最全大数据面试题及答案（二）

Monica · 5 个月前

1.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?

假如每个url大小为10bytes，那么可以估计每个文件的大小为50G×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，可以采用分治的思想来解决。

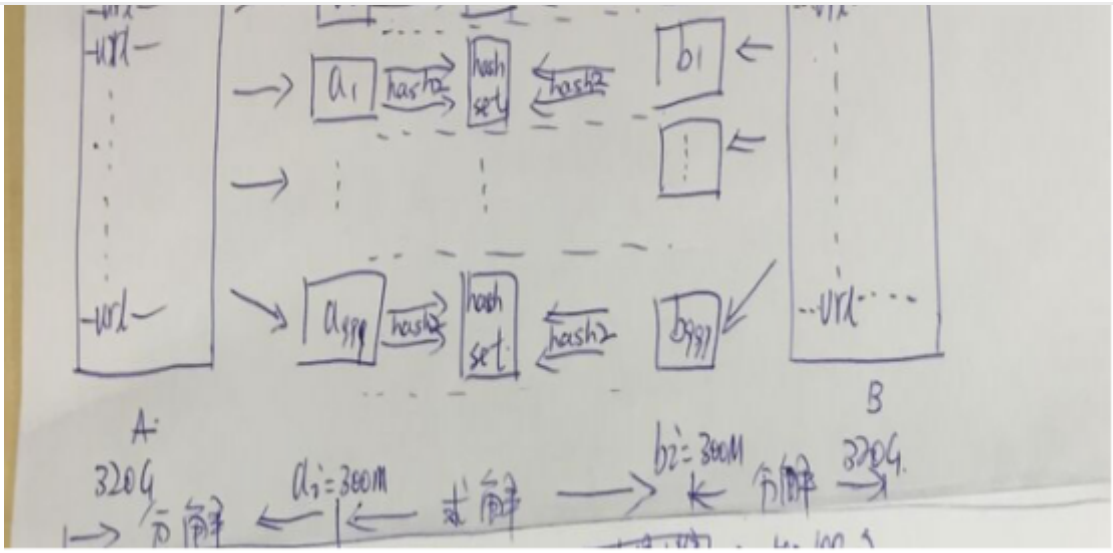
Step1：遍历文件a，对每个url求取hash(url)%1000，然后根据所取得的值将url分别存储到1000个小文件(记为a0,a1,...,a999，每个小文件约300M);

Step2:遍历文件b，采取和a相同的方式将url分别存储到1000个小文件(记为b0,b1,...,b999);

巧妙之处：这样处理后，所有可能相同的url都被保存在对应的小文件(a0vsb0,a1vsb1,...,a999vsb999)中，不对应的小文件不可能有相同的url。然后我们只要求出这个1000对小文件中相同的url即可。

Step3：求每对小文件ai和bi中相同的url时，可以把ai的url存储到hash_set/hash_map中。然后遍历bi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。

草图如下(左边分解A，右边分解B，中间求解相同url)：



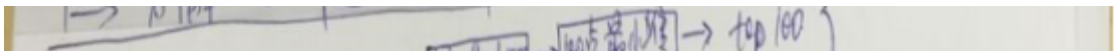
2.有一个1G大小的一个文件，里面每一行是一个词，词的大小不超过16字节，内存限制大小是1M，要求返回频数最高的100个词。

