Contents

Chapter 1

1.1 添加 UI 控件

- fluidPage() 是一个布局函数，用于设置页面的基本视觉结构。
- selectInput()是一个输入控件，允许用户通过提供值与应用程序交互。
- verbatimTextOutput() 和 tableOutput() 是输出控件，告诉 Shiny将渲染输出放在哪里。verbatimTextOutput()显示代码并 tableOutput()显示表格。

1.2

您可以通过包装一段代码并将 `reactive({...})` 其分配给变量来创建反应式表达式，并且可以通过像函数一样调用它来使用反应式表达式。但是，虽然看起来您正在调用函数，但响应式表达式有一个重要的区别：它仅在第一次调用时运行，然后缓存其结果，直到需要更新为止。

Chapter 2

基础 UI

2.1 输出

请注意，有两个渲染函数的行为略有不同：

- renderText() 将结果组合成一个字符串，并且通常与 textOutput()
- renderPrint() 打印结果，就像您在 R 控制台中一样，并且通常与 verbatimTextOutput()

2.1.1 表格

有两种用于在表中显示数据框的选项：

- tableOutput() 与 renderTable() 渲染一个静态数据表，一次性显示所有数据。
- dataTableOutput() 与 renderDataTable() 呈现一个动态表，显示固定数量的行以及用于更改哪些行可见的控件。

tableOutput() 对于小型、固定的 summary（例如模型系数）最有用；如果您想向用户公开完整的数据框，则 dataTableOutput() 最合适。

2.1.2 绘图

您可以使用 plotOutput() 和 renderPlot() 显示任何类型的 R 图形（base、ggplot2 或其他）。

2.1.3 下载

您可以让用户使用 downloadButton() 或 downloadLink() 来下载文件。

Chapter 3

响应式基础

3.1 server 函数

3.1.1 input

参数 input 是一个类似列表的对象，其中包含从浏览器发送的所有输入数据，根据输入 ID 命名。与普通的列表不同，input 对象是只读的。如果你尝试在服务函数内的修改输入，你将收到错误。发生此错误是因为 input 反映了浏览器中发生的情况，而浏览器是 Shiny 的“单一事实来源”。如果你可以修改 R 中的值，则可能会导致不一致，即输入滑块在浏览器中表示一件事，而 input$count 在 R 中表示不同的内容。这将使编程变得具有挑战性！稍后，在 Chapter 6 中，你将学习如何使用诸如 updateNumericInput() 修改浏览器中的值之类的功能，然后 input$count 进行相应的更新。

关于 input 更重要的一件事是：它对谁可以阅读是有选择性的。**要读取 input，必须处于由 renderText() 或 reactive() 函数创建的响应式上下文中。**

3.1.2 输出

output 与 input 非常相似：它也是一个根据输出 ID 命名的类似列表的对象。主要区别在于使用它来发送输出而不是接收输入。你总是要把 output 对象与 render 函数结合使用。

渲染函数做了两件事：

- 它设置了一个特殊的响应上下文，可以自动跟踪输出使用的输入。
- 它将 R 代码的输出转换为适合在网页上显示的 HTML。

与 input 一样，output 对如何使用它很挑剔。

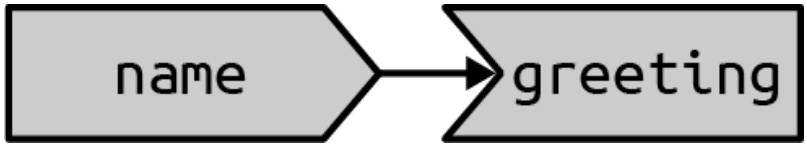

Figure 3.1: 响应图显示了输入和输出的连接方式

3.2 响应式编程

Shiny 的重要思想：你不需要告诉输出何时更新，因为 Shiny 会自动为你计算出来。

3.2.1 命令式编程 imperative programming 与声明式 declarative programming 编程

命令和 recipes 之间的区别是两种重要编程风格之间的主要区别之一：

- 在命令式编程中，你发出特定命令，它会立即执行。这是你在分析脚本中习惯的编程风格：命令 R 加载数据、转换数据、可视化数据，并将结果保存到磁盘。
- 在声明式编程中，你表达更高级别的目标或描述重要的约束，并依靠其他人来决定如何和/或何时将其转化为行动。这是你在 Shiny 中使用的编程风格。

