RCloud MQ ® V6

产品用户使用手册

（RCloud MQ Version 6.0）

北京中软国际信息技术有限公司

2014 年

目 录

第一章 概述 5

1.1 面向读者 5

1.2 用户导读 5

1.3 名词解释 6

第二章 产品介绍 9

2.1 消息中间件 9

2.1.1 中间件的分类 9

2.1.2 消息中间件的介绍 9

2.1.3 消息中间件的原理 10

2.1.4 消息中间件和分布式对象调用的比较 10

2.2 JMS 简述 12

2.2.1 JMS 组成元素 12

2.2.2 JMS 模型简介 12

2.2.3 JMS 接口简介 13

2.2.4 JMS 工作流程 14

2.3 RCLOUD MQ 介绍 15

2.3.1 Rcloud MQ 的特点 16

2.3.2 Rcloud MQ 工作模式 17

第三章 安装部署指南 18

3.1 部署环境 18

3.2 产品目录 18

3.3 独立模式 19

3.3.1 主文件配置 19

3.3.2 JMS 配置文件 20

3.3.3 用户配置文件 21

3.3.4 JNDI 配置文件 23

3.3.5 日志配置文件 23

3.3.6 客户端配置文件 24

3.4 集成模式 25

3.5 嵌入模式 25

第四章 快速开发指南 27

4.1 点对点消息发送 27

4.2 点对点消息接收 27

4.3 REST 消息广播 28

第五章 服务管理指南 30

5.1 WEB 管理 30

5.1.1 服务监控 30

5.1.2 队列管理 31

5.1.3 主题管理 31

5.1.4 订阅管理 32

5.2 JMX 管理 32

5.3 JMS 管理 33

第六章 服务配置指南 34

6.1 传输层的配置 34

6.1.1 接收器（Acceptor）配置 34

6.1.2 连接器（Connector）配置 34

6.1.3 配置 Netty 传输层 35

6.1.4 配置 Stomp 协议 40

6.1.5 配置 AMQP 协议 41

6.2 REST 服务配置 42

6.2.1 Rest 服务配置 42

6.2.2 Rest 参数配置 43

6.3 队列和路由配置 44

6.3.1 模版路由 *44*

6.3.2 模板表达式 *45*

6.3.3 队列配置 *45*

6.4 消息配置 48

6.4.1 消息重发和死信配置 *48*

6.4.2 过期消息配置 *50*

6.4.3 超大消息配置 *51*

6.4.4 分页转存配置 *51*

6.4.5 消息分组配置 *53*

6.4.6 消息通知 *54*

6.4.7 消息转发分流 *55*

6.4.8 重复消息检测 *56*

6.4.9 消息拦截 *57*

6.4.10 定时消息 *58*

6.4.11 最新值消息 *58*

6.5 流量控制 59

6.5.1 消费者（consumer）流量控制 *59*

6.5.2 生产者（producer）流量控制 *61*

6.6 持久化配置 64

6.6.1 绑定日志 *64*

6.6.2 JMS 日志 *64*

6.6.3 消息日志 *65*

6.6.4 消息非持久化 *67*

6.7 可靠性配置 67

6.7.1 事务保证 *67*

6.7.2 非事务性消息发送的保证 *68*

6.7.3 非事务性通知的保证 *68*

6.7.4 异步发送通知 *69*

6.7.5 客户端重连 70

6.7.6 失效客户端 72

6.7.7 客户端的故障检测 72

6.8 安全性配置 73

6.8.1 基于角色的地址安全 73

6.8.2 基于套接字的 SSL 安全传输 74

6.8.3 用户信息管理 75

6.8.4 自定义用户管理模型 75

6.8.5 JAAS 安全管理器 76

6.9 线程池配置 76

6.9.1 服务器端线程的管理 76

6.9.2 客户端线程的管理 78

第七章 集群配置指南 79

7.1 集群概述 79

7.2 服务器发现 79

7.2.1 广播组 80

7.2.2 发现组 82

7.3 服务端的消息负载均衡 84

7.3.1 配置集群连接 84

7.3.2 集群用户的安全信息 86

7.4 客户端的消息负载均衡 87

7.5 显式指定集群服务器 88

7.5.1 在客户端指定服务器列表 88

7.5.2 指定服务器列表以组成集群 89

7.6 消息再分配 89

7.7 集群拓扑结构 90

7.7.1 对称式集群 91

7.7.2 链式集群 91

7.8 HA 配置 92
7.8.1 Master-Backup 对 *92*
7.8.2 失效备援的模式 *96*
第八章 配置优化指南 100
8.1 持久层的优化 100
8.2 优化 JMS 100
8.3 其它优化 101
8.4 传输层的优化 103
8.5 优化虚拟机 103

第一章 概述

1.1 面向读者

本手册详细描述了 RCloud MQ6.0（以下简称 RCMQ）的安装部署过程及使用方法，主要面向基于该产品的二次开发人员、实施部署人员以及最终用户的系统管理人员。

1.2 用户导读

第二章概要介绍了 MoM 的概念，JMS 规范，RCMQ 的工作机制和主要特性；

第三章分别介绍 RCMQ 的三种工作模式的安装部署和应用场景；

第四章简要介绍 RCMQ 的客户端开发接口，以两个简单的例子描述了如何使用 RCMQ 进行快速开发过程；

第五章介绍了 RCMQ 提供的多种服务管理方式，实现对服务器进行有效管控；

第六章详细介绍了 RCMQ 实现 JMS 服务的各种参数配置；

第七章主要介绍 RCMQ 集群配置；

第八章对系统使用常见问题进行说明。

1.3 名词解释

◆ RCloud

RCloud 是中软国际的 PaaS 云平台。

◆ IPaaS

IPaaS 是 RCloud 的一个服务集成平台，分为 ESB，ETL，EI 和 RCMQ 以及消息中间件五个产品。

◆ RCMQ

RCMQ 是中软国际的云应用消息中间件。

◆ MOM

MOM 是指消息中间件，它提供了以松散耦合的灵活方式集成应用程序的一种机制。它们提供了基于存储和转发的应用程序之间的异步数据发送，即应用程序彼此不直接通信，而是与作为中介的 MOM 通信。MOM 提供了有保证的消息发送(至少是在尽可能地做到这一点)，应用程序开发人员无需了解远程过程调用(PRC)和网络/通信协议的细节。

◆ JMS

JMS 即 Java 消息服务（Java Message Service）应用程序接口是一个 Java 平台中关于面向消息中间件（MOM）的 API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

2002 年 3 月 18 日正式推出 JMS1.1 规范，2013 年底推出了最新的 JMS2.0 规范。

◆ JNDI

JNDI(Java Naming and Directory Interface)是 SUN 公司提供的一种标准的 Java 命名系统接口，JNDI 提供统一的客户端 API，通过不同的访问提供者接口 JNDI SPI 的实现，由管理者将 JNDI API 映射为特定的命名服务和目录系统，使得 Java 应用程序可以和这些命名服务和目录服务之间进行交互。务将名称和对象联系起来，使得我们可以用名称访问对象。

◆ JMX

JMX（Java Management Extensions，即 Java 管理扩展）是一个为应用程序、设备、系统等植入管理功能的框架。JMX 可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议，灵活的开发无缝集成的系统、网络和服务管理应用。

◆ JCA

JCA (J2EE 连接器架构，Java Connector Architecture)是对 J2EE 标准集的重要补充。因为它注重的是将 Java 程序连接到非 Java 程序和软件包中间件的开发。连接器特指基于 Java 连接器架构的源适配器，其在 J2EE1.3 规范中被定义。JCA 连接器同时提供了一个重要的能力，即它使 J2EE 应用服务器能够集成任何使用 JCA 适配器的企业信息系统（EIS），大大简化了异构系统的集成。

◆ XA

X/Open 组织（即现在的 Open Group ）定义的分布式事务处理模型。是交易中间件与数据库之间的接口规范（即接口函数），交易中间件用它来通知数据库事务的开始、结束以及提交、回滚等。 XA 接口函数由数据库厂商提供。

◆ MDB

MDB(Message DrivenBean)消息驱动 Bean.它是 EJB 跟 JMS 的一个整合,跟 SessionBean 一样,MDB 是由 EJB 容器进行管理,同时也可以利用 EJB 所提供的系统服务诸如事务,安全等.

消息驱动 Bean 是一个异步消息消费者。当消息到达消息驱动 Bean 服务的目的地或终端时，容器调用消息驱动 Bean。

◆ JAAS

Java Authentication Authorization Service（JAAS，Java 验证和授权 API）提供了灵活和可伸缩的机制来保证客户端或服务器端的 Java 程序。

◆ AMQP

AMQP（高级消息队列协议）协议是一个二进制协议，拥有一些现代特点：多信道、协商式、异步、安全、跨平台、中立、高效。它使得遵从该规范的客户端应用和消息中间件服务器的全功能互操作成为可能。目标是实现一种在全

行业广泛使用的标准消息中间件技术，以便降低企业和系统集成的开销，并且向大众提供工业级的集成服务。通过 AMQP，让消息中间件的能力最终被网络本身所具有，并且通过消息中间件的广泛使用发展出一系列有用的应用程序。

◆ STOMP

Streaming Text Orientated Message Protocol，即流文本定向消息协议。是一个专为实现客户端之间通过中间服务器进行异步通信的简单可操作的协议，它为这些互相通信客户端和服务器定义了一种基于文本的消息通信格式。它是一种为 MOM 设计的简单文本协议。

它提供了一个可互操作的连接格式，允许 STOMP 客户端与任意 STOMP 消息代理(Broker)进行交互，类似于 OpenWire(一种二进制协议)。

由于其设计简单，很容易开发客户端，因此在多种语言和多种平台上得到广泛应用。

◆ REST

表述性状态转移（Representational State Transfer，简称 REST）是 Roy Fielding 博士在 2000 年他的博士论文中提出来的一种软件架构风格。它是一种针对网络应用的设计和开发方式，可以降低开发的复杂性，提高系统的可伸缩性。随着云技术的全面推进，REST 架构风格被越来越推崇而成为主流。

◆ Netty

Netty 是 JBoss 推出的一个高性能的网络编程框架，Netty 使用 NIO 技术，提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序。

和 apache 的 Mina 框架一样出色，成为 java 底层传输的不二之选。

第二章 产品介绍

2.1 消息中间件

2.1.1 中间件的分类

按照 IDC 的分类方法，中间件可分为六类：

- 终端仿真/屏幕转换
- 数据访问中间件(UDA)
- 远程过程调用中间件(RPC)
- 消息中间件(MOM)
- 交易中间件(TPM)
- 对象中间件

2.1.2 消息中间件的介绍

中间件是指利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。

消息中间件可以即支持同步方式，又支持异步方式。异步中间件比同步中间件具有更强的容错性，在系统故障时可以保证消息的正常传输。异步中间件技术又分为两类：广播方式和发布/订阅方式。由于发布/订阅方式可以指定哪种类型的用户可以接受哪种类型的消息，更加有针对性，事实上已成为异步中间件的非正式标准。目前主流的消息中间件产品有 IBM 的 MQSeries，BEA 的 MessageQ 和 Sun 的 JMS 等。

2.1.3 消息中间件的原理

面向消息的中间件(MOM)，提供了以松散耦合的灵活方式集成应用程序的一种机制。它们提供了基于存储和转发的应用程序之间的异步数据发送，即应用程序彼此不直接通信，而是与作为中介的 MOM 通信。MOM 提供了有保证的消息发送(至少是在尽可能地做到这一点)，应用程序开发人员无需了解远程过程调用(PRC)和网络/通信协议的细节。

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。

消息中间件适用于需要可靠的数据传送的分布式环境。采用消息中间件机制的系统中，不同的对象之间通过传递消息来激活对方的事件，完成相应的操作。发送者将消息发送给消息服务器，消息服务器将消息存放在若干队列中，在合适的时候再将消息转发给接收者。消息中间件能在不同平台之间通信，它常被用来屏蔽掉各种平台及协议之间的特性，实现应用程序之间的协同，其优点在于能够在客户和服务器之间提供同步和异步的连接，并且在任何时刻都可以将消息进行传送或者存储转发，这也是它比远程过程调用更进一步的原因。

MOM 将消息路由给应用程 B，这样消息就可以存在于完全不同的计算机上，MOM 负责处理网络通信。如果网络连接不可用，MOM 会存储消息，直到连接变得可用时，再将消息转发给应用程序 B。

灵活性的另一方面体现在，当应用程序 A 发送其消息时，应用程序 B 甚至可以不处于执行状态。MOM 将保留这个消息，直到应用程序 B 开始执行并试着检索消息为止。这还防止了应用程序 A 因为等待应用程序 B 检索消息而出现阻塞。 这种异步通信要求应用程序的设计与现在大多数应用程序不同，不过，对于时间无关或并行处理，它可能是一个极其有用的方法。

2.1.4 消息中间件和分布式对象调用的比较

分布式对象调用，如 CORBA，RMI 和 DCOM，提供了一种通讯机制，透

明地在异构的分布式计算环境中传递对象请求，而这些对象可以位于本地或远程机器。它通过在对象与对象之间提供一种统一的接口，使对象之间的调用和数据共享不再关心对象的位置、实现语言及所驻留的操作系统。这个接口就是面向对象的中间件。

尽管面向对象的中间件是一种很强大的规范被广泛应用，但是面对大规模的复杂分布式系统，这些技术也显示出了局限性：

1.同步通信：客户发出调用后，必须等待服务对象完成处理并返回结果后才能继续执行。

2.客户和服务对象的生命周期紧密耦合：客户进程和服务对象进程都必须正常运行，如果由于服务对象崩溃或网络故障导致客户的请求不可达，客户会接收到异常。

为了解决这些问题，出现了面向消息的中间件，它较好地解决了以上的问题。

消息中间件作为一个中间层软件，它为分布式系统中创建、发送、接收消息提供了一套可靠通用的方法，实现了分布式系统中可靠的、高效的、实时的跨平台数据传输。消息中间件减少了开发跨平台和网络协议软件的复杂性，它屏蔽了不同操作系统和网络协议的具体细节，面对规模和复杂度都越来越高的分布式系统，消息中间件技术显示出了它的优越性：

1.采用异步通信模式：发送消息者可以在发送消息后进行其它的工作，不用等待接收者的回应，而接收者也不必在接到消息后立即对发送者的请求进行处理;

2.客户和服务对象生命周期的松耦合关系：客户进程和服务对象进程不要求都正常运行，如果由于服务对象崩溃或者网络故障导致客户的请求不可达，客户不会接收到异常，消息中间件能保证消息不会丢失。

2.2 JMS 简述

2.2.1 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。

2.2.2 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。

2.2.3 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）。

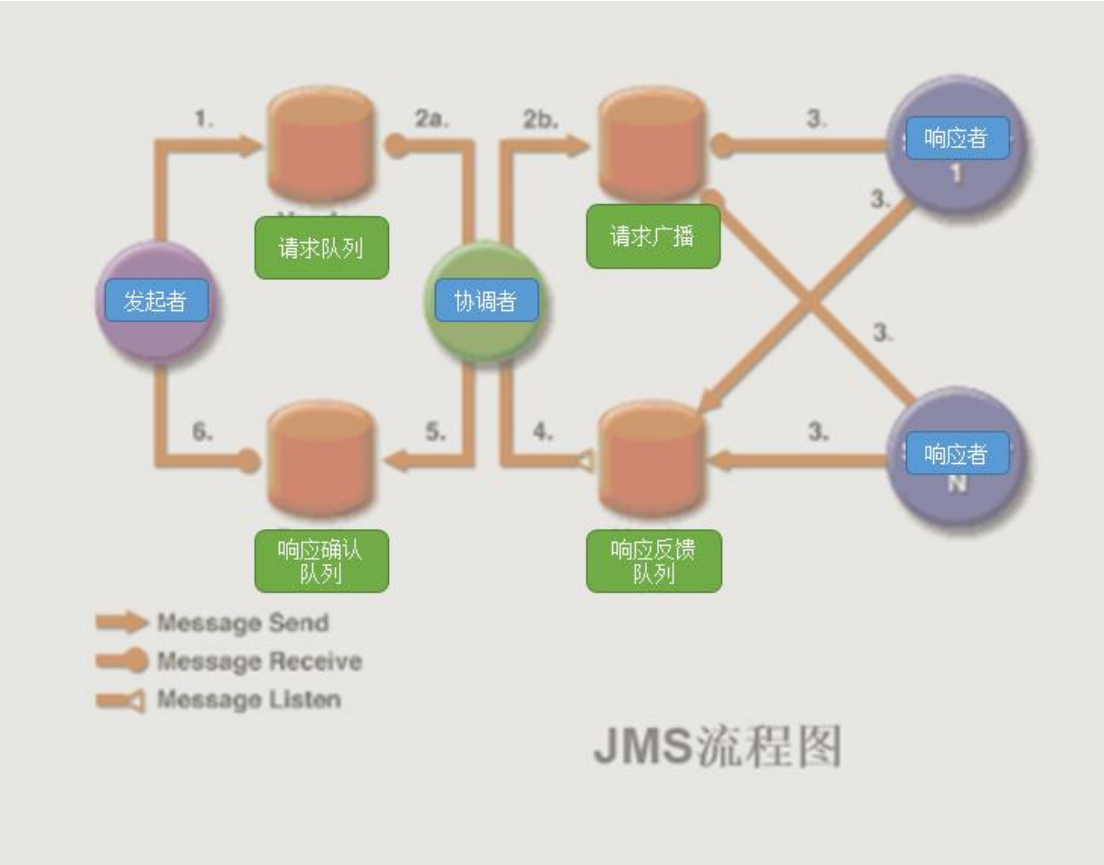
消息接口非常灵活，并提供了许多方式来定制消息的内容。

◆ 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。

2.2.4 JMS 工作流程

从概念上说，生产者和消费者是独立的，但实际应用基本上一个实体既是消费者又是生产者，下面用一个结合了队列和主题的复杂的场景来陈述 JMS 的工作流程。



1. 发起者通过请求队列向协调者发起请求；
2. 协调者在请求主题中向众多响应者通告请求；
3. 响应者在反馈队列中向协调者提交响应；
4. 协调者在确认队列中向发起者反馈收集来的响应；

这个流程有三类实体和四个队列，既有点对点模式，又有发布订阅模式，有同步消息，有异步消息，一个实体既有生产者的角色又有消费者的脚色，但对于一个特定队列来说，其生产者或消费者的角色是确定的。

