

第4章

向量对象运算



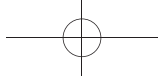
- 4-1 数值型的向量对象
- 4-2 常见向量对象的数学运算函数
- 4-3 Inf、-Inf、NA的向量运算
- 4-4 R语言的字符串数据属性
- 4-5 探索对象的属性
- 4-6 向量对象元素的存取
- 4-7 逻辑向量
- 4-8 不同长度向量对象相乘的应用
- 4-9 向量对象的元素名称



R语言最重要的特色是向量(vector)对象的概念，如果你学过其他计算机语言，应该知道一维数组(array)的概念，其实所谓的向量对象就是类似一组一维数组的数据，在此组数据中，每个元素的数据类型是一样的。不过向量的使用比其他高级语言灵活太多了，R语言的开发团队将此一维数组数据称为向量。

说穿了，R语言就是一种处理向量的语言。

其实R语言中最小的工作单位是向量对象，至于前面章节笔者当作实例使用的一些对象变量，从技术上讲可将那些对象变量看作是一个只含一个元素的向量对象变量。至今为止，在输出每一个数据时，首先出现的是“[1]”，中括号内的“1”表示接下来是从对象的第1个元素开始输出的。对数学应用而言，向量对象元素大都是数值数据型的，R语言的更重要的功能是向量对象元素可以是其他数据型，本书将在以后章节中一一介绍。



4-1 数值型的向量对象

数值型的向量对象可分为规则型的数值向量对象或不规则型的数值向量对象。

4-1-1 建立规则型的数值向量对象使用序列符号

从起始值到最终值，每次递增1，如果是负值则每次增加-1。例如从1到5，可用1:5方式表达；从11到16，可用11:16方式表达。在“1:5”或“11:16”的表达式中的“:”符号，即冒号，在R语言中被称为序列符号(sequence)。

实例 ch4_1：使用序列号“:”建立向量对象。

```
> x <- 1:5      # 设定向量变量对象包含1到5共5个元素
> x
[1] 1 2 3 4 5
> x <- 11:16    # 设定向量变量对象包含11到16共6个元素
> x
[1] 11 12 13 14 15 16
>
```

这种方式也可以应用于负值，每次增加-1。例如，从-3到-7，可用-3:-7的方式表达。

实例 ch4_2：使用序列号建立含负数的向量对象。

```
> x <- -3:-7    # 设定向量变量对象包含-3到-7共5个元素
> x
[1] -3 -4 -5 -6 -7
>
```

同理，这种方式也可以应用于实数，每次增加正1或负1。

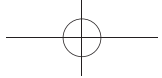
实例 ch4_3：使用序列号建立实数的向量对象。

```
> x <- 1.5:5.5   # 设定向量变量对象包含1.5到5.5共5个元素
> x
[1] 1.5 2.5 3.5 4.5 5.5
> x <- -1.8:-3.8 # 设定向量变量对象包含-1.8到-3.8共3个元素
> x
[1] -1.8 -2.8 -3.8
>
```

在建立向量对象时，如果写成1.5:4.7，结果会如何呢？这相当于建立含下列元素的向量对象，即1.5、2.5、3.5、4.5共4个元素，至于多余部分即4.5至4.7之间的部分则可不理会。如果向量对象元素为负值时，依此类推。

实例 ch4_4：另一个使用序列号建立实数的向量对象。

```
> x <- 1.5:4.7   # 设定向量变量对象包含1.5到4.5共4个元素
> x
[1] 1.5 2.5 3.5 4.5
> x <- -1.3:-5.2 # 设定向量变量对象包含-1.3到-4.3共4个元素
> x
[1] -1.3 -2.3 -3.3 -4.3
>
```



4-1-2 简单向量对象的运算

向量对象的一个重要功能是向量对象在执行运算时，向量对象内的所有元素将同时执行运算。

实例 ch4_5：将每一个元素加3的执行情形。

```
> x <- 1:5
> y <- x + 3
> y
[1] 4 5 6 7 8
>
```

一个向量对象也可以与另一个向量对象相加。

实例 ch4_6：向量对象相加的实例。

```
> x <- 1:5
> y <- x + 6:10 # 设定x向量加6:10向量，结果设定给向量y
> y
[1] 7 9 11 13 15
>
```

读至此节，相信各位读者一定已经感觉到R语言的强大功能了，如果上述命令使用非向量语言，需使用循环命令处理每个元素，要好几个步骤才可完成。在执行向量对象元素运算时，也可以处理不相同长度的向量对象运算，但先决条件是较长的向量对象的长度是较短的向量对象的长度的倍数。如果不是倍数，会出现错误信息。

实例 ch4_7：不同长度向量对象相加，出现错误的实例。

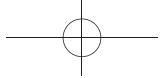
```
> x <- 1:5
> y <- x + 5:8
Warning message:
In x + 5:8 : 较长的物件长度并非较短物件长度的倍数
```

由于上述较长的向量对象有5个元素，较短向量对象有4个元素，所以较长向量的长度非较短向量的倍数，因此最后执行后出现警告信息。

实例 ch4_8：不同长度的向量对象相加，较长向量对象的长度是较短向量对象的长度的倍数的运算实例。

```
> x <- 1:3
> y <- x + 1:6
> y
[1] 2 4 6 5 7 9
>
```

上述的运算规则是，向量对象y的长度与较长的向量对象的长度相同，其长度是6，较长向量对象第1个元素与1:3的1相加，较长向量对象的第2个元素与1:3的2相加，较长向量对象的第3个元素与1:3的3相加，较长向量对象的第4个元素与1:3的1相加，较长向量对象第5个元素与1:3的2相加，较长向量对象第6个元素与1:3的3相加。未来如果碰上不同倍数的情况，运算规则可依此类推。



实例 ch4_9：下列是另一个不同长度向量对象相加的实例。

```
> x <- 1:5
> y <- 5
> x + y
[1] 6 7 8 9 10
>
```

在上述实例中，x向量对象有5个元素，y向量对象有1个元素，碰上这种加法，相当于每个x向量元素均加上y向量的元素值。在之前的实例中，在输出时，笔者均直接输入向量对象变量，即可在Console窗口打印此向量对象变量，在此例中，可以看到第3行，即使仍是一个数学运算，Console窗口仍将打印此数学运算的结果。

4-1-3 建立向量对象：seq()函数

seq()函数可用于建立一个规则型的数值向量对象，它的使用格式如下所示。

```
seq( from, to, by = width, length.out = numbers)
```

上述from是数值向量对象的起始值，to是数值向量对象的最终值，by则指出每个元素的增值。如果省略by参数同时没有length.out参数存在，则增值是1或-1。length.out参数字段可设定seq()函数所建立的元素个数。

实例 ch4_10：使用seq()建立规则型的数值向量对象。

```
> seq(1, 9)                # 建立1:9向量
[1] 1 2 3 4 5 6 7 8 9
> seq(1, 9, by = 2)         # 建立1至9间增值为2的向量
[1] 1 3 5 7 9
> seq(1, 9, by = pi)        # 建立1至9间增值为pi的向量
[1] 1.000000 4.141593 7.283185
> seq(1.5, 4.5, by = 0.5)   # 建立1.5至4.5间增值为0.5的向量
[1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5
> seq(1, 9, length.out = 5) # 建立1至9间元素个数为5的向量
[1] 1 3 5 7 9
>
```

4-1-4 连接向量对象：c()函数

c()函数的c为concatenate的缩写。这个函数并不是一个建立向量对象的函数，只是一个将向量元素连接起来的函数。

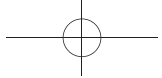
实例 ch4_11：使用c()函数建立一个简单的向量对象。

```
> x <- c(1, 3, 7, 2, 9)    # 一个含5个元素的向量
> x
[1] 1 3 7 2 9
>
```

