0、彩蛋

先来看看大佬们的代码是怎么写的：

```
1 package org.apache.rocketmq.common.protocol.header;
```

```
2
3  /**
4   * Use short variable name to speed up FastJson deserialization process.
5   */
6  public class SendMessageRequestHeaderV2 implements CommandCustomHeader {
7      @CFNotNull
8      private String a; // producerGroup;
9      @CFNotNull
10     private String b; // topic;
11     @CFNotNull
12     private String c; // defaultTopic;
13     @CFNotNull
14     private Integer d; // defaultTopicQueueNums;
15     @CFNotNull
16     private Integer e; // queueId;
17     @CFNotNull
18     private Integer f; // sysFlag;
19     @CFNotNull
20     private Long g; // bornTimestamp;
21     @CFNotNull
22     private Integer h; // flag;
23     @CFNullable
24     private String i; // properties;
25     @CFNullable
26     private Integer j; // reconsumeTimes;
27     @CFNullable
28     private boolean k; // unitMode = false;
29     private Integer l; // consumeRetryTimes
30     @CFNullable
31     private boolean m; //batch
32 }
```

> 这些abcdefg你爱了吗？以后我就这么写变量名，以注释写真实的含义。谁在骂我我就贴给他顶级开源项目的代码，让他自己慢慢品。

1、说说你们公司线上生产环境用的是什么消息中间件?

见【2、多个mq如何选型？】

2、多个mq如何选型?

MQ	描述
RabbitMQ	erlang开发，对消息堆积的支持并不好，当大量消息积压的时候，会导致 RabbitMQ 的性能急剧下降。每秒钟可以处理几万到十几万条消息。
RocketMQ	java开发，面向互联网集群化功能丰富，对在线业务的响应时延做了很多的优化，大多数情况下可以做到毫秒级的响应，每秒钟大概能处理几十万条消息。
Kafka	Scala开发，面向日志功能丰富，性能最高。当你的业务场景中，每秒钟消息数量没有那么多的时候，Kafka 的时延反而会比较高。所以，Kafka 不太适合在线业务场景。
ActiveMQ	java开发，简单，稳定，性能不如前面三个。小型系统用也ok，但是不推荐。推荐用互联网主流的。

3、为什么要使用MQ?

因为项目比较大，做了分布式系统，所有远程服务调用请求都是**同步执行**经常出问题，所以引入了mq

作用	描述
解耦	系统耦合度降低，没有强依赖关系
异步	不需要同步执行的远程调用可以有效提高响应时间
削峰	请求达到峰值后，后端service还可以保持固定消费速率消费，不会被压垮

4、RocketMQ由哪些角色组成，每个角色作用和特点是什么?

角色	作用
Nameserver	无状态，动态列表；这也是和zookeeper的重要区别之一。zookeeper是有状态的。
Producer	消息生产者，负责发消息到Broker。
Broker	就是MQ本身，负责收发消息、持久化消息等。
Consumer	消息消费者，负责从Broker上拉取消息进行消费，消费完进行ack。

5、RocketMQ中的Topic和JMS的queue有什么区别?

queue就是来源于数据结构的FIFO队列。而Topic是个抽象的概念，每个Topic底层对应N个queue，而数据也真实存在queue上的。

6、RocketMQ Broker中的消息被消费后会立即删除吗?

不会，每条消息都会持久化到CommitLog中，每个Consumer连接到Broker后会维持消费进度信息，当有消息消费后只是当前Consumer的消费进度（CommitLog的offset）更新了。

追问：那么消息会堆积吗？什么时候清理过期消息？

4.6版本默认48小时后会删除不再使用的CommitLog文件

- 检查这个文件最后访问时间
- 判断是否大于过期时间
- 指定时间删除，默认凌晨4点

源码如下：

```
/**
 * {@link org.apache.rocketmq.store.DefaultMessageStore.CleanCommitLogService#isTimeToDelete()}
 */
private boolean isTimeToDelete() {
    // when = "04";
    String when = DefaultMessageStore.this.getMessageStoreConfig().getDeleteWhen();
    // 是04点，就返回true
    if (UtilAll.isItTimeToDo(when)) {
        return true;
    }
    // 不是04点，返回false
    return false;
}

/**
 * {@link org.apache.rocketmq.store.DefaultMessageStore.CleanCommitLogService#deleteExpiredFiles()}
 */
private void deleteExpiredFiles() {
    // isTimeToDelete()这个方法是判断是不是凌晨四点，是的话就执行删除逻辑。
    if (isTimeToDelete()) {
        // 默认是72，但是broker配置文件默认改成了48，所以新版本都是48。
        long fileReservedTime = 48 * 60 * 60 * 1000;
        deleteCount = DefaultMessageStore.this.commitLog.deleteExpiredFile(72 * 60 * 60 * 1000, xx, xx, xx);
    }
}

/**
 * {@link org.apache.rocketmq.store.CommitLog#deleteExpiredFile()}
 */
public int deleteExpiredFile(xxx) {
    // 这个方法的主逻辑就是遍历查找最后更改时间+过期时间，小于当前系统时间的话就删了（也就是小于48小时）。
    return this.mappedFileQueue.deleteExpiredFileByTime(72 * 60 * 60 * 1000, xx, xx, xx);
}
```

7、RocketMQ消费模式有几种?

