

Shiny in action

Introduction

以下章节为您提供了有用的技术。我认为每个人都应该从第[20.2.1章](#)开始，因为它为您提供了开发和调试应用程序的重要工具，并在您遇到困难时获得帮助。

在那之后，章节之间没有规定的顺序，联系也相对较少：我建议快速浏览以了解各章节主题。因此，如果将来出现相关问题，您可能会记住这些工具，并深入阅读您目前需要的部分。以下是主要主题的快速浏览：

- 第[6章](#)：详细介绍了在页面上布局输入和输出组件的各种方式，以及如何使用主题自定义其外观。
- 第[7章](#)：向您展示了如何将直接交互添加到您的绘图中，以及如何显示以其他方式生成的图像。
- 第[8章](#)：涵盖了在应用程序运行时向用户提供反馈的技术（包括内联错误、通知、进度条和对话框）。
- 第[9章](#)：讨论了如何将文件传输到您的应用程序和从您的应用程序下载。
- 第[10章](#)：向您展示了如何在应用程序运行时动态修改用户界面。
- 第[11章](#)：展示了如何以用户书签的方式记录应用程序状态。
- 第[12章](#)：向您展示了如何允许用户在使用tidyverse软件包时选择变量。

让我们从开发应用程序的工作流程开始。

Chapter 5. Workflow

如果你要写很多shiny应用程序（既然你正在读这本书，我希望你会读！），那么值得在你的基本工作流程中投入一些时间。改善工作流程是投入时间的好地方，因为从长远来看，它往往会有巨大的红利。它不仅会增加您写R代码所花费的时间比例，而且因为您可以更快地看到结果，它使编写Shiny应用程序的过程更愉快，并帮助您更快地提高技能。

本章的目标是帮助您改进三个重要的shiny工作流程：

- 创建应用程序、进行更改和实验的基本开发周期。
- 调试，找出流程中代码出了什么问题，然后集思广益，通过解决方案来修复它。
- 编写reprex（自包含的代码块），来说明问题。复制是一种强大的调试技术，如果您想从他人那里获得帮助，它们是必不可少的。

5.1 Development workflow

优化您的开发工作流程的目标是减少从进行更改到看到结果之间的时间。你越快地进行尝试，你越快地进行实验，你就能越快地成为一个更好的shiny开发人员。这里有两个主要的工作流程需要优化：首次创建应用程序，以及加快调整代码和尝试结果的迭代周期。

5.1.1 Creating the app

您将使用相同的六行R代码启动每个应用程序：

```

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

```

您可能很快就会厌倦输入该代码，因此RStudio提供了一些快捷方式：

- 如果您已经打开了 `app.R`，请键入 `shinyapp`，然后按 `Shift+Tab` 键插入Shiny应用程序片段
- 如果您想启动新项目，请转到 `File menu`，选择 `“New Project”`，然后选择 `“Shiny Web Application”`，如图5.1所示。

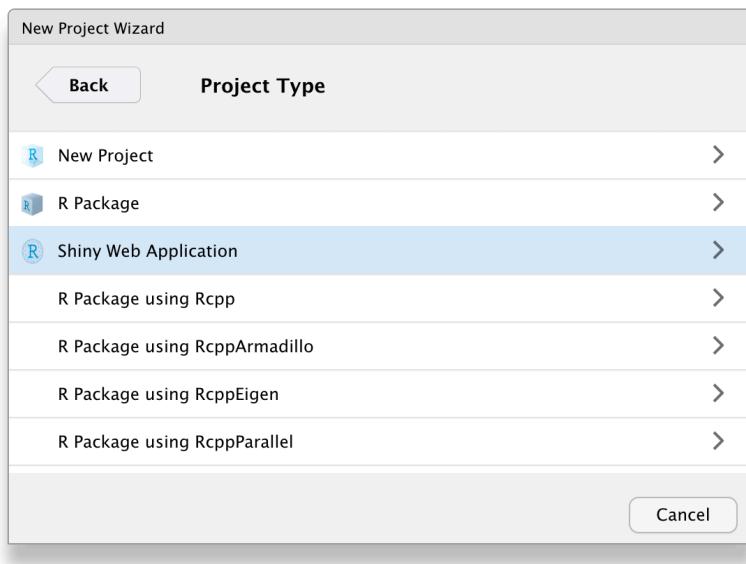


图5.1：要在RStudio中创建新的Shiny应用程序，请选择“Shiny Web Application”作为项目类型

您可能认为学习这些快捷方式不值得，因为您每天只创建一两个应用程序，但创建简单的应用程序是在开始更大的项目之前检查您是否掌握了基本概念的好方法，它们是调试的绝佳工具。

5.1.2 Seeing your changes

最多，你每天会创建几个应用程序，但你会运行数百次应用程序，因此掌握开发工作流程尤为重要。减少迭代时间的第一种方法是避免单击“Run App”按钮，而是学习键盘快捷键 `Cmd/Ctrl+Shift+Enter`。这为您提供了以下开发工作流程：

1. 写一些代码。
2. 基于 `Cmd/Ctrl + Shift + Enter` 执行应用程序。
3. 交互式实验该应用程序。
4. 关闭应用程序。
5. 转到1。

进一步提高迭代速度的另一种方法是打开自动重新加载并在后台作业中运行应用程序，如<https://github.com/sol-e/ng/background-jobs/tree/master/shiny-job>中所述。使用此工作流程，一旦您保存文件，您的应用程序将重新启动：无需关闭和重新启动。这导致了更快的工作流程：

1. 编写一些代码，并按 `Cmd/Ctrl + S` 保存文件。
2. 交互式实验。

3. 转到1。

这种技术的主要缺点是调试要困难得多，因为应用程序是在单独的进程中运行的。

随着您的应用程序越来越大，您会发现“交互式实验”步骤开始变得繁重。记住重新检查应用程序中可能因更改而受到影响的每个组件都太难了。稍后，在第21章中，您将学习自动测试的工具，该工具允许您将正在运行的交互式实验转换为自动代码。这可以让您更快地运行测试（因为它们是自动的），并且意味着您不能忘记运行重要的测试。开发测试需要一些初始投资，但对于大型应用程序来说，投资回报丰厚。

5.1.3 Controlling the view

默认情况下，当您运行该应用程序时，它将出现在弹出窗口中。您可以从运行应用程序下拉菜单中选择另外两个选项，如图1.1所示：

- 在查看器窗格中运行（Run in Viewer Pane）：在查看器窗格中打开应用程序（通常位于IDE的右侧）。它对较小的应用程序很有用，因为您可以在运行应用程序代码的同时看到它。
- 运行外部（Run External）：在您通常的网络浏览器中打开该应用程序。它对较大的应用程序很有用，当你想在大多数用户都会体验到的环境中看到你的应用程序是什么样子时，使用这种方式。

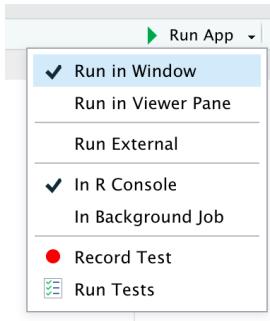


图1.1：运行应用程序按钮允许您选择正在运行的应用程序的显示方式。

5.2 Debugging

当你开始编写应用程序时，几乎可以保证会出现问题。大多数错误的原因是你的Shiny理想模型与Shiny的实际作用不匹配。当你阅读这本书时，你的思维模型会得到改善，这样你就会犯更少的错误，当你犯错误时，更容易发现问题。然而，在任何语言方面都需要多年的经验，你才能可靠地编写第一次有效的代码。这意味着您需要开发一个强大的工作流程来识别和修复错误。在这里，我们将重点关注Shiny应用程序特有的挑战；如果您是R中调试的新手，请从Jenny Bryan的rstudio::conf (2020) 主题演讲“[Object of 'closure' is not subsettable](#)”开始。

有三个主要问题案例，我们将在下面讨论：

- 你收到一个意想不到的错误。这是最简单的情况，因为您将获得一个回溯，允许您准确定位错误发生的位置。一旦你确定了问题，你需要系统地测试你的假设，直到你发现你的期望和现实之间的差异。交互式调试器是此过程的强大助手。
- 您没有收到任何错误，但有些值不正确。在这里，您需要使用交互式调试器以及您的调查技能来追踪根本原因。
- 所有值都是正确的，但它们不会在您预期时更新。这是最具挑战性的问题，因为它是Shiny独有的，因此您无法利用现有的R调试技能。

当出现这些情况时，这很令人沮丧，但你可以把它们变成练习调试技能的机会。

我们将在下一节中讲到另一个重要技术：做一个最小的可重现的例子。如果你陷入困境，需要从别人那里获得帮助，那么创建一个最小的例子至关重要。但在调试自己的代码时，创建一个最小的示例也是一项非常重要的技能。通常，您有很多运行良好的代码，以及具有非常少量问题的代码。如果您可以通过删除有效的代码来缩小有问题的代码范围，您将能够更快地对解决方案进行重进。这是我每天使用的一种技术。

5.2.1 Reading tracebacks

在R中，每个错误都伴随着一个回溯（**traceback**）或调用堆栈，从字面上追溯到导致错误的调用序列。例如，以这个简单的调用顺序为例：`f()` 调用 `g()` 调用 `h()`，它调用乘法运算符：

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

如果此代码出错，如下所示：

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

您可以调用 `traceback()` 来查找针对这个问题的调用顺序：

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

我认为通过将其颠倒来最容易理解追溯：

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

这现在告诉你导致错误的调用顺序—`f()` 调用 `g()` 调用 `h()`（错误出现的地方）。

5.2.2 Tracebacks in Shiny

不幸的是，您无法在Shiny中使用 `traceback()`，因为您无法在应用程序运行时运行代码。相反，Shiny将自动为您打印回溯。例如，使用我上面定义的 `f()` 函数来执行这个简单的应用程序：

```
library(shiny)

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

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)
```

如果您运行此应用程序，您将在应用程序中看到一条错误消息，并在控制台中看到一条回溯：

```
Error in *: non-numeric argument to binary operator  
169: g [app.R#4]  
168: f [app.R#3]  
167: renderPlot [app.R#13]  
165: func  
125: drawPlot  
111: <reactive:plotObj>  
95: drawReactive  
82: renderFunc  
81: output$plot  
1: runApp
```

为了了解发生了什么，我们再次从将其颠倒过来，这样您就可以按照它们出现的顺序看到调用的顺序：

```
Error in *: non-numeric argument to binary operator  
1: runApp  
81: output$plot  
82: renderFunc  
95: drawReactive  
111: <reactive:plotObj>  
125: drawPlot  
165: func  
167: renderPlot [app.R#13]  
168: f [app.R#3]  
169: g [app.R#4]
```

调用堆栈有三个基本部分：

- 前几个调用，启动应用程序。在这种情况下，您只会看到 `runApp()`，但根据您启动应用程序的方式，您可能会看到更复杂的内容。例如，如果您调用 `source()` 来运行应用程序，您可能会看到以下内容：

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

一般来说，您可以在第一个 `runApp()` 之前忽略任何内容；这只是让应用程序运行的设置代码。

- 接下来，您将看到一些负责调用反应式表达式的内部Shiny代码：

```
81: output$plot
82: renderFunc
95: drawReactive
111: <reactive:plotObj>
125: drawPlot
165: func
```

在这里，发现 `output$plot` 真的很重要—这说明了您的哪些反应式表达式（`plot`）导致了错误。接下来的几个功能是内部的，您可以忽略它们。

- 最后，在最底部，您将看到您编写的代码：

```
167: renderPlot [app.R#13]
168: f [app.R#3]
169: g [app.R#4]
```

这是在 `renderPlot()` 中调用的代码。在这里，你应该注意，因为这里出现了文件路径和行号，这让你知道这是你的代码。

如果您在应用程序中遇到错误，但没有看到回溯，请确保您正在使用 `Cmd/Ctrl+Shift+Enter` 运行应用程序（如果不在RStudio中，请调用 `runApp()`），并且您已经保存了运行该应用程序的文件。其他运行应用程序的方式并不总是捕捉到进行回溯所需的信息。

5.2.3 The interactive debugger

一旦您找到了错误的来源，并想找出导致错误的原因，您拥有的最强大的工具就是**交互式调试器（interactive debugger）**。调试器暂停执行，并为您提供一个交互式R控制台，您可以在其中运行任何代码来找出问题所在。有两种方法可以启动调试器：

- 在源代码中添加 `browser()`。这是启动交互式调试器的标准R方式，无论您如何运行Shiny，它都会工作。
`browser()` 的另一个优点是，因为它是R代码，您可以通过将其与 `if` 语句组合来使其有条件。这允许您仅为有问题的输入启动调试器。

```
if (input$value == "a") {
  browser()
}
# Or maybe
if (my_reactive() < 0) {
  browser()
}
```

- 通过单击行号左侧添加RStudio断点。您可以通过单击红色圆圈来删除断点。

```
23: server <- function(input, output, session) {
24:   territory <- reactive({
25:     req(input$territory)
26:     if (input$territory == "NA") {
```

断点的优点是它们不是代码，因此您不必担心意外将它们添加到版本控制系统中。

如果您使用的是RStudio，当您在调试器中时，图5.2中的工具栏将显示在控制台的顶部。工具栏是记住您现在可用的调试命令的简单方法。它们也可以在RStudio之外使用；您只需记住一个字母命令即可激活它们。三个最有用的命令是：

- 下一步（按n）：执行函数中的下一步。请注意，如果您有一个名为n的变量，则需要使用print(n)来显示其值。
- 继续（按c）：离开交互式调试，并继续定期执行该功能。如果您已经修复了坏状态，并希望检查函数是否正确进行，这非常有用。
- 停止（按q）：停止调试，终止函数，并返回全局工作区。一旦你弄清楚问题在哪里，并准备好修复它并重新加载代码，就使用它。



图5.2：RStudio的调试工具栏

除了使用这些工具逐行浏览代码外，您还将编写和运行一堆交互式代码来跟踪出错。调试是系统地将您的期望与现实进行比较的过程，直到您发现不匹配。如果您是R中调试的新手，您可能需要阅读“Advanced R”的[Debugging chapter](#)，以学习一些调试技术。

5.2.4 Case study

一旦你消除了不可能的东西，无论剩下什么，无论多么不可能，都一定是真相—夏洛克·福尔摩斯

为了演示基本的调试方法，我将向您展示我在编写第[10.1.2](#)节时遇到的一个小问题。我首先向您展示基本上下文，然后您将看到我在没有交互式调试工具的情况下解决的问题，一个需要交互式调试的问题，并发现最后的惊喜。

最初的目标很简单：我有一个销售数据集，我想按地区进行筛选。以下是数据的基本情况：

```
sales <- readr::read_csv("sales-dashboard/sales_data_sample.csv")
sales <- sales[c(
  "TERRITORY", "ORDERDATE", "ORDERNUMBER", "PRODUCTCODE",
  "QUANTITYORDERED", "PRICEEACH"
)]
sales
#> # A tibble: 2,823 × 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
```

以下是地区信息：

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

当我刚开始解决这个问题时，我认为它足够简单，我可以在不做任何其他研究的情况下编写应用程序：

```

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))
}

```

我想，这是一个八行应用程序，不可能会出什么问题？好吧，当我打开应用程序时，我看到了很多缺失的值，无论我选择什么领域。最有可能的代码是选择数据显示的反应式表达式代码：`sales[sales$TERRITORY == input$territory,]`。因此，我停止了该应用程序，并迅速验证了这个问题：

```

sales[sales$TERRITORY == "EMEA", ]
#> # A tibble: 2,481 × 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

```

哎呀！`TERRITORY` 包含一堆缺失值，这意味着 `sales$TERRITORY == "EMEA"` 将包含一堆缺失值：

```

head(sales$TERRITORY == "EMEA", 25)
#> [1] NA  TRUE  TRUE   NA   NA   NA  TRUE  TRUE   NA  TRUE FALSE  NA
#> [13] NA  NA   TRUE   NA  TRUE  TRUE   NA   NA  TRUE FALSE  TRUE  NA
#> [25] TRUE

```

当我使用这些缺失的值，提取 `sales` 数据框子集”时，这些缺失的值都会变成缺失的行；输入中的任何缺失值都将保留在输出中。有很多方法可以解决这个问题，但我决定使用 `subset()`，因为会自动删除缺失的值，并减少我需要键入 `sales` 的次数。然后我仔细检查了这确实有效：

```

subset(sales, TERRITORY == "EMEA")
#> # A tibble: 1,407 × 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

```

这修复了大多数问题，但当我在地区下拉菜单中选择 `NA` 时，我仍然遇到了一个问题：仍然没有出现行。因此，我再次检查了控制台：

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

然后我想起了，这当然行不通，因为缺失的值具有传染性：

```
head(sales$TERRITORY == NA, 25)
#> [1] NA NA
```

您可以使用另一个技巧来解决这个问题：从 `==` 切换到 `%in%`：

```
head(sales$TERRITORY %in% NA, 25)
#> [1] TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE TRUE FALSE FALSE TRUE
#> [13] TRUE TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE FALSE FALSE TRUE
#> [25] FALSE
subset(sales, TERRITORY %in% NA)
#> # A tibble: 1,074 × 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
```

所以我更新了应用程序并再次尝试。还是没用！当我在下拉菜单中选择“NA”时，我没有看到任何行。

此时，我想我已经在控制台上做了我能做的一切，我需要进行一个实验，以弄清楚为什么Shiny内部的代码没有按照我预期的方式工作。我猜问题最有可能的来源是 `selected` 的反应式表达式，所以我在那里添加了一个 `browser()` 语句。（这使它成为两行的反应式表达式，所以我也需要把它包装在 `{}`。）

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

现在，当我的应用程序运行时，我立即被转到交互式控制台中。我的第一步是验证我是否处于问题出现的地方，所以我运行了 `subset(sales, TERRITORY %in% input$territory)`，它返回了一个空的数据框，所以我知道我在我要去的地方。如果我没有看到这个问题，我会键入 `c` 让应用程序继续运行，然后与更多应用程序进行交互，以便它进入故障状态。

然后，我检查了 `subset()` 的输入是否符合我的预期。我首先仔细检查 `sales` 数据集，看起来还不错。我真的没有想到它会被损坏，因为应用程序中没有任何东西触及它，但仔细检查你正在做的每一个假设是最安全的。`sales` 看起来还不错，所以问题一定在 `TERRITORY %in% input$territory`。由于 `TERRITORY` 是 `sales` 的一部分，我首先检查了 `input$territory`：

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

我盯着这个看了一会儿，因为它看起来也还行。然后我突然想到了！我以为是 `NA`，但实际上不是 `"NA"`！现在我可以在Shiny应用程序之外重现问题：

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

然后，我找到了一个简单的修复程序，并将其应用于我的server中，并重新运行了该应用程序：

```
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))  
}
```

万岁！问题解决了！但这对我来说相当惊讶—Shiny默默地将 `NA` 转换为 `"NA"` 所以我也提交了一份错误报告：[http://github.com/rstudio/shiny/issues/2884](https://github.com/rstudio/shiny/issues/2884)。

几周后，我再次看了这个例子，并开始思考不同的地区。我们有欧洲、中东和非洲（EMEA）和亚太地区（APAC）。北美在哪里？然后我明白了：源数据可能使用了缩写NA，而R将其读作缺失值。因此，真正的修复应该发生在数据加载期间：

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

这让生活变得简单多了！

当涉及到调试时，这是一个常见的模式：在完全了解问题的根源之前，你通常需要剥开多层洋葱。

5.2.5 Debugging reactivity

最难调试的问题类型是当您的反应式表达式以意想不到的顺序触发时。在本书中，我们推荐的可以帮助您调试这个问题的工具相对较少。在下一节中，您将学习如何创建针对此类问题的至关重要的最小重复的调试方法。在书的后面，您将了解更多关于基本理论以及类似反应式表达式日志的工具，<https://github.com/rstudio/reactlog>。但现在，我们将专注于一种在这里有用的经典技术：“打印”调试（print debugging）。

打印调试的基本想法是，每当您需要了解部分代码何时被计算时，调用 `print()`，并显示重要变量的值。我们称之为“打印”调试（因为在大多数语言中，您会使用打印函数），但在R中，使用 `message()` 更有意义：

- `print()` 专为显示数据向量而设计，因此它将引号放在字符串周围，并以 `[1]` 开头第一行。

- `message()` 将其结果发送到“标准错误（standard error）”，而不是“标准输出（standard output）”。这些是描述输出流的技术术语，您通常不会注意到，因为它们在交互式运行时都以相同的方式显示。但是，如果您的应用程序托管在其他地方，则发送到“标准错误”的输出将记录在日志中。

我还建议将 `message()` 与 `glue::glue()` 耦合，这样可以很容易地在消息中显示文本和值。如果您以前没有见过 `glue`，那么其基本思想是，包在 `{}` 中的任何东西都将被求值并插入到输出中：

```
library(glue)
name <- "Hadley"
message(glue("Hello {name}"))
#> Hello Hadley
```

最后一个有用的工具是 `str()` 它可以打印任何对象的详细结构。如果您需要仔细检查您有您期望的对象类型，这特别有用。

这是一个展示一些基本想法的应用程序。注意我如何在 `reactive()` 中使用 `message()`。我必须执行计算，发送消息，然后返回之前计算的值。

```
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()
})
```

当我启动应用程序时，控制台显示：

```
Updating y from 2 to 2
New total is 6
```

如果我把 `x` 滑块拖到 `3`，我看到了

```
Updating y from 2 to 6
New total is 8
New total is 12
```

如果你发现结果有点令人惊讶，不要担心。您将了解更多关于第8章和第3.3章中发生的事情。

5.3 Getting help

如果你在尝试这些技术后仍然陷入困境，那么可能是时候问问别人了。获得帮助的好地方是[Shiny社区网站](#)。本网站被许多Shiny用户以及Shiny软件包本身的开发人员阅读。如果你想通过帮助他人来提高你的shiny技能，这也是一个参观的好地方。

为了尽快获得最有用的帮助，您需要创建一个复制或可复制的示例（reproducible example）。reprex的目标是提供尽可能小的R代码片段，以说明问题，并且可以在另一台计算机上轻松运行。创建一个reprex是常见的礼貌（也符合你自己的最佳利益）：如果你想让别人帮助你，你应该让他们尽可能容易！

制作reprex是礼貌的，因为它将问题的基本要素捕获成其他人都可以运行的形式，这样无论谁试图帮助你，都可以快速看到问题到底是什么，并可以轻松尝试可能的解决方案。

5.3.1 Reprex basics

复制器只是一些R代码，当您将其复制并粘贴到另一台计算机上的R会话时，该代码有效。这是一个简单的shiny应用程序的reprex：

```
library(shiny)
ui <- fluidPage(
  selectInput("n", "N", 1:10),
  plotOutput("plot")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    n <- input$n * 2
    plot(head(cars, n))
  })
}
shinyApp(ui, server)
```

此代码不会对正在运行的计算机做出任何假设（除了安装了Shiny!）因此，任何人都可以运行此代码并看到问题：该应用程序抛出一个错误，说“non-numeric argument to binary operator”。

清楚地说明问题是获得帮助的第一步，因为任何人都可以通过复制和粘贴代码来重现问题，他们可以轻松地探索您的代码并测试可能的解决方案。（在这种情况下，您需要 `as.numeric(input$n)` 因为 `selectInput()` 在 `input$n` 创建一个字符串。）

5.3.2 Making a repress

制作reprex的第一步是创建一个包含运行代码所需的一切的自包含文件。您应该通过启动新的R会话，然后运行代码来检查它是否有效。确保您没有忘记加载任何使您的应用程序正常工作的软件包。

通常，让您的应用程序在别人的计算机上运行的最具挑战性的部分是消除仅存储在自己计算机上的数据的使用。有三个有用的模式：

- 通常，您正在使用的数据与问题没有直接关系，相反，您可以使用 `mtcars` 或 `iris` 等内置数据集。
- 其他时候，您也许可以编写一些R代码，以创建一个说明问题的数据集：

```
mydata <- data.frame(x = 1:5, y = c("a", "b", "c", "d", "e"))
```

- 如果这两种技术都失败了，您可以使用 `dput()` 将数据转换为代码。例如，`dput(mydata)` 生成将重新创建 `mydata` 的代码：

```
dput(mydata)
#> structure(list(x = 1:5, y = c("a", "b", "c", "d", "e")), class = "data.frame",
row.names = c(NA,
#> -5L))
```

一旦你有了那个代码，你可以把它放在你的reprex中来生成 `mydata`：

```
mydata <- structure(list(x = 1:5, y = structure(1:5, .Label = c("a", "b",
"c", "d", "e"), class = "factor")), class = "data.frame", row.names = c(NA,
-5L))
```

通常，在原始数据上运行 `dput()` 将生成大量代码，因此请找到说明问题的数据子集。您提供的数据集越小，其他人就越容易帮助您解决问题。

如果从磁盘读取数据似乎是问题不可减少的一部分，那么最后的策略是提供一个包含 `app.R` 和所需数据文件的完整项目。提供这个的最佳方式是作为托管在GitHub上的RStudio项目，但如果做不到这一点，您可以仔细制作一个可以在本地运行的zip文件。确保您使用相对路径（即 `read.csv("my-data.csv")` 而不是 `read.csv("c:\\my-user-name\\files\\my-data.csv")`），以便您的代码在另一台计算机上运行时仍然有效。

您还应该考虑阅读器，并花费一些时间格式化代码，使其易于阅读。如果您采用[tidyverse style guide](#)，您可以使用[styler](#)包自动重新格式化代码；这会迅速将您的代码变成更易于阅读的代码。

5.3.3 Making a minimal repress

创建一个可重现的示例是一个很好的第一步，因为它允许其他人精确地重现您的问题。然而，通常有问题的代码通常会隐藏在运行良好的代码中，因此您可以通过修剪代码来使帮助者更容易执行你的代码。

创建尽可能小的reprex对于Shiny应用程序尤为重要，这些应用程序通常很复杂。如果你能提取出你正在苦苦挣扎的应用程序的确切部分，而不是强迫潜在的帮助者理解你的整个应用程序，你将得到更快、更高质量的帮助。作为额外的好处，这个过程通常会引导你发现问题所在，所以你不必等待别人的帮助！

将一堆代码简化为基本问题是一项技能，一开始你可能不太擅长。没关系！即使是代码复杂性的降低很小，也会帮助帮助你的人，随着时间的推移，你的reprex缩小技能也会提高。

如果您不知道代码的哪个部分触发了问题，找到它的一个好方法是从应用程序中逐个删除部分代码，直到问题消失。如果删除特定代码使问题停止，则该代码很可能与问题有关。或者，有时从一个新的、空的应用程序开始并逐步构建它，直到你再次发现问题为止，这更简单。

一旦您简化了应用程序以演示问题，就值得进行最终检查：

- `ui` 中的每个输入和输出都与问题有关吗？
- 您的应用程序是否有复杂的布局，您可以简化以帮助关注手头的问题？您是否删除了所有使您的应用程序看起来

来不错的UI自定义，但与问题无关？

- `server()` 中是否有任何您现在可以删除的反应式表达式？
- 如果您尝试了多种方法来解决问题，您是否删除了所有不起作用的尝试？
- 您加载的每个软件包都需要说明问题吗？你能通过用虚拟代码替换函数来消除软件包吗？

这可能是一项大量的工作，但回报是巨大的：通常，当你进行重复时，你会发现解决方案，如果没有，获得帮助要容易得多。

5.3.4 Case study

为了说明制作顶级reprex的过程，我将使用[Scott Novgoratz](#)在[RStudio社区](#)上发布的一个例子。初始代码非常接近复制，但不太可重现，因为它忘记了加载一对软件包。作为起点，我：

- 添加了缺少的 `library(lubridate)` 和 `library(xts)`
- `ui` 和 `server` 拆分为单独的对象。
- 使用 `styler::style_selection()` 重新格式化代码。

