Introduction to Ajax

Sang Shin
Java Technology Architect
Sun Microsystems, Inc.
sang.shin@sun.com
www.javapassion.com

Agenda

1. What is Rich User Experience?
2. Rich Internet Application (RIA) Technologies
3. AJAX: Real-life examples & Usage cases
4. What is and Why AJAX?
5. Technologies used in AJAX
6. Anatomy of AJAX operation
7. XMLHttpRequest Methods & Properties
8. DOM APIs & InnerHTML
9. AJAX Security
10. JavaScript debugging tools

Topics Covered in Other Presentations

- AJAX Toolkits & Frameworks
- JSON (JavaScript Object Notation)
- Dojo Toolkit
- DWR (Direct Web Remoting)
- AJAX-enabled JSF Components
- Google Web Toolkit (GWT)
- jMaki
- Phobos (MVC-based server-side scripting)
- Ajax and Portlet/Portal
- Wicket and Shale (as AJAX-aware Web applicaion frameworks)
- JavaScript Programming Best Practices

1. Rich User Experience for Web Application

Rich User Experience

- Take a look at a typical desktop application (Spreadsheet app, etc.)
- The program responses intuitively and quickly
- The program gives a user meaningful feedback's instantly
  - > A cell in a spreadsheet changes color when you hover your mouse over it
  - > Icons light up as mouse hovers them
- Things happen naturally
  - > No need to click a button or a link to trigger an event

Characteristics of Conventional Web Applications (Apps without Ajax)

- “Click, wait, and refresh” user interaction
  - > Any communication with the server forces a page refresh
- Synchronous “request/response” communication model
  - > The user has to wait for the response
- Page-driven: Workflow is based on pages
  - > Page-navigation logic is determined by the server

Issues of Conventional Web Application

- Interruption of user operation
  - > Users cannot perform any operation while waiting for a response
- Loss of operational context during refresh
  - > Loss of information on the screen
  - > Loss of scrolled position
- No instant feedback's to user activities
  - > A user has to wait for the next page
- Constrained by HTML
  - > Lack of useful widgets

These are the reasons why Rich Internet Application (RIA) technologies were born.

2. Rich Internet Application (RIA) Technologies

Rich Internet Application (RIA) Technologies

- Applet
- Macromedia Flash/Air
- Java WebStart
- DHTML
- DHTML with Hidden IFrame
- Ajax
- Sliverlight (Windows only)
- JavaFX (Java Platform)

Applet

- Pros:
  - > Can use full Java APIs
  - > Custom data streaming, graphic manipulation, threading, and advanced GUIs
  - > Well-established scheme
- Cons:
  - > Code downloading time could be significant
  - > Reliability concern - a mal-functioning applet can crash a browser
- There is renewed interest in applet, however, as a RIA technology with Java SE 10 Update 10
  - > Solves old applet problems

Macromedia Flash

- Designed for playing interactive movies originally
- Programmed with ActionScript
- Implementation examples
  - Macromedia Flex
  - Laszlo suite (open source)
- Pros:
  - Good for displaying vector graphics
- Cons:
  - Browser needs a Flash plug-in
  - ActionScript is proprietary

Java WebStart

- Desktop application delivered over the net
  - > Leverages the strengths of desktop apps and applet
- Pros
  - > Desktop experience once loaded
  - > Leverages Java technology to its fullest extent
  - > Disconnected operation is possible
  - > Application can be digitally signed
  - > Incremental redeployment
- Cons
  - > Old JRE-based system do not work
  - > First-time download time could be still significant

DHTML (Dynamic HTML)

- DHTML = JavaScript + DOM + CSS
- Used for creating interactive applications
- No asynchronous communication, however
  - > Full page refresh still required
  - > Reason why it has only a limited success

DHTML with Hidden IFrame

- IFrame was introduced as a programmable layout to a web page
  - An IFrame is represented as an element of a DOM tree
  - You can move it, resize it, even hide it while the page is visible
- An hidden IFrame can add asynchronous behavior
  - The visible user experience is uninterrupted – operational context is not lost
- It is still a hack

