

Web Development with R

Bay Area useR Group

Jeroen Ooms
jeroenooms@gmail.com

January 2010, San Francisco

R and Javascript: A match made in heaven?

R and Javascript have a lot in common:

- Both accessible, loose scripting languages.
- Both have big and quickly growing communities.
- Both have lots of libraries/packages available.
- Both have become de facto standards (by accident?).
- Both don't like Java.

But they also complete each other:

- R is powerfull: interfaces C, Fortran, Databases, Latex, etc.
Many computing packages available.
- Javascript is nice for the user, interfaces to DOM, CSS,
many widget/design packages available.

What is a Web Application

Some examples:

- yeroon.net/ggplot2 [video]
- yeroon.net/lme4 [video]
- [Pubertyplot](#) [video]
- [Stockplot](#) [video]
- [IRT tool](#) [video]

What is a Web Application

This presentation is about:

- Making Web Interfaces for Simple R applications.
- R runs on the server.
- User only needs a standard webbrowser.
- Programming in R and HTML/Javascript (No Java!).
- AJAX interfaces.

It is not about:

- Applications that require advanced queueing systems for running multiple simultaneous R jobs.
- Distributed Computing.
- Java.

Why Web Applications?

Convenient:

- Making new tools available to a wide audience.
- User only needs a standard web browser.
- Cross-platform.

Server-based by design:

- Efficient use of resources.
- Easier to maintain.
- Integration with existing services/databases/etc.

Because you have to:

- More and more services only available from the browser.
- Many people can't or don't want to install software.

Hello World!

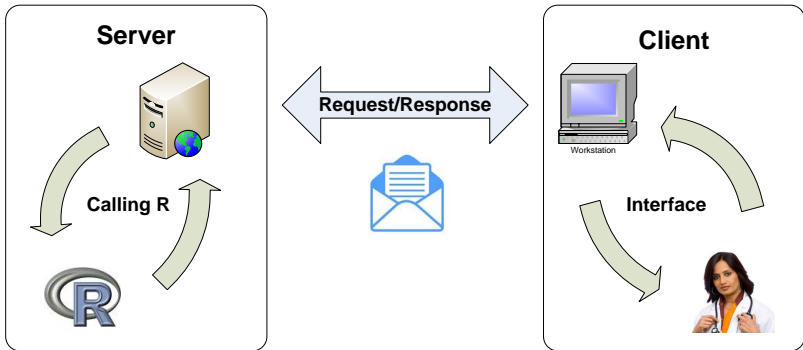
RApache

- RApache is a module for the Apache web server.
- It allows you to embed R scripts in a webpage.
- It is one of the ways of connecting the HTTPD to R.

Demo:

- Brew scripting.
- GET/POST.
- Try Catch.

Web application Setup



ggplot2 example

1. Interface:
 - Interactive webpage with menu, buttons, layer list, etc, that enables the user to specify a plot.
2. Request:
 - A configuration of a plot to draw (settings, mappings, layers)
3. Server:
 - Decodes the configuration, and draws the requested figure.
4. Response:
 - The plot figure.
5. Interface:
 - Updates figure within the page, and makes appropriate changes in GUI.

The Message Specification

Where the magic happens:

- How to get objects from the client's browser into R on the server?
- HTTP basically only takes strings for arguments.

Protocol Design:

- Maintain a request object in the browser (client).
- Define a way to encode your javascript object into a string.
- Use this string as argument to an HTTP/AJAX request.
- Decode the string on the server back to an object.
- Idem for the response message.

Message Specification: XML and JSON.

- For advanced web applications use a standard data format to communicate with the server.
- 2 obvious candidates: XML and JSON.
- You can use CRAN packages XML and rjson for parsing.
- XML is most widely used, probably preferable if you want to integrate your application in other software (consider SOAP).
- JSON is lighter and supports array's. Preferred for large datasets.

Example of XML Syntax

```
<myModel>
  <family>Gaussian</family>
  <deviance>3569.23</deviance>
  <coefficients>
    <coef>
      <name>Intercept</name>
      <value>5.69</value>
    </coef>
    <coef>
      <name>Age</name>
      <value>0.36</value>
    </coef>
    <coef>
      <name>Gender</name>
      <value>2.54</value>
    </coef>
  </coefficients>
</myModel>
```

Example of JSON Syntax

```
{ "myModel": {  
  "family": "Gaussian",  
  "deviance": 3569.23,  
  "coefficients":  
    [ {"Intercept": 5.69}, {"Age": 0.36}, {"Gender": 2.54} ]  
}
```

