

2.1 介绍

在这一章中，我们将创建一个简单的shiny app。我将要向你展示shiny app所需最简单的组件。然后你将学会如何运行和停止shiny app。接下来，你将学习shiny app的两个最重要的组件：UI部分（用户界面）被人看到的部分、和server部分（让你shiny app在可以稳定运行的部分）。shiny使用的是反应式编程。也就是说shiny会自动的更新输出当输入改变的时候。所以当我们把shiny app的反应表达式学完时，这章就结束了。

如果还没有安装shiny安装包，现在应该安装：

```
install.packages("shiny")
```

安装好之后，加载到R的会话里面：

```
library(shiny)
```

2.2 创建应用目录和文件

有很多种方法创建一个shiny app。最简单的是在一个文件夹下，创建一个叫app.R的文件。这个app.R的文件将告诉shiny。你的shiny是长得什么样，背后运行行为是什么。

试一试创建一个文件夹，在文件夹下添加一个叫app.R的文件，用Rstudio打开这个文件。并将下面的代码写到app.R里面：

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

Rstudio 小提示：你可以很轻松的使用Rstudio来创建一个文件夹和一个app.R文件。一步步操作流程如下：“File” 然后 选择“New Project”然后选择“New Directory” 然后选择 “Shiny Web Application”。或者你已经创建一个app.R的文件了，你可以快速的输入“shinyapp”。然后回车（我的win10系统上，输入shinyapp的时候就有了，然后回车自动出现shiny app框架，但是原文是按 Shift + tab键。我也不知道为啥）。