消费模型由Consumer决定，消费维度为Topic。

- 集群消费

> 1.一条消息只会被同Group中的一个Consumer消费
>
> 2.多个Group同时消费一个Topic时，每个Group都会有一个Consumer消费到数据

- 广播消费

> 消息将对一 个Consumer Group 下的各个 Consumer 实例都消费一遍。即即使这些 Consumer 属于同一个Consumer Group ，消息也会被 Consumer Group 中的每个 Consumer 都消费一次。

8、消费消息是push还是pull?

RocketMQ没有真正意义的push，都是pull，虽然有push类，但实际底层实现采用的是**长轮询机制**，即拉取方式

> broker端属性 longPollingEnable 标记是否开启长轮询。默认开启

源码如下：

```
// {@link org.apache.rocketmq.client.impl.consumer.DefaultMQPushConsumerImpl#pullMessage()}
// 看到没，这是一只披着羊皮的狼，名字叫PushConsumerImpl，实际干的确是pull的活。

// 拉取消息，结果放到pullCallback里
this.pullAPIWrapper.pullKernelImpl(pullCallback);
```

追问：为什么要主动拉取消息而不使用事件监听方式?

事件驱动方式是建立好长连接，由事件（发送数据）的方式来实时推送。

如果broker主动推送消息的话有可能push速度快，消费速度慢的情况，那么就会造成消息在consumer端堆积过多，同时又不能被其他consumer消费的情况。而pull的方式可以根据当前自身情况来pull，不会造成过多的压力而造成瓶颈。所以采取了pull的方式。

9、broker如何处理拉取请求的?

Consumer首次请求Broker

- Broker中是否有符合条件的消息
- 有 ->
    - 响应Consumer
    - 等待下次Consumer的请求
- 没有
    - 挂起consumer的请求，即不断开连接，也不返回数据
    - 使用consumer的offset，
        - DefaultMessageStore#ReputMessageService#run方法
            - 每隔1ms检查commitLog中是否有新消息，有的话写入到pullRequestTable

        - 当有新消息的时候返回请求
    - PullRequestHoldService 来Hold连接，每个5s执行一次检查pullRequestTable有没有消息，有的话立即推送

10、RocketMQ如何做负载均衡?

通过Topic在多Broker中分布式存储实现。

producer端

发送端指定message queue发送消息到相应的broker，来达到写入时的负载均衡

- 提升写入吞吐量，当多个producer同时向一个broker写入数据的时候，性能会下降
- 消息分布在多broker中，为负载消费做准备

默认策略是随机选择：

- producer维护一个index
- 每次取节点会自增
- index向所有broker个数取余
- 自带容错策略

其他实现：

- SelectMessageQueueByHash
    - hash的是传入的args
- SelectMessageQueueByRandom
- SelectMessageQueueByMachineRoom 没有实现

也可以自定义实现**MessageQueueSelector**接口中的select方法

```
MessageQueue select(final List<MessageQueue> mqs, final Message msg, final
Object arg);
```

consumer端

采用的是平均分配算法来进行负载均衡。

其他负载均衡算法

平均分配策略(默认)(AllocateMessageQueueAveragely) 环形分配策略(AllocateMessageQueueAveragelyByCircle) 手动配置分配策略(AllocateMessageQueueByConfig) 机房分配策略(AllocateMessageQueueByMachineRoom) 一致性哈希分配策略(AllocateMessageQueueConsistentHash) 靠近机房策略(AllocateMachineRoomNearby)

追问：当消费负载均衡consumer和queue不对等的时候会发生什么?

