介绍

如果你将要写很多shiny app（希望在阅读完本书之后）。在工作流程上投入一定的时间是值得的。花时间改进工作流程，从长远来看，往往带来可观的回报，让你在写R代码的时间更多（错误更少了），成果更快的出来。减少shiny 开发过程中的痛苦，并且更快的提高技能。
这一章那个的目的是提高shiny 的工作流程：

1. 创建应用，减少开发周期
2. 调试应用，工作流程帮你找到代码中的问题
3. 写reprexes，每个代码块解决特定的问题。reprexes是一个强大的解决bug技术，如果想从别的地方获得帮助，他们是必不可少的。

6.1 开发工作流程

优化你的开发工作流程是为了减少更改的时间和看到结果的时间，迭代越快，就可以尝试的更多，可以进行的试验更多，您就可以成为一个更好的shiny开发人员。这里有两个主要的工作流程去优化：第一次创建应用程序；加快代码迭代和实验结果的周期。

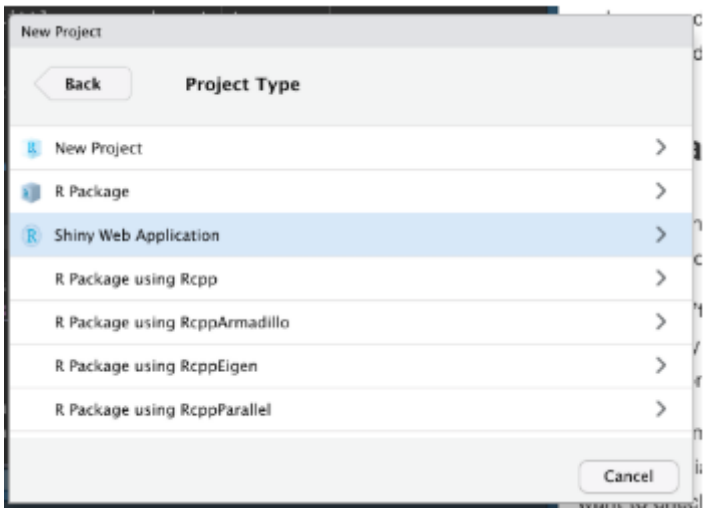
6.1.1 创建应用

一个基本的shiny刚开始可能是这样的：

```
library(shiny)
ui <- fluidPage(
)
server <- function(input, output, session) {
}
shinyApp(ui, server)
```

也有可能是使用Rstudio的快捷键：

1. 如果有了`app.R`文件，可能是输入`shinyapp`然后按`shift + Tab`来插入shiny app的模板
2. 如开始一个新的工程，可能是从菜单里面选择（这些在第二章都介绍过了）



你可能认为这是没有必要学习快捷键的，因为使用他的频率非常少，但是创建一个简单的shiny app是在开始较大的项目之前检查基本框架的最好方法，并且是最好的调试工具。（从简单开始，代码要规范）

6.1.2 发现改变

你创建了一个shiny app。但是你会使用上百次，所以掌握这基本的工作流程是非常有必要的。第一个减少迭代周期的方法是少使用`Run app`这个按钮，而是使用快捷键`Cmd/Ctrl + shift + Enter`。下面是工作流程：

1. 写代码
2. 用快捷键`Cmd/Ctrl + shift + Enter`启动app
3. 和这个app交互检查
4. 关闭app
5. 从第一步再次运行

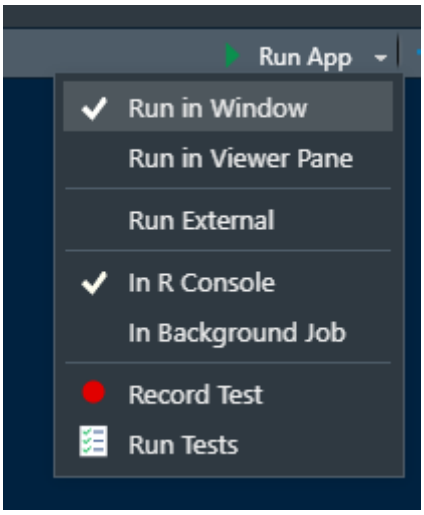
另外一个让你shiny app多次运行检查的是加个全局设置：`options(shiny.autoreload=TRUE)`，然后按照这个教程（https://github.com/sol-eng/background-jobs/tree/master/shiny-job ）来运行该shiny app。保存文件后，将重新启动app。无需关闭并重新启动，这将导致更快的工作流程：