2.3 Rcloud MQ 介绍

Rcloud MQ 是一个多协议、可嵌入、高性能、可集群的异步消息系统。是纯 java 开发的消息中间件，实现了 JMS1.1 和 JMS2 的规范。可以在任何 Java 6+ 的平台上运行。

Rcloud MQ 是中软国际 Rcloud 平台的一个标准产品，用来给 IPaaS 平台提供消息服务支持。

2.3.1 Rcloud MQ 的特点

Rcloud MQ 作为一个 MoM，不但实现了主流消息中间件全部的功能，而且有自身的许多特点。

◆ JMS2.0 支持

JMS2.0 是 2013 年发布了公共审查草案，因此许多主流 JMS 服务器都还不支持许多特性。

◆ 性能出众

消息中间件作为分布式系统中的核心组件，性能是衡量其好坏的重要指标。Rcloud MQ 借鉴各主流 MoM 的设计理念，通过独特的架构，不但对非持久化的消息处理达到了非常高的性能。独特高效的文件系统使持久消息处理接近非持久消息的性能。

◆ 功能全面

包括点对点，分发订阅模式，同步异步发送，消息过滤，消息优先级，消息过期，坏死队列，消息转发，消息中继，超大消息；支持多种传输模式和协议；支持 JAAS 安全规范；XA 事务规范和 JMX 管理规范。

◆ 简约设计

Rcloud MQ 设计上遵从了简约的原则。对第三方软件的依赖极少。根据不同的需要， 既可以单独运行，也可以运行于 JEE 应用服务器中。它还可以嵌入到你自己的应用程序中。

◆ 高可用性

Rcloud MQ 提供自动客户端失效备援（automatic client failover）功能，能保证在服务器故障时没有消息丢失或消息重复。

◆ 支持集群

Rcloud MQ 提供超级灵活的集群方案。可以控制集群进行消息负载均衡的方式，根据每个节点接收者（consumer）的多少以及是否具有接收状态，消息在

集群中可以进行智能化负载均衡。

还能够在集群中的节点间进行消息的再分发，以避免在某个节点出现消息匮乏（starvation）现象。 也可以非常灵活地配置消息路由。

◆ REST 服务

Rcloud MQ 提供 REST 服务接口，不但使 Rcloud MQ 便于和各种客户端集成，也使得 Rcloud MQ 具备云消息服务的能力。

◆ 全面管控

Rcloud MQ 提供多种对服务和队列进行管控的方式，使得系统管理及其便捷和高效。对队列吞吐情况的详细监控也便于开发人员分析系统的瓶颈，优化应用。

◆ 极易开发

Rcloud MQ 在开发方面提供了不同层次的接口，不但提供了标准的 API，也提供了复杂的核心 API，更提供了极其傻瓜的简化 API，方便了客户端开发。

2.3.2 Rcloud MQ 工作模式

Rcloud MQ 提供了三种模式可供系统使用。

◆ 独立模式

Rcloud MQ 自带了 Web 服务，JNDI 服务和 Rest 服务，因此他可以独立运行提供全部的 JMS 功能。该模式也支持集群工作，第三章会详细该模式下服务安装和配置。

◆ 集成模式

Rcloud MQ 提供了标准的 JCA 适配器，利用它可以将 RCloudMQ 轻松地集成到任何一个符合 JEE 规范的应用服务器或 servlet 容器中。第三章会详细该模式下服务安装和配置。

◆ 嵌入模式

如果你的应用程序内部需要消息服务，但同时你又不想将消息服务暴露为单独的服务器，你可以在应用中直接对 Rcloud MQ 实例化，使你的应用具备 JMS 能力。该模式不提供集群功能。第三章会详细该模式下服务安装和配置。

第三章 安装部署指南

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 安装包的产品目录结构如下。

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

3.3 独立模式

独立模式只需要安装好JDK（至少1.6），启动startup即可。

3.3.1 主文件配置

这里先讲普通模式下的配置，集群模式下的配置在第六章详细说明。

主文件是mq-configuration.xml，配置服务的主要工作参数。包括数据路径，持久化路径；客户端连接器和接受器；

```
- <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:RCloudMQ">
    <paging-directory>${data.dir:../data}/paging</paging-directory>
    <bindings-directory>${data.dir:../data}/bindings</bindings-directory>
    <journal-directory>${data.dir:../data}/journal</journal-directory>
    <journal-min-files>10</journal-min-files>
    <large-messages-directory>${data.dir:../data}/large-messages</large-messages-directory>
  - <connectors>
      - <connector name="netty">
            <factory-class>org.RCloudMQ.core.remoting.impl.netty.NettyConnectorFactory</factory-class>
            <param value="${RCloudMQ.remoting.netty.host:localhost}" key="host"/>
            <param value="${RCloudMQ.remoting.netty.port:5445}" key="port"/>
        </connector>
      - <connector name="netty-throughput">
            <factory-class>org.RCloudMQ.core.remoting.impl.netty.NettyConnectorFactory</factory-class>
            <param value="${RCloudMQ.remoting.netty.host:localhost}" key="host"/>
            <param value="${RCloudMQ.remoting.netty.batch.port:5455}" key="port"/>
            <param value="50" key="batch-delay"/>
        </connector>
    </connectors>
  - <acceptors>
      - <acceptor name="netty">
            <factory-class>org.RCloudMQ.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
            <param value="${RCloudMQ.remoting.netty.host:localhost}" key="host"/>
            <param value="${RCloudMQ.remoting.netty.port:5445}" key="port"/>
        </acceptor>
      - <acceptor name="netty-throughput">
            <factory-class>org.RCloudMQ.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
            <param value="${RCloudMQ.remoting.netty.host:localhost}" key="host"/>
            <param value="${RCloudMQ.remoting.netty.batch.port:5455}" key="port"/>
            <param value="50" key="batch-delay"/>
            <param value="false" key="direct-deliver"/>
        </acceptor>
    </acceptors>
```

I相关文件路径配置

链接器配置

接受器配置

```
- <security-settings>
    - <security-setting match="#">
          <permission roles="guest" type="createNonDurableQueue"/>
          <permission roles="guest" type="deleteNonDurableQueue"/>
          <permission roles="guest" type="consume"/>
          <permission roles="guest" type="send"/>
      </security-setting>
  </security-settings>
- <address-settings>
          <!--default for catch all-->
    - <address-setting match="#">
          <dead-letter-address>jms.queue.DLQ</dead-letter-address>
          <expiry-address>jms.queue.ExpiryQueue</expiry-address>
          <redelivery-delay>0</redelivery-delay>
          <max-size-bytes>10485760</max-size-bytes>
          <message-counter-history-day-limit>10</message-counter-history-day-limit>
          <address-full-policy>BLOCK</address-full-policy>
      </address-setting>
  </address-settings>
</configuration>
```

权限配置

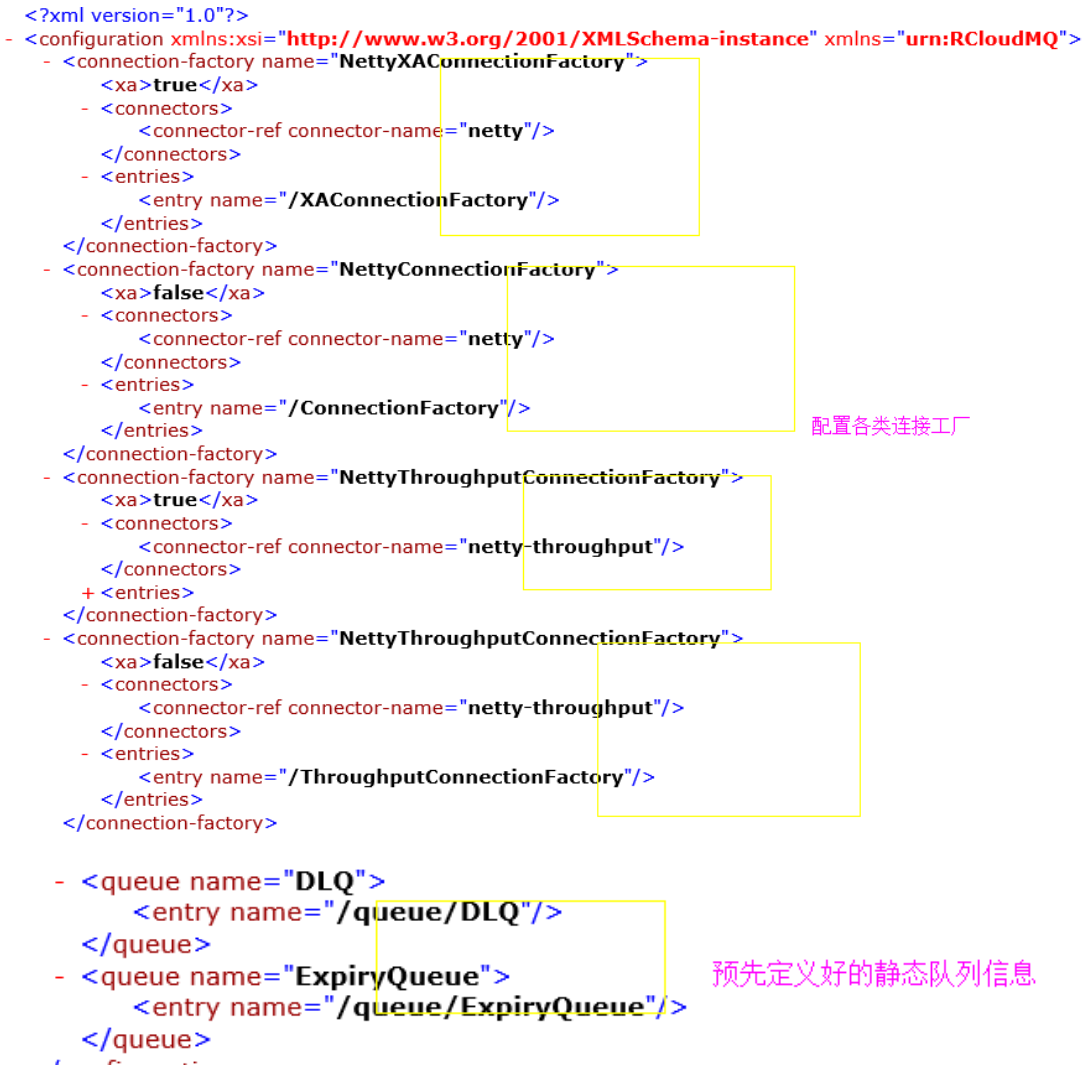
地址工作参数配置

该文件一般作为全局使用，建议不要包含其他JMS相关的队列信息，和应用相关的统一在jms.xml中配置。

除了权限配置部分外，其他参数建议使用缺省配置，如需要做特殊配置，附录一有详细的配置参数说明。

3.3.2 JMS 配置文件

mq-jms.xml文件是和应用相关的jms的配置文件，定义了各种连接工厂和静态定义的的队列信息。该文件要根据系统需要进行定义和配置。




```
<?xml version="1.0"?>
- <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:RCloudMQ">
  - <connection-factory name="NettyXAConnectionFactory">
      <xa>true</xa>
    - <connectors>
        <connector-ref connector-name="netty"/>
      </connectors>
    - <entries>
        <entry name="/XAConnectionFactory"/>
      </entries>
    </connection-factory>
  - <connection-factory name="NettyConnectionFactory">
      <xa>false</xa>
    - <connectors>
        <connector-ref connector-name="netty"/>
      </connectors>
    - <entries>
        <entry name="/ConnectionFactory"/>
      </entries>
    </connection-factory>
  - <connection-factory name="NettyThroughputConnectionFactory">
      <xa>true</xa>
    - <connectors>
        <connector-ref connector-name="netty-throughput"/>
      </connectors>
    + <entries>
    </connection-factory>
  - <connection-factory name="NettyThroughputConnectionFactory">
      <xa>false</xa>
    - <connectors>
        <connector-ref connector-name="netty-throughput"/>
      </connectors>
    - <entries>
        <entry name="/ThroughputConnectionFactory"/>
      </entries>
    </connection-factory>

  - <queue name="DLQ">
      <entry name="/queue/DLQ"/>
    </queue>
  - <queue name="ExpiryQueue">
      <entry name="/queue/ExpiryQueue"/>
    </queue>
```

3.3.3 用户配置文件

mq-user.xml 配置文件用来配置角色和权限信息，保证了 JMS 服务的安全访问。

```
    <user name="bill" password="RCloudMQ">
        <role name="user" />
    </user>

    <user name="andrew" password="RCloudMQ1">
        <role name="europe-user" />
        <role name="user" />
    </user>

    <user name="frank" password="RCloudMQ2">
        <role name="us-user" />
        <role name="news-user" />
        <role name="user" />
    </user>

    <user name="sam" password="RCloudMQ3">
        <role name="news-user" />
        <role name="user" />
    </user>

</configuration>
```

该文件定义了 jms 客户端的用户信息和权限分配，role 是在主配置文件里定义的。下面是在 mq-configuration.xml 文件对应的配置。

```
    <security-setting match="jms.topic.news.europe.#">
        <permission type="createDurableQueue" roles="user"/>
        <permission type="deleteDurableQueue" roles="user"/>
        <permission type="createNonDurableQueue" roles="user"/>
        <permission type="deleteNonDurableQueue" roles="user"/>
        <permission type="send" roles="europe-user"/>
        <permission type="consume" roles="news-user"/>
    </security-setting>

    <security-setting match="jms.topic.news.us.#">
        <permission type="createDurableQueue" roles="user"/>
        <permission type="deleteDurableQueue" roles="user"/>
        <permission type="createNonDurableQueue" roles="user"/>
        <permission type="deleteNonDurableQueue" roles="user"/>
        <permission type="send" roles="us-user"/>
        <permission type="consume" roles="news-user"/>
    </security-setting>
```

这里可以详细到作用的队列，用通配符来筛选

具体可执行的操作

就是角色，和用户表对应

3.3.4 JNDI 配置文件

jndi.properties 文件是用来配置 JNDI 的命名工厂。

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming\:org.jnp.interfaces
```

其 JNDI 的配置参数在 mq-beans.xml 中定义。

```
<bean name="JNDIServer" class="org.jnp.server.Main">
  <property name="namingInfo">
    <inject bean="Naming"/>
  </property>
  <property name="port">1099</property>
  <property name="bindAddress">192.168.2.101</property>
  <property name="rmiPort">1098</property>
  <property name="rmiBindAddress">192.168.2.101</property>
</bean>
```

一般在嵌入模式不需要配置，在集成模式下使用 JEE 自带的 JNDI 服务。

3.3.5 日志配置文件

logging.properties 文件定义了日志相关的参数。

```
# 根日志级别
logger.level=INFO
# RCloudMQ 日志级别
logger.org.RCloudMQ.core.server.level=INFO
logger.org.RCloudMQ.journal.level=INFO
logger.org.RCloudMQ.utils.level=INFO
logger.org.RCloudMQ.jms.level=INFO
logger.org.RCloudMQ.integration.bootstrap.level=INFO

# 控制台日志
handler.CONSOLE=org.jboss.logmanager.handlers.ConsoleHandler
handler.CONSOLE.properties=autoFlush
handler.CONSOLE.level=FINE
handler.CONSOLE.autoFlush=true
handler.CONSOLE.formatter=PATTERN

# 文件日志
handler.FILE=org.jboss.logmanager.handlers.FileHandler
handler.FILE.level=FINE
handler.FILE.properties=autoFlush,fileName
handler.FILE.autoFlush=true
handler.FILE.fileName=logs/RCloudMQ.log
handler.FILE.formatter=PATTERN
```

3.3.6 客户端配置文件

mq-client.xml 文件是客户端应用中需要配置的文件，一般来说 Rcloud MQ 不需要该文件。因为 Rcloud MQ 提供了 JMS 客户端实例，因此有个配置范例。客户端开发时需要参考配置该文件。

```
<configuration xmlns="urn:RCloudMQ"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:RCloudMQ /schema/RCloudMQ-configuration.xsd">

   <!-- Connectors -->
   <connectors>
      <connector name="netty-connector">
         <factory-class>com.chinasoft.rcloud.mq.remoting.impl.netty.NettyConnectorFactory
         </factory-class>
      </connector>
   </connectors>

</configuration>
```

3.4 集成模式

在集成模式下，RCloudMQ 的使用也很简单，加入 RCloudMQ.jar 包，在 classpath 下加入配置文件，然后修改 web.xml 配置文件即可。

其他配置文件的参考上面独立模式的配置，如果是在 JEE 集成环境下实现集群，需要根据实际的容器进行调整，总体方案可以参考独立模式下的集群配置方式。

3.5 嵌入模式

嵌入模式的使用也十分简单，只需要加入 RCloudMQ.jar 包，在应用中实例服务，并启动即可。

该模式不支持集群方式。


IcssMQ嵌入式服务
// IcssMq嵌入式服务器
//获取服务实例，
EmbedServerMgr server = EmbedServerMgr.getServerMgr(null);
//启动服务
server.start();

第四章 快速开发指南

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。

4.1 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。

```
String msg =  request.getParameter("msg");
if(msg != null){
   try {
      MqJmsClient.getClient().sendMsg(msg);
      json = "true";
   } catch (Exception e) {
      json = "false";
   }
}
```

4.2 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。

