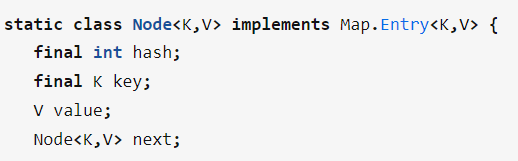
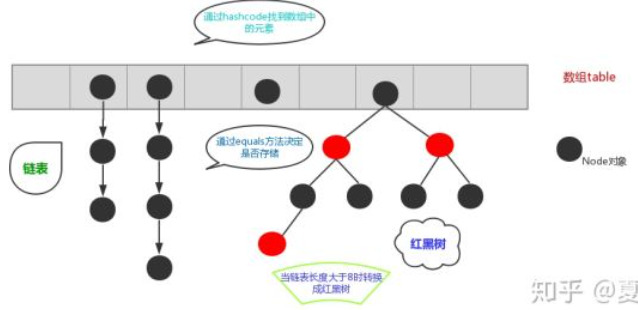
1. **哈希表的实现，底层是什么，特点是什么。**

源码中每个节点是Node<K,V>



Node是一个内部类，这里的key为键，value为值，next指向下一个元素，可以看出HashMap中的元素不是一个单纯的键值对，还包含下一个元素的引用。

由数组+链表（或红黑树）组成，主干是一个Node<K,V>数组（也就是下图的table），初始值为空:



数组的每个位置都对应一个hash，同属于这个hash的有链表（或红黑树）上的所有Node。

在执行put操作时，会先计算出key的hashcode，再对该hashcode取一次hash来定位要将该元素放到数组的哪个位置。

HashMap中有两个重要的参数：初始容量大小和加载因子，初始容量大小是创建时给数组分配的容量大小，默认值为16，用数组容量大小乘以加载因子得到一个值，一旦数组中存储的元素个数超过该值就会调用rehash方法将数组容量增加到原来的两倍，专业术语叫做扩容。在做扩容的时候会生成一个新的数组，原来的所有数据需要重新计算哈希码值重新分配到新的数组，所以扩容的操作非常消耗性能。

1. **给10w个字符串，每个长10个字符，设计一个数据结构及算法存储数据，并判断输入一个新字符串是否在已有的字符串中**。

自己想的：类似位图的结构，列是所有字符，对应有那个字符的话那一位就是1，否则是0，输入的新字符串也获得一个位图索引，然后寻找有没有这个索引对应的。

1. **100w个数据取出topk，方法及时间复杂度，有什么优化的方法。**

* 优化的快排。O(k\*100w)

采用快排中的partition划分思想，即单趟划分后，枢轴s前面的数据都比他小，后面的数据都比他大，此时看s的下标，如果等于k-1则直接返回s；如果大于k-1，则对它前面的部分进行递归排序；如果小于k则对它后面的部分进行递归排序。

* 堆排序。O(lgk\*100w)

使用一个大小为k的小顶堆，每次拿堆顶元素，k次之后就拿到了第k小的数。。

1. 什么是符号表？C和C++的符号表有什么区别？
2. memcpy是怎么实现的？
3. **虚拟内存作用？**

虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。

1. **new和malloc区别？new的实现**

* new是c++关键字，需要编译器支持；malloc是库函数，需要头文件支持
* new做两件事：分配内存和调用类的构造函数，delete是：调用类的析构函数和释放内存。而malloc和free只是分配和释放内存。

1. 什么是伪共享？怎么解决？
2. **如何从海量数据中选出最大的1000个数？用多线程怎么实现？**

（1）优化的快排（见第3题）

（2）首先把数组分成若干个部分，让这些部分在不同的线程里进行排序，然后进行对这些数组进行归并 https://blog.csdn.net/m0\_56190554/article/details/127701465

