htmlwidgets 中文教程

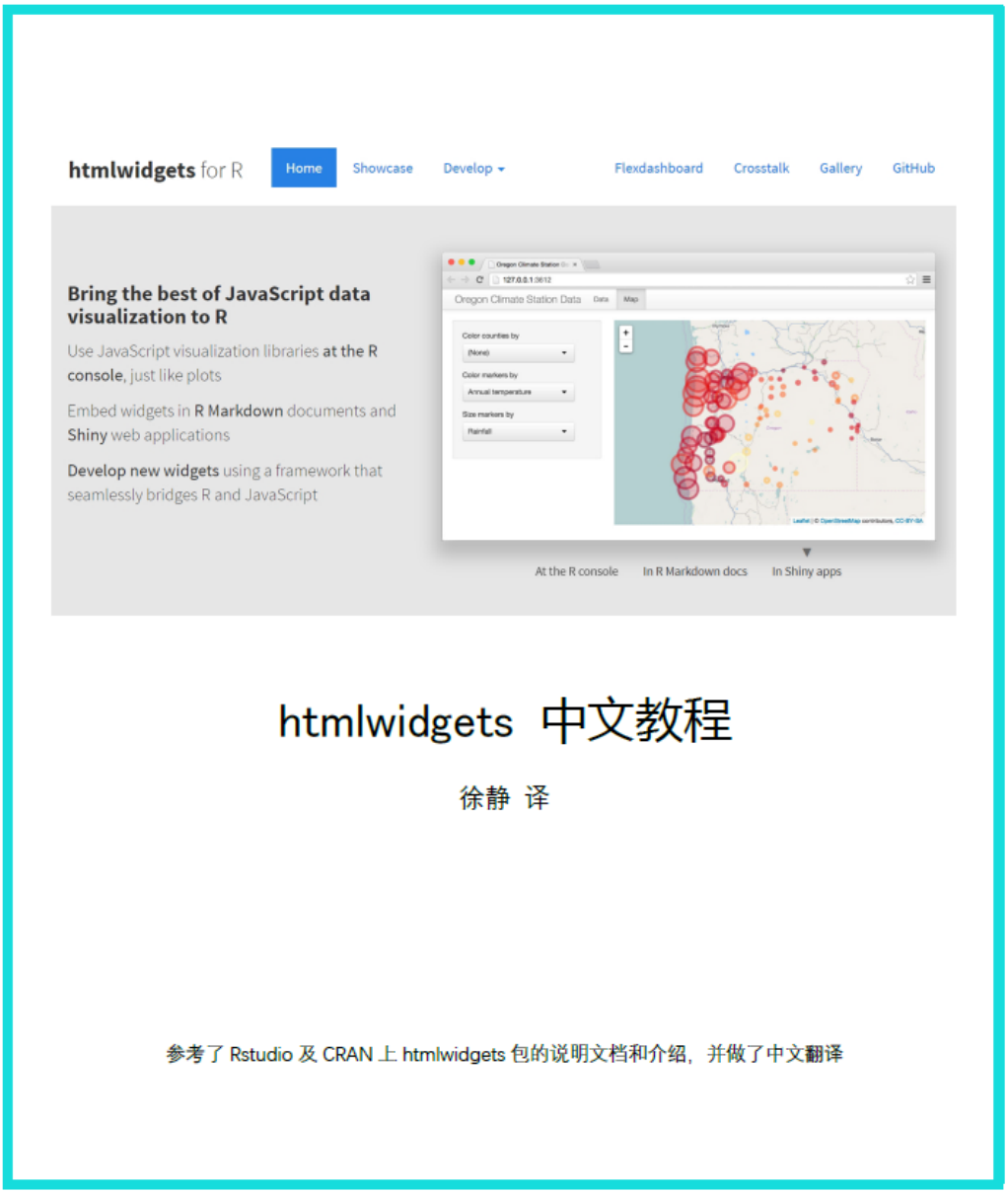
徐静译

2018-07-05

Contents

声明

htmlwidgets是 R 语言中非常有划时代意义的包，因为有了 htmlwidgets 使得 R 语言在交互可视化和基于 JavaScript 的编程有了实质性的进步。htmlwidgets 目前没有中文说明文档和教程，该文档是对官方文档的详细翻译，译者水平有限，读者可以在https://github.com/DataXujing/htmlwidgets_CN/issues中留言指正。



序言

htmlwidgets，一个用来创建 HTML 控件的包，可以运行在 R 命令行, R Markdown, Shiny。

```
Type Package
Title HTML Widgets for R
Version 1.2
Description A framework for creating HTML widgets that render in various
contexts including the R console, 'R Markdown' documents, and 'Shiny'
web applications.
License MIT + file LICENSE
VignetteBuilder knitr
Imports grDevices, htmltools (>= 0.3), jsonlite (>= 0.9.16), yaml
Suggests knitr (>= 1.8)
Enhances shiny (>= 1.0.5)
URL https://github.com/ramnathv/htmlwidgets
BugReports https://github.com/ramnathv/htmlwidgets/issues
RoxygenNote 6.0.1
NeedsCompilation no
Author Ramnath Vaidyanathan [aut, cph],Yihui Xie [aut],JJ Allaire [aut, cre],Joe Cheng [aut],
Kenton Russell [aut, cph],RStudio [cph]
Maintainer JJ Allaire <jj@rstudio.com>
Repository CRAN
Date/Publication 2018-04-19 12:43:03 UTC
```

目前针对于 htmlwidgets 并没有详细的中文教程, 译者会持续更新翻译 htmlwidgets，并推出更多基于 htmlwidgets 的 R 包。

徐静

联信商务咨询有限公司

2017-06-02

关于译者

徐静：

硕士研究生, 目前的研究兴趣主要包括：数理统计，统计机器学习，深度学习，网络爬虫，前端可视化，R 语言和 Python 语言的超级粉丝，多个 R 包和 Python 模块的作者，现在正逐步向 Java 迁移。

Graduate students,the current research interests include: mathematical statistics, statistical machine learning, deep learning, web crawler, front-end visualization. He is a super fan of R and Python, and the author of several R packages and Python modules, and now gradually migrating to Java.