```
}else if(opt.equals("getMsg")){
   try {
      json = MqJmsClient.getClient().getMsg();
   } catch (Exception e) {
      json = "false";
   }
```

4.3 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。

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。

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。

第五章 服务管理指南

Rcloud MQ 提供了多种管理方式来实现 JMS 服务的高效便捷管理。

5.1 WEB 管理

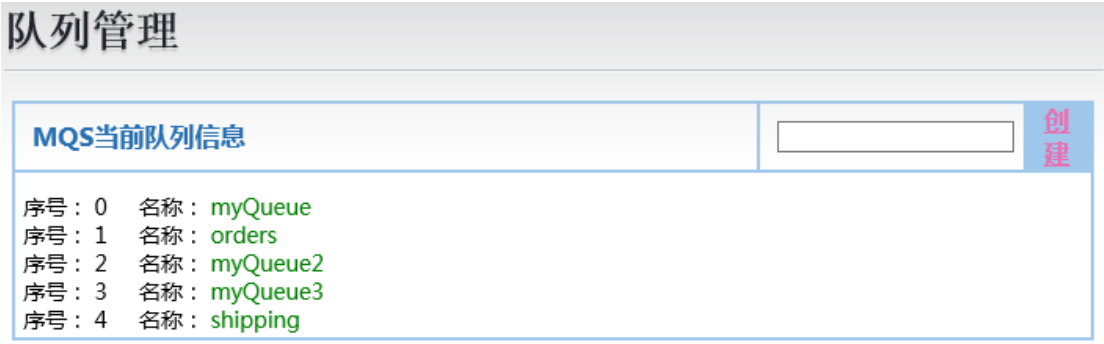
Rcloud MQ 服务器自带了一个 web 管理服务，通过访问 localhost:8080/mqAdmin/就可以进入管理界面。

5.1.1 服务监控

服务监控提供了服务器的运行监控，客户端连接监控，和当前负载监控，如果是集群模式下，还提供节点监控功能。

5.1.2 队列管理

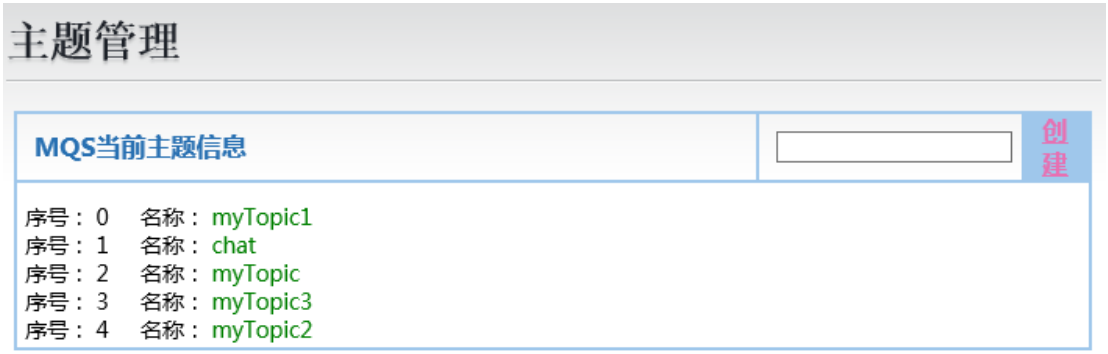
队列管理提供了对当前队列的查询，新建队列，删除队列，清空队列以及查看队列的详细负载信息等操作。



5.1.3 主题管理

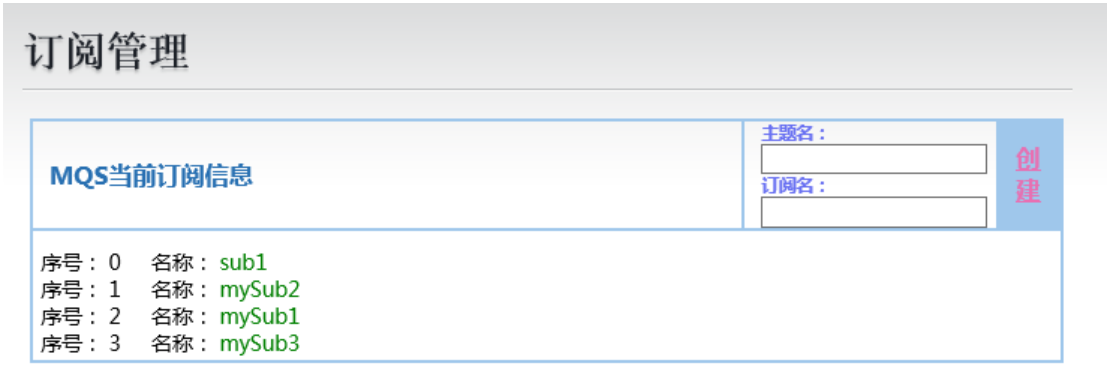
主题管理提供了对当前发布者的查询，新建主题，删除主题，清空主题以

及查看主题的详细负载信息和订阅信息等操作。



5.1.4 订阅管理

订阅管理提供了对当前订阅者的查询，新建订阅，删除订阅，清空订阅以及查看订阅的详细负载信息等操作。



5.2 JMX 管理

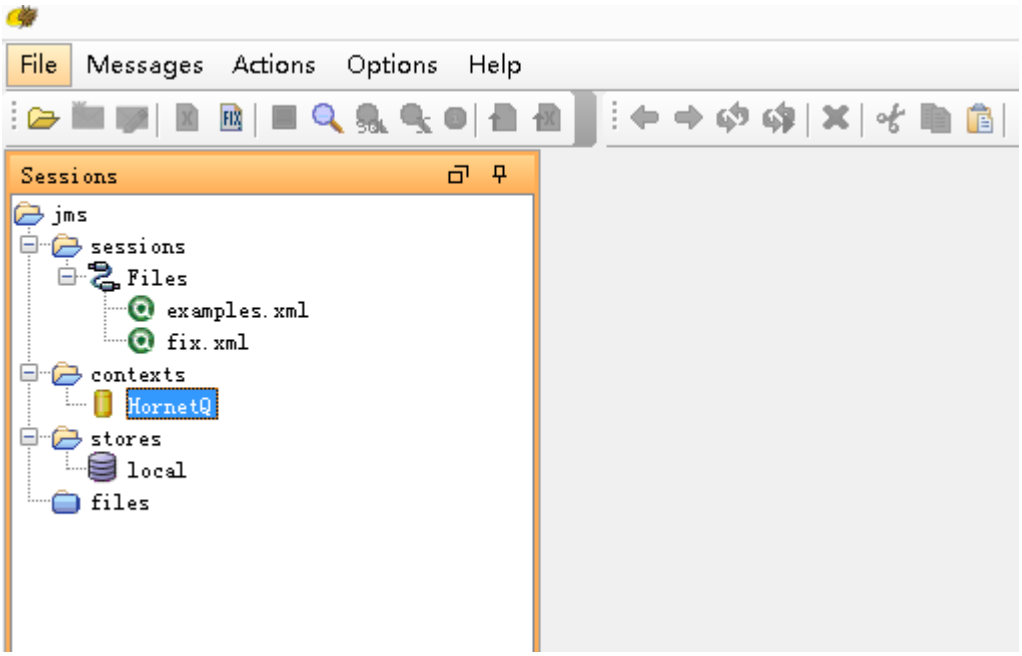
JMX 是标准的 java 管理接口，CloudMQ 实现了该接口，因此可以很方便的实现管理。

默认情况下 JMX 接口是打开的。因此可以使用 jconsole 来管理本地的 RCloudMQ。将 mq-configuration.xml 文件中的 jmx-management-enabled 设置为 false 就可以关闭 JMX。

```
<jmx-management-enabled>false</jmx-management-enabled>
```

5.3 JMS 管理

RCloudMQ 实现了 JMS 管理的接口，因此可以通过 JMS 进行管理。Hermes 提供的开源 JMS 工具功能十分强大，可以进行各种监控。



第六章 服务配置指南

Rcloud MQ 提供了非常丰富的 JMS 功能，一般来说按照系统缺省的配置方式可以满足大部分需求，如果需要进行某些方面的调整，则需要用户进行参数配置。

6.1 传输层的配置

RCloudMQ 的传输层是“可插拔的”。通过灵活的配置和一套服务提供接口（SPI），RCloudMQ 可以很容易地更换其传输层来满足不同的需求。

6.1.1 接收器（Acceptor）配置

接收器定义的是如何在服务器端接收客户端的连接，在主配置文件中进行配置，一般配置格式如下：

```
<acceptors>
   <acceptor name="netty">
      <factory-class>
         com.chinasoft.rcloud.mq.core.remoting.impl.netty.NettyAcceptorFactory
      </factory-class>
      <param key="port" value="5446"/>
   </acceptor>
</acceptors>
```

每个接收器都要定义其与 MQ 服务器连接的方式,以上的例子中我们定义了一个 Netty 接收器。它在端口 5446 监听连接请求。

所有接收器都在 acceptors 单元（element）内定义。在 acceptors 内可以有零个或多个接收器的定义。每个服务器所拥有的接收器的数量是没有限制的。

6.1.2 连接器（Connector）配置

连接器是定义客户端如何连接到服务器。连接器的配置在 connectors 单元中。

可以定义一个或多个连接器。每个服务器配置的连接器 数量是没有限制的。

```
<connectors>
    <connector name="netty">
        <factory-class>
            com.chinasoft.rcloud.mq.core.remoting.impl.netty.NettyConnectorFactory
        </factory-class>
        <param key="port" value="5446"/>
    </connector>
</connectors>
```

一般来说，连接器的配置是在客户端进行，即配置在 mq-client.xml 文件中，客户端通过 JDNI 来获取连接工厂进行连接。如下例子就是引用了主配置文件中的连接器实例。

```
<connection-factory name="ConnectionFactory">
    <connectors>
       <connector-ref connector-name="netty"/> //在这里进行引用
    </connectors>
    <entries>
        <entry name="ConnectionFactory"/>
        <entry name="XAConnectionFactory"/>
    </entries>
</connection-factory>
```

6.1.3 配置 Netty 传输层

RCloudMQ 的传输层默认使用 Netty 技术实现 TCP 连接，可以配置多种不同的方法来满足实际传输需求。它可以使用传统的 Java IO（阻塞方式）、NIO（非阻塞）或直接使用 TCP socket 及 SSL。或者使用 HTTP 或 HTTPS 协议。同时还可能使用 servlet 进行传输。

6.1.3.1 配置通用 TCP

RCloudMQ 缺省的是使用非加密的基于 TCP socket 的传输。它可以使用阻塞式的 Java IO 或非阻塞式的 Java NIO。 我们建议在服务器端采用非阻塞式的 NIO 以获得良好的并发处理能力。当并发能力并不是很重要时，可以使用阻塞式 的方式以增加响应的速度。

下面分别介绍几个常用的配置项的使用方法。

use-nio

如果设为true则使用非阻塞的Java NIO。如果false则使用传统的阻塞方式的Java IO。

我们建议使用Java NIO处理并行连接。因为Java NIO不是为每一个连接分配一个线程，所以它要比传统的阻塞式 Java IO具有更强的并发连接的处理能力。如果你不需要处理并发连接，那么使用旧的阻塞式的IO性能会好一些。

这个参 数的默认值在服务器端是false，在客户端是false。

tcp-send-buffer-size

这个参数指定了TCP的发送缓冲大小，单位是字节。 默认值是32768字节(32KiB)。

这个参数要根据你的网络的带宽与时延的情况而调整。

TCP的发送 / 接收缓冲的大小可以用下面公式来计算：

缓冲大小 = 带宽 * RTT

其中带宽的单位是 每秒字节数，

RTT（网络往返程时间）的单位是秒。 使用ping工具可以方便地测量出RTT。

对于快速网络可以适当加大缓冲的大小。

batch-delay

配置该参数，在数据包写入传输层之前有一个 最大延时（毫秒），达到批量写入的目的。

这样可以提高小消息的发送效率。但这样做会增加单个消息的平均发送 延迟。

默认值为0毫秒。

direct-deliver

消息到达服务器后，一般会由开启不同的线程来将消息传递 到接收者。这样可以使服务的吞吐量和可扩展性达到最佳，特别是在多核的系统上效果更为明显。但是线程切换 会带来一些传递的延迟。

如果你希望延迟最小，并不在意吞吐量的话，可以将参数direct-deliver设为true。

如果你更希望有 较大的吞吐量的话，将它设为false。

默认值是true。

nio-remoting-threads

如果使用NIO，默认情况下MQ会使用Cpu内核（或超线程）数量三倍的线程来处理接收的数据包。

如果你想改变这个数量， 你可以设定本参数。

默认的值是-1，表示其线程数为3倍内核数。

更多详细的配置属性说明参见附录表。

6.1.3.2 配置 SLL 加密 TCP

SSL 的配置与普通 TCP 相似。它采用了安全套接字层（SSL）来提供加密的 TCP 连接。普通 TCP 的配置参数都对它适用，另外需要配置以下属性。

ssl-enabled 必须设为true以使用SSL。 key-store-path 存放SSL密钥的路径（key store)。这是存放客户端证书的地方。 key-store-password 用于访问key store的密码。 trust-store-path 服务器端存放可信任客户证书的路径。 trust-store-password 用于访问可信任客户证书（trust store)的密码。

下面是一个配置 SSL 的实际例子。

```
<!-- SSL connector -->
<connector name="netty-ssl-connector">
  <factory-class>
    com.chinasoft.rcloud.mq.remoting.impl.netty.NettyConnectorFactory
  </factory-class>
  <param key="host" value="localhost"/>
  <param key="port" value="5500"/>
  <param key="ssl-enabled" value="true"/>
  <param key="key-store-path" value="F:/ssl/keystore"/>
  <param key="key-store-password" value="test"/>
</connector>
```

```
<!-- SSL acceptor -->
<acceptor name="netty-ssl-acceptor">
  <factory-class>
     com.chinasoft.rcloud.mq.remoting.impl.netty.NettyAcceptorFactory
  </factory-class>
  <param key="host" value="localhost"/>
  <param key="port" value="5500"/>
  <param key="ssl-enabled" value="true"/>
```

```
  <param key="key-store-path" value="F:/ssl/keystore"/>
  <param key="key-store-password" value="test"/>
  <param key="trust-store-path" value="F:/ssl/truststore"/>
  <param key="trust-store-password" value="test"/>
</acceptor>
```

6.1.3.3 配置 HTTP 传输

Netty HTTP 通过 HTTP 通道传送数据包。在有些用户环境中防火墙只允许有 HTTP 通信，这时采用 Netty HTTP 作为 MQ 的传输层就能解决问题。普通 TCP 的配置参数都对它适用，另外需要配置以下属性。

```
http-enabled
  如果要使用HTTP，这个参数必须设为true。

http-client-idle-time
  客户端空闲时间。
  如果客户端的空闲时间超过 这个值，Netty就会发送一个空的HTTP请求以保持连接不被关闭。

http-client-idle-scan-period
  扫描空闲客户端的间隔时间。单位是毫秒。

http-response-time
  服务器端向客户端发送空的http响应前的最大等待时间。

http-server-scan-period
  服务器扫描需要响应的客户端的时间间隔。单位是毫秒。

http-requires-session-id
  如果设为true，客户端在第一次请求后将等待接收一个会话ID。
  Http连接器用它来连接servlet接收器（不建议这样使用）。
```

下面是一个配置 HTTP 连接器的例子。

```
<!-- SSL connector -->
<connectors>
  <connector name="netty-connector">
    <factory-class>
   com.chinasoft.rcloud.mq.remoting.impl.netty.NettyConnectorFactory
    </factory-class>
```

```
    <param value="true" key="http-enabled"/>
    <param value="8080" key="port"/>
  </connector>
</connectors>
```

6.1.3.4 配置 Servlet 传输

MQ 可以使用 Netty servlet 来传输消息。使用 servlet 可以将 MQ 的数据通过 HTTP 传送到一个 运行的 servlet，再由 servlet 转发给 MQ 服务器。

servlet 与 HTTP 的不同之处在于，当用 HTTP 传输时，MQ 如同一个 web 服务器，它监听在某个端口上的 HTTP 请求并返回响应。比如 80 端口或 8080 端口。而当使用 servlet 时，MQ 的传输数据是通过运行在某一 servlet 容器 中的一个特定的 servlet 来转发的。而这个 sevlet 容器中同时还可能运行其他的应用，如 web 服务。

当一个公司有多个应用 但只允许一个 http 端口可以访问时，servlet 传输可以很好的解决 MQ 的传输问题。

配置 servlet 传输，需要做两方面的配置。第一是在主配置文件中定义连接器信息。

```
<!-- Servlet connector -->
<connector name="netty-servlet">
  <factory-class>
    com.chinasoft.rcloud.mq.remoting.impl.netty.NettyConnectorFactory
  </factory-class>
  <param key="host" value="localhost"/>
  <param key="port" value="5500"/>
  //下面两个参数进行设置
  <param key="use-servlet" value="true"/>
  <param key="servlet-path" value="/messaging/MqServlet"/>
</connector>
```

```
<!-- Servlet acceptor -->
<acceptor name="netty-invm">
  <factory-class>
     com.chinasoft.rcloud.mq.remoting.impl.netty.NettyAcceptorFactory
  </factory-class>
  //下面这两个个参数进行设置
```

```
  <param key="use-invm" value="true"/>
  <param key="host" value="icss.RCloudMQ"/>
</acceptor>
```

第二是需要在 web.xml 下新部署一个 servlet。

```
<servlet>
  <servlet-name>MQServlet</servlet-name>
  <servlet-class>
      com.chinasoft.rcloud.mq.socket.http.HttpTunnelingServlet
  </servlet-class>
  <init-param>
   <param-name>endpoint</param-name>
   <param-value>local:icss.RCloudMQ</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>MQServlet</servlet-name>
  <url-pattern>/MQServlet</url-pattern>
</servlet-mapping>
```

6.1.4 配置 Stomp 协议

Stomp 是一个基于文本的协议。使用 Stomp 协议的客户端可以与 Stomp 的代理（broker）进行通迅。

Stomp 客户端支持多种语言和平台，因此它有着很好的互操作性。

RCloudMQ 支持 Stomp 协议，有两种方式来实现。

6.1.4.1 普通 Stomp 支持

RCloudMQ 内建组件来支持 Stomp 功能。要使用 Stomp 发送与接收消息，必须配置一个 NettyAcceptor，其中的 protocol 参数值应设为 stomp：

```
<acceptor name="stomp-acceptor">
  <factory-class>
    com.chinasoft.rcloud.mq.remoting.impl.netty.NettyAcceptorFactory
```

```
  </factory-class>
  <param key="protocol"  value="stomp"/>
  <param key="port"  value="61613"/>
</acceptor>
```

RCloudMQ 支持 Stomp1.1 和 Stomp1.2。

6.1.4.2 通过 Web Sockets 使用 Stomp

RCloudMQ 还支持通过 Web Sockets 使用 Stomp。任何支持 Web Socket 的浏览器中可以利用 MQ 来发送和接收 Stomp 消息。

要使用些功能，必须配置一个 NettyAcceptor，并设置 protocol 的值为 stomp_ws：

```
<acceptor name="stomp-ws-acceptor">
  <factory-class>
    com.chinasoft.rcloud.mq.remoting.impl.netty.NettyAcceptorFactory
  </factory-class>
  <param key="protocol"  value="stomp_ws"/>
  <param key="port"  value="61614"/>
</acceptor>
```

6.1.5 配置 AMQP 协议

AMQP 是一个高级消息交换协议，用来在不同的中间件进行消息交换。RcloudMQ 支持 AMQP1.0 协议，配置也很简单，配置一个 NettyAcceptor，其中的 protocol 参数值应设为 amqp 即可。

