1

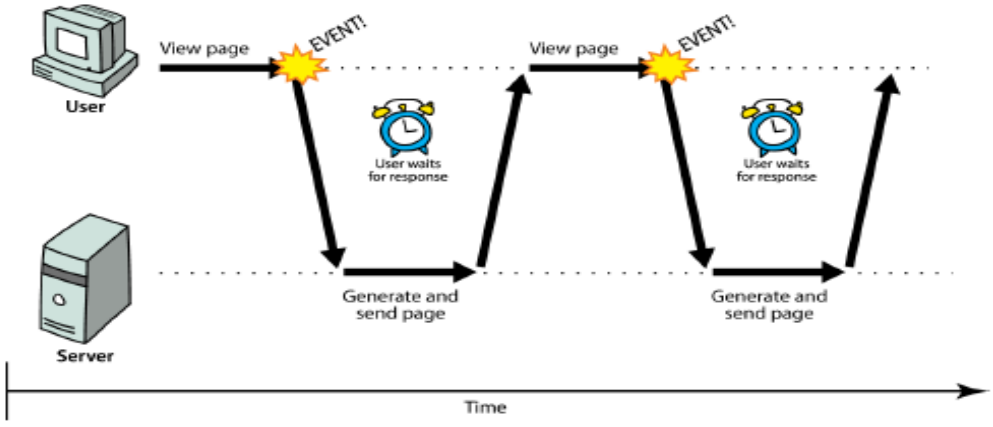
CSC 443: Web Programming

Haidar Harmanani

Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

Synchronous web communication

2



- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

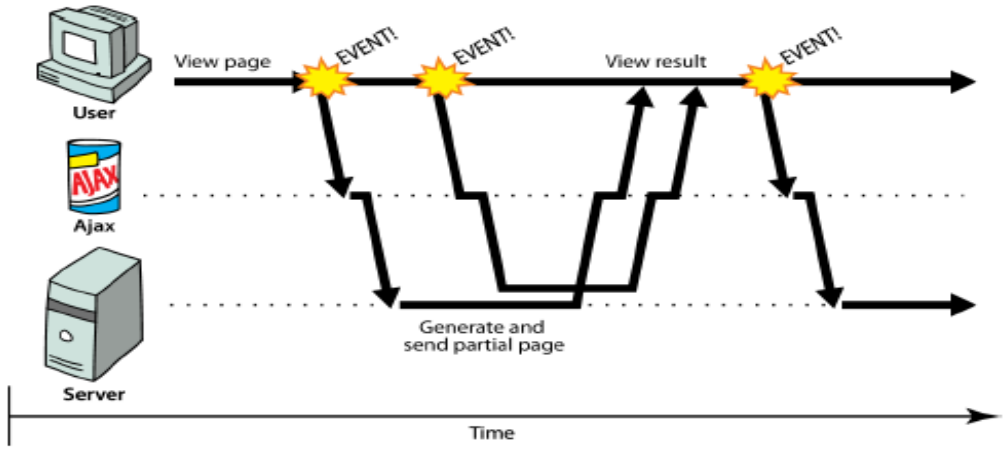
Web applications and Ajax

- **web application**: a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9

Web applications and Ajax

- **Ajax**: Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page without making the user wait
  - avoids the "click-wait-refresh" pattern
  - Example: Google Suggest

Asynchronous web communication



- **asynchronous**: user can keep interacting with page while data loads
  - communication pattern made possible by Ajax

`XMLHttpRequest` (and why we won't use it)

- JavaScript includes an XMLHttpRequest object that can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- it can do this asynchronously (in the background, transparent to user)
- the contents of the fetched file can be put into current web page using the DOM

