# R 语言编程：基于 tidyverse

## 第 27 讲　R Shiny

张敬信

2022 年 3 月 27 日

哈尔滨商业大学

Shiny 包，可以轻松从 R 直接构建交互式 Web 应用，可以在网页上托管独立 app，也可以将其嵌入 R Markdown 文档中或构建 dashboards（仪表盘）。还可以使用 CSS 主题，htmlwidgets（网页部件）和 JavaScript 操作扩展 Shiny app.

Shiny 主要是为数据科学家设计的，让您可以在没有 HTML、CSS 或 JavaScript 知识的情况下创建相当复杂的 Shiny app.

Shiny 扩展了基于 R 的分析，通过将 R 代码和数据包装成一个额外的互动层，以更好地进行可视化、分析、输出等。这提供了一种强大的方式，使得任何用户[1]都可以与数据进行互动、探索和理解数据。

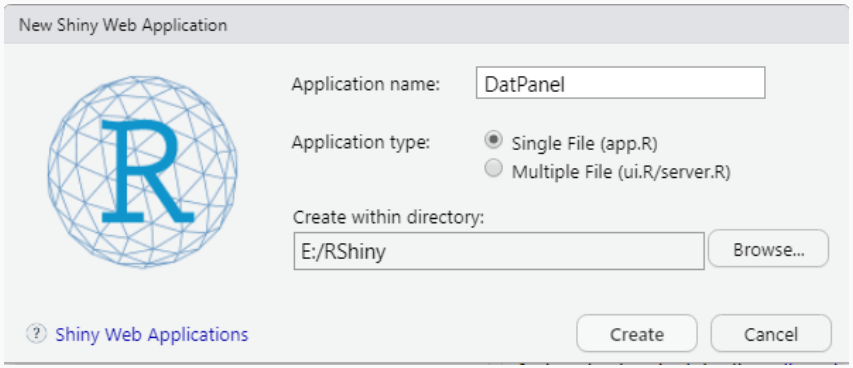
Shiny app 应用场景：

- 开发设计辅助教学工具，让学生交互式探索统计学方法或模型
- 设计动态数据分析报表或仪表盘，给同事领导以交互式的结果呈现

[1]甚至是非 R 用户！

# 一. Shiny 基本语法

在 RStudio 创建 Shiny app. 点击 New File -> Shiny Web App...




选择路径，输入 app 名字，点击 Create. 则在该路径下创建同名的文件夹，里面有一个 app.R 文件。

该文件就是一个简单的 Shiny app 的模板，点击 Run App 按钮或用 Ctrl + Shift + Enter 运行，则生成 app 并在浏览窗口打开该，点击 Open in Browser 可在浏览器打开。

# 1. Shiny app 基本结构

每个 Shiny app 的 app.R 都具有同样的结构：

```
library(shiny)
# 定义 UI
ui = fluidPage(
  ...
)
# 定义 server 逻辑
server = function(input, output) {
    ...
}
# 运行 app
shinyApp(ui = ui, server = server)
```

由三部分构成：**ui（用户界面）、server（服务器）和 shinyApp()（接受 ui 和 server 对象，并运行 app）**。

Shiny 设计为将 Web app 的前端和后端组件分开：

- ui 代表“用户界面”，定义了用户看到并与之交互的前端控件，如图、表、滑块、按钮等。
- Server 负责后端逻辑：接受用户输入，并使用这些输入来定义应该发生什么样的数据转换，以及将什么传回前端以供用户查看和交互。

# (1) ui (前端)

ui 定义了用户与 Shiny app 交互时所能看到的东西。ui 通常用来设置：

- 用户界面的布局，为输入和输出安排位置
- 输入控件，允许用户向 server 发送命令
- 来自 server 的输出

定义 ui 之前，可以根据需要加载其他包、读入数据、定义函数等。

关于页面布局，先用函数 `fluidPage()` 创建整体页面布局，其内常用的两种布局是：

- **侧边面板 + 主面板：**用 `titlePanel()` 创建标题面板，用 `sidebarLayout()` 创建带侧边栏的布局，其内又常包括 `sidebarPanel()` 侧边面板和 `mainPanel` 主面板，主面板内还可以继续用 `tabPanel()` 创建标签面板；
- **直接按行列布局：**用 `fluidRow()` 控制若干控件属于一行，一行的宽度是 12 个单位，其内再用 `column()` 划分列宽，再将输入或输出置于其中[2]。

[2] `mainPanel` 主面板中可以用与 HTML5 标签一致的函数控制文本格式，比如 `h3()` 是三级标题和 `br()` 是空一行，`strong()` 是加粗，`em()` 是强调斜体，`code()` 是代码格式等.