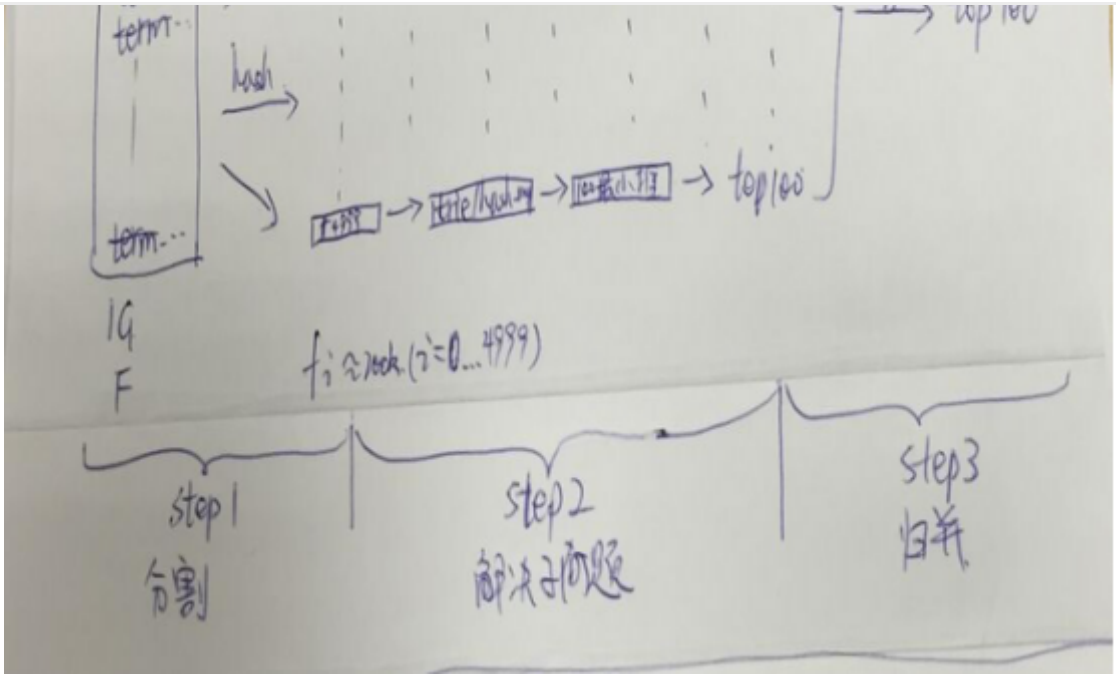
Step1：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件(记为f0,f1,...,f4999)中，这样每个文件大概是200k左右，如果其中的有的文件超过了1M大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过1M;

Step2：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词及相应的频率存入文件，这样又得到了5000个文件;

Step3：把这5000个文件进行归并(类似与归并排序);

草图如下(分割大问题，求解小问题，归并)：





3.现有海量日志数据保存在一个超级大的文件中，该文件无法直接读入内存，要求从中提取某天出访问百度次数最多的那个IP。

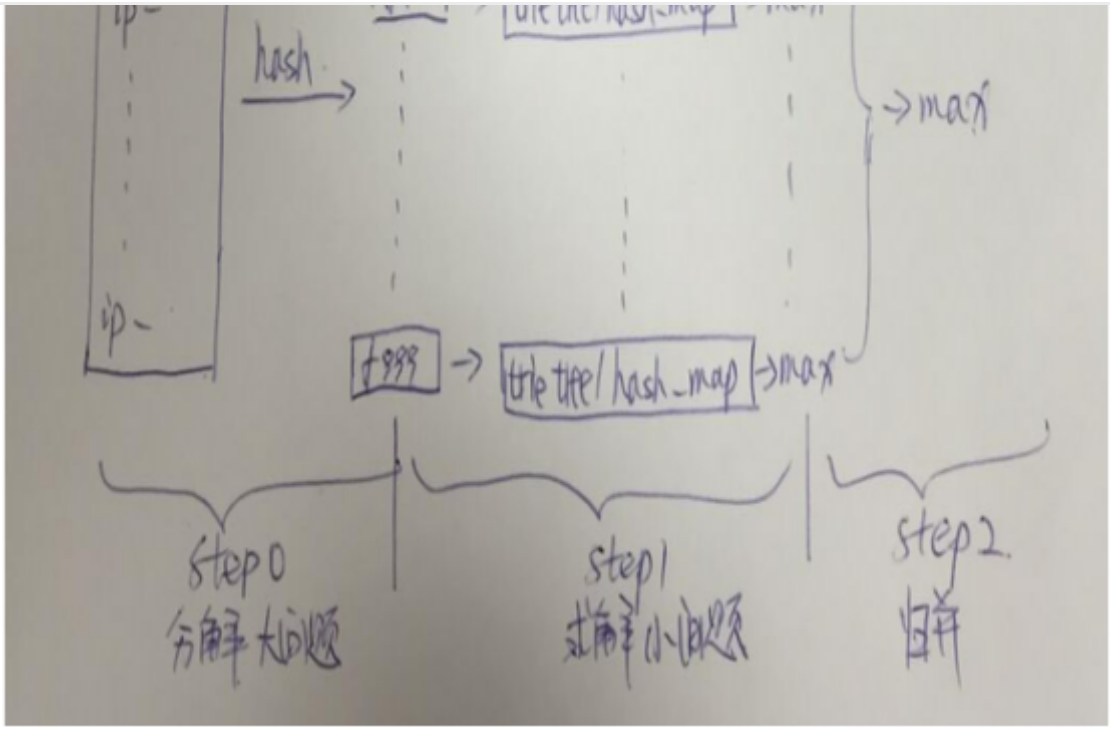
Step1：从这一天的日志数据中把访问百度的IP取出来，逐个写入到一个大文件中;

