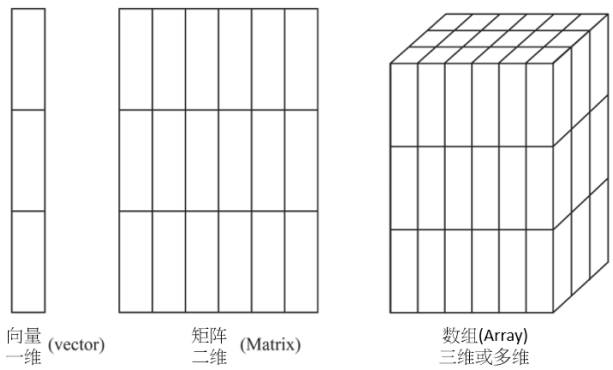
第5章 处理矩阵与更高维数据

向量(vector)对象相当于是Microsoft Excel表格的一行(row)或一列(column)，同时存放着相同类型的数据。在真实的世界里，这是不够的，我们常碰上需要处理不同类型的数据的情况。

在数据中，一维数据称向量，二维数据称矩阵(matrix)，超过二维的数据称三维或多维数组(array)对象。如下图所示。



此外，也可将向量称一维数组，将矩阵称二维数组，其余则依维度数称N维数组。

5-1 矩阵

若是将向量想成线，可将矩阵想成面。对R语言程序设计师而言，首先要思考的是如何建立矩阵。

5-1-1 建立矩阵

建立矩阵可使用matrix( )函数，格式如下：

```
matrix(data, nrow =, ncol =, byrow = logical, dimnames = NULL)
```

- data：数据。
- nrow：预计行的数量。
- ncol：预计列的数量。
- byrow：逻辑值。预设是FALSE，表示先按列填数据，第1列填满再填第2列，其他依此类推，因此，若先填列则可省略此参数。如果该值是TRUE则按行填数据，即第1行填满再填第2行，其他依此类推。
- dimnames：矩阵的属性，即行名和列名。

实例ch5_1：建立first.matrix，数据为1:12，4行的矩阵。

```
> first.matrix <- matrix(1:12, nrow = 4)
> first.matrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
```

实例ch5_2：建立second.matrix，数据为1:12，4行矩阵，byrow设为TRUE。

```
> second.matrix <- matrix(1:12, nrow = 4, byrow = TRUE)
> second.matrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
>
```

实例ch5_3：建立third.matrix，数据为1:12，4行矩阵，byrow设为FALSE。这个实例的执行结果与ch5_1相同。

```
> third.matrix <- matrix(1:12, nrow = 4, byrow = FALSE)
> third.matrix
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
```

5-1-2 认识矩阵的属性

str( )函数也可以查看矩阵对象的结构。

实例ch5_4：使用str( )函数查看first.matrix和second.matrix的结构。

```
> str(first.matrix)
 int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
> str(second.matrix)
 int [1:4, 1:3] 1 4 7 10 2 5 8 11 3 6 ...
>
```

使用nrow( )函数可以得到矩阵的行数。

实例ch5_5：使用nrow( )函数查看first.matrix和second.matrix的行数。

```
> nrow(first.matrix)
[1] 4
> nrow(second.matrix)
[1] 4
>
```

使用ncol( )函数可以得到矩阵的列数。

实例ch5_6：使用ncol( )函数查看first.matrix和second.matrix的列数。

```
> ncol(first.matrix)
[1] 3
> ncol(second.matrix)
[1] 3
>
```

使用dim( )函数则可以获得矩阵对象的行数和列数。

实例ch5_7：使用dim( )函数查看first.matrix和second.matrix的行数和列数。

```
> dim(first.matrix)
[1] 4 3
> dim(second.matrix)
[1] 4 3
>
```

此外，length( )函数也可用于取得矩阵或三维数组或多维数组对象的元素个数。

实例ch5_8：取得first.matrix和second.matrix的元素个数。

```
> length(first.matrix)
[1] 12
> length(second.matrix)
[1] 12
>
```

is.matrix( )函数可用于检查对象是否是矩阵。

实例ch5_9：检查first.matrix和second.matrix是否是矩阵。

```
> is.matrix(first.matrix)
[1] TRUE
> is.matrix(second.matrix)
[1] TRUE
>
```

is.array( )函数可用于检查对象是否是数组。

实例ch5_10：检查first.matrix和second.matrix是否是数组。

```
> is.array(first.matrix)
[1] TRUE
> is.array(second.matrix)
[1] TRUE
>
```

5-1-3 将向量组成矩阵

R语言提供的rbind( )函数可将两个或多个向量组成矩阵，每个向量各自占用一行。

实例ch5_11：使用rbind( )函数，简单地将两个向量组成矩阵的实例。

```
> v1 <- c(7, 11, 15)            # 向量1
> v2 <- c(5, 10, 9)             # 向量2
> a1 <- rbind(v1, v2)           # 組合
> a1
   [,1] [,2] [,3]
v1    7   11   15
v2    5   10    9
>
```

由以上程序可以看到矩阵左边保留了原向量对象的名称，后面章节会介绍如何使用这个向量名称。

实例ch5_12：矩阵也可以和向量组合成矩阵。

```
> v3 <- c(3, 6, 12)             # 向量3
> a2 <- rbind(a1, v3)           # 組合
> a2
   [,1] [,2] [,3]
v1    7   11   15
v2    5   10    9
v3    3    6   12
>
```

