R 语言编程：基于 tidyverse

第 14 讲　数据处理神器: data.table 包

张敬信

2022 年 2 月 13 日

哈尔滨商业大学

一. data.table 包概述

- data.table 包是 data.frame 的高性能版本，不依赖其他包就能胜任各种数据操作，速度超快。
- data.table 的高性能来源于内存管理（引用语法）、并行化和大量精细优化，内存能-应付的数据，追求性能的话，data.table 是最优选择没有之一。
- data.table 语法高度抽象、简洁、一致：**用 i 选择行，用 j 操作列，根据 by 分组**

```
dt[i, j, by]
```

图 1: data.table 包最简语法

- `j` 表达式非常强大和灵活，可以**选择、修改、汇总、计算新列**，甚至可以接受任意表达式。
- 需要记住的最关键一点是：**只要返回等长元素或长度为 1 元素的** list，**每个** list **元素将转化为结果** `data.table` **的一列**。
- `data.table` 高度抽象的语法无疑增加了学习成本，底层用 `data.table`，上层用 `tidyverse` 语法包装的包，如 `dtplyr`，`tidyfst` 等，也是不错的选择。

二. 通用语法

1. 创建 data.table

```
library(data.table)
dt = data.table(
  x = 1:2,
  y = c("A", "B"))
dt
#>    x y
#> 1: 1 A
#> 2: 2 B
```

- as.data.table() 可将数据框、列表、矩阵等转化为 data.table; 若只想按引用转化，用 setDT().

2. 引用语法

- 高效计算的编程都支持引用语法（浅拷贝），相当于是对象只有一个在内存放着，不做多余复制，用两个指针都指向该同一对象，操作哪个指针，都是在修改该同一对象。
- `data.table` 使用`:=` 算符，做整列或部分列替换时都不做任何拷贝，因为`:=` 算符是通过引用就地（in-place）更新 `data.table` 的列。
- 若想要复制不想按引用（修改数据本身），使用 `DT2 = copy(DT1)`.

3. 键与索引

data.table 支持设置**键**和**索引**，使得选择行、做数据连接更加方便快速（快 170 倍）：- 键：一级有序索引 - 索引：自动二级索引

```
setkey(dt, v1, v3)          # 设置键
setindex(dt, v1, v3)        # 设置索引
```

二者的主要不同：

- 使用键时，数据在内存中做物理重排序；而使用索引时，顺序只是保存为属性
- 键是显式定义的；索引可以手动创建，也可以运行时创建
- 索引与参数 on 连用；键的使用是可选的，但为了可读性建议使用键

4. 特殊符号

data.table 提供了一些辅助操作的特殊符号:

- .(): 代替 list()
- :=: 按引用方式增加、修改列
- .N: 行数
- .SD: 每个分组的数据子集，除了 by 或 keyby 的列
- .SDcols: 与 .SD 连用，用来选择包含在.SD 中的列
- .BY: 包含所有 by 分组变量的 list
- .I: 整数向量 seq_len(nrow(x))，例如 DT[, .I[which.max(somecol)], by=grp]
- .GRP: 分组索引，1 代表第 1 分组，2 代表第 2 分组，...
- .NGRP: 分组数
- .EACHI: 用于 by/keyby = .EACHI 表示根据 i 表达式的每一行分组

5. 链式操作

data.table 也有自己专用的管道操作，称为链式操作：

```
DT[…][…][…]          # 或者写开为
DT[
   …
  ][
    …
    ][
      …
      ]
```

三. 数据读写

`fread()` 和 `fwrite()`:

