



**Red Hat AMQ 2021.Q3**

**AMQ .NET クライアントの使用**

AMQ Clients 2.10 向け



## Red Hat AMQ 2021.Q3 AMQ .NET クライアントの使用

---

AMQ Clients 2.10 向け

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

## 法律上の通知

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Using\_the\_AMQ\_.NET\_Client.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本ガイドでは、クライアントのインストールや設定、実例の実行、他の AMQ コンポーネントでのクライアントの使用方法について説明します。

## 目次

多様性を受け入れるオープンソースの強化 .....	4
<b>第1章 概要</b> .....	<b>5</b>
1.1. 主な特長 .....	5
1.2. サポート対象の標準およびプロトコル .....	5
1.3. サポートされる構成 .....	5
1.4. 用語および概念 .....	5
1.5. 本書の表記慣例 .....	6
sudo コマンド .....	6
ファイルパス .....	6
変数テキスト .....	6
<b>第2章 インストール</b> .....	<b>7</b>
2.1. 前提条件 .....	7
2.2. 「INSTALLING ON RED HAT ENTERPRISE LINUX」を参照してください。 .....	7
2.3. 「INSTALLING ON MICROSOFT WINDOWS」を参照してください。 .....	8
<b>第3章 スタートガイド</b> .....	<b>9</b>
3.1. 前提条件 .....	9
3.2. RED HAT ENTERPRISE LINUX で HELLOWORLD の実行 .....	9
3.3. MICROSOFT WINDOWS での HELLO WORLD の実行 .....	9
<b>第4章 例</b> .....	<b>10</b>
4.1. メッセージの送信 .....	10
サンプルの実行 .....	11
4.2. メッセージの受信 .....	11
サンプルの実行 .....	12
<b>第5章 ネットワーク接続</b> .....	<b>13</b>
5.1. 接続 URI .....	13
5.2. 再接続とフェイルオーバー .....	13
<b>第6章 セキュリティー</b> .....	<b>14</b>
6.1. ユーザーとパスワードを使用した接続 .....	14
6.2. SASL 認証の設定 .....	14
6.3. SSL/TLS トランスポートの設定 .....	14
<b>第7章 送信者およびレシーバー</b> .....	<b>15</b>
7.1. オンデマンドでキューとトピックの作成 .....	15
7.2. 永続サブスクリプションの作成 .....	16
7.3. 共有サブスクリプションの作成 .....	16
<b>第8章 メッセージ配信</b> .....	<b>18</b>
8.1. メッセージの送信 .....	18
8.2. メッセージの受信 .....	18
<b>第9章 ロギング</b> .....	<b>19</b>
9.1. ログ出力レベルの設定 .....	19
9.2. プロトコルロギングの有効化 .....	19
<b>第10章 相互運用性</b> .....	<b>20</b>
10.1. 他の AMQP クライアントとの相互運用 .....	20
10.2. AMQ JMS での相互運用 .....	24
JMS メッセージタイプ .....	24

10.3. AMQ BROKER への接続	24
10.4. AMQ INTERCONNECT への接続	25
<b>付録A 証明書の管理</b>	<b>26</b>
A.1. 証明書の認証局証明書のインストール	26
A.2. クライアント証明書のインストール	26
A.3. クライアント証明書を使用した HELLO WORLD	27
<b>付録B プログラムの例</b>	<b>28</b>
B.1. 前提条件	28
B.2. HELLOWORLD SIMPLE	28
HelloWorld-simple コマンドラインオプション	28
HelloWorld-simple サンプル呼び出し	28
B.3. HELLOWORLD ROBUST	28
HelloWorld-robust コマンドラインオプション	29
HelloWorld-robust サンプル呼び出し	29
B.4. INTEROP.DRAIN.CS、INTEROP.SPOUT.CS (パフォーマンス演習)	29
Interop.Drain コマンドラインオプション	29
Interop.Spout コマンドラインオプション	30
Interop.Spout および Interop.Drain サンプル呼び出し	30
B.5. INTEROP.CLIENT、INTEROP.SERVER (リクエスト - レスポンス)	30
Interop.Client コマンドラインオプション	31
Interop.Server コマンドラインオプション	31
Interop.Client、Interop.Server サンプル呼び出し	31
PeerToPeer.Client コマンドラインオプション	31
PeerToPeer.Server コマンドラインオプション	31
PeerToPeer.Client、PeerToPeer.Server サンプル呼び出し	31
<b>付録C サブスクリプションの使用</b>	<b>33</b>
C.1. アカウントへのアクセス	33
C.2. サブスクリプションのアクティベート	33
C.3. リリースファイルのダウンロード	33
C.4. パッケージを受信するためのシステムの登録	33
<b>付録D サンプルでの AMQ BROKER の使用</b>	<b>35</b>
D.1. ブローカーのインストール	35
D.2. ブローカーの起動	35
D.3. キューの作成	35
D.4. ブローカーの停止	36