这产生了以下重复：

```
library(xts)
library(lubridate)
library(shiny)

ui <- fluidPage(
  uiOutput("interaction_slider"),
  verbatimTextOutput("breaks")
)
server <- function(input, output, session) {
  df <- data.frame(
    dateTimes = c(
      "2019-08-20 16:00:00",
      "2019-08-20 16:00:01",
      "2019-08-20 16:00:02",
      "2019-08-20 16:00:03",
      "2019-08-20 16:00:04",
      "2019-08-20 16:00:05"
    ),
    var1 = c(9, 8, 11, 14, 16, 1),
    var2 = c(3, 4, 15, 12, 11, 19),
    var3 = c(2, 11, 9, 7, 14, 1)
  )

  timeSeries <- as.xts(df[, 2:4],
    order.by = strptime(df[, 1], format = "%Y-%m-%d %H:%M:%S"))
}
print(paste(min(time(timeSeries)), as.POSIXt(min(time(timeSeries))), sep = " "))
print(paste(max(time(timeSeries)), as.POSIXt(max(time(timeSeries))), sep = " "))

output$interaction_slider <- renderUI({
  sliderInput(
    "slider",
    "Select Range:",
```

```

    min = min(time(timeSeries)),
    max = max(time(timeSeries)),
    value = c(min, max)
  )
})

brks <- reactive({
  req(input$slider)
  seq(input$slider[1], input$slider[2], length.out = 10)
})

output$breaks <- brks
}
shinyApp(ui, server)

```

如果你运行这个reprex，你会在最初的帖子中看到同样的问题：一个错误，说明“min、max和value的类型不匹配。每个都必须是Date、POSIXt或number”。这是一个坚实的reprex：我可以很容易地在我的电脑上运行它，它立即说明了问题。然而，它有点长，所以不清楚是什么导致了问题。

为了使这个reprex更简单，我们可以仔细处理每行代码，看看它是否重要。在这样做的时候，我发现：

- 删除以 `print()` 开头的两行不会影响错误。这两行使用 `lubridate::is.POSIXt()`，这是lubridate的唯一用途，所以一旦我删除了它们，我就不再需要加载lubridate。
- `df` 是一个调用 `timeSeries` 转换为xts数据类型的的数据框。但使用 `timeSeries` 的唯一方式是通过返回日期时间 `time(timeSeries)`。因此，我创建了一个新的变量 `datetime`，其中包含一些虚拟的日期时间数据。这仍然产生相同的错误，所以我删除了 `timeSeries` 和 `df`，由于这是唯一使用xts的地方，我也删除了 `library(xts)`

这些更改一起产生了一个新的 `server()`，看起来像这样：

```

datetime <- Sys.time() + (86400 * 0:10)

server <- function(input, output, session) {
  output$interaction_slider <- renderUI({
    sliderInput(
      "slider",
      "Select Range:",
      min = min(datetime),
      max = max(datetime),
      value = c(min, max)
    )
  })

  brks <- reactive({
    req(input$slider)
    seq(input$slider[1], input$slider[2], length.out = 10)
  })

  output$breaks <- brks
}

```

接下来，我注意到这个例子使用了一种相对复杂的Shiny技术，其中UI是在server函数中生成的。但在这里，`renderUI()`不使用任何反应式表达式输入，因此，如果移除server函数，并进入UI，它应该以同样的方式工作。

这产生了一个特别好的结果，因为现在错误发生得更早，甚至在我们启动应用程序之前：

```
ui <- fluidPage(  
  sliderInput("slider",  
    "Select Range:",  
    min = min(datetime),  
    max = max(datetime),  
    value = c(min, max)  
,  
  verbatimTextOutput("breaks")  
)  
#> Error: Type mismatch for `min`, `max`, and `value`.  
#> i All values must have same type: either numeric, Date, or POSIXt.
```

现在，我们可以从错误消息中获取提示，并查看我们输入到`min`、`max`和`value`的每个输入，以查看问题所在：

```
min(datetime)  
#> [1] "2022-08-23 23:09:34 UTC"  
max(datetime)  
#> [1] "2022-09-02 23:09:34 UTC"  
c(min, max)  
#> [[1]]  
#> function (... , na.rm = FALSE) .Primitive("min")  
#>  
#> [[2]]  
#> function (... , na.rm = FALSE) .Primitive("max")
```

现在问题很明显：我们还没有分配`min`和`max`变量，所以我们不小心将`min()`和`max()`函数传递到`sliderInput()`。解决这个问题的一个方法是使用`range()`代替：

```
ui <- fluidPage(  
  sliderInput("slider",  
    "Select Range:",  
    min = min(datetime),  
    max = max(datetime),  
    value = range(datetime)  
,  
  verbatimTextOutput("breaks")  
)
```

这是创建reprex的相当典型的结果：一旦您将问题简化为关键组件，解决方案就变得显而易见。创建一个好的reprex是一种令人难以置信的强大调试技术。

为了简化这个reprex，我不得不做一堆实验和阅读我不太熟悉的函数。如果这是你的reprex，这样做通常要容易得多，因为你已经理解了代码的意图。尽管如此，你经常需要做一堆实验来弄清楚问题到底来自哪里。这可能令人沮丧，而且很耗时，但它有很多好处：

- 它使您能够创建对问题的描述，任何了解Shiny的人都可以访问，而不是任何了解Shiny和您正在工作的特定领域的人。
- 您将建立一个更好的代码工作方式的模型，这意味着您将来不太可能犯相同或类似的错误。
- 随着时间的推移，您将越来越快地创建reprexes，这将成为您在调试时使用的技术之一。
- 即使你没有创建一个完美的reprex，你为改善你的reprex所做的任何工作都比其他人做的工作更少。如果您试图从软件包开发人员那里获得帮助，这一点尤为重要，因为他们通常对时间有很多要求。

当我试图在[RStudio社区](#)上帮助某人使用他们的应用程序时，创建reprex始终是我做的第一件事。这不是我用来摆脱我不想帮助的人的工作练习：这正是我开始的地方！

5.4 Summary

本章为您提供了一些有用的工作流程，用于开发应用程序、调试问题和获得帮助。这些工作流程可能看起来有点抽象，很容易被忽视，因为它们没有具体地改进单个应用程序。但我认为工作流程是我的“秘密”能力之一：我能够完成这么多事情的原因之一是我花时间分析和改进我的工作流程。我强烈建议你也这样做！

关于布局和主题的下一章是实用技术的第一章。无需按顺序阅读；请随意跳到当前应用程序所需的章节。

Chapter 6. Layout, themes, HTML

6.1 Introduction

在本章中，您将解锁一些用于控制应用程序整体外观的新工具。我们将首先讨论页面布局（包括单个和“多个”），这些布局允许您组织输入和输出。然后，您将了解Bootstrap，Shiny使用的CSS工具包，以及如何使用主题自定义其整体视觉外观。最后，我们将简要讨论shiny隐藏发生的事情，这样如果您了解HTML和CSS，您就可以进一步定制Shiny应用程序。

```
library(shiny)
```

6.2 Single page layouts

在第2章中，您了解了构成应用程序交互式组件的输入和输出。但我没有谈论如何在页面上布局它们，相反，我只是使用`fluidPage()`尽快将它们放在一起。虽然这对于学习Shiny来说很好，但它不会创建可用或具有视觉吸引力的应用程序，所以现在是时候学习更多的布局功能了。

布局函数提供了应用程序的高级视觉结构。布局由函数调用的层次结构创建，其中R中的层次结构与生成的HTML中的层次结构相匹配。这有助于您理解布局代码。例如，当您查看这样的布局代码时：

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

关注函数调用的层次结构：

```
fluidPage(  
  titlePanel(),  
  sidebarLayout(  
    sidebarPanel(),  
    mainPanel()  
)  
)
```

即使你还没有学会这些功能，你也可以通过阅读它们的名字来猜测发生了什么。您可能会想象此代码将生成一个经典的应用程序设计：顶部的标题栏，然后是边栏（包含滑块）和主面板（包含绘图）。通过缩进轻松查看层次结构的能力是使用一致样式的好主意的原因之一。

在本节的其余部分，我将讨论帮助您设计单页应用程序的功能，然后在下一节中进入多页应用程序。我还建议查看 Shiny[应用程序布局指南](#)；它有点过时，但包含一些有用的技术。

6.2.1 Page functions

最重要但最不有趣的布局函数是 `fluidPage()`。到目前为止，您在几乎所有示例中都看到了它。但它在做什么，如果你单独使用它，会发生什么？图6.1显示了结果：它看起来是一个非常无聊的应用程序，但幕后有很多，因为 `fluidPage()` 设置了Shiny需要的所有HTML、CSS和JavaScript。

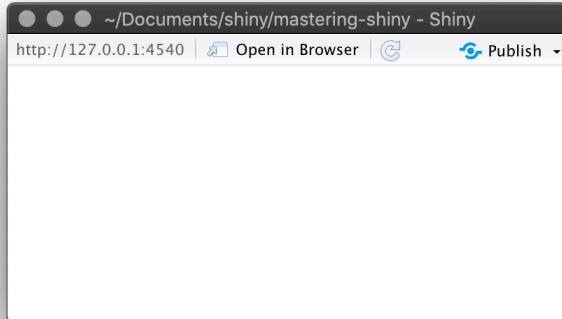


图6.1：仅由 `fluidPage()` 组成的UI

除了 `fluidPage()`，Shiny还提供了其他一些页面功能，这些功能在更专业的情况下可以派上用场：

`fixedPage()` 和 `fillPage()`。`fixedPage()` 的工作原理与 `fluidPage()` 类似，但具有固定的最大宽度，这可以防止您的应用程序在更大的屏幕上变得不合理的宽。`fillPage()` 填充了浏览器的全部高度，如果您想制作一个占据整个屏幕的图，则非常有用。您可以在他们的文档中找到详细信息。

6.2.2 Page with sidebar

要制作更复杂的布局，您需要在 `fluidPage()` 内调用布局函数。例如，要制作两列布局，输入在左侧，输出在右侧，您可以使用 `sidebarLayout()` 以及它的 `titlePanel()`, `sidebarPanel()` 和 `mainPanel()`。以下代码显示了生成图6.2的基本结构。

```

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

```

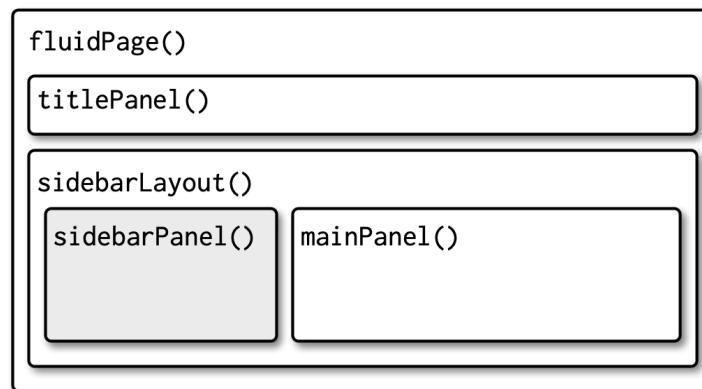


图6.2：带边栏的基本应用程序的结构

为了使其更逼真，让我们添加一个输入和输出，以创建一个非常简单的应用程序，演示中心极限定理，如图[6.3](#)所示。如果您自己运行此应用程序，您可以增加样本数量，以查看分布图形的变化。

```

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)
}

```

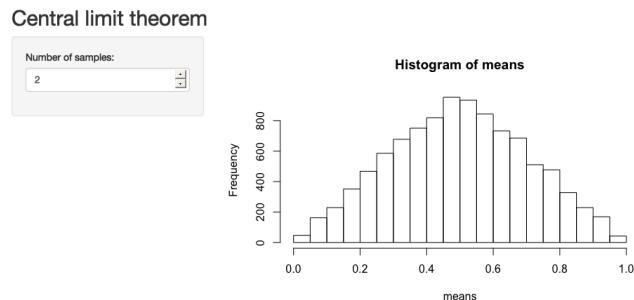


图6.3：一个常见的应用程序设计是将控件放在边栏中，并在主面板中显示结果

6.2.3 Multi-row

在shiny布局中，`sidebarLayout()` 建立在灵活的多行布局之上，您可以直接使用该布局来创建视觉上更复杂的应用程序。像往常一样，您从`fluidPage()`开始。然后，您使用`fluidRow()`创建行，使用`column()`以下模板生成图6.4所示的结构。

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

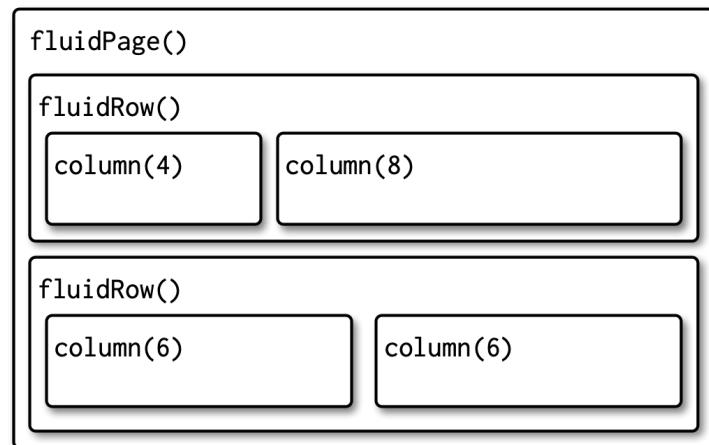


图6.4：简单多行应用程序的底层结构

每行由12列组成，`column()`的第一个参数给出了要占用的列数。12列布局为您提供了相当大的灵活性，因为您可以轻松创建2列、3列或4列布局，或使用窄列创建间隔。您可以在第4.4节中看到此布局的示例。

如果您想了解有关使用网格系统设计的更多信息，我强烈推荐有关该主题的经典文本：Josef Müller-Brockman的“[平面设计中的网格系统](#)”。

6.2.4 Exercises

1. 阅读`sidebarLayout()`的文档，以确定边栏和主面板的宽度（以列为单位）。你能使用`fluidRow()`和`column()`重新创建它的外观吗？你错过了什么？
2. 修改中央极限定理应用程序，将边栏放在右侧而不是左侧。
3. 创建一个包含两个图块的应用程序，每个图块占宽度的一半。将控件放在绘图下方的全宽容器中。

6.3 Multi-page layouts

随着您的应用程序的复杂性增长，不可能将所有内容都放在一个页面上。在本节中，您将学习`tabPanel()`的各种用途，这些用途会造成多个页面的错觉。这是一种错觉，因为您仍然会有一个具有单个底层HTML文件的应用程序，但它现在被分解成碎片，一次只能看到一块。

多页应用程序与模块配对特别好，您将在第19章中了解这一点。模块允许您以与划分用户界面相同的方式，对`server`函数进行分组。然后，通过定义明确的连接，创建进行交互的独立组件。

6.3.1 Tabsets

将页面分解成碎片的简单方法是使用`tabsetPanel()`及其配对的`tabPanel()`。正如您在下面的代码中看到的，`tabsetPanel()`为任意数量的`tabPanel()`创建一个容器，而`tabPanel()`又可以包含任何其他HTML组件。图6.5显示了一个简单的例子。

```
ui <- fluidPage(  
  tabsetPanel(  
    tabPanel("Import data",  
      fileInput("file", "Data", buttonLabel = "Upload..."),  
      textInput("delim", "Delimiter (leave blank to guess)", ""),  
      numericInput("skip", "Rows to skip", 0, min = 0),  
      numericInput("rows", "Rows to preview", 10, min = 1)  
    tabPanel("Set parameters"),  
    tabPanel("Visualise results")  
)
```

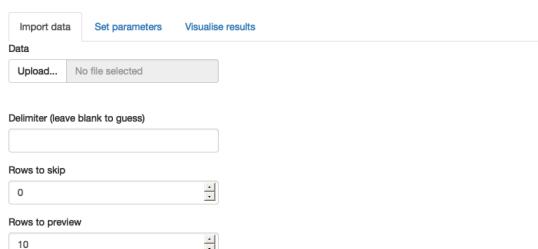


图6.5：`tabsetPanel()`允许用户选择单个`tabPanel()`进行查看

如果您想知道用户选择了哪个选项卡，您可以向 `tabsetPanel()` 提供 `id` 参数，它将成为输入。图6.6显示了这一点。

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      textOutput("panel")  
    ),  
    mainPanel(  
      tabsetPanel(  
        id = "tabset",  
        tabPanel("panel 1", "one"),  
        tabPanel("panel 2", "two"),  
        tabPanel("panel 3", "three")  
      )  
    )  
  )  
)  
server <- function(input, output, session) {  
  output$panel <- renderText({  
    paste("Current panel: ", input$tabset)  
  })  
}
```



图6.6：当您使用 `id` 参数时，选项卡集成为输入。这允许您根据当前可见的选项卡使应用程序的行为不同。

注意：`tabsetPanel()` 可以在应用程序中的任何地方使用；将选项卡集嵌套在其他组件（包括选项卡！）中是完全可以的。

6.3.2 Navlists and navbars

由于选项卡是水平显示的，因此您可以使用多少选项卡有基本限制，特别是当它们具有长标题时。`navbarPage()` 和 `navbarMenu()` 提供了两种替代布局，允许您使用更多具有较长标题的选项卡。

`navlistPanel()` 类似于 `tabsetPanel()`，但它不是水平运行标签标题，而是在边栏中垂直显示它们。它还允许您添加带有普通字符串的标题，如以下代码所示，该代码生成图6.7。

```

ui <- fluidPage(
  navlistPanel(
    id = "tabset",
    "Heading 1",
    tabPanel("panel 1", "Panel one contents"),
    "Heading 2",
    tabPanel("panel 2", "Panel two contents"),
    tabPanel("panel 3", "Panel three contents")
  )
)

```



图6.7: `navlistPanel()` 垂直而不是水平显示选项卡标题。

另一种方法是使用 `navbarPage()` 它仍然水平运行选项卡标题，但您可以使用 `navbarMenu()` 添加下拉菜单以增加层次结构。图6.8显示了一个简单的示例。

```

ui <- navbarPage(
  "Page title",
  tabPanel("panel 1", "one"),
  tabPanel("panel 2", "two"),
  tabPanel("panel 3", "three"),
  navbarMenu("subpanels",
    tabPanel("panel 4a", "four-a"),
    tabPanel("panel 4b", "four-b"),
    tabPanel("panel 4c", "four-c")
  )
)

```

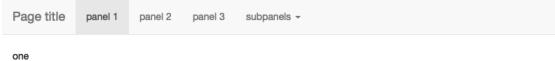


图6.8: `navbarPage()` 在页面顶部设置一个水平导航栏。

这些布局使您能够创建丰富且令人满意的程序。要走得更远，您需要了解有关底层设计系统的更多信息。

6.4 Bootstrap

要继续您的应用程序定制之旅，您需要了解更多关于Shiny使用的Bootstrap框架的信息。Bootstrap是HTML约定、CSS样式和JS片段的集合，捆绑成一个方便的形式。Bootstrap源于最初为Twitter开发的框架，在过去的10年里，它已经发展成为网络上最受欢迎的CSS框架之一。Bootstrap在R中也很受欢迎——您无疑已经看到许多由 `rmarkdown::html_document()` 生成的文档，并使用了 `pkgdown` 制作的许多软件包网站，两者都使用Bootstrap。

作为一名Shiny开发人员，您不需要过多考虑Bootstrap，因为Shiny函数会自动为您生成Bootstrap兼容的HTML。但很高兴知道Bootstrap存在，因为那时：

- 您可以使用 `bslib::bs_theme()` 自定义代码的视觉外观，第6.5节。
- 您可以使用 `class` 参数，设置Bootstrap类名称自定义一些布局、输入和输出，正如您在第2.2.7节中看到的那样。
- 您可以制作自己的函数，来生成Shiny不提供的Bootstrap组件，如“[实用程序类](#)”中所述。

你也可以使用完全不同的CSS框架。一些现有的R包，通过将流行的替代品包装到Bootstrap来简化：

- [shiny.semantic](#)，由[Appslon制作](#)，建立在[formantic UI](#)之上。
- [RInterface](#)的[shinyMobile](#)建立在[framework 7](#)之上，专为移动应用程序设计。
- [埃里克·安德森 \(Eric Anderson\)](#) 的[Shinymaterial](#)建立在谷歌[Material design](#)框架之上。
- 同样由RStudio设计的[shinydashboard](#)提供了一个旨在创建仪表板的布局系统。

您可以在<https://github.com/nanxstats/awesome-shiny-extensions>上找到更完整且积极维护的列表。

6.5 Themes

Bootstrap在R社区中无处不在，很容易感到风格疲劳：一段时间后，每个Shiny应用程序和Rmd开始看起来都一样。解决方案是使用`bslib`包进行设计主题。`bslib`是相对较新的软件包，允许您覆盖许多Bootstrap默认值，以创建您独特的外观。当我写这篇文章时，`bslib`主要仅适用于Shiny，但正在进行中，以将其增强版的主题包装到RMarkdown、`pkgdown`中。

如果您正在为公司制作应用程序，我强烈建议您在制作上花一点时间—将您的应用程序匹配企业风格。

6.5.1 Getting started

使用`bslib::bs_theme()`创建一个主题，然后将其应用于具有页面布局函数`theme`参数的应用程序：

```
fluidPage(  
  theme = bslib::bs_theme(...)  
)
```

如果没有指定，Shiny将使用自创建以来基本使用的经典Bootstrap v3主题。默认情况下，`bslib::bs_theme()`将使用Bootstrap v4。如果您只使用内置组件，使用Bootstrap v4而不是v3不会造成问题。如果您使用了自定义HTML，它可能会造成问题，因此您可以强制其保留在`version = 3`的v3中。

6.5.2 Shiny themes

更改应用程序整体外观的最简单方法是使用`bslib:: bs_theme()`的`bootswatch`参数选择一个预制的“bootswatch”主题。图6.9显示了以下代码的结果，将“darkly”切换为其他主题。

```
ui <- fluidPage(  
  theme = bslib::bs_theme/bootswatch = "darkly"),  
  sidebarLayout(  
    sidebarPanel(  
     textInput("txt", "Text input:", "text here"),  
      sliderInput("slider", "Slider input:", 1, 100, 30)  
    ),  
    mainPanel(
```

```

h1(paste0("Theme: darkly")),
h2("Header 2"),
p("Some text")
)
)
)

```

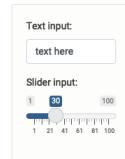
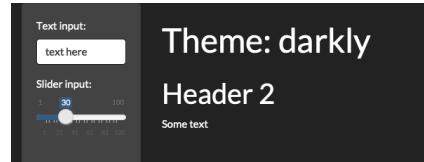


图6.9：具有四个bootswatch主题的相同应用程序：黑暗（darkly）、平坦（flatly）、砂岩（sandstone）和单一（united）

或者，您可以使用 `bs_theme()` 的其他参数，构建自己的主题，如 `bg`（背景颜色）、`fg`（前景颜色）和 `base_font`：

```

theme <- bslib::bs_theme(
  bg = "#0b3d91",
  fg = "white",
  base_font = "Source Sans Pro"
)

```

预览和自定义主题的简单方法是使用 `bslib::bs_theme_preview(theme)`。这将打开一个shiny应用程序，显示应用许多标准控件时主题的样子，并为您提供用于自定义最重要参数的交互式控件。

6.5.3 Plot themes

如果您已经大量定制了应用程序的样式，您可能还需要自定义您的绘图样式。幸运的是，这真的很容易，这要归功于[thematic](#)包，该包会自动为ggplot2、lattice和base-plots匹配适合的主题样式。只需在server函数中调用[thematic_shiny\(\)](#)。这将自动确定应用程序主题的所有设置，如图6.10所示。

```
library(ggplot2)

ui <- fluidPage(
  theme = bslib::bs_theme(bootswatch = "darkly"),
  titlePanel("A themed plot"),
  plotOutput("plot"),
)

server <- function(input, output, session) {
  thematic::thematic_shiny()

  output$plot <- renderPlot({
    ggplot(mtcars, aes(wt, mpg)) +
      geom_point() +
      geom_smooth()
  }, res = 96)
}
```

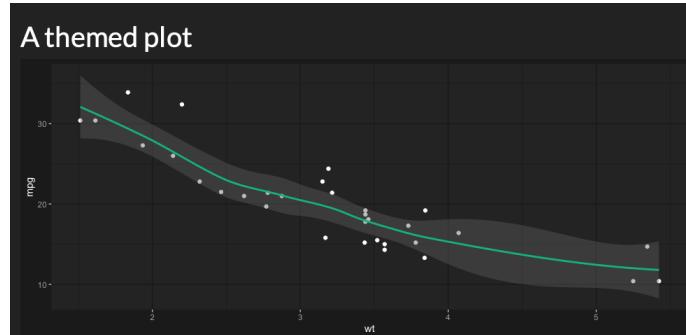


图6.10：使用[thematic::thematic_shiny\(\)](#)确保ggplot2自动匹配应用程序主题

6.5.4 Exercises

1. 使用[bslib::bs_theme_preview\(\)](#)来制作最丑陋的主题。

6.6 Under the hood

Shiny的设计是，作为R用户，您无需了解HTML的细节。但是，如果您知道一些HTML和CSS，则可以进一步定制Shiny。不幸的是，教授HTML和CSS超出了这本书的范围，但一个很好的开始是MDN的[HTML](#)和[CSS基础](#)教程。

最重要的是，所有输入、输出和布局函数背后都没有魔力：它们只是生成HTML。您可以通过直接在控制台中执行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>
```

请注意，这是`<body>`标签的内容；Shiny的其他部分负责生成`<head>`。如果您想包含额外的CSS或JS依赖项，您需要学习`htmltools::htmlDependency()`。两个好的开始是<https://blog.r-hub.io/2020/08/25/js-r/#web-dependency-management>和<https://unleash-shiny.rinterface.com/htmltools-dependencies.html>。

可以将您自己的HTML添加到`ui`中。一种方法是将HTML代码与`HTML()`函数一起包含。在下面的示例中，我使用“原始字符串常量”（`r"()"`），以便更容易在字符串中包含引号：

```
ui <- fluidPage(  
  HTML(r"(  
    <h1>This is a heading</h1>  
    <p class='my-class'>This is some text!</p>  
    <ul>  
      <li>First bullet</li>  
      <li>Second bullet</li>  
    </ul>  
)"))
```

如果您是HTML/CSS专家，您可以完全跳过`fluidPage()`，并提供原始HTML。有关更多详细信息，请参阅“[使用HTML构建您的整个UI](#)”。

或者，您可以使用Shiny提供的HTML助手。最重要的元素（如`h1()`和`p()`）都具有规则函数，所有其他函数都可以通过其他标签助手访问。命名参数成为属性，未命名参数成为子参数，因此我们可以将上述HTML重新创建为：

```
ui <- fluidPage(  
  h1("This is a heading"),  
  p("This is some text", class = "my-class"),  
  tags$ul(  
    tags$li("First bullet"),  
    tags$li("Second bullet"))  
)
```

使用代码生成HTML的一个优点是，您可以将现有的Shiny组件交织到自定义结构中。例如，下面的代码制作了一个包含两个输出的文本段落，一个是粗体的：

```
tags$p(
  "You made ",
  tags$b("$", textOutput("amount", inline = TRUE)),
  " in the last ",
  textOutput("days", inline = TRUE),
  " days "
)
```

请注意 `inline = TRUE` 的使用；`textOutput()` 默认是制作一个完整的段落。

要了解有关使用HTML、CSS和JavaScript制作引人注目的用户界面的更多信息，我强烈推荐David Granjon的[杰出用户界面与Shiny](#)。

6.7 Summary

本章为您提供了制作复杂而有吸引力的shiny应用程序所需的工具。您已经学习了Shiny功能，这些功能允许您布局单页和多页应用程序（如 `fluidPage()` 和 `tabsetPanel()`），以及如何使用主题自定义整体视觉外观。您还了解了shiny隐藏的正在发生的事情：您知道Shiny使用Bootstrap，输入和输出函数只是返回HTML，您也可以自己创建HTML。

在下一章中，您将了解更多关于应用程序的另一个重要视觉组件：图形。

Chapter 7. Graphics

我们在第2章中简要讨论了 `renderPlot()`；它是在您的应用程序中显示图形的强大工具。本章将向您展示如何充分利用它来创建交互式绘图，响应鼠标事件的绘图。您还将学习其他一些有用的技术，包括制作具有动态宽度和高度的绘图，以及使用 `renderImage()` 显示图像。