命令式代码是 assertive；声明式代码是 passive-aggressive。

3.2.2 响应图

响应图是了解应用程序工作原理的强大工具。随着你的应用程序变得越来越复杂，制作响应图的快速高级草图通常很有用，以提醒你所有部分如何组合在一起。在本书中，我们将向你展示响应图，以帮助你理解示例的工作原理，稍后在 **??** 中，你将学习如何使用 reactlog 来为你绘制图表。

3.2.3 响应表达式

你将在响应图中看到一个更重要的组件：响应表达式。响应式表达式接受输入并产生输出，因此它们具有结合输入和输出特征的形状。希望这些形状能帮助你记住组件如何组合在一起。

Figure 3.2: 输入和表达式是响应式生产者；表达式和输出是响应式消费者

3.2.4 执行顺序

重要的是要理解代码运行的顺序完全由响应图决定。这与大多数 R 代码不同，大多数 R 代码的执行顺序由行的顺序决定。

3.3 响应表达式

响应式表达式具有输入和输出的风格：

- 与输入一样，你可以在输出中使用响应式表达式的结果。
- 与输出一样，响应式表达式依赖于输入并自动知道何时需要更新。

这种二元性意味着我们需要一些新的词汇：我将使用生产者（producer）来指代响应式输入和表达式，使用消费者（consumer）来指代响应式表达式和输出。

3.3.1 简化图形

你可能熟悉编程的“三规则”：每当你将某些内容复制并粘贴三次时，你应该弄清楚如何减少重复（通常通过编写函数）。这很重要，因为它减少了代码中的重复量，这使得代码更容易理解，并且随着需求的变化更容易更新。

然而，在 Shiny 中，我认为你应该考虑一规则：每当你复制并粘贴某些内容时，你应该考虑将重复的代码提取到响应式表达式中。该规则对于 Shiny 来说更为严格，因为响应式表达式不仅使人们更容易理解代码，还提高了 Shiny 有效重新运行代码的能力。

3.4 控制评估时间

3.4.1 定时失效

想象一下，你想通过不断地重新，以便你看到动画而不是静态图。我们可以通过一个新功能来提高更新频率：reactiveTimer()。reactiveTimer() 是一个响应式表达式，依赖于隐藏输

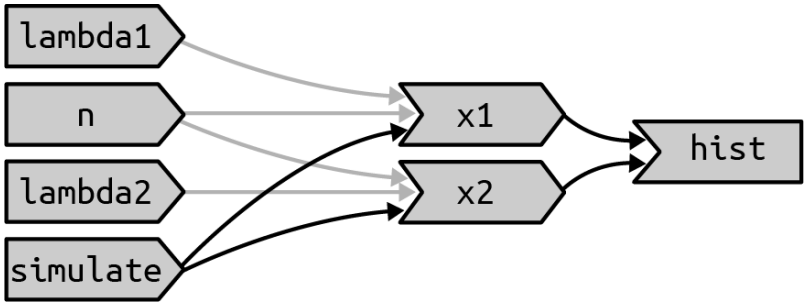

Figure 3.3: 根据需要，lambda1、lambda2、n 不再对 x1、 x2 具有响应性依赖：更改它们的值将不会触发计算。将箭头保留为非常浅的灰色只是为了提醒你 x1 和 x2 继续使用这些值，但不再对它们产生响应性依赖。

入：当前时间。当你希望响应式表达式比其他方式更频繁地使其自身无效时，可以使用 reactiveTimer()。

3.4.2 在点击时执行

假设在一个定时器中，每 0.5 秒执行一次代码，但是代码的执行时间需要 1 秒，那么 Shiny 需要做的事情越来越多。如果用户快速点击更改某个参数同样会直到 Shiny 需要执行的事情越来越多，尤其是在涉及昂贵计算时。

如果你的应用程序中出现这种情况，你可能希望要求用户通过单击按钮来选择执行昂贵的计算。这是 actionButton() 一个很好的用例。

我们需要一个新工具 eventReactive()：一种使用输入值而不对其产生响应性依赖的方法，它有两个参数：第一个参数指定要依赖的内容，第二个参数指定要计算的内容。