## 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。これは大規模な取り組みであるため、これらの変更は今後の複数のリリースで段階的に実施されます。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。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、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である AMQ Clients の一部です。クライアントの概要は、「[AMQ Clients Overview](#)」を参照してください。本リリースに関する詳細は、『[AMQ Clients 2.10 Release Notes](#)』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 リファレンス](#) を参照してください。

## 1.1. 主な特長

- SSL/TLS でのセキュアな通信
- 柔軟な SASL 認証
- AMQP とネイティブデータ型間のシームレスな変換
- AMQP 1.0 のすべての機能と機能へのアクセス
- 完全な IntelliSense API ドキュメントを含む統合開発環境

## 1.2. サポート対象の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、および EXTERNAL
- IPv6 での最新の TCP

## 1.3. サポートされる構成

AMQ .NET でサポートされている設定については、Red Hat カスタマーポータル内の「[Red Hat AMQ 7 でサポートされる構成](#)」を参照してください。

## 1.4. 用語および概念

本セクションでは、コア API エンティティを紹介し、それらが一緒に操作する方法を説明します。

表1.1 API の用語

エンティティ	説明
接続	ネットワーク上の2つのピア間の通信用のチャネル

エンティティ	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
送信者のリンク	メッセージをターゲットに送信するためのチャンネル
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ンネル
ソース	メッセージに対する名前付きポイント
ターゲット	メッセージの名前付き宛先
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ピア間で転送されます。リンクはセッション上で確立されます。セッションはコネクション上で確立されます。

送信ピアは、メッセージを送信するために送信側のリンクを作成します。送信側には、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、メッセージを受信するための受信側リンクを作成します。受信側リンクには、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. 本書の表記慣例

### sudo コマンド

本書では、root 権限を必要とするコマンドには **sudo** が使用されています。何らか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

本書では、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

本書には、実際の環境に固有の値に置き換える必要がある変数を含むコードブロックが含まれています。変数テキストは中括弧で囲まれ、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、以下の例では、`<project-dir>` を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

本章では、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ビルドするには、Visual Studio をインストールする必要があります。

