**一面：**

HashMap 实现原理，ConcurrentHashMap 实现原理(重点题P20)

红黑树，为什么允许局部不平衡

答：**严格的平衡二叉树**，平衡条件必须满足(**所有结点的左右子树高度差不超过1**)。不管我们是执行插入还是删除操作，只要不满足上面的条件，就要通过旋转来保存平衡，而因为旋转非常**耗时**，由此我们可以知道**AVL树适合用于插入与删除次数比较少，但查找多的情况**。相对于要求严格的AVL树来说，红黑树的旋转次数少，插入最多两次旋转，删除最多三次旋转，所以对于插入，删除操作较多的情况下，搜索较少的，我们就用红黑树。

TCP，UDP 区别，为什么可靠和不可靠

答：没有ack那些三次握手、也没重发什么

一次 HTTP 请求的全过程，包括域名解析、定位主机等(重点题P25)

TCP 三次握手

MySQL 事务是什么？四大特性，四大隔离级别

ConcurrentHashMap 和 Hashtable 区别(java集合 P16)

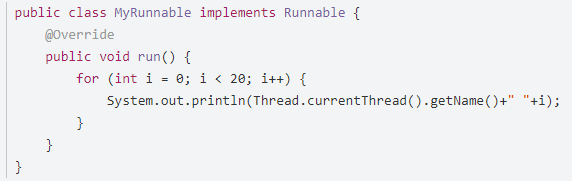
spring IOC 和 AOP，以及各有什么优点

有哪几种常用的线程池（多线程P6）

什么情况下使用 Runnable 和 Thread 创建线程，Runnable 和 Callable 的区别

答：Thread和Runnable的区别：1.避免Java**单继承的局限性**，当一个类已有了父类，假如这个类有多线程操作，就只能实现Runnable接口；2. 解耦合，Thread类包含对线程的各种操作，继承就把自己都交给了Thread类，耦合在一起，不需要别的操作，就只实现Runnable接口就好，将自己和Thread类分离。





相同点：1）都是接口2）都可以编写多线程程序3）都采用Thread.start()启动线程。

不同点：1）Runnable没有返回值；Callable可以返回执行结果，是个泛型，和Future、FutureTask配合可以用来获取异步执行的结果

2)Callable接口的call()方法允许抛出异常；Runnable的run()方法异常只能在内部消化，不能往上继续抛

注：Callalbe接口支持返回执行结果，需要调用**FutureTask.get()**得到，此方法会阻塞主进程的继续往下执行，如果不调用不会阻塞。

线程方法中的异常如何处理，副线程可以捕获到吗

答：正常情况下，如果不做特殊的处理，在主线程中是不能够捕获到子线程中的异常的。

1）设置每一个线程执行时候的异常处理。如果每一个线程的异常处理相同，使用Thread的静态方法默认异常处理器，否则**为线程设置不同的“未捕获异常处理器” UncaughtExceptionHandler；**

2）最简单是在子线程的执行方法中，把可能发生异常的地方，用try ... catch ... 语句包起来

3）使用线程池提交一个能获取到返回信息的方法，也就是ExecutorService.submit(Callable)，在submit之后可以获得一个线程执行结果的Future对象，而如果子线程中发生了异常，通过**future.get()**获取返回值时，可以捕获到ExecutionException异常，从而知道子线程中发生了异常。

synchronized 和锁的区别，什么情况下使用 synchronized 和 ReentrantLock

JVM 的对象分配在哪个区，Class 对象分配在哪个区

答：堆内存是用来存放由new创建的对象和数组，即动态申请的内存都存放在堆内存；

栈内存是用来存放在函数中定义的一些基本类型的变量和对象的引用变量。

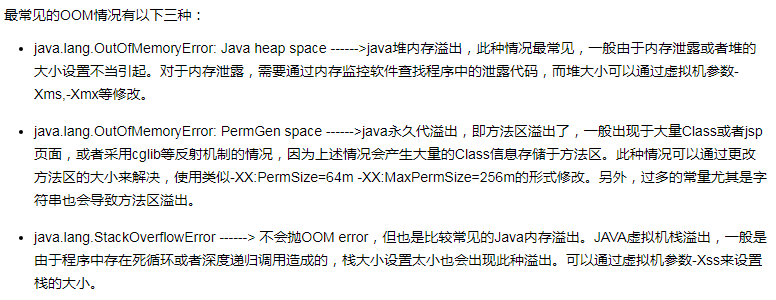
例子：局部变量存放在栈；new函数和函数申请的内存在堆；函数调用参数，函数返回值，函数返回地址存放在栈

**二面：**

常用的设计模式介绍：单例模式、装饰者模式等

Java 会出现内存溢出吗？什么情况下会出现？

答：如果某些资源没有关闭（例如I/O文件和数据库连接没有关闭）、垃圾回收堆空间不足。



双亲委派模型，为什么这样做？（JAVA P2）

对象什么情况下进入老年代？

1.**大对象**：所谓的大对象是指需要大量连续内存空间的java对象，最典型的大对象就是**那种很长的字符串以及数组**，大对象对虚拟机的内存分配就是坏消息，尤其是一些朝生夕灭的短命大对象，写程序时应避免。

2.**长期存活的对象**：虚拟机给每个对象定义了一个对象年龄(Age)计数器，如果对象在Eden出生并经过第一次Minor GC后仍然存活，并且能被Survivor容纳的话，将被移动到Survivor空间中，并且对象年龄设为1,。对象在**Survivor区**中每熬过一次Minor GC，年龄就增加1，当他的**年龄增加到一定程度(默认是15岁)**， 就将会被晋升到老年代中。对象晋升到老年代的年龄阈值，可以通过参数-XX:MaxTenuringThreshold设置。

3.**动态对象年龄判定**：为了能更好地适应不同程度的内存状况，虚拟机并不是永远地要求对象的年龄必须达到了MaxTenuringThreshold才能晋升到老年代，**如果在Survivor空间中相同年龄的所有对象大小的总和大于Survivor空间的一半，年龄大于或等于年龄的对象就可以直接进入老年代**，无须等到MaxTenuringThreshold中要求的年龄。

快速排序说一下过程（数据结构 P1）

BIO、NIO（如何实现的）、AIO（NIO P2）

消息中间件有哪些？他们之间的优劣势？

Redis，持久化框架（Redis P11）

栈和队列

垃圾回收算法（JVM P14）

MySQL 的索引（MySQL P1）

Tomcat(JVM) 类加载器

OOM 内存泄漏，什么情况下会出现，如何排查

答：CPU过高怎么解决：

1.top 查看哪个进程Id cpu比较高

2）然后通过 top-Hp<pid>来查看该进程中有哪些线程CPU过高，一般超过80%就是比较高的。这样我们就能得到CPU消耗比较高的线程id；

3）接着通过该线程id转化为十六进制表示（printf "%x\n" tid tid是线程Id 输出：7082

），在 jstack日志中查看当前线程具体的堆栈信息。（jstack pid |grep tid -A 30 ：pid进程Id tid线程Id 可以具体查看堆栈信息）

4）之后可以区分导致CPU过高的原因具体是Full GC次数过多还是代码中有比较耗时的计算了。

如果是Full GC次数过多，那么通过 jstack得到的线程信息会是类似于VM Thread之类的线程，而如果是代码中有比较耗时的计算，那么我们得到的就是一个线程的具体堆栈信息。

5）我们可以根据堆栈信息，直接定位到UserController的34行，查看代码中具体是什么原因导致计算量如此之高。

总结下：最后，总结下排查CPU故障的方法和技巧有哪些：

jstack：Java提供的命令。可以查看某个进程的当前线程栈运行情况。根据这个命令的输出可以定位某个进程的所有线程的当前运行状态、运行代码，以及是否死锁等等。

内存过高怎么解决：

1）用 top命令查看系统CPU的占用情况，找到CPU占用量最大的进程id；

2）用top -Hp pid 查看该进程的各个线程运行情况，找到最大的线程id；

3）用jstack查看这个线程大的原因，出现VM Thread代表垃圾回收的线程，需将线程id转换为16进制查看；

4）用jstack-gcutil pid 1000 线程id查看实时GC情况，这里FGC指的是Full GC数量，进一步证实了是由于内存溢出导致的系统缓慢；

5）用jmap -dump:format=b,file=temp.dump pid 下载dump文件到本地，通过MAT分析对象的引用链的方式来定位具体频繁创建对象的地方，进行处理即可。

jmap命令——jmap命令有下面几种常用的用法：

jmap -histo:live [pid] >a.log，然后分析具体的对象数目和占用内存大小，从而定位代码。

jmap -dump:live,format=b,file=xxx.xxx [pid]，然后利用MAT工具分析是否存在内存泄漏等等。

其中，jmap -histo:live [pid] 可以查看当前Java进程创建的活跃对象数目和占用内存大小。jmap -dump:live,format=b,file=xxx.xxx [pid] 则可以将当前Java进程的内存占用情况导出来，方便用专门的内存分析工具（例如：MAT）来分析。这个命令对于分析是否有内存泄漏很有帮助。

