# 消息队列（Message Queue）

JMS的概念：

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。Java消息服务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持。

“消息队列”是在消息的传输过程中保存消息的容器。它具有低耦合、可靠投递、广播、流量控制、最终一致性等一系列功能，成为异步RPC的主要手段之一。当今市面上有很多主流的消息中间件，如老牌的ActiveMQ、RabbitMQ，炙手可热的Kafka（高吞吐，处理大数据），阿里巴巴自主开发RocketMQ（处理高并发）等。

可以消息中间件看成是邮局，有人要送信往邮局发送信封即可，不用管是否能送达，消息中间件会根据信件地址找到需要推送消息的地址将消息推送到指定地方。

MQ的使用场景：

1.异步业务场景

2.解耦

3.削峰

MQ的最终目的是提高系统并发量，减小数据库压力。

## MQ基本组成：

### Broker

消息服务器，作为server提供消息核心服务

### Producer

消息生产者，业务的发起方，负责生产消息传输给broker，

Rocket中producer的基本操作步骤

1. 创建消息生产者（DefaultProducer）
2. 设置namesrv的地址namesrvAddr
3. 开启producer producer.start()
4. 创建消息Message，指定消息中的参数
5. 发送消息
6. 关闭producer

//创建DefaultMQProducer

DefaultMQProducer defaultMQProducer = new DefaultMQProducer("steven\_group");

//设置namesrvAddr

defaultMQProducer.setNamesrvAddr(namesrvAddr);

try {

defaultMQProducer.start();

} catch (MQClientException e) {

e.printStackTrace();

}

Message message = null;

//创建消息

try {

message = new Message("topic-demo",//主题

"tags",//消息过滤标志

"keys\_1",//消息唯一值

"hello! world!".getBytes(RemotingHelper.DEFAULT\_CHARSET)//消息body

);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

//发送消息

try {

SendResult result = defaultMQProducer.send(message);

System.out.println(result);

} catch (MQClientException e) {

e.printStackTrace();

} catch (RemotingException e) {

e.printStackTrace();

} catch (MQBrokerException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

//关闭生产者

defaultMQProducer.shutdown();

执行结果：

SendResult [sendStatus=SEND\_OK, msgId=AC100D3D175814DAD5DC830BDB0A0000, offsetMsgId=C0A8AF6600002A9F0000000000000000, messageQueue=MessageQueue [topic=topic-demo, brokerName=broker-a, queueId=0], queueOffset=0]

//其中queueId表示当前消息存在哪个队列里

Consumer

消息消费者，业务的处理方，负责从broker获取消息并进行业务逻辑处理

消费者基本操作

1. 创建 消费者 DefaultMQPushConsumer
2. 设置namesrv地址
3. 设置subscribe，这里主要读取主题信息
4. 创建消息监听MessageListener
5. 开启consumer consumer.start()
6. 获取消息信息
7. 返回消息读取状态

// 创建 消费者 DefaultMQPushConsumer

DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("steven\_group");

// 设置namesrv地址

consumer.setNamesrvAddr(nameSrvAddr);

// 设置消息拉取上限

consumer.setConsumeMessageBatchMaxSize(10);

// 设置subscribe，这里主要读取主题信息

try {

consumer.subscribe("topic-demo",

"tags"

);

} catch (MQClientException e) {

e.printStackTrace();

}

// 创建消息监听MessageListener

consumer.setMessageListener(new MessageListenerConcurrently() {

@Override

public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> list, ConsumeConcurrentlyContext consumeConcurrentlyContext) {

// 获取消息信息

//迭代消息

for (MessageExt msg:list) {

System.out.println(msg.getTopic());

try {

System.out.println(new String(msg.getBody(), RemotingHelper.DEFAULT\_CHARSET));

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

return ConsumeConcurrentlyStatus.RECONSUME\_LATER;

}

System.out.println(msg.getTags());

}

// 返回消息读取状态

return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;

}

});

// 开启consumer

try {

consumer.start();

} catch (MQClientException e) {

e.printStackTrace();

