1.2 数据结构 i：向量、矩阵、多维数组

数据结构是为了便于存储不同类型的数据而设计的**数据容器**。学习数据结构，就是要把各个数据容器的特点、适合存取什么样的数据理解透彻，只有这样才能在实际中选择最佳的数据容器，数据容器选择的合适与否，直接关系到代码是否高效简洁，甚至能否解决问题。

R 中常用的数据结构可划分为：

- 同质数据类型（homogeneous），即所存储的一定是相同类型的元素，包括向量、矩阵、多维数组；
- 异质数据类型（heterogeneous），即可以存储不同类型的元素，这大大提高了存储的灵活性，但同时也降低了存储效率和运行效率，包括列表、数据框。

另外，还有字符串、日期时间数据、时间序列数据、空间地理数据等。

R 中的数据结构还有一种从**广义向量**[3]的角度的划分：

- **原子向量**：各个值都是同类型的，包括 6 种类型：`logical`、`integer`、`double`、`character`、`complex`、`raw`，其中 `integer` 和 `double` 也统称为 `numeric`；
- **列表**：各个值可以是不同类型的，NULL 表示空向量（长度为 0 的向量）

向量都有两个属性：`type`（类型）、`length`（长度）；还能以属性的方式向向量中任意添加额外的 `metadata`（元数据），属性可用来创建扩展向量，以执行一些新的操作。常用的扩展向量有：

- 基于整数型向量构建的因子
- 基于数值型向量构建的日期和日期时间
- 基于数值型向量构建的时间序列
- 基于列表构建的数据框和 tibble

> 列表是广义向量，从这个角度有助于理解 `purrr::map_*()` 系列的泛函式编程。

1.2.1 向量（一维数据）

向量是由一组相同类型的原始值构成的序列，可以是一组数值、一组逻辑值、一组字符串等。

常用的向量有：数值向量、逻辑向量、字符向量。

数值向量

数值向量就是由数值组成的向量，单个数值是长度为 1 的数值向量

[3]由一系列可以根据位置索引的元素构成，元素可以很复杂和不同类型。

```
x = 1.5
x
```

```
## [1] 1.5
```

可以用 numeric() 来创建全为 0 的指定长度的数值向量：

```
numeric(10)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

R 中经常用函数 c() 实现将多个对象合并到一起：

```
c(1, 2, 3, 4, 5)
```

```
## [1] 1 2 3 4 5
```

```
c(1, 2, c(3, 4, 5))    # 将多个数值向量合并成一个数值向量
```

```
## [1] 1 2 3 4 5
```

创建等差的数值向量，用 : 或者函数 seq()，基本格式为：

```
seq(from, to, by, length.out, along.with, ...)
```

其中，

from：设置首项（默认为 1）；

to：设置尾项；

by：设置等差值（默认为 1 或 -1）；

length.out：设置序列长度；

along.with：以该参数的长度作为序列长度。

```
1:5                    # 同 seq(5) 或 seq(1,5)
```

```
## [1] 1 2 3 4 5
```

```
seq(1, 10, 2)          # 从 1 开始，到 10 结束，步长为 2
```

```
## [1] 1 3 5 7 9
```

```
seq(3, length.out=10)
```

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

© 张敬信

创建重复的数值向量用函数 rep()，基本格式为：

```
rep(x, times,length.out, each, ...)
```

其中，

x：为要重复的序列；

times：设置序列重复次数；

length.out：设置产生的序列的长度；

each：设置每个元素分别重复的次数（默认为 1）。

```
x = 1:3
rep(x, 2)
```

```
## [1] 1 2 3 1 2 3
```

```
rep(x, each = 2)
```

```
## [1] 1 1 2 2 3 3
```

```
rep(x, c(2, 1, 2))              # 按照规则重复序列中的各元素
```

```
## [1] 1 1 2 3 3
```

```
rep(x, each = 2, length.out = 4)
```

```
## [1] 1 1 2 2
```

```
rep(x, each = 2, times = 3)
```

```
##  [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
```

注意，R 中两个不同长度的向量做运算，短的会自动循环补齐以配合长的。

