L6: R function

Nan He, School of Basic Medical Sciences, Southern Medical University

2026-02-12

目录

R 中的函数（function）与包内函数编写规范

本节我们将系统介绍如何在 R 中定义和使用函数、函数参数与返回值的设计、作用域（scope）问题；随后介绍在 R 包中撰写函数的规范（使用 roxygen2 注释生成文档、@export 等），并给出用 `source()` 和调试工具单步调试函数的实用流程。

1. 函数的基本形式

R 中最基本的函数定义形式：

```
fn_name <- function(arg1, arg2 = 1, ...) {
  # 函数体
  result <- arg1 + arg2
  return(result)
}
```

要点：

- 函数由 `function()` 创建，参数可以有默认值；
- `...` 用于接收额外可变参数，常用于透传给其它函数（例如绘图或 `read_*` 系列）；
- 如果没有显式 `return()`，函数会返回最后一个求值表达式的值。

下面用一个生信常见的小函数举例：计算 TPM（Transcripts Per Million）从基因计数和基因长度（单位 bp）得到 TPM 值。

```
compute_tpm <- function(counts, length_bp) {
  # counts: 命名向量或数值向量，表示原始计数
  # length_bp: 与 counts 对应的基因长度，单位 bp
  if (length(counts) != length(length_bp)) stop("counts 与 length_bp 长度不一致")

  length_kb <- length_bp / 1000
  rpk <- counts / length_kb
  tpm <- rpk / sum(rpk, na.rm = TRUE) * 1e6
  return(tpm)
}

# 测试
counts <- c(GeneA = 500, GeneB = 1000, GeneC = 50)
length_bp <- c(GeneA = 2000, GeneB = 1000, GeneC = 1500)
compute_tpm(counts, length_bp)
```

```
##     GeneA     GeneB     GeneC
## 194805.19 779220.78  25974.03
```

这段代码展示了常见的参数检查（长度一致性）、单位转换、以及向量化操作（避免循环）。

2. 参数设计与默认值

良好的参数设计可以提高函数的可用性与可维护性。

建议：

- 把必须提供的参数放在最前面；带有合理默认值的参数放后面；
- 使用具说明性的参数名（如 `counts`, `length_bp`, `method`）；
- 对可能为 `NULL` 或 NA 的参数，要在函数内部处理或给出明确定义；
- 使用 `...` 以支持可扩展性，但在文档中明确说明 `...` 会传递给哪个底层函数。

示例：为 `compute_tpm` 添加 `na.rm` 与 `min_counts` 两个参数。

```
compute_tpm2 <- function(counts, length_bp, na.rm = TRUE, min_counts = 0) {
  if (length(counts) != length(length_bp)) stop("counts 与 length_bp 长度不一致")
  if (na.rm) {
    keep <- !is.na(counts) & !is.na(length_bp)
```

```
    counts <- counts[keep]
    length_bp <- length_bp[keep]
  }
  counts[counts < min_counts] <- 0
  length_kb <- length_bp / 1000
  rpk <- counts / length_kb
  tpm <- rpk / sum(rpk, na.rm = TRUE) * 1e6
  return(tpm)
}

compute_tpm2(counts, length_bp, min_counts = 10)
```

```
##     GeneA     GeneB     GeneC
## 194805.19 779220.78  25974.03
```

3. 返回值设计

函数应尽量返回明确且易用的数据结构：

- 简单计算返回向量或标量；
- 复杂处理返回 `list` 或 `data.frame`，并在文档中说明每个字段含义；
- 如果函数会改变全局变量，尽量避免——优先返回结果并让调用者决定是否赋值。

示例：一个返回多个结果的统计函数。

```
summary_stats <- function(x) {
  x <- as.numeric(x)
  res <- list(
    n = length(x),
    mean = mean(x, na.rm = TRUE),
    sd = sd(x, na.rm = TRUE)
  )
  return(res)
}

summary_stats(c(1,2,3,NA,5))
```

```
## $n
## [1] 5
##
## $mean
## [1] 2.75
```

```
##
## $sd
## [1] 1.707825
```

4. 作用域（Scope）与副作用

R 中的变量作用域遵循词法作用域（lexical scope）。函数内部创建的变量默认是局部的；若想修改外部变量，需显式使用 `<<-` 或 `assign()`，一般不推荐。

示例：局部变量与修改全局变量对比

```
global_var <- 0

fn_local <- function(x) {
  y <- x + 1  # 局部变量 y
  return(y)
}

fn_modify_global <- function(x) {
  global_var <<- x  # 修改外部变量（不推荐）
}

fn_local(2)
```

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

```
global_var
```

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

```
fn_modify_global(5)
global_var
```

```
## [1] 5
```

