# ****BigData面试题总结****

## ****JAVA相关****

### ****1-1）List与set的区别？****

老掉牙的问题了，还在这里老生常谈：List特点：元素有放入顺序，元素可重复，Set特点：元素无放入顺序，元素不可重复。

### ****1-2）数据库的三大范式？****

原子性、一致性、唯一性

### ****1-3）java的io类的图解****

### ****1-4）对象与引用对象的区别****

对象就是好没有初始化的对象，引用对象即使对这个对象进行了初始化，这个初始化可以使自己的直接new的也可以是直接其他的赋值的，那么背new或者背其他赋值的我们叫做是引用对象，最大的区别于

### ****1-5）谈谈你对反射机制的理解及其用途？****

反射有三种获取的方式，分别是：forName/getClass/直接使用class方式使用反射可以获取类的实例

### ****1-6）列出至少五种设计模式****

设计方式有工厂法，懒加载，观察者模式，静态工厂，迭代器模式，外观模式、、、、

### ****1-7）RPC原理？****

Rpc分为同步调用和一部调用，异步与同步的区别在于是否等待服务器端的返回值。Rpc的组件有RpcServer,RpcClick,RpcProxy,RpcConnection,RpcChannel,RpcProtocol,RpcInvoker等组件，

### ****1-8）ArrayList、Vector、LinkedList的区别及其优缺点？HashMap、HashTable的区别及优缺点？****

ArrayList和Vector是采用数组方式存储数据的,是根据索引来访问元素的，都可以

根据需要自动扩展内部数据长度，以便增加和插入元素，都允许直接序号索引元素，但

是插入数据要涉及到数组元素移动等内存操作，所以索引数据快插入数据慢，他们最大

的区别就是synchronized同步的使用。

LinkedList使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但

是插入数据时只需要记录本项的前后项即可，所以插入数度较快！

如果只是查找特定位置的元素或只在集合的末端增加、移除元素，那么使用Vector

或ArrayList都可以。如果是对其它指定位置的插入、删除操作，最好选择LinkedList

HashMap、HashTable的区别及其优缺点：

HashTable中的方法是同步的HashMap的方法在缺省情况下是非同步的因此在多线程环境下需要做额外的同步机制。

HashTable不允许有null值key和value都不允许，而HashMap允许有null值key和value都允许因此HashMap使用containKey（）来判断是否存在某个键。

HashTable使用Enumeration，而HashMap使用iterator。

Hashtable是Dictionary的子类，HashMap是Map接口的一个实现类。

### ****1-9）使用StringBuffer而不是String****

当需要对字符串进行操作时，使用StringBuffer而不是String，String是read-only的，如果对它进行修改，会产生临时对象，而StringBuffer是可修改的，不会产生临时对象。

### ****1-10）集合的扩充****

ArrayListlist=newArrayList(90000);list扩充多少次？？

**public**ArrayList(){

**this**(10);

}

默认的扩充是10由此计算

### ****1-11）java的拆包与封包的问题****

System.**out**.println("5"+2);

52

### ****1-12）Java中Class.forName和ClassLoader.loadClass的区别****

Class.forName("xx.xx")等同于Class.forName("xx.xx",true,CALLClass.class.getClassLoader())，第二个参数(bool)表示装载类的时候是否初始化该类，即调用类的静态块的语句及初始化静态成员变量。

ClassLoaderloader=Thread.currentThread.getContextClassLoader();//也可以用(ClassLoader.getSystemClassLoader())

Classcls=loader.loadClass("xx.xx");//这句话没有执行初始化

forName可以控制是否初始化类，而loadClass加载时是没有初始化的。

### ****1-13）hashMap与hashTable的区别****

HashMapHashtable

父类AbstractMapDictiionary

是否同步否是

k，v可否null是否

Hashtable和HashMap采用的hash/rehash[算法](http://lib.csdn.net/base/datastructure)都大概一样，所以性能不会有很大的差异。

### ****1-14）怎样实现数组的反转****

ArrayListarrayList=newArrayList();

arrayList.add("A");

arrayList.add("B");

对数组进行反转

Collections.reverse(arrayList);

### ****1-15）请使用JAVA实现二分查找****

一般的面试者都是些向看看你的思路，所以一般答题时只需要把思路写出来即可。

具体的实现如下：

二分查找就是折半查找，要想折半就必须把原来的数据进行排序，才能方便的查找：

实现代码如下：

publicstaticintbinarySearch(int[]srcArray,intdes){

intlow=0;

inthigh=srcArray.length-1;

while(low<=high){

intmiddle=(low+high)/2;

if(des==srcArray[middle]){

returnmiddle;

}elseif(des<srcArray[middle]){

high=middle-1;

}else{

low=middle+1;

}

}

return-1;

}

### ****1-16）java中有两个线程怎样等待一个线程执行完毕****

可以使用join关键字

### ****1-17）hashmaphashtablecurrentHashMap的使用区别****

hashmaphashtable的醉的的区别在于hashtable是线程安全的，而hashmap不是线程安全的，currentHashMap也是线程安全的。

ConcurrentHashMap是使用了锁分段技术技术来保证线程安全的。所分段的技术是：讲数据分成一段一段的储存，给每一段的数据添加一把锁，当线程访问一个数据时，其他的数据可以被访问。

### ****1-18）简单描述一下java的gc机制，常用的JAVA调优的方法，OOM如何产生的，如何处理OOM问题？？？****

1、程序在运行时会产生很多的对象的信息，当这些对象的信息没有用时，则会被gc回收

2、调优的方式主要是调节年轻代与老年代的内存的大小

3、OOM是OutOfMemory的缩写(搞得跟多高大上似的)就是线程创建的多了，没有及时的回收过来所产生的，代码如下：

publicclassJavaVMStackOOM{

privatevoiddontStop(){

while(true){

}

}

publicvoidstackLeakByThread(){

while(true){

Threadthread=newThread(newRunnable(){

@Override

publicvoidrun(){

dontStop();

}

});

thread.start();

}

}

publicstaticvoidmain(String[]args){

JavaVMStackOOMoom=newJavaVMStackOOM();

oom.stackLeakByThread();

