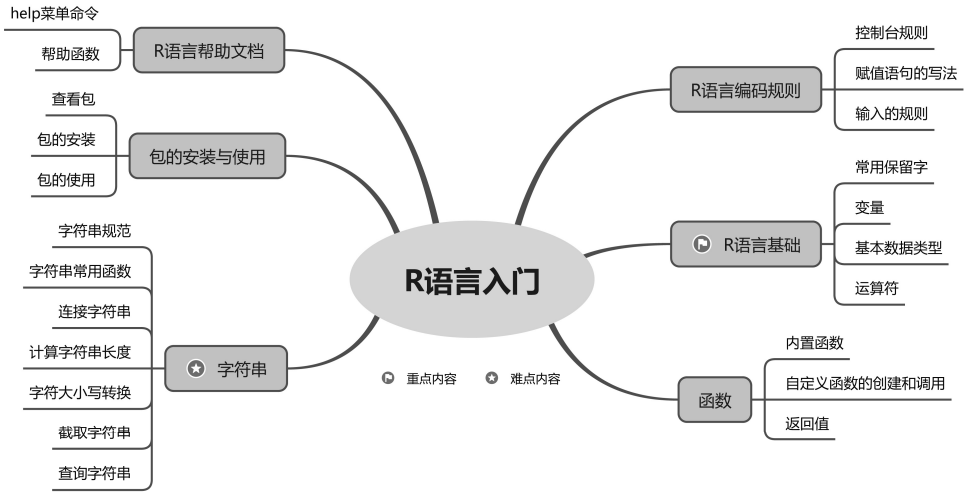
第 3 章

R 语言入门

从本章开始正式学习 R 语言。首先是入门知识，包括 R 语言编码规则，R 语言常用保留字、变量、基本数据类型和运算符，以及如何安装和使用 R 语言提供的程序包，最后介绍如何使用 R 语言提供的帮助文档，为 R 语言开发奠定基础。

本章知识架构及重难点如下。



3.1 R 语言编码规则

3.1.1 控制台规则

在 RGui 中编写代码主要使用控制台（R Console），下面先来了解控制台的规则。

（1）在 R Console 中直接编写代码时，在命令提示符“>”后输入代码并按 Enter 键，输入的代码会立刻运行。

（2）选择“文件”→“新建程序脚本”命令后，在 R 编辑器中输入代码并按 Enter 键，会换行而不会立刻运行。运行代码时，需要选择运行区域，然后选择“编辑”→“运行当前行或所选代码”命令。

在 R 编辑器中，必须用鼠标手动选中需要运行的代码，否则只运行光标所在位置的那一行代码。

（3）在 R Console 中编写代码无法直接通过鼠标拖曳改变代码的位置。例如，输入代码“print("奋发图强")”后，无法通过鼠标将“发”拖曳到“奋”前，选中“发”后按 Delete 键也无法将其删除，只能通过键盘上的左右方向键改变光标位置，按 Backspace 键进行删除。通过按键盘上的上下方向键可以显示上一行或下一行代码。

3.1.2 赋值语句的写法

R 语句由函数和赋值语句构成。需要注意的是，R 语言使用“<-”作为赋值符号，而不像其他语言那样使用“=”作为赋值符号。另外，“<-”符号的前后要各空一个格。

例如，将变量 a 赋值为 99，代码如下：

```
a <- 99
```

与 Python 一样，R 语言也使用“#”作为注释符号，被注释的内容不参与编译。需要注意，R 语言不支持多行注释。

3.1.3 输入的规则

（1）R 语言本身区分大小写，因此 a 和 A 在 R 语言里是两个变量。例如：

```
> a <- 99
> A <- 100
> a
[1] 99
> A
[1] 100
```

（2）多段代码可以使用分号（;）隔开。例如：

```
print("a");print("b");print("c")
```

（3）当输入的函数有误时，R 语言直接提示错误。当输入的函数不完整时，按 Enter 键自动出现一个加号“+”提醒，在“+”后可继续输入代码。如果代码还是不完整，按 Enter 键再次出现加号“+”，直到代码输入完整才停止提醒，如图 3.1 所示。

```
> print("奋发图强"
+
+ )
[1] "奋发图强"
```

图 3.1 加号“+”提醒

注意

如果代码中的错误是 R 语言也不认识的，就只能按 Esc 键退出。当输入行开头变成“>”时，才可以继续输入代码。

3.2 R 语言基础

3.2.1 常用保留字

保留字是 R 语言中被赋予特定意义的一些单词。在编写代码时，不可以把这些保留字作为变量、

函数、类、模块和其他对象的名称来使用。R 语言中的常用保留字如表 3.1 所示。

表 3.1　R 语言中常用的保留字

if	TRUE	next	NA	Inf
else	FALSE	repeat	return	NaN
for	function	break	while	NULL

表 3.1 中有几个比较特殊的保留字，是数据处理过程中经常遇到的保留字。

（1）NA：表示缺失值，是 Not Available 的缩写。

（2）Inf：表示无穷大，是 Infinite 的缩写。

（3）NaN：表示非数值，即不是一个数，是 Not a Number 的缩写。

（4）NULL：表示空值。

3.2.2　变量

程序运行过程中，其值可以发生改变的量称为“变量”。每个变量都有一个用于标识自己的名字，如 a。在 R 语言中，不需要先声明变量名及其类型，直接赋值即可创建各种类型的变量。

