AJAX for Java Web Applications

Session: 1

Introduction to AJAX

Objectives

- Explain Asynchronous JavaScript and XML
- Describe the Document Object Model (DOM)
- Explain XMLHTTPRequest object methods and its properties

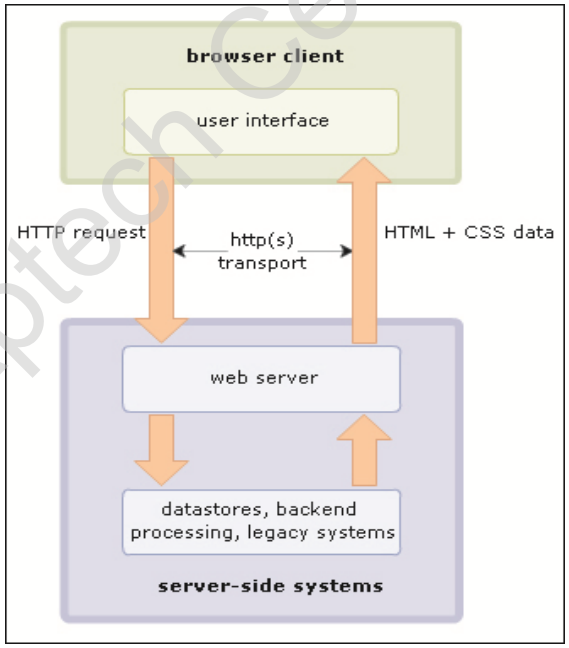
Introduction

- Asynchronous JavaScript and XML (AJAX) is the art of updating parts of a Web page without reloading the whole Web page and exchanging data with a server.

- AJAX allows Web pages to be updated asynchronously by exchanging small amounts of data behind the scenes with the server (asynchronously).

Conventional Web Applications

- Conventional Web applications work in a simple manner.
  - Client-side - User clicks a Submit button, a link, and so on.
  - An HTTP request is triggered by the browser and transmitted to the server.
  - Server-side - Components process the HTTP request and send the results back to the client browser.
  - Client-side - Receives the results from the server and displays in the browser the results after refreshing the Web page.
- This is termed as 'click, wait, and refresh', as the user clicks, waits, and then views the results after the browser refreshes the Web page.
- Following figure shows the conventional Web application model:

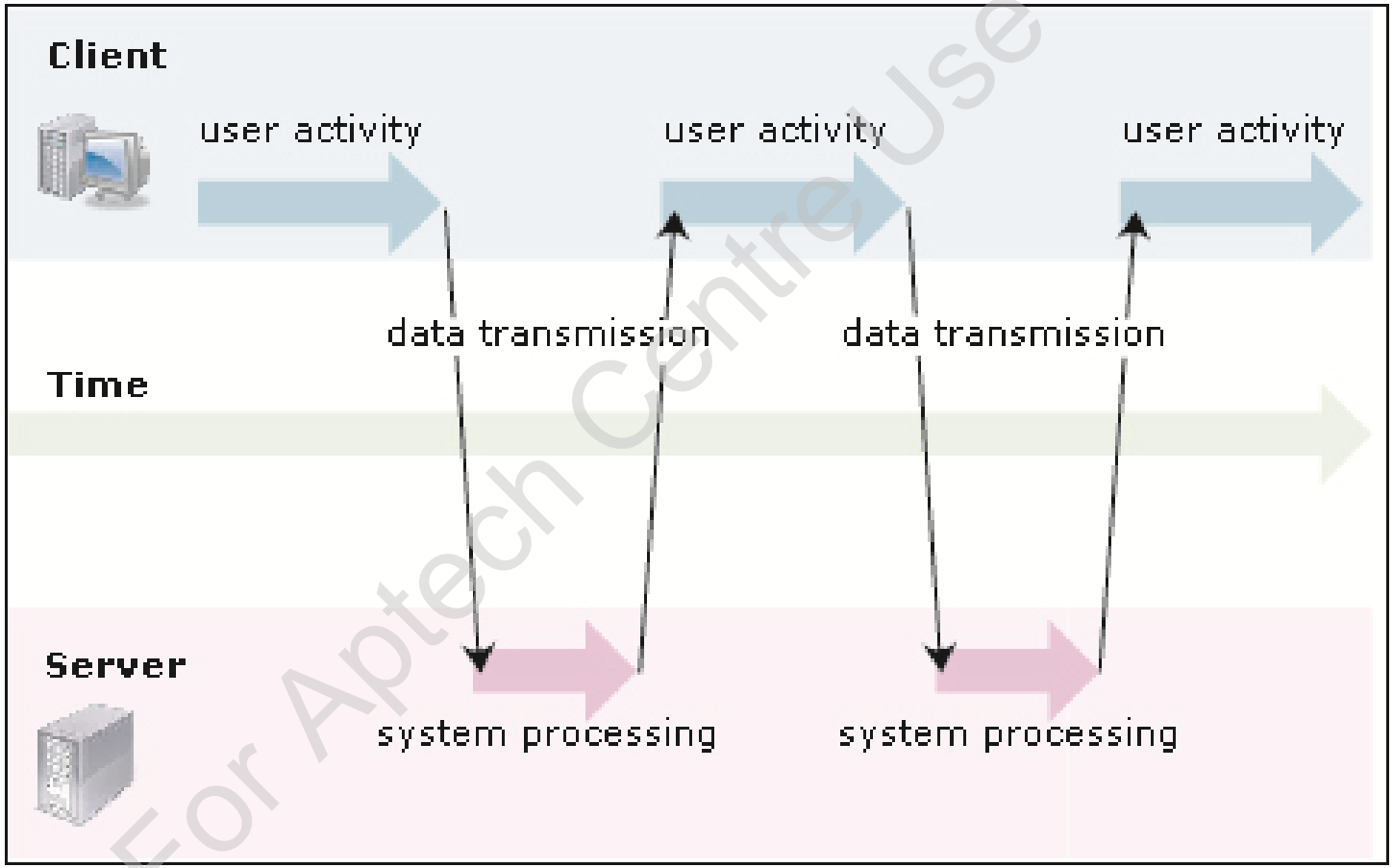


Drawbacks of Conventional Web Applications 1-2

- Conventional Web applications communicate with the server synchronously.
- Once an HTTP request is generated, the user can no longer interact with the application.
- The user is forced to stay idle while the browser refreshes the Web page to display the results retrieved from the server.
- The page refresh also means that the existing browser contents vanish, making the application completely unreadable.
- Conventional Web applications also lack rich user interface.