Shiny 提供了一系列内置的控件，每个控件都用同名的函数创建。例如，actionButton 函数创建动作按钮，sliderInput 函数创建滑动条。

```
library(shiny)
# 定义 UI
ui = fluidPage(
 titlePanel(" 常用控件"),
  fluidRow(
   column(3, h3(" 按钮"),  #
          actionButton("action", " 点击"),
          br(), br(),
          submitButton(" 提交")),
   column(3, h3(" 单选框"),
      checkboxInput("checkbox", " 选项 A", value = TRUE)),
```

```
  column(3,
    checkboxGroupInput("checkGroup", h3(" 多选框"),
      choices = list(" 选项 1" = 1, " 选项 2" = 2,
                     " 选项 3" = 3),
      selected = 1)),
 column(3, dateInput("date", h3(" 输入日期"),
                     value = "2021-01-01"))),
fluidRow(
  column(3, dateRangeInput("dates", h3(" 日期范围"))),
  column(3, fileInput("file", h3(" 文件输入"))),
  column(3, h3(" 帮助文本"),
         helpText(" 注: 帮助文本不是真正的部件, 但提供了",
                  " 一种易于实现的方式为其他部件添加文本.")),
  column(3, numericInput("num", h3(" 输入数值"), value=1))),
```

```
fluidRow(
  column(3, radioButtons("radio", h3(" 单选按钮"),
                         choices = list(" 选项 1" = 1,
                                        " 选项 2" = 2,
                                        " 选项 3" = 3),
                         selected = 1)),
  column(3, selectInput("select", h3(" 下拉选择"),
                         choices = list(" 选项 1" = 1,
                                        " 选项 2" = 2,
                                        " 选项 3" = 3),
                         selected = 1)),
```

```
    column(3, sliderInput("slider1", h3(" 滑动条"),
                          min = 0, max = 100, value = 50),
              sliderInput("slider2", "",
                          min = 0, max = 100,
                          value = c(25, 75))),
    column(3, textInput("text", h3(" 文本输入"),
                        value = " 输入文本..."))))
)
# 定义 server 逻辑:
# 空白逻辑是 app 对控件的输入什么都不做, 不产生任何输出
server = function(input, output) {}
# 运行 app
shinyApp(ui = ui, server = server)
```