Figure 3.3 体现了这种思想。

3.5 观察员

有些操作不会在页面中展示，但是需要记录，比如说调式消息、发送数据等等，这应该使用输入和输出的 render，而是需要使用观察员 observers。

observeEvent() 它为你提供了一个重要的调试工具。

observeEvent() 与 eventReactive() 非常相似。它有两个重要的参数：eventExpr 和 handlerExpr。第一个参数是要依赖的输入或表达式；第二个参数是将运行的代码。

observeEvent() 和 eventReactive() 之间有两个重要的区别：

- 你没有将 observeEvent() 的结果分配给变量

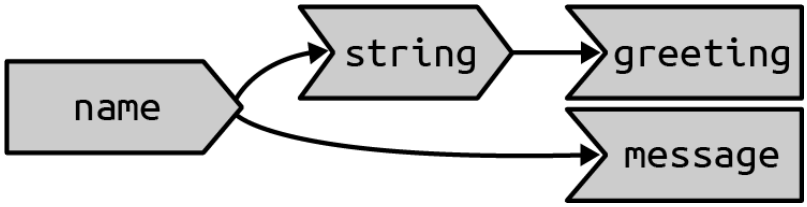

Figure 3.4

- 你无法从其他响应性消费者那里引用它

观察员和产出密切相关。您可以将输出视为具有特殊的副作用：更新用户浏览器中的HTML。为了强调这种接近性，我们将在响应图中以相同的方式绘制它们。这会产生 Figure 3.4 所示的响应图。

Part I

Shiny in Action

Chapter 4

布局、主题、HTML

4.1 单页布局

布局函数提供应用程序的高级视觉结构。布局是由函数调用的层次结构创建的，其中 R 中的层次结构与生成的 HTML 中的层次结构相匹配。这有助于你理解布局代码。

4.1.1 页面功能

最重要但最无趣的布局函数是 fluidPage()，它看起来是一个非常无聊的应用程序，但幕后有很多工作，因为 fluidPage() 设置了 Shiny 所需的所有 HTML、CSS 和 JavaScript。

除了之外 fluidPage()，Shiny 还提供了一些其他页面函数，可以在更特殊的情况下派上用场：fixedPage() 和 fillPage()。 fixedPage() 工作原理类似 fluidPage()，但有一个固定的最大宽度，这可以防止你的应用程序在更大的屏幕上变得不合理的宽度。 fillPage() 填充浏览器的整个高度，如果你想制作占据整个屏幕的绘图，则非常有用。你可以在他们的文档中找到详细信息。

4.1.2 带侧边栏的页面

要制作更复杂的布局，你需要在 fluidPage(). 例如，要制作一个左侧输入、右侧输出的两列布局，你可以使用 sidebarLayout()（以及它的朋友 titlePanel()、sidebarPanel() 和 mainPanel()）。

