第2章　数据与运算

内容导航

CONTENTS

- **向量、对象和函数**是R语言的核心组成部分和关键概念。
- 向量是R语言中用来组成数据的最小单位，是学习其他内容的基础。
- 面向对象是R语言的一大特点，类是对象的概念性抽象，而对象是类的具体实例。
- R是函数式编程语言，R内置了大量的函数，用户也可以创建自定义函数。R环境中所有的操作都类似于使用计算器，用户输入函数名和数据，R系统完成函数计算。

```
> a <- 1                          #给a赋值1
> a                               #显示a的值
[1] 1
> int_vec <- c (1L, 2L, 3L)       #用c ()创建向量，L表示整数常数
> int_vec
[1] 1 2 3
> 2:5
[1] 2 3 4 5
```

- R中不存在标量类型的数据，一个整数或字符也被看成长度为1的向量。
- 用赋值的方式可以创建向量。
- 用c ()函数也可以创建向量。
- 还可以用符号“:”函数创建向量

```
> a <- 1                          #将1赋值给变量a
> my_vector <- c (1, 3, 5)#创建一个向量，并将其赋值给变量my_vector
> my_matrix <- matrix(1:6, nrow = 2, ncol = 3) #创建一个两行三列的矩阵
> #创建一个名为my_func的函数，这个函数接收两个参数，并返回两参数值之和
> my_func <- function (x, y) return (x + y)          #创建一个函数
```

- R语言中万事万物皆对象。
- 使用赋值语句就可以创建对象，包括向量、矩阵和函数。

```
> v1 <- c (168, 173, NA, 180)                    #创建一个含有缺失值的向量
> #NA即“not available”，表示缺失值
> mean (v1)                                      #求v1的均值
[1] NA
> #可见默认情况下mean函数无法处理含有缺失值的数据
> #na.rm是排除缺失值的选项，用于排除v1中的NA值
> mean (v1, na.rm=TRUE)
[1] 173.6667
```

- 函数是由若干语句组成的执行特定功能的代码块。
- 使用函数可以减少代码重复，让程序更为简洁、高效，也增加了代码的可读性。
- R语言内置了大量的函数。
- 用户可以自己编写自定义函数。

- 在R中，对象和函数用标识符唯一表示。
- 标识符由**一串字母数字字符**（包括数字、字母表中的字母的大小写形式）、**点号"."**和**下划线"_"**组成。
- 如果一个标识符以点号打头，ls ()函数在默认情况下不会将其显示出来。
- 像“…”、“..1”、和“..2”这些特殊标识符在R语言里都属于合法标识符。
- R中的保留字不能用作对象名使用。

控制字	if, else, repeat, while, function, for, in, next, break
特殊值	TRUE, FALSE, NULL, Inf, NaN
缺失值	NA NA_integer_ NA_real_ NA_complex_ NA_character_
特殊参数	..., ..1, ..2

内容导航

CONTENTS

- 数据分析就是处理各种数据对象以获得深入知识的过程。
- 在计算机中，不同的数据类型要和它们具体的物理存储结构以及所支持的操作方式联系起来。
- R语言中提供了可以支持数据分析中的主要任务的基本数据类型、运算、以及在此基础上派生的复杂的数据结构。
- 在R语言中，**变量既不需要申明，其类型也可以改变**。
- 在对变量赋值时系统会根据所使用数值的类型给变量设定类型。

- R中共有6种基本数据类型，也叫原子数据类型。
- 分别是：逻辑型（logical）、浮点型（double）、整数型（integer）、字符型（character）、复数类型（complex）和原始类型（raw）。
- 前四种较为常见，浮点型又称双精度浮点数，和整型数合称为数值型。

