AJAX

Objectives

- Explain how modern websites use AJAX technology to provide a desktop like experience
- Differences between AJAX and non-AJAX web pages
- Using AJAX in JavaScript
- Using JSON for handling AJAX responses
- AJAX security issues

AJAX Introduction and History

Introduction

Sites like Gmail, Google Maps and Flickr behave more like desktop applications than web sites

These sites are powered by AJAX

AJAX is **Asynchronous JavaScript and XML**

Jesse James Garrett coined the term in 2005

Ajax History

Goal of Ajax:

Provide Web-based applications with responsiveness approaching that of desktop applications

Web applications that benefit from Ajax:

Those that have **frequent interactions** between the client and the server

AJAX Technologies

AJAX Technologies

AJAX uses the **`XMLHttpRequest`** to transfer data between client and server

XML is used as the form for the data flowing between client and server

JavaScript is used to dynamically display and interact with all of the information

Traditional Web Applications vs. Ajax Applications

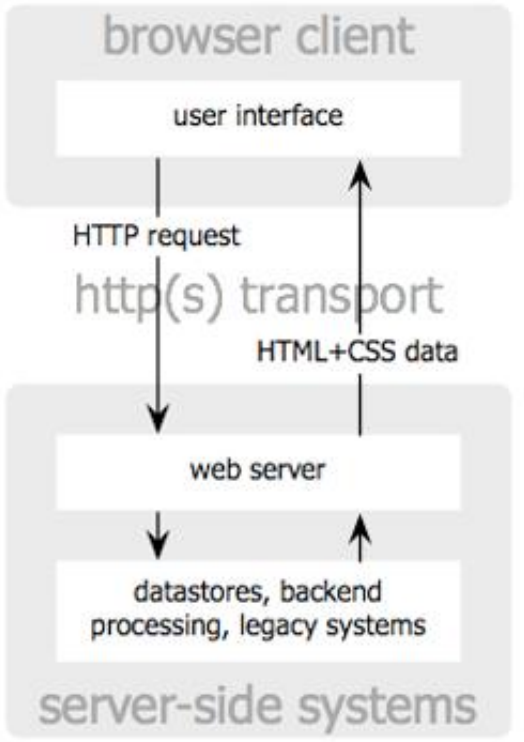
Traditional web applications user interactions:

- Send data/queries to web servers
- Wait for the server to respond
- Refresh the page

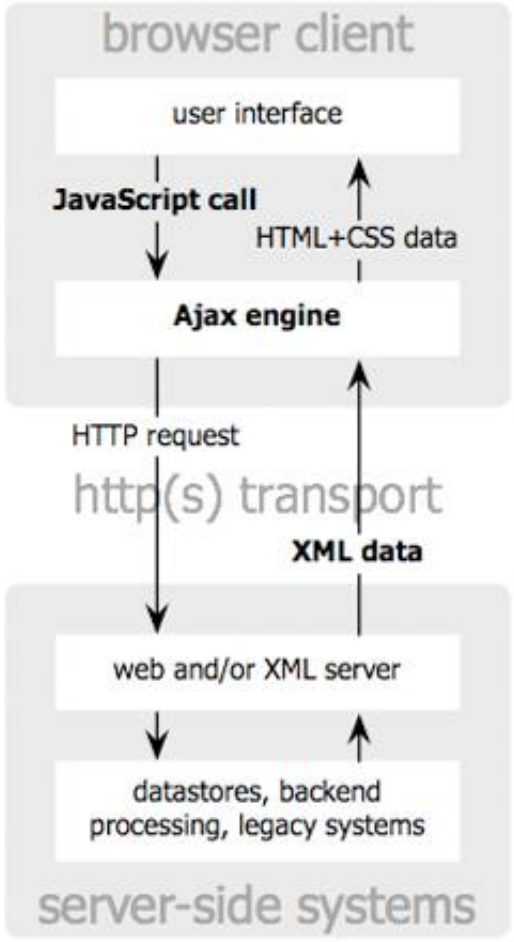
Ajax Applications

- The `XMLHttpRequstObject` sends requests to the server
- The ***callback function*** updates the page with the requested data arrives

