11.1 INTRODUCTION TO SPRING FRAMEWORK

Spring Framework is an open source application framework for Java platform that provides comprehensive infrastructure support for developing Enterprise level Java applications. The framework's first release was written by Rod Johnson under the Apache 2.0 license in June 2003. It is a light-weight framework for the development of enterprise-based applications due to the POJO (Plain Old Java Object) model. Declarative transaction management, remote access to the logic using RMI or web services, mailing facilities and various ways for data persistence in a database can be configured using Spring. This framework can also be used in modular fashion meaning it allows using in parts and leave the other components which is not required by the application. For example, spring framework can be used for all layer implementations for a real time application or it can be used for a particular layer of a real time application development. Apart from struts which is only related to front end and hibernate which is only related to database, Spring enables to develop application related to all layers.

The infrastructure is handled by Spring so that the application logic can be focused. An application from "plain old Java objects" (POJOs) and applying enterprise services to POJOs can be built using Spring. This ability of the framework is applied to the Java SE programming model and to full and partial Java EE also.

As an application developer, the following advantages are provided by the Spring platform:

- → Enables a Java method to execute in a database transaction without having to deal with transaction APIs.
- → Enables a Java method to be a remote procedure without having to deal with remote APIs.
- → Enables a local Java method to have a management operation without having to deal with JMX (Java Management Extensions) APIs.
- → Enables a local Java method to contain a message handler without having to deal with JMS (Java Message Service) APIs.

Spring has gained popularity as it is simple due to its usage of the POJO/POJI model, for testing a spring application there is no need for a server or a container like that for an EJB or a struts application and Spring objects follow loose Coupling by using dependency injection mechanism with the help of POJO/POJI model.

→ **Modules :**

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.

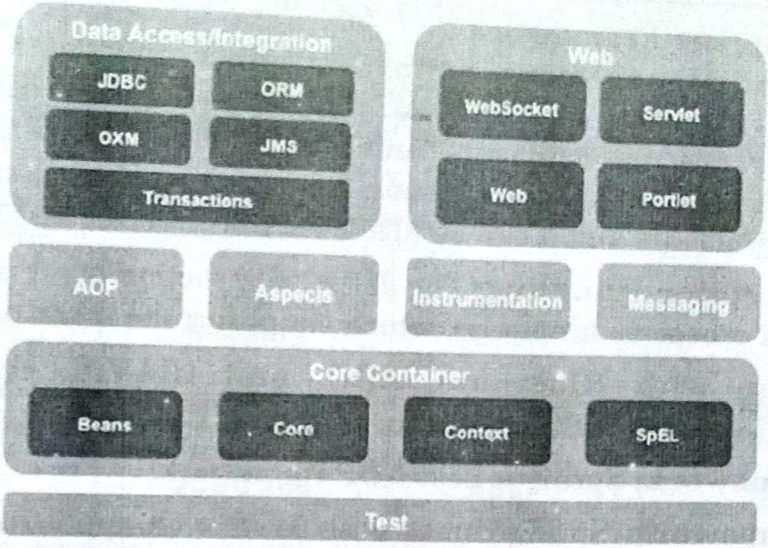

FIGURE 10.1 : OVERVIEW OF THE SPRING FRAMEWORK

The Spring Framework provides a range of services using the following modules:

- **Aspect-oriented programming:** enables implementing cross-cutting concerns.
- **Inversion of control container:** configuration of application components and lifecycle management of Java objects, done mainly via dependency injection
- **Model–view–controller:** an HTTP- and servlet-based framework providing hooks for extension and customization for web applications and RESTful Web services.
- **Authentication and authorization:** configurable security processes that support a range of standards, protocols, tools and practices via the Spring Security sub-project.
- **Convention over configuration:** Spring Roo module is a rapid application development solution for Spring-based enterprise applications.
- **Data access:** working with relational database management systems on the Java platform using JDBC and object-relational mapping tools and with NoSQL databases
- **Messaging:** configurative registration of message listener objects for transparent message-consumption from message queues via JMS, improvement of message sending over standard JMS APIs
- **Remote access framework:** configurative RPC-style marshalling of Java objects over networks supporting RMI, CORBA and HTTP-based protocols including Web services (SOAP)
- **Transaction management:** unifies several transaction management APIs and coordinates transactions for Java objects
- **Remote management:** configurative exposure and management of Java objects for local or remote configuration via JMX
- **Testing:** support classes for writing unit tests and integration tests

[1] Features of Spring Framework :

- **Transaction Management:** Spring framework provides a generic abstraction layer for transaction management. This allowing the developer to add the pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Spring's transaction support is not tied to J2EE environments and it can be also used in container less environments.
- **JDBC Exception Handling:** The JDBC abstraction layer of the Spring offers a meaningful exception hierarchy, which simplifies the error handling strategy.
- **Integration with Hibernate, JDO, and iBATIS:** Spring provides best Integration services with Hibernate, JDO and iBATIS.
- **AOP Framework:** Spring is best AOP framework.
- **MVC Framework:** Spring comes with MVC web application framework, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. But other frameworks can be easily used instead of Spring MVC Framework.

1.2 SPRING ARCHITECTURE

Spring is well-organized architecture consisting of seven modules. This is shown in the figure below.

[1] Spring Framework definition :

(1) Spring AOP: One of the key components of Spring is the AOP framework. AOP is used in Spring:

(a) To provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management, which builds on Spring's transaction abstraction.

(b) To allow users to implement custom aspects, complementing their use of OOP with AOP.

(2) Spring ORM: The ORM package is related to the database access. It provides integration layers for popular object-relational mapping APIs, including JDO, Hibernate and iBatis.

(3) Spring Web: The Spring Web module is part of Spring's web application development stack, which includes Spring MVC.

(4) Spring DAO: The DAO (Data Access Object) support in Spring is primarily for standardizing the data access work using the technologies like JDBC, Hibernate or JDO.

(5) Spring Context: This package builds on the beans package to add support for message sources and for the Observer design pattern, and the ability for application objects to obtain resources using a consistent API.

(6) Spring Web MVC: This is the Module which provides the MVC implementations for the web applications.

(7) **Spring Core:** The Core package is the most import component of the Spring Framework. This component provides the Dependency Injection features. The BeanFactory provides a factory pattern which separates the dependencies like initialization, creation and access of the objects from your actual program logic.

