|  |
| --- |
| 葵花宝典 |
|  |
| **编者：郑可佳**  **日期：2017/4/17** |
|  |



目 录

[葵花宝典 0](#_Toc481840237)

[1 Linux基础 4](#_Toc481840238)

[1.1 Linux分区操作 4](#_Toc481840239)

[1.2 Linux命令 4](#_Toc481840240)

[1.2.1 grep命令 4](#_Toc481840241)

[1.2.2 Sed命令 4](#_Toc481840242)

[1.2.3 awk命令 5](#_Toc481840243)

[1.2.4 查看某端口是否被占用 5](#_Toc481840244)

[1.3 Shell编程 5](#_Toc481840245)

[1.3.1 变量 5](#_Toc481840246)

[1.3.2 Shell算数运算 6](#_Toc481840247)

[1.3.3 Shell内置测试判断 7](#_Toc481840248)

[1.3.4 控制语句 9](#_Toc481840249)

[1.3.5 函数使用 13](#_Toc481840250)

[1.3.6 Crontab计划任务 13](#_Toc481840251)

[2 Windows系统执行hadoop程序 14](#_Toc481840252)

[3 Maven基础 15](#_Toc481840253)

[3.1 环境搭建 15](#_Toc481840254)

[3.2 pom文件 15](#_Toc481840255)

[3.2.1 配置 15](#_Toc481840256)

[3.3 maven 常用命令 15](#_Toc481840257)

[3.4 Maven配置镜像仓库 16](#_Toc481840258)

[3.5 dos命令生成Maven骨架 16](#_Toc481840259)

[3.6 maven 依赖传递 17](#_Toc481840260)

[3.7 maven生命周期 17](#_Toc481840261)

[3.8 maven 常见错误 17](#_Toc481840262)

[4 Hadoop基础 17](#_Toc481840263)

[4.1 Hadoop安装 17](#_Toc481840264)

[4.2 MapReduce深入理解 17](#_Toc481840265)

[4.3 MapReduce二次排序 17](#_Toc481840266)

[4.3.1 组合key 17](#_Toc481840267)

[4.3.2 分区规则 19](#_Toc481840268)

[4.3.3 分组规则 20](#_Toc481840269)

[4.4 MapReduce Join 21](#_Toc481840270)

[4.5 HA机制 21](#_Toc481840271)

[4.5.1 需求背景 21](#_Toc481840272)

[4.5.2 HA配置 22](#_Toc481840273)

[4.6 Resourcemanager HA 22](#_Toc481840274)

[4.7 HDFS Federation 22](#_Toc481840275)

[4.8 分布式拷贝工具DistCP 22](#_Toc481840276)

[5 Zookeeper基础 22](#_Toc481840277)

[5.1 Zookeeper安装 22](#_Toc481840278)

[6 Hive基础 22](#_Toc481840279)

[6.1 MySQL安装 22](#_Toc481840280)

[6.2 Hive安装 23](#_Toc481840281)

[6.3 Hive使用 23](#_Toc481840282)

[6.3.1 分区表 23](#_Toc481840283)

[6.3.2 加载数据 23](#_Toc481840284)

[6.3.3 导出数据 24](#_Toc481840285)

[6.3.4 having关键词 24](#_Toc481840286)

[6.3.5 数据导入导出 24](#_Toc481840287)

[6.3.6 四个by关键词 25](#_Toc481840288)

[6.3.7 UDF函数 25](#_Toc481840289)

[6.3.8 Hive的三种启动方式 26](#_Toc481840290)

[6.3.9 数据存储 26](#_Toc481840291)

[6.3.10 SQL执行Python脚本 29](#_Toc481840292)

[7 sqoop基础 30](#_Toc481840293)

[7.1 sqoop安装 30](#_Toc481840294)

[7.2 sqoop使用 30](#_Toc481840295)

[7.2.1 查看数据库 30](#_Toc481840296)

[7.2.2 导入数据 30](#_Toc481840297)

[7.2.3 导出数据 34](#_Toc481840298)

[7.2.4 Shell导入导出数据 34](#_Toc481840299)

[8 flume基础 35](#_Toc481840300)

[8.1 flume安装 35](#_Toc481840301)

[8.2 flume使用 35](#_Toc481840302)

[8.2.1 安装rpm 35](#_Toc481840303)

[8.2.2 启动xinetd服务 35](#_Toc481840304)

[8.2.3 启动flume服务 35](#_Toc481840305)

[8.2.4 连接a1.conf配置文件指定的端口 36](#_Toc481840306)

[8.2.5 收集hive日志 36](#_Toc481840307)

[8.2.6 动态创建hdfs文件夹 37](#_Toc481840308)

[9 Oozie基础 37](#_Toc481840309)

[9.1 Oozie安装 37](#_Toc481840310)

[9.2 运行job 37](#_Toc481840311)

[9.3 编写MapReduce action 38](#_Toc481840312)

[9.4 编写hive action 40](#_Toc481840313)

[9.5 编写sqoop action 41](#_Toc481840314)

[9.6 编写shell action 41](#_Toc481840315)

[9.7 Oozie修改时区 42](#_Toc481840316)

[9.7.1 修改oozie-site.xml文件 42](#_Toc481840317)

[9.7.2 修改web页面显示时区 42](#_Toc481840318)

[9.8 编写定时任务 43](#_Toc481840319)

[9.9 测试cron任务 43](#_Toc481840320)

[9.10 多个action 44](#_Toc481840321)

[10 Hue基础 45](#_Toc481840322)

[10.1 Hue安装 45](#_Toc481840323)

[11 Hbase基础 45](#_Toc481840324)

[11.1 Hbase安装 45](#_Toc481840325)

[11.2 Hbase使用 45](#_Toc481840326)

[11.2.1 Java api操作hbase表 45](#_Toc481840327)

[11.2.2 运行hbase自带MapReduce程序 47](#_Toc481840328)

[11.2.3 编写hbase的MapReduce 47](#_Toc481840329)

[11.2.4 Importtsv导入hdfs文件到hbase表 48](#_Toc481840330)

[11.2.5 bulk批量导入数据到hbase表 48](#_Toc481840331)

[11.2.6 表预分区 49](#_Toc481840332)

[11.2.7 Hbase与hive集成 49](#_Toc481840333)

[11.2.8 Hbase与hue集成 50](#_Toc481840334)

# Linux基础

## Linux分区操作

（1）磁盘分区

主分区 + 扩展分区(逻辑分区) <= 4 3+1 2+1 1+1

Linux系统默认所有设备文件都在/dev下面

/dev/sda --硬盘1 sda1第一个分区 sda2第二个分区 sda3第三个分区

/dev/sdb --硬盘2 sdb1第一个分区 sdb2第二个分区 sdb3第三个分区

/dev/sdc --硬盘3

# fdisk -l 查看系统所有硬盘的分区情况

（1）系统一共有几块硬盘，每个硬盘的容量大小

（2）每个硬盘的分区情况（硬盘空间是否还有剩余）

分区步骤：

(1)fdisk 设备名称（/dev/sdb）

(2)partx -a /dev/sdb

(3)格式化 # mkfs.ext4 /dev/sdb6

(4)挂载 # mount /dev/sdb6 /mnt （临时生效）

修改/etc/fstab （永久生效）

/dev/sdb6 /wwwroot ext4 defaults 0 0 （/wwwroot挂载点）

## Linux命令

### grep命令

案例：取出符合某条件并且不符合某条件的字符串

ifconfig | grep 'inet addr:' |grep -v '127.0.0.1'

（包含inet addr 并且不包含127.0.0.1 -v表示反选）

### Sed命令

用法：sed '匹配条件/执行的动作' /etc/passwd

或者

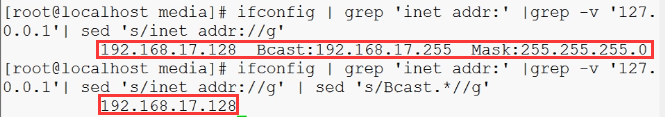
cat /etc/passwd | sed '匹配条件/执行的动作'

替换： sed 's/old/new/g'

s：替换 old：要替换的字符串 new：替换后的字符串 g:整行替换所有匹配的字符串

案例：取出符合某条件并且不符合某条件的字符串并进行替换

ifconfig | grep 'inet addr:' |grep -v '127.0.0.1'| sed 's/inet addr://g' | sed 's/Bcast.\*//g'



### awk命令

用法：awk -F: '{print $1}' /etc/passwd

-F指定分割符（这里为:） $0代表整行 $1代表第一列 $2第二列……. /etc/passwd操作的文件

用途：取出目标文件的低级列字符串

### 查看某端口是否被占用

lsof -i:3306

或

netstat -anp|grep 80

## Shell编程

### 变量

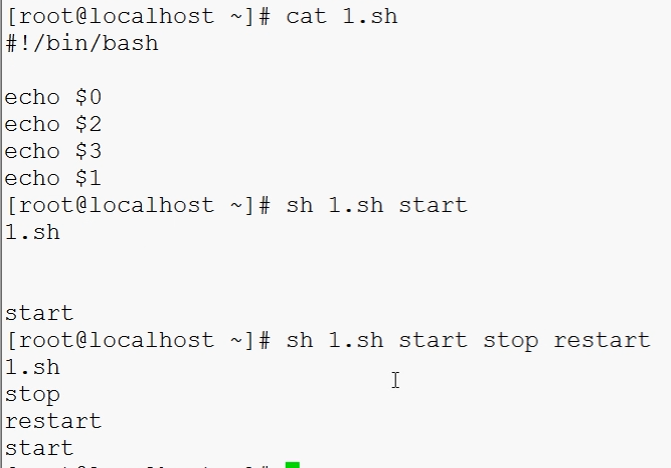
#### 位置变量

--通常和脚本联合使用

--$0 脚本名称

--$1、$2、$3、$4…$9 位置参数

案例：新建1.sh脚本，运行该脚本，并传入参数，$0对应脚本名称，$1 $2 ……分别顺序对应脚本传入的参数



#### 预定义变量

--$? 表示程序退出的代表(一般0代表执行成功，非0表示执行失败)

