timestamp	System.DateTime	System.DateTime

表10.3 AMQ .NET およびその他の AMQ クライアントタイプ (1/2)

エンコード前の AMQ .NET タイプ	AMQ C++ タイプ	AMQ JavaScript タイプ
null	nullptr	null
System.Boolean	bool	boolean
System.Char	wchar_t	number
system.String	std::string	string
System.Byte[]	proton::binary	string
system.SByte	int8_t	number
System.Int16	int16_t	number
System.Int32	int32_t	number
System.Int64	int64_t	number
System.Byte	uint8_t	number
System.UInt16	uint16_t	number
System.UInt32	uint32_t	number
System.UInt64	uint64_t	number

エンコード前の AMQ .NET タイプ	AMQ C++ タイプ	AMQ JavaScript タイプ
System.Single	float	number
system.Double	double	number
Amqp.List	std::vector	Array
Amqp.Map	std::map	object
System.Guid	proton::uuid	number
Amqp.Symbol	proton::symbol	string
System.DateTime	proton::timestamp	number

表10.4 AMQ .NET およびその他の AMQ クライアントタイプ (2/2)

エンコード前の AMQ .NET タイプ	AMQ Python タイプ	AMQ Ruby タイプ
null	None	nil
System.Boolean	bool	true, false
System.Char	unicode	String
System.String	unicode	String
System.Byte[]	bytes	String
System.SByte	int	Integer
System.Int16	int	Integer
System.Int32	long	Integer
System.Int64	long	Integer
System.Byte	long	Integer
System.UInt16	long	Integer
System.UInt32	long	Integer
System.UInt64	long	Integer

エンコード前の AMQ .NET タイプ	AMQ Python タイプ	AMQ Ruby タイプ
System.Single	**float**	**Float**
System.Double	**float**	**Float**
Amqp.List	**list**	**Array**
Amqp.Map	**dict**	**Hash**
System.Guid	-	-
Amqp.Symbol	**str**	**Symbol**
System.DateTime	**long**	**Time**

10.2. AMQ JMS での相互運用

AMQP は JMS メッセージングモデルへの標準マッピングを定義します。このセクションでは、そのマッピングのさまざまな側面について説明します。詳細は、AMQ JMS 相互運用性 の章を参照してください。

JMS メッセージタイプ

AMQ .NET は、本文タイプが異なる、単一のメッセージを提供します。一方、JMS API は異なるメッセージタイプを使用してさまざまな種類のデータを表します。次の表は、特定の本文タイプが JMS メッセージタイプにどのようにマップされるかを示しています。

作成される JMS メッセージタイプをさらに明示的に制御するには、**x-opt-jms-msg-type** メッセージアノテーションを設定できます。詳細は、AMQ JMS 相互運用性 の章を参照してください。

表10.5 AMQ .NET および JMS メッセージタイプ

AMQ .NET ボディータイプ	JMS メッセージタイプ
System.String	**TextMessage**
null	**TextMessage**
System.Byte[]	**BytesMessage**
それ以外のタイプ	**ObjectMessage**

10.3. AMQ BROKER への接続

AMQ Broker は AMQP 1.0 クライアントと相互運用するために設計されています。以下を確認して、ブローカーが AMQP メッセージング用に設定されていることを確認します。

- ネットワークファイアウォールのポート 5672 が開いている。

- AMQ Broker AMQP アクセプターが有効になっている。デフォルトのアクセプター設定 を参照してください。
- 必要なアドレスがブローカーに設定されている。アドレス、キュー、およびトピック を参照してください。
- ブローカーはクライアントからのアクセスを許可するように、クライアントは必要なクレデンシャルを送信するように設定されている。ブローカーのセキュリティー を参照してください。

10.4. AMQ INTERCONNECT への接続

AMQ Interconnect は AMQP 1.0 クライアントであれば機能します。以下をチェックして、コンポーネントが正しく設定されていることを確認します。

- ネットワークファイアウォールのポート 5672 が開いている。
- ルーターはクライアントからのアクセスを許可するように、クライアントは必要なクレデンシャルを送信するように設定されます。ネットワーク接続のセキュリティー保護 を参照してください。

