Outline

• What Ajax is
• A few demos
• First steps programming an Ajax application
• Real programming: JavaScript doesn't suck
• Design patterns and libraries
• Where to learn more
What this talk is about

• What Ajax is, or means, for those who don't know (Hands, please? Anyone?)
• Why Ajax matters
• Serious programming in JavaScript
• Distributed programming (remoting) in general, because there are alternatives to Ajax
What this talk is not about

- Novice JavaScript-ing
- Stupid DHTML tricks
- Web Services
- Whether or not Ajax sux/rox/blows/...
- How to use Backbase or Echo2 or ...
- Jesse James Garrett
In case you didn't know

- Ajax is a snappy term given (last year) to a collection of technologies that have been around since the previous century.
- Ajax stands for **Asynchronous Javascript And XML**.
- The XML part isn't required... plain text and JSON are often used instead, but Ajat and Ajaj sound stupid.
Why Ajaxify?

• Because many web applications in which every action causes the whole freakin' page to refresh are annoying

• Because you can create rich client applications without users installing the applications

• Because the techniques are (1) cross-browser, (2) server technology-agnostic, (3) require no plugins, and (4) mostly use open standards!
DEMO TIME !!!

- Google Maps  http://maps.google.com
- Google Suggest  http://www.google.com/webhp?complete=1
- Rico Demos  http://openrico.org/rico/demos.page
- ajaxpattens.org Demos  http://www.ajaxify.com/run/
- script.aculo.us Demos  http://wiki.script.aculo.us/scriptaculous/show/Demos
What was behind those demos?

- Two basic things:
  1. The actual document object in the browser was being modified dynamically (those were not Flash movies, applets, or other plugins)
  2. The browser was making asynchronous requests and processing the server's responses (again, without the full page refresh)
Now do it yourself

<html>
    <head>
        <title>Gettin staitrit wi Ajax</title>
        <script type="text/javascript" src="basicajax.js"></script>
    </head>

    <body>
        <p>Whit's the
           <a href="javascript:fetchText('services/now.jsp', 'x')">time</a>?
           <span id="x"></span>.  Thenk ye.  Fare ye weel.</p>
    </body>
</html>
And the script...

function createXMLHttpRequest() {
    try {return new ActiveXObject("Msxml2.XMLHTTP");} catch (e) {} 
    try {return new ActiveXObject("Microsoft.XMLHTTP");} catch (e) {} 
    try {return new XMLHttpRequest();} catch (e) {} 
    return null;
}

function fetchText(url, id) {
    var xmlhttp = createXMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            document.getElementById(id).innerHTML = xmlhttp.responseText;
        }
    };
    xmlhttp.open('GET', url);
    xmlhttp.send(null);
}
Four defining principles of Ajax

- The browser hosts an application, not content
- The server delivers data, not content
- User interaction can be fluid and continuous
- This is real coding and requires discipline

from David Crane and Eric Pascarello,
Ajax in Action, Manning, 2006
Ajax programmers gotta know...

- HTTP, at least the basics
- XML and XHTML
- The DOM
- CSS
- JavaScript, really, really well

So let's spend some time on JavaScript...
JavaScript

• Small, elegant, very expressive, very powerful
• Flexible, dynamic (it's got eval)
• Closest to Lisp, nothing at all like Java
• OO done with prototypes, not classes
• "World's most misunderstood programming language"
• Based on ECMA-262 international standard
JavaScript programming

- Like all good scripting languages, not much boilerplate — just start coding:

```javascript
document.write("<b>Some cubes for " + today() + "</b>);
function today() {
    return new Date();
}
var n = 10;
for (i = 0; i < n; i++) {
    document.write("<br />" + cubed(i));
}
function cubed(x) {
    return x * x * x;
}
```
JavaScript data types

• There are only six
  – Undefined: only value is undefined
  – Null: only value is null
  – Boolean: only values are true and false
  – Number: e.g., 3, -9.872E31, 0x4C, NaN
  – String: e.g., "Hello", "Ol\u00e9"
  – Object: everything else
JavaScript objects

