看了阿里的面经，我觉得我是个废物

Java3y 今天

以下文章来源于程序员乔戈里 ，作者乔戈里直系学弟

程序员乔戈里

乔戈里是BAT大厂后端工程师，技术栈有Java、C++、PHP、Python，专注分享自己的...

本文公众号来源：程序员乔戈里

作者：乔戈里直系学弟

本文已收录至我的GitHub

整个面试分三块进行：

1. 基础知识，主要问了计网和数据库以及算法

- TCP 的三次握手和四次挥手过程

> 腾讯的面试官大大也问了这个问题，并且特别提醒了我要注意记住 tcp 连接和断开时客户端和服务器端的状态，真是超级感谢啊，这点原来还真的一直没有注意过。
> 首先我们来回顾一下 TCP 的数据传输单元，TCP 传送的数据单元称为报文段。一个 TCP 报文段分为 TCP 首部和 TCP 数据两部分，整个 TCP 报文段都封装在 IP 数据报中的数据部分，TCP 首部长度是4的整数倍，其中有固定的20个字节，剩余的可变动的就是选项和填充
> 「最常见的可选字段是**最长报文大**小，又称为MSS（Maximum Segment Size），每个连接

方通常都在通信的第一个报文段（为建立连接而设置SYN标志为1的那个段）中指明这个选项，它**表示本端所能接受的最大报文段的长度**。」，20个固定的字节包括了源端口号（2 字节）、目的端口（2字节）、seq序列号（4字节）、确认号ack（4字节）、以及确认位ACK等等。