AJAX

- DHTML plus Asynchronous communication capability through XMLHttpRequest
- Pros
  - > Emerges as a viable RIA technology
  - > Good industry momentum
  - > Several toolkits and frameworks are emerging
  - > No need to download code & no plug-in required
- Cons
  - > Still some browser incompatibility
  - > JavaScript is hard to maintain and debug
- AJAX-enabled JSF components will help

3. AJAX:
Real-life Examples & Usecases

Real-Life Examples of AJAX Apps

- Google maps
  - > http://maps.google.com/
- Goolgle Suggest
  - > http://www.google.com/webhp?complete=1&hl=en
- NetFlix
  - > http://www.netflix.com/BrowseSelection?lnkctr=nmhbs
- Gmail
  - > http://gmail.com/
- Yahoo Maps (new)
  - > http://maps.yahoo.com/
- Many more are popping everywhere

AJAX: Demo Google Maps, Yahoo Maps New

Key Aspects of Google Maps

- A user can drag the entire map by using the mouse
  - Instead of clicking on a button or something
  - The action that triggers the download of new map data is not a specific click on a link but a moving the map around with a mouse
- Behind the scene - AJAX is used
  - The map data is requested and downloaded asynchronously in the background
- Other parts of the page remains the same
  - No loss of operational context

Usage cases for AJAX

- Real-time server-side input form data validation
  - > User IDs, serial numbers, postal codes
  - > Removes the need to have validation logic at both client side for user responsiveness and at server side for security and other reasons
- Auto-completion
  - > Email address, name, or city name may be auto-completed as the user types
- Master detail operation
  - > Based on a user selection, more detailed information can be fetched and displayed

Usage cases for AJAX

- Advanced GUI widgets and controls
  - > Controls such as tree controls, menus, and progress bars may be provided that do not require page refreshes
- Refreshing data
  - > HTML pages may poll data from a server for up-to-date data such as scores, stock quotes, weather, or application-specific data

Demo: AJAX Sample Apps

javapassion.com/handsonlabs/ajaxbasics2/#Exercise_1

Demo Scenario

- Run sample AJAX applications within NetBeans IDE
  - > Auto completion
  - > Data validation
  - > Progress bar
- You can try this demo yourself
  - > These applications are provided as part of the hands-on lab.
  - > www.javapassion.com/handsonlabs/4257_ajaxbasics2.zip

4. AJAX:
What is and Why AJAX?

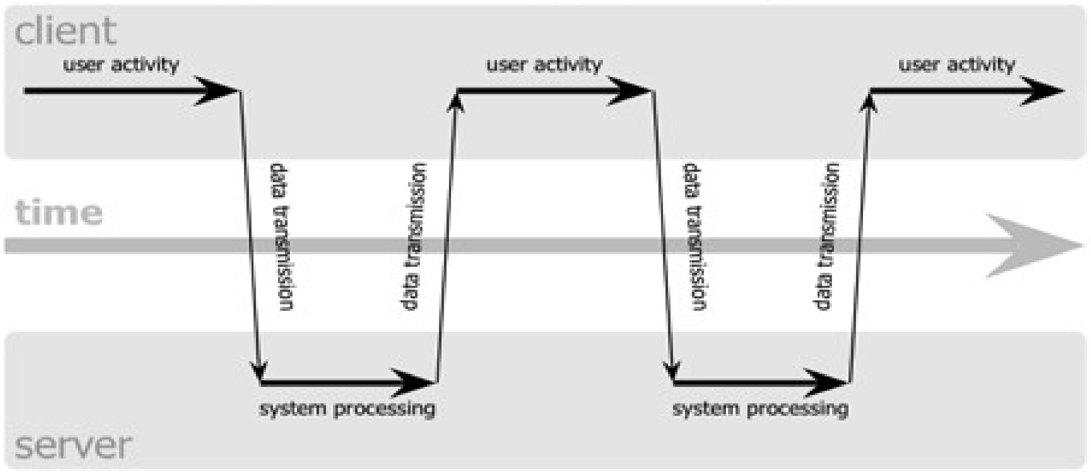
Why AJAX?