Traditional Web Applications vs. Ajax Applications



classic
web application model

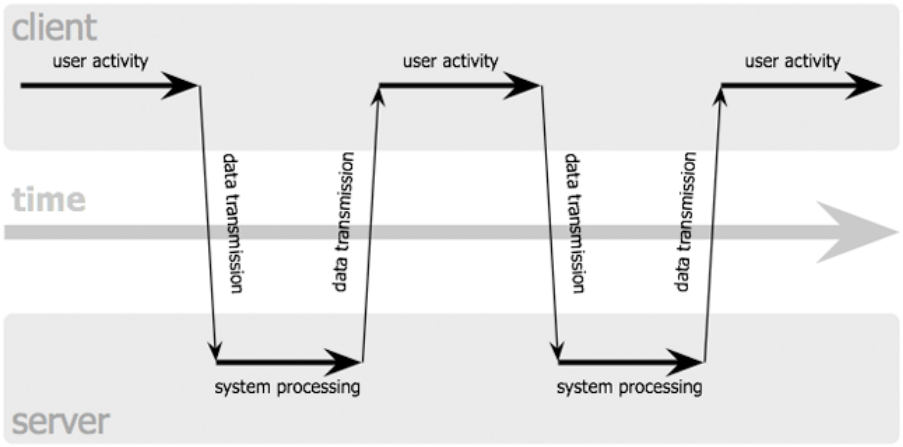


Ajax
web application model

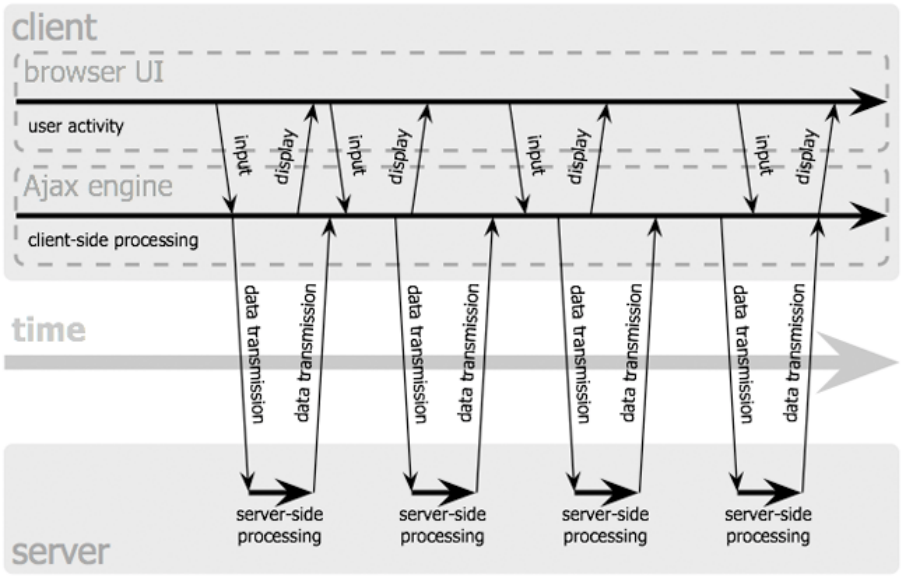
Jesse James Garrett / adaptivepath.com

Traditional Web Applications vs. Ajax Applications

classic web application model (synchronous)



Ajax web application model (asynchronous)



Jesse James Garrett / adaptivepath.com

Programming using AJAX

Instantiating an asynchronous request

Create an instance of XMLHttpRequest object

```
xhr = new XMLHttpRequest();
```

Register its `onreadystatechange` event handler

```
xhr.onreadystatechange = showContents;
```

Use its `open` method to set up the request

```
xhr.open("GET", url, true);
```

Use its send method to initiate the request

```
xhr.send(null);
```

Callback function

Progress of the request is monitored by the `readystate` property

The status of the HTTP request is contained in the `status` property

`readyState` Property Values

0: Uninitiated; object contains no data

1: Loading: object is currently loading its data

2: Loaded; object has finished loading its data

3: Interactive; user may interact with the object even though its is not fully loaded

4: Complete; object has finished initializing

HTTP status codes

200: the request was successful

404: the requested resource was not found

500: there was an error while the server was processing the request

Properties of the `XMLHttpRequest` Object

`onreadystate`

`readystate`

`responseText`

`responseXML`

`status`

`statusText`

Methods of the `XMLHttpRequestObject`

```
open

send

setRequestHeader

getResponseHeader

getAllResponseHeaders

abort
```

Ajax Example

Example application: Help the user fill a form

- The form gathers client information - asks for the zip code before the names of the city and state
- As soon as the zip code is entered, the application sends a request to the server, which looks up the city and state for the given zip code and returns them to the form
  - Must register an event handler on the blur event of the zip code text box
- Uses JavaScript to put the city and state names in the form
- Uses **PHP** on the server to look up the city and state

Popcorn Example [link]

End of Session

Course Number: CPSC-24700

Instructor: Eric Pogue

Handling Returned Data

Return Document Forms

The returned data from the asynchronous has to be integrated into the HTML document

If the data is **HTML**, then most common approach is to place an empty `div` element in the original document

The **`innerHTML`** property of the `div` element is assigned the new content

Return Document Forms

Example:

```
<div id = "replaceable_list">
  <h2> 2007 US Champion/Runnerup - baseball </h2>
  <ul>
    <li> Boston Red Socks </li>
    <li> Colorado Rockies </li>
  </ul>
</div>
```

Now, if the user selects a different sport, say football, the HTML response fragment could have the following:

```
<h2> 2007 US Champion/Runnerup - football </h2>
<ul>
 <li> New York Giants </li>
 <li> New England Patriots </li>
</ul>
```

Return Document Forms

The returned fragment can be inserted in the `div` element with

```
var divDom =
     document.getElementById("replaceable_list");
divDom.innerHTML = xhr.responseText;
```

The disadvantage of using HTML for the return document is it works well only if markup is what is wanted.