**三面**：

介绍你实践的性能优化案例，以及你的优化思路

SQL 慢查询的优化方案，索引和表的优化方案。（MySQL P18）

请描述一致 hash 算法

答：一致性Hash算法将整个哈希值空间组织成一个虚拟的圆环，如假设某哈希函数H的值空间为**0-2^32-1**（即哈希值是一个32位无符号整形）。在一致性Hash算法中，如果一台服务器不可用，则受影响的数据仅仅是此服务器到其环空间中前一台服务器（即沿着**逆时针**方向行走遇到的第一台服务器）之间数据，其它不会受到影响。如果增加一台服务器，则受影响的数据仅仅是新服务器到其环空间中前一台服务器（即沿着**逆时针**方向行走遇到的第一台服务器）之间数据，其它数据也不会受到影响。**一致性Hash算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。**当服务节点少比如2个，大量数据集中到其中一个，导致数据倾斜，这时可以建立**虚拟节点**映射，加上虚拟编号#，比如32个。

分布式 session 的共享方案有哪些，有什么优劣势

答：1.session同步：**多个服务器之间同步session**，这样可以保证每个服务器上都有全部的session信息，不过当服务器数量比较多的时候，同步是会有延迟甚至同步失败；

2. **使用Nginx**（或其他复杂均衡软硬件）**中的ip绑定策略**，**同一个ip只能在指定的同一个机器访问**，但是这样做风险也比较大，而且也是去了负载均衡的意义；

3. **把session放到Redis中存储**，虽然架构上变得复杂，并且需要多访问一次Redis，但是这种方案带来的好处也是很大的：实现session共享，可以水平扩展（增加Redis服务器），服务器重启session不丢失（不过也要注意session在Redis中的刷新/失效机制），不仅可以跨服务器session共享，甚至可以跨平台（例如网页端和APP端）。

高并发情况，系统的优化方案有哪些，以及优先级排序。

**答：硬件、操作系统层面的优化：**多核CPU、增加内存、固态硬盘；集群负载均衡，硬件负载F5；数据进行动静分离。

**应用系统架构优化：**分布式应用间高效的RPC通讯框架或微服务解耦、缓存、多线程（锁的控制）、mq。

**数据库优化**：拆分、读写分离、索引、sql规范

**运维优化**：做好日志记录、安全预警、灾备替换等

**一面：**

ArrayList 和 linkedlist 区别。ArrayList 是否会越界。

ArrayList 和 hashset 有何区别。hashset 存的数是有序的么。

答：Hashset跟hashmap底层一样，只是不允许key重复，先计算hashcode，重复可以，再看equals，不等就尾加入，相等不加。不能保证有序。

volatile 和 synchronized 区别（多线程 P2）

Redis 的数据结构

Redis 是基于内存的么

答：redis是一个内存数据库, 所有数据基本上都存在于内存当中, 会定时以追加或者快照的方式刷新到硬盘中

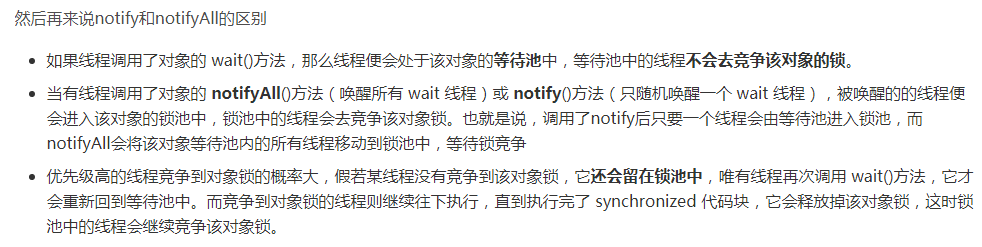
Redis 的 list zset 的底层实现

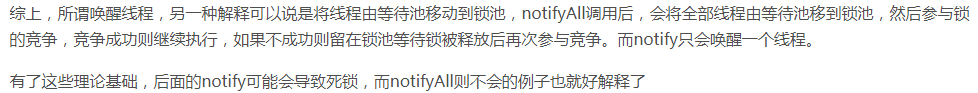
http 和 https 的区别（tcp P7）

手写冒泡

手写单例包括多线程下

Java 线程间怎么实现同步，notify()与 notifyAll()的区别





数据库的悲观锁和乐观锁应用场景。

排序算法的复杂度，快速排序非递归实现。

海量数据过滤，黑名单过滤一个url。

答：布隆过滤器。给url根据算法设计个key，存hashmap，查询o（1）

**二面：**

list set map 底层使用什么实现的有哪些典型实现

hashmap 扩容是怎么扩容的，为什么是 2 的幂

concurrenthashmap 为什么线程安全，采用了什么措施应对高并发

线程池的参数有什么意义

Springmvc 请求流程

Spring IOC，autowired 如何实现（自动装配）

答：提到@Autowired我们一般都知道叫依赖注入。为我们注入一个定义好的bean。java的注解实现的核心技术是反射。在容器启动，为对象赋值的时候，遇到@Autowired注解，**会用后置处理器机制，来创建属性的实例，然后再利用反射机制，将实例化好的属性，赋值给对象上，这就是Autowired的原理。**将Spring容器中的bean自动的和我们需要这个bean的类组装在一起协同使用。

Spring boot

SpringClound 的基本架构设计

Dubbo 和 SpringClound 的区别在哪里，优劣势

**三面：**

分布式架构设计哪方面比较熟悉

讲讲你对 CDN 的了解，与分布式缓存和本地缓存的区别

答：CDN就是在**用户和**我们的**服务器之间加一个缓存机制**，通过这个缓存机制**动态获取IP地址根据地理位置，让用户到最近的服务器访问**。CDN能够缓存JavaScript脚本，css样式表，图片，图标，Flash等静态资源文件（不包括html页面），这些静态资源文件的访问频率很高，将其缓存在CDN可以极大地提高网站的访问速度。

本地缓存的优势除了缓存这个点之外还有一个就是没有网络开销，在大并发量时用好本地缓存很重要；分布式缓存比如Redis，除了缓存这个优势的点之外，能够无限扩容量是另外一个比较重要的优势点，结合这个去在业务中使用缓存是很重要的。

多线程和高并发有什么区别

答：高并发是指较多用户同时访问服务。高并发可以由多线程实现，但是多线程不代表就是高并发。

高并发下有哪些常用的技术解决方案，举三个高并发场景设计例子

说一个你对 JVM 优化的实际案例，包括实际步骤和方法

* -Xmx（**最大堆内存**） 与 –Xms（**初始堆内存**） 相同以避免JVM反复重新申请内存。-Xmx 的大小约等于系统内存大小的一半，即充分利用系统资源，又给予系统安全运行的空间。
* -Xmn（**年轻代**）1256m 设置年轻代大小为1256MB。此值对系统性能影响较大，Sun官方推荐配置年轻代大小为整个堆的3/8。
* -Xss128k 设置较小的线程栈以支持创建更多的线程，支持海量访问，并提升系统性能。
* -XX:**SurvivorRatio**=6 设置年轻代中Eden区与Survivor区的比值。系统默认是8，根据经验设置为6，则2个Survivor区与1个Eden区的比值为2:6，一个Survivor区占整个年轻代的1/8。
* -XX:**ParallelGCThreads**=8 配置并行收集器的线程数，即同时8个线程一起进行垃圾回收。此值一般配置为与CPU数目相等。
* -XX:+**UseConcMarkSweepGC** 设置年老代为并发收集。CMS（ConcMarkSweepGC）收集的目标是尽量减少应用的暂停时间，减少Full GC发生的几率，利用和应用程序线程并发的垃圾回收线程来标记清除年老代内存，适用于应用中存在比较多的长生命周期对象的情况。

Docker 有使用过和了解吗？Docker 和 JVM 的区别是什么？

Docker 的基本架构和使用场景？

负载均衡有接触过哪些开源框架，优劣势是什么？

答：LVS、Nginx

数据库分库分表需要怎样来实现？（MySQL P22）

数据库端的常用优化策略？

如果让你来设计秒杀系统，你的设计思路是什么，为什么要这样设计？

面试总结

java 的基础知识点，主要围绕在集合类和多线程等：ArrayList、LinkedList、HashSet、HashpMap

的数据结果，以及如何扩容、以及 ConcurrentHashMap 相关的多线程安全等。

JVM 的内存分配、几个常见的垃圾回收算法以及原理、还有对应的 JVM 优化参数需要牢记。

网络：TCP 的三次握手等网络都必问，重点掌握网络协议。

Redis：作为分布式缓存的主力，基本也是 BAT 每次必考，重点是 Redis 的数据结构、内存、算法、持久化，以及与别的缓存 memcached 的优劣势。

多线程：状态流转、多线程的实现，以及与高并发的区别等。

Spring 框架问得是最多的，BAT 非常喜欢问，重点掌握。

最后就是分布式架构设计

常用的分布式架构设计方案：**单点登录**、分布式缓存、存储、消息的选型，还有就是数据库端的优化方案(需要提前了解)。

最好能提前了解深入一个类似秒杀这样的项目，如果面试官问到类似的项目，你能把设计思路讲出来，这对你的面试结果是很大的加分项。

**一面**

3.Spring 的 bean 的作用域？（比如：singleton，prototype 等）

4.Spring 的 IOC 实现原理？没有无参构造函数能实例化吗？有参构造函数注入？（xml 配置）

5.通过反射，谈到了**方法区**，然后，类加载机制？

答：方法区(Method Area)是 Java虚拟机规范 中定义的运行时数据区域之一，它与堆(heap)一样在线程之间共享。

方法区主要用来存储运行时常量池、静态变量、类信息、JIT编译后的代码等数据。方法区是 JVM 规范所描述的抽象概念，在实际的 JVM 实现中，它不一定是由单一的特殊区域所实现，不同的实现可以放在不同的地方。