}

4、既然知道以上的现象，在写代码时应该注意，不要过多的创建线程的数目。

## ****Linux相关****

### ****2-1）关闭不必要的服务****

A、使用ntsysv命令查看开启与关闭的服务

B、停止打印服务

[root@hadoop1/]#/etc/init.d/cupsstop

[root@hadoop1/]#chkconfigcupsoff

### ****2-2）关闭IP6****

[root@hadoop1/]#vim/etc/modprobe.conf

在下面添加一下配置：

aliasnet-pf-10off

aliasipv6off

### ****2-3）调整文件的最大的打开数****

查看当前的文件的数量：[root@hadoop1/]#ulimit-a

修改配置：

[root@hadoop1/]#vi/etc/security/limits.conf在文件最后加上：

\*softnofile65535

\*hardnofile65535

\*softnproc65535

\*hardnproc65535

### ****2-4）修改linux内核参数****

[root@hadoop1/]#vi/etc/sysctl.conf

在文本的最后追加一下内容：

net.core.somaxconn=32768

表示物理内存使用到90%（100-10=90）的时候才使用swap交换区

### ****2-5）关闭noatime****

在最后追加一下内容

/dev/sda2/dataext3noatime,nodiratime00

### ****2-6)请用shell命令把某一个文件下的所有的文件分发到其他的机器上****

Scp-r/user/localhadoop2:/user/local

### ****2-7）echo1+1&&echo"1+1"会输出什么****

[root@hadoop1~]#echo1+1&&echo"1+1"

1+1

1+1

[root@hadoop1~]#echo1+1&&echo"1+1"&&echo"1+"1

1+1

1+1

1+1

### ****2-8）在当前的额目录下找出包含祖母a并且文件的额大小大于55K的文件****

[root@hadoop1test]#find.|grep-ri"a"

a.text:a

后半句没有写出来，有时间在搞

### ****2-9）linux用什么命令查看cpu,硬盘，内存的信息？****

Top命令

## ****Hadoop相关****

### ****3-1）简单概述hdfs原理，以及各个模块的职责****

1、客户端向nameNode发送要上传文件的请求

2、nameNode返回给用户是否能上传数据的状态

3、加入用户端需要上传一个1024M的文件，客户端会通过Rpc请求NameNode，并返回需要上传给那些DataNode(分配机器的距离以及空间的大小等),namonode会选择就近原则分配机器。

4、客户端请求建立block传输管道chnnel上传数据

5、在上传是datanode会与其他的机器建立连接并把数据块传送到其他的机器上

6、dataNode向namenode汇报自己的储存情况以及自己的信息

7、档第一个快上传完后再去执行其他的复制的传送

### ****3-2）mr的工作原理****

1、当执行mr程序是，会执行一个Job

2、客户端的jobClick会请求namenode的jobTracker要执行任务

3、jobClick会去HDFS端复制作业的资源文件

4、客户端的jobClick会向namenode提交作业,让namenode做准备

5、Namenode的jobTracker会去初始化创建的对象

6、Namenode会获取hdfs的划分的分区

7、Namenode去检查TaskTracker的心跳信息，查看存活的机器

8、当执行的datenode执行任务时Datenode会去HDFS获取作业的资源的文件

9、TaskTracker会去执行代码，并登陆JVM的执行渠道

10、JVM或执行MapTask或者ReduceTask

11、执行终结

### ****3-3）怎样判断文件时候存在****

这是[Linux](http://lib.csdn.net/base/linux)上的知识，只需要在IF[-f]括号中加上-f参数即可判断文件是否存在

### ****3-4）fsimage和edit的区别？****

大家都知道namenode与secondarynamenode的关系，当他们要进行数据同步时叫做checkpoint时就用到了fsimage与edit，fsimage是保存最新的元数据的信息，当fsimage数据到一定的大小事会去生成一个新的文件来保存元数据的信息，这个新的文件就是edit，edit会回滚最新的数据。

### ****3-5）hdfs中的block默认保存几份？****

不管是hadoop1.x还是hadoop2.x都是默认的保存三份，可以通过参数dfs.replication就行修改，副本的数目要根据机器的个数来确定。

### ****3-6）列举几个配置文件优化？****

Core-site.xml文件的优化

fs.trash.interval

默认值：0

说明：这个是开启hdfs文件删除自动转移到垃圾箱的选项，值为垃圾箱文件清除时间。一般开启这个会比较好，以防错误删除重要文件。单位是分钟。

dfs.namenode.handler.count

默认值：10

说明：[Hadoop](http://lib.csdn.net/base/hadoop)系统里启动的任务线程数，这里改为40，同样可以尝试该值大小对效率的影响变化进行最合适的值的设定。

mapreduce.tasktracker.http.threads

默认值：40

说明：map和reduce是通过http进行数据传输的，这个是设置传输的并行线程数。

### ****3-7)谈谈数据倾斜，如何发生的，并给出优化方案****

数据的倾斜主要是两个的数据相差的数量不在一个级别上，在只想任务时就造成了数据的倾斜，可以通过分区的方法减少reduce数据倾斜性能的方法，例如;抽样和范围的分区、自定义分区、数据大小倾斜的自定义侧咯

### ****3-8）简单概括安装hadoop的步骤****

1.创建[hadoop](http://lib.csdn.net/base/hadoop)帐户。

2.setup.改IP。

3.安装[Java](http://lib.csdn.net/base/java)，并修改/etc/profile文件，配置java的环境变量。

4.修改Host文件域名。

5.安装SSH，配置无密钥通信。

6.解压hadoop。

7.配置conf文件下hadoop-env.sh、core-site.sh、mapre-site.sh、hdfs-site.sh。

8.配置hadoop的环境变量。

9.Hadoopnamenode-format

10.Start-all.sh

### ****3-9）简单概述hadoop中的角色的分配以及功能****

Namenode:负责管理元数据的信息

SecondName:做namenode冷备份，对于namenode的机器当掉后能快速切换到制定的Secondname上

DateNode:主要做储存数据的。

JobTracker:管理任务，并把任务分配到taskTasker

TaskTracker：执行任务的

### ****3-10）怎样快速的杀死一个job****

1、执行hadoopjob-list拿到job-id

2、Hadoopjobkillhadoop-id

### ****3-11)新增一个节点时怎样快速的启动****

Hadoop-daemon.shstartdatanode

### ****3-12)你认为用java,streaming,pipe方式开发map/reduce,各有什么优点****

开发mapReduce只用过java与[Hive](http://lib.csdn.net/base/hive),不过使用java开发mapreduce显得笨拙，效率也慢，基于java慢的原因于是[hive](http://lib.csdn.net/base/hive)，这样就方便了查询与设计

