Red Hat AMQ 7.7

AMQ Ruby クライアントの使用

AMQ Clients 2.7 向け

Last Updated: 2023-02-03

Red Hat AMQ 7.7 AMQ Ruby クライアントの使用

AMQ Clients 2.7 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

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

第1章 概要

AMQ Ruby は、メッセージングアプリケーションを開発するためのライブラリーです。また、AMQP メッセージを送受信する Ruby アプリケーションを作成できます。

重要

AMQ Ruby クライアントは、テクノロジープレビュー機能に限定されます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

AMQ Ruby 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 2.7 リリースノート を参照してください。

AMQ Ruby は、Apache Qpid の Proton API をベースとしています。詳細な API ドキュメントは、 AMQ Ruby API リファレンス を参照してください。

1.1. 主な特長

- 既存のアプリケーションとの統合を簡素化するイベント駆動型の API
- セキュアな通信用の SSL/TLS
- 柔軟な SASL 認証
- 自動再接続およびフェイルオーバー
- AMQP と言語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

1.2. サポートされる標準およびプロトコル

AMQ Ruby 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
- ANONYMOUS、PLAIN、SCRAM、EXTERNAL、および GSSAPI (Kerberos) を含む、Cyrus SASL でサポートされる 単純な認証およびセキュリティーレイヤー (SASL) メカニズム
- IPv6 での最新の TCP

1.3. サポートされる構成

AMQ Ruby は、以下に示す OS と言語のバージョンをサポートしています。詳細は、Red Hat AMQ 7 Supported Configurations を参照してください。

- Ruby 2.0 を使用する Red Hat Enterprise Linux 7
- Ruby 2.5 を使用する Red Hat Enterprise Linux 8

AMQ Ruby は、次の AMQ コンポーネントおよびバージョンとの組み合わせでサポートされています。

- AMQ ブローカーのすべてのバージョン
- AMQ Interconnect のすべてのバージョン
- AMQ Online のすべてのバージョン
- A-MQ 6 バージョン 6.2.1 以降

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
Container	接続の最上位のコンテナー。
接続	ネットワーク上の 2 つのピア間の通信チャネル。これにはセッションが含まれます。
Session	メッセージの送受信を行うためのコンテキスト。送信者および受信者が含まれます。
sender	メッセージをターゲットに送信するためのチャネル。これにはターゲットがあります。
receiver	ソースからメッセージを受信するためのチャネル。これにはソースがあります。
Source	メッセージの名前付きの発信元。
Target	メッセージの名前付き受信先。
メッセージ	情報のアプリケーション固有の部分。
Delivery	メッセージの転送。

AMQ Ruby は **メッセージ** を送受信します。メッセージは、**senders** と **receivers** を介して、接続されたピアの間で転送されます。送信側および受信側は **セッション** 上で確立されます。セッションは**接続**上で確立されます。接続は、一意に識別された 2 つの **コンテナー** 間で 確立されます。コネクションには複数のセッションを含めることができますが、多くの場合、必要ありません。API を使用すると、セッションが必要でない限り、セッションを無視できます。

送信ピアは、メッセージ送信用の送信者を作成します。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。受信ピアは、メッセージ受信用の受信者を作成します。受信側には、リモートピアでキューまたはトピックを識別する **ソース** があります。

メッセージの送信は **配信** と呼ばれます。メッセージとは、送信される内容のことで、ヘッダーやアノテーションなどのすべてのメタデータが含まれます。配信は、そのコンテンツの移動に関連するプロトコルエクスチェンジです。

配信が完了したことを示すには、送信側または受信側セットのいずれかが解決します。送信側または受信側が解決されたことを知らせると、その配信の通信ができなくなります。受信側は、メッセージを受諾するか、拒否するかどうかを指定することもできます。

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
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

本章では、環境に AMQ Ruby 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。
- 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。
- AMQ Ruby を使用するには、お使いの環境に Ruby 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 を参照してください。

手順

1. **subscription-manager** コマンドを使用して、必要なパッケージリポジトリーをサブスクライブします。必要に応じて、**<variant>** を Red Hat Enterprise Linux のバリアントの値 (例えば、**server** または **workstation**) に置き換えます。

   Red Hat Enterprise Linux 7

   ```
   $ sudo subscription-manager repos --enable=amq-clients-2-for-rhel-7-<variant>-rpms
   ```

   Red Hat Enterprise Linux 8

   ```
   $ sudo subscription-manager repos --enable=amq-clients-2-for-rhel-8-x86_64-rpms
   ```

2. **yum** コマンドを使用して、**rubygem-qpid_proton** パッケージおよび **rubygem-qpid_proton-doc** パッケージをインストールします。

