UNIT 4
Server Site Programming

Introduction to active server pages (ASP)

- ASP stands for **A**ctive **S**erver **P**ages
- ASP is a Microsoft Technology
- ASP is a program that runs inside **IIS**
- IIS stands for **I**nternet **I**nformation **S**ervices
- IIS comes as a free component with **Windows 2000**
- IIS is also a part of the **Windows NT 4.0 Option Pack**
- The Option Pack can be **downloaded** from Microsoft
- **PWS** is a smaller - but fully functional - version of IIS
- PWS can be found on your **Windows 95/98 CD**
- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML, XML, and scripts
- Scripts in an ASP file are executed on the server
- An ASP file has the file extension ".asp"

ASP.NET

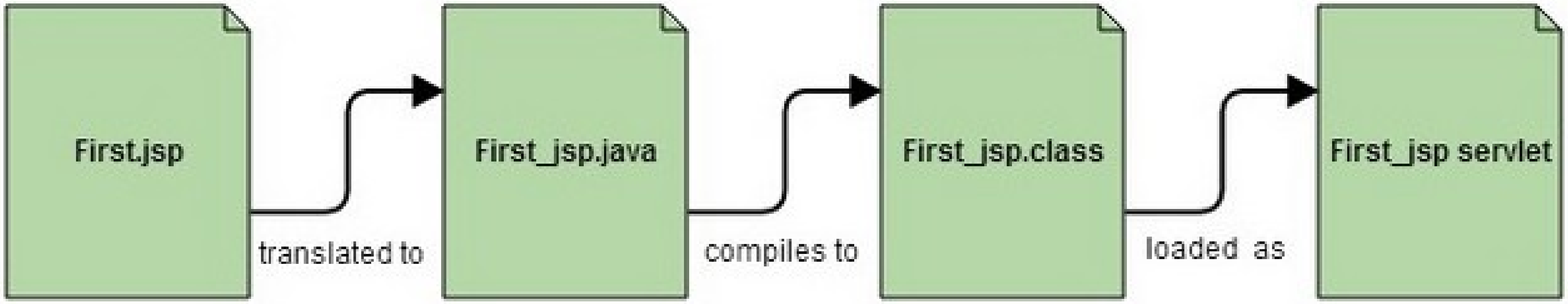
- ASP.NET is a new ASP generation. It is not compatible with Classic ASP, but ASP.NET may include Classic ASP.
- ASP.NET pages are compiled, which makes them faster than Classic ASP.
- ASP.NET has better language support, a large set of user controls, XML-based components, and integrated user authentication.
- ASP.NET pages have the extension .aspx, and are normally written in VB (Visual Basic) or C# (C sharp).
- User controls in ASP.NET can be written in different languages, including C++ and Java.
- When a browser requests an ASP.NET file, the ASP.NET engine reads the file, compiles and executes the scripts in the file, and returns the result to the browser as plain HTML

Introduction to JSP

- **JSP** technology is used to create web application. It focuses more on presentation logic of the web apllication.**JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets. Servlet adds HTML code inside Java code while JSP adds Java code inside HTML. Everything a Servlet can do, a JSP page can also do it.

- JSP enables us to write HTML pages containing tags that run powerful Java programs. **JSP separates presentation and business logic** as Web designer can design and update JSP pages without learning the Java language and Java Developer can also write code without concerning the web design.

JSP processing



First Simple Interactive JSP example

- *helloJsp.jsp* Hello User example: the HTML page takes a user name from a HTML form and sends a request to a JSP page, and JSP page generates a dynamic HTML greeting page based on the data which comes with the request

- The request may come from a Web form page request or from a query string following an URL address of this JSP page.

- HTML file named *index.html* is placed in the *JSP* directory which is this Web application ROOT directory under *webapps*.

