大数据工程师面试题

# 选择题

## 下面哪个程序负责 HDFS 数据存储。

a)NameNode b)Jobtracker c)Datanode d)secondaryNameNode e)tasktracker

答案 C datanode

## HDfS 中的 block 默认保存几份？

a)3 份 b)2 份 c)1 份 d)不确定

答案 A 默认 3 份

## 下列哪个程序通常与NameNode在一个节点启动?

a)SecondaryNameNode b)DataNode c)TaskTracker d)Jobtracker

答案 D

## HDFS 默认 Block Size

a)32MB b)64MB c)128MB

答案：B

## 下列哪项通常是集群的最主要瓶颈

a)CPU b)网络 c)磁盘 IO d)内存

答案：C 磁盘

首先集群的目的是为了节省成本，用廉价的 pc 机，取代小型机及大型机。小型机和大型机有什么特点？

1.cpu 处理能力强

2.内存够大，所以集群的瓶颈不可能是 a 和 d

3.如果是互联网有瓶颈，可以让集群搭建内网。每次写入数据都要通过网络（集群是内网），然后还要写入 3 份数据，所以 IO 就会打折扣。

## 关于 SecondaryNameNode 哪项是正确的？

a)它是 NameNode 的热备 b)它对内存没有要求

c)它的目的是帮助 NameNode 合并编辑日志，减少 NameNode 启动时间

d)SecondaryNameNode 应与 NameNode 部署到一个节点

答案 C。

## 下列哪项可以作为集群的管理？

a)Puppet b)Pdsh c)Cloudera Manager d)Zookeeper

答案 ABD

具体可查看什么是 Zookeeper，Zookeeper 的作用是什么，在 Hadoop 及 hbase 中具体作用是什么。

## Client 端上传文件的时候下列哪项正确

a)数据经过 NameNode 传递给 DataNode

b)Client 端将文件切分为 Block，依次上传

c)Client 只上传数据到一台 DataNode，然后由 NameNode 负责 Block 复制工作

答案 B

分析：Client 向 NameNode 发起文件写入的请求。NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息。Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个DataNode 块中。具体查看HDFS 体系结构简介及优缺点。

## 下列哪个是 Hadoop 运行的模式

a)单机版 b)伪分布式 c)分布式

答案 ABC 单机版,伪分布式只是学习用的。

# 面试题

## Hadoop的核心配置是什么？

Hadoop的核心配置通过两个xml文件来完成：1，hadoop-default.xml；2，hadoop-site.xml。这些文件都使用xml格式，因此每个xml中都有一些属性，包括名称和值，但是当下这些文件都已不复存在。

## 那当下又该如何配置？

Hadoop现在拥有3个配置文件：1，core-site.xml；2，hdfs-site.xml；3，mapred-site.xml。这些文件都保存在conf/子目录下。

## “jps”命令的用处？

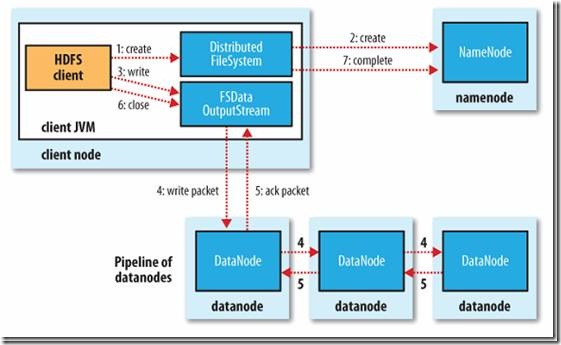
这个命令可以检查Namenode、Datanode、Task Tracker、 Job Tracker是否正常工作。

## mapreduce的原理?



## HDFS存储的机制?

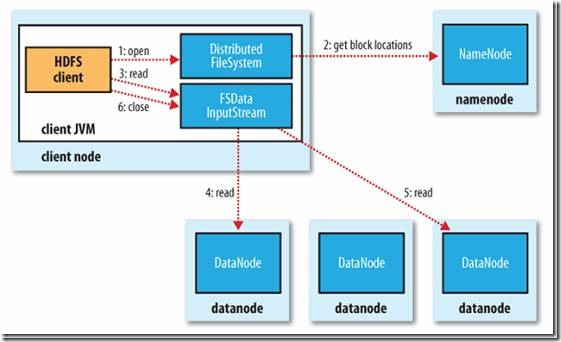
### hdfs写流程



流程：

1. client链接namenode存数据
2. namenode记录一条数据位置信息（元数据），告诉client存哪。
3. client用hdfs的api将数据块（默认是64M）存储到datanode上。
4. datanode将数据水平备份。并且备份完将反馈client。
5. client通知namenode存储块完毕。
6. namenode将元数据同步到内存中。
7. 另一块循环上面的过程。

### 读流程



流程：

1. client链接namenode，查看元数据，找到数据的存储位置。
2. client通过hdfs的api并发读取数据。
3. 关闭连接。

## 举一个简单的例子说明mapreduce是怎么来运行的 ?

wordcount的例子

## 用mapreduce来实现下面需求？

现在有10个文件夹,每个文件夹都有1000000个url.现在让你找出top1000000url。

解答：topk

(还可以用treeMap, 到1000000了每来一个都加进去, 删掉最小的)

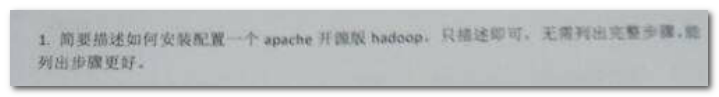
## hadoop中Combiner的作用?

combiner是reduce的实现，在map端运行计算任务，减少map端的输出数据。

作用就是优化。

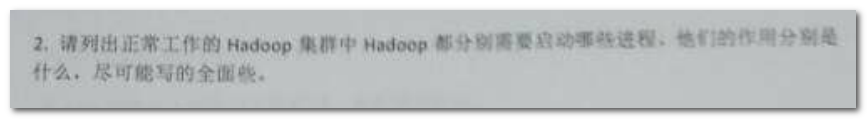
但是combiner的使用场景是mapreduce的map和reduce输入输出一样。

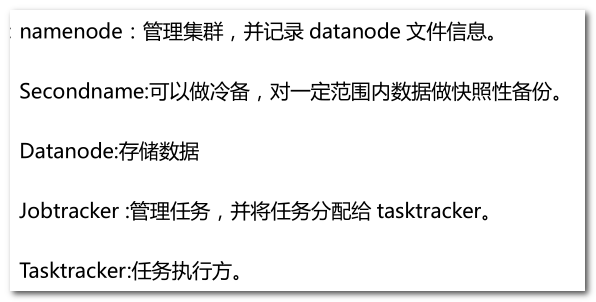
## 简述hadoop安装



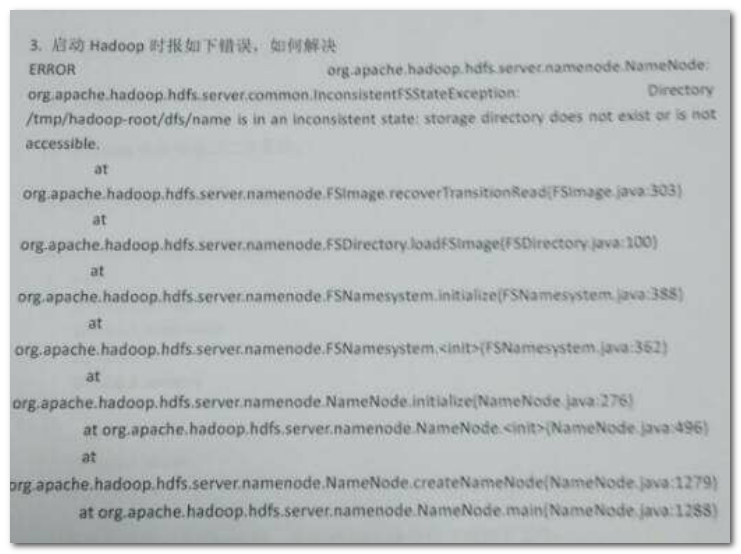


## 请列出hadoop进程名



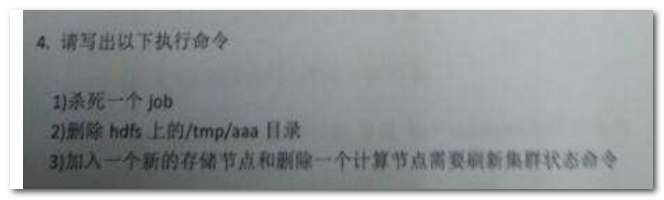


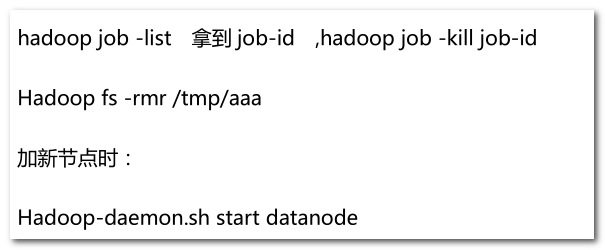
## 解决下面的错误

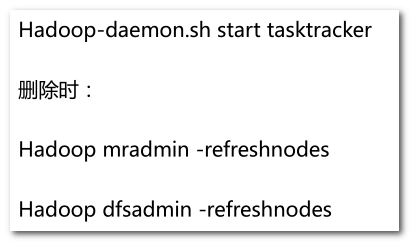


1. 权限问题，可能曾经用root启动过集群。(例如hadoop搭建的集群,是tmp/hadoop-hadoop/.....)
2. 可能是文件夹不存在
3. 解决: 删掉tmp下的那个文件,或改成当前用户

