
ACADGILD
LEARN. DO. EARN
J2EE

ACADGILD

Java Server Page

Session - 21

LEARN. DO. EARN

Agenda

ACADGILD

Sl No	Agenda Title
1	Java Server Page (JSP)
2	Structure of Java Server Page (JSP)
3	Implicit Objects in JSP
4	JavaBeans with JSP

Java Server Page (JSP)

ACADGILD

- **JSP** stands for Java Server Pages
- **JSP** is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- So, JSP contains mixture of HTML & Java code.
- Java code handles all the business logic for a page whereas HTML manages display of the page.

Java Server Page (JSP) (Contd.)

ACADGILD

- Example code snippet for JSP:

```
<html>
<body>
<% out.print("Hello JSP!"); %>
</body>
</html>
```

- Code output on screen:

  Hello JSP!

Structure of Java Server Page (JSP)

ACADGILD

```
<html>
<body>
<form action="SayHello.jsp" method="get">
Name: <input type="text" name="name">
<input type="submit">
</form>
</body>
</html>
```

FirstPage.jsp

SayHello.jsp

- **FirstPage.jsp** displays a form & accepts name from user.
- On click of a button, it redirects user to **SayHello.jsp** in which code sends name of user.
- In **SayHello.jsp**, we are fetching name put by **FirstPage.jsp** & greeting user hello.
- This is how, JSPs communicates with each other to send and receive information.

Implicit Objects in JSP

ACADGILD

- **JSP container** offers developers a basic set of objects in each page.
- Developers can call these objects directly without declaring them explicitly.
- These objects are called as **JSP implicit objects**.
- **JSP Implicit Objects** are also called pre-defined variables.

Implicit Objects in JSP (Contd.)

ACADGILD

- In all, there are total 9 implicit objects in JSP. They are as follows:
  - request: Object associated with the request from client
  - response: Object associated with the response to the client
  - out: Object used to send output to the client
  - session: Object associated with the request
  - application: Object associated with application context
  - config: Object associated with the page
  - pageContext: This encapsulates use of server-specific features
  - page: This is used to call methods defined by the translated Servlet (same as this)
  - Exception: This object allows exception data to be accessed by designated JSP

JavaBeans with JSP

ACADGILD

- Specially constructed Java class, coded according to JavaBeans API specifications are called as **JavaBeans**.

- Typical **JavaBeans** has:
  - Default i.e. no-argument constructor.
  - Number of properties which can be read or written
  - Number of "getter" and "setter" methods for the properties
  - Implement Serializable interface & are Serializable

- In JSPs, we can use this JavaBeans by declaring useBean action.

- Once declared, the bean becomes a scripting variable & any scripting elements along with custom tags can use that bean in the JSP.

JavaBeans with JSP

ACADGILD

- Use of useBean tag can be done as follows:

  <jsp:useBean id = “name” class = “package.class” >

  For example, <jsp:useBean id = “date” class=“java.Util.Date” >

Agenda

ACADGILD

Sl No	Agenda Title
1	Database in JSP
2	Working with Expression Language
3	Structure of Java Server Page
4	Custom Tags in JSP
5	Custom Tags with Body
6	Custom Tags with Attribute

Database in JSP

ACADGILD

- **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.

- Applications get access to the database through an instance of the Java Database connectivity interface named **javax.sql.DataSource**.

- Assuming JDBC as pre requisite, let's see now how to connect to database which is already created in MySQL.

Database in JSP (Contd.)

ACADGILD

- Now, as we have already created database & table in MySQL manually, we will follow set of steps for connecting to that database & fetch back records from created table. We will assume following schema for database.

Table name = books

Id	Name	Author	Price
1	Let Us C	Yashwant K.	350
2	C++	Balguru S.	440

Database in JSP (Contd.)

