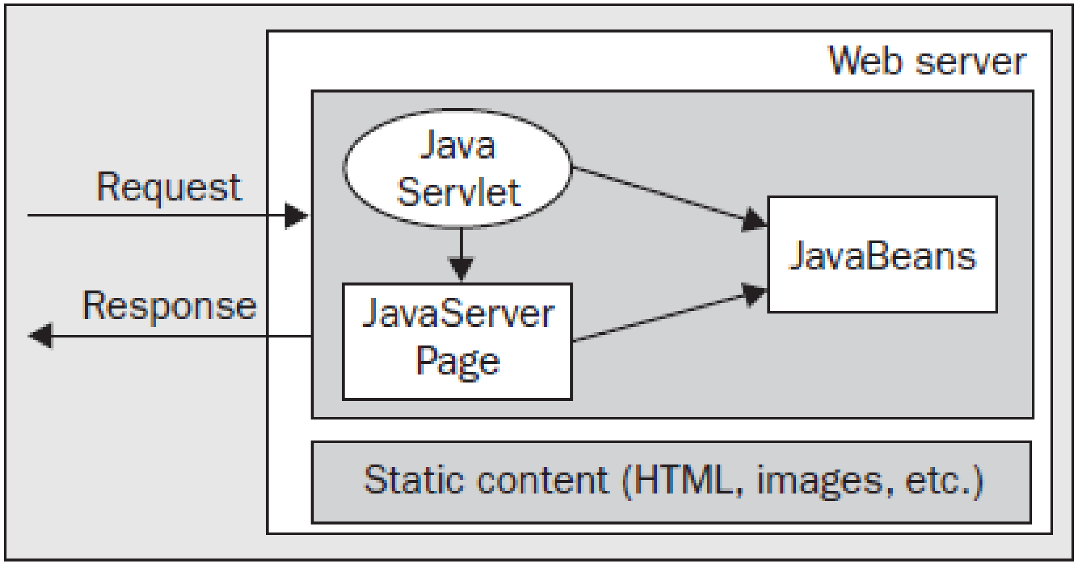
# CSC309: Introduction to Web Programming

# Lecture 11

*Wael Aboulsaadat*

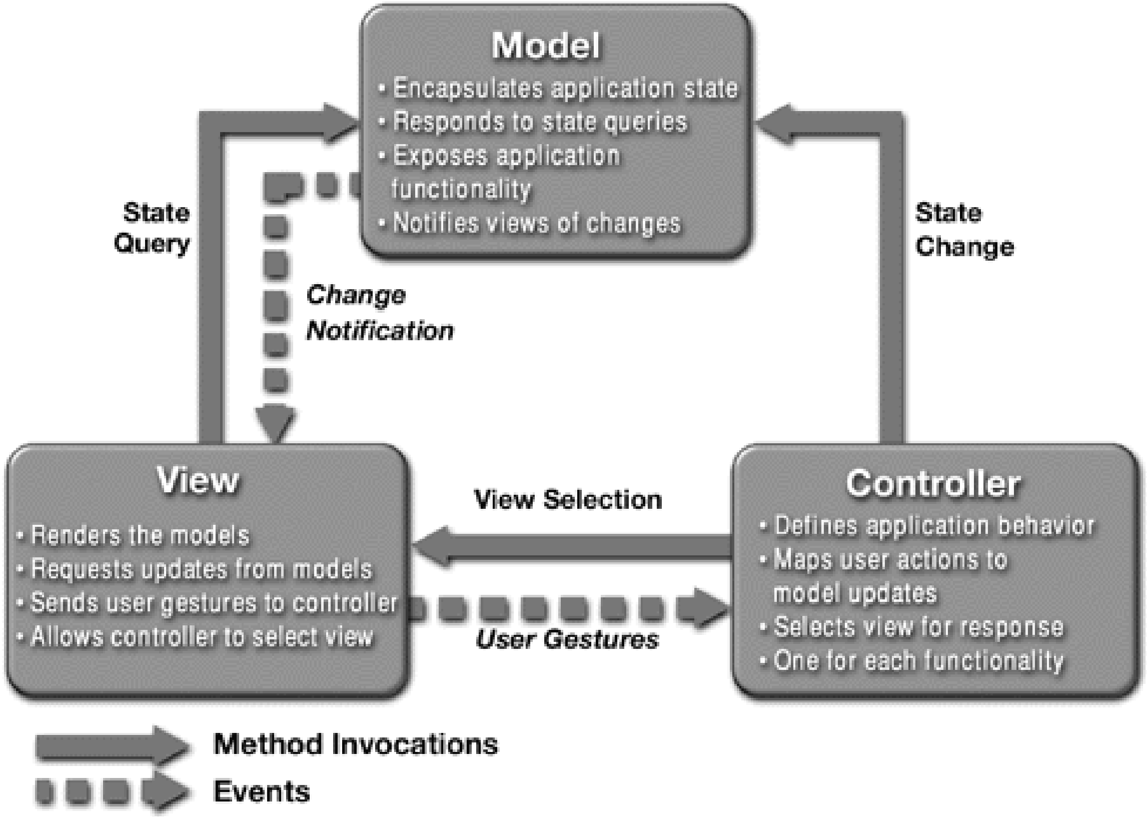
# Servlets+JSP

## Model 2 Architecture



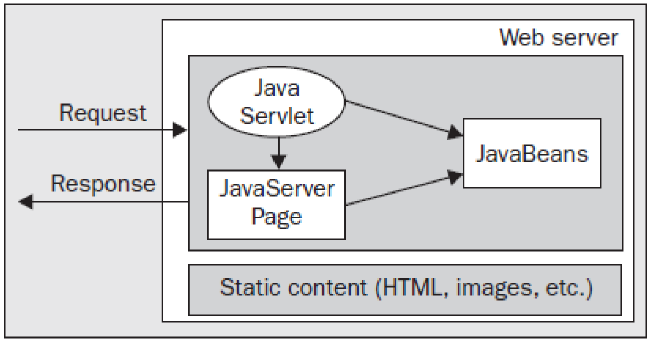
# Servlets+JSP

## Model 2 Architecture = MVC Design Pattern


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

# Servlets+JSP

## Model 2 Architecture

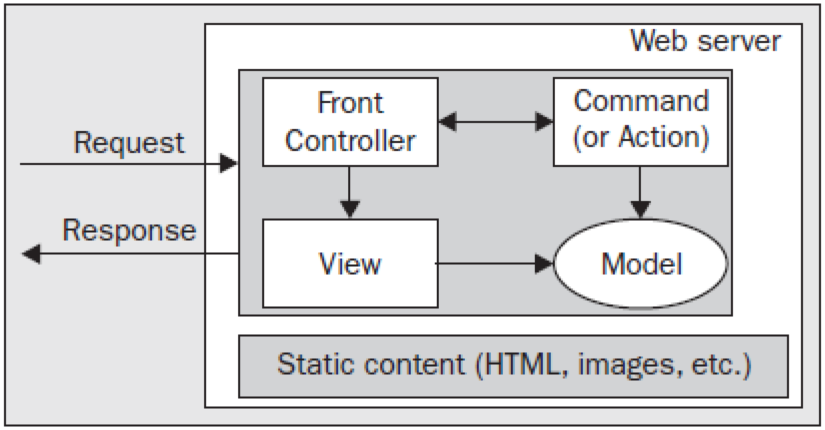


### *Controller Servlet*

- Act as a single point of entry for requests
- Process the requests, accessing and modifying the underlying model
- Delegate the task of presenting information to a specific view component

# Servlets+JSP

## Model 2 Architecture + Command Pattern

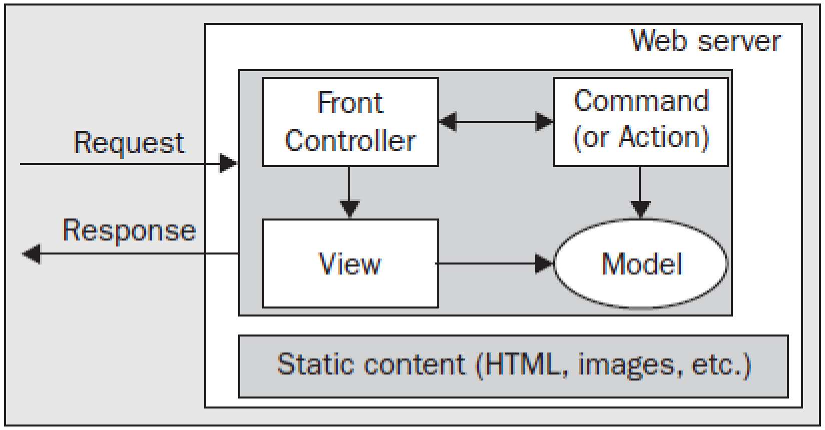


***Controller Servlet***

- Act as a single point of entry for requests
- Process the requests by dispatching it to a specific command
  - (command access and modify the underlying object model)
- Delegate the task of presenting information to a specific view component

