RCloud MQ 快速入门手册

（RCloud Version 6.0）

中软国际有限公司

2014 年 07 月

目 录

第一章 概述 3

1.1 导读 3

1.2 准备工作 4

第二章 JMS 规范简介 4

2.1 JMS 组成元素 4

2.2 JMS 模型简介 5

2.3 JMS 接口简介 5

第三章 安装 RCLOUD MQ 服务 7

3.1 部署环境 7

3.2 产品目录 7

第四章 启动 RCLOUD MQ 服务 8

第五章 开发客户端程序 9

5.1 点对点消息发送 9

5.2 点对点消息接收 10

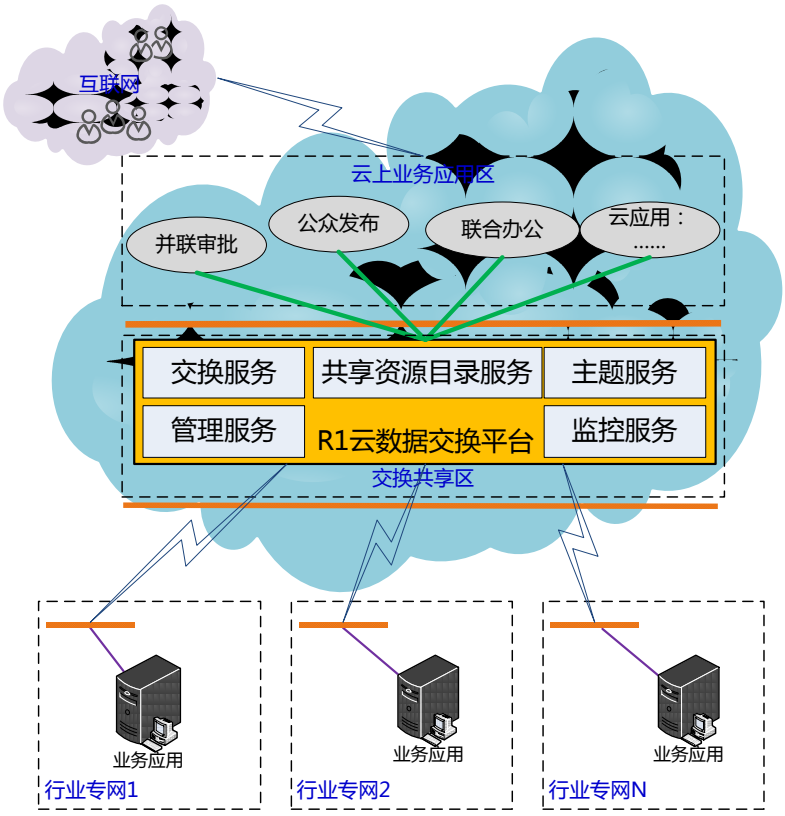
5.3 REST 消息广播 11

第六章 联系我们 18

第一章 概述

1.1 导读

RCloud 是中软国际最新推出的 PaaS 产品，主要应用于解决云应用间、云应用与传统遗留应用间提供相互连接、消息共享与交互等集成服务，是实现云平台中应用集成不可或缺的能力，目前可以运行在阿里云环境中。典型场景如下图：



RCloud 平台部署在云环境中，部分公共服务应用部署在云环境中，还有部分业务应用部署在行业专网中，这两部分应用之间有交互需求，云环境与行业专网之间一般都有防火墙隔离。

