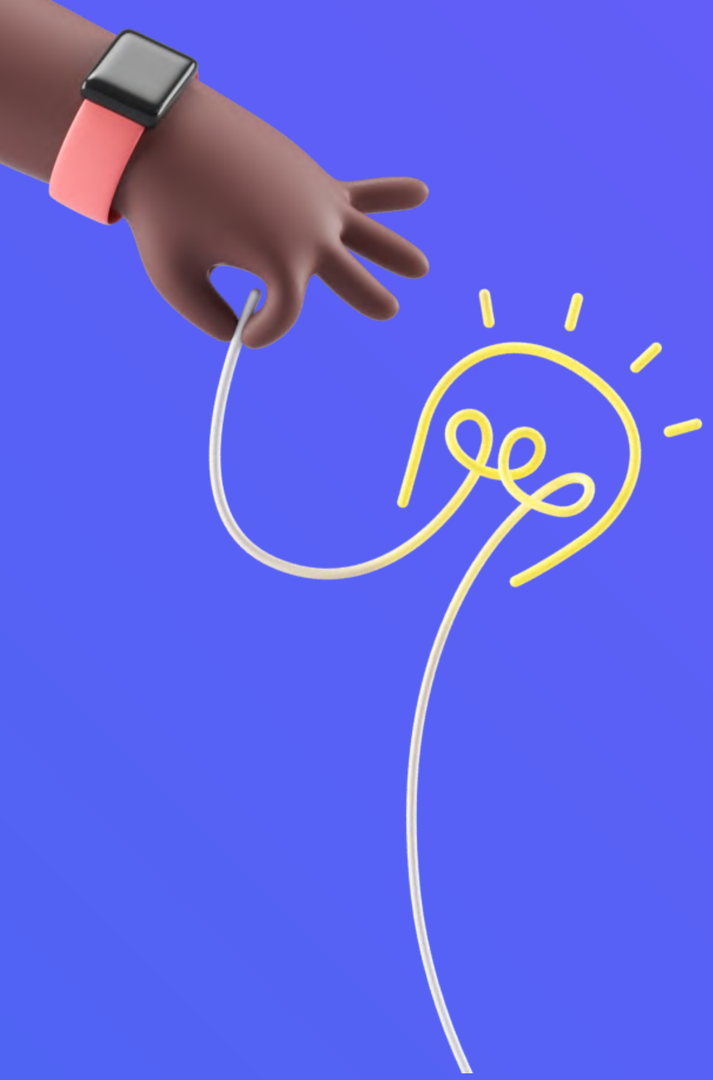
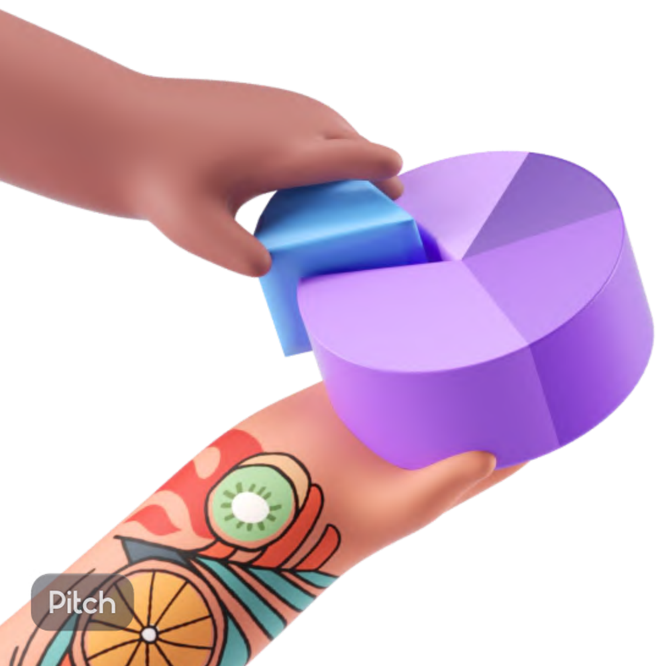