--$# 代表当前shell的参数个数

--$\* 代表所有参数（整体读取）

--$@ 代表所有参数（逐个读取）

--$$ 当前进程的PID

--$! 后台运行的最后一个进程的PID号

#### 自定义变量

--语法格式为：name=[value]

--注意：

变量等号两边不能有空格

变量对大小写敏感

--定义好后使用（$变量名）来调用变量的值

### Shell算数运算

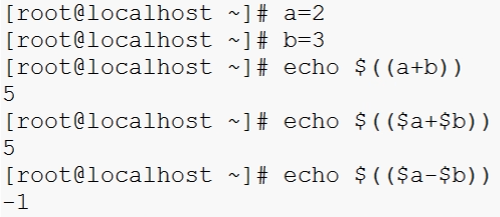
-- + - \* / %

--$((expression))

--$[expression]

--expr expression

（$((expression))与$[expression]效果一样）



（在表达式内变量前的$可加可不加）

a=2

b=3

# echo $(($a+$b))

# echo $[$a+$b]

# expr 5 + 2

# expr $a + $b

（expr运算符两边一定要有空格，并且引用变量是一定要有$符号）

### Shell内置测试判断

两种方式（效果一样）：

--test 测试表达式

--[ 测试表达式 ] 最常见的

--注意： 测试表达式与中括号直接一定要有空格

#### 数值比较

-eq 等于则为真

-ne 不等于则为真

-gt 大于则为真

-ge 大于等于则为真

-lt 小于则为真

-le 小于等于则为真

&& 逻辑与

-- cmd1 && cmd2 cmd1成功了才会执行cmd2

|| 逻辑或

-- cmd1 || cmd2 cmd1失败了才会执行cmd2

; 无逻辑关系

-- cmd1 ; cmd2 cmd1执行完后，执行cmd2

案例1：

[root@localhost ~]# echo $a $b

2 3

[root@localhost ~]# test $a -lt $b

[root@localhost ~]# echo $?

0

[root@localhost ~]# test $a -gt $b

[root@localhost ~]# echo $?

1:

案例2：

[root@localhost ~]# echo $a $b

2 3

[root@localhost ~]# [ $a -lt $b ]

[root@localhost ~]# echo $?

0

[root@localhost ~]# [ $a -gt $b ]

[root@localhost ~]# echo $?

1

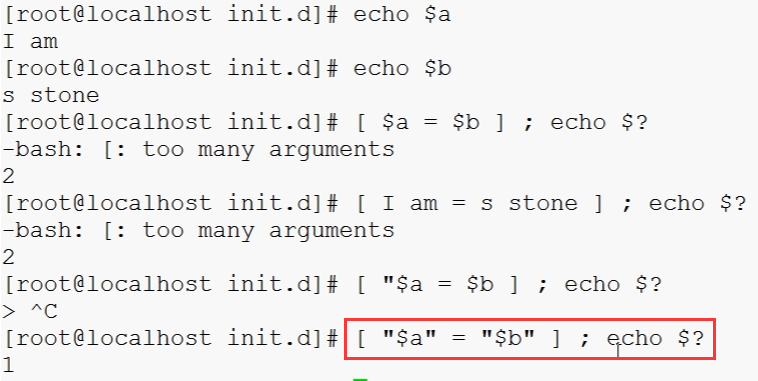
#### 字符串测试

= 等于则为真

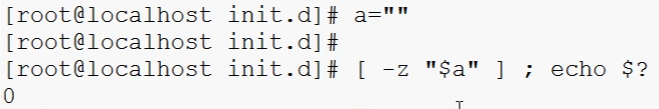
!= 不相等则为真

-z 字串 字串长度0则为真

-n 字串 字串长度不0则为真



（字符串比较时最好给变量加双引号）



#### 文件测试

-e 文件名 如果文件存在则为真

-d 文件名 如果文件存在且为目录则为真

-f 文件名 如果文件存在且为普通文件则为真

-r 文件名 如果文件存在且可读则为真

-w 文件名 如果文件存在且可写则为真

-x 文件名 如果文件存在且可执行则为真

-s 文件名 如果文件存在且至少有一个字符则为真

-b 文件名 如果文件存在且为块特殊文件则为真

-c 文件名 如果文件存在且为字符型特殊文件则为真

（文件名也可以为目录）

#### 其它

①Linux还提供了非（！）、或（-o）、与（-a）三个逻辑操作符，用于将测试条件连接起来，其优先顺序为：！最高，-a次之，-o最低。

②取字符串操作

${变量名:offset:length}

案例1：

[root@localhost ~]# a="201604091527"

[root@localhost ~]# echo $a

201604091527

[root@localhost ~]# echo ${a:2:3}

160

[root@localhost ~]# echo ${a:4:2}

04

案例2：

取字符串 echo ${变量名%.\*}

[root@localhost ~]# a="123.log"

[root@localhost ~]# echo ${a%.\*}

123

### 控制语句

#### for语句

语法1：

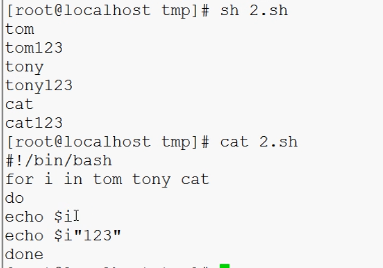
for 变量 in 值1 值2 … 值N

do

命令序列

Done

案例1：



案例2：在某文件夹下创建50个名字为dir1 dir2 ……..的文件

#!/bin/bash

for i in {1..50}

do

mkdir /usr/local/src/dir$i

done

（有同样效果：

#!/bin/bash

for i in `seq 50` #反引号代表命令预先执行

do

mkdir /usr/local/src/dir$i

done

）

语法2：

for ((初始化变量；结束循环条件；运算))

do

命令序列

Done

案例1：计算1+2+…+50的值

#!/bin/bash

for((i=1;i<=10;i++));do

SUM=$((SUM+i))

done

echo $SUM

#### while语句

语法1：

while [ 条件 ]

do

命令序列

Done

案例1：

#!/bin/bash

i=1

while [ $i -le 10 ]

do

SUM=$((SUM+i))

i=$[i+1]

done

echo $SUM

语法2：

while read -r line

do

命令序列

Done

案例1：读取并输出文件内容

#!/bin/bash

while read -r line

do

echo $line:HELLO

done < /etc/passwd

#< /etc/passwd为将该文件内容输出给while语句，只能结合read使用

案例2：读取并输出文件每行以：分隔的第一列的内容并在每行内容后面添加:HELLO

#!/bin/bash

while read -r line

do

echo `echo $line | awk -F: '{print $1}'`:HELLO

done < /etc/passwd

#### if与case语句

##### if语句

语法1：

if 条件

then

命令序列

fi

语法2：

if 条件

then

命令序列

else

命令序列

Fi

语法3：

if 条件

then

命令序列

elif 条件

then

命令序列

elif 条件

then

命令序列

else

命令序列

Fi

案例1：判断是否存在/tmp/123目录，存在则列出该目录的文件否则新建该目录

#!/bin/bash

if [ -d /tmp/123 ];then

ls /tmp/123

else

mkdir /tmp/123

fi

##### case语句

语法1：

case $变量名称 in

条件1）

命令序列

；；

条件2）

命令序列

；；

条件3）

命令序列

；；

\*）

Esac

语法2：

case $变量名称 in

条件1|条件2）

命令序列

；；

条件3|条件4）

命令序列

；；

条件5|条件6）

命令序列

；；

\*）

esac

案例1：

#!/bin/bash

case $1 in

top)

top # top命令用来显示执行中的程序进程

;;

free)

free # free命令用来显示内存的使用情况

;;

df)

df #查看磁盘盘剩余空间

;;

\*)

echo "usage:$0{top|free|df}" #前面都不匹配输出

esac

### 函数使用

语法1：

name() {

命令序列

}

语法2：

function name {

命令序列

}

案例1：声明sum加法函数并调用

#!/bin/bash

sum(){

echo $(($1+$2))

}

sum 5 6

### Crontab计划任务

#### 一次性计划任务

at --指定时间执行特定命令

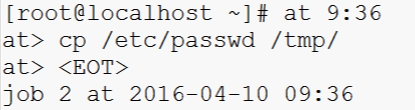
用法：at [时间]

案例1：

at 4:17 #指定在当天凌晨4点17分执行计划任务

at> cp /etc/passwd /tmp #计划任务内容

at> <EOT> #输入完成后，按ctrl+d结束



#### 周期性计划任务

crontab --周期性执行计划任务

用法：crontab [-u 用户] [-l|-r|-e]