其次，我们来详细讲解一下三次握手、四次挥手的过程：

- 三次握手

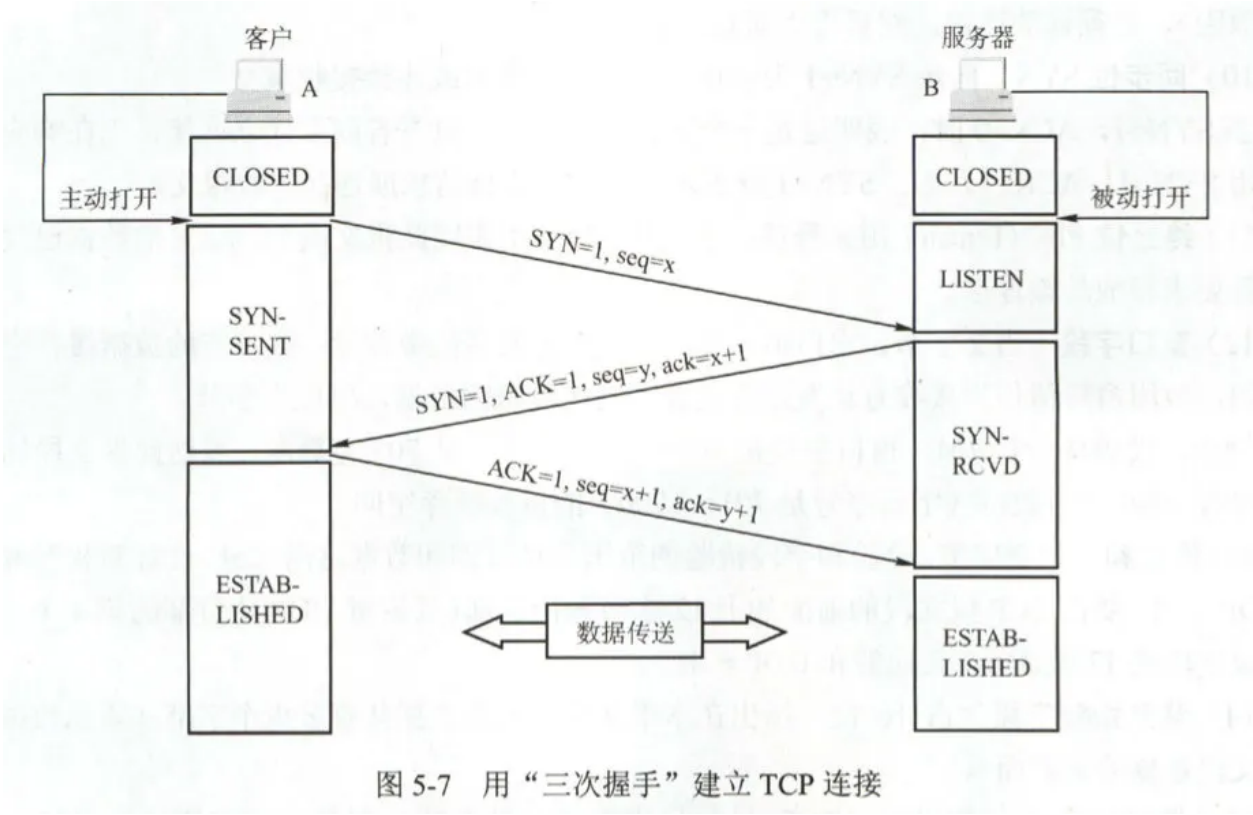

图 5-7 用“三次握手”建立 TCP 连接

首先，在三次握手建立连接的阶段，是不会传输 TCP 报文段的，传输的是 传输控制块（TCB）， 传输控制块 TCB(Transmission Control Block)存储了每一个连接中的一些重要信息，如：TCP 连接表，指向发送和接收缓存的指针，指向重传队列的指针，当前的发送和接收序号等等。

1. 最开始的 Client 和 Server 都是处于 Closed，由于服务器端不知道要跟谁建立连接，所以其只能被动打开，然后监听端口，此时 Server 处于 Listen 状态；
2. 而 Client 会主动打开，然后构建好 TCB 「SYN= 1，seq = x」，发送给服务器端，此时 Client 会将状态置为 SYN_SEND 「同步已发送」；
3. 服务器端收到客户端发来的同步请求后，会将状态置为 SYN_RECV「同步已接收」，同时会构建好 TCB「SYN = 1，seq = y，ACK = 1，ack = x + 1」发送给客户端；
4. 客户端接收到了服务器端发来的传输控制块之后，会将自己的状态改为 ESTABLISHED「建立连接」，然后发送确认报文（ACK= 1，seq = x + 1，ack = y + 1）；
5. 服务器端在收到了客户端发来的报文之后，也将状态置为 ESTABLISHED「建立连接」，至此，三次握手结束，当然在这里，可以带 tcp 报文段信息过来了，因为此时客户端已经可以保证是可靠的传输了，所以在这一端可以发送报文段了。

几个问题：

1. **为何不直接在第一次握手就带上报文段消息，非要第三次才可以带？**

因为 TCP 是要保证数据的不丢失且可靠，如果在第一次就带上报文段消息，此次建立连接很有可能就会失败，那么就不能保证数据的不丢失了，在不可靠的机制上进行这种操作，换来的代价太大，每次发送报文段的资源也会增大，得不偿失；
而第三次握手的时候，客户端已经知道服务器端准备好了，所以只要告诉服务器端自己准备好了就okay了，所以此时带上报文段信息没有任何问题。

2. **可不可以只握手两次？**

肯定是不可以的，三次握手主要是解决这样一个常见的问题，客户端发送了第一个请求连接并且没有丢失，只是因为在网络结点中滞留的时间太长了，由于TCP的客户端迟迟没有收到确认报文，以为服务器没有收到，此时重新向服务器发送这条报文，此后客户端和服务器经过两次握手完成连接，传输数据，然后关闭连接。此时此前滞留的那一次请求连接，网络通畅了到达了服务器，这个报文本该是失效的，但是，两次握手的机制将会让客户端和服务器再次建立连接，这将导致不必要的错误和资源的浪费。
如果采用的是三次握手，就算是那一次失效的报文传送过来了，服务端接受到了那条失效报文并且回复了确认报文，但是客户端不会再次发出确认。由于服务器收不到确认，就知道客户端并没有请求连接。———————————————————— 版权声明：本文为CSDN博主「小书go」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。原文链接：https://blog.csdn.net/qzcsu/article/details/72861891