### 2.2. 「INSTALLING ON RED HAT ENTERPRISE LINUX」を参照してください。

#### 手順

1. ブラウザーを開き、[access.redhat.com/downloads](https://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** カテゴリで **Red Hat AMQ Client** 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 2.10.0 .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-2.10.0-dotnet-core.zip
```

.zip ファイルの内容を展開すると、**amq-clients-2.10.0-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、本書全体で **<install-dir>** と呼ばれます。

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** 行をそのファイルに追加します。

または、**<install-dir>/nupkg** ディレクトリー内の **.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. 「INSTALLING ON MICROSOFT WINDOWS」を参照してください。

### 手順

1. ブラウザーを開き、[access.redhat.com/downloads](https://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** カテゴリで **Red Hat AMQ Client** 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 2.10.0 .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 ファイルの内容を展開すると、**amq-clients-2.10.0-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、本書全体で **<install-dir>** と呼ばれます。

## 第3章 スタートガイド

本章では、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

- お使いの環境の [インストール](#) 手順を完了する必要があります。
- インターフェース **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、「[Creating a queue](#)」を参照してください。

### 3.2. RED HAT ENTERPRISE LINUX で HELLOWORLD の実行

Hello World の例では、ブローカーへの接続を作成し、グリーティングが含まれるメッセージを **amq.topic** キューに送信し、それを受け取ります。成功すると、受け取ったメッセージをコンソールに出力します。

`<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 での HELLO WORLD の実行

Hello World の例では、ブローカーへの接続を作成し、グリーティングが含まれるメッセージを **amq.topic** キューに送信し、それを受け取ります。成功すると、受け取ったメッセージをコンソールに出力します。

#### 手順

1. `<install-dir>` にアクセスし、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

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

詳細は、[AMQ .NET サンプルスイート](#) および [AMQP.Net Lite](#) の例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ます。ターゲット <address> の送信側は <message-body> が含まれるメッセージを送信し、接続を閉じて終了します。

#### 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 namespace 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、インターフェース、および値タイプを提供します。

**2** **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] **target** は、リモートホストのメッセージ宛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、メッセージが送信されることができるクライアント **SenderLink** です。リンクは、任意に **send-1** と命名します。お使いの環境で意味を持つリンク名を使用し、ビジーなシステムでのトラフィックを特定するのに役立ちます。リンク名は制限されませんが、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た後、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ていきます。

### 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>** のレシーバーを作成し、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); 7

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 namespace 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、インターフェース、および値タイプを提供します。
- 2 `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、メッセージを受信できるクライアント **ReceiverLink** です。そのリンクは、任意に **recv-1** と名付けられます。お使いの環境で意味を持つリンク名を使用し、ビジーなシステムでのトラフィックを特定するのに役立ちます。リンク名は制限されませんが、同じセッション内で一意である必要があります。
- 8 メッセージを受信されます。
- 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転送されます。
- 10 すべてのメッセージを受信した後、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ていきます。

### 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

本セクションでは、AMQP リモートピアへの接続に使用される Connection URI 文字列の標準的なフォーマットを説明します。

```
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 amqps** - 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** - 接続の転送先となる任意のネットワークポート。port 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 の例

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

### 5.2. 再接続とフェイルオーバー

AMQ.NET は再接続および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。コード例については、[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 の **user** フィールドおよび **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 **ユーザー** フィールドが存在すると、このエクスチェンジを制御します。**ユーザー** を指定すると、SASL 認証情報が交換されます。**ユーザー** 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 トランスポートの設定

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 送信者およびレシーバー

クライアントは送信側と受信側のリンクを使用して、メッセージの配信にチャンネルを表します。送信者と受信側は一方向で、メッセージの送信元はソースエンド、メッセージの送信先はターゲットエンドとなります。

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. オンデマンドでキューとトピックの作成

一部のメッセージサーバーは、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。送信側またはレシーバーが割り当てられている場合、サーバーは送信側の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を持つ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の場合) 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** 機能を設定することで、希望のものを指定できます。

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. キューとトピックの自動作成のために、メッセージサーバーを設定します。多くの場合、これはデフォルト設定になります。
2. 以下の例のように、送信側のターゲットまたは受信側ソースに **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: 要求時に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、クライアントが閉じられると、メッセージ受信側は破棄されます。ただし、永続サブスクリプションは永続的であるため、クライアントはこれらのサブスクリプションの割り当てを解除してから、後で再度アタッチすることができます。デタッチ中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 ID とレシーバー名を組み合わせることで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。サブスクリプションが回復できるようにするには、これらの値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 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** プロパティおよび **ExpiryPolicy** プロパティを設定して、受信側のソースを耐久性のあるものにします。

```
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作成し、ソースプロパティを適用します。

```
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。共有されているので、複数のクライアントは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、レシーバーソースに **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 ID とレシーバー名を組み合わせることで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見つけることができるように、これらの値が安定している必要があります。**shared** に加えて **global** 機能が設定されている場合、レシーバー名のみを使用してサブスクリプションを特定します。

