**关于自己**

1. 自我介绍（一般1分钟左右，着重突出你的 1.积极上进 2.做过的有关案例 3.对这家公司的兴趣和看法）

**学习上：**

是这样的，研究生的很多时间除了上课，大部分都花在做我们老师的项目上面了，我们导师项目比较偏，算交叉学科，主要通过改进智能算法、寻找某个特定气候现象的适应度函数（物理能量公式），结合气象领域中的海气耦合模式来解最优值的问题；说实话，我们导师项目涉及到的技术和当下互联网公司所需要的技术不是很对接，所以自己在剩余时间也自学了一些专业课程，比如Java web的SSH框架、前端框架Bootstrap等等。其实自己对于互联网行业挺热爱的，自己选择后端，也是希望在以后的工作中，朝着全栈工程师的方向发展。总的来说，自己是愿意学习、肯学习的，因为农村人嘛，也愿意努力、吃苦。

**项目上：**

**自己课外做过的项目：**项目不大，都是一些类似课程作业的项目

（1）研一《数据库设计》大作业，开发了一款“研究生管理系统”，两个人合作的，比较简单，分为研究生、导师、管理员三大模块，自己主要负责三大模块的后台编码工作，包括简单的增删查过、文件上传、下载等功能实现；

（2）参加了中兴公司的算法竞赛，是求解多个约束条件的最短路径问题，约束条件是必须经过两条边、不能经过给定的一条边、必须经过两个点，自己使用遗传算法二进制编码方式求解出最优解；

（3）参加了上海市“气象+大数据”应用创新大赛，构建了一个基于PSO算法优化的BP-RBF-SVM的组合预测模型，预测农产品期货价格走势；

（4）和班里的同学接了一个项目，为我们学校汽车学院的一个老师，设计开发了一款介绍他们实验室的小型web项目，基本上就是前端页面的CSS布局、javascript动画、基于bootstrap的响应式。

**对于这家公司的兴趣和看法：**

1. 我有哪些优秀特质（着重突出针对这个岗位，需要凸显的特质，有的放矢）

**能够吃苦、抗压性较强、能够适应高强度工作**

1. 我有什么需要提高、最希望提升的特质？ （比如：经验还不够丰富）

**没有啥实习经验，实战经验比较少**

4、说一说你做过的印象深刻的项目（着重说一个，你是怎么思考的，怎么解决了问题的，事后如何复盘的）

**关于这家公司**

5、用http://news.baidu.com搜索公司名称、创始人名字，认真读完前5页新闻

9、能说出3-5家，这家公司的竞争对手/业务比较接近的公司

**关于产品**

12、亲自下载、注册、试用要去面试的公司的产品，必须超过15分钟，最好覆盖各个功能模块

16、你还有什么想问的？（一般可以问一下对方公司：面试官对自己有什么期望？）

是这样的，我想问一下，公司对于这次招实习生的技术上面有没有具体硬性要求，可能我技术这块确实不是太好，但是，我后期是可以实习5-6个月的嘛，所以我会非常努力把技术学好的！

HashMap和Hashtable的区别

1. 两者最主要的区别在于Hashtable是线程安全，而HashMap则非线程安全  
   Hashtable的实现方法里面都添加了synchronized关键字来确保线程同步，因此相对而言HashMap性能会高一些
2. HashMap可以使用null作为key，而Hashtable则不允许null作为key  
   虽说HashMap支持null值作为key，不过建议还是尽量避免这样使用，因为一旦不小心使用了，若因此引发一些问题，排查起来很是费事
3. HashMap的初始容量为16，Hashtable初始容量为11
4. 扩容方式不同，**一个是2倍，一个是2倍+1**
5. 两者计算hash的方法不同  
   Hashtable计算hash是直接使用key的hashcode对table数组的长度直接进行取模

HashMap计算hash对key的hashcode进行了二次hash，以获得更好的散列值，然后对table数组长度取摸

**三次握手 四次挥手**

1. 序号：Seq序号、
2. 确认序号：Ack序号，Ack=Seq+1、
3. ACK：确认序号有效
4. SYN：发起一个新连接。
5. FIN：释放一个连接。

（1）第一次握手：

Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

（2）第二次握手：

Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

（3）第三次握手：

Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，**Client和Server进入ESTABLISHED状态，完成三次握手**，随后Client与Server之间可以开始传输数据了。

第一次挥手：

Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入**FIN\_WAIT\_1**状态。

第二次挥手：

Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入**CLOSE\_WAIT**状态。

第三次挥手：

Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入**LAST\_ACK**状态。

第四次挥手：

Client收到FIN后，Client进入**TIME\_WAIT**状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入**CLOSED状**态，完成四次挥手。

问题1： 为什么要四次挥手？

答：根本原因是，一方发送FIN只表示自己发完了所有要发的数据，但还允许对方继续把没发完的数据发过来

问题1： 为什么要三次握手？

答：三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收机能正常。

**HTTP：是互联网上应用最为广泛的一种网络协议，是一个客户端和服务器端请求和应答的标准（TCP），用于从WWW服务器传输超文本到本地浏览器的传输协议，它可以使浏览器更加高效，使网络传输减少。**

**HTTPS：是以安全为目标的HTTP通道**，简单讲是HTTP的安全版，即**HTTP下加入SSL层**，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。

**HTTPS协议的主要作用可以分为两种：一种是建立一个信息安全通道，来保证数据传输的安全；另一种就是确认网站的真实性。**

1）http是超文本传输协议，信息通过明文传输，https 则是具有安全性的ssl加密传输协议，是安全办的http，使用加密方式传输信息；

2）http和https使用的是完全不同的连接方式用的端口也不一样,前者是80,后者是443。

3）http的连接很简单,是无状态的，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，**要比http协议安全**

**this.show(O)、super.show(O)、this.show((super)O)、super.show((super)O)**

有没有既线程安全效率又高的Map？（ConcurrentHashMap）

ConcurrentHashMap的工作机制，通过把整个Map分为N个Segment（类似HashTable），可以提供相同的线程安全，但是效率提升N倍，默认提升16倍。

