# UNIT – V
# STRUTS

**Introduction to Struts:**

Definition:

Struts is a framework that promotes the use of Model-View-Controller(MVC) architecture for designing large-scale applications. The framework includes a set of custom tag libraries and their associated java classes along with various utility classes.

**Model-View-Controller Architecture:**

Definition:

MVC is a way to build applications that promotes complete separation between business logic and presentation logic. MVC represents three components:

- Model
- View
- Controller

**View:**

The View is the user interface i.e., the screens that the end-user of an application actually sees and interacts with. Examples are HTML,JSP's etc..

**Controller:**

When the user enters some data in the view and clicks on submit button the request is sent to the Controller. The controllers job is to take the data entered by the user and pass it to the Model. A JavaBean is a controller in a J2EE application.

**Model:**

A Model is a separate Java class that contains the actual business logic. The model executes the business logic and returns some result back to the controller. The controller takes this and presents the user with a new view. A servlet is an example of a Model.

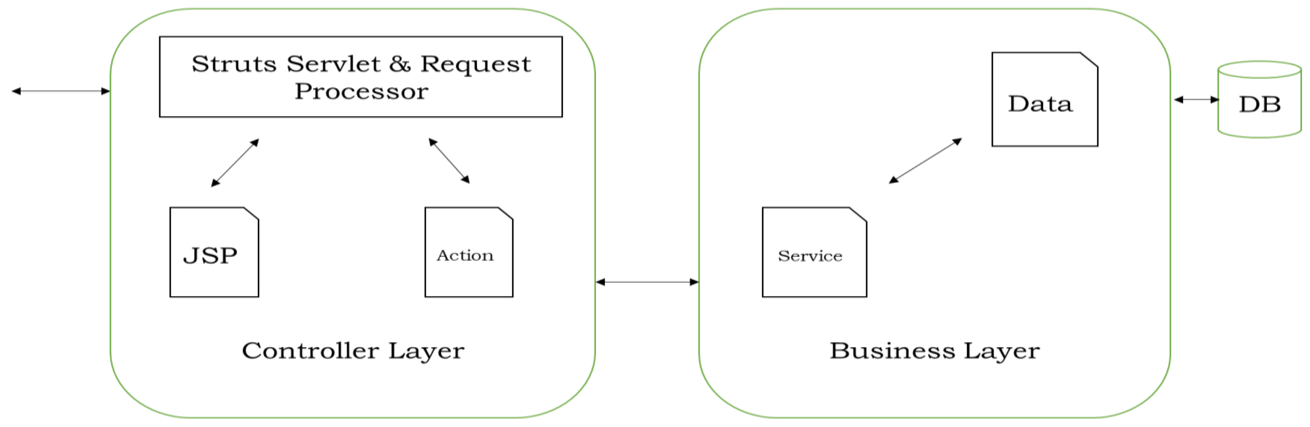
**What is Struts Framework?**

The struts framework is a standard for developing well-architectured web applications. It has the following features:

1. It is an Open source

2. It is based on MVC
3. It stores application routing information and request mapping information in a single core file called struts-config.xml.

**Struts Architecture:**



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

1. From View-to-Action.
2. From Action-to-View.

**Struts Components:**

Various Struts components are:

**Controller:**

This receives all incoming requests. Its primary function is the mapping of a request URL to an action class.

**Struts-config.xml:**

This file contains all of the routing and configuration information for the struts application.

**Action classes:**

Action classes act as a bridge between user-invoked URI's and business services.