RCloud MQ 是 RCloud 平台的消息中间件产品。它实现多种复杂网络环境下的消息交换。

RCloud MQ 完全符合 JMS1.1 标准和支持最新的 JMS2.0 标准，并提供丰富的功能来构建 JMS 应用。

本手册快速指导用户安装配置和运行 MQ 服务，并使用 API 进行简单的 JMS

客户端开发，最后运行测试结果。使得开发人员能第一时间对 RCloud MQ 的使用和开发流程有个最直观的了解，详细的开发过程请参见产品开发手册。

1.2 准备工作

为了进行开发，你必须有 RCloud MQ 安装包和 JDK6+的环境，另外为了进行测试您最好有个 web 容器比如 Tomcat6 来运行示例。

第二章 JMS 规范简介

2.1 JMS 组成元素

- JMS 提供者

连接面向消息中间件的，JMS 接口的一个实现。提供者可以是 Java 平台的 JMS 实现，也可以是非 Java 平台的面向消息中间件的适配器。

- JMS 客户

生产或消费基于消息的 Java 的应用程序或对象。

- JMS 生产者

创建并发送消息的 JMS 客户。

- JMS 消费者

接收消息的 JMS 客户。

- JMS 消息

包括可以在 JMS 客户之间传递的数据的对象

- JMS 队列

一个容纳那些被发送的等待阅读的消息的区域。队列暗示，这些消息将按照顺序发送。一旦一个消息被阅读，该消息将被从队列中移走。

- JMS 主题

一种支持发送消息给多个订阅者的机制。

2.2 JMS 模型简介

◆ 点对点（队列）模型

该模型中，一个生产者向一个特定的队列发布消息，一个消费者从该队列中读取消息。生产者知道消费者的队列，并直接将消息发送到消费者的队列。

这种模式被概括为：

只有一个消费者将获得消息。 生产者不需要在接收者消费该消息期间处于运行状态，接收者也同样不需要在消息发送时处于运行状态。

◆ 发布者/订阅者（主题）模型

该模型支持向一个特定的消息主题发布消息。0 或多个订阅者可能对接收来自特定消息主题的消息感兴趣。在这种模型下，发布者和订阅者彼此不知道对方。这种模式好比是匿名公告板。

这种模式被概括为：

多个消费者可以获得同一消息 。在发布者和订阅者之间存在时间依赖性。发布者需要建立一个订阅（subscription），以便客户能够购订阅。订阅者必须保持持续的活动状态以接收消息。

2.3 JMS 接口简介

◆ ConnectionFactory 接口（连接工厂）

用户用来创建到 JMS 提供者的连接的被管对象。JMS 客户通过可移植的接口访问连 接，这样当下层的实现改变时，代码不需要进行修改。 管理员在 JNDI 名字空间中配置连接工厂，这样，JMS 客户才能够查找到它们。根据消息类型的不同，用户将使用队列连接工厂，或者主题连接工厂。

◆ Connection 接口（连接）

连接代表了应用程序和消息服务器之间的通信链路。在获得了连接工厂后，就可以创建一个与 JMS 提供者的连接。根据不同的连接类型，连接允许用户创建会话，以发送和接收队列和主题到目标。

◆ Destination 接口（目标）

目标是一个包装了消息目标标识符的被管对象，消息目标是指消息发布和接收的地点，或者是队列，或者是主题。JMS 管理员创建这些对象，然后用户通过 JNDI 发现它们。和连接工厂一样，管理员可以创建两种类型的目标，点对点模型的队列，以及发布者/订阅者模型的主题。

◆ MessageConsumer 接口（消息消费者）

由会话创建的对象，用于接收发送到目标的消息。消费者可以同步地（阻塞模式），或异步（非阻塞）接收队列和主题类型的消息。

◆ MessageProducer 接口（消息生产者）

由会话创建的对象，用于发送消息到目标。用户可以创建某个目标的发送者，也可以创建一个通用的发送者，在发送消息时指定目标。

◆ Message 接口（消息）

是在消费者和生产者之间传送的对象，也就是说从一个应用程序传送到另一个应用程序。一个消息有三个主要部分：

1. 消息头（必须）：包含用于识别和为消息寻找路由的操作设置。
2. 一组消息属性（可选）：包含额外的属性，支持其他提供者和用户的兼容。可以创建定制的字段和过滤器（消息选择器）。
3. 一个消息体（可选）：允许用户创建五种类型的消息（文本消息，映射消息，字节消息，流消息和对象消息）。

消息接口非常灵活，并提供了许多方式来定制消息的内容。

◆ Session 接口（会话）

表示一个单线程的上下文，用于发送和接收消息。由于会话是单线程的，所以消息是连 续的，就是说消息是按照发送的顺序一个一个接收的。会话的好处是它支持事务。如果用户选择了事务支持，会话上下文将保存一组消息，直到事务被提交才发送这 些消息。在提交事务之前，用户可以使用回滚操作取消这些消息。一个会话允许用户创建消息生产者来发送消息，创建消息消费者来接收消息。

。

第三章 安装 RCloud MQ 服务

3.1 部署环境

操作系统：Windows、Unix、Linux

JDK：JDK 1.6 以上

应用服务器：标准 JEE 服务器

3.2 产品目录

Rcloud MQ 使用 java 开发，只需要有 JDK 环境即可，不需要安装就可以运行。发布包产品目录结构如下。

目录结构	说明
conf	服务器的配置文件
doc	使用手册和开发手册
bin	服务启动文件
data	数据文件，存放持久化和大消息数据
logs	工作日志
lib	服务器和客户端 jar 包
examples	使用示例
bin 文件夹下目录结构	
startup.bat	
startup.sh	
shutdown.bat	
shutdown .sh	
conf 文件夹下目录结构	
mq-configuration.xml	服务主配置文件
mq-jms.xml	JMS 配置文件

mq-user.xml	安全配置文件
mq-beans.xml	Bean 配置文件
jndi.properties	JNDI 配置文件
logging.properties	Log 配置文件
mq-client.xml	客户端配置文件