付録A 証明書の管理

A.1. 認証局証明書のインストール

SSL/TLS 認証は、信頼できる認証局 (CA) が発行するデジタル証明書に依存します。クライアントによって SSL/TLS 接続が確立されると、AMQP ピアはサーバー証明書をクライアントに送信します。このサーバー証明書は、クライアントの **Trusted Root Certification Authorities** 証明書ストアの CA のいずれかで署名する必要があります。

ユーザーが Red Hat AMQ Broker で使用する自己署名証明書を作成する場合、ユーザーは証明書に署名するために CA を作成する必要があります。次に、自己署名の CA ファイル **ca.crt** をインストールして、クライアント SSL/TLS ハンドシェイクを有効にできます。

1. 管理者コマンドプロンプトから、MMC Certificate Manager プラグイン **certmgr.msc** を実行します。
2. 左側の **Trusted Root Certification Authorities** ディレクトリーを展開して、**Certificates** を公開します。
3. **Certificates** を右クリックし、**All Tasks** を選択してから **Import** を選択します。
4. **Next** をクリックします。
5. ファイル **ca.crt** を参照します。
6. **Next** をクリックします。
7. **Place all certificates in the following store**を選択します。
8. 証明書ストアの **Trusted Root Certification Authorities** を選択します。
9. **Next** をクリックします。
10. **Finish** をクリックします。

証明書のインストールに関する詳細は、 Microsoft Certificate Services および SSL の管理 を参照してください。

A.2. クライアント証明書のインストール

SSL/TLS およびクライアントの証明書を使用するには、クライアントの秘密鍵を持つ証明書を、クライアントシステムの適切な証明書ストアにインポートする必要があります。

1. 管理者コマンドプロンプトから、MMC Certificate Manager プラグイン **certmgr.msc** を実行します。
2. 左側の **Personal** ディレクトリーを展開して **証明書** を公開します。
3. **Certificates** を右クリックし、**All Tasks** を選択してから **Import** を選択します。
4. **Next** をクリックします。
5. **Browse** をクリックします。
6. ファイルタイププルダウンで、**Personal Information Exchange(\.pfx;*.p12)** を選択します。

7. ファイル **client.p12** を選択し、Open をクリックします。
8. **Next** をクリックします。
9. 秘密鍵のパスワードフィールドで、パスワードを入力します。デフォルトのインポートオプションを受け入れます。
10. **Next** をクリックします。
11. **Place all certificates in the following store**を選択します。
12. 証明書ストア **Personal** を選択します。
13. **Next** をクリックします。
14. **Finish** をクリックします。

A.3. クライアント証明書を使用した HELLO WORLD

クライアントがブローカーに証明書を返す前に、AMQ .NET ライブラリーに、使用する証明書を提供する必要があります。クライアント証明書ファイル **client.crt** が、**SChannel** 接続の起動時に使用される証明書の一覧に追加されます。

```
factory.SSL.ClientCertificates.Add(
    X509Certificate.CreateFromCertFile(certfile)
    );
```

この例では、**certfile** は、Personal 証明書ストアにインストールされている **client.p12** 証明書への完全パスです。完全な例は **HelloWorld-client-certs.cs** にあります。このソースファイルとサポートするプロジェクトファイルは SDK で利用できます。

付録B プログラムの例

B.1. 前提条件

- **amq.topic** および **service_queue** という名前のキューと、その読み取り/書き込みパーミッションがある Red Hat AMQ Broker。その場合、ブローカーは IP アドレス **10.10.1.1** にあります。
- 適切なパーミッションを持つソースおよびターゲット名 **amq.topic** を持つ Red Hat AMQ Interconnect。この場合、ルーターは IP アドレス **10.10.2.2** にあります。

すべての例は、**<ihnstall-dir>\bin\Debug** から実行されます。

B.2. HELLOWORLD SIMPLE

HelloWorld-simple は、同じアドレスに Sender と Receiver を作成する簡単な例です。アドレスにメッセージを送信し、アドレスからメッセージを読み取り、結果を出力します。

HelloWorld-simple コマンドラインオプション

```
Command line:
  HelloWorld-simple [brokerUrl [brokerEndpointAddress]]
Default:
  HelloWorld-simple amqp://localhost:5672 amq.topic
```

