## Java ->Servlet (Java classes which takes request send the by user and Response to the user in form plain text or HTML Format)

## Java->Servlet(pares and decide the view (jsp) to show ->JSP(View send the request and display the Response in a html format)

## JEE-Java Enterprsie EJB

## What is Spring?

A Java application framework which allows the user to create a Java enterprise application with inversion of control container is known as Spring Framework. The Spring Framework is open source with extensions for building web applications on the top of the Java EE (Enterprise Edition) platform.

Plain Old Java Objects (POJOs) can be built from Spring with enterprise services non-invasively. Following are some examples to the Java SE programming model:

* A Java method can be created in a database without the interaction of APIs
* A remote procedure of local Java without involving remove APIs
* A Java method for a message handler without JMS (**Java Messaging Service**) APIs
* A management operation by a local Java method without JMX (**Java Management Extensions**) APIs

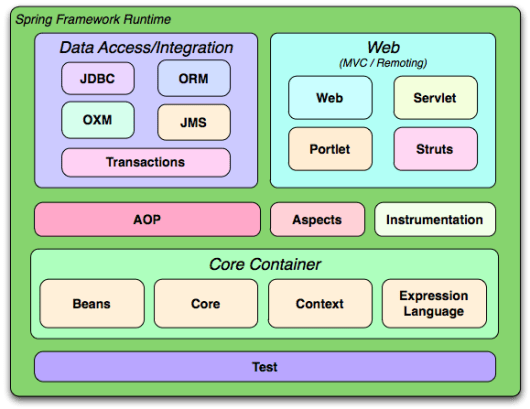
Spring is a great framework for the development of Enterprise grade applications. It is a light-weight framework for the development of enterprise-ready applications. It provides very simple and rich facilities to integrate various frameworks, technologies, and services in the applications. One of the main reason for using the Spring framework is to keep the code as simple as possible. It pushes the way to develop enterprise applications with **loosely coupled simple java beans**. For this reason, the spring framework can also be called a Plain Old Java Object (POJO) framework.

Let’s see the next section about the modules in the Spring framework. This Spring tutorial explains a lot of major modules of the Spring Framework.

## Modules in Spring Framework

The Spring Framework has been categorized into modules. All modules will not be used in an application, so every module is dependent on the type of application. **There are 20 modules organized in Spring Framework. Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation and Test are the groups.**

A configuration model and a dependency injection mechanism belong to the core container located in the heart of the framework. Beyond that, there are different application architectures such as transactional data, messaging, persistence and web for the foundational support. Servlet-based Spring MVC web-framework is also there in parallel with the Spring WebFlux reactive web framework.



Spring is an open-source framework developed by Spring Source, a division of VMware. Spring framework can be summarized in two ways. Let’s see Spring into two parts in this Spring Tutorial.

## Part 1: The Core Container

In this Spring tutorial, I have explained the Spring framework as a container and as a framework.

The Core Container has Beans, Core, Context and Expression Language modules.

Core and Beans are the basic parts of the framework with Dependency Injection and IoC features. The BeanFactory is a sophisticated implementation of the factory pattern. There is no need for programmatic singletons and they also allow decoupling the specification and configuration of dependencies from the actual program logic.

Context is another module which is built on the solid base by the Core and Beans. This module is a way to access the objects in a framework-style and is similar to a JNDI registry. It has the features from the Bean modules and supports internationalization, resource-loading, event-propagation and the transparent creation. EJB, JMX and basic remoting are also supported by Java EE in this module. The Context module has a focal point which is known as the ApplicationContext.

The most powerful expression language used for manipulation and querying an object graph at runtime is said to be in the Expression Language. It acts as an extension of the unified expression language in the JSP 2.1 specification. This language helps in setting and getting property values, method invocation, and access of context in arrays, property assignment, logical and arithmetic operators and retrieval of objects.

Spring framework can be described as a lightweight container, as it does not involve installation, configuration, start and stop activities associated with a container. It is just a simple collection of few Java ARchive (JAR) files that need to be added to the classpath. The Spring Container takes the classes in the application, creates objects, and manages the life cycle of those objects.

Let’s move to the second part of the Spring framework in this spring tutorial.

## ****Part 2: The Framework****

Spring framework can be described as an Application Programming Interface(API) containing a large collection of the classes, methods, interfaces, annotations, XML tags that can be used in an application. The API provides a variety of factory and facade classes that help you to use any framework or functionality very easily in your application.