在上一章笔者讲解了有关baskets.NBA2016.Jordon和baskets.NBA2016.Lin这两个向量对象，下列是将这两个对象组成矩阵的实例。

实例ch5_13：将baskets.NBA2016.Jordon和baskets.NBA2016.Lin组成矩阵的实例。

```
> baskets.NBA2016.Lin
[1]  7  8  6 11  9 12
> baskets.NBA2016.Jordon
[1] 12  8  9 15  7 12
> baskets.NBA2016.Team <- rbind(baskets.NBA2016.Lin, baskets.NBA2016.Jordon)
> baskets.NBA2016.Team
                      [,1] [,2] [,3] [,4] [,5] [,6]
baskets.NBA2016.Lin      7    8    6   11    9   12
baskets.NBA2016.Jordon  12    8    9   15    7   12
>
```

cbind( )函数可将两个或多个向量组成矩阵，功能类似rbind( )。不过，它是以每个向量各占一列的方式来组织向量的。

实例ch5_14：使用cbind( )函数重新设计实例ch5_11。

```
> v1 <- c(7, 11, 15)              # 向量1
> v2 <- c(5, 10, 9)               # 向量2
> a3 <- cbind(v1, v2)             # 組合
> a3
     v1 v2
[1,]  7  5
[2,] 11 10
[3,] 15  9
>
```

实例ch5_15：使用cbind( )将两个向量与一个矩阵组成矩阵的应用实例。

```
> cbind(1:3, 11:13, matrix(21:26, nrow = 3))
     [,1] [,2] [,3] [,4]
[1,]    1   11   21   24
[2,]    2   12   22   25
[3,]    3   13   23   26
>
```

5-2 取得矩阵元素的值

使用索引执行矩阵元素的存取与上一章所述的存取向量元素的方法类似。

5-2-1 矩阵元素的取得

与向量相同，索引值必须在中括号内，中括号中的第一个参数是行，第二个是列。

实例ch5_16：使用实例ch5_12所建矩阵对象a2，取得a2[2, 1]和a2[1, 3]对应的值。

```
> a2
   [,1] [,2] [,3]
v1    7   11   15
v2    5   10    9
v3    3    6   12
> a2[2, 1]
v2
 5
> a2[1, 3]
v1
15
>
```

在取得矩阵元素内容时，如果原矩阵有行名或列名，那么行名与列名也将同时列出。假设有一个my.matrix矩阵，其内容如下。

```
> my.matrix <- matrix(1:20, nrow = 4)
> my.matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
>
```

下列是一系列取得my.matrix矩阵内容值的实例(ch5_17至ch5_22)。

实例ch5_17：取得my.matrix[3, 5]的实例。

```
> my.matrix[3, 5]
[1] 19
>
```

实例ch5_18：取得my.matrix[2,]，相当于取得第2行的所有元素。

```
> my.matrix[2, ]
[1]  2  6 10 14 18
>
```

注：当某一索引被省略时，则代表该维度的行或列均必须被计算在内。

实例ch5_19：取得my.matrix[, 3]，相当于取得第3列的所有元素。

```
> my.matrix[ , 3]
[1]  9 10 11 12
>
```

实例ch5_20：取得my.matrix[2, c(3,4)]，相当于取得第2行第3列和第4列的元素。

```
> my.matrix[2, c(3,4)]
[1] 10 14
>
```

也可将上述命令改写成下列的命令格式。

```
> my.matrix[2, 3:4]
[1] 10 14
>
```

实例ch5_21：取得my.matrix[3:4, 4:5]，相当于取得第3行到第4行和第4列到第5列的元素。所取得的也是一个矩阵。

```
> my.matrix[3:4, 4:5]
     [,1] [,2]
[1,]   15   19
[2,]   16   20
>
```

实例ch5_22：取得第3行和第4行的所有元素。

```
> my.matrix[3:4, ]
     [,1] [,2] [,3] [,4] [,5]
[1,]    3    7   11   15   19
[2,]    4    8   12   16   20
>
```

5-2-2 使用负索引取得矩阵元素

对于矩阵，使用负索引，相当于拿掉负索

引所指的行或列。

实例 ch5_23：取得第 3 行，第 4 列以外的所有元素。

```
> my.matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> my.matrix[-3, -4]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   17
[2,]    2    6   10   18
[3,]    4    8   12   20
>
```

实例 ch5_24：取得第 3 行和第 4 行，第 4 列以外的所有元素。

```
> my.matrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> my.matrix[-c(3:4), -4]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   17
[2,]    2    6   10   18
>
```

5-3 修改矩阵的元素值

修改矩阵的值与修改向量的元素值类似。

实例 ch5_25：将 my.matrix[3, 2] 的值为 100。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3  100   11   15   19
[4,]    4    8   12   16   20
> my.matrix[3, 2] <- 100       # 修改
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3  100   11   15   19
[4,]    4    8   12   16   20
>
```

我们也可以直接更改整行或整列的元素值。

实例 ch5_26：修改 my.matrix 矩阵，将第 3 行的元素值都改成 101 的应用实例。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3  100   11   15   19
[4,]    4    8   12   16   20
> my.matrix[3,  ] <- 101
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]  101  101  101  101  101
[4,]    4    8   12   16   20
>
```

实例 ch5_27：修改 my.matrix 矩阵，将整个第 4 列的元素值修改的应用实例。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]  101  101  101  101  101
[4,]    4    8   12   16   20
> my.matrix[ , 4] <-  c(3, 9)
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9    3   17
[2,]    2    6   10    9   18
[3,]  101  101  101    3  101
[4,]    4    8   12    9   20
>
```

实例 ch5_28：修改 my.matrix 矩阵，将整个第 4 列元素值修改的应用实例。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9    3   17
[2,]    2    6   10    9   18
[3,]  101  101  101    3  101
[4,]    4    8   12    9   20
> my.matrix[ , 4] <-  c(25:28)
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101  101  101   27  101
[4,]    4    8   12   28   20
>
```

