

## 3.1 介绍

现在你已经掌握了基本的app结构，我们将探索细节让shiny app更加有趣。正如上一章节你所见到的，shiny将前端代码（用户见到的）和后端代码（控制着shiny app的运行行为）分开的。在这一章，我们将更加深入探索前端，包括shiny提供HTML格式的输入、输出、布局。

学习更多的前端将让你可以生成更加引人注目的、但是更加简单的shiny app。在下一章，你将会学到更多由shiny后端提供的强大的反应式编程，使得你可创建更多丰富的反应来响应交互。

和之前一样，每次都要加载shiny包：

```
library(shiny)
```

## 3.2 输入

在之前的章节，我们可以看到你使用的一些函数，比如 `sliderInput()`，`selectInput()`，`textInput()`，`numericInput()`。这些函数插到你的UI部分。接下来我们将介绍所有输入的通用结构，并且快速概述shiny内置的输入部分。

### 3.2.1 通用结构

所有的输入函数的第一个参数 `inputId` 都是相同的，这是一个标识符用来连接前端和后端，如果你的UI部分有个输入id叫"name"，那么在server部分使用 `input$name` 来获得它。

这个 `inputId` 有两个限制：

1. 必须是字母、数字、下划线组成的字符串（不能有空格、破折号、句号等特殊的符号）。具体的规则和R里的变量命名规则相同。
2. 必须是独一无二的。如果不是，在server部分就不能使用这个控件。

大多数输入控件的第二个参数叫：`label`。这个是为控件创建一个易于人发现的标签。shiny对这个标签的字符串没有任何要求，这个标签是被展示在前端的。所以注意shiny app的使用者的阅读感受。输入控件的第三个参数是：`value`，尽量设置为默认数值，剩下的参数都是一些输入控件独有的。

当创建一个输入控件的时候，我建议按照位置对输入控件的 `inputId` 和 `label` 设置参数。其他的参数设置变量的时候，要写出对应的参数名字。（这句话啥意思？就是说，输入控件的第一个参数就直接写 `min`，而不用写成 `inputId = min`；第二参数直接写成 `Limit (minimum)`，而不用写成 `label = Limit (minimum)`；剩下的参数，最后都写成 `value = 50`，`min = 0`，`max = 100`，而不是写成 `50`，`0`，`100`。因为前两个参数名字在输入控件里面都是统一的，后面的参数名字不一定统一。）

```
sliderInput("min", "Limit (minimum)", value = 50, min = 0, max = 100)
```

下面的各小节描述了shiny的输入部分，根据这些控件产生数据的类型粗略的分组。目的是给你一个快速的总览，介绍每一个控件的重要参数，而不是尽可能全面的描述每一个输入函数的参数。想要获得更多的细节，可以阅读参考文档

### 3.2.2 文本

收集小数量的文本可以使用 `textInput()`；输入密码可以使用 `passwordInput()`；（隐藏了输入的字符串，所以别人看不到，你要确保密码不会被暴露，除非你接受过代码加密方面的培训，否则我们不建议你使用）

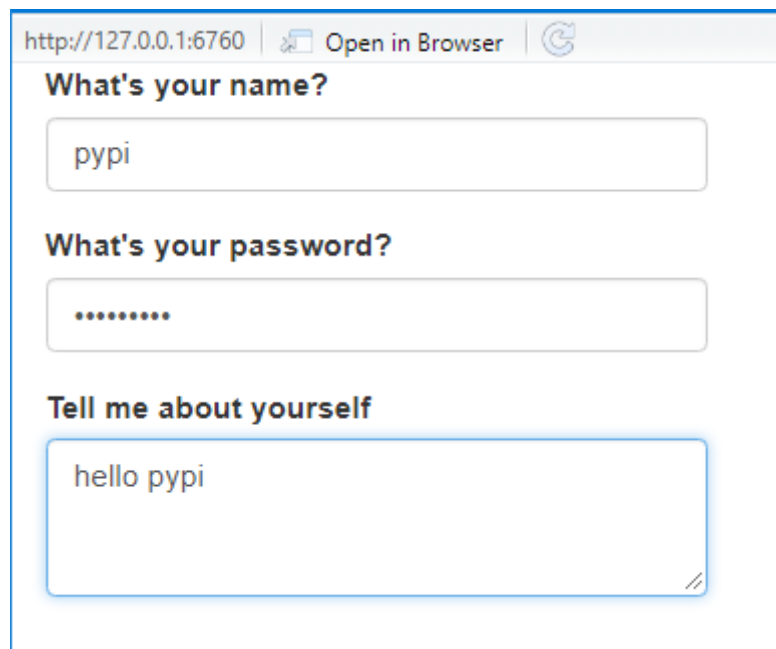
输入段落可以使用 `textAreaInput()`

```
library(shiny)

ui <- fluidPage(
  textInput("name", "What's your name?"),
  passwordInput("password", "What's your password?"),
  textAreaInput("story", "Tell me about yourself", rows = 3)
)

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

shinyApp(ui, server)
```