参数：

-u：指定某个用户，不加-u选项则为当前用户

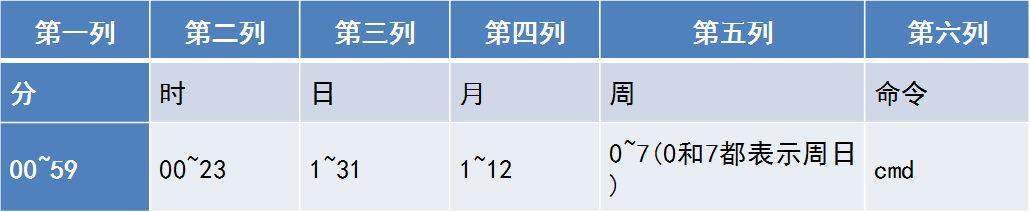
-e：制定计划任务

-l：列出计划任务

①确保crontab启动

service crond status #查看crontab状态

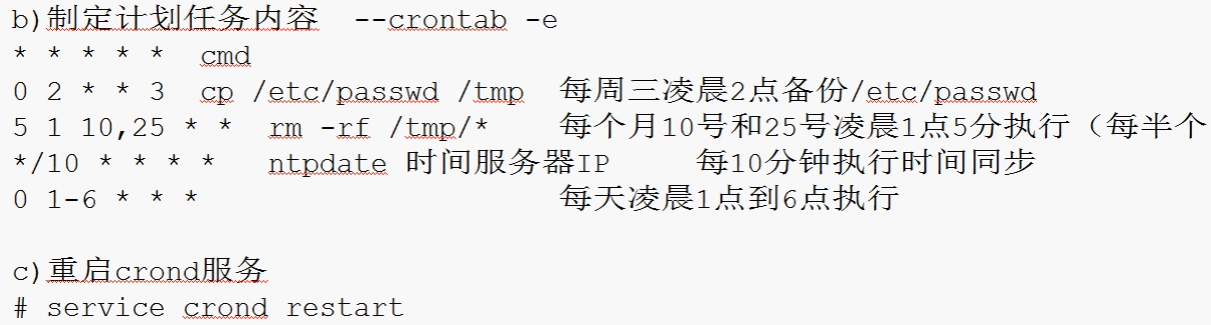
②任务格式



如上图，如果时间是时间段，可以使用横杠（-）来表示一段连续的时间，使用（，）表示若干不连续的时间，使用星号（\*）表示所有的时间，使用除号（/）表示间隔时间。

案例1：每十分钟执行一次时间同步

\*/10 \* \* \* \* ntpdate 时间服务器IP



添加新计划任务后要重启crontab服务。

# Windows系统执行hadoop程序

1. 解压hadoop压缩包
2. 下载winutils.exe将其放到bin目录
3. 配置环境变量

HADOOP\_HOME

D:\Hadoop\hadoop-2.5.0

Path

%HADOOP\_HOME%\bin

1. 重启系统

# Maven基础

## 环境搭建

1. 添加系统变量

我的电脑（计算机）----右击--->属性--->高级系统设置--->环境变量---->系统变量下新建的M2\_HOME=E:\softwares\apache-maven-3.3.9

---->添加M2\_HOME变量到path 路径中

1. 验证

验证环境变量：mvn -v 或者 mvn –version

## pom文件

### 配置

<!-- 指定当前pom 的版本-->

<modelVersion>4.0.0</modelVersion>

<!-- 项目包名: 公司地址名称反写+项目名称-->

<groupId>com.ibeifeng.maven</groupId>

<!--项目模块名称：一般为 项目名-模块名 -->

<artifactId>maven-demo1</artifactId>

<!--

标识当前项目版本号

第一个.号之前的数字标识 表示大版本号

第二个.号之前的数字标识 表示分支版本号

第三个.号之前的数字标识 表示小版本号

SNAPSHOT：快照版本

Release 发布版本

Alpha :内部测试版本

GA：正式发布的版本

-->

<version>1.0.0SNAPSHOT</version>

## maven 常用命令

1. mvn -v 或者 mvn -version：

验证环境变量

2. mvn help:system ：

打印出所有的系统属性和环境变量

3. mvn compile：

编译项目源代码（不会编译test 目录的元代）（会产生target 文件）

4. mvn test: 运行应用程序中的单元测试

5. mvn test-compile 编译测试代码，compile 之后生成的targer 文件夹 主程序编译在classes 文件夹下面，测试程序代码放在test-classes 文件夹下

6. mvn clean 删除target 文件夹

## Maven配置镜像仓库

镜像仓库：为国内的服务器，下载速度更快

conf/settings.xml

--mirrors

--mirror:

<mirror>

<id>repo2</id>

<mirrorOf>central</mirrorOf>

<name>Human Readable Name for this Mirror.</name>

<url>http://repo2.maven.org/maven2/</url>

</mirror>

NOTE:

maven setttings.xml 不要去修改maven 安装目录下的conf/settings.xml （全局），推荐大家 拷贝settings.xml 到你对应的本地仓库目录下面(C:\Users\daibin\.m2/settings.xml)

## dos命令生成Maven骨架

简单使用：

mvn archetype:generate

817 回车 提供的骨架 maven-archetype-quickstart

一步到位：

mvn archetype:generate -DgroupId=com.ibeifeng.maven04 -DartifactId=maven-demo04 -Dversion=1.0-SNAPSHOT -Dpackage=com.ibeifeng.maven04

-DgroupId 公司名称反写+项目名称

-DartifactId 项目名称-模块名称

-Dversion 项目版本号

-Dpackage 源码包目录

NOTE:DgroupId 和 Dpackage 是相同的，官方推荐的，看起来更加清晰

maven的Responsitory 里面支持的archetype 大概有1600+，

其实常用的archetypee 就那么几个

1.quick start

2.webapp

3.simple

很自然的就会去想到 能不能用什么简便的方式只从上面这个list 里选择就可以

怎么去实现？

解决思路：

1. mvn archetype:crawl

:会在本地仓库目录(C:\Users\daibin\.m2\repository)下生成archetype-catalog.xml

2. 将生成archetype-catalog.xml 上移到C:\Users\daibin\.m2 目录下面

3.mvn archetype:generate -DarchetypeCatalog=local

## maven 依赖传递

[maven 依赖传递](参考文件/maven%20依赖传递.docx)

## maven生命周期

[maven生命周期](参考文件/maven生命周期.docx)

## maven 常见错误

[maven 常见错误](参考文件/maven%20常见错误.docx)

# Hadoop基础

## Hadoop安装

[Hadoop伪分布式安装教程](参考文件/Hadoop伪分布式安装教程.docx)

[Hadoop分布式安装教程](参考文件/Hadoop分布式安装教程.docx)

## MapReduce深入理解

[MapReduce过程图解](参考文件/MapReduce过程.png)

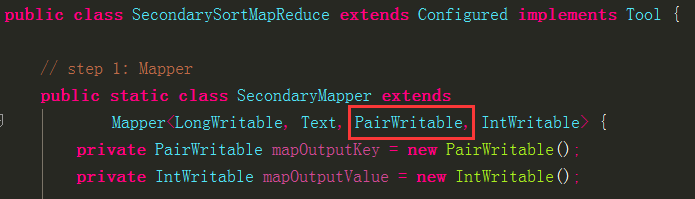
分区时不同的key可能分到不同的分区，分区后进行排序再分组，而分组时相同的key一定会分到相同的组，并且会形成一个keyvalues对，values为该key的所有value的集合。

进入reduce阶段时，若ReduceTasks个数设置为一个，即setup函数和cleanup函数只会被调用一次，调用reduce函数时传入的数据为同一个分组的数据，有多少个分组reduce函数就会被调用多少次。若ReduceTasks个数设置为多个，即各个分组会被分到不同的ReduceTasks，每一个ReduceTasks启动时都会调用一次setup函数，结束时调用一次cleanup函数，若分到该ReduceTask的分组为n个即reduce函数会被调用n次。

## MapReduce二次排序

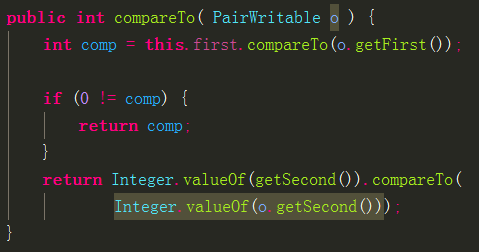
### 组合key

[自定义数据类型作为组合key](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/SecondarySortMapReduce.java)

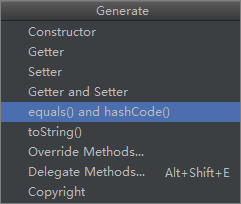


[自定义数据类型生成步骤：](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/PairWritable.java)

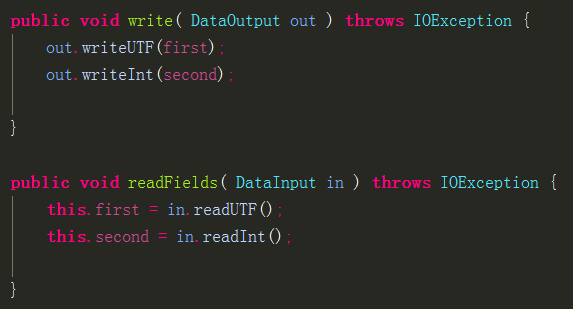
1. 实现WritableComparable接口，添加未实现方法；
2. 添加类成员变量并生成get和set方法；
3. 生成无参和有参构造方法；
4. 实现compareTo方法，用来定义排序规则；



1. 生成hashCode和equals方法；



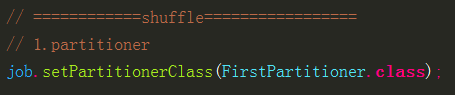
1. 生成tostring方法；
2. 实现写和度方法（两个方法参数顺序要一致）



### 分区规则

[保证原来的分区，需要自定义分区规则](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/FirstPartitioner.java)

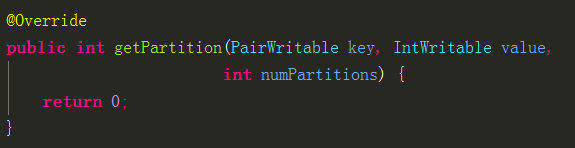
1. Job设置分区类



1. 新建分区类，继承Partitioner类，泛型中的类为Map输出的类型



1. 生成getPartition方法



1. 实现getPartition方法

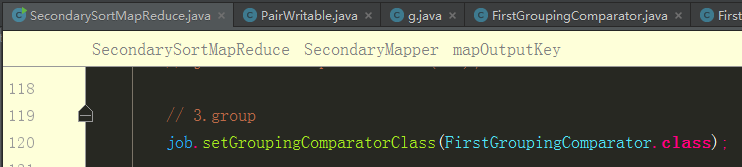
hashPartitioner分区是MR框架默认的分区计算方法



### 分组规则

[保证原来的分组，需要自定义分组规则。](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/FirstGroupingComparator.java)