Apache Struts 2.x

Group 6

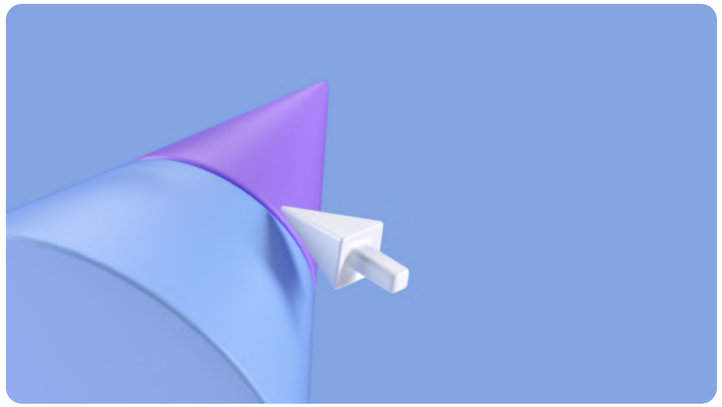
A little revision about Struts



Re-introduce

Struts is an open source framework used for developing **J2EE** web applications using **Model View Controller (MVC)** design pattern. It uses and extends the **Java Servlet API** to encourage developers to adopt an MVC architecture

Three key components

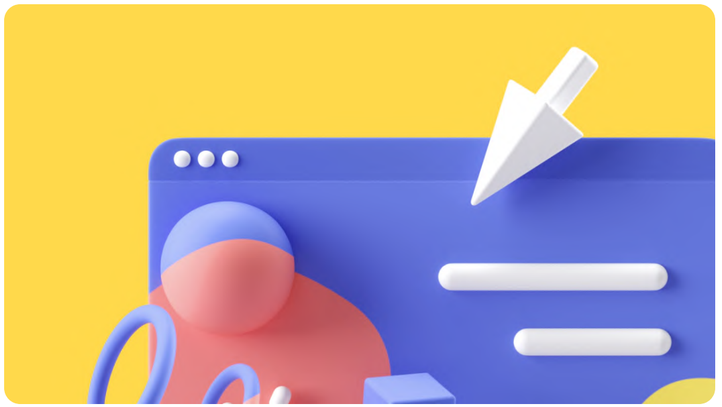


Request handler

Provided by the application developer that is mapped to a standard URI

Response handler

Transfers control to another resource which completes the response



Tag library

Helps developers create interactive form-based applications with server pages

What is struts 2.x

- Released in 2014

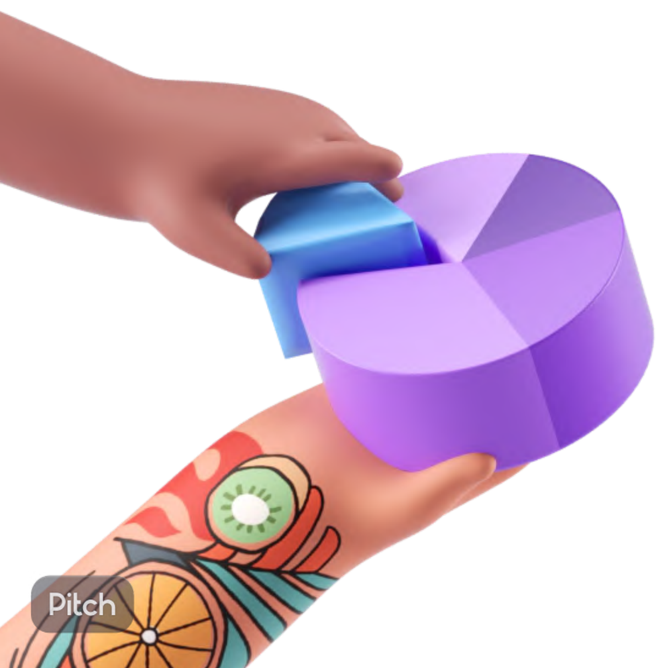
- It uses and extends the Java Servlet API to encourage developers to adopt a model–view–controller (MVC) architecture

- supports multiple view options like JSP, Freemarker, Velocity, XSLT.