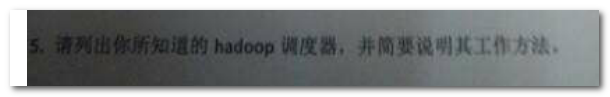
## 写出下面的命令

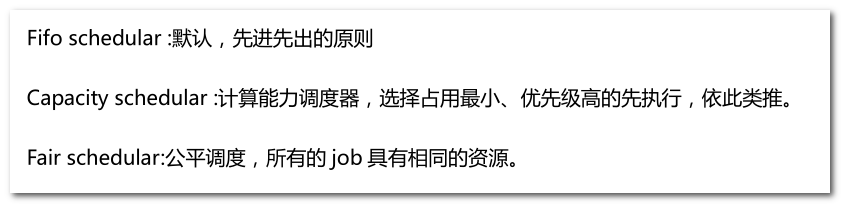






## 简述hadoop的调度器





## 列出你开发mapreduce的语言



java

## 书写程序



wordcount

## 不同语言的优缺点



hadoop是java写的，java的集成效果最好，并且平台环境统一。

## hive有哪些保存元数据的方式，个有什么特点。



1. 内存数据库derby，安装小，但是数据存在内存，不稳定
2. mysql数据库，数据存储模式可以自己设置，持久化好，查看方便。

## combiner和partition的作用



combiner是reduce的实现，在map端运行计算任务，减少map端的输出数据。

作用就是优化。

但是combiner的使用场景是mapreduce的map输出结果和reduce输入输出一样。

partition的默认实现是hashpartition，是map端将数据按照reduce个数取余，进行分区，不同的reduce来copy自己的数据。

partition的作用是将数据分到不同的reduce进行计算，加快计算效果。

## hive内部表和外部表的区别

内部表：加载数据到hive所在的hdfs目录，删除时，元数据和数据文件都删除

外部表：不加载数据到hive所在的hdfs目录，删除时，只删除表结构。

## hbase的rowkey怎么创建好？列族怎么创建比较好？

hbase存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。(位置相关性)

一个列族在数据底层是一个文件，所以将经常一起查询的列放到一个列族中，列族尽量少，减少文件的寻址时间。

## 用mapreduce怎么处理数据倾斜问题？

数据倾斜：map /reduce程序执行时，reduce节点大部分执行完毕，但是有一个或者几个reduce节点运行很慢，导致整个程序的处理时间很长，这是因为某一个key的条数比其他key多很多（有时是百倍或者千倍之多），这条key所在的reduce节点所处理的数据量比其他节点就大很多，从而导致某几个节点迟迟运行不完，此称之为数据倾斜。

用hadoop程序进行数据关联时，常碰到数据倾斜的情况，这里提供一种解决方法。

自己实现partition类，用key和value相加取hash值：

方式1：

源代码：

**public** **int** getPartition(K key, V value,

**int** numReduceTasks) {

**return** (key.hashCode() & Integer.***MAX\_VALUE***) % numReduceTasks;

}

修改后

**public** **int** getPartition(K key, V value,

**int** numReduceTasks) {

**return** ((（key).hashCode()+value.hashCode()） & Integer.***MAX\_VALUE***) % numReduceTasks;

}

方式2：

**public** **class** HashPartitioner<K, V> **extends** Partitioner<K, V> {

**private** **int** aa= 0;

/\*\* Use {@link Object#hashCode()} to partition. \*/

**public** **int** getPartition(K key, V value,

**int** numReduceTasks) {

**return** (key.hashCode()+(aa++) & Integer.***MAX\_VALUE***) % numReduceTasks;

}

## hadoop框架中怎么来优化

（1） 从应用程序角度进行优化。由于mapreduce是迭代逐行解析数据文件的，怎样在迭代的情况下，编写高效率的应用程序，是一种优化思路。

（2） 对Hadoop参数进行调优。当前hadoop系统有190多个配置参数，怎样调整这些参数，使hadoop作业运行尽可能的快，也是一种优化思路。

（3） 从系统实现角度进行优化。这种优化难度是最大的，它是从hadoop实现机制角度，发现当前Hadoop设计和实现上的缺点，然后进行源码级地修改。该方法虽难度大，但往往效果明显。

（4）linux内核参数调整

### 从应用程序角度进行优化

（1） 避免不必要的reduce任务

如果mapreduce程序中reduce是不必要的，那么我们可以在map中处理数据, Reducer设置为0。这样避免了多余的reduce任务。

（2） 为job添加一个Combiner

为job添加一个combiner可以大大减少shuffle阶段从map task拷贝给远程reduce task的数据量。一般而言，combiner与reducer相同。

（3） 根据处理数据特征使用最适合和简洁的Writable类型

Text对象使用起来很方便，但它在由数值转换到文本或是由UTF8字符串转换到文本时都是低效的，且会消耗大量的CPU时间。当处理那些非文本的数据时，可以使用二进制的Writable类型，如IntWritable， FloatWritable等。二进制writable好处：避免文件转换的消耗；使map task中间结果占用更少的空间。

（4） 重用Writable类型

很多MapReduce用户常犯的一个错误是，在一个map/reduce方法中为每个输出都创建Writable对象。例如，你的Wordcout mapper方法可能这样写：

public void map(...) {

…

for (String word : words) {

output.collect(new Text(word), new IntWritable(1));

}

}

这样会导致程序分配出成千上万个短周期的对象。Java垃圾收集器就要为此做很多的工作。更有效的写法是：

class MyMapper … {

Text wordText = new Text();

IntWritable one = new IntWritable(1);

public void map(...) {

for (String word: words) {

wordText.set(word);

output.collect(wordText, one);

}

}

}

（5） 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。

### 对参数进行调优

**查看linux的服务，可以关闭不必要的服务**

ntsysv

**停止打印服务**

#/etc/init.d/cups stop

#chkconfig cups off

**关闭ipv6**

#vim /etc/modprobe.conf

添加内容

alias net-pf-10 off

alias ipv6 off

**调整文件最大打开数**

查看： ulimit -a    结果：open files (-n) 1024

临时修改： ulimit -n 4096

持久修改：

vi /etc/security/limits.conf在文件最后加上：

**\* soft nofile 65535  
\* hard nofile 65535  
\* soft nproc 65535  
\* hard nproc 65535**

**修改linux内核参数**

vi /etc/sysctl.conf

添加

net.core.somaxconn = 32768

#web应用中listen函数的backlog默认会给我们内核参数的net.core.somaxconn限制到128，而nginx定义的NGX\_LISTEN\_BACKLOG默认为511，所以有必要调整这个值。

**调整swap分区什么时候使用：**

查看：cat /proc/sys/vm/swappiness

设置：vi /etc/sysctl.conf

            在这个文档的最后加上这样一行: vm.swappiness=10

            表示物理内存使用到90%（100-10=90）的时候才使用swap交换区

**关闭noatime**

vi /etc/fstab

/dev/sda2    /data     ext3  noatime,nodiratime  0 0

**设置readahead buffer**

blockdev --setra READAHEAD 512 /dev/sda

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>  #++++maptask的最大数

            <value>2</value>

        </property>

     <property>

            <name>mapred.tasktracker.reduce.tasks.maximum</name>  #++++reducetask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

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RPC hander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>

**启用推测执行机制**

推测执行(Speculative Execution)是指在分布式集群环境下，因为程序BUG，负载不均衡或者资源分布不均等原因，造成同一个job的多个task运行速度不一致，有的task运行速度明显慢于其他task（比如：一个job的某个task进度只有10%，而其他所有task已经运行完毕），则这些task拖慢了作业的整体执行进度，为了避免这种情况发生，Hadoop会为该task启动备份任务，让该speculative task与原始task同时处理一份数据，哪个先运行完，则将谁的结果作为最终结果。