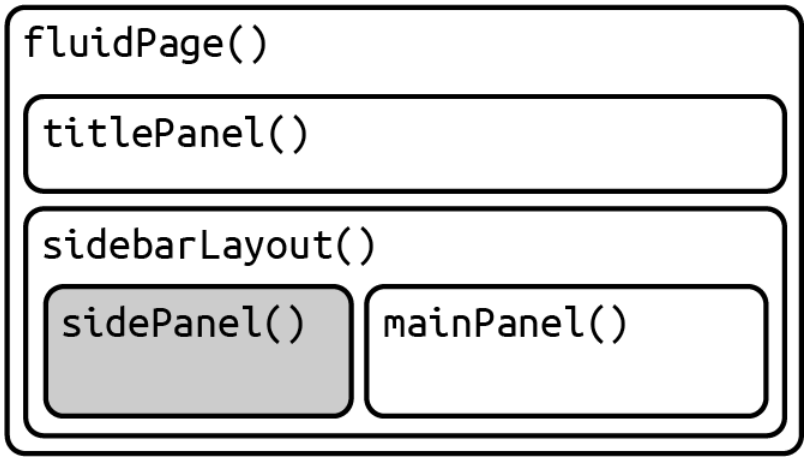

Figure 4.1: 带有侧边栏的基本应用程序的结构

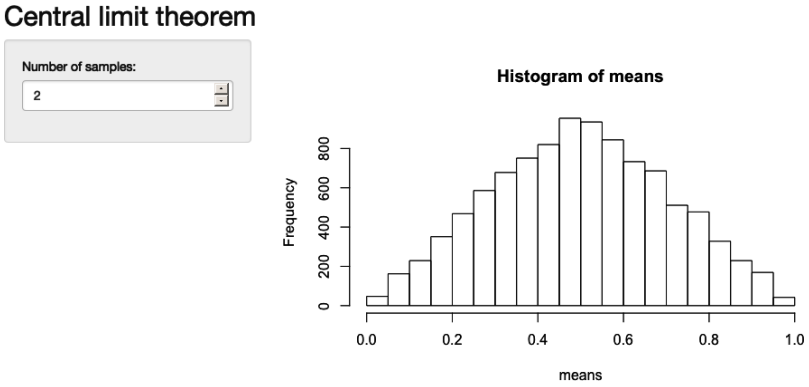

Figure 4.2: 常见的应用程序设计是将控件放在侧边栏中并在主面板中显示结果

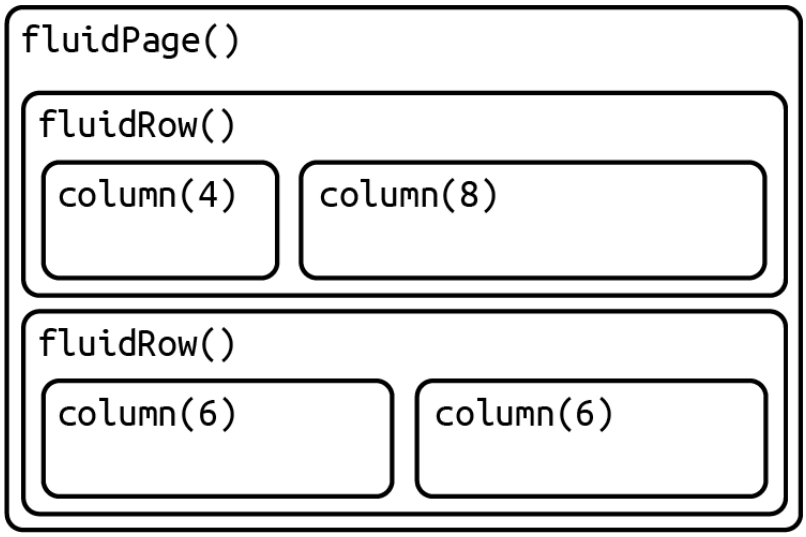

Figure 4.3: 简单多行应用程序的底层结构

4.1.3 多行

在底层，sidebarLayout() 它构建在灵活的多行布局之上，你可以直接使用它来创建视觉上更复杂的应用程序。像往常一样，你从 fluidPage() 开始。然后，你可以使用 fluidRow() 来创建行，并使用 column() 来创建列。

每行由 12 列组成，column() 第一个参数给出要占用的列数。12 列布局为你提供了极大的灵活性，因为你可以轻松创建 2 列、3 列或 4 列布局，或使用窄列来创建间隔。

4.2 多页面布局

随着你的应用程序变得越来越复杂，可能无法将所有内容都放在一个页面上。在本节中，你将学习 tabPanel() 创建多个页面错觉的各种用法。这是一种错觉，因为你仍然拥有一个带有单个底层 HTML 文件的应用程序，但它现在被分成了几部分，并且一次只能看到一个部分。

多页面应用程序与模块配合得特别好，你将在 **??** 中了解这些模块。模块允许你以与划分用户界面相同的方式划分服务器功能，创建仅通过明确定义的连接进行交互的独立组件。

4.2.1 选项卡集

将页面分成多个部分的简单方法是使用 tabsetPanel() 以及与它近似的 tabPanel()。如果你想知道用户选择了哪个选项卡，你可以提供参数 id，它将成为 tabsetPanel() 的输入。

注意：tabsetPanel()可以在应用程序中的任何位置使用；如果需要的话，将选项卡集嵌套在其他组件（包括选项卡集!）中是完全可以的。

4.2.2 导航列表和导航栏

由于选项卡是水平显示的，因此可以使用的选项卡数量存在根本限制，特别是当它们具有长标题时。 navbarPage() 与 navbarMenu() 提供两种替代布局，让你可以使用更多带有更长标题的选项卡。

navlistPanel() 类似于 tabsetPanel()，但它不是水平运行选项卡标题，而是在侧边栏中垂直显示它们。它还允许你添加带有纯字符串的标题。另一种方法是使用 navbarPage()：它仍然水平运行选项卡标题，但你可以使用 navbarMenu() 添加下拉菜单以获得额外的层次结构级别。