6.synchronized 的实现原理？Volatile 能保证原子性吗？为什么？

7.hashmap 和 concurrenthashmap 的 size 方法怎么实现的

8.JVM 的调优参数？（-Xmn，-Xms 等具体参数设置）

9.线程池优点，参数，如果我想实现 newSingleThreadPoll，应该怎么配置，构造方法传什么参数

10.mysql 死锁，怎么解决，如果不要求执行顺序，死锁怎么解决

11.ioc 和 aop 原理

12.线程的五态？转化过程？

答：每个线程都需经历新生、就绪、运行、阻塞和死亡五种状态，线程从新生到死亡的状态变化称为生命周期。

用new运算符和Thread类或其子类建立一个线程对象后，该线程就处于新生状态。

调用start，线程进入就绪状态，等待系统的调度（时间片轮转调度）。

当系统调度，进入运行状态。

正常结束或者异常退出，进程进入死亡状态。

处于运行状态的线程若遇到sleep或wait或io阻塞，线程处于等待状态，sleep结束或io结束，转为就绪；wait需要notify()/notifyALL来唤醒线程，唤醒后的线程处于锁定状态，获取了“同步锁”，之后，线程才转为就绪状态。

13.TCP 三次握手，为什么三次握手？

14.JVM 内存分区？（主存，工作内存，堆，栈。。。。）

15.讲一下 GC？

16.为什么要用老年代和新生代？

17.新生代进入老生代的情况？

18.新生代的分区？

答：1个eden，2个survivor

**二面**

变着法的问了一大堆线程池的知识 （主要考对应的参数）

java 内存模型

lock 和 synchronized 的区别

reentrantlock 的实现

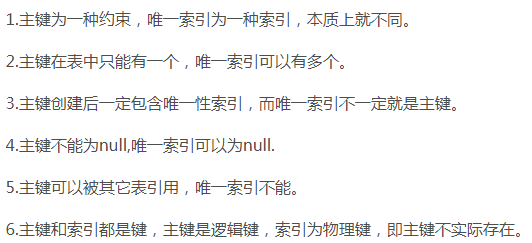
hashmap 和 concurrenthashmap

B+树和 B-树的区别

复合索引

聚集索引和非聚集索引的区别？

数据库索引 主键和唯一索引有什么区别



索引失效条件，什么时候该建立索引

innDB 和 MyISAM 的区别？

线程安全（阻塞同步(悲观锁加锁)，非阻塞同步(乐观CAS)，无同步）

**三面**

主要高并发和分布式架构设计

服务器模型以及之间的区别

线程池的设计

线程池如何做负载均衡

如何实现线程调度算法



复合索引是如何实现的？

如何设计单点登录，单点登录的原理（就是session共享）

答：到用户在单点系统登录后，单点登录系统会将用户的身份信息，存放到redis中，并且给客户端种一个token, 这样任何一个需要登录的系统，主要配置一个拦截器然后用客户提交的token调用单点登录系统换取用户信息，从而完成了分布式的seesion共享问题。

redis 缓存和 memcached 缓存的区别，以及各自的优劣势

大型高并发网站如何做性能优化：Web 性能、数据库性能、应用服务器性能等。

实践中如何优化 MySQL:SQL 语句及索引的优化、数据库表结构的优化、系统配置的优化、硬件的优化

分库分表和读写分离如何设计

微服务架构：dubbo 和 springcloud 的区别，以及各自对应的使用场景。

**淘宝一面:**

**面试题目**

1）讲一下 JVM

2）讲一下 JVM 的分代回收以及具体算法

3）将一下 JVM 的垃圾收集器，G1 和 CMS 有啥区别？

4）讲一下一个变量从产生到结束所经历的过程，讲一下字符串常量的过程？

5）将一下线程安全问题产生的原因？

6）讲一下乐观锁和悲观锁

7）乐观锁是怎么保证一致性的

使用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。何谓数据版本？即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的 “version” 字段来实现。当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的version值进行比对，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据。

8）Integer 和 int 有啥区别，integer 中有哪些特殊的函数？

9）讲一下数据库的隔离等级

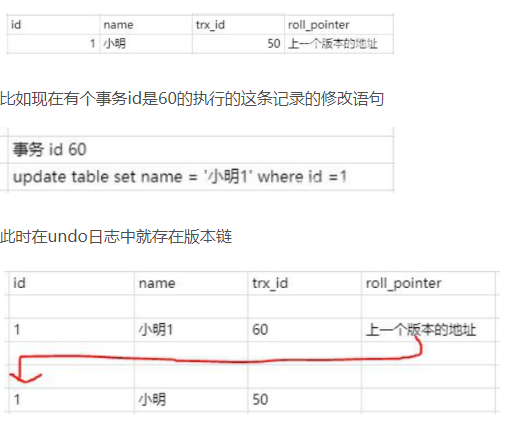
10）说一下 MVCC

就是多版本并发控制。

**版本链**

在InnoDB引擎表中，它的聚簇索引记录中有两个必要的隐藏列：

* trx\_id这个id用来存储的每次对某条聚簇索引记录进行修改的时候的事务id。
* roll\_pointer每次对哪条聚簇索引记录有修改的时候，都会把老版本写入undo日志中。这个roll\_pointer就是存了一个指针，它指向这条聚簇索引记录的上一个版本的位置，通过它来获得上一个版本的记录信息。(注意插入操作的undo日志没有这个属性，因为它没有老版本)



**ReadView**

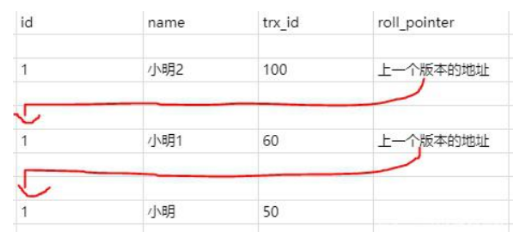
ReadView中主要就是有个列表来存储我们系统中当前活跃着的读写事务，也就是begin了还未提交的事务。通过这个列表来判断记录的某个版本是否对当前事务可见。假设当前列表里的事务id为[80,100]。

* 如果你要访问的记录版本的事务id为50，比当前列表最小的id80小，那说明这个事务在之前就提交了，所以对当前活动的事务来说是可访问的。
* 如果你要访问的记录版本的事务id为70,发现此事务在列表id最大值和最小值之间，那就再判断一下是否在列表内，如果在那就说明此事务还未提交，所以版本不能被访问。如果不在那说明事务已经提交，所以版本可以被访问。
* 如果你要访问的记录版本的事务id为110，那比事务列表最大id100都大，那说明这个版本是在ReadView生成之后才发生的，所以不能被访问。

这些记录都是去版本链里面找的，先找最近记录，如果最近这一条记录事务id不符合条件，不可见的话，再去找上一个版本再比较当前事务的id和这个版本事务id看能不能访问，以此类推直到返回可见的版本或者结束。

举个例子 ，在已提交读隔离级别下：

比如此时有一个事务id为100的事务，修改了name,使得的name等于小明2，但是事务还没提交。则此时的版本链是



那此时另一个事务发起了select 语句要查询id为1的记录，那此时生成的ReadView 列表只有[100]。那就去版本链去找了，首先肯定找最近的一条，发现trx\_id是100,也就是name为小明2的那条记录，发现在列表内，所以不能访问。

这时候就通过指针继续找下一条，name为小明1的记录，发现trx\_id是60，小于列表中的最小id,所以可以访问，直接访问结果为小明1。

那这时候我们把事务id为100的事务提交了，并且新建了一个事务id为110也修改id为1的记录，并且不提交事务



这时候之前那个select事务又执行了一次查询,要查询id为1的记录。

这个时候关键的地方来了

如果你是**已提交读隔离级别，这时候你会重新一个ReadView**，那你的活动事务列表中的值就变了，变成了[110]。

按照上的说法，你去版本链通过trx\_id对比查找到合适的结果就是小明2。

如果你是**可重复读隔离级别，这时候你的ReadView还是第一次select时候生成的ReadView**,也就是列表的值还是[100]。所以select的结果是小明1。所以第二次select结果和第一次一样，所以叫可重复读！

**也就是说已提交读隔离级别下的事务在每次查询的开始都会生成一个独立的ReadView,而可重复读隔离级别则在第一次读的时候生成一个ReadView，之后的读都复用之前的ReadView。**

这就是Mysql的MVCC,通过版本链，实现多版本，可并发读-写，写-读。通过ReadView生成策略的不同实现不同的隔离级别。

11）说一聚簇索引和非聚簇索引的有什么不同

**淘宝二面:**

1、问了冒泡排序，快排，和归并排序及优缺点和优化

2，网络方面有 osi 七层，tcp/ip 五层，分别有哪些协议及作用

3，爬虫用的什么数据结构

4、tcp 的流量控制和拥塞控制

5，mysql 用的什么存储引擎，这个存储引擎用的什么数据结构 ，有哪些优缺点，怎么使用

6，jvm 的垃圾回收机制和垃圾收集器

7、spring 当中事物的隔离级别

8、jdk1.8 concurrenthashmap 的新的特性，有没有看过源码

9、 threadlocal 了解吗

10，问了 redis 的一些问题，项目中有(扩容，失效 key 清理策略等)

11，剩下的都是项目的东西(kafka filebeat elk 原理，主从选举，复制等)