ACADGILD

Step 1:

As we are going to use few classes in java.sql package, we will import that package in our JSP as:

```
<%@ page import="java.sql.*" %>
```

Step 2:

Load JDBC driver.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Step 3:

Try to establish connection.

```
Connection connection = DriverManager.getConnection("","","");
```

Here, 1st parameter is connection URL, 2nd is username & 3rd is password.

Make sure to use your own parameters.

Step 4:

Prepare a query to be fired. SQL statement is called as query in world of database.

```
PreparedStatement statement = connection.prepareStatement("select * from books");
```

Database in JSP (Contd.)

ACADGILD

Step 5:

- Execute query.
- As it's select query, it will return data from table books.
- **executeQuery()** method returns result in form of ResultSet.
- Collect it in variable so that we can iterate through it & read each record.

```
ResultSet resultOfQuery = statement.executeQuery();
```

Step 6:

Traverse the resultset & read every field of every record.

```
while(resultOfQuery .next())
{
        resultOfQuery .getString(1);
}
```

Loop will keep fetching string from resultOfQuery which is at 1st index.
i.e. name of book

Step 7:

Close the connection.

```
connection.close();
```

Database in JSP (Contd.)

ACADGILD

So, combining whole code again, we will get:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection connection =
DriverManager.getConnection("","","");
PreparedStatement statement =
connection.prepareStatement("select * from books");
ResultSet resultOfQuery = statement.executeQuery();
while(resultOfQuery.next())
{
        resultOfQuery.getString(1);
}
connection.close();
```

- Once we get fields of every record, we can use them accordingly.
- We can put it in table or can save it in file or pass it to another JSP for processing etc.

Working with Expression Language

ACADGILD

- **JSP Expression Language (EL)** makes it possible to easily access application data stored in JavaBeans components and other objects like request, session, application etc.

- **JSP EL** allows you to create arithmetic expressions as well as logical expressions.

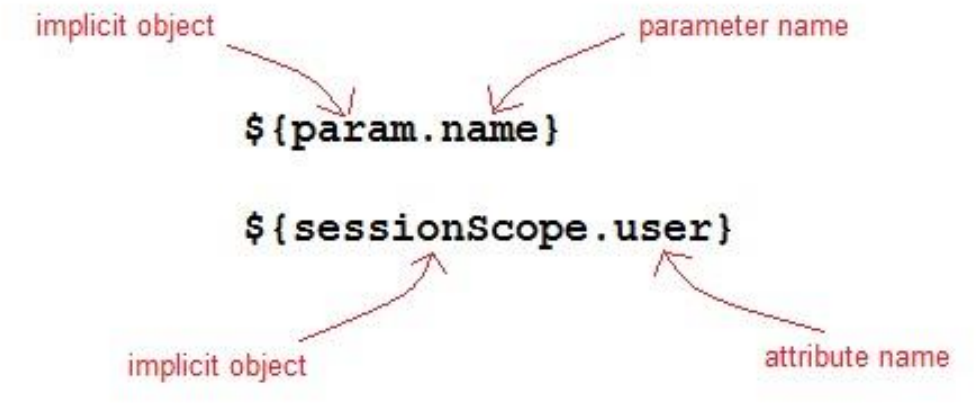
- The purpose of EL is to produce script-less JSP pages.

- Basic syntax for EL is:

  ${ expression }

Working with Expression Language (Contd.)

ACADGILD



- Within a JSP EL expression, you can use integers, floating point numbers, strings, the built-in constants true and false for boolean values, and null.
- There are many implicit objects, operators and reserve words in EL.

Structure of Java Server Page (JSP)

ACADGILD

FirstPage.jsp

```
<html>
<head>
<title>Welcome Page</title>
</head>
<body>
<h1>Welcome${param.name}</h1>
</body>
</html>
```

Hello.jsp