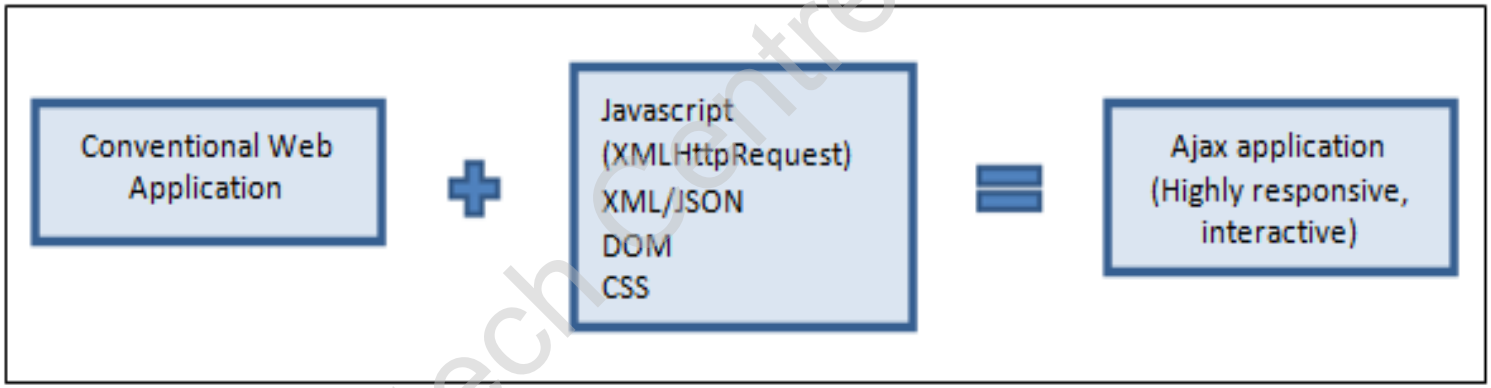
Drawbacks of Conventional Web Applications 2-2

- Following figure shows the synchronous model for conventional Web application:



Using AJAX

- AJAX is a group of technologies. HTML and CSS can be used to markup and style information and the DOM is accessed with JavaScript. For exchanging data asynchronously, JavaScript and the XMLHttpRequest object is used which avoids reloading of full page.
- Following figure shows the technologies used in AJAX Web application:



- With AJAX, the page can be designed as follows:
  - Create the server code which returns only the content for the requests it receives.
  - The initial requests loads everything by default, but when the links are clicked, only the requested contents will be loaded through AJAX, whereas the other contents will remain static on the Web page.

AJAX Technologies

JavaScript

- Facilitates the creation of XMLHttpRequest objects.
- XMLHttpRequest objects facilitate asynchronous communication between the client and the server.
- Asynchronous communication permits the user to continue interacting with the Web page on the client-side, while the XMLHttpRequest object retrieves data from the server.
- Thus, the user does not experience a page refresh.

XML/JSON

- The server-side components process the client request and send a corresponding response back to the client.
- The response contains the data requested by the client in XML/JSON format.
- With wide range of support and other advantages, JSON is mostly used in client-server communication, instead of XML.

DOM

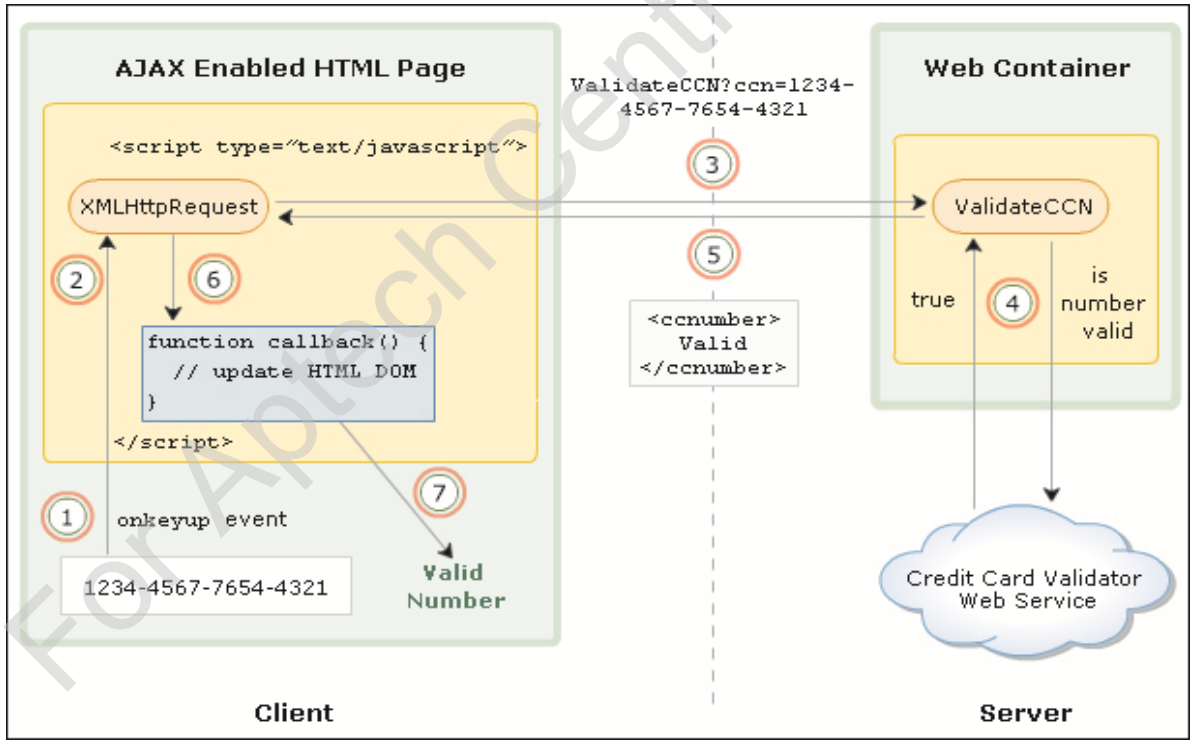
- DOM allows parsing the XML response received from the server and extract the data from it.
- It allows accessing the Web page's DOM tree to add or update existing nodes with new data received from the server.

