第七章 项目

1. 数据库中有多少条记录？考虑过分库分表吗？

根据阿里《Java开发手册》提出的单表行数超过500w行或者是单表容量超过2GB，推荐使用分库分表来看，mysql单表性能瓶颈在500w左右（具体也与表结构设计和机器配置有关）

分库分表方案：

- 分库：使用mycat或者是Sharing-JDBC等中间件来作为数据库的代理
- 分表：单表业务量过大，可以采用按业务分表或者是按时间分表等手段来切分

分库分表，结合实际需求来操作，一般来说在一个项目的初期是不会考虑分库分表的，随着业务量和数据量的不断上升，才有一些性能上的要求，这个时候假如数据库有瓶颈了就可以考虑使用分库分表来提高系统的性能。

2. 项目的并发量是多少？

首先，衡量项目的并发量有两个指标一个是QPS，一个是TPS。

QPS 每秒能处理查询数目，但现在一般也用于单服务接口每秒能处理请求数。
TPS 每秒处理的事务数目(一个事物中可能包含对于多个接口的访问)，如果完成该事务仅为单个服务接口，我们也可以认为它就是QPS。

并发量是要根据项目的实际情况来说的，你比如你一家小公司的话，注册用户大概也就是个几万，其中活跃用户，顶多也就是个4%-7%,这样算下来活跃用户，大概数量级就是几百，在回答并发量的时候，你应该说平时的qps上大概也就是个几十，差不多上百左右(tps比qps适当少点)，高峰期的并发量，比你的活跃用户数量在多一点几百(qps比如4,5百，tps的话2，3百)，大概就是这个思路，具体数字可以改，实际的高并发项目，是可遇而不可求的

如果面试官觉得，并发量低，没必要使用微服务，那么你可以说，公司一直在拓展业务，微服务架构可伸缩性很好，而且更加灵活，可以更好的应对用户量的改变，还可以最大程度的代码复用，因为可以让不同的客户端，复用同一套商城的微服务，而且使用微服务，可以让项目的打包，上线更为灵活等等就是说微服务的好处。

如果面试官问，那你们的压测结果，峰值大概能承受多大的并发量，那我们说一个大概的数量级就可以了，比如qps大概1000-1500，tps大概800-1000

3. 如何限流？

- 应用级别
  - 限制瞬时并发数 ，如nginx的limit_conn模块，用来限制瞬时并发连接数、nginx的limit_req模块，限制每秒的平均速率。
  - 限流总并发/连接/请求数，如配置tomcat的线程池的参数：

    acceptCount:如果Tomcat的线程都忙于响应，新来的连接会进入队列排队，如果超出排队大小，则拒绝连接

    maxConnections:瞬时最大连接数，超出的会排队等待

    maxThreads:Tomcat能启动用来处理请求的最大线程数，如果请求处理量一直远远大于最大线程数则可能会僵死
- 服务级别
  - 限制总资源数

使用线程池，连接池等等，比如分配给每个应用的数据库连接是100个，那么该应用在同时最多能使用100个资源，超出了可以等待或者抛出异常

- 限流某个接口
  - 限制某个接口的总并发量：秒杀大闸，信号量，AtomicLong等等
  - 平滑限流某个接口的请求数：令牌桶，RateLimiter

4. 说一下服务熔断，服务降级和限流?

服务熔断，熔断这一概念来源于电子工程中的断路器。在互联网系统中，当下游服务因访问压力过大而响应变慢或失败，上游服务为了保护系统整体的可用性，可以暂时切断对下游服务的调用。这种牺牲局部，保全整体的措施就叫做熔断。

服务降级，通常指因为某些原因导致无法正常调用服务原本的功能，而执行服务的降级方法，而直接返回一个提前准备好的非正常数据(比如默认数据，可能已经过期的缓存数据等)。这样，虽然提供的是一个有损的服务，但却保证了整个系统的稳定性和可用性。

服务熔断，为了避免在服务调用链中，某个服务导致的问题，层层级联，最终导致整个链路都不可用，而暂时切断对不稳定或出现问题的服务的调用，以避免局部不稳定因素导致整体的雪崩的情况。发生服务熔断之后，对于被熔断的服务调用，也可以走服务降级逻辑，此时我们称这种服务降级为熔断降级