# Servlets+JSP

## Model 2 Architecture + Command Pattern

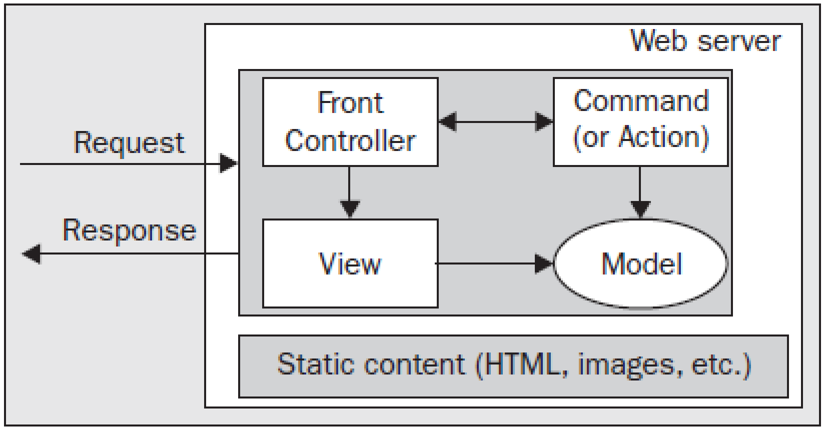


```
import java.util.HashMap;
public class ActionHelper {
private static HashMap actions = new HashMap();
static {
actions.put("ViewTopic", "forum.ViewTopicAction");
actions.put("Login", "forum.LoginAction");
actions.put("Logout", "forum.LogoutAction");
actions.put("NewResponse", "forum.NewResponseAction");
actions.put("ProcessNewResponse",
"forum.ProcessNewResponseAction");
actions.put("DeleteResponse", "forum.DeleteResponseAction");
}
```

```
public static Action getAction(String
name) {
Action action = null;
try {
    Class c =
       Class.forName((String)actions.
               get(name));
     action = (Action)c.newInstance()
}
catch (Exception e) {
e.printStackTrace();
}
return action;
} }
```

# Servlets+JSP

## Model 2 Architecture + Command Pattern

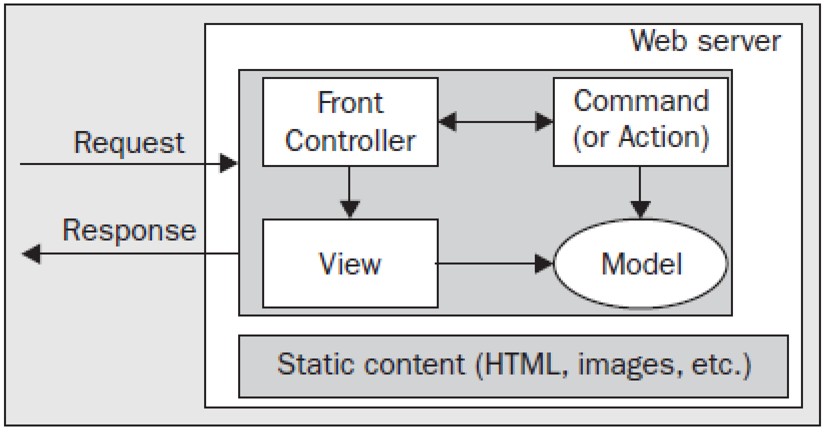


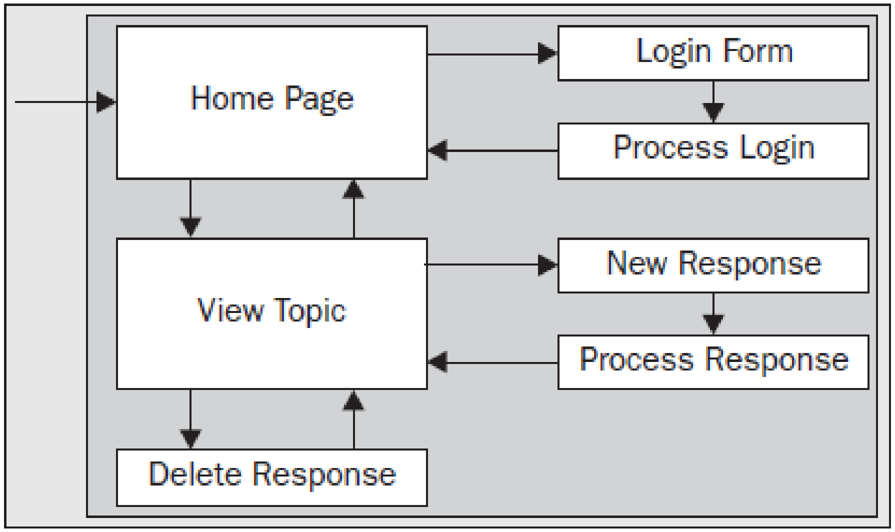
```
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;

public class FrontController extends HttpServlet {

protected void processRequest(     HttpServletRequest req,
                                   HttpServletResponse res)
                         throws ServletException, IOException {

String actionName = req.getPathInfo().substring(1);

Action action = ActionHelper.getAction(actionName);

String nextView = action.process(req, res);
```

```
RequestDispatcher dispatcher =
   getServletContext().
      getRequestDispatcher(nextView);
  dispatcher.forward(req, res);
}

protected void doGet(HttpServletRequest req,
                     HttpServletResponse res)
throws ServletException, IOException {
    processRequest(req, res);
}
```

# Servlets+JSP

## Model 2 Architecture + Command Pattern





# JSP

# JSP Example

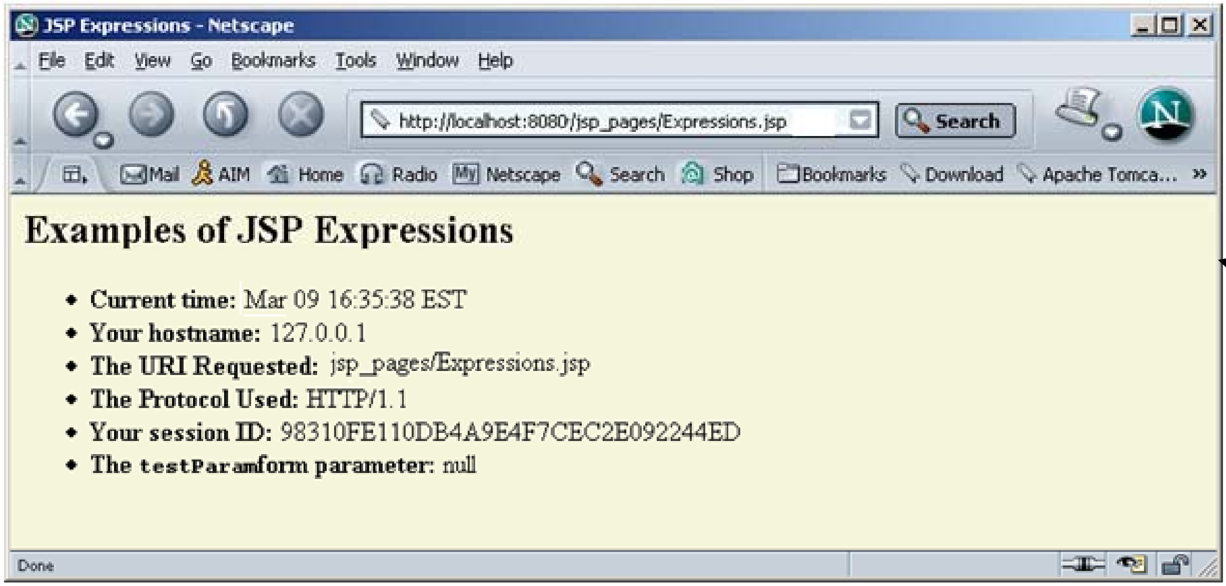
JSP Engine/Container

```
<HTML> <BODY> <H2>JSP Expressions</H2>
<UL>
 <LI>Current time: <%= new java.util.Date() %>
 <LI>Your hostname: <%= request.getRemoteHost() %>
 <LI>The URI Requested: <%= request.getRequestURI() %>
 <LI>The Protocol Used: <%= request.getProtocol() %>
 <LI>Your session ID: <%= session.getId() %>
 <LI>The <CODE>testParam</CODE> form parameter:
     <%= request.getParameter("testParam") %>
</UL> </BODY> </HTML>
```

```
<HTML> <BODY> <H2>JSP Expressions</H2>
<UL>
 <LI>Current time:
 <LI>Your hostname:
 <LI>The URI Requested:
 <LI>The Protocol Used:
 <LI>Your session ID:
 <LI>The <CODE>testParam</CODE> form
parameter:
</UL> </BODY> </HTML>
```

