

data.table

参考文献

- 原文:

<https://github.com/Rdatatable/data.table/wiki/Getting-started>

- 中文翻译:

<http://youngspring1.github.io/post/2016/2016-03-13-datatable1/>

本章内容

- 基础：基本语法、聚合
- 语义引用
- 主键、基于二分法搜索的**subset**
- 二次索引和自动索引
- 数据拆分和合并

data.table介绍

data.table

- R语言data.table包是自带包data.frame的增强版本，用于数据框格式数据的处理，最大的特点是快，包括两个方面：一方面是编码快，代码简洁，只要一行命令就可以完成诸多任务；另一方面是处理快，内部处理的步骤进行了程序上的优化，大大加快数据运行速度。因此，在对大数据处理上，使用data.table无疑具有极高的效率。

data.table介绍（续）

- data.table offers fast and memory efficient: file reader and writer, aggregations, updates, equi, non-equi, rolling, range and interval joins, in a short and flexible syntax, for faster development.

术语约定

- 在本章中：
 - subset特指对行的选择
 - select特指对列的选择

基本语法

使用data.table分析数据

- 数据操作，比如subset、group、update、join等，它们本质上都是相关的。
- data.table集成了这些相关联的操作的好处：
 - 可以通过简洁一致的语法来实现想要的操作。
 - 不必为每种数据操作记忆不同的函数，减少大脑负担，从而能够流畅地进行分析。
 - 自动优化数据操作：通过精确知道每步操作所需要的数据，自动优化数据操作，从而提高运行速度和内存效率。

数据集

- 使用NYC-flights14的数据。它包含了2014年1月到10月纽约机场发出的所有航班信息。
- 下载地址：
<https://github.com/arunsrinivasan/flights/wiki/NYC-Flights-2014-data>
- flights14.csv共253316条记录。

读取数据集

- 可以使用data.table的fread()函数，快速直接读取航班数据：

```
setwd("C:\\Users\\lenovo\\Documents\\软件学院\\大数据班\\R语言基础课件") #改变工作目录到csv文件所在目录
```

```
library(data.table)
```

```
flights <- fread("flights14.csv")
```

```
dim(flights) #查看flights的维数（行数和列数）
```

```
[1] 253316      17
```

创建data.table (1/2)

- 和data.frame相似，可以通过data.table()创建一个data.table。
- 例：

```
DT = data.table(ID = c("b","b","b","a","a","c"), A  
= 1:6, B = 7:12, C=13:18)
```

```
DT
```

	ID	A	B	C
1:	b	1	7	13
2:	b	2	8	14
3:	b	3	9	15
4:	a	4	10	16
5:	a	5	11	17
6:	c	6	12	18

行号后面有冒号

```
class(DT$ID) #class()用来查看对象类型
```

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

但如果是data.frame会把字符串自动转换为“factor”类型。（见第2章“因子”）

创建data.table (2/2)

- 可以通过`as.data.table()`将已经存在的对象转化成data.table。
- 例：将data.frame A转化为data.table B
`B<-as.data.table(A)`

data.table和data.frame的区别

- 不同于data.frames，字符型的列，不会被自动转化成因子。
- 行号后面有个冒号，用于隔开第一列的内容。
- 如果数据的条目超过了全局选项 `datatable.print.nrows` 所定义的数值（默认是100条），那么只会输出数据最开头和最末尾的5行。
- `data.table`不能设置行的名称。稍后讲原因。

一般形式（data.table增强的方式）

- 不同data.frame，我们能做的可不仅仅局限于subset行或者select列。
- data.table的语法：

DT[i, j, by]

含义：Take DT, **subset** rows using i, then **calculate** j, **grouped by** by.