4.3 Bootstrap

要继续您的应用程序定制之旅，您需要更多地了解 Shiny 使用的Bootstrap框架。Bootstrap 是 HTML 约定、CSS 样式和 JS 片段的集合，它们捆绑成一种方便的形式。Bootstrap 源自最初为 Twitter 开发的框架，在过去 10 年中已发展成为网络上最流行的 CSS 框架之一。

作为一个 Shiny 开发者，你不需要对 Bootstrap 考虑太多，因为 Shiny 函数会自动为你生成 bootstrap 兼容的 HTML。但很高兴知道 Bootstrap 的存在，因为这样：

- 您可以使用 bslib::bs_theme() 自定义代码的视觉外观
- 您可以使用class参数通过 Bootstrap 类名来自定义一些布局、输入和输出
- 您可以创建自己的函数来生成Shiny 未提供的Bootstrap 组件

也可以使用完全不同的 CSS 框架。许多现有的 R 包通过包装 Bootstrap 的流行替代品使这一切变得容易：

- shiny.semantic，由 Appsilon 开发，建立在 formantic UI 之上。
- shinyMobile 由 RInterface 开发，构建于框架 7之上，专为移动应用程序而设计。
- shinymaterial 由 Eric Anderson开发，建立在 Google 的 Material 设计框架之上。
- shinydashboard 也由 RStudio 提供，提供了一个旨在创建仪表板的布局系统。

您可以在 awesome-shiny-extensions 找到更完整且积极维护的列表。

4.4 主题

Bootstrap 在 R 社区中如此普遍，以至于很容易产生风格疲劳：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 包进行主题化。bslib 是相对较新

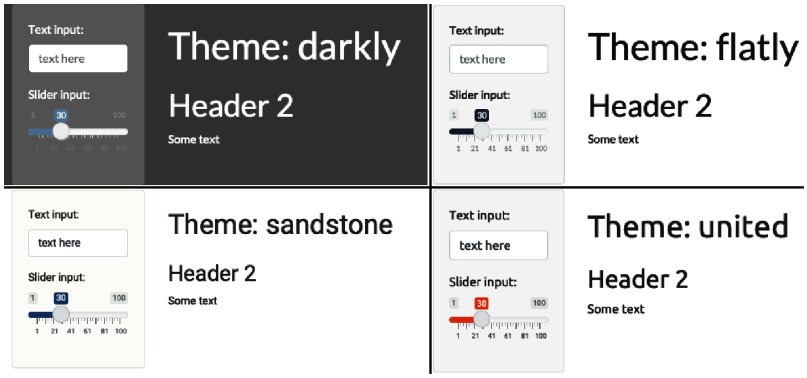

Figure 4.4: 同一个应用程序具有四个 bootswatch 主题：darkly、flatly、sandstone 和 United

的软件包，它允许您覆盖许多 Bootstrap 默认值，以创建独特的外观。

4.4.1 shiny 主题

更改应用程序整体外观的最简单方法是使用的参数选择预制的“bootswatch”主题。

预览和自定义主题的一种简单方法是使用 `bslib::bs_theme_preview(theme)`。这将打开一个闪亮的应用程序，显示应用许多标准控件时主题的外观，并为您提供用于自定义最重要参数的交互式控件。

4.4.2 plot 主题

如果您对应用程序的样式进行了大量自定义，您可能还需要自定义绘图以匹配。幸运的是，这真的很容易，这要归功于 thematic 包，它自动为 ggplot2、lattice和 base 提供主题。只需在您的服务器调用 `thematic_shiny()` 功能即可。这将自动确定应用程序主题的所有设置。

Chapter 5

图形

5.1 交互性

plotOutput() 最酷的事情之一是，它不仅可以作为显示绘图的输出，还可以作为响应指针事件的输入。这允许你创建交互式图形，用户可以直接与绘图上的数据进行交互。交互式图形是一种强大的工具，具有广泛的应用范围。

5.1.1 基础知识

绘图可以响应四种不同的鼠标23事件：click、dblclick（双击）、hover（当鼠标在同一位置停留一小会儿时）和brush（矩形选择工具）。要将这些事件转换为闪亮的输入，你可以向相应的参数提供一个字符串 plotOutput()，例如 `plotOutput("plot", click = "plot_click")`。这将创建一个 `input$plot_click` 可用于处理绘图上的鼠标单击的。

