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



目 录

[葵花宝典 0](#_Toc480320954)

[1 Linux基础 2](#_Toc480320955)

[1.1 Linux分区操作 2](#_Toc480320956)

[1.2 Linux命令 2](#_Toc480320957)

[1.2.1 grep命令 2](#_Toc480320958)

[1.2.2 Sed命令 2](#_Toc480320959)

[1.2.3 awk命令 3](#_Toc480320960)

[1.3 Shell编程 3](#_Toc480320961)

[1.3.1 变量 3](#_Toc480320962)

[1.3.2 Shell算数运算 4](#_Toc480320963)

[1.3.3 Shell内置测试判断 4](#_Toc480320964)

[1.3.4 控制语句 7](#_Toc480320965)

[1.3.5 函数使用 10](#_Toc480320966)

[1.3.6 Crontab计划任务 11](#_Toc480320967)

[2 Jhg 12](#_Toc480320968)

[2.1 Kjh 12](#_Toc480320969)

# 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

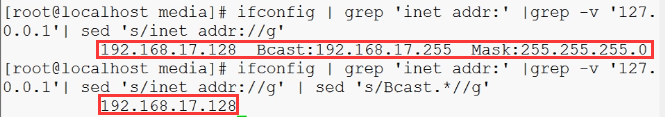
cat /etc/passwd | sed '匹配条件/执行的动作'

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

## Shell编程

### 变量

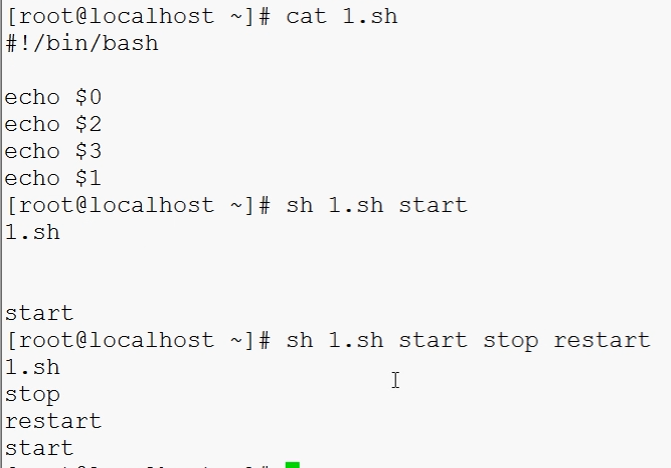
#### 位置变量

--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

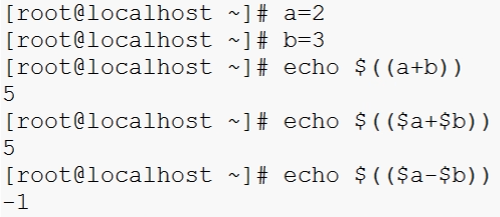
-- + - \* / %

--$((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

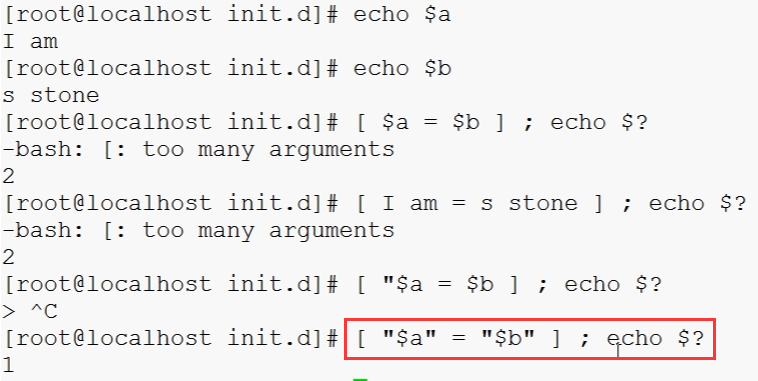
#### 字符串测试

= 等于则为真

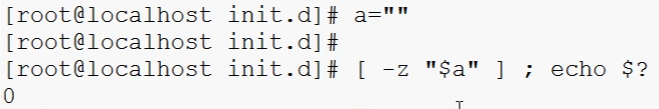
!= 不相等则为真

-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：

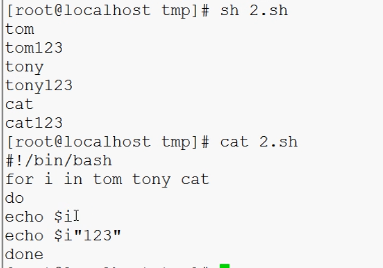
for 变量 in 值1 值2 … 值N

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

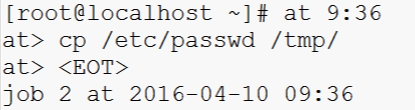
用法：at [时间]

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

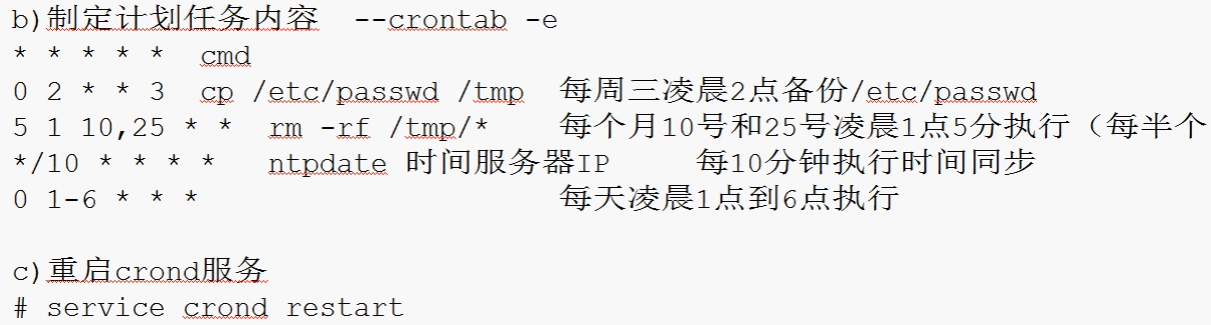
②任务格式



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)

