大数据知识点

# hadoop知识点

## 1.1hadoop是什么？

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

* 解决问题：

海量数据的存储（HDFS）

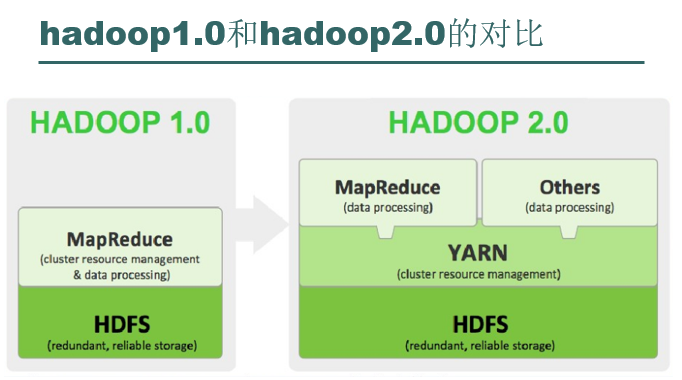
海量数据的分析（MapReduce）

资源管理调度（YARN）

* HDFS: Hadoop Distributed File System 分布式文件系统
* YARN: Yet Another Resource Negotiator 资源管理调度系统
* Mapreduce：分布式运算框架

**理解**：hadoop1.x和hadoop2.x的区别：

YARN在hadoop1.x中是没有的，之后在hadoop2.x中才存在。YARN是从MapReduce中抽象出来的，抽象出公共的部分，形成的

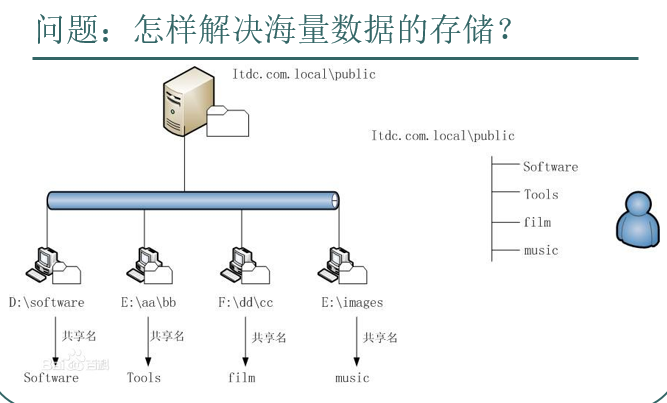


## 1.2 hadoop具体能干什么？

* hadoop擅长日志分析，facebook就用Hive来进行日志分析，2009年时facebook就有非编程人员的30%的人使用HiveQL进行数据分析；淘宝搜索中的自定义筛选也使用的Hive；利用Pig还可以做高级的数据处理，包括Twitter、LinkedIn 上用于发现您可能认识的人，可以实现类似Amazon.com的协同过滤的推荐效果。淘宝的商品推荐也是！在Yahoo！的40%的Hadoop作业是用pig运行的，包括垃圾邮件的识别和过滤，还有用户特征建模。（2012年8月25新更新，天猫的推荐系统是hive，少量尝试mahout！）

## 怎样解决海量数据的存储

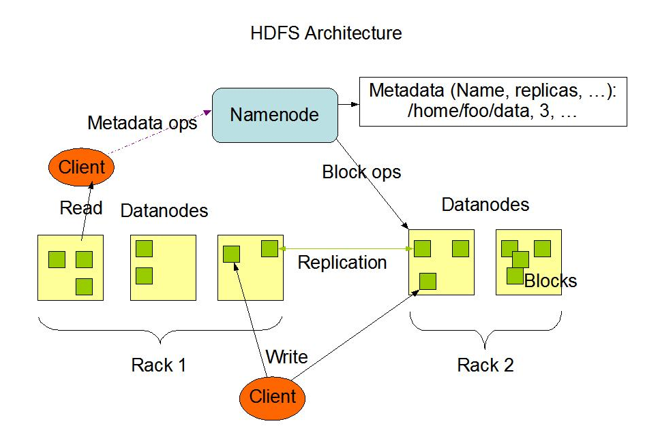
### 1.3.1 平常解决方式



缺点：1、对数据分析处理，该种方式不太合理

2、如果其中一个损坏，则其中的文件会丢失。

### 1.3.2 hadoop方法解决方式



* 主从结构

主节点， *namenode*

从节点，有很多个: *datanode*

* namenode负责：

接收用户操作请求

维护文件系统的目录结构

管理文件与block之间关系，block与datanode之间关系

* datanode负责：

存储文件

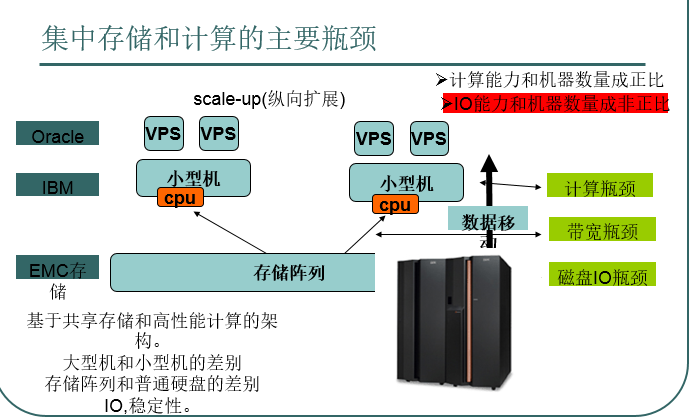
文件被分成block存储在磁盘上

为保证数据安全，文件会有多个副本

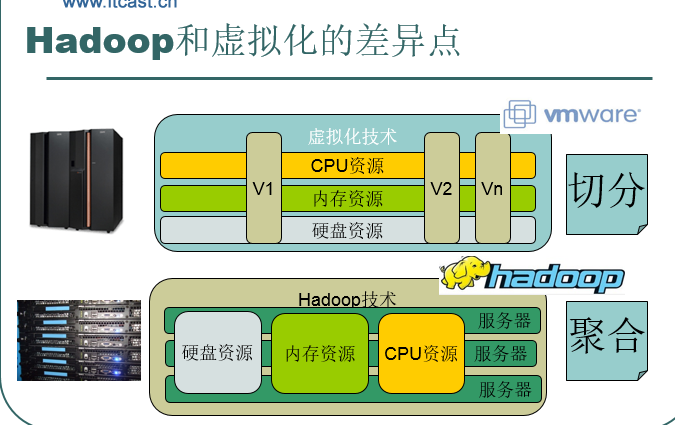
### 1.3.3 hadoop特点：

* 扩容能力（Scalable）：能可靠地（reliably）存储和处理千兆字节（PB）数据。
* 成本低（Economical）：可以通过普通机器组成的服务器群来分发以及处理数据。这些服务器群总计可达数千个节点。
* 高效率（Efficient）：通过分发数据，hadoop可以在数据所在的节点上并行地（parallel）处理它们，这使得处理非常的快速。
* 可靠性（Reliable）：hadoop能自动地维护数据的多份副本，并且在任务失败后能自动地重新部署（redeploy）计算任务

### 1.3.4 集中存储和计算的主要瓶颈



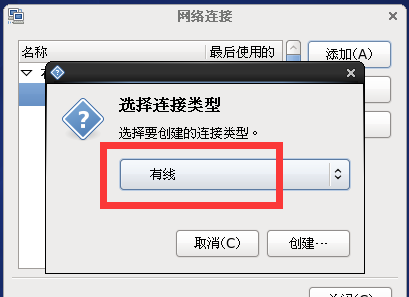
### 1.3.5 hadoop和虚拟化的差异点

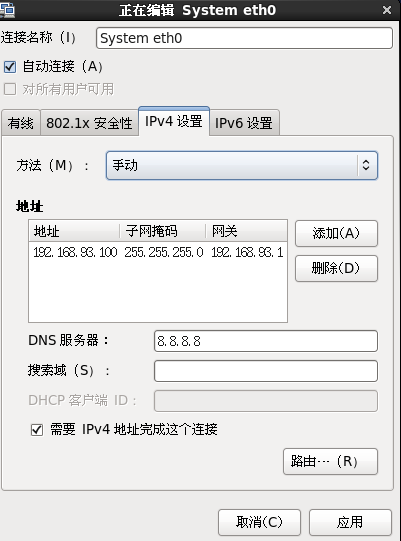


## linux常用配置

### ip地址的配置方法

#### 1.4.1.1 图形化界面方式（第一种方式）





#### 1.4.1.2修改配置文件方式（屌丝程序猿专用）

**vim /etc/sysconfig/network-scripts/ifcfg-eth0**

|  |
| --- |
| DEVICE="eth0"  BOOTPROTO="static" ###  HWADDR="00:0C:29:3C:BF:E7"  IPV6INIT="yes"  NM\_CONTROLLED="yes"  ONBOOT="yes"  TYPE="Ethernet"  UUID="ce22eeca-ecde-4536-8cc2-ef0dc36d4a8c"  IPADDR="192.168.1.101" ###  NETMASK="255.255.255.0" ###  GATEWAY="192.168.1.1" ### |

注意：改完配置文件文件后不会自动生效，必须重启linux服务器（reboot）或者重启network服务(sudo service network restart)

### 1.4.2关闭图形化界面

第一步：在图形化界面直接输入 init 3

第二步：修改 /etc/inittab 文件 修改最后一行

id:3:initdefault: 改成3即可。

### 1.4.3修改主机名和让普通用户具备sudo执行权限

#######新增删除用户操作#########

添加用户test: adduser test

修改test用户密码：passwd test

删除用户test: userdel test

删除用户及用户目录：userdel -r test

#####让普通用户具备sudo执行权限

切换到root

然后 vi /etc/sudoers 加入一行

root ALL=(ALL) ALL

hadoop ALL=(ALL) ALL

#########修改主机名#########

vim /etc/sysconfig/network

NETWORKING=yes

HOSTNAME=itcast ###

sudo hostname weekend111

exit

立刻使用，但是需要先退出重新登录下。

修改文件夹用户权限：把root改为hadoop用户



sudo chown hadoop:hadoop moudle/ software/



### 1.4.4 关闭防火墙

关闭防火墙

#查看防火墙状态

service iptables status

#关闭防火墙

service iptables stop

#查看防火墙开机启动状态

chkconfig iptables --list

#关闭防火墙开机启动

chkconfig iptables off

永久关闭防火墙步骤：

1. 首先运行service iptables stop
2. 然后在运行chkconfig iptables off
3. 然后可以查看service iptables status

说明：必须先执行第一步，先关闭防火墙，在修改配置文件防火墙自动启动

### 1.4.5 修改ip地址与主机名的映射关系

把主机名和ip地址关联进行关系映射。

|  |
| --- |
| vim /etc/hosts  192.168.1.101 itcast |

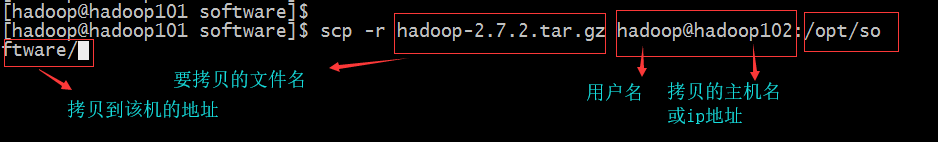
### 1.4.6 安装jdk

|  |
| --- |
| 上传alt+p 后出现sftp窗口，然后put d:\xxx\yy\ll\jdk-7u\_65-i585.tar.gz    2.2解压jdk  #创建文件夹  mkdir /home/hadoop/app  #解压  tar -zxvf jdk-7u55-linux-i586.tar.gz -C /home/hadoop/app    2.3将java添加到环境变量中  vim /etc/profile  #在文件最后添加  export JAVA\_HOME=/home/hadoop/app/jdk-7u\_65-i585  export PATH=$PATH:$JAVA\_HOME/bin  注意：如果linux中自带jdk这样配置将会显示不出安装的jdk信息，所以需要把$PATH放置在后面  export PATH=$JAVA\_HOME/bin: $PATH  #刷新配置  source /etc/profile |

### 1.4.7 SCP命令

#### Scp推送(主动推送到别的主机)

scp -r hadoop-2.7.2.tar.gz hadoop@hadoop102:/opt/software/



#### Scp拉取(主动从别的主机拉取到自己主机)

scp -r [hadoop@hadoop102:/opt/software/hadoop-2.7.2.tar.gz](mailto:hadoop@hadoop102:/opt/software/hadoop-2.7.2.tar.gz) ./



### 1.4.8 rsync远程同步工具

rsync主要用于远程同步工具，主要用于备份和镜像，具有速度快，避免复制相同内容和支持符号链接的有点。

（1）查看rsync使用说明

man rsync | more

（2）基本语法

rsync -rvl $pdir/$fname $user@hadoop$host:$pdir

命令 命令参数 要拷贝的文件路径/名称 目的用户@主机:目的路径

选项

-r 递归

-v 显示复制过程

-l 拷贝符号连接

（3）案例实操

把本机/opt/tmp目录同步到hadoop103服务器的root用户下的/opt/tmp目录

rsync -rvl /opt/tmp/\* root@hadoop103:/op t/tmp

### 1.4.9编写集群分发脚本xsync

1）需求分析：循环复制文件到所有节点的相同目录下。

（1）原始拷贝：

rsync -rvl /opt/module root@hadoop103:/opt/

（2）期望脚本：

xsync 要同步的文件名称

（3）在/usr/local/bin这个目录下存放的脚本，可以在系统任何地方直接执行，需要制定路径。

2）案例实操：

（1）在/usr/local/bin目录下创建xsync文件，文件内容如下：

|  |
| --- |
| #!/bin/bash  #1 获取输入参数个数，如果没有参数，直接退出  pcount=$#  if((pcount==0)); then  echo no args;  exit;  fi  #2 获取文件名称  p1=$1  fname=`basename $p1`  echo fname=$fname  #3 获取上级目录到绝对路径  pdir=`cd -P $(dirname $p1); pwd`  echo pdir=$pdir  #4 获取当前用户名称  user=`whoami`  #5 循环  for((host=103; host<105; host++)); do  #echo $pdir/$fname $user@hadoop$host:$pdir  echo --------------- hadoop$host ----------------  rsync -rvl $pdir/$fname $user@hadoop$host:$pdir  done |

（2）修改脚本 xsync 具有执行权限

[root@hadoop102 bin]# chmod a+x xsync

（3）调用脚本形式：xsync 文件名称

### 1.4.10 编写分发脚本xcall

1）需求分析：在所有主机上同时执行相同的命令

xcall +命令

2）具体实现

（1）在/usr/local/bin目录下创建xcall文件，文件内容如下：

|  |
| --- |
| #!/bin/bash  pcount=$#  if((pcount==0));then  echo no args;  exit;  fi  echo -------------localhost----------  #本机的时候加上下面一句  #$@  for((host=101; host<=108; host++)); do  echo ----------hadoop$host---------  ssh hadoop$host $@  done |

