

talk08 练习与作业

目录

0.1 练习和作业说明	1
0.2 talk08 内容回顾	1
0.3 练习与作业：用户验证	2
0.4 练习与作业 1: loop 初步	2
0.5 练习与作业 2: loop 进阶，系统和其它函数	4
0.6 练习与作业 3: loop 进阶， <code>purrr</code> 包的函数	17
0.7 练习与作业 4: <code>pmap</code> 和 <code>map</code> 的更多用法	24
0.8 练习与作业 5: 并行计算	29

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] "lucas"
```

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

```
## [1] "/Users/lucas"
```

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 `runif` 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；
```

```
# Task 01
```

```
for (i in 1:7) {  
  result <- i^2  
  print(result)  
}
```

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

```
## [1] 4
```

```
## [1] 9
```

```
## [1] 16
```

```
## [1] 25
```

```
## [1] 36
```

```
## [1] 49
```

```
# Task 02
```

```
column_names =  
  names(iris)  
for (col_name in column_names) {  
  col_name_length =  
    nchar(col_name)  
  cat(col_name,  
      "(",  
      col_name_length,  
      ")\n")  
}
```

```
## Sepal.Length ( 12 )
```

```
## Sepal.Width ( 11 )
```

```
## Petal.Length ( 12 )
```

```
## Petal.Width ( 11 )
```

```
## Species ( 7 )
```

```
# Task 03
random_number =
  rnorm(1)
while (random_number <= 1) {
  print(random_number)
  random_number =
    rnorm(1)
}
```

```
## [1] 0.4737349
```

```
# Task 04
fibonacci =
  c(0, 1)
count = 2
while (
  tail(fibonacci, 1) +
  fibonacci[length(fibonacci) - 1]
  <= 1000000) {
  next_fib =
    tail(fibonacci, 1) +
    fibonacci[length(fibonacci) - 1]
  fibonacci =
    c(fibonacci, next_fib)
  count =
    count + 1
}
print(count)
```

```
## [1] 31
```

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;  
# Prepare the matrix  
set.seed(123)  
mat =  
  matrix(  
    rnorm(10000),  
    nrow = 100)  
  
# Task 01  
row_means =  
  rowMeans(mat)  
col_means =  
  colMeans(mat)  
  
# Task 02  
# Average number of rows  
row_means_apply =  
  apply(mat, 1, mean)  
# Average number of cols  
col_means_apply =  
  apply(mat, 2, mean)
```

```
# Task 03
row_sums =
  rowSums(mat)
col_sums =
  colSums(mat)

# Task 04
# Sum number of rows
row_sums_apply =
  apply(mat, 1, sum)
# Sum number of cols
col_sums_apply =
  apply(mat, 2, sum)

# Task 05
custom_stats =
  function(x) {
    return(
      c(
        mean =
          mean(x),
        sum =
          sum(x),
        sd =
          sd(x))
    )
  }

# Calculating row statistics
row_stats =
  t(apply(mat, 1, custom_stats))

# Calculating col statistics
col_stats =
```

```
    apply(mat, 2, custom_stats)

# Print the results
print("Row Means:")

## [1] "Row Means:"

head(row_means, n = 3)

## [1] -0.02668423 -0.08311430 -0.18342356

print("Col Means:")

## [1] "Col Means:"

head(col_means, n = 3)

## [1] 0.09040591 -0.10754680 0.12046511

print("Row Means (apply):")

## [1] "Row Means (apply):"

head(row_means_apply, n = 3)

## [1] -0.02668423 -0.08311430 -0.18342356

print("Col Means (apply):")

## [1] "Col Means (apply):"
```

```
head(col_means_apply, n = 3)
```

```
## [1]  0.09040591 -0.10754680  0.12046511
```

```
print("Row Sums:")
```

```
## [1] "Row Sums:"
```

```
head(row_sums, n = 3)
```

```
## [1] -2.668423 -8.311430 -18.342356
```

```
print("Col Sums:")
```

```
## [1] "Col Sums:"
```

```
head(col_sums, n = 3)
```

```
## [1]  9.040591 -10.754680  12.046511
```

```
print("Row Sums (apply):")
```

```
## [1] "Row Sums (apply):"
```

```
head(row_sums_apply, n = 3)
```

```
## [1] -2.668423 -8.311430 -18.342356
```

```
print("Col Sums (apply):")
```

```
## [1] "Col Sums (apply):"
```



```
head(col_sums_apply,n = 3)
```

```
## [1] 9.040591 -10.754680 12.046511
```

```
print("Custom Row Statistics:")
```

```
## [1] "Custom Row Statistics:"
```

```
head(row_stats,n = 3)
```

```
##           mean      sum      sd
## [1,] -0.02668423 -2.668423 0.9589688
## [2,] -0.08311430 -8.311430 0.9371213
## [3,] -0.18342356 -18.342356 1.0357881
```