上述x是一个向量对象，共有5个元素，内容分别是1、3、7、2、9。

适度地为变量取一个容易记的变量名称，可以增加程序的可读性。例如，我们想建立NBA球星Lin，2016年前6场进球数的向量对象，那么假设他的每场进球数如下所示：

7, 8, 6, 11, 9, 12



此时可用**baskets.NBA2016.Lin**当变量名称，相信这样处理后，即使程序放再久，开发人员也可以轻易了解程序内容。

实例 ch4_12：建立 NBA 球星进球数的向量对象。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> baskets.NBA2016.Lin
[1] 7 8 6 11 9 12
>
```

如果球星**Lin**的进球皆是2分球，则他每场得分如下。

实例 ch4_13：计算 NBA 球星的得分。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> scores.NBA2016.Lin <- baskets.NBA2016.Lin * 2
> scores.NBA2016.Lin
[1] 14 16 12 22 18 24
>
```

假设队友**Jordan**前6场进球数分别是10, 5, 9, 12, 7, 11，我们可以用如下方式计算每场两个人的得分总计。

实例 ch4_14：计算 NBA 球星 **Lin** 和 **Jordan** 的每场总得分。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> baskets.NBA2016.Jordan <- c(10, 5, 9, 12, 7, 11)
> total <- ( baskets.NBA2016.Jordan + baskets.NBA2016.Lin ) * 2
> total
[1] 34 26 30 46 32 46
>
```

先前介绍可以使用**c()**函数，将元素连接起来，其实也可以使用该函数将两个向量对象连接起来，下面是将**Lin**和**Jordan**进球数连接起来，结果是一个含12个元素的向量对象的实例。

实例 ch4_15：使用 **c()** 函数建立向量对象，其中 **c()** 函数内有多个向量对象参数。

```
> all.baskets.NBA2016 <- c(baskets.NBA2016.Lin, baskets.NBA2016.Jordan)
> all.baskets.NBA2016
[1] 7 8 6 11 9 12 10 5 9 12 7 11
>
```

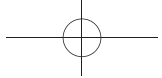
从上述执行结果可以看到，**c()**函数保持每个元素在向量对象内的顺序，这个功能很重要，因为未来我们要讲解如何从向量对象中存取元素值。

4-1-5 重复向量对象：rep()函数

如果向量对象内某些元素是重复的，则可以使用**rep()**函数建立这种类型的向量对象，它的使用格式如下所示。

rep(x, times = 重复次数, each = 每次每个元素重复次数, length.out = 向量长度)

如果**rep()**函数内只含有**x**和**times**参数，则“**times =**”参数可省略。



实例 ch4_16 : 使用 rep () 函数建立向量对象的应用。

```
> rep(5, 5)                #重复向量元素5, 共5次
[1] 5 5 5 5 5
> rep(5, times = 5)        #重复向量元素5, 共5次
[1] 5 5 5 5 5
> rep(1:5, 3)              #重复向量1:5, 共3次
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:3, times = 3, each = 2) #重复向量1:3, 共3次, 每个元素出现2次
[1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
> rep(1:3, each = 2, length.out = 8) #重复向量1:3, 每个元素出现2次, 向量元素个数是8
[1] 1 1 2 2 3 3 1 1
>
```

4-1-6 numeric()函数

numeric()函数也是建立一个向量对象, 主要是可用于建立一个固定长度的向量对象, 同时向量对象元素默认值是0。

实例 ch4_17 : 建立一个含10个元素的向量对象, 同时这些向量对象元素值皆为0。

```
> x <- numeric(10)        #建立一个含10个元素值为0的向量
> x                        #验证结果
[1] 0 0 0 0 0 0 0 0 0 0
>
```

4-1-7 程序语句短语跨行的处理

在本章4-1-5节的最后一个实例中, 可以很明显看到rep()函数包含说明文字已超出一行, 其实R语言是可以识别这行的命令未结束, 下一行是属于同一条命令的。除了上述状况外, 下列是几种可能发生程序跨行的状况。

(1) 该行以数学符号(+、-、*、/)作为结尾, 此时R语言的编译程序会知道下一行是接续此行的。

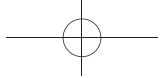
实例 ch4_18 : 以数学符号作结尾, 了解程序跨行的处理。

```
> all.baskets.NBA2016 <- baskets.NBA2016.Jordan +
+ baskets.NBA2016.Lin
> all.baskets.NBA2016
[1] 17 13 15 23 16 23
>
```

(2) 使用左括号“(”, R语言编辑器会知道在下一行出现的片断数据是同一括号内的命令, 直至出现右括号“)”, 才代表命令结束。

实例 ch4_19 : 使用左括号“(”和右括号“)”, 了解程序跨行的处理。

```
> x <- rep(1:5, times = 2,)
> x <- rep(1:5, times = 2,
+       each = 2)
> x
[1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5
>
```



(3) 字符串是指双引号间的文字字符，在设定字符串时，如果有了第一个双引号，但尚未出现第二个双引号，R语言编辑器可以知道下一行出现的字符串是属于同一字符串向量变量的数据，但此时换行字符“\n”将被视为字符串的一部分。

注：有关字符串数据的概念，将在4-4节说明。

实例 ch4_20：使用字符串，了解程序跨行的处理。

```
> coffee.Knowledge <- "Coffee is mainly produced
+ in frigid regions."
> coffee.Knowledge
[1] "Coffee is mainly produced\nin frigid regions."
>
```

4-2 常见向量对象的数学运算函数

研读至此，如果你学过其他高级计算机语言，你会发现向量对象变量已经取代了一般计算机程序语言的变量，这是一种新的思维，同时在阅读本节的常用向量对象的数学运算函数后，你将发现为何R语言这么受欢迎。

1. 常见运算

sum()：可计算所有元素的和。

max()：可计算所有元素的最大值。

min()：可计算所有元素的最小值。

mean()：可计算所有元素的平均值。

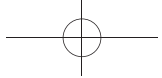
实例 ch4_21：sum()、max()、min() 和 mean() 函数的应用。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> sum(baskets.NBA2016.Lin)      #计算Lin的总进球数
[1] 53
> max(baskets.NBA2016.Lin)     #计算Lin的最高进球数
[1] 12
> min(baskets.NBA2016.Lin)     #计算Lin的最低进球数
[1] 6
> mean(baskets.NBA2016.Lin)    #计算Lin的平均进球数
[1] 8.833333
>
```

此外，这几个函数也可以对多个向量对象变量执行运算。

实例 ch4_22：sum()、max() 和 min() 函数的参数含有多个向量对象变量的应用。

```
> baskets.NBA2016.Jordan <- c(10, 5, 9, 15, 7, 11)
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> sum(baskets.NBA2016.Lin, baskets.NBA2016.Jordan) #计算2人的总进球数
[1] 110
> max(baskets.NBA2016.Lin, baskets.NBA2016.Jordan) #计算2人的最高进球数
[1] 15
> min(baskets.NBA2016.Lin, baskets.NBA2016.Jordan) #计算2人的最低进球数
[1] 5
>
```



2. prod()函数

prod(): 计算所有元素的积。

实例 ch4_23 : 使用prod() 函数执行阶乘的运算。

```
> prod(1:5)          # 计算从1乘到5，相当于factorial(5)
[1] 120
>
```

这个函数可以用在排列组合的计算中，如假设有5个数字，请问有几种组合？在实际操作前，可以先简化该问题，假设有两个数字，会有多少种排列方式？很容易，是两种排列方式。那有3个数字呢？是6种排列方式。如果是4个数字呢？是24种排列方式。

实例 ch4_24 : 有2、3或4个数字，计算排列组合方法有多少种的应用。

```
> prod(1:2)
[1] 2
> prod(1:3)
[1] 6
> prod(1:4)
[1] 24
>
```

3. 累积运算函数

cumsum(): 计算所有元素的累积和。

cumprod(): 计算所有元素的累积积。

cummax(): 可返回各元素从向量起点到该元素位置所有元素的最大值。

cummin(): 可返回各元素从向量起点到该元素位置所有元素的最小值。

实例 ch4_25 : 累积函数的应用。

```
> baskets.NBA2016.Jordan
[1] 10 5 9 15 7 11
> cumsum(baskets.NBA2016.Jordan)
[1] 10 15 24 39 46 57
> cumprod(baskets.NBA2016.Jordan)
[1] 10 50 450 6750 47250 519750
> cummax(baskets.NBA2016.Jordan)
[1] 10 10 10 15 15 15
> cummin(baskets.NBA2016.Jordan)
[1] 10 5 5 5 5 5
>
```