5.1.2 点击

点事件返回一个相对丰富的列表，其中包含大量信息。最重要的组成部分是 x 和 y，它们给出了数据坐标中事件的位置。但我不会谈论这种数据结构，因为你只在相对罕见的情况下需要。相反，你将使用 nearPoints() 帮助程序，它返回一个数据框，其中包含与点击接近的行（一个观测值），处理一堆繁琐的细节。

这里我们给出 nearPoints() 四个参数：绘图下的数据框、输入事件以及轴上变量的名称。如果你使用 ggplot2，则只需提供前两个参数，因为 xvar 和 yvar 可以从绘图数据结构自动估算。

你可能想知道到底 nearPoints() 返回了什么。这是一个使用的好地方 browser()。

另一种使用方法 nearPoints() 是 `allRows = TRUE` 与 `addDist = TRUE` 和一起使用。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）。
- `selected_` 表示它是否靠近单击事件（即，是否是在 `allRows = FALSE` 时返回的行）。

5.1.3 其他点事件

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。clickOpts() 如果需要，你可以通过提供 dblclickOpts() 或 hoverOpts() 来代替给出输入 ID 的字符串，从而获得对事件的额外控制。

你可以在一张图上使用多种交互类型。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。

5.1.4 画笔

在绘图上选择点的另一种方法是使用画笔(brushing)，即由四个边定义的矩形选择。click在Shiny中，一旦你掌握了 click 和 nearPoints() 画笔的使用就很简单：你只需切换到brush 参数和 brushedPoints() 助手即可。

brushOpts() 用于控制颜色 (fill 和 stroke)，或 direction = “x” 或 “y” 将 brush 限制为单一维度（例如，对于刷牙时间序列很有用）。

5.1.5 修改绘图

当你在与之交互的同一个绘图中显示变化时，交互性的真正魅力就显现出来了。

reactiveVal() 与 reactive() 类似。你可以通过调用其初始值来创建响应式值 reactiveVal()，并以与响应式相同的方式检索该值。最大的区别在于，你还可以更新响应性值，并且引用它的所有响应性使用者都将重新计算。响应式值使用特殊的语法进行更新 - 你可以像函数一样调用它，第一个参数是新值。

5.1.6 交互限制

在我们继续之前，了解交互式图中的基本数据流以了解它们的局限性非常重要。基本流程是这样的：

1. JavaScript 捕获鼠标事件。
2. Shiny 将鼠标事件数据发送回 R，告诉应用程序输入现在已过时。

3. 所有下游响应性消费者都被重新计算。
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，瓶颈往往是绘制绘图所需的时间。根据情 plot 的复杂程度，这可能需要几分之一秒的时间。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 传输回浏览器所需的时间。

5.2 动态高度和宽度

本章的其余部分不如交互式图形那么令人兴奋，但包含一些需要在某个地方介绍的重要材料。

首先，可以使绘图大小具有响应性，因此宽度和高度会根据用户操作而变化。为此，请为零参数函数 renderPlot() 的参数 width 和 height 提供值 - 现在必须在服务器而不是 UI 中定义这些函数，因为它们可以更改。这些函数应该没有参数并返回所需的像素大小。它们在响应性环境中进行评估，以便您可以动态调整绘图的大小。

5.3 图像

如果您想显示现有图像（而不是绘图），则可以使用 renderImage()。

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

- contentType，定义图像的 MIME 类型。如果未提供，Shiny 会根据文件扩展名进行猜测，因此仅当您的图像没有扩展名时才需要提供此文件
- 图像的 width 和 height（如果已知）
- 任何其他参数（例如class或 alt）将作为属性添加到 HTML `<img>` 中的标记中

您还必须提供 deleteFile 值。不幸的是，renderImage() 最初设计用于处理临时文件，因此它在渲染图像后会自动删除图像。这显然是非常危险的，因此在 Shiny 1.5.0 中行为发生了变化。现在，shiny 不再删除图像，而是强制您明确选择您想要的行为。

Chapter 6

向用户反馈

你通常可以通过让用户更深入地了解正在发生的事情来提高应用程序的可用性。当输入没有意义时，这可能会采取更好的消息形式，或者在需要很长时间的操作时采取进度条。

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。然后，我们将继续进行**通知**，向用户发送一般消息，以及**进度条**，进度条提供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**危险**的操作，以及如何通过确认对话框或撤消操作的能力让用户安心。