The *Core* package is the most fundamental part of the framework and provides the **IoC**and **Dependency Injection**features. The basic concept here is the **BeanFactory**, which provides a sophisticated implementation of the factory pattern which removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

## Data Access/Integrations

This layer comprises of OXM, Object Relational Mapping (ORM), Java Database Connectivity (JDBC), JMS and Transaction modules. The OXM module has an abstraction layer that supports mapping implementation for XStream, JiBX, XMLBeans, Castor and JAXB. A JDBC-abstraction layer is formed in the JDBC module which removes the need to do tedious parsing and coding of itself with database-vendor specific error codes. The ORM module is popular due to its integration functionality with iBatis, Hibernate, JDO and JPA.

In this package, the O/R mapping frameworks can be utilized with a combination of features which are supported by Spring, like the simple declarative transaction management feature. For producing and consuming messages, the Java Messaging Service (JMS) module has been built. The implementation of special interfaces in programmatic and declarative transaction management for classes is only found in the Transaction module of POJOs.

## Aspect Oriented Programming (AOP)

Aspect-Oriented Programming (AOP) is an approach which allows global properties to define method-interceptors and pointcuts in an executable program. The main aim of this programming paradigm is to increase modularity without modifying the code instead of modifying via point-cut specification. It provides integration with AspectJ. The Instrumentation module has class support and a classloader implementation in certain application servers. It also includes programming methods and tools which support modularization of concerns of the source code.

## Spring’s MVC

The Web layer is built up of Web-Servlet, Web-Portlet, and Web-Struts. The Web module has ‘multipart file-upload functionality’ as integration features. The initialization of the IoC container can take place using servlet listeners. It also contains a web-oriented application context of Spring’s remoting support. In Web-Servlet module, web applications are implemented by model-view-controller (MVC). This MVC of the Spring framework has both web forms and model code with a clear separation. The Web-Portlet module has the implementation of MVC in a portlet environment and mirrors the functionality of Web-Servlet module. The support classes for integration with a class Struts web tier is only found in the Web-Struts module.

## Spring Test

This module covers the testing of Spring components with TestNG or JUnit. It balances the consistency of loading in ApplicationContexts and caching of them in Spring. Mock objects are created to test the code in isolation.

## Object XML Mapping

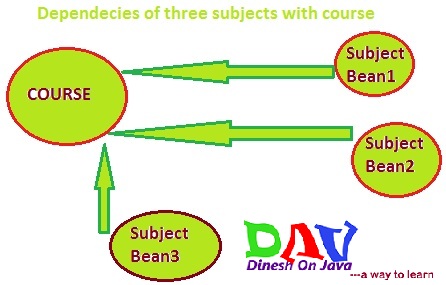
Spring 3.0 introduces the OXM module that was earlier not a part of the core framework. OXM is a mechanism that marshals or converts the object into the XML format and vice versa. There are lots of OXM frameworks, such as Caster, Xstream, JiBX, Java API for XML Binding(JAXP), and XML means. Spring 3.0 provides a uniform API to access any of these OXM frameworks for marshalling and unmarshalling object and XML.

## Dependency Injection (DI)

The technology that Spring is most identified with is the **Dependency Injection (DI)** flavour of Inversion of Control. The Inversion of Control (IoC) is a general concept, and it can be expressed in many different ways and Dependency Injection is merely one concrete example of Inversion of Control.

### **What is dependency injection exactly?**

Let’s look at these two words separately. Here the **dependency** part translates into an association between two classes. For example, class **X** is dependent on class **Y**. Now, let’s look at the second part, **injection**. All this means is that class **Y** will get injected into class**X** by the IoC. Here we don’t need to write lots of code to create the instance of the dependent classes.



Dependency injection promotes loose coupling. It paves the way for the removal of the usual factory and utility classes that we write in our applications.  
Look Following in this Spring Tutorial:

* The **DAO**classes use **Data Sources,**we can inject it into DAO classes.
* The **Service**classes may need to add a few**java beans**

Dependency Injection is also known as Inversion of Control or IoC.  The IoC refers to the control of creating instances. The container has a responsibility to control for creating and constructing objects. The container creates objects and injects them into our applications. Let’s look into the benefits of Spring Tutorial.

