R 语言编程

战立侃

2019 年 4 月 24 日

2.

目录

第一章　R 语言简介

1.1　什么是 R 语言

R 是一个用于统计计算和统计作图的系统 (R Core Team, 2018)。R 系统既是一种运行环境，又是一门编程语言。

作为一种运行环境，R 系统不是不同功能的简单累加，而是一个由多种软件功能组成的有机整体。R 系统具有与其他编程语言的接口，也具有高级图形、读入某些系统函数、运行脚本程序等、和调试等功能。R 系统包含了大量统计程式，如线性、广义线性和非线性模型、时间序列分析、经典参数和非参数检验、聚类和平滑分析等。R 系统中也包含了大量灵活的作图环境，以对数据进行有效展示。

作为一门编程语言，R 语言的前身是 S 语言。S 语言由贝尔实验室的 John Chambers 和同事们于 1977 年创立。John Chambers 还因此获得了 1998 年计算机协会的软件系统奖，因为 “S 系统永久性的改变了人们分析、可视化和操作数据的方式”。到目前为止，S 语言有三种引擎或三种实现方式：旧 S 引擎（S_3）、新 S 引擎（S_4）和 R。与 S 引擎不同，R 语言的底层实现及其语义系统都来自于 Scheme 语言。因为 R 和 S 语言的差异仅存在于其实现方式或引擎上，所以从表面看起来 R 语言与 S 语言非常相似，大部分用 S 语言编写的代码在 R 语言下都可以运行。R 语言的内核是一种解释性语言。R 语言允许分支、循环、以及借助函数进行模块化编程。R 中大部分用户可见函数均由 R 编写。出于运算效率性的考虑，用户也可以通过接口调用 C、C++、或 Fortran 语言编写的程序。

R 语言最初由新西兰奥克兰大学的 Ross Ihaka 和 Robert Gentleman 撰写，R 语言名字的由来也是因为这两个作者名字的首字母都是 R。R 是一款免费的开源软件，其源代码受自由软件基金会的 GNU 通用公共授权条款保护。R 可以在大部分操作系统上编译和运行，如 Unix、Linux、Windows、和 MAC OS。读者可以访问 R 语言官方网站了解更多信息：

- R 官方网站：https://www.r-project.org

1.1.1 S 语言和 R 语言的关系

1.2 软件的安装

1.2.1 R 的安装

R 软件的最新版代码存放在 R 综合典藏网 (CRAN)。R 综合典藏网是由遍布世界的一系列 ftp 和服务器组成的网络结构，用以存放完全相同的、最新版的 R 代码、软件包、以及相关文档。为了提高下载速度，建议读者选择就近的典藏网镜像点下载 R 代码和相关软件包。网站提供可在 liunix、mac OSX 和 windows 系统下安装的 R 语言的可执行版安装包。作为一种开放源代码软件，R 综合典藏网还提供了没有编译过的 R 软件的源代码版，供软件高级使用者下载。下面是 Windows 系统和 Mac 系统下 R 可执行安装包的下载链接：

- Windows 系统：https://cran.r-project.org/bin/windows/base/
- Mac 系统：https://cran.r-project.org/bin/macosx/

软件安装完成后，我们可以用下面的命令查看本机安装的 R 软件版本，例如作者电脑安装的 R 软件版本是：`R version 3.5.3 (2019-03-11)`。

```
R.version.string # 如下函数也有类似功能：version、
# R.version、R.Version()、getRversion()

## [1] "R version 3.5.3 (2019-03-11)"
```

1.2.2 启动管理

R 每次启动时，都会查看一些文件。使用者可以通过改变这些文件对 R 的启动进行管理。在默认的情况下，如果.Renviron和 .Rprofile 存在，R 每次启动都会读入这两个文件。这两个文件的命名方式与常见文件命名方式不一样：它们只有扩展名，没有主文件名。这些文件通常是配置文件，在操作系统中通常是被隐藏的。