变量的命名并不是任意的，应遵循以下几条规则。

（1）变量名可以包含英文字母、数字、下画线和英文句号（.）。

（2）变量名不能存在中文（新版本可以使用中文，但不建议）、空格、“-”、“$”等符号。

（3）不能以数字和下画线开头。

（4）变量名以”.”符号开头的，后面不能是数字，如果是数字会变成 0.××××。

（5）变量名不能是 R 语言的保留字（如表 3.1 中常用的保留字）。

（6）谨慎使用小写字母 l 和大写字母 O，容易与 1 和 0 混淆。

（7）应选择有意义的单词作为变量名。

为变量赋值可以通过符号“<-”来实现，最好在“<-”符号前后各空一个格。

例如，创建一个变量 number，并为其赋值为 1024，代码如下：

```
number <- 1024              # 创建变量 number 并赋值为 1024，该变量为数值型
```

上述创建的变量 number 就是数值型的变量。

如果直接为变量赋值一个字符串，那么该变量就是字符型。代码如下：

```
myname <- "明日科技"        # 字符型的变量
```

另外，R 语言是一种动态类型语言，也就是说，变量的类型可以改变。例如，在 RGui 中创建变量 myname，为其赋值"明日科技"，输出变量类型会发现该变量为字符型；为其赋值 1024，输出变量类型会发现该变量为数值型。代码如下：

```
> myname <- "明日科技"
> print(mode(myname))
[1] "character"
> myname <- 1024
> print(mode(myname))
[1] "numeric"
```

说明

在 R 语言中，使用内置函数 mode()可以返回变量类型。

3.2.3 基本数据类型

R 语言的基本数据类型包括数值型（numeric，含数字、整型、双整型）、字符型（character）、逻辑型（logical）等，如表 3.2 所示。例如，一个人的姓名通常使用字符型存储，年龄通常使用数值型存储，婚否则可以使用逻辑型存储。

表 3.2 基本数据类型

数 据 类 型	说　明	举　例
numeric	数值型	包括数字，如 1、3.14、-99；整型，如 1、2、3、4、5……（必须为整数）；双整型，如 88、0.88、-88
character	字符型	"明日科技"
logical	逻辑型	TRUE、FALSE、NA
complex	复数类型	3.14+12.5j

下面对基本的数据类型进行详细介绍。

1．数值型（numeric）

数值型又可以分为数字（numeric）、整型（integer）、双整型（double）等。其中，整型用来表示整数数值，即没有小数部分的数值。R 语言中，在数字后面添加 L，即声明该数字以整型方式储存。在计算机内存中，整型比双整型更加准确（除非该整数非常大或非常小）。

双整型用于储存普通数值型数据，包括正数、负数和小数。R 语言中，输入的任何一个数值都默认以 Double 类型存储。在数据科学里，常被称为数值型（numeric）。

2．字符型（character）

字符型向量用以储存一小段文本，在 R 语言中字符要加双引号表示。字符型向量中的单个元素被称为“字符串（string）”，字符串不仅可以包含英文字母和中文，也可以由数字或符号组成。需要注意的是一定要加上引号，如"mingrisoft"、"明日科技"，不加引号会报错，单引号或双引号都可以。

3．逻辑型（logical）

逻辑型变量也就是我们常说的布尔变量（Boolean），其值为 TRUE 和 FALSE，或者大写的 T 和 F，需要注意的是英文字母全是大写。另外一种取值为 NA 缺失值（未知值），数值型和字符型也有 NA。

4．复数类型（complex）

R 语言中的复数与数学中的复数完全一致，都由实部和虚部组成，使用 j 或 J 表示虚部。当表示一个复数时，可以将其实部和虚部相加。例如，一个复数的实部为 3.14，虚部为 12.5j，则这个复数可表示为 3.14+12.5j。复数类型主要用来存储数据的原始字节。

读者可通过以下 3 个函数查看数据类型。

☑　class()函数：用于查看数据类型。

☑　mode()函数：用于查看数据大类。

☑　typeof()函数：用于查看数据细类。

示例代码如下：

```
class("mr")
class(TRUE)
class(99)
mode("mr")
mode(TRUE)
mode(99)
typeof("mr")
typeof(TRUE)
typeof(99)
```

运行程序，结果如图 3.2 所示。从运行结果得知不同函数的返回结果略有不同。

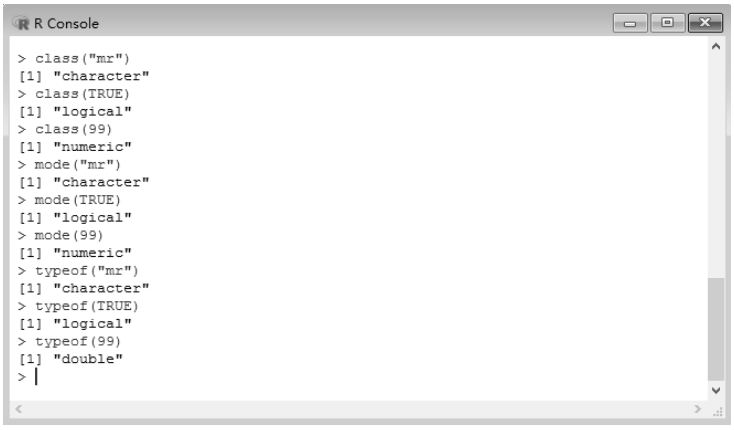


图 3.2　查看数据类型

除此之外，在程序中还可以对数据类型进行判断和转换，主要用到的函数如表 3.3 所示。

表 3.3　数据类型的判断和转换函数

数 据 类 型	判 断 函 数	转 换 函 数	数 据 类 型	判 断 函 数	转 换 函 数
numeric	is.numeric()	as.numeric()	character	is.character()	as.character()
interger	is.interger()	as.interger()	complex	is.complex()	as.complex()
logical	is.logical()	as.logical()			