Chapter 1

简介

1.1 概况

htmlwidgets包提供了一个 R 语言链接 Javascript 库的框架,HTML 控件能够：

- 在 R 命令中做分析比如方便的 R 作图
- 和 R Markdown 结合在一起
- 和 shiny 结合在一起
- 保存为独立的网页，通过电子邮件，Dropbox 等 ad-oc 共享。

通过遵循一小部分易于遵循的约定，可以创建非常小的代码和 HTML 控件，所有控件包含如下部分：

1. *Dependencies*: 这些是控件用到的需要声明的 Javascript 和 CSS
2. *R binding*: 这是终端用户将调用的功能，以向控件提供输入数据，并制定控件应该如何呈现各种选项，这包括在 shiny 应用程序中使用控件所需要的一些简短的样板功能。
3. *javaScript binding*: 这是 JavaScript 代码，把所有的东西粘在一起。将 R 绑定中收集的数据和选项传递给底层的 JavaScript 库

已经有非常多的包基于 htmlwidgets 去完成, 包括：

- leaflet – 交互的地图绘制包
- dygraphs – 交互时间序列绘图包
- networkD3 – 基于 D3.js 的交互网络图可视化
- sparkline – 小型的内联图
- DT – 表格可视化
- rthreejs – 交互 3D 图

包的作者包括：Ramnath Vaidyanathan, Joe Cheng, JJ Allaire, Yihui Xie, and Kenton Russell 等。

HTML 控件一般会寄存在一个 R 包中，并且应该包含他们的依赖关系的所有源代码，例如这里译者写的以个基于 htmlwidgets 的 R 包:XuJIngd3plus。这是为了确保依赖的控件的完全可重复的 (既不需要联网，也不需要运行服务器)，说白了在你 R 包中，应该包含所有的源码包括你底层调用的 JavaScript 包或 CSS。

1.2 简单的开始

如果你懂 R 语言和一点 JavaScript, 创建自己的小控件非常简单，最先要做的就是要安装 htmlwidgets, 在 CRAN 上：

```
install.packages('htmlwidgets')
```

你也可以在 GitHub 上安装开发版本：

```
devtools::install_github('ramnathv/htmlwidgets')
```

通过包中自带的说明文档，让你快速的熟悉 htmlwidgets 并进入开发者状态, 包括：

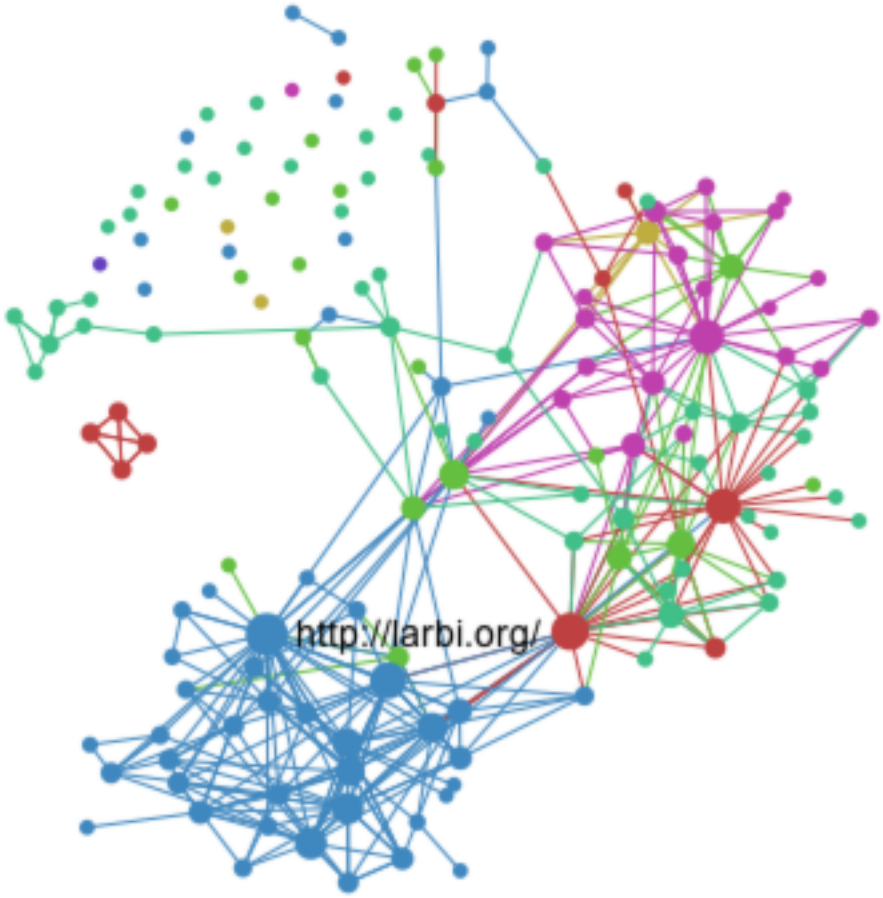
- Introduction to HTML Widgets
- HTML Widget Sizing
- HTML Widgets: Advanced Topics

我们会持续把他们翻译成中文, 让中国人看起来更爽。

1.3 例子 (sigma.js)

首先，我们将通过创建一个控件来封装sigma.js图形可视化库。当我们完成后，我们可以用来显示GEXF(Graph Exchange XML Format) 数据文件的交互可视化，例如：

```
library(sigma)
data <- system.file("examples/ediaspora.gexf.xml", package = "sigma")
sigma(data)
```



注意上面的输出仅仅是一个静态图像，你可以按照下文的 Demo 做一个交互的版本。

创建这种绑定所需的代码非常少。下面我们将一步一步地介绍所有的控件。然后，我们将描述如何创建自己的控件（包括为所有核心组件自动生成基本的脚手架）。

1.3.1 文件布局

假设我们的控件被命名为 **sigma**, 并且位于同名的 R 包中。我们的 JavaScript 绑定源代码文件名为 sigma.js。由于我们的控件将读取 GEXF 数据文件，我们还需要包括基础 sigma.min.js 库以及 GEXF 插件，下面是我们将添加到包中的文件：

```
R/
| sigma.R

inst/
|-- htmlwidgets/
|   |-- sigma.js
|   |-- sigma.yaml
|   |-- lib/
|   |   |-- sigma-1.0.3/
|   |   |   |-- sigma.min.js
|   |   |   |-- plugins/
|   |   |   |   |-- sigma.parsers.gexf.min.js
```

请注意，JavaScript,YAML 和其他依赖项都包含在 inst/htmlwidgets 目录中 (随后将被安装到一个名为 htmlwidgets 的包的子目录中)。

1.3.2 依赖关系

依赖项是控件使用的 JavaScript 和 CSS 资源。依赖项包含在 inst/htmlwidgets/lib 目录中。依赖关系是使用 YAML 配置文件指定的，该文件使用控件的名称作为其基本文件名。以下是我们的 sigma.yaml 文件的样子

```
dependencies:
  - name: sigma
    version: 1.0.3
    src: htmlwidgets/lib/sigma-1.0.3
    script:
      - sigma.min.js
      - plugins/sigma.parsers.gexf.min.js
```

依赖关系 src 详述了引用目录，包含库和指定的 JavaScript 代码文件。如果包含多个 JS 脚本，每一个占每一行，并且以 '-开头。同时你可以天剑 stylesheet 条目，还有元条目或头条目，多依赖关系可以在一个 YAML 文件中声明，更多的请参考 htmlDependency 函数，该函数在htmltools包中

