R 语言元编程阅读笔记

Cheng Jun

https://github.com/chengjun90

2018 年 5 月 30 日

目录

1 准备工作

1.1 说明

Mailund（2017）的 Metaprogramming in R: Advanced Statistical Programming for Data Science,Analysis and Finance 是一本关于 R 语言元编程的书。

在 2017 年在读完 Wickham 写的 Advanced R 对元编程有一定的了解，还写了两份分享：

- R 语言进阶 | 非标准计算 base
- R 语言进阶 | 非标准计算 tidyeval

现在 Metaprogramming in R 是专门讲元编程，可以读一读。以下是阅读过程中的一些笔记。

1.2 写作环境

版本信息：

```
devtools::session_info()
```

```
## Session info -------------------------------------------------------------

##  setting  value
##  version  R version 3.5.0 (2018-04-23)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  Chinese (Simplified)_China.936
##  tz       Asia/Taipei
##  date     2018-05-30

## Packages -----------------------------------------------------------------

##  package    * version date       source
##  backports    1.1.2   2017-12-13 CRAN (R 3.5.0)
##  base       * 3.5.0   2018-04-23 local
##  compiler     3.5.0   2018-04-23 local
##  datasets   * 3.5.0   2018-04-23 local
##  devtools     1.13.5  2018-02-18 CRAN (R 3.5.0)
##  digest       0.6.15  2018-01-28 CRAN (R 3.5.0)
```

```
##  evaluate    0.10.1  2017-06-24 CRAN (R 3.5.0)
##  graphics  * 3.5.0   2018-04-23 local
##  grDevices * 3.5.0   2018-04-23 local
##  htmltools   0.3.6   2017-04-28 CRAN (R 3.5.0)
##  knitr       1.20    2018-02-20 CRAN (R 3.5.0)
##  magrittr    1.5     2014-11-22 CRAN (R 3.5.0)
##  memoise     1.1.0   2017-04-21 CRAN (R 3.5.0)
##  methods   * 3.5.0   2018-04-23 local
##  Rcpp        0.12.17 2018-05-18 CRAN (R 3.5.0)
##  rmarkdown   1.9     2018-03-01 CRAN (R 3.5.0)
##  rprojroot   1.3-2   2018-01-03 CRAN (R 3.5.0)
##  rticles     0.4.1   2017-05-16 CRAN (R 3.5.0)
##  stats     * 3.5.0   2018-04-23 local
##  stringi     1.2.2   2018-05-02 CRAN (R 3.5.0)
##  stringr     1.3.1   2018-05-10 CRAN (R 3.5.0)
##  tools       3.5.0   2018-04-23 local
##  utils     * 3.5.0   2018-04-23 local
##  withr       2.1.2   2018-03-15 CRAN (R 3.5.0)
##  yaml        2.1.19  2018-05-01 CRAN (R 3.5.0)
```

1.3 许可协议说明

2 Anatomy of a Function

2.1 理解函数

上来就告诉读者函数有三个部分 formals，body 和 environment。

```
myfun <- function(x){
    mean(x)
}
formals(myfun)
```

```
## $x
```

```
body(myfun)
```

```
## {
##     mean(x)
## }
```

```
environment(myfun)
```

```
## <environment: R_GlobalEnv>
```

```
myfun <- function(x = 1, y = 2) {
    z = x + y
    return(z)
}
formals(myfun)
```

```
## $x
## [1] 1
##
## $y
## [1] 2
```

```
body(myfun)
```

```
## {
##     z = x + y
##     return(z)
## }
```

```
environment(myfun)
```

```
## <environment: R_GlobalEnv>
```

上面两个例子就可以发现 formals 主要是函数参数和参数的默认值。用 str 一看便知。

```
str(formals(myfun))
```

```
## Dotted pair list of 2
##  $ x: num 1
##  $ y: num 2
```