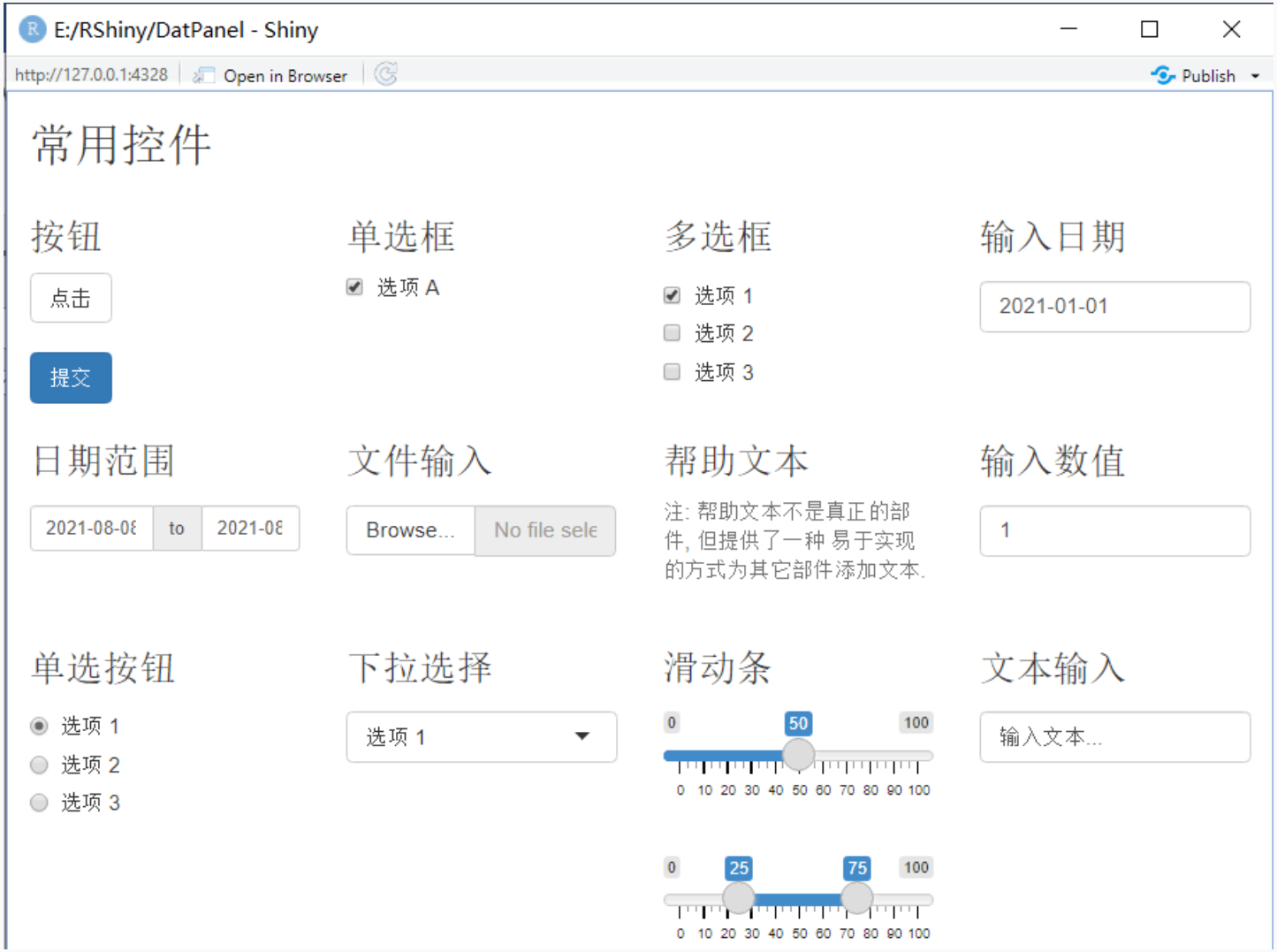



**图 1:** Shiny 常用控件面板

# (2) server (**后端**)

用户通过键盘/鼠标操作 ui 上输入控件，就会改变输入，后端 server 一旦接受到一组新的输入，就立马解读输入，对数据进行处理，并创建输出，再将输出送回 ui，用户就会看到交互的结果。

ui 很简单，因为面向每个用户的都是相同的用户界面；真正复杂的是设计 server(), 因为所有的后端处理、响应计算、交互逻辑都在它里面完成。

server() 函数有 3 个参数[3]：input, output, session, 它们是在会话开始时由 Shiny 创建的，连接到一个特定的会话。

ui 和 server 是分开设计的，这就需要把它们中的输入、输出联系起来。

[3]一般使用，只关心前两个即可.

```r
library(shiny)
ui = fluidPage(
    textInput("name", " 请输入您的姓名: "),
    textOutput("greeting")
)
server = function(input, output, session) {
    output$greeting = renderText({
        paste0(" 您好 ", input$name, "! ")
    })
}
shinyApp(ui = ui, server = server)
```

运行结果非常简单：输入姓名，输出您好姓名！

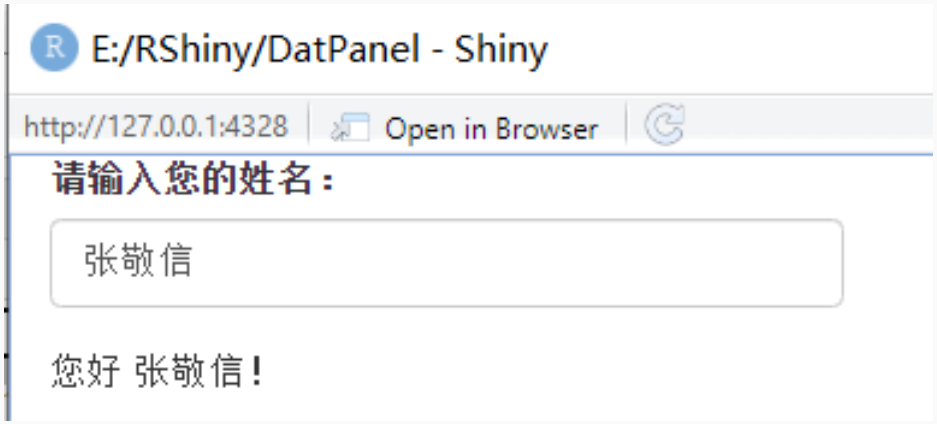



**图 2:** 简单的 Shiny 姓名交互

- 关于输入 input

input 是类似列表的对象，在 ui 中定义输入时，都提供一个该输入的 ID，该 ID 将会内化为 input 同名的一个成分，同时将接受到的用户输入作为该成分的内容。

比如，textInput("name", " 请输入您的姓名：") 定义了一个 ID 为"name" 的文本输入，则 input 对象自动生成一个名为"name" 的成分：input$name，它将接受用户交互时输入的文本作为其内容。

- 关于输出 output

output 是类似列表的对象，在 ui 中定义输出时，都提供一个该输出的 ID，该 ID 将会内化为 output 同名的一个成分，将用来存放随后渲染输出函数生成的输出。

比如，textOutput("greeting") 定义了一个 ID 为"greeting" 的文本输出，则 output 对象自动生成一个同名的成分：output$greeting，随后被赋值为 renderText() 生成的输出。

