# 介绍

## 官网主页

主页：<http://rocketmq.apache.org/>

Github: <https://github.com/apache/rocketmq>

## 特点与概念

Apache RocketMQ 是阿里开源的一款高性能、高吞吐量的分布式消息中间件.

**特点**

支持 Broker 和 Consumer 端消息过滤

支持发布订阅模型，和点对点，

支持拉 pull 和推 push 两种消息模式

单一队列百万消息、亿级消息堆积

支持单 master 节点，多 master 节点，多 master 多 slave 节点

任意一点都是高可用，水平拓展，Producer、Consumer、队列都可以分布式

消息失败重试机制、支持特定 level 的定时消息

新版本底层采用 Netty

4.3.x 支持事务

适合金融类业务，高可用性跟踪和审计功能。

**基本概念**

1 消息模型（Message Model）

RocketMQ主要由 Producer、Broker、Consumer 三部分组成，其中Producer 负责生产消息，Consumer 负责消费消息，Broker 负责存储消息。Broker 在实际部署过程中对应一台服务器，每个 Broker 可以存储多个Topic的消息，每个Topic的消息也可以分片存储于不同的 Broker。Message Queue 用于存储消息的物理地址，每个Topic中的消息地址存储于多个 Message Queue 中。ConsumerGroup 由多个Consumer 实例构成。

2 消息生产者（Producer）

负责生产消息，一般由业务系统负责生产消息。一个消息生产者会把业务应用系统里产生的消息发送到broker服务器。RocketMQ提供多种发送方式，同步发送、异步发送、顺序发送、单向发送。同步和异步方式均需要Broker返回确认信息，单向发送不需要。

3 消息消费者（Consumer）

负责消费消息，一般是后台系统负责异步消费。一个消息消费者会从Broker服务器拉取消息、并将其提供给应用程序。从用户应用的角度而言提供了两种消费形式：拉取式消费、推动式消费。

4 主题（Topic）

表示一类消息的集合，每个主题包含若干条消息，每条消息只能属于一个主题，是RocketMQ进行消息订阅的基本单位。

5 代理服务器（Broker Server）

消息中转角色，负责存储消息、转发消息。代理服务器在RocketMQ系统中负责接收从生产者发送来的消息并存储、同时为消费者的拉取请求作准备。代理服务器也存储消息相关的元数据，包括消费者组、消费进度偏移和主题和队列消息等。

6 名字服务（Name Server）

名称服务充当路由消息的提供者。生产者或消费者能够通过名字服务查找各主题相应的Broker IP列表。多个Namesrv实例组成集群，但相互独立，没有信息交换。

7 拉取式消费（Pull Consumer）

Consumer消费的一种类型，应用通常主动调用Consumer的拉消息方法从Broker服务器拉消息、主动权由应用控制。一旦获取了批量消息，应用就会启动消费过程。

8 推动式消费（Push Consumer）

Consumer消费的一种类型，该模式下Broker收到数据后会主动推送给消费端，该消费模式一般实时性较高。

9 生产者组（Producer Group）

同一类Producer的集合，这类Producer发送同一类消息且发送逻辑一致。如果发送的是事务消息且原始生产者在发送之后崩溃，则Broker服务器会联系同一生产者组的其他生产者实例以提交或回溯消费。

10 消费者组（Consumer Group）

同一类Consumer的集合，这类Consumer通常消费同一类消息且消费逻辑一致。消费者组使得在消息消费方面，实现负载均衡和容错的目标变得非常容易。要注意的是，消费者组的消费者实例必须订阅完全相同的Topic。RocketMQ 支持两种消息模式：集群消费（Clustering）和广播消费（Broadcasting）。

11 集群消费（Clustering）

集群消费模式下,相同Consumer Group的每个Consumer实例平均分摊消息。

12 广播消费（Broadcasting）

广播消费模式下，相同Consumer Group的每个Consumer实例都接收全量的消息。

13 普通顺序消息（Normal Ordered Message）

普通顺序消费模式下，消费者通过同一个消费队列收到的消息是有顺序的，不同消息队列收到的消息则可能是无顺序的。

14 严格顺序消息（Strictly Ordered Message）

严格顺序消息模式下，消费者收到的所有消息均是有顺序的。

15 消息（Message）

消息系统所传输信息的物理载体，生产和消费数据的最小单位，每条消息必须属于一个主题。RocketMQ中每个消息拥有唯一的Message ID，且可以携带具有业务标识的Key。系统提供了通过Message ID和Key查询消息的功能。

16 标签（Tag）

为消息设置的标志，用于同一主题下区分不同类型的消息。来自同一业务单元的消息，可以根据不同业务目的在同一主题下设置不同标签。标签能够有效地保持代码的清晰度和连贯性，并优化RocketMQ提供的查询系统。消费者可以根据Tag实现对不同子主题的不同消费逻辑，实现更好的扩展性。

## 当前版本

本文使用的版本：4.7.1

# 安装

## 下载源码

<https://github.com/apache/rocketmq/releases>

## 打包

解压源码mvn打包：

cd rocketmq-all-4.7.1/

mvn -Prelease-all -DskipTests clean install –U

#源码打包后的目录

cd distribution/target/rocketmq-4.7.1/rocketmq-4.7.1

## 启动

### Linux

启动 Name Server：

> nohup sh bin/mqnamesrv &

> tail -f ~/logs/rocketmqlogs/namesrv.log

The Name Server boot success...

