#### 为什么使用消息队列？

1. 面试题

为什么使用消息队列？消息队列有什么优点和缺点？Kafka activemq rabbitmq rocketmq都有什么区别以及适合哪些场景？

1. 面试官心理分析

* 第一、知道不知道你们系统里为什么要用消息队列这个东西？
* 第二、既然用来额，那么知不知道有什么好处？
* 第三、既然用来MQ，有没有做过调研？

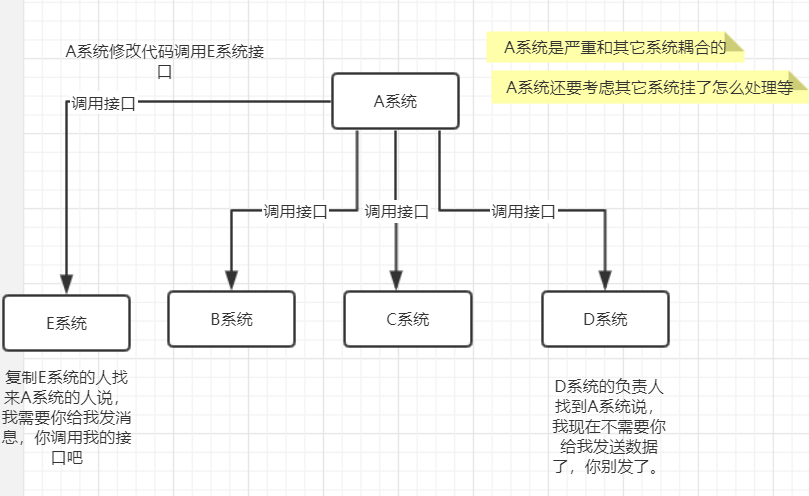
1. 题目剖析

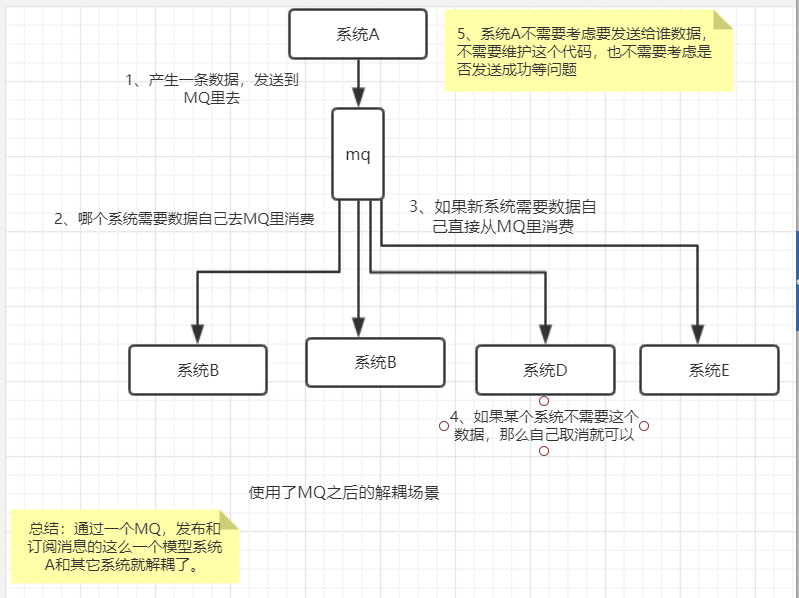
1)期望回答：有什么业务场景，这个业务场景有什么技术挑战，如果不用MQ可能会很麻烦，使用了MQ之后给自己带来了什么好处。