##	R:	i		j		by
##	SQL:	where		select update		group by

↑
SQL语句

subset行 (1/2)

L用于显示地指明这个是整数。有时可以提高计算速度，减少内容开销。默认"numeric"是uses 8 字节；integer是4字节。

- 例：获取六月份所有从“JFK”机场起飞的航班

```
ans <- flights[origin == "JFK" & month == 6L]
```

- 说明：

- 列可以像变量一样被引用。因此，我们不需要加上 **flights\$** 前缀，比如 `flights$dest` 和 `flights$month`，而是直接简单地引用 `dest` 和 `month` 这两列。
- 满足 `origin == "JFK" & month == 6L` 这两个条件的行会被抽出来。既然我们没有指定其他的条件，一个包含原数据里面所有列的 `data.table` 会被返回。
- *语法里面 `[i,j,k]` 的逗号不是必须的，当然如果指定了逗号，比如 `flights[dest == "JFK" & month == 6L,]` 也是没问题的。但在 `data.frame` 里面，逗号却是必须的。

subset行 (2/2)

- 获取 flights 开头的两行
ans <- flights[1:2]

排序

- 例：先按 **origin** 列 的升序，再按 **dest** 的降序排列。通过 **R** 语言的基础函数 **order()** 来完成这个功能。

```
ans <- flights[order(origin, -dest)]
```



- **order()** 函数是被优化过的
 - 我们可以对字符型的列使用减号 “-” 来实现降序排列。
 - 函数 **order()** 其实调用了 **data.table** 的快速基数排序函数 **forder()**，它比 **base::order** 快很多。
- 对 **data.table** 还是使用我们熟悉的函数，但可以显著地提高分析效率。

select列 (1/4)

- 例：选取 `arr_delay` 列，返回值是向量。

```
ans <- flights[, arr_delay]
```

```
head(ans)
```

```
[1] 13 13 9 -26 1 0
```



- 说明：
 - 既然列可以作为变量被引用，我们可以直接引用我们想选取的列。
 - 既然我们想选取所有的行，我们毋需指定参数 `i`。
 - 返回了所有行的 `arr_delay` 列。

select列 (2/4)

- 例：选取 `arr_delay` 列，返回值是 **data.table**

```
ans <- flights[, list(arr_delay)]
```

```
head(ans)
  arr_delay
1:      13
2:      13
3:       9
4:     -26
5:       1
6:       0
```

- 说明

- 我们用 **list()** 把列名 **arr_delay** 包围起来，它可以确保返回值是 **data.table**。正如前面一个例子，如果不这样做，返回值就是一个向量。
- **data.table** 也允许用 **.()** 来包围列名，它是 **list()** 的别名，它们的效果是同样的。即 `ans <- flights[, .(arr_delay)]`。

select列 (3/4)

- 例：选取 `arr_delay` 列和 `dep_delay` 列

```
ans <- flights[, .(arr_delay, dep_delay)]
```

```
head(ans)
```

 **list()** 的别名

	arr_delay	dep_delay
1:	13	14
2:	13	-3
3:	9	2
4:	-26	-8
5:	1	2
6:	0	4

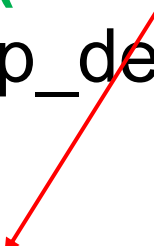
- 说明：
 - 只要参数 `j` 返回一个 `list`，这个 `list` 的每一个元素都会被转换成结果 `data.table` 的一列。

select列 (4/4)

- 选取 `arr_delay` 列和 `dep_delay` 列，并把列名改为 `delay_arr` 和 `delay_dep`。

```
ans <- flights[, .(delay_arr = arr_delay,  
delay_dep = dep_delay)]
```

```
head(ans)
```



	delay_arr	delay_dep
1:	13	14
2:	13	-3
3:	9	2
4:	-26	-8
5:	1	2
6:	0	4

在参数j里运算

- 例：有多少航班完全没有延误？

```
ans <- flights[, sum((arr_delay + dep_delay)<0)]
```

```
ans
```

```
[1] 141814
```

- 说明：j参数除了能select列之外，还可以处理表达式，即**对列进行计算**。当我们引用列时，列本质上就是变量，因此我们可以对这些变量使用函数进行计算。
- 另外，`flights[, (arr_delay + dep_delay)<0]` 的结果是253316个布尔值。

在参数i选取，在参数j运算（1/2）

- 例：在六月份，从” JFK” 机场起飞的航班中，计算起飞和到达的平均延误时间。

```
ans <- flights[origin == "JFK" & month ==  
6L, .(m_arr=mean(arr_delay), m_dep=mean(dep_delay))]
```

```
ans  
      m_arr      m_dep  
1: 5.839349 9.807884
```

