1、问介绍一下自己，以及技术栈
redis,java，简单的 python.数据库主要 mysql，次 oracle

2、redis 基础数据结构有哪些
key String,hash,LinkedList,set（集合）,zset(有序集合)

3、mq 的作用
message queue
本质是对消息异步处理，将部分没那么重要的功能从主功能中整理出来。比如日志功能
解耦，将这些消息功能提取出来后，类似发邮件，发警告等消息提醒，可以从 mq 直接获取，使得系统更加的灵活
消峰。通过消息的排列处理，避免热点数据压垮系统的情况
缺点：增加了系统复杂性，系统的可用性降低（复杂后的链路变长，容易出问题）
常见的 kafka,rocketmq,activemq

4、如何解决迸发操作，例如同一账号同一时间不同设备同时点击领取了一张优惠券
通过 redis 缓存处理。
比如商品和相关的优惠券， 对应的规则关系，甚至是用户关系，都可以提前写到 redis 中缓存起来。 通过提前将算法结果保存起来， 在执行的时候，只需要查找，而不是实施的进行匹配

5、自动化日常如何执行的。一次多长时间如何缩短时间
一般情况使用 jenkins 自动跑。代码提交流程 xxxxxx
缩短自动化时间两种方法，一种是使用多线程进行测试验证，还有一种是服务切割。比如我本次提交了 200 个用例，那么可以通过设置 suit.xml 配置文件指定本次验证服务目录进行验证

6、jmeter 压测如何保障 tps 始终在一个数值
在负载逐渐升高的情况下，tps 却长期不变。这并不是说明性能很稳定，而是说明我们单位时间内的单线程 tps 是在逐渐降低的（单位时间 tps/总线程）。
再分析响应时间，我们的响应时间其实也是在逐渐升高，从侧面反映出线程的 tps 是在下降的。

7、压测内存过高如何解决
通过 top 命令，可以确认下，到底是哪个进程导致 CPU 变高，top
使用 top –Hp pid 来对该进程下的线程进行观察。展示的 pid 就是对应的线程了。
jstack 查找这个线程的信息
jstack [进程]|grep –A 10 [线程的 16 进制]
常见线程描述：
wait on monitor entry： 被阻塞的,这种情况是有问题的
runnable ： 注意 IO 线程
in Object.wait()： 注意非线程池等待

8、spring 中怎么运用数据库？ 2、spring IOC 的理解？ 3、spring bean 的理解和使用？ 4、支付回调延迟时，有什么方案可以去做监控？从代码上面怎么去实现？
引入 7 个包，配置下 xml 文件，当前更流行 springboot 使用 yarml 配置
控制反转：一般情况是纵向流程，耦合性较高。使用 spring 的控制反转，我们前端 jsp 或者 js 调用 spring 配置文件，通过每个 id 找到 bean，进而实现服务调用
依赖：通过容器对设置 bean，并配置响应 id 等参数；一般容器初始化或者首次调用 getbean 时触发； 通过 setbean 的形式实现参数注入
注入：构造器注入，通过 xml 设置 property 实现；通过 set 注入，通过 setbean 实现

9、索引数据结构介绍，和 B+ 树区别
数据库索引，是数据库管理系统中一个排序的数据结构，主要有
B 树索引、Hash 索引两种

哈希索引就是采用一定的哈希算法，把键值换算成新的哈希值，检索时不需要类似 B+ 树那样从根节点到叶子节点逐级查找，只需一次哈希算法即可立刻定位到相应的位置，速度非常快。
哈希索引缺点 Hash 索引只支持等值比较, Hash 索引无法被用来避免数据的排序操作,Hash 索引不支持多列联合索引的最左匹配规则,Hash 索引在任何时候都不能避免表扫描
mysql 中用的最多是 B+ 树，

B+ 树的特征：

1.有 k 个子树的中间节点包含有 k 个元素（B 树中是 k–1 个元素），每个元素不保存数据，只用来索引，所有数据都保存在叶子节点。

2.所有的叶子结点中包含了全部元素的信息，及指向含这些元素记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。