1.3.3 R 绑定 (R binding)

我们需要为用户提供一个调用我们的控件的 R 函数。通常，该函数将接受输入数据以及控制控件的显示的各种选项。下面是 sigma 的 R 函数：

```r
#' @import htmlwidgets
#' @export
sigma <- function(gexf, drawEdges = TRUE, drawNodes = TRUE,
                  width = NULL, height = NULL) {

  # read the gexf file
```

```
  data <- paste(readLines(gexf), collapse="\n")

  # create a list that contains the settings
  settings <- list(
    drawEdges = drawEdges,
    drawNodes = drawNodes
  )

  # pass the data and settings using 'x'
  x <- list(
    data = data,
    settings = settings
  )

  # create the widget
  htmlwidgets::createWidget("sigma", x, width = width, height = height)
}
```

函数包含两类输入：GEXF 数据文件和一些附加的设置参数用来控制如何显示图片。这些输入都集中在一个叫做 x 的列表中，然后灌入到 htmlwidgets::createWidget 函数。这个 x 变量随后将被用于 sigma 的 JavaScript 绑定（下面将对此进行描述），指定的任何宽度或高度参数也会被转发到 widget（默认情况下，控件自动调整大小，因此通常不需要显式的宽度或高度）。

我们也希望 sigma 控件能够在 shiny 应用中使用，因此我们添加了下面的公式化的 shiny output 和 render 函数（对于所有的控件来说，它总是相同的）

```
#' @export
sigmaOutput <- function(outputId, width = "100%", height = "400px") {
  htmlwidgets::shinyWidgetOutput(outputId, "sigma", width, height, package = "sigma")
}
#' @export
#https://blog.csdn.net/songzhilian22/article/details/49487467
renderSigma <- function(expr, env = parent.frame(), quoted = FALSE) {
  if (!quoted) { expr <- substitute(expr) } # force quoted
  htmlwidgets::shinyRenderWidget(expr, sigmaOutput, env, quoted = TRUE)
}
```

1.3.4 JavaScript 绑定 (JavaScript binding)

注意：在 htmlwidgets0.5.2 和更早的版本中使用了一个更老、更不直观的 JavaScript 绑定 API，并在 htmlwidgets 的更新版本中继续支持。有关遗留绑定 API 的详细信息，请参见此归档版本。新的控件件被鼓励使用下面描述的更新的 API。谜题中的第三个部分是激活控件所需的 JavaScript。按照惯例，我们将在文件 inst/htmlwidgets/sigma.js 中定义 JavaScript 绑定。下面是绑定的完整源代码：

```
HTMLWidgets.widget({

  name: "sigma",

  type: "output",

  factory: function(el, width, height) {

    // create our sigma object and bind it to the element
    var sig = new sigma(el.id);
```

```
    return {
      renderValue: function(x) {

        // parse gexf data
        var parser = new DOMParser();
        var data = parser.parseFromString(x.data, "application/xml");

        // apply settings
        for (var name in x.settings)
          sig.settings(name, x.settings[name]);

        // update the sigma object
        sigma.parsers.gexf(
          data,          // parsed gexf data
          sig,           // sigma object
          function() {
            // need to call refresh to reflect new settings and data
            sig.refresh();
          }
        );
      },

      resize: function(width, height) {

        // forward resize on to sigma renderers
        for (var name in sig.renderers)
          sig.renderers[name].resize(width, height);
      },

      // Make the sigma object available as a property on the widget
      // instance we're returning from factory(). This is generally a
      // good idea for extensibility--it helps users of this widget
      // interact directly with sigma, if needed.
      s: sig
    };
  }
});
```

我们为控件提供了名称和类型，再加上一个工厂函数，它采用 el（将承载这控件的 HTML 元素）、宽度和高度（HTML 元素的宽度和高度，以像素为单位），您总是可以使用 OffStStand 宽度和 OffSETHEE 来实现这一点。

工厂函数要准备启动接收 HTML 元素的值。在这个案例，我们创建一个新的 sigma 元素和把它的 DOM 元素的 ID，承载页面的控件。

我们稍后需要访问 sigma 对象（以更新它的数据和设置），因此我们将其保存为变量 sig。请注意，直接在工厂函数内部声明的变量与特定的控件实例/el 绑定。

工厂函数的返回值被称为控件实例对象。它是 htmlwidgets 运行时和正在包装的 JavaScript 可视化之间的桥梁。顾名思义，每个控件实例对象负责管理页面上的单个控件实例。

您创建的控件实例对象必须有一个所需的方法，并且可以有一个可选的方法：

1. 所需的 renderValue 方法实际上将动态数据和设置填充到 WEB 的 DOM 元素中。x 包含控件数据和设置。我们解析和更新 GEXF 数据，将设置应用到我们先前创建的 sig Sigma 对象，最后调用刷新以反映屏幕上的新值。这种方法可以重复调用不同的数据（例如：在 shiny 中），所以一定要考虑到这种可能性。如果它对你的控件有意义，考虑使您的可视化转换顺利地从 x 的一个值开始到另一个。
2. 每当包含控件的元素被调整大小时，就会调用可选的大小调整方法。不执行此方法的唯一原因是如果您的

控件自然缩放（当它的元素大小改变时不需要附加的 JavaScript 代码）。在 sigma .js 的情况下，我们将大小调整信息转发给每个底层 sigma 渲染器。

所有 JavaScript 库都处理初始化、绑定到 DOM 元素、动态更新数据和稍微不同地调整大小。创建控件的 JavaScript 方面的大部分工作是将这三个函数工厂、渲染值和大小正确地映射到底层库的行为。sigma.js 示例使用一个简单的对象文字来创建它的控件实例对象，但是您也可以使用基于类的对象或任何其他样式的对象，只要 obj.renderValue(x) 和 obj.resize(width, height)（宽度，高度）可以调用它。

可以在控件实例对象上添加其他方法和属性。虽然它们不会被 htmlwidgets 本身调用，但它们可能对知道一些 JavaScript 的控件的用户有用，并希望通过添加自定义 JS 代码（例如使用 htmlwidgets::onRender R 函数）来进一步定制您的控件。在这种情况下，我们添加一个 s 属性，使 sigma 对象本身可用。

1.3.5 演示

我们的控件现在完成了！如果您想在不重放所有代码的情况下测试它，您可以从 GitHub 安装它如下：

```
devtools::install_github('jjallaire/sigma')
```

下面是代码的示例，其中包含了包中包含的一些示例数据：

```
library(sigma)
sigma(system.file("examples/ediaspora.gexf.xml", package = "sigma"))
```

如果在 R 控制台中执行此代码，您将看到在 RStudio Viewer 中显示的控件（或者如果不运行 RStudio，则在外部浏览器中）。如果将其包含在 R Markdown 文档中，则窗口控件将嵌入到文档中。