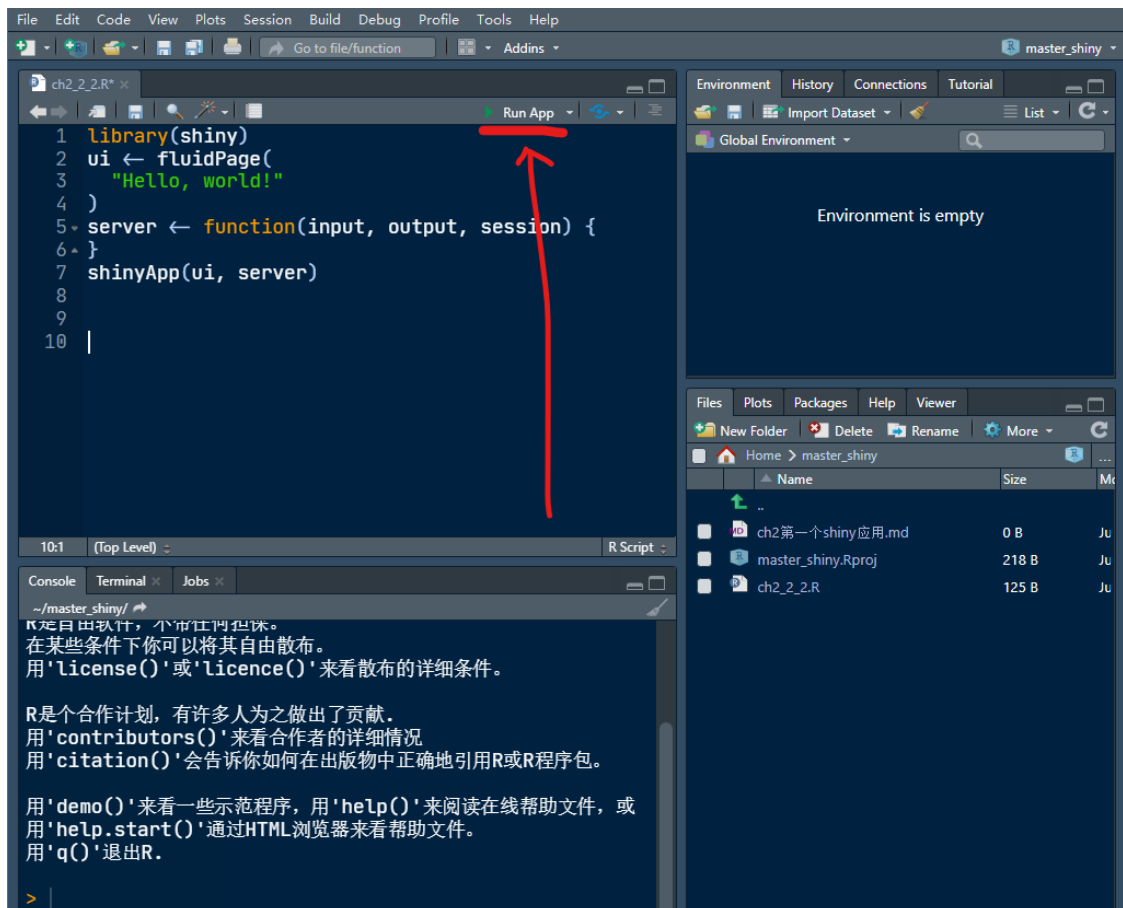
运行好之后，会出现和上面的差不多的代码。实际上，我们的app.R做了四件事：

1. 使用 `library(shiny)` 加载shiny包。
2. 定义了交互界面，也就是和人交互的html网页部分。在上面的代码，可以看到 `hello world!`。
3. 在 `server` 里面，定义着这个shiny app的行为。但是 `server` 是空的，所以现在这个shiny app啥都做不了。
4. `shinyApp(ui, server)` 意味着从 `ui`, `server` 构建并开始一个shiny app。

2.3 运行应用、停止应用

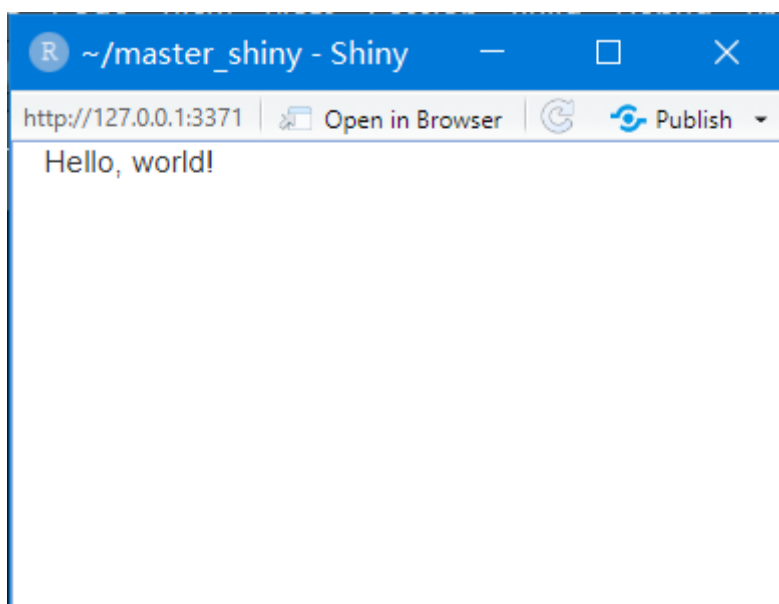
这里有几个方式运行这个app：

1. 点击 Run App 按钮，在文本编辑窗口的右上角。



2. 使用键盘快捷键： `Cmd/ctrl + shift + Enter`
3. 如果没有使用Rstudio。可以使用 `source()` 函数来加载这个 `app.R` 文件。或者使用 `shiny::runApp()` 来加载 `app.R`。

随便选择上面的任意一个方式，运行这个shiny app。如果你的运行结果和下面的图一样，那么恭喜你，你的第一个shiny app成功了。



在没有关闭上面这个小窗口的时候，回到Rstudio。看R的console（R的对话界面，或者是控制台）。你也看到这样的一句话（可能具体的数值不一样）：

#Listening on http://127.0.0.1:3371

上面这句话说，你可以在你的计算机的3371端口上找到我们正在运行的shiny app。其中127.0.0.1是标准地址，后面的3371是端口（可能每个人运行的端口都不一样，没有关系）将上面的网站链接复制 `http://127.0.0.1:3371` 到浏览器，就可以在浏览器中打开这个shiny app。