• Don't declare classes — just create objects and add properties on the fly:

```javascript
var x = new Object();
x.age = 17;
x.height = 65.3;
var y = x.age + x['height'];
var z = {age: 30, color: "red", total: 3};
var cool = {triple: {a: 4, b: undefined, c: {4: null}}, 7: "stuff"};

function midpoint(p, q) {
    return {x: (p.x+q.x)/2, y: (p.y+q.y)/2}
}
```
JavaScript arrays

• Arrays are just objects with properties that are integers:

```javascript
var a = new Array();
a[0] = 5 * a[i];
var b = new Array(10);  // initialized with 10 cells
var c = new Array(1, 5, 25); // initialized with 3 cells
var d = [1, 5, 25];
var e = [1, true, [1,2], {x:5, y:6}, "Hi"];
var f = {triple: {a:4, b:"dog", c:[1,null]}, 7: "stuff"};
var g = [f.triple.a, f[7]];
var y = a.length;
```
JavaScript functions

• A function is just an object. Three examples:

```javascript
function successor(x) {return x + 1;}
var sum = function(x,y) {return x + y;}
var predecessor = new Function("x", "return x - 1;" ); // slow
```

• Since a function is just an object it can
  – have properties that are other objects
  – be a property of another object (in which case it is called a method) — whoa, sounds like OOP...
  – be passed as an argument to another function
  – be returned as a value from another function
More JavaScript functions

- Functional programming rocks

```javascript
function plusSix(x) {return x + 6;}
function squared(x) {return x * x;}
function twice(f, x) {return f(f(x));}
assert(twice(plusSix, 5) == 17);
assert(twice(squared, 4) == 256);
function compose(f, g) {return function(x) {return g(f(x));}}
var squareThenPlusSix = compose(squared, plusSix);
assert(squareThenPlusSix(10) == 106);
assert(compose(plusSix, squared)(5) == 121);
document.write("twice expects "+ twice.length + " arguments");
```
If you call a function that is a property of an object, then within that function the expression "this" refers to the containing object.

```javascript
var x = {a: 1, b: function(x) {return x + this.a;}};
document.write(x.b(2)); // writes 3
```

"this" is evaluated dynamically, not statically:

```javascript
function f(x) {return x + this.a;};
var x = {a: 10, b: f};
document.write(x.b(2)); // writes 12
```
Prefixing a function call with "new" makes "this" refer to a new object instead:

```javascript
function Point() {
    this.x = 0;
    this.y = 0;
    this.move = function(dx, dy) {this.x += dx; this.y += dy;}
    this.reflect = function() {this.x = -this.x; this.y = -this.y;}
}

var p = new Point();  // Note the use of "new"
p.move(52, 40);
p.reflect();
document.write(p.x + " " + p.y);  // writes 12
```
Prototypes

● Oops, the code on the last slide sucked! Each new point gets a copy of the move and reflect functions. Here is the right way:

```javascript
function Point(x, y) {
    this.x = x || 0;
    this.y = y || 0;
}

Point.prototype.move = function(dx, dy) {this.x += dx; this.y += dy;}
Point.prototype.reflect = function() {this.x = -this.x; this.y = -this.y;}
```

● Also note the cool "default parameters" here.
Constructors, prototypes, instances

```javascript
var p = new Point(5, 8);
```

Don't worry, you get used to this.
Built-in JavaScript objects

Object Object.prototype Function Function.prototype Array
Array.prototype String String.prototype Boolean Boolean.prototype
Number Number.prototype Math Date Date.prototype RegExp
RegExp.prototype Error Error.prototype ...

Navigator Document Window Location Screen Event History

Anchor Area Applet Base Basefont Body Button Checkbox
FileUpload Form Frame Frameset Hidden Iframe Image Link
Meta Option Password Radio Reset Select Style Submit Table
TableData TableHeader TableRow Text TextArea
XMLHttpRequest

There's also a "global" object you don't refer to directly: its properties (e.g. eval, parseInt, decodeURI, Infinity, NaN, etc.) are used w/o qualification
### XMLHttpRequest properties