第四章 启动 RCloud MQ 服务

默认情况下，我们使用 RCloudMQ 的缺省配置即可满足一般应用需求。启动 startup.exe 启动服务。

控制台提示服务启动 OK，可以进入 web 管理平台进行查看服务器的信息。

第五章 开发客户端程序

RCloudMQ 在客户端开发 API 进行了不同需求的封装，既可以使用标准的 JMS 接口进行开发，也可以使用简化接口开发，还可以使用高级接口进行复杂情况下的应用开发。

这里我们介绍两个快速开发的示例来说明 RCloudMQ 在客户的应用开发方面的便捷。

5.1 点对点消息发送

点对点是 JMS 的标准模式之一，我们在客户端使用简化接口后，发送一个消息和接收一个消息只需要几行简单的代码就可以完成！

发送消息只需要一个命令就可以完成，用户不需要去了解工厂、会话、生产者、消息、队列等概念和配置。系统已经进行了封装。

我们写一个 MqSend.java 的发送类，使用 MqJMSClient 接口的 sendMsg 来发送消息，完整的代码如下：

```
package com.chinasofti.mq;

import com.chinasofti.mq.client.core.MqJmsClient;

public class MqSend {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {

        String msg = "发送消息测试！";
        //发送普通文本消息直接调用MqJmsClient的sendMsg即可。
        MqJmsClient.getClient().sendMsg(msg);

        System.out.println("发送消息:" + msg);
    }
}
```

非常简单吧，RCloudMQ 就是封装了复杂的工厂连接等对象，提供了一个简

洁的方法来满足大部分的简单需要。

我们直接运行 main 方法，发送消息到服务器。

下面是运行的结果：

```
log4j:WARN No appenders could be found for logger (org.jboss.logging).
log4j:WARN Please initialize the log4j system properly.
消息发送中...发送消息测试！
发送消息:发送消息测试！
```

为了验证我们的消息是否发送成功，我们编写下面的接收程序来接收消息。

5.2 点对点消息接收

接收一个消息也只需要一个命令就可以完成。我们写一个 MqGet.java 的接收类，使用 MqJMSClient 接口的 getMsg 来接收消息，完整的代码如下：

```
package com.chinasofti.mq;

import com.chinasofti.mq.client.core.MqJmsClient;

public class MqGet {

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {

        String msg = "接收消息测试！";
        //接收普通文本消息直接调用MqJmsClient的getMsg即可。
        msg = MqJmsClient.getClient().getMsg();

        System.out.println("接收到消息:" + msg);
    }
}
```

运行 main 方法，我们看输出结果：

```
log4j:WARN No appenders could be found for logger (org.jboss.logging).
log4j:WARN Please initialize the log4j system properly.
==>收到消息...发送消息测试！
接收到消息:发送消息测试！
```

我们收到了发送者发送的消息，上面第一行信息是 MqJMSClient 的调试信息。

5.3 REST 消息广播

RCloudMQ 提供了 REST 服务，使得客户端开发完全用 http 协议就可以完成 JMS 的消息发送和接收。

这个例子展示了使用 rest 服务后只需要 js 就可以完成 JMS 的功能。

主要有三个核心的部分：

一、初始化 rest 配置

```
function initializeSenderAndTop(topic)
{
    var xhr = createXHR();
    xhr.open("HEAD", "topics/" + topic, true);
    xhr.onreadystatechange = function()
    {
        if (xhr.readyState == 4)
        {
            if (xhr.status == 200)
            {
                // getting the links from the rest resource
                topicSender = xhr.getResponseHeader("msg-create");
                subscriptions = xhr.getResponseHeader("msg-pull-subscriptions");

                // just adding the report
                document.getElementById("errors").innerHTML = "Subscriptions URL: " + subscriptions;
            }
        }
    }
    // this will send the request from javascript
    xhr.send(null);
}
```

方法参数 topic 是一个主题的队列名称，在 mq-jms.xml 中进行配置。

二、发送消息

```
function postMessage(user, message)
{
    var xhr = createXHR();
    xhr.open("POST", topicSender, false);
    xhr.setRequestHeader("Content-Type", "text/plain");
    xhr.send(user + ": " + message);
    if (xhr.status == 201)
    {
      topicSender = xhr.getResponseHeader("msg-create-next");
    }
    else
    {
      document.getElementById("errors").innerHTML = "Failed to send message: " + topicSender;
    }
}
```

