



小·D·课堂

愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第一章深入剖析高性能RocketMQ4.X实战课程概要

第1集 互联网架构之深入剖析高性能RocketMQ4.X实战课程介绍

简介：深入剖析高性能RocketMQ4.X实战简介、适用人群和学习基础



小·D·课堂

愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第二章 JMS介绍和消息中间件核心知识

第1集 JMS消息服务介绍和使用场景

简介：讲解什么是JMS，消息队列的使用场景

- 什么是JMS: Java消息服务 (Java Message Service),Java平台中关于面向消息中间件的接口
- JMS是一种与厂商无关的 API，用来访问消息收发系统消息，它类似于JDBC(Java Database Connectivity)。这里，JDBC 是可以用来访问许多不同关系数据库的 API
- 使用场景：
 - 核心应用
 - 解耦：订单系统-》物流系统
 - 异步：用户注册-》发送邮件，初始化信息

- 削峰：秒杀、日志处理
- 跨平台、多语言
- 分布式事务、最终一致性
- RPC调用上下游对接，数据源变动->通知下属

第2集 消息中间件常见概念和编程模型

简介：讲解消息中间件的常见概念和基础编程模型

- 常见概念
 - JMS提供者：连接面向消息中间件的，JMS接口的一个实现，RocketMQ,ActiveMQ,Kafka等等
 - JMS生产者(Message Producer)：生产消息的服务
 - JMS消费者(Message Consumer)：消费消息的服务
 - JMS消息：数据对象
 - JMS队列：存储待消费消息的区域
 - JMS主题：一种支持发送消息给多个订阅者的机制
 - JMS消息通常有两种类型：点对点 (Point-to-Point)、发布/订阅 (Publish/Subscribe)
- 基础编程模型
 - MQ中需要用的一些类
 - ConnectionFactory：连接工厂，JMS 用它创建连接
 - Connection：JMS 客户端到JMS Provider 的连接
 - Session：一个发送或接收消息的线程
 - Destination：消息的目的地;消息发送给谁.
 - MessageConsumer / MessageProducer：消息消费者，消息生产者

第3集 主流消息队列和技术选型讲解

简介：对比当下主流的消息队列和选择问题

- Apache ActiveMQ、Kafka、RabbitMQ、RocketMQ
 - ActiveMQ：<http://activemq.apache.org/>
 - Apache出品，历史悠久，支持多种语言的客户端和协议，支持多种语言Java, .NET, C++ 等，基于JMS Provider的实现
 - 缺点：吞吐量不高，多队列的时候性能下降，存在消息丢失的情况，比较少大规模使用
 - Kafka：<http://kafka.apache.org/>

- 是由Apache软件基金会开发的一个开源流处理平台，由Scala和Java编写。Kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理大规模的网站中的所有动作流数据(网页浏览，搜索和其他用户的行动)，副本集机制，实现数据冗余，保障数据尽量不丢失；支持多个生产者和消费者

缺点：不支持批量和广播消息，运维难度大，文档比较少，需要掌握Scala

- RabbitMQ: <http://www.rabbitmq.com/>

- 是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不错

缺点：使用Erlang开发，阅读和修改源码难度大

- RocketMQ: <http://rocketmq.apache.org/>

- 阿里开源的一款的消息中间件，纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点，性能强劲(零拷贝技术)，支持海量堆积，支持指定次数和时间间隔的失败消息重发，支持consumer端tag过滤、延迟消息等，在阿里内部进行大规模使用，适合在电商，互联网金融等领域使用



小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第三章 RocketMQ4.X基础介绍和阿里云Linux服务器快速部署

第1集 阿里巴巴开源RocketMQ4.x消息队列介绍

简介：阿里开源消息队列 RocketMQ4.x介绍和新概念讲解

- Apache RocketMQ作为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支持分布式事务
 - 适合金融类业务，高可用性跟踪和审计功能。
- 概念
 - Producer:消息生产者
 - Producer Group:消息生产者组，发送同类消息的一个消息生产组
 - Consumer:消费者
 - Consumer Group:消费同类消息的多个实例
 - Tag:标签，子主题（二级分类）对topic的进一步细化,用于区分同一个主题下的不同业务的消息
 - Topic:主题, 如订单类消息，queue是消息的物理管理单位，而topic是逻辑管理单位。一个topic下可以有多个queue，
默认自动创建是4个，手动创建是8个
 - Message：消息,每个message必须指定一个topic
 - Broker：MQ程序，接收生产的消息，提供给消费者消费的程序
 - Name Server：给生产和消费者提供路由信息，提供轻量级的服务发现、路由、元数据信息，可以多个部署，互相独立（比zookeeper更轻量）
 - Offset: 偏移量，可以理解为消息进度
 - commit log: 消息存储会写在Commit log文件里面
 - 走读官网地址，学会如何学习新技术 <http://rocketmq.apache.org/>
 - 学习资源
 - <http://jm.taobao.org/2017/01/12/rocketmq-quick-start-in-10-minutes/>
 - <https://www.jianshu.com/p/453c6e7ff81c>

第2集 RocketMQ4.x本地源码部署(苹果系统底层是Unix系统)

简介:RocketMQ4.x本地快速部署

- 安装前提条件(推荐) 64bit OS, Linux/Unix/Mac (Windows不兼容) 64bit JDK 1.8+;
- 快速开始 <http://rocketmq.apache.org/docs/quick-start/> 下载安装包：<http://mirror.bit.edu.cn/apache/rocketmq/4.4.0/rocketmq-all-4.4.0-source-release.zip>

```
unzip rocketmq-all-4.4.0-source-release.zip
cd rocketmq-all-4.4.0/
mvn -Prelease-all -DskipTests clean install -U
cd distribution/target/apache-rocketmq
最终路径 rocketmq-all-4.4.0/distribution/target/apache-rocketmq
```

- 最新版本部署存在问题：

- Please set the JAVA_HOME variable in your environment, We need java(x64)
- 解决：本地需要配置 JAVA_HOME 使用命令 vim ~/.bash_profile

```
JAVA_HOME="/Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home"
export JAVA_HOME
CLASS_PATH="$JAVA_HOME/lib"
PATH=".$PATH:$JAVA_HOME/bin"
```

- 解压压缩包

- 启动nameServer

```
nohup sh bin/mqnamesrv &
```

- 查看日志 tail -f nohup.out (结尾：The Name Server boot success. serializeType=JSON 表示启动成功)
- 启动broker (-n指定nameserver地址，nameserver服务端口为9876, broker默认端口 10911)

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

- 关闭nameserver broker执行的命令

```
sh bin/mqshutdown broker
sh bin/mqshutdown namesrv
```

- 使用 jps查看进程

- 验证是否成功

```
#设置名称服务地址
export NAMESRV_ADDR=localhost:9876
#投递消息
sh bin/tools.sh org.apache.rocketmq.example.quickstart.Producer

SendResult [sendStatus=SEND_OK, msgId= ...

#消费消息
sh bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer

ConsumeMessageThread_%d Receive New Messages: [MessageExt...
```

第3集 源码安装RocketMQ4.x可视化控制台

简介：源码安装RocketMQ4.x可视化控制台

- 下载 <https://github.com/apache/rocketmq-externals>
- 编译打包 `mvn clean package -Dmaven.test.skip=true`
 - 存在问题，因为rocketmq-console 里面 pom.xml 版本问题

```
[ERROR] Failed to execute goal on project rocketmq-console-ng: Could not
resolve dependencies for project org.apache:rocketmq-console-
ng:jar:1.0.0: The following artifacts could not be resolved:
org.apache.rocketmq:rocketmq-tools:jar:4.4.0-SNAPSHOT,
org.apache.rocketmq:rocketmq-namesrv:jar:4.4.0-SNAPSHOT,
org.apache.rocketmq:rocketmq-broker:jar:4.4.0-SNAPSHOT: Could not find
artifact org.apache.rocketmq:rocketmq-tools:jar:4.4.0-SNAPSHOT -> [Help
1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the
-e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
```

