Ajax – Challenges and Trends

Atul Kahate

Head – Technology Practice - Oracle Financial Services Software Limited (*formerly* i-flex solutions limited*)*

The Problems, First!

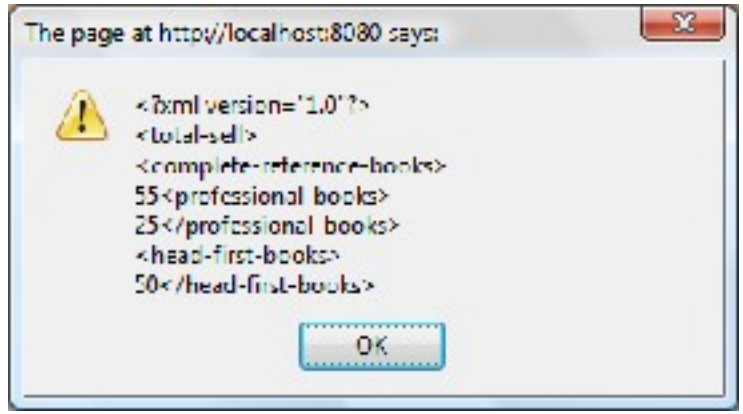
Bad MIME Types

- Server-side code does not set the MIME type to *text/xml*
  - Browser does not know what to do with it!

- Example (D:\tomcat\webapps\ajax\books-xml-version\booksXMLWithoutCorrectMIMEType.html)
- URL: http://localhost:8080/ajax/books-xml-version/booksXMLWithoutCorrectMIMEType.html

Not Well-Formed XML

- Server sends an XML that is not well-formed
  - Browser cannot parse it!



- Example (D:\tomcat\webapps\ajax\books-xml-version\getUpdatedBookSalesInvalidXML.html)

Invalid XML

- Server sends an invalid XML (does not conform to a DTD/schema)
  - Browser handles it happily!
  - XHR object does not have any built-in mechanisms for validating XML – defeats one of the basic purposes of XML!
  - We must explicitly do some work using DOM parsers etc

Encoding/Decoding Requests

- How should we encode data to be sent from the browser to the server?
  - Traditionally, the browser handles this automatically by converting it into *x-www-form-urlencoded* name-value pairs (both in GET and POST) - *See next slide*
  - This can be handled on the server side easily, since that is the default
- Using Ajax may mean usage of other encoding formats – how to send and receive this?

Standard Encoding

- x-www-form-urlencoded
  - Standard format for data transmission from browser to server
  - Name-value pairs, separated by &
  - Example: GET /login.jsp?username=atul&password=atul HTTP/1.1
  - JavaScript provides a *encodeURIComponent ()* method to convert existing data into *URL encoded* format
  - May not be handled automatically in standard Ajax libraries

Encoding/Decoding Responses

- How should we encode data to be returned from the server to the browser?
  - Traditionally, the server sends XHTML data back to the browser
  - With Ajax involved, it needs to be changed to other formats, such as XML, JSON, etc

Security Considerations

- Should data be sent back and forth in plain text?
  - Alternatives: Use Base-64 encoding to transform plain ASCII into non-understandable ASCII-like syntax – *See next slide*
  - Perform some basic encryption
- Best is to use SSL, instead of such “visual encryption” techniques

Base-64 Example

Plain text in ASCII	After Base-64 conversion
Hi everyone, I am ASCII - getting converted into base-64.	SGkgZXZlcnlvbmUsIEkgYW0gQVNDSUkgLSBnZXR0aW5nIGNvbnZlcnRlZCBpbnRvIGJhc2UtNjQu

Transmission Considerations

- XML is quite bulky

Plain text	XML
Value-1, Value-2, …, Value-N	<?xml version = “1.0”?> <data_items> <item>Value-1</item> <item>Value-2</item> … <item>Value-N</item> </data_items>

Networking Considerations

What could go wrong?

- A request never returns
- A request returns, but too slowly
  - Server's response comes too slowly for the client to make any use of it
  - Employ timeouts and abort/retry the request after a specified time period has elapsed
- A request returns, and is in error
  - Client-induced errors (e.g. wrong URL or wrong parameters)
  - Server-side errors (e.g. permission problem)

Dealing with Multiple Requests

- More than one Ajax request could be sent
- Utilizes the true power of Ajax, but can also cause confusions
- From HTTP 1.1 specifications
  - A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.
- As a result, if we send more than two concurrent Ajax requests, the first must complete before others receive any attention

Server availability

- Since HTTP is stateless, the only way to check if a connection is still usable is to actually use it!
- One approach is to keep pinging the server every few seconds to see if it is alive, and then sending the request
- Impractical in many situations, but unavoidable in a few ones (e.g. chatting)

Client Availability

- Server also needs to know if client is active! – *Same problem of HTTP being connectionless occurs here.*
- Normally, server knows the presence of clients because clients keep returning cookies back to the server
- In Ajax, we may inform the server that the client is active, by capturing events and sending them to the server (e.g. mouse movement, key pressing, etc)
- Approach similar to the previous one, except that here the client is informing the server that it is alive, not checking if the server is still active