（2）修改脚本 xcall 具有执行权限

[root@hadoop102 bin]# chmod a+x xcall

（3）调用脚本形式： xcall 操作命令

[root@hadoop102 ~]# xcall rm -rf /opt/tmp/profile

## 1.5 hadoop伪集群安装

第一步：首先配置好一个虚拟机，然后打开，配置ip地址，添加sudo执行权限，关联主机名和ip地址的映射关系。

第二步：安装jdk

第三步：安装hadoop （该事例使用的是。hadoop-2.4.1.tar.gz）

先上传hadoop-2.4.1.tar.gz压缩包，然后解压

### 1.5.1 hadoop安装配置

第一步：首先进入hadoop解压文件夹中 cd /home/hadoop/app/hadoop-2.4.1

然后进入share文件夹中删除doc文件夹

原因：doc中是一些文档，删除可以加快安装部署（也可不删除，没有什么太大的必要）

Hadoop的jar包在share中的hadoop文件夹中 /home/hadoop/app/hadoop-2.4.1/share/Hadoop

第二步：修改配置文件 /home/hadoop/app/hadoop-2.4.1/etc/Hadoop

修改hadoop-env.sh（环境变量）文件

原因：因为原本为自动获取，但是总是获取不到，所以直接修改为jdk的目录

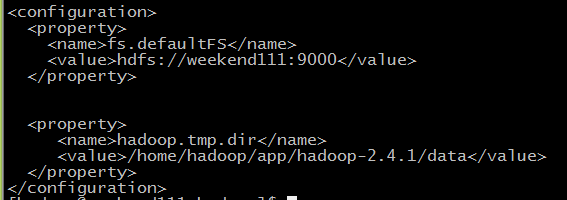
原先的：export JAVA\_HOME=${ JAVA\_HOME }

修改之后：export JAVA\_HOME=/home/hadoop/app/jdk1.7.0\_65

修改core-site.xml（核心配置文件）文件

在configuration标签中新增一下内容：

|  |
| --- |
| <!-- 指定HADOOP所使用的文件系统schema（URI），HDFS的老大（NameNode）的地址 -->  <property>  <name>fs.defaultFS</name>  <value>hdfs://weekend111:9000</value>  </property>  <!-- 指定hadoop运行时产生文件的存储目录 （不用手动创建data文件夹）-->  <property>  <name>hadoop.tmp.dir</name>  <value> /home/hadoop/app/hadoop-2.4.1/data </value>  </property> |



修改hdfs-site.xml（HDFS存储文件）文件（hdfs-site.xml hdfs-default.xml）

在configuration标签中新增一下内容：

|  |
| --- |
| <!-- 指定HDFS副本的数量 -->  <property>  <name>dfs.replication</name>  <value>1</value>  </property> |

修改mapred-site.xml （mapreduce的配置文件：数据分析）

(mv mapred-site.xml.template mapred-site.xml)

mv mapred-site.xml.template mapred-site.xml

vim mapred-site.xml

在configuration标签中新增一下内容：

|  |
| --- |
| <!-- 指定mr运行在yarn上 -->  原因：之所以指定，是指定之后会在集群中run，不然就是单机版程序  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property> |

修改 yarn-site.xml 文件

在configuration标签中新增一下内容：

|  |
| --- |
| <!-- 指定YARN的老大（ResourceManager）的地址 -->  <property>  <name>yarn.resourcemanager.hostname</name>  <value>weekend110</value>  </property>  <!-- reducer获取数据的方式 -->  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property> |

### 1.5.2 将hadoop添加到环境变量

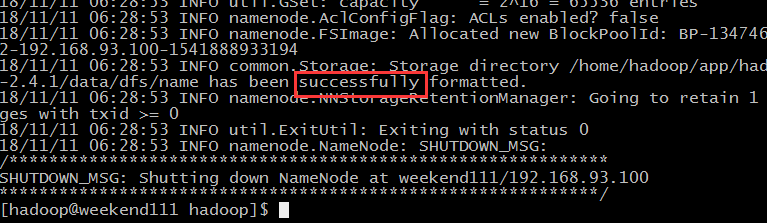
|  |
| --- |
| vim /etc/proflie  export JAVA\_HOME=/home/hadoop/app/jdk1.7.0\_65  export HADOOP\_HOME=/home/hadoop/app/hadoop-2.4.1  export PATH=$JAVA\_HOME/bin:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$PATH  source /etc/profile |

### 1.5.3 hadoop常用的命令

* 格式化namenode（是对namenode进行初始化）

hdfs namenode -format (hadoop namenode -format)

成功样例：



* 启动hadoop

先启动HDFS

sbin/start-dfs.sh

再启动YARN

sbin/start-yarn.sh

* 验证是否启动成功

使用jps命令验证

27408 NameNode

28218 Jps

27643 SecondaryNameNode

28066 NodeManager

27803 ResourceManager

27512 DataNode

http://192.168.1.101:50070 （HDFS管理界面）

http://192.168.1.101:8088 （MR管理界面）

也可以使用主机名访问，但是在windows中不识别，需要配置hosts文件

C:\Windows\System32\drivers\etc 文件所在目录路径。配置完成即可访问

例如：http://weekend111:50070

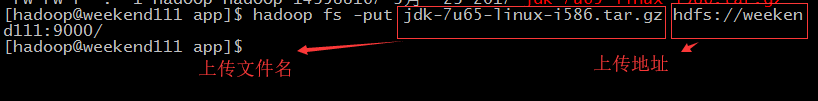
### 1.5.4 配置ssh免密登录

|  |
| --- |
| #生成ssh免登陆密钥  #进入到我的home目录  cd ~/.ssh  ssh-keygen -t rsa （四个回车）  执行完这个命令后，会生成两个文件id\_rsa（私钥）、id\_rsa.pub（公钥）  将公钥拷贝到要免登陆的机器上  ssh-copy-id localhost  ssh登录原理：    详细配置ssh通用免密登录步骤：   1. 首先在回到home目录：cd ~ 2. 使用ssh-keygen -t rsa （四个回车，一切使用默认） 3. 使用ll –a 查看隐藏文件 4. 进入.ssh隐藏文件目录 5. 拷贝id\_rsa.pub 文件到需要免密登录的机器 6. 然后把id\_rsa.pub文件的内容追加到authorized\_keys(如果没有则进行创建，创建之后修改权限，只能管理员用户可以操作) 追加的命令 cat id\_rsa.pub（公钥所在目录） >> ./authorized\_keys   配置ssh免密登录hadoop伪集群：   1. 第一步到第四步同上 2. 如果没有authorized\_keys文件则创建(创建之后需要修改权限，只能管理员用户可以操作 chmod 600 文件名)，然后追加公钥文件中的内容 cat ../id\_rsa.pub（公钥所在目录） >> ./authorized\_keys |

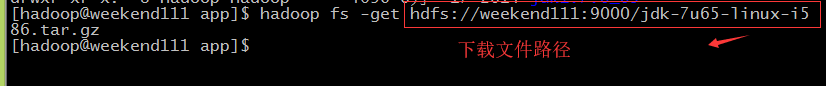
## 1.6. hdfs常用操作命令

hdfs://weekend111:9000/ 可以省略不写。

上传命令：hadoop fs -put jdk-7u65-linux-i586.tar.gz hdfs://weekend111:9000/



下载命令：hadoop fs -get hdfs://weekend111:9000/jdk-7u65-linux-i586.tar.gz



运行例子命令：hadoop jar hadoop-mapreduce-examples-2.4.1.jar pi 5 5

创建文件夹：hadoop fs -mkdir hdfs://weekend111:9000/wordcount

也可以直接省略hdfs://weekend111:9000/

计算文件中的数量：hadoop jar hadoop-mapreduce-examples-2.4.1.jar wordcount /wordcount/input /wordcount/output

使用方法

运行的jar包

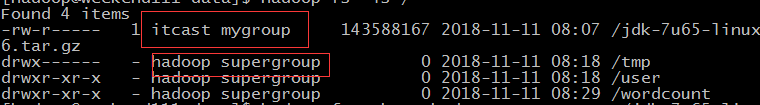
计算输出的文件目录

查找的文件目录

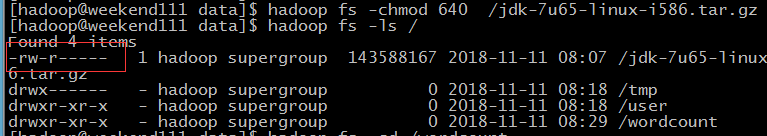
查看目录下的文件：hadoop fs -ls /wordcount/output

查看文件中的内容: hadoop fs -cat /wordcount/output/part-r-00000

修改文件所属用户：hadoop fs -chown hadoop:supergroup /jdk-7u65-linux-i586.tar.gz



修改文件权限：hadoop fs -chmod 640 /jdk-7u65-linux-i586.tar.gz 让其他用户没有可读权限



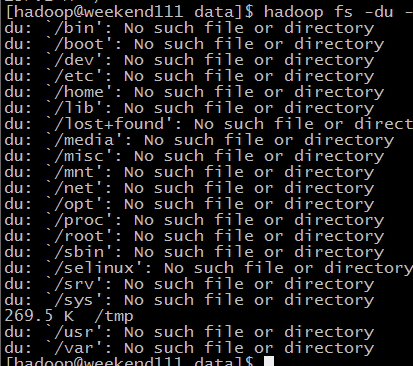
查看磁盘空间：hadoop fs -df -h /



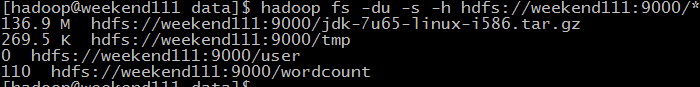
查看文件下的总大小：hadoop fs -du -s -h /



问题：hadoop fs -du -s -h /\* 不能这样写，否则会当成本地文件去进行统计



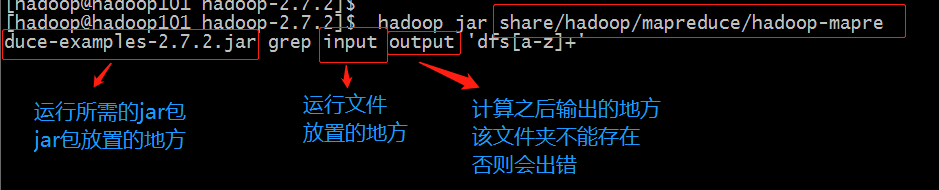
正确的书写方式： hadoop fs -du -s -h hdfs://weekend111:9000/\*



删除文件： fs –rm -r /aa/bb

### 本地运行grep案列

hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z]+



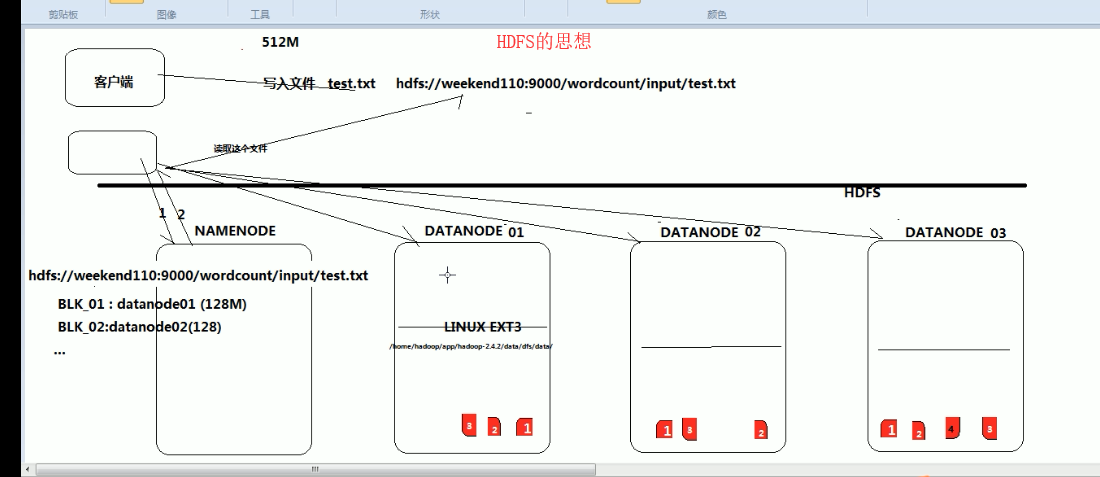
## 1.7 HDFS的实现思想

|  |
| --- |
| 1. hdfs是通过分布式集群来存储文件，为客户端提供了一个便捷的访问方式，就是一个虚拟的目录结构。  2.文件存储到hdfs集群中去的时候是被切分成block的  3.文件的block存放在若干台datanode节点上  4.hdfs文件系统中的文件与真实的block之间有映射关系，由namenode管理。  5.每一个block在集群中会存储多个副本，好处是可以提高数据的可靠性，还可以提高访问的吞吐量 |

**HDFS实现机制：**

**首先客户端先存入数据，进行分块保存在不同的dataNode**

**接着其它机器访问时，首先访问服务器所暴露在外的地址链接，其实就是直接访问nameNode接点，在nameNode节点查找出所要访问的信息所保存的dataNode节点信息，然后在去访问不同的dataNode去获取所要访问的信息。**



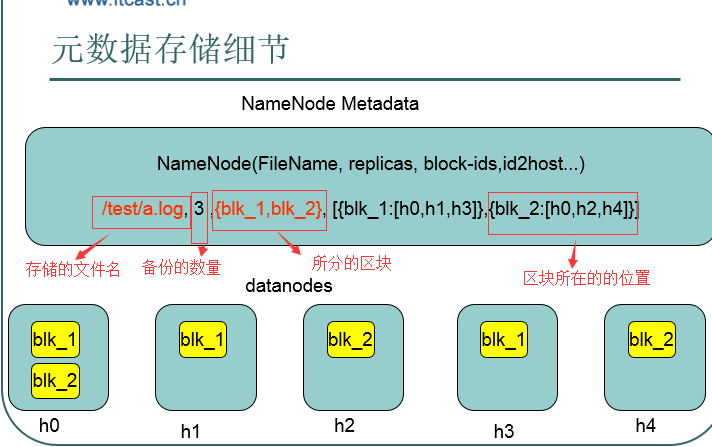
## 1.8 NN（NameNode）元数据和DN（dataNode）

### 1.8.1 NN（NameNode）元数据管理机制

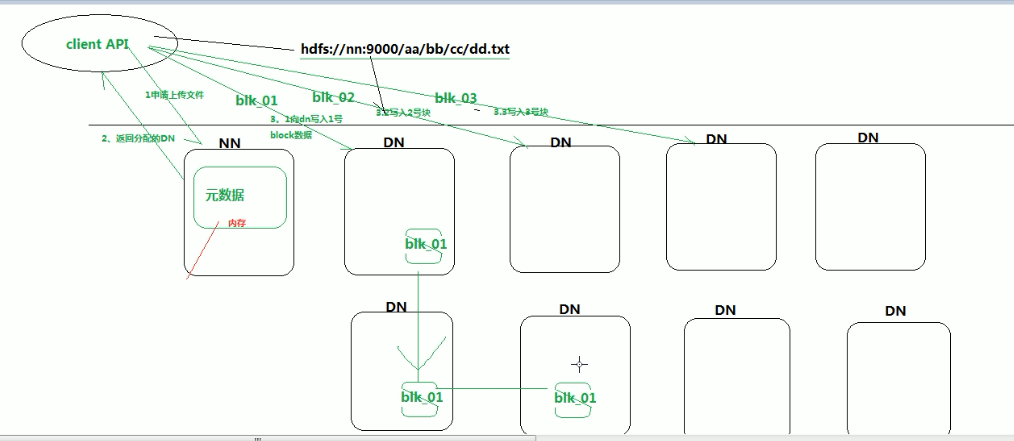
NN：nameNode

DN：dataNode

元数据存储细节：

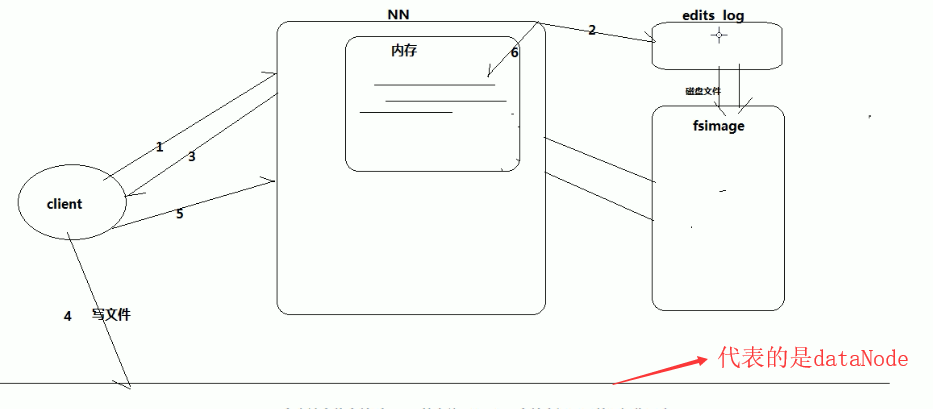


存储流程：



元数据管理机制：

|  |
| --- |
| 1. 客户端上传文件时，NN首先往edits log 文件中记录元数据操作日志 2. 客户端开始上传文件，完成后返回成功信息给NN，NN就在内存中写入这次上传操作的新产生的元数据信息 3. 每当editslog写满时，需要将这一段时间的新的元数据刷到fsimage文件中。（那么怎么刷到fsimage文件中，其实需要secondNameNode，然后使用checkpoint进行检验editslog文件。） |