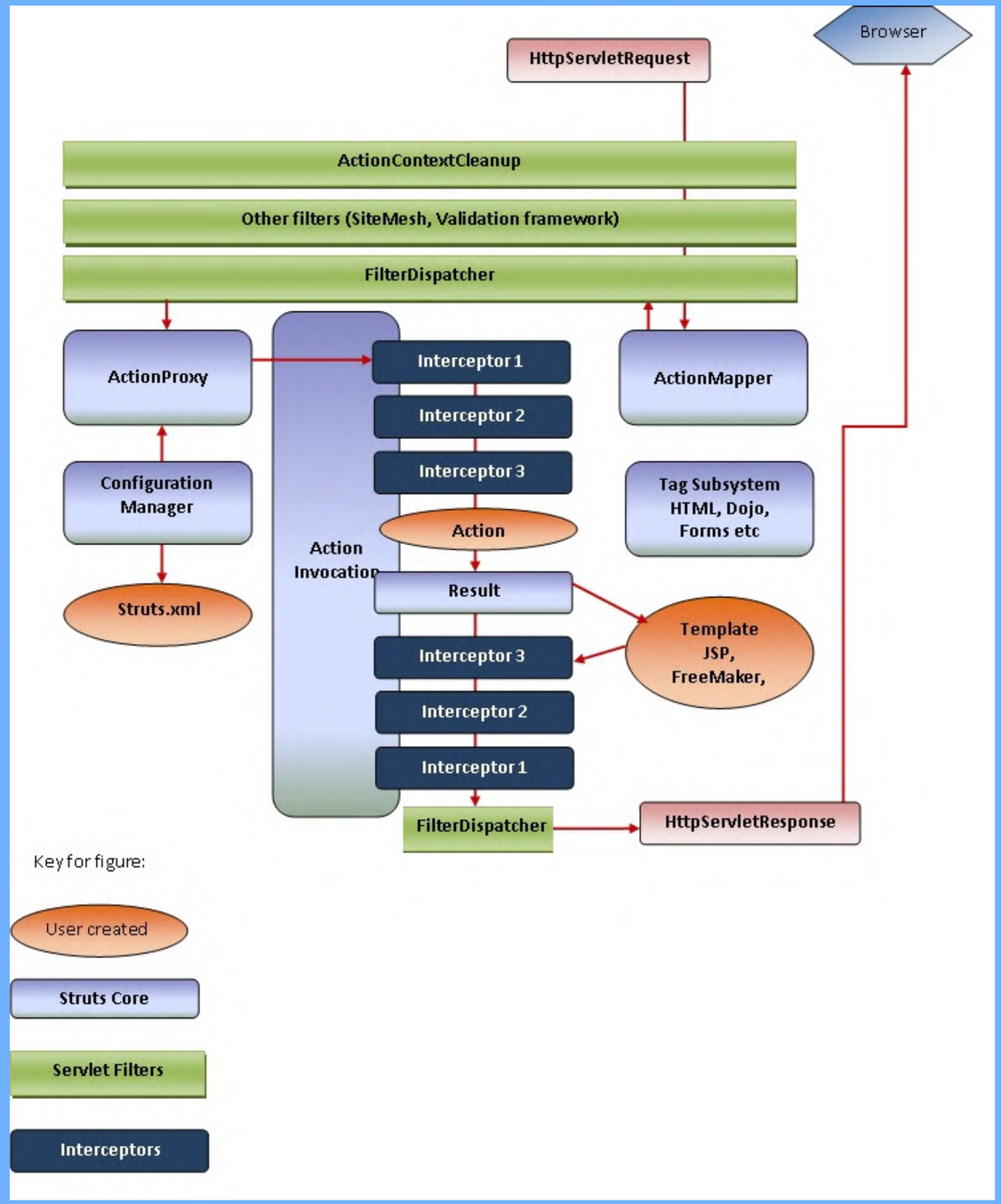
- Integration with other frameworks such as Spring, Hibernate.

