**第十九章 shell脚本的基础**

**本节所讲内容：**

**19.1 shell 基本语法**

**19.2 SHELL变量及运用**

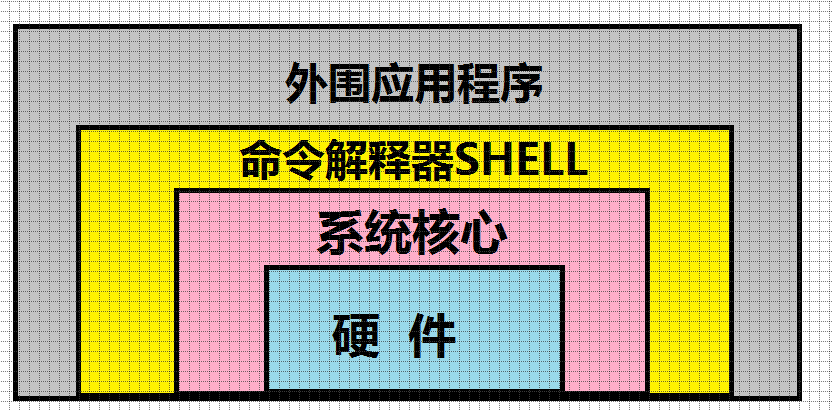
**19.3 数学运算**

**19.4 实战-升级系统中的java版本到1.8版本-为后期安装Hadoop集群做准备**

# 19.1 shell 基本语法

## 19.1.1 什么是shell?

**Shell是一个命令解释器，它在操作系统的最外层，负责直接与用户进行对话，把用户的输入解释给操作系统，并处理各种各样的操作系统的输出结果，输出到屏幕反馈给用户。这种对话方式可是交互也可以是非交互式的**

****

**我们所输入的命令计算机是不识别的，这时就需要一种程序来帮助我们进行翻译，变成计算机能识别的二进制程序，同时又把计算机生成的结果返回给我们。**

## 19.1.2 编程语言分类

**编程语言主要用：低级语言和高级语言**

**1）低级语言：**

**机器语言：二进制语言**

**汇编语言：符号语言，使用助记符来代替操作码，也就是用符号代替机器语言的二进制码**

**它们都是面向机器的语言**

**2）高级语言：**

**它是比较接近自然语言或者说人类语言的一种编程，用人们能够容易理解的方式进行编写程序，**

**静态语言：编译型语言 如：c 、 c++ 、 java，**

**动态语言：解释型语言 如： php 、 shell 、 python 、 perl**

**gcc编译器：（解释器） 将人类理解的语言翻译成机器理解的语言**

## 19.1.3 什么是SHELL脚本？

**shell脚本：就是说我们把原来linux命令或语句放在一个文件中，然后通过这个程序文件去执行时，我们就说这个程序为shell脚本或shell程序；我们可以在脚本中输入一系统的命令以及相关的语法语句组合，比如变量，流程控制语句等，把他们有机结合起来就形成了一个功能强大的shell脚本**