```
2:3 + 1:5
```

```
## [1] 3 5 5 7 7
```

逻辑向量

逻辑向量，是一组逻辑值（TRUE 或 FALSE，或简写为 T 或 F）的向量。

```
c(1, 2) > c(2, 1)          # 等价于 c(1 > 2, 2 > 1)
```

```
## [1] FALSE  TRUE
```

```
c(2, 3) > c(1, 2, -1, 3)  # 等价于 c(2 > 1, 3 > 2, 2 > -1, 3 > 3)
```

```
## [1]  TRUE  TRUE  TRUE FALSE
```

© 张敬信

除了比较运算符外，还可以用 %in% 判断元素是否属于集合：

```
c(1, 4) %in% c(1, 2, 3)   # 左边向量每一个元素是否属于右边集合
```

```
## [1]  TRUE FALSE
```

> match(v1, v2) 逐个检查向量 v1 中元素是否在向量 v2 中，若是则返回该元素，否则返回 NA。

字符向量

字符（串）向量，是一组字符串组成的向量，R 中单引号和双引号都可以用来生成字符向量。

```
"hello, world!"
```

```
## [1] "hello, world!"
```

```
c("Hello", "World")
```

```
## [1] "Hello" "World"
```

```
c("Hello", "World") == "Hello, World"
```

```
## [1] FALSE FALSE
```

要想字符串中出现单引号或双引号，需要用转义符 \ 来做转义，或者单双引号错开，用函数 cat() 生成字符串：

```
cat("Is \"You\" a Chinese name?")
```

```
# Is "You" a Chinese name?
```

```
cat('Is "You" a Chinese name?')
```

```
# Is "You" a Chinese name?
```

```
'Is "You" a Chinese name?'
```

```
# [1] "Is \"You\" a Chinese name?"
```

> R 中还有不常用的复数向量、原（raw）向量。

访问向量子集

即访问向量的一些特定元素或者某个子集。注意，R 中的索引是从 1 开始的。

使用元素的位置来访问：

```
v1 = c(1, 2, 3, 4)
v1[2]                # 第 2 个元素
v1[2:4]              # 第 2-4 个元素
v1[-3]               # 除了第 3 个之外的元素
```

也可以放任意位置的数值向量，但是注意不能既放正数又放负数：

```
v1[c(1,3)]
v1[c(1, 2, -3)]     # 报错
```

访问不存在的位置也是可以的，返回 NA：

```
v1[3:6]
```

使用逻辑向量来访问，输入与向量相同长度的逻辑向量，以此决定每一个元素是否要被获取：

```
v1[c(TRUE, FALSE, TRUE, FALSE)]
```

这可以引申为“根据条件访问向量子集”：

```
v1[v1 <= 2]        # 同 v1[which(v1 <= 2)] 或 subset(v1, v1<=2)
v1[v1 ^ 2 - v1 >= 2]
which.max(v1)      # 返回向量 v1 中最大值所在的位置
which.min(v1)      # 返回向量 v1 中最小值所在的位置
```

对向量子集赋值，替换相应元素

对向量子集赋值，就是先访问到向量子集，再赋值。

```
v1[2] = 0
v1[2:4] = c(0, 1, 3)
v1[c(TRUE, FALSE, TRUE, FALSE)] = c(3, 2)
v1[v1 <= 2] <- 0
```

注意，若对不存在的位置赋值，前面将用 NA 补齐：

```
v1[10] <- 8
v1
```

对向量元素命名

可以在创建向量的同时对其每个元素命名：

```
x <- c(a = 1, b = 2, c = 3)
x
```

```
## a b c
## 1 2 3
```

命名后，就可以通过名字来访问向量元素：

```
x[c("a", "c")]
x[c("a", "a", "c")]    # 重复访问也是可以的
x["d"]                 # 访问不存在的名字
```

获取向量元素的名字：

```
names(x)
```

```
## [1] "a" "b" "c"
```

更改向量元素的名字：

```
names(x) <- c("x", "y", "z")
x["z"]
```

```
## z
## 3
```

移除向量元素的名字：

```
names(x) <- NULL
x
```

```
## [1] 1 2 3
```

© 张敬信

[ ] 与 [[ ]] 的区别

[ ] 可以提取对象的子集，[[ ]] 可以提取对象中的元素。

二者的区别：以向量为例，可以将一个向量比作 10 盒糖果，你可以使用 [ ] 获取其中的 3 盒糖果，使用 [[ ]] 打开盒子并从中取出一颗糖果。

对于未对元素命名的向量，使用 [ ] 和 [[ ]] 取出一个元素会产生相同的结果。但已对元素命名的向量，二者会产生不同的结果：