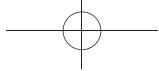
4. 差值运算函数

diff(): 返回各元素与下一个元素的差。

由于是返回每个元素与下一个元素的差值，所以结果向量对象会比原先向量对象少一个元素。

实例 ch4_26 : diff() 函数的应用。

```
> baskets.NBA2016.Jordan
[1] 10 5 9 15 7 11
> diff(baskets.NBA2016.Jordan)
[1] -5 4 6 -8 4
>
```



5. 排序函数

`sort(x, decreasing = FALSE)`: 默认是从小排到大, 所以如果是从小排到大, 则可以省略 `decreasing` 参数。如果设定 “`decreasing = TRUE`”, 则是从大排到小。

`rank()`: 传回向量对象, 这个向量对象的内容是原向量对象的各元素在原向量对象从小到大排序后, 在所得向量对象中的次序。

`rev()`: 这个函数可将向量对象颠倒排列。

实例 ch4_27 : 排序函数的应用。

```
> baskets.NBA2016.Jordan
[1] 10 5 9 15 7 11
> sort(baskets.NBA2016.Jordan)           #从小排到大
[1] 5 7 9 10 11 15
> sort(baskets.NBA2016.Jordan, decreasing = TRUE) #从大排到小
[1] 15 11 10 9 7 5
> rank(baskets.NBA2016.Jordan)
[1] 4 1 3 6 2 5
>
```

实例 ch4_28 : 向量颠倒排列的应用。

```
> x <- c(7, 11, 4, 9, 6)
> rev(x)
[1] 6 9 4 11 7
>
```

6. 计算向量对象长度的函数

`length()`: 可计算向量对象的长度, 也就是向量对象元素个数。

实例 ch4_29 : 计算向量对象的长度。

```
> baskets.NBA2016.Jordan           #先检查此向量的元素内容
[1] 10 5 9 15 7 11
> x <- baskets.NBA2016.Jordan      #列出此向量的元素个数
> length(x)
[1] 6
>
```

很明显, 该向量对象的元素有6个, 所以传回长度是6。

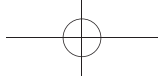
7. 基本统计函数

`sd()`: 计算样本的标准差。

`var()`: 计算样本的方差。

实例 ch4_30 : 基本统计函数的使用。

```
> sd(c(11, 15, 18))
[1] 3.511885
> var(14:16)
[1] 1
>
```



4-3 Inf、-Inf、NA的向量运算

前一小节所介绍的向量允许元素含有正无限大(Inf)、负无限大(-Inf)和缺失值(NA)。任何整数或实数值与Inf相加，结果均是Inf。任何整数或实数值与-Inf相加，结果均是-Inf。

实例 ch4_31：向量对象运算，其中函数内含 Inf 和 -Inf。

```
> max(c(43, 98, Inf))
[1] Inf
> sum(c(33, 98, Inf))
[1] Inf
> min(c(43, 98, Inf))
[1] 43
> min(c(43, 98, -Inf))
[1] -Inf
> sum(c(65, -Inf, 999))
[1] -Inf
> |
```

如果函数中的向量对象的参数包含NA，则运算结果是NA。

实例 ch4_32：向量对象运算，其中函数参数内含 NA。

```
> max(c(98, 54, 123, NA))
[1] NA
>
```

为了克服向量对象元素可能有缺失值NA的情形，通常在函数内加上“na.rm = TRUE”参数，这样函数碰上有向量对象的参数是NA时，也可正常运算了。

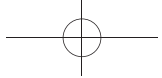
实例 ch4_33：向量对象运算，其中向量对象的元素内含 NA，同时函数的参数含“na.rm = TRUE”。

```
> max(c(98, 54, 123, NA), na.rm = TRUE)
[1] 123
> sum(c(100, NA, 200), na.rm = TRUE)
[1] 300
> min(c(98, 54, 123, NA), na.rm = TRUE)
[1] 54
> |
```

特别需要注意的是，diff()函数与累积函数cummax()、cummin()相同，无法使用去掉缺失值NA的参数“na.rm = TRUE”。

实例 ch4_34：diff()和累积函数无法使用“na.rm = TRUE”参数的实例。

```
> x <- c(9, 7, 11, NA, 1)
> cummin(x)
[1] 9 7 7 NA NA
> cummax(x)
[1] 9 9 11 NA NA
> diff(x)
[1] -2 4 NA NA
>
```



上述`cummin()`和`cummax()`函数由于计算到第4个向量对象的元素碰上NA，自此以后的结果皆以NA表示。对于`diff()`函数而言，第3个元素11和第4个元素NA比较是传回NA，第4个NA元素和第5个元素1比较也是传回NA。

4-4 R语言的字符串数据属性

至今所介绍的向量数据大都是整数，其实常见的R语言是可以有下列数据类型的。

- **integer**: 整数。
- **double**: R语言在处理实数运算时，预设是用双精度实数计算和存储。
- **character**: 字符串。

处理字符串向量对象与处理整数向量对象类似，可以使用`c()`函数建立字符串向量，应特别注意字符串可以用双引号(“ ”)也可以用单引号(‘ ’)。

实例 ch4_35: 建立一个字符串向量对象，并验证结果，本实例同时用双引号(“ ”)和单引号(‘ ’)。

```
> x <- c("Hello R World")
> x
[1] "Hello R World"
> x.New <- ('Hello R World')
> x.New
[1] "Hello R World"
>
```

实例 ch4_36: 另外两种字符串向量对象的建立。

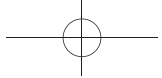
```
> x1 <- c("H", "e", "l", "l", "o")
> x1
[1] "H" "e" "l" "l" "o"
> x2 <- c("Hello", "R", "World")
> x2
[1] "Hello" "R"      "World"
>
```

4-2节所介绍的`length()`函数也可应用于字符串向量对象，可由此了解向量对象的长度(即元素的个数)。请留意，必须接着上述实例，执行下列实例。

实例 ch4_37: 延续上一个实例，计算向量对象的长度。

```
> length(x)
[1] 1
> length(x1)
[1] 5
> length(x2)
[1] 3
>
```

`nchar()`函数可用于列出字符串向量对象每一个元素的字符数。



实例 ch4_38：延续上一个实例，计算向量对象每一个元素的字符数。

```
> nchar(x)
[1] 13
> nchar(x1)
[1] 1 1 1 1 1
> nchar(x2)
[1] 5 1 5
>
```

对上述两个实例的运行结果进行综合整理，结果如下所示。

“Hello R World”：向量对象的长度是1，字符数是13。

“H” “e” “l” “l” “o”：向量对象的长度是5，每一个元素的字符数是1。

“Hello” “R” “World”：向量对象的长度是3，每一个元素的字符数分别是5、1、5。

4-5 探索对象的属性

4-5-1 探索对象元素的属性

至今笔者已介绍整数向量对象、实数向量对象、字符串向量对象，在R语言程序的设计过程中，可能会有一时无法知道对象变量属性的情形，这时可以使用下列函数判断对象属性，判断结果如果是真则传回TRUE，否则传回FALSE。

- ◀ is.integer()：对象是否为整数。
- ◀ is.numeric()：对象是否为数字。
- ◀ is.double()：对象是否为双精度实数。
- ◀ is.character()：对象是否为字符串。

实例 ch4_39：判断对象元素是否为整数的应用。

```
> x1 <- c(1:5)      #整数向量
> x2 <- c(1.5, 2.5)  #实数向量
> x3 <- c("Hello")  #字符串向量
> is.integer(x1)
[1] TRUE
> is.integer(x2)
[1] FALSE
> is.integer(x3)
[1] FALSE
>
```

对以下实例而言，x1、x2、x3对象内容与

上述相同。

实例 ch4_40：判断对象元素是否为数字的应用。

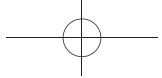
```
> is.numeric(x1)
[1] TRUE
> is.numeric(x2)
[1] TRUE
> is.numeric(x3)
[1] FALSE
>
```