类型	说明	判断函数	R语言形式
逻辑型	表示逻辑值的二值数据，只有TRUE或FALSE两个取值。在R中，逻辑表达式的赋值会得到逻辑型数据，例如比较两个数的大小2>1等于TRUE	is.logocal()	TRUE，2 <= 1
浮点型	用十进制表示的实数，如1，1.1等，是用于计算的基本数据形式	is.double ()	3.14
整数型	用于描述整数，如1，2，3。需要注意的是，**在R语言中，在整数后加上字符L才代表整型数，否则会被视为浮点数**	is.integer ()	3L
字符型	用于表示一个字符串	is.character ()	"Hello", "3.14"
复数型	用于表示复数值，其中虚部用i表示，例如2+3i	is.complex ()	1 + i
原始型	用于保存原始的字节，其中每个字节用两个十六进制数表示，例如A3	is.raw ()	00

```
> dbl_var <- c (1, 1.1, 1.11, 1.111)
> dbl_var                                          #查看向量的值
[1] 1.000 1.100 1.110 1.111
> typeof (dbl_var)       #查看对象的类型，浮点数也叫双精度型
[1] "double"
> is.integer (2)      #判断是否整型数，整数会被默认为浮点型数值
[1] FALSE
> is.integer (2L)          #整数后加上后缀L以避免混淆
[1] TRUE
```

- 创建一个双精度浮点数向量，参数有整型数，也有浮点数。
- 整数1被自动转换成浮点数1.000
- 在整数后面加上后缀L创建整数类型值，否则R会将数字自动处理成浮点型。

```
> #F和T是FALSE和TRUE的简写形式
> log_var <- c (TRUE, FALSE, F, T)
> chr_var <- c ("Is these ", ’ will be ’ , "combined to one sentence?")
> str (chr_var)
 chr [1:3] "Is these " "will be" "combined to one sentence?"
> #使用cat函数可将多个对象连接并输出到屏幕上
> chr_var <- cat ("This", 'is a', "cat")
This is a cat
```

- R是大小写敏感的，必须使用大写的逻辑值来创建逻辑型向量。
- 字符型的值输入时需要加上引号，可以是双引号"，也可以是单引号’
- 两者除了在包含对方时不用转义字符之外，是完全等价的。

```
> cat ("You should use \"her\" in this 'passage'.\n")
You should use "her" in this 'passage'.
> cat ('You should use "her" in this \'passage\'.\n')
You should use "her" in this 'passage'.
> x <- "long\tlines can be\nbroken with newlines"
> writeLines(x)                                    #输出文本行
long    lines can be
broken with newlines
```

- 用户希望产生的字符串中含有引号时，需要转义符来实现。
- 一些特殊字符是不能直接从键盘输入，需要使用“\”加特定的符号的方式产生特殊字符。
- R中把这种做法叫做转义，如\n表示换行，\b表示退格等。

```
> int_var <- 2L; dbl_var <- c (1.1, 2.2, 3.3)
> chr_var <- c ("a ", ' b ' , "c"); log_var <- c(TRUE, FALSE, F, T)
> is.numeric (dbl_var)                    #判断是否数值型
[1] TRUE
> is.numeric (int_var)
[1] TRUE
> types <- c (typeof(dbl_var), typeof (int_var), is.character (chr_var))
> types
[1] "double"  "integer"  "TRUE"
```

- 用typeof ()函数可以得到原子向量的类型。
- 用“is”函数可以检查某向量是否属于某种特定的类型。
- is.numeric ()函数对整数型和浮点型数据同时有效。
- 用is.atomic ()检查对象是否是原子类型。

```
> a<-"My Name"                    #双引号作为字符串分界符
> a                               #显示a的值
[1] "My Name"
> b<-'My Name'                    #单引号也可以作为分界符
> identical(a,b)                  #判断a和b是否全等
[1] TRUE
> typeof("3.1415")                #检查常数类型
[1] "character"
```

- 变量用于在内存中存储可以改变的数据。
- 变量需要满足命名规范。
- 常量（也称常数）表示不会发生变化的量。
- 一个常整数会被优先当作浮点数处理，所以需要在常数值后面加上后缀L表示其属于整型数，而后面跟着i的则是复数。

```
> v <- c(1, 2, 3)                    #把数值1，2，3合并成一个向量
> length(v) <- 4                     #扩大向量v 的长度到4
> v                                  #显示v，未经赋值的元素成为NA
[1]  1  2  3 NA
> 1 / 0
[1] Inf
> 0 / 0
[1] NaN
```

