JavaScript ES6

Lesson 11-JavaScript with JSON

Lesson Objectives

JSON Object

JSON.stringify and JSON.parse

Ajax

XmlHttpRequest

© 2017 Capgemini. All rights reserved.

JSON Introduction

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.

It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

A JSON object can be stored in its own file, which is basically just a text file with an extension of .json, and a MIME type of application/json.

JSON is purely a data format — it contains only properties, no methods.

JSON requires double quotes to be used around strings and property names. Single quotes are not valid.

JSON Introduction

Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work.

We can validate JSON using an application like JSONLint.

JSON can actually take the form of any data type that is valid for inclusion inside JSON, not just arrays or objects. So for example, a single string or number would be a valid JSON object.

Unlike in JavaScript code in which object properties may be unquoted, in JSON, only quoted strings may be used as properties.

JSON Type

Number : integer, real or floating point
String : double-quoted Unicode with backslashes
Boolean : true and false
Array : ordered sequence of comma-separated values enclosed in square brackets
Object : collection of comma-separated “key”:value pairs enclosed in curly braces
null

JSON Object Notation

A JSON object is an unordered set of name/value pairs

- A JSON object begins with { (left brace) and ends with } (right brace)
- Each name is followed by : (colon) and the name/value pairs are separated by , (comma) and enclosed with in quotes.

The JSON.parse function deserializes JSON text to produce a JavaScript value.

```
var data = {"Name":"Abcd", "age":55}

var dataparsed = eval(data);

console.log(dataparsed.Name);
console.log(dataparsed.age);
```

JSON Object Notation

The JSON.stringify function serializes a JavaScript value to JSON text.

```
function Employee(name, age, salary) {
    this.Name = name;
    this.age = age;
    this.salary = salary;
}

var employeeObject = new Employee('Abcd',25,5118);

console.log(employeeObject);
```

AJAX

- "Asynchronous JavaScript And XML"
- AJAX is not a programming language, but a technique for making the user interfaces of web applications more responsive and interactive
- It provide a simple and standard means for a web page to communicate with the server without a complete page refresh.

Why AJAX?

Intuitive and natural user interaction

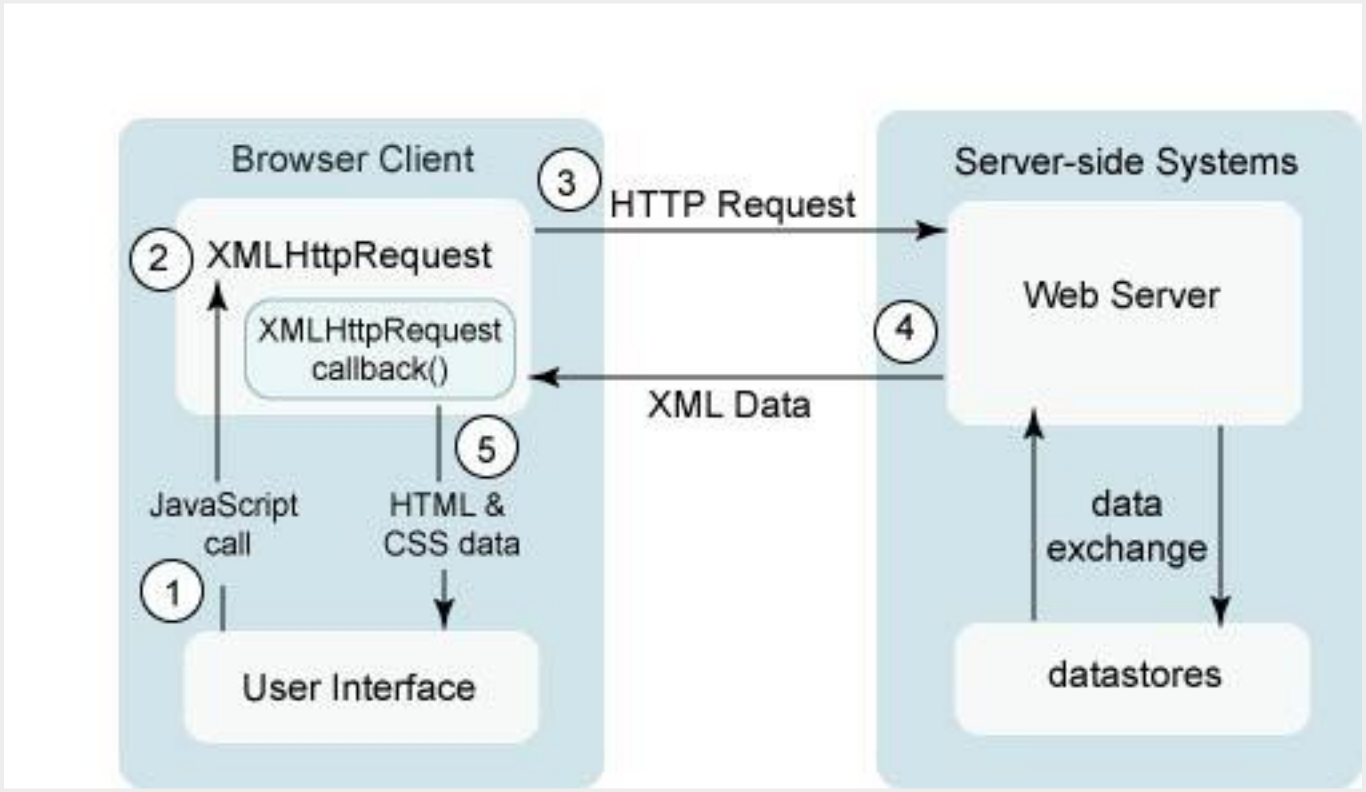
- No clicking required. Call can be triggered on any event
- Mouse movement is a sufficient event trigger