实例 ch5_29：修改矩阵子集的应用实例，这个实例将修改 my.matrix[3:4, 2:3]。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101  101  101   27  101
[4,]    4    8   12   28   20
> my.matrix[3:4 , 2:3] <-  c(10, 31, 22, 99)
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101   10   22   27  101
[4,]    4   31   99   28   20
> .
```

实例 ch5_30：用一个小矩阵，修改原矩阵的子集。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101   10   22   27  101
[4,]    4   31   99   28   20
> my.matrix[3:4 , 2:3] <-  matrix(1:4, nrow = 2)
> my.matrix                    # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101    1    3   27  101
[4,]    4    2    4   28   20
>
```

实例 ch5_31：用一个小矩阵，修改原矩阵的子集，采用行优先排列的方式。

```
> my.matrix                    # 修改前
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101    1    3   27  101
[4,]    4    2    4   28   20
> my.matrix[3:4 , 2:3] <-  matrix(5:8, nrow = 2, byrow = TRUE)
> my.matrix   # 修改后
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   25   17
[2,]    2    6   10   26   18
[3,]  101    5    6   27  101
[4,]    4    7    8   28   20
>
```

5-4 降低矩阵的维度

使用负索引取得矩阵的部分元素时，如果所取得的部分元素仅有一行或一列，那么R语言将自动降低对象维度，使其从矩阵对象变向量对象。

实例 ch5_32：将 3 行 4 列的矩阵降为向量的应用实例，这个实例会舍弃第 2 行和第 3 行。

```
> simple.matrix <- matrix(1:12, nrow = 3)
> simple.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> simple.matrix[-c(2, 3), ]
[1]  1  4  7 10
>
```

其实，如果舍弃一个矩阵对象的某个元素，那么整个矩阵对象也将降为向量对象。

实例 ch5_33：将 3 行 4 列的矩阵降为向量的应用实例，这个实例会舍弃元素“2” 和“3”，最后整个矩阵将变为向量。

```
> simple.matrix <- matrix(1:12, nrow = 3)
> simple.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> simple.matrix[-c(2, 3)]
 [1]  1  4  5  6  7  8  9 10 11 12
>
```

假设有数行或数列的矩阵，其部分元素被舍弃，只剩一行或一列时，如果仍希望此对象以矩阵方式呈现，可增加“drop = FALSE”参数。

实例 ch5_34：类似实例 ch5_32 将 3 行 4 列的矩阵降为 1 行 4 列，但对象仍保持矩阵格式。

```
> simple.matrix <- matrix(1:12, nrow = 3)
> simple.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> simple.matrix[-c(2, 3), , drop = FALSE]
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
>
```

5-5 矩阵的行名和列名

其实直接输入矩阵对象名称就可以了解该矩阵对象的行名和列名。

实例 ch5_35：了解前一节所建的 simple.matrix 矩阵对象的行名和列名。

```
> simple.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>
```

从上述执行结果可知，simple.matrix没有行名和列名。

实例 ch5_36：了解程序实例 ch5_13 所建 baskets.NBA2016.TEAM 对象的行名和列名。

```
> baskets.NBA2016.Team
                    [,1] [,2] [,3] [,4] [,5] [,6]
baskets.NBA2016.Lin    7    8    6   11    9   12
baskets.NBA2016.Jordon  12    8    9   15    7   12
>
```

由上述执行结果可知，baskets.NBA2016.TEAM对象有2个行名，分别是baskets.NBA2016.Lin和baskets.NBA2016.Jordon。不过，此对象没有列名。

5-5-1 取得和修改矩阵对象的行名和列名

rownames( )函数可以取得和修改矩阵对象的行名。
colnames( )函数可以取得和修改矩阵对象的列名。

实例 ch5_37：使用 rownames( ) 函数取得 baskets.NBA2016.Team 和 simple.matrix 的行名。

```
> rownames(simple.matrix)
NULL
> rownames(baskets.NBA2016.Team)
[1] "baskets.NBA2016.Lin"    "baskets.NBA2016.Jordon"
>
```

从上述实例可知，我们已经使用rownames( )函数取得了baskets.NBA2016.Team的行名，但是名称似乎太长了，下一个实例是更改行名。

实例ch5_38：将矩阵对象baskets.NBA2016.Team的两个行名分别改成Lin和Jordon。

```
> rownames(baskets.NBA2016.Team) <- c("Lin", "Jordon")
> rownames(baskets.NBA2016.Team)
[1] "Lin"    "Jordon"
>
```

从实例ch5_36可知baskets.NBA2016.Team矩阵对象共有6列，其实每一列代表每一场球，我们可参考下列实例，设定对象的列名。

实例ch5_39：设定baskets.NBA2016.Team对象的列名。

```
> colnames(baskets.NBA2016.Team)    # 了解目前没有 列名
[1] "1st" "2nd" "3th" "4th" "5th" "6th"
> colnames(baskets.NBA2016.Team) <- c("1st", "2nd", "3th", "4th", "5th", "6th")  # 设定 列名
> colnames(baskets.NBA2016.Team)    # 验证结果
[1] "1st" "2nd" "3th" "4th" "5th" "6th"
> baskets.NBA2016.Team              # 另一方式验证结果
       1st 2nd 3th 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
>
```

