*JavaScript and AJAX*
2nd Edition

# Tutorial 12

Programming with AJAX

## Objectives

- Explore the history of AJAX
- Create an HTTP request object
- Submit an HTTP request
- Retrieve the text of a request response
- Explore the structure of XML
- Retrieve an RSS news feed

# Objectives (cont'd)

- Parse an XML document
- Study the structure of JSON objects
- Upload and run a Perl CGI script
- Create an autocomplete application
- Retrieve and process JSON data

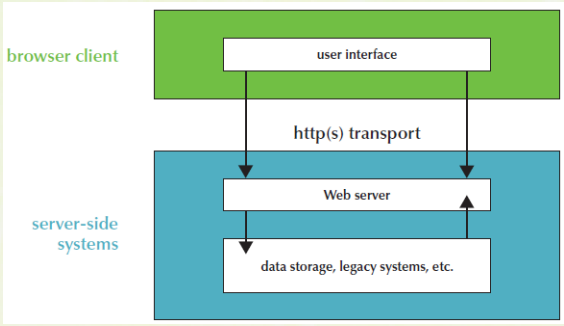


# Introducing AJAX

- **AJAX (Asynchronous JavaScript and XML)**
  - Collection of programming techniques that allow browsers to retrieve data on demand from the Web without having to load a new Web page or reload the current page
  - AJAX is not a programming language; it represents the combination of several techniques (HTML, JavaScript, XML, and asynchronous communication) to create fully interactive Web applications

# The Classic Web Application Model

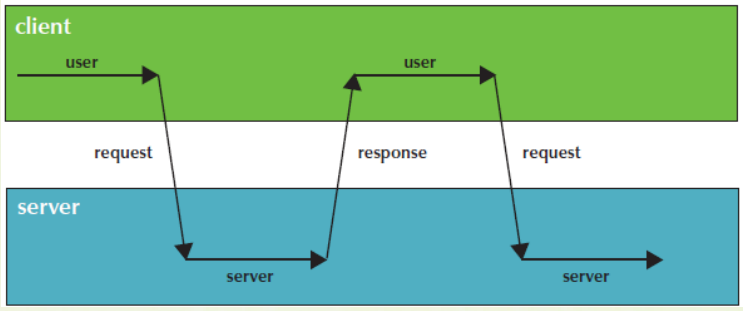
- Users interact with Web servers through a Web page loaded on their browser
- Level of information is exchanged in pages: one page to request information; another page to report on results of the request






# The Classic Web Application Model

- Exchange of information is **synchronous** because the user sends a request and must wait for a response from the server before doing anything else

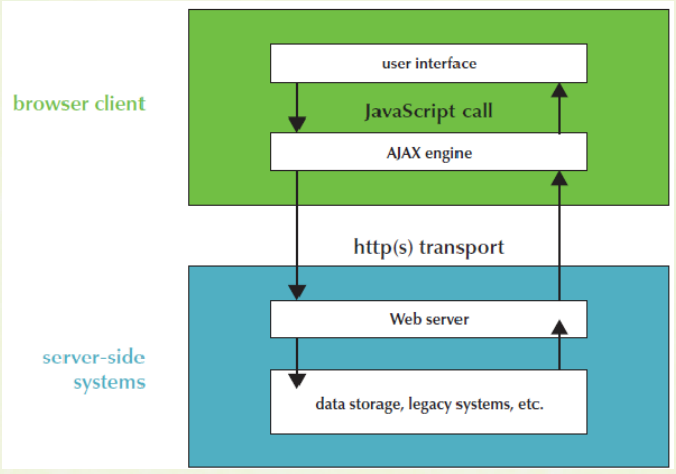


# The AJAX Web Application Model

- Adds an intermediary between user and server-side system – an **AJAX engine** – that is responsible for communicating with the server and returning information from server to user interface
- User interface interacts with the AJAX engine using code written in JavaScript, but the engine communicates with the server using the same HTTP request protocol that the classic model employs to request/receive Web pages

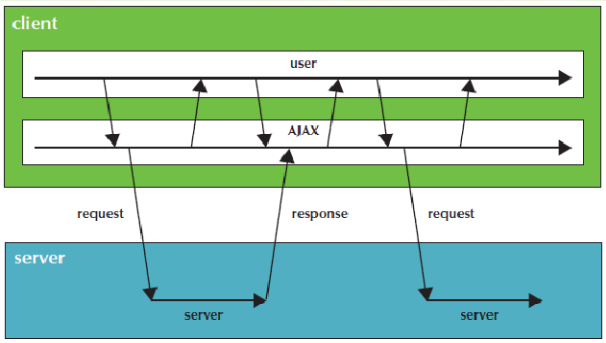


# The AJAX Web Application Model



# The AJAX Web Application Model

- Communication is **asynchronous;** AJAX engine handles server requests, leaving user free to interact with the rest of the Web page






