CS 696 Emerging Web and Mobile Technologies
Spring Semester, 2011
Doc 13 Ajax & Node.js
Mar 8, 2011

References

jQuery Documentation, http://api.jquery.com/jQuery.get/

Ajax Programming, http://en.wikipedia.org/wiki/Ajax_(programming)

XMLHttprequest, http://en.wikipedia.org/wiki/XMLHttpRequest

Node.js Documentation, http://nodejs.org/docs/v0.4.2/api/

Up and Running with Node.js, Tom Hughes-Croucher, http://ofps.oreilly.com/titles/9781449398583/index.html

Wikipedia, as noted on individual slides

Data

Many Web and Mobile Apps require data from server

PhoneGap does not provide direct access to network socket

How to get data from server

Ajax

Asynchronous JavaScript and XML

Initially Ajax used:

HTML and CSS

Presentation

DOM

Dynamic display of data

XML

Encoding data

XMLHttpRequest

Asynchronous communication

JavaScript

Program logic

http://en.wikipedia.org/wiki/Ajax_(programming)

XMLHttpRequest

Makes http/https request to server without loading new page

Used to get

- Web page (html)
- Media (images, sound, video)
- Data (xml, json)
- code (scripts)

JSON - JavaScript Object Notation

Created as alternative to XML for transmitting data between server and web browser

JSON parsers exist in over 40 languages

JSON data types

true
false
null
"strings"
12.345 (numbers)
12
12.3e12
arrays
objects

array
[12, "cat", true]
["this", "is", "an array"]

object
{"key":"value"}
{"name":"Roger", "age": 12}
{"colors: ["red","black", "blue"]}

JQuery XMLHttpRequest Shortcuts

Supports get and post

jQuery.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])

url - were to send the request
data - map or string sent to server
success(data, textStatus, jqXHR) - function called with result of request
data - data returned by server
textStatus -
jqXHR - superset of XMLHttpRequest object

dataType - type of data requested, optional
xml, json, script, html

Equivalent Versions

jQuery.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])

$.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])

```
$.ajax({
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

Example

```
$.get('ajax/test.html', function(data) {
  $('.result').html(data);
  alert('Load was performed.');
});
```

Sample Requests & Server

```
$.get('http://127.0.0.1:8000/test', "cat");
```

http://127.0.0.1:8000/test?cat

```
$.get('http://127.0.0.1:8000/test', { name: "John", time: "2pm" });
```

http://127.0.0.1:8000/test?name=John&time=2pm

```
$.get('http://127.0.0.1:8000/test', { 'choices[]': ["Jon", "Susan"]} );
```

http://127.0.0.1:8000/test?choices%5B%5D=Jon&choices%5B%5D=Susan

Tuesday, March 8, 2011

%5B%5D = [] but is url encoded

Server - Handling Requests

There are many different systems to handle web requests

- php
  - 18+ frameworks
- .net
  - 7+ frameworks
- perl
  - 6+ frameworks
- Java
  - 34+ frameworks
- Ruby
  - 6+ frameworks
- Python
  - 15+ frameworks
- Seaside (Smalltalk)
- Kepler (Lua)
- Lift (Scala)
- Nitrogen (Erlang)
- etc

http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Node.js

http://nodejs.org/

JavaScript on desktop/server side

Event-driven non-blocking I/O framework

Scalable network programs

Uses Googles V8 JavaScript Engine

Compiles JavaScript to machine code

Used in HP's WebOS Phones and tablets

Up and Running with Node.js

Early draft of Node.js book by Tom Hughes-Croucher

http://ofps.oreilly.com/titles/9781449398583/index.html

Delays in I/O

To read a file with blocking I/O

Get File descriptor from file name
"Open" file for reading
Wait for hard drive head to get to correct location
Wait as hard drive spins to read file contents
May require multiple movement of hard drive head

While this happens your code waits

Threads

Common way to scale performance

While one thread is blocked on I/O another thread can perform work

Typical server

- One high priority thread accepts connects from clients
- Once accepted a client connection is give to a worker thread

Thread Issues

- Overhead
  - Memory
  - Time in context switches
  - Managing threads

- Programming issues
  - Communication between threads
  - Deadlock
  - Livelock
  - Multiple threads accessing same data

Node.js

To be highly scalable it does not use:

- Threads*
- Blocking I/O

Instead uses callbacks

- When OS has data for you to read you callback function is called

*Well it does have proccesses and will use WebWorkers.

Reading a File

```
var fs = require('fs');

function processFileContents(error, data) {
  if (error) throw error;
  console.log(data);
}

fs.readFile('Client.html','utf8', processFileContents );
```


Node.js does support synchronous reading of files

Second Example - Reading Chunks

```
var fs = require('fs');

var spool = "";

function processFileContents( data) {
  spool += data;
}

function doneReading( ) {
  console.log(spool);
}

var fileStream = fs.createReadStream('Client.html',{encoding: 'utf8'} );
fileStream.on('data', processFileContents);
fileStream.on('end' , doneReading);
```


Don't block the event Loop. Depending on what one is doing it maybe better to use Buffer for better performance.

Simple WebServer

```
var http = require('http');
var server = http.createServer()
server.on('request', handleRequest);
server.listen(8000, "127.0.0.1");

console.log('Server running at http://127.0.0.1:8000/');

function handleRequest(request, response) {
    console.log(request.url);
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end(data);;
}
```

Simple WebServer - Standard Example