CSS

- CSS enables adding desktop application like look-and-feel to Web applications.

Working of AJAX

- Consider the Web page of an AJAX-enabled Web application.
- A user needs to provide a credit card number.
- The application validates the credit card number asynchronously using the Validate CCN servlet and a Web service on the server.
- Based on the result of validation, a corresponding message is displayed next to the text box.
- Following figure shows the various steps involved in processing the AJAX request and getting the response from the server:



Document Object Model 1-2

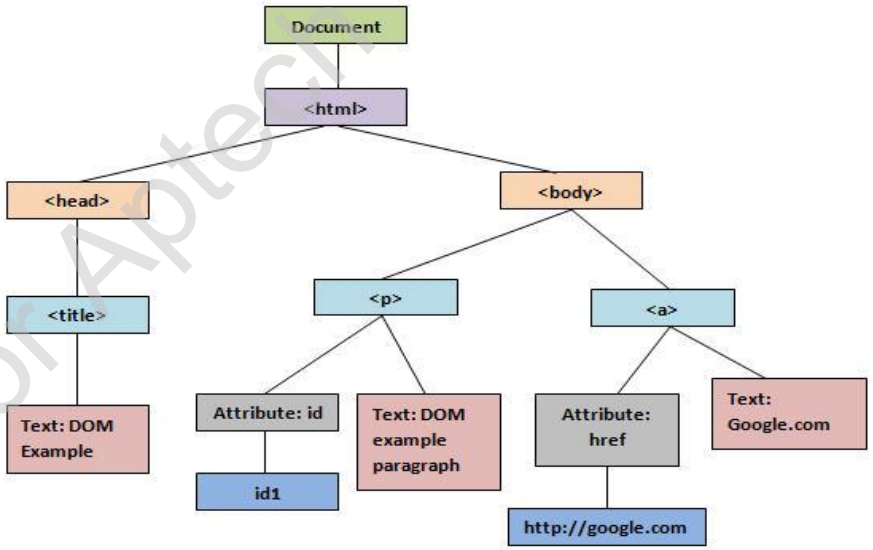
- The Document Object Model is a platform and language-independent standard for representing HTML and XML documents.
- DOM is a standard object model.
- DOM allows accessing and manipulating the HTML and XML document content.
- DOM facilitates dynamic modification of Web pages.
- DOM represents HTML or XML document as a collection of objects referred as nodes.
- Based on how these objects are placed in the document, DOM connects each of these nodes creating a tree like structure.
- DOM classifies every node as a specific type of node.
- For example, tags are classified as element nodes, whereas text within the tags is classified as text nodes.

Document Object Model 2-2

- Consider the Example.html file demonstrated in the following Code Snippet:

```
<!DOCTYPE html>
<html>
  <head>
     <title> DOM Example </title>
   </head>
   <body>
      <p id="id1"> DOM example paragraph</p>
      <a href="http://google.com"> Google.com </a>
   </body>
</html>
```

- A tree structure can be created as shown in the following figure:

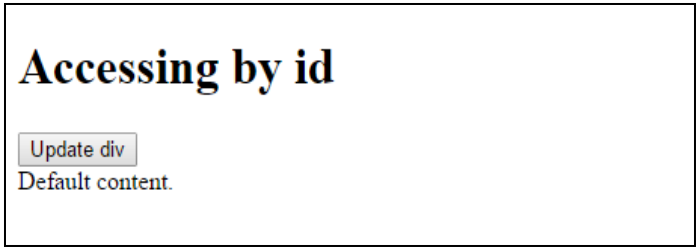


HTML DOM Methods 1-4

- **getElementById(id)** - searches for and returns the element whose id is specified as the input parameter.
  - **Syntax**: getElementById(id)
  - **Example**:

```
...
<script type="text/javascript">
function updateDiv(theDiv,theText){
document.getElementById(theDiv).innerHTML =
theText; }
</script>
</head>
<body>
<h1>Accessing by id</h1>
<input type="button" value="Update div"
onclick="updateDiv('demo','Content updated
by accessing id!!!');">
<div id="demo">Default content.</div>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:



HTML DOM Methods 2-4

- **getElementsByTagName(name)** - searches for and returns an array of all such elements whose name matches to the one specified as the input parameter.
  - **Syntax**: getElementsByTagName(name)
  - **Example:**

```
...
<script type="text/javascript">
function updateByTagName(theTagName){
x = document.getElementsByTagName(theTagName);
for (i=0;i < x.length; i++){
   x[i].innerHTML+=" - Modified content by
accessing the tag name.";
}
}
</script>
</head>
<body>
<h1>Accessing by tag name!!!</h1>
<input type="button" value="Update By Tag Name"
onclick="updateByTagName('p');"/>
<p>        Hello World 1!        </p>
<p>        Hello World 2!        </p>
<p>        Hello World 3!        </p>
<p>        Hello World 4!        </p>
</body>
...
```

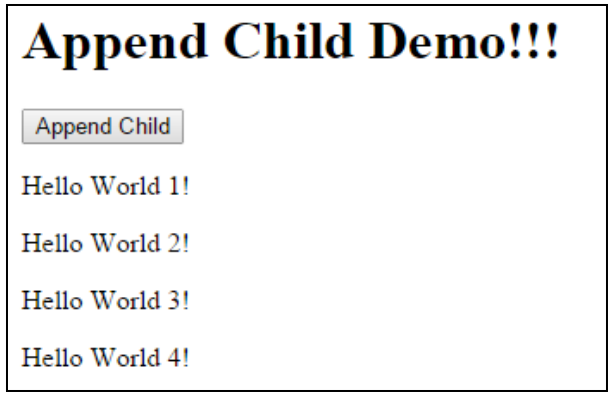
  - Following figure shows the output before and after the button is clicked:

HTML DOM Methods 3-4

- **appendChild(node)** - appends the node specified as input parameter to the tree structure.
  - **Syntax:** appendChild(node)
  - **Example:**

