**017/7/4 京东Java开发 内推**

1. TCP/UDP的区别，两者的使用场景

①两者区别：

TCP的优点： 可靠，稳定

TCP的缺点： 慢，效率低，占用系统资源高，易被攻击（因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用。传输前需要连接，连接需要维护。

UDP的优点： 快，比TCP稍安全

UDP的缺点： 不可靠，不稳定

https://www.cnblogs.com/xiaomayizoe/p/5258754.html

②使用场景：

对数据有实时性要求的选择UDP

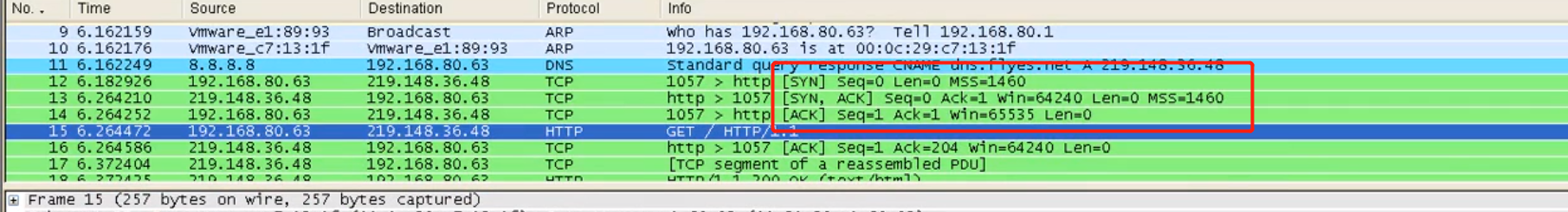
* 如果客户端和服务器都可以独立发包，但是偶尔发生延迟可以容忍（比如：在线的纸牌游戏，许多MMO类的游戏），那么使用**TCP长连接**吧。
* 如果客户端和服务器都可以独立发包，而且无法忍受延迟（比如：大多数的多人动作类游戏，一些MMO类游戏），那么使用**UDP**吧。

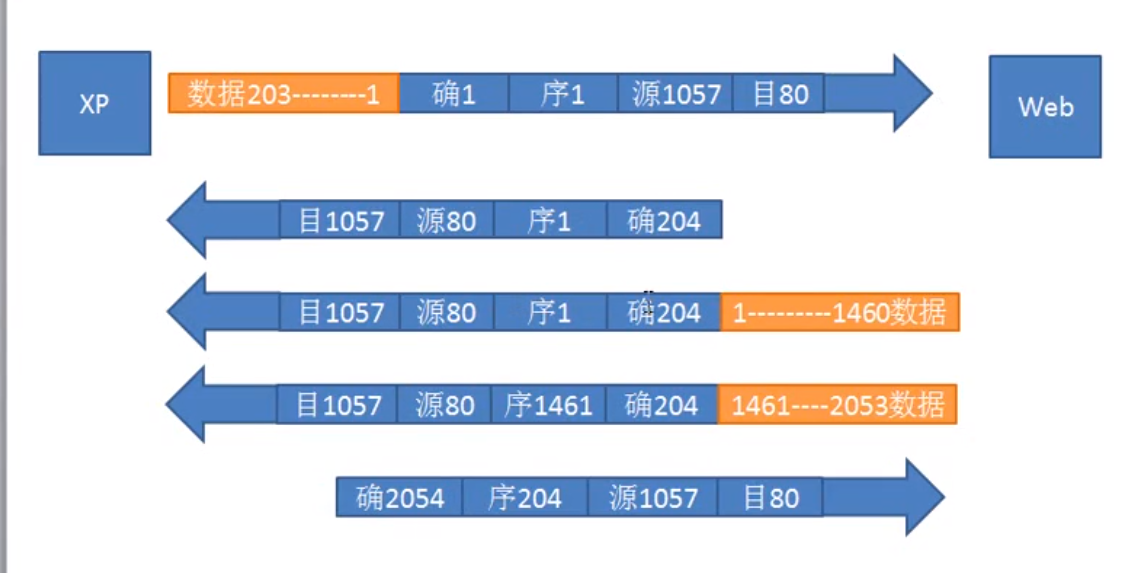
UDP:QQ聊天，视频聊天

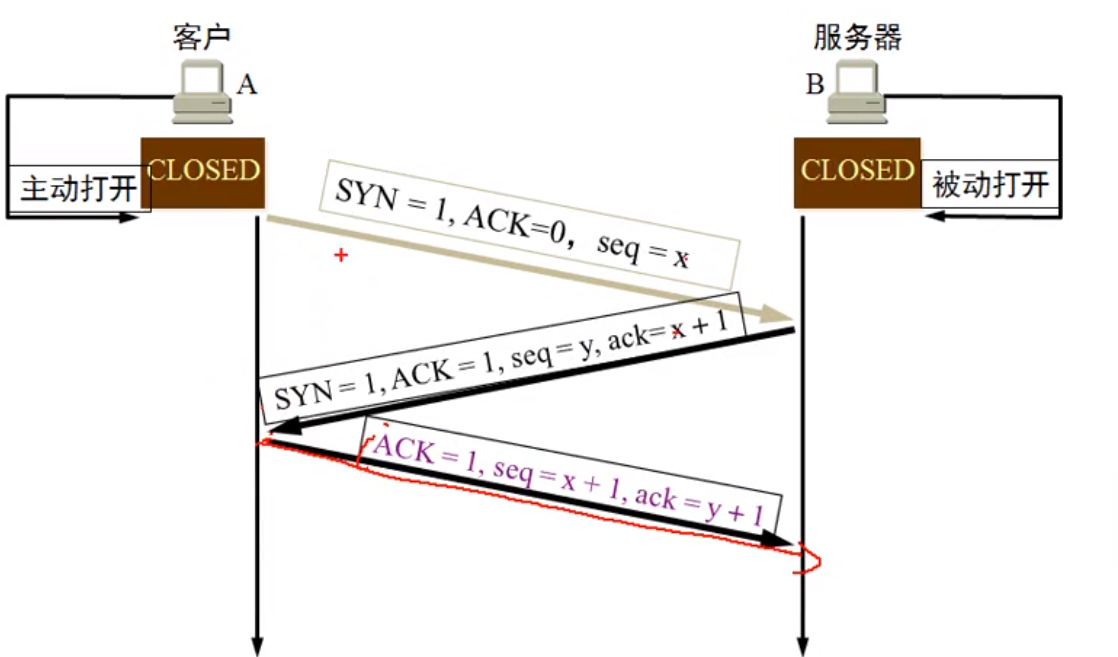
TCP:下载

拿视频网站来说，你完全可以缓冲 20s 再播放，不会带来什么影响，但如果画面有马赛克之类的东西出现肯定是不好的，所以用 TCP。  
而对于视频聊天，如果缓冲 5s，相信整个聊天已经没法愉快的进行了，而这时出现一些画面质量的损失也可以被接受，所以用 UDP。

