This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.

Building Outputs

Right out of the box, Shiny makes it easy to include plots, simple tables, and text as outputs in your application; but we imagine that you'll also want to display outputs that don't fit into those categories. Perhaps you need an interactive choropleth map or a googleVis motion chart.

Similar to custom inputs, if you have some knowledge of HTML/CSS/JavaScript you can also build reusable, custom output components. And you can bundle up output components as R packages for other Shiny users to use.

Server-Side Output Functions

Start by deciding the kind of values your output component is going to receive from the user's server side R code.

Whatever value the user's R code returns is going to need to somehow be turned into a JSON-compatible value (Shiny uses RJSONIO to do the conversion). If the user's code is naturally going to return something RJSONIO-compatible – like a character vector, a data frame, or even a list that contains atomic vectors – then you can just direct the user to use a function on the server. However, if the output needs to undergo some other kind of transformation, then you'll need to write a wrapper function that your users will use instead (analogous to `renderPlot` or `renderTable`).

For example, if the user wants to output time series objects then you might create a `renderTimeSeries` function that knows how to translate `ts` objects to a simple list or data frame:

```
renderTimeSeries <- function(expr, env=parent.frame(), quoted=FALSE) {
    # Convert the expression + environment into a function
    func <- exprToFunction(expr, env, quoted)

    function() {
      val <- func()
      list(start = tsp(val)[1],
           end = tsp(val)[2],
           freq = tsp(val)[3],
           data = as.vector(val))
    }
}
```

which would then be used by the user like so:

```
output$timeSeries1 <- renderTimeSeries({
    ts(matrix(rnorm(300), 100, 3), start=c(1961, 1), frequency=12)
})
```

Design Output Component Markup

At this point, we're ready to design the HTML markup and write the JavaScript code for our output component.

For many components, you'll be able to have extremely simple HTML markup, something like this:

```
<div id="timeSeries1" class="timeseries-output"></div>
```

We'll use the `timeseries-output` CSS class as an indicator that the element is one that we should bind to. When new output values for `timeSeries1` come down from the server, we'll fill up the div with our visualization using JavaScript.

Write an Output Binding

Each custom output component needs an *output binding*, an object you create that tells Shiny how to identify instances of your component and how to interact with them. (Note that each *instance* of the output component doesn't need its own output binding object; rather, all instances of a particular type of output component share a single output binding object.)

An output binding object needs to have the following methods:

`find(scope)`
: Given an HTML document or element (`scope`), find any descendant elements that are an instance of your component and return them as an array (or array-like object). The other output binding methods all take an `el` argument; that value will always be an element that was returned from `find`.

  A very common implementation is to use jQuery's `find` method to identify elements with a specific class, for example:

  ```
  exampleOutputBinding.find = function(scope) {
    return $(scope).find(".exampleComponentClass");
  };
  ```

`getId(el)`
: Return the Shiny output ID for the element `el`, or `null` if the element doesn't have an ID and should therefore be ignored. The default implementation in `Shiny.OutputBinding` reads the `data-output-id` attribute and falls back to the element's `id` if not present.

`renderValue(el, data)`
: Called when a new value that matches this element's ID is received from the server. The function should render the data on the element. The type/shape of the `data` argument depends on the server logic that generated it; whatever value is returned from the R code is converted to JSON using the RJSONIO package.

`renderError(el, err)`
: Called when the server attempts to update the output value for this element, and an error occurs. The function should render the error on the element. `err` is an object with a `message` String property.

`clearError(el)`
: If the element `el` is currently displaying an error, clear it.

Register Output Binding

Once you've created an output binding object, you need to tell Shiny to use it:

```
Shiny.outputBindings.register(exampleOutputBinding, "yourname.exampleOutputBinding");
```

The second argument is a string that uniquely identifies your output binding. At the moment it is unused but future features may depend on it.

← Previous Next →