RabbitMQ面试题

1、公司生产环境用的是什么消息中间件?

> 这个首先你可以说下你们公司选用的是什么消息中间件，比如用的是RabbitMQ，然后可以初步给一些你对不同MQ中间件技术的选型分析。

a. ActiveMQ

- 然后你可以说说RabbitMQ，他的好处在于可以支撑高并发、高吞吐、性能很高，同时有非常完善便捷的后台管理界面可以使用。
- 另外，他还支持集群化、高可用部署架构、消息高可靠支持，功能较为完善。

b. RabbitMQ

- 国内各大互联网公司落地大规模RabbitMQ集群支撑自身业务的case较多，国内各种中小型互联网公司RabbitMQ的实践也比较多。
- RabbitMQ的开源社区很活跃，较高频率的迭代版本，来修复发现的bug以及进行各种优化，因此综合考虑过后，公司采取了RabbitMQ。

- RabbitMQ也有一点缺陷，就是他自身是基于erlang语言开发的，所以导致较为难以分析里面的源码，也较难进行深层次的源码定制和改造，毕竟需要较为扎实的erlang语言功底才可以。

c. RocketMQ

- RocketMQ，是阿里开源的，经过阿里的生产环境的超高并发、高吞吐的考验，性能卓越，同时还支持分布式事务等特殊场景。
- RocketMQ是基于Java语言开发的，适合深入阅读源码，有需要可以站在源码层面解决线上生产问题，包括源码的二次开发和改造。

d. Kafka

- Kafka提供的消息中间件的功能明显较少一些，相对上述几款MQ中间件要少很多。
- Kafka的优势在于专为超高吞吐量的实时日志采集、实时数据同步、实时数据计算等场景来设计。
- Kafka在大数据领域中配合实时计算技术（比如Spark Streaming、Storm、Flink）使用的较多。但是在传统的MQ中间件使用场景中较少采用。

2、Kafka、ActiveMQ、RabbitMQ、RocketMQ 有什么优缺点?

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类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 可以达到几百/几千的级别，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，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，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，延迟最低	ms 级	延迟在 ms 级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，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	基本不丢	经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	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

> 综上，各种对比之后，有如下建议：

- 一般的业务系统要引入 MQ，最早大家都用 ActiveMQ，但是现在确实大家用的不多了，没经过大规模吞吐量场景的验证，社区也不是很活跃，所以大家还是算了吧，我个人不推荐用这个了；
- 后来大家开始用 RabbitMQ，但是确实 erlang 语言阻止了大量的 Java 工程师去深入研究和掌控它，对公司而言，几乎处于不可控的状态，但是确实人家是开源的，比较稳定的支持，活跃度也高；

- 不过现在确实越来越多的公司会去用 RocketMQ，确实很不错，毕竟是阿里出品，但社区可能有突然黄掉的风险（目前 RocketMQ 已捐给 Apache，但 GitHub 上的活跃度其实不算高）对自己公司技术实力有绝对自信的，推荐用 RocketMQ，否则回去老老实实用 RabbitMQ 吧，人家有活跃的开源社区，绝对不会黄。
- 所以**中小型公司**，技术实力较为一般，技术挑战不是特别高，用 RabbitMQ 是不错的选择；**大型公司**，基础架构研发实力较强，用 RocketMQ 是很好的选择。
- 如果是**大数据领域**的实时计算、日志采集等场景，用 Kafka 是业内标准的，绝对没问题，社区活跃度很高，绝对不会黄，何况几乎是全世界这个领域的事实性规范。

3、解耦、异步、削峰是什么?

解耦：A 系统发送数据到 BCD 三个系统，通过接口调用发送。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？A 系统负责人几乎崩溃...A 系统跟其它各种乱七八糟的系统严重耦合，A 系统产生一条比较关键的数据，很多系统都需要 A 系统将这个数据发送过来。如果使用 MQ，A 系统产生一条数据，发送到 MQ 里面去，哪个系统需要数据自己去 MQ 里面消费。如果新系统需要数据，直接从 MQ 里消费即可；如果某个系统不需要这条数据了，就取消对 MQ 消息的消费即可。这样下来，A 系统压根儿不需要去考虑要给谁发送数据，不需要维护这个代码，也不需要考虑人家是否调用成功、失败超时等情况。