三次握手：

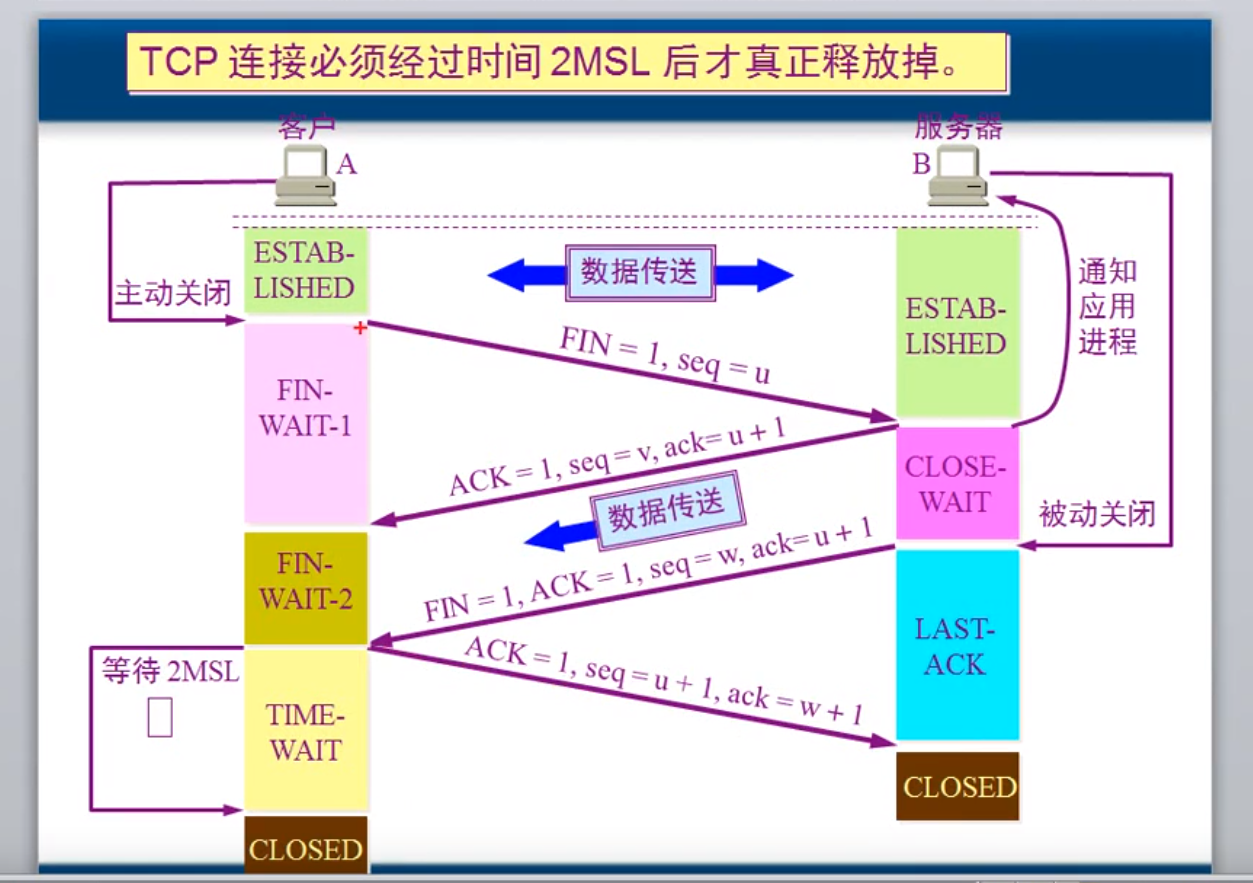


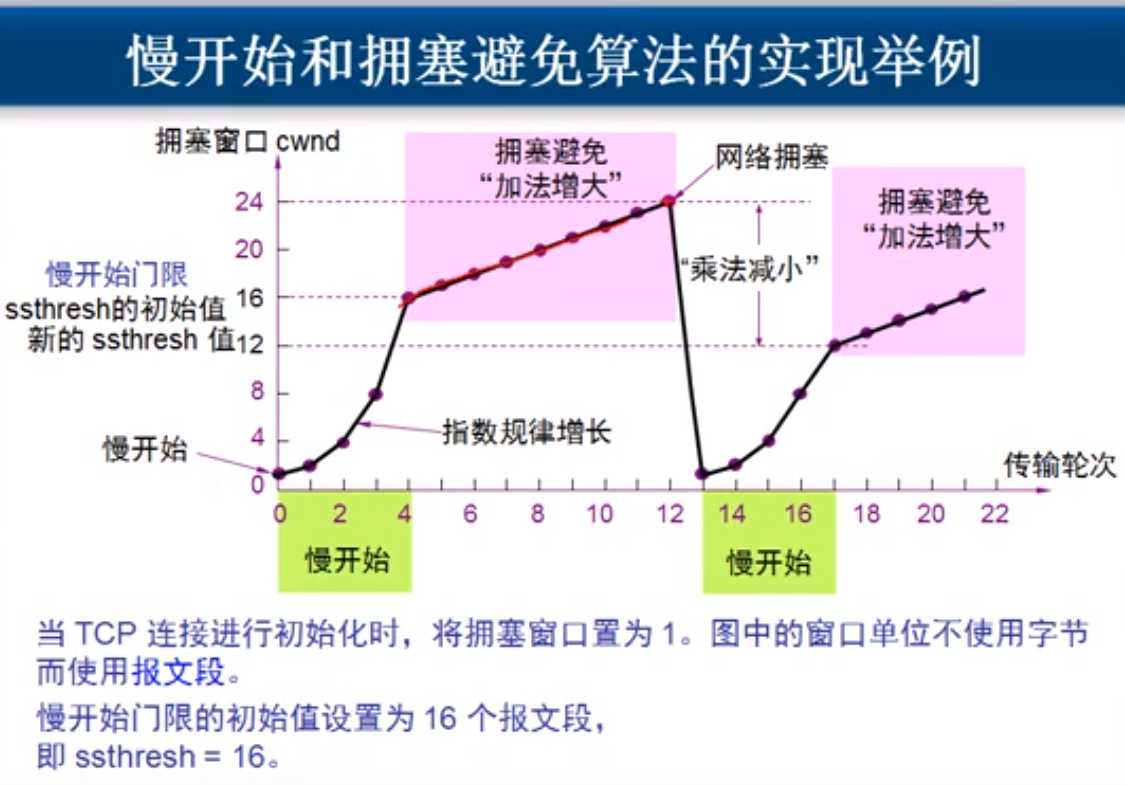


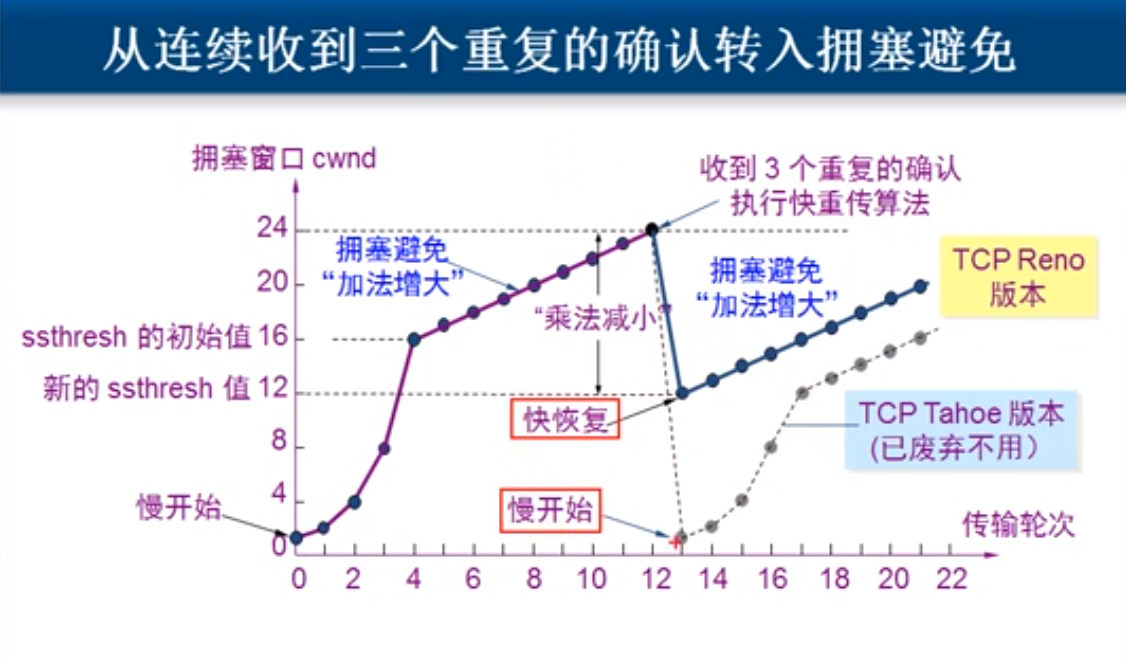


为什么需要第三次：如果A发了两次连接请求给B，一次快一次慢，B收到快的请求后给A回复，A三次握手建立连接。这时候B又收到了A的慢请求，任然给A回复，如果没有三次握手，就会占用B的资源，有了三次握手的话，B如果一定时间内没收到A的回复，就会放弃等待。

