# RHadoop培训 之 R基础课

[](http://blog.fens.me/wp-content/uploads/2013/07/r-title.png)

**前言**  
覆盖R基础知识，快速上手，RHadoop环境的搭建基础课。

**目录**

1. 背景知识
2. 开发环境
3. R语法
4. R基本函数
5. R的扩展包

**1. 背景知识**

**R起源**  
R 是一个有着统计分析功能及强大作图功能的软件系统，是由奥克兰大学统计学系的Ross Ihaka和Robert Gentleman 共同创立。由于R 受Becker, Chambers & Wilks 创立的S 和Sussman 的Scheme两种语言的影响，所以R 看起来和S 语言非常相似。

R 是一个世界范围统计工作者共同协作的产物，至2013 年2 月共计近5000 个包可在互联网上自由下载，这些都是各行业数据分析同行的工作结晶。

**R的特点**

1. 有效的数据处理和保存机制。
2. 拥有一整套数组和矩阵的操作运算符。
3. 一系列连贯而又完整的数据分析中间工具。
4. 图形统计可以对数据直接进行分析和显示，可用于多种图形设备。
5. 一种相当完善、简洁和高效的程序设计语言。它包括条件语句、循环语句、用户自定义的递归函数以及输入输出接口。
6. R语言是彻底面向对象的统计编程语言。
7. R语言和其它编程语言、数据库之间有很好的接口。
8. R语言是自由软件，可以放心大胆地使用，但其功能却不比任何其它同类软件差。
9. R语言具有丰富的网上资源

**R的下载和安装**  
R是一个免费的自由软件，它有UNIX、LINUX、MacOS和WINDOWS版本，都是可以免费下载和使用的,在那儿可以下载到R的安装程序、各种外挂程序和文档。在R的安装程序中只包含了8个基础模块，其他外在模块可以通过CRAN获得。

R的官方网站： http://www.r-project.org/

**Linux Ubuntu的R安装**

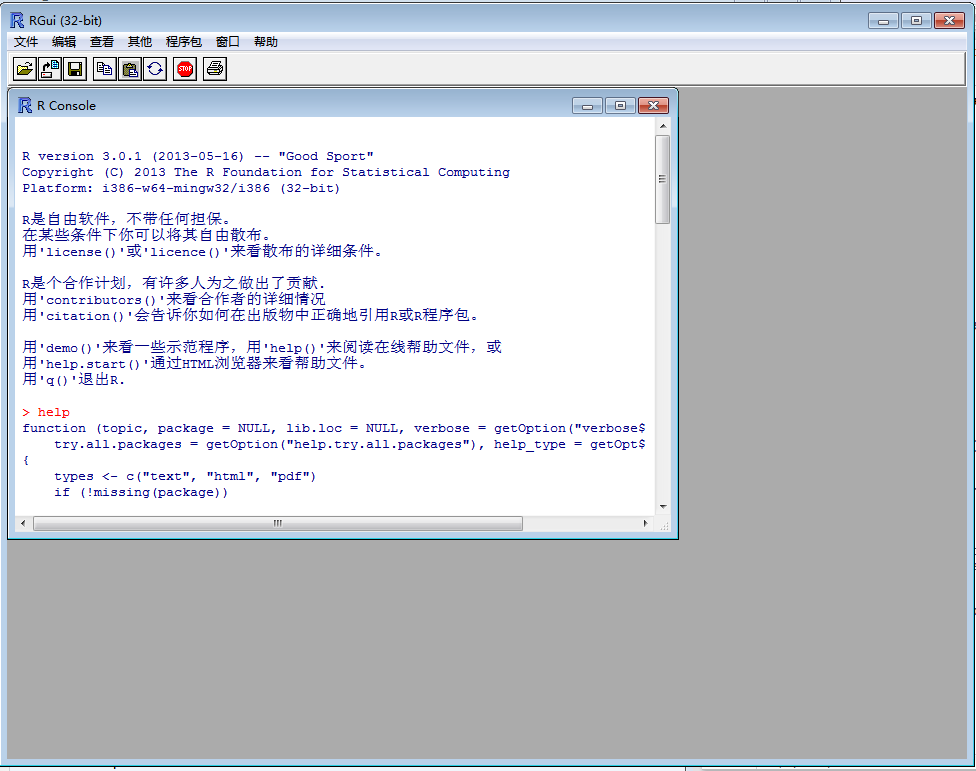
~ sudo vi /etc/apt/sources.list

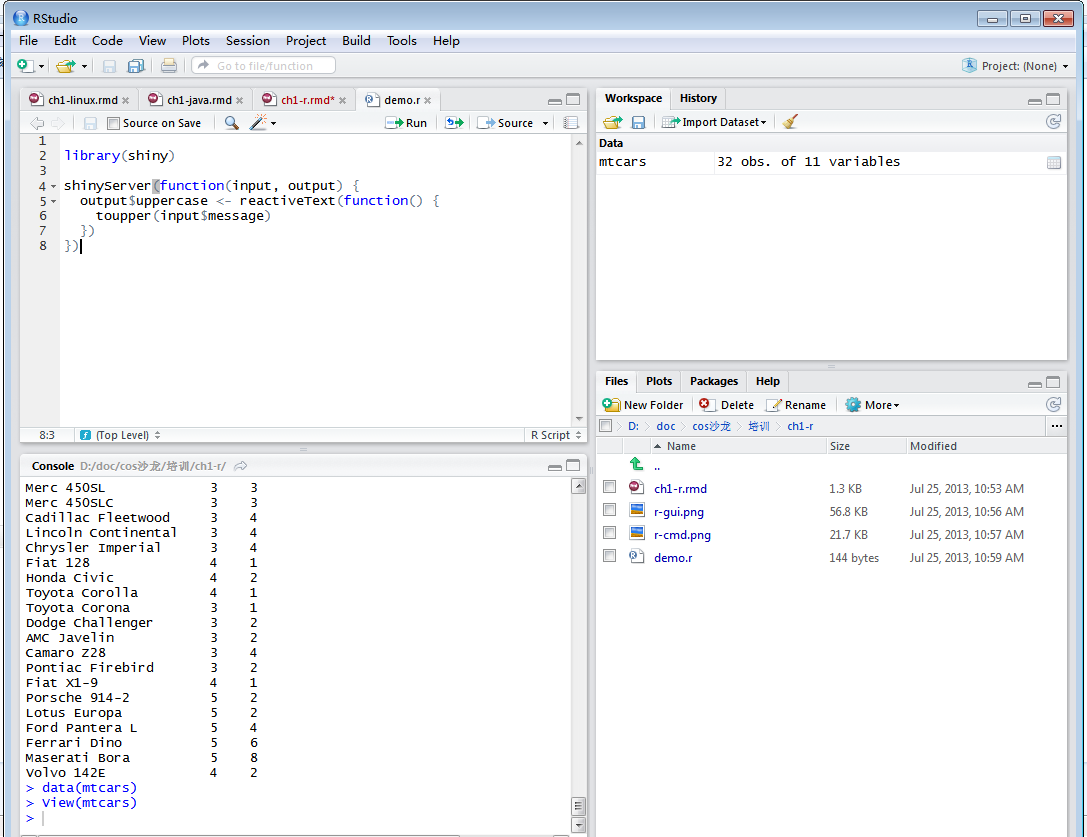
deb http://mirror.bjtu.edu.cn/cran/bin/linux/ubuntu precise/

~ sudo apt-get update

~ sudo apt-get install r-base-core=2.15.3-1precise0precise1

**2. 开发环境**

R默认的界面环境：  
[](http://blog.fens.me/wp-content/uploads/2013/07/r-gui.png)

RStudio的IDE开发环境： http://www.rstudio.com/  
[](http://blog.fens.me/wp-content/uploads/2013/07/r-studio.png)

**3. R语法**

R是一种语法非常简单的表达式语言(expression language),大小写敏感。  
可以在R 环境下使用的命名字符集依赖于R 所运行的系统和国家(系统的locale 设置),允许数字,字母,“.”,“\_”

**1). 命名**  
命名必须以”.”或者字母开头，以”.”开头时第二个字符不允许是数字。

**2). 基本命令**  
基本命令要么是表达式（expressions），要么就是赋值（assignments）。

* 表达式，命令将被解析，并将结果显示在屏幕上，同时清空该命令所占内存。
* 赋值，命令将被解,并把值传给变量,但结果不会自动显示在屏幕上。

命令可以被”;”隔开或者另起一行。基本命令可以通过大括弧{},放在一起构成一个复合表达式。

注释：一行中以井号”#”开头  
换行：如果一条命令在一行结束的时候在语法上还不完整，换行提示符，默认是+

**3). 基本的对象**  
R创建和控制的实体被称为对象。它们可以是变量，数组，字符串，函数，或者其他通过这些实体定义的一般性的结构。

* 矩阵（matrix）或者更为一般的数组（array）是多维的广义向量。实际上，它们就是向量，而且可以同时被两个或者更多个索引引用，并且以特有的方式显示出来。
* 因子（factor）为处理分类数据提供的一种有效方法。
* 列表（list）是一种泛化（general form）的向量。它没有要求所有元素是同一类型，许多时候它本身就是向量和列表类型。列表为统计计算的结果返回提供了一种便利的方法。
* 数据框（data frame）是和矩阵类似的一种结构。在数据框中，列可以是不同的对象。可以把数据框看作是一个行表示观测个体并且（可能）同时拥有数值变量和分类变量的`数据矩阵’ 。许多实验数据都可以很好的用数据框描述：处理方式是分类变量而响应值是数值变量。
* 函数（function）是可以保存在项目工作空间的R 对象。该对象为R 提供了一个简单而又便利的功能扩充方法。见编写你自己的函数

在R会话过程中，对象是通过名字创建和保存的。objects(), ls()可以显示当前会话的对象名字。rm()可以删除对象。

**对象持久化**  
R 会话中创建的所有对象可以永久地保存在一个文件中以便于以后的R 会话调用。在每一次R 会话结束的时候，你可以保存当前所有可用的对象。如果你想这样做，这些对象将会写入当前目录下一个叫.RData的文件中，并且所有在这次会话中用过的命令行都会被保存在.Rhistory 的文件中。当R 再次在同一目录下启动，这些对象将从这个文件中重新导入工作空间。同时，相关的历史命令文件也会被导入。

**4). 向量和赋值**  
向量是由一串有序数值构成的序列

x <- c(10.4, 5.6, 3.1, 6.4, 21.7)

函数c()完成的赋值语句。这里的函数c() 可以有任意多个参数，而它返回的值则是一个把这些参数首尾相连形成的向量。

赋值也可以用函数assign()实现。

assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))

赋值符<-,->可以看作是该命令一个语义上的缩写。

c(10.4, 5.6, 3.1, 6.4, 21.7) -> x

**向量运算**  
在算术表达式中使用向量将会对该向量的每一个元素都进行同样算术运算。  
出现在同一个表达式中的向量最好是长度一致。如果他们的长度不一样，该表达式的值将是一个和其中最长向量等长的向量。  
表达式中短的向量会被循环使用以达到最长向量的长度。  
对于一个常数就是简单的重复。

v <- 2\*x + y + 1

**逻辑向量**  
逻辑向量元素可以被赋予的值,有TRUE，FALSE 和NA 逻辑向量可以由条件式（conditions）产生 temp <- x > 13

**字符向量**  
字符向量就是字符串，可以用双引号和单引号作分割符。

paste():可以把单独的字符连成字符串，可以有任意多的参数。参数中的任何数字都将被显式地强制转换成字符串，而且以同样的方式在终端显示。默认的分隔符是单个的空格符。

修改分隔符换成”“

labs <- paste(c("X","Y"), 1:10, sep="")

**索引向量：通过索引值可以选择和修改一个数据集的子集**  
一个向量的子集元素可以通过向量名后面的方括号中加入索引向量得到。如果一个表达式的结果是向量，则我们可以直接在表达式的末尾方括号中加入索引向量以得到结果向量的子向量.

* **逻辑向量**:索引向量必须和被挑选元素的向量长度一致。向量中对应索引向量元素为TRUE 的元素将会被选中，而那些对应FALSE 的元素则被忽略。

y <- x[!is.na(x)]

* **正整数向量**:索引向量必须是1, 2, … , length(x)的子向量。索引向量中索引对应的元素将会被选中，并且在结果向量中的次序和索引向量中的次序一致。这种索引向量可以是任意长度的，结果向量的长度和索引向量完全一致。

x[1:10]

* **负整数向量**:这种索引向量指定被排除的元素而不是包括进来。

y <- x[-(1:5)]

* **字符串向量**:这可能仅仅用于一个对象可以用names 属性来识别它的元素。名字向量的子向量可以像上面第二条提到的正整数标签一样使用。
* fruit <- c(5, 10, 1, 20)
* names(fruit) <- c("orange”, “banana”, “apple”, “peach”)
* lunch <- fruit[c(“apple”,“orange”)]

**5).运算符**  
算术运算符：

+ - \* /

逻辑运算符：

<，<=，>，>=，==,!=, &, |, !