1. 写代码，并使用`Ctrl + s`保存代码
2. 和软件交互
3. 回到1

上面的代码不适用于大的shiny app，因为当你的app越来愈大，交互式实验越来越行不通，很难记住检查应用的步骤，在17章将介绍，自动工具来测试你的shiny app。意味着你需要写自动化代码来检测你的shiny app。但是相对于大型的shiny app。这些时间真的是相当值得的。

6.1.3 控制shiny的窗口

运行shiny app的时候，会跳出来一个窗口，但是这里有个按钮，可以帮助你选择shiny app运行在哪里。打开`Run App`按钮旁边的向下按钮：



1. `Run in Viewer Pane`指的是将shiny app运行在`Rstudio`的`Viewer`这个窗口里面（通常在右下角），对于比较小的app是非常有用的：边运行代码边看shiny界面。
2. `Run External`是打开你常用的浏览器，通常是适用于shiny app内容比较丰富的（shiny app比较大的时候）。

6.2 调试

6.2.1 阅读错误回溯

每一个错误都有`traceback`或者`call stack`。从字面上追溯一系列的调用错误。比如下面：`f()`调用`g()`，`g()`调用`h()`。

```
f <- function(x) g(x)
g <- function(x) h(x)
h <- function(x) x * 2
```

发生错误了：

```
f("a")
#> Error in x * 2: non-numeric argument to binary operator
```

他们调用堆栈的顺序如下：

```
1: f("a")
2: g(x)
3: h(x)
```

如果经常使用R，那你肯定对`traceback()`肯定非常熟悉，你可以使用直接修改，一次次测试哪一个函数错了，但是在shiny app开始运行的时候，你修改了代码之后，正在运行的shiny app不会马上发生变化。但是，shiny app在运行的时候，会自动将这些错误打印出来。
比如，下面的简单的app代码：

```
library(shiny)

ui <- fluidPage(
  selectInput("n", "N", 1:10),
  plotOutput("plot")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    n <- f(input$n)
    plot(head(cars, n))
  }, res = 96)
}
shinyApp(ui, server)
```

如果你运行了代码，在控制台里面会看到这个`traceback`。

```
Warning: Error in *: non-numeric argument to binary operator
  173: g [~/.active-rstudio-document#4]
  172: f [~/.active-rstudio-document#3]
  171: renderPlot [~/.active-rstudio-document#13]
  169: func
  129: drawPlot
  115: <reactive:plotObj>
   99: drawReactive
   86: origRenderFunc
   85: output$plot
    5: runApp
    3: print.shiny.appobj
    1: source
```

为了理解发生了什么，将上面的`traceback`从下往上读：

```
Warning: Error in *: non-numeric argument to binary operator
    1: source
    3: print.shiny.appobj
    5: runApp
   85: output$plot
   86: origRenderFunc
   99: drawReactive
  115: <reactive:plotObj>
  129: drawPlot
  169: func
  171: renderPlot [sample.R#13]
  172: f [sample.R#3]
  173: g [sample.R#4]
```

上面的代码分成三个基础部分：

1. 启动app

```
1: source
3: print.shiny.appobj
5: runApp
```

这里的代码是，先是使用代码`source()`加载代码，然后使用`print.shiny.appobj()`调用`runApp()`来启动app。通常，可以忽视这段代码。只是当代码可以在shiny里面可以运行。

2. shiny内置的一些反应式代码：

```
 85: output$plot
 86: origRenderFunc
 99: drawReactive
115: <reactive:plotObj>
129: drawPlot
169: func
```

这里出现的`output$plot`是说这个会出现了错误，下面的是一些底层的函数，可以忽略。

3. 在非常的下面，就可以看到被你写的代码：

```
171: renderPlot [sample.R#13]
172: f [sample.R#3]
173: g [sample.R#4]
```

