R_loops

周世祥

2020/5/2

```
# 在代码中加入 fig.showtext = TRUE, fig.align='center', 同时, 需要 library(showtext)、showtext::shou
#
# “`{r fig.showtext = TRUE, fig.align=’ center’ ,echo=FALSE, message=FALSE, warning=FALSE}
# library(showtext)
# showtext::showtext.begin()
# ggplot(data= china_edge_avg_url_rt, aes(x=date, y=avg_url_rt, group=CDN_ISP, shape=CDN_ISP, colo
# #geom_smooth(size=1)
# geom_line(linetype = 2,size =1) +
# geom_point(size =2)+
# xlab(” ) +
# ylab(‘平均响应时间: ms’ ) +
# labs(title=’ xxxxxxxxxxxxxxxxxx’ )+
# theme(legend.position=” top” ,axis.text.x = element_text(angle = 60, hjust = 0.5, vjust = 0.5),
# text = element_text(color = “black”, size = 13),plot.title = element_text(hjust = 0.5))
# showtext::showtext.end()
# ——————————————————
#  https://blog.csdn.net/u012111465/article/details/79945372
```

R 中的基本循环

```
x <- as.integer(1:5)
```

```
cat("1 + 1 =", 1+1, "\n")
```

```
## 1 + 1 = 2
```

```
cat("1 + 2 =", 1+2, "\n")
```

```
## 1 + 2 = 3
```

```r
cat("1 + 3 =", 1+3, "\n")
```

```
## 1 + 3 = 4
```

```r
cat("1 + 4 =", 1+4, "\n")
```

```
## 1 + 4 = 5
```

```r
cat("1 + 5 =", 1+5, "\n")
```

```
## 1 + 5 = 6
```

```r
for(i in 1:5) {
  cat("1 + ",i, " = ", 1 + i,"\n")
  }
```

```
## 1 +  1  =  2
## 1 +  2  =  3
## 1 +  3  =  4
## 1 +  4  =  5
## 1 +  5  =  6
```

```r
a <- "This is my first loop" %>%
  strsplit(" ") %>% unlist()
print(a[1])
```

```
## [1] "This"
```

```r
print(a[2])
```

```
## [1] "is"
```

```r
print(a[3])
```

```
## [1] "my"
```

```r
print(a[4])
```

```
## [1] "first"
```

```r
print(a[5])
```

```
## [1] "loop"
```

```r
# for(i in 1:length(a)) print(a[i])
for(i in 1:length(a)) {
  cat(" 单词",i, " 是:  ", a[i],"\n")
```

```
}
```

```
## 单词 1 是:  This
## 单词 2 是:  is
## 单词 3 是:  my
## 单词 4 是:  first
## 单词 5 是:  loop
```

简单应用

读取指定文件夹内的全部 csv 文件，读取到 R 中，并将这些数据集首尾相连转换成用户熟悉的数据框格式。

for 循环批量读取文件。

```
filenames <- list.files("RawData/", pattern = "*.csv",full.names = T)

data <- list()

for(i in seq_along(filenames)){ # 借助 seq_along 函数创建 for 循环的范围
  data[[i]] <- read.csv(filenames[i],stringsAsFactors = F)
}
df <- do.call(rbind, data)
str(df)
```

```
## 'data.frame':    150 obs. of  12 variables:
##  $ Sepal.L..Setosa    : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.W..Setosa    : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.L..Setosa    : num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.W..Setosa    : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Sepal.L..Versicolor: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Sepal.W..Versicolor: num  3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
##  $ Petal.L..Versicolor: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
##  $ Petal.W..Versicolor: num  1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
##  $ Sepal.L..Virginica : num  6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 ...
##  $ Sepal.W..Virginica : num  3.3 2.7 3 2.9 3 3 2.5 2.9 2.5 3.6 ...
##  $ Petal.L..Virginica : num  6 5.1 5.9 5.6 5.8 6.6 4.5 6.3 5.8 6.1 ...
##  $ Petal.W..Virginica : num  2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8 1.8 2.5 ...
```

当有脏数据时，或者原始数据文件中列数不同时，上述代码将会在整合部分报错。

while 循环

```
i<- 1
while(i<=5){
  cat("1 + ",i, " = ",1+i,"\n")
  i<- i+1
}
```

```
## 1 +  1  =  2
## 1 +  2  =  3
## 1 +  3  =  4
## 1 +  4  =  5
## 1 +  5  =  6
```

两种循环可以转换。理解循环的三要素：初始值，判别机制，主体。

apply 函数家族

baseR 中的 “*apply” 函数家族。对 R 对象执行一个或多个功能函数，然后返回各自特定的数据格式。baseR 中有 8 个以 apply 结尾的函数。