1. **链表中倒数第k个元素**

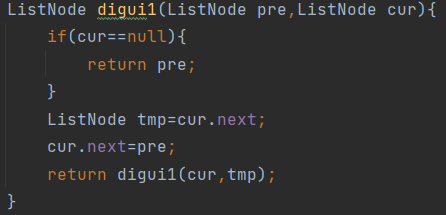
双指针，设置p1和p2，p1先在head的位置，p2往后移动k个位置，

然后p1跟着p2往后移动，当p2移动到最后面的null的时候，p1就到达了倒数第k个位置。

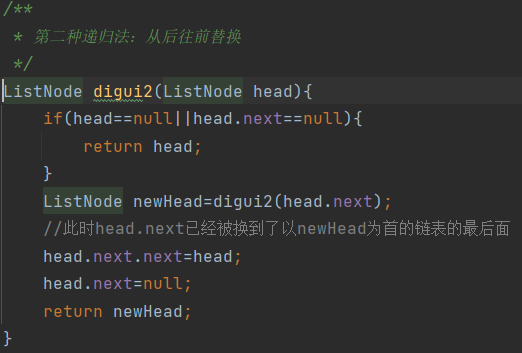
1. **原地反向输出链表**

* 递归法：

-第一种递归法：类比迭代法，从前往后翻转：

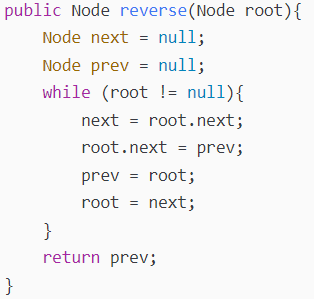


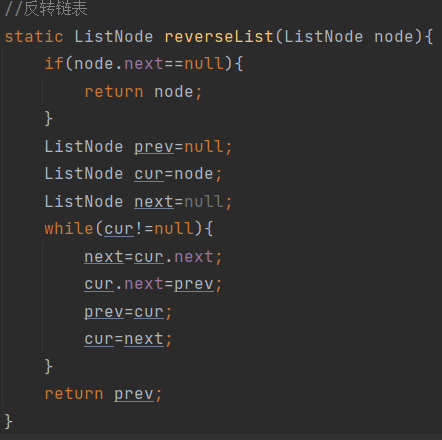
-第二种递归法：从后往前翻转：



* 非递归法：

prev是现在这个节点的前面部分，next是现在这个节点的后面部分，root是正在修改其next的部分





1. **删除链表倒数n个节点**

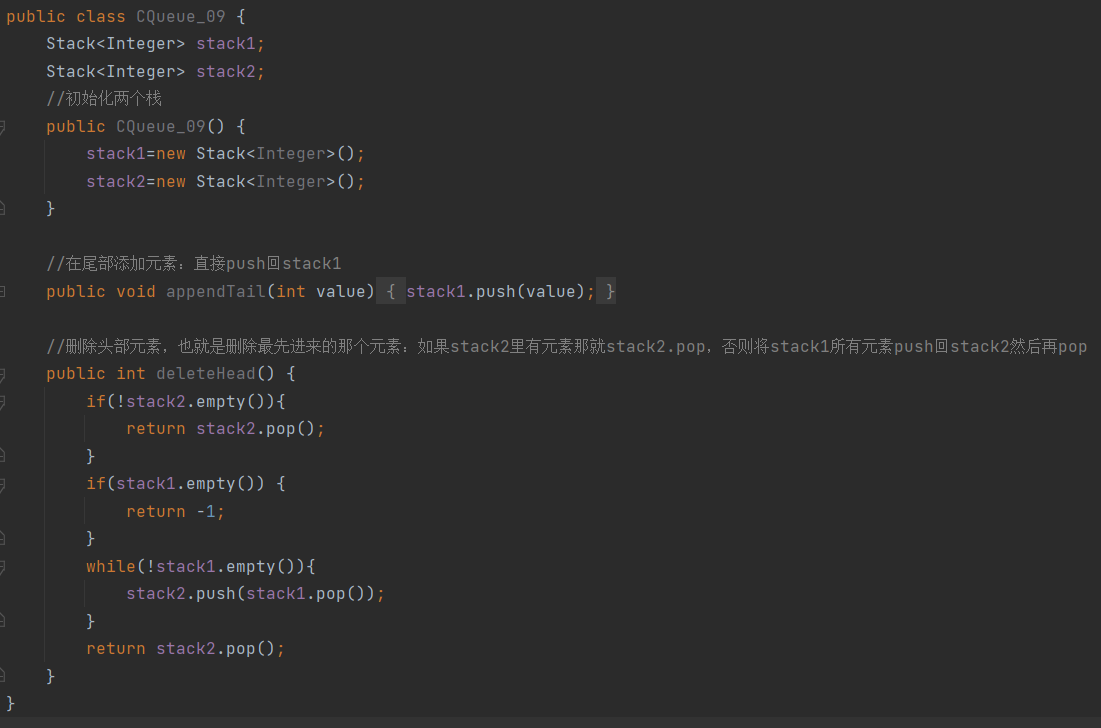
和第10题一样用双指针法找到这个倒数第n个节点，然后把p1设置为null即可。

【如果是要删除倒数第n个节点，则是p1和p2第一次移动n+1格，则第二次移动的时候p1到达的是倒数第n+1个节点，然后把它后面的节点删除即可：p1.next=p1.next.next】