   ```
   $ sudo yum install rubygem-qpid_proton rubygem-qpid_proton-doc
   ```

パッケージの使用方法は、付録B *Red Hat Enterprise Linux パッケージの使用* を参照してください。

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

- 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。
- **examples** 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、グリーティングを含むメッセージを**examples**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をコンソールに出力します。

examples ディレクトリーに移動し、**helloworld.rb** の例を実行します。

```
$ cd /usr/share/proton/examples/ruby/
$ ruby helloworld.rb amqp://127.0.0.1 examples
Hello World!
```

第4章 例

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

その他の例については、AMQ Ruby サンプルスイート 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て接続を切断して終了します。

例: メッセージの送信

```
require 'qpid_proton'

class SendHandler < Qpid::Proton::MessagingHandler
  def initialize(conn_url, address, message_body)
    super()

    @conn_url = conn_url
    @address = address
    @message_body = message_body
  end

  def on_container_start(container)
    conn = container.connect(@conn_url)
    conn.open_sender(@address)
  end

  def on_sender_open(sender)
    puts "SEND: Opened sender for target address '#{sender.target.address}'\n"
  end

  def on_sendable(sender)
    message = Qpid::Proton::Message.new(@message_body)
    sender.send(message)

    puts "SEND: Sent message '#{message.body}'\n"

    sender.close
    sender.connection.close
  end
end

if ARGV.size == 3
  conn_url, address, message_body = ARGV
else
  abort "Usage: send.rb <connection-url> <address> <message-body>\n"
end

handler = SendHandler.new(conn_url, address, message_body)
container = Qpid::Proton::Container.new(handler)
container.run
```

サンプルの実行
サンプルプログラムを実行するには、サンプルプログラムをローカルファイルにコピーし、**ruby** コマンドを使用して呼び出します。

```
$ ruby send.rb amqp://localhost queue1 hello
```

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

```
require 'qpid_proton'

class ReceiveHandler < Qpid::Proton::MessagingHandler
  def initialize(conn_url, address, desired)
    super()

    @conn_url = conn_url
    @address = address

    @desired = desired
    @received = 0
  end

  def on_container_start(container)
    conn = container.connect(@conn_url)
    conn.open_receiver(@address)
  end

  def on_receiver_open(receiver)
    puts "RECEIVE: Opened receiver for source address '#{receiver.source.address}'\n"
  end

  def on_message(delivery, message)
    puts "RECEIVE: Received message '#{message.body}'\n"

    @received += 1

    if @received == @desired
      delivery.receiver.close
      delivery.receiver.connection.close
    end
  end
end

if ARGV.size > 1
  conn_url, address = ARGV[0..1]
else
  abort "Usage: receive.rb <connection-url> <address> [<message-count>]\n"
end

begin
  desired = Integer(ARGV[2])
```

```
rescue TypeError
  desired = 0
end

handler = ReceiveHandler.new(conn_url, address, desired)
container = Qpid::Proton::Container.new(handler)
container.run
```

サンプルの実行

サンプルプログラムを実行するには、サンプルプログラムをローカルファイルにコピーし、**ruby** コマンドを使用して呼び出します。

```
$ ruby receive.rb amqp://localhost queue1
```

第5章 ネットワーク接続

5.1. 接続 URL

接続 URL は、新規接続の確立に使用される情報をエンコードします。

接続 URL 構文

```
scheme://host[:port]
```

- **スキーム** - 暗号化されていない TCP の **amqp**、または SSL/TLS 暗号化のある TCP の **amqps** のいずれかの接続トランスポート。
- **ホスト** - リモートのネットワークホスト。値は、ホスト名または数値の IP アドレスの場合があります。IPv6 アドレスは角括弧で囲む必要があります。
- **ポート** - リモートネットワークポート。.この値はオプションです。デフォルト値は、**amqp** スキームの場合は 5672 で、**amqps** スキームの場合は 5671 です。

接続 URL サンプル

```
amqps://example.com
amqps://example.net:56720
amqp://127.0.0.1
amqp://[::1]:2000
```

第6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。

6.1. 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。

詳細は、以下の例を参照してください。

- queue-send.rb
- queue-receive.rb
- topic-send.rb
- topic-receive.rb

6.2. 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。

例

6.3. 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。

例

第7章 ロギング

7.1. プロトコルロギングの有効化

クライアントは AMQP プロトコルフレームをコンソールに記録できます。多くの場合、このデータは問題の診断時に重要になります。

プロトコルロギングを有効にするには、**PN_TRACE_FRM** 環境変数を **1** に設定します。

例: プロトコルロギングの有効化

```
$ export PN_TRACE_FRM=1
$ <your-client-program>
```

プロトコルロギングを無効にするには、**PN_TRACE_FRM** 環境変数の設定を解除します。

第8章 相互運用性

本章では、AMQ Ruby 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。

8.1. 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 Ruby 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リファレンスを参照してください。

表8.1 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8.2 エンコード前およびデコード後における AMQ Ruby タイプ