6.1 验证

你可以向用户提供的第一个也是最重要的反馈是他们给了你错误的输入。

6.1.1 验证输入

向用户提供额外反馈的一个好方法是通过 shinyFeedback 包。使用它是一个两步过程。首先，添加 useShinyFeedback() 到ui. 这将设置所需的 HTML 和 JavaScript，以实现有吸引力的错误消息显示。然后在你的 server() 函数中，你调用反馈函数之一：feedback()、feedbackWarning()、feedbackDanger()和 feedbackSuccess()。他们都有三个关键参数：

1. inputId，应放置反馈的输入的 id。
2. show，逻辑决定是否显示反馈。
3. text，要显示的文本。

它们还具有 color 和 icon 可用于进一步自定义外观的参数。

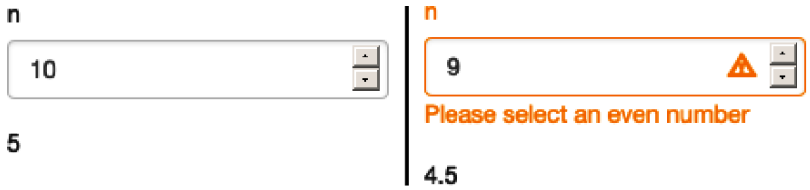

Figure 6.1: 用于 feedbackWarning() 显示无效输入的警告。左侧的应用程序显示有效的输入，右侧的应用程序显示无效（奇数）输入并带有警告消息。

请注意，虽然显示了错误消息，但输出仍会更新。通常你不希望出现这种情况，因为无效的输入可能会导致你不希望向用户显示的无信息 R 错误。要阻止输入触发响应性更改，你需要一个新工具：req()，“required”的缩写。当 req() 的输入不正确时，它会发送一个特殊信号来告诉 Shiny 响应式没有所需的所有输入，因此应该“暂停”。

6.1.2 用 req() 取消执行

It’s easiest to understand req() by starting outside of validation. You may have noticed that when you start an app, the complete reactive graph is computed even before the user does anything. This works well when you can choose meaningful default values for your inputs. But that’s not always possible, and sometimes you want to wait until the user actually does something. This tends to crop up with three controls:

- 在 textInput() 中，你已经使用过 `value = ""` 并且在用户键入某些内容之前不想执行任何操作。
- 在 selectInput() 中，你提供了一个空选择 `""` 并且在用户做出选择之前你不想执行任何操作。
- 在 fileInput() 中，在用户上传任何内容之前，其结果为空。

我们需要某种方法来“暂停”响应，以便在某些条件成立之前不会发生任何事情。这就是 req() 的工作，它在允许响应式生产者继续之前检查所需的值。

req() 通过发出特殊条件信号来工作[1]。这种特殊情况会导致所有下游响应和输出停止执行。从技术上讲，它使任何下游响应性消费者处于无效状态。

req() 被设计为仅当用户提供了值时才会继续，而不管输入控件 `req(input$x)` 的类型。如果需要，你还可以与自己的逻辑语句一起使用。

[1] https://adv-r.hadley.nz/conditions.html

6.1.3 req() 及验证

6.1.4 验证输出

当问题与单个输入相关时，shinyFeedback 非常有用。但有时无效状态是输入组合的结果。在这种情况下，将错误放在输入旁边（你会将其放在哪个输入旁边？）并没有真正意义，而是将其放在输出中更有意义。

你可以使用 shiny 内置的工具 validate() 来做到这一点。当在响应或输出内部调用时，validate(message) 停止执行其余代码，而是在任何下游输出中显示 message。

6.2 通知

如果没有问题而你只是想让用户知道发生了什么，那么你需要一个通知。在 Shiny 中，通知是通过 showNotification() 和堆叠在页面右下角创建的。共有三种基本使用方法 showNotification()：

1. 显示在固定时间后自动消失的瞬时通知。
2. 在进程启动时显示通知并在进程结束时将其删除。
3. 使用渐进式更新来更新单个通知。

6.2.1 瞬时通知

showNotification() 最简单的使用方法是使用单个参数调用它：你想要显示的消息。默认情况下，该消息将在 5 秒后消失，你可以通过设置覆盖该消息 duration，或者用户可以通过单击关闭按钮提前将其关闭。如果你想让通知更加突出，你可以将 type 参数设置为“message”、“warning”或“error”之一。