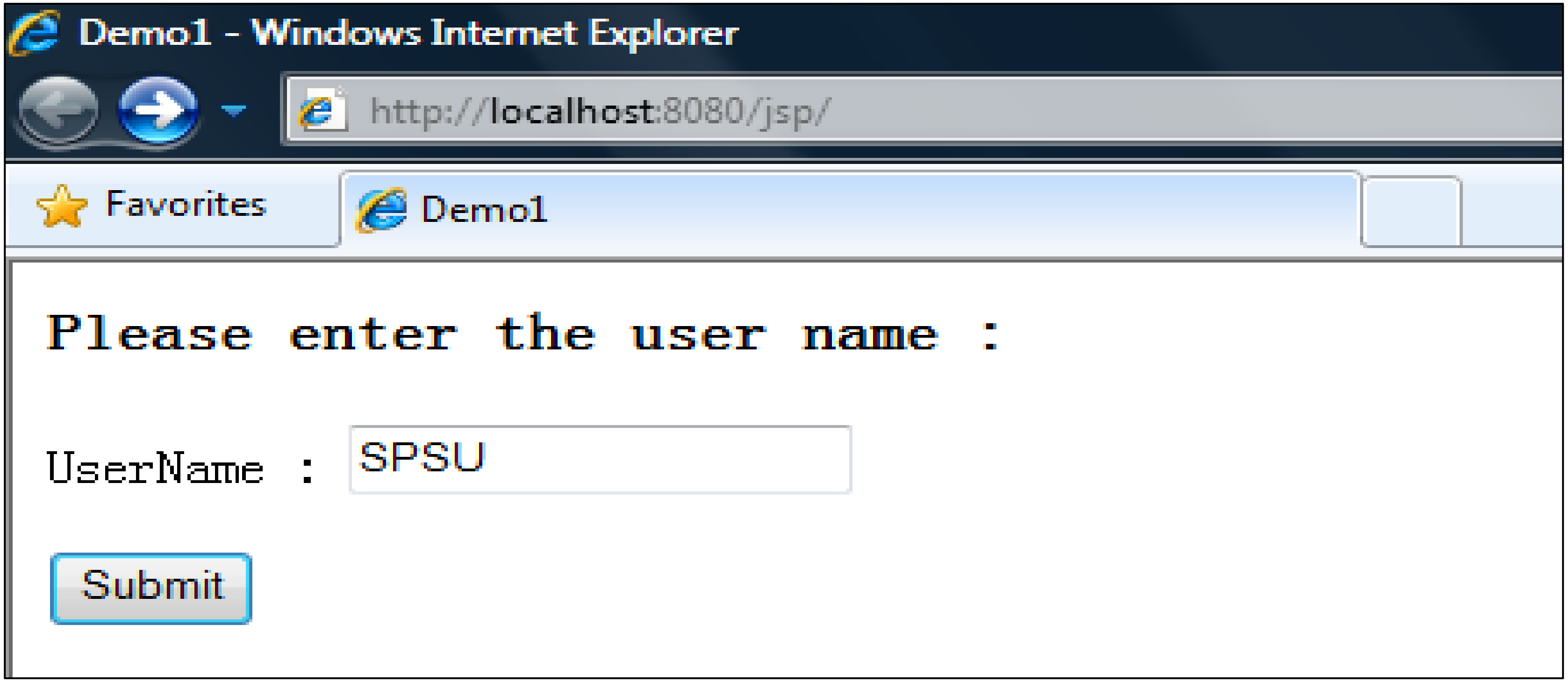
First Simple Interactive JSP example (cont.)

```
<html>
<head>
  <title>Demo1</title>
</head>
<body>
<h3>Please enter the user name :</h3><p>
<form action="/jsp/helloJsp.jsp">
UserName : <input type="text" name="userName"><br><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Since index.html is the default html name for Tomcat that you even don’t need to specify the index.html in the URL address of the browser.

First Simple Interactive JSP example (cont.)

- This HTML takes a string of a user name from the HTML form and submits a request to *helloJsp.jsp* as specified in the action attribute of the HTML form. For example, a user types SPSU in the form and pushes the Submit button.



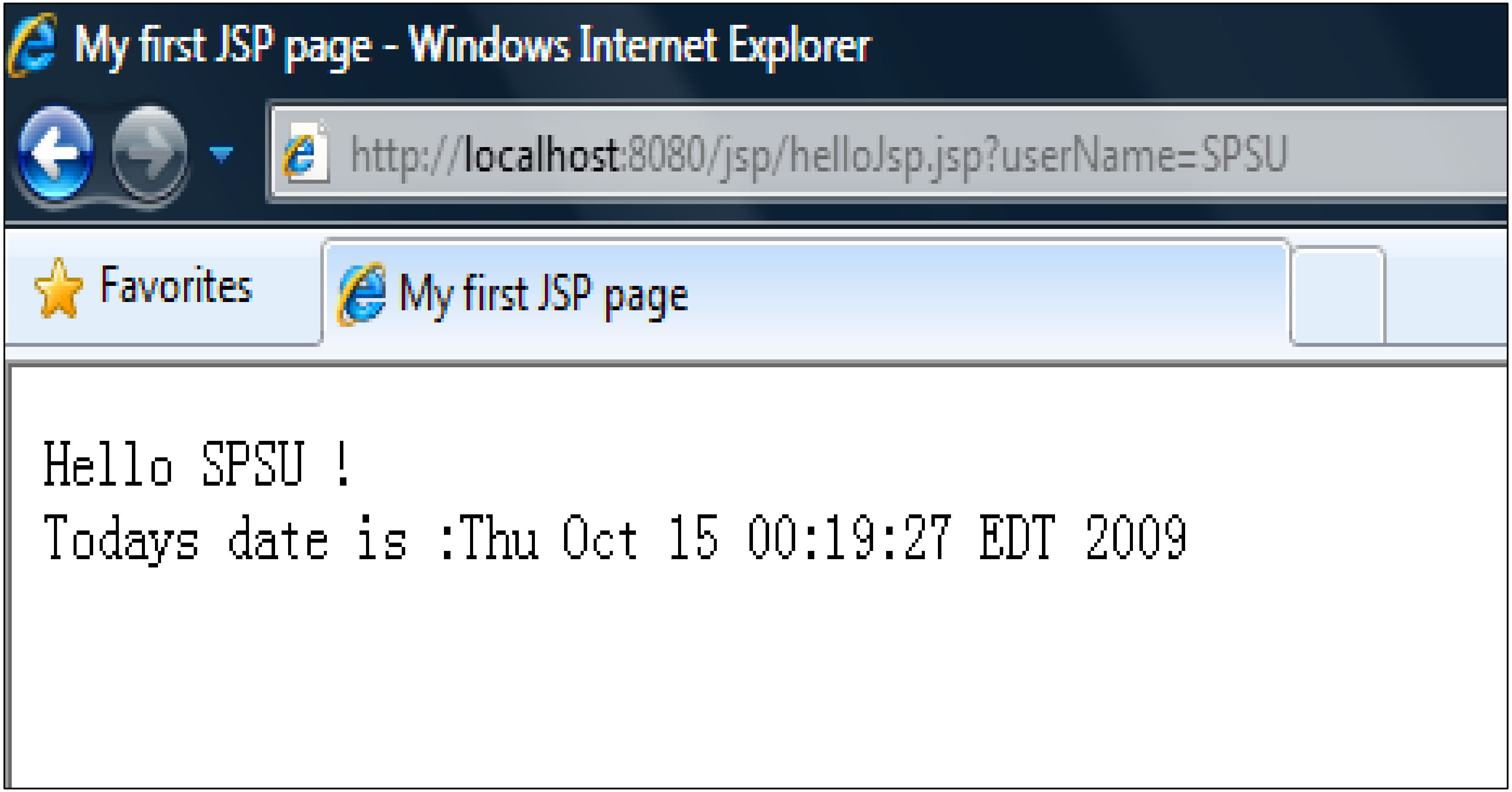
Demo1 - Windows Internet Explorer

http://localhost:8080/jsp/

Please enter the user name :

UserName : SPSU

First Simple Interactive JSP example (cont.)



My first JSP page - Windows Internet Explorer
http://localhost:8080/jsp/helloJsp.jsp?userName=SPSU
Favorites
My first JSP page