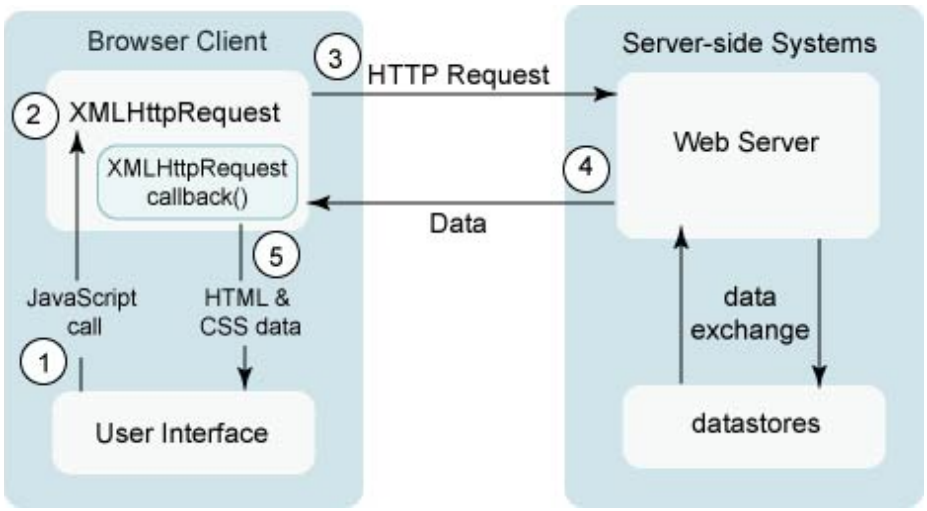
`XMLHttpRequest` (and why we won't use it)

- sounds great!...
- ... but it is clunky to use, and has various browser incompatibilities
- Prototype provides a better wrapper for Ajax, so we will use that instead

A typical Ajax request

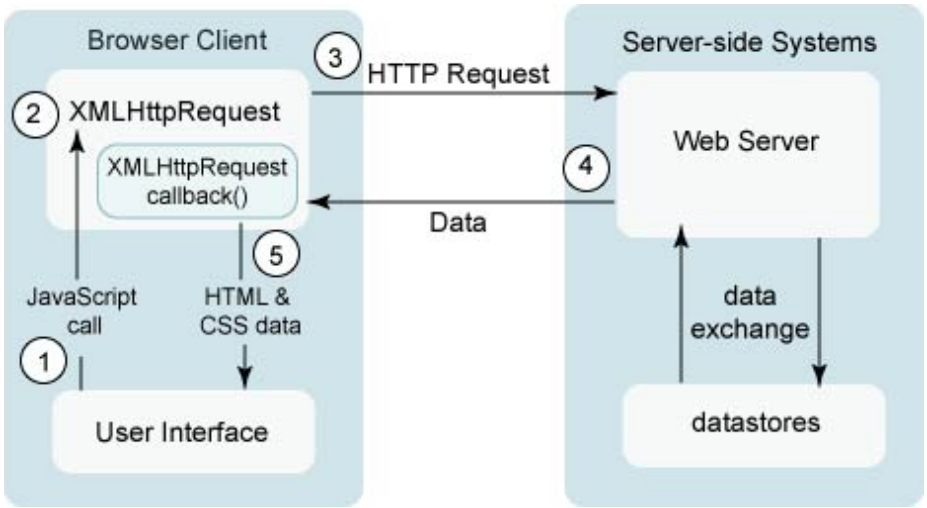
1. user clicks, invoking an event handler
2. handler's code creates an `XMLHttpRequest` object
3. `XMLHttpRequest` object requests page from server
4. server retrieves appropriate data, sends it back
5. `XMLHttpRequest` fires an event when data arrives
   - this is often called a callback
   - you can attach a handler function to this event
6. your callback event handler processes the data and displays it