```
var http = require('http');

http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end('Hello World\n');
}).listen(8000, "127.0.0.1");

console.log('Server running at http://127.0.0.1:8000/');
```


Same as previous slide. This is the standard Node.js example but is a bit more compact and harder to read

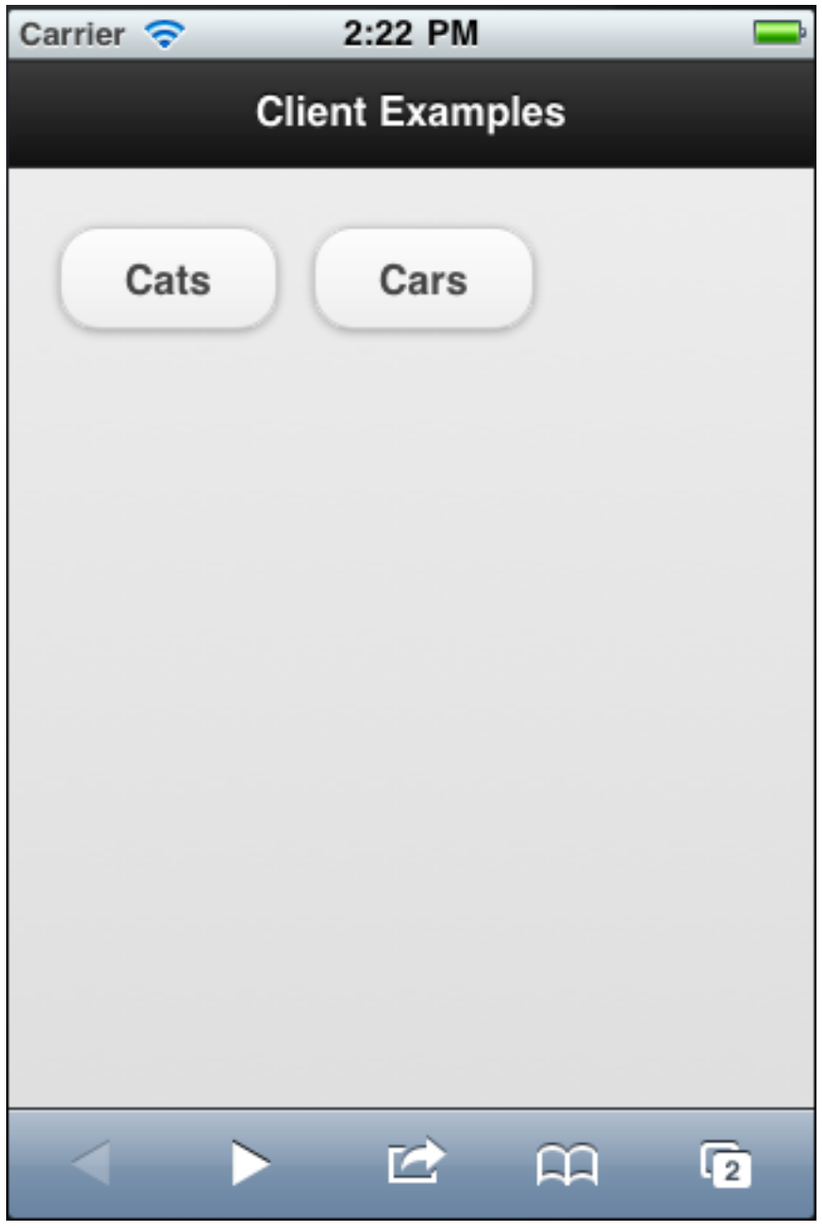
Complete Example

Use clicks on "Cats" or "Cars" button

Connect to server to get list of "Cats" or "Cars"

Add list to next page

Display new page with list



HTML

```
<body>

<div data-role="page" >
	<div data-role="header">
		<h1>
			Client Examples
		</h1>
	</div>
	<div data-role="content">
		<a href="#result" data-role="button" data-inline="true" onclick="sendData('cats')">Cats</a>
		<a href="#result" data-role="button" data-inline="true" onclick="sendData('cars')">Cars</a>
	</div>

</div>
<div data-role="page" id="result">
	<div data-role="header">
		<h1>Requested Info</h1>
	</div>
	<div data-role="content">
		<div id="data">
		</div>
	</div>

</div>
```

Client Side JavaScript

```
<script src="http://code.jquery.com/jquery-1.5.min.js"></script>
<script src="http://code.jquery.com/mobile/1.0a3/jquery.mobile-1.0a3.min.js"></script>
<script>
    function sendData(data) {
        $.get('http://127.0.0.1:8000', {type:data}, displayData, 'json');
    }

    function displayData(data,text,unused) {
        $("#data").html(arrayToList(data));
    }

    function arrayToList(array) {
        var startList = '<ul data-role="listview" data-theme="a"><li>';
        var endList = '</li></ul>';
        var listContents = array.join("</li><li>");
        return startList + listContents + endList;
    }
</script>
```

Server

```
var http = require('http');
var server = http.createServer();
server.on('request', handleRequest);
server.listen(8000, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8000/');

function handleRequest(request, response) {
      var category = requestedCategory(request.url);
      var requestedData = dataFor(category);
      returnData(requestedData, response);
}

function returnData(data, response) {
  response.writeHead(200, {'Content-Type': 'application/json'});
  response.end(data);
}
```

Server Continued

```
function requestedCategory(url) {
	var urlParts = require('url').parse(url, true);
	return urlParts['query']['type'];
}

function dataFor(category) {
	if (category == 'cats')
		return '["Persian", "Siamese", "Siberian"]';
	if (category == 'cars')
		return '["Audi", "BMW", "Honda"]';
	return "[]";
}
```