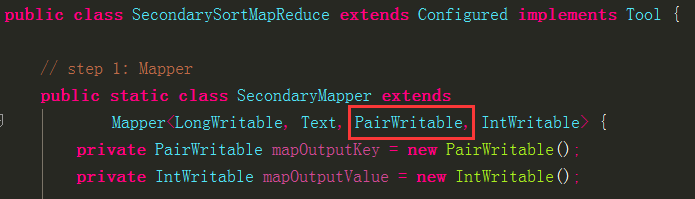
分区时不同的key可能分到不同的分区，分区后进行排序再分组，而分组时相同的key一定会分到相同的组，并且会形成一个keyvalues对，values为该key的所有value的集合。

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

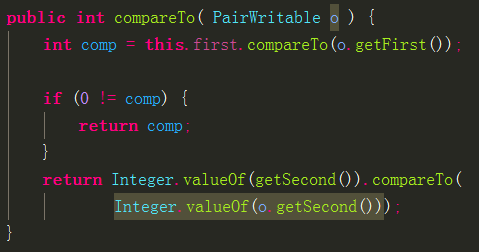
### 组合key

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

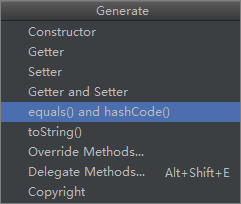


[自定义数据类型生成步骤：](WorkSpace/bigdata-hdfs/src/main/java/com/ibeifeng/mapreduce/PairWritable.java)

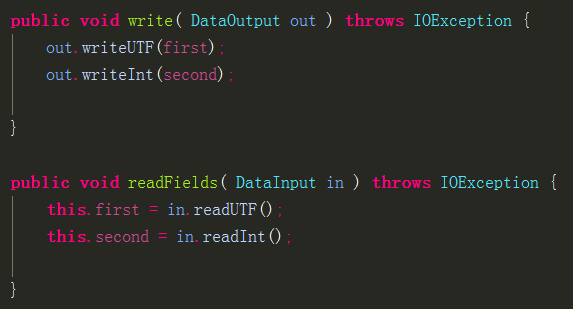
1. 实现WritableComparable接口，添加未实现方法；
2. 添加类成员变量并生成get和set方法；
3. 生成无参和有参构造方法；
4. 实现compareTo方法，用来定义排序规则；



1. 生成hashCode和equals方法；



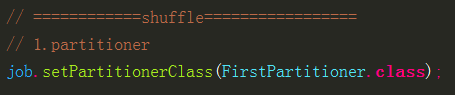
1. 生成tostring方法；
2. 实现写和度方法（两个方法参数顺序要一致）



### 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)

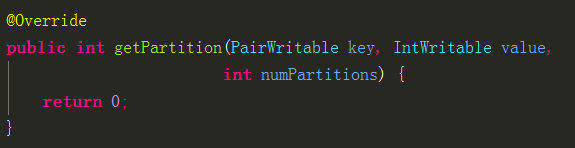
1. Job设置分区类



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

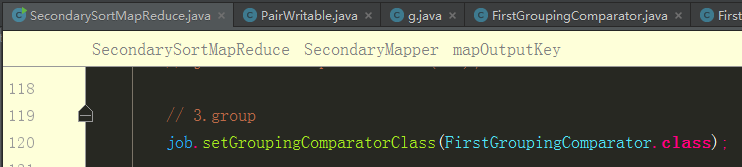
hashPartitioner分区是MR框架默认的分区计算方法



### 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)

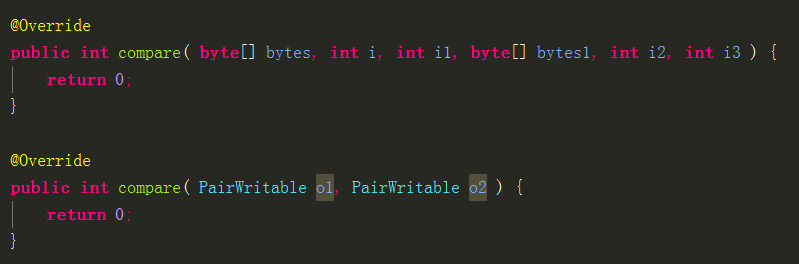
1. Job设置分组类



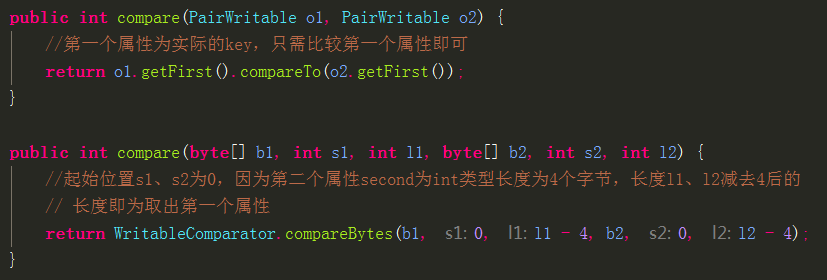
1. 新建分组类，实现RawComparator接口，泛型中的类为要分组排序的数据类



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

# 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)