Red Hat AMQ 2021.Q3

使用 AMQ .NET 客户端

用于 AMQ 客户端 2.10

Last Updated: 2023-02-20

Red Hat AMQ 2021.Q3 使用 AMQ .NET 客户端

用于 AMQ 客户端 2.10

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.

法律通告

摘要

本指南描述了如何安装和配置客户端，运行实践示例，并将您的客户端与其他 AMQ 组件搭配使用。

目录

使开源包含更多 4

第 1 章 概述 5
1.1. 主要特性 5
1.2. 支持的标准和协议 5
1.3. 支持的配置 5
1.4. 术语和概念 5
1.5. 文档惯例 6
sudo 命令 6
文件路径 6
变量文本 6

第 2 章 安装 7
2.1. 先决条件 7
2.2. 在 RED HAT ENTERPRISE LINUX 上安装 7
2.3. 在 MICROSOFT WINDOWS 上安装 7

第 3 章 入门 9
3.1. 先决条件 9
3.2. 在 RED HAT ENTERPRISE LINUX 上运行 HELLOWORLD 9
3.3. 在 MICROSOFT WINDOWS 上运行 HELLO WORLD 9

第 4 章 示例 10
4.1. 发送消息 10
运行示例 11
4.2. 接收消息 11
运行示例 12

第 5 章 网络连接 13
5.1. 连接 URI 13
5.2. 重新连接和故障转移 13

第 6 章 安全性 14
6.1. 使用用户和密码连接 14
6.2. 配置 SASL 身份验证 14
6.3. 配置 SSL/TLS 传输 14

第 7 章 发送者和接收方 15
7.1. 按需创建队列和主题 15
7.2. 创建持久订阅 15
7.3. 创建共享订阅 16

第 8 章 消息发送 18
8.1. 发送消息 18
8.2. 接收消息 18

第 9 章 日志记录 19
9.1. 设置日志输出级别 19
9.2. 启用协议日志记录 19

第 10 章 互操作性 20
10.1. 与其他 AMQP 客户端互操作 20
10.2. 使用 AMQ JMS 进行互操作 24
JMS 消息类型 24

10.3. 连接到 AMQ BROKER 24
10.4. 连接到 AMQ INTERCONNECT 25

附录 A. 管理证书 26
A.1. 安装证书颁发机构证书 26
A.2. 安装客户端证书 26
A.3. 使用客户端证书的 HELLO WORLD 27

附录 B. 示例程序 28
B.1. 先决条件 28
B.2. HELLOWORLD 简单 28
helloworld-simple 命令行选项 28
helloworld-simple 示例调用 28
B.3. HELLOWORLD 稳健 28
helloworld-robust 命令行选项 29
helloworld-robust 示例调用 29
B.4. INTEROP.DRAIN.CS、INTEROP.SPOUT.CS（性能练习器） 29
Interop.Drain 命令行选项 29
Interop.Spout 命令行选项 29
Interop.Spout 和 Interop.Drain 示例调用 30
B.5. INTEROP.CLIENT, INTEROP.SERVER(REQUEST-RESPONSE) 30
Interop.Client 命令行选项 30
Interop.Server 命令行选项 31
Interop.Client, Interop.Server 示例调用 31
peertopeer.Client 命令行选项 31
peertopeer.Server 命令行选项 31
peertopeer.Client、PeerToPeer.Server 示例调用 31

附录 C. 使用您的订阅 32
C.1. 访问您的帐户 32
C.2. 激活订阅 32
C.3. 下载发行文件 32
C.4. 为系统注册软件包 32

附录 D. 将 AMQ BROKER 与示例搭配使用 34
D.1. 安装代理 34
D.2. 启动代理 34
D.3. 创建队列 34
D.4. 停止代理 34

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 CTO Chris Wright 信息。

第 1 章 概述

AMQ .NET 是用于 .NET 平台的轻量级 AMQP 1.0 库。它可让您编写发送和接收 AMQP 消息的 .NET 应用程序。

AMQ .NET 是 AMQ 客户端的一部分，这是一套支持多种语言和平台的消息传递库。有关客户端的概述，请参阅 AMQ 客户端概述。有关此发行版本的详情，请参考 AMQ Clients 2.10 发行注记。

