R 语言编程概述：3. 编写 R 函数

张金龙

2018 年 6 月 23 日

纯文本文件的编码

- ASCII：纯英文字母与数字等
- GB18030：中文
- GB2312：中文（已不推荐）
- GBK：中文（已不推荐）
- UTF-8：通用语言，可显示绝大部分语言的字符

R 函数和 R 帮助文档，只能用纯 ASCII 字符编写，用 ASCII 或者 UTF-8 编码。不能出现其他语言的字符。

函数

函数是对一些程序语句的封装。一个函数往往完成一项特定的功能，例如，求标准差 sd、求平均值、绘图等。

编写函数，可以减少重复书写代码，R 脚本程序更为清晰、简洁。

函数的结构：

- 函数头
  - 函数名
  - 指向函数名的 `<- function()` 标识
  - 小括号内的参数
- 函数体：花括号以内的部分

```
myfunction <- function(arg1, arg2, ... ){
    statements
    return(object)
}
```

函数名

- 除了用在 apply 或 replicate 中的函数之外，大部分函数都应该有适当的名称。
- 因为函数也是 R 的对象，所以要遵循 R 对象命名的规则，**最好用英文字母开头，不能用数字开头。**
- 函数名中不要有非 ASCII 码的字符。

参数和参数命名

定义函数时，参数为形式参数，并没有真正运行。但是：

- 参数本身是有顺序的，各参数之间用逗号间隔。
- 参数本身是有名称的，名称要符合 R 对象的命名规则。
- 参数本身是可以传递的，如...。
- 参数可以有默认值

给参数起名时，应将其看做普通的 R 对象，参数名称最好能表明其意义。运行函数时参数都将作为普通 R 对象处理。

参数默认值

在定义函数时，指定的参数的值称为默认值，如:

```
pd <- function (samp, tree, include.root = TRUE)
{
    if (is.null(tree$edge.length)) {
        stop("Tree has no branch lengths, cannot compute pd")
    }
    if (include.root) {
        if (!is.rooted(tree)) {
            stop("Rooted tree required to calculate PD
                  with include.root=TRUE argument")
        }
        tree <- node.age(tree)
    }
    ...
}
```

参数传递

由于变量的作用域不同。每一个函数的参数，理论上都应该只用在自己的函数体中。

但有时在函数体中调用的函数参数过多，例如 points.default 函数:

```
points.default <- function (x, y = NULL, type = "p", ...) {
    plot.xy(xy.coords(x, y), type = type, ...)
}
```

其中`...` 称为参数传递。在声明中使用`...`，用户可以直接在调用 `points.default` 函数时，使用 `plot.xy` 的所有参数。

使用其他程序包中的函数

在 R 脚本中，用 `library()` 导入需要的程序包，再直接调用该程序包的函数即可。

但是在 R 函数中，使用其他程序包的函数，最好指明函数或者数据的来源。

例如，某函数中要用到 vegan 程序包的 `vegdist` 函数，那么在函数中，最好用:: 指明函数是在 vegan 程序包中，格式为：`vegan::vegdist()`

程序包中是否能使用其他程序包中的函数，还涉及到程序包的命名空间 NAMESPACE 等。

返回值 return

函数都是为了完成一定的功能，如计算、绘图、生成文件等。

以用于计算的函数为例，运行函数时，用户输入数据后，函数需要提供相应结果，这个结果就是返回值。

- 大部分函数都需要有返回值。
- 返回值只能是一个对象，不能是多个对象。这里的对象类型包括 vector，matrix，dataframe，list 等。
- 如果要返回的结果包含多个对象，则需要放入一个 list 中。

异常及其处理

用户提供的参数不一定能满足函数运行的要求，例如用户提供了错误的数据类型，或者提供的参数的范围不符合要求。

例如除数为一个字符串类型，此时需要报错：

```
> 12/"a"
Error in 12/"a" : non-numeric argument to binary operator
```

不同严重程度的异常

- 如果输入的参数经过类型转换等，函数能够正常运行，此时用 `warning()` 提示用户数据格式不服，但是经过了转换即可。
- 如果函数内部运行时遇到的错误十分严重，函数不能再运行，就要用 `stop()` 终止运行，并告诉用户哪里错了，应该输入什么类型的数据等。
- 若只希望用户看到一些信息，用 `message()` 即可。

```
stop()
warning()
message()
```

编写函数时，应该考虑各种可能的异常情况。

编写高效的函数

1. 要向量化
2. 尽量避免使用循环，如果一定要用到循环，循环中不宜多次读写特别大的数据
3. 尽量将数据的访问和读写等调整到循环之外

```
a <- runif(10)
b <- 20
a * b
```

```
##  [1] 16.515469 14.295674  3.977144  9.618342 14.168138  6.6
##  [8]  7.147923 12.130750 19.038994
```

面向对象编程：类 Class

以一定形式组织起来的数据构成了类。如冰淇淋，有甜筒、雪糕等；水饺，有三鲜、鸡蛋韭菜、玉米等多种，但是统称为水饺。

同理，数据结构固定的 R 对象，也可以归为一类 Class。

例如，用 ape::read.tree 读取的进化树，类型为 phylo

```
library(ape)
data(bird.orders)
class(bird.orders)
```

```
## [1] "phylo"
```

