**每特学院 Java互联网架构培训-www.itmayiedu.com 余胜军老师QQ644064779**

**RocketMQ**

# RocketMQ概述

RocketMQ 是一款分布式、队列模型的消息中间件，具有以下特点： 能够保证严格的消息顺序 提供丰富的消息拉取模式 高效的订阅者水平扩展能力 实时的消息订阅机制 亿级消息堆积能力

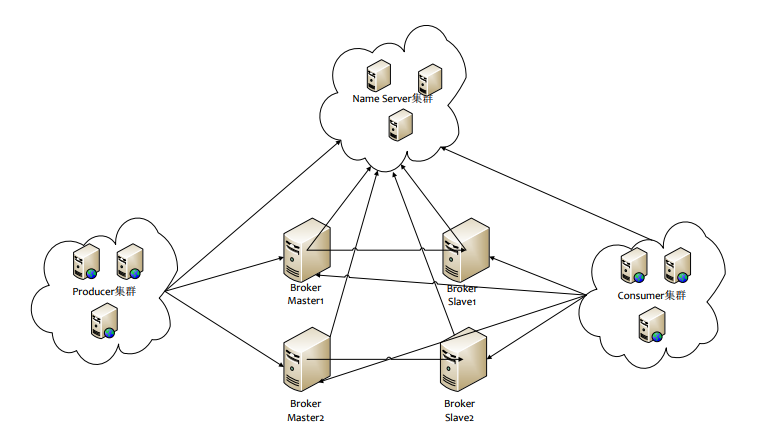
# RocketMQ包含的组件

NameServer：单点，供Producer和Consumer获取Broker地址

Producer：产生并发送消息

Consumer：接受并消费消息

Broker：消息暂存，消息转发



## Name Server

Name Server是RocketMQ的寻址服务。用于把Broker的路由信息做聚合。客户端依靠Name Server决定去获取对应topic的路由信息，从而决定对哪些Broker做连接。

Name Server是一个几乎无状态的结点，Name Server之间采取share-nothing的设计，互不通信。

对于一个Name Server集群列表，客户端连接Name Server的时候，只会选择随机连接一个结点，以做到负载均衡。

Name Server所有状态都从Broker上报而来，本身不存储任何状态，所有数据均在内存。

如果中途所有Name Server全都挂了，影响到路由信息的更新，不会影响和Broker的通信。

## Broker

Broker是处理消息存储，转发等处理的服务器。

Broker以group分开，每个group只允许一个master，若干个slave。

只有master才能进行写入操作，slave不允许。

slave从master中同步数据。同步策略取决于master的配置，可以采用同步双写，异步复制两种。

客户端消费可以从master和slave消费。在默认情况下，消费者都从master消费，在master挂后，客户端由于从Name Server中感知到Broker挂机，就会从slave消费。

Broker向所有的NameServer结点建立长连接，注册Topic信息。

# RocketMQ优点

1.强调集群无单点，可扩展

2.任意一点高可用，水平可扩展

3.海量消息堆积能力，消息堆积后，写入低延迟。

4.支持上万个队列

5.消息失败重试机制

6.消息可查询

7.开源社区活跃

8.成熟度（经过双十一考验）

# RocketMQ环境安装

## 服务器配置

192.168.110.187 nameServer1,brokerServer1

192.168.110.188 nameServer2,brokerServer2

## 添加Host文件

vi /etc/hosts

192.168.110.187 rocketmq-nameserver1

192.168.110.187 rocketmq-master1

192.168.110.188 rocketmq-nameserver2

192.168.110.188 rocketmq-master2

service network restart

注意: Error:No suitable device found: no device found for connection "System eth0"

解决办法:

|  |
| --- |
| (1)ifconfig -a 查看物理 MAC HWADDR 的值  (2)vim 编辑文件 /etc/sysconfig/network-scripts/ifcfg-eth0中修改ifconfig中查出的MAC HWADDR值； |

