WKW SCHOOL

2019

CI6206 Internet Programming

AJAX

Wong Twee Wee

Ver1.1

AJAX INTRO

- AJAX - Asynchronous JAvascript and XML
  - Better user experience
    - "asynchronous“- means it can perform server request and response without a page refresh.
    - Update portions of a page based upon user events.
  - More flexibility
    - It can send & receive information in a variety of formats(JSON, XML, HTML,text files).
    - Javascript or jQuery

AJAX INTRO

- Asynchronous JavaScript and XML
  - Combination of standards-based technologies
    - **XHTML and CSS – presentation**
    - **Document Object Model – dynamic display and interaction**
    - **XML/JSON/HTML – data representation**
    - **JavaScript/jQuery/Angularjs etc – coding**
    - **XMLHttpRequest – JavaScript object for retrieving XML asynchronously**
  - Example
    - Google Suggest
    - Google maps
    - W3C

JAVASCRIPT FRAMEWORK

(MOST POPULAR)

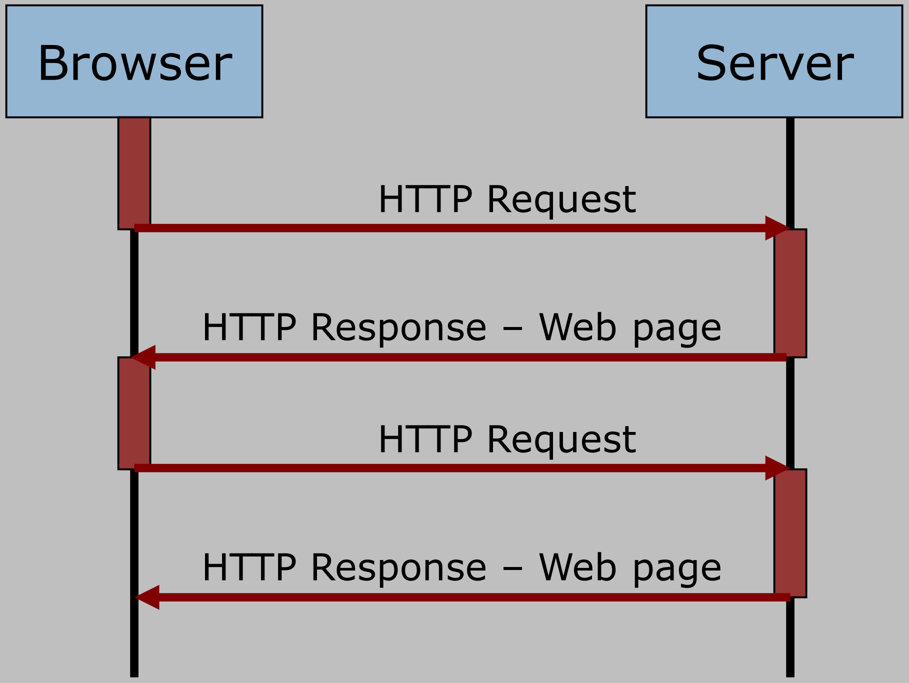
Framework	Descriptions
jQuery	A JavaScript library that provides an Ajax framework and other utilities, and jQuery UI, a plug-in that provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets.
MooTools	A compact and modular JavaScript framework best known for its visual effects and transitions.
Prototype	A JavaScript framework that provides Ajax and other utilities, and Script.aculo.us, a plug-in for animations and interface development.
YUI Library	A set of utilities and controls, for building richly interactive web applications using techniques such as DOM scripting, DHTML and Ajax.
ASP.NET AJAX	A set of extensions to ASP.NET for implementing Ajax functionality.
Spry framework	An open source Ajax framework developed by Adobe which is used in the construction of Rich Internet applications. It is no longer maintained.[3]
Dojo Toolkit	An Open Source DHTML toolkit written in JavaScript.
Ext JS	A library that extends Prototype, Jquery and YUI until version 1.0. Since version 1.1 a standalone Ajax framework.
Backbone.js	Loosely based on the Model-View-Controller application design paradigm
AngularJS	A client side JavaScript MVC framework to develop a dynamic web application.
Unified.JS	A part of the JavaScript language framework.

AJAX VS JAVASCRIPT

AJAX	JAVASCRIPT
AJAX sends requests to the server and does not wait for the response. It performs other operations on the page during that time.	JAVASCRIPT make a request to the server and waits for response.
AJAX does not require the page to refresh for downloading the whole page	JAVASCRIPT manages and controls a web page after being downloaded.
AJAX minimizes the overload on the server since the script needs to request once.	JAVASCRIPT posts a request that updates the script every time.

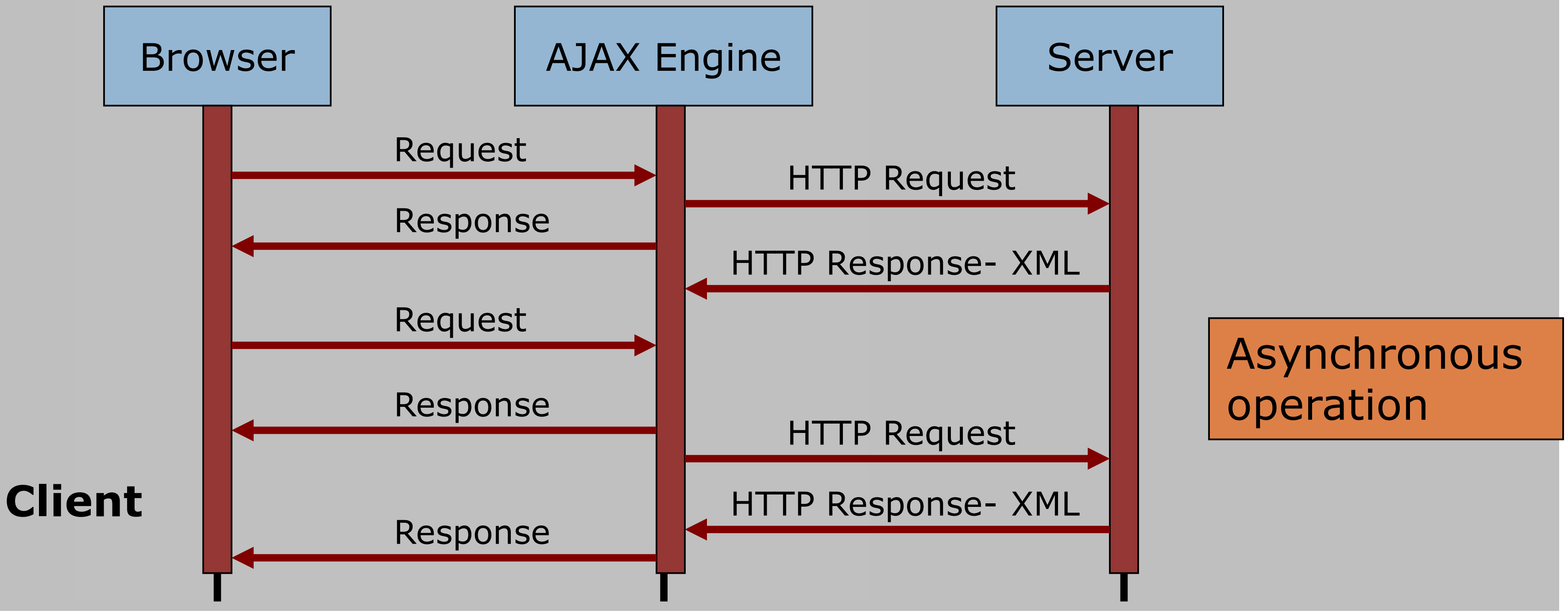
CLASSIC WEB APPLICATION

1. **Browser hosts applications**
   - **Traditional Web applications**
     - **Client renders content, server contains logic**
     - **Synchronous operation**



