

AJAX

This work is licensed under the
Creative Commons Attribution 2.0 Belgium License.

To view a copy of this license,
visit <http://creativecommons.org/licenses/by/2.0/be/>
or send a letter to
Creative Commons,
171 Second Street, Suite 300,
San Francisco, California, 94105,
USA.



Planning

- Introduction to AJAX
 - Building AJAX pages: Basic (with exercises)
 - Building AJAX pages: DOJO (with exercises)
 - AJAX in web application frameworks
-

AJAX

introduction to AJAX

Agenda

- *What is AJAX?*
 - Why AJAX?
 - AJAX dissected
-

What is AJAX?

- Asynchronous
 - JavaScript
 - and XML
-

What is AJAX?

- Asynchronous
 - by default, JavaScript executes on the UI thread
 - asynchronous communication with the server
 - we create and send the request
 - code continues to run and control returns to the user
 - when the response comes back, it is handled
 - benefit: responsiveness
 - drawback: increased complexity
-

What is AJAX?

- JavaScript
 - scripting language
 - dynamic and weakly typed
 - prototype-based with first-class functions
 - semantics similar to functional languages
 - standardized as ECMAScript
 - supported by most modern browsers
 - but sometimes disabled for security reasons
 - although standardized, browser-specific issues exist
-

What is AJAX?

- XML
 - eXtensible Markup Language
 - commonly used as a data format for exchanging information with the server
 - typically with REST-style web services
 - other possibilities for data formats, often easier to parse and less verbose (network transfer)
 - plain text
 - JSON
-

Agenda

- What is AJAX?
 - *Why AJAX?*
 - AJAX dissected
-

Why AJAX?

- Compare with other technologies
 - client applications
 - 'traditional' web applications
 - Benefits of AJAX and potential drawbacks
 - Alternatives
 - rich internet applications
-

Why AJAX?

- client applications
 - run on the desktop
 - often communicate with some kind of server
 - benefits:
 - feature-rich and responsive
 - excellent integration other desktop applications
 - drawbacks:
 - need to be installed on every machine
 - hard to do version management on client/server
-

Why AJAX?

- web applications
 - accessible through the browser
 - actual code runs on a back-end server
 - benefits:
 - browser as the universal client: easy to access
 - easy to manage and upgrade (server-side)
 - drawbacks:
 - request/response model
 - HTML + CSS don't allow for rich user interaction
-

Benefits of AJAX

- All the benefits of a 'traditional' web application
 - But allows for...
 - increased responsiveness
 - more rapid user feedback
 - richer user interactions
 - desktop application 'look and feel' (more desktop-like UI widgets)
 - better bandwidth usage when only transferring the data that has changed
-

Potential drawbacks

- increased complexity
 - always more involved than plain HTML+CSS
 - cross-browser compatibility of DHTML, CSS and JavaScript
 - yet another language
 - it resembles everything you know, but...
 - performance
 - initial page loading
 - 'chatty' interaction with web server
-

Potential drawbacks

- browser integration
 - browser history
 - back button
 - SEO and analytics
 - 'classical' tools for search engine optimization and web analytics don't work
 - security
 - exchanging business data over the internet in a format that should be easily parseable by client
 - cross-site scripting and code injection
-

Alternatives

- Rich internet application technology
 - usually also run in browser
 - AJAX is one of them
 - other technology options
 - Adobe Flash or OpenLaszlo
 - WPF and SilverLight
 - Java applets and JavaFX
 - Java applications and Java Web Start
 - UI languages such as Mozilla XUL
-

Agenda

- What is AJAX?
 - Why AJAX?
 - *AJAX dissected*
-

AJAX dissected

- Typical AJAX application lifecycle
 - browser loads a page with initial UI
 - user interacts with UI
 - browser asynchronously calls on server functionality
 - UI is updated on-the-fly to match server data
 - application ends when another page is loaded
 - A more pragmatical approach
 - traditional web application, with forms and submits
 - forms are 'enhanced' using AJAX technology: input suggestion, validation, data tables, ...
-

AJAX dissected

- A single event lifecycle
 - user interacts with UI and generates event
 - event is handled in JavaScript
 - start a request to the server
 - user continues to work with UI (possibly already triggering another event)
 - response is coming back from the server
 - receive the response in the background
 - update the UI to show the incoming data
-