推测执行优化机制采用了典型的以空间换时间的优化策略，它同时启动多个相同task（备份任务）处理相同的数据块，哪个完成的早，则采用哪个task的结果，这样可防止拖后腿Task任务出现，进而提高作业计算速度，但是，这样却会占用更多的资源，在集群资源紧缺的情况下，设计合理的推测执行机制可在多用少量资源情况下，减少大作业的计算时间。

mapred.map.tasks.speculative.execution  默认是true

mapred.rduce.tasks.speculative.execution  默认是true

**设置是失败容忍度**

mapred.max.map.failures.percent   作业允许失败的map最大比例  默认值0，即0%

mapred.max.reduce.failures.percent  作业允许失败的reduce最大比例  默认值0，即0%

mapred.map.max.attemps  失败后最多重新尝试的次数 默认是4

mapred.reduce.max.attemps  失败后最多重新尝试的次数 默认是4

**启动jvm重用功能**

mapred.job.reuse.jvm.num.tasks  默认值1，表示只能启动一个task，若为-1，表示可以最多运行数不限制

**设置任务超时时间**

mapred.task.timeout  默认值600000毫秒，也就是10分钟。

**合理的控制reduce的启动时间**

mapred.reduce.slowstart.completed.maps  默认值0.05  表示map任务完成5%时，开始启动reduce任务

**跳过坏记录**

 当任务失败次数达到该值时，才会进入skip mode，即启用跳过坏记录数功能,也就是先试几次，不行就跳过

mapred.skip.attempts.to.start.skipping 默认值 2

map最多允许跳过的记录数

mapred.skip.map.max.skip.records 默认值0，为不启用

reduce最多允许跳过的记录数

mapred.skip.reduce.max.skip.records 默认值0，为不启用

换记录存放的目录

mapred.skip.out.dir  默认值${mapred.output.dir}/\_logs/

## 我们开发job时，是否可以去掉reduce阶段。

可以。设置reduce数为0 即可。

## datanode在什么情况下不会备份

datanode在强制关闭或者非正常断电不会备份。

## combiner出现在那个过程

出现在map阶段的map方法后。

## hdfs的体系结构

hdfs有namenode、secondraynamenode、datanode组成。

为n+1模式

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个datanode中有一个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。

## 描述一下hadoop中，有哪些地方使用了缓存机制，作用分别是什么？

在mapreduce提交job的获取id之后，会将所有文件存储到分布式缓存上，这样文件可以被所有的mapreduce共享。

## 如何确定hadoop集群的健康状态

通过页面监控,脚本监控。

## 生产环境中为什么建议使用外部表？

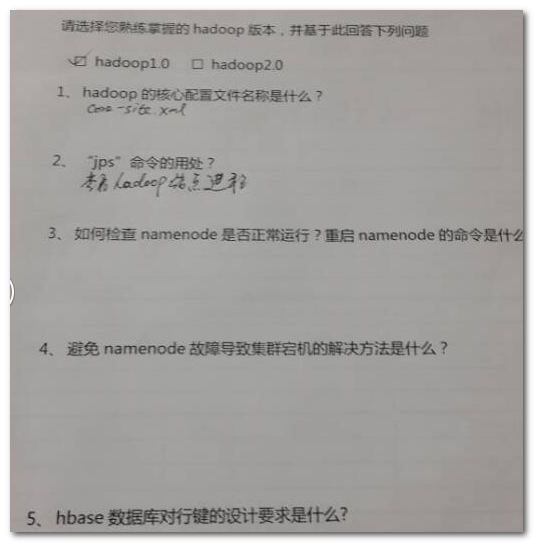
1、因为外部表不会加载数据到hive，减少数据传输、数据还能共享。

2、hive不会修改数据，所以无需担心数据的损坏

1. 删除表时，只删除表结构、不删除数据。

# 15期新增

## 新增

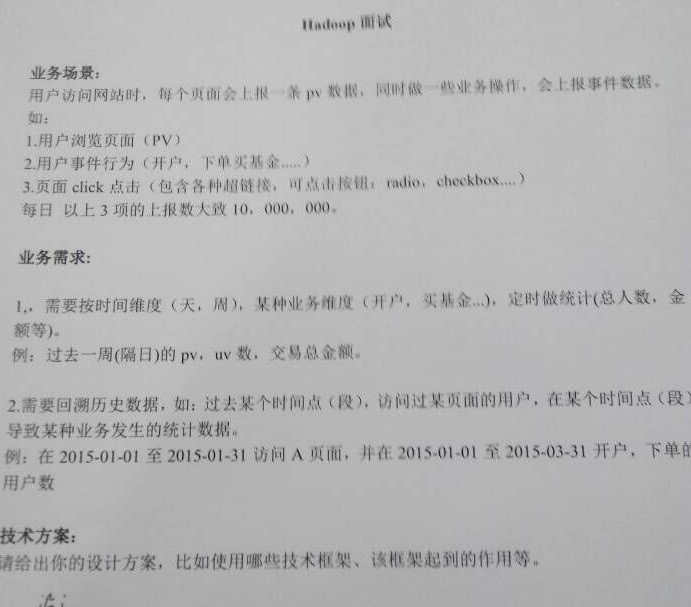


1. 通过节点信息和浏览器查看，通过脚本监控

hadoop-deamon.sh start namenode

hdfs-deamon.sh start namenode

1. 自己书写脚本监控重启
2. 行健以字典序排列，设计时充分利用这个特点，将经常一起查询的行健设计在一起，例如时间戳结尾，用户名开头（位置相关性）



1. 用hive分析业务数据即可
2. 将数据导入到hive中

sql的设计思路：多表关联

1. 找到所有在2015-01-01到2015-01-31时间内访问A页面的用户
2. 在这些用户中删选在2015-01-01到2015-03-31下单的用户
3. 统计总数

## 你们数据库怎么导入hive 的,有没有出现问题

在导入hive的时候，如果数据库中有blob或者text字段，会报错，解决方案在sqoop笔记中

## 公司技术选型可能利用storm 进行实时计算,讲解一下storm

描述下storm的设计模式，是基于work、excutor、task的方式运行代码，由spout、bolt组成等等

## 一个datanode 宕机,怎么一个流程恢复

将datanode数据删除，重新当成新节点加入即可。

## Hbase 的特性,以及你怎么去设计 rowkey 和 columnFamily ,怎么去建一个table

hbase是列式数据库，rowkey是字典序的，设计时的规则同上。

每个列族是一个文件，将经常一起查询的列放到同一个列族中，减少文件的寻址时间。

## Redis,传统数据库,hbase,hive 每个之间的区别

redis：分布式缓存，强调缓存，内存中数据

传统数据库：注重关系

hbase：列式数据库，无法做关系数据库的主外键，用于存储海量数据，底层基于hdfs

hive：数据仓库工具，底层是mapreduce。不是数据库，不能用来做用户的交互存储

## shuffle 阶段,你怎么理解的

shuffle的过程说清楚，目的说清楚

## Mapreduce 的 map 数量 和 reduce 数量 怎么确定 ,怎么配置

map的数量有数据块决定，reduce数量随便配置。

## 唯一难住我的是他说实时计算,storm 如果碰上了复杂逻辑,需要算很长的时间,你怎么去优化,怎么保证实时性

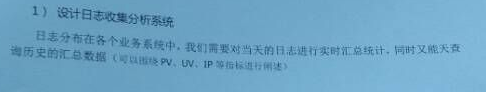
## Hive 你们用的是外部表还是内部表,有没有写过UDF,hive 的版本

外部表和内部表的区别

## Hadoop 的版本

1.04、1.20都为稳定版，是两个常用的hadoop1版本。

## 实时流式计算 的结果内容有哪些,你们需要统计出来么

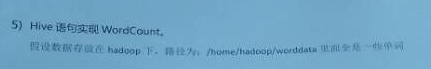


1. 通过flume将不同系统的日志收集到kafka中
2. 通过storm实时的处理PV、UV、IP
3. 通过kafka的consumer将日志生产到hbase中。
4. 通过离线的mapreduce或者hive，处理hbase中的数据



大体分为3个部分:

1. 离线hadoop技术分享（mapreduce、hive）
2. nosql数据库hbase分享
3. 实时流计算分享



1. 建表
2. 分组（group by）统计wordcount

select word,count(1) from table1 group by word;



可以估计每个文件的大小为50亿×64=298G，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。

1. 将文件存储到hdfs中，这样每个文件为64M或者是128M
2. 分别对两个文件的url进行去重、排序输出，这样能排除a文件中相同的url，b文件也一样
3. 对a、b两个文件处理后的结果进行wordcount，并且在reduce中判断单词个数，个数为2的时候输出，这样就找到了a、b文件中的相同url。
4. 此计算步骤中的每一步加载到内存中的文件大小都不会超过64M，远远小于4G。



topk，强调使用treemap是为了节省内存计算空间。  


flume：日志收集系统，主要用于系统日志的收集

kafka：消息队列，进行消息的缓存和系统的解耦

storm：实时计算框架，进行流式的计算。



简单地说，就是一个变量和常量的关系。StringBuffer对象的内容可以修改；而String对象一旦产生后就不可以被修改，重新赋值其实是两个对象。

StringBuilder：线程非安全的

StringBuffer：线程安全的

　　　　当我们在字符串缓冲去被多个线程使用是，JVM不能保证StringBuilder的操作是安全的，虽然他的速度最快，但是可以保证StringBuffer是可以正确操作的。当然大多数情况下就是我们是在单线程下进行的操作，所以大多数情况下是建议用StringBuilder而不用StringBuffer的，就是速度的原因。



1 HashMap不是线程安全的

            hastmap是一个接口 是map接口的子接口，是将键映射到值的对象，其中键和值都是对象，并且不能包含重复键，但可以包含重复值。HashMap允许null key和null value，而hashtable不允许。

2   HashTable是线程安全的一个Collection。

HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于Hashtable。 HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。 HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。 Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。 最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。 Hashtable和HashMap采用的hash/rehash算法都大概一样，所以性能不会有很大的差

public static void main(String args[]) { HashTable h=new HashTable(); h.put("用户1",new Integer(90)); h.put("用户2",new Integer(50)); h.put("用户3",new Integer(60)); h.put("用户4",new Integer(70)); h.put("用户5",new Integer(80)); Enumeration e=h.elements(); while(e.hasMoreElements()){ System.out.println(e.nextElement()); }

