Asynchronous JavaScript + XML (Ajax)

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

References: w3schools, Wikipedia

Lecture outline

1. Ajax concepts
2. Using XMLHttpRequest
    - Prototype's Ajax facilities

Ajax concepts

web data sources, and the pages that love them

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
- today we'll learn ways to connect to web applications that serve data
- we'll also learn the **Ajax** technique for retrieving and displaying data on our web pages

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:

  ```
  http://science.slashdot.org/article.pl?sid=07/04/20/1651219
  ```

  - the above URL tells the server `science.slashdot.org` to run the program `article.pl` with certain 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 ASCIImations:
  - the program is called `get_ascii.php`
  - on server `faculty.washington.edu` in folder `/stepp/190m/`
  - accepts required parameter `name` specifying the student's UW NetID
  - accepts optional parameter `file` specifying the student's ASCIImation file name (if no value is passed, uses `asciimation.txt`).
- what URL will request `essigw`'s animation with default file?
- what URL will request `amylocke`'s animation with file name `asciianimation.txt`?

What is Ajax?

- Ajax: Asynchronous JavaScript + XML
- not a programming language; a way of using JS
- a way to download data from a server without reloading your page
- allows dynamically displaying data or updating the page without disturbing the user experience
- aids in the creation of rich, user-friendly web sites
  - the most excellent CSE 14x Diff Tool
  - other examples: Google Suggest, Facebook, Flickr, A9

Web applications

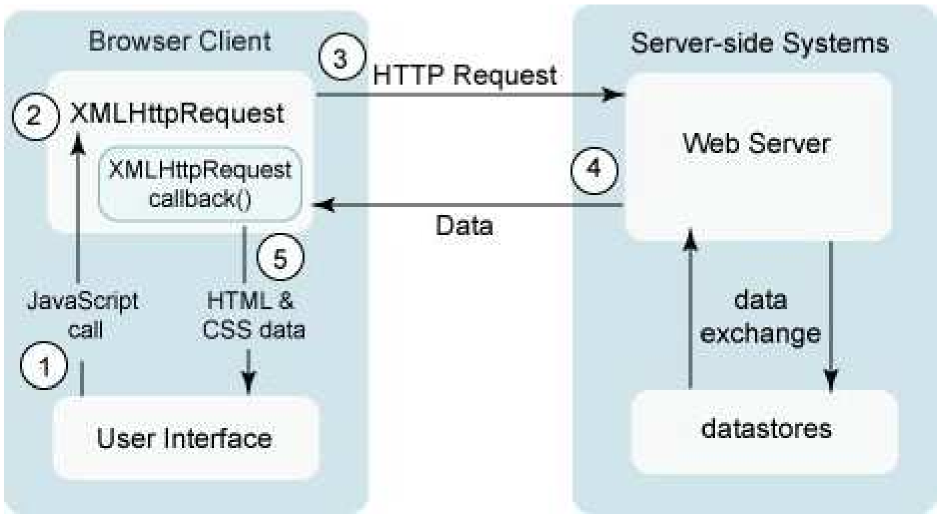
- **web application**: a web site that mimics the look, feel, and overall user experience of a desktop application
  - web app presents a continuous user experience rather than disjoint pages
  - as much as possible, "feels" like a normal program to the user
- some of Google's web apps
  - Gmail, Google Maps, Google Docs and Spreadsheets
- many web apps use Ajax to battle these problems of web pages:
  - slowness / lack of UI responsiveness
  - lack of user-friendliness
  - jarring nature of "click-wait-refresh" pattern

Core Ajax concepts

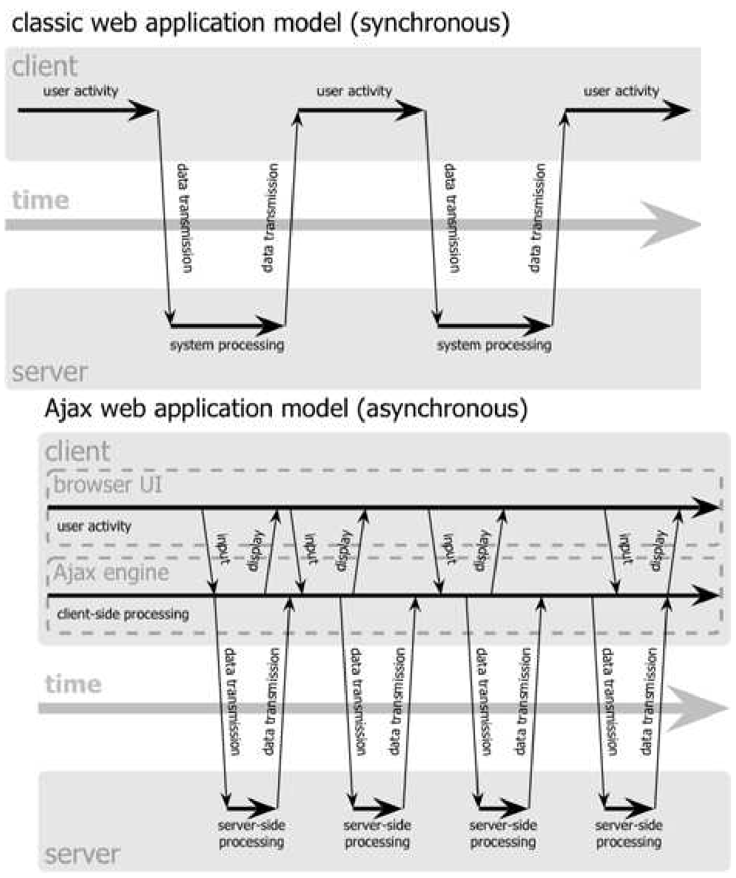
- JavaScript's `XMLHttpRequest` object can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera (with minor compatibilities)
- it can do this **asynchronously** (in the background, transparent to user)
- contents of fetched file can be put into current web page using DOM
- result: user's web page updates dynamically without a page reload