AJAX

Building AJAX pages: Basic

Agenda

- *Basic AJAX*
 - More on the DOM
 - Sending and receiving XML
 - Sending and receiving JSON
-

Basic AJAX

- Getting XMLHttpRequest
 - Sending the request
 - Handling the response callback
 - Update the HTML document
-

Getting XMLHttpRequest

- Three ways to do this
 - `new XMLHttpRequest()` for most browsers
 - `new ActiveXObject("Msxml2.XMLHTTP")`
 - `new ActiveXObject("Microsoft.XMLHTTP")`

Gettting XMLHttpRequest

```
var request = false;

try {
    request = new XMLHttpRequest();
} catch (ms) {
    try {
        request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (ms2) {
        request = new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

Sending the request

- open the request
- set the callback method
- send the request

```
function myAjaxFunction() {  
    request.open("GET", "resource.txt", true);  
    request.onreadystatechange = updatePage;  
    request.send(null);  
}
```

Sending the request

- `open()` method takes three parameters
 - HTTP method (e.g. GET, POST, HEAD, ...)
 - URL
 - asynchronous (`true`) or synchronous (`false`)
 - sandbox security model
 - URL should be in the same domain
 - prepares request, does not open a connection
-

Sending the request

- assign a callback method
 - onreadystatechange
- use name (JavaScript has first-class functions) or existing function or define one

```
request.onreadystatechange = myCallbackMethod;
```

```
function myCallbackMethod() {  
    //something here  
}
```

or

```
request.onreadystatechange = function() {  
    //something here  
}
```

Sending the request

- `send()` method of `XMLHttpRequest`
 - one parameter: content to send
- use `null` when using GET/HEAD
 - encode parameters in the URL
(remember to use `escape(...)` for doing this)

Handling the response callback

- Callback method is called more than once
- To determine when/how request is done
 - readyState
 - status
- responseText holds server response

```
function updatePage() {  
    if (request.readyState == 4 && request.status == 200) {  
        var answer = request.responseText;  
        //do something useful with the response  
    }  
}
```

Handling the response callback

- Ready states
 - 0: request is uninitialized (before `open()`)
 - 1: request is setup, but not send yet (before `send()`)
 - 2: request is sent (processing at server-side)
 - 3: partial data available, but response not finished
 - 4: response is complete
 - Not all browsers report all ready states
 - You usually only need to deal with state 4
-

Handling the response callback

- HTTP codes
 - 200: OK
 - 301: moved
 - 302: temporarily moved
 - 403: not authorized
 - 404: not found
 - 500: error at server-side
 - Usually only interested in 200
 - Optionally report the others for troubleshooting
-

Update the HTML document

- Manipulate the document's DOM tree
- More on this later, but for now
 - document.getElementById() finds the element
 - modify value (for form fields) or innerHTML property

```
document.getElementById("divOrSpan").innerHTML =  
    "New <em>inner</em> HTML text";
```

```
document.getElementById("field").value = "new field value";
```

Exercise

- Our local pub wants to build a beer recommendation kiosk
 - Every day, the pub owner will change recommendation.txt
 - contains recommended beer and the price for it in a |-separated format
 - Can you update the page to ...?
 - load the recommendation asynchronously
 - show it in the designated span
-

Agenda

- Basic AJAX
 - *More on the DOM*
 - Sending and receiving XML
 - Sending and receiving JSON
-

More on the DOM

- Everything is a node
 - Manipulating the DOM tree
 - document
 - element
 - attribute
 - text
-

Everything is a node

- Every part of an HTML document is represented as a node in the DOM tree
 - the document itself
 - doctype
 - elements
 - attributes
 - text
-

An example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="nl">
<head>
<title>Exercise 2 : Ordering a beer</title>
</head>
<body>
<h1>Exercise 2 : Ordering a beer</h1>
<div id="content">
<form action="#">
<p><label for="beer">Beer:</label><input type="text" id="beer" /></p>
<p><input type="button" onclick="orderBeer();" value="Add to order"
/></p>
</form>
<h2>Waiter comments</h2>
<span id="comments">Ready to take <em>your</em> order now...</span>
<h2>Your order</h2>
<table id="order"><tbody/></table>
</div>
</body>
</html>
```