1. **两个栈模拟队列**

添加元素的时候一律push回stack1，

删除头部元素的时候如果stack2里有元素就stack2.pop，否则将stack1的元素全部push回stack2再pop



1. **两个队列模拟栈**

和13题有所不同

放元素的时候，放入为空的那个队列。

删除元素的时候，看哪个队列不为空，在不为空的那个队列里进行如下操作：

一个一个拿元素出来放到另一个队列，直到拿到最后一个元素就返回

因此，两个队列是交替着成为空队列，每次pop都会导致一个队列成为空队列。

1. **操作系统死锁产生的条件**

四个必要条件：

* 互斥条件：一个资源每次只能被一个进程使用。
* 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
* 不剥夺条件：进程已获得的资源，在末使用完之前，不能强行剥夺，只能在使用完后自己释放。
* 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。B等A，C等B……

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之

一不满足，就不会发生死锁。

1. **死锁恢复，原作者漏了进程回退**

打破死锁的两种办法：

* 进程终止：简单地终止一个或多个进程以打破循环等待。

两种终止方法：

* + 终止所有死锁进程
  + 一次只终止一个死锁进程，直到死锁循环被解开
* 资源抢占：从一个或多个死锁进程那里抢占资源。

需要考虑三个问题：

* + 从哪些进程抢占资源，要选择代价最小的。代价因素包括许多参数如死锁进程所拥有的资源数量，死锁进程到现在为止在其执行过程中所消耗的时间等等。
  + 回滚。某进程的资源被抢占之后，该进程就不能正常执行了，要将进程回滚到某个安全状态。要确定该安全状态并不容易——最简单的是完全回滚，即终止进程并重新执行，更为有效的方法是只将进程回滚到足够打破死锁。
  + 饥饿。为了保证资源不会总是从同一进程中被抢占，不能仅以代价因素作为选择依据，在代价因素中加上回滚次数，回滚的越多则越不可能继续被作为牺牲品。