Consumer和queue会优先平均分配，如果Consumer少于queue的个数，则会存在部分Consumer消费多个queue的情况，如果Consumer等于queue的个数，那就是一个Consumer消费一个queue，如果Consumer个数大于queue的个数，那么会有部分Consumer空余出来，白白的浪费了。

11、消息重复消费

影响消息正常发送和消费的**重要原因是网络的不确定性。**

引起重复消费的原因

- ACK

正常情况下在consumer真正消费完消息后应该发送ack，通知broker该消息已正常消费，从queue中剔除

当ack因为网络原因无法发送到broker，broker会认为词条消息没有被消费，此后会开启消息重投机制把消息再次投递到consumer

- 消费模式

在CLUSTERING模式下，消息在broker中会保证相同group的consumer消费一次，但是针对不同group的consumer会推送多次

解决方案

- 数据库表

处理消息前，使用消息主键在表中带有约束的字段中insert

- Map

单机时可以使用map *ConcurrentHashMap* -> putIfAbsent guava cache

- Redis

分布式锁搞起来。

12、如何让RocketMQ保证消息的顺序消费

> 你们线上业务用消息中间件的时候，是否需要保证消息的顺序性?
>
> 如果不需要保证消息顺序，为什么不需要?假如我有一个场景要保证消息的顺序，你们应该如何保证?

首先多个queue只能保证单个queue里的顺序，queue是典型的FIFO，天然顺序。多个queue同时消费是无法绝对保证消息的有序性的。所以总结如下：

同一topic，同一个QUEUE，发消息的时候一个线程去发送消息，消费的时候 一个线程去消费一个queue里的消息。

追问：怎么保证消息发到同一个queue?

Rocket MQ给我们提供了MessageQueueSelector接口，可以自己重写里面的接口，实现自己的算法，举个最简单的例子：判断`i % 2 == 0`，那就都放到queue1里，否则放到queue2里。

```
for (int i = 0; i < 5; i++) {
    Message message = new Message("orderTopic", ("hello!" + i).getBytes());
    producer.send(
        // 要发的那条消息
        message,
        // queue 选择器 ，向 topic中的哪个queue去写消息
        new MessageQueueSelector() {
            // 手动 选择一个queue
            @Override
            public MessageQueue select(
                // 当前topic 里面包含的所有queue
                List<MessageQueue> mqs,
                // 具体要发的那条消息
```

```
                Message msg,
                // 对应到 send（） 里的 args，也就是2000前面的那个0
                Object arg) {
                // 向固定的一个queue里写消息，比如这里就是向第一个queue里写消息
                if (Integer.parseInt(arg.toString()) % 2 == 0) {
                    return mqs.get(0);
                } else {
                    return mqs.get(1);
                }
            }
        },
        // 自定义参数：0
        // 2000代表2000毫秒超时时间
        i, 2000);
}
```

13、RocketMQ如何保证消息不丢失

首先在如下三个部分都可能会出现丢失消息的情况：

- Producer端
- Broker端
- Consumer端

13.1、Producer端如何保证消息不丢失

- 采取send()同步发消息，发送结果是同步感知的。
- 发送失败后可以重试，设置重试次数。默认3次。

> producer.setRetryTimesWhenSendFailed(10);

- 集群部署，比如发送失败了的原因可能是当前Broker宕机了，重试的时候会发送到其他Broker上。

13.2、Broker端如何保证消息不丢失

- 修改刷盘策略为同步刷盘。默认情况下是异步刷盘的。

> flushDiskType = SYNC_FLUSH

- 集群部署，主从模式，高可用。

13.3、Consumer端如何保证消息不丢失

- 完全消费正常后在进行手动ack确认。

14、rocketMQ的消息堆积如何处理

> 下游消费系统如果宕机了，导致几百万条消息在消息中间件里积压，此时怎么处理?
>
> 你们线上是否遇到过消息积压的生产故障?如果没遇到过，你考虑一下如何应对?

首先要找到是什么原因导致的消息堆积，是Producer太多了，Consumer太少了导致的还是说其他情况，总之先定位问题。

然后看下消息消费速度是否正常，正常的话，可以通过上线更多consumer临时解决消息堆积问题

追问：如果Consumer和Queue不对等，上线了多台也在短时间内无法消费完堆积的消息怎么办?

- 准备一个临时的topic
- queue的数量是堆积的几倍
- queue分布到多Broker中
- 上线一台Consumer做消息的搬运工，把原来Topic中的消息挪到新的Topic里，不做业务逻辑处理，只是挪过去
- 上线N台Consumer同时消费临时Topic中的数据
- 改bug
- 恢复原来的Consumer，继续消费之前的Topic

RocketMQ中的消息只会在commitLog被删除的时候才会消失，不会超时。也就是说未被消费的消息不会存在超时删除这情况。

追问：堆积的消息会不会进死信队列?