AMQ .NET 基于 AMQP.Net Lite。有关详细的 API 文档，请参阅 AMQ .NET API 参考。

1.1. 主要特性

- 用于安全通信的 SSL/TLS
- 灵活的 SASL 身份验证
- AMQP 和原生数据类型之间的无缝转换
- 访问 AMQP 1.0 的所有特性和功能
- 带有完整 *IntelliSense* API 文档的集成开发环境

1.2. 支持的标准和协议

AMQ .NET 支持以下业界认可的标准和网络协议：

- 高级消息队列协议(AMQP)版本 1.0
- 传输层安全(TLS)协议的版本 1.0、1.1、1.2 和 1.3，后跟 SSL
- 简单身份验证和安全层(SASL)机制 ANONYMOUS、PLAIN 和 EXTERNAL
- 使用 IPv6的现代 TCP

1.3. 支持的配置

如需当前与 AMQ .NET 支持的配置相关的信息，请参阅红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

1.4. 术语和概念

本节介绍核心 API 实体，并描述它们如何协同运作。

表 1.1. API 术语

实体	描述
连接	个网络上两个同级之间进行通信的频道
会话	用于发送和接收消息的上下文
发件人链接	用于向目标发送消息的频道

实体	描述
接收器链接	从源接收信息的频道
Source	消息的命名源点
目标	消息的命名目的地
消息	应用程序数据的可可变拥有者

AMQ .NET 发送并接收 *消息*。消息通过 *链路* 在连接的同级之间传输。通过 *会话* 建立链接。通过 *连接* 建立会话。

发送对等点会创建一个 *发送者链接* 来发送邮件。发件人链接有一个 *目标*，可在远程同级上标识队列或主题。接收客户端创建 *接收器链接* 来接收消息。接收器链路具有一个 *源*，用于标识远程对等点上的队列或主题。

1.5. 文档惯例

sudo 命令
在本文档中，**sudo** 用于任何需要 root 权限的命令。使用 **sudo** 时要小心，因为任何更改都可能会影响整个系统。有关 **sudo** 的详情请参考 使用 sudo 命令。

文件路径
在这个文档中，所有文件路径都对 Linux、UNIX 和类似操作系统有效（例如 **/home/andrea**）。在 Microsoft Windows 中，您必须使用等效的 Windows 路径（例如 **C:\Users\andrea**）。

变量文本
本文档包含代码块，它们需要使用特定于环境的值替换。变量文本括在箭头大括号内，样式为圆形单空间。例如，在以下命令中，将 ***<project-dir>*** 替换为您的环境的值：

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

第 2 章 安装

本章介绍了在您的环境中安装 AMQ .NET 的步骤。

2.1. 先决条件

- 您必须有 订阅 才能访问 AMQ 发行文件和存储库。
- 要在 Red Hat Enterprise Linux 中使用 AMQ .NET，您必须安装 .NET Core 3.1 开发人员工具。如需更多信息，请参阅 .NET Core 3.1 入门指南。
- 要在 Microsoft Windows 上使用 AMQ .NET 构建程序，您必须安装 Visual Studio。

2.2. 在 RED HAT ENTERPRISE LINUX 上安装

流程

1. 打开浏览器并登录红帽客户门户网站 **产品下载页面**，网址为 access.redhat.com/downloads。
2. 在 **INTEGRATION AND AUTOMATION** 类别中找到 **红帽 AMQ 客户端** 条目。
3. 单击 **Red Hat AMQ Clients**。此时会打开 **Software Downloads** 页面。
4. 下载 **AMQ 客户端 2.10.0 .NET Core**.zip 文件。
5. 使用 **unzip** 命令将文件内容提取到您选择的目录中。

```
$ unzip amq-clients-2.10.0-dotnet-core.zip
```

当您提取 .zip 文件的内容时，将创建一个名为 **amq-clients-2.10.0-dotnet-core** 的目录。这是安装的顶级目录，在整个文档中被称为 ***<install-dir>***。

6. 使用文本编辑器创建文件 **$HOME/.nuget/NuGet/NuGet.Config** 并添加以下内容：

```
<?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>
```

如果您已有 **NuGet.Config** 文件，请在其中添加 **amq-clients** 行。

另外，您还可以将 ***<install-dir>*/nupkg** 目录中的 .nupkg 文件移动到现有软件包源位置。

2.3. 在 MICROSOFT WINDOWS 上安装

流程

1. 打开浏览器并登录红帽客户门户网站 **产品下载页面**，网址为 access.redhat.com/downloads。
2. 在 **INTEGRATION AND AUTOMATION** 类别中找到 **红帽 AMQ 客户端** 条目。