```
print("Custom Col Statistics:")
```

```
## [1] "Custom Col Statistics:"
```

```
head(col_stats,n = 3)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## mean 0.09040591 -0.1075468 0.1204651 -0.03622291 0.1058509 -0.04229996
## sum 9.04059086 -10.7546798 12.0465110 -3.62229084 10.5850925 -4.22999578
## sd 0.91281588 0.9669866 0.9498790 1.03878122 0.9893458 0.93872815
##           [,7]      [,8]      [,9]      [,10]      [,11]      [,12]
## mean -0.1496441 0.1058735 0.09358971 -0.01919274 0.1199406 -0.01917556
## sum -14.9644141 10.5873547 9.35897144 -1.91927403 11.9940576 -1.91755583
## sd 1.0282366 1.0100100 1.05180659 1.02033166 1.0429982 0.99767147
##           [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## mean -0.002539253 -0.06021128 0.09298927 0.03548695 0.08903406 0.03105473
## sum -0.253925262 -6.02112781 9.29892666 3.54869516 8.90340573 3.10547347
## sd 0.928007043 1.01382919 0.98614286 0.88929622 1.09648466 1.02443405
```

```

##          [,19]      [,20]      [,21]      [,22]      [,23]      [,24]
## mean 0.06518418 0.07288886 -0.1184286 -0.1328738 -0.1038760 0.07303035
## sum  6.51841753 7.28888607 -11.8428579 -13.2873799 -10.3876010 7.30303453
## sd   1.04354899 1.08710170  0.8867420  0.8671375  0.9234582 1.01178395
##          [,25]      [,26]      [,27]      [,28]      [,29]      [,30]
## mean -0.03805934  0.1060876 -0.1149985 -0.01519487 0.04754731 0.09564058
## sum  -3.80593362 10.6087594 -11.4998513 -1.51948694 4.75473105 9.56405766
## sd    0.99456268 0.9751193  1.0359941  0.94454890 1.01726182 1.10530160
##          [,31]      [,32]      [,33]      [,34]      [,35]      [,36]
## mean -0.05389207 -0.1543210 0.03879604 0.05114384 0.005222176 0.06795743
## sum  -5.38920679 -15.4321040 3.87960435 5.11438357 0.522217582 6.79574304
## sd    1.07991180  0.8705622 1.07626589 0.84481290 1.047729024 1.07836057
##          [,37]      [,38]      [,39]      [,40]      [,41]      [,42]
## mean -0.01763709 -0.2499180 0.07249396 0.148551 -0.0830015 -0.04217516
## sum  -1.76370864 -24.9917976 7.24939600 14.855096 -8.3001504 -4.21751621
## sd    0.97329530  0.8768267 0.91526278 1.095703  1.1023174  0.78330286
##          [,43]      [,44]      [,45]      [,46]      [,47]      [,48]
## mean -0.01227263 -0.04126713 -0.05545621 0.06173919 0.05972319 -0.06701344
## sum  -1.22726277 -4.12671273 -5.54562141 6.17391864 5.97231916 -6.70134424
## sd    1.01742000  1.08455734  1.00191901 1.00882076 0.96523497  0.98094922
##          [,49]      [,50]      [,51]      [,52]      [,53]      [,54]
## mean -0.05332069 -0.08863744 0.06803094 0.03556477 -0.0782075 -0.03531996
## sum  -5.33206943 -8.86374425 6.80309354 3.55647721 -7.8207500 -3.53199625
## sd    0.94231314  1.11142003 0.88940618 0.86387437 0.9641162  1.00713013
##          [,55]      [,56]      [,57]      [,58]      [,59]      [,60]
## mean  0.1002368 0.08108622 -0.02297392 0.01378047 0.03658164 0.1460111
## sum  10.0236830 8.10862250 -2.29739242 1.37804736 3.65816433 14.6011054
## sd    1.0771408 1.01709562 1.01292952 0.98644219 1.02524416 1.0675179
##          [,61]      [,62]      [,63]      [,64]      [,65]      [,66]
## mean 0.05149166 -0.2379236 0.08115196 -0.1689926 0.1111330 0.1544749
## sum  5.14916559 -23.7923602 8.11519559 -16.8992567 11.1133042 15.4474925
## sd   0.93443052  1.1540099 1.04262975  1.0431884 0.9165161 0.9676328
##          [,67]      [,68]      [,69]      [,70]      [,71]      [,72]

```