## First Program in Spring Framework

In order to develop a code in spring framework, you’ll need to have an eclipse ID and have eclipse software configured in your machine. After it is ready, there are five basic steps to create and run the program:

1. Create a bean class
2. XML File
3. The main method called Demo class
4. Load jar files
5. Run the code

There is a class Employee and for this class, there is an ID like employee ID where the employee name is coming in the below code. We will save the address of the employee with other details. Now, we create an object and a reference. We are now calling the constructor and have defined the objects in a conventional way. Spring has a core module which follows inversion of control which is meant by this term inversion of control. Let us now try to understand this core concept.

class Employee {

// Attributes

int eid;

String ename;

String address;

char gender;

//Methods

//....

}

//Object Construction

Employee eRef = new Employee ();

eRef.eid = 100;

eRef.enam = "test";

eRef.address = "California";

eRef.gender = "Male";

Spring IOC states that we don’t create objects and now objects will be configured in an XML file. There is one module known as ‘sprint container’ and this module is responsible to construct the Java objects by parsing XML pipe. Here, we are not constructing the objects. They will be constructed by the spring container by parsing the XML file. We need to mention the data which these attributes will store in an XML file which is known as inversion of control. Thus, having control over the object construction is given to the spring container. The benefit of using this is that our XML files are not a part of source code so we can configure and manipulate the values.

Another class is created which is named as Client and it is the main method. The object construction has employee id, employee name, employee address and prints all the details.

public class Client {

public static void main (string [] args ) {

//Object construction

Employee eRef = new Employee ();

eRef.setEid(100);

eRef.setEname("test”);

eRef.setEaddress("California");

system.out.println("Employee Details: "+eRef);

}

}

The application is run and the output is generated successfully. Now in order to use spring code framework, an XML file is required. Recalling the above five steps, the bean class here in this example is Employee class and the demo class is the Client class. We loaded the jar files and now we need an XML file. We are including that file in our SRC folder of the project. When an XML file is opened, there is a root tag of DS and beans here.

beans xmlns="http://www.springframework.org/schema/beans"

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-2.5.xsd">

<bean id="emp1" class="com.demo.Employee">

<property name="eid" value = "101"/> <property name="ename" value= "Test4"/>

<property name="eaddress" value= "California"/>

</bean>

<bean id="emp1" class="com.demo.Employee">

<property name="eid" value = "102"/>

<property name="ename" value= "Test1"/>

<property name="eaddress" value= "California"/>

</bean>

 </beans>

After adding object details inside the XML file, we will now do the inversion of control. It will write an API, known as a resource from the Spring framework. If we take that interface now, the resource will become a new classpath which will have only the name of the employee. We have given the name of this XML file here and thereafter we got an interface as ravine factory. Bean factory is a new XML factory from the spring core container. It will parse the XML file and construct the objects.

//Spring Way | IOC (Inversion of Control)

Resource resource = new ClassPathResource("spring.xml");

BeanFactory factory = new XMLBeanFactory(resource);

Employee e1 = factory.getBean("emp1");

Employee e2 = factory.getBean("emp2", Employee.class);

## Implementation of Inversion of Control in Spring Framework

The Inversion of Control (IoC) is also said to be Dependency Injection (DI) which is a process for objects letting them define their dependencies through construct arguments, a factory method or properties. All of them are set on the object instance returned from a factory method. These dependencies are now injected by the container when it creates the bean.

The process is reversed and itself controlling the instantiation by using the direct construction of classes. The packages org.springframework.context and org.springframework.beans are the main components for IoC container. An advanced configuration mechanism which manages every type of object is known as the BeanFactory interface. The sub-interface of BeanFactory is called ApplicationContext. It is more feasible in adding integration with Spring’s AOP features event publication, message resource handling and application-layer specific contexts, for example, the web applications with the use of WebApplicationContext. The BeanFactory has the configuration framework and the ApplicationContext provides more enterprise-specific functionality. The BeanFactory is a subset of the ApplicationContext.

For example, there is a drawing application which draws different shapes a triangle and a circle in Square. With the code, we can draw certain shapes on the screen. A class Circle is there with object C and the circle object has a draw method and a triangle object. The triangle object has a draw method. Now, all these objects can be used to draw a circle or triangle on the screen. An application class is there where we can instantiate a circle object inside this class and a circle can be drawn by instantiating a triangle object inside the class.