"Partial screen update" replaces the "click, wait, and refresh" user interaction model

- Only user interface elements that contain new information are updated (fast response)

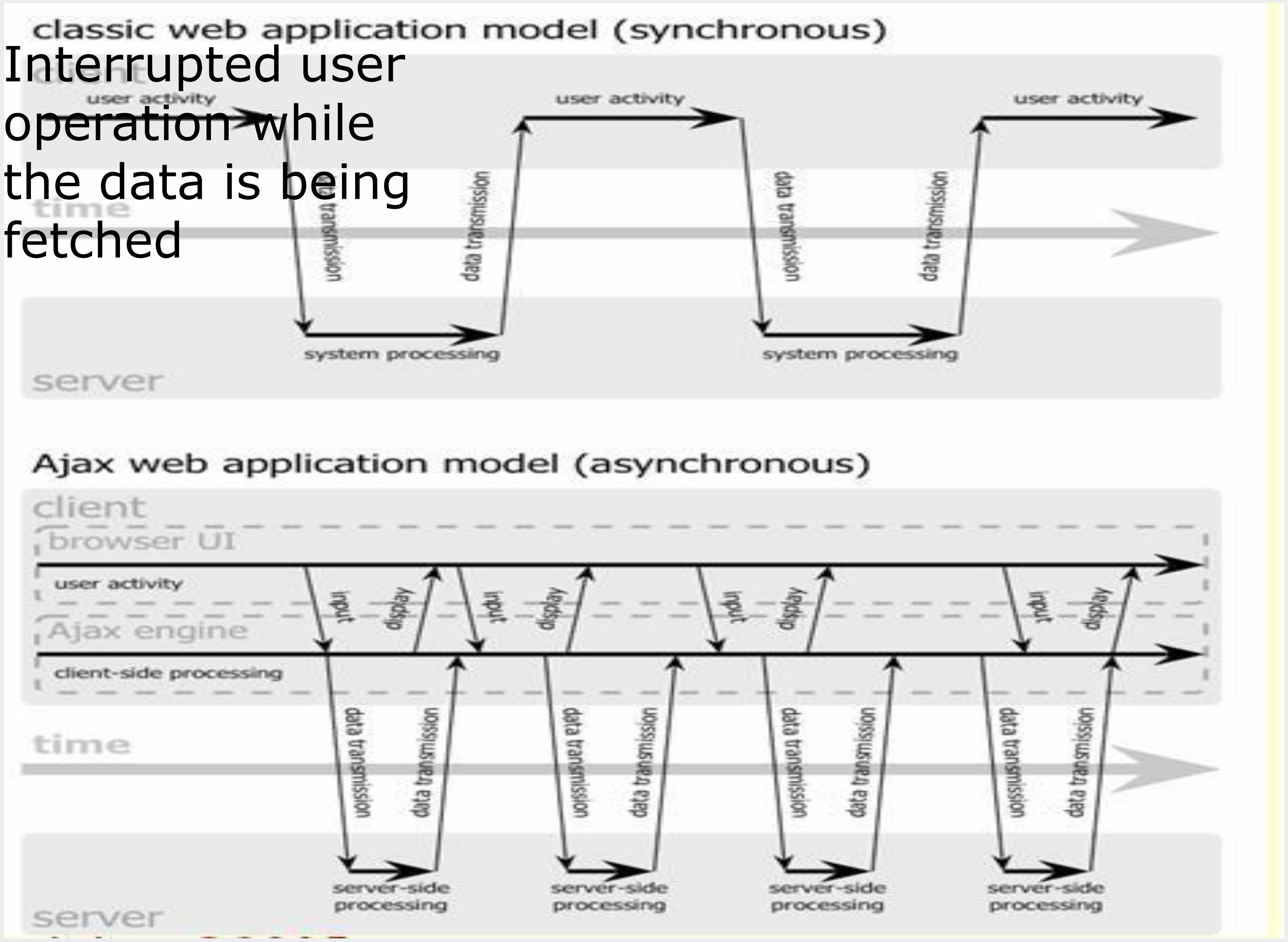
The rest of the user interface remains displayed as it is without interruption (no loss of operational context)