Everything is a node

- document node:
 - represents an entire HTML document
 - element node:
 - represents an HTML element (<head/>, , ..)
 - attribute node:
 - attributes on an element (id, for, value, ...)
 - text node
 - represents a block of text in the HTML document
-

Everything is node

- Some terminology
 - `<html/>` is called the root element node
 - `<head/>` is the parent element node of `<title/>`
 - `<title/>` is a child node of `<head/>`
 - ``, `<form/>` and `<table/>` are sibling nodes they have a common parent node, i.e. `<div/>`
 - Not in the API, but used sometimes
 - ancestor
 - descendant
-

Manipulating the DOM

- JavaScript API for manipulating the DOM
 - create nodes
 - read nodes
 - update nodes
 - delete nodes
 - Every change to DOM is automatically shown by your browser
-

Working with nodes

- All node types have common properties
 - nodeName
 - nodeValue
 - parentNode, childNodes
 - firstChild, lastChild, previousSibling, nextSibling
 - attributes
 - Not everything works for every node type
 - childNodes/attributes only make sense for elements
 - Remarks on nodeName/nodeValue
-

Working with nodes

- All node types also have common methods
 - insertBefore(newChild, referenceNode)
 - replaceChild(newChild, oldChild)
 - removeChild(oldChild)
 - appendChild(newChild)
 - hasChildNodes()
 - hasAttributes()
-

An example

```
//getting the <html/> element
//why do we use lastChild for that?
var html = document.lastChild;

//what child nodes does the <html/> have?
for (i = 0; i < html.childNodes.length; i++) {
    alert(html.childNodes[i].nodeName);
}

//which of these will show the page <h1> header text?
alert(html.lastChild.firstChild.nodeValue);
alert(html.lastChild.firstChild.firstChild.nodeValue);
```

Working with document node

- The document is stored in a global variable named 'document'
 - A special property 'documentElement' refers to the root element of the document
 - Methods for creating new nodes
 - `createElement(elementName);`
 - `createAttribute(attributeName);`
 - `createTextNode(text);`
 - Convenience method: `getElementById(...)`
-

An example

```
// let's change the comments <span/>
// what will be the result?
var comments = document.getElementById("comments");
    comments.replaceChild(
        document.createTextNode("Ready to deliver "),
        comments.firstChild);

//...and add a row to the table
var table = document.getElementById("order");
var row = document.createElement("tr");
row.addCell = function(value) {
    var column = document.createElement("td");
    column.appendChild(
        document.createTextNode(value));
    row.appendChild(column);
};
row.addCell("Cell 1");
row.addCell("Cell 2");
row.addCell("Cell 3");
table.getElementsByTagName('tbody')[0].appendChild(row);
```

Working with elements

- Methods for accessing attributes
 - `getAttribute(name)`
 - `removeAttribute(name)`
 - `setAttribute(name, value)`
 - `getAttributeNode(name)`
 - `removeAttributeNode(name)`
 - Convenience method
 - `getElementsByTagName(name)` returns list of element nodes matching the name
-

Working with attributes

- Attributes and the DOM
 - are represented by nodes but...
 - they do not show up in the child nodes
 - You can work with them like other node types
 - use `getAttributeNode/removeAttributeNode` on the owning Element
 - Often easier to use the other Element methods
 - `getAttribute`, `removeAttribute`, `setAttribute`
-

An example

```
var content = document.getElementById("content");

//let us use getElementByTagName to get to the <span/>
var span = content.getElementsByTagName("span")[0];
var img = document.createElement("img");
img.setAttribute("src",
    "http://www.bierebel.com/images/bouteilles/gildenbier.gif");
content.insertBefore(
    img, span.nextSibling);
```

Working with text nodes

- A few other methods are available on text nodes
 - appendData(text)
 - insertData(position, text)
 - replaceData(position, length, text)

```
// let's change the comments <span/> once again  
var comments = document.getElementById("comments");  
comments.firstChild.replaceData(9, 4, "deliver");
```

Agenda

- Basic AJAX
 - More on the DOM
 - *Sending and receiving XML*
 - Sending and receiving JSON
-