- Intuitive and natural user interaction
  - No clicking required
  - Mouse movement is a sufficient event trigger
- "Partial screen update" replaces the "click, wait, and refresh" user interaction model
  - Only user interface elements that contain new information are updated asynchronously (no interruption to user operation)
  - The rest of the user interface remains displayed without interruption (no loss of operational context)
- Data-driven (as opposed to page-driven)
  - UI is handled in the client while the server provides data

Why AJAX?

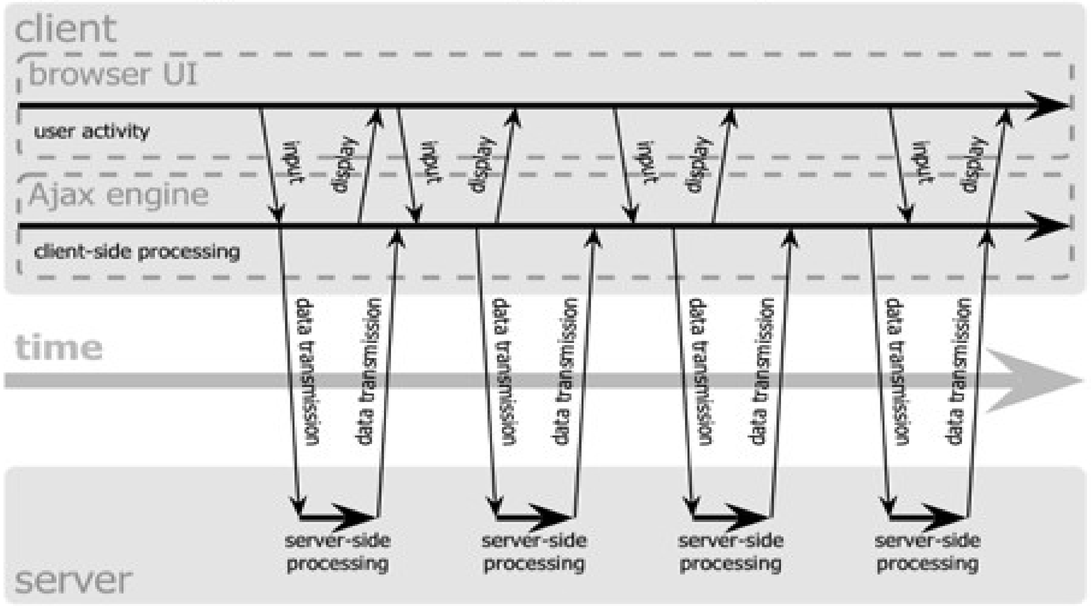
- Asynchronous communication replaces "synchronous request/response model."
  - A user can continue to use the application while the client program requests information from the server in the background
  - Separation of displaying from data fetching

classic web application model (synchronous)