Introduction to How Does Ajax works



Introduction to How Does Ajax works

Interrupted user operation while the data is being fetched



Uninterrupted user operation while data is being fetched

Introduction to How Does Ajax works

JavaScript.

- Loosely typed scripting language
- Allows programmatic interaction with the browser's capabilities
- JavaScript function is called when an event in a page occurs

DOM:

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents

Introduction to How Does Ajax works

CSS

- Allows for a clear separation of the presentation from the content and may be changed programmatically by JavaScript

HTTP

- XMLHttpRequest

XML

- which represents the data passed between the server and client

Introduction to Ajax

JavaScript object

- Created within a JavaScript function

XMLHttpRequest object for asynchronously exchanging the XML data between the client and the server

Communicates with a server via standard HTTP GET/POST

XMLHttpRequest object works in the background

- Does not interrupt user operation

XML with Ajax works

open("method", "URL", syn/asyn)
- Assigns destination URL, method, mode

send(content)
- Sends request including string or DOM object data

abort()
- Terminates current request

XML with Ajax works

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

onreadystatechange
- Event handler that fires at each state change
- You implement your own function that handles this

readyState values – 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

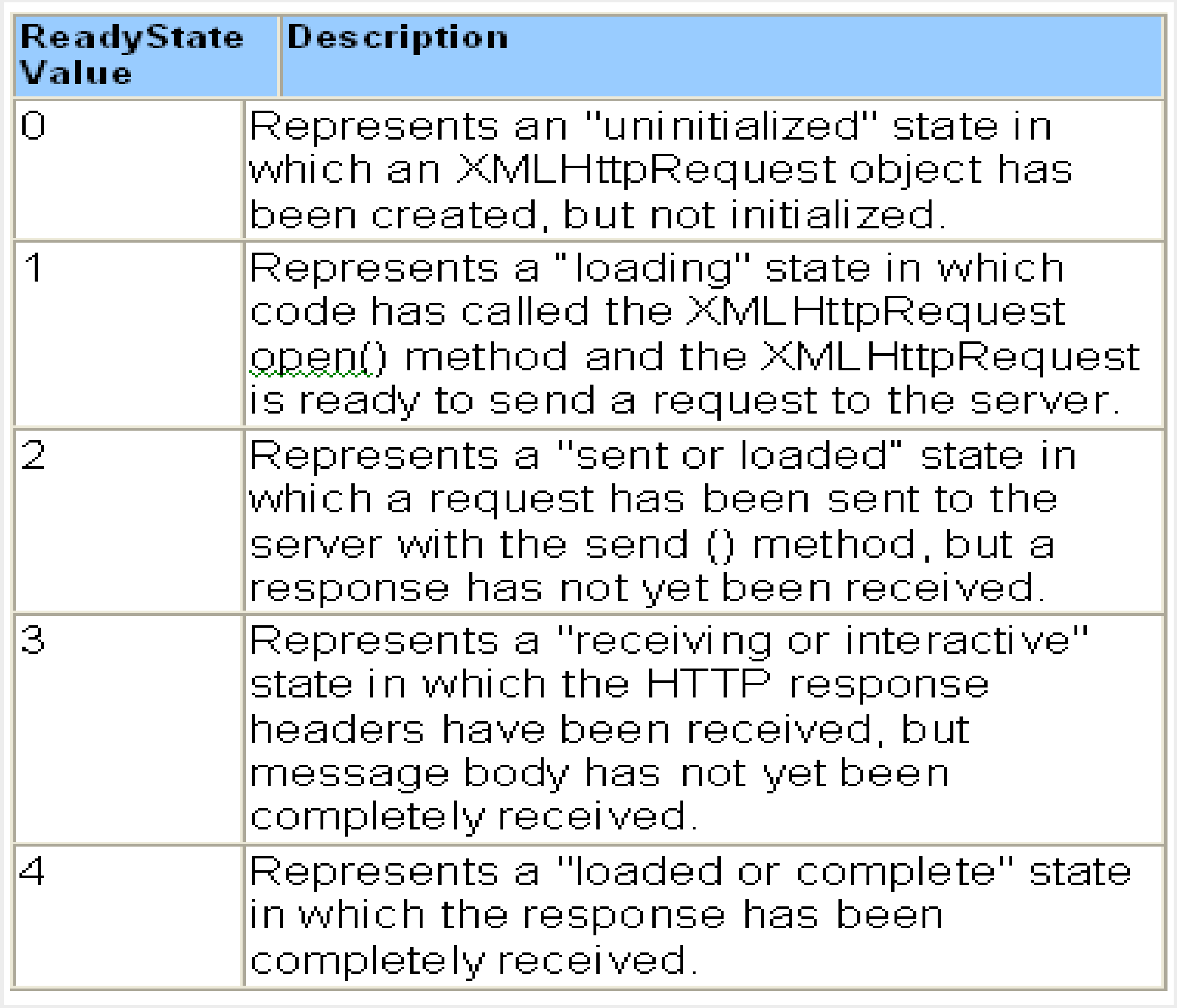
XmlHTTPRequest

responseText
- String version of data returned from server

responseXML
- XML DOM document of data returned

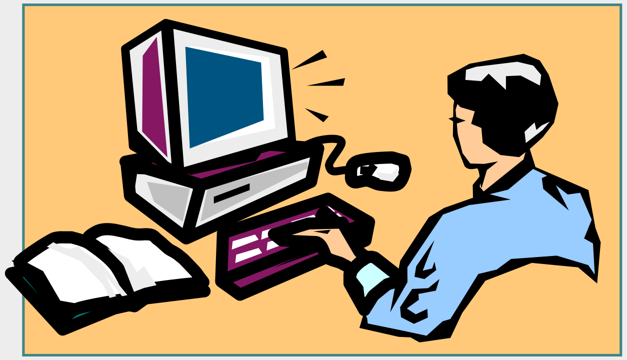
statusText
- Status text returned from server