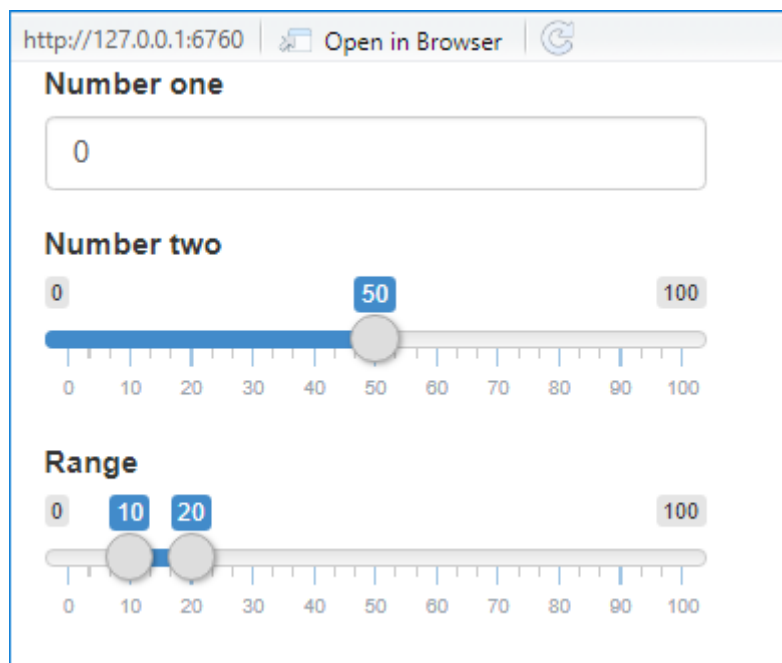


如果希望输入的文本具有某些属性，可以使用 `validate()`。这个函数将在第八章介绍。

### 3.2.3 数值

输入数值，可以使用滑块 `sliderInput()`；或者使用约束的文本框：`numericInput()`；如果希望输入长度为2的向量，可以使用带有两端的滑块：`sliderInput()`。

```
ui <- fluidPage(
  numericInput("num", "Number one", value = 0, min = 0, max = 100),
  sliderInput("num2", "Number two", value = 50, min = 0, max = 100),
  sliderInput("rng", "Range", value = c(10, 20), min = 0, max = 100)
)
```



通常我建议将滑块用在一些精度要求不高场景中，因为将滑块移动的到一个具体的数值是太难了。滑块是可以高度定制的，可以定制外观，具体的可以看：<https://shiny.rstudio.com/articles/sliders.html>

### 3.2.4 日期

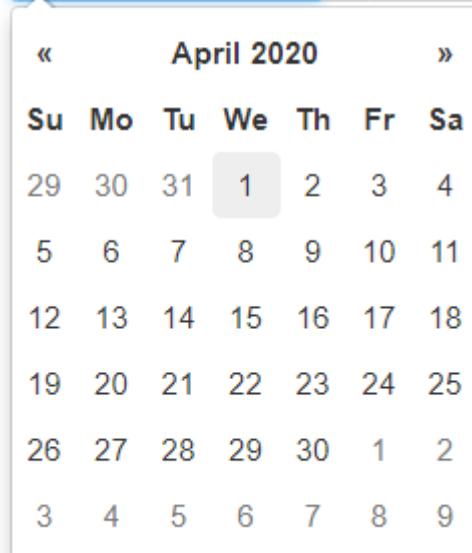
选择一个日期可以使用：`dateInput()`。如果选择日期范围，可以使用：`dateRangeInput()`。这两个函数提供一个可以选择日期的日历表，添加 `datesdisabled` 和 `daysofweekdisabled` 可以限制输入的有效范围。

```
ui <- fluidPage(  
  dateInput("dob", "When were you born?" ),  
  dateRangeInput("holiday", "When do you want to go on vacation next?" )  
)
```

When were you born?

When do you want to go on vacation next?

to



时间格式、语言、星期几默认都是美国标准，如果要做一个全球通用的shiny app。请设置格式，使得时间更加适用于你的用户。

## 3.2.5 选择

允许用户从预定的选项中选择有两种方式：`selectInput()` 和 `radioButtons()`。

```
animals <- c("dog", "cat", "mouse", "bird", "other", "I hate animals")

ui <- fluidPage(
  selectInput("state", "what's your favourite state?", state.name),
  radioButtons("animal", "what's your favourite animal?", animals)
)
```

http://127.0.0.1:6760 | Open in Browser

**What's your favourite state?**

Alabama

**What's your favourite animal?**

☒ dog

☐ cat

☐ mouse

☐ bird

☐ other

☐ I hate animals

单选按钮有两个优秀的特点：将所有的可能都列出来了，这样适用于一些简短的列表，并且可以使用 `choiceNames` 和 `choiceValues` 参数简单文本以外的东西。

```
ui <- fluidPage(
  radioButtons("rb", "Choose one:",
    choiceNames = list(
      icon("angry"),
      icon("smile"),
      icon("sad-tear")
    ),
    choiceValues = list("angry", "happy", "sad")
  )
)
```

http://127.0.0.1:6760 | Open in Brow

**Choose one:**

☒ 😞

☐ 😐

☐ 😊

可以使用 `selectInput()` 创建下拉菜单，不管列表长度有多少，每次下拉的菜单占用的面积都是相同的；还可以设置 `multiple=TRUE` 允许用户选择多个标签。

```
ui <- fluidPage(
  selectInput(
    "state", "what's your favourite state?", state.name,
    multiple = TRUE
  )
)
```

What's your favourite state?

ca|

California

North Carolina

South Carolina

没有办法使用圆形按钮选择多个标签，但是可以使用相似的输入函数：`checkboxGroupInput()`。

```
ui <- fluidPage(  
  checkboxGroupInput("animal", "What animals do you like?", animals)  
)
```

What animals do you like?