12，后面扩展的问了一些大数据相关的，问我一些大数据处理框架是否有了解

整个过程四十分钟左右

**淘宝三面**

1）讲一下 Spring AOP 和 IOC 的底层实现

2）说一下 hashcode 的作用？HashMap 的底层实现？HashMap 和 HashTable 的区别

3）说一下 concurrentHashMap 和 hashTable 在性能上的区别？以及这种差异形成的原因

4）讲一下堆以及堆排序

5）说一下 B+tree 和二叉搜索树的区别？说一下二叉搜索树和 AVL 树、红黑树之间的差别

6）给你两个文件（字符串形式的）如何找出他们之间的不同地方？

答：大文件分成1000个小文件，把其中一个存到hashmap，遍历另一个，依次比较找对应位置没有的。

7）你刚刚说的能怎么优化？

**淘宝四面 交叉面**

本来以为三面结束就是 hr 面了，又收到一面交叉面

给你 50 亿行字符串，机器 4G 内存（只能一台机器），找出重复次数最多的那行字符串？（以行为单位，每行不超过 10 个字符）50亿\*10=50G

2.设计一个算法，实现两个10g大文件在 10m的内存中将两个大文件中重复的放进第三个文件。分治算法

快速排序的平均复杂多少？最坏情况是什么？（这个题估计就是缓和一下尴尬的气氛）

**支付宝一面**

介绍一下自己。

项目参与的核心设计有哪些

ArrayList 和 LinkedList 底层

HashMap 及线程安全的 ConcurrentHashMap，以及各自优劣势

Java 如何实现线程安全

Synchronized 和 Lock 哪个更好？

HashMap 中的 get()方法是如何实现的？

HashMap 可以用在哪些场景？

JVM，垃圾回收机制，内存划分等

SQL 优化，常用的索引？

还有什么问题需要问的。

**支付宝二面**

没有自我介绍，直接问做过哪些 Java 开发相关的项目。

对哪些技术比较熟悉？

多线程状态图，状态如何流转？（线程5个状态转换）

死锁，死锁原因

页锁、乐观锁、悲观锁？

乐观锁如何保证线程安全？

用过线程池吗，对应的好处，如何用？

两个 10G 的文件，里面是一些 url，内存只有 1G，如何将这两个文件合并，找到相同的 url？

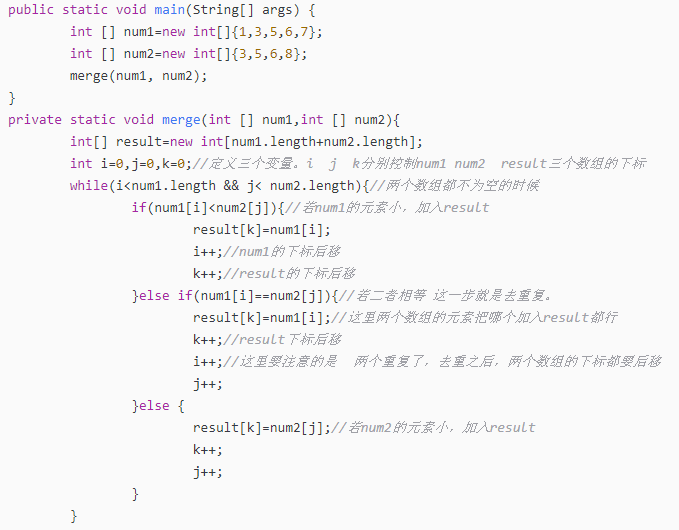
**1000 个多并发线程，10 台机器，每台机器 4 核的，设计线程池大小**。

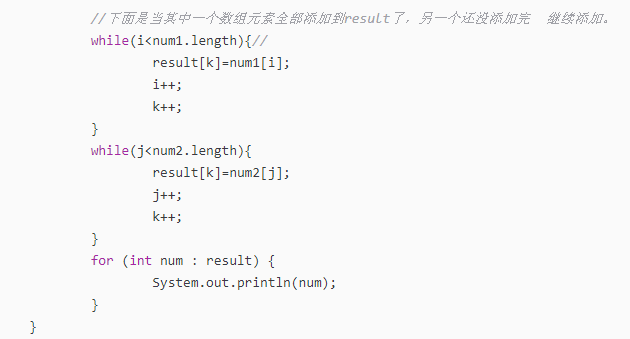
答：2N+1cpu

一个线程 总时间= 空闲时间+真正执行时间

N**核服务器**，通过执行业务的单线程分析出**本地计算时间**为x，**等待时间**为y，则**工作线程数**（线程池线程数）设置为 N\*(x+y)/x，能让CPU的利用率最大化。

代码题：两个有序数组，数组中存在重复数字，合并成一个有序数组，去除重复数字。





说一下自己的优点。

**支付宝三面**

jvm 性能调优都做了什么

数据库性能调优如何做

分布式系统原理：CAP，最终一致性，幂等操作等

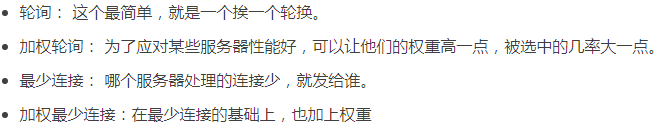
高并发情况下，我们系统是如何支撑大量的请求的

集群如何同步会话状态（session共享）

常用 NOSQL，有做过比较?

什么情况会出现雪崩，以及如何应对？

负载均衡的原理



数据库事务属性

**美团内推（一面+二面+三面）面试题**

Redis 介绍

了解 redis 源码么

了解 redis 集群么

Hashmap 的原理，增删的情况后端数据结构如何位移

hashmap 容量为什么是 2 的幂次

第一个截图是向HashMap中添加元素putVal()方法的部分源码，可以看出，向集合中添加元素时，会使用(n - 1) & hash的计算方法来得出该元素在集合中的位置；而第二个截图是HashMap扩容时调用resize()方法中的部分源码，可以看出会新建一个tab，然后遍历旧的tab，将旧的元素进过e.hash & (newCap - 1)的计算添加进新的tab中，也就是(n - 1) & hash的计算方法，其中n是集合的容量，hash是添加的元素进过hash函数计算出来的hash值。

  HashMap的容量为什么是2的n次幂，和这个(n - 1) & hash的计算方法有着千丝万缕的关系，符号&是按位与的计算，这是位运算，计算机能直接运算，特别高效，按位与&的计算方法是，只有当对应位置的数据都为1时，运算结果也为1，当HashMap的容量是2的n次幂时，(n-1)的2进制也就是1111111\*\*\*111这样形式的，这样与添加元素的hash值进行位运算时，能够充分的散列，使得添加的元素均匀分布在HashMap的每个位置上，减少hash碰撞，

hashset 的源码

我看了一下HashSet的底层实现，发现被骗了，因为我发现它的底层是依托HashMap来实现的。这就让我很气了啊，好吧，我不气了，还是来看看它怎么来实现的吧，首先我们知道，HashSet是实现了Set接口的，所以，我们说，它是不允许有重复元素的，那么它是怎么保证没有重复元素的呐，其实它的原理就是hashMap判断两个对象是否是同一个元素是一样的。简单先说一下，就是hashCode和equals,首先去table表上去看看有没有相同的key值，没有的话就存，有的话去比较equals，你可能会问，为什么有的话，还要去比较equals，因为我告诉你，是因为即使两个完全不相同的对象它的hashCode也有可能相同。但是不相同的两个hashCode一定不相同。

object 类你知道的方法

hashcode 和 equals

你重写过 hashcode 和 equals 么，要注意什么

注意：1.equals与hashCode的定义必须一致，两个对象equals为true，就必须有相同的hashCode。反之，如果两个对象的hashcode相同，equals不一定相同。2.当重写equals方法时，一定要同时把hashcode方法一并重写，因为要保证在实现hash表的扩展性

假设现在一个学生类，有学号和姓名，我现在 hashcode 方法重写的时候，只将学号参与计算，会出现什么情况？

往 set 里面 put 一个学生对象，然后将这个学生对象的学号改了，再 put 进去，可以放进set 么？并讲出为什么

Redis 的持久化？有哪些方式，原理是什么？

讲一下稳定的排序算法和不稳定的排序算法

稳定的排序算法：通俗地讲就是能保证排序前2个相等的数其在序列的前后位置顺序和排序后它们两个的前后位置顺序相同。在简单形式化一下，如果Ai = Aj，Ai原来在位置前，排序后Ai还是要在Aj位置前；没错，其实就是有两个排序关键字的时候，稳定排序可以让第一个关键字排序的结果服务于第二个关键字排序中数值相等的那些数。

不稳定的排序算法：排序之后在序列中的相对位置发生变化；

讲一下快速排序的思想

**美团二面（现场）**

讲一下数据的 acid

什么是一致性

什么是隔离性

Mysql 的隔离级别

每个隔离级别是如何解决

Mysql 要加上 nextkey 锁，语句该怎么写

如果在transaction1(Tr1)进行一个  
 select \* from table1 where id < 4 lock in share mode.  
 这里会在table1上加一个next\_key lock(间隙锁)，基本原理是什么呢？大致是这样的，内存中有一个lock hash。是一个key（类似于tableid+pageid+offset）到value（所加的锁）--- 这就是行锁的原理。所以 id<4的话，会给0 1 2 4(假设当前数据库没有3)加上行锁，这样就保证了不会出现插入id=3.5这种事情的发生。