### ****3-13）简单概述hadoop的join的方法****

Hadoop常用的jion有reducesidejoin,mapsidejoin,SemiJoin不过reducesidejoin与mapsidejoin比较常用，不过都是比较耗时的。

### ****3-14）简单概述hadoop的combinet与partition的区别****

combine和partition都是函数，中间的步骤应该只有shuffle！combine分为map端和reduce端，作用是把同一个key的键值对合并在一起，可以自定义的,partition是分割map每个节点的结果，按照key分别映射给不同的reduce，也是可以自定义的。这里其实可以理解归类。

### ****3-15)hdfs的数据压缩算法****

Hadoop的压缩算法有很多，其中比较常用的就是gzip算法与bzip2算法，都可以可通过CompressionCodec来实现

### ****3-16）hadoop的调度****

Hadoop的调度有三种其中fifo的调度hadoop的默认的，这种方式是按照作业的优先级的高低与到达时间的先后执行的，还有公平调度器：名字见起意就是分配用户的公平获取共享集群呗!容量调度器:让程序都能货到执行的能力，在队列中获得资源。

### ****3-17）reduce后输出的数据量有多大？****

输出的数据量还不是取决于map端给他的数据量，没有数据reduce也没法运算啊!!

### ****3-18)datanode在什么情况下不会备份？****

Hadoop保存的三个副本如果不算备份的话，那就是在正常运行的情况下不会备份，也是就是在设置副本为1的时候不会备份，说白了就是单台机器呗！！还有datanode在强制关闭或者非正常断电不会备份。

### ****3-19）combine出现在那个过程？****

Hadoop的map过程，根据意思就知道结合的意思吗，剩下的你们就懂了。想想wordcound

### ****3-20)hdfs的体系结构？****

HDFS有namenode、secondraynamenode、datanode组成。

namenode负责管理datanode和记录元数据

secondraynamenode负责合并日志

datanode负责存储数据

### ****3-21)hadoopflush的过程？****

Flush就是把数据落到磁盘，把数据保存起来呗!

### ****3-22)什么是队列****

队列的实现是链表，消费的顺序是先进先出。

### ****3-23）三个datanode，当有一个datanode出现错误会怎样？****

第一不会给储存带来影响，因为有其他的副本保存着，不过建议尽快修复，第二会影响运算的效率，机器少了，reduce在保存数据时选择就少了，一个数据的块就大了所以就会慢。

### ****3-24）mapReduce的执行过程****

首先map端会Text接受到来自的数据，text可以把数据进行操作，最后通过context把key与value写入到下一步进行计算，一般的reduce接受的value是个集合可以运算，最后再通过context把数据持久化出来。

### ****3-25）Cloudera提供哪几种安装CDH的方法****

·Clouderamanager

·Tarball

·Yum

·Rpm

### ****3-26）选择题与判断题****

http://blog.csdn[.NET](http://lib.csdn.net/base/dotnet)/jiangheng0535/article/details/16800415

### ****3-27）hadoop的combinet与partition效果图****

### ****3-28）hadoop的机架感知（或者说是扩普）****

看图说话

数据块会优先储存在离namenode进的机器或者说成离namenode机架近的机器上，正好是验证了那句话不走网络就不走网络，不用磁盘就不用磁盘。

### ****3-29）文件大小默认为64M，改为128M有啥影响？****

这样减少了namenode的处理能力，数据的元数据保存在namenode上，如果在网络不好的情况下会增到datanode的储存速度。可以根据自己的网络来设置大小。

### ****3-30）datanode首次加入cluster的时候，如果log报告不兼容文件版本，那需要namenode执行格式化操作，这样处理的原因是？****

这样处理是不合理的，因为那么namenode格式化操作，是对文件系统进行格式

化，namenode格式化时清空dfs/name下空两个目录下的所有文件，之后，会在目

录dfs.name.dir下创建文件。

文本不兼容，有可能时namenode与datanode的数据里的namespaceID、

clusterID不一致，找到两个ID位置，修改为一样即可解决。

### ****3-31）什么hadoopstreaming？****

提示：指的是用其它语言处理

### ****3-32）MapReduce中排序发生在哪几个阶段？这些排序是否可以避免？为什么？****

一个MapReduce作业由Map阶段和Reduce阶段两部分组成，这两阶段会对数

据排序，从这个意义上说，MapReduce框架本质就是一个DistributedSort。在Map

阶段，在Map阶段，MapTask会在本地磁盘输出一个按照key排序（采用的是快速

排序）的文件（中间可能产生多个文件，但最终会合并成一个），在Reduce阶段，每

个ReduceTask会对收到的数据排序，这样，数据便按照Key分成了若干组，之后以

组为单位交给reduce（）处理。很多人的误解在Map阶段，如果不使用Combiner

便不会排序，这是错误的，不管你用不用Combiner，MapTask均会对产生的数据排

序（如果没有ReduceTask，则不会排序，实际上Map阶段的排序就是为了减轻Reduce

端排序负载）。由于这些排序是MapReduce自动完成的，用户无法控制，因此，在

hadoop1.x中无法避免，也不可以关闭，但hadoop2.x是可以关闭的。

### ****3-33）hadoop的shuffer的概念****

Shuffer是一个过程，实在map端到reduce在调reduce数据之前都叫shuffer,主要是分区与排序，也就是内部的缓存分分区以及分发（是reduce来拉数据的）和传输

### ****3-34）hadoop的优化****

1、优化的思路可以从配置文件和系统以及代码的设计思路来优化

2、配置文件的优化：调节适当的参数，在调参数时要进行[测试](http://lib.csdn.net/base/softwaretest)