Server side

Client side


JSP Expressions - Netscape
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/jsp_pages/Expressions.jsp
Search
Mail AIM Home Radio Netscape Search Shop Bookmarks Download Apache Tomca...
Done

## Examples of JSP Expressions

- **Current time:** Mar 09 16:35:38 EST
- **Your hostname:** 127.0.0.1
- **The URI Requested:** jsp_pages/Expressions.jsp
- **The Protocol Used:** HTTP/1.1
- **Your session ID:** 98310FE110DB4A9E4F7CEC2E092244ED
- **The `testParam`form parameter:** null

# The JSP Framework

- Idea:
  - Use regular HTML for most of page
  - Mark servlet code with special tags
  - Entire JSP page gets translated into a servlet (once), and servlet is what actually gets invoked (for each request)

# JSP elements reviewed

- JSP elements include:
  - □ Scriptlet  enclosed in **<%** and **%>** markers: a small script in Java to perform arbitrary functions.  Executed in the underlying servlet context.
  - □ Expression:  anything between **<%=** and **%>** markers is evaluated by the JSP engine as a Java expression in the servlet context.
  - □ JSP directive enclosed in **<%@**  and **%>** markers—passes information to the JSP engine (guides “compilation”).
  - □ ***JSP actions or tags*** are a set of customizable XML-style tags for particular actions, e.g. predefine **jsp:useBean** instantiates a JavaBean class on the server.

# JSP makes extensive use of Java Beans

- JavaBeans API provides a standard format for Java classes
- Supported by
  - Visual manipulation tools
  - Auto-discovery of class information
- Basic criteria for Beans
  - Zero argument constructor
  - No public variables
  - Access methods in the format of
    - getXxx
    - setXxx
    - isXxx

# Example JavaBean

```
public class PersonBean implements java.io.Serializable {

    private String name;
    private boolean deceased;

    /** No-arg constructor (takes no arguments). */
    public PersonBean() {
    }

    public String getName() {
        return this.name;
    }

   public void setName(final String name) {
        this.name = name;
    }
```

```
    public boolean isDeceased() {
        return this.deceased;
     }

   public void setDeceased(final
           boolean deceased)
    {
        this.deceased = deceased;
     }
}
```

# How to use a Bean in JSP?

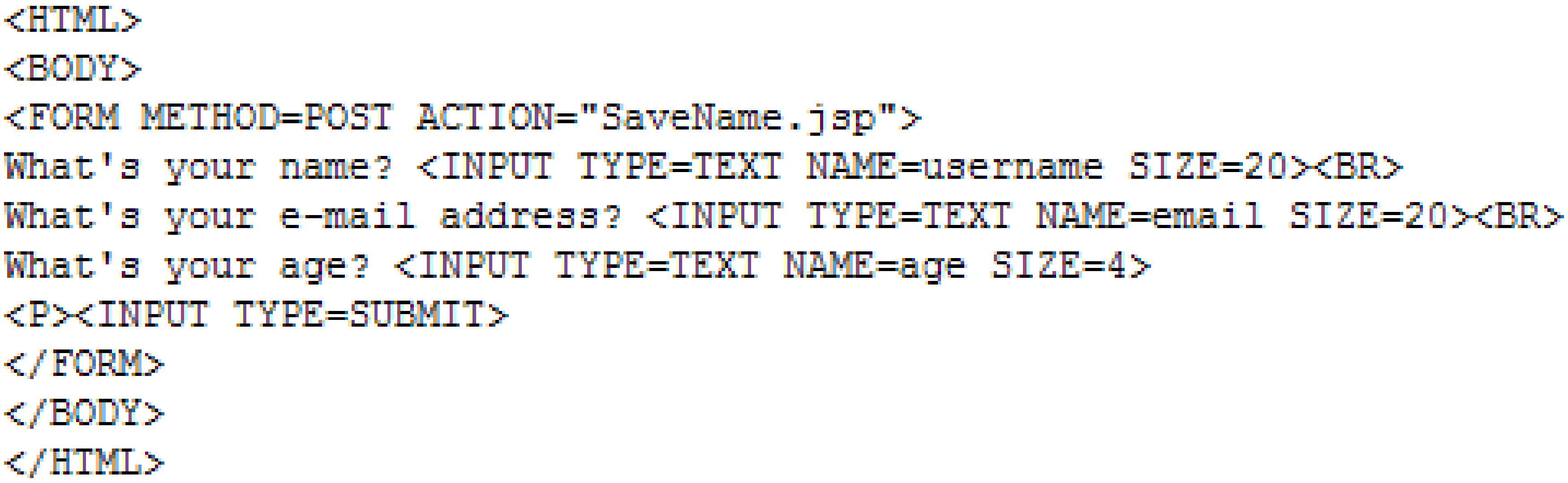
- E.g.

  <jsp:useBean id=“book1” class=“Book” />

  - □ Creates an instance of Book and bind to book1
  - □ It may be thought as equivalent to

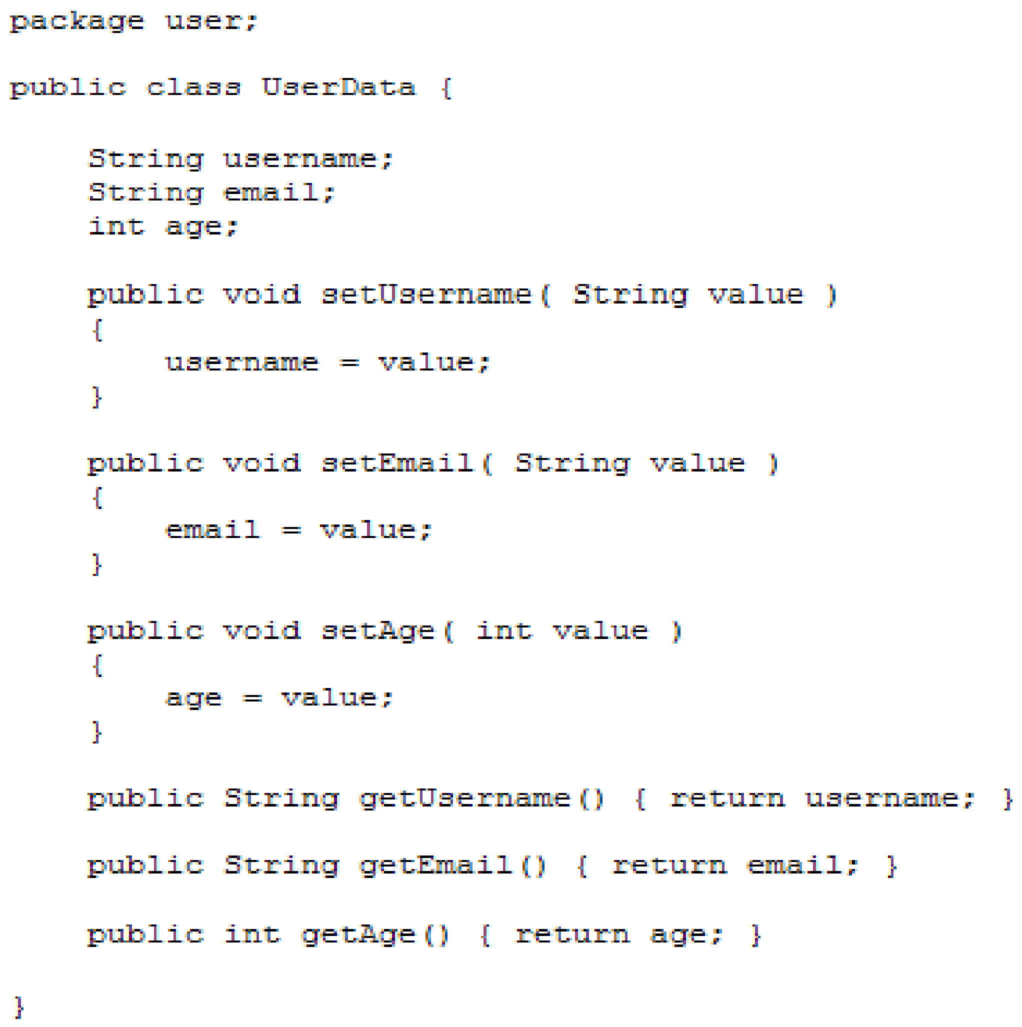
  <% Book book1= new coreservlets.Book() %>

