

Server-Side Data

CSE 190 M (Web Programming), Spring 2008
University of Washington

Except where otherwise noted, the contents of this presentation are © Copyright 2008 Marty Stepp and Jessica Miller and are licensed under the Creative Commons Attribution 2.5 License.



Web data

- most interesting web pages revolve around data
 - examples: Google, IMDB, Digg, Facebook, YouTube, Rotten Tomatoes
 - can take many formats: text, HTML, XML, multimedia
- many of them allow us to access their data
- some even allow us to submit our own new data

URLs and web servers

`http://server/path/file`

- usually when you type a URL in your browser:
 - your computer looks up the server's IP address using DNS
 - your browser connects to that IP address and requests the given file
 - the web server software (e.g. Apache) grabs that file from the server's local file system, and sends back its contents to you
- some URLs actually specify *programs* that the web server should run, and then send their output back to you as the result:

`https://webster.cs.washington.edu/quote2.php`

- the above URL tells the server `webster.cs.washington.edu` to run the program `quote2.php` and send back its output

Server-Side web programming



- server-side pages are programs written using one of many web programming languages/frameworks
 - examples: PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl
- the web server contains plugins that allow it to run those programs and send back their output as responses to web requests
- each framework has its pros and cons
 - we will use PHP for server-side programming in this course

Parameterized programs

- most server-side web programs accept **parameters** that guide their execution
- uses for parameters:
 - specify which data is to be fetched (article #20598; calender for 4/11/08)
 - specify an action to be performed (display all events; show appointments in sorted order)
 - submit new data to the server (save this new appointment; delete all events)

Passing parameters (query strings)

```
http://www.google.com/search?q=colbert&ie=utf-8
```

- query string: a way of encoding parameters into a URL

```
http://server/path/program?query_string
```

- a query string has the following format:

```
field1=value1&field2=value2&field3=value3. . .
```

- preceded by a ?
- name=value pairs separated by &
- the above URL runs the program `search`, with parameter `q` set to `colbert` and the parameter `ie` set to `utf-8`
 - the program outputs the HTML search results

Web data example

- we have set up a program to retrieve student ASCIIimations:
 - the program is called `ascii.php`
 - on server `https://webster.cs.washington.edu` in the `/stepp/ajax` folder
 - accepts required parameter `name` specifying the student's last name
- example: to fetch Marty Stepp's ASCII, you'd fetch the URL:
`https://webster.cs.washington.edu/stepp/ajax/ascii.php?name=stepp`

Submitting data to a web server

- though web browsers mostly retrieve data from servers, sometimes they also want to send new data onto the server
- examples:
 - Hotmail: Send a message
 - Flickr: Upload a new photo
 - Google Calendar: Create a new appointment
- the data is sent in HTTP requests to the server
 - through Ajax
 - through XHTML **forms** (seen later)
- the data is placed into the request as parameters

GET requests and submitting data

An HTTP GET request is not an appropriate way to submit data to a web server.

- GET requests embed their parameters in their URLs
- URLs are limited in length (~ 1024 characters)
- URLs cannot contain special characters without URL-encoding them
 - example: space → %20
- private data in a URL can be seen or modified by users

HTTP GET vs. POST requests

Recall from our first lecture, HTTP allows several kinds of web requests:

- **GET** : asks a server for a page or data
 - if request has parameters, they are sent in the URL as a query string
- **POST** : submits data to a web server and retrieves the server's response
 - if request has parameters, they are embedded in the request packet, not the URL
- **PUT** : uploads an entire file to a web server
 - useful for large uploads such as image files and email attachments

For submitting data, a POST request is more appropriate than a GET.

Creating a POST request

```
new Ajax.Request(  
  "url",  
  {  
    method: "post", // optional  
    parameters: { name: value, name: value, ..., name: value },  
    onSuccess: functionName,  
    onFailure: functionName  
  }  
);
```

- Ajax.Request can also be used to post data to a web server
- method should be changed to "post" (or omitted; POST is default)
- any query parameters should be passed as a `parameters` parameter, written between `{ }` braces as `name: value` pairs
 - GET request parameters can also be passed this way, if you like

Practice problem: Submitting ASCII art

- Let's revisit the `ascii.php` example from a previous lecture.
- Suppose that the `ascii.php` service also accepts POST requests, where you can submit a new ASCIIimation into the system.
- The POST requires two parameters:
 - `name` : the last name of the student whose art is being submitted (String)
 - `ascii` : the text of the ASCIIimation to store (String)
- Modify our page to be able to both retrieve and submit ASCII art to this service.

Debugging Ajax Code

Finding and fixing problems when interacting with server-side data

Ajax code bugs

When writing Ajax programs, there are new kinds of bugs that are likely to appear.

- Nothing happens!
- The `responseText` or `responseXML` has no properties.
- The data isn't what I expect.

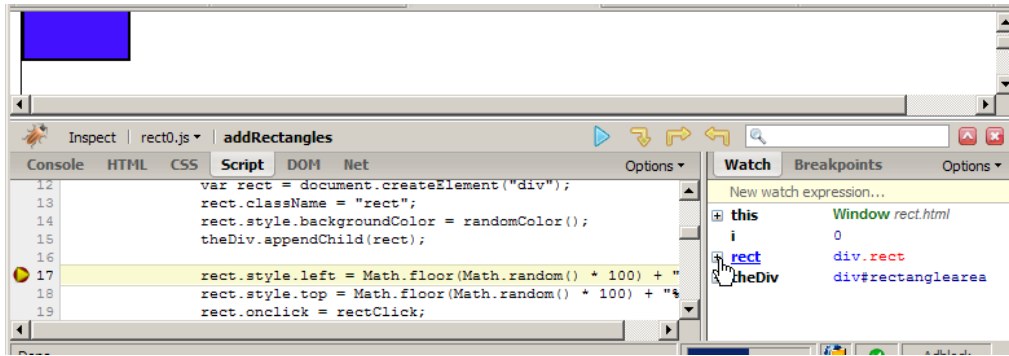
How do we find and fix such bugs?