```
...
<script type="text/javascript">
function appendChildTest(theTagName){
var x =
document.getElementsByTagName(theTagName);
var aNewNode = document.createElement("p");
var aTextNode = document.createTextNode("Hello
World Child!!!");
aNewNode.appendChild(aTextNode);
x[0].appendChild(aNewNode);
}
</script>
</head>
<body>
<h1>Append Child Demo!!!</h1>
<input type="button" value="Append Child"
onclick="appendChildTest('p');"/>
   <p> Hello World 1! </p>
   <p> Hello World 2!</p>
   <p> Hello World 3! </p>
   <p> Hello World 4! </p>
</body>>
...
```

  - Following figure shows the output before and after the button is clicked:

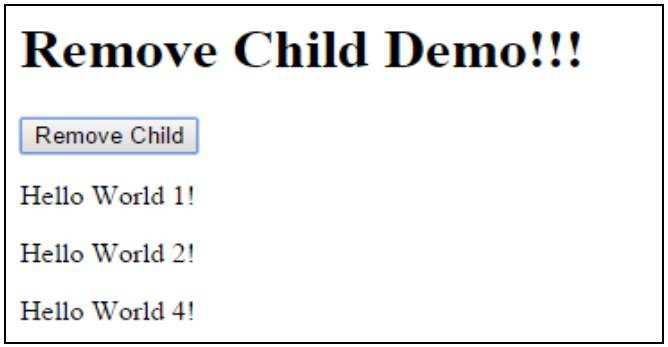


HTML DOM Methods 4-4

- **removeChild(node)** - removeChild() method removes the node specified as input parameter from the tree structure.
  - **Syntax:** removeChild(node)
  - **Example:**

```
...
<script type="text/javascript">
function removeChildTest(theTagName,thenthChild)
{
var x =
document.getElementsByTagName(theTagName);
x[0].parentNode.removeChild(x[thenthChild]);
}
</script>
</head>
<body>
<h1>Remove Child Demo!!!</h1>
<input type="button" value="Remove Child"
onclick="removeChildTest('p',2);"/>
    <p> Hello World 1! </p>
    <p> Hello World 2!</p>
    <p> Hello World 3! </p>
    <p> Hello World 4! </p>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:

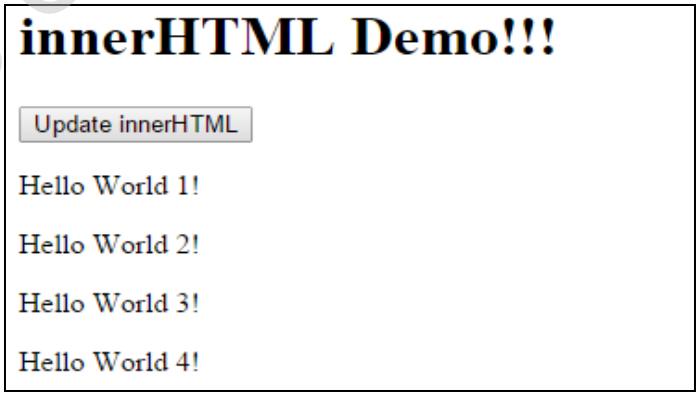


HTML DOM Properties 1-5

- **innerHTML** - provides access to the content of an element.
  - **Example:**

```
...
<script type="text/javascript">
function
updateInnerHTML(theTagName,thenthChild){
var x =
document.getElementsByTagName(theTagName);
x[thenthChild].innerHTML="Updated
content!!!";
}
</script>
</head>
<body>
<h1>innerHTML Demo!!!</h1>
<input type="button" value="Update
innerHTML"
onclick="updateInnerHTML('p',2);"/>
   <p> Hello World 1! </p>
   <p> Hello World 2!</p>
   <p> Hello World 3! </p>
   <p> Hello World 4! </p>
  </body>
...
```

  - Following figure shows the output before and after the button is clicked:

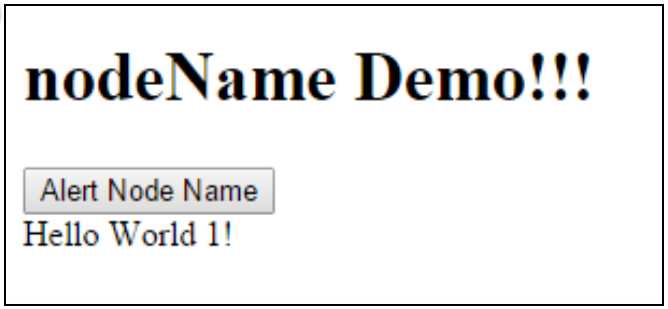


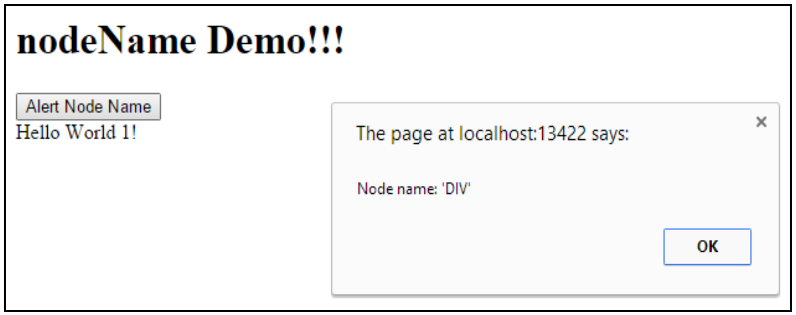
HTML DOM Properties 2-5

- **nodeName** - provides access to the name of the current node.
  - **Example:**

```
...
<script type="text/javascript">
function displayNodeName(tagID){
var x =
document.getElementById(tagID);
alert("Node name: '"+x.nodeName+"'");
}
</script>
</head>
<body>
<h1>nodeName Demo!!!</h1>
<input type="button" value="Alert Node
Name"
onclick="displayNodeName('hello');"/>
<div id=”hello”> Hello World 1! </div>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:



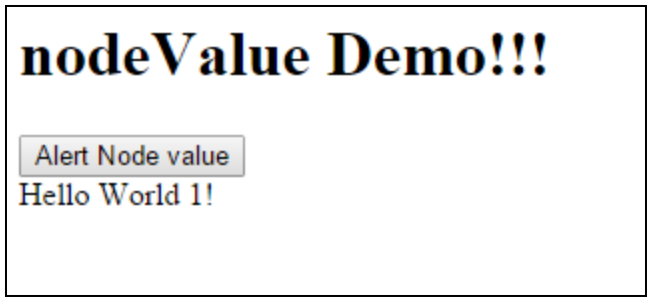


HTML DOM Properties 3-5

- **childNodes** - provides access to all the child nodes of the given node.
- **nodeValue** - used to display a node's content.
  - **Example:**