Or for example a dataframe:

```
{ "myData": {  
  "Age": [9, 8, 12, 6, 7, 8, 9, 8, 10, 11, 9, 6, 8],  
  "Gender": ["M", "F", "F", "M", "F", "M", "F", "F", "M", "F", "M", "F", "F"],  
  "Treatment": [1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0]  
}
```

Example: ggplot2



ggplot webapp: 3 R-scripts

ggplot-upload:

- Starts R session and reads a datafile from POST.
- Imports datafile to dataframe.
- Saves the session.
- Returns datafile variable info and session ID.

ggplot-png:

- Decodes plot-config and Session ID from POST.
- Loads the appropriate R session.
- Draws the plot and saves in a www dir.
- Returns a link to the figure (or an error).

ggplot-pdf

- Idem for PDF.

Javascript fragment

plotConfig is the object that holds information about the current plot. The function drawplot encodes and posts it.

```
function drawplot(){
  Ext.getCmp('workspace').load({
    timeout: 60, //timeout for load is in seconds
    url:"../brew/ggplot-png",
    method: "POST",
    params: {
      "plotRequest" : plotConfig.toJSON()
    },
    callback: function(el,success,response) {
      Ext.getCmp('drawButton').enable();
      plotResult = tryDecode(response.responseText);
    }
  });
}
```

Ime Request

Request Object:

```
{  
  "x": "map_mpa",  
  "y": "map_rating",  
  "colour": "map_mpa",  
  "facet": {},  
  "layers": {  
    "layer80667": {  
      "geom": "boxplot"  
    },  
    "layer39359": {  
      "geom": "point",  
      "position": "set_jitter"  
    }  
  },  
  "dataFile": 178032,  
  "width": 916,  
  "height": 270  
}
```

Server Fragment (R)

```
drawPlot <- function(){
  plotRequest <- fromJSON(as.character(POST$plotRequest));

  sessionID <- plotRequest$sessionID;
  load(sessionID);

  plotConfig <- plotRequest$plotConfig;
  fileName <- randomName();

  png(paste("/var/www/ggplot2/plots/", fileName, ".png", sep=""));
  drawPlot(plotConfig);
  dev.off();

  cat("<img src=\"/ggplot2/\", fileName, ".png", sep="");
}
```

Some Advice

- Use Firefox + Webdev toolbar + firebug for development.
- Design a protocol that makes sense semantically.
- Don't use R to generate Layout or HTML. This soon become unmanageable.
- Only computing at the server, do all layout stuff at the client.

Connecting your HTTPD to R

RApache/brew:

- Module for Apache webserver + scripting package.
- New R session for every http request.
- Fast, good caching.

Calling Rscript from php/cgi

- No need for extra plugins.
- Works on windows.
- Extra sensitive to code injection!

Rserve:

- Keeps an R session alive.
- Requires additional Rserve client.
- If session gets messed up, you're screwed.

Catch errors

To catch errors and return them to the user:

```
drawPlot <- function(){  
  //where actual stuff happens  
}  
  
printError <- function(e){  
  print("Oh no! An error occured:\n");  
  print(e$message);  
}  
  
tryCatch(drawPlot,error=printError);
```

Time limits

To limit the time of a session, start the R script with:

```
setTimeLimit (elapsed=40) ;
```

Be sure to reset the timer on the end of the script:

```
setTimeLimit () ;
```

Images

Format to display images:

- PNG, SVG, PDF, Flash.
- Bitmap or Vector Based?
- Browser support.
- File size.

Security

- Watch out for code injection!
- Parse user data safely. Don't let a user specify which function to run.
- If necessary, remove any weird characters from argument strings.

Graphical Interfaces

- There are many free and open-source javascript libraries that provide ready-to-use interfaces and widgets.
- Easily create Grids, Tabs, Windows, Drag/drop, Trees, Forms, Toolbars, etc with only a few lines of code.

| Framework | License |
|-------------------------------------|-------------------|
| Prototype/Scriptaculous | MIT |
| Yahoo! User Interface Library (YUI) | BSD |
| jQuery | GPL or MIT |
| Dojo | BSD |
| Ext | GPL or Commercial |

Thank you for your attention!

- Jeffrey Horner (2009). rapache: Web application development with R and Apache.
<http://biostat.mc.vanderbilt.edu/rapache/>
- Hadley Wickham (2009). ggplot2: An implementation of the Grammar of Graphics.
<http://CRAN.R-project.org/package=ggplot2>
- van Buuren S, Ooms JCL (2009). Stage line diagram: An age-conditional reference diagram for tracking development. *Statistics in Medicine*.
- w3schools: Full Web Building Tutorials - All Free.