```
<acceptor name="amqp-acceptor">
  <factory-class>
    com.chinasoft.rcloud.mq.remoting.impl.netty.NettyAcceptorFactory
  </factory-class>
  <param key="protocol"  value="AMQP"/>
  <param key="port"  value="5672"/>
</acceptor>
```

6.2 Rest 服务配置

RcloudMQ 提供基于 Rest 架构的服务接口，使得您可以大幅度提升系统的可靠性和可伸缩性。通过使用 Rest 接口，可以使用 HTTP 来实现消息的接收和发送，简化了系统开发的复杂性。

下面就是通过发送一个 http 消息来实现向订单队列来发送一个订单信息。

```
POST /queue/orders/create HTTP/1.1
Host: example.com
Content-Type: application/xml

<order>
   <name>Bill</name>
   <item>iPhone 4</item>
   <cost>$199.99</cost>
</order>
```

由于是 http 消息，因此可以用任何 web 技术来实现客户端开发。

6.2.1 Rest 服务配置

RcloudMQ 的 Rest 服务是采用开源框架 RESTEasy 来实现的，系统默认提供 Rest 服务，在 web.xml 配置文件里进行了配置。

```
    <filter>
        <filter-name>Rest-Messaging</filter-name>
        <filter-class>
          org.jboss.resteasy.plugins.server.servlet.FilterDispatcher
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>Rest-Messaging</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

6.2.2 Rest 参数配置

为了配置 rest 服务的相关参数，首先需要在 web.xml 中设置配置文件的路径信息。系统默认的是 mq-rest.xml 文件。

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <display-name>icss-mq-server</display-name>

  <context-param>
    <param-name>rest.messaging.config.file</param-name>
    <param-value>mq-rest.xml</param-value>
  </context-param>
```

下面是一个配置示例：

```
<rest-messaging>
  <server-in-vm-id>0</server-in-vm-id>
  <use-link-headers>false</use-link-headers>
  <default-durable-send>false</default-durable-send>
  <dups-ok>true</dups-ok>
  <topic-push-store-dir>topic-push-store</topic-push-store-dir>
  <queue-push-store-dir>queue-push-store</queue-push-store-dir>
  <producer-time-to-live>0</producer-time-to-live>
  <producer-session-pool-size>10</producer-session-pool-size>
  <session-timeout-task-interval>1</session-timeout-task-interval>
  <consumer-session-timeout-seconds>
   300
   </consumer-session-timeout-seconds>
  <consumer-window-size>-1</consumer-window-size>
</rest-messaging>
```

各个参数的解释如下表：

server-in-vm-id 该参数设定rest服务和MQ进行通信的server id，默认为0。
use-link-headers 该参数确定是否使用客户端的链接头，默认为false。
default-durable-send 该参数设定当客户请求中没有持久化参数时，是否进行持久化。默认为false。

dups-ok

该参数设定true,则发送数据时不会启用重复检测协议。默认为true。

topic-push-store-dir

该参数为相对或绝对路径，用来存放那些需要推送的注册主题。

queue-push-store-dir

该参数为相对或绝对路径，用来存放那些需要推送的注册队列。

producer-session-pool-size

MQ发送消息时采用session池，该参数设定会话池的大小，默认为10。

producer-time-to-live

该参数设定信息发送的延迟时间，默认为0。

session-timeout-task-interval

该参数设定检查pull消息的客户端会话是否超时的间隔时间，默认为1秒。

consumer-session-timeout-seconds

该参数设定pull消息的客户端会话超时时间，默认为300秒。

consumer-window-size

该参数设定接收者流控制容器大小。默认为-1即不限流。

6.3 队列和路由配置

6.3.1 模版路由

RCloudMQ 提供使用带通配符的地址对消息进行路由。使用这个功能就可以允许创建“话题组”

例如，当创建一个队列时使用了地址 queue.news.#，那么它就能接收所有和这个地址通配符相配的每一个地址的消息。这如 queue.news.europe 或 queue.news.usa 或 queue.news.usa.sport 等。这样一个消息接收者可以接收一组相关的地址的消息，而不是只能指定一个具体的地址。

RCloudMQ 模式支持模版路由，在主配置文件进行配置。

wild-card-routing-enabled
是否支持模板路由，默认为true。

6.3.2 模板表达式

RCloudMQ 使用了一种专门的通配符语法来配置安全、地址及接收者（consumer）的创建。这种语法与 AMQP 所用的语法相似。

- 一个RCloudMQ的通配符表达式是由一些由“.”分隔的单词组成。
- 特殊字符“#”和“*”在表达式中可作为一个单词，它们代表 特殊的意义。
- 字符“#”表示“零或多个单词的任意排列”。
- 字符“*”表示“一个单词”。

因此，通配符表达式“news.europe.#”可以匹配“news.europe”、“news.europe.sport”、“news.europe.politics”以及“news.europe.politics.regional”，但是与“news.usa”、 “news.usa.sport” 及 “entertainment”不相匹配。

通配符“news.*”与“news.europe”匹配，但不与“news.europe.sport”匹配。

通配符“news.*.sport”与“news.europe.sport”及“news.usa.sport”匹配，但与 “news.europe.politics”不匹配。

6.3.3 队列配置

RcloudMQ 可以静态方式预先定义队列，也可以在客户端程序中动态创建队列，在动态创建时，还可以创建临时队列，在会话关闭时自动销毁队列。

有两种队列，JMS 标准队列和 MQ 核心队列。JMS 标准队列在 mq-jms.xml 文件中进行配置。核心队列在主配置文件中进行配置。

6.3.3.1 JMS 队列配置

下面是一个常见的 jms 队列示例。

```
<queue name="selectorQueue">
      <entry name="/queue/selectorQueue"/>
      <selector string="color='red'"/>
      <durable>true</durable>
</queue>
```

配置参数说明如下：

name

该属性定义了队列的名字。例子中我们采用了一种命名的惯例，因些对应的核心队列的名字是jms.queue.selectorQueue。

entry

该属性定义的名字用来将队列绑定于JNDI。这是必不可少的。一个队列可以有多个entry定义，每个定义中的名字都绑定到同一个队列。

selector

该属性定义的是队列的选择器。定义了选择器后，只有与选择器相匹配的消息才能被加到队列中。 这是一个可选项。如果没有定义选择器，队列将默认没有选择器。

durable

该属性定义了队列是否是一个可持久的队列。这也是一个可选项，默认值是true。

6.3.3.2 核心队列配置

下面是对上面 jms 队列按照 mq 队列配置的示例。

```
<queues>
    <queue name="jms.queue.selectorQueue">
        <address>jms.queue.selectorQueue</address>
        <filter string="color='red'"/>
      <durable>true</durable>
    </queue>
</queues>
```

配置参数基本一致，区别说明如下：

- 队列的name属性是队列的真正名字，不是JMS中的名字。
- address一项定义了消息路由的地址。
- 没有entry单元。
- filter的定义使用核心过滤器语法，不是JMS的选择器语法。

6.3.3.3 通过地址来配置队列

为了对批量队列进行统一属性配置，RCloudMQ 提供通过在地址配置中使用通配符的方式来完成该操作，极大的方便了队列的管理工作。

```
<address-settings>
   <address-setting match="jms.queue.#">
      <dead-letter-address>jms.queue.deadLetterQueue</dead-letter-address>
      <max-delivery-attempts>3</max-delivery-attempts>
      <redelivery-delay>5000</redelivery-delay>
      <expiry-address>jms.queue.expiryQueue</expiry-address>
      <last-value-queue>true</last-value-queue>
      <max-size-bytes>100000</max-size-bytes>
      <page-size-bytes>20000</page-size-bytes>
      <redistribution-delay>0</redistribution-delay>
      <send-to-dla-on-no-route>true</send-to-dla-on-no-route>
      <address-full-policy>PAGE</address-full-policy>
    </address-setting>
</address-settings>
```

上表是一个地址配置的示例，在主配置文件里。下表说明每个属性的用法。

max-delivery-attempts

该参数定义了最大重传递的次数。一个消息如果反复传递超过 了这个值将会被发往死信地址 dead-letter-address。

redelivery-delay

该参数定义了重新传递的延迟。它控制MQ在重新传递一个被取消的消息时要等待的时间。

expiry-address

该参数定义了过期消息的发送地址。

last-value-queue

该参数定义一个队列是否使用最新值。

max-size-bytes

该参数用来设定地址的最大内存值。当消息占用内存超过此值时，进入分页转存模式。默认值为-1，即不进行分页转存。

page-size-bytes

用来设置地址的分页转存时每个分页文件的大小。 缺省为10MB (10 * 1024 * 1024 字节)。

redistribution-delay

定义了当最后一个接收者关闭时重新分配队列消息前所等待的时间。

send-to-dla-on-no-route

当一个消息被送到某个地址时，可能不会被路由到任何一个队列。 例如该地址没有绑定任何队列的情况，或者它所有的队列的选择器与该消息不匹配时。这样的消息通常情况下会被丢弃。这时 如果将这个参数设为true，则如果这个地址配置了死信地址的话，这样的消息就会被发送到该地址的死信地址（DLA）。

address-full-policy

该属性有三个可能的值：PAGE、 DROP 或 BLOCK。它决定了如果地址的消息所占用的内存达到了max-size-bytes所定义的值时，如何处理后继到来的消息。 默认值是PAGE，就是将后续的消息分页转存到磁盘上。DROP则表示丢弃后续的消息。BLOCK表示阻塞消息的发送方发送后续的消息。

6.4 消息配置

RcloudMQ 提供了强大的消息管理管理功能，通过不同的配置就可以应对复杂的应用需求。

6.4.1 消息重发和死信配置

当消息传递失败（比如相关的事务发生回滚），失败的消息将退回到队列中准备重新传递。这样就会出现一种情况极端，就是同一个消息会被反复的传递而总不成功，以至于使系统处于忙的状态。

对于这样的消息 RCloudMQ 提供两种处理方法：延迟再传递和死信地址。

6.4.1.1 延迟再传递

这种方法是让消息再次传递时有一定的时间延迟，这样客户端就有机会从故障中恢复，同时网络连接和 CPU 资源 也不致于被过度占用。

延迟再传递对于时常出现故障或事务回滚的客户端十分有用。如果没有延迟，整个系统可能会处于一种”疯狂“的状态。 就是消息被传递、回滚、再传递，这样反复不间断地进行着，将宝贵的 CPU 和网络资源占用。

延迟再传递是在主配置文件的地址属性中进行配置。

```
<address-settings>
    <!-- delay redelivery of messages for 5s -->
    <address-setting match="jms.queue.exampleQueue">
       <redelivery-delay>5000</redelivery-delay>
    </address-setting>
</address-settings>
```

上例中对 exampleQueue 地址发送时配置了 5 秒的延迟时间。下表是对参数的配置说明。

redelivery-delay
该参数定义了延迟发送的时间，默认值是0，即没有延时。

6.4.1.2 死信地址

通过定义一个死信地址也可以防止同一个消息被无休止地传递： 当一个消息被重复传递一定次数后，就会从队列中删除并传递到定义好的死信地址中。

这些死信中的消息之后可以转发到某个队列中，以便于系统管理员分析处理。

每个 MQ 的地址可以有一个死信地址。当一个消息被反复传递达一定次数时，它就会被从队列中删除并送到死信地址。这些死信消息可以被接收进行分析处理。

死信地址定义也是在地址设定中（address-setting）。

```
<address-settings>
    <address-setting match="jms.queue.exampleQueue">
       <dead-letter-address>jms.queue.deadLetterQueue</dead-letter-address>
       <max-delivery-attempts>3</max-delivery-attempts>
    </address-setting>
</address-settings>
```

上例中对 exampleQueue 地址配置了死信地址和重试次数。下面是对参数的配置说明。

dead-letter-address
该参数定义了死信地址，如果没有配置，则重传指定次数后删除消息。

```
max-delivery-attempts
    该参数定义了重传次数，默认为10，如果配置为-1则会反复重传。
```

6.4.2 过期消息配置

消息在发送时有一个可选的生存时间属性。

如果一个消息已经超过了它的生存时间，MQ 不再将它传递给任何接收者。服务器会将过期的消息抛弃。

MQ 的地址可以配置一个过期地址，当消息过期时，它们被从队列中删除并被转移到过期地址中。多个不同的队列可以绑定到一个过期地址上。这些过期的消息过后可以接收下来以供分析用。

6.4.2.1 配置过期地址

过期地址定义是在地址设定中（address-setting）。

```
<address-settings>
    <address-setting match="jms.queue.exampleQueue">
       <expiry-address>jms.queue.expiryQueue</expiry-address>
    </address-setting>
</address-settings>
```

如果没有定义过期地址，当一个消息过期时，它将被删除。配置过期地址时可以使用通配符 来给一组地址配置过期地址。

6.4.2.2 配置过期回收线程

为了发现是否有消息过期，RCloudMQ 有一个回收线程定期地检查队列中的消息。在主配置文件中可以对回收线程进行配置。

```
message-expiry-scan-period
    该参数定义过期消息的扫描间隔（单位毫秒，默认为30000ms）。如果设为-1，则关闭扫描。

message-expiry-thread-priority
    该参数定义了回收线程的优先级（为0到9的整数，9优先级最高。默认是3）。
```

6.4.3 超大消息配置

RCloudMQ 支持超大消息的发送和接收。消息的大小不受客户端或服务器端的内存限制。它只受限于其磁盘空间的大小。

大消息的发送和接收是在客户端中用 stream 来实现，这里只介绍在服务器端的配置。

`large-message-directory`

该参数定义大消息的存放路径。

`min-large-message-size`

该参数定义了大消息的最小字节，任何消息的大小如果超过了该值就被视为大消息。一旦成为大消息，它将被分成小的片段来传送默认值是`100KB`。

6.4.4 分页转存配置

RcloudMQ 提供分页转存功能，可以在有限的内存下支持包含百万消息的超大规模的队列。

当有限的内存无法装得下如此多的消息时，MQ 将它们分页转存到磁盘中，在内存有空闲时再将消息分页装载到内存。通过这样的处理，不需要服务器有很大的内存就可以支持大容量的队列。

通过配置可以给一个地址设置一个最大消息值。当这个地址消息数在内存中超过了这个值时，MQ 开始将多余的消息转存到磁盘中。

6.4.4.1 转存路径配置

分页转存路径是在主配置文件中作为全局参数进行配置的，MQ 会为每个地址创建一个文件夹。

```
<paging-directory>/somewhere/paging-directory</paging-directory>
```

6.4.4.2 转存参数配置

分页转存参数是在主配置文件中针对每个地址进行配置的。

```
<address-settings>
     <address-setting match="jms.someaddress">
        <max-size-bytes>104857600</max-size-bytes>
        <page-size-bytes>10485760</page-size-bytes>
        <address-full-policy>PAGE</address-full-policy>
     </address-setting>
  </address-settings>
```

上面是一个分页转存配置示例，下表来详细说明每个参数。

max-size-bytes

该参数定义地址的最大内存值。当消息占用内存超过此值时，进入分页转存模式。 默认为-1 （关闭分页转存功能）。

page-size-bytes

该参数定义了个分页文件的大小。 默认为10MB (10 * 1024 * 1024 字节)。

address-full-policy

该参数定义消息超出内存的处理方式，如果要分页缓存功能则必须设置为PAGE。

PAGE表示多余的消息会被保存到磁盘。

如果设为DROP，那么多余的消息将会被丢弃。

如果设为BLOCK，当消息占满设定的最大 内存时，在客户端消息的发送者将被阻塞，不能向服务器发送更多的消息。

系统默认为PAGE。

6.4.4.1 注意事项

消息启用分页缓存后，需要有以下几方面会和普通消息的处理有很大差别，需要特别注意调整参数。

- **主题模式下的分页缓存**

当使用主题模式时，有多个队列绑定到一个地址。当该地址启用了分页缓

冲功能，如果有一个队列的消费速度很慢，则其他的队列会一直处于等待状态，只有等到最后一个队列传递了这些消息后，那些转存的消息被加载到内存，其它队列才有机会得到更多的消息。

- **主题模式下的分页缓存**

当使用主题模式时，有多个队列绑定到一个地址。当该地址启用了分页缓冲功能，如果有一个队列的消费速度很慢，则其他的队列会一直处于等待状态，只有等到最后一个队列传递了这些消息后，那些转存的消息被加载到内存，其它队列才有机会得到更多的消息。

- **分页转存与消息选择器**

当使用对地址使用消息选择器时，如果该地址的消息正在分页缓存，则消息的过滤只对内存中的消息起作用。

- **分页转存与未通知的消息**

如果消息没有被通知，它会一直留在服务器的内存中，占用着内存资源。只要消息在被接收者收到并通知后，它才会在服务器端被清除，空出内存空间以便转存在磁盘上的消息被装载到内存进行传递。如果没有通知，消息不会被清除，也就不会空出内存空间，转存到磁盘上的消息也就无法装载到内存进行传递。于是在接收端就会呈现出死机的现象。如果消息的通知是依靠 ack-batch-size 的设定进行的批量通知，那么一定要注意不要将分页转存的消息临界值设得小于 ack-batch-size，否则你的系统可能会发生死机现象!

6.4.5 消息分组配置

RcloudMQ 提供消息分组功能，可以使得具有同一 groupId 的消息只被同一个消费者接受。消息组在需要同一个接收者按顺序处理某类消息的时候很有用。

消息分组可以在客户端程序中动态指定，也可以在连接工厂中配置。

```
<connection-factory name="ConnectionFactory">
      <connectors>
         <connector-ref connector-name="netty-connector"/>
      </connectors>
      <entries>
```