**[IOC](https://www.baidu.com/s?wd=IOC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm164mvf3nWPbrju-rHFb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nvPWTLnjcs" \t "https://www.cnblogs.com/ScarecrowAnBird/p/_blank):[控制反转](https://www.baidu.com/s?wd=%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm164mvf3nWPbrju-rHFb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nvPWTLnjcs" \t "https://www.cnblogs.com/ScarecrowAnBird/p/_blank)也叫[依赖注入](https://www.baidu.com/s?wd=%E4%BE%9D%E8%B5%96%E6%B3%A8%E5%85%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm164mvf3nWPbrju-rHFb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nvPWTLnjcs" \t "https://www.cnblogs.com/ScarecrowAnBird/p/_blank)。利用了[工厂模式](https://www.baidu.com/s?wd=%E5%B7%A5%E5%8E%82%E6%A8%A1%E5%BC%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm164mvf3nWPbrju-rHFb0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1nvPWTLnjcs" \t "https://www.cnblogs.com/ScarecrowAnBird/p/_blank)。IOC就是典型的工厂模式，通过sessionfactory去注入实例。**

1. 将对象交给容器管理，你只需要在spring配置文件总配置相应的bean，以及设置相关的属性，让spring容器来生成类的实例对象以及管理对象。2.在spring容器启动的时候，spring会把你在配置文件中配置的bean都初始化好，3.在你需要调用的时候，就把它已经初始化好的那些bean分配给你需要调用这些bean的类（假设这个类名是A），分配的方法就是调用A的setter方法来注入，而不需要你在A里面new这些bean了。

spring的IoC容器是spring的核心，spring AOP是spring框架的重要组成部分

AOP就是典型的代理模式的体现

1）总而言之， GC 主要进行回收的内存是 JVM 中的方法区和堆；

2）如何判断一个对象已经死去？ 是垃圾

3)关于方法区中需要回收的是一些废弃的常量和无用的类。  
1.废弃的常量的回收。这里看引用计数就可以了。没有对象引用该常量就可以放心的回收了。  
2.无用的类的回收。什么是无用的类呢？  
A.该类所有的实例都已经被回收。也就是 Java 堆中不存在该类的任何实例；  
对于堆中的对象，主要用可达性分析判断一个对象是否还存在引用，如果该对象没有任何引用就应该被回收

对于方法区中的常量和类，当一个常量没有任何对象引用它，它就可以被回收了。而对于类，如果可以判定它为无  
用类，就可以被回收了。

1. Java 中为什么会有 GC 机制呢？  
   • **安全性考虑；** -- for security.  
   • 减少内存泄露； -- erase memory leak in some degree.  
   • 减少程序员工作量。
2. 既然有 GC 机制，为什么还会有内存泄露的情况 （2017-11-16-wl）  
   理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一  
   个重要原因）。然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收，因此也会导致内存泄  
   露的发生
3. 解释内存中的栈 (stack) 、堆 (heap) 和方法区 (method area) 的用法  
   通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用 JVM 中的栈空间；而通过 new 关键字和构造器创建的对象则放在堆空间，堆是垃圾收集器管理的主要区域，由于现在的垃圾收集器  
   都采用分代收集算法，所以堆空间还可以细分为新生代和老生代；

方法区和堆都是各个线程共享的内存区域，用于存储已经被 JVM 加载  
的类信息、常量、静态变量、 JIT 编译器编译后的代码等数据；程序中的字面量（literal）如直接书写的 100、 "hello"  
和常量都是放在常量池中，常量池是方法区的一部分。

栈空间用光了会引发 StackOverflowError，而堆和常量池空间不足则会引发 OutOfMemoryError。

GC（Garbage Collector）在回收对象前首先必须发现那些无用的对象，如何去发现定位这些无用的对象？常用  
的搜索算法如下：  
1） 引用计数器算法（废弃）  
引用计数器算法是给每个对象设置一个计数器，当有地方引用这个对象的时候，计数器+1，当引用失效的时候，  
计数器-1，当计数器为 0 的时候， JVM 就认为对象不再被使用，是“垃圾”了。  
引用计数器实现简单，效率高；但是不能解决循环引用问问题（A 对象引用 B 对象， B 对象又引用 A 对象，但是  
A,B 对象已不被任何其他对象引用），同时每次计数器的增加和减少都带来了很多额外的开销，所以在 JDK1.1 之后，  
这个算法已经不再使用了。  
2） 根搜索算法（使用）  
根搜索算法是通过一些“GC Roots”对象作为起点，从这些节点开始往下搜索，搜索通过的路径成为引用链  
（Reference Chain），当一个对象没有被 GC Roots 的引用链连接的时候，说明这个对象是不可用的

通过上面的算法搜索到无用对象之后，就是回收过程， 回收算法如下：  
1） 标记—清除算法（Mark-Sweep）（DVM 使用的算法）  
标记—清除算法包括两个阶段：“标记”和“清除”。在标记阶段，确定所有要回收的对象，并做标记。清除阶段  
紧随标记阶段，将标记阶段确定不可用的对象清除。标记—清除算法是基础的收集算法，标记和清除阶段的效率不高，  
而且清除后回产生大量的不连续空间，这样当程序需要分配大内存对象时，可能无法找到足够的连续空间。

1. 复制算法（Copying）  
   复制算法是把内存分成大小相等的两块，每次使用其中一块，当垃圾回收的时候，把存活的对象复制到另一块上，  
   然后把这块内存整个清理掉。复制算法实现简单，运行效率高，但是由于每次只能使用其中的一半，造成内存的利用率  
   不高。现在的 JVM 用复制方法收集新生代，由于新生代中大部分对象（98%）都是朝生夕死的，所以两块内存的比例  
   不是 1:1(大概是 8:1)。
2. 标记—整理算法（Mark-Compact）  
   标记—整理算法和标记—清除算法一样，但是标记—整理算法不是把存活对象复制到另一块内存，而是把存活对  
   象往内存的一端移动，然后直接回收边界以外的内存。标记—整理算法提高了内存的利用率，并且它适合在收集对象  
   存活时间较长的老年代。
3. 分代收集（Generational Collection）  
   分代收集是根据对象的存活时间把内存分为新生代和老年代，根据各个代对象的存活特点，每个代采用不同的垃  
   圾回收算法。新生代采用复制算法，老年代采用标记—整理算法。垃圾算法的实现涉及大量的程序细节，而且不同的  
   虚拟机平台实现的方法也各不相同
4. 策略模式