如果我们想要修改某个列名，那么可参考下列实例。

实例ch5_40：将第4列的列名由“4th”改成“4”。本实例笔者会先复制一份矩阵对象baskets.NEW，然后再使用这份新的对象执行修改列名的操作。

```
> baskets.NBA2016.Team
       1st 2nd 3th 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> baskets.New <- baskets.NBA2016.Team
> colnames(baskets.New)[4] <- "4"
> baskets.New         # 验证结果
       1st 2nd 3th  4 5th 6th
Lin      7   8   6 11   9  12
Jordon  12   8   9 15   7  12
>
```

如果我们想要将整个列名或行名删除，那么只要将整个列名或行名设为NULL即可。

实例ch5_41：将baskets.New对象的列名删除。

```
> baskets.New         # 检查 列名
       1st 2nd 3th  4 5th 6th
Lin      7   8   6 11   9  12
Jordon  12   8   9 15   7  12
> colnames(baskets.New) <- NULL
> baskets.New         # 验证结果
       [,1] [,2] [,3] [,4] [,5] [,6]
Lin       7    8    6   11    9   12
Jordon   12    8    9   15    7   12
>
```

5-5-2 dimnames( )函数

行名和列名事实上是存在于dimnames的属性中，我们可以使用dimnames( )函数取得和修改这个属性值。

实例 ch5_42：使用 dimnames( ) 函数取得矩阵对象的行名和列名。

```
> dimnames(baskets.New)
[[1]]
[1] "Lin"    "Jordon"

[[2]]
NULL
```

由上述执行结果可以知道，目前baskets.New对象的两个行名分别是“Lin”“Jordon”，没有列名。

实例 ch5_43：使用 dimnames( ) 函数设定矩阵对象的列名。

```
> dimnames(baskets.New)[[2]] <- c("1st", "2nd", "3rd", "4th", "5th", "6th")
> dimnames(baskets.New)
[[1]]
[1] "Lin"    "Jordon"

[[2]]
[1] "1st" "2nd" "3rd" "4th" "5th" "6th"

>
```

5-6 将行名或列名作为索引

R语言的重要特色是，当一个矩阵有了行名和列名后，可以用这些名称代替数字型的索引，取得矩阵对象的元素，让整个程序的可读性更高。

实例 ch5_44：使用 baskets.New 对象，取得 Lin 第 3 场的进球数。

```
> baskets.New["Lin", "3rd"]
[1] 6
>
```

实例 ch5_45：使用 baskets.New 对象，取得 Jordon 第 2 场和第 5 场的进球数。

```
> baskets.New["Jordon", c("2nd", "5th")]
2nd 5th
  8   7
>
```

实例ch5_46：使用baskets.New对象，取得Jordon所有场次的进球数。

```
> baskets.New["Jordon", ]
1st 2nd 3rd 4th 5th 6th
 12   8   9  15   7  12
>
```

实例ch5_47：使用baskets.New对象，取得第5场所有球员的进球数。

```
> baskets.New[ , "5th"]
   Lin Jordon
     9      7
>
```

5-7 矩阵的运算

5-7-1 矩阵与一般常数的四则运算

当碰上矩阵对象与一般常数的运算时，只要将各个元素与该常数分别执行运算即可。在正式介绍实例前，笔者先建立下列m1.matrix矩阵。

```
> m1.matrix <- matrix(1:12, nrow = 3)
> m1.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

实例ch5_48:将m1.matrix矩阵加3的实例。

```
> m2.matrix <- m1.matrix + 3
> m2.matrix
     [,1] [,2] [,3] [,4]
[1,]    4    7   10   13
[2,]    5    8   11   14
[3,]    6    9   12   15
>
```

实例ch5_49:将m2.matrix矩阵减1的实例。

```
> m3.matrix <- m2.matrix - 1
> m3.matrix
     [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    4    7   10   13
[3,]    5    8   11   14
>
```

实例ch5_50:将m3.matrix矩阵乘5的实例。

```
> m4.matrix <- m3.matrix * 5
> m4.matrix
     [,1] [,2] [,3] [,4]
[1,]   15   30   45   60
[2,]   20   35   50   65
[3,]   25   40   55   70
>
```

实例ch5_51：将m4.matrix矩阵除以2的实例。

```
> m5.matrix <- m4.matrix / 2
> m5.matrix
     [,1] [,2] [,3] [,4]
[1,]  7.5 15.0 22.5 30.0
[2,] 10.0 17.5 25.0 32.5
[3,] 12.5 20.0 27.5 35.0
>
```

实例ch5_52：将m1.matrix加上m2.matrix，执行两个矩阵相加的实例。

```
> m6.matrix <- m1.matrix + m2.matrix
> m6.matrix
     [,1] [,2] [,3] [,4]
[1,]    5   11   17   23
[2,]    7   13   19   25
[3,]    9   15   21   27
>
```

注：两个矩阵能进行四则运算的先决条件是它们彼此的维度相同，否则会出现错误信

息。有意思的是，R语言是允许矩阵对象和向量对象相加的，只要矩阵的行数与向量长度相同即可，可参考下列实例。

实例ch5_53：矩阵与向量相加的运算实例。

```
> m1.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m7.matrix <- m1.matrix + 11:13
> m7.matrix
     [,1] [,2] [,3] [,4]
[1,]   12   15   18   21
[2,]   14   17   20   23
[3,]   16   19   22   25
>
```

如果矩阵的列数与向量长度相同，也可以进行相加运算，但一般不常用，读者可以自行测试了解。矩阵也可与向量相乘，只要向量长度与矩阵行数相同即可。

实例ch5_54：矩阵与向量相乘的运算实例。