Sending and receiving XML

- Sending XML requests
 - Receiving XML responses
 - Handling XML responses
-

Sending XML requests

- Construct the XML
 - using plain string manipulation
 - using the DOM API
(but you still have to convert to string afterwards)
 - Sending the XML request
 - encode in URL using `escape()`
 - submit in request body (e.g. POST)
 - MIME type for sending should be `text/xml`
-

Receiving XML responses

- Receive as text
 - just use `request.responseText` as before
- Receive as XML document
 - use `request.responseXML`

An example

```
var xml = "<order><table>" + escape(table) + "</table><line><beer>"  
        + escape(beer) + "</beer><quantity>" + escape(quantity)  
        + "</quantity></line></order>";
```

```
request.open("POST", "services/order", true);
```

```
//the default value is text/plain
```

```
//we need to override that to ensure correct data transmission
```

```
request.setRequestHeader("Content-Type", "text/xml");
```

```
request.onreadystatechange = updatePage;
```

```
//we send the XML in the body of the POST request
```

```
request.send(xml);
```

```
function updatePage() {
```

```
    if (request.readyState == 4 && request.status == 200) {
```

```
        var xml = request.responseXML;
```

```
        //now do your stuff with the XML document
```

```
    }
```

```
}
```

Handling XML responses

- Use the DOM API to handle XML responses
 - About the DOM
 - Everything is a node
 - Manipulating the DOM tree
 - Wait a second...
 - ...we covered all this already, didn't we?
-

Exercise

- The beer recommendation kiosk you built, is a huge success.
 - We now want a beer ordering module in there.
 - Change the provided page to
 - load beers.xml asynchronously
 - add the info for the ordered beer to the 'order' table
 - add a comment (the description) to the ``, recommend something else if the user orders a beer we don't sell
-

Agenda

- Basic AJAX
 - More on the DOM
 - Sending and receiving XML
 - *Sending and receiving JSON*
-

Sending and receiving JSON

- What is JSON?
 - Sending JSON requests
 - Receiving JSON responses
-

What is JSON?

- JavaScript Object Notation
 - lightweight data interchange format
 - human-readable
 - part of ECMAScript standard since 1999
 - Two basic datatypes
 - key/value pairs become a JavaScript Object
 - ordered value list becomes a JavaScript Array
 - Information and scripts on <http://www.json.org>
-

What is JSON?

- Syntax basics
 - an object is unordered list of key/value pairs
 - : is used to separate the key and value
 - , is used to separate the pairs
 - a value can be any one of
 - string, number, boolean, object or array
 - an object begins with { and ends with }
 - an array begins with [and ends with]
-

An example

```
{ "table": 10 ,  
  "lines": [{"id": "geuze", "quantity": 1},  
            {"id": "leffe", "quantity": 3}]  
}
```

- An order object has
 - a table number
 - an array of order lines
 - Every order line object has
 - a beer id
 - the ordered quantity
-

What is JSON?

- Creating object from JSON
 - directly in code for defining objects
 - from a string variable using `eval()`
 - using `String.parseJSON()`
 - Creating JSON from objects
 - using `Object.toJSONString()`
 - `json.js` is open-source JavaScript
 - provides `parseJSON()` and `toJSONString()` methods
-

An example

```
//using JSON directly in JavaScript code
var order = { "table": 10 ,
              "lines": [{"id":"geuze", "quantity":1},
                       {"id":"leffe", "quantity":3}]
            };
alert("Table " + order.table + " has ordered "
      + order.lines.length + " distinct products");

//making a JSON string out of the order object
var jsonString = order.toJSONString();
alert(jsonString);

//recreating the order objects from a JSON string
var order2 = eval("(" + jsonString + ")");
var order3 = jsonString.parseJSON();

//both orders are exactly the same
alert(order2.lines[0].id + " -> " + order3.lines[0].quantity);
alert(order3.lines[1].id + " -> " + order2.lines[1].quantity);
```

Sending JSON requests

- Very similar to sending XML
 - send the JSON string in the request body
 - but you shouldn't specify the text/xml MIME type
 - JSON is supported by almost every server-side language or technology
 - Java, PHP, Ruby, C, C#, ...
 - Full list on <http://www.json.org>
-