HelloWorld-simple サンプル呼び出し

```
$ HelloWorld-simple
Hello world!
```

デフォルトでは、このプログラムは localhost:5672 で実行しているブローカーに接続します。コマンドラインで、ホスト、ポート、および AMQP エンドポイントアドレスを明示的に指定します。

```
$ HelloWorld-simple amqp://someotherhost.com:5672 endpointname
```

デフォルトでは、このプログラムはメッセージの宛先を **amq.topic** に指定します。一部の Amqp ブローカー amq.topic は事前に定義されたエンドポイントアドレスで、ブローカー設定なしですぐに利用できます。このアドレスがブローカーに存在しない場合は、ブローカー管理ツールを使用してこれを作成します。

B.3. HELLOWORLD ROBUST

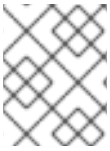
HelloWorld-robust は、簡単な例のすべての機能と、追加オプションと処理を共有します。

- 単純なペイロード以外のメッセージプロパティーへのアクセス。
  - Header
  - DeliveryAnnotations
  - MessageAnnotations
  - Properties

    - ApplicationProperties
    - BodySection
    - Footer
- 接続シャットダウンシーケンス

HelloWorld-robust コマンドラインオプション

```
Command line:
  HelloWorld-robust [brokerUrl [brokerEndpointAddress [payloadText [enableTrace]]]]
Default:
  HelloWorld-robust amqp://localhost:5672 amq.topic "Hello World"
```



注記

enableTrace 引数を簡単に存在させると、トレースが有効になります。引数には任意の値を保持できます。

HelloWorld-robust サンプル呼び出し

```
$ HelloWorld-robust
Broker: amqp://localhost:5672, Address: amq.topic, Payload: Hello World!
body:Hello World!
```

HelloWorld-robust を使用すると、ユーザーはペイロード文字列を指定でき、トレースプロトコルのロギングを有効にできます。

```
$ HelloWorld-robust amqp://localhost:5672 amq.topic "My Hello" loggingOn
```

B.4. INTEROP.DRAIN.CS、INTEROP.SPOUT.CS (パフォーマンス演習)

AMQ .NET サンプル **Interop.Drain** および **Interop.Spout** は、Red Hat AMQ Interconnect との対話について説明しています。この場合、メッセージブローカーはありません。代わりに、Red Hat AMQ Interconnect はクライアントプログラムが要求したアドレスを登録し、それらの間でメッセージをルーティングします。

Interop.Drain コマンドラインオプション

```
$ Interop.Drain.exe --help
Usage: interop.drain [OPTIONS] --address STRING
Create a connection, attach a receiver to an address, and receive messages.

Options:
 --broker [amqp://guest:guest@127.0.0.1:5672] - AMQP 1.0 peer connection address
 --address STRING     []      - AMQP 1.0 terminus name
 --timeout SECONDS    [1]     - time to wait for each message to be received
 --forever            [false] - use infinite receive timeout
 --count INT          [1]     - receive this many messages and exit; 0 disables count based exit
 --initial-credit INT [10]    - receiver initial credit
 --reset-credit INT   [5]     - reset credit to initial-credit every reset-credit messages
 --quiet              [false] - do not print each message's content
```

```
 --help                    - print this message and exit

Exit codes:
 0 - successfully received all messages
 1 - timeout waiting for a message
 2 - other error
```

Interop.Spout コマンドラインオプション

```
$ interop.spout --help
Usage: Interop.Spout [OPTIONS] --address STRING
Create a connection, attach a sender to an address, and send messages.

Options:
 --broker [amqp://guest:guest@127.0.0.1:5672] - AMQP 1.0 peer connection address
 --address STRING  []      - AMQP 1.0 terminus name
 --timeout SECONDS [0]     - send for N seconds; 0 disables timeout
 --durable         [false] - send messages marked as durable
 --count INT       [1]     - send this many messages and exit; 0 disables count based exit
 --id STRING       [guid]  - message id
 --replyto STRING  []      - message ReplyTo address
 --content STRING  []      - message content
 --print           [false] - print each message's content
 --help                    - print this message and exit

Exit codes:
 0 - successfully received all messages
 2 - other error
```