```
Hello SPSU !
Todays date is :Thu Oct 15 00:19:27 EDT 2009
```

Tomcat Server

- Apache Tomcat is an Apache module that provides a web server in addition to the Apache web server. The Tomcat web server supports Java Servlets and JavaServer pages.

- **Tomcat**, is an open-source web server and servlet container developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including Java Servlet,JavaServer Pages(JSP), Java EL, and WebSocket, and provides a "pureJava" HTTP web server environment for Java code to run in.

- Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under theApache License 2.0 license, and is open-source software.

Implicit JSP objects

- JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.
- JSP supports nine Implicit Objects which are listed below:

Declaring variables and methods

- The **JSP declaration tag** is used *to declare fields and methods*.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

- **Syntax of JSP declaration tag**
- The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

Error Handling and Debugging

- Using System.out.println():

System.out.println() is easy to use as a marker to test whether a certain piece of code is being executed or not. We can print out variable values as well.

- Using the JDB Logger:

The J2SE logging framework is designed to provide logging services for any class running in the JVM. So we can make use of this framework to log any information.

Data base action

Accessing a database from a JSP Page

Java Server Pages has Standard Tag Library which includes the number of actions for the database access to improve the simple database-driven Java Server Page applications. Basically these actions are used to provide the following features:

- Using a connection pool for better performance and scalability.
- The features are to support the queries, updates, and insertion process.
- To handle the most common data-type conversions.
- To Support a combination of databases.

Database Connectivity

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

Database Programming using JDBC

Required Steps

The following steps are required to create a new Database using JDBC application −

Import the packages: Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.** will suffice.

Register the JDBC driver: Requires that you initialize a driver so you can open a communications channel with the database.

Open a connection: Requires using the *DriverManager.getConnection()* method to create a Connection object, which represents a physical connection with the database server.

To create a new database, you need not give any database name while preparing database URL as mentioned in the below example.

Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.

Clean up the environment . Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

Example