服务熔断与降级实现框架：Hystrix，Sentinel，限流见上一题

5. JWT的原理?

1）前端通过表单将用户名和密码发送到后端接口。

2）后端核对用户名和密码后，将用户的id及其他非敏感信息作为JWT Payload，将其与头部分别进行base64编码后签名，生成JWT

3）后端将JWT字符串作为登录成功的结果返回给前端，前端可以将JWT存到localStorage或者sessionStorage中，退出登录时，前端删除保存的JWT信息即可。

4）前端每次在请求时，将JWT放到header中的Authorization或者cookie中

5）后端验证JWT的有效性

6）验证通过后，进行其他逻辑操作

6. JWT的Token，假如被抓包，别人仿造一个请求，那怎么处理?

实际上JWT本身只能做到，防止数据篡改，你说的这个问题，实际上是JWT的重放攻击，这个问题的核心就是，客户端每次发起请求的时候，所使用的token信息，只能被使用一次，要是有一个token被多次使用，从第二次开始token无效。

怎么实现呢?

- 每次前端在发起请求的时候，对于JWT的token信息利用提前协商好的公钥进行加密，
  并且在加密的时候，对token+时间戳进行加密，请求服务器端
- 服务器端利用私钥进行解密，检查有没有时间戳并且看下token中的jti这个有没有已经被使用过了，
  如果已经使用过了，那就是无效请求，不予处理，如果没有时间戳，那说明不是普通用户发起的请求，也不予处理
- 如果请求合法，最终会在给前端的响应中，放入一个新生成的token，新生成的token中给一个新的唯一的jti，
  并且用私钥加密，加密的时候不用像前端一样在加时间戳，然后返回用户，用户下次就带着这个新的token发起请求

- 这样一来，中间人无法伪造客户端请求
  a. 如果使用客户端已经使用过的token，那么服务器端每个使用过的token以jti为唯一标识都是记录过的，第二次就无效了
  b. 如果在服务器端返回响应的时候，获取服务器端返回的token，先于客户端发起请求，也不行，因为正常的客户端请求
  是需要对token+时间戳进行加密的，而中间人没有秘钥，无法哪出token，无法以token + 时间戳的方式发起请求

7. Nginx如何做负载均衡?

Nginx 的 upstream目前支持的分配算法：

- 轮询 ——1：1 轮流处理请求（默认）

  每个请求按时间顺序逐一分配到不同的应用服务器，如果应用服务器down掉，自动剔除，剩下的继续轮询。

- 权重

  通过配置权重，指定轮询几率，权重和访问比率成正比，用于应用服务器性能不均的情况。

- ip_哈希算法

  每个请求按访问ip的hash结果分配，这样每个访客固定访问一个应用服务器，可以解决session共享的问题。

通过在upstream参数中添加的应用服务器IP后添加指定参数即可实现

8. MQ消费者如果消费不成功怎么办呢?

- MQ如何消费不成功，RocketMQ会把该消息丢入重试队列当中去
- MQ重试队列默认有重试机制，会反复去投递直到投递成功或者是在16次（默认的，可更改）之后假如还没有消费失败的话会把该消息投递到MQ死信队列里面
- 我们可以通过后台管理系统从死信队列里面把没消费成功的消息取出来，然后手动的去进行库存的扣减

9. 遇到消息堆积问题如何解决

- 首先，要先定位发生消息堆积的原因，恢复其消费。
- 临时扩容，新建一个Topic，给Topic分配原来n倍的queue的数量
- 让后写一个临时的分发程序，用来消费积压的消息，不做任何耗时操作，就从原来的Topic里把消息，
  读出来，写入到这个新的Topic里
- 然后临时使用n倍的机器来部署 consumer，每一批 consumer 消费一个临时 queue 的数据，
  这样一来相当于，给临时将 queue 资源和 consumer 资源扩大 n倍，以正常的 n 倍速度
  来消费数据。
- 等积压的消息被消费完成之后，再恢复原来的架构，重新使用原来的架构消费消息

10. 关于秒杀下单过程中的库存扣减问题

首先，在秒杀下单过程中究竟应该加悲观锁还是乐观锁?

