talk08 练习与作业

目录

0.1 练习和作业说明

将相关代码填写入以 “‘{r} “‘ 标志的代码框中，运行并看到正确的结果；

完成后，用工具栏里的”Knit” 按键生成 PDF 文档；

将 PDF 文档改为：姓名-学号-talk08 作业.pdf，并提交到老师指定的平台/钉群。

0.2 talk08 内容回顾

- for loop
- apply functions
- dplyr 的本质是遍历
- map functions in purrr package
- 遍历与并行计算

0.3 练习与作业：用户验证

请运行以下命令，验证你的用户名。

如你当前用户名不能体现你的真实姓名，请改为拼音后再运行本作业！

```
Sys.info()[["user"]]
```

```
## [1] "mingyuwang"
```

```
Sys.getenv("HOME")
```

```
## [1] "C:/Users/rhong/Documents"
```

0.4 练习与作业 1：loop 初步

0.4.1 loop 练习（部分内容来自 r-exercises.com 网站）

1. 写一个循环，计算从 1 到 7 的平方并打印 `print`；
2. 取 `iris` 的列名，计算每个列名的长度，并打印为下面的格式：`Sepal.Length (12)`；
3. 写一个 `while` 循环，每次用 `rnorm` 取一个随机数字并打印，直到取到的数字大于 1；
4. 写一个循环，计算 Fibonacci 序列的值超过 1 百万所需的循环数；注：Fibonacci 序列的规则为：`0, 1, 1, 2, 3, 5, 8, 13, 21 ...`；

```
## 代码写这里，并运行；
library("tidyverse")
library(reshape2)
```

```
## 代码写这里，并运行；
# 1. 写一个循环，计算从 1 到 7 的平方并打印 print；
for (i in 1:7) {
  print(i^2)
}
```

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
## [1] 36
## [1] 49
```

```
# 2. 取 iris 的列名，计算每个列名的长度，并打印为下面的格式： Sepal.Length (12)；
for (i in colnames(iris)) {
  print(paste(i, "(Length: ", nchar(i), ")", sep = ""))
}
```

```
## [1] "Sepal.Length(Length: 12)"
## [1] "Sepal.Width(Length: 11)"
## [1] "Petal.Length(Length: 12)"
## [1] "Petal.Width(Length: 11)"
## [1] "Species(Length: 7)"
```

```
# 3. 写一个 while 循环，每次用 rnorm 取一个随机数字并打印，直到取到的数字大于 1；
while (TRUE) {
  x <- rnorm(1)
  print(x)
  if (x > 1) {
    break
  }
}
```

```
## [1] 0.6960949
## [1] 1.488448
```

```r
# 4. 写一个循环，计算 Fibonacci 序列的值超过 1 百万所需的循环数

x <- 0
y <- 1
i <- 0
while (TRUE) {
  z <- x + y
  x <- y
  y <- z
  i <- i + 1
  if (z > 1000000) {
    break
  }
}
print(i)
```

```
## [1] 30
```

0.5 练习与作业 2：loop 进阶，系统和其它函数

0.5.1 生成一个数字 matrix，并做练习

生成一个 100 x 100 的数字 matrix：

1. 行、列平均，用 rowMeans, colMeans 函数；
2. 行、列平均，用 apply 函数
3. 行、列总和，用 rowSums, colSums 函数；
4. 行、列总和，用 apply 函数

5. 使用自定义函数，同时计算：
    - 行平均、总和、sd
    - 列平均、总和、sd

```
## 代码写这里，并运行；
# 生成一个 100 x 100 的数字 matrix:
x <- matrix(rnorm(10000), nrow = 100, ncol = 100)
message("1. 行、列平均，用 rowMeans, colMeans 函数; ")
```