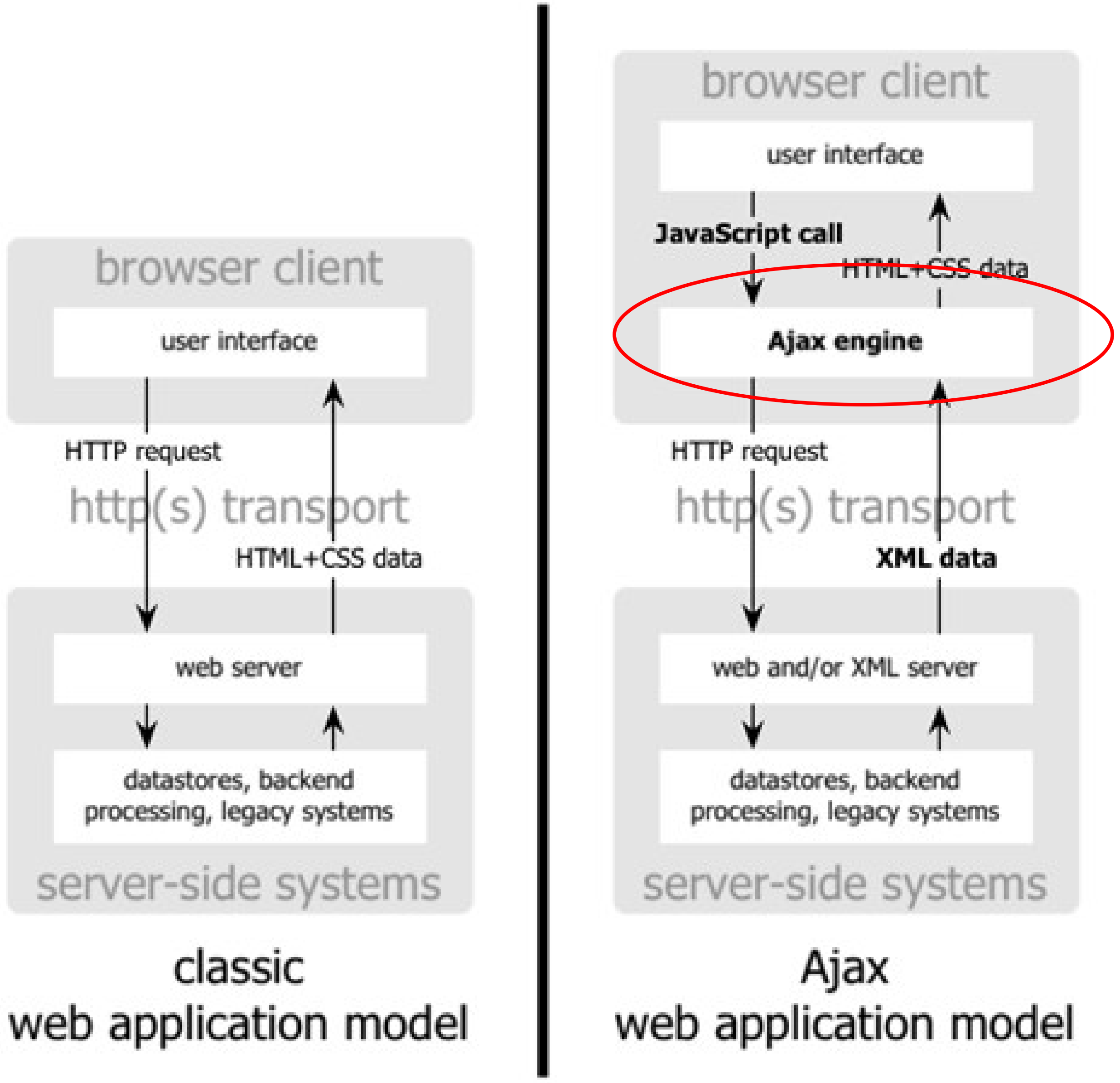


<span style="color:red">Interrupted</span> user operation while the data is being fetched

Ajax web application model (asynchronous)



Uninterrupted user operation while data is being fetched


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend processing, legacy systems
server-side systems
classic web application model
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend processing, legacy systems
server-side systems
Ajax web application model

5. AJAX: Technologies Used in AJAX

Technologies Used In AJAX

- Javascript
  - > Loosely typed scripting language
  - > JavaScript function is called when an event in a page occurs
  - > Glue for the whole AJAX operation
- DOM
  - > Represents the structure of XML and HTML documents
  - > API for accessing and manipulating structured documents
- CSS
  - > Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript
- XMLHttpRequest
  - > JavaScript object that performs asynchronous interaction with the server