- 解决 `<rocketmq.version>4.4.0-SNAPSHOT</rocketmq.version>`
改为 `<rocketmq.version>4.4.0</rocketmq.version>`
 - target目录 通过java -jar的方式运行, 启动后是 8080端口
- 其他常见问题：

1) 无法连接获取broker信息

修改配置文件,名称路由地址为 namesrvAddr, 例如我本机地址为
src/main/resources/application.properties
rocketmq.config.namesrvAddr=127.0.0.1:9876

2) 连接不成功

在阿里云, 腾讯云或者虚拟机, 记得检查端口号和防火墙是否启动
阿里云控制台有安全组, 需要开放对应的端口

第4集 RocketMQ4.x可视化控制台讲解

简介: 讲解新版的RocketMQ可视化管理后台

- 管理控制台地址 127.0.0.1:8080

测试中遇到的问题

maybe your broker machine memory too small

原因: 磁盘空间不够

第5集 阿里云Linux服务器介绍和使用讲解

简介: 阿里云服务器介绍和使用讲解

第6集 阿里云Linux服务器部署JDK8实战

简介: 在阿里云服务器上安装JDK8和配置环境变量

- 地址: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- linux下使用wget下载jdk8:
进到目录/usr/local/software 配置环境变量

```
解压: tar -zxvf jdk-8u201-linux-x64.tar.gz
重命名: mv jdk1.8.0_201 jdk8
vim /etc/profile
加入
export JAVA_HOME=/usr/local/software/jdk8
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME PATH CLASSPATH
使用 source /etc/profile 让配置立刻生效
```

- windows 命令行工具：putty
- windows 远程上传工具: wscp
- Mac 命令行工具: Item2
- Mac 远程上传工具：Filezilla

第7集阿里云Linux服务器安装Maven实战

简介: Linux服务器下安装Maven

```
解压: tar -zxvf apache-maven-3.6.0-bin.tar.gz
重命名: mv apache-maven-3.6.0 maven
vim /etc/profile
export PATH=/usr/local/software/maven/bin:$PATH

立刻生效: source /etc/profile
```

第8集 阿里云服务器源码部署RocketMQ4.X（Linux系统）

简介: 官方下载最新源码包，阿里云Linux服务器部署，解决内存不够问题

- 文档地址: <http://rocketmq.apache.org/docs/quick-start/>
- Linux 解压安装 yum install unzip
- 常见问题
 - NameServer内存不够怎么处理

- 找到 runserver.sh 修改 JAVA_OPT

报错问题如下

```
[root@iZwz94swl88z3yfl7lpmmSZ apache-rocketmq]# sh bin/mqnamesrv
Java HotSpot(TM) 64-Bit Server VM warning: Using the DefNew
young collector with the CMS collector is deprecated and will
likely be removed in a future release
Java HotSpot(TM) 64-Bit Server VM warning:
UseCMSCompactAtFullCollection is deprecated and will likely be
removed in a future release.
Java HotSpot(TM) 64-Bit Server VM warning: INFO:
os::commit_memory(0x00000006ec800000, 2147483648, 0) failed;
error='Cannot allocate memory' (errno=12)
#
# There is insufficient memory for the Java Runtime Environment
to continue.
# Native memory allocation (mmap) failed to map 2147483648 bytes
for committing reserved memory.
# An error report file with more information is saved as:
# /usr/local/software/rocketmq-all-
4.4.0/distribution/target/apache-rocketmq/hs_err_pid8993.log
```

解决如下 编辑 bin/runserver.sh:

```
JAVA_OPT="${JAVA_OPT} -server -Xms256m -Xmx256m -Xmn256m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

- **Broker内存不足**

- 找到 runbroker.sh 修改 JAVA_OPT

```
JAVA_OPT="${JAVA_OPT} -server -Xms528m -Xmx528m -Xmn256m"
```

第9集 阿里云源码安装RocketMQ4.X控制台（Linux系统）

简介：阿里云服务器安装RocketMQ控制台

- 上传源码包-》解压-》进入rocketmq-console目录-》编译打包 mvn clean package -Dmaven.test.skip=true

务必修改下面两个，再进行编译打包

- 修改 pom.xml 版本号 (官方bug)
- 修改application.xml里面的nameserver地址
- 进入target目录，启动 java -jar rocketmq-console-ng-1.0.0.jar
- 守护进程方式启动 nohup java -jar rocketmq-console-ng-1.0.0.jar &

第10集 RocketMQ4.4.X源码导入IDEA

简介：将RocketMQ源码导入IDEA,为后续阅读源码做准备

- 下载源码，解压，导入即可



小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第四章 Springboot2.X整合RocketMQ4.X实战

第1集 Springboot2.x整合RocketMQ4.x实战发送消息

简介：Springboot2.x整合RocketMQ4.x实战，加入相关依赖，开发生产者代码

注意：记得启动nameser和broker

- 快速创建springboot项目 <https://start.spring.io/>
- 加入相关依赖

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>4.3.0</version>
</dependency>
```

- Message对象
 - topic: 主题名称
 - tag: 标签，用于过滤
 - key: 消息唯一标示，可以是业务字段组合

- body: 消息体,字节数组
- 注意 发送消息到Broker, 需要判断是否有此topic启动broker的时候,
本地环境建议开启自动创建topic, 生产环境建议关闭自动化创建topic
建议先手工创建Topic, 如果靠程序自动创建, 然后再投递消息, 会出现延迟情况
- 概念模型: 一个topic下面对应多个queue,可以在创建Topic时指定, 如订单类topic
- 通过可视化管理后台查看消息

常见错误一

```
org.apache.rocketmq.remoting.exception.RemotingTooMuchRequestException:
sendDefaultImpl call timeout
```

原因: 阿里云存在多网卡, rocketmq都会根据当前网卡选择一个IP使用, 当你的机器有多块网卡时, 很有可能会出问题。比如, 我遇到的问题是机器上有两个IP, 一个公网IP, 一个私网IP, 因此需要配置 broker.conf 指定当前的公网ip, 然后重新启动broker

新增配置: conf/broker.conf (属性名称brokerIP1=broker所在的公网ip地址)

新增这个配置: brokerIP1=120.76.62.13

启动命令: `nohup sh bin/mqbroker -n localhost:9876 -c ./conf/broker.conf &`

常见错误二

```
MQClientException: No route info of this topic, TopicTest1
```

原因: Broker 禁止自动创建 Topic, 且用户没有通过手工方式创建 此Topic, 或者broker和 Nameserver网络不通

解决:

通过 `sh bin/mqbroker -m` 查看配置

autoCreateTopicEnable=true 则自动创建topic

Centos7关闭防火墙 `systemctl stop firewalld`

常见错误三

控制台查看不了数据, 提示连接 10909错误

原因: Rocket默认开启了VIP通道, VIP通道端口为10911-2=10909

解决: 阿里云安全组需要增加一个端口 10909

其他错误:

https://blog.csdn.net/qq_14853889/article/details/81053145
<https://blog.csdn.net/wangmx1993328/article/details/81588217#%E5%BC%82%E5%B8%B8%E8%AF%B4%E6%98%8E>
<https://www.jianshu.com/p/bfd6d849f156>
<https://blog.csdn.net/wangmx1993328/article/details/81588217>

第2集 Springboot2.x整合RocketMQ4.x实战消费消息

简介：Springboot2.x整合RocketMQ4.x实战，开发消费者代码，常见问题处理

- 自动创建topic：autoCreateTopicEnable=true 无效原因：客户端版本要和服务端版本保持一致
- 创建消费者

```
consumer = new DefaultMQPushConsumer(consumerGroup);
consumer.setNamesrvAddr(nameServerAddr);

consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET)
;

consumer.subscribe(topic, "*");
consumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
    try {
        Message msg = msgs.get(0);
        System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), new String(msgs.get(0).getBody()));
        String topic = msg.getTopic();
        String body = new String(msg.getBody(), "utf-8");
        String tags = msg.getTags();
        String keys = msg.getKeys();
        System.out.println("topic=" + topic + ", tags=" + tags + ",
keys=" + keys + ", msg=" + body);
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        return ConsumeConcurrentlyStatus.RECONSUME_LATER;
    }
});

consumer.start();
```

