语义引用
（Reference Semantics）

重新读取数据集

```
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")
```

data.table的语义引用

- data.table的语义引用，它允许通过**引用（reference）**来**add/update/delete列**。
- 本节内容
  - 简要讨论“语义引用”，然后比较操作符“:=”的两种不同的形式。
  - 学习如何在参数j里面使用操作符“:=”来add/update/delete列，如何与参数i和by相结合。
  - 了解操作符“:=”的副作用，并学习如何用copy() 来避免这些副作用。

影子拷贝 vs 深度拷贝

影子拷贝，只是一份指向列的指针向量的拷贝，它会随着data.frame或者data.table的变化而变化。但在内存里，数据不是真的被复制了。深度拷贝，正相反，它会复制整个数据，并且保存在内存里。

背景

- DF是一个data.frame（与之前的DT数据一样）。

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

- 当我们执行下面的命令：

  DF$C <- 18:13 # (1) -- replace entire column

  DF$C[DF$ID == "b"] <- 15:13 # (2) -- subassign in column 'C'

- 说明：
  - 在R语言V3.1之前的版本里，上面这两种方法都会导致对整个data.frame的深度拷贝。而且还会拷贝多次。为了提高效率避免冗余操作，**data.tabel使用了操作符“:=”**。R里面本来就有定义了这个操作符，但却没有使用。
  - 在R语言V3.1之后的版本里，方法(1)只做影子拷贝，处理性能有了很大提升。然而，方法(2)还是会做深度拷贝。这就意味着，对于同样的查询语句，想要选取的列越多，需要做的深度拷贝就越多。

操作符“:=”

- 在参数j中，操作符“:=”有两种使用方法：

a.左右等式的形式

```
DT[, c("colA", "colB", ...) := list(valA, valB, ...)]
DT[, colA := valA] #只有一列时
```

b.函数形式

反引号

```
DT[, `:=`(colA = valA,    # valA is assigned to colA
          colB = valB,          # valB is assigned to colB
          ...
)]
```

- 注意：我们没有把运算的结果赋值给一个变量。因为完全没必要。因为我们直接更新了data.table。

添加列

- 例：对每次航班，添加 speed (km/hr) 和 total delay (minutes) 两列。
- 写法一：

```
flights[, `:=`(speed = distance / (air_time/60),
delay = arr_delay + dep_delay)]
```

- 写法二：

```
flights[, c("speed", "delay") := list(distance/(air_time/60),
arr_delay + dep_delay)]
```

更新列（sub-assign）

- 查看一下fligths 里的 hour列

  flights[, sort(**unique**(hour))]

  #unique用来获得一个vector，里面hour所有可能取值。

```
[1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

- hour列有25种不同的取值。但是0点和24点应该是一样的，我们来把24点全部替换成0点

  flights[hour == 24L, hour **:=** 0L] #和参数i一起使用

- 操作符“:=”没有返回值。需要查看运行结果时可以在查询语句的最后加一对方括号[]。例如 flights[hour == 24L, hour := 0L][]

删除列

- 例：删除 delay列
- 方法一：

```
flights[, c("delay") := NULL]
```

- 方法二：

```
flights[, `:=`(delay = NULL)]
```

- 说明：
  - 将一列赋值为 NULL，就会删除那一列。删除立即生效。
  - 使用左右等式的形式的时候，除了指定列名，也可以指定列号，但不要这么做。指定列名是好的编码习惯。
  - 为了方便，如果只需要删除一列，可以去掉 c("")，只指定列名，像这样：flights[, delay := NULL]

“:=”和分组

- 例：追加一列，用来保存某对起飞 / 到达机场间的最快飞行速度。

```
flights[, max_speed := max(speed), by=.(origin, dest)]
```

“:=”和复数列

- 例：再追加两列，用于保存每个月的最大起飞延误时间和到达延误时间。

```
in_cols  = c("dep_delay", "arr_delay")
out_cols = c("max_dep_delay", "max_arr_delay")
flights[, c(out_cols) := lapply(.SD, max), by =
month, .SDcols = in_cols]
```

- 为了更好的可读性，这里使用了左右等式的形式。并事先保存了输入和输出的列名到变量中。

C()不可省略，否则语法上就变成单个对象赋值了。

将后添加的列删除

- 在进行后面的学习前，先删除刚刚追加的几列：speed, max_speed, max_dep_delay 和 max_arr_delay。

```
flights[, c("speed", "max_speed",
"max_dep_delay", "max_arr_delay") := NULL]
```

“:=”和copy()

- 操作符“**:=**”会更新原数据。当我们不想更新原数据时，可以用函数 copy()。

“:=”的副作用

- 例：想创建一个函数，用于返回每个月的最快速度。

```
foo <- function(DT) {
  DT[, speed := distance / (air_time/60)]
  DT[, .(max_speed = max(speed)), by=month]
}
ans = foo(flights)
```

- 此时flights被增加了speed列，但没有max_speed列。

copy()

- 希望使用操作符“:=”的功能，但是不想改变原数据，可以用函数 copy() 来做到这一点。
- 函数 copy() 对输入参数进行深度拷贝，因此对副本做的所有更新操作，都不会对原数据生效。

```
flights[, speed := NULL]
foo <- function(DT) {
  DT <- copy(DT) # deep copy
  DT[, speed := distance / (air_time/60)]  #不会修改flights
  DT[, .(max_speed = max(speed)), by=month]
}
ans <- foo(flights)
```