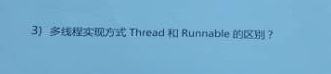
总结：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| hashmap | 线程不安全 | 允许有null的键和值 | 效率高一点、 | 方法不是Synchronize的要提供外同步 | 有containsvalue和containsKey方法 | HashMap 是Java1.2 引进的Map interface 的一个实现 | HashMap是Hashtable的轻量级实现 |
| hashtable | 线程安全 | 不允许有null的键和值 | 效率稍低、 | 方法是是Synchronize的 | 有contains方法方法 | 、Hashtable 继承于Dictionary 类 | Hashtable 比HashMap 要旧 |

**Vector & ArrayList**1）  Vector的方法都是同步的(Synchronized),是线程安全的(thread-safe)，而ArrayList的方法不是，由于线程的同步必然要影响性能，因此,ArrayList的性能比Vector好。   
2） 当Vector或ArrayList中的元素超过它的初始大小时,Vector会将它的容量翻倍,而ArrayList只增加50%的大小，这样,ArrayList就有利于节约内存空间。

**linkedlist & ArrayList**

ArrayList 采用的是数组形式来保存对象的，这种方式将对象放在连续的位置中，所以最大的缺点就是插入删除时非常麻烦  
LinkedList 采用的将对象存放在独立的空间中，而且在每个空间中还保存下一个链接的索引 但是缺点就是查找非常麻烦 要丛第一个索引开始  
  
Hashtable和HashMap类有三个重要的不同之处。第一个不同主要是历史原因。Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现。   
  
也许最重要的不同是Hashtable的方法是同步的，而HashMap的方法不是。这就意味着，虽然你可以不用采取任何特殊的行为就可以在一个多线程的应用程序中用一个Hashtable，但你必须同样地为一个HashMap提供外同步。一个方便的方法就是利用Collections类的静态的synchronizedMap()方法，它创建一个[线程安全](https://www.baidu.com/s?wd=%E7%BA%BF%E7%A8%8B%E5%AE%89%E5%85%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nWn3mvw9mHb4P1Khnj630ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1TvnWmsnH0z" \t "_blank)的Map对象，并把它作为一个封装的对象来返回。这个对象的方法可以让你同步访问潜在的HashMap。这么做的结果就是当你不需要同步时，你不能切断Hashtable中的同步（比如在一个单线程的应用程序中），而且同步增加了很多处理费用。   
  
第三点不同是，只有HashMap可以让你将空值作为一个表的条目的key或value。HashMap中只有一条记录可以是一个空的key，但任意数量的条目可以是空的value。这就是说，如果在表中没有发现搜索键，或者如果发现了搜索键，但它是一个空的值，那么get()将返回null。如果有必要，用containKey()方法来区别这两种情况。   
  
一些资料建议，当需要同步时，用Hashtable，反之用HashMap。但是，因为在需要时，HashMap可以被同步，HashMap的功能比Hashtable的功能更多，而且它不是基于一个陈旧的类的，所以有人认为，在各种情况下，HashMap都优先于Hashtable。   
  
关于Properties   
有时侯，你可能想用一个hashtable来映射key的字符串到value的字符串。[DOS](https://www.baidu.com/s?wd=DOS&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nWn3mvw9mHb4P1Khnj630ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1TvnWmsnH0z" \t "_blank)、Windows和Unix中的环境字符串就有一些例子，如key的字符串[PATH](https://www.baidu.com/s?wd=PATH&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nWn3mvw9mHb4P1Khnj630ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1TvnWmsnH0z" \t "_blank)被映射到value的字符串C:\WINDOWS;C:\WINDOWS\[SYSTEM](https://www.baidu.com/s?wd=SYSTEM&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nWn3mvw9mHb4P1Khnj630ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1TvnWmsnH0z" \t "_blank)。Hashtables是表示这些的一个简单的方法，但Java提供了另外一种方法。   
  
Java.util.Properties类是Hashtable的一个子类，设计用于String keys和values。Properties对象的用法同Hashtable的用法相象，但是类增加了两个节省时间的方法，你应该知道。   
  
Store()方法把一个Properties对象的内容以一种可读的形式保存到一个文件中。Load()方法正好相反，用来读取文件，并设定Properties对象来包含keys和values。   
  
注意，因为Properties扩展了Hashtable，你可以用超类的put()方法来添加不是String对象的keys和values。这是不可取的。另外，如果你将store()用于一个不包含String对象的Properties对象，store()将失败。作为put()和get()的替代，你应该用setProperty()和getProperty()，它们用String参数。



在java中可有两种方式实现多线程，一种是继承Thread类，一种是实现Runnable接口；Thread类是在java.lang包中定义的。一个类只要继承了Thread类同时覆写了本类中的run()方法就可以实现多线程操作了，但是一个类只能继承一个父类，这是此方法的局限。

AD：

在java中可有两种方式实现多线程，一种是继承Thread类，一种是实现Runnable接口；Thread类是在java.lang包中定义的。一个类只要继承了Thread类同时覆写了本类中的run()方法就可以实现多线程操作了，但是一个类只能继承一个父类，这是此方法的局限。

下面看例子：

1. **package** org.thread.demo;
2. **class** MyThread **extends** Thread{
3. **private** String name;
4. **public** MyThread(String name) {
5. **super**();
6. **this**.name = name;
7. }
8. **public** **void** run(){
9. **for**(**int** i=0;i<10;i++){
10. System.out.println("线程开始："+**this**.name+",i="+i);
11. }
12. }
13. }
14. **package** org.thread.demo;
15. **public** **class** ThreadDemo01 {
16. **public** **static** **void** main(String[] args) {
17. MyThread mt1=**new** MyThread("线程a");
18. MyThread mt2=**new** MyThread("线程b");
19. mt1.run();
20. mt2.run();
21. }
22. }

但是，此时结果很有规律，先第一个对象执行，然后第二个对象执行，并没有相互运行。在JDK的文档中可以发现，一旦调用start()方法，则会通过JVM找到run()方法。下面启动start()方法启动线程：

1. **package** org.thread.demo;
2. **public** **class** ThreadDemo01 {
3. **public** **static** **void** main(String[] args) {
4. MyThread mt1=**new** MyThread("线程a");
5. MyThread mt2=**new** MyThread("线程b");
6. mt1.start();
7. mt2.start();
8. }
9. };

这样程序可以正常完成交互式运行。那么为啥非要使用start();方法启动多线程呢？

在JDK的安装路径下，src.zip是全部的java源程序，通过此代码找到Thread中的start()方法的定义，可以发现此方法中使用了private native void start0();其中native关键字表示可以调用操作系统的底层函数，那么这样的技术成为JNI技术（java Native Interface）

**Runnable接口**

在实际开发中一个多线程的操作很少使用Thread类，而是通过Runnable接口完成。

1. **public** **interface** Runnable{
2. **public** **void** run();
3. }

例子：

1. **package** org.runnable.demo;
2. **class** MyThread **implements** Runnable{
3. **private** String name;
4. **public** MyThread(String name) {
5. **this**.name = name;
6. }
7. **public** **void** run(){
8. **for**(**int** i=0;i<100;i++){
9. System.out.println("线程开始："+**this**.name+",i="+i);
10. }
11. }
12. };

但是在使用Runnable定义的子类中没有start()方法，只有Thread类中才有。此时观察Thread类，有一个构造方法：public Thread(Runnable targer)此构造方法接受Runnable的子类实例，也就是说可以通过Thread类来启动Runnable实现的多线程。（start()可以协调系统的资源）：

1. **package** org.runnable.demo;
2. **import** org.runnable.demo.MyThread;
3. **public** **class** ThreadDemo01 {
4. **public** **static** **void** main(String[] args) {
5. MyThread mt1=**new** MyThread("线程a");
6. MyThread mt2=**new** MyThread("线程b");
7. **new** Thread(mt1).start();
8. **new** Thread(mt2).start();
9. }
10. }

**两种实现方式的区别和联系：**

在程序开发中只要是多线程肯定永远以实现Runnable接口为主，因为实现Runnable接口相比继承Thread类有如下好处：

* 避免点继承的局限，一个类可以继承多个接口。
* 适合于资源的共享

以卖票程序为例，通过Thread类完成：

1. **package** org.demo.dff;
2. **class** MyThread **extends** Thread{
3. **private** **int** ticket=10;
4. **public** **void** run(){
5. **for**(**int** i=0;i<20;i++){
6. **if**(**this**.ticket>0){
7. System.out.println("卖票：ticket"+**this**.ticket--);
8. }
9. }
10. }
11. };

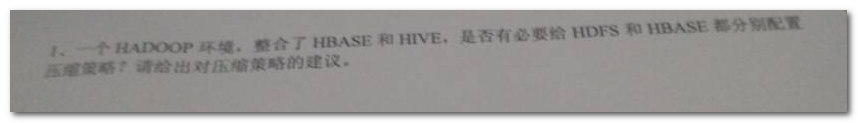
下面通过三个线程对象，同时卖票：

1. **package** org.demo.dff;
2. **public** **class** ThreadTicket {
3. **public** **static** **void** main(String[] args) {
4. MyThread mt1=**new** MyThread();
5. MyThread mt2=**new** MyThread();
6. MyThread mt3=**new** MyThread();
7. mt1.start();//每个线程都各卖了10张，共卖了30张票
8. mt2.start();//但实际只有10张票，每个线程都卖自己的票
9. mt3.start();//没有达到资源共享
10. }
11. }