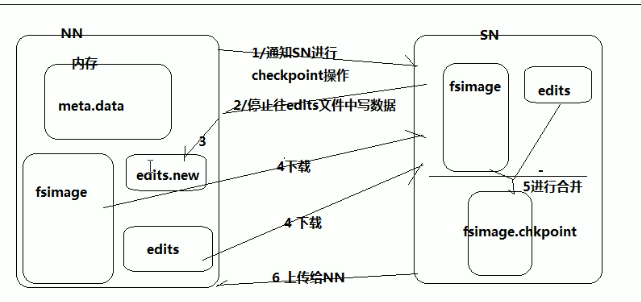


### 1.8.2 SN（SecondaryNameNode ）

在该路径下/home/hadoop/app/hadoop-2.4.1/data/dfs/name/current可以查看edits文件

editslog元数据日志合并到fsimage中：

SN：SecondNameNode



|  |
| --- |
| 1. secondary通知namenode切换edits文件 2. secondary从namenode获得fsimage和edits(通过http) 3. secondary将fsimage载入内存，然后开始合并edits 4. secondary将新的fsimage发回给namenode 5. namenode用新的fsimage替换旧的fsimage  * fs.checkpoint.period 指定两次checkpoint的最大时间间隔，默认3600秒。 * fs.checkpoint.size    规定edits文件的最大值，一旦超过这个值则强制checkpoint，不管是否到达最大时间间隔。默认大小是64M。 |

NN的职责：

|  |
| --- |
| 1. 维护元数据信息 2. 维护hdfs的目录树 3. 响应客户端的请求 |

### 问题：如果这个NameNode荡机了怎么办？

在这个机器中装两个NameNode，让数据同步（接下来会讲解）

### 1.8.3 DN（DataNode）工作原理

* 提供真实文件数据的存储服务。
* 文件块（block）：**最基本的存储单位。**对于文件内容而言，一个文件的长度大小是size，那么从文件的０偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个Block。**HDFS默认Block大小是128MB，**以一个256MB文件，共有256/128=2个Block.

dfs.block.size

* **不同于普通文件系统的是，HDFS中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间**
* Replication。多复本。默认是三个。
  + hdfs-site.xml的dfs.replication属性
* 方法：上传大于128MB的文件，观察块大小
* 验证：使用 http://hadoop0:50070 观察

## 1.9 HDFS的java客户端编写

第一步:首先上传文件eclipse-jee-luna-SR2-linux-gtk.tar.gz并解压

第二步：打开eclipse并创建一个java工程

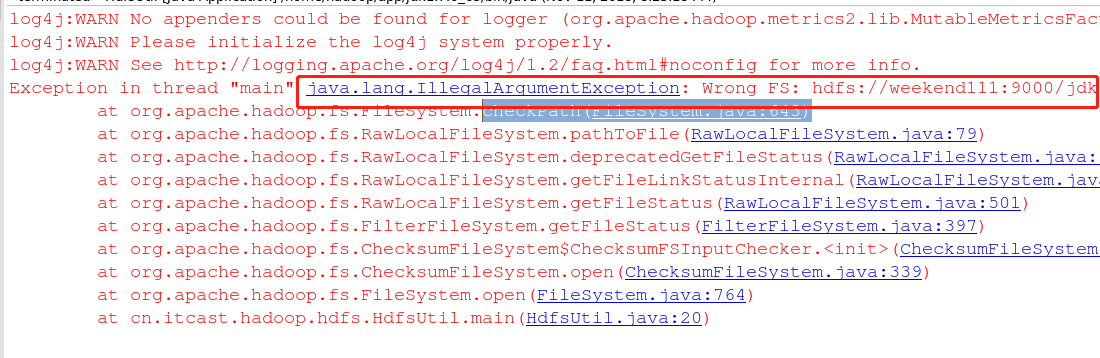
第三步：创建一个user Library 导入 hadoop-2.4.1\hadoop-2.4.1\share\hadoop\hdfs 目录下的的hadoop-hdfs-2.4.1.jar和lib中的jar包，同时导入common文件中的核心包和lib中的jar包 坏境搭建完成

第四步：写代码



|  |
| --- |
| Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(conf);  Path src = new Path("hdfs://weekend111:9000/jdk-7u65-linux-i586.tar.gz");  FSDataInputStream in = fs.open(src);  FileOutputStream os = new FileOutputStream("/home/hadoop/download/jdk.tgz");  IOUtils.copy(in, os); |

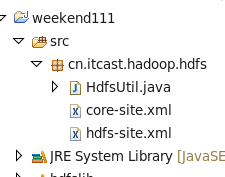
错误：



不认识hdfs://weekend111:9000/ 这个字符串

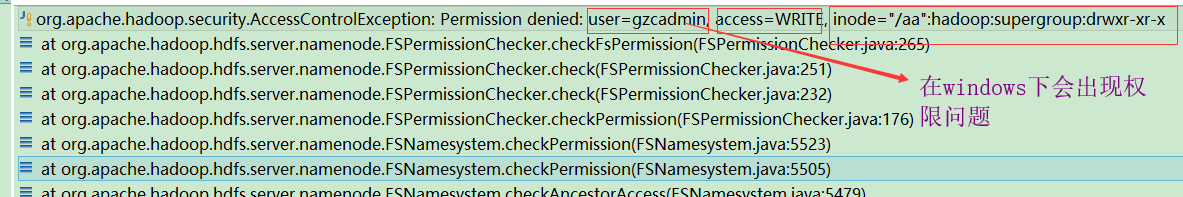
解决办法有两种：

第一种：把\hadoop-2.4.1\hadoop-2.4.1\etc\hadoop 目录中的core-site.xml文件和hdfs-site.xml文件拷贝到当前工程中。例如：



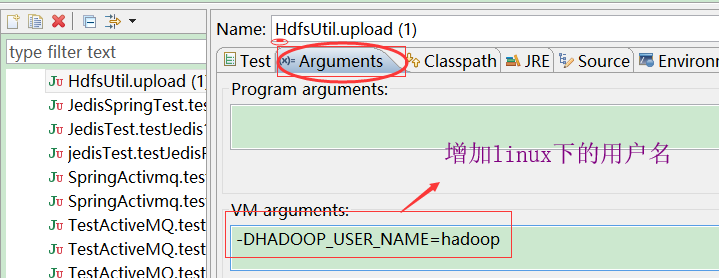
第二种：在加载配置文件时进行设置conf.set("fs.defaultFS", "hdfs://weekend111:9000/");

在windows下进行连接的问题：会出现权限验证的问题



解决：

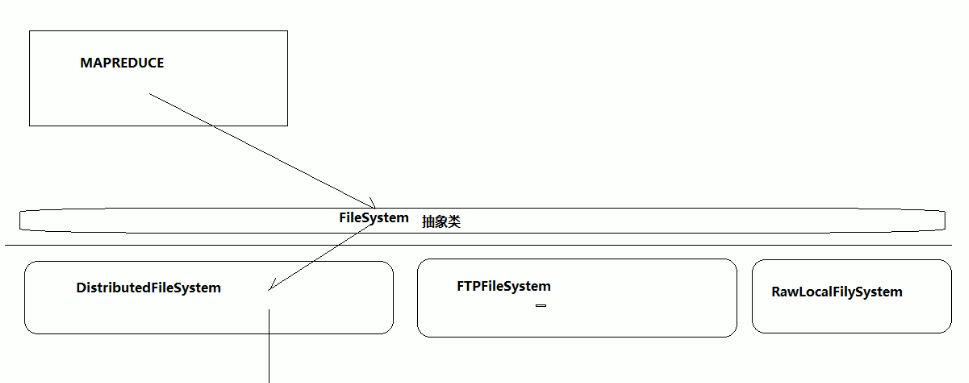
第一种：增加DHAOOP\_USER\_NAME=linux下的用户名（可以使用hadoop的权限的用户名）

  
第二种：fs = FileSystem.get(new URI("hdfs://weekend111:9000/"), conf, "hadoop"); 在获取的时候直接指定hadoop可以操作的用户名

### 1.9.1 hdfs下载数据源码流程



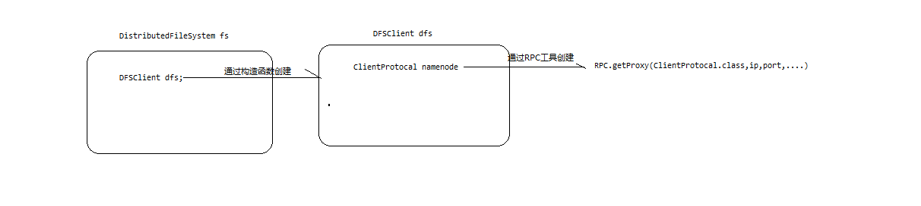
### 1.9.2 FileSystem的思想



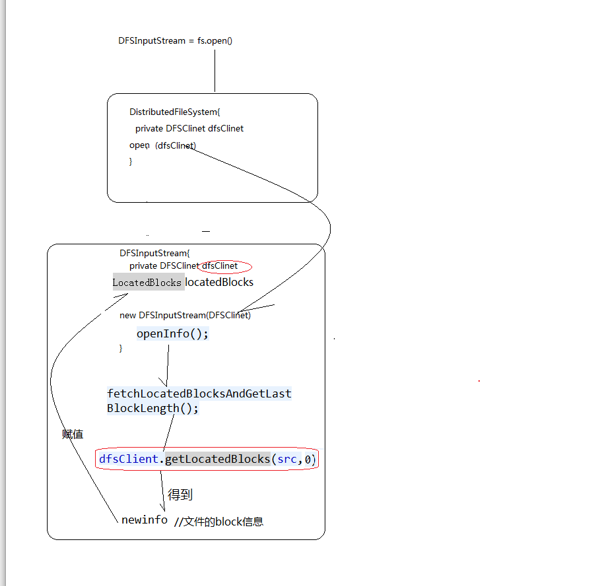
Mapreduce调用FileSystem抽象类，然后由FileSystem去实例化具体的实现类，mapreduce则不关心调用具体哪一个实现类。

### 1.9.3 获取FileSystem的源码分析





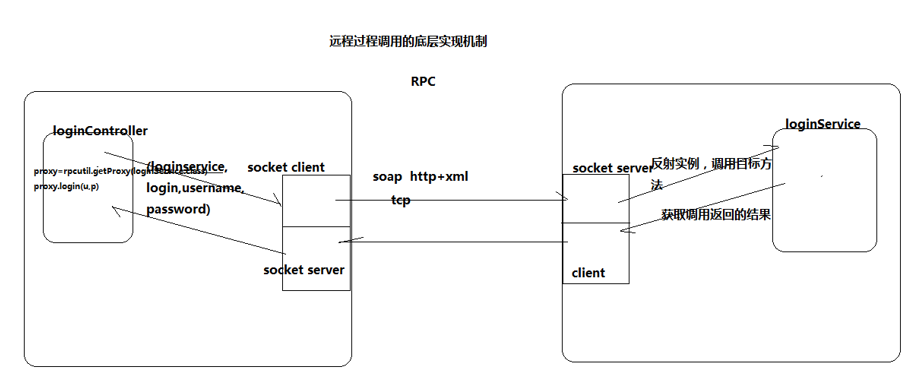
### 1.9.4 open流程



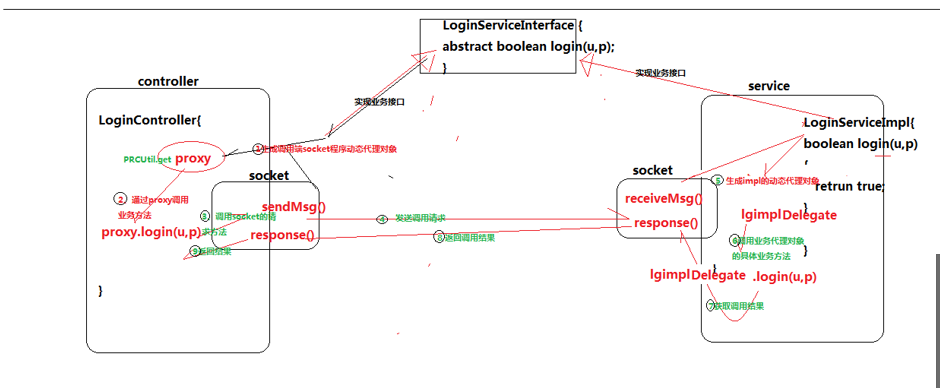


## 1.10 hadoop中的RPC框架

**远程过程调用的底层实现机制：**



**具体实现：**



|  |
| --- |
| 1. **生成调用端socket程序动态代理对象(需要实现loginServiceInterface接口)** 2. **通过proxy调用业务方法** 3. **调用socket的请求方法** 4. **发送调用请求** 5. **生成impl的动态代理对象** 6. **调用业务代理对象的具体方法** 7. **获取调用结果到生成socket端** 8. **生成socket端返回调用结果到调用socket端** 9. **调用端返回结果** |

### 1.10.1 什么是RPC？

|  |
| --- |
| * RPC——远程过程调用协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。 * RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息的到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。 * hadoop的整个体系结构就是构建在RPC之上的(见org.apache.hadoop.ipc)。 |

### 10.2 rpc的demo实例

Windows端：



Linux端：



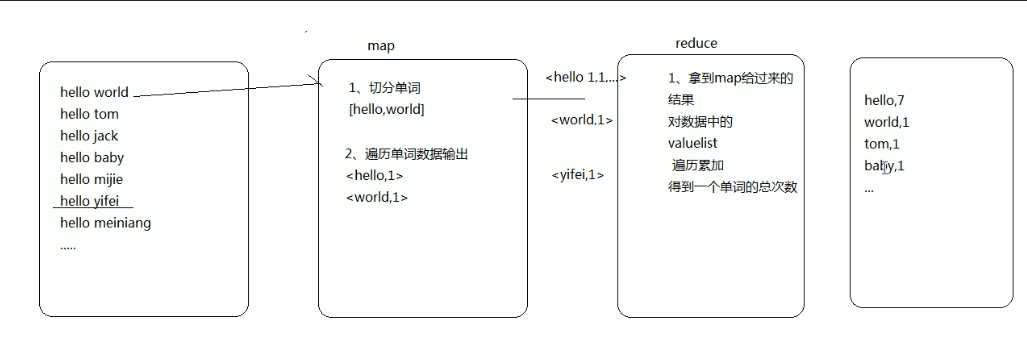
## 1.11 mr(MapReduce)

|  |
| --- |
| 1.执行MR的命令：  hadoop jar <jar在linux的路径> <main方法所在的类的全类名> <参数>  例子：  hadoop jar /root/wc1.jar cn.itcast.d3.hadoop.mr.WordCount hdfs://itcast:9000/words /out2  2.MR执行流程  (1).客户端提交一个mr的jar包给JobClient(提交方式：hadoop jar ...)  (2).JobClient通过RPC和JobTracker进行通信，返回一个存放jar包的地址（HDFS）和jobId  (3).client将jar包写入到HDFS当中(path = hdfs上的地址 + jobId)  (4).开始提交任务(任务的描述信息，不是jar, 包括jobid，jar存放的位置，配置信息等等)  (5).JobTracker进行初始化任务  (6).读取HDFS上的要处理的文件，开始计算输入分片，每一个分片对应一个MapperTask  (7).TaskTracker通过心跳机制领取任务（任务的描述信息）  (8).下载所需的jar，配置文件等  (9).TaskTracker启动一个java child子进程，用来执行具体的任务（MapperTask或ReducerTask）  (10).将结果写入到HDFS当中 |

