目录

[课程大纲（HADOOP快速入门） 2](#_Toc439057207)

[1. HADOOP 快速入门 3](#_Toc439057208)

[什么是HADOOP 3](#_Toc439057209)

[HADOOP产生背景 3](#_Toc439057210)

[HADOOP在大数据、云计算中的位置和关系 3](#_Toc439057211)

[国内外HADOOP应用案例介绍 4](#_Toc439057212)

[国内HADOOP的就业情况分析 5](#_Toc439057213)

[HADOOP生态圈以及各组成部分的简介 6](#_Toc439057214)

[分布式系统概述 6](#_Toc439057215)

[2. HIVE快速入门 7](#_Toc439057216)

[2.1 Hive基本介绍 7](#_Toc439057217)

[2.2 Hive的基本使用 8](#_Toc439057218)

[2.3 数据仓库基本知识 9](#_Toc439057219)

[3. 数据分析案列演示 10](#_Toc439057220)

[3.1 需求分析 10](#_Toc439057221)

[3.1.1案例名称 10](#_Toc439057222)

[3.1.2 案例需求描述 10](#_Toc439057223)

[3.1.3 web点击流日志的数据格式 10](#_Toc439057224)

[3.1.4 分析指标 11](#_Toc439057225)

[3.1.5 统计结果数据可视化 11](#_Toc439057226)

[3.2 数据来源分析 12](#_Toc439057227)

[3.2.1 企业中获取数据的几种方式 12](#_Toc439057228)

[3.2.2 数据采集 12](#_Toc439057229)

[3.3 数据处理流程 13](#_Toc439057230)

[数据预处理/加载入库 13](#_Toc439057231)

[使用Hive做数据ETL 14](#_Toc439057232)

[使用Hive运算业务指标 16](#_Toc439057233)

[将结果数据导出到mysql（sqoop） 17](#_Toc439057234)

[结果展现——数据可视化 17](#_Toc439057235)

[4. 集群搭建 18](#_Toc439057236)

[4.1 HADOOP集群搭建 18](#_Toc439057237)

[4.1.1集群简介： 18](#_Toc439057238)

[4.1.2服务器准备 18](#_Toc439057239)

[4.1.3网络环境准备 18](#_Toc439057240)

[4.1.4服务器系统设置 18](#_Toc439057241)

[4.1.5 Jdk环境安装 19](#_Toc439057242)

[4.1.6 HADOOP安装部署 19](#_Toc439057243)

[4.1.7 启动集群 21](#_Toc439057244)

[4.1.8 测试 21](#_Toc439057245)

[4.2 Hive搭建 22](#_Toc439057246)

[Hive的配置安装 22](#_Toc439057247)

[Hive的使用 23](#_Toc439057248)

[Hive运行测试 23](#_Toc439057249)

# 课程大纲（HADOOP快速入门）

|  |  |  |
| --- | --- | --- |
| HADOOP快速入门 | HADOOP快速入门 | HADOOP产生背景 |
| HADOOP在大数据、云计算中的位置和关系 |
| 国内外HADOOP应用案例介绍 |
| 国内HADOOP的就业情况分析及课程大纲介绍 |
| 分布式系统概述 |
| HADOOP生态圈以及各组成部分的简介 |
| Hive快速入门 | Hive快速入门 | Hive基本介绍 |
| Hive的使用 |
| 数据仓库基本知识 |
| 数据分析案例演示 | 需求分析 | 案列：定义需求、介绍数据格式 |
| 数据获取 | 企业中获取数据的几种方式 |
| 将文件直接导入到数据仓库 |
| 将数据库的数据导入到数据仓库（sqoop） |
| 数据处理 | 使用Hive对数进行清洗（ETL的过程） |
| 数据计算 | 使用Hive对数据进行计算 |
| 数据展现 | 将结果数据导出到mysql（sqoop） |

学习目标：

第一天接触具体的大数据框架，总目标是让学习者建立起大数据和分布式的宏观概念

1、理解hadoop是什么，用于做什么，大体上怎么用：hadoop 存储。计算。

2、理解hive是什么，用于做什么，大体上怎么用

3、通过一个案例的演示说明，理解数据挖掘系统的基本流程和结构

# http://b.hiphotos.baidu.com/baike/w%3D268/sign=0a0d07558501a18bf0eb1549a62e0761/3c6d55fbb2fb4316ecfbfb0322a4462308f7d3e7.jpghttp://a.cphotos.bdimg.com/timg?image&quality=100&size=b4000_4000&sec=1455525399&di=b1ac3db74b67c0699630af086c53d03e&src=http://www.china-cloud.com/uploads/allimg/121018/54-12101P92R1U7.jpgHADOOP背景介绍

hadoop之父Doug Cutting

1. 

## 1.1 什么是HADOOP

1. HADOOP是apache旗下的一套开源**软件平台**
2. HADOOP提供的功能：利用服务器集群，根据用户的自定义业务逻辑，**对海量数据进行分布式处理**
3. HADOOP的核心组件有
   * 1. HDFS（分布式文件系统）Hadoop distributed File System
     2. YARN（运算资源调度系统）
     3. MAPREDUCE（分布式运算编程框架）
4. 广义上来说，HADOOP通常是指一个更广泛的概念——HADOOP生态圈

## 1.2 HADOOP产生背景

1. HADOOP**最早起源于Nutch**。Nutch的设计目标是构建一个大型的全网搜索引擎，包括网页抓取、索引、查询等功能，但随着抓取网页数量的增加，**遇到了严重的可扩展性问题——**如何解决数十亿网页的存储和索引问题。
2. 2003年、2004年**谷歌发表的两篇论文为该问题提供了可行的解决方案**。

——分布式文件系统（GFS），可用于处理海量网页的**存储 Google File Syste**

——分布式计算框架MAPREDUCE，可用于处理海量网页的**索引计算**问题。mapReduce

-------分布式数据的存储BIGTable

Nutch的开发人员完成了相应的**开源实现HDFS和MAPREDUCE**，并从Nutch中剥离成为独立项目HADOOP，到2008年1月，HADOOP成为Apache顶级项目，迎来了它的快速发展期。

## 1.3 HADOOP在大数据、云计算中的位置和关系

1. 云计算是分布式计算、并行计算、网格计算、多核计算、网络存储、虚拟化、负载均衡等传统计算机技术和互联网技术融合发展的产物。借助IaaS(基础设施即服务)、PaaS(平台即服务)、SaaS（软件即服务）等业务模式，把强大的计算能力提供给终端用户。
2. 现阶段，云计算的**两大底层支撑技术**为“**虚拟化**”和“**大数据技术**”
3. 而HADOOP则是云计算的PaaS层的解决方案之一，并不等同于PaaS，更不等同于云计算本身。

## 1.4 国内外HADOOP应用案例介绍

**1、HADOOP应用于数据服务基础平台建设（数据仓库）**



**2/HADOOP用于用户画像**



**3、HADOOP用于网站点击流日志数据挖掘**



## 1.5 国内HADOOP的就业情况分析

1. HADOOP就业整体情况
2. 大数据产业已纳入**国家十三五规划**
3. 各大城市都在进行**智慧城市项目**建设，而智慧城市的根基就是大数据综合平台
4. 互联网时代数据的种类，增长都呈现**爆发式增长**，各行业对数据的价值日益重视
5. 相对于传统JAVAEE技术领域来说，大数据领域的**人才相对稀缺**
6. 随着现代社会的发展，数据处理和数据挖掘的重要性只会增不会减，因此，大数据技术是一个尚在蓬勃发展且具有**长远前景的领域**
7. HADOOP就业职位要求

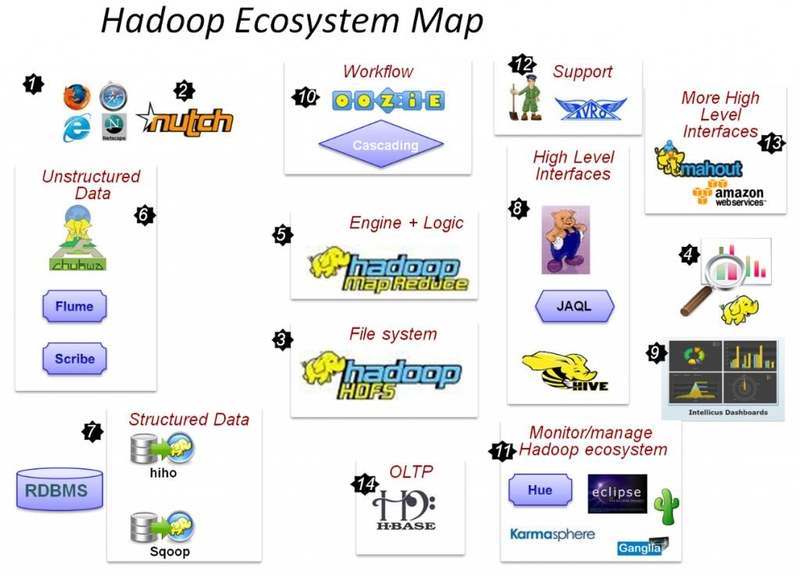
大数据是个复合专业，包括应用开发、软件平台、算法、数据挖掘,数据仓库等，因此，**大数据技术领域的就业选择是多样的**，但就HADOOP而言，通常都需要具备以下技能或知识：

1. HADOOP分布式集群的平台搭建
2. HADOOP分布式文件系统HDFS的原理理解及使用
3. HADOOP分布式运算框架MAPREDUCE的原理理解及编程
4. Hive数据仓库工具的熟练应用
5. Flume、sqoop、oozie等辅助工具的熟练使用
6. Shell/python等脚本语言的开发能力
7. HADOOP相关职位的薪资水平