3、代码的优化：combiner的个数尽量与reduce的个数相同，数据的类型保持一致，可以减少拆包与封包的进度

4、系统的优化：可以设置[linux](http://lib.csdn.net/base/linux)系统打开最大的文件数预计网络的带宽MTU的配置

5、为job添加一个Combiner，可以大大的减少shuffer阶段的maoTask拷贝过来给远程的reducetask的数据量，一般而言combiner与reduce相同。

6、在开发中尽量使用stringBuffer而不是string，string的模式是read-only的，如果对它进行修改，会产生临时的对象，二stringBuffer是可修改的，不会产生临时对象。

7、修改一下配置：

一下是修改mapred-site.xml文件

修改最大槽位数

槽位数是在各个tasktracker上的mapred-site.xml上设置的，默认都是2

<property>

<name>mapred.tasktracker.map.tasks.maximum</name>

task的最大数

<value>2</value>

</property>

<property>

<name>mapred.tasktracker.reduce.tasks.maximum</name>

ducetask的最大数

<value>2</value>

</property>

调整心跳间隔

集群规模小于300时，心跳间隔为300毫秒

mapreduce.jobtracker.heartbeat.interval.min心跳时间

北京市昌平区建材城西路金燕龙办公楼一层电话：400-618-9090

mapred.heartbeats.in.second集群每增加多少节点，时间增加下面的值

mapreduce.jobtracker.heartbeat.scaling.factor集群每增加上面的个数，心跳增多少

启动带外心跳

mapreduce.tasktracker.outofband.heartbeat默认是false

配置多块磁盘

mapreduce.local.dir

配置RPChander数目

mapred.job.tracker.handler.count默认是10，可以改成50，根据机器的能力

配置HTTP线程数目

tasktracker.http.threads默认是40，可以改成100根据机器的能力

选择合适的压缩方式

以snappy为例：

<property>

<name>mapred.compress.map.output</name>

<value>true</value>

</property>

<property>

<name>mapred.map.output.compression.codec</name>

<value>org.apache.hadoop.io.compress.SnappyCodec</value>

</property>

### ****3-35)3个datanode中有一个个datanode出现错误会怎样？****

这个datanode的数据会在其他的datanode上重新做备份。

### ****3-36）怎样决定mapreduce的中的map以及reduce的数量****

在mapreduce中map是有块的大小来决定的，reduce的数量可以按照用户的业务来配置。

### ****3-37）两个文件合并的问题****

[**给定a、b两个文件，各存放50亿个url，每个url各占用64字节，内存限制是4G，如何找出a、b文件共同的url？**](http://www.cnblogs.com/zemliu/archive/2012/08/16/2641036.html)

主要的思想是把文件分开进行计算，在对每个文件进行对比，得出相同的URL,因为以上说是含有相同的URL所以不用考虑数据倾斜的问题。详细的解题思路为：

可以估计每个文件的大小为5G\*64=300G，远大于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，然后根据所得值将url分别存储到1000个小文件（设为a0,a1,...a999）当中。这样每个小文件的大小约为300M。遍历文件b，采取和a相同的方法将url分别存储到1000个小文件(b0,b1....b999)中。这样处理后，所有可能相同的url都在对应的小文件(a0vsb0,a1vsb1....a999vsb999)当中，不对应的小文件（比如a0vsb99）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。

比如对于a0vsb0，我们可以遍历a0，将其中的url存储到hash\_map当中。然后遍历b0，如果url在hash\_map中，则说明此url在a和b中同时存在，保存到文件中即可。

如果分成的小文件不均匀，导致有些小文件太大（比如大于2G），可以考虑将这些太大的小文件再按类似的方法分成小小文件即可

### ****3-38）怎样决定一个job的map和reduce的数量****

map的数量通常是由hadoop集群的DFS块大小确定的，也就是输入文件的总块数，reduce端是复制map端的数据，相对于map端的任务，reduce节点资源是相对于比较缺少的，同时运行的速度会变慢，争取的任务的个数应该是0.95过着1.75。

### ****3-39）hadoop的sequencefile的格式，并说明下什么是JAVA的序列化，如何实现JAVA的序列化****

1、hadoop的序列化(sequencefile)是一二进制的形式来保存的

2、Java的序列化是讲对象的内容进行流化

3、实现序列化需要实现Serializable接口便可以了

### ****3-40）简单概述一下hadoop1与hadoop2的区别****

Hadoop2与hadoop1最大的区别在于HDFS的[架构](http://lib.csdn.net/base/architecture)与mapreduce的很大的区别，而且速度上有很大的提升，hadoop2最主要的两个变化是：namenode可以集群的部署了，hadoop2中的mapreduce中的jobTracker中的资源调度器与生命周期管理拆分成两个独立的组件，并命名为YARN

### ****3-41）YARN的新特性****

YARN是hadoop2.x之后才出的，主要是hadoop的HA(也就是集群)，磁盘的容错，资源调度器

### ****3-42）hadoopjoin的原理****

实现两个表的join首先在map端需要把表标示一下，把其中的一个表打标签，到reduce端再进行笛卡尔积的运算，就是reduce进行的实际的链接操作。

### ****3-43）hadoop的二次排序****

Hadoop默认的是HashPartitioner排序，当map端一个文件非常大另外一个文件非常小时就会产生资源的分配不均匀，既可以使用setPartitionerClass来设置分区，即形成了二次分区。

### ****3-44）hadoop的mapreduce的排序发生在几个阶段？****

发生在两个阶段即使map与reduce阶段

### ****3-45）请描述mapreduce中shuffer阶段的工作流程，如何优化shuffer阶段的？****

Mapreduce的shuffer是出在maptask到reducetask的这段过程中，首先会进入到copy过程，会通过http方式请求maptask所在的taskTracker获取maptask的输出的文件，因此当maptask结束，这些文件就会落到磁盘中，merge实在map端的动作，只是在map拷贝过来的数值，会放到内存缓冲区中，给shuffer使用，reduce阶段，不断的merge后最终会把文件放到磁盘中。