那如果真人有溺水了怎么办？有人会打120，有人会做人工呼吸（必须得是个美女才行），有人会用拳头叩击心前区急救。他们的做法都是对的，只是不同情况，不同的人会有不同的策略！

**定义一组算法，将每个算法都封装起来，并且使它们之间可以互换**

由三个角色构成

****1、Rescue封装角色****

也叫上下文角色，起承上启下的封装作用，屏蔽高层模块对策略、算法的直接访问、封装可能存在的变化。

****2、Strategy抽象策略角色****

策略、算法家族的抽象，通过为接口，定义每个策略或算法必须具有的方法和属性。

****3、ConcreteStrategy具体策略角色****

实现抽象策略中的操作，该类含有具体的算法。

1. 什么是线程池，如何使用？  
   线程池就是事先将多个线程对象放到一个容器中，当使用的时候就不用 new 线程而是直接去池中拿线程即可，节  
   省了开辟子线程的时间，提高的代码执行效率。

减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务。

真正的线程池接口是ExecutorService。

//创建一个可重用固定线程数的线程池

ExecutorService pool = Executors. newSingleThreadExecutor();

//将线程放入池中进行执行

pool.execute(t1);

//关闭线程池

pool.shutdown();

volatile和synchronized的区别

volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取； synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。

volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的

volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞。

字节流和字符流

1. 字节流 FileInputStream FileOutputStream BufferedInputStream BuffferedOutputStream
2. 字符流 FileReader InputStreamReader BufferedReader FileWriter InputStreamWriter BufferedWriter
3. 字节流如何转为字符流  
   字节输入流转字符输入流通过 InputStreamReader 实现，该类的构造函数可以传入 InputStream 对象。  
   字节输出流转字符输出流通过 OutputStreamWriter 实现，该类的构造函数可以传入 OutputStream 对象。
4. 如何将一个 java 对象序列化到文件里  
   在 java 中能够被序列化的类必须先实现 Serializable 接口，该接口没有任何抽象方法只是起到一个标记作用

序列化：通过ObjectOutputStream 和 writeObject（）

反序列化：通过ObjectInputStram 和 readObject（）

1. 如何实现对象克隆？ (2017-11-12-wl)  
   有两种方式。  
   1). 实现 Cloneable 接口并重写 Object 类中的 clone()方法；  
   2). 实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆，
2. 请用两个队列模拟堆栈结构
3. Map 中的 key 和 value 可以为 null 么？
4. 传统使用类 Thread和接口 Runnable 实现 、使用 ExecutorService、 Callable、 Future 实现有返回结果的多线程
5. 间接相互制约。一个系统中的多个线程必然要共享某种系统资源，如共享 CPU，共享 I/O 设备，所谓间接相  
   互制约即源于这种资源共享，打印机就是最好的例子，线程 A 在使用打印机时，其它线程都要等待。  
   直接相互制约。这种制约主要是因为线程之间的合作，如有线程 A 将计算结果提供给线程 B 作进一步处理，  
   那么线程 B 在线程 A 将数据送达之前都将处于阻塞状态。  
   间接相互制约可以称为互斥，直接相互制约可以称为同步
6. 子线程运行执行 10 次后，主线程再运行 5 次。这样交替执行三遍
7. 表示查看所有进程里 CMD 是 java 的进程信息。  
   ps -aux | grep java  
   -aux 显示所有状态  
   kill 命令用于终止进程。  
   例如： kill -9 [PID]
8. 列出文件列表： ls 【参数 -a -l】  
   创建目录和移除目录： mkdir rmdir  
   用于显示文件后几行内容： tail  
   打包： tar -xvf  
   打包并压缩： tar -zcvf  
   查找字符串： grep  
   显示当前所在目录： pwd  
   创建空文件： touch  
   编辑器： vim vi
9. SQL 之连接查询（左连接和右连接的区别）（2017-11-15-lyq）  
   外连接：  
   左连接（左外连接）：以左表作为基准进行查询，左表数据会全部显示出来，右表如果和左表匹配的  
   数据则显示相应字段的数据，如果不匹配则显示为 null。  
   右连接（右外连接）：以右表作为基准进行查询，右表数据会全部显示出来，左表如果和右表匹配的  
   数据则显示相应字段的数据，如果不匹配则显示为 null。  
   全连接：**先以左表进行左外连接，再以右表进行右外连接**

1、READ-UNCOMMITTED隔离级别，当两个事务同时进行时，即使事务没有提交，所做的修改也会对事务内的查询做出影响，这种级别显然很不安全。但是在表对某行进行修改时，会对该行加上行共享锁

2、READ-COMMITTED事务隔离级别，只有在事务提交后，才会对另一个事务产生影响，并且在对表进行修改时，会对表数据行加上行共享锁

3、REPEATABLE-READ事务隔离级别，当两个事务同时进行时，其中一个事务修改数据对另一个事务不会造成影响，即使修改的事务已经提交也不会对另一个事务造成影响。在事务中对某条记录修改，会对记录加上行共享锁，直到事务结束才会释放。

4、SERIALIZABLE事务隔离级别最严厉，在进行查询时就会对表或行加上共享锁，其他事务对该表将只能进行读操作，而不能进行写操作。

1、海量日志数据，提取出某日访问百度次数最多的那个IP。

    既然是海量数据处理，那么可想而知，给我们的数据那就一定是海量的。针对这个数据的海量，我们如何着手呢?对的，无非就是分而治之/hash映射 + hash统计 + 堆/快速/归并排序，说白了，就是先映射，而后统计，最后排序：

1）分而治之**/hash映射：针对数据太大，内存受限，**只能是：把大文件化成(取模映射)小文件，即16字方针：大而化小，各个击破，缩小规模，逐个解决

2）hash\_map统计：当大文件转化了小文件，那么我们便可以采用常规的hash\_map(ip，value)来进行频率统计。

3）堆/快速排序：统计完了之后，便进行排序(可采取堆排序)，得到次数最多的IP。

   具体而论，则是： “首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有个2^32个IP。同样可以采用映射的方法，比如%1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP（可以采用hash\_map对那1000个文件中的所有IP进行频率统计，然后依次找出各个文件中频率最大的那个IP）及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。”--[十道海量数据处理面试题与十个方法大总结](http://blog.csdn.net/v_JULY_v/archive/2011/03/26/6279498.aspx" \t "https://blog.csdn.net/v_july_v/article/details/_blank)。