数学函数：

log，exp，sin，cos，tan，sqrt ，max ，min，range，length，sum，prod，var

注：var(x): 等价于sum((x-mean(x))^2)/(length(x)-1)

**6). 控制语句**  
**条件语句:if**

if (expr1 ) expr2 else expr3

**循环控制：for,repeat,while**

for (name in expr1 ) expr2

其中name 是循环变量，expr1是一个向量表达式，而expr2常常是根据虚拟变量name 而设计的成组表达式。在name 访问expr1所有可以取到的值时，expr2都会运行。

**警告：相比其他程序语言，R代码里面很少使用for(),执行效率很低**

repeat expr

while (condition) expr

关键字break：可以用于结束任何循环，甚至是非常规的。它是结束repeat 循环的唯一办法。  
关键字next：可以用来结束一次特定的循环，然后直接跳入"下一次"循环。

**7). 生成正则序列**  
1:30 语句：等价于向量c(1, 2, …, 29, 30) 30:1 语句：可用来产生一个逆向的数列。  
seq：数列生成中最为常用的工具。seq(1,30,1) rep：把一个数的完整拷贝多次，保持数列顺序req(x,times=5)

**8). 缺损值**  
在某些情况下，向量的元素可能有残缺. 当一个元素或者值在统计的时候"不可得到"（not available）或者"值丢失" （missing value），相关位置可能会被保留并且赋予一个特定的值NA。 任何含有NA 数据的运算结果都将是NA。 函数is.na(x)返回一个和x同等长度的向量。它的某个元素值为TRUE 当且仅当x中对应元素是NA。

z <- c(1:3,NA); ind <- is.na(z)

第二种"缺损"值，也称为非数值NaN(Not a Number)

0/0 或 Inf

**9). 对象**  
**内在属性：模式和长度**  
R操作的实体是对象。实数或复数向量，逻辑向量和字符串向量之类的对象属于"原子"型的对象，因为它们的元素都是一样的类型或模式。R的对象类型包括数值型，复数型，逻辑型,字符型,和原生型。向量必须保证它的所有元素是一样的模式。因此任何给定的向量必须明确属于逻辑性，数值型，复数型，字符型或者原生型.

列表是任何模式的对象的有序序列。列表被认为是一种"递归"结构而不是原子结构,因为它们的元素可以以它们各自的方式单独列出。函数和表达式也是递归结构。

所有对象都有模式(mode)和长度(length)两个内在属性

mode(x),length(x)

**外部属性**  
attributes(x):给出对象当前定义的非内在属性的列表。  
attr(object, name): 可以用来设置的外部属性。

z<-c(1:3,NA)

attr(z, "name") <- "abc"

attributes(z)

**对象的类属性**  
R里面的所有对象都属于类(class)，可以通过函数class(x)查看。  
对于简单的向量，类是对应的模式(mode)："numeric"，"logical"，"character" 或者"list"  
其他的类型，像"matrix"，"array"，"factor" 和"data.frame" 就可能是其他值。

**10). 因子(factor)**  
假定我们有一份来自澳大利亚所有州和行政区的信息样本 以及他们各自所在地的州名。

state <- c("tas", "sa", "qld", "nsw", "nsw", "nt")

在字符向量中，"有序"意味着以字母排序的。

创建因子factor：

statef <- factor(state)

statef

[1] tas sa qld nsw nsw nt

Levels: nsw nt qld sa tas

levels():可以用来得到因子的水平（levels）。

levels(statef)

[1] "nsw" "nt" "qld" "sa" "tas"

**函数tapply()和不规则数组**  
为计算样本中每个州的平均收入，我们可以用函数tapply()：

incomes <- c(60, 49, 40, 61, 64, 60)

incmeans <- tapply(incomes, statef, mean)

> incmeans

nsw nt qld sa tas

62.5 60.0 40.0 49.0 60.0

函数tapply() 可以用来处理一个由多个分类因子决定的向量下标组合。例如，我们可能期望通过州名和性别把这税务会计师分类。

**有序因子**  
因子的水平是以字母顺序排列的，或者显式地在factor中指定。有时候因子的水平有自己的自然顺序并且这种顺序是有意义的。ordered()就是用来创建这种有序因子, ordered()和factor 基本完全一样。

大多数情况下，有序和无序因子的唯一差别在于前者显示的时候反应了各水平的顺序。在线性模型拟合的时候，两种因子对应的对照矩阵的意义是完全不同的。

**11). 数组**  
数组可以看作是带有多个下标类型相同的元素集合。  
维度向量(dimension vector)是一个正整数向量。如果它的长度为k，那么该数组就是k-维的。

向量只有在定义了dim属性后才能作为数组在R中使用。  
假定，z是一个含1500个元素的向量

z<-seq(1,1500)

dim(z) <- c(3,5,100)

attributes(z)

$dim

[1] 3 5 100

对dim属性的赋值使得该向量成一个3 ×5 ×100 的数组

**数组索引**  
数组元素可以通过给定数组名及其后方括号中用逗号隔开的下标访问。可以根据索引数组去给数组中不规则的元素集合赋值或者将数组中特定的元素返回到一个向量中

**array()**  
除了用设定一个向量dim 属性的方法来构建数组，它还可直接通过函数array将向量转换得到.

假定向量h 有24个或更少的数值，那么命令

h<-seq(1,24)

Z <- array(h, dim=c(3,4,2))

#等价操作

dim(Z) <- c(3,4,2)

**向量和数组混合运算**

* + 表达式运算是从左到右进行的
  + 短的向量操作数将会被循环使用以达到其他操作数的长度
  + 有且只有短的向量和数组在一起，数组必须有一样的属性dim，否则返回一个错误
  + 向量操作数比矩阵或者数组操作数长时会引起错误
  + 如果数组结构给定，同时也没有关于向量的错误信息和强制转换操作，结果将是一个和它的数组操作数属性dim 一致的数组。

**数组的外积**  
数组一个非常重要的运算就是外积运算（outer product）。如果a 和b 是两个数值数组，它们的外积将是这样的一个数组：维度向量通过连接两个操作数的维度向量得到；数据向量则由a的数据向量元素和b的数据向量元素的所有可能乘积得到。外积是通过特别的操作符%o%实现：

ab <- a %o% b

ab <- outer(a, b, "\*")

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 4 6

[3,] 3 6 9

命令中的乘法操作符可以被任意一个双变量函数代替。

x<-c(1,2,3);y<-c(2,3,4)

f <- function(x, y) cos(y)/(1 + x^2)

z <- outer(x, y, f)

两个常规向量的外积是一个双下标的数组(就是矩阵，最大秩为1)。

**数组的广义转置**  
函数aperm(a, perm) 可以用来重排一个数组a

B <- aperm(z, c(2,1))

#等价操作

t(z)

**12). 矩阵**

矩阵是一个双下标的数组. R包括许多只对矩阵操作的操作符和函数。例如上面提到的t(X)就是矩阵的转置函数。函数nrow(A) 和ncol(A) 将会分别返回矩阵A 的行数和列数。

**矩阵相乘**  
操作符%\*% 用于矩阵相乘。  
如果A和B是大小一样的方阵，那么

A \* B

将是一个对应元素乘积的矩阵，而

A %\*% B

则是一个矩阵积。如果x 是一个向量，那么

x %\*% A %\*% x

crossprod():可以完成"矢积"运算，也就是说crossprod(X,y) 和t(X) %\*% y 等价，而且在运算上更为高效。  
diag():返回以该向量元素为对角元素的对角矩阵。

**性方程和求逆**  
求解线性方程组是矩阵乘法的逆运算。当下面的命令运行后，

b <- A %\*% x

如果仅仅给出A 和b，那么x 就是该线性方程组的根。在R 里面，用命令

solve(A,b)

矩阵的逆可以用下面的命令计算，

solve(A)

**特征值和特征向量**  
ev<-eigen(Sm):用来计算矩阵Sm 的特征值和特征向量。这个函数的返回值是一个 含有values 和vectors 两个分量的列表。

ev <- eigen(Sm)

ev$values表示Sm 的特征值向量，ev$vec 则是相应特征向量构成的一个矩阵。

**奇异值分解和行列式**  
svd(M): 可以把任意一个矩阵M作为一个参数, 且对M 进行奇异值分解。这包括一个和M 列空间一致的正交列U 的矩阵，一个和M 行空间一致的正交列V 的矩阵，以及一个正元素D 的对角矩阵，如M = U %\*% D %\*% t(V)。D 实际上以对角元素向量的形式返回。svd(M) 的结果是由d, u 和v构成的一个列表。

如果M 是一个方阵，就不难看出

absdetM <- prod(svd(M)$d)

计算M 行列式的绝对值。如果在各种矩阵中都需要这种运算，我们可以把它定义为一个R 函数

absdet <- function(M) prod(svd(M)$d)

此后, 我们可以把absdet() 当一个R 函数使用了。R有一个计算行列式的内置函数det和另外一个给出符号和模的函数。

evals <- eigen(Sm, only.values = TRUE)$values

**cbind()和rbind()构建分块矩阵**  
函数cbind() 和rbind():把向量和矩阵拼成一个新的矩阵。cbind() 把矩阵横向合并成一个大矩阵（列方式），而rbind()是纵向合并（行方式）。  
对数组实现连接操作的函数c()  
将一个数组强制转换成简单向量的标准方法是用函数as.vector()。

vec <- as.vector(X)

#等价操作

vec <- c(X)

**因子的频率表**  
单个因子会把各部分数据分成不同的组。类似的是，一对因子可以实现交叉分组等。函数table() 可以从等长的不同因子中计算出频率表。如果有k 个因子参数，那么结果将是一个k-维的频率分布数组。

statefr <- table(statef)

statefr <- tapply(statef, statef, length)

**13). 列表(list)**  
R的列表是一个以对象的有序集合构成的对象。列表中包含的对象又称为它的分量（components）。分量可以是不同的模式或类型，如一个列表可以同时包括数值向量，逻辑向量，矩阵，复向量，字符数组，函数等等。

Lst <- list(name="Fred", wife="Mary", no.children=3,child.ages=c(4,7,9))

分量常常会被编号，并且可以利用这种编号来访问分量。如果列表Lst 有四个分量，这些分量则可以用Lst[[1]], Lst[[2]], Lst[[3]] 和Lst[[4]] 独立访问。因为Lst 是一个列表，所以函数length(Lst) 给出的仅仅是分量的数目. 列表的分量可以被命名，这种情况下可以通过名字访问。

**构建和修改列表**  
list():将已有的对象构建成列表。  
Lst[5] <- list(matrix=Mat)

**列表连接**  
当连接函数c() 的参数中有列表对象时，结果就是一个列表模式的对象。它的分量是那些当作参数的列表。

list.ABC <- c(list.A, list.B, list.C)

**14). 数据框**  
数据框是一个属于data.frame类的列表。

对于可能属于数据框的列表对象有下面一些限制条件，

* + 分量必须是向量(数值, 字符, 逻辑)，因子，数值矩阵，列表或者其他数据框;
  + 矩阵，列表和数据框为新的数据框提供了尽可能多的变量，因为它们各自拥有列，元素或者变量;
  + 数值向量，逻辑值，因子保持原有格式，而字符向量会被强制转换成因子并且它的水平就是向量中出现的独立值;
  + 在数据框中以变量形式出现的向量结构必须长度一致，矩阵结构必须有一样的行数.
  + 数据框常常会被看作是一个由不同模式和属性的列构成的矩阵。

**创建数据框**  
可以通过函数data.frame 创建符合上面对列(分量)限制的数据框对象：

accountants <- data.frame(home=statef, loot=incomes)

符合数据框限制的列表可被函数as.data.frame() 强制转换成数据框。

**绑定任意的列表**  
attach() 是一个泛型函数。它不仅允许搜索路径绑定目录和数据框，而且还可以绑定其他对象。所以任何其他"list" 模式的对象都可以这样绑定：

attach(any.old.list)

任何被绑定的东西都可利用detach 通过位置编号或者名字（最好采用名字）去 除绑定。

**管理搜索路径**  
search(): 显示当前的搜索路径。它可以用来跟踪已被绑定或者绑定去除的列表和数据框(以及包)。

search()

[1] “.GlobalEnv” “Autoloads” “package:base”

其中.GlobalEnv 是工作空间

lentils 被绑定后，我们可以看到

search()

[1] ".GlobalEnv" "lentils" "Autoloads" "package:base"

ls(2)

[1] "u" "v" "w"

ls (或者objects) 可以用来查看搜索路径中任何位置的内容。

**15). 读数据**  
大的数据对象常常是从外部文件中读入，而不是在R 对话时用键盘输入的。  
**read.table()函数**  
为了可以直接读取整个数据框，外部文件常常要求有特定的格式。 第一行可以有该数据框各个变量的名字。 随后的行中第一个条目是行标签，其他条目是各个变量的值。  
**scan() 函数**  
假定有三个数据向量，长度一致并且要求并行读入。其中，第一个向量是字符模式，另外两个是数值模式，文件是input.dat。第一步是用scan() 以列表的形式读入这三个向量，  
**访问内置数据**  
R 提供了大约100个内置的数据集(在包datasets 中)，其他的包(包括和R捆绑发布的推荐包) 也提供了一些作为例子的数据集。可以用下面的命令查看当前能访问的数据集列表

data()

**从其他R 包里面导入数据**  
为了访问某个特定包的数据，可以使用参数package，例如

data(package=“rpart”)

data(Puromycin, package=“datasets”)

如果一个包已经被library 绑定，那么它的数据集会被自动地加入到R 的搜索路径中去。

**编辑数据**  
edit(x):调用数据框和矩阵时，R 会产生一个电子表形式的编辑环境。

xnew <- edit(xold)

**16) 编写函数**  
R语言允许用户创建自己的函数（function）对象,如mean(), var(),postscript() 等等,这些函数都是用R 写的，因此在本质上和用户写的没有差别。

一个函数是通过下面的语句形式定义的，

name <- function(args, ...) {}

其中expression 是一个R 表达式(常常是一个成组表达式)，它利用参数argi 计算最终 的结果。该表达式的值就是函数的返回值。 可以在任何地方以name(expr1 , expr2 , …) 的形式调用函数。

函数定义如下：

twosam <- function(y1, y2) {

n1 <- length(y1); n2 <- length(y2)

yb1 <- mean(y1); yb2 <- mean(y2)

s1 <- var(y1); s2 <- var(y2)

s <- ((n1-1)\*s1 + (n2-1)\*s2)/(n1+n2-2)

tst <- (yb1 - yb2)/sqrt(s\*(1/n1 + 1/n2))

tst

}

a<-1:3;b<-5:7

twosam(a,b)

**参数命名和默认值**  
和产生正则序列一样，如果被调用函数的参数以"name=object"方式给出，它们可以用任何顺序设置。但是，参数赋值序列可能以未命名的，位置特异性的方式给出，同时也有可能在这些位置特异性的参数后加上命名参数赋值。 因此，如果有下面方式定义的函数fun1

fun1 <- function(data, data.frame, graph, limit) {}

**… 参数**  
一个函数的参数设置可以传递给另外一个函数。这个可以通过给函数增加一个额外的参数来实现。  
举例如plot

plot

function (x, y, ...)