```
> m1.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m8.matrix <- m1.matrix * 1:3
> m8.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    4   10   16   22
[3,]    9   18   27   36
>
```

上述命令在执行时，相当于矩阵第一行所有元素与向量的第一个元素相乘，此例是乘1。矩阵第二行所有元素与向量第二个元素相乘，此例是乘2。矩阵第三行的所有元素与向量的第三个元素相乘，此例是乘3。

注：“×”乘号是对单一元素逐步操作的，如果是要计算矩阵的内积，则需使用另一个特殊的矩阵相乘符号“ %×% ”，详细信息将在5-7-4节说明。

5-7-2 行和列的运算

在4-2节中笔者介绍了向量常用的函数sum( )和mean( )，这些函数已被修改可应用于矩阵。

- ◀ rowSums( )：计算行中元素的总和。
- ◀ colSums( )：计算列中元素的总和。
- ◀ rowMeans( )：计算行中元素的平均数。
- ◀ colMeans( )：计算列中元素的平均数。

实例ch5_55：利用rowSums( )和rowMeans( )函数，以及baskets.New对象计算Lin和Jordon的总进球数和平均进球数。

```
> baskets.New
       [,1] [,2] [,3] [,4] [,5] [,6]
Lin       7    8    6   11    9   12
Jordon   12    8    9   15    7   12
> rowSums(baskets.New)       # 计算总进球数
   Lin Jordon
    53     63
> rowMeans(baskets.New)      # 计算平均进球数
      Lin    Jordon
 8.833333 10.500000
>
```

使用上述rowSums( )和rowMeans( )函数一次可计算所有行的数据，假设只想要一个人的数据，可使用sum( )和mean( )函数。

实例ch5_56：利用sum( )和mean( )函数，以及baskets.New对象计算Lin的总进球数和平均进球数。

```
> baskets.New
       1st 2nd 3rd 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> sum(baskets.New["Lin", ])
[1] 53
> mean(baskets.New["Lin", ])
[1] 8.833333
>
```

实例ch5_57：使用baskets.New对象计算各场次的总进球数和平均每位球员的进球数。

```
> baskets.New
       1st 2nd 3rd 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> colSums(baskets.New)
1st 2nd 3rd 4th 5th 6th
 19  16  15  26  16  24
> colMeans(baskets.New)
 1st  2nd  3rd  4th  5th  6th
 9.5  8.0  7.5 13.0  8.0 12.0
>
```

使用上述colSums( )和colMeans( )函数一次可计算所有列的数据，假设只想要一场比赛的数据，可使用sum( )和mean( )函数。

实例ch5_58：使用baskets.New对象计算第3场次的总进球数和平均每位球员的进球数。

```
> baskets.New
       1st 2nd 3rd 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> sum(baskets.New[ , "3rd"])
[1] 15
> mean(baskets.New[ , "3rd"])
[1] 7.5
>
```

5-7-3 转置矩阵

t( )函数可执行矩阵转置，转置矩阵后，矩阵的行列元素将互相对调。

实例ch5_59：对baskets.New矩阵执行转置。

```
> baskets.New
       1st 2nd 3rd 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> t(baskets.New)
    Lin Jordon
1st   7     12
2nd   8      8
3rd   6      9
4th  11     15
5th   9      7
6th  12     12
>
```

5-7-4 %*%矩阵相乘

矩阵对象相乘的运算基本上和数学矩阵相乘是一样的。

实例ch5_60：分别使用“ * ”和“ %*% ”计算矩阵和向量的乘法。

```
> m1.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m9.matrix <- m1.matrix * 1:4
> m9.matrix
     [,1] [,2] [,3] [,4]
[1,]    1   16   21   20
[2,]    4    5   32   33
[3,]    9   12    9   48
> m10.matrix <- m1.matrix %*% 1:4
> m10.matrix
     [,1]
[1,]   70
[2,]   80
[3,]   90
>
```

读者可以试着比较上述运算结果。

实例5_61：两个3行3列的矩阵乘法的应用。

```
> m11.matrix <- matrix(1:9, nrow = 3)
> m11.matrix %*% m11.matrix
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
>
```

矩阵相乘时最常发生的错误是两个相乘矩阵的维度不符合矩阵运算原则，此时会出现 “非调和自变量”错误信息，如下列程序所示。

```
> n1 <- matrix(1:9, nrow = 3)
> n2 <- matrix(1:8, nrow = 2)
> n1 %*% n2
Error in n1 %*% n2 : 非调和自变量
>
```

5-7-5 diag( )函数

diag( )函数很灵活，当第一个参数是矩阵时，可传回矩阵对角线的向量值。

实例ch5_62：在各种不同维度的数组中，传回矩阵对角线的向量值。

```
> m1.matrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> diag(m1.matrix)
[1] 1 5 9
> baskets.New
       1st 2nd 3rd 4th 5th 6th
Lin      7   8   6  11   9  12
Jordon  12   8   9  15   7  12
> diag(baskets.New)
[1] 7 8
>
```

diag( )函数的另一个用法是传回矩阵，此矩阵对角线是使用第一个参数的向量值，其余填0。该命令的格式如下所示。

```
diag( x, nrow, ncol)
```

其中x是向量，nrow是矩阵行数，ncol是矩阵列数。若省略nrow和ncol则用x向量元素个数(假设是n)建立n行n列矩阵。

实例ch5_63：使用diag( )函数传回矩阵的实例。