其中，is 族函数用于判断数据类型，返回值为 TRUE 和 FALSE；as 族函数用于实现数据类型之间的转换。代码如下：

```
is.numeric("明日科技")          # 判断"明日科技"是否为数值型
as.numeric("88")               # 将字符串"88"转换为数值型 88
```

3.2.4　运算符

运算符是一些特殊的符号，主要用于数学计算、比较大小和逻辑运算等。R 语言中的运算符主要

包括算术运算符、关系（比较）运算符、逻辑运算符、赋值运算符和其他运算符。在 R 语言中，运算符主要用于向量运算，下面介绍一些常用的运算符。

1．算术运算符

在 R 语言中，算术运算符主要用于向量运算，即元素与元素之间的算术运算，如表 3.4 所示。

表 3.4　算术运算符

算术运算符	说　　明	举　　例	运 行 结 果
+	向量相加	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a+b)	3 3 1 5
-	向量相减	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a-b)	1　1 -1　5
*	向量相乘	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a*b)	2 2 0 0
/	向量相除	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a/b)	2　2　0 Inf
%%	两个向量求余，返回除法的余数	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a%%b)	0　0　0 NaN
%/%	两个向量取整除，返回商的整数部分	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a%/%b)	2　2　0 Inf
^或**	乘方，如 x^y（或 x**y）返回 x 的 y 次方。将第二个向量作为第一个向量的指数	a<-c(2,2,0,5) b<-c(1,1,1,0) print(a^b)	2 2 0 1

说明

向量是 R 语言中最简单、最重要的一种数据结构，由一系列同一种数据类型的有序的元素构成，类似一维数组。另外，上述代码 c()函数主要用于创建向量。例如 x<-c(1,2)，是将 1 和 2 两个数组合成向量(1,2)，并存入变量 x 中。有关向量更加详细的介绍可参见 4.1 节。

下面重点看一下 R 语言和其他编程语言有哪些不同。

（1）R 语言中，乘方运算既可以使用“^”符号，也可以使用“**”符号。

（2）除法“/”运算与 C/C++不同，在 C/C++中若不能整除则向下取整；R 语言与 Python 是一样的，均采用浮点数计算。例如，下面的代码分别为 R 语言和 Python 的除法运行结果。

R 语言代码：

```
> 1/3
[1] 0.3333333
```

Python 代码：

```
>>> 1/3
0.3333333333333333
```

【例 3.1】计算学生成绩的分差及平均分（**实例位置：资源包\Code\03\01**）

某学生 3 门课程的成绩如图 3.3 所示，通过编程实现：

（1）Python 和 R 语言课程的分数之差；

（2）3 门课程的平均分。

运行 RGui，新建程序脚本，名称为 demo.R。在该文件中定义 3 个变量，用于存储各门课程的分数，然后应用减法运算符计算分数差，应用加法运算符和除法运算符计算平均成绩，最后输出计算结果。代码如下：

```
python<-95                                  # 定义变量，存储 Python 的分数
english<-92                                 # 定义变量，存储 English 的分数
R<-89                                       # 定义变量，存储 R 语言的分数
sub<- python – R                            # 计算 Python 和 R 语言的分数差
avg<-(python + english + R) / 3             # 计算平均成绩
print(paste("Python 和 R 语言课程的分数之差:",sub,"分"))
print(paste("3 门课的平均分：  ",avg,"分"))
```

运行程序，结果如图 3.4 所示。

课程	分数
Python	95
English	92
R 语言	89

图 3.3　学生成绩

```
> python<-95                                  # 定义变量，存储Python的分数
> english<-92                                 # 定义变量，存储English的分数
> R<-89                                       # 定义变量，存储R语言的分数
> sub<- python - R                            # 计算Python和R语言的分数差
> avg<-(python + english + R) / 3             # 计算平均成绩
> print(paste("Python和R语言课程的分数之差:",sub,"分"))
[1] "Python和R语言课程的分数之差: 6 分"
> print(paste("3门课程的平均分：  ",avg,"分"))
[1] "3门课程的平均分：  92 分"
```

图 3.4　计算学生成绩的分差及平均分

- 代码解析

第 6～7 行代码中的 paste()函数用于连接字符串。

2．比较运算符

在 R 语言中，比较运算符可对两个向量中的元素逐个进行比较，比较后的结果为布尔值。常用的比较运算符如表 3.5 所示。

表 3.5　比较运算符

关系运算符	作　　用	举　　例	运 行 结 果
>	大于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a>b)	FALSE FALSE TRUE FALSE
<	小于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a<b)	TRUE TRUE FALSE TRUE

续表

关系运算符	作　　用	举　　例	运 行 结 果
==	等于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a==b)	FALSE FALSE FALSE FALSE
!=	不等于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a!=b)	TRUE TRUE TRUE TRUE
>=	大于或等于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a>=b)	FALSE FALSE TRUE FALSE
<=	小于或等于	a<-c(1,5,8,4) b<-c(4,15,4,5) print(a<=b)	TRUE TRUE FALSE TRUE

3．逻辑运算符

逻辑运算符可对真（TRUE）、假（FALSE）两种布尔值进行运算，即对两个向量中的元素逐个进行比较，运算后的结果仍是一个布尔值。R 语言中，逻辑运算符仅适用于逻辑型、数值型或复杂类型的向量。常用的逻辑运算符如表 3.6 所示。