3.所有的中间节点元素都同时存在于子节点，在子节点元素中是最大（或最小）元素。

B+ 树的优势：

1.单一节点存储更多的元素，使得查询的 IO 次数更少。
2.所有查询都要查找到叶子节点，查询性能稳定。
3.所有叶子节点形成有序链表，便于范围查询。

10、什么情况下可以不回表查询
回表查询 ，需要扫码两遍索引树 ,先定位主键值，再定位行记录，它的性能较扫一遍索引树更低。
怎么避免？不是必须的字段就不要出现在 SELECT 里面。或者 b,c 建联合索引。但具体情况要具体分析，索引字段多了，存储和插入数据时的消耗会更大。这是个平衡问题。

10、MySQL 事务隔离级别
SQL 标准定义了四个隔离级别： 读取未提交，读取已提交，可重复读，可串行化

READ-UNCOMMITTED(读取未提交)： 最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读。
READ-COMMITTED(读取已提交)： 允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生。
REPEATABLE-READ(可重复读)： 对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。
SERIALIZABLE(可串行化)： 最高的隔离级别，完全服从 ACID 的隔离级别。所有的事务依次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重复读以及幻读。
隔离级别 脏读 不可重复读 幻影读
READ-UNCOMMITTED √ √ √
READ-COMMITTED × √ √
REPEATABLE-READ × × √
SERIALIZABLE × × ×

11、为什么选择 Redis 做缓存
由于 redis 访问速度块、支持的数据类型比较丰富，所以 redis 很适合用来存储数据,另外结合 expire，我们可以设置过期时间然后再进行缓存更新操作

12、数据库和缓存的数据一致性怎么保证
正常的缓存步骤是：
1、查询缓存数据是否存在

16、哨兵挂了怎么办（所以哨兵应该也是集群）
哨兵是 redis 集群架构中非常重要的一个组件，主要功能如下：

集群监控：负责监控 redis master 和 slave 进程是否正常工作
消息通知：如果某个 redis 实例有故障，那么哨兵负责发送消息作为报警通知给管理员
故障转移：如果 master node 挂掉了，会自动转移到 slave node 上
配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址
哨兵的核心知识
故障转移时，判断一个 master node 是宕机了，需要大部分的哨兵都同意才行，涉及到了分布式选举的问题
哨兵至少需要 3 个实例，来保证自己的健壮性
哨兵 + redis 主从的部署架构，是不会保证数据零丢失的，只能保证 redis 集群的高可用性
即使部分哨兵节点挂掉了，哨兵集群还是能正常工作的，因为如果一个作为高可用机制重要组成部分的故障转移系统本身就是单点，那么就不靠谱。

17、Redis 集群选举 master 过程（这个不会，说了 MySQL 集群的选主过程）
当 slave 发现自己的 master 变为 FAIL 状态时，便尝试进行 Failover，以期成为新的 master。由于挂掉的 master 可能会有多个 slave，从而存在多个 slave 竞争成为 master 节点的过程， 其过程如下：
1.slave 发现自己的 master 变为 FAIL
2.将自己记录的集群 currentEpoch 加 1，并广播 FAILOVER_AUTH_REQUEST 信息
3.其他节点收到该信息，只有 master 响应，判断请求者的合法性，并发送 FAILOVER_AUTH_ACK，对每一个 epoch 只发送一次 ack
4.尝试 failover 的 slave 收集 FAILOVER_AUTH_ACK
5.超过半数后变成新 Master
6.广播 Pong 通知其他集群节点。

18、MQ 是怎么防止消息丢失的

20、如果使用无界等待队列会有什么问题

当线程在执行任务时需要调用远程服务，当调用远程服务异常时，就会导致线程处理每个任务都需要等待很长的时间；

处理任务的速度慢，产生任务的速度快，而任务队列是没有边界的，就会导致队列变得越来越大，从而导致内存飙升，还可能导致 OOM 内存溢出。

21、介绍一下锁

锁是用来控制多个线程访问共享资源的方式，一般来说，一个锁能够防止多个线程同时访问共享资源

1、同步锁

同一时刻，一个同步锁只能被一个线程访问。以对象为依据，通过 synchronized 关键字来进行同步，实现对竞争资源的互斥访问。

2、独占锁（可重入的互斥锁）

互斥，即在同一时间点，只能被一个线程持有；可重入，即可以被单个线程多次获取。什么意思呢？根据锁的获取机制，它分为“公平锁”和“非公平锁”。Java 中通过 ReentrantLock 实现独占锁，默认为非公平锁。