☐ dog

☐ cat

☐ mouse

☐ bird

☐ other

☐ I hate animals

如果喜欢一些简单的单选问题 (yes/no)。可以使用：`checkboxInput()`。

```
ui <- fluidPage(  
  checkboxInput("cleanup", "Clean up?", value = TRUE),  
  checkboxInput("shutdown", "Shutdown?")  
)
```

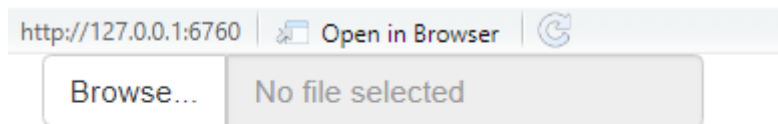
☒ Clean up?

☐ Shutdown?

## 3.2.6 文件上传

允许用户上传文件可以使用函数：`fileInput()`。

```
ui <- fluidPage(  
  fileInput("upload", NULL)  
)
```

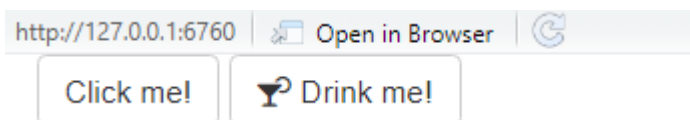


`fileInput()` 这函数在server部分需要特别的处理，具体可以看第九章。

## 3.2.7 动作按钮

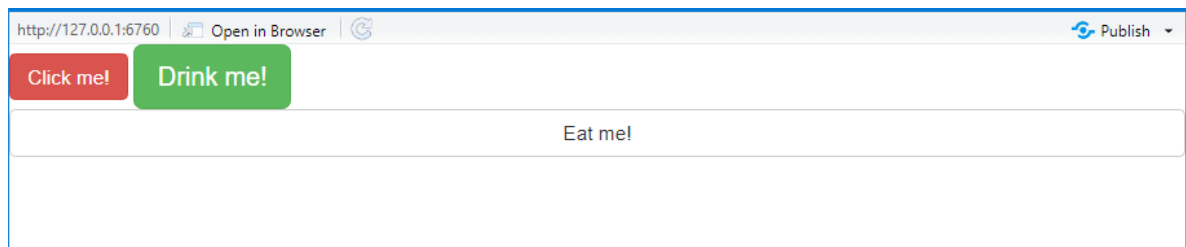
让用户使用 `actionButton()` 或者 `actionLink()`。这函数本来就是和 `observeEvent()` 或者 `eventReactive()` 在server部分成对出现的。我到现在还没讨论他们，他们将在下一章出现。

```
ui <- fluidPage(  
  actionButton("click", "Click me!"),  
  actionButton("drink", "Drink me!", icon = icon("cocktail"))  
)
```



你可以使用这些函数的 `class` 参数来自定义他们的外观。设置值为 `btn-primary`, `btn-success`, `btn-info`, `btn-warning` 或者 `btn-danger`。同样的可以设置他们的大小使用 `btn-lg`, `btn-sm`, `btn-xs`。最后，可以使用 `btn-block` 使按钮跨越元素的整个宽度。这些都是基于CSS的，想要了解更多，可以看这个链接：<http://bootstrapdocs.com/v3.3.6/docs/css/#buttons>

```
library(shiny)  
  
animals <- c("dog", "cat", "mouse", "bird", "other", "I hate animals")  
  
ui <- fluidPage(  
  fluidRow(  
    actionButton("click", "Click me!", class = "btn-danger"),  
    actionButton("drink", "Drink me!", class = "btn-lg btn-success")  
  ),  
  fluidRow(  
    actionButton("eat", "Eat me!", class = "btn-block")  
  )  
)  
server <- function(input, output, session) {  
  
}  
  
shinyApp(ui, server)
```



## 3.2.8 练习

# 3.3 输出

输出在UI部分创建一个占位符，随后由server函数填充，就像是输入一样，输出占有一个独一无二的id在输出函数里面，如果你的UI部分创建一个输出id叫"plot"。那么在server部分就是使用output\$plot。

前端的每一个输出都与后端的渲染函数耦合。有三种输出类型，对应于报表中的：文字、表格、图表。下面各节将介绍前端的输出的基本函数，以及相应的后端渲染函数。

## 3.3.1 文本

输出的是常规的文本使用：`textOutput()`。输出固定的代码或者控制台的结果使用：``verbatimTextOutput()`。

```
ui <- fluidPage(
  textOutput("text"),
  verbatimTextOutput("code")
)
server <- function(input, output, session) {
  output$text <- renderText({
    "Hello friend!"
  })
  output$code <- renderPrint({
    summary(1:10)
  })
}
```

Hello friend!