我们还可以在 shiny 应用程序中使用控件：

```
library(shiny)
library(sigma)

gexf <- system.file("examples/ediaspora.gexf.xml", package = "sigma")

ui = shinyUI(fluidPage(
  checkboxInput("drawEdges", "Draw Edges", value = TRUE),
  checkboxInput("drawNodes", "Draw Nodes", value = TRUE),
  sigmaOutput('sigma')
))

server = function(input, output) {
  output$sigma <- renderSigma(
    sigma(gexf,
          drawEdges = input$drawEdges,
          drawNodes = input$drawNodes)
  )
}

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

1.4 创建你自己的 widgets

1.4.1 需求（Requirements）

要实现一个控件，您需要创建一个新的 R 包，而这又取决于 htmlwidgets 包。可以在 CRAN 中安装：

```
install.packages("htmlwidgets")
```

1.4.2 脚手架（Scaffolding）

要创建一个新的控件，可以调用 **scaffoldWidget**函数来生成控件的基本结构。函数将：

- 创建.R,.js,.yaml 等控件需要的文件
- 如果提供，取一个Bower包名称并自动下载 JavaScript 库（及其依赖项），并将所需的条目添加到.yaml 文件中。

这个方法是非常推荐的，因为它确保你开始使用正确的文件结构。下面是一个示例，假设您希望在一个新的同名包中创建名为 "mywidget" 的小部件：

```
devtools::create("mywidget")                    # create package using devtools
setwd("mywidget")                               # navigate to package dir
htmlwidgets::scaffoldWidget("mywidget")     # create widget scaffolding
devtools::install()
```

这将创建一个简单的控件，它使用单个文本参数，并在控件 HTML 元素中显示该文本。你可以这样试试：

```
library(mywidget)
mywidget("hello, world")
```

这是最可能的控件，并且还没有包含一个 JavaScript 库来连接（注意，scaffoldWidget 可以可选地包含通过 JavaScript 库依赖关系的 bowerPkg 参数）这是最可能的小部件，并且还没有包含一个 JavaScript 库来连接（注意，scaffoldWidget 可以可选地包含通过 JavaScript 库依赖关系的 bowerPkg 参数）。在开始开发之前，您应该查看上面的介绍性示例，以确保您理解各个组件，并在下一节中查看与文章相关联的附加文章和示例。

1.4.3 更多

1.4.3.1 其他

还有更多的文章覆盖更高级的领域：

- HTML Widget Sizing：解释自定义大小调整策略以及何时可能需要使用它们，并描述在 JavaScript 绑定中实现调整大小的方法。
- HTML Widgets: Advanced Topics：描述支持每个控件实例数据、数据转换（例如，将数据帧转换为 D3 数据集）以及提供 live JavaScript 对象（例如函数定义）的控件选项的框架特征。

当大多数 JavaScript 库需要一些额外的交互以保持它们的大小与它们的包含元素同步时，HTML Widget Sizing 就显得尤为重要。

1.4.3.2 例子

学习其他包的代码是了解更多关于创建小部件的一个好方法：

- networkD3
- dygraphs
- sparkline

1.4.3.3 问题

如果您对开发控件或开发过程中遇到的问题有疑问，请毫不犹豫地在项目的 GitHub 存储库上发布一个问题。

Chapter 2

HTML 空间尺寸调整

2.1 概述

在 HTML 控件的工作中，就像 R 中的绘图 (plot)，HTML 控件智能的将他们自己的大小放在容器中，无论是在 Rstudio Viewer 中，knitr 中的图还是在 Shiny UI 中的面板。**htmlwidgets**框架提供了一种丰富的机制来指定控件的大小调整行为。

这种大小调整机制是为了解决影响控件的自然尺寸的以下约束：

- **The kind of widget it is**. 有些控件可能仅仅需要设计成小的，固定尺寸 (例如sparkline), 而有些控件可能需要基于像素点做不断地调整 (例如network graphs)
- **The context into which the widget is rendered.**有些控件在 R Markdown 中看起来是以 $980px \times 480px$, 相同的控件在 Rstudio 的 Viewer 中看起来要小的多。

分两步处理控件的大小：

1. 首先为控件指定大小调整策略，这是通过 createWidget 函数中的 sizingPolicy 参数实现的。大多数控件可以接受默认的大小调整策略 ((或者只覆盖其中的一个或两个方面）并获得满意的大小调整行为（详见下文）。)
2. 框架使用大小调整策略来计算给定的窗口中所呈现的窗口的正确宽度和高度。然后将其大小信息传递给控件 JavaScript 绑定的初始化和调整大小方法。控件将大小的信息传递给底层的 JS 库。

2.2 指定大小调整策略

默认 HTML 窗口大小调整策略使用与 R 图相同大小的语义来处理控件。当在 R 控制台上打印时，窗口小部件显示在 RStudio Viewer 中，并且大小以填充查看器窗格（模数任何填充）。当在 R Markdown 文档中呈现时，控件大小基于默认的图片文件的大小。

需要注意的是对于大多数的控件来说默认的大小是很不错的选择，你不需要去创建一个大小调整的策略。如果你想轻微的修改大小策略，使得与默认的不同，那么你可以调用 sizingPolicy 函数，然后把结果传递给 createWidget 函数。例如：

```
htmlwidgets::createWidget(
  "sigma",
  x,
  width = width,
  height = height,
  sizingPolicy = htmlwidgets::sizingPolicy(
    viewer.padding = 0,
    viewer.paneHeight = 500,
```

```
    browser.fill = TRUE
  )
)
```

2.2.1 例子

networkD3包对所有的控件直接使用了自定义的大小策略，simpleNetwork 控件消除填充（因为 D3 已经提供填充），并且指定当它在一个独立的 Web 浏览器中显示时，它希望填充尽可能多的空间：

```
sizingPolicy(padding = 0, browser.fill = TRUE)
```

sankeyNetwork 控件需要比 Rstudio Viewer 提供的更多的空间或典型的 knitr 图片，因此它禁用了那些自动调整大小的行为。针对于 knitr 文档他还提供了一个恰当的默认的宽和高：