```
import java.sql.*;      // Use classes in java.sql package
public class JdbcSelectTest {          // Save as "JdbcSelectTest.java"
public static void main(String[] args) {
        try (
             // Step 1: Allocate a database "Connection" object Connection
              conn = DriverManager.getConnection(
            "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx");
            // MySQL


            // Step 2: Allocate a "Statement" object in the Connection Statement
            stmt = conn.createStatement();
            ) {


          // Step 3: Execute a SQL SELECT query, the query result
         // is returned in a "ResultSet" object.
         String strSelect = "select title, price, qty from books";
         System.out.println("The SQL query is: " + strSelect);
       // Echo For debugging System.out.println();
        ResultSet rset = stmt.executeQuery(strSelect);
```

```
 // Step 4: Process the ResultSet by scrolling the cursor forward via next().
          // For each row, retrieve the contents of the cells with
getXxx(columnName).
            System.out.println("The records selected are:");
           int rowCount = 0;
           while(rset.next()) {
                      // Move the cursor to the next row
                       String title = rset.getString("title");
                      double price = rset.getDouble("price");
                        int qty = rset.getInt("qty");
                     System.out.println(title + ", " + price + ", " + qty); ++rowCount;
             }
            System.out.println("Total number of records = " + rowCount);
       }
        catch(SQLException ex) {
                ex.printStackTrace();
          }
     // Step 5: Close the resources - Done automatically by try-with-resources
      }
 }
```

Development of java beans in JSP

- A java bean is a simple java component which should satisfy the below mentioned points.
- A java bean should not have any public variables. All the variables should be accessed using the getter/setter methods.
- Java bean constructor should be a no argument constructor. To meet this requirement better leave the file without creating any constructor with arguments or create a no argument constructor explicitly.

Below code fragment is an example of a basic bean.

Listing 1: Course.java - Java Bean