- 速度超快、稳健
- 自动检测分隔符、列类型、行数
- 支持多种通用格式（除了 Excel），支持 URLs 甚至操作系统指令

1. 读入数据

```r
fread("DT.csv")
fread("DT.txt", sep = "\t")
# 选择部分行列读取
fread("DT.csv", select = c("V1", "V4"))
fread("DT.csv", drop = "V4", nrows = 100)
# 读取压缩文件
fread(cmd = "unzip -cq myfile.zip")
fread("myfile.gz")
# 批量读取
c("DT.csv", "DT.csv") %>%
  lapply(fread) %>%
  rbindlist()          # 多个数据框/列表按行合并
```

2. 写出数据

```
fwrite(DT, "DT.csv")
fwrite(DT, "DT.csv", append = TRUE)    # 追加内容
fwrite(DT, "DT.txt", sep = "\t")
# 支持写出列表列
fwrite(setDT(list(0, list(1:5))), "DT2.csv")
# 写出到压缩文件
fwrite(DT, "myfile.csv.gz", compress = "gzip")
```

四. 数据连接

1. **按行合并**
   - `rbind(DT1, DT2, ...)`: 按行堆叠多个 `data.table`
   - `rbindlist(DT_list, idcol)`: 堆叠多个 `data.table` 构成的 `list`

2. **六种连接**
   - **左连接**: 保留 `x` 所有行，合并匹配的 `y` 中的列

```
y[x, on = "v1"]          # 注意是以 x 为左表
y[x]                     # 若 v1 是键
merge(x, y, all.x = TRUE, by = "v1")
```

注: 若表 `x` 与 `y` 中匹配列的列名不同，可以用 `by.x = "c1", by.y = "c2"`, 多个匹配列，套 `c()`.

- **右连接**：保留 y 所有行，合并匹配的 x 中的列

```
merge(x, y, all.y = TRUE, by = "v1")
```

- **内连接**：只保留 x 中与 y 匹配的行，合并匹配的 y 中的列

```
merge(x, y, by = "v1")
```

- **全连接**：保留 x 和 y 中的所有行，合并匹配的列

```
merge(x, y, all = TRUE, by = "v1")
```

- **半连接**：根据在 y 中，来筛选 x 中的行

```
x[y$v1, on = "v1", nomatch = 0]
```

- **反连接**：根据不在 y 中，来筛选 x 中的行

```
x[!y, on = "v1"]
```

3. **集合运算**

```
fintersect(x, y)
fsetdiff(x, y)
funion(x, y)
fsetequal(x, y)
```

1. 宽变长

宽表的特点是：表比较宽，本来该是”值”的，却出现在”变量（名）“中。这就需要给它变到”值”中，新起个列名存为一列，即宽表变长表。

- 每一行只有 1 个观测的情形，对比 pivot_longer()

```
DT = fread("datas/分省年度 GDP.csv", encoding = "UTF-8")
DT
#>          地区 2019 年 2018 年 2017 年
#> 1:    北京市  35371  33106  28015
#> 2:    天津市  14104  13363  18549
#> 3:    河北省  35105  32495  34016
#> 4: 黑龙江省  13613  12846  15903
```

- 对照 pivot_longer 来看

```
DT %>%
  pivot_longer(-地区, names_to = " 年份", values_to = "GDP")
```

```
DT %>%
  melt(measure = 2:4, variable = " 年份", value = "GDP") %>%
  head(4)
#>         地区    年份    GDP
#> 1:    北京市 2019 年 35371
#> 2:    天津市 2019 年 14104
#> 3:    河北省 2019 年 35105
#> 4: 黑龙江省 2019 年 13613
```

