构建高效率的数据流水线在R中使用管道操作

Build efficient data streamline with pipelining in R

厦门大学王亚南经济研究院
任坤

例子

1. 生成10000个随机数服从均值为10、标准差为1的正态分布
2. 从中取出一个大小为100、不放回的样本
3. 对每个数字取对数
4. 对序列计算差分
5. 用红色线段绘制图像

例子

1. rnorm
2. sample
3. log
4. diff
5. plot

```
plot(diff(log(sample(rnorm(10000,mean=10,sd=1),size=
100,replace=FALSE))),col="red",type="l")
```

传统写法的弊端

```
plot(diff(log(sample(rnorm(10000,mean=10,sd=1),size=
100,replace=FALSE))),col="red",type="l")
```

不直观	不易维护
• 括号嵌套复杂不易读 • 编码顺序与执行顺序相反 • 提高可读性则导致引入许多“一次性变量”	• 无法轻易增减操作步骤 • 修改流程容易出现错误

传统写法的弊端

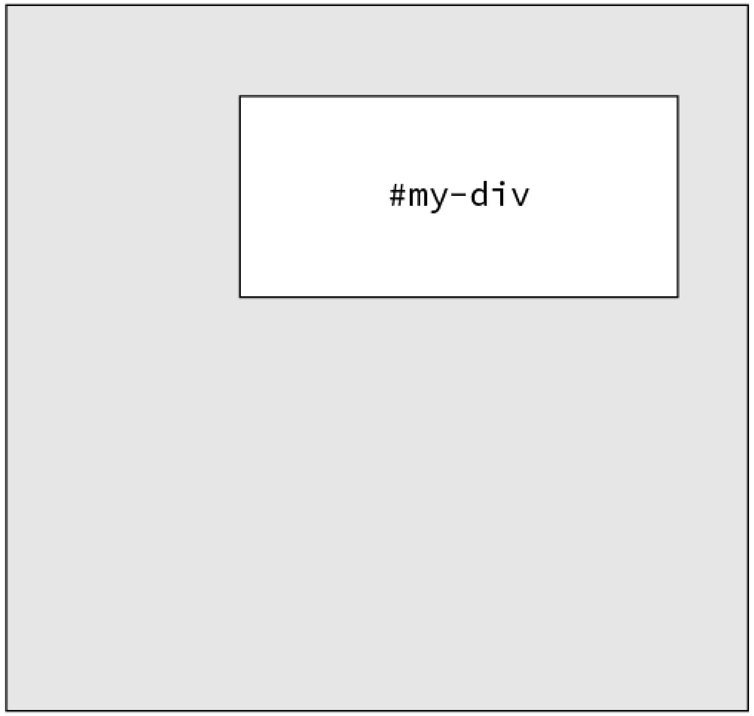
```
x <- rnorm(10000,mean=10,sd=1)
xs <- sample(x,100,replace=FALSE)
dl <- diff(log(xs))
plot(dl,col="red",type="l")
```

其他编程语言如何解决此问题？

- JavaScript
- C#
- F#
- …

JavaScript: Method Chaining in jQuery

```
$('#my-div')
   .css('background', 'blue')
   .height(100)
   .fadeIn(200);
```



C#: Lambda expression + LINQ

```
class Person
{
    public string FirstName;
    public string LastName;
    public DateTime BirthDate;
    public string Occupation;
    public Person Spouse;
    public List<Person> Children;
}
```

C#: Lambda expression + LINQ

```
var persons = new List<Person>();
// ... load some data ...
var avgAge = persons
    .Where(p => p.LastName == "Smiths")
    .Where(p => p.Spouse.BirthDate < new DateTime(1990, 1, 1))
    .OrderByDescending(p => p.Children.Count)
    .Take(5)
    .SelectMany(p => p.Children)
    .OrderBy(p => p.BirthDate)
    .Take(6)
    .Average(p => (DateTime.Now - p.BirthDate).TotalDays / 365.0);
```