- Struts2 is based on WebWorks2 framework

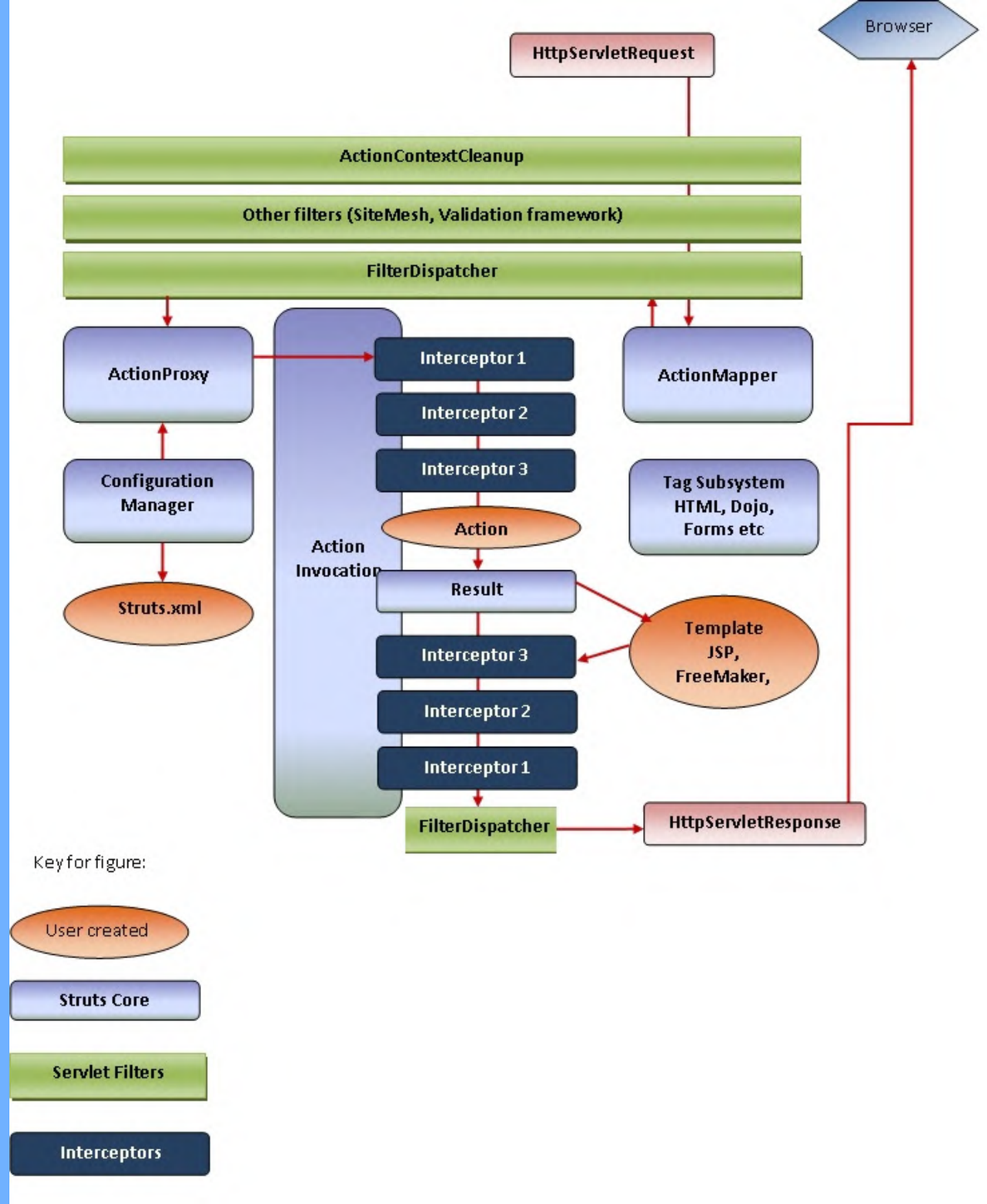
Struts 2 Architecture



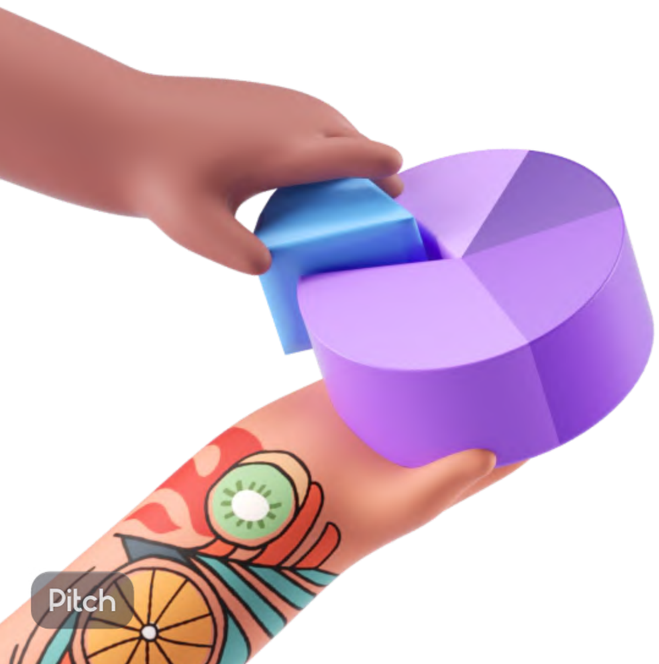
1. Interceptor: responsible for workflow implementation, avoids double submission problem, logging, validation, etc.
2. Action: provides the processing logic which is necessary to service the request from the user.
3. ActionProxy: uses ConfigurationFile manager and creates an instance of ActionInvocation
4. ActionInvocation: instantiate Action class. ActionInvocation will take help of struts.xml file to determine which method of the Action class should be called.
5. ActionMapper: checking whether incoming request is mapped to any action or interceptor or nothing.
6. Configuration Manager: The java object representation for the struts.xml file, it contains all the information regarding framework configuration.
7. Controller: inspecting each incoming request to determine which action should handle the request.
8. Result: the requested data which was created after successful execution action class.