//Circle

draw()

//Triangle

draw()

//Application Class

Triangle myTrianlge = new Triangle();

myTriangle.draw();

Circle myCircle = new Circle();

myCircle.draw();

Polymorphism technique is used here instead of calling the method of the object itself. In the below code, we are handling this parent object and going to call the “draw off the parent” object. Now this parent object could be an abstract class.

//Shape

Share shape = new Triangle();

shape.draw();

Shape shape = new Circle();

Shape.draw();

Instead of having the hard-coded code, we are now writing a method called myDrawMethod and this method is going to take shape as a parameter. Now this shape can draw anything such as a circle or triangle. It just calls the shape and draws now depending on what object we pass to myDrawMethod. In this example, we actually removed the dependency of the circle or triangle from this method, but still, we have to pass this object to myDrawMethod. There has to be another piece of the class which has the initialization, so somebody has to do this shape equal to the new triangle somewhere else in the class. It has to happen this way and then we need to call myDrawMethod by passing this shape. We need to have this new instantiation.

Shape shape = new Triangle();

myDrawMethod(shape);

A drawing class is there which will have a class member variable called shape. It’s not going to be a member variable triangle or member variable of the circle. Actually, it is a member variable of type shape now in the following piece of code:

//Drawing class

protected class Drawing {

private Shape shape;

public setShape (Shape shape) {

this.shape = shape;

}

}

//Different class

Trianlge myTrianlge = new Triangle();

drawing.setShape(myTrianlge);

drawing.drawShape();

A class is there which has a private shape object and now a setter is defined here which is public and sets shape object. It can accept both a circle and a triangle as it is a shape object and whatever shape is passed to this object. A method drawshape() is here which draws even it calls a draw method of shape whatever has been set. It will not do an initialization as there is no new shape here, so we are going to assume that somebody will instantiate that and provide an object to this class. Now there is no dependency from a triangle or shape and if we want to draw a triangle, we don’t have to modify the drawing class as it will be going to remain the same.

What we need to do is to pass a triangle to the setter to draw a triangle and vice versa for the circle. We are using a different class which equals new triangle and produce a new object of this drawing class. So now in our drawing class, we have an instance of a specific shape like a triangle or a circle instead of having an instance of a specific shape, it has an instance of the parent shape object. Along with that, we have a different class which has a triangle and then this class is going to pass the triangle to the drawing class. After that, the drawing class is going to have a triangle leaving this class passes a circle to the drawing class which will have a circle. It’s open for a new object to be instantiated as long as the object is a shape so it might draw a new shape.

//Shape

draw()

//Triangle

draw()

The basic concept is to modify this drawing class by passing a corresponding object in a shape and then the drawing class is going to draw it. In this example, we are separating the whole dependency out of a class in which the drawing class doesn’t really know what it’s on the dependent. The advantage of this is that if what it has to draw changes, the user doesn’t have to modify the drawing class because it doesn’t really know any specific shape. It can draw any shape and this is happening because the dependency of the drawing class to a shape object is not owned by the drawing class. There is no relationship, it just has a shape, but does not own any sort of relationship of the actual shape that is drawing. The drawing class has the dependency to the triangle is actually injected into the drawing class by a completely different class so this is the principle of dependency injection here.

Let’s see the next section about the benefits of the Spring framework in the Spring tutorial.

## Benefits of Using Spring Framework

Here I have listed following list in this Spring Tutorial of few of the great benefits of using Spring Framework:

* Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
* Spring has multiple modules in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.
* Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.
* Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.
* Spring’s web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
* Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
* Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
* Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Eclipse

Java

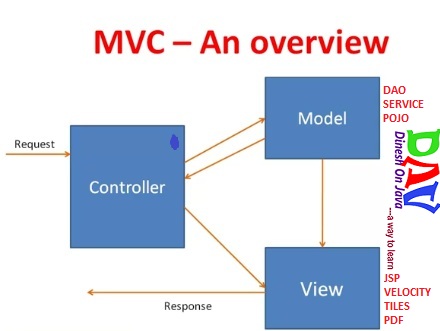
Tomcat Server Apatche Website

Spring Framework Sprig Frame work

Spring MVC

**Spring MVC (Model view controller)** is based on the MVC design pattern, it is a software architecture design pattern.  It provides solution to layer an application by separating three concerns business, presentation and control flow.

* The **Model** can be some DAO layer or some Service Layers which give some information about request or requested information or Model can be a POJO which encapsulates the application data given by the controller.
* The **View** is responsible for rendering the model data and in general it generates HTML output that the client’s browser can interpret.
* The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.