AJAX WEB APPLICATION

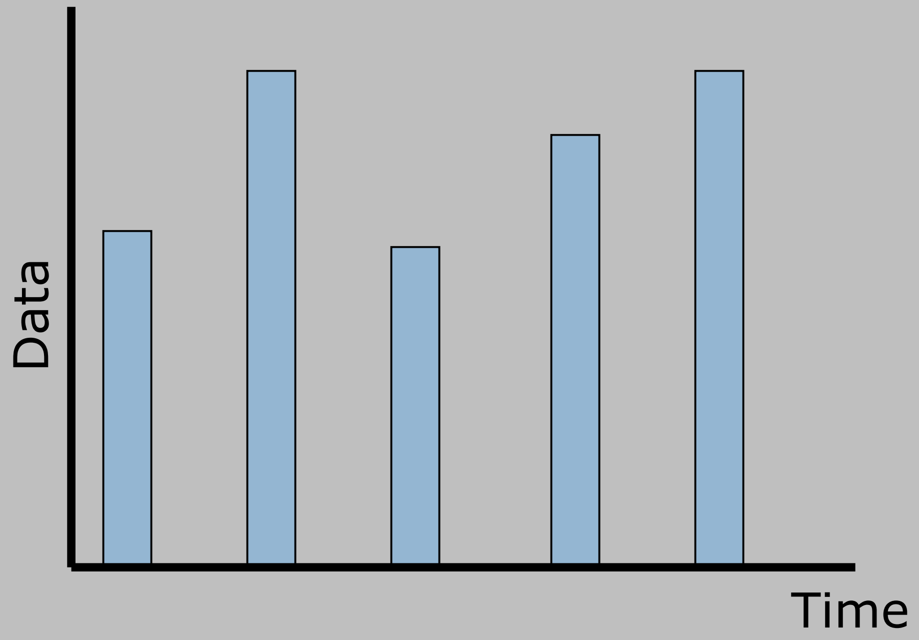
1. Browser hosts applications
   - AJAX applications
     - Client contains some logic – AJAX engine
     - Handles interaction between client and server
     - JavaScript code that persists throughout session