在本章中，我们需要ggplot2和Shiny，因为这是我将用于大多数图形的。

```
library(shiny)
library(ggplot2)
```

7.1 Interactivity

`plotOutput()` 最酷的事情之一是，除了显示绘图的输出外，它也可以是响应指针事件的输入。这允许您创建交互式图形，用户在其中直接与绘图上的数据进行交互。交互式图形是一个强大的工具，具有广泛的应用。我没有空间向您展示所有的可能性，所以在这里我将专注于基础知识，然后向您指出资源，以了解更多信息。

7.1.1 Basics

绘图可以响应四个不同的鼠标事件：`click`（单击）、`dblclick`（双击）、`hover`（当鼠标在同一位置停留一段时间时）和`brush`（矩形选择工具）。要将这些事件转换为shiny的输入，您向相应的 `plotOutput()` 参数提供一个字符串，例如 `plotOutput("plot", click = "plot_click")`，这会创建一个 `input$plot_click`，您可以使用它来处理绘图上的鼠标点击。

这是一个处理鼠标点击的非常简单的示例。我们注册了 `plot_click` 输入，然后使用鼠标点击的坐标，更新输出。[图7.1](#)显示了结果。

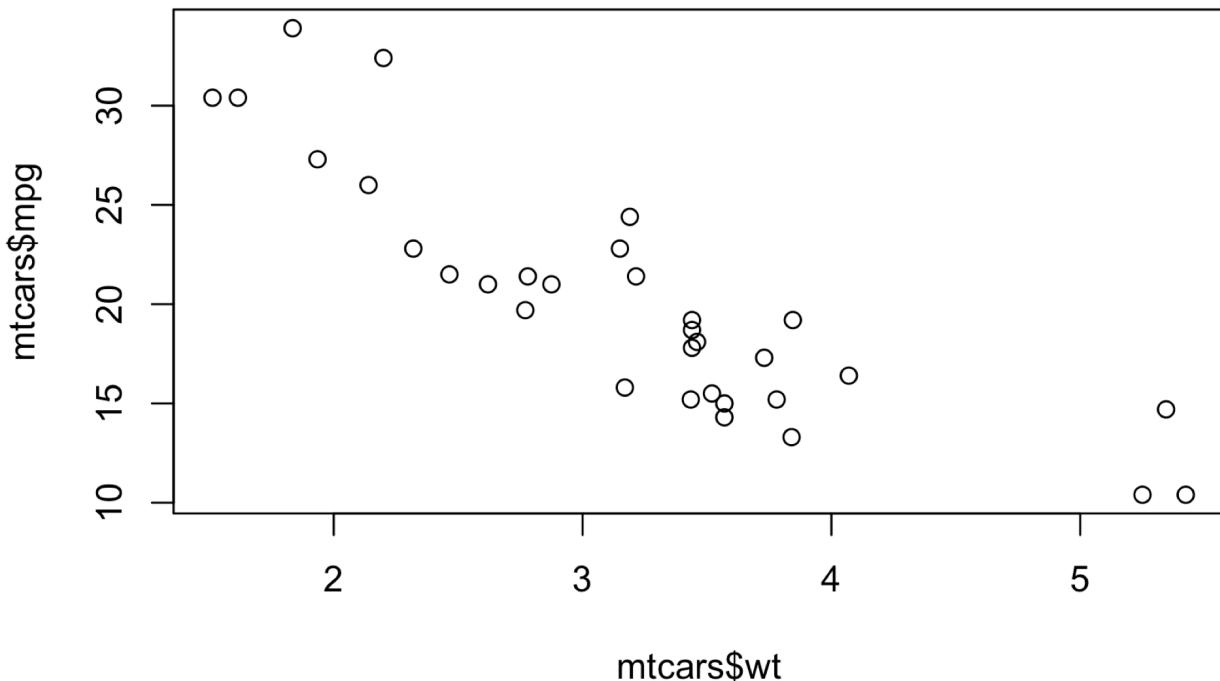
```

ui <- fluidPage(
  plotOutput("plot", click = "plot_click"),
  verbatimTextOutput("info")
)

server <- function(input, output) {
  output$plot <- renderPlot({
    plot(mtcars$wt, mtcars$mpg)
  }, res = 96)

  output$info <- renderPrint({
    req(input$plot_click)
    x <- round(input$plot_click$x, 2)
    y <- round(input$plot_click$y, 2)
    cat("[ ", x, ", ", y, " ]", sep = "")
  })
}

```



[1.51, 30.4]

图7.1：单击左上角，更新打印的坐标，见<https://hadley.shinyapps.io/ms-click>。

(注意使用 `req()` 以确保应用程序在第一次单击之前不会做任何事情，并且坐标是以底层 `wt` 和 `mpg` 变量为单位的。)

以下各节更详细地描述了这些事件。我们将从点击事件开始，然后简要讨论密切相关的`dblclick`和`hover`。然后，您将了解`brush`事件，该事件提供了一个由其四边（`xmin`、`xmax`、`ymin`和`ymax`）定义的矩形“画笔”。然后，我将举几个基于点击事件，更新绘图的例子，然后讨论Shiny中交互式图形的一些局限性。

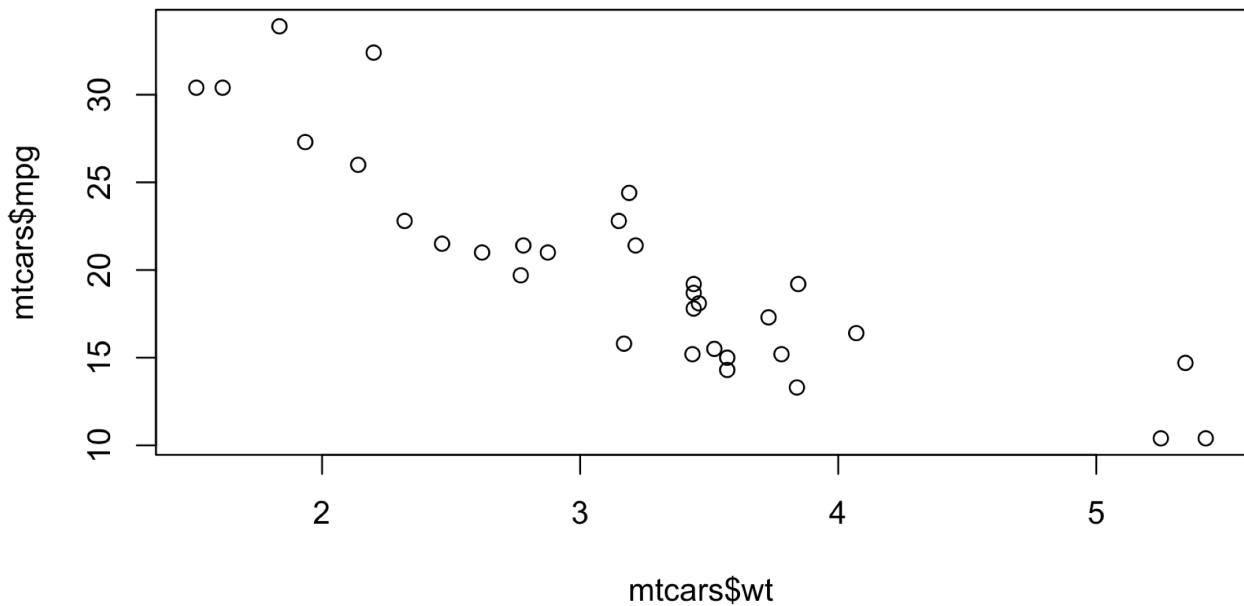
7.1.2 Clicking

点事件返回一个包含大量信息的相对丰富的列表。最重要的组件是`x`和`y`，它们给出了事件在数据坐标中的位置。但我不打算谈论这个数据结构，因为您只需要在相对罕见的情况下（如果您确实想要详细信息，请在Shiny gallery中使用[此应用程序](#)）。相反，您将使用`nearPoints()`助手，该助手返回一个包含点击附近行的数据框，处理一堆繁琐的细节。

以下是`nearPoints()`在操作的简单示例，显示了有关事件附近点的数据表。图7.2显示了该应用程序的屏幕截图。

```
ui <- fluidPage(
  plotOutput("plot", click = "plot_click"),
  tableOutput("data")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    plot(mtcars$wt, mtcars$mpg)
  }, res = 96)

  output$data <- renderTable({
    nearPoints(mtcars, input$plot_click, xvar = "wt", yvar = "mpg")
  })
}
```



mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
30.40	4.00	95.10	113.00	3.77	1.51	16.90	1.00	1.00	5.00	2.00

图7.2: `nearPoints()` 将绘图坐标转换为数据行, 从而可以轻松显示您点击的实时查看的点的基础数据。 <https://hadley.shinyapps.io/ms-nearPoints>

在这里, 我们给 `nearPoints()` 四个参数: 绘图数据、输入事件和绘图使用的x-y轴上变量的名称。如果您使用 `ggplot2`, 您只需提供前两个参数, 因为 `xvar` 和 `yvar` 可以自动从绘图数据结构中获取。出于这个原因, 我将在本章的其余部分使用 `ggplot2` 重新实现的示例:

```
ui <- fluidPage(
  plotOutput("plot", click = "plot_click"),
  tableOutput("data")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    ggplot(mtcars, aes(wt, mpg)) + geom_point()
  }, res = 96)

  output$data <- renderTable({
    req(input$plot_click)
    nearPoints(mtcars, input$plot_click)
  })
}
```

您可能想知道 `nearPoints()` 到底返回了什么。这是使用 `browser()` 的好地方, 我们在第5.2.3节中讨论了这一点:

```

...
output$data <- renderTable({
  req(input$plot_click)
  browser()
  nearPoints(mtcars, input$plot_click)
})
...

```

现在，在我启动应用程序并单击一个点后，转向后端交互式调试器界面，在那里我可以运行 `nearPoints()`，看看它返回什么：

```

nearPoints(mtcars, input$plot_click)
#>          mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Datsun 710 22.8     4   108 93 3.85 2.32 18.61   1   1     4     1

```

另一种使用 `nearPoints()` 的方法是使用 `allRows = TRUE` 和 `addDist = TRUE`。这将返回带有两个新列的原始数据框：

- `dist_` 给出行和事件之间的距离（以像素为单位）。换句话说，当 `addDist=TRUE` 时，将返回所有行（用于绘图的 x 和 y 组成的点）到该点击事件坐标的距离，是一个与行数相等的距离值向量；否则，只返回一行距离点击事件最近的行（形成的数据框）。
- `selected_` 表明它是否靠近点击事件（即当 `allRows = FALSE`，它是否会返回一行）。换句话说，当 `allRows=TRUE` 时，将返回一个与行数相等的逻辑值向量，`TRUE` 表示被选中，`FALSE` 表示未被选中；否则，只返回选中行的数据框。

我们稍后会看到一个例子。

7.1.3 Other point events

同样的方法，同样适用于 `click`、`dblclick` 和 `hover`：只需更改参数的名称即可。如果需要，您可以通过提供 `clickOpts()`、`dblclickOpts()` 或 `hoverOpts()`，而不是提供输入 ID 的字符串，来获得对事件的额外控制。这些很少需要，所以我不会在这里讨论它们；详情请参阅文档。

您可以在一个图中使用多种交互类型。只需确保向用户解释他们可以做什么：使用鼠标事件与应用程序交互的一个缺点是它们不能立即被发现。

7.1.4 Brushing

在绘图上，选择点的另一种方法是使用 `brush`，即由四条边定义的矩形选择。在 Shiny 中，一旦你掌握了 `click` 和 `nearPoints()`，使用 `brush` 就很简单：你只需切换到 `brush` 参数和 `brushedPoints()` 助手。

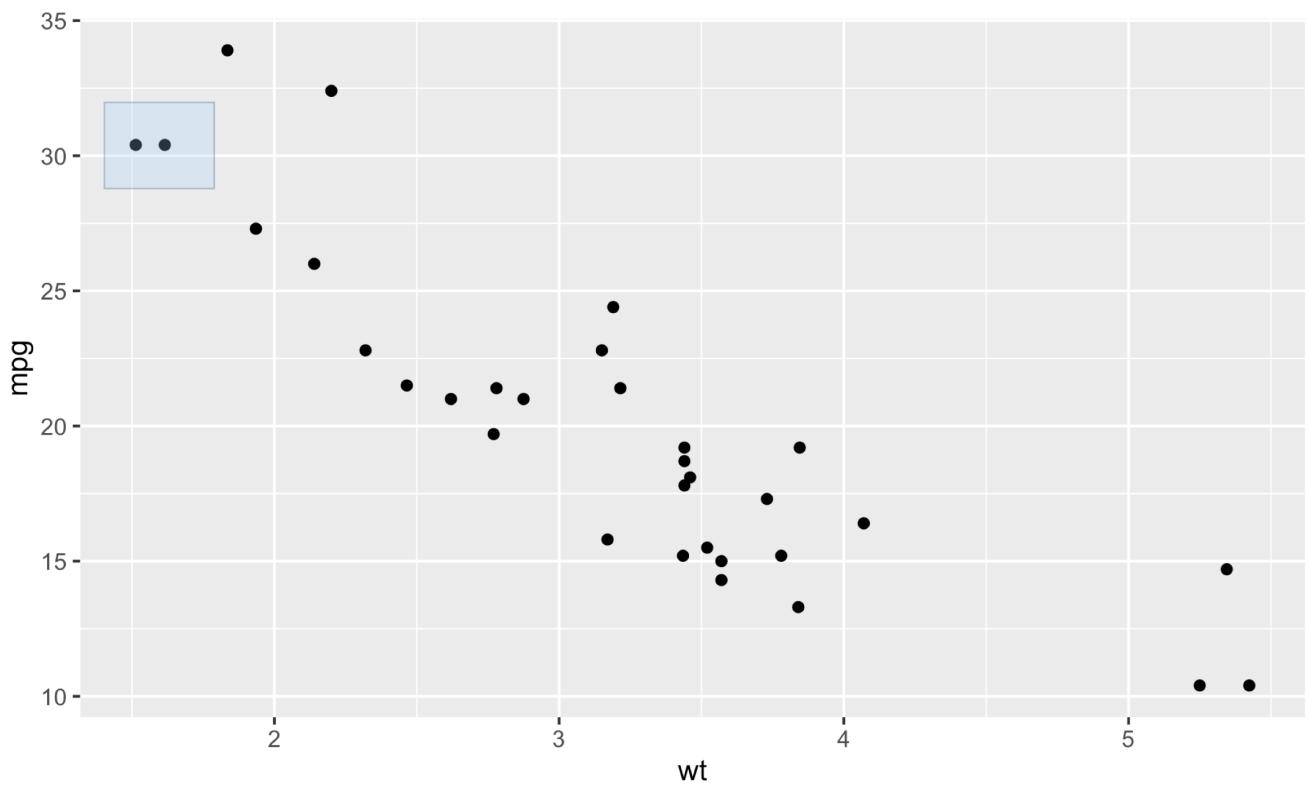
这是另一个简单的示例，显示了 `brush` 选择了哪些点。图 7.3 显示了结果。

```

ui <- fluidPage(
  plotOutput("plot", brush = "plot_brush"),
  tableOutput("data")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    ggplot(mtcars, aes(wt, mpg)) + geom_point()
  }, res = 96)

  output$data <- renderTable({
    brushedPoints(mtcars, input$plot_brush)
  })
}

```



mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
30.40	4.00	75.70	52.00	4.93	1.61	18.52	1.00	1.00	4.00	2.00
30.40	4.00	95.10	113.00	3.77	1.51	16.90	1.00	1.00	5.00	2.00

图7.3：设置brush参数为用户提供了可拖动的“brush”。在这个应用程序中，brush下面的点显示在表格中。在<https://hadley.shinyapps.io/ms-brushedPoints>上观看动态效果。

使用 `brushOpts()` 控制颜色（`fill` 和 `stroke`），或将brush限制为单一维度，`direction = "x"` 或 `"y"`”（例如，用于brush时间序列）。

7.1.5 Modifying the plot

到目前为止，我们已经在另一个输出中显示了交互的结果。但互动的真正之美来自于你展示与你互动的绘图的变化。不幸的是，这需要您尚未了解的高级反应技术：`reactiveVal()`。我们将在第16章的`reactiveVal()`讲解，但我想在这里展示它，因为它是一种非常有用的技术。读完该章后，您可能需要重新阅读本节，但希望即使没有所有理论，您也会了解潜在的应用。

正如您从名称中猜到的，`reactiveVal()`与`reactive()`相当相似。您可以通过调用具有初始值的`reactiveVal()`，创建一个具有初始值的反应式表达式（以下简称为反应值），并以与反应式表达式相同的方式检索该值：

```
val <- reactiveVal(10)
val()
#> [1] 10
```

最大的区别在于，您还可以更新反应值，所有引用它的反应式表达式都会重新计算。反应值使用特殊语法进行更新——您像函数一样调用它，第一个参数是新值：

```
val(20)
val()
#> [1] 20
```

这意味着使用当前值更新反应值，如下所示：

```
val(val() + 1)
val()
#> [1] 21
```

不幸的是，如果您真的尝试在控制台中运行此代码，您将收到错误，因为它必须在反应式表达式环境中运行。这使得实验和调试更具挑战性，因为您需要`browser()`或类似功能在对`shinyApp()`的调用中暂停执行。这是我们在第16章稍后讨论的挑战之一。

现在，让我们把学习`reactiveVal()`的挑战放在一边，并向您展示如何使用该技术。想象一下，你想可视化点击和图上点之间的距离。在下面的应用程序中，我们首先创建一个反应值来存储这些距离，用一个常量初始化它，在我们单击任何东西之前将使用。然后，我们使用`observeEvent()`在单击鼠标时更新反应值，以及一个用点大小可视化距离的ggplot。总而言之，这看起来像下面的代码，结果如图7.4所示。

```
set.seed(1014)
df <- data.frame(x = rnorm(100), y = rnorm(100))

ui <- fluidPage(
  plotOutput("plot", click = "plot_click", )
)
server <- function(input, output, session) {
  dist <- reactiveVal(rep(1, nrow(df)))
  observeEvent(input$plot_click,
    dist(nearPoints(df, input$plot_click, allRows = TRUE, addDist = TRUE)$dist_)
  )
}
```

```

output$plot <- renderPlot({
  df$dist <- dist()
  ggplot(df, aes(x, y, size = dist)) +
    geom_point() +
    scale_size_area(limits = c(0, 1000), max_size = 10, guide = NULL)
}, res = 96)
}

```

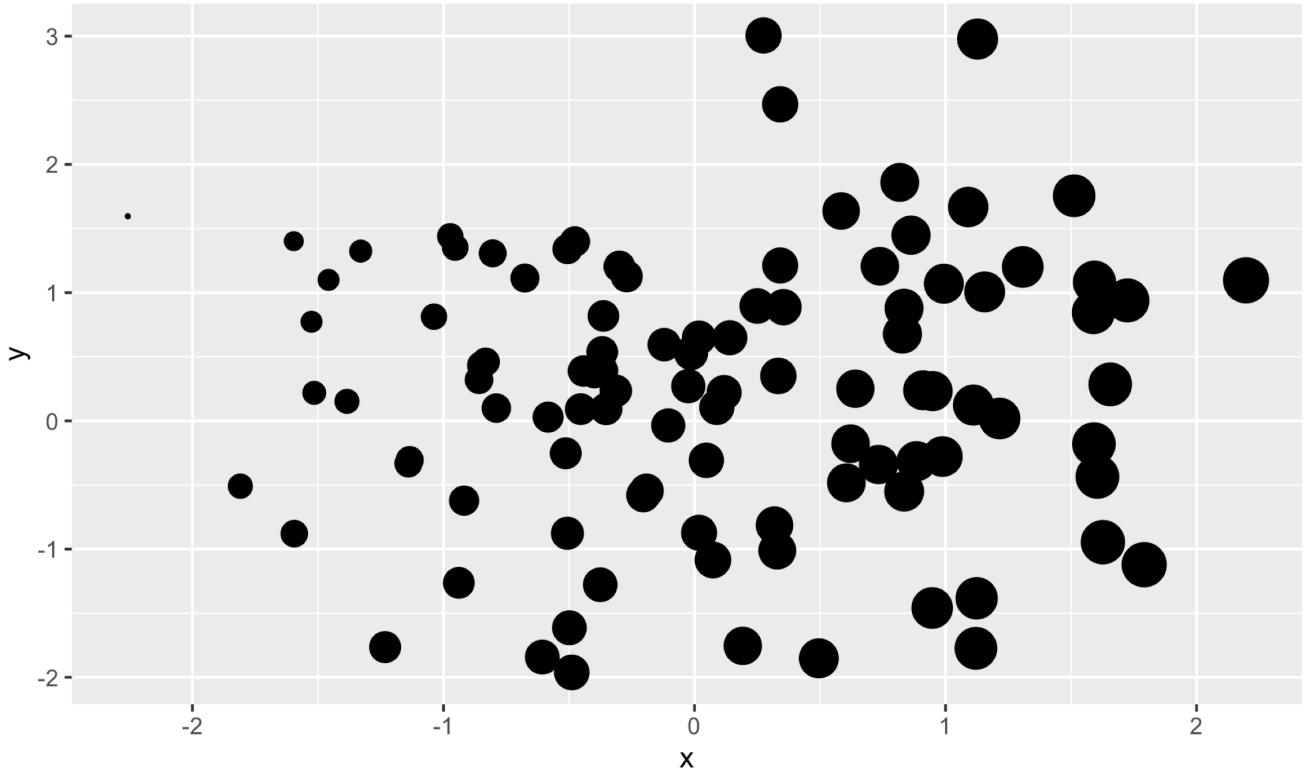


图7.4：此应用程序使用 `reactiveVal()` 来存储到上次单击的点的距离，然后将其映射到点大小。在这里，我显示了点击最左边一个点的结果，见<https://hadley.shinyapps.io/ms-modifying-size>。

这里需要注意两种重要的ggplot2技术：

- 在绘制之前，我会将距离添加到数据框中。我认为在可视化之前，将相关变量放在一个数据框中是一个很好的做法。
- 我将 `limits` 设置为 `scale_size_area()`，以确保大小在点击之间具有可比性。为了找到正确的范围，我做了一些互动实验，但如果需要，你可以计算出确切的细节（见本章末尾的练习）。

这里有一个更复杂的想法。我想用brush在选择中逐步添加点。在这里，我使用不同的颜色显示选择，但您可以想象许多其他应用程序。为了使这项工作有效，我将 `reactiveVal()` 初始化为 `FALSE` 向量，然后使用 `brushedPoints()` 和 `|` 将brush下的任何点添加到选择中。为了给用户一些重新开始的方式，我双击重置选择。图7.5显示了正在运行的应用程序的几张屏幕截图。

```

ui <- fluidPage(
  plotOutput("plot", brush = "plot_brush", dblclick = "plot_reset")
)
server <- function(input, output, session) {
  selected <- reactiveVal(rep(FALSE, nrow(mtcars)))
  observeEvent(input$plot_brush, {
    ...
  })
}

```

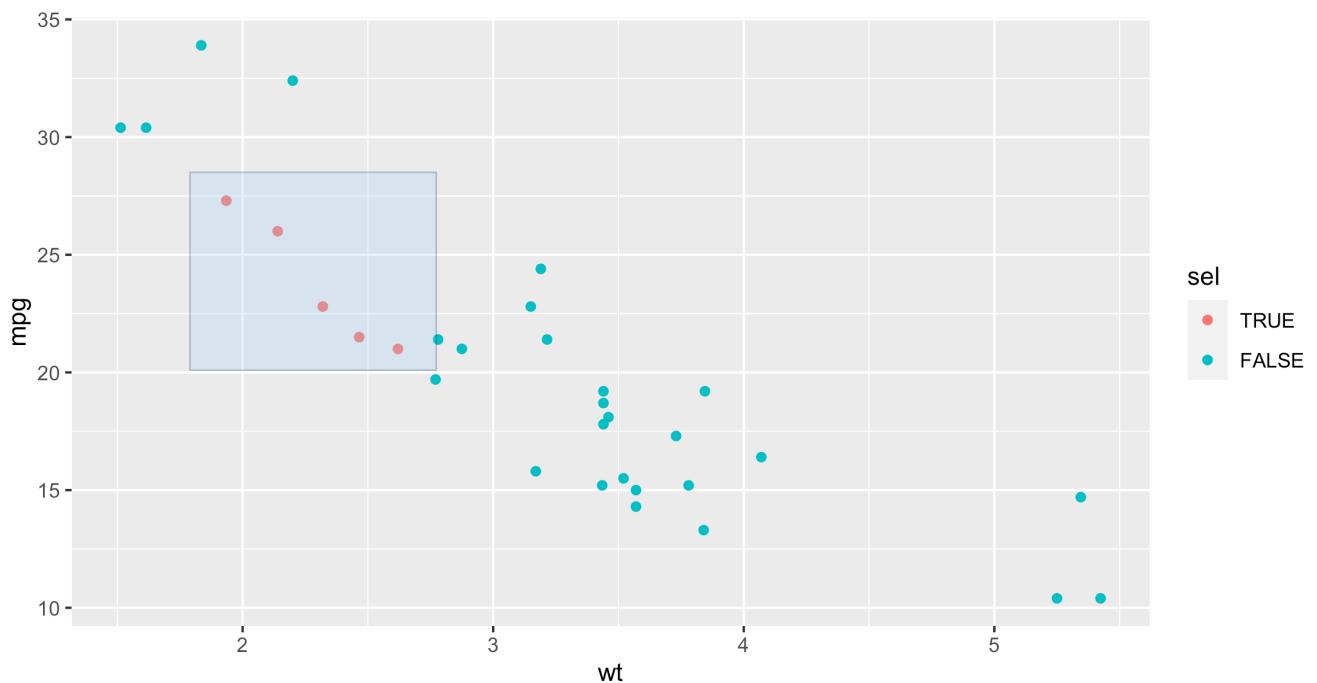
```

brushed <- brushedPoints(mtcars, input$plot_brush, allRows = TRUE)$selected_
selected(brushed | selected())
})

observeEvent(input$plot_reset, {
  selected(rep(FALSE, nrow(mtcars)))
})

output$plot <- renderPlot({
  mtcars$sel <- selected()
  ggplot(mtcars, aes(wt, mpg)) +
    geom_point(aes(colour = sel)) +
    scale_colour_discrete(limits = c("TRUE", "FALSE"))
}, res = 96)
}

```



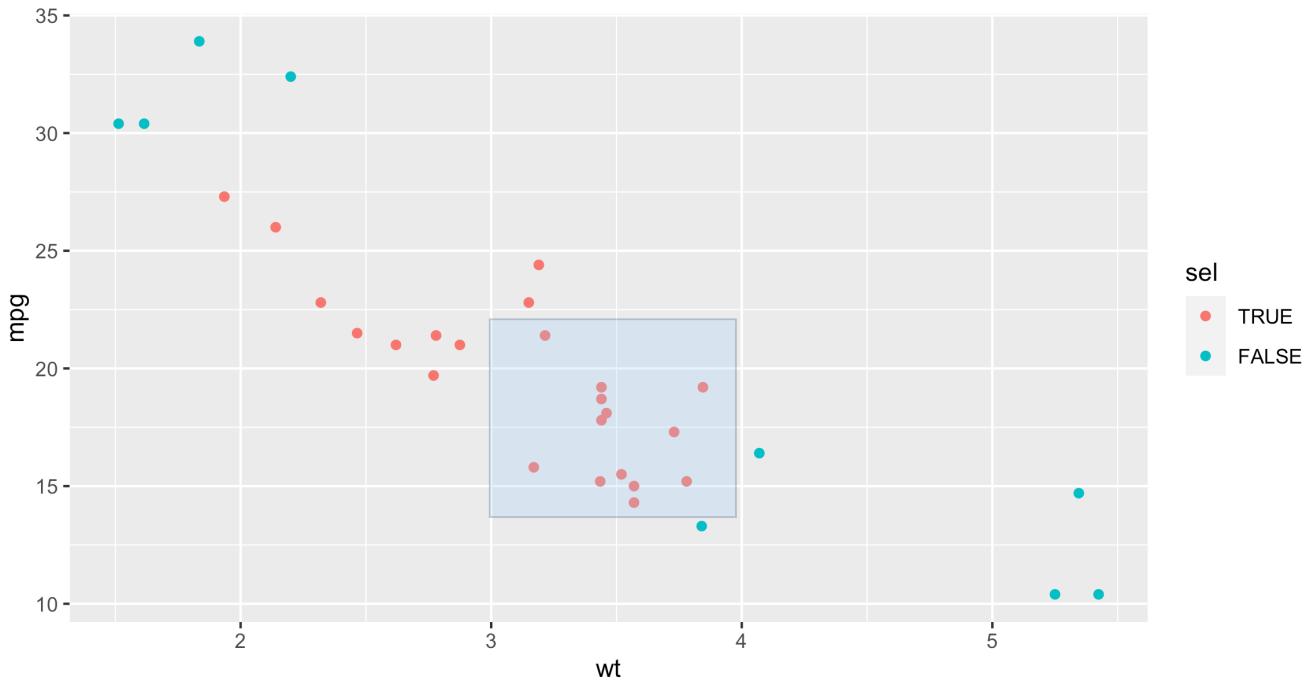


图7.5：此应用程序使brush “persistent”，因此拖动它会增加当前选择。

我再次设置了比例的限制，以确保图例（和颜色）在第一次点击后不会改变。

7.1.6 Interactivity limitations

在我们继续之前，了解交互式绘图中的基本数据流很重要，以了解其局限性。基本流程是这样的：

1. JavaScript捕获鼠标事件。
2. Shiny将鼠标事件数据发回R，告诉应用程序输入现已过期。
3. 所有下游反应式表达式都被重新计算。
4. `plotOutput()`生成一个新的PNG并将其发送到浏览器。

对于本地应用程序来说，瓶颈往往是绘图所花费的时间。根据绘图的复杂程度，这可能需要显著的几分之一秒。但对于托管应用程序，您还必须考虑将事件从浏览器传输到R，然后将渲染的绘图从R传输到浏览器所需的时间。