这里也只需要 user 用户和 message 消息两个参数即可发送消息。

三、接收消息

```
function receiveMessage()
{
    var xhr = createXHR();
    if (reconnect)
    {
      document.getElementById("connection").innerHTML = "Trying to reconnect: " + subscriptions + " retries: " + count++;
      xhr.open("POST", subscriptions, true);
      xhr.onreadystatechange = function()
      {
        if (xhr.readyState == 4)
        {
          var status = xhr.status;
          if (status == 201)
          {
            nextMessage = xhr.getResponseHeader("msg-consume-next");
            document.getElementById("connection").innerHTML = "Connected to: " + nextMessage;
            count = 1;
            reconnect = false;
          }
          setTimeout("receiveMessage()", 800);
        }
      }
```

这里使用 ajax 和 rest 服务进行交互，用户完全不用去注意底层的细节实现而专注业务开发。

完整代码如下，我们建一个 rest.jsp 文件，只要在 web 里访问该文件就可以进行消息的广播了，完全不需要 JMS 的接口，也不需要上面的 MqJMSClient 接口。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script type="text/javascript">
        <!--
        function createXHR()
        {
```

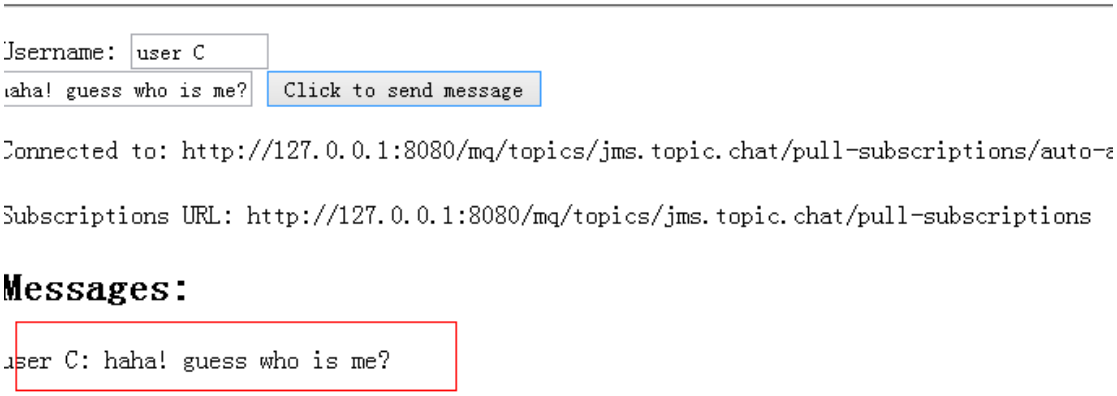
```
            var request = false;
            try
            {
                request = new ActiveXObject('Msxml2.XMLHTTP');
            }
            catch (err2)
            {
                try
                {
                    request = new ActiveXObject('Microsoft.XMLHTTP');
                }
                catch (err3)
                {
                    try
                    {
                        request = new XMLHttpRequest();
                    }
                    catch (err1)
                    {
                        request = false;
                    }
                }
            }
            return request;
        }
        var topicSender;
        var nextMessage;
        var subscriptions;
        var count = 0;
        var reconnect = true;

        function initializeSenderAndTop(topic)
        {
            var xhr = createXHR();
            xhr.open("HEAD", "topics/" + topic, true);
            xhr.onreadystatechange = function()
            {
                if (xhr.readyState == 4)
                {
                    if (xhr.status == 200)
                    {
                        // getting the links from the rest resource
```

```
                        topicSender = xhr.getResponseHeader("msg-create");
                        subscriptions = xhr.getResponseHeader("msg-pull-subscriptions");

                        // just adding the report
                        document.getElementById("errors").innerHTML = "Subscriptions URL: "
+ subscriptions;
                    }else{
                        alert(xhr.status);
                    }
                }
            }
            // this will send the request from javascript
            xhr.send(null);
        }

        function postMessage(user, message)
        {
            var xhr = createXHR();
            xhr.open("POST", topicSender, false);
            //xhr.setRequestHeader("Content-Type", "text/plain");
            xhr.send(user + ": " + message);

            if (xhr.status == 201)
            {
                topicSender = xhr.getResponseHeader("msg-create-next");
            }
            else
            {
                document.getElementById("errors").innerHTML = "Failed to send message: " +
topicSender;
            }
        }

