Red Hat AMQ 2021.Q3

使用 AMQ Ruby 客户端

用于 AMQ 客户端 2.10

Last Updated: 2023-02-20

Red Hat AMQ 2021.Q3 使用 AMQ Ruby 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。

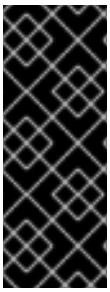
目录

使开源包含更多

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 Ruby 是用于开发消息传递应用程序的库。它可让您编写发送和接收 AMQP 消息的 Ruby 应用。



重要

AMQ Ruby 客户端只是一个技术预览功能。技术预览功能不被红帽产品服务等级协议 (SLA) 支持，且可能在功能方面有缺陷。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的详情，请参阅 https://access.redhat.com/support/offerings/techpreview/。

AMQ Ruby 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 Ruby 基于 Apache Qpid 中的 Proton API。如需详细的 API 文档，请参阅 AMQ Ruby API 参考。

1.1. 主要特性

- 简化与现有应用程序的集成的事件驱动的 API
- 用于安全通信的 SSL/TLS
- 灵活的 SASL 身份验证
- 自动重新连接和故障转移
- AMQP 和原生语言数据类型之间的无缝转换
- 访问 AMQP 1.0 的所有特性和功能

1.2. 支持的标准和协议

AMQ Ruby 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
- Cyrus SASL 支持 的简单身份验证和安全层(SASL)机制，包括 ANONYMOUS、PLAIN、SCRAM、EXTERNAL 和 GSSAPI(Kerberos)
- 使用 IPv6的现代 TCP

1.3. 支持的配置

有关 AMQ Ruby 支持的配置，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
Container	连接的顶级容器。
连接	个网络上两个同级之间的通信通道。它包含会话。
会话	用于发送和接收消息的上下文。它包含发送方和接收方。
sender	用于将消息发送到目标的频道。它有一个目标。
receiver	从源接收信息的频道。它有一个源。
Source	消息的指定来源点。
目标	消息的指定目的地。
消息	特定于应用的信息。
交付	消息传输。

AMQ Ruby 发送并接收 *消息*。消息通过 *发送方和接收方在连接的对等点* 之间传输 。通过 *会话* 创建发件人和接收方。通过 *连接* 建立会话。连接在两个唯一标识 *的容器* 之间建立。虽然连接可以有多个会话，但通常不需要。API 允许您忽略会话，除非您需要它们。

发送对等点会创建一个发送者来发送消息。发送方具有在远程同级上标识队列或主题 *的目标*。接收方创建接收方来接收消息。接收方具有一个 *源*，用于标识远程对等点上的队列或主题。

消息的发送称为 *发送*。消息是发送的内容，包括标头和注释等所有元数据。交付是指与该内容的传输相关的协议交换。

为了表示某一交付已完成，发件人或接收方都可处理该交付。当另一边了解到它已被实施时，将不会再交流该交付。接收方也可以指示它接受还是拒绝消息。

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

本章指导您完成在您的环境中安装 AMQ Ruby 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 上安装软件包，您必须 注册您的系统。
- 要使用 AMQ Ruby，您必须在您的环境中安装 Ruby。

2.2. 在 RED HAT ENTERPRISE LINUX 上安装

流程

1. 使用 **subscription-manager** 命令订阅所需的软件包软件仓库。将 ***<version>*** 替换为主发行流的 **2** 或 **2.9** 用于长期支持发行流。如果需要，使用 Red Hat Enterprise Linux 变体的值替换 ***<variant>***（例如： **server** 或 **workstation**）。

   Red Hat Enterprise Linux 7

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

   Red Hat Enterprise Linux 8

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

2. 使用 **yum** 命令安装 **rubygem-qpid_proton** 和 **rubygem-qpid_proton-doc** 软件包。

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

有关使用软件包的详情请参考 附录 B, *使用 Red Hat Enterprise Linux 软件包* 。

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 **examples** 的队列。如需更多信息，请参阅 创建队列。

3.2. 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 **examples** 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。

更改到示例目录并运行 **helloworld.rb** 示例。

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

第 4 章 示例

本章介绍如何通过示例程序使用 AMQ Ruby。

如需更多示例，请参阅 AMQ Ruby 示例套件 和 Qpid Proton Ruby 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

```
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
```