```
## mean -0.09230807 -0.1635888 0.0468654 -0.1113391 0.05067045 -0.04929186
## sum -9.23080748 -16.3588773 4.6865402 -11.1339105 5.06704482 -4.92918617
## sd 0.87696277 1.0257226 0.9645486 0.9469142 1.01823607 0.87801873
##      [,73]      [,74]      [,75]      [,76]      [,77]      [,78]
## mean 0.1629300 -0.005236446 0.1073088 -0.1465294 0.09553372 -0.03913617
## sum 16.2929984 -0.523644574 10.7308791 -14.6529425 9.55337199 -3.91361747
## sd 0.8941483 1.008655635 0.9944261 1.0769782 0.96638316 1.01031030
##      [,79]      [,80]      [,81]      [,82]      [,83]      [,84]
## mean 0.05489841 -0.1765393 -0.0459872 -0.00540858 0.1283985 0.154827
## sum 5.48984112 -17.6539282 -4.5987198 -0.54085797 12.8398513 15.482699
## sd 1.05630156 1.0659721 1.0161120 1.07501409 0.9959257 0.969051
##      [,85]      [,86]      [,87]      [,88]      [,89]      [,90]
## mean 0.01586283 0.08128843 -0.04852987 0.02417753 0.1095373 -0.2148035
## sum 1.58628257 8.12884279 -4.85298714 2.41775337 10.9537307 -21.4803458
## sd 1.11207550 1.06293318 0.94756707 1.04856646 1.0506158 1.1474328
##      [,91]      [,92]      [,93]      [,94]      [,95]      [,96]
## mean 0.006974302 -0.06152943 -0.05830342 -0.09212069 0.05559785 -0.06251836
## sum 0.697430160 -6.15294266 -5.83034172 -9.21206910 5.55978494 -6.25183599
## sd 0.994143844 0.98719987 1.03118035 0.83965976 1.06511882 0.98403060
##      [,97]      [,98]      [,99]      [,100]
## mean -0.0103759 -0.09106338 -0.1305609 -0.03451664
## sum -1.0375903 -9.10633772 -13.0560897 -3.45166380
## sd 0.9922376 0.91410182 1.0321389 0.99787848
```

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；
```

```
# Using tapply
```

```
# Task 01
```

```
cylinder_mpg =
```

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

```
print(cylinder_mpg)
```

```
##           4           6           8
```

```
## 26.66364 19.74286 15.10000
```

```
# Task 02
```

```
cylinder_wt =
```

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

```
print(cylinder_wt)
```

```
##           4           6           8
```

```
## 2.285727 3.117143 3.999214
```

```
# using dplyr
```

```
library(dplyr)
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
## filter, lag
```

```
## The following objects are masked from 'package:base':  
##  
## intersect, setdiff, setequal, union
```

```
# Task 01  
cylinder_mpg_dplyr =  
  mtcars %>%  
  group_by(cyl) %>%  
  summarise(  
    mean_mpg =  
      mean(mpg))  
  
print(cylinder_mpg_dplyr)
```

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

```
# Task 02  
cylinder_wt_dplyr =  
  mtcars %>%  
  group_by(cyl) %>%  
  summarise(  
    mean_wt =  
      mean(wt))  
  
print(cylinder_wt_dplyr)
```

```
## # 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；
# Task 01
my_list =
  list(a = 1:10, b = letters[1:5], c = LETTERS[1:8])

# Using lapply
lengths_lapply =
  lapply(my_list, length)

# Using sapply
lengths_sapply =
  sapply(my_list, length)

# Print the result
print("Using lapply:")
```

```
## [1] "Using lapply:"
```

```
print(lengths_lapply)
```

```
## $a  
## [1] 10  
##  
## $b  
## [1] 5  
##  
## $c  
## [1] 8
```

```
print("Using sapply:")
```

```
## [1] "Using sapply:"
```

```
print(lengths_sapply)
```