注：都是指定要变形的列、为存放变形列的列名中的”值”指定新列名、为存放变形列中的”值”指定新列名。

- 每一行有多个观测的情形

```
load("datas/family.rda")
DT = as.data.table(family)    # family 数据
DT
#>    family dob_child1 dob_child2 gender_child1 gender_child
#> 1:      1 1998-11-26 2000-01-29             1
#> 2:      2 1996-06-22       <NA>             2            M
#> 3:      3 2002-07-11 2004-04-05             2
#> 4:      4 2004-10-10 2009-08-27             1
#> 5:      5 2000-12-05 2005-02-28             2
```

```
DT %>%
  melt(measure = patterns("^dob", "^gender"),
       value = c("dob", "gender"), na.rm = TRUE)
#>    family variable        dob gender
#> 1:      1        1 1998-11-26      1
#> 2:      2        1 1996-06-22      2
#> 3:      3        1 2002-07-11      2
#> 4:      4        1 2004-10-10      1
#> 5:      5        1 2000-12-05      2
#> 6:      1        2 2000-01-29      2
#> 7:      3        2 2004-04-05      2
#> 8:      4        2 2009-08-27      1
#> 9:      5        2 2005-02-28      1
```

2. 长变宽（与宽变长互逆）

长表的特点是：表比较长。

有时候需要将分类变量的若干水平值，变成变量（列名），这就是长表变宽表。

- 只有 1 个列名列和 1 个值列的情形

```
load("datas/animals.rda")
DT = as.data.table(animals)      # 农场动物数据
head(DT, 4)
#>       Type Year Heads
#> 1:   Sheep 2015 24943
#> 2: Cattle 1972  2189
#> 3:   Camel 1985   559
#> 4:   Camel 1995   368
```

- 对照 pivot_wider() 来看

```
DT %>%
  pivot_wider(names_from = Type, values_from = Heads,
              values_fill = 0)
```

```
DT %>%
  dcast(Year ~ Type, value = "Heads", fill = 0) %>%
  head(4)
#>    Year Camel Cattle Goat Horse Sheep
#> 1: 1970   633   2108 4207  2315 13311
#> 2: 1971   632   2176 4195  2270 13420
#> 3: 1972   625   2189 4338  2239 13716
#> 4: 1973   603   2235 4442  2185 14073
```

- dcast() 函数第 1 个参数是公式形式，~ 左边是不变的列，右边是”变量名”来源列，参数 value 指定”值”的来源列。

- 有多个列名列和多个值列的情形

```
us_rent_income %>%
  as.data.table() %>%
  dcast(GEOID + NAME ~ variable,
        value = c("estimate", "moe")) %>%
  head()
#>    GEOID       NAME income rent
#> 1:    01    Alabama    136    3
#> 2:    02     Alaska    508   13
#> 3:    04    Arizona    148    4
#> 4:    05   Arkansas    165    5
#> 5:    06 California    109    3
#> 6:    08   Colorado    109    5
```

3. 数据分割与合并

函数 split(DT, by) 可将 data.table 分割为 list, 然后就可以接 map_*() 函数逐分组迭代。

- 拆分列

```
DT = as.data.table(table3)
DT
#>         country year              rate
#> 1: Afghanistan 1999       745/19987071
#> 2: Afghanistan 2000      2666/20595360
#> 3:      Brazil 1999    37737/172006362
#> 4:      Brazil 2000    80488/174504898
#> 5:       China 1999 212258/1272915272
#> 6:       China 2000 213766/1280428583
```

- 将 case 列拆分为两列, 并删除原列

```
DT[, c("cases", "population") :=
     tstrsplit(DT$rate, split = "/")][, rate := NULL][]
#>         country year  cases population
#> 1: Afghanistan 1999    745   19987071
#> 2: Afghanistan 2000   2666   20595360
#> 3:      Brazil 1999  37737  172006362
#> 4:      Brazil 2000  80488  174504898
#> 5:       China 1999 212258 1272915272
#> 6:       China 2000 213766 1280428583
```

- 合并列

```
DT = as.data.table(table5)
DT
#>         country century year              rate
#> 1: Afghanistan      19   99       745/19987071
#> 2: Afghanistan      20   00      2666/20595360
#> 3:      Brazil      19   99    37737/172006362
#> 4:      Brazil      20   00    80488/174504898
#> 5:       China      19   99 212258/1272915272
#> 6:       China      20   00 213766/1280428583
```