先说消息队列的常见使用场景，比较核心的有：解耦、异步、削峰。

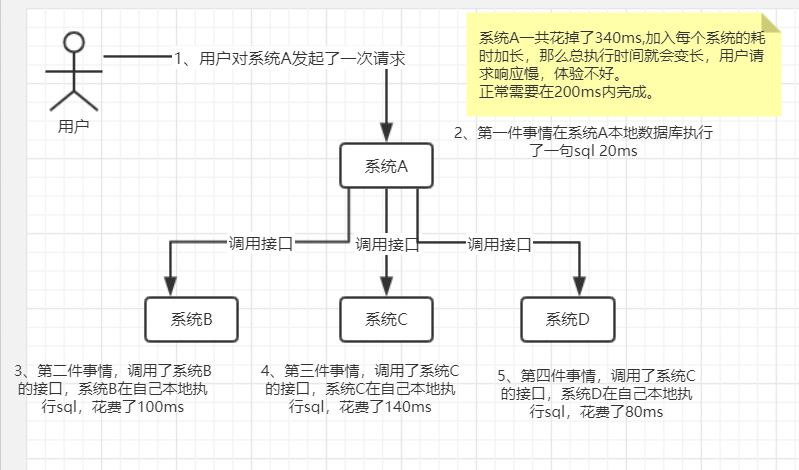
* 解耦：A系统发送数据到BCD三个系统，接口调用发送，那如果E系统也要这个数据呢？那如果C系统现在不需要了呢？现在A系统又要发送第二种数据了呢？如果A系统必须时时刻刻考虑BCDE系统挂了如果处理？要不要重发？要不要把消息存起来？

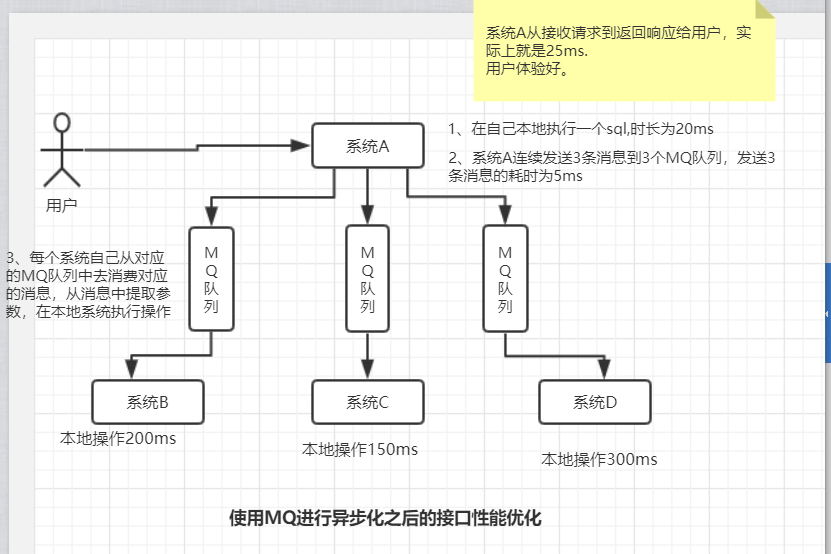
技巧：考虑一下自己的系统中是否有类似的场景，就是一个系统或者模块，调用了多个系统或者模块，相互之间的调用很复杂，维护起来很麻烦，但是这个其实不需要直接同步调用接口的，如果用MQ异步解耦也是可以的，就可以考虑在系统中运用MQ去进行系统的解耦。