Debugging in Firebug



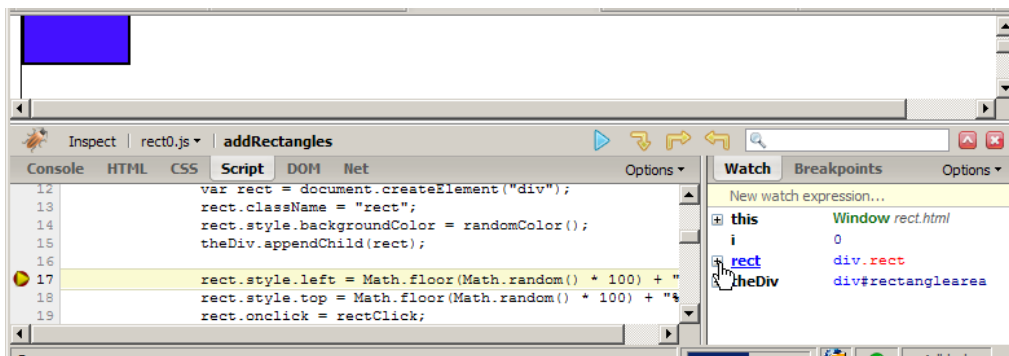
- open Firebug, click **Script** tab
- click to the left of a line to set a **breakpoint**
- refresh page
- when page runs, if it gets to that line in the JS code, program will halt





Breakpoints



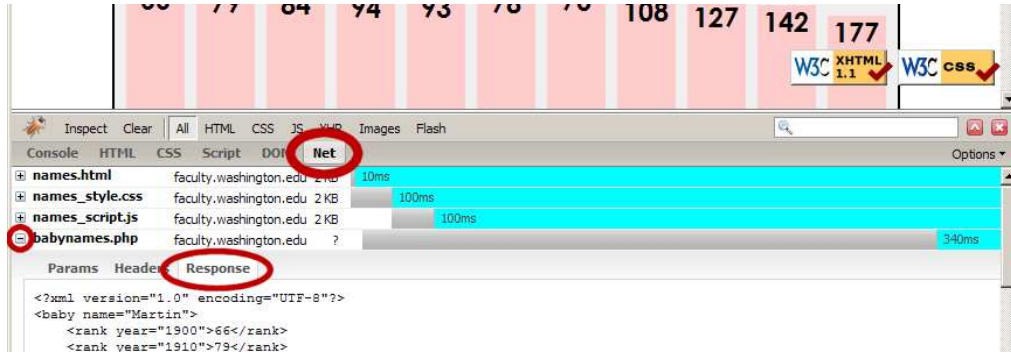
- **data:** once you've stopped at a breakpoint, you can examine any variables in the **Watch** tab at right
 - can click  to see properties/methods inside any object
 - **this** variable holds data about current object, or global data
 - if the object is global or not listed, type its name in the "New watch expression..." box
 - make sure Options → Show DOM Properties is checked, so you can see any DOM-related values


Stepping through code



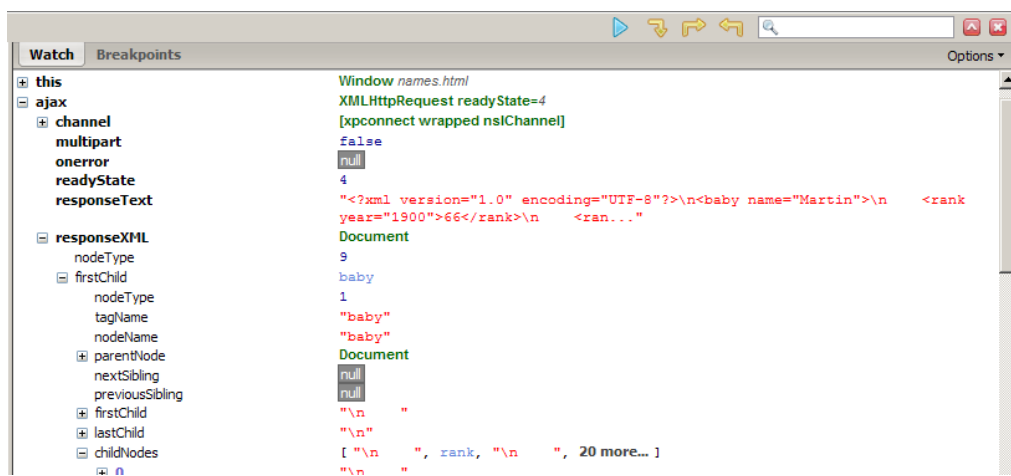
- **code:** once stopped at a breakpoint, you can continue execution:
 -  **continue** (F8): start the program running again
 -  **step over** (F10): run the current line of code completely, then stop again
 -  **step into** (F11): run the current line of code, but if it contains any calls to other methods, jump into those and stop
 -  **step out** (Shift-F11): run the current function to completion and return, then stop

Debugging Ajax code



- Net tab shows each request, its parameters, response, any errors
- expand a request with  and look at **Response** tab to see Ajax result

Debugging responseXML



- can examine the entire XML document, its node/tree structure