类 Class 以 APE 的 phylo 为例

ape 程序包中，读取的进化树均为 phylo 类型。每个 phylo 对象的本质都是一个 list，包括如下基本组件：

- `edge`：拓扑关系
- `Nnode`：节点数
- `tip.label`：物种名
- `edge.length`：枝长

APE 的 phylo 类

```
str(bird.orders)
```

```
## List of 4
##  $ edge       : int [1:44, 1:2] 24 25 26 26 25 27 28 28 27
##  $ Nnode      : int 22
##  $ tip.label  : chr [1:23] "Struthioniformes" "Tinamiformes
##  $ edge.length: num [1:44] 2.1 4.1 21.8 21.8 3 1.3 21.6 21.
##  - attr(*, "class")= chr "phylo"
```

```
names(bird.orders)
```

```
## [1] "edge"        "Nnode"       "tip.label"   "edge.length"
```

泛函数和子函数

- 遇到类型为 phylo 的数据结构，如果用户调用了 plot，R 就会主动调用 plot.phylo 函数完成绘图。
- 这里 plot 称为泛型函数
- plot.phylo 称为子函数。

查看 plot 有多少种子函数

```
methods("plot")[1:10] # 只显示前十个
```

```
##  [1] "plot.acf"              "plot.ACF"              "plot.au
##  [4] "plot.compareFits"      "plot.correlogram"      "plot.co
##  [7] "plot.data.frame"       "plot.decomposed.ts"    "plot.de
## [10] "plot.dendrogram"
```

R 会根据数据的类型 class，用不同的子函数处理。这就是为什么有些数据直接绘制箱线图，有些直接绘制散点图。

S3 面向对象编程：设定 class

通过以上方式实现面向对象编程的方法，称为 S3 方法，S3 在 R 中应用最为广泛。

```
# 设定对象的类
ttt <- round(runif(5), 3)
attr(ttt,"class") <- "relinsuo" # 方法 1
class(ttt) <- "relinsuo"        # 方法 2
class(ttt)                      # 查看类
```

```
## [1] "relinsuo"
```

```
inherits(ttt,"relinsuo")        # 判断是否属于某类
```

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

S3 面向对象编程：定义泛型函数和子函数

```
# 设定泛型函数
peixun <- function(x, ...) UseMethod("peixun")

# 定义子函数
peixun.relinsuo <- function(x) {
  print(paste("Random numbers:", x))
}
peixun(ttt)
```

```
## [1] "Random numbers: 0.873" "Random numbers: 0.707" "Random
## [4] "Random numbers: 0.518" "Random numbers: 0.019"
```

S3 面向对象编程：查看泛型函数有哪些子函数

```
methods(peixun)
```

```
## [1] peixun.relinsuo
## see '?methods' for accessing help and source code
```

如果泛型函数已经定义了，如 plot，则直接定义子函数即可。

S4 面向对象编程：创建类

S3 面向对象编程简单直接，但是很容易引入错误。

为了更好地实现面向对象编程，Chambers 等人引入了 S4。

```
# 设置 forestplot 类
setClass("forestplot",
         slots = list(name = "character",
                      starting_year = "numeric",
                      area="numeric"))
```

S4 面向对象编程：继承

当某一种新的对象，与已经定义好的数据有类似的数据结构，即可用继承的方式，生成新的类。继承保证了前一种对象对应的所有函数在新的类中都可以使用。

```
setClass("subtropical_forestplot",
         slots = list(
           name="character",
           starting_year = "numeric",
           area="numeric",
           longitude="numeric",
           latitude = "numeric"),
          contains="forestplot")
```

S4 面向对象编程：创建泛型函数

```
# 实例化，创建一个 S4 对象，名叫 gutianshan
gutianshan <- new("subtropical_forestplot",
                  name="gutianshan",
                  starting_year = 2006,
                  area = 24,
                  longitude = 120,
                  latitude = 30)

# 创建泛型函数
setGeneric("age",function(object) standardGeneric("age"))
```

```
## [1] "age"
```

S4 面向对象编程：创建子函数

为什么要创建子函数?

```
# 创建子函数
setMethod("age", signature(object = "subtropical_forestplot"),
          function(object){
            cat(paste(object@name, "plot is",
                      lubridate::year(Sys.Date()) -
                        object@starting_year),
                "years old.\n")
          }
          )
```

S4 面向对象编程：S4 调用

```
# 通过调用泛型函数，调用子函数
age(gutianshan)
```

```
## gutianshan plot is 12 years old.
```

```
# 查看某泛型函数下对应的方法
showMethods(age)
```

```
## Function: age (package .GlobalEnv)
## object="subtropical_forestplot"
```

```
# 访问 S4 对象的属性，需要用 @
gutianshan@name
```

```
## [1] "gutianshan"
```

```
gutianshan@starting_year
```

```
## [1] 2006
```

编程风格 Programming Style

目的：便于阅读，结构清晰，减少注释

- R 脚本在开头处注明脚本的目的、作者、日期
- 所需要的程序包总在最开始处集中导入，避免在脚本中部导入程序包
- 变量命名：主要用名词，要能表述变量的含义，减少用 a、b、x 等命名变量
- 函数命名：函数用动词
- 缩进：按照一定的层次缩进
- 括号、引号的配对：书写时，输入对应括号，再在括号里面编写
- 运算符号前后要有空格
- 除非是 S3 类，在对象名中少用"."
- 变量命名等前后风格要一致
- 适当注释，宜注释为什么，而不是做什么

参考：https://nanx.me/rstyle/

练习与答疑