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，自己本地写库要 3ms，BCD 三个系统分别写库要 300ms、450ms、200ms。最终请求总延时是 3 + 300 + 450 + 200 = 953ms，接近 1s，用户感觉搞个什么东西，慢死了慢死了。用户通过浏览器发起请求。如果使用 MQ，那么 A 系统连续发送 3 条消息到 MQ 队列中，假如耗时 5ms，A 系统从接受一个请求到返回响应给用户，总时长是 3 + 5 = 8ms。

削峰：减少高峰时期对服务器压力。

4、消息队列有什么缺点

4.1. **系统可用性降低**

本来系统运行好好的，现在你非要加入个消息队列进去，那消息队列挂了，你的系统不是呵呵了。因此，系统可用性会降低；

4.2. **系统复杂度提高**

加入了消息队列，要多考虑很多方面的问题，比如：一致性问题、如何保证消息不被重复消费、如何保证消息可靠性传输等。因此，需要考虑的东西更多，复杂性增大。

4.3. **一致性问题**

A 系统处理完了直接返回成功了，人都以为你这个请求就成功了；但是问题是，要是 BCD 三个系统那里，BD 两个系统写库成功了，结果 C 系统写库失败了，咋整？你这数据就不一致了。

5、Rabbitmq一般用在什么场景

（1）服务间异步通信

（2）顺序消费

（3）定时任务

（4）请求削峰

6、简单说RabbitMQ有哪些角色

- Broker： 简单来说就是消息队列服务器实体
- Exchange： 消息交换机，它指定消息按什么规则，路由到哪个队列

- Queue： 消息队列载体，每个消息都会被投入到一个或多个队列
- Binding： 绑定，它的作用就是把exchange和queue按照路由规则绑定起来
- Routing Key： 路由关键字，exchange根据这个关键字进行消息投递
- VHost： vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。
- Producer： 消息生产者，就是投递消息的程序
- Consumer： 消息消费者，就是接受消息的程序
- Channel： 消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务

7、RabbitMQ有几种工作模式

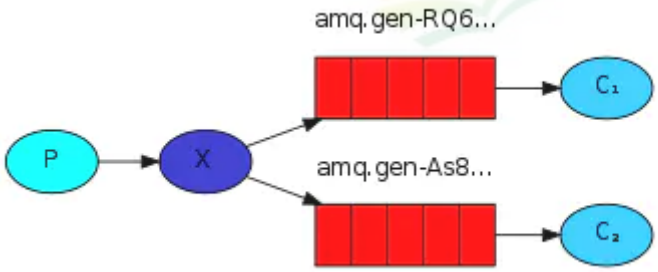
7.1 .simple模式（即最简单的收发模式）

- 消息产生消息，将消息放入队列
- 消息的消费者(consumer) 监听 消息队列,如果队列中有消息,就消费掉,消息被拿走后,自动从队列中删除(隐患 消息可能没有被消费者正确处理,已经从队列中消失了,造成消息的丢失，这里可以设置成手动的ack,但如果设置成手动ack，处理完后要及时发送ack消息给队列，否则会造成内存溢出)。

7.2 work工作模式(资源的竞争)

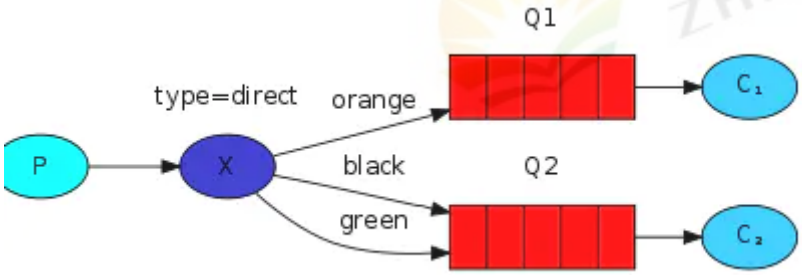
- 消息产生者将消息放入队列消费者可以有多个,消费者1,消费者2同时监听同一个队列,消息被消费。C1 C2共同争抢当前的消息队列内容,谁先拿到谁负责消费消息(隐患：高并发情况下,默认会产生某一个消息被多个消费者共同使用,可以设置一个开关(syncronize) 保证一条消息只能被一个消费者使用)。

7.3 publish/subscribe发布订阅(共享资源)