如果用Runnable就可以实现资源共享，下面看例子：

1. **package** org.demo.runnable;
2. **class** MyThread **implements** Runnable{
3. **private** **int** ticket=10;
4. **public** **void** run(){
5. **for**(**int** i=0;i<20;i++){
6. **if**(**this**.ticket>0){
7. System.out.println("卖票：ticket"+**this**.ticket--);
8. }
9. }
10. }
11. }
12. **package** org.demo.runnable;
13. **public** **class** RunnableTicket {
14. **public** **static** **void** main(String[] args) {
15. MyThread mt=**new** MyThread();
16. **new** Thread(mt).start();//同一个mt，但是在Thread中就不可以，如果用同一
17. **new** Thread(mt).start();//个实例化对象mt，就会出现异常
18. **new** Thread(mt).start();
19. }
20. };

虽然现在程序中有三个线程，但是一共卖了10张票，也就是说使用Runnable实现多线程可以达到资源共享目的。



hdfs在存储的时候不会将数据进行压缩，如果想进行压缩，我们可以在向hdfs上传数据的时候进行压缩。

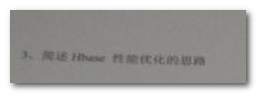
* 1. 采用压缩流

|  |
| --- |
| //压缩文件  public static void compress(String codecClassName) throws Exception{  Class<?> codecClass = Class.forName(codecClassName);  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(conf);  CompressionCodec codec = (CompressionCodec)ReflectionUtils.newInstance(codecClass, conf);  //指定压缩文件路径  FSDataOutputStream outputStream = fs.create(new Path("/user/hadoop/text.gz"));  //指定要被压缩的文件路径  FSDataInputStream in = fs.open(new Path("/user/hadoop/aa.txt"));  //创建压缩输出流  CompressionOutputStream out = codec.createOutputStream(outputStream);  IOUtils.copyBytes(in, out, conf);  IOUtils.closeStream(in);  IOUtils.closeStream(out);  } |

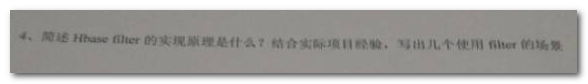
* 1. 采用序列化文件

|  |
| --- |
| public void testSeqWrite() throws Exception {  Configuration conf = new Configuration();// 创建配置信息  conf.set("fs.default.name", "hdfs://master:9000");// hdfs默认路径  conf.set("hadoop.job.ugi", "hadoop,hadoop");// 用户和组信息  String uriin = "hdfs://master:9000/ceshi2/";// 文件路径  FileSystem fs = FileSystem.get(URI.create(uriin), conf);// 创建filesystem  Path path = new Path("hdfs://master:9000/ceshi3/test.seq");// 文件名  IntWritable k = new IntWritable();// key，相当于int  Text v = new Text();// value，相当于String  SequenceFile.Writer w = SequenceFile.createWriter(fs, conf, path,  k.getClass(), v.getClass());// 创建writer  for (int i = 1; i < 100; i++) {// 循环添加  k.set(i);  v.set("abcd");  w.append(k, v);  }  w.close();  IOUtils.closeStream(w);// 关闭的时候flush  fs.close();  } |

hbase为列存数据库，本身存在压缩机制，所以无需设计。

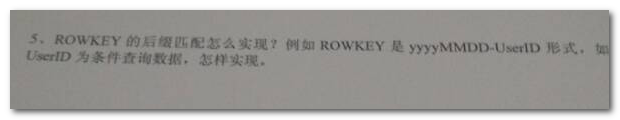


* 1. 在库表设计的时候，尽量考虑rowkey和columnfamily的特性
  2. 进行hbase集群的调优：见hbase调优

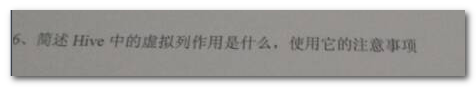


hbase的filter是通过scan设置的，所以是基于scan的查询结果进行过滤。

1. 在进行订单开发的时候，我们使用rowkeyfilter过滤出某个用户的所有订单
2. 在进行云笔记开发时，我们使用rowkey过滤器进行redis数据的恢复。



使用rowkey过滤器实现



Hive提供了三个虚拟列：

INPUT\_\_FILE\_\_NAME

BLOCK\_\_OFFSET\_\_INSIDE\_\_FILE

ROW\_\_OFFSET\_\_INSIDE\_\_BLOCK

但ROW\_\_OFFSET\_\_INSIDE\_\_BLOCK默认是不可用的，需要设置hive.exec.rowoffset为true才可以。可以用来排查有问题的输入数据。

INPUT\_\_FILE\_\_NAME, mapper任务的输出文件名。

BLOCK\_\_OFFSET\_\_INSIDE\_\_FILE, 当前全局文件的偏移量。对于块压缩文件，就是当前块的文件偏移量，即当前块的第一个字节在文件中的偏移量。

hive> SELECT INPUT\_\_FILE\_\_NAME, BLOCK\_\_OFFSET\_\_INSIDE\_\_FILE, line

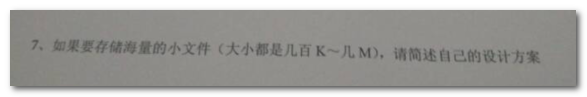
> FROM hive\_text WHERE line LIKE '%hive%' LIMIT 2;

har://file/user/hive/warehouse/hive\_text/folder=docs/

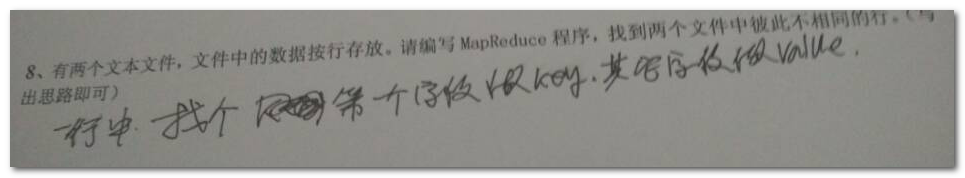
data.har/user/hive/warehouse/hive\_text/folder=docs/README.txt 2243

har://file/user/hive/warehouse/hive\_text/folder=docs/

data.har/user/hive/warehouse/hive\_text/folder=docs/README.txt 3646



* 1. 将小文件打成har文件存储
  2. 将小文件序列化到hdfs中



写个mapreduce链 用依赖关系，一共三个mapreduce，第一个处理第一个文件，第二个处理第二个文件，第三个处理前两个的输出结果，第一个mapreduce将文件去重，第二个mapreduce也将文件去重，第三个做wordcount，wordcount为1的结果就是不同的

# 共同朋友



思路：例如A，他的朋友是B\C\D\E\F\，那么BC的共同朋友就是A。所以将BC作为key，将A作为value，在map端输出即可！其他的朋友循环处理。

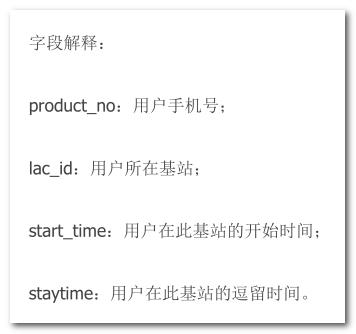
|  |
| --- |
| import java.io.IOException;  2.  import java.util.Set;  3.  import java.util.StringTokenizer;  4.  import java.util.TreeSet;  5.    6.  import org.apache.hadoop.conf.Configuration;  7.  import org.apache.hadoop.fs.Path;  8.  import org.apache.hadoop.io.Text;  9.  import org.apache.hadoop.mapreduce.Job;  10. import org.apache.hadoop.mapreduce.Mapper;  11. import org.apache.hadoop.mapreduce.Reducer;  12. import org.apache.hadoop.mapreduce.Mapper.Context;  13. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  14. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  15. import org.apache.hadoop.util.GenericOptionsParser;  16.  17. public class FindFriend {  www.aboutyun.com  18.  19. public static class ChangeMapper extends Mapper<Object, Text, Text,  Text>{  20. @Override  21. public void map(Object key, Text value, Context context) throws  IOException, InterruptedException {  22. StringTokenizer itr = new StringTokenizer(value.toString());  23. Text owner = new Text();  24. Set<String> set = new TreeSet<String>();  25. owner.set(itr.nextToken());  26. while (itr.hasMoreTokens()) {  27. set.add(itr.nextToken());  28. }  29. String[] friends = new String[set.size()];  30. friends = set.toArray(friends);  31.  32. for(int i=0;i<friends.length;i++){  33. for(int j=i+1;j<friends.length;j++){  34. String outputkey = friends[i]+friends[j];  35. context.write(new Text(outputkey),owner);  36. }  37. }  38. }  39. }  40.  41. public static class FindReducer extends Reducer<Text,Text,Text,Text>  {  42. public void reduce(Text key, Iterable<Text> values,  43. Context context) throws IOException,  InterruptedException {  44. String commonfriends ="";  www.aboutyun.com  45. for (Text val : values) {  46. if(commonfriends == ""){  47. commonfriends = val.toString();  48. }else{  49. commonfriends =  commonfriends+":"+val.toString();  50. }  51. }  52. context.write(key, new  Text(commonfriends));  53. }  54. }  55.  56.  57. public static void main(String[] args) throws IOException,  58. InterruptedException, ClassNotFoundException {  59.  60. Configuration conf = new Configuration();  61. String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  62. if (otherArgs.length < 2) {  63. System.err.println("args error");  64. System.exit(2);  65. }  66. Job job = new Job(conf, "word count");  67. job.setJarByClass(FindFriend.class);  68. job.setMapperClass(ChangeMapper.class);  69. job.setCombinerClass(FindReducer.class);  70. job.setReducerClass(FindReducer.class);  71. job.setOutputKeyClass(Text.class);  72. job.setOutputValueClass(Text.class);  73. for (int i = 0; i < otherArgs.length - 1; ++i) {  www.aboutyun.com  74. FileInputFormat.addInputPath(job, new Path(otherArgs[i]));  75. }  76. FileOutputFormat.setOutputPath(job,  77. new Path(otherArgs[otherArgs.length - 1]));  78. System.exit(job.waitForCompletion(true) ? 0 : 1);  79.  80. }  81.  82. } |