```
...
<script type="text/javascript">
function displayNodeValue(tagId)   {
var x =
document.getElementById(tagId);
alert("Node Value:
'"+x.childNodes[0].nodeValue+"'");
}
</script>
</head>
<body>
<h1>nodeValue Demo!!!</h1>
<input type="button" value="Alert Node
value"
onclick="displayNodeValue('div1');"/>
<div id="div1"> Hello World 1! </div>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:

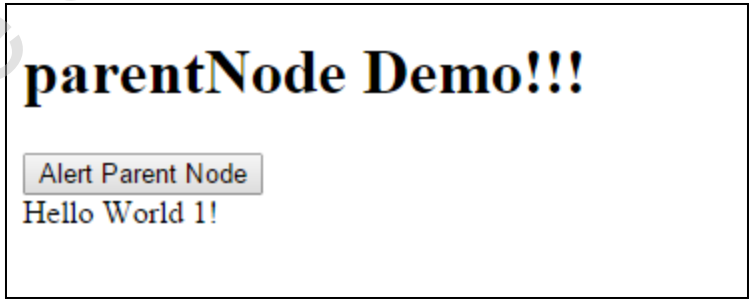


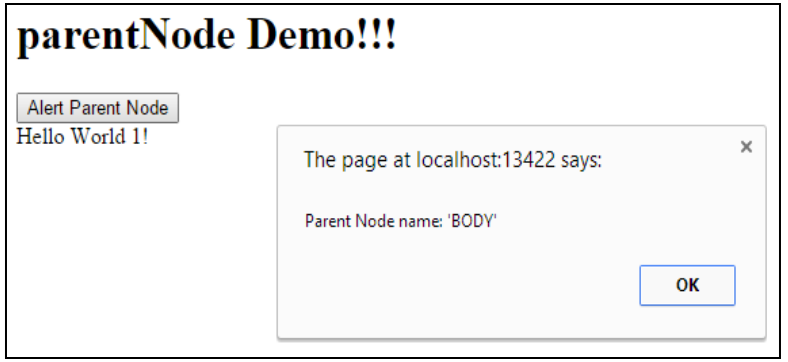
HTML DOM Properties 4-5

- **parentNode** - provides access to the parent node of the current node.
  - **Example:**

```
...
<script type="text/javascript">
function displayParentNode(tagId){
var x =
document.getElementById(tagId);
alert("Parent Node name:
'"+x.parentNode.nodeName+"'");
}
</script>
</head>
<body>
<h1>parentNode Demo!!!</h1>
<input type="button" value="Alert
Parent Node"
onclick="displayParentNode('div1');"/>
<div id="div1"> Hello World 1! </div>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:



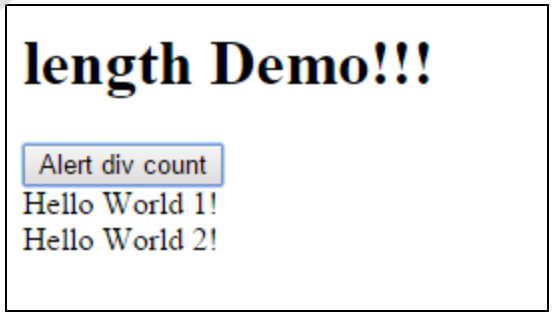


HTML DOM Properties 5-5

- **length** - returns the number of nodes present in a node list.
  - **Example:**

```
...
<script type="text/javascript">
function displayBodyLength(theTagName){
var x =
document.getElementsByTagName(theTagName
);
alert("Number of div nodes:
'"+x.length+"'");
}
</script>
</head>
<body>
<h1>length Demo!!!</h1>
<input type="button" value="Alert div
count"
onclick="displayBodyLength('div');"/>
<div id="div1"> Hello World 1! </div>
<div id="div2"> Hello World 2! </div>
</body>
...
```

  - Following figure shows the output before and after the button is clicked:



XML DOM Methods 1-4

- XML Document Object Model (XML DOM) allows accessing and manipulating XML documents using methods similar to that of HTML DOM.
- Following Code Snippet demonstrates the Student.xml document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<students>
          <record id="1">
                    <name>Ashley
Andrews</name>
                    <gender>male</gender>
                    <age>12 yrs</age>
                    <grade>7th</grade>
                    <division>A</division>
          </record>
          <record id="2">
                    <name>Ashley
Mathias</name>
                    <gender>male</gender>
                    <age>5 yrs</age>
                    <grade>1st</grade>
                    <division>B</division>
          </record>
          <record id="3">
                    <name>Ashley Bill</name>
                    <gender>male</gender>
                    <age>6 yrs</age>
                    <grade>1st</grade>
                    <division>A</division>
          </record>
</students>
```

- Following Code Snippet demonstrates the script that must be added to load the XML document as a JavaScript instance:

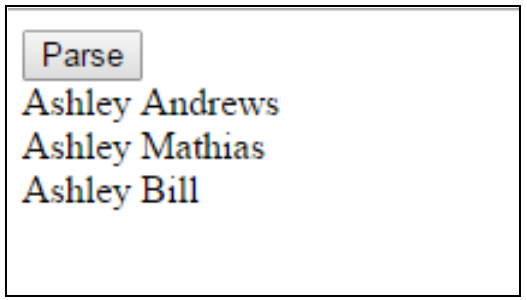
```
<script type="text/javascript">
function loadXMLDoc(theDocName){
if (window.XMLHttpRequest){
  xhttp=new XMLHttpRequest();
}
else{
  xhttp=new
ActiveXObject("Microsoft.XMLHTTP");
}
xhttp.open("GET",theDocName,false);
xhttp.send();
return xhttp.responseXML;
}
</script>
```