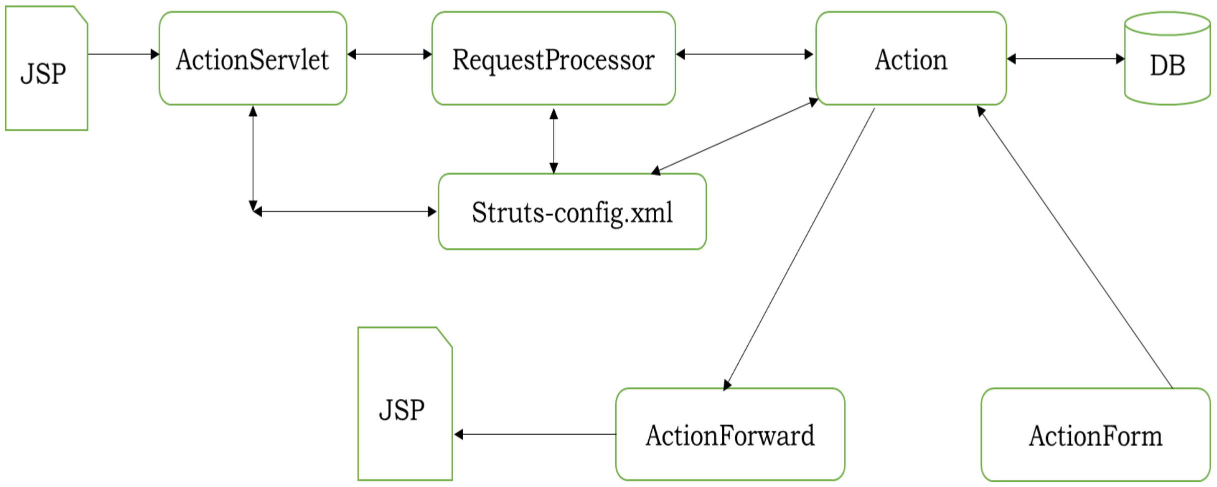
**View resources:**

View resources consists of JSP's, HTML, Java script and stylesheet files etc..

**ActionForms:**

ActionForms greatly simplify user form validation by capturing user data from the HTTP request.

**Struts Request Lifecycle:**



There are several struts controller classes. They are:

1. ActionServlet
2. RequestProcessor
3. ActionMapping
4. Action
5. ActionForm
6. ActionForward

All the above classes reside in org.apache.struts.action package.

**ActionServlet:**

ActionServlet is the concrete class and extends the javax.servlet.http.HttpServlet. It performs two important things:

1. On start-up it reads the struts configuration file and loads it into memory.
2. It then intercepts HTTP request and handles it appropriately.

In order to load the struts-config.xml into memory it's information from web.xml file must be retrieved into ActionServlet using init() method. The listing of web.xml can be as shown below:

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
                <param-name>config</param-name>
                <param-value>/WEB-INF/config/struts-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
```

**<load-on-startup>:**

When the user request is received for the first time, then to read struts-config.xml from deployment descriptor through init() method and loading into memory is time consuming process. In order to avoid this delay we specify <load-on-startup> as 1 that instructs the servlet container to retrieve struts-config file directly into memory.

The second task the ActionServlet performs is to intercept HTTP request based on the URL pattern and handle appropriately. The URL pattern can either be a path or suffix. This is specified in web.xml as shown below:

```
<servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

**RequestProcessor:**

ActionServlet intercepts the HTTP request and it delegates it to another class called RequestProcessor by invoking its process() method. RequestProcessor first retrieves the appropriate XML block from struts-config.xml to map the request to corresponding Action class. This process is done through ActionMapping.

**ActionMapping:**

ActionMapping is the class that holds the mapping between a URL and Action. A sample ActionMapping from the struts-config.xml file looks as follows:

```
<action path="/login" type="LoginAction">
```

**ActionForm:**

ActionForm is the class that reads data from user form and provides it to Action class.

**Action:**

The Action class provides an entry point where we can start with our application code to process the request.

RequestProcessor instantiates the Action class specified in the ActionMapping and invokes the execute() method. The signature of execute() method is as follows:

public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest req,HttpServletResponse res)throws Exception

**ActionForward:**

The execute() method returns the next view shown to the end user. ActionForward is the class that encapsulates the next view information. The findForward() method on the ActionMapping instance retrieves the ActionForward as follows:

ActionForward forward=mapping.findForward(“success”);

“success” is the logical name that is passed as the keyword in findForward().

The findForward() method searches for the logical name success and retrieves the corresponding view to be displayed to the user.