R 中的数据都是有维度的，二维的点面，三维的立方体，我们可以搜索一下 netCDF 格式的气象数据看看。

for 循环对于一个指定的向量中的不同值，按照相同的运算规则重复执行若干次。

其实 apply 家族中的函数都是可以用 for 循环来书写的，不过相比之下，*apply 更高效。原因就是向量化，也是 R 的特征之一。

lapply “线性” 数据迭代

list+apply 的组合。对一个列表型或向量型数据应用一个函数。

```
x <- 1:10
y <- 10:20
z <- 20:30

lapply(list(x,y,z), mean)
```

```
## [[1]]
## [1] 5.5
##
## [[2]]
```

```
## [1] 15
##
## [[3]]
## [1] 25
```

如果向量中有默认值的话：

```
x <- c(1:10,NA)
  y <- c(10:20,NA)
  z <- c(20:30,NA)
  lapply(list(x,y,z), mean)
```

```
## [[1]]
## [1] NA
##
## [[2]]
## [1] NA
##
## [[3]]
## [1] NA
```

```
lapply(list(x,y,z), mean, na.rm = TRUE)
```

```
## [[1]]
## [1] 5.5
##
## [[2]]
## [1] 15
##
## [[3]]
## [1] 25
```

或者：效果一样。

```
lapply(list(x,y,z), function(x){
  #function 可以无限地扩展 lapply 函数功能，function 中的 x 并不具体指代向量 x，只是一个泛指名称。用户可
    mean(x,na.rm = T)
  })
```

```
## [[1]]
## [1] 5.5
##
## [[2]]
```

```
## [1] 15
##
## [[3]]
## [1] 25
```

lapply 实现批量文件读取

```
filenames <- list.files("RawData/", pattern = "*.csv",full.names = T)
datalist <- lapply(filenames, function(x){
  df <- read.csv(x,stringsAsFactors = F)
  df <- df %>%
    select(1,3,5) # 选取每个文件内的第 1,3,5 列。

  })
names(datalist)
```

```
## NULL
```

```
df <- do.call(rbind, datalist) # 对返回的列表进行整合及检视

str(df)
```

```
## 'data.frame':    150 obs. of  3 variables:
##  $ Sepal.L..Setosa    : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Petal.L..Setosa    : num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Sepal.L..Versicolor: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
```

sapply: 简化版的 lapply

```
lapply(X, FUN, ...)

sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

前三个参数与 lapply 完全一致。使用 sapply，其返回值为一个非列表向量。

```r
x <- 1:10
y <- 10:20
z <- 20:30

sapply(list(x,y,z), mean)
```

```
## [1]  5.5 15.0 25.0
```

```r
x <- c(1:10,NA)
y <- c(10:20,NA)
z <- c(20:30,NA)
sapply(list(x,y,z), mean)
```

```
## [1] NA NA NA
```

```r
sapply(list(x,y,z), mean, na.rm = TRUE)
```

```
## [1]  5.5 15.0 25.0
```

```r
sapply(list(x,y,z), function(x){
   mean(x,na.rm = T)
  })
```

```
## [1]  5.5 15.0 25.0
```

```r
filenames <- list.files("RawData/", pattern = "*.csv",full.names = T)
datalist <- sapply(filenames, function(x){
  df <- read.csv(x,stringsAsFactors = F)
},simplify = F,USE.NAMES = T)
names(datalist)
```

```
## [1] "RawData/iris (1).csv" "RawData/iris (2).csv" "RawData/iris (3).csv"
```

```r
df <- do.call(rbind, datalist)
str(df)
```

```
## 'data.frame':    150 obs. of  12 variables:
##  $ Sepal.L..Setosa    : num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.W..Setosa    : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.L..Setosa    : num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.W..Setosa    : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Sepal.L..Versicolor: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Sepal.W..Versicolor: num  3.2 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 ...
##  $ Petal.L..Versicolor: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
```

```
##  $ Petal.W..Versicolor: num  1.4 1.5 1.5 1.3 1.5 1.3 1.6 1 1.3 1.4 ...
##  $ Sepal.L..Virginica : num  6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 ...
##  $ Sepal.W..Virginica : num  3.3 2.7 3 2.9 3 3 2.5 2.9 2.5 3.6 ...
##  $ Petal.L..Virginica : num  6 5.1 5.9 5.6 5.8 6.6 4.5 6.3 5.8 6.1 ...
##  $ Petal.W..Virginica : num  2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8 1.8 2.5 ...
```

apply 多维数据处理利器

二维 matrix 和三维 array

```
names(df)
```

```
##  [1] "Sepal.L..Setosa"     "Sepal.W..Setosa"     "Petal.L..Setosa"
##  [4] "Petal.W..Setosa"     "Sepal.L..Versicolor" "Sepal.W..Versicolor"
##  [7] "Petal.L..Versicolor" "Petal.W..Versicolor" "Sepal.L..Virginica"
## [10] "Sepal.W..Virginica"  "Petal.L..Virginica"  "Petal.W..Virginica"
```