不会，消息在消费失败后会进入重试队列（%RETRY%+ConsumerGroup），16次（默认16次）才会进入死信队列（%DLQ%+ConsumerGroup）。

源码如下：

```
public class SubscriptionGroupConfig {
    private int retryMaxTimes = 16;
}

// {@link org.apache.rocketmq.broker.processor.SendMessageProcessor#asyncConsumerSendMsgBack}
// maxReconsumeTimes = 16
int maxReconsumeTimes = subscriptionGroupConfig.getRetryMaxTimes();
// 如果重试次数大于等于16，则创建死信队列
if (msgExt.getReconsumeTimes() >= maxReconsumeTimes || delayLevel < 0) {
    // MixAll.getDLQTopic()就是给原有groupname拼上DLQ，死信队列
    newTopic = MixAll.getDLQTopic(requestHeader.getGroup());
    // 创建死信队列
    topicConfig = this.brokerController.getTopicConfigManager().createTopicInSendMessageBackMethod(xxx)
}
```

扩展：每次重试的时间间隔：

```
public class MessageStoreConfig {
    // 每隔如下时间会进行重试，到最后一次时间重试失败的话就进入死信队列了。
    private String messageDelayLevel = "1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h";
}
```

> 看到这个源码你可能蒙蔽了，这不是18个时间间隔嘛。怎么是16次？继续看下面代码，我TM也懵了。
>
> ```
> /**
> ```

```
 * {@link org.apache.rocketmq.client.impl.consumer.DefaultMQPushConsumerImpl#sendMessageBack()}
 *
 * sendMessageBack()这个方法是消费失败后会请求他，意思是把消息重新放到队列，进行重试。
 */
public void sendMessageBack(MessageExt msg, int delayLevel, final String brokerName) {
    Message newMsg = new Message();
    // ！！！我TM，真相了，3 + xxx。他是从第三个开始的。也就是舍弃了前两个时间间隔，18 - 2 = 16。也就是说第一次重试是在10s，第二次30s。
    // TMD！！！
    // TMD！！！
    // TMD！！！
    newMsg.setDelayTimeLevel(3 + msg.getReconsumeTimes());
    this.mQClientFactory.getDefaultMQProducer().send(newMsg);
 }
```

15、RocketMQ在分布式事务支持这块机制的底层原理?

> 你们用的是RocketMQ?RocketMQ很大的一个特点是对分布式事务的支持，你说说他在分布式事务支持这块机制的底层原理?

分布式系统中的事务可以使用TCC（Try、Confirm、Cancel）、2pc来解决分布式系统中的消息原子性

RocketMQ 4.3+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

RocketMQ实现方式：

Half Message： 预处理消息，当broker收到此类消息后，会存储到RMQ_SYS_TRANS_HALF_TOPIC的消息消费队列中

检查事务状态： Broker会开启一个定时任务，消费RMQ_SYS_TRANS_HALF_TOPIC队列中的消息，每次执行任务会向消息发送者确认事务执行状态（提交、回滚、未知），如果是未知，Broker会定时去回调在重新检查。

超时： 如果超过回查次数，默认回滚消息。

也就是他并未真正进入Topic的queue，而是用了临时queue来放所谓的half message，等提交事务后才会真正的将half message转移到topic下的queue。

16、如果让你来动手实现一个分布式消息中间件，整体架构你会如何设计实现?

我个人觉得从以下几个点回答吧：

- 需要考虑能快速扩容、天然支持集群
- 持久化的姿势
- 高可用性
- 数据0丢失的考虑
- 服务端部署简单、client端使用简单

17、看过RocketMQ 的源码没有。如果看过，说说你对RocketMQ 源码的理解?

要真让我说，我会吐槽蛮烂的，首先没任何注释，可能是之前阿里巴巴写了中文注释，捐赠给apache后，apache觉得中文注释不能留，自己又懒得写英文注释，就都给删了。里面比较典型的设计模式有单例、工厂、策略、门面模式。单例工厂无处不在，策略印象深刻比如发消息和消费消息的时候queue的负载均衡就是N个策略算法类，有随机、hash等，这也是能够快速扩容天然支持集群的必要原因之一。持久化做的也比较完善，采取的CommitLog来落盘，同步异步两种方式。

18、高吞吐量下如何优化生产者和消费者的性能?

开发

- 同一group下，多机部署，并行消费
- 单个Consumer提高消费线程个数
- 批量消费
    - 消息批量拉取
    - 业务逻辑批量处理

运维

- 网卡调优
- jvm调优
- 多线程与cpu调优
- Page Cache

19、再说说RocketMQ 是如何保证数据的高容错性的?

- 在不开启容错的情况下，轮询队列进行发送，如果失败了，重试的时候过滤失败的Broker
- 如果开启了容错策略，会通过RocketMQ的预测机制来预测一个Broker是否可用
- 如果上次失败的Broker可用那么还是会选择该Broker的队列
- 如果上述情况失败，则随机选择一个进行发送
- 在发送消息的时候会记录一下调用的时间与是否报错，根据该时间去预测broker的可用时间

> 其实就是send消息的时候queue的选择。源码在如下：
>
> `org.apache.rocketmq.client.latency.MQFaultStrategy#selectOneMessageQueue()`

20、任何一台Broker突然宕机了怎么办?