Java 的内存模型，垃圾回收

线程池的参数

每个参数解释一遍

然后面试官设置了每个参数，给了是个线程，让描述出完整的线程池执行的流程

Nio 和 IO 有什么区别

Nio 和 aio 的区别

Spring 的 aop 怎么实现

Spring 的 aop 有哪些实现方式

动态代理的实现方式和区别

Linux 了解么

怎么查看系统负载

Cpu load 的参数如果为 4，描述一下现在系统处于什么情况

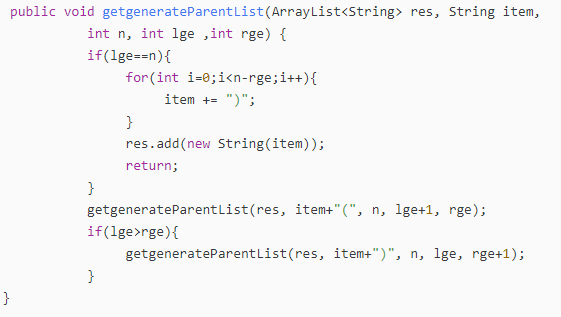
Linux，查找磁盘上最大的文件的命令

Linux，如何查看系统日志文件

手撕算法：leeetcode 原题 22，Generate Parentheses，给定 n 对括号，请写一个函数以将其生成新的括号组合，并返回所有组合结果。

样例：给定 n = 3, 可生成的组合如下:"((()))", "(()())", "(())()", "()(())", "()()()"

解题思路：采用递归树的思想，当左括号数大于右括号数时可以加左或者右括号，否则只能加左括号，当左括号数达到n时，剩下全部）。



**美团三面（现场）**

数据库的索引原理

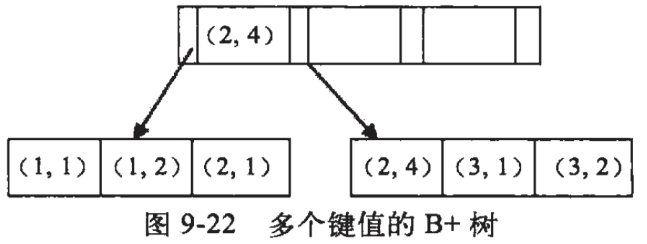
非聚簇索引和聚簇索引

索引的使用注意事项

联合索引

从底层解释最左匹配原则

最左匹配原则都是针对联合索引来说的，我们都知道索引的底层是一颗B+树，那么联合索引当然还是一颗B+树，只不过联合索引的健值数量不是一个，而是多个。构建一颗B+树只能根据一个值来构建，因此数据库依据联合索引最左的字段来构建B+树。  
例子：假如创建一个（a,b)的联合索引，那么它的索引树是这样的：



可以看到a的值是有顺序的，1，1，2，2，3，3，而b的值是没有顺序的1，2，1，4，1，2。所以b = 2这种查询条件没有办法利用索引，因为联合索引首先是按a排序的，b是无序的。

同时我们还可以发现在a值相等的情况下，b值又是按顺序排列的，但是这种顺序是相对的。所以最左匹配原则遇上范围查询就会停止，剩下的字段都无法使用索引。例如a = 1 and b = 2 a,b字段都可以使用索引，因为在a值确定的情况下b是相对有序的，而a>1and b=2，a字段可以匹配上索引，但b值不可以，因为a的值是一个范围，在这个范围中b是无序的。

Mysql 对联合索引有优化么？会自动调整顺序么？哪个版本开始优化？

会bca会调整成abc

Redis 的应用

Redis 的持久化的方式和原理

**百度一面（现场）**

. Java 中的多态

继承、方法重载和重写

. 为什么要同时重写 hashcode 和 equals

. Hashmap 的原理

. Hashmap 如何变线程安全，每种方式的优缺点

. 垃圾回收机制

. Jvm 的参数你知道的说一下

. 设计模式了解的说一下啊

. 手撕一个单例模式

算法题目

. 手撕算法：反转单链表

. 手撕算法：实现类似微博子结构的数据结构，输入一系列父子关系，输

出一个类似微博评论的父子结构图

. 手写 java 多线程

. 手写 java 的 soeket 编程，服务端和客户端

. 手撕算法： 爬楼梯，写出状态转移方程

. 智力题：时针分针什么时候重合

22

**百度二面（现场）**

. 服务器如何负载均衡，有哪些算法，哪个比较好，一致性哈希原理，怎么避免 DDOS 攻击请求打到少数机器。

. TCP 连接中的三次握手和四次挥手，四次挥手的最后一个 ack 的作用是

什么，为什么要 time wait，为什么是 2msl。

1、为了保证客户端发送的最后一个ACK报文段能够到达服务器。因为这个ACK有可能丢失，从而导致处在LAST-ACK状态的服务器收不到对FIN-ACK的确认报文。服务器会超时重传这个FIN-ACK，接着客户端再重传一次确认，重新启动时间等待计时器。最后客户端和服务器都能正常的关闭。假设客户端不等待2MSL，而是在发送完ACK之后直接释放关闭，一但这个ACK丢失的话，服务器就无法正常的进入关闭连接状态。

2、他还可以防止已失效的报文段。客户端在发送最后一个ACK之后，再经过经过2MSL，就可以使本链接持续时间内所产生的所有报文段都从网络中消失。从保证在关闭连接后不会有还在网络中滞留的报文段去骚扰服务器。

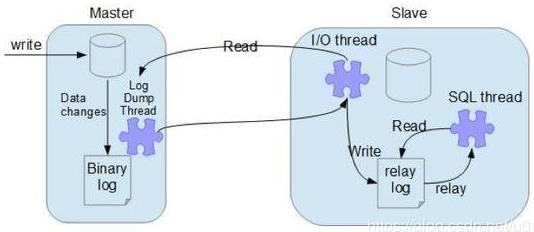
注意：在服务器发送了FIN-ACK之后，会立即启动超时重传计时器。客户端在发送最后一个ACK之后会立即启动时间等待计时器。

. 数据库的备份和恢复怎么实现的，主从复制怎么做的，什么时候会出现数据不一致，如何解决。

(1)首先，mysql主库在事务提交时会把数据库变更作为事件Events记录在二进制文件binlog中；mysql主库上的sys\_binlog控制binlog日志刷新到磁盘。

(2)主库推送二进制文件binlog中的事件到从库的中继日志relay log,之后从库根据中继日志重做数据库变更操作。通过逻辑复制，以此来达到数据一致。

Mysql通过3个线程来完成主从库之间的数据复制：其中BinLog Dump线程跑在主库上，I/O线程和SQl线程跑在从库上。当从库启动复制（start slave）时，首先创建I/O线程连接主库，主库随后创建Binlog Dump线程读取数据库事件并发给I/O线程，I/O线程获取到数据库事件更新到从库的中继日志Realy log中去，之后从库上的SQl线程读取中继日志relay log 中更新的数据库事件并应用。



binlog

主库中可以设置binlog，binlog是主库中保存更新事件日志的二进制文件。

主节点 binary log dump 线程

如上图所示：当从节点连接主节点时，主节点会创建一个log dump 线程，用于发送bin-log的内容。在读取bin-log中的操作时，此线程会对主节点上的bin-log加锁，当读取完成，甚至在发动给从节点之前，锁会被释放。

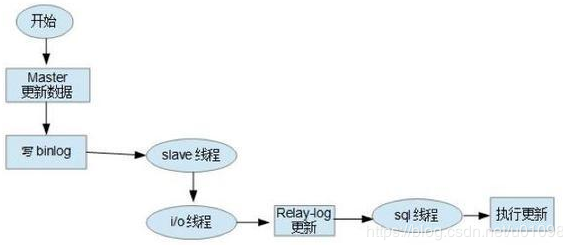
从节点I/O线程

当从节点上执行start slave命令之后，从节点会创建一个I/O线程用来连接主节点，请求主库中更新的bin-log。I/O线程接收到主节点binlog dump 进程发来的更新之后，保存在本地relay-log中。

从节点SQL线程

SQL线程负责读取relay log中的内容，解析成具体的操作并执行，最终保证主从数据的一致性。

对于每一个主从连接，都需要三个进程来完成。当主节点有多个从节点时，主节点会为每一个当前连接的从节点建一个binary log dump 进程，而每个从节点都有自己的I/O进程，SQL进程。从节点用两个线程将从主库拉取更新和执行分成独立的任务，这样在执行同步数据任务的时候，不会降低读操作的性能。比如，如果从节点没有运行，此时I/O进程可以很快从主节点获取更新，尽管SQL进程还没有执行。如果在SQL进程执行之前从节点服务停止，至少I/O进程已经从主节点拉取到了最新的变更并且保存在本地relay日志中，当服务再次起来之后，就可以完成数据的同步。

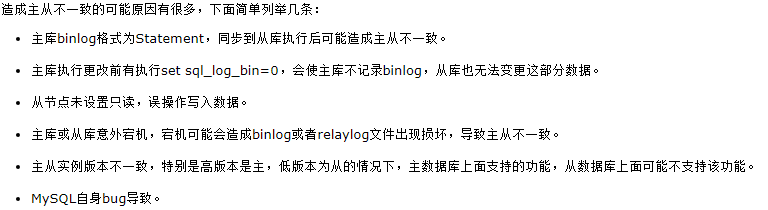


1.节点上的I/O 进程连接主节点，并请求从指定日志文件的指定位置（或者从最开始的日志）之后的日志内容；

2.主节点接收到来自从节点的I/O请求后，通过负责复制的I/O进程根据请求信息读取指定日志指定位置之后的日志信息，返回给从节点。返回信息中除了日志所包含的信息之外，还包括本次返回的信息的bin-log file 的以及bin-log position；

3.从节点的I/O进程接收到内容后，将接收到的日志内容更新到本机的relay log中，并将读取到的binary log文件名和位置保存到master-info 文件中，以便在下一次读取的时候能够清楚的告诉Master“我需要从某个bin-log 的哪个位置开始往后的日志内容，请发给我”；