```
        <entry name="ConnectionFactory"/>
      </entries>
      <autogroup>true</autogroup>
      <group-id>Group-0</group-id>
  </connection-factory>
```

在使用集群模式时，消息组的配置比较特别，详细内容在集群配置的章节里描述。

6.4.6 消息通知

JMS 规定了三种消息通知方式：

AUTO_ACKNOWLEDGE 自动应答

CLIENT_ACKNOWLEDGE 客户应答

DUPS_OK_ACKNOWLEDGE 允许副本应答

另外 RcloudMQ 还支持一种 JMS 不支持的模式，应用程序在出现故障时可以容忍消息丢失，这样可以在消息在传递给客户端之前就通知服务器。称为 pre-acknowledge。

这种模式的缺点是消息在通知后，如果系统出现故障时，消息可能丢失。并且在系统重启后该消息不能恢复。

使用 pre-acknowledgement 模式可以节省网络传输和 CPU 处理资源。

注意如果你使用 pre-acknowledge 模式，在接收端不支持事务。因为这个模式不是在提交时通知消息，是在消息在传递之前就通知了。

配置方式是在 mq-jms.xml 定义连接器时指定。

```
  <connection-factory name="ConnectionFactory">
      <connectors>
        <connector-ref connector-name="netty-connector"/>
      </connectors>
      <entries>
        <entry name="ConnectionFactory"/>
      </entries>
      <pre-acknowledge>true</pre-acknowledge>
```

```
</connection-factory>
```

6.4.7 消息转发分流

RcloudMQ 提供一个转发器，可以将路由到一个地址的消息透明地转发到其它的地址去，不需要客户端的参与。

转发器可以是唯一（exclusive）的，即消息只转发到新的地址，不发到原来的地址；也可以是不唯一（non-exclusive）的，即消息在发往原有地址的同时，它的一个拷贝被发往新的地址。不唯一的转发器可以在应用中将消息进行分流（splitting）。比如在一个订单系统中它可以用于监视发往订单队列中的每个订单消息。

转发器还可以带一个可选的消息选择器。只有被选择器选择的消息才会被转发。

转发器只在同一个服务器中的地址间进行转发。如果要向另外服务器中的地址进行转发，可以采用转发器与桥配合来实现。先将消息通过转发器转发到一个存储与转发的队列中，再由一个桥将这个队列的消息转发到远程服务器的目的 地址中。

由转发器与桥进行配合可以组成复杂的路由系统。在服务器中由一组转发器可以形成一个消息路由表。如果加上桥，就 可以进一步形成一个分布式的可靠的消息路由网。

转发器的配置在主配置文件中定义。可以配置零个或多个转发器。

6.4.7.1 唯一式转发器

下面是一个唯一式转发器的例子。它将所有符合条件的消息转发到新的地址，而旧的地址将不能得到这些消息。

```
<divert name="prices-divert">
   <address>jms.topic.priceUpdates</address>
   <forwarding-address>jms.queue.priceForwarding</forwarding-address>
   <filter string="office='New York'"/>
```

```
      <transformer-class-name>
         org.RCloudMQ.jms.example.AddForwardingTimeTransformer
      </transformer-class-name>
      <exclusive>true</exclusive>
   </divert>
```

在这里我们定义了一相名为“prices-divert”的转发器，它将发往“jms.topic.priceUpdates”（对应 JMS 话题 priceUpdates）的消息转向另一个本地地址“jms.queue.priceForwarding”（对应 JMS 队列 priceForwarding）。

我们还配置了一相消息过滤器。只有 office 属性值为 New York 的消息才被转发到新地址，其它消息则继续发往原地址。如果没有定义过滤器，所有消息将被转发。

本例中还配置了一个转换器的类。当每转发一个消息时，该转换器就被执行一次。转换器可以对消息在转发前进行更改。

6.4.7.1 不唯一转发器

不唯一转发器将消息的拷贝转发到新的地址中， 原消息则继续发往原有地址。

因此不唯一转发器可以将消息进行分流（splitting）。

不唯一转发器的配置和唯一转发器的配置基本是一样的，也可以带一个可选的过滤器和转换器。

```
   <divert name="order-divert">
      <address>jms.queue.orders</address>
      <forwarding-address>jms.topic.spyTopic</forwarding-address>
      <exclusive>false</exclusive>
   </divert>
```

6.4.8 重复消息检测

RcloudMQ 具有强大的自动检测重复消息的功能，应用层无需实现复杂的重复检测。

6.4.8.1 消息重复原因分析

当客户端向服务器端发送消息时，或者从一个服务器向另一个服务器传递消息时，如果消息发送后目标服务器或者连接出现故障，导致发送一方没有收到发送成功的确认信息，发送方因此就无法确定消息是否已经成功发送到了目标地址。

如果上述的故障发生在消息被成功接收并处理后，但是在向发送方返回功能确认信息之前，那么消息实际上可以到达其目的地址；如果故障发生在消息的接收及处理过程中，则消息不会到达其目的地址。从发送方的角度看，它无法区分这两种情况。

当服务器恢复后，客户端面临困难的选择。它知道服务器出了故障，但是不知道刚刚发送的消息是否成功到达目的地址。如果它重新发送这个消息，就有可能造成消息的重复。如果这个消息是一个订单的话，重复发送消息就会产生两个相同的订单，这当然不是所希望的結果。

将消息的发送放到一个事务中也不能解决这个问题。如果在事务提交的过程中发生故障，同样不能确定这个事务是否提交成功！

为了解决这个问题，RcloudMQ 提供了自动消息重复检测功能。

6.4.8.2 在消息发送中应用重复检测

在消息发送中启用重复检测功能十分简单：你只需将消息的一个特殊属性设置一个唯一值。你可以用任意方法来 计算这个值，但是要保证它的唯一性。当目标服务器接收到这个消息时，它会检查这个属性是否被设置，如果设置了，就检查内存缓存中是否已经接收到了相同值的消息。如果发现已经接收过具有相同属性值的消息，它将忽略这个消息。

6.4.9 消息拦截

RcloudMQ 提供消息拦截拦截功能。拦截器可以拦截进入服务器的数据包。

每进入服务器一个数据包，拦截器就被调用一次，允许一些特殊和处理，例如对包的审计、过滤等。拦截器可以对数据包进行改动。

拦截器必须要实现 Interceptor 接口。

```
public interface Interceptor
{
   boolean intercept(Packet packet, RemotingConnection connection)
               throws RCloudMQException;
}
```

拦截器的配置在 mq-configuration.xml 主配置文件中：

```
<remoting-interceptors>
   <class-name>org.RCloudMQ.jms.example.LoginInterceptor</class-name>
   <class-name>
   org.RCloudMQ.jms.example.AdditionalPropertyInterceptor
   </class-name>
</remoting-interceptors>
```

也可以在客户端代码中调用连接工厂时加载调用拦截器。

6.4.10 定时消息

RCloudMQ 提供消息定时发送功能，使得消息按照设定的时间提交到服务器。

定时消息的功能是由客户端 API 调用时设定的，具体接口参加开发手册相关内容，服务不需要做额外配置。

6.4.11 最新值消息

RCloudMQ 提供最新值队列功能。最新值队列是一种特殊的队列，当一个新消息到达一个最新值队列时，它会将所有与该消息定义的 Last-Value 相同的旧消息抛弃。换句话说，只有最新的消息被保留下来。

一个典型的用例是股价信息，通常你只关心一支股票的最新价格。

最新值队列的配置在主配置文件的 address-setting 内定义：

```
<address-setting match="jms.queue.lastValueQueue">
    <last-value-queue>true</last-value-queue>
</address-setting>
```

默认的 last-value-queue 值是 false。可以使用通配符来匹配地址。

6.5 流量控制

RCloudMQ 提供有效的流量控制，对客户端与服务器之间，或者服务器之间的数据流量进行限制，目的是防止通迅双方由于大量数据而过载。

6.5.1 消费者（consumer）流量控制

接收端流量控制是指对客户端的接收者接收消息流的控制。通常为了提高效率，在客户端通常将消息放入缓存，然后再将缓存中的消息传递给接收者（consumer）。当接收者处理消息的速度小于服务器向其发送消息的速度时，就可能造成消息在客户端不断积累，最终引起内存溢出的错误。

6.5.1.1 基于窗口的流量控制

默认情况下 RCloudMQ 的接收者一端会将消息进行缓存以提高性能。如果不这样做，那每次接收者收到一个消息，都得通知服务器传递下一个消息，然后服务器再将下一个消息传递过来。这就增加了通信的次数。

对于每一次消息传递都有一个网络的往返通信，这样降低了性能。

为了避免这样，RCloudMQ 将每个接收者的消息提前接收到一处缓存中。每个缓存的最大值由 consumer-window-size 参数决定（单位字节）。

```
consumer-window-size的默认值是 1 MB （1024 * 1024 字节）
它的值可以是：
-1 代表大小无限制的缓存。
0 代表不缓存消息（零接收缓冲）。
>0 代表缓存的最大字节数。
```

合理设置接收者的窗口大小可以显著提高性能。下面是两个极端的例子：

- 快速接收者

所谓快速接收者是指消息的接收者处理消息的速度大于等于它的接收速度。

对于快速接收者或以将 consumer-window-size 设为 -1，使得客户端的消息缓存的大小无限制。

请谨慎使用这一设置值： 如果接收者的消息处理速度比接收速度小，可造成客户端内存溢出。

- 慢接收者

所谓慢接收者是指接收者每处理一个消息就要花很多时间。这样将缓存关闭就比较合理。服务器可以将多余的 消息传递给其它的接收者。

假设一个队列有 2 个接收者。其中一个接收者非常慢。消息被轮流传递到两个接收者。其中的快速接收者 很快将其缓存中的消息处理完毕。同时慢接收者的缓存中还有一些消息等待处理。这样快速接收者在一段时间 内就处于空闲状态。

这时，将 consumer-window-size 设为 0 （没有缓存），就可以将它变成慢接收者。这样在慢接收者一方不会缓存消息，这使得快的接收者可以处理更多的消息，而不至于处于空闲 状态。

这说明将它设置为 0 可以控制一个队列的消息在多个接收者之间的消息分配。

大多数情况下很难判断哪些接收者是快速的，哪些是慢速的。往往很多接收者是处于两者之间。这样对于 consumer-window-size 的值就要视具体情况而定。有时需要进行一定的测试来决定它的最佳值。通常情况下将其设为 1MB 可以满足大多数的应用情况。

6.5.1.2 基于速率流量控制

我们还可以通过控制速率的方法来控制流。这是一种像调节节流阀的形式。这种方法保证一个接收者接收消息的速率不会超过设定的值。

速率必须是一个正整数。它代表最大接收速度，单位是消息每秒。将它设为-1 就会关闭速率流控制。 默认值是-1。

可以通过配置 mq-jms.xml 来进行速率流控制。

```
<connection-factory name="ConnectionFactory">
      <connectors>
         <connector-ref connector-name="netty-connector"/>
      </connectors>
      <entries>
         <entry name="ConnectionFactory"/>
      </entries>
      <!—每秒10条消息-->
      <consumer-max-rate>10</consumer-max-rate>
 </connection-factory>
```

速率流控制可以与窗口流控制结合使用。速率控制只规定了客户端每秒接收多少消息。因此如果你设定了一个较低的速率，同时又设定了一个大的缓存窗口，那么客户端的缓存将会很快饱和。

6.5.2 生产者（producer）流量控制

RCloudMQ 还可以控制客户端向服务器发送消息的速度，以避免服务器因大量数据过载。

6.5.2.1 基于窗口的流量控制

与接收者的相应的控制相似。在默认条件下，发送者要有足够的份额（credits）才可以向服务器的地址发送消息。这个份额就是消息的大小。

当发送者的份额不足时，它要向服务器请求更多的份额以便发送更多的消

息。

发送者一次向服务器请求的份额值被称为窗口大小。

于是窗口大小就是指发送者向服务器不间断发送消息的总的最大字节数。当发送完毕时需再向服务器请求份额。这样就避免了服务器消息过载的情况。

可以通过配置 mq-jms.xml 来进行控制。

```
<connection-factory name="ConnectionFactory">
   <connectors>
      <connector-ref connector-name="netty-connector"/>
   </connectors>
   <entries>
      <entry name="ConnectionFactory"/>
   </entries>
   <producer-window-size>10</producer-window-size>
</connection-factory>
```

6.5.2.2 限定发送者窗口流控制

通常情况下客户端请求多少份额，MQ 服务器就给予多少份额。然而我们还可以针对每个地址来设定一个最大的份额值，以使服务器给出的份额都不大于该值。这样可以防止一个地址的内存溢出。

例如，如果有一个队列称为“myqueue”。将它的最大内存值设为 10MB，则服务器就会控制给出的份额以保证向该队列的地址发送消息时不会占大于 10MB 的内存空间。

当一个地址已经满了的时候，发送者将会阻塞直到该地址有了多余的空间为止，即地址中的消息被接收了一部分后使得 地址腾出了一些空间。

我们将这种控制方法称为限定发送者窗口流控制。这是一种有效的防止服务器内存溢出的手段。

它可以看成是分页转存（paging）的另一种方法。分页转存不阻塞发送者，它将消息转存到存贮介质上以节省内存的空间。

要配置一个地址的最大容量并告诉服务器在地址满了的情况下阻塞发送

者，你需要为该地址定义一个 AddressSettings 并设定 max-size-bytes 和 address-full-policy。

这个配置对所有注册到该地址的队列有效。即所有注册队列的总内存将不超过 max-size-bytes。对于 JMS topic 情况则意谓着该 topic 的所有订阅的内存不能超过 max-size-bytes 的设定值。

```
<address-settings>
   <address-setting match="jms.queue.exampleQueue">
      <max-size-bytes>100000</max-size-bytes>
      <address-full-policy>BLOCK</address-full-policy>
   </address-setting>
</address-settings>
```

上面的例子将 JMS 队列"exampleQueue"的最大内存值设为 100000 字节并且阻塞发送者以防止消息量超过这个值。

注意必须设置 BLOCK 的策略才能打开限定发送者窗口控制。

请注意默认的配置下当一个地址中的消息量达到 10MB 时，其所有的消息发送者将变为阻塞状态，也就是说 在没有接收的情况下你不能向一个地址不阻塞地一次发送超过 10MB 的消息。要想增加这个限制，可以加大 max-size-bytes 参数的值，或者调整地址的消息容量限制。

6.5.2.3 速率流控制

RCloudMQ 也可以控制发送者发送消息的速率。单位是每秒消息数。通过设定速率可保证发送者的发送速率不超过某个值。

速率必须是一个正整数。如果设为 -1 则关闭速率流控制。默认值是-1。

```
<connection-factory name="ConnectionFactory">
      <connectors>
         <connector-ref connector-name="netty-connector"/>
      </connectors>
      <entries>
         <entry name="ConnectionFactory"/>
```

```
        </entries>
        <!—每秒50条消息-->
       <producer-max-rate>50</producer-max-rate>
    </connection-factory>
```

6.6 持久化配置

RCloudMQ 提供了使用文件作为消息持久化的方式来提升系统的性能，针对不同的操作系统，使用了 NIO 和 AIO 的技术。

RcloudMQ 系统有好几种不同功能的日志：绑定日志，JMS 日志和消息日志。

绑定日志用来保存和系统相关的信息；

JMS 日志用来保存和队列工厂等相关的信息；

消息日志用来保存和消息相关的信息。

6.6.1 绑定日志

RcloudMQ 绑定日志的配置在主配置文件中，主要是设定文件的路径信息。

indings-directory

这是绑定日志的位置。默认值是data/bindings。

create-bindings-dir

如果设置为true，那么在 bindings-directory 所设定的位置不存在的情况下会自动创建它。默认值是true。

6.6.2 JMS 日志

RcloudMQ 中 JMS 日志的配置和绑定日志公用一个配置路径。

indings-directory

这是绑定日志的位置。默认值是data/bindings。

create-bindings-dir

如果设置为true，那么在 bindings-directory 所设定的位置不存在的情况下会自动创建它。默认值是true。

6.6.3 消息日志

RcloudMQ 消息日志也是在主配置文件中进行配置，有许多不同的配置选项供不同的应用场景使用，诸多参数对系统的性能也有很大影响。

journal-directory

这是消息日志文件所在的目录。默认值是 data/journal。

为以达到最佳性能，我们建议将日志设定到属于它自己的物理卷中以减少磁头运动。如果日志的位置与 其它进程共用（如数据库，绑定日志或事务的日志等）则磁头的运动显然要增加很多。性能也就没有保证了。

如果消息日志是贮存在SAN中，我们建议每个日志都拥有自己的LUN（逻辑单元）。

create-journal-dir

如果设为true，则当journal-directory所指定的日志目录不存在时，会自动创建它。

默认值是true。

journal-type

有效值是NIO 或者 ASYNCIO。

如果你的平台不是Linux或者你没有安装 libaio，RCloudMQ会自动检测到并使用NIO。

journal-sync-transactional

如果设为true，RCloudMQ会保证在事务的边界操作时（commit，prepare和rollback）将事务数据写到磁盘上。

默认的值是 true。

journal-sync-non-transactional

如果设为true RCloudMQ将保证每次都将非事务性消息数据（发送和通知）保存到磁盘上。

默认值是 true。

journal-file-size

每个日志文件的大于。单位为字节。

默认值是 10485760 bytes (10MB)。

journal-min-files

最少日志文件数。当MQ启动时会创建这一数量的文件。

创建并初始化日志文件是一项费时的操作，通常不希望这些操作在服务运行时执行。预先创建并初始化这些 日志文件将会使MQ在工作时避免浪费不必要的时间。

根据你的应用中队列中消息量的实际要求可以适当调节这一参数。

journal-max-io

写请求被放到一个队列中，然后再被发送到系统中执行。这个参数限制了在任一时间队列中可以存放的最大数量的写请求。如果队列达到这个限制，任何新的写请求都将被阻塞，直到队列中有空位为止。

当使用NIO时，这个参数必须为 1。
当使用AIO时，它的默认值是500。
系统根据不同类型的日志提供不同的默认值。(NIO 为 1, AIO 为 500)。

如果是AIO，这个参数的上限不能超过操作系统的限制，这个值通常为65536.

journal-buffer-timeout

日志模块中有一个内部缓冲。每次写的内容并不是都立即写到磁盘上，而是先放到这个内部缓存中。当这个缓存已满时，或 者超过了一定的时间（timeout），才将缓存的数据存到硬盘上。NIO和AIO都有这一特点。采用缓存的方式可以很好地满足大量并发写数据的需要。

这一参数规定了缓存的失效时间，如果过了这个时间，即使缓存还没有满，也将数据写入磁盘中。AIO的写入能力通常要比NIO强。因此系统对于不同类型的日志有着不同的默认值。
（ NIO的默认值是 3333333 纳秒，即每秒300次。 而AIO则是500000纳秒，即每秒2000次。）

注意：
加在这个参数有可能会增加系统的吞吐量，但可能会降低系统的响应能力。通常情况下默认值应该是比较理想的折中选择。

journal-buffer-size

AIO的定时缓冲的大小，默认值为490KB。

journal-compact-min-files

进行整理压缩日志操作的最少文件数。当日志文件少于这个数时，系统不会进行文件的整理压缩。

默认值是 10。

journal-compact-percentage

开始整理压缩的界限值。

当系统中有效数据的比例少于这个值时系统开始整理压缩日志。注意是否进行压缩还要受到

journal-compact-min-files参数的控制。

这一参数的默认值是 30。

6.6.4 消息非持久化

在一些情况下，消息系统并不需要持久化。这时可以配置 RcloudMQ 不使用持久层。只要将主配置文件中的 persistence-enabled 参数设为 false 即可。

注意如果你将该参数设为 false 来关闭持久化，就意味着所有的绑定数据、消息数据、超大消息数据、重复 ID 缓冲以及转移（paging）数据都将不会被持久。