Broker主从架构以及多副本策略。Master收到消息后会同步给Slave，这样一条消息就不止一份了，Master宕机了还有slave中的消息可用，保证了MQ的可靠性和高可用性。而且Rocket MQ4.5.0开始就支持了Dlegder模式，基于raft的，做到了真正意义的HA。

21、Broker把自己的信息注册到哪个NameServer上?

这么问明显在坑你，因为Broker会向所有的NameServer上注册自己的信息，而不是某一个，是每一个，全部！

22、RocketMQ为什么自研nameserver而不用zk?

- RocketMQ只需要一个轻量级的维护元数据信息的组件，引入zk的话维护成本变高还强依赖了一个其他的中间件。
- RocketMQ追求的是AP而不是CP，也就是需要高可用。

23、RocketMq的工作流程是怎样的

1）首先启动NameServer。NameServer启动后监听端口，等待Broker、Producer以及Consumer连上来

2）启动Broker。启动之后，会跟所有的NameServer建立并保持一个长连接，定时发送心跳包。心跳包中包含当前Broker信息(ip、port等)、Topic信息以及Borker与Topic的映射关系

3）创建Topic。创建时需要指定该Topic要存储在哪些Broker上，也可以在发送消息时自动创建Topic

4）Producer发送消息。启动时先跟NameServer集群中的其中一台建立长连接，并从NameServer中获取当前发送的Topic所在的Broker；然后从队列列表中轮询选择一个队列，与队列所在的Broker建立长连接，进行消息的发送

5）Consumer消费消息。跟其中一台NameServer建立长连接，获取当前订阅Topic存在哪些Broker上，然后直接跟Broker建立连接通道，进行消息的消费

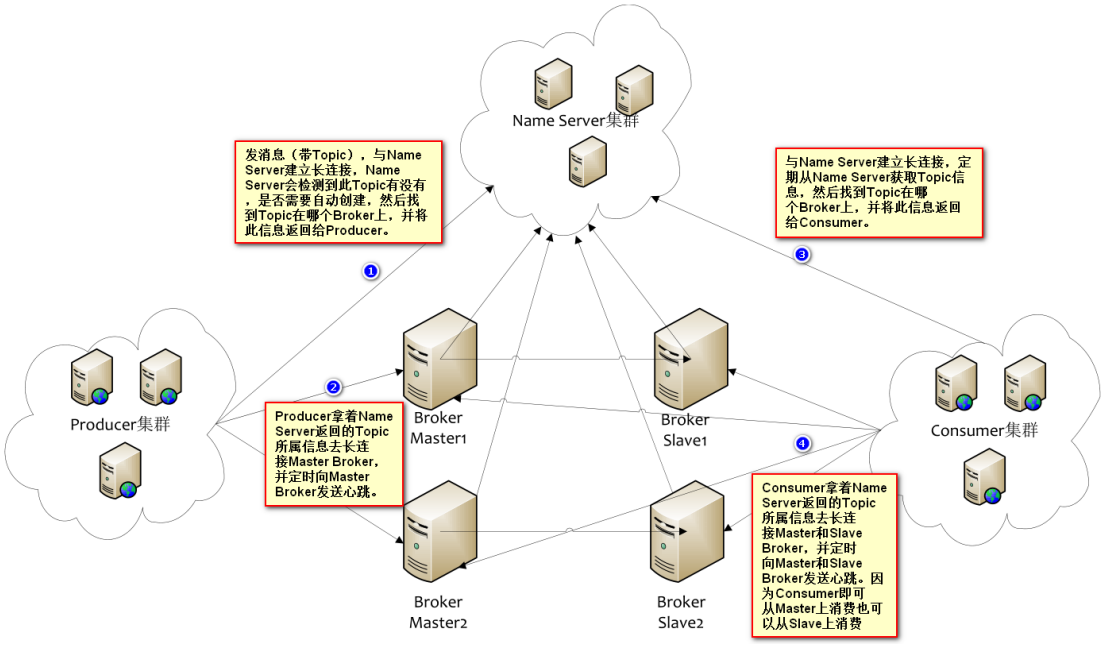
24、RocketMq性能比较高的原因?

RocketMq采用文件系统存储消息，采用顺序写的方式写入消息，使用零拷贝发送消息，这三者的结合极大地保证了RocketMq的性能。

25、RocketMQ是如何实现定时消息的?