实例 ch4_41：判断对象元素是否为双精度实数的应用。

```
> is.double(x1)
[1] FALSE
> is.double(x2)
[1] TRUE
> is.double(x3)
[1] FALSE
>
```

实例 ch4_42：判断对象元素是否为字符串的应用。

```
> is.character(x1)
[1] FALSE
> is.character(x2)
[1] FALSE
> is.character(x3)
[1] TRUE
>
```



4-5-2 探索对象的结构

`str()`函数可用于探索对象的结构。对于向量对象而言,可由此了解对象的数据类型、长度和元素内容。

实例 ch4_43:探索对象的结构。

```
> baskets.NBA2016.Lin      # 先了解向量对象内容
[1] 7 8 6 11 9 12
> str(baskets.NBA2016.Lin) # 验证与了解向量结构
num [1:6] 7 8 6 11 9 12
>
```

从上述执行结果可知, `baskets.NBA2016.Lin`对象的结构是数据类型,即`num`(数值),有1个维度,长度是6,元素内容分别是7、8、6、11、9、12。如果元素太多,则只列出部分元素内容。下列是查询字符串对象`x1`和`x2`的结构实例。

实例 ch4_44:探索另外两个对象的结构。

```
> x1 <- c("H", "e", "l", "l", "o")      # 建立对象x1
> str(x1)
chr [1:5] "H" "e" "l" "l" "o"
> x2 <- c("Hello", "R", "World")        # 建立对象x2
> str(x2)
chr [1:3] "Hello" "R" "World"
>
```

4-5-3 探索对象的数据类型

对于向量对象而言,可以使用`class()`函数,了解此对象元素的数据类型。

实例 ch4_45: `class()`函数的应用,了解对象元素的数据类型。

```
> x1 <- c(1:5)
> x2 <- c(1.5, 2.5)
> x3 <- c("Hello!")
> class(x1)
[1] "integer"
> class(x2)
[1] "numeric"
> class(x3)
[1] "character"
>
```

需特别留意的是,如果向量对象内的元素同时包含整数、实数、字符时,若使用`class()`判别它的数据类型,将返回“character”(字符)。

```
> x4 <- c(x1, x2, x3)
> class(x4)
[1] "character"
>
```

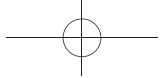
4-6 向量对象元素的存取

4-6-1 使用索引取得向量对象的元素

了解向量对象的概念后,本节将介绍如何取得向量内的元素,由先前实例可以看到每一个数据输出时,输出数据左边均有“[1]”,中括号内的“1”代表索引值,表示是向量对象的第1个元素。R语言与C语言不同,它的索引(index)值是从1开始(C语言从0开始)的。

实例 ch4_46:认识向量对象的索引。

```
> numbers_List <- 25:1
> numbers_List
[1] 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4
[23] 3 2 1
>
```



在上述实例中，`numbers_List`向量对象的第1个元素是25，对应索引[1]，第2个元素是24，对应索引[2]，第23个元素是3，对应索引[23]。

实例 ch4_47：延续前一实例，分别从向量对象 `numbers_List` 取得第3个数据、第19个数据和第24个数据的实例。

```
> numbers_List[3]
[1] 23
> numbers_List[19]
[1] 7
> numbers_List[24]
[1] 2
>
```

上述只是很普通的命令，R语言的酷炫之处在于索引也可以是一个向量对象，这个向量对象可用`c()`函数建立起来。所以可以用下列简单的命令取代上述命令，取得索引值为3、19和24的值。

实例 ch4_48：延续前一实例，索引也可以是向量的应用实例。

```
> numbers_List[c(3, 19, 24)]
[1] 23 7 2
>
```

此外，我们也可以用下列已建好的向量对象当作索引取代上述实例。

实例 ch4_49：延续前一实例，索引也可以是向量对象的另一个应用实例。

```
> index_List <- c(3, 19, 24)
> numbers_List[index_List]
[1] 23 7 2
>
```

其实上述利用索引取得原向量部分元素(也可称子集)的过程为取子集(subsetting)。

4-6-2 使用负索引挖掘向量对象内的部分元素

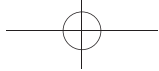
我们可以利用索引取得向量对象的元素，也可以利用索引取得向量对象内不含特定索引所对应的部分元素，方法是使用负索引。

实例 ch4_50：取得向量对象内不含第2个元素的所有其他元素。

```
> numbers_List # 原先向量内容
[1] 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6
[21] 5 4 3 2 1
> numbers_List <- numbers_List[-2]
> numbers_List # 新向量内容
[1] 25 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
[21] 4 3 2 1
>
```

由上述实例可以看到新`number_List`向量对象不含元素内容24。此外，负索引也可以是一个向量对象，因此也可以利用此特性取得负索引向量对象所指以外的元素。

实例 ch4_51：负索引也可以是一个向量对象的应用，如下是取得第1个到第15个以外元素的实例。



```
> numbers_List      # 原先向量内容
[1] 25 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5
[21]  4  3  2  1
> numbers_List <- numbers_List[-(1:15)]
> numbers_List      # 新向量内容
[1] 9 8 7 6 5 4 3 2 1
>
```

需留意的是，索引内使用“-(1:15)”，而不是“-1:15”。可参考下列实例。

实例 ch4_52：错误使用索引的实例。

```
> numbers_List[-1:15]
Error in numbers_List[-1:15] : 只有负数下标中才能有 0
>
```

4-6-3 修改向量对象元素值

使用向量对象做数据记录时，难免会有错，碰上这类情况，可以使用本节的方法修改向量对象元素值。下列是将Jordan第2场进球数修改为8的实例。

实例 ch4_53：修改向量对象元素值的应用实例。

```
> baskets.NBA2016.Jordan      # 列出各场次的进球数
[1] 10  5  9 15  7 11
> baskets.NBA2016.Jordan[2] <- 8 # 修正第2场进球数为8
> baskets.NBA2016.Jordan      # 验证结果
[1] 10  8  9 15  7 11
>
```

从上述结果，可以看到第2场进球数已经修正为8球了。此外，上述修改向量对象的索引参数也可以是一个向量对象，例如，假设第1场和第6场，Jordan的进球数皆是12，此时可使用下列方式修正。

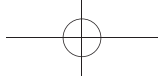
实例 ch4_54：一次修改多个向量对象元素的应用实例。

```
> baskets.NBA2016.Jordan      # 列出各场次的进球数
[1] 10  8  9 15  7 11
> baskets.NBA2016.Jordan[c(1, 6)] <- 12 # 修正新的进球数
> baskets.NBA2016.Jordan      # 验证结果
[1] 12  8  9 15  7 12
>
```

当修改向量对象元素数据时，原始数据就没了，所以建议各位读者，在修改前可以先建立一份备份，下列是实例。

实例 ch4_55：修改向量对象前，先做备份的应用实例。

```
> baskets.NBA2016.Jordan
[1] 12  8  9 15  7 12
> copy.baskets.NBA2016.Jordan <- baskets.NBA2016.Jordan
> baskets.NBA2016.Jordan
[1] 12  8  9 15  7 12
> copy.baskets.NBA2016.Jordan
[1] 12  8  9 15  7 12
>
```



实例 ch4_56 : 下列是将 Jordan 第 6 场进球数修改为 14 的实例。

```
> baskets.NBA2016.Jordan
[1] 12 8 9 15 7 12
> baskets.NBA2016.Jordan[6] <- 14
> baskets.NBA2016.Jordan
[1] 12 8 9 15 7 14
> copy.baskets.NBA2016.Jordan
[1] 12 8 9 15 7 12
>
```

由上述实例可以看到 Jordan 第 6 场进球数已经被修正为 14。如果现在想将 Jordan 的各场次进球数数据复原为原先备份的向量对象值，可参考下列实例。

实例 ch4_57 : 复原原先备份的向量对象的应用实例。

```
> baskets.NBA2016.Jordan          #列出各场次的进球数
[1] 12 8 9 15 7 14
> copy.baskets.NBA2016.Jordan      #列出原先备份向量值
[1] 12 8 9 15 7 12
> baskets.NBA2016.Jordan <- copy.baskets.NBA2016.Jordan    #回复原先的备份值
> baskets.NBA2016.Jordan          #验证结果
[1] 12 8 9 15 7 12
>
```

4-6-4 认识系统内建的数据集 letters 和 LETTERS

本小节将以 R 语言系统内建的数据集 letters 和 LETTERS 为例，讲解如何取得向量的部分元素或称取子集(subsetting)。

实例 ch4_58 : 认识系统内建的数据集 letters 和 LETTERS。

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
> LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
>
```

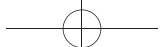
实例 ch4_59 : 取得 letters 对象索引值 10 和 18 所对应的元素。

```
> letters[c(10, 18)]
[1] "j" "r"
>
```

实例 ch4_60 : 取得 LETTERS 对象索引值 21 至 26 所对应的元素。

```
> LETTERS[21:26]
[1] "U" "V" "W" "X" "Y" "Z"
>
```

对前面的实例而言，由于我们知道有 26 个字母，所以可用“21:26”取得最后 6 个元素。但是有许多数据集，我们不知道它们的元素个数，应该怎么办？R 语言提供 tail() 函数，可解决这方面的困扰，可参考下列实例。



实例 ch4_61: 使用 `tail()` 函数取得 `LETTERS` 对象最后 8 个元素。并且测试, 如果省略第 2 个参数, 会列出多少个元素?