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 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作成し、ソースプロパティを適用します。

```
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、コネクション、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、メッセージが受信および処理されたり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に重要です。デフォルトでは、ロギングは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 delegate 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

最低ログレベル **Error** は、エラーイベントのみをトレースし、最も少ないログメッセージを生成します。高いログレベルは、それ以下のすべてのログレベルを含み、より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

ログレベルの **Frame** の処理は異なります。トレースレベル **Frame** を設定すると、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には、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、**Frame** と論理的に ORed されている必要があります。以下は例になります。

```
// 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: 委譲のロギング

```
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

本章では、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](#) を使用して構成されます。この一般的な形式を使用するのは、異なる言語の AMQP クライアントが、相互運用できることが理由です。

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 エンコード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 のタイプに関する詳細は、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リファレンス](#) を参照してください。

表10.1 AMQP 型

AMQP 型	説明
<b>null</b>	空の値
<b>boolean</b>	true または false の値
<b>char</b>	単一の Unicode 文字
<b>string</b>	Unicode 文字のシーケンス
<b>binary</b>	バイト数のシーケンス
<b>byte</b>	署名済み 8 ビットの整数
<b>short</b>	署名済み 16 ビット整数
<b>int</b>	署名付き 32 ビット整数
<b>long</b>	署名済み 64 ビット整数
<b>ubyte</b>	署名なし 8 ビット整数
<b>ushort</b>	未署名の 16 ビット整数
<b>uint</b>	署名のない 32 ビット整数
<b>ulong</b>	未署名の 64 ビット整数
<b>float</b>	32 ビット浮動小数点数



AMQP 型	説明
<b>double</b>	64 ビット浮動小数点数
<b>array</b>	単一タイプの値シーケンス
<b>list</b>	変数タイプの値シーケンス
<b>map</b>	異なるキーから値へのマッピング
<b>uuid</b>	ユニバーサル一意識別子
<b>symbol</b>	制限されたドメインからの7ビットの ASCII 文字列
<b>timestamp</b>	絶対ポイント (時間単位)

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
<b>null</b>	<b>null</b>	<b>null</b>
<b>boolean</b>	<b>System.Boolean</b>	<b>System.Boolean</b>
<b>char</b>	<b>System.Char</b>	<b>System.Char</b>
<b>string</b>	<b>System.String</b>	<b>System.String</b>
<b>binary</b>	<b>System.Byte[]</b>	<b>System.Byte[]</b>
<b>byte</b>	<b>System.SByte</b>	<b>System.SByte</b>
<b>short</b>	<b>System.Int16</b>	<b>System.Int16</b>
<b>int</b>	<b>System.Int32</b>	<b>System.Int32</b>
<b>long</b>	<b>System.Int64</b>	<b>System.Int64</b>
<b>ubyte</b>	<b>System.Byte</b>	<b>System.Byte</b>
<b>ushort</b>	<b>System.UInt16</b>	<b>System.UInt16</b>
<b>uint</b>	<b>System.UInt32</b>	<b>System.UInt32</b>
<b>ulong</b>	<b>System.UInt64</b>	<b>System.UInt64</b>

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
<b>System.Single</b>	<b>float</b>	<b>number</b>
<b>System.Double</b>	<b>double</b>	<b>number</b>
<b>Amqp.List</b>	<b>std::vector</b>	<b>Array</b>
<b>Amqp.Map</b>	<b>std::map</b>	<b>object</b>
<b>System.Guid</b>	<b>proton::uuid</b>	<b>number</b>
<b>Amqp.Symbol</b>	<b>proton::symbol</b>	<b>string</b>
<b>System.DateTime</b>	<b>proton::timestamp</b>	<b>number</b>

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 のタイプ	AMQ Ruby タイプ
<b>null</b>	<b>None</b>	<b>nil</b>
<b>System.Boolean</b>	<b>bool</b>	<b>true, false</b>
<b>System.Char</b>	<b>unicode</b>	<b>String</b>
<b>System.String</b>	<b>unicode</b>	<b>String</b>
<b>System.Byte[]</b>	<b>bytes</b>	<b>String</b>
<b>System.SByte</b>	<b>int</b>	<b>Integer</b>
<b>System.Int16</b>	<b>int</b>	<b>Integer</b>
<b>System.Int32</b>	<b>long</b>	<b>Integer</b>
<b>System.Int64</b>	<b>long</b>	<b>Integer</b>
<b>System.Byte</b>	<b>long</b>	<b>Integer</b>
<b>System.UInt16</b>	<b>long</b>	<b>Integer</b>
<b>System.UInt32</b>	<b>long</b>	<b>Integer</b>
<b>System.UInt64</b>	<b>long</b>	<b>Integer</b>

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

## 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的なマッピングを定義します。本項では、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](#)」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。以下の表は、特定のボディ型が JMS メッセージタイプにマッピングする方法を示しています。

結果として生成される JMS メッセージタイプの明示的な制御を行うために、**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 ボディタイプ	JMS メッセージタイプ
