海量数据处理：十道面试题与十个海量数据处理方法总结

作者：July、youwang、yanxionglu。

时间：二零一一年三月二十六日
本文之总结：教你如何迅速秒杀掉：99%的海量数据处理面试题。有任何问题，欢迎随时交流、指正。

出处：**http://blog.csdn.net/v_JULY_v**。

第一部分、十道海量数据处理面试题

1、海量日志数据，提取出某日访问百度次数最多的那个 IP。

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，比如模 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 进行频率统计，然后再找出频率最大的几个）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。
或者如下阐述（雪域之鹰）：

算法思想：分而治之+Hash

1.IP 地址最多有 2^32=4G 种取值情况，所以不能完全加载到内存中处理；

2.可以考虑采用“分而治之”的思想，按照 IP 地址的 Hash(IP)%1024 值，把海量 IP 日志分别存储到 1024 个小文件中。这样，每个小文件最多包含 4MB 个 IP 地址；

3.对于每一个小文件，可以构建一个 IP 为 key，出现次数为 value 的 Hash map，同时记录当前出现次数最多的那个 IP 地址；

4.可以得到 1024 个小文件中的出现次数最多的 IP，再依据常规的排序算法得到总体上出现次数最多的 IP；

2、搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。

假设目前有一千万个记录（这些查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。），请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

典型的 Top K 算法，还是在这篇文章里头有所阐述，详情请参见：十一、从头到尾彻底解析 Hash 表算法。

文中，给出的最终算法是：

第一步、先对这批海量数据预处理，在 O（N）的时间内用 Hash 表完成统计（之前写成了排序，特此订正。July、2011.04.27）；

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）。ok，更多，详情，请参考原文。

或者：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

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。

方案：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。

对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 个词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

还是典型的 TOP K 算法，解决方案如下：

方案 1：

顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。

找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排

序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。

对这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

5、 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 5G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

遍历文件 a，对每个 url 求取 hash(url)%1000，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 a0,a1,...,a999）中。这样每个小文件的大约为 300M。

遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件（记为 b0,b1,...,b999）。这样处理后，所有可能相同的 url 都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bloom filter 日后会在本 BLOG 内详细阐述。

6、在 2.5 亿个整数中找出不重复的整数，注，内存不足以容纳这 2.5 亿个整数。

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32 * 2 bit=1 GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01 变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

7、腾讯面试题：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？

与上第 6 题类似，我的第一反应时快速排序+二分查找。以下是其它更好的方法：

方案 1：oo，申请 512M 的内存，一个 bit 位代表一个 unsigned int 值。读入

40 亿个数，设置相应的 bit 位，读入要查询的数，查看相应 bit 位是否为 1，为 1 表示存在，为 0 表示不存在。

dizengrong：

方案 2：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：
又因为 2^32 为 40 亿多，所以给定一个数可能在，也可能不在其中；

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)，方案 2 完。

附：这里，再简单介绍下，位图方法：

使用位图法判断整形数组是否存在重复

判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效率还能提高一倍。

欢迎，有更好的思路，或方法，共同交流。

8、怎么在海量数据中找出重复次数最多的一个？

方案 1：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）。

9、上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：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。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

10、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：这题是考虑时间效率。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）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一个。

附、100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

致谢：http://www.cnblogs.com/youwang/。

第二部分、十个海量数据处理方法大总结

ok，看了上面这么多的面试题，是否有点头晕。是的，需要一个总结。接下来，本文将简单总结下一些处理海量数据问题的常见方法，**而日后，本 BLOG 内会具体阐述这些方法。**

下面的方法全部来自 http://hi.baidu.com/yanxionglu/blog/博客，对海量数据的处理方法进行了一个一般性的总结，当然这些方法可能并不能完全覆盖所有的问题，但是这样的一些方法也基本可以处理绝大多数遇到的问题。下面的一些问题基本直接来源于公司的面试笔试题目，方法不一定最优，如果你有更好的处理方法，欢迎讨论。

一、Bloom filter

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集

基本原理及要点：

对于原理来说很简单，位数组+k 个独立 hash 函数。将 hash 函数对应的值的位数组置 1，查找时如果发现所有 hash 函数对应位都是 1 说明存在，很明显这个过程并不保证查找的结果是 100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个 counter 数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数 n，确定位数组 m 的大小及 hash 函数个数。当 hash 函数个数 k=(ln2)*(m/n)时错误率最小。在错误率不大于 E 的情况下，m 至少要等于 n*lg(1/E)才能表示任意 n 个元素的集合。但 m 还应该更大些，因为还要保证 bit 数组里至少一半为 0，则 m 应该>=nlg(1/E)*lge 大概就是 nlg(1/E)1.44 倍(lg 表示以 2 为底的对数)。