- 将 century 和 year 列合并为新列 new, 并删除原列

```
DT[, new := paste0(century, year)][,
      c("century", "year") := NULL][]
#>         country                rate  new
#> 1: Afghanistan       745/19987071 1999
#> 2: Afghanistan     2666/20595360 2000
#> 3:      Brazil   37737/172006362 1999
#> 4:      Brazil   80488/174504898 2000
#> 5:       China 212258/1272915272 1999
#> 6:       China 213766/1280428583 2000
```

1. 选择行

用 i 表达式，选择行。

- 根据索引

```
dt[3:4,]                    # 或 dt[3:4]
dt[!3:7,]                   # 反选, 或 dt[-(3:7)]
```

- 根据逻辑表达式

```
dt[v2 > 5]
dt[v4 %chin% c("A","C")]  # 比 %in% 更快
dt[v1==1 & v4=="A"]
```

- 删除重复行

```
unique(dt)
unique(dt, by = c("v1","v4"))    # 返回所有列
```

- 删除包含 NA 的行

```
na.omit(dt, cols = 1:4)
```

- 行切片

```
dt[sample(.N, 3)]                    # 随机抽取 3 行
dt[sample(.N, .N * 0.5)]             # 随机抽取 50% 的行
# v1 值最大的行
dt[frankv(-v1, ties.method = "dense") < 2]
```

- 其他

```
dt[v4 %like% "^B"]                      # v4 值以 B 开头
dt[v2 %between% c(3,5)]                 # 闭区间
dt[between(v2, 3, 5, incbounds = FALSE)]  # 开区间
dt[v2 %inrange% list(-1:1, 1:3)]  # v2 值属于多个区间的某个
dt[inrange(v2, -1:1, 1:3, incbounds = TRUE)] # 同上
```

2. 排序行

```
dt[order(v1)]              # 默认按 v1 从小到大
dt[order(-v1)]             # 按 v1 从大到小
dt[order(v1, -v2)]         # 按 v1 从小到大, v2 从大到小
```

- 若按引用对行重排序:

```
setorder(DT, V1, -V2)
```

3. 操作列

用 j 表达式操作列。

- 选择一列或多列

```
# 根据索引
dt[[3]]                 # 或 dt[["v3"]], dt$v3, 返回向量
dt[, 3]                 # 或 dt[, "v3"], 返回 data.table
# 根据列名
dt[, .(v3)]             # 或 dt[, list(v3)]
dt[, .(v2,v3,v4)]
dt[, v2:v4]
dt[, !c("v2","v3")]     # 反选列
```

- 反引用列名：data.table 需要字符串函数、正则表达式构造出列名向量，再通过反引用选择相应的列

```
cols = c("v2", "v3")
dt[, ..cols]
dt[, !..cols]
cols = paste0("v", 1:3)           # v1, v2, ...
cols = union("v4", names(dt))     # v4 列提到第 1 列
cols = grep("v", names(dt))       # 列名中包含"v"
cols = grep("^(a)", names(dt))    # 列名以"a" 开头
cols = grep("b$", names(dt))      # 列名以"b" 结尾
cols = grep(".2", names(dt))      # 正则匹配".2" 的列
cols = grep("v1|X", names(dt))    # v1 或 x
dt[, ..cols]
```

- 调整列序

```
cols = rev(names(DT))                    # 或其他列序
setcolorder(DT, cols)
```

- 修改列名

```
setnames(DT, old, new)
```

- 修改因子水平

```
DT[, setattr(sex, "levels", c("M", "F"))]
```

data.table 修改列，是用列赋值符号:= （不执行复制），直接对原数据框修改。

- 修改或增加一列

```
dt[, v1 := v1 ^ 2][]          # 修改列，加 [] 输出结果
dt[, v2 := log(v1)]           # 增加新列
dt[, .(v2 = log(v1), v3 = v2 + 1)]  # 只保留新列
```