**先手写一个脚本体验一下：**

**[root@xuegod63 ~]# vim test.sh #写入以下内容**

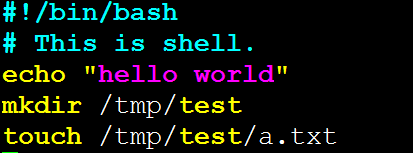
**#!/bin/bash**

**# This is shell.**

**echo "hello world"**

**mkdir /tmp/test**

**touch /tmp/test/a.txt**

****

**注释：**

**1、!/bin/bash 作用：告诉脚本使用的是哪种命令解释器。如不指shell，以当前shell作为执行的shell。**

**2、在shell中以#表示开头，整个行就被当作一个注释。执行时被忽略。**

**3、shell程序一般以.sh结尾**

**[root@xuegod63 ~]# chmod +x test.sh**

**[root@xuegod63 ~]# ./test.sh #执行**

**hello world**

**创建shell程序的步骤：**

** 第一步：创建一个包含命令和控制结构的文件。**

** 第二步：修改这个文件的权限使它可以执行。 使用chmod +x test.sh**

** 第三步：检测语法错误**

** 第四步：执行 ./example**

**shell脚本的执行通常有以下几种方式**

**1、/root/test.sh 或者 ./test.sh (当前路径下执行脚本的话要有执行权限chmod +x test.sh)**

**2、bash test.sh 或 sh test.sh （这种方式可以不对脚本文件添加执行权限）**

**3、source test.sh (可以没有执行权限)**

**4、sh < test.sh 或者 cat test.sh |sh(bash)**

# 19.2 SHELL变量及运用

## 19.2.1 shell变量

**变量是shell 传递数据的一种方法。变量是用来代表每个值的符号名。我们可以把变量当成一个容器，通过变量，可以在内存中存储数据。也可以在脚本执行中进行修改和访问存储的数据**

**变量的设置规则：**

1. **变量名称通常是大写字母，它可以由数字、字母（大小写）和下划线\_组成。变量名区分大小写；但是大家要注意变量名称不能以数字开头**
2. **等号 = 用于为变量分配值，在使用过程中等号两边不能有空格**
3. **变量存储的数据类型是整数值和字符串值**
4. **在对变量赋于字符串值时，建议大家用引号将其括起来。因为如果字符串中存在空格隔符号。需要使用单引号或双引号**
5. **要对变量进行调用，可以在变量名称前加美元符号$**
6. **如果需要增加变量的值，那么可以进行变量值的叠加。不过变量需要用双引号包含“$变量名”或用${变量名}包含**