一般来说，这意味着不可能创建Shiny应用程序，其中动作和响应被视为即时的（即绘图似乎与您的动作同时更新）。如果您需要这种速度，您必须在JavaScript中执行更多计算。一种方法是使用包装JavaScript图形库的R包。现在，当我写这本书时，我认为你会获得plotly包的最佳体验，正如Carson Sievert的《[Interactive web-based data visualization with R, plotly, and shiny](#)》一书中所记录的那样。

7.2 Dynamic height and width

本章的其余部分不如交互式图形令人兴奋，但包含一些重要的技术。

首先，可以使绘图大小变化，因此宽度和高度会根据用户操作而变化。为此，请为`renderPlot()`的`width`和`height`参数提供零参数函数—这些函数必须在server中定义，而不是在UI中定义，因为它们可以更改。这些函数应该没有参数，并以像素为单位返回所需的大小。它们在渲染环境中进行调用，以便您可以使绘图的大小具有动态性。

以下应用程序说明了基本想法。它提供了两个直接控制绘图大小的滑块。几个示例屏幕截图如图7.6所示。请注意，当您调整绘图大小时，数据保持不变：您不会获得新的随机数。

```

ui <- fluidPage(
  sliderInput("height", "height", min = 100, max = 500, value = 250),
  sliderInput("width", "width", min = 100, max = 500, value = 250),
  plotOutput("plot", width = 250, height = 250)
)
server <- function(input, output, session) {
  output$plot <- renderPlot(
    width = function() input$width,
    height = function() input$height,
    res = 96,
    {
      plot(rnorm(20), rnorm(20))
    }
  )
}

```

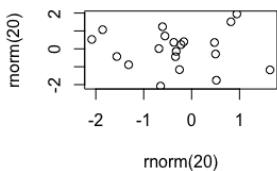
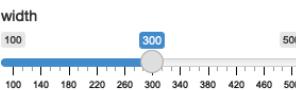
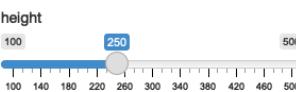
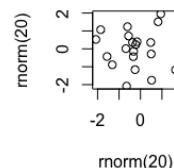
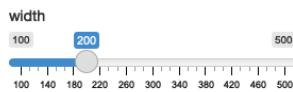
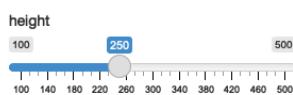


图7.6：您可以使绘图大小动态变化，使其响应用户操作。这张图显示了改变宽度的效果。在 <https://hadley.shinyapps.io/ms-resize> 上观看动态。

在实际应用程序中，您将在 `width` 和 `height` 函数中使用更复杂的表达式。例如，如果您在ggplot2中使用分面图，您可以使用它来增加图块的大小，以保持单个分面大小大致相同。

7.3 Images

如果您想显示现有图像（而不是绘图），您可以使用 `renderImage()`。例如，您可能有一个想要向用户显示的照片目录。以下应用程序通过显示可爱的小狗照片来说明 `renderImage()` 的基础知识。这些照片来自 <https://unsplash.com>，这是我最喜欢的开源照片来源。

```
puppies <- tibble::tribble(
  ~breed, ~id, ~author,
  "corgi", "eoqnr8ikwFE", "alvannee",
  "labrador", "KCdYn0xu2fU", "shaneguymon",
  "spaniel", "TzjMd7i5WQI", "_redo_"
)

ui <- fluidPage(
  selectInput("id", "Pick a breed", choices = setNames(puppies$id, puppies$breed)),
  htmlOutput("source"),
  imageOutput("photo")
)
server <- function(input, output, session) {
  output$photo <- renderImage({
    list(
      src = file.path("puppy-photos", paste0(input$id, ".jpg")),
      contentType = "image/jpeg",
      width = 500,
      height = 650
    )
  }, deleteFile = FALSE)

  output$source <- renderUI({
    info <- puppies[puppies$id == input$id, , drop = FALSE]
    HTML(glue::glue("<p>
      <a href='https://unsplash.com/photos/{info$id}'>original</a> by
      <a href='https://unsplash.com/@{info$author}'>{info$author}</a>
    </p>"))
  })
}
```

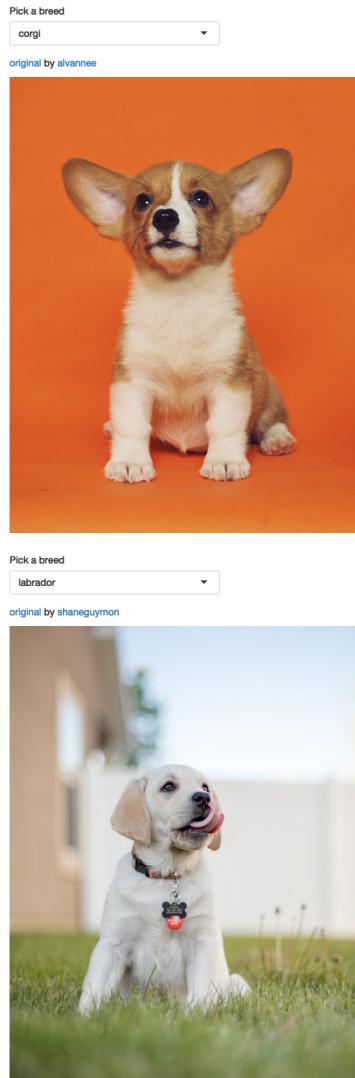


图7.7：一个使用 `renderImage()` 显示小狗可爱图片的应用程序。在<https://hadley.shinyapps.io/ms-puppies>上观看动态。

`renderImage()` 需要返回一个列表。唯一关键的参数是 `src`，这是图像文件的本地路径。您可以额外提供：

- `contentType`，它定义了图像的MIME类型。如果没有提供，Shiny将从文件扩展名中猜测，因此只有当您的图像没有扩展名时，您才需要提供这个。
- 图像的 `width` 和 `height`（如果已知）。
- 任何其他参数，如 `class` 或 `alt` 都将作为属性添加到HTML中的 `` 标签中。

您还必须提供 `deleteFile` 参数。不幸的是，`renderImage()` 最初被设计为处理临时文件，因此在渲染后会自动删除图像。这显然非常危险，所以在Shiny 1.5.0中的行为发生了变化。现在，shiny不再删除图像，而是迫使您明确选择您想要的行为。

您可以在<https://shiny.rstudio.com/articles/images.html>上了解有关 `renderImage()` 的更多信息，并查看您可能使用它的其他方式。

7.4 Summary

可视化是通信数据的强大方式，本章为您提供了一些高级技术来增强您的shiny应用程序的能力。接下来，您将学习向用户提供有关应用程序的反馈技巧，这对于提醒用户操作尤为重要。

Chapter 8. User feedback

通过让用户更深入地了解正在发生的事情来使您的应用程序更易于使用。比如，当输入没有意义时，以消息的形式展示到前端。当运行的程序，所需时间比较长时，以进度条的方式展示进度等。本章的目标是向您展示一些常用的用户反馈选项。

我们将从验证技术开始，当输入（或输入组合）处于无效状态时通知用户。我们也可以向用户发送一般消息和进度条，这些进度条提供了由许多小步骤组成的耗时操作的详细信息。最后，我们将讨论一些危险操作的提醒信息，这样您可以通过确认对话框或撤销操作的决定是否执行该危险操作。

在本章中，我们将使用Andy Merlino的[shinyFeedback](#)和John Coene的[waiter](#)。您还应该留意Joe Cheng的软件包[shinyvalidate](#)，该软件包目前正在开发中。

```
library(shiny)
```

8.1 Validation

您可以给用户的第一个也是最重要的反馈是，他们给了您错误的输入。这类似于在R中编写好的函数：用户友好的函数提供清晰的错误消息，描述预期的输入是什么以及您如何违反这些规则。思考用户如何滥用您的应用程序，允许您在UI中提供信息性消息，而不是允许错误渗透到R代码中并产生错误。

8.1.1 Validating input

向用户提供额外反馈的好方法是通过[shinyFeedback](#)软件包。通过两步就可以实现基本的反馈。首先，您将`useShinyFeedback()`添加到`ui`中。这为有吸引力的错误消息显示设置了所需的HTML和JavaScript：

```
ui <- fluidPage(  
  shinyFeedback::useShinyFeedback(),  
  numericInput("n", "n", value = 10),  
  textOutput("half")  
)
```

然后，在`server()`函数中，调用其中一个反馈函数：`feedback()`、`feedbackWarning()`、`feedbackDanger()`和`feedbackSuccess()`。它们都有三个关键参数：

- `inputId`，应该放置需要反馈的输入ID。
- `show`，从逻辑上决定是否显示反馈。
- `text`，要显示的文本。

它们还有`color`和`icon`参数，您可以使用它们来进一步自定义外观。有关更多详细信息，请参阅文档。

让我们看看这是如何在一个真实的例子中结合在一起的，假设我们只允许偶数。图8.1显示了结果。

```

server <- function(input, output, session) {
  half <- reactive({
    even <- input$n %% 2 == 0
    shinyFeedback:::feedbackWarning("n", !even, "Please select an even number")
    input$n / 2
  })

  output$half <- renderText(half())
}

```

The figure consists of two side-by-side screenshots of a Shiny application's user interface. Both screenshots show a numeric input field with a dropdown arrow on its right. In the left screenshot, the input field contains the value '10'. Below the input field, the text 'Please select an even number' is displayed in orange, indicating an invalid input. In the right screenshot, the input field contains the value '9'. Below the input field, the text 'Please select an even number' is also displayed in orange, along with a small yellow triangle warning icon.

图8.1：使用 `feedbackWarning()` 显示无效输入的警告。左侧的应用程序显示有效输入，右侧的应用程序显示带有警告消息的无效（奇）输入。请访问<https://hadley.shinyapps.io/ms-feedback>查看动态。

请注意，显示错误消息，但输出仍在更新。通常，您不希望这样，因为无效的输入可能会导致错误的结果。要阻止输入触发反应性更改，您需要一个新工具：`req()` 是“required”的缩写。看起来像这样：

```

server <- function(input, output, session) {
  half <- reactive({
    even <- input$n %% 2 == 0
    shinyFeedback:::feedbackWarning("n", !even, "Please select an even number")
    req(even)
    input$n / 2
  })

  output$half <- renderText(half())
}

```

当 `req()` 的输入不为真时，它会发送一个特殊信号来告诉Shiny，反应器没有它需要的所有输入，因此它应该被“暂停”。在我们重新与 `validate()` 协同使用它之前，我们将进行简短的题外话来讨论这个问题。

8.1.2 Cancelling execution with `req()`

您可能已经注意到，当您启动应用程序时，甚至在用户做任何事情之前，就会计算出完整的反应图。当您可以为输入选择有意义的默认值时，这很有效。但这并不总是可能的，有时你想等到用户真正做了一些事情。这往往伴随着三个控件出现：

- 在 `textInput()` 中，您使用了 `value = ""`，在用户键入内容之前，您不想做任何事情。
- 在 `selectInput()` 中，您提供了一个空选项 `" "`，在用户做出选择之前，您不想做任何事情。
- 在 `fileInput()` 中，在用户上传任何内容之前，其结果为空。我们将在第9.1节中回到这一点。

我们需要一些方法来“暂停”反应，这样在某种情况出现之前，什么都不会发生。这是 `req()` 的工作，它在渲染环境后续代码执行之前检查所需的值。

例如，考虑以下应用程序，该应用程序将用英语或毛利语生成问候语。如果您运行此应用程序，您将看到一个错误，如图8.2所示，因为`greetings`向量中没有对应于""默认选项的条目。

```
ui <- fluidPage(  
  selectInput("language", "Language", choices = c("", "English", "Maori")),  
  textInput("name", "Name"),  
  textOutput("greeting")  
)  
  
server <- function(input, output, session) {  
  greetings <- c(  
    English = "Hello",  
    Maori = "Kia ora"  
)  
  output$greeting <- renderText({  
    paste0(greetings[[input$language]], " ", input$name, "!")  
  })  
}
```

The screenshot shows a user interface for a Shiny application. At the top is a dropdown menu labeled "Language". Below it is a text input field labeled "Name". Underneath these fields, the text "Error: subscript out of bounds" is displayed in red, indicating a runtime error.

图8.2：该应用程序在加载时显示一个不提供信息的错误，因为尚未选择语言

我们可以使用`req()`来解决这个问题，如下所示。现在，在用户提供语言和名称的值之前，不会显示任何内容，如图8.3所示。

```
server <- function(input, output, session) {  
  greetings <- c(  
    English = "Hello",  
    Maori = "Kia ora"  
)  
  output$greeting <- renderText({  
    req(input$language, input$name)  
    paste0(greetings[[input$language]], " ", input$name, "!")  
  })  
}
```

Language

Name

Language

English

Name

Language

English

Name

Hadley

Hello Hadley!

图8.3：通过使用 `req()`，提供语言和名称后才会显示输出，见<https://hadley.shinyapps.io/ms-require-simple-2/>。

`req()` 通过发出特殊条件信号来工作。这种特殊条件导致所有下游反应和输出停止执行。从技术上讲，它使任何下游反应式表达式处于无效状态。我们将在第16章中回到这个术语。

`req()` 旨在使 `req(input$x)` 仅在用户提供值的情况下继续进行，无论输入控件的类型如何。如果需要，您也可以将 `req()` 与自己的逻辑语句一起使用。例如，`req(input$a > 0)`，将允许在 `a` 大于0时进行计算；这通常是您在执行验证时使用的形式，我们将在接下来看到。

8.1.3 `req()` and validation

让我们结合 `req()` 和 `shinyFeedback` 来解决一个更具挑战性的问题。我将回到我们在第1章中制作的简单应用程序，该应用程序允许您选择内置数据集并查看其内容。我将通过使用 `textInput()` 而不是 `selectInput()` 来使其更通用、更复杂。用户界面变化很小：

```
ui <- fluidPage(
  shinyFeedback::useShinyFeedback(),
  textInput("dataset", "Dataset name"),
  tableOutput("data")
)
```

但 `server` 函数需要变得更复杂一点。我们将以两种方式使用 `req()`：

- 我们只想在用户输入值的情况下继续计算，所以我们使用 `req(input$dataset)`
- 然后，我们检查提供的名称是否确实存在。如果没有，我们会显示一条错误消息，然后使用 `req()` 取消计算。请注意，使用 `cancelOutput = TRUE`：通常取消响应，将重置所有下游输出；使用 `cancelOutput = TRUE`，使它们显示最后一个符合条件的值。这对 `textInput()` 很重要，它可能会在您键入名称时触发更新。

结果如图8.4所示。

```
server <- function(input, output, session) {
  data <- reactive({
    req(input$dataset)

    exists <- exists(input$dataset, "package:datasets")
    shinyFeedback::feedbackDanger("dataset", !exists, "Unknown dataset")
    req(exists, cancelOutput = TRUE)

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

  output$data <- renderTable({
    head(data())
  })
}
```

The figure consists of three vertically stacked screenshots of a Shiny application. Each screenshot shows a simple interface with a text input field labeled 'Dataset name' and a table below it.

- Screenshot 1:** The 'Dataset name' input field is empty. Below it is an empty table with columns: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.
- Screenshot 2:** The 'Dataset name' input field contains the value 'iris'. Below it is a table with the same five columns. The data rows are: 5.10, 3.50, 1.40, 0.20, setosa; 4.90, 3.00, 1.40, 0.20, setosa; 4.70, 3.20, 1.30, 0.20, setosa; 4.60, 3.10, 1.50, 0.20, setosa; 5.00, 3.60, 1.40, 0.20, setosa; and 5.40, 3.90, 1.70, 0.40, setosa.
- Screenshot 3:** The 'Dataset name' input field contains the value 'iri'. A red error message 'Unknown dataset' is displayed above the table. Below it is an empty table with the same five columns.

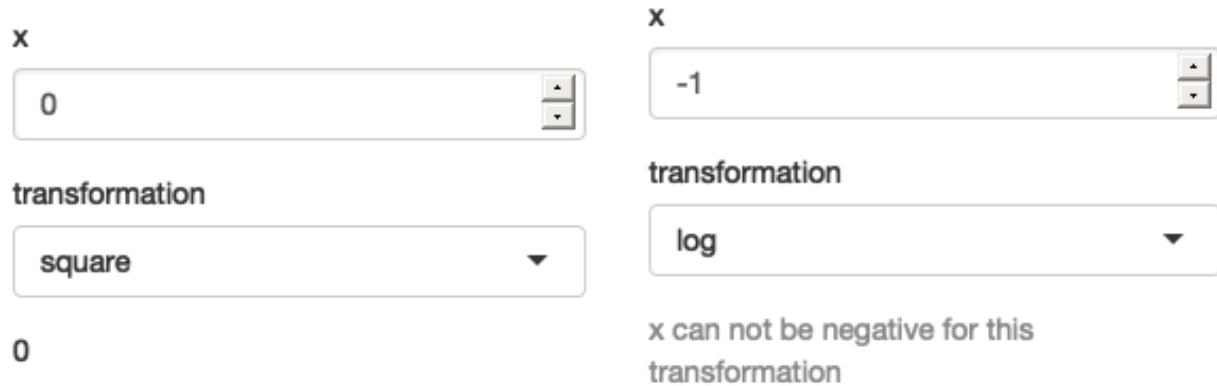
图8.4：加载时，表是空的，因为数据集名称是空的。数据在我们键入有效的数据集名称（iris）后显示。在 <https://hadley.shinyapps.io/ms-require-cancel> 观看动态。

8.1.4 Validate output

当问题与单个输入有关时，shinyFeedback很棒。但有时无效输入是所有输入组合的结果。在这种情况下，将错误放在输入旁边真的没有意义（你会把它放在哪个旁边？）。相反，把它放在输出中更有意义。

您可以使用内置在shiny中的工具进行此工作：`validate()`。当在响应程序或输出中调用`validate(message)`时，将停止执行其余代码，而是在任何下游输出中显示`message`。以下代码显示了一个不能对负数进行log或平方根运算的简单示例。您可以在图8.5中看到结果。

```
ui <- fluidPage(  
  numericInput("x", "x", value = 0),  
  selectInput("trans", "transformation",  
    choices = c("square", "log", "square-root"))  
,  
  textOutput("out")  
)  
  
server <- function(input, output, session) {  
  output$out <- renderText({  
    if (input$x < 0 && input$trans %in% c("log", "square-root")) {  
      validate("x can not be negative for this transformation")  
    }  
  
    switch(input$trans,  
      square = input$x ^ 2,  
      "square-root" = sqrt(input$x),  
      log = log(input$x)  
    )  
  })  
}
```



The figure shows a Shiny user interface with two panels. The left panel has an input field labeled 'x' containing '0' and a dropdown menu labeled 'transformation' with 'square' selected. The right panel has an input field labeled 'x' containing '-1' and a dropdown menu labeled 'transformation' with 'log' selected. Below the right panel, a message box displays the text 'x can not be negative for this transformation'.

图8.5：如果输入有效，输出显示转换。如果输入组合无效，则输出将替换为提示信息。

8.2 Notification

如果没有问题，而您只是想让用户知道发生了什么，那么您想要一个通知。在Shiny中，使用`showNotification()`创建通知，并堆叠在页面右下角。使用`showNotification()`有三种基本方法：

- 显示固定时间后自动消失的瞬态通知。
- 在进程开始时显示通知，并在进程结束时将其删除。
- 使用渐进式更新更新单个通知。

下面将讨论这三种技术。

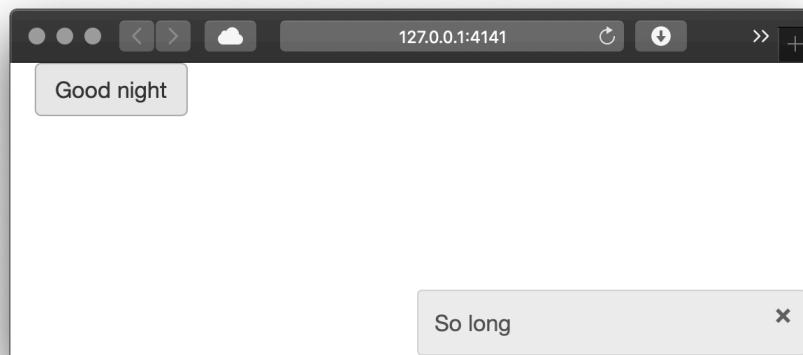
8.2.1 Transient notification

使用 `showNotification()` 的最简单方法是使用单个参数调用它：您想要显示的消息。很难用屏幕截图捕捉这种行为，所以如果您想看到它的实际效果，请访问<https://hadley.shinyapps.io/ms-notification-transient>。

```
ui <- fluidPage(  
  actionButton("goodnight", "Good night")  
)  
server <- function(input, output, session) {  
  observeEvent(input$goodnight, {  
    showNotification("So long")  
    Sys.sleep(1)  
    showNotification("Farewell")  
    Sys.sleep(1)  
    showNotification("Auf Wiedersehen")  
    Sys.sleep(1)  
    showNotification("Adieu")  
  })  
}
```

默认情况下，消息将在5秒后消失，您可以通过设置 `duration` 来覆盖，或者用户可以通过单击关闭按钮提前关闭它。如果您想让通知更突出，您可以将 `type` 参数设置为“消息（message）”、“警告（warning）”或“错误（error）”之一。见图8.6。

```
server <- function(input, output, session) {  
  observeEvent(input$goodnight, {  
    showNotification("So long")  
    Sys.sleep(1)  
    showNotification("Farewell", type = "message")  
    Sys.sleep(1)  
    showNotification("Auf Wiedersehen", type = "warning")  
    Sys.sleep(1)  
    showNotification("Adieu", type = "error")  
  })  
}
```



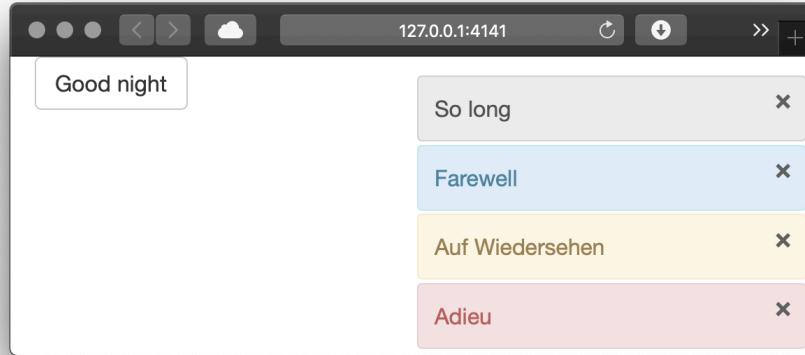


图8.6：单击“Good night”后通知的进展：第一个通知出现，三秒钟后显示所有通知，然后通知开始消失。在<https://hadley.shinyapps.io/ms-notify-persistent>观看动态。

8.2.2 Removing on completion

将通知的存在与长期运行的任务联系起来通常很有用。在这种情况下，您希望在任务开始时显示通知，并在任务完成时删除通知。要做到这一点，你需要：

- 设置 `duration = NULL` 和 `closeButton = FALSE`，以便通知在任务完成之前保持可见。
- 存储 `showNotification()` 返回的 `id`，然后传递此值给 `removeNotification()`。最可靠的方法是使用 `on.exit()`，它确保无论任务是否完成（成功或出现错误）都会删除通知。您可以在[Changing and restoring state](#)中了解有关 `on.exit()` 的更多信息。

以下示例将这些片段放在一起，以显示在阅读大型csv文件时，如何让用户保持最新状态：

```
server <- function(input, output, session) {
  data <- reactive({
    id <- showNotification("Reading data...", duration = NULL, closeButton = FALSE)
    on.exit(removeNotification(id), add = TRUE)

    read.csv(input$file$datapath)
  })
}
```

一般来说，这类通知将处于响应状态，因为这可以确保长期运行的计算，仅在需要时重新运行。

8.2.3 Progressive updates

正如您在第一个示例中看到的，对 `showNotification()` 多个调用通常会创建多个通知。相反，您可以通过从第一次调用中捕获 `id`，并在后续调用中使用它，来更新单个通知。如果您的长期运行任务有多个子组件，这很有用。您可以在<https://hadley.shinyapps.io/ms-notification-updates>中看到结果。

```
ui <- fluidPage(
  tableOutput("data")
)

server <- function(input, output, session) {
  notify <- function(msg, id = NULL) {
    showNotification(msg, id = id, duration = NULL, closeButton = FALSE)
  }

  data <- reactive({
    id <- notify("Reading data...")
    on.exit(removeNotification(id), add = TRUE)
    Sys.sleep(1)

    notify("Reticulating splines...", id = id)
    Sys.sleep(1)

    notify("Herding llamas...", id = id)
    Sys.sleep(1)

    notify("Orthogonalizing matrices...", id = id)
    Sys.sleep(1)

    mtcars
  })

  output$data <- renderTable(head(data()))
}
```

8.3 Progress bars

对于长期运行的任务，最好的反馈类型是进度条。除了告诉你处理的过程，它还能帮助你估计还需要多久：你应该深呼吸，去喝杯咖啡，还是明天回来？在本节中，我将展示两种显示进度条的技术，一种内置在Shiny中，一种来自John Coene开发的[waiter](#)包。

不幸的是，这两种技术都有相同的主要缺点：要使用进度条，您需要能够将大任务分成已知数量的小块，每个小块大约需要相同的时间。这通常很难，特别是因为底层代码通常用C语言编写，它无法向您传达进度更新。我们正在开发[progress package](#)中的工具，以便dplyr、readr和vroom等软件包有一天会生成进度条，您可以轻松地转发到Shiny。

8.3.1 Shiny

要使用Shiny创建进度条，您需要使用 `withProgress()` 和 `incProgress()`。想象一下，你有一些运行缓慢的代码，看起来像这样：

```
for (i in seq_len(step)) {  
  x <- function_that_takes_a_long_time(x)  
}
```

您从 `withProgress()` 封装代码块开始。这会在代码开始时显示进度条，并在完成后自动将其删除：

```
withProgress({  
  for (i in seq_len(step)) {  
    x <- function_that_takes_a_long_time(x)  
  }  
})
```

然后在每个步骤后调用 `incProgress()`：

```
withProgress({  
  for (i in seq_len(step)) {  
    x <- function_that_takes_a_long_time(x)  
    incProgress(1 / length(step))  
  }  
})
```

`incProgress()` 的第一个参数是增加进度条的进度。默认情况下，进度条从0开始，以1结束，因此将1除以步骤数将确保进度条在循环结束时完成（相当于进度条每一次延伸的长度）。