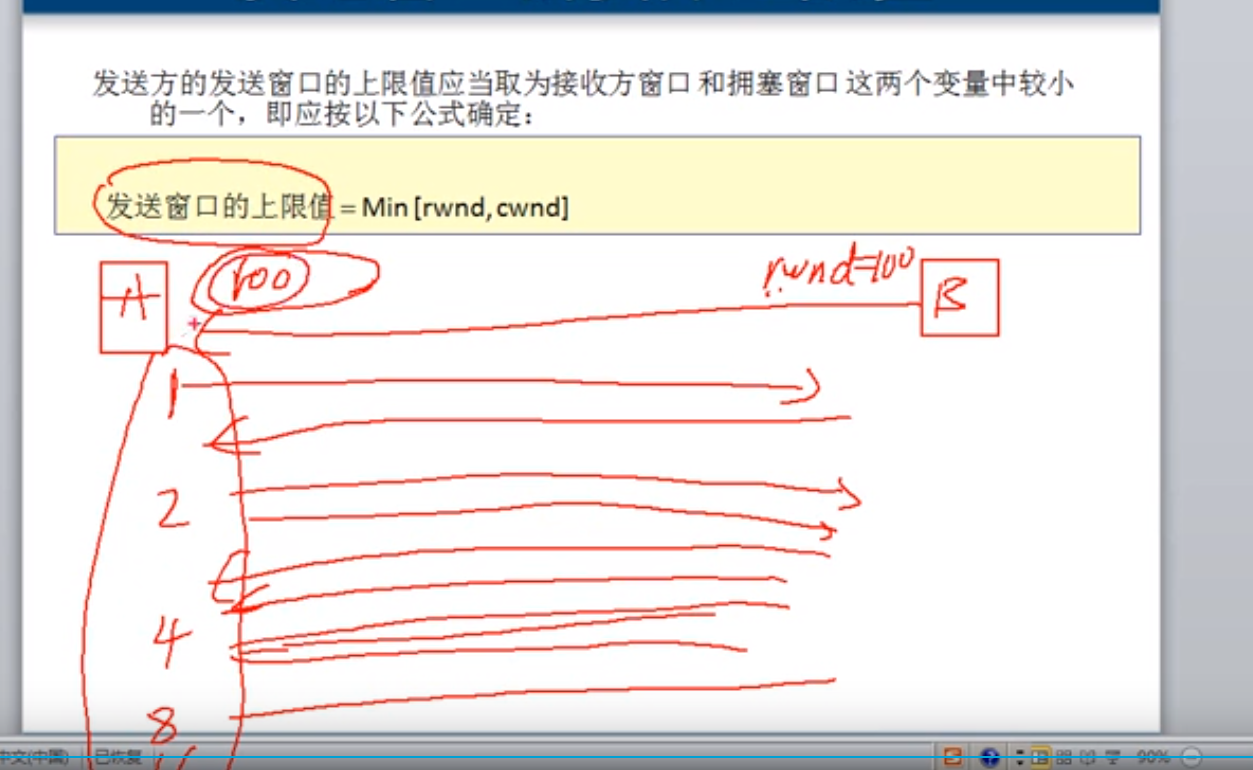
流量控制：是通过滑动窗口来控制的







<https://www.cnblogs.com/hongdada/p/11206679.html>



1. Hashmap底层实现原理，为什么线程不安全

①HashMap的底层是一个Entry数组结构，数组中的每一项对应了一个链表，这种结构称“链表散列”的数据结构，即数组和链表的结合体；也叫散列表、哈希表。

HahMap存储对象的过程如下：

1、对HahMap的Key调用hashCode()方法，返回int值，即对应的hashCode；

2、把此hashCode作为哈希表的索引，查找哈希表的相应位置，若当前位置内容为NULL，则把hashMap的Key、Value包装成Entry数组，放入当前位置；

3、若当前位置内容不为空，则继续查找当前索引处存放的链表，利用equals方法，找到Key相同的Entry数组，则用当前Value去替换旧的Value；

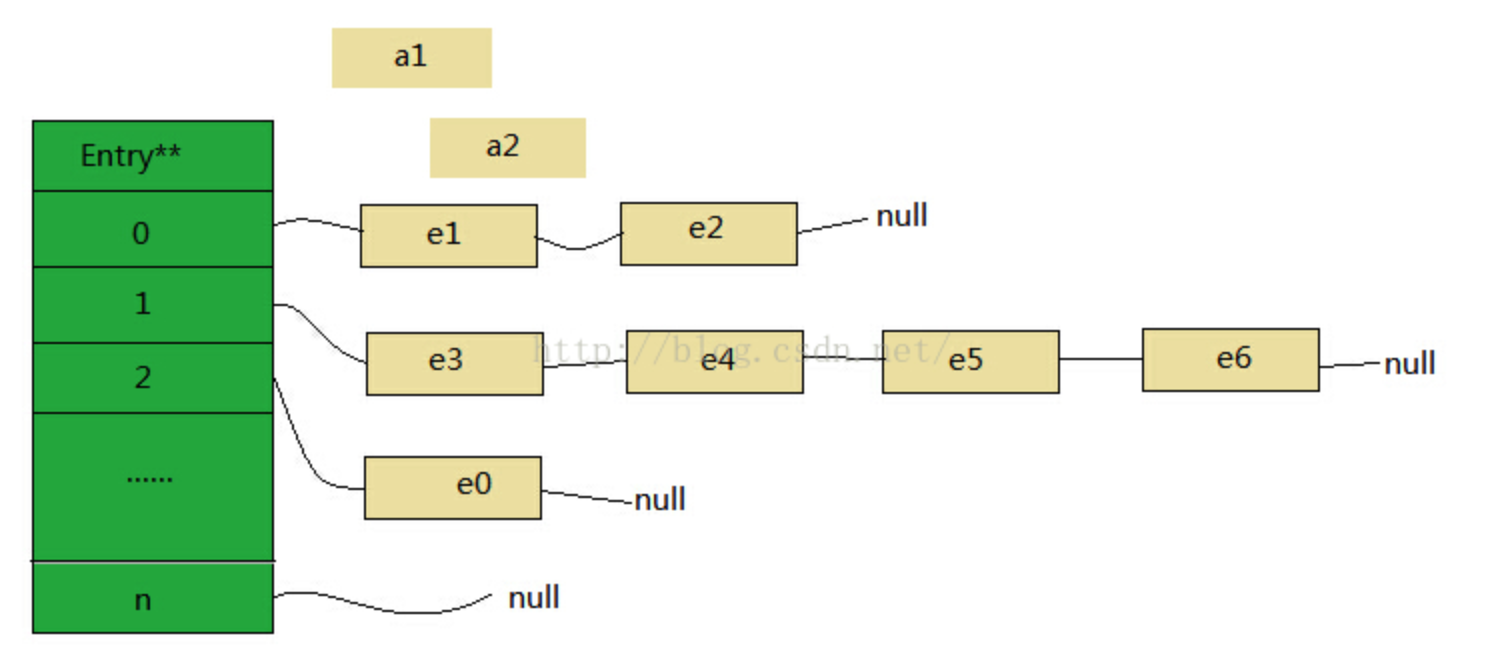
4、若未找到与当前Key值相同的对象，则把当前位置的链表后移（Entry数组持有一个指向下一个元素的引用），把新的Entry数组放到链表表头；在jdk1.8之前是插入头部的，在jdk1.8中是插入尾部的（插在头部可以避免尾部遍历。尾部遍历是为了避免在新列表插入数据时，遍历队尾的位置。JDK1.8的优化：通过增加tail指针，既避免了死循环问题（让数据直接插入到队尾），又避免了尾部遍历。）

**在JDK1.8中，当链表长度超过阈值（8）时，将链表转换为红黑树，这样大大减少了查找时间。在同一个位置的个数又达到了8个以上，如果数组的长度还小于64的时候，则会扩容数组。如果数组的长度大于等于64了的话，在会将该节点的链表转换成树。**