- The request is sent from client's browser. This results invoke the servlet container which in turn is passed through standard filter chain, including: ActionContextCleanUp, SiteMesh and FilterDispatcher.
- The FilterDispatcher filter is called which consults the ActionMapper to determine whether an Action should be invoked.
- If ActionMapper finds an Action needs to be invoked, the FilterDispatcher delegates control to ActionProxy.
- ActionProxy reads the configuration file. ActionProxy creates an instance of ActionInvocation class and delegates the control.
- ActionInvocation is responsible for command pattern implementation. It invokes the Interceptors one by one and then invoke the Action.
- Once the Action returns, the ActionInvocation is responsible for looking up the proper result associated with the Action result code mapped in struts.xml.
- The Interceptors are executed again in reverse order and the response is returned to the FilterDispatcher. The result is then sent to the servlet container which in turns sends it back to client.



Struts 2 Controller



Struts 2 Controller

- The controller in struts 2 framework is FilterDispatcher. It's the first component to act in the request processing cycle
- Initializing Action class and Execute specified method according to configuration file or annotation
- Read the returned string value after execution to determine the result or view for the user
- Look for the execute() method or the specified method in the configuration
- Generate the result or view based on the String result returned by the execute function or the specified method
- Mapping in XML file:

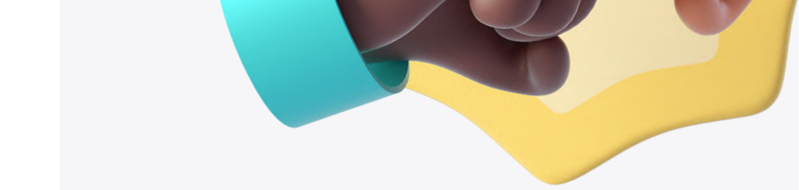
```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</servlet-class>
</filter>
<filter-mapping>
    <filter-name>struts2</servlet-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Struts 2 Action

What this section is all about

Struts 2 Action

- **Actions** are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework. Each URL is mapped to a specific action, which provides the processing logic which is necessary to service the request from the user.
- But the action also serves in two other important capacities. Firstly, the action plays an important role in the transfer of data from the request through to the view, whether its a JSP or other type of result. Secondly, the action must assist the framework in determining which result should render the view that will be returned in the response to the request.
- The only requirement for actions in Struts2 is that there must be one non-argument method that returns either a String or Result object and must be a POJO (Plain Old Java Object). If the non-argument method is not specified, the default behavior is to use the execute() method.



Basic Action Class

- Generally, execute method should be specified that represents the business logic.

```
public class Welcome {
public String execute(){
    return "success";
}
}
```

Action Interface

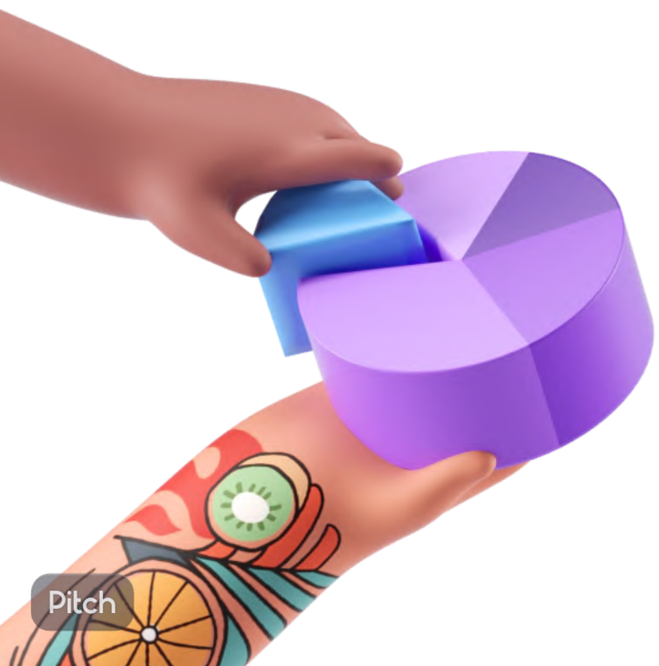
- Action interface provides 5 constants that can be returned form the action class. They are:

```
public static final String SUCCESS = "success";
public static final String ERROR = "error";
public static final String LOGIN  = "login";
public static final String INPUT = "input";
public static final String NONE = "none";
```

- Action interface contains only one method execute that should be implemented overridden by the action class even if you are not forced.
- If we implement the Action interface, we can directly use the constants instead of values.

