**第1部分 入门**

**简介**

接下来四章的目标是让你尽快编写Shiny应用程序。在第1章中，我将从一个小而完整的APP开始，向您展示一个应用程序的所有主要部分，以及它们是如何组装在一起的。然后在第2章和第3章中，您将开始深入了解Shiny应用程序的两个主要部分：前端(用户在浏览器中看到的内容)和后端(使其工作的代码)。我们将在第4章中以一个案例研究结束，以帮助巩固您迄今所学的概念。

**1您的第一个Shiny应用**

**1.1简介**

在本章中，我们将创建一个简单的Shiny应用程序。我将首先向您展示一个Shiny应用程序所需的最小样板，然后您将学习如何启动和停止它。接下来，您将学习每个Shiny应用程序的两个关键组件：定义应用程序外观的UI(用户界面的缩写)和定义应用程序如何工作的服务器函数。Shiny使用反应式编程在输入发生变化时自动更新输出，因此我们将学习Shiny应用程序的第三个重要组成部分：反应式表达式，来结束本章。

如果尚未安装Shiny，请立即使用以下命令安装：

install.packages("shiny")

如果您已经安装了Shiny，请使用packageVersion(“Shiny”)检查您是否有1.5.0或更高版本。

然后在当前R会话中加载：

library(shiny)

**1.2创建应用程序目录和文件**

有几种方法可以创建一个Shiny应用程序。最简单的方法是为您的应用程序创建一个新目录，并放入一个名为app.R的文件。这个app.R文件将告诉Shiny你的应用程序应该是什么样子，以及它应该怎样工作。

通过创建一个新目录并添加一个app.R文件来尝试一下，如下所示：

library(shiny)

ui <- fluidPage(

 "Hello, world!"

)

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

}

shinyApp(ui, server)

这是一个完整的Shiny应用程序，虽然平凡！仔细看上面的代码，我们的app.R做了四件事：

1、它调用library(shiny)来加载shinny包。

2、它定义了用户界面，即人与之交互的HTML网页。在本例中，它是一个包含“Hello, world!”字样的页面。

3、它通过定义一个服务器函数来指定我们的应用程序的行为。它目前是空的，所以我们的应用程序什么都不做，但我们很快就会回来重新讨论这个问题。

4、它执行shinyApp(ui, server)，从用户界面和服务器函数构建并启动一个Shiny应用程序。

RStudio提示：有两种方便的方法可以在RStudios中创建新应用：

1、创建新目录和app.R文件，它包含一个基本的APP。单击File|New Project，然后选择New Directory和Shiny Web Application，一步完成。

2、如果您已经创建了app.R文件，您可以通过键入“shinyapp”并按Shift+Tab快速添加应用程序样板。

**1.3运行和停止**

有几种方法可以运行此应用程序：

单击文档工具栏中的RunApp(图1.1)按钮。

使用键盘快捷键：Cmd/Ctrl+Shift+Enter。

如果您没有使用RStudio，您可以(source())**[[1]](#footnote-1)**读取整个文档，或者使用包含app.R的目录的路径调用shiny::runApp()。



图1.1:RunApp按钮位于源代码窗格的右上角。

选择其中一个选项，并检查您是否看到与图1.2中相同的应用程序。祝贺您！您已经开发了第一个Shiny应用程序。





图1.2：运行上述代码时，您将看到一个非常基本的Shiny应用程序

在关闭应用程序之前，请返回RStudio并查看R控制台。你会注意到它的输出是这样的：

#> Listening on http://127.0.0.1:3827

它告诉您可以找到您的应用程序的URL：127.0.0.1是一个标准地址，表示“这台计算机”，3827是一个随机分配的端口号。您可以在任何兼容[[2]](#footnote-2)的web浏览器中输入该URL，以打开应用程序的另一个副本。

另外要注意，R正忙着：R命令行提示不可见，控制台工具栏上显示了一个停止标志图标。当一个Shiny应用程序运行时，它会“阻塞”R控制台。这意味着在Shiny应用程序停止之前，您无法在R控制台上运行新命令。

您可以使用以下任一选项停止应用程序并返回对控制台的访问：

单击R控制台工具栏上的停止标志图标。

单击控制台，然后按Esc键(如果未使用RStudio，请按Ctrl+C键)。

关闭Shiny应用程序窗口。

Shiny应用程序开发的基本工作流程是编写一些代码、启动应用程序、跑跑应用程序、编写更多代码，然后重复。如果您使用的是RStudio，您甚至不需要停止并重新启动应用程序来查看您的更改-您可以按工具箱中的“重新加载应用程序”按钮或使用Cmd/Ctrl+Shift+Enter键盘快捷键。我将在第5章中介绍其他的工作流模式。