**在函数中赋值**  
注意任何在函数内部的普通赋值都是局部的暂时的，当退出函数时都会丢失。因此函数中的赋值语句X <- qr(X) 不会影响调用该函数的程序赋值情况。

“强赋值"操作符 <<- :如果想在一个函数里面全局赋值或者永久赋值

fscope<-function(){

a<-1

b<<-2

c=3

}

**作用域**  
函数内部的变量可以分为三类：形式参数(formal parameters)，局部变量(local variables),自由变量(free variables)。

* + 形式参数是出现在函数的参数列表中的变量。它们的值由实际的函数参数绑定形式参数的过程决定的。
  + 局部变量由函数内部表达式的值决定的。既不是形式参数又不是局部变量的变量是自由变量。
  + 自由变量如果被赋值将会变成局部变量

z<-45

f <- function(x) {

y <- 2\*x

print(x)

print(y)

print(z)

}

x 是形式参数，y 是局部变量，z 是自由变量。

**定制环境**  
可以修改位置初始化文件，并且每个目录都可以有它特有的一个初始化文件。利用函数.First 和.Last。位置初始化文件的路径可以通过环境变量R PROFILE 设置。这个文件包括你每次执行R时一些自动运行的命令。

**类，泛型函数和面向对象**  
一个对象的类决定了它会如何被一个泛型函数处理。相反，一个泛型函数由参数自身类的种类来决定完成特定工作或者事务的。如果参数缺乏任何类属性，或者在该问题中有一个不能被任何泛型函数处理的类，泛型函数会有一种默认的处理方式。

下面的一个例子使这个问题变得清晰。类机制为用户提供了为特定问题设计和编写泛型函数的便利。在众多泛型函数中，plot() 用于图形化显示对象，summary()用于各种类型的概述分析，以及anova() 用于比较统计模型。 能以特定方式处理类的泛型函数的数目非常庞大。

methods() 得到当前对某个类对象可用的泛型函数列表：

methods(class="data.frame")

相反，一个泛型函数可以处理的类同样很多。例如，plot() 有默认的方法和变 量处理对象类"data.frame"，"density"，"factor"，等等。

一个完整的列表同样可以通过函数methods():

methods(plot)

**17) R中的统计模型**  
线性模型,对于常规的多重模型拟合，最基本的函数是lm()。

fm2 <- lm(y ~ x1 + x2, data = production)

将会拟合y 对x1 和x2 的多重回归模型和一个隐式的截距项

**提取模型信息的泛型函数**  
lm() 的返回值是一个模型拟合结果对象；技术上就是属于类"lm” 的一个结果列表。关于拟合模型的信息可以用适合对象类"lm" 的泛型函数显示，提取，图示等等。

add1 coef effects kappa predict residuals alias

deviance family labels print step anova drop1

formula plot proj summary

* + anova(object1 , object2) 比较一个子模型和外部模型，并且产生方差分析表。
  + coef(object) 提取回归系数(矩阵)。全称：coefficients(object).
  + deviance(object) 残差平方和，若有权重可加权。
  + formula(object) 提取模型公式信息。
  + plot(object) 产生四个图，显式残差，拟合值和一些诊断图。
  + predict(object, newdata=data:frame) 提供的数据框必须有同原始变量一样标签的变量。结果是对应于data:frame中决定变量预测值的向量或矩阵。
  + predict.gam(object,newdata=data:frame) predict.gam() 是安全模式的predict()。它可以用于lm, glm和gam 拟合对象。在正交多项式作为原始的基本函数并且增加新数据意味着必须使用不同的原始基本函数。
  + print(object) 简要打印一个对象的内容
  + residuals(object) 提取残差(矩阵)，有权重时可加权，省略方式：resid(object)。
  + step(object) 通过增加或者减少模型中的项并且保留层次来选择合适的模型。在逐步搜索过程中，AIC (Akaike信息规范)值最大的模型将会被返回。
  + summary(object) 显示较详细的模型拟合结果

**18). 图形工具**  
图形工具是R 环境里面一个非常重要和多用途的组成部分。我们可以用这些图形工具显示各种各样的统计图并且创建一些全新的图。

图形工具既可交互式使用，也可以批处理使用。在许多情况下，交互式使用是最有效的。打开R 时，它会启动一个图形设备驱动（device driver）。该驱动会打开特定的图形窗口（graphics window）以显示交互式的图片。尽管这些都是自动实现的，了解用于UNIX 系统的X11() 命令和Windows 系统的windows() 命令是非常有用的。一旦设备驱动启动，R 绘图命令可以用来产生统计图或者设计全新的图形显示。

绘图命令可以分成了三个基本的类：

* + 高级绘图命令: 在图形设备上产生一个新的图区，它可能包括坐标轴，标签，标题等等。
  + 低级画图命令: 会在一个已经存在的图上加上更多的图形元素，如额外的点，线和标签。
  + 交互式图形命令: 允许你交互式地用定点设备（如鼠标）在一个已经存在的图上添加图形信息或者提取图形信息。

**高级绘图命令**  
plot(),这是一个泛型函数：产生的图形依赖于第一个参数的类型或者类。 pairs(X),描绘多元数据提供了两个非常有用的函数

**低级图形函数**  
高级图形函数不能准确产生你想要的图。低级图形命令可以在当前图上精确增加一些额外信息(如点，线或者文字)。 points(x, y) lines(x, y)

**数学标注**  
在某些情况下，在一个图上加上数学符号和公式是非常有用的。在R 里面，这可以通过函数expression 实现，

text(x, y, expression(paste(bgroup("(", atop(n, x), ")"), p^x, q^{n-x})))

**图像设备输出**

* + X11() 用UNIX 类型的系统的X11 桌面系统
  + windows() 用于Windows 系统
  + quartz() 用于MacOS X 系统
  + postscript() 用于PostScript 打印机，或者创建PostScript 文件。
  + pdf() 创建可以插入PDF 文件中PDF 文件
  + png() 创建PNG 位图文件。(不总是有效的：参考它的帮助文件)
  + jpeg() 创建JPEG 位图文件，非常适用于影

**19). 包(packages)**  
所有的R 函数和数据集是保存在包里面的。只有当一个包被载入时，它的内容才可以被访问。这样做一是为了高效，二是为了帮助包的开发者防止命名和其他代码中的名字冲突。

library():查看系统中安装的包 library(plyr):加载plyr包 CRAN.packages() 连接因特网，并且允许自动更新和安装包。 search()为了查看当前有那些包载入了

**标准包**  
标准包构成R 原代码的一个重要部分。它们包括允许R 工作的的基本函数，和本文档中描述的数据集，标准统计和图形工具。在任何R 的安装版本中，它们都会被自动获得。

**捐献包和CRAN**  
世界各地的作者为R 捐献了好几百个包。其中一些包实现了特定的统计方法，另外一些给予数据和硬件的访问接口，其他则作为教科书的补充材料。 可以从CRAN (http://CRAN.R-project.org/ 和它的镜像)和其他一些资源，如Bioconductor (http://www.bioconductor.org/) 下载得到

**命名空间**  
包有命名空间(namespaces),并且现在所有基本的和推荐的的包都依赖于包datasets。

它们允许包的作者隐藏函数和数据，即只允许内部用户使用，它们防止函数在一个用户使用相同名字时被破坏，它们提供了一种访问特定包的某个对象的方法。

有两个操作符和命名空间相关。 双冒号操作符:: 选择一个特定命名空间得到的函数定义。可以通过base::t 使用，因为它是在包base 中定义的。 三冒号操作符::: 可能会出现在一些R 代码中：它有点像双冒号操作符，但可以访问隐藏对象。

包常常是包之间依赖的（inter-dependent），载入其中一个可能会引起其他包的自动载入。

**4. R基本函数**

请查看：R参考卡片，[点击下载](http://doc.fens.me/cos/R-refcard.pdf)

**5. R的扩展包**

**1). plyr (数据处理)**  
plyr是一个数据处理的包，可以把大的数据集通过一些条件拆分成小的数据集的功能包。

数据集baseball(http://www.baseball-databank.org/)  
21,699 records covering 1,288 players, spanning 1871 to 2007

举例：击球得分：RBI(Run batted in)

install.packages("plyr")

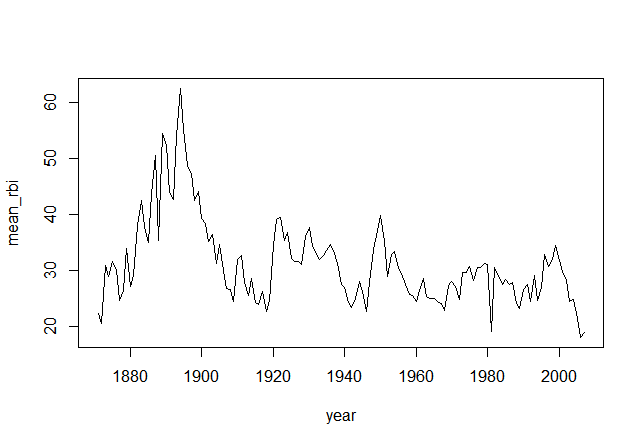
library(plyr)

?baseball

ddply(baseball, .(lg), c("nrow", "ncol"))

rbi <- ddply(baseball, .(year), summarise, mean\_rbi = mean(rbi, na.rm = TRUE))

plot(mean\_rbi ~ year, type = "l", data = rbi)

[](http://blog.fens.me/wp-content/uploads/2013/07/rbi-plyr.png)

**2). stringr (字符串处理)**  
stringr是一个字符串处理的包，可以方便地进行各种字符串的操作。

install.packages("stringr")

library(stringr)

fruits <- c("one apple", "two pears", "three bananas")

str\_replace(fruits, "[aeiou]", "-")

str\_replace\_all(fruits, "[aeiou]", "-")

str\_replace\_all(fruits, "([aeiou])", "")

str\_replace\_all(fruits, "([aeiou])", "\\1\\1")

str\_replace\_all(fruits, "[aeiou]", c("1", "2", "3"))

str\_replace\_all(fruits, c("a", "e", "i"), "-")

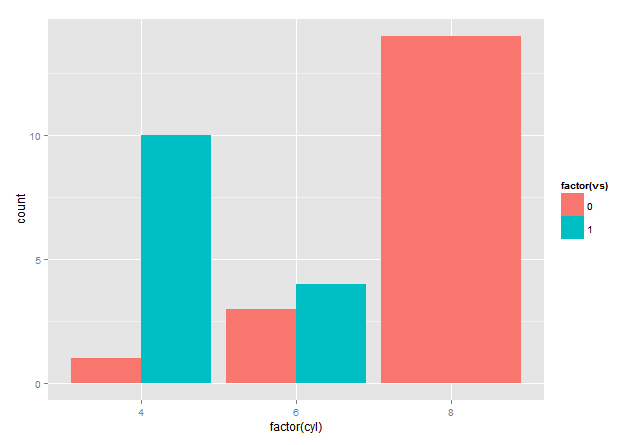
**3). ggplot2 (可视化)**  
ggplot2专业级的可视化绘图包

install.packages("ggplot2")

library(ggplot2)

g<-ggplot(mtcars, aes(x=factor(cyl), fill=factor(vs)))

g+geom\_bar(position="dodge")

[](http://blog.fens.me/wp-content/uploads/2013/07/ggplot2.png)

<http://blog.fens.me/rhadoop-r-basic/>

# RHadoop培训 之 Java基础课

**目录**

1. 背景知识
2. 开发环境
3. JAVA的编译及运行环境
4. JAVA语法
5. JDK基本包介绍
6. JAVA项目(ant, maven)

**1. 背景知识**

**Java起源**

Java是由Sun Microsystems公司于 1995年5月推出的Java面向对象程序设计语言（以下简称Java语言）和Java平台的总称。由James Gosling和同事们共同研发，并在1995年正式推出。用Java实现的HotJava浏览器（支持Java applet）显示了Java的魅力：跨平台、动态的Web、Internet计算。从此，Java被广泛接受并推动了Web的迅速发展，常用的浏览器均支持Javaapplet。另一方面，Java技术也不断更新。(2010年Oracle公司收购了SUN)

**Java组成**

Java由四方面组成：Java编程语言、Java类文件格式、Java虚拟机和Java应用程序接口平台(Java API)。

Java应用编程接口为Java应用提供了一个独立于操作系统的标准接口，可分为基本部分和扩展部分。在硬件或操作系统平台上安装一个Java平台之后，Java应用程序就可运行。Java平台已经嵌入了几乎所有的操作系统。这样Java程序可以只编译一次，就可以在各种系统中运行。常用的Java平台基于Java1.5，最近版本为Java8.0。

**Java体系**

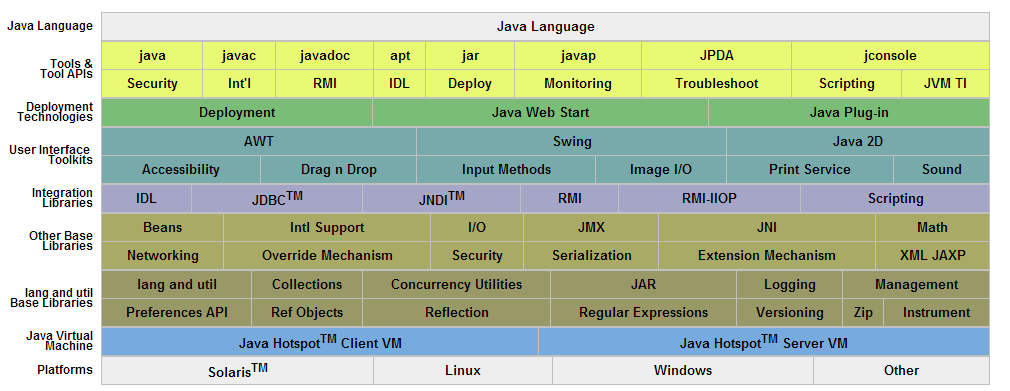
JavaSE，standard edition，标准版，是我们通常用的一个版本.