```
x <- c(a = 1, b = 2, c = 3)
x["a"]          # 取出标签为"a" 的糖果盒
```

```
## a
## 1
```

```
x[["a"]]        # 取出标签为"a" 的糖果盒里的糖果
```

```
## [1] 1
```

由于 [[ ]] 只能用于提取出一个元素，因此不适用于提取多个元素的情况，所以 [[ ]] 不能用于负整数，因为负整数意味着提取除特定位置之外的所有元素。使用含有不存在的位置或名称来创建向量子集时将会产生缺失值。但当使用 [[ ]] 提取一个位置超出范围或者对应名称不存在的元素时，该命令将会无法运行并产生错误信息。

以下三个语句会报错：

```
x[[c(1, 2)]]
x[[-1]]
x[["d"]]
```

对向量排序

向量排序函数 sort()，基本格式为：

```
sort(x, decreasing, na.last, ...)
```

其中，

x：为排序对象（数值型或字符型）；

decreasing：默认为 FALSE 即升序，TURE 为降序；

na.last：默认为 FALSE，若为 TRUE，则将向量中的 NA 值放到序列末尾。

函数 order()，返回元素排好序的索引，以其结果作为索引访问元素，正好是排好序的元素。

函数 rank()，返回值是该向量中对应元素的 ‘‘排名’’。

```
x = c(1,5,8,2,9,7,4)
sort(x)
```

© 张敬信

```
## [1] 1 2 4 5 7 8 9
```

```
order(x)      # 默认升序，排名第 2 的元素在原向量的第 4 个位置
```

```
## [1] 1 4 7 2 6 3 5
```

```
x[order(x)]  # 同 sort(x)
```

```
## [1] 1 2 4 5 7 8 9
```

```
rank(x)       # 默认升序，第 2 个元素排名第 4 位
```

```
## [1] 1 4 6 2 7 5 3
```

还有函数 `rev()`，可将序列进行反转，即 `1,2,3` 变成 `3,2,1`。

1.2.2 矩阵（二维数据）

矩阵是一个用两个维度表示和访问的向量。因此，适用于向量的性质和方法大多也适用于矩阵：矩阵也要求元素是同一类型，数值矩阵、逻辑矩阵等。

创建矩阵

函数 `matrix()` 将一个向量创建为矩阵，其基本格式为：

```
matrix(x, nrow, ncol, byrow, dimnames, ...)
```

其中，

x：为数据向量作为矩阵的元素；

nrow：设定行数；

ncol：设定列数；

byrow：设置是否按行填充，默认为 FALSE（按列填充）；

dimnames：用字符型向量表示矩阵的行名和列名。

```
matrix(c(1, 2, 3,
         4, 5, 6,
         7, 8, 9), nrow = 3, byrow = FALSE)
```

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

```
matrix(c(1, 2, 3,
         4, 5, 6,
```

```
        7, 8, 9), nrow = 3, byrow = TRUE)
```

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

对矩阵的行列命名：

```
matrix(1:9, nrow = 3, byrow = TRUE,
       dimnames = list(c("r1","r2","r3"), c("c1","c2","c3")))
```

```
##    c1 c2 c3
## r1  1  2  3
## r2  4  5  6
## r3  7  8  9
```

也可以创建后再命名：

```
m1 = matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), ncol = 3)
rownames(m1) = c("r1", "r2", "r3")
colnames(m1) = c("c1", "c2", "c3")
m1
```