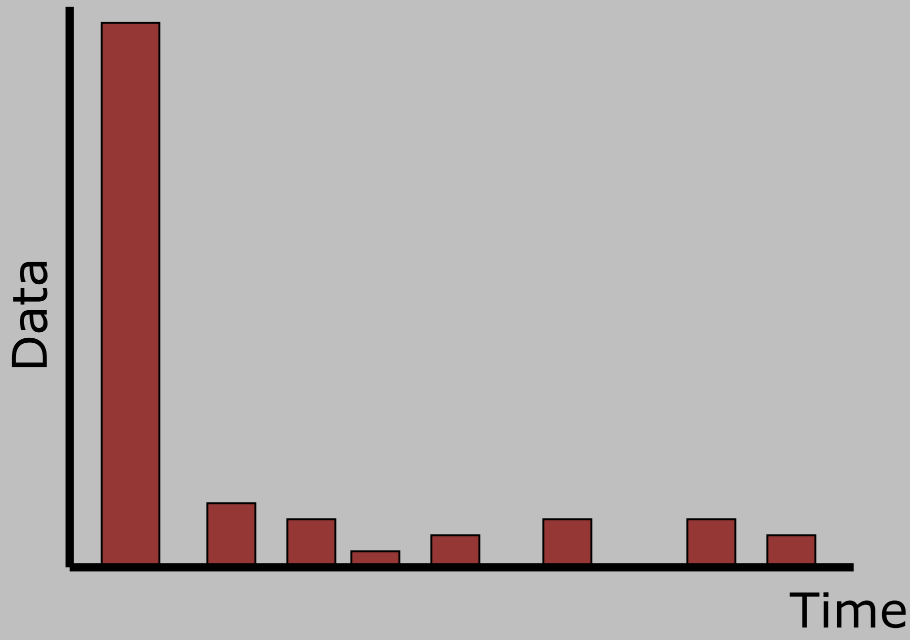
AJAX CHARACTERISTICS

2. **The server delivers data not content**
   - Plain text
   - XML document

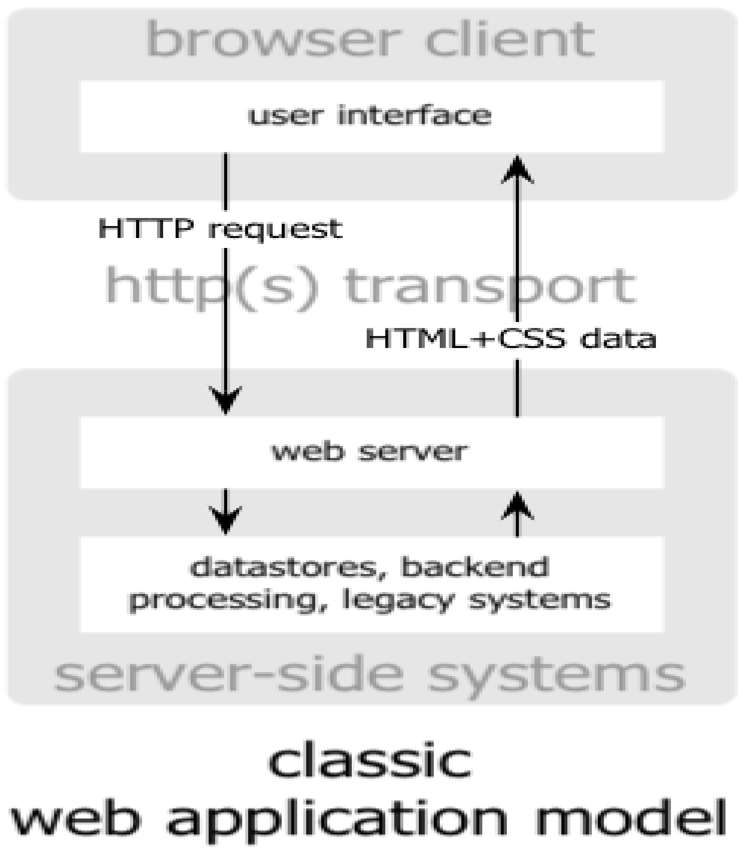
Traditional Web Application

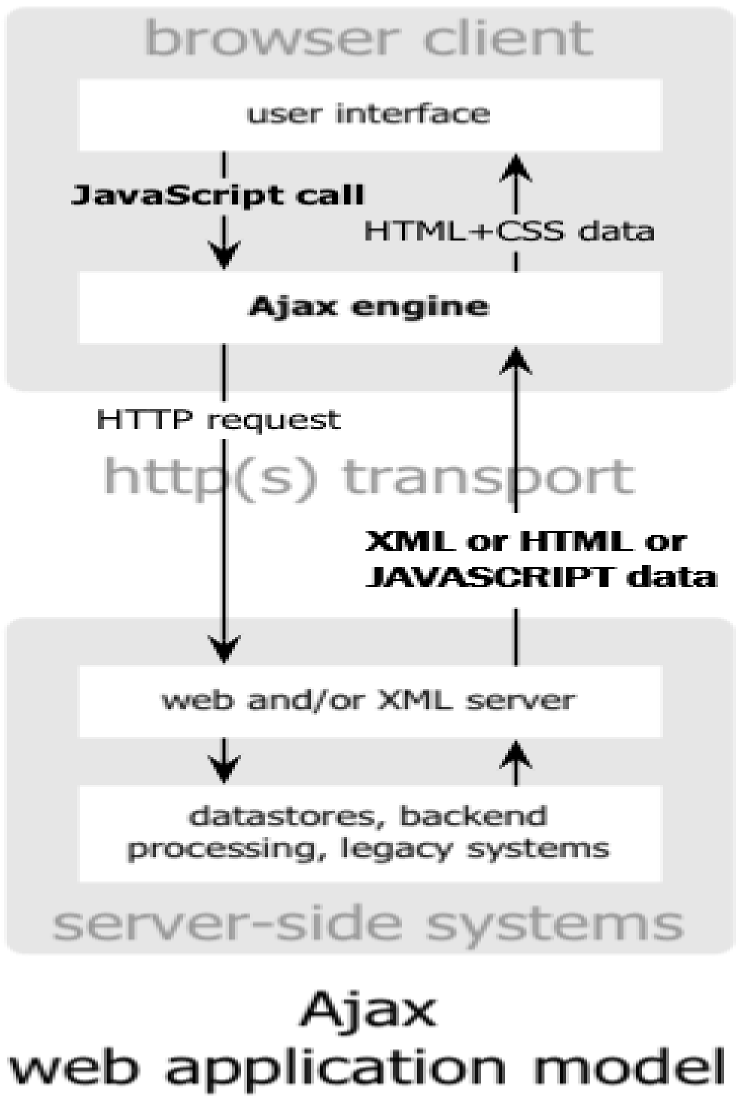


AJAX Application

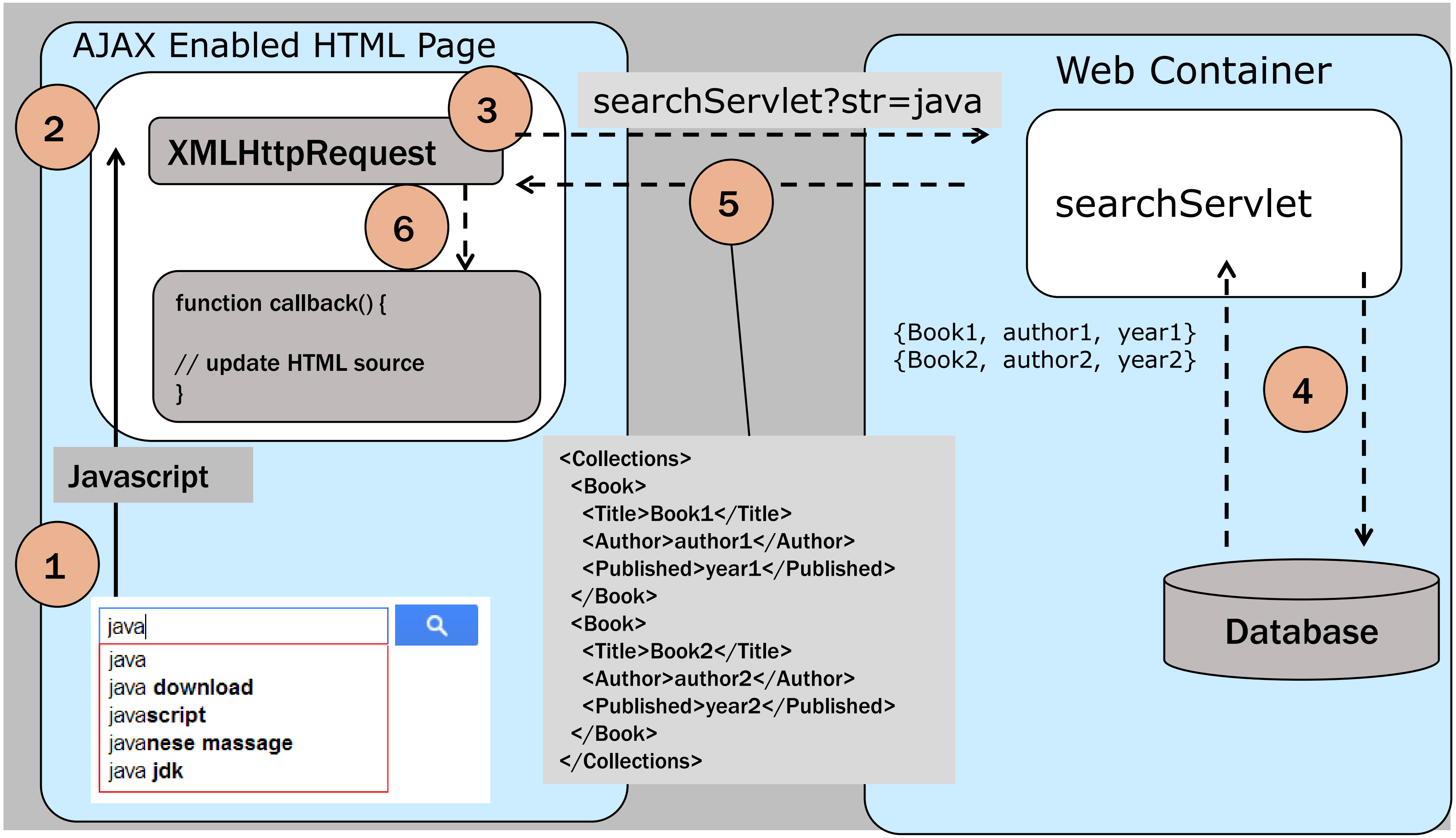


Classic vs Ajax



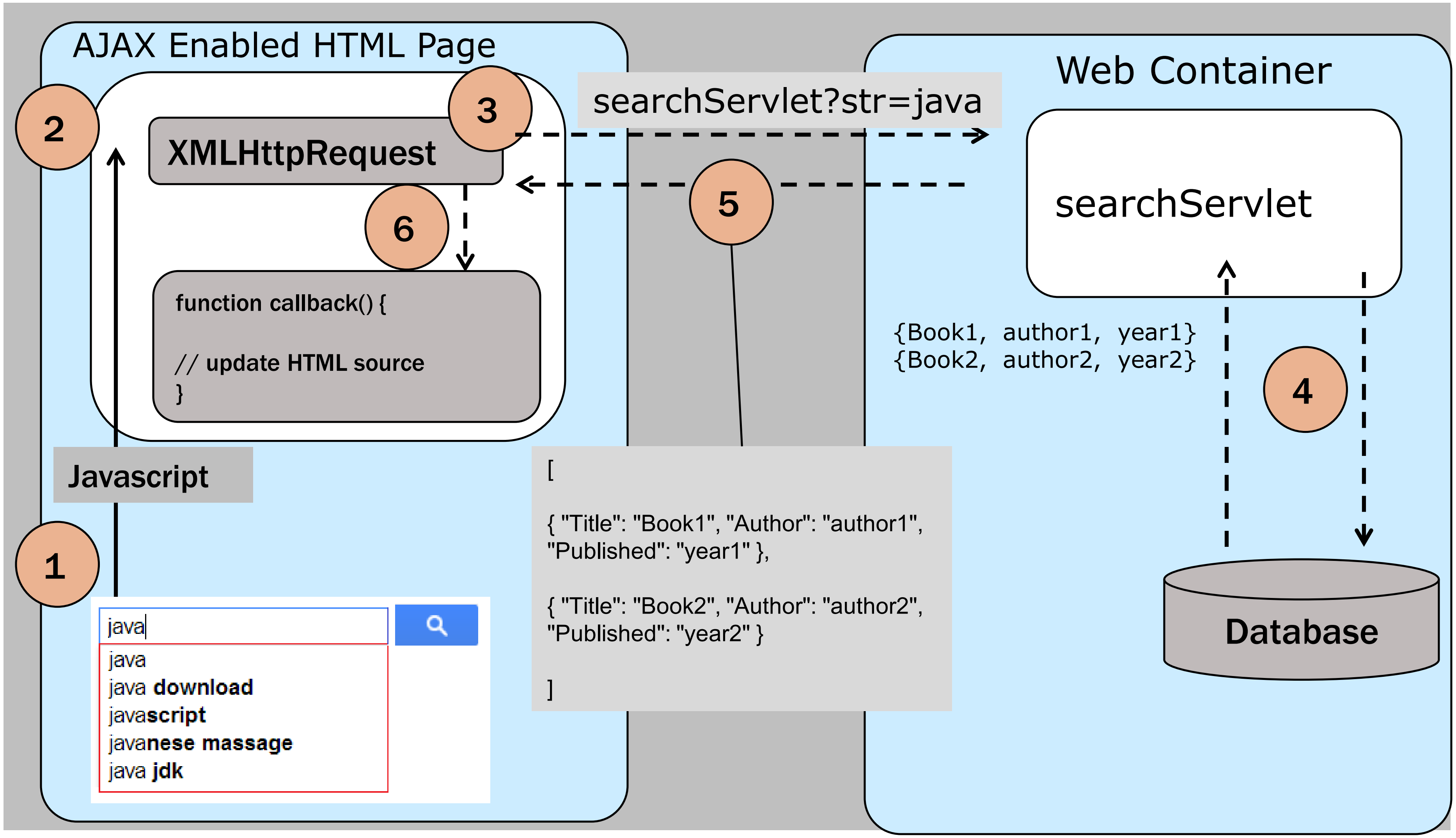


AJAX SAMPLE APP



```
<Collections>
  <Book>
    <Title>Book1</Title>
    <Author>author1</Author>
    <Published>year1</Published>
  </Book>
  <Book>
    <Title>Book2</Title>
    <Author>author2</Author>
    <Published>year2</Published>
  </Book>
</Collections>
```

AJAX SAMPLE APP



```
[

{ "Title": "Book1", "Author": "author1",
"Published": "year1" },

{ "Title": "Book2", "Author": "author2",
"Published": "year2" }

]
```

STEPS OF AJAX OPERATION

- 1.A client event occurs
- 2.An XMLHttpRequest object is created
- 3.The XMLHttpRequest object is configured
- 4.The XMLHttpRequest object makes an asynchronous request
- 5.A servlet returns an XML/JSON/Text document containing the result
- 6.The XMLHttpRequest object calls the callback() function and processes the result
- 7.The HTML DOM is updated

THE XMLHTTPREQUEST OBJECT

- Allows client-side JavaScript code to make HTTP requests to the server asynchronously
  - Supports GET, POST, HEAD, etc.
  - Can be used to retrieve text-based data
    - `XMLHttpRequest.responseText`
    - `XMLHttpRequest.responseXML`

THE XMLHTTPREQUEST OBJECT

- Sending requests
  - open(method, url, $\boxed{\text{async, user, password)}}$
    - method – GET, POST, etc
    - url – relative or absolute
    - async – true or false
    - user, password – for authentication
  - setRequestHeader(name, value)
    - Used for POST
  - send(data)
    - data – URL encoded
    - Used for POST