A typical Ajax request



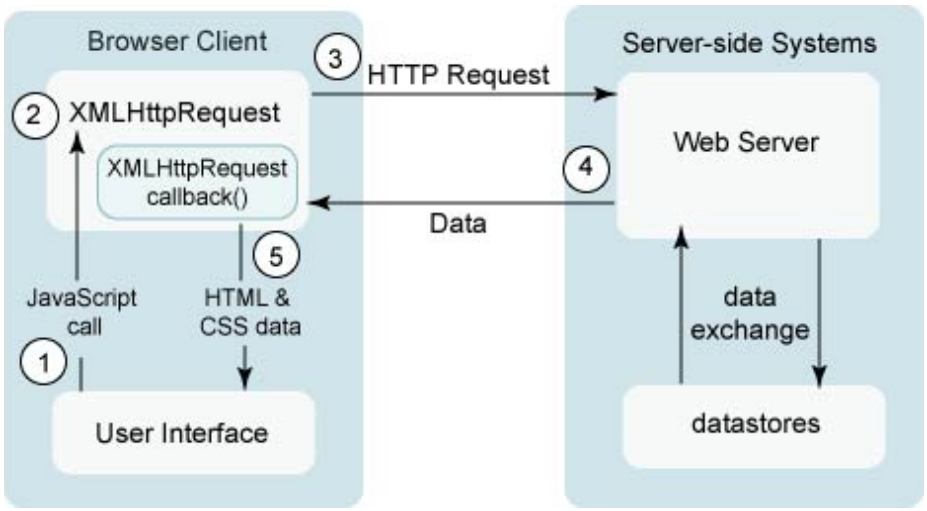
1. user clicks, invoking an event handler

A typical Ajax request



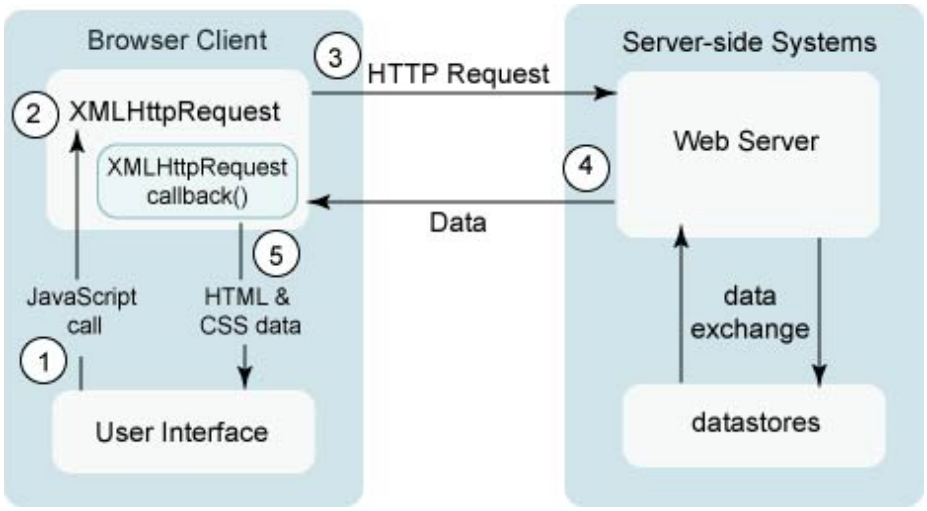
2. handler's code creates an XMLHttpRequest object

A typical Ajax request



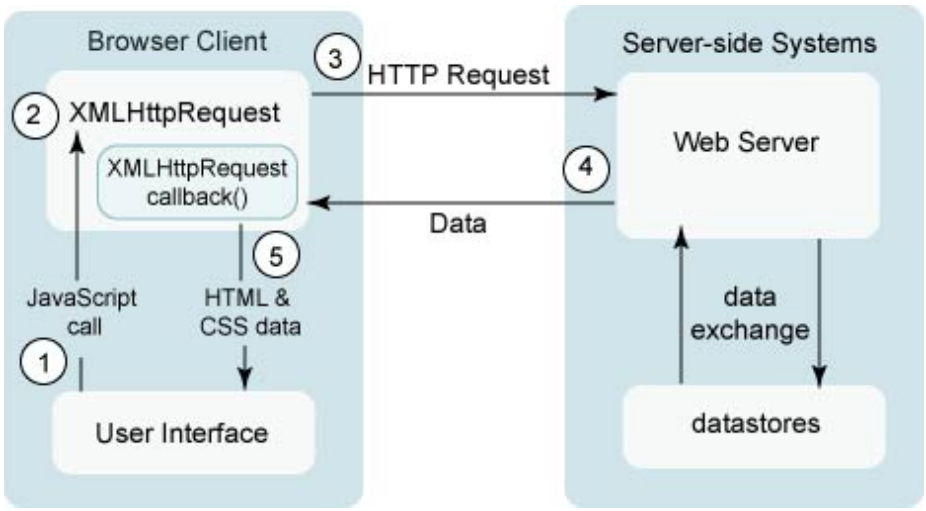
3. XMLHttpRequest object requests page from server