结果：

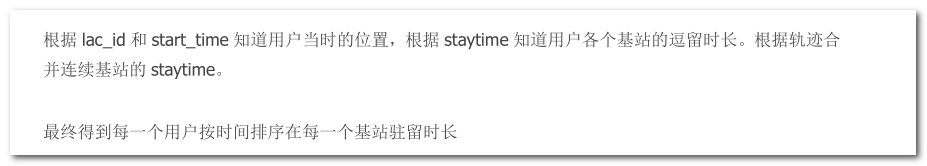
|  |
| --- |
| 1. AB E:C:D  2. AC E:B  3. AD B:E  4. AE C:B:D  5. BC A:E  6. BD A:E  7. BE C:D:A  8. BF A  9. CD E:A:B  10. CE A:B  11. CF A  12. DE B:A  13. DF A  14. EF A |

# 基站逗留时间

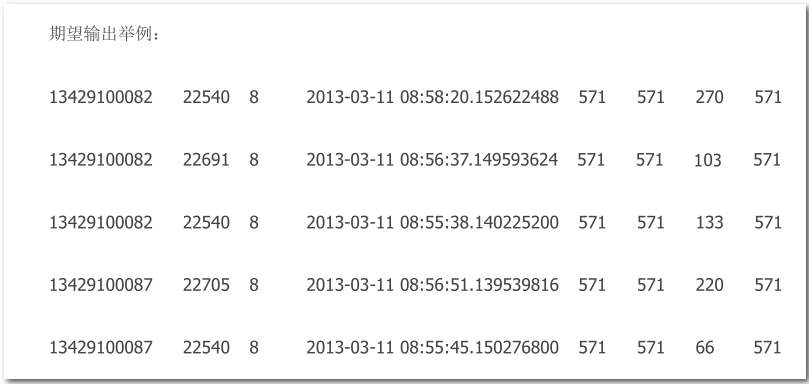




需求：



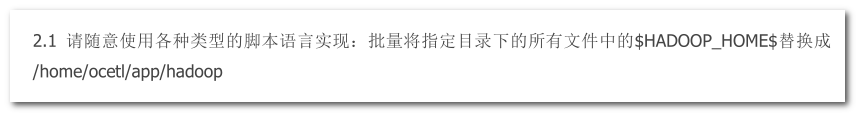
期望：



思路：

将数据导入hive表中，查询时，用电话号码和时间排序即可！

# 脚本替换



脚本：随意命名为aaa.sh

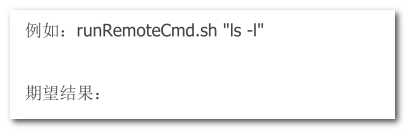
|  |
| --- |
| #!/bin/bash  ls $1 | while read line  do  sed -i 's,\$HADOOP\_HOME\$,\/home\/aa,g' $1$line  echo $1$line  done |

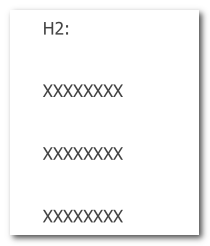
脚本执行命令：替换/home/hadoop/test/下的所有文件

./aaa.sh /home/hadoop/test/

# 一键执行







脚本：

vi runRemoteCmd.sh

|  |
| --- |
| #!/bin/bash  $1  ssh -q hadoop@slave1 "$1"  ssh -q hadoop@slave2 "$1" |

执行命令

|  |
| --- |
| ./runRemoteCmd.sh "ls -l" |

# 大数据面试汇总

1. 讲解一下MapReduce 的一些基本流程

任务提交流程，任务运行流程

1. 你们数据库怎么导入hive 的,有没有出现问题

使用sqoop导入，我们公司的数据库中设计了text字段，导致导入的时候出现了缓存不够的情况（见云笔记），开始解决起来感觉很棘手，后来查看了sqoop的文档，加上了limit属性，解决了。

1. 公司技术选型可能利用storm 进行实时计算,讲解一下storm

从storm的应用，代码书写，运行机制讲

1. 问你java 集合类的数据结构,比如hashmap

看java面试宝典

1. 问你知不知道concurrent 包下的东西,例如concurrenthashmap

看java面试宝典

1. 公司最近主要在自然语言学习去开发,有没有接触过

没有用过

# 面试问题:

1. 从前到后从你教育背景(学过哪些课)到各个项目你负责的模块,问的很细(本以为他是物理学博士,但是所有的技术都懂)
2. hadoop 的 namenode 宕机,怎么解决

先分析宕机后的损失，宕机后直接导致client无法访问，内存中的元数据丢失，但是硬盘中的元数据应该还存在，如果只是节点挂了，重启即可，如果是机器挂了，重启机器后看节点是否能重启，不能重启就要找到原因修复了。但是最终的解决方案应该是在设计集群的初期就考虑到这个问题，做namenode的HA。

1. 一个datanode 宕机,怎么一个流程恢复

Datanode宕机了后，如果是短暂的宕机，可以实现写好脚本监控，将它启动起来。如果是长时间宕机了，那么datanode上的数据应该已经被备份到其他机器了，那这台datanode就是一台新的datanode了，删除他的所有数据文件和状态文件，重新启动。

1. 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，所以在设计初期只需要考虑rowkey 和 columnFamily即可，rowkey有位置相关性，所以如果数据是练习查询的，最好对同类数据加一个前缀，而每个columnFamily实际上在底层是一个文件，那么文件越小，查询越快，所以讲经常一起查询的列设计到一个列簇，但是列簇不宜过多。

1. Redis,传统数据库,hbase,hive 每个之间的区别(问的非常细)

Redis是缓存，围绕着内存和缓存说

Hbase是列式数据库，存在hdfs上，围绕着数据量来说

Hive是数据仓库，是用来分析数据的，不是增删改查数据的。

1. 公司之后倾向用spark 开发,你会么(就用java代码去写)

会，spark使用scala开发的，在scala中可以随意使用jdk的类库，可以用java开发，但是最好用原生的scala开发，兼容性好，scala更灵活。

# 面试问题:

1. 笔试: java基础(基本全忘,做的很烂,复习大数据连单例都忘了怎么写)

复习java面试宝典

1. 开始介绍项目,直接用大数据项目介绍,项目经理也懂大数据
2. Mapreduce 一些流程,经过哪些步骤

Map—combiner—partition—sort—copy—sort—grouping—reduce

1. 说下对hadoop 的一些理解,包括哪些组件

详谈hadoop的应用，包括的组件分为三类，分别说明hdfs，yarn，mapreduce

1. 详细讲解下你流式实时计算的项目部署以及收集的结果情况

讲解storm集群的部署方案，项目的大小，使用的worker数，数据收集在hbase或者hdfs，好处是什么

1. 你的数据库是不是很大么,有没有分表,分区,你是怎么实现的

数据库的分表在设计初期是按照月份进行拆分的，不同的月份查询不同的表。分区没弄过。

1. 开始问java的一些东西(从各种框架原理到各种复杂SQL)
2. 多线程,并发,垃圾回收机制,数据结构(问这些,基本觉得看你是不是高级程序员了)

多线程要知道操作方式，线程安全的锁，并且要知道lock锁

垃圾回收机制需要详细了解（见云笔记），主要从内存划分，垃圾回收主要的工作区域，垃圾回收器的种类，各有什么优缺点，用在哪里合适。

数据结构基本的要知道，复杂的参考相关的书籍。

# 面试问题:

1. BI小组的3个年轻学生一起技术面试(一个是南开博士
2. 数据量多少,集群规模多大,型号

一般中型的电商或者互联网企业，日志量每天在200-500M左右，集群规模在30-50台左右，机器一般为dell的2000左右的服务器，型号不定

大型的互联网公司据网上资料显示，日志量在GP-PB不等，集群规模在500-4000不等，甚至更多，机器型号不确定。

1. 项目,mapreduce

介绍整个mapreduce项目流程，数据采集—数据聚合—数据分析—数据展示等

1. 实时流式计算框架,几个人,多长时间,细节问题,包括讲flume ,kafka ,storm 的各个的组件组成,你负责那一块,如果需要你搭建你可以完成么?
2. 你觉得spark 可以完全替代hadoop 么?