```
persistence-enabled = false
```

6.7 可靠性配置

RCloudMQ 提供多种有效的机制来保障客户端信息完整可靠的提交到服务器，极大简化了客户端在可靠性方面的投入。

6.7.1 事务保证

RCloudMQ 支持消息事务。在应用中需要提交或回滚事务时，客户端将提交或回滚的请求发送到服务器，客户端阻塞等待服务器的响应。

当服务器端在收到事务提交或事务回滚的请求时，它将事务信息记录到日志（journal）中。然后向客户端发回响应。

RCloudMQ 提供两种方式来处理如何向客户端发回响应，在主配置文件中进行配置。

journal-sync-transactional

事务响应的处理方式。

当是false，服务器向客户端发回响应时事务的处理結果不一定已经被保存到磁盘中。可能会在之后的某个时间保存。如果期间服务器发生故障那么事务的处理信息可能丢失。

当是 true 时，服务器将保证在向客户端发回响应时，事务的处理信息已经被保存到了磁盘中。

默认值是 true。

这个参数设为false可以提高性能，但是要以牺牲事务的持久性为代价。

6.7.2 非事务性消息发送的保证

使用非事务性会话发送消息时，经过适当配置，客户端在发送后以阻塞的方式等待，直到确认发出的消息已经到达服务器后再返回。可以对持久化或非持久化的消息分别配置，在 mq-jms.xml 的连接工厂中进行配置。

block-on-durable-send

非事务持久化时响应的处理方式。

如果设为true则通过非事务性会话发送持久消息时，每次发送都将阻塞直到消息到达服务器并返回通知为止。

默认值是 true。

block-on-non-durable-send

非事务非持久化时响应的处理方式。

如果设为true， 则通过非事务性会话发送非持久化的消息时，每次发送都将阻塞直到消息到达服务器并返回通知为止。

默认值是 false。

将发送设置为阻塞方式会降低程序的效率。因为每次发送都需要一次网络往返的过程，然后才可以进行下次发送。 这样发送消息的速度将受网络往返时间（RTT）的限制。这样你的网络带宽就可能没有被充分利用。为了提高效率，我们建议采用事务来批量发送消息。因为在事务中，只有在提交或回滚时阻塞。另外你还可以利用异步发送通知功能。

6.7.3 非事务性通知的保证

当客户端使用非事务性会话向服务器通知消息收到时，可以配置 RCloudMQ

使得客户端的通知阻塞直到服务器收到了通知并返回为止。

在 mq-jms.xml 的连接工厂中进行配置。

```
block-on-acknowledge
    消息是否以同步方式通知。

    如果该参数设为true则所有的通过非事务会话的消息通知都是阻塞式的。如果你想要的消息传递策略是最多一次的话，那么你需要将此参数设为true。

    默认值是false。
```

6.7.4 异步发送通知

如果你使用的是非事务会话来发送消息，并且希望保证每个发送出去的消息都到达服务器的话，你可以将 MQ 配置成阻塞的方式。这样做的一个缺点是性能的降低。因为这样每发送一个消息就需要一次网络的往返通信。如果网络时延越长，消息发送的效率就越低。同时网络的带宽对消息的发送没有影响。

我们来做一个简单的计算。假设有一个 1GB 的网络，客户端与服务器间往返时间为 0.25ms。

这样，在阻塞方式的情况下，客户端最大的消息发送速度为 1000/ 0.25 = 4000 消息每秒。

如果每个消息的大小< 1500 字节，而且网络的最大传输单元（MTU）是 1500 字节。那么理论上 1GB 的网络 最大的传输速率是 (1024 * 1024 * 1024 / 8) / 1500 = 89478 消息每秒！尽管这不是一个精确的工程计算但 你可以看出阻塞式的发送对性能的影响会有多大。

为了解决这个问题，MQ 提供了一种新的功能，称为异步发送通知。它允许消息以非阻塞的方式发送，同时从另一个连接流中异步地接收服务器的通知。这样就使得消息的发送与通知分开来，避免了阻塞方式带来的缺点。在保证消息可靠发送到服务器的同时提高了吞吐量。

异步通知的实现时在客户端编程时设置回掉来实现的，具体细节参考开发手册，在服务端不需要额外配置，需要注意的是，为了使异步发送通知正常工作你必须确保 confirmation-window-size 的值为一个正整数，例如 10MB。

6.7.5 客户端重连

RcloudMQ 提供了连接失效时的诸多可靠性措施，在客户端在与服务器的连接出现故障时，可以自动地重新建立连接并恢复与服务器的通迅。

6.7.5.1 100%透明的会话恢复

如果网络出现暂时性连接故障(没有超过连接 TTL 失效连接的检测时间)，并且服务器没有重启的情况下，当前的会话还会存在服务器中，其状态如同客户端没有断开。

在这种情况下，当客户端重新连接上服务器后，RcloudMQ 自动将客户端和会话与服务器端的会话重新连接起来。整个过程 对于客户端是完全透明的，在客户端就好像什么都没有发生一样。

具体工作原理如下：

客户端再向服务器发送命令时，它将每个命令保存到内存的一块缓存中。当连接出现故障时客户端会尝试与该服务器恢复会话。做为恢复协议的一部分，服务器在会话恢复时通知客户端最后一个成功接收的命令 id。

根据这个命令 id，客户端可以判断它的缓存中是否有命令还未被服务器成功接收。如果有，客户端可以重新发送这些命令。

当服务器成功接收了 ConfirmationWindowSize 字节的命令时，会向客户端发送一个命令确认，以使客户端及时清除缓存。

在 mq-jms.xml 的连接工厂中进行配置。

```
confirmation-window-size
   会话恢复的确认窗口大小。

   参数的单位是字节。
   如果该参数是值设为-1，则关闭缓存，即关闭了重新恢复功能，迫使进行重新连接。

   默认值是-1（表示没有自动恢复）。
```

6.7.5.2 会话重新连接

有时服务器发生故障后进行了重启。这时服务器将丢失所有当前的会话，上面所述的会话恢复就不能做到完全透明了。

在这种情况下，MQ 自动地重新建立连接并重新创建会话和接收者。

下表是客户端用于重新连接的参数：

retry-interval

两次重新连接尝试间隔的时间。

单位是毫秒。

默认值是2000毫秒。

retry-interval-multiplier

下一次重试时间间隔的系数。

即下一次重试的时间间隔是本次时间间隔乘以该参数。

假设retry-interval为1000 ms，并且我们 将retry-interval-multiplier设为2.0，如果 第一次尝试失败，则等待1000毫秒后进行第二次重试，如果再失败，则每三次重 试要在2000毫秒后进行，第四次要等待4000毫秒，

默认值是1。表示每次重试间隔相同的时间。

max-retry-interval

重试间的最大时间间隔。

使用retry-interval-multiplier可以使重试的时间间隔以指数级增加。 有可能造成时间间隔增加到一个非常大的数值。通过设置一个最大值可对其增长进行限制。

默认值是2000毫秒。

reconnect-attempts

它表示要进行多少重试后才放弃并退出。

-1表示进行无限次重试。

默认值是0。

下面是一个综合配置示例。

```
<connection-factory name="ConnectionFactory">
```

```
    <connectors>
       <connector-ref connector-name="netty"/>
    </connectors>
    <entries>
       <entry name="ConnectionFactory"/>
       <entry name="XAConnectionFactory"/>
    </entries>
    <retry-interval>1000</retry-interval>
    <retry-interval-multiplier>1.5</retry-interval-multiplier>
    <max-retry-interval>60000</max-retry-interval>
    <reconnect-attempts>1000</reconnect-attempts>
</connection-factory>
```

6.7.6 失效客户端

RcloudMQ 提供了对出现故障的客户端或者异常退出的客户端（即客户端在退出时没有合理的关闭相关资源）进行失效连接管理的诸多资源回收措施。

RcloudMQ 的资源回收是完全可配置的。每个 ClientSessionFactory 有一个连接 TTL 的参数。这个参数的意义是当客户端的一个连接没有任何数据到达服务器时，服务器充许这个连接有效的最长时间。客户端通过定时向服务器端发送“ping“数据包来维持连接的有效，以免被服务器关掉。如果服务器在 TTL 指定的时间内没有收到任何数据包，则认为该连接无效，继而关闭与该连接相关的所有的会话（session）。

在主配置文件中配置 TTL 参数。

```
connection-ttl
    连接的存活时间。

    设为-1表示服务器永远不检测超时的连接。

    默认的连接TTL值是60000毫秒，即一分钟。
```

6.7.7 客户端的故障检测

前面讲述了客户端如何向服务器发送 ping 以及服务端如何清理失效的连接。发送 ping 还有另外一个目的，就是能让客户端检测网络或服务器是否出现

故障。

从客户端的角度来看，只要客户端能从一个连接不断接收服务器的数据，那么这个连接就被认为是一个有效的连接。

如果在属性 client-failure-check-period 所定义的时间内（单位毫秒）客户端没有收到任何数据，客户端就认为这们连接发生了故障。

该参数在 mq-jms.xml 的连接工厂中进行配置。

```
client-failure-check-period
    客户端检查连接失效的间隔时间。

    这一参数通常要比服务器端的连接TTL小许多，以使 客户端在出现短暂连接故障的情况下可以与服务器成功地重新连接。

    -1表示客户端不检查连接的有效性。即不论是否有数据来自服务器，连接都一直有效。

    默认值是30000毫秒，即半分钟。
```

6.8 安全性配置

RCloudMQ 提供了多种安全管理模式。一方面可以使用缺省的基于角色的权限管理，也可以实现接口使用自己的权限管理，还可以使用 JAAS 来进行安全管理。

6.8.1 基于角色的地址安全

RCloudMQ 采用了基于角色的安全模型来配置地址的安全以及其队列的安全。

队列的权限有 7 种，它们是：

- createDurableQueue

  允许用户在相应的地址上创建持久的队列。

- deleteDurableQueue

  允许用户在相应的地址上删除相应的持久的队列。

- createNonDurableQueue

允许用户在相应地址上创建非持久的队列。

- deleteNonDurableQueue。

  允许用户在相应地址上删除非持久队列。

- send

  允许用户向相应地址发送消息。

- consume

  允许用户从相应地址上的队列接收消息。

- manage

  允许用户调用管理操作，即向管理地址发管理消息。

下面是主配置文件中关于安全配置的一个示例。

```
<security-setting match="globalqueues.europe.#">
    <permission type="createDurableQueue" roles="admin"/>
    <permission type="deleteDurableQueue" roles="admin"/>
    <permission type="createNonDurableQueue" roles="admin, guest, europe-users"/>
    <permission type="deleteNonDurableQueue" roles="admin, guest, europe-users"/>
    <permission type="send" roles="admin, europe-users"/>
    <permission type="consume" roles="admin, europe-users"/>
</security-setting>
```

地址的配置使用通配符模式，只有具有 admin 角色的用户才可以创建和删除绑定到以"globalqueues.europe."开始的地址的持久化队列。

具有 admin、guest 或 europe-users 角色的用户可以在以开头的地址上创建临时的队列。

任何具有 admin 或 europe-users 角色的用户可以向以"globalqueues.europe."开头的地址 发送消息，并从绑定到相同地址上的队列接收消息。

在每个 xml 文件中可以有零个或多个 security-setting。

6.8.2 基于套接字的 SSL 安全传输

RcloudMQ 提供消息传输时采用 SSL 来进行安全保护，参见 5.1 章节的配置

说明。

6.8.3 用户信息管理

RcloudMQ 提供了一个简单的用户管理模型。在 mq-user.xml 中配置系统的用户信息。

```
<configuration xmlns="urn:RCloudMQ"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:RCloudMQ ../schemas/RCloudMQ-users.xsd ">
    <defaultuser name="guest" password="guest">
        <role name="guest"/>
    </defaultuser>
    <user name="tim" password="marmite">
        <role name="admin"/>
    </user>
    <user name="andy" password="doner_kebab">
        <role name="admin"/>
        <role name="guest"/>
    </user>
    <user name="jeff" password="camembert">
        <role name="europe-users"/>
        <role name="guest"/>
    </user>
</configuration>
```

系统默认提供一个缺省用户，当客户端创建会话时如果没有提供用户名 / 密码时，就会使用这个用户。根据上述配置，这个默认用户是 guest 并且他的角色是 guest。一个默认用户可以有多个角色。

6.8.4 自定义用户管理模型

RcloudMQ 提供了一个 SecurityManager 接口，可以让用户自己实现安全管理。具体使用参加开发手册相关介绍。

6.8.5 JAAS 安全管理器

JAAS 表示“Java 认证与授权服务”。它是 Java 平台标准的一部分。它提供了进行安全认证与授权的通用接口。它允许你插入自己的安全管理模块。

RcloudMQ 提供了对 JAAS 的支持。

```
<bean name="MySecurityManager"
      class="org.jboss.security.MySecurityManager">
 <start ignored="true" />
 <stop ignored="true" />
 <property name="ConfigurationName">
     org.RCloudMQ.jms.example.ExampleLoginModule
 </property>
 <property name="Configuration">
     <inject bean="ExampleConfiguration" />
 </property>
 <property name="CallbackHandler">
     <inject bean="ExampleCallbackHandler" />
 </property>
</bean>
```

注意你需要为 JAAS 安全管理器提供三个参数：

ConfigurationName: LoginModule 的名字。

Configuration: Configuration 的实现。

CallbackHandler: CallbackHandler 实现，用于用户交互。

6.9 线程池配置

RCloudMQ 提供对线程的有效管理，通过配置不同的选项来适应不同的应用场景。

6.9.1 服务器端线程的管理

每个 MQ 服务器都有一个线程池为一般线程使用，另外还有一个可计划线程池。Java 的可计划线程池不能作为标准的线程池使用，因此我们采用了两个

单独的线程池。

当使用旧的 Java（阻塞）IO 时，使用了一个单独的线程池来处理连接。但是旧的 Java IO 要求一个线程配一个连接，所以如果你的应用有很多并发的连接，这个线程池会很快用光所有的线程，造成服务器出现“挂起”现象。因此，对于大量并发连接的应用，一定要使用 NIO。

如果使用 NIO，默认情况下会使用系统中处理器内核（或超线程）数量三倍的线程来处理接收的数据包。

6.9.1.1 服务器端调度线程池

服务器端可计划线程池可以定期地或延迟地执行所交给的任务，它用来完成 RCloudMQ 中绝大部分这样的任务。

最大线程数可以在主配置文件中进行配置，通常这个线程池不需要很大数量的线程。

```
scheduled-thread-pool-max-size
    调度线程池的最大线程数。

    默认值是5。
```

6.9.1.2 服务器端通用线程池

服务器端绝大部分的异步操作都是由这个线程池来完成的。在它的内部使用了一个 java.util.concurrent.ThreadPoolExecutor 的实例。

最大线程数可以在主配置文件中进行配置。

```
thread-pool-max-size
    线程池的最大线程数。

    如果将参数设为-1则表示该线程池没有线程限制。也就是说当线程不够用时，线程池就会创建新的线程。当任务不多时，空闲的线程将会超时并被关闭。
    如果这个参数的值是一个大于零的整数n，则该线程池的线程数是有限的。当所有线程都 处于忙的状态并且线程数已经达到n时，任何新的请求都将被阻塞直到有线程空闲为止。

    默认值是30。
```

在设置线程上限时，我们建议要非常谨慎。因为如何线程数量过低会造成死锁情况的发生。

6.9.2 客户端线程的管理

在客户端 RCloudMQ 有一个静态的可计划线程池和一个静态的通用线程池，它们在一个 JVM 中由同一个 classloader 装载的所有客户端共同使用。

静态的可计划的线程池的最大线程数为 5，通用线程池则没有线程数限制。

配置方法是在 mq-jms.xml 的连接工厂中进行配置。

```
<connection-factory name="ConnectionFactory">
    <connectors>
       <connector-ref connector-name="netty"/>
    </connectors>
    <entries>
        <entry name="ConnectionFactory"/>
        <entry name="XAConnectionFactory"/>
    </entries>
    <use-global-pools>false</use-global-pools>
    <scheduled-thread-pool-max-size>5</scheduled-thread-pool-max-size>
    <thread-pool-max-size>-1</thread-pool-max-size>
</connection-factory>
```

第七章 集群配置指南

在高并发大数据的应用下，集群是一个服务器所必须具备的特性。Rcloud MQ 提供了服务集群功能，自动实现了信息负载的均衡机制。

7.1 集群概述

RCloudMQ 集群是由一组 RCloudMQ 服务器组成的集合，它们协同合作进行消息处理。集群中每个主节点就是一个 RCloudMQ 服务器，它管理自己的连接并处理自己的消息。要将一个 RCloudMQ 服务器配置为集群服务器，需要将主配置文件中 clustered 的值设为 true。默认值是 false。

要组成一个集群，每个节点都要在其核心配置文件 mq-configuration.xml 中声明集群连接，用来建立与集群中其它节点的通迅。每两个节点间都是通过内部的一个核心桥连接的。这些连接的建立是透明的，集群连接的作用是在集群的各个节点间进行负载平衡。

RCloudMQ 可以采用不同的拓扑结构来组成集群。

RcloudMQ 实现了客户端的负载均衡机制，使得客户端如何均衡其与集群各节点的连接，在节点间来合理的分配消息以避免消息匮乏（starvation）。

RcloudMQ 也提供了服务器发现功能，即服务器通过广播的方式将自己的连接信息告诉客户端或其它服务器，以使它们能与其建立连接，不需要额外的配置。

7.2 服务器发现

服务器发现是指服务器通过广播的方式将自己的连接设置发送到网络上的机制，它有两个目的：

◆ 被消息客户端发现。

客户端接到广播后可以知道集群中有哪些服务器处于工作状态以及如何与

它们建立连接。虽然客户端可以在初始化时接受一个集群服务器的列表，但是这样做与广播方式相比不够灵活。比如集群中有服务器离开或新加入时，列表的方式不能及时更新这些信息。

◆ 被其它服务器发现。

通过广播，服务器之间可以自动建立彼此间的连接，不需要事先知道集群中其它服务器的信息。

服务器发现使用 UDP 协议来广播连接设置。如果网络中 UDP 被关闭，则不能使用服务器发现功能。只有静态地用显式地指定服务器的方法来设置集群或集群的客户端。

7.2.1 广播组

服务器以广播组的方式来广播它的连接器信息。连接器定义了如何与该服务器建立连接的信息。

广播组包括了一系列的连接器对。每个连接器对由主服务器的连接器和备份（可选）服务器连接器信息组成。 广播组还定义了所使用的 UDP 的在址和端口信息。

广播组的配置在服务器主配置文件 mq-configuration.xml 中。一个 RcloudMQ 服务器可以有多个广播组。所有的广播组必须定义在 broadcast-groups 内。

下面是一个广播组的配置例子：