注意，代码 v3 = v2 + 1 中的 v2 是原始的 v2 列，而不是前面新计算的 v2 列；若想使用新计算的列，可以用

```
dt[, c("v2", "v3") := .(temp <- log(v1), v3 = temp + 1)]
```

- 增加多列

```
dt[, c("v6","v7") := .(sqrt(v1), "x")]  # 或者
dt[, ':='(v6 = sqrt(v1),
          v7 = "x")]                    # v7 列的值全为 x
```

- 同时修改多列：data.table 选择并操作多列是借助 lapply() 以及特殊符号：
  - .SD: 每个分组的数据子集，除了 by 或 keyby 的列
  - .SDcols: 与.SD 连用，用来选择包含在.SD 中的列，支持索引、列名、连选、反选、正则表达式、条件判断函数

```
# 应用函数到所有列
DT[, lapply(.SD, as.character)]
# 应用函数到满足条件的列
DT[, lapply(.SD, Rescale),  # Rescale() 为自定义的归一化函数
   .SDcols = is.numeric]
# 应用函数到指定列
DT =  as.data.table(iris)
DT[, .SD * 10, .SDcols = patterns("(Length)|(Width)")]
```

注意，上述同时修改多列的代码，都是只保留新列，若要保留所有列，需要准备新列名 cols, 再在 j 表达式中使用 (cols) := ...

- 删除列

```
dt[, v1 := NULL]
dt[, c("v2","v3") := NULL]
cols = c("v2","v3")
dt[, (cols) := NULL]    # 注意, 不是 dt[, cols := NULL]
```

- 重新编码

```
# 一分支
dt[v1 < 4, v1 := 0]
# 二分支
dt[, v1 := fifelse(v1 < 0, -v1, v1)]
# 多分支
dt[, v2 := fcase(v2 < 4, "low",
                 v2 < 7, "middle",
                 default = "high")]
```

- 前移/后移运算

```
# 1,2,3 -> NA,1,2
shift(x, n = 1, fill = NA, type = "lag")
# 1,2,3 -> 2,3,NA
shift(x, n = 1, fill = NA, type = "lead")
```

七. 分组汇总

用 by 表达式指定分组[1]。

```
DT = readxl::read_xlsx("datas/ExamDatas_NAs.xlsx") %>%
  as.data.table()
```

- 未分组汇总

```
DT[, .(math_avg = mean(math, na.rm = TRUE))]
#>    math_avg
#> 1:       68
```

[1] data.table 是根据 by 或 keyby 分组，区别是，keyby 会排序结果并创建键，使得更快地访问子集.

- 简单的分组汇总

```
DT[, .(n = .N,
       math_avg = mean(math,  na.rm = TRUE),
       math_med = median(math)),
   by = sex]
#>      sex  n math_avg math_med
#> 1:    女 25     70.8       NA
#> 2:    男 24     64.6       NA
#> 3: <NA>  1     85.0       85
```

- 可以直接在 by 中使用判断条件或表达式，特别是根据整合单位的日期时间[2]汇总：

```
date = as.IDate("2021-01-01") + 1:50
dt = data.table(date, a = 1:50)
head(dt)
#>          date a
#> 1: 2021-01-02 1
#> 2: 2021-01-03 2
#> 3: 2021-01-04 3
#> 4: 2021-01-05 4
#> 5: 2021-01-06 5
#> 6: 2021-01-07 6
```

[2] data.table 提供快速处理日期时间的 IDateTime 类.

```
dt[, mean(a), by = list(mon = month(date))]   # 按月平均
#>    mon   V1
#> 1:   1 15.5
#> 2:   2 40.5
```