- NA是not available的缩写，表示缺失值。
- Inf表示无穷大，在前面添加负号则表示负无穷大。
- NaN是not a number的缩写，表示无意义的值。
- NULL表示空数据。

内容导航

CONTENTS

- R语言中提供了与其他程序设计语言类似的基本运算，主要的一元与二元运算操作类型包括：算术操作、关系操作、逻辑操作和赋值操作等。
- 运算符用于描述程序代码所需执行的运算，运算通常会作用于一个或多个数据（叫做操作数）来进行。同其他编程语言类似，R语言提供了丰富的内置运算符。

算术运算符	关系运算符	逻辑运算符	赋值运算符
+ 加法 - 减法 * 乘法 / 除法 %% 求余 %/% 求商 ^ 指数运算	> 大于 < 小于 == 等于 >= 大于等于 <= 小于等于 != 不等于	& 逻辑与 | 逻辑或 ! 逻辑非 && 逻辑与 || 逻辑或 (&&和||只考虑第一个元素)	<-,<<- 向左赋值 = 向左赋值 ->,->> 向右赋值

```
> x <- 5;y <- 8                    #给变量赋初值
> x + y                            #加法运算
[1] 13
> y / x                            #除法运算
[1] 1.6
> y %/% x                          #整除，向下取整
[1] 1
> y %% x                           #模除
[1] 3
```

- 算术运算是指加法与乘法之类的数学运算。
- **R中的除法与模除运算符与C语言不一致。**

```
> x <- 5;y <- 8                    #给变量赋初值
> x < y                            #比较两个变量的关系
[1] TRUE
> x <= y
[1] TRUE
> x == y
[1] FALSE
> y >= 10                          #比较变量与常数
[1] FALSE
```

- 关系运算用于比较两个操作数之间的关系。
- 关系运算产生的是逻辑值，TRUE，或FALSE。
- 这些结果可以进一步用于完成其他任务，例如条件判断。

```
> log_var <- c(F, F, T)                #初始化逻辑型向量log_var
> log_var2 <- c(T, F, T)               #初始化逻辑型向量log_var2
> log_var && log_var2                  #只对向量第一个元素做与操作
[1] FALSE
> log_var & log_var2                   #对向量所有元素做与操作
[1] FALSE FALSE  TRUE
```

- 逻辑运算即与、或、非等布尔运算。
- 符号&&和||带有一种“短路”的效果，即只考虑两个操作向量的第一个元素的运算结果，后面的元素会被自动忽略。
- 与&和|相比结果可能不同。

```
> x <- 123
> x
[1] 123
> x = 345
> x
[1] 345
> 123 -> y
> y
[1] 123
```

- 赋值运算用于给变量赋值。
- 在R语言中，“=”和“<-”（由“<”和“-”两个字符组成），“->”都可用于赋值。

```
> matrix (1:6, nrow = 2, ncol = 3, byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>#下面是错误用法，相当于调用matrix(1:6, 2, 1, 3)
> matrix (1:6, 2, byrow <- TRUE, 3)
     [,1]
[1,]    1
[2,]    2
```

- 使用=给函数参数赋值。
- 使用<-和->给局部（当前环境中的）变量赋值，使用运算符<<-和->>给全局变量赋值（父环境中的变量）。

内容导航

CONTENTS

- **数据类型**指的是变量取值的类型（如整数、字符、逻辑值等）。
- **数据结构**则是指组织和管理数据的方式，以及定义在数据结构之上的运算。
- 可以按照数据的维度（一维、二维或n维）以及同一数据对象内是否允许存在不同类型的数据来划分数据结构。