Step2：注意到IP是32位的，最多有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;

Step3：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;

Step4：在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

草图如下：



4.LVS和HAProxy相比，它的缺点是什么？

之前，的确是用LVS进行过MySQL集群的负载均衡，对HAProxy也有过了解，但是将这两者放在眼前进行比较，还真没试着了解过。面试中出现了这么一题，面试官给予的答案是LVS的配置相当繁琐，后来查找了相关资料，对这两种负载均衡方案有了更进一步的了解。LVS的负载均衡性能之强悍已经达到硬件负载均衡的F5的百分之60了，而HAproxy的负载均衡和Nginx负载均衡，均为硬件负载均衡的百分之十左右。由此可见，配置复杂，相应的效果也是显而易见的。在查找资料的过程中，试着将LVS的10种调度算法了解了一下，看似数量挺多的10种算法其实在不同的算法之间，有些只是有着一些细微的差别。在这10种调度算法中，静态调度算法有四种，动态调度算法有6种。

静态调度算法：

①RR轮询调度算法

这种调度算法不考虑服务器的状态，所以是无状态的，同时也不考虑每个服务器的性能，比如我有1-N台服务器，来N个请求了，第一个请求给第一台，第二个请求给第二台，，，第N个请求给第N台服务器，就酱紫。

②加权轮询

这种调度算法是考虑到服务器的性能的，你可以根据不同服务器的性能，加上权重进行分配

③基于目的地址的hash散列

这种调度算法和基于源地址的hash散列异曲同工，都是为了维持一个session，基于目的地址的hash散列，将记住同一请求的目的地址，将这类请求发往同一台目的服务器。简而言之，就是发往这个目的地址的请求都发往同一台服务器。而基于源地址的hash散列，就是来自同一源地址的请求都发往同一台服务器。

④基于源地址的hash散列

上述已讲，不再赘述。

动态调度

①最少连接调度算法

这种调度算法会记录响应请求的服务器上所建立的连接数，每接收到一个请求会相应的将该服务器的所建立连接数加1，同时将新来的请求分配到当前连接数最少的那台机器上。

②加权最少连接调度算法

这种调度算法在最少连接调度算法的基础上考虑到服务器的性能。当然，做这样子的考虑是有其合理性存在的，如果是同一规格的服务器，那么建立的连接数越多，必然越增加其负载，那么仅仅根据最少连接数的调度算法，必然可以实现合理的负载均衡。但如果，服务器的性能不一样呢?比如我有一台服务器，最多只能处理10个连接，现在建立了3个，还有一台服务器最多能处理1000条连接，现在建立了5个，如果单纯地按照上述的最少连接调度算法，妥妥的前者嘛，但前者已经建立了百分之三十的连接了，而后者连百分之一的连接还没有建立，试问，这合理吗?显然不合理。所以加上权重，才算合理。相应的公式也相当简单：active*256/weight。