启动 Broker：

> nohup sh bin/mqbroker -n localhost:9876 &

> tail -f ~/logs/rocketmqlogs/broker.log

The broker[%s, 172.30.30.233:10911] boot success...

关闭 Server：

> sh bin/mqshutdown broker

The mqbroker(36695) is running...

Send shutdown request to mqbroker(36695) OK

> sh bin/mqshutdown namesrv

The mqnamesrv(36664) is running...

Send shutdown request to mqnamesrv(36664) OK

测试：

> export NAMESRV\_ADDR=localhost:9876

> sh bin/tools.sh org.apache.rocketmq.example.quickstart.Producer

SendResult [sendStatus=SEND\_OK, msgId= ...

> sh bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer

ConsumeMessageThread\_%d Receive New Messages: [MessageExt...

### Windows

设置环境变量：

ROCKETMQ\_HOME="D:\rocketmq" #你的rocketmq的目录

NAMESRV\_ADDR="localhost:9876"

启动 Name Server

.\bin\mqnamesrv.cmd

启动 Broker

.\bin\mqbroker.cmd -n localhost:9876 autoCreateTopicEnable=true

测试：

.\bin\tool.cmd org.apache.rocketmq.example.quickstart.Producer

.\bin\tool.cmd org.apache.rocketmq.example.quickstart.Consumer

# 控制台

## 下载源码

<https://github.com/apache/rocketmq-externals/tree/master/rocketmq-console>

## 修改配置

可修改application.properties，设置端口、nameserver、启动登录保护等等，可参考github上的说明。

<https://github.com/apache/rocketmq-externals/blob/master/rocketmq-console/doc/1_0_0/UserGuide_CN.md>

## 编译源码

mvn clean package –Dmaven.test.skip=true

## 启动console

Java –jar rocketmq-console-ng-2.0.0.jar

即可进入rocketmq管理界面

# 配置

官方配置说明

<https://github.com/apache/rocketmq/blob/master/docs/cn/best_practice.md>

注意有区分broker的配置，客户端配置，客户端有公共配置、生产端配置、消费端配置

# 整合SpringBoot

## 添加maven依赖

<dependency>

<groupId>**org.apache.rocketmq**</groupId>

<artifactId>**rocketmq-spring-boot-starter**</artifactId>

<version>**2.1.1**</version>

</dependency>

## 配置

rocketmq:

name-server: localhost:9876

# 生产者

producer:

group: my-group1

sendMessageTimeout: 3000

# 单元测试

## 生产者

@Autowired

private RocketMQTemplate rocketMQTemplate;

@GetMapping**(**"/send1"**)**

public Object send1**()** **{**

String des **=** "string-topic:tagA"**;**

String msg **=** "这是消息内容"**;**

//同步发送消息

rocketMQTemplate**.**syncSend**(**des**,** msg**);**

**return** "send ok"**;**

**}**

@GetMapping**(**"/send2"**)**

public Object send2**()** **{**

String des **=** "string-topic:tagB"**;**

String msg **=** "这是消息内容"**;**

String hashKey **=** "20200001"**;**

//发送消息并等待回应

String receive **=** rocketMQTemplate**.**sendAndReceive**(**des**,** msg**,** String**.**class**,** hashKey**);**

**return** receive**;**

**}**

@GetMapping**(**"/send3"**)**

public Object send3**()** **{**

User user **=** **new** User**();**

user**.**setAge**(**23**);**

user**.**setUsername**(**"tom"**);**

String des **=** "string-topic:tagC"**;**

//发送自定义对象

rocketMQTemplate**.**convertAndSend**(**des**,** user**);**

**return** "send ok"**;**

**}**

## 消费者

@Service

@RocketMQMessageListener**(**topic **=** "${demo.rocketmq.topic}"**,**

consumerGroup **=** "string\_consumer"**,** selectorExpression **=** "${demo.rocketmq.tag}"**)**

public class StringConsumer **implements** RocketMQListener**<**String**>** **{**

public void onMessage**(**String message**)** **{**

System**.**out**.**printf**(**"------- StringConsumer received: %s \n"**,** message**);**

**}**

**}**

@Service

@RocketMQMessageListener**(**topic **=** "${demo.rocketmq.topic}"**,**

consumerGroup **=** "string\_reply\_consumer"**,** selectorExpression **=** "tagB"**)**

public class StringReplyConsumer **implements** RocketMQReplyListener**<**String**,**String**>** **{**

public String onMessage**(**String message**)** **{**

System**.**out**.**printf**(**"------- StringReplyConsumer received: %s \n"**,** message**);**

**return** "reply string"**;**

**}**

**}**

@Service

@RocketMQMessageListener**(**topic **=** "${demo.rocketmq.topic}"**,**

consumerGroup **=** "user\_consumer"**,** selectorExpression **=** "tagC"**)**

public class UserConsumer **implements** RocketMQListener**<**User**>** **{**

public void onMessage**(**User message**)** **{**

System**.**out**.**printf**(**"------- UserReplyConsumer received: %s \n"**,** message**);**

**}**

**}**

## 更多示例

更多示例参考官方的example:

<https://github.com/apache/rocketmq-spring>