## 上传安装包

# 上传alibaba-rocketmq-3.2.6.tar.gz文件至/usr/local

# tar -zxvf alibaba-rocketmq-3.2.6.tar.gz -C /usr/local

# mv alibaba-rocketmq alibaba-rocketmq-3.2.6

# ln -s alibaba-rocketmq-3.2.6 rocketmq

## 创建存储路径【两台机器】

mkdir /usr/local/rocketmq/store

# mkdir /usr/local/rocketmq/store/commitlog

# mkdir /usr/local/rocketmq/store/consumequeue

# mkdir /usr/local/rocketmq/store/index

## RocketMQ配置文件【两台机器】

# vim /usr/local/rocketmq/conf/2m-noslave/broker-a.properties

# vim /usr/local/rocketmq/conf/2m-noslave/broker-b.properties

## 修改日志配置文件【两台机器】

# mkdir -p /usr/local/rocketmq/logs

# cd /usr/local/rocketmq/conf && sed -i 's#${user.home}#/usr/local/rocketmq#g' \*.xml

## 修改启动NameServer【两台机器】

vim /usr/local/rocketmq/bin/runbroker.sh

|  |
| --- |
| JAVA\_OPT="${JAVA\_OPT} -server -Xms1g -Xmx1g -Xmn512m - XX:PermSize=128m -XX:MaxPermSize=320m" |

vim /usr/local/rocketmq/bin/runserver.sh

|  |
| --- |
| JAVA\_OPT="${JAVA\_OPT} -server -Xms1g -Xmx1g -Xmn512m - XX:PermSize=128m -XX:MaxPermSize=320m" |

## 启动NameServer【两台机器】

# cd /usr/local/rocketmq/bin  
# nohup sh mqnamesrv &

## 启动BrokerServer A

cd /usr/local/rocketmq/bin  
# nohup sh mqbroker -c /usr/local/rocketmq/conf/2m-noslave/broker-a.properties >/dev/null 2>&1 &  
# netstat -ntlp  
# jps  
# tail -f -n 500 /usr/local/rocketmq/logs/rocketmqlogs/broker.log  
# tail -f -n 500 /usr/local/rocketmq/logs/rocketmqlogs/namesrv.log

## 启动BrokerServer B

cd /usr/local/rocketmq/bin  
# nohup sh mqbroker -c /usr/local/rocketmq/conf/2m-noslave/broker-b.properties >/dev/null 2>&1 &  
# netstat -ntlp  
# jps  
# tail -f -n 500 /usr/local/rocketmq/logs/rocketmqlogs/broker.log  
# tail -f -n 500 /usr/local/rocketmq/logs/rocketmqlogs/namesrv.log