大数据技术或具体到HADOOP的就业需求目前主要集中在北上广深一线城市，**薪资待遇普遍高于传统JAVAEE开发人员**，以北京为例：



## 1.6 HADOOP生态圈以及各组成部分的简介



各组件简介

重点组件：

HDFS：分布式文件系统

MAPREDUCE：分布式运算程序开发框架

HIVE：基于大数据技术（文件系统+运算框架）的SQL数据仓库工具

HBASE：基于HADOOP的分布式海量数据库

**ZOOKEEPER：分布式协调服务基础组件**

Mahout：基于mapreduce/spark/flink等分布式运算框架的机器学习算法库

Oozie：工作流调度框架

Sqoop：数据导入导出工具

Flume：日志数据采集框架

# 2 分布式系统概述

*注：由于大数据技术领域的各类技术框架基本上都是分布式系统，因此，理解hadoop、storm、spark等技术框架，都需要具备基本的分布式系统概念*

## 2.1 分布式软件系统(Distributed Software Systems)

* 该软件系统会划分成多个子系统或模块，各自运行在不同的机器上，子系统或模块之间通过网络通信进行协作，实现最终的整体功能
* 比如分布式操作系统、分布式程序设计语言及其编译(解释)系统、分布式文件系统和分布式数据库系统等。

## 2.2 分布式软件系统举例：solrcloud

1. 一个solrcloud集群通常有多台solr服务器
2. 每一个solr服务器节点负责存储整个索引库的若干个shard（数据分片）
3. 每一个shard又有多台服务器存放若干个副本互为主备用
4. 索引的建立和查询会在整个集群的各个节点上并发执行
5. solrcloud集群作为整体对外服务，而其内部细节可对客户端透明

**总结：利用多个节点共同协作完成一项或多项具体业务功能的系统就是分布式系统。**

## 2.3 分布式应用系统模拟开发

**需求：**可以实现由主节点将运算任务发往从节点，并将各从节点上的任务启动；

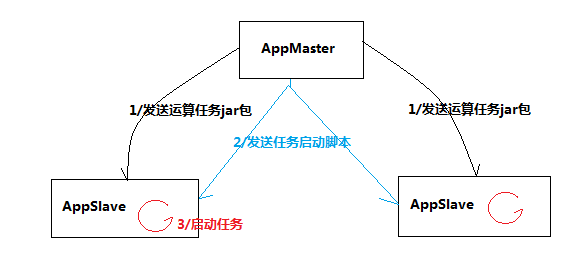
**程序清单：**

AppMaster

AppSlave/APPSlaveThread

Task

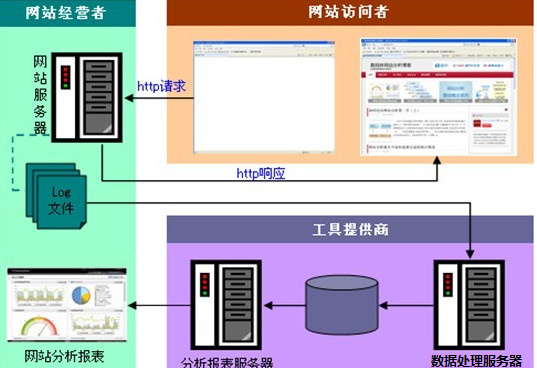
**程序运行逻辑流程：**



# 3. 离线数据分析流程介绍

*注：本环节主要感受数据分析系统的宏观概念及处理流程，初步理解hadoop等框架在其中的应用环节，不用过于关注代码细节*

一个应用广泛的数据分析系统：“web日志数据挖掘”



## 3.1 需求分析

### 3.1.1 案例名称

“网站或APP点击流日志数据挖掘系统”。

### 3.1.2 案例需求描述

“Web点击流日志”包含着网站运营很重要的信息，通过日志分析，我们可以知道网站的访问量，哪个网页访问人数最多，哪个网页最有价值，广告转化率、访客的来源信息，访客的终端信息等。

### 3.1.3 数据来源

本案例的数据主要由**用户的点击行为记录**

获取方式：在页面预埋一段js程序，为页面上想要监听的标签绑定事件，只要用户点击或移动到标签，即可触发ajax请求到后台servlet程序，用log4j记录下事件信息，从而在web服务器（nginx、tomcat等）上形成不断增长的日志文件。

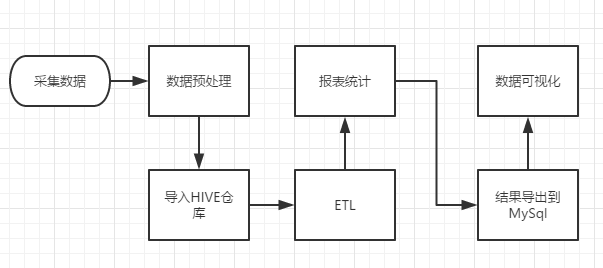
形如：

|  |
| --- |
| 58.215.204.118 - - [18/Sep/2013:06:51:35 +0000] "GET /wp-includes/js/jquery/jquery.js?ver=1.10.2 HTTP/1.1" 304 0 "http://blog.fens.me/nodejs-socketio-chat/" "Mozilla/5.0 (Windows NT 5.1; rv:23.0) Gecko/20100101 Firefox/23.0" |

## 3.2 数据处理流程

### 3.2.1 流程图解析

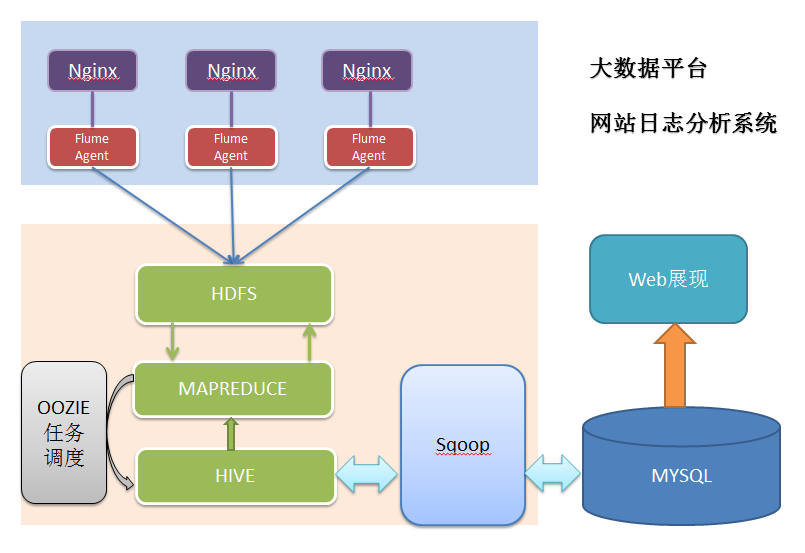
本案例跟典型的BI系统极其类似，整体流程如下：



但是，由于本案例的前提是处理海量数据，因而，流程中各环节所使用的技术则跟传统BI完全不同，后续课程都会一一讲解：

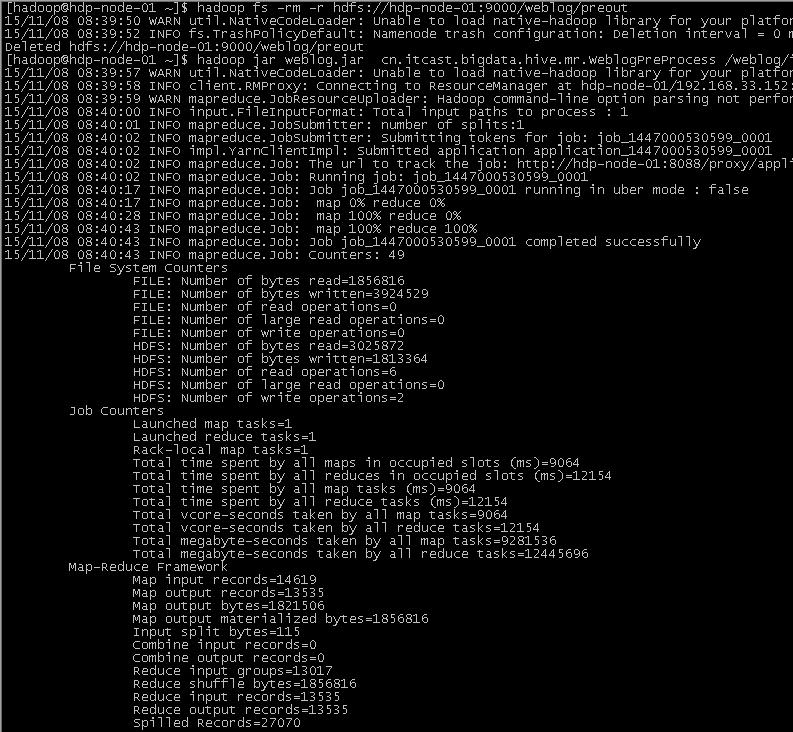
1. 数据采集：定制开发采集程序，或使用开源框架FLUME
2. 数据预处理：定制开发mapreduce程序运行于hadoop集群
3. 数据仓库技术：基于hadoop之上的Hive
4. 数据导出：基于hadoop的sqoop数据导入导出工具
5. 数据可视化：定制开发web程序或使用kettle等产品
6. 整个过程的流程调度：hadoop生态圈中的oozie工具或其他类似开源产品