解答：由上面第1题，我们知道，数据大则划为小的，如如一亿个Ip求Top 10，可先%1000将ip分到1000个小文件中去，并保证一种ip只出现在一个文件中，再对每个小文件中的ip进行hashmap计数统计并按数量排序，最后归并或者最小堆依次处理每个小文件的top10以得到最后的结。

1. 寻找热门查询，300万个查询字符串中统计最热门的10个查询

hash\_map统计：先对这批海量数据预处理。具体方法是：维护一个Key为Query字串，Value为该Query出现次数的HashTable，即hash\_map(Query，Value)，每次读取一个Query，如果该字串不在Table中，那么加入该字串，并且将Value值设为1；如果该字串在Table中，那么将该字串的计数加一即可。最终我们在O(N)的时间复杂度内用Hash表完成了统计；

堆排序：第二步、借助堆这个数据结构，找出Top K，时间复杂度为N‘logK。即借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比。所以，我们最终的时间复杂度是：O（N） + N' \* O（logK），（N为1000万，N’为300万）。

3、有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M。返回频数最高的100个词。  
 由上面那两个例题，分而治之 + hash统计 + 堆/快速排序这个套路，我们已经开始有了屡试不爽的感觉。下面，再拿几道再多多验证下。请看此第3题：又是文件很大，又是内存受限，咋办?还能怎么办呢?无非还是：

分而治之/hash映射：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件（记为x0,x1,...x4999）中。这样每个文件大概是200k左右。如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M。

hash\_map统计：对每个小文件，采用trie树/hash\_map等统计每个文件中出现的词以及相应的频率。

堆/归并排序：取出出现频率最大的100个词（可以用含100个结点的最小堆）后，再把100个词及相应的频率存入文件，这样又得到了5000个文件。最后就是把这5000个文件进行归并（类似于归并排序）的过程了。

4、海量数据分布在100台电脑中，想个办法高效统计出这批数据的TOP10。

    如果每个数据元素只出现一次，而且只出现在某一台机器中，那么可以采取以下步骤统计出现次数TOP10的数据元素：

堆排序：在每台电脑上求出TOP10，可以采用包含10个元素的堆完成（TOP10小，用最大堆，TOP10大，用最小堆，比如求TOP10大，我们首先取前10个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是TOP10大）。

求出每台电脑上的TOP10后，然后把这100台电脑上的TOP10组合起来，共1000个数据，再利用上面类似的方法求出TOP10就可以了。

****7、怎么在海量数据中找出重复次数最多的一个？****

    方案：先做hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

****8、上千万或上亿数据（有重复），统计其中出现次数最多的前N个数据。****

    方案：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用hash\_map/搜索二叉树/红黑树等来进行统计次数。然后利用堆取出前N个出现次数最多的数据。

****9、一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前10个词，请给出思想，给出时间复杂度分析。****

方案1：如果文件比较大，无法一次性读入内存，可以采用hash取模的方法，将大文件分解为多个小文件，对于单个小文件利用hash\_map统计出每个小文件中10个最常出现的词，然后再进行归并处理，找出最终的10个最常出现的词。  
    方案2：通过hash取模将大文件分解为多个小文件后，除了可以用hash\_map统计出每个小文件中10个最常出现的词，也可以用trie树统计每个词出现的次数，时间复杂度是O(n\*le)（le表示单词的平准长度），最终同样找出出现最频繁的前10个词（可用堆来实现），时间复杂度是O(n\*lg10)。

****11. 一个文本文件，找出前10个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次读入内存，问最优解。****  
    方案1：首先根据用hash并求模，将文件分解为多个小文件，对于单个文件利用上题的方法求出每个文件件中10个最常出现的词。然后再进行归并处理，找出最终的10个最常出现的词。

12、如何从100万个数中找出其中最大的100个数？

2.先取出前100个数，维护一个100个数的最小堆，遍历一遍剩余的元素，在此过程中维护堆就可以了。具体步骤如下：

step1：取前m个元素（例如m=100），建立一个小顶堆。保持一个小顶堆得性质的步骤，运行时间为O（lgm);建立一个小顶堆运行时间为m\*O（lgm）=O(m lgm);

step2:顺序读取后续元素，直到结束。每次读取一个元素，如果该元素比堆顶元素小，直接丢弃 ，如果大于堆顶元素，则用该元素替换堆顶元素，然后保持最小堆性质。最坏情况是每次都需要替换掉堆顶的最小元素，因此需要维护堆的代价为(N-m)\*O(lgm); 最后这个堆中的元素就是前最大的10W个。时间复杂度为O(N lgm）。

捆绑法：当要求某几个元素必须相邻（挨着）时，先将这几个元素看做一个整体，（比如：原来3个元素，整体考虑之后看成1个元素）然后将这个整体和其它元素进行考虑。这时要注意：一般整体内部各元素如果在前后顺序上有区别的还需进行一定的顺序考虑。

插空法：当要求某几个元素必须不相邻（挨着）时，可先将其它元素排好，然后再将要求不相邻的元素根据题目要求插入到已排好的元素的空隙或两端位置。

插隔板法：指在解决若干相同元素分组，要求每组至少一个元素时，采用将比分组数目少1的隔板插入到元素中的一种解题策略。题目特点：“若干相同元素分组”、“ 每组至少一个元素”。

例1（08-57）一张节目表上原有3个节目，如果保持这3个节目的相对顺序不变，再添进去2个新节目，有多少种安排方法？

A.20      B.12       C.6        D.4

分两种情况考虑

1、  这两个新节目挨着，那么三个节目有4个空，又考虑到这两个节目的先后顺序共有2×C41=8种

2、  这两个节目不挨着，那么三个节目有4个空，这就相当于考虑两个数在4个位置的排列，由P42=4×3=12种

综上得，共8+12=20种 此题中使用了捆绑法和插空法。

例2：A、B、C、D、E五个人排成一排，其中A、B两人不站一起，共有（ ）种站法。

A.120     B.72       C.48       D.24

选B  插空法

我们来这样考虑，因A、B两人不站一起，故可考虑的位置C、D、E，C、D、E三个人站在那有一共留出4个空，将A、B分别放入这4个空的不同的空中，那就是4个空中取2个空的全排列，即P42=12。这样考虑了之后，还有一点就是C、D、E三个人也存在一个排列问题，即P33=6，综上，共有6\*12=72种