类型	维度	说明
原子向量	1维	同一对象中必须是同类型的数据
列表	1维	允许对象中存在不同类型的数据
矩阵	2维	同一对象中必须是同类型的数据
数据框	2维	允许对象中存在不同类型的数据
数组	多维	同一对象中必须是同类型的数据

- 数据类型转换包括**自动转换**与**强制转换**两种方式。
- 利用转换函数可以方便地完成类型转换。
- 原子向量要求向量中的元素的数据类型相同，否则自动将所有的元素统一成兼容度最高的类型。
- 各类型兼容性从小到大依次为：逻辑型、整数型、浮点型、字符型。

转换目标	转换函数	转换规则
数值型	as.numeric()	FALSE -> 0 TRUE -> 1 “1”, “2”, … ->1, 2, … 其他字符 -> NA
逻辑型	as.logical()	0 ->FALSE 除0外的其他数字 “FALSE”, “F” -> FALSE “TRUE”, “T” ->TRUE 其他字符 -> NA
字符型	as.character()	1, 2, … -> “1”, “2”, … FALSE -> “FALSE” TRUE -> “TRUE”

```
> int_var <- 2L; dbl_var <- c(1.1, 2.2, 3.3)
> chr_var <- c ("a ", ’ b ’ , "c"); log_var <- c (TRUE, FALSE, F, T)
> typeof (c (int_var, dbl_var))          #整数型被自动转换为浮点型
[1] "double"
> typeof (c (int_var, log_var))          #逻辑型被自动转换为整数型
[1] "integer"
> typeof (c (int_var, chr_var))          #整型数被自动转换为字符型
[1] "character"
```

- **类型兼容度从低到高依次为：逻辑型、整数型、浮点型、字符型。**

```
> vector (mode = 'logical', length = 3)     #使用vector ()创建向量
[1] FALSE FALSE FALSE
> numeric (3)                               #使用numeric ()创建向量
[1] 0 0 0
#使用c ()非常灵活，可以有不同类型的参数，但是它们会被强制转换成同一个类型
> vec_var <- c (1, "1", TRUE)
> mode (vec_var)                            #查看vec_var向量的类型
[1] "character"
```

- 一个向量应属于某种基本数据类型，向量中的所有元素都必须属于这种数据类型。
- **创建向量可以使用vector函数，它有两个参数：类型（mode）和长度（length）。**
- 也可以用**c ()**函数来组合一些元素创建向量。

```
> array (1:12, c(3,4))          #用array ()创建一个3乘4的数组
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

- 数组相较于向量而言多了一个维度信息。
- 数组可以是多维的，但数组中的元素类型依然要保持一致。
- 数组可通过**array ()**函数创建。

```
> #nrow和ncol用以指定矩阵的行数与列数
> matrix(1:12, nrow=4, ncol=3)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

- 矩阵是维度为2的数组。
- 最基本的创建矩阵的方法是使用**matrix ()**函数

```
> ls_var <- list (1:3, "a", c(T, F, T), list(name = c("Jack", "Rose")))
> str(ls_var)                                    #查看ls_var对象的结构信息
List of 4
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ :List of 1
  ..$ name: chr [1:2] "Jack" "Rose"
```

- 列表是R中一种灵活的数据结构，是一些对象的有序集合。
- 与原子向量最大的区别在于，列表允许将若干任意类型的对象整合到单个对象名下，即列表允许存在不同类型的数据。
- 可使用**list ()**函数来创建列表。

```
> name <- c ("Jack", "Rose")              #字符型向量name
> age <- c (12, 10)                       #浮点数型向量age
> df_var <- data.frame (name, age)  #创建data.frame对象
> print (df_var)
  name age
1 Jack  12
2 Rose  10
> df_var$age                                  #使用"$"符号取数据框中的一列
[1] 12 10
```

- **数据框中不同的列属于不同类型，但同一列中的数据必须是相同类型的数据。**
- 一个典型的数据框类似于数据库中的表。
- 可使用函数**data.frame()**创建一个数据框。