### 3.2.2 项目技术架构图

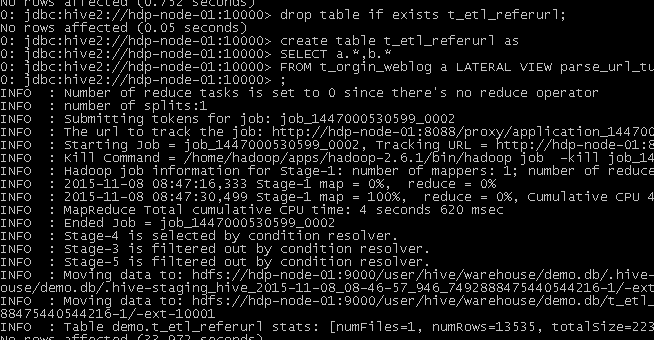


### 3.2.3 项目相关截图（感性认识，欣赏即可）

1. Mapreudce程序运行



1. 在Hive中查询数据



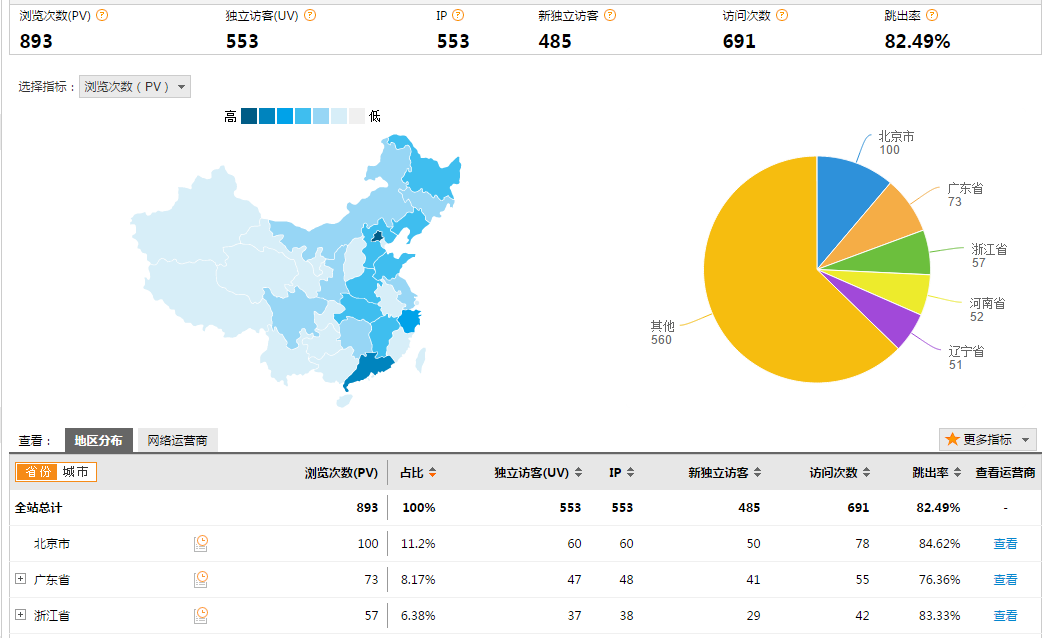
1. 将统计结果导入mysql

|  |
| --- |
| ./sqoop export --connect jdbc:mysql://localhost:3306/weblogdb --username root --password root --table t\_display\_xx --export-dir /user/hive/warehouse/uv/dt=2014-08-03 |

## 3.3 项目最终效果

经过完整的数据处理流程后，会周期性输出各类统计指标的报表，在生产实践中，最终需要将这些报表数据以可视化的形式展现出来，本案例采用web程序来实现数据可视化

效果如下所示：



# 4. 集群搭建

## 4.1 HADOOP集群搭建

### 4.1.1集群简介

HADOOP集群具体来说包含两个集群：HDFS集群和YARN集群，两者逻辑上分离，但物理上常在一起

HDFS集群：

负责海量数据的存储，集群中的角色主要有 NameNode / DataNode

YARN集群：

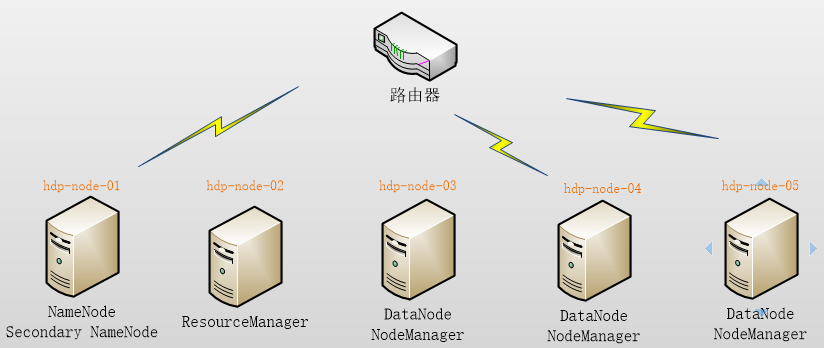
负责海量数据运算时的资源调度，集群中的角色主要有 ResourceManager /NodeManager

*(那mapreduce是什么呢？它其实是一个应用程序开发包)*

本集群搭建案例，以5节点为例进行搭建，角色分配如下：

|  |
| --- |
| hdp-node-01 NameNode SecondaryNameNode  hdp-node-02 ResourceManager  hdp-node-03 DataNode NodeManager  hdp-node-04 DataNode NodeManager  hdp-node-05 DataNode NodeManager |

部署图如下：



master hadoop 主机。

scecondary NameNode 为master 提供服务，读取：DateNode 中文件的信息。提交给master

### 4.1.2服务器准备

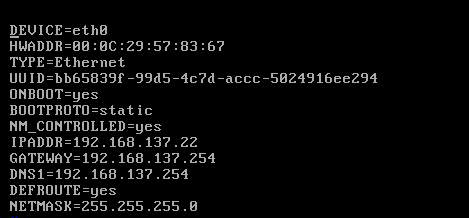
本案例使用虚拟机服务器来搭建HADOOP集群，所用软件及版本：

* Vmware 11.0
* Centos 6.5 64bit

### 4.1.3网络环境准备

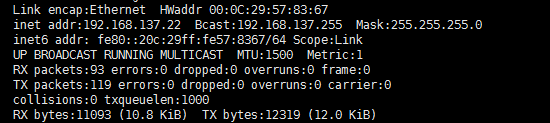
1. 安装centos系统，并修改IP地址：192.168.137.22





配置完成后，重启服务：service network restart。

查看IP地址：ifconfig



1. 关闭防火墙：

service iptables stop :关闭防火墙服务

chkconfig iptables off ：关闭防火墙配置文件。

1. 开启sshd 服务

service sshd start

1. 修改主机名

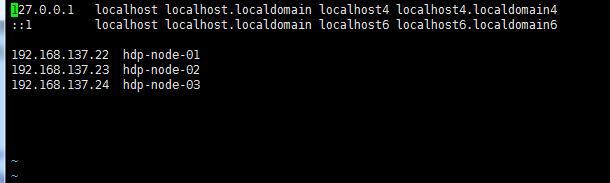
临时修改：hostname hdp-node-01

永久修改：vi /etc/sysconfig/network



1. 修改主机映射：

vi /etc/hosts



1. 添加用户，分配sudoer权限

useradd huaqiang

passwd 123456

添加sudoers 文件可读权限。

chmod 640 sudoers

编辑sudoers文件。首行添加：

huaqiang ALL=(root)NOPASSWD:ALL

配置完成后：使用root权限。可使用sudo 即可。

1. 安装JDK

* 将JDK 上传到/home/huaqiang/app目录
* 添加可执行权限。

#chmod 764 jdk...

* 解压缩JDK

#tar -zxf jdk-7u67-linux-x64.tar.gz -C ./

* 配置环境变量：

#sudo vi /etc/profile

文件末尾添加代码：

#SET JAVA\_HOME

export JAVA\_HOME=/home/huaqiang/app/jdk1.7.0\_67

export PATH=$PATH:$JAVA\_HOME/bin:

* 刷新资源文件

#source /etc/profile

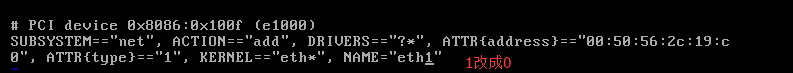
测试是否安装成功

java javac java -version

### 克隆其他两台主机。

* 克隆之后首先修改克隆机的mac地址。
* 00:50:56:2F:AF:62
* 修改网卡配置。





* 开启02,03 主机，并按照以上步骤，修改IP.主机名，以及映射。

### 设置SSH免密码登录

计算机A每次SSH远程连接计算机B时，都需要输入密码。当有些软件需要频繁使用SSH连接其他计算机时，这个时候需要人工填写计算机B的密码，这个会造成大量无用操作，不利于程序的使用。

在为了避免这类问题的发生，而且也要兼顾安全问题（设置SSH免密码登陆后，SSH这台计算就不需要密码即可登陆，存在安全隐患），使用公钥和私钥方式解决SSH免密码登陆问题，而且只在一边做，另一边不做，这样也达到了安全的问题。一般是那台机器需要SSH远程其他机器，就在这台机器上设置SSH免登陆操作。

"公私钥"认证方式简单的解释:首先在客户端上创建一对公私钥 （公钥文件：~/.ssh/id\_rsa.pub； 私钥文件：~/.ssh/id\_rsa）。然后把公钥放到服务器上（~/.ssh/authorized\_keys）, 自己保留好私钥.在使用ssh登录时,ssh程序会发送私钥去和服务器上的公钥做匹配.如果匹配成功就可以登录了。

ssh 无密码登录要使用公钥与私钥。linux下可以用ssh-keygen生成公钥/私钥对.