A typical Ajax request

1. user clicks, invoking event handler
2. that handler's JS code creates an `XMLHttpRequest` object
3. `XMLHttpRequest` object requests a document from a web server
4. server retrieves appropriate data, sends it back
5. `XMLHttpRequest` fires event to say that the data has arrived
   - this is often called a **callback**
   - you can attach a handler to be notified when the data has arrived
6. your callback event handler processes the data and displays it



Asynchronous communication



- **synchronous**: user must wait while new pages load
- **asynchronous**: user can keep interacting with page while data loads

The `XMLHttpRequest` object

the core JavaScript object that makes Ajax possible

- methods: `abort`, `getAllResponseHeaders`, `getResponseHeader`, **`open`**, **`send`**, `setRequestHeader`
- properties: **`onreadystatechange`**, `readyState`, **`responseText`**, `responseXML`, `status`, `statusText`
- IE6 sucks and requires a special `ActiveXObject` instead

Using XMLHttpRequest

Levels of using XMLHttpRequest

1. synchronized, text-only (SJAT?)
2. asynchronous, text-only (AJAT?)
3. asynchronous w/ Prototype (AJAP?)
4. asynchronous w/ XML data (Ajax ... seen next lecture)

1. Synchronized requests (bad)

```
// this code is in some control's event handler
var ajax = new XMLHttpRequest();
ajax.open("GET", url, false);
ajax.send(null);
do something with ajax.responseText;
```

- create the request object, open a connection, send the request
- when `send` returns, the fetched text will be stored in request's `responseText` property

Why synchronized requests suck

- your code waits for the request to completely finish before proceeding
- easier for you to program, but ...
- the user's *entire browser LOCKS UP* until the download is completed
- a terrible user experience (especially if the file is very large)

2. Asynchronous requests, basic idea

```
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function;
ajax.open("GET", url, true);
ajax.send(null);
// don't process ajax.responseText here, but in your function
```

- attach an event handler to the `onreadystatechange` event
- handler will be called when request state changes, e.g. finishes
- *function* contains code to run when request is complete

The `readyState` property

- holds the status of the `XMLHttpRequest`
- possible values for the `readyState` property:

State	Description
0	not initialized
1	set up
2	sent
3	in progress
4	complete

- `readyState` changes → `onreadystatechange` handler runs
- usually we are only interested in `readyState` of 4 (complete)

Asynchronous `XMLHttpRequest` template

```
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function() {
  if (ajax.readyState == 4) {   // 4 means request is finished
    do something with ajax.responseText;
  }
};
ajax.open("GET", url, true);
ajax.send(null);
```

- most Ajax code uses an **anonymous function** as the event handler
  - useful to declare it as an inner anonymous function, because then it can access surrounding local variables (e.g. `ajax`)

What if the request fails?

```
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function() {
  if (ajax.readyState == 4) {
    if (ajax.status == 200) {   // 200 means request succeeded
      do something with ajax.responseText;
    } else {
      code to handle the error;
    }
  }
};
ajax.open("GET", url, true);
ajax.send(null);
```

- web servers return status codes for requests (200 means Success)
- you may wish to display a message or take action on a failed request

Prototype's Ajax model

```
new Ajax.Request(
  "url",
  {
    option : value,
    option : value,
    ...
    option : value
  }
);
```

- Prototype's `Ajax.Request` object constructor accepts 2 parameters:
  1. the **URL** to fetch, as a String,
  2. a set of **options**, as an array of key:value pairs in `{}` braces
- hides some of the icky details (`onreadystatechanged`, etc.)
- works in IE, FF, etc.