```
##    c1 c2 c3
## r1  1  4  7
## r2  2  5  8
## r3  3  6  9
```

特殊矩阵：

```
diag(1:4, nrow = 4)      # 对角矩阵
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    2    0    0
## [3,]    0    0    3    0
## [4,]    0    0    0    4
```

函数 `as.vector()`，可将矩阵转化为向量，元素按列读取。

访问矩阵子集

矩阵是用两个维度表示和访问的向量，可以用一个二维存取器 `[ , ]` 来访问，这类似于构建向量子集时用的一维存取器 `[]`。

© 张敬信

可以为每个维度提供一个向量来确定一个矩阵的子集。方括号中的第 1 个参数是行选择器，第 2 个参数是列选择器。与构建向量子集一样，可以在两个维度中使用数值向量、逻辑向量和字符向量。

```
m1[1,2]              # 提取第 1 行，第 2 列的单个元素
m1[1:2, 2:4]         # 提取第 1 至 2 行，第 2 至 4 列的元素
m1[c("r1","r3"), c("c1","c3")]  # 提取行名为 r1 和 r3，列名为 c1 和 c3 的元素
```

若一个维度空缺，则选出该维度的所有元素：

```
m1[1,]               # 提取第 1 行，所有列元素
m1[,2:4]             # 提取所有行，第 2 至 4 列的元素
```

负数表示在构建矩阵子集时可排除该位置，这和向量中的用法一致：

```
m1[-1,]              # 提取除了第 1 行之外的所有元素
m1[,-c(2,4)]         # 提取除了第 2 和 4 列之外的所有元素
```

注意，矩阵是一个用两个维度表示和访问的向量，但它本质上仍然是一个向量。因此，向量的一维存取器也可以用来构建矩阵子集：

```
m1[3:7]
```

```
## [1] 3 4 5 6 7
```

由于向量只包含相同类型的元素，矩阵也是如此。所以它们的操作方式也相似。若输入一个不等式，则返回同样大小的逻辑矩阵：

```
m1 > 3
```

```
##       c1   c2   c3
## r1 FALSE TRUE TRUE
## r2 FALSE TRUE TRUE
## r3 FALSE TRUE TRUE
```

根据它就可以选择矩阵元素或赋值：

```
m1[m1 > 3]   # 注意选出来的结果是向量
```

```
## [1] 4 5 6 7 8 9
```

矩阵运算

- `A+B`，`A-B`，`A*B`，`A/B`：矩阵四则运算，要求矩阵同型，类似 Matlab 中的点运算，分别将对应位置的元素做四则运行；
- `A %*% B`：矩阵乘法，要求 A 的列数 = B 的行数。

© 张敬信

1.2.3 多维数组（多维数据）

向量/矩阵向更高维度的自然推广。具体来说，多维数组就是一个维度更高（通常大于 2）、可访问的向量。数组也要求元素是同一类型。

创建多维数组

函数 `array()` 将一个向量创建为多维数组，基本格式为：

```
array(x, dim, dimnames, ...)
```

其中，
x：为数据向量作为多维数组的元素；
dim：设置多维数组各维度的维数；
dimnames：设置多维数组各维度的名称。

```
a1 = array(1:24, dim = c(3, 4, 2))
a1
```

```
## , , 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
```

也可以在创建数组时对每个维度进行命名：

```
a1 = array(1:24, dim = c(3, 4, 2),
           dimnames=list(c("r1","r2","r3"),
                         c("c1","c2","c3","c4"), c("k1","k2")))
```

或者创建之后再命名[4]

[4] `list` 是创建列表（见下节）。

```
a1 = array(1:24, dim = c(3, 4, 2))
dimnames(a1) = list(c("r1","r2","r3"),
                    c("c1","c2","c3","c4"), c("k1","k2"))
```

构建多维数组子集

第 3 个维度姑且称为“页”

```
a1[2,4,2]           # 提取第 2 行，第 4 列，第 2 页的元素
a1["r2","c4","k2"]  # 提取第 r2 行，第 c4 列，第 k2 页的元素
a1[1,2:4,1:2]       # 提取第 1 行，第 2 至 4 列，第 1 至 2 页的元素
a1[,,2]             # 提取第 2 页的所有元素
dim(a1)             # 返回多维数组 a1 的各维度的维数
```

在想象多维数组时，为了便于形象地理解，可以将其维度依次想象为与“书”相关的概念：行、列、页、本、层、架、室……

本节部分内容参阅 (任坤2017) 和 (Hadley Wickham 2017)。

© 张敬信