3、公平锁

是按照通过 CLH 等待线程按照先来先得的规则，线程依次排队，公平的获取锁，是独占锁的一种。Java 中，ReetrantLock 中有一个 Sync 类型的成员变量 sync，它的实例为 FairSync 类型的时候，ReetrantLock 为公平锁。设置 sync 为 FairSync 类型，只需——Lock lock = new ReetrantLock(true)。

4、非公平锁

是当线程要获取锁时，它会无视 CLH 等待队列而直接获取锁。ReetrantLock 默认为非公平锁，或——Lock lock = new ReetrantLock(false)。

5、共享锁

能被多个线程同时获取、共享的锁。即多个线程都可以获取该锁，对该锁对象进行处理。典型的就是读锁——ReentrantReadWriteLock.ReadLock。即多个线程都可以读它，而且不影响其他线程对它的读，但是大家都不能修改它。CyclicBarrier， CountDownLatch 和 Semaphore 也都是共享锁。

22、介绍一下锁升级过程

synchronized 锁的四种状态是在 jdk1.6 之后引入的,分别为:(无锁->偏向锁->轻量级锁->重量级锁) 这几个状态会随着竞争情况逐渐升级。

a:无锁

无锁的特点就是修改操作在循环内进行，线程会不断的尝试修改共享资源。如果没有冲突就修改成功并退出，否则就会继续循环尝试。也就是 CAS（CAS 是基于无锁机制实现的）。

b.偏向锁

偏向锁是指一段同步代码一直被一个线程所访问，那么该线程会自动获取锁，降低获取锁的代价。偏向锁只有遇到其他线程尝试竞争偏向锁时，持有偏向锁的线程才会释放锁，线程不会主动释放偏向锁。

c.轻量级锁

是指当锁是偏向锁的时候，被另外的线程所访问，偏向锁就会升级为轻量级锁，其他线程会通过自旋的形式尝试获取锁，不会阻塞，从而提高性能。

若当前只有一个等待线程，则该线程通过自旋进行等待。但是当自旋超过一定的次数，或者一个线程在持有锁，一个在自旋，又有第三个来访时，轻量级锁升级为重量级锁。

d.重量级锁

在轻量级锁状态下,如果有第三个来访时,就会自动升级成重量级锁

23、介绍一下 ReEntrantLock 底层实现，介绍一下 AQS**********

ReentrantLock 的底层实现机制是 AQS(Abstract Queued Synchronizer 抽象队列同步器)。AQS 没有锁之类的概念，它有个 state 变量，是个 int 类型，为了好理解，可以把 state 当成锁，AQS 围绕 state 提供两种基本操作“获取”和“释放”，有条双向队列存放阻塞的等待线程。AQS 的功能可以分为独占和共享，ReentrantLock 实现了独占功能（每次只能有一个线程能持有锁）。

AQS：AbstractQuenedSynchronizer 抽象的队列式同步器。是除了 java 自带的 synchronized 关键字之外的锁机制。

24、怎么查找一个文件里的某一个字符串的位置
Pattern 和 Matcher 。编译子串,创建 Matcher 对象，依照正则表达式，该对象可以与任意字符序列匹配

25、HashMap，源码级别的问了，包括为什么线程不安全!!!!!!!!!!
HashMap 底层是链表模式存储的。当链表复杂度达到一定层次后，会采用红黑树的形式取数。一个 key 过来后，
先根据 key 的 hashcode 对比 treenode 获取位置，之后再进行取数。
因为 hashmap 是异步操作，所以线程不安全
红黑树的好处就是可以切换。通过旋转或者变色维持树的平衡性