- 常见问题

1、Caused by:

```
org.apache.rocketmq.remoting.exception.RemotingConnectException: connect to <172.17.42.1:10911> failed
```

2、com.alibaba.rocketmq.client.exception.MQClientException: Send [1] times, still failed, cost [1647]ms, Topic: TopicTest1, BrokersSent: [broker-a, null, null]

3、org.apache.rocketmq.client.exception.MQClientException: Send [3] times, still failed, cost [497]ms, Topic: TopicTest, BrokersSent: [Book-Air.local, MacBook-Air.local, MacBook-Air.local]

解决: 多网卡问题处理

1、设置producer: producer.setVipChannelEnabled(false);

2、编辑ROCKETMQ 配置文件: broker.conf (下列ip为自己的ip)

```
namesrvAddr = 192.168.0.101:9876
```

```
brokerIP1 = 192.168.0.101
```

4、DESC: service not available now, maybe disk full, CL:

解决: 修改启动脚本runbroker.sh, 在里面增加一句话即可:

```
JAVA_OPT="${JAVA_OPT} -
```

```
Drocketmq.broker.diskSpaceWarningLevelRatio=0.98"
```

(磁盘保护的百分比设置成98%, 只有磁盘空间使用率达到98%时才拒绝接收producer消息)

常见问题处理

<https://blog.csdn.net/sqzhao/article/details/54834761>

<https://blog.csdn.net/mayifan0/article/details/67633729>

<https://blog.csdn.net/a906423355/article/details/78192828>



小D课堂 愿景: "让编程不在难学, 让技术与生活更加有趣" 更多教程请访问 xdclass.net

第五章 高级篇幅之RocketMQ4.X集群架构讲解

第1集 RocketMQ4.X集群模式架构分析

简介: 讲解RocketMQ4.X多种集群模式讲解

1. 单节点:

优点：本地开发测试，配置简单，同步刷盘消息一条都不会丢

缺点：不可靠，如果宕机，会导致服务不可用

2. 主从(异步、同步双写)：

优点：同步双写消息不丢失，异步复制存在少量丢失，主节点宕机，从节点可以对外提供消息的消费，但是不支持写入

缺点：主备有短暂消息延迟，毫秒级，目前不支持自动切换，需要脚本或者其他程序进行检测然后进行停止broker,重启让从节点成为主节点

3. 双主：

优点：配置简单，可以靠配置RAID磁盘阵列保证消息可靠，异步刷盘丢失少量消息

缺点：master机器宕机期间，未被消费的消息在机器恢复之前不可消费，实时性会受到影响

4. 双主双从，多主多从模式（异步复制）

优点：磁盘损坏，消息丢失的非常少，消息实时性不会受影响，Master 宕机后，消费者仍然可以从Slave消费

缺点：主备有短暂消息延迟，毫秒级，如果Master宕机，磁盘损坏情况，会丢失少量消息

5. 双主双从，多主多从模式（同步双写）

优点：同步双写方式，主备都写成功，向应用才返回成功，服务可用性与数据可用性都非常高

缺点：性能比异步复制模式略低，主宕机后，备机不能自动切换为主机

推荐方案2、4、5

第2集 消息可靠性之同步、异步刷盘

简介：讲解什么是同步刷盘和异步刷盘，主从模式如何保障消息可靠性

- 内存+磁盘
- 什么是异步刷盘(数据可能丢失,性能高):
- 什么是同步刷盘：数据安全性高
- 选择：各有优缺点，看业务需要

第3集 消息可靠性之同步、异步复制

简介：讲解消息的同步和异步复制

- Master - Slave节点里面
- 异步复制：数据可能丢失，性能高

- 同步复制: 数据安全性高, 性能低一点
- 最终推荐这种方式: 同步双写(即M-S同步复制), 异步刷盘

第4集 RocketMQ4.X集群高可用之主从模式搭建上集

简介: 使用RocketMQ4.X搭建主从节点上集

机器列表

```
server1 ssh root@192.168.159.129
server2 ssh root@192.168.159.130
server3 ssh root@192.168.159.131
server4 ssh root@192.168.159.132
```

软件: RocketMQ4.X + JDK8 + Maven +CentOS7

第5集 RocketMQ4.X集群高可用之主从模式搭建下集

简介: 使用RocketMQ4.X搭建主从节点下集

机器列表

```
server1 ssh root@192.168.159.129
server2 ssh root@192.168.159.130
server3 ssh root@192.168.159.131
server4 ssh root@192.168.159.132
```

1、修改RocketMQ(启动内存配置, 两个机器都要修改)

```
vim runserver.sh
JAVA_OPT="{JAVA_OPT} -server -Xms528m -Xmx528m -Xmn256m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

```
vim runbroker.sh
JAVA_OPT="{JAVA_OPT} -server -Xms528m -Xmx528m -Xmn256m"
```

启动两个机器的 nameserver
nohup sh bin/mqnamesrv &

全路径
/usr/local/software/rocketmq/distribution/target/apache-rocketmq

2、编辑并启动rocketmq命令

主节点
nohup sh bin/mqbroker -c conf/2m-2s-async/broker-a.properties &

```
namesrvAddr=192.168.159.129:9876;192.168.159.130:9876
brokerClusterName=XdcClassCluster
brokerName=broker-a
brokerId=0
deleteWhen=04
fileReservedTime=48
brokerRole=ASYNC_MASTER
flushDiskType=ASYNC_FLUSH
```

从节点
nohup sh bin/mqbroker -c conf/2m-2s-async/broker-a-s.properties &

```
namesrvAddr=192.168.159.129:9876;192.168.159.130:9876
brokerClusterName=XdcClassCluster
brokerName=broker-a
brokerId=1
deleteWhen=04
fileReservedTime=48
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
```

3、使用管控台

修改事项

pom.xml 里面的rocketmq版本号
application.properties里面的nameserver

增加 rocketmq.config.namesrvAddr=192.168.159.129:9876;192.168.159.130:9876

```
mvn install -Dmaven.test.skip=true
```

```
java -jar rocketmq-console-ng-1.0.0.jar
```

参考命令

centos7关闭防火墙

```
systemctl stop firewalld
```

远程拷贝到本地

```
scp xdclass@192.168.0.106:/Users/xdclass/Desktop/xdclass/消息队列/data/第3章/第7集/apache-maven-3.6.0-bin.tar.gz /usr/local/software
```