}

### Topic

主题，发布订阅模式下的消息统一汇集地，不同生产者向topic发送消息，由MQ服务器分发到不同的订阅者，实现消息的广播

### Queue

队列，PTP模式下，特定生产者向特定queue发送消息，消费者订阅特定的queue完成指定消息的接收

将消息发送到指定队列，保证消息发送的顺序

### Message

消息体，根据不同通信协议定义的固定格式进行编码的数据包，来封装业务数据，实现消息的传输

## RocketMQ

推荐在github上下载rocketmq-console-ng练习使用

### 配置

#所属集群名字

brokerClusterName=rocketmq-cluster

#broker名字，注意此处不同的配置文件填写的不一样

brokerName=broker-a

#0 表示 Master，>0 表示 Slave

brokerId=0 #nameServer地址，分号分割

brokerIP1 = 阿里云IP

namesrvAddr=阿里云IP:9876;腾讯云IP:9876

#在发送消息时，自动创建服务器不存在的topic，默认创建的队列数

defaultTopicQueueNums=4

#是否允许 Broker 自动创建Topic，建议线下开启，线上关闭

autoCreateTopicEnable=true

#是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭

autoCreateSubscriptionGroup=true

#Broker 对外服务的监听端口

listenPort=10911

#删除文件时间点，默认凌晨 4点

deleteWhen=04

#文件保留时间，默认 48 小时

fileReservedTime=120

#commitLog每个文件的大小默认1G

mapedFileSizeCommitLog=1073741824

#ConsumeQueue每个文件默认存30W条，根据业务情况调整

mapedFileSizeConsumeQueue=300000

#destroyMapedFileIntervalForcibly=120000

#redeleteHangedFileInterval=120000

#检测物理文件磁盘空间

diskMaxUsedSpaceRatio=88

#存储路径

storePathRootDir=/usr/local/rocketmq-all-4.4.0-bin-release/store

#commitLog 存储路径

storePathCommitLog=/usr/local/rocketmq-all-4.4.0-bin-release/store/commitlog

#消费队列存储路径存储路径

storePathConsumeQueue=/usr/local/rocketmq-all-4.4.0-bin-release/store/consumequeue

#消息索引存储路径

storePathIndex=/usr/local/rocketmq-all-4.4.0-bin-release/store/index

#checkpoint 文件存储路径

storeCheckpoint=/usr/local/rocketmq-all-4.4.0-bin-release/store/checkpoint

#Broker 的角色

#- ASYNC\_MASTER 异步复制Master

#- SYNC\_MASTER 同步双写Master

#- SLAVE brokerRole=ASYNC\_MASTER

#刷盘方式

#- ASYNC\_FLUSH 异步刷盘

#- SYNC\_FLUSH 同步刷盘 flushDiskType=ASYNC\_FLUSH

#checkTransactionMessageEnable=false

#abort 文件存储路径

abortFile=/usr/javawork/apache-rocketmq/store/abort

#限制的消息大小

maxMessageSize=65536

#flushCommitLogLeastPages=4

#flushConsumeQueueLeastPages=2

#flushCommitLogThoroughInterval=10000

#flushConsumeQueueThoroughInterval=60000

关于配置文件，需要关注的点有：

#所属集群名字

brokerClusterName=rocketmq-cluster

#broker名字，注意此处不同的配置文件填写的不一样

brokerName=broker-a

#0 表示 Master，>0 表示 Slave

brokerId=0

brokerIP1 = 阿里云IP

#nameServer地址，分号分割

namesrvAddr= rocketmq-nameserver1 :9876; rocketmq-nameserver2 :9876

# 在发送消息时，自动创建服务器不存在的topic，默认创建的队列数

defaultTopicQueueNums=4

# 是否允许 Broker 自动创建Topic，建议线下开启，线上关闭

autoCreateTopicEnable=true

# 是否允许 Broker 自动创建订阅组，建议线下开启，线上关闭

autoCreateSubscriptionGroup=true

# 删除文件时间点，默认凌晨 4点

deleteWhen=04

#文件保留时间，默认 48 小时

fileReservedTime=120

#commitLog每个文件的大小默认1G

mapedFileSizeCommitLog=1073741824

#ConsumeQueue每个文件默认存30W条，根据业务情况调整

mapedFileSizeConsumeQueue=300000

### 启动：

先启动namesrv

nohup sh mqnamesrv &

再启动broker

nohup sh mqbroker -c /usr/local/rocketmq-all-4.4.0-bin-release/conf/2m-2s-async/broker-a.properties > /dev/null 2>&1 &