以下是在完整的Shiny应用程序中的外观，如图8.7所示。

```
ui <- fluidPage(  
  numericInput("steps", "How many steps?", 10),  
  actionButton("go", "go"),  
  textOutput("result")  
)  
  
server <- function(input, output, session) {  
  data <- eventReactive(input$go, {  
    withProgress(message = "Computing random number", {  
      for (i in seq_len(input$steps)) {  
        Sys.sleep(0.5)  
        incProgress(1 / input$steps)  
      }  
      runif(1)  
    })  
  })  
  
  output$result <- renderText(round(data(), 2))  
}
```

How many steps?

go



图8.7：进度条有助于指示计算需要运行多久。在<https://hadley.shinyapps.io/ms-progress>观看动态。

需要注意的几件事：

- 我使用可选 `message` 参数在进度条中添加了一些解释性文本。
- 我使用 `Sys.sleep()` 来模拟长时间运行的操作；在您的代码中，这将是一个缓慢的函数。
- 我允许用户通过将按钮与 `eventReactive()` 相结合来控制事件何时开始。对于任何需要进度条的任务来说，这是良好做法。

8.3.2 Waiter

内置的进度条非常适合基础设计，但如果您想要提供更多视觉选项的东西，您可以尝试[waiter](#)包。调整上述代码以与waiter一起工作。在UI中，我们添加了 `use_waitress()`：

```
ui <- fluidPage(  
  waiter::use_waitress(),  
  numericInput("steps", "How many steps?", 10),  
  actionButton("go", "go"),  
  textOutput("result")  
)
```

Waiter进度条的界面有点不同。waiter包使用R6对象将所有与进度相关的函数捆绑到单个对象中。如果您以前从未使用过R6对象，请不要太担心细节；您只需复制并粘贴此模板即可。基本生命周期如下所示：

```
# Create a new progress bar  
waitress <- waiter::Waitress$new(max = input$steps)  
# Automatically close it when done  
on.exit(waitress$close())  
  
for (i in seq_len(input$steps)) {  
  Sys.sleep(0.5)  
  # increment one step  
  waitress$inc(1)  
}
```

我们可以在Shiny应用程序中使用它，如下所示：

```

server <- function(input, output, session) {
  data <- eventReactive(input$go, {
    waitress <- waiter::Waitress$new(max = input$steps)
    on.exit(waitress$close())

    for (i in seq_len(input$steps)) {
      Sys.sleep(0.5)
      waitress$inc(1)
    }

    runif(1)
  })

  output$result <- renderText(round(data(), 2))
}

```

默认显示是页面顶部的进度条（您可以看到<https://hadley.shinyapps.io/ms-waiter>），但有很多方法可以自定义输出：

- 您可以覆盖默认 `theme` 以使用以下之一：
 - `overlay`：隐藏整个页面的不透明进度条
 - `overlay-opacity`：一个半透明的进度条，覆盖整个页面
 - `overlay-percent`：一个不透明的进度条，也显示数字百分比。
- 您可以通过设置 `selector` 参数将其叠加在现有输入或输出上，而不是显示整个页面的进度条，例如：

```
waitress <- Waitress$new(selector = "#steps", theme = "overlay")
```

8.3.3 Spinners

有时您不知道操作需要多长时间，您只想显示一个动画旋转器（Spinners），让用户确信正在发生一些事情。您也可以使用waiter包执行此任务；只需从使用 `Waitress` 切换到使用 `Waiter`：

```

ui <- fluidPage(
  waiter::use_waiter(),
  actionButton("go", "go"),
  textOutput("result")
)

server <- function(input, output, session) {
  data <- eventReactive(input$go, {
    waiter <- waiter::Waiter$new()
    waiter$show()
    on.exit(waiter$hide())

    Sys.sleep(sample(5, 1))
    runif(1)
  })
  output$result <- renderText(round(data(), 2))
}

```

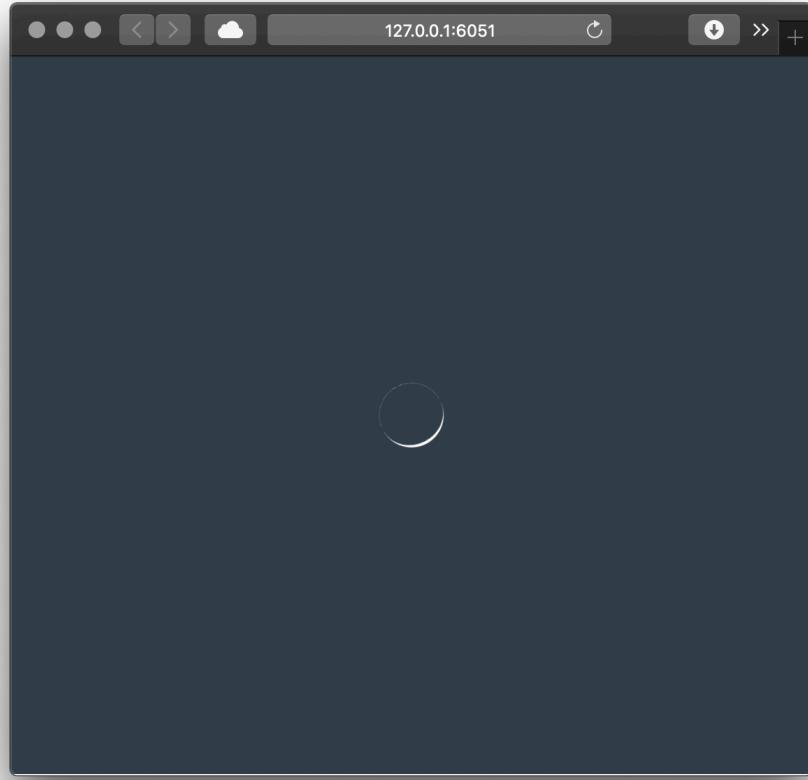


图8.8：当有事情发生时，“waiter”会显示整个应用程序旋转器。在<https://hadley.shinyapps.io/ms-spinner-1>观看动态。

像 `Waitress` 一样，您也可以使用 `waiter` 进行特定输出。当输出更新时，这些 `waiter` 可以自动删除旋转器，因此代码更简单：

```
ui <- fluidPage(  
  waiter::use_waiter(),  
  actionButton("go", "go"),  
  plotOutput("plot"),  
)  
  
server <- function(input, output, session) {  
  data <- eventReactive(input$go, {  
    waiter::Waiter$new(id = "plot")$show()  
  
    Sys.sleep(3)  
    data.frame(x = runif(50), y = runif(50))  
  })  
  
  output$plot <- renderPlot(plot(data()), res = 96)  
}
```

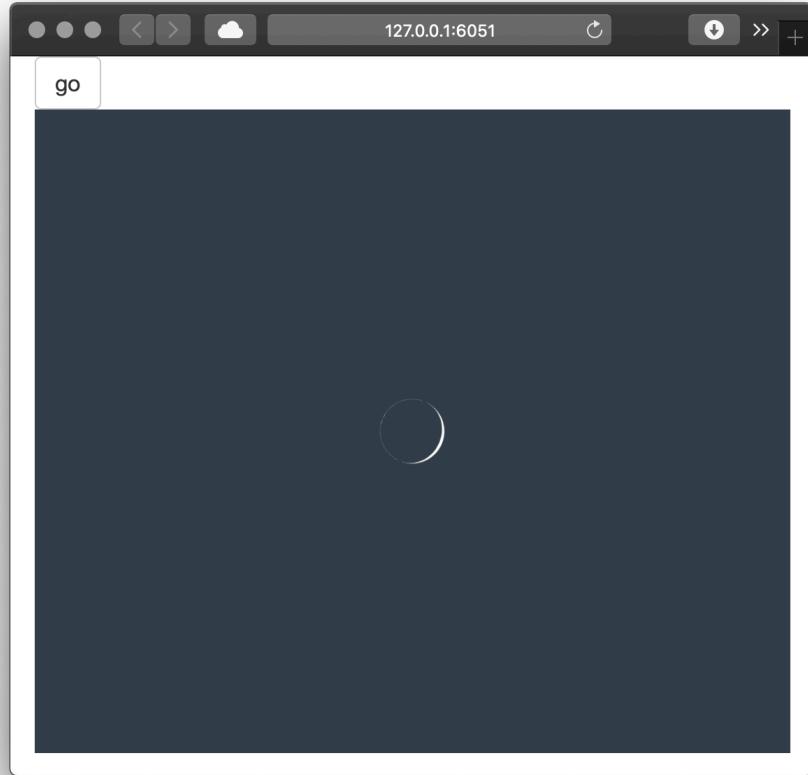


图8.9：您可以显示单个输出的旋转器。在<https://hadley.shinyapps.io/ms-spinner-2>观看动态。

waiter package提供了各种各样的旋转器可供选择；请参阅您的选项 `?waiter::spinners`，然后选择一个（例如）`Waiter$new(html = spin_ripple())`

一个更简单的替代方案是使用Dean Attali的[shinycssloaders](#)软件包。它使用JavaScript来监听Shiny事件，因此它甚至不需要服务器端的任何代码。相反，您只需使用 `shinycssloaders::withSpinner()` 来封装输出代码。旋转器会在执行时出现，执行结束后消失，展示运行的结果。

```
library(shinycssloaders)

ui <- fluidPage(
  actionButton("go", "go"),
  withSpinner(plotOutput("plot")),
)
server <- function(input, output, session) {
  data <- eventReactive(input$go, {
    Sys.sleep(3)
    data.frame(x = runif(50), y = runif(50))
  })

  output$plot <- renderPlot(plot(data()), res = 96)
}
```

8.4 Confirming and undoing

有时，一个动作是潜在的危险的，你要么想确保用户真的想这样做，要么你想让他们有能力在为时已晚之前退出。本节中的三种技术可以解决这个问题，并为您提供了一些如何在应用程序中实现它们的技术。

8.4.1 Explicit confirmation

防止用户意外执行危险操作的最简单方法是要求明确确认。最简单的方法是使用一个对话框，该对话框强制用户从一小组操作中选择一个。在Shiny中，您创建一个带有`modalDialog()`的对话框。这被称为“modal”对话框，因为它创建了一个新的交互“mode”；在处理完对话框之前，您无法与主应用程序进行交互。

想象一下，您有一个shiny应用程序，可以从目录（或数据库中的行等）中删除一些文件。这很难撤销，所以你想确保用户真的要执行该动作。您可以创建一个对话框，如图8.10所示，该对话框需要明确确认，如下所示：

```
modal_confirm <- modalDialog(  
  "Are you sure you want to continue?",  
  title = "Deleting files",  
  footer = tagList(  
    actionButton("cancel", "Cancel"),  
    actionButton("ok", "Delete", class = "btn btn-danger")  
  )  
)
```

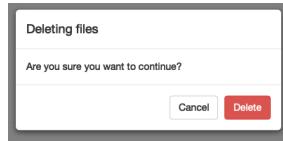


图8.10：一个对话框，确认您是否要删除某些文件。

在创建对话框时，需要考虑一些小但重要的细节：

- 你应该称这些按钮为什么？最好是描述性的，所以避免是/否或继续/取消，转而重述关键动词。
- 你应该如何定义按钮？您是先取消（像Mac一样），还是先继续（像Windows一样）？你最好的选择是符合你认为大多数人会使用的平台。
- 你能让危险的选择更明显吗？在这里，我使用了`class = "btn btn-danger"`来突出按钮的样式。

Jakob Nielsen在<http://www.useit.com/alertbox/ok-cancel.html>上有更多好建议。

让我们在一个真正的（如果非常简单）应用程序中使用这个对话框。我们的用户界面公开了一个“删除所有文件（delete all the files）”的按钮：

```
ui <- fluidPage(  
  actionButton("delete", "Delete all files?")  
)
```

`server()`中有两个新想法：

- 我们使用`showModal()`和`removeModal()`来显示和隐藏对话框。
- 我们观察UI从`modal_confirm`生成的事件。这些对象不是在`ui`中静态创建的，而是由`showModal()`动态添加到`server()`中。你会在第10章中更详细地看到这个想法。

```

server <- function(input, output, session) {
  observeEvent(input$delete, {
    showModal(modal_confirm)
  })

  observeEvent(input$ok, {
    showNotification("Files deleted")
    removeModal()
  })
  observeEvent(input$cancel, {
    removeModal()
  })
}

```

8.4.2 Undoing an action

显式确认对于很少执行的破坏性操作最有用。如果您想减少频繁操作造成的错误，您应该避免它。例如，这种技术对推特不起作用—如果有一个对话框说“你确定你想发这个推特吗？”你很快就会学会自动点击是，当你在推特上发表后，注意到10s的错别字时，你仍然会感到同样的遗憾。

在这种情况下，更好的方法是等待几秒钟后再实际执行操作，让用户有机会注意到任何问题并撤销它们。这并不是真正的撤销（因为你实际上什么都没做），但这是一个用户会理解的令人回味的词。

我用一个网站来说明这项技术，我个人希望有一个撤销按钮：Twitter。Twitter UI的本质非常简单：有一个文本区域来编写您的推文，还有一个按钮可以发送它：

```

ui <- fluidPage(
  textAreaInput("message",
    label = NULL,
    placeholder = "What's happening?",
    rows = 3
  ),
  actionButton("tweet", "Tweet")
)

```

Server函数相当复杂，需要一些我们没有讨论过的技术。不要太担心理解代码，专注于基本思想：我们在`observeEvent()`中使用一些特殊的参数，在几秒钟后运行一些代码。最大的新想法是，我们捕获`observeEvent()`的结果并将其保存到一个变量中；这允许我们销毁刚刚填写的内容，因此真正发送推特的代码永远不会运行。可以在<https://hadley.shinyapps.io/ms-undo>上查看具体的实现。

```

runLater <- function(action, seconds = 3) {
  observeEvent(
    invalidateLater(seconds * 1000), action,
    ignoreInit = TRUE,
    once = TRUE,
    ignoreNULL = FALSE,
    autoDestroy = FALSE
  )
}

```

```

server <- function(input, output, session) {
  waiting <- NULL
  last_message <- NULL

  observeEvent(input$tweet, {
    notification <- glue::glue("Tweeted '{input$message}'")
    last_message <- input$message
    updateTextAreaInput(session, "message", value = "")

    showNotification(
      notification,
      action = actionButton("undo", "Undo?"),
      duration = NULL,
      closeButton = FALSE,
      id = "tweeted",
      type = "warning"
    )

    waiting <- runLater({
      cat("Actually sending tweet...\n")
      removeNotification("tweeted")
    })
  })

  observeEvent(input$undo, {
    waiting$destroy()
    showNotification("Tweet retracted", id = "tweeted")
    updateTextAreaInput(session, "message", value = last_message)
  })
}

```

8.4.3 Trash

对于几天后您可能后悔的行为，更复杂的模式是在计算机上实现垃圾或回收箱等工具。当您删除文件时，它不会被永久删除，而是被移动到特定的存储位置，这需要单独的操作才能清空。这就像“撤销”选项；你有很多时间后悔你的行为。这也有点像确认；你必须做两个单独的操作才能使删除永久化。

这种技术的主要缺点是，它实施起来要复杂得多（您必须有一个单独的“存储单元”，存储撤销操作所需的信息），并需要用户定期干预以避免积累。出于这个原因，我认为它超出了所有范围，除了最复杂的Shiny应用程序，所以我不打算在这里展示实现。

8.5 Summary

本章为您提供了许多工具，以帮助向用户传达您的应用程序发生了什么。从某种意义上说，这些技术大多是可选的。但是，虽然您的应用程序将在没有它们的情况下工作，但它们深思熟虑的应用程序可能会对用户体验质量产生巨大影响。当您是应用程序的唯一用户时，您通常可以省略反馈，但使用它的人越多，深思熟虑的通知就越有回报。

在下一章中，您将学习如何向用户传输文件和从用户传输文件。

Chapter 9. Uploads and downloads

上传文件和下载文件是应用程序的常见功能。您可以使用它上传数据进行分析，或将结果下载为数据集或报告。本章展示了将文件传入和传出应用程序所需的UI和server组件。

```
library(shiny)
```

9.1 Upload

我们将首先讨论文件上传，向您展示基本的UI和server组件，然后展示它们如何在一个简单的应用程序中结合在一起。

9.1.1 UI

支持文件上传所需的UI很简单：只需将 `fileInput()` 添加到您的UI中。

```
ui <- fluidPage(  
  fileInput("upload", "Upload a file")  
)
```

像大多数其他UI组件一样，只有两个必需的参数：`id` 和 `label`。`width`、`buttonLabel` 和 `placeholder` 参数允许您以其他方式调整外观。我不会在这里讨论它们，但你可以阅读更多关于它们的信息 [?fileInput](#)。

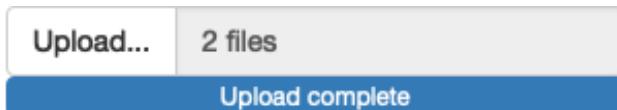
9.1.2 Server

在server上处理 `fileInput()` 比其他输入要复杂一些。大多数输入返回简单的向量，但 `fileInput()` 返回一个有四列的数据框：

- `name`：用户计算机上的原始文件名。
- `size`：文件大小，以字节为单位。默认情况下，用户只能上传最多5 MB的文件。您可以在启动Shiny之前通过设置 `shiny.maxRequestSize` 选项来增加此限制。例如，允许多达10 MB的运行
`options(shiny.maxRequestSize = 10 * 1024^2)`。
- `type`：文件的“MIME type”。这是文件类型，通常来自扩展名，在Shiny应用程序中很少需要。
- `datapath`：服务器上上传数据的路径。将此路径视为短暂的：如果用户上传更多文件，此文件可能会被删除。数据总是保存到一个临时目录，并给出一个临时名称。

我认为理解这个数据结构的最简单方法是制作一个简单的应用程序。运行以下代码并上传一些文件，以了解Shiny提供的数据。在我上传了几张小狗照片（来自图[9.1的第7.3节](#)）后，您可以看到结果。

```
ui <- fluidPage(  
  fileInput("upload", NULL, buttonLabel = "Upload...", multiple = TRUE),  
  tableOutput("files")  
)  
server <- function(input, output, session) {  
  output$files <- renderTable(input$upload)  
}
```



name	size	type	datapath
eoqnr8ikwFE.jpg	183186	image/jpeg	/tmp/RtmpdmeUj/022d21f6d4342af5aa0cf39/0.jpg
KCdYn0xu2fU.jpg	116343	image/jpeg	/tmp/RtmpdmeUj/022d21f6d4342af5aa0cf39/1.jpg

图9.1：这个简单的应用程序可以让您确切地看到Shiny为上传的文件计算了一些文件信息，并将文件保存在临时目录中。在<https://hadley.shinyapps.io/ms-upload>观看动态。

请注意，我使用 `label` 和 `buttonLabel` 参数来温和地自定义外观，并使用 `multiple = TRUE` 来允许用户上传多个文件。

9.1.3 Uploading data

如果用户正在上传数据集，您需要注意两个细节：

- `input$upload` 在页面加载时初始化为 `NULL`，因此您需要 `req(input$upload)` 来确保您的代码等到第一个文件上传。
- `accept` 参数允许您限制可能的输入。最简单的方法是提供文件扩展名的字符矢量，如 `accept = ".csv"`。但 `accept` 参数只是对浏览器的建议，并不总是强制执行，因此也验证它，是良好做法（例如第8.1节）。在R中获取文件扩展名的最简单方法是 `tools::file_ext()`。请注意，它会从扩展名中删除前导 `.`。

将所有这些想法放在一起，为我们提供了以下应用程序，您可以在其中上传一个 `.csv` 或 `.tsv` 文件，并查看前 `n` 行。在<https://hadley.shinyapps.io/ms-upload-validate>中查看它。

```
ui <- fluidPage(  
  fileInput("upload", NULL, accept = c(".csv", ".tsv")),  
  numericInput("n", "Rows", value = 5, min = 1, step = 1),  
  tableOutput("head"))  
  
server <- function(input, output, session) {  
  data <- reactive({  
    req(input$upload)  
  
    ext <- tools::file_ext(input$upload$name)  
    switch(ext,  
      csv = vroom::vroom(input$upload$datapath, delim = ","),  
      tsv = vroom::vroom(input$upload$datapath, delim = "\t"),  
      validate("Invalid file; Please upload a .csv or .tsv file")  
    )  
  })  
  
  output$head <- renderTable({  
    head(data(), input$n)  
  })  
}
```

请注意，由于 `multiple = FALSE` (默认值)，`input$file` 将是单个数据框，`input$file$name` 和 `input$file$datapath` 将是长度为1的字符向量。

9.2 Download

接下来，我们将实现文件下载，向您展示基本的UI和server组件，然后演示如何使用它们来允许用户下载数据或报告。

9.2.1 Basics

同样，用户界面很简单：使用 `downloadButton(id)` 或 `downloadLink(id)` 为用户提供单击下载文件的按钮。结果如图9.2所示。

```
ui <- fluidPage(  
  downloadButton("download1"),  
  downloadLink("download2")  
)
```



图9.2：下载按钮和下载链接

您可以使用与 `actionButtons()` 相同的 `class` 和 `icon` 参数自定义它们的外观，如第[2.2.7节](#)所述。

与其他输出不同，`downloadButton()` 不与渲染函数配对，需要与 `downloadHandler()` 配对：

```
output$download <- downloadHandler(  
  filename = function() {  
    paste0(input$dataset, ".csv")  
  },  
  content = function(file) {  
    write.csv(data(), file)  
  }  
)
```

`downloadHandler()` 有两个参数，指向两个函数：

- `filename` 应该是一个没有参数，但返回文件名（作为字符串）的函数。此函数的工作是创建一个在下载对话框中显示给用户的文件名称。
- `content` 应该是一个只有一个参数的函数，`file`，这是保存文件的路径。此函数的工作是将文件保存在 Shiny知道的地方，以便将其发送给用户。

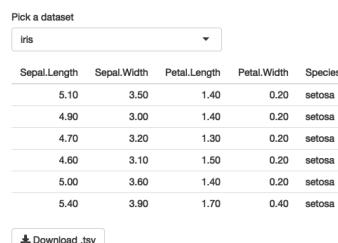
这是一个不寻常的界面，但它允许Shiny控制文件的保存位置（因此可以将其放置在安全位置），同时您仍然可以控制该文件的内容。

接下来，我们将把这些部分放在一起，展示如何使用户下载数据文件或报告。

9.2.2 Downloading data

下面的应用程序展示了数据下载的基本原理，允许您将数据包中的任何数据集作为一个文件下载，如图9.3所示。我建议使用`.tsv`（制表符分隔值）而不是`.csv`（逗号分隔值），因为许多欧洲国家使用逗号分隔数字的整数和小数部分（例如，1.23与1.23）。这意味着它们不能使用逗号分隔字段，而是在所谓的“c”sv文件中使用分号！您可以通过使用制表符分隔的文件来避免这种复杂性，这些文件在任何地方都以相同的方式工作。

```
ui <- fluidPage(  
  selectInput("dataset", "Pick a dataset", ls("package:datasets")),  
  tableOutput("preview"),  
  downloadButton("download", "Download .tsv")  
)  
  
server <- function(input, output, session) {  
  data <- reactive({  
    out <- get(input$dataset, "package:datasets")  
    if (!is.data.frame(out)) {  
      validate(paste0("'", input$dataset, "' is not a data frame"))  
    }  
    out  
  })  
  
  output$preview <- renderTable({  
    head(data())  
  })  
  
  output$download <- downloadHandler(  
    filename = function() {  
      paste0(input$dataset, ".tsv")  
    },  
    content = function(file) {  
      vroom::vroom_write(data(), file)  
    }  
  )  
}
```



Pick a dataset				
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

图9.3：一个更丰富的应用程序，允许您选择内置数据集并在下载前预览。在<https://hadley.shinyapps.io/ms-download-data>观看动态。

请注意，使用`validate()`仅允许用户下载数据集中的数据。更好的方法是预先过滤列表，但这可以让您看到另一个`validate()`的应用程序。

9.2.3 Downloading reports

除了下载数据，您可能还希望使用应用程序的用户下载一份报告，总结Shiny应用程序中交互式探索的结果。这是相当多的工作，因为您还需要以不同的格式显示相同的信息，但它对高风险应用程序非常有用。

生成此类报告的一个强大方法是使用[参数化的RMarkdown文档](#)。参数化的RMarkdown文件在YAML元数据中有一个`params`字段：

```
title: My Document
output: html_document
params:
  year: 2018
  region: Europe
  printcode: TRUE
  data: file.csv
```

在文档中，您可以使用`params$year`、`params$region`等引用这些值。YAML元数据中的值是默认值；您通常会通过在调用`rmarkdown::render()`中提供`params`来覆盖它们。这使得从同一报告生成许多不同的报告变得容易。

这是一个简单的示例，改编自<https://shiny.rstudio.com/articles/generating-reports.html>，它更详细地描述了这种技术。关键想法是从`downloadHandler()`的`content`参数中调用`rmarkdown::render()`。如果您想生成其他输出格式，只需更改输出格式即可，并确保更新扩展名（例如`.pdf`）。在<https://hadley.shinyapps.io/ms-download-rmd>上查看它。

```
ui <- fluidPage(
  sliderInput("n", "Number of points", 1, 100, 50),
  downloadButton("report", "Generate report")
)

server <- function(input, output, session) {
  output$report <- downloadHandler(
    filename = "report.html",
    content = function(file) {
      params <- list(n = input$n)

      id <- showNotification(
        "Rendering report...",
        duration = NULL,
        closeButton = FALSE
      )
      on.exit(removeNotification(id), add = TRUE)

      rmarkdown::render("report.Rmd",
        output_file = file,
        params = params,
        envir = new.env(parent = globalenv())
      )
    }
  )
}
```

通常至少需要几秒钟才能呈现 `.Rmd`，所以这是一个使用第8.2节通知的好地方。

还有其他几个值得了解的技巧：

- RMarkdown在当前工作目录中工作，这在许多部署场景中会失败（例如在shinyapps.io上）。您可以通过在应用程序启动时（即server函数之外），将报告复制到临时目录来解决这个问题：

```
report_path <- tempfile(fileext = ".Rmd")
file.copy("report.Rmd", report_path, overwrite = TRUE)
```

然后在调用 `rmarkdown::render()` 中将 `"report.Rmd"` 修改为 `report_path`

```
rmarkdown::render(report_path,
  output_file = file,
  params = params,
  envir = new.env(parent = globalenv())
)
```

- 默认情况下，RMarkdown将在当前进程中呈现报告，这意味着它将从Shiny应用程序中继承许多设置（如加载的软件包、选项等）。为了提高鲁棒性，我建议使用 `callr` 包在单独的R会话中运行 `render()`：

```
render_report <- function(input, output, params) {
  rmarkdown::render(input,
    output_file = output,
    params = params,
    envir = new.env(parent = globalenv())
  )
}

server <- function(input, output) {
  output$report <- downloadHandler(
    filename = "report.html",
    content = function(file) {
      params <- list(n = input$slider)
      callr::r(
        render_report,
        list(input = report_path, output = file, params = params)
      )
    }
  )
}
```

您可以在Mastering Shiny GitHub repo中找到 [rmarkdown-report/](#) 中所有这些部件的组合。

[shinymeta](#) 软件包解决了一个相关问题：有时您需要能够将Shiny应用程序的当前状态转换为可重现的报告，该报告将来可以重新运行。在Joe Cheng的useR中了解更多信息！2019年主题演讲“[Shiny holy grail](#)：具有可复制性的互动性”。

9.3 Case study

最后，我们将完成一个小型案例研究，上传一个文件（带有用户提供的分隔符），预览它，使用Sam Firke的[janitor packages](#)执行一些可选的转换，然后让用户将其下载为`.tsv`。

为了更容易理解如何使用该应用程序，我使用`sidebarLayout()`将应用程序分为三个主要步骤：

1. 上传和解析文件：

```
ui_upload <- sidebarLayout(  
  sidebarPanel(  
    fileInput("file", "Data", buttonLabel = "Upload..."),  
    textInput("delim", "Delimiter (leave blank to guess)", ""),  
    numericInput("skip", "Rows to skip", 0, min = 0),  
    numericInput("rows", "Rows to preview", 10, min = 1)  
,  
  mainPanel(  
    h3("Raw data"),  
    tableOutput("preview1")  
)  
)
```

2. 清理文件。

```
ui_clean <- sidebarLayout(  
  sidebarPanel(  
    checkboxInput("snake", "Rename columns to snake case?"),  
    checkboxInput("constant", "Remove constant columns?"),  
    checkboxInput("empty", "Remove empty cols?")  
,  
  mainPanel(  
    h3("Cleaner data"),  
    tableOutput("preview2")  
)  
)
```

3. 下载文件。

```
ui_download <- fluidRow(  
  column(width = 12, downloadButton("download", class = "btn-block"))  
)
```

组装成一个`fluidPage()`

```
ui <- fluidPage(  
  ui_upload,  
  ui_clean,  
  ui_download  
)
```

同样的组织架构，使理解应用程序变得更容易：

```
server <- function(input, output, session) {
  # Upload -----
  raw <- reactive({
    req(input$file)
    delim <- if (input$delim == "") NULL else input$delim
    vroom::vroom(input$file$datapath, delim = delim, skip = input$skip)
  })
  output$preview1 <- renderTable(head(raw(), input$rows))

  # Clean -----
  tidied <- reactive({
    out <- raw()
    if (input$snake) {
      names(out) <- janitor::make_clean_names(names(out))
    }
    if (input$empty) {
      out <- janitor::remove_empty(out, "cols")
    }
    if (input$constant) {
      out <- janitor::remove_constant(out)
    }

    out
  })
  output$preview2 <- renderTable(head(tidied(), input$rows))

  # Download -----
  output$download <- downloadHandler(
    filename = function() {
      paste0(tools::file_path_sans_ext(input$file$name), ".tsv")
    },
    content = function(file) {
      vroom::vroom_write(tidied(), file)
    }
  )
}
```

The screenshot shows a Shiny application interface with two main sections:

- Raw data** (Left Panel):
 - A "Data" section with a "Upload..." button and a "No file selected" message.
 - Input fields for "Delimiter (leave blank to guess)" and "Rows to skip" (set to 0).
 - Input field for "Rows to preview" (set to 10).
- Cleaner data** (Right Panel):
 - Checkboxes for "Rename columns to snake case?", "Remove constant columns?", and "Remove empty cols?".
 - A "Download" button at the bottom.