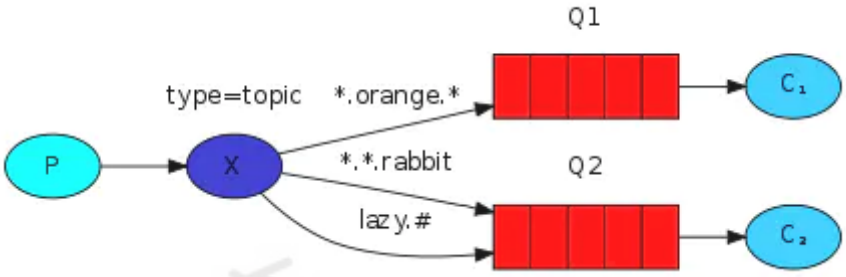
- 每个消费者监听自己的队列;
- 生产者将消息发给broker，由交换机将消息转发到绑定此交换机的每个队列，每个绑定交换机的队列都将接收到消息。

7.4.routing路由模式



- 消息生产者将消息发送给交换机按照路由判断,路由是字符串(info) 当前产生的消息携带路由字符(对象的方法),交换机根据路由的key,只能匹配上路由key对应的消息队列,对应的消费者才能消费消息;
- 根据业务功能定义路由字符串
- 从系统的代码逻辑中获取对应的功能字符串,将消息任务扔到对应的队列中。
- 业务场景:error 通知;EXCEPTION;错误通知的功能;传统意义的错误通知;客户通知;利用key路由,可以将程序中的错误封装成消息传入到消息队列中,开发者可以自定义消费者,实时接收错误;

7.5 topic 主题模式(路由模式的一种)