定时消息是指消息发到Broker后，不能立刻被Consumer消费，要到特定的时间点或者等待特定的时间后才能被消费。 其实定时消息实现原理比较简单，如果一个topic对应的消息在发送端被设置为定时消息，那么会将该消息先存放在topic为SCHEDULE_TOPIC_XXXX的消息队列中，并将原始消息的信息存放在commitLog文件中，由于topic为SCHEDULE_TOPIC_XXXX，所以该消息不会被立即消息，然后通过定时扫描的方式，将到达延迟时间的消息，转换为正确的消息，发送到相应的队列进行消费。

26、RocketMq的部署架构了解吗?

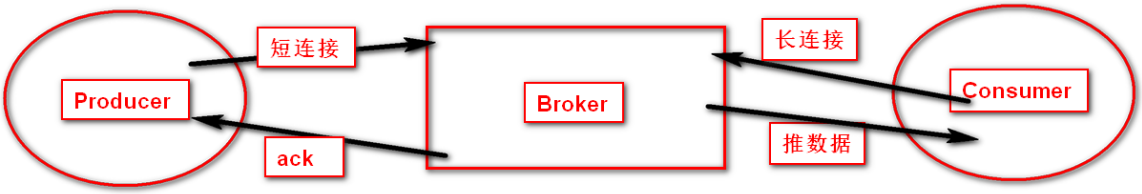


27、RocketMQ有哪几种部署类型？分别有什么特点?

- 单Master

也称1M。简单来说就是只部署一台Broker（MQ本身）作为主节点提供服务。

这种方式风险最大，一旦Broker重启或者宕机时，会导致整个服务不可用，不建议线上环境使用此方式。



- 多Master

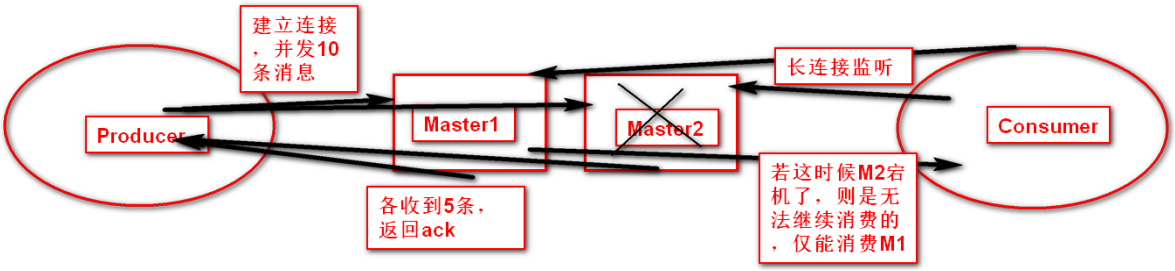
也称NM（多主）模式。简单来说就是多个Broker都作为主节点（可读可写）去提供服务。无Slave，全是Master。

优点：配置简单，单个Master宕机或者重启维护对应用无影响，在磁盘配置为RAID10（不懂的Google了解下就行了）时，即使机器宕机不可恢复的情况下，由于RAID10磁盘非常可靠，消息也不会丢（异步刷盘会丢失少量，同步刷盘一条不丢，下面会详细说到），性能是最高的。

缺点：单台机器宕机期间，这台机器上未被消费的消息在机器恢复之前不可订阅，消息实时性会受到影响。

例如：

一个Producer的A主题要发送10条消息（一个主题的消息可以发送到不同Master上）到M1和M2两台Broker上，比如M1和M2各五条，当发送完M1和M2的时候恰巧M2宕机了，这时候Consumer是无法继续消费M2上的消息的，所以会消息实时性会受到影响。只能等到M2重新恢复后才可继续消费。数据不会丢失。