AMQP 型	エンコード前の AMQ Ruby タイプ	デコード後の AMQ Ruby タイプ
null	**nil**	**nil**
boolean	**true, false**	**true, false**
char	**-**	**String**
string	**String**	**String**
binary	**-**	**String**
byte	**-**	**Integer**
short	**-**	**Integer**
int	**-**	**Integer**
long	**Integer**	**Integer**
ubyte	**-**	**Integer**
ushort	**-**	**Integer**
uint	**-**	**Integer**
ulong	**-**	**Integer**

AMQP 型	エンコード前の AMQ Ruby タイプ	デコード後の AMQ Ruby タイプ
float	-	Float
double	Float	Float
array	-	Array
list	Array	Array
map	Hash	Hash
symbol	Symbol	Symbol
timestamp	Date, Time	Time

表8.3 AMQ Ruby 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 Ruby タイプ	AMQ C++ タイプ	AMQ JavaScript タイプ
nil	nullptr	null
true, false	bool	boolean
String	std::string	string
Integer	int64_t	number
Float	double	number
Array	std::vector	Array
Hash	std::map	object
Symbol	proton::symbol	string
Date, Time	proton::timestamp	number

表8.4 AMQ Ruby 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 Ruby タイプ	AMQ .NET タイプ	AMQ Python タイプ
nil	null	None
true, false	System.Boolean	bool

エンコード前の AMQ Ruby タイプ	AMQ .NET タイプ	AMQ Python タイプ
String	**System.String**	**unicode**
Integer	**System.Int64**	**long**
Float	**System.Double**	**float**
Array	**Amqp.List**	**list**
Hash	**Amqp.Map**	**dict**
Symbol	**Amqp.Symbol**	**str**
Date, Time	**System.DateTime**	**long**

8.2. 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。本セクションでは、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 Interoperability の章を参照してください。

JMS メッセージタイプ

AMQ Ruby 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 Interoperability の章を参照してください。

表8.5 AMQ Ruby および JMS メッセージタイプ

AMQ Ruby ボディータイプ	JMS メッセージタイプ
String	**TextMessage**
nil	**TextMessage**
-	**BytesMessage**
それ以外のタイプ	**ObjectMessage**

8.3. 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ます。Broker Security を参照してください。

8.4. 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 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。

A.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。

A.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. **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。

A.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。

A.4. パッケージ用のシステムの登録

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。

詳細は、How to Register and Subscribe a System to the Red Hat Customer Portal を参照してください。

付録B RED HAT ENTERPRISE LINUX パッケージの使用

本セクションでは、Red Hat Enterprise Linux の RPM パッケージとして配信されるソフトウェアを使用する方法を説明します。

B.1. 概要

ライブラリーやサーバーなどのコンポーネントには多くの場合、複数のパッケージが関連付けられています。それらをすべてインストールする必要はありません。必要なものだけをインストールできます。

プライマリーパッケージは、通常、追加の修飾子がない最もシンプルな名前です。このパッケージは、プログラムのランタイム時にコンポーネントを使用するために必要なすべてのインターフェイスを提供します。

-devel で終わる名前を持つパッケージには、C ライブラリーおよび C++ ライブラリーのヘッダーが含まれます。このパッケージに依存するプログラムを構築する際の、コンパイル時に必要になります。

-docs で終わる名前を持つパッケージには、コンポーネントのドキュメントとサンプルプログラムが含まれます。

RPM パッケージ を使用する方法は、以下のリソースのいずれかを参照してください。

- Red Hat Enterprise Linux 6 - ソフトウェアのインストールおよび管理
- Red Hat Enterprise Linux 7 - ソフトウェアのインストールおよび管理
- Red Hat Enterprise Linux 8 - ソフトウェアパッケージの管理

B.2. パッケージの検索

パッケージを検索するには、**yum search** コマンドを使用します。検索結果にはパッケージ名が含まれます。パッケージ名は、このセクションに記載されている他のコマンドで **<package>** の値として使用できます。

```
$ yum search <keyword>...
```

B.3. パッケージのインストール

パッケージをインストールするには、**yum install** コマンドを使用します。

```
$ sudo yum install <package>...
```

B.4. パッケージ情報のクエリー

システムにインストールされているパッケージを一覧表示するには、**rpm -qa** コマンドを使用します。

```
$ rpm -qa
```

特定のパッケージに関する情報を取得するには、**rpm -qi** コマンドを使用します。

```
$ rpm -qi <package>
```

パッケージに関連するファイルを一覧表示するには、**rpm -ql** コマンドを使用します。

```
$ rpm -ql <package>
```

付録C 例で AMQ ブローカーの使用

AMQ Ruby の例では、名前が **examples** 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。

C.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。

C.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** とログが記録されます。

```
$ 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
...
```

C.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 **examples** という名前のキューを作成します。

```
$ <broker-instance-dir>/bin/artemis queue create --name examples --address examples --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。

C.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：2023-01-28 12:24:39 +1000