③最短期望调度算法

这种算法，是避免出现上述加权最少连接调度算法中的一种特殊情况，导致即使加上权重，调度器也无差别对待了，举个栗子：

假设有三台服务器ABC，其当前所建立的连接数相应地为1,2,3，而权重也是1,2,3。那么如果按照加权最少连接调度算法的话，算出来是这样子的：

A:1256/1=256

B:2256/2=256

C:3256/3=256

A、B、C中任选一台，将请求发过去。

而最短期望将active256/weight的算法改进为(active+1)256/weight

那么还是之前的例子：

A:(1+1)256/1=2/1256=2256

B:(2+1)256/2=3/2256=1.5256

C:(3+1)256、3=4/3256≈1.3256

显然C

④永不排队算法

将请求发给当前连接数为0的服务器上。

⑤基于局部的最少连接调度算法

这种调度算法应用于Cache系统，维持一个请求到一台服务器的映射，其实我们仔细想想哈，之前做的一系列最少连接相关的调度算法。考虑到的是服务器的状态与性能，但是一次请求并不是单向的，就像有一个从未合作过的大牛，他很闲，你让他去解决一个之前碰到过的一个问题，未必有找一个之前已经跟你合作过哪怕现在不怎么闲的臭皮匠效果好哦~，所以基于局部的最少连接调度算法，维持的这种映射的作用是，如果来了一个请求，相对应的映射的那台服务器，没有超载，ok交给老伙伴完事吧，俺放心，如果那台服务器不存在，或者是超载的状态且有其他服务器工作在一半的负载状态，则按最少连接调度算法在集群其余的服务器中找一台将请求分配给它。

⑥基于复制的局部最少连接调度算法

这种调度算法同样应用于cache系统，但它维持的不是到一台服务器的映射而是到一组服务器的映射，当有新的请求到来，根据最小连接原则，从该映射的服务器组中选择一台服务器，如果它没有超载则交给它去处理这个请求，如果发现它超载，则从服务器组外的集群中，按最少连接原则拉一台机器加入服务器组，并且在服务器组有一段时间未修改后，将最忙的那台服务器从服务器组中剔除。

5.Sqoop用起来感觉怎样?

说实话，Sqoop在导入数据的速度上确实十分感人，通过进一步了解，发现Sqoop1和

相比，Sqoop无论是从导入数据的效率上还是从支持插件的丰富程度上，Sqoop还是相当不错滴!!

6.ZooKeeper的角色以及相应的Zookepper工作原理?

果然，人的记忆力是有衰减曲线的，当面试官抛出这个问题后，前者角色，我只答出了两种(leader和follower)，后者原理压根就模糊至忘记了。所以恶补了一下，涉及到Zookeeper的角色大概有如下四种：leader、learner(follower)、observer、client。其中leader主要用来决策和调度，follower和observer的区别仅仅在于后者没有写的职能，但都有将client请求提交给leader的职能，而observer的出现是为了应对当投票压力过大这种情形的，client就是用来发起请求的。而Zookeeper所用的分布式一致性算法包括leader的选举其实和-原始部落的获得神器为酋长，或者得玉玺者为皇帝类似，谁id最小，谁为leader，会根据你所配置的相应的文件在相应的节点机下生成id，然后相应的节点会通过getchildren()这个函数获取之前设置的节点下生成的id，谁最小，谁是leader。并且如果万一这个leader挂掉了或者堕落了，则由次小的顶上。而且在配置相应的zookeeper文件的时候回有类似于如下字样的信息：Server.x=AAAA:BBBB:CCCC。其中的x即为你的节点号哈，AAAA对应你所部属zookeeper所在的ip地址，BBBB为接收client请求的端口，CCCC为重新选举leader端口。

7.HBase的Insert与Update的区别?

这个题目是就着最近的一次项目问的，当时实现的与hbase交互的三个方法分别为insert、delete、update。由于那个项目是对接的一个项目，对接的小伙伴和我协商了下，不将update合并为insert，如果合并的话，按那个项目本身，其实通过insert执行overwrite相当于间接地Update，本质上，或者说在展现上是没什么区别的包括所调用的put。但那仅仅是就着那个项目的程序而言，如果基于HBaseshell层面。将同一rowkey的数据插入HBase，其实虽然展现一条，但是相应的timestamp是不一样的，而且最大的版本数可以通过配置文件进行相应地设置。

8.请简述大数据的结果展现方式。

1)报表形式

基于数据挖掘得出的数据报表，包括数据表格、矩阵、图形和自定义格式的报表等，使用方便、设计灵活。

2)图形化展现

提供曲线、饼图、堆积图、仪表盘、鱼骨分析图等图形形式宏观展现模型数据的分布情况，从而便于进行决策。

3)KPI展现

可度量的目标快速评估进度。

4)查询展现

按数据查询条件和查询内容，以数据表格来汇总查询结果，提供明细查询功能，并可在查询的数据表格基础上进行上钻、下钻、旋转等操作。

9.例举身边的大数据。

i.QQ，微博等社交软件产生的数据

ii.天猫，京东等电子商务产生的数据

iii.互联网上的各种数据

10.简述大数据的数据管理方式。

答：对于图像、视频、URL、地理位置等类型多样的数据，难以用传统的结构化方式描述，因此需要使用由多维表组成的面向列存储的数据管理系统来组织和管理数据。也就是说，将数据按行排序，按列存储，将相同字段的数据作为一个列族来聚合存储。不同的列族对应数据的不同属性，这些属性可以根据需求动态增加，通过这样的分布式实时列式数据库对数据统一进行结构化存储和管理，避免了传统数据存储方式下的关联查询。

11.什么是大数据?