JavaEE，enterprise edition，企业版，使用这种JDK开发JavaEE应用程序.

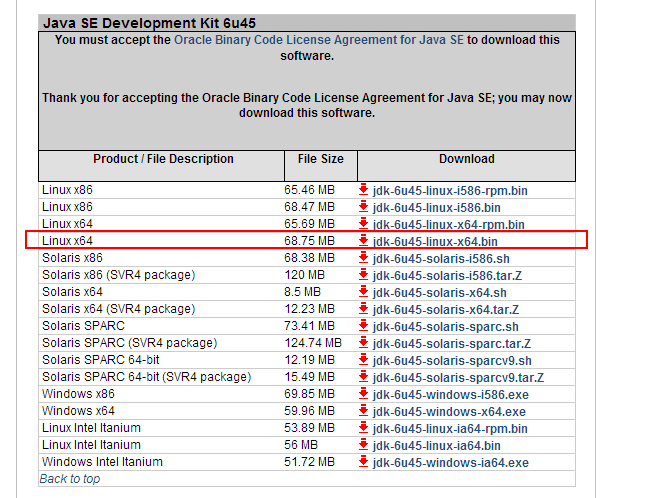
JavaME，micro edition，主要用于移动设备、嵌入式设备上的java应用程序.

**Java版本**

JavaSE Platform JDK6:

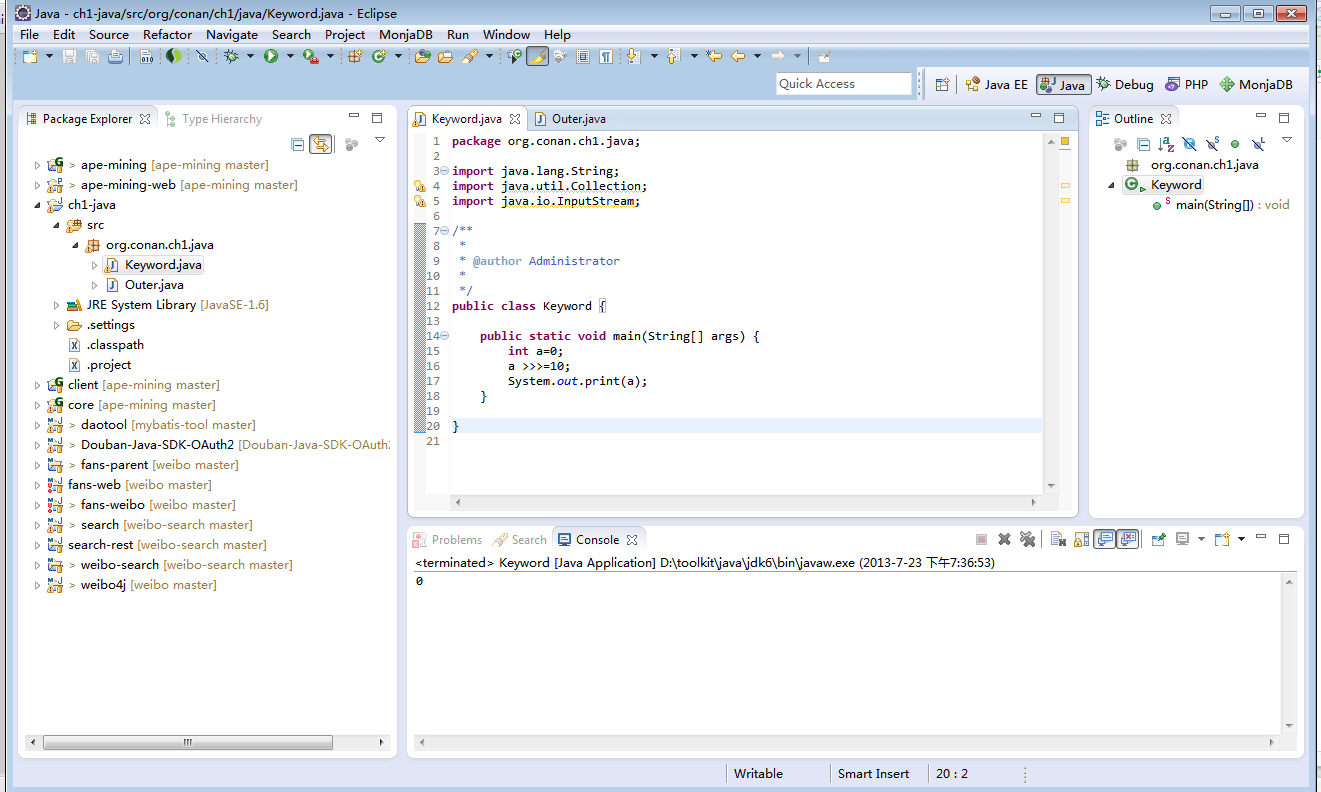
[](http://blog.fens.me/wp-content/uploads/2013/07/jdk-arch.png)

下载JDK6. http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase6-419409.html

[](http://blog.fens.me/wp-content/uploads/2013/07/jdk16.png)

**2. 开发环境**

EclipseIDE集成开发环境

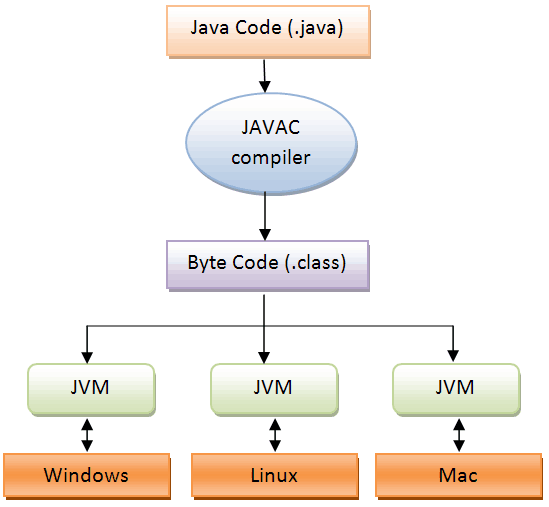
[](http://blog.fens.me/wp-content/uploads/2013/07/eclipse.png)

**3. JAVA的编译及运行环境(java,javac)**

**运行原理：**

先编译： \*.java文件 => \*.class文件(字节码)

运行： \*.classs => JVM（JAVA虚拟机）

[](http://blog.fens.me/wp-content/uploads/2013/07/java-compile.png)

**4. JAVA语法**

**1). 命名标识符**

必须以字母、美元符号或下划线开头。数字不能开头

第一个字符之后可以是任意长度的包含数字、字母、美元符号、下划线的任意组合。

不能使用Java关键字和保留字做标识符

大小写敏感的，Unicode字符会被视为普通字母对待。

A1.java, 1A.java, \_a1.java, $\_a1.java

**2). 关键字**

JavaSE6一共有50个关键字:

abstract continue for new switch

assert default goto package synchronized

boolean do if private this

break double implements protected throw

byte else import public throws

case enum instanceof return transient

catch extentds int short try

char final interface static void

class finally long strictfp volatile

const float native super while

**类和接口:**

* class:声明类
* interface：声明接口
* abstract：声明抽象类
* implements：实现接口
* extentds: 类继承
* import：包导入
* package：声明属于哪一个包
* static：静态属性，一个类下所有对象共享的属性。
* throws：声明异常
* void：空对象

**基本数据类型**

* int：整型
* long：长整型
* short：短整型
* float：单精度浮点型
* double：双精度浮点型
* boolean：布尔值true和false
* byte：字节型
* char：字符型
* enum：枚举类型

**流程控制**

* if：条件语句
* else：条件分支
* for:循环语句
* while:循环语句
* do和while：循环语句
* break:跳出循环语句
* continue:只跳出本次循环，还要继续执行下一次的循环
* return：返回
* switch：多条件语句
* case：多条件分支
* default：默认值

**访问范围**

* private：声明私有类型，只能供当前类内部访问
* 默认属性：不加任何访问权限修饰符，可以被同包的类访问
* protected：声明保护类型，可以被同包的类和不同包的子类访问
* public：声明公共类型，可以供所有其它类访问

**类的实例**

* new：创建一个新的对象实例
* this：引用相前对象实例
* super：引用超类对象实例

**异常处理**

* try：开始捕获错误的语句
* catch：捕获错误的语句
* finally：无论有没有异常发生都执行代码

**很少会用到的**

* native是方法修饰符。调用其他语言的JNI接口
* transient：变量修饰符。对象存储时，变量状态不会被持久化
* volatile：修饰变量。在每次被线程访问时，都强迫从共享内存中重读该成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存
* strictfp修饰类和方法，意思是FP-strict，精确浮点，符合IEEE-754规范的。当一个class或interface用strictfp声明，内部所有的float和double表达式都会成为strictfp的。接口的method不能被声明为strictfp的，class的可以
* goto:被保留的关键字

**3). 变量**

所谓变量，就是值可以被改变的量。

public class Variable{

public static void main(String[] args){

String myName ="nabula";

myName = "nebulayao";

System.out.println(myName);

}

}

**4). 常量**

所谓常量，值不允许改变的量。要声明一个常量用final 修饰，常量按照Java命名规范需要用全部大写，单词之间用下划线隔开：

// 游戏方向设定 北 南 东 西

final int NORTH = 1;

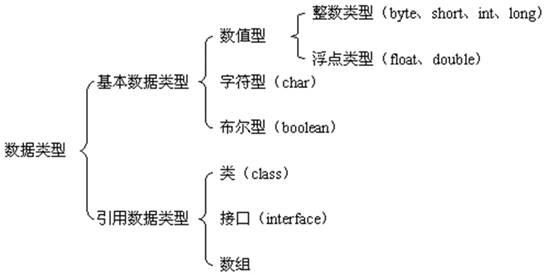
final int SOUTH = 2;

final int EAST = 3;

final int WEST = 4;

**5). 数据类型**

* 分为基本数据类型(Primitive Type)
* 引用数据类型(Reference Data Type)

[](http://blog.fens.me/wp-content/uploads/2013/07/datatype.jpg)

**6). 赋值**

“=” 是个赋值运算符，都是把右侧的值赋予左侧的变量。

public class Assignment{

public static void main(String[] args){

int a=1;

int b=1,c=1;

int d=a=b=c=2;

System.out.println("a="+a+" b="+b +" c="+c+" d="+d);

}

}

**7). 注释**

程序中的注释是程序设计者与程序阅读者之间沟通的手段，是写给程序员看的代码。通常情况下编译器会忽略注释部分，不做语法检查。

Java中的注释有三种：

* // 注释一行
* /\* 注释若干行 \*/
* /\*\*注释若干行，并写入 javadoc 文档 \*/

**8). 运算符**

**算术运算符**

+(加), –(减), \*(乘), /(除), %(求余), ++(递增), –-(递减)

**逻辑运算符**

逻辑运算符只对布尔型操作数进行运算并返回一个布尔型数据。

6个逻辑运算符：&& , || ,& , | ,！, ^

**位运算符**

位运算符是对整数操作数以二进制的每一位进行操作，返回结果也是一个整数。

逻辑位运算符：按位取反~, 按位与&, 按位或 |, 按位异或 ^,移位运算符有 左移<< , 右移>>

**赋值运算符**

运算符 举例 等效表达式

+= a += b a = a+b

-= a -= b a = a-b

\*= a \*= b a = a\*b

/= a /= b a = a/b

%= a %= b a = a%b

&= a &= b a = a&b

|= a |= b a = a|b

^= a ^= b a = a^b

<<= a <<= b a = a<<b

>>= a >>= b a = a>>b

>>>= a >>>= b a = a>>>b

**关系运算符**

运算符 功能 举例 运算结果 可运算类型

> 大于 'a'>'b' false 整数、浮点数、字符

< 小于 2<3.0 true 整数、浮点数、字符

== 等于 'x'==88 true 任意

!= 不等于 true!=true false 任意

>= 大于等于 6.6>=8.9 false 整数、浮点数、字符

<= 小于等于 'M'<=88 true 整数、浮点数、字符

**相等性运算符**  
== 和 != 这两个关系运算符比较的是两个相似事物的是否相等。

对于整数和浮点数之间的相等性如果值相等就相等。  
对于引用类型的变量之间，看他们是否引用了同一个对象，引用地址相等就相等。

**instanceof**  
instanceof 运算符只值能用于比较对象引用变量，可以检查对象是否是某种类型。这里的类型是指类、接口类型数组

**三目运算符**  
对于条件表达式b?x:y，先计算条件b，然后进行判断。如果b的值为true，计算x的值，运算结果为x的值；否则，计算y的值，运算结果为y的值。

**运算符优先级**

1. () [] .
2. ! +(正) -(负) ~ ++ –
3. \* / %
4. +(加) -(减)
5. << >> >>>
6. < <= > >= instanceof
7. == !=
8. &(按位与)
9. ^
10. |
11. &&
12. ||
13. ?:
14. = += -= \*= /= %= &= |= ^= ~= <<= >>= >>>=