### 1.11.1 MapReduce介绍及wordcount

hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount wcinput/ wcoutput

图示：

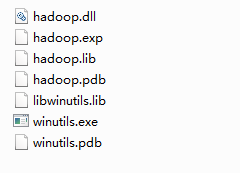


代码：



注意：要想在windows本地运行，需要配置hadoop的环境变量

同是需要在hadoop的bin目录中新增如下文件



首先配置HADOOP\_HOME

路径：E:\BaiduNetdiskDownload\hadoop-2.4.1

然后在path中追加

%HADOOP\_HOME%\bin;%HADOOP\_HOME%\sbin;

**Linux上运行：**

不需要增加conf.set("fs.defaultFS", "hdfs://weekend111:9000/");

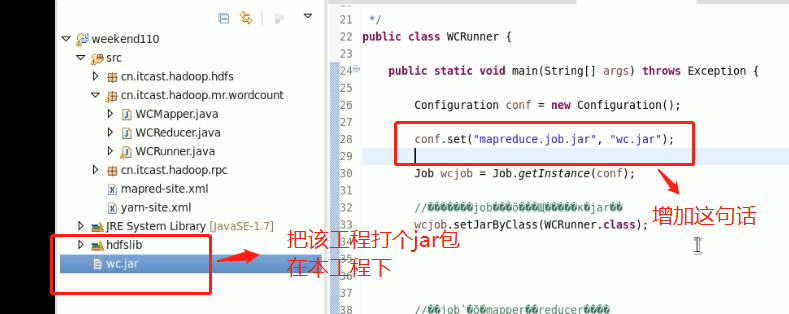
或者core.site和hdfs.site

**运行hadoop集群上的文件：**

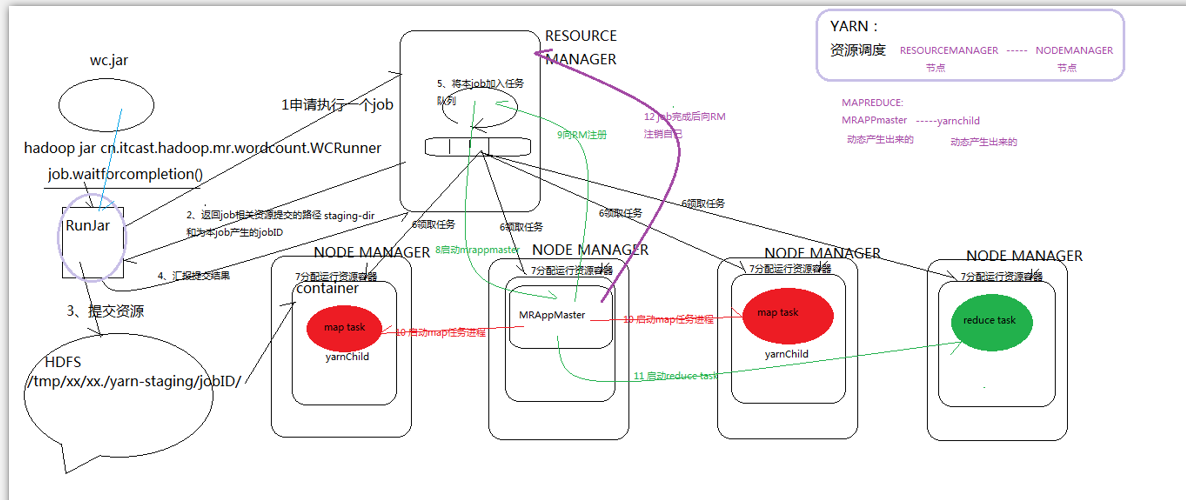
增加conf.set("fs.defaultFS", "hdfs://weekend111:9000/");

或者core.site和hdfs.site

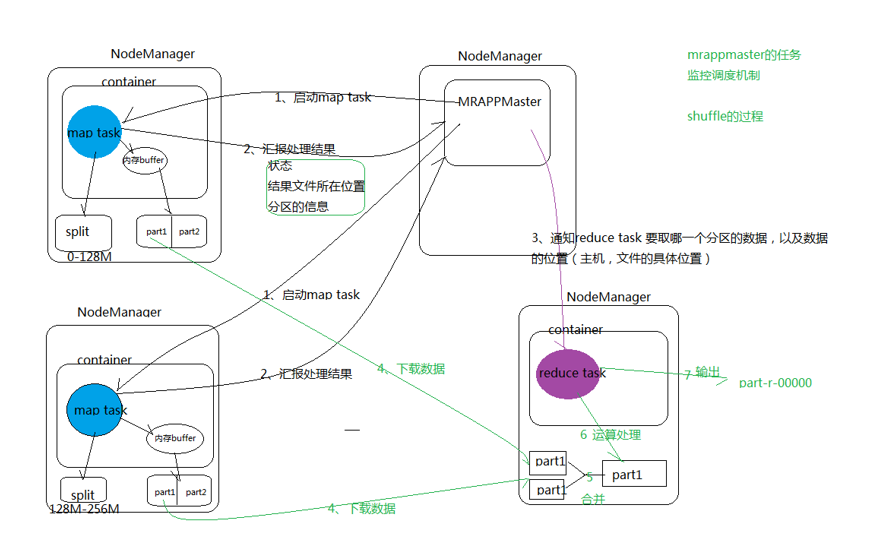
还需增加图片上面的显示：



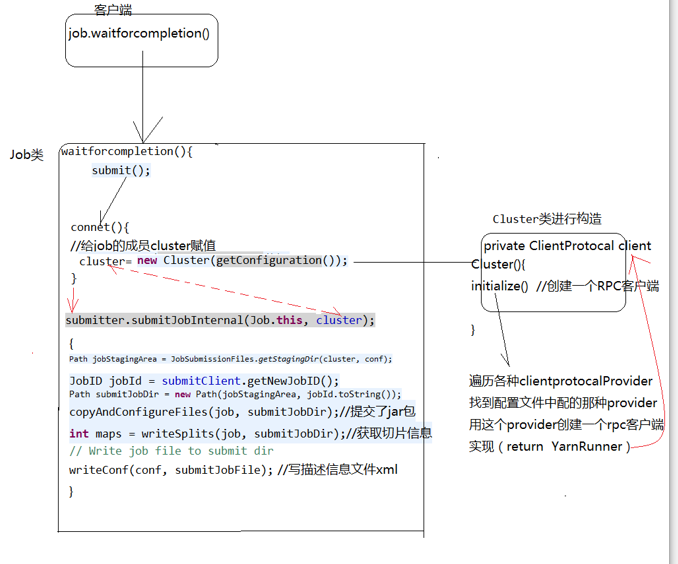
### 1.11.2 job提交逻辑和yarn框架调度机制



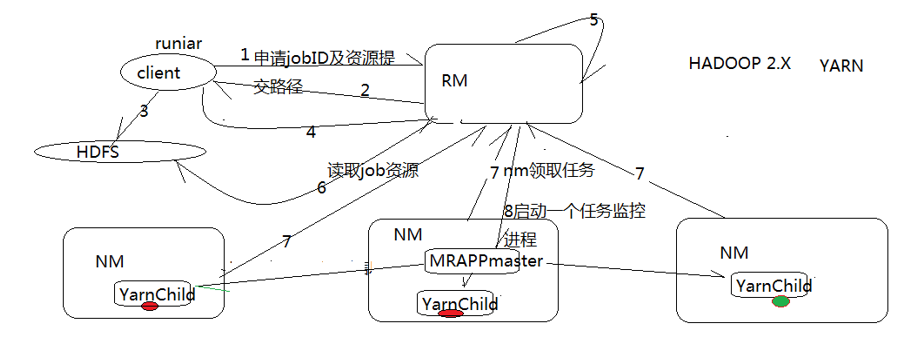
**Job提交补充**



Yarn提交job的源码流程：



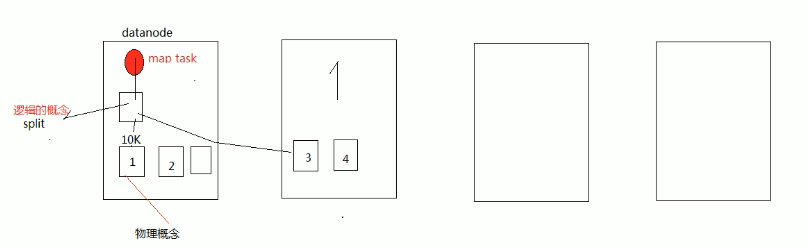
Yarn提交job的详细流程：



### 1.11.3 MR程序的几种提交运行模式

|  |
| --- |
| 本地模型运行  1/在windows的eclipse里面直接运行main方法，就会将job提交给本地执行器localjobrunner执行  ----输入输出数据可以放在本地路径下（c:/wc/srcdata/）  ----输入输出数据也可以放在hdfs中(hdfs://weekend110:9000/wc/srcdata)      2/在linux的eclipse里面直接运行main方法，但是不要添加yarn相关的配置，也会提交给localjobrunner执行  ----输入输出数据可以放在本地路径下（/home/hadoop/wc/srcdata/）  ----输入输出数据也可以放在hdfs中(hdfs://weekend110:9000/wc/srcdata)        集群模式运行  1/将工程打成jar包，上传到服务器，然后用hadoop命令提交 hadoop jar wc.jar cn.itcast.hadoop.mr.wordcount.WCRunner  2/在linux的eclipse中直接运行main方法，也可以提交到集群中去运行，但是，必须采取以下措施：  ----在工程src目录下加入 mapred-site.xml 和 yarn-site.xml  ----将工程打成jar包(wc.jar)，同时在main方法中添加一个conf的配置参数　conf.set("mapreduce.job.jar","wc.jar");  3/在windows的eclipse中直接运行main方法，也可以提交给集群中运行，但是因为平台不兼容，需要做很多的设置修改  ----要在windows中存放一份hadoop的安装包（解压好的）  ----要将其中的lib和bin目录替换成根据你的windows版本重新编译出的文件  ----再要配置系统环境变量 HADOOP\_HOME 和 PATH  ----修改YarnRunner这个类的源码 |

### 1.11.4 mr程序运行时，所运行的map任务数量



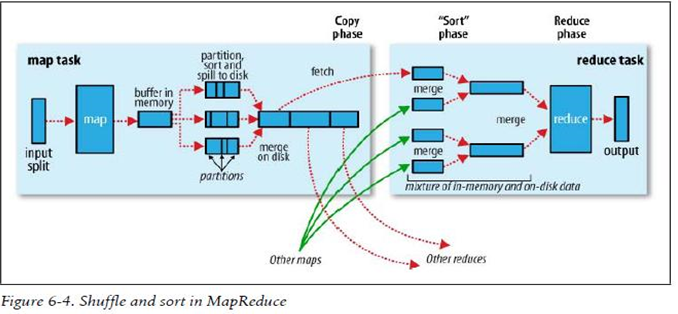
图形描述：

1.map task 的并发数是由切片的数量决定的，有多少个切片，就启动多少个map task

2. 切片是一个逻辑概念，指的就是文件中数据的偏移量范围

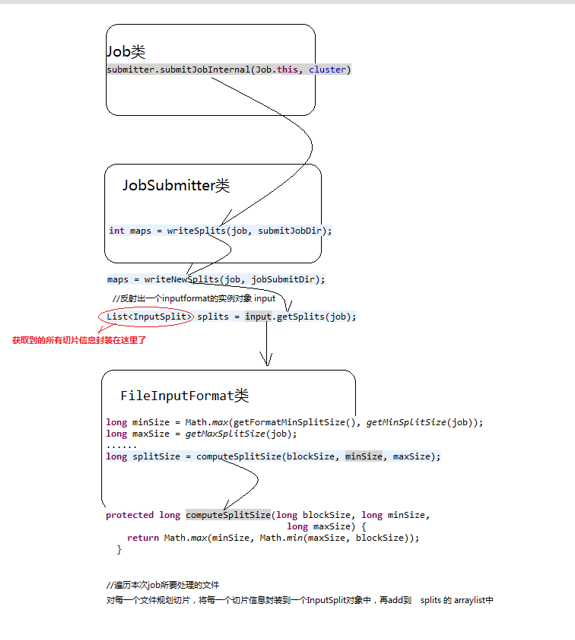
3. 切片的具体大小应该根据所处理的文件的大小来调整

### 1.11.5 Shuffle机制

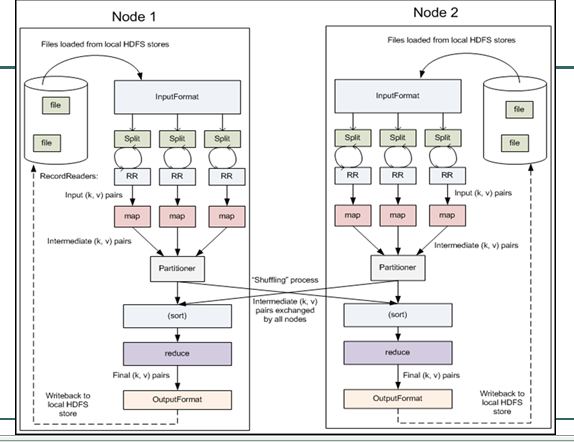


|  |
| --- |
| 1. 每个map有一个环形内存缓冲区，用于存储任务的输出。默认大小100MB（io.sort.mb属性），一旦达到阀值0.8（io.sort.spill.percent）,一个后台线程把内容写到(spill)磁盘的指定目录（mapred.local.dir）下的新建的一个溢出写文件。 2. 写磁盘前，要partition,sort。如果有combiner，combine排序后数据。 3. 等最后记录写完，合并全部溢出写文件为一个分区且排序的文件。 4. Reducer通过Http方式得到输出文件的分区。 5. TaskTracker为分区文件运行Reduce任务。复制阶段把Map输出复制到Reducer的内存或磁盘。一个Map任务完成，Reduce就开始复制输出。 6. 排序阶段合并map输出。然后走Reduce阶段。 |