```
scp root@192.168.0.106:/Users/xdclass/Desktop/xdclass/消息队列/data/第3章/第7集/apache-maven-3.6.0-bin.tar.gz /usr/local/software
```

第6集 故障演练之主节点Broker退出保证消息可用

简介：讲解主节点Broker退出后，从节点可继续被消费者消费

步骤

- 发送一条消息，关闭主节点，关闭主节点之后不能写入
- 从节点提供数据供外面消费，但不能接受新消息
- 主节点上线后同步从节点已经被消费的数据(offset同步)

第7集 RocketMQ4.X主从同步必备知识点

简介：讲解RocketMQ主从同步必备知识点

- Broker分为master与slave，一个master可以对应多个Slave，但一个slave只能对应一个master，**master与slave通过相同的Broker Name来匹配**，不同的broker Id来定义是master还是slave
 - Broker向所有的NameServer结点建立长连接，定时注册Topic和发送元数据信息
 - NameServer定时扫描(默认2分钟)所有存活broker的连接, 如果超过时间没响应则断开连接(心跳检测)，但是consumer客户端不能感知，consumer定时(30s)从NameServer获取topic的最新信息，所以broker不可用时，consumer最多最需要30s才能发现
(Producer的机制一样，在未发现broker宕机前发送的消息会失败)
- 只有master才能进行写入操作，slave不允许写入只能同步，同步策略取决于master的配置。
- 客户端消费可以从master和slave消费，默认消费者都从master消费，如果在master挂后，客户端从NameServer中感知到Broker宕机，就会从slave消费，感知非实时，存在一定的滞后性，slave不能保证master的消息100%都同步过来了，会有少量的消息丢失。但一旦master恢复，未同步过去的消息会被最终消费掉
- 如果consumer实例的数量比message queue的总数量还多的话，多出来的consumer实例将无法分到queue，也就无法消费到消息，也就无法起到分摊负载的作用，所以需要控制让queue的总数量大于等于consumer的数量



小D课堂

愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第六章 RocketMQ生产者核心配置和核心知识讲解

第1集 消息队列RocketMQ4.X生产者核心配置讲解

简介：消息队列RocketMQ4.X核心配置讲解

- 生产者常见核心配置

- compressMsgBodyOverHowmuch : 消息超过默认字节4096后进行压缩
- retryTimesWhenSendFailed : 失败重发次数
- maxMessageSize : 最大消息配置, 默认128k
- topicQueueNums : 主题下面的队列数量, 默认是4
- autoCreateTopicEnable : 是否自动创建主题Topic, 开发建议为true, 生产要为false
- defaultTopicQueueNums : 自动创建服务器不存在的topic, 默认创建的队列数
- autoCreateSubscriptionGroup: 是否允许 Broker 自动创建订阅组, 建议线下开发开启, 线上关闭
- brokerClusterName : 集群名称
- brokerId : 0表示Master主节点 大于0表示从节点
- brokerIP1 : Broker服务地址
- brokerRole : broker角色 `ASYNC_MASTER/ SYNC_MASTER/ SLAVE`
- deleteWhen : 每天执行删除过期文件的时间, 默认每天凌晨4点
- flushDiskType : 刷盘策略, 默认为 `ASYNC_FLUSH`(异步刷盘), 另外是`SYNC_FLUSH`(同步刷盘)
- listenPort : Broker监听的端口号
- mappedFileSizeCommitLog : 单个commitlog文件大小, 默认是1GB
- mappedFileSizeConsumeQueue: ConsumeQueue每个文件默认存30W条, 可以根据项目调整
- storePathRootDir : 存储消息以及一些配置信息的根目录 默认为用户的 `${HOME}/store`
- storePathCommitLog: commitlog存储目录默认为`${storePathRootDir}/commitlog`
- storePathIndex: 消息索引存储路径
- syncFlushTimeout : 同步刷盘超时时间
- diskMaxUsedSpaceRatio : 检测可用的磁盘空间大小, 超过后会写入报错

第2集 核心知识之RocketMQ4.X消息发送状态

简介: 讲解RocketMQ消息常见发送状态

- 消息发送有同步和异步
- Broker消息投递状态讲解
 - FLUSH_DISK_TIMEOUT
 - 没有在规定时间内完成刷盘 (刷盘策略需要为`SYNC_FLUSH` 才会出这个错误)
 - FLUSH_SLAVE_TIMEOUT
 - 主从模式下, broker是`SYNC_MASTER`, 没有在规定时间内完成主从同步

- SLAVE_NOT_AVAILABLE
 - 从模式下, broker是SYNC_MASTER, 但是没有找到被配置成Slave的Broker
- SEND_OK
 - 发送成功, 没有发生上面的三种问题

第3集 核心知识之RocketMQ4.X生产和消费消息重试及处理

简介: 讲解RocketMQ消息生产和消费异常重试和阈值设定

- 生产者Producer重试 (异步和SendOneWay下配置无效)
 - 消息重投(保证数据的高可靠性),本身内部支持重试, 默认次数是2,
 - 如果网络情况比较差, 或者跨集群则建改多几次
- 消费端重试
 - 原因: 消息处理异常、broker端到consumer端各种问题, 如网络原因闪断, 消费处理失败, ACK返回失败等等问题。
 - 注意:
 - 重试间隔时间配置 ,默认每条消息最多重试 16 次
 - `messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h`
 - 超过重试次数人工补偿
 - 消费端去重
 - 一条消息无论重试多少次, 这些重试消息的 Message ID, key 不会改变。
 - 消费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,

第4集 RocketMQ4.X异步发送消息和回调实战



简介: 讲解使用RocketMQ异步发送消息

- 官方文档: <https://rocketmq.apache.org/docs/simple-example/>
- 核心:

```
producer.send(message, new SendCallback(){
    onSuccess(){}
    onException(){}
})
```

注意：官方例子：如果异步发送消息，调用`producer.shutdown()`后会失败

异步发送：不会重试，发送总次数等于1

第5集 RocketMQ OneWay发送消息及多种场景对比



简介：讲解使用RocketMQ发送oneway消息和使用场景，多种发送模式对比

- **SYNC** :
 - 应用场景：重要通知邮件、报名短信通知、营销短信系统等
- **ASYNC** : 异步
 - 应用场景：对RT时间敏感,可以支持更高的并发，回调成功触发相对应的业务，比如注册成功后通知积分系统发放优惠券
- **ONEWAY** : 无需要等待响应
 - 官方文档：<https://rocketmq.apache.org/docs/simple-example/>
 - 使用场景：主要是日志收集，适用于某些耗时非常短，但对可靠性要求并不高的场景，也就是LogServer, 只负责发送消息，不等待服务器回应且没有回调函数触发，即只发送请求不等待应答

汇总对比

发送方式	发送 TPS	发送结果反馈	可靠性
同步发送	快	有	不丢失
异步发送	快	有	不丢失
单向发送	最快	无	可能丢失

第6集 RocketMQ延迟消息实战和电商系统中应用



简介：讲解消息队列的延迟消息的使用，和在电商系统中应用场景

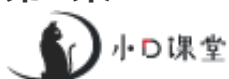
- 什么是延迟消息：

- Producer 将消息发送到消息队列 RocketMQ 服务端，但并不期望这条消息立马投递，而是推迟到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息，目前支持固定精度的消息
- 代码：rocketmq-store > MessageStoreConfig.java 属性 messageDelayLevel

```
▪ "1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h";
```

- 使用message.setDelayTimeLevel(xxx) //xxx是级别，1表示配置里面的第一个级别，2表示第二个级别
 - 定时消息：目前rocketmq开源版本还不支持，商业版本则有，两者使用场景类似
- 使用场景
 - 通过消息触发一些定时任务，比如在某一固定时间点向用户发送提醒消息
 - 消息生产和消费有时间窗口要求：比如在天猫电商交易中超时未支付关闭订单的场景，在订单创建时会发送一条 延时消息。这条消息将会在 30 分钟以后投递给消费者，消费者收到此消息后需要判断对应的订单是否已完成支付。如支付未完成，则关闭订单。如已完成支付则忽略

第7集 RocketMQ生产者之MessageQueueSelector实战



简介：生产消息使用MessageQueueSelector投递到Topic下指定的queue，

- 应用场景：顺序消息，分摊负载
- 默认topic下的queue数量是4，可以配置

```
//可以使用Jdk8的lambda表达式，只有一个方法需要被实现
producer.send(message, new MessageQueueSelector(){
    select(List<MessageQueue> mqs, Message msg, Object
arg){

        Integer queueNum = (Integer)arg;
        return mqs.get(queueNum);
    }
},0)
```

- 支持同步，异步发送指定的MessageQueue
- 选择的queue数量必须小于配置的，否则会出错

第8集 讲解顺序消息在电商和证券系统中应用场景



简介：基础介绍顺序消息和对应可以使用的场景，订单系统，

- 什么是顺序消息：消息的生产和消费顺序一致
 - **全局顺序**：topic下面全部消息都要有序(少用)
 - 性能要求不高，所有的消息严格按照 FIFO 原则进行消息发布和消费的场景，并行度成为消息系统的瓶颈, 吞吐量不够.
 - 在证券处理中，以人民币兑换美元为例子，在价格相同的情况下，先出价者优先处理，则可以通过全局顺序的方式按照 FIFO 的方式进行发布和消费
 - **局部顺序**：只要保证一组消息被顺序消费即可（RocketMQ使用）
 - 性能要求高
 - 电商的订单创建，同一个订单相关的创建订单消息、订单支付消息、订单退款消息、订单物流消息、订单交易成功消息 都会按照先后顺序来发布和消费
(阿里巴巴集团内部电商系统均使用局部顺序消息，既保证业务的顺序，同时又能保证业务的高性能)
- 顺序发布：对于指定的一个 Topic，客户端将按照一定的先后顺序发送消息
- 顺序消费：对于指定的一个 Topic，按照一定的先后顺序接收消息，即先发送的消息一定会先被客户端接收到。
- 注意：
 - 顺序消息暂不支持广播模式
 - 顺序消息不支持异步发送方式，否则将无法严格保证顺序

第9集 核心知识之RocketMQ顺序消息讲解



简介：讲解RocketMQ顺序消息的使用和讲解

- 生产端保证发送消息有序，且发送到同一个Topic的同个queue里面，RocketMQ的确是能保证FIFO的

- 例子：订单的顺序流程是：创建、付款、物流、完成，订单号相同的消息会被先后发送到同一个队列中，

根据MessageQueueSelector里面自定义策略，根据同个业务id放置到同个queue里面，如订单号取模运算再放到selector中，同一个模的值都会投递到同一条queue