F#: Pipeline operator |>

```
let |> x f = f x

let add x y = x + y
let multiply x y = x * y
let show x = printf x

10
|> add 3
|> multiply 2
|> show
```

F#: Pipeline operator |>

```
type Person =
    { FirstName : string
      LastName : string
      BirthDate : DateTime }
let persons = // ... load some data ...
let avgAge =
    persons
    |> List.filter(fun p -> p.LastName = "Smiths")
    |> List.filter(fun p -> p.BirthDate < DateTime(1990,1,1))
    |> List.averageBy(fun p -> (DateTime.Now - p.BirthDate).TotalDays / 365.0)
```

串联或者管道操作的优势

直观

- 嵌套**简单、易读**
- 编码顺序与执行顺序**相同**
- **避免**引入过多“一次性变量”

易维护

- 可以**轻易**增减操作步骤
- 修改流程**不易**出现错误

R中的管道操作符（magrittr::%>%）

```
rnorm(10000,mean=10,sd=1) %>%
  sample(size=100,replace=FALSE) %>%
  log %>%
  diff %>%
  plot(col="red",type="l")
```

R中的管道操作符（dplyr::%.%）

```
hflights %.%
  mutate(Speed=Distance/ActualElapsedTime) %.%
  group_by(UniqueCarrier) %.%
  summarize(n=length(Speed),
    speed.mean=mean(Speed,na.rm = T),
    speed.median=median(Speed,na.rm=T),
    speed.sd=sd(Speed,na.rm=T)) %.%
  mutate(speed.ssd=speed.mean/speed.sd) %.%
  arrange(desc(speed.ssd))
```

R中的管道操作符（magrittr+dplyr）

```
iris %>%
  filter(Species == "virginica") %>%
  select(-Species) %>%
  colMeans

iris %>%
  filter(., Species == "virginica") %>%
  select(., -Species) %>%
  colMeans
```

存在的问题

- 试图统一不同种类的管道操作
  - 传递到函数的第一个参数
  - 传递到特定符号（例如“.”）
- 造成的问题
  - 必须分析、猜测用户试图使用哪种管道操作
  - 然后才能决定采用何种表达式变换和解析
  - 对下面的表达式无法适用：

```
rnorm(100) %>%
  sample(.,length(.)*0.2,FALSE) %>%
  plot(.,main=sprintf("length: %d",length(.)))
```

pipeR: 在R中定义3种管道操作符

- 管道操作符的本质
- 在R中进行3种管道操作
  - %>%: 将结果输送到函数的第一个参数
  - %>>%: 将结果输送到表达式中的“.”符号
  - %|>%: 将结果输送给 lambda 表达式

R编程知识回顾：自定义符号

```
`%^_^%` <- function(from,to) {
  cat(paste(from,"smiles to",to))
}

> "Ken" %^_^% "Jenny"
Ken smiles to Jenny
```

R编程知识回顾：表达式对象（Expression）

```
> expr1 <- expression(1+1)
> expr1
expression(1 + 1)
> eval(expr1)
[1] 2
```

R编程知识回顾：环境（Environment）

```
> x <- 2
> y <- 3
> env <- new.env()
> env$x <- 1
> eval(expression(x))
[1] 2
> eval(expression(x),envir = env)
[1] 1
> eval(expression(x+y),envir = env)
[1] 4
```

%>%: 将结果输送到函数的第一个参数

```
`%>%` <- function(.,fun) {
  . <- substitute(.)
  fun <- as.list(substitute(fun))
  call <- as.call(c(fun[1],.,fun[-1]))
  eval(call,envir = parent.frame())
}

rnorm(100) %>% plot
rnorm(100) %>% plot()
rnorm(100) %>% plot(col="red")
rnorm(100) %>% sample(size=100,replace=FALSE) %>% hist
```

%>%: 将结果输送到函数的第一个参数

```
rnorm(10000,mean=10,sd=1) %>%
  sample(size=100,replace=FALSE) %>%
  log %>%
  diff %>%
  plot(col="red",type="l")
```