在 ui 中用" 定义输出函数" 定义的每一个某类型的输出，都在 server() 中有一个对应的" 渲染输出函数" 来生成该类型的输出。

本例中，在 ui 中用” 定义输出函数”textOutput("greeting") 定义了一个文本输出，其 ID 为”greeting”, 则在 server() 中就有一个与它对应的” 渲染输出函数”renderText() 来生成文本输出，再赋值给 output 与 ID 同名的成分。

**注：** rendText() 函数中的一对 { } 是将多行代码打包成一个整体。

Shiny 支持渲染多种类型的输出对象，常用的 ui 中” 定义输出” 的函数，与 server() 中相对应” 渲染输出” 的函数：

**表 1:** Shiny 输出对象

| ui **定义输出函数** | server **渲染输出函数** | **输出对象** |
|---|---|---|
| `DT::dataTableOutput` | `DT::renderDataTable` | 数据表 |
| `imageOutput` | `renderImage` | 图片 (文件链接) |
| `plotOutput` | `renderPlot` | R 图形 |
| `plotly::plotlyOutput` | `plotly::renderPlotly` | 交互 R 图形 |
| `tableOutput` | `renderTable` | 表格 |
| `textOutput` | `renderText` | 文本 |
| `verbatimTextOutput` | `renderText` | 固定宽度文本 |
| `uiOutput` | `renderUI` | Shiny 标签或 HTML 网页 |

# 二. 响应表达式

命令式编程：你发出一个具体的命令，它就会立即执行。但这不适用于 Shiny app.

以刚才的问候 app 为例，server() 中的核心代码：

```
output$greeting = renderText({
  paste0(" 您好 ", input$name, "!")
})
```

这不是简单地把" 您好" 与姓名拼接再发送给 output$greeting, 想想看：

> **您只发出一次指令，但是 Shiny 在用户每次更新 input$name 的时候都会执行这个动作!**

代码并没有告诉 Shiny 创建字符串并将其发送给浏览器，而是告知 Shiny 如果需要，它可以如何创建字符串。至于何时（甚至是否）运行该代码，则由 Shiny 决定。决定何时执行代码是 Shiny 的责任，而不是你的责任。把你的 app 看作是为 Shiny 提供配方，而不是给他命令。

这是一种**声明式编程**，优势之一是它允许 app 非常懒惰：一个 Shiny App 将只做输出控件所需的最小量的工作。

**注：**这也造成了 Shiny 代码不再是从上到下的顺序执行，所以，要理清 Shiny 代码的执行顺序，更重要的是自己开发 Shiny app 时，绘制响应图是非常有必要的!

Shiny app 交互时，输入控件的输入一发生改变，Shiny 就要做出响应：重新计算、生成输出、发送给 ui. 这就对 app 运行效率要求很高，所以就非常需要避免不必要的重复计算。Shiny 中有一种非常重要的机制，叫作**响应表达式**，就是专用于此的。

**响应表达式**与函数类似，是使用控件输入完成相应计算并返回值的 R 表达式。每当控件更改时，响应式表达式都会更新返回值。

**响应表达式**比函数更聪明：

- 响应表达式在首次运行时会保存其结果
- 下次调用响应式表达式时，它将检查保存的值是否已过期（即其依赖的控件输入是否已更改）
- 若该值已过期，则响应对象将重新计算它（然后保存新结果）
- 如果该值是最新的，则响应表达式将返回保存的值，而不进行任何计算（从而提高 app 运行效率）

用 `reactive()` 函数创建响应表达式，响应表达式通常是多行代码构成，所以需要用大括号括起来。使用响应表达式的返回结果，类似调用无参数函数：*名字*`()`.

在 Shiny app 的制作中，要尽可能地把交互计算提取出来，作为响应表达式。

以演示**中心极限定理**为例：

设 $X_1, \cdots, X_n$ 为任意期望为 $\mu$, 方差为 $\sigma^2$（有限）分布的抽样，则当 $n$ 足够大时，$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ 近似服从 $N\left(\mu, \frac{\sigma^2}{n}\right)$.

# (1) 设计想要做哪些交互、怎么交互

- 让用户有几种分布可以选择，下拉选项输入
- 让用户可以改变随机变量个数，滑动条输入
- 让用户可以改变每个随机变量数据量，滑动条输入
- 对样本均值数据绘制直方图，图形输出
- 让用户可以改变直方图的条形数，滑动条输入