### ****3-46）mapreduce的combiner的作用是什么，什么时候不易使用？？****

Mapreduce中的Combiner就是为了避免map任务和reduce任务之间的数据传输而设置的，Hadoop允许用户针对maptask的输出指定一个合并函数。即为了减少传输到Reduce中的数据量。它主要是为了削减Mapper的输出从而减少网络带宽和Reducer之上的负载。

在数据量较少时不宜使用。

3-47）

## ****Zookeeper相关****

### ****4-1）写出你对zookeeper的理解****

随着[大数据](http://lib.csdn.net/base/hadoop)的快速发展，多机器的协调工作，避免主要机器单点故障的问题，于是就引入管理机器的一个软件，他就是zookeeper来协助机器正常的运行。

Zookeeper有两个角色分别是leader与follower，其中leader是主节点，其他的是副节点，在安装配置上一定要注意配置奇数个的机器上，便于zookeeper快速切换选举其他的机器。

在其他的软件执行任务时在zookeeper注册时会在zookeeper下生成相对应的目录，以便zookeeper去管理机器。

### ****4-2）zookeeper的搭建过程****

主要是配置文件zoo.cfg配置dataDir的路径一句dataLogDir的路径以及myid的配置以及server的配置，心跳端口与选举端口

## ****Hive相关****

### ****5-1）hive是怎样保存元数据的****

保存元数据的方式有：内存[数据库](http://lib.csdn.net/base/mysql)rerdy，本地[MySQL](http://lib.csdn.net/base/mysql)数据库，远程[mysql](http://lib.csdn.net/base/mysql)数据库，但是本地的mysql数据用的比较多，因为本地读写速度都比较快

### ****5-2）外部表与内部表的区别****

先来说下Hive中内部表与外部表的区别：

Hive创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

### ****5-3）对于hive，你写过哪些UDF函数，作用是什么****

UDF:userdefinedfunction的缩写，编写hiveudf的两种方式extendsUDF重写evaluate第二种extendsGenericUDF重写initialize、getDisplayString、evaluate方法

### ****5-4)Hive的sortby和orderby的区别****

orderby会对输入做全局排序，因此只有一个reducer（多个reducer无法保证全局有序）只有一个reducer，会导致当输入规模较大时，需要较长的计算时间。

sortby不是全局排序，其在数据进入reducer前完成排序.

因此，如果用sortby进行排序，并且设置mapred.reduce.tasks>1，则sortby只保证每个reducer的输出有序，不保证全局有序。

### ****5-5）hive保存元数据的方式以及各有什么特点？****

1、Hive有内存数据库derby数据库，特点是保存数据小，不稳定

2、mysql数据库，储存方式可以自己设定，持久化好，一般企业开发都用mysql做支撑

### ****5-6）在开发中问什么建议使用外部表？****

1、外部表不会加载到hive中只会有一个引用加入到元数据中

2、在删除时不会删除表，只会删除元数据，所以不必担心数据的

### ****5-7）hivepartition分区****

分区表，动态分区

### ****5-8）insertinto和overridewrite区别？****

insertinto：将某一张表中的数据写到另一张表中

overridewrite：覆盖之前的内容。

## ****Hbase相关****

### ****6-1）Hbase的rowkey怎么创建比较好？列族怎么创建比较好？****

Rowkey是一个二进制码流，Rowkey的长度被很多开发者建议说设计在10~100个字节，不过建议是越短越好，不要超过16个字节。在查找时有索引会加快速度。

Rowkey散列原则、Rowkey唯一原则、针对事务数据Rowkey设计、针对统计数据的Rowkey设计、针对通用数据的Rowkey设计、支持多条件查询的RowKey设计。

总结设计列族：

1、一般不建议设计多个列族

2、数据块的缓存的设计

3、激进缓存设计

4、布隆过滤器的设计(可以提高随机读取的速度)

5、生产日期的设计

6、列族压缩

7、单元时间版本

### ****6-2）Hbase的实现原理****

[Hbase](http://lib.csdn.net/base/hbase)的实现原理是rpcProtocol

### ****6-3)hbase过滤器实现原则****

感觉这个问题有问题，过滤器多的是啦，说的是哪一个不知道!!!!

[hbase](http://lib.csdn.net/base/hbase)的过滤器有：RowFilter、PrefixFilter、KeyOnlyFilter、RandomRowFilter、InclusiveStopFilter、FirstKeyOnlyFilter、ColumnPrefixFilter、ValueFilter、ColumnCountGetFilter、SingleColumnValueFilter、SingleColumnValueExcludeFilter、WhileMatchFilter、FilterList

你看这么多过滤波器呢，谁知道你问的那个啊！！

比较常用的过滤器有：RowFilter一看就知道是行的过滤器，来过滤行的信息。PrefixFilter前缀的过滤器，就是把前缀作为参数来查找数据呗！剩下的不解释了看过滤器的直接意思就OK了很简单。

### ****6-4）描述HBase,zookeeper搭建过程****

Zookeeper的问题楼上爬爬有步骤，hbase主要的配置文件有hbase.env.sh主要配置的是JDK的路径以及是否使用外部的ZK，hbase-site.xml主要配置的是与HDFS的链接的路径以及zk的信息，修改regionservers的链接其他机器的配置。

### ****6-5）hive如何调优？****

在优化时要注意数据的问题，尽量减少数据倾斜的问题，减少job的数量，同事对小的文件进行成大的文件，如果优化的设计那就更好了，因为hive的运算就是mapReduce所以调节mapreduce的参数也会使性能提高，如调节task的数目。

### ****6-6）hive的权限的设置****

Hive的权限需要在hive-site.xml文件中设置才会起作用，配置默认的是false，需要把hive.security.authorization.enabled设置为true，并对不同的用户设置不同的权限，例如select,drop等的操作。

### ****6-7)hbase写数据的原理****

1.首先，Client通过访问ZK来请求目标数据的地址。

2.ZK中保存了-ROOT-表的地址，所以ZK通过访问-ROOT-表来请求数据地址。

3.同样，-ROOT-表中保存的是.META.的信息，通过访问.META.表来获取具体的RS。

4..META.表查询到具体RS信息后返回具体RS地址给Client。

5.Client端获取到目标地址后，然后直接向该地址发送数据请求。

### ****6-8）hbase宕机了如何处理？****

HBase的RegionServer宕机超过一定时间后，HMaster会将其所管理的region重新分布到其他活动的RegionServer上，由于数据和日志都持久在HDFS中，