.Renviron 主要是用来存储环境变量(environment variable)的。环境变量对整个系统可见，系统内所有程序都能读取环境变量的值。.Renviron中的环境变量对整个 R 系统有效，但是不改变操作系统的设置。文件中环境变量以 `Name = value` 的形式存在，每行设定一个环境变量。这些环境变量可以告诉 R 在哪儿找到某些外部项目、可以用来存储某些用户特定的信息等。系统中 .Renviron文件有三个可能的存储路径：当前用户主目录、R 软件主目录、和 R 软件当前工作区目录。每次会话（session）中 R 只使用一个.Renviron文件和 .Rprofile文件。文件搜索优先级为：当前工作区目录 > 当前用户主目录 > R 软件主目录。例如，如果当前工作区目录内有该文件，则当前用户主目录和 R 软件主目录下的文件将被忽略。读者可以通过下列命令查看当前工作区目录、当前户主目录、和 R 软件安装主目录：

```
getwd()              # 当前工作区目录
Sys.getenv("HOME")   # 当前用户主目录.
Sys.getenv("R_HOME") # R 软件主目录，输出等价于 `R.home()'

## [1] "/Users/lzhan/Documents/ADMIN/Website/WritingR/R-Programing"
```

```
## [1] "/Users/lzhan"
## [1] "/Library/Frameworks/R.framework/Resources"
```

用下面的命令查看 .Renviron 在上述三个目录下是否存在：

```
file.exists(file.path(getwd(), ".Renviron"))
file.exists(file.path(Sys.getenv("HOME"), ".Renviron"))
file.exists(file.path(Sys.getenv("R_HOME"), ".Renviron"))

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

上述运行结果显示，作者电脑中只有一个 .Renviron 文件，位于当前用户主目录下 /Users/lzhan。如果电脑中该文件在上述三个目录中都不存在，可以用下面的命令在当前用户主目录下创立该文件：

```
renviron <- file.path(Sys.getenv("HOME"), ".Renviron")
if (!file.exists(renviron)) file.create(renviron)
```

然后，可以用下面命令打开和修改该 .Renviron：

```
file.edit(renviron)
```

.Renviron中需要设置的一个环境变量叫做 R_LIBS_USER，记用户自己安装软件包时的安装路径。如果要把用户自己安装的 R 软件包将安装在当前用户目录的下 R 文件夹内，则可以在 .Renviron 中加入下面一行代码：

```
R_LIBS_USER = "~/R" # 目录中不要有空格
```

如果上述路径不存在，我们也可以用下面的命令创建该路径：

```
path <- file.path("~", "R")
if (!dir.exists(path)) file.create(path)
```

.Renviron 需要设置的另外一个环境变量叫做 R_MAX_VSIZE。从版本 R 3.5.0 开始，R 语言有了一个对内存分配的限制。R 3.5.0版本介绍里有下面一段话：

> The environment variable R_MAX_VSIZE can now be used to specify the maximal vector heap size. On macOS, unless specified by this environment variable, the maximal vector heap size is set to the maximum of 16GB and the available physical memory. This is to avoid having the R process killed when macOS over-commits memory. – CHANGES IN R 3.5.0

当输入的数据超过 16GB 或超过电脑的物理内存时，电脑会产生如下报错：

```
Error: vector memory exhausted (limit reached?)
```

为了解决这个问题。我们需要在环境变量文件 .Renviron 中加入下面一行：

```
R_MAX_VSIZE = 100GB
```

我们还可以通过下面的命令查看所有可能的环境变量和查看当前用户下的环境变量设置：

```
Sys.info()                    # 输出系统和用户信息
Sys.getenv()                  # 获取当前环境变量设置
help("environment variables") # 获取所有可能的环境变量，等价于：
?"environment variables"
```

.Rprofile 是 R 每次启动时都会首先自动运行的一系列 R 语言代码。.Rprofile 也有与 .Renviron 文件一样的三个可能存在目录，有相同的优先级顺序。我们也可以运行类似命令查看该文件是否存在、打开和修改该文件。