# JSP form handling via Beans (e.g.)



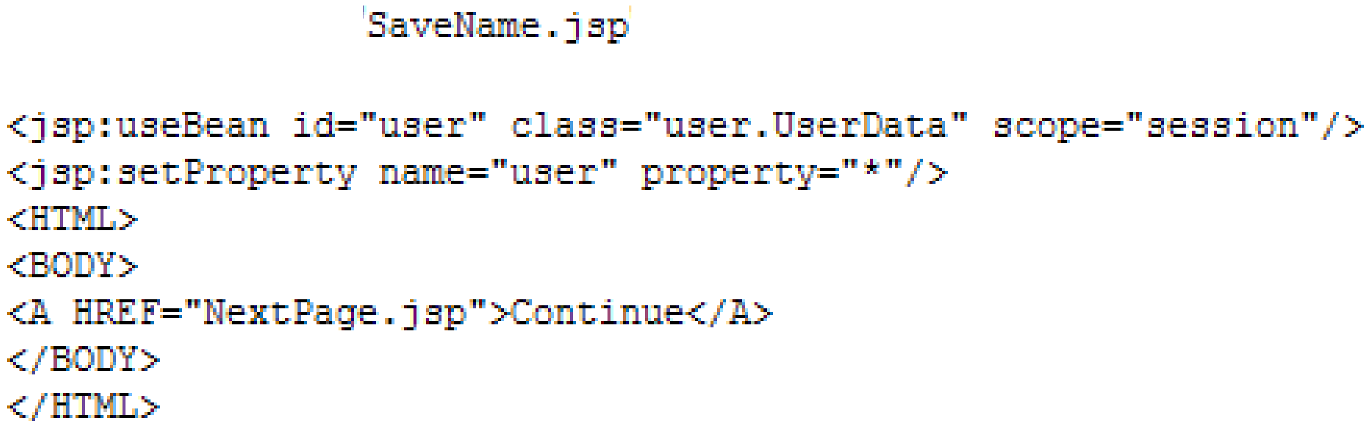
```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username SIZE=20><BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email SIZE=20><BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

# JSP form handling via Beans (e.g.)



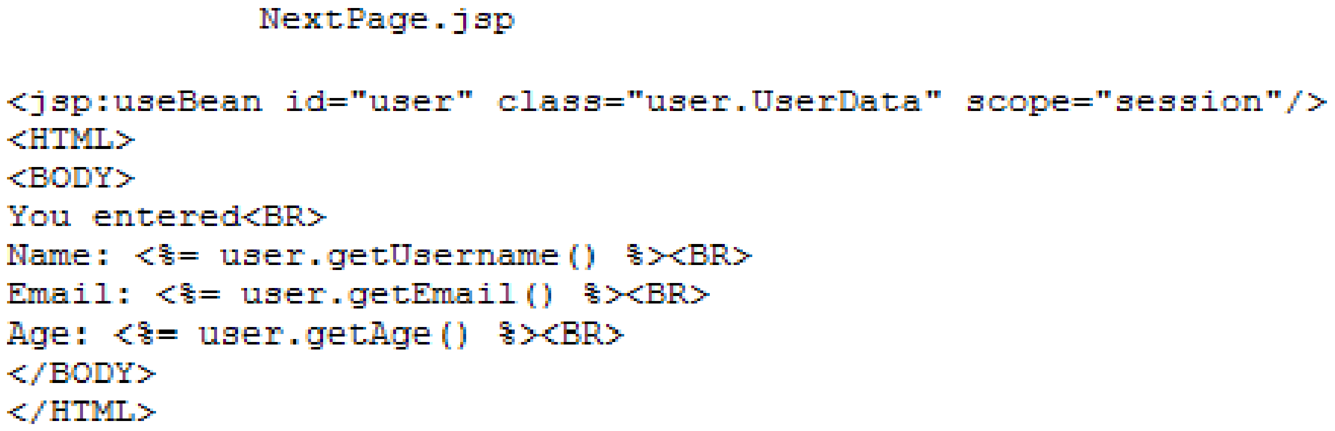
```
package user;

public class UserData {

    String username;
    String email;
    int age;

    public void setUsername( String value )
    {
        username = value;
    }

    public void setEmail( String value )
    {
        email = value;
    }

    public void setAge( int value )
    {
        age = value;
    }

    public String getUsername() { return username; }

    public String getEmail() { return email; }

    public int getAge() { return age; }

}
```

# JSP form handling via Beans (e.g.)



SaveName.jsp

```
<jsp:useBean id="user" class="user.UserData" scope="session"/>
<jsp:setProperty name="user" property="*"/>
<HTML>
<BODY>
<A HREF="NextPage.jsp">Continue</A>
</BODY>
</HTML>
```



NextPage.jsp