## RocketMQ Console

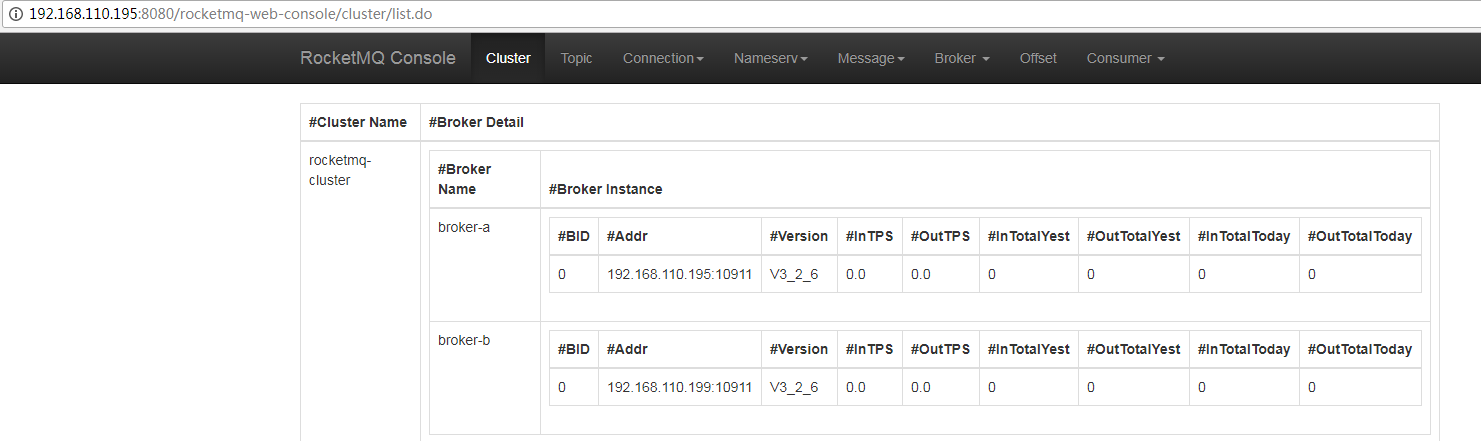
将rocketmq-web-console 部署到webapps目录中。

/usr/local/apache-tomcat-7.0.65/webapps/rocketmq-web-console/WEB-INF/classes/

修改config.properties

rocketmq.namesrv.addr=192.168.110.195:9876;192.168.110.199:9876

## 运行效果



## 安装jdk环境

vi /etc/profile

|  |
| --- |
| export JAVA\_HOME=/usr/local/jdk1.7.0\_80  export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  export PATH=$JAVA\_HOME/bin:$PATH |

source /etc/profile

# Java操作RocketMQ

## Pom

|  |
| --- |
| **<dependencies>**  **<dependency>**  **<groupId>com.alibaba.rocketmq</groupId>**  **<artifactId>rocketmq-client</artifactId>**  **<version>3.0.10</version>**  **</dependency>**  **<dependency>**  **<groupId>com.alibaba.rocketmq</groupId>**  **<artifactId>rocketmq-all</artifactId>**  **<version>3.0.10</version>**  **<type>pom</type>**  **</dependency>**  **<dependency>**  **<groupId>ch.qos.logback</groupId>**  **<artifactId>logback-classic</artifactId>**  **<version>1.1.1</version>**  **</dependency>**  **<dependency>**  **<groupId>ch.qos.logback</groupId>**  **<artifactId>logback-core</artifactId>**  **<version>1.1.1</version>**  **</dependency>**  **<dependency>**  **<groupId>junit</groupId>**  **<artifactId>junit</artifactId>**  **<version>4.10</version>**  **<scope>test</scope>**  **</dependency>**  **</dependencies>** |

## 生产者

|  |
| --- |
| **public** **class** Producer {  **public** **static** **void** main(String[] args) **throws** MQClientException {  DefaultMQProducer producer = **new** DefaultMQProducer("rmq-group");  producer.setNamesrvAddr("192.168.110.195:9876;192.168.110.199:9876");  producer.setInstanceName("producer");  producer.start();  **try** {  **for** (**int** i = 0; i < 10; i++) {  Thread.*sleep*(1000); // 每秒发送一次MQ  Message msg = **new** Message("itmayiedu-topic", // topic 主题名称  "TagA", // tag 临时值  ("itmayiedu-"+i).getBytes()// body 内容  );  SendResult sendResult = producer.send(msg);  System.***out***.println(sendResult.toString());  }  } **catch** (Exception e) {  e.printStackTrace();  }  producer.shutdown();  }  } |

## 消费者

|  |
| --- |
| **public** **class** Consumer {  **public** **static** **void** main(String[] args) **throws** MQClientException {  DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer("rmq-group");  consumer.setNamesrvAddr("192.168.110.195:9876;192.168.110.199:9876");  consumer.setInstanceName("consumer");  consumer.subscribe("itmayiedu-topic", "TagA");  consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {  **for** (MessageExt msg : msgs) {  System.***out***.println(msg.getMsgId()+"---"+**new** String(msg.getBody()));  }  **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });  consumer.start();  System.***out***.println("Consumer Started.");  }  } |