切片代码流程：

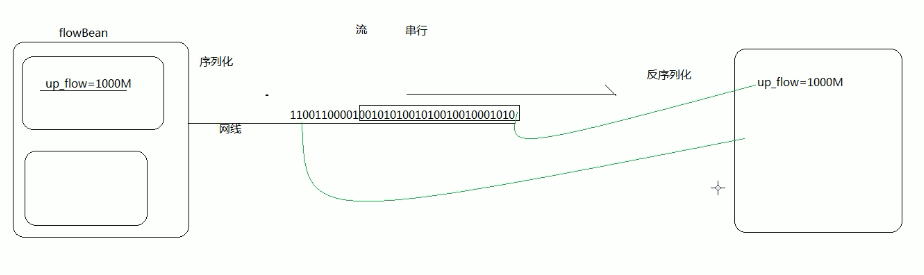


1.11.6 MapReduce组件的全貌



## 1.12 hadoop中的序列化机制

Hadoop中的序列化机制：



Jdk中的序列化机制：



**重点**：**jdk中自带的序列化机制，会传递一个对象的继承结构信息，会很完整的还原，**

**但hadoop中的序列化机制就不会传递对象的继承结构信息**

## 1.13 实例

### 1.13.1 流量求和mr程序开发

所用流量文件：



程序开发：



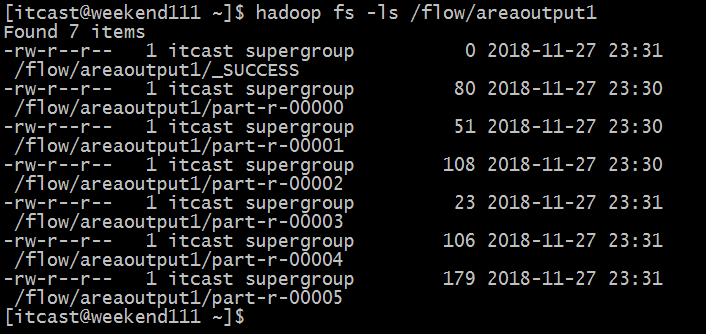
### 1.13.2 hadoop中自定义排序（流量大的往前或优先）

### 1.13.3 自定义分组



分组成功后展示：



### 1.13.4 倒排索引mr的实现

代码：

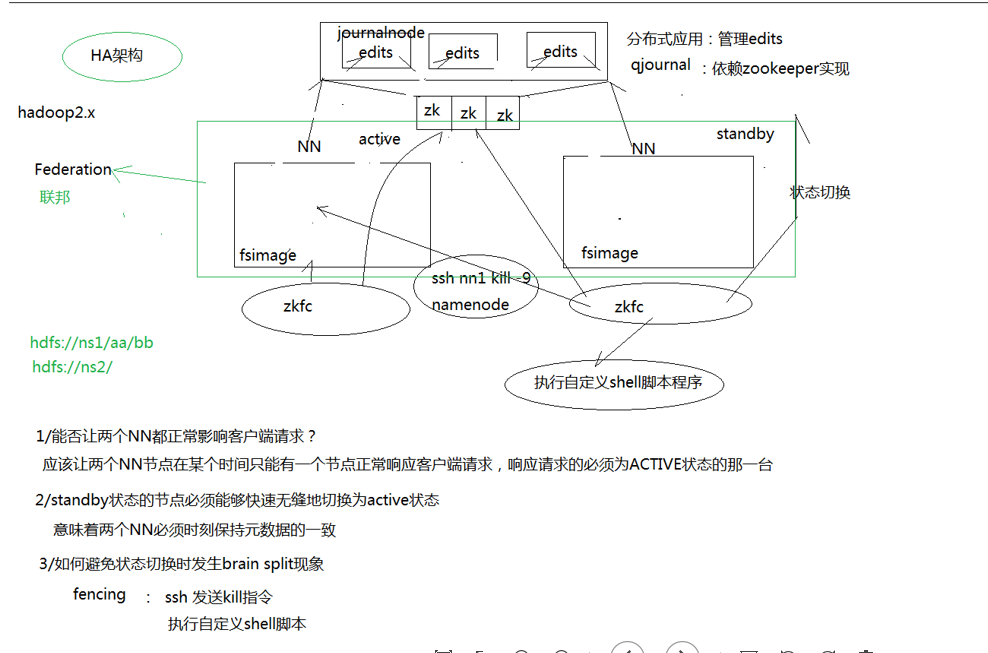


多个方法在一个main方法中的写法：

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  Configuration conf = **new** Configuration();  //先构造job\_one  Job job\_one = Job.*getInstance*(conf);  job\_one.setJarByClass(InverseIndexStepOne.**class**);  job\_one.setMapperClass(StepOneMapper.**class**);  job\_one.setReducerClass(StepOneReducer.**class**);  //.......  Job job\_two = Job.*getInstance*(conf);  job\_two.setJarByClass(InverseIndexStepTwo.**class**);    job\_two.setMapperClass(StepTwoMapper.**class**);  job\_two.setReducerClass(StepTwoReducer.**class**);  job\_two.setOutputKeyClass(Text.**class**);  job\_two.setOutputValueClass(Text.**class**);  //读取路径  FileInputFormat.*setInputPaths*(job\_two, **new** Path(args[0]));  //输出路径  //检查一下参数所指定的输出路径是否存在，如果已存在,先删除  Path outpath = **new** Path(args[1]);  FileSystem fs = FileSystem.*get*(conf);  **if**(fs.exists(outpath)) {  fs.delete(outpath, **true**);  }  FileOutputFormat.*setOutputPath*(job\_two, **new** Path(args[1]) );    //先提交job\_one执行  **boolean** one\_result = job\_one.waitForCompletion(**true**);  **if**(one\_result) {  System.***out***.println(job\_two.waitForCompletion(**true**)?0:1);  }  } |

## 1.14 HA机制

### 1.14.1 NN(NamNode)高可用方案：



### 1.14.2 hadoop集群HA模式部署

**HA分布式集群配置自己整理步骤：**

|  |
| --- |
| 1. 首先配置7台机器，在5、6、7、机器上面配置zookeeper集群   每台机器的配置环境如下：  第一台：namenode、zkfc  第二台：namenode、zkfc  第三台：resourceManager  第四台：resourceManager  第五台：zookeeper、journalnode、datanode、nodemanager  第六台：zookeeper、journalnode、datanode、nodemanager  第七台：zookeeper、journalnode、datanode、nodemanager  提示：datanode（数据存储）和nodemanager（管理计算）总是在一起的，在本地取计算的节点数据，可以起到优化的作用  每一台机器的准备工作，首先配置每台机器的普通用户具有sudo操作权限，然后配置id地址和主机名关联和修改主机名，java环境是必须的线配置好的和防火墙先关闭。   1. 安装zookeeper集群（在weekend05）   2.1解压  tar -zxvf zookeeper-3.4.5.tar.gz -C /weekend/  2.2修改配置  cd /weekend/zookeeper-3.4.5/conf/  cp zoo\_sample.cfg zoo.cfg  vim zoo.cfg  修改：dataDir=/weekend/zookeeper-3.4.5/tmp  在最后添加：  server.1=weekend05:2888:3888  server.2=weekend06:2888:3888  server.3=weekend07:2888:3888  保存退出  然后创建一个tmp文件夹  mkdir /weekend/zookeeper-3.4.5/tmp  再创建一个空文件  touch /weekend/zookeeper-3.4.5/tmp/myid  最后向该文件写入ID  echo 1 > /weekend/zookeeper-3.4.5/tmp/myid  2.3将配置好的zookeeper拷贝到其他节点(首先分别在weekend06、weekend07根目录下创建一个weekend目录：mkdir /weekend)  scp -r /weekend/zookeeper-3.4.5/ weekend06:/weekend/  scp -r /weekend/zookeeper-3.4.5/ weekend07:/weekend/    注意：修改weekend06、weekend07对应/weekend/zookeeper-3.4.5/tmp/myid内容  weekend06：  echo 2 > /weekend/zookeeper-3.4.5/tmp/myid  weekend07：  echo 3 > /weekend/zookeeper-3.4.5/tmp/myid   1. 在第一台机上传文件hadoop-2.4.1.tar.gz 并解压，   3.1然后修改hadoop-env.sh文件中的JAVA\_HOME，修改为jdk的确切路径  export JAVA\_HOME=/usr/local/java/jdk1.7.0\_65   * 1. 修改core-site.xml文件   按照hadoop2.4.1集群搭建.txt文件中进行配置   * 1. 修改hdfs-site.xml文件   按照hadoop2.4.1集群搭建.txt文件中进行配置  3.4修改mapred-site.xml文件  按照hadoop2.4.1集群搭建.txt文件中进行配置   * 1. 修改yarn-site.xml文件   按照hadoop2.4.1集群搭建.txt文件中进行配置   1. 七台机器配置无秘登录机制   生成密钥的命令：ssh-keygen -t rsa  使用start-dfs.sh命令启动nn（NameNode）和dn(dataNode)，需要启动wk01，wk02,wk05  Wk06,wk07，也就是需要配置这几台的无秘登录，生成wk01的密钥，然后拷贝到这5台机器上面，包括本机也需要设置无秘登录，命令：ssh-copy-id weekend07  重点：两个namenode之间要配置ssh免密码登陆，别忘了配置weekend02到weekend01的免登陆在weekend02上生产一对钥匙  ssh-keygen -t rsa  ssh-copy-id -i weekend01    使用start-yarn.sh命令启动rm(resourceManager)和nm(nodeManager),需要启动wk03,wk04,wk05,wk06,wk07,生成wk03的密钥，然后拷贝到除wk04的机器，因为wk04上面的resourceManager需要手动启动，start-yarn.sh无法自动启动    5． start-dfs.sh怎样确定datanode在哪里，需要修改slaves，至于start-yarn.sh去启动nodeManager，也需要去确定nodemanager在哪里，也需要修改slaves，所以这两个slaves可以不一样，但是现在因为datanode和nodemanager同时在一台机器上面，所以是相同的。  修改slaves文件在/home/hadoop/app/hadoop-2.4.1/etc/hadoop目录下，至于start-dfs.sh和start-yarn.sh启动需要分别配置slaves。  首先start-dfs.sh的配置：（在weekend01）    Start-yarn.sh的配置：（在weekend03上）   1. 复制配置好的hadoop   复制之前先去/home/hadoop/app/hadoop-2.4.1/share 目录下把doc文件删除，否则会复制变慢。  复制命令：  scp -r hadoop-2.4.1/ weekend02:/home/hadoop/app/  scp -r hadoop-2.4.1/ weekend03:/home/hadoop/app/  scp -r hadoop-2.4.1/ weekend04:/home/hadoop/app/  scp -r hadoop-2.4.1/ weekend05:/home/hadoop/app/  scp -r hadoop-2.4.1/ weekend06:/home/hadoop/app/  scp -r hadoop-2.4.1/ weekend07:/home/hadoop/app/   1. 启动集群，严格按照按照hadoop2.4.1集群搭建.txt文件中从2.5开始的步骤进行启动   重点：使用hdfs namenode –format 格式化之后，需要让这两个namenode格式化之后fsimage完全相同，所以需要把第一台格式化之后的文件拷贝过去。  scp -r tmp/ weekend02:/home/hadoop/app/hadoop-2.4.1/  或者在第二个namenode（本次配置的机器即weekend02）上面使用hdfs namenode –bootstrapStandby  启动start-yarn.sh在weekend03上启动，另一个resourceManager启动则不能使用start-yarn.sh，需要使用yarn-daemon.sh start resourcemanager启动 |

### 1.14.3 hadoop—hdfs动态增加节点和副本管理

手动切换HA中的状态：

强制转换nn1为standby

Hdfs haadmin –transitionToStandby nn1 --forcemanual

强制转换nn1为active：

Hdfs haadmin –transitionToActive nn1 --forcemanual

查看nn1节点状态：hdfs haadmin -getServiceState nn1

Datanode超时时间：



Hdfs冗余数据块自动删除：



如果想要观察这个动态增加节点和副本数量管理，需要修改hdfs-site.xml文件，新增

**datanode节点超时时间设置**

**<property>**

**<name>heartbeat.recheck.interval</name>**

**<value>2000</value>**

**</property>**

**<property>**

**<name>dfs.heartbeat.interval</name>**

**<value>1</value>**

**</property>**

**和HDFS冗余数据块的自动删除设置：**

**<property>**

**<name>dfs.blockreport.intervalMsec</name>**

**<value>10000</value>**

**<description>Determines block reporting interval in milliseconds.</description>**

**</property>**

然后把这个文件复制到其它机器上面之后，在新增加一台机器，配置hadoop环境启动datanode，不过可能会出现clusterID不一样的错误，需要删除配置的存放数据的文件，在重新启动，这个集群上面配置的存储的目录文件是tmp，删除tmp文件之后在重新启动。

启动之后，tmp里面保存的副本数据会直接存储在新增节点的tmp文件里面。

### 1.14.4 使用 java api 访问HA

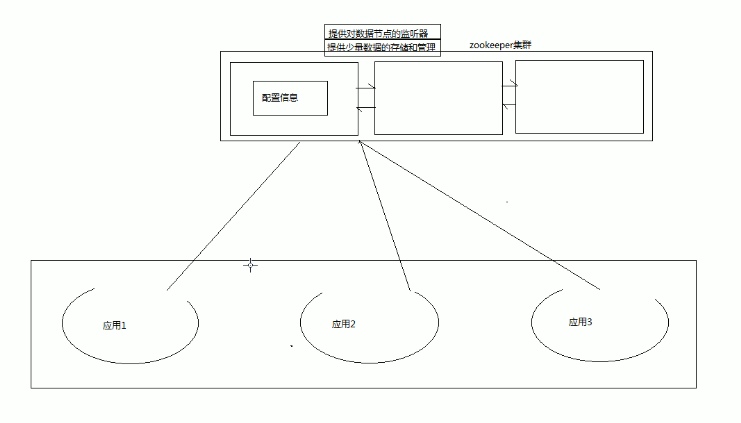
java代码：

|  |
| --- |
| Configuration conf = **new** Configuration();  FileSystem fs = FileSystem.*get*(**new** URI("hdfs://ns1/"), conf, "hadoop");  fs.copyFromLocalFile(**new** Path("g:/test.txt"), **new** Path("hdfs://ns1/")); |

因为不识别ns1，所以需要在src目录下增加两个配置文件hdfs-site.xml和core-site.xml

# zookeeper知识总结

## zookeeper集群图解：



## 2.1 zookeeper 概念

### 2.1.1 什么是zookeeper

|  |
| --- |
| * Zookeeper 是 Google 的 Chubby一个开源的实现，是 Hadoop 的分布式协调服务 * 它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等 |

### 2.1.2 为什么使用zookeeper

|  |
| --- |
| * 大部分分布式应用需要一个主控、协调器或控制器来管理物理分布的子进程（如资源、任务分配等） * 目前，大部分应用需要开发私有的协调程序，缺乏一个通用的机制 * 协调程序的反复编写浪费，且难以形成通用、伸缩性好的协调器 * ZooKeeper：提供通用的分布式锁服务，用以协调分布式应用 |

### 2.1.3 zookeeper能帮助我们做什么

|  |
| --- |
| * Hadoop2.0,使用Zookeeper的事件处理确保整个集群只有一个活跃的NameNode,存储配置信息等. * HBase,使用Zookeeper的事件处理确保整个集群只有一个HMaster,察觉HRegionServer联机和宕机,存储访问控制列表等. |

### 2.1.4 zookeeper的特性

|  |
| --- |
| * **Zookeeper是简单的** * **Zookeeper是富有表现力的** * **Zookeeper具有高可用性** * **Zookeeper采用松耦合交互方式** * **Zookeeper是一个资源库** |

### 2.1.5 zookeeper数据特性

zookeeper管理客户所存放的数据采用的是类似文件树的结构：

每一个节点叫做一个node

/order-anlalyze(<1M)/aa(info\_data<1M)

Zookeeper存活的机制，必须是大于等于配置的节点数的一半

**zookeeper的数据模型：**

|  |
| --- |
| * 层次化的目录结构，命名符合常规文件系统规范 * 每个节点在zookeeper中叫做znode,并且其有一个唯一的路径标识 * 节点Znode可以包含数据和子节点，但是EPHEMERAL类型的节点不能有子节点 * Znode中的数据可以有多个版本，比如某一个路径下存有多个数据版本，那么查询这个路径下的数据就需要带上版本 * 客户端应用可以在节点上设置监视器 * 节点不支持部分读写，而是一次性完整读写 |

**Zookeeper的节点：**

|  |
| --- |
| * Znode有两种类型，短暂的（ephemeral）和持久的（persistent） * Znode的类型在创建时确定并且之后不能再修改 * 短暂znode的客户端会话结束时，zookeeper会将该短暂znode删除，短暂znode不可以有子节点 * 持久znode不依赖于客户端会话，只有当客户端明确要删除该持久znode时才会被删除 * Znode有四种形式的目录节点，PERSISTENT、PERSISTENT\_SEQUENTIAL、EPHEMERAL、EPHEMERAL\_SEQUENTIAL |