**9). 流程控制**  
**if-else分支控制语句**  
if后的小括号不能省略，括号里表达式的值最终必须返回的是布尔值  
如果条件体内只有一条语句需要执行，那么if后面的大括号可以省略  
对于给定的if，else语句是可选的，else if 语句也是可选的  
else和else if同时出现时，else必须出现在else if之后  
如果有多条else if语句同时出现，那么如果有一条else if语句的表达式测试成功，那么会忽略掉其他所有else if和else分支。  
如果出现多个if,只有一个else的情形，else子句归属于最内层的if语句

if(){}  
if(){}else{}  
if(){}else if(){}  
if(){}else if(){}else{}

**switch分支控制语句**  
switch表达式必须能被求值成char byte short int 或 enum  
case常量必须是编译时解析的常量

switch(){case}  
switch(){case,default}  
switch(){case,break,default}

**while循环**  
当条件为真时执行while循环，一直到条件为假时再退出循环体。while括号后的表达式，要求和if语句一样是一个布尔值，判断是否进入循环的条件。

**do-while循环**  
进行循环之前，先执行一次代码。其他操作同while循环。

**for循环**  
当知道可以循环多少次时，是使用for循环的最佳时机。

for循环的三个部分任意部分都可以省略，最简单的for循环就是这样的 for(;;){ }  
中间的条件表达式必须返回一个布尔值，用来作为是否进行循环的判断依据  
初始化语句可以由初始化多个变量，多个变量之间可以用逗号隔开，在for循环中声明的变量作用范围就只在for循环内部  
最后的迭代语句一般是i++,j++ 这样的表达式。

**for-each循环**  
for-each循环又叫增强型for循环，它用来遍历数组和集合中的元素

for(type var : arr) {}

**跳出循环 break 、continue**  
break:用来终止循环或switch语句  
continue:用来终止循环的当前迭代  
当存在多层循环时，不带标签的break和continue只能终止离它所在的最内层循环

**10). 面向对象**  
**类和对象的概念**  
人类自古就喜欢听故事，也喜欢写故事，我们从小也被要求写作文，为了帮助你写作文。老师还总结了一些规律，譬如记叙文六要素：时间、地点、人物、起因、经过、结果。 有了这样指导性的东西，我们写作文的时候就简单了许多。

面向对象程序语言的核心思想就是把一个事物的状态和行为封装起来作为一个整体看待。类描述的就是对象知道知道什么和执行什么。

举例一架飞机：  
以顾客角度看飞机，名字波音777，座位数380人，飞行速度940公里每小时，行为飞行从A地送到B地。  
以航空公司角度看飞机，名字波音777，资产编号HNHK20100321，购买价格18.7亿人民币，行为能赚钱。

我们从不同角度去看待和抽象同一架飞机它的状态和行为不相同。

**类和对象的关系**  
类是对象的骨架，对象是根据骨架建造出来的实例。

举例我们设计一个模拟WOW的格斗游戏  
人和怪兽战斗，战斗需要武器。那么圣骑士就是个"类"，人类圣骑士"锦马超"就是一个对象。双手剑件是个类，那么拿在“锦马超”手里的“霜之哀伤”就是一个对象。

举例我们要建立一个全班同学的通讯录  
设计一个通讯录的格式，包括姓名、性别、手机号、QQ号、宿舍号。然后我们按照一定的格式印出来，交由每个同学填写，那么每个同学填写的那一份就叫对象，我们填写的通讯录格式本身就是类。

**定义类创建对象**  
定义一个类的步骤是：定义类名，编写类的属性（状态），编写类的方法（行为）

public class Dog {

private int size;// 定义了狗的大小的属性

public void setSize(int size) {// 定义设置大小的方法

if (size > 0 && size < 10) {

this.size = size;

} else {

size = 1;

}

}

public int getSize() {// 定义获取大小的方法

return size;

}

public void bark(){ // 定义狗叫的方法

if(size<5){

System.out.println("汪汪汪!");

}else{

System.out.println("嗷!嗷!");

}

}

public static void main(String[] args) {//定义main方法

Dog xiaoHang = new Dog();//创建了名字叫小黄的狗对象

xiaoHang.setSize(3);//设置它的大小属性

xiaoHang.bark(); //调用它的叫方法

Dog daHang = new Dog();//创建了名字叫大黄的狗对象

daHang.setSize(7); //设置它的大小属性

daHang.bark();//调用它的叫方法

}

}

**面向对象的三大特性(封装、继承、多态)**  
**封装(encapsulation)**：就是把属性私有化，提供公共方法访问私有对象。 有两层意思，第一隐藏数据，第二把数据和对数据操作的方法绑定。

1 修改属性的可见性来限制对属性的访问。  
2 为每个属性创建一对赋值方法和取值方法，用于对这些属性的访问。  
3 在赋值和取值方法中，加入对属性的存取限制。

封装的优点：  
1 隐藏类的实现细节；  
2 可加入控制逻辑，限制对属性的不合理操作；  
3 便于修改，增强代码的可维护性；

**继承(inheritance)**：同类事物之间有它的共同性也有各自的独特性，把共同的部分抽离出来作为父类，保留自己特有的属性和方法做为子类。通过继承让子类也具有父类的属性和方法。

举例：马儿都有四条腿，马儿都有会跑  
这些共同性抽象出来就成了马类；其中有一些马是白色的马，还有一些是黑色的马，就成了白马类和黑马类。那么白马类和马类之间的关系就是继承关系。它们是父子关系，马类是父类、白马类是子类。  
继承的优点：

简化了人们对事物的认识和描述，清晰的体现了相关类间的层次关系。  
功能抽象、继承促进了代码复用、继承也带来了多态性。

**多态(polymorphism)**：多态就是“一种定义、多种实现” 。  
当父类对象引用变量引用子类对象时，被引用对象的类型决定了调用的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。

public abstract class Vehicle {

abstract void go(Address address);

}

public class Car extends Vehicle{

@Override public void go(Address address){

System.out.println("Car to " + address.getName());

}

}

public class Plane extends Vehicle{

@Override void go(Address address) {

System.out.println("Plane to " + address.getName());

}

}

//多态，父类引用指向子类对象，实际传过来的是抽象类Vehicle的子类，然后编译器会根据具体实现类，来找实现方法。

public void drive(Vehicle v){

　　 v.go(new Address("杭州(abstract)"));

}

**多态的优点**

* 可替换性。多态对已存在的代码具有可替换性。
* 可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。
* 接口性。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的
* 灵活性。它在应用中体现了灵活多样的操作，提高了使用效率。
* 简化性。多态简化了对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。值得注意的是，多态并不能够解决提高执行速度的问题，因为它基于动态装载和地址引用，或称动态绑定。

**11). 包：类的组织和管理方式**  
包的用途有以下三种：

* 将功能相近的类放在同一个包里，方便使用和查找
* 类名相同的文件可以放在不同的包里而不会产生冲突
* 可以依据包设定访问权限

**12). 方法：类或对象的行为**  
**方法的声明**  
修饰符，可选，用于指定谁有权限访问此方法。  
返回值类型，必选，用于指定该方法的返回值数据类型；如果该方法没有返回值，则要用关键字 void 进行标示。方法的返回值只能有一个。  
参数列表，可以有0到多个，多个参数之间要用逗号隔开，参数的写法形如：String[] args, int age 这样。  
方法名，必选，这个……，好吧命名规则是方法名和变量名的首字母要小写，别丢我人，弄个大写方法名出来。  
方法体，可选，这个……，  
大括号，大括号不写的方法叫抽象方法。

public void go(int size) throws Exception;

**方法重写override**  
当子类继承父类时，如果子类方法的签名和父类方法的签名相同时，子类的方法就覆盖了父类的方法，我们称之为重写。  
举例：动物会喝水，但是猫喝水和人喝水的具体行为就不同。

重写规则：  
参数列表必须与重写的方法的参数列表完全匹配（方法签名相同）。如果不匹配，你得到的将是方法重载。  
返回类型必须与超类中被重写方法中原先声明的返回类型或其子类型相同。  
访问级别的限制性可以比被重写方法弱，但是访问级别的限制性一定不能比被重写方法的更严格。  
仅当实例方法被子类继承时，它们才能被重写。子类和超类在同一个包内时，子类可以重写未标示为private和final的任何超类方法。不同包的子类只能重写标示为public或protected的非final方法。  
无论父类的方法是否抛出某种运行时异常，子类的重写方法都可以抛出任意类型的运行时异常。  
重写方法一定不能抛出比被重写方法声明的检验异常更新或更广的检验异常，可以抛出更少或更有限的异常。  
不能重写标示为final的方法。  
不能重写标示为static的方法。  
如果方法不能被继承，那么方法不能被重写。

**方法重载overload**  
在Java 中允许类定义中多个方法的方法名相同，只要它们的参数声明不同即可，这种方法就被称为重载（overloaded)。如狗的叫声，参数不同叫声也不同。

重载规则：  
重载方法必须改变方法参数列表  
重载方法可以改变返回类型  
重载方法可以改变访问修饰符  
重载方法可以声明新的或更广的检验异常  
方法能够在同一个类或者一个子类中被重载

**13) main方法:**  
main()方法是Java应用程序的入口方法。这个方法和其他的方法有很大的不同，比如方法的名字必须是main，方法必须是public static void 类型的，方法必须接收一个字符串数组的参数等等。

public staitc void main(String[] args){}

**14) 构造函数**  
在Java中，对象是构造出来的，用new关键字来标示这个创建的过程。  
构造函数或者说构造方法不是方法。它们之间的三大区别：

构造函数没有返回值  
构造函数名称和类名相同  
构造函数不能用final,static,abstract

**15). 接口**  
**为什么要有接口**  
Java中只支持单继承，只能有一个父类。为了提供类似多重继承的功能，Java提供了接口的功能。

接口的几个规则  
接口名用 interface 修饰, 相对应的 类名用 class修饰  
接口里定义的方法都是抽象方法，可以用abstract修饰，当然也可以不用它修饰  
接口只能被实现 ( implements )  
可以用抽象类实现接口，也就是说虽然实现了，但是没有真正写接口的任何方法，它把责任推给了抽象类的子类  
普通类实现接口，则必须按照接口的契约，实现接口所定义的所有方法。  
接口可以继承接口，或者说一个接口可以是另一个接口的父类  
一个接口可以继承多个父类，也就是说一个接口之间可以多重继承。

**16). 抽象类和抽象方法**  
用abstract 修饰的类定义，我们称之为抽象类，抽象类不能被实例化。  
用abstract 修饰的方法，我们称之为抽象方法，抽象方法不能有方法体。

面向对象中，所有的对象都是某一个类的实例，但是并不是每个类都可以实例化成一个对象。如果一个类中没有足够的信息来描绘一个具体的对象，那么这个类就不能被实例化，我们称之为抽象类。  
抽象类实际上是定义了一个标准和规范，等着他的子类们去实现，譬如动物这个抽象类里定义了一个发出声音的抽象方法，它就定义了一个规则，那就是谁要是动物类的子类，谁就要去实现这个抽象方法。

**17). 初始化块**  
我们已经知道在类中有两个位置可以放置执行操作的代码，这两个位置是方法和构造函数。初始化块是第三个可以放置执行操作的位置。当首次加载类（静态初始化块）或者创建一个实例（实例初始化块）时，就会运行初始化块。

初始化块的语法相当简单，它没有名称，没有参数，也没有返回值，只有一个大括号，用static 修饰的初始化块就要静态初始化块。  
静态初始化块在首次加载类时会运行一次。  
实例初始化块在每次创建对象时会运行一次。  
实例初始化块在构造函数的super()调用之后运行。  
初始化块之间的运行顺序取决于他们在类文件中出现的顺序，出现在前面的先执行。  
初始化块从书写惯例上应该写在靠近类文件的顶部，构造函数附近的某个位置。

**18). 内部类**  
**普通内部类**  
所谓内部类，就是嵌套在类中的类。被嵌套的类叫外部类。

public class Outer {

public class Inner{

}

}

内部类就像一个实例成员一样存在于外部类中。  
内部类可以访问外部类的所有成员就想访问自己的成员一样没有限制。  
内部类中的this指的是内部类的实例对象本身，如果要用外部类的实例对象就可以用类名.this的方式获得。  
内部类对象中不能有静态成员，原因很简单，内部类的实例对象是外部类实例对象的一个成员。

内部类的创建方法：

在外部类的内部，可以用 Inner inner = new Inner(); 方法直接创建  
在外部类外部，必须先创建外部类实例，然后再创建内部类实例。  
Outer outer = new Outer();  
Inner inner = outer.new Inner();

**静态内部类**  
和普通的类一样，内部类也可以有静态的。不过和非静态内部类相比，区别就在于静态内部类没有了指向外部的引用。除此之外，在任何非静态内部类中，都不能有静态数据，静态方法或者又一个静态内部类（内部类的嵌套可以不止一层）。不过静态内部类中却可以拥有这一切。这也算是两者的第二个区别吧。

public class Outer {

public static class Inner{

}

}

**局部内部类**  
们也可以把类定义在方法内部，这时候我们称这个类叫局部内部类。

局部内部类的地位和方法内的局部变量的位置类似，因此不能修饰局部变量的修饰符也不能修饰局部内部类，譬如public、private、protected、static、transient等  
局部内部类只能在声明的方法内是可见的，因此定义局部内部类之后，想用的话就要在方法内直接实例化，记住这里顺序不能反了，一定是要先声明后使用，否则编译器会说找不到。  
局部内部类不能访问定义它的方法内的局部变量，除非这个变量被定义为final 。

interface World {

String sayHello();

}

public class Outer {

public class Inner {

public World hello() {

class World2 implements World {

private String name;

@Override

public String sayHello() {

return "hello " + name;

}

}

return new World2();

}

}

}

**匿名内部类**  
当我们把内部类的定义和声明写到一起时，就不用给这个类起个类名而是直接使用了，这种形式的内部类根本就没有类名，因此我们叫它匿名内部类。  
匿名内部类可以是某个类的继承子类也可以是某个接口的实现类。

interface World {

String sayHello();

}

public class Outer {

public class Inner {

public World hello() {

return new World() {

private String name;

@Override

public String sayHello() {

return "hello " + name;

}

};

}

}

}

**19). 数组**  
数组是有序数据的集合，数组中的每个元素具有相同的数组名和下标来唯一地确定数组中的元素。

**一维数据**

int[] number1;

int number2[];