XMLHttpRequest

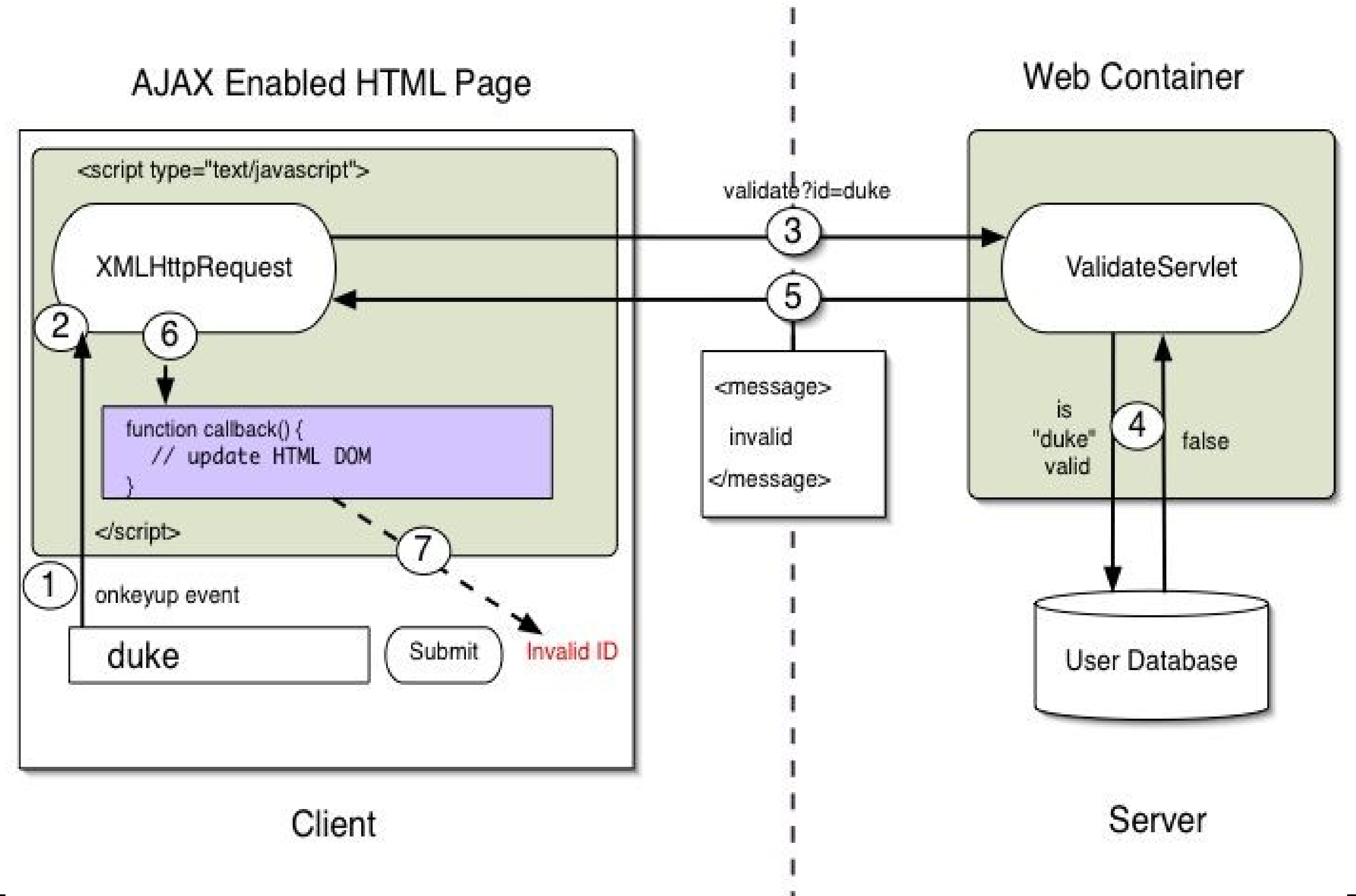
- JavaScript object
- Adopted by modern browsers
  - Mozilla™, Firefox, Safari, and Opera
- Communicates with a server via standard HTTP GET/POST
- XMLHttpRequest object works in the background for performing asynchronous communication with the backend server
  - Does not interrupt user operation

Server-Side AJAX Request Processing

- Server programming model remains the same
  - It receives standard HTTP GETs/POSTs
  - Can use Servlet, JSP, JSF, whatever web technologies...
- With minor caveats
  - Could have more frequent and finer-grained requests from clients (design issue)
  - Response content type can be
    - text/xml
    - text/plain
    - text/json
    - text/javascript

6. AJAX: Anatomy Of AJAX Interaction using “Data Validation” Sample Application

Anatomy of an AJAX Interaction (Data Validation Example)



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 async. request
5. The ValidateServlet returns an XML document containing the result
6. The XMLHttpRequest object calls the callback() function and processes the result
7. The HTML DOM is updated

1. A Client event occurs

- A JavaScript function is called as the result of an event
- Example: validateUserId() JavaScript function is mapped as a event handler to a onkeyup event on input form field whose id is set to “userid”

```
<input type="text"
            size="20"
              id="userid"
            name="id"
         onkeyup="validateUserId();">
```