3. 单击 **Red Hat AMQ Clients**。此时会打开 **Software Downloads** 页面。
4. 下载 **AMQ Clients 2.10.0 .NET**.zip 文件。
5. 通过右键单击 zip 文件并选择"**提取所有"，将文件内容提取到** 您选择的目录。

当您提取 .zip 文件的内容时，将创建一个名为 **amq-clients-2.10.0-dotnet** 的目录。这是安装的顶级目录，在整个文档中被称为 ***<install-dir>***。

第 3 章 入门

本章将引导您完成设置环境并运行简单消息传递程序的步骤。

3.1. 先决条件

- 您必须为您的环境完成 安装过程。
- 您必须有一个 AMQP 1.0 消息代理，侦听接口 **localhost** 和端口 **5672** 上的连接。它必须启用匿名访问。如需更多信息，请参阅 启动代理。
- 您必须有一个名为 **amq.topic** 的队列。如需更多信息，请参阅 创建队列。

3.2. 在 RED HAT ENTERPRISE LINUX 上运行 HELLOWORLD

Hello World 示例创建与代理的连接，发送一条包含问候语的消息到 **amq.topic** 队列，然后重新接收它。成功后，它会将收到的消息打印到控制台。

更改到 ***<install-dir>*/examples/netcoreapp3/HelloWorld-simple**，并使用 **dotnet run** 构建和执行程序。

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

3.3. 在 MICROSOFT WINDOWS 上运行 HELLO WORLD

Hello World 示例创建与代理的连接，发送一条包含问候语的消息到 **amq.topic** 队列，然后重新接收它。成功后，它会将收到的消息打印到控制台。

流程

1. 导航到 ***<install-dir>***，然后在 Visual Studio 中打开 **amqp.sln** 解决方案文件。
2. 从 **Build 菜单中选择 Build Solution** 以编译解决方案。
3. 打开命令提示窗口并执行以下命令发送和接收消息：

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

第 4 章 示例

本章介绍如何通过示例程序使用 AMQ .NET。

如需了解更多示例，请参阅 AMQ .NET 示例套件 和 AMQP.Net Lite 示例。

4.1. 发送消息

这个客户端程序使用 **<connection-url>** 连接到服务器，为目标 **<address>** 创建一个发送程序，发送一条包含 **<message-body>** 的消息，关闭连接，然后退出。

示例：发送消息

```
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;** 导入 Amqp 命名空间中定义的类型。AMQP 由项目引用 *Amqp.Net.dll* 来定义，提供与 AMQ .NET 关联的所有类、接口和值类型。

2 命令行 arg[0] **url** 是 AMQP 连接的主机或虚拟主机的网络地址。此字符串描述了连接传输、用户和密码凭据，以及远程主机上连接的端口号。*URL* 可以寻址代理、独立对等点或路由器网络的入口点。

3. 命令行 arg[1] **target** 是远程主机中消息目标端点或资源的名称。
4. 命令行 arg[2] **count** 是要发送的信息数。
5. **peerAddr** 是创建 AMQP 连接所需的结构。
6. 创建 AMQP 连接。
7. **sender** 是一个可以 *发送邮件的客户端 SenderLink*。该链接被任意命名 *send-1*。使用适合您的环境中的链接名称，并帮助识别繁忙系统中的流量。链接名称不受限制，但必须在同一个会话中唯一。
8. 在消息发送循环中创建一个新消息。
9. 消息发送到 AMQP 对等点。
10. 发送所有消息后，协议对象会按顺序关闭。

运行示例
要运行示例程序，请编译并从命令行执行。如需更多信息，请参阅 第 3 章 入门。

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

4.2. 接收消息

这个客户端程序使用 **<connection-url>** 连接到服务器，为源 **<address>** 创建一个接收器，并在其终止或到达 **<count>** 信息前接收信息。

示例：接收消息

```
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();                             8
            receiver.Accept(msg);                                         9
            Console.WriteLine("Received: " + msg.Body.ToString());
```