Introduction to Ajax
States of XMLHttpRequest

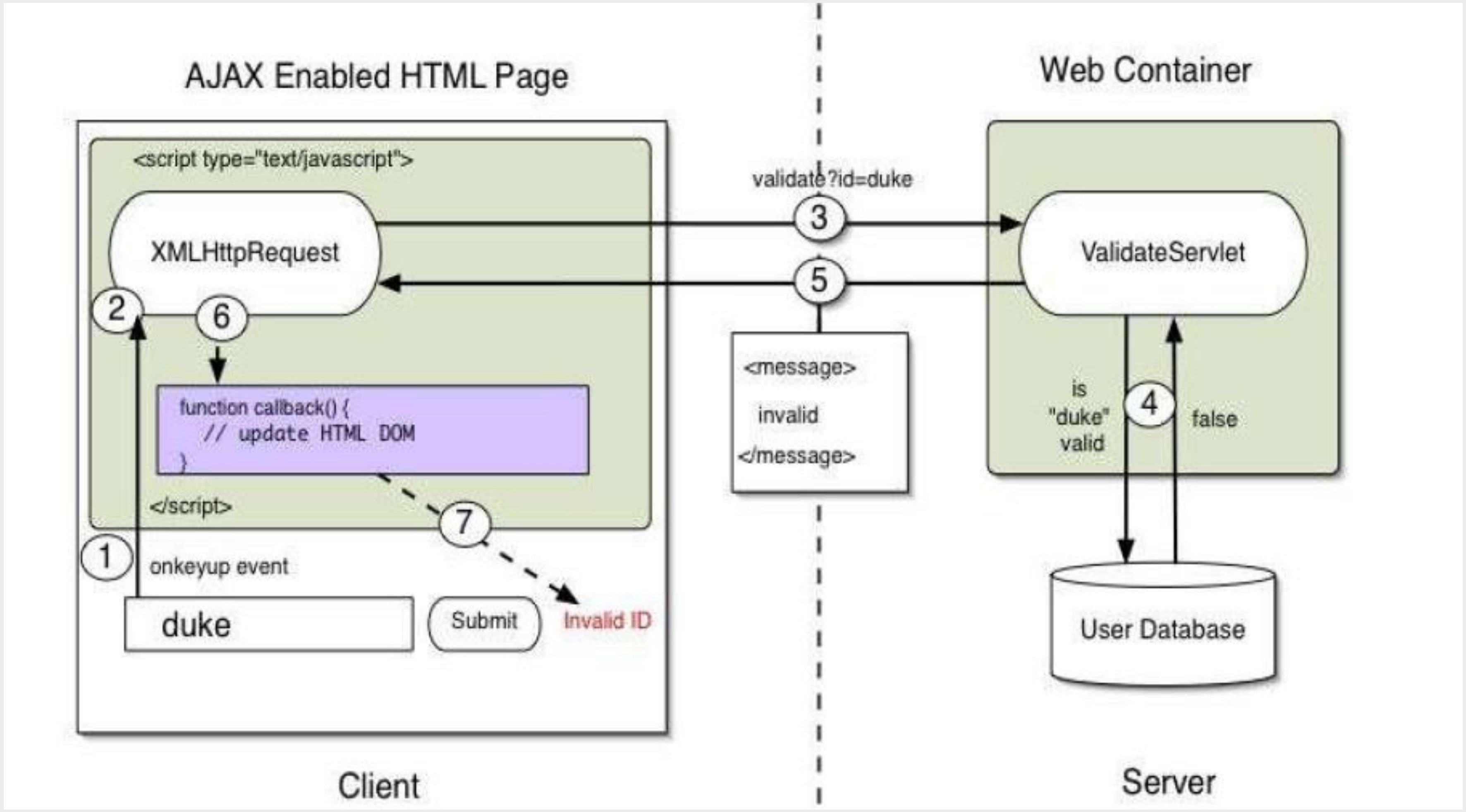


ReadyState Value	Description
0	Represents an "uninitialized" state in which an XMLHttpRequest object has been created, but not initialized.
1	Represents a "loading" state in which code has called the XMLHttpRequest open() method and the XMLHttpRequest is ready to send a request to the server.
2	Represents a "sent or loaded" state in which a request has been sent to the server with the send () method, but a response has not yet been received.
3	Represents a "receiving or interactive" state in which the HTTP response headers have been received, but message body has not yet been completely received.
4	Represents a "loaded or complete" state in which the response has been completely received.

Demo



Ajax Interaction using XMLHttpRequest



XMLHttpRequest Object

Steps Of Interaction

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

XMLHttpRequest Object

A Client event occurs

A JavaScript function is called as the result of an event

Example: validateUserId() JavaScript function is mapped to a onkeyup event on a link or form component

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

XMLHttpRequest Object

An XMLHttpRequest object is created and Configured

```
var req;

function initRequest(url) {

if (window.XMLHttpRequest) {

req = new XMLHttpRequest();

} else if (window.ActiveXObject) {

isIE = true;

req = new ActiveXObject("Microsoft.XMLHTTP");

}

} function validateUserId() {

if (!target) target = document.getElementById("userid");

var url = "validate?id=" + escape(target.value);

initRequest(url);

req.onreadystatechange = processRequest;

req.open("GET", url, true);

req.send(null);

}
```