运行示例

要运行示例程序，将其复制到本地文件中并使用 **ruby** 命令调用它。如需更多信息，请参阅 第 3 章 入门。

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

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

```
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
```

运行示例

要运行示例程序，将其复制到本地文件中并使用 **ruby** 命令调用它。如需更多信息，请参阅 第 3 章 入门。

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

第 5 章 网络连接

5.1. 连接 URL

连接 URL 对用于建立新连接的信息进行编码。

连接 URL 语法

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

- *scheme* - 连接传输，可以为未加密 TCP 或 **amqp** 用于使用 SSL/TLS 加密的 TCP 进行连接传输。**amqps**
- *主机* - 远程网络主机。该值可以是主机名或数字 IP 地址。IPv6 地址必须括在方括号中。
- *port* - 远程网络端口。这个值是可选的。**amqp** 方案默认值为 5672，**amqps** 方案为 5671。

连接 URL 示例

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

第 6 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。

6.1. 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。

如需了解更多详细信息，请参阅以下示例：

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

6.2. 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。

示例

6.3. 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。

示例

第 7 章 日志记录

7.1. 启用协议日志记录

客户端可以将 AMQP 协议框架记录到控制台。诊断问题时，这些数据通常至关重要。

要启用协议日志记录，将 **PN_TRACE_FRM** 环境变量设置为 **1** ：

示例：启用协议日志记录

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

要禁用协议日志，请取消设置 **PN_TRACE_FRM** 环境变量。

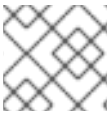
第 8 章 互操作性

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

8.1. 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 Ruby 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 8.1. 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 8.2. 在编码之前和解码后，AMQ Ruby 类型

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

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

编码前 AMQ Ruby 类型	AMQ .NET 类型	AMQ Python 类型
nil	null	None
true, false	System.Boolean	bool

编码前 AMQ Ruby 类型	AMQ .NET 类型	AMQ Python 类型
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. 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 Ruby 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 8.5. AMQ Ruby 和 JMS 消息类型

AMQ Ruby 正文类型	JMS 消息类型
String	**TextMessage**
nil	**TextMessage**
-	**BytesMessage**
任何其他类型	**ObjectMessage**

8.3. 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。

8.4. 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. 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。

A.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。

A.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。

A.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。

A.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 B. 使用 RED HAT ENTERPRISE LINUX 软件包

这部分论述了如何将软件包作为 Red Hat Enterprise Linux 的 RPM 软件包使用。

为确保此产品的 RPM 软件包可用，您必须先 注册您的系统。

B.1. 概述

库或服务器等组件通常具有多个与之相关联的软件包。您不必全部安装它们。您只能安装您需要的产品。

主软件包通常具有最简单的名称，没有额外的限定符。此软件包提供在程序运行时使用组件所需的所有接口。

名称以 **-devel** 结尾的软件包包含 C 和 C++ 库的标头。编译时需要这些，以构建依赖于此软件包的程序。

名称以 **-docs** 结尾的软件包包含组件的文档和示例程序。

有关使用 RPM 软件包的更多信息，请参阅以下资源之一：

- Red Hat Enterprise Linux 7 - 安装和管理软件
- Red Hat Enterprise Linux 8 - 管理软件包

B.2. 搜索软件包

要搜索软件包，请使用 **yum search** 命令。搜索结果包括软件包名称，您可以在本节中列出的其他命令中用作 ***<package>*** 的值。

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

B.3. 安装软件包

要安装软件包，使用 **yum install** 命令。

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

B.4. 查询软件包信息

要列出系统中安装的软件包，使用 **rpm -qa** 命令。

```
$ rpm -qa
```

要获取有关特定软件包的信息，请使用 **rpm -qi** 命令。

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

要列出与软件包关联的所有文件，请使用 **rpm -ql** 命令。

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

附录 C. 将 AMQ BROKER 与示例搭配使用

AMQ Ruby 示例需要一个正在运行的消息代理，它带有名为 **examples** 的队列。使用以下步骤安装和启动代理并定义队列。

C.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>***。

C.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
...
```

C.3. 创建队列

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

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

C.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 15:47:25 +1000 修订