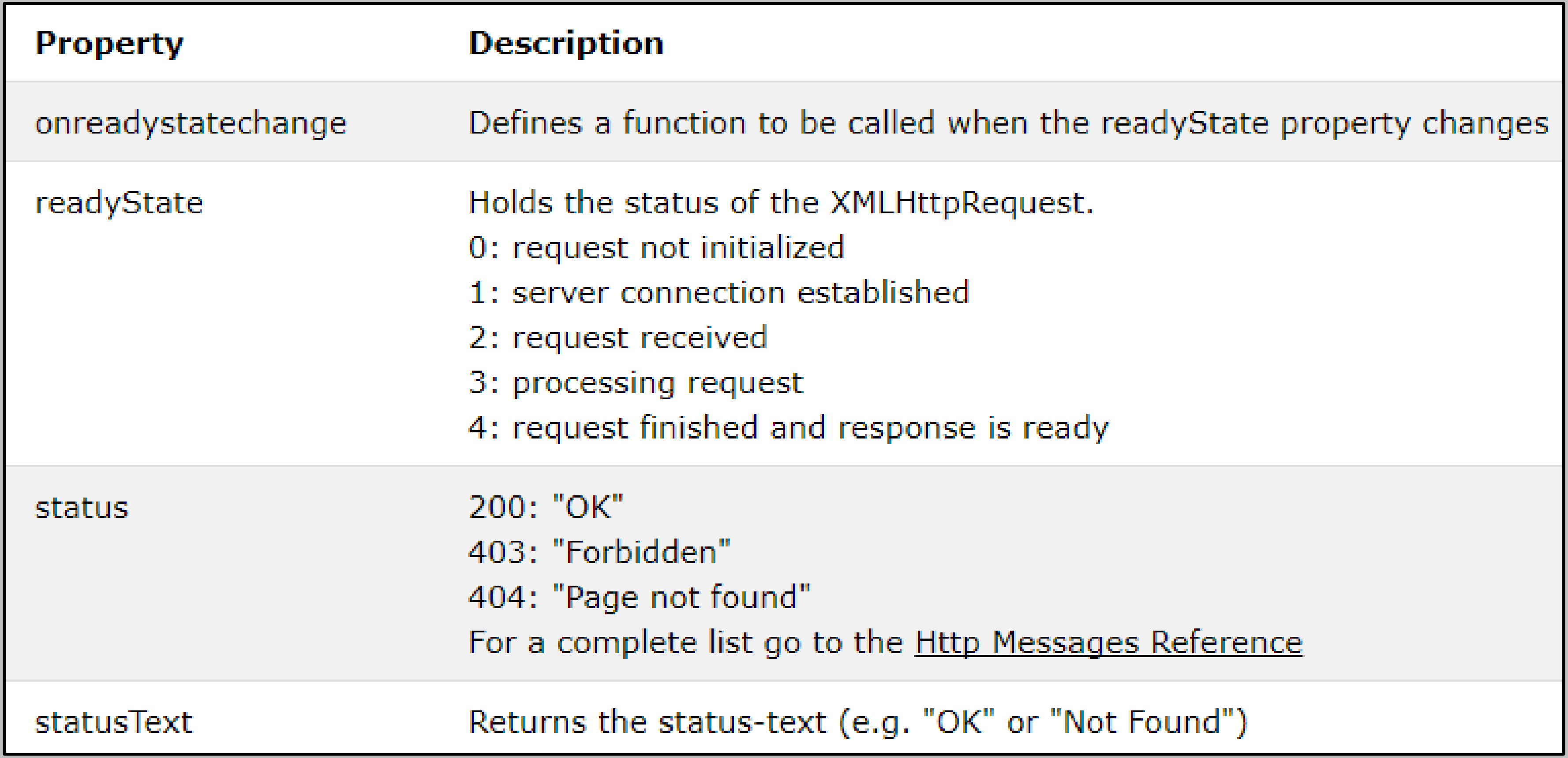
optional

```
var req = createRequest();
req.open("POST", "post-server", true);
req.setRequestHeader("Content-Type",
   "application/x-www-form-urlencoded");

var query = "name=" + escape(myname) +
   "&phone=" + escape(myphone);

req.send(query);
```

THE `XMLHTTPREQUEST` OBJECT

- **Retrieving responses asynchronously**
  - `readyState`
    - **0 – uninitialized;** `open()` **not invoked**
    - **1 – loading;** `send()` **not invoked**
    - **2 – loaded; data not yet available**
    - **3 – interactive; some data received**
    - $\boxed{\textbf{4 – completed; all data received}}$ look out for this
  - `status` **and** `statusText`
    - **HTTP status code (e.g.** `200`**) and message**
  - `onreadystatechange`
    - **Specify event handler to be called when** `readyState` **changes**

AJAX PROPERTY

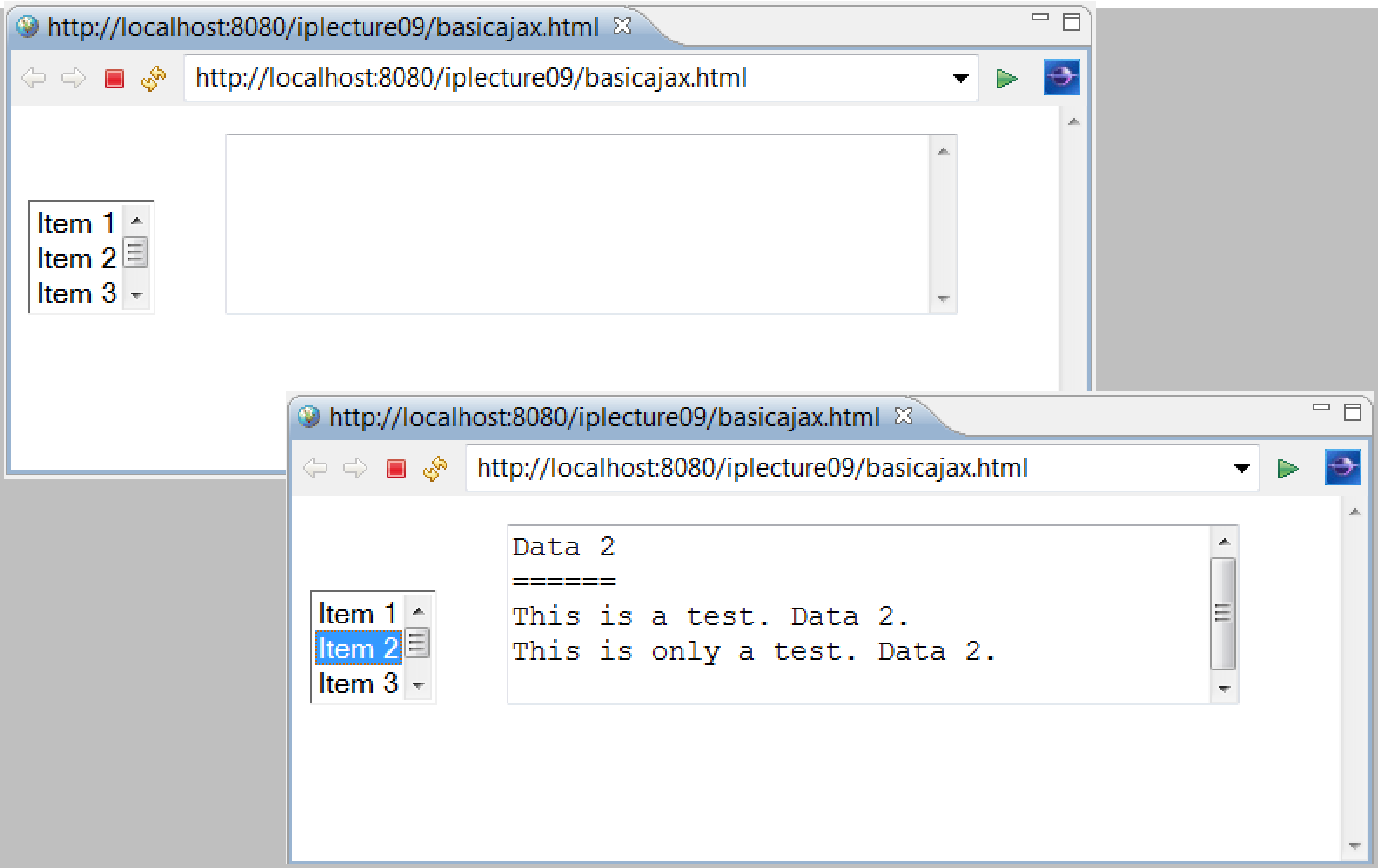


Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the <u>Http Messages Reference</u>
statusText	Returns the status-text (e.g. "OK" or "Not Found")