Interop.Spout および Interop.Drain サンプル呼び出し

1 つのウィンドウで Interop.drain を実行します。Drain は、1 つのメッセージが到達するまで待機します。

```
$ Interop.Drain.exe --broker amqp://10.10.2.2:5672 --forever --count 1 --address amq.topic
```

別のウィンドウで Interop.spout を実行します。Spout はメッセージをブローカーアドレスに送信し、終了します。

```
$ interop.spout --broker amqp://10.10.2.2:5672 --address amq.topic
$
```

これで、最初のウィンドウで drain は spout からメッセージを受信し、終了します。

```
$ Interop.Drain.exe --broker amqp://10.10.2.2:5672 --forever --count 1 --address amq.topic
Message(Properties=properties(message-id:9803e781-14d3-4fa7-8e39-c65e18f3e8ea:0),
ApplicationProperties=, Body=
$
```

B.5. INTEROP.CLIENT、INTEROP.SERVER (REQUEST-RESPONSE)

この例は、クライアントから文字列を受け入れ、大文字に変換し、クライアントに送信する単純なブローカーベースのサーバーを示しています。これは、2 つのコンポーネントで設定されます。

- client - poetry の行をサーバーに送信し、応答を出力します。

- server - 受信文字列を大文字に変換し、リクエスト元に返す簡単なサービス。

この例では、サーバーとクライアントは **service_queue** という名前のブローカーのサービスエンドポイントを共有します。サーバーはサービスエンドポイントでメッセージをリッスンします。クライアントは、一時的な動的 ReplyTo キューを作成し、一時的な名前をリクエストに埋め込み、リクエストをサーバーに送信します。各リクエストを受信して処理した後、サーバーはクライアントの一時的な ReplyTo アドレスに返信を送信します。

Interop.Client コマンドラインオプション

```
Command line:
  Interop.Client [peerURI [loopcount]]
Default:
  Interop.Client amqp://guest:guest@localhost:5672 1
```

Interop.Server コマンドラインオプション

```
Command line:
  Interop.Server [peerURI]
Default:
  Interop.Server amqp://guest:guest@localhost:5672
```

Interop.Client、Interop.Server サンプル呼び出し

プログラムは、以下のコマンドラインで起動できます。

```
$ Interop.Server.exe amqp://guest:guest@localhost:5672
$ Interop.Client.exe amqp://guest:guest@localhost:5672
```

PeerToPeer.Server は、コマンドラインで指定されたアドレスにリスナーを作成します。このアドレスは、受信接続をリッスンする **ContainerHost** クラスオブジェクトを初期化します。受信メッセージは非同期で **RequestProcessor** クラスオブジェクトに転送されます。

PeerToPeer.Client はサーバーへの接続を開き、サーバーへのメッセージの送信を開始します。

PeerToPeer.Client コマンドラインオプション

```
Command line:
  PeerToPeer.Client [peerURI]
Default:
  PeerToPeer.Client amqp://guest:guest@localhost:5672
```

PeerToPeer.Server コマンドラインオプション

```
Command line:
  PeerToPeer.Server [peerURI]
Default:
  PeerToPeer.Server amqp://guest:guest@localhost:5672
```

PeerToPeer.Client、PeerToPeer.Server サンプル呼び出し

1 つのウィンドウで PeerToPeer.Server を実行します。

```
$ PeerToPeer.Server.exe
Container host is listening on 127.0.0.1:5672
Request processor is registered on request_processor
```

```
Press enter key to exist...
Received a request hello 0
...
```

別のウィンドウで PeerToPeer.Client を実行します。PeerToPeer.Client は、サーバーにメッセージを送信し、受信時に応答を出力します。

```
$ PeerToPeer.Client.exe
Running request client...
Sent request properties(message-id:command-request,reply-to:client-57db8f65-6e3d-474c-a05e-8ca63b69d7c0) body hello 0
Received response:  body reply0
Received response:  body reply1
^C
```

