Dynamic Web Content: HTML Forms CGI Web Servers and HTTP

Duncan Temple Lang
Dept. of Statistics
UC Davis

Dynamic Content

- We are all used to fetching pages from a Web server.
- Most are prepared by a human and put on the Web site for others to read. Each reader sees the same page.
- The Web browser (client) can perform client-side computations on the page to present it differently. And the content can be dynamic via Javascript code in the page that operates as the page is being rendered (e.g. layout) and as it is being viewed (e.g. mouse operations)
- We, however, are interested in server-side dynamic content.

Server-side Dynamic Pages

- A request to the Web server returns a new, machine-generated page,
  e.g. google' search result page,
  travelocity's flight purchase page, ...
- The server receives the request and executes some code to create a new HTML document (or other type) and returns that.
- What makes this interesting is that the request can specify inputs that govern how the page is created.
- This is very similar to calling an R function with different inputs and generating output in the form of an HTML page

Client-Server components

- We are in a client-server setup, like with relational databases.
  One application acts as a client and connects to the server and sends a request; the server sends a reply.
- Our client is a browser and specifically an HTML form that provides ways to specify inputs to the server.
- Our "server" is an R script on the Web server that gets the inputs and generates an HTML page based these inputs.
- To provide dynamic access to R functionality via the Web, you need to create both pieces.

Outline

- Demo of an HTML form
- The elements of HTML form processing
  - HTML forms,
  - CGI scripts & CGIwithR package.
- Security issues.
- Basics of HTTP - how this all works.
- Details for project.
  accounts, server, Web server directory layout

Basic Demo

- A basic simulation to illustrate the Central Limit Theorem.
- User specifies inputs for
  - the sample size,
  - the distribution from which to sample,
  - the number of repetitions,
  - the statistic to compute for each sample,
  - whether to create numerical and/or graphical summary.

```
...
<form method="GET"
       action="/cgi-bin/R.cgi/simulation.R">
....
<input type="text" name="n" value="3" size="4">
.....
<select name="distribution" size="1">
  <option value="rnorm">Normal
  <option value="rbinom">Binomial
  <option value="rpois">Poisson
 </select>
.....
Histogram: <input type="checkbox" name="output"
       value="histogram" checked="checked">
Summary: <input type="checkbox" name="output"
        value="summary" checked="checked">
<input type="hidden"
       name="showAttribution" value="TRUE">
.....
<input type="submit"></center>
</form>
```

Basics of HTML Forms

- Create a form in an HTML document with
  <form> …</form>
- Specify the action which is the location (URI) of the script to run when the form is submitted.
- Also specify how the requested is to be submitted:
  GET or POST.
- Also add `enctype` attribute if necessary (more later).

<INPUT> Tags

- The elements of the form are created via <INPUT type="">, <SELECT>/<OPTION> and <TEXTAREA> tags in the HTML form
- Different `type` values
  TEXT, PASSWORD, CHECKBOX, RADIO,
  SUBMIT, RESET, IMAGE, HIDDEN, FILE.
- HTML Form Tutorial (http://www.w3schools.com/html/html_forms.asp) & another: these describe the different characteristics of these HTML elements to control their appearance and behavior.
- O'Reilly books: HTML - The Definitive Guide (chap. 8), and CGI Programming.

Key Aspects of <INPUT>...

- Each form element has a name attribute.
- This is the name of the parameter that is sent in the HTTP request.
- The selected `value`(s) for the element is sent as the value(s) for that parameter.
- In our example, get
  n = "3", output = c("histogram", "summary"),
  distribution = "rnorm", statistic = "median".
- The method attribute of <form> tag specifies how these name-values are sent to the Web server in the request.

The CGI script

- When a FORM is submitted, a request is sent to the Web server which identifies the URI as a script to run.
- The Web server calls that script passing it the inputs.
- This mechanism is called the Common Gateway Interface - CGI
- Typically, CGI scripts are located in the a cgi-bin area of the Web server's file system.
  Only these can be run via a Web request.
- The cgi-bin directory is not directly accessible via a Web browser, but in a directory parallel to the Web documents.