```
## 1. 行、列平均，用rowMeans, colMeans函数；
```

```
rowMeans(x) %>% head(5)
```

```
## [1] -0.11010038  0.14313223 -0.03176710  0.05318953  0.24766986
```

```
colMeans(x) %>% head(5)
```

```
## [1] -0.005230823  0.047767743 -0.096283989  0.023166845 -0.197002327
```

```
message("2. 行、列平均，用 apply 函数")
```

```
## 2. 行、列平均，用 apply 函数
```

```
apply(x, 1, mean) %>% head(5)
```

```
## [1] -0.11010038  0.14313223 -0.03176710  0.05318953  0.24766986
```

```
apply(x, 2, mean) %>% head(5)
```

```
## [1] -0.005230823  0.047767743 -0.096283989  0.023166845 -0.197002327
```

```
message("3. 行、列总和，用 rowSums, colSums 函数; ")
```

```
## 3. 行、列总和，用rowSums, colSums 函数;
```

```
rowSums(x) %>% head(5)
```

```
## [1] -11.010038  14.313223  -3.176710   5.318953  24.766986
```

```
colSums(x) %>% head(5)
```

```
## [1]  -0.5230823   4.7767743  -9.6283989   2.3166845 -19.7002327
```

```
message("4. 行、列总和，用 apply 函数")
```

```
## 4. 行、列总和，用 apply 函数
```

```
apply(x, 1, sum) %>% head(5)
```

```
## [1] -11.010038  14.313223  -3.176710   5.318953  24.766986
```

```
apply(x, 2, sum) %>% head(5)
```

```
## [1]  -0.5230823   4.7767743  -9.6283989   2.3166845 -19.7002327
```

```
message("5. 使用自定义函数，同时计算 行平均、总和、 sd")
```

```
## 5. 使用自定义函数，同时计算 行平均、总和、 sd
```

```
myfun <- function(x) {
  c(mean = mean(x), sum = sum(x), sd = sd(x))
}
apply(x, 1, myfun)[, 1:5]
```

```
##              [,1]       [,2]       [,3]       [,4]       [,5]
## mean  -0.1101004  0.1431322 -0.0317671 0.05318953  0.2476699
## sum  -11.0100379 14.3132229 -3.1767103 5.31895306 24.7669858
## sd     1.0783366  0.8826473  0.9837945 1.14208711  0.9522987
```

```
message("5. 使用自定义函数，同时计算 列平均、总和、 sd ")
```

```
## 5. 使用自定义函数，同时计算 列平均、总和、 sd
```

```
apply(x, 2, myfun)[, 1:5]
```

```
##               [,1]       [,2]        [,3]       [,4]         [,5]
## mean -0.005230823 0.04776774 -0.09628399 0.02316685  -0.1970023
## sum  -0.523082265 4.77677426 -9.62839891 2.31668453 -19.7002327
## sd    1.055106946 0.84492557  0.98739919 0.92131154   1.1024724
```

0.5.2 用 mtcars 进行练习

用 tapply 练习：

1. 用 **汽缸数**分组，计算 **油耗**的 **平均值**；
2. 用 **汽缸数**分组，计算 **wt** 的 **平均值**；

用 dplyr 的函数实现上述计算

```
## 代码写这里，并运行；
# 用 tapply 练习：
message("1. tapply 用汽缸数分组，计算油耗的平均值")
```

```
## 1. tapply 用汽缸数分组， 计算油耗的平均值
```

```
tapply(mtcars$mpg, mtcars$cyl, mean)
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

```
message("1. tapply 用汽缸数分组，计算 wt 的平均值")
```

```
## 1. tapply 用汽缸数分组， 计算 wt 的平均值
```

```
tapply(mtcars$wt, mtcars$cyl, mean)
```

```
##        4        6        8
## 2.285727 3.117143 3.999214
```

```
message("3. dplyr 用汽缸数分组，计算油耗的平均值")
```

```
## 3. dplyr 用汽缸数分组， 计算油耗的平均值
```

```
mtcars %>%
  group_by(cyl) %>%
  summarise(mean_mpg = mean(mpg))
```

```
## # A tibble: 3 x 2
##     cyl mean_mpg
##   <dbl>    <dbl>
## 1     4     26.7
## 2     6     19.7
## 3     8     15.1
```

```
message("3. dplyr 用汽缸数分组，计算 wt 的平均值")
```

```
## 3. dplyr 用汽缸数分组， 计算 wt 的平均值
```

```
mtcars %>%
  group_by(cyl) %>%
  summarise(mean_wt = mean(wt))
```

```
## # A tibble: 3 x 2
##     cyl mean_wt
##   <dbl>   <dbl>
## 1     4    2.29
## 2     6    3.12
## 3     8    4.00
```