```
> tail(LETTERS, 8)
[1] "S" "T" "U" "V" "W" "X" "Y" "Z"
> tail(LETTERS)
[1] "U" "V" "W" "X" "Y" "Z"
>
```

由上述实例可知, `tail()` 函数的第一个参数是数据集的对象名称, 第二个参数是预计取得多少元素, 如果省略第二个参数, 系统自动返回 6 个元素。`head()` 函数使用方式与 `tail()` 函数相同, 但是返回数据集的最前面的元素。

实例 ch4_62: 使用 `head()` 函数取得 `LETTERS` 对象的前 8 个元素。并且测试, 如果省略第 2 个参数, 会列出多少个元素?

```
> head(LETTERS, 8)
[1] "A" "B" "C" "D" "E" "F" "G" "H"
> head(LETTERS)
[1] "A" "B" "C" "D" "E" "F"
>
```

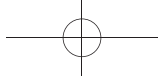
4-7 逻辑向量

4-7-1 基本应用

在先前介绍的函数运算中, 笔者偶尔穿插使用了 `TRUE` 和 `FALSE`, 这个值在 R 语言中被称为逻辑值, 这一节将对此做一个完整的说明。有些函数在使用时会传回 `TRUE` 或 `FALSE`, 例如, 3-4 节所介绍的 `is.finite()`、`is.infinite()`, 基本原则是, 如果函数执行结果是真, 则返回 `TRUE`, 如果是假, 则返回 `FALSE`。这两个值对于程序流程的控制很重要, 未来章节会对其做详细的说明。

本节主要介绍含逻辑值的向量对象, 当一个函数内的参数含有逻辑向量时, 整个 R 语言的设计将显得更灵活。R 语言可以用比较两个值的方式返回逻辑值。如表所示。

表达式	说明
<code>x == y</code>	如果 <code>x</code> 等于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x != y</code>	如果 <code>x</code> 不等于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x > y</code>	如果 <code>x</code> 大于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x >= y</code>	如果 <code>x</code> 大于或等于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x < y</code>	如果 <code>x</code> 小于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x <= y</code>	如果 <code>x</code> 小于或等于 <code>y</code> , 则传回 <code>TRUE</code>
<code>x & y</code>	相当于 AND 运算, 如果 <code>x</code> 和 <code>y</code> 皆是 <code>TRUE</code> 则传回 <code>TRUE</code>
<code>x y</code>	相当于 OR 运算, 如果 <code>x</code> 或 <code>y</code> 是 <code>TRUE</code> 则传回 <code>TRUE</code>
<code>!x</code>	相当于 NOT 运算, 则传回非 <code>x</code>
<code>xor(x, y)</code>	相当于 XOR 运算, 如果 <code>x</code> 和 <code>y</code> 不同, 则传回 <code>TRUE</code>



对于上述比较的表达式而言，**x**和**y**也可以是一个向量对象。

实例 ch4_63：下列实例是如果 **Jordan** 在比赛中的进球数高于 10 球则输出 **TRUE**，否则输出 **FALSE**。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> baskets.NBA2016.Jordan > 10
[1] TRUE FALSE FALSE TRUE FALSE TRUE
>
```

which() 函数所使用的参数是一个比较表达式，可以列出符合条件的索引值，相当于可以找出向量对象中的哪些元素是符合条件的。

实例 ch4_64：下列实例是列出 **Jordan** 进球超过 10 球的场次。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> which(baskets.NBA2016.Jordan > 10)
[1] 1 4 6
>
```

which.max()：可列出最大值的第1个索引值。

which.min()：可列出最小值的第1个索引值。

一个向量对象的最大值可能会出现好几次，分别对应不同的索引，**which.max()** 函数则只列出第1个出现的最大值所对应索引值，**which.min()** 与 **which.max()** 的含义相似，是列出第1个出现的最小值所对应索引值。

实例 ch4_65：下列实例是列出进球数最多和最少的场次。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> which.max(baskets.NBA2016.Jordan)
[1] 4
> which.min(baskets.NBA2016.Jordan)
[1] 5
>
```

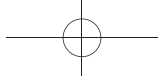
实例 ch4_66：下列是将 **Jordan** 和 **Lin** 做比较，同时列出 **Jordan** 进球数较多的场次。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> baskets.NBA2016.Lin        #了解Lin的各场次进球数
[1] 7 8 6 11 9 12
> best.baskets <- baskets.NBA2016.Jordan > baskets.NBA2016.Lin
> which(best.baskets)
[1] 1 3 4
>
```

在上述实例中，可以发现 **Jordan** 和 **Lin** 有两场比赛进球数相同，如果修改，列出 **Jordan** 与 **Lin** 进球数相同或 **Jordan** 进球数较多的场次，则可以参考下列实例。

实例 ch4_67：列出 **Jordan** 与 **Lin** 进球数相同或 **Jordan** 进球数较多的场次。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> baskets.NBA2016.Lin        #了解Lin的各场次进球数
[1] 7 8 6 11 9 12
> best.baskets <- baskets.NBA2016.Jordan >= baskets.NBA2016.Lin
> which(best.baskets)
[1] 1 2 3 4 6
>
```



当然我们也可以继续延伸使用**best.baskets**向量对象。

实例 ch4_68：下列实例是使用**best.baskets** 向量对象列出 Jordan 在得分较多或与 Lin 相同的比赛中的实际进球数，同时也列出 Lin 的进球数。

```
> baskets.NBA2016.Jordan[best.baskets]
[1] 12 8 9 15 12
> baskets.NBA2016.Lin[best.baskets]
[1] 7 8 6 11 12
>
```

4-7-2 Inf、-Inf和缺失值NA的处理

使用逻辑表达式进行筛选满足一定条件的值时，若是碰上NA，会如何呢？请看下列实例。

实例 ch4_69：NA 在逻辑表达式的应用。

```
> x <- c(9, 1, NA, 8, 6)
> x[x > 5]
[1] 9 NA 8 6
>
```

从上述实例看，好像是NA大于5，所以NA也返回。

非也。

任何比较，对于NA而言均是返回NA，可参考下列实例。

实例 ch4_70：NA 在逻辑表达式中的另一个应用。

```
> x <- c(9, 1, NA, 8, 6)
> x > 5
[1] TRUE FALSE NA TRUE TRUE
>
```

接下来考虑的是Inf和-Inf，可参考下列的实例。

实例 ch4_71：Inf 在逻辑表达式的应用。

```
> x <- c(9, 1, Inf, 8, 6)
> x[x > 5]
[1] 9 Inf 8 6
>
```

由上述实例可知，**Inf**的确大于5所以上述也返回**Inf**的索引。可以用下列实例验证这个结果。

实例 ch4_72：Inf 在逻辑表达式中的另一个应用。

```
> x <- c(9, 1, Inf, 8, 6)
> x > 5
[1] TRUE FALSE TRUE TRUE TRUE
>
```

