R 语言-代码规范 (Google's R Style Guide)

Fan Cheng

2018 年 10 月 15 日

原文链接：https://google.github.io/styleguide/Rguide.xml?from=timeline&isappinstalled=0

R 是一个高级编程语言主要用于统计计算和图形。R 编程风格指南的目标是使我们的 R 代码更容易阅读、分享和验证。以下 R 代码规则是在谷歌的整个 R 合作用户社区下进行设计的。

1 符号和命名 (Notation and Naming)

1.1 文件名 (File names)

File names should end in .R and, of course, be meaningful.

GOOD: predict_ad_revenue.R

BAD: foo.R

1.2 标识符（Identifiers）

不要使用下划线 (_) 或连字符 (-) 标识符。标识符应按照以下命名约定。变量名的首选形式都是小写字母和单词分开用点 (variable.name), 但也接受 variableName; 函数名最初的大写字母和没有点 (FunctionName);

variable.name is preferred, variableName is accepted

GOOD: avg.clicks

OK: avgClicks

BAD: avg_Clicks

FunctionName

GOOD: CalculateAvgClicks

BAD: calculate_avg_clicks , calculateAvgClicks

Make function names verbs.

Exception: When creating a classed object, the function name.

2 语法（Syntax）

每行最大长度（Line Length）The maximum line length is 80 characters.

2.1 缩进（Indentation）

当缩进代码, 使用两个空间。绝不使用制表符或混合制表符和空格。例外: 括号内发生换行时, 使其与括号内的第一个字符对齐。

2.2 间距（Spacing）

当使用所有二进制运算符 (如 =,+,-,<, 等) 在两端空格。例外: 当符号 = 是函数调用时的传递参数周围不用空格隔开。不要在符号“，”前空格隔开，但需要在“，”后添加空格。

GOOD:

```
tab.prior <- table(df[df$days.from.opt < 0, "campaign.id"])
total <- sum(x[, 1])
total <- sum(x[1, ])
```

BAD:

```
tab.prior <- table(df[df$days.from.opt<0, "campaign.id"])  # Needs spaces around '<'
tab.prior <- table(df[df$days.from.opt < 0,"campaign.id"]) # Needs a space after the comma
tab.prior<- table(df[df$days.from.opt < 0, "campaign.id"])  # Needs a space before <
tab.prior<-table(df[df$days.from.opt < 0, "campaign.id"])  # Needs spaces around <
total <- sum(x[,1])  # Needs a space after the comma
total <- sum(x[ ,1])  # Needs a space after the comma, not before
```

在左括号之前添加一个空格，除了函数的调用。

GOOD:

```
if (debug)
```

BAD:

```
if(debug)
```

Extra spacing (i.e., more than one space in a row) is okay if it improves alignment of equals signs or arrows (<-).

```
plot(x    = x.coord,
     y    = data.mat[, MakeColName(metric, ptiles[1], "roiOpt")],
     xlab = "dates",
     ylab = metric,
     main = (paste(metric, " for 3 samples ", sep = "")))
```

2.3 花括号（Curly Braces）

一个左括号不应该自己一行; 而一个右括号应该总是一行。当一个代码块是一个单独声明时你可以不适用花括号。但是，你必须考虑其他相同的情况，以保持一致。

```
if (is.null(ylim)) {  ylim <- c(0, 0.06) }
```

or (but not both)

```
if (is.null(ylim))  ylim <- c(0, 0.06)
```

Always begin the body of a block on a new line. BAD:

```
if (is.null(ylim)) ylim <- c(0, 0.06)
if (is.null(ylim)) {ylim <- c(0, 0.06)}
```

2.4 花括号与 else

一个 else 语句应该总是被花括号包围在同一行。

Good:

```
if (condition) {
  one or more lines
} else {
  one or more lines
}
```

Bad:

```
if (condition) {
  one or more lines
}
else {
  one or more lines
}
```

2.5 赋值（Assignment）

Use <-, not =, for assignment.

GOOD:

```
x <- 5
```

BAD:

```
x = 5
```

3 Organization

3.1 总体布局和排序（General Layout and Ordering）

如果每个人都使用相同的一般顺序, 我们能够更快和更容易阅读和理解彼此的脚本。一般开头需包含：1. 版权声明注释 2. 作者评论 3. 文件描述的评论, 包括程序的目的, 输入和输出 4.source() 和 library() 声明 5. 函数定义 6. 已执行的语句

单元测试应该在一个单独的文件名为 originalfilename_test.R。

3.2 代码注释（Commenting Guidelines）

简短的注释可以放置在代码之后，用空格 + # + 空格隔开, 较长的注释可以单独一行。

```
# Create histogram of frequency of campaigns by pct budget spent.
hist(df$pct.spent,
     breaks = "scott",  # method for choosing number of buckets
     xlab   = "Fraction of budget spent",
     ylab   = "Frequency (count of campaignids)")
```

3.3 函数定义和调用（Function Definitions and Calls）

函数定义应该首先列出参数没有默认值, 紧随其后的是那些有默认值的。在函数定义和函数调用时, 允许多个参数一行，但是换行只允许在参数之间进行。

GOOD：

```
PredictCTR <- function(query, property, num.days,
                       show.plot = TRUE)
```

BAD:

```
PredictCTR <- function(query, property, num.days, show.plot =
                       TRUE)
```

理想情况下, 单元测试应该作为样本函数调用 (共享库例程)。

3.4 函数说明 (Function Documentation)

在函数定义之下应该包含一个分段注释。这些注释应该包含一句关于函数的描述，一段关于该函数的参数列表的的描述 (包括数据类型), 和一个返回值的描述。这些注释需具有足够的描述性的, 调用者可以通过阅读注释即可懂得如何调用该函数。

Example

```
CalculateSampleCovariance <- function(x, y, verbose = TRUE) {
  # Computes the sample covariance between two vectors.
```

```
  #
  # Args:
  #   x: One of two vectors whose sample covariance is to be calculated.
  #   y: The other vector. x and y must have the same length, greater than one,
  #      with no missing values.
  #   verbose: If TRUE, prints sample covariance; if not, not. Default is TRUE.
  #
  # Returns:
  #   The sample covariance between x and y.

  n <- length(x)
  # Error handling
  if (n <= 1 || n != length(y)) {
    stop("Arguments x and y have different lengths: ",
         length(x), " and ", length(y), ".")
  }
  if (TRUE %in% is.na(x) || TRUE %in% is.na(y)) {
    stop(" Arguments x and y must not have missing values.")
  }
  covariance <- var(x, y)
  if (verbose)
    cat("Covariance = ", round(covariance, 4), ".\n", sep = "")
  return(covariance)
}
```

Functions 错误应该使用 `stop()` 进行提醒。

4 最后的话

Use common sense and BE CONSISTENT.

If you are editing code, take a few minutes to look at the code around you and determine its style. If others use spaces around their if clauses, you should, too. If their comments have little boxes of stars around them, make your comments have little boxes of stars around them, too. The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you are saying, rather than on how you are saying it. We present global style rules here so people know the vocabulary. But local style is also important. If code you add to a file looks drastically different from the existing code around it, the discontinuity will throw readers out of their rhythm when they go to read it. Try to avoid this. OK, enough writing about writing code; the code itself is much more interesting. Have fun!