A typical Ajax request



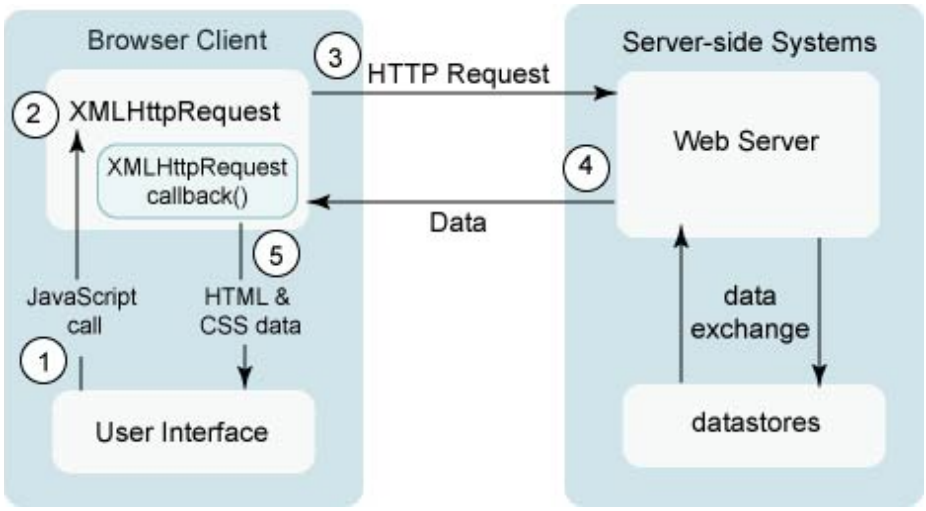
4. server retrieves appropriate data, sends it back

A typical Ajax request



5. XMLHttpRequest fires an event when data arrives
   - A callback to which you can attach a handler function

A typical Ajax request



6. your callback event handler processes the data and displays it

XMLHttpRequest methods

- the core JavaScript object that makes Ajax possible

Method	Description
open(*method*, *url*, *async*)	specifies the URL and HTTP request method
send() send(*postData*)	sends the HTTP request to the server, with optional POST parameters
abort()	stops the request
getAllResponseHeaders(), getResponseHeader(*name*), setRequestHeader(*name*,*value*)	for getting/setting raw HTTP headers

XMLHttpRequest properties

Property	Description
responseText	the entire text of the fetched page, as a string
responseXML	the entire contents of the fetched page, as an XML document tree (seen later)
status	the request's HTTP status code (200 = OK, etc.)
statusText	HTTP status code text (e.g. "Bad Request" for 400)
timeout	how many MS to wait before giving up and aborting the request (default 0 = wait forever)
readyState	request's current state *(0 = not initialized, 1 = set up, 2 = sent, 3 = in progress, 4 = complete)*

1. Synchronized requests (bad)

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

- 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 page is very large or slow to transfer)
- **Better solution**
  - **Use an asynchronous request that notifies you when it is complete**
  - **This is accomplished by learning about the event properties of `XMLHttpRequest`**

XMLHttpRequest events

Event	Description
load	occurs when the request is completed
error	occurs when the request fails
timeout	occurs when the request times out
abort	occurs when the request is aborted by calling abort()
loadstart, loadend, progress, readystatechange	progress events to track a request in progress

2. Asynchronous requests, basic idea

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", url, true);
ajax.send();
...
function functionName() {
  do something with this.responseText;
}
```
JS

- attach an event handler to the load event
- handler will be called when request state changes, e.g. finishes
- function contains code to run when request is complete
  - inside your handler function, this will refer to the ajax object
  - you can access its `responseText` and other properties

What if the request fails?

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

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

Handling the error event

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.onerror = errorFunctionName;
ajax.open("GET", "url", true);
ajax.send();
...
function functionName(e) {
  do something with e, this.status, this.statusText, ...
}
```
JS

- the graceful way to handle errors is to listen for the error event
- the handler is passed the error/exception as a parameter
- you can examine the error, as well as the request status, to determine what went wrong