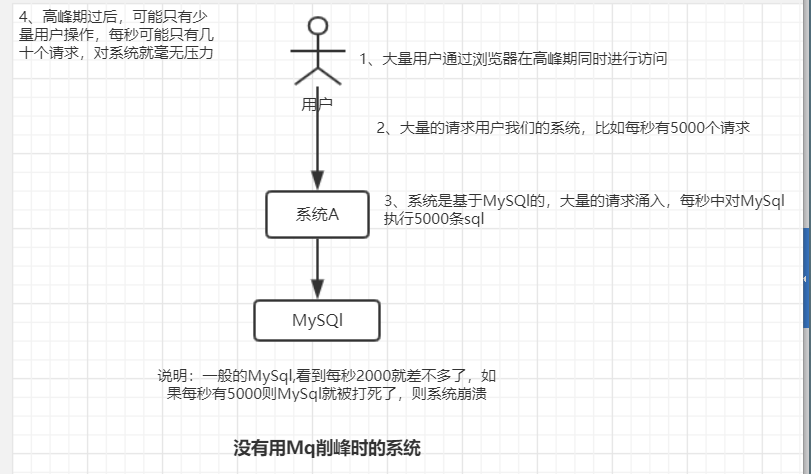


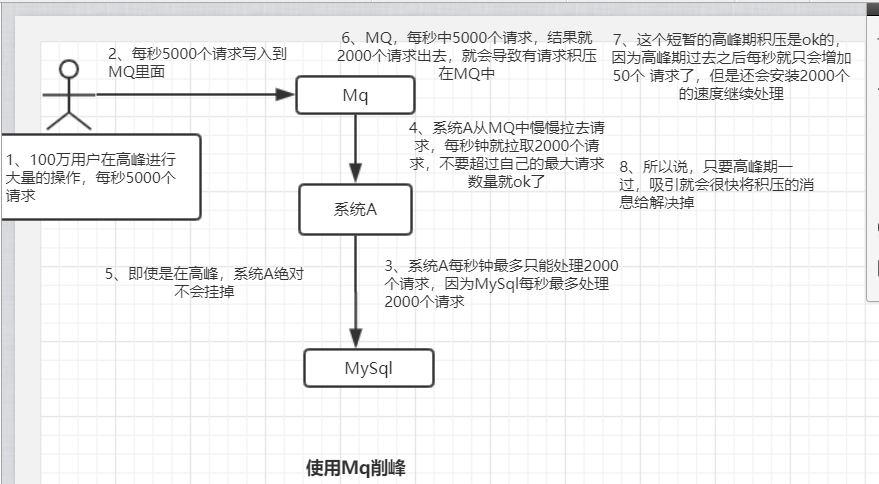
* 异步：A系统接收一个请求，需要在自己本地写库，还需要在BCD三个系统写库，自己本地3ms，BCD三个系统分别要300ms、450ms、200ms。最终总延时将会是953ms。用户体验差。





* 削峰：A系统搞活动，平时正常并发量也就每秒100个，搞活动的时候会爆发到10000个，如果超出了系统的最大处理能力，系统就会崩溃。





1. 消息队列有什么优点和缺点

优点：解耦、异步、削峰

缺点：

系统可用性降低：系统引入的外部依赖过多，越容易挂掉，本来只有A系统调用BCD三个接口就好了，4个系统好好的，引入了MQ，万一MQ挂了怎么办。

系统复杂性提高：硬生生加了MQ进来，如何保证消息没有重复消费？如果处理消息丢失的情况？怎么保证消息的顺序性？

一致性问题：A系统处理完直接返回成功了，用户以为是整个请求成功了，但是如果BCD系统中有一个每次处理成功，怎么办？

1. Kafka、activeMq、rabbitMq、rocketMq有什么优点和缺点

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 特性 | ActiveMq | RabbitMq | RocketMq | Kafaka |
| 单机吞吐量 | 万级，吞吐量比RocketMq和Kafka低一个数量级 | 万级，吞吐量比RocketMq和Kafka低一个数量级 | 10万级，RocketMq也是可以支持高吞吐量的一种MQ | 10万级，kafka最大的优点就是吞吐量高。  一般配合大数据的系统来进行实时数据计算，日志采集等场景。 |
| Topic数量对吞吐量的影响 |  |  | Topic可以大到几百，几千个的级别，吞吐量会有较小幅度的下降。  这是RocketMQ的一大优势，在同等机器下，可以支持大量的topic | Topic从几十个到几百个的时候，吞吐量会大幅下降，  所以在同等机器下，kafka尽量保证topic数量不要过多，如果要支撑大规模topicm需要增加更多的机器资源， |
| 时效性 | ms级 | 微秒级，这是rabbitmq的一大特点，延迟是最低的 | Ms级 | 延迟在ms级以内 |
| 可用性 | 高，基于主从架构实现高可用 | 高，基于主从架构实现高可用 | 非常高，分布式架构 | 非常高，kafka是分布式的，一个数据多个副本，少数机器宕机不会丢失数据，不会导致不可用。 |
| 消息可靠性 | 有较低的概率丢失数据 |  | 经过参数优化配置，可以做到0丢失 | 经过参数优化配置，可以做到0丢失。 |
| 功能支持 | MQ领域的功能及其完备 | 基于erlang语言开发，所以并发性能能力很强，性能机器好，延时很低 | MQ功能极为完善，还是分布式的，好扩展 | 功能较为简单，主要支持简单的MQ功能，在大数据领域的实时计算以及日志采集被大规模使用，是事实上的标准。 |
| 优劣势总结 | 非常成熟，功能强大，在业内大量的公司以及项目中都有应用。  偶尔会有较低概率丢失消息，而且现在社区以及国内应用越来越少，官方社区对5.x维护越来越少，而且确实主要是基于解耦和异步来的，较少在大规模吞吐量的场景中使用 | Erlang语言看法，性能好，延时很低，而且开源提供的管理界面非常棒用起来很好用，社区维护比较活跃，在国内近几年用rabbit也较多一些，  缺点是吞吐量小一些，这是因为他做的实现机制比较重，而且erlang开发，比较难维护，需要会使用erlang进行定制源码开发。  吞吐量万级，MQ功能完善。 | 接口简单易用，而且毕竟在阿里大规模使用过，  日处理消息百亿之多，可以做到大规模吞吐，性能也非常好，分布式系统扩展方便社区维护也还可以，可靠性和可用性都是ok的，还可以支持大规模的topic数量，支持复杂的mq业务场景，  而且一个很大的优势就是是基于java的，比较容易掌控。  社区活跃度也还可以，接口文档相对来说简单一些，接口不是按照标准的JMS开发的，可能需要修改大量的代码。  如果阿里抛弃了这个技术，社区就可能不会再更新，需要自己维护。 | Kafka的特点其实很明显就是仅仅提供很少的核心功能，但是提供超高的吞吐量，ms级的延迟，极高的可用性以及可靠性，而且分布式可以任意扩展。  同时kafka最好是支撑较少的topic数量即可，保证其超高吞吐量。  而且kafka唯一的一点劣势是有可能消息重复消费对数据的准确性有影响。 |