```
            }

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

1. **using Amqp;** 导入 Amqp 命名空间中定义的类型。AMQP 由项目引用 *Amqp.Net.dll* 来定义，提供与 AMQ .NET 关联的所有类、接口和值类型。
2. 命令行 arg[0] **url** 是 AMQP 连接的主机或虚拟主机的网络地址。此字符串描述了连接传输、用户和密码凭据，以及远程主机上连接的端口号。*URL* 可以寻址代理、独立对等点或路由器网络的入口点。
3. 命令行 arg[1] **source** 是远程主机中消息源端点或资源的名称。
4. 命令行 arg[2] **count** 是要发送的信息数。
5. **peerAddr** 是创建 AMQP 连接所需的结构。
6. 创建 AMQP 连接。
7. **receiver** 是可接收消息的客户端 *ReceiverLink*。该链接被任意命名 *recv-1*。使用适合您的环境中的链接名称，并帮助识别繁忙系统中的流量。链接名称不受限制，但必须在同一个会话中唯一。
8. 接收消息。
9. 消息被接受。这会将消息的所有权从对等传输到接收方。
10. 收到所有消息后，协议对象会以有序的方式关闭。

运行示例

要运行示例程序，请编译并从命令行执行。如需更多信息，请参阅 第 3 章 入门。

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

第 5 章 网络连接

5.1. 连接 URI

这部分论述了用于连接到 AMQP 远程对等点的连接 URI 字符串的标准格式。

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

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

- **scheme amqp** - 连接使用 TCP 传输，并将默认端口设置为 672。
- **scheme amqps** - 连接使用 SSL/TLS 传输，并将默认端口设置为 671。
- **user** - 可选的连接身份验证用户名.如果存在 *用户名*，客户端会在连接启动时启动 AMQP SASL 用户凭据交换。
- **password** - 可选的连接身份验证密码。
- **主机** - 连接要定向到的网络主机。
- **port** - 连接要定向到的可选网络端口。默认 *端口* 值由 AMQP 传输方案决定。

连接 URI 示例

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

5.2. 重新连接和故障转移

AMQ .NET 不提供重新连接和故障转移，但可通过拦截连接错误和重新连接在应用程序中实施。例如，请参阅 ReconnectSender.cs 示例。

第 6 章 安全性

6.1. 使用用户和密码连接

AMQ .NET 可以使用用户和密码验证连接。

要指定用于身份验证的凭证，请在连接 URL 中设置 **user** 和 **password** 字段。

示例：使用用户和密码连接

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

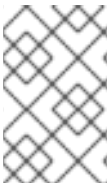
6.2. 配置 SASL 身份验证

与远程同级的客户端连接可能会交换 SASL 用户名和密码凭据。在连接 URI 中存在 *user* 字段可控制此交换。如果指定了 *用户*，则交换 SASL 凭据；如果没有 *用户*，则不交换 SASL 凭据。

默认情况下，客户端支持 **EXTERNAL**、**PLAIN** 和 **ANONYMOUS** SASL 机制。

6.3. 配置 SSL/TLS 传输

使用 SSL/TLS 实现与服务器的安全通信。可以为 SSL/TLS Handshake 配置客户端，也可以配置为 SSL/TLS Handshake 和客户端证书身份验证。如需更多信息，请参阅管理证书 部分。



注意

TLS 服务器名称引用(SNI)由客户端库自动处理。但是，SNI 仅针对使用 *amqps* 传输方案（其中主机为完全限定域名或主机名）的地址发出信号。当主机是数字 IP 地址时，SNI 不会发出信号。

第 7 章 发送者和接收方

客户端使用发送方和接收器链接来表示用于发送消息的通道。发送者和接收方是单向的，消息来源的结尾，消息目的地的目标结束。

源和目标通常指向消息代理上的队列或主题。源也用于表示订阅。

7.1. 按需创建队列和主题

某些消息服务器支持按需创建队列和主题。连接了发送方或接收方时，服务器使用发送方目标地址或接收器源地址来创建名称与该地址匹配的队列或主题。

消息服务器通常默认为创建队列（用于一对一消息发送）或主题（一对多消息发送）。客户端可以通过在源或目标中设置 **queue** 或 **topic** 功能来指示它首选的内容。

要选择队列或主题语义，请按照以下步骤执行：

1. 配置您的消息服务器，以自动创建队列和主题。这通常是默认配置。
2. 在发送者目标或接收器源中设置 **queue** 或 **topic** 功能，如下例所示。