```
sizingPolicy(viewer.suppress = TRUE,
             knitr.figure = FALSE,
             browser.fill = TRUE,
             browser.padding = 75,
             knitr.defaultWidth = 800,
             knitr.defaultHeight = 500)
```

2.2.2 可用选项

以下是在大小调整策略中可以指定的各种选项：

Option	Description
defaultWidth	The default width used to display the widget. This parameter specifies the default width for viewing in all conte
defaultHeight	The default height used to display the widget. This parameter specifies the default height for viewing in all con
padding	Padding around the widget (in pixels). This parameter specifies the padding for viewing in all contexts (browse
viewer.defaultWidth	The default width used to display the widget within the RStudio Viewer.
viewer.defaultHeight	The default height used to display the widget within the RStudio Viewer.
viewer.padding	Padding around the widget when displayed in the RStudio Viewer (defaults to 15 pixels).
viewer.fill	When displayed in the RStudio Viewer, automatically size the widget to the viewer dimensions (note that viewe
viewer.suppress	Never display the widget within the RStudio Viewer (useful for widgets that require a large amount of space for
viewer.paneHeight	Request that the RStudio Viewer be forced to a specific height when displaying this widget.
browser.defaultWidth	The default width used to display the widget within a standalone web browser.
browser.defaultHeight	The default height used to display the widget within a standalone web browser
browser.padding	Padding around the widget when displayed in a standalone browser (defaults to 40 pixels).
browser.fill	When displayed in a standalone web browser, automatically size the widget to the browser dimensions (note th
browser.external	When displaying in a browser, always use an external browser (via browseURL()). Defaults to FALSE, which w
knitr.defaultWidth	The default width used to display the widget within documents generated by knitr (e.g. R Markdown).
knitr.defaultHeight	The default height used to display the widget within documents generated by knitr (e.g. R Markdown).
knitr.figure	Apply the default knitr fig.width and fig.height to the widget when it’s rendered within R Markdown docume

2.3 JavaScript 调整大小方法

设置大小策略允许 htmlwidgets 在展示图像的区域计算控件的宽和高，然而你仍然需要把你定义的这种大小信息传递给底层你创建控件所使用的 JavaScript 库。

每一个 JavaSript 库在处理动态大小调整问题时都有一些不同，有些可能自动调整，有些可能需要通过 resize() 函数调用布局，有些可能需要通过设置数据或其他选项实现。无论哪种情况，htmlwidhets 框架将把计算的大小传递给函数 factory 和 resize 函数. 下面是一个空的 JavaScript binding 的说明例子：

```
HTMLWidgets.widget({

  name: "demo",

  type: "output",

  factory: function(el, width, height) {

    return {
      renderValue: function(x) {

      },

      resize: function(width, height) {

      }
    };
  }
});
```

您所处理的宽度和高度是由您决定的，它取决于您正在创建控件的基础 JavaScript 库的大小调整语义。

2.3.1 例子

2.3.1.1 dygraphs

在dygraphs控件中，调整大小的实现现对简单，因为 dygraphs 库包含一个 resize() 方法来自动的将该图大小映射到封闭 HTML 元素。

```
resize: function(width, height) {
  if (dygraph)
    dygraph.resize();
}
```

2.3.1.2 forceNetwork

在forceNetwork通过适用于承载 D3 网络可视化的 <svg>标签来传递宽和高的大小变化。

```
factory: function(el, width, height) {

  // instance data
  var el = el;
  var force = d3.layout.force();

  d3.select(el).append("svg")
    .attr("width", width)
    .attr("height", height);

  return {
    renderValue: function(x) {
```

```
      // implementation excluded
    },

    resize: function(width, height) {

      d3.select(el).select("svg")
        .attr("width", width)
        .attr("height", height);

      force.size([width, height]).resume();
    }
  };
}
```

我们看到在不同的 JavaScript 库中，resize 方法可能会提供灵活的自动调整大小的逻辑，这是我们需要注意的。

Chapter 3

HTML 控件：高级主题

3.1 本节概述

本部分将介绍创建控件的几个主要方面，这些控件并不是所有控件都需要的，但他是获得绑定到某些类型的 JavaScript 库才能正常工作的重要部分。涵盖的主题包括：

- 将 R 对象的 JSON 表示转换为 JavaScript 库所需的表示（例如，R 数据帧到 D3 数据集）。
- 在 JavaScript 绑定中跟踪特定于实例的控件数据。
- 将 JavaScript 函数从 R 传递到 JavaScript（例如用户提供的格式化或绘图功能）。
- 生成自定义 HTML 以封装控件（默认为 `<div >`但一些库需要不同的元素，例如 `<SPAN >`）

3.2 数据变换

R 对象传递一个参数 x 给 createWidget() 函数，并使用内部函数 htmlwidgets:::toJSON() 转换成 JSON 字符串，默认情况下，它基本上是 jsonlite::toJSON() 的包装函数。但是，有时这种表示并不是您所连接的 JavaScript 库所要求的。有两个 JavaScript 函数可以用来转换 JSON 数据。

3.2.1 HTMLWidgets.dataframeToD3()

R 数据框是'long form'(长型数据: 数组名加一个向量) 然而 d3 需要一个'wide form'(宽型数据: 每一行都有以键值对形式表示)，在 R 中使用 dataframeToD3() 函数就可以把数据变化成 JavaScript 对用格式的数据

用一个例子来说明 R 的 long-form 数据

```
{
  "Sepal.Length": [5.1, 4.9, 4.7],
  "Sepal.Width": [3.5, 3, 3.2],
  "Petal.Length": [1.4, 1.4, 1.3],
  "Petal.Width": [0.2, 0.2, 0.2],
  "Species": ["setosa", "setosa", "setosa"]
}
```

使用 HTMLWidgets.dataframeToD3()，将会变成：

```
[
  {
    "Sepal.Length": 5.1,
```

```
    "Sepal.Width": 3.5,
    "Petal.Length": 1.4,
    "Petal.Width": 0.2,
    "Species": "setosa"
  },
  {
    "Sepal.Length": 4.9,
    "Sepal.Width": 3,
    "Petal.Length": 1.4,
    "Petal.Width": 0.2,
    "Species": "setosa"
  },
  {
    "Sepal.Length": 4.7,
    "Sepal.Width": 3.2,
    "Petal.Length": 1.3,
    "Petal.Width": 0.2,
    "Species": "setosa"
  }
]
```

作为一个实际例子，simpleNetwork 接受包含 R 侧的网络链接的数据框，然后将其转换为 JavaScript renderValue 函数内的 D3 表示：

```
renderValue: function(x) {

  // convert links data frame to d3 friendly format
  var links = HTMLWidgets.dataframeToD3(x.links);

  // ... use the links, etc ...

}
```

3.2.2 HTMLWidgets.transposeArray2D()

有时二维数组需要类似的换位。为此，提供了 transposeArray2D() 函数。下面是一个示例数组：

```
[
  [5.1, 4.9, 4.7, 4.6, 5, 5.4, 4.6, 5],
  [3.5, 3, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4],
  [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5],
  [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2],
  ["setosa", "setosa", "setosa", "setosa", "setosa", "setosa", "setosa", "setosa"]
]
```

HTMLWidgets.transposeArray2D() 把其变换成：

```
[
  [5.1, 3.5, 1.4, 0.2, "setosa"],
  [4.9, 3, 1.4, 0.2, "setosa"],
  [4.7, 3.2, 1.3, 0.2, "setosa"],
  [4.6, 3.1, 1.5, 0.2, "setosa"],
  [5, 3.6, 1.4, 0.2, "setosa"],
  [5.4, 3.9, 1.7, 0.4, "setosa"],
  [4.6, 3.4, 1.4, 0.3, "setosa"],
```

```
  [5, 3.4, 1.5, 0.2, "setosa"]
]
```

dygraphs 控件就使用了这种变换：

```
renderValue: function(x) {

    // ... code excluded ...

