二次索引和自动索引

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")
```

本节内容

- 学习二级索引。
- 继续学习快速subset，但这次我们使用新的参数on，它能自动设置二级索引。
- 学习自动索引。自动索引能自动创建二级索引，能够使用优化后R的原生语法来做subset。

二级索引

什么是二级索引

- 二级索引和data.table的主键类似，但有以下两点不同：
  - 它不会在内存里将整个data.table重新排序。它只会计算某列的顺序，将这个顺序向量保存在一个额外的叫做index的属性里面。
  - 一个data.table可以有多个二级索引。

设置二级索引

- 将origin列设置为该data.table的二级索引

  setindex(flights, **origin**)

  head(flights)

- 说明：
  - 函数setindex 和 setindexv()可以对data.table添加二级索引。
  - 注意flights实际上没有按照origin列的升序重新排列。但setkey()会重新排序！
  - setindex(flights, NULL)会删除所有的二级索引。

获取二级索引

```
indices(flights)
# [1] "origin"
setindex(flights, dest)
indices(flights)
# [1] "origin"      "dest"
```

- 说明：
  - 函数indices()返回一个data.table所有的二级索引。如果该data.table没有二级索引，那么返回NULL。
  - 注意我们对 origin列,dest列创建了另一个二级索引的时候，我们**不会丢掉之前创建的第一个二级索引**。也就是说，我们可以创建多个二级索引。

为什么使用二级索引（1/2）

- 对一个data.table重新排序成本太高。比如，用主键origin列来subset所有“JFK”。需要：

  setkey(flights, origin)

  flights[.("JFK")]
- 这时：setkey()需要：
  - a.计算得出origin列的排序向量，并且
  - b.基于刚刚的排序向量，对整个data.table重新排序
- 排序并不算耗时，而**重新排序非常耗时**。
- 除非我们需要对某一列重复地进行subset，否则二分法快速subset的高效可能被重新排序抵消。

为什么使用二级索引（2/2）

- 问题：
  - 为添加 / 更新列而对整个data.table重新排序并不理想。
  - 一最多只能有一个主键。
- 二级索引可以被重用
  - 既然一个data.table中可以有多个二级索引，并且创建一个二级索引就和将一个排序向量保存为属性一样简单，那么创建二级索引后，我们可以省下重新排序的时间。
  - 参数on使得语法更简洁，并且能自动创建并重用二级索引。

使用参数on和索引进行快速subset

参数on

- 通过在运行时计算二级索引来进行subset。免去每次使用setindex()来指定二级索引。
- 通过查看属性，易于重用已经存在的二级索引。
- 语法简单。 注意参数on也可以用来指定主键。事实上，为了更佳的可读性，我们鼓励，即使用主键进行subset，也使用参数on。

参数i里的subset（1/2）

- 例：subset所有**origin**是“**JFK**”的行

  flights["JFK", **on = "origin"**]

  #或写成 flights[.("JFK"), on = "origin"]

  #或 flights[list("JFK"), on = "origin"]

- 说明：
  - 这段语句执行的subset也是通过创建二级索引，基于快速二分法搜索的。但记住，它不会把这个二级索引自动创建为data.table的一个属性。当然后面我们也会教你如何将它设置为一个属性。
  - 如果我们已经添加了一个二级索引了，那么参数on就可以直接使用这个二级索引，而不是再对整个航班信息flights进行计算来生成该二级索引。

参数i里的subset（2/2）

- 选取所有从“JFK”起飞到达“LAX”的所有航班。

  flights[.("JFK", "LAX"), on = **c("origin", "dest")**]

- 说明：
  - 在参数i里面指定取值，在**参数on里面指定列名**。参数on必须是一个字符型的向量。
  - 因为计算索引非常快，所以我们不需要使用setindex()。除非你需要对某一列重复地进行subset操作。

参数j里的select

- 例：返回满足条件 origin = “LGA” 并且 dest = “TPA”的 **arr_delay列**的值。

```
flights[.("LGA", "TPA"), .(arr_delay), on = c("origin", "dest")]
```

Chaining

- 在上页例子的基础上，使用chaining来将结果降序排列。
- flights[.("LGA", "TPA"), .(arr_delay), on = c("origin", "dest")]**[order(-arr_delay)]**

参数j里的计算

- 找出满足条件 origin = “LGA”并且dest = “TPA”的 arr_delay列的最大值。
- flights[.("LGA", "TPA"), **max(arr_delay)**, on = c("origin", "dest")]

参数j里使用操作符“:=”进行sub-assign

- 把hours列中的24全部替换成0，但是这次使用参数on。

```
flights[.(24L), hour := 0L, on = "hour"]
```

- 说明：
  - 以前，只是为了更新一些行的hour列的取值，我们不得不调用函数setkey()将hour列设置为主键，这必须对整个data.table进行重新排序。但是现在，用参数on，原数据的顺序并没有改变，操作反而更快了！而代码还是如此简洁。

通过参数keyby聚合

- 例：找到每月从“JFK”起飞的航班起飞的最长延误时间，并按照月份排序。
- ans <- flights["JFK", max(dep_delay), keyby = month, on = "origin"]

参数mult

- 参数mult和上一节一样。它的默认值是“all”。我们可以选择是第一条还是最后一条符合条件的行被返回。
- 例：subset满足条件dest ＝ “BOS” 和 “DAY”的第一行
  - flights[c("BOS", "DAY"), on = "dest", **mult = "first"**]
- 例：subset满足条件 origin =“LGA” 或者 “JFK” 或者 “EWR”，并且 dest =“XNA”的最后一行。
  - flights[.(c("LGA", "JFK", "EWR"), "XNA"), on = c("origin", "dest"), **mult = "last"**]

参数nomatch

- 如果查询语句没有找到任何匹配的数据，通过指定参数nomatch，我们可以选择是返回 NA，还是忽略。
- 例：忽略上例中的NA行。
- flights[.(c("LGA", "JFK", "EWR"), "XNA"), mult = "last", on = c("origin", "dest"), **nomatch = 0L**]

自动索引

自动索引概述

- 当我们**第一次对某一列使用 == 或者 %in% 的时候，会自动创建一个二级索引**，它会被用来进行subset。
- 目前，自动索引只支持操作符 == 和 %in% 。而且只对一列起作用。
- 当某一列被自动创建为二级索引，会作为data.table的属性保存起来。这跟参数on不同，参数on会每次创建一个临时索引。

创建数据集

```
set.seed(1L)
dt = data.table(x = sample(1e5L, 1e7L, TRUE), y = runif(100L))
print(object.size(dt), units = "Mb")
# 114.4 Mb
```

自动索引

- 当我们**第一次对某一列使用 == 或者 %in% 的时候，会自动创建一个二级索引**，它会被用来进行subset。

```
t1 <- system.time(ans <- dt[x == 989L])
#第一次运行，自动创建索引。
```

```
   用户  系统  流逝
0.33 0.01 0.34
```

```
t2 <- system.time(dt[x == 989L])
#利用索引进行subset，速度快很多。
```

```
   用户  系统  流逝
      0     0     0
```

- 可以通过设置全局参数关闭自动索引：options(datatable.auto.index = FALSE)。