```
file.exists(file.path(getwd(), ".Rprofile"))
file.exists(file.path(Sys.getenv("HOME"), ".Rprofile"))
file.exists(file.path(Sys.getenv("R_HOME"), ".Rprofile"))

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

```
Rprofile <- file.path(Sys.getenv("HOME"), ".Rprofile")
if (!file.exists(Rprofile)) file.create(Rprofile)
```

```
file.edit(Rprofile)
```

.Rprofile 与 .Renviron 不同的地方在于，该文件存储的是一系列合法的 R 语言代码。用户如果希望 R 每次启动都运行某些 R 语言代码，或有一些个人选项要配置，都可以写在这个文件里。例如，默认情况下，我们每次安装 R 扩展包都需要人为的选定 CRAN 的镜像站点。想要省掉这个麻烦，可以在 .Rprofile 文件中加入下面这段代码，告诉 R 始终使用兰州大学的镜像站点下载扩展包：

```
options(
 repos = c(
  CRAN = "https://mirror.lzu.edu.cn/CRAN/")
)
```

每个操作系统都有一个叫 PATH 的环境变量。环境变量是用冒号 ':' 分割开来的一系列路径。当用户在命令窗口（Linux 系统下叫终端，Terminal）窗口下输入一个命令时，系统就会在这一系列路径下搜索该命令：如果找到就运行它，如果找不到就报错。因为作者有时需要把某些 md 或 Rmd 文件转化成 pdf 文件，这就需要用到某些 Tex 引擎。作者电脑的 TexLive 安装在路径 /Library/TeX/texbin下。所以可以在 .Rprofile 添加下面一行把 TexLive 添加到 PATH 环境变量中：

```
old_path = Sys.getenv("PATH")
new_path = paste(old_path, "/Library/TeX/texbin", sep = ":")
Sys.setenv("PATH" = new_path)
rm(old_path, new_path)
```

把编码方式设置为 'UTF-8'：

```
cat('Sys.setlocale(, "en_US.UTF-8")',
    file = '~/.Rprofile', append = TRUE)
```

有关启动管理的更多设置，可以用下面命令进行了解：

```
help("Startup")
```

1.2.3 插件包的安装

R 的所有函数和数据集都存储在 R 的插件包 (package) 里。只有把一个软件包载入的时候，其内容才能被使用。

插件包通过 **library()** 函数从一个软件库（library）中载入。所以软件库是一个包含了安装好的 R 软件包的路径。R 软件的主路径是：R_HOME/library。例如,本机 R 软件的主路径是：/Library/Frameworks/R.framework/Resources。所以默认情况下,插件包的安装路径是：/Library/Frameworks/R.framework/Resources/library。

研究者也可以利用函数 .libPaths() 或通过添加或改变 .Rprofile 中环境变量 R_LIBS 或 R_LIBS_USER 的值来设定新的软件库路径。

```
.Library
.Library.site
.libPaths()

## [1] "/Library/Frameworks/R.framework/Resources/library"
## character(0)
## [1] "/Users/lzhan/R"
## [2] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library"
```

软件包通常有一个命名空间（namespace）。

其中 R 软件的标准包或基础包（base）是 R 源代码的组成部分，在 R 软件启动的时候始终被加载。R 基础包包含了保证 R 正常运行的一些基本函数。

插件包是一个包含了用于拓展 R 的一些文件的存储路径。插件包可以分为源代码版和可执行版。

当一个源代码版插件包包含一些需要编译的代码时，

软件库（library）是插件包（package）的安装路径，所以软件库有时又被称为软件库路径（library directory）或软件库输（library tree）