示例：发送到按需创建的队列

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

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

示例：从按需创建的主题接收

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

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

如需更多信息，请参阅以下示例：

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

7.2. 创建持久订阅

持久订阅是远程服务器上代表消息接收器的一种状态。通常，当客户端关闭时，消息接收器会被丢弃。但是，由于持久订阅是永久的，客户端可以从它们分离，然后在以后重新连接。当客户端重新附加时，分离时收到的所有消息都可用。

持久订阅通过组合客户端容器 ID 和接收器名称组成订阅 ID 来唯一标识。这些必须具有稳定值，以便可以恢复订阅。

要创建持久订阅，请按照以下步骤执行：

1. 将连接容器 ID 设置为 stable 值，如 **client-1**:

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

2. 设置 **Durable** 和 **ExpiryPolicy** 属性，将接收器源配置为持久性：

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

3. 使用稳定名称（如 **sub-1** ）创建接收器，并应用源属性：

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

要从订阅分离，请在不明确关闭接收方的情况下关闭连接。要终止订阅，请直接关闭接收方。

如需更多信息，请参阅 DurableSubscribe.cs 示例。

7.3. 创建共享订阅

共享订阅是远程服务器上代表一个或多个消息接收器的一种状态。由于它是共享的，所以多个客户端可以从同一消息流消耗。

客户端通过在接收器源上设置 **shared** 功能来配置共享订阅。

共享订阅通过组合客户端容器 ID 和接收器名称组成订阅 ID 来唯一标识。这些必须具有稳定值，以便多个客户端进程可以找到相同的订阅。如果除了 **shared** 外还设置了 **global** 能力，则只使用接收器名称来标识订阅。

要创建共享订阅，请按照以下步骤执行：

1. 将连接容器 ID 设置为 stable 值，如 **client-1**:

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

2. 通过设置 **shared** 功能，将接收器源配置为共享：

```
Source source = new Source()
{
    Address = "notifications",
```

```
    Capabilities = new Symbol[] {"shared"},
};
```