```
import com.opensymphony.xwork2.Action;
public class Welcome implements Action{
public String execute(){
    return SUCCESS;
}
}
```

Struts 2 ActionForm



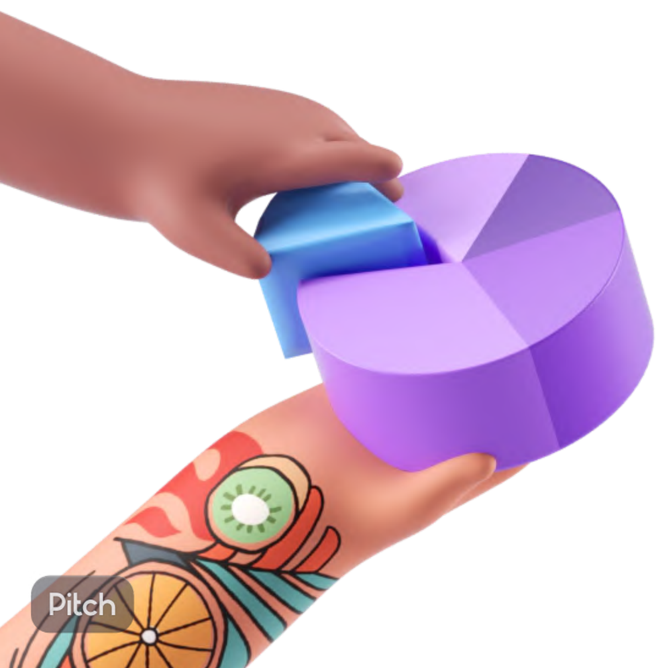
Strut 2 ActionForm

- Similar to Struts 1.x, Action Form in struts 2.x maintains the session state for web application and the Action Form object is automatically populated on the server side with data entered from a form on the client side.
- In Struts 2, to collect the information we'll use a Struts 2 form. When creating this form the key concept we need to employ is to tie each form field to a specific instance field of an object.
- The Struts 2 form will submit to an action. We'll need to define that action in our struts.xml file.

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Register</title>
  </head>
  <body>
    <h3>Register for a prize by completing this form.</h3>

    <s:form action="register">
      <s:textfield name="personBean.firstName" label="First name" />
      <s:textfield name="personBean.lastName" label="Last name" />
      <s:textfield name="personBean.email"  label ="Email"/>
      <s:textfield name="personBean.age"  label="Age"  />
      <s:submit/>
    </s:form>
  </body>
</html>
```

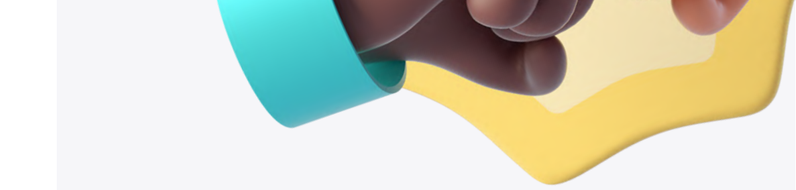
```
<action name="register" class="org.apache.struts.register.action.Register" method="execute">
    <result name="success">/thankyou.jsp</result>
</action>
```