## (2) 绘制响应图

响应图是描述输入和输出的连接方式的一种图形，绘制响应图（草图）是制作或理解他人 Shiny app 的好用工具，本例的响应图如下：

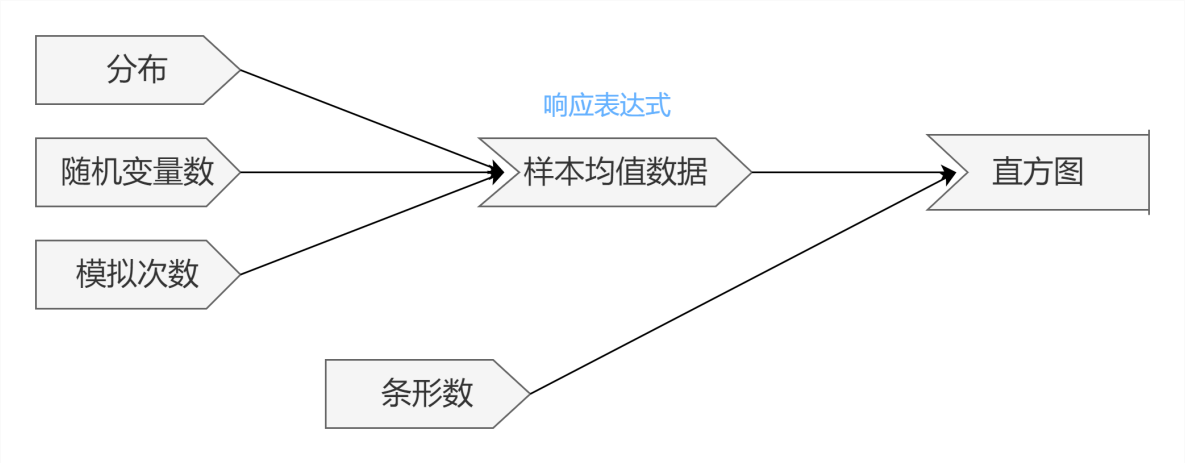



**图 3:** 设计响应图

## (3) 定义 ui: 侧边栏放置输入控件，主面板输出直方图

```
ui = fluidPage(
  titlePanel(" 演示中心极限定理"),
  sidebarLayout(position = "right",   # 放到右侧
    sidebarPanel(
      selectInput("distr", " 分布: ",
                  c(" 均匀", " 二项", " 泊松", " 指数")),
      sliderInput("samples", " 随机变量数: ", 1, 100, 10,
                  step = 1),
      sliderInput("nsim", " 模拟样本量: ", 1000, 10000, 1000,
                  step = 100),
      sliderInput("bins", " 条形数", min = 10, max = 100,
                  value = 50),
      helpText(" 说明: 从下拉选项选择分布, 并用滑动条选择
               随机变量数和模拟样本量.")),
    mainPanel(plotOutput("plot"))))
```

## (4) 定义 server()

根据响应图，需要实现从输入到生成 $\bar{X}$ 的样本数据，并放入响应表达式。实际上这与自定义函数是一样的，除了把”函数”外形以及参数多一步从 input 取出来。

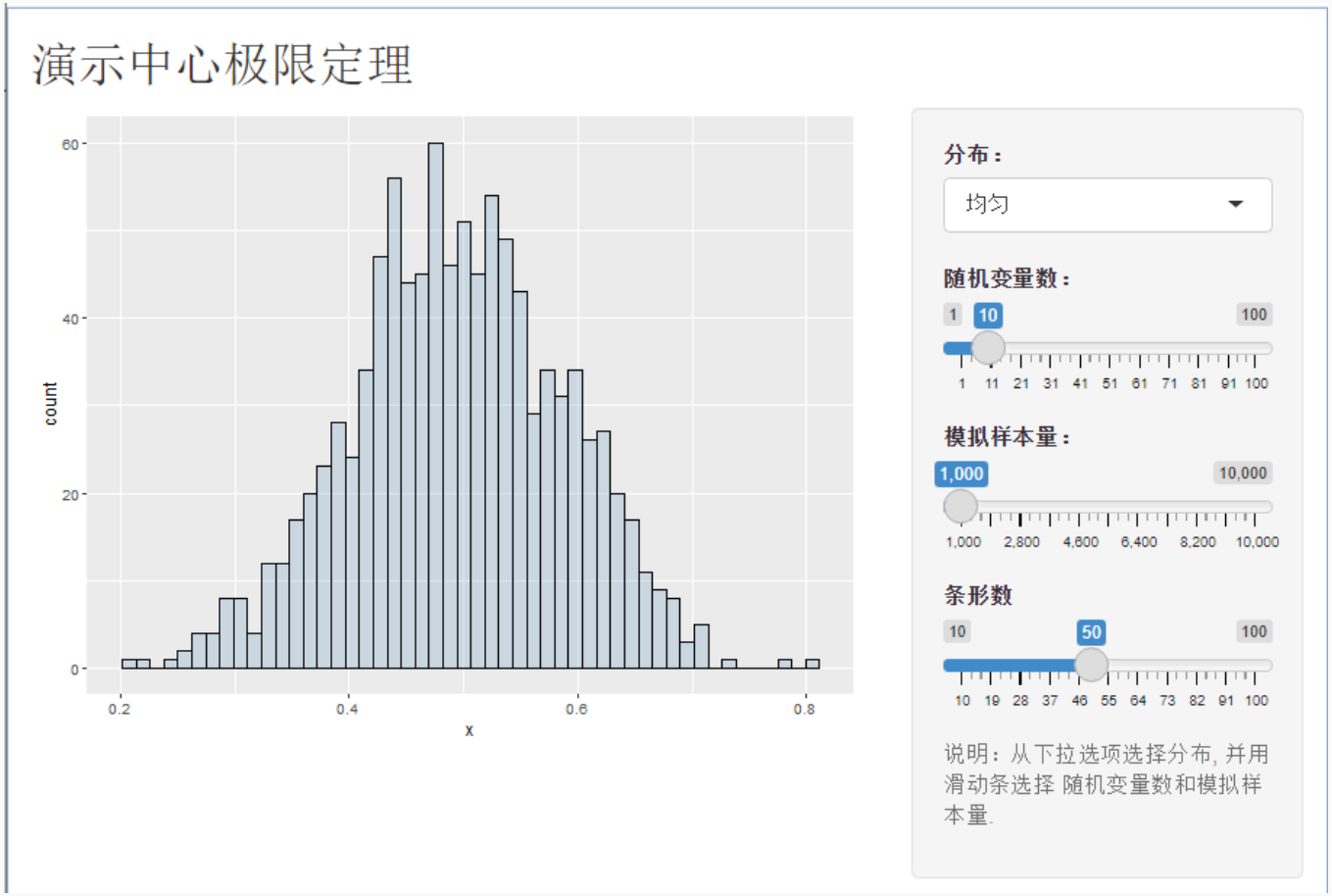
从 input 取出 3 个输入：sample（随机变量个数）、nsim（模拟样本数）、distr（分布），利用 switch() 根据分布名生成随机数，采用一次全部生成再分配给各个随机变量（矩阵），按行取平均得到样本均值的样本，再定义成数据框方便用于 ggplot 绘图。