Connection Rates

- Even if a connection is up and running, there are chances that it is quite slow
- In some situations, it may be a better idea to check the user's connection speed and then send appropriate (amount of) data to the user
  - Example: d:\tomcat\webapps\examples\calculatePageLoadTime.html

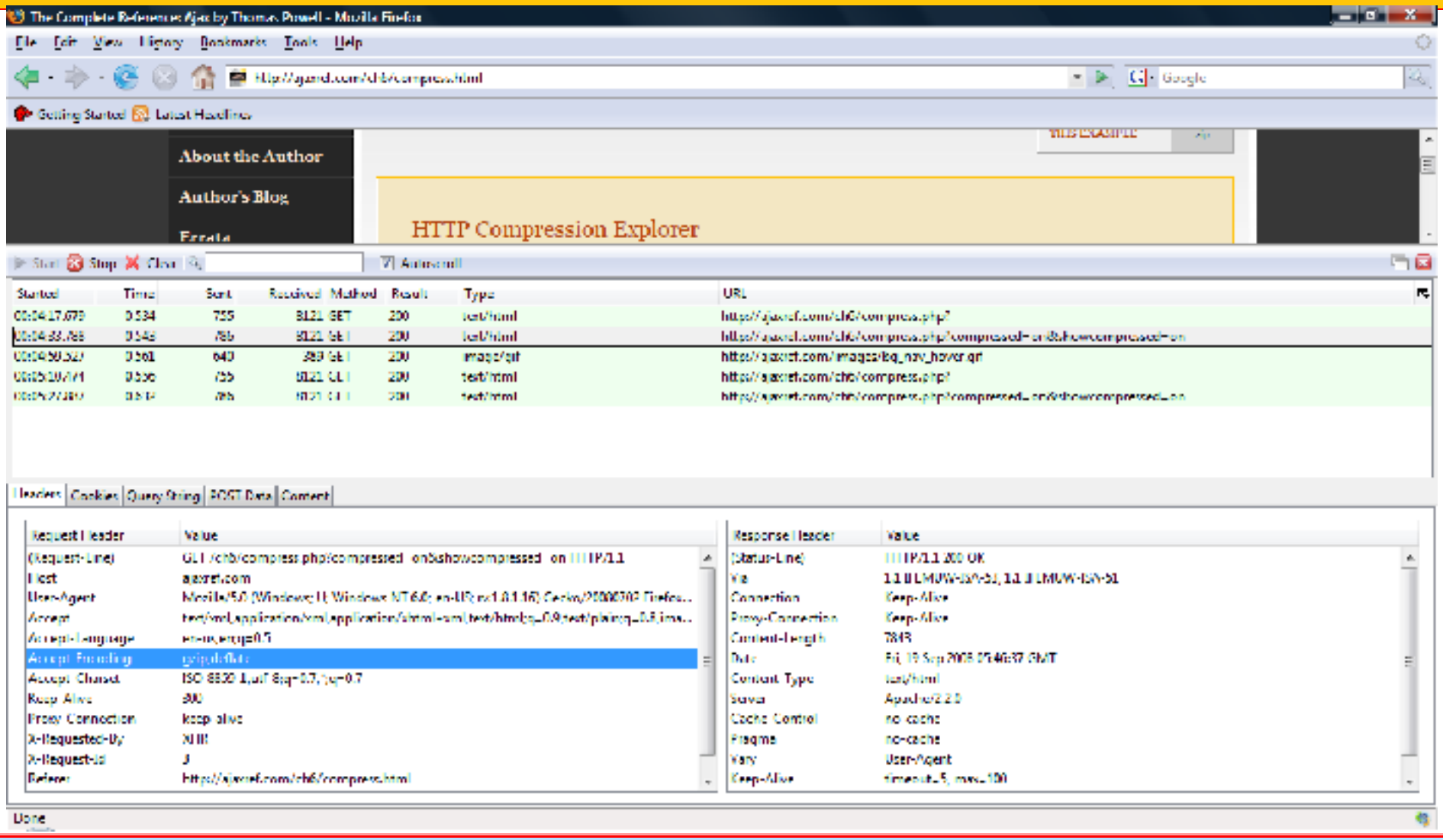
Improving Ajax Performance

Golden Rule

- “Send little, less often”
- Send as little data as required and do not ask for more data or re-request unless we need to
- Leads to two aspects
  - Compression
  - Caching

HTTP Compression - 1

- Web standard for compressing various kinds of payloads sent over HTTP (HTML, XML, CSS, JavaScript, etc)
- Up to 80% of transmission requirements can be reduced by this
- Remember that the browser has an *Accept-Encoding* header, which can also have *qzip and deflate* as one of the options (*see next slide)*



HTTP Compression - 2

- HTTP compression is transparent – user does not even realize it, but the contents travel much faster from the server to the browser
- Involves more CPU cycles on the Web server (however, if it becomes too much, most commercial implementations of HTTP stop compression)
- We can use
  - *Apache: mod_gzip* or *deflate built-in*
  - IIS: httpZip