THE XMLHTTPREQUEST PROPERTIES

- responseText
  - String version of data returned from the server
  - Response in HTML
- responseXML
  - XML document of data returned from the server
  - Response in XML

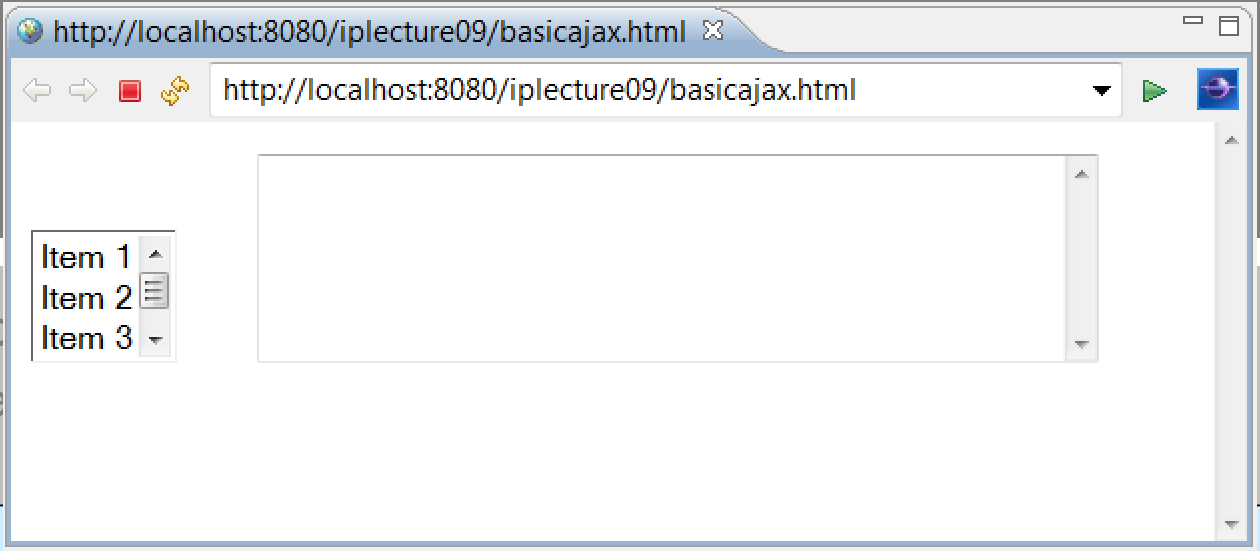
BASIC STEPS IN AJAX


http://localhost:8080/iplecture09/basicajax.html
http://localhost:8080/iplecture09/basicajax.html
Item 1
Item 2
Item 3
http://localhost:8080/iplecture09/basicajax.html
http://localhost:8080/iplecture09/basicajax.html
Item 1
Item 2
Item 3

```
Data 2
======
This is a test. Data 2.
This is only a test. Data 2.
```

1. Construc

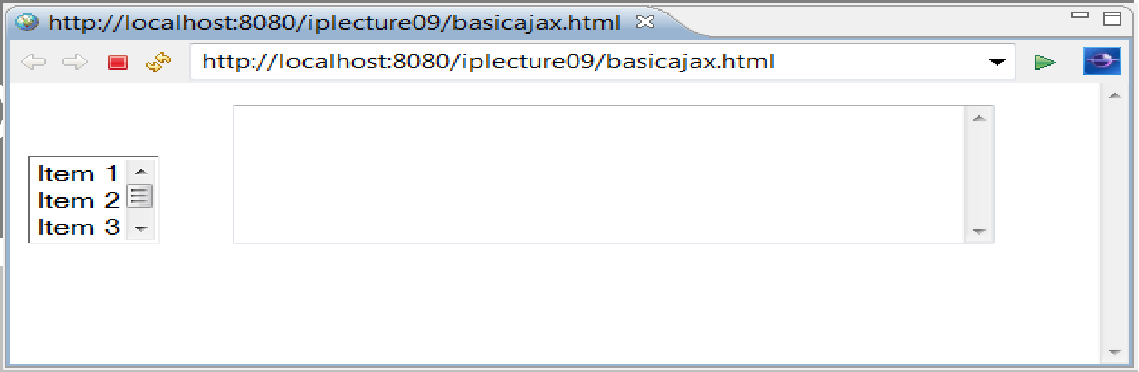
- Include e


http://localhost:8080/iplecture09/basicajax.html

Item 1
Item 2
Item 3

```
<body>
    <form name="myform">
        <select name="items" size="3" onChange="loadURL(this)">
        <option value="data1.txt">Item 1</option>
        <option value="data2.txt">Item 2</option>
        <option value="data3.txt">Item 3</option>
        <option value="data4.xml">Item 4</option>
        </select> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

        <textarea name="results" rows=5 cols=40>
        </textarea>
    </form>
</body>
```

BASIC


http://localhost:8080/iplecture09/basicajax.html
http://localhost:8080/iplecture09/basicajax.html
Item 1
Item 2
Item 3

1. Construct Web page
   - Include event handlers to process user input

```
<body>
    <form name="myform">
        <select name="items" size="3" onChange="loadURL(this)">
```

```
function loadURL(selectmenu) {
   var idx = selectmenu.selectedIndex;
   var file = selectmenu.options[idx].value;
   sendRequest("/servlets/MyServlet?file=" + file);
}
```

```
        <textarea name="results" rows=5 cols=40>
        </textarea>
    </form>
</body>
```

BASIC STEPS IN AJAX

2. **Create** XMLHttpRequest **object**

```
function createRequest() {

    var req = null;

    if (XMLHttpRequest) {
        req = new XMLHttpRequest();
    }
    else if (ActiveXObject ) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else {
        req = null;
    }
    return req;
}
```

BASIC STEPS IN AJAX

3. **Send user interactions on page to server**

```
function sendRequest(url) {
    request = createRequest();
    if (request == null) {
        return;
    }
    request.open("GET", url, true);
    request.send(null);
    request.onreadystatechange = processRequestChange;

}
```