Prototype Ajax methods and properties

- options that can be passed to the `Ajax.Request` constructor:
  - **`method`** : how to fetch the request from the server (default `"post"`)
  - **`parameters`** : query parameters to pass to the server, if any
  - `asynchronous` (default `true`), `contentType`, `encoding`, `requestHeaders`
- events in the `Ajax.Request` object that you can handle:
  - **`onSuccess`** : request completed successfully
  - **`onFailure`** : request was unsuccessful
  - `onCreate`, `onComplete`, `onException`, `on###` (handler for HTTP error code ###)

A more typical Prototype Ajax template

```
  new Ajax.Request(
    "url",
    {
      method: "get",
      onSuccess: functionName
    }
  );
  ...

function functionName(ajax) {
  do something with ajax.responseText;
}
```

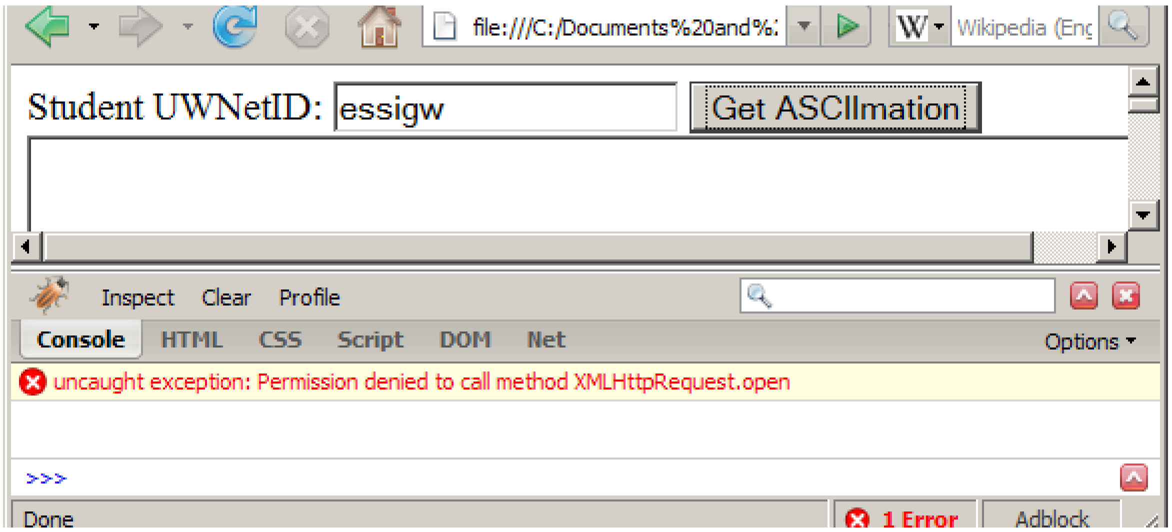
- most Ajax requests we'll do in this course are GET requests
- attach a handler to the request's `onSuccess` event
- the handler accepts the `XMLHttpRequest` object, `ajax`, as a parameter

Handling Ajax errors w/ Prototype

```
  new Ajax.Request(
    "url",
    {
      method: "get",
      onSuccess: functionName,
      onFailure: ajaxFailure
    }
  );
  ...
function ajaxFailure(ajax) {
  alert("Error making Ajax request to URL:\n" + url +
        "\n\nServer status:\n" + ajax.status + " " + ajax.statusText +
        "\n\nServer response text:\n" + ajax.responseText);
}
```

- for user's (and developer's) benefit, show a message if a request fails
- a good failure message shows the HTTP error code and status text

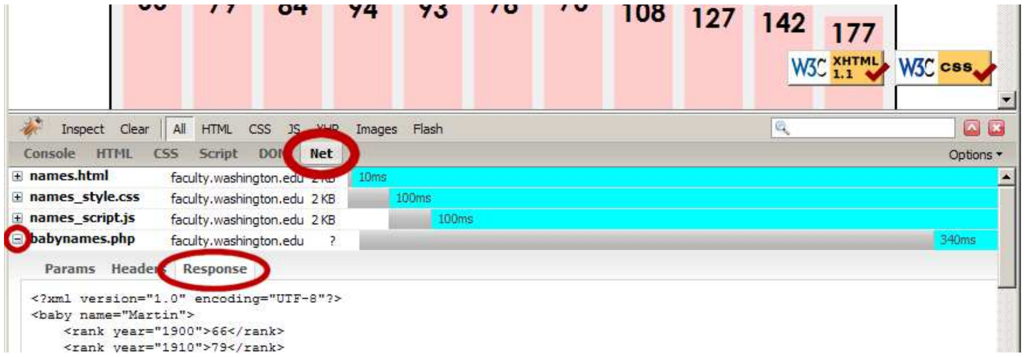
`XMLHttpRequest` security restrictions



uncaught exception: Permission denied to call method XMLHttpRequest.open

- cannot be run from a web page stored on your hard drive
- can only be run on a web page stored on a web server
- can only fetch files from the same site that the page is on
  - `www.foo.com/a/b/c.html` can only fetch from `www.foo.com`

Debugging Ajax code



names.html	faculty.washington.edu	2 KB	10ms
names_style.css	faculty.washington.edu	2 KB	100ms
names_script.js	faculty.washington.edu	2 KB	100ms
babynames.php	faculty.washington.edu	?	340ms

```
<?xml version="1.0" encoding="UTF-8"?>
<baby name="Martin">
    <rank year="1900">66</rank>
    <rank year="1910">79</rank>
```

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