首先有一个每个元素都是链表（可能表述不准确）的数组，当添加一个元素（key-value）时，就首先计算元素key的hash值，以此确定插入数组中的位置，但是可能存在同一hash值的元素已经被放在数组同一位置了，这时就添加到同一hash值的元素的后面，他们在数组的同一位置，但是形成了链表，同一各链表上的Hash值是相同的，所以说数组存放的是链表。而当链表长度太长时，链表就转换为红黑树，这样大大提高了查找的效率**。**

**当链表数组的容量超过初始容量的0.75时，再散列将链表数组扩大2倍，把原链表数组的搬移到新的数组中**

②通过Entry内部的next变量可以知道使用的是链表，这时候我们可以知道，如果多个线程，在某一时刻同时操作HashMap并执行put操作，而有大于两个key的hash值相同，如图中a1、a2，这个时候需要解决碰撞冲突，而解决冲突的办法上面已经说过，对于链表的结构在这里不再赘述，暂且不讨论是从链表头部插入还是从尾部初入，这个时候两个线程如果恰好都取到了对应位置的头结点e1，而最终的结果可想而知，a1、a2两个数据中势必会有一个会丢失，如图所示：



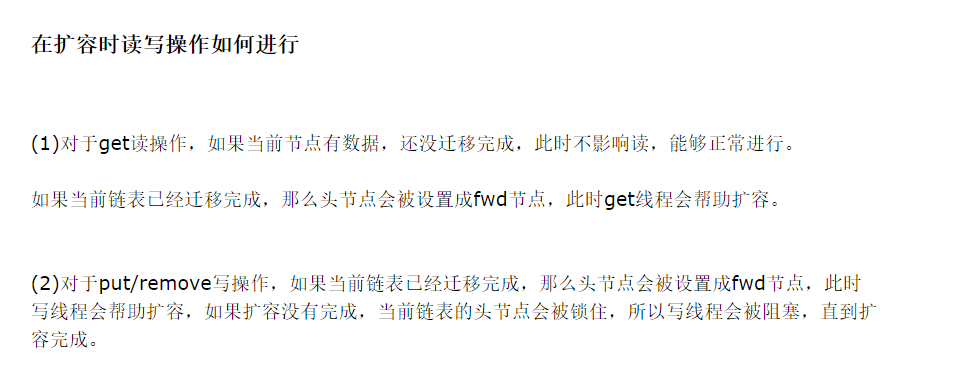
可以看到扩容方法resize()也不是同步的，通过代码我们知道在扩容过程中，会新生成一个新的容量的数组，然后对原数组的所有键值对重新进行计算和写入新的数组，之后指向新生成的数组。

当多个线程同时检测到总数量超过门限值的时候就会同时调用resize操作，各自生成新的数组并rehash后赋给该map底层的数组table，结果最终只有最后一个线程生成的新数组被赋给table变量，其他线程的均会丢失。而且当某些线程已经完成赋值而其他线程刚开始的时候，就会用已经被赋值的table作为原始数组，这样也会有问题。



扩容时，把旧数组上的节点移到扩容后的数组上，已经移动完成的节点或者空节点设为fwd节点，hash值为moved。其他线程碰到还没移动完的节点就会对该节点加锁，然后移动该节点

<https://www.cnblogs.com/zerotomax/p/8687425.html>



<https://www.cnblogs.com/lfs2640666960/p/9621461.html>

1. 为什么要使用连接池

数据库连接是非常占用资源的，尤其是在高并发的情况下，如果每次都去建立数据库连接就会有性能问题，也会影响一个应用程序的延展性，

数据库连接池负责分配、管理、释放连接，它保证应用程序可以重复使用同一个连接而不需要每次都建立数据库连接，如果数据库连接时间超过设置的最长数据库连接时间会自动释放链接，为了避免因为没有释放链接而导致的数据库连接泄露，因此，数据库连接池可以明显的提高数据库的连接性能。

1. 进程和线程的区别

进程：进程是资源（CPU、内存等）分配的基本单位，它是程序执行时的一个实例，Linux系统函数fork()可以在父进程中创建一个子进程。

线程：线程是程序执行时的最小单位。

一个程序至少一个进程，一个进程至少一个线程。

* 地址空间：同一进程的线程共享本进程的地址空间，而进程之间则是独立的地址空间。
* 资源拥有：同一进程内的线程共享本进程的资源如内存、I/O、cpu等，但是进程之间的资源是独立的。

　　　　　一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程都死掉。所以多进程要比多线程健壮。

　　　　　进程切换时，消耗的资源大，效率高。所以涉及到频繁的切换时，使用线程要好于进程。同样如果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

* 执行过程：每个独立的进程程有一个程序运行的入口、顺序执行序列和程序入口。但是线程不能独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

1. Java异常

Java标准库内建了一些通用的异常，这些类以Throwable为顶层父类。

Throwable又派生出Error类和Exception类。

错误：Error类以及他的子类的实例，代表了JVM本身的错误。错误不能被程序员通过代码处理，Error很少出现。因此，程序员应该关注Exception为父类的分支下的各种异常类。

异常：Exception以及他的子类，代表程序运行时发送的各种不期望发生的事件。可以被Java异常处理机制使用，是异常处理的核心。

1. Java查看日志

**https://blog.csdn.net/roadmore/article/details/78650488**

cat -n test.log |grep “地形” | more 得到关键日志的行号

cat -n test.log |tail -n +92|head -n 20   
tail -n +92表示查询92行之后的日志   
head -n 20 则表示在前面的查询结果里再查前20条记录

cat -n test.log |grep “地形” >xxx.txt

grep ‘2014-12-17 16:17:20’ test.log

sed -n ‘/2014-12-17 16:17:20/,/2014-12-17 16:17:36/p’ test.log



9.29 招银网络电面

1. 常用的线程池（重点）

**①newCachedThreadPool**

* 工作线程的创建数量几乎没有限制(其实也有限制的,数目为Interger. MAX\_VALUE), 这样可灵活的往线程池中添加线程。
* 如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为1分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。SynchronousQueue

**②newFixedThreadPool**

* 创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。
* FixedThreadPool是一个典型且优秀的线程池，它具有线程池提高程序效率和节省创建线程时所耗的开销的优点。但是，在线程池空闲时，即线程池中没有可运行任务时，它不会释放工作线程，还会占用一定的系统资源。LinkedblockingQueue

③newsingleThreadExecutor

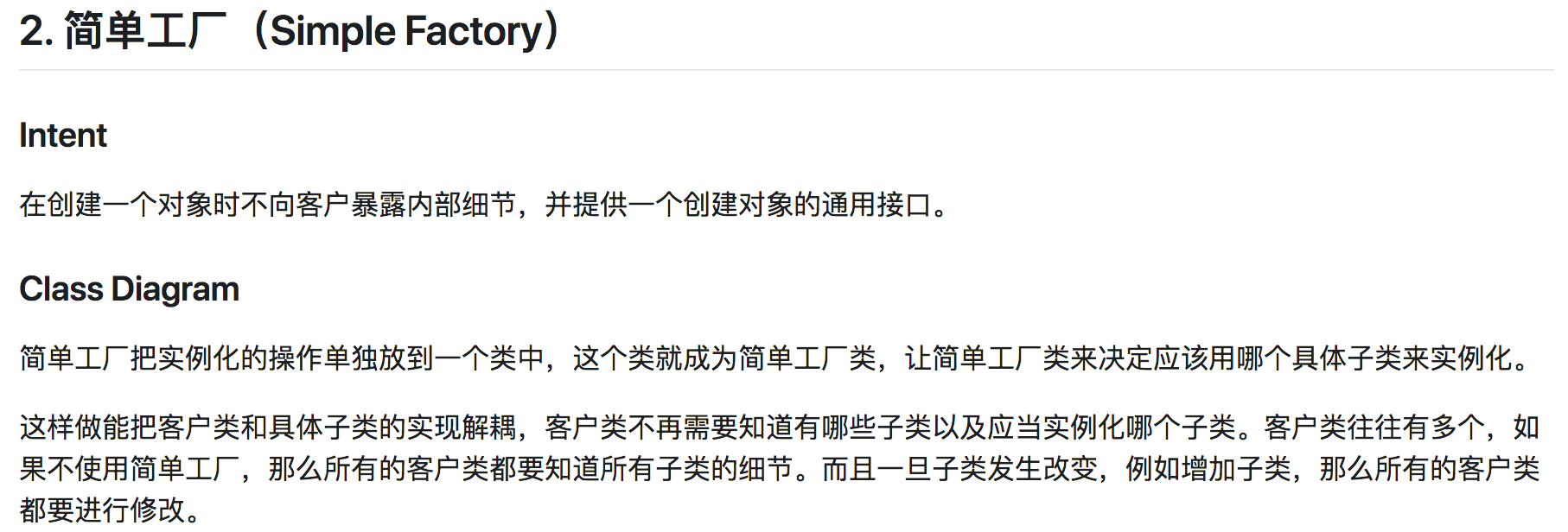
创建一个单线程化的Executor，即只创建唯一的工作者线程来执行任务，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO,优先级)执行。如果这个线程异常结束，会有另一个取代它，保证顺序执行。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的。LinkedblockingQueue