6.2.2 完成后移除

将通知的存在与长时间运行的任务联系起来通常很有用。在这种情况下，你希望在任务开始时显示通知，并在任务完成时删除通知。为此，你需要：

- 设置duration = NULL，closeButton = FALSE使通知保持可见，直到任务完成。
- 存储 showNotification() 返回的 id 值，然后将该值传递给 removeNotification()。最可靠的方法是使用on.exit()，这可以确保无论任务如何完成（成功或有错误），通知都会被删除。你可以在更改和恢复状态中了解更多 on.exit() 信息。

一般来说，这些类型的通知将以响应方式存在，因为这可以确保长时间运行的计算仅在需要时重新运行。

6.2.3 渐进式更新

多次调用 showNotification() 通常会创建多个通知。你可以通过捕获第一个调用的 id 并在后续调用中使用它来更新单个通知。如果你的长时间运行的任务有多个子组件，这非常有用。

6.3 进度条

对于长时间运行的任务，最好的反馈类型是进度条。除了告诉你在此过程中所处的位置外，它还可以帮助你估计需要多长时间。不幸的是，这两种技术都存在相同的主要缺点：要使用进度条，你需要能够将大任务划分为已知数量的小块，每个小块花费大致相同的时间。这通常很困难，特别是因为底层代码通常是用 C 编写的，并且无法向你传达进度更新。

6.3.1 shiny

要使用 Shiny 创建进度条，你需要使用 withProgress() 和 incProgress()。你首先将其包装在 withProgress(). 这会在代码启动时显示进度条，并在完成后自动将其删除。然后在每一步之后调用 incProgress()，第一个参数是进度条增量的量。默认情况下，进度条从 0 开始，到 1 结束，因此增量除以 1 除以步数将确保进度条在循环结束时完成。

6.3.2 waiter

内置进度条对于基础知识来说非常有用，但如果你想要提供更多视觉选项的东西，你可以尝试 waiter 包。waiter 包使用 R6 对象将所有与进度相关的功能捆绑到一个对象中。

默认显示是页面顶部的细进度条，但有多种方法可以自定义输出：

- 你可以用使用以下之一覆盖默认值 theme：
  - overlay：隐藏整个页面的不透明进度条
  - overlay-opacity：覆盖整个页面的半透明进度条
  - overlay-percent：不透明的进度条，还显示数字百分比。
- 你可以通过设置 selector 参数将其覆盖在现有输入或输出上，而不是显示整个页面的进度条

6.3.3 旋转器

有时你并不确切知道操作需要多长时间，而你只想显示一个动画旋转器，让用户放心某些事情正在发生。

一个更简单的替代方案是使用Dean Attali 的 shinycssloaders 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，你只需使用 shinycssloaders::withSpinner() 包装你想要在无效时自动获取微调器的输出即可。

6.4 确认和撤销

有时某个操作可能存在危险，你要么想确保用户确实想要执行该操作，要么想让他们能够在为时已晚之前退出。

6.4.1 显式确认

保护用户免遭意外执行危险操作的最简单方法是要求明确的确认。最简单的方法是使用一个对话框，强制用户从一小组操作中进行选择。在 Shiny 中，你可以使用 modalDialog()。这被称为“模式”对话框，因为它创建了一种新的交互“模式”；在处理完该对话框之前，你无法与主应用程序交互。

创建对话框时需要考虑一些小但重要的细节：

- 这些按钮应该怎么称呼？最好是描述性的，因此避免使用是/否或继续/取消，以重述关键动词。
- 您应该如何订购按钮？您是先取消（如 Mac），还是先继续（如 Windows）？您最好的选择是镜像您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，我习惯于 `class = "btn btn-danger"` 突出显示按钮的样式。

6.4.2 撤消操作

显式确认对于不经常执行的破坏性操作最有用。如果你想减少频繁操作所造成的错误，你应该避免它。在这种情况下，更好的方法是在实际执行操作之前等待几秒钟，让用户有机会注意到任何问题并撤消它们。

6.4.3 垃圾

对于几天后您可能会后悔的操作，更复杂的模式是在计算机上实现垃圾箱或回收站等功能。当您删除文件时，它不会被永久删除，而是会被移至保留单元，这需要单独的操作才能清空。这就像类固醇的“撤消”选项；你有很多时间为你的行为后悔。这也有点像确认；您必须执行两个单独的操作才能使删除永久化。