```
public MessageQueue select(List<MessageQueue> mqs, Message msg, Object arg) {  
    //如果是订单号是字符串，则进行hash,得到一个hash值  
    Long id = (Long) arg;  
    long index = id % mqs.size();  
    return mqs.get((int)index);  
}
```

- 消费端要在保证消费同个topic里的同个队列，不应该用MessageListenerConcurrently，应该使用MessageListenerOrderly，自带单线程消费消息，不能再Consumer端再使用多线程去消费，消费端分配到的queue数量是固定的，集群消费会锁住当前正在消费的队列集合的消息，所以会保证顺序消费。
- 官方例子 <https://rocketmq.apache.org/docs/order-example/>

第10集 案例实战之RocketMQ顺序消息生产者投递



简介：讲解使用代码编写案例，进行RocketMQ顺序消息生产者投递

第11集 案例实战之RocketMQ顺序消息消费者消费实战

简介：讲解使用代码编写案例，进行RocketMQ顺序消息消费实战

- MessageListenerConcurrently
- MessageListenerOrderly
 - Consumer会平均分配queue的数量

- 并不是简单禁止并发处理，而是为每个Consumer Quene加个锁，消费每个消息前，需要获得这个消息所在的Queue的锁，这样同个时间，同个Queue的消息不被并发消费，但是不同Queue的消息可以并发处理

扩展思维：为什么高并发情况下ConcurrentHashMap比HashTable和HashMap更高效且线程安全？

提示：分段锁Segment



小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第七章 RocketMQ消费者核心配置和核心知识讲解

第1集 消息队列RocketMQ4.X消费者核心配置讲解



小D课堂

简介：消息队列RocketMQ4.X消费者核心配置讲解

- consumeFromWhere配置(某些情况失效：参考<https://blog.csdn.net/a417930422/article/details/83585397>)
 - CONSUME_FROM_FIRST_OFFSET：初次从消息队列头部开始消费，即历史消息（还储存在broker的）全部消费一遍，后续再启动接着上次消费的进度开始消费
 - CONSUME_FROM_LAST_OFFSET：默认策略，初次从该队列最尾开始消费，即跳过历史消息，后续再启动接着上次消费的进度开始消费
 - CONSUME_FROM_TIMESTAMP：从某个时间点开始消费，默认是半个小时以前，后续再启动接着上次消费的进度开始消费
- allocateMessageQueueStrategy:
 - 负载均衡策略算法，即消费者分配到queue的算法，默认值是AllocateMessageQueueAveragely即取模平均分配
- offsetStore：消息消费进度存储器 offsetStore 有两个策略：
 - LocalFileOffsetStore 和 RemoteBrokerOffsetStore 广播模式默认使用LocalFileOffsetStore 集群模式默认使用RemoteBrokerOffsetStore
- consumeThreadMin 最小消费线程池数量

- consumeThreadMax 最大消费线程池数量
- pullBatchSize: 消费者去broker拉取消息时，一次拉取多少条。可选配置
- consumeMessageBatchMaxSize: 单次消费时一次性消费多少条消息，批量消费接口才有用，可选配置
- messageModel : 消费者消费模式， CLUSTERING——默认是集群模式CLUSTERING
BROADCASTING——广播模式

第2集 集群和广播模式下RocketMQ消费端处理



简介：讲解集群模式下消费端消费消息流程

- Topic下队列的奇偶数会影响Consumer个数里面的消费数量
 - 如果是4个队列，8个消息，4个节点则会各消费2条，如果不对等，则负载均衡会分配不均，
 - 如果consumer实例的数量比message queue的总数量还多的话，多出来的consumer实例将无法分到queue，也就无法消费到消息，也就无法起到分摊负载的作用，所以需要控制让queue的总数量大于等于consumer的数量
- **集群模式(默认):**
 - Consumer实例平均分摊消费生产者发送的消息
 - 例子：订单消息，一般是只被消费一次
- **广播模式:**
 - 广播模式下消费消息：投递到Broker的消息会被每个Consumer进行消费，一条消息被多个Consumer消费，广播消费中ConsumerGroup暂时无用
 - 例子：群公告，每个人都需要消费这个消息
- 怎么切换模式：通过setMessageModel()

第3集 RocketMQ4.X里面的标签Tag实战和消息过滤原理

简介：讲解RocketMQ里面的Tag作用和消息过滤原理

- 一个Message只有一个Tag，tag是二级分类
 - 订单：数码类订单、食品类订单
- 过滤分为Broker端和Consumer端过滤
 - Broker端过滤，减少了无用的消息的进行网络传输，增加了broker的负担
 - Consumer端过滤，完全可以根据业务需求进行实习，但是增加了很多无用的消息传输

- 一般是监听 *, 或者指定 tag, || 运算, SLQ92, FilterServer等;
 - tag性能高, 逻辑简单
 - SQL92 性能差点, 支持复杂逻辑 (只支持PushConsumer中使用) MessageSelector.bySql
 - 语法: >, <=, IS NULL, AND, OR, NOT 等, sql where后续的语法即可(大部分)
- 注意: 消费者订阅关系要一致, 不然会消费混乱, 甚至消息丢失
 - 订阅关系一致: 订阅关系由 Topic和 Tag 组成, 同一个 group name, 订阅的 topic和tag 必须是一样的
- 在Broker 端进行MessageTag过滤, 遍历message queue存储的 message tag和 订阅传递的tag 的 hashcode不一样则跳过, 符合的则传输给Consumer, 在consumer queue存储的是对应的 hashcode, 对比也是通过hashcode对比; Consumer收到过滤消息后也会进行匹配操作, 但是是对比真实的message tag而不是hashcode
 - consume queue存储使用hashcode定长, 节约空间
 - 过滤中不访问commit log, 可以高效过滤
 - 如果存在hash冲突, Consumer端可以进行再次确认
- 如果想使用多个Tag, 可以使用sql表达式, 但是不建议, 单一职责, 多个队列

常见错误:

The broker does not support consumer to filter message by SQL92

解决: broker.conf 里面配置如下

enablePropertyFilter=true

备注, 修改之后要重启Broker

master节点配置: vim conf/2m-2s-async/broker-a.properties

slave节点配置: vim conf/2m-2s-async/broker-a-s.properties

第4集 PushConsumer、PullConsumer消费模式分析



简介: 讲解PushConsumer/PullConsumer消费消息模式分析

- Push和Pull优缺点分析
 - Push
 - 实时性高; 但增加服务端负载, 消费端能力不同, 如果Push推送过快, 消费端会出现很多问题

- Pull
 - 消费者从Server端拉取消息，主动权在消费者端，可控性好；但 间隔时间不好设置，间隔太短，则空请求，浪费资源；间隔时间太长，则消息不能及时处理
- 长轮询：Client请求Server端也就是Broker的时候，Broker会保持当前连接一段时间 默认是15s，如果这段时间内有消息到达，则立刻返回给Consumer.没消息的话 超过15s，则返回空，再进行重新请求；主动权在Consumer中，Broker即使有大量的消息 也不会主动提送Consumer， 缺点：服务端需要保持Consumer的请求，会占用资源，需要客户端连接数可控 否则会一堆连接
- PushConsumer本质是长轮训
 - 系统收到消息后自动处理消息和offset，如果有新的Consumer加入会自动做负载均衡，
 - 在broker端可以通过longPollingEnable=true来开启长轮询
 - 消费端代码：DefaultMQPushConsumerImpl->pullMessage->PullCallback
 - 服务端代码：broker.longpolling
 - 虽然是push，但是代码里面大量使用了pull，是因为使用长轮训方式达到Push效果，既有pull有的，又有Push的实时性
 - 优雅关闭：主要是释放资源和保存Offset, 调用shutdown()即可，参考 @PostConstruct、@PreDestroy
- PullConsumer需要自己维护Offset(参考官方例子)
 - 官方例子路径：org.apache.rocketmq.example.simple.PullConsumer
 - 获取MessageQueue遍历
 - 客户维护Offset,需用用户本地存储Offset，存储内存、磁盘、数据库等
 - 处理不同状态的消息 FOUND、NO_NEW_MSG、OFFSET_ILLRGL、NO_MATCHED_MSG、4种状态
 - 灵活性高可控性强，但是编码复杂度会高
 - 优雅关闭：主要是释放资源和保存Offset，需用程序自己保存好Offset，特别是异常处理的时候



小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第八章 高级篇幅之消息队列Offset和CommitLog讲解

第1集 源码解读RocketMQ消息偏移量Offset讲解

简介：讲解消息偏移量Offset

- 什么是offset
 - message queue是无限长的数组，一条消息进来下标就会涨1，下标就是offset，消息在某个MessageQueue里的位置，通过offset的值可以定位到这条消息，或者指示Consumer从这条消息开始向后处理
 - message queue中的maxOffset表示消息的最大offset, maxOffset并不是最新的那条消息的offset，而是最新消息的offset+1，minOffset则是现存在的最小offset。

fileReserveTime=48 默认消息存储48小时后，消费会被物理地从磁盘删除，message queue的min offset也就对应增长。所以比minOffset还要小的那些消息已经不在broker上了，就无法被消费
- 类型(父类是OffsetStore):
 - 本地文件类型
 - DefaultMQPushConsumer的BROADCASTING模式，各个Consumer没有互相干扰，使用LocalFileOffsetStore，把Offset存储在本地
 - Broker代存储类型
 - DefaultMQPushConsumer的CLUSTERING模式，由Broker端存储和控制Offset的值，使用RemoteBrokerOffsetStore
 - 阅读源码的正确姿势：
 - 先有思路，明白大体流程
 - 再看接口
 - 再看实现类
- 有什么用
 - 主要是记录消息的偏移量，有多个消费者进行消费
 - 集群模式下采用RemoteBrokerOffsetStore, broker控制offset的值
 - 广播模式下采用LocalFileOffsetStore, 消费端存储
- 建议采用pushConsumer，RocketMQ自动维护OffsetStore，如果用另外一种pullConsumer需要自己进行维护OffsetStore

第2集 RocketMQ消息存储CommitLog讲解

简介：讲解消息队列CommitLog分析

- 消息存储是由ConsumeQueue和CommitLog配合完成
 - ConsumeQueue: 是逻辑队列，CommitLog是真正存储消息文件的，存储的是指向物理存储的地址

Topic下的每个message queue都有对应的ConsumeQueue文件，内容也会被持久化到磁盘

默认地址：store/consumequeue/{topicName}/{queueid}/fileName

- 什么是CommitLog:
 - 消息文件的存储地址
 - 生成规则:
 - 每个文件的默认1G = 1024 * 1024 * 1024, commitlog的文件名fileName, 名字长度为20位, 左边补零, 剩余为起始偏移量; 比如00000000000000000000代表了第一个文件, 起始偏移量为0, 文件大小为1G=1 073 741 824Byte; 当这个文件满了, 第二个文件名字为00000000001073741824, 起始偏移量为1073741824, 消息存储的时候会顺序写入文件, 当文件满了则写入下一个文件
 - 判断消息存储在哪个CommitLog上
 - 例如 1073742827 为物理偏移量, 则其对应的相对偏移量为 1003 = 1073742827 - 1073741824, 并且该偏移量位于第二个 CommitLog。
- Broker里面一个Topic
 - 里面有多个MesssageQueue
 - 每个MessageQueue对应一个ConsumeQueue
 - ConsumeQueue里面记录的是消息在CommitLog里面的物理存储地址

第3集 高性能分析之ZeroCopy零拷贝技术讲解

简介: 讲解ZeroCopy零拷贝技术讲解和分析

- 高效原因
 - CommitLog顺序写, 存储了MessagBody、message key、tag等信息
 - ConsumeQueue随机读 + 操作系统的PageCache + 零拷贝技术ZeroCopy
 - 零拷贝技术