%>>%: 将结果输送到表达式中的“.”符号

```
`%>>%` <- function(.,expr) {
  env <- new.env(parent = parent.frame())
  env$. <- .
  expr <- substitute(expr)
  eval(expr,envir = env)
}

rnorm(100) %>>% plot(.)
rnorm(100) %>>% plot(., col="red")
rnorm(100) %>>% sample(., size=length(.)*0.5)
mtcars %>>% lm(mpg ~ cyl + disp, data=.) %>% summary
```

%>>%: 将结果输送到表达式中的“.”符号

```
rnorm(100) %>>%
  sample(.,length(.)*0.2,FALSE) %>>%
  plot(.,main=sprintf("length: %d",length(.)))

rnorm(100) %>>% {
  par(mfrow=c(1,2))
  hist(.,main="hist")
  plot(.,col="red",main=sprintf("%d",length(.)))
}
```

%>>%: 将结果输送到表达式中的“.”符号

```
rnorm(10000,mean=10,sd=1) %>>%
  sample(.,size=length(.)*0.1,replace=FALSE) %>%
  log %>%
  diff %>>%
  plot(.,col="red",type="l",
    main=sprintf("length: %d",length(.)))

mtcars %>>%
  lm(mpg ~ ., data=.) %>%
  summary
```

%|>%: 将结果输送给 lambda 表达式

```
rnorm(100) %|>% (x ~ plot(x))

`%|>%` <- function(.,lambda) {
  env <- new.env(parent = parent.frame())
  eval(as.call(list(`<-`,lambda[[2]],.)),envir = env)
  eval(lambda[[3]],envir = env)
}

mtcars %|>%
  (df ~ lm(mpg ~ ., data=df)) %>%
  summary
```

%|>%: 将结果输送给 lambda 表达式

```
filter <- df ~ df[c("mpg","cyl","disp")]
reg <- df ~ lm(mpg ~ ., data=df)
resplot <- m ~ plot(m$residuals,col="red")

mtcars %|>%
  filter %|>%
  reg %|>%
  resplot
```

三种管道操作符的混合使用

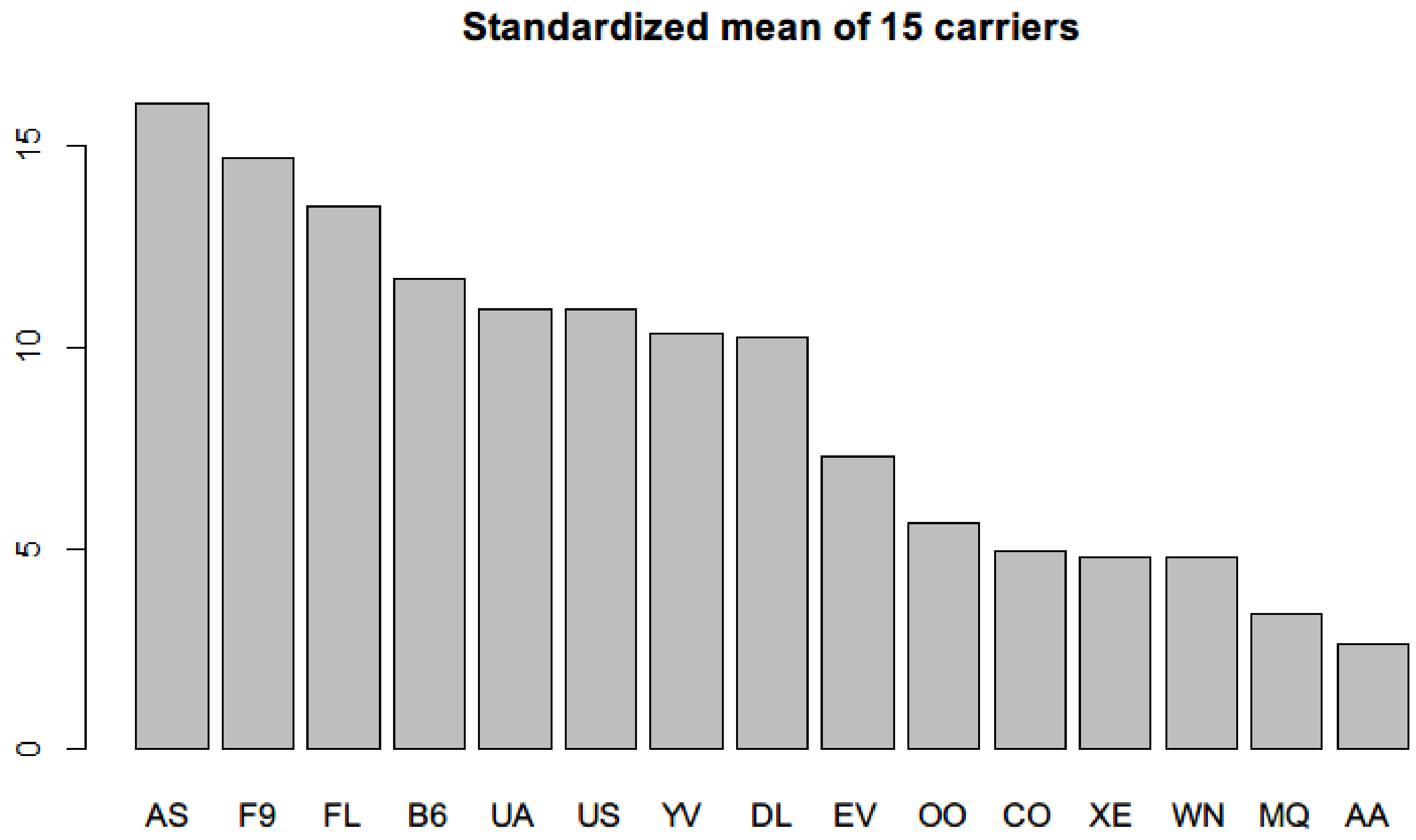
```
mtcars %|>%
  (df ~ lm(mpg ~ ., data=df)) %>%
  summary %>>%
  .$fstatistic
```