付録C サブスクリプションの使用

AMQ は、ソフトウェアサブスクリプションから提供されます。サブスクリプションを管理するには、Red Hat カスタマーポータルでアカウントにアクセスします。

C.1. アカウントへのアクセス

手順

1. access.redhat.com に移動します。
2. アカウントがない場合は、作成します。
3. アカウントにログインします。

C.2. サブスクリプションのアクティベート

手順

1. access.redhat.com に移動します。
2. **My Subscriptions** に移動します。
3. **Activate a subscription** に移動し、16 桁のアクティベーション番号を入力します。

C.3. リリースファイルのダウンロード

.zip、.tar.gz およびその他のリリースファイルにアクセスするには、カスタマーポータルを使用してダウンロードする関連ファイルを検索します。RPM パッケージまたは Red Hat Maven リポジトリーを使用している場合は、この手順は必要ありません。

手順

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **JBOSS INTEGRATION AND AUTOMATION**カテゴリーの **Red Hat AMQ** エントリーを見つけます。
3. 必要な AMQ 製品を選択します。**Software Downloads** ページが開きます。
4. コンポーネントの **Download** リンクをクリックします。

C.4. パッケージ用システムの登録

この製品の RPM パッケージを Red Hat Enterprise Linux にインストールするには、システムが登録されている必要があります。ダウンロードしたリリースファイルを使用している場合は、この手順は必要ありません。

手順

1. access.redhat.com に移動します。

2. **Registration Assistant** に移動します。
3. ご使用の OS バージョンを選択し、次のページに進みます。
4. システムの端末に一覧表示されたコマンドを使用して、登録を完了します。

システムを登録する方法は、以下のリソースを参照してください。

- Red Hat Enterprise Linux 7 - システム登録およびサブスクリプション管理
- Red Hat Enterprise Linux 8 - システム登録およびサブスクリプション管理

付録D 例で AMQ ブローカーの使用

AMQ .NET の例では、名前が **amq.topic** というキューが含まれる実行中のメッセージブローカーが必要です。以下の手順に従って、ブローカーをインストールして起動し、キューを定義します。

D.1. ブローカーのインストール

Getting Started with AMQ Brokerの手順に従って、ブローカーをインストール して、ブローカーインスタンスを作成 します。匿名アクセスを有効にします。

以下の手順では、ブローカーインスタンスの場所を **<broker-instance-dir>** と呼びます。

D.2. ブローカーの起動

手順

1. **artemis run** コマンドを使用してブローカーを起動します。

```
$ <broker-instance-dir>/bin/artemis run
```

2. 起動時にログに記録された重大なエラーがないか、コンソールの出力を確認してください。ブローカーでは、準備が整うと **Server is now live** とログに記録されます。

```
$ example-broker/bin/artemis run

        __  __  ____    ____          _
      /\   |  \/  |/ __ \  |  _ \        | |
     /  \  | \  / | |  | | | |_) |_ __ ___ | | _____ _ __
    / /\ \ | |\/| | |  | | |  _ <| '__/ _ \| |/ / _ \ '__|
   / _____ \| |  | | |__| | | |_) | | | (_) |   <  __/ |
  /_/    \_\_|  |_|\____/  |____/|_|  \___/|_|\_\___|_|

   Red Hat AMQ <version>

2020-06-03 12:12:11,807 INFO  [org.apache.activemq.artemis.integration.bootstrap]
AMQ101000: Starting ActiveMQ Artemis Server
...
2020-06-03 12:12:12,336 INFO  [org.apache.activemq.artemis.core.server] AMQ221007:
Server is now live
...
```

D.3. キューの作成

新しいターミナルで、**artemis queue** コマンドを使用して **amq.topic** という名前のキューを作成します。

```
$ <broker-instance-dir>/bin/artemis queue create --name amq.topic --address amq.topic --auto-
create-address --anycast
```

プロンプトで質問に Yes または No で回答するように求められます。すべての質問に **N** (いいえ) と回答します。

キューが作成されると、ブローカーはサンプルプログラムで使用できるようになります。

D.4. ブローカーの停止

サンプルの実行が終了したら、**artemis stop** コマンドを使用してブローカーを停止します。

```
$ <broker-instance-dir>/bin/artemis stop
```

改訂日時: 2022-11-12 21:31:51 +1000