Set event handler and send user's data

BASIC STEPS IN AJAX

4. **Create event handler to receive responses from server**

```
function processRequestChange() {
    if(request == null) {
        return;
    }

    if (request.readyState == READY_STATE_COMPLETE) {
        data = request.responseText;
        myform.results.value = data;
    }
}
```

Get status of request

EXAMPLE (CLIENT)

```
function doCompletion() {
   if (completeField.value == "") {
           clearTable();
   } else {
           var url = "autocomplete?action=complete&id=" +
           escape(completeField.value);
           var req = initRequest(url);
           req.onreadystatechange = function() {
           if (req.readyState == 4) {
                   if (req.status == 200) {
                           parseMessages(req.responseXML);
                   } else if (req.status == 204){
                           clearTable();
                   }
           } };
           req.open("GET", url, true);
           req.send(null);
   }
}
```

XML DATA

```
<employees>
  <employee>
      <id>3</id>
      <firstName>George</firstName>
      <lastName>Murphy</lastName>
  </employee>
  <employee>
      <id>2</id>
      <firstName>Greg</firstName>
      <lastName>Murphy</lastName>
  </employee>
</employees>
```

EXAMPLE (SERVER)

Servlet doGet()

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException,
   ServletException {
...
   String targetId = request.getParameter("id");
   Iterator it = employees.keySet().iterator();
   while (it.hasNext()) {
           EmployeeBean e = (EmployeeBean)employees.get((String)it.next());
           if ((e.getFirstName().toLowerCase().startsWith(targetId) ||
                       e.getLastName().toLowerCase().startsWith(targetId)) && !targetId.equals("")) {
                       sb.append("<employee>");
                       sb.append("<id>" + e.getId() + "</id>");
                       sb.append("<firstName>" + e.getFirstName() + "</firstName>");
                       sb.append("<lastName>" + e.getLastName() + "</lastName>");
                       sb.append("</employee>");
                       namesAdded = true; } // if
   } // while
   if (namesAdded) {
           response.setContentType("text/xml");
           response.setHeader("Cache-Control", "no-cache");
           response.getWriter().write("<employees>" + sb.toString() + "</employees>");
   } else {
           response.setStatus(HttpServletResponse.SC_NO_CONTENT);
   }
} // doGet
```

PROCESSING RESPONSE

Processing the response

```
function parseMessages(responseXML) {
   clearTable();
   var employees = responseXML.getElementsByTagName("employees")[0];
   if (employees.childNodes.length > 0) {
           completeTable.setAttribute("bordercolor", "black");
           completeTable.setAttribute("border", "1");
   } else {
           clearTable();
   }
   for (loop = 0; loop < employees.childNodes.length; loop++) {
           var employee = employees.childNodes[loop];
           var firstName = employee.getElementsByTagName("firstName")[0];
           var lastName = employee.getElementsByTagName("lastName")[0];
           var employeeId = employee.getElementsByTagName("id")[0];

           appendEmployee(firstName.childNodes[0].nodeValue,lastName.childNodes[0].
   nodeValu, employeeId.childNodes[0].nodeValue);
   }
}
```

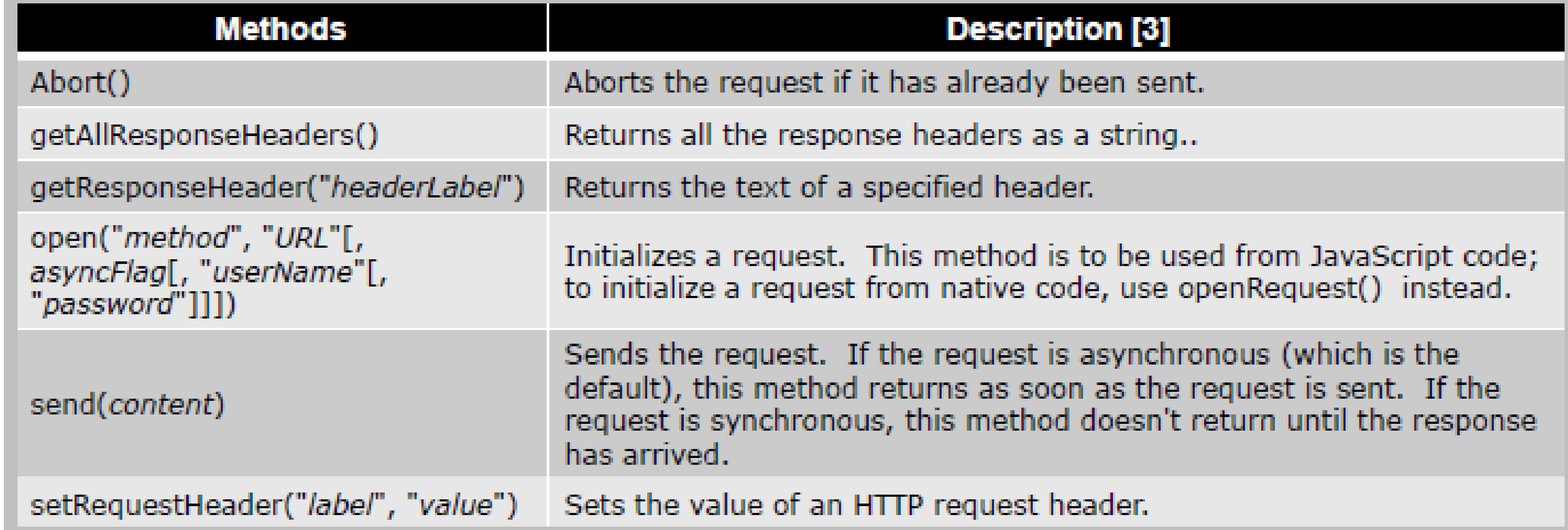
XMLHTTPREQUEST OBJECT

Properties	Description [3]
onreadystatechange	A JavaScript function object that is called whenever the readyState attribute changes. The callback is called from the user interface thread.
readyState	Returns values that indicate the current state of the object.

Value	State	Description
0	UNINITIALIZED	open() has not been called yet.
1	LOADING	send() has not been called yet.
2	LOADED	send() has been called, and headers and status are available.
3	INTERACTIVE	Downloading; responseText holds partial data.
4	COMPLETED	The operation is complete.

Properties	Description [3]
responseText	The response to the request as text, or null if the request was unsucessful or has not yet been sent.
responseXML	The response to the request as a DOM Document object, or null if the request was unsuccessful, has not yet been sent, or cannot be parsed as XML. The response is parsed as if it were a text/xml stream.
status	The status of the response to the request. This is the HTTP result code (for example, status is 200 for a successful request).
statusText	The response string returned by the HTTP server. Unlike status, this includes the entire text of the response message ("200 OK", for example).

XMLHTTPREQUEST OBJECT



Methods	Description [3]
Abort()	Aborts the request if it has already been sent.
getAllResponseHeaders()	Returns all the response headers as a string..
getResponseHeader("*headerLabel*")	Returns the text of a specified header.
open("*method*", "*URL*"[, *asyncFlag*[, "*userName*"[, "*password*"]]])	Initializes a request. This method is to be used from JavaScript code; to initialize a request from native code, use openRequest() instead.
send(*content*)	Sends the request. If the request is asynchronous (which is the default), this method returns as soon as the request is sent. If the request is synchronous, this method doesn't return until the response has arrived.
setRequestHeader("*label*", "*value*")	Sets the value of an HTTP request header.

JQUERY/AJAX -SAMPLE