**变量的分类**

**按照变量的作用可以分成4类：**

**1、用户自定义变量**

**2、环境变量：这种变量中主要保存的是和系统操作环境相关的数据。**

**3位置参数变量：这种变量主要是用来向脚本当中传递参数或数据的，变量名不能自定义，变量作用是固定的。**

**4、预定义变量：是Bash中已经定义好的变量，变量名不能自定义，变量作用也是固定的。**

**按照变量作用域可以分成2类：全局变量和局部变量。**

**局部变量是shell 程序内部定义的，其使用范围仅限于定义它的程序，对其它程序不可见。包括：用户自定义变量、位置变量和预定义变量。**

** 全局变量是环境变量，其值不随shell 脚本的执行结束而消失。**

## 19.2.2 用户定义变量

**变量名命名规则：由字母或下划线打头，不允许数字开头,后面由字母、数字或下划线组成，并且大小写字母意义不同。在使用变量时，在变量名前加$**

**例1：给变量VAR1赋值**

**[root@xuegod63 ~]# VAR1=123**

**[root@xuegod63 ~]# echo $VAR1**

**123**

**variable [ˈveəriəbl] 变量**

**例2：错误的赋值方式,不允许数字开头，等号两边不能有空格**

**[root@xuegod63 ~]# VAR2 =456**

**bash: VAR2: 未找到命令...**

**[root@xuegod63 ~]# VAR2= 456**

**bash: 456: 未找到命令...**

**[root@xuegod63 ~]# VAR2 = 456**

**bash: VAR2: 未找到命令...**

**[root@xuegod63 ~]# 3VAR2 = 456**

**bash: 3VAR2: 未找到命令...**

**例3：变量值的叠加，使用${}**

**$name是${name}的简化版本，但是在某些情况下，还必须使用花括号引起的方式来消除歧义并避免意外的结果**

**[root@xuegod63 ~]# VAR4=mysql**

**[root@xuegod63 ~]# echo $VAR4**

**mysql**

**[root@xuegod63 ~]# echo $VAR4-db.log**

**mysql-db.log**

**[root@xuegod63 ~]# echo $VAR4.db.log**

**mysql.db.log**

**[root@xuegod63 ~]# echo $VAR4db.log #发现输出的结果不是我们想要的，怎么办？**

**.log**

**[root@xuegod63 ~]# echo ${VAR4}db.log**

**mysqldb.log**

## 19.2.3 命令的替换,使用$()或反引号

**例1：在命令就调用date命令**

**扩展：date命令是显示或设置系统时间与日期。**

**-s<字符串>：根据字符串来设置日期与时间。字符串前后必须加上双引号；**

**<+时间日期格式>：指定显示时，使用特定的日期时间格式。**

**例：格式化输出：**

**[root@xuegod63 ~]# date +"%Y-%m-%d" #今天时间，一般备份数据需要用这个**

**2018-05-25**

**[root@xuegod63 ~]# date +"%Y-%m" #只显示年月**

**2018-05**

**[root@xuegod63 ~]# date +"%Y-%m-%d %H:%M.%S" #日期加时间**

**2018-05-25 17:51.36**

**[root@xuegod63 ~]# date +"%Y/%m/%d %H/%M/%S" #使用/做分隔符**

**2018-05-25 17-51-43**

**[root@xuegod63 ~]# date +"%Y-%m-%d-%H-%M-%S" #使用-做分隔符，一般备份数据需要用这个**

**注： %y 年份只显示2位， %Y年份显示4位**

**date命令加减操作：**

**date +%Y%m%d #显示当天年月日**

**date -d "+1 day" +%Y%m%d #显示明天的日期**

**date -d "-1 day" +%Y%m%d #显示昨天的日期**

**date -d "-1 month" +%Y%m%d #显示上一月的日期**

**date -d "+1 month" +%Y%m%d #显示下一月的日期**

**date -d "-1 year" +%Y%m%d #显示前一年的日期**

**date -d "+1 year" +%Y%m%d #显示下一年的日期**

**设定时间：**

**date -s 20180523 #设置成20120523，这样会把具体时间设置成空00:00:00**

**date -s 01:01:01 #设置具体时间，不会对日期做更改**

**date -s "2018-05-23 01:01:01" #这样可以设置全部时间**

**例1： 在命令中调用date命令输出值**

**[root@xuegod63 ~]# echo `date`**

**2018年 05月 25日 星期五 17:41:29 CST**

**[root@xuegod63 ~]# echo $(date)**

**2018年 05月 25日 星期五 17:41:42 CST**

**[root@xuegod63 ~]# echo `date +"%Y-%m-%d"`**

**2012-05-23**

## 19.2.4 命令的嵌套使用，使用$( $( ))

**[root@xuegod63 ~]# find /root/ -name \*.txt**

**[root@xuegod63 ~]# VAR6=$(tar zcvf root.tar.gz $(find /root/ -name \*.txt))**

**tar: 从成员名中删除开头的“/”**

**[root@xuegod63 ~]# echo $VAR6 #查看值， VAR6中存储着tar的标准输出**

**/root/.cache/tracker/db-version.txt /root/.cache/tracker/db-locale.txt /root/.cache/tracker/parser-sha1.txt /root/.cache/tracker/locale-for-miner-user-guides.txt /root/.cache/tracker/locale-for-miner-apps.txt /root/.cache/tracker/last-crawl.txt**

