

Contents

1	1
1.1 添加 UI 控件	1
1.2	1
2 基础 UI	3
2.1 输出	3
2.1.1 表格	3
2.1.2 绘图	3
2.1.3 下载	3
3 响应式基础	5
3.1 server 函数	5
3.1.1 input	5
3.1.2 输出	5
3.2 响应式编程	6
3.2.1 命令式编程 imperative programming 与声明式 declarative programming 编程	6
3.2.2 响应图	6
3.2.3 响应表达式	6
3.2.4 执行顺序	7
3.3 响应表达式	7
3.3.1 简化图形	7
3.4 控制评估时间	7
3.4.1 定时失效	7
3.4.2 在点击时执行	8
3.5 观察员	8

I	Shiny in Action	11
4	布局、主题、HTML	13
4.1	单页布局	13
4.1.1	页面功能	13
4.1.2	带侧边栏的页面	13
4.1.3	多行	15
4.2	多页面布局	15
4.2.1	选项卡集	15
4.2.2	导航列表和导航栏	16
4.3	Bootstrap	16
4.4	主题	16
4.4.1	shiny 主题	17
4.4.2	plot 主题	17
5	图形	19
5.1	交互性	19
5.1.1	基础知识	19
5.1.2	点击	19
5.1.3	其他点事件	20
5.1.4	画笔	20
5.1.5	修改绘图	20
5.1.6	交互限制	20
5.2	动态高度和宽度	21
5.3	图像	21

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$count` 在 R 中表示不同的内容。这将使编程变得具有挑战性！稍后，在 ?? 中，你将学习如何使用诸如 `updateNumericInput()` 修改浏览器中的值之类的功能，然后 `input$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。

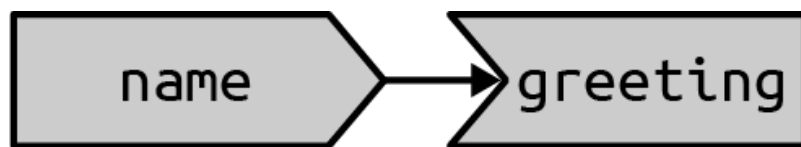


Figure 3.1: 响应图显示了输入和输出的连接方式

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

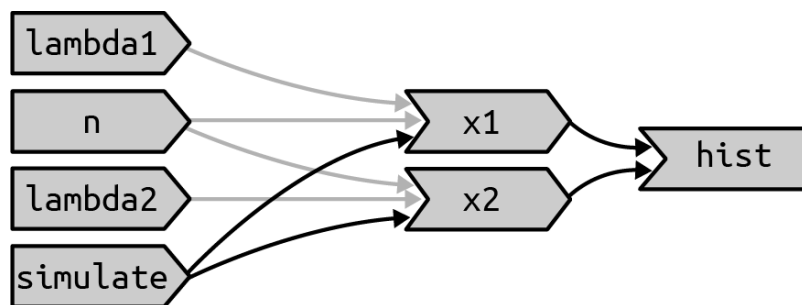


Figure 3.3: 根据需要，`lambda1`、`lambda2`、`n` 不再对 `x1`、`x2` 具有响应性依赖：更改它们的值将不会触发计算。将箭头保留为非常浅的灰色只是为了提醒你 `x1` 和 `x2` 继续使用这些值，但不再对它们产生响应性依赖。

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，比如说调试消息、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

