Binding javascript (d3.js) to shiny

First, I am fairly unfamiliar with javascript and its library d3.js, but I am familiar with R. Creating dashboards using Shiny has been fun and easy (thanks to stackoverflow). Now I want to expand it by connect d3 elements to it.

I'm looking for information sources on how to actually bind javascript to Shiny (R dashboard) and explain what is actually going on.

Background: I did the tutorial on js and jquery on w3schools and learned (a bit) about d3 using Scott Murray's book (Interactive Data visualization for the web). I hoped this would be enough to make me understand the examples and explanation concerning how to build custom input/output bindings on the Shiny website:

http://shiny.rstudio.com/articles/building-inputs.html

But unfortunately I don't and I can't seem to find any examples which are in minimal working code. Many examples on github are to complex for me to dissect, most probably because of my little experience with javascript. Here is an examples of custom input binding with javascript:

https://github.com/jcheng5/shiny-js-examples/tree/master/input

Here is an example of an input & output binding I try to unfold:

```
<script src="http://d3js.org/d3.v3.js"></script>
<script type="text/javascript">
(function(){
  // Probably not idiomatic javascript.

  this.countValue=0;

  // BEGIN: FUNCTION
  updateView = function(message) {

    var svg = d3.select(".d3io").select("svg")

    svg.append("text")
      .transition()
      .attr("x",message[0])
      .attr("y",message[1])
      .text(countValue)
      .each("end",function(){
        if(countValue<100) {
          countValue+=1;
          $(".d3io").trigger("change");
        }
      })
  }
  // END: FUNCTION

  //BEGIN: OUTPUT BINDING
  var d3OutputBinding = new Shiny.OutputBinding();
  $.extend(d3OutputBinding, {
    find: function(scope) {
      return $(scope).find(".d3io");
    },
    renderError: function(el,error) {
      console.log("Foe");
    },
    renderValue: function(el,data) {
      updateView(data);
      console.log("Friend");
    }
  });
  Shiny.outputBindings.register(d3OutputBinding);
  //END: OUTPUT BINDING

  //BEGIN: INPUT BINDING
  var d3InputBinding = new Shiny.InputBinding();
  $.extend(d3InputBinding, {
    find: function(scope) {
      return $(scope).find(".d3io");
    },
    getValue: function(el) {
      return countValue;
    },
    subscribe: function(el, callback) {
      $(el).on("change.d3InputBinding", function(e) {
        callback();
      });
    }
```

```
  });
  Shiny.inputBindings.register(d3InputBinding);
 //END: OUTPUT BINDING

})()
</script>
```

Where "d3io" is a div element in the ui, updateView() is a function. Here is the ui:

```
#UI
library(shiny)

d3IO <- function(inputoutputID) {
  div(id=inputoutputID,class=inputoutputID,tag("svg","")) #; eerst zat ; erbij, maar werkt
blijkbaar ook zonder
}

# Define UI for shiny d3 chatter application
shinyUI(pageWithSidebar(

  # Application title
  headerPanel("D3 Javascript chatter",
              "Demo of how to create D3 I/O and cumulative data transfer"),

  sidebarPanel(
    tags$p("This widget is a demonstration of how to wire shiny direct to javascript,
without any input elements."),
    tags$p("Each time a transition ends, the client asks the server for another packet of
information, and adds it
            to the existing set"),
    tags$p("I can't claim this is likely to be idiomatic javascript, because I'm a novice,
but it allows d3 apps
            to do progressive rendering.  In real use, a more complex request/response
protocol will probably be
            required.  -AlexBBrown")
  ),

  mainPanel(
    includeHTML("d3widget.js"),
    d3IO("d3io") #Creates div element that d3 selects
    )
))
```

Here is the server file:

```
# SERVER
library(shiny)
# Define server logic required to respond to d3 requests
shinyServer(function(input, output) {

  # Generate a plot of the requested variable against mpg and only
  # include outliers if requested
  output$d3io <- reactive(function() {
    if (is.null(input$d3io)) {
      0;
    } else {
      list(rnorm(1)*400+200,rnorm(1)*400+200);
    }
  })
})
```

Specific questions:

1) The server.r seems to get input called "d3io" (input$d3io) since this is not defined in ui.r, I reasoned it must come from the javascript file. Which element does it actually refer to?

2) I have trouble understanding the custom binding part:

```
var d3OutputBinding = new Shiny.OutputBinding();
  $.extend(d3OutputBinding, {
    find: function(scope) {
      return $(scope).find(".d3io");
    },
    renderError: function(el,error) {
      console.log("Foe");
    },
    renderValue: function(el,data) {
      updateView(data);
      console.log("Friend");
    }
  });
  Shiny.outputBindings.register(d3OutputBinding);
```

My understanding is:

Create a new shiny outputbinding, first find the class .d3io (div element), if error then write to console "Foe" (is this special code?), if not error then renderValue using the function updateView using data (Where does it receive this value from?) and write to console "Friend". Finally register output.