2. An XMLHttpRequest object is created

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

3. An XMLHttpRequest object is configured with a callback function

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest; // callback function
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

4. XMLHttpRequest object makes an async. request

```
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

- URL is set to validate?id=greg

5. The ValidateServlet returns an XML document containing the results (Server)

```
public  void doGet(HttpServletRequest request, HttpServletResponse  response)
     throws IOException, ServletException {

  String targetId = request.getParameter("id");

  if ((targetId != null) && !accounts.containsKey(targetId.trim())) {
         response.setContentType("text/xml");
         response.setHeader("Cache-Control", "no-cache");
         response.getWriter().write("<valid>true</valid>");
     } else {
         response.setContentType("text/xml");
         response.setHeader("Cache-Control", "no-cache");
         response.getWriter().write("<valid>false</valid>");
     }
  }
```

6. XMLHttpRequest object calls callback() function and processes the result

- The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the readyState of the XMLHttpRequest object

```
function processRequest() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            var message = ...;


...
```

7. The HTML DOM is updated

- JavaScript technology gets a reference to any element in a page using DOM API
- The recommended way to gain a reference to an element is to call
  - document.getElementById("userIdMessage"), where "userIdMessage" is the ID attribute of an element appearing in the HTML document
- JavaScript technology may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify child elements

```
<script type="text/javascript">
function setMessageUsingDOM(message) {
    var userMessageElement = document.getElementById("userIdMessage");
    var messageText;
    if (message == "false") {
        userMessageElement.style.color = "red";
        messageText = "Invalid User Id";
    } else {
        userMessageElement.style.color = "green";
        messageText = "Valid User Id";
    }
    var messageBody = document.createTextNode(messageText);
    // if the messageBody element has been created simple replace it otherwise
    // append the new element
    if (userMessageElement.childNodes[0]) {
        userMessageElement.replaceChild(messageBody,
                                         userMessageElement.childNodes[0]);
    } else {
        userMessageElement.appendChild(messageBody);
    }
}
</script>
<body>
   <div id="userIdMessage"></div>
</body>
```

7. AJAX: XMLHttpRequest Methods & Properties

XMLHttpRequest Methods

- open("HTTP method", "URL", syn/asyn)
  - Assigns HTTP method, destination URL, mode
- send(content)
  - Sends request including string or DOM object data
- abort()
  - Terminates current request
- getAllResponseHeaders()
  - Returns headers (labels + values) as a string
- getResponseHeader("header")
  - Returns value of a given header
- setRequestHeader("label","value")
  - Sets Request Headers before sending

XMLHttpRequest Properties

- onreadystatechange
  - Set with an JavaScript event handler that fires at each state change
- readyState – current status of request
  - 0 = uninitialized
  - 1 = loading
  - 2 = loaded
  - 3 = interactive (some data has been returned)
  - 4 = complete
- status
  - HTTP Status returned from server: 200 = OK

XMLHttpRequest Properties

- responseText
  - String version of data returned from the server
- responseXML
  - XML document of data returned from the server
- statusText
  - Status text returned from server

8. AJAX: DOM API & InnerHTML

Browser and DOM

- Browsers maintain an object representation of the documents being displayed
  - In the form of Document Object Model (DOM)
  - It is readily available as document JavaScript object
- APIs are available that allow JavaScript code to modify the DOM programmatically

DOM APIs vs. innerHTML

- DOM APIs provide a means for JavaScript code to navigate/modify the content in a page

```
function setMessageUsingDOM(message) {
        var userMessageElement = document.getElementById("userIdMessage");
        var messageText;
        if (message == "false") {
           userMessageElement.style.color = "red";
           messageText = "Invalid User Id";
        } else {
           userMessageElement.style.color = "green";
           messageText = "Valid User Id";
        }
        var messageBody = document.createTextNode(messageText);
        if (userMessageElement.childNodes[0]) {
           userMessageElement.replaceChild(messageBody,
               userMessageElement.childNodes[0]);
        } else {
           userMessageElement.appendChild(messageBody);
        }
}
```

DOM APIs vs. innerHTML

- Using innerHTML is easier: Sets or retrieves the HTML between the start and end tags of the object