Example Ajax error handler

```
var ajax = new XMLHttpRequest();
...
ajax.onerror = ajaxFailure;
...

function ajaxFailure(exception) {
  alert("Error making Ajax request:" +
        "\n\nServer status:\n" + this.status + " " + this.statusText +
        "\n\nServer response text:\n" + this.responseText);
  if (exception) {
    throw exception;
  }
}
```
JS

- for user's (and developer's) benefit, show an error message if a request fails

Passing query parameters to a request

- to pass parameters, concatenate them to the URL yourself
  - you may need to URL-encode the parameters by calling the JS encodeURIComponent(string) function on them
  - won't work for POST requests (see next slide)

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", "url?name1=value1&name2=value2&...", true);
ajax.send();
```
JS

Creating a POST request

```js
var params = new FormData();
params.append("name", value);
params.append("name", value);

var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("POST", "url", true);
ajax.send(params);
```

- use a FormData object to gather your POST query parameters
- pass the FormData to the request's send method
- method passed to open should be changed to "POST"

Ajax: The jQuery Way!

- jQuery provides a nice wrapper around AJAX
  - Similar to Prototype
- `jQuery.ajax( url [, options] )`
  - Perform an asynchronous HTTP (Ajax) request

jQuery and AJAX: Options

- url: A string containing the URL to which the request is sent
- type: The type of request to make, which can be either “POST” or “GET”
- data: The data to send to the server when performing the Ajax request
- success: A function to be called if the request succeeds
- accepts: The content type sent in the request header that tells the server what kind of response it will accept in return
- dataType: The type of data expected back from the server
- error: A function to be called if the request fails
- async: Set this options to false to perform a synchronous request
- cache: Set this options to false to force requested pages not to be cached by the browser
- complete: A function to be called when the request finishes (after success and error callbacks are executed)
- contents: An object that determines how the library will parse the response
- contentType: The content type of the data sent to the server
- password: A password to be used with XMLHttpRequest in response to an HTTP access authentication request
- statusCode: An object of numeric HTTP codes and functions to be called when the response has the corresponding code
- timeout: A number that specifies a timeout (in milliseconds) for the request

jQuery and AJAX: AjaxEvents

- .ajaxComplete(): Register a handler to be called when Ajax requests complete.
- .ajaxError(): Register a handler to be called when Ajax requests complete with an error.
- .ajaxSend(): Attach a function to be executed before an Ajax request is sent
- .ajaxStart(): Register a handler to be called when the first Ajax request begins
- .ajaxStop(): Register a handler to be called when all Ajax requests have completed.
- .ajaxSuccess(): Attach a function to be executed whenever an Ajax request completes successfully

jQuery and AJAX: Methods

- jQuery.ajax(): Perform an asynchronous HTTP (Ajax) request.
  - .load(): Load data from the server and place the returned HTML into the matched element
  - jQuery.get(): Load data from the server using a HTTP GET request.
  - jQuery.post(): Load data from the server using a HTTP POST request.
  - jQuery.getJSON(): Load JSON-encoded data from the server using a GET HTTP request.
  - jQuery.getScript(): Load a JavaScript file from the server using a GET HTTP request, then execute it.
  - .serialize(): Encode a set of form elements as a string for submission.
  - .serializeArray(): Encode a set of form elements as an array of names and values.

jQuery and AJAX


```
$.ajax({
   url: "someURL.php",
   type: "POST",
   data: {},              // data to be sent to the server
   dataType: "xml"
   }).done(function(data) {
      // Do stuff with data
   }).fail(function(xhr, status) {
      // Respond to an error
});
```

Ajax: GET and POST

- Similar to Prototype
  - $.get(URL, callback);
  - $.post(URL, data, callback);

```
var myURL = "someScript.php"; // or some server-side script
$.post(
myURL, // URL of script
{ // data to submit in the form of an object
name: "John Smith",
age: 433
},
function(data, status) { ... } // callback function
);
```


jQuery and AJAX


```
$("button").click(function(){
    $.post("demo_test.asp", function(data, status){
        alert("Data: " + data + "\nStatus: " + status);
    });
});