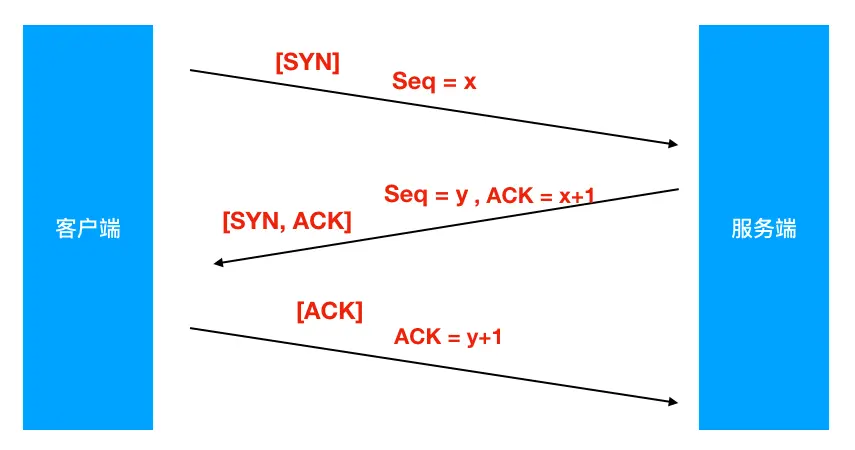
1. **死锁避免和预防**

* 针对互斥条件：互斥条件一般不可能禁止，某些资源必须同一时间只能由一个进程操作才能保证其安全性。
* 针对请求与保持条件：可以要求进程一次性请求所有资源，如果无法一次性请求那么就进行等待。但是这样做存在的问题有：
  + 一个进程可能会被阻塞很长时间，以等待可以一次性获取到所有资源的时机。
  + 从实际角度出发，进程在持有一部分资源的时候就可以继续正常运行，其次某个资源该进程只会持有一小部分时间，大部分时间都不会使用该资源。
  + 进程可能无法预知它将来需要的资源
* 针对不剥夺条件：当占有某个资源的进程在进一步尝试获取其它资源时被拒绝，那么该进程必须释放自己占有的资源，在必要时可以尝试重新获取这个被释放的资源；或者当一个进程请求的资源被其他资源占有时，操作系统可以抢占这个持有资源的进程，要求它释放资源。
* 针对循环等待条件：可以通过定义资源获取的访问顺序——若定义资源获取序列{ A , B , C , D } ，当获取到资源 B 时，只能够尝试获取资源 C 或者 D 。当然这个预防方式可能是低效的，可能会在没有必要的情况下拒绝资源的获取。

1. 最多能产生多少个tcp链接，回答了跟tcp报文端口大小有关，但是具体的数值不太记得了，面试官说跟tcp报文端口大小无关？事后看了下，还跟内存大小有关。
2. 服务端怎么标志一个tcp链接，应该是根据ip和端口，我说是创建socket返回的id(因为前面在说服务端套接字什么的，下意识就往这方面想了)
3. **9个小球，找出重量不一样的**

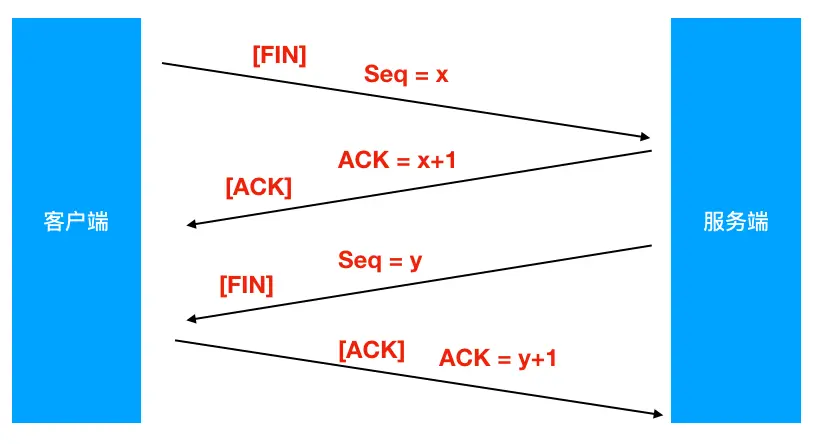
原作者回答错了，说了四次，实际上三次就够，先分成三份，第一次放两份，如果平衡就知道有问题的球在剩下一份，拿那一份测一下就知道是重了还是轻了，再从这三个球中拿两个出来，就能知道结果

1. 字节序和内存对齐
2. 智能指针
3. **TCP三次握手、四次挥手**
4. 三次握手：

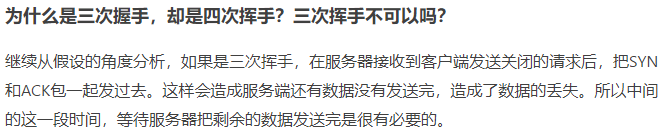


* 第一次：客户端发送SYN位为1，序列号为x的报文，表示请求连接
* 第二次：服务端回复SYN和ACK位为1，序列号为y，确认号x+1，表示同意连接
* 第三次，客户端回复ACK为为1，确认号y+1，表示客户端收到了服务端的同意信息。

1. 四次挥手：



* 第一次：客户端发送FIN位为1，序列号为x的报文，表示请求断开连接
* 第二次：服务端回复ACK为1，确认号为x+1，表示同意断开
* 第三次：服务端发完之前的最后一个报文，FIN为1，序列号为y，告诉客户端准备关闭了，发送完成后，服务端进入LAST\_ACK状态，等待客户端确认。
* 第四次：客户端接收到服务端的关闭请求，再发送ACK标记的确认包，进入TIME\_WAIT状态，等待服务端可能请求重传的ACK包。服务端接收到ACK包后，关闭连接，进入CLOSED状态。客户端在等待固定时间(两个最大段生命周期)后，没有接收到服务的ACK包，认为服务器已关闭连接，自己也关闭连接，进入CLOSED状态。



（多了的那一次握手是要等到服务端发送完最后一个数据报文，两者才能关闭连接）