```
> diag(1:5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    2    0    0    0
[3,]    0    0    3    0    0
[4,]    0    0    0    4    0
[5,]    0    0    0    0    5
> diag(1, 3, 3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(1, 2, 4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
> diag(1:2, 3, 4)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    1    0
```

5-7-6 solve( )函数

使用solve()这个函数可传回反矩阵，使用这个函数时要小心，有时会碰上小数被舍弃的问题。

实例ch5_64：反矩阵的应用。

```
> n3 <- matrix(1:4, nrow = 2)
> solve(n3)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
>
```

5-7-7 det( )函数

det是指数学中的行列式(determinant)，这个函数可以计算矩阵的行列式值。

实例ch5_65：det( )函数的应用。

```
> n3
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> det(n3)
[1] -2
>
```

5-8 三维或高维数组

在R语言中，如果将矩阵的维度加1，则得三维数组，这个维度是可视需要而持续增加的。虽然R程序设计师较少用到三维或更高维的数组数据结构，但在某些含时间序列的应用中，是有可能用到的。

5-8-1 建立三维数组

array( )函数可用于建立三维数组，笔者直接以实例解说。

实例5_66:建立一个元素为1:24的三维数组，行数是3，列数是4，表格数是2。

```
> first.3array <- array(1:24, dim = c(3, 4, 2))
> first.3array
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

>
```

由上述实例可知，第一个表格填完后再填第二个表，而填表方式与填矩阵方式相同。此外，我们也可以使用dim( )函数建立三维数

组，方法是将一个向量，利用dim( )函数转成三维数组。

实例5_67：用dim( ) 函数重建上一个实例的三维数组的实例。

```
> second.3array <- 1:24
> dim(second.3array) <- c(3, 4, 2)
> second.3array
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24

>
```

5-8-2 identical( )函数

identical( )函数主要是用于比较两个对象是否完全相同。

实例ch5_68：比较first.3array和second.3array对象是否完全相同。

```
> identical(first.3array, second.3array)
[1] TRUE
>
```

5-8-3 取得三维数组的元素

取得三维数组元素的方法与取得向量或矩阵元素的方法相同，也是使用索引，可参考下列实例。

实例ch5_69：取得第2个表格中，第1行，第3列元素。

```
> first.3array[1, 3, 2]
[1] 19
> .
```

实例ch5_70:取得第2个表格中，除去第3行，第1至3列的元素。

```
> first.3array[-3, 1:3, 2]
     [,1] [,2] [,3]
[1,]   13   16   19
[2,]   14   17   20
>
```

由上述结果可以发现，初始first.3array为数组对象，经筛选后，变成矩阵。如果期待筛选完，对象仍是三维数组，那么可加上参数“drop = FALSE”。

实例ch5_71：重新设计ch5_70，保持筛选结果是三维数组。

```
> first.3array[-3, 1:3, 2, drop = FALSE]
, , 1

     [,1] [,2] [,3]
[1,]   13   16   19
[2,]   14   17   20

>
```

实例ch5_72：筛选出每个表格的第3行数据。

```
> first.3array[3, , ]
     [,1] [,2]
[1,]    3   15
[2,]    6   18
[3,]    9   21
[4,]   12   24
>
```

细心的读者应该发现，原先第3行的数据，已经不是筛选后第3行的数据了。这是因为降维度后，第1个表格的数据以列优先方式先填充，第2个表格再依方式填充，所以可以得到上述结果。

实例ch5_73：筛选每个表格第2列的数据。

```
> first.3array[ , 2, ]
     [,1] [,2]
[1,]    4   16
[2,]    5   17
[3,]    6   18
>
```

5-9 再谈class( )函数

在前一章我们介绍使用class( )函数时，如果将向量变量放在此函数内，可列出此向量变量元素的数据类型，如果将矩阵放入此函数内，结果如何呢？

实例ch5_74：class( ) 函数的参数是矩阵变量的应用。

```
> first.matrix <- matrix(1:12, nrow = 4)
> class(first.matrix)
[1] "matrix"
>
```

上述执行结果是矩阵“matrix”。

实例ch5_75：class( ) 函数的参数是数组变量的应用。

```
> first.3array <- array(1:24, dim = c(3, 4, 2))
> class(first.3array)
[1] "array"
>
```

上述命令的执行结果是“array”。同样的方法可以应用于未来几章要介绍的因子(factor)、数据框(data frame)和列表(list)。但是如果class( )函数放入的参数是变量(例如矩阵)的特定元素，则将显示该元素的数据类型。

实例ch5_76：class( ) 函数参数是矩阵特定元素的应用。

```
> first.matrix <- matrix(1:12, nrow = 4)
> class(first.matrix[2, 3])
[1] "integer"
>
```

本章习题

一. 判断题

(　　) 1. 使用rbind() 函数将两个向量做行合并，向量的长度不一定要相等。

(　　) 2. 有如下两个命令。

```
> x <- matrix(1:12, nrow = 4, byrow = TRUE)
> x
```

上述命令执行后，下列的执行结果是正确的。

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

(　　) 3. 有如下命令。

```
> str(x)
 int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
```

由上述执行结果可知，x是一个矩阵。

(　　) 4. 有如下两个命令。

```
> x <- matrix(1:12, nrow = 4)
> is.array(x)
```

上述命令的执行结果如下所示。

```
[1] TRUE
```

(　　) 5. 有如下两个命令。

```
> x <- matrix(1:12, nrow = 3)
> x[-c(2, 3)]
```

上述命令的执行结果如下所示。

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
```