Content Optimization - Do's

- Remove unnecessary white spaces from your code (HTML, JavaScript, CSS)
- Remove comments
- Remap colour values to their smallest forms (e.g. instead of #ff0000, use *red*)
- Remove useless tags (e.g. unnecessary <*meta*> tags)

Content Optimization - Don't's

- Remove quotes around attributes
- Remove implicit attributes (e.g. do not just say *<script>* instead of *<script type = "text/javascript">*)
- Eliminate *doctype* declaratiobs
- Remove optional closing tags
- Substitute longer tags with shorter ones (e.g. instead of **, using *<b>*)

CSS Optimizations

- Remove white space
- Remove comments
- Remap colours
- Combine, reduce, and remove CSS rules
  - Example – compare the two declarations

```
p {
  font-family: arial;
  line-height: 48pt;
  font-weight: bold;
}
```

```
p {font-size: 36pt; font: bold 36pt/48pt arial;}
```

JavaScript Optimizations

- Remove comments
- Remove white space and look for code optimizations (e.g. instead of *x = x + 1*, use *x++*)
- Change variable and function names to smaller ones
- Be aware that non-usage of semicolons can cause problems
  - E.g.

    x = x + 1

    y = y + 1
  - A simple white space remover tool will change this to

    x = x + 1 y = y + 1

DOM-JavaScript Related Issues

- (Source: http://www.coachwei.com/blog/_archives/2008/1/22/3480119.html)

1. Arrays operations in JavaScript are very slow (*10 to 100 times slower than the other operations*)
2. HTML DOM operations are slow (See next slide)
3. The *eval* operation is very slow in Firefox
4. Creating new objects (e.g. var myObject = new MyObject (10)) are very expensive in Firefox
5. IE string operations are very slow

DOM Operations Speed

	IE7		FireFox		Safari	
HTML DOM Operation	Time(us)	%	Time(us)	%	Time(us)	%
Change text using innerHTML	469.0	9979%	234.0	6000%	109.0	7032%
Create a text node on HTML Dom	1093.0	23255%	156.0	4000%	110.0	7097%
Change the class name of an element	422.0	8979%	47.0	1205%	109.0	7032%
getElementById	86.8	1846%	15.7	402%	3.9	252%
getElementsByTagName("div")	153.1	3257%	18.0	462%	5.5	352%
getElementsByName	93.8	1995%	44.6	1142%	4.7	303%
placeDiv.getAttribute("id")	29.7	632%	46.8	1200%	5.5	352%
placeDiv.attributes["id"]	31.3	666%	225.0	5769%	6.3	403%

Security Concerns

JavaScript Security

- Obfuscation: Concealing the meaning
- Same-origin policy: Scripts loaded from one Web site cannot get/set properties of a document loaded from a different site

HTTP Authentication Basics

- Traditional authentication approaches
  - Built-in HTTP-based authentication provided by browsers
    - Basic
    - Digest
  - Custom authentication
    - Usage of cookies or forms

Basic Authentication - Problem

- *XMLHttpRequest* object allows us to send the user ID and password to the server for authenticating the user in the *open* method

```
xhr.open (request.method, request.url, request. async,
    request.username, request.password);
```

- Causes the user ID and password to be sent in clear text (look for an HTTP header by the name *Authorization:Basic* in the HTTP request sent to the server for a protected resource – you will get the Base-64 encoded values of user ID and password, which can be decoded back easily)

Basic Authentication - Solutions

- Use digest authentication – Browser automatically computes hash of the password and sends it to the server
- Use SSL – Would make things slow!
- Use custom authentication – Store digested password in a cookie and use SSL

Back Button Problems

Back Button Problem - Basics

- Users use the *back button* as if it is an *undo* action
- Research says that over 40% of all clicks in a Web browser are for the *back button*
- Why does it not work in Ajax?
  - Ajax does not actually change the URL or update a browser's navigation
  - State-changing hyper links are used instead (i.e. on the same page, several actions happen – they are not cached by the browser)
  - When a user clicks the *back* button, the user expects to see the state before *his/her* previous action – but because of using Ajax, the user goes to the actual previous page cached by the browser

Back Button Problem - Solutions

- Hash method
  - Does not work in Internet Explorer
  - Incompatible – so, ignored here
- Use of iFrame
  - Every time the user takes an action, load an invisible iFrame with a distinct URL in the browser
  - Forces the browser to load the page into the browser cache

Auto Commit

User Behavior in Transaction Processing

- In Web applications, users do not expect the *auto commit* behavior
- For example, when a user enters something in an *amount* field, the user expects to click on some button say *Submit*) to complete the transaction
- Ajax is the opposite – without the user knowing about it, an asynchronous request may be sent to the server, and committing the data as well!
- Hence, there is an explicit need to add some confirmation screen/button without annoying the user

Questions/Comments?

Thank You!