4.Slave 的 SQL线程检测到relay-log 中新增加了内容后，会将relay-log的内容解析成在祝节点上实际执行过的操作，并在本数据库中执行。



第一种情况：比如说执行脚本时，为了更快的执行完，在脚本里增加了set sql\_log\_bin=0。那么这个脚本的所有数据变更将无法应用到从库，这个时候主从数据就不一致了，解决的方法是先停掉主从复制，然后手动在从库执行下这个脚本，最后开启主从复制即可。

第二种情况：可能你的从库并未设置只读，同事因不太清楚架构，误操作导致在从库做了数据写入，这种情况应该及时反馈并解决。解决方法：如果这些语句确实需要执行，则可以在主库先执行set sql\_log\_bin=0，然后再执行语句；如果不需要执行这些语句，则需要在从库上回滚掉先前的误操作。

不过有时候情况并不是那么简单，可能遇到比较多的情况是：主从两个实例已经运行很久了，某日进行一致性检验发现主从不一致了，很难找到具体发生不一致的原因及时间。那么这个时候应该怎么办呢，有人说，从库重做一遍，虽然这也是一种解决方法，但是这个方案恢复时间比较慢，而且有时候从库也是承担一部分的查询操作的，不能贸然重建。下面重点讲下这种情况下的修复方法。

手动重建不一致的表。

. Linux 查看 cpu 占用率高的进程

. 手撕算法：给定一个数字三角形，找到从顶部到底部的最小路径和。每

一步可以移动到下面一行的相邻数字上。

. 然后继续在这个问题上扩展

. 求出最短那条的路径

. 递归求出所有的路径

. 设计模式讲一下熟悉的

. 会不会滥用设计模式

. 多线程条件变量为什么要在 while 体里

. 你遇到什么挫折，怎么应对和处理

**百度三面（现场）**

. Redis 的特点

. Redis 的持久化怎么做，aof 和 rdb，有什么区别，有什么优缺点。

. Redis 使用哨兵部署会有什么问题，我说需要扩容的话还是得集群部署。

运维复杂，哨兵选举期间，不能对外提供服务

. 说一下 JVM 内存模型把，有哪些区，分别干什么的

. 说一下 gc 算法，分代回收说下

. MySQL 的引擎讲一下，有什么区别，使用场景呢

. 分布式事务了解么

. 反爬虫的机制，有哪些方式

**蚂蚁金服一面：分布式架构 50 分钟**

2、微服务架构是什么，它的优缺点？

3、ACID CAP BASE 理论

4、分布式一致性协议，二段、三段、TCC，优缺点

5、RPC 过程

6、服务注册中心宕机了怎么办？

7、微服务还有其他什么组件

8、分布式架构与微服务的关系

9、你有什么问题要问我的。

**蚂蚁金服二面：1 个小时**

1、各种排序算法、未排序常规数据查找第 K 大的数，时间复杂度。

2、二叉树的深度

3、虚拟内存分页了解不？(纯皮毛)

应用程序来说对物理内存地址一无所知。它只可能通过虚拟内存地址来进行数据读写。程序中表达的内存地址，也都是虚拟内存地址。进程对虚拟内存地址的操作，会被操作系统翻译成对某个物理内存地址的操作。由于翻译的过程由操作系统全权负责，所以应用程序可以在全过程中对物理内存地址一无所知。因此，C程序中表达的内存地址，都是虚拟内存地址。

记录对应关系最简单的办法，就是把对应关系记录在一张表中。为了让翻译速度足够地快，这个表必须加载在内存中。不过，这种记录方式惊人地浪费。如果树莓派1GB物理内存的每个字节都有一个对应记录的话，那么光是对应关系就要远远超过内存的空间。由于对应关系的条目众多，搜索到一个对应关系所需的时间也很长。这样的话，会让树莓派陷入瘫痪。

因此，Linux采用了分页（paging）的方式来记录对应关系。所谓的分页，就是以更大尺寸的单位页（page）来管理内存。在Linux中，通常每页大小为4KB。

4、进程和线程区别？

5、第一二三范式是什么？

6、一个表一千个列值为 true 和 false，写 sql 查询 有 300 个列值为 true 的行。

7、脏读和幻读是什么？

JVM

8、什么对象会从新生代晋升到老年代

(1)、Eden区满时，进行Minor GC，当Eden和一个Survivor区中依然存活的对象无法放入到Survivor中，则通过分配担保机制提前转移到老年代中。

(2)、若对象体积太大, 新生代无法容纳这个对象，-XX:PretenureSizeThreshold即对象的大小大于此值, 就会绕过新生代, 直接在老年代分配, 此参数只对Serial及ParNew两款收集器有效。

(3)、长期存活的对象将进入老年代。

        虚拟机对每个对象定义了一个对象年龄（Age）计数器。当年龄增加到一定的临界值时，就会晋升到老年代中，该临界值由参数：-XX:MaxTenuringThreshold来设置。

        如果对象在Eden出生并在第一次发生MinorGC时仍然存活，并且能够被Survivor中所容纳的话，则该对象会被移动到Survivor中，并且设Age=1；以后每经历一次Minor GC，该对象还存活的话Age=Age+1。

(4)、动态对象年龄判定。

        虚拟机并不总是要求对象的年龄必须达到MaxTenuringThreshold才能晋升到老年代，如果在Survivor区中相同年龄（设年龄为age）的对象的所有大小之和超过Survivor空间的一半，年龄大于或等于该年龄（age）的对象就可以直接进入老年代，无需等到MaxTenuringThreshold中要求的年龄。

9、一个任务分成十个任务，最后汇总计算，不能用 fork/join（countdownloch）

13、 安全协议有哪些 、https 是啥？

**三面**

1、从 ConcurrentHashMap 一路问到锁&锁优化->LongAdder->伪共享->缓存行填充->cas 等诸多技术细节；

2、从 hystrix 一路问到原理->自己如何实现->如何优化->响应流编程(reactive streams);

3、从简单的生产者消费者模式设计到如何高效健壮实现等等。

**四面**

1、如何倒序输出单向链表？

2、个人直接想法是用栈先进后出的特点，把链表数据读到栈里然后输出。

3、有更好的实现方式吗？

4、主要问项目情况，然后根据一个项目，问如果量级扩大 1000 倍，你会怎么做？有哪些优化措施？高性能&高可用措施？

**五面：HR 面**

给大家抛出几大深坑问题：

1、个人的职业规划是什么

2、你遇到的最大问题或者是困难是什么

3、你如何看待阿里

4、你能为阿里带来什么

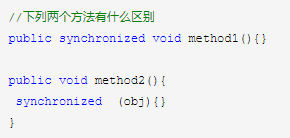
5、你的优缺点是什么

**头条**

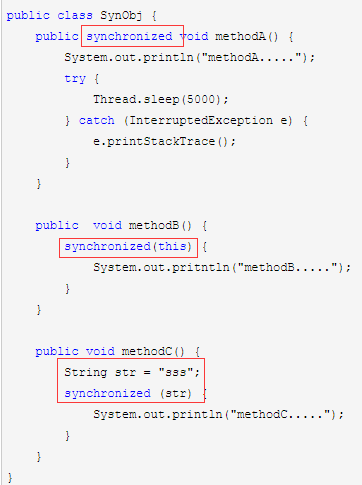
1. 聊项目，画项目架构图，画一个用户从发起请求 到接收到响应 中间经过哪些服务 每个服务做什么事情的流程图
2. 讲项目中的难点、挑战，你是如何解决的
3. redis 中有几种类型 & 各自底层怎么实现的 & 项目中哪个地方用了什么类型，怎么使用的
4. redis如何实现分布式锁，zk如何实现分布式锁，两者的区别。如果service还没执行完，分布式锁在redis中已经过期了，怎么解决这种问题

答：默认情况下,加锁的时间是30秒.如果加锁的业务没有执行完,那么到 30-10 = 20秒的时候,就会进行一次续期,把锁重置成30秒. 客户端1加锁的锁key默认生存时间才30秒，如果超过了30秒，客户端1还想一直持有这把锁，怎么办呢？简单！只要客户端1一旦加锁成功，就会启动一个watch dog看门狗，他是一个后台线程，会每隔10秒检查一下，如果客户端1还持有锁key，那么就会不断的延长锁key的生存时间。

1. synchronized底层实现，加在方法上和加在同步代码块中编译后的区别、类锁、对象锁



一种是上面的同步方法，即用synchronized来修饰方法，另一种是提供的同步代码块。



这段代码的打印结果是，methodA…..methodC…..会很快打印出来，methodB…..会隔一段时间才打印出来，那么methodB为什么不能像methodC那样很快被调用呢？  
  
在启动线程1调用方法A后，接着会让线程1休眠5秒钟，这时会调用方法C，注意到方法C这里用synchronized进行加锁，这里锁的对象是str这个字符串对象。但是方法B则不同，是用当前对象this进行加锁，注意到方法A直接在方法上加synchronized，这个加锁的对象是什么呢？显然，这两个方法用的是一把锁。  
  
\*由这样的结果，我们就知道这样同步方法是用什么加锁的了，由于线程1在休眠，这时锁还没释放，导致线程2只有在5秒之后才能调用方法B，由此，可知两种加锁机制用的是同一个锁对象，即当前对象。   
另外，同步方法直接在方法上加synchronized实现加锁，同步代码块则在方法内部加锁，很明显，同步方法锁的范围比较大，而同步代码块范围要小点，一般同步的范围越大，性能就越差，一般需要加锁进行同步的时候，肯定是范围越小越好，这样性能更好\*。