XML DOM Methods 2-4

- **getElementsByTagName(name)** - searches for and returns elements whose name matches with the input parameter.
  - **Syntax:** getElementsByTagName(name)
  - **Example:**

```
...
<script type="text/javascript">
...
function
printNames(theXML,theNodeName,theIdToUpdate){
  xmlDoc = loadXMLDoc(theXML);
  x = xmlDoc.getElementsByTagName(theNodeName);
  var OPStr="";
  for (i = 0;i < x.length;i++){
    OPStr+=x[i].childNodes[0].nodeValue+"<br/>";
  }
document.getElementById(theIdToUpdate).innerHTML =
OPStr;
}
</script>  </head>
 <body>
<input type="button"
onclick='printNames("Student.xml","name","divOP");'
value="Parse"/>
div id="divOP"> </div>
 </body>
...
```

  - Following figure shows the output after the button is clicked:

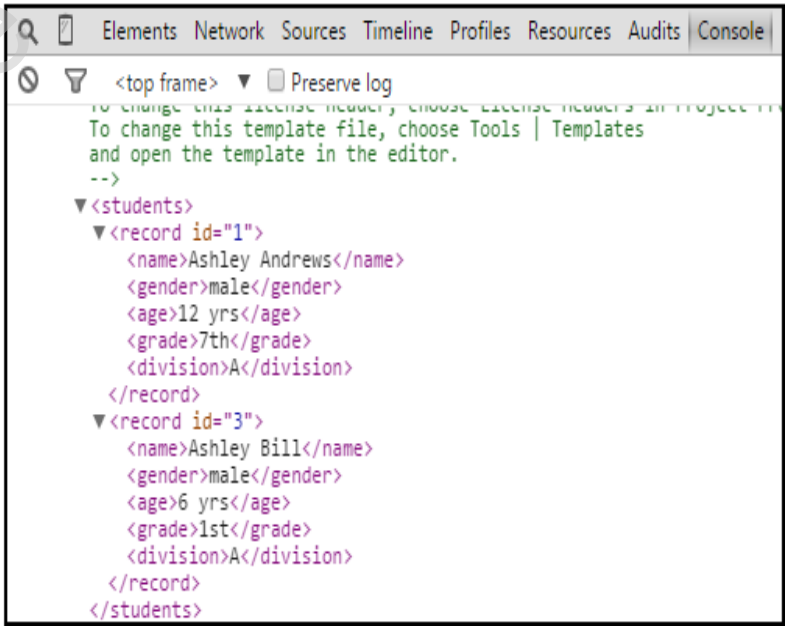


XML DOM Methods 3-4

- **removeChild(node)** - removes the node specified as input parameter from the tree structure.
  - **Syntax:** removeChild(node)
  - **Example:**

```
...
function
removeNode(theXML,theNodeName,theIdToRemove)
{
var xmlDoc = loadXMLDoc(theXML);
var x =
xmlDoc.getElementsByTagName(theNodeName);
var temp = x[theIdToRemove];
x[0].parentNode.removeChild(temp);
console.log(xmlDoc);
}
</script>
</head>
<body>
<input type="button"
onclick='removeNode("Student.xml","record",1
);' value="Remove a node"/>
</body>
...
```

  - Following figure shows the node has been removed:

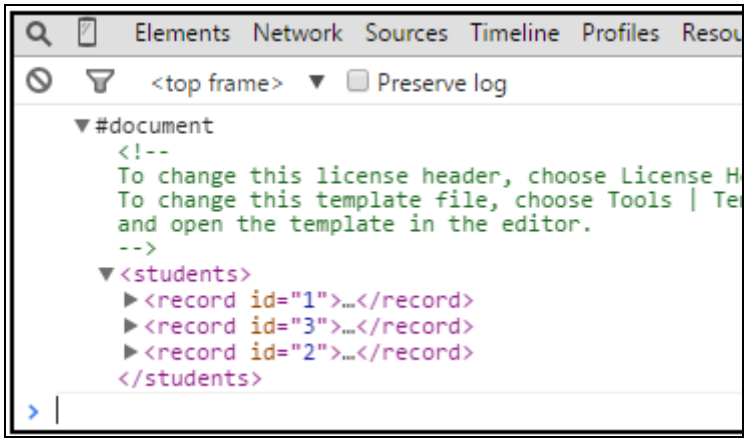


XML DOM Methods 4-4

- **appendChild(node)** - appends the node specified as input parameter to the tree structure.
  - **Syntax:** appendChild(node)
  - **Example:**

```
...
function
removeAndAppendNode(theXML,theNodeName,theId
ToRemove){
var xmlDoc = loadXMLDoc(theXML);
var x =
xmlDoc.getElementsByTagName(theNodeName);
var temp = x[theIdToRemove];
x[0].parentNode.removeChild(temp);
x[0].parentNode.appendChild(temp);
console.log(xmlDoc);
}
</script>
</head>
<body>
<input type="button"
onclick='removeAndAppendNode("Student.xml","
record",1);' value="Re-append a node"/>
</body>...
...
```

  - Following figure shows the node at index 1 (id - 2) is removed from its current location in the XML document and appended at the end:



XML DOM Properties 1-4

nodeName	• Provides access to the name of the current node.
nodeValue	• Used to display a node's value.
parentNode	• Provides access to the parent node of the current node.
childNodes	• Provides access to all the child nodes of the given element.
length	• Returns the number of nodes present in a node list.
attributes	• Returns a list containing the specified node's attributes. If the specified node is not an element, this property returns NULL.

XML DOM Properties 2-4

- Following Code Snippet demonstrates the use of the different DOM properties:

```
...
<script type="text/javascript">
...
function updateTable(theXML,theNodeName){
  var xmlDoc = loadXMLDoc(theXML);
  var x = xmlDoc.getElementsByTagName(theNodeName);

   document.getElementById("nodeNameId").innerHTML=x[0].nodeName;
document.getElementById("childNodesid").innerHTML=x[0].childNodes[1].nodeName;

document.getElementById("nodeValueId").innerHTML=x[0].childNodes[1].childNodes[
0].nodeValue;

document.getElementById("parentNodeId").innerHTML=x[0].parentNode.nodeName;

document.getElementById("lengthid").innerHTML=x.length;

document.getElementById("attributesid").innerHTML=x[0].attributes.getNamedItem(
"id").value;
</script>
</head>
```