**zookeeper的角色：**

|  |
| --- |
| * 领导者（leader），负责进行投票的发起和决议，更新系统状态 * 学习者（learner），包括跟随者（follower）和观察者（observer），follower用于接受客户端请求并想客户端返回结果，在选主过程中参与投票 * Observer可以接受客户端连接，将写请求转发给leader，但observer不参加投票过程，只同步leader的状态，observer的目的是为了扩展系统，提高读取速度 * 客户端（client），请求发起方 |

**zookeeper工作原理：**

|  |
| --- |
| * Zookeeper的核心是原子广播，这个机制保证了各个server之间的同步。实现这个机制的协议叫做Zab协议。Zab协议有两种模式，它们分别是恢复模式和广播模式。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数server的完成了和leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和server具有相同的系统状态。 * 一旦leader已经和多数的follower进行了状态同步后，他就可以开始广播消息了，即进入广播状态。这时候当一个server加入zookeeper服务中，它会在恢复模式下启动，发现leader，并和leader进行状态同步。待到同步结束，它也参与消息广播。Zookeeper服务一直维持在Broadcast状态，直到leader崩溃了或者leader失去了大部分的followers支持。 * 广播模式需要保证proposal被按顺序处理，因此zk采用了递增的事务id号(zxid)来保证。所有的提议(proposal)都在被提出的时候加上了zxid。实现中zxid是一个64为的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch。低32位是个递增计数。 * 当leader崩溃或者leader失去大多数的follower，这时候zk进入恢复模式，恢复模式需要重新选举出一个新的leader，让所有的server都恢复到一个正确的状态。 |

**ACL：**

|  |
| --- |
| * 身份验证模式有三种： * digest:用户名，密码 * host:通过客户端的主机名来识别客户端 * ip： 通过客户端的ip来识别客户端 * new ACL(Perms.READ,new Id("host","example.com"));   这个ACL对应的身份验证模式是host，符合该模式的身份是example.com，权限的组合是：READ |

## 2.2 zookeeper安装（单机和集群）

### 2.2.1 zookeeper单机安装

|  |
| --- |
| * 下载ZooKeeper：<http://labs.renren.com/apache-mirror/zookeeper/zookeeper-3.4.3/zookeeper-3.4.3.tar.gz> * 解压：tar xzf [zookeeper-3.4.3.tar.gz](http://labs.renren.com/apache-mirror/zookeeper/zookeeper-3.4.3/zookeeper-3.4.3.tar.gz) * 在conf目录下创建一个配置文件zoo.cfg，tickTime=2000 dataDir=/Users/zdandljb/zookeeper/data dataLogDir=/Users/zdandljb/zookeeper/dataLog clientPort=2181 * 启动ZooKeeper的Server：sh bin/zkServer.sh start, 如果想要关闭，输入：zkServer.sh stop |

注解：只运行在一台服务器上，适合测试环境；Zookeeper 的启动脚本在 bin 目录下；在启动脚本之前，还有几个基本的配置项需要配置一下， tickTime ：这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳；dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里；clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。当这些配置项配置好后，就可以启动 Zookeeper 了，启动后使用命令echo ruok | nc localhost 2181检查 Zookeeper 是否已经在服务

### 2.2.2 zookeeper集群安装：

**注解：**

|  |
| --- |
| Zookeeper 不仅可以单机提供服务，同时也支持多机组成集群来提供服务 , 实际上 Zookeeper 还支持另外一种伪集群的方式，也就是可以在一台物理机上运行多个 Zookeeper 实例；nitLimit：这个配置项是用来配置 Zookeeper 接受客户端（这里所说的客户端不是用户连接 Zookeeper 服务器的客户端，而是 Zookeeper 服务器集群中连接到 Leader 的 Follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 10 个心跳的时间（也就是 tickTime）长度后 Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是 5\*2000=10 秒；syncLimit：这个配置项标识 Leader 与 Follower 之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime 的时间长度，总的时间长度就是 2\*2000=4 秒；server.A=B：C：D：其中 A 是一个数字，表示这个是第几号服务器；B 是这个服务器的 ip 地址；C 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；D 表示的是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。如果是伪集群的配置方式，由于 B 都是一样，所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。除了修改 zoo.cfg 配置文件，集群模式下还要配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面就有一个数据就是 A 的值，Zookeeper 启动时会读取这个文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是那个 server。分别在3台机器上启动ZooKeeper的Server：sh bin/zkServer.sh start；运行于一个集群上，适合生产环境，这个计算机集群被称为一个“集合体”（ensemble）。Zookeeper通过复制来实现高可用性，只要集合体中半 数以上的机器处于可用状态，它就能够保证服务继续。为什么一定要超过半数呢？这跟Zookeeper的复制策略有关：zookeeper确保对znode 树的每一个修改都会被复制到集合体中超过半数的机器上。 |

**配置：**

|  |
| --- |
| * 创建myid文件，server1机器的内容为：1，server2机器的内容为：2，server3机器的内容为：3 * 在conf目录下创建一个配置文件zoo.cfg，tickTime=2000   **注意：**dataDir的所指定的目录需要先创建好  dataDir=/Users/zdandljb/zookeeper/data  dataLogDir=/Users/zdandljb/zookeeper/dataLog  clientPort=2181  initLimit=5  syncLimit=2  server.1=server1:2888:3888  server.2=server2:2888:3888  server.3=server3:2888:3888 |

两台虚拟机之间的传递复制文件命令：

scp -r zookeeper-3.4.5/ 192.168.93.5:/home/hadoop/app



## zookeeper应用场景

### 场景一：统一命名服务

|  |
| --- |
| * 分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。 * Name Service 是 Zookeeper 内置的功能，只要调用 Zookeeper 的 API 就能实现 |

### 场景二：配置管理

|  |
| --- |
| * 配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。 * 将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。 |

### 2.3.3场景三：集群管理

|  |
| --- |
| * Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。 * Zookeeper 不仅能够维护当前的集群中机器的服务状态，而且能够选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能 Leader Election      * zk.create("/testRootPath/testChildPath1","1".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL); * zk.create(“/testRootPath/testChildPath2”,“2”.getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL); * zk.create("/testRootPath/testChildPath3","3".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL); * zk.create("/testRootPath/testChildPath4","4".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL); * System.out.println(zk.getChildren("/testRootPath", false)); * 打印结果：[testChildPath10000000000, testChildPath20000000001, testChildPath40000000003, testChildPath30000000002]   规定编号最小的为master,所以当我们对SERVERS节点做监控的时候，得到服务器列表，只要所有集群机器逻辑认为最小编号节点为master，那么master就被选出，而这个master宕机的时候，相应的znode会消失，然后新的服务器列表就被推送到客户端，然后每个节点逻辑认为最小编号节点为master，这样就做到动态master选举 |

### 场景四：共享锁

|  |
| --- |
| * 共享锁在同一个进程中很容易实现，但是在跨进程或者在不同 Server 之间就不好实现了。Zookeeper 却很容易实现这个功能，实现方式也是需要获得锁的 Server 创建一个 EPHEMERAL\_SEQUENTIAL 目录节点，然后调用 getChildren方法获取当前的目录节点列表中最小的目录节点是不是就是自己创建的目录节点，如果正是自己创建的，那么它就获得了这个锁，如果不是那么它就调用 exists(String path, boolean watch) 方法并监控 Zookeeper 上目录节点列表的变化，一直到自己创建的节点是列表中最小编号的目录节点，从而获得锁，释放锁很简单，只要删除前面它自己所创建的目录节点就行了。 |

### 场景五：队列管理

|  |
| --- |
| * Zookeeper 可以处理两种类型的队列：当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列；队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型 * 创建一个父目录 /synchronizing，每个成员都监控目录 /synchronizing/start 是否存在，然后每个成员都加入这个队列（创建 /synchronizing/member\_i 的临时目录节点），然后每个成员获取 / synchronizing 目录的所有目录节点，判断 i 的值是否已经是成员的个数，如果小于成员个数等待 /synchronizing/start 的出现，如果已经相等就创建 /synchronizing/start。 |

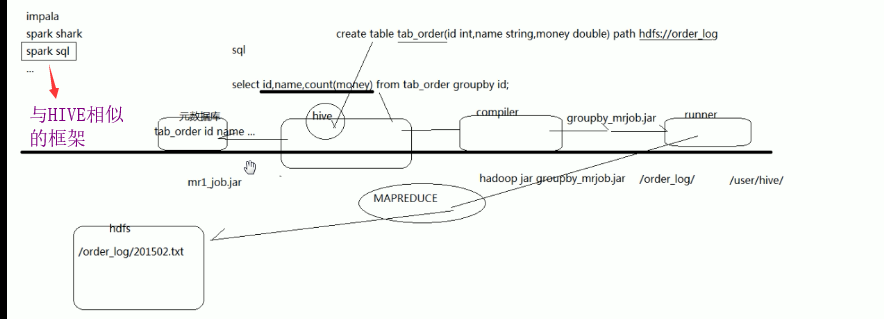
### 总结：

|  |
| --- |
| * Zookeeper 作为 Hadoop 项目中的一个子项目，是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，如它管理 Hadoop 集群中的 NameNode，还有 Hbase 中 Master Election、Server 之间状态同步等。 * Zoopkeeper 提供了一套很好的分布式集群管理的机制，就是它这种基于层次型的目录树的数据结构，并对树中的节点进行有效管理，从而可以设计出多种多样的分布式的数据管理模型 |

# Hive知识点

## Hive概述

### 3.1.1什么是Hive？



|  |
| --- |
| * Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL ），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 QL ，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。 * Hive是SQL解析引擎，它将SQL语句转译成M/R Job然后在Hadoop执行。 * Hive的表其实就是HDFS的目录/文件，按表名把文件夹分开。如果是分区表，则分区值是子文件夹，可以直接在M/R Job里使用这些数据。 |

### Hive 系统架构

|  |
| --- |
| 图例：     * 用户接口主要有三个：CLI，JDBC/ODBC和 WebUI  1. CLI，即Shell命令行 2. JDBC/ODBC 是 Hive 的Java，与使用传统数据库JDBC的方式类似 3. WebGUI是通过浏览器访问 Hive  * Hive 将元数据存储在数据库中(metastore)，目前只支持 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等 * 解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划（plan）的生成。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行 * Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 \* 的查询，比如 select \* from table 不会生成 MapRedcue 任务） |

## hive应用

### hive安装启动

#### 3.2.1.1 普通的安装配置：

该配置会使用hive的默认配置在不同的地方启动会生成不同的数据库文件，不能进行通用。

第一步：直接上传hive-0.12.0.tar.gz文件，并解压到相应的文件夹就可以直接启动使用

第二步：启动，进入/home/hadoop/app/hive-0.12.0/bin文件夹，使用./hive启动

#### 3.2.1.2 hive的元数据库mysql方式的安装配置：

|  |
| --- |
| 第一步：直接上传hive-0.12.0.tar.gz文件，并解压到相应的文件夹就可以直接启动使用  第二步：然后进入/home/hadoop/app/hive-0.12.0/conf文件夹，复制cp hive-default.xml.template hive-site.xml 文件  第三步：查看是否安装mysql数据库  安装mysql数据库：首先先切换到root用户，然后上传MySQL-server-5.1.73-1.glibc23.i386.rpm  MySQL-client-5.1.73-1.glibc23.i386.rpm 两个文件。  配置mysql metastore（切换到root用户）  配置HIVE\_HOME环境变量  rpm -qa | grep mysql  rpm -e mysql-libs-5.1.66-2.el6\_3.i686 --nodeps  rpm -ivh MySQL-server-5.1.73-1.glibc23.i386.rpm  rpm -ivh MySQL-client-5.1.73-1.glibc23.i386.rpm  修改mysql的密码  /usr/bin/mysql\_secure\_installation（这一步注意使用翻译工具翻译英文，当然英文大佬就免了。）  （注意：删除匿名用户，允许用户远程连接）  登陆mysql  mysql -u root -p  第四步：新建hive-site.xml文件：增加如下内容  <configuration>  <property>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:mysql://weekend01:3306/hive?createDatabaseIfNotExist=true</value>  </property>  <property>  <name>javax.jdo.option.ConnectionDriverName</name>  <value>com.mysql.jdbc.Driver</value>  </property>  <property>  <name>javax.jdo.option.ConnectionUserName</name>  <value>root</value>  </property>  <property>  <name>javax.jdo.option.ConnectionPassword</name>  <value>root</value>  </property>  </configuration>  第五步：安装hive和mysql完成后，将mysql的连接jar包拷贝到$HIVE\_HOME/lib目录下，如果出现没有权限的问题，在mysql授权(在安装mysql的机器上执行)  mysql -uroot -p  #(执行下面的语句 \*.\*:所有库下的所有表 %：任何IP地址或主机都可以连接)  GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY 'root' WITH GRANT OPTION;  FLUSH PRIVILEGES;  第六步：建表(默认是内部表)  create table trade\_detail(id bigint, account string, income double, expenses double, time string) row format delimited fields terminated by '\t';  建分区表  create table td\_part(id bigint, account string, income double, expenses double, time string) partitioned by (logdate string) row format delimited fields terminated by '\t';  建外部表  create external table td\_ext(id bigint, account string, income double, expenses double, time string) row format delimited fields terminated by '\t' location '/td\_ext';  第七步：创建分区表  普通表和分区表区别：有大量数据增加的需要建分区表  create table book (id bigint, name string) partitioned by (pubdate string) row format delimited fields terminated by '\t';  分区表加载数据  load data local inpath './book.txt' overwrite into table book partition (pubdate='2010-08-22');  load data local inpath '/root/data.am' into table beauty partition (nation="USA");  select nation, avg(size) from beauties group by nation order by avg(size); |

### hive相应操作

**数据库操作：**

hive语法与sql语法相近，但是插入操作不同。

创建数据库：create database wk01;

使用数据库:use wk01;

创建表结构：create table t\_first(id int,name string,salary string);

但字段类型类似与java

**表操作：hive不支持单条数据插入**

**创建表：（该表为内部表的创建）**

**create table t\_order(id int, name string,rongliang string,price double)**

**row format delimited --一行数据为一条**

**fields terminated by '\t'; -- 每个字段以制表符分隔开 注意不要用空格而是使用tab键**

**数据：**

****

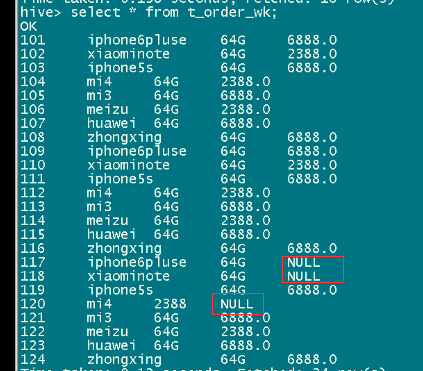
**加载数据：load data local inpath '/home/hadoop/hivetestdata/xxx.data' into table t\_order\_wk;**

**可以使用select \* from t\_order\_wk;查看导入的数据。**

**也可以直接使用hadoop文件上传的方法，上传到/user/hive/warehouse/wk01.db/t\_order\_wk目录下一样可以。（但是必须相应的字段必须对应保证每个字段都存在数据）**

**上传之后使用select \* from t\_order\_wk;查看。**

**如果字段不对应，那么最后面的字段就会为NULL，如果字段多出来，后面多出来的字段就不会取出来。**



### hive的常用操作

1. 建表(默认是建内部表)：

**create table t\_order(id int, name string,rongliang string,price double)**

**row format delimited --一行数据为一条**

**fields terminated by '\t';**

**加载本地数据：load data local inpath '/home/hadoop/hivetestdata/xxx.data' into table t\_order\_wk;**

加载集群中的数据：**load data inpath '/uuu.data' into table t\_order\_wk;**

**重点：内部表在导入集群中的文件数据时，会把集群中的那个文件移动到导入的表文件的目录下，会造成集群文件系统紊乱；外部表在指定导入的文件数据时，不会移动文件在集群中原来的位置，而且所指定的表的空间目录下也不会有任何改变。**

1. 建外部表：

create external table t\_order(id int, name string,rongliang string,price double)

row format delimited fields terminated by '\t';

localtion '/hive\_ext';