Java中使用关键字new创建数组对象，格式为：

int[] number1 = new int[5];

**多维数据**

String[][] s1; //二维数组

String[][][][] s2; //四维数组

String[] s3[]; //怪异写法的二维数组

**构建数组 | 创建数组 | 实例化数组**  
构建数组意味着在堆上创建数组对象（所有的对象都存储在堆上，堆是一种内存存储结构，既然要存储就设计空间分配问题，因此此时需要指定数组的大小）。而此时虽然有了数组对象，但数组对象里还没有值。

int[] scores; //声明数组  
scores = new int[34]; //创建数组  
int[] i = new int[22]; //声明并创建数组  
声明数组时如果没写长度，只是定义了一个变量，只没有分配空间。

int[][] xy= new int[2][3]; //声明并创建二维数组  
int[][] mn= new int[2][]; //声明并创建二维数组,只创建第一级数组也是可以的。

**初始化数组 | 给数组赋值**  
初始化数组就是把内容放在数组中。数组中的内容就是数组的元素。他们可以是基本数据类型也可以是引用数据类型。如同引用类型的变量中保存的是指向对象的引用而不是对象本身一样。数组中保存的也是对象的引用而不是对象本身。

Pig[] pigs = new Pig[3]; //声明并创建猪数组  
pigs[0] = new Pig(); //给每一个元素赋值，创建了三个猪对象，此时数组里才真正有了对象  
pigs[1] = new Pig(); //数组用下标来赋值和访问，下标写在[]中，数组下标最大是声明数量减1  
pigs[2] = new Pig();

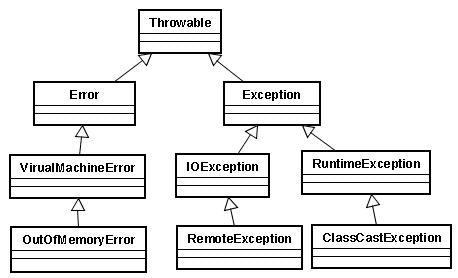
int[] numbers;  
numbers=new int[]{0,1,2,3,4,5,6,7,8,9}; //创建匿名数组并赋值  
int[][] xy= new int[][]{{2,3},{4,5},{5,6}}; //创建二维匿名数组并赋值

**20). 异常**  
**异常及异常的分类**  
异常是指在程序中出现的异常状况，在Java中异常被抽象成一个叫做Throwable的类。

其中如果程序出错并不是由程序本身引起的，而是硬件等其他原因引起的，我们称之为Error，一般情况下Error一旦产生，对程序来说都是致命的错误，程序本身无能为力，所以我们可以不对Error作出任何处理和响应。

异常如果是由程序引起的我们称之为Exception，Exception又分两种，我们把运行时才会出现的异常叫做 RuntimeException，RuntimeException我们不好在程序编写阶段加以事先处理，而其他异常则可以在程序编写和编译阶段加以事先检查和处理，我们把这种异常叫做检验异常。

**程序只需要捕捉和处理检验异常**  
相应的我们把除检验异常(可检查异常)之外的异常称之为非检验异常，包括Error和RuntimeException ，非检验异常可以捕捉也可以不捕捉，更多的时候我们不捕捉，因为捕捉了我们也没办法处理，譬如程序运行时发生了一个VirtualMachineError异常，虚拟机都出错了，作为运行在虚拟机内的程序又有什么办法处理呢？

[](http://blog.fens.me/wp-content/uploads/2013/07/exception.jpg)

**异常的处理**  
try{}catch(){}finally{}

**异常处理的几条规则**  
try用于定义可能发生异常的代码段，这个代码块被称为监视区域，所有可能出现检验异常的代码写在这里。  
catch代码段紧跟在try代码段后面，中间不能有任何其他代码。  
try后面可以没catch代码段，这实际上是放弃了捕捉异常，把异常捕捉的任务交给调用栈的上一层代码。  
try后面可以有一个或者多个catch代码段，如果有多个catch代码段那么程序只会进入其中某一个catch。  
catch捕捉的多个异常之间有继承关系的话，要先捕捉子类后捕捉父类。  
finally代码段可以要也可以不要。  
如果try代码段没有产生异常，那么finally代码段会被立即执行，如果产生了异常，那么finally代码段会在catch代码段执行完成后立即执行。  
可以只有try和finally没有catch。

**5. JDK基本包介绍**

JDK（Java Development Kit）是Sun Microsystems针对Java开发员的产品。自从Java推出以来，JDK已经成为使用最广泛的Java SDK。JDK 是整个Java的核心，包括了Java运行环境、Java工具和Java基础类库。JDK是学好Java的第一步。

**1). 基础包**

* java.lang： 这个是系统的基础类，比如String等都是这里面的，这个包是唯一一个可以不用引入(import)就可以使用的包
* java.io: 这里面是所有输入输出有关的类，比如文件操作等
* java.nio;为了完善io包中的功能，提高io包中性能而写的一个新包 ，例如NIO非堵塞应用
* java.net: 这里面是与网络有关的类，比如URL，URLConnection等。
* java.util: 这个是系统辅助类，特别是集合类Collection，List，Map等。
* java.sql: 这个是数据库操作的类，Connection， Statement，ResultSet等
* javax.servlet: 这个是JSP，Servlet等使用到的类

**2). 字符串(java.lang.String)**  
程序开发的工作中80%的操作都和字符串有关。

public final class String implements java.io.Serializable, Comparable, CharSequence{}

String永远不可能有子类，它的实例也是无法改变的。

**创建字符串对象**

String s1 = new String("Milestone");

String s2 = "String";

第一种是常规写法，创建一个对象当然就可以用new跟上个构造函数完成。第二种写法，最常用，效率也高。  
字符串操作中的加号

我们经常要把两个或者更多的字符串拼接成一个字符串，除了普通的连接字符串的方法以外，Java语言专门为String提供了一个字符串连接符号“+”

String s3 = "a"+"b";

**字符串中的常用方法**

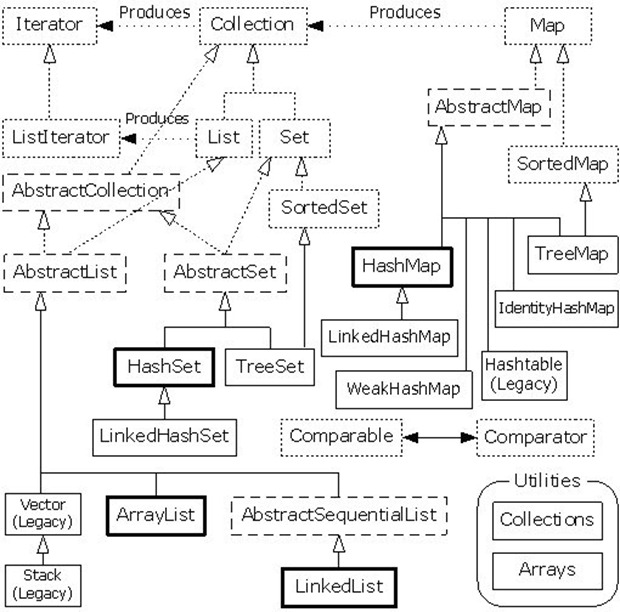
* charAt() 返回位于指定索引处的字符串
* concat() 将一个字符串追加到另一个字符串的末尾
* equalseIgnoseCase() 判断两个字符串的相等性，忽略大小写
* length() 返回字符串中的字符个数
* replace() 用新字符代替指定的字符
* substring() 返回字符串的一部分
* toLowerCase() 将字符串中的大写字符转换成小写字符返回
* toString() 返回字符串的值
* toUpperCase() 将字符串中的小写字符转换成大写字符返回
* trim() 删除字符串前后的空格
* splite() 将字符串按照指定的规则拆分成字符串数组

**3). 集合(java.util.Collection)**  
colection 集合，用来表示任何一种数据结构 Collection 集合接口，指的是 java.util.Collection接口，是 Set、List 和 Queue 接口的超类接口 Collections 集合工具类，指的是 java.util.Collections 类。

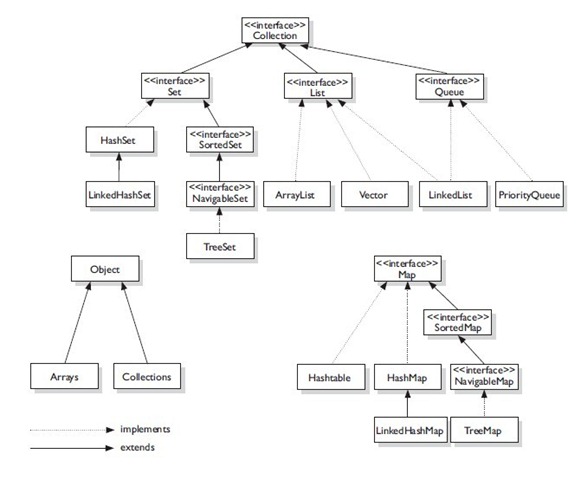
**集合类和集合接口**

Collection , Set , SortedSet , List , Map , SortedMap , Queue , NavigableSet , NavigableMap, Iterator, HashMap , Hashtable ,TreeMap , LinkedHashMap , HashSet , LinkedHashSet ,TreeSet , ArrayList , Vector , LinkedList , PriorityQueuee , Collections , Arrays

* + - List 关注事物的索引列表
    - Set 关注事物的唯一性
    - Queue 关注事物被处理时的顺序
    - Map 关注事物的映射和键值的唯一性

[](http://blog.fens.me/wp-content/uploads/2013/07/collection.jpg)

上图中加粗线的ArrayList 和 HashMap 是我们重点讲解的对象。下面这张图看起来层级结构更清晰些。

[](http://blog.fens.me/wp-content/uploads/2013/07/collection2.jpg)

**Collection 接口**  
Collection接口是 Set 、List 和 Queue 接口的父接口，提供了多数集合常用的方法声明，包括 add()、remove()、contains() 、size() 、iterator() 等。

* + - add(E e) 将指定对象添加到集合中
    - remove(Object o) 将指定的对象从集合中移除，移除成功返回true,不成功返回false
    - contains(Object o) 查看该集合中是否包含指定的对象，包含返回true,不包含返回flase
    - size() 返回集合中存放的对象的个数。返回值为int
    - clear() 移除该集合中的所有对象，清空该集合。
    - iterator() 返回一个包含所有对象的iterator对象，用来循环遍历
    - toArray() 返回一个包含所有对象的数组,类型是Object
    - toArray(T[] t) 返回一个包含所有对象的指定类型的数组

List接口

List 关心的是索引，与其他集合相比，List特有的就是和索引相关的一些方法：get(int index) 、 add(int index,Object o) 、 indexOf(Object o) 。  
ArrayList 可以将它理解成一个可增长的数组，它提供快速迭代和快速随机访问的能力。  
LinkedList 中的元素之间是双链接的，当需要快速插入和删除时LinkedList成为List中的不二选择。  
Vector 是ArrayList的线程安全版本，性能比ArrayList要低，现在已经很少使用

Set接口

Set关心唯一性，它不允许重复。  
HashSet 当不希望集合中有重复值，并且不关心元素之间的顺序时可以使用此类。  
LinkedHashset 当不希望集合中有重复值，并且希望按照元素的插入顺序进行迭代遍历时可采用此类。  
TreeSet 当不希望集合中有重复值，并且希望按照元素的自然顺序进行排序时可以采用此类。（自然顺序意思是某种和插入顺序无关，而是和元素本身的内容和特质有关的排序方式，譬如“abc”排在“abd”前面。）

Queue接口

Queue用于保存将要执行的任务列表。  
LinkedList 同样实现了Queue接口，可以实现先进先出的队列。  
PriorityQueue 用来创建自然排序的优先级队列。

Map接口

Map关心的是唯一的标识符。他将唯一的键映射到某个元素。当然键和值都是对象。  
HashMap 当需要键值对表示，又不关心顺序时可采用HashMap。  
Hashtable 注意Hashtable中的t是小写的，它是HashMap的线程安全版本，现在已经很少使用。  
LinkedHashMap 当需要键值对，并且关心插入顺序时可采用它。  
TreeMap 当需要键值对，并关心元素的自然排序时可采用它。

**ArrayList的使用**  
ArrayList是一个可变长的数组实现，读取效率很高，是最常用的集合类型。

在Java5版本之前我们使用：  
List list = new ArrayList();

在Java5版本之后，我们使用带泛型的写法：  
List list = new ArrayList();

上面的代码定义了一个只允许保存字符串的列表，尖括号括住的类型就是参数类型，也成泛型。带泛型的写法给了我们一个类型安全的集合。

**Map的接口**  
Map接口的常用方法如下表所示：

* + - put(K key, V value) 向集合中添加指定的键值对
    - putAll(Map t) 把一个Map中的所有键值对添加到该集合
    - containsKey(Object key) 如果包含该键，则返回true
    - containsValue(Object value) 如果包含该值，则返回true
    - get(Object key) 根据键,返回相应的值对象
    - keySet() 将该集合中的所有键以Set集合形式返回
    - values() 将该集合中所有的值以Collection形式返回
    - remove(Object key) 如果存在指定的键，则移除该键值对，返回键所对应的值，如果不存在则返回null
    - clear() 移除Map中的所有键值对，或者说就是清空集合
    - isEmpty() 查看Map中是否存在键值对
    - size() 查看集合中包含键值对的个数，返回int类型

因为Map中的键必须是唯一的，所以虽然键可以是null，只能由一个键是null，而Map中的值可没有这种限制，值为null的情况经常出现，因此get(Object key)方法返回null,有两种情况一种是确实不存在该键值对，二是该键对应的值对象为null。为了确保某Map中确实有某个键，应该使用的方法是 containsKey(Object key) 。