```
apply(df, 1, mean) %>% as.tibble() %>% .[1:10,] # 1 是按行，2 是按列
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
```

```
## # A tibble: 10 x 1
##    value
##    <dbl>
##  1  3.72
##  2  3.38
##  3  3.66
##  4  3.26
##  5  3.59
##  6  3.75
##  7  3.27
##  8  3.33
##  9  3.42
## 10  3.52
```

```
apply(df, 2, mean) %>% as.tibble()
```

```
## # A tibble: 12 x 1
##    value
##    <dbl>
```

```
##  1 5.01
##  2 3.43
##  3 1.46
##  4 0.246
##  5 5.94
##  6 2.77
##  7 4.26
##  8 1.33
##  9 6.59
## 10 2.97
## 11 5.55
## 12 2.03
```

当数据框内含不同的数值类型时，apply 函数可能无法给出正确的结果。

vapply 迭代的安全模式

是安全版的 sapply，有略微的速度优势。会检查 FUN 参数 X 中每一个数据值，以确保所有值长度和类型均一致。

```
number1 <- list(as.integer(c(1:5)), as.integer(c(5,2,4,7,1)))
number2 <- list(as.integer(c(1:4)), as.integer(c(5,2,4,7,1)))
#number2 列表中第一元素无数字 5，但未见提醒
sapply(number1, function(x) x[x==5] )
```

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

```
sapply(number2, function(x) x[x==5] )
```

```
## [[1]]
## integer(0)
##
## [[2]]
## [1] 5
```

```
vapply(number1, function(x) x[x==5], as.integer(0))
```

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

```
vapply(number2, function(x) x[x==5], as.integer(0) )
```

```
## Error in vapply(number2, function(x) x[x == 5], as.integer(0)): 值的长度必需为1，
##  但FUN(X[[1]])结果的长度却是0
```

mapply 对多个列表进行函数计算

multivariate 多变量的意思。

优雅的循环 purrr 包

purrr 包的出现减少了初学者学习理解循环的难度。### map 函数家族调用函数，对目标数据中的每一个元素进行相同的运算。

```
map(.x, .f, ...)
# .x 是列表或原子向量
# .f 是任意函数
```

```
1:10 %>%
  map(rnorm, n = 10) %>%
  map_dbl(mean) #dbl 返回浮点型
```

```
##  [1] 0.6716515 1.9166627 3.5836338 4.0168177 5.0963960 5.8747074 6.9382283
##  [8] 8.4626993 8.7493359 9.8208052
```

```
library(purrr)
a <- 1:5
b <- 2:4
c <- 1:9
x <- list(a, b, c)
x
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] 2 3 4
##
## [[3]]
## [1] 1 2 3 4 5 6 7 8 9
```

```
walk(x,print) #walk 函数运行结果会返回为输入的值，而不是列表
```

```
## [1] 1 2 3 4 5
## [1] 2 3 4
## [1] 1 2 3 4 5 6 7 8 9
```

```
map_df(x,tibble) %>%
  head()
```

```
## # A tibble: 6 x 1
##   `<int>`
##     <int>
## 1       1
## 2       2
## 3       3
## 4       4
## 5       5
## 6       2
```

```
map_df(x,data.frame) %>%
  head()
```

```
##   .x..i..
## 1       1
## 2       2
## 3       3
## 4       4
## 5       5
## 6       2
```

map2 和 pmap: 对两个及以上的元素进行迭代运算

```
x<- c(" 朱俊鹏"," 杨再林"," 李世成")
y<-c(" 男"," 女"," 男")
z<-c("1801xxx","1802xxx","1803xxx")
map2(x,y,paste)
```

```
## [[1]]
## [1] "朱俊鹏 男"
##
## [[2]]
## [1] "杨再林 女"
##
## [[3]]
## [1] "李世成 男"
```

如果想返回字符串向量。

```
map2_chr(x,y,paste)
```

```
## [1] "朱俊鹏 男" "杨再林 女" "李世成 男"
```

```
pmap_chr(list(x,y,z),paste) # 三个变量内容都整合
```

```
## [1] "朱俊鹏 男 1801xxx" "杨再林 女 1802xxx" "李世成 男 1803xxx"
```

向量操纵函数

accumulate 和 reduce 函数

```
a <- 1:10
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
a %>%
  accumulate(sum)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
a %>%
  accumulate(sum, .init = 0)
```

```
##  [1]  0  1  3  6 10 15 21 28 36 45 55
```

```
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
a %>%
  accumulate(sum, .init = 1)
```

```
##  [1]  1  2  4  7 11 16 22 29 37 46 56
```

```
a %>%
  reduce(sum)#reduce 函数只给出最后的结果
```

```
## [1] 55
```

参考文献

刘健邬书豪，《R 数据科学实战工具详解与案例分析》，机械工业出版社，2019 年 7 月。