**。。。**

**实战：分享一个系统时间错误，引起tar报警告**

**[root@xuegod63 ~]# date -s "2012-03-03 21:25:00"**

**[root@xuegod63 ~]# tar zxvf root.tar.gz -C /opt/**

**root/.cache/tracker/db-version.txt**

**tar: root/.cache/tracker/db-version.txt：时间戳 2017-09-19 13:05:18 是未来的 168094624.438537189 秒之后**

**注： 如果弹出这个消息，是因为咱们的当前系统的时间不对。 比如：当前系统的时间晚于文件的mtime时间**

**[root@xuegod63 ~]# ls /opt/root/.mozilla/firefox/wggimqlt.default/ -a #解压成功**

## 19.2.5 shell中单引号和双引号区别

**‘’ 在单引号中所有的字符包括特殊字符（$,'',`和\）都将解释成字符本身而成为普通字符。**

**“” 在双引号中，除了$, '', `和\以外所有的字符都解释成字符本身，拥有“调用变量的值”、“引用命令”和“转义符”的特殊含义**

**注：\转义符，跟在\之后的特殊符号将失去特殊含义，变为普通字符。如\$将输出“$”符号，而不当做是变量引用**

**例：**

**[root@xuegod63 ~]# echo $VAR1**

**123**

**[root@xuegod63 ~]# echo \$VAR1**

**$VAR1**

**例1：给变量值赋于多个单词，需要使用单引号和双引号**

**[root@xuegod63 ~]# VAR8='xuegdo mk'**

**[root@xuegod63 ~]# echo $VAR8**

**xuegdo mk**

**例2：赋值时单引号和双引号的区别**

**[root@xuegod63 ~]# VAR8='xuegod mk $VAR1'**

**[root@xuegod63 ~]# VAR9="xuegod mk $VAR1" #双引中$符号有作用**

**[root@xuegod63 ~]# echo $VAR8**

**xuegod mk $VAR1**

**[root@xuegod63 ~]# echo $VAR9**

**xuegod mk 123**

**注：单引号之间的内容原封不动赋值给变量， 双引号之间的内容如有特殊符号会保留它的特殊含义**

**删除变量**

**[root@xuegod63 ~]# unset VAR1**

**[root@xuegod63 ~]# echo $VAR1**

## 19.2.6 环境变量

**在bash shell中，环境变量分为两类：全局变量和局部变量**

**全局变量：对于shell会话和所有的子shell都是可见的**

**局部变量： 它只在自己的进程当中使用**

**例1：局部变量**

**[root@xuegod63 ~]# VAR1=123**

**[root@xuegod63 ~]# echo $VAR1**

**123**

**[root@xuegod63 ~]# vim a.sh**

**#!/bin/bash**

**echo $VAR1**

**[root@xuegod63 ~]# echo $VAR1**

**123**

**[root@xuegod63 ~]# bash a.sh #执行a.sh 时，会使用另一个bash去执行，就访问不到$VAR1的值**

**例2：env命令查看所全局变量**

**[root@xuegod63 ~]# env**

**[root@xuegod63 ~]# env | grep PATH**

**PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin**

**例3：使用export把这个局部变量输出为全局变量**

**[root@xuegod63 ~]# export VAR1=xuegod**

**[root@xuegod63 ~]# echo $VAR1**

**xuegod**

**[root@xuegod63 ~]# vim a.sh #写入以下内容**

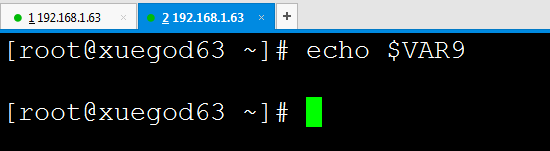
**#!/bin/bash**

**echo $VAR1**

**[root@xuegod63 ~]# bash a.sh #引用全局变量成功**

**xuegod**

**互动： 虽然我们设置了export全局变量，但是新开的xshell连接中，还是读不到变量VAR1，怎么办？**

****

**例3：让变量永久生效，可以把定义好的变量写入配置文件**

**当登录系统或新开启一个ssh连接启动bash进程时，一定会加载这4个配置文件：**

**[root@xuegod63 ~]# vim /etc/profile #系统全局环境和登录系统的一些配置**

**[root@xuegod63 ~]# vim /etc/bashrc #shell全局自义配置文件，用于自定义shell**

**[root@xuegod63 ~]# vim /root/.bashrc #用于单独自定义某个用户的bash**

**[root@xuegod63 ~]# vim /root/.bash\_profile #用户单独自定义某个用户的系统环境**

**互动：如何知道新建一个ssh连接，加载这4个配置文件先后顺序？**

**有思路：1，没有思路：2**

**要会思考，思路很重要！