- 四次挥手

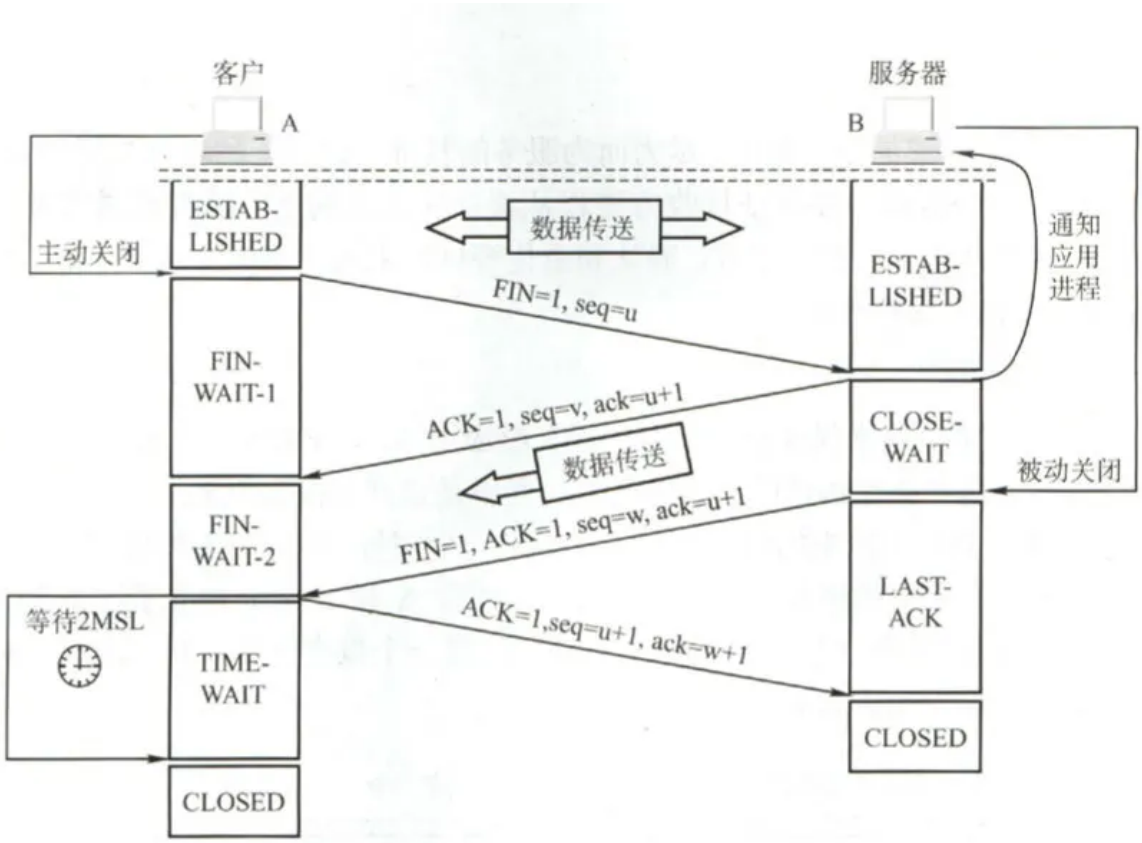

图 5-8 用“四次握手”释放 TCP 连接

1. 最开始客户端和服务器端都是 ESTABLISHED 的状态，然后客户端会主动关闭，而服务器端则是被动关闭。
2. 客户端发送一个 FIN 报文段，seq = 结束的报文段序号 + 1「假设为 u」，告诉服务器端，客户端需要关闭了，此时将客户端的状态变为 FIN-WAIT-1，等待服务器端的反馈；
3. 服务器在接收到了客户端发来的 FIN 包之后，会发一条 ack报文反馈给客户端，其中报文中包括 ACK = 1，seq = v，ack = u+1，告诉客户端收到了客户端要关闭的消息了，同时服务器端会通知应用进程需要关闭连接了，并将自己的状态置为 CLOSE-WAIT；
4. 由于服务器端可能还有一些数据没处理完，所以需要一段时间的等待，当处理完了之后，会再发一条报文，其中 FIN = 1，ACK = 1，seq = w，ack = u+1，告知客户端，服务器端现在可以关闭了，并将服务器端的状态由 CLOSE-WAIT 变为 LAST-ACK；
5. 客户端在收到了服务器端发来的消息之后，会发一条ack报文「ACK = 1，seq = u+1，ack = w+1」回去，告知服务器端，客户端已经知道了你准备好关闭了，此时会将客户端的状态由 FIN-WAIT-2 置为 TIME-WAIT，在两个最长报文段传输时间过后，会自动将客户端的状态由 TIME-WAIT 置为 CLOSED。
6. 服务器端收到消息之后，就将状态由 LAST-ACK 置为了 CLOSED，自此，四次挥手全部结束。