```
<form method="post" action=" Hello.jsp ">
Name <input type="text" name="user" >
<input type="submit" value="submit">
</form>
```

Custom Tags in JSP

ACADGILD

- A **custom tag** is a user-defined JSP language element..
- A tag defined by user according to his/her requirement is a **custom tag**.
- When a JSP page containing a custom tag is translated into a Servlet, the tag is converted to operations on an object called a **tag handler**. [**Tag handler** is a class which handles custom tags].
- The **Web container** then invokes those operations when the JSP page's Servlet is executed.
- **JSP tag extensions** let you create new tags that you can insert directly into a JSP
- To create a **custom JSP tag**, we must first create a Java class that acts as a tag handler.
- After that, we need to create our own **tag library file** in *WEB-INF* folder
- Let’s go ahead & create our own new tag now…

Custom Tags in JSP (Contd.)

ACADGILD

Step 1:

1. Create your own class & extend it by **SimpleTagSupport** class.
2. Now, override doTag() method in it & write following code snippet into it.

```
JspWriter out = getJspContext().getOut();
    out.println("Hello Custom Tag!");
```

Your class should look like,

```
public class MyOwnTag extends SimpleTagSupport
{
    public void doTag() throws JspException, IOException
    {
     JspWriter out = getJspContext().getOut();
     out.println("Custom tag getting executed!");
    }
}
```

Custom Tags in JSP (Contd.)

ACADGILD

Step 2:

1. Now, we need to create tag library file. (.tld Tag Library Descriptor file)
2. Navigate to **<Tomcat-Installation-Directory>webapps\ROOT\WEB-INF**
3. Create custom.tld file in that directory
4. Put following code inside it:

```
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>Sample TLD File</short-name>
<tag>
<name>Hello</name>
<tag-class>your.package.name.MyOwnTag</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```

Custom Tags in JSP (Contd.)

ACADGILD

Step 3:

1. We are ready to use our custom tag in JSP.
2. We will provide name of directory which hosts our tld file.
3. Create new JSP & write following snippet into it:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
    <head>
     <title>Custom Tag Demo</title>
    </head>
    <body>
     <ex:Hello/>
    </body>
</html>
```

Output = Custom tag getting executed!

Custom Tags with Body

ACADGILD

- If we need custom tags with attribute or body, we need to follow same set of instructions but with few changes in tag handler class.

- **Step 1:**

  Tag handler class will look like:

```
public class MyOwnTag  extends SimpleTagSupport
{
    StringWriter sw = new StringWriter();
    public void doTag() throws JspException, IOException
    {
     getJspBody().invoke(sw);
     getJspContext().getOut().println(sw.toString());
    }
}
```

Custom Tags with Body (Contd.)

ACADGILD

Step 2:

1. Accordingly, we will make changes in TLD file also.
2. Add following to our TLD file:

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>your.package. MyOwnTag </tag-class>
    <body-content>scriptless</body-content>
  </tag>
</taglib>
```

Custom Tags with Body (Contd.)

ACADGILD

Step 3: Create new JSP & write following snippet into it:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
 <head>
  <title>A sample custom tag</title>
 </head>
 <body>
  <ex:Hello>
     This is message body
  </ex:Hello>
 </body>
</html>
```

Custom Tags with Attribute

ACADGILD

Step 1:

```
public class MyOwnTag extends SimpleTagSupport {
    private String message;
    public void setMessage(String msg)
    {
                this.message = msg;
    }
    StringWriter sw = new StringWriter();
    public void doTag() throws JspException, IOException {
                if (message != null)
        {
                JspWriter out = getJspContext().getOut();
                        out.println( message );
                } else {
                getJspBody().invoke(sw);
                        getJspContext().getOut().println(sw.toString());
                }
    }
}
```

Custom Tags with Attribute (Contd.)

ACADGILD

Step 2:

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>your,package.MyOwnTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
      <name>message</name>
    </attribute>
  </tag>
</taglib>
```

Custom Tags with Attribute (Contd.)

ACADGILD

Step 3:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello message="This is custom tag" />
  </body>
</html>
```


Questions?

THANK YOU

For more details contact us at:

Support - +91 8884666874

Email us at - support@acadgild.com