R 的安装，“pkg” datasets, utils, grDevices, graphics, stats, methods

```
getOption("defaultPackages")

## [1] "datasets" "utils" "grDevices" "graphics" "stats"
"methods"
```

library() 函数完成了两个功能：加载软件包的命名阈并负载软件包，以使得软件包的进入 search path

现在 CRAN 总共有 14035 个软件包，并且在持续增长中。

```
(.packages()) # 载入的R包
```

```
## [1] "knitr"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

```
search()
```

```
##  [1] ".GlobalEnv"        "package:knitr"     "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

```
loadedNamespaces()
```

```
##  [1] "compiler"  "magrittr"  "graphics"  "tools"     "utils"
##  [6] "grDevices" "stats"     "datasets"  "stringi"   "highr"
## [11] "knitr"     "methods"   "stringr"   "xfun"      "base"
## [16] "evaluate"
```

```
library()
base::t
```

```
## function (x)
## UseMethod("t")
## <bytecode: 0x7fe2bc476148>
## <environment: namespace:base>
```

```
getAnywhere(t)
```

```
## A single object matching 't' was found
## It was found in the following places
##   package:base
```

```
##   namespace:base
## with value
##
## function (x)
## UseMethod("t")
## <bytecode: 0x7fe2bc476148>
## <environment: namespace:base>
```

```
(.packages()) # 载入的R包
install.packages("pkg", dependencies = TRUE)
update.packages(ask = FALSE)
devtools::install_github('likanzhan/acqr')
search()
```

可用软件包:
https://cran.r-project.org/web/packages/index.html
软件包的安装位置
已经安装的软件包

```
installed.packages() #不运行
```

1.2.4 Rstudio 的安装

RStudio 是一个 R 的集成开发环境（IDE）。Rstudio 包括一个控制台，一个突出句法、支持脚本直接运行的编辑器，以及一些其他工具，如作图、历史命令、调试和工作区控制等。Rstudio 有一个开源免费版本和一个商业版本，可以在 Windows、Mac 和 Linux 等桌面系统上运行。读者可以登陆 RStudio 的官方网站下载和安装关于 RStudio。

- RStudio 官方网站:https://www.rstudio.com/products/rstudio/download

1.3 获得帮助

- 获得帮助
  https://www.r-project.org/help.html

1. ‘help’ 和 ‘?’ 命令

```
help(lm)
help("lm")
?lm
?"lm"
```

卸载 R

```
sudo rm -Rf /Library/Frameworks/R.framework /Applications/R.app \
   /usr/local/bin/R /usr/local/bin/Rscript
```

第二章　对象

对象：任何一个存在于 R 中的物体都是一个 R 语言的对象。
函数：任何一个发生在 R 中的事件都是一个函数的调用。
接口：与其他软件的接口是 R 语言的一部分。

2.1　基本类型

2.1.1　向量组

2.1.2　列表

2.1.3　语言对象

第三章　表达式求值

第四章　函数

第五章　面向对象编程

第六章　语言的计算

第七章　接口问题

```
sessionInfo()

## R version 3.5.3 (2019-03-11)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.4
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dyli
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.22
##
## loaded via a namespace (and not attached):
## [1] compiler_3.5.3 magrittr_1.5   tools_3.5.3    stringi_1.4.3
```

```
## [5] highr_0.7      stringr_1.4.0  xfun_0.5       evaluate_0.13
```

附录 A　符号说明

- 函数用红色表示，如 **library()**
- 函数的论元用蓝色表示，如 R_LIBS
- 函数的输出结果或文件名，用打字机方式表示，如 `.Rprofile`
- 没有运行的代码或注释，用褐色斜体表示，如 *# R.Version() # 不运行*

附录 B　编码风格

代码的编码风格本身没有对错之分。但当代码编写遵循一定规则时，将更有利于代码的理解、交流和维护。本部分的主要内容来自 *Advanced R* (Wickham, 2015) 一书的第五章。