说明

- 我们首先在i参数里，找到所有符合 origin (机场)是"JFK"，并且 month (月份)是 6 这样条件的行。此时，我们还没有subset整个data.table。
- 然后，对于参数j，它只使用了两列。我们需要分别计算这两列的平均值 mean()。这时，我们才subset那些符合i参数里条件的列，然后计算它们的平均值。
- 因为这三个参数（i，j和by）都被指定在同一个方括号中，data.table能同时接受这三个参数，并在计算之前，选取最优的计算方法，而不是分步骤计算。所以，我们可以避免对整个data.table计算，同时，在计算速度和内存使用量这两方面，取得最优的效果。

在参数i选取，在参数j运算（2/2）

- 例：在六月份，从” JFK” 机场起飞的航班一共有多少？
ans <- flights[origin == "JFK" & month == 6L, length(dest)]
 - 函数 length() 需要一个参数。我们只需要知道，结果里有多少行数据。我们可以使用任何一列作为函数 length() 的参数。
- 特别的符号 **.N**
 - .N 是一个内建的变量，**它表示当前的分组中，对象的数目**。在下一节，当它**和 by 一起使用**的时候，我们会发现它特别有用。**还没有涉及到分组的时候，它只是简单地返回行的数目。**
- 上例可改写为
ans <- flights[origin == "JFK" & month == 6L, **.N**]

像data.frame一样用列名引用

- 例：用data.frame的方式，选取 arr_delay 和 dep_delay 两列。
注：不写with参数也可！

```
ans <- flights[, c("arr_delay", "dep_delay"), with=FALSE]
```

– 参数 with是根据 R里面的函数 with() 演变而来的。在 data.table里，我们设置 with=FALSE，使得我们不能再像变量一样引用列了。

– 等价于：flights[, .(arr_delay, dep_delay)]

- 我们还可以使用 - 或 ! 来排除列。

```
ans <- flights[, !c("arr_delay", "dep_delay"), with=FALSE]
```

#or

```
ans <- flights[, -c("arr_delay", "dep_delay"), with=FALSE]
```

聚合

用by分组（1/4）

- 例：如何获取每个机场起飞的航班数？

```
ans <- flights[, .(N), by=.(origin)]
```

- 说明：

- **.N** 表示当前的分组中，对象的数目。**先按照 origin 列分组，再用 .N 获取每组的数目。**
- 原始数据里，机场是按照“JFK”，“LGA”然后“EWR”的顺序排列的。被分组的那一列变量的顺序，也体现在结果里面。
- 既然我们没有在参数j里面指定列名，那这一列就自然是 **N** 了。
- **by**也接受一个包含列名的字符向量作为参数。这在写代码的时候特别有用，比如设计一个函数，它的参数是要被分组的列。
- 当参数j和by里面**只有一列**，我们可以**省略 .()**。刚刚的任务我们可以这样做：

```
ans <- flights[, .N, by=origin]
```

用by分组（2/4）

- 例： 如何获取美航（**carrier code**代码是“AA”）在每个机场起飞的航班数？

```
ans <- flights[carrier == "AA", .N, by=origin]
```

- 说明：
 - 我们首先通过参数*i*，指定表达式 **carrier == "AA"**，选取符合条件的行。
 - 对于这些行，我们再按 **origin** 分组，获取每组的数目。再次声明，实际上没有列被重新创建，因为参数*j*表达式不需要获取列，因此在计算速度和内存使用量这两方面，取得最优的效果。

用by分组 (3/4)

- 例：如何获取美航在所有机场的起 / 降的数目？

```
ans <- flights[carrier == "AA", .N, by=.(origin,dest)]
```



	origin	dest	N
1:	JFK	LAX	3387
2:	LGA	PBI	245
3:	EWB	LAX	62
4:	JFK	MIA	1876
5:	JFK	SEA	298
6:	EWB	MIA	848
7:	JFK	SFO	1312

- 参数by 可以接受多个列。

用by分组（4/4）

- 例：如何获取美航每个月在所有机场的起 / 降的平均延误时间？

```
ans <- flights[carrier == "AA", .(mean(arr_delay),  
mean(dep_delay)), by = .(origin, dest, month)]
```

	origin	dest	month	V1	V2
1:	JFK	LAX	1	6.590361	14.2289157
2:	LGA	PBI	1	-7.758621	0.3103448
3:	EWR	LAX	1	1.366667	7.5000000
4:	JFK	MIA	1	15.720670	18.7430168
5:	JFK	SEA	1	14.357143	30.7500000