如何关闭shiny app。有以下几个方法：

1. R的控制台窗口的右上角有个红色的 stop 按钮。
2. 点击控制台，然后按 `Esc` 按钮（如果用的不是Rstudio软件，使用 `ctrl + c` 结束程序）。
3. 关闭这个shiny app的窗口。

2.4 添加UI控件

我们现在的UI部分太小了，向里面加入输入、输出控件。我们的目的是使用datasets包的内置数据集，来建立一个简单的shiny app。将你的UI代码部分用下面代码替代：

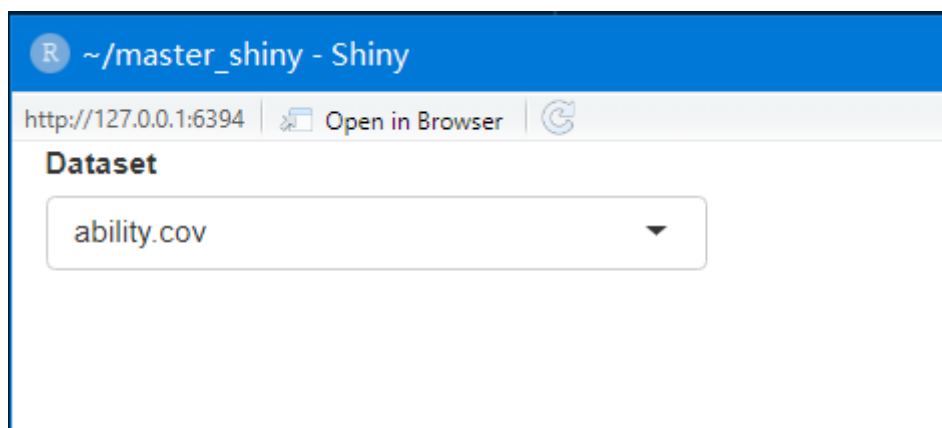
```
ui <- fluidPage(  
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),  
  verbatimTextOutput("summary"),  
  tableOutput("table")  
)
```

这里小demo使用下面4个新的函数：

1. `fluidPage()` 是一个布局函数，用于设置shiny app的页面，将在章节3.4学到更多。
2. `selectInput()` 是一个输入控件，shiny app的按钮提供一个可选择列表，shiny app的使用者可以自己选择。在这个小案例中。在一个小盒子（其实就是一个选择框）里面选择R内置的数据集。关于输入控件，将在章节3.2学到更多。
3. `verbatimTextOutput()` 和 `tableOutput()` 是输出控件，是用来告诉shiny将渲染的输出在UI的哪里显示（关于渲染，后面会介绍）。`verbatimTextOutput()` 显示R的代码。`tableOutput()` 显示表格。关于这些将在章节3.3学到更多。

布局函数，输入输出函数有着不同的用处。但是本质都是一样的：用来生成HTML。如果在shiny app应用程序之外调用他们中的任意一个。可以在浏览器的控制台看到这些HTML输出。不要害怕有错误，多看看代码和后台运行情况。这些将在章22学到更多。

将上面的代码继续运行，可以看到下面的页面，可以看到有个选择的框（也就是上面说的小盒子）。但是我们只能看到输入，没有看到输出，因为我们还没告诉shiny输入和输出的内在联系。



完整代码如下：

```
library(shiny)
ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)

server <- function(input, output, session) {
}

shinyApp(ui, server)
```

2.5 添加行为

接下来，就是在 `server` 部分定义输出。这样才能将输出展示在UI里面。

shiny使用的是反应式编程。使得shiny app有交互性。反应式编程(reactive programming)将在章节4有着详细的介绍。但是现在，只需要知道：反应式编程是告诉shiny如何去计算，而不是马上计算（我感觉就是把路给你铺好，你接下来好好的走这条路，但是你现在不用出发）。这个区别就像是给你菜谱和马上做一个三明治一样。

在这里案例里，将告诉shiny。如何填充上面代码里面的 `summary` 和 `table` 这两个输出。我们给输出提供了一个菜谱。将上面的server函数用下面的代码替代：

```
server <- function(input, output, session) {
  output$summary <- renderPrint({
    dataset <- get(input$dataset, "package:datasets")
    summary(dataset)
  })

  output$table <- renderTable({
    dataset <- get(input$dataset, "package:datasets")
    dataset
  })
}
```