**

**答：可以每个文件的最后，追加一个echo命令，输出一下文件的名字**

**[root@xuegod63 ~]# echo 'echo /etc/profile ' >> /etc/profile**

**[root@xuegod63 ~]# echo 'echo /etc/bashrc' >> /etc/bashrc**

**[root@xuegod63 ~]# echo 'echo /root/.bashrc ' >> /root/.bashrc**

**[root@xuegod63 ~]# echo 'echo /root/.bash\_profile ' >> /root/.bash\_profile**

**[root@xuegod63 ~]# ssh root@192.168.1.63 #弹出以下信息，就知道有优先级了**

**/etc/profile**

**/etc/bashrc**

**/root/.bashrc**

**/root/.bash\_profile**

**互动：知道加载的顺序有什么用？**

**可以在这里添加木马程序，只要管理登录系统，就触发木马程序！ 现在大家知道学习操作系统原理的作用了吧。**

**[root@xuegod63 ~]# echo 'touch /tmp/profile.txt ' >> /etc/profile**

**[root@xuegod63 ~]# echo 'touch /tmp/bash\_profile.txt ' >> /root/.bash\_profile**

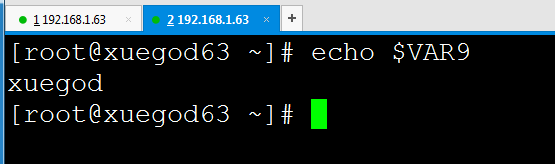
**下面开始插入永久变量：**

**[root@xuegod63 ~]# vim /etc/profile #在文件的最后插入**

**export VAR9=xuegod #=等号两边不能有空格**

**[root@xuegod63 ~]# source /etc/profile #重新加载profile文件**

**新打开的链接中，也有了**

****

## 19.2.7 设置PATH环境变量

**SHELL要执行某一个程序，它要在系统中去搜索这个程序的路径，path变量是用来定义命令和查找命令的目录，当我们安装了第三方程序后，可以把第三方程序bin目录添加到这个path路径内，就可以在全局调用这个第三方程序的**

**例1：**

**[root@xuegod63 ~]# vim /opt/backup**

**#!/bin/bash**

**echo "Backup data is OK!"**

**[root@xuegod63 ~]# chmod +x /opt/backup**

**[root@xuegod63 ~]# /opt/backup**

**[root@xuegod63 ~]# backup**

**bash: backup: 未找到命令...**

**将backup命令添加PATH中**

**[root@xuegod63 ~]# PATH=/opt/:$PATH**

**[root@xuegod63 ~]# backup #发现命令可以直接执行了，不用写全路径了**

**[root@xuegod63 ~]# vim /etc/profile # 在文件最后追加以下内容，永久生效**

**export PATH=/opt/:$PATH**

**[root@xuegod63 ~]# source /etc/profile #重新加载配置文件，使用配置生效**

## 19.2.8 shell位置变量

**Shell解释执行用户的命令时，将命令行的第一个字符作为命令名，而其它字符作为参数。**

**$0 获取当前执行shell脚本的文件文件名，包括脚本路径,命令本身**

**$n 获取当前脚本的第n个参数 n=1,2.....n 当n大于9时 用${10}表示。**

**例子：**

**[root@xuegod63 ~]# vim print.sh**

**#!/bin/bash**

**echo "本shell脚本的文件名： $0"**

**echo "第1个参数： $1"**

**echo "第2个参数： $2"**

**echo "第3个参数： $3"**

**echo "第4个参数： $4"**

**[root@xuegod63 ~]# chmod +x print.sh**

**[root@xuegod63 ~]# ./print.sh 111 222 a.txt 444**

**本shell脚本的文件名： ./print.sh**

**第1个参数： 111**

**第2个参数： 222**

**第3个参数： a.txt**

**第4个参数： 444**

**使用场景：服务器启动传参数**

**[root@xuegod63 ~]# /etc/init.d/network restart**

## 19.2.9 特殊变量

**有些变量是一开始执行Script脚本时就会设定，且不能被修改，但我们不叫它只读的系统变量，而叫它特殊变量。这些变量当一执行程序时就有了，以下是一些特殊变量：**

|  |  |
| --- | --- |
| **$\*** | **以一个单字符串显示所有向脚本传递的参数；**  **如"$\*"用【"】括起来的情况、以"$1 $2 … $n"的形式输出所有参数** |
| **$#** | **传递到脚本的参数个数** |
| **$$** | **当前进程的进程号PID** |
| **$?** | **显示最后命令的退出状态；0表示没有错误，其他任何值表明有错误** |
| **$!** | **后台运行的最后一个进程的进程号pid** |