```
<broadcast-groups>
   <broadcast-group name="my-broadcast-group">
      <local-bind-address>172.16.9.3</local-bind-address>
      <local-bind-port>54321</local-bind-port>
      <group-address>231.7.7.7</group-address>
      <group-port>9876</group-port>
      <broadcast-period>2000</broadcast-period>
      <connector-ref connector-name="netty-connector"
        backup-connector="backup-connector"/>
   </broadcast-group>
</broadcast-groups>
```

有些广播组的参数是可选的，通常情况下可以使用默认值。

name 每个广播组需要有一个唯一的名字。
local-bind-address 套接字的本地绑定地址。 如果在服务器中有多个网络接口卡时，必须要指定使用的是哪个接口。如果这个参数没有指定，那么将使用系统内核所选定的IP地址。
local-bind-port 套接字的本地绑定端口。 通常情况下 可以使用其默认值-1，表示使用随机的端口。 这个参数总是和 local-bind-address一起定义。
group-address 广播地址。 它是一个D类的IP地址， 取值范围是224.0.0.0到239.255.255.255。 地址224.0.0.0是保留地址，所以不能使用。 这个参数是必需指定。
group-port 广播的UDP端口。 是一个必需指定的参数。
broadcast-period 指定两次广播之间的时间间隔，单位毫秒。 这是一个可选参数，它的默认值是1000毫秒。
connector-ref 要广播的连接器以及可选的备份连接器。 connector-name属性的值是连接器的名字， backup-connector属性是备份连接器的名字，是可选属性。

7.2.2 发现组

广播组规定了如何广播连接器的信息，发现组则定义的如何接收连接器的信息。

一个发现组包括了一系列的连接器对——每个连接器对代表一个不同的服务器广播的连接器信息。每当接收一次广播，这个连接对的列表就被更新一次。

如果在一定时间内没有收到某个服务器的广播，则其相应的连接器对将从列表中删除。

发现组有两个作用：

- 在创建集群连接时用来判断集群中哪些服务器是可以连接的。
- 客户端用来发现哪些服务器可以连接。

7.2.2.1 在服务器端配置发现组

服务器端的发现组定义在主配置文件 mq-configuration.xml 中。所有的发现组都必须在 discovery-groups 内定义。发现组可以定义多个，下面是一个发现组的定义示例：

```
<discovery-groups>
   <discovery-group name="my-discovery-group">
      <local-bind-address>172.16.9.7</local-bind-address>
      <group-address>231.7.7.7</group-address>
      <group-port>9876</group-port>
      <refresh-timeout>10000</refresh-timeout>
   </discovery-group>
</discovery-groups>
```

下表是对每个参数的解释：

name 每个发现组需要有一个唯一的名字。
local-bind-address 套接字的本地绑定地址。

如果你的主机有多个网络接口，你可能希望发现组只监听一个指定的 网络接口。这个参数就可以用于这个目的。它是一个可选参数。
group-address 广播地址。 它需要与广播组的 group-address一致才可以收到广播组的信息。 这个参数是必需指定。
group-port 广播的UDP端口。 它需要与广播组的 group-port值相同才可以收到广播组的信息。 这是一个必要参数。
refresh-timeout 这个参数决定了在收到某个服务器的广播后，需要等待 多长时间下一次广播必须收到，否则将该服务器的连接器对从列表中删除。 通常这个参数的值应该远大于广播组的broadcast-period，否则会使服务器信息由于小的时间差异而丢失。 这个参数是可选的，它的默认值是10000毫秒（10秒）。

7.2.2.2 在客户端配置发现组

客户端来发现可以连接的服务器列表即可以用 API 来实现，也可以在工厂配置文件中来进行配置。

下面是一个配置示例：

```
<connection-factory name="ConnectionFactory">
   <discovery-group-ref discovery-group-name="my-discovery-group"/>
    <entries>
       <entry name="ConnectionFactory"/>
    </entries>
</connection-factory>
```

其中 discovery-group-ref 的值是定义在 mq-configuration.xml 文件中的一个发现组。

当连接工厂从 JNDI 下载到客户端时，使用它创建连接就会在列表中的服务器间进行负载均衡。客户端通过监听发现组中的广播地址可以不断更新这个服务器列表。

7.3 服务端的消息负载均衡

如果集群和各节点间定义了集群连接，RCloudMQ 可以对到达一个节点的消息进行负载均衡。

举一个简单的例子。一个集群有 4 个节点，分别称为节点 A、B、C 和节点 D。它们组成了一个对称式集群，在每个节点上部署了一个名为 OrderQueue 的队列。

一个客户端 Ca 连接到节点 A 并向其发送订单消息。客户端 Pa、Pb、Pc 和 Pd 分别连接到节点 A、B、C 和 D 并接收处理这些订单消息。如果在节点 A 中没有定义集群连接，那么订单消息都发送到节点 A 中的队列 OrderQueue 中。因此只有连接到节点 A 的客户端 Pa 才能接收到订单消息。

如果在节点 A 定义了集群连接的话，发送到节点 A 的消息被轮流（round-robin）从节点 A 分配到各个节点上的 OrderQueue 队列中。这种消息分配完全在服务器端完成，客户端只向节点 A 发送消息。

例如到达节点 A 的消息可能以下列顺序进行分配：B、D、C、A、B、D、C、A、B、D。具体的顺序取决于节点启动的先后，但是其算法是不变的（即 round-robin）。

RCloudMQ 集群连接在进行消息负载均衡时，可以配置成统一负载均衡模式，即不管各个节点上有无合适的接收者，一律在所有节点间进行消息的分配。也可以配置成为智能负载均衡模式，即只将消息分配到有合适接收者的节点上。

7.3.1 配置集群连接

集群连接将一组服务器连接成为一个集群，消息可以在集群的节点之间进行负载均衡。集群连接的配置在主配置文件中的 cluster-connection 内。一个 MQ

服务器可以有零个或多个集群连接。 下面是一个典型的配置例子：

```
<cluster-connections>
    <cluster-connection name="my-cluster">
        <address>jms</address>
        <retry-interval>500</retry-interval>
        <use-duplicate-detection>true</use-duplicate-detection>
        <forward-when-no-consumers>false</forward-when-no-consumers>
        <max-hops>1</max-hops>
        <discovery-group-ref discovery-group-name="my-discovery-group"/>
    </cluster-connection>
</cluster-connections>
```

下表说明每个参数的配置方法。

address

每个集群连接只服务于发送到以这个参数的值为开始的地址的消息。

这个地址可以为任何值，而且可以配置多个集群连接，每个连接的地址值可以不同。这样MQ可以同时对不同地址同时进行消息的负载均衡。有的地址甚至可能在其它集群的节点中。这也就意谓着 一个MQ服务器可以同时参与到多个集群中。

要注意别造成多个集群连接的地址互相重复。比如，地址“europe“和”europe.news“就互相重复，就会造成同一个消息会被多个集群连接进行分配，这样有可能发生重复传递。

这个参数是必需指定。

retry-interval

集群重连的时间间隔。

一个集群连接实际上在内部是用桥将两 个节点连接起来。如果集群连接已经创建但是目的节点还未启动，或正在重启，这时集群连接就会不断重试与这个节点的连接，直到节点启动完毕连接成功为止。

单位是毫秒。它与桥的参数retry-interval 的含义相同。

这个参数是可选的，默认值是500毫秒。

use-duplicate-detection

集群是否启用重复检测。

集群连接使用桥来连接各节点，而桥可以通过配置向每个转发的消息添加一个重复id的属性。如果目的节点崩溃并重启，消息可以被重新发送。 重复检测的功能就是在这种情况下将重复发送的消息进行过

滤并丢弃。

这参数是可选的，默认值是true。

forward-when-no-consumers

这个参数决定了是否向没有合适接收者的节点分配消息。即不管有没有合适的接收者，消息在所有的节点间轮流分配。

如果这个参数设为true，则消息就会轮流在每个节点间分配，不管是否节点上有没有相应的接收者（或者有接收者但是具有不匹配的选择器）。注意，如果其它节点中没有与本节点同名的队列，MQ不会将消息转发到那些节点中去，不受本参数的限制。

如果参数设为false，MQ中将消息转发到集群中那些有着适合接收者的节点中。如果接收者有选择器，则至少有一个选择器与所转发的消息匹配才可，否则不转发。

本参数是可选的，默认值是false。

max-hops

当一个集群连接在确定进行消息负载均衡的节点组时，这些节点不一定是与本节点直接相连的节点。MQ可以通过其它MQ节点作为中介向那些非直接相连的节点转发消息。

这样可以使MQ组成更加复杂的拓扑结构并且仍可提供消息的负载均衡。

本参数是可选参数，它的默认值是 1，表示消息只向直接相连的节点进行负载均衡。

discovery-group-ref

这个参数决定了使用哪个发现组来获得集群服务器的列表。 集群连接与列表中的服务器建立连接。

7.3.2 集群用户的安全信息

当集群中两个节点建立连接时，RCloudMQ 使用一个集群用户和集群密码。它们定义在主配置文件中：

```
<cluster-user>RCLOUDMQ.CLUSTER.ADMIN.USER</cluster-user>
<cluster-password>CHANGE ME!!</cluster-password>
```

警告：

强烈建议在实际应用中不要使用默认的值，否则任意远程客户端会使用这

些默认值连接到服务器上。

7.4 客户端的消息负载均衡

RCloudMQ 的客户端负载均衡使同一个会话工厂每次创建一个会话时，都连接到集群不同的节点上。这样可以使所的有会话均匀分布在集群的各个节点上，而不会‘拥挤’到某一个节点上。

客户端负载均衡的策略是可配置的。RCloudMQ 提供两种现成的负载均衡策略。你也可以实现自己的策略。

◆ 轮询策略（Round Robin）

这个策略是先随机选择一个节点作为第一个节点，然后依次选择各个节点。

例如一个顺序可能是 B, C, D, A, B, C, D, A, B，另一个也可能是 D, A, B, C, D,A, B, C, D, A 或者 C, D, A, B, C, D, A, B, C, D, A 等等。

◆ 随机策略

每次都是随机选择一个节点来建立会话。

◆ 自定义策略

只需要实现接口 ConnectionLoadBalancingPolicy 即可。

客户端的负载均衡配置很简单，是在连接工厂中进行。

```
<connection-factory name="ConnectionFactory">
    <discovery-group-ref discovery-group-name="my-discovery-group"/>
    <entries>
        <entry name="ConnectionFactory"/>
    </entries>
    <connection-load-balancing-policy-class-name>
    com.chinasoft.mq.loadbalance.RandomConnectionLoadBalancingPolicy
    </connection-load-balancing-policy-class-name>
</connection-factory>
```

连接工厂进行负载均衡的服务器组可以有两种方法来确定：

- 显式指定服务器
- 使用发现组功能

7.5 显式指定集群服务器

有的网络并不开放 UDP，所以就不能使用服务器发现功能来获取服务器列表。

在这种情况下，可以显式地在每个节点或客户端指定服务器的列表。下面介绍如何做：

7.5.1 在客户端指定服务器列表

配置也是在 mq-jms.xml 的连接工厂中进行，下面是个配置示例。

```
<connection-factory name="ConnectionFactory">
   <connectors>
      <connector-ref connector-name="my-connector1"
           backup-connector-name="my-backup-connector1"/>
      <connector-ref connector-name="my-connector2"
           backup-connector-name="my-backup-connector2"/>
      <connector-ref connector-name="my-connector3"
           backup-connector-name="my-backup-connector3"/>
   </connectors>
   <entries>
      <entry name="ConnectionFactory"/>
   </entries>
</connection-factory>
```

其中的 connection-factory 内可以包含零或多个 connector-ref。每个 connector-ref 都拥有 connector-name 属性和一个可选的 backup-connector-name 属性。connector-name 属性指向的是一个在主配置文件中定义的连接器。而 backup-connector-name 属性也是指向在主配置文件中定义的一个连接器。

连接工厂这样就保存有一组[连接器，备份连接器]对，用于客户端在创建连接时的负载均衡。

7.5.2 指定服务器列表以组成集群

下面我们考虑一个对称集群的例子，我们配置了每个集群连接，但是不使用发现功能来获得服务器信息。我们采用配置的方法来显式指定集群的所有成员。

下面就是一个集群连接的配置示例，在主配置文件中进行配置。

```
<cluster-connections>
    <cluster-connection name="my-explicit-cluster">
        <address>jms</address>
        <connector-ref connector-name="my-connector1"
            backup-connector-name="my-backup-connector1"/>
        <connector-ref connector-name="my-connector2"
            backup-connector-name="my-backup-connector2"/>
        <connector-ref connector-name="my-connector3"
            backup-connector-name="my-backup-connector3"/>
    </cluster-connection>
</cluster-connections>
```

cluster-connection 中可以包括零或多个 connector-ref，每个 connector-ref 都有一个 connector-name 属性和一个可选的 backup-connector-name 属性。connector-name 属性指向一个在主配置文件 mq-configuration.xml 中定义的一个连接器，它是主连接器。可选的 backup-connector-name 指向的也是在 mq-configuration.xml 文件中定义的一个连接器。

7.6 消息再分配

集群的另一个重要功能是消息的再分配。前面我们知道在服务器端可以对消息大集群节点间进行轮流方式的负载均衡。如果 forward-when-no-consumers 参数为 false，消息将不会转发到那些没有相应接收者的节点中。这样可以有效避免了消息被送到一个不可能被接收的节点上。但仍然有一个问题无法解决：就是如果在消息发到一个节点后，它的接收者被关闭，那么这些消息仍然不能被接收了，造成了一种消息匮乏情形。这种情况下如何处理？

这里就需要消息再分配功能。通过配置，RCloudMQ 可以将没有接收者的队列中的消息再次分配到有接收者的节点上去。

通过配置，消息可以在队列最后一个接收者关闭时立即进行，也可以配置成等待一段时间再进行。默认消息再分配功能是关闭的。

消息再分配功能可以基于地址进行配置，即在地址设置中指定再分配的延时。

下面是主配置文件中消息再分配的配置示例：

```
<address-settings>
   <address-setting match="jms.#">
      <redistribution-delay>0</redistribution-delay>
   </address-setting>
 </address-settings>
```

上面 address-settings 中设置的 redistribution-delay 值为 0。它适用于所有以“jms“开头的 地址。由于所有 JMS 队列与话题订阅都绑定到以”jms“为开头的地址，所以上述配置的立即方式（没有延迟）消息再分配适用于所有的 JMS 队列和话题订阅。

```
redistribution-delay
    定义了队列最后一个接收者关闭后在进行消息再分配前所等待的时间，单位毫秒。

    如果其值是0，表示立即进行消息再分配。
    -1表示不会进行消息再分配。

    默认值是-1。
```

通常为消息分配定义一个延迟是有实际意义的。很多时候当一个接收者被关闭时，很快就会有一个新的接收者被创建。 在这种情况下加一延迟可以使消息继续在本地进行接收，而不会将消息转发到别处。

7.7 集群拓扑结构

RCloudMQ 集群可以有多种拓扑结构。下面说明两个最常见的结构配置。

7.7.1 对称式集群

对称式集群可能是最常见的集群方式了，在一个对称集群中，每一个节点都与集群中其它任一节点相连。换句话说，集群中任意两个节点的连接都只有一跳（hop）。

要组成一个对称式的集群，每个节点在定义集群连接时要将属性 max-hops 设为 1。通常集群连接将使用服务器发现的功能来获得集群中其它服务器的连接信息。当然在 UDP 不可用的时候，也可以通过显式方式为集群连接指定服务器。

在对称集群中，每个服务器都知道集群中其它服务器中的所有队列信息，以及它们的接收者信息。利用这些信息它可以决定如何进行消息的负载均衡及消息再分配。

7.7.2 链式集群

在链式集群中，并不是每个节点都与其它任何节点直接相连，而是由两个节点组成头和尾，其余节点在中间连接成为一个链的结构。

比如有三个节点 A、B 和 C。节点 A 在一个网络中，它有许多消息的发送者向它发送订单消息。由于公司的政策，订单的接收者需要在另一个网络中接收消息，并且这个网络需要经过其它第三个网络才可以访问。这种情况下我们将节点 B 部署到第三个网络中，作为节点 A 与节点 C 的中间节点将两个节点连接起来。当消息到达节点 A 时，被转发到节点 B， 然后又被转发到节点 C 上，这样消息就被 C 上的接收者所接收。节点 A 不需要直接与节点 C 连接，但是所有三个节点仍然组成了一个集群。

要想组成一个这样的集群，节点 A 的集群连接要指向节点 B，节点 B 的集群连接要指向 C。本例我们只组成一个单向的链式集群，即我们只将消息按节点 A->B->C 的方向流动，而不要向 C->B->A 方向流动。

对于这种集群拓扑，我们需要将 max-hops 设为 2. 这个值可以使节点 C 上队列的信息传送到节点 B，再传送到节点 A。因此节点 A 就知道消息到达时即将其转发给节点 B。尽管节点 B 可能没有接收者，可它知道再经过一跳就可以

将消息转到节点 C，那里就有接收者了。

7.8 HA 配置

HA(High Available)，高可用性。是指当系统中有一台甚至多台服务器发生故障时还能继续运转的能力。作为高可用性的一部分，失效备援（Failover）的含意是当客户端当前连接的服务器发故障时，客户端可以将连接转到另一台正常的服务器，从而能够继续工作。

7.8.1 Master-Backup 对

RCloudMQ 可以将两个服务器以 M－B 对的形式连接在一起。目前 RCloudMQ 允许一个主服务器有一个备份服务器，一个备份服务器只有一个主服务器。在正常情况下主服务器工作，备份服务器只有当主服务器发生失效备援时工作。

没有发生失效备援时，主服务器为客户端提供服务，备份服务器处于待机状态。当客户端在失效备援后连接到备份服务器时，备份服务器开始激活并开始工作。

7.8.1.1 高可用性（HA）的模式

RCloudMQ 的高可用性有两种模式：一种模式通过由主服务器日志向备份服务器日志复制数据。另一种模式则是主服务器与备份服务器间存贮共享。

注意：

只有持久消息才可以在失效备援时不丢失。所有非持久消息则会丢失。

7.8.1.1.1 数据复制

在这种模式下，保存在 RCloudMQ 主服务器中日志中的数据被复制到备份

服务器日志中。注意我们并不复制服务器的全部状态，而是只复制日志和其它的持久性质的操作。

复制的操作是异步进行的。数据通过流的方式复制，复制的結果则通过另一个流来返回。通过这样的异步方式 我们可以获得比同步方式更大的吞吐量。

当用户得到确认信息如一个事务已经提交、准备或加滚，或者是一个持久消息被发送时，RCloudMQ 确保这些状态 已经复制到备份服务器上并被持久化。

数据复制这种方式不可避免地影响性能，但是另一方面它不依赖于昂贵的文件共享设备（如 SAN）。它实际上是一种无共享的 HA 方式。

采用数据复制的失效备援比采用共享存储的失效备援要快，这是因为备份服务器在失效备援时不用重新装载日志。

7.8.1.1.1.1 配置过程

- 配置主服务器

首先在主服务器的主配置文件中配置备份服务器。 配置的参数是 backup-connector-ref。这个参数指向一个连接器。这个连接器也在主服务器上配置。它定义了如何与备份服务器建立连接。

下面就是一个配置示例：