一个很常见的问题，为何不能三次挥手呢?

- 首先如果去掉最后一次挥手，那么服务器端就不知道自己要关闭的报文有没有传输成功，可能半路上就失败了，但是此时客户端不知道，导致客户端一直在等待服务器关闭，但是此时服务器端直接就关闭了；
- 如果中间的两次挥手合并，那是肯定不行的，因为此时服务器端可能还有很多报文未处理完，此时直接关闭肯定会对传输有很大影响。

为什么客户端在收到 服务器端发来的 FIN 包后要等 2 个最长报文段传输时间?
防止最后自己发去的 ack 没传送到服务器，如果服务器没收到客户端的 ack，肯定会选择重发一次 FIN 包，那么此时如果客户端已经关闭了，客户端就不能再发 ack 确认收到了，至于为何是 2 个报文段传输时间，因为刚好一去一回嘛... 2 个最长报文传输时间没有 FIN 包发来，就说明服务器已经关闭了，客户端也就可以安心关闭了。

这文章整挺好：https://blog.csdn.net/qzcsu/article/details/72861891

- http 和 tcp 的区别

> tcp 是传输层协议，http是应用层协议，http在传输层就是使用的 tcp。

- 排序算法有哪些

> 这个简单。排序算法分为比较算法和非比较算法，其中比较算法包括交换排序「冒泡和快排」、选择排序「简单选择排序和堆排序」、插入排序「直接插入排序、希尔排序」、归并排序「二路归并和多路归并」，非比较排序有计数排序、桶排序、基数排序。「公式：不稳定的有：快些选堆」
>
> - 冒泡排序。稳定的，平均时间复杂度为 $O(n^2)$，最好时间复杂度那肯定就是一次循环 $O(n)$，最坏时间复杂度为 $O(n^2)$。空间复杂度 $O(1)$。
> - 快速排序。不稳定，平均时间复杂度为$O(nlogn)$，最好的时间复杂度为$O(nlogn)$，最坏就是选定的基准值在最边上，这样就是$O(n^2)$，注意哦，快排的空间复杂度平均是 $O(logn)$，最差 $O(n)$。
> - 简单选择排序。不稳定，平均、最好、最坏时间复杂度都为$O(n^2)$。空间复杂度 $O(1)$。
> - 堆排序。不稳定，平均、最好、最坏的时间复杂度为$O(nlogn)$。空间复杂度 $O(1)$。
> - 直接插入排序。稳定。最好$O(n)$，平均、最坏时间复杂度$O(n^2)$。空间复杂度 $O(1)$。
> - 希尔排序。不稳定。最好$O(n)$，平均$O(n1.3)$，最坏肯定是$O(n^2)$。空间复杂度$O(1)$。
> - 归并排序。稳定。最好、最坏、最差时间复杂度$O(nlogn)$，空间复杂度$O(n)$。
> - 计数排序。稳定，空间换时间。适合数比较集中在一起的，这样k就少了，时间复杂度为$O(n+k)$，空间复杂度也为$O(n+k)$。「个人还是觉得其实空间复杂度为$O(k)$，因为我可以把值放回去的时候可以放到原数组上，所以是$O(k)$。」
> - 桶排序，桶越多，时间复杂度很简单，为$O(n+k)$，空间复杂度最坏为$O(n+k)$，其中 n 是因为桶内部所有元素得排序， k 是指桶的数量。
> - 基数排序，时间复杂度$O(n*k)$，k为最大数的位数，空间复杂度为$O(n)$。