表 3.6　逻辑运算符

逻辑运算符	用　　途	举　　例	运 行 结 果
&	与（and），按元素进行逻辑运算。两侧都为 TRUE，则结果为 TRUE。否则结果为 FALSE	a<-c(1,5,FALSE,8,FALSE) b<-c(4,15,FALSE,4,TRUE) print(a&b)	TRUE TRUE FALSE TRUE FALSE
&&	同上，不同的是仅判断向量中第一个元素	print(a&&b)	TRUE
|	或，两侧都为 FALSE，则结果为 FALSE。有一侧为真，则结果为 TRUE	print(a|b)	TRUE TRUE FALSE TRUE TRUE
||	同上，不同的是仅判断向量中第一个元素，如果其中一个为 TRUE，则结果为 TRUE	print(a||b)	TRUE
!	逻辑非运算符	print(!a)	FALSE FALSE TRUE FALSE TRUE

4．赋值运算符

在 R 语言中，赋值运算符用于为向量赋值，如表 3.7 所示。

表 3.7　赋值运算符

赋值运算符	用　　途	举　　例	运 行 结 果
<-	左分配赋值（即运算符左边为变量名，右边为向量值）	a1<-c(2,4,6,8,10) print(a1)	2 4 6 8 10
=		a2=c(2,4,6,8,10) print(a2)	2 4 6 8 10
<<-		a3<<-c(2,4,6,8,10) print(a3)	2 4 6 8 10
-<	右分配赋值（即运算符右边为变量名，左边为向量值）	c(2,4,6,8,10)->b1 print(b1)	2 4 6 8 10
-<<		c(2,4,6,8,10)->>b2 print(b2)	2 4 6 8 10

以上赋值运算符，在编程过程中常用的是标准赋值运算符“<-”。

5．其他运算符

还有一些特殊的运算符，用于特定目的，而不是一般的数学运算或逻辑运算，如表 3.8 所示。

表 3.8　其他运算符

其他运算符	用　　途	举　　例	运 行 结 果
:	用于创建公差为 1 或-1 的等差数列的向量，形式为 x:y	a<-c(1:9) print(a)	1 2 3 4 5 6 7 8 9
%in%	一种特殊的比较运算符，形式为 x %in% y 表示向量 x 中的元素是否符合向量 y 中的元素	c(1,3,4) %in% c(2,3,4) c(1,3) %in% c(2,3,4)	FALSE　TRUE　TRUE FALSE　TRUE
%*%	矩阵乘法	m1=matrix(1:4,2) m2=matrix(1:4,2) print(m1%*%m2)	[,1]　[,2] [1,]　7　15 [2,]　10　22

3.3　函　　数

我们可以把实现某一功能的代码定义为函数，在需要使用时，随时调用即可，十分方便。对于函数，简单地理解就是可以执行某项工作的代码块，类似积木块可以反复地使用。

在 R 语言中包括大量的内置函数可以在程序中直接调用。当然，也可以自己创建和使用函数，这种称为自定义函数。下面分别进行介绍。

3.3.1　内置函数

内置函数指的是 R 语言自带的函数，包括查看和改变路径的函数、数学函数、字符串函数等。例如，进行统计计算时常用的 min()、max()、sum()、mean()等函数。

内置函数可以在程序中直接调用，代码如下：

```
min(23:98)          # 最小值
max(23:98)          # 最大值
sum(23:98)          # 求和
mean(23:98)         # 平均值
```

运行程序，结果如图 3.5 所示。

```
> # 最小值
> min(23:98)
[1] 23
> # 最大值
> max(23:98)
[1] 98
> # 求和
> sum(23:98)
[1] 4598
> # 平均值
> mean(23:98)
[1] 60.5
```

图 3.5　内置函数

3.3.2　自定义函数的创建和调用

1．创建一个函数

用户自己创建的函数称为自定义函数，可以理解为用户创建了一个具有某种用途的工具。在 R 语言中，自定义函数主要使用 function 实现，语法格式如下：

```
function_name <- function(arg_1, arg_2, ...)
{
    Function body
}
```

参数说明如下。

- ☑ function_name：函数名称。
- ☑ arg_1, arg_2, ...：参数，是一个占位符。当函数被调用时，将传递一个值到参数。参数是可选的，也就是说，函数可能不包含参数。参数也可以有默认值。
- ☑ Function body：函数体，定义函数功能的代码块。

【例 3.2】自定义计算 BMI 指数的函数（**实例位置：资源包\Code\03\02**）

自定义计算 BMI（体脂）指数的函数 fun_bmi()。运行 RGui，新建程序脚本，编写如下代码：

```
fun_bmi <- function(height,weight)
{
    weight/(height*height)
}
```

代码解析

在上述代码中，fun_bmi 是创建的函数名，height 和 weight 是该函数的参数（形参），大括号中的内容为函数体，用于计算 BMI 指数。

说明

大括号也可以省略，因为函数体只有一行，省略后的代码如下：

```
fun_bmi <- function(height,weight)
    weight/(height*height)
```

2．调用函数

调用函数也就是执行函数。如果把创建函数理解为创建一个具有某种用途的工具，那么调用函数就相当于使用该工具。

例如，调用例 3.2 创建的 fun_bmi()函数，示例代码如下：

```
fun_bmi(1.6,65)
```

运行程序，结果为：

```
[1] 25.39062
```

上述代码也可以指定 x 和 y 参数，示例代码如下：

```
fun_bmi(x=1.6,y=65)
```

在不特别指定 x 和 y 的情况下，R 语言自动按位置进行匹配，即 1.6 为第一个参数，65 为第二个参数。

如果自定义函数中没有参数，可以直接调用函数名。示例代码如下：

```
myfun <- function()
{
    print(4*5)
}
myfun()
```