```
<backup-connector-ref connector-name="backup-connector"/>

<connectors>
    <!-- 这个连接器用于连接备份服务      -->
    <!-- 备份服务器在主机"192.168.0.11"上，端口"5445" -->
    <connector name="backup-connector">
      <factory-class>
              comchinasoft.mq.core.remoting.impl.netty.NettyConnectorFactory
       </factory-class>
      <param key="host" value="192.168.0.11"/>
      <param key="port" value="5445"/>
    </connector>
</connectors>
```

- 配置备份服务器

其次在备份服务器上，我们设置了备份服务器的标志，并且配置了相应的

接受器以便主服务器能够建立连接。同时我们将 shared-store 参数设为 false。

```
<backup>true</backup>

<shared-store>false<shared-store>

<acceptors>
    <acceptor name="acceptor">
        <factory-class>
                comchinasoft.mq.core.remoting.impl.netty.NettyConnectorFactory
        </factory-class>
        <param key="host" value="192.168.0.11"/>
        <param key="port" value="5445"/>
    </acceptor>
</acceptors>
```

为了使备份服务器正常工作，一定要保证它与主服务器有着同样的桥、预定义的队列、集群连接、 广播组和发现组。最简单的作法是拷贝主服务器的全部配置然后再进行上述的修改。

7.8.1.1.1.2 备份服务器与主服务器间的同步

为了能正常工作，备份服务器与主服务器必须同步。这意谓着备份服务器不能是当前任意一个备份服务器。如果你这样做，主服务器将不能成功启动，在日志中会出现异常。

要想将一个现有的服务器配置成一个备份服务器，你需要将主服务器的 data 文件夹拷贝到并覆盖这个备份服务器的相同文件夹，这样做保证了备份服务器与主服务器的持久化数据完全一致。

当失效备援发生后，备份服务器代替主服务器工作，原来的主服务器失效。这时简单的重启主服务器是不行的。要想将主服务器与备份重新进行同步，就必须先将主服务器和备份服务器同时停止，再将 主服务器的数据拷贝到备份服务器，然后再启动。

7.8.1.1.2 存贮共享

使用存贮共享，主服务器与备份服务器共用相同目录的日志数据，通常是一个共享的文件系统。这包括转存目录，日志目录，大消息及绑定日志。

当发生失效备援时，工作由备份服务器接管。它首先从共享的文件系统中读取主服务器的持久数据，然后才能接受客户端的连接请求。

与数据复制方式不同的是这种方式需要一个共享的文件系统，主服务器与备份服务器都可以访问。典型的高性能的共享系统是存贮区域网络（SAN）系统。我们不建议使用网络附加存贮（NAS），如 NFS，来存贮共享 日志（主要的原因是它们比较慢）。

共享存贮的优点是不需要在主服务器与备份服务器之间进行数据复制，因此对性能不会造成影响。

共享存贮的缺点是它需要一个共享文件系统。同时，当备份服务器激活时它需要首先从共享日志中读取相应的信息，从而占用一定的时间。

如果你需要在一般工作情况下保持高性能，并且拥有一个快速的 SAN 系统，同时能够容忍较慢的失效备援过程（取决于数据量在多少），我们建议你采用存贮共享方式的高可用性。

7.8.1.1.2.1 配置过程

要使用存贮共享模式，在两个服务器的主配置文件进行配置即可。

```
<shared-store>true<shared-store>
```

此外，备份服务器必须显式地进行指定：

```
<backup>true</backup>
```

另外，需要将主服务器和备份服务器的日志文件位置指向同一个共享位置。

7.8.1.1.2.2 备份服务器与主服务器间的同步。

由于主备服务器之间共享存贮，所以它们不需要进行同步。但是它需要主

备服务器同时工作以提供高可用性。如果一量发生失效备援后，就需要在尽可能早的时间内将备份服务器（处于工作状态）停下来，然后再启动主服务器和备份服务器。

7.8.2 失效备援的模式

RCloudMQ 定义了两种客户端的失效备援：

自动客户端失效备援

应用层的客户端失效备援

RCloudMQ 还支持 100％透明的同一个服务器的自动连接恢复（适用于网络的临时性故障）。这与失效备援很相似，只不过连接的是同一个服务器。

在发生失效备援时，如果客户端有非持久或临时队列的接收者时，这些队列会自动在备份服务器上重新创建。对于非持久性的队列，备份服务器事先是没有它们的信息的。

7.8.2.1 自动客户端失效备援

RCloudMQ 的客户端可以配置主／备份服务器的信息，当客户端与主服务器的连接发生故障时，可以自动检测到故障并进行失效备援处理，让客户端连接到备份服务器上。备份服务器可以自动重新创建所有在失效备援之前存在的会话与接收者。客户端不需要进行人工的连接恢复工作，从而节省了客户端的开发工作。

RCloudMQ 的客户端在参数 client-failure-check-period 规定的时间内如果没有收到数据包，则认为连接发生故障。 当客户端认为连接故障时，它就会尝试进行失效备援。

RCloudMQ 有几种方法来为客户端配置主／备服务器对的列表。可以采用显式指定的方法，或者采用更为常用的服务器发现的方法。要使客户端具备自动失效备援，在客户端的配置中必须要指定重试的次数要大于零。

有时你需要在主服务器正常关机的情况下仍然进行失效备援，在

mq-jms.xml（参数为 failover-on-server-shutdown）文件中进行相应的配置。将上述属性设置为 true。 这个属性的默认值是 false。这表示如果主服务器是正常关机，客户端将不会进行失效备援。

说明：

默认正常关机不会导致失效备援。

默认情况下至少创建了一个与主服务器的连接后失效备援才会发生。换句话说，如果客户端每一次创建与 主服务器的连接失败，它会根据参数 reconnection-attempts 的设置进行连接重试，而不是进行失效备援。如果重试次数超过的该参数的值，则连接失败。

在有些情况下，你可能希望在初始连接失败和情况下自动连接到备份服务器，那么你可以直接设置 failover-on-initial-connection 为 true。默认的值是 false。

7.8.2.1.1 关于服务器的复制

RCloudMQ 在主服务器向备份服务器复制时，并不复制服务器的全部状态。所以当一个会话在备份服务器中重新创建后，它并不知道发送过的消息或通知过的消息。在失效备援的过程中发生的消息发送或通知也可能丢失。

理论上如果进行全部状态的复制，我们可以提供 100％的透明的失效备援，不会失去任何的消息或通知。 但是这样做要付出很大的代价：即所有信息都要进行复制（包括队列，会话等等）。也就是要求复制服务器的每个状态信息，主服务器的每一步操作都将向其备份进行复制，并且要在全局内保持顺序的一致。这样做就极难保证高性能和可扩展性，特别是考虑到多线程同时改变主服务器的状态的情况，要进行全状态复制就更加困难。

一些技术可以用来实现全状态复制，如虚拟同步技术 （virtual synchrony）。但是这些技术往往没有很好的可扩展性，并且将所有操作都 进行序列化，由单一线程进行处理，这样明显地将底了并行处理能力。

另外还有其它一些多线程主动复制技术，比如复制锁状态或复制线程调度等。这些技术使用 Java 语言非常难于实现。

因此得出结论，采用大量牺牲性能来换取 100％透明的失效备援是得不偿失的。没有 100％透明的失效备援我们仍然可以轻易地保证一次且只有一次的传递。这是通过在发生故障时采用重复检测结合事务重试来实现的。

7.8.2.1.2 失效备援时阻塞调用的处理

如果当发生失效备援时客户端正面进行一个阻塞调用并等待服务器的返回，新创建的会话不会知道这个调用，因此客户端可能永远也不会得到响应，也就可能一直阻塞在那里。

为了防止这种情况的发生，RCloudMQ 在失效备援时会解除所有的阻塞调用，同时抛出一个 javax.jms.JMSException 异常（如果是 JMS）客户端需要自行处理这个异常，并且进行必要的操作重试。

如果被解除阻塞的调用是 commit()或者 prepare()，那么这个事务会被自动地回滚，并且 RCloudMQ 会抛出一个 javax.jms.TransactionRolledBackException。

7.8.2.1.3 事务的失效备援处理

如果在一个事务性会话中，在当前事务中消息已经发出或通知，则服务器在这时如果发生失效备援，它不能保证发出的消息或通知没有丢失。

因此这个事务就会被标记为只回滚，任何尝试提交的操作都会抛出一个 javax.jms.TransactionRolledBackException 异常。

客户端需要自行处理这些异常，进行必要的回滚处理。注意这里不需要人工将会话进行回滚－此时它已经被回滚了。用户可以通过同一个会话重试该事务操作。

如果是在提交过程中发生了失效备援，服务器将这个阻塞调用解除。这种情况下客户端很难确定在事故发生之前事务是否在主服务器中得到了处理。

为了解决这个问题，客户端可以在事务中使用重复检测，并且在提交的调用被解除后重新尝试事务操作。如果在失效备援前事务确实在主服务器上已经

完成提交，那么当事务进行重试时，重复检测功能可以保证重复发送的消息被丢弃，这样避免了消息的重复。

说明：

通过处理异常和重试，适当处理被解除的阻塞调用并配合重复检测功能，RCloudMQ 可以在故障条件下保证一次并且只有一次的消息传递，没有消息丢失和消息重复。

7.8.2.1.4 非事务会话的失效备援处理

如果会话是非事务性的，那么通过它的消息或通知在故障时可能会丢失。

如果你在非事务会话中要保证一次并且只有一次的消息传递，你需要使用重复检测功能，并适当处理被解除的阻塞调用。

7.8.2.2 应用层的失效备援

在某些情况下你可能不需要自动的客户端失效备援，希望自己来处理连接故障，使用自己的重新连接方案等。我们把它称之为应用层失效备援，因为它是发生在应用层的程序中。

为了实现应用层的失效备援，你可以使用监听器（listener）的方式。如果使用的是 JMS，你需要在 JMS 连接上 设置一个 ExceptionListener 类。这个类在连接发生故障时由 RCloudMQ 调用。在这个类中你可以将旧的连接关闭，使用 JNDI 查找新的连接工厂并创建新的连接。这里你可以使用 HA-JNDI 来保证新的连接工厂来自于另一个服务器。

第八章 配置优化指南

8.1 持久层的优化

将消息日志放到单独的物理卷上。如果与其它数据共享，例如事务管理、数据库或其它日志等，那么就会增加读写的负担，磁头会在多个不同文件之间频繁地移动，极大地降低性能。我们的日志系统采用的是只添加的模式，目的就是最大程度减少磁头的移动。如果磁盘被共享，那么这一目的将不能达到。另外如果你使用分页转存或大消息功能时，你最好分别将它们放到各自的独立卷中。

尽量减少日志文件的数量。journal-min-files 参数的设置应以满足平均运行需要为准。如果你发现系统中经常有新的日志文件被创建，这说明持久的数据量很大，你需要适当增加这个参数的值，以使 RCloudMQ 更多时候是在重用文件，而不是创建新文件。

日志文件的大小。日志文件的大小最好要与磁盘的一个柱面的容量对齐。默认值是 10MB，它在绝大多数的系统中能够满足需要。

使用 AIO 日志。在 Linux 下，尽量使用 AIO 型的日志。AIO 的可扩展性要好于 Java 的 NIO。

优化 journal-buffer-timeout。如果增加它的值，吞吐量会增加，但是延迟也会增加。

如果使用 AIO，适当增加 journal-max-io 可能会提高性能。如果使用的是 NIO，请不要改变这个参数。

8.2 优化 JMS

- 关闭消息 id

如果你不需要这个 id，用 MessageProducer 的 setDisableMessageID()方法可

以关闭它。这可以减少消息的大小并且省去了创建唯一 ID 的时间。

- 关闭消息的时间戳

如果不需要时间戳，用 MessageProducer 的 setDisableMessageTimeStamp()方法将其关闭。

- 尽量避免使用 ObjectMessage

ObjectMessage 会带来额外的开销。ObjectMessage 使用 Java 的序列化将它序列化为字节流。在对小的对象进行序列化会占用大量的空间，使传输的数据量加大。另外，Java 的序列化与其它定制的技术相比要慢。只有在不得以的情况下才使用它。比如当你在运行时不知道对象的具体类型时，可以用 ObjectMessage。

- 避免使用 AUTO_ACKNOWLEDGE

AUTO_ACKNOWLEDGE 使得每收到一个消息就要向服务器发送一个通知——这样增加的网络传输的负担。如果可能，尽量使用 DUPS_OK_ACKNOWLEDGE 或者 CLIENT_ACKNOWLEDGE。或者使用事务性会话，将通知在提交时批量完成。

- 避免持久化消息

默认情况下 JMS 消息是持久的。如果你不需要持久消息，则将其设定为非持久。 持久消息都会被写到磁盘中，这给系统带来了明显的负担。

- 将多个发送或通知放到一个事务中完成

这样 RCloudMQ 只需要一次网络的往返来发生事务的提交，而不是每次发送 或通知就需要一次网络的往返通迅。

8.3 其它优化

- 使用异步发送通知

如果你在非事务条件下发送持久的消息，并且要保证在 send()返回时持久消息已经到达服 务器，不要使用阻塞式发送的方式，应该使用异步发送通知的方式。

- 使用预先通知模式。

预先通知就是在消息发往客户端之前进行通知。它节省了正常的消息通知所占用的通迅时间。

- 关闭安全

将主配置文件 mq-configuration.xml 中的 security-enabled 参数设为 false 以关闭安全。这可以带来一些性能的提高。

- 关闭持久化

如果不你不需要消息持久化，可以将主配置文件 mq-configuration.xml 中的 persistence-enabled 参数设为 false 来完全关闭持久功能。

- 采用延迟方式事务同步

将主配置文件 mq-configuration.xml 中的 journal-sync-transactional 参数设为 false 可以得到更好的事务持久化的性能。但是这样做可能会造成在发生故障时事务的丢失。

- 采用延迟方式非事务同步

将主配置文件 mq-configuration.xml 中的 journal-sync-non-transactional 参数设为 false 可以得到更好的非事务持久化的性能。但是这样做可能会造成在发生故障时持久消息的丢失。

- 采用非阻塞方式发送消息

将文件 mq-jms.xml 中的参数 block-on-non-durable-send 设为 false 可以使消息发送时不阻塞等待服务器的响应。

如果你的接收者速度很快，你可以增加 consumer-window-size。这样实际上就关闭了流控制的功能。

- 套接字 NIO 与旧的 IO 对比

默认情况下 RCloudMQ 在服务器端使用套接字 NIO 技术，而在客户端则使用旧的（阻塞） IO。NIO 比旧的阻塞式 IO 有更强的可扩展性，但是也会带来一些延时。如果你的服务器要同时有数千个连接，使用 NIO 效果比较好。但是如果 连接数并没有这么多，你可以配置接收器使用旧的 IO 还提高性能。

8.4 传输层的优化

- TCP 缓存大小

如果你的网络速度很快，并且你的主机也很快，你可以通过增加 TCP 的发送和接收缓存来提高性能。

- 增加服务器中文件句柄数量限制

如果你的服务器将要处理很多并行的连接，或者客户端在快速不停地打开和关闭连接，你要确保在服务器端有足够的文件句柄以供使用。

利用参数 batch-delay 并将参数 direct-deliver 设为 false 来提高小消息的处理效率。

8.5 优化虚拟机

- JDK

我们强烈建议使用最新的 Java 虚拟机。它在很多方面对以前 Java 5 的虚拟机进行了改进，特别是在网络功能方面。

- 拉圾回收

为了使服务器的运行比较平滑，建议使用并行拉圾回收的算法。例如在 Sun 的 JDK 使用 JVM 选项-XX:+UseParallelGC.

- 内存设置

尽量为服务器分配更多的内存。使用 JVM 参数-Xms 和-Xmx 来为你的服务器分配内存。建议两个参数的设为相同的值。

- 主动选项（Aggressive options）

不同 JVM 有不同的 JVM 优化参数。对于 Sun 的 Hotspot JVM，在这里有一个完整的参数列表。我们建议至少要使用 -XX:+AggressiveOpts 和 -XX:+UseFastAccessorMethods 选项。