26、死锁
评注:这问题一面的时候问过了，嗯，凸显！
回答: 死锁是指两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去，如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则就会因争夺有限的资源而陷入死锁。

产生死锁的原因主要是：
因为系统资源不足。
进程运行推进的顺序不合适。
资源分配不当等。

27、Synchronized 和 ReentrantLock 锁机制，怎么判断重入锁的，会不会是死锁?
评注:并发基础问题，懂并发编程的，应该都会。
回答:
先答区别:
API 方面:synchronized 既可以修饰方法，也可以修饰代码块。ReentrantLock 只能在方法体中使用。
公平锁:synchronized 的锁是非公平锁，ReentrantLock 默认情况下也是非公平锁，但可以通过带有布尔值的构造函数要求使用公平锁。

29、进程之间如何保证同步?
临界区 适用同步方法，通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问
互斥量 为协调共同对一个共享资源的单独访问而设计的。
信号量 为控制一个具有有限数量用户资源而设计，它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目。互斥量是信号量的一种特殊情况，当信号量的最大资源数=1 就是互斥量了
事件 用来通知线程有一些事件已发生，从而启动后继任务的开始

针对临界值，四个规则：有空让进，无空等待，多个则一，让权等待，有限等待

30、分布式锁
评注:此题问的没头没尾的，分布式锁可以问的点很多，比如实现方式啊？性能差距啊?
回答:这题如果要详答，看我的另一篇文章《分布式之抉择分布式锁》
分布式锁有三种实现方式：数据库、缓存、Zookeeper，这里我直接罗列一下各种锁的对比吧
从理解的难易程度角度（从低到高）
数据库 > 缓存 > Zookeeper
从实现的复杂性角度（从低到高）
Zookeeper >= 缓存 > 数据库
从性能角度（从高到低）
缓存 > Zookeeper >= 数据库
从可靠性角度（从高到低）
Zookeeper > 缓存 > 数据库

31、对象 GC
java 堆，方法区，本地方法区，虚拟机栈，程序计数器
程序运行主要在 java 堆中，java 数组，对象存在区域。
方法区存放静态变量，常量等
垃圾回收算法：

31、垃圾回收算法
评注:一面问过，把一面的回答贴过来!
回答:
标记 – 清除算法、标记整理算法、复制算法、压缩算法

32、JVM 参数
标准参数 – 举例：–jar，–verbose:gc 输出每次 gc 情况
非标准参数 –X 举例：–Xint，解释执行模式，所有的字节码将被直接执行，而不会编译成本地码；
非 stable 参是 –XX 举例：–XX:+PrintGCDetails 开启 GC 详细信息 –XX:MaxPermSize=64m 老生代对象能占用内存的最大值

33、OOM 出现的有哪些场景？为什么会发生?
评注:常规题，只是情况太多了!
回答:
OOM for Heap (java.lang.OutOfMemoryError: Java heap space):heap 的最大值不满足需要，将设置 heap 的最大值调高即可。
OOM for StackOverflowError (Exception in thread "main" java.lang.StackOverflowError):如果线程请求的栈深度大于虚拟机所允许的最大深度，将抛出 StackOverflowError 异常。
OOM for GC (java.lang.OutOfMemoryError: GC overhead limit exceeded):此 OOM 是由于 JVM 在 GC 时，对象过多，导致内存溢出。
OOM for native thread created (java.lang.OutOfMemoryError: unable to create new native thread):这个异常问题本质原因是我们创建了太多的线程，而能创建的线程数是有限制的，导致了异常的发生。
OOM for allocate huge array (Exception in thread "main": java.lang.OutOfMemoryError: Requested array size exceeds VM limit):此类信息表明应用程序试图分配一个大于堆大小的数组。例如，如果应用程序 new 一个数组对象，大小为 512M，但是最大堆大小为 256M，因此 OutOfMemoryError 会抛出，因为数组的大小超过虚拟机的限制。
OOM for small swap (Exception in thread "main": java.lang.OutOfMemoryError: request bytes for . Out of swap space?)：抛出这类错误，是由于从 native 堆中分配内存失败，并且堆内