

htmlwidgets 中文教程

徐静译

2018-07-04

Contents

声明	5
序言	7
关于译者	9
1 简介	11
1.1 概况	11
1.2 简单的开始	11
1.3 例子 (sigma.js)	12
1.4 创建你自己的 widgets	17
2 HTML 空间尺寸调整	19
2.1 概述	19
2.2 指定大小调整策略	19
3 HTML 控件：高级主题	21
4 htmlwidgets 包中函数的总结	23
5 总结	25
6 参考文献	27

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。



Figure 1:

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

2018-07-02

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



Figure 1.1:

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。同时你可以添加 `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：

```
## @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 `OffSttStand` 宽度和 `OffSETHHEE` 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，则在外部分浏览器中）。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

Chapter 3

HTML 控件：高级主题

正在翻译中...

Chapter 4

htmlwidgets 包中函数的总结

正在翻译中...

Chapter 5

总结

总结

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