```
read(file, tmp_buf, len);
write(socket, tmp_buf, len);
```

- 例子: 将一个File读取并发送出去 (Linux有两个上下文, 内核态, 用户态)
 - File文件的经历了4次copy
 - 调用read, 将文件拷贝到了kernel内核态
 - CPU控制 kernel态的数据copy到用户态
 - 调用write时, user态下的内容会copy到内核态的socket的buffer中
 - 最后将内核态socket buffer的数据copy到网卡设备中传送
 - 缺点: 增加了上下文切换、浪费了2次无效拷贝(即步骤2和3)
 - ZeroCopy:

- 请求kernel直接把disk的数据传输给socket，而不是通过应用程序传输。Zero copy大大提高了应用程序的性能，减少不必要的内核缓冲区跟用户缓冲区间的拷贝，从而减少CPU的开销和减少了kernel和user模式的上下文切换，达到性能的提升
- 对应零拷贝技术有mmap及sendfile
 - mmap:小文件传输快
 - RocketMQ 选择这种方式，mmap+write 方式，小块数据传输，效果会比 sendfile 更好
 - sendfile:大文件传输比mmap快
- Java中的TransferTo()实现了Zero-Copy
- 应用：Kafka、Netty、RocketMQ等都采用了零拷贝技术



XD课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第九章 高级篇幅之RocketMQ4.x分布式事务消息

第1集 分布式事务消息介绍

简介：讲解什么是分布式事务消息

- 什么是分布式事务
 - 来源：单体应用—>拆分为分布式应用
 - 一个接口需要调用多个服务，且操作不同的数据库，数据一致性难保障，
- 常见解决方案
 - 2PC：两阶段提交, 基于XA协议
 - TCC：Try、Confirm、Cancel
 - 下单：
 - 事务消息最终一致性：
 - 更多...
- 框架
 - GTS -> 开源 Fescar

- 地址: <https://github.com/alibaba/fescar>
- LCN
 - 地址: <https://github.com/codingapi/tx-lcn>

第2集 新特性RocketMQ4.X分布式事务消息架构讲解



简介: 讲解RocketMQ分布式事务消息的总体架构

- RocketMQ事务消息:
 - RocketMQ 提供分布事务功能, 通过 RocketMQ 事务消息能达到分布式事务的最终一致
- 半消息Half Message:
 - 暂不能投递的消息(暂不能消费), Producer已经将消息成功发送到了Broker端, 但是服务端未收到生产者对该消息的二次确认, 此时该消息被标记成“暂不能投递”状态, 处于该种状态下的消息即半消息
- 消息回查:
 - 由于网络闪断、生产者应用重启等原因, 导致某条事务消息的二次确认丢失, 消息队列 RocketMQ 服务端通过扫描发现某条消息长期处于“半消息”时, 需要主动向消息生产者询问该消息的最终状态 (Commit 或是 Rollback), 该过程即消息回查。
- 整体交互流程



- Producer向broker端发送消息。
- 服务端将消息持久化成功之后, 向发送方 ACK 确认消息已经发送成功, 此时消息为半消息。
- 发送方开始执行本地事务逻辑。
- 发送方根据本地事务执行结果向服务端提交二次确认 (Commit 或是 Rollback), 服务端收到 Commit 状态则将半消息标记为可投递, 订阅方最终将收到该消息; 服务端收到 Rollback 状态则删除半消息, 订阅方将不会接受该消息
- 在断网或者是应用重启的特殊情况下, 上述步骤 4 提交的二次确认最终未到达服务端, 经过固定时间后服务端将对该消息发起消息回查
- 发送方收到消息回查后, 需要检查对应消息的本地事务执行的最终结果
- 发送方根据检查得到的本地事务的最终状态再次提交二次确认, 服务端仍按照步骤 4 对半消息进行操作
- RocketMQ事务消息的状态
 - COMMIT_MESSAGE: 提交事务消息, 消费者可以消费此消息
 - ROLLBACK_MESSAGE: 回滚事务消息, 消息会在broker中删除, 消费者不能消费
 - UNKNOWN: Broker需要回查确认消息的状态
- 关于事务消息的消费
 - 事务消息consumer端的消费方式和普通消息是一样的, RocketMQ能保证消息能被consumer

收到（消息重试等机制，最后也存在consumer消费失败的情况，这种情况出现的概率极低）。

第3集 RocketMQ分布式事务消息实战上集



简介：讲解RocketMQ分布式事务消息实战上集

- TransactionMQProducer基础介绍和使用
- 自定义线程池和消息生产者结合