答：大数据是指无法在容许的时间内用常规软件工具对其内容进行抓取、管理和处理的数据。

12.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;

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Hashmap，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;

13.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TopK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TopK，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。

14.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

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。

15.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TOPK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。

16.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

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Bloomfilter，4G内存大概可以表示340亿bit。将其中一个文件中的url使用Bloomfilter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Bloomfilter，如果是，那么该url应该是共同的url(注意会有一定的错误率)。

Bloomfilter日后会在本BLOG内详细阐述。

17.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*2bit=1GB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。

18.腾讯面试题：给40亿个不重复的unsignedint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unsignedint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:

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。

19.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

20.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。

21.一个文本文件，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)。

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。

一、Bloomfilter

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

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bloomfilter内存上通常都是节省的。

扩展：

Bloomfilter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。Countingbloomfilter(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。SpectralBloomFilter(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openhashing，也称为拉链法;另一种就是closedhashing，也称开地址法，openedaddressing。

扩展：

d-lefthashing中的d是多个的意思，我们先简化这个问题，看一看2-lefthashing。2-

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：bloomfilter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99999999，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。

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directaddrtable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(Invertedindex)

适用范围：搜索引擎，关键字查询

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=“itiswhatitis”

T1=“whatisit”

T2=“itisabanana”

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：

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).ThecanonicalexampleapplicationofMapReduceisaprocesstocountheappearancesof

eachdifferentwordinasetofdocuments:

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。

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

【某公司笔试面试题】

1使用mr，spark,sparksql编写wordcount程序

【Spark版本】

```
valconf=newSparkConf().setAppName("wd").setMaster("local[1]")
valsc=newSparkContext(conf,2)
//加载
vallines=sc.textFile("tructField("name",DataTypes.StringType,true)")
valparis=lines.flatMap(line=>line.split("^A"))
valwords=paris.map((_,1))
valresult=words.reduceByKey(_+_).sortBy(x=>x._1,false)
//打印
result.foreach(
wds=>{
println("单词："+wds._1+"个数："+wds._2)
}
)
sc.stop()
```

【sparksql版本】

```
valconf=newSparkConf().setAppName("sqlWd").setMaster("local[1]")
valsc=newSparkContext(conf)
valsqlContext=newSQLContext(sc)
//加载
```

valwords=lines.flatMap(x=>x.split("")).map(y=>Row(y))

valstructType=StructType(Array(StructField("name",DataTypes.StringType,true)))

valdf=sqlContext.createDataFrame(rows,structType)

df.registerTempTable("t_word_count")

sqlContext.udf.register("num_word",(name:String)=>1)

sqlContext.sql("selectname,num_word(name)fromt_word_count").groupBy(df.col("name")).count().show()

sc.stop()

2hive的使用，内外部表的区别，分区作用，UDF和Hive优化

(1)hive使用：仓库、工具

(2)hive内外部表：内部表数据永久删除，外部表数据删除后、其他人依然可以访问

(3)分区作用：防止数据倾斜

(4)UDF函数：用户自定义的函数(主要解决格式，计算问题)，需要继承UDF类

java代码实现

classTestUDFHiveextendsUDF{

publicStringevalute(Stringstr){

try{

return"hello"+str

}catch(Exceptione){

returnstr+"error"