HashMap使用  
HashMap是最常用的Map集合，它的键值对在存储时要根据键的哈希码来确定值放在哪里。

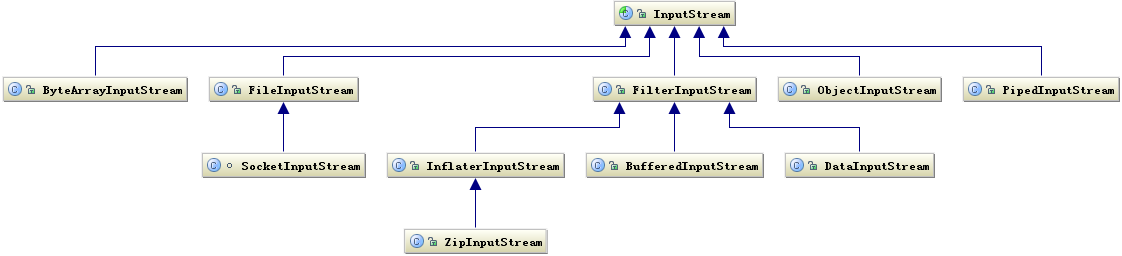
在Java5版本之前我们使用：  
Map map = new HashMap();

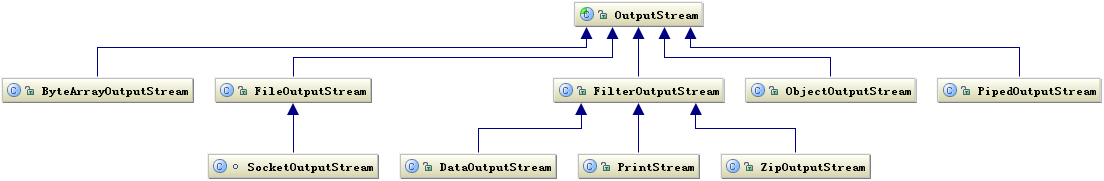
在Java5版本之后，我们使用带泛型的写法：  
Map map = new HashMap();

**4). IO(java.io)**  
Java 的 I/O 操作类在包 java.io 下，大概有将近 80 个类，但是这些类大概可以分成四组，分别是：

* + - 基于字节操作的 I/O 接口：InputStream 和 OutputStream
    - 基于字符操作的 I/O 接口：Writer 和 Reader
    - 基于磁盘操作的 I/O 接口：File
    - 基于网络操作的 I/O 接口：Socket

前两组主要是根据传输数据的数据格式，后两组主要是根据传输数据的方式，虽然 Socket 类并不在 java.io 包下，但是我仍然把它们划分在一起，因为我个人认为 I/O 的核心问题要么是数据格式影响 I/O 操作，要么是传输方式影响 I/O 操作，也就是将什么样的数据写到什么地方的问题，I/O 只是人与机器或者机器与机器交互的手段，除了在它们能够完成这个交互功能外，我们关注的就是如何提高它的运行效率了，而数据格式和传输方式是影响效率最关键的因素了。

**输入**  
[](http://blog.fens.me/wp-content/uploads/2013/07/input.png)

**输出**  
[](http://blog.fens.me/wp-content/uploads/2013/07/output.png)

**同步与异步**  
所谓同步就是一个任务的完成需要依赖另外一个任务时，只有等待被依赖的任务完成后，依赖的任务才能算完成，这是一种可靠的任务序列。要么成功都成功，失败都失败，两个任务的状态可以保持一致。而异步是不需要等待被依赖的任务完成，只是通知被依赖的任务要完成什么工作，依赖的任务也立即执行，只要自己完成了整个任务就算完成了。至于被依赖的任务最终是否真正完成，依赖它的任务无法确定，所以它是不可靠的任务序列。我们可以用打电话和发短信来很好的比喻同步与异步操作。

**阻塞与非阻塞**  
阻塞与非阻塞主要是从 CPU 的消耗上来说的，阻塞就是 CPU 停下来等待一个慢的操作完成 CPU 才接着完成其它的事。非阻塞就是在这个慢的操作在执行时 CPU 去干其它别的事，等这个慢的操作完成时，CPU 再接着完成后续的操作。虽然表面上看非阻塞的方式可以明显的提高 CPU 的利用率，但是也带了另外一种后果就是系统的线程切换增加。增加的 CPU 使用时间能不能补偿系统的切换成本需要好好评估。

**两种的方式的组合**  
分别是：同步阻塞、同步非阻塞、异步阻塞、异步非阻塞，这四种方式都对 I/O 性能有影响。下面给出分析，并有一些常用的设计用例参考。

* + - 同步阻塞:最常用的一种用法，使用也是最简单的，但是 I/O 性能一般很差，CPU 大部分在空闲状态。
    - 同步非阻塞:提升 I/O 性能的常用手段，就是将 I/O 的阻塞改成非阻塞方式，尤其在网络 I/O 是长连接，同时传输数据也不是很多的情况下，提升性能非常有效。这种方式通常能提升 I/O 性能，但是会增加 CPU 消耗，要考虑增加的 I/O 性能能不能补偿 CPU 的消耗，也就是系统的瓶颈是在 I/O 还是在 CPU 上。
    - 异步阻塞:这种方式在分布式数据库中经常用到，例如在网一个分布式数据库中写一条记录，通常会有一份是同步阻塞的记录，而还有两至三份是备份记录会写到其它机器上，这些备份记录通常都是采用异步阻塞的方式写 I/O。异步阻塞对网络 I/O 能够提升效率，尤其像上面这种同时写多份相同数据的情况。
    - 异步非阻塞:这种组合方式用起来比较复杂，只有在一些非常复杂的分布式情况下使用，像集群之间的消息同步机制一般用这种 I/O 组合方式。如 Cassandra 的 Gossip 通信机制就是采用异步非阻塞的方式。它适合同时要传多份相同的数据到集群中不同的机器，同时数据的传输量虽然不大，但是却非常频繁。这种网络 I/O 用这个方式性能能达到最高。

**6. JAVA项目(Ant, Maven)**

**ANT**  
Apache Ant，是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具，大多用于Java环境中的软件开发。由Apache软件基金会所提供。默认情况下，它的buildfile(XML文件)名为build.xml。每一个buildfile含有一个 和至少一个默认的，这些targets包含许多task elements。每一个task element有一个用来被参考的id，此id必须是唯一的。

http://ant.apache.org/

**Maven**  
Apache Maven，是一个软件（特别是Java软件）项目管理及自动构建工具，由Apache软件基金会所提供。基于项目对象模型（缩写：POM）概念，Maven利用一个中央信息片断能管理一个项目的构建、报告和文档等步骤。曾是Jakarta项目的子项目，现为独立Apache项目。

http://maven.apache.org

# RHadoop培训 之 Linux基础课

**目录**

1. 背景知识
2. 文件系统
3. 常用命令
4. vi编辑器
5. 用户管理
6. 磁盘管理
7. 网络管理
8. 系统管理
9. 软件包管理
10. 常用软件

**1. 背景知识**

**Linux起源**  
Linux是一个诞生于网络、成长于网络且成熟于网络的奇特的操作系统。1991年，芬兰大学生Linus萌发了开发一个自由的UNIX操作系统的想法。

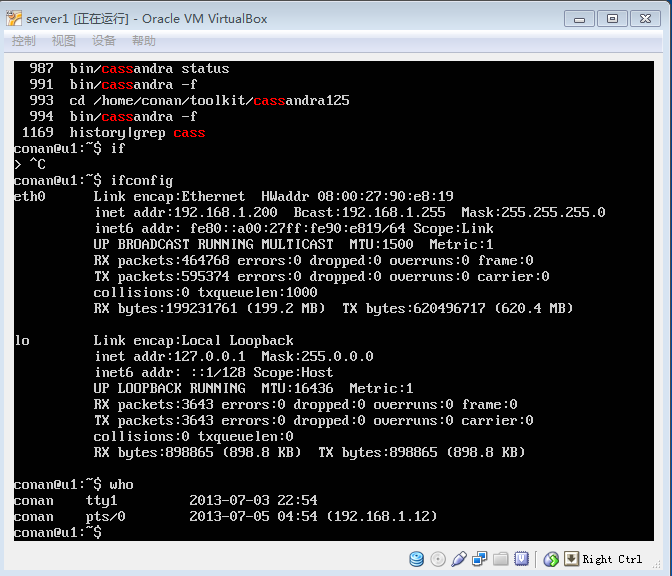
Ubuntu Linux最早于2004年作为Debian的一个分支出现，其创始人是南非企业家Mark Shuttleworth。Ubuntu项目由Shuttleworth的公司Canonical和社区志愿开发者共同努力开发而成，目的是实 现一个现代 版的Linux版本，使其在桌面系统上真正具有竞争力，更适合主流非技术用户使用。

Ubuntu的重点在于提高易用性，并且坚持定时发布新版本，即每隔六个月发布一个新版本。这确保了用户不再使用过时的软件。

<http://www.ubuntu.com/>

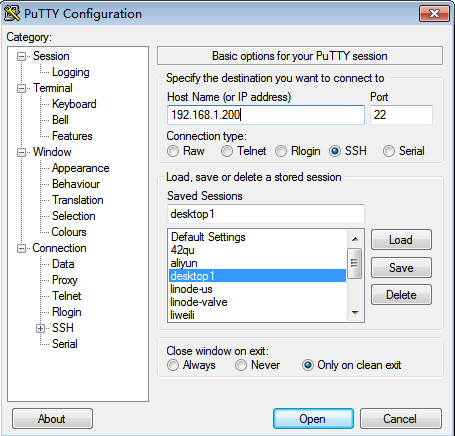
**Linux安装**

* 在物理上面安装Linux (略…)
* 在虚拟机上面安装Linux

[](http://blog.fens.me/wp-content/uploads/2013/07/vbox-linux.png)

**客户端软件**

Putty，http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[](http://blog.fens.me/wp-content/uploads/2013/07/putty.png)

**2. 文件系统**

**a.目录结构**

ls /

bin dev home lib lost+found mnt proc run selinux sys usr vmlinuz

boot etc initrd.img lib64 media opt root sbin srv tmp var

**b.常用目录介绍**

/bin 所有用户都能执行的命令

/dev 设备特殊文件

/etc 系统管理和配置文件

/home 用户主目录的基点

/sbin 系统管理员才能执行的命令

/tmp 公用的临时文件存储点

/root 系统管理员的主目录

/mnt 系统提供这个目录是让用户临时挂载其他的文件系统

/var 某些大文件的溢出区，比方说各种服务的日志文件

**c.简单了解的目录介绍**

/lib 标准程序设计库

/lib64 标准程序设计库

/lost+found 这个目录平时是空的，系统非正常关机而留下恢复的文件

/proc 正在运行的内核信息映射

/sys 硬件设备的驱动程序信息

/usr 最庞大的目录，要用到的应用程序和文件几乎都在这个目录

/usr/bin 众多的应用程序

/boot 该目录默认下存放的是Linux的启动文件和内核

/media 挂载多媒体设备

/opt 第三方软件在安装时默认会找这个目录,所以你没有安装此类软件时它是空的

**d. 文件和目录**

ls -l /etc/passwd

-rw-r--r-- 1 root root 1146 May 28 20:48 /etc/passwd

第一列：文件类型和权限

第二列：i节点，硬件连接数

第三列：用户

第四列：用户组

第五列：文件大小

第六列：最近一次修改时间mtime

第七列：文件或者目录名

上面第一列： -rw-r--r--

1列：文件类型 (-普通文件, d目录, l链接文件)

2-10列：用户权限，用户组权限，其他人权限, 读r(4)，写w(2)，执行x(1)

rw- : 4+2+0=6

r-- : 4+0+0=4

r-- : 4+0+0=4

chmod 644 /etc/passwd

**3. 常用命令**

man: 查询帮助文档

ls: 列出目前下的所有内容 (a, l)

cd: 变换工作目录(~, . , .., -)

pwd: 显示当前目录

touch: 创建一个空文件或者改变创建时间

mkdir: 创建一个目录(p)

rmdir: 删除一个空目录(p)

rm: 删除文件和目录 (rf)

cp: 复制文件和目录(rf)

mv: 移动文件和目录

cat: 显示一个文件内容

head: 显示文件前N行 (-n)

tail: 显示文件后N行(-n)

less : 分屏显示文件内容

wc: 统计计数 (l,w,m)

chmod: 给目录设置权限(R)

chown: 修改目录和文件用户或者用户组 (R)

chgrp: 修改目录和文件用户组(R)

which: 按路径查找命令

whereis: 按资料库查找命令

locate: 按文件包查询

find: 按条件查询

gzip: 压缩和解压文件(-n,d)

zcat: 查看gzip压缩文件内容

zip: 以zip压缩文件和目录 (r)

unzip: 解压zip文件

bzip2: 以bz2压缩和解压文件 (z,d)

bzcat: 查看bz2压缩文件内容

tar: 打包并压缩文件(czvf, xzvf)

shutdown: 定时关机(-h 18:00, -r重启)

halt: 关机

init: 切换运行级别(0关机)

poweroff: 关机

reboot: 重启

history: 查看命令行历史

grep: 使用正则表达式搜索文本

echo: 打印环境变量

export: 查看环境变量，或者声明变量

clear: 清空屏幕

alias: 设置别名(alias cls='clear')

locale: 当前系统语言环境

sudo: 切换成root用户运行

> : 输入到文件 (locale > a.txt)

>> : 合并到文件末尾 (locale >> a.txt)

| : 管道过滤 (cat a.txt| grep en\_US)

ln: 建立链接文件(s软链接)

sh: 执行一个shell脚本

date: 查看当前系统时间

time：统计程序消耗时间

**4. vi编辑器**

**a. vi的基本概念**  
基本上vi可分为三种操作状态，分别是命令模式(Command mode)、插入模式(Insert mode)和底线命令模式(Last line mode)，

* Comand mode：控制屏幕光标的移动，字符或光标的删除，移动复制某区段及进入Insert mode下，或者到Last line mode。
* Insert mode：唯有在Insert mode下，才可做文字数据输入，按Esc等可回到Comand mode。
* Last line mode：将储存文件或离开编辑器，也可设置编辑环境，如寻找字符串、列出行号等。