```
//监听器 ,执行本地事务
TransactionListener transactionListener = new TransactionListenerImpl();

//创建事务消息发送者
TransactionMQProducer producer = new
TransactionMQProducer("unique_group_name");

//创建自定义线程池
//@param corePoolSize    池中所保存的核心线程数
//@param maximumPoolSize  池中允许的最大线程数
//@param keepActiveTime   非核心线程空闲等待新任务的最长时间
//@param timeunit         keepActiveTime参数的时间单位
//@param blockingqueue    任务队列
ExecutorService executorService = new ThreadPoolExecutor(2, 5, 100,
TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2000), new ThreadFactory()
{
    @Override
    public Thread newThread(Runnable r) {
        Thread thread = new Thread(r);
        thread.setName("client-transaction-msg-check-thread");
        return thread;
    }
});

//设置producer基本属性
producer.setNamesrvAddr(JmsConfig.NAME_SERVER);
producer.setExecutorService(executorService);
producer.setTransactionListener(transactionListener);
producer.start();
```


第4集 RocketMQ分布式事务消息实战下集

简介：讲解RocketMQ分布式事务消息实战下集

- TransactionListener使用
 - executeLocalTransaction 执行本地事务
 - checkLocalTransaction 回查消息，要么commit 要么rollback，reconsumeTimes不生效
- 注意点：TransactionMQProducer 的groupName要唯一，不能和普通的producer一样
- 本地访问路径：http://localhost:8081/api/v1/pay_cb?tag=xdclass222&otherParam=2

第5集 源码分析之分布式事务消息发送端拆解

简介：讲解RocketMQ分布式事务消息sendMessageInTransaction源码分析

 小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第十章 双主双从高性能RocketMQ服务搭建

第1集 双主双从高性能MQ服务架构讲解

简介：讲解怎样搭建高可用集群，避免单点故障问题

- 主从架构：Broker角色，Master提供读写，Slave只支持读
 - Consumer不用配置，当Master不可用或者繁忙的时候，Consumer会自动切换到Slave节点进行能读取
- 架构讲解，4台机器
 - 两个部署Broker-Master 和 NameServer

- 两个部署Broker-Slave 和 NameServer

第2集 RocketMQ双主双从搭建相关软件准备

简介：讲解RocketMQ搭建双主双从集群环境相关准备

- 4台机器, 2台部署NameServer, 4台都部署Broker, 双主双从 同步复制, 异步刷盘
- jdk、maven、rocketmq上传和安装
- 机器列表

```
server1 ssh root@192.168.159.133    部署nameServer  Broker-a
server2 ssh root@192.168.159.130    部署nameServer      Broker-a-s
server3 ssh root@192.168.159.131                                Broker-b
server4 ssh root@192.168.159.132                                Broker-
b-s
```

1、修改RocketMQ(启动内存配置, 4个机器都要修改, 其中runbroker.sh修改4个, runserver.sh修改2个)

```
vim runserver.sh
JAVA_OPT="${JAVA_OPT} -server -Xms528m -Xmx528m -Xmn256m -
XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"

vim runbroker.sh
JAVA_OPT="${JAVA_OPT} -server -Xms528m -Xmx528m -Xmn256m"

启动两个机器的 nameserver
nohup sh bin/mqnamesrv &

全路径
/usr/local/software/rocketmq/distribution/target/apache-rocketmq
```

2、编辑并启动rocketmq命令

broker-a主节点

```
nohup sh bin/mqbroker -c conf/2m-2s-sync/broker-a.properties &
```

```
namesrvAddr=192.168.159.133:9876;192.168.159.130:9876
```

```
brokerClusterName=XdcClassCluster
```

```
brokerName=broker-a
```

```
brokerId=0
```

```
deleteWhen=04
```

```
fileReservedTime=48
```

```
brokerRole=SYNC_MASTER
```

```
flushDiskType=ASYNC_FLUSH
```

```
defaultTopicQueueNums=4
```

```
#是否允许自动创建Topic，建议线下开启，线上关闭
```

```
autoCreateTopicEnable=true
```

```
#是否允许自动创建订阅组，建议线下开启，线上关闭
```

```
autoCreateSubscriptionGroup=false
```

```
#存储路径，根据需求进行配置绝对路径，默认是家目录下面
```

```
#storePathRootDir=
```

```
#storePathCommitLog
```

broker-a从节点

```
nohup sh bin/mqbroker -c conf/2m-2s-sync/broker-a-s.properties &
```

```
namesrvAddr=192.168.159.133:9876;192.168.159.130:9876
```

```
brokerClusterName=XdcClassCluster
```

```
brokerName=broker-a
```

```
brokerId=1
```

```
deleteWhen=04
```

```
fileReservedTime=48
```

```
brokerRole=SLAVE
```

```
flushDiskType=ASYNC_FLUSH
```

```
defaultTopicQueueNums=4
```

```
#是否允许自动创建Topic，建议线下开启，线上关闭
```

```
autoCreateTopicEnable=true
```

```
#是否允许自动创建订阅组，建议线下开启，线上关闭
```

```
autoCreateSubscriptionGroup=false
```

```
#存储路径，根据需求进行配置绝对路径，默认是家目录下面
```

```
#storePathRootDir=
```

```
#storePathCommitLog
```

broker-b主节点

```
nohup sh bin/mqbroker -c conf/2m-2s-sync/broker-b.properties &
```

```

namesrvAddr=192.168.159.133:9876;192.168.159.130:9876
brokerClusterName=XdclassCluster
brokerName=broker-b
brokerId=0
deleteWhen=04
fileReservedTime=48
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH

defaultTopicQueueNums=4
#是否允许自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=false

#存储路径，根据需求进行配置绝对路径，默认是家目录下面
#storePathRootDir=
#storePathCommitLog

```

```

broker-b从节点
nohup sh bin/mqbroker -c conf/2m-2s-sync/broker-b-s.properties &

namesrvAddr=192.168.159.133:9876;192.168.159.130:9876
brokerClusterName=XdclassCluster
brokerName=broker-b
brokerId=1
deleteWhen=04
fileReservedTime=48
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH

defaultTopicQueueNums=4
#是否允许自动创建Topic，建议线下开启，线上关闭
autoCreateTopicEnable=true
#是否允许自动创建订阅组，建议线下开启，线上关闭
autoCreateSubscriptionGroup=false

#存储路径，根据需求进行配置绝对路径，默认是家目录下面
#storePathRootDir=
#storePathCommitLog

```

参考命令

```
CentOS 6.5关闭防火墙
service iptables stop
```

```
centos7关闭防火墙
systemctl stop firewalld
systemctl stop firewalld.service
```

第3集 双主双从集群搭建和控制台配置讲解

简介：讲解双主双从搭建和控制台配置

注意：如果连接不了broker，日志提示连接的端口少2位，记得检查防火墙是否关闭

修正上节课的错误 **brokername**名称

- 使用管控台 安装在server1机器里面

修改事项

pom.xml 里面的rocketmq版本号

路径 /usr/local/software/rocketmq-externals-master/rocketmq-console/src/main/resources

application.properties里面的nameserver

增加 rocketmq.config.namesrvAddr=192.168.159.133:9876;192.168.159.130:9876