3.3.3　返回值

前面创建的函数都只是为了完成特定任务，并不需要返回什么。但在实际开发中，通常需要返回一个执行结果。这就好比主管向下级职员下达命令，职员做完后，还需要反馈结果给主管。函数返回值的作用就是将函数的处理结果返回给调用它的程序。

在 R 语言中，可以在函数体中使用 return()函数为自定义函数指定返回值。如果不使用 return()函数，则默认将最后执行的语句的值作为返回值。如果自定义函数需要多个返回值，则可以打包在一个列表（list）中。另外，函数的返回值可以是任何 R 语言对象。

例如，定义一个计算 BMI 指数的 fun_bmi()函数并要求返回 BMI 值，代码如下：

```
fun_bmi <- function(height,weight)
{
    bmi=weight/(height*height)
    return(bmi)
}
fun_bmi(1.5,70)
```

3.4　字　符　串

在程序开发中，姓名、性别（男或女）、商品名称、类别等通常用字符串表示，因此经常需要对字符串做各种操作处理。常用的字符串操作有连接字符串、计算字符串长度、字符大小写转换、截取字符串等。

3.4.1　字符串规范

在 R 语言中，通常使用一对单引号（''）或双引号（""）将字符串括起来。具体规范如下。

☑ 单引号或双引号应成对出现，分别位于字符串的开头和结尾。如'mrsoft'、"mrsoft"。
☑ 字符串的打印和显示通常采用双引号形式表示。
☑ 如果字符串中已经包含单引号，则再次出现单引号需要进行转义（使用反斜杠“\”）。
☑ 单引号不能插入以单引号开头和结尾的字符串中。
☑ 单引号可以插入以双引号开头和结尾的字符串中。
☑ 双引号不能插入以双引号开头和结尾的字符串中。

示例代码如下：

```
a <- '编程改变命运'
print(a)
b <- "编程改变命运"
print(b)
c <- '编程\'改变命运'
print(c)
d <- "编程'改变'命运"
print(d)
```

运行程序，结果如图 3.6 所示。

```
> a <- '编程改变命运'
> print(a)
[1] "编程改变命运"
> b <- "编程改变命运"
> print(b)
[1] "编程改变命运"
> c <- '编程\'改变命运'
> print(c)
[1] "编程'改变命运"
> d <- "编程'改变'命运"
> print(d)
[1] "编程'改变'命运"
```

图 3.6 字符串规范

3.4.2 字符串常用函数

字符串常用处理函数如表 3.9 所示。

表 3.9 字符串处理函数

函 数	说 明
strsplit()	字符串分割函数
paste(s1，s2，sep=)	字符串连接函数
paste0(s1,s2)	字符串连接函数（直接无缝连接）
nchar(s)	计算每个字符串的长度
length(s)	计算字符串总长度
substr(s,start,stop)	字符串截取函数
substring(s,start,stop=lenth(s))	字符串截取函数，可以不指定末尾位置，默认为字符串末尾
sub()	字符串替换函数，替换第一个匹配的字符串
gsub()	字符串替换函数，替换所有匹配的字符串
chartr()	字符串替换函数。用指定为参数的新字符替换字符串中出现该字符的所有匹配项
grep()	关键字匹配查询，返回匹配位置
grepl()	关键字匹配查询，返回布尔值
tolower()	小写转换
toupper()	大写转换
trimws()	去除字符串首尾的空格
match()	整词匹配查询，匹配模板

3.4.3 连接字符串

在 R 语言中，使用 paste()函数可以将字符串连接起来。paste()函数可以将任何数量的参数组合在

一起，语法格式如下：

```
paste(..., sep = " ", collapse = NULL)
```

参数说明如下。

☑ ...：要组合在一起的任意数量的变量。

☑ sep：参数之间的分隔符，可选参数。

☑ collapse：用于去除两个字符串之间的空格。注意，这里去除的不是一个字符串中两个字符之间的空格。

【例 3.3】字符串的连接（**实例位置：资源包\Code\03\03**）

使用 paste()函数连接各种字符串。运行 RGui，新建程序脚本，编写如下代码：

```
a <- "www"
b <- 'mrsoft'
c <- "com "
print(paste(a,b,c))
print(paste(a,b,c, sep = "."))
print(paste(a,b,c, sep = ".", collapse = ""))
```

运行程序，结果如图 3.7 所示。

```
> a <- "www"
> b <- 'mrsoft'
> c <- "com "
> print(paste(a,b,c))
[1] "www mrsoft com "
> print(paste(a,b,c, sep = "."))
[1] "www.mrsoft.com "
> print(paste(a,b,c, sep = ".", collapse = ""))
[1] "www.mrsoft.com "
```

图 3.7　连接字符串

3.4.4　计算字符串长度

nchar()函数用于计算字符串中字符的个数（包含空格的字符数），语法格式如下：

```
nchar(x)
```

其中，参数 x 表示一个向量。

示例代码如下：

```
myval <- nchar("吉林省明日科技有限公司 ")
print(myval)
myval <- nchar("www.mingrisoft.com")
print(myval)
```

运行程序，结果如图 3.8 所示。

```
> myval1 <- nchar("吉林省明日科技有限公司 ")
> print(myval1)
[1] 12
> myval2 <- nchar("www.mingrisoft.com")
> print(myval2)
[1] 18
```

图 3.8　计算字符串长度

3.4.5　字符大小写转换

在 R 语言中，toupper()函数和 tolower()函数用于改变字符串中字符的大小写，语法格式如下：

```
toupper(x)
tolower(x)
```