**Struts Configuration file:**

```
<struts-config>
        <data-sources>
        </data-sources>
        <form-beans>
                <form-bean name=“bean1” type=“com.example.FormBeanOne”/>
                 <form-bean name=“bean2” type=“com.example.FormBeanTwo”/>
        </form-beans>
        <global-exceptions>
                <exception type=“com.example.SomeException”
                path=“/WEB-INF/jsp/error.jsp”>
        </global-exceptions>
        <global-forwards>
```

```
            <forward name="login" path="/WEB-INF/jsp/login.jsp">
            <forward name="home" path="/WEB-INF/jsp/home.jsp">
      </global-forwards>
<action-mappings>
      <action path="/login" type="com.example.action.LoginAction" name="bean1"
             input="/WEB-INF/jsp/login.jsp">
            <forward name="ok" path="/WEB-INF/jsp/home.jsp"/>
            <forward name="fail" path="/WEB-INF/jsp/login.jsp"/>
      </action>
</action-mappings>
<controller ProcessorClass="org.apache.struts.files.RequestProcessor"/>
</struts-config>
```

**Hello World Example:**

**HelloworldForm.java:**

```
package com.example.form;
import org.apache.struts.action.ActionForm;
public class HelloWorldForm extends ActionForm {
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

**HelloWorldAction.java:**

```
package com.example.action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import com.example.form.HelloWorldForm;
public class HelloWorldAction extends Action {
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
    HelloWorldForm hwForm = (HelloWorldForm) form;
    hwForm.setMessage("Hello World");
    return mapping.findForward("success");
    }
}
```

**struts-config.xml:**

```
<struts-config>
    <form-beans>
            <form-bean name="helloWorldForm" type="com.example.form.HelloWorldForm"/>
    </form-beans>
   <global-forwards>
        <forward name="helloWorld" path="/helloWorld.do"/>
    </global-forwards>
    <action-mappings>
        <action path="/helloWorld“
                   type="com.example.action.HelloWorldAction" name="helloWorldForm">
            <forward name="success" path="/helloWorld.jsp" />
        </action>
    </action-mappings>
</struts-config>
```

**web.xml:**

```
<web-app>
<servlet>
```

```
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
 </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

**index.jsp:**

```
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<logic:redirect forward="helloWorld"/>
```

**helloWorld.jsp:**

```
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
<bean:write name="helloWorldForm" property="message"/>
</body>
</html>
```

**Form Validation in struts:**

We can define our own validation logic in struts by implementing the Validateable interface in the action class. The workflow interceptor is used to get information about the error messages defined in the action class.

**Workflow Interceptor:**

The workflow interceptor checks if there is any validation error. It doesn't perform any validation. It is applied when action class implements the Validateable interface. The input is the default parameter for this interceptor that determines the result to be invoked for the action or field error.

**Validateabale interface:**

The Validateable interface must be implemented to perform validation logic in the action class. It contains only one method validate() that must be overridden in the action class to define the validation logic. Signature of the validate method is:

```
public void validate();
```

ActionSupport class implements Validateable and interface, so we can inherit the ActionSupport class to define the validation logic and error messages.

**Steps to perform validation:**

The steps are as follows:

1. create the form to get input from the user
2. Define the validation logic in action class by extending the ActionSupport class and overriding the validate method
3. Define result for the error message by the name input in struts.xml file

**Example to perform validation:**

In this example, we are creating 4 pages :

1. index.jsp for input from the user.
2. RegisterAction.java for defining the validation logic.
3. struts.xml for defining the result and action.
4. welcome.jsp for the view component.

**Index.jsp:**

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:form action="register">
```

```
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:submit value="register"></s:submit>
</s:form>
```

**RegisterAction.java:**

```
package com.example;
public class RegisterAction extends ActionSupport{
private String name,password;
public void validate() {
   if(name.length()<1)
      addFieldError("name","Name can't be blank");
   if(password.length()<6)
      addFieldError("password","Password must be greater than 5");
}
public String execute(){
   return "success";
}
}
```

**struts.xml:**

```
<struts>
<action name="register" class="com.example.RegisterAction">
<result>welcome.jsp</result>
<result name="input">index.jsp</result>
</action>
</struts>
```

**welcome.jsp:**

```
<%@ taglib uri="/struts-tags" prefix="s" %>
Name:<s:property value="name"/><br/>
Password:<s:property value="password"/><br/>
```