$(document).ajaxSuccess(function(){
    alert("AJAX request successfully completed");
});
```

jQuery and AJAX

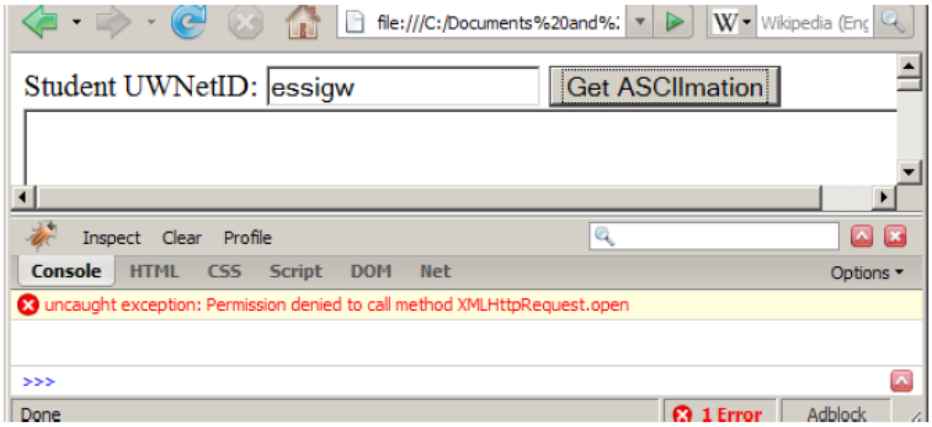

```
$.ajax({
   url: 'http://api.joind.in/v2.1/talks/10889',
   data: {
      format: 'json'
   },
   error: function() {
      $('#info').html('<p>An error has occurred</p>');
   },
   dataType: 'jsonp',
   success: function(data) {
      var $title = $('<h1>').text(data.talks[0].talk_title);
      var $description = $('<p>').text(data.talks[0].talk_description);
      $('#info')
         .append($title)
         .append($description);
   },
   type: 'GET'
});
```


AJAX

- $(selector).load(URL, data, callback)

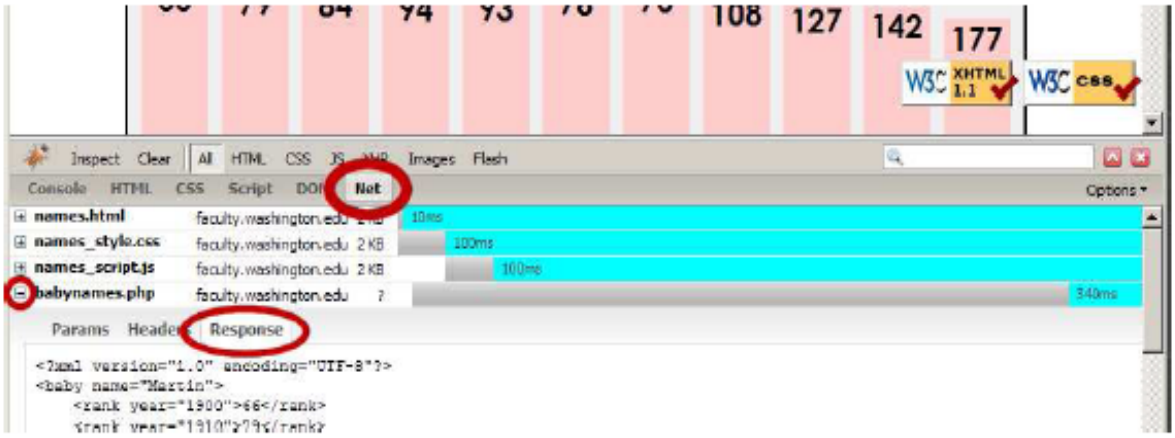
```
var myURL = "http://www.mysite.com/myFile.txt";
$("#myButton").click( function() {
   // Pass in the URL and a callback function.
   // xhr is the XMLHttpRequest object.
   $("#myDiv").load(myURL, function(response, status, xhr) {
      if(status == "success")
         alert(response);
      else if(status == "error")
         alert("Error: " + xhr.statusText);
   });
});
```

XMLHttpRequest security restrictions



- 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



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