很明显，当比较**Inf**是否大于5时，是返回**TRUE**的。接下来，下列是用-Inf测试的实例。

实例 ch4_73：-Inf 在逻辑表达式的应用。

```
> x <- c(9, 1, -Inf, 8, 6)
> x[x > 5]
[1] 9 8 6
> x > 5
[1] TRUE FALSE FALSE TRUE TRUE
>
```

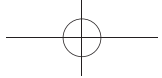
很明显，-Inf是小于5，所以返回**FALSE**。

4-7-3 多组逻辑表达式的应用

再度使用Jordan的进球数，下列实例可得到Jordan的最高进球数和最低进球数。

实例 ch4_74：得到 Jordan 最高进球数和最低进球数。

```
> baskets.NBA2016.Jordan          #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> max.baskets.Jordan <- max(baskets.NBA2016.Jordan)
> min.baskets.Jordan <- min(baskets.NBA2016.Jordan)
>
```



有了以上数据，可用下列方法求得某区间的数据。

实例 ch4_75：下列是列出不是最高进球数和最低进球数的场次和进球数。

```
> max.baskets.Jordan <- max(baskets.NBA2016.Jordan) #最高进球数
> min.baskets.Jordan <- min(baskets.NBA2016.Jordan) #最低进球数
> lower.baskets <- baskets.NBA2016.Jordan < max.baskets.Jordan #非最高进球场次
> upper.baskets <- baskets.NBA2016.Jordan > min.baskets.Jordan #非最低进球场次
> range.basket.Jordan <- lower.baskets & upper.baskets #我们要的区间场次
> which(range.basket.Jordan) #列出我们要的区间场次
[1] 1 2 3 6
> baskets.NBA2016.Jordan[range.basket.Jordan] #列出区间场次的进球数
[1] 12 8 9 12
>
```

由上述运算可知，lower.baskets是得到非最高进球数的场次[1, 2, 3, 5, 6]，upper.baskets是得到非最低进球数的场次[1, 2, 3, 4, 6]，接着我们用逻辑运算符“&”，可以得到非最高进球数与最低进球数的场次是[1, 2, 3, 6]。

4-7-4 NOT表达式

从4-7-2节的实例可知，若向量对象中含缺失值NA，会造成我们使用时的错乱，当碰上这类状况时，可先用“is.na()”函数判断是否含有NA，然后再用“!is.na()”，即可剔除NA，可参考下列实例。

实例 ch4_76：NOT表达式和is.na()函数的应用。

```
> x <- c(9, 1, NA, 8, 6)
> x[x > 5 & !is.na(x)]
[1] 9 8 6
>
```

若与4-7-2节的实例做比较，则可以看到NA被剔除了。

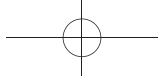
4-7-5 逻辑值TRUE和FALSE的运算

R语言和其他高级语言一样(例如C语言)，可以将TRUE视为1，将FALSE视为0使用。下列实例可列出，Jordan共有几场进球数比Lin多或一样多。

实例 ch4_77：列出Jordan共有几场进球数表现比Lin多或一样多。

```
> baskets.NBA2016.Jordan #了解Jordan的各场次进球数
[1] 12 8 9 15 7 12
> baskets.NBA2016.Lin #了解Lin的各场次进球数
[1] 7 8 6 11 9 12
> better.baskets <- baskets.NBA2016.Jordan >= baskets.NBA2016.Lin
> sum(better.baskets)
[1] 5
>
```

any()函数的用法是，只要参数向量对象有1个元素是TRUE，则传回TRUE。



实例 ch4_78 : any () 函数的应用。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12  8  9 15  7 12
> baskets.NBA2016.Lin         #了解Lin的各场次进球数
[1]  7  8  6 11  9 12
> better.baskets <- baskets.NBA2016.Jordan > baskets.NBA2016.Lin
> any(better.baskets)
[1] TRUE
> .
```

在上述实例中，笔者将better.baskets调整为Jordan的进球数需大于Lin的进球数，才传回TRUE。由于仍有3场比赛Jordan的进球数是大于Lin的进球数，所以any()函数返回TRUE。

另外一个常用函数是all()，用法是：所有参数需是TRUE，才传回TRUE。

实例 ch4_79 : all () 函数的应用。

```
> baskets.NBA2016.Jordan      #了解Jordan的各场次进球数
[1] 12  8  9 15  7 12
> baskets.NBA2016.Lin         #了解Lin的各场次进球数
[1]  7  8  6 11  9 12
> better.baskets <- baskets.NBA2016.Jordan >= baskets.NBA2016.Lin
> all(better.baskets)
[1] FALSE
> .
```

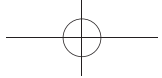
在上述实例，笔者将better.baskets调整为Jordan的进球数需大于或等于Lin的进球数，才传回TRUE。虽然有5场比赛Jordan的进球数大于Lin的进球数，但仍有1场比赛Jordan的进球数小于Lin的进球数，因此all()函数仍返回FALSE。

4-8 不同长度向量对象相乘的应用

在实例ch4_7和ch4_8中，笔者介绍了两个不同长度向量对象相加的实例，本节将讲解两个不同长度向量对象相乘的应用实例，不同长度向量对象相乘的基本原则是，长的向量对象长度是短的向量对象长度的倍数，本节将直接以实例作说明。

实例 ch4_80 : 假设baskets.Balls.Jordan向量对象，奇数元素是单场2分球的进球数，偶数元素是单场3分球的进球数，请由此数据求出Jordan的总得分及平均得分。

```
> #列出6场球赛2分球和3分球的进球数
> baskets.Balls.Jordan <- c(12, 3, 8, 2, 9, 4, 15, 5, 7, 2, 12, 3)
> scores.Jordan <- baskets.Balls.Jordan * c(2, 3)      # 计算得分向量
> scores.Jordan                                         # 列出得分向量
[1] 24  9 16  6 18 12 30 15 14  6 24  9
> sum(scores.Jordan)                                   # 列出Jordan 6场比赛总得分
[1] 183
> scores.Average.Jordan <- sum(scores.Jordan) / 6      # 求出Jordan 6场比赛平均得分
> scores.Average.Jordan                               # 列出Jordan 6场比赛平均得分
[1] 30.5
> .
```



由上述实例可以看到**baskets.Balls.Jordan**的奇数元素会乘**c(2, 3)**中的2，偶数元素会乘**c(2, 3)**中的3，所以可以产生得分**scores.Jordan**向量对象，其中奇数元素是2分球产生的分数，偶数元素是3分球产生的分数。接着可以很轻松地计算6场比赛的总得分和平均得分。

4-9 向量对象的元素名称

4-9-1 建立简单含元素名称的向量对象

虽然我们可以使用索引很方便地取得向量对象的元素，但R语言有一个强大的功能是为向量对象的每一个元素命名，未来我们也可以利用对象的元素名称引用元素内容。下列是建立向量对象，同时给对象元素命名的方法。

```
object <- c(name1 = data1, name2 = data2 ...)
```

实例 ch4_81：为Jordan的前三场NBA得分，建立一个含元素名称的向量对象。在本实例中，除了建立此含元素名称的向量对象**baskets.NBA.Jordan**外，同时列出各元素名称、元素值和此对象的结构。

```
> baskets.NBA.Jordan <- c(first = 28, second = 31, third = 35)
> baskets.NBA.Jordan[1]
first
28
> baskets.NBA.Jordan[2]
second
31
> baskets.NBA.Jordan[3]
third
35
> str(baskets.NBA.Jordan)
Named num [1:3] 28 31 35
- attr(*, "names")= chr [1:3] "first" "second" "third"
>
```

4-9-2 names()函数

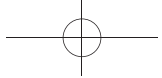
使用**names()**函数可以查询向量对象元素的名称，也可更改向量元素的名称。

实例 ch4_82：查询前一实例所建的元素名称。

```
> names(baskets.NBA.Jordan)
[1] "first" "second" "third"
>
```

names()函数也可以用来修改元素名称。

实例 ch4_83：修改对象**baskets.NBA.Jordan**的元素名称，并验证结果。