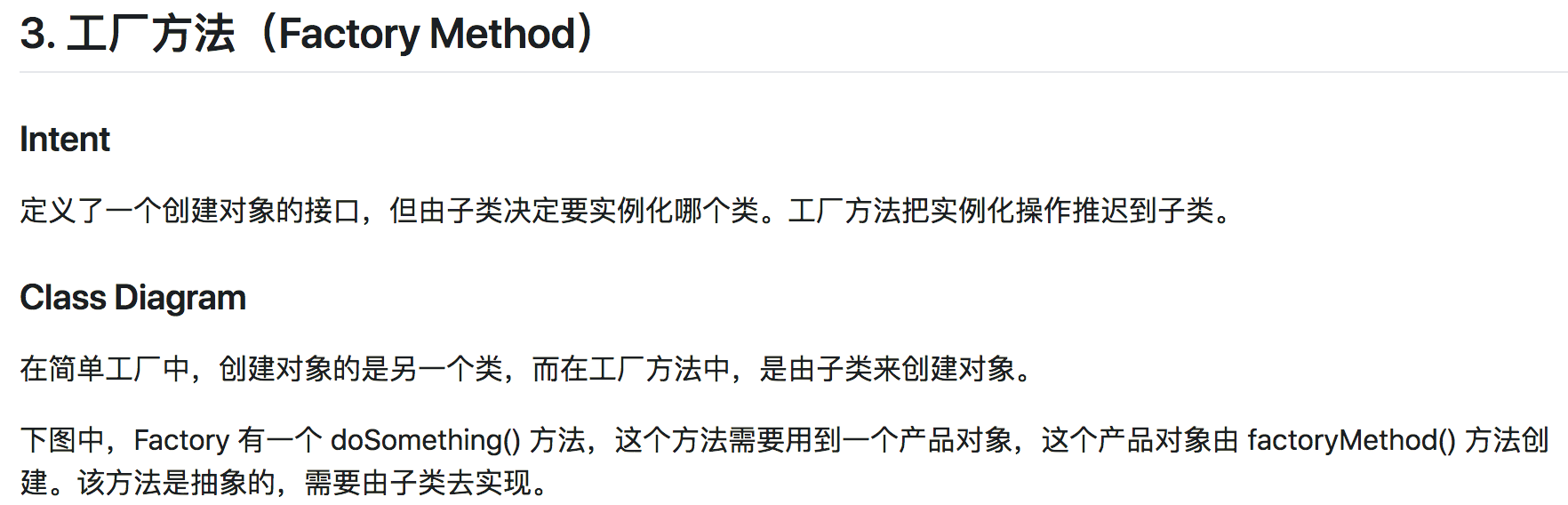
④newScheduleThreadPool

创建一个定长的线程池，而且支持定时的以及周期性的任务执行，支持定时及周期性任务执行。

线程池的核心参数有哪些？如果队列满了且达到了最大线程数怎么办？饱和策略（拼多多提前批一面）

1. spring IOC AOP
2. session和cookie
3. 设计模式（重点）







对象池：

<https://www.cnblogs.com/songjy2116/p/7747790.html>

1. 数据库索引及使用
2. 堆栈的区别

1.栈里面存储的是基本类型的变量和对象的引用变量；堆里面存储的是由new创建的对象和数组

2. 栈存取速度比堆要快，仅次于直接位于CPU中的寄存器, 但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性，堆的优势是可以动态地分配内存大小，在运行时动态分配内存，存取速度较慢

3.栈可以实现数据共享

、

10.12 平安科技一面

1. String为何为不可变类

(1)不可变对象可以提高String Pool(字符串常量池)的效率和安全性。如果你知道一个对象是不可变动 ，那么需要拷贝的对象的内容时就不用复制它本身二只是复制它的地址，复制地址(通常一个指针的大小)需要很小的内存，效率也很好。二对于其他引用同一个对象的其他变量也不会造成影响。

　　(2)可不变对象对于多线程滴安全的，因为在多线程同时进行的情况下，一个可变对象的值很可能被其他线程改变这样会造成不可预期的结果么人使用不可变对象就可以避免这种情况出现。

（3）因为字符串是不可变的，所以在它创建的时候hashcode就被缓存了，不需要重新计算。这就使得字符串很适合作为Map中的键，字符串的处理速度要快过其它的键对象。这就是HashMap中的键往往都使用字符串。

String类不可变性的好处

1.只有当字符串是不可变的，字符串池才有可能实现。字符串池的实现可以在运行时节约很多heap空间，因为不同的字符串变量都指向池中的同一个字符串。但如果字符串是可变的，那么String interning将不能实现(译者注：String interning是指对不同的字符串仅仅只保存一个，即不会保存多个相同的字符串。因为这样的话，如果变量改变了它的值，那么其它指向这个值的变量的值也会一起改变。  
2.如果字符串是可变的，那么会引起很严重的安全问题。譬如，数据库的用户名、密码都是以字符串的形式传入来获得数据库的连接，或者在socket编程中，主机名和端口都是以字符串的形式传入。因为字符串是不可变的，所以它的值是不可改变的，否则黑客们可以钻到空子，改变字符串指向的对象的值，造成安全漏洞。  
3.因为字符串是不可变的，所以是多线程安全的，同一个字符串实例可以被多个线程共享。这样便不用因为线程安全问题而使用同步。字符串自己便是线程安全的。  
4.类加载器要用到字符串，不可变性提供了安全性，以便正确的类被加载。譬如你想加载java.sql.Connection类，而这个值被改成了myhacked.Connection，那么会对你的数据库造成不可知的破坏。  
5.因为字符串是不可变的，所以在它创建的时候hashcode就被缓存了，不需要重新计算。这就使得字符串很适合作为Map中的键，字符串的处理速度要快过其它的键对象。这就是HashMap中的键往往都使用字符串。

1. Spring 默认是单例什么时候使用多例

1. 什么是单例多例: 所谓单例就是所有的请求都用一个对象来处理，比如我们常用的service和dao层的对象通常都是单例的，而多例则指每个请求用一个新的对象来处理，比如action;

2. 如何产生单例多例: 在通用的SSH中，单例在spring中是默认的，如果要产生多例，则在配置文件的bean中添加scope="prototype";

3. 为什么用单例多例： 之所以用单例，是因为没必要每个请求都新建一个对象，这样子既浪费CPU又浪费内存； 之所以用多例，是为了防止并发问题；即一个请求改变了对象的状态，此时对象又处理另一个请求，而之前请求对对象状态的改变导致了对象对另一个请求做了错误的处理； 用单例和多例的标准只有一个： 当对象含有可改变的状态时（更精确的说就是在实际应用中该状态会改变），则多例，否则单例；