dplyr+pipeR

```
library(dplyr)
library(hflights)
library(pipeR)
data(hflights)
```

```
hflights %>%
  mutate(Speed=Distance/ActualElapsedTime) %>%
  group_by(UniqueCarrier) %>%
  summarize(n=length(Speed),speed.mean=mean(Speed,na.rm = T),
    speed.median=median(Speed,na.rm=T),
    speed.sd=sd(Speed,na.rm=T)) %>%
  mutate(speed.ssd=speed.mean/speed.sd) %>%
  arrange(desc(speed.ssd)) %>>%
  barplot(.$speed.ssd, names.arg = .$UniqueCarrier,
    main=sprintf("Standardized mean of %d carriers", nrow(.)))
```

dplyr+pipeR



Standardized mean of 15 carriers
0
5
10
15
AS
F9
FL
B6
UA
US
YV
DL
EV
OO
CO
XE
WN
MQ
AA

总结

- %>%: 把符号左边的对象传输到右边**函数**的第一个参数，右边可以是函数名称（name），也可以是函数调用（call）；
- %>>%: 把符号左边的对象传输到右边**表达式**中的“.”，右边可以是任意表达式，引用前面的对象时必须用“.”来表示；
- %|>%: 把符号左边的对象传输到右边的lambda表达式来决定如何计算，其中lambda表达式必须是(x ~ f(x))，x定义符号名称，f(x)是任意关于x的表达式。
- 三种管道符号可以任意串接，但使用时必须明确意义。

使用案例：正则表达式分情况匹配

```
readline("? ") %>%
 str_detect(c("^a","^b","^c","^.")) %>%
 structure(names=c("a","b","c","no")) %>%
 which %>%
 head(1) %>%
 names %>%
 switch(
   a="case 1",
   b="case 2",
   c="case 3",
   no="no case") %>%
 cat
```

任坤<renkun@outlook.com>
http://renkun.me/
GitHub: renkun-ken

pipeR项目主页
http://renkun.me/pipeR/

GitHub源代码
https://github.com/renkun-ken/pipeR

谢谢！