<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onreadystatechange</td>
<td>the callback (see next slide)</td>
</tr>
<tr>
<td>readystatechange</td>
<td></td>
</tr>
<tr>
<td>open(method, url, async)</td>
<td>connect</td>
</tr>
<tr>
<td>setRequestHeader(key, value)</td>
<td></td>
</tr>
<tr>
<td>send(content)</td>
<td>already parsed for you!</td>
</tr>
<tr>
<td>responseText</td>
<td>200 or 403 or 404 or ...</td>
</tr>
<tr>
<td>responseXML</td>
<td>&quot;OK&quot; or &quot;Forbidden&quot; or ...</td>
</tr>
<tr>
<td>status</td>
<td></td>
</tr>
<tr>
<td>statusText</td>
<td>returns a big ugly string</td>
</tr>
<tr>
<td>getResponseHeader(key)</td>
<td></td>
</tr>
<tr>
<td>getAllResponseHeaders()</td>
<td></td>
</tr>
<tr>
<td>abort()</td>
<td></td>
</tr>
</tbody>
</table>
Ready states

0 = uninitialized
1 = loading
2 = loaded
3 = interactive
4 = complete

Every time the ready state changes, your callback is called
Passing request parameters

- This is just one way (simple, not the best):

```html
<form onsubmit="javascript:postAndFetchText('services/temperature_converter.jsp', 'value=' + document.getElementById('value').value + '&units=' + document.getElementById('units').value, 'result'); return false;">
  <input id="value" width="20" />  
  <select id="units">
    <option>F</option><option>C</option><option>K</option>
  </select>
  <input type="submit" value="Convert" />
</form>
```
function postAndFetchText(url, content, id) {
var xmlhttp = createXMLHttpRequest();
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById(id).innerHTML = xmlhttp.responseText;
    }
};
xmlhttp.open('POST', url);
xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xmlhttp.send(content);
}
Getting XML back

- Suppose our temperature converter replies with things like this:
  - <error>Units must be c, k, or f</error>
  - <error>Bad temperature: 34fesf42</error>
  - <data><c>5</c><f>41</f><k>278.15</k></data>

- The responseXML property will contain a XML tree; navigate this the usual way
- We'll want to set a CSS style for the response based on the root element.
function postAndFetchXml(url, content, handler) {
    var xmlhttp = createXMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            handler(xmlhttp.responseXML);
        }
    }
    xmlhttp.open('POST', url);
    xmlhttp.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
    xmlhttp.send(content);
}
function temperatureHandler(xmlDoc) {
    var root = xmlDoc.documentElement.nodeName;
    var result = document.getElementById('result');

    if (root == 'error') {
        result.style.color = 'red';
        result.innerHTML = xmlDoc.documentElement.firstChild.nodeValue;
    } else if (root == 'data') { // (this could use some refactoring)
        var k = xmlDoc.getElementsByTagName("k")[0].firstChild.nodeValue;
        var c = xmlDoc.getElementsByTagName("c")[0].firstChild.nodeValue;
        var f = xmlDoc.getElementsByTagName("f")[0].firstChild.nodeValue;
        result.style.color = 'blue';
        result.innerHTML = "<ul><li>" + f + "&deg;F</li><li>" + c + "&deg;C</li><li>" + k + " K</li></ul>";

    ...
}
Organizing the application

<html>
<head>
    <title>Temperature Conversion</title>
    <script type="text/javascript" src="basicajax.js"></script>
    <script type="text/javascript" src="temperature.js"></script>
</head>

<body>
    <form onsubmit="javascript:postAndFetchXml(
        'services/temperature_converter.jsp',
        'value=' + $('value').value + '&units=' + $('units').value,
        temperatureHandler); return false;">
        Temperature: <input id="value" width="20" />
    </form>
    ...
</body>
Instead of plain text or XML, have the server send back JSON.... you can eval it directly

```json
{"menu": {
   "id": "file",
   "value": "File:",
   "popup": {
      "menuItem": [
         {
            "value": "New", "onclick": "CreateNewDoc()"
         },
         {
            "value": "Open", "onclick": "OpenDoc()"
         },
         {
            "value": "Close", "onclick": "CloseDoc()"
         }
      ]
   }
}}
```
When the application gets big...

- Make "packages"

```javascript
var acme = {
    version: 2.0,
    getReq: function() {...},
    getTextAsync(url, id, params): function {...},
    getXmlDocAsync(url, handler, params): function {...},
    ...
}
```

- There's more: Look online for ways to things like getting the effect of private, static, read-only, inheritance, etc.
This is tedious — help me!!!!