- 多Master多Slave，异步复制

也称多Master多Slave，异步复制的方式。简单来说就是N台Master，每个Master又有N台（一般1台）Slave进行主从。形成NMNS。

每个Master配置N（一般是1）个Slave，有多对Master-Slave，HA采用异步复制方式，主备短暂消息延迟（毫秒级别的延迟）。

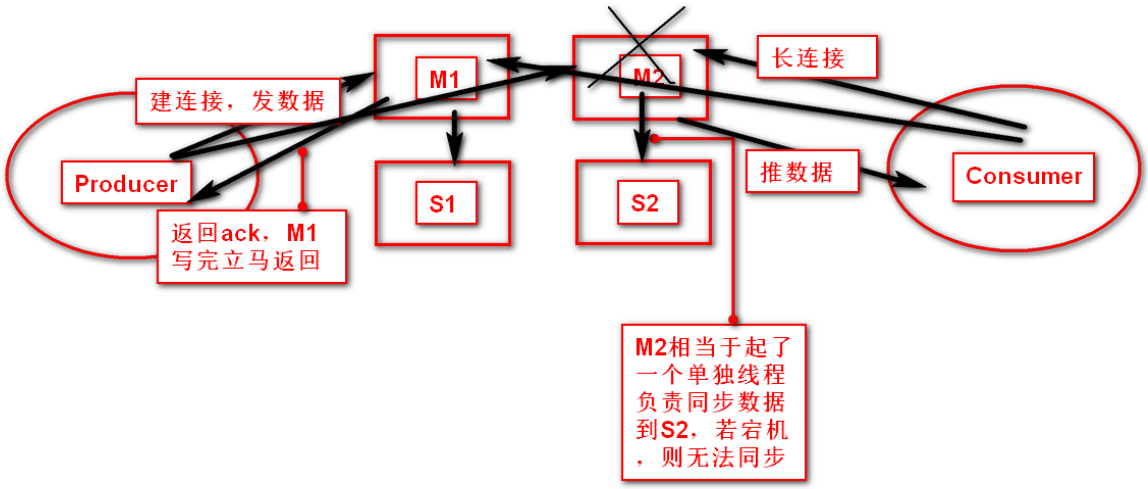
优点：即使磁盘损坏，丢失的消息非常少**（但是会丢失）**，且消息实时性不会受到影响，因为Master宕机后，消费者仍然可以从Slave上进行消费，此过程对应用完全透明，不需要人工干预，性能同多Master模式几乎一样。

缺点：Master宕机，磁盘损坏情况，会丢失少量消息。

目前主宕机后，备机不能自动切换为主节点。(4.3.1版本)

为什么会丢失少量消息?

因为异步复制原理是Producer发送消息到Broker，这时候Broker有Master和Slave，当Master确认将消息存储成功后就会立刻给Producer返回一个ack，而并不会等到Slave也存储成功后才会发送ack，也就说是这时候Slave的异步复制过去的，若Master存储完毕，返回ack了，突然之间宕机了（毫秒级别的，但是也有可能发生呀），并且磁盘损坏了（无法恢复了）。这时候数据就丢失了。



- 多Master多Slave，同步双写

也称多Master多Slave，同步双写的方式。简单来说就是N台Master，每个Master又有N台（一般1台）Slave进行主从。形成NMNS。

每个Master配置N（一般是1）个Slave，有多对Master-Slave，HA采用同步双写的方式，**主备都写成功，向应用返回ack确认。**

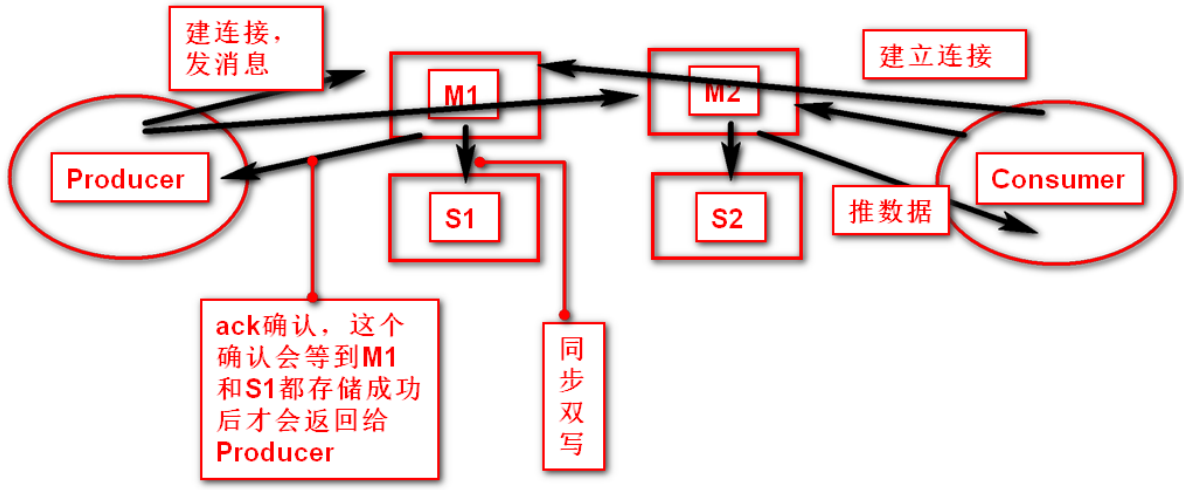
优点：数据与服务都没有单点，Master宕机情况下，消息无延迟，服务可用性与数据可用性都非常高。**数据不会丢失。**

缺点：性能比异步复制模式略低，大约低10%左右，发送单个消息的RT会略高。

目前主宕机后，备机不能自动切换为主节点。(4.3.1版本)

为什么不会丢失少量消息?