**1.4添加UI控件**

接下来，我们将向UI添加一些输入和输出，这样就不会太少了。我们将制作一个非常简单的应用程序，向您展示datasets包中包含的所有内置数据集。

用以下代码替换您的ui：

ui <- fluidPage(

 selectInput("dataset", label = "Dataset", choices = ls("package:datasets")),

 verbatimTextOutput("summary"),

 tableOutput("table")

)

此示例使用了四个新函数：

fluidPage()是一个布局函数，用于设置页面的基本视觉结构。您将在第6.2节中了解有关它们的更多信息。

selectInput()是一个输入控件，允许用户通过提供值与应用程序交互。在本例中，它是一个带有标签“Dataset”的选择框，允许您选择R附带的一个内置数据集。您将在第2.2节中了解有关输入的更多信息。

VerbatimTexOutput()和tableOutput()是输出控件，它们告诉Shiny将渲染输出放在何处(稍后我们将讨论如何放置)。verbatimTextOutput()显示代码输出的文本，tableOutput()显示表格。您将在第2.3节中了解有关输出的更多信息。

布局函数、输入和输出有不同的用途，但它们在本质上是相同的：它们都只是生成HTML的奇特方式，如果你在一个Shiny应用程序之外调用它们，你会看到HTML在控制台上打印出来。不要害怕到处看看这些不同的布局和控件在引擎盖下是如何工作的。

继续并再次运行应用程序。现在您将看到图1.3，一个包含选择框的页面。我们只看到输入，而没有两个输出，因为我们还没有告诉Shiny输入和输出是如何关联的。



图1.3：具有UI的数据集应用程序

**1.5添加行为**

接下来，我们将通过在服务器函数中定义输出来实现输出。

Shiny使用反应式编程使应用程序具有交互性。在第3章中，您将了解更多关于反应式编程的内容，但现在，请注意，它涉及到告诉Shiny如何执行计算，而不是命令Shiny实际去执行。这就像给某人一份食谱和要求他们给你做三明治之间的区别。

我们将通过提供这些输出的“食谱”，告诉Shiny如何在示例应用程序中填写数据的汇总和表格输出。将空服务器函数替换为以下内容：

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

 })

}

赋值运算符(<-)的左侧为输出$ID，表示您正在使用该ID为Shiny的输出提供食谱。赋值的右侧使用特定的渲染函数包装您提供的一些代码。每个render{Type}函数设计用于生成特定类型的输出(例如文本、表格和绘图)，并且通常与{type}Output函数配对。例如，在这个应用程序中，renderPrint()与verbatimTextOutput()配对以显示具有固定宽度(逐字)文本的统计汇总，renderTable()与tableOutput()成对以显示表格中的输入数据。

再次运行应用程序并四处试试，观察更改输入时输出的变化。图1.4显示了打开应用程序时应该看到的内容。



图1.4：现在我们提供了连接输出和输入的服务器函数，我们有了一个功能齐全的应用程序

请注意，每当更改输入数据集时，汇总和表都会更新。此依赖关系是隐式创建的，因为我们在输出函数中引用了input$dataset。input$dataset由id为dataset的UI组件的当前值填充，并将使输出在该值更改时自动更新。这就是反应的本质：当输入发生变化时，输出会自动反应(重新计算)。

**1.6使用反应式表达式减少重复**

即使在这个简单的示例中，我们也有一些重复的代码：以下行出现在两个输出中。

dataset <- get(input$dataset, "package:datasets")

在每一种编程中，重复代码都是不好的做法；这可能造成计算上的浪费，更重要的是，它增加了维护或调试代码的难度。这在这里并不重要，但我想用一个非常简单的上下文来说明基本思想。

在传统的R脚本中，我们使用两种技术来处理重复代码：要么使用变量来表示值，要么使用函数来表示计算。不幸的是，由于您将在第13.2节中了解的原因，这两种方法在这里都不起作用，我们需要一种新的机制：反应表达式。

通过将代码块包装在reactive({...})中并将其分配给变量来创建反应表达式，通过像函数一样调用来使用反应表达式。但是，虽然看起来像是在调用函数，但反应表达式有一个重要的区别：它只在第一次调用时运行，然后缓存其结果，直到需要更新为止。

我们可以更新server()以使用反应表达式，如下所示。该应用程序的行为相同，但工作效率更高，因为它只需要检索一次数据集，而不是两次。

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

 # Create a reactive expression

 dataset <- reactive({

 get(input$dataset, "package:datasets")