```
package com.javaBeans;
public class Course {
          private String title;
          private String code;
          public String getTitle() {
                    return title;
           }
           public void setTitle(String title) {
                     this.title = title;
           }
            public String getCode() {
                     return code;
             }
            public void setCode(String code) {
                      this.code = code;
            }
 }
```

How to embed java beans in a JSP file?

- Java beans can be directly integrated in the jsp page which gives the user a flexibility to work with java reusable java components.

- Following JSP standard actions embed the Java bean in a JSP file.

<jsp:useBean>

<jsp:getProperty>

<jsp:setProperty>

- Load Java bean inside a JSP:

To start working with java beans inside a jsppage , first the bean should be loaded into the page. Once the bean is loaded , the variable properties of the bean can be accessed.

- Hence to load a bean the standard action is used. The basic syntax of the action is as follows:
- <jsp:useBean id=”course1” class=”com.Course” />

- Above syntax representation means that “instantiate an object of the class ‘Course’ , binding it to a variable name specified in the ‘id’ attribute”

- Writing above syntax in a JSP page creates an object referencing to the class “Course” and the name of the object is “course1”.

- <jsp:useBean> has some other attributes which provides additional benefits when creating a bean object.
- <jsp:useBean> “scope” attribute allows the bean object to be sharable across the application.

- Depending on the values of the scope attribute , if the bean is shared and has the same id and scope on the other page, the same bean object is associated to the other jsp page. The property values also persist the same when the same object is associated in different pages .

- If the bean is not sharable or the id and scope are different <jsp:useBean> instantiates a new object of the class

Working with bean properties

- After the bean gets loaded into the page, the properties can be accessed using the following standard actions.

<jsp:getProperty>

<jsp:setProperty>

<jsp:getProperty>

- This standard action accesses a property of the bean to get the value and put inside a jsp page.

The basic syntax of the <jsp:getProperty> is as follows:

<jsp:getProperty name=”course1” property=”title”/>

- The above syntax tells the compiler to get the value of the variable “title” of the object “course1”.

The attribute name in the above syntax represents the object created using the action. The value of the name attribute of the <jsp:getProperty> and the id attribute of the <jsp:useBean> property should be same to refer to the object created.

- The attribute property holds the name of the any variable of the bean loaded.

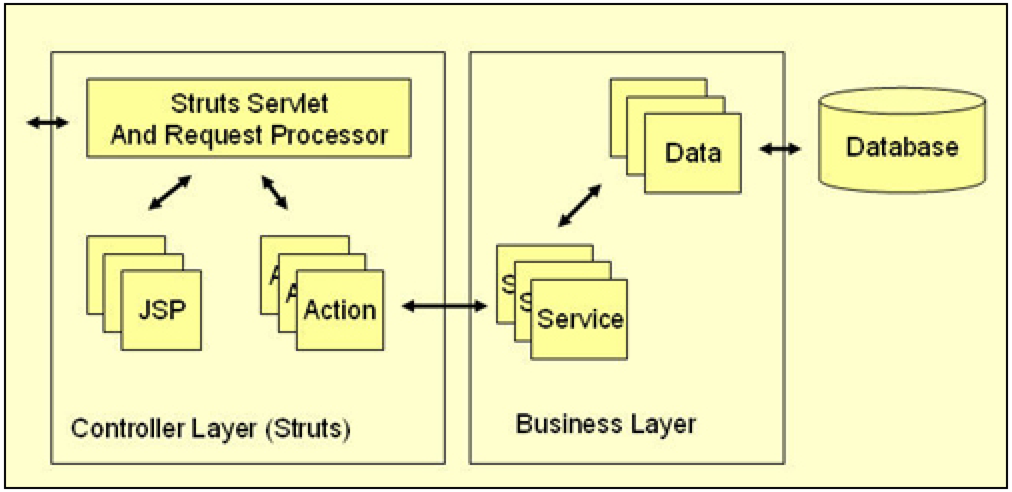
To get all the properties of the bean inside the jsp page the syntax of the <jsp:getProperty> should be

<jsp:getProperty name = “course1” property=”*” />

Introduction to Struts framework

- The Struts Framework is a standard for developing well-architected Web applications. It has the following features:
- Open source