说明：除非确有必要，否则不要让函数产生副作用（修改外部状态）。函数应当是”纯函数”（相对纯粹）：相同输入应返回相同输出。

闭包（closure）是 R 的一个重要特性：函数返回另一个函数，并保留创建时的环境。举例：创建一个带默认正则化参数的 scaler。

```
make_scaler <- function(center = TRUE, scale = TRUE) {
  function(x) {
    x <- as.numeric(x)
    if (center) x <- x - mean(x, na.rm = TRUE)
    if (scale) x <- x / sd(x, na.rm = TRUE)
```

```
    x
  }
}

sc <- make_scaler(center = TRUE, scale = FALSE)
sc(c(1,2,3,4))
```

```
## [1] -1.5 -0.5  0.5  1.5
```

5. 编写可测试、可读的函数（最佳实践）

小结性的建议：

1. 单一职责：函数只做一件事；
2. 输入验证：对输入类型、长度、NA 等进行检查并给出明确错误信息；
3. 明确返回值类型和结构；
4. 写示例与单元测试（测试框架如 `testthat`）；
5. 注释清楚但不要冗余；函数复杂时拆分为小函数。

示例：改写一个更健壮的 TPM 计算函数（带参数检查与文档示例）。

```
compute_tpm_safe <- function(counts, length_bp, min_counts = 0, na.rm = TRUE) {
  if (!is.numeric(counts)) stop("counts 必须为数值向量")
  if (!is.numeric(length_bp)) stop("length_bp 必须为数值向量")
  if (length(counts) != length(length_bp)) stop("counts 与 length_bp 长度不一致")

  if (na.rm) {
    keep <- !is.na(counts) & !is.na(length_bp)
    counts <- counts[keep]
    length_bp <- length_bp[keep]
  }

  counts[counts < min_counts] <- 0
  length_kb <- length_bp / 1000
  if (any(length_kb == 0)) stop(" 基因长度不能为 0")

  rpk <- counts / length_kb
  denom <- sum(rpk, na.rm = TRUE)
  if (denom == 0) return(rep(0, length(rpk)))
  tpm <- rpk / denom * 1e6
  names(tpm) <- names(counts)
  return(tpm)
```

```
}

compute_tpm_safe(counts, length_bp)
```

```
##     GeneA     GeneB     GeneC
## 194805.19 779220.78  25974.03
```

6. 在 R 脚本中组织函数并用 source() 调试

在日常分析中，我们通常把一组相关函数放在单独的脚本文件，如 R/functions_tpm.R，然后在主脚本中 source() 加载：

```
# 在主脚本中
source('R/functions_tpm.R')
res <- compute_tpm_safe(my_counts, my_lengths)
```

调试技巧：

- debug(func)：在下一次调用 func() 时进入交互调试，可以单步执行；
- debugonce(func)：仅调试下一次调用；
- 在函数内部插入 browser()：运行到这里会进入交互式调试；
- traceback()：当出错后查看调用栈；
- trace()：在函数运行时插入临时代码（如打印变量）；

示例：演示 debug() 与 browser()。

```
tmp_fn <- function(x) {
  a <- sqrt(x)
  b <- log(a)
  return(b)
}

# 使用 debug(): 下次调用时会进入调试模式
debug(tmp_fn)
tmp_fn(10)
```

```
## debugging in: tmp_fn(10)
## debug: {
##     a <- sqrt(x)
##     b <- log(a)
##     return(b)
## }
## debug: a <- sqrt(x)
```

```
## debug: b <- log(a)
## debug: return(b)
## exiting from: tmp_fn(10)
```

```
## [1] 1.151293
```

```
undebug(tmp_fn)

# 在函数中插入 browser()
tmp_fn2 <- function(x) {
  a <- sqrt(x)
  browser()
  b <- log(a)
  return(b)
}
try(tmp_fn2(10))
```

```
## Called from: tmp_fn2(10)
## debug: b <- log(a)
## debug: return(b)
```

```
## [1] 1.151293
```

```
# 出错后的 traceback()
bad_fn <- function(x) {
  stop(" 故意出错")
}
tryCatch(bad_fn(1), error = function(e) message(" 出错: ", e$message))
# 如果没有 catch，运行后可用 traceback() 查看
```

在 source() 调试时常见问题：

- source() 默认为全局环境执行（local = FALSE），如果想把函数载入某个环境可以设置 local = TRUE 或指定环境；
- 推荐把函数脚本放到项目 R/ 目录并使用 devtools::load_all() 进行开发加载，模拟包加载环境。

示例：使用 devtools（开发包时）

```
# 在包开发目录下
devtools::load_all()
devtools::document()  # 生成 roxygen 文档
```

