AJAX
ASYNCHRONOUS JAVASCRIPT AND XML
Rich-client asynchronous transactions

• In 2005, Jesse James Garrett wrote an online article titled “Ajax: A New Approach to Web Applications” (www.adaptivepath.com/ideas/essays/archives/000385.php)

• This article outlined a technique that he referred to as Ajax, short for Asynchronous JavaScript+XML, consisting in making server requests for additional data without unloading the web page, for a better user experience

• Garrett explained how this technique could be used to change the traditional click-and-wait paradigm that the Web had been stuck in since its start
Synchronous (classic) web application model
Asynchronous web application model

- Web page, images, etc.
- AJAX engine communicates with the server
- Client works independently

- Data query
- Data response

- Web server
- Database

- Browser UI: application runs continuously (partial UI updates)
- Ajax engine: asynchronous data exchange with the server
- Server-side processing

4/21/2016
Example
Rich-client transactions

Client-side application → Client → Internet → Web server → Application → Database

Client-side application: Rich-client transactions

Client: Web server

Internet: TCP/IP, browser, application

Web server: command, send, param, HTML

Application: query, data

Database: DOM, events, runtime, browser, URL

DOM: display page, events

http: http & POST

TCP/IP: send

server: HTML

Application: query, data

Database: DOM, events, runtime, browser, URL

DOM: display page, events
Rich-client asynchronous transactions

Client-side application → Client → Internet → Web server → Application → Database

Client-side application

- DOM
- runtime
- events

Client

- URL
- display page
- browser
- http

Internet

- TCP/IP
- command
- send

Web server

- http
- HTML
- application

Application

- param
- query
- data

Database

- query

TCP/IP

- http

HTML

- display page
- events

DOM

- runtime

data

XML/JSON

4/21/2016
Adopted standards

• Dynamic HTML: DOM, Javascript, CSS
  – JavaScript
  – DOM (XHTML Document Object Model) to allow on-the-fly modification of the web page
  – CSS 3 to modify attribute and handle objects

• AJAX: Asynchronous Javascript and XML
  – XMLHttpRequest for asynchronous communication to the server
  – Data transfer formats: JSON, XML, RDF, RSS, Atom, FOAF, ...

• Mash-up technology
Rich-client transactions

Browser

Runtime

Web server

Application server

Database server

4/21/2016
Rich-client transactions

Web server (Apache, IIS)

HTML file

Web application (ASP, PHP, JSP, ...)

Database Server DBMS

File

URL

Browser

Display

Mouse/Keyboard

Browser

Layout engine

Javascript engine

DOM

Internet

Image file

Javascript

CSS stylesheets

CSS

Image file

HTML file

Database Server DBMS

Data base

4/21/2016

Ajax
Rich-client asynchronous transactions

4/21/2016
Asynchronous Javascript and XML

• AJAX is not a technology but group of inter-related technologies
• AJAX technologies include
  – HTML and CSS
  – DOM
  – JavaScript
  – XML or JSON (for carrying data to and from server)
  – XMLHttpRequest (for asynchronous communication between client and server)
• AJAX term coined in 2005 but
  – 1996: Iframe tag allows fetching content asynchronously
  – 1999: Microsoft introduced the XMLHttpRequest ActiveX in IE5, later adopted by all browsers as JS XMLHttpRequest
  – 2006: W3C draft specification of XMLHttpRequest
  – 2008: W3C draft on XMLHttpRequest 2 (now merged)
Other asynchronous tags

• How to load asynchronously (beside AJAX)?
• Asynchronous tags
  – `<img>` not really helpful for text data
  – Invisible `<iframe>`: inline frame, used to embed another document within the current HTML document
    <iframe src="demo_iframe.htm" width="200" height="200"></iframe>
  – `<script>` widely used
• Dynamic script tag injection
  – When the new `<script>` is added to the page, its “src” URL is automatically downloaded and executed.

```javascript
var script = document.createElement("script");
script.setAttribute("src", url);
document.head.appendChild(script);
```
Data exchange formats: XML and JSON

- There was a time when XML was the de facto standard for transmitting structured data over the Internet
  - But XML is a verbose and redundant language
- JSON (JavaScript Object Notation) is a light-weight data format, not a programming language

XML

```xml
<siblings>
  <sibling>
    <firstName>Anna</firstName>
    <lastName>Clayton</lastName>
  </sibling>
  <sibling>
    <firstName>Alex</firstName>
    <lastName>Clayton</lastName>
  </sibling>
</siblings>
```

JSON

```json
{  "employees":[   {"firstName":"John", "lastName":"Doe"},   {"firstName":"Anna", "lastName":"Smith"},   {"firstName":"Peter", "lastName":"Jones"}  ]}
```
“JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate” – JSON.org

Important: JSON is a subset of JavaScript

JSON is built on two structures

- A collection of name/value pairs: in various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array. { ... }
- An ordered list of values: in most languages, this is realized as an array, vector, list, or sequence. [ ... ]
JSON example

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
  },
  "phoneNumbers": [
    "212 555-1234",
    "646 555-4567"
  ]
}
```
JSON data structures

object