Receiving JSON responses

- Receiving JSON responses
 - just use `request.responseText` to get the JSON String
- Handling the response
 - parse String into Objects/Arrays
 - allows for easier handling compared to XML DOM

An example

```
request.open("POST", "orders_for_waiter", true);
request.onreadystatechange = function() {
    if (request.readyState == 4) {
        if (request.status == 200) {
            var order = eval("(" + request.responseText + ")");
            //you can now start working with your order Object
        }
    }
};
request.send(waiter.toJSONString());
```

Exercise

- Our bartender also wants his part of the AJAX magic: an order processing kiosk application
 - On the server there are several JSON files
 - the order directory contains a set of order files
 - the root contains a price list file (beers.json)
 - When a waiter types in an order number:
 - retrieve the order data and show the table number
 - create a list of everything that was ordered
 - calculate the total amount the order
-

Wrap up

- One Object to rule them all
 - XmlHttpRequest
 - Several ways to exchange data
 - plain text
 - XML
 - JSON
 - Everything else is just basic JavaScript and DOM
-

AJAX

Building AJAX pages with DOJO

Agenda

- *Why use a JavaScript Toolkit?*
 - Why use DOJO?
 - History of DOJO
 - Layers of DOJO
 - AJAX with DOJO
 - Using the DOJO widgets
-

Why use a JavaScript Toolkit?

- Ease JavaScript development
 - JavaScript doesn't have a large system library
 - Hides complex language idioms
 - Solutions for long-standing JavaScript issues
 - Hiding cross-browser incompatibilities
 - Backward and forward compatibility
 - Several options available:
 - DOJO, Rico, Prototype, Scriptaculous, ...
-

Agenda

- Why use a JavaScript Toolkit?
 - *Why use DOJO?*
 - History of DOJO
 - Layers of DOJO
 - AJAX with DOJO
 - Using the DOJO widgets
-

Why use DOJO?

- A very rich library
 - powerful AJAX abstraction
 - graceful degradation
 - bookmarking support
 - AOP-style event system
 - widgets to support UI development
 - animation
 - useful helper methods
 - ...
-

Why use DOJO?

- A strong and active developer community
 - including support from IBM and Sun Microsystems
 - Used in a lot of (open-source) projects
 - Apache MyFaces: JSF implementation
 - Apache Struts²: web application framework
 - Apache Sling: RESTful web applications using JCR
 - Apache Geronimo: application server
 - IBM's Project Zero: creating web mashups
 - ...
-

Agenda

- Why use a JavaScript Toolkit?
 - Why use DOJO?
 - *History of DOJO*
 - Layers of DOJO
 - AJAX with DOJO
 - Using the DOJO widgets
-

History of DOJO

- DOJO 0.4x
 - started in 2004 as a JavaScript DHTML toolkit
 - AJAX support available very early on
 - DOJO 1.x
 - major rewrite, leaner and smaller codebase
 - 1.0 released in Nov 2007, 1.1 in Mar 2008
 - three major parts:
 - DOJO: the core toolkit
 - Dijix: a UI widget framework
 - Dojox: Dojo extensions, less commonly used features
-

Agenda

- Why use a JavaScript Toolkit?
 - Why use DOJO?
 - History of DOJO
 - *Layers of DOJO*
 - AJAX with DOJO
 - Using the DOJO widgets
-

Layers of DOJO

- Package system
 - Language utilities
 - Event system
 - UI utilities
 - Widgets and custom widgets
-

Package system

- Building reusable blocks of code often results in many .js files
 - Problem
 - multiple HTTP requests required to get everything
 - need to ensure correct order for things to work
 - DOJO package system
 - allows you to specify dependencies between modules to only import what you really need
 - package everything in a single file automatically
-

Package system

- Specify dependencies
 - use `dojo.require("...")`
 - `require()` will go out and fetch what is necessary
 - you can add your own modules if you want
 - Compact JavaScript files
 - Everything in a single file for easy downloading
 - Trim whitespace, encode static text, ...
 - Several builds available and you can make your own for optimal performance
-

Language utilities

- Several convenience methods and add-ons
 - wrappers for common idioms
 - forward-compatibility provisioning
 - functional programming APIs
 - shortcuts to commonly used features
-