基本上每一个输出，在shiny的server里面写成的形式都是下面这样的：

```
output$ID <- renderTYPE({
  # Expression that generates whatever kind of output
  # renderTYPE expects
})
```

赋值操作 `<-` 的左边是 `output$ID`。意思要将右边的运算结果赋给output的ID。赋值操作的右边是指定的渲染函数，在渲染函数里面放入你的代码（注意要使用`{}`将代码包括起来）。在上面的代码中，使用的是 `renderPrint()` 和 `renderPlot()` 渲染你的代码运行结果。

每一个 `render*` 渲染函数都有对应的 `*output` 输出函数。我们使用 `renderPrint()` 来构造和显示定宽文本（普通文本）。使用 `renderTable()` 将数据框用表格来显示。

运行下面的完整代码：

```
library(shiny)
ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
```

```

tableOutput("table")
)

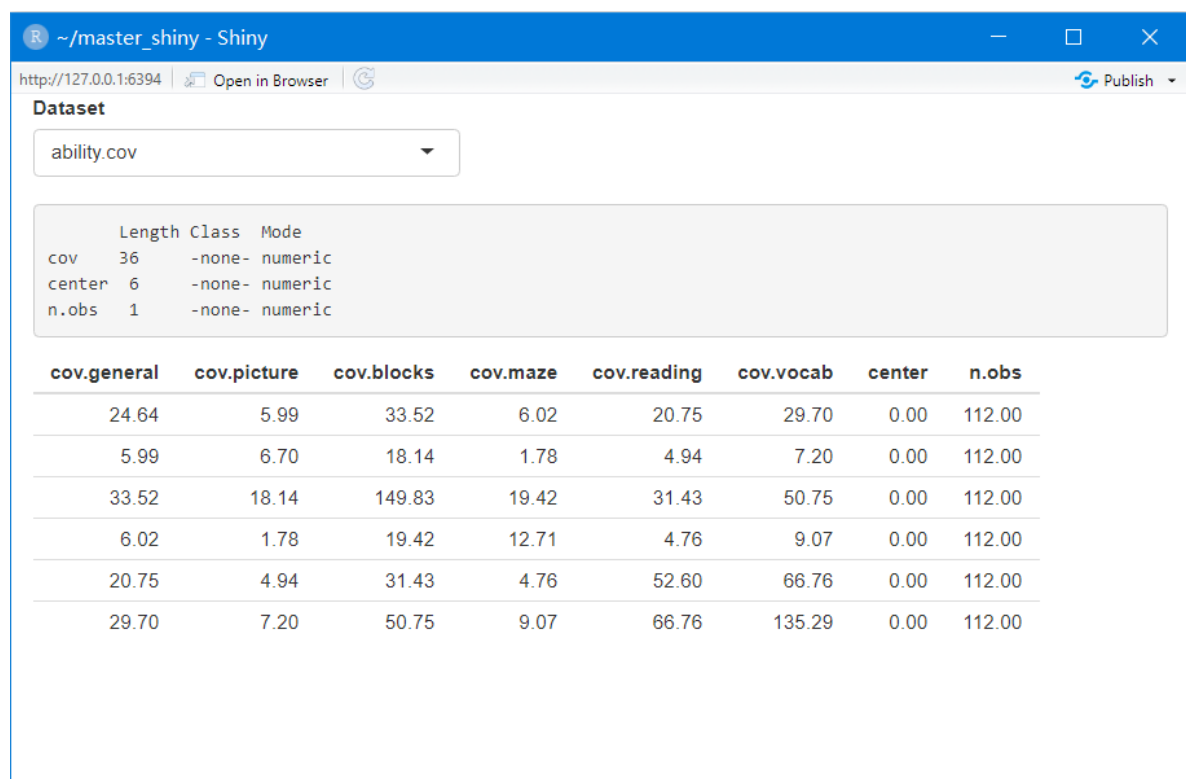
server <- function(input, output, session) {
  output$summary <- renderPrint({
    dataset <- get(input$dataset, "package:datasets")
    summary(dataset)
  })

  output$table <- renderTable({
    dataset <- get(input$dataset, "package:datasets")
    dataset
  })
}

shinyApp(ui, server)