4. 何时用单例？何时用多例？ 对于struts2来说，action必须用多例，因为action本身含有请求参数的值，即可改变的状态； 而对于STRUTS1来说，action则可用单例，因为请求参数的值是放在actionForm中，而非action中的； 另外要说一下，并不是说service或dao一定是单例，标准同第3点所讲的，就曾见过有的service中也包含了可改变的状态，同时执行方法也依赖该状态，但一样用的单例，这样就会出现隐藏的BUG,而并发的BUG通常很难重现和查找

1. 对象的创建方式和类的获取方式

对象的创建方式：new 反射机制（newInstance) clone() 反序列化

类的获取方式：Class.forName("类的路径"） 类名.Class 实例.getClass

1. HashMap底层hashcode的计算方式，如何实现HashMap线性安全

https://blog.csdn.net/qq\_38182963/article/details/78940047

http://www.cnblogs.com/zqsky/p/6777654.html

当length总是2的n次方时，  h & (length - 1)   等价于   hash对length取模

Map m = Collections.synchronizedMap(new HashMap())

1. hashmap的加载因子有什么用，加载因子为0或1会怎样；hashmap的初始容量为什么是16。

加载因子是表示Hsah表中元素的填满的程度.若:加载因子越大,填满的元素越多,好处是,空间利用率高了,但:冲突的机会加大了.反之,加载因子越小,填满的元素越少,好处是:冲突的机会减小了,但:空间浪费多了.  
加载因子越高   空间利用率提高了  但是查询时间 和添加时间增加

https://blog.csdn.net/qq\_35583089/article/details/80048285

反观长度16或者其他2的幂,length - 1的值是所有二进制位全为1,这种情况下,index的结果等同于hashcode后几位的值  
只要输入的hashcode本身分布均匀,hash算法的结果就是均匀的  
所以,HashMap的默认长度为16,是为了降低hash碰撞的几率

<https://www.cnblogs.com/owenma/p/8551505.html>



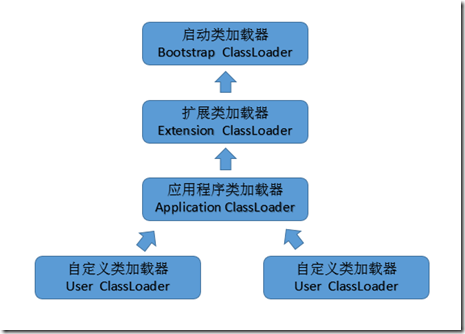
1. #{}和${}可以替换吗？

#{}: 解析为一个 JDBC 预编译语句（prepared statement）的参数标记符，一个 #{ } 被解析为一个参数占位符 。

   ${}: 仅仅为一个纯碎的 string 替换，在动态 SQL 解析阶段将会进行变量替换。

#{}能够防止SQL注入

1. 类加载



https://www.cnblogs.com/fanjie/p/6916784.html

加载-链接（验证-准备-解析）-初始化

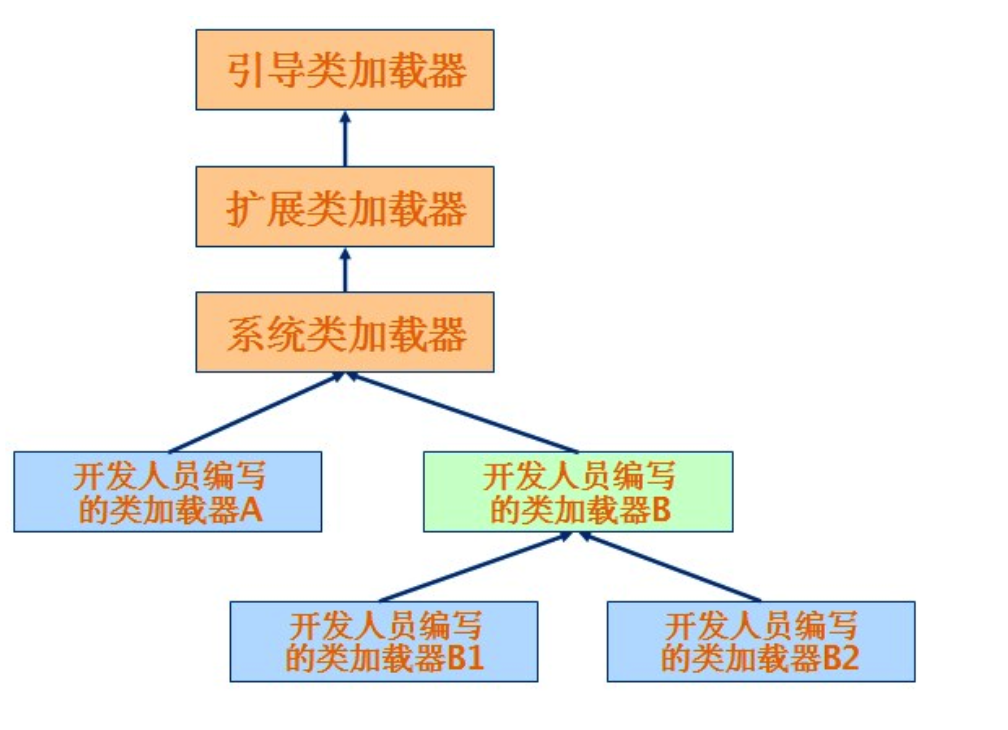
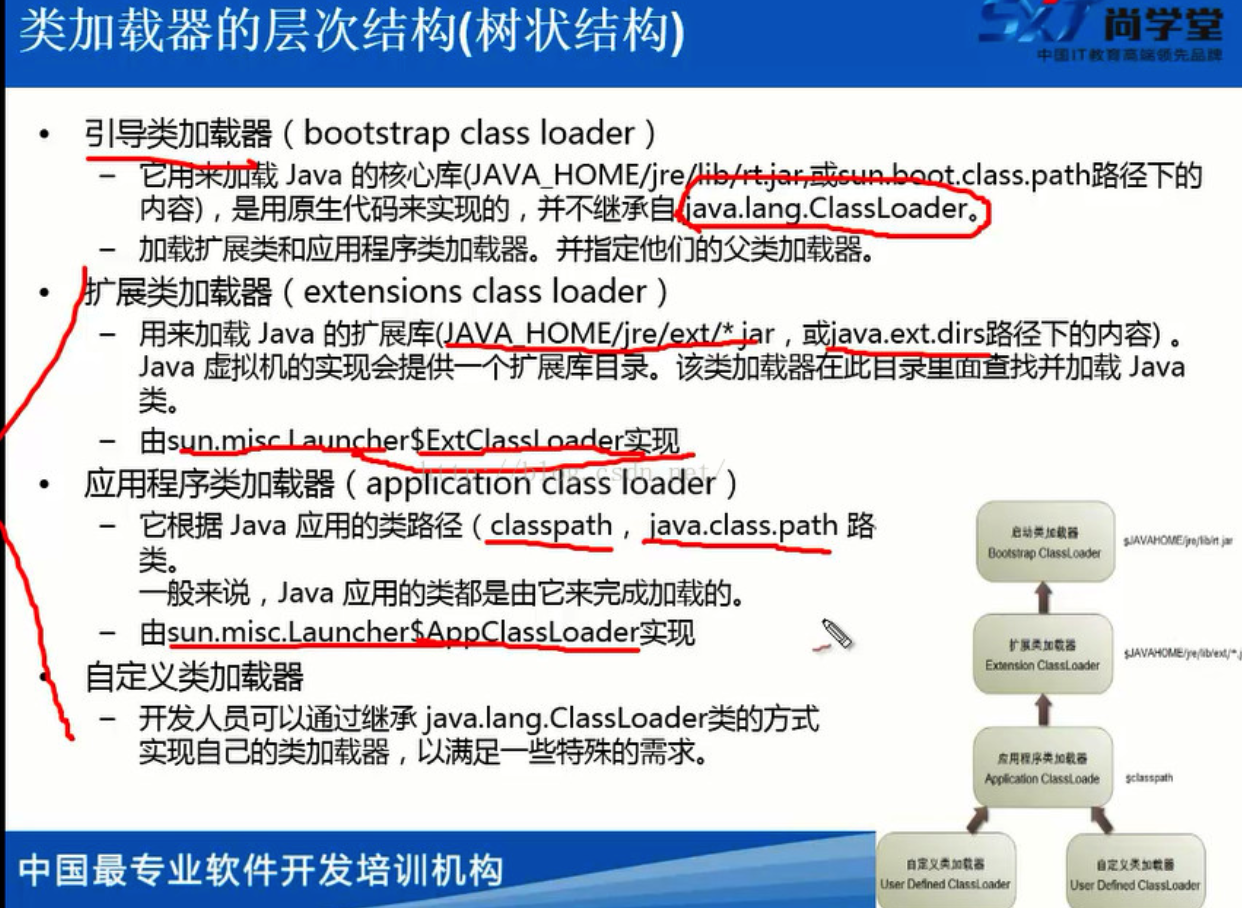
1. HTTP是有连接还是无连接的