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.00	3.25	5.50	5.50	7.75	10.00

注意上面的代码，在render\*函数里面需要`{}`。除非你的代码只有一行。你可以写更加紧凑的server函数，我认为这是一个很好的风格。因为你在渲染函数里面尽量减少计算。

```
server <- function(input, output, session) {
  output$text <- renderText("Hello friend!")
  output$code <- renderPrint(summary(1:10))
}
```

注意有两个渲染函数可以与任意文本输出函数一起使用：

1. `renderText()` 可以显示代码返回 (return) 的结果
2. `renderPrint()` 可以显示代码打印的结果



为了理解上面的区别，可以看下面的函数，打印出 `a`, `b`。但是返回 (return) `c`。一个代码可以打印多个结果，但是只能有一个返回 (return)值。

```
print_and_return <- function() {  
  print("a")  
  print("b")  
  "c"  
}  
x <- print_and_return()  
#> [1] "a"  
#> [1] "b"  
x  
#> [1] "c"
```

### 3.3.2 表格

有两个选项用表格显示数据框：

1. `tableOutput()` 和 `renderTable()` 渲染的是静态的数据。一次性显示所有数据。
2. `dataTableOutput()` 和 `renderDataTable()` 渲染的是动态数据，可以使用按钮来控制显示一定数量的行。

`tableOutput()` 通常被用来显示小的、固定统计表（比如模型的系数）。`dataTableOutput()` 通常向用户公开完整的数据框。

```
ui <- fluidPage(  
  tableOutput("static"),  
  dataTableOutput("dynamic")  
)  
server <- function(input, output, session) {  
  output$static <- renderTable(head(mtcars))  
  output$dynamic <- renderDataTable(mtcars, options = list(pageLength = 5))  
}
```

<http://127.0.0.1:6760>
[Open in Browser](#)
[Publish](#)

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.00	6.00	160.00	110.00	3.90	2.62	16.46	0.00	1.00	4.00	4.00
21.00	6.00	160.00	110.00	3.90	2.88	17.02	0.00	1.00	4.00	4.00
22.80	4.00	108.00	93.00	3.85	2.32	18.61	1.00	1.00	4.00	1.00
21.40	6.00	258.00	110.00	3.08	3.21	19.44	1.00	0.00	3.00	1.00
18.70	8.00	360.00	175.00	3.15	3.44	17.02	0.00	0.00	3.00	2.00
18.10	6.00	225.00	105.00	2.76	3.46	20.22	1.00	0.00	3.00	1.00

Show 
Search:

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21	6	160	110	3.9	2.62	16.46	0	1	4	4
21	6	160	110	3.9	2.875	17.02	0	1	4	4
22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360	175	3.15	3.44	17.02	0	0	3	2

mpg

cyl

disp

hp

drat

wt

qsec

vs

am

gear

carb

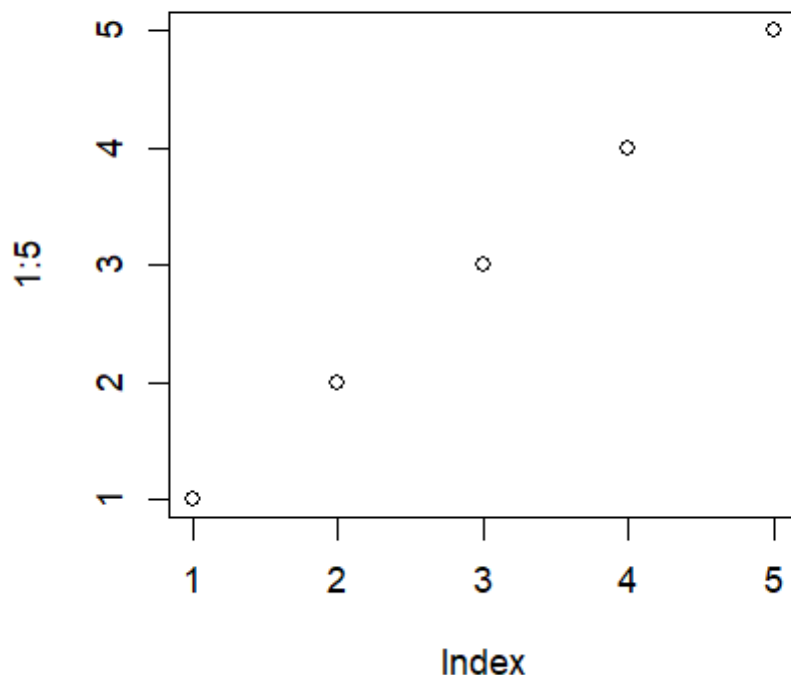
Showing 1 to 5 of 32 entries

Previous
1
2
3
4
5
6
7
Next

### 3.3.3 图表

`plotOutput()` 和 `renderPlot()` 可以显示R的图形（base系统、ggplot2系统等）。

```
ui <- fluidPage(
  plotOutput("plot", width = "400px")
)
server <- function(input, output, session) {
  output$plot <- renderPlot(plot(1:5), res = 96)
}
```