图9.4：一个允许用户上传文件，执行一些简单清理，然后下载结果的应用程序。在 <https://hadley.shinyapps.io/ms-case-study> 观看动态效果。

9.4 Exercises

1. 使用Thomas Lin Pedersen的[ambient](#)包来产生[worley noise](#)并下载它的PNG。
2. 创建一个应用程序，允许您上传csv文件，选择一个变量，然后对该变量执行 `t.test()`。用户上传csv文件后，您需要使用 `updateSelectInput()` 来填写可用变量。详情请参阅第[10.1](#)节。
3. 创建一个应用程序，允许用户上传csv文件，选择一个变量，绘制直方图，然后下载直方图。对于额外的挑战，允许用户从 `.png`、`.pdf` 和 `.svg` 输出格式中进行选择。
4. 编写一个应用程序，允许用户使用Ryan Timpe的[brickr](#)包从任何 `.png` 文件创建Lego马赛克。完成基础组件后，添加控件，允许用户选择马赛克的大小（砖块），并选择使用“universal”还是“generic”调色板。
5. 第[9.3](#)节中的最终应用程序包含这个大型反应：

```
tidied <- reactive({  
  out <- raw()  
  if (input$snake) {  
    names(out) <- janitor::make_clean_names(names(out))  
  }  
  if (input$empty) {  
    out <- janitor::remove_empty(out, "cols")  
  }  
  if (input$constant) {  
    out <- janitor::remove_constant(out)  
  }  
  
  out  
})
```

将其分解成多个部分，以便（例如）在 `input$empty` 更改时不会重新运行 `janitor::make_clean_names()`。

9.5 Summary

在本章中，您学习了如何使用 `fileInput()` 和 `downloadButton()` 向用户传输文件。大多数挑战要么是处理上传的文件，要么生成要下载的文件，所以我向您展示了如何处理几个常见情况。如果我没有在这里涵盖你的具体挑战，你需要运用你自己独特的创造力来解决问题😊。

下一章将帮助您在处理用户提供的数据时应对一个常见的挑战：您需要动态调整用户界面，以更好地适应数据。我将从一些易于理解并可以应用于许多情况的简单技术开始，逐渐完成代码生成的动态用户界面。

Chapter 10. Dynamic UI

到目前为止，我们已经看到了用户界面和服务器功能之间的完美分离：用户界面在应用程序启动时是静态定义的，因此它无法响应应用程序中发生的任何事情。在本章中，您将学习如何创建动态用户界面，使用 `server` 函数中运行的代码更改UI。

创建动态用户界面有三种关键技术：

- 使用 `update` 系列函数来修改输入控件的参数。
- 使用 `tabsetPanel()` 有条件地显示和隐藏用户界面的部分内容。
- 使用 `uiOutput()` 和 `renderUI()` 用代码生成用户界面的选定部分。

这三种工具通过修改输入和输出，为您提供相当大的能力来响应用户。我将展示一些更有用的方法，你可以应用它们，但最终你只受制于你的创造力。与此同时，这些工具可能会使您的应用程序更难推理，因此请谨慎部署它们，并始终努力使用最简单的技术来解决问题。

```
library(shiny)
library(dplyr, warn.conflicts = FALSE)
```

10.1 Updating inputs

我们将从一个简单的技术开始，该技术允许您在创建后修改输入：更新函数家族（the update family of functions）。每个输入控件，例如 `textInput()` 都与更新函数（update function）配对，例如 `updateTextInput()`，该函数允许您在创建控件后对其进行修改。

以以下代码中的示例为例，结果如图10.1所示。该应用程序有两个输入，可以控制另一个输入的范围（`min` 和 `max`），即滑块。关键想法是每当 `min` 或 `max` 输入发生变化时，使用 `observeEvent()` 触发 `updateSliderInput()`。

```
ui <- fluidPage(
  numericInput("min", "Minimum", 0),
  numericInput("max", "Maximum", 3),
  sliderInput("n", "n", min = 0, max = 3, value = 1)
)
server <- function(input, output, session) {
  observeEvent(input$min, {
    updateSliderInput(inputId = "n", min = input$min)
  })
  observeEvent(input$max, {
    updateSliderInput(inputId = "n", max = input$max)
  })
}
```



图10.1：加载应用程序（左），在增加最大值（中间）后，然后减少最小值（右）。在 <https://hadley.shinyapps.io/ms-update-basics> 观看动态效果。

更新函数看起来与其他Shiny函数有点不同：它们都将输入的名称（作为字符串）作为 `inputId` 参数。其余参数对应于创建后可以修改的输入构造函数的参数。

为了帮助您掌握更新函数，我将展示几个更简单的示例，然后我们将使用分层选择框深入研究一个更复杂的案例研究，最后讨论循环引用的问题。

10.1.1 Simple uses

更新函数最简单的用途是为用户提供一些便利。例如，也许您想让将参数重置为初始值变得容易。以下片段显示了如何将 `actionButton()`，`observeEvent()` 和 `updateSliderInput()` 与图10.2所示的输出结合起来。

```
ui <- fluidPage(  
  sliderInput("x1", "x1", 0, min = -10, max = 10),  
  sliderInput("x2", "x2", 0, min = -10, max = 10),  
  sliderInput("x3", "x3", 0, min = -10, max = 10),  
  actionButton("reset", "Reset")  
)  
  
server <- function(input, output, session) {  
  observeEvent(input$reset, {  
    updateSliderInput(inputId = "x1", value = 0)  
    updateSliderInput(inputId = "x2", value = 0)  
    updateSliderInput(inputId = "x3", value = 0)  
  })  
}
```

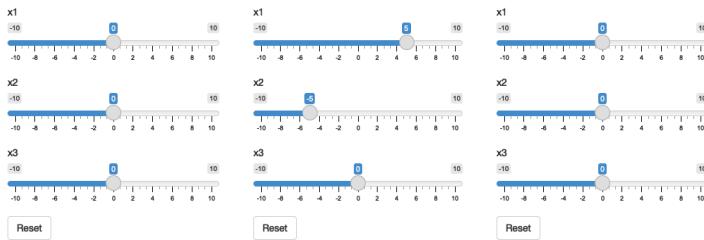


图10.2：加载应用程序（左），拖动一些滑块（中间），然后单击重置（右）。在 <https://hadley.shinyapps.io/ms-update-reset> 观看动态效果。

一个类似的应用程序是调整操作按钮的文本，以便您确切地知道它将做什么。图10.3显示了以下代码的结果。

```
ui <- fluidPage(  
  numericInput("n", "Simulations", 10),  
  actionButton("simulate", "Simulate")  
)  
  
server <- function(input, output, session) {  
  observeEvent(input$n, {  
    label <- paste0("Simulate ", input$n, " times")  
    updateActionButton(inputId = "simulate", label = label)  
  })  
}
```



图10.3：加载的应用程序（左），在将模拟设置为1（中）后，然后将模拟设置为100（右）。在 <https://hadley.shinyapps.io/ms-update-button> 观看动态效果。

以这种方式使用更新功能的方法有很多；当您处理复杂的应用程序时，请留意向用户提供更多信息的方法。一个特别重要的应用程序通过分步过滤，更容易从一长串可能的选项中选择。这通常是“分层选择框”的问题。

10.1.2 Hierarchical select boxes

更新函数的一个更复杂但特别有用的应用是允许跨多个类别进行交互式深入研究。我将用一些来自<https://www.kaggle.com/kyanyoga/sample-sales-data>的销售仪表板的想象数据来说明它们的用法。

```
sales <- vroom::vroom("sales-dashboard/sales_data_sample.csv", col_types = list(), na = "")  
sales %>%  
  select(TERRITORY, CUSTOMERNAME, ORDERNUMBER, everything()) %>%  
  arrange(ORDERNUMBER)  
#> # A tibble: 2,823 × 25  
#>   TERRITORY CUSTOM...¹ ORDER...² QUANT...³ PRICE...⁴ ORDER...⁵ SALES ORDER...⁶ STATUS QTR_ID  
#>   <chr>      <chr>       <dbl>     <dbl>     <dbl>     <dbl> <dbl> <chr>    <chr>    <dbl>  
#> 1 NA          Online ...  10100     30      100        3 5151 1/6/20... Shipp...  1  
#> 2 NA          Online ...  10100     50      67.8       2 3390 1/6/20... Shipp...  1  
#> 3 NA          Online ...  10100     22      86.5       4 1903. 1/6/20... Shipp...  1  
#> 4 NA          Online ...  10100     49      34.5       1 1689. 1/6/20... Shipp...  1  
#> 5 EMEA         Blauer ... 10101     25      100        4 3782 1/9/20... Shipp...  1  
#> 6 EMEA         Blauer ... 10101     26      100        1 3773. 1/9/20... Shipp...  1  
#> 7 EMEA         Blauer ... 10101     45      31.2       3 1404 1/9/20... Shipp...  1  
#> 8 EMEA         Blauer ... 10101     46      53.8       2 2473. 1/9/20... Shipp...  1  
#> 9 NA          Vitachr... 10102     39      100        2 4808. 1/10/2... Shipp...  1  
#> 10 NA         Vitachr... 10102     41      50.1       1 2056. 1/10/2... Shipp...  1  
#> # ... with 2,813 more rows, 15 more variables: MONTH_ID <dbl>, YEAR_ID <dbl>,  
#> # PRODUCTLINE <chr>, MSRP <dbl>, PRODUCTCODE <chr>, PHONE <chr>,  
#> # ADDRESSLINE1 <chr>, ADDRESSLINE2 <chr>, CITY <chr>, STATE <chr>,  
#> # POSTALCODE <chr>, COUNTRY <chr>, CONTACTLASTNAME <chr>,  
#> # CONTACTFIRSTNAME <chr>, DEALSIZE <chr>, and abbreviated variable names  
#> #   ¹CUSTOMERNAME, ²ORDERNUMBER, ³QUANTITYORDERED, ⁴PRICEEACH, ⁵ORDERLINENUMBER,  
#> #   ⁶ORDERDATE
```

对于这个演示，我将重点关注数据中的自然层次结构：

- 每个地区都包含客户。
- 每个客户都有多个订单。
- 每个订单都包含行。

我想创建一个用户界面，您可以在其中：

- 选择一个地区以查看所有客户。
- 选择一个客户以查看所有订单。
- 选择一个订单以查看底层行。

UI的本质很简单：我将创建三个选择框和一个输出表。`customername` 和 `ordernumber` 选择框的选择将动态生成，因此我设置了 `choices = NULL`。

```

ui <- fluidPage(
  selectInput("territory", "Territory", choices = unique(sales$TERRITORY)),
  selectInput("customername", "Customer", choices = NULL),
  selectInput("ordernumber", "Order number", choices = NULL),
  tableOutput("data")
)

```

在server函数中，我自上而下地创建一下代码：

1. 我创建一个反应式表达式 `territory()`，其中包含与所选地区匹配的 `sales` 行。
2. 每当 `territory()` 发生变化时，我都会更新 `input$customername` 选择框中的 `choices` 列表。
3. 我创建了另一个反应式表达式 `customer()`，其中包含与所选客户匹配的 `territory()` 行。
4. 每当 `customer()` 发生变化时，我都会更新 `input$ordernumber` 选择框中的 `choices` 列表。
5. 我在 `output$data` 中显示所选订单。

您可以在下面看到该组织：

```

server <- function(input, output, session) {
  territory <- reactive({
    filter(sales, TERRITORY == input$territory)
  })
  observeEvent(territory(), {
    choices <- unique(territory()$CUSTOMERNAME)
    updateSelectInput(inputId = "customername", choices = choices)
  })

  customer <- reactive({
    req(input$customername)
    filter(territory(), CUSTOMERNAME == input$customername)
  })
  observeEvent(customer(), {
    choices <- unique(customer()$ORDERNUMBER)
    updateSelectInput(inputId = "ordernumber", choices = choices)
  })

  output$data <- renderTable({
    req(input$ordernumber)
    customer() %>%
      filter(ORDERNUMBER == input$ordernumber) %>%
      select(QUANTITYORDERED, PRICEEACH, PRODUCTCODE)
  })
}

```

Territory	Territory	Territory			
NA	EMEA	EMEA			
NA	Customer	Customer			
EMEA	Reims Collectables	Lyon Souveniers			
APAC	Daedalus Designs Imports				
Japan	Herkku Gifts				
10107	Auto Canal Petit				
	La Rochelle Gifts				
	Toys of Finland, Co.				
	Romeo Metal Import				
		Order number			
		10134			
QUANTITYORDERED	PRICEEACH	PRODUCTCODE	QUANTITYORDERED	PRICEEACH	PRODUCTCODE
30.00	95.70	S10_1678	41.00	94.74	S10_1678
39.00	99.91	S10_2016	27.00	100.00	S10_2016
27.00	100.00	S10_4698	28.23		
21.00	100.00	S12_2823	25.00	86.74	S32_4485
29.00	70.87	S18_2625	44.00	74.85	S50_4713
25.00	100.00	S24_1578			
38.00	83.03	S24_2000			
20.00	92.90	S32_1374			

图10.4: 我选择“EMEA”（左），然后选择“Lyon Souveniers”（中），然后（右）查看订单。在<https://hadley.shinyapps.io/ms-update-nested>观看动态效果。在<https://hadley.shinyapps.io/ms-update-nested>上尝试这个简单的示例，或在<https://github.com/hadley/mastering-shiny/tree/master/sales-dashboard>上查看更完整的应用程序。

10.1.3 Freezing reactive inputs

有时，这种分层选择可以短暂地创建一组无效的输入，导致不受欢迎的输出闪烁。例如，考虑这个简单的应用程序，您选择一个数据集，然后选择一个变量进行总结：

```
ui <- fluidPage(
  selectInput("dataset", "Choose a dataset", c("pressure", "cars")),
  selectInput("column", "Choose column", character(0)),
  verbatimTextOutput("summary")
)

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

  observeEvent(input$dataset, {
    updateSelectInput(inputId = "column", choices = names(dataset()))
  })

  output$summary <- renderPrint({
    summary(dataset()[, input$column])
  })
}
```

如果您在<https://hadley.shinyapps.io/ms-freeze>上尝试实时应用程序，您会注意到，当您切换数据集时，摘要输出将短暂闪烁。这是因为`updateSelectInput()`只有在所有输出和观察者运行后才有影响，所以暂时有一个状态，您有数据集B和数据集A中的变量，因此输出包含`summary(NULL)`

您可以通过使用`freezeReactiveValue()`“冻结”输入来解决这个问题。这确保了使用输入的任何反应式表达式或输出，在下一轮完整失效之前都不会更新。

```

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

  observeEvent(input$dataset, {
    freezeReactiveValue(input, "column")
    updateSelectInput(inputId = "column", choices = names(dataset()))
  })

  output$summary <- renderPrint({
    summary(dataset()[[input$column]])
  })
}

```

请注意，无需“解冻”输入值；在Shiny检测到会话和服务器再次同步后，会自动发生这种情况。

您可能想知道何时应该使用 `freezeReactiveValue()`？当您动态更改输入 `value`，始终使用它，实际上是很好的做法。实际修改需要一些时间才能流到浏览器，然后回到Shiny，在此期间，对值的任何读取充其量都是浪费的，最坏的情况是会导致错误。使用 `freezeReactiveValue()` 告诉所有下游计算，输入值是陈旧的，他们应该省下精力，直到它有用。

10.1.4 Circular references

如果您想使用更新函数来更改输入的当前 `value`，我们需要讨论一个重要问题。从Shiny的角度来看，使用更新函数修改 `value` 与用户通过单击或键入修改值没有什么不同。这意味着更新函数可以以与人类完全相同的方式触发更新。这意味着您现在正在超越纯反应式编程的界限，您需要开始担心循环引用和无限循环。

例如，以以下简单的应用程序为例。它包含一个单个输入控件和一个将其值增加一的observer。每次 `updateNumericInput()` 运行时，它都会更改 `input$n`，导致 `updateNumericInput()` 再次运行，因此应用程序被困在无限循环中，不断增加 `input$n` 的值。

```

ui <- fluidPage(
  numericInput("n", "n", 0)
)
server <- function(input, output, session) {
  observeEvent(input$n,
    updateNumericInput(inputId = "n", value = input$n + 1)
)
}

```

您不太可能在自己的应用程序中创建如此明显的问题，但如果更新多个相互依赖的控件，它可能会出现，如下一个示例所示。

10.1.5 Inter-related inputs

一个容易获得循环引用的地方是，当您在应用程序中有多个“真相来源”时。例如，想象一下，您想创建一个温度转换应用程序，您可以在其中以摄氏度或华氏度输入温度：

```

ui <- fluidPage(
  numericInput("temp_c", "Celsius", NA, step = 1),
  numericInput("temp_f", "Fahrenheit", NA, step = 1)
)

```

```

)
server <- function(input, output, session) {
  observeEvent(input$temp_f, {
    c <- round((input$temp_f - 32) * 5 / 9)
    updateNumericInput(inputId = "temp_c", value = c)
  })

  observeEvent(input$temp_c, {
    f <- round((input$temp_c * 9 / 5) + 32)
    updateNumericInput(inputId = "temp_f", value = f)
  })
}

```

如果您玩这个应用程序，<https://hadley.shinyapps.io/ms-temperature>，您会注意到它大部分有效，但您可能会注意到它有时会触发多个更改。例如：

- 设置120 F，然后单击向下箭头。
- F更改为119，C更新为48。
- 48 C转换为118 F，所以F再次变为118。
- 幸运的是，118 F仍然是48 C，所以更新就到此为止。

没有办法解决这个问题，因为你有一个想法（温度），应用程序中有两个表达式（摄氏度和华氏度）。在这里，我们很幸运，循环迅速收敛到满足两个约束的值。一般来说，你最好避免这些情况，除非你愿意非常仔细地分析你创建的基础动态系统的收敛属性。

10.1.6 Exercises

1. 使用更新 `input$date` 的server函数完成下面的用户界面，这样您只能选择 `input$year` 中的日期。

```

ui <- fluidPage(
  numericInput("year", "year", value = 2020),
  dateInput("date", "date")
)

```

2. 用一个server函数完成下面的用户界面，该函数根据 `input$state` 更新 `input$county` 选项。对于额外的挑战，还将路易斯安那州的标签从“县”改为“教区”，将阿拉斯加的标签改为“自治市”。

```

library(openintro, warn.conflicts = FALSE)
#> Loading required package: airports
#> Loading required package: cherryblossom
#> Loading required package: usdata
#> Registered S3 methods overwritten by 'readr':
#>   method           from
#>   as.data.frame.spec_tbl_df vroom
#>   as_tibble.spec_tbl_df    vroom
#>   format.col_spec         vroom
#>   print.col_spec          vroom
#>   print.collector         vroom
#>   print.date_names        vroom
#>   print.locale            vroom

```

```
#> str.col_spec           vroom
states <- unique(county$state)

ui <- fluidPage(
  selectInput("state", "State", choices = states),
  selectInput("county", "County", choices = NULL)
)
```

3. 使用server函数完成下面的用户界面，该功能根据 `input$continent` 更新 `input$country` 选项。使用 `output$data` 显示所有匹配的行。

```
library(gapminder)
continents <- unique(gapminder$continent)

ui <- fluidPage(
  selectInput("continent", "Continent", choices = continents),
  selectInput("country", "Country", choices = NULL),
  tableOutput("data")
)
```

4. 扩展上一个应用程序，以便您还可以选择选择所有大洲，从而查看所有国家。您需要将 "(All)" 添加到选择列表中，然后在过滤时特别处理。
5. <https://community.rstudio.com/t/29307> 中描述的问题的核心是什么？

10.2 Dynamic visibility

复杂性的下一步是有选择地显示和隐藏用户界面的部分内容。如果您了解一点JavaScript和CSS，则有更复杂的方法，但有一种有用的技术不需要任何额外的知识：用选项卡集隐藏可选的UI（如第6.3.1节中介绍）。这是一个聪明的黑客，允许您根据需要，显示和隐藏用户界面，而无需从头开始重新生成（您将在下一节中学习）。

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput("controller", "Show", choices = paste0("panel", 1:3))
    ),
    mainPanel(
      tabsetPanel(
        id = "switcher",
        type = "hidden",
        tabPanelBody("panel1", "Panel 1 content"),
        tabPanelBody("panel2", "Panel 2 content"),
        tabPanelBody("panel3", "Panel 3 content")
      )
    )
  )
)

server <- function(input, output, session) {
  observeEvent(input$controller, {
    updateTabsetPanel(inputId = "switcher", selected = input$controller)
  })
}
```

```
    })  
}
```



图10.5：选择面板1（左），然后是面板2（中间），然后是面板3（右）。在<https://hadley.shinyapps.io/ms-dynamic-panels>观看动态效果。

这里有两个主要想法：

- 使用带有隐藏选项卡的选项卡集面板。
- 使用 `updateTabsetPanel()` 从服务器切换选项卡。

这是一个简单的想法，但当与一点创造力相结合时，它会给你相当大的力量。以下两节说明了几个小例子，说明了如何在实践中使用它。

10.2.1 Conditional UI

想象一下，你想要一个允许用户从正态、均匀和指数分布进行模拟数据的应用程序。每个分布都有不同的参数，因此我们需要一些方法来显示不同分布的不同控件。在这里，我将把每个发行版的唯一用户界面放在它自己的 `tabPanel()`，然后将三个选项卡排列成 `tabsetPanel()`

```
parameter_tabs <- tabsetPanel(  
  id = "params",  
  type = "hidden",  
  tabPanel("normal",  
    numericInput("mean", "mean", value = 1),  
    numericInput("sd", "standard deviation", min = 0, value = 1)  
,  
  tabPanel("uniform",  
    numericInput("min", "min", value = 0),  
    numericInput("max", "max", value = 1)  
,  
  tabPanel("exponential",  
    numericInput("rate", "rate", value = 1, min = 0),  
)  
)
```

然后，我会将其嵌入到更完整的UI中，该UI允许用户选择样本数量并显示结果的直方图：

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      selectInput("dist", "Distribution",  
        choices = c("normal", "uniform", "exponential")  
,  
      numericInput("n", "Number of samples", value = 100),
```

```

    parameter_tabs,
),
mainPanel(
  plotOutput("hist")
)
)
)
)

```

请注意，我已仔细地将 `input$dist` 中的 `choices` 与选项卡面板的名称进行了匹配。这使得编写下面的 `observeEvent()` 代码变得容易，该代码在分发更改时自动切换控件。该应用程序的其余部分使用您已经熟悉的技巧。见图10.6中的最终结果。

```

server <- function(input, output, session) {
  observeEvent(input$dist, {
    updateTabsetPanel(inputId = "params", selected = input$dist)
  })

  sample <- reactive({
    switch(input$dist,
      normal = rnorm(input$n, input$mean, input$sd),
      uniform = runif(input$n, input$min, input$max),
      exponential = rexp(input$n, input$rate)
    )
  })
  output$hist <- renderPlot(hist(sample()), res = 96)
}

```

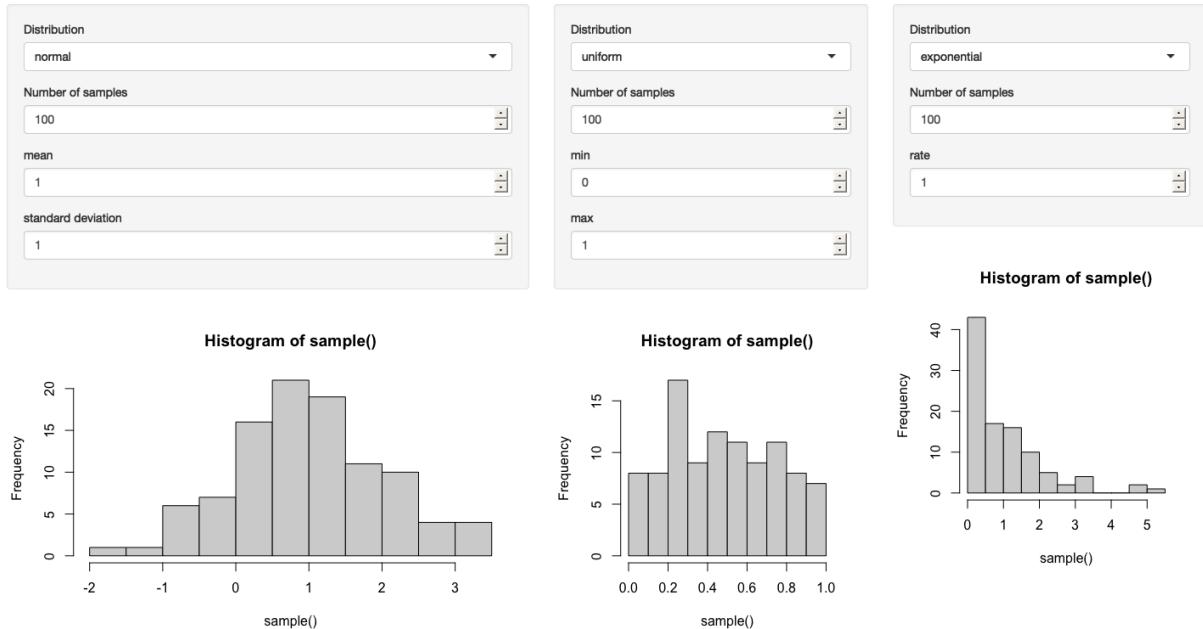


图10.6：正态（左）、均匀（中）和指数（右）分布的结果。在<https://hadley.shinyapps.io/ms-dynamic-conditional>观看直播动态效果。

请注意，（例如）`input$mean` 的值与用户是否可见无关。底层HTML控件仍然存在；你只是看不到它。

10.2.2 Wizard interface

您也可以使用这个想法来创建一个“向导”，这是一种界面类型，通过将信息分布在多个页面上，可以更容易地收集一堆信息。在这里，我们在每个“页面”中嵌入动作按钮，使前进和后退变得容易。结果如图10.7所示。

```
ui <- fluidPage(  
  tabsetPanel(  
    id = "wizard",  
    type = "hidden",  
    tabPanel("page_1",  
      "Welcome!",  
      actionButton("page_12", "next")  
    ),  
    tabPanel("page_2",  
      "Only one page to go",  
      actionButton("page_21", "prev"),  
      actionButton("page_23", "next")  
    ),  
    tabPanel("page_3",  
      "You're done!",  
      actionButton("page_32", "prev")  
    )  
  )  
)  
  
server <- function(input, output, session) {  
  switch_page <- function(i) {  
    updateTabsetPanel(inputId = "wizard", selected = paste0("page_", i))  
  }  
  
  observeEvent(input$page_12, switch_page(2))  
  observeEvent(input$page_21, switch_page(1))  
  observeEvent(input$page_23, switch_page(3))  
  observeEvent(input$page_32, switch_page(2))  
}
```



图10.7：向导界面：在多个页面上部分复杂的UI。在这里，我们用一个非常简单的示例来演示这个想法，单击旁边进入下一页。在<https://hadley.shinyapps.io/ms-wizard>观看动态效果。

注意使用 `switch_page()` 函数来减少服务器代码中的重复量。我们将在第18章中回到这个想法，然后在第19.4.2节中创建一个模块来自动化向导界面。

10.2.3 Exercises

- 仅当用户选中“高级”复选框时，才使用隐藏选项卡集显示其他控件。
- 创建一个应用程序，绘制 `ggplot(diamonds, aes(carat))`，但允许用户选择使用哪个geom：
`geom_histogram()`, `geom_freqpoly()`, `geom_density()`。使用隐藏选项卡集允许用户根据geom选择不同的参数：`geom_histogram()` 和 `geom_freqpoly()` 有一个 `binwidth` 参数；`geom_density()` 有一个 `bw` 参数。
- 修改您在上一个练习中创建的应用程序，允许用户选择是否显示每个geom（即他们可以选择0、1、2或3），而不是总是使用一个geom。确保您可以独立控制直方图和频率分布图的binwidth参数。

10.3 Creating UI with code

有时，上述技术都没有为您提供所需的功能：更新函数仅允许您更改现有输入，并且选项卡集仅在您拥有一组固定和已知的可能组合时有效。有时，您需要根据其他输入创建不同的类型或数量的输入（或输出）。这个最终技术让您有能力做到这一点。

值得注意的是，您总是用代码创建用户界面，但到目前为止，您总是在应用程序启动之前完成。此技术使您能够在应用程序运行时创建和修改用户界面。这个解决方案有两个部分：

- `uiOutput()` 在您的 `ui` 中插入一个占位符。这留下了一个“漏洞”，您的服务器代码稍后可以填充。
- `renderUI()` 在 `server()` 中调用，用动态生成的UI填充占位符。

我们将用一个简单的示例看看这是如何运作的，然后深入研究一些现实的用途。

10.3.1 Getting started

让我们从一个简单的应用程序开始，该应用程序可以动态创建输入控件，通过其他两个输入完成类型和标签控件的产生。生成的应用程序如图10.8所示。

```
ui <- fluidPage(  
 textInput("label", "label"),  
  selectInput("type", "type", c("slider", "numeric")),  
  uiOutput("numeric")  
)  
server <- function(input, output, session) {  
  output$numeric <- renderUI({  
    if (input$type == "slider") {  
      sliderInput("dynamic", input$label, value = 0, min = 0, max = 10)  
    } else {  
      numericInput("dynamic", input$label, value = 0, min = 0, max = 10)  
    }  
  })  
}
```



The figure consists of three side-by-side screenshots of a Shiny application interface. Each screenshot shows a different state of the UI based on the selection in the 'type' dropdown.