例3：A、B、C、D、E五个人排成一排，其中A、B两人必须站一起，共有（ ）种站法。

A.120     B.72       C.48       D.24

选C  捆绑法

此题和上一题实质是一样的，我们来这样考虑，A、B两人既然必须站在一起，那么索性我们就把他们看成一个人，那么我们就要考虑其和C、D、E共4个人的全排列，即P44=24，又因为A、B两人虽然是站在一起了，但还要考虑一个谁在前谁在后的问题，这有两种情况，也就是P22=2，综上，共有48种。

例4：将8个完全相同的球放到3个不同的盒子中，要求每个盒子至少放一个球，一共有多少种方法？

A. 20     B.21       C.23       D.24

选B  插隔板法

解决这道题只需将8个球分成三组，然后依次将每一个组分别放到一个盒子中即可。8个球分成3个组可以这样，用2个隔板插到这8个球中，这样就分成了3个组。这时我们考虑的问题就转化成了我们在8个球的空隙中放2个隔板有多少种放法的问题。8个球有7个空隙，7个空隙要放2个隔板，就有C72种放法，即21种.

例5：有9颗相同的糖，每天至少吃1颗，要4天吃完，有多少种吃法？

A. 20     B.36       C.45       D.56

选D 插隔板法

原理同上，只需用3个隔板放到9颗糖形成的8个空隙中，即可分成4天要吃的。就有C83种。C83=56种。

1.马路上有编号1,2,3...10的十盏路灯，为节约用电而又不影响照明，可以把其中3盏灯关掉，但不可以同时关掉相邻的两盏，在两端的灯都不能关掉的情况下，有()种不同的关灯方法。

答：采用插隔板法，即8灯关3,余5灯亮,5灯之间6个空,插入3盏不亮灯即C（6,3）

2.房间里有8人，分别佩戴着从1号到8号的纪念章，任选3人记录其纪念章号码，最大的号码为6的概率()

答：8个里边任意选3个选法总数：C(3,8)=56

最大为6，则需要在1~5里边任意选2个：C(2,5)=10

所以10/56=5/28。

3.58同城北京租房列表页共有3个广告位，广告库中共有5个经纪人，每个经纪人发布了2条广告房源参与此列表页3个广告位的随机展示(即每条广告房源获得展示的概率是一样的),则此列表页展示时，同时展示同一个经纪人的两条房源的概率是（）

答：共5\*2=10条广告，广告列有3空位，所以是总共情形：C10 3=120；

同一经纪人（C5 1）的两条房源（共3条，已有2条，则需要在剩下的8条中选1条，C10-2 1）：C 5 1\*C 10-2 1=40；  
所以40/120=1/3。

格式一：

　　01.struct tagPhone

　　02.{

　　03.     char   A; //1

　　04.     int    B;//4

　　05.     short  C;//2

06.}Phone;

3\*4=12

　　格式二：

　　01.struct tagPhone

　　02.{

　　03.     char   A;//1

　　04.     short  C;//2

　　05.     int    B;//4

06.}Phone2;

2\*4 = 8

　　格式三：

　　01.struct tagPhone3

　　02.{

　　03.     char   A;

　　04.     char   B[2];

　　05.     char   C[4];

06.}Phone3;

1\*7 = 7

我们都知道，char类型占用1个字节，int型占用4个字节，short类型占用2个字节，long占用8个，double占用16个；

**union所占的空间大小是成员变量中所占空间最大的那个变量的字节数**

比如union test

{  
     short s;

     char  c;  
};

那么union的在内存中占的长度是sizeof(short)=2B

3.struct所占内存空间的大小则是所有成员变量的sizeof的总和。但是为了提高CPU对内存的存取速度许多编译器都对struct的变量存放出现了“对齐”，而且struct的总的字节数是struct的成员变量中最长的变量所占字节数的整数倍。所谓的“对齐”就是成员变量的偏移量应该是该变量所占字节长度的整数倍。

比如 struct test

{

     char c; //1B

     short s; //2B

     int i; //4B

     long l; //4B

     double d; //8B  
};

那么struct test在内存中所占的空间是：sizeof(char)+1+sizeof(short)+sizeof(int)+sizeof(long)+4+sizeof(double)=24B,其中的数字是作为填充，填充的内存空间没有实际的意义

比如 struct test1

{

      double d; //8B

     short s; //2B

     char c; //1B

};

那么struct test1在内存中所占的空间是：sizeof(double)+sizeof(short)+sizeof(char)+5=16B,其中的数字是作为填充，

|  |
| --- |
| union H{      int a;      int array[5];      char c;  };  struct I{      int a;      short b;      union H c;      char d[5];  }; |

　　根据对union的介绍，union H的大小sizeof(H)=20;在struct I中，a和b占据了前8个字节,c应该从4的整数倍开始排列，因此sizeof(I)=4+2+2(补齐）+20+5+3(补齐)=36。在计算机上的大小以及各个元素的其实地址如图10所示。

|  |
| --- |
| struct inner{      char a;      double b;      char c;  };    union data{      struct innner a;      int b;      char c;  }; |

　　其中sizeof(inner)=24,为data中的占用空间最大的元素；占用空间最大的元类型元素为b,为4字节,因此无需再额外的补全。所以sizeof(inner)=24。

|  |
| --- |
| union G  {      char  name[30];      double al;      char sex;      int age;      float height;    }; |

此时，G需要占用空间最大的是元素char name[30],但是元类型中占用空间最大的是double，为8个字节。但是联合体G需要是al的整数倍且大于等于30，因此sizeof(G)=32,

如图8所示。

int a=10,\*p; **//a变量地址和a变量地址所指向的内容**

p=&a

int a=10;

int \*p=&a;

首先我们可以理解 int\* 这个是要定义一个指针p，然后因为这个指针存储的是地址所以要对a取地址(&)将值赋给指针p，也就是说这个指针p指向a。

很多新手都会对这两种定义方法感到迷惑，其实他俩的意思是一样的。第一种定义方法定义了int型的变量a和指针p，然后将a的地址赋给p。第二种是在定义指针p的同时将a的地址赋给指针p。我们姑且理解为" int \* "是定义指针的标志。