举个例子我们假设错误率为 0.01，则此时 m 应大概是 n 的 13 倍。这样 k 大概是 8 个。

注意这里 m 与 n 的单位不同，m 是 bit 为单位，而 n 则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多 bit 的。所以使用 bloom filter 内存上通常都是节省的。

扩展：

Bloom filter 将集合中的元素映射到位数组中，用 k（k 为哈希函数个数）个映射位是否全 1 表示元素在不在这个集合中。Counting bloom filter（CBF）将位数组中的每一位扩展为一个 counter，从而支持了元素的删除操作。Spectral Bloom Filter（SBF）将其与集合元素的出现次数关联。SBF 采用 counter 中的最小值来近似表示元素的出现频率。

问题实例：给你 A,B 两个文件，各存放 50 亿条 URL，每条 URL 占用 64 字节，内存限制是 4G，让你找出 A,B 文件共同的 URL。如果是三个乃至 n 个文件呢？

根据这个问题我们来计算下内存的占用，4G=2^32 大概是 40 亿*8 大概是 340 亿，n=50 亿，如果按出错率 0.01 算需要的大概是 650 亿个 bit。现在可用的是 340 亿，相差并不多，这样可能会使出错率上升些。另外如果这些 urlip 是一一对应的，就可以转换成 ip，则大大简单了。

二、Hashing

适用范围：快速查找，删除的基本数据结构，通常需要总数据量可以放入内存

基本原理及要点：

hash 函数选择，针对字符串，整数，排列，具体相应的 hash 方法。

碰撞处理，一种是 open hashing，也称为拉链法；另一种就是 closed hashing，也称开地址法，opened addressing。

扩展：

d-left hashing 中的 d 是多个的意思，我们先简化这个问题，看一看 2-left hashing。2-left hashing 指的是将一个哈希表分成长度相等的两半，分别叫做 T1 和 T2，给 T1 和 T2 分别配备一个哈希函数，h1 和 h2。在存储一个新的 key 时，同时用两个哈希函数进行计算，得出两个地址 h1[key]和 h2[key]。这时需要检查

T1 中的 h1[key]位置和 T2 中的 h2[key]位置，哪一个位置已经存储的（有碰撞的）key 比较多，然后将新 key 存储在负载少的位置。如果两边一样多，比如两个位置都为空或者都存储了一个 key，就把新 key 存储在左边的 T1 子表中，2-left 也由此而来。在查找一个 key 时，必须进行两次 hash，同时查找两个位置。

问题实例：

1).海量日志数据，提取出某日访问百度次数最多的那个 IP。

IP 的数目还是有限的，最多 2^32 个，所以可以考虑使用 hash 将 ip 直接存入内存，然后进行统计。

三、bit-map

适用范围：可进行数据的快速查找，判重，删除，一般来说数据范围是 int 的 10 倍以下

基本原理及要点：使用 bit 数组来表示某些元素是否存在，比如 8 位电话号码

扩展：bloom filter 可以看做是对 bit-map 的扩展

问题实例：

1)已知某个文件内包含一些电话号码，每个号码为 8 位数字，统计不同号码的个数。

8 位最多 99 999 999，大概需要 99m 个 bit，大概 10 几 m 字节的内存即可。

2)2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

将 bit-map 扩展一下，用 2bit 表示一个数即可，0 表示未出现，1 表示出现一次，2 表示出现 2 次及以上。或者我们不用 2bit 来进行表示，我们用两个 bit-map 即可模拟实现这个 2bit-map。

四、堆

适用范围：海量数据前 n 大，并且 n 比较小，堆可以放入内存

基本原理及要点：最大堆求前 n 小，最小堆求前 n 大。方法，比如求前 n 小，我们比较当前元素与最大堆里的最大元素，如果它小于最大元素，则应该替换那个最大元素。这样最后得到的 n 个元素就是最小的 n 个。适合大数据量，求前 n 小，n 的大小比较小的情况，这样可以扫描一遍即可得到所有的前 n 元素，效率很高。

扩展：双堆，一个最大堆与一个最小堆结合，可以用来维护中位数。

问题实例：

1)100w 个数中找最大的前 100 个数。

用一个 100 个元素大小的最小堆即可。

五、双层桶划分----其实本质上就是【分而治之】的思想，重在“分”的技巧上！

适用范围：第 k 大，中位数，不重复或重复的数字

基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行。可以通过多次缩小，双层只是一个例子。

扩展：

问题实例：

1).2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

有点像鸽巢原理，整数个数为 2^32,也就是，我们可以将这 2^32 个数，划分为 2^8 个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区域，然后不同的区域在利用 bitmap 就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。

2).5 亿个 int 找它们的中位数。

这个例子比上面那个更明显。首先我们将 int 划分为 2^16 个区域，然后读取

数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

实际上，如果不是 int 是 int64，我们可以经过 3 次这样的划分即可降低到可以接受的程度。即可以先将 int64 分成 2^24 个区域，然后确定区域的第几大数，在将该区域分成 2^20 个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有 2^20，就可以直接利用 direct addr table 进行统计了。