- 对某些列做汇总

```
DT[, lapply(.SD, mean), .SDcols = patterns("h"),
   by = .(class, sex)] %>%      # 或用 by = c("class", "sex")
  head(5)
#>    class sex chinese math english
#> 1: 六 1 班   女    80.7   NA    67.4
#> 2: 六 1 班   男      NA 79.7    64.7
#> 3: 六 2 班   女    92.2 73.8    63.8
#> 4:  <NA>   男    90.0 86.0    72.0
#> 5: 六 2 班   男    75.4   NA    42.6
```

- 对所有列做汇总

```
DT[, name := NULL][, lapply(.SD, mean, na.rm = TRUE),
                   by = .(class, sex)] %>%
  head(5)
#>    class sex chinese math english moral science
#> 1: 六 1 班   女    80.7 77.2    67.4  8.33    9.57
#> 2: 六 1 班   男    57.0 79.7    64.7  8.67    9.33
#> 3: 六 2 班   女    92.2 73.8    63.8  8.33    9.00
#> 4:  <NA>   男    90.0 86.0    72.0  9.00   10.00
#> 5: 六 2 班   男    75.4 68.8    42.6  8.80    9.25
```

- 对满足条件的列做汇总

```
DT[, lapply(.SD, mean, na.rm = TRUE), by = class,
   .SDcols = is.numeric]
#>    class chinese math english moral science
#> 1: 六 1 班    77.8 78.0    66.6  8.44    9.50
#> 2: 六 2 班    82.9 71.2    52.0  8.62    9.12
#> 3:  <NA>    90.0 86.0    72.0  9.00   10.00
#> 4: 六 3 班    67.3 38.8    67.7  5.33    6.11
#> 5: 六 4 班    85.0 77.1    58.3  8.67    8.20
#> 6: 六 5 班    72.6 72.0    62.3  8.56    8.44
```

- 分组计数

```
DT = na.omit(DT)
DT[, .N, by = .(class, cut(math, c(0, 60, 100)))] %>%
  print(topn = 2)
#>      class       cut N
#>  1: 六 1 班 (60,100] 5
#>  2: 六 1 班   (0,60] 1
#> ---
#>  8: 六 5 班 (60,100] 5
#>  9: 六 5 班   (0,60] 1
```

- 上述分组计数会忽略频数为 0 的分组，若要显示出来：

```
DT[, Bin := cut(math, c(0, 60, 100))]
DT[CJ(class = class, Bin = Bin, unique = TRUE),
   on = c("class","Bin"), .N, by = .EACHI] %>%
  head(5)
#>    class      Bin N
#> 1: 六 1 班   (0,60] 1
#> 2: 六 1 班 (60,100] 5
#> 3: 六 2 班   (0,60] 2
#> 4: 六 2 班 (60,100] 4
#> 5: 六 3 班   (0,60] 5
```

注：函数 CJ() 相当于 expand_grid(), 生成所有两两组合（笛卡儿积）。

- 分组选择行：data.table 提供了辅助函数：first(), last(), uniqueN() 等

```
DT[, first(.SD), by = class]
DT[, .SD[3], by = class]                    # 每组第 3 个观测
DT[, tail(.SD, 2), by = class]              # 每组后 2 个观测
# 选择每个班男生数学最高分的观测
DT[sex == " 男", .SD[math == max(math)], by = class]
```

最后注

- 本节是分别按 `i`，`j`，`by` 讲解语法，真正使用的时候，是三者组合起来使用，即同时**对 `i` 选择的行，根据 `by` 分组，做 `j` 操作**；
- `data.table` 中习惯用 `lapply()`, 换成 `map()` 也是一样的，好处是支持函数的 `purrr`-风格公式。

本篇主要参阅 (张敬信, 2022), `A data.table and dplyr tour`, 以及 `data.table` 包文档，模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

张敬信 (2022). *R* 语言编程：基于 *tidyverse*. 人民邮电出版社, 北京.

谢益辉 (2021). *rmarkdown: Dynamic Documents for R*.

黄湘云 (2021). *Github: R-Markdown-Template*.