However, oftentimes, it is **data that is returned**, in which case it must be **parsed out of the HTML**

Return Document Forms

The returned data could also be XML

For the previous example, the following would be returned:

```
<header> 2007 US Champion/Runnerup - football
</header>
<list_item> New York Giants </list_item>
<list_item> New England Patriots </list_item>
```

Problem: the **XML returned must also be parsed**.

Return Document Forms

Two approaches:

- Use the DOM binding parsing methods
- This has two disadvantages:
  - » Writing the parsing code is **tedious**
  - » Support for DOM parsing methods is a bit inconsistent over various browsers

- Use XSLT style sheets

Using JSON

JSON

Due to the issues with having to parse HTML or XML, ***JavaScript Object Notation (JSON)*** is often used

Part of the JavaScript standard, 3rd edition

It is a method of representing objects as strings, using two structures:

- Collections of name/value pairs
- Arrays of values

JSON

Each object is represented as a list of property names and values contained in curly braces, e.g.:

```
{"propName1" : value1, "propName2" : value2}
```

Arrays are represented in JSON using square brackets, e.g.: `[value1, value2, value2]`

Values can be string, number, JSON representation of an object, true, false or null

JSON Example

Example

```
{"employees" :
  [
    {"name" : "Dew, Dawn", "address" :
      "1222 Wet Lane"},
    {"name" : "Do, Dick", "address" :
      "332 Doer Road"},
    {"name" : "Deau, Donna", "address" :
      "222 Donne Street"}
  ]
}
```

This object consists of **one property/value pair**, whose value is an **array of three objects**, each with **two property/value pairs**

JSON Example

Array element access can be used to retrieve the data elements, e.g.:

```
var address2 = myObj.employees[1].address;
```

puts "332 Doer Road" in *address2*

Parsing JSON

JSON objects are returned in **`responseText`**

We can convert JSON strings into JavaScript objects with the **`eval`** function

Note: this creates a potential security risk

Use a JSON parser instead:

```
var response = xhr.responseText;
var myObj = JSON.parse(response);
```

Parsing JSON

Two reason JSON strings are commonly used to communicate in client/server interaction:

1. JSON strings are easier to create and parse than XML
2. JSON strings require fewer bytes

Parsing JSON

Example return document

```
{"top_two":
  [
    {"sport": "football", "team":
                        "New York Giants"},
    {"sport": "football", "team":
                        "New England Patriots"},
  ]
}
```

Parsing JSON

The processing to put it in the HTML document:

```
var myObj = JSON.parse(response);
document.write("<h2> 2007 US Champion/Runnerup"
       + myObj.top_two[0].sport + "</h2>");
document.write("<ul> <li>" +
       myObj.top_two[0].team + "</li>");
document.write("<li>" + myObj.top_two[1].team
       + "</li></ul>");
```

Ajax Toolkits

Writing Ajax applications can difficult - need to know a lot about the DOM, CSS, JavaScript…

Ajax toolkits are libraries of functions that make it easy for you to add Ajax functionality to your web pages

Ajax Tookits

- Dojo
- Yahoo! User Interface Library
- Google Web Toolkit
- JQuery
- Prototype

AJAX Security Issues

Security and Ajax

Some security issues to keep in mind when using Ajax:

Non-Ajax applications often have just one or only a few server-side sources of responses

Ajax applications often have **many server-side programs** that produce small amounts of data.

This increases the attack surface of the whole application

Security and Ajax

Therefore Ajax developers should put **security code in the client code**

Also, **must include in the server code**, because intruders can change the code on the client

The common problem comes from **cross-site scripting**

Security and Ajax

Cross-site scripting means servers providing JavaScript code as an Ajax response

- Such code could be modified by an intruder before it is run on the client
- All such code must be scanned before it is interpreted

Intruder code could also come to the client from text boxes used to collect return data

- It could include script tags with malicious code

Summary

- AJAX is Asynchronous JavaScript and XML
- Goal of Ajax is to provide Web-based applications with responsiveness approaching that of desktop applications
- AJAX uses the XMLHttpRequest to transfer data between client and server
- The callback function updates the page with the requested data arrives
- Due to the issues with having to parse HTML or XML, JavaScript Object Notation (JSON) is often used
- JSON objects are returned in responseText which are then parsed by a JSON parser
- Ajax toolkits are libraries of functions that make it easy for you to add Ajax functionality to your web pages
- Cross-site scripting is a security vulnerability that involves servers providing JavaScript code as an Ajax response