这样我们就可以通过\*p来找到指针所指向的变量a的地址，然后对地址中的值(值是10)进行操作。

//接着上面的代码

printf("%p",p) //结果是一个地址(p指向的变量a的地址)。

printf("%d",\*p) //结果是10，变量a的值。

printf("%d",&p) //结果是一个地址(指针p的地址，因为指针也是一个变量自己也有地址的)

**数组名和指针的区别？**

  数组名是一个地址，它可以被理解为一个常指针(它只能指向本数组首元素的地址)。而指针可以指其他的变量等等。

int str[5]={1,2,3,4,5};int \*p=str;

printf("%d",\*p); //输出为1，数组的首元素。

printf("%d",\*str); //输出为1，数组的首元素。

printf("%d",str[0]); //输出为1，数组的首元素。

printf("%p",p); //输出为地址，数组的地址。

printf("%p",str); //输出为地址，数组的地址。

printf("%d",\*(p+1)); //输出为2，数组的第二个元素。

printf("%d",\*(srt+1)); //输出为2，数组的第二个元素。

什么是空指针，野指针？

      在函数内部对指针进行赋值操作后，在函数执行完毕后，该指针就成了一个空指针(变量从栈中弹出)。但是通过动态地址分配(堆空间)的指针不会有上述问题。

        free(p)后释放内存的指针为野指针。

删除一个new了的数组。有必要的话。比如非标准的类( new CMyClass)，在Type \*p = new Type[N]; delete []p;的最后最好再加一句:   p = NULL

野指针，**也就是指向不可用内存区域的指针**。通常对这种指针进行操作的话，将会使程序发生不可预知的错误。   
“野指针”不是NULL指针，是指向“垃圾”内存的指针

**二、指针p被free或者delete之后，没有置为NULL，为野指针**。例：

char \*p = (char \*) malloc(100);

strcpy(p, “hello”);

free(p); // p 所指的内存被释放，但是p所指的地址仍然不变

if(p != NULL) // 没有起到防错作用

strcpy(p, “world”); // 出错

**关于const对于指针的限制问题**

int a=10,b=20;int \* const p=&a; //无法使p指向其他地址。

p=&b; //这样的操作是不允许的。

\*p=20; //可以这样操作。

const int \* p=&a; //p可以指向其他地址。

p=&b; //可以操作。

\*p=10; //操作不被允许。

const int \* const p=&a //p不能指向其他地址，也不能改变地址中的值。

p=&b; //操作不被允许。

\*p=20; //操作不被允许。

1、条件概率：由P(B|A)＝P(AB)/P(A)得P(AB)＝P(B|A)•P(A)．  
2．在10个形状大小均相同的球中有6个红球和4个白球，不放回地依次摸出2个球，在第1次摸出红球的条件下，第2次也摸到红球的概率为(　　)  
A.3/5　　　  B.2/5　　　  
C.1/10　　　  D.5/9  
[答案]　D

红黑性质的二叉搜索树：

1. 每个节点或是红色的，或是黑色的

2. 根节点是黑色的

3.每个叶节点（NIL）是黑色的

4.如果一个节点是红色的，则它的两个子节点都是黑色的

5.对于每个几点，从该节点到其所有后代叶节点的简单路径上，均包含相同数目的黑色节点。

null既不是一种对象也不是一种类型，它仅是一种特殊的值，你可以将其赋给任何引用类型，也可以转化成任何类型。

但是null可以赋给变量类型，不可以赋给基本类型，如int，boolean等。

子类继承父类的方法是，控制符必须大于或等于父类的访问控制符

进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，**但线程之间没有单独的地址空间**，**一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮**，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程

1、interface 方法不能是private，如果是static的必须实现该方法。所以static不能修饰接口

2、有**虚拟内存，**可以申请大于物理内存的内存空间，但是32位系统最多只能寻址4GB空间。

例如：已知一棵二叉树的[前序遍历](https://www.baidu.com/s?wd=%E5%89%8D%E5%BA%8F%E9%81%8D%E5%8E%86&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBrjm4rynvuADvPjm3nywh0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHnsrHn3PHT1" \t "https://zhidao.baidu.com/question/_blank)和后序遍历序列分别**为ABC和CBA，**则以下四棵二叉树均符合要求：  
 A A A A  
 \ \ / /  
 B B B B  
 \ / / \  
 C C C C

start()用来启动一个线程，当调用start方法后，系统才会开启一个新的线程，进而调用run()方法来执行任务，而单独的调用run()就跟调用普通方法是一样的，已经失去线程的特性了。因此在启动一个线程的时候一定要使用start()而不是run()

A：增加索引会增加磁盘占用

B：建立索引可以提升查询速度，即读速度；但在一定程度上降低写速度

C：数据库一般使用B\*树作为索引

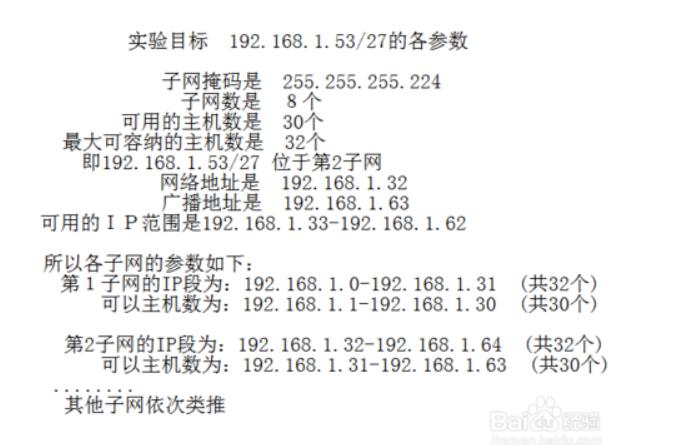
D：删除数据需要调整索引，所以会降低效率

关于D,在HTTP请求中的cookie是明文传递的，所以安全性成问题。（除非用HTTPS)

Cookie是客户端保存用户状态，Session是服务器端保存用户状态

Cookie是由服务器保存在客户端的文本文件，大小限制在4kb左右

多进程每个进程有自己的地址空间(address space)，线程则共享地址空间。所有其它区别都是由此而来的：   
1、**速度：**线程产生的速度快，线程间的通讯快、切换快等，因为他们在同一个地址空间内。   
2、**资源利用率：**线程的资源利用率比较好也是因为他们在同一个地址空间内。   
3、**同步问题：**线程使用公共变量/内存时需要使用同步机制还是因为他们在同一个地址空间内