- JavaScript libraries (especially remoting help)
  - prototype, Sarissa, dojo, Mochikit
- Special effects
  - script.aculo.us, Rico, dojo, moo.fx, ...
- Full-scale application frameworks (may or may not make writing JavaScript unnecessary)
  - DWR, Echo2, BackBase, JSON-RPC, Ruby on Rails, Bindows, Atlas, Google Web Toolkit
prototype, the library

- Rico and script.aculo.us are built on prototype, and so are many other libraries and applications.
- Written by Sam Stephenson
- Read about it: http://prototype.conio.net/
- prototype.js is at http://prototype.conio.net/dist/prototype-1.4.0.js
- Check out the function called $ (yeeeeeaaah) 
  - $('x') is sorta like document.getElementById('x')
Mochikit

- JavaScript library
- Nice Python-like features (repr, not toString)
- Non-obtrusive (doesn't hack built-ins)
- Has unit tests and documentations
- Makes DOM manipulation easy as innerHTML
- Great event handling (custom events, too)
- Great logging
- Others (higher order fns, iters, dates, async...)
- Not a flashy widget library
The standard library JavaScript never had
Full featured

Custom Widgets
Widget System
UI
Event
Language Utilities
Package System (!)
Design patterns

- Real programmers know design patterns
- A design pattern is a named solution approach to a recurrent problem
- Title, problem, context, solution
- Idea credited to Christopher Alexander
- The GoF popularized them in the world of OOP in 1994 (abstract factory, observer, visitor, builder, state, flyweight, template method, strategy, singleton, ...)
Ajax design patterns

- http://www.ajaxpatterns.org
  - 8 foundational technology patterns
  - 25 programming patterns
  - 28 functionality and usability patterns
  - 8 development practices
- http://developer.yahoo.com/ypatterns
  - Lookup (get data when needed)
  - Persist (real-time saving)
  - Update (visual changes)
Example Patterns

- AJAX Stub
- Call Tracking
- On-demand JavaScript
- XML Data Island
- Submission Throttling
- Multistage Download
- Predictive Fetch
- Cross Domain Proxy
More Example Patterns

- Drilldown
- Microlink
- Live Form
- Live Search
- Progress Indicator
- Suggestion
- Drag and Drop
- Status Area
- Highlight
More On Programming

• Sometimes useful to do class-based inheritance ("classical OOP") in JavaScript
• Example approaches:
  – Iniside Stephenson's prototype library
  – Article on Douglas Crockford's site
  – Dean Edwards's Base
  – Daniel Romano's technique
  – Many others (Fuecks, Lindsey, Daniel, ...)

Romano's Classical Inheritance

```javascript
Function.prototype.extend = function(sub) {
    sub.prototype = new this;
    sub.prototype._super = this;
    sub.prototype.constructor = sub;
    return sub;
}

var Shape = function(c) {this.color = c;}
Shape.prototype.getColor = function() {return this.color;}
var Circle = Shape.extend(function(c, r) {this._super(c); this.radius = r;});
Circle.prototype.getRadius = function() {return this.radius;}
```
Ajax mistakes

- Like many technologies, Ajax is easy to misuse.
- The original "Ajax mistakes" blog post (as far as I know):
  
  http://alexbosworth.backpackit.com/pub/67688

  Let's check it out

- Search "Ajax mistakes" for more (on your own)
Performance

- Probably shouldn't rely on JS interpreter to optimize your code, so you might want to explicitly cache things
- JavaScript memory leaks come from holding on to references too long
  - Cyclic references can be common in Ajax
  - DOM Inspector, modi, Venkman debugger, Mochikit logging, can help
Alternatives to Ajax

- Flash
- IFRAMES
- Java Applets
- Java WebStart
- XUL
- XAML
- Others?

Article at http://www.ajaxinfo.com/default~viewart~8.htm
Conclusions

- Decent Ajax apps require serious programming
- JavaScript has a lot of power and flexibility many people used to be unaware of
- Lots of toolkits and frameworks out there
- Know some design patterns
- Be aware of common Ajax mistakes
- Know tools: DOM inspector or modi (beta), JS console, debuggers, etc.
- Know some Ajax alternatives