# RocketMQ重试机制

MQ 消费者的消费逻辑失败时，可以通过设置返回状态达到消息重试的结果。

MQ 消息重试只针对集群消费方式生效；广播方式不提供失败重试特性，即消费失败后，失败消息不再重试，继续消费新的消息。

|  |
| --- |
| **public** **class** Consumer {  **public** **static** **void** main(String[] args) **throws** MQClientException {  DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer("rmq-group");  consumer.setNamesrvAddr("192.168.110.195:9876;192.168.110.199:9876");  consumer.setInstanceName("consumer");  consumer.subscribe("itmayiedu-topic", "TagA");  consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {  **for** (MessageExt msg : msgs) {  System.***out***.println(msg.getMsgId() + "---" + **new** String(msg.getBody()));  }  **try** {  **int** i = 1 / 0;  } **catch** (Exception e) {  e.printStackTrace();  // 需要重试  **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  }  // 不需要重试  **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });  consumer.start();  System.***out***.println("Consumer Started.");  }  } |

注意:每次重试后，消息ID都不一致，所以不能使用消息ID判断幂等。

## RocketMQ如何解决消息幂等

注意:每次重试后，消息ID都不一致，所以不能使用消息ID判断幂等。

解决办法:使用自定义全局ID判断幂等，例如流水ID、订单号

使用msg.setKeys 进行区分

### 生产者:

|  |
| --- |
| **public** **class** Producer {  **public** **static** **void** main(String[] args) **throws** MQClientException {  DefaultMQProducer producer = **new** DefaultMQProducer("rmq-group");  producer.setNamesrvAddr("192.168.110.195:9876;192.168.110.199:9876");  producer.setInstanceName("producer");  producer.start();  **try** {  **for** (**int** i = 0; i < 1; i++) {  Thread.*sleep*(1000); // 每秒发送一次MQ  Message msg = **new** Message("itmayiedu-topic", // topic 主题名称  "TagA", // tag 临时值  ("itmayiedu-6" + i).getBytes()// body 内容  );  msg.setKeys(System.*currentTimeMillis*() + "");  SendResult sendResult = producer.send(msg);  System.***out***.println(sendResult.toString());  }  } **catch** (Exception e) {  e.printStackTrace();  }  producer.shutdown();  }  } |

### 消费者:

|  |
| --- |
| **static** **private** Map<String, String> *logMap* = **new** HashMap<>();  **public** **static** **void** main(String[] args) **throws** MQClientException {  DefaultMQPushConsumer consumer = **new** DefaultMQPushConsumer("rmq-group");  consumer.setNamesrvAddr("192.168.110.195:9876;192.168.110.199:9876");  consumer.setInstanceName("consumer");  consumer.subscribe("itmayiedu-topic", "TagA");  consumer.registerMessageListener(**new** MessageListenerConcurrently() {  @Override  **public** ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {  String key = **null**;  String msgId = **null**;  **try** {  **for** (MessageExt msg : msgs) {  key = msg.getKeys();  **if** (*logMap*.containsKey(key)) {  // 无需继续重试。  System.***out***.println("key:"+key+",无需重试...");  **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  msgId = msg.getMsgId();  System.***out***.println("key:" + key + ",msgid:" + msgId + "---" + **new** String(msg.getBody()));  **int** i = 1 / 0;  }  } **catch** (Exception e) {  e.printStackTrace();  **return** ConsumeConcurrentlyStatus.***RECONSUME\_LATER***;  } **finally** {  *logMap*.put(key, msgId);  }  **return** ConsumeConcurrentlyStatus.***CONSUME\_SUCCESS***;  }  });  consumer.start();  System.***out***.println("Consumer Started.");  } |

# 参考文献

https://help.aliyun.com/document\_detail/44397.html?spm=a2c4g.11186623.6.577.sPsbuu