//在master机下生成公钥/私钥对。

# ssh-keygen -t rsa

//拷贝pubkey公钥，并修改其名称为：authorized\_keys

cp id\_rsa.pub ./authorized\_keys

//将master机子上的key给其他机子上发送一份

# scp ~/.ssh/authorized\_keys slave1:/home/huaqiang/.ssh/

# scp ~/.ssh/authorized\_keys slave2:/home/huaqiang/.ssh/

ssh-copy-id hadoop102

ssh-copy-id hadoop103

ssh-copy-id hadoop104

使用ssh进行登录验证

ssh slave1

ssh slave2

### 4.1.6 HADOOP安装部署

* 上传HADOOP安装包
* 规划安装目录 /home/hadoop/apps/hadoop-2.6.1
* 解压安装包
* 修改配置文件 $HADOOP\_HOME/etc/hadoop/

|  |
| --- |
| <property>  <name>fs.defaultFS</name>  <value>hdfs://master:8020</value>  </property>  <property>  <name>io.file.buffer.size</name>  <value>4096</value>  </property> |

### 修改hdfs-site.xml

|  |
| --- |
| <property>  <name>dfs.replication</name>  <value>2</value>  </property>  <property>  <name>dfs.block.size</name>  <value>134217728</value>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>/home/hadoopdata/dfs/name</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>/home/hadoopdata/dfs/data</value>  </property>  <property>  <name>fs.checkpoint.dir</name>  <value>/home/hadoopdata/checkpoint/dfs/cname</value>  </property>  <property>  <name>fs.checkpoint.edits.dir</name>  <value>/home/hadoopdata/checkpoint/dfs/cname</value>  </property>  <property>  <name>dfs.http.address</name>  <value>master:50070</value>  </property>  <property>  <name>dfs.secondary.http.address</name>  <value>slave1:50090</value>  </property>  <property>  <name>dfs.webhdfs.enabled</name>  <value>true</value>  </property>  <property>  <name>dfs.permissions</name>  <value>false</value>  </property> |

### 修改mapred-site.xml文件

|  |
| --- |
| <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  <final>true</final>  </property>  <property>  <name>mapreduce.jobhistory.address</name>  <value>master:10020</value>  </property>  <property>  <name>mapreduce.jobhistory.webapp.address</name>  <value>master:19888</value>  </property> |

### 修改yarn-site.xml

|  |
| --- |
| <property>  <name>yarn.resourcemanager.hostname</name>  <value>master </value>  </property>  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.resourcemanager.address</name>  <value>master:8032</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>master:8030</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>master:8031</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>master:8033</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>master:8088</value>  </property> |

### 修改slaves文件

|  |
| --- |
| 注意:每个机器名占一行。  master  slave1  slave2 |

### 拷贝Hadoop 到 datanode02 datanode03 主机上。

scp -r ./hadoop-2.7.1 hadoop-datanode-02:/home/huaqiang/app/

scp -r ./hadoop-2.7.1 hadoop-datanode-03:/home/huaqiang/app/

### 4.1.7 启动集群

初始化HDFS

|  |
| --- |
| bin/hadoop namenode -format |

启动HDFS

|  |
| --- |
| sbin/start-dfs.sh |

启动YARN

|  |
| --- |
| sbin/start-yarn.sh |

### 4.1.8 测试

#### 1、上传文件到HDFS

从本地上传一个文本文件到hdfs的/wordcount/input目录下

|  |
| --- |
| [HADOOP@hdp-node-01 ~]$ HADOOP fs -mkdir -p /wordcount/input  [HADOOP@hdp-node-01 ~]$ HADOOP fs -put /home/HADOOP/somewords.txt /wordcount/input |

#### 2、运行一个mapreduce程序

在HADOOP安装目录下，运行一个示例mr程序

|  |
| --- |
| cd $HADOOP\_HOME/share/hadoop/mapreduce/  hadoop jar mapredcue-example-2.6.1.jar wordcount /wordcount/input /wordcount/output |

开启服务：

Wei:UI 查看启动：

[http://](http://hadoop-yarn.beicai.com:50070/)[master.com:50070/](http://hadoop-yarn.beicai.com:50070/) hdfs 启动

[http://master.com:8088/cluster](http://hadoop-yarn2.beicai.com:8088/cluster) yarn 启动

启动history服务才能看到webUI节点

$mr-jobhistory-daemon.sh start historyserver

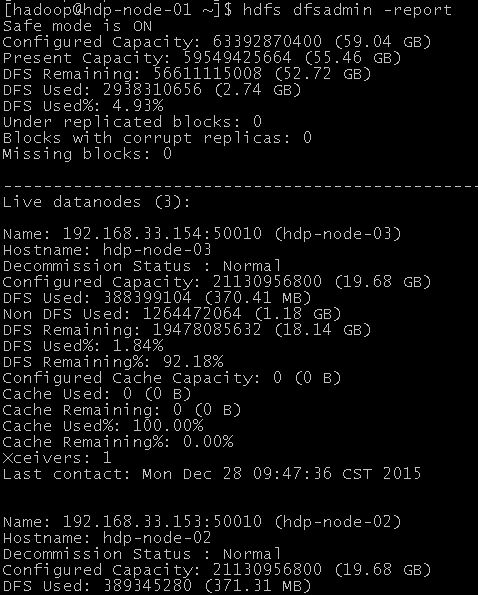
[http://master.com:19888/](http://hadoop-yarn1.beicai.com:19888/jobhistory)[jobhistory](http://hadoop-yarn1.beicai.com:19888/jobhistory) mr 启动

# 5 集群使用初步

## 5.1 HDFS使用

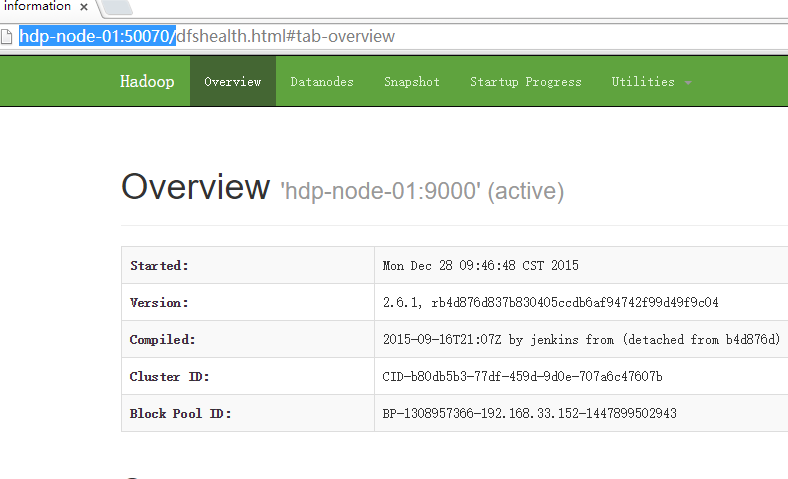
1、查看集群状态

命令： hdfs dfsadmin –report



可以看出，集群共有3个datanode可用

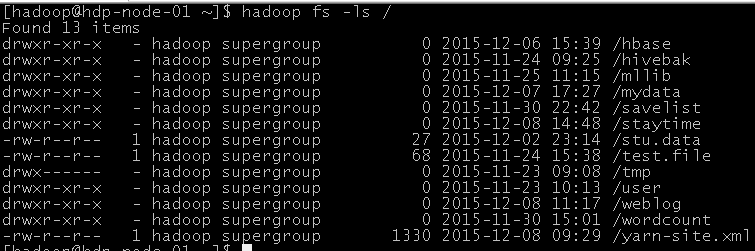
也可打开web控制台查看HDFS集群信息，在浏览器打开<http://hdp-node-01:50070/>



2、上传文件到HDFS

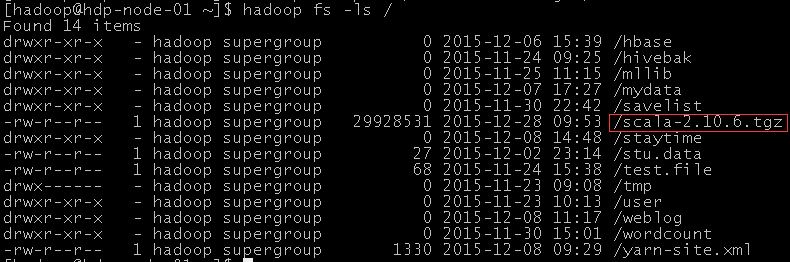
* 查看HDFS中的目录信息

命令： hadoop fs –ls /



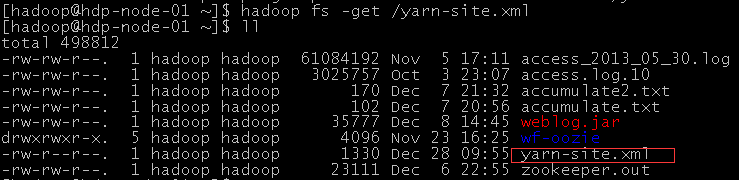
* 上传文件

命令： hadoop fs -put ./ scala-2.10.6.tgz to /



* 从HDFS下载文件

命令： hadoop fs -get /yarn-site.xml



## 5.2 MAPREDUCE使用

mapreduce是hadoop中的分布式运算编程框架，只要按照其编程规范，只需要编写少量的业务逻辑代码即可实现一个强大的海量数据并发处理程序

### 5.2.1 Demo开发——wordcount

1、需求

从大量（比如T级别）文本文件中，统计出每一个单词出现的总次数

2、mapreduce实现思路

Map阶段：

1. 从HDFS的源数据文件中逐行读取数据
2. 将每一行数据切分出单词
3. 为每一个单词构造一个键值对(单词，1)
4. 将键值对发送给reduce

Reduce阶段：

1. 接收map阶段输出的单词键值对
2. 将相同单词的键值对汇聚成一组
3. 对每一组，遍历组中的所有“值”，累加求和，即得到每一个单词的总次数
4. 将(单词，总次数)输出到HDFS的文件中
5. 具体编码实现