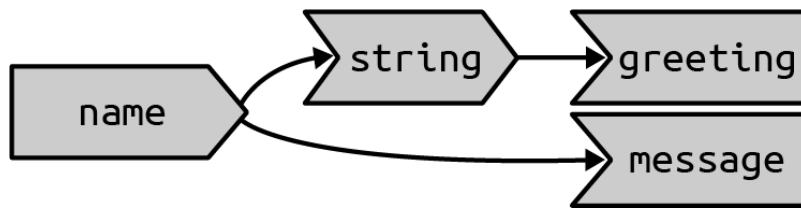


Figure 3.4

- 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()`）。

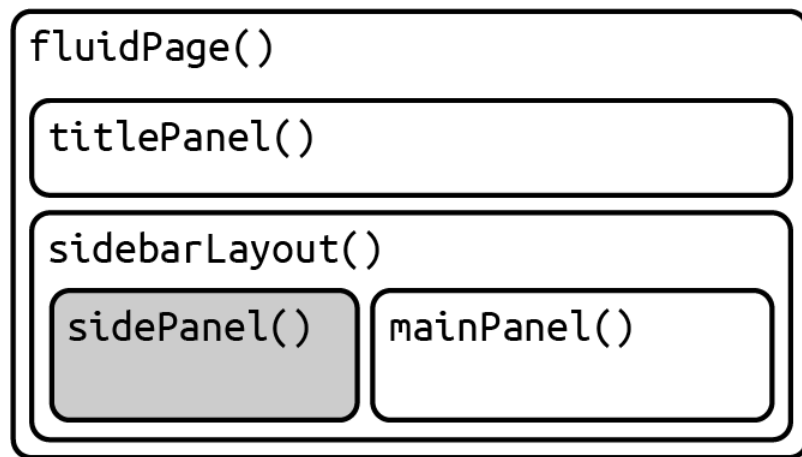


Figure 4.1: 带有侧边栏的基本应用程序的结构

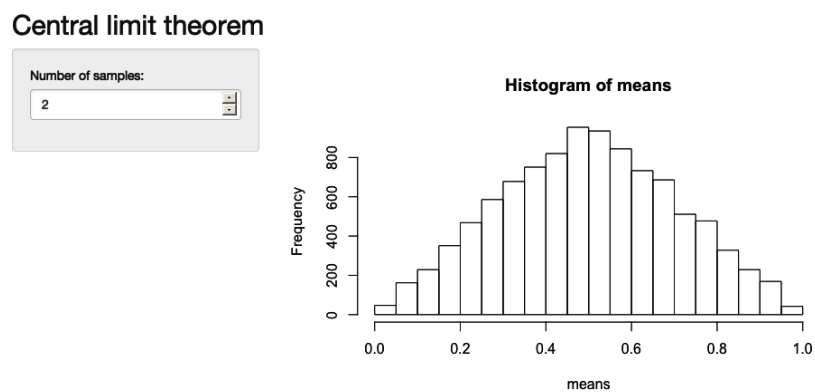


Figure 4.2: 常见的应用程序设计是将控件放在侧边栏中并在主面板中显示结果

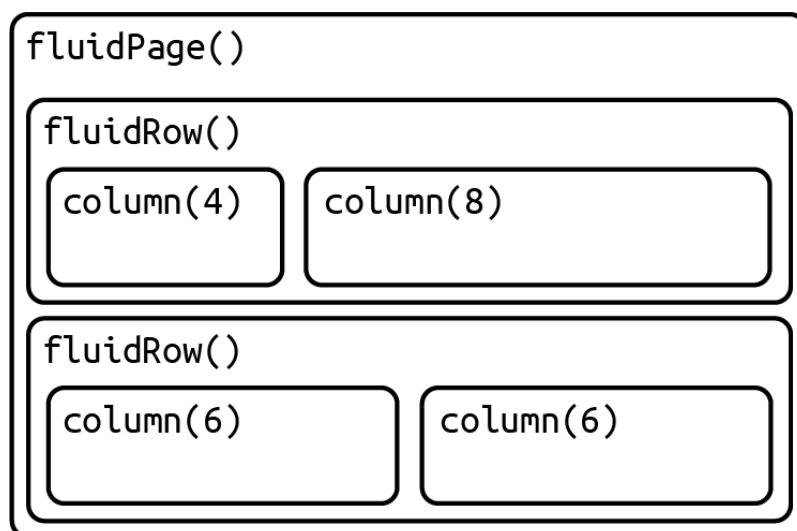


Figure 4.3: 简单多行应用程序的底层结构

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()` 以及与其近似的 `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](https://www.bslib.org/) 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 `` 中的标记中

您还必须提供 `deleteFile` 论点。不幸的 `renderImage()` 是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。