```
function setMessageUsingDOM(message) {
            var userMessageElement = document.getElementById("userIdMessage");
            var messageText;
            if (message == "false") {
                userMessageElement.style.color = "red";
                messageText = "Invalid User Id";
            } else {
                userMessageElement.style.color = "green";
                messageText = "Valid User Id";
            }
           userMessageElement.innerHTML = messageText;
}
```

Do I Have To Use XmlHttpRequest to Write Ajax application?

Ajax Frameworks and Toolkits

- In general, you are going to use Ajax frameworks and toolkits
- These toolkits provide higher-level API, which hides the complexity of XmlHttpRequest

9. AJAX Security

AJAX Security: Server Side

- AJAX-based Web applications use the same server-side security schemes of regular Web applications
  - > You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programatic)
- AJAX-based Web applications are subject to the same security threats as regular Web applications
  - > Cross-site scripting
  - > Injection flaw

AJAX Security: Client Side

- JavaScript code is visible to a user/hacker
  - > Hacker can use the JavaScript code for inferring server side weaknesses
  - > Obfustication or compression can be used
- JavaScript code is downloaded from the server and executed (“eval”) at the client
  - > Can compromise the client by mal-intended code
- Downloaded JavaScript code is constrained by sand-box security model
  - > Can be relaxed for signed JavaScript

Recommendations from OWASP

- Use .innerText instead of .innerHtml
- Don't use eval
- Encode data before its use
- Avoid serialization
- Avoid building XML dynamically

10. JavaScript Development Tools (Try these tools with “AJAX Basics & Dev. Tools” Hands-on Lab)

Development Tools for NetBeans IDE

- Building AJAX Applications over NetBeans is not that much different from building regular Web applications
- NetBeans supports JavaScript editor and debugger

Development Tools on Mozilla Browser

- Mozilla FireBug debugger (add-on)
  - This is the most comprehensive and most useful JavaScript debugger
  - This tool does things all other tools do and more
- Mozilla JavaScript console
- Mozilla DOM inspector (comes with Firefox package)
- Mozilla Venkman JavaScript debugger (add-on)
- Mozilla LiveHTTPHeaders HTTP monitor (similar to NetBeans HTTP monitor)

Mozilla FireBug Debugger

- Spy on XMLHttpRequest traffic
- JavaScript debugger for stepping through code one line at a time
- Inspect HTML source, computed style, events, layout and the DOM
- Status bar icon shows you when there is an error in a web page
- A console that shows errors from JavaScript and CSS
- Log messages from JavaScript in your web page to the console (bye bye "alert debugging")
- An JavaScript command line (no more "javascript:" in the URL bar)

11. AJAX:
Current Issues & Futures

Current Issues of AJAX

- Complexity is increased
  - Server side developers will need to understand that presentation logic will be required in the HTML client pages as well as in the server-side logic
  - Page developers must have JavaScript technology skills
- AJAX-based applications can be difficult to debug, test, and maintain
  - JavaScript is hard to test - automatic testing is hard
  - Weak modularity in JavaScript - namespace collision possible
  - Lack of design patterns or best practice guidelines yet
- Toolkits/Frameworks still maturing

Current Issues of AJAX

- No standardization of the XMLHttpRequest yet
  - Future version of IE will address this
- No support of XMLHttpRequest in old browsers
  - Iframe will help
- JavaScript technology dependency & incompatibility
  - Must be enabled for applications to function
  - Still some browser incompatibilities
- JavaScript code is visible to a hacker
  - Poorly designed JavaScript code can invite security problem

Browsers Which Support XMLHttpRequest

- Mozilla Firefox 1.0 and above
- Netscape version 7.1 and above
- Apple Safari 1.2 and above.
- Microsoft Internet Exporer 5 and above
- Konqueror
- Opera 7.6 and above

AJAX Futures

- AJAX-enabled JSF Component libraries
- Standardization of XMLHttpRequest
- Better browser support
- Better and Standardized Framework support
- More best practice guidelines in the programming model

AJAX Basics

Sang Shin
Java Technology Architect
Sun Microsystems, Inc.
sang.shin@sun.com
www.javapassion.com