注意：如果连不上虚拟机上的mq，十有八九是防火墙的问题，首先service iptables stop，如果不行，再systemctl stop firewalld.service。

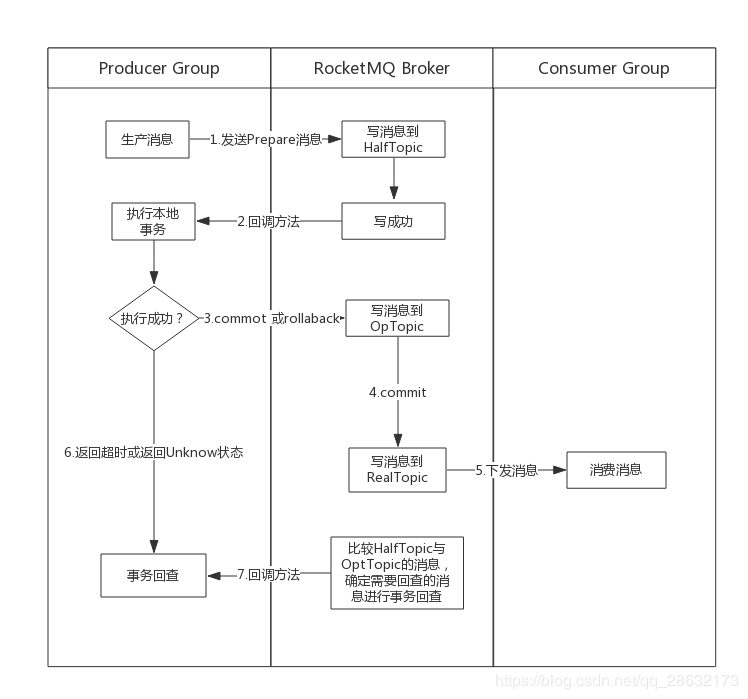
### 关闭

关闭broker服务 ：sh mqshutdown broker

关闭namesrv服务：sh mqshutdown namesrv

### RocketMQ对事务支持

原理图



生产者发送消息到broker,该消息是prepare消息，且事务消息的发送是同步发送的方式。

broker接收到消息后，会将该消息进行转换，所有的事务消息统一写入Half Topic，该Topic默认是RMQ\_SYS\_TRANS\_HALF\_TOPIC ,写入成功后会给生产者返回成功状态。

本地生产获取到该消息的事务Id，进行本地事务处理。

本地事务执行成功提交Commit，失败则提交Rollback,超时提交或提交Unknow状态则会触发broker的事务回查。

若提交了Commit或Rollback状态，Broker则会将该消息写入到Op Topic,该Topic默认是RMQ\_SYS\_TRANS\_OP\_HALF\_TOPIC，该Topic的作用主要记录已经Commit或Rollback的prepare消息，Broker利用Half Topic和Op Topic计算出需要回查的事务消息。如果是commit消息，broker还会将消息从Half取出来存储到真正的Topic里，从而消费者可以正常进行消费，如果是Rollback则不进行其他操作

如果本地事务执行超时或返回了Unknow状态，则broker会进行事务回查。若生产者执行本地事务超过6s则进行第一次事务回查，总共回查15次，后续回查间隔时间是60s，broker在每次回查时会将消息再在Half Topic写一次。回查次数和时间间隔都是可配置的。

执行事务回查时，生产者可以获取到事务Id，检查该事务在本地执行情况，返回状态同第一次执行本地事务一样。

从上述流程可以看到事务消息其实只是保证了生产者发送消息成功与本地执行事务的成功的一致性，消费者在消费事务消息时，broker处理事务消息的消费与普通消息是一样的，若消费不成功，则broker会重复投递该消息16次，若仍然不成功则需要人工介入。

事务消息的成功投递是需要经历三个Topic的，分别是：

Half Topic：用于记录所有的prepare消息

Op Half Topic：记录已经提交了状态的prepare消息

Real Topic：事务消息真正的Topic,在Commit后会才会将消息写入该Topic，从而进行消息的投递。

### 广播模式

在consumer中设置接收消息的类型为broadcasting

### 集群模式

Rocket的几种模式：

1. 单master模式
2. 多master模式 没有slave
3. 多master多slave模式 ，异步复制（异步刷盘，同步复制工作中建议使用这种模式，HA）
4. 多master多slave模式，同步双写