- Screenshot 1 (Type: slider):** The 'label' input field contains 'label'. The 'type' dropdown is set to 'slider'. Below it is a slider input with a value of 0, ranging from 0 to 10.
- Screenshot 2 (Type: numeric):** The 'label' input field contains 'label'. The 'type' dropdown is set to 'numeric'. Below it is a numeric input field with a value of 0.
- Screenshot 3 (Type: numeric, with 'label' input):** The 'label' input field contains 'My label'. The 'type' dropdown is set to 'numeric'. Below it is a numeric input field with a value of 0. The numeric input field has a small 'My label' placeholder next to its value.

图10.8：加载应用程序（左），然后将类型更改为数字（中间），然后标签为“my label”。在 <https://hadley.shinyapps.io/ms-render-simple> 观看动态效果。

如果您自己运行此代码，您会注意到它在应用程序加载后需要几分之一秒钟才能显示。这是因为它是反应性的：应用程序必须加载，触发一个反应事件，该事件调用server函数，产生HTML插入到页面中。这是 renderUI() 的缺点之一；过于依赖它可能会造成一个滞后的UI。为了获得良好的性能，请使用本章前面描述的技术，努力尽可能多地固定用户界面。

这种方法还有另一个问题：当您更改控件时，您将丢失当前选择的值。保持现有状态是使用代码创建UI的最大挑战之一。这就是为什么选择性地显示和隐藏UI是更好的方法之一，如果它对你有用的话—因为你没有破坏和重新创建控件，你不需要做任何事情来保留值。然而，在许多情况下，我们可以通过将新输入的 value 设置为现有控件的当前值来解决问题：

```
server <- function(input, output, session) {  
  output$numeric <- renderUI({  
    value <- isolate(input$dynamic)  
    if (input$type == "slider") {  
      sliderInput("dynamic", input$label, value = value, min = 0, max = 10)  
    } else {  
      numericInput("dynamic", input$label, value = value, min = 0, max = 10)  
    }  
  })  
}
```

isolate() 的使用很重要。我们将在第15.4.1节中讲解它的作用，但在这里，它确保了我们不会创建一个反应依赖项，只会在“input\$type”或“input\$label”更改时更改。

10.3.2 Multiple controls

当您生成任意数量或类型的控件时，动态用户界面最有用。这意味着您将使用代码生成UI，我建议使用函数式编程来完成此类任务。在这里，我将使用 purrr::map() 和 purrr::reduce()，您当然也可以使用 lapply() 和 Reduce() 函数做同样的事情。

```
library(purrr)
```

如果您不熟悉函数式编程的 map() 和 reduce()，您可以阅读[函数式编程](#)。我们还将在第18章中回到这个想法。这些都是复杂的想法，所以如果你第一次通读时没有意义，不要有压力。

为了使这个具体化，想象一下您希望用户能够提供自己的调色板。他们将首先指定他们想要多少种颜色，然后为每种颜色提供一个值。ui 非常简单：我们有一个控制输入数量的 numericInput()，一个生成文本框的 uiOutput()，以及一个证明我们已经正确地将所有内容组合在一起的 textOutput()。

```
ui <- fluidPage(  
  numericInput("n", "Number of colours", value = 5, min = 1),  
  uiOutput("col"),  
  textOutput("palette"))
```

server函数很短，但包含了所有需要的计算：

```

server <- function(input, output, session) {
  col_names <- reactive(paste0("col", seq_len(input$n)))

  output$col <- renderUI({
    map(col_names(), ~ textInput(.x, NULL))
  })

  output$palette <- renderText({
    map_chr(col_names(), ~ input[[.x]] %||% "")
  })
}

```

- 我使用一个反应式表达式 `col_names()`，来存储我即将生成的每个颜色输入的名称。
- 然后，我使用 `map()` 创建一个 `textInput()` 列表，每个名称各一个。`renderUI()` 将此HTML组件列表添加到UI中。
- 我需要使用一种新技巧来访问输入值的值。到目前为止，我们总是用 `$` 访问 `input` 的组件，例如 `input$col1`。但在这里，我们有字符向量中的输入名称，如 `var <- "col1"`。`$` 在此场景中不再起作用，因此我们需要转到 `[[`，即 `input[[var]]`。
- 我使用 `map_chr()` 将所有值收集到一个字符矢量中，并显示在 `output$palette` 中。不幸的是，在浏览器呈现新输入之前，有一个短暂的时期，其值为 `NULL`。这导致 `map_chr()` 出错，我们通过使用 `%||%` 函数来修复该错误：每当左侧为 `NULL`，它都会返回右侧的值。

您可以在图10.9中看到结果。

图10.9：加载应用程序（左），将n设置为3（中间），然后输入一些颜色（右）。在 <https://hadley.shinyapps.io/ms-render-palette> 观看动态效果。

如果您运行此应用程序，您会发现一种非常烦人的行为：每当您更改颜色数量时，您输入的所有数据都会消失。我们可以通过使用与以前相同的技术来解决这个问题：将 `value` 设置为 `(isolate)` 当前值。我还会调整外观以看起来更好一点，包括在绘图中显示选定的颜色。示例屏幕截图如图10.10所示。

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      numericInput("n", "Number of colours", value = 5, min = 1),
      uiOutput("col"),
    ),
    mainPanel(
      plotOutput("plot")
    )
)

```

```

}

server <- function(input, output, session) {
  col_names <- reactive(paste0("col", seq_len(input$n)))

  output$col <- renderUI({
    map(col_names(), ~ textInput(.x, NULL, value = isolate(input[[.x]])))
  })

  output$plot <- renderPlot({
    cols <- map_chr(col_names(), ~ input[[.x]] %||% "")
    # convert empty inputs to transparent
    cols[cols == ""] <- NA

    barplot(
      rep(1, length(cols)),
      col = cols,
      space = 0,
      axes = FALSE
    )
  }, res = 96)
}

```

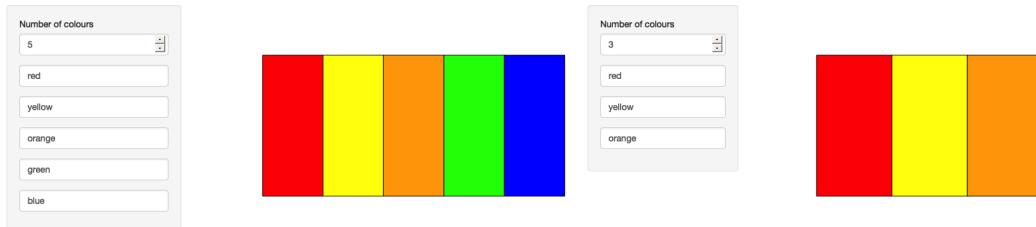


图10.10：填写彩虹的颜色（左），然后将颜色数量减少到3（右）；请注意，保留现有颜色。在 <https://hadley.shinyapps.io/ms-render-palette-full> 观看动态效果。

10.3.3 Dynamic filtering

为了结束这一章，我将创建一个应用程序，允许您动态过滤任何数据框。每个数字类型列将获得一个范围滑块，每个因子变量类型列将获得一个多重选择操作，因此（例如）如果一个数据框有三个数字类型列和两个因子类型列，该应用程序将有三个滑块和两个选择框。

我将从一个为单个变量创建UI的函数开始。它将返回数字输入的范围滑块，因子输入的多重选择，所有其他类型的 `NULL`（无）。

```

make_ui <- function(x, var) {
  if (is.numeric(x)) {
    rng <- range(x, na.rm = TRUE)
    sliderInput(var, var, min = rng[1], max = rng[2], value = rng)
  } else if (is.factor(x)) {
    levs <- levels(x)
    selectInput(var, var, choices = levs, selected = levs, multiple = TRUE)
  } else {
    # Not supported
    NULL
  }
}

```

然后，我将编写该函数的服务器端等价代码：它接受输入控件的变量和值，并返回一个逻辑向量，说明是否包含每个观察。使用逻辑向量可以轻松组合来自多列的结果。

```

filter_var <- function(x, val) {
  if (is.numeric(x)) {
    !is.na(x) & x >= val[1] & x <= val[2]
  } else if (is.factor(x)) {
    x %in% val
  } else {
    # No control, so don't filter
    TRUE
  }
}

```

然后，我可以“手动”使用这些函数为 `iris` 数据集生成一个简单的过滤用户界面：

```

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      make_ui(iris$Sepal.Length, "Sepal.Length"),
      make_ui(iris$Sepal.Width, "Sepal.Width"),
      make_ui(iris$Species, "Species")
    ),
    mainPanel(
      tableOutput("data")
    )
  )
)

server <- function(input, output, session) {
  selected <- reactive({
    filter_var(iris$Sepal.Length, input$Sepal.Length) &
      filter_var(iris$Sepal.Width, input$Sepal.Width) &
      filter_var(iris$Species, input$Species)
  })

  output$data <- renderTable(head(iris[selected(), ], 12))
}

```

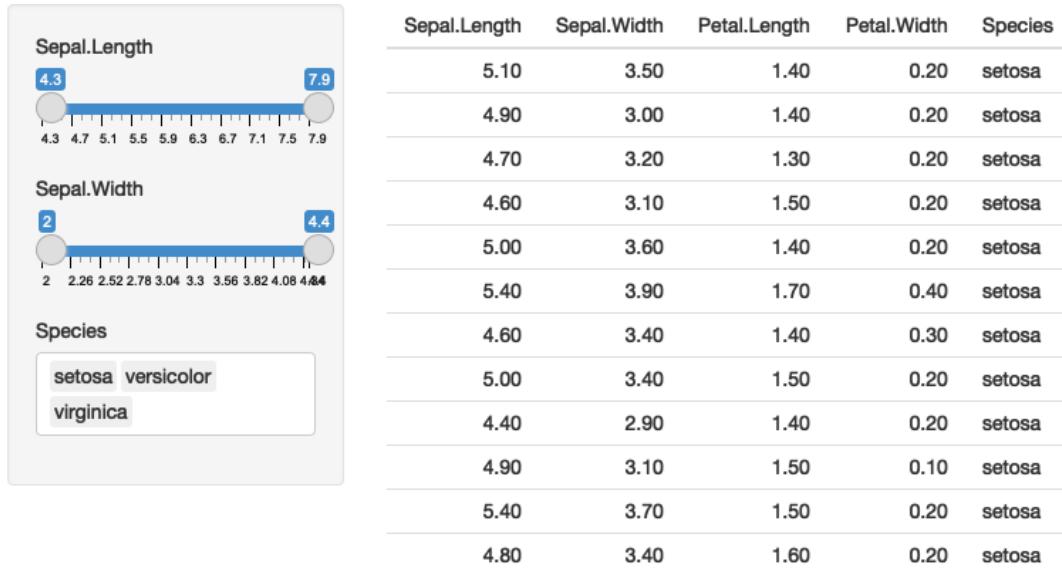


图10.11: iris数据集的简单过滤器界面

你可能会注意到，我厌倦了复制和粘贴，所以该应用程序只能使用三列。我可以通过使用一点函数式编程使其与所有列一起工作：

- 在 `ui` 中使用 `map()` 为每个变量生成一个控件。
- 在 `server()` 我使用 `map()` 为每个变量生成选择向量。然后，我使用 `reduce()` 获取每个变量的逻辑向量，并通过 `&` 将每个向量组合成一个单一的逻辑向量。

同样，如果您不了解这里到底发生了什么，请不要太担心。主要的一点是，一旦你掌握了函数式编程，你就可以编写非常简洁的代码，生成复杂的动态应用程序。

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      map(names(iris), ~ make_ui(iris[ [.x ] ], .x))
    ),
    mainPanel(
      tableOutput("data")
    )
  )
)
server <- function(input, output, session) {
  selected <- reactive({
    each_var <- map(names(iris), ~ filter_var(iris[ [.x ] ], input[ [.x ] ]))
    reduce(each_var, ~ .x & .y)
  })

  output$data <- renderTable(head(iris[selected(), ], 12))
}
```

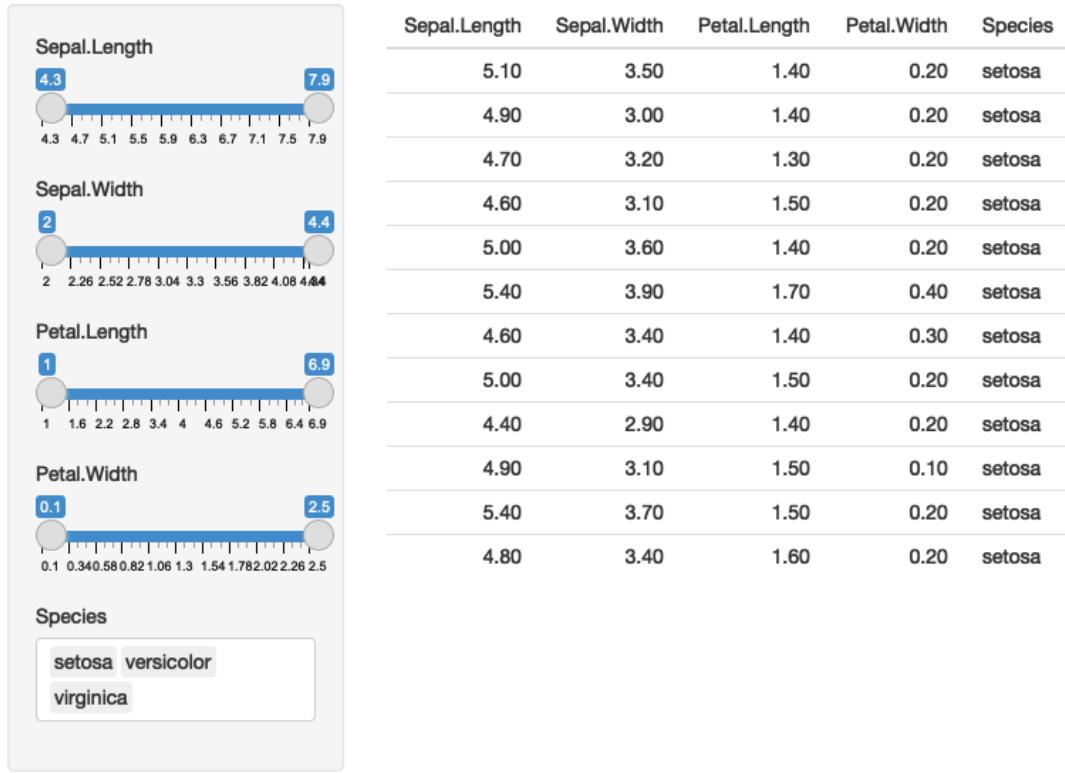


图10.12：使用函数式编程为 `iris` 数据集构建过滤应用程序。

从那里开始，使用任何数据框都是一个简单的概括。在这里，我将使用datasets包中的数据框来说明它，但您很容易地想象如何将其扩展到用户上传的数据。见图10.13中的结果。

```
dfs <- keep(ls("package:datasets"), ~ is.data.frame(get(.x, "package:datasets")))

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      selectInput("dataset", label = "Dataset", choices = dfs),
      uiOutput("filter")
    ),
    mainPanel(
      tableOutput("data")
    )
  )
)
server <- function(input, output, session) {
  data <- reactive({
    get(input$dataset, "package:datasets")
  })
  vars <- reactive(names(data()))

  output$filter <- renderUI(
    map(vars(), ~ make_ui(data()[[.x]], .x))
  )

  selected <- reactive({
    each_var <- map(vars(), ~ filter_var(data()[[.x]], input[[.x]]))
  })
}
```

```

    reduce(each_var, `&`)
  })

output$data <- renderTable(head(data()[selected(), ], 12))
}

```

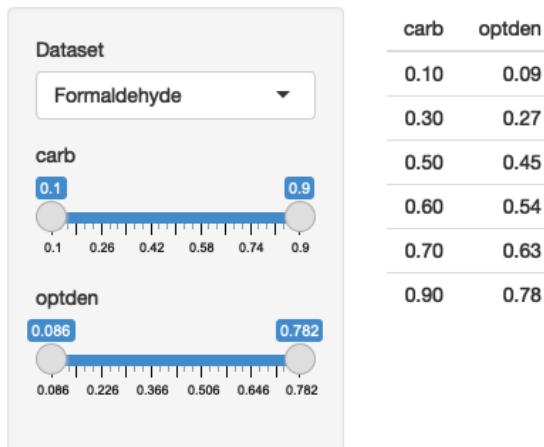


图10.13：从所选数据集的字段自动生成的动态用户界面。在<https://hadley.shinyapps.io/ms-filtering-final>观看动态效果。

10.3.4 Dialog boxes

在我们结束之前，我想提一个相关技术：对话框（dialog boxes）。您已经在第8.4.1节中看到了它们，其中对话框的内容是一个固定的文本字符串。但由于`modalDialog()`是从`server`函数中调用的，您实际上可以以与`renderUI()`相同的方式动态生成内容。如果您想在继续常规应用程序流程之前强迫用户做出一些决定，这是一项有用的技术。

10.3.5 Exercises

- 根据本节中的初始示例，采用这个非常简单的应用程序：

```

ui <- fluidPage(
  selectInput("type", "type", c("slider", "numeric")),
  uiOutput("numeric")
)
server <- function(input, output, session) {
  output$numeric <- renderUI({
    if (input$type == "slider") {
      sliderInput("n", "n", value = 0, min = 0, max = 100)
    } else {
      numericInput("n", "n", value = 0, min = 0, max = 100)
    }
  })
}

```

您如何使用动态可见性来实现它？如果您实现动态可见性，那么在更改控件时如何保持值同步？

- 解释这个应用程序是如何工作的。为什么当你第二次点击输入密码按钮时，密码会消失？

```

ui <- fluidPage(
  actionButton("go", "Enter password"),
  textOutput("text")
)
server <- function(input, output, session) {
  observeEvent(input$go, {
    showModal(modalDialog(
      passwordInput("password", NULL),
      title = "Please enter your password"
    )))
  })

  output$text <- renderText({
    if (!isTruthy(input$password)) {
      "No password"
    } else {
      "Password entered"
    }
  })
}

```

3. 在第[10.3.1](#)节的应用程序中，如果您从 `value <- isolate(input$dynamic)` 中删除 `isolate()` 会发生什么？
4. 添加对日期和日期时间列 `make_ui()` 和 `filter_var()` 的支持。
5. （高级）如果您知道[S3 OOP](#)系统，请考虑如何用通用函数替换 `make_ui()` 和 `filter_var()` 中的 `if` 块。

10.4 Summary

在阅读本章之前，您仅限于在运行 `server` 函数之前静态创建用户界面。现在，您已经学会了如何修改用户界面并完全重新创建它以响应用户操作。动态用户界面将极大地增加应用程序的复杂性，因此，如果您发现自己难以调试正在发生的事情，请不要感到惊讶。始终记得使用最简单的技术来解决问题，并回到第[5.2](#)节中的调试建议。

下一章切换策略来谈论书签（bookmarking），从而可以与他人共享应用程序的当前状态。

Chapter 11. Bookmarking

默认情况下，与大多数网站相比，Shiny应用程序有一个主要缺点：您无法将应用程序添加为书签以返回同一位置，也无法通过电子邮件中的链接与他人共享您的工作。这是因为，默认情况下，Shiny不会在其URL中公开应用程序的当前状态。然而，幸运的是，您可以通过一些额外的工作来改变这种行为，本章将向您展示如何操作。

```
library(shiny)
```

11.1 Basic idea

让我们用一个简单的应用程序来制作一个可以添加书签的应用程序。这个应用程序绘制了Lissajous图，这些图复制了钟摆的运动。这个应用程序可以产生各种有趣的模式，您可能想要分享。

```
ui <- fluidPage(  
  sidebarLayout(  
    sidebarPanel(  
      sliderInput("omega", "omega", value = 1, min = -2, max = 2, step = 0.01),  
      sliderInput("delta", "delta", value = 1, min = 0, max = 2, step = 0.01),  
      sliderInput("damping", "damping", value = 1, min = 0.9, max = 1, step = 0.001),  
      numericInput("length", "length", value = 100)  
    ),  
    mainPanel(  
      plotOutput("fig")  
    )  
  )  
)  
  
server <- function(input, output, session) {  
  t <- reactive(seq(0, input$length, length.out = input$length * 100))  
  x <- reactive(sin(input$omega * t() + input$delta) * input$damping ^ t())  
  y <- reactive(sin(t()) * input$damping ^ t())  
  
  output$fig <- renderPlot({  
    plot(x(), y(), axes = FALSE, xlab = "", ylab = "", type = "l", lwd = 2)  
  }, res = 96)  
}  
}
```

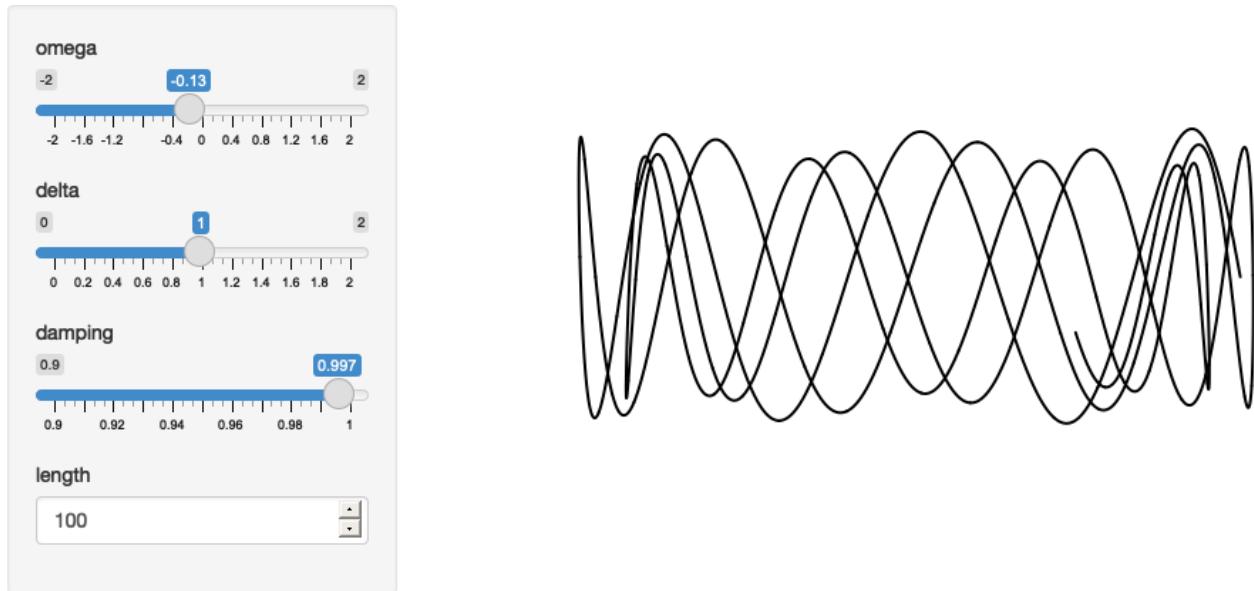


图11.1：此应用程序允许您使用钟摆模型生成感兴趣的数字。和你的朋友分享一个链接不是很酷吗？

我们需要做三件事，才能使这个应用程序可以分享（书签）：

1. 在用户界面中添加 `bookmarkButton()`。这会生成一个按钮，用户单击该按钮以生成书签的URL。
2. 将 `ui` 变成一个函数。您需要这样做，因为已添加书签的应用程序必须重放已添加书签的值：有效地，Shiny 修改了每个输入控件的默认 `value`。这意味着不再有单个静态UI，而是多个可能的UI，这些UI依赖于URL中

的参数；即它必须是一个函数。

3. 将 `enableBookmarking = "url"` 添加到 `shinyApp()` 调用中。

做出这些改变：

```
ui <- function(request) {  
  fluidPage(  
    sidebarLayout(  
      sidebarPanel(  
        sliderInput("omega", "omega", value = 1, min = -2, max = 2, step = 0.01),  
        sliderInput("delta", "delta", value = 1, min = 0, max = 2, step = 0.01),  
        sliderInput("damping", "damping", value = 1, min = 0.9, max = 1, step = 0.001),  
        numericInput("length", "length", value = 100),  
        bookmarkButton()  
      ),  
      mainPanel(  
        plotOutput("fig")  
      )  
    )  
  )  
}  
shinyApp(ui, server, enableBookmarking = "url")
```

您可以在<https://hadley.shinyapps.io/ms-bookmark-url>上试用。如果您调试该应用程序并将一些有趣的状态添加为书签，您将看到生成的URL看起来像这样：

- `https://hadley.shinyapps.io/ms-bookmark-url/?_inputs_&damping=1&delta=1&length=100&omega=1`
- `https://hadley.shinyapps.io/ms-bookmark-url/?_inputs_&damping=0.966&delta=1.25&length=100&omega=-0.54`
- `https://hadley.shinyapps.io/ms-bookmark-url/?_inputs_&damping=0.997&delta=1.37&length=500&omega=-0.9`