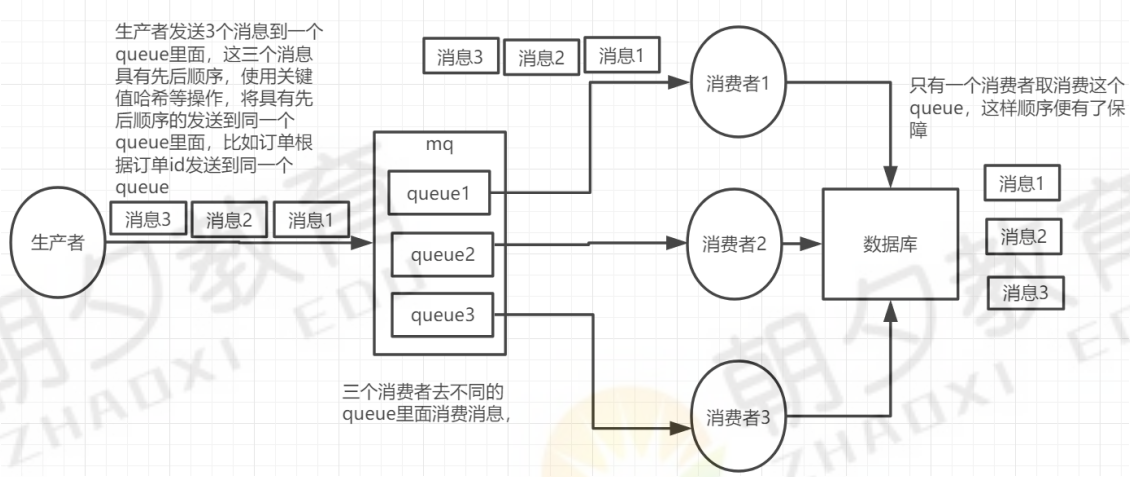


- `*`,`#`代表通配符

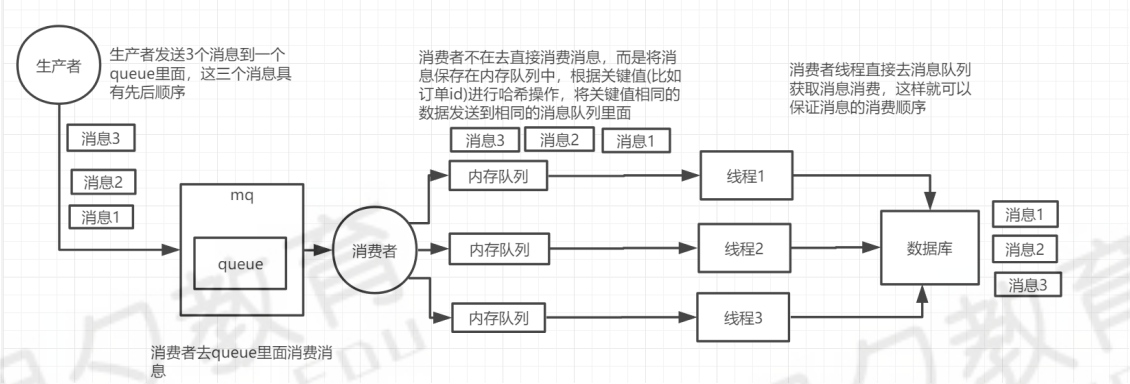
- * 代表多个单词, # 代表一个单词
- 路由功能添加模糊匹配
- 消息产生者产生消息,把消息交给交换机
- 交换机根据key的规则模糊匹配到对应的队列,由队列的监听消费者接收消息消费

8、如何保证RabbitMQ消息的顺序性?

- 将原来的一个queue拆分成多个queue，每个queue都有一个自己的consumer。该种方案的核心是生产者在投递消息的时候根据业务数据关键值（例如订单ID哈希值对订单队列数取模）来将需要保证先后顺序的同一类数据（同一个订单的数据） 发送到同一个queue当中。



- 一个queue就一个consumer，在consumer中维护多个内存队列，根据业务数据关键值（例如订单ID哈希值对内存队列数取模）将消息加入到不同的内存队列中，然后多个真正负责处理消息的线程去各自对应的内存队列当中获取消息进行消费。



9、消息怎么路由?

- 消息提供方->路由->一至多个队列消息发布到交换器时，消息将拥有一个路由键（routing key），在消息创建时设定。通过队列路由键，可以把队列绑定到交换器上。消息到达交换器后，RabbitMQ 会将消息的路由键与队列的路由键进行匹配（针对不同的交换器有不同的路由规则）；
- 常用的交换器主要分为一下三种：

  1. fanout：如果交换器收到消息，将会广播到所有绑定的队列上
  2. direct：如果路由键完全匹配，消息就被投递到相应的队列
  3. topic：可以使来自不同源头的消息能够到达同一个队列。 使用 topic 交换器时，可以使用通配符

10、如何保证消息不被重复消费

> 题型分析：为什么会出现消息重复？消息重复的原因有两个：1.生产时消息重复，2.消费时消息重复。

- 分析重复消费原因

  生产时消息重复

  由于生产者发送消息给MQ，在MQ确认的时候出现了网络波动，生产者没有收到确认，实际上MQ已经接收到了消息。这时候生产者就会重新发送一遍这条消息。生产者中如果消息未被确认，或确认失败，我们可以使用定时任务+（redis/db）来进行消息重试。

  消费时消息重复

  消费者消费成功后，再给MQ确认的时候出现了网络波动，MQ没有接收到确认，为了保证消息被消费，MQ就会继续给消费者投递之前的消息。这时候消费者就接收到了两条一样的消息。

- 解决方案
  - 让每个消息携带一个全局的唯一ID，即可保证消息的幂等性
    - 消费者获取到消息后先根据id去查询redis/db是否存在该消息。
    - 如果不存在，则正常消费，消费完毕后写入redis/db。
    - 如果存在，则证明消息被消费过，直接丢弃。

11、如何确保消息接收方消费了消息?

发送方确认模式

- 将信道设置成 confirm 模式（发送方确认模式），则所有在信道上发布的消息都会被指派一个唯一的 ID。
  一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个确认给生产者（包含消息唯一 ID）。
- 如果 RabbitMQ 发生内部错误从而导致消息丢失，会发送一条 nack（notacknowledged，未确认）消息。
- 发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

接收方确认机制

- 消费者接收每一条消息后都必须进行确认（消息接收和消息确认是两个不同操作）。只有消费者确认了消息，RabbitMQ 才能安全地把消息从队列中删除。
- 这里并没有用到超时机制，RabbitMQ 仅通过 Consumer 的连接中断来确认是否需要重新发送消息。也就是说，只要连接不中断，RabbitMQ 给了 Consumer 足够长的时间来处理消息。保证数据的最终一致性；

下面罗列几种特殊情况

- 如果消费者接收到消息，在确认之前断开了连接或取消订阅，RabbitMQ 会认为消息没有被分发，然后重新分发给下一个订阅的消费者。（可能存在消息重复消费的隐患，需要去重）
- 如果消费者接收到消息却没有确认消息，连接也未断开，则 RabbitMQ 认为该消费者繁忙，将不会给该消费者分发更多的消息。

12、如何保证RabbitMQ消息的可靠传输?