}

}

(5)Hive优化：看做mapreduce处理

a排序优化：sortby效率高于orderby

b分区：使用静态分区(statu_date="20160516",location="beijin")，每个分区对应hdfs上的一个目录

c减少job和task数量：使用表链接操作

d解决groupby数据倾斜问题：设置hive.groupby.skewindata=true，那么hive会自动负载均衡

e小文件合并成大文件：表连接操作

f使用UDF或UDAF函数：hive中UDTF编写和使用(转) - ggjucheng - 博客园

3Hbase的rk设计，Hbase优化

aowkey:hbase三维存储中的关键(rowkey：行键，columnKey(family+quilaty)：列键，timestamp：时间戳)

owkey字典排序、越短越好

使用id+时间：9527+20160517使用hash散列：dsakjkdfuwdsf+9527+20160518

应用中，rowkey一般10~100bytes,8字节的整数倍，有利于提高操作系统性能

bHbase优化

分区：RegionSplit()方法NUMREGIONS=9

column不超过3个

硬盘配置，便于regionServer管理和数据备份及恢复

分配合适的内存给regionserver

其他：

hbase查询

(2)scan

使用startRow和endRow限制

4Linux常用操作

aawk：

awk-F:`BEGIN{print"nameip"}{print$1$7}END{print"结束"}`/etc/passwd

last|head-5|awk`BEGIN{print"nameip"}{print$1$3}END{print"结束了"}`

bsed

5java线程2种方式实现、设计模式、链表操作、排序

(1)2种线程实现

aThread类继承

TestCLth=newTestCL()//类继承Thread

th.start()

b实现Runnable接口

Threadth=newThread(newRunnable(){

publicvoidrun(){

//实现

}

})

th.start()

(2)设计模式，分为4类

a创建模式：如工厂模式、单例模式

c行为模式：观察者模式

d线程池模式

6【最熟悉的一个项目简介、架构图、使用的技术、你负责哪块】

7cdh集群监控

(1)数据库监控(2)主机监控(3)服务监控(4)活动监控

8计算机网络工作原理

将分散的机器通过数据通信原理连接起来，实现共享!

9hadoop生态系统

hdfsmapreducehivehbasezookeeperlume

hdfs原理及各个模块的功能mapreduce原理mapreduce优化数据倾斜

11系统维护：hadoop升级datanode节点

12【讲解项目要点：数据量、多少人、分工、运行时间、项目使用机器、算法、技术】

13【学会向对方提问】

14jvm运行机制及内存原理

运行：

I加载.class文件

II管理并且分配内存

III垃圾回收

内存原理：

IJVM装载环境和配置

II装载JVM.dll并初始化JVM.dll

15hdfs、yarn参数调优

mapreduce.job.jvm.num.tasks

默认为1，设置为-1，重用jvm

16Hbase、Hive、impala、zookeeper、Storm、spark原理和使用方法、使用其架构图讲解

【某公司笔试题】

1、如何为一个hadoop任务设置mappers的数量

答案：

使用job.setNumMapTask(intn)手动分割，这是不靠谱的

官方文档：“Note:Thisisonlyahinttotheframework”说明这个方法只是提示作用，不起决定性作用

实际上要用公式计算：

Max(min.split，min(max.split，block))就设置分片的最大最下值computeSplitSize()设置

参考：深度分析如何在Hadoop中控制Map的数量 - 张贵宾的技术专栏 - 博客频道 - CSDN.NET

2、有可能使hadoop任务输出到多个目录中么?如果可以，怎么做?

答案：在1.X版本后使用MultipleOutputs.java类实现

源码：

```
MultipleOutputs.addNamedOutput(conf,"text2",TextOutputFormat.class,Long.class,String.class);

MultipleOutputs.addNamedOutput(conf,"text3",TextOutputFormat.class,Long.class,String.class);
```

参考：MapReduce中的自定义多目录/文件名输出HDFS - leejun2005的个人页面 - 开源中国

发音：Multiple['m?lt?pl]--》许多的

3、如何为一个hadoop任务设置要创建的reducer的数量

答案：job.setNumReduceTask(intn)

或者调整hdfs-site.xml中的mapred.tasktracker.reduce.tasks.maximum默认参数值

4、在hadoop中定义的主要公用InputFormats中，哪一个是默认值：

(A)TextInputFormat

(B)KeyValueInputFormat

(C)SequenceFileInputFormat

答案：A

5、两个类TextInputFormat和KeyValueTextInputFormat的区别?

答案：

?FileInputFormat的子类：

TextInputFormat(默认类型，键是LongWritable类型，值为Text类型，key为当前行在文件中的偏移量，value为当前行本身);

?KeyValueTextInputFormat(适合文件自带key，value的情况，只要指定分隔符即可，比较实用，默认是分割);

源码：

StringsepStr=job.get("mapreduce.input.keyvaluelinerecordreader.key.value.separator","");

注意：在自定义输入格式时，继承FileInputFormat父类

6、在一个运行的hadoop任务中，什么是InputSpilt?