An XMLHttpRequest object is configured with Callback function

```
var req;

function initRequest(url) {

if (window.XMLHttpRequest) {

req = new XMLHttpRequest();

} else if (window.ActiveXObject) {

isIE = true;

req = new ActiveXObject("Microsoft.XMLHTTP");

}

} function validateUserId() {

if (!target) target = document.getElementById("userid");

var url = "validate?id=" + escape(target.value);

initRequest(url);

req.onreadystatechange = processRequest;

req.open("GET", url, true);

req.send(null);

}
```

XMLHttpRequest Object

XMLHttpRequest object makes an async. request

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

URL is set to validate?id=greg

The request is processed by the Validate Servlet at the Server

```
public class ValidationServlet extends HttpServlet {
private ServletContext context;
private HashMap accounts = new  HashMap();
public void init(ServletConfig config) throws ServletException {
this.context = config.getServletContext();
accounts.put("greg","account data");
accounts.put("duke","account data");
}
```

The request is processed by the Validate Servlet at the Server (contd..)

```
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>");

} }
```

The ValidateServlet returns an XML document containing the results

```
public void doGet(HttpServletRequest request, HttpServletResponse

response)

throws IOException, ServletException {

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

if ((targetId != null) && !users.containsKey(targetId.trim())) {

response.setContentType("text/xml");

response.setHeader("Cache-Control", "no-cache");

response.getWriter().write("valid");

} else {

response.setContentType("text/xml");

response.setHeader("Cache-Control", "no-cache");

response.getWriter().write("invalid");

} } }
```

XML Http Request object calls callback() function and processes the result

The XMLHttpRequest object was configured to call the processRequest() function when there are changes to the readyState of the XMLHttpRequest object

```
-function processRequest() {

if (req.readyState == 4) {

if (req.status == 200) {

var message =

req.responseXML.getElementsByTagName("valid")[0].childNo

des[0].nodeValue;

setMessageUsingDOM(message);
...
```

XMLHttpRequest Object

The HTML DOM is updated

JavaScript technology can gain a reference to any element in the HTML DOM using a number of APIs

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

XMLHttpRequest Object
The HTML DOM is updated (contd..)

```
<script type="text/javascript">

function setMessage(message) {

mdiv = document.getElementById("userIdMessage");

if (message == "invalid") {

mdiv.innerHTML = "<div style=\"color:red\">Invalid User Id</

div>";

} else {

mdiv.innerHTML = "<div style=\"color:green\">Valid User Id</

div>";

}

}

</script>

<body>

<div id="userIdMessage"></div>

</body>
```

XMLHttpRequest Object

Change the Body content of an Element

```
<script type="text/javascript">

function setMessage(message) {

mdiv = document.getElementById("userIdMessage");

if (message == "invalid") {

mdiv.innerHTML = "<div style=\"color:red\">Invalid User Id</

div>";

} else {

mdiv.innerHTML = "<div style=\"color:green\">Valid User Id</

div>";

}

}

</script>

<body>

<div id="userIdMessage"></div>

</body>
```

Ajax using JSON

```
{ "weatherdetails":
        {
                "weatherreport":
                        [
                            {"city": "Mumbai", "weather": "32" },
                            {"city": "Delhi", "weather": "28" },
                            {"city": "Chennai", "weather": "34" },
                            {"city": "Kolkata", "weather": "26" }
                        ]
        }
}
```

JSON data

```
xmlHttpRequest.open("GET","FetchWeather.jsp?city="+city+"&type="+type, true);

           xmlHttpRequest.onreadystatechange = StateChangeForJSON;

                xmlHttpRequest.send(null);
```

Configuring ajax Request and initializing

```
function StateChangeForJSON()
{

if(xmlHttpRequest.readyState == 4 && xmlHttpRequest.status == 200)
       {
              var json = eval('('+ xmlHttpRequest.responseText +')');
              for(var i=0; i < json.weatherdetails.weatherreport.length; i++)
              {.
               .  //code for populating the HTML elements form the JSON data recived from server
               .
```

Receiving JSON data using XMLHttpRequest object

Demo

Demo1
Demo2
DemoOnAjax

Lab

Lab 3

Summary

In this lesson we have learned about –

JSON Object
JSON.stringify and JSON.parse
Ajax