(　　) 6. 使用names( )函数可以更改矩阵的行名和列名。

(　　) 7. 有如下命令。

```
> dimnames(x)
[[1]]
[1] "A" "B" "C"

[[2]]
NULL
```

由上述执行结果可以知道，目前x对象行名分别是“A”“B”“C”，没有列名。

(　　) 8. R语言是允许矩阵和向量相加的，只要矩阵的行数与向量长度相同即可。

(　　) 9. 如下两个命令。

```
> x1 <- matrix(1:9, nrow = 3)
> x2 <- matrix(1:8, nrow = 2)
> x1 %*% x2
```

上述命令的执行结果如下所示。

```
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

(　　) 10. 有如下命令。

```
> diag(1, 3, 3)
```

上述命令执行结果如下所示。

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

(　　) 11. 可使用下列命令，建立一个元素为1:24的三维数组，行数是3，列数是4，表格数是2。

```
> x <- array(1:24, dim = c(3, 4, 2))
```

二. 单选题

(　　) 1. 已知如下3个向量。

a <- c(1, 2, 3)

b <- c(4, 5, 6)

c <- c(7, 8, 9)

想要生成如下矩阵。

1 2 3

4 5 6

7 8 9

可以使用下列哪个命令？

A. cbind(a, b, c)　　B. rbind(a, b, c)

C. matrix(a, b, c)　　D. matrix(c(a, b, c), ncol = 3)

(　　) 2. 以下命令会得到哪个输出结果？

```
> x <- c(1, 3, 5)
> y <- c(3, 2, 10)
> cbind(x, y)
```

A. 长度为3的向量　　B. 一个3x2的矩阵

C. 一个3x3的矩阵　　D. 一个2x3的矩阵

(　　) 3. 以下命令会得到哪个输出结果？

```
> x <- matrix(4:15, nrow = 3 )
> x
```

A.

```
     [,1] [,2] [,3] [,4]
[1,]    4    7   10   13
[2,]    5    8   11   14
[3,]    6    9   12   15
```

B.

```
     [,1] [,2] [,3]
[1,]    4    8   12
[2,]    5    9   13
[3,]    6   10   14
[4,]    7   11   15
```

C.

```
     [,1] [,2] [,3] [,4]
[1,]    4    5    6    7
[2,]    8    9   10   11
[3,]   12   13   14   15
```

D.

```
     [,1] [,2] [,3]
[1,]    4    5    6
[2,]    7    8    9
[3,]   10   11   12
[4,]   13   14   15
```

(　　) 4. 以下命令会得到下列哪个结果？

```
> x <- matrix(1:12, nrow = 3)
> x[2, 3]
```

A. [1] 6　　B. [1] 5　　C. [1] 8　　D. [1] 9

(　　) 5. 以下命令会得到哪个输出结果？

```
> x <- matrix(1:12, nrow = 3)
> ncol(x)
```

A. [1] 3　　B. [1] 4　　C. [1] 5　　D. [1] 6

(　　) 6. 以下命令会得到哪个结论？

```
> dim(x)
[1] 3 4
```

A. x对象行数是3　　B. x对象行数是4

C. x对象列数是3　　D. x对象行数是7

(　　) 7. 以下命令会得到哪个输出结果？

```
> dim(x)
[1] 3 4
> length(x)
```

A. [1] 3　　B. [1] 4　　C. [1] 7　　D. [1] 12

(　　)8. 以下命令会得到哪个输出结果？

```
> cbind(4:6, 11:13, matrix(1:6, nrow = 3))
```

A.
```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

B.
```
     [,1] [,2] [,3] [,4]
[1,]    4    7   10   13
[2,]    5    8   11   14
[3,]    6    9   12   15
```

C.
```
     [,1] [,2] [,3] [,4]
[1,]    4   11    1    4
[2,]    5   12    2    5
[3,]    6   13    3    6
```

D.
```
     [,1] [,2] [,3] [,4]
[1,]    2    5    8   11
[2,]    3    6    9   12
[3,]    4    7   10   13
```

(　　) 9. 以下命令会得到哪个输出结果？

```
> x <- matrix(10:21, nrow = 3)
> x[2, ]
```

A. [1] 11 14 17 20　　B. [1] 10 13 16 19

C. [1] 10 11 12　　D. [1] 13 14 15

(　　) 10. 以下命令会得到哪个输出结果？

```
> x <- matrix(1:20, nrow = 4)
> x[3:4, 4:5]
```

A.
```
     [,1] [,2]
[1,]    9   13
[2,]   10   14
```

B.
```
     [,1] [,2]
[1,]   15   19
[2,]   16   20
```

C.
```
     [,1] [,2]
[1,]    3    7
[2,]    4    8
```

D.
```
     [,1] [,2]
[1,]    6   10
[2,]    7   11
```

(　　) 11. 以下命令会得到哪个输出结果？

```
> x <- matrix(1:20, nrow = 4)
> x[-c(3:4), -2]
```

A.
```
     [,1] [,2] [,3] [,4]
[1,]    1    9   13   17
[2,]    2   10   14   18
```

B.
```
     [,1] [,2] [,3] [,4]
[1,]    5    9   13   17
[2,]    6   10   14   18
```

C.
```
     [,1] [,2] [,3] [,4]
[1,]    2   10   14   18
[2,]    3   11   15   19
[3,]    4   12   16   20
```

D.
```
     [,1] [,2] [,3]
[1,]    1    5   17
[2,]    3    7   19
[3,]    4    8   20
```

(　　) 12. 以下命令会得到哪个输出结果？

```
> x <- matrix(1:20, nrow = 4)
> rowSums(x)
```

A. [1] 2.5 6.5 10.5 14.5 18.5　　B. [1] 10 26 42 58 74

C. [1] 45 50 55 60　　D. [1] 9 10 11 12

(　　) 13. 以下命令会得到哪个输出结果？

```
> x <- array(1:24, dim = c(3, 4, 2))
> x[1, 2, 2]
```