- 堆排序的稳定性，如何实现堆排序，具体细节

> 这个很简单，就不详细说了。

- 归并排序的稳定性，如何实现归并排序，具体细节

> 简单。

- 说一下jdk自带的排序用到了哪些排序算法，展开讲一下

> 1. Arrays.sort() & Collections.sort()
> 2. JDK中的自带的排序算法实现原理精彩总结

> jdk层面实现的sort总共是两类，一个是 Arrays.sort()， Collections.sort()；
>
> 1. Arrays.sort()
>
> a. 如果数组内元素是基本数据类型，最主要采用的是双轴快速排序「其实就是三路快排一模一样的思路，只不过三路快排中间是 = pivot1，而双轴快速排序是（pivot1，pivot2），具体戳链接：https://www.cnblogs.com/nullzx/p/5880191.html 。
> 总结就是数组长度小于47的时候是用直接插入算法，大于47并且小于286是采用双轴快速排序，大于286如果连续性好「也就是元素大多有序，有一个flag专门用来记录数组元素的升降次数，代表这个数组的连续性」采用的是归并排序，否则还是依旧采用双轴快速排序。
>
> b. 如果数组内元素是对象，采用的是TimSort.sort()，跟 Collections.sort()一样，都是采用的这个函数，这是归并排序算法和插入排序的结合。
>
> 2. Collections.sort()，采用 TimSort.sort()。
>
> TimSort.sort() 大概原理：
>
> 1. 当待排序元素**小于32个**时，采用**二分插入排序**，是插入排序的一种改进，可以减少插入排序比较的次数。当找到插入位置时，直接利用System.copy()函数即可。
> 2. 当待排序元素大于等于32个时，进行归并排序（和传统的归并不一样），首先将排序元素分区，每个分区的大小区间为[16,32)，然后依次对每个分区进行排序（具体实现依然是二分插入排序），排完序的分区压入栈（准确的说不是栈，而是一个数组，用来记录排序好的分区），当栈内的分区数满足条件时，进行分区合并，合并为一个更大的分区，当栈中只剩一个分区时，排序完成。

- mysql如何优化

> 建索引

- 索引的建立的原则有哪些

> 除了运用最左前缀、索引下推、考虑索引长度，还有哪些是建立索引需要考虑的 「这个实在想不到了」

- 红黑树和平衡二叉树的区别，各自的优势特点，以及红黑树如何进行添加数据「具体说一下旋转过程，我只说了我博客上写了，具体的给忘了...」

> 这个二面真得复习复习。

2. java方面

- 讲一下双亲委派模型，为什么要设计双亲委派模型

> 双亲委派模型：一个类加载器在加载类时，先把这个请求委托给自己的父类加载器去执行，如果父类加载器还存在父类加载器，就继续向上委托，直到顶层的启动类加载器。如果父类加载器能够完成类加载，就成功返回，如果父类加载器无法完成加载，那么子加载器才会尝试自己去加载。
> 好处：**java类随着它的类加载器一起具备了一种带有优先级的层次关系**。
> 这种双亲委派模式的设计原因：可以避免类的重复加载，另外也避免了java的核心API被篡改。

- 违反双亲委派模型，我不小心说了jdbc，然后问我jdbc是如何连接到数据库的，具体流程是什么，我就说了个反射...没复习到位

> 《从双亲委派模型到jdbc》
> https://jeromememory.github.io/2020/03/19/%E4%BB%8E%E5%8F%8C%E4%BA%B2%E5%A7%94%E6%B4%BE%E6%A8%A1%E5%9E%8B%E5%88%B0%20jdbc.html

- 讲一下jmm，为何这样设计

> java memeory model ，java 内存模型，设计的目的是屏蔽掉各种硬件和操作系统之间的差异性，实现让 Java 在各种平台下都能能到一致的并发效果。jmm 中，分为主内存和工作内存，其中每个线程拥有自己的工作内存，而主内存是所有线程共享的。这里我遇到疑问了，在周志华老师那本书中，先是讲主内存中存储了所有的变量，那必然就包括了线程的局部变量，那难道我线程使用自己的局部变量，也要从主内存中拷贝一份副本到工作内存中呢？这是不是和说主内存是共享区域产生了矛盾呢？书中还说可以类比，主内存就是跟堆差不多，而工作内存类似于栈，那之前说的主内存存储了所有的变量，这句话是不是有问题呢？个人觉得主内存不可能存储所有的变量...应该就是类似于堆存储共享变量...