# 面试问题:

1. 一些传统的hadoop 问题,mapreduce 他就问shuffle 阶段,你怎么理解的

Shuffle意义在于将不同map处理后的数据进行合理分配，让reduce处理，从而产生了排序、分区。

1. Mapreduce 的 map 数量 和 reduce 数量 怎么确定 ,怎么配置

Map无法配置，reduce随便配置

1. 唯一难住我的是他说实时计算,storm 如果碰上了复杂逻辑,需要算很长的时间,你怎么去优化

拆分复杂的业务到多个bolt中，这样可以利用bolt的tree将速度提升

1. Hive 你们用的是外部表还是内部表,有没有写过UDF(当然吹自己写过了),hive 的版本

外部表，udf，udaf等，hive版本为1.0

1. Hadoop 的版本

如果是1.0版本就说1.2，如果是2.0版本，就说2.6或者2.7

1.2为官方稳定版本，2.7为官方稳定版本。

Apache Hadoop 2.7.1于美国时间2015年07月06日正式发布，本版本属于稳定版本，是自Hadoop 2.6.0以来又一个稳定版，同时也是Hadoop 2.7.x版本线的第一个稳定版本，也是 2.7版本线的维护版本，变化不大，主要是修复了一些比较严重的Bug

1. 实时流式计算的结果内容有哪些,你们需要统计出来么(我就说highchart展示)

简单介绍日志监控、风控等结果内容，统计出来显示在报表或者邮件中。

1. 开始问java相关,包括luecne,solr(倒排索引的原理),框架呀,redis呀

# 京东商城 - 大数据

（1）Java篇

1、JVM，GC（算法，新生代，老年代），JVM结构

2、hashcode，hashMap，list，hashSet，equals（结构原理），A extends  B（类的加载顺序）

1.父类静态代码块；

2.子类静态代码块；

3.父类非静态代码块；

4.父类构造函数；

5.子类非静态代码块；

6.子类构造函数；

3、多线程，主线程，次线程，唤醒，睡眠

略

4、常见算法：冒泡算法，排序算法，二分查找，时间复杂度

略

（2）Flume篇

1、数据怎么采集到Kafka，实现方式

使用官方提供的flumeKafka插件，插件的实现方式是自定义了flume的sink，将数据从channle中取出，通过kafka的producer写入到kafka中，可以自定义分区等。

2、flume管道内存，flume宕机了数据丢失怎么解决

1、Flume的channel分为很多种，可以将数据写入到文件

2、防止非首个agent宕机的方法数可以做集群或者主备

3、flume配置方式，flume集群（问的很详细）

Flume的配置围绕着source、channel、sink叙述，flume的集群是做在agent上的，而非机器上。

4、flume不采集Nginx日志，通过Logger4j采集日志，优缺点是什么？

优点：Nginx的日志格式是固定的，但是缺少sessionid，通过logger4j采集的日志是带有sessionid的，而session可以通过redis共享，保证了集群日志中的同一session落到不同的tomcat时，sessionId还是一样的，而且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于紧密，而flume的方式比较灵活，拔插式比较好，不会影响项目性能。

5、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的方式记录之前的日志。

（3）Kafka篇

1、容错机制

分区备份，存在主备partition

2、同一topic不同partition分区

？？？？

3、kafka数据流向

Producer 🡪 leader partition 🡪 follower partition(半数以上) 🡪consumer

4、kafka+spark-streaming结合丢数据怎么解决？

|  |
| --- |
| spark streaming从1.2开始提供了数据的零丢失，想享受这个特性，需要满足如下条件：   1. 数据输入需要可靠的sources和可靠的receivers 2. 应用metadata必须通过应用driver checkpoint 3. WAL（write ahead log）  可靠的sources和receivers spark streaming可以通过多种方式作为数据sources（包括kafka），输入数据通过receivers接收，通过replication存储于spark中（为了faultolerance，默认复制到两个spark executors），如果数据复制完成，receivers可以知道（例如kafka中更新offsets到zookeeper中）。这样当receivers在接收数据过程中crash掉，不会有数据丢失，receivers没有复制的数据，当receiver恢复后重新接收。  [image](https://raw.githubusercontent.com/jacksu/utils4s/master/spark-knowledge/images/spark-streaming-kafka/spark-reliable-source-reliable-receiver.png) metadata checkpoint 可靠的sources和receivers，可以使数据在receivers失败后恢复，然而在driver失败后恢复是比较复杂的，一种方法是通过checkpoint metadata到HDFS或者S3。metadata包括：   * configuration * code * 一些排队等待处理但没有完成的RDD（仅仅是metadata，而不是data） [image](https://raw.githubusercontent.com/jacksu/utils4s/master/spark-knowledge/images/spark-streaming-kafka/spark-metadata-checkpointing.png)   这样当driver失败时，可以通过metadata checkpoint，重构应用程序并知道执行到那个地方。 数据可能丢失的场景 可靠的sources和receivers，以及metadata checkpoint也不可以保证数据的不丢失，例如：   * 两个executor得到计算数据，并保存在他们的内存中 * receivers知道数据已经输入 * executors开始计算数据 * driver突然失败 * driver失败，那么executors都会被kill掉 * 因为executor被kill掉，那么他们内存中得数据都会丢失，但是这些数据不再被处理 * executor中的数据不可恢复  WAL 为了避免上面情景的出现，spark streaming 1.2引入了WAL。所有接收的数据通过receivers写入HDFS或者S3中checkpoint目录，这样当driver失败后，executor中数据丢失后，可以通过checkpoint恢复。 [image](https://raw.githubusercontent.com/jacksu/utils4s/master/spark-knowledge/images/spark-streaming-kafka/spark-wal.png) At-Least-Once 尽管WAL可以保证数据零丢失，但是不能保证exactly-once，例如下面场景：   * Receivers接收完数据并保存到HDFS或S3 * 在更新offset前，receivers失败了 [image](https://raw.githubusercontent.com/jacksu/utils4s/master/spark-knowledge/images/spark-streaming-kafka/spark-wall-at-least-once-delivery.png) * Spark Streaming以为数据接收成功，但是Kafka以为数据没有接收成功，因为offset没有更新到zookeeper * 随后receiver恢复了 * 从WAL可以读取的数据重新消费一次，因为使用的kafka High-Level消费API，从zookeeper中保存的offsets开始消费  WAL的缺点 通过上面描述，WAL有两个缺点：   * 降低了receivers的性能，因为数据还要存储到HDFS等分布式文件系统 * 对于一些resources，可能存在重复的数据，比如Kafka，在Kafka中存在一份数据，在Spark Streaming也存在一份（以WAL的形式存储在hadoop API兼容的文件系统中）  Kafka direct API 为了WAL的性能损失和exactly-once，spark streaming1.3中使用Kafka direct API。非常巧妙，Spark driver计算下个batch的offsets，指导executor消费对应的topics和partitions。消费Kafka消息，就像消费文件系统文件一样。  [image](https://raw.githubusercontent.com/jacksu/utils4s/master/spark-knowledge/images/spark-streaming-kafka/spark-kafka-direct-api.png)   1. 不再需要kafka receivers，executor直接通过Kafka API消费数据 2. WAL不再需要，如果从失败恢复，可以重新消费 3. exactly-once得到了保证，不会再从WAL中重复读取数据  总结 主要说的是spark streaming通过各种方式来保证数据不丢失，并保证exactly-once，每个版本都是spark streaming越来越稳定，越来越向生产环境使用发展。 |

5、kafka中存储目录data/dir.....topic1和topic2怎么存储的，存储结构，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。

（4）Hive篇

1、hive partition分区

分区表，动态分区

2、insert into 和 override write区别？

insert into：将某一张表中的数据写到另一张表中

override write：覆盖之前的内容。

3、假如一个分区的数据主部错误怎么通过hivesql删除hdfs

alter table ptable drop partition (daytime='20140911',city='bj');

元数据，数据文件都删除，但目录daytime= 20140911还在

（5）Storm篇

 1、开发流程，容错机制

开发流程：

* 1. 写主类（设计spout和bolt的分发机制）
  2. 写spout收集数据
  3. 写bolt处理数据，根据数据量和业务的复杂程度，设计并行度。

容错机制：采用ack和fail进行容错，失败的数据重新发送。

2、storm和spark-streaming：为什么用storm不同spark-streaming

3、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………………

4、sqoop命令

|  |
| --- |
| sqoop import --connect jdbc:mysql://192.168.56.204:3306/sqoop --username hive --password hive --table jobinfo --target-dir /sqoop/test7 --inline-lob-limit 16777216 --fields-terminated-by '\t' -m 2 |
| sqoop create-hive-table --connect jdbc:mysql://192.168.56.204:3306/sqoop --table jobinfo --username hive --password hive --hive-table sqtest --fields-terminated-by "\t" --lines-terminated-by "\n"; |