**例子：**

**[root@xuegod63 ~]# vim special\_variable.sh #写入以一下内容**

**#!/bin/bash**

**echo "$\* 表示这个程序的所有参数 "**

**echo "$# 表示这个程序的参数个数"**

**echo "$$ 表示程序的进程ID "**

**touch /tmp/b.txt &**

**echo "$! 执行上一个后台指令的PID"**

**echo "$$ 表示程序的进程ID "**

**echo "$? 表示上一个程序执行返回结果 "**

**[root@xuegod63 ~]# bash special\_variable.sh 11 22 33 44 55**

**11 22 33 44 55 表示这个程序的所有参数**

**11 22 33 44 55 表示这个程序的所有参数**

**5 表示这个程序的参数个数**

**45502 表程序的进程ID**

**45504 执行上一个后台指令的PID**

**45502 表程序的进程ID**

**0 表示上一个程序执行返回结果**

**例2：常用的环境变量**

**[root@xuegod63 opt]# cd /opt/**

**[root@xuegod63 opt]# vim env.sh**

**#!/bin/bash**

**echo $HOME**

**echo $PATH**

**echo $PWD**

**[root@xuegod63 opt]# bash env.sh**

**/root**

**/opt/:/opt/:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/root/bin**

**/opt**

# 19.3 数学运算

## 19.3.1 expr命令

**(1)对数字的基本计算，做比较时，输出结果假为0，1为真；特殊符号用转义符**

**[root@xuegod63 ~]# expr 2 \> 5**

**0**

**[root@xuegod63 ~]# expr 6 \> 5**

**1**

**[root@xuegod63 ~]# expr 3 \* 5**

**expr: 语法错误**

**[root@xuegod63 ~]# expr 3 \\* 5**

**15**

**[root@xuegod63 ~]# expr 3 \+ 5**

**8**

**(2)对字符串的处理**

**[root@xuegod63 ~]# expr length "ni hao"**

**6**

**[root@xuegod63 ~]# expr substr "ni hao" 2 4 #从第2个开始，截取4个字符出来**

**i ha**

## 19.3.2 使用$(( ))

**格式：$（（表达式1，表达2））**

**特点：**

**1、在双括号结构中，所有表达式可以像c语言一样，如：a++,b--等。a++ 等价于 a=a+1**

**2、在双括号结构中，所有变量可以不加入：“$”符号前缀。**

**3、双括号可以进行逻辑运算，四则运算**

**4、双括号结构 扩展了for，while,if条件测试运算**

**5、支持多个表达式运算，各个表达式之间用“，”分开**

**常用的算数运算符**

|  |  |
| --- | --- |
| **运算符** | **意义** |
| **++ --** | **递增及递减，可前置也可以后置** |
| **+ - ! ~** | **一元运算的正负号 逻辑与取反** |
| **+ - \* / %** | **加减乘除与余数** |
| **< <= > >=** | **比较大小符号** |
| **== !=** | **相等 不相等** |
| **>> <<** | **向左位移 向右位移** |
| **& ^ |** | **位的与 位的异或 位的或** |
| **&& ||** | **逻辑与 逻辑或** |
| **？ :** | **条件判断** |

**例1：**

**[root@xuegod63 opt]# b=$((1+2))**

**[root@xuegod63 opt]# echo $b**

**3**

**[root@xuegod63 opt]# echo $((2\*3))**

**6**

**例2：递增和递减**

**[root@xuegod63 opt]# echo $((b++))**

**4**

**[root@xuegod63 opt]# echo $((++b))**

**6**

**说明： a++或a--为先赋值再+1或减1 ； ++a或--a为先加1或减1，然后再进行赋值**

**例3：**

**求1到100的和**

**[root@xuegod63 opt]# echo $((100\*(1+100)/2))**

**5050**

# 19.4 实战-升级系统中的java版本到1.8版本-为后期安装Hadoop集群做准备

## 19.4.1 安装jdk java运行环境

**上传jdk-8u161-linux-x64.rpm软件包到xuegod63**

**[root@xuegod63 ~]# rpm -ivh jdk-8u161-linux-x64.rpm**

**[root@xuegod63 ~]#rpm -pql /root/jdk-8u161-linux-x64.rpm #通过查看jdk的信息可以知道jdk的安装目录在/usr/java**

**[root@xuegod63 ~]#vim /etc/profile  #在文件的最后添加以下内容：**

**export JAVA\_HOME=/usr/java/jdk1.8.0\_161**

**export JAVA\_BIN=/usr/java/jdk1.8.0\_161/bin**

**export PATH=${JAVA\_HOME}/bin:$PATH**

**export CLASSPATH=.:${JAVA\_HOME}/lib/dt.jar:${JAVA\_HOME}/lib/tools.jar**

**[root@xuegod63 ~]#source /etc/profile #使配置文件生效**

**验证java运行环境是否安装成功：**

**[root@xuegod63 ~]# java -version**

**java version "1.8.0\_161"**

**Java(TM) SE Runtime Environment (build 1.8.0\_161-b12)**

**Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)**

**如果出现安装的对应版本，说明java运行环境已经安装成功。**

**注：这里只是升级了jdk的版本，因为在我安装的系统中已经安装了jdk。**

**总结：**

**19.1 shell 基本语法**

**19.2 SHELL变量及运用**

**19.3 数学运算**

**19.4 实战-升级系统中的java版本到1.8版本-为后期安装Hadoop集群做准备**