- 为何要有工作内存，有了主内存和工作内存不是更麻烦啊，要不断的复制移动数据，为何不能直接对主内存操作「这个也没答上来」

> 这就跟为何要提出寄存器和缓存一样的道理，如果所有的操作都在内存中完成，那速度实在是太慢了，只有工作在寄存器和缓存中，速度才能让人满意，而这里的主内存就类比为内存，工作内存就类比为寄存器和缓存。

- 什么是线程安全 「多线程方面一个问题都没问...血亏」

> 多个线程访问一个对象，无需考虑环境和额外同步，调用这个对象的行为就能得到正确的答案，就说明这个对象是线程安全的。

- 举三个例子分别描述 jmm的三个特性「原子性、有序性、可见性」导致的线程安全问题

> 不遵循原子性：volatile 变量的自加，复合操作，导致线程不安全；
> 不遵循有序性：比如共享变量，多个线程同时访问，不按序，每个都拷贝一份到自己的工作内存，必然会导致线程不安全；
> 不遵循可见性：普通变量，跟有序性一样的例子，每个都从主内存拷贝一份变量的副本到工作内存，必然会导致线程不安全。

- 讲一下 RunTimeException 的造成的原因「非检查型异常」，并说一下为什么不处理 RunTimeException?

> RuntimeException是Exception子类。而Exception还有其它类型的异常，我们统一称为非Runtime异常。RuntimeException的特点是非检查型异常，也就是Java系统中**允许可以不被**

> **catch，在运行时抛出**。而其它定非运行时异常如果抛出的话必须显示的catch，否则编译不过。
> RuntimeException常见异常：
> 1 NullPointerException，空指针异常。
> 2 NumberFormatException，字符串转化成数字时。
> 3 ArrayIndexOutOfBoundsException， 数组越界时。
> 4 StringIndexOutOfBoundsException， 字符串越界时。
> 5 ClassCastException，类型转换时。
> 6 UnsupportedOperationException，该操作不支持，一般子类不实现父类的某些方法时。
> 7 ArithmeticException，零作为除数等。
> 8 IllegalArgumentException，表明传递了一个不合法或不正确的参数
> 运行时出现错误，说明你的代码有问题，程序已经无法继续运行，所以对RuntimeException的处理时不必要的。之所以不处理，目的就是要是程序停止，修正代码。

3. 项目

- 主要就问了下我最近在做什么项目，到什么阶段了，有多少人用；
- 问我爬虫的实现「面试官貌似没有做过这方面的东西」
- 我提了一嘴 xxl-job，面试官应该也没用过，就没有深究，本来还想讲讲kafka的，结果直接没问...白准备了

最后就问了一个很常见的算法问题：

256M 的内存如何对 16g的数组进行排序

> 多路归并，因为没要求存储，只要求了内存，可以多路归并，加入每个元素都是 1M，则可以最多分成 256 组，然后进行归并。
> 具体描述：采用外部排序，先将16 g数组分成 256 M 一组，然后分别读入内存进行内部排序「比如说可以使用快排」，将这些组内元素全部排好序之后，然后运用败者树和置换-选择排序，进行多路归并，即可。

这里其实可以引申出好多问题：

1. **海量数据 求最大的 K个数问题，如何解决?**

> - **按位划分区域，可以尽快的缩小范围，比如最高位 0 分一堆，1 分成一堆而且不用排序，这是第一选择。**

- 最经典的方法当然是 堆 了，比如要求前1000个最大的数，那就直接建一个 1000 大小的小根堆，然后遍历，只要发现后面的数比小根堆的根节点大，就把根节点和该数交换，重新调整堆，遍历完之后，堆中的数自然就是最大的 1000 个数了；
- 当然能使用堆排序的前提是内存中要能够放得下这个 K，如果放不下呢？那就只能外部排序了，排序完之后拿到第 K 大的数即可，当然排序前可以和方法一搭配一下。

2. 海量数据求中位数，如何解决?