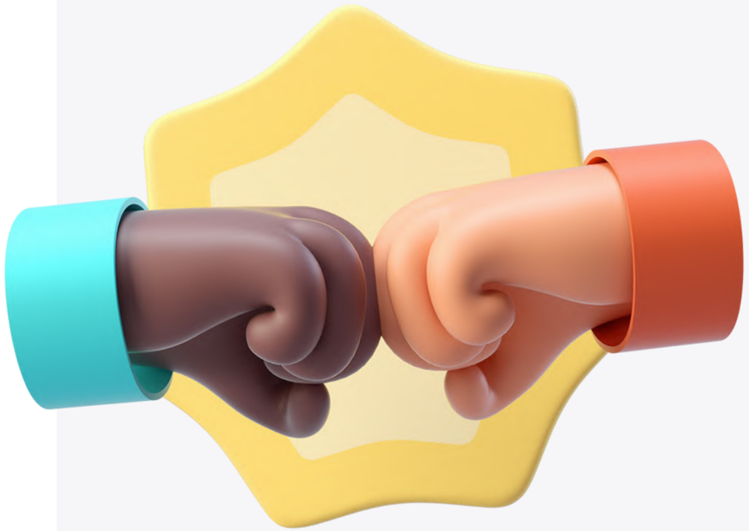
Struts 2 Tiles



Struts 2 Tiles

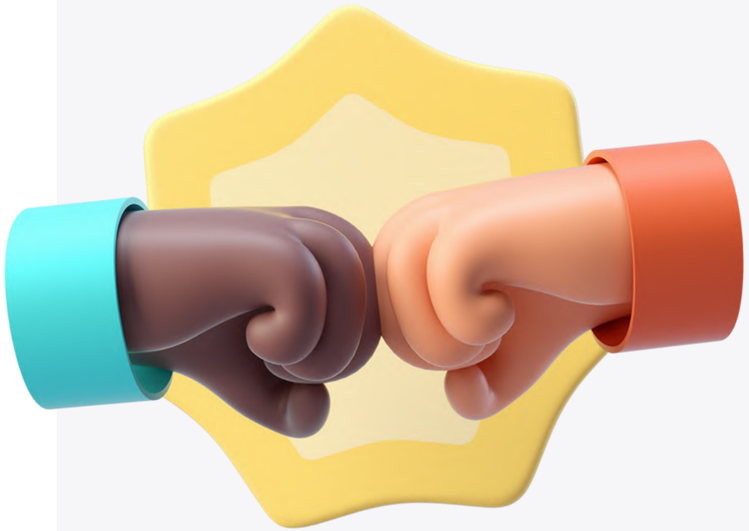
- Tiles Framework is a template composition framework. Tiles was originally built to simplify the development of web application user interfaces, but it is no longer restricted to the JavaEE web environment.
- Tiles allows authors to define page fragments which can be assembled into a complete page at runtime. These fragments, or tiles, can be used as simple includes in order to reduce the duplication of common page elements or embedded within other tiles to develop a series of reusable templates. These templates streamline the development of a consistent look and feel across an entire application.
- We can customize the layout of the struts 2 application by integrating with tiles framework. In tiles framework, we manage all the tile by our Layout Manager page.





Advantages of using Struts 2 Tiles

- Customization by centralized page We can customize the layout of all the pages by single page (centralized page) only.
- Code reusability A single part e.g. header or footer can be used in many pages. So it saves coding.
- Easy to modify If any part (tile) is modified, we don't need to change many pages.
- Easy to remove If any part (tile) of the page is removed, we don't need to remove the code from all the pages. We can remove the tile from our layout manager page.



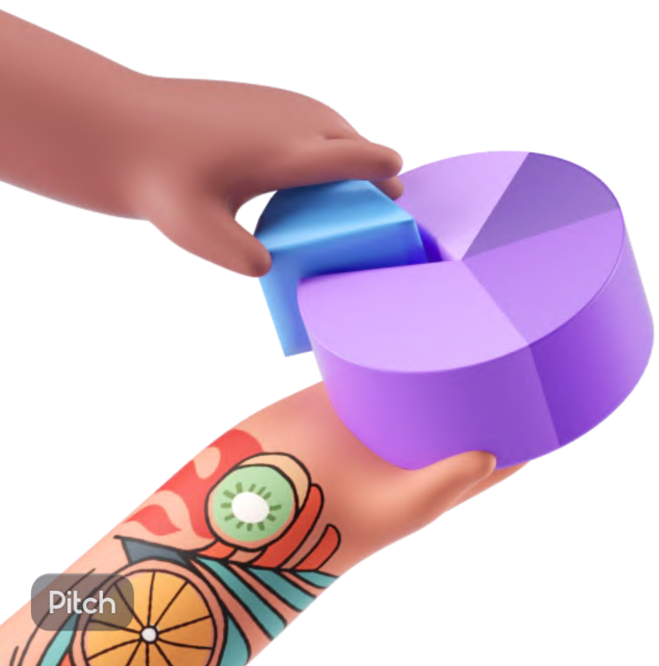
Struts 2 Tiles

- To enable tiles support within a Struts2 application:

1. Include the struts-tiles-plugin as a dependency in your web application.
2. Register the tiles listener.
3. All package definitions which require tiles support must either extend the tiles-default package or must register the [Tiles Result] type definition.
4. Configure your actions to utilize a tiles definition:
5. Instead of xml configuration you can use annotations:
6. You have to define Tiles Definitons in a tiles.xml file.