Hope you guys can help! I'm creating a document with the steps on "The necessary steps to learn how to implement javascript into shiny when you don't know any javascript", I would love that!:)

Cheers, Long

javascript r d3.js shiny

asked Oct 30 '14 at 10:28

Sweetbabyjesus
148 1 9

+1 on "The necessary steps to learn how to implement javascript into shiny when you don't know any javascript" – Vincent Oct 30 '14 at 15:01

4 Answers

Hi Sweetbabyjesus (so fun to say). You had two questions:

> 1) The server.r seems to get input called "d3io" (input$d3io) since this is not defined in ui.r, I reasoned it must come from the javascript file. Which element does it actually refer to?

That phrase `input$d3io` has the following components:

- `input` is a parameter passed into the function - it's a list that stores the current values of all the widgets in the app.
- `$` is the member selector.
- `d3io` refers to the content of the div element with that id ('d3IO("d3io")') in the mainPanel of the UI.

> 2) I have trouble understanding the custom binding part:

```
var d3OutputBinding = new Shiny.OutputBinding();
```

That's right, this creates an instance of Shiny.OutputBinding and assigns it to the variable d3OutputBinding.

```
$.extend(d3OutputBinding, {
  find: function(scope) {
    return $(scope).find(".d3io");
  },
  renderError: function(el,error) {
    console.log("Foe");
  },
  renderValue: function(el,data) {
    updateView(data);
    console.log("Friend");
  }
});
```

This code extends the behaviour of d3OutputBinding with three functions called `find`, `renderError` and `renderValue`. Those three functions are required for a Shiny.OutputBinding.

`find` is the key because it returns a list of elements that should be passed into the two render functions via their `el` parameter. Notice it's returning elements whose css class is "d3io" - that's the same div mentioned earlier.

Note that `extend()` is a function of jQuery javascript library, and the `$` in this context is an alias for the jQuery object.

```
Shiny.outputBindings.register(d3OutputBinding);
```

Lets Shiny know that this newly configured object should be put to use now.

Cheers, Nick

answered Dec 12 '14 at 7:41

gknicker
3,942 1 9 22

I'm going to take a step back and assume you want the amazing results D3 is capable of, but aren't necessarily tied to D3. Essentially, I'll be answering this question:

What are the necessary steps to learn how to implement JavaScript into Shiny when you don't know any JavaScript?

While D3 is amazingly powerful, it's also notoriously difficult to master - even for many folks who

are quite comfortable with JavaScript. While I love D3 and use it almost every day, I'd recommend against it in this case. Instead, there's library called Plotly, which uses D3 in the background, but is built specifically for the scientific community and data scientists, so it's very friendly to the R community.

They have a thorough tutorial for connecting to Shiny and even have a ggplot2 converter if you're already familiar with that syntax, as many in the R world are. Unless your needs are very unusual, Plotly will likely serve your needs just as well as writing directly in D3, with a much more friendly learning curve.

answered Nov 9 '14 at 23:38

Chris Fritz
649 7 16

Plotly looks very user friendly, good suggestion. Rather than creating D3 images, I'm looking for the answer how to bind D3.js to shiny so I can tap in the existing awesome pool of D3 images (without the need to actually create them), which is greater than what plotly offers. I prefer to make use of the direct source (which is ambitious). – Sweetbabyjesus Nov 10 '14 at 10:43

Are you familiar with the rCharts package? It can work pretty well with Shiny and most of the output options are based on D3 variants. Two examples.

answered Dec 9 '14 at 23:37

Joe Jansen
306 2 5

Very busy with work, I haven't had the chance to post it. Note that this is a workaround using the customMessageHandler (and I'm not using custom input/output binding). Here goes:

Objective: Send data from data frame to create a D3JS tree using customMessageHandler.

Path: I've managed to send data in data.frame format to a d3js tree. After you click the actionbutton, it changes the data in the data frame to JSON format, then sends it to the js file which creates the tree. The data of the tree is hard-coded on "server.r".

Where's the code? On my github! https://github.com/SweetBabyJesus/shiny-d3js-simple-binding

Original: I created a tree algorithm based on CHAID to create insights from large datasets. People can upload their csv to the dashboard which then spits out the d3js tree:) The code is somewhat lengthy, therefore I cut it down for you and created a minimal code example.

Hope you like it.

Cheers, Long

answered Jan 21 at 11:00

Sweetbabyjesus
148 1 9