相应表达式命名为 Xbar，故在 renderPlot() 中使用时用的是 Xbar().

```
server = function(input, output) {
    Xbar = reactive({
        n = input$samples        # 随机变量个数
        m = input$nsim           # 模拟样本量
        xs = switch(input$distr,
                    " 均匀" = runif(m * n, 0, 1),
                    " 二项" = rbinom(m * n, 10, 0.3),
                    " 泊松" = rpois(m * n, 5),
                    " 指数" = rexp(m * n), 1)
        data.frame(x = rowMeans(matrix(xs, ncol = n)))
    })
```

```
    output$plot = renderPlot({
        ggplot(Xbar(), aes(x)) +
          geom_histogram(alpha = 0.2, bins = input$bins,
                         fill = "steelblue",
                         color = "black")
    })
}
```

最后，将它们组装到 `app.R`, 注意需要加载 `ggplot2` 包。运行 App, 得到：



# 案例：探索性数据展板

Shiny 最常用的场景是设计动态数据分析报表或仪表盘，给他人以交互式的结果呈现。最后再看一个用 Shiny 制作探索性数据展板的案例。

以交互探索 gapminder 数据为例，该数据包含 142 个国家 1952-2007 年 5 年间隔的预期寿命、人均 GDP 等。

我们设计这样一些交互需求：

- 让用户选择国家，下拉选项输入
- 对该国家预期寿命绘制折线图，图形输出
- 表格输出该国家的数据子集，并可以导出到文件，数据表输出

**ui 用户界面布局：**选用侧边栏 + 主面板，在侧边栏采用下拉选项输入国家；主面板所选国家的预期寿命图形和数据表，设计为可通过标签切换的两个页面。

**server() 交互逻辑：**将从用户输入国家到筛选出该国家的数据放入响应表达式，用于随后的渲染输出图形和渲染输出数据表。

另外，为了增加图形的可交互性（移动鼠标显示当前数据），使用 plotly 包的 plotlyOutput 对象；为了增加数据表的可交互性（换页显示、可导出到文件），使用 DT 包的 dataTableOutput 对象。