其中，参数 x 表示一个向量。

示例代码如下：

```
myval <- toupper("mingrisoft.COM")
print(myval)
myval <- tolower("mingrisoft.COM")
print(myval)
```

运行程序，结果如图 3.9 所示。

```
> myval <- toupper("mingrisoft.COM")
> print(myval)
[1] "MINGRISOFT.COM"
> myval <- tolower("mingrisoft.COM")
> print(myval)
[1] "mingrisoft.com"
```

图 3.9　大小写转换

3.4.6　截取字符串

substring()函数用于截取字符串中的一部分字符，语法格式如下：

```
substring(x,first,last)
```

参数说明如下。

☑　x：字符向量。

☑　first：要提取的第一个字符的位置。

☑　last：要提取的最后一个字符的位置。

示例代码如下：

```
myval <- substring("www.mingrisoft.com", 5, 14)
print(myval)
```

运行程序，结果如图 3.10 所示。

```
> myval <- substring("www.mingrisoft.com", 5, 14)
> print(myval)
[1] "mingrisoft"
```

图 3.10　字符串截取

3.4.7　查询字符串

grep()函数和 grepl()函数可在向量中查询指定字符串。

1．grep()函数

grep()函数用于在向量 x 中寻找含有特定字符串（由 pattern 参数指定）的元素，返回其在 x 中的下标。语法格式如下：

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE, invert = FALSE)
```

主要参数说明如下。

☑　pattern：字符串类型，正则表达式，用于指定搜索模式。当 fixed 为 TRUE 时，表示待搜索的字符串。

☑　x：字符串向量，表示被搜索的字符串。

☑　value：逻辑值，为 FALSE 时，grep 返回搜索结果的位置信息；为 TRUE 时，返回结果位置的值。

2．grepl()函数

grepl()函数用于查询向量 x 中是否包含特定的字符串，返回一个逻辑向量，值为 TRUE 或 FALSE。语法格式如下：

```
grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)
```

参数说明参见 grep()函数。

【例 3.4】查找指定的字符（**实例位置：资源包\Code\03\04**）

创建一个字符串向量，分别使用 grep()函数和 grepl()函数查找指定字符串。运行 RGui，新建程序

脚本，编写如下代码：

```
str1 <- c("m", "r", "mrsoft")          # 创建一个字符串向量
grep("m", str1)                        # 查找包含 m 的元素所在的位置
grepl("r", str1)                       # 判断每个元素是否包含 r，返回的是逻辑向量
grep("m|r", str1)                      # 同时匹配多个内容，查找包含 m 或者 r 的元素所在的位置
grepl("m|r", str1)                     # 同时匹配多个内容，判断每个元素是否包含 m 或者 r，返回的是逻辑向量
```

运行程序，结果如图 3.11 所示。

```
> #创建一个字符串向量
> str1 <- c("m", "r", "mrsoft")
> #查找包含m的元素所在的位置
> grep("m", str1)
[1] 1 3
> #判断每个元素是否包含r，返回的是逻辑向量
> grepl("r", str1)
[1] FALSE  TRUE  TRUE
> #同时匹配多个内容，查找包含m或者r的元素所在的位置
> grep("m|r", str1)
[1] 1 2 3
> #同时匹配多个内容，判断每个元素是否包含m或者r，返回的是逻辑向量
> grepl("m|r", str1)
[1] TRUE TRUE TRUE
```

图 3.11　使用 grep()函数和 grepl()函数查找指定字符串

3.5　包的安装与使用

3.5.1　查看包

R 语言中的程序包（以下简称包）是 R 函数、编译代码和样本数据集的集合，存储在 R 语言安装目录下的 library 文件夹中，如图 3.12 所示。

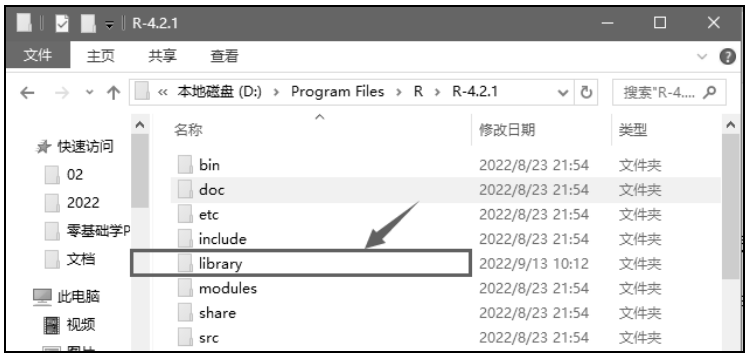


图 3.12　包的存储位置

在默认情况下，R 语言自带了一些包，如 base、boot、class 等。如果想了解 R 语言安装了哪些包，可以使用 library()函数查看，示例代码如下：

```
library()
```

运行程序，结果如图 3.13 所示。

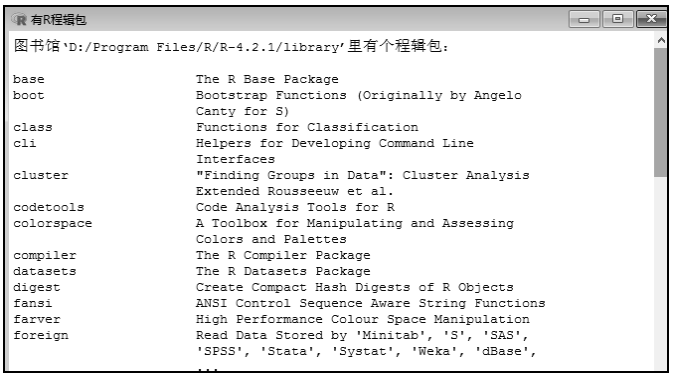


图 3.13　已经安装的包

3.5.2　包的安装

有时需要用到一些特殊功能的包（即扩展包），用户可以自行下载并安装它，方法有以下两种。

（1）在 RGui 控制台输入安装命令“install.packages("R 包名") ”来安装包。

例如，安装包 ggplot2，代码如下：