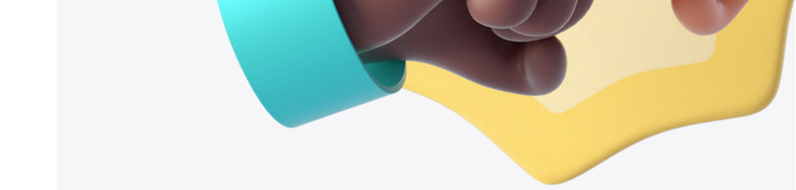
```
<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
        "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">

<tiles-definitions>

    <definition name="fooLayout" template="/WEB-INF/tiles/layout.jsp">
        <put-attribute name="title" value="Tiles Sample"/>
        <put-attribute name="header" value=".header"/>
        <put-attribute name="body" value=".bodyp"/>
    </definition>

    <definition name="tilesWorks" extends="fooLayout">
        <put-attribute name="header" value="/WEB-INF/tiles/header.jsp"/>
        <put-attribute name="body" value="/WEB-INF/tiles/body.jsp"/>
    </definition>

</tiles-definitions>
```

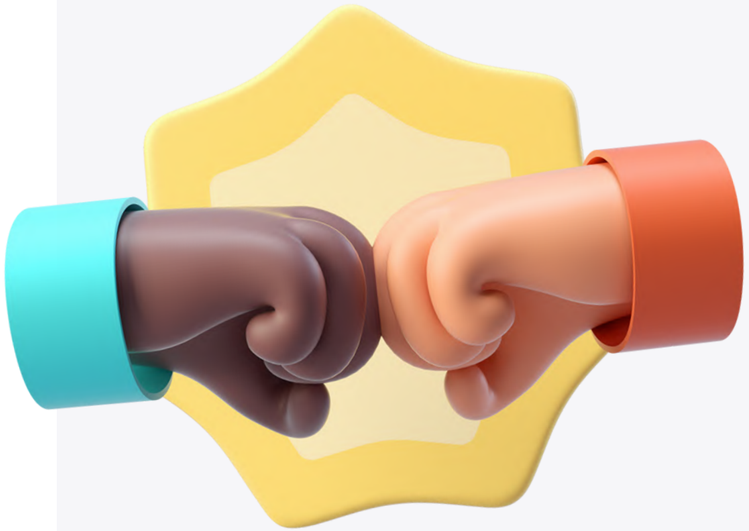
Struts 2 Validator Framework



Validation Framework

- Struts Action 2 relies on a validation framework provided by XWork to enable the application of input validation rules to your Actions before they are executed. Struts2 Validation Framework allows us to separate the validation logic from actual Java/JSP code, where it can be reviewed and easily modified later.
- The Struts2 Validation Framework alleviates much of the headache associated with handling data validation, allowing you to focus on validation code and not on the mechanics of capturing data and redisplaying incomplete or invalid data.
- Validation framework comes with set of useful routines to handle form validation automatically and it can handle both server side as well as client side form validation. If certain validation is not present, you can create your own validation logic by implementing java interface com.opensymphony.xwork2.Validator and plug it into validation framework as a re-usable component.





Define validation logic

- In order to define validation logic for particular form, we first have to create an XML file which will hold this data. Struts2 define a specific naming convention in defining validation xml files. The format is <ActionClassName>-validation.xml.

```
<%@ page contentType="text/html; charset=UTF-8"%><%@ taglib prefix="s"
uri="/struts-tags"%>
<html>
    <head>
        <title>Customer Form - Struts2 Demo | ViralPatel.net</title>
    </head>
    <body>
        <h2>Customer Form</h2>
        <s:form action="customer.action" method="post">
            <s:textfield name="name" key="name" size="20" />
            <s:textfield name="age" key="age" size="20" />
            <s:textfield name="email" key="email" size="20" />
            <s:textfield name="telephone" key="telephone" size="20" />
            <s:submit method="addCustomer" key="label.add.customer"
            align="center" />
        </s:form></body>
</html>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC  "-//OpenSymphony Group//XWork Validator 1.0.2/
/EN"  "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="name">
        <field-validator type="requiredstring">
            <param name="trim">true</param><message key="errors.required" /
            >
        </field-validator>
    </field>
    <field name="age">
        <field-validator type="required">
            <message key="errors.required" />
        </field-validator>
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">100</param>
            <message key="errors.range"/>
        </field-validator>
    </field>
    <field name="email">
        <field-validator type="requiredstring">
            <message key="errors.required" />
        </field-validator>
        <field-validator type="email">
            <message key="errors.invalid" />
        </field-validator>
    </field>
    <field name="telephone">
        <field-validator type="requiredstring">
            <message key="errors.required" />
        </field-validator>
    </field>
</validators>
```

Demo