<b>System.String</b>	<b>TextMessage</b>
<b>null</b>	<b>TextMessage</b>
<b>System.Byte[]</b>	<b>BytesMessage</b>
それ以外のタイプ	<b>ObjectMessage</b>

## 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。以下をチェックして、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ます。「[デフォルトのアクセプター設定](#)」を参照してください。
- 必要なアドレスはブローカーで設定されます。「[Addresses, Queues, and Topics](#)」を参照してください。
- ブローカーはクライアントからアクセスを許可するよう設定され、クライアントは必要なクレデンシャルを送信するよう設定されます。[Broker Security](#) を参照してください。

## 10.4. AMQ INTERCONNECT への接続

AMQ Interconnect は AMQP 1.0 クライアントと動作します。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アクセスを許可するよう設定され、クライアントは必要なクレデンシャルを送信するよう設定されます。「[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. 証明書の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。クライアントが 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、クライアントの信頼されているルート認証局の証明書ストアの 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 で使用できるように自己署名証明書を作成する場合、ユーザーは証明書を署名するために 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することができます。

1. 管理者コマンドプロンプトで、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** を参照して選択します。
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、『[Managing Microsoft Certificate Services and SSL](#)』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で、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に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** は、**SChannel** 接続の起動時に使用される証明書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た **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** という名前のキューと、**service\_queue** という名前のキューがあり、どちらも読み取り/書き込みの権限を持っている Red Hat AMQ Broker。そのため、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が設定されたソースとターゲットの名前 **amq.topic** を持つ Red Hat AMQ Interconnect。この図については、ルーターは IP アドレス **10.10.2.2** にあります。

すべての例では、`<install-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を **amq.topic** に対応しています。一部の Amqp ブローカー amq.topic は事前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。引数はどの値も保持できます。

## 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 では、ユーザーが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することができます。

```
$ 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。その代わりに、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ち続けます。

```
$ 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 はスパウトからのメッセージを受け取り、その後終了します。

```
$ 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 (リクエスト - レスポンス)

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。これには2つのコンポーネントがあります。

- クライアント - 一行の詩をサーバーに送り、レスポンスを出力します。

- サーバー - 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、一時的に動的な 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、サーバーが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、次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、受信したレスポンスを出力します。

```
$ 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](https://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](https://access.redhat.com) に移動します。
2. サブスクリプション 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 リポジトリを使用している場合、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](https://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** カテゴリで **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. パッケージを受信するためのシステムの登録

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、お使いのシステムを登録する必要があります。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](https://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 サンプルでの AMQ BROKER の使用

AMQ .NET のサンプルでは、**amq.topic** という名前の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

『**AMQ Broker の使用**』の説明に従い **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、**<broker-instance-dir>** としてブローカーインスタンスの場所を参照します。

### 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. コンソールの出力で、起動時にログに記録される重要なエラーの有無を確認します。ブローカーは、準備が整う際に **Server is now live** をログに記録します。

```
$ 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
```

yes または no の質問への回答を求めるプロンプトが表示されます。そのすべてに no (**N**) 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と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: 2021-08-29 15:57:24 +1000