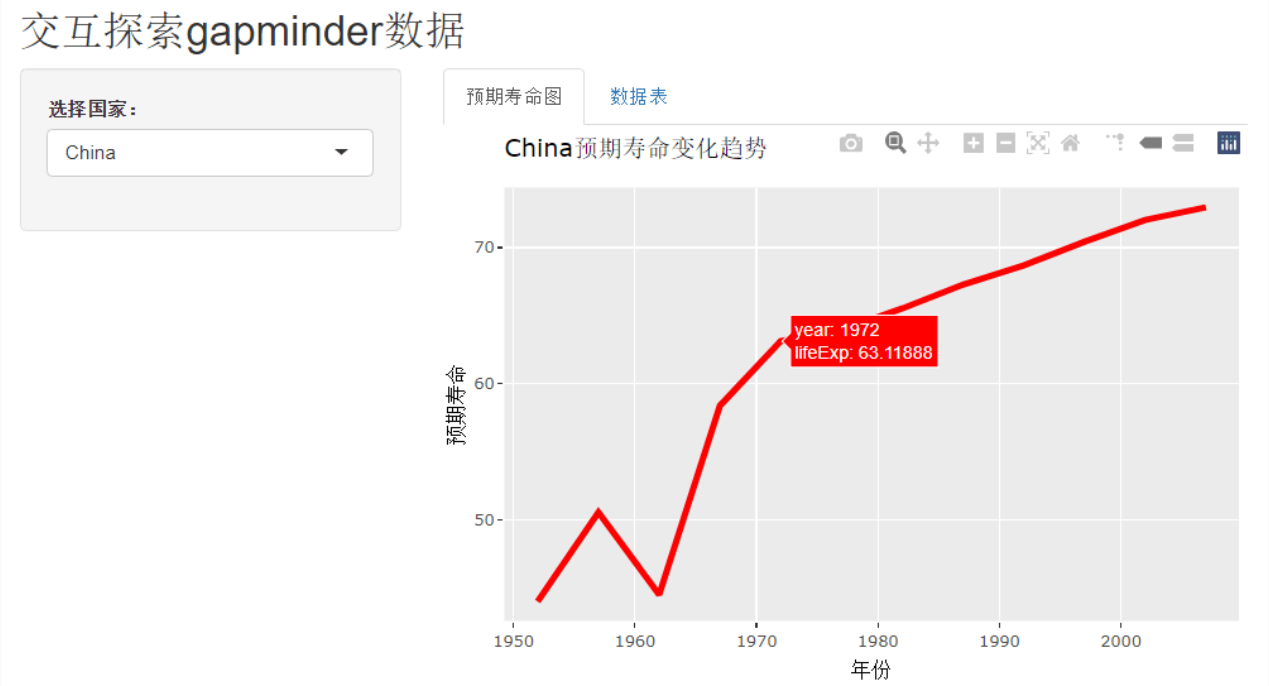
```
library(shiny)
library(tidyverse)
load("datas/gapminder.rda")    # 载入数据
countries = unique(gapminder$country)
```