该操作不会导致数据丢失。所以数据的一致性和安全性是有保障的。

但是重新分配的region需要根据日志恢复原RegionServer中的内存MemoryStore表，这会导致宕机的region在这段时间内无法对外提供服务。

而一旦重分布，宕机的节点重新启动后就相当于一个新的RegionServer加入集群，为了平衡，需要再次将某些region分布到该server。

因此，RegionServer的内存表memstore如何在节点间做到更高的可用，是HBase的一个较大的挑战。

### ****6-9）Hbase中的metastore用来做什么的？****

Hbase的metastore是用来保存数据的，其中保存数据的方式有有三种第一种于第二种是本地储存，第二种是远程储存这一种企业用的比较多

### ****6-10)hbase客户端在客户端怎样优化？****

Hbase使用JAVA来运算的，索引Java的优化也适用于hbase，在使用过滤器事记得开启bloomfilter可以是性能提高3-4倍，设置HBASE\_HEAPSIZE设置大一些

### ****6-11）hbase是怎样预分区的？****

如何去进行预分区，可以采用下面三步：

1.取样，先随机生成一定数量的rowkey,将取样数据按升序排序放到一个集合里

2.根据预分区的region个数，对整个集合平均分割，即是相关的splitKeys.

3.HBaseAdmin.createTable(HTableDescriptortableDescriptor,byte[][]splitkeys)可以指定预分区的splitKey，即是指定region间的rowkey临界值

### ****6-12）怎样将mysql的数据导入到hbase中？****

不能使用sqoop，速度太慢了，提示如下：

A、一种可以加快批量写入速度的方法是通过预先创建一些空的regions，这样当

数据写入HBase时，会按照region分区情况，在集群内做数据的负载均衡。

B、hbase里面有这样一个hfileoutputformat类，他的实现可以将数据转换成hfile

格式，通过new一个这个类，进行相关配置,这样会在hdfs下面产生一个文件，这个

时候利用hbase提供的jruby的loadtable.rb脚本就可以进行批量导入。

### ****6-13）谈谈HBase集群安装注意事项？****

需要注意的地方是ZooKeeper的配置。这与hbase-env.sh文件相关，文件中

HBASE\_MANAGES\_ZK环境变量用来设置是使用hbase默认自带的Zookeeper还

是使用独立的ZooKeeper。HBASE\_MANAGES\_ZK=false时使用独立的，为true时

使用默认自带的。

某个节点的HRegionServer启动失败，这是由于这3个节点的系统时间不一致相

差超过集群的检查时间30s。

### ****6-14)简述HBase的瓶颈****

Hbase主要的瓶颈就是传输问题，在操作时大部分的操作都是需要对磁盘操作的

### ****6-15）Redis,传统数据库,hbase,hive每个之间的区别****

[Redis](http://lib.csdn.net/base/redis)是基于内存的数据库，注重实用内存的计算，hbase是列式数据库，无法创建主键，地从是基于HDFS的，每一行可以保存很多的列，hive是数据的仓库，是为了减轻mapreduce而设计的，不是数据库是用来与红薯做交互的。

### ****6-16）Hbase的特性,以及你怎么去设计rowkey和columnFamily,怎么去建一个table****

因为hbase是列式数据库，列非表schema的一部分，所以只需要考虑rowkey和columnFamily即可，rowkey有为的相关性，最好数据库添加一个前缀，文件越小，查询速度越快，再设计列是有一个列簇，但是列簇不宜过多。

### ****6-17）Hhase与hive的区别****

ApacheHBase是一种Key/Value系统，它运行在HDFS之上。和Hive不一样，Hbase的能够在它的数据库上实时运行，而不是运行MapReduce任务。Hive被分区为表格，表格又被进一步分割为列簇。列簇必须使用schema定义，列簇将某一类型列集合起来（列不要求schema定义）。例如，“message”列簇可能包含：“to”,”from”“date”,“subject”,和”body”.每一个key/value对在Hbase中被定义为一个cell，每一个key由row-key，列簇、列和时间戳。在Hbase中，行是key/value映射的集合，这个映射通过row-key来唯一标识。Hbase利用Hadoop的基础设施，可以利用通用的设备进行水平的扩展。

### ****6-18）描述hbase的scan和get功能以及实现的异同****

HBase的查询实现只提供两种方式：1、按指定RowKey获取唯一一条记录，get方法（org.apache.hadoop.hbase.client.Get）2、按指定的条件获取一批记录，scan方法（org.apache.hadoop.hbase.client.Scan）实现条件查询功能使用的就是scan方式

### ****6-19）****[HBasescansetBatch和setCaching的区别](http://blog.csdn.net/caoli98033/article/details/44650497)

can可以通过setCaching与setBatch方法提高速度（以空间换时间），

setCaching设置的值为每次rpc的请求记录数，默认是1；cache大可以优化性能，但是太大了会花费很长的时间进行一次传输。

setBatch设置每次取的columnsize；有些row特别大，所以需要分开传给client，就是一次传一个row的几个column。

### ****6-20）hbase中cell的结构****

cell中的数据是没有类型的，全部是字节码形式存贮。

### ****6-21）hbase中region太多和region太大带来的冲突****

Hbase的region会自动split，当region太时，regio太大时分布会不均衡，同时对于大批量的代入数据建议如下：

1、还是必须让业务方对rowkey进行预分片，对业务数据rowkey进行md5或者其他的hash策略，让数据尽量随机分布而不是顺序写入。

2、随时观察region的大小，是否出现大region的情况。

## ****Flume相关****

### ****7-1）flume不采集Nginx日志，通过Logger4j采集日志，优缺点是什么？****

在nginx采集日志时无法获取session的信息，然而logger4j则可以获取session的信息，logger4j的方式比较稳定，不会宕机。缺点：不够灵活，logger4j的方式和项目结合过滤紧密，二flume的方式就比较灵活，便于插拔式比较好，不会影响项目的性能。