答案：InputSplit是MapReduce对文件进行处理和运算的输入单位，只是一个逻辑概念，每个InputSplit并没有对文件实际的切割，只是记录了要处理的数据的位置(包括文件的path和hosts)和

长度(由start和length决定)，默认情况下与block一样大。

7、Hadoop框架中，文件拆分是怎么被调用的?

答案：JobTracker，创建一个InputFormat的实例，调用它的getSplits()方法，把输入目录的文件拆分成FileSplist作为Mappertask的输入，生成Mappertask加入Queue。

源码中体现了拆分的数量

longgoalSize=totalSize/(numSplits==0?1:numSplits);

longminSize=Math.max(job.getLong(org.apache.hadoop.mapreduce.lib.input.

FileInputFormat.SPLIT_MINSIZE,1),minSplitSize);//minSplitSize默认是1

8、分别举例什么情况下使用combiner,什么情况下不会使用?

答案：Combiner适用于对记录汇总的场景(如求和)，但是，求平均数的场景就不能使用Combiner了

9、Hadoop中job和Tasks之间的区别是什么?

答案:

job是工作的入口，负责控制、追踪、管理任务，也是一个进程

包含maptask和reducetask

Tasks是map和reduce里面的步骤，主要用于完成任务，也是线程

10、Hadoop中通过拆分任务到多个节点运行来实现并行计算，但是某些节点运行较慢会拖慢整个任务的运行，hadoop采用何种机制应对这种情况?

答案：结果查看监控日志，得知产生这种现象的原因是数据倾斜问题

解决:

(1)调整拆分mapper的数量(partition数量)

(2)增加jvm

(3)适当地将reduce的数量变大

灵活性?

答案：用可执行文件作为Mapper和Reducer，接受的都是标准输入，输出的都是标准输出

12、参考下面的M/R系统的场景：

--HDFS块大小为64MB

--输入类型为FileInputFormat

--有3个文件的大小分别是：64k65MB127MB

Hadoop框架会把这些文件拆分为多少块?

答案：

64k------->一个block

65MB---->两个文件：64MB是一个block，1MB是一个block

127MB--->两个文件：64MB是一个block,63MB是一个block

13、Hadoop中的RecordReader的作用是什么?

答案：属于split和mapper之间的一个过程

将inputsplit输出的行为一个转换记录，成为key-value的记录形式提供给mapper

14、Map阶段结束后，Hadoop框架会处理：Partitioning,shuffle和sort,在这个阶段都会发生了什么?

答案：

MR一共有四个阶段，splitmapshuffreduce在执行完map之后，可以对map的输出结果进行分区，

分区：这块分片确定到哪个reduce去计算(汇总)

排序：在每个分区中进行排序，默认是按照字典顺序。

Group：在排序之后进行分组

答案：

Partitioner是在map函数执行context.write()时被调用。

用户可以通过实现自定义的?Partitioner来控制哪个key被分配给哪个?Reducer。

查看源码知道：

如果没有定义partitioner，那么会走默认的分区Hashpartitioner

publicclassHashPartitionerextendsPartitioner{

/**Use{@linkObject#hashCode()}topartition.*/

publicintgetPartition(Kkey,Vvalue,intnumReduceTasks){

return(key.hashCode()&Integer.MAX_VALUE)%numReduceTasks;

}

}

16、什么是Combiner?

答案：这是一个hadoop优化性能的步骤，它发生在map与reduce之间

目的：解决了数据倾斜的问题，减轻网络压力，实际上时减少了maper的输出

源码信息如下：

publicvoidreduce(Textkey,Iteratorvalues,

OutputCollectoroutput,Reporterreporter)

throwsIOException{

LongWritablemaxValue=null;

while(values.hasNext()){

LongWritablevalue=values.next();

```
maxValue=value;
}elseif(value.compareTo(maxValue)>0){
maxValue=value;
}
}
output.collect(key,maxValue);
}
```

在collect实现类中，有这样一段方法

```
publicsynchronizedvoidcollect(Kkey,Vvalue)
throwsIOException{
outCounter.increment(1);
writer.append(key,value);
if((outCounter.getValue()%progressBar)==0){
progressable.progress();
}

}
```

大数据 大数据处理 大数据运维

还没有评论

写下你的评论