这个代码在`renderPlot()`里面，应该注意这个几个函数（有函数所在的文件名字和所在的行数）。

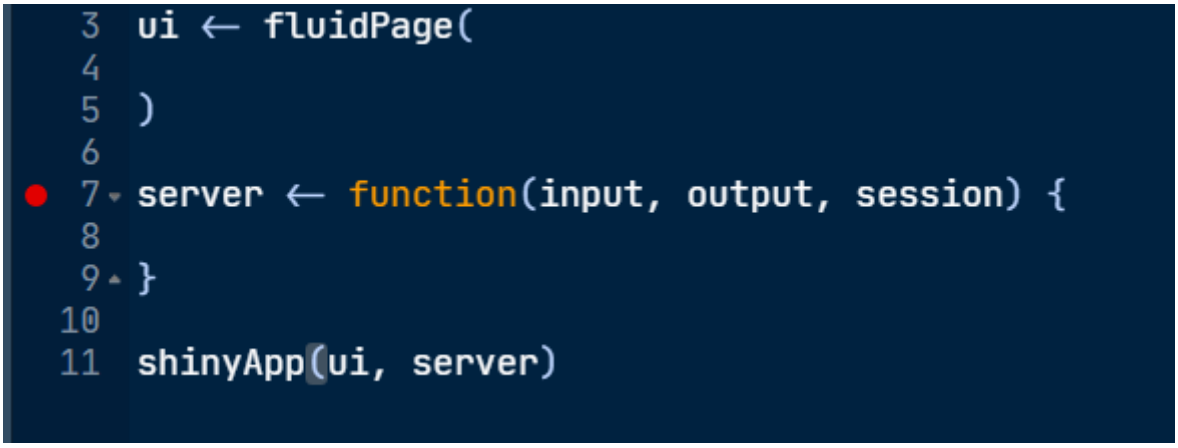
6.2.2 交互式调试

如果已经确定错误的位置，想要搞清楚到底是什么原因照成的，最有用的调试工具就是`interactive debuger`。这种断点式调试，给你展示一个调试的控制台，让你在运行代码的时候弄清楚怎么错了，有两种方式来启动调试。

1. 在代码里面加入`browser()`函数，这个R里面标准的启动交互式调试的方法，不管你是否在运行shiny。可以将这个和`if`一起使用：

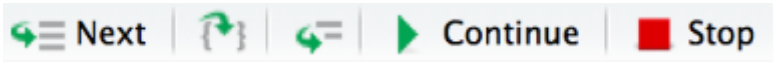
```
if (input$debug) {
  browser()
}
```

2. 在Rstudio里面添加一个断点，只要点击代码左边的数字左边的空白就行了，如果想取消断点，再点击代码左边的数字左边的空白。



这个优点就是，不需要使用代码就可以调试。

如果使用Rstudio，在进入调试的时候，会看到这个工具栏：



工具栏是记住现有的调试命令的简单方法，只需要记住字母就可以调试代码，这里介绍三个最重要的调试命令：

1. Next， n 执行下一个函数，如果你有个变量叫n，需要使用`print(n)`来显示这个n的值。
2. Continue，c 退出交互式调试，并在常规状态下执行函数，这是当你修改好这个函数，打算检测这个修改好的函数对不对的时候。
3. Stop， Q 停止调试，终止函数（退出调试模式）。这样一般用于你确定问题出在哪里，并且修改好它，然后重新加载代码。

除了这些逐行运行的代码外，还将运行一堆交互式代码来了解为什么看到错误。调试是系统的将您的期望和现实进行比较直到发现不匹配的过程。

6.2.3 案例学习

为了强调这个调试的基本知识，我将展示我在写章节10.1.2遇到的一些bug。你将要看到我是如何使用这些交互式工具来解决bug的。

最初的目的很简单，我有个数据集叫`sales`，我想要根据`territory`来筛选，这个是我的代码：

```
sales <- readr::read_csv("sales-dashboard/sales_data_sample.csv")
sales <- sales[c("TERRITORY", "ORDERDATE", "ORDERNUMBER", "PRODUCTCODE",
"QUANTITYORDERED", "PRICEEACH")]
sales
#> # A tibble: 2,823 x 6
#>   TERRITORY ORDERDATE      ORDERNUMBER PRODUCTCODE QUANTITYORDERED PRICEEACH
#>   <chr>     <chr>                <dbl> <chr>                <dbl>     <dbl>
#> 1 <NA>      2/24/2003 0:00       10107 S10_1678                30      95.7
#> 2 EMEA      5/7/2003 0:00        10121 S10_1678                34      81.4
#> 3 EMEA      7/1/2003 0:00        10134 S10_1678                41      94.7
#> 4 <NA>      8/25/2003 0:00       10145 S10_1678                45      83.3
#> # … with 2,819 more rows
```