六、数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据的设计实现方法，对海量数据的增删改查进行处理。

七、倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询

基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。

以英文为例，下面是要被索引的文本：

T0 = "it is what it is"

T1 = "what is it"

T2 = "it is a banana"

我们就能得到下面的反向文件索引：

"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

检索的条件"what","is"和"it"将对应集合的交集。

正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。

扩展：

问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。

八、外排序

适用范围：大数据的排序，去重

基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树

扩展：

问题实例：

1).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 个字节，内存限制大小是 1M。返回频数最高的 100 个词。

这个数据具有很明显的特点，词的大小为 16 个字节，但是内存只有 1m 做 hash 有些不够，所以可以用来排序。内存可以当输入缓冲区使用。

九、trie 树

适用范围：数据量大，重复多，但是数据种类小可以放入内存

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

问题实例：

1).有 10 个文件，每个文件 1G，每个文件的每一行都存放的是用户的 query，每个文件的 query 都可能重复。要你按照 query 的频度排序。

2).1000 万字符串，其中有些是相同的(重复),需要把重复的全部去掉，保留没有重复的字符串。请问怎么设计和实现？

3).寻找热门查询：查询串的重复度比较高，虽然总数是 1 千万，但如果除去重复后，不超过 3 百万个，每个不超过 255 字节。

十、分布式处理 mapreduce

适用范围：数据量大，但是数据种类小可以放入内存
基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。
扩展：
问题实例：

1).The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents:

2).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。

3).一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 O(N)个数并对它们操作。如何找到 N^2 个数的中数(median)？

经典问题分析

上千万 or 亿数据（有重复），统计其中出现次数最多的前 N 个数据,分两种情况：可一次读入内存，不可一次读入。

可用思路：trie 树+堆，数据库索引，划分子集分别统计，hash，分布式计算，近似统计，外排序

所谓的是否能一次读入内存，实际上应该指去除重复后的数据量。如果去重后数据可以放入内存，我们可以为数据建立字典，比如通过 map，hashmap，trie，然后直接进行统计即可。当然在更新每条数据的出现次数的时候，我们可以利用一个堆来维护出现次数最多的前 N 个数据，当然这样导致维护次数增加，不如完全统计后在求前 N 大效率高。

如果数据无法放入内存。一方面我们可以考虑上面的字典方法能否被改进以适应这种情形，可以做的改变就是将字典存放到硬盘上，而不是内存，这可以参考数据库的存储方法。

当然还有更好的方法，就是可以采用分布式计算，基本上就是 map-reduce 过程，首先可以根据数据值或者把数据 hash(md5)后的值，将数据按照范围划分

到不同的机子，最好可以让数据划分后可以一次读入内存，这样不同的机子负责处理各种的数值范围，实际上就是 map。得到结果后，各个机子只需拿出各自的出现次数最多的前 N 个数据，然后汇总，选出所有的数据中出现次数最多的前 N 个数据，这实际上就是 reduce 过程。

实际上可能想直接将数据均分到不同的机子上进行处理，这样是无法得到正确的解的。因为一个数据可能被均分到不同的机子上，而另一个则可能完全聚集到一个机子上，同时还可能存在具有相同数目的数据。比如我们要找出现次数最多的前 100 个，我们将 1000 万的数据分布到 10 台机器上，找到每台出现次数最多的前 100 个，归并之后这样不能保证找到真正的第 100 个，因为比如出现次数最多的第 100 个可能有 1 万个，但是它被分到了 10 台机子，这样在每台上只有 1 千个，假设这些机子排名在 1000 个之前的那些都是单独分布在一台机子上的，比如有 1001 个，这样本来具有 1 万个的这个就会被淘汰，即使我们让每台机子选出出现次数最多的 1000 个再归并，仍然会出错，因为可能存在大量个数为 1001 个的发生聚集。因此不能将数据随便均分到不同机子上，而是要根据 hash 后的值将它们映射到不同的机子上处理，让不同的机器处理一个数值范围。

而外排序的方法会消耗大量的 IO，效率不会很高。而上面的分布式方法，也可以用于单机版本，也就是将总的数据根据值的范围，划分成多个不同的子文件，然后逐个处理。处理完毕之后再对这些单词的及其出现频率进行一个归并。实际上就可以利用一个外排序的归并过程。

另外还可以考虑近似计算，也就是我们可以通过结合自然语言属性，只将那些真正实际中出现最多的那些词作为一个字典，使得这个规模可以放入内存。

ok，更多请参见本文总结：**教你如何迅速秒杀掉：99%的海量数据处理面试题**。

以上有任何问题，欢迎指正。谢谢大家。