### ****7-2）flume和kafka采集日志区别，采集日志时中间停了，怎么记录之前的日志。****

Flume采集日志是通过流的方式直接将日志收集到存储层，而kafka试讲日志缓存在kafka

集群，待后期可以采集到存储层。Flume采集中间停了，可以采用文件的方式记录之前的日志，而kafka是采用offset(偏移量)的方式记录之前的日志。

## ****Kafka相关****

### ****8-1）kafka中怎样储存数据，哟及结构的，data.....目录下有多少个分区，每个分区的存储格式是什么样的？****

1、topic是按照“主题名-分区”存储的

2、分区个数由配置文件决定

3、每个分区下最重要的两个文件是0000000000.log和000000.index，0000000.log

以默认1G大小回滚。

## ****Spark相关****

### ****9-1）mr和spark区别，怎么理解spark-rdd****

Mr是文件方式的分布式计算框架，是将中间结果和最终结果记录在文件中，map和reduce的数据分发也是在文件中。

[Spark](http://lib.csdn.net/base/spark)是内存迭代式的计算框架，计算的中间结果可以缓存内存，也可以缓存硬盘，但是不是每一步计算都需要缓存的。

Spark-rdd是一个数据的分区记录集合，是利用内存来计算的，spark之所以快是因为有内存的模式

### ****9-2）简单描述spark的wordCount的执行过程****

[Scala](http://lib.csdn.net/base/scala)>sc.textFile("/usr/local/words.txt")

res0:org.apache.spark.rdd.RDD[String]=/usr/local/words.txtMapPartitionsRDD[1]attextFileat<console>:22

[scala](http://lib.csdn.net/base/scala)>sc.textFile("/usr/local/words.txt").flatMap(\_.split(""))

res2:org.apache.spark.rdd.RDD[String]=MapPartitionsRDD[4]atflatMapat<console>:22

scala>sc.textFile("/usr/local/words.txt").flatMap(\_.split("")).map((\_,1))

res3:org.apache.spark.rdd.RDD[(String,Int)]=MapPartitionsRDD[8]atmapat<console>:22

scala>sc.textFile("/usr/local/words.txt").flatMap(\_.split("")).map((\_,1)).reduceByKey(\_+\_)

res5:org.apache.spark.rdd.RDD[(String,Int)]=ShuffledRDD[17]atreduceByKeyat<console>:22

scala>sc.textFile("/usr/local/words.txt").flatMap(\_.split("")).map((\_,1)).reduceByKey(\_+\_).collect

res6:Array[(String,Int)]=Array((dageda,1),(xiaoli,1),(hellow,4),(xisdsd,1),(xiaozhang,1))

### ****9-3）按照需求使用spark编写一下程序****

A、当前文件a.text的格式为，请统计每个单词出现的个数

A,b,c,d

B,b,f,e

A,a,c,f

sc.textFile(“/user/local/a.text”).flatMap(\_.split(“,”)).map((\_,1)).ReduceByKey(\_+\_).collect()

或：

**package**cn.bigdata

**import**org.apache.spark.SparkConf

**import**org.apache.spark.SparkContext

**import**org.apache.spark.rdd.RDD

**object**Demo{

/\*

a,b,c,d

b,b,f,e

a,a,c,f

c,c,a,d

\*计算第四列每个元素出现的个数

\*/

**def**main(args:Array[String]):Unit={

**val**conf:SparkConf=**new**SparkConf().setAppName("demo").setMaster("local")

**val**sc:SparkContext=**new**SparkContext(conf)

**val**data:RDD[String]=sc.textFile("f://demo.txt")

//数据切分

**val**fourthData:RDD[(String,Int)]=data.map{x=>

**val**arr:Array[String]=x.split(",")

**val**fourth:String=arr(3)

(fourth,1)

}

**val**result:RDD[(String,Int)]=fourthData.reduceByKey(\_+\_);

println(result.collect().toBuffer)

}

}

B、HDFS中有两个文件a.text与b.text,文件的格式为(ip,username),如：a.text,b.text

a.text

127.0.0.1xiaozhang

127.0.0.1xiaoli

127.0.0.2wangwu

127.0.0.3lisi

B.text

127.0.0.4lixiaolu

127.0.0.5lisi

每个文件至少有1000万行，请用程序完成一下工作，

1）每个文件的个子的IP

2)出现在b.text而没有出现在a.text的IP

3)每个user出现的次数以及每个user对应的IP的个数

**代码如下：**

1）各个文件的ip数

packagecn.bigdata

importjava.util.concurrent.Executors

importorg.apache.hadoop.conf.Configuration

importorg.apache.hadoop.fs.FileSystem

importorg.apache.hadoop.fs.LocatedFileStatus

importorg.apache.hadoop.fs.Path

importorg.apache.hadoop.fs.RemoteIterator

importorg.apache.spark.SparkConf

importorg.apache.spark.SparkContext

importorg.apache.spark.rdd.RDD

importorg.apache.spark.rdd.RDD.rddToPairRDDFunctions

//各个文件的ip数

objectDemo2{

valcachedThreadPool=Executors.newCachedThreadPool()

defmain(args:Array[String]):Unit={

valconf:SparkConf=newSparkConf().setAppName("demo2").setMaster("local")

valsc:SparkContext=newSparkContext(conf)

valhdpConf:Configuration=newConfiguration

valfs:FileSystem=FileSystem.get(hdpConf)

vallistFiles:RemoteIterator[LocatedFileStatus]=fs.listFiles(newPath("f://txt/2/"),true)

while(listFiles.hasNext){

valfileStatus=listFiles.next

valpathName=fileStatus.getPath.getName

cachedThreadPool.execute(newRunnable(){

overridedefrun():Unit={

println("======================="+pathName)

analyseData(pathName,sc)

}

})

}

}

defanalyseData(pathName:String,sc:SparkContext):Unit={

valdata:RDD[String]=sc.textFile("f://txt/2/"+pathName)

valdataArr:RDD[Array[String]]=data.map(\_.split(""))

valipAndOne:RDD[(String,Int)]=dataArr.map(x=>{

valip=x(0)

(ip,1)