**外部表和内部表删除表时的区别：当内部表删除表结构时，内部表的目录结构会在集群中被删除，而且内部表文件中的原先的文件数据也会一起被删除；但是外部表删除的时候，只是会把数据库中的表结构删除掉，但是不会删除集群中的文件数据。**

1. 以查询的方式建表结构：(根据select语句建表结构)用于创建一些临时表存放中间结果

create table t\_order\_simple

as

select id,name,price from t\_order\_wk;

1. 批量插入数据:（insert）不支持单条的数据插入，但是支持批量的数据插入

create table tab\_ip\_like like tab\_ip; --insert批量插入时，该表必须先存在

insert overwrite table tab\_ip\_like

select \* from tab\_ip;

1. 分区表（partition）：

create table t\_order\_pt(id int, name string,rongliang string,price double)

row format delimited fields terminated by '\t'

partitioned by (month string);

增加分区：ALTER TABLE tab\_cts ADD PARTITION (partCol = 'dt') location '/external/hive/dt';

分区表导入数据：**load data local inpath '/home/hadoop/hivetestdata/xxx.data' into table t\_order\_pt partition(month=’201812’);**

查询：select \* from t\_order\_pt where month=’201812’;--相当于把分区当成一个字段来使用。

### hql语法及自定义函数

1. //write to hdfs 将结果写入到hdfs的文件中

insert overwrite local directory '/home/hadoop/hivetemp/test.txt' select \* from tab\_ip\_part where part\_flag='part1';

insert overwrite directory '/hiveout.txt' select \* from tab\_ip\_part where part\_flag='part1';

1. //array

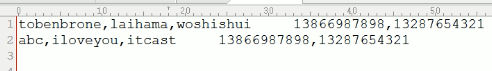
create table tab\_array(a array<int>,b array<string>)

row format delimited

fields terminated by '\t'

collection items terminated by ',';

数据类型:



3. //map

create table tab\_map(name string,info map<string,string>)

row format delimited

fields terminated by '\t'

collection items terminated by ','

map keys terminated by ':';

数据类型：



1. //cli shell 通过shell执行hive的hql语句

hive -S -e 'select country,count(\*) from tab\_ext' > /home/hadoop/hivetemp/e.txt

有了这种执行机制，就使得我们可以利用脚本语言(bash shell,python)进行hql语句进行批量执行。

select \* from tab\_ext sort by id desc limit 5;

select a.ip,b.book from tab\_ext a join tab\_ip\_book b on(a.name=b.name);

5. //UDF 自定义函数

select if(id=1,first,no-first),name from tab\_ext;

hive>add jar /home/hadoop/hiveareaudf.jar; --上传打包的jar包

hive> create temporary function getarea(所起的函数名称) as 'cn.itcast.bigdata.PhoneNbrToArea'(所写的java类的全路径);

select id,name,ip,fanyi(country) from tab\_ip\_ext;

实现:

|  |
| --- |
| 原始数据：  1388766545 200 300  1399766545 200 300  1344766545 200 300  1366766545 200 300Select getarea(phonenbr),upflow,downflow from t\_flow;  经过函数改变之后的数据：  1388766545 beijing 200 300  1399766545 tianjing 200 300  1344766545 shanghai 200 300  1366766545 nanjing 200 300  写一个java类，定义上述的函数逻辑打成jar包上传到hive中  Java类：    在hive中创建一个函数getarea，跟jar包中的自定义java类建立联系。  创建表：create table t\_flow(phonenbr string,upflow int,downflow int)  > row format delimited fields terminated by '\t';  add jar /home/hadoop/hiveareaudf.jar;  create temporary function getarea as 'cn.itcast.bigdata.PhoneNbrToArea';  select getarea(phonenbr),upflow,downflow from t\_flow; |

6. //CLUSTER <--相对高级一点，你可以放在有精力的时候才去学习>

create table tab\_ip\_cluster(id int,name string,ip string,country string)

clustered by(id) into 3 buckets;

load data local inpath '/home/hadoop/ip.txt' overwrite into table tab\_ip\_cluster;

set hive.enforce.bucketing=true;

insert into table tab\_ip\_cluster select \* from tab\_ip;

select \* from tab\_ip\_cluster tablesample(bucket 2 out of 3 on id);

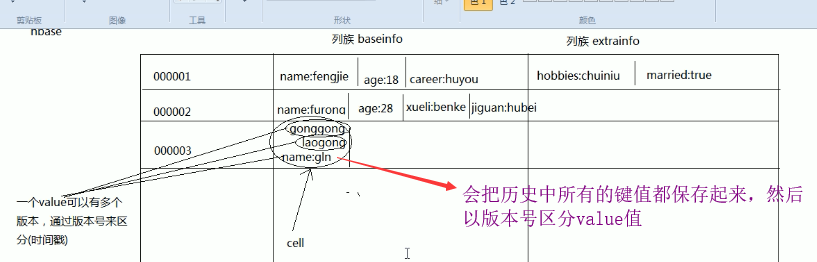
# hbase知识

## 4.1 Hbase简介：

### 4.1.1 什么是Hbase

* HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。HBase利用Hadoop HDFS作为其文件存储系统，利用Hadoop MapReduce来处理HBase中的海量数据，利用Zookeeper作为协调工具。

### 4.1.2 hbase表结构



Hbase表结构：建表时，不需要限定表中的字段，只需要指定若干个列族

插入数据时，列族中可以存储任意多个列(KV,列名&列值)

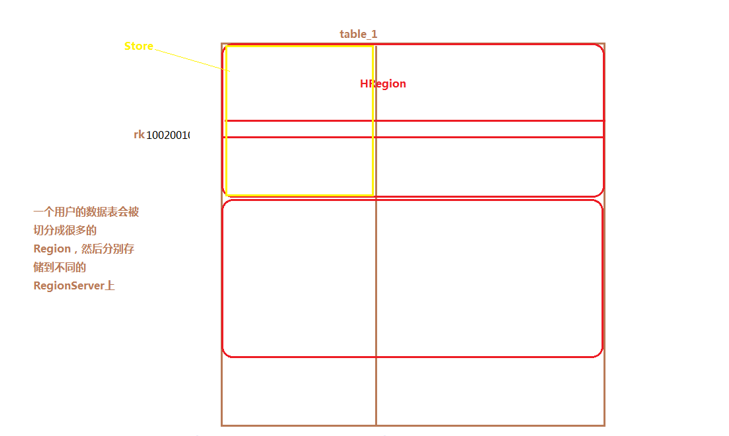
要查询某一个具体字段的值，需要指定的坐标：表名---->行键---->列族(ColumnFamily)：列名（qualifer）----->版本

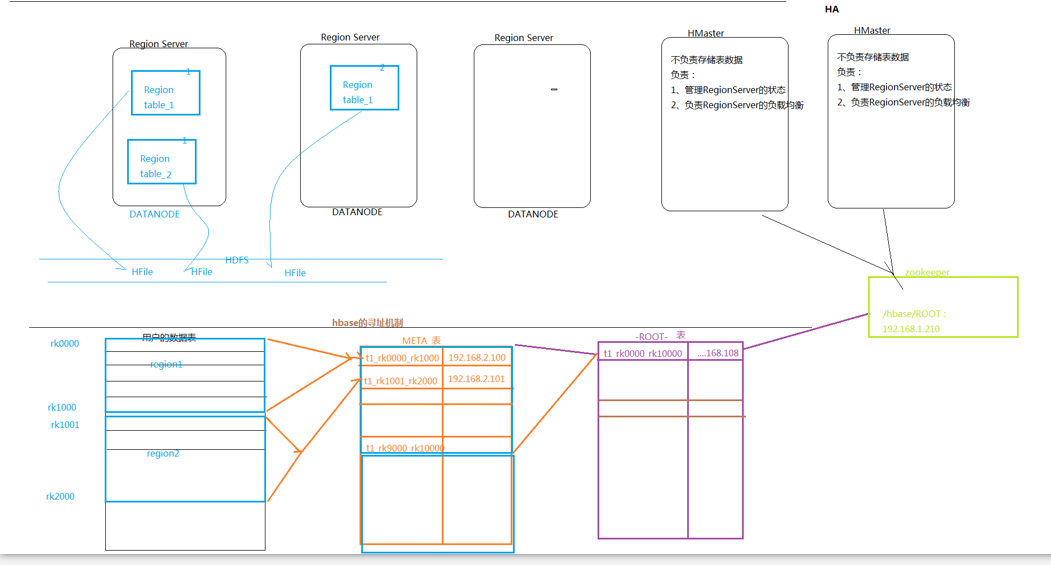
### 4.1.3 Hbase基础知识

|  |
| --- |
| * **架构体系**   + Client 包含访问hbase 的接口，client 维护着一些cache 来加快对hbase 的访问，比如regione 的位置信息   + Zookeeper     - 保证任何时候，集群中只有一个running master     - 存贮所有Region 的寻址入口     - 实时监控Region Server 的状态，将Region server 的上线和下线信息，实时通知给Master     - 存储Hbase 的schema,包括有哪些table，每个table 有哪些column family   + Master 可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master运行     - 为Region server 分配region     - 负责region server 的负载均衡     - 发现失效的region server 并重新分配其上的region   Region Server   * 维护Master 分配给它的region，处理对这些region 的IO 请求 * 负责切分在运行过程中变得过大的region   可以看出，client 访问hbase 上数据的过程并不需要master 参与，寻址访问先zookeeper再regionserver，数据读写访问regioneserver。HRegionServer主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。 |

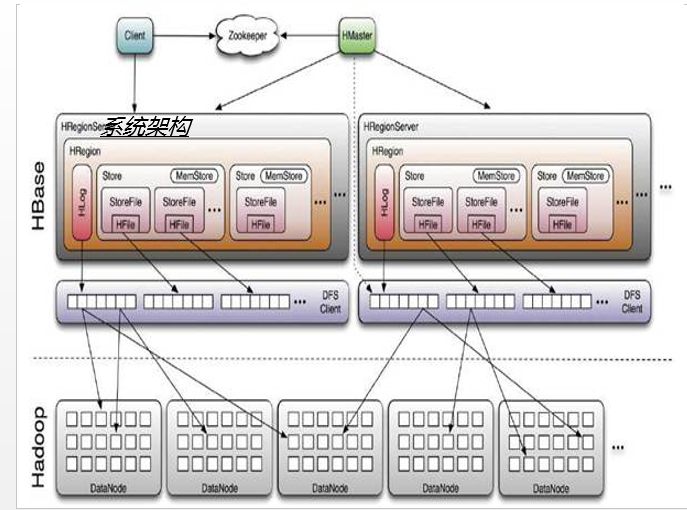
## hbase集群架构及表存储机制

hbase集群结构：（自己画的简单的）





Hbase集群结构(完整版)：



### 4.2.1 Hbase集群搭建步骤：



在weekend03和weekend04机器上面安装Hbase；

在weekend05，weekend06，weekend07上面安装regionServer；

|  |
| --- |
| Hbase集群安装：   1. 首先上传Hbase安装包，选择的安装包要和安装好的hadoop版本相对应。 2. 解压上传的安装包 3. 然后修改hbase-env.sh文件：   修改JAVA\_HOME路径为本机安装jdk的路径  export JAVA\_HOME=/usr/local/java/jdk1.7.0\_65  修改export HBASE\_MANAGES\_ZK的值，改为false，因为使用的不是自带的zookeeper  export HBASE\_MANAGES\_ZK=false   1. 修改hbase-site.xml文件：相应的zookeeper集群地址需要正确配置   <configuration>  <!-- 指定hbase在HDFS上存储的路径 -->  <property>  <name>hbase.rootdir</name>  <value>hdfs://ns1/hbase</value>  </property>  <!-- 指定hbase是分布式的 -->  <property>  <name>hbase.cluster.distributed</name>  <value>true</value>  </property>  <!-- 指定zk的地址，多个用“,”分割 -->  <property>  <name>hbase.zookeeper.quorum</name>  <value>weekend05:2181,weekend06:2181,weekend07:2181</value>  </property>  </configuration>   1. 修改regionservers文件，在该文件中增加zookeeper集群的地址   weekend05  weekend06  weekend07   1. 复制hadoop中的core-site.xml和hdfs-site.xml两个文件，在/home/hadoop/app/hadoop-2.4.1/etc/hadoop目录下，   复制Hbase目录的conf目录/home/hadoop/app/hbase-0.96.2-hadoop2/conf  cp /home/hadoop/app/hadoop-2.4.1/etc/hadoop/{core-site.xml,hdfs-site.xml} ./   1. 可以先删除hbase-0.96.2-hadoop2目录下的docs文件夹，在拷贝   scp -r hbase-0.96.2-hadoop2/ weekend04:/home/hadoop/app  scp -r hbase-0.96.2-hadoop2/ weekend05:/home/hadoop/app  scp -r hbase-0.96.2-hadoop2/ weekend06:/home/hadoop/app  scp -r hbase-0.96.2-hadoop2/ weekend07:/home/hadoop/app   1. 启动hbase   在/home/hadoop/app/hbase-0.96.2-hadoop2/bin目录下执行./start-hbase.sh启动  在另一台机器上面启动master命令：  ./hbase-demon.sh start master  单独启动regionserver服务：  /home/hadoop/app/hbase-0.96.2-hadoop2/bin/hbase-daemon.sh start regionserver    Hbase的web访问页面：主机地址:60010 例：192.168.93.103:60010 |

### 4.2.2 Hbase-shell 知识点

1. 开启hbase-shell 命令行模式 ./hbase shell

2. 创建表：create 'mygirls' ,{NAME => 'base\_info', VERSIONS => 3},{NAME => 'extra\_info'}

3. 查询表：list

4.查看表结构：describe 'mygirls'

5.删除表：首先使用disable 'mygirls',然后使用drop 'mygirls'删除。

6. 插入数据：put 'mygirls','0001','base\_info:name','fengjie'

7.获取表中的数据：get 'mygirls', '0003'

8.查询全表的数据：scan 'mygirls'