- 乐观锁，主要适用于读多写少的场景，在这样的场景下，不保证一定会读到最新的数据，但是读操作是不会被阻塞的。
- 但是，我们来分析下，由于我们只在扣减库存时，对于扣减库存的操作，使用基于Redis的分布式锁进行加锁，所以，如果有其他线程此时要去读库存，是不会被阻塞的，所以即使我们在扣减库存的时候使用悲观锁进行加锁，也不会存在读阻塞的情况。
- 同时，使用乐观锁，在更新数据的时候，至少要访问两次甚至多次数据库，因为首先要读取待修改数据的版本号，然后才是访问数据库去更新数据，如果发现在更新数据库时，之前读取到的数据的

版本号，小于待更新数据的当前版本号，那么此时，说明已经有其他线程同时修改了数据，更新失败，需要继续重复上述过程，所以因为使用乐观锁，一次更新需要多次访问数据库，所以导致了在高并发场景下，乐观锁是不适用的
- 所以在我们的秒杀扣减库存的过程中，既存在高并发，又不会出现读阻塞，所以加锁适用的应该是悲观锁

其次，如果有面试官质疑，使用悲观锁，因为会导致阻塞，所以会不会导致扣减库存的执行效率不高?

关于这个问题，我们首先要明确的一点是，确实因为线程阻塞会导致上下文的切换，所以相比之下，确实会导致在并发场景下，扣减库存的执行效率不是非常理想，我们站在小公司的立场，并发量本来就不是很高，所以从这个角度来说，虽然使用了悲观锁，但是完全是可以应付这个体量的并发请求的，而且在网关处，我们也是做了限流的，限制了同时处理的请求数量。

如果面试官在进一步提问，有没有更好的库存扣减的方式，那么此时我们可以说，另外一种解决方案:

我们可以说，将秒杀库存全部放入到Redis中来存储，这样做的好处是:

- 因为Redis的工作线程是单线程，所以利用lua脚本，完成在Redis中扣减库存的操作，天然就能保证扣减库存的原子操作，无需加锁，所以实现了无锁化
- Redis中执行扣减库存，又是纯内容操作

11. 支付状态，如果因为网络原因，支付宝没有及时通知到，导致你们认为用户订单超时未支付，但实际用户已经支付了，怎么办?

这种情况就属于比较极端的情况了，但是也并非不会出现，因为网络问题是不可控的。如何解决这个问题呢?

- 在取消执行取消订单超时自动取消的逻辑的时候，为了防止以上问题的发生，当我们发现订单的支付记录，还是未支付的状态的时候，此时可以在主动去查询下用户的订单支付状态，如果已支付，那么什么都不做，修改支付状态为已支付。
- 如果还是未支付，那么此时，调用支付宝的交易取消接口，取消本次订单的支付交易，防止在取消订单后，用户又去支付(或者这里还有另外一种方案，就是设置交易的超时时间，当然一旦设置超时时间，用户在支付的话支付宝会提示二维码失效)
- 然后，在执行订单超时取消的逻辑即可

12 下单的pipeline模式中间有一步失败了该如何处理?

关于这个问题，我们首先要清楚，不同的设计模式，改变的仅仅只是代码的书写方式，但是不管代码的书写方式如何变化，它都不会影响代码要实现的功能。

虽然使用了管道设计模式，由多个handler来完成，但是从实现角度，多个Handler方法中handle方法的执行，本质上还是一连串的方法调用，这一连串的方法调用一定是在某一个方法里执行，所以从对数据库访问的角度，我们完全可以理解为我们在一个方法里，写了很多访问数据库的代码，完全可以让它们成为一个事物

所以，要回答这个问题，我们就要首先分析，这个事物是什么类型的事物。通常一个下单过程可能会调用别的服务，所以要让整个下单过程成为一个事物，那么这个事物会是一个分布式事务，所以我们可以说，我们可以说下单过程我们用的是基于Seata的AT模式实现的分布式事务，失败之后会回滚。

可能又有同学会说，老师为啥下单实现的分布式事务用Seata实现的分布式事务，而秒杀下单却用基于RocketMQ事物消息实现，因为我们认为普通的下单接口，访问量不是很大，所以用Seata的AT模式是没啥问题的，但是在秒杀服务中，秒杀下单接口的访问量相对较大，所以适合用基于RocketMQ事物消息实现，因为这种方式是异步的，适合更高的并发

13. 项目中的promoService中的线程池设置成100个，为什么设置成100?