(1)定义一个mapper类

|  |
| --- |
| //首先要定义四个泛型的类型  //keyin: LongWritable valuein: Text  //keyout: Text valueout:IntWritable  public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{  //map方法的生命周期： 框架每传一行数据就被调用一次  //key : 这一行的起始点在文件中的偏移量  //value: 这一行的内容  @Override  protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  //拿到一行数据转换为string  String line = value.toString();  //将这一行切分出各个单词  String[] words = line.split(" ");  //遍历数组，输出<单词，1>  for(String word:words){  context.write(new Text(word), new IntWritable(1));  }  }  } |

(2)定义一个reducer类

|  |
| --- |
| //生命周期：框架每传递进来一个kv 组，reduce方法被调用一次  @Override  protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  //定义一个计数器  int count = 0;  //遍历这一组kv的所有v，累加到count中  for(IntWritable value:values){  count += value.get();  }  context.write(key, new IntWritable(count));  }  } |

(3)定义一个主类，用来描述job并提交job

|  |
| --- |
| public class WordCountRunner {  //把业务逻辑相关的信息（哪个是mapper，哪个是reducer，要处理的数据在哪里，输出的结果放哪里。。。。。。）描述成一个job对象  //把这个描述好的job提交给集群去运行  public static void main(String[] args) throws Exception {  Configuration conf = new Configuration();  Job wcjob = Job.getInstance(conf);  //指定我这个job所在的jar包  // wcjob.setJar("/home/hadoop/wordcount.jar");  wcjob.setJarByClass(WordCountRunner.class);    wcjob.setMapperClass(WordCountMapper.class);  wcjob.setReducerClass(WordCountReducer.class);  //设置我们的业务逻辑Mapper类的输出key和value的数据类型  wcjob.setMapOutputKeyClass(Text.class);  wcjob.setMapOutputValueClass(IntWritable.class);  //设置我们的业务逻辑Reducer类的输出key和value的数据类型  wcjob.setOutputKeyClass(Text.class);  wcjob.setOutputValueClass(IntWritable.class);    //指定要处理的数据所在的位置  FileInputFormat.setInputPaths(wcjob, "hdfs://hdp-server01:9000/wordcount/data/big.txt");  //指定处理完成之后的结果所保存的位置  FileOutputFormat.setOutputPath(wcjob, new Path("hdfs://hdp-server01:9000/wordcount/output/"));    //向yarn集群提交这个job  boolean res = wcjob.waitForCompletion(true);  System.exit(res?0:1);  } |

### 5.2.2 程序打包运行

1. 将程序打包
2. 准备输入数据

vi /home/hadoop/test.txt

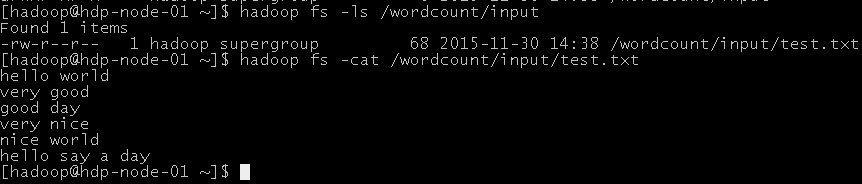
|  |
| --- |
| Hello tom  Hello jim  Hello ketty  Hello world  Ketty tom |

在hdfs上创建输入数据文件夹：

hadoop fs mkdir -p /wordcount/input

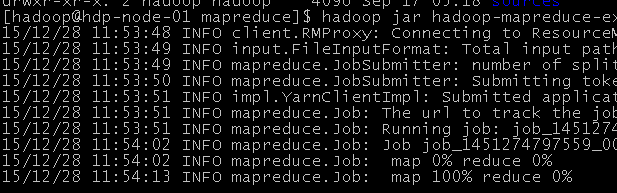
将words.txt上传到hdfs上

hadoop fs –put /home/hadoop/words.txt /wordcount/input



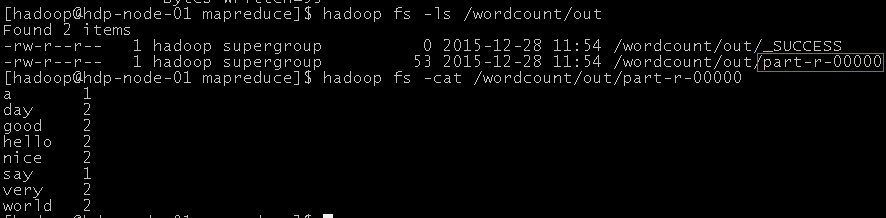
1. 将程序jar包上传到集群的任意一台服务器上
2. 使用命令启动执行wordcount程序jar包

$ hadoop jar wordcount.jar cn.itcast.bigdata.mrsimple.WordCountDriver /wordcount/input /wordcount/out



1. 查看执行结果

$ hadoop fs –cat /wordcount/out/part-r-00000



HDFS

# \*\*\*\*\*\*HDFS基本概念篇\*\*\*\*\*\*

# HDFS前言

1. HDFS( Hadoop distributed File System )

* 设计思想

分而治之：将大文件、大批量文件，分布式存放在大量服务器上，**以便于采取分而治之的方式对海量数据进行运算分析；**

* 在大数据系统中作用：

为各类分布式运算框架（如：mapreduce，spark，hive , tez，……）提供数据存储服务

* 重点概念：文件切块，副本存放，元数据

# HDFS的概念和特性

linux: 一切皆文件. 本机.一台linux操作系统.

hadoop: 分布式文件系统

**首先，它是一个文件系统**，用于存储文件，通过统一的命名空间——目录树来定位文件

**其次，它是分布式的**，由很多服务器联合起来实现其功能，集群中的服务器有各自的角色；

**重要特性如下：**

1. HDFS中的文件在物理上是**分块存储（block）**，块的大小可以通过配置参数( dfs.blocksize)来规定，默认大小在hadoop2.x版本中是128M，老版本中是64M

1G文件. 1024/128=

1. HDFS文件系统会给客户端提供一个**统一的抽象目录树**，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data
2. **目录结构及文件分块信息(元数据)**的管理由namenode节点承担

——namenode是HDFS集群主节点，负责维护整个hdfs文件系统的目录树，以及每一个路径（文件）所对应的block块信息（block的id，及所在的datanode服务器）

1. 文件的各个block的存储管理由datanode节点承担

---- datanode是HDFS集群从节点，每一个block都可以在多个datanode上存储多个副本（副本数量也可以通过参数设置dfs.replication）

1. HDFS是设计成适应一次写入，多次读出的场景，且不支持文件的修改

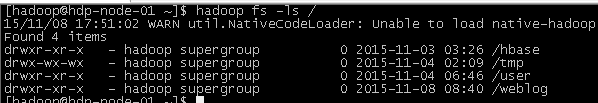
*(注：适合用来做数据分析，并不适合用来做网盘应用，因为，不便修改，延迟大，网络开销大，成本太高)*

# \*\*\*\*\*\*HDFS基本操作篇\*\*\*\*\*\*

# 3. HDFS的shell(命令行客户端)操作

## 3.1 HDFS命令行客户端使用

HDFS提供shell命令行客户端，使用方法如下：



## 3.2 命令行客户端支持的命令参数

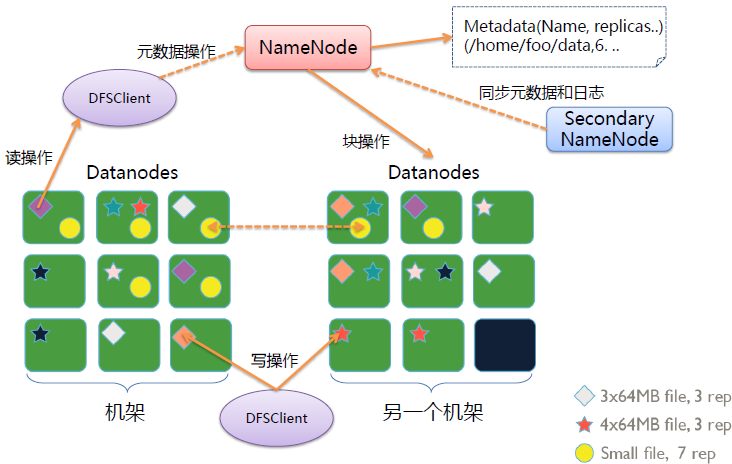
|  |
| --- |
| [-appendToFile <localsrc> ... <dst>]  [-cat [-ignoreCrc] <src> ...]  [-checksum <src> ...]  [-chgrp [-R] GROUP PATH...]  [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]  [-chown [-R] [OWNER][:[GROUP]] PATH...]  [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]  [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]  [-count [-q] <path> ...]  [-cp [-f] [-p] <src> ... <dst>]  [-createSnapshot <snapshotDir> [<snapshotName>]]  [-deleteSnapshot <snapshotDir> <snapshotName>]  [-df [-h] [<path> ...]]  [-du [-s] [-h] <path> ...]  [-expunge]  [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]  [-getfacl [-R] <path>]  [-getmerge [-nl] <src> <localdst>]  [-help [cmd ...]]  [-ls [-d] [-h] [-R] [<path> ...]]  [-mkdir [-p] <path> ...]  [-moveFromLocal <localsrc> ... <dst>]  [-moveToLocal <src> <localdst>]  [-mv <src> ... <dst>]  [-put [-f] [-p] <localsrc> ... <dst>]  [-renameSnapshot <snapshotDir> <oldName> <newName>]  [-rm [-f] [-r|-R] [-skipTrash] <src> ...]  [-rmdir [--ignore-fail-on-non-empty] <dir> ...]  [-setfacl [-R] [{-b|-k} {-m|-x <acl\_spec>} <path>]|[--set <acl\_spec> <path>]]  [-setrep [-R] [-w] <rep> <path> ...]  [-stat [format] <path> ...]  [-tail [-f] <file>]  [-test -[defsz] <path>]  [-text [-ignoreCrc] <src> ...]  [-touchz <path> ...]  [-usage [cmd ...]] |