```
<jsp:useBean id="user" class="user.UserData" scope="session"/>
<HTML>
<BODY>
You entered<BR>
Name: <%= user.getUsername() %><BR>
Email: <%= user.getEmail() %><BR>
Age: <%= user.getAge() %><BR>
</BODY>
</HTML>
```

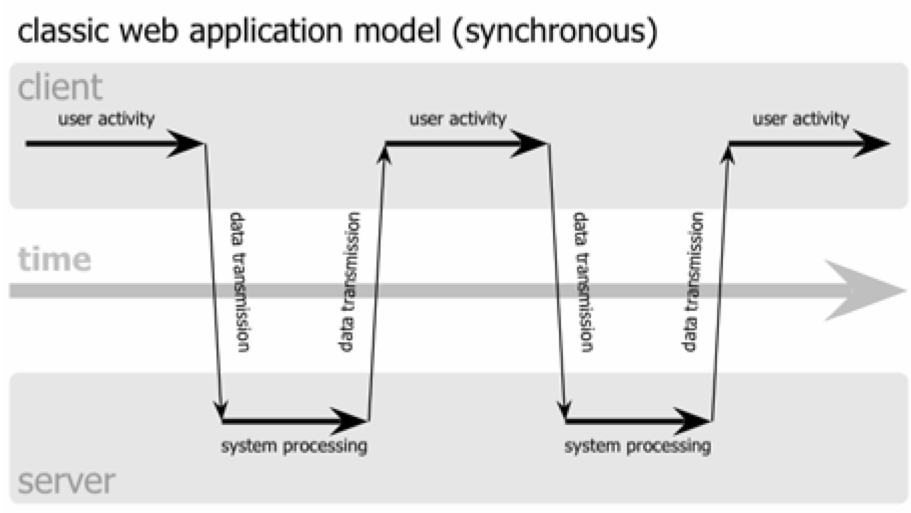
# AJAX

# What is AJAX ?

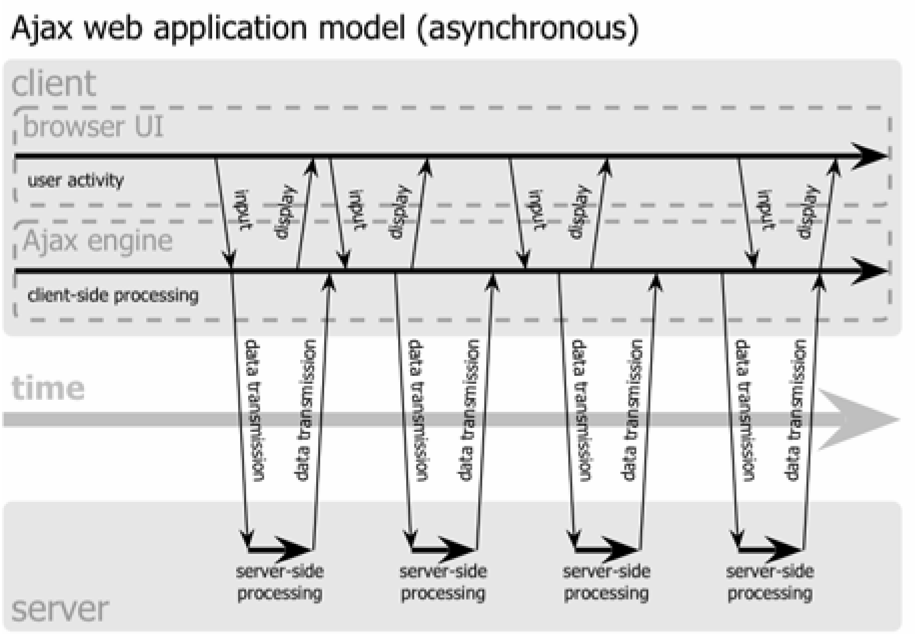
- Ajax: Asynchronous JavaScript and XML

- The word "**Asynchronous**" in AJAX means that the request to the server will be made. The response will be made available by the server after it has finished processing the request, without having to wait for it explicitly, to come back. i.e. you *don’t have to wait for an answer.*

# What is AJAX ? – cont'd

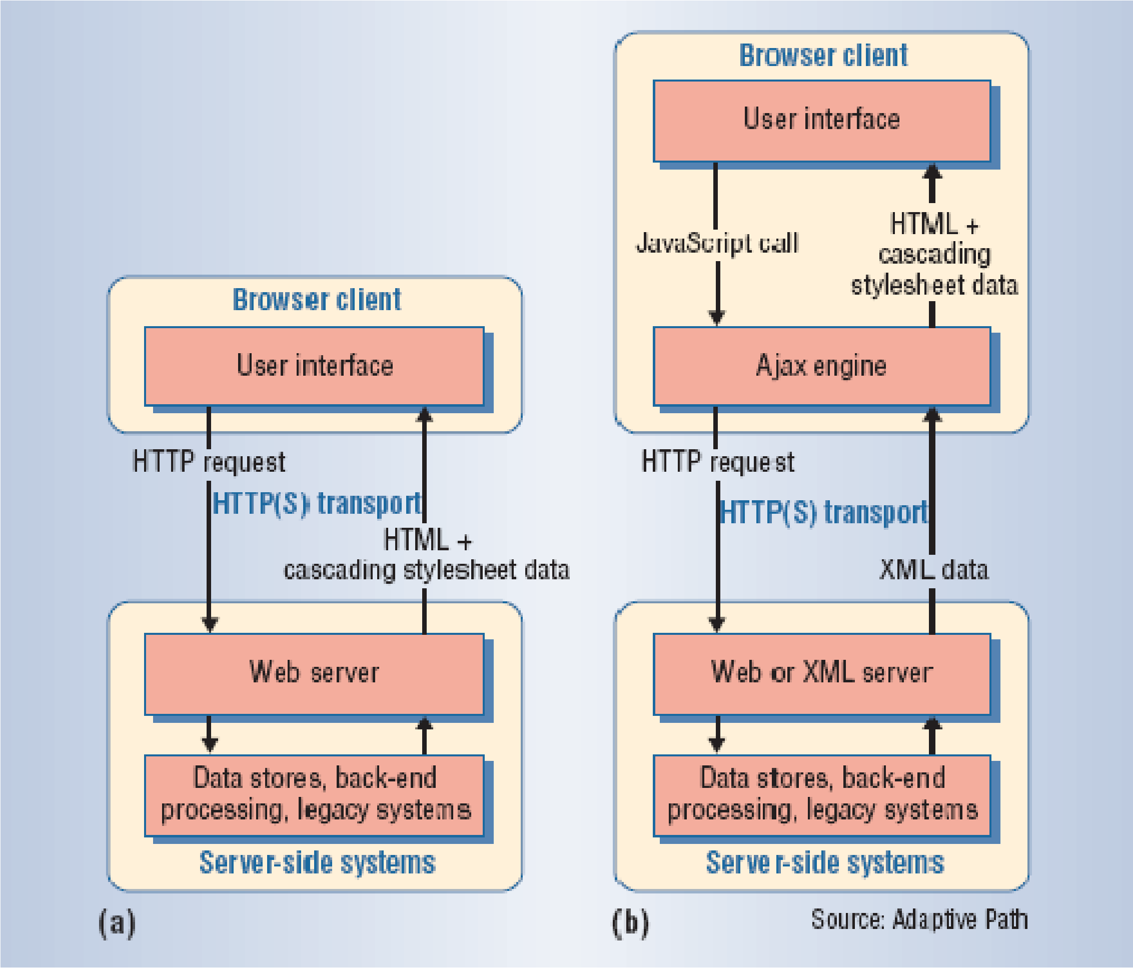


Delayed response due to processing by server



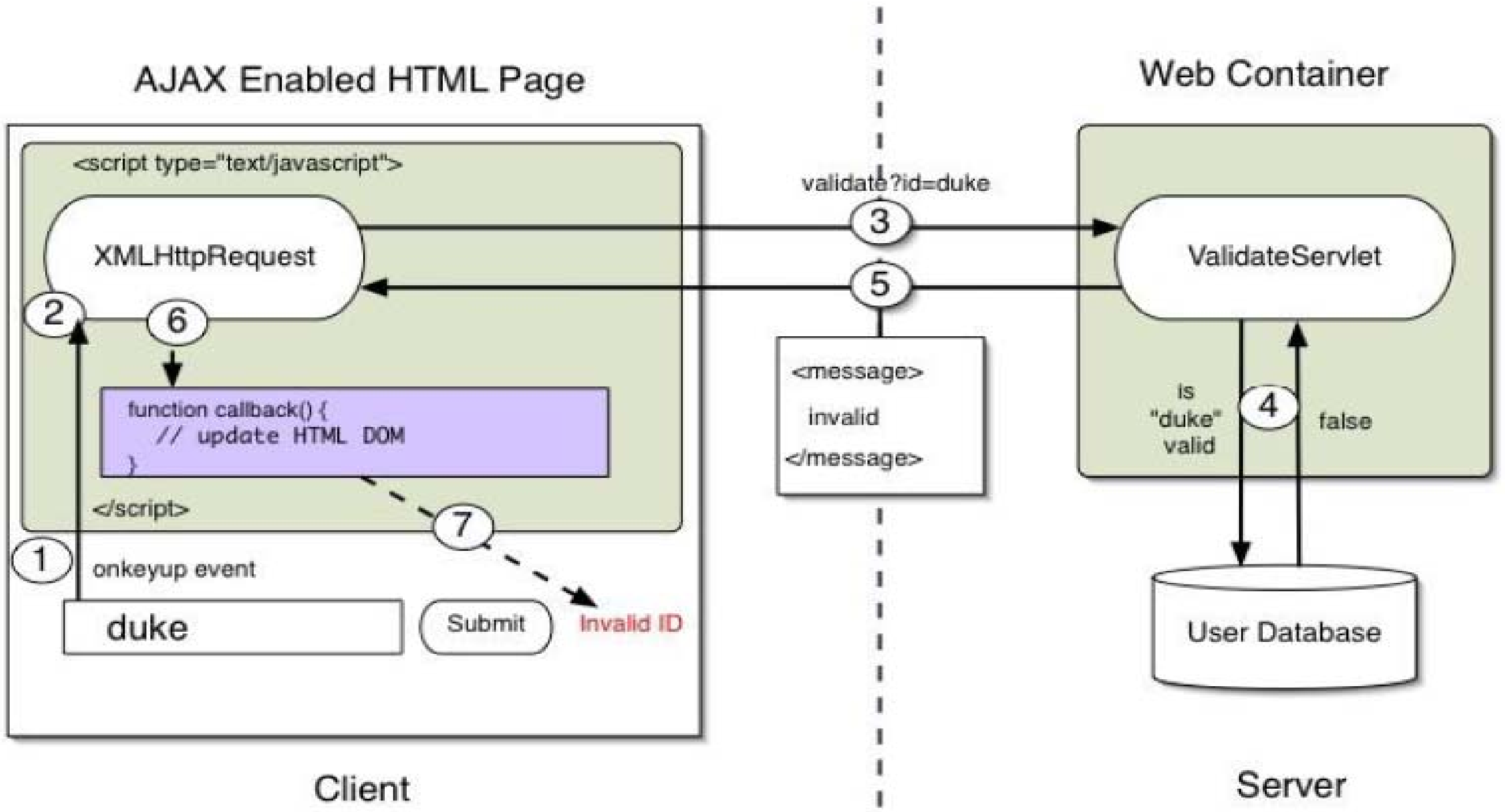
Immediate response

# What is AJAX ? – cont'd



Source: Adaptive Path

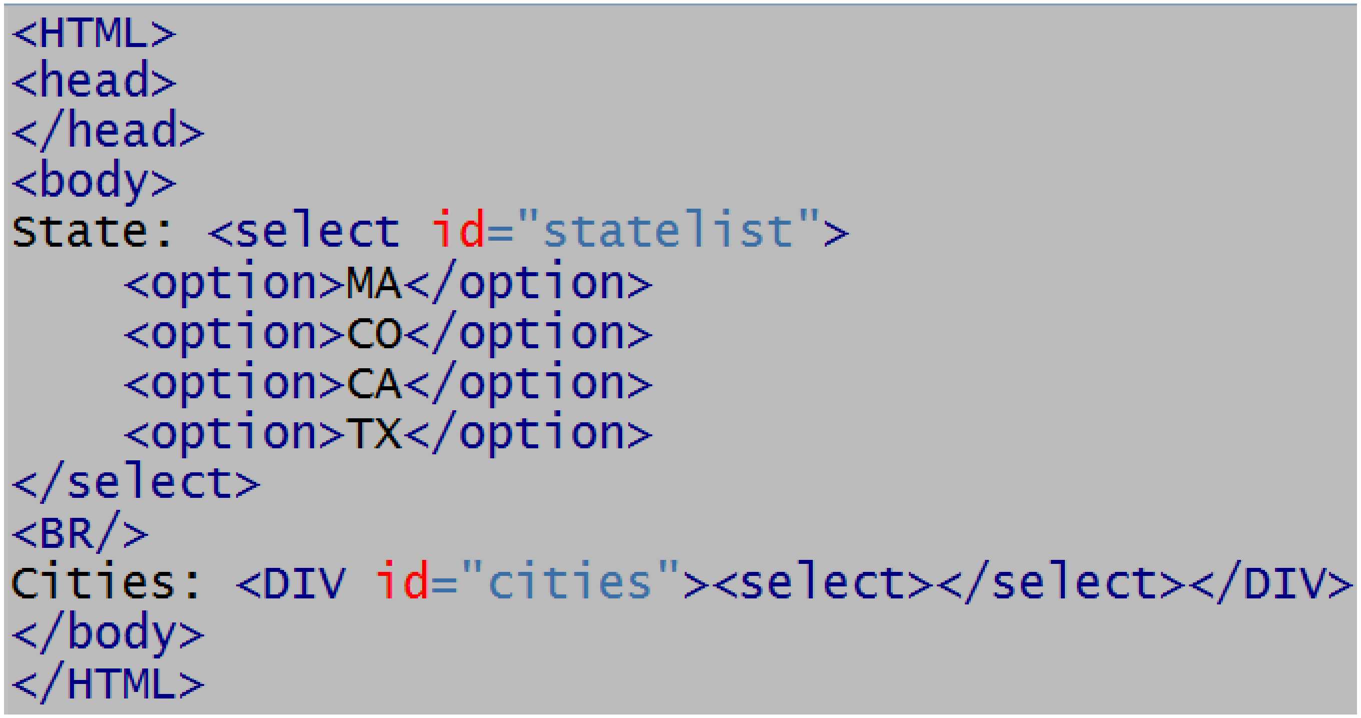
# What is AJAX ? – cont'd



# Let’s Create a Contrived Example

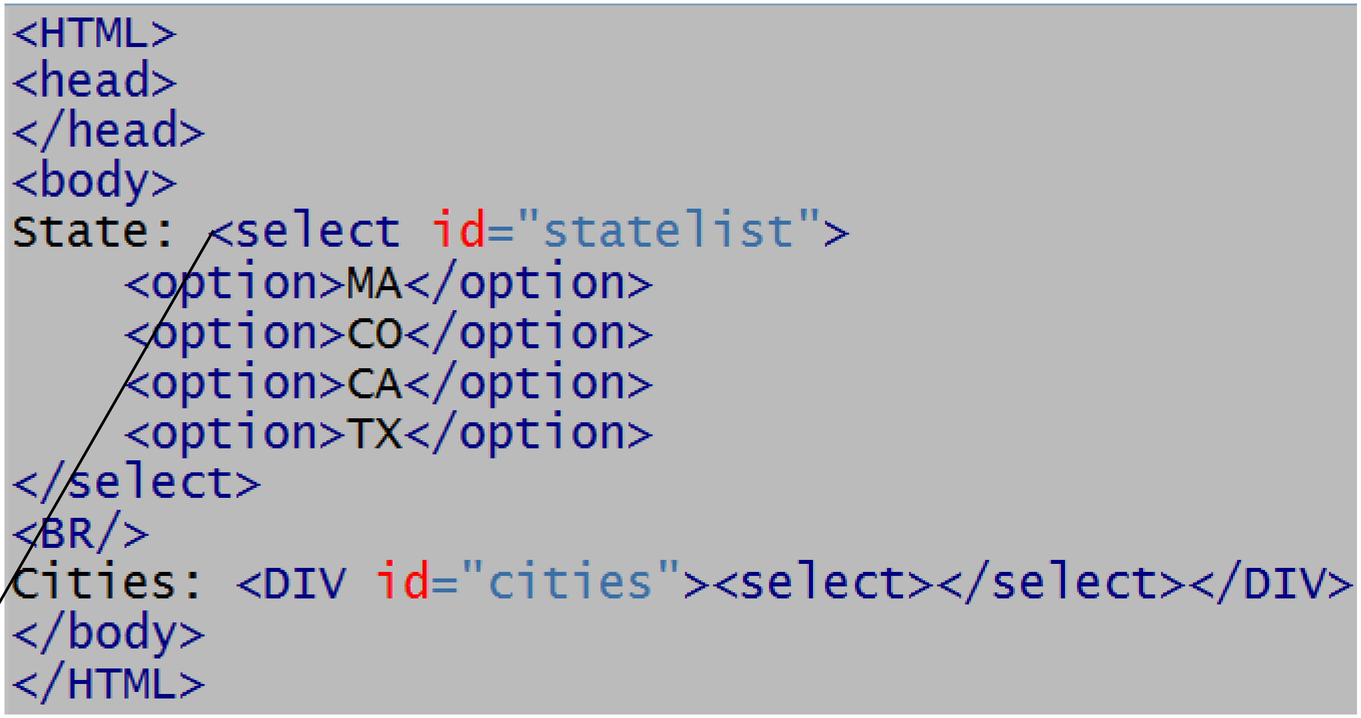
- Would be nice to see Ajax at work without worrying about server side code
- So, here is a contrived example
- We will have static pages on server
- Based on user selection, we will pull appropriate data from these pages and display

# Starting with HTML

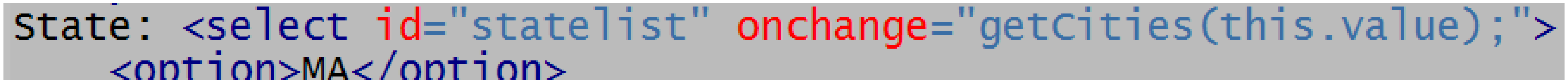