子网掩码：子网数：可用的主机数：最大容纳的主机数

然后本题分析：父进程为P，  
第一个fork()后，产生S1；  
第二个fork()后，P产生一个子进程S2，S1产生一个子进程S1\_1(S1\_1的父进程为S1)；  
第三个fork()后，P产生子进程S3，S2产生子进程S2\_1,S1进程产生S1\_2;S1\_1产生一个子进程S1\_1\_1  
因此调用printf的进程有P，由P产生的S1，S2，S3；由S1产生的S1\_1，S1\_2；由S2产生的S2\_1;由S1\_1产生的S1\_1\_1，共有8个

1.物理独立性是指用户的应用程序与存储在磁盘上的数据库中数据是[相互独立](https://www.baidu.com/s?wd=%E7%9B%B8%E4%BA%92%E7%8B%AC%E7%AB%8B&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnW63nh7BrH-hmvwbuWRz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWf3nHnvnHfsrjmkP1R1rHmd" \t "https://zhidao.baidu.com/question/_blank)的。即，数据在磁盘上怎样存储由DBMS管理，用户程序不需要了解，应用程序要处理的只是数据的[逻辑结构](https://www.baidu.com/s?wd=%E9%80%BB%E8%BE%91%E7%BB%93%E6%9E%84&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnW63nh7BrH-hmvwbuWRz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWf3nHnvnHfsrjmkP1R1rHmd" \t "https://zhidao.baidu.com/question/_blank)，这样当数据的物理存储改变了，应用程序不用改变。  
2.逻辑独立性 逻辑独立性是指用户的应用程序与数据库的[逻辑结构](https://www.baidu.com/s?wd=%E9%80%BB%E8%BE%91%E7%BB%93%E6%9E%84&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnW63nh7BrH-hmvwbuWRz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWf3nHnvnHfsrjmkP1R1rHmd" \t "https://zhidao.baidu.com/question/_blank)是[相互独立](https://www.baidu.com/s?wd=%E7%9B%B8%E4%BA%92%E7%8B%AC%E7%AB%8B&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnW63nh7BrH-hmvwbuWRz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWf3nHnvnHfsrjmkP1R1rHmd" \t "https://zhidao.baidu.com/question/_blank)的，即，当数据的[逻辑结构](https://www.baidu.com/s?wd=%E9%80%BB%E8%BE%91%E7%BB%93%E6%9E%84&from=1012015a&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnW63nh7BrH-hmvwbuWRz0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWf3nHnvnHfsrjmkP1R1rHmd" \t "https://zhidao.baidu.com/question/_blank)改变时，用户程序也可以不变。  
前者是应用程序与数据库的数据，后者是应用程序与数据库的逻辑结构

白盒测试中语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、条件组合覆盖

黑盒测试：等价类划分、边界值分析

1，Session 是放在服务器上的。

2，Cookie 是放在本地浏览器的，单个大小不会超过 4KB 一个站点最多保存 20 个。

3，服务器会对每个用户生成一个 Session 其中 SessionID 用来区分用户。

4，在本地保存这个 SessionID 可以将其保存在 Cookie 中。

5，Session 在服务器上，一定时间之后是由服务器销毁的。

所谓Belady现象是指：采用FIFO算法时，如果对—个进程未分配它所要求的全部页面，有时就会出现分配的页面数增多但缺页率反而提高的异常现象。 FIFO的性能较差，

第一次握手序列号是x；

第二次握手序列号y，确认号x+1；

第三次握手序列号x+1，确认号y+1

&a是数组指针，其类型为int(\*)[5];

而指针加1要根据指针类型加上一定的值，不同类型的指针+1之后增加的大小不同，a是长度为5的int数组指针，所以要加5\*sizeof(int)，所以ptr实际是a[5],但是ptr与（&a+1）类型是不一样的，这点非常重要，所以ptr-1只会减去sizeof(int\*),a，&a的地址是一样的，但意思就不一样了，a是数组首地址，也就是a[0]的地址，&a是对象（数组）首地址，a+1是数组下一元素的地址，即a[1],&a+1是下一个对象的地址，即a[5]。

1、main是主线程，在main中创建了thread线程，在main中调用了thread.join()，那么等thread结束后再执行main代码。

2、物理页可以在进程间共享，两个进程共使用了1,2,3,4,6,7,9，共7个物理页。

7\*512=3584

3、const p\_char pa 被宏定义替换为const char \*pa 是一个 指向常量字符的指针

const char\_p pb 被解释为char \* const pb 是一个指针常量

1. cat  由第一行开始显示档案内容   
   tac  从最后一行开始显示，可以看出 tac 是 cat 的倒着写！   
   more 一页一页的显示档案内容   
   less 与 more 类似，但是比 more 更好的是，他可以往前翻页！   
   head 只看头几行   
   tail 只看尾巴几行
2. 排序算法中不稳定的算法有：快速排序，简单选择排序，希尔排序，堆排序。（速记：“简兮快怼”）
3. **选择排序是不稳定的排序方法**
4. 静态代码块内容先执行，接着执行父类非静态代码块和构造方法，然后执行子类非静态代码块和构造方法
5. A 进程不同于线程，各自拥有独立的地址空间 B 主线程是否影响其他线程取决于二者之间的依赖关系 C进程和线程都可能引起死锁
6. 每一次读取volatile修饰的变量都会从内存中读取；volatile修饰的变量每次必须去地址读，而不能用上一次的缓存，所以读取速度反而会**慢**
7. 在linux环境下，查看日志文件的最后100行数据的正确方式是？

tail -100 a.log

1. 在linux环境下，查看日志文件的最前100行数据的正确方式是？

head -100 a.log

11、硬链接相当于允许文件用友多个有效的路径名，只有当删除最后一个硬链接时源文件才删除

软连接相当于快捷方式，也成为符号链接，包含源文件位置信息，源文件删除的同时，所有软链接随之删除

12、数据链路层：ARP,RARP

网络层： IP,ICMP,IGMP

传输层：TCP ,UDP,UGP

应用层：Telnet,FTP,SMTP,SNMP.

