BrowserVizDemo

Paul Shannon

October 30, 2017

Contents

1 Introduction

BrowserVizDemo implements an (initially *very* simple, minimally-featured) interactive R-to-web-browser x-y plotting capability. It is a direct subclass of *BrowserViz* and therefore uses its simple four-field JSON messages transported over websockets (see below). It thus illustrates the benefits of creating visualization tools in a web browser connected interactively to an R session. Powerful interactive graphics libraries are currently, and increasingly, available for HTML5 browsers running Javascript; *d3* and *cytoscape.js* are two examples.

BrowserVizDemo thus serves two purposes:

- It demonstrates the few simple steps required to create an interactive R/browser application by subclassing the *BrowserViz* package.
- It could be extended into a full, web-based replacement for the base *R* `plot` function, and perhaps inspire the creation of, or porting of, other popular R visualization tools to web brower graphics.

Please see the vignette for the Bioconductor package *BrowserViz* for a discussion of the underlying architecture, techniques and the message format used to communicate between R and the browser.

Note that extending *BrowserVizDemo* to include more capabilities, or to create other packages derived from *BrowserViz*, requires two sets of programming skills:

- A solid understanding of R.
- Good knowledge of Javascript, jQuery, and, quite probably, of d3.

For those having these skills, or willing to acquire them, is virutally no limit to the rich, interactive, graphical exploratory data analyses tools that can be created, in which the graphical richness is matched by all of the interactive power of the R programming language.

2 Technical Overview

(Please see the vignette for *BrowserViz* for a more comprehensive treatment of this topic.)

Just as the ubiquitous and language-neutral *websockets* protocol provides the *BrowserVizDemo* communication mechanism, so does *JSON* provide the message notation. Native data types in R (a named list) and Javascript (an object, with key:value pairs) are easily converted to and from JSON by libraries standard in each language. We have adopted a simple, adaptable data structure flexible enough for all of the uses so far encountered. In JSON (and Javascript):

```
{{cmd: "setWindowTitle", status: "request", callback:"handleResponse",
  payload: "BrowserVizDemo Demo"}}
```

Websocket servers both send and receive messages. Thus a typical *BrowserVizDemo* event begins with sending a message from one environment to the other, and often concludes with some sort of a return or "callback" message.

- *cmd*: the name of the operation the sender wishes to be performed by the receiver.
- *status*: might be "success", "failure", "error", "deferred response".
- *callback*: provided by the sender, this specifies the operation which the receiver is to call *in the client* after it (the receiver) completes the operation it was asked to perform.
- *payload* An open-ended data structure, sometimes empty, as simple as a character string, as complex as any conceivable deeply nested list.

3 The BrowserVizDemo Application

The *BrowserVizDemo* package adds only two methods to those provided by its base class:

- *plot*: takes an x and y vector as arguments
- *getSelection*: returns the names of all d3 selected points, in the browser plot, to R. d3 points are selected by dragging the mouse; the selected region is indicated by a red-bordered box.

These methods are inherited from *BrowserViz*:

- *port*
- *ready*
- *browserResponseReady*
- *getBrowserResponse*
- *closeWebSocket*
- *send*
- *getBrowserWindowSize*
- *getBrowserWindowTitle*
- *setBrowserWindowTitle*

S4 class inheritance thus proves very powerful. *BrowserViz* has almost 400 lines of R code; *BrowserVizDemo* has few than 60. In addition to the utility methods listed above, the base class provides code which greatly simplifies the *plumbing* needed to connect R and the browser: web socket setup, port choice, send and receive, JSON transformations, send-message/receive-response latencies, specifying functions to be called when incoming messages arrive. This ensures that the R programming of any derived class is as simple as possible.

We provide a similar benefit for the Javascript, though neither proper classes nor inheritance come native with the language. Instead *BrowserViz* includes, and we elsewhere host, *BrowserViz.js* a module of about 300 lines, which exposes about fifteen utility functions. These, like their counterparts in the BrowserViz S4 class, provide socket initialization, functiond dispatch, management of functions to call when the HTML DOM is ready, and when the socket connection has been opened and is ready for use.

We recommend that the HTML/Javascript/CSS portion of every BrowserViz subclass include this file in its header. The capabilities it provides are then available:

```
hub = BrowserViz();
demo = BrowserVizDemo();
demo.setHub(hub)
demo.addMessageHandlers()
hub.addOnDocumentReadyFunction(demo.initializeUI);
  // configure and open socket, run queued functions, enable
  // dispatch of incoming commands their Javascript handler functions
hub.start();
```

```
> library(BrowserVizDemo)
> plotter <- BrowserVizDemo(port=8000:8100);  # plenty of ports to choose from

[1] waiting in BrowserViz.ready, browserResponseReady not yet true
[1] browserResponseReady now true, after 1 sleepInterval/s of 0.100000

> stopifnot(ready(plotter))

[1] waiting in BrowserViz.ready, browserResponseReady not yet true
[1] browserResponseReady now true, after 1 sleepInterval/s of 0.100000

> title <- "simple xy plot test";
> setBrowserWindowTitle(plotter, title)

[1] "simple xy plot test"

> plot(plotter, 1:10, (1:10)^2)

[1] ""

> selectedPoints <- getSelection(plotter)
>   # selectedPoints will be an empty list -unless- you have selected some in the browser with your mouse.
> closeWebSocket(plotter)
```