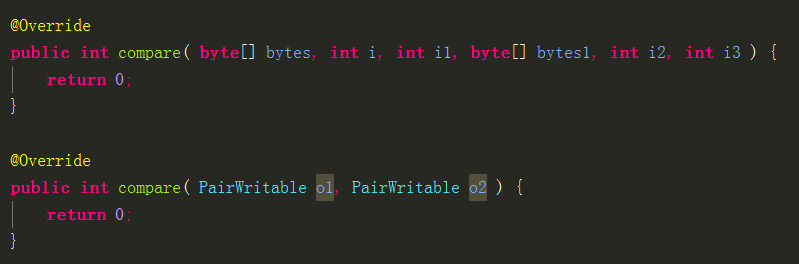
1. Job设置分组类



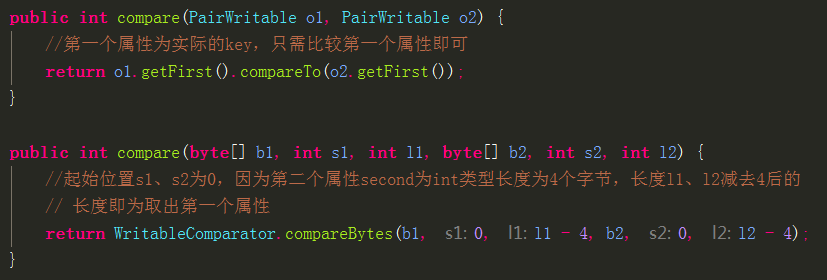
1. 新建分组类，实现RawComparator接口，泛型中的类为要分组排序的数据类



1. 生成compare方法



1. 实现compare方法



## MapReduce Join

[示例代码](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/join)

1、举例，网上购物（账户，订单）

账户信息（customer）----》小表

用户ID、名称、收货地址、电话联系方式

cid cname address phone

订单信息（order）-----》大表

订单ID、用户ID、价格、名称

oid cid price pname

2、组合，长信息

cid cname address phone oid cid price pname

3、设计<key value>，两张表中的cid作为一个公共的连接点，map输出的key

stepup()准备工作--》在map没有处理之前，先将小表customer的数据读到，存储，加到内存中

map()-----》读取大表order

cleanup()清理工作

将cid作为一个key,<cid,customerinfo>

<cid,orderinfo>

4、map join适合数据量较小的

5、reduce join

Reduce input key value

<cid,list(cInfo,orderInfo,orderInfo,orderInfo,orderInfo....)>

map

customer

<cid,cInfo>

cInfo

tag：customer , order

data: cInfo， ordrInfo

order

<cid,orderInfo>

设置一个标识：判断是用户还是订单

reduce join 可以叫 shuffle join

NOTE：最终reduce函数接受到的数据都是经过shuffle阶段后把相同key放在一起的数据，两表之间的join正是利用了shuffle阶段将相同key发送给同一个reduce接收的特点成功实现join。

## HA机制

### 需求背景

1、元数据同步

-》保证两个namenode内存中存储的文件系统元数据是一致的

2、思路，namenode启动过程

-》读取fsimage和edits文件-》读取后会生成新的fsimage和edits文件

-》另一个namenode同样需要去读取这两个文件，变化后的edits日志文件，同样需要读取

-》注册心跳、块的报告，需要向另一个namenode实时的汇报

3、日志文件安全性

-》cloudera公司提出：分布式日志存储方案

-》找到一个datanode节点目录，zookeeper有一个2n+1概念 n+1

-》写多份再读取，节点数目必须是奇数

-》还可以存储在zookeeper

4、通过代理来让客户端判断现在对外提供服务的是哪台namenode

5、两个namenode，但是必须在任何的情况下，只能有一个namenode对外提供服务

-》隔离

6、JournalNode日志节点

7、secondaryNameNode在HA架构下就不需要了

### HA配置

[HA机制配置](参考文件/HA机制配置.docx)

## Resourcemanager HA

[Resourcemanager HA配置](参考文件/Resourcemanager%20HA配置.docx)

## HDFS Federation

[HDFS Federation配置](参考文件/HDFS%20Federation配置.docx)

## 分布式拷贝工具DistCP

分布式拷贝

作用：用于大的内部或者外部的集群之间的数据拷贝

底层是：MapReduce，但是只有map没有reduce

用法：

distcp <srcurl>源端 <desturl>目标端

案例：

将第二台机器的文件拷贝到第一台的HDFS目录中去

$ hadoop distcp hdfs://MyDream2:8020/ wc.input hdfs://MyDream1:8020/ conf

# Zookeeper基础

## Zookeeper安装

[Zookeeper单机安装](参考文件/Zookeeper单机安装.docx)

[Zookeeper分布式安装](参考文件/Zookeeper分布式安装.docx)

# Hive基础

## MySQL安装

[MySQL安装](参考文件/MySQL安装.docx)

## Hive安装

[Hive安装](参考文件/Hive安装.docx)

## Hive使用

建表

create table student(id int, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

显示表信息

desc extended student ;

desc formatted student ;

加载数据

load data local inpath '/opt/datas/student.txt'into table db\_hive.student ;

显示函数

show functions ;

查看函数用法/查看函数用法并举例

desc function upper ;

desc function extended upper ;

查看变量

set

### 分区表

建表

create EXTERNAL table IF NOT EXISTS default.emp\_partition(

empno int,

ename string,

job string,

mgr int,

hiredate string,

sal double,

comm double,

deptno int

)

partitioned by (month string,day string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

加载数据

load data local inpath '/opt/datas/emp.txt' into table default.emp\_partition partition (month='201509',day='13') ;

查询

select \* from emp\_partition where month = '201509' and day = '13' ;

修复分区

msck repair table tablename;

### 加载数据

load data [local] inpath 'filepath' [overwrite] into table tablename [partition (partcol1=val1,...)];

\* 原始文件存储的位置

\* 本地 local

\* hdfs

\* 对表的数据是否覆盖

\* 覆盖 overwrite

\* 追加

\* 分区表加载，特殊性

partition (partcol1=val1,...) （以及分区，二级分区，三级分区………）

1）加载本地文件到hive表

load data local inpath '/opt/datas/emp.txt' into table default.emp ;

2）加载hdfs文件到hive中

load data inpath '/user/beifeng/hive/datas/emp.txt' overwrite into table default.emp ;

3）加载数据覆盖表中已有的数据

load data inpath '/user/beifeng/hive/datas/emp.txt' into table default.emp ;

4）创建表是通过insert加载

create table default.emp\_ci like emp ;

insert into table default.emp\_ci select \* from default.emp ;

5）创建表的时候通过location指定加载

### 导出数据

1. 导出到本地

insert overwrite local directory '/opt/datas/hive\_exp\_emp2'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' COLLECTION ITEMS TERMINATED BY '\n'

select \* from default.emp ;

hive -e "select \* from default.emp ;" > /opt/datas/exp\_res.txt

1. 导出到hdfs

insert overwrite directory '/user/beifeng/hive/hive\_exp\_emp'

select \* from default.emp ;

### having关键词

\* where 是针对单条记录进行筛选

\* having 是针对分组结果进行筛选

求每个部门的平均薪水大于2000的部门

select deptno, avg(sal) avg\_sal from emp group by deptno having avg\_sal > 2000;

### 数据导入导出

#### Export

导出，将Hive表中的数据，导出到外部

#### Import

导入，将外部数据导入Hive表中

EXPORT TABLE default.emp TO '/user/beifeng/hive/export/emp\_exp' ;

export\_target\_path：

指的是HDFS上路径

import table db\_hive.emp from '/user/beifeng/hive/export/emp\_exp';

### 四个by关键词

#### order by

对全局数据的一个排序，仅仅只有一个reduce

select \* from emp order by empno desc ;

#### sort by

对每一个reduce内部数据进行排序的，全局结果集来说不是排序

set mapreduce.job.reduces= 3;

select \* from emp sort by empno asc ;

insert overwrite local directory '/opt/datas/sortby-res' select \* from emp sort by empno asc ;

#### distribute by

分区partition

类似于MapReduce中分区partition,对数据进行分区，结合sort by进行使用

insert overwrite local directory '/opt/datas/distby-res' select \* from emp distribute by deptno sort by empno asc ;

注意事项：

distribute by 必须要在sort by 前面。

#### cluster by

当distribute by和sort by 字段相同时，可以使用cluster by ;

insert overwrite local directory '/opt/datas/cluster-res' select \* from emp cluster by empno ;

不同时可

select mid, money, name from store cluster by mid sort by money

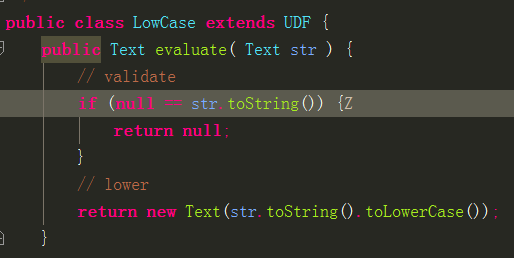
 注意被cluster by指定的列只能是降序，不能指定asc和desc。

### UDF函数

#### Creating Custom UDFs

[示例代码](WorkSpace/Hive-Projects/src/main/java/LowCase.java)

1)First, you need to create a new class that extends UDF, with one or more methods named evaluate.