A. [1] 13　　B. [1] 14　　C. [1] 15　　D. [1] 16

( ) 14. 以下命令会得到哪个输出结果？

```
> x <- array(1:24, dim = c(3, 4, 2))
> class(x[1, 2, 2])
> class(x)
```

A. `[1] "integer"`　　B. `[1] "array"`

C. `[1] "character"`　　D. `[1] "matrix"`

三. 多选题

( ) 1. 以下哪些class命令的执行结果为“matrix”？(选择3项)

A.
```
> class(cbind(c(1, 2), c(2, 4)))
```

B.
```
> class(c(1, 2))
```

C.
```
> a <- 1:6
> dim(a) <- c(2, 3)
> class(a)
```

D.
```
> a <- matrix(0,1,2)
> class(a)
```

E.
```
> class(1+2*3/4-5)
```

( ) 2. 有一个命令如下所示。

```
> x <- matrix(1:12, nrow = 3)
```

以下哪些命令可将矩阵的行名分别设为“R1”“R2”和“R3”？(选择两项)

A.
```
> rownames(x) <- c("R1", "R2", "R3")
```

B.
```
> colnames(x) <- c("R1", "R2", "R3")
```

C.
```
> rownames(x) <- ("R1", "R2", "R3")
```

D.
```
> dimnames(x)[[1]] <- c("R1", "R2", "R3")
```

E.
```
> dimnames(x)[[2]] <- c("R1", "R2", "R3")
```

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. 建立以下元素内容为1:30的矩阵。

(1) 5行6列的矩阵，排列使用默认值。

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    6   11   16   21   26
[2,]    2    7   12   17   22   27
[3,]    3    8   13   18   23   28
[4,]    4    9   14   19   24   29
[5,]    5   10   15   20   25   30
```

(2) 5行6列的矩阵，排列使用byrow = TRUE。

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    7    8    9   10   11   12
[3,]   13   14   15   16   17   18
[4,]   19   20   21   22   23   24
[5,]   25   26   27   28   29   30
```

(3) 使用str( )函数列出上述矩阵。

```
int [1:5, 1:6] 1 2 3 4 5 6 7 8 9 10 ...

int [1:5, 1:6] 1 7 13 19 25 2 8 14 20 26 ...
```

2. 有3个向量。

x1 <- c(10, 12, 14)

x2 <- c(7, 14, 5)

x3 <- c(15, 3, 19)

(1) 使用rbind( )函数将上述向量组成矩阵A1。

```
> A1
   [,1] [,2] [,3]
x1   10   12   14
x2    7   14    5
x3   15    3   19
```

(2) 使用cbind( )函数将向量组成矩阵A2。

```
> A2
     x1 x2 x3
[1,] 10  7 15
[2,] 12 14  3
[3,] 14  5 19
```

(3) 列出A1矩阵[1:2,]对应的元素。

```
   [,1] [,2] [,3]
x1   10   12   14
x2    7   14    5
```

(4) 列出A1矩阵[1:2, 2:3] 对应的元素。

```
   [,1] [,2]
x1   12   14
x2   14    5
```

(5) 列出A2矩阵[, 2:3] 对应的元素。

```
     x2 x3
[1,]  7 15
[2,] 14  3
[3,]  5 19
```

(6) 列出A2矩阵[2:2, 2:3] 对应的元素。

```
x2 x3
14  3
```

(7) 取得A1矩阵第1行以外的矩阵元素。

```
   [,1] [,2] [,3]
x2    7   14    5
x3   15    3   19
```

(8) 取得A2矩阵第2列以外的矩阵元素。

```
     x1 x3
[1,] 10 15
[2,] 12  3
[3,] 14 19
```

3. NBA 球星5人得分向量数据如下：

Lin. 7, 8, 6, 11, 9, 12

Jordon. 12, 8, 9, 15, 7, 12

Curry. 13, 9, 6, 11, 9, 13

Antony. 12, 11, 9, 13, 8, 14

Kevin. 7, 10, 8, 6, 5, 9

请转成矩阵。

```
        [,1] [,2] [,3] [,4] [,5] [,6]
Lin        7    8    6   11    9   12
Jordon    12    8    9   15    7   12
Curry     13    9    6   11    9   13
Antony    12   11    9   13    8   14
Kevin      7   10    8    6    5    9
```

4. 为上一题的NBA球星数据矩阵设置行名(使用球星名字)和列名(使用场次编号)。

```
     1st 2nd 3rd 4th 5th 6th
Lin    7   8   6  11   9  12
Jor   12   8   9  15   7  12
Cur   13   9   6  11   9  13
Ant   12  11   9  13   8  14
Kev    7  10   8   6   5   9
```

5. 使用rowSums( )函数为上述球星计算总得分。

```
Lin Jor Cur Ant Kev
 53  63  61  67  45
```

6. 使用rowMeans( )函数为上述球星计算平均得分。

```
     Lin       Jor       Cur       Ant       Kev
8.833333 10.500000 10.166667 11.166667  7.500000
```

7. 收集2个班级，5位同学的数学和R语言成绩，学生数据用ID表示，然后将数据建立为三维数组。

```
, , class-A

      R-score math
ID-01      71   76
ID-02      72   77
ID-03      73   78
ID-04      74   79
ID-05      75   80

, , class-B

      R-score math
ID-01      81   86
ID-02      82   87
ID-03      83   88
ID-04      84   89
ID-05      85   90
```