0.5.3 练习 lapply 和 sapply

1. 分别用 lapply 和 sapply 计算下面 list 里每个成员 vector 的长度：

```
list( a = 1:10, b = letters[1:5], c = LETTERS[1:8] );
```

2. 分别用 lapply 和 sapply 计算 mtcars 每列的平均值；

```
## 代码写这里，并运行；
# 1. 分别用 lapply 和 sapply 计算下面 list 里每个成员 vector 的长度：
x <- list(a = 1:10, b = letters[1:5], c = LETTERS[1:8])
message("1. 用 lapply 计算 list 里每个成员的长度")
```

```
## 1. 用 lapply 计算 list 里每个成员的长度
```

```
lapply(x, length)
```

```
## $a
## [1] 10
```

```
##
## $b
## [1] 5
##
## $c
## [1] 8
```

```
message("2. 用 sapply 计算 list 里每个成员的长度")
```

```
## 2. 用 sapply 计算 list 里每个成员的长度
```

```
sapply(x, length)
```

```
##  a  b  c
## 10  5  8
```

```
# 2. 分别用 lapply 和 sapply 计算 mtcars 每列的平均值;
message("3. 用 lapply 计算 mtcars 每列的平均值")
```

```
## 3. 用 lapply 计算 mtcars 每列的平均值
```

```
lapply(mtcars, mean)
```

```
## $mpg
## [1] 20.09062
##
## $cyl
## [1] 6.1875
##
## $disp
## [1] 230.7219
##
## $hp
```

```
## [1] 146.6875
##
## $drat
## [1] 3.596563
##
## $wt
## [1] 3.21725
##
## $qsec
## [1] 17.84875
##
## $vs
## [1] 0.4375
##
## $am
## [1] 0.40625
##
## $gear
## [1] 3.6875
##
## $carb
## [1] 2.8125
```

```
message("4. 用 sapply 计算 mtcars 每列的平均值")
```

```
## 4. 用 sapply 计算 mtcars 每列的平均值
```

```
sapply(mtcars, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

0.6 练习与作业 3：loop 进阶，purr 包的函数

0.6.1 map 初步

生成一个变量：

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

用 map 计算：

- 列平均值、总和和中值

```
## 代码写这里，并运行；
df <- tibble(a = rnorm(10), b = rnorm(10), c = rnorm(10), d = rnorm(10))
# 用 map 计算：列 平均值、总和和中值
message(" 列平均值")
```

```
## 列平均值
```

```
df %>% map_dbl(mean)
```

```
##          a          b          c          d
## -0.5717923  0.4927493  0.1643024 -0.2588650
```

```
message(" 列总和")
```

```
## 列总和
```

```
df %>% map_dbl(sum)
```

```
##         a         b         c         d
## -5.717923  4.927493  1.643024 -2.588650
```

```
message(" 列中值")
```

```
## 列中值
```

```
df %>% map_dbl(median)
```

```
##          a          b          c          d
## -0.6967289  0.6434377  0.1437223 -0.2110906
```

0.6.2 map 进阶

用 map 配合 purr 包中其它函数，用 mtcars：

为每一个 **汽缸数**计算燃油效率 mpg 与重量 wt 的相关性（Pearson correlation），得到 p 值和 correlation coefficient 值。

```
## 代码写这里，并运行;

df1 <- mtcars %>%
  split(.$cyl) %>%
  map_dbl(~ cor.test(.$mpg, .$wt)$p.value)
df2 <- mtcars %>%
  split(.$cyl) %>%
  map_dbl(~ cor.test(.$mpg, .$wt)$estimate)
data.frame(df1, df2) %>%
  rownames_to_column(var = "cyl") %>%
  rename(correlation = df2, pValue = df1)
```

```
##   cyl     pValue correlation
## 1   4 0.01374278  -0.7131848
## 2   6 0.09175766  -0.6815498
## 3   8 0.01179281  -0.6503580
```

0.6.3 keep 和 discard

1. 保留 iris 中有 factor 的列，并打印前 10 行；
2. 去掉 iris 中有 factor 的列，并打印前 10 行；