Language utilities

- `dojo.forEach(array, function)`
 - JavaScript will have a `for (... in ...)` in the future
 - it is here now already
 - use anonymous function ...
 - ... or any other function that takes a single value as a parameter

```
dojo.forEach(beers, function(beer) {  
    //do something with the beer variable  
});
```

Language utilities

- `dojo.byId(id)`
 - shorthand for `document.getElementById(id)`
 - avoids known bugs in some browsers
 - `dojo.addOnLoad(function)`
 - similar to a body onload function
 - you can have multiple callback functions
 - use it to avoid having to put JavaScript code in the middle or bottom of your page
-

Language utilities

- String manipulation
 - packaging: `dojo.require("dojo.string")`
 - `dojo.string.pad(value, length, symbol)`
 - pad the value to the specified length using symbol
 - `dojo.string.trim` (aka `dojo.trim`): trim whitespace
 - `dojo.string.substitute(string, object)`
 - substitute `${value}` in string with values from object
-

Language utilities

- Logging and debugging
 - traditionally, we use a lot of `alert(...)`
 - now, you can do logging using Dojo
 - `console.debug()`
 - `console.info()`
 - `console.warn()`
 - `console.error()`
 - integrates with Firebug console
-

Event system

- Allows for AOP style programming in JavaScript
 - Any function can be notified when any other function fires
 - Also fixes inconsistencies across browsers
 - event object
 - memory leak related to variable scoping
-

Event system

- use `dojo.connect()`
 - to connect global/object functions
 - to connect events to functions

```
connections = [ ];
```

```
//when global function updatePage is called, also call updateFooter  
connections[0] = dojo.connect(null, "updatePage", updateFooter);
```

```
//when the link is updated, also update the table  
connections[1] = dojo.connect(link, "update", table, "update");
```

```
//when the link is clicked, call the global updatePage function  
connections[2] =  
    dojo.connect(dojo.byId("link"), 'onclick', updatePage);
```

```
//remove all connections  
dojo.disconnect(connections);
```

Event system

- Use pub/sub event for loose coupling
 - `dojo.subscribe("topic", context, "functionName")`
 - subscribe to events on the topic
 - any event will call method with event data
 - returns a subscription handle
 - `dojo.publish("topic", [array of parameters])`
 - publish the array of data objects to the topic
 - `dojo.unsubscribe(handle)`
 - unsubscribe the specified handle
-

Event system

```
//subscribe a global function
dojo.subscribe("events/demo", null, global);
//hold on to the handle
var topic = dojo.subscribe("events/demo", object, "myfunc");

//publish event to two subscribers
//subscribing functions should take one parameter
dojo.publish("events/demo", ["Hello"]);

//remove a subscription
dojo.unsubscribe(topic);

//now the event is only received by one function
dojo.publish("events/demo", ["...world"]);
```

UI utilities

- Several utilities for building fancy UI
 - some of them became part of Dijit or Dojox with 1.x
 - `dojo.query()`
 - AJAX enablement
-

dojo.query()

- Selecting DOM nodes
 - by tag
 - by CSS selector
 - by id

```
//select by tag name (case-insensitive)  
dojo.query("div");
```

```
//select by CSS class  
console.info(dojo.query(".teaser"));
```

```
//select by ID (equals dojo.byId("comments"))  
console.info(dojo.query("#comments"));
```

Agenda

- Why use a JavaScript Toolkit?
 - Why use DOJO?
 - History of DOJO
 - Layers of DOJO
 - *AJAX with DOJO*
 - Using the DOJO widgets
-

AJAX in DOJO

- Dojo has several AJAX helper methods (covers core REST HTTP methods)
 - `dojo.xhrGet` for HTTP GET requests
 - `dojo.xhrPost` for HTTP POST requests
 - `dojo.xhrDelete` for HTTP DELETE requests
 - `dojo.xhrPut` for HTTP PUT requests
 - Methods takes a single object as parameter, containing the full request details
-

AJAX in DOJO

- Getting text files from the server
 - url refers to the URL to GET
 - load refers to the callback function
 - two parameters: response data and I/O arguments
 - any function with two parameters will do
 - Dojo will take of a lot of the technical issues
 - creating an XMLHttpRequest the right way
 - dealing with state changes and HTTP codes
 - getting the text of the response
-