这其实是一个开放性问题，主要是看你自己对线程池的调优有没有思考 过，或者是自己的理解，我这里就不给出标准答案了 这个线程池大小设置其实是和CPU性能有关的，当然也和你的应用有关，一般来说分为三种应用，一种是CPU 密集型，一种是IO密集型，还有一种是混合型 我们这个是WEB应用，属于IO密集型服务，一般也部署在IO密集型机器上

在《JAVA并发编程的艺术》一书中提到了如何设置线程池的大小，假如现在机器的CPU是n核，一般如果是

CPU密集型，那么参考值可以设置为n+1，如果是IO密集型，那么CPU可以设置为2n+1

当然以上仅供参考，然后你跟面试官说完这个之后，你就可以说，我们公司的服务器配置是什么样子的，一开

始我们也是这样配置的， 但是经过我们的压力测试之后，我发现我把线程池大小优化为多少多少（假如是100）的时候并发效率最高

14 如何实现提交订单的幂等性，保证正确处理重复的下单请求

- 用户每次进入提交订单页面的时候, 前端异步调用发号器接口，在提交订单页面生成唯一的序列号，该唯一序列号返回前端同时，
  也向redis中存储一份, 存的时候，key是序列号，值是针对这个key的一个计数值，初值为0
- 用户在订单页面,点击提交订单的时候，传递该序号，后端通过redis的原子方法++，对该系列号对应的计数值++，然后做判断，如果大于2，则代表请求重复，
  对该请求不做处理即可。
- 这样就可以防止用户重复提交订单。

15 关于购物车数据的存储问题

关于购物出数据的存储，虽然我们只是用Redis来存储了购物车数据，但是在实际开发中，购物车数据在Redis中应该仅仅只是最为缓存，购物车数据应该还是要在数据库中存储的，那么这里就涉及到一个问题，就是Redis缓存和数据库数据中数据的一致性问题了，即所谓缓存双写一致性问题。那么这个问题如何解决呢?

在回答这个问题的时候，面试官可能会想听对于更新缓存时的一些思考，主要集中在两个问题

- 更新数据时，究竟是先更新数据库，还是先更新缓存(其实都会存在问题)
- 对于缓存数据，究竟是更新，还是删除
- 这里大家可以去搜一下Cache Aside Pattern，主要看下分析过程，因为这个模式也不能完全解决问题，之所以提这个模式，因为Facebook号称也在用

那么生产中可能会使用什么样的解决方案呢?

- 最简单的解决方案，如果数据的时效性不高，此时可以仅仅Redis中缓存数据的过期时间即可。
- 加分布式锁(加的这个分布式锁，你还可以说是读写锁），加锁的好处在于，实现并发请求的同步，让每个更新数据库及缓存的操作，一个一个的顺序完成，这样可以规避并发场景下更新缓存数据错乱的情况。
- 但是加锁，一定程度上，会带来性能瓶颈，如何解决这个问题呢？可以使用中间件来解决这个问题，大家可以简单了解下Canal这个阿里开源的中间件，可以用它实现，实时的数据库数据和Redis数据的同步

大家在说的时候，可以说我们使用的方式是加锁方式，即第二种方案解决的，因为我们是小公司，并发量没有那么高，这种方案是可行的，但是你可以提一下第三种方案。

16. 一台服务器并发量多少，一共多少台服务器，怎么部署的

这个问题其实也是看机器配置的，不同配置的机器并发能力肯定不一样，我这里就拿4c8t的机器来举例说明吧 其实一台服务器正常的并发量并不高，可以通过Jmeter等压测工具来测试。一般来说公司的线上环境的机器 QPS在经过优化之后大约在1000-2000之间，具体是多少取决于你做了哪些优化（线程池优化，Tomcat优 化，Nginx优化，JVM优化等）

一共部署多少台服务器这个问题，大家可能没什么概念 一般来说，小公司，正常的服务至少部署2个服务器（为什么至少部署两个呢？ 因为一个挂了还有另外一个可 以用嘛，保证服务的高可用，这是最低要求了）

如果应用对并发需求比较高的话，例如秒杀服务，可以针对QPS来动态扩容（为什么可以动态扩容呢？因为 Dubbo是支持负载均衡的，只要你把同样的服务启动了，启动之后注册到注册中心，那么走Dubbo接口调用的 时候就会自动自动走到你新跑起来的服务）