```
$.ajax({
    type: "POST",
    url: "example.php",
    data: "name=John&location=Boston“
 }).done( function(msg)  {
     alert( "Data Saved: " + msg );
 }).fail( function( xmlHttpRequest, statusText, errorThrown ) {
     alert(
             "Your form submission failed.\n\n"
                + "XML Http Request: " + JSON.stringify( xmlHttpRequest )
                + ",\nStatus Text: " + statusText
                + ",\nError Thrown: " + errorThrown );
 });
```

THE DOCUMENT OBJECT MODEL

- DOM forms a bridge between the elements on a web page and scripts that manipulate these objects
  - Defines a set of objects which has methods, properties and event handlers
    - Browser object model technically part of DOM
  - Defines a standardized way to access them
  - Browser-dependent (standardization through W3C DOM)

AJAX uses DOM to modify UI dynamically

THE DOCUMENT OBJECT MODEL

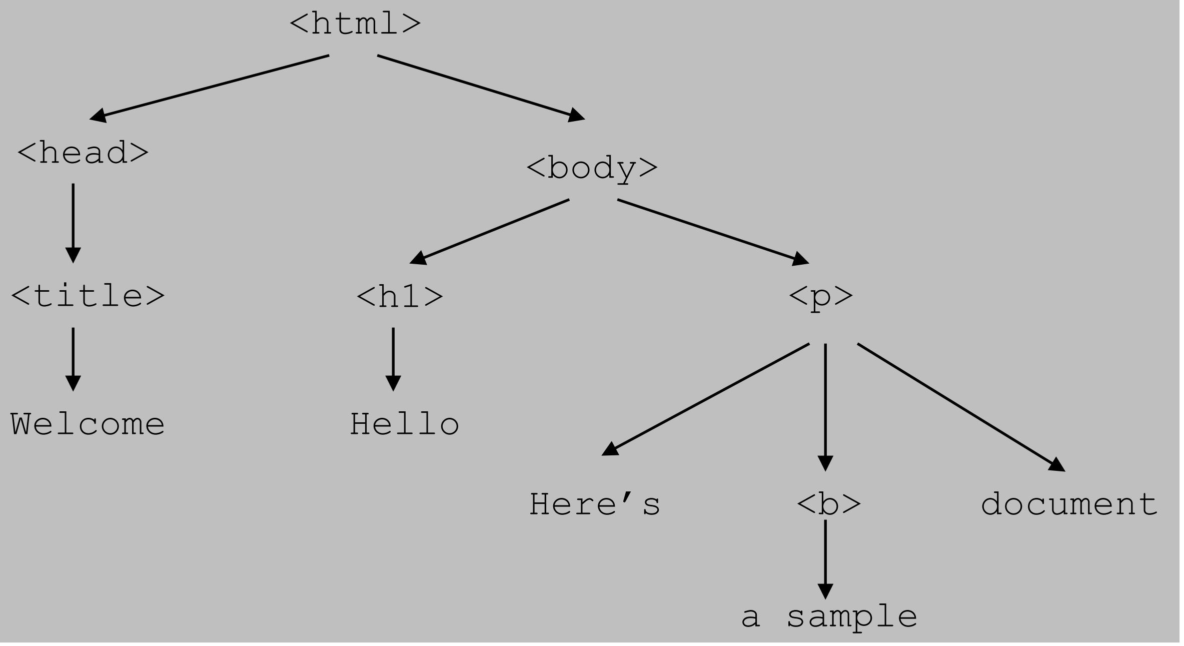
- Tree structure organization of Web page
  - Root – <html>
  - Content – <body>
  - Others – paragraphs, tables, lists, etc.
  - Access via document object (Web page root)
- Some DOM objects

Object	Description
Element	HTML elements
Attribute	Attributes of a HTML tag
Text	Text between HTML tags

THE DOCUMENT OBJECT MODEL

- Elements have
  - Single parent element
  - Zero or more child elements
  - Any number of attributes

```
<html>
<head>
<title>Welcome</title>
</head>
<body>
<h1>Hello</h1>
<p align="center">
  Here’s <b>a sample</b>
  document
</p></body></html>
```



$$\begin{tikzcd}
 & \text{<html>} \arrow[dl] \arrow[dr] & & & \\
\text{<head>} \arrow[d] & & \text{<body>} \arrow[dl] \arrow[dr] & & \\
\text{<title>} \arrow[d] & \text{<h1>} \arrow[d] & & \text{<p>} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Welcome} & \text{Hello} & \text{Here's} & \text{<b>} \arrow[d] & \text{document} \\
 & & & \text{a sample} &
\end{tikzcd}$$

THE DOCUMENT OBJECT MODEL

- **Some node properties**

Property	Description
`firstChild`	First child node of element
`lastChild`	Last child node of element
`nextSibling`	Returns next sibling node of current element
`nodeName`	Name of node
`nodeType`	1 = element; 2 = attribute; 3 = text
`nodeValue`	Value of node in plain text
`parentNode`	Parent node of element
`innerHTML`	Markup and content within element

THE DOCUMENT OBJECT MODEL

- **Some node methods**

Method	Description
appendChild(node)	Appends new child node
hasChildNodes()	Returns true if node has children
removeChild(node)	Removes child node
replaceChild(new, old)	Replaces old child with new one
insertBefore(new, current)	Inserts new node in list of children
setAttributeNode(attr)	Adds new attribute to node
getAttributeNode(attr)	Returns specified attribute
getAttribute(name)	Returns value of specified attribute

THE DOCUMENT OBJECT MODEL

- **Some** `document` **methods**

Method	Description
`createAttribute(name)`	Creates new attribute specified by name
`createElement(type)`	Creates new element specified by `type`
`createTextNode(data)`	Creates new text node
`getElementById(id)`	Returns element specified by `id`
`getElementsByTagName(tag)`	Returns list of elements specified by `tag`

LINKS

- CSS, JavaScript, DOM and HTML
  - http://www.w3schools.com/
- XMLHTTPRequest
  - http://msdn2.microsoft.com/en-us/library/ms760305.aspx
- AJAX
  - http://www.adaptivepath.com/publications/essays/archives/000385.php
  - http://developer.mozilla.org/en/docs/AJAX:Getting_Started