3. 使用稳定名称（如 **sub-1** ）创建接收器，并应用源属性：

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

如需更多信息，请参阅 SharedSubscribe.cs 示例。

第 8 章 消息发送

8.1. 发送消息

要发送消息，请创建一个连接、会话和发送程序链接，然后使用 **Message** 对象调用 **Sender.Send()** 方法。

示例：发送消息

```
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);
```

如需更多信息，请参阅 Send.cs 示例。

8.2. 接收消息

要接收消息，创建一个连接、会话和接收器链接，然后调用 **Receiver.Receive()** 方法并使用返回的 **Message** 对象。

示例：接收消息

```
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() 调用告诉远程对等点，消息已被接收和处理。

如需更多信息，请参阅 Receive.cs 示例。

第 9 章 日志记录

日志记录在故障排除和调试中非常重要。默认情况下，日志记录处于关闭状态。要启用日志记录，您必须设置日志级别，并提供委派功能来接收日志消息。

9.1. 设置日志输出级别

该程序库在不同级别上发出日志追踪：

- 错误
- Warning
- 信息
- 详细

最低日志级别 *Error*，仅跟踪错误事件，并生成最少的日志消息。较高的日志级别包括它下面的所有日志级别，并生成更大的日志消息卷。

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

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

9.2. 启用协议日志记录

日志级别 *帧* 处理方式不同。设置追踪级别 *帧* 可启用 AMQP 协议标头和帧的追踪输出。

其他日志级别的跟踪必须在逻辑上使用 Frame 进行 ORed， 才能同时获取正常的追踪输出和 AMQP 帧追踪。例如

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

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

以下代码将 AMQP 帧写入控制台。

示例：日志记录委托

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

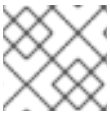
第 10 章 互操作性

本章讨论如何与其他 AMQ 组件结合使用 AMQ .NET。有关 AMQ 组件兼容性的概述，请参阅 产品简介。

10.1. 与其他 AMQP 客户端互操作

AMQP 消息通过 AMQP 类型系统来 构成。这种常见格式是不同语言的 AMQP 客户端能够相互互操作的原因之一。

发送消息时，AMQ .NET 会自动将语言原生类型转换为 AMQP 编码的数据。接收消息时，会进行反向转换。



注意

有关 AMQP 类型的更多信息，请参阅 Apache Qpid 项目维护 的交互式类型参考。

表 10.1. AMQP 类型

AMQP 类型	描述
null	一个空值
boolean	true 或 false 值
char	单个 Unicode 字符
string	Unicode 字符序列
binary	字节序列
byte	签名的 8 位整数
short	签名的 16 位整数
int	签名的 32 位整数
long	签名的 64 位整数
ubyte	未签名的 8 位整数
ushort	未签名的 16 位整数
uint	未签名 32 位整数
ulong	未签名 64 位整数
float	32 位浮动点数

AMQP 类型	描述
double	64 位浮动点数
array	单个类型值序列
list	变量类型的一系列值
map	从不同键到值的映射
uuid	通用唯一标识符
symbol	来自受限域的 7 位 ASCII 字符串
timestamp	绝对时间点

表 10.2. 编码之前和解码后 AMQ .NET 类型

AMQP 类型	编码前 AMQ .NET 类型	解码后 AMQ .NET 类型
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 类型	编码前 AMQ .NET 类型	解码后 AMQ .NET 类型
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 和其他 AMQ 客户端类型（2 中的 1 个）

编码前 AMQ .NET 类型	AMQ C++ 类型	AMQ JavaScript 类型
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

编码前 AMQ .NET 类型	AMQ C++ 类型	AMQ JavaScript 类型
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 和其他 AMQ 客户端类型（2 个）

编码前 AMQ .NET 类型	AMQ Python 类型	AMQ Ruby 类型
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

编码前 AMQ .NET 类型	AMQ Python 类型	AMQ Ruby 类型
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. 使用 AMQ JMS 进行互操作

AMQP 定义与 JMS 消息传递模型的标准映射。本节讨论该映射的方方面面。如需更多信息，请参阅 AMQ JMS Interoperability 一章。

JMS 消息类型

AMQ .NET 提供单一消息类型，其正文类型可能有所不同。相比之下，JMS API 使用不同的消息类型来表示不同类型的数据。下表指明了特定正文类型如何映射到 JMS 消息类型。

为了更明确地控制生成的 JMS 消息类型，您可以设置 **x-opt-jms-msg-type** 消息注解。如需更多信息，请参阅 AMQ JMS Interoperability 一章。

表 10.5. AMQ .NET 和 JMS 消息类型

AMQ .NET 正文类型	JMS 消息类型
System.String	**TextMessage**
null	**TextMessage**
System.Byte[]	**BytesMessage**
任何其他类型	**ObjectMessage**

10.3. 连接到 AMQ BROKER

AMQ Broker 旨在与 AMQP 1.0 客户端互操作。检查以下内容以确保为 AMQP 消息传递配置了代理：

- 网络防火墙中的端口 5672 已打开。