一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。

三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

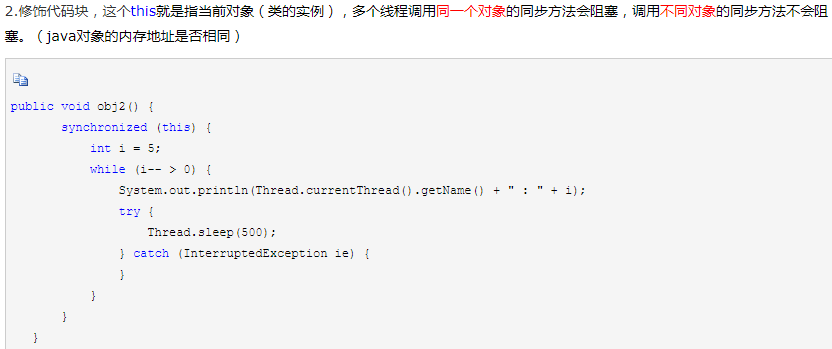
四、当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

第二篇

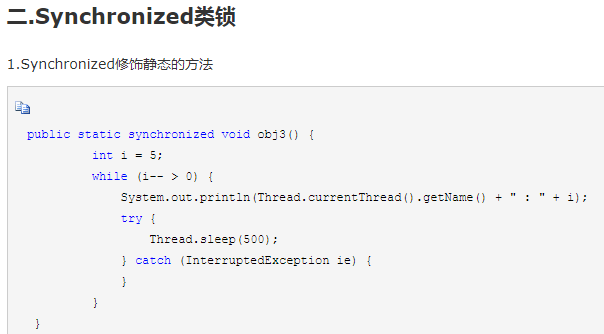
synchronized 关键字，它包括两种用法：synchronized 方法和 synchronized 块。    
1. synchronized 方法：通过在方法声明中加入 synchronized关键字来声明 synchronized 方法。如：    
public synchronized void accessVal(int newVal);    
synchronized 方法控制对类成员变量的访问：每个类实例对应一把锁，每个 synchronized 方法都必须获得调用该方法的类实例的锁方能执行，否则所属线程阻塞，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放，此后被阻塞的线程方能获得该锁，重新进入可执行状态。这种机制确保了同一时刻对于每一个类实例，其所有声明为 synchronized 的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该类实例对应的锁），从而有效避免了类成员变量的访问冲突（只要所有可能访问类成员变量的方法均被声明为 synchronized）。    
在 Java 中，不光是类实例，每一个类也对应一把锁，这样我们也可将类的静态成员函数声明为 synchronized ，以控制其对类的静态成员变量的访问。    
synchronized 方法的缺陷：若将一个大的方法声明为synchronized 将会大大影响效率，典型地，若将线程类的方法 run() 声明为synchronized ，由于在线程的整个生命期内它一直在运行，因此将导致它对本类任何 synchronized 方法的调用都永远不会成功。当然我们可以通过将访问类成员变量的代码放到专门的方法中，将其声明为 synchronized ，并在主方法中调用来解决这一问题，但是 Java 为我们提供了更好的解决办法，那就是 synchronized 块。

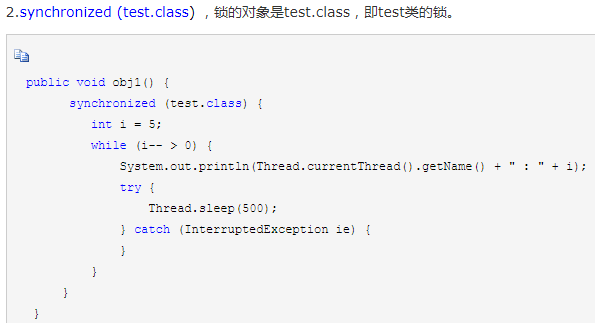
2. synchronized 块：通过 synchronized关键字来声明synchronized 块。语法如下：    
synchronized(syncObject) {    
//允许访问控制的代码    
}    
synchronized 块是这样一个代码块，其中的代码必须获得对象 syncObject （如前所述，可以是类实例或类）的锁方能执行，具体机制同前所述。由于可以针对任意代码块，且可任意指定上锁的对象，故灵活性较高。











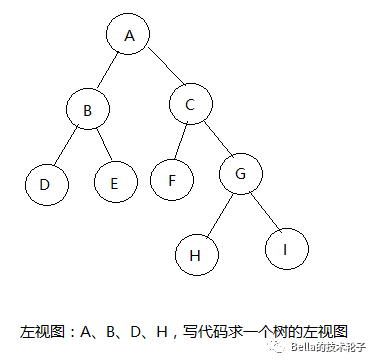
那么问题来了：在一个类中有两方法，分别用synchronized 修饰的静态方法（类锁）和非静态方法（对象锁）。多线程访问两个方法的时候，线程会不会阻塞？**不会阻塞。**

到这么我们应该知道了：

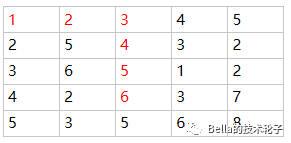
1，要满足方法同步（或者代码块同步）就必须保证多线程访问的是同一个对象（在java内存中的地址是否相同）。

2，类锁和对象锁同时存在时，多线程访问时不会阻塞，因为他们不是一个锁。

1. 锁升级的过程
2. java运行时区域及各个区域的作用、对GC的了解、java内存模型及为什么要这么设计
3. 对索引的理解，组合索引，索引的最佳实践
4. countDownLatch用过没有，在项目中如何使用的，对aqs 的了解
5. 写生产者消费者问题，考虑高并发的情况，可以使用Java 类库，白纸写代码
6. 如下图所示

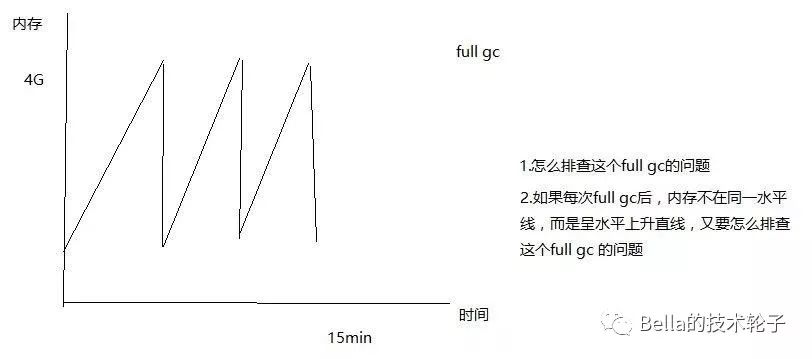


1. 设计一个发号器，考虑集群和高并发的情况，要求发号器生成的id是递增趋势，通过id可以区分出来是今天生成的id还是昨天生成的id，但是生成的id中不能直接带有日期，要具有一定的混淆功能，白纸写代码
2. 一个二位数组，每个元素都可以往上下左右四个方向走，寻找最长递增路径。如下图所示，最长递增路径即红色字体路径。白纸写代码。



**美团**

1. 数据库和缓存的一致性问题。先更新数据库，再更新缓存，若更新完数据库了，还没有更新缓存，此时有请求过来了，访问到了缓存中的数据，怎么办？
2. 聚簇索引/非聚簇索引，mysql索引底层实现，为什么不用B-tree，为什么不用hash，叶子结点存放的是数据还是指向数据的内存地址，使用索引需要注意的几个地方
3. mysql默认的事务隔离级别，mvcc，rr怎么实现的，rc如何实现的
4. mysql间隙锁有没有了解，死锁有没有了解，写一段会造成死锁的sql语句，死锁发生了如何解决，mysql有没有提供什么机制去解决死锁
5. 谈下对GC的了解，何为垃圾，有哪些GC算法，有哪些垃圾回收器，cms和g1的区别，emm，还有一个直击灵魂的问题，看过cms的源码吗，笑cry
6. 有没有排查过线上oom的问题，如何排查的
7. 有没有使用过jvm自带的工具，如何使用的
8. 假设有下图所示的一个full gc 的图，纵向是内存使用情况，横向是时间，你如何排查这个full gc的问题，怎么去解决你说出来的这些问题



1. 说说对java中集合类的理解，项目中用过哪些，哪个地方用的，如何使用的
2. 对CAS的理解，CAS带来的问题，如何解决这些问题
3. volatile底层、synchronized底层、锁升级的过程、MESI
4. 如何保证RocketMQ 消息的顺序性，如何解决重复消费问题
5. 项目中如何保证接口的幂等操作
6. 讲一讲对redis 的了解，项目中如何使用的，哪个地方使用的，为什么要使用
7. 哨兵机制、redis两种备份方式的区别，项目中用的哪种，为什么
8. 讲一讲对分布式锁的了解
9. 项目中系统监控怎么做的
10. 如何理解Spring中的AOP 和 IOC，以及DI，读过Spring源码没有
11. 读过MyBatis源码没有
12. 说一个你了解最多的框架，说出你的理解
13. 如何理解分布式事务，为什么会出现这个问题，如何去解决，了解哪些分布式事务中间件
14. 聊一聊对分库分表的理解
15. hystrix功能 & 在项目中怎么使用的 & hystrix 怎么检测断路器是否要开启/关闭 & hystrix 实现原理，除hystrix之外的其他熔断限流中间件有了解没有，了解多少说多少
16. dubbo有了解没有
17. 怎么理解java 中和 mysql 中的乐观锁、悲观锁
18. 一致性hash