说完了 formals 来看 body。

```
body(myfun)
```

```
## {
##     z = x + y
##     return(z)
## }
```

str(body(myfun)) 的结果看着头大。

```
str(body(myfun))
```

```
##  language {  z = x + y; return(z) }
##  - attr(*, "srcref")=List of 3
##   ..$ : 'srcref' int [1:8] 1 33 1 33 33 33 1 1
##   .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x000000001c
##   ..$ : 'srcref' int [1:8] 2 5 2 13 5 13 2 2
##   .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x000000001c
##   ..$ : 'srcref' int [1:8] 3 5 3 13 5 13 3 3
##   .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x000000001c
##  - attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x000000001c94aff8
##  - attr(*, "wholeSrcref")= 'srcref' int [1:8] 1 0 4 1 0 1 1 4
##   ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile' <environment: 0x000000001c94a
```

```
class(body(myfun))
```

```
## [1] "{"
```

```
typeof(body(myfun))
```

```
## [1] "language"
```

eval 可以对 body 进行求值，不过要给参数传值。

```
eval(body(myfun), list(x = 1, y = 2))
```

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

2.2 修改函数

```
myfun <- function(x = 1, y = 2) {
    z = x + y
}
print(myfun(3,4))
```

```
## [1] 7
```

```
body(myfun)
```

```
## {
##     z = x + y
## }
```

```
fun_body <- body(myfun)
length(fun_body)
```

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

```
fun_body[[1]]
```

```
## `{`
```

```
fun_body[[2]]
```

```
## z = x + y
```

```
fun_body[[2]] <- quote(x * y)
eval(fun_body, list(x = 3, y = 4))
```

```
## [1] 12
```

不过，读完了这一节，还没掌握如何深度地修改 body。

3 Inside a Function Call

路人甲飘过……

4 Expressions and Environments

路人乙飘过……

不搞那么深入。

5 Manipulating Expressions

```
qu <- quote(12 + 3)
qu
```

```
## 12 + 3
```

```
length(qu)
```

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

```
qu[[1]]
```

```
## `+`
```

```
qu[[2]]
```

```
## [1] 12
```

```
qu[[3]]
```

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

```
qu[[3]] <- quote(6)
eval(qu)
```

```
## [1] 18
```

可以对表达式进行修改。

6 Working with Substitutions

6.1 quote

quote 函数主要是用来捕获表达式。

```
quote(1 + 2)
```

```
## 1 + 2
```

```
quote(mean(mtcars$mpg))
```

```
## mean(mtcars$mpg)
```

bquote 是对.() 进行计算，然后再捕获表达式。示例如下。

```
bquote(mean(.(mtcars$mpg)))
```

```
## mean(c(21, 21, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2,
## 17.8, 16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4, 33.9, 21.5,
## 15.5, 15.2, 13.3, 19.2, 27.3, 26, 30.4, 15.8, 19.7, 15, 21.4))
```

```
bquote(.(1 + 2 ) + 3)
```

```
## 3 + 3
```

6.2 parse 和 deparse

parse 和 deparse 是一个相对的函数。

parse 把字符串解析成表达式，而 deparse 则把表达式逆解析为字符串。

```
deparse(quote(1 + 2))
```

```
## [1] "1 + 2"
```

```
class(deparse(quote(1 + 2)))
```

```
## [1] "character"
```

```
parse(text = "1+2")
```

```
## expression(1 + 2)
```

```
class(parse(text = "1+2"))
```

```
## [1] "expression"
```

6.3 substitute

substitute 可以捕捉表达式而不进行计算。substitute() 返回的结果和 quote() 有一定差异。

```
substitute(1 + 2)
```

```
## 1 + 2
```

```
class(substitute(1 + 2))
```

```
## [1] "call"
```

6.4 非标准计算

非标准计算，Nonstandard Evaluation。

```
eval(quote(x * y), envir = list(x = 4, y = 25))
```

```
## [1] 100
```

mean(mtcars$mpg) 可以计算出 mpg 的均值。如果是用非标准计算，有以下两个例子：

```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

```
eval(quote(mean(mpg)), envir = mtcars)
```

```
## [1] 20.09062
```

```
exps <- "mean(mpg)"
eval(parse(text = exps), envir = mtcars)
```

```
## [1] 20.09062
```

上面这两个例子可能是一般 R 语言用户最可能用到的非标准计算。我自己用到就是利用字符函数拼接成一个大字符串，然后拿去给 parse()，然后交给 eval()，在计算的时候要告诉 eval() 你所在的求值环境，即给 envir 传值。

7 结语

这里只是了解一些 R 自带的元编程功能。

更进一步可以参看第三方包rlang，其中设计了一个 *tidy eval* 框架。

哈哈，元编程了解够用就行。路人丙飘过……