```
install.packages("ggplot2")
```

此时将显示一个 CRAN 镜像站点的列表，选择一个适合的镜像站点，如图 3.14 所示，单击“确定”按钮开始安装。

如果需要一次安装多个包，代码如下：

```
Install.packages(c("包 1","包 2"))
```

（2）在 RStudio 的资源管理窗口中，选择 Packages 进入 Packages 窗口，在包列表中选中需要安装的包，然后单击 Install 按钮下载并安装包，如图 3.15 所示。

图 3.14　CRAN 镜像列表

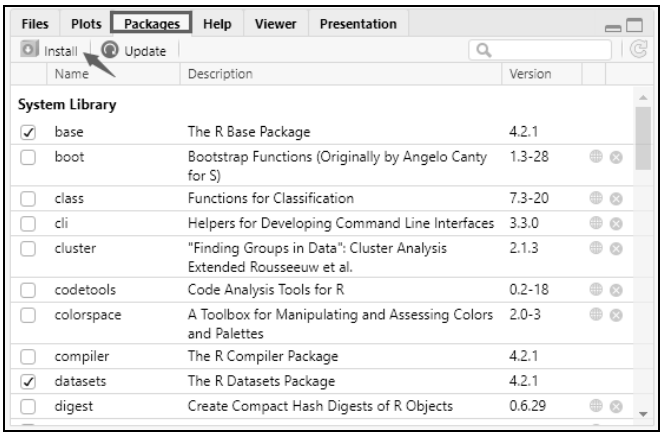


图 3.15　通过 RStudio 资源管理窗口安装包

3.5.3　包的使用

包安装完成后，在 RGui 控制台或 RStudio 代码编辑窗口中输入“library(包名)”或“require(包名)”就可以使用这些包了。如要使用包 lubridate，代码如下：