    // transpose array
    x.attrs.file = HTMLWidgets.transposeArray2D(x.attrs.file);

    // ... more code excluded ...
}
```

3.2.3 自定义 JSON 串行化器

当 htmlwidgets 中的默认 JSON 序列化器无法按您预期的方式工作时，您可能会发现需要自定义控件数据的 JSON 序列化。对于实现控件的包的作者，JSON 序列化有两个定制级别：您可以自定义 jsonlite::toJSON() 的参数的默认值，或者只需定制整个函数。

1. jsonlite::toJSON() 有很多参数，并且我们已经改变了它的很多默认值。下面是我们在 htmlwidgets 中使用的 JSON 序列化器：

```
function (x, ..., dataframe = "columns", null = "null", na = "null",
    auto_unbox = TRUE, digits = getOption("shiny.json.digits",
        16), use_signif = TRUE, force = TRUE, POSIXt = "ISO8601",
    UTC = TRUE, rownames = FALSE, keep_vec_names = TRUE, strict_atomic = TRUE)
{
    if (strict_atomic)
        x <- I(x)
    jsonlite::toJSON(x, dataframe = dataframe, null = null, na = na,
        auto_unbox = auto_unbox, digits = digits, use_signif = use_signif,
        force = force, POSIXt = POSIXt, UTC = UTC, rownames = rownames,
        keep_vec_names = keep_vec_names, json_verbatim = TRUE,
        ...)
}
```

例如，我们通过列而非行将数据框转化为 JSON(后者是 jsonlite::toJSON 的默认设置)，如果要更改任何参数的默认值，可以将属性 TOJSON ARGS 附加到将被传递给 createWidgets() 的 widgets 数据。例如：

```
fooWidget <- function(data, name, ...) {
  # ... process the data ...
  params <- list(foo = data, bar = TRUE)
  # customize toJSON() argument values
  attr(params, 'TOJSON_ARGS') <- list(digits = 7, na = 'string')
  htmlwidgets::createWidget(name, x = params, ...)
}
```

在上面的例子中，我们将数字的默认值从 16 改为 7，NA 从 NULL 改为 String. 您是否需要向用户公开这样的定制，这取决于包的作者。例如，您可以在控件中附加一个参数，这样用户就可以自定义 JSON 序列化程序的行为：

```
fooWidget <- function(data, name, ..., JSONArgs = list(digits = 7)) {
  # ... process the data ...
  params <- list(foo = data, bar = TRUE)
  # customize toJSON() argument values
```

```
  attr(params, 'TOJSON_ARGS') <- JSONArgs
  htmlwidgets::createWidget(name, x = params, ...)
}
```

还可以使用全局选项 htmlwidgets.TOJSON_ARGS 为当前会话中的所有控件自定义 JSON 序列化参数，例如：

```
options(htmlwidgets.TOJSON_ARGS = list(digits = 7, pretty = TRUE))
```

2. 如果不想使用 jsonlite, 可以通过将属性 TOJSON_FUNC 附加到 widget 数据，完全重写序列化函数，例如：

```
fooWidget <- function(data, name, ...) {
  # ... process the data ...
  params <- list(foo = data, bar = TRUE)
  # customize the JSON serializer
  attr(params, 'TOJSON_FUNC') <- MY_OWN_JSON_FUNCTION
  htmlwidgets::createWidget(name, x = params, ...)
}
```

这里 MY_OWN_JSON_FUNCTION 函数可以是一个将 R 对象转换为 JSON 的任意 R 函数。如果您还指定了 TOJSON_ARGS 属性，它也将传递给您的自定义 JSON 函数。注意这些自定义 JSON 序列化程序的特性要求在 Shiny 的应用程序中呈现控件时，Shiny 的版本大于 0.111。

3.3 传递 JavaScript 函数

正如您所期望的，从 R 传递到 JavaScript 的字符向量被转换成 JavaScript 字符串。但是，如果您希望允许用户提供自定义的 JavaScript 函数用于格式化、绘图或事件处理，该怎么办？对于这种情况，htmlwidgets 包包含一个 JS() 函数，它允许您在客户端接收到一个字符值时将其作为 JavaScript 进行评估。

例如，dygraphs 控件包括允许用户为各种上下文提供回调函数的 dyCallbacks 函数。这些回调被标记为包含 JavaScript，以便它们可以在客户端上转换成实际的 JavaScript 函数：

```
callbacks <- list(
  clickCallback = JS(clickCallback)
  drawCallback = JS(drawCallback)
  highlightCallback = JS(highlightCallback)
  pointClickCallback = JS(pointClickCallback)
  underlayCallback = JS(underlayCallback)
)
```

另一个例子是 DT 包控件，用户可以在加载和初始化表之后制定一个带有 JavaScript 的 intCallback 来执行。

```
datatable(head(iris, 20), options = list(
  initComplete = JS(
    "function(settings, json) {",
    "$(this.api().table().header()).css({'background-color': '#000', 'color': '#fff'});",
    "}")
))
```

如果将多个参数传递给 JS(),（例如上面的示例中），它们将被级联成由 \n分隔的单个字符串。

3.4 自定义控件 HTML

通常，控件的 HTML“外壳” 只是一个 元素，而这是对应于新的控件的默认行为，而这些控件不是以其他方式指定的。然而有时你需要不同的标签类型，例如 sparkline 控件中需要 ，因此实现以下自定义 HTML 生成函数：

```
sparkline_html <- function(id, style, class, ...){
  tags$span(id = id, class = class)
}
```

请注意，这个函数是在由 widgetname_html 实现的小程序包中查找的，因此它不必从包中正式导出或以其他方式注册到 htmlwidgets。

大多数的控件都不需要自定义 HTML 函数，但是如果需要为你的控件生成自定义的 HTML（例如你需要一个 `<input>`或 `<span>`而不是一个 `<div>`), 那么你应该是用 htmltools 包。

Chapter 4

htmlwidgets 包中函数的总结

一个完整的 R 包提交到 CRAN, 都会有一个函数使用的说明文档或手册，htmlwidgets也不例外, 我们将按照原说明文档的函数顺序，翻译解释 htmlwidgets 包中函数的用途和参数设置。这其中有些方法我们在前三章中已经提到。

4.1 htmlwidgets 包函数

4.1.1 htmlwidgets-package

使用 R 创建 HTML 控件的包的信息

描述

htmlwidgets 包提供了一个简单的创建 R 链接 JavaScript 包的框架，使用该框架创建控件可以：

1. 在 R 控制台上使用 JavaScript 可视化库，就像绘图一样
2. 在 R Markdown 文档和 Shiny Web 应用程序中嵌入控件
3. 对保存网页，共享分析成果

在 R 中可以参考下面文档 (这些说明文档译者已经在前三章翻译)