Script Programming Languages

- The scripts can be written in any language, e.g. C, Perl, Python, shell or R.
- Each language has utilities that make it easier to process the inputs into usable `arguments'.
- In R, there is a package called CGIwithR that makes using R scripts via CGI significantly easier.
- In the cgi-bin/ directory, we place the R.cgi and .Rprofile from that package.
- Then, we write our R script, say myCGI.R and put that into the cgi-bin/ directory.

Accessing the script

- To make use of this script (myCGI.R) in the HTML form, specify the script as in
  <form ... action="/cgi-bin/R.cgi/myCGI.R">
- This runs myCGI.R via the shell script R.cgi.
- Note R.cgi is an existing file, not a directory.
  You don't need to do anything to use it.
- All printed output (to the console/screen) generated in the R script is treated as HTML and displayed in the resulting document in the client Web browser,
  i.e. all output from cat() and print().

```
source("simulate.S") # Load work function for simulation.
ans = simulate(as.integer(formData$n), as.integer(formData$NumRep),
               formData$statistic, formData$distribution)

cat("<h2>Sample distribution of", formData$statistic,
    " for n =", formData$n, "from ", formData$distribution, "</h2>")

# Generate numerical summary if requested.
if("summary" %in% formData$output) {
   invisible(capture.output(library(R2HTML)))
   HTML(summary(ans), file = stdout())
}

# Generate density plot if requested.
if("histogram" %in% formData$output) {
 webPNG("hist.png", type = "jpeg", graphDir = "../htdocs/tmp/")
 dens = density(ans)
 hist(ans, prob = TRUE, ylim = c(0, max(dens$y)), main = "Density of
sample values")
 points(dens, col = "red")
 img(src = "hist.png", graphURLroot = "/tmp/")
}
```

Accessing the form inputs

- How does the R CGI script get the inputs from the form?
- When the script is run, there is already an R object - formData - that is a named list containing the elements of the form.
- formData$n yields "3" - convert it to a number with as.integer(formData$n)
- For the output, there may be multiple entries (e.g. "output" and "summary") so formData$output would be a character vector of length 2 if both are selected.

R2HTML Package

- The R2HTML can convert many types of R objects into HTML text.
- Load it in the R script via the command
  invisible(capture.output(library(R2HTML)))
  This hides any messages the package annoyingly produces when it is loaded.
- Then, to output an R object as HTML, use
  HTML(obj, file = stdout())
  e.g. HTML(summary(x), file = stdout())

R2HTML

- If you don't like what it produces, look at the documentation to customize it
- Or, generate the HTML content yourself from the R object.
- The CGIwithR package provides some basic functions to do this: img(), tag() & untag(), br(), lf(), mailto(), linkto().
- Also, the XML package has facilities for creating HTML/XML.

R Graphics in CGI

- Can't display graphics in a regular R window.
  Want to create JPEG, PNG, GIF, TIFF files and insert them inline into the HTML document via the
     <img src="filename">
  element.
- 2 issues: graphics device, file system.
- Might use jpeg() or png() graphics devices in R, as usual, but unfortunately, these typically require a connection to an X11 server. So can't use these.
- Instead, use other approaches. GDD or external program ghostscript.

Image Directories

- When the R script is running, the current working directory is the cgi-bin/ directory.
- We don't want to create the graphics file there, especially since that is not part of the document tree for the Web server and so cannot be seen by clients.
- So we create a directory, say Rimages/, under the htdocs/ directory and arrange to write the files there.
- When we write the <img src=> element in HTML, we need to specify <img src="Rimages/myFile.png">
- But when creating the graphical display, we are in a different directory and so must write to ../htdocs/Rimages

CGIwithR - webPNG(), img()

- Using CGIwithR, you can create graphics using webPNG("filename") and insert the image into the HTML content using img("filename").
- There are two global variables that control where the image is created and the URI in which it can be referenced: graphDir and graphURLroot.
- These are set in your cgi-bin/.Rprofile, so you don't have to worry about them.