- 说明：
 - 没有在参数j表达式中指定列名，它们会自动命名为(V1, V2)。
 - 再次声明，原数据里面的顺序，会反映在结果中。

参数keyby

- `data.table`本身就被设计成能保持原数据的顺序。但有时我们希望根据分组的变量排序。
- `ans <- flights[carrier == "AA", .(mean(arr_delay), mean(dep_delay)), keyby=(origin, dest, month)]`
- 说明：
 - 我们做的，**只是把 by 改为了 keyby**。这会自动的将结果按照升序排列。注意 `keyby()` 是在数据操作完成后才进行。
 - 实际上 `keyby` 做的不只是排序。它在排序之后，设置一个叫做**sorted**的属性。稍后会学习更多关于 **keys** 的内容。

chaining表达式

- 例：美航在所有机场的起 / 降的数目，并让结果按origin的升序、按dest的降序排列。
- 按照之前知识，可以这样用两条语句做：
ans <- flights[carrier == "AA", .N, by = .(origin, dest)]
ans <- ans[order(origin, -dest)]
- 但是这么做会生成一个临时变量。可以通过添加**chaining表达式**，避免生成临时变量。

```
ans <- flights[carrier == "AA", .N, by=.(origin, dest)][order(origin, -dest)]
```

- 我们可以一个接一个地添加表达式，做一系列操作，就像这样：**DT[...][...][...]**。也可以换行写：**DT[...
][...
][...
]**

by表达式

- 例：有多少航班起飞延误并且到达延误？有多少航班起飞延误和到达没延误.....
- `ans <- flights[, .N, .(dep_delay>0, arr_delay>0)]`

	dep_delay	arr_delay	N
1:	TRUE	TRUE	72836
2:	FALSE	TRUE	34583
3:	FALSE	FALSE	119304
4:	TRUE	FALSE	26593

← 26593个航班起飞延误但
却提前/准时到达了。

- 说明
 - 注意，我们没有在by表达式里面指定任何列名。然而结果里面，列名还是自动的生成了。
 - * 我们可以在表达式里面指定其他的列，比如：
`ans <- flights[, .N, .(origin, dep_delay>0, arr_delay>0)]`

在参数j里面指定多个列（1/2）

- 例：如何为每一列（要是100列呢）计算mean()？
- 特殊的语法.**SD**:
 - **.SD**是Subset of Data的缩写。它自身就**是一个data.table**，**包含通过by 分组后的每一组**。

```
DT = data.table(ID = c("b","b","b","a","a","c"), A = 1:6, B = 7:12, C=13:18)
```

```
DT[, lapply(.SD, mean), by=ID]
```

	ID	A	B	C
1:	b	1	7	13
2:	b	2	8	14
3:	b	3	9	15
4:	a	4	10	16
5:	a	5	11	17
6:	c	6	12	18



	ID	A	B	C
1:	b	2.0	8.0	14.0
2:	a	4.5	10.5	16.5
3:	c	6.0	12.0	18.0

lapply对每列计算平均值。

lapply函数和**sapply**函数相似，前者返回列表，后者返回向量。

• 说明

- **.SD** 分别包含了ID是 a、b、c的所有行，它们分别对应了各自的组。我们应用函数 **lapply()** 对每列计算平均值。
- 每一组返回包含三个平均数的list，这些构成了最终返回的data.table。
- 既然函数 **lapply()** 返回 list，我们就**不需要在外面多加 .()** 了。

在参数j里面指定多个列（2/2）

- 例：获取 `arr_delay` 和 `dep_delay` 这两列的平均值，并且按照 `origin`, `dest` 和 `month` 来分组。
 - 参数 **.SDcols**
 - `.SDcols = c("arr_delay", "dep_delay")` 设置 **.SD** 只包含 `arr_delay` 和 `dep_delay` 这两列。`.SDcols=3:6` 设置 **.SD** 包含3到6列。可以使用 `-` 或者 `!` 来移除列，`!(colA:colB)` 或 `-(colA:colB)` 表示移除从 `colA` 到 `colB` 的所有列。
- ```
flights[carrier == "AA", lapply(.SD, mean), by=.(origin, dest, month), .SDcols=c("arr_delay", "dep_delay")]
```

|    | origin | dest | month | arr_delay | dep_delay  |
|----|--------|------|-------|-----------|------------|
| 1: | JFK    | LAX  | 1     | 6.590361  | 14.2289157 |
| 2: | LGA    | PBI  | 1     | -7.758621 | 0.3103448  |
| 3: | EWB    | LAX  | 1     | 1.366667  | 7.5000000  |
| 4: | JFK    | MIA  | 1     | 15.720670 | 18.7430168 |
| 5: | JFK    | SEA  | 1     | 14.357143 | 30.7500000 |