默认情况下，`plotOutput()` 将占据其容器的整个宽度（稍后介绍），并且高度是400px。你可以使用 `height` 和 `width` 参数来覆盖。我们建议始终将 `res = 96`，这样使你的shiny输出的图和Rstudio中的图相匹配。

图表是特别的，因为不仅可以作为输入，还可以作为输出。`plotOutput()` 有一系列的像 `click`、`dblclick` 和 `hover` 参数，如果传递一个字符串，如：`click = "plot_click"`，这将创建一个反应式输入（`input$plot_click`）。可用于处理绘图上的用户交互。我们将在以后介绍（具体哪一章还没介绍）。

### 3.3.4 下载

你可以允许用户下载文件使用 `downloadButton()` 和 `downloadLink()`。这些需要一些server里面的技术，将在第九章详细介绍。

### 3.3.5 练习

## 3.4 布局

你现在知道一系列的输入输出函数，你需要能改在前端对他们进行重新排列。这就是布局函数的用处，提供shiny app的高级视觉结构，在这里，我们将重点介绍 `fluidPage()`。这个函数提供了shiny app布局的样式，在以后的章节，将了解像 `dashboards` 和 `dialog box` 样式的布局系列。

### 3.4.1 总览

布局由调用函数的产生层次构建。R中的层次结构和输出的层次结构相匹配。（这句话就是说，shiny app的一些布局可以由布局函数来创建，使用了哪些布局函数，就会在ui端出现对应的布局）。当你看到下面的复杂的函数：

```
fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs", "Observations:", min = 0, max = 1000, value = 500)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

首先着重介绍函数调用的层次结构：