# The AJAX Web Application Model

- Asynchronous communication
  - User does not have to wait for a response from the server to do other tasks on the Web page
  - Several AJAX engines can be employed simultaneously to allow for exchange of data from several different sources
  - Because the server is generating only specific and smaller chunks of information and not an entire Web page, the load on the server is lessened

# AJAX Summary

- A way of developing Web applications that involve asynchronous communication, allowing the user to request information from the server while remaining free to do other tasks on the browser
- Employs JavaScript as the scripting language for handling the interchange of information
- Response from the server can be packaged in XML documents that are easily processed by JavaScript



# Limitations of AJAX

- Requires JavaScript to be running on user's browser
  - If a user disables JavaScript on the browser for security reasons, AJAX applications are disabled
- Does not record user's browser history: cannot use browser's Back button to retrieve information or bookmark information returned by AJAX
- Can pose a problem for users with special needs
- If too many AJAX engines run simultaneously, making calls to the Web server, communication with the server may be compromised

# Exploring the XMLHttpRequest Object

- Code for any AJAX application follows four basic steps:
  - Create an instance of an AJAX engine
  - Define what server resource is opened by the engine
  - Send any necessary data to the server
  - Retrieve a response from the server and process it
- AJAX engines are created using the **XMLHttpRequest object**, a native JavaScript object that stores an HTTP request to the server
- Internet Explorer 5 provides support for request objects through built-in support for the XML language using an XML parser called **MSXML**



# Exploring the XMLHttpRequest Object

- Creating an XMLHttpRequest object
  - To create a request object with non–IE browsers and Internet Explorer 7.0 or higher:

    var *req* = new XMLHttpRequest();
  - For Internet Explorer 5.0 and 6.0:

    var *req* = new ActiveXObject(*pid*);

# Exploring the XMLHttpRequest Object

- XMLHttpRequest constructor

```
/ * Add new code below */

// XMLHttpRequest Constructor
if (typeof XMLHttpRequest == "undefined") {
   XMLHttpRequest = function() {

      // Array of MSXML PIDs
      var pids = ["Msxml2.XMLHTTP.6.0",
                  "Msxml2.XMLHTTP.3.0",
                  "Msxml2.XMLHTTP",
                  "Microsoft.XMLHTTP"];

      // Test each PID
      for (var i = 0; i < pids.length; i++) {
         try {
            return new ActiveXObject(pids[i]);
         } catch (e) {}
      }

      // Halt if unable to create request object
      throw new Error("Unable to create request object");
   }
}
```

tests whether the browser supports the native XMLHttpRequest object

array of possible ActiveX program IDs

tests each program ID by creating a request object

if the browser is unable to create the ActiveX request object, it throws an error, halting the program



# Exploring the XMLHttpRequest Object

- Methods associated with the XMLHttpRequest object

Used in every AJAX application

| Method | Description |
|---|---|
| *req*.abort() | Stops any transfer of data involving the *req* request object |
| *req*.getAllResponseHeaders() | Returns a string containing the HTTP headers received by the *req* request object |
| *req*.getResponseHeader(*header*) | Returns the text of the HTTP *header* received by the *req* request object |
| *req*.open(*method*, *url*, *async*, *user*, *pwd*) | Specifies the transaction *method* and *url* applied to the destination of the request; the *async*, *user*, and *pwd* parameters are optional and indicate whether to set up asynchronous communication, and supply a username and password for the transaction |
| *req*.overrideMimeType(*mime*) | Overrides the mime-type of the response from the server, substituting *mime* as the mime-type by which the response should be interpreted (not supported by Internet Explorer) |
| *req*.send(*content*) | Specifies any *content* to be included with the HTTP request |
| *req*.setRequestHeader(*name*, *value*) | Specifies a *name*/*value* pair to the header being sent with the HTTP request |

# Exploring the XMLHttpRequest Object

- The open() method specifies the server resource to access with the request object:

  *req*.open(*method*, *url*, *async*, *user*, *pwd*)

  - The *method* parameter has two possible values: GET and POST
    - The **get method** sends data to the Web server by attaching a field name/value pair to the URL
    - The **post method** sends data in a separate stream from the URL (several advantages)
- The send() method sends data to the server:

  *req*.send(*content*)



# Exploring the XMLHttpRequest Object

- Retrieving an HTML fragment
  - Create a request object to retrieve contents of the file via the get method, running this code in response to the load event of the window object