# 对每组subset .SD

- 返回每个月的前两行
- `ans <- flights[, head(.SD, 2), by=month]`

|    | month | year | day | dep_time | dep_delay | arr_time | arr_delay | cancelled | carrier | tailnum | flight | origin | dest | air_time | distance | hour | min |
|----|-------|------|-----|----------|-----------|----------|-----------|-----------|---------|---------|--------|--------|------|----------|----------|------|-----|
| 1: | 1     | 2014 | 1   | 914      | 14        | 1238     | 13        | 0         | AA      | N338AA  | 1      | JFK    | LAX  | 359      | 2475     | 9    | 14  |
| 2: | 1     | 2014 | 1   | 1157     | -3        | 1523     | 13        | 0         | AA      | N335AA  | 3      | JFK    | LAX  | 363      | 2475     | 11   | 57  |
| 3: | 2     | 2014 | 1   | 859      | -1        | 1226     | 1         | 0         | AA      | N783AA  | 1      | JFK    | LAX  | 358      | 2475     | 8    | 59  |
| 4: | 2     | 2014 | 1   | 1155     | -5        | 1528     | 3         | 0         | AA      | N784AA  | 3      | JFK    | LAX  | 358      | 2475     | 11   | 55  |
| 5: | 3     | 2014 | 1   | 849      | -11       | 1306     | 36        | 0         | AA      | N784AA  | 1      | JFK    | LAX  | 375      | 2475     | 8    | 49  |
| 6: | 3     | 2014 | 1   | 1157     | -3        | 1529     | 14        | 0         | AA      | N787AA  | 3      | JFK    | LAX  | 368      | 2475     | 11   | 57  |
| 7: | 4     | 2014 | 1   | 1912     | 0         | 1927     | 15        | 0         | MO      | N514MO  | 2405   | LGA    | BNA  | 112      | 764      | 19   | 17  |

- 说明:
  - .SD 包含了每组的所有行。我们可以简单的subset各组数据的前两行。
  - 对每组数据，`head(.SD, 2)`返回的data.table同时也是个list。所以不需要用 `.()` 包围起来。

# data.table语法总结（1/4）

- data.table的语法形式是：

DT[i, j, by]

- 参数i:
  - 用来subset行，不需要使用 DT\$，而是将列当做变量来使用。
  - 我们可以使用order()排序。为了得到更快速的效果，其实是使用了data.table里面的快速排序。

# data.table语法总结 (2/4)

- 参数j:
  - 以data.table的形式选取列: `DT[, .(colA, colB)]`。
  - 以data.frame的形式选取列: `DT[, c("colA", "colB"), with=FALSE]`。
  - 按列进行计算: `DT[, .(sum(colA), mean(colB))]`。
  - 如果需要对列重命名: `DT[, .(sA = sum(colA), mB = mean(colB))]`。
  - 和i共同使用: `DT[colA > value, sum(colB)]`。

# data.table语法总结（3/4）

- 参数by:
  - 通过by，我们可以指定列，或者列名，甚至表达式，进行分组。结合by和i，参数j可以十分灵活，从而实现强大的功能。
  - by可以指定多个列，也可以指定表达式。
  - 我们可以用 keyby，对分组的结果自动排序。



# data.table语法总结（4/4）

- 我们可以在参数j中指定 **.SD** 和 **.SDcols**，对复数的列进行操作。例如：
  - 把函数**fun** 应用到所有 **.SDcols**指定的列上，同时对参数**by**指定的列进行分组：  
**DT[, **lapply**(**.SD**, fun), by=., **.SDcols**=...]**。
  - 返回每组前两行：**DT[, head(.SD, 2), by=.]**。
  - 三个参数联合使用：**DT[col > val, head(.SD, 1), by=.]**。