 })

 output$summary <- renderPrint({

 # Use a reactive expression by calling it like a function

 summary(dataset())

 })

 output$table <- renderTable({

 dataset()

 })

}

我们将多次回到反应式编程，但即使只掌握了输入、输出和反应表达式的粗略知识，也有可能构建非常有用的Shiny应用程序！

**1.7小结**

在本章中，您创建了一个简单的应用程序-它不是非常令人兴奋或有用，但您看到了使用现有的R知识构建web应用程序是多么容易。在接下来的两章中，您将了解更多关于用户界面和反应式编程的信息，这是Shiny的两大基石。现在是一个很好的时机来获取一份Shiny的备忘录。这是一个很好的资源，可以帮助你快速记忆Shiny应用程序的主要组件。



图1.5：Shiny备忘录，可从https://www.rstudio.com/resources/cheatsheets/访问。

**1.8练习**

1、创建一个应用程序，通过名字向用户打招呼。您还不了解执行此操作所需的所有函数，因此我在下面包含了一些代码行。想想你要用哪行，然后在一个Shiny应用程序中复制并粘贴到正确的位置。

tableOutput("mortgage")

output$greeting <- renderText({

 paste0("Hello ", input$name)

})

numericInput("age", "How old are you?", value = NA)

textInput("name", "What's your name?")

textOutput("greeting")

output$histogram <- renderPlot({

 hist(rnorm(1000))

}, res = 96)

2、假设您的朋友想设计一款应用程序，允许用户将数字(x)设置在1和50之间，并显示该数字乘以5的结果。这是他们的第一次尝试：

library(shiny)

ui <- fluidPage(

 sliderInput("x", label = "If x is", min = 1, max = 50, value = 30),

 "then x times 5 is",

 textOutput("product")

)

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

 output$product <- renderText({

 x \* 5

 })

}

shinyApp(ui, server)

但不幸的是，它有一个错误：



你能帮助他们发现并纠正错误吗？

3、从上一个练习扩展应用程序，允许用户设置乘数y的值，以便应用程序生成x\*y的值。最终结果应如下所示：



4、以下面的应用程序为例，它为上一个练习中描述的最后一个应用程序添加了一些附加功能。有什么新尝试？如何通过使用反应表达式减少应用程序中重复代码的数量。

library(shiny)

ui <- fluidPage(

 sliderInput("x", "If x is", min = 1, max = 50, value = 30),

 sliderInput("y", "and y is", min = 1, max = 50, value = 5),

 "then, (x \* y) is", textOutput("product"),

 "and, (x \* y) + 5 is", textOutput("product\_plus5"),

 "and (x \* y) + 10 is", textOutput("product\_plus10")

)

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

 output$product <- renderText({

 product <- input$x \* input$y

 product

 })

 output$product\_plus5 <- renderText({

 product <- input$x \* input$y

 product + 5

 })

 output$product\_plus10 <- renderText({

 product <- input$x \* input$y

 product + 10

 })

}

shinyApp(ui, server)

5、下面的应用程序与您在本章前面看到的应用程序非常相似：您从包中选择一个数据集(这次我们使用的是ggplot2包)，然后应用程序打印出数据的汇总和绘图。它还遵循良好的实践，并使用反应表达式来避免代码冗余。然而，下面提供的代码中有三个bug。你能找到并修复它们吗？

library(shiny)

library(ggplot2)

datasets <- c("economics", "faithfuld", "seals")

ui <- fluidPage(

 selectInput("dataset", "Dataset", choices = datasets),

 verbatimTextOutput("summary"),

 tableOutput("plot")

)

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

 dataset <- reactive({

 get(input$dataset, "package:ggplot2")

 })

 output$summmry <- renderPrint({

 summary(dataset())

 })

 output$plot <- renderPlot({

 plot(dataset)

 }, res = 96)

}

shinyApp(ui, server)

1. 外面的额外的()很重要。 shinyApp() 仅在打印时创建一个应用程序，()强制打印source()文件中的最后一个结果，否则将不可见地返回。 [↑](#footnote-ref-1)
2. Shiny努力支持所有现代浏览器，您可以在<https://www.rstudio.com/about/platform-support/>看到完整的列表。请注意，IE11之前的Internet Explorer版本在直接从R会话运行Shiny时不兼容。但是，在Shiny Server或ShinyApps.io上部署的Shiny应用程序可以与IE10一起使用（不再支持早期版本的IE)。 [↑](#footnote-ref-2)