**b. 命令模式(Command mode)**

移动光标类命令:

h ：光标左移一个字符

l ：光标右移一个字符

space：光标右移一个字符

Backspace：光标左移一个字符

k或Ctrl+p：光标上移一行

j或Ctrl+n ：光标下移一行

Enter ：光标下移一行

w或W ：光标右移一个字至字首

b或B ：光标左移一个字至字首

e或E ：光标右移一个字至字尾

) ：光标移至句尾

( ：光标移至句首

}：光标移至段落开头

{：光标移至段落结尾

$：光标移至当前行尾

屏幕翻滚类命令

Ctrl+u：向文件首翻半屏

Ctrl+d：向文件尾翻半屏

Ctrl+f：向文件尾翻一屏

Ctrl+b；向文件首翻一屏

插入文本类命令

i ：在光标前

I ：在当前行首

a：光标后

A：在当前行尾

o：在当前行之下新开一行

O：在当前行之上新开一行

删除命令

do：删至行首

d$：删至行尾

ndd：删除当前行及其后n-1行

**c. 插入模式(Insert mode)**  
编辑文本信息

**d. 底线命令模式(Last line mode)**

：n1,n2 co n3：将n1行到n2行之间的内容拷贝到第n3行下

：n1,n2 m n3：将n1行到n2行之间的内容移至到第n3行下

：n1,n2 d ：将n1行到n2行之间的内容删除

：w ：保存当前文件

：e filename：打开文件filename进行编辑

：x：保存当前文件并退出

：q：退出vi

：q!：不保存文件并退出vi

：!command：执行shell命令command

：n1,n2 w!command：将文件中n1行至n2行的内容作为command的输入并执行之，若不指定n1，n2，则表示将整个文件内容作为command的输入

：r!command：将命令command的输出结果放到当前行

**5. 用户管理**

useradd：命令用来建立用户帐号和创建用户的起始目录。

passwd: 设定账号的密码

groupadd: 将新组加入系统

adduser: 增加用户到sudo

groupadd hadoop

useradd hadoop -g hadoop;

passwd hadoop

adduser hadoop sudo

mkdir /home/hadoop

chown -R hadoop:hadoop /home/hadoop

sudo: 切换到root执行命令

sudo –i: 切换到root账号

su: 切换其他用户

who: 显示系统中有哪些用户登陆系统。

whoami: 显示当前操作的用户名

ls -l /etc/passwd

-rw-r--r-- 1 root root 1146 May 28 20:48 /etc/passwd

**6. 磁盘管理**

fdisk: 硬盘分区工具(l)

partprobe: 不重启让分区生效

mkfs: 格式化硬盘(-t ext4)

mount: 挂载分区

umount: 取消挂载分区

/etc/fstab:开机自动挂载

/dev/vdb1 /home/cos/hadoop ext4 defaults 0 0

df: 查看当前硬盘的使用情况(ah)

du: 查看目录下的文件夹大小(shd)

**7. 网络管理**

**a. 网络配置**

hostname: 显示或修改主机名

/etc/hostname

u1

/etc/hosts：配置主机名与ip的映射

127.0.0.1 localhost

127.0.1.1 u1

# The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

/etc/network/interfaces: 配置网卡信息

auto lo

iface lo inet loopback

auto eth0

#iface eth0 inet dhcp

iface eth0 inet static

address 192.168.1.200

netmask 255.255.255.0

gateway 192.168.1.1

/etc/init.d/networking restart: 重启网络配置

ifup eth0: 激活eth0网卡

ifdown eth0: 关闭eth0网卡

/etc/resolv.conf: DNS配置

nameserver 192.168.1.1

nameserver 8.8.8.8

**b. 网络命令**

ping: 检查主机网络接口状态

netstat: 检查整个Linux网络状态。

route: 检查本地路由信息

ifconfig: 检查网卡的信息

nslookup：机器的IP地址和其对应的域名

dig:用于询问DNS域名服务器的灵活的工具

traceroute：追踪网络数据包的路由途径

**8. 系统管理**

**a. 系统命令**

ps: 检查系统进程(aux)

kill: 指定进程ID杀进程(-9)

killall: 通过批量名字杀死进程(-9)

top: 查看系统运行状态，以CPU占用最高排序

free: 查看系统内存资源

**b. 系统日志**

dmesg：dmesg用来显示开机信息，kernel会将开机信息存储在ring buffer。

/var/log/syslog: 查看系统日志

**9. 软件包管理**

**a. dpkg**  
dpkg是Debian系统的后台包管理器,类似RPM。也是Debian包管理系统的中流砥柱,负责安全卸载软件包,配置,以及维护已安装的软件包。由于ubuntu和Debian乃一脉相承，所以很多命令是不分彼此的。

dpkg -i package.deb 安装包

dpkg -r package 删除包

dpkg -P package 删除包（包括配置文件）

dpkg -L package 列出与该包关联的文件

dpkg -l package 显示该包的版本

dpkg –unpack package.deb 解开 deb 包的内容

dpkg -S keyword 搜索所属的包内容

dpkg -l 列出当前已安装的包

dpkg -c package.deb 列出 deb 包的内容

dpkg –configure package 配置包

**b. apt-get**  
apt-get是一条linux命令，适用于deb包管理式的操作系统，主要用于自动从互联网的软件仓库中搜索、安装、升级、卸载软件或操作系统。是debian，ubuntu发行版的包管理工具，与红帽中的yum工具非常类似。

软件源：

/etc/apt/sources.list

deb http://mirrors.163.com/ubuntu/ precise main universe restricted multiverse

deb-src http://mirrors.163.com/ubuntu/ precise main universe restricted multiverse

deb http://mirrors.163.com/ubuntu/ precise-security universe main multiverse restricted

deb-src http://mirrors.163.com/ubuntu/ precise-security universe main multiverse restricted

deb http://mirrors.163.com/ubuntu/ precise-updates universe main multiverse restricted

deb http://mirrors.163.com/ubuntu/ precise-proposed universe main multiverse restricted

deb-src http://mirrors.163.com/ubuntu/ precise-proposed universe main multiverse restricted

deb http://mirrors.163.com/ubuntu/ precise-backports universe main multiverse restricted

deb-src http://mirrors.163.com/ubuntu/ precise-backports universe main multiverse restricted

deb-src http://mirrors.163.com/ubuntu/ precise-updates universe main multiverse restricted

deb http://mirror.bjtu.edu.cn/cran/bin/linux/ubuntu precise/

命令:

apt-cache search : (package 搜索包)

apt-cache show : (package 获取包的相关信息，如说明、大小、版本等)

sudo apt-get install : (package 安装包)

sudo apt-get install : (package - - reinstall 重新安装包)

sudo apt-get -f install : (强制安装?#"-f = --fix-missing"当是修复安装吧...)

sudo apt-get remove : (package 删除包)

sudo apt-get remove --purge : (package 删除包，包括删除配置文件等)

sudo apt-get autoremove --purge : (package 删除包及其依赖的软件包+配置文件等（只对6.10有效，强烈推荐）)

sudo apt-get update : 更新源

sudo apt-get upgrade : 更新已安装的包

sudo apt-get dist-upgrade :升级系统

sudo apt-get dselect-upgrade :使用 dselect 升级

apt-cache depends : (package 了解使用依赖)

apt-cache rdepends : (package 了解某个具体的依赖?#当是查看该包被哪些包依赖吧...)

sudo apt-get build-dep : (package 安装相关的编译环境)

apt-get source : (package 下载该包的源代码)

sudo apt-get clean && sudo apt-get autoclean :清理下载文件的存档 && 只清理过时的包

sudo apt-get check :检查是否有损坏的依赖

**10. 常用软件**

**a. 远程管理**

telnet: 远程登录Linux主机

ssh: 远程安全登录Linux主机

scp: 基于ssh登陆进行安装的远程文件传输

ftp: 通过ftp协议远程文件传输

wget: 通过http协议远程文件下载

**b. 程序编译**

./configure：是用来检测你的安装平台的目标特征的。configure是一个脚本，用来确定所处系统的细节，比如使用何种编译器、何种库，以及编译器和库的保存位置，并把Makefile.in的相应部分进行替换，形成Makefile。

make: 是用来编译的，它从Makefile中读取指令，然后编译。

make install: 是用来安装的，它也从Makefile中读取指令，安装到指定的位置。

make clean: 删除临时文件

**c. 产生密钥**

ssh-keygen:产生SSH认证RSA密钥

ssh 是一个专为远程登录会话和其他网络服务提供安全性的协议。默认状态下ssh链接是需要密码认证的，可以通过添加系统认证（即公钥-私钥）的修改，修改后系统间切换可以避免密码输入和ssh认证.

~ ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (/home/conan/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/conan/.ssh/id\_rsa.

Your public key has been saved in /home/conan/.ssh/id\_rsa.pub.

The key fingerprint is:

40:9b:fe:ad:76:29:79:20:b5:7c:74:e4:c9:46:73:f1 conan@conan

The key's randomart image is:

+--[ RSA 2048]----+

| . .. |

| . o + .. |

| + = + E|

| . .. . \* |

| .oSo o |

| ..+.. |

| ..+.. |

| +.+ |

| ..+ |

+-----------------+

ls ~/.ssh

id\_rsa id\_rsa.pub known\_hosts authorized\_keys

id\_rsa：私钥

id\_rsa.pub: 公钥

known\_hosts：登陆认证过的主机信息

authorized\_keys：公钥授权文件

**d. 时间管理**  
参考文章：<http://blog.fens.me/linux-ntp/>

ntp：服务器是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源做同步化，它可以提供高精准度的时间校正，在LAN上与标准间差小于1毫秒，WAN上几十毫秒，且通过加密确认的方式来防止恶毒的协议攻击。

#服务器配置：

sudo apt-get install ntp

ps -aux|grep ntp

ntp 23717 0.0 0.0 39752 2184 ? Ss 19:40 0:00 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 126:135

/etc/init.d/ntp status

\* NTP server is running

#客户端同步：

ntpdate 0.cn.pool.ntp.org

22 Jul 16:45:05 ntpdate[6790]: step time server 218.75.4.130 offset 7618978.781348 sec

**e. 定时任务**

crontab:设置任务调度(el)

分钟m(0-59), 小时h(1-23), 日期dom(1-31), 月份mon(1-12), 星期down(0-6,0表示周日)

crobtab -e

# m h dom mon dow command

# 每周一早点5:00给目录打包

# 0 5 \* \* 1 tar -zcf /var/backups/home.tgz /home/

# 每天同步2次时间。(8:00, 16:00)

# 0 8,16 \* \* \* /usr/sbin/ntpdate 192.168.1.79

**f. 网络文件系统**

nfs：网络文件系统（NFS，Network File System）是一种将远程主机上的分区（目录）经网络挂载到本地系统的一种机制，通过对网络文件系统的支持，用户可以在本地系统上像操作本地分区一样来对远程主机的共享分区（目录）进行操作。

#服务器端安装

sudo apt-get install nfs-kernel-server

#配置访问权限

vi /etc/exports

/home/conan/.ssh \*(rw,no\_root\_squash,sync)

#重启

sudo /etc/init.d/nfs-kernel-server restart

#查看配置列表

showmount -e

Export list for conan:

/home/conan/.ssh \*

#创建目录

sudo mkdir /mnt/ssh

#挂载目录

sudo mount 192.168.1.201:/home/conan/.ssh /mnt/ssh/

#查看目录

ls /mnt/ssh/

id\_rsa id\_rsa.pub known\_hosts

**g. 域名服务器**  
参考文章：<http://blog.fens.me/vps-ip-dns/>

bind9：一种开源的DNS（Domain Name System）协议的实现，包含对域名的查询和响应所需的所有软件。它是互联网上最广泛使用的一种DNS服务器。

#安装

sudo apt-get install bind9

cd /etc/bind

#配置

vi named.conf.options

forwarders {

8.8.8.8;

8.8.4.4;

};

# 设置wtmart.com域

~ vi named.conf.local

zone "wtmart.com" {

type master;

file "/etc/bind/db.wtmart.com";

};

# 配置域名指向IP

~ vi db.wtmart.com

$TTL 604800

@ IN SOA wtmart.com. root.wtmart.com. (

2 ; Serial

604800 ; Refresh

86400 ; Retry

2419200 ; Expire

604800 ) ; Negative Cache TTL

;

@ IN NS wtmart.com.

@ IN A 173.255.193.179

@ IN AAAA ::1

ip IN A 173.255.193.179

lin IN A 116.24.133.86

#重启服务器

sudo /etc/init.d/bind9 restart

**h. 数据库服务器**

mysql: 是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle公司。MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL的SQL语言是用于访问数据库的最常用标准化语言。MySQL软件采用了双授权政策（本词条“授权政策”），它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择MySQL作为网站数据库。

#安装

sudo apt-get install mysql-server

#配置允许其他计算访问

sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf

#设置字符集UTF-8

vi /etc/mysql/my.cnf

[client]

port = 3306

socket = /var/run/mysqld/mysqld.sock

default-character-set=utf8

[mysqld]

#

# \* Basic Settings

#

default-storage-engine=INNODB

character-set-server=utf8

collation-server=utf8\_general\_ci

#查看字符集

mysql> show variables like '%char%';

+--------------------------+----------------------------+

| Variable\_name | Value |

+--------------------------+----------------------------+

| character\_set\_client | utf8 |

| character\_set\_connection | utf8 |

| character\_set\_database | utf8 |

| character\_set\_filesystem | binary |

| character\_set\_results | utf8 |

| character\_set\_server | utf8 |

| character\_set\_system | utf8 |

| character\_sets\_dir | /usr/share/mysql/charsets/ |

+--------------------------+----------------------------+

8 rows in set (0.00 sec)

#重启

sudo service mysql restart