Advanced Shiny

David Granjon

2019-04-30

Contents

Pre-requisite

This book is intended for people already familiar with shiny. Some knowledges about HTML, CSS and Javascript are also appreciated but not mandatory.

Since several years, significant efforts have been made to make shiny production ready. Today there exists tools to bridge the gap between "artisanal" and "professionnal apps". Golem is a package that helps during this transition from local to production. Additionally, there exists prod-ready solutions to host these apps such as RStudio Connect, shiny server pro, shinyproxy, … However, most of shiny app in the wild still have the same design, in spite of shinydashboard and shinythemes development. There are alternatives such as shinymaterial and shiny semantic, but if we consider the number of existing HTML templates, very few are made available for shiny.

End of november 2018, the RinteRface project was officially released … TO DO

About the author

David Granjon … TO DO

Chapter 1

Advance Design

In this chapter, you will learn how to build your own html templates taken from the web and package them, so that they can be re-used at any time by anybody.

1.1 Selecting a good template

There exists tons of HTML templates over the web. However, only a few part will be suitable for shiny, mainly because of what follows:

- shiny is built on top of bootstrap 3 (HTML, CSS and Javascript framework), meaning that going for another framework might not be straightforward. However, shinymaterial and shiny.semantic are examples showing this can be possible.
- shiny relies on jQuery (currently v 1.12.4 for shiny, whereas the latest version is 3.3.1). Consequently, all templates based upon React, Vue and other Javascript framework will not be natively supported. Again, there exist some examples for React with shiny and more generally, the reactR package developed by Kent Russell (**?** on Twitter) and Alan Dipert from RStudio.

See the github repository for more details about all dependencies related to the shiny package.

Therefore in the following, we will restict ourself to Bootstrap (3 and 4) together with jQuery. Don't be disapointed since there is still a lot to say.

> Notes: As shiny depends on Bootstrap 3.3.7, we recommand the user who would like to experiment Boostrap 4 features to be particularly careful about potential incompatibilies. See a working example here with bs4Dash.

A good source of **open source** HTML templates is Colorlib and Creative Tim. You might also buy your template, but forget about the packaging option, which would be illegal in this particular case, unless you have a legal agreement with the author (very unlikely however).

Chapter 2

About htmltools

While building a custom html template, you will need to know more about the wonderful htmltools developed by Winston Chang, member of the shiny core team. It has the same spirit as devtools, that is, making your web developer life easier. What follows does not have the pretention to be an exhaustive guide about this package. Yet, it will provide you yith the main tools to be more efficient.

2.1 HTML Tags

Both shiny and htmltools contain tags. However, by experience, htmltools contains more exported tags than shiny. For instance, the HTML `<nav></nav>` tag, namely `tags$nav()` in R is not included in the shiny package but in htmltools.

Within your package code, your tags will be like:

```
# we use htmltools tags instead of shiny
htmltools::tags$div(...)
```

If you had to gather multiple tags together, prefer `tagList()` as `list()`, although the HTML output is the same. The first has the shiny.tag.list class in addition to list.

2.1.1 Notations

Whether to use `tags$div` or `div` is the tag is exported by default. For instance, you could use `htmltools::div` but not `htmltools::nav` since nav does not have a dedicated function (only for p, h1, h2, h3, h4, h5, h6, a, br, div, span, pre, code, img, strong, em, hr). Rather use `htmltools::tags$nav`. Alternatively, there exists a function (in shiny and htmltools) called `withTags()`. Wrapping your code in this function enables you to use `withTags(nav(), ...)` instead of `tags$nav()`.

2.1.2 Alternative way to write tags

htmltools and shiny come with the `HTML()` function that you can feed with raw HTML:

```
HTML('<div>Blabla</div>')
# will render exactly like
div("Blabla")

# but there class is different
```

```
class(HTML('<div>Blabla</div>'))
class(div("Blabla"))
```

You will not be able to use tag related functions, as in the following parts. Therefore, I strongly recommand using R and not mixing HTML in R.

2.1.3 Tags structure

According to the htmltools `tag()` function, a tag has: - a name such as span, div, h1 ... - attributes, which you can access with `tag$attribs` - children, which you can access with `tag$children` - a class, namely "shiny.tag"

For instance:

```
# create the tag
myTag <- div(
  class = "divclass",
  id = "first",
  h1("Here comes your baby"),
  span(class = "child", id = "baby", "Ouinnnnn")
)

# access its name
myTag$name

# access its attributes (id and class)
myTag$attribs

# access children (returns a list of 2 elements)
myTag$children
```

How to modify the class of the second child?

```
second_children <- myTag$children[[2]]
second_children$attribs$class <- "adult"
myTag

# Hummm, this is not working ...
```

The code above is wrong. Indeed, by assigning `myTag$children[[2]]` to second_children, `second_children$attribs$class <- "adult"` modifies the class of the copy and not the original object. Only one way:

```
myTag$children[[2]]$attribs$class <- "adult"
myTag
```

For strongly nested tags, you will see that the following section contains amazing functions, such as `tagAppenChild()`.

2.1.4 Useful functions for Tags

htmltools and shiny have powerful functions to easily add attributes to tags, check for existing attributes, get attributes and add other tags to a list of tags.