（6）Redis篇

1、基本操作，存储格式

略

（7）Mysql篇

1、mysql集群的分布式事务

京东自主开发分布式MYSQL集群系统

2、mysql性能优化（数据方面）

数据的分表、分库、分区

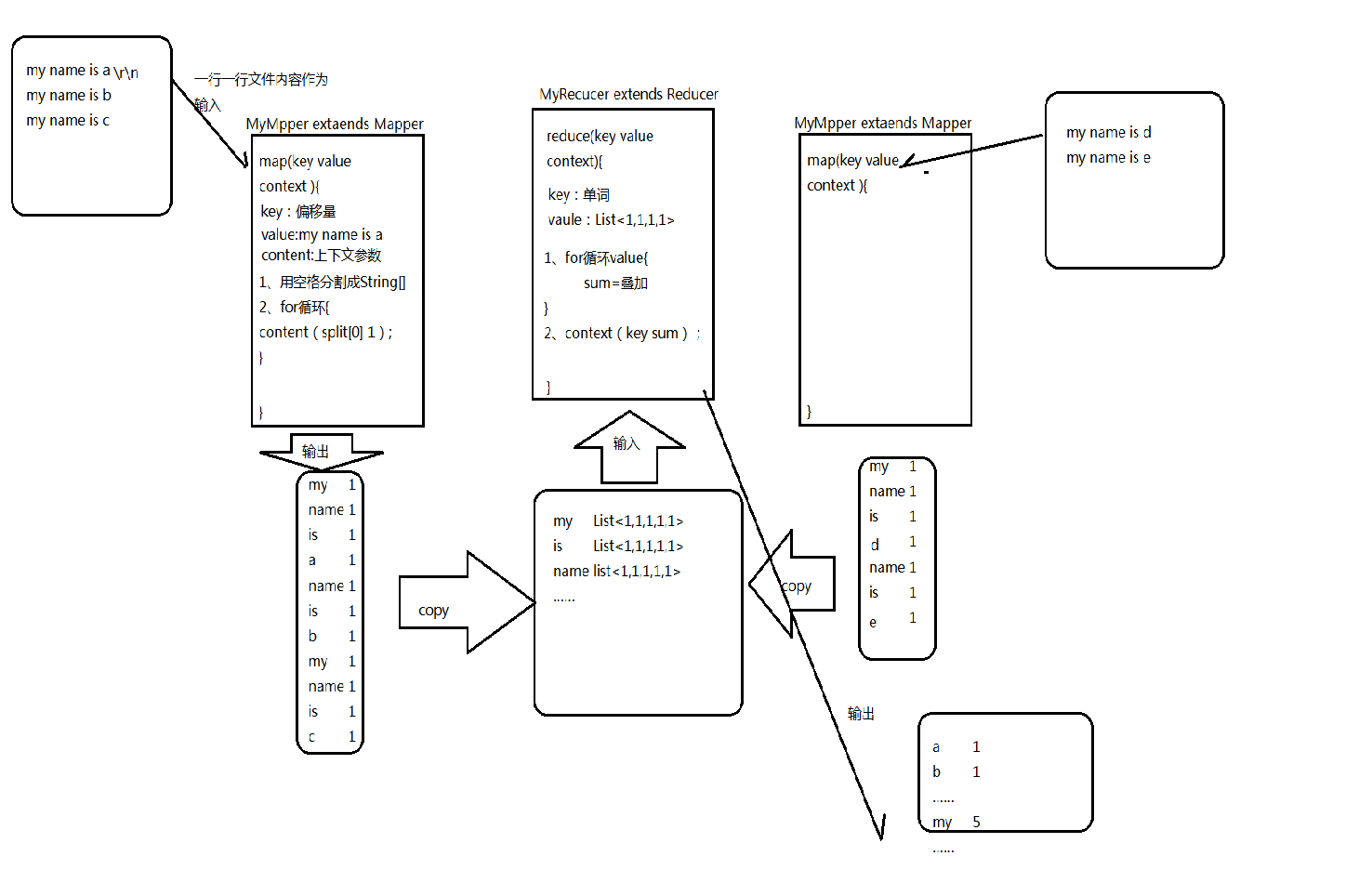
（6）Hadoop篇

1、hadoop HA  两个namenode和zk之间的通信，zk的选举机制？

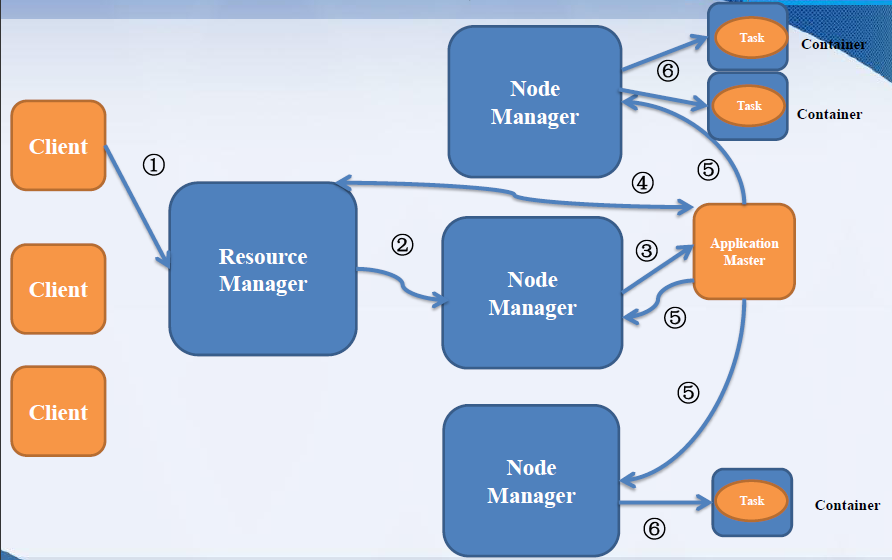
HA是通过先后获取zk的锁决定谁是主

Zk的选举机制，涉及到全新机群的选主和数据恢复的选主

2、mr运行机制



3、yarn流程



1) 用户向YARN 中提交应用程序， 其中包括ApplicationMaster 程序、启动ApplicationMaster 的命令、用户程序等。

2) ResourceManager 为该应用程序分配第一个Container， 并与对应的NodeManager 通信，要求它在这个Container 中启动应用程序的ApplicationMaster。

3) ApplicationMaster 首先向ResourceManager 注册， 这样用户可以直接通过ResourceManage 查看应用程序的运行状态，然后它将为各个任务申请资源，并监控它的运行状态，直到运行结束，即重复步骤4~7。

4) ApplicationMaster 采用轮询的方式通过RPC 协议向ResourceManager 申请和领取资源。

5) 一旦ApplicationMaster 申请到资源后，便与对应的NodeManager 通信，要求它启动任务。

6) NodeManager 为任务设置好运行环境（包括环境变量、JAR 包、二进制程序等）后，将任务启动命令写到一个脚本中，并通过运行该脚本启动任务。

7) 各个任务通过某个RPC 协议向ApplicationMaster 汇报自己的状态和进度，以让ApplicationMaster 随时掌握各个任务的运行状态，从而可以在任务失败时重新启动任务。在应用程序运行过程中，用户可随时通过RPC 向ApplicationMaster 查询应用程序的当前运行状态。

8) 应用程序运行完成后，ApplicationMaster 向ResourceManager 注销并关闭自己。

（7）Hbase

1、涉及到概念，文档

（8）Spark篇

1、spark原理

Spark应用转换流程

|  |
| --- |
| * 1. spark应用提交后，经历了一系列的转换，最后成为task在每个节点上执行   2. RDD的Action算子触发Job的提交，生成RDD DAG   3. 由DAGScheduler将RDD DAG转化为Stage DAG，每个Stage中产生相应的Task集合   4. TaskScheduler将任务分发到Executor执行   5. 每个任务对应相应的一个数据块，只用用户定义的函数处理数据块 |

|  |
| --- |
| **Driver运行在Worker上**  通过org.apache.spark.deploy.Client类执行作业，作业运行命令如下：    作业执行流程描述：  1、客户端提交作业给Master  2、Master让一个Worker启动Driver，即SchedulerBackend。Worker创建一个DriverRunner线程，DriverRunner启动SchedulerBackend进程。  3、另外Master还会让其余Worker启动Exeuctor，即ExecutorBackend。Worker创建一个ExecutorRunner线程，ExecutorRunner会启动ExecutorBackend进程。  4、ExecutorBackend启动后会向Driver的SchedulerBackend注册。SchedulerBackend进程中包含DAGScheduler，它会根据用户程序，生成执行计划，并调度执行。对于每个stage的task，都会被存放到TaskScheduler中，ExecutorBackend向SchedulerBackend汇报的时候把TaskScheduler中的task调度到ExecutorBackend执行。  5、所有stage都完成后作业结束。 |
| **Driver运行在客户端**    作业执行流程描述：  1、客户端启动后直接运行用户程序，启动Driver相关的工作：DAGScheduler和BlockManagerMaster等。  2、客户端的Driver向Master注册。  3、Master还会让Worker启动Exeuctor。Worker创建一个ExecutorRunner线程，ExecutorRunner会启动ExecutorBackend进程。  4、ExecutorBackend启动后会向Driver的SchedulerBackend注册。Driver的DAGScheduler解析作业并生成相应的Stage，每个Stage包含的Task通过TaskScheduler分配给Executor执行。  5、所有stage都完成后作业结束。 |