> 题型分析：
> 消息不可靠的情况可能是消息丢失，劫持等原因；
> 丢失又分为：生产者丢失消息、消息列表丢失消息、消费者丢失消息；

生产者丢失消息：

- 从生产者弄丢数据这个角度来看，RabbitMQ提供transaction和confirm模式来确保生产者不丢消息；

- 事务机制：发送消息前，开启事务（channel.txSelect()）,然后发送消息，如果发送过程中出现什么异常，事务就会回滚（channel.txRollback()）,如果发送成功则提交事务（channel.txCommit()）。然而，这种方式有个缺点：吞吐量下降；
- confirm模式：一旦channel进入confirm模式，所有在该信道上发布的消息都将会被指派一个唯一的ID（从1开始），一旦消息被投递到所有匹配的队列之后；rabbitMQ就会发送一个ACK给生产者（包含消息的唯一ID），这就使得生产者知道消息已经正确到达目的队列了；如果rabbitMQ没能处理该消息，则会发送一个Nack消息给你，你可以进行重试操作。

消息队列丢数据：

- 处理消息队列丢数据的情况，一般是开启持久化磁盘的配置。这个持久化配置可以和confirm机制配合使用，你可以在消息持久化磁盘后，再给生产者发送一个Ack信号。这样，如果消息持久化磁盘之前，rabbitMQ阵亡了，那么生产者收不到Ack信号，生产者会自动重发。

13、为什么不应该对所有的 message 都使用持久化机制?

- 首先，必然导致性能的下降，因为写磁盘比写 RAM 慢的多，message 的吞吐量可能有 10 倍的差距。
- 其次，message 的持久化机制用在 RabbitMQ 的内置 cluster 方案时会出现"坑爹"问题。矛盾点在于，若 message 设置了 persistent 属性，但 queue 未设置 durable 属性，那么当该 queue 的 owner node 出现异常后，在未重建该 queue 前，发往该 queue 的 message 将被 blackholed；若 message 设置了 persistent 属性，同时 queue 也设置了 durable 属性，那么当 queue 的 owner node 异常且无法重启的情况下，则该 queue 无法在其他 node 上重建，只能等待其 owner node 重启后，才能恢复该 queue 的使用，而在这段时间内发送给该 queue 的 message 将被 blackholed 。
- 所以，是否要对 message 进行持久化，需要综合考虑性能需要，以及可能遇到的问题。若想达到 100,000 条/秒以上的消息吞吐量（单 RabbitMQ 服务器），则要么使用其他的方式来确保 message 的可靠 delivery，要么使用非常快速的存储系统以支持全持久化（例如使用 SSD）。另外一种处理原则是：仅对关键消息作持久化处理（根据业务重要程度），且应该保证关键消息的量不会导致性能瓶颈。

14、如何保证RabbitMQ高可用的?

> RabbitMQ 是比较有代表性的，因为是基于主从（非分布式）做高可用性的，我们就以 RabbitMQ 为例子讲解第一种 MQ 的高可用性怎么实现。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，一般就是学习时候用的，没人在生产环境使用单机模式
- **普通集群模式**：
  - 意思就是在多台机器上启动多个 RabbitMQ 实例，每个机器启动一个。
  - 你创建的 queue，只会放在一个 RabbitMQ 实例上，但是每个实例都同步 queue 的元数据（元数据可以认为是 queue 的一些配置信息，通过元数据，可以找到 queue 所在实例）。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。这方案主要是提高吞吐量的，就是说让集群中多个节点来服务某个 queue 的读写操作。
- **镜像集群模式**：
  - 这种模式，才是所谓的 RabbitMQ 的高可用模式。跟普通集群模式不一样的是，在镜像集群模式下，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，就是说，每个 RabbitMQ 节点都有这个 queue 的一个完整镜像，包含 queue 的全部数据的意思。然后每次你写消息到 queue 的时候，都会自动把消息同步到多个实例的 queue 上。RabbitMQ 有很好的管理控制台，就是在后台新增一个策略，这个策略是镜像集群模式的策