#### 如何保证消息队列的高可用性？

1. 面试题

如何保证消息队列的高可用性？

1. 面试官心理分析

针对MQ的那些缺点如何解决？

1. 面试题剖析

针对你所使用的来回答问题

1. RabbitMq的高可用性

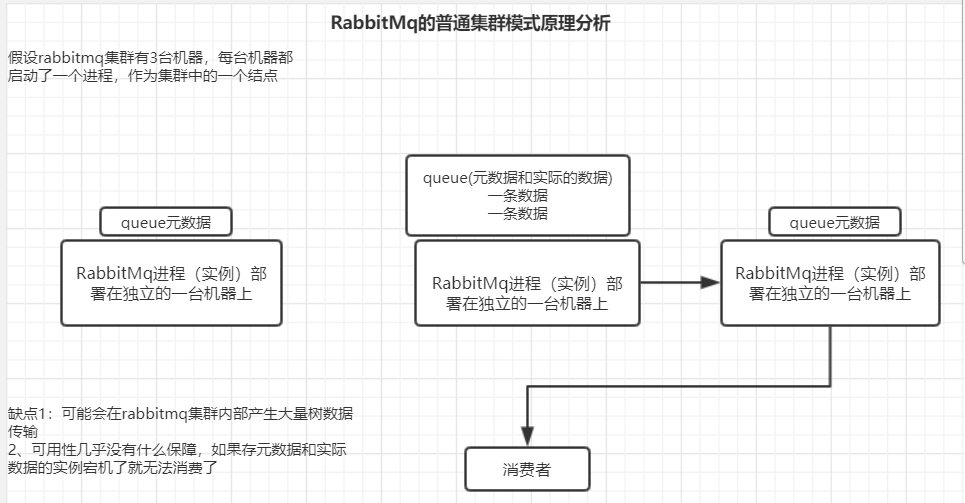
RabbitMQ是比较有代表性的，因为是基于主从做高可用，我们以他为例子来讲解第一种MQ的高可用如何实现。