**Advantages of Spring MVC Framework-**

* Supports [RESTful URLs](http://en.wikipedia.org/wiki/Representational_state_transfer).
* Annotation based configuration(i.e. you may reduce the metadata file or less of configuration).
* Supports to plug with other MVC frameworks like Struts, Struts2, WebWorks etc.
* Flexible in supporting different view types like JSP, velocity, XML, PDF, Tiles etc.

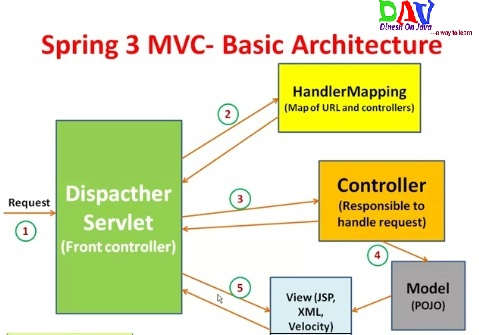
**Front Controller**

Front Controller is very important component one which route the all the requests into framework control that means when ever requests land on different controllers it queues that request to the controller of framework without this MVC framework will not may be able to take control of the request at landing at the application. So front controller is not only capture the request but also the following responsibility-

* It initialize the framework to cater to the requests.
* Load the map of all URLs and the components responsible to handle the request.
* Prepare the map for the views.

## Spring MVC Basic Architecture

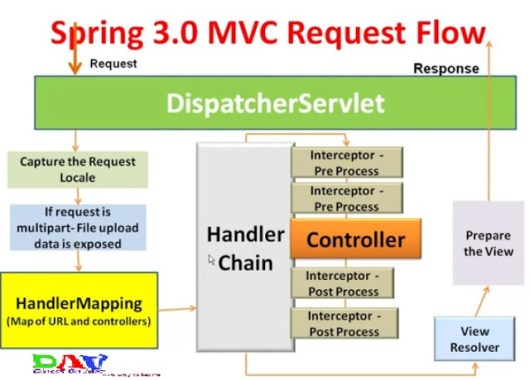
The **Spring web MVC framework** provides **model-view-controller** architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.



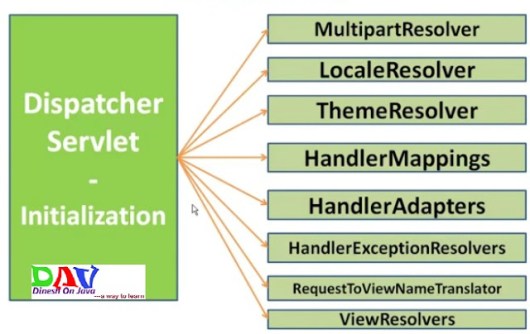
In Spring MVC framework **Dispatcher Servlet** access Front Controller which handles all coming requests and queses for forward to the different controller.  
**1.** Whenever request lands the dispatcher servlet consult with HandlerMapping  
(**HandlerMapping**– is a component which have the map of URL and Controller which need to be invoked for that particular request which lands with URL)  
**2.** then Dispatcher servlet has information about which is controller need to be invoked  
**3.** then that controller will be invoked  
**4.** and Controller can request the model for some information (about some DAO, Service layer or Data in POJO, or data in database using business logic)  
**5.** once process has been done then dispatcher servlet get the response then dispatcher servlet will get view resolver to build the view and view resolver look out what view has being configured it has been JSP, Velocity, XML etc. based this configuratin view has been prepared and the information from model i.e. POJO it will be put on the view and response will be send back to browser.

## Spring  MVC Request Flow

**1.** Request lands to Front Controller i.e. **DispatcherServlet**  
**2.** Capture the Request **Locale**i.e use for internationalization i.e Read **.properties** files  
**3.** Check for multipart-file(**MIME type header** or not) upload data from distributed application  
**4.** Consult with **HandlerMapping**for which Controller to be invoked  
**5.** and Then responsibility is given to the **Handler Chain**  
**6.** This **Handler Chain** is responsible to be invoked some of the **interceptors**that needs to be invoked before of a controller and after the controller that means interceptors are here like very to similar to the filters that help to separate the **pre-process logic** and **post-process logic**.  
**7.** After process of pre-process interceptor return to the controller process the post-process logic.  
**8.** Then return to the view resolver prepared the view based