2)Usage

Jar包在本地

add jar /opt/datas/hiveudf.jar ;

create temporary function my\_lower as "com.beifeng.senior.hive.udf.LowerUDF" ;

select ename, my\_lower(ename) lowername from emp limit 5 ;

jar包需上传

CREATE FUNCTION self\_lower AS 'com.beifeng.senior.hive.udf.LowerUDF' USING JAR 'hdfs://MyDream:8020/user/beifeng/hive/jars/hiveudf.jar';

select ename, self\_lower(ename) lowername from emp limit 5 ;

### Hive的三种启动方式

1， hive 命令行模式

进入hive安装目录，输入bin/hive的执行程序，或者输入 hive –service cli

用于Linux平台命令行查询，查询语句基本跟MySQL查询语句类似

2， hive web界面的启动方式

bin/hive –service hwi （& 表示后台运行）

用于通过浏览器来访问hive，浏览器访问地址是：127.0.0.1:9999/hwi

3， hive 远程服务 (端口号10000) 启动方式

bin/hive –service hiveserver2 &（&表示后台运行）

用java，Python等程序实现通过jdbc等驱动的访问hive就用这种启动方式了。

### 数据存储

\* 按行存储数据

\* 按列存储数据

create table page\_views(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE ;

load data local inpath '/opt/datas/page\_views.data' into table page\_views ;

dfs -du -h /user/hive/warehouse/page\_views/ ;

18.1 M /user/hive/warehouse/page\_views/page\_views.data

>>>>>>>orc

create table page\_views\_orc(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS orc ;

insert into table page\_views\_orc select \* from page\_views ;

dfs -du -h /user/hive/warehouse/page\_views\_orc/ ;

2.6 M /user/hive/warehouse/page\_views\_orc/000000\_0

>>>>>>>> parquet

create table page\_views\_parquet(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS PARQUET ;

insert into table page\_views\_parquet select \* from page\_views ;

dfs -du -h /user/hive/warehouse/page\_views\_parquet/ ;

13.1 M /user/hive/warehouse/page\_views\_parquet/000000\_0

select session\_id,count(\*) cnt from page\_views group by session\_id order by cnt desc limit 30 ;

select session\_id,count(\*) cnt from page\_views\_orc group by session\_id order by cnt desc limit 30 ;

-------

select session\_id from page\_views limit 30 ;

select session\_id from page\_views\_orc limit 30 ;

select session\_id from page\_views\_parquet limit 30 ;

========================================================

create table page\_views\_orc\_snappy(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS orc tblproperties ("orc.compress"="SNAPPY");

insert into table page\_views\_orc\_snappy select \* from page\_views ;

dfs -du -h /user/hive/warehouse/page\_views\_orc\_snappy/ ;

--------------

create table page\_views\_orc\_none(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS orc tblproperties ("orc.compress"="NONE");

insert into table page\_views\_orc\_none select \* from page\_views ;

dfs -du -h /user/hive/warehouse/page\_views\_orc\_none/ ;

--------------------

set parquet.compression=SNAPPY ;

create table page\_views\_parquet\_snappy(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

STORED AS parquet;

insert into table page\_views\_parquet\_snappy select \* from page\_views ;

dfs -du -h /user/hive/warehouse/page\_views\_parquet\_snappy/ ;

总结：

在实际的项目开发当中，hive表的数据

\* 存储格式

orcfile / qarquet

\* 数据压缩

Snappy

### SQL执行Python脚本

建表

CREATE TABLE u\_data\_new (

userid INT,

movieid INT,

rating INT,

weekday INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

添加脚本进入到hive

add FILE /opt/datas/ml-100k/weekday\_mapper.py;

执行脚本

INSERT OVERWRITE TABLE u\_data\_new

SELECT

TRANSFORM (userid, movieid, rating, unixtime) -- input from source table

USING 'python weekday\_mapper.py' -- script

AS (userid, movieid, rating, weekday) --output from python

FROM u\_data;

INSERT OVERWRITE TABLE u\_data\_new

SELECT

TRANSFORM (userid, movieid, rating, unixtime)

USING 'python weekday\_mapper.py'

AS (userid, movieid, rating, weekday)

FROM u\_data;

SELECT weekday, COUNT(\*) FROM u\_data\_new GROUP BY weekday;

SELECT weekday, COUNT(1) cnt FROM u\_data\_new GROUP BY weekday order by cnt desc;

脚本内容

import sys

import datetime

for line in sys.stdin:

line = line.strip()

userid, movieid, rating, unixtime = line.split('\t')

weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()

print '\t'.join([userid, movieid, rating, str(weekday)])

# sqoop基础

## sqoop安装

[sqoop安装](参考文件/sqoop安装.docx)

## sqoop使用

### 查看数据库

sqoop list-databases \

--connect jdbc:mysql://MyDream:3306 \

--username root \

--password 123456

### 导入数据

--添加测试mysql表

CREATE TABLE `my\_user` (

`id` tinyint(4) NOT NULL AUTO\_INCREMENT,

`account` varchar(255) DEFAULT NULL,

`passwd` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

);

INSERT INTO `my\_user` VALUES ('1', 'admin', 'admin');

INSERT INTO `my\_user` VALUES ('2', 'pu', '12345');

INSERT INTO `my\_user` VALUES ('3', 'system', 'system');

INSERT INTO `my\_user` VALUES ('4', 'zxh', 'zxh');

INSERT INTO `my\_user` VALUES ('5', 'test', 'test');

INSERT INTO `my\_user` VALUES ('6', 'pudong', 'pudong');

INSERT INTO `my\_user` VALUES ('7', 'qiqi', 'qiqi');

#### 导入数据

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user

#### 导入数据并设置导入目录和map个数

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_user \

--num-mappers 1

#### 导入数据并制定存储格式

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_user\_parquet \

--fields-terminated-by ',' \

--num-mappers 1 \

--as-parquetfile

#### 导入指定列的数据

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_user\_column \

--num-mappers 1 \

--columns id,account

#### 导入sql查询后的数据

\* 在实际的项目中，要处理的数据，需要进行初步清洗和过滤

\* 某些字段过滤

\* 条件

\* join

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--query 'select id, account from my\_user' \

--target-dir /user/beifeng/sqoop/imp\_my\_user\_query \

--num-mappers 1

>>>>>>>>>>>>>>出错了？？？？

15/09/01 07:10:39 ERROR tool.ImportTool: Encountered IOException running import job: java.io.IOException: Query [select id, account from my\_user] must contain '$CONDITIONS' in WHERE clause.

at org.apache.sqoop.manager.ConnManager.getColumnTypes(ConnManager.java:300)

at org.apache.sqoop.orm.ClassWriter.getColumnTypes(ClassWriter.java:1833)

at org.apache.sqoop.orm.ClassWriter.generate(ClassWriter.java:1645)

at org.apache.sqoop.tool.CodeGenTool.generateORM(CodeGenTool.java:96)

必须加上where $CONDITIONS

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--query 'select id, account from my\_user where $CONDITIONS' \

--target-dir /user/beifeng/sqoop/imp\_my\_user\_query \

--num-mappers 1

#### 导入数据并压缩和删除已存在目录和指定字段分隔符

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_sannpy \

--delete-target-dir \

--num-mappers 1 \

--compress \

--compression-codec org.apache.hadoop.io.compress.SnappyCodec \

--fields-terminated-by '\t'

#### 增量数据的导入

有一个唯一标识符，通常这个表都有一个字段，类似于插入时间createtime

\* query //通过sql查询导入

where createtime => 20150924000000000 and createtime < 20150925000000000

\* sqoop //通过sqoop指定字段并指定该字段上次导入的最后一个记录

Incremental import arguments:

--check-column <column> Source column to check for incremental

change

--incremental <import-type> Define an incremental import of type

'append' or 'lastmodified'

--last-value <value> Last imported value in the incremental

check column

导入id=4以后的记录

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_incr \

--num-mappers 1 \

--incremental append \

--check-column id \

--last-value 4

#### 直接导入，不走MapReduce

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--target-dir /user/beifeng/sqoop/imp\_my\_incr \

--num-mappers 1 \

--delete-target-dir \

--direct

#### 导入数据到hive表

Hive数据存储在hdfs上

use default ;

drop table if exists user\_hive ;

create table user\_hive(

id int,

account string,

password string

)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

sqoop import \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--fields-terminated-by '\t' \

--delete-target-dir \

--num-mappers 1 \

--hive-import \

--hive-database default \

--hive-table user\_hive

### 导出数据

#### 导出数据到MySQL表

touch /opt/datas/user.txt

vi /opt/datas/user.txt

12,beifeng,beifeng

13xuanyun,xuanyu

bin/hdfs dfs -mkdir -p /user/beifeng/sqoop/exp/user/

bin/hdfs dfs -put /opt/datas/user.txt /user/beifeng/sqoop/exp/user/

sqoop export \

--connect jdbc:mysql://MyDream:3306/test \

--username root \

--password 123456 \

--table my\_user \

--export-dir /user/beifeng/sqoop/exp/user/ \

--num-mappers 1

--input-fields-terminated-by '\t'

--input-fields-terminated-by '\t'

### Shell导入导出数据

shell scripts

## step 1

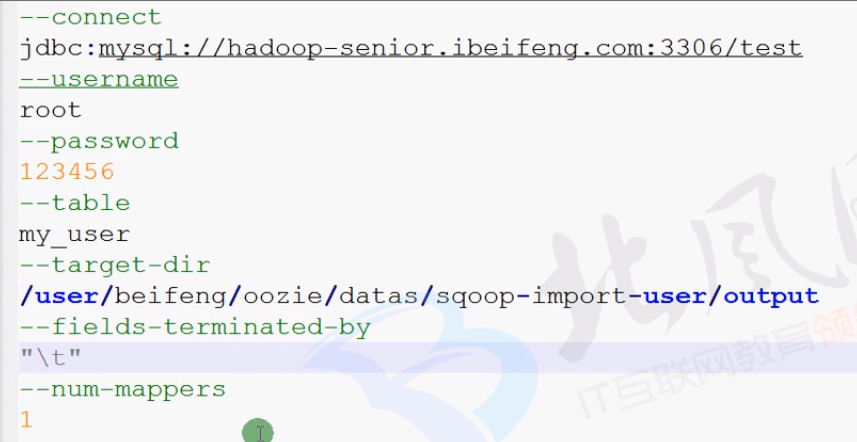
load data ...