        function receiveMessage()
        {
            var xhr = createXHR();
            if (reconnect)
            {
                document.getElementById("connection").innerHTML = "Trying to reconnect: " +
subscriptions + " retries: " + count++;
                xhr.open("POST", subscriptions, true);
                xhr.onreadystatechange = function()
```

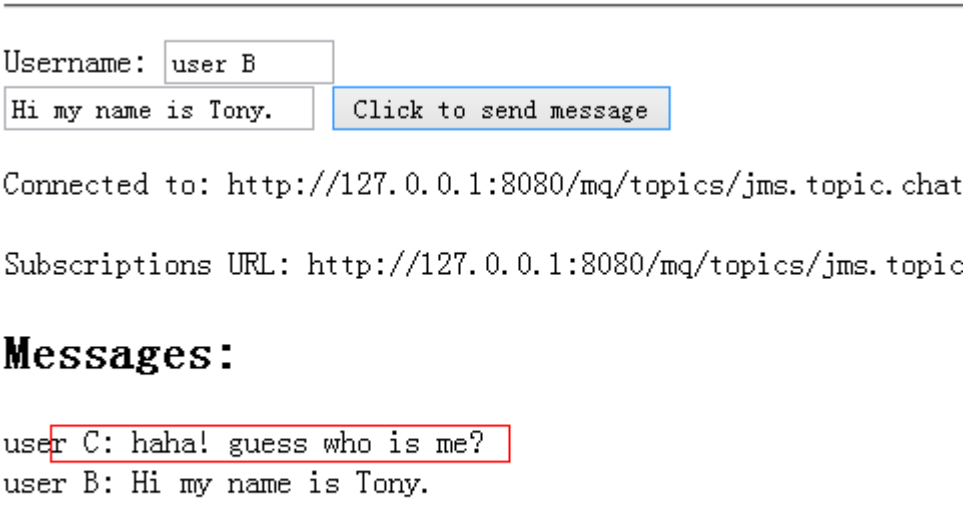
```
                {
                    if (xhr.readyState == 4)
                    {
                        var status = xhr.status;

                        if (status == 201)
                        {
                            nextMessage = xhr.getResponseHeader("msg-consume-next");
                            document.getElementById("connection").innerHTML = "Connected to: "
+ nextMessage;
                            count = 1;
                            reconnect = false;
                        }
                        setTimeout("receiveMessage()", 800);
                    }
                }
                xhr.send(null);
            }
            else
            {
                xhr.open("POST", nextMessage, true);
                xhr.setRequestHeader("Accept-Wait", "10")
                xhr.onreadystatechange = function()
                {
                    if (xhr.readyState == 4)
                    {
                        var status = xhr.status;

                        if (status == 200)
                        {
                            document.getElementById("next").innerHTML = xhr.responseText +
"\n" + document.getElementById("next").innerHTML;
                            nextMessage = xhr.getResponseHeader("msg-consume-next");
                        }
                        else
                        {
                            reconnect = true;
                        }
                        setTimeout("receiveMessage()", 800);
                    }
                }
                xhr.send(null);
```

```
            }
        }
        initializeSenderAndTop('jms.topic.chat');

    setTimeout("receiveMessage()", 800);

        // -->
    </script>
    <title>Rest 广播 Demo</title></head>

<body bgcolor="#FFFFFF">

<p><font size="+3">Rest 广播 Demo</font></p>
<hr>
<FORM name="ajax" method="POST" action="">

    <p>
        Username: <input type="text" name="user" value="Billy" size="10"><br/>
        <input type="text" name="message">
        <INPUT type="BUTTON" value=" Click to send message "
                ONCLICK="postMessage(this.form.user.value, this.form.message.value)">
    </p>

</FORM>
<div id="connection"></div>
<br>

<div id="errors"></div>

<p>

<h2>Messages:</h2></p>
<pre>
<div id="next"></div>
</pre>

</body>
</html>
```

为了验证上面代码，我们使用多个窗口进行测试，一个客户端发送的消息可以被所有接收的客户端所消费。



上面是用户三发送了一个消息，自己接收到了。



上面是用户二发送的消息，也接收到了用户三的消息。

这个结果是用户一发送的消息，他接收到了其他两个人的消息。

这里只提供了几个简单的例子来说明使用 RCloudMQ 开发 JMS 应用是多么的简洁和方便。但这并不是说 RCloudMQ 只能实现简单的功能，RCloudMQ 提供了非常全面的 JMS 功能，需要了解相关开发技术，请参阅产品开发手册。

第六章 联系我们

北京中软国际信息技术有限公司

地址：北京中关村科学院南路 2 号融科资讯中心 C 座北楼 12F-15F

电话：010-82861666　传真：010-82862809

Email：resourceone@chinasofti.com.cn

ResourceOne 社区：http://resourceone.chinasofti.com