# 分布式缓存

Redis

1. 使用redis有哪些好处？

(1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)

(2) 支持丰富数据类型，支持string，list，set，sorted set，hash

(3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行

(4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

Redis支持的数据类型？

3. redis常见性能问题和解决方案：

(1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件

(2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次

(3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内

(4) 尽量避免在压力很大的主库上增加从库

(5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3…这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

String字符串：

格式: set key value

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个键最大能存储512MB。

Hash（哈希）

格式: hmset name key1 value1 key2 value2

Redis hash 是一个键值(key=>value)对集合。

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

List（列表）

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）

格式: lpush name value

在 key 对应 list 的头部添加字符串元素

格式: rpush name value

在 key 对应 list 的尾部添加字符串元素

格式: lrem name index

key 对应 list 中删除 count 个和 value 相同的元素

格式: llen name

返回 key 对应 list 的长度

Set（集合）

格式: sadd name value

Redis的Set是string类型的无序集合。

集合是通过哈希表实现的，所以添加，删除，查找的复杂度都是O(1)。

zset(sorted set：有序集合)

格式: zadd name score value

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。

不同的是每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大的排序。

zset的成员是唯一的,但分数(score)却可以重复。

什么是Redis持久化？Redis有哪几种持久化方式？优缺点是什么？

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。

Redis 提供了两种持久化方式:RDB（默认） 和AOF

RDB：

rdb是Redis DataBase缩写

功能核心函数rdbSave(生成RDB文件)和rdbLoad（从文件加载内存）两个函数

AOF:

Aof是Append-only file缩写

每当执行服务器(定时)任务或者函数时flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作

aof写入保存：

WRITE：根据条件，将 aof\_buf 中的缓存写入到 AOF 文件

SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

存储结构:

内容是redis通讯协议(RESP )格式的命令文本存储。

比较：

1、aof文件比rdb更新频率高，优先使用aof还原数据。

2、aof比rdb更安全也更大

3、rdb性能比aof好

4、如果两个都配了优先加载AOF

刚刚上面你有提到redis通讯协议(RESP )，能解释下什么是RESP？有什么特点？（可以看到很多面试其实都是连环炮，面试官其实在等着你回答到这个点，如果你答上了对你的评价就又加了一分）

RESP 是redis客户端和服务端之前使用的一种通讯协议；

RESP 的特点：实现简单、快速解析、可读性好

For Simple Strings the first byte of the reply is "+" 回复

For Errors the first byte of the reply is "-" 错误

For Integers the first byte of the reply is ":" 整数

For Bulk Strings the first byte of the reply is "$" 字符串

For Arrays the first byte of the reply is "\*" 数组

Redis 有哪些架构模式？讲讲各自的特点

单机版

特点：简单

问题：

1、内存容量有限 2、处理能力有限 3、无法高可用。

主从复制

Redis 的复制（replication）功能允许用户根据一个 Redis 服务器来创建任意多个该服务器的复制品，其中被复制的服务器为主服务器（master），而通过复制创建出来的服务器复制品则为从服务器（slave）。 只要主从服务器之间的网络连接正常，主从服务器两者会具有相同的数据，主服务器就会一直将发生在自己身上的数据更新同步 给从服务器，从而一直保证主从服务器的数据相同。

特点：

1、master/slave 角色

2、master/slave 数据相同

3、降低 master 读压力在转交从库

问题：

无法保证高可用

没有解决 master 写的压力

哨兵

Redis sentinel 是一个分布式系统中监控 redis 主从服务器，并在主服务器下线时自动进行故障转移。其中三个特性：

监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作。

特点：

1、保证高可用

2、监控各个节点

3、自动故障迁移

缺点：主从模式，切换需要时间丢数据

没有解决 master 写的压力

集群（proxy 型）：

Twemproxy 是一个 Twitter 开源的一个 redis 和 memcache 快速/轻量级代理服务器； Twemproxy 是一个快速的单线程代理程序，支持 Memcached ASCII 协议和 redis 协议。