```
vignette("develop_intro", package = "htmlwidgets") #第一章
vignette("develop_sizing", package = "htmlwidgets") #第二章
vignette("develop_advanced", package = "htmlwidgets") #第三章
```

包的源码地址：https://github.com/ramnathv/htmlwidgets

作者

Ramnath Vaidyanathan, Joe Cheng, JJ Allaire, and Yihui Xie

4.1.2 createWidget

创建 HTML 控件函数

描述

基于控件的 YAML 文件和给定的 JavaScript 库创建一个 HTML 控件。

用法

```
createWidget(name, x, width = NULL, height = NULL,
  sizingPolicy = htmlwidgets::sizingPolicy(), package = name,
  dependencies = NULL, elementId = NULL, preRenderHook = NULL)
```

参数

name: 控件名称 (要和 YAML 和 JS 文档中的名称保持一致)

x: 传入控件的数据，要转化为 JSON 格式的数据。

width: 控件的宽度，默认值是 NULL, 默认会自适应调整宽度

height: 控件的高度，默认值是 NULL, 默认会自适应调整高度

sizingPoicy: 调整控件大小的策略，详见第二章

package: 定义控件所在的包（默认就是控件的名称）

dependencies: 控件额外的依赖 (YAML 文件定义之外)，特别是对于一些动态的依赖选项是有必要的

elementID: 使用控件的显示元素 ID(而不是自动生成的元素 ID) , 如果您有 JavaScript 与特定的实例控件交互这是很有必要的

preRenderHook: 一个运行在控件上的函数。

细节

更多细节可以参考第一章

值

htmlwidgets 对象, 这将在不同的上下文中智能地将自己打印到 HTML 中。包括 R 控制台，在 R Markdown 文档中，以及在 Shiny 的输出绑定中。

4.1.3 getDependency

获取 htmlwidgets 的 JS 和 CSS 依赖关系

描述

获取 htmlwidgets 的 JS 和 CSS 依赖关系

用法

```
getDependency(name, package = name)
```

参数

name: 控件的名称

package: 包的名称，默认是控件的名称

4.1.4 htmlwidgets-shiny

Shiny 连接 HTML 控件

描述

创建在 Shiny 中使用的 output 和 render 函数

用法

```
shinyWidgetOutput(outputId, name, width, height, package = name,
  inline = FALSE, reportSize = FALSE)

shinyRenderWidget(expr, outputFunction, env, quoted)
```

参数

outputId: 输出对应的 ID

name: 控件创建的名称

width,height: 必须是有效的 CSS 单元 (像：“100%”,“400px”,“auto”) 或者是数字

package: 包含控件的包

inline: 对输出使用一个行内的标签 (<spqn>)

reportSize: 应该在 Shiny 的会话客户端数据中报告控件的容器大小吗？

expr: 一个产生 HTML 控件的而表达式

outputFunction: Shiny 输出函数，与 render 函数对应

env: 在什么环境变量下计算 expr

quoted: expr 是引用的表达式（用 quote()）吗？这是有用的，如果你想保存变量中的表达式。

细节

这些功能放在控件内部，为 Shiny 创建控件和渲染控件所用，详细的可见下面的例子。

值

创建 Shiny 可用的 output 和 render 函数对

例子

```
# shiny output binding for a widget named 'foo'
fooOutput <- function(outputId, width = "100%", height = "400px") {
    htmlwidgets::shinyWidgetOutput(outputId, "foo", width, height)
  }

# shiny render function for a widget named 'foo'
renderFoo <- function(expr, env = parent.frame(), quoted = FALSE) {
    if (!quoted) { expr <- substitute(expr) } # force quoted
    htmlwidgets::shinyRenderWidget(expr, fooOutput, env, quoted = TRUE)
  }
```

4.1.5 JS

把 R 中的字符串转化成合法的 JavaScript 脚本

描述

JS() 函数把接受到的字符向量转化成客户端的 JavaScript 脚本

用法

```
JS(...)
```

参数

...: 一个字符串，字符串的内容就是 JavaScript 脚本

作者

Yihui Xie

例子

```
library(htmlwidgets)
JS('1 + 1')
list(x = JS('function(foo) {return foo;}'), y = 1:10)
JS('function(x) {', 'return x + 1;', '}')
```

4.1.6 onRender

渲染后执行自定义的 JavaScript 代码

描述

使用这个函数来补充控件内置的 JavaScript 渲染逻辑自定义 JavaScript 代码，只针对这个特定的控件对象。

用法

```
onRender(x, jsCode, data = NULL)
```

参数

x: 一个 HTML 控件

jsCode: JS 代码字符串

data: 传给 JS 的数据，转化为 JSON 数据

例如

```
## Not run:
library(leaflet)
# This example uses browser geolocation. RStudio users:
# this won't work in the Viewer pane; try popping it
# out into your system web browser.
leaflet() %>% addTiles() %>%
  onRender("
  function(el, x) {
    // Navigate the map to the user's location
    this.locate({setView: true});
    }
    ")

# This example shows how you can make an R data frame available
# to your JavaScript code.

meh <- "&#x1F610;";
yikes <- "&#x1F628;";
df <- data.frame(
        lng = quakes$long,
        lat = quakes$lat,
        html = ifelse(quakes$mag < 5.5, meh, yikes),
        stringsAsFactors = FALSE
        )

leaflet() %>% addTiles() %>%
  fitBounds(min(df$lng), min(df$lat), max(df$lng), max(df$lat)) %>%
    onRender("
          function(el, x, data) {
                for (var i = 0; i < data.lng.length; i++) {
                    var icon = L.divIcon({className: '', html: data.html[i]});
```

```
                        L.marker([data.lat[i], data.lng[i]], {icon: icon}).addTo(this);
                        }
                    }
            ", data = df)

## End(Not run)
```

4.1.7 onStaticRenderComplete

静态渲染后执行 JavaScript 代码

描述

该机制是为运行代码定制控件实例而设计的，这是无法在页面加载时间完成的，因为控件实例还没有被创建。

用法

```
onStaticRenderComplete(jsCode)
```

值

创建了一个 htmltools 包中的 tags$script 对象

例子

```
## Not run:
library(leaflet)
library(htmltools)
library(htmlwidgets)

page <- tagList(
    leaflet() %>% addTiles(),
    onStaticRenderComplete(
      "HTMLWidgets.find('.leaflet').setZoom(4);"
        )
      )

print(page, browse = TRUE)

## End(Not run)
```

4.1.8 prependContent

将附加 HTML 内容添加到控件中

描述

使用这些函数将额外的 HTML 内容（主要是 JavaScript 和/或 CSS 样式）附加到控件，用于在独立模式下进行渲染（即在 R 控制台上打印）或在 knitr 文档中进行渲染。当在 Shiny 的控件渲染函数中运行时，这些函数不受支持，并且将在该上下文中使用警告。允许多个调用，并且稍后的调用不会撤销以前调用的影响。

用法