```
<HTML>
<head>
</head>
<body>
State: <select id="statelist">
    <option>MA</option>
    <option>CO</option>
    <option>CA</option>
    <option>TX</option>
</select>
<BR/>
Cities: <DIV id="cities"><select></select></DIV>
</body>
</HTML>
```

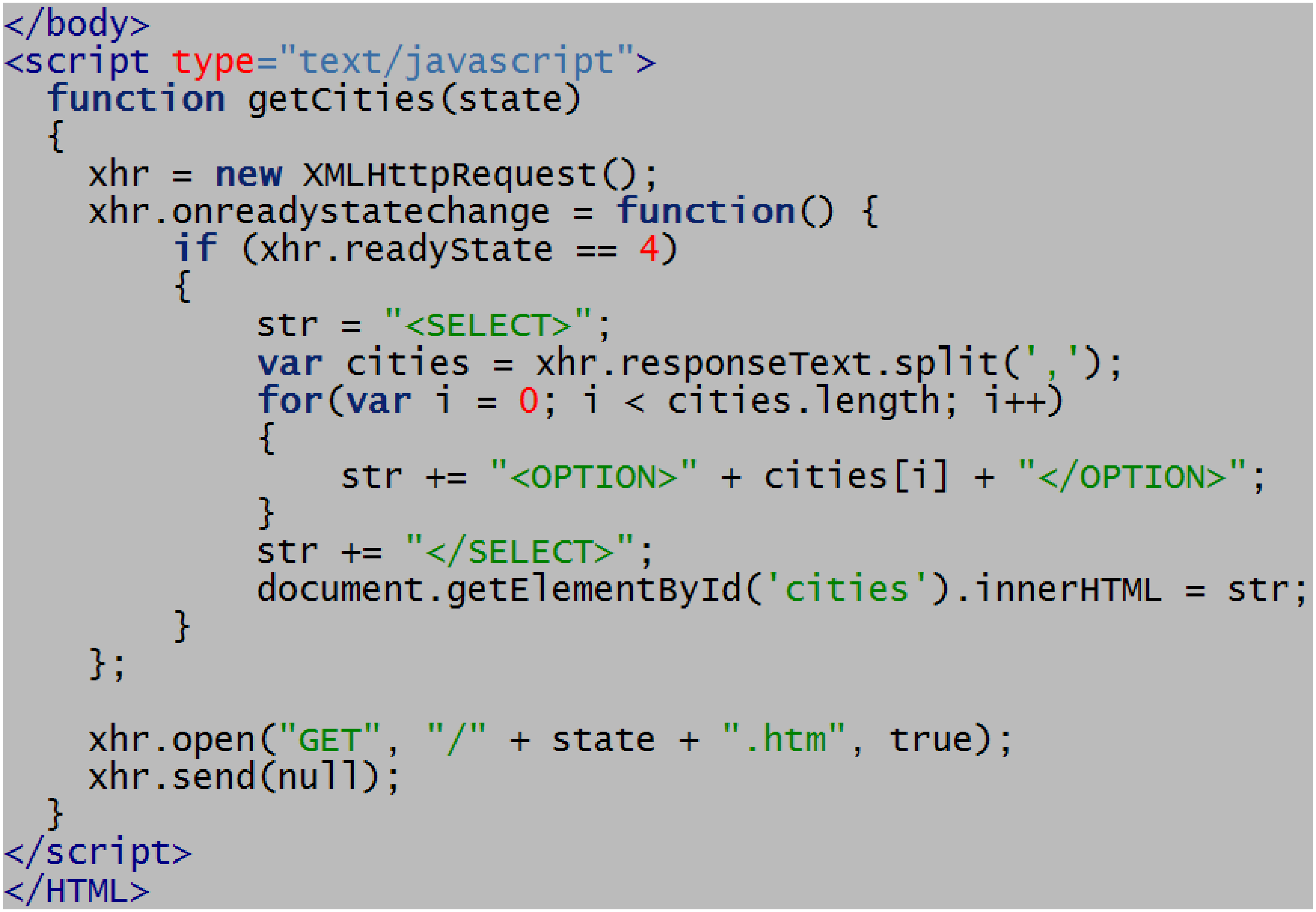
# Providing an Event Handler