Rabbitmq有三种模式：单机模式、普通集群模式、镜像集群模式。

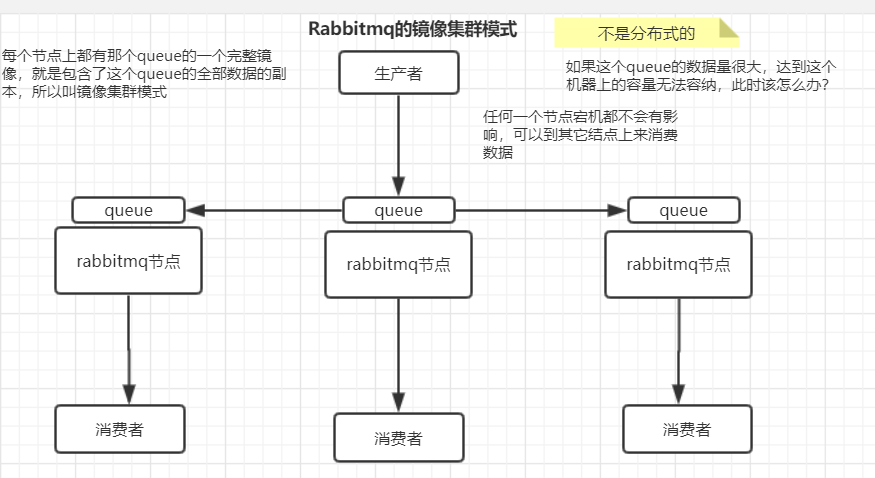
单机模式：

就是demo级别的，一般就是你本地用一下，生产上没人会使用单机模式

普通集群模式：多台基取上启动多个rabbitmq实例，每个机器启动一个。但是你创建的queue,只会放在一个rabbitmq上，但是每个实例都同步queue的数据。完了你消费的时候，实际上如果连接到了另外一个实例，那么那个实例会从queue所在实例上拉取数据过来。这种方式确实很麻烦，也不怎么好，没做到所谓的分布式，就是个普通的集群，因为这导致你要么消费者每次随机连接一个实例然后拉取数据，要么固定连接那个queue所在实例消费数据，前者有数据拉去的开销，后者导致单实例性能瓶颈。而且如果那个放queue的实例宕机了，会导致接下来其他实例就无法从那个实例拉起，如果你开启了消息持久化，让rabbitmq落地存储消息的话，消息不一定会丢，得等这个实例恢复了，然后才可以继续从这个queue拉取数据。