```
mvn install -Dmaven.test.skip=true
```

```
java -jar rocketmq-console-ng-1.0.0.jar
```

本地测试记得关闭防火墙(4个机器都要)

```
CentOS 6.5关闭防火墙
service iptables stop
```

```
centos7关闭防火墙
systemctl stop firewalld
systemctl stop firewalld.service
```

第4集 RocketMQ生产环境操作和推荐配置



简介：讲解生产环境RocketMQ的使用流程和推荐配置

- Topic创建线上禁止开启自动创建
- 一般是有专门的后台进行队列的CRUD，应用上线需要申请队列名称
- 生产环境推荐配置
 - NameServer配置多个不同机器多个节点
 - 多Master, 每个Master带有Slave
 - 主从设置为SYNC_MASTER同步双写
 - Producer用同步方式投递Broker
 - 刷盘策略为SYNC_FLUSH(性能好点则可以为ASYNC_FLUSH)
- 性能分析思路
 - CPU: top
 - 网卡: sar -n DEV 2 10、netstat -t、iperf3
 - 磁盘: iostat -x dm 1
 - JVM: jstack、jinfo、MAT



小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第十一章 消息队列面试专题

第1集 为什么使用消息队列，怎么选择技术



简介：讲解为什么消息队列，消息队列选型问题

- 异步
 - 例子：
- 解耦：
 - 例子：
- 削峰：
 - 例子：
- 缺点：
 - 系统可用性越低：外部依赖越多，依赖越多，出问题风险越大
 - 系统复杂性提高：需要考虑多种场景，比如消息重复消费，消息丢失
 - 需要更多的机器和人力：消息队列一般集群部署，而且需要运维和监控，例如topic申请等

消息队列选择问题：Apache ActiveMQ、Kafka、RabbitMQ、RocketMQ

- ActiveMQ: <http://activemq.apache.org/>
 - Apache出品，历史悠久，支持多种语言的客户端和协议，支持多种语言Java, .NET, C++等，基于JMS Provider的实现

缺点：吞吐量不高，多队列的时候性能下降，存在消息丢失的情况，比较少大规模使用
- Kafka: <http://kafka.apache.org/>
 - 是由Apache软件基金会开发的一个开源流处理平台，由Scala和Java编写。Kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理大规模的网站中的所有动作流数据(网页浏览，搜索和其他用户的行动)，副本集机制，实现数据冗余，保障数据尽量不丢失；支持多个生产者和消费者

缺点：不支持批量和广播消息，运维难度大，文档比较少，需要掌握Scala，二次开发难度大
- RabbitMQ: <http://www.rabbitmq.com/>
 - 是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、JMS、C、用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不错

缺点：使用Erlang开发，阅读和修改源码难度大
- RocketMQ: <http://rocketmq.apache.org/>
 - 阿里开源的一款的消息中间件，纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点，性能强劲(零拷贝技术)，支持海量堆积，支持指定次数和时间间隔的失败消息重发，支持consumer端tag过滤、延迟消息等，在阿里内部进行大规模使用，适合在电商，互联网金融等领域使用

缺点：部分实现不是按照标准JMS规范，有些系统要迁移或者引入队列需要修改代码

- 上述两个方式都可以，但是不能用于分布式锁，考虑原子问题，但是排重可以不考虑原子问题，数据量多需要设置过期时间
- 数据库去重表
- 某个字段使用Message的key做唯一索引

第3集 RocketMQ如何保证消息的可靠性传输

简介：讲解如何保证消息的可靠性，处理消息丢失的问题

- producer端
 - 不采用oneway发送，使用同步或者异步方式发送，做好重试，但是重试的Message key必须唯一
 - 投递的日志需要保存，关键字段，投递时间、投递状态、重试次数、请求体、响应体
- broker端
 - 双主双从架构，NameServer需要多节点
 - 同步双写、异步刷盘 (同步刷盘则可靠性更高，但是性能差点，根据业务选择)
- consumer端
 - 消息消费务必保留日志，即消息的元数据和消息体
 - 消费端务必做好幂等性处理
- 投递到broker端后
 - 机器断电重启：异步刷盘，消息丢失；同步刷盘消息不丢失
 - 硬件故障：可能存在丢失，看队列架构

第4集 消息发生大量堆积应该怎么处理

简介：如果消息大量堆积在broker里面，应该怎么处理

线上故障了，怎么处理

- 消息堆积了10小时，有几千万条消息待处理，现在怎么办？
- 修复consumer, 然后慢慢消费？也需要几小时才可以消费完成，新的消息怎么办？

正确的姿势

- 临时topic队列扩容，并提高消费者能力，但是如果增加Consumer数量，但是堆积的topic里面的message queue数量固定，过多的consumer不能分配到message queue
- 编写临时处理分发程序，从旧topic快速读取到临时新topic中，新topic的queue数量扩容多倍，然后再启动更多consumer进行在临时新的topic里消费

第5集 RocketMQ高性能的原因分析 小D课堂

简介:讲解RocketMQ高性能的原因分析，高可用架构

- MQ架构配置
 - 顺序写，随机读，零拷贝
 - 同步刷盘SYNC_FLUSH和异步刷盘ASYNC_FLUSH, 通过flushDiskType配置
 - 同步复制和异步复制，通过brokerRole配置，ASYNC_MASTER, SYNC_MASTER, SLAVE
 - 推荐同步复制(双写)，异步刷盘
- 发送端高可用
 - 双主双从架构：创建Topic对应的时候，MessageQueue创建在多个Broker上
即相同的Broker名称，不同的brokerid(即主从模式)；当一个Master不可用时，组内其他的Master仍然可用。
但是机器资源不足的时候，需要手工把slave转成master，目前不支持自动转换，可用shell处理
- 消费高可用
 - 主从架构：Broker角色，Master提供读写，Slave只支持读
 - Consumer不用配置，当Master不可用或者繁忙的时候，Consumer会自动切换到Slave节点进行能读取
- 提高消息的消费能力
 - 并行消费
 - 增加多个节点

- 增加单个Consumer的并行度，修改consumerThreadMin和consumerThreadMax
- 批量消费，设置Consumer的consumerMessageBatchMaxSize, 默认是1，如果为N，则消息多的时候，每次收到的消息为N条
- 择 Linux Ext4 文件系统，Ext4 文件系统删除 1G 大小的文件通常耗时小于 50ms，而 Ext3 文件系统耗时需要 1s，删除文件时磁盘IO 压力极大，会导致 IO 操作超时

第6集 消息队列常见面试题回答总结

简介:讲解RocketMQ常见面试题汇总

- 为什么使用消息队列
- 消息队列技术选择
- 如果保证消息队列高可用
- 如何保证消息传输的可靠性
- 如何消息的重复消费
- 如何保证消息的顺序消费
- 消息堆积怎么处理



小D课堂

愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

第十二章 RocketMQ高并发抗压实战课程总结

第1集 RocketMQ消息队列课程总结和后续规划

简介：总结RocketMQ课程和后续新的安排

- 掌握消息队列核心知识
- 可以在公司中实际使用RocketMQ，搭建高可用架构
- 掌握RocketMQ分布式事务知识

- 掌握RocketMQ高性能原因分析

综合项目实战，秒杀 或 拼团 或 聚合支付网关

技术选择：

- 框架：SpringBoot2.x + SpringCloud或者Dubbo + RocketMQ4.x + Redis5.x
- 部署：Nginx 负载均衡 + Docker 应用集群部署 + 阿里云CentOS7
- 基础环境：JDK8 + IDEA
- 数据库：Mysql 主从
- 性能测试：Jmeter5.x压测
- 监控报警：钉钉机器人 或者 Zabbix

小D课堂，愿景：让编程不在难学，让技术与生活更加有趣

相信我们，这个是可以让你学习更加轻松的平台，里面的课程绝对会让你技术不断提升

欢迎加小D讲师的微信：jack794666918

我们官方网站：<https://xdclass.net>

千人IT技术交流QQ群：718617859

重点来啦：免费赠送你干货文档大集合，包含前端，后端，测试，大数据，运维主流技术文档（持续更新）

<https://mp.weixin.qq.com/s/qYnjcDYGFDQorWmSfE7lpQ>