**滴滴**

1. 处理过线上oom问题没有，如何处理的
2. 遇到过线上服务器cpu飙高的情况没有，如何处理的
3. 线上有没有遇到其他问题，如何处理的
4. 对线程池的理解，项目中哪个地方使用了，如何使用的，用的Excutor框架中的哪个实现类，为什么用这个
5. 对CAS的理解，CAS带来的问题，如何解决这些问题
6. volatile底层、synchronized底层、锁升级的过程、MESI
7. 对mysql索引的理解、对组合索引的理解、索引的最佳实践
8. 分布式锁的实现、对比redis分布式锁 & zk分布式锁
9. 唯一id如何实现的，snowflake实现原理，snowflake有哪些问题，如何避免根据订单号可以推算出今天的订单量
10. 如果线上一个功能是用栈结构实现的，使用过程中要注意哪些问题，为什么
11. 怎么理解线程安全
12. 怎么理解接口幂等，项目中如何保证的接口幂等
13. 怎么理解微服务，服务如何划分，可以从哪几个方面去划分，为什么这样划分，微服务带来了哪些好处，哪些坏处，如何看待这个问题
14. 如何理解网关，网关带来的好处和坏处，如何解决
15. hystrix功能 & 在项目中怎么使用的 & hystrix 怎么检测断路器是否要开启/关闭 & hystrix 实现原理
16. 怎么理解命令模式和观察者模式，手写一个观察者模式或者命令模式的代码，策略模式也行
17. 掌握哪些设计模式，常用哪些，项目中如何使用的，为什么用这个，不用那个，手写一个线程安全的单例模式
18. 如何设计一个秒杀系统
19. 如果我现在就是要实现每秒10w请求，不能熔断限流，如何去设计
20. 假设现在双十一零点，大量下单请求，如何对这些订单进行分库分表，为什么
21. 服务A调用服务B中一个接口，服务B调用服务C中一个接口，如何实现若服务B响应服务A成功，则服务C一定响应服务B成功，需要考虑系统性能问题
22. 递归使用中有什么需要注意的地方，递归写法一般可以用什么去替换
23. 有两个表，table a，table b，写sql查询出仅在table a中的数据、仅在table b中的数据、既在table a 又在table b 中的数据
24. spring 源码有了解没有
25. myBatis源码有了解没有
26. mysql事务隔离级别、mvcc

我：既然现在很多业务线都是Go了，有没有考虑把剩余的业务线也转成Go呀？

面试官：我认为，语言只是工具，语言不应该是影响开发的一个因素吧。

面试官说的很有道理。

**京东**

1. 一个final修饰的属性，定义的时候没有初始化，在无参构造函数中初始化，可以吗，为什么
2. 说说对java中集合类的理解，项目中用过哪些，哪个地方用的，如何使用的，为什么不用其他的集合类
3. hashMap，concurrentHashMap底层实现，
4. list删除是怎么实现的，遍历的时候可以删除吗，为什么

答：普通for循环可以；迭代器删除报错；使用listIterator提供的remove()可以；增强for报错（for a : list）

1. redis中有哪些数据结构，了解过其底层怎么实现的吗，和java中相似的数据结构的对比
2. redis是单线程的还是多线程的，为什么这么快
3. redis hash中某个key过大，变为String类型的大key，怎么处理，使用中如何避免出现这种问题
4. 设计模式在项目中哪个地方用到了，怎么使用的，能不能画一个你熟悉的设计模式的UML图，手写单例模式，手写静态内部类实现的单例模式
5. 讲一讲mysql索引，实际工作中，哪些场景用了b+tree索引，哪些场景用了hash索引
6. explain 可以看到哪些信息，什么信息说明什么，explain的结果列讲一下
7. Spring源码看过没有，会多少讲多少
8. MyBatis源码看过没有，会多少讲多少
9. cas，cas的缺点，如何解决
10. aqs，countDownLatch如何实现
11. 线程池如何实现，核心线程数和最大线程数设置成多少，为什么这么设置，项目中哪个地方使用了线程池，使用时需要注意什么
12. mysql事务隔离级别，幻读，脏读，项目中用什么事务隔离级别，为什么
13. volatile底层原理、synchronized实现机制，
14. 对XA、TCC的理解，了解哪些分布式事务框架，有什么缺点
15. feign 和 dubbo，了解多少说多少
16. eureka 和 zookeeper，了解多少说多少
17. hystrix 和 sentinel，了解多少说多少
18. Spring cloud alibaba，了解多少说多少
19. 对分库分表、读写分离的了解，了解多少说多少
20. 画一下java 线程几个状态 及 状态之间互相转换的图

**others**

1. kafka 如何保证消息顺序消费、在consumer group 中新增一个consumer 会提高消费消息的速度吗、那如果我想提高消息消费的速度，我要怎么办
2. redis几种数据结构 及 底层，项目中如何使用的redis
3. 哨兵机制、选举算法



选举完成后，从节点正式提升为主节点，并需要更新其在集群中负责的配置

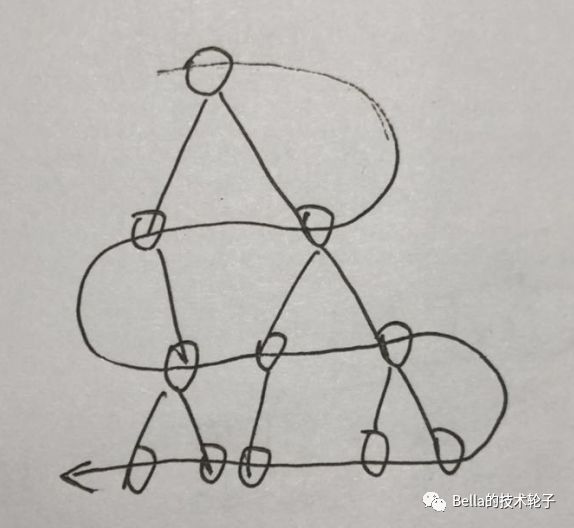
1. 多路复用的几种方式以及区别
2. 对线程池的理解，在项目中如何使用的，多个线程之间如何共享数据，多个进程之间如何共享数据
3. hashMap、concurrentHashMap的区别及底层实现、hashMap和hashTable的区别
4. 什么是红黑树，什么是b-tree，为什么hashMap中用红黑树不用其他树
5. 对mysql 索引的理解，为什么mysql索引中用b+tree，不用b-tree 或者其他树，为什么不用hash 索引
6. 数据库和缓存的双写一致性问题

**每日一淘**：

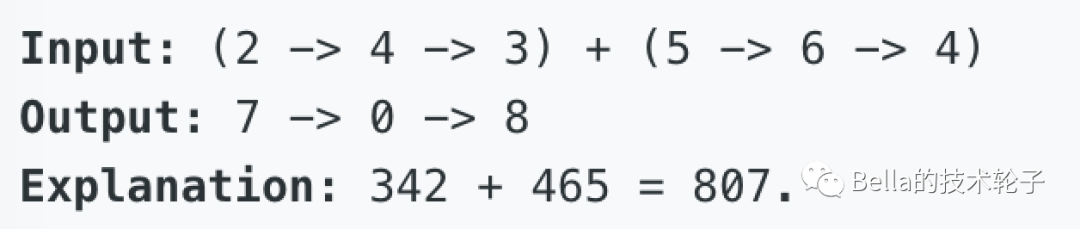
1. 用过哪些Object类的方法，如何使用的
2. java如何实现序列化的，Serialization底层如何实现的
3. countDownLatch如何实现的
4. 服务A调用服务B，用户请求服务A，发现返回较慢，如何定位这个问题
5. TIME\_WAIT是什么状态还记得吗，什么情况下网络会出现这个状态

**算法题**

1. [1,1,2,2,3,4,4,5,5,5] 找出不重复的元素（黄包车）hashset
2. 反转链表，要求时间复杂度O(N)，空间复杂度O(1) （火币）栈
3. 非递归实现斐波那契数列 （爱奇艺）
4. 这一周股市价格为[2,6,1,4,8]，求哪一天买入哪一天卖出，可获得最大收益，最大收益为多少 （爱奇艺）
5. 按照箭头方向查找二叉树 （金山云）



1. 一个整形无序数组，里面三个数只和等于一个目标值，求这三个数 （小米）
2. 链表问题 （小米）



1. 扑克牌问题 （小米）

有十张扑克牌，从上面开始抽，抽出一张放桌子上，然后再抽出一张放扑克牌的最下面，这样循环往复的操作，直到手里的牌都没有了。这时，桌子上牌的顺序正好是1 2 3 4 5 6 7 8 9 10。要求写代码求出原顺序

1. 手写大顶堆 （linkedMe）
2. 手写LRU 算法 （火币）
3. 字符串相加 （滴滴）

两个数字类型的字符串，直接转int或者double肯定都放不下，然后求这两个数的和，返回值还是字符串，15分钟时间，要求无bug（从低位逐步加，高位用0补位）

1. 寻找目标值位置 （滴滴）

有一个二维数组，数组横向有序，纵向有序，求目标值的位置，10分钟时间

1. 求字符串“efabcbaefehiabcba”中最长的回文数，不去重（美团）再弄一个反向的字符串；以某个考察左右是否相同
2. 反转int类型的值x，不要借用String，只用int 即可。&& 针对该程序，写出其应有的测试用例 （美团）
3. top K 问题（每日一淘）