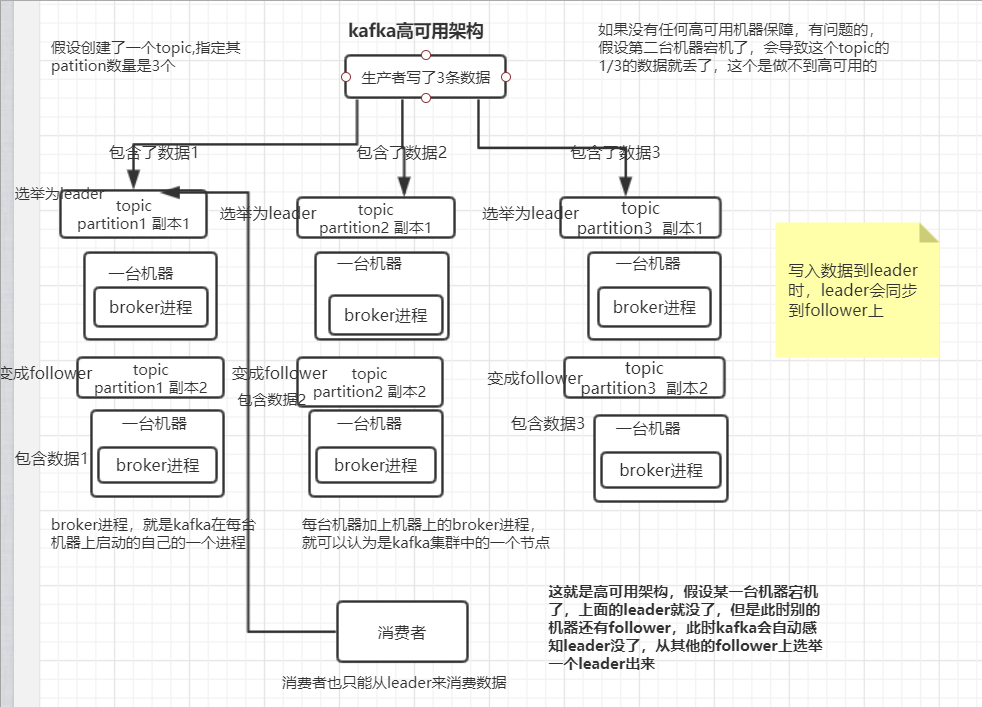


镜像集群模式：这种才是所谓的ranbbitmq的高可用模式，跟普通集群模式不一样的是，你创建的queue,无论元数据还是queue里的消息都会存在于多个实例上，然后每次你写消息到queue的时候，都会自动把消息到多个实例的queue里进行消息同步。这样的话，好处就是你任何一个机器宕机了，没事，别的机器都可以用。坏处在于，第一，这个性能开销也太大了，消息同步所有机器，导致网络带宽压力和消耗很重。第二，这样就没有扩展性了，如果某个queue负载很重，你加机器，新增的机器也包含了这个queue的所有数据，并没有办法来扩展你的queue。如何开启这个镜像模式？在rabbitmq的管理控制台新增一个策略，这个策略就是镜像集群模式的策略，指定的时候可以要求数据同步到所有的结点，也可以要求同步到知道数量的结点，然后你再次创建queue的时候，应用这个策略，就会自动将数据同步到其它的结点上去了。



1. Kafka的高可用性

Kafka0.8以后，提供了HA机制，就是replica副本机制，每个patition的数据都会同步到其它机器上，形成自己的多个replica副本，然后所有replica会选举一个leader出来，那么生成和消费都跟这个leader打交道，然后其它replica就是follower，写的时候leader会负责把数据同步到所哟follower上，读的时候就之前读leader上数据即可。只能读写leader?很简单，要是你可以随意读写每个follower，那么就要care数据一致性的问题，系统复杂性太高，很容易出问题，kafka会均匀的将一个partition的所有replica分布在不同的机器上，这样才可以提高容错性。这样就可以高可用了，因为如果某个broker宕机了，那么broker上的partition在其他机器上都有副本的，如果这上面有某个partition的leader ,那么次吃会重新选举一个新的leader出来，大家继续读写那个新的leader即可，这就是所谓的高可用性了。写数据的时候，生产者就写leader,然后leader将数据落地写本地磁盘，接着其他follower自己主动从leader来pull数据，一旦所有follower同步数据了，就会发送ack给leader,leader收到所有ack之后就会返回写成功的消息给生产者。消费的时候，只会从leader去读，但是只有一个消息已经被所有follower都同步成功返回ack得时候，这个消息才会被消费者读到。



1. 面试题

如何保证消息不被重复消费？

1. 面试官心理分析

消费就需要考虑会不会重复消费，能不能保证避免？

1. 面试题剖析

Rabbitmq、rocketmq、kafka都有可能会出现重复消费的问题，因为这个不是mq自己保证的，是给你保证的，以kafka为例来说。

Kafka有个offset的概念，就是每个消息写进去，都有一个offset,代表他的序号，然后consumer消费了数据之后，每隔一段时间，会把自己消费国的消息的offset提交以下，代表已经消费过了，下次我要是重启什么的就继续从上次消费到的offset来继续消费。

但是总有意外比如你直接kill进程有可能导致消费了但是offset没提交，就会导致重启之后再消费一次。

保证幂等性的几个思路：

1. 比如拿数据要写库，你先根据主键查一下，如果这数据都有了，就先别插入了，update一下
2. 比如写redis，那没问题，反正每次都是set，天然幂等
3. 需要让生产者发送每条数据的时候，里面加一个全局的唯一的id，类似订单id之类的东西，然后你这里消费到了之后先根据这个id去比如redis里查一下，之前消费过吗？如果没有消费过就处理，然后这个id写redis，如果消费过了，就不要再处理。
4. 面试题

如何保证消息的可靠性传输（如何处理消息丢失的问题）

1. 面试官心理分析

如果mq传递核心的消息，比如计费之类的，扣费的一些消息，比如设计和研发一套核心的广告平台，计费系统是一个很重的业务，操作是很耗时的，所以说广告系统整体的架构里面，实际是将计费做成异步化的，然后中间就是加了一个MQ。例如在广告主投放了一个广告，约定的是每次用户点击一次就扣费一次，结果是用户动不动就点击了一次，扣费的时候搞的消息丢了，公司就会不断的少几块钱。这样积少成多，这就是造成了公司的巨大损失。

1. 面试题剖析

丢数据，一般分为两种，要么是MQ自己弄丢了，要么是我们消费的时候弄丢了。

我们可以从RabbitMQ和Kafka分别来进行分析。

RabbitMQ一般来说都是承载公司的核心业务的，数据是绝对不能弄丢的。