为了了解正在发生的事情，让我们把第一个URL分成几块：

- `http://` 是用于与应用程序通信的“协议”。这将始终是 `http` 或 `https`。
- `hadley.shinyapps.io/ms-bookmark-url` 是应用程序的位置。
- `?_inputs_&damping=1&delta=1&length=100&omega=1` 之后的一切是“参数”。每个参数都由 `&` 分隔，如果您将其分开，您可以在应用程序中看到每个输入的值：
 - `damping=1`
 - `delta=1`
 - `length=100`
 - `omega=1`

因此，“生成书签”意味着在URL参数中记录输入的当前值。如果你在本地构建的shiny，网址看起来会略有不同：

- `http://127.0.0.1:4087/?_inputs_&damping=1&delta=1&length=100&omega=1`
- `http://127.0.0.1:4087/?_inputs_&damping=0.966&delta=1.25&length=100&omega=-0.54`
- `http://127.0.0.1:4087/?_inputs_&damping=0.997&delta=1.37&length=500&omega=-0.9`

大多数作品都是相同的，只是你会看到类似 `127.0.0.1:4087` 的描述，而不是 `hadley.shinyapps.io/ms-bookmark-url`。`127.0.0.1` 是一个始终指向您自己的计算机的特殊地址，`4087` 是一个随机分配的端口。通常，不同的应用程序会获得不同的路径或IP地址，但当您在自己的计算机上托管多个应用程序时，这是不可能的。

11.1.1 Updating the URL

另一个选项不是提供显式按钮，而是在浏览器中自动更新URL。这允许您的用户在浏览器中使用用户书签命令，或从位置栏复制和粘贴URL。

自动更新URL需要在server函数中加入一些样板：

```
# Automatically bookmark every time an input changes
observe({
  reactiveValuesToList(input)
  session$doBookmark()
})
# Update the query string
onBookmarked(updateQueryString)
```

这为我们提供了以下更新的服务器功能：

```
server <- function(input, output, session) {
  t <- reactive(seq(0, input$length, length = input$length * 100))
  x <- reactive(sin(input$omega * t() + input$delta) * input$damping ^ t())
  y <- reactive(sin(t()) * input$damping ^ t())

  output$fig <- renderPlot({
    plot(x(), y(), axes = FALSE, xlab = "", ylab = "", type = "l", lwd = 2)
  }, res = 96)

  observe({
    reactiveValuesToList(input)
    session$doBookmark()
  })
  onBookmarked(updateQueryString)
}
shinyApp(ui, server, enableBookmarking = "url")
```

这产生了<https://hadley.shinyapps.io/ms-bookmark-auto>—由于URL现在会自动更新，您现在可以从用户界面中删除书签按钮。

11.1.2 Storing richer state

到目前为止，我们已经使用了 `enableBookmarking = "url"`，它将状态直接存储在URL中。这是一个很好的开始，因为它非常简单，并且可以在任何地方部署Shiny应用程序。然而，正如您可以想象的那样，如果您有大量输入，URL将变得非常长，而且显然无法捕获上传的文件。

对于这些情况，您可能希望使用 `enableBookmarking = "server"`，它将状态保存到服务器上的 `.rds` 文件。这总是会生成一个简短、不透明的URL，但需要在服务器上进行额外的存储。

```
shinyApp(ui, server, enableBookmarking = "server")
```

shinyapps.io目前不支持服务器端书签，因此您需要在本地尝试此功能。如果您这样做，您将看到书签按钮生成URL，例如：

- `http://127.0.0.1:4087/?_state_id_=0d645f1b28f05c97`
- `http://127.0.0.1:4087/?_state_id_=87b56383d8a1062c`
- `http://127.0.0.1:4087/?_state_id_=c8b0291ba622b69c`

它们与您工作目录中的匹配目录配对：

- `shiny_bookmarks/0d645f1b28f05c97`
- `shiny_bookmarks/87b56383d8a1062c`
- `shiny_bookmarks/c8b0291ba622b69c`

服务器书签的主要缺点是它需要将文件保存在服务器上，而且这些文件需要保留多长时间并不清楚。如果您将复杂状态添加为书签，并且从未删除过这些文件，您的应用程序将随着时间的推移占用越来越多的磁盘空间。如果您确实删除了文件，一些旧书签将停止工作。

11.2 Bookmarking challenges

自动书签依赖于反应图。它用保存的值填充输入，然后重新执行所有反应表达式和输出，只要您的应用程序的反应图是比较直观的，这将产生您看到的相同应用程序。本节简要介绍了一些需要额外维护的案例：

- 如果您的应用程序使用随机数，即使所有输入都相同，结果也可能不同。如果始终生成相同的数字真的很重要，您需要考虑如何使您的随机过程可重现。最简单的方法是使用`repeatable()`。有关更多详细信息，请参阅文档。
- 如果您有选项卡，并希望将活动选项卡添加为书签并恢复，请确保在对`tabsetPanel()`的调用中提供`id`
- 如果有不应添加书签的输入，例如，它们包含不应共享的私人信息，请在`server`函数的某个地方调用`setBookmarkExclude()`。例如，`setBookmarkExclude(c("secret1", "secret2"))`，将确保`secret1`和`secret2`输入不被添加书签。
- 如果您在自己的`reactiveValues()`对象中手动管理反应状态（正如我们将在第16章中讨论的那样），则需要使用`onBookmark()`和`onRestore()`回调，来手动保存和加载您的附加状态。有关更多详细信息，请参阅[高级书签](#)。

11.3 Exercises

1. 生成应用程序来可视化`ambient::noise_simplex()`的结果。您的应用程序应该允许用户控制频率、形状、空白度和增益，并可添加书签。如何确保从书签重新加载时图像看起来完全相同？（想想`seed`论点意味着什么）。
2. 制作一个简单的应用程序，允许您上传csv文件，然后将其添加为书签。上传一些文件，然后查看`shiny_bookmarks`。文件如何与书签对应？（提示：您可以使用`readRDS()`查看Shiny正在生成的缓存文件）。

11.4 Summary

本章展示了如何为您的应用程序启用书签。这是为用户提供的绝佳功能，因为它允许他们轻松与他人共享工作。接下来，我们将讨论如何在Shiny应用程序中使用整洁的评估。整洁评估是许多tidyverse函数的一个功能，如果您想允许用户更改（例如）dplyr管道或ggplot2图形中的变量，您需要了解它。

Chapter 12. Tidy evaluation

如果您将Shiny与tidyverse一起使用，您几乎肯定会遇到具有整洁评估的编程挑战。整个tidyverse都使用Tidy评估，以使交互式数据探索更加流畅，但它附带成本：很难间接引用变量，因此更难编程。

在本章中，您将学习如何在shiny应用程序中包装ggplot2和dplyr函数。（如果您不使用tidyverse，您可以跳过本章😊。）在函数和包中包装ggplot2和dplyr函数的技术略有不同，其他资源涵盖了这些资源，例如[Using ggplot2 in packages](#)或[Programming with dplyr](#)。

```
library(shiny)
library(dplyr, warn.conflicts = FALSE)
library(ggplot2)
```

12.1 Motivation

想象一下，我想创建一个应用程序，允许您过滤数值类型列以选择大于阈值的行。你可能会写这样的东西：

```
num_vars <- c("carat", "depth", "table", "price", "x", "y", "z")
ui <- fluidPage(
  selectInput("var", "Variable", choices = num_vars),
  numericInput("min", "Minimum", value = 1),
  tableOutput("output")
)
server <- function(input, output, session) {
  data <- reactive(diamonds %>% filter(input$var > input$min))
  output$output <- renderTable(head(data()))
}
```

The screenshot shows a Shiny user interface. At the top, there is a dropdown menu labeled 'Variable' with 'carat' selected. Below it is a numeric input field labeled 'Minimum' with the value '1'. At the bottom is a table output showing the first few rows of a dataset. The columns are labeled 'carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y', and 'z'. The rows show data points where the 'carat' value is greater than 1.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.50	55.00	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.80	61.00	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.90	65.00	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.40	58.00	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.30	58.00	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.80	57.00	336	3.94	3.96	2.48

图12.1：一个试图在用户选择的变量上选择大于阈值的行的应用程序。

如图12.1所示，该应用程序运行时没有错误，但它没有返回正确的结果—所有行中`carat`的值都小于1。本章的目标是帮助您了解为什么在shiny中不起作用，以及为什么dplyr认为`filter(diamonds, "carat" > 1)`成立。

这是一个间接问题：通常使用tidyverse函数时，您直接在函数调用中键入变量的名称。但现在你想间接引用它：变量（`carat`）存储在另一个变量（`input$var`）中。

这句话对你来说可能有字面意义，但它有点令人困惑，因为我使用“变量”来表示两个略微不同的东西。如果我们能通过引入两个新术语来消除这两种用途的歧义，就更容易理解发生了什么：

- **env变量**（环境变量）是您使用`<-`创建的“编程”变量。`input$var`是env变量。
- **数据变量**（数据框变量）是位于数据框内的“统计”变量。`carat`是数据变量。

有了这些新术语，我们可以使间接问题更加清晰：我们在env变量（`input$var`）中存储了一个数据变量（`carat`），我们需要一些方法来告诉dplyr这一点。根据您正在使用的函数是“数据屏蔽”函数还是“整洁选择”函数，有两种略微不同的方法可以做到这一点。

12.2 Data-masking

数据屏蔽函数允许您在“当前”数据框中使用变量，而无需任何额外的语法。它用于许多dplyr函数，如`arrange()`, `filter()`, `group_by()`, `mutate()`和`summarise()`以及ggplot2的`aes()`。数据屏蔽很有用，因为它允许您在没有任何额外语法的情况下使用数据变量。

12.2.1 Getting started

让我们从这个对`filter()`的调用开始，它使用数据变量（`carat`）和env变量（`min`）：

```
min <- 1
diamonds %>% filter(carat > min)
#> # A tibble: 17,502 × 10
#>   carat     cut      color clarity depth table price     x     y     z
#>   <dbl> <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1  1.17 Very Good J      I1     60.2    61  2774  6.83  6.9   4.13
#> 2  1.01 Premium   F      I1     61.8    60  2781  6.39  6.36  3.94
#> 3  1.01 Fair      E      I1     64.5    58  2788  6.29  6.21  4.03
#> 4  1.01 Premium   H      SI2    62.7    59  2788  6.31  6.22  3.93
#> 5  1.05 Very Good J      SI2    63.2    56  2789  6.49  6.45  4.09
#> 6  1.05 Fair      J      SI2    65.8    59  2789  6.41  6.27  4.18
#> # ... with 17,496 more rows
```

将其与base-R等价代码进行比较：

```
diamonds[diamonds$carat > min, ]
```

在大多数（但不是全部）base-R函数中，您必须引用`$`，访问数据变量。这意味着您通常必须多次重复数据框的名称，但确实清楚地说明了什么是数据变量，什么是env变量。它还使使用间接变得简单，因为您可以将数据变量的名称存储在env变量中，然后从`$`切换到`[[`：

```
var <- "carat"
diamonds[diamonds[[var]] > min, ]
```

我们如何通过整洁的评估获得相同的结果？我们需要一些方法将`$`重新添加到图片中。幸运的是，在数据屏蔽函数中，如果您想明确使用的是数据变量还是env变量，您可以使用`.data`或`.env`：

```
diamonds %>% filter(.data$carat > .env$min)
```

现在我们可以从`$`切换到`[[`：

```
diamonds %>% filter(.data[[var]] > .env$min)
```

让我们通过更新我们本章开始撰写的server函数，来检查它是否有效：

```
num_vars <- c("carat", "depth", "table", "price", "x", "y", "z")
ui <- fluidPage(
  selectInput("var", "Variable", choices = num_vars),
  numericInput("min", "Minimum", value = 1),
  tableOutput("output")
)
server <- function(input, output, session) {
  data <- reactive(diamonds %>% filter(.data[[input$var]] > .env$input$min))
  output$output <- renderTable(head(data()))
}
```

carat	cut	color	clarity	depth	table	price	x	y	z
1.17	Very Good	J	I1	60.20	61.00	2774	6.83	6.90	4.13
1.01	Premium	F	I1	61.80	60.00	2781	6.39	6.36	3.94
1.01	Fair	E	I1	64.50	58.00	2788	6.29	6.21	4.03
1.01	Premium	H	SI2	62.70	59.00	2788	6.31	6.22	3.93
1.05	Very Good	J	SI2	63.20	56.00	2789	6.49	6.45	4.09
1.05	Fair	J	SI2	65.80	59.00	2789	6.41	6.27	4.18

图12.2：我们的应用程序现在可以工作了，因为我们已经明确了`.data`和`.env`及`[[`与`$`。在<https://hadley.shinyapps.io/ms-tidied-up>观看动态效果。

图12.2显示我们取得了成功—我们只看到`carat`值大于1的结果。现在您已经了解了基础知识，我们将开发一些更现实但仍然简单的shiny应用程序。

12.2.2 Example: ggplot2

让我们将这个想法应用于动态图，允许用户通过选择出现在`x`轴和`y`轴上的变量，来创建散点图。结果如图12.3所示。

```

ui <- fluidPage(
  selectInput("x", "X variable", choices = names(iris)),
  selectInput("y", "Y variable", choices = names(iris)),
  plotOutput("plot")
)
server <- function(input, output, session) {
  output$plot <- renderPlot({
    ggplot(iris, aes(.data[[input$x]], .data[[input$y]])) +
      geom_point(position = ggforce::position_auto())
  }, res = 96)
}

```

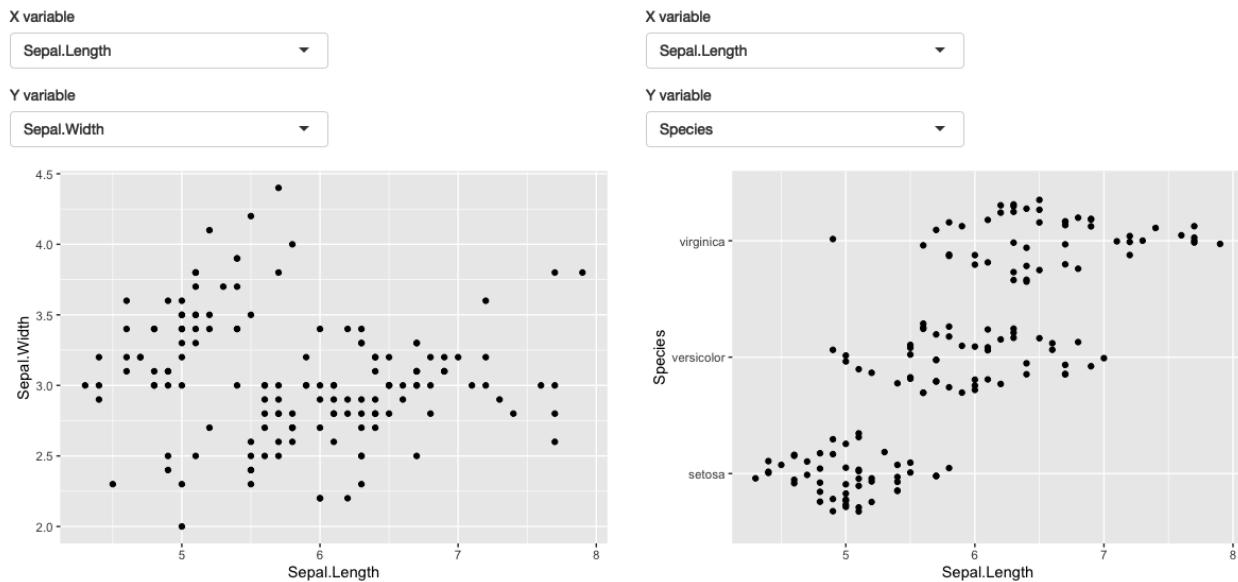


图12.3：一个简单的应用程序，允许您选择哪些变量绘制在 x 轴和 y 轴上。在<https://hadley.shinyapps.io/ms-ggplot2>观看动态效果。

在这里，我使用了 `ggforce::position_auto()`。无论x和y变量是连续的还是离散的，`geom_point()`都能很好地工作。或者，我们可以允许用户选择geom。以下应用程序使用 `switch()` 语句生成一个反应性geom，稍后添加到绘图中。

```

ui <- fluidPage(
  selectInput("x", "X variable", choices = names(iris)),
  selectInput("y", "Y variable", choices = names(iris)),
  selectInput("geom", "geom", c("point", "smooth", "jitter")),
  plotOutput("plot")
)
server <- function(input, output, session) {
  plot_geom <- reactive({
    switch(input$geom,
      point = geom_point(),
      smooth = geom_smooth(se = FALSE),
      jitter = geom_jitter()
    )
  })
  output$plot <- renderPlot({

```

```

ggplot(iris, aes(.data[[input$x]], .data[[input$y]])) +
  plot_geom()
}, res = 96
}

```

这是使用用户选择的变量进行编程的挑战之一：您的代码必须变得更加复杂，以处理用户可能生成的所有情况。

12.2.3 Example: dplyr

同样的技术也适用于dplyr。以下应用程序扩展了之前的简单示例，允许您选择要过滤的变量、要选择的最小值和要排序的变量。

```

ui <- fluidPage(
  selectInput("var", "Select variable", choices = names(mtcars)),
  sliderInput("min", "Minimum value", 0, min = 0, max = 100),
  selectInput("sort", "Sort by", choices = names(mtcars)),
  tableOutput("data")
)
server <- function(input, output, session) {
  observeEvent(input$var, {
    rng <- range(mtcars[[input$var]])
    updateSliderInput(
      session, "min",
      value = rng[[1]],
      min = rng[[1]],
      max = rng[[2]]
    )
  })
}

output$data <- renderTable({
  mtcars %>%
    filter(.data[[input$var]] > input$min) %>%
    arrange(.data[[input$sort]])
})
}

```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
33.90	4.00	71.10	65.00	4.22	1.83	19.90	1.00	1.00	4.00	1.00
30.40	4.00	75.70	52.00	4.93	1.61	18.52	1.00	1.00	4.00	2.00
32.40	4.00	78.70	66.00	4.08	2.20	19.47	1.00	1.00	4.00	1.00
27.30	4.00	79.00	66.00	4.08	1.94	18.90	1.00	1.00	4.00	1.00
30.40	4.00	95.10	113.00	3.77	1.51	16.90	1.00	1.00	5.00	2.00
26.00	4.00	120.30	91.00	4.43	2.14	16.70	0.00	1.00	5.00	2.00

图12.4: 一个简单的应用程序，允许您选择一个变量到阈值，并选择如何对结果进行排序。在 <https://hadley.shinyapps.io/ms-dplyr> 观看动态效果。

大多数其他问题可以通过将 `.data` 与您现有的编程技能相结合来解决。例如，如果您想有条件地按升序或降序排序呢？

```
ui <- fluidPage(  
  selectInput("var", "Sort by", choices = names(mtcars)),  
  checkboxInput("desc", "Descending order?"),  
  tableOutput("data")  
)  
server <- function(input, output, session) {  
  sorted <- reactive({  
    if (input$desc) {  
      arrange(mtcars, desc(.data[[input$var]]))  
    } else {  
      arrange(mtcars, .data[[input$var]])  
    }  
  })  
  output$data <- renderTable(sorted())  
}
```

随着您提供更多的控制，您会发现您的代码变得越来越复杂，创建全面和用户友好的用户界面变得越来越困难。这就是为什么我一直专注于用于数据分析的代码工具：创建好的用户界面真的很难！

12.2.4 User supplied data

在我们继续讨论整洁的选择之前，我们需要讨论最后一个主题：用户提供的数据。以图12.5所示的这个应用程序为例：它允许用户上传一个tsv文件，然后选择一个变量并按其进行过滤。它将适用于您可能尝试的绝大多数输入。

```
ui <- fluidPage(  
  fileInput("data", "dataset", accept = ".tsv"),  
  selectInput("var", "var", character()),  
  numericInput("min", "min", 1, min = 0, step = 1),  
  tableOutput("output")  
)  
server <- function(input, output, session) {  
  data <- reactive({  
    req(input$data)  
    vroom::vroom(input$data$datapath)  
  })  
  observeEvent(data(), {  
    updateSelectInput(session, "var", choices = names(data()))  
  })  
  observeEvent(input$var, {  
    val <- data()[[input$var]]  
    updateNumericInput(session, "min", value = min(val))  
  })  
  
  output$output <- renderTable({  
    req(input$var)
```

```

data() %>%
  filter(.data[[input$var]] > input$min) %>%
  arrange(.data[[input$var]]) %>%
  head(10)
}
}

```

dataset
 No file selected

var

min

图12.5：一个过滤用户提供的数据的应用程序，具有令人惊讶的故障模式，见实时<https://hadley.shinyapps.io/m-s-user-supplied>。

这里使用 `filter()` 有一个微妙的问题。我们将上述部分代码提取出来，将其在base-R环境中执行分析。示例如下：

```

df <- data.frame(x = 1, y = 2)
input <- list(var = "x", min = 0)

df %>% filter(.data[[input$var]] > input$min)
#>     x y
#> 1 1 2

```

如果您尝试此代码，您会发现它似乎对绝大多数数据框都工作正常。然而，有一个微妙的问题：如果数据框中包含一个称为 `input` 的变量，会发生什么？

```

df <- data.frame(x = 1, y = 2, input = 3)
df %>% filter(.data[[input$var]] > input$min)
#> Error in `filter()`:
#> ! Problem while computing `..1 = .data[["x"]]] > input$min` .
#> Caused by error in `input$min`:
#> ! $ operator is invalid for atomic vectors

```

我们收到一条错误消息，因为 `filter()` 正在尝试评估 `df$input$min`：

```

df$input$min
#> Error in df$input$min: $ operator is invalid for atomic vectors

```

这个问题是由于数据变量和env变量的模糊性，以及如果两者都可用，数据屏蔽更喜欢使用数据变量。我们可以通
过使用 `.env`，告诉 `filter()` 在env变量中只查找 `min` 来解决这个问题：

```
df %>% filter(.data[[input$var]] > .env$input$min)
#>   x y input
#> 1 1 2      3
```

请注意，在处理用户提供的数据时，您只需要担心这个问题；在处理自己的数据时，您可以确保数据变量的名称不会与env变量的名称发生冲突（如果它们意外发生，您将立即发现它）。

12.2.5 Why not use base R?

此时，您可能会想知道没有`filter()`是否更好，以及您是否应该使用等效的基本R代码：

```
df[df[[input$var]] > input$min, ]
#>   x y input
#> 1 1 2      3
```

这是一个完全合法的立场，只要您知道`filter()`为您所做的工作，以便您可以生成等效的base-R代码。在这种情况下：

- 如果`df`只包含一列，您将需要`drop = FALSE`（否则，您将获得一个向量而不是数据框）。
- 您需要使用`which()`或类似的代码来删除任何缺失的值。
- 您不能进行分组过滤（例如`df %>% group_by(g) %>% filter(n() == 1)`）。

一般来说，如果您将dplyr用于非常简单的情况，您可能会发现不使用数据屏蔽的base-R函数更容易。然而，在我看来，tidyverse的优点之一是对边缘情况进行了仔细思考，以便函数工作得更一致。

12.3 Tidy-selection

除了数据屏蔽，整洁评估的另一个重要部分：整洁选择。整洁选择提供了一种按位置、名称或类型选择列的简明方法。它用于`dplyr::select()`和`dplyr::across()`以及来自tidyverse的许多函数，如`pivot_longer()`，`pivot_wider()`，`separate()`，`extract()`和`unite()`

12.3.1 Indirection

间接引用变量使用`any_of()`或`all_of()`：两者接纳的参数都是包含数据变量名称的字符向量构建的env变量。唯一的区别是，如果您提供数据框中不存在的变量名，会发生什么：`all_of()`将抛出错误，而`any_of()`将默默地忽略它。

例如，以下应用程序允许用户使用多重选择输入以及`all_of()`选择任意数量的变量：

```
ui <- fluidPage(
  selectInput("vars", "Variables", names(mtcars), multiple = TRUE),
  tableOutput("data")
)

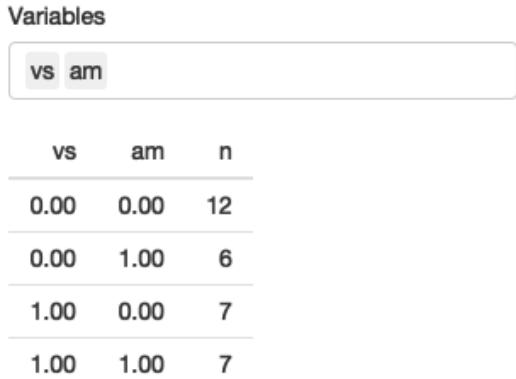
server <- function(input, output, session) {
  output$data <- renderTable({
    req(input$vars)
    mtcars %>% select(all_of(input$vars))
  })
}
```

12.3.2 Tidy-selection and data-masking

当您使用整洁选择的函数时，处理多个变量是微不足道的：您只需将包含变量名的字符向量（一种env变量）传递到`any_of()`或`all_of()`。如果我们也能在数据屏蔽函数中做到这一点，那不是很好吗？这就是在dplyr 1.0.0中添加的`across()`函数的想法。它允许您在数据屏蔽函数中使用整洁选择。

`across()`通常与一个或两个参数一起使用。第一个参数选择变量，在`group_by()`或`distinct()`等函数中非常有用。例如，以下应用程序允许您选择任意数量的变量并计算其唯一值。

```
ui <- fluidPage(  
  selectInput("vars", "Variables", names(mtcars), multiple = TRUE),  
  tableOutput("count")  
)  
  
server <- function(input, output, session) {  
  output$count <- renderTable({  
    req(input$vars)  
  
    mtcars %>%  
      group_by(across(all_of(input$vars))) %>%  
      summarise(n = n(), .groups = "drop")  
  })  
}
```



The screenshot shows a Shiny application interface. At the top, there is a header labeled "Variables". Below it is a "selectInput" field with two options: "vs" and "am". The "vs" option is currently selected, indicated by a grey background. Below the input field is a table output area. The table has three columns: "vs", "am", and "n". The data in the table is:

vs	am	n
0.00	0.00	12
0.00	1.00	6
1.00	0.00	7
1.00	1.00	7

图12.6：此应用程序允许您选择任意数量的变量并计算其唯一组合。在<https://hadley.shinyapps.io/ms-across>观看动态效果。

第二个参数是应用于每个选定列的函数（或函数列表）。这使得它非常适合`mutate()`和`summarise()`，您通常希望以某种方式转换每个变量。例如，以下代码允许用户选择任意数量的分组变量，以及任意数量的变量以汇总统计。

```

ui <- fluidPage(
  selectInput("vars_g", "Group by", names(mtcars), multiple = TRUE),
  selectInput("vars_s", "Summarise", names(mtcars), multiple = TRUE),
  tableOutput("data")
)

server <- function(input, output, session) {
  output$data <- renderTable({
    mtcars %>%
      group_by(across(all_of(input$vars_g))) %>%
      summarise(across(all_of(input$vars_s), mean), n = n())
  })
}

```

12.4 `parse()` + `eval()`

在我们出发之前，值得对 `paste() + parse() + eval()` 进行简短评论。如果您不知道这个组合是什么，您可以跳过本节，但如果您已经使用过它，我想转达一个小小的注意事项。

这是一种诱人的方式，因为它只需要学习很少的新想法。但它有一些主要的缺点：因为您将字符串粘贴在一起，因此很容易意外创建无效代码，或者代码可能会被滥用来做您不想做的事情。如果这是一个只有你使用的shiny应用程序，这并不重要，但这不是一个好习惯—否则很容易在你更广泛共享的应用程序中意外地创建一个安全漏洞。我们将在第[22](#)章中讲解这个想法。

（如果这是解决问题的唯一方法，你不应该感到难过，但当你有更多的心理空间时，我建议花一些时间弄清楚如何在不操纵字符串的情况下做到这一点。这将帮助你成为一个更好的R程序员。）

12.5 Summary

在本章中，您学习了如何创建shiny应用程序，让用户选择哪些变量将输入到 `dplyr::filter()` 和 `ggplot2::aes()` 等tidyverse函数中。这需要你了解一个你以前不需要考虑过的关键区别：数据变量和env变量之间的区别。一旦你掌握了这些函数的技巧，你就会释放向非R用户展示tidyverse的数据分析能力的能力。

这是这本书“Shiny in Action”部分的最后一章。现在您已经拥有了制作一系列有用应用程序所需的工具，我将专注于提高您对Shiny理论的理解。