略，指定的时候是可以要求数据同步到所有节点的，也可以要求同步到指定数量的节点，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他的节点上去了。

- 这样的好处在于，你任何一个机器宕机了，没事儿，其它机器（节点）还包含了这个 queue 的完整数据，别的 consumer 都可以到其它节点上去消费数据。坏处在于，第一，这个性能开销也太大了吧，消息需要同步到所有机器上，导致网络带宽压力和消耗很重！RabbitMQ 一个 queue 的数据都是放在一个节点里的，镜像集群下，也是每个节点都放这个 queue 的完整数据。

15、如何解决消息队列的延时以及过期失效问题？消息队列满了以后该怎么处理？有几百万消息持续积压几小时，说说怎么解决？

> 线上挺常见的，一般不出，一出就是大 case。一般常见于，举个例子，消费端每次消费之后要写 mysql，结果 mysql 挂了，消费端 hang 那儿了，不动了；或者是消费端出了个什么岔子，导致消费速度极其慢。

一般这个时候，只能临时紧急扩容了，具体操作步骤和思路如下：

- 先修复 consumer 的问题，确保其恢复消费速度，然后将现有 consumer 都停掉。
- 新建一个 topic，partition 是原来的 10 倍，临时建立好原先 10 倍的 queue 数量。
- 然后写一个临时的分发数据的 consumer 程序，这个程序部署上去消费积压的数据， **消费之后不做耗时的处理**，直接均匀轮询写入临时建立好的 10 倍数量的 queue。
- 接着临时征用 10 倍的机器来部署 consumer，每一批 consumer 消费一个临时 queue 的数据。这种做法相当于是临时将 queue 资源和 consumer 资源扩大 10 倍，以正常的 10 倍速度来消费数据。
- 等快速消费完积压数据之后， **得恢复原先部署的架构**， **重新**用原先的 consumer 机器来消费消息。

> 必须考虑的特殊情况

15.1、消息过期失效了

假设你用的是 RabbitMQ，RabbtiMQ 是可以设置过期时间的，也就是 TTL。如果消息在 queue 中积压超过一定的时间就会被 RabbitMQ 给清理掉，这个数据就没了。那这就是第二个坑了。这就不是说数据会大量积压在 mq 里，而是**大量的数据会直接搞丢。**

这个情况下，就不是说要增加 consumer 消费积压的消息，因为实际上没啥积压，而是丢了大量的消息。我们可以采取一个方案，就是**批量重导**，这个我们之前线上也有类似的场景干过。就是大量积压的时候，我们当时就直接丢弃数据了，然后等过了高峰期以后，比如大家一起喝咖啡熬夜到晚上12点以后，用户都睡觉了。这个时候我们就开始写程序，将丢失的那批数据，写个临时程序，一点一点的查出来，然后重新灌入 mq 里面去，把白天丢的数据给他补回来。也只能是这样了。

假设 1 万个订单积压在 mq 里面，没有处理，其中 1000 个订单都丢了，你只能手动写程序把那 1000 个订单给查出来，手动发到 mq 里去再补一次。

15.2都快写满了

如果消息积压在 mq 里，你很长时间都没有处理掉，此时导致 mq 都快写满了，咋办？这个还有别的办法吗？没有，谁让你第一个方案执行的太慢了，你临时写程序，接入数据来消费，**消费一个丢弃一个，都不要了**，快速消费掉所有的消息。然后走第二个方案，到了晚上再补数据吧。

16、RabbitMQ中消息可能有的几种状态?

alpha: 消息内容(包括消息体、属性和 headers) 和消息索引都存储在内存中 。

beta: 消息内容保存在磁盘中，消息索引保存在内存中。

gamma: 消息内容保存在磁盘中，消息索引在磁盘和内存中都有 。

delta: 消息内容和索引都在磁盘中 。

17.死信队列?

DLX，全称为 Dead-Letter-Exchange，死信交换器，死信邮箱。当消息在一个队列中变成死信 (dead message) 之后，它能被重新被发送到另一个交换器中，这个交换器就是 DLX，绑定 DLX 的队列就称之为死信队列。

18.导致的死信的几种原因?

- 消息被拒（Basic.Reject /Basic.Nack) 且 requeue = false。
- 消息TTL过期。
- 队列满了，无法再添加。