```
fluidPage(
  titlePanel(),
  sidebarLayout(
    sidebarPanel(
      sliderInput("obs")
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
```

虽然不知道函数的任何作用，但是看这个函数的名字就能猜出来这个shiny app长啥样。你可以想象这个代码是一个经典的shiny app布局。顶端是标题栏，包含滑块的侧边栏，包含绘图的主面板。

<http://127.0.0.1:6760>
[Open in Browser](#)
[Publish](#)

# Hello Shiny!

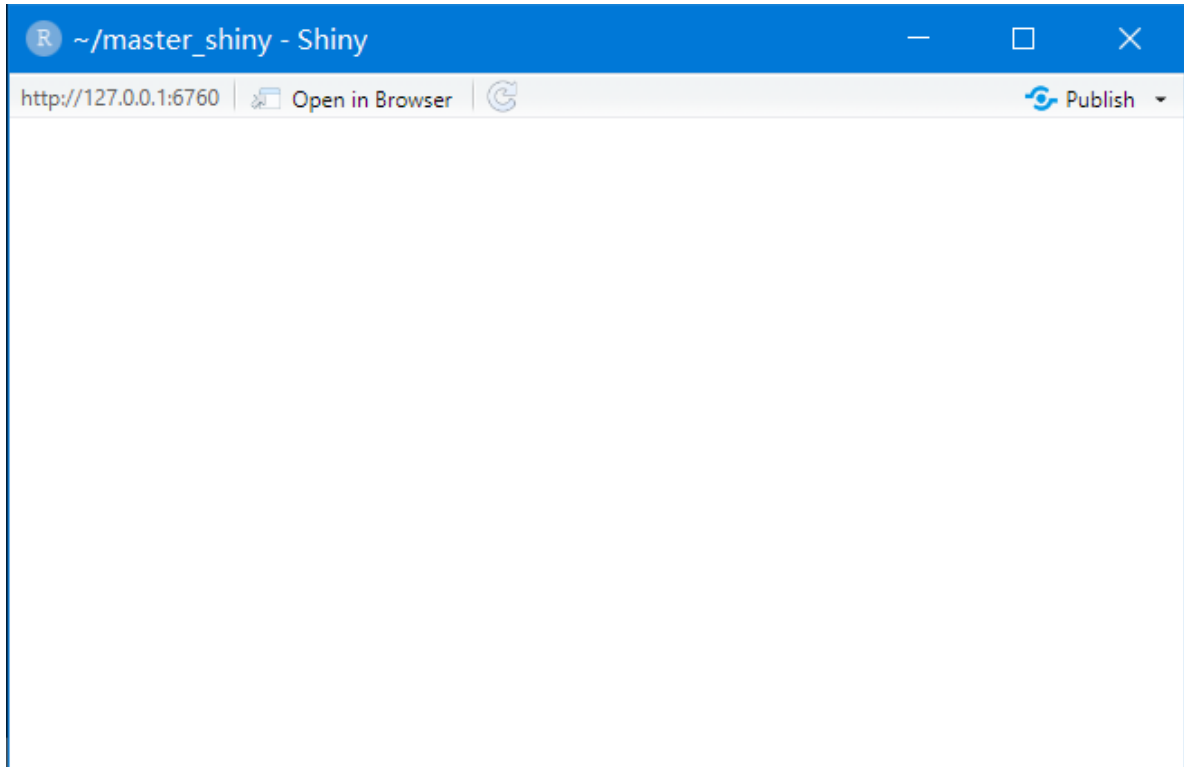
**Observations:**

0 500 1,000

0 100 200 400 600 800 1,000

## 3.4.2 页面功能

最重要的，但是最无趣的函数就是 `fluidPage()`。在上面的例子，你已经看到了，不管你的shiny app需要多少个输入和输出，都需要放入 `fluidPage()` 函数中。如果只使用 `fluidPage()` 会发生什么情况？



这个看起来非常无聊，因为这个页面内容都没有。但是在这个页面的背后，`fluidPage()` 做了大量的工作，这个页面函数为shiny app做了一系列的必要准备：HTML、CSS、JS。`fluidPage()` 使用的布局系统为Bootstrap系统 (<https://getbootstrap.com/>)。（目前shiny使用的是Bootstrap的版本为3.3.7。不久的，Rstudio将更新shiny使用的Bootstrap的版本到4.0.0）

我们将在未来的一些章节讨论如何使用Bootstrap的一些知识来控制shiny app的外观。使用你的shiny app看起来更加的美观，与公司风格指南更加匹配。

从技术上来说，`fluidPage()` 是shiny app所需要的全部，因为你可以将你的所有的输入、输出控件直接放入其中。虽然这对于学习shiny的基础知识是不错的选择，但是将所有的输入、输出都集中在一起看起来并不舒服，因此你需要学习更多的布局。在这里，介绍两个常见的结构，一个是带有侧边栏的页面，一个是有多行的页面。

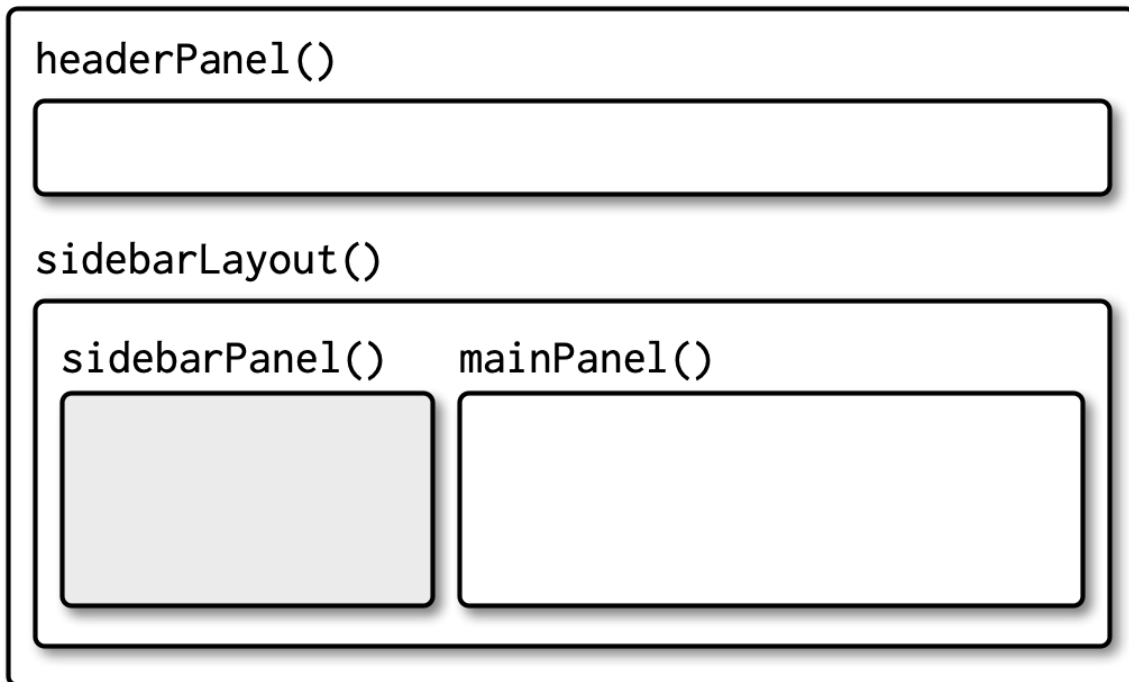
### 3.4.3 带有侧边栏的页面

`sidebarLayout()`，和 `titlePage()`，`sidebarPage()`，`mainPanel()`，可以建议的创建两列布局，其中输入在左边的侧边栏，右边是输出框。基础的Rcode展示如下：

```
fluidPage(  
  titlePanel(  
    # app title/description  
  ),  
  sidebarLayout(  
    sidebarPanel(  
      # inputs  
    ),  
    mainPanel(  
      # outputs  
    )  
  )  
)
```

布局效果如下：

## fluidPage()



下面的代码展示了如何使用上面的代码创建一个简单的shiny app用来演示中心极限定理，动手运行一下这个代码，就可以看到随着增加数量，数据趋向于正太分布。

```
library(shiny)

ui <- fluidPage(
  titlePanel("Central limit theorem"),
  sidebarLayout(
    sidebarPanel(
      numericInput("m", "Number of samples:", 2, min = 1, max = 100)
    ),
    mainPanel(
      plotOutput("hist")
    )
  )
)