## 3.2 常用命令参数介绍

|  |
| --- |
| -help  功能：输出这个命令参数手册 |
| **-ls**  **功能：显示目录信息**  *示例： hadoop fs -ls hdfs://hadoop-server01:9000/*  *备注：这些参数中，所有的hdfs路径都可以简写*  *-->hadoop fs -ls / 等同于上一条命令的效果* |
| **-mkdir**  **功能：在hdfs上创建目录**  *示例：hadoop fs -mkdir -p /aaa/bbb/cc/dd* |
| **-moveFromLocal**  **功能：从本地剪切粘贴到hdfs**  *示例：hadoop fs - moveFromLocal /home/hadoop/a.txt /aaa/bbb/cc/dd*  **-moveToLocal**  **功能：从hdfs剪切粘贴到本地**  *示例：hadoop fs - moveToLocal /aaa/bbb/cc/dd /home/hadoop/a.txt* |
| **--appendToFile**  **功能：追加一个文件到已经存在的文件末尾**  *示例：hadoop fs -appendToFile ./hello.txt hdfs://hadoop-server01:9000/hello.txt*  *可以简写为：*  *Hadoop fs -appendToFile ./hello.txt /hello.txt* |
| **-cat**  **功能：显示文件内容**  *示例：hadoop fs -cat /hello.txt*  **-tail**  **功能：显示一个文件的末尾**  *示例：hadoop fs -tail /weblog/access\_log.1*  **-text**  **功能：以字符形式打印一个文件的内容**  *示例：hadoop fs -text /weblog/access\_log.1* |
| **-chgrp**  **-chmod**  **-chown**  **功能：linux文件系统中的用法一样，对文件所属权限**  *示例：*  *hadoop fs -chmod 666 /hello.txt*  *hadoop fs -chown someuser:somegrp /hello.txt* |
| **-copyFromLocal**  **功能：从本地文件系统中拷贝文件到hdfs路径去**  *示例：hadoop fs -copyFromLocal ./jdk.tar.gz /aaa/*  **-copyToLocal**  **功能：从hdfs拷贝到本地**  *示例：hadoop fs -copyToLocal /aaa/jdk.tar.gz* |
| **-cp**  **功能：从hdfs的一个路径拷贝hdfs的另一个路径**  *示例： hadoop fs -cp /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2*  **-mv**  **功能：在hdfs目录中移动文件**  *示例： hadoop fs -mv /aaa/jdk.tar.gz /* |
| **-get**  **功能：等同于copyToLocal，就是从hdfs下载文件到本地**  示例：hadoop fs -get /aaa/jdk.tar.gz  **-getmerge**  **功能：合并下载多个文件**  *示例：比如hdfs的目录 /aaa/下有多个文件:log.1, log.2,log.3,...*  hadoop fs -getmerge /aaa/log.\* ./log.sum |
| **-put**  **功能：等同于copyFromLocal**  *示例：hadoop fs -put /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2* |
| **-rm**  **功能：删除文件或文件夹**  *示例：hadoop fs -rm -r /aaa/bbb/*  **-rmdir**  **功能：删除空目录**  *示例：hadoop fs -rmdir /aaa/bbb/ccc* |
| **-df**  **功能：统计文件系统的可用空间信息**  *示例：hadoop fs -df -h /*  **-du**  **功能：统计文件夹的大小信息**  *示例：*  *hadoop fs -du -s -h /aaa/\** |
| **-count**  **功能：统计一个指定目录下的文件节点数量**  *示例：hadoop fs -count /aaa/* |
| **-setrep**  **功能：设置hdfs中文件的副本数量**  *示例：hadoop fs -setrep 3 /aaa/jdk.tar.gz*  *<这里设置的副本数只是记录在namenode的元数据中，是否真的会有这么多副本，还得看datanode的数量>* |

# \*\*\*\*\*\*HDFS原理篇\*\*\*\*\*\*

# hdfs的工作机制

1. 

## 4.1 概述

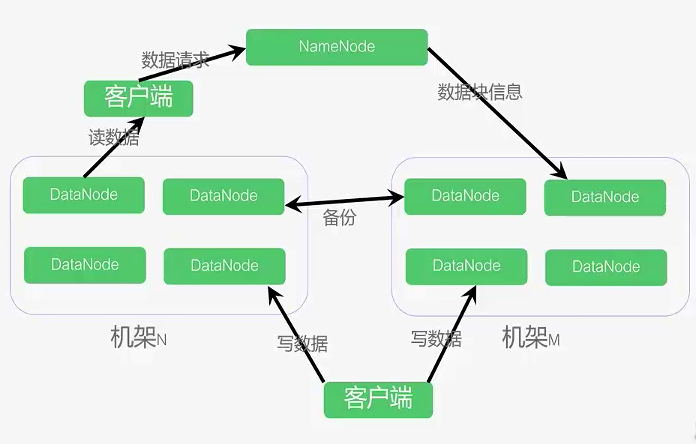
1. HDFS集群分为两大角色：NameNode(主节点)、DataNode(从节点)、
2. NameNode负责管理整个文件系统的元数据（block日志，镜像文件fsimage ）
3. DataNode 负责管理用户的文件数据块
4. 文件会按照固定的大小（blocksize）切成若干块后分布式存储在若干台datanode上
5. 每一个文件块可以有多个副本，并存放在不同的datanode上
6. Datanode会定期向Namenode汇报自身所保存的文件block信息，而namenode则会负责保持文件的副本数量
7. HDFS的内部工作机制对客户端保持透明，客户端请求访问HDFS都是通过向namenode申请来进行

## 4.2 HDFS写数据流程

### 4.2.1 概述

客户端要向HDFS写数据，首先要跟namenode通信以确认可以写文件并获得接收文件block的datanode，然后，客户端按顺序将文件逐个block传递给相应datanode，并由接收到block的datanode负责向其他datanode复制block的副本

### 4.2.2 详细步骤图



### 4.2.3 详细步骤解析

1、客户端根namenode通信请求上传文件，namenode检查目标文件是否已存在，父目录是否存在

2、namenode返回是否可以上传

3、client请求第一个 block该传输到哪些datanode服务器上

4、namenode返回3个datanode服务器ABC

5、client请求3台dn中的一台A上传数据（本质上是一个RPC调用，建立pipeline），A收到请求会继续调用B，然后B调用C，将真个pipeline建立完成，逐级返回客户端

6、client开始往A上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，A收到一个packet就会传给B，B传给C；A每传一个packet会放入一个应答队列等待应答

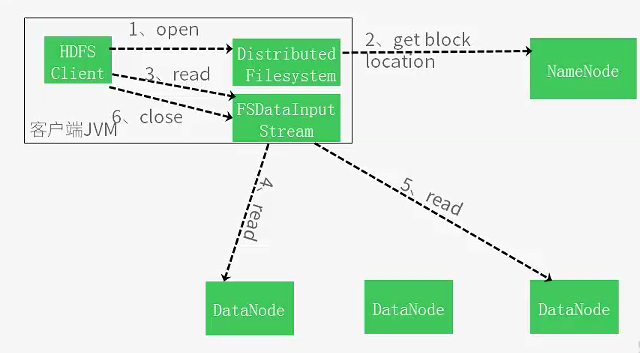
7、当一个block传输完成之后，client再次请求namenode上传第二个block的服务器。

## 4.3. HDFS读数据流程

### 4.3.1 概述

客户端将要读取的文件路径发送给namenode，namenode获取文件的元信息（主要是block的存放位置信息）返回给客户端，客户端根据返回的信息找到相应datanode逐个获取文件的block并在客户端本地进行数据追加合并从而获得整个文件

### 4.3.2 详细步骤图



### 4.3.3 详细步骤解析

1、跟namenode通信查询元数据，找到文件块所在的datanode服务器

2、挑选一台datanode（就近原则，然后随机）服务器，请求建立socket流

3、datanode开始发送数据（从磁盘里面读取数据放入流，以packet(包)为单位来做校验）

4、客户端以packet为单位接收，现在本地缓存，然后写入目标文件

# 5. NAMENODE工作机制

学习目标：理解namenode的工作机制尤其是**元数据管理**机制，以增强对HDFS工作原理的理解，及培养hadoop集群运营中“性能调优”、“namenode”故障问题的分析解决能力

## 5.1 NAMENODE职责

NAMENODE职责：

负责客户端请求的响应

元数据的管理（查询，修改）

## 5.2 元数据管理

namenode对数据的管理采用了三种存储形式：

内存元数据(NameSystem)

磁盘元数据镜像文件

数据操作日志文件（可通过日志运算出元数据）

### 5.2.1 元数据存储机制

A、内存中有一份完整的元数据(**内存meta data**)

B、磁盘有一个“准完整”的元数据镜像（**fsimage**）文件(在namenode的工作目录中)

C、用于衔接内存metadata和持久化元数据镜像fsimage之间的操作日志（**edits文件**）*注：当客户端对hdfs中的文件进行新增或者修改操作，操作记录首先被记入edits日志文件中，当客户端操作成功后，相应的元数据会更新到内存meta.data中*

5.2.3 元数据的checkpoint

每隔一段时间，会由secondary namenode将namenode上积累的所有edits和一个最新的fsimage下载到本地，并加载到内存进行merge（这个过程称为checkpoint）

secondary Namenode 的作用:为namenode提供服务: 合并日志和镜像文件.