```
    string
       ^
    string
       ^
    value
       ^
    value
       ^
```

array

```
    value
       ^
    value
       ^
```

string

```
    Any UNICODE character except " or \ or control character
    ^
    quotation mark
    ^
    reverse solidus
    ^
    solidus
    ^
    backspace
    ^
    formfeed
    ^
    newline
    ^
    carriage return
    ^
    horizontal tab
    ^
    4 hexadecimal digits
    ^
```

value

```
    string
    ^
    number
    ^
    object
    ^
    array
    ^
    true
    ^
    false
    ^
    null
    ^
```

number

```
    digit
    ^
    digit
    ^
    digit
    ^
    digit
    ^
```

4/21/2016 Ajax
Asynchronous request/response

- What does asynchronous means?
  - The function (A) that sends the HTTP request returns before the response is received (does not wait)
  - Another function (B), the callback, is called when the browser gets the response
  - Attention: you must put the actions to do after the data is received after B, not after A
**XMLHttpRequest (XHR) object**

- Internet Explorer 5 was the first browser to introduce the XHR object
- Internet Explorer 7+, Firefox, Opera, Chrome, and Safari all support a native XHR object that can be created using the XMLHttpRequest constructor
  ```javascript
  var xhr = new XMLHttpRequest();
  ```

- To use an XHR object, the first step is to call the method `open()`, which accepts three arguments
  - The type of request to be sent ("get", "post", …)
  - The URL for the request
  - A Boolean value indicating if the request should be sent asynchronously
- `Open()` does not actually send the request, it simply prepares a request to be sent
  ```javascript
  // asynchronous GET request for example.php
  xhr.open("get", "example.php", true);
  ```
XMLHttpRequest (XHR) object

• To send the specified request, you must call the send() method
  – The send() method accepts a single argument, which is data to be sent as the body of the request
  – If no body data needs to be sent, null is required
• Once send() is called, the request is dispatched to the server
• If the request is synchronous, the JavaScript code will wait for the response to return before continuing execution

```javascript
xhr.open("get", "example.php", false);
xhr.send(null);
```
XMLHttpRequest (XHR) object

• When a response is received, the XHR object properties contain useful data
  – responseText: the text that was returned as the body of the response
  – responseXML: contains an XML DOM document with the response data if the response has a content type of “text/xml” or “application/xml”
  – status: the HTTP status of the response
  – statusText: the description of the HTTP status

• When a response is received, the first step is to check the status property to ensure that the response was returned successfully
  – Generally, HTTP status codes in the 200s are considered successful

```javascript
if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304){
    alert(xhr.responseText);
} else { alert("Request was unsuccessful: " + xhr.status); }
```
XMLHttpRequest (XHR) object

• Although it’s possible to make synchronous requests, most of the time it’s better to make asynchronous requests that allow JavaScript code execution to continue without waiting for the response.

• The XHR object has a readyState property that indicates what phase of the request/response cycle is currently active:
  - 0 — Uninitialized: the open() method hasn’t been called yet
  - 1 — Open: the open() method has been called but send() has not been called
  - 2 — Sent: the send() method has been called but no response has been received
  - 3 — Receiving: some response data has been retrieved
  - 4 — Complete: all of the response data has been retrieved and is available
XMLHttpRequest (XHR) object

- Whenever the readyState changes from one value to another, the readystatechange event is fired
  - Opportunity to check the value of readyState with an onreadystatechange event handler

```javascript
var xhr = createXHR();
xhr.onreadystatechange = function(){
  if (xhr.readyState == 4){
    if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304){
      document.getElementById('span_result').innerHTML = xhr.responseText;
    } else {
      alert("Request was unsuccessful: " + xhr.status);
    }
  }
};
xhr.open("get", "example.txt", true);
xhr.send(null);
```
GET requests

• The most common type of request to execute is a GET, which is typically made when the server is being queried for some sort of information
  – If necessary, query-string arguments can be appended to the end of the URL to pass information to the server
  – For XHR, this query string must be present and encoded correctly on the URL that is passed into the open() method

```javascript
xhr.open("get", "example.php?name1=value1&name2=value2", true);
```
POST requests

• The second most frequent type of request is POST, which is typically used to send data to the server that should save data
  – The body of a POST request can contain a very large amount of data, and that data can be in any format
• `setRequestHeader(header, value):` adds HTTP headers to the request
  – `Content-Type` indicates to the server the type of data (MIME type) you are sending in the request body
  – `setRequestHeader('Content-Type', 'application/json')` to send a JSON string to the server

```javascript
xhr.open("post", "postexample.php", true);
xhr.setRequestHeader("Content-Type", "application/json");
```
License

• This work is licensed under the Creative Commons “Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 3,0)” License.
• You are free:
  – to Share - to copy, distribute and transmit the work
  – to Remix - to adapt the work
• Under the following conditions:
  – Attribution - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  – Noncommercial - You may not use this work for commercial purposes.
  – Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
• To view a copy of this license, visit http://creativecommons.org/license/by-nc-sa/3.0/