这个是`sales`的`TERRITORY`变量：

```
unique(sales$TERRITORY)
#> [1] NA      "EMEA"  "APAC"  "Japan"
```

在我刚开始写这个代码的时候，我为了很简单：

```
ui <- fluidPage(
  selectInput("territory", "territory", choices = unique(sales$TERRITORY)),
  tableOutput("selected")
)
server <- function(input, output, session) {
  selected <- reactive(sales[sales$TERRITORY == input$territory, ])
  output$selected <- renderTable(head(selected(), 10))
}
```

我觉得这不就是8行的小app，能有什么问题呢，但是当我打开这个shiny app。我看到很多缺失值，不管我选`territory`是多少，都可以看到缺失值。所以停止这个app。验证我的数据筛选对不对。我是这么做的：

```
sales[sales$TERRITORY == "EMEA", ]
#> # A tibble: 2,481 x 6
#>   TERRITORY ORDERDATE     ORDERNUMBER PRODUCTCODE QUANTITYORDERED PRICEEACH
#>   <chr>     <chr>               <dbl> <chr>                <dbl>     <dbl>
#> 1 <NA>      <NA>                   NA <NA>                    NA        NA
#> 2 EMEA      5/7/2003 0:00       10121 S10_1678                34      81.4
#> 3 EMEA      7/1/2003 0:00       10134 S10_1678                41      94.7
#> 4 <NA>      <NA>                   NA <NA>                    NA        NA
#> # … with 2,477 more rows
```

芜湖～我发现问题了，`territory`包含着大量的缺失值导致出现`NA == NA`。所以我想到下面几个方案：

1. 使用`subset()`或者`dplyr::filter()`函数，这些将自动移除缺失值。
2. 将缺失值筛选掉：`sales[!is.na(sales$TERRITORY) & sales$TERRITORY == "EMEA", ]`
3. 使用`which()`函将逻辑向量转换为整数向量为TRUE的位置。
4. 使用`%in%`而不是`==`。因为`NA %in% NA`返回TRUE。比如说：`sales[sales$TERRITORY %in% "EMEA", ]`

我决定使用`subset()`结合`==`，效果如下：shi

```
subset(sales, TERRITORY == "EMEA")
#> # A tibble: 1,407 x 6
#>   TERRITORY ORDERDATE       ORDERNUMBER PRODUCTCODE QUANTITYORDERED PRICEEACH
#>   <chr>     <chr>                 <dbl> <chr>                <dbl>     <dbl>
#> 1 EMEA      5/7/2003 0:00         10121 S10_1678                34      81.4
#> 2 EMEA      7/1/2003 0:00         10134 S10_1678                41      94.7
#> 3 EMEA      11/11/2003 0:00       10180 S10_1678                29      86.1
#> 4 EMEA      11/18/2003 0:00       10188 S10_1678                48     100
#> # … with 1,403 more rows
```

但是这个情况依然存在，当我选择`NA`的时候。

```
subset(sales, TERRITORY == NA)
#> # A tibble: 0 x 6
#> # … with 6 variables: TERRITORY <chr>, ORDERDATE <chr>, ORDERNUMBER <dbl>,
#> #   PRODUCTCODE <chr>, QUANTITYORDERED <dbl>, PRICEEACH <dbl>
```

因此我使用`subset()`结合`%in%`，效果不错：

```
subset(sales, TERRITORY %in% NA)
#> # A tibble: 1,074 x 6
#>   TERRITORY ORDERDATE       ORDERNUMBER PRODUCTCODE QUANTITYORDERED PRICEEACH
#>   <chr>     <chr>                 <dbl> <chr>                 <dbl>     <dbl>
#> 1 <NA>      2/24/2003 0:00        10107 S10_1678                 30      95.7
#> 2 <NA>      8/25/2003 0:00        10145 S10_1678                 45      83.3
#> 3 <NA>      10/10/2003 0:00       10159 S10_1678                 49     100
#> 4 <NA>      10/28/2003 0:00       10168 S10_1678                 36      96.7
#> # … with 1,070 more rows
```