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

```
# Task 02  
# Using lapply  
avg_by_column_lapply =  
  lapply(mtcars, mean)  
print("Using lapply:")
```

```
## [1] "Using lapply:"
```

```
print(avg_by_column_lapply)
```

```
## $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
```

```
# Using sapply
avg_by_column_sapply =
  sapply(mtcars, mean)
print("Using sapply:")
```



```
## [1] "Using sapply:"
```

```
print(avg_by_column_sapply)
```

```
##      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;
# Load the packages
library(purrr)
```

```
library(dplyr)

# Preparing the data
df = tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

# Calculating
results =
  map(df,
    ~c(mean = mean(.),
        sum = sum(.),
        median = median(.)))

# Bind the results
result_df =
  bind_rows(results)

# Print the results
print(result_df)
```

```
## # A tibble: 4 x 3
##   mean    sum median
##   <dbl> <dbl>   <dbl>
## 1  0.317  3.17  0.0291
## 2 -0.278 -2.78 -0.153
## 3  0.225  2.25 -0.0483
## 4  0.177  1.77 -0.0715
```

0.6.2 map 进阶

用 `map` 配合 `purrr` 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；
```

```
# Load the package
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
```

```
## v forcats 1.0.0      v stringr 1.5.0
```

```
## v ggplot2 3.4.3      v tibble 3.2.1
```

```
## v lubridate 1.9.2     v tidyr 1.3.0
```

```
## v readr 2.1.4
```

```
## -- Conflicts ----- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag() masks stats::lag()
```

```
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
```

```
# Calculating
```

```
result =
```

```
  mtcars %>%
```

```
  group_by(cyl) %>%
```

```
  nest() %>%
```

```
  mutate(
```

```
    correlation =
```

```
      map(data, ~cor.test(.$mpg, .$wt)),
```

```
    p_value =
```

```
      map_dbl(correlation, "p.value"),
```

```
    correlation_coefficient =
```

```
      map_dbl(correlation, "estimate"))
```

```
# Print the result
print(result$p_value)
```

```
## [1] 0.09175766 0.01374278 0.01179281
```

```
print(result$correlation_coefficient)
```

```
## [1] -0.6815498 -0.7131848 -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；
library(dplyr)
library(purrr)

# Task 01
iris_with_factors =
  iris %>% keep(is.factor)
head(iris_with_factors, 10)
```

```
##      Species
## 1   setosa
## 2   setosa
## 3   setosa
## 4   setosa
## 5   setosa
## 6   setosa
## 7   setosa
```

```
## 8   setosa
## 9   setosa
## 10  setosa
```

```
# Task 02
iris_without_factors =
  iris %>% discard(is.factor)
head(iris_without_factors, 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；
# Load the package
library(purrr)
```

```
# Create three vectors
vector1 =
  c(1, 3, 5, 6, 10)
vector2 =
  c(1, 2, 3, 7, 8, 10)
vector3 =
  c(1, 2, 3, 4, 8, 9, 10)

# Find the common elements
common_elements =
  reduce(
    list(
      vector1,
      vector2,
      vector3),
    intersect)

# Print the results
print(common_elements)
```

```
## [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；  
# Example  
library(purrr)  
library(tidyverse)  
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_join =  
  dfs %>% reduce(full_join);
```

```
## Joining with `by = join_by(name)`  
## Joining with `by = join_by(name)`
```

```
# Task  
# Load the library  
library(tidyverse)  
  
# Create data frame  
data_full =  
  data.frame(  
    `Name` = character(),  
    `Key` = character(),  
    `Value` = character()  
  )  
  
data_list =  
  list(  
    data.frame(Name = "John", Key = "Age", Value = "30"),  
    data.frame(Name = "John", Key = "Sex", Value = "M"),
```

```
data.frame(Name = "Mary", Key = "Sex", Value = "F"),
data.frame(Name = "Mary", Key = "Treatment", Value = "A")
)

data_full =
  do.call(rbind, data_list)