XML DOM Properties 3-4

```
<body>
<input type="button" onclick='updateTable("Student.xml","record");' value="Update
table from XML"/>
<table border="1">
<col width="80%">
<col width="20">
<tbody>
<tr>
  <td> DOM Property </td>
   <td> Output </tx>
<tr>
   <tr>
      <td> xmlDocObject.getElementsByTagName(theNodeName)[0].nodeName </td>
      <td> <span id="nodeNameId"> </span> </td>
   </tr>
   <tr>
      <td> xmlDocObject.getElementsByTagName(theNodeName)[0].nodeValue </td>
      <td> <span id="nodeValueId"> </span> </td>
   </tr>
   <tr>
      <td> xmlDocObject.getElementsByTagName(theNodeName)[0].parentNode.nodeName
      </td>
      <td> <span id="parentNodeId"> </span> </td>
   </tr>
```

XML DOM Properties 4-4

```
<tr>
    <td> xmlDocObject.getElementsByTagName(theNodeName)[0].childNodes[1].nodeName
    </td>
    <td> <span id="childNodesid"> </span> </td>
</tr>
<tr>
    <td> xmlDocObject.getElementsByTagName(theNodeName).length </td>
    <td> <span id="lengthid"> </span> </td>
</tr>
<tr>
    <td>
xmlDocObject.getElementsByTagName(theNodeName).attributes.getNamedItem("id").value
   </td>
   <td> <span id="attributesid"> </span> </td>
</tr>
</tbody> </table> </body> </html>
```

- Following figure shows the result generated by the script on clicking the button:

Update table from XML

DOM Property	Output
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeName	record
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeValue	Ashley Andrews
xmlDocObject.getElementsByTagName(theNodeName)[0].parentNode.nodeName	students
xmlDocObject.getElementsByTagName(theNodeName)[0].childNodes[1].nodeName	name
xmlDocObject.getElementsByTagName(theNodeName).length	3
xmlDocObject.getElementsByTagName(theNodeName).attributes.getNamedItem("id").value	1

XMLHttpRequest Object

XMLHttpRequest is an API used by scripting languages for transferring and manipulating XML data to and from a Web server.

This is done by establishing an independent connection channel between client-side and server-side of a Web page using HTTP.

XMLHttpRequest object fetches data in XML, JSON, and plain data format.

XMLHttpRequest Methods

- Following table lists the methods of XMLHttpRequest object:

Method	Description
abort()	Cancels the current request.
getAllResponseHeaders()	Returns the complete set of HTTP headers as a string.
getResponseHeader(headerName)	Returns the value of the specified HTTP header.
open(method, URL, [, async [, user [, password]]])	Specifies the method, URL, and other optional attributes of a request. The method parameter can have a value of 'GET', 'HEAD', or 'POST'. The request is to be handled asynchronously or not is specified by 'async' parameter.
send(content)	Sends the request.
setRequestHeader(label, value)	Adds a label/value pair to the HTTP header to be sent.

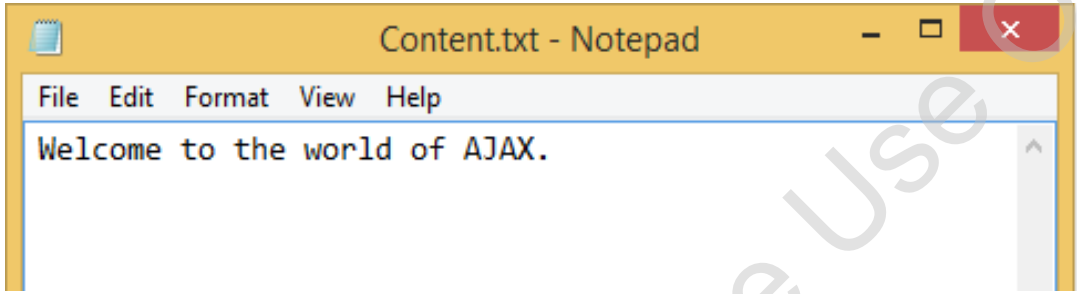
XMLHttpRequest Properties 1-4

- Following table lists the properties of XMLHttpRequest object:

Property	Description
onreadystatechange	It is an event handler which fires at every state change.
readyState	The current state of the XMLHttpRequest object is defined by readyState property. 0 - Request is not initialized 1 - Request has been set up 2 - Request has been sent 3 - Request is in process 4 - Request is completed
responseText	The response returned is a string or Text format.
responseXML	Returns the response as XML. This XML can be parsed.
status	Returns the status as a number. For example, 200 for "OK" and 404 for "Not Found“.
statusText	Returns the status as a string. For example, "OK" or "Not Found".

XMLHttpRequest Properties 2-4

- Following figure shows the contents of the Content.txt file:



- Following Code Snippet demonstrates the use of XMLHttpRequest object and its properties and methods:

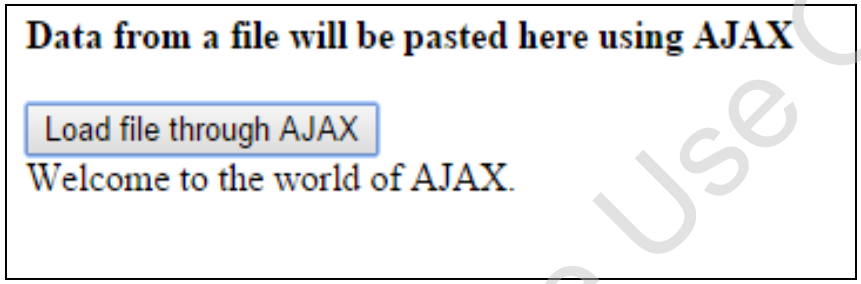
```
...
<script type="text/javascript">
  var ajaxRequest;  // The variable that makes AJAX possible!

 function SendRequest(){
  try {
     ajaxRequest = new XMLHttpRequest();
  } catch (e) {
       try {
         ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
       }catch (e){
           try {
              ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
           } catch (e){
```

XMLHttpRequest Properties 3-4

```
            try {
              ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e){
                alert("Your browser broke!");
                return false;
            }
    }
}
ajaxRequest.onreadystatechange = function (){
          if (ajaxRequest.readyState == 4) {
               var ajaxDisplay = document.getElementById('ajaxDiv');
               ajaxDisplay.innerHTML = ajaxRequest.responseText;
          }
   }
      ajaxRequest.open("GET", "Content.txt", true);
      ajaxRequest.send();
}
</script>
<title> AJAX example </title>
</head>
<body>
<div style="font-weight:bold;"> Data from a file will be pasted here using
AJAX</div>
<br />
<input type="button" value="Load file through AJAX"
onclick="SendRequest();"></input>
<div id="ajaxDiv"> </div>
  </body> </html>
```

XMLHttpRequest Properties 4-4

- Following figure shows the output after clicking the button:



- On event, onload, the request is sent asynchronously and the data is requested from a file named, Content.txt.
- The XMLHttpRequest object xmlDoc is created in the SendRequest() function.
- Next, the open() method is called where the type of method used in the connection, the URL, and whether the request is synchronous or asynchronous is specified.
- Then, the send() method is called to send the request.
- The response is accessed using the responseText property of the XMLHttpRequest object and the response is displayed in the html page.
- The file contents are loaded asynchronously and the div, ajaxDiv is updated to display the text from the file.