HTTP是无状态链接，因为要实现有状态链接的效果，所以采用session和cookie两种机制。 客户端请求需要包含服务器指定的唯一的sessionID，如果请求中没有sessionID，服务器将为该请求创建一个session和与之相关联的 sessionID并把这个sessionID放入响应之中，并加上一行特殊的指示以提示浏览器按照指示生成相应的cookie，将响应返回给客户端，客 户端的浏览器按照规则解析响应，如果浏览器支持cookie则将sessionID和其他信息存储起来（persistent cookie 或者session cookie），下次再次访问同一服务器时候浏览器将对应的cookie和请求头一起发送给服务器，服务器按照sessionID进行检索找到对应的 session进行处理。

HTTP是基于TCP的，HTTP使用TCP而不是UDP的原因在于（打开）一个网页必须传送很多数据，而[TCP协议](https://www.baidu.com/s?wd=TCP%E5%8D%8F%E8%AE%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)提供传输控制，按顺序组织数据，和错误纠正。

TCP的面向连接是传输层的，而HTTP的无状态则是应用层的

1. 类加载器有哪些

1. Java内存泄露

1.静态集合类，比如hashmap和vector。如果这些容器是静态的，由于他们的生命周期和程序一致，那么容器中的对象在程序结束前是不会被释放的。

2.各种连接，例如数据库的连接、网络连接和IO连接等

3.监听器，在释放对象的时候却没有去删除这些监听器，增加了内存泄漏的机会。

4.变量不合理的作用域，把函数的中变量定义到类中了。

5.单例模式

**6、内部类持有外部类，**如果一个外部类的实例对象的方法返回了一个内部类的实例对象，这个内部类对象被长期引用了，即使那个外部类实例对象不再被使用，但由于内部类持有外部类的实例对象，这个外部类对象将不会被垃圾回收，这也会造成内存泄露。

**7、改变哈希值，**当一个对象被存储进HashSet集合中以后，就不能修改这个对象中的那些参与计算哈希值的字段了，否则，对象修改后的哈希值与最初存储进HashSet集合中时的哈希值就不同了，在这种情况下，即使在contains方法使用该对象的当前引用作为的参数去HashSet集合中检索对象，也将返回找不到对象的结果，这也会导致无法从HashSet集合中单独删除当前对象，造成内存泄露

1. UDP如何实现稳定可靠

 传输层无法保证数据的可靠传输，只能通过应用层来实现了。实现的方式可以参照tcp可靠性传输的方式，只是实现不在传输层，实现转移到了应用层。

         实现确认机制、重传机制、窗口确认机制。

内存的使用情况、查看哪个端口被哪个进程占用、查看日志、磁盘使用情况 df

1、free -m（以MB的方式查看）（-h自动选择合适的单位)

2、netstat -anp|grep 8080 ps -ef|grep firefox（进程）

3、cat head tail（tail -f 实时查看日志，默认10行)

4、df显示磁盘空间使用情况 du 查看目录大小

du，disk usage,是通过搜索文件来计算每个文件的大小然后累加，du能看到的文件只是一些当前存在 的，没有被删除的。他计算的大小就是当前他认为存在的所有文件大小的累加和。  
df，disk free，通过文件系统来快速获取空间大小的信息，当我们删除一个文件的时候，这个文件不 是马上就在文件系统当中消失了，而是暂时消失了，当所有程序都不用时，才会根据OS的规则释放掉已 经删除的文件， df记录的是通过文件系统获取到的文件的大小，他比du强的地方就是能够看到已经删除 的文件，而且计算大小的时候，把这一部分的空间也加上了，更精确了。当文件系统也确定删除了该文件后，这时候du与df就一致了。

1. ps和top的区别

Ps 只为您提供当前进程的快照。要即时查看最活跃的进程，可使用 top。  
Top 实时地提供进程信息。它还拥有交互式的状态，允许用户输入命令，如 n 后面跟有 5 或 10 等数字。其结果是指示 top 显示 5 或 10 个最活跃的进程。Top 持续运行，直到您按 "q" 退出 top 为止。

1. 线程的五种状态以及转换

　1、新状态：线程对象已经创建，还没有在其上调用start()方法。

　　2、就绪状态：当线程有资格运行，但调度程序还没有把它选定为运行线程时线程所处的状态。当start()方法调用时，线程首先进入可运行状态。在线程运行之后或者从阻塞、等待或睡眠状态回来后，也返回到可运行状态。

　　3、运行状态：线程调度程序从可运行池中选择一个线程作为当前线程时线程所处的状态。这也是线程进入运行状态的唯一一种方式。

　　4、等待/阻塞/睡眠状态：这是线程有资格运行时它所处的状态。实际上这个三状态组合为一种，其共同点是：线程仍旧是活的，但是当前没有条件运行。换句话说，它是可运行的，但是如果某件事件出现，他可能返回到可运行状态。

　　5、死亡态：当线程的run()方法完成时就认为它死去。这个线程对象也许是活的，但是，它已经不是一个单独执行的线程。线程一旦死亡，就不能复生。如果在一个死去的线程上调用start()方法，会抛出java.lang.IllegalThreadStateException异常。

Thread.sleep(long millis)和Thread.sleep(long millis, int nanos)静态方法强制当前正在执行的线程休眠（暂停执行），以“减慢线程”。当线程睡眠时，它入睡在某个地方，在苏醒之前不会返回到可运行状态。当睡眠时间到期，则返回到可运行状态。