- 启用了 AMQ Broker AMQP 接收器。请参阅 默认接收器设置。
- 代理上配置了必要的地址。请参阅地址、队列和主题。

- 代理配置为允许来自您的客户端的访问，客户端被配置为发送所需的凭证。请参阅 Broker 安全。

10.4. 连接到 AMQ INTERCONNECT

AMQ 互连可与任何 AMQP 1.0 客户端配合工作。检查以下内容以确保正确配置了组件：

- 网络防火墙中的端口 5672 已打开。
- 路由器配置为允许从您的客户端进行访问，并且客户端配置为发送所需的凭据。请参阅保护网络连接。

附录 A. 管理证书

A.1. 安装证书颁发机构证书

SSL/TLS 身份验证依赖于受信任的证书颁发机构(CA)发布的数字证书。当客户端建立 SSL/TLS 连接时，AMQP 对等点向客户端发送服务器证书。此服务器证书必须由客户端 *受信任的根证书颁发机构* 证书存储中的其中一个 CA 签名。

如果用户为 Red Hat AMQ Broker 创建自签名证书，则用户必须创建一个 CA 为证书签名。然后，用户可以通过安装自签名 CA 文件 **ca.crt** 来启用客户端 SSL/TLS 握手。

1. 在管理员命令提示符中运行 MMC 证书管理器插件 **certmgr.msc**。
2. 展开左侧 **的受信任的根证书颁发机构** 文件夹，以公开 **证书**。
3. 右键单击 **Certificates**，然后选择 **All Tasks**，然后选择 **Import**。
4. 点 **Next**。
5. 浏览到选择文件 **ca.crt**。
6. 点 **Next**。
7. 选择 **"位置"所有证书位于以下存储中：**
8. 选择证书存储 **受信任的根证书颁发机构**。
9. 点 **Next**。
10. 点 **Finish**。

有关安装证书的更多信息，请参阅管理 Microsoft 证书服务和 SSL。

A.2. 安装客户端证书

要使用 SSL/TLS 和客户端证书，必须将含有客户端私钥的证书导入到客户端系统上的正确证书存储中。

1. 在管理员命令提示符中运行 MMC 证书管理器插件 **certmgr.msc**。
2. 展开左侧的 **Personal** 文件夹，以显示 **证书**。
3. 右键单击 **Certificates**，然后选择 **All Tasks**，然后选择 **Import**。
4. 点 **Next**。
5. 单击 **Browse**。
6. 在文件类型下拉菜单中，选择 **Personal Information Exchange(\.pfx;*.p12)**。
7. 选择文件 **client.p12** 并点击 **Open**。
8. 点 **Next**。
9. 输入私钥密码字段的密码。接受默认导入选项。
10. 点 **Next**。

11. 选择 **"位置"所有证书位于以下存储中：**
12. 选择证书存储 **个人.**
13. 点 **Next**。
14. 点 **Finish**。

A.3. 使用客户端证书的 HELLO WORLD

在客户端将证书返回到代理之前，必须告知 AMQ .NET 库要使用的证书。客户端证书文件 **client.crt** 添加到要在 **SChannel** 连接启动过程中使用的证书列表中。

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

在本例中，**certfile** 是个人证书 *存储* 中安装的 **client.p12** 证书的完整路径。在 **HelloWorld-client-certs.cs** 中找到一个完整的示例。SDK 中提供了此源文件和支持的项目文件。

附录 B. 示例程序

B.1. 先决条件

- 带有名为 **amq.topic** 的队列以及名为 **service_queue** 的队列的 Red Hat AMQ Broker 具有读/写权限。在这个示例中，代理位于 IP 地址 **10.10.1.1**。
- Red Hat AMQ 使用源和目标名称 **amq.topic** 及适当的权限进行互连。在这个示意图中，路由器位于 IP 地址 **10.10.2.2**。

所有示例都从 **<install-dir>\bin\Debug** 运行。

B.2. HELLOWORLD 简单

helloworld-simple 是一个简单的示例，它为同一地址创建 Sender 和 Receiver，发送消息到该地址，从地址读取消息，然后打印结果。

helloworld-simple 命令行选项

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

helloworld-simple 示例调用

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

默认情况下，该程序连接到 localhost:5672 上运行的代理。在命令行中明确指定主机和端口，以及 AMQP 端点地址：

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

默认情况下，该程序将其消息发送到 **amq.topic**。在某些 Amqp 代理中，amq.topic 是一个预定义的端点地址，它立即可用，没有代理配置。如果代理中没有这个地址，则使用代理管理工具创建它。

B.3. HELLOWORLD 稳健

helloworld-robust 通过附加选项和处理共享简单示例的所有功能：

- 访问简单有效负载之外的消息属性：
  - 标头
  - DeliveryAnnotations
  - MessageAnnotations
  - 属性
  - ApplicationProperties
  - BodySection

    - 页脚

- 连接关闭序列

helloworld-robust 命令行选项

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

注意

简单的 *enableTrace* 参数存在可启用追踪。参数可以保存任何值。

helloworld-robust 示例调用

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

helloworld-robust 允许用户指定有效负载字符串并启用追踪协议日志记录。

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

B.4. INTEROP.DRAIN.CS、INTEROP.SPOUT.CS（性能练习器）

AMQ .NET 示例 *Interop.Drain* 和 *Interop.Spout* 演示了与红帽 AMQ 互连的交互。在这种情况下没有消息代理。相反，Red Hat AMQ Interconnect 注册客户端程序请求的地址，并在它们之间路由消息。

Interop.Drain 命令行选项