```
library(lubridate)
```

或者

```
require(lubridate)
```

如果编写代码时需要某个函数，则可以通过“包名::函数名”命令临时加载该包。代码如下：

```
lubridate::floor_date()
```

3.6　R 语言帮助文档

学会使用帮助文档不仅可以提高学习效率、缩短学习路径，还可以解决日常编程中遇到的问题。本节将介绍如何使用 R 语言中的帮助文档。

3.6.1　help 菜单命令

RStudio 中可以通过 Help 菜单查看帮助信息。选择 Help→R Help 命令，即可打开帮助页面，如图 3.16 所示。

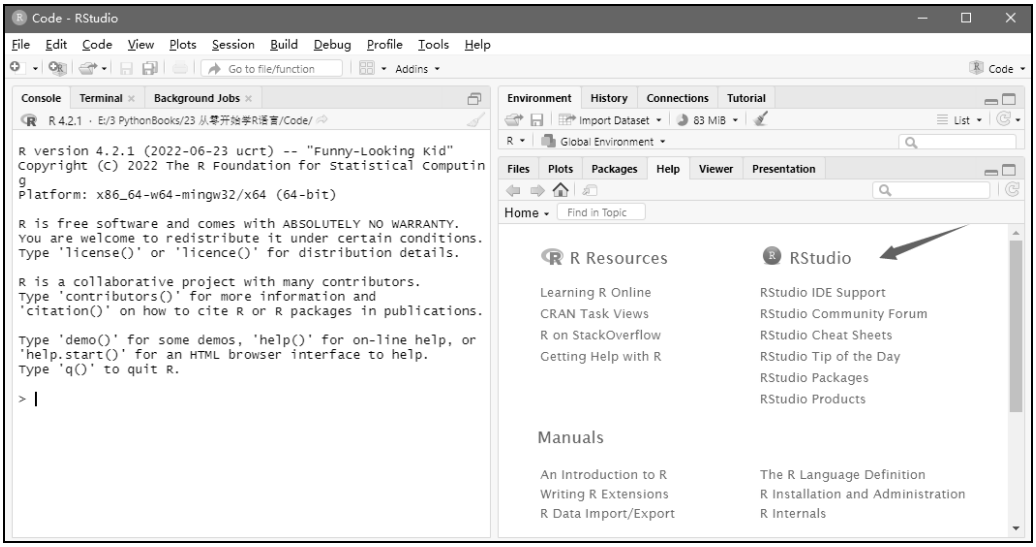


图 3.16　通过 Help 命令查看帮助信息

也可以使用搜索引擎输入关键字进行搜索。例如，在搜索框中输入“list”，下方会显示 list()函数

的相关帮助信息，如图 3.17 所示。

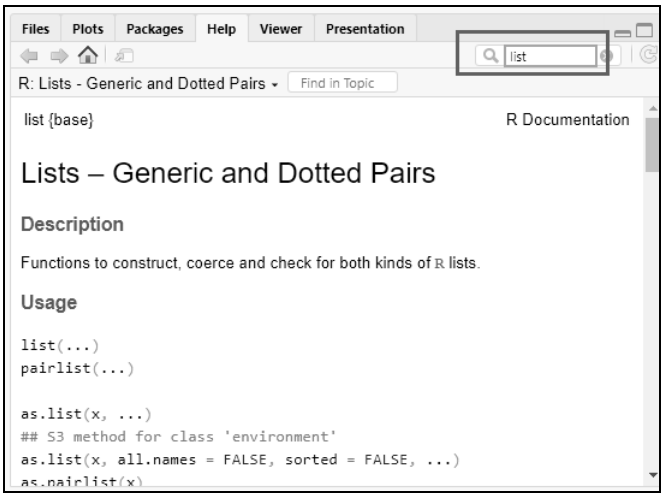


图 3.17　list()函数的帮助信息

在 RStudio 代码编辑窗口的指定关键词处按 F1 键，也可以显示相关的帮助信息。

3.6.2　帮助函数

R 语言还提供了一些帮助函数，用于获取函数的参数解释和使用示例等。这些函数需要在 RGui 或 RStudio 的控制台中使用，如表 3.10 所示。

表 3.10　帮助函数

函　　数	用　　途
help.start()	打开帮助文档首页
help()	查看函数的帮助文档，如“help("list")”或“?list”可查看 list 函数的帮助文档（引号可以省略）
help.search()	通过关键词搜索相关帮助文档，如“help.search("c")”或“??c”
args()	显示函数的参数，如 args(list)
example()	查看函数的使用示例，如 example("list")
demo()	列出包的应用场景，如 demo(graphics)
apropos()	列出名称中含有关键字的内容，如 apropos("c")　　# 列出所有包含 c 的内容 apropos("c",mode="funtion")　　# 只列出函数
data()	列出当前已加载包中包含的可用的示例数据集
RSiteSearch()	通过关键字搜索在线文档，如“RSiteSearch("list")”可通过关键词 list 搜索在线文档
vignette()	列出当前已安装包中所有可用的 vignette 文档，如“vignette("manage")”可显示 manage 的 vignette 文档

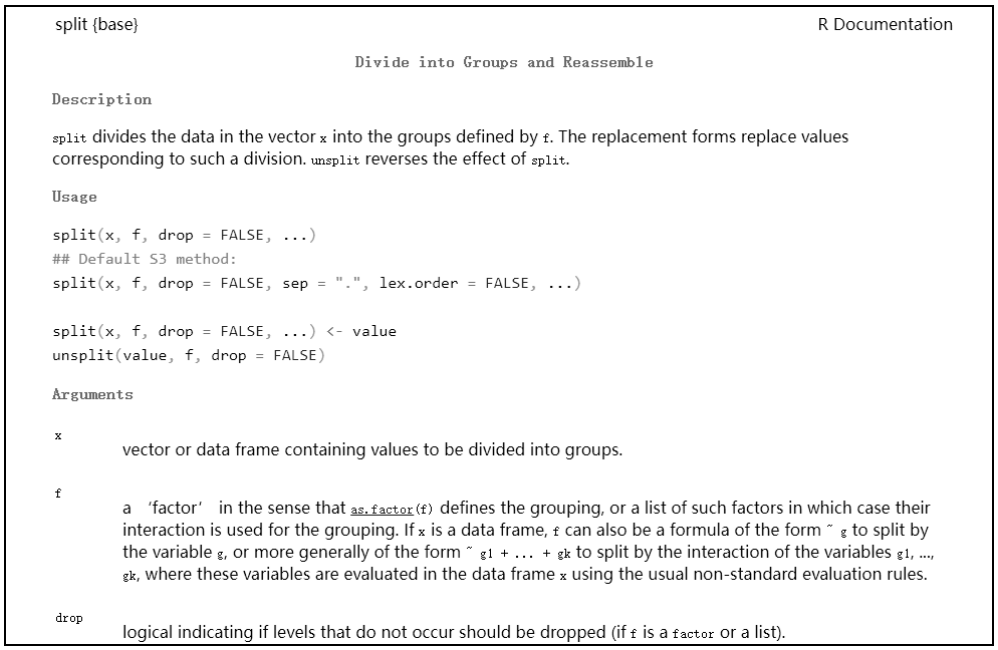
说明

vignette 文档包含很多内容，也更加规范，例如里面有简介、教程、开发文档等，可以通过 vignette()函数来查看，不过不是每个包都包含这种格式的文档。

例如，想了解 split()函数的用法，可在 RGui 控制台中输入如下代码：

```
help("split")
```

按 Enter 键，将打开帮助页面，如图 3.18 所示，其中包括语法、参数说明和示例等。



split {base}　　R Documentation

Divide into Groups and Reassemble

Description

split divides the data in the vector x into the groups defined by f. The replacement forms replace values corresponding to such a division. unsplit reverses the effect of split.

Usage

```
split(x, f, drop = FALSE, ...)
## Default S3 method:
split(x, f, drop = FALSE, sep = ".", lex.order = FALSE, ...)

split(x, f, drop = FALSE, ...) <- value
unsplit(value, f, drop = FALSE)
```

Arguments

x　vector or data frame containing values to be divided into groups.

f　a 'factor' in the sense that as.factor(f) defines the grouping, or a list of such factors in which case their interaction is used for the grouping. If x is a data frame, f can also be a formula of the form ~ g to split by the variable g, or more generally of the form ~ g1 + ... + gk to split by the interaction of the variables g1, ..., gk, where these variables are evaluated in the data frame x using the usual non-standard evaluation rules.

drop　logical indicating if levels that do not occur should be dropped (if f is a factor or a list).

图 3.18　split()函数的帮助页面

3.7　要 点 回 顾

本章主要学习了 R 语言编码规则和 R 语言基础知识，读者应熟练掌握。同时，语句的写法和输入规则，应熟练掌握的是 R 语言包的安装与使用以及如何使用帮助，这些都是在日常编程过程中需要频繁使用的操作。