1. 使用ping 命令 ping 网站服务器地址，检查服务器是否正常; 4. 登陆服务器，使用**netstat -ano**命令检查80端口是否打开； 5. 登陆服务器，使用 **ps -aux|grep http**命令检查 http 进程是否存在；
2. 用两个栈模拟队列先进先出，模拟其add和romve功能，给出思路和代码。
3. 给两个字符串，输出其最长共同字符串的长度：如  
    S1: asdfghjqweryuiase   
    S2: astyfrtfghjqwsa   
    其最长共同字符串为fghjqw 长度为6，给出代码。

16、

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 操作方式 | 数据位置 | 明文密文 | 数据安全 | 长度限制 | 应用场景 |
| GET | **HTTP包头** | **明文** | **不安全** | **长度较小** | **查询数据** |
| POST | **HTTP正文** | 可明可密 | 安全 | 支持较大数据传输 | 修改数据 |

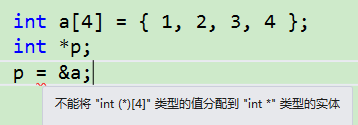
****1.范围：要回收哪些区域？****

**Java方法栈、本地方法栈以及PC计数器随方法或线程的结束而自然被回收，**

所以这些区域不需要考虑回收问题。Java堆和方法区是GC回收的重点区域，

2、将7723810的各位数字打乱排序,可组成的不同的7位自然数的个数是?

A66 / 2 \*6

1. 注意题目所说的是有向图，因此是7条边，若为无向图，则是6条边；**强连通图**
2. 有9个球,其中一个的质量与其他的不同,有一个天平,通过最多几次可以找出那个质量不一样的球?
3. ****
4. 如果要添加新列，正确的语句为：alter table table-name add id int；不需要添加column

1.实践中如何优化[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "https://blog.csdn.net/a724888/article/details/_blank)

**1) SQL语句及索引的优化**

**2) [数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "https://blog.csdn.net/a724888/article/details/_blank)表结构的优化**

**3) 系统配置的优化**

**4) 硬件优化**

**数据库索引**，**是数据库管理系统中一个排序的数据结构**，以协助快速查询、更新数据库表中数据。**索引的实现通常使用B树及其变种B+树。**

**为表设置索引要付出代价的**：一是增加了数据库的存储空间，二是在**插入和修改数据时要花费较多的时间**(因为索引也要随之变动)。

**原理：**每个节点分别**包含索引键值和一个指向对应数据记录物理地址的指针**，这样就可以运用二叉查找在O(log2n)的复杂度内获取到相应数据

8.数据库优化的思路

这个我借鉴了慕课上关于数据库优化的课程。

1. **SQL语句优化**
2. 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。  
   2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：  
   select id from t where num is null  
   可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：  
   select id from t where num=0  
   3）很多时候用 exists 代替 in 是一个好的选择  
   4）用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤
3. 避免在WHERE子句中使用in，not  in，

可以使用 exist 和not exist代替 in和not in。

4) 不要在建立的索引的数据列上进行下列操作:  
**（1）避免对索引字段进行计算操作**

**（2）避免在索引字段上使用not，<>，!=**

**（3）避免在索引列上使用IS NULL和IS NOT NULL**

**（4）避免在索引列上出现数据类型转换**

**（5）避免在索引字段上使用函数**

例如：where trunc(create\_date)=trunc(:date1)  
虽然已对create\_date 字段建了索引，但由于加了TRUNC，使得索引无法用上。此处正确的写法应该是  
where create\_date>=trunc(:date1) and create\_date

（6）避免建立索引的列中使用空值。

5) 查询的模糊匹配

**3.什么情况下设置了索引但无法使用，索引无效**

**1) 以”%”开头的LIKE语句，模糊匹配：**红色标识位置的百分号会导致相关列的索引无法使用

2) Or语句前后没有同时使用索引

3) 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型，会使索引无效，产生全表扫描。）

**4) 在索引列上使用IS NULL 或IS NOT NULL操作。索引是不索引空值的**，所以这样的操作不能使用索引，可以用其他的办法处理，例如：数字类型，判断大于0，字符串类型设置一个默认值，判断是否等于默认值即可

**5) 在索引字段上使用not，<>，!=，eg<> 操作符（不等于）**：不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。 用其它相同功能的操作运算代替，如 a<>0 改为 a>0 or a<0

**6) 对索引字段进行计算操作**

**7) 在索引字段上使用函数**

**2.索引优化**

看上文索引

**3.数据库结构优化**

1）范式优化： 比如消除冗余（节省空间。。） 2）反范式优化：比如适当加冗余等（减少join） 3）拆分表： 分区将数据在物理上分隔开，不同分区的数据可以制定保存在处于不同磁盘上的数据文件里。这样，当对这个表进行查询时，只需要在表分区中进行扫描，而不必进行全表扫描，明显缩短了查询时间，另外处于不同磁盘的分区也将对这个表的数据传输分散在不同的磁盘I/O，一个精心设置的分区可以将数据传输对磁盘I/O竞争均匀地分散开。对数据量大的时时表可采取此方法。可按月自动建表分区。  
4）拆分其实又分垂直拆分和水平拆分： 案例： 简单购物系统暂设涉及如下表： 1.产品表（数据量10w，稳定） 2.订单表（数据量200w，且有增长趋势） 3.用户表 （数据量100w，且有增长趋势） 以[mysql](http://www.2cto.com/database/MySQL/" \t "https://blog.csdn.net/boonya/article/details/_blank)为例讲述下水平拆分和垂直拆分，mysql能容忍的数量级在百万静态数据可以到千万 垂直拆分： 解决问题：表与表之间的io竞争 不解决问题：单表中数据量增长出现的压力 方案： 把产品表和用户表放到一个server上 订单表单独放到一个server上 水平拆分： 解决问题：单表中数据量增长出现的压力 不解决问题：表与表之间的io争夺  
方案： **用户表通过性别拆分为男用户表和女用户表** **订单表通过已完成和完成中拆分为已完成订单和未完成订单** 产品表 **未完成订单放一个server上 已完成订单表盒男用户表放一个server上 女用户表放一个server上(女的爱购物 哈哈)**

**4.服务器硬件优化**

这个么多花钱咯！

**4.如何设计一个高并发的系统**

1) 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引优化

2) 使用缓存、尽量减少数据库IO

3) 分布式数据库、分布式缓存

4) 服务器的负载均衡