```
$ 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 命令行选项

```
$ 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 和 Interop.Drain 示例调用
在一个窗口中运行 Interop.drain。排空会永远等待一条消息到达。

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

在另一个窗口中运行 Interop.spout。spout 将消息发送到代理地址并退出。

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

现在，在第一个窗口中排空将收到来自 spout 的消息，然后退出。

```
$ 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)

这个示例显示了一个基于代理的简单服务器，它将接受来自客户端的字符串，将它们转换为大写，并将它们发回到客户端。它有两个组成部分：

- client - 将位置行发送到服务器并打印响应.
- server - 一个简单的服务，它将传入字符串转换为大写字符串，并将它们返回到请求者。

在本例中，服务器和客户端在名为 **service_queue** 的代理中共享服务端点。服务器在服务端点侦听消息。客户端创建临时动态 ReplyTo 队列，将临时名称嵌入请求中，然后将请求发送到服务器。在收到和处理每个请求后，服务器会将回复发送到客户端的临时 ReplyTo 地址。

Interop.Client 命令行选项

```
Command line:
  Interop.Client [peerURI [loopcount]]
```

```
Default:
  Interop.Client amqp://guest:guest@localhost:5672 1
```

Interop.Server 命令行选项

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

Interop.Client, Interop.Server 示例调用

可使用以下命令行启动程序：

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

peertopeer.Server 在命令行中提供的地址上创建一个侦听器。此地址初始化侦听传入连接的 *ContainerHost* 类对象。收到的消息异步转发到 *RequestProcessor* 类对象。

peertopeer.Client 打开与服务器的连接，并开始向服务器发送消息。

peertopeer.Client 命令行选项

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

peertopeer.Server 命令行选项

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

peertopeer.Client、PeerToPeer.Server 示例调用

在一个窗口中运行 PeerToPeer.Server

```
$ 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
...
```

在另一个窗口中，运行 PeerToPeer.Client。peertopeer.Client 发送消息，并在收到消息时打印响应。

```
$ 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
```

附录 C. 使用您的订阅

AMQ 通过软件订阅提供。要管理您的订阅，请访问红帽客户门户中的帐户。

C.1. 访问您的帐户

流程

1. 转至 access.redhat.com。
2. 如果您还没有帐户，请创建一个帐户。
3. 登录到您的帐户。

C.2. 激活订阅

流程

1. 转至 access.redhat.com。
2. 导航到 **My Subscriptions**。
3. 导航到 **激活订阅** 并输入您的 16 位激活号。

C.3. 下载发行文件

要访问 .zip、.tar.gz 和其他发布文件，请使用客户门户查找要下载的相关文件。如果您使用 RPM 软件包或 Red Hat Maven 存储库，则不需要这一步。

流程

1. 打开浏览器并登录红帽客户门户网站 **产品下载页面**，网址为 access.redhat.com/downloads。
2. 查找 **INTEGRATION 类别中 的红帽 AMQ** 条目。
3. 选择所需的 AMQ 产品。此时会打开 **Software Downloads** 页面。
4. 单击组件的 **Download** 链接。

C.4. 为系统注册软件包

要在 Red Hat Enterprise Linux 上安装此产品的 RPM 软件包，必须注册您的系统。如果您使用下载的发行文件，则不需要这一步。

流程

1. 转至 access.redhat.com。
2. 进入 **Registration Assistant**。
3. 选择您的操作系统版本，再继续到下一页。
4. 使用您的系统终端中列出的命令完成注册。

有关注册您的系统的更多信息，请参阅以下资源之一：

- Red Hat Enterprise Linux 7 - 注册系统并管理订阅
- Red Hat Enterprise Linux 8 - 注册系统并管理订阅

附录 D. 将 AMQ BROKER 与示例搭配使用

AMQ .NET 示例需要一个正在运行的消息代理，其中包含名为 **amq.topic** 的队列。使用以下步骤安装和启动代理并定义队列。

D.1. 安装代理

按照 AMQ Broker 入门 以 安装代理 并创建代理 实例 中的内容进行操作。启用匿名访问。

以下步骤将代理实例的位置称为 ***<broker-instance-dir>***。

D.2. 启动代理

流程

1. 使用 **artemis run** 命令启动代理。

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

2. 检查控制台输出中是否有启动过程中记录的严重错误。代理日志记录 **Server is now live** （当它就绪时）。

```
$ 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. 创建队列

在新终端中，使用 **artemis queue** 命令创建名为 **amq.topic** 的队列。

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

系统将提示您回答一系列"是"或无问题。对所有设备回答 **N**。

队列创建后，代理就可与示例程序配合使用。

D.4. 停止代理

运行完示例后，使用 **artemis stop** 命令停止代理。

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

2021-08-31 15:46:35 +1000 修订