怎么部署的？ 一般来说，部署的工作和开发无关，是运维的事情，你可以直接说运维打包部署 当然如果你的公司比较小，那么也可能是开发部署，那么部署这里就有很多种方式了

- 打成war包启动tomcat（Springboot应用不推荐这么做）
- 直接线上Jar包启动，这个是比较原始的方式，现在很少有公司这么做了（命令：Java -jar xxx.jar
- 后台(后台的意思是说关掉控制台服务不会停止)启动命令：nohup java -jar xxx.jar &）
- 利用打包工具Jenkins打包启动（这个工具大概是干什么的大家可以去了解一下，去工作了会用即可）
- 利用打包工具Jenkins配合Docker直接pull docker镜像下来部署上线

17. 项目的数据量有多大

其实这个问题也是和第二个问题紧密相关的 回答这个问题的时候你得有个概念，那么就是你这个项目日活人数有多少

我这里举个例子，假如你这个购物网站，日活100人，那么你的表（尤其是记录一些操作的操作流水表）的数

据量在一个月之后大概是在十万级（假如日活是100人的话，其实是不需要做高并发接口设计的） 依次类推（当然我 这里只说是购物网站，假如你做的软件游戏的话那么这个比例会大的多，因为游戏会产生很多记录）

一般来说，一个网站的日活数据，是一个公司的的核心数据，面试官问到了可以直接说不方便说（如果项目没 上线就算了), 假如你说的表的数据达到了千万级别，那么你得有一个需要分库分表的概念了，因为Mysql的性能在你的单表 数据量超过500w（大概数字）之后会急剧下滑 这个时候你可以说一些关于分库分表相关的东西

18. 生产中如何定位问题

这个其实是一个比较大的问题了，是什么样的问题呢？我说一下大概的思路吧 如果是你的JAVA应用出了问题

1，出现了问题肯定是首先要看日志，看是哪个地方抛出了什么样的异常

2，然后根据异常分析相对应的原因，一般来说面试官常问的无非是那几个，什么内存溢出啊，CPU负载飙高

（一般来说发生这个事出现了死循环），接口响应慢啊之类的，大家可以去网上看一下

我这里拿内存溢出举例子 ,一般来说如果是出现了内存溢出，那么你要知道是哪里内存溢出了吧，那么这个时候你需要知道整个JVM内存

分布的情况是什么样子的，这个时候我们可以使用JDK自带的命令行工具（jmap）把这个JVM的内存结构dump 下来（不止jmap可以，一般来说如果你的服务已经挂了，那么使用jmap，如果你的服务没有挂， 那么使用 jstack，当然也可以使用第三方一些开源的工具，大家有兴趣可以自己了解一下）

然后dump下来的之后你会得到一个dump格式的文件，这个文件需要怎么打开呢?

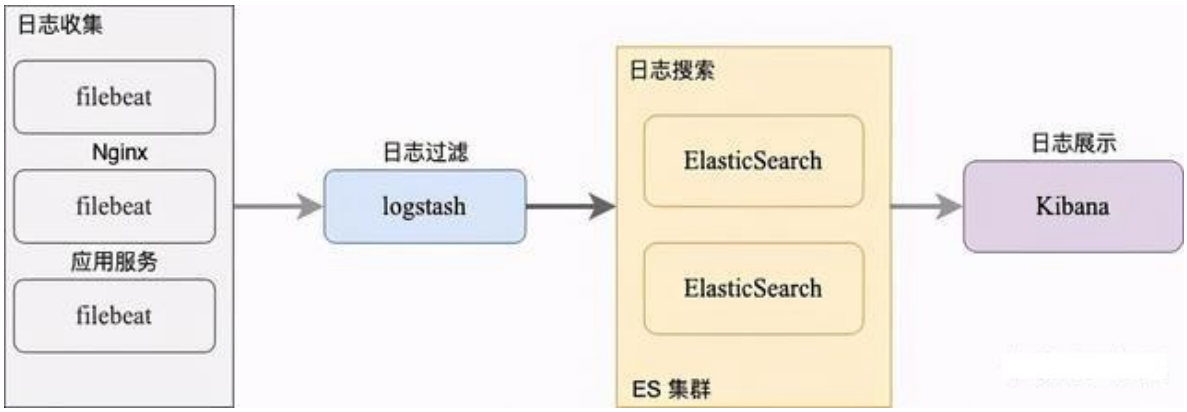
例如jdk自带的visualvm工具打开，打开之后你可以看一下哪些对象占用内存比较高，是不是正常的，假如 同一个类型的对象占有超过了50%（这个数字只是举个例子），那么我们可以认为该对象在不断的创建，那么 这个时候就根据这个对象去找相应的代码改掉即可，假如整个内存分配都很正常，只是单纯的内存不够了，这 个时候可以考虑优化JVM的参数，例如 -Xmx（大家还记得吗？）