#### checkpoint的附带作用

namenode和secondary namenode的工作目录存储结构完全相同，所以，当namenode故障退出需要重新恢复时，可以从secondary namenode的工作目录中将fsimage拷贝到namenode的工作目录，以恢复namenode的元数据

# 6. DATANODE的工作机制

## 6.1 概述

1、Datanode工作职责：

存储管理用户的文件块数据

定期向namenode汇报自身所持有的block信息（通过心跳信息上报）

（这点很重要，因为，当集群中发生某些block副本失效时，集群如何恢复block初始副本数量的问题）

|  |
| --- |
| <property>  <name>dfs.blockreport.intervalMsec</name>  <value>3600000</value>  <description>Determines block reporting interval in milliseconds.</description>  </property> |

2、Datanode掉线判断时限参数

datanode进程死亡或者网络故障造成datanode无法与namenode通信，namenode不会立即把该节点判定为死亡，要经过一段时间，这段时间暂称作超时时长。HDFS默认的超时时长为10分钟+30秒。如果定义超时时间为timeout，则超时时长的计算公式为：

timeout = 2 \* heartbeat.recheck.interval + 10 \* dfs.heartbeat.interval。

而默认的heartbeat.recheck.interval 大小为5分钟，dfs.heartbeat.interval默认为3秒。

需要注意的是hdfs-site.xml 配置文件中的heartbeat.recheck.interval的单位为毫秒，dfs.heartbeat.interval的单位为秒。所以，举个例子，如果heartbeat.recheck.interval设置为5000（毫秒），dfs.heartbeat.interval设置为3（秒，默认），则总的超时时间为40秒。

|  |
| --- |
| <property>  <name>heartbeat.recheck.interval</name>  <value>2000</value>  </property>  <property>  <name>dfs.heartbeat.interval</name>  <value>1</value>  </property> |

## 6.2 观察验证DATANODE功能

上传一个文件，观察文件的block具体的物理存放情况：

在每一台datanode机器上的这个目录中能找到文件的切块：

/home/hadoop/app/hadoop-2.4.1/tmp/dfs/data/current/BP-193442119-192.168.2.120-1432457733977/current/finalized

# \*\*\*\*\*\*HDFS应用开发篇\*\*\*\*\*\*

# 7. HDFS的java操作

*hdfs在生产应用中主要是客户端的开发，其核心步骤是从hdfs提供的api中构造一个HDFS的访问客户端对象，然后通过该客户端对象操作（增删改查）HDFS上的文件*

## 7.1 搭建开发环境

1、引入依赖

|  |  |
| --- | --- |
| |  | | --- | | <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-client</artifactId>  <version>2.6.1</version>  </dependency> | |

*注：如需手动引入jar包，hdfs的jar包----hadoop的安装目录的share下*

2、window下开发的说明

建议在linux下进行hadoop应用的开发，不会存在兼容性问题。如在window上做客户端应用开发，需要设置以下环境：

1. 在windows的某个目录下解压一个hadoop的安装包
2. 将安装包下的lib和bin目录用对应windows版本平台编译的本地库替换
3. 在window系统中配置HADOOP\_HOME指向你解压的安装包
4. 在windows系统的path变量中加入hadoop的bin目录

## 7.2 获取api中的客户端对象

在java中操作hdfs，首先要获得一个客户端实例

|  |
| --- |
| Configuration conf = new Configuration()  FileSystem fs = FileSystem.get(conf) |

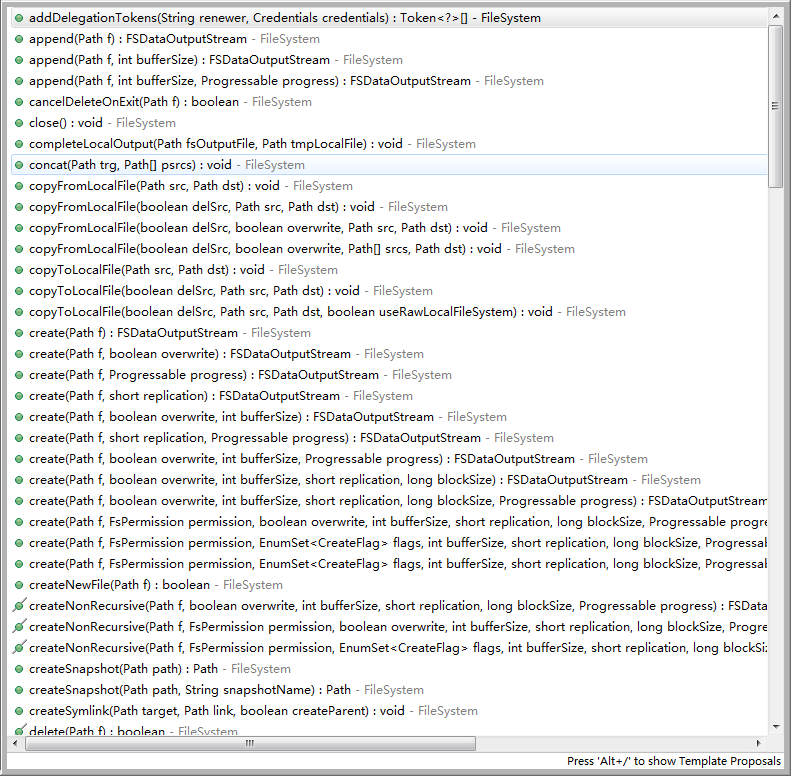
而我们的操作目标是HDFS，所以获取到的fs对象应该是DistributedFileSystem的实例；

get方法是从何处判断具体实例化那种客户端类呢？

**——从conf中的一个参数 fs.defaultFS的配置值判断；**

如果我们的代码中没有指定fs.defaultFS，并且工程classpath下也没有给定相应的配置，conf中的默认值就来自于hadoop的jar包中的core-default.xml，默认值为： file:///，则获取的将不是一个DistributedFileSystem的实例，而是一个本地文件系统的客户端对象

## 7.3 DistributedFileSystem实例对象所具备的方法



## 7.4 HDFS客户端操作数据代码示例：

### 7.4.1 文件的增删改查

|  |
| --- |
| public class HdfsClient {  FileSystem fs = null;  @Before  public void init() throws Exception {  // 构造一个配置参数对象，设置一个参数：我们要访问的hdfs的URI  // 从而FileSystem.get()方法就知道应该是去构造一个访问hdfs文件系统的客户端，以及hdfs的访问地址  // new Configuration();的时候，它就会去加载jar包中的hdfs-default.xml  // 然后再加载classpath下的hdfs-site.xml  Configuration conf = new Configuration();  conf.set("fs.defaultFS", "hdfs://hdp-node01:9000");  /\*\*  \* 参数优先级： 1、客户端代码中设置的值 2、classpath下的用户自定义配置文件 3、然后是服务器的默认配置  \*/  conf.set("dfs.replication", "3");  // 获取一个hdfs的访问客户端，根据参数，这个实例应该是DistributedFileSystem的实例  // fs = FileSystem.get(conf);  // 如果这样去获取，那conf里面就可以不要配"fs.defaultFS"参数，而且，这个客户端的身份标识已经是hadoop用户  fs = FileSystem.get(new URI("hdfs://hdp-node01:9000"), conf, "hadoop");  }  /\*\*  \* 往hdfs上传文件  \*  \* @throws Exception  \*/  @Test  public void testAddFileToHdfs() throws Exception {  // 要上传的文件所在的本地路径  Path src = new Path("g:/redis-recommend.zip");  // 要上传到hdfs的目标路径  Path dst = new Path("/aaa");  fs.copyFromLocalFile(src, dst);  fs.close();  }  /\*\*  \* 从hdfs中复制文件到本地文件系统  \*  \* @throws IOException  \* @throws IllegalArgumentException  \*/  @Test  public void testDownloadFileToLocal() throws IllegalArgumentException, IOException {  fs.copyToLocalFile(new Path("/jdk-7u65-linux-i586.tar.gz"), new Path("d:/"));  fs.close();  }  @Test  public void testMkdirAndDeleteAndRename() throws IllegalArgumentException, IOException {  // 创建目录  fs.mkdirs(new Path("/a1/b1/c1"));  // 删除文件夹 ，如果是非空文件夹，参数2必须给值true  fs.delete(new Path("/aaa"), true);  // 重命名文件或文件夹  fs.rename(new Path("/a1"), new Path("/a2"));  }  /\*\*  \* 查看目录信息，只显示文件  \*  \* @throws IOException  \* @throws IllegalArgumentException  \* @throws FileNotFoundException  \*/  @Test  public void testListFiles() throws FileNotFoundException, IllegalArgumentException, IOException {  // 思考：为什么返回迭代器，而不是List之类的容器  RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);  while (listFiles.hasNext()) {  LocatedFileStatus fileStatus = listFiles.next();  System.out.println(fileStatus.getPath().getName());  System.out.println(fileStatus.getBlockSize());  System.out.println(fileStatus.getPermission());  System.out.println(fileStatus.getLen());  BlockLocation[] blockLocations = fileStatus.getBlockLocations();  for (BlockLocation bl : blockLocations) {  System.out.println("block-length:" + bl.getLength() + "--" + "block-offset:" + bl.getOffset());  String[] hosts = bl.getHosts();  for (String host : hosts) {  System.out.println(host);  }  }  System.out.println("--------------为angelababy打印的分割线--------------");  }  }  /\*\*  \* 查看文件及文件夹信息  \*  \* @throws IOException  \* @throws IllegalArgumentException  \* @throws FileNotFoundException  \*/  @Test  public void testListAll() throws FileNotFoundException, IllegalArgumentException, IOException {  FileStatus[] listStatus = fs.listStatus(new Path("/"));  String flag = "d-- ";  for (FileStatus fstatus : listStatus) {  if (fstatus.isFile()) flag = "f-- ";  System.out.println(flag + fstatus.getPath().getName());  }  }  } |