因为同步双写原理是Producer发送消息到Broker，这时候Broker有Master和Slave，当Master确认将消息存储成功并且等到Slave也同步存储成功后才会发送ack确认给Producer，所以数据不会丢。



28、RocketMQ如何保证高可用性?

- 集群化部署NameServer
- 集群化部署多broker

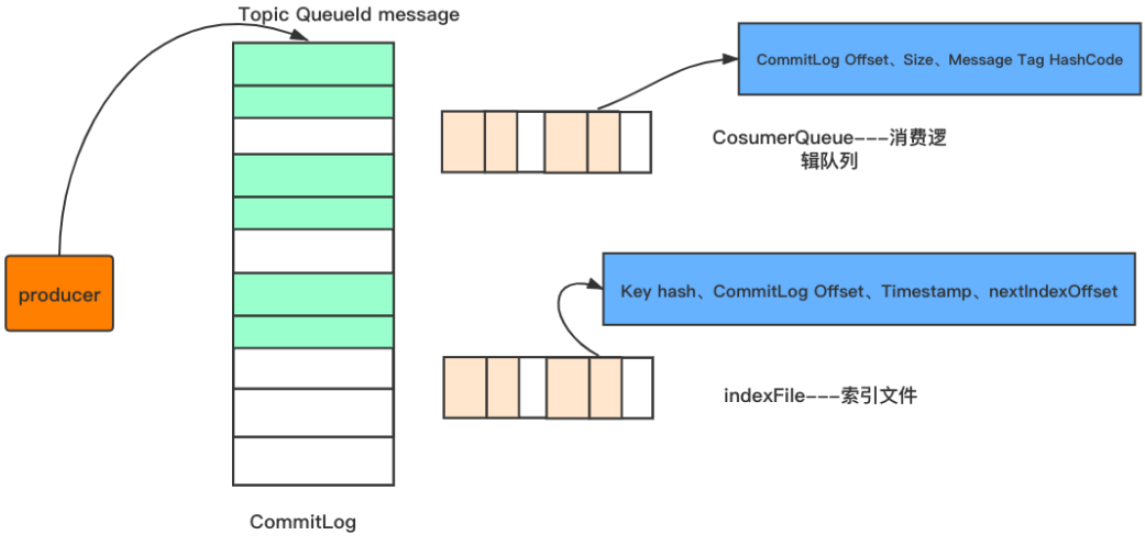
RocketMQ 4.5之后支持了一种叫做Dledger机制，基于Raft协议实现的一个机制。基于Dledger实现RocketMQ高可用自动切换。RocketMQ 4.5版本之前需要手动改配置进行切换。

29、RocketMQ的存储机制了解吗?

RocketMq采用文件系统存储消息，并采用顺序写写入消息，使用零拷贝发送消息，极大得保证了RocketMq的性能。

30、RocketMQ的存储结构是怎样的?

如图所示，消息生产者发送消息到broker，都是会按照顺序存储在CommitLog文件中，每个commitLog文件的大小为1G



CommitLog：存储所有的消息元数据，包括Topic、QueueId以及message

CosumerQueue：消费逻辑队列：存储消息在CommitLog的offset

IndexFile：索引文件：存储消息的key和时间戳等信息，使得RocketMq可以采用key和时间区间来查询消息

也就是说，rocketMq将消息均存储在 CommitLog 中，并分别提供了CosumerQueue和IndexFile两个索引，来快速检索消息

31、如果Broker宕了，NameServer是怎么感知到的?

Broker会定时（30s）向NameServer发送心跳

然后 NameServer会定时（10s）运行一个任务，去检查一下各个Broker的最近一次心跳时间，如果某个Broker超过120s都没发送心跳了，那么就认为这个Broker已经挂掉了。

32、Master Broker突然挂了，这样会怎么样?

RocketMQ 4.5版本之前，用Slave Broker同步数据，尽量保证数据不丢失，但是一旦Master故障了，Slave是没法自动切换成Master的。

所以在这种情况下，如果Master Broker宕机了，这时就得手动做一些运维操作，把Slave Broker重新修改一些配置，重启机器给调整为Master Broker，这是有点麻烦的，而且会导致中间一段时间不可用。

RocketMQ 4.5之后支持了一种叫做Dledger机制，基于Raft协议实现的一个机制。基于Dledger实现RocketMQ高可用自动切换。

我们可以让一个Master Broker对应多个Slave Broker， 一旦 Master Broker 宕机了，在多个 Slave 中通过 Dledger 技术 将一个 Slave Broker 选为新的 Master Broker 对外提供服务。在生产环境中可以是用Dledger机制实现自动故障切换，只要10秒或者几十秒的时间就可以完成