yield()方法的作用是：暂停当前正在执行的线程对象，并执行其他线程。

Thread的非静态方法join()让一个线程B“加入”到另外一个线程A的尾部。在A执行完毕之前，B不能工作。

1. sleep方法和wait方法异同点是什么？

相同点：可以让线程处于冻结状态。

不同点：

1.Sleep必须指定时间。Wait可以指定时间，可以不指定时间。

2. Sleep时间到，线程处于临时阻塞或者运行,Wait如果没有指定时间，必须要通过notify或者notifyAll唤醒。

3.Sleep不一定非要定义在同步中。Wait一定要定义在同步中。

4.如果都定义在同步中，线程执行到sleep，不会释放锁。线程执行到wait，会释放锁。

1. 请求头

Accept:浏览器能够处理的内容类型

Accept-Charset:浏览器能够显示的字符集

Accept-Encoding：浏览器能够处理的压缩编码

Accept-Language：浏览器当前设置的语言

Referer：发出请求的页面的URL

User-Agent：浏览器的用户代理字符串

**Cache-Control**

1. 二叉树

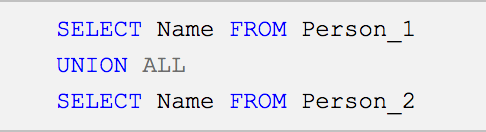
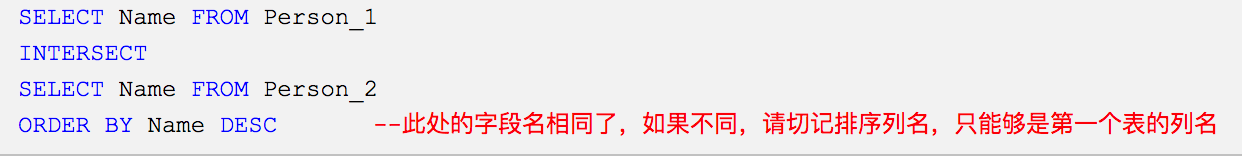
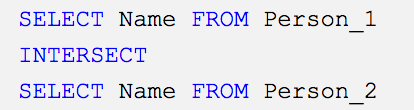
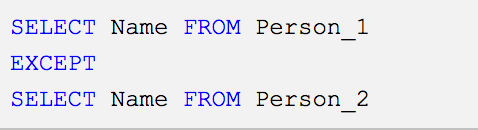
(1)[完全二叉树](https://baike.baidu.com/item/%E5%AE%8C%E5%85%A8%E4%BA%8C%E5%8F%89%E6%A0%91)——若设二叉树的高度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，第h层有[叶子结点](https://baike.baidu.com/item/%E5%8F%B6%E5%AD%90%E7%BB%93%E7%82%B9)，并且叶子结点都是从左到右依次排布，这就是[完全二叉树](https://baike.baidu.com/item/%E5%AE%8C%E5%85%A8%E4%BA%8C%E5%8F%89%E6%A0%91)。

(2)[满二叉树](https://baike.baidu.com/item/%E6%BB%A1%E4%BA%8C%E5%8F%89%E6%A0%91)——除了叶结点外每一个结点都有左右子叶且叶子结点都处在最底层的二叉树。

(3)[平衡二叉树](https://baike.baidu.com/item/%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91/10421057)——[平衡二叉树](https://baike.baidu.com/item/%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91/10421057)又被称为AVL树（区别于[AVL](https://baike.baidu.com/item/AVL/7543015)算法），它是一棵二叉排序树，且具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵[平衡二叉树](https://baike.baidu.com/item/%E5%B9%B3%E8%A1%A1%E4%BA%8C%E5%8F%89%E6%A0%91/10421057)。

1. SQL的并集、差集、交集

UNION-EXCEPT-INTERSECT

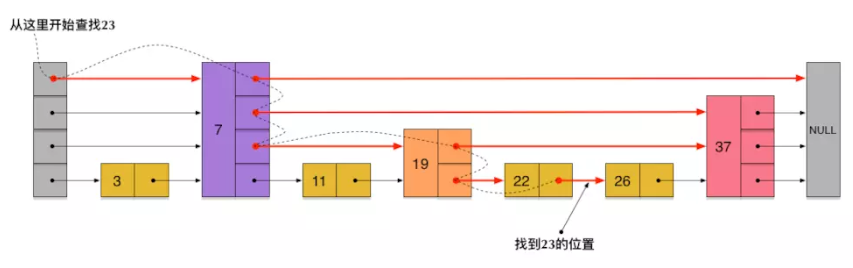
30、redis数据结构：

Redis 数据类型分为：字符串类型、散列类型、列表类型、集合类型、有序集合类型。

列表类型存储了一个有序的字符串列表。常用的操作是向两端插入新的元素。时间复杂度为 O（1）。结构为一个链表。记录头和尾的地址。

有序集合是使用散列表和跳跃表实现的。所以和列表相比，操作中间元素的速度也很快。时间复杂度尾 O(log(N))

在做范围查找的时候，平衡树比skiplist操作要复杂。平衡树的插入和删除操作可能引发子树的调整，逻辑复杂，而skiplist的插入和删除只需要修改相邻节点的指针，操作简单又快速。



Zset的数据结构<https://www.jianshu.com/p/cc379427ef9d>

31、

1. -Xms 为jvm启动时分配的内存，比如-Xms200m，表示分配200M

2. -Xmx 为jvm运行过程中分配的最大内存，比如-Xms500m，表示jvm进程最多只能够占用500M内存

3. -Xss 为jvm启动的每个线程分配的内存大小，默认JDK1.4中是256K，JDK1.5+中是1M

-Xmn ：设置年轻代大小。整个堆大小=年轻代大小 + 年老代大小 + 持久代大小 。持久代一般固定大小为64m，所以增大年轻代后，将会减小年老代大小。此值对系统性能影响较大，Sun官方推荐配置为整个堆的3/8。

32、JWT：<https://blog.csdn.net/why15732625998/article/details/78534711>

33、https:

服务端将公钥发给客户端，客户端生成个客户端秘钥，将这个客户端秘钥使用公钥加密发给服务端，服务端使用私钥解密，得到客户端秘钥，讲返回的信息用这个客户端秘钥加密发给客户端。

<https://blog.csdn.net/qq_38289815/article/details/80969419>

<https://www.jianshu.com/p/14cd2c9d2cd2>

项目里的单点登录<https://www.2cto.com/kf/201807/763241.html>

34、redis设置过期时间的原理（拼多多提前批）：

<https://blog.csdn.net/qq_27469747/article/details/80128184>

待整理

<https://www.nowcoder.com/discuss/225574?type=all&order=time&pos=&page=1>



35、BIO、NIO、AIO 区别和应用场景

<https://blog.csdn.net/lisha006/article/details/82856906>

对比BIO的同步阻塞IO操作，实际上NIO是同步非阻塞IO，一个线程在同步的进行轮询检查，Selector不断轮询注册在其上的Channel，某个Channel上面发生读写连接请求，这个Channel就处于就绪状态，被Selector轮询出来，然后通过SelectionKey可以获取就绪Channel的集合，进行后续的I/O操作。

NIO：会等数据准备好后，再交由应用进行处理，数据的读取/写入过程依然在应用线程中完成，只是将等待的时间剥离到单独的线程中去，节省了数据准备时间，因为多路复用机制，Selector会得到复用，对于那些读写过程时间长的，NIO就不太适合。

AIO：读完（内核内存拷贝到用户内存）了系统再通知应用，使用回调函数，进行业务处理，AIO能够胜任那些重量级，读写过程长的任务。

AIO不需要通过多路复用器对注册的通道进行轮询操作即可实现异步读写。什么意思呢？NIO采用轮询的方式，一直在轮询的询问stream中数据是否准备就绪，如果准备就绪发起处理。但是AIO就不需要了，AIO框架在windows下使用windows IOCP技术，在Linux下使用epoll多路复用IO技术模拟异步IO， 即：应用程序向操作系统注册IO监听，然后继续做自己的事情。操作系统发生IO事件，并且准备好数据后，在主动通知应用程序，触发相应的函数（这就是一种以订阅者模式进行的改造）。由于应用程序不是“轮询”方式而是订阅-通知方式，所以不再需要selector轮询，由channel通道直接到操作系统注册监听。

36、阻塞队列

ArrayBlockingQueue与LinkedBlockingQueue的比较

相同点：ArrayBlockingQueue和LinkedBlockingQueue都是通过condition通知机制来实现可阻塞式插入和删除元素，并满足线程安全的特性；

不同点：1. ArrayBlockingQueue底层是采用的数组进行实现，而LinkedBlockingQueue则是采用链表数据结构；

ArrayBlockingQueue插入和删除数据，只采用了一个lock，而LinkedBlockingQueue则是在插入和删除分别采用了putLock和takeLock，这样可以降低线程由于线程无法获取到lock而进入WAITING状态的可能性，从而提高了线程并发执行的效率。

PriorityBlockingQueue通过cas来保证扩容安全，扩容时会释放lock保证性能

37、传统的进程间通信的手段：

（1）管道（Pipe）：管道可用于具有亲缘关系进程间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。

（2）命名管道（named pipe）：命名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。命名管道在文件系统中有对应的文件名。

（3）信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；类似于中断

（4）消息队列：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺

（5）共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

（6）内存映射（mapped memory）：内存映射允许任何多个进程间通信，每一个使用该机制的进程通过把一个共享的文件映射到自己的进程地址空间来实现它。

（7）信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。

（8）套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。

而在java中我们实现多线程间通信则主要采用"共享变量"和"管道流"这两种方法