```
<HTML>
<head>
</head>
<body>
State: <select id="statelist">
    <option>MA</option>
    <option>CO</option>
    <option>CA</option>
    <option>TX</option>
</select>
<BR/>
Cities: <DIV id="cities"><select></select></DIV>
</body>
</HTML>
```

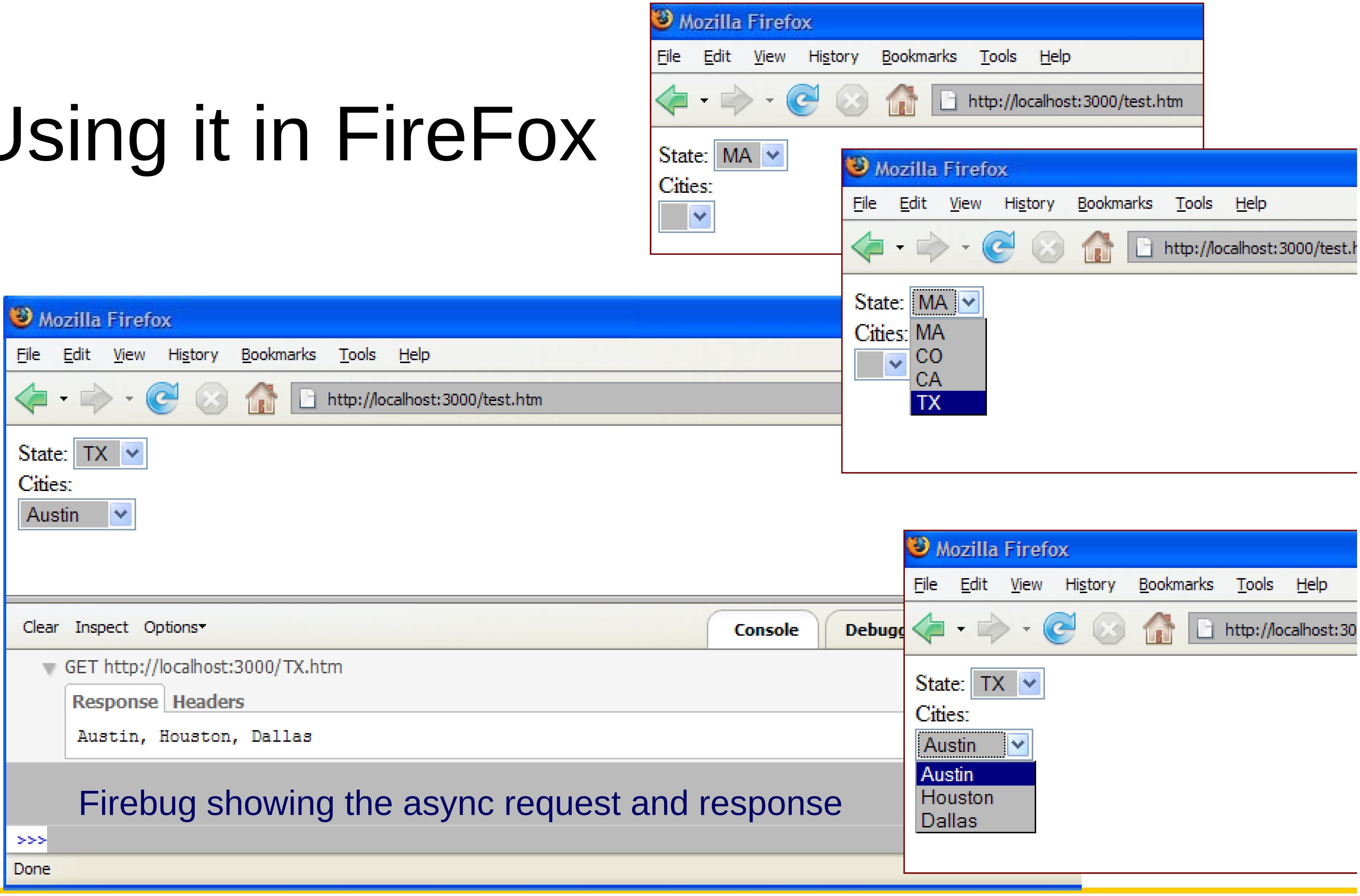


```
State: <select id="statelist" onchange="getCities(this.value);">
    <option>MA</option>
```



```
</body>
<script type="text/javascript">
 function getCities(state)
 {
  xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4)
    {
      str = "<SELECT>";
      var cities = xhr.responseText.split(',');
      for(var i = 0; i < cities.length; i++)
      {
        str += "<OPTION>" + cities[i] + "</OPTION>";
      }
      str += "</SELECT>";
      document.getElementById('cities').innerHTML = str;
    }
  };

  xhr.open("GET", "/" + state + ".htm", true);
  xhr.send(null);
 }
</script>
</HTML>
```

# Using it in FireFox



http://localhost:3000/test.htm

State: MA
Cities:

State: MA
Cities:
MA
CO
CA
TX

State: TX
Cities:
Austin

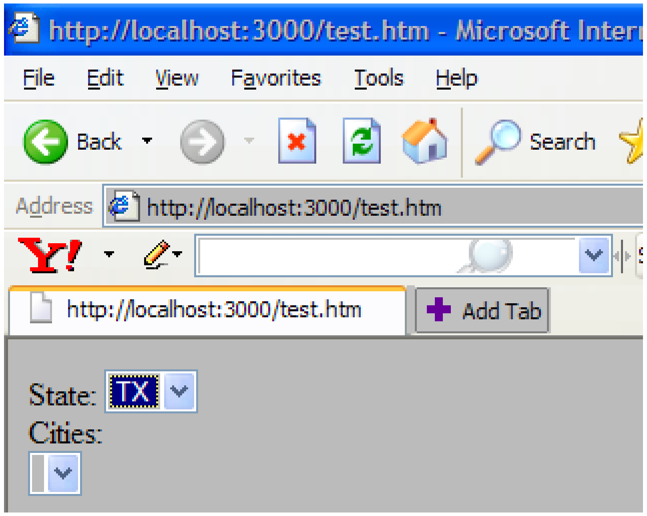
GET http://localhost:3000/TX.htm

```
Austin, Houston, Dallas
```

State: TX
Cities:
Austin
Austin
Houston
Dallas

Firebug showing the async request and response

# How about using IE?


http://localhost:3000/test.htm - Microsoft Inter
File Edit View Favorites Tools Help
Back
Search
Address
http://localhost:3000/test.htm
Y!
http://localhost:3000/test.htm
Add Tab

State: TX

Cities:

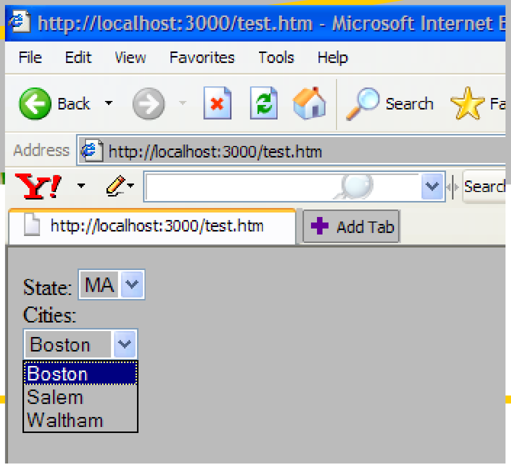
Does not work!

# Microsoft started it!…

- Microsoft started it with the ActiveX object
- Other browsers have followed, but…
- No standard
- Making it work with IE…

# For IE