# Spread it
result_tidyverse =
  data_full %>%
  spread(key = `Key`, value = `Value`)
result_tidyverse$Age =
  as.numeric(result_tidyverse$Age)

print(result_tidyverse)
```

```
##   Name Age Sex Treatment
## 1 John  30  M      <NA>
## 2 Mary  NA  F         A
```

0.7 练习与作业 4: pmap 和 map 的更多用法

请参考 <https://r4ds.had.co.nz/iteration.html> 的 Mapping over multiple arguments 部分

0.7.1 map2

运行以下代码，查看输出结果。用 for 循环重现计算结果。

```
mu <- list(5, 10, -3);
sigma <- list(1, 5, 10);
map2(mu, sigma, rnorm, n = 5)
```



```
## 代码写这里，并运行；
```

```
# Example
```

```
mu <- list(5, 10, -3);
```

```
sigma <- list(1, 5, 10);
```

```
map2(mu, sigma, rnorm, n = 5)
```

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

```
## [1] 5.749278 4.463402 4.366117 6.065397 6.322708
```

```
##
```

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

```
## [1] 11.091503 3.712076 14.126324 6.264098 9.179697
```

```
##
```

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

```
## [1] 10.63365394 -19.43866101 4.45335944 -0.06955236 -0.29811515
```

```
# Task
```

```
mu = list(5, 10, -3)
```

```
sigma = list(1, 5, 10)
```

```
n = 5
```

```
# Calculating
```

```
results = vector("list", length(mu))
```

```
# Using 'for'
```

```
for (i in 1:length(mu)) {
```

```
  results[[i]] =
```

```
    rnorm(
```

```
      n,
```

```
      mean = mu[[i]],
```

```
      sd = sigma[[i]])
```

```
}
```

```
# Print the result
```

```
print(results)
```

```
## [[1]]
## [1] 5.262239 4.697671 5.152479 4.662372 5.247955
##
## [[2]]
## [1] 6.914691 11.571684 17.371821 5.284800 7.628093
##
## [[3]]
## [1] -8.632784 8.689311 -6.331444 6.852970 -3.547983
```

0.7.2 pmap

运行以下代码，查看输出结果。用 for 循环重现计算结果。

```
params <- tribble(
  ~mean, ~sd, ~n,
    5,    1,  1,
  10,    5,  3,
  -3,   10,  5
)
params %>%
  pmap(rnorm)
```

```
## 代码写这里，并运行；
# Example
params <- tribble(
  ~mean, ~sd, ~n,
    5,    1,  1,
  10,    5,  3,
  -3,   10,  5
)
```

```
params %>%  
  pmap(rnorm)
```

```
## [[1]]  
## [1] 4.357922  
##  
## [[2]]  
## [1] 14.166952  6.769464  7.615705  
##  
## [[3]]  
## [1] -7.742218  6.559735 18.120634 -16.470769 -11.629351
```

```
print(params)
```

```
## # A tibble: 3 x 3  
##   mean    sd    n  
##   <dbl> <dbl> <dbl>  
## 1     5     1     1  
## 2    10     5     3  
## 3    -3    10     5
```

```
# Task  
# Load the library  
library(dplyr)  
  
# Processing  
params = tribble(  
  ~mean, ~sd, ~n,  
    5,    1,  1,  
   10,    5,  3,  
   -3,   10,  5  
)
```

```
result_list = list()

for (i in 1:nrow(params)) {
  mean_val = params$mean[i]
  sd_val = params$sd[i]
  n_val = params$n[i]

  result_list[[i]] =
    c(mean_val, sd_val, n_val)
}

for (i in 1:nrow(params)) {
  print (
    rnorm(
      n = params$n[i],
      mean = params$mean[i],
      sd = params$sd[i]
    )
  )
}
```

```
## [1] 3.963296
## [1] 3.609022 10.916591 10.739960
## [1] -6.731752 -8.076998 3.297711 -23.407334 10.079462
```

```
result_df =
  as.data.frame(do.call(rbind, result_list))
colnames(result_df) =
  c("mean", "sd", "n")

print(result_df)
```

```
## mean sd n
```

```
## 1    5  1 1
## 2   10  5 3
## 3   -3 10 5
```

0.8 练习与作业 5: 并行计算

0.8.1 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
```

```
library(parallel);
library(foreach);
```

```
##
## Attaching package: 'foreach'

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

```
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.
## -31499.78 -2697.61   -16.55   -101.19   2410.16   32899.92
```

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；
cpus = parallel::detectCores()
cpus_used = cpus - 1

# Print the data
print(cpus)
```

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

```
print(cpus_used)
```

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