AJAX in DOJO

```
var args = {  
    url: "recommendation.txt",  
    load: function(response, ioArgs) {  
        var answer = response.split("|");  
    }  
};  
dojo.xhrGet(args);
```

```
//or shorter  
dojo.xhrGet({  
    url: "recommendation.txt",  
    load: function(response, ioArgs) {  
        var answer = response.split("|");  
    }  
});
```

AJAX in DOJO

- Working with XML and JSON
 - add handleAs to args on xhrGet & co
 - handleAs="json"
 - handleAs="xml"
 - Response in load: callback method becomes
 - the Object/Array parsed from the JSON String
 - a DOM object for XML document
-

AJAX in DOJO

```
dojo.xhrGet({
  url: "beers.xml",
  handleAs: "xml",
  load: function(response, ioArgs) {
    //response is the XML Document
  }
});
```

```
dojo.xhrGet({
  url: 'beers.json',
  handleAs: 'json',
  load: function(response, ioArgs) {
    //response is the Object/Array
  }
});
```

AJAX in DOJO

- Other callback functions
 - error:
 - callback function when errors occur
 - handle:
 - if you want more control over the request processing
 - Other options
 - timeout:
 - cancel and call error callback if time-out exceeded
 - preventCache (boolean, default = false)
 - sync (boolean, default = false)
-

AJAX in DOJO

- Sending data
 - content:
 - request parameter object
 - is transformed/escaped into URL parameters by DOJO
 - form:
 - DOM node for form who's data is to be sent
 - query (also postData, putData):
 - the content to be sent in the request body
 - handleAs:
 - json-comment-filtered, json-comment-optional, javascript
-

Exercise

- We decide to upgrade our existing applications to DOJO.
 - Update the three previous exercises to
 - use `dojo.xhrGet()` to build AJAX calls
 - use `forEach()` and `byId()` to make code more maintainable and readable
 - use the event system to loosely couple event to functions
 - use pub/sub events to update the `` and `<table/>` in the waiter's desk exercise
-

Agenda

- Why use a JavaScript Toolkit?
 - Why use DOJO?
 - History of DOJO
 - Layers of DOJO
 - AJAX with DOJO
 - *Using the DOJO widgets*
-

UI Widgets

- AJAX applications also need more rich UI widgets
 - In Dojo 1.x, this is provided by Dijit
 - Dijit contains everything you need for a widget
 - JavaScript code
 - CSS style sheets
 - Dijit can also be used for creating own widgets
-

Getting started with Dijit

- Enabling a page for Dijit
 - importing the CSS style sheet
 - configuring Dojo to parse the entire page on load
 - `dojo.require("dojo.parser")`
 - setting the CSS style class on the body tag
-

Getting started with Dijit

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="nl">
<head>
<script type="text/javascript" src="js/dojo/dojo/dojo.js"
djConfig="parseOnLoad: true"><!-- DOJO Toolkit --></script>

<script type="text/javascript"><!--
dojo.require("dojo.parser");
--></script>

<style type="text/css">
    @import "js/dojo/dijit/themes/tundra/tundra.css";
</style>
</head>

<body class="tundra">

</body>
</html>
```

Using a widget

- Just a few steps required
 - `dojo.require()` the widget code
 - add `dojoType="widget.name"` to your HTML element
 - add any additional configuration info
 - Samples can be found in Dojo/Dijit distribution
 - if installed in <http://localhost/js>
<http://localhost/js/dojo/dijit/themes/themeTester.html>
-

An example: NumberTextBox

- Using widget `dijit.form.NumberTextBox`
 - `dojo.require("dijit.form.NumberTextBox");`
 - `<input dojoType="dijit.form.NumberTextBox"/>`
 - configure additional properties, some examples:
 - `required="true"` to make the field required
 - `constraints="{min:0, places:0}"` for a positive integer

```
<input type="text" id="number"  
      dojoType="dijit.form.NumberTextBox"  
      required="true" constraints="{min:0, places:0}"/>
```

An example: TabContainer

- Using widget `dijit.layout.TabContainer`
 - `dojo.require("dijit.layout.TabContainer")`
 - `dojo.require("dijit.layout.ContentPane")`
 - create a container `<div/>`
 - `dojoType="dijit.layout.TabContainer"`
 - use `style="width: ...; height: ..."` to specify size
 - for every tab, create a child `<div/>`
 - `dojoType="dijit.layout.ContentPane"`
 - `title="..."` specifies the tab title
-

An example: TabContainer