```
> names(baskets.NBA.Jordon) <- c("Game1", "Game2", "Game3") # 修改元素名称
> baskets.NBA.Jordon
Game1 Game2 Game3
   28   31   35
>
```

如果想要删除向量对象的元素名称，只要将其设为NULL即可，例如下列命令可以将上述实例所建向量对象baskets.NBA.Jordon的元素名称删除。

```
names(baskets.NBA.Jordon) <- NULL
```

month.name是系统内建一个数据集，此向量对象内容如下所示。

```
> month.name
[1] "January" "February" "March" "April" "May"
[6] "June" "July" "August" "September" "October"
[11] "November" "December"
>
```

有了以上数据集，我们可以用另一种方式为向量建立元素名称。

实例 ch4_84：建立一个月份表，这个月份表的元素含当月月份的英文名称和当月天数。

```
> month.data <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
> names(month.data) <- month.name
> month.data # 列出结果
  January February  March   April    May    June    July   August
       31        28       31       30     31     30     31       31
September  October November December
       30        31       30       31
>
```

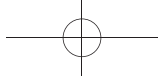
实例 ch4_85：列出天数为30天的月份。

```
> names(month.data[month.data == 30])
[1] "April" "June" "September" "November"
>
```

4-9-3 使用系统内建的数据集islands

这个数据集含有全球48个岛屿的名称及面积，其内容如下所示。

```
> islands
      Africa      Antarctica      Asia      Australia
      11506      5500      16988      2968
Axel Heiberg      Baffin      Banks      Borneo
      16      184      23      280
Britain      Celebes      Celon      Cuba
      84      73      25      43
Devon      Ellesmere      Europe      Greenland
      21      82      3745      840
Hainan      Hispaniola      Hokkaido      Honshu
      13      30      30      89
Iceland      Ireland      Java      Kyushu
      40      33      49      14
Luzon      Madagascar      Melville      Mindanao
      42      227      16      36
Moluccas      New Britain      New Guinea      New Zealand (N)
      29      15      306      44
New Zealand (S)      Newfoundland      North America      Novaya Zemlya
      58      43      9390      32
Prince of Wales      Sakhalin      South America      Southampton
      13      29      6795      16
Spitsbergen      Sumatra      Taiwan      Tasmania
      15      183      14      26
Tierra del Fuego      Timor      Vancouver      Victoria
      19      13      12      82
>
```



上述数据集是依照英文字母排列此数据元素的，下列是一系列取此数据集子集的实例。

实例 ch4_86：取子集并依岛屿大小从大到小排列。

```
> newislands <- sort(islands, decreasing = TRUE)
> newislands
```

Asia	Africa	North America	South America
16988	11506	9390	6795
Antarctica	Europe	Australia	Greenland
5500	3745	2968	840
New Guinea	Borneo	Madagascar	Baffin
306	280	227	184
Sumatra	Honshu	Britain	Ellesmere
183	89	84	82
Victoria	Celebes	New Zealand (S)	Java
82	73	58	49
New Zealand (N)	Cuba	Newfoundland	Luzon
44	43	43	42
Iceland	Mindanao	Ireland	Novaya Zemlya
40	36	33	32
Hispaniola	Hokkaido	Moluccas	Sakhalin
30	30	29	29
Tasmania	Celon	Banks	Devon
26	25	23	21
Tierra del Fuego	Axel Heiberg	Melville	Southampton
19	16	16	16
New Britain	Spitsbergen	Kyushu	Taiwan
15	15	14	14
Hainan	Prince of Wales	Timor	Vancouver
13	13	13	12

实例 ch4_87：取面积最小的10个岛屿。

```
> small10.islands <- tail(sort(islands, decreasing = TRUE), 10)
> small10.islands
```

Melville	Southampton	New Britain	Spitsbergen
16	16	15	15
Kyushu	Taiwan	Hainan	Prince of Wales
14	14	13	13
Timor	Vancouver		
13	12		

```
>
```

如果只想取得岛屿的名称，可参考下列实例。

实例 ch4_88：取面积最大的10个岛屿的名称，只列出名称。

```
> big10.islands <- names(head(sort(islands, decreasing = TRUE), 10))
> big10.islands
```

[1] "Asia"	"Africa"	"North America"	"South America"
[5] "Antarctica"	"Europe"	"Australia"	"Greenland"
[9] "New Guinea"	"Borneo"		

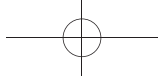
```
>
```

实例 ch4_89：以不用 head() 函数的方式，完成前一个实例。

```
> big10.islands <- names(sort(islands, decreasing = TRUE)[1:10])
> big10.islands
```

[1] "Asia"	"Africa"	"North America"	"South America"
[5] "Antarctica"	"Europe"	"Australia"	"Greenland"
[9] "New Guinea"	"Borneo"		

```
>
```



本章习题

一. 判断题

() 1. 有如下两个命令。

```
> x <- -2.5:-3.9  
> length(x)
```

上述命令执行结果如下所示。

```
[1] 3
```

() 2. 有如下两个命令。

```
> x <- 1:3  
> y <- x + 9:11
```

上述命令执行后，下列的执行结果是正确的。

```
> y  
[1] 10 11 12
```

() 3. 下列命令在执行时会出现Warning message。

```
> x <- 1:5  
> y <- x + 1:10
```

() 4. 在R语言的Console窗口，若某行命令以数学符号(+、-、*、/)做结尾，此时R语言的编译程序会知道下一行命令是接续此行。

() 5. 有如下两个命令。

```
> x <- c(7, 12, 6, 20, 9)  
> sort(x)
```

上述命令执行结果如下所示。

```
[1] 20 12 9 7 6
```

() 6. 有如下命令。

```
> sum(c(99, NA, 101, NA), na.rm = TRUE)
```

上述命令执行时会有错误信息产生。

() 7. 字符串是可以双引号(“”)也可以用单引号(‘’)括起来的。

() 8. 有如下4个命令。

```
> x1 <- c(1:2)  
> x2 <- c(1.5:2.5)  
> x3 <- c(x1, x2)  
> class(x3)
```

上述命令的执行结果如下所示。

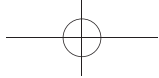
```
[1] "numeric"
```

() 9. 有如下两个命令。

```
> x <- 1:5  
> x[-(2:5)]
```

上述命令的执行结果如下所示。

```
[1] 1
```



() 10. 有如下两个命令。

```
> head(letters)
[1] "a" "b" "c" "d" "e" "f"
> letters[c(1, 5)]
```

上述命令的执行结果如下所示。

```
[1] "e"
```

() 11. 有如下两个命令。

```
> x <- c(10, NA, 3, 8)
> x[x > 6]
```

上述命令的执行结果如下所示。

```
[1] 10 NA 8
```

() 12. 有如下3个命令。

```
> x <- c(10, Inf, 3, 8)
> y <- x > 6
> any(y)
```

上述命令的执行结果如下所示。

```
[1] FALSE
```

() 13. 有如下3个命令。

```
> x <- c(5, 7)
> names(x) <- c("Game1", "Game2")
> names(x) <- NULL
```

上述命令相当于是将x向量对象的元素值设为0。

() 14. 有如下两个命令。

```
> x.small <- names(head(sort(islands)))
> y.small <- names(sort(islands)[1:6])
```

上述x.small和y.small两个向量对象的内容相同。

() 15. R语言逻辑运算的结果只可能有两种：TRUE或者FALSE。

() 16. 有如下命令。