如果你还想回答的更加深入一点，那么这个时候你可以往GC上面靠，说看分析发现mirrorGC很频繁，或者是 fullGC很频繁，那么这个时候是不是可以调节JVM young区和old区的比例呢？或者是调整JVM垃圾回收的 暂停时间， 大家可以往更深入了想一下，一般如果你回答到了这里那么认为你有线上JVM调优经验，已经不止 两年的水平了

19 关于项目日志

在微服务架构的项目中，每个服务的日志可能分布在不同机器上，我们在分析问题查看日志的时候，肯定不可能去每台机器上查看每个日志文件，效率太低。所以在微服务架构的项目中，需要有一个日志中心，我们需要查看日志的时候，我们直接去日志中心查看即可，目前比较流行的日志中心解决方案就是ELK了。

- Elasticsearch：是一个开源分布式时实分析搜索引擎，我们学过就不在具体解释了
- Logstash ： Logstash是一个用来搜集、分析、过滤日志的工具。它支持几乎任何类型的日志，包括系统日志、错误日志和自定义应用程序日志。它可以从许多来源接收日志，这些来源包括syslog、消息传递（例如 RabbitMQ）和JMX，它能够以多种方式输出数据，包括电子邮件、websockets和Elasticsearch。
- Kibana：也是一个完全开源的工具，kibana可以为Logstash和Elasticsearch提供图形化的日志分析。Web界面，可以汇总，分析和搜索重要数据日志。



在ELK实现的日志中心中，本身Logstash可以收集，传输日志文件数据，但是Logstash自己的日志文件数据的收集和传输性能不是很好，所以日志文件数据的收集和传输，由另外一个组件叫Filebeat来完成，Filebeat能快速收集日志文件数据，日志数据的收集交给Filebeat，然后由他把收集的数据交过Loastash，将日志数据做过滤转化为指定格式，然后写入Elasticsearch，然后我们开发者就可以在Kibana中搜索日志数据了。

查询日志时需要注意:

- 在查看日志的时候，我们可以根据我们在日志中的标记，或者日期等条件查询日志。
- 如果想要查看一个请求处理，涉及到的多个服务的日志，我们可以通过TranceId来查询，因为我们可以在一个请求开始处理的时候生成一个TraceId，然后在各个服务中传递这个TraceId, 处理同一个请求的日志，在各个服务中TraceId相同，通过TraceId就可以看到一个请求被处理的完整日志

20 公司的情况

- 公司的定位属于小公司，50人以下，大概就20-30人
- 公司的组织，分为技术部，财务部，人事部，运营部，采购和物流部(或采购部，或者物流部)，客服部

    - 运营部：负责市场推广、产品促销、吸引用户等工作
    - 采购和物流部：负责商品采购、商品库存管理、发货等工作
    - 财务部：负责财务整理、核算、成本控制等工作
    - 技术部：负责技术方面，如设计、开发、维护等工作，为其它部门服务
    - 客服部：负责售后、用户问题收集、直接与终端用户打交道，提升用户满意度等工作
    - 人事部：主要就是平时的一些考勤统计，招人等工作
- 技术部门的人员组成
    - UI，即界面设计，1人
    - 前端开发：1-2人
    - 如果有移动端的话，如果是Android和IOS的APP，那就2个，一个Android一个IOS，如果是微信小程序的话，那就一个
    - 后端开发：3-6个
    - 测试：1-2(如果开发人数说的少，那测试就1个)
    - 产品：1-2
    - 运维：1个

21 项目的时间周期

- 开发周期，大概4-6个月

如果我们说项目上线了(比如你找了一个类似的网站)，此时还有可能被问到项目的迭代周期，那我们可以做如下回答

- 基本2-4周就会有一次迭代，增加新功能，或者改良已有功能,或者对已有bug的修复(如果出现了严重bug的话, 那么就不属于正常的迭代周期，改完就会重新上线)
- 基本2-3个月就会有一次，大的版本迭代