- Based on the Model-View-Controller (MVC) design paradigm, distinctly separating all three levels:
  - **Model:** application state
  - **View:** presentation of data (JSP, HTML)
  - **Controller:** routing of the application flow
- Implements the JSP Model 2 Architecture
- Stores application routing information and request mapping in a single core file, struts-config.xml
- The Struts Framework, itself, only fills in the View and Controller layers. The Model layer is left to the developer.

Architecture Overview

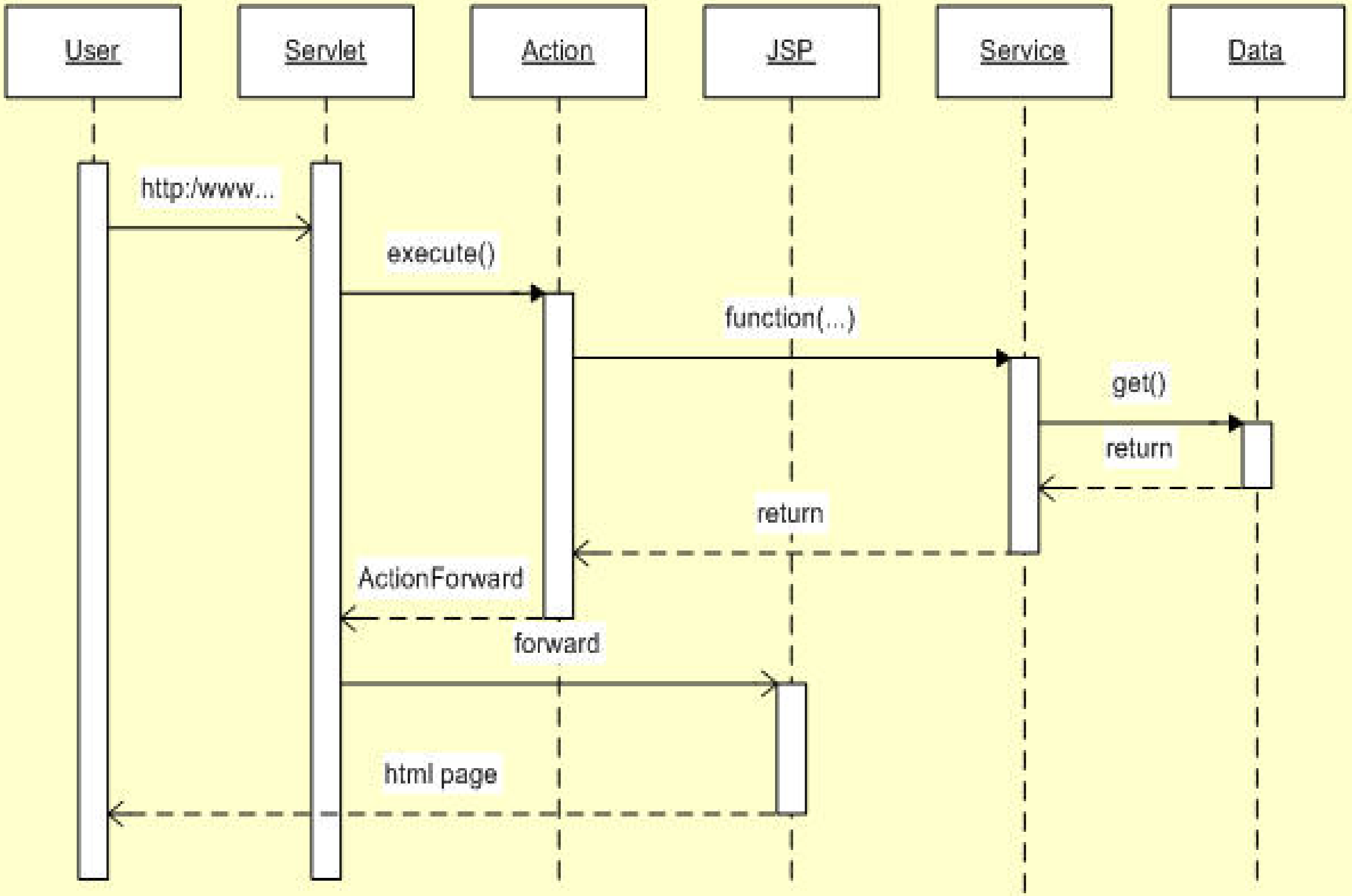


All incoming requests are intercepted by the Struts servlet controller. The Struts Configuration file struts-config.xml is used by the controller to determine the routing of the flow. This flows consists of an alternation between two transitions:

1.From view to action: A user clicks on a link or submits a form on an HTML or JSP page. The controller receives the request, looks up the mapping for this request, and forwards it to an action. The action in turn calls a Model layer (Business layer) service or function.

2. From action to view:After the call to an underlying function or service returns to the action class, the action forwards to a resource in the View layer and a page is displayed in a web browser.

Sequence diagram



- **User** clicks on a link in an HTML page.
- **Servlet** controller receives the request, looks up mapping information in struts-config.xml, and routes to an action.
- **Action** makes a call to a Model layer service.
- **Service** makes a call to the Data layer (database) and the requested data is returned.
- **Service** returns to the action.
- **Action** forwards to a View resource (JSP page)
- **Servlet** looks up the mapping for the requested resource and forwards to the appropriate JSP page.
- **JSP** file is invoked and sent to the browser as HTML.
- **User** is presented with a new HTML page in a web browser.

Struts Components

The Controller

This receives all incoming requests. Its primary function is the mapping of a request URI to an action class selecting the proper application module. It's provided by the framework.

The struts-config.xml File

This file contains all of the routing and configuration information for the Struts application. This XML file needs to be in the WEB-INF directory of the application.

Action Classes

It's the developer's responsibility to create these classes. They act as bridges between user-invoked URIs and business services. Actions process a request and return an ActionForward object that identifies the next component to invoke. They're part of the Controller layer, not the Model layer.

View Resources

View resources consist of Java Server Pages, HTML pages, JavaScript and Stylesheet files, Resource bundles, JavaBeans, and Struts JSP tags.

Struts Components

ActionForms

These greatly simplify user form validation by capturing user data from the HTTP request. They act as a "firewall" between forms (Web pages) and the application (actions). These components allow the validation of user input before proceeding to an Action. If the input is invalid, a page with an error can be displayed.

Model Components

The Struts Framework has no built-in support for the Model layer. Struts supports any model components:

JavaBeans

EJB

CORBA

JDO

any other