AJAX in Java with Servlet and JSP 1-5

- Servlets provide a platform-independent, component-based method for building Web-based applications. Servlets can use Java APIs.
- JavaServer Pages (JSP) are HTML pages with embedded Java code saved with the extension, '.jsp'. JSPs can use Java APIs.
- Consider an example that demonstrates two files, index.jsp and TimeServlet.java.
- The index.jsp page consists of a JavaScript code wherein the request is passed to the TimeServlet servlet asynchronously and the current time of the server is received as the response.

AJAX in Java with Servlet and JSP 2-5

- Following Code Snippet depicts the index.jsp page:

```
...
<script>
function ajaxAsyncRequest(){
  try{
         ajaxRequest = new XMLHttpRequest();
 }catch (e){
     try{
              ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
         }catch (e){
              try{
               ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
            }catch (e){
                alert("Your browser broke!");
                return false;
            }
         }
  }
ajaxRequest.onreadystatechange = function (){
     if (ajaxRequest.readyState == 4) {
          var ajaxDisplay = document.getElementById('message');
          ajaxDisplay.innerHTML = ajaxRequest.responseText;
          } };
   ajaxRequest.open("GET","TimeServlet", true);
   ajaxRequest.send(null);
}
```

AJAX in Java with Servlet and JSP 3-5

```
</script>
</head> <body>
<div class="container">
The server time will be displayed on clicking the button
<input type="button" value="Show Server Time" onclick='ajaxAsyncRequest();'/>
</div>
    <br>
    <div id="message">
    </div>
</body> </html>
```

- Following Code Snippet depicts the servlet class, TimeServlet:

```
...
public class TimeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @Override
    public void doGet (HttpServletRequest request,HttpServletResponse response)
            throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();
        Date currentTime= new Date();
        String message = String.format("Currently time is %tr on %tD.",currentTime,
currentTime);
        out.print(message);
        out.flush();
    }
}
```

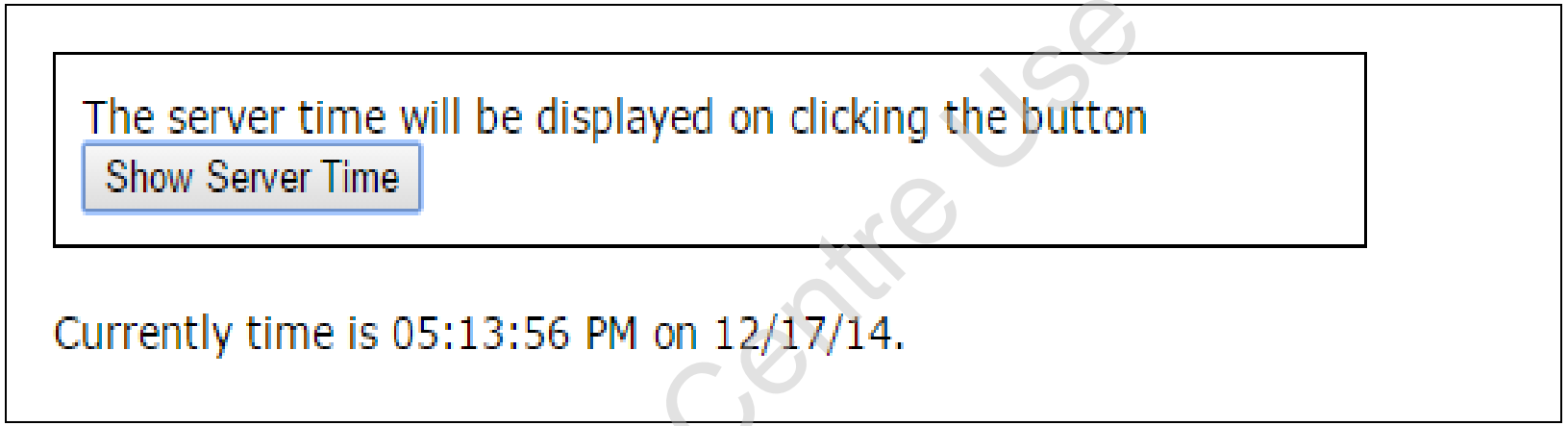
AJAX in Java with Servlet and JSP 4-5

- Following Code Snippet shows the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
          version="3.1">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
    <servlet-name>TimeServlet</servlet-name>
    <servlet-class>TimeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
    <servlet-name>TimeServlet</servlet-name>
    <url-pattern>/TimeServlet</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

AJAX in Java with Servlet and JSP 5-5

- Following figure shows the output for the index.jsp page after clicking the button:



- On clicking the Show Server Time button, an asynchronous request is sent to the server.
- The doGet() method of the servlet is executed which returns current time of the server.

Summary

- Conventional Web applications communicate with the server synchronously. Once an HTTP request is generated, the user can no longer interact with the application.
- Synchronous communication between the client and server introduces long delays in the request-response cycle.
- AJAX comprises JavaScript, XML, DOM, and CSS. These technologies enable development of highly responsive and rich UI-based Web applications.
- The Document Object Model is a platform and language-independent standard for representing HTML and XML documents.
- Using properties and methods that are supported by HTML/XML DOM API, the content of a HTML/XML can be manipulated dynamically at client-side.
- XMLHttpRequest is an API supported by JavaScripting for transferring and manipulating data to and from a Web server by establishing an asynchronous connection channel between client-side and server-side using HTTP.
- XMLHttpRequest object fetches data in XML, JSON, and plain data format.