### 7.4.2 通过流的方式访问hdfs

|  |
| --- |
| */\*\**  *\* 相对那些封装好的方法而言的更底层一些的操作方式*  *\* 上层那些mapreduce spark等运算框架，去hdfs中获取数据的时候，就是调的这种底层的api*  *\* @author*  *\**  *\*/*  *public class StreamAccess {*    *FileSystem fs = null;*  *@Before*  *public void init() throws Exception {*  *Configuration conf = new Configuration();*  *fs = FileSystem.get(new URI("hdfs://hdp-node01:9000"), conf, "hadoop");*  *}*        *@Test*  *public void testDownLoadFileToLocal() throws IllegalArgumentException, IOException{*    *//先获取一个文件的输入流----针对hdfs上的*  *FSDataInputStream in = fs.open(new Path("/jdk-7u65-linux-i586.tar.gz"));*    *//再构造一个文件的输出流----针对本地的*  *FileOutputStream out = new FileOutputStream(new File("c:/jdk.tar.gz"));*    *//再将输入流中数据传输到输出流*  *IOUtils.copyBytes(in, out, 4096);*      *}*      */\*\**  *\* hdfs支持随机定位进行文件读取，而且可以方便地读取指定长度*  *\* 用于上层分布式运算框架并发处理数据*  *\* @throws IllegalArgumentException*  *\* @throws IOException*  *\*/*  *@Test*  *public void testRandomAccess() throws IllegalArgumentException, IOException{*  *//先获取一个文件的输入流----针对hdfs上的*  *FSDataInputStream in = fs.open(new Path("/iloveyou.txt"));*      *//可以将流的起始偏移量进行自定义*  *in.seek(22);*    *//再构造一个文件的输出流----针对本地的*  *FileOutputStream out = new FileOutputStream(new File("c:/iloveyou.line.2.txt"));*    *IOUtils.copyBytes(in,out,19L,true);*    *}*        */\*\**  *\* 显示hdfs上文件的内容*  *\* @throws IOException*  *\* @throws IllegalArgumentException*  *\*/*  *@Test*  *public void testCat() throws IllegalArgumentException, IOException{*    *FSDataInputStream in = fs.open(new Path("/iloveyou.txt"));*    *IOUtils.copyBytes(in, System.out, 1024);*  *}*  *}* |

### 7.4.3 场景编程

在mapreduce 、spark等运算框架中，有一个核心思想就是将运算移往数据，或者说，就是要在并发计算中尽可能让运算本地化，这就需要获取数据所在位置的信息并进行相应范围读取

以下模拟实现：获取一个文件的所有block位置信息，然后读取指定block中的内容

|  |
| --- |
| *@Test*  *public void testCat() throws IllegalArgumentException, IOException{*    *FSDataInputStream in = fs.open(new Path("/weblog/input/access.log.10"));*  *//拿到文件信息*  *FileStatus[] listStatus = fs.listStatus(new Path("/weblog/input/access.log.10"));*  *//获取这个文件的所有block的信息*  *BlockLocation[] fileBlockLocations = fs.getFileBlockLocations(listStatus[0], 0L, listStatus[0].getLen());*  *//第一个block的长度*  *long length = fileBlockLocations[0].getLength();*  *//第一个block的起始偏移量*  *long offset = fileBlockLocations[0].getOffset();*    *System.out.println(length);*  *System.out.println(offset);*    *//获取第一个block写入输出流*  *// IOUtils.copyBytes(in, System.out, (int)length);*  *byte[] b = new byte[4096];*    *FileOutputStream os = new FileOutputStream(new File("d:/block0"));*  *while(in.read(offset, b, 0, 4096)!=-1){*  *os.write(b);*  *offset += 4096;*  *if(offset>=length) return;*  *};*  *os.flush();*  *os.close();*  *in.close();*  *}* |

# 8. 案例1：开发shell采集脚本

## 8.1需求说明

点击流日志每天都10T，在业务应用服务器上，需要准实时上传至数据仓库（Hadoop HDFS）上

## 8.2需求分析

一般上传文件都是在凌晨24点操作，由于很多种类的业务数据都要在晚上进行传输，为了减轻服务器的压力**，避开高峰期**。

如果需要伪实时的上传，则采用定时上传的方式

## 8.3技术分析

**HDFS SHELL**: hadoop fs –put xxxx.tar /data 还可以使用 Java Api

满足上传一个文件，不能满足定时、周期性传入。

**定时调度器**：

**Linux crontab**

crontab -e

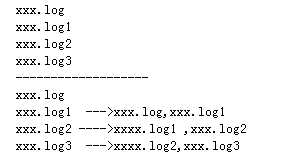
\*/5 \* \* \* \* $home/bin/command.sh //五分钟执行一次

系统会自动执行脚本，每5分钟一次，执行时判断文件是否符合上传规则，符合则上传

## 8.4实现流程

### 8.4.1日志产生程序

日志产生程序将日志生成后，产生一个一个的文件，使用滚动模式创建文件名。



日志生成的逻辑由业务系统决定，比如在log4j配置文件中配置生成规则，如：当xxxx.log 等于10G时，滚动生成新日志

|  |
| --- |
| *log4j.logger.msg=info,msg*  *log4j.appender.msg=cn.maoxiangyi.MyRollingFileAppender*  *log4j.appender.msg.layout=org.apache.log4j.PatternLayout*  *log4j.appender.msg.layout.ConversionPattern=%m%n*  *log4j.appender.msg.datePattern='.'yyyy-MM-dd*  *log4j.appender.msg.Threshold=info*  *log4j.appender.msg.append=true*  *log4j.appender.msg.encoding=UTF-8*  *log4j.appender.msg.MaxBackupIndex=100*  *log4j.appender.msg.MaxFileSize=10GB*  *log4j.appender.msg.File=/home/hadoop/logs/log/access.log* |

细节：

1. 如果日志文件后缀是1\2\3等数字，该文件满足需求可以上传的话。把该文件移动到准备上传的工作区间。
2. 工作区间有文件之后，可以使用hadoop put命令将文件上传。

阶段问题：

1. 待上传文件的工作区间的文件，在上传完成之后，是否需要删除掉。

### 8.4.2伪代码

使用ls命令读取指定路径下的所有文件信息，

ls | while read line

//判断line这个文件名称是否符合规则

if line=access.log.\* (

将文件移动到待上传的工作区间

)

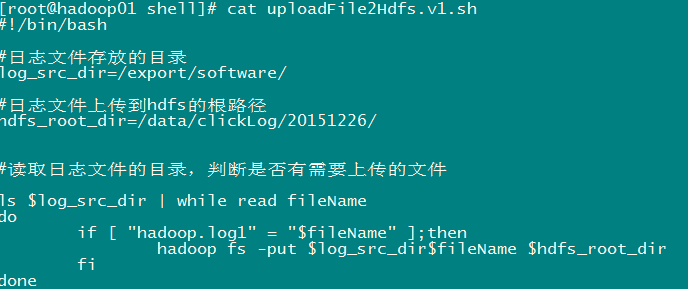
//批量上传工作区间的文件

hadoop fs –put xxx

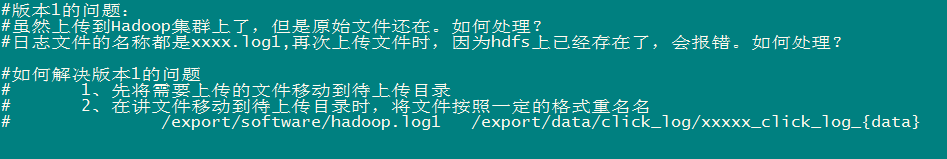
**脚本写完之后**，配置linux定时任务，每5分钟运行一次。

## 8.5代码实现

代码第一版本，实现基本的上传功能和定时调度功能



代码第二版本：增强版V2(基本能用，还是不够健全)





## 8.6效果展示及操作步骤

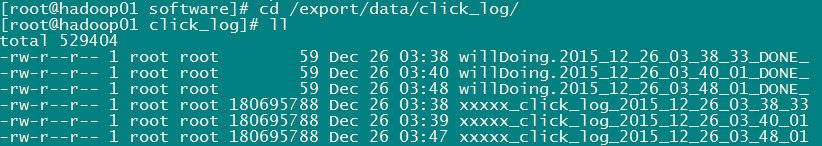
1、日志收集文件收集数据，并将数据保存起来，效果如下：



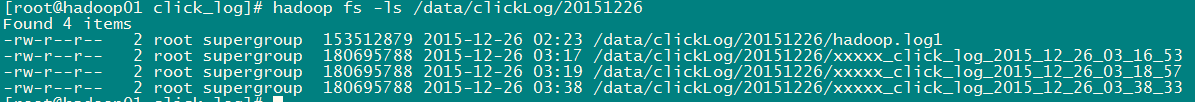
2、上传程序通过crontab定时调度



3、程序运行时产生的临时文件



4、Hadoo hdfs上的效果



# 9. 案例2：开发JAVA采集程序

## 9.1 需求

从外部购买数据，数据提供方会实时将数据推送到6台FTP服务器上，我方部署6台接口采集机来对接采集数据，并上传到HDFS中

提供商在FTP上生成数据的规则是以小时为单位建立文件夹(2016-03-11-10)，每分钟生成一个文件（00.dat,01.data,02.dat,........）

提供方不提供数据备份，推送到FTP服务器的数据如果丢失，不再重新提供，且FTP服务器磁盘空间有限，最多存储最近10小时内的数据

由于每一个文件比较小，只有150M左右，因此，我方在上传到HDFS过程中，需要将15分钟时段的数据合并成一个文件上传到HDFS

为了区分数据丢失的责任，我方在下载数据时最好进行校验

## 9.2 设计分析