因此，我就更新我的shiny app了，但是当我又运行的时候，在选择为`NA`的时候，我一行数据都看不到。
我虽然在R的控制台里面，我做了我该做的，但是在shiny app里面竟然不行，我就怀疑是不是`reacive()`这个反应式的问题。所以我在这个函数里面加入了`browser()`语句，代码如下：

```
server <- function(input, output, session) {
  selected <- reactive({
    browser()
    subset(sales, TERRITORY %in% input$territory)
  })
  output$selected <- renderTable(head(selected(), 10))
}
```

我将上面的代码逐行调试（我在一开始将输入设置为`NA`，这个更加方便我调试）。我发现数据没问题，subset函数也没问题，输入也没问题，但是运行的结果就是没有数据，

```
input$territory
#> [1] "NA"
```

```
subset(sales, TERRITORY %in% "NA")
#> # A tibble: 0 x 6
#> # … with 6 variables: TERRITORY <chr>, ORDERDATE <chr>, ORDERNUMBER <dbl>,
#> #   PRODUCTCODE <chr>, QUANTITYORDERED <dbl>, PRICEEACH <dbl>
```

看到了吧，由于`input$territory`输出的是`"NA"`，这个是字符串，不是`NA`（这个是缺失标志）。`sales`的`territory`变量里面肯定没有`"NA"`。，我修改好代码，如下：

```
server <- function(input, output, session) {
  selected <- reactive({
    if (input$territory == "NA") {
      subset(sales, is.na(TERRITORY))
    } else {
      subset(sales, TERRITORY == input$territory)
    }
  })
  output$selected <- renderTable(head(selected(), 10))
}
```

几个星期以后，我再次看到这个`territory`。这里有Europe、EMEA、APAC，那么北美（North America）去哪里了，所以说，NA有可能是北美的意思，但是R将他它判断为缺失值了，所以当我们读这个数据的时候，使用下面这个代码就不会有这个误会：

```
sales <- readr::read_csv("sales-dashboard/sales_data_sample.csv", na = "")
unique(sales$TERRITORY)
#> [1] "NA"    "EMEA"  "APAC"  "Japan"
```

6.2.4 反应式调试

如果在反应式里面调试代码，因为你不知道你的运行顺序，可以使用`print()`或者`message()`，当然使用`str()`或者使用`glue::glue()`也是非常不错的选择。

这是一个一个小demo，用来传递一些小的方法，注意我现在是将`message()`放到`reactive()`里面。会打印我更新的数据。

```
ui <- fluidPage(
  sliderInput("x", "x", value = 1, min = 0, max = 10),
  sliderInput("y", "y", value = 2, min = 0, max = 10),
  sliderInput("z", "z", value = 3, min = 0, max = 10),
  textOutput("total")
)
server <- function(input, output, session) {
  observeEvent(input$x, {
    message(glue("Updating y from {input$y} to {input$x * 2}"))
    updateSliderInput(session, "y", value = input$x * 2)
  })

  total <- reactive({
    total <- input$x + input$y + input$z
    message(glue("New total is {total}"))
    total
  })

  output$total <- renderText({
    total()
  })
}
```

6.3 获得帮助

在上面的方法都试过之后，如果还是不行，可以尝试问问别人：https://community.rstudio.com/c/shiny 这个社区都是使用Rstudio、R的人聚集而来的。

为了让别人更快的知道你的错误，你需要创建一个小代码（create a reprex），可以让别人尝试，帮助你弄清楚到底发生了什么，你的这个小代码越简单越好。

下面部分被我删除了，都是关于求助的礼仪，不管是在群还是在论坛上，最好附上的代码和数据。如果数据保密，就做个类似的，只要是能反映出你的错误即可。

内容可能会一直更新，可以查看我的微信公众号：pypi

获得最新的关于这部分的内容

微信扫一扫：

知乎：https://www.zhihu.com/people/fa-fa-1-94

csdn：https://blog.csdn.net/yuanzhoulvpi

github: https://github.com/yuanzhoulvpi2017/master_shiny_CN

如果有错误，欢迎指正，邮箱联系我：yuanzhoulvpi@outlook.com