2 Additional Examples

- 2 similar examples that share the same script and can even use the same form (with redundant elements).
- Illustrate techniques for getting dataset via HTML form.
- Perform a 1-variable bootstrap.
- User specifies the sample values, the statistic, the number of boostrap samples to generate and what output to create (histogram and/or numerical summary).
- In this case, the user provides data - 2 approaches.
  - insert into a <textarea name="values"> element
  - upload a file via <input type="file" name="dataFile">

Bootstrap Example

- In the <textarea> example, the formData$values contains the text from that element. We get the values via
  data = scan(textConnection(formData$values))
- In the case of the file upload, we have
  <input type="file" name="dataFile">
- formData$dataFile then contains the contents of that file, and we can use scan(textConnection()) again.

Differences

- While the two examples are very similar, and the same R script can be used which "gets" the data from different HTML elements, the underlying CGI mechanism is very different.
- The <textarea> version can be done with
  <form action="GET"…
  The <input type="file"> must be done with
  <form action="POST" enctype="multipart/form-data"…
- In the first case, the data come as
  URI?name=value&name=value&….
- `distribution=rnorm&n=3&statistic=mean&NumRep=1000&output=histogram&output=summary&R%3A%3Ascript=simulate.S&showAttribution=TRUE`

- In the second case, they arrive as

```
-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="values"


-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="dataFile";
filename="bootData"
Content-Type: application/octet-stream

102
104
106
108

-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="statistic"

median
-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="NumRep"

1000
-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="output"

histogram
-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="output"

summary
-----------------------------8233788401435426128966544303
Content-Disposition: form-data; name="R::script"

simulate.S
-----------------------------8233788401435426128966544303--
```

- In addition to the different formats, where the data are located is different for GET and POST forms.
- For action = GET, the data are given in an environment variable named QUERY_STRING.
- For action = POST, they are available from standard input.
- CGIwithR hides these details and just presents formData.

Debugging CGI scripts

- Difficult since not running interactively.
- Get the script running within a regular R session, e.g. as a call to a function first
- Print diagnostic information to standard output so it goes into the resulting document.
- Or cat(“some message”, file = stderr()) which sends it to the Web server’s log file.
- Run
  ```
  tail -f logs/error_log
  ```
  in a separate window when submitting the request.

HTTP Error Log

```
[Mon Nov 28 06:02:02 2005] [error] [client 127.0.0.1] Error in cat(list(...), file, sep,
fill, labels, append) : , referer: http://localhost:9764/s141group1/htdocs/
bootstrapFile.html
[Mon Nov 28 06:02:02 2005] [error] [client 127.0.0.1] \targument 3 not yet handled by
cat, referer: http://localhost:9764/s141group1/htdocs/bootstrapFile.html
[Mon Nov 28 06:02:02 2005] [error] [client 127.0.0.1] Execution halted, referer: http://
localhost:9764/s141group1/htdocs/bootstrapFile.html
[Mon Nov 28 06:02:03 2005] [error] [client 127.0.0.1] File does not exist: /Users/
s141http/httpd/htdocs/favicon.ico
[Mon Nov 28 06:03:08 2005] [error] [client 127.0.0.1] Error: object "verbose" not found,
referer: http://localhost:9764/s141group1/htdocs/bootstrapFile.html
[Mon Nov 28 06:03:08 2005] [error] [client 127.0.0.1] Execution halted, referer: http://
localhost:9764/s141group1/htdocs/bootstrapFile.html
[Mon Nov 28 06:03:08 2005] [error] [client 127.0.0.1] File does not exist: /Users/
s141http/httpd/htdocs/favicon.ico
[Mon Nov 28 06:16:13 2005] [error] [client 127.0.0.1] File does not exist: /Users/
s141http/httpd/htdocs/favicon.ico
[Mon Nov 28 06:16:43 2005] [error] [client 127.0.0.1] File does not exist: /Users/
s141http/httpd/htdocs/favicon.ico
[Mon Nov 28 06:19:03 2005] [error] [client 127.0.0.1] Error: object "verbose" not found,
referer: http://localhost:9764/s141group1/htdocs/bootstrapFile.html
[Mon Nov 28 06:19:04 2005] [error] [client 127.0.0.1] File does not exist: /Users/
s141http/httpd/htdocs/favicon.ico
```