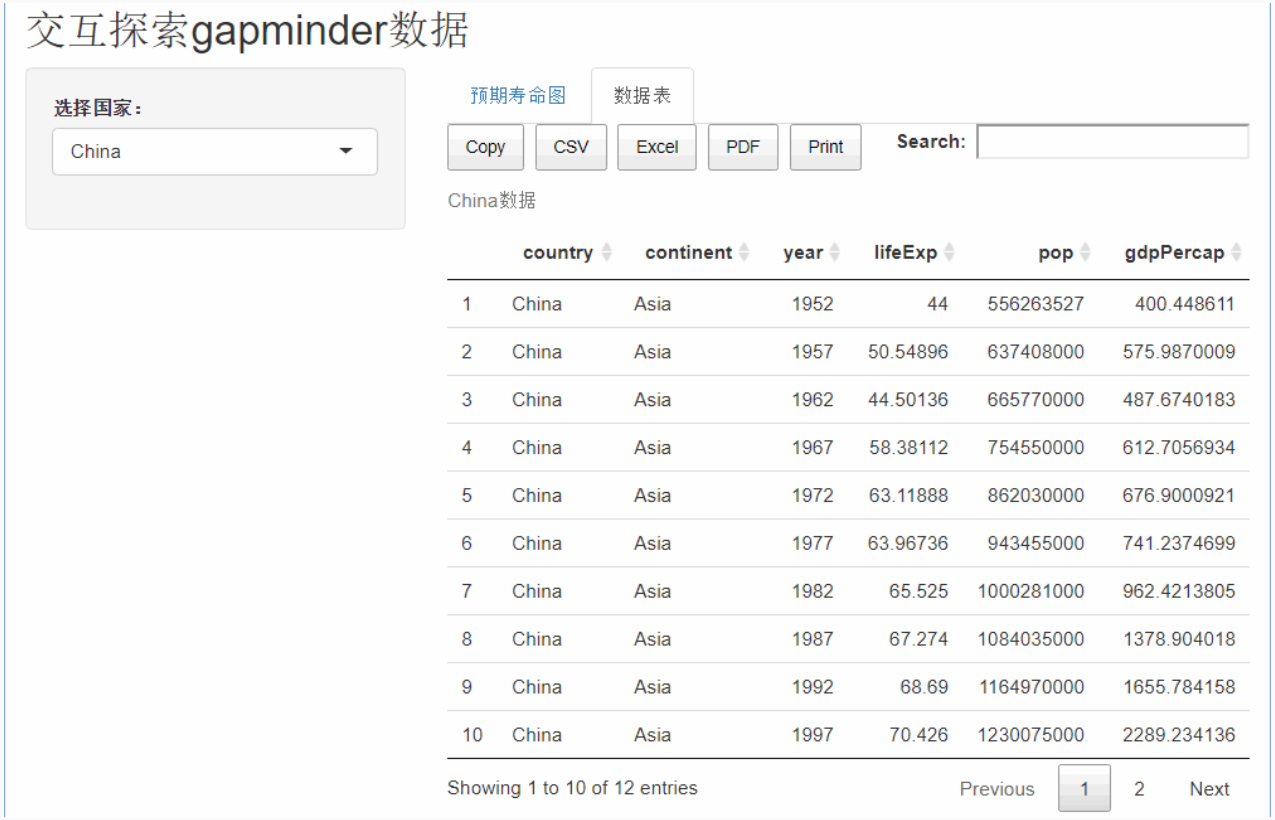
```r
# 用户界面
ui = fluidPage(
  titlePanel(" 交互探索 gapminder 数据"),
  sidebarLayout(          # 侧边栏带下拉选项选择国家
    sidebarPanel(
      selectInput("name", " 选择国家: ",
                  choices = countries,
                  selected = "China")),
      mainPanel(          # 主面板带图形和数据表的切换标签
        tabsetPanel(
          tabPanel(" 预期寿命图",
                   plotly::plotlyOutput("gap_plot")),
          tabPanel(" 数据表",
                   DT::dataTableOutput("gap_data"))))))
```

```r
# 定义服务器逻辑：绘制折线图、创建数据表
server = function(input, output) {
  selected = reactive({
    gapminder %>%
      filter(country == input$name)
  })
  # 绘制可交互折线图
  output$gap_plot = plotly::renderPlotly({
    p = ggplot(selected(), aes(year, lifeExp)) +
          geom_line(color = "red", size = 1.2) +
          labs(title = paste0(input$name,
                              " 预期寿命变化趋势"),
               x = " 年份", y = " 预期寿命")
    plotly::ggplotly(p)    # 渲染 plotly 对象
  })
```

```
  # 创建数据表
  output$gap_data = DT::renderDataTable({
    DT::datatable(selected(), extensions = "Buttons",
                  caption = paste0(input$name, " 数据"),
                  options = list(dom = "Bfrtip",
                    buttons = c("copy", "csv", "excel", "pdf"
    })
}
# 运行 App
shinyApp(ui = ui, server = server)
```

运行该 app, 默认在预期寿命图界面，点击数据表可切换到数据表页面：




交互探索gapminder数据
选择国家：
China
预期寿命图
数据表
Copy
CSV
Excel
PDF
Print
Search:
China数据
country
continent
year
lifeExp
pop
gdpPercap
1 China Asia 1952 44 556263527 400.448611
2 China Asia 1957 50.54896 637408000 575.9870009
3 China Asia 1962 44.50136 665770000 487.6740183
4 China Asia 1967 58.38112 754550000 612.7056934
5 China Asia 1972 63.11888 862030000 676.9000921
6 China Asia 1977 63.96736 943455000 741.2374699
7 China Asia 1982 65.525 1000281000 962.4213805
8 China Asia 1987 67.274 1084035000 1378.904018
9 China Asia 1992 68.69 1164970000 1655.784158
10 China Asia 1997 70.426 1230075000 2289.234136
Showing 1 to 10 of 12 entries
Previous
1
2
Next

有了该 Shiny app，用户可以很方便地选择国家，查看该国家的预期寿命变化趋势图并能在图上交互，还能将该国家的数据导出到文件。

最后，关于分享做好的 Shiny app：

- 以 R 脚本分享，这是最简单的方法，但需要用户有 R 和 Shiny 环境，且知道如何运行它
- 以网页形式分享，用户只要联网用浏览器就能交互使用它，但是这需要托管到云服务器，RStudio 提供了 shineapps.io（免费受限），还有 Shiny 的配套服务器程序 Shiny Server，但是只能部署在支持 Ubuntu 和 CentOS/RHEL 的 Linux 服务器。

本篇主要参阅 (张敬信, 2022), (Wickham, 2021), 以及 R4WRDS: Simple Shiny, Shiny in seven lessons，模板感谢 (黄湘云, 2021), (谢益辉, 2021).

# 参考文献


Wickham, H. (2021). *Mastering Shiny*. O’ Reilly. 9781492047384.

张敬信 (2022). *R* 语言编程：基于 *tidyverse*. 人民邮电出版社, 北京.

谢益辉 (2021). *rmarkdown: Dynamic Documents for R*.

黄湘云 (2021). *Github: R-Markdown-Template*.