- `tagAppendAttributes()`: this function allow you to add a new attribute to the current tag. For instance, assuming you created a div for which you forgot to add and id attribute:

```
mydiv <- div("Where is my brain")
mydiv <- tagAppendAttributes(mydiv, id = "here_it_is")
```

You can pass as many attributes as you want, including non standard attributes such as `data-toggle` (see Bootstrap 3 tabs for instance):

```
mydiv <- tagAppendAttributes(mydiv, `data-toggle` = "tabs")
# even though you could proceed as follows
mydiv$attribs[["aria-controls"]] <- "home"
```

- `tagHasAttribute()`: to check if a tag has a specific attribute

```
# I want to know if div has a class
mydiv <- div(class = "myclass")
has_class <- tagHasAttribute(mydiv, "class")
has_class

# if you are familiar with %>%
has_class <- mydiv %>% tagHasAttribute("class")
has_class
```

- `tagGetAttribute()`: to get the value of the targeted attributes, if it exists, otherwise NULL.

```
mydiv <- div(class = "test")
# returns the class
tagGetAttribute(mydiv, "class")
# returns NULL
tagGetAttribute(mydiv, "id")
```

- `tagAppendChild()` and `tagAppendChildren()`: add other tags to an existing tag. Whereas `tagAppendChild()` only takes on tag, you can pass a list of tags to `tagAppendChildren()`.

```
mydiv <- div(class = "parent", id = "mother", "Not the mama!!!")
otherTag <- span("I am your child")
mydiv <- tagAppendChild(mydiv, otherTag)
```

You might wonder why there is no `tagRemoveChild()` or `tagRemoveAttributes()`.

2.1.5 Other interesting functions

The brighter package written by Colin Fay contains very neat functions to edit your tags. Particularly, the `tagRemoveAttributes()`

```
remotes::install_github("Thinkr-open/brighter")
library(brighter)
```

```
mydiv <- div(class = "test", id = "coucou", "Prout")
tagRemoveAttributes(mydiv, "class", "id")
```

Up to you to create new functions to add in this package and do a nice PR.

2.2 Dependency utilities

When creating a new template, you sometimes need to import custom HTML dependencies that do not come along with shiny. No problem, htmltools is here for you (shiny also contains these functions).

2.2.1 The dirty approach

Let's consider the following example. I want to include a bootstrap 4 card in a shiny app. This example is taken from an interesting question here. The naive approach would be to include the HTML code directly in the app code

```
library(shiny)

# we create the card function before
my_card <- function(...) {
  htmltools::withTags(
    div(
      class = "card border-success mb-3",
      div(class = "card-header bg-transparent border-success"),
      div(
        class = "card-body text-success",
        h3(class = "card-title", "title"),
        p(class = "card-text", ...)
      ),
      div(class = "card-footer bg-transparent border-success", "footer")
    )
  )
}

# we build our app
shinyApp(
    ui = fluidPage(
      fluidRow(
       column(
        width = 6,
        align = "center",
        br(),
        my_card("blablabla. PouetPouet Pouet.")
       )
     )
    ),
    server = function(input, output) {}
)
```

and desesperately see that nothing is displayed. If you remember, this was expected since shiny does not contain bootstrap 4 dependencies and this card is unfortunately a bootstrap 4 object. Don't panic! We just need to tell shiny to load the css we need to display this card (if required, we could include the javascript as well). We could use either `includeCSS()`, `tags$head(tags$link(rel = "stylesheet", type = "text/css", href = "custom.css"))`. See more here.

```
shinyApp(
    ui = fluidPage(
      # load the css code
      includeCSS(path = "https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"),
      fluidRow(
       column(
        width = 6,
        align = "center",
        br(),
        my_card("blablabla. PouetPouet Pouet.")
```

```
      )
    )
  ),
  server = function(input, output) {}
)
```

The card is ugly (which is another problem we will fix later) but at least displayed.

When I say this approach is dirty, it is because it will not be easily re-usable by others. Instead, we prefer a packaging approach, like in the next section.

2.2.2 The clean approach

We will use the `htmlDependency` and `attachDependencies` functions from htmltools. The htmlDependency takes several arguments:

- the name of your dependency
- the version (useful to remember on which version it is built upon)
- a path to the dependency (can be a CDN or a local folder)
- script and stylesheet to respectively pass css and scripts

```
# handle dependency
card_css <- "bootstrap.min.css"
bs4_card_dep <- function() {
  htmltools::htmlDependency(
    name = "bs4_card",
    version = "1.0",
    src = c(href = "https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/"),
    stylesheet = card_css
  )
}
```

We create the card tag and give it the bootstrap 4 dependency through the `attachDependencies()` function.

```
# create the card
my_card <- function(...) {
  cardTag <- htmltools::withTags(
    div(
      class = "card border-success mb-3",
      div(class = "card-header bg-transparent border-success"),
      div(
        class = "card-body text-success",
        h3(class = "card-title", "title"),
        p(class = "card-text", ...)
      ),
      div(class = "card-footer bg-transparent border-success", "footer")
    )
  )

  # attach dependencies
  htmltools::attachDependencies(cardTag, bs4_card_dep())

}
```

We finally run our app:

```r
# run shiny app
ui <- fluidPage(
  title = "Hello Shiny!",
  fluidRow(
    column(
      width = 6,
      align = "center",
      br(),
      my_card("blablabla. PouetPouet Pouet.")
    )
  )
)

shinyApp(ui, server = function(input, output) { })
```

With this approach, you could develop a package of custom dependencies that people could use when they need to add custom elements in shiny.

Chapter 3

Template Skeleton

Now that you have the basis about tags and dependencies, we can go through the template organisation...
TO DO

Bibliography