7. 在 R 包中撰写函数的标准与 roxygen2 注释

当需要把函数分发或重复复用时，建议把函数放入 R 包中。R 包中的函数放在 `R/` 目录中，每个函数文件应尽量小而聚焦。使用 `roxygen2` 注释可以直接在源代码上方写注释，随后用 `devtools::document()` 自动生成 `NAMESPACE` 和 Rd 文档。

roxygen2 注释模板示例：

```r
#' 计算 TPM
#'
#' 读取原始计数与基因长度，返回 TPM 向量。
#'
#' @param counts 数值向量或命名向量，原始计数
#' @param length_bp 数值向量，与 counts 对应，单位 bp
#' @param na.rm 布尔，是否移除 NA
#' @param min_counts 数值，小于该值的计数将视为 0
#' @return 命名数值向量，TPM
#' @examples
#' counts <- c(G1 = 100, G2 = 50)
#' length_bp <- c(G1 = 2000, G2 = 1000)
#' compute_tpm_safe(counts, length_bp)
#' @export
compute_tpm_safe <- function(counts, length_bp, min_counts = 0, na.rm = TRUE) {
  # 函数体同上（为示例重复定义）
  if (!is.numeric(counts)) stop("counts 必须为数值向量")
  if (!is.numeric(length_bp)) stop("length_bp 必须为数值向量")
  if (length(counts) != length(length_bp)) stop("counts 与 length_bp 长度不一致")
  if (na.rm) {
    keep <- !is.na(counts) & !is.na(length_bp)
    counts <- counts[keep]
    length_bp <- length_bp[keep]
  }
  counts[counts < min_counts] <- 0
  length_kb <- length_bp / 1000
  if (any(length_kb == 0)) stop(" 基因长度不能为 0")
  rpk <- counts / length_kb
  denom <- sum(rpk, na.rm = TRUE)
  if (denom == 0) return(rep(0, length(rpk)))
  tpm <- rpk / denom * 1e6
  names(tpm) <- names(counts)
  return(tpm)
```

```
}
```

关键点：

- `#'` 开头的注释会被 roxygen2 解析；
- 常用标签：`@title`（可选）、`@description`、`@param`、`@return`、`@examples`、`@export`、`@importFrom`；
- `@export` 会在 NAMESPACE 中添加 `export()`，使函数对包用户可见；
- `@importFrom pkg fun` 可用于在 NAMESPACE 中声明函数依赖，避免在运行时使用 `pkg::fun`。但在包内部代码中推荐使用 `pkg::fun` 或在 DESCRIPTION 中 Declare Imports。

生成文档与安装流程（简要）：

1. 在包工程根目录下，编辑 `DESCRIPTION`（填写 Package, Title, Version, Authors@R, Depends/Imports）；
2. 把函数放到 `R/` 目录；
3. 运行 `devtools::document()` 自动生成 `man/` 下的 Rd 文件和 `NAMESPACE`；
4. 运行 `devtools::check()` 检查包规范；
5. `devtools::install()` 或 `remotes::install_local()` 安装包。

示例命令（开发者机器上执行）：

```
devtools::document()
devtools::check()
devtools::install()
```

8. 实战：把函数写在脚本中并转为包函数的最小实践

步骤演示（本地项目）：

1. 在项目 `R/` 下创建 `tpm.R`，把 `compute_tpm_safe()` 与 roxygen 注释放入其中；
2. 在 `DESCRIPTION` 中写明包名和依赖（如 `Imports: dplyr`）；
3. 使用 `devtools::load_all()` 调试并在交互中测试；
4. 写 `tests/testthat/test-tpm.R` 做单元测试。

示例 `tests`（简要）：

```
# tests/testthat/test-tpm.R
library(testthat)
test_that("TPM 计算基本正确", {
  counts <- c(G1 = 10, G2 = 0)
  lengths <- c(G1 = 1000, G2 = 1000)
  t <- compute_tpm_safe(counts, lengths)
  expect_equal(sum(t), 1e6)
})
```

```
## Test passed
```

9. 常见陷阱与调试建议

- 注意参数向量的命名和对齐（`names()`），不要依赖顺序匹配；
- 小心 `...` 的透传导致参数被错误地传给下游函数；
- 写示例时要包含边界条件（零向量、全 NA、长度不匹配）；
- 使用 `testthat` 保持回归稳定性；
- `devtools::check()` 会捕捉到文档错误、未导出的符号等问题，务必在发布前运行。

调试流程建议：

1. 在交互式环境用 `source()` 或 `devtools::load_all()` 加载函数；
2. 使用 `debugonce()` 或在函数中插入 `browser()` 单步调试；
3. 出错后用 `traceback()` 查找调用栈，并使用 `rlang::last_error()` 获取更详细信息（若使用 `rlang`）；
4. 写最小可复现示例，方便定位问题并写测试。

10. 小结

本节覆盖了：

- R 函数定义与参数设计；
- 返回值与作用域（避免副作用）；
- 闭包与高阶函数的简单示例；
- 在脚本中组织函数并用 `source()`/`debug()` 调试；
- 将函数写入 R 包、roxygen2 注释与文档自动生成；
- 单元测试与 `devtools` 流程建议。