```
## 代码写这里，并运行；
message(" 保留 iris 中有 factor 的列，并打印前 10 行")
```

```
## 保留 iris 中有 factor 的列，并打印前10行
```

```
iris %>% keep(is.factor) %>% head(10)
```

```
##    Species
## 1   setosa
## 2   setosa
## 3   setosa
## 4   setosa
## 5   setosa
## 6   setosa
## 7   setosa
## 8   setosa
## 9   setosa
## 10  setosa
```

```
message(" 去掉 iris 中有 factor 的列，并打印前 10 行")
```

```
## 去掉 iris 中有 factor 的列，并打印前10行
```

```
iris %>% discard(is.factor) %>% head(10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1           5.1         3.5          1.4         0.2
## 2           4.9         3.0          1.4         0.2
## 3           4.7         3.2          1.3         0.2
## 4           4.6         3.1          1.5         0.2
## 5           5.0         3.6          1.4         0.2
## 6           5.4         3.9          1.7         0.4
## 7           4.6         3.4          1.4         0.3
## 8           5.0         3.4          1.5         0.2
## 9           4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
```

0.6.4 用 reduce

用 reduce 得到以下三个 vector 中共有的数字：

```
c(1, 3, 5, 6, 10),
  c(1, 2, 3, 7, 8, 10),
  c(1, 2, 3, 4, 8, 9, 10)
```

```
## 代码写这里，并运行；
x <- list(c(1, 3, 5, 6, 10), c(1, 2, 3, 7, 8, 10), c(1, 2, 3, 4, 8, 9, 10))
reduce(x, intersect)
```

```
## [1]  1  3 10
```

0.6.5 运行以下代码，观察得到的结果，并用 tidyverse 包中的 spread 等函数实现类似的结果

```
dfs <- list(
  age = tibble(name = "John", age = 30),
  sex = tibble(name = c("John", "Mary"), sex = c("M", "F")),
  trt = tibble(name = "Mary", treatment = "A")
);

dfs %>% reduce(full_join);
```

```
## 代码写这里，并运行;
dfs <- list(
  age = tibble(name = "John", age = 30),
  sex = tibble(name = c("John", "Mary"), sex = c("M", "F")),
  trt = tibble(name = "Mary", treatment = "A")
)
dfs %>% reduce(full_join)
```

```
## Joining, by = "name"
## Joining, by = "name"

## # A tibble: 2 x 4
##   name    age sex   treatment
##   <chr> <dbl> <chr> <chr>
## 1 John     30 M     <NA>
## 2 Mary     NA F     A
```

```
melt(dfs[[1]], id.vars = "name") %>%
  rbind(melt(dfs[[2]], id.vars = "name")) %>%
  rbind(melt(dfs[[3]], id.vars = "name")) %>%
  spread(key = variable, value = value)
```

```
##   name  age sex treatment
```

```
## 1 John   30    M      <NA>
## 2 Mary <NA>    F         A
```

练习与作业4：并行计算

安装相关包，成功运行以下代码，观察得到的结果，并回答问题

- parallel
- foreach
- iterators

```r
library(parallel)
library(foreach)

##
## 载入程辑包：'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```r
library(iterators)

## 检测有多少个 CPU --
(cpus <- parallel::detectCores())
```

```
## [1] 8
```

```
## 创建一个 data.frame
d <- data.frame(x = 1:10000, y = rnorm(10000))

## make a cluster --
cl <- makeCluster(cpus - 1)

## 分配任务 ...
res <- foreach(row = iter(d, by = "row")) %dopar% {
  return(row$x * row$y)
}
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
## 注意在最后关闭创建的 cluster
stopCluster(cl)

summary(unlist(res))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -30793.01  -2508.99     10.92     67.63   2674.87  30051.96
```

问：你的系统有多少个 CPU？此次任务使用了多少个？**答**：用代码打印出相应的数字即可：

```
## 代码写这里，并运行；
message(" 我的系统有", cpus, " 个 CPU")
```

```
## 我的系统有8个CPU
```

```
message(" 此次任务使用了", cpus - 1, " 个 CPU")
```

```
## 此次任务使用了7个CPU
```