```
> x[ is.na(x) ] <- 0
```

上述命令执行后，会将x对象内的所有缺失值以0替代。

() 17. 有如下命令。

```
> x <- seq(-10, 10, 15)
```

上述命令执行后，x向量对象的最大值是10。

二. 单选题

() 1. 假设有n个字母，想了解这n个字母的排列组合方法，下列哪一个函数可以最方便解这类问题？

A. max()

B. mean()

C. sd()

D. prod()

() 2. 以下命令会得到以下哪个数值结果？

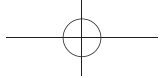
```
> x <- 1:3
> y <- x + 1:6
> y
```

A. [1] 1 3 5

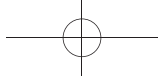
B. [1] 2 4 5

C. [1] 2 4 6 5 7 9

D. [1] 2 4 5 6 8 9



- () 3. 以下命令会得到以下哪个数值结果？
`> seq(1, 9, length.out = 5)`
A. [1] 1 3 5 7 9
B. [1] 1 6
C. [1] 1 2 3 4 5 6
D. [1] 5 6 7 8 9
- () 4. 以下数值结果来自以下哪个命令？
[1] 2 2 2
A. `> rep(3, 2)`
B. `> rep(2, 3)`
C. `> rep(2, 2, 2)`
D. `> rep(3, 2, 2)`
- () 5. 以下命令会得到以下哪个数值结果？
`> x <- mean(8:12)`
`> x`
A. [1] 10
B. [1] 8
C. [1] 12
D. [1] 5
- () 6. 以下命令会得到以下哪个数值结果？
`> x <- c(12, 7, 8, 4, 19)`
`> rank(x)`
A. [1] 12 7 8 4 19
B. [1] 4 7 8 12 19
C. [1] 4 2 3 1 5
D. [1] 19 12 8 7 4
- () 7. 以下命令会得到以下哪个数值结果？
`> max(c(9, 99, Inf, NA))`
A. [1] 9
B. [1] 99
C. [1] Inf
D. [1] NA
- () 8. 以下命令会得到以下哪个数值结果？
`> max(c(9, 99, Inf, NA), na.rm = TRUE)`
A. [1] 9
B. [1] 99
C. [1] Inf
D. [1] NA
- () 9. 以下命令会得到以下哪个数值结果？
`> x <- c("Hi!", "Good", "Morning")`
`> nchar(x)`
A. [1] 3 4 7
B. [1] 3
C. [1] 14
D. [1] 7 7
- () 10. 以下命令会得到以下哪个数值结果？
`> head(letters, 5)`
[1] "a" "b" "c" "d" "e"
`> letters[c(1, 5)]`
A. [1] "a"
B. [1] "a" "e"
C. [1] "b"
D. [1] "b" "c" "d"
- () 11. 以下命令会得到以下哪个数值结果？
`> x <- c(8, 12, 19, 4, 5)`
`> which.max(x)`
A. [1] 19
B. [1] 3
C. [1] 4
D. [1] 5
- () 12. 以下命令会得到以下哪个数值结果？
`> x <- c(6, 9, NA, 4, 2)`
`> x[x > 5 & !is.na(x)]`
A. [1] 6 9
B. [1] 6 9 NA
C. [1] 6 9 NA 4 2
D. [1] 4 2



() 13. 有以下命令。

```
> x1 <- c(9, 6, 8, 3, 4)
> x2 <- c(6, 10, 1, 2, 5)
> y <- x1 >= x2
```

将y放进哪一个函数内可以得到下列结果？

```
[1] FALSE
```

- A. any() B. rev()
C. sort() D. all()

() 14. 使用head()或tail()函数，若省略第2个参数，系统将自动返回多少个元素？

- A. 1 B. 3 C. 5 D. 6

() 15. 有以下命令。

```
> x <- 1:10
> names(x) <- letters[x]
> x
  a b c d e f g h i j
1 2 3 4 5 6 7 8 9 10
```

以下哪种方法不能传回x向量的前5个元素？即：

```
a b c d e
1 2 3 4 5
```

- A. x[“a”, “b”, “c”, “d”, “e”] B. x[1:5]
C. head(x, 5) D. x[letters[1:5]]

() 16. 以下命令集会得到以下哪个数值结果？

```
> x <- seq(-2, 2, 0.5)
> length(x)
```

- A. [1] 5 B. [1] 9 C. [1] 2 D. [1] 8

() 17. 以下命令集会得到以下哪个数值结果？

```
> c(3, 2, 1) == 2
```

- A. [1] TRUE B. [1] FALSE
C. [1] FALSE TRUE FALSE D. [1] NA

三. 多选题

() 1. 以下哪些方式可以用来计算1, 2, 3, 4的平均值？执行结果如下所示。(选择2项)

```
[1] 2.5
```

- A. mean(1, 2, 3, 4) B. mean(c(1, 2, 3, 4))
C. sum(c(1, 2, 3, 4))/4 D. max(c(1, 2, 3, 4))
E. ave(c(1, 2, 3, 4))

() 2. 以下哪些函数可以用来产生如下x向量？(选择3项)

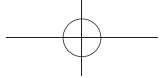
```
[1] 1 2 3 4 5 6 7 8 9 10
```

- A. seq(10) B. seq_len(10) C. numeric(10)
D. 1:10 E. seq(1,10,10)

四. 实际操作题(如果题目有描述不详细时，请自行假设条件)

1. 建立家人的向量数据。

(1) 将家人或亲人(至少10人)的名字建立为字符向量对象，同时为每一个元素建立名称，并



打印出来。

```
> fname
[1] "Austin" "Ben" "Charlie" "Danial" "Ellen" "Frank"
[7] "Golden" "Helen" "Ivan" "Jessie"
```

(2) 将家人血型(至少10人)建立为字符串向量对象, 可用英文, 同时为每一个元素建立名称, 并打印出来。

```
> fblood
Austin Ben Charlie Danial Ellen Frank Golden Helen
  "A"   "O"   "O"   "B"   "O"   "B"   "A"   "AB"
Ivan Jessie
  "O"   "O"
```

(3) 将家人或亲人(至少10人)的年龄建立为整数向量对象, 同时为每一个元素建立名称, 并打印出来。

```
> fage
Austin Ben Charlie Danial Ellen Frank Golden Helen
    22   23    21    20    20    19    18    18
Ivan Jessie
   19    20
```

(4) 将上述所建的年龄向量, 执行从小排序到大。

```
> agesort
Golden Helen Frank Ivan Danial Ellen Jessie Charlie
    18    18    19    19    20    20    20    21
Austin Ben
    22    23
```

(5) 将上述所建的年龄向量, 执行从大排序到小。

```
> reagesort
Ben Austin Charlie Danial Ellen Jessie Frank Ivan
    23    22    21    20    20    20    19    19
Golden Helen
    18    18
```

2. 参考实例ch4_84, 列出当月有31天的月份。

```
> names(month.data[month.data==31])
[1] "January" "March" "May" "July" "August" "October"
[7] "December"
```

3. 使用系统内建数据集islands, 列出面积排序第30和35名的岛名称和面积。

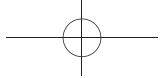
```
New Zealand (S) Honshu
              58      89
```

4. 使用系统内建数据集islands, 列出面积排序前15和最后15的岛名称和面积。

```
Vancouver Hainan Prince of Wales Timor
      12      13      13      13
Kyushu Taiwan New Britain Spitsbergen
    14      14      15      15
Axel Heiberg Melville Southampton Tierra del Fuego
    16      16      16      19
Devon Banks Celon
    21      23      25
```

5. 使用系统内建数据集islands, 分别列出排在奇数位和偶数位的岛名称和面积。

```
Britain Honshu Sumatra Baffin
      84      89      183     184
Madagascar Borneo New Guinea Greenland
      227      280      306      840
Australia Europe Antarctica South America
      2968      3745      5500      6795
North America Africa Asia
      9390      11506      16988
```



6. 使用系统内建数据集islands，分别列出排序奇数的岛名称和面积。

Vancouver	Prince of Wales	Kyushu	New Britain
12	13	14	15
Axel Heiberg	Southampton	Devon	Celon
16	16	21	25
Moluccas	Hispaniola	Novaya Zemlya	Mindanao
29	30	32	36
Luzon	Newfoundland	Java	Celebes
42	43	49	73
Victoria	Honshu	Baffin	Borneo
82	89	184	280
Greenland	Europe	South America	Africa
840	3745	6795	11506

7. 使用系统内建数据集islands，分别列出排序偶数的岛名称和面积。

Hainan	Timor	Taiwan	Spitsbergen
13	13	14	15
Melville	Tierra del Fuego	Banks	Tasmania
16	19	23	26
Sakhalin	Hokkaido	Ireland	Iceland
29	30	33	40
Cuba	New Zealand (N)	New Zealand (S)	Ellesmere
43	44	58	82
Britain	Sumatra	Madagascar	New Guinea
84	183	227	306
Australia	Antarctica	North America	Asia
2968	5500	9390	16988