```

输出结果如下：



The screenshot shows a Shiny application window titled '~/.master_shiny - Shiny'. The URL bar shows 'http://127.0.0.1:6394'. The app has a 'Dataset' dropdown menu with 'ability.cov' selected. Below the menu, there is a summary table for the dataset:

	Length	Class	Mode
cov	36	-none-	numeric
center	6	-none-	numeric
n.obs	1	-none-	numeric

Below the summary table is a large table showing the covariance matrix for the 'ability.cov' dataset. The columns are labeled 'cov.general', 'cov.picture', 'cov.blocks', 'cov.maze', 'cov.reading', 'cov.vocab', 'center', and 'n.obs'.

cov.general	cov.picture	cov.blocks	cov.maze	cov.reading	cov.vocab	center	n.obs
24.64	5.99	33.52	6.02	20.75	29.70	0.00	112.00
5.99	6.70	18.14	1.78	4.94	7.20	0.00	112.00
33.52	18.14	149.83	19.42	31.43	50.75	0.00	112.00
6.02	1.78	19.42	12.71	4.76	9.07	0.00	112.00
20.75	4.94	31.43	4.76	52.60	66.76	0.00	112.00
29.70	7.20	50.75	9.07	66.76	135.29	0.00	112.00

值得注意的是，我没有写任何代码来检测shiny app的 `input$dataset` 是否变换了再更新两个输出结果。原因是两个输出都是反应式的：他们自动再计算，当输入部分变化的时候。因为我编写的两个渲染代码都是使用了 `input$dataset`。只要用户在UI部分选择新的东西，这两个output就会自动在背后计算然后再在UI上呈现出来。

2.6 使用反应表达式减少重复

仔细观察上面的这个完整的代码，在server这个函数里面 `dataset <- get(input$dataset, "package:datasets")` 出现了两次。不管在什么编程语言里面，重复的代码都是不明智的：浪费计算资源、调试代码、维护代码成本等，虽然在我们的这个小案例里面没有那么夸张。但是还是要传递一个基本的概念。

在传统的R代码里面，我们使用两种方法处理需要重复计算的部分：将其保存到变量里面、使用函数获得计算结果。具体原因将在18.2中说明。但是这些在shiny的反应式编程里面都不适用。我们需要一个新的机制：反应表达式。

可以使用 `reactive({})` 这个代码块来创建一个反应表达式，并将其赋值给一个变量，然后就可以像是使用函数一样调用这个反应表达式。反应表达式有个特点：一经过运行之后，会将数据缓存，直到更新。

下面的代码和上面的代码基本上没有区别，但是减少了计算量，工作效率更高。因为只用 `get` 这行代码只用运行一次了。

```
library(shiny)
ui <- fluidPage(
  selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),
  verbatimTextOutput("summary"),
  tableOutput("table")
)

server <- function(input, output, session) {
  dataset <- reactive({
    get(input$dataset, "package:datasets")
  })

  output$summary <- renderPrint({
    summary(dataset())
  })

  output$table <- renderTable({
    dataset()
  })
}

shinyApp(ui, server)
```

我们在以后会更加的介绍反应式编程。到目前为止，虽然我们知道有限的输入、输出、浅显的反应式，但是我们依然可以做出有用的shiny app。

2.7 速查表

可以看官网链接：<https://github.com/rstudio/cheatsheets/raw/master/shiny.pdf>

或者看我之前的文章：https://mp.weixin.qq.com/s?_biz=MzU3MDkzNjk1OQ==&mid=2247485297&idx=1&sn=a6e21349a4c93a50111511441e88314a&chksm=fce69c61cb911577f7514e6b4b127dc32cf1123a41b48a9be8b473c4a7acd2e309736bda07b1&token=2015185885&lang=zh_CN#rd

微信扫一扫：



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

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