## step 2

bin/hive -f xxxx

## step 3

编写脚本，将之前的命令写进脚本



可以不写某些参数，执行脚本的时候再追加：

sqoop --options-file /opt/datas/sqoop-import-hdfs.txt --num-mappers 1

# flume基础

## flume安装

[flume安装](参考文件/flume安装.docx)

## flume使用

上传[Telnet工具](参考文件/telnet-rpms)到虚拟机

### 安装rpm

mv netcat-1.10-891.2.x86\_64.rpm ../

rpm -ivh ./\*.rpm

### 启动xinetd服务

/etc/rc.d/init.d/xinetd restart

### 启动flume服务

Flume命令

[root@MyDream ~]# flume-ng

Usage: /home/hadoop/apache-flume-1.5.0-cdh5.3.6-bin/bin/flume-ng <command> [options]...

commands:

agent run a Flume agent

global options:

--conf,-c <conf> use configs in <conf> directory

-Dproperty=value sets a Java system property value

agent options:

--name,-n <name> the name of this agent (required)

--conf-file,-f <file> specify a config file (required if -z missing)

上传[配置文件](参考文件/flume脚本/a1.conf)到flume安装目录下的conf目录

启动服务：

flume-ng agent --conf conf --name agent-test --conf-file test.conf

或者

flume-ng agent -c conf -n agent-test -f test.conf

解释：

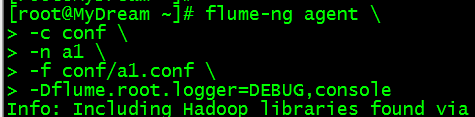
flume-ng agent \

-c conf \ 启用flume安装目录下的conf文件夹

-n a1 \ agent的名字为a1

-f conf/a1.conf \ 指定配置文件为conf目录下的a1.conf

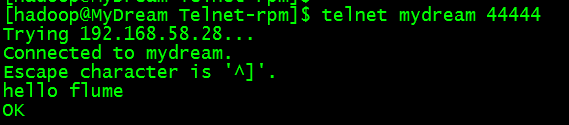
-Dflume.root.logger=DEBUG,console 显示信息到控制台



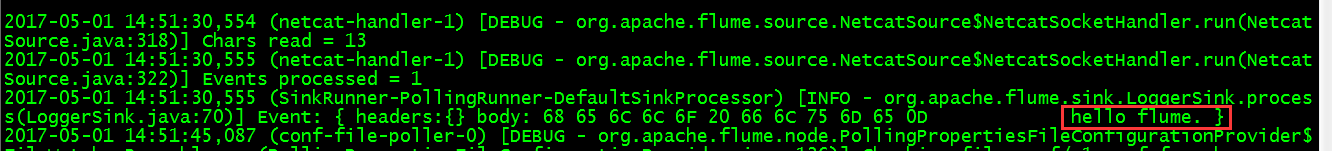
### 连接a1.conf配置文件指定的端口

telnet mydream 44444

发送信息



Flume收到信息



### 收集hive日志

编辑[配置文件](参考文件/flume脚本/flume-tail.conf)

新建上传日志的hdfs目录

hdfs dfs -mkdir -p /user/hadoop/flume/hive-log

启动脚本：

flume-ng agent \

-c conf \

-n a2 \

-f conf/flume-tail.conf \

-Dflume.root.logger=DEBUG,console

同步日志到hdfs成功



NOTE：若配置文件里面配置的是HA访问hdfs，需将hadoop的core-site.xml和hdfs-site.xml两个配置文件放到flume的conf文件下下面去

### 动态创建hdfs文件夹

按日期创建文件夹

hdfs://hadoop-senior.ibeifeng.com:8020/user/beifeng/flume/applogs/%Y%m%d

需设置使用服务器时间戳

hdfs.useLocalTimeStamp = true

编辑[配置文件](参考文件/flume脚本/flume-app.conf)

新建上传日志的hdfs目录

hdfs dfs -mkdir -p /user/hadoop/flume/sp-log

启动脚本：

flume-ng agent \

-c conf \

-n a2 \

-f conf/flume-app.conf \

-Dflume.root.logger=DEBUG,console

同步日志到hdfs成功

# Oozie基础

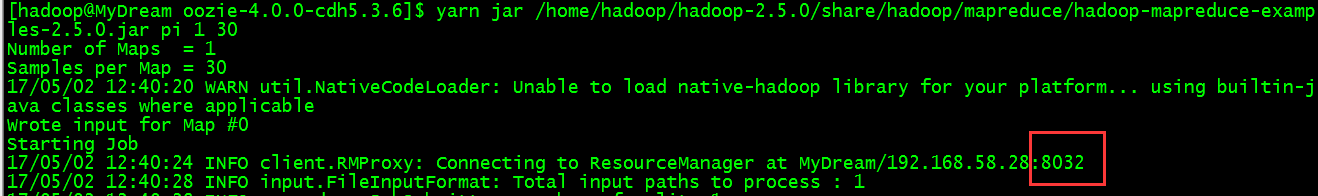
## Oozie安装

[Oozie安装](参考文件/Oozie安装.docx)

## 运行job

测试jobtrack的端口

yarn jar /home/hadoop/hadoop-2.5.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.0.jar pi 1 30



修改properties文件

修改/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/map-reduce/job.properties

nameNode=hdfs://MyDream:8020

jobTracker=MyDream:8032

运行job

oozie job -oozie http://localhost:11000/oozie -config examples/apps/map-reduce/job.properties –run

## 编写MapReduce action

新建文件夹

mkdir myMapReduce

复制example到文件夹下

cp -r /home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/map-reduce/\* ./

编辑workflow.xml

（本配置过程相当于在编写MapReduce程序时的job设置）

替换为新api的参数

mapred.job.queue.name替换为mapreduce.job.queuename

mapred.mapper.class替换为mapreduce.job.map.class

mapred.reducer.class替换为mapreduce.job.reduce.class

mapred.input.dir替换为mapreduce.input.fileinputformat.inputdir

mapred.output.dir替换为mapreduce.output.fileoutputformat.outputdir

删除

<property>

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

<value>1</value>

</property>

添加

<property>

<name>mapreduce.map.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.map.output.value.class</name>

<value>org.apache.hadoop.io.IntWritable</value>

</property>

<property>

<name>mapreduce.job.output.key.class</name>

<value>org.apache.hadoop.io.Text</value>

</property>

<property>

<name>mapreduce.job.output.value.class</name>

<value>org.apache.hadoop.io.IntWritable</value>

</property>

开头添加，开启新api

<property>

<name>mapred.maper.new-api</name>

<value>true</value>

</property>

<property>

<name>mapred.reducer.new-api</name>

<value>true</value>

</property>

自定义名字（第2 3个名字不能超过20个字符）

<workflow-app xmlns="uri:oozie:workflow:0.2" name="myWordcount">

<start to="wc-node"/>

<action name="wc-node">

Job.properties自定义应用路径

examplesRoot=examples

oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/map-reduce/workflow.xml

可改为：

OozieAppPath=user/hadoop/wordcount/app 定义变量

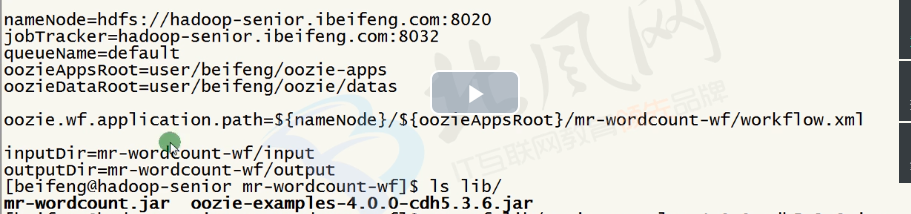
OozieDatapath= user/hadoop/wordcount/data 定义变量

oozie.wf.application.path=${nameNode}/${OozieAppPath}/myWordcount/workflow.xml

指定配置文件路径，需上传myWordcount目录下的文件到hdfs的该路径下

拷贝依赖的jar包（要运行的编写的MapReducejar包）到myMapReduce的lib目录下

更改workflow.xml的maper类和reducer类型自己编写的maper类 reducer类

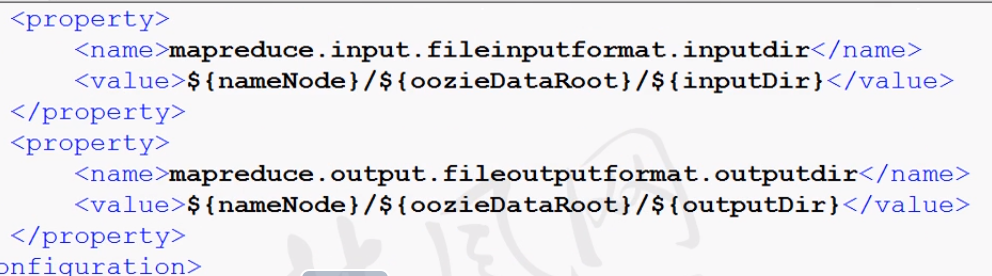


将自定义的MapReduce输出结果的目录配置进

<prepare>

<delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/${outputDir}"/>

</prepare>



运行

oozie job -oozie http://localhost:11000/oozie -config wordcount/app /myWordcount /job.properties –run



## 编写hive action

参考/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/hive

其中hive新旧api都接受，所以不用更改api参数

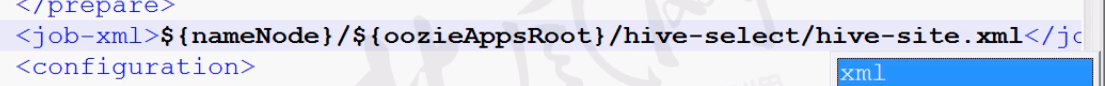
将hive的hive-site.xml文件放到复制/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/hive出来后的目录下，和xml properties 脚本放在同一个文件夹，并在该文件夹新建lib目录，将MySQL jdbc jar包放到该目录。

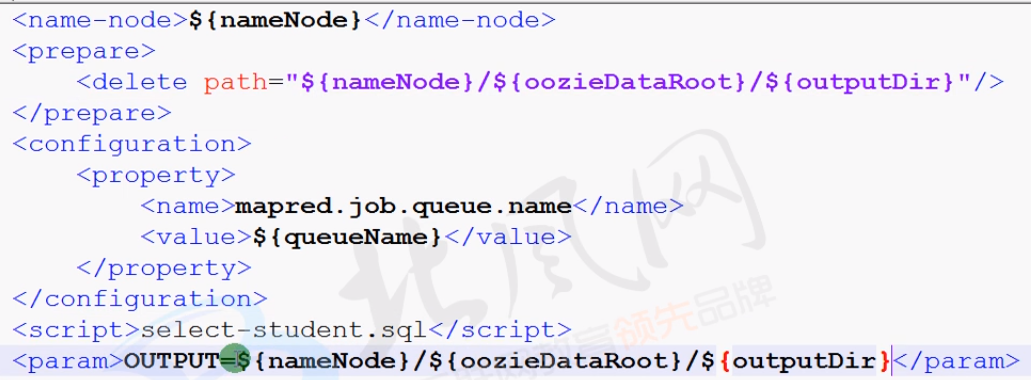
编辑job.properties



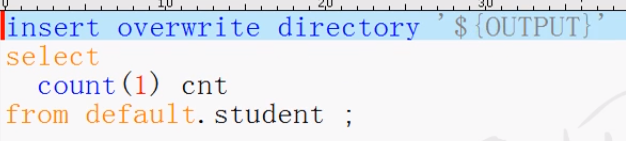
编辑workflow.xml

配置hive-site.xml文件路径





编辑脚本



其他同MapReduce action，参考[9.3](#_编写MapReduce__action)