```
State: <select id="statelist" onchange="getCities(this.value);">
    <option value="MA">MA</option>
    <option value="CO">CO</option>
    <option value="CA">CA</option>
    <option value="TX">TX</option>
</select>
<BR/>
Cities: <DIV id="cities"><select></select></DIV>
</body>
<script type="text/javascript">
  function getCities(state)
  {
    xhr = new ActiveXObject("Microsoft.XMLHTTP");
        // Earlier versions used new ActiveXObject("Msxml2.XMLHTTP");
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4)
        {
            str = "<SELECT>";
```


http://localhost:3000/test.htm - Microsoft Internet E
File
Edit
View
Favorites
Tools
Help
Back
Search
Fa
Address
http://localhost:3000/test.htm
Searc
http://localhost:3000/test.htm
Add Tab

State: MA

Cities:

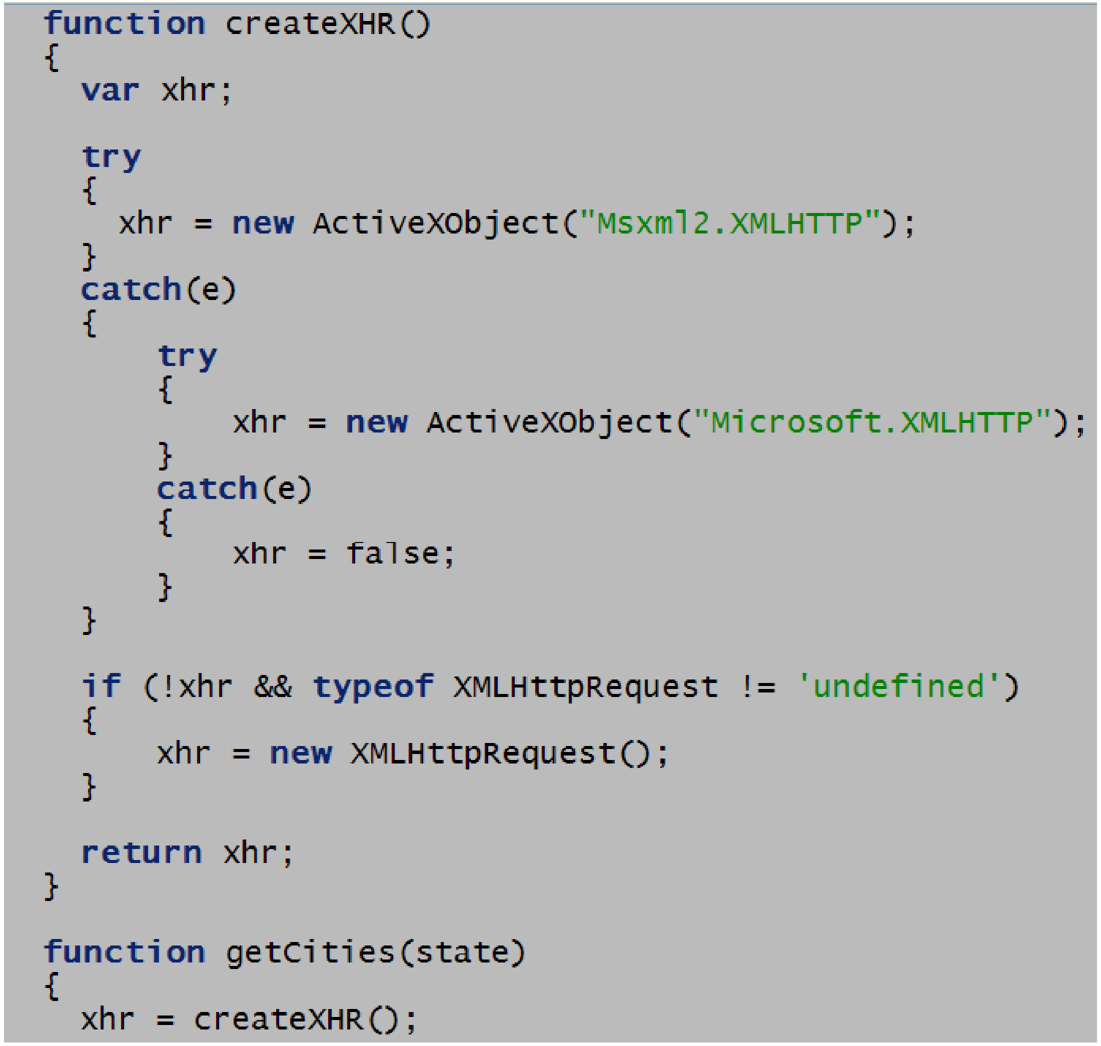
Boston

Boston
Salem
Waltham

# But, then…

- FireFox does not work any more!
- What about other browser variations?
- Need to deal with these differences…

# Dealing with differences…



```
function createXHR()
{
  var xhr;

  try
  {
    xhr = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch(e)
  {
      try
      {
          xhr = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch(e)
      {
          xhr = false;
      }
  }

  if (!xhr && typeof XMLHttpRequest != 'undefined')
  {
      xhr = new XMLHttpRequest();
  }

  return xhr;
}

function getCities(state)
{
  xhr = createXHR();
```

# Rich Internet Applications (RIA) using Ajax

- Examples:
  - http://www.icefaces.org/main/demos/
  - http://demo.backbase.com/explorer/#|examples/welcome.xml
  - And DHTMLX…

# Other (RIA) Technologies

- Applet (Swing)
- Macromedia Flash
- Java WebStart
- DHTML
- DHTML with Hidden IFrame

# What is AJAX ? – cont’d

- Ajax isn’t a new technology or programming language.
- It is a technique used to develop interactive web applications that are able to process a user request immediately.

# Ajax Component technologies

- Dynamic HTML
- XML
- Cascading stylesheets (.css)
- Document object model (DOM)
- JavaScript
- XMLHttpRequest

# How AJAX Works ?

1. A JavaScript function creates and configures an XMLHttpRequest object on the client, and specifies a JavaScript callback function.

2. The XMLHttpRequest object makes an asynchronous call to the web server.

3. The web server processes the request and returns an XML document that contains the result.

4. The XMLHttpRequest object calls the callback function and exposes the response from the web server so that the request can be processed.

5. The client updates the HTML DOM representing the page

# XMLHttpRequest ?

- XMLHttpRequest object is the key to Ajax programming.
- It’s main purpose is to put an asynchronous http request to the web server.
- Because of this asynchronous call to the web server, you are allowed to continue using the page without the interruption of a browser refresh and the loading of a new or revised page.
- This object has few properties.

# Properties of XMLHttpRequest

- Property 1: objXML*Http.onreadystatechange*

  This property holds the reference of function which is going to process the response from the server.

*objXMLHttp.onreadystatechange = procRequest;*

* *"procRequest "  is the function which will process the response*

# Properties of XMLHttpRequest

- Property 2 : objXML*Http. readyState*

  This property holds the status of server response.

  *objXMLHttp.readyState = [state];*

| State | Description |
|---|---|
| 0 | The request is not initialized |
| 1 | The request has been set up |
| 2 | The request has been sent |
| 3 | The request is in process |
| 4 | The request is complete |

# Pros

- Ajax applications are more responsive to user actions and the programs don’t experience page-reloading-related interruptions.
- Ajax applications are usually fast because the approach minimizes traffic to the server by sending and requesting just the minimum amount of data needed.

# Cons

- When pages with Ajax are indexed, the data displayed by Ajax calls is not present in the page
  - ***workarounds*** *are required to get dynamic Ajax content indexed*
  - *it's simply not there in the page!*
- Content rendered via AJAX is done via javascript
  - Search engine spiders cannot invoke javascript events to retrieve AJAX content

# Cons - contd…

- Increased development time and costs
- Available frameworks and components still need to completely mature
- Not all concerns regarding security and user privacy have been answered
- Conflicts in modern browsers - navigate and create bookmarks

# Cons - contd…

- Not meant to be used in every application
- Concern of accessibility
- XMLHttpRequest object itself

# Workarounds - Go on an AJAX diet

- Use hidden divs rather than Ajax
  - *Search engines will index the content*
- Need AJAX content to be indexed
  - *Provide hidden links that can be spidered*
- Limit the complexity of the logic
  - *Javascript-free Ajax*
- AJAX should be limited to providing a better user experience through RIAs (Rich Internet Applications)

# Conclusion

- Keeping the advantages and disadvantages in mind one has to choose whether or not to have AJAX to retrieve the content/data in their application.

# Third-party libraries and frameworks

- Cross-browser libraries
  - □ X library
  - □ Sarissa
  - □ Prototype
- Widgets and widget suites
  - □ Scriptaculous
  - □ Rico
- Application frameworks
  - □ DWR, JSON, JPSpan (down), and SAJAX
  - □ Backbase
  - □ Echo2
  - □ Ruby on Rails
  - □ Ajax.Net