9.查询所有的记录包括以前的10个版本查询：scan 'mygirls', {RAW => true, VERSIONS => 10}

### hbase使用java api操作

自己理解：



标准写法：



# storm 知识

## 5.1 storm 简介

### 5.1.1 storm 特点：

|  |
| --- |
| Storm是一个开源的分布式实时计算系统，可以简单、可靠的处理大量的数据流。被称作“实时的hadoop”。Storm有很多使用场景：如实时分析，在线机器学习，持续计算， 分布式RPC，ETL等等。Storm支持水平扩展，具有高容错性，保证每个消息都会得到处理，而且处理速度很快（在一个小集群中，每个结点每秒可以处理 数以百万计的消息）。Storm的部署和运维都很便捷，而且更为重要的是可以使用任意编程语言来开发应用。  编程模型简单  　　在大数据处理方面相信大家对hadoop已经耳熟能详，基于Google Map/Reduce来实现的Hadoop为开发者提供了map、reduce原语，使并行批处理程序变得非常地简单和优美。  同样，Storm也为大数据 的实时计算提供了一些简单优美的原语，这大大降低了开发并行实时处理的任务的复杂性，帮助你快速、高效的开发应用。  可扩展  　　在Storm集群中真正运行topology的主要有三个实体：工作进程、线程和任务。Storm集群中的每台机器上都可以运行多个工作进程，每个 工作进程又可创建多个线程，每个线程可以执行多个任务，任务是真正进行数据处理的实体，我们开发的spout、bolt就是作为一个或者多个任务的方式执 行的。  因此，计算任务在多个线程、进程和服务器之间并行进行，支持灵活的水平扩展。  高可靠性  　　Storm可以保证spout发出的每条消息都能被“完全处理”，这也是直接区别于其他实时系统的地方，如S4。  　　spout发出的消息后续可能会触发产生成千上万条消息，可以形象的理解为一棵消息树，其中spout发出的消息为树根，Storm会跟踪 这棵消息树的处理情况，只有当这棵消息树中的所有消息都被处理了，Storm才会认为spout发出的这个消息已经被“完全处理”。如果这棵消息树中的任 何一个消息处理失败了，或者整棵消息树在限定的时间内没有“完全处理”，那么spout发出的消息就会重发。  高容错性  　　如果在消息处理过程中出了一些异常，Storm会重新安排这个出问题的处理单元。Storm保证一个处理单元永远运行（除非你显式杀掉这个处理单元）。  　　当然，如果处理单元中存储了中间状态，那么当处理单元重新被Storm启动的时候，需要应用自己处理中间状态的恢复。  Storm集群和Hadoop集群表面上看很类似。Hadoop上运行的是MapReduce jobs，而在Storm上运行的是拓扑（topology）；Hadoop擅长于分布式离线批处理，而Storm设计为支持分布式实时计算；Hadoop新的spark组件提供了在hadoop平台上运行storm的可能性 |

### storm 的基本概念

|  |
| --- |
| 在深入理解Storm之前，需要了解一些概念：  Topologies ： 拓扑，也俗称一个任务  Spouts ： 拓扑的消息源  Bolts ： 拓扑的处理逻辑单元  tuple：消息元组  Streams ： 流  Stream groupings ：流的分组策略  Tasks ： 任务处理单元  Executor :工作线程  Workers ：工作进程  Configuration ： topology的配置 |

### 5.1.3 storm 与 hadoop对比

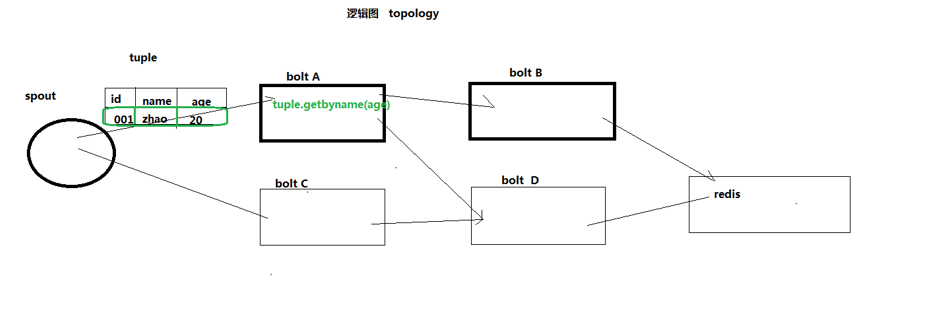
|  |
| --- |
| Topology 与 Mapreduce  一个关键的区别是： 一个MapReduce job最终会结束， 而一个topology永远会运行（除非你手动kill掉）  Nimbus 与 ResourManager  在Storm的集群里面有两种节点： 控制节点（master node）和工作节点（worker node）。控制节点上面运行一个叫Nimbus后台程序，它的作用类似Hadoop里面的JobTracker。Nimbus负责在集群里面分发代码，分配计算任务给机器， 并且监控状态。  Supervisor (worker进程)与NodeManager(YarnChild)  每一个工作节点上面运行一个叫做Supervisor的节点。Supervisor会监听分配给它那台机器的工作，根据需要启动/关闭工作进程。每一个工作进程执行一个topology的一个子集；一个运行的topology由运行在很多机器上的很多工作进程组成。 |

## storm体系架构

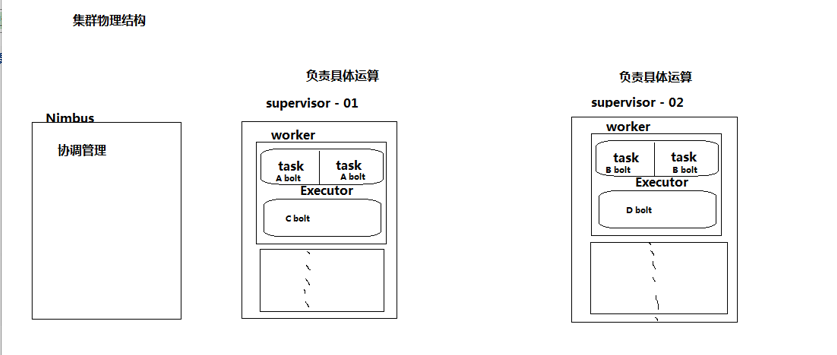
### 5.2.1 storm 中的Nimbus和Supervisor

|  |
| --- |
| Nimbus和Supervisor之间的所有协调工作都是通过Zookeeper集群完成。  Nimbus进程和Supervisor进程都是快速失败（fail-fast)和无状态的。所有的状态要么在zookeeper里面， 要么在本地磁盘上。  这也就意味着你可以用kill -9来杀死Nimbus和Supervisor进程， 然后再重启它们，就好像什么都没有发生过。这个设计使得Storm异常的稳定。 |

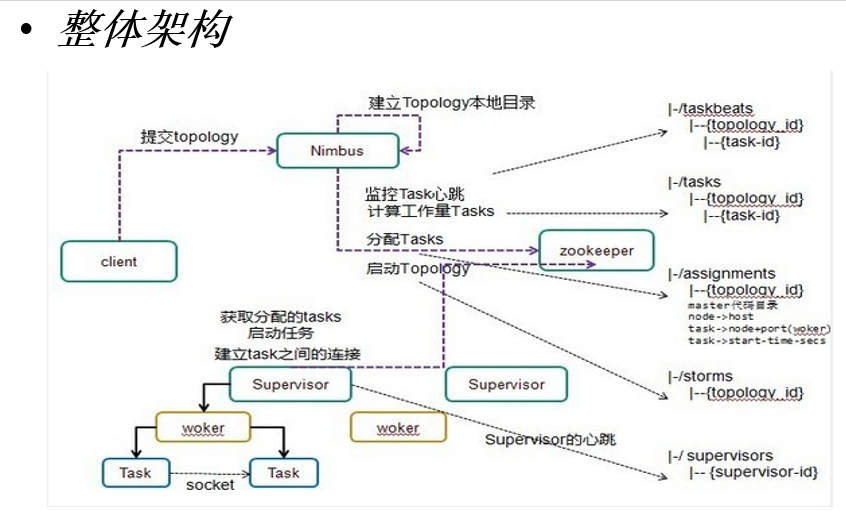
### topology 逻辑图



### 集群物理结构：



### 5.2.4整体架构



## storm集群安装

资料文档安装步骤：



按照这样的方式配置，是可以动态负载的

注意配置的时候，一定要注意空格，前面留一个空格

启动storm 程序：

启动方式一：

启动nimbus主机：

在/home/hadoop/app/storm092/bin目录下直接使用./storm nimbus

在/home/hadoop/app/storm092/bin目录下直接使用./storm ui 开启网页访问

http://192.168.93.101:8080/index.html

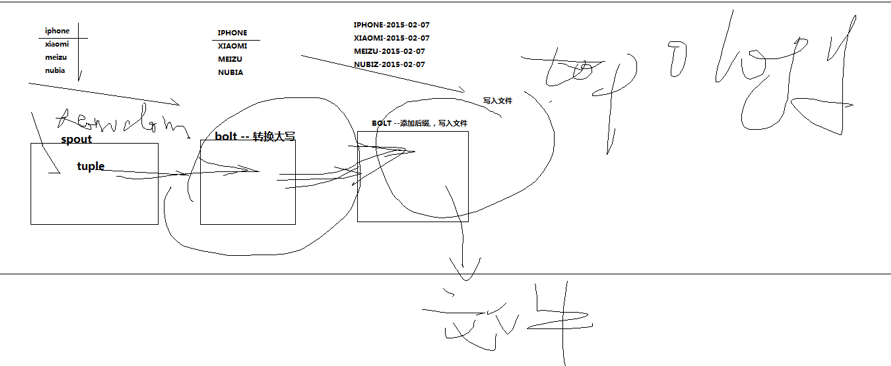
启动supervisor主机：

在/home/hadoop/app/storm092/bin目录下直接使用./storm supervisor

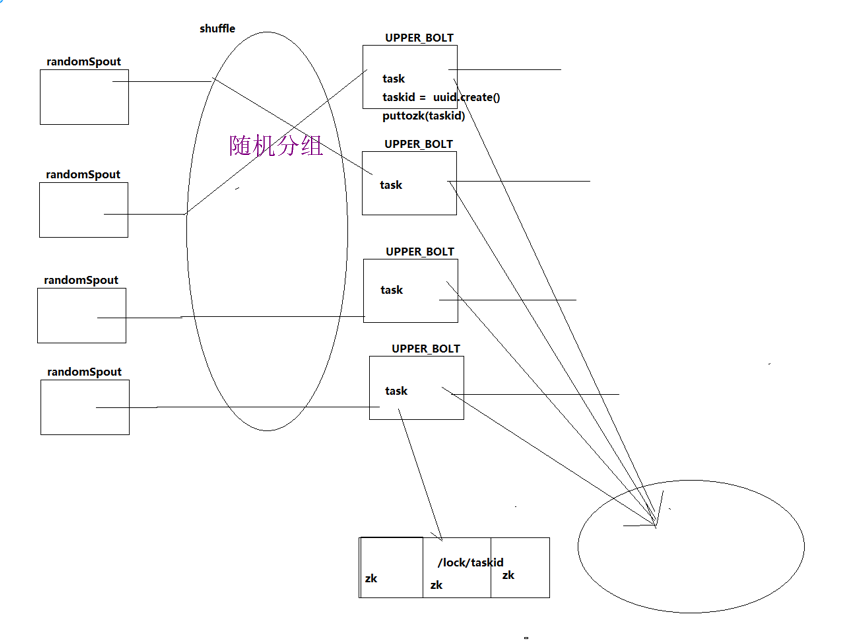
注意：每次增加一个suoervisor，就会增加4个slots

不过这是一个客户端的方式

## 5.4 storm编程规范及demo编写







### 5.4.1运行程序：



首先

切换到目录/home/hadoop/app/storm092/bin，然后运行

./storm jar /home/hadoop/stormdemo.jar cn.itcast.storm.demo.TopMain

### 5.4.2 停止topology

查看当前运行的topo： storm list  
命令格式：storm kill 【拓扑名称】  
样例：storm kill wordcountTop  
#杀掉wordcountTop拓扑。

storm的深入学习：

分布式共享锁的实现

事务topology的实现机制及开发模式

在具体场景中的跟其他框架的整合（flume/activeMQ/kafka(分布式的消息队列系统) /redis/hbase/mysql cluster）

# 6、Kafka知识

## 6.1 kafka简介

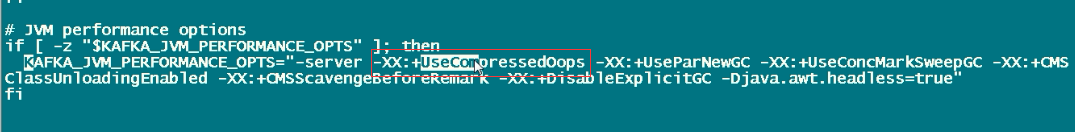
|  |
| --- |
| 1/kafka是一个分布式的消息缓存系统  2/kafka集群中的服务器都叫做broker  3/kafka有两类客户端，一类叫producer（消息生产者），一类叫做consumer（消息消费者），客户端和broker服务器之间采用tcp协议连接  4/kafka中不同业务系统的消息可以通过topic进行区分，而且每一个消息topic都会被分区，以分担消息读写的负载  5/每一个分区都可以有多个副本，以防止数据的丢失  6/某一个分区中的数据如果需要更新，都必须通过该分区所有副本中的leader来更新  7/消费者可以分组，比如有两个消费者组A和B，共同消费一个topic：order\_info,A和B所消费的消息不会重复  比如 order\_info 中有100个消息，每个消息有一个id,编号从0-99，那么，如果A组消费0-49号，B组就消费50-99号  8/消费者在具体消费某个topic中的消息时，可以指定起始偏移量 |

## kafka集群安装



注意：kafka2.10版本时

当jdk为1.7的时候，会不识别UseComoressedOops字段，该字段为jdk1.8的，所以删除该段配置即可(图上圈中的地方)修改文件：/home/hadoop/app/kafka\_210/bin目录下kafka-run-class.sh文件



**自己总结：**

1. **进入/home/hadoop/app/kafka\_210/config目录，修改 server.properties 文件**

**修改broker.id =0，**

**log.dirs=/home/hadoop/app/kafka\_210/kafka-logs**

**zookeeper.connect=weekend01:2181,weekend02:2181,weekend03:2181**

1. **把weekend01上面的kafka复制到weekend02，weekend03上面**

**scp -r kafka\_210/ weekend03:/home/hadoop/app**

**修改/home/hadoop/app/kafka\_210/config目录下的server.properties 文件中broker.id =1**

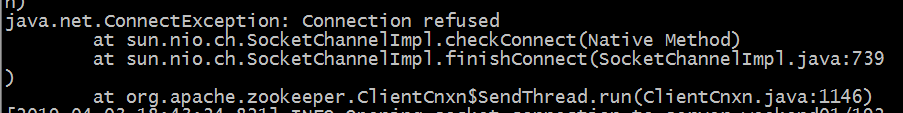
**scp -r kafka\_210/ weekend03:/home/hadoop/app**

**修改/home/hadoop/app/kafka\_210/config目录下的server.properties 文件中broker.id =2**

1. **启动broker**

**启动broker之前需要先启动自己配置的zookeeper，不然会报错误**

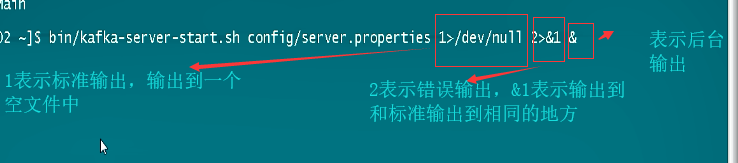
**错误：**



**bin/kafka-server-start.sh config/server.properties**

**后台启动输出：**

**bin/kafka-server-start.sh config/server.properties 1>/dev/null 2>&1 &**



1. 在kafka集群中创建一个topic

bin/kafka-topics.sh --create --zookeeper weekend01:2181 --replication-factor 3 --partitions 1 --topic order(自己创建话题的名称)

查看自己创建的话题：bin/kafka-topics.sh --list --zookeeper weekend01:2181

1. 用一个producer向某一个topic中写入消息

bin/kafka-console-producer.sh --broker-list weekend01:9092 --topic mygirls(自己创建的话题)

1. 用一个comsumer从某一个topic中读取信息

bin/kafka-console-consumer.sh --zookeeper weekend01:2181 --from-beginning --topic mygirls

7. 查看一个topic的分区及副本状态信息

bin/kafka-topics.sh --describe --zookeeper weekend01:2181 --topic order

## 客户端编程

生产者启动错误：

java.nio.channels.UnresolvedAddressException

需要在本地电脑上配置hosts文件，需要映射

192.168.93.101 weekend01

192.168.93.102 weekend02

192.168.93.103 weekend03

192.168.93.104 weekend04

192.168.93.105 weekend05

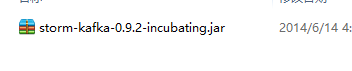
192.168.93.106 weekend06

192.168.93.107 weekend07

客户端编程：



## storm-kafka整合



Storm-kafka整合jar包