# Exploring the XMLHttpRequest Object

- Processing the XMLHttpRequest response
  - The AJAX application retrieves and interprets server's response to that request
  - Stages of interaction between browser and server:
    - Uninitialized: Request object is created
    - Loading: Open() method is called
    - Loaded: Browser successfully sends the request using the send() method, but has received no response from the server
    - Interactive: Browser begins receiving a partial response from the server
    - Complete: Server has completed sending a response to the browser



# Exploring the XMLHttpRequest Object

- Properties of the request object

| Property | Description |
|---|---|
| *req*.onreadystatechange | Event handler that fires whenever the readyState value of the *req* request object changes |
| *req*.readyState | Integer indicating current state of the *req* request object, where: 0=uninitialized, 1=loading, 2=loaded, 3=interactive, and 4=complete |
| *req*.responseText | Text returned by the *req* request object |
| *req*.responseXML | XML document returned by the *req* request object |
| *req*.status | Numeric code returned by server indicating the communication status of the request |
| *req*.statusText | Text string returned by the server indicating the communication status of the request |

# Exploring the XMLHttpRequest Object

- Before processing the response, confirm that the request was successfully handled by the server by using the status property of the request object

| Status Code | Description | Status Code | Description |
|---|---|---|---|
| 100 | Continue | 403 | Forbidden |
| 200 | OK | 404 | Not Found |
| 302 | Found | 408 | Request timeout |
| 400 | Bad request | 500 | Internal Server Error |



# Exploring the XMLHttpRequest Object

- To test when the server has responded to a request:

```
req.onreadystatechange = function() {
  if (this.readyState == 4) {
    if (this.status = 200) {
      process the response
    }
  }
}
```

- To retrieve the text of the response:

  *req*.responseText

- To retrieve an XML document of the response:

  *req*.responseXML

# Exploring the XMLHttpRequest Object

- To create and test AJAX code:
  - Load your files to a Web server, or
  - Install a personal Web server such as Windows **IIS (Internet Information Service)** on your computer that will run the HTTP protocol locally



# Exploring the XMLHttpRequest Object

- Placing an AJAX application onto a server
  - You must have an account on the server with a username and password
  - Options for transferring files:
    - Use the ftp protocol
    - Copy the files into the server's root folder
    - Use the Internet Explorer browser to open an ftp connection on your server

# Introducing XML

- **XML (Extensible Markup Language)**
  - A language used to create structured documents
  - Shares common structure with HTML documents
    - Elements are indicated with markup tags that contain textual content; element names are descriptive
    - A markup tag can contain attributes that describe a feature of the element, and a single root element contains all other elements in the document



# Introducing XML

**XML declaration**

Single root element (students)

Student, name, and photo elements and the id and grade attributes provide information about individual students

```
<?xml version="1.0" encoding="UTF-8" ?>
<students>
   <student id-"124-09" grade-"junior">
      <name>Tom Aaron</name>
      <photo>taaron.jpg</photo>
   </student>
   <student id="341-08" grade="senior">
      <name>Linda Wu</name>
      <photo>lkwu.jpg</photo>
   </student>
</students>
```

# Introducing XML

- Creating **XML vocabularies**
  - Although tags and attribute names may vary from one XML vocabulary to another, basic structure of a single root element, nested elements, and element attributes remains constant across all vocabularies
  - Typical **XHTML** document structure

```
<?xml version="1.0" encoding="UTF-8" ?>
<html>
   <head>
      head elements
   </head>
   <body>
      body elements
   </body>
</html>
```



# Introducing XML

- **Really Simple Syndication (RSS)** language
  - One of the more important XML vocabularies
  - Used to distribute news articles, or any content that changes on a regular basis, to a group of subscribers
  - Subscribers receive periodic updates using a **feed reader** or an **aggregator** (e.g., iTunes)
  - A **podcast** delivers audio or video content, but the language that organizes and describes it is RSS
  - RSS code follows conventions of all XML documents

# Building an RSS Newsreader

- XML documents and HTML documents share same document object model
  - Create element and attribute nodes, navigate through the node tree, and apply many of the same document object methods
- Not all JavaScript methods used for HTML documents are compatible with XML documents
- RSS documents can be retrieved from many sources; create a custom news feed object that can parse RSS documents
  - Add a constructor function
  - Create the custom items property
  - Create the parseToHTML() method of the RSSFeed object



# Building an RSS Newsreader

- Reading an XML document with AJAX
  - To retrieve an XML document object:

    var *xmlDoc* = *req*.responseXML
  - To reference a collection of elements within the XML document:

    var *elemArr* =
    *xmlDoc*.getElementsByTagName(*elem*)
  - To return text of an element in an XML document:

    *elemArr*[*i*].firstChild.nodeValue

# Transforming XML with XSLT

- **Extensible Stylesheet Language Transformation (XSLT)**
  - A style language developed for XML documents
  - Allows developers to easily transform contents of an XML document into another document format
  - A more efficient approach to transforming an XML document into HTML format than working with the document object model



# Introducing JSON

- **JSON (JavaScript ObjectNotation):** A data format based on the object literal format

```
{
   "name1": "value1",
   "name2": "value2",
   "name3": "value3"
...
}
```

- Advantage of JSON over XML
  - JSON is a terse language with a minimum of extraneous characters

# Introducing JSON

- JSON documents can contain several layers of nested values
- Size of a JSON data file can be further reduced through the use of array literals
- Following this general approach, can add more nested fields and arrays to achieve complicated data structures



# Introducing JSON

- Advantage of JSON over XML
  - Easier to retrieve specific data values with JSON; its data structures can be treated as custom JavaScript objects with data fields referenced as object properties
- Disadvantage of JSON over XML
  - Less readable by the layperson

# Introducing JSON

- Reading a JSON object with JavaScript
  - To retrieve a document containing a JSON object:

    *req*.responseText
  - To convert JSON text into a JavaScript object:

    var *jsonData* = eval( "(" + *req*.responseText + ")" );



# Working with a CGI Script

- A CGI script searches a collection of articles and returns the HTML code of articles that match a search keyword:

  sbarchives.cgi?skey=*keyword*
- The sbarchives.cgi script is written in **Perl**, a common server-side scripting language used to generate and manipulate text strings

# Working with a CGI Script

- Loading a Perl script to a Web server
  - Confirm you have an account on the Web server that can run CGI scripts
  - Determine location of Perl executable files within your account
  - Change first line of Perl script to reflect location of Perl folder
  - If using an external Web server, use FTP client or file transfer program to transfer script file to cgi-bin folder in root folder of your account; use ASCII file transfer
  - Change protection of script file on server to be executable by owner, owner's group, and the public

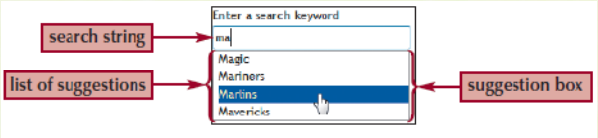


# Working with a CGI Script

- Interacting with a CGI script
  - Add an event handler to the newsfeed.js file to set up the properties of the search box
  - Create a request object named reqSearch that accesses the sbarchives.cgi file using the value of the sInput box as the skeyparameter value
  - Add code to define the reqSearch object

# Building an Autocomplete Input Box

- An autocomplete box provides suggestions to the user about possible values to enter into search boxes




- A CGI script named sbkeywords.cgi returns an array of suggested keywords giving a text string:

  sbkeywords.cgi?suggest=*substring*



# Building an Autocomplete Input Box

- Creating the suggestion box
  - Create a div element to store the suggestion box
  - Turn off browser's autocomplete feature
- Writing the event handler
  - Use setTimeout() method to add a 0.3 second delay between keyup event and submission of request object

# Building an Autocomplete Input Box

- The response from the sbkeywords.cgi script is a JSON object containing an array named searchResults
- Store this object in a variable named json:

  var json = eval("(" + this.responseText +")");
- To retrieve the array of suggested search terms:

  json.searchResults
- Elements in the generated HTML fragment for the suggestion box:

```
<div id="suggestBox">
   <div class="suggestion">jscn.searchResults[0]</div>
   <div class="suggestion">jscn.searchResults[1]</div>
   <div class="suggestion">jscn.searchResults[2]</div>
...
</div>
```



# Building an Autocomplete Input Box

- Autocomplete applications allow users to enter suggested keywords into search box with a mouse click
- Add an onclick event handler for each suggestion to the setupSearch()function
- Add code to toggle the className attribute in response to onmouseover and onmouseout events
- Suggestion box can be automatically hidden when user clicks the search icon

*New Perspectives on JavaScript and AJAX, 2nd Edition*

# AJAX Applications with jQuery

- **jQuery** is a lightweight JavaScript library that contains common DOM, event, and AJAX functions
- jQuery makes it easy to create AJAX applications
  - Simplest AJAX function offered is load() function – loads an HTML file into an element within the Web page
  - Also supports the $.ajax() function to exchange data from a Web server
- Writing AJAX requests in the jQuery library is quicker and more powerful than in native JavaScript code