})

valcounts:RDD[(String,Int)]=ipAndOne.reduceByKey(\_+\_)

valsortedSort:RDD[(String,Int)]=counts.sortBy(\_.\_2,false)

sortedSort.saveAsTextFile("f://txt/3/"+pathName)

}

}

2）出现在b.txt而没有出现在a.txt的ip

packagecn.bigdata

importjava.util.concurrent.Executors

importorg.apache.spark.SparkConf

importorg.apache.spark.SparkContext

importorg.apache.spark.rdd.RDD

/\*

\*出现在b.txt而没有出现在a.txt的ip

\*/

objectDemo3{

valcachedThreadPool=Executors.newCachedThreadPool()

defmain(args:Array[String]):Unit={

valconf=newSparkConf().setAppName("Demo3").setMaster("local")

valsc=newSparkContext(conf)

valdata\_a=sc.textFile("f://txt/2/a.txt")

valdata\_b=sc.textFile("f://txt/2/b.txt")

valsplitArr\_a=data\_a.map(\_.split(""))

valip\_a:RDD[String]=splitArr\_a.map(x=>x(0))

valsplitArr\_b=data\_b.map(\_.split(""))

valip\_b:RDD[String]=splitArr\_b.map(x=>x(0))

valsubRdd:RDD[String]=ip\_b.subtract(ip\_a)

subRdd.saveAsTextFile("f://txt/4/")

}

}

3）

packagecn.bigdata

importorg.apache.spark.SparkConf

importorg.apache.spark.SparkContext

importorg.apache.spark.rdd.RDD

importscala.collection.mutable.Set

/\*

\*每个user出现的次数以及每个user对应的ip数

\*/

objectDemo4{

defmain(args:Array[String]):Unit={

valconf=newSparkConf().setAppName("Demo4").setMaster("local")

valsc=newSparkContext(conf)

valdata:RDD[String]=sc.textFile("f://txt/5/")

vallines=data.map(\_.split(""))

valuserIpOne=lines.map(x=>{

valip=x(0)

valuser=x(1)

(user,(ip,1))

})

valuserListIpCount:RDD[(String,(Set[String],Int))]=userIpOne.combineByKey(

x=>(Set(x.\_1),x.\_2),

(a:(Set[String],Int),b:(String,Int))=>{

(a.\_1+b.\_1,a.\_2+b.\_2)

},

(m:(Set[String],Int),n:(Set[String],Int))=>{

(m.\_1++n.\_1,m.\_2+n.\_2)

})

valresult:RDD[String]=userListIpCount.map(x=>{

x.\_1+":userCount:"+x.\_2.\_2+",ipCount:"+x.\_2.\_1.size

})

println(result.collect().toBuffer)

}

}

## ****Sqoop相关****

### 10-1）sqoop在导入到MySql数据库是怎样保证数据重复，如果重复了该怎么办？？

在导入时在语句的后面加上一下命令作为节点：

--incrementalappend\

--check-columnid\

--last-value1208

## ****Redis相关****

### ****10-1）redis保存磁盘的时间****

#Note:youcandisablesavingatallcommentingallthe"save"lines.

#

#Itisalsopossibletoremoveallthepreviouslyconfiguredsave

#pointsbyaddingasavedirectivewithasingleemptystringargument

#likeinthefollowingexample:

#

#save""

save9001

save30010

save6010000

## ****环境配置****

### ****1）你们的集群规模？****

这个得看个人在公司的规模，下面介绍一下我们公司的一些配置：

联想Systemx3750服务器，价格3.5万，内存容量32G，产品类型机架式，硬盘接口SSD,CPU频率2.6GH,CPU数量2颗，三级缓存15MB,cpu核心6核，cpu线程数12线程,最大内存支持1.5T,网络是千兆网卡,可插拔时硬盘接口12个卡槽,配置1T的容量

详细：http://detail.zol.com.cn/server/index1101243.shtml

名字软件运行管理

Hadoop1JDK,hadoopnamenode

Hadoop2JDK,hadoopnamenode

Hadoop3JDK,hadoopsecondaryNamenode

Hadoop4JDK,hadoopsecondaryNamenode

Hadoop5JDK,hadoopdatanode

Hadoop6JDK,hadoopdatanode

Hadoop7JDK,hadoopdatanode

Hadoop8JDK,hadoopdatanode

Hadoop9JDK,hadoopdatanode

Hadoop10JDK,zookeeper,tomcat,mvn,kafkaleader

Hadoop11JDK,zookeeper,tomcat,mvn,kafkafollower

Hadoop12JDK,zookeeper,tomcat,mvn,kafkafollower

Hadoop13JDK,hive,mysql,svn,logstarhhive,mysql,svn

Hadoop14JDK,hbase,mysql备份datanode

Hadoop15JDK,nginx,Log日志手机datanode

数据就是每天访问的Log日志不是很大，有的时候大有的时候小的可怜

### ****2) 你在项目中遇到了哪些难题，是怎么解决的？****

1、在执行任务时发现副本的个数不对，经过一番的查找发现是超时的原因，修改了配置文件hdfs-site.xml：中修改了超时时间。

2、由于当时在分配各个目录空间大小时，没有很好的分配导致有的目录的空间浪费，于是整体商量后把储存的空间调大了一些。

## 大数据算法问题

### 1、海量日志数据，提取出某日访问百度次数最多的那个IP。

解决方案：首先是将这一天，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。

### 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。

解决方案：第一步、先对这批海量数据预处理，在O(N)的时间内用Hash表完成排序;然后，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)。

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。

解决方案：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。

解决方案：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?

解决方案：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。

通读文件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。

通读文件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)。

### 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。

解决方案：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?

解决方案：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。

方案2：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02.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02.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。

### 8、怎么在海量数据中找出重复次数最多的一个?

解决方案：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前N个数据。

解决方案：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。

解决方案：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)。

**大数据的本质：从数据中挖掘价值**

**云计算的本质：共享服务**

## 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字。所以一个简单的改进就是countingBloomfilter，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-lefthashing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：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。

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

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=“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：

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).ThecanonicalexampleapplicationofMapReduceisaprocesstocounttheappearancesof

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。

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。

## 其他