Debugging

- Post-mortem debugging via dump.frames() and debugger() functions.
- In the script, arrange to have the state dumped when there is an error
  ```
  options(error = quote({dump.frames("/tmp/group1/last.dump", to.file = TRUE); q()}))
  ```
- If there is an error, load the rda file and debug:
  ```
  load(“/tmp/group1/last.dump.rda”)
  debugger(get(“/tmp/group1/last.dump.rda”)
  ```
- May give more information.

Debugging

- You can save the formData object (e.g. save(formData, file = "/tmp/group1/formData.rda") and then source() the R script into an interactive R session.
- Not everything will be the same (.Rprofile, directory,...), but it will probably make things easier.
- But only for that particular data.

Debugging

- Avoid going through the Web browser in either of 2 ways
  - set environment variable FORM_DATA to text of query (POST or GET) and run

    ```
    R --vanilla < myScript.R
    ```
  - Use programmatic query, e.g. library(RCurl) and getForm() or postForm()

    ```
    getForm("http://localhost:9764/cgi-bin/simulation.R",
              n = "3", NumRep = "1000",
              statistic = "mean", distribution = "rnorm",
              output = "summary")
    ```

Security

- CGI scripts run on the server, not the user's client machine. So if a computation does something bad, it is on the server.
- A script can consume the computational resources of the machine, denying them to others - (DOS attack). So put limits on the computations, inputs, etc.
- If you allow arbitrary R commands to be run, the client can send highly destructive ones, intentionally or not.

Example

- E.g. in our simulation example, we allow the form writer to specify an R expressions for the statistic to simulate.
- I can hijack the HTML page and add my own R expression:

  ```
  function(x) {
      print(list.files("."))
      1
  }
  ```
  and now I can see the names of files that were supposed to be hidden from clients, or I can display their contents, or REMOVE them!

Basics of HTTP

- WWW and the Internet is based on many layers, each one is quite simple.
- HTTP - Hyper Text Transfer Protocol is the mechanism by which an application can communicate with a Web server to request a page.
- HTTP is a relatively primitive language for exchanging for conducting conversations about specific topics.
- Client is typically a Web browser, but can be any program that can talk HTTP.

Example HTTP session

- The following mimics what your Web browser does when you request a Web page.
- telnet www.omegahat.org 80

  connecting to host (omegahat) on port 80 which is where the Web server is by default

- GET /index.html HTTP/1.1
  Host: www.omegahat.org

  GET is the command, /index.html is the name of the file being requested.
  HTTP/1.1 is the particular dialect of HTTP being spoken and so the Web server knows to use that form.
  Specify Host again because a server can masquerade as many virtual hosts.
  Note the last line is a blank line that tells the Web server the request is complete.

The Server Reply

```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2005 14:26:46 GMT
Server: Apache/2.0.52 (Unix)
Last-Modified: Tue, 08 Nov 2005 17:59:29 GMT
ETag: "165f8b-447c-88782240"
Accept-Ranges: bytes
Content-Length: 17532
Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML//EN\">
<html> <head>
<title>The Omega Project for Statistical
Computing</title>
.............
```

Header Information

- There are lots of fields both the client and server can specify in the header to control the conversation.
- Keep-alive allows the client to ask the server to keep the connection open for period of time.
  This makes subsequent requests to that server faster.
- This improves efficiency when we, e.g., download a page and then go back to the server to fetch several images referenced in that page.
- Header information can also specify what type of file, character set encoding, version, part of a file, etc. we want.

Advanced HTTP

- GET is the simplest command.
  Specifies the resource we want.
- Forms posted with GET put the name=value parameters from the form into the filename in the GET command
  e.g. GET /cgi-bin/script?x=1&y=abc HTTP/1.1
- POST is another when we specify a request.
- POST puts the query into the BODY of the HTTP request.