## 编写sqoop action

Sqoop使用的是新的api，需要向[9.3](#_编写MapReduce__action)一样替换为新的api参数。

Command为sqoop导入导出数据命令，其中字段分隔符要用双引号引起来

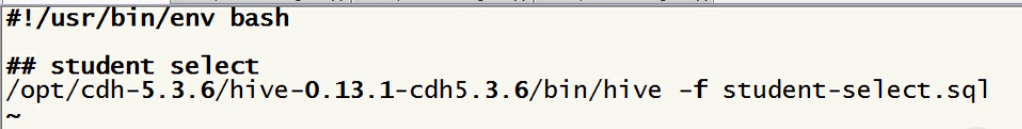
新建lib目录，复制MySQL jdbc jar包进去



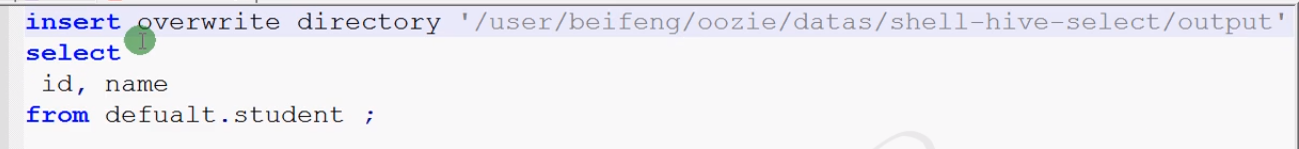
## 编写shell action

参考/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/shell

编辑student-select.sh



编辑student-select.sql



编辑job.properties



编辑workflow.xml



## Oozie修改时区

### 修改oozie-site.xml文件

添加

<property>

<name>oozie.processing.timezone</name>

<value>GMT+0800</value>

</property>

禁用最小时间频率

<property>

<name>oozie.service.coord.check.maximum.frequency</name>

<value>false</value>

</property>

### 修改web页面显示时区

修改/home/hadoop/oozie-4.0.0-cdh5.3.6/oozie-server/webapps/oozie/oozie-console.js

179行

function getTimeZone() {

Ext.state.Manager.setProvider(new Ext.state.CookieProvider());

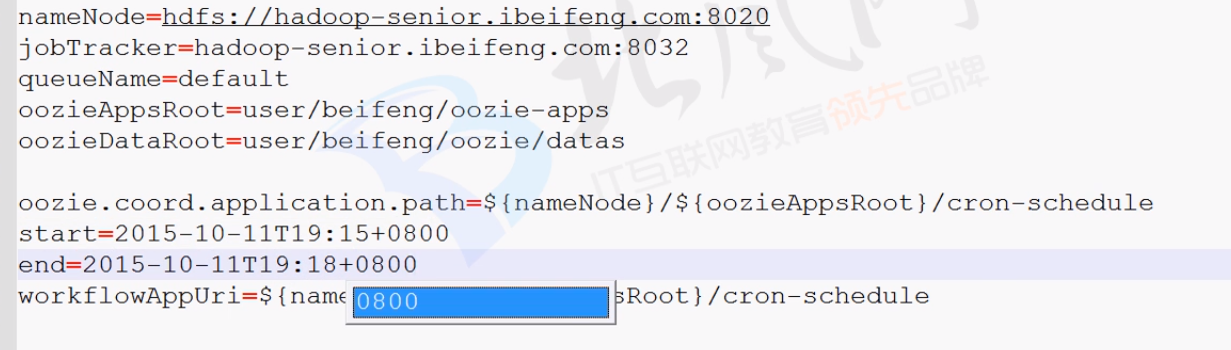
return Ext.state.Manager.get("TimezoneId","GMT+0800");

}

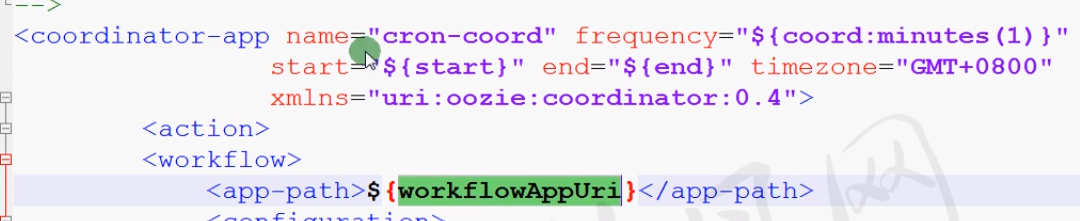
## 编写定时任务

参考/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/cron-schedule

编辑job.properties



编辑coordinator.xml

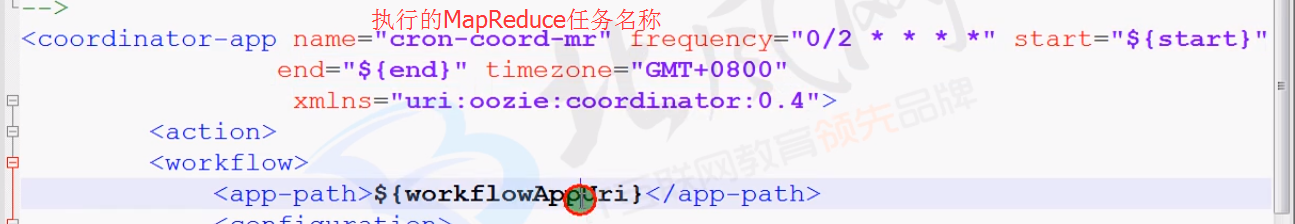


## 测试cron任务

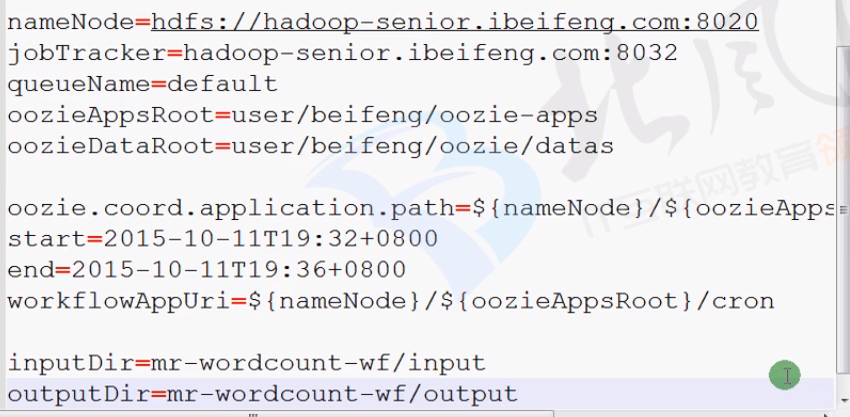
参考/home/hadoop/oozie-4.0.0-cdh5.3.6/examples/apps/cron