```
prependContent(x, ...)
appendContent(x, ...)
```

参数

x: HTML 标签对象

...: 有效的标签，文本或者 HTML. 或者是他们的一个列表

值

输出一个调整了的 HTML 控件

4.1.9 saveWidget

把控件保存称 HTML 文件

描述

把控件保存成 HTML 文件

用法

```
saveWidget(widget, file, selfcontained = TRUE, libdir = NULL,
  background = "white", title = class(widget)[[1]], knitrOptions = list())
```

参数

widget: 需要保存的控件

file: 保存的路径

selfcontained: 是否保存成一个自包含的 HTML 文档

libdir: HTML 依赖的包的路径

background: HTML 背景色

title: 产生主页的标题

knitrOptions: 一个 knitr 代码块的选项列表

4.1.10 scaffoldWidget

为 HTML 控件创建实现脚手架

描述

将一个 HTML 控件实现的最小代码添加到一个 R 包中。这个函数必须从要添加小部件的包的根目录中执行。

用法

```
scaffoldWidget(name, bowerPkg = NULL, edit = interactive())
```

参数

name: 控件的名称

bowerPkg: 这个控件基于 Bower 包的可选名称。如果您指定这个参数然后 Bower 将被用来自动下载控件。源代码和依赖项，并将它们添加到小部件的 YAML 中。

edit: 在创建脚手架后，自动打开控件的 JavaScript 源文件。

4.1.11 setWidgetIdSeed

为控件元素 ID 设置随机种子

描述

设置用于生成控件元素 ID 的随机种子。调用这个函数而不是依赖默认行为确保跨会话的稳定控件 ID。

用法

```
setWidgetIdSeed(seed, kind = NULL, normal.kind = NULL)
```

参数

seed: 一个值，整数或者为 NULL

kind: character or NULL

normal.kind: character string or NULL.

4.1.12 sizingPolicy

创建一个空间大小调整策略

描述

Define the policy by which HTML widgets will be sized in various containers (e.g. Browser, RStudio Viewer, R Markdown, Shiny). Note that typically widgets can accept the default sizing policy (or override only one or two aspects of it) and get satisfactory sizing behavior via the automatic sizing logic built into the htmlwidgets framework (see the notes below for the most typical exceptions to this).

用法

```
sizingPolicy(defaultWidth = NULL, defaultHeight = NULL, padding = NULL,
  viewer.defaultWidth = NULL, viewer.defaultHeight = NULL,
  viewer.padding = NULL, viewer.fill = TRUE, viewer.suppress = FALSE,
  viewer.paneHeight = NULL, browser.defaultWidth = NULL,
  browser.defaultHeight = NULL, browser.padding = NULL,
  browser.fill = FALSE, browser.external = FALSE,
  knitr.defaultWidth = NULL, knitr.defaultHeight = NULL,
  knitr.figure = TRUE)
```

参数

defaultWidth: The default width used to display the widget. This parameter specifies the defaultwidth for viewing in all contexts (browser, viewer, and knitr) unless it is specifically overridden with e.g. browser.defaultWidth.

defaultHeight: The default height used to display the widget. This parameter specifies the default height for viewing in all contexts (browser, viewer, and knitr) unless it is specifically overridden with e.g. browser.defaultHeight

padding: Padding around the widget (in pixels). This parameter specifies the padding for viewing in all contexts (browser and viewer) unless it is specifically overriden by e.g. browser.padding.

viewer.defaultWidth: The default width used to display the widget within the RStudio Viewer

viewer.defaultHeight: The default height used to display the widget within the RStudio Viewer.

viewer.padding: Padding around the widget when displayed in the RStudio Viewer (defaults to 15 pixels).

viewer.fill: When displayed in the RStudio Viewer, automatically size the widget to the viewer dimensions (note that viewer.padding is still applied). Default to TRUE.

viewer.suppress: Never display the widget within the RStudio Viewer (useful for widgets that require a large amount of space for rendering). Defaults to FALSE.

viewer.paneHeight: Request that the RStudio Viewer be forced to a specific height when displaying this widget.

browser.defaultWidth: The default width used to display the widget within a standalone web browser.

browser.defaultHeight: The default height used to display the widget within a standalone web browser.

browser.padding: Padding around the widget when displayed in a standalone browser (defaults to 40 pixels).

browser.fill: When displayed in a standalone web browser, automatically size the widget tothe browser dimensions (note that browser.padding is still applied). Defaults to FALSE.

browser.external: When displaying in a browser, always use an external browser (via browseURL()). Defaults to FALSE, which will result in the use of an internal browser within RStudio v1.1 and higher.

knitr.defaultWidth: The default width used to display the widget within documents generated by knitr (e.g. R Markdown).

knitr.figure: Apply the default knitr fig.width and fig.height to the widget when it's rendered within R Markdown documents. Defaults to TRUE.

细节

The default HTML widget sizing policy treats the widget with the same sizing semantics as an R plot. When printed at the R console the widget is displayed within the RStudio Viewer and sized to fill the Viewer pane (modulo any padding). When rendered inside an R Markdown document the widget is sized based on the default size of figures in the document.

You might need to change the default behavior if your widget is extremely large. In this case you might specify viewer.suppress = TRUE and knitr.figure = FALSE as well provide for a larger default width and height for knitr.

You also might need to change the default behavior if you widget already incorporates padding. Inthis case you might specify viewer.padding = 0. For additional details on widget sizing:vignette("develop_sizing", package = "htmlwidgets")

相关细节可以参考第二章

值

一个控件大小的调整策略

Chapter 5

总结

R 语言之所以在数据科学领域受到如此的重视，特别是在交互可视化方面要优于 Python(虽然 Python 也有像 bokeh,pyecharts,dash 等优秀的交互可视化模块)。我认为其主要原因在于 R 通过 htmlwidgets 包很好的融合 JavaScript，使得 R 可视化可以尽情的调用漂亮的 JavaScript 前端可视化的库，像 Echarts,D3 等等，并且原生的与 R Markdown,Shiny 结合实现动态的基于 Web 的系统原型搭建和自动化分析报告展示。

通过 htmlwidgets 的学习，相信大家应该明白像 Leaflet,rbokeh,networkD3,recharts,threejs,DataTables 等等这样的优秀的交互可视化和数据展示的 R 包是如何实现的了，并能基于 JavaScript 实现自己的交互可视化 R 包，期待读者基于 htmlwidgets 的 R 包的问世。

最后引用 Rstudio 官网对于 htmlwigets 包的描述：

- 把最好的 JavaScript 数据可视化应用到 R
- 在 R 控制台上使用 JavaScript 可视化库，就像绘图一样
- 在 R Markdown 文档和 Shiny Web 应用程序中嵌入控件
- 使用一个无缝连接 R 和 JavaScript 的框架开发新的控件

Chapter 6

参考文献

[1]. GitHub 上 htmlwidgets 的地址
[2]. CRAN 上 htmlwidgets 的地址
[3]. Rstudio 官网上的地址
[4]. 官方说明文档