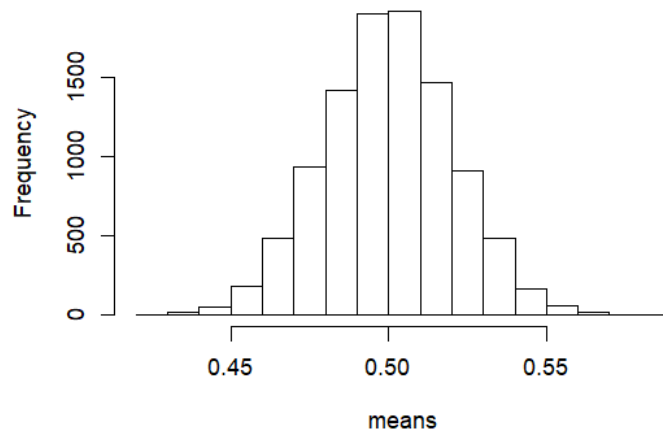
server <- function(input, output, session) {
  output$hist <- renderPlot({
    means <- replicate(1e4, mean(runif(input$m)))
    hist(means, breaks = 20)
  }, res = 96)
}

shinyApp(ui, server)
```

# Central limit theorem

Number of samples:

Histogram of means

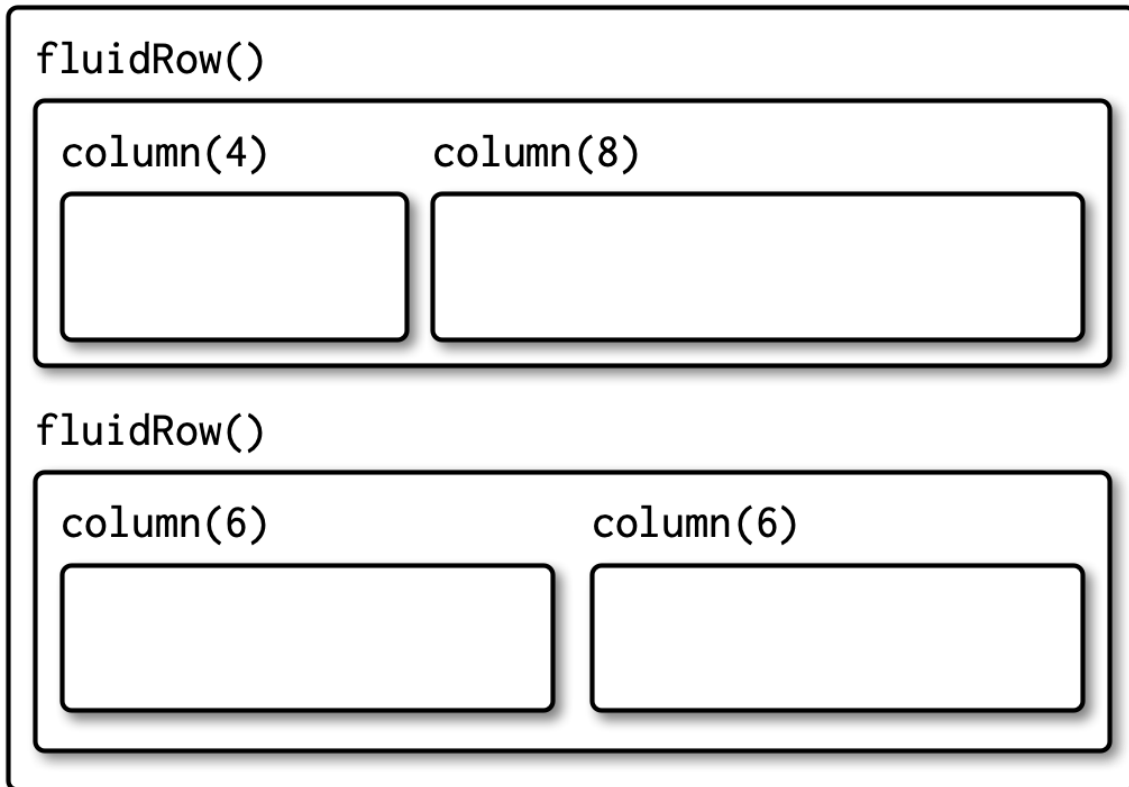


## 3.4.4 多行

从shiny app底层来说, `sidebarLayout()` 建立在灵活的多行布局上, 可以直接用来做更加好看的、复杂的shiny app。和之前一样, 都需要 `fluidPage()`。然后使用 `fluidRow()` 和 `columns()` 创建列。下面是代码模板:

```
fluidPage(  
  fluidRow(  
    column(4,  
      ...  
    ),  
    column(8,  
      ...  
    )  
  ),  
  fluidRow(  
    column(6,  
      ...  
    ),  
    column(6,  
      ...  
    )  
  )  
)  
)
```

fluidPage()



注意 `column()` 的第一个参数是宽度，每一块的宽度加起来必须等于12。因此你可以轻松的设置2列，3列，4列等（只要他们的宽度加起来等于12就行）。

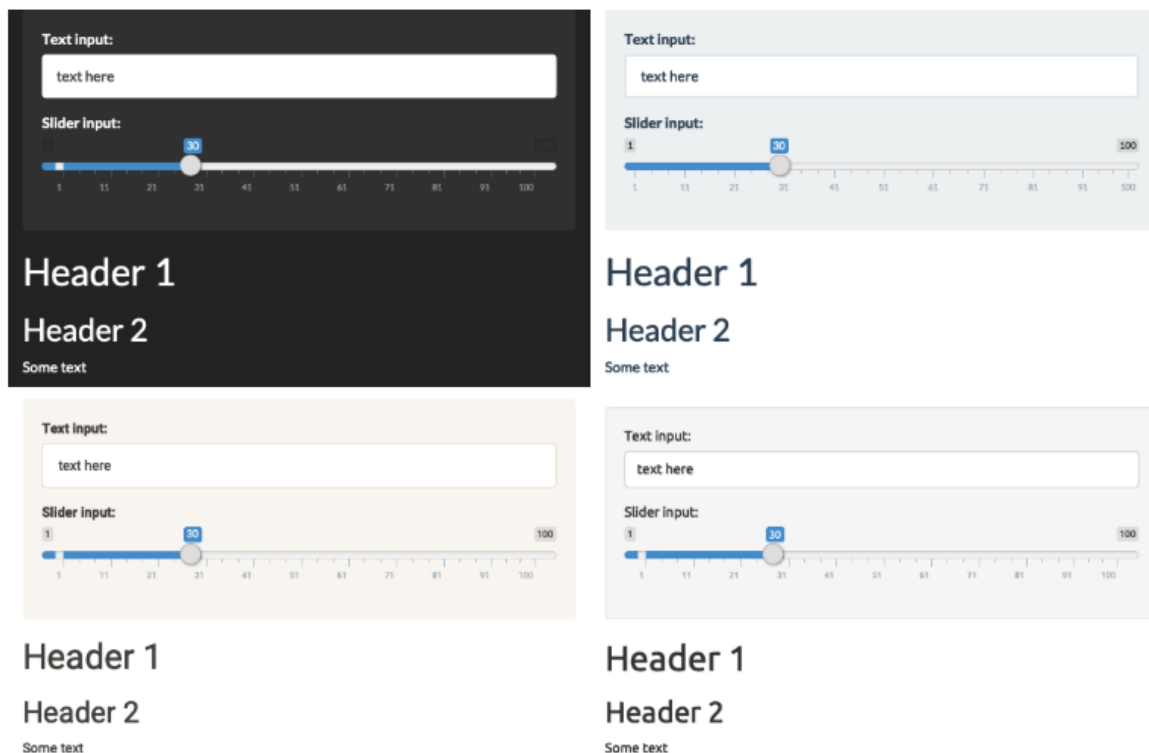
### 3.5.5 主题

在未来的章节里面，我们将详细的介绍shiny的一些主题（你看到的UI的部分，美化shiny的UI）。创建一个复杂的主题将花费大量的时间，但是值得去做。但是也可以使用 `shinythemes` 包来快速获得一些成型的主题。代码如下：

```
theme_demo <- function(theme) {  
  fluidPage(  
    theme = shinythemes::shinytheme(theme),  
    sidebarLayout(  
      sidebarPanel(  
        textInput("txt", "Text input:", "text here"),  
        sliderInput("slider", "Slider input:", 1, 100, 30)  
      ),  
      mainPanel(  
        h1("Header 1"),  
        h2("Header 2"),  
        p("Some text")  
      )  
    )  
  )  
}
```

theme\_demo("darkly")  
theme\_demo("flatly")  
theme\_demo("sandstone")  
theme\_demo("united")





### 3.4.6 练习

## 3.5 内置的机制

在前面的案例中，你可能惊讶于我可以函数创建shiny。`theme_demo()` 函数可以运行是因为shiny的代码就是R的代码。你可以使用现有的工具减少重复工作。记住一个准则：如果一个代码复制粘贴超过3次，你就应该考虑写个函数或者使用循环（也可以使用 `purrr::map()` 系列或者 `lapply()` 系列）。

所有的输入、输出、布局返回的都是HTML。HTML支撑着每一个网站。你可以直接在控制台运行UI函数来查看HTML：

```
fluidPage(  
  textInput("name", "what's your name?")  
)
```

```
<div class="container-fluid">  
  <div class="form-group shiny-input-container">  
    <label for="name">what's your name?</label>  
    <input id="name" type="text" class="form-control" value=""/>  
  </div>  
</div>
```

这就是shiny的设局原理，作为shiny的用户，不需要知道HTML的具体知识。当然如果你早就了解HTML。你可以直接使用HTML对任何标签进行自定义。shiny对此不排斥。可以将高级功能和低级的HTML组合使用。将在第22章详细介绍这些内容，你将学到更多关于直接编写HTML的低级功能。

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](https://github.com/yuanzhoulvpi2017/master_shiny_CN)

如果有错误，欢迎指正，邮箱联系我: [yuanzhoulvpi@outlook.com](mailto:yuanzhoulvpi@outlook.com)