1. 名称

   文件名称要以 '.R' 结尾，且有一定意义。下面几个是好的例子：

   ```
   fit-models.R
   utility-functions.R
   foo.R
   model.r
   ```

   下面两个例子则不好：

   ```
   foo.R
   model.r
   ```

   变量和函数名称应该用小写字母，并用下划线分割名称中的不同单词。尽量用名词表示变量名称，用动词表示函数名称。名称尽量简短且有意义。下面是两个比较好的例子:

```
day_one
day_1
```

下面几个是不好的例子：

```
dayone
Day_One
first_day_of_the_month
dim1
```

尽量避免用已经存在的函数和变量名称命名自己的对象，避免引起误解。例如下面三个定义就容易引起误解：

```
T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

2. 空格

所有前缀性算子前都要加一个空格，如 '=, +, -, <-' 等。始终在逗号后加一个空格，永远不要在逗号前加空格。下面是一个好例子：

```
average <- mean(feet / 12 + inches, na.rm = TRUE)
```

而下面则是一个坏例子

```
average<-mean(feet/12+inches,na.rm=TRUE)
```

但冒号两侧是不加空格的，无论是单冒号':'、双冒号'::'、还是三连冒号':::'。下面是两个好例子：

```
x <- 1:10
base::get
```

而下面则是两个坏例子：

```
x <- 1 : 10
base :: get
```

只要圆括号不是出现在函数调用中，圆括号前也要加一个空格。下面是两个好例子：

```
if (debug) do(x)
plot(x, y)
```

而下面则是两个坏例子：

```
if(debug)do (x)
plot (x, y)
```

当然棉麻种也可以添加更多的空格，为了让不同行中的等号'=' 或附值符号‘<-’对齐，例如

```
list(
total = a + b + c,
mean  = (a + b + c) / n
)
```

圆括号 '()' 或方括号 "[]" 两侧不能加空格。一个例外情况是括号旁边有一个逗号。下面是两个是好例子：

```
if (debug) do(x)
diamonds[5, ]
```

下面三个例子则不好：

```
if ( debug ) do(x)
x[1,]
x[1 ,]
```

3. 大括号

   左侧大括号 '{' 必须处于一行代码的结尾处：它必须不能在一行的开端，而且它的后面必须重新开始一行。右侧大括号 '}' 必须单独成为一行，除非其后跟着算子 'else'。大括号内的代码必须缩进。下面是两个好例子：

```
if (y < 0 && debug) {
  message("Y is negative")
}

if (y == 0) {
log(x)
} else {
   y ^ x
 }
```

下面这两个例子则不好：

```
if (y < 0 && debug)
message("Y is negative")

if (y == 0) {
  log(x)
}
else {
  y ^ x
}
```

当命令语句非常短时，也可以放在同一行。例如：

```
if (y < 0 && debug) message("Y is negative")
```

4. 代码长度

   每一行代码长度不应该超过 80 个字符。这是因为在字体大小合理的情况下，一行长度为 80 字符的代码是可以打印在一张纸上的。如果你发现自己的代码过长，或许应该考虑把一个复杂的函数切分为不同的函数。

5. 缩进

   代码的缩进要用两个空格，而不能用两个制表符或一个空格加一个制表符。唯一的例外是当某函数的定义包含很多行时，要把第二行缩进到定义开始的位置。例如：

```
long_function <- function(a = "a long argument",
                          b = "another argument",
                          c = "another long argument") {
  # As usual code is indented by two spaces
}
```

6. 赋值

   附值符号用 '<-' 而不能用等号 '='。例如，

```
x <- 5
x = 5
```

7. 注释

   代码的注释部分应该以井号加一个空格开始 '# '。注视应该解释原因而不是解释发生了什么。可以用注释符号加一系列 '-' 或 '=' 的方式把代码分成不同的区块，方便阅读。

参考文献

R Core Team. (2018). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria. Retrieved from https://www.R-project.org/

Wickham, H. (2015). *Advanced R*. Broken Sound Parkway, NW: CRC Press.