```
<div id="container" dojoType="dijit.layout.TabContainer"
    style="width: 800px; height: 600px;">

<div title="Recommend a beer" dojoType="dijit.layout.ContentPane">
Recommend a beer
</div>

<div title="Ordering desk" dojoType="dijit.layout.ContentPane">
Order a drink
</div>

<div title="Waiter's desk" dojoType="dijit.layout.ContentPane">
Are you really a waiter?
</div>

</div>
```

Exercise

- Instead of having three different pages, we are now going to create a single page using Dojo widgets:
 - create a tabbed pane
 - add a tab for every exercise
 - make the number field on the last exercise required and numeric only
-

Wrap up

- We have only scratched the surface of Dojo/Dijit UI widgets
 - For more information, look at www.dojotoolkit.org
-

AJAX

AJAX in web application frameworks

Agenda

- *AJAX in J2EE web application development*
 - Two examples
 - Apache MyFaces
 - Apache Struts²
-

AJAX and J2EE

- Base technology for web applications
 - Servlet
 - JSP
 - Tag libraries
 - Filters
 - Hard to build web applications that are
 - secure
 - scalable
 - reliable
 - ...
-

AJAX and J2EE

- Use a web application framework
 - JSF
 - Struts²
 - Wicket
 - Velocity
 - Sling
 - ...
 - Many of them have AJAX support
 - prefer using that to DIY AJAX page construction
-

Agenda

- AJAX in J2EE web application development
 - Two examples
 - *Apache MyFaces*
 - Apache Struts²
-

AJAX in JSF

- JavaServer Faces
 - UI component can encapsulate AJAX
 - Apache MyFaces
 - JSF Core implementation
 - Extension UI component libraries
 - Tomahawk
 - Trinidad
 - Tobago
 - Extension UI components with AJAX support
-

AJAX in JSF

- An example:
 - input suggestions while typing
 - implemented using JSF, AJAX, DOJO and DWT
 - On the UI side
 - add an `<s:inputSuggestAjax />` tag
 - specify the method to get the suggestions
 - On the backend side
 - create a normal Java method that return suggestions
-

AJAX in JSF

- An example
 - on the UI side

```
<s:inputSuggestAjax  
    suggestedItemsMethod="#{bean.getSuggestItems}"  
    value="#{bean.selectedValue}"/>
```

- on the backend side

```
public List getSuggestItems(String prefix) {  
    List li = new ArrayList();  
    //make a list of suggestions here  
    return li;  
}
```

Agenda

- AJAX in J2EE web application development
 - Two examples
 - Apache MyFaces
 - *Apache Struts²*
-

AJAX in Struts²

- Struts² was previously known as WebWork
 - Struts² comes with an AJAX theme
 - based on DOJO
 - features:
 - AJAX client side validation
 - remote form submission
 - AJAX-enabled `<a/>` and `<div/>` tags
 - autocomplete tag
-

AJAX in Struts²

- An example
 - auto-complete input text field
 - On the UI side
 - use an `<s:autocompleter/>` tag
 - On the server side
 - action should create JSON response
 - Struts² has a JSON plugin to ease the task of sending/receiving JSON from your action classes
-

AJAX in Struts2

- On the UI side

```
<s:url id="json" value="/JSONList.action" />
<form action="/someurl">
  <s:autocompleter theme="ajax" href="%{json}" name="state"/>
  <input type="submit">
</form>
```

- On the server side

```
<interceptor-ref name="json">
  <param name="root">bean1.bean2</param>
</interceptor-ref>
```

```
<result type="json">
  <param name="root">
    person.job
  </param>
</result>
```

Wrap up

- Frameworks help you build AJAX applications
 - tag libraries encapsulate client-side logic
 - libraries to help you build AJAX request handlers
 - Underlying technology
 - standard J2EE server-side technology
 - + everything we learned about JavaScript, XML, JSON, DOJO, ...
-

The end