1. 可以按照位来分组，比如说最高位是0的一组，是 1 的一组，这样可以统计出那一组更少，这样就排除了一大半，然后继续这样排查，最终缩小范围后直接内部排序。
2. 直接外部排序，然后取中间值，最笨的方法。

3. 在海量数据中找出出现频率最高的前k个数，例如，在搜索引擎中，统计搜索最热门的10个查询词；在歌曲库中统计下载最高的前10首歌等。

1. 如果重复率很高，可以采用前缀树，因为 trie 树适用于数据量大，重复多，但是数据种类小必须得可以放入内存；
2. 按照 hash 进行分组，这样就能避免相同的数分到不同区域去了，导致不好统计。hash 分组完毕后，然后用前缀树 或者 hashmap 来计算每个组的前 k 个频率最高的数，最后对各个组的前 k 个数进行统计即可。

4. 给40亿个不重复的unsigned int的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中?

这里我们把40亿个数中的每一个用32位的二进制来表示 假设这40亿个数开始放在一个文件中。
然后将这40亿个数分成两类: 1.最高位为0 2.最高位为1 并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20亿，而另一个>=20亿（这相当于折半了）；与要查找的数的最高位比较并接着进入相应的文件再查找
再然后把这个文件为又分成两类: 1.次最高位为0 2.次最高位为1
并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10亿，而另一个>=10亿（这相当于折半了）；与要查找的数的次最高位比较并接着进入相应的文件再查找。.....
以此类推，就可以找到了,而且时间复杂度为O(logn)。

大概统计一下，海量数据求 TopK 的普遍方法：

- **最快的不需要排序就能排除一大堆的数据的方法就是看“位”，比如最高位为 0 的分一块，为 1 的分一块，这样迅速就分出一大块不需要的了，尤其适合找中位数，等分的差不多了就可以进行内部排序了。**
- **堆排序，适用于求海量数据最大 K or 最小的 K 个数；**
- **分治hash，适用于那些内存很小，数据很大，但是又想求最大的 K 个众数的问题，可以先 hash 到很多个组，然后在组内部使用 hashmap 或者 前缀树 「google等字符」，取到组内前 K 个众数，最后进行组间比较久okay了；**
- **当然不能忘了万能法，那就是外部排序，然后再进行相应的处理。**

最后的最后，让我们再来做个附加题：

先来看几个比较常见的例子

- 字处理软件中，需要检查一个英语单词是否拼写正确
- 在 FBI，一个嫌疑人的名字是否已经在嫌疑名单上
- 在网络爬虫里，一个网址是否被访问过
- yahoo, gmail等邮箱垃圾邮件过滤功能

这几个例子有一个共同的特点： **如何判断一个元素是否存在一个集合中?**

> 这里必须介绍一下 bitmap 这个方法了，例如我要从海量数据中找一个数**是否出现过**，就可以用位图的思路去做，如果数字是 7 ，那就在第 7 位 置 1，如果该位置已经是 1 了，那就代表出现过，不用更改。
> 如果问题变为从海量数据中找一个数**是否出现过一次**，那这个时候就得用 2 bitmap 来表示了，也就是一个数如果出现一次，置为 01 ，出现过两次，置为 10，然后再出现，都是 10，这个时候如果我们只用一位，是不能表示出出现的次数的。
> 至于我们常说的布隆过滤器，其实也就是在bitmap之前进行一个hash，例如将字符串进行hash成数组，然后使用位图，解决这类问题。

☕ 各类知识点总结

下面的文章都有对应的**原创精美**PDF，在持续更新中，可以来找我催更~

- 142页的Spring
- 92页的Mybatis
- 129页的多线程

- 141页的Servlet
- 158页的JSP
- 76页的集合
- 64页的JDBC
- 105页的数据结构和算法

扫码或者微信搜**Java3y** 免费领取**原创**思维导图、精美PDF。在公众号回复「**888**」领取，**PDF内容纯手打**有任何不懂欢迎来问我。

原创电子书

原创思维导图