将之前的任意MapReduce action的workflow.xml和lib复制过来

编辑coordinator.xml



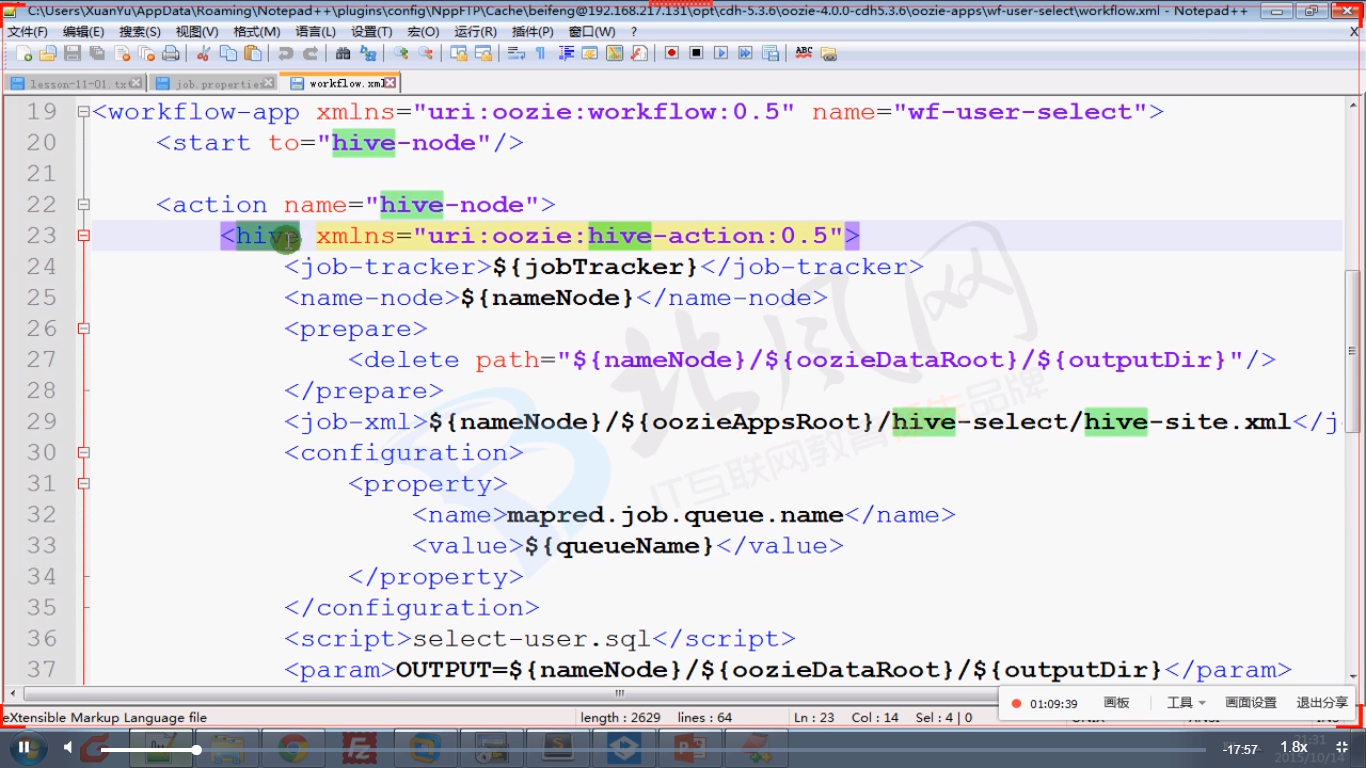
编辑job.properties

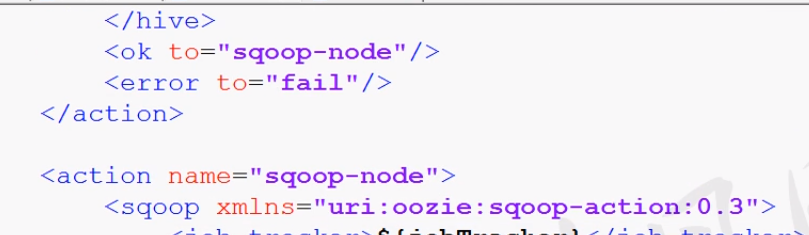


## 多个action

将两个或多个

配置workflow.xml action跳转

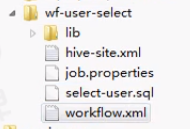




Job.properties



目录结构



任务的调度执行周期在coordinator.xml配置。

# Hue基础

## Hue安装

[Hue安装](参考文件/Hue安装.docx)

# Hbase基础

HBase新版本中，有了类似于RDBMS中DataBase的概念

命令空间

用户自定义的表，默认情况下命名空间

default

系统自带的元数据表的命名空间

hbase

## Hbase安装

[Hbase安装](参考文件/Hbase安装.docx)

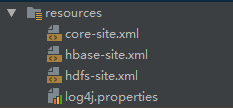
## Hbase使用

### Java api操作hbase表

1. 导入jar包

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <hadoop.version>2.5.0</hadoop.version>  
 <hive.version>0.13.1</hive.version>  
 <hbase.version>0.98.6-hadoop2</hbase.version>  
</properties>  
<repositories>  
 <repository>  
 <id>nexus-aliyun</id>  
 <name>Nexus aliyun</name>  
 <url>http://maven.aliyun.com/nexus/content/groups/public</url>  
 </repository>  
</repositories>  
<dependencies>  
 <dependency>  
 <groupId>org.apache.hbase</groupId>  
 <artifactId>hbase-server</artifactId>  
 <version>${hbase.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hbase</groupId>  
 <artifactId>hbase-client</artifactId>  
 <version>${hbase.version}</version>  
 </dependency>  
</dependencies>

1. 导入hadoop和hbase配置文件



1. 编写[Java类](WorkSpace/HbaseProjects/src/main/hbaseOperations/HBaseOperation.java)

### 运行hbase自带MapReduce程序

1. 设置hadoop类路径

export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`

1. 命令提示

$HADOOP\_HOME/bin/yarn jar $HBASE\_HOME/lib/hbase-server-0.98.6-hadoop2.jar

CellCounter: Count cells in HBase table

completebulkload: Complete a bulk data load.

copytable: Export a table from local cluster to peer cluster

export: Write table data to HDFS.

import: Import data written by Export.

importtsv: Import data in TSV format.

rowcounter: Count rows in HBase table

verifyrep: Compare the data from tables in two different clusters. WARNING: It doesn't work for incrementColumnValues'd cells since the timestamp is changed after being appended to the log.

1. 运行一个命令

$HADOOP\_HOME/bin/yarn jar $HBASE\_HOME/lib/hbase-server-0.98.6-hadoop2.jar rowcounter 'user'

#### TSV和CSV区别

TSV

Tab分割

>> student.tsv

1001 zhangsan 26 shanghai

CSV

逗号分割

>> student.csv

1001,zhangsan,26,shanghai

### 编写hbase的MapReduce

从一张hbase表导出数据到另一张hbase表

[User2BasicMapReduce.java](WorkSpace/HbaseProjects/src/main/hbaseOperations/User2BasicMapReduce.java)

1. Mapper类继承TableMapper类
2. 将rowkey对应的value放进一个put里面
3. Reducer类继承TableReducer类
4. 写进context
5. 打包运行jar

export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`

$HADOOP\_HOME/bin/yarn jar $HADOOP\_HOME/jars/hbase-mr-user2basic.jar

### Importtsv导入hdfs文件到hbase表

1. 准备数据

student.tsv

10001 zhangsan 35 male beijing 0109876543

10002 lisi 32 male shanghia 0109876563

10003 zhaoliu 35 female hangzhou 01098346543

10004 qianqi 35 male shenzhen 01098732543

1. 上传数据

hdfs dfs –put student.tsv /user/hadoop/hbase/importtsv

1. 创建hbase表

create ‘student’ , ‘info’

1. 设置hadoop类路径

export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`:${HBASE\_HOME}/conf

1. 导入数据

${HADOOP\_HOME}/bin/yarn jar \

${HBASE\_HOME}/lib/hbase-server-0.98.6-hadoop2.jar importtsv \

-Dimporttsv.columns=HBASE\_ROW\_KEY,\

info:name,info:age,info:sex,info:address,info:phone \

student \

hdfs://MyDream:8020/user/hadoop/hbase/importtsv

### bulk批量导入数据到hbase表

不占内存

1. 设置hadoop类路径

export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`:${HBASE\_HOME}/conf

1. 将要导入的文件转换成Hfile格式

${HADOOP\_HOME}/bin/yarn jar \

${HBASE\_HOME}/lib/hbase-server-0.98.6-hadoop2.jar importtsv \

-Dimporttsv.columns=HBASE\_ROW\_KEY,\

info:name,info:age,info:sex,info:address,info:phone \

-Dimporttsv.bulk.output=hdfs://hadoop-senior.ibeifeng.com:8020/user/hadoop/hbase/hfileoutput \

student2 \

hdfs://MyDream:8020/user/hadoop/hbase/importtsv

1. 导入Hfile格式数据到hbase表

${HADOOP\_HOME}/bin/yarn jar \

${HBASE\_HOME}/lib/hbase-server-0.98.6-hadoop2.jar \

completebulkload \

hdfs://MyDream:8020/user/hadoop/hbase/hfileoutput \

student

### 表预分区

创建表后一般导入数据，当导入数据过多时region分割成两个region时负担由一个regionserver承担，容易产生问题，预分区将建立多个region由多个regionserver管理数据，负载平衡。

按时间创建分区

1. 命令行指定分区

create 'bflogs', 'info', SPLITS => ['20151001000000000', '20151011000000000', '20151021000000000']

插入数据时按rowkey插入对应region。

1. 文本文件指定分区

create 'bflogs2', 'info', SPLITS\_FILE => '/opt/datas/bflogs-split.txt'

bflogs-split.txt

20151001000000000

20151011000000000

20151021000000000

### Hbase与hive集成

1. 将hbase jar包导入hive

ln -s $HBASE\_HOME/lib/hbase-server-0.98.6-hadoop2.jar $HIVE\_HOME/hbase-server-0.98.6-hadoop2.jar

ln -s $HBASE\_HOME/lib/hbase-client-0.98.6-hadoop2.jar $HIVE\_HOME/hbase-client-0.98.6-hadoop2.jar

ln -s $HBASE\_HOME/lib/hbase-protocol-0.98.6-hadoop2.jar $HIVE\_HOME/hbase-protocol-0.98.6-hadoop2.jar

ln -s $HBASE\_HOME/lib/hbase-it-0.98.6-hadoop2.jar $HIVE\_HOME/hbase-it-0.98.6-hadoop2.jar

ln -s $HBASE\_HOME/lib/htrace-core-2.04.jar $HIVE\_HOME/htrace-core-2.04.jar

ln -s $HBASE\_HOME/lib/hbase-hadoop2-compat-0.98.6-hadoop2.jar $HIVE\_HOME/lib/hbase-hadoop2-compat-0.98.6-hadoop2.jar

ln -s $HBASE\_HOME/lib/hbase-hadoop-compat-0.98.6-hadoop2.jar $HIVE\_HOME/lib/hbase-hadoop-compat-0.98.6-hadoop2.jar

ln -s /opt/modules/hbase-0.98.6-hadoop2/lib/high-scale-lib-1.1.1.jar /opt/modules/hive-0.13.1/lib/high-scale-lib-1.1.1.jar

1. 配置hive的hive-site.xml文件

<property>

<name>hbase.zookeeper.quorum</name>

<value>MyDream</value>

</property>

两种方式：

管理表

创建hive表的时候，指定数据存储在hbase表中。

CREATE TABLE hbase\_table\_1(key int, value string)

STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")

TBLPROPERTIES ("hbase.table.name" = "xyz");

>> 外部表

现在已经存在一个HBase表，需要对表中数据进行分析。

CREATE EXTERNAL TABLE hbase\_user(id int, name string,age int)

STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:name,info:age")

TBLPROPERTIES ("hbase.table.name" = "user");

### Hbase与hue集成

1. 配置hue的hue.ini文件

hbase\_clusters=(Cluster|MyDream:9090)

hbase\_conf\_dir=/home/hadoop/hbase-0.98.6-hadoop2/conf

1. 启动thrift服务

hbase-daemon.sh start thrift

1. 启动hadoop和hbase master

start-all.sh

start-hbase.sh

hbase-daemon.sh start master

1. 启动hue

Supervisor

# CDH安装

## CM及CDH5.7安装文档

[CM及CDH5.7安装文档](参考文件/CM及CDH5.7安装文档.docx)

[参考文件](测试数据/课程十一、企业大数据平台高级应用)

# Storm基础

## Storm安装

[Storm伪分布式安装](参考文件/Storm伪分布式安装.docx)