特点：1、多种 hash 算法：MD5、CRC16、CRC32、CRC32a、hsieh、murmur、Jenkins

2、支持失败节点自动删除

3、后端 Sharding 分片逻辑对业务透明，业务方的读写方式和操作单个 Redis 一致

缺点：增加了新的 proxy，需要维护其高可用。

failover 逻辑需要自己实现，其本身不能支持故障的自动转移可扩展性差，进行扩缩容都需要手动干预

集群（直连型）：

从redis 3.0之后版本支持redis-cluster集群，Redis-Cluster采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

特点：

1、无中心架构（不存在哪个节点影响性能瓶颈），少了 proxy 层。

2、数据按照 slot 存储分布在多个节点，节点间数据共享，可动态调整数据分布。

3、可扩展性，可线性扩展到 1000 个节点，节点可动态添加或删除。

4、高可用性，部分节点不可用时，集群仍可用。通过增加 Slave 做备份数据副本

5、实现故障自动 failover，节点之间通过 gossip 协议交换状态信息，用投票机制完成 Slave到 Master 的角色提升。

缺点：

1、资源隔离性较差，容易出现相互影响的情况。

2、数据通过异步复制,不保证数据的强一致性

什么是一致性哈希算法？什么是哈希槽？

这两个问题篇幅过长 网上找了两个解锁的不错的文章

https://www.cnblogs.com/lpfuture/p/5796398.html

https://blog.csdn.net/z15732621582/article/details/79121213

Redis常用命令？

Keys pattern

\*表示区配所有

以bit开头的

查看Exists key是否存在

Set

设置 key 对应的值为 string 类型的 value。

setnx

设置 key 对应的值为 string 类型的 value。如果 key 已经存在，返回 0，nx 是 not exist 的意思。

删除某个key

第一次返回1 删除了 第二次返回0

Expire 设置过期时间（单位秒）

TTL查看剩下多少时间

返回负数则key失效，key不存在了

Setex

设置 key 对应的值为 string 类型的 value，并指定此键值对应的有效期。

Mset

一次设置多个 key 的值，成功返回 ok 表示所有的值都设置了，失败返回 0 表示没有任何值被设置。

Getset

设置 key 的值，并返回 key 的旧值。

Mget

一次获取多个 key 的值，如果对应 key 不存在，则对应返回 nil。

Incr

对 key 的值做加加操作,并返回新的值。注意 incr 一个不是 int 的 value 会返回错误，incr 一个不存在的 key，则设置 key 为 1

incrby

同 incr 类似，加指定值 ，key 不存在时候会设置 key，并认为原来的 value 是 0

Decr

对 key 的值做的是减减操作，decr 一个不存在 key，则设置 key 为-1

Decrby

同 decr，减指定值。

Append

给指定 key 的字符串值追加 value,返回新字符串值的长度。

Strlen

取指定 key 的 value 值的长度。

persist xxx(取消过期时间)

选择数据库（0-15库）

Select 0 //选择数据库

move age 1//把age 移动到1库

Randomkey随机返回一个key

Rename重命名

Type 返回数据类型

08

使用过Redis分布式锁么，它是怎么实现的？

先拿setnx来争抢锁，抢到之后，再用expire给锁加一个过期时间防止锁忘记了释放。

如果在setnx之后执行expire之前进程意外crash或者要重启维护了，那会怎么样？

set指令有非常复杂的参数，这个应该是可以同时把setnx和expire合成一条指令来用的！

09

使用过Redis做异步队列么，你是怎么用的？有什么缺点？

一般使用list结构作为队列，rpush生产消息，lpop消费消息。当lpop没有消息的时候，要适当sleep一会再重试。

缺点：

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如rabbitmq等。

能不能生产一次消费多次呢？

使用pub/sub主题订阅者模式，可以实现1:N的消息队列。

10

什么是缓存穿透？如何避免？什么是缓存雪崩？何如避免？

缓存穿透

一般的缓存系统，都是按照key去缓存查询，如果不存在对应的value，就应该去后端系统查找（比如DB）。一些恶意的请求会故意查询不存在的key,请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

如何避免？

1：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该key对应的数据insert了之后清理缓存。

2：对一定不存在的key进行过滤。可以把所有的可能存在的key放到一个大的Bitmap中，查询时通过该bitmap过滤。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

如何避免？

1：在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。

2：做二级缓存，A1为原始缓存，A2为拷贝缓存，A1失效时，可以访问A2，A1缓存失效时间设置为短期，A2设置为长期

3：不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。