The following diagram represents the Spring Framework Architecture.

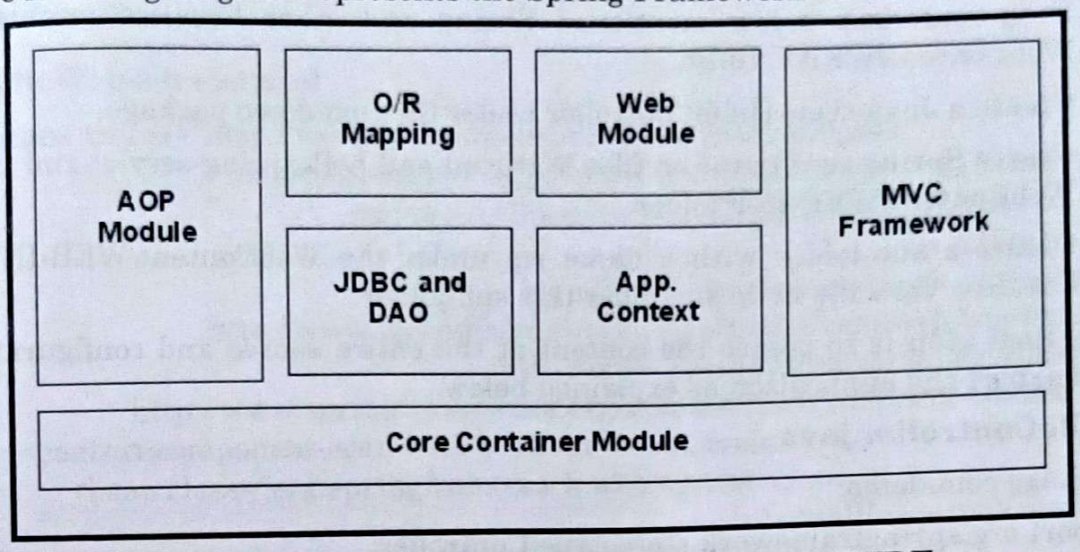

FIGURE 10.2 : FRAMEWORK ARCHITECTURE

11.3 SPRING & MVC

Spring MVC helps in building flexible and loosely coupled web applications. The Model-view-controller design pattern helps in separating the business logic, presentation logic and navigation logic. Models are responsible for encapsulating the application data. The Views render response to the user with the help of the model object. Controllers are responsible for receiving the request from the user and calling the back-end services.

The figure below shows the flow of request in the Spring MVC Framework.

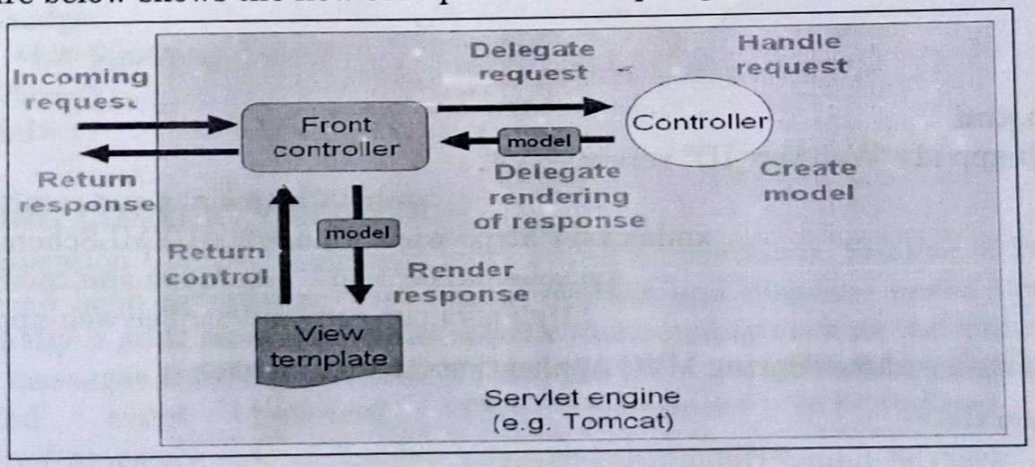

FIGURE 10.3 : REQUEST FLOW IN SPRING MVC FRAMEWORK

[1] Spring and MVC Example :

The following steps creates a basic HelloSpring example using Eclipse:

(1) Create a Dynamic Web Project with a name HelloSpring and create a package com.demo under the src folder in the created project.

(2) Drag and drop below mentioned Spring and other libraries into the folder WebContent/WEB-INF/lib.

(3) Create a Java class HelloController under the com.demo package.

(4) Create Spring configuration files Web.xml and hellospring-servlet.xml under the WebContent/WEB-INF folder.

(5) Create a sub-folder with a name jsp under the WebContent/WEB-INF folder. Create a view file hello.jsp under this sub-folder.

The final step is to create the content of the entire source and configuration files and export the application as explained below.

HelloController.java

```
package com.demo;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.ui.ModelMap;
@Controller
@RequestMapping("/hello")
public class HelloController{
    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC!");
        return "hello";
    }
}
```

Web.xml

```
<web-app id="WebApp_ID" version="2.4"
                                        xmlns="http://java.sun.com/xml/ns/j2ee"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                              xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                                http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>Spring MVC Application</display-name>
    <servlet>
        <servlet-name>HelloSpring</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
```

```
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloSpring</servlet-name>
            <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

HelloWeb-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
                    xmlns:context="http://www.springframework.org/schema/context"
                            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                                    xsi:schemaLocation="
                                        http://www.springframework.org/schema/beans
                    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                                        http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <context:component-scan base-package="com.demo" />
        <bean class="org.springframework.web.servlet.view.
                                                    InternalResourceViewResolver">
            <property name="prefix" value="/WEB-INF/jsp/" />
            <property name="suffix" value=".jsp" />
        </bean>
    </beans>
```

hello.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
    <head>
        <title>Hello World</title>
    </head>
    <body>
      <h2>${message}</h2>
    </body>
</html>
```

11.4 SPRING CONTEXT DEFINITION

The Application Context is spring's more advanced container. Similar to BeanFactory it can load bean definitions, wire beans together and dispense beans upon request. Additionally it adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the *org.springframework.context.ApplicationContext* interface.

The *ApplicationContext* includes all functionality of the *BeanFactory*, it is generally recommended over the *BeanFactory*. BeanFactory can still be used for light weight applications like mobile devices or applet based applications.

The most commonly used ApplicationContext implementations are:

→ **FileSystemXmlApplicationContext**: This container loads the definitions of the beans from an XML file. Here you need to provide the full path of the XML bean configuration file to the constructor.

→ **ClassPathXmlApplicationContext** This container loads the definitions of the beans from an XML file. Here you do not need to provide the full path of the XML file but you need to set CLASSPATH properly because this container will look bean configuration XML file in CLASSPATH.

→ **WebXmlApplicationContext:** This container loads the XML file with definitions of all beans from within a web application.

11.5 INVERSION OF CONTROL (IOC) IN SPRING

In Spring, the ***Inversion of Control (IoC)*** principle is implemented using the ***Dependency Injection (DI)*** design pattern.

[1] Dependency Injection (DI) :

Martin Fowler, the inventor of the concept of "Dependency Injection (DI)" as a part of Inversion of Control (IoC) which is a general concept, and it can be expressed in many different ways in the Spring Framework. Dependency Injection is merely one concrete example of Inversion of Control.

The application classes in a complex java application should be independent so that the other Java classes can be reused and can be tested independently of other classes while doing unit testing. Dependency Injection joins these classes together and at the same time keeps them independent. Exactly, "dependency injection" term can be well defined when it is distinguished separately into two parts. Here the dependency part states about the association between two classes. For example, class A is dependent on class B. The second part, injection states that all this means is that class B will get injected into class A by the IoC.

Dependency injection is implemented by passing parameters to the constructor or by post-construction using setter methods.

Numerous organizations and institutions use the Spring Framework in this manner to engineer robust and *maintainable* applications.

11.6 ASPECT ORIENTED PROGRAMMING IN SPRING (AOP)

Aspect oriented programming (AOP) is an important part of the Spring framework. It redefines the programming model of OOP (Object-Oriented Programming).

The functions that span multiple points of an application are called **cross-cutting concerns** and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects including logging, declarative transactions, security, and caching etc. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.

The Dependency Injection helps to decouple the application objects from each other whereas AOP helps to decouple cross-cutting concerns from the objects that they affect.

The AOP module of Spring Framework provides aspect-oriented programming implementation allowing defining method-interceptors and point-cuts to cleanly decouple code that implements functionality that should be separated.

AOP is used in the Spring Framework to-

→ Provide declarative enterprise services, especially as a replacement for EJB declarative services like *declarative transaction management*.

→ Allow users to implement custom aspects, complementing their use of OOP with AOP.

:: QUESTIONS ::

(1) Give a brief introduction of Spring Framework.
(2) Explain in detail the Spring Architecture.
(3) Explain in detail Spring & MVC.
(4) Define Spring Context in brief.
(5) What is Inversion of Control (IoC) in Spring?
(6) Write a note on Aspect Oriented programming in Spring (AOP).

:: M.C.Q. ::

1. Spring Framework is a _____.
   (a) Java platform (b) Java Enterprise
   (c) Java Environment (d) All of three
2. Spring enables you to build applications from_____.
   (a) POJOs (b) POOJs (c) PJs (d) None
3. POJOs stand for _____.
   (a) Plain Objects Java Old (b) Plan Old Java Objects
   (c) Plain Old Java Objects (d) None of these
4. IoC stands for _____.
   (a) Inverse of Control (b) Inverse Of Control
   (c) (a) & (b) ***(d) Inversion of Control***
5. AOP stands for _____.
   (a) Aspect Oriented Program ***(b) Aspect Oriented Programming***
   (c) (a) & (b) (d) All of these
6. The Core Container consists of the _____ modules.
   (a) spring-core (b) spring-beans (c) spring-context ***(d) All of these***
7. The Context (spring-context) module builds on the solid base provided by th _____ modules.
   (a) Core & Beans (b) Beans & Core (c) Content & Beans ***(d) (a) & (b)***

12.1 UNDERSTANDING STRUTS FRAMEWORK

Struts is an open source framework based on the popular Model, View, Controller (MVC) architecture and also extends the functionality of the Java Servlet API. This framework provides the functionalities based on standard technologies, such as JSP pages, JavaBeans, resource bundles, and XML for creating, maintaining and extending flexible web applications.

The core part of the Struts framework is defined with a flexible control layer that has the basis on the standard technologies like Java Servlets, JavaBeans, ResourceBundles, and XML, as well as various Jakarta Commons packages. Struts application architectures are based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm.

Struts own a Controller component and integrate with other technologies to provide the Model and the View. For the Model, Struts can interact with standard data access technologies, like JDBC and EJB, as well as most any third-party packages, like Hibernate, iBATIS (or the new MyBatis), or Object Relational Bridge. For the View, Struts works well with JavaServer Pages, including JSTL (JavaServer Pages Standard Tag Library) and JSF (Java Server Faces), as well as Velocity Templates, XSLT (EXtensible Stylesheet Language Transformations), and other presentation systems.

The Struts framework provides the invisible solid foundation every professional web application needs to survive. Struts help to create an extensible development environment for the application based on published standards and proven design patterns.

Finally, there are several different ways of looking at Struts. The three main ways are that Struts is:

→ **An MVC Framework:** Struts provide a unified framework for deploying Servlet and JSP applications that use the MVC architecture.

→ **A Collection of Utilities:** Struts provide utility classes so that many of the most common tasks in **Web** application development can be handled.

→ **A Set of JSP Custom Tag Libraries:** Struts provide custom tag libraries for output of bean properties, generating HTML forms, iterating over different types of data structures, and conditionally HTML output.

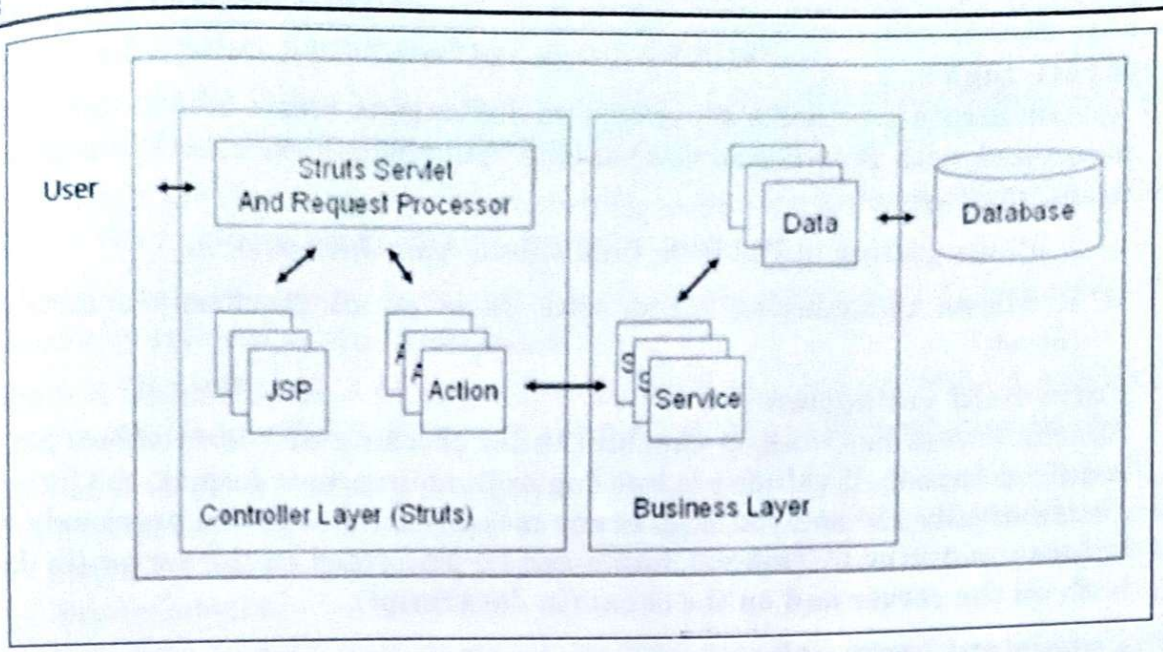

FIGURE 11.1 : STRUTS FRAMEWORK

12.2 COMPARISION WITH MVC USING REQUESTDISPATCHER AND THE EL

The following comparison shows the advantages and disadvantages of using Struts as compared to the RequestDispatcher and the EL (Expression Language) with MVC.

→ **Advantages :**

Struts provide certain benefits as compared to MVC with RequestDispatcher and the EL:

» **Centralized File Based configuration :**
Struts values are represented in XML or property files instead of the static java code in the programs. There is no need to modify or recompile the java code but many modifications can be applied using a single file. This is beneficial to the Java Web developers as they can indulge into faster development of the business logic and other tasks without bothering about the system design.

» **Form Beans :**
In JSP a property="*" can be used with jsp:setProperty to automatically populate a JavaBean component based on incoming request parameters. But this capability is unavailable to servlets in the standard API, even though with MVC it is really servlets, not JSP pages that should usually be the target of form submissions. Apache Struts is able to extend this capability to Java code and adds in several useful utilities, all of which serve to greatly simplify the processing of request parameters.

» **Bean tags :**
Apache Struts provides a set of custom JSP tags (bean:write, in particular) to allow easy output the properties of JavaBeans components. Basically, these are concise and powerful variations of the standard jsp:useBean and jsp:getProperty tags.

» **HTML tags :**
Apache Struts provides a set of custom JSP tags to create HTML forms that are associated with JavaBeans components. This bean/form association serves two useful purposes:

→ It allows getting initial form-field values from Java objects.

→ It allows redisplaying forms with some or all previously entered values intact.

» **Form field validation :**
Apache Struts has built-in capabilities for checking that form values are in the required format. If values are missing or in an improper format, the form can be automatically redisplayed with **error** messages and with the previously entered values maintained. This validation can be performed on the server (in Java), or both on the server and on the client (in JavaScript).

» **Consistent approach :**
Struts encourage consistent use of MVC throughout the application.

→ **Disadvantages :**
Although Struts has a number of significant advantages over the standard Servlet and JSP APIs alone, due to its complexity it has some serious drawbacks as well:

» **Bigger Learning Curve :**
MVC can be used with the standard RequestDispatcher on the basis of standard JSP and Servlet APIs. To use MVC with Struts one has to be comfortable with the standard JSP and Servlet APIs and a large and elaborate framework that is almost equal in size to the core system. This drawback is especially significant with smaller projects, near-term deadlines, and less experienced developers; much time can be spent learning Struts for building an actual system.

» **Worse Documentation :**
Compared to the standard servlet and JSP APIs, Struts has fewer online resources, and many first-time users find the online Apache documentation confusing and poorly organized.

» **Less Transparent :**
With Struts applications, there is a lot more going on behind the scenes than with normal Java-based Web applications. As a result, Struts applications are:

→ Harder to understand

→ Harder to benchmark and optimize

» **Rigid Approach :**
The flip side of the benefit that Struts encourages a consistent approach to MVC is that Struts makes it difficult (but by no means impossible) to use other approaches.

Now, Struts have some significant disadvantages. Struts have some significant advantages, but the overhead and complexity of Struts can be a burden as well.

12.3 DOWNLOADING AND CONFIGURING STRUTS

This section walks through the process of downloading the Struts software, setting up your environment, and testing a simple application.

(1) Download the Struts zip file. Start at http://jakarta.apache.org/site/binindex.cgi , or follow the link from http://jakarta.apache.org/struts/.

(2) Unzip into any chosen directory. For example, unzip into C:\jakarta-struts-1.1. Normally referred as struts_install_dir.

(3) Update CLASSPATH. Add struts_install_dir/lib/struts.jar to the CLASSPATH used by the compiler or IDE (not server). Here are three possible ways of setting it:

→ Set it in autoexec.bat file. E.g., on Windows if unzipped into C:\jakarta-struts-1.1, the following can be added to C:\autoexec.bat: set CLASSPATH=C:\jakarta-struts-1.1\lib\struts.jar;%CLASSPATH%. On Unix/Linux, use .cshrc, .bashrc, or .profile instead.

→ Set it using the system settings. On WinXP, go to the Start menu and select Control Panel, then System, then the Advanced tab, then the Environment Variables button. On Win2K/WinNT, go to the Start menu and select Settings, then Control Panel, then System, then Environment. Either way, enter the CLASSPATH value from the previous bullet.

→ Set it in editor or IDE. Most IDEs have a way of specifying the JAR files needed to compile projects. Or, make a small .bat file (Windows) or shell script (Unix/Linux) that supplies the struts.jar file as the argument to -classpath for javac.

→ Install an XML parser. In JDK 1.4 or Apache Tomcat, this step is not necessary, since they already come with an XML parser. But, for JDK 1.2 or 1.3 with another server XML parsing libraries are needed can be obtained at http://xml.apache.org/.

12.4 TESTING STRUTS

(1) Install struts-blank.war.
Install the Web application from struts_install_dir/webapps/struts-blank.war on server. For example, with Apache Tomcat, copy struts_install_dir/webapps/struts-blank.war to tomcat_install_dir/webapps/.

(2) Start or restart the server.
Most servers only recognize new Web apps when the server is started.

(3) Access http://localhost/struts-blank/.
This URL assumes are running the server on desktop and are using port 80. In general, access http://hostname:port/struts-blank.

12.5 STRUTS FLOW OF CONTROL

The Struts framework provides several components that make up the **Control** layer of a MVC-style application. These include a controller Servlet, developer-defined request handlers, and several supporting objects.

The Struts custom tag libraries provide direct support for the **View** layer of a MVC application. Some of these access the control-layer objects. Others are generic tags found convenient when writing applications. Other taglibs, including JSTL, can also be used with Struts. Other presentation technologies, like Velocity Templates and XSLT can also be used with Struts.

The **Model** layer in a MVC application is often project-specific. Struts is designed to make it easy to access the business-end of your application, but leaves that part of the programming to other products, like JDBC, Enterprise Java Beans, Object Relational Bridge, or Simper, to name a few.

Let's step through how this all fits together.

When initialized, the controller parses a configuration file (struts-config.xml) and uses it to deploy other control layer objects. Together, these objects form the **Struts Configuration**. The Struts Configuration defines (among other things) the ActionMappings [org.apache.struts.action.ActionMappings] for an application.

The Struts controller servlet consults the ActionMappings as it routes HTTP requests to other components in the framework. Requests may be forwarded to JavaServer Pages or Action [org.apache.struts.action.Action] subclasses provided by the Struts developer. Often, a request is first forwarded to an Action and then to a JSP (or other presentation page). The mappings help the controller turn HTTP requests into application actions.

An individual ActionMapping [org.apache.struts.action.ActionMapping] will usually contain a number of properties including:

→ a **request path** (or "URI"),

→ the **object type** (Action subclass) to act upon the request, and

→ other **properties** as needed.

The Action object can handle the request and respond to the client (usually a Web browser) or indicate that control should be forwarded elsewhere. For example, if a login succeeds, a login action may wish to forward the request onto the mainMenu page.

Action objects have access to the application's controller servlet, and so have access to that servlet's methods. When forwarding control, an Action object can indirectly forward one or more shared objects, including JavaBeans, by placing them in one of the standard contexts shared by Java Servlets.

For example, an Action object can create a shopping cart bean, add an item to the cart, place the bean in the session context, and then forward control to another mapping. That mapping may use a JavaServer Page to display the contents of the user's cart. Since each client has their own session, they will each also have their own shopping cart.

In a Struts application, most of the business logic can be represented using JavaBeans. An Action can call the properties of a JavaBean without knowing how it actually works. This encapsulates the business logic, so that the Action can focus on error handling and where to forward control.

JavaBeans can also be used to manage input forms. A key problem in designing Web applications is retaining and validating what a user has entered between requests. With Struts, one can define own set of input bean classes, by subclassing ActionForm [org.apache.struts.action.ActionForm]. The ActionForm class makes it easy to store and validate the data for your application's input forms. The ActionForm bean is automatically saved in one of the standard, shared context collections, so that it can be used by other objects, like an Action object or another JSP.

The form bean can be used by a JSP to collect data from the user, by an Action object to validate the user-entered data, and then by the JSP again to re-populate the form fields. In the case of validation errors, Struts has a shared mechanism for raising and displaying error messages.

Another element of the Struts Configuration is the ActionFormBeans [org.apache.struts.action.ActionFormBeans]. This is a collection of descriptor objects that are used to create instances of the ActionForm objects at runtime. When a mapping needs an ActionForm, the servlet looks up the form-bean descriptor by name and uses it to create an ActionForm instance of the specified type.

Here is the sequence of events that occur when a request calls for a mapping that uses an ActionForm:

- → The controller servlet either retrieves or creates the ActionForm bean instance.
- → The controller servlet passes the bean to the Action object.
- → If the request is being used to submit an input page, the Action object can examine the data. If necessary, the data can be sent back to the input form along with a list of messages to display on the page. Otherwise the data can be passed along to the business tier.
- → If the request is being used to create an input page, the Action object can populate the bean with any data that the input page might need.

The Struts framework includes custom tags that can automatically populate fields from a JavaBean. All most JavaServer Pages really need to know about the rest of the framework is the field names to use and where to submit the form.

Other Struts tags can automatically output messages queued by an Action or ActionForm and simply need to be integrated into the page's markup. The messages are designed for localization and will render the best available message for a user's locale.

The Struts framework and its custom tag libraries were designed from the ground-up to support the internationalization features built into the Java platform. All the field labels and messages can be retrieved from a message resource. To provide messages for another language, simply add another file to the resource bundle.

Internationalism aside, other benefits to the message resources approach are consistent labeling between forms, and the ability to review all labels and messages from a central location.

For the simplest applications, an Action object may sometimes handle the business logic associated with a request. **However, in most cases, an Action object should invoke another object, usually a JavaBean, to perform the actual business logic.** This lets the Action focus on error handling and control flow, rather than business logic. To allow reuse on other platforms, business-logic JavaBeans should not refer to any Web application objects. The Action object should translate needed details from the HTTP request and pass those along to the business-logic beans as regular Java variables.

In a database application, for example:

→ A business-logic bean will connect to and query the database,

→ The business-logic bean returns the result to the Action,

→ The Action stores the result in a form bean in the request,

→ The JavaServer Page displays the result in a HTML form.

Neither the Action nor the JSP need to know (or care) from where the result comes. They just need to know how to package and display it.

The Struts release also includes several Developer Guides covering various aspects of the frameworks, along with sample applications, the standard Javadoc API, and, of course, the complete source code! Struts is distributed under the Apache Software Foundation license. The code is copyrighted, but is free to use in any application.

12.6 PROCESSING REQUESTS WITH ACTION OBJECTS

There are six basic steps in processing requests with Action Objects. With Struts, the normal processing flow is that a form submits data to a URL of the form *xyz*.do. That address is mapped by struts-config.xml to an Action object, whose execute method handles the request. One of the arguments to execute is a form bean that is automatically created and whose properties are automatically populated with the incoming form data. The Action object then invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope. The Action uses mapping.findForward to return a condition, and the conditions are mapped by struts-config.xml to various JSP pages. The system forwards the request to the appropriate JSP page, where the bean:write tag is used to output properties of the form bean and results beans. Optionally, both the input form and the final JSP pages can use bean:message to output standard messages and labels as defined in a system property file.

Here are the minimum steps needed to create an application that follows this process. This section will focus on the basic flow and the Action objects.

[1] Modify struts-config.xml.
Use WEB-INF/struts-config.xml to:

- Designate Action classes to handle requests for *xyz*.do.
- Specify the URLs that apply in various situations.
- Declare any form beans that are being used.

Be sure to restart the server after modifying struts-config.xml; the file is read only when the Web application is first loaded.

[2] Define a form bean.
This bean is a class the extends ActionForm and will represent the data submitted by the user. It is automatically populated when the input form is submitted.

[3] Create results beans.
In the MVC architecture, the business-logic and data-access code create the results and the JSP pages present them. To transfer the results from one layer to the other, they are stored in beans. These beans differ from form beans in that they need extend no particular class, and they represent the *output of* the computational process, not *the input to* the process.

[4] Create an Action object to handle requests.
The struts-config.xml file designates the Action object that handles requests for various URLs. The Action objects themselves need to do the real work: invoke the appropriate business- and data-access-logic, store the results in beans, and designate the type of situation (missing data, database error, success category 1, success category 2, etc.) that is appropriate for the results. The struts-config.xml file then decides which JSP page should apply to that situation.

[5] Create form that invokes *xyz*.do.
You need to create an input form whose ACTION corresponds to one of the *.do addresses listed in struts-config.xml. At a minimum, the input form should look like this:

```
<FORM ACTION=".../xyz.do" ...>...</FORM>
```

[6] Display results in JSP.
Since Struts is built around the MVC architecture, these JSP pages should avoid JSP scripting elements whenever possible. For basic Struts, these JSP pages usually use the Struts bean:write tag, but in JSP 2.0 the JSP 2.0 expression language is a viable alternative. In complex cases, the JSP pages also use the JSP Standard Tag Library (JSTL) or the Struts looping/logic tags.

In most cases, the JSP pages only make sense when the request is funneled through the Action, since it is the Action that invokes the logic that creates the data that the JSP pages will display. So, the JSP pages are placed in WEB-INF, where RequestDispatcher can forward to them but where clients cannot access them directly. If the JSP pages makes sense independently of the Action (e.g., if they display session data), then the JSP pages should be placed in a regular subdirectory of the Web application, and the forward entries in struts-config.xml should say <forward ... **redirect="true"**/>.

12.7 HANDLING REQUEST PARAMETERS WITH FORMBEANS

In the previous example, explicitly called request.getParameter to obtain the form data. We used this manual approach because it is required to concentrate only on the action mappings, forward entries, and structure of the Action class. In a real application, however, making repeated request.getParameter calls is tedious and error prone. Fortunately, Struts has a very convenient and powerful facility that simplifies this process: form beans.

Basically, a form bean is little more than a JavaBean with property names that correspond to request parameter names. When a user submits a request, the system instantiates the bean; populates it based on the incoming request parameters (i.e., if any incoming parameter name matches a bean property name, it calls the corresponding setter method); stores it in request, session, or application scope; and passes it to the execute method of your Action. That way, there is no need to call request.getParameter a zillion times, but instead get all of the form data in one fell swoop. This idea is sort of what jsp:setProperty with property="*" does , except that jsp:setProperty is exactly backward. It allows the process in a JSP page (where you don't want to), but doesn't expose the underlying Java API (so can call it from Java code). In addition, Struts form beans have two additional capabilities: a reset method that gets called before any of the setter methods is called, and a validate method that is called after all of the setter methods are called.

12.8 STRUTS FLOW OF CONTROL: UPDATES FOR BEAN USE

The Struts Flow of Control and the six steps to implement it using the flow of the request through the Struts framework is presented using bean.

- → The user requests a form. In real applications, the input form is almost always built with the Struts html: tags. Here use normal HTML.
- → The form is submitted to a URL of the form xyz.do. The form contains an ACTION of the form xyz.do. The Struts system receives the request, where a mapping in struts-config.xml associates the address with an Action object.
- → The execute method of the Action object is invoked. Struts instantiates a form bean; calls all setter methods for form parameter names that match bean property names; stores the bean in request, session, or application scope; and passes the bean to execute as the second argument. Rather than calling request.getParameter explicitly, the execute method just uses the form bean. The execute method also invokes business logic and data-access logic, placing the results in normal beans stored in request, session, or application scope. The execute method then uses mapping.findForward to return various conditions that are mapped by struts-config.xml to various JSP pages.
- → Struts forward the request to the appropriate JSP page. Struts invoke the JSP page given in the forward entry. That JSP page uses bean:write or the JSP 2.0 EL to output the properties of either the results beans or the form bean.

Next, review the six basic steps in using Struts, emphasizing the bean-related parts.

[1] The Six Basic Steps in Using Struts :

In this section, the emphasis is on form beans and results beans.

Step 1 : Modify struts-config.xml

You need to add three main entries to WEB-INF/struts-config.xml.

Use action entries to map incoming .do addresses to Action classes. This step is mostly done the same way as in the previous examples, but now you also list the name and scope of the form bean, as shown here:

```
<action path="..." type="..." name="beanName" scope="request">
```

Unfortunately, the default scope in Struts is session, so always specify scope="request" unless you have a specific need for session tracking.

Use forward entries to map return conditions to JSP pages. This is done exactly the same way as in the previous examples.

Declare form beans. Put a form-bean entry in the form-beans section for each bean that is associated with an Action. Give each form bean a name.

```
<form-beans>
  <form-bean name="beanName" type="package.Class"/>
</form-beans>
```

Step 2 : Define a Form Bean

A form bean is a class that extends ActionForm and represents the data submitted by the user. It should have a bean property for each incoming request parameter. For instance, if the input form has firstName and lastName parameters, the bean must have setFirstName and setLastName methods, and usually also has getFirstName and getLastName methods.

When the user requests the URL corresponding to the Action, the bean is automatically instantiated, the setter methods that match any nonempty parameters are called, the bean is stored in the appropriate scope, and the bean is passed to the execute method.

Step 3 : Create the Results Beans

In the MVC architecture, results beans are just normal beans that hold data that the presentation-layer JSP pages will display. The form bean represents the input data, the data that came from the HTML form. In most applications, the more important type of data is the result data, the data created by the business logic to represent the results of the computation or database lookup.

For the beans to be available to the output JSP, you need to store them in the request, session, or application scope. Simply call the setAttribute method on the corresponding scope object (HttpServletRequest, HttpSession, ServletContext) just as in normal non-Struts applications. It is also possible to create extra properties in the form bean to be used to store the results. Both approaches are common, although it is slightly clearer to have separate beans for the results.

Step 4 : Define an Action Class to Handle the Requests

When using form beans, however, all you need to do is cast the ActionForm to your specific class, then access the already fully populated bean. For example:

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
    throws Exception {
        MyFormBean myBean = (MyFormBean)form;
        accessInputData(myBean);
        ...
}
```

Step 5 : Create a Form That Invokes xyz.do

In a real Struts application, you would use the Struts html:form custom tag to build the input form. For now, use the standard HTML FORM element.

Step 6 : Display the Results in a JSP Page

Finally, you need to present the results in the JSP page specified in the forward element. Use the Struts bean:write tag to output the bean properties.

```
<bean:write name="beanName" property="beanProperty"/>
```

Also, if the servlet engine supports the JSP 2.0 API, then you may consider using the JSP 2.0 EL as an alternative to the bean:write tag.

12.9 PREPOPULATING AND REDISPLAYING INPUT FORMS

In the previous section, it is showed how to create form beans that the system instantiates and populates automatically. It is also showed how to output bean properties in results pages. In this section, it is shown how to associate form beans with the input page using the Struts html: tags. Associating form beans with input forms provides three advantages:

[1] It guarantees that the request parameter names and the bean property names stay in sync. If you write the form by hand, you might change the parameter names but forget to change the bean. When using the html: tags, you explicitly refer to the bean property names when defining text fields or other input elements, so if you use the wrong name, you get an immediate error message.

[2] It lets you prepopulate text fields with initial values. By associating a bean with the input page, the initial text field values can come from your application. So, for example, if the available health plans change in your database, you don't have to separately remember to change hard-coded HTML on the page that asks employees to choose a health plan.

[3] It lets you redisplay values in the text fields. If you send the user back to the original form, the originally entered values are automatically filled in. That's because the form is already associated with a bean, but now the bean is the one that was just created from the request.

[1] Struts Flow of Control: Use of html tags and form beans:

First, review the Struts flow of control, emphasizing use of the html: tags and form beans to prepopulate fields of the input page. Then, details on the syntax of the html: tags are given and provide an example of their use.

→ The user requests a form. This form is built with html:form, html:text, and similar elements. These tags keep input field names in sync with bean property names and let initial values of text fields and other input elements come from the application.

→ The form is submitted to a URL of the form xyz.do. The html:form tag automatically prepends the Web application name and appends .do. That way, the action listed in the form and the path listed in action match exactly. The URL is mapped in struts-config.xml to an Action class.

→ The execute method of the Action object is invoked. One of the arguments to execute is a form bean that is automatically created and whose properties are automatically populated based on incoming request parameters of the same name. The Action invokes business logic and data-access logic, placing the results in normal beans stored in the request, session, or application scope. The Action uses mapping.findForward to return a condition, and the conditions are mapped by struts-config.xml to various JSP pages.

→ Struts forwards request to the appropriate JSP page. The page can use bean:write or the JSP 2.0 EL to output bean properties.

The Six Basic Steps in Using Struts (with prepopulating and redisplaying forms)

Now, review the six steps in using Struts, but emphasize the use of the html: tags to associate beans with input forms.

1.	Modify struts-config.xml. Map incoming URLs to Action objects, map return conditions to JSP pages, and declare form beans.
2.	Define a form bean. Define a class that extends ActionForm and has properties that match the request parameter names. Declare the bean in the form-beans section of struts-config.xml.
3.	Create the results beans. If the output JSP pages need data resulting from the business logic, then create results beans to hold the data. These beans need extend no special class and need not be declared in struts-config.xml.
4.	Define an Action class to handle the request. Create an Action class with an execute method to process the request. Obtain input data from the second argument to execute (the form bean). Call business logic and data-access logic and store the results in beans. Use mapping.findForward to return a result condition.
5.	Create a form that invokes xyz.do. Create a form to collect the user's input. Use <html:form action="/path-as-in-struts-config">

	To instantiate a form bean and associate it with the form. Then, use various input elements such as <html:text name="beanPropertyName"/> The text field is prepopulated with the result of the getter method corresponding to beanPropertyName, and, when the form is submitted, the text field value is stored in a form bean by calling the setter method corresponding to beanPropertyName.
6.	Display the results in a JSP page. Use bean:write or the JSP expression language to output bean properties.

Details of these steps are as below :

Step 1 : Modify struts-config.xml

You need three main entries in struts-config.xml.

- → Use action entries to map incoming .do addresses to Action classes. The path attribute should exactly match the action of html:form, the type attribute should be the fully qualified Action class name, the name attribute should match a name from a form-bean element, and the scope attribute should be request or session.
- → Use forward entries to map return conditions to JSP pages. The name attribute should match a return condition from the execute method of the Action and the path attribute should list a JSP page in WEB-INF. If the same forward entry appears in more than one action, you can move the entry to the global-forwards section.
- → Declare form beans. Put a form-bean entry in the form-beans section for each bean that is associated with an Action. Give each form bean a name.

Remember to restart the server after modifying struts-config.xml.

Step 2 : Define a Form Bean

Your bean will extend ActionForm and have a bean property for each incoming request parameter. In addition to using the bean in the execute method of the Action and in the final JSP pages, use the bean in the input form to give names and values to the various input elements.

Step 3 : Create the Results Beans

Results beans are normal beans as used in the MVC approach when implemented directly with RequestDispatcher, and are created and used in the same way as described in the previous section.

Step 4 : Define an Action Class to Handle the Request

As in the previous section, rather than calling request.getParameter explicitly, use the form bean passed as the second argument of the execute method. Cast the ActionForm argument to the specific form bean class, then use the appropriate getter methods to access the properties of the object.

Step 5 : Create a Form That Invokes xyz.do

Rather than using the standard HTML FORM and INPUT tags, use html:form and html:text (and related tags). The html:form tag associates a bean with the form. The html:text tag automatically uses bean property names for each text field NAME and bean property values for each text field VALUE.

Step 6 : Display the Results in a JSP Page

As before, the JSP page uses bean:write or the JSP EL to output properties of the form and result beans.

[2] Using Struts html : Tags

Here the use of Struts html: tags is shown. To effectively use these tags, the following techniques are required:

→ Using the Struts html:form element instead of the standard HTML FORM element to declare the form.

→ Using html:text elements to declare the input fields of the form.

Before using any of the html: tags, declare the tag library:

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
```

[3] Using Properties Files :

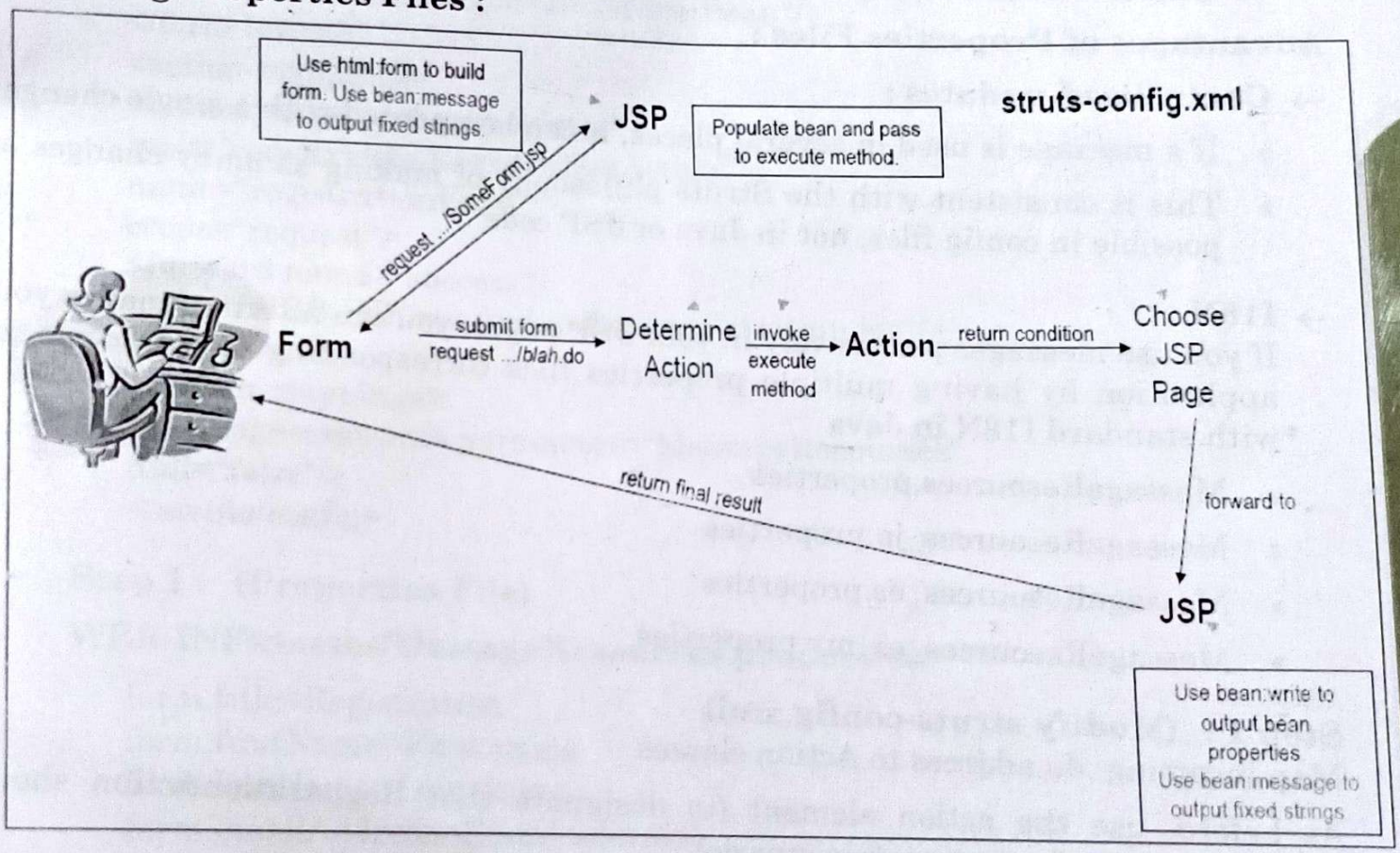

FIGURE 2 : THE PROPERTIES FILE

(1) Create a properties file in WEB-INF/classes
E.g., WEB-INF/classes/MessageResources.properties

(2) Define strings in the properties file

some.key1=first message

some.key2=second message

some.key3=some parameterized message: {0}

(3) Load the properties file in struts-config.xml

– <message-resources parameter="MessageResources"/>

(4) Output the messages in JSP pages
Load the tag library :

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>

Output the messages using bean:message

→ First message is <bean:message key="some.key1"/>

→ **Second :** <bean:message key="some.key2"/>

→ **Third :** <bean:message key="some.key3" arg0="replacement"/>

Advantages of Properties Files :

→ **Centralized updates :**

» If a message is used in several places, it can be updated with a single change.

» This is consistent with the Struts philosophy of making as many changes as possible in config files, not in Java or JSP code.

→ **I18N :**
If you use messages pervasively in your JSP pages, you can internationalize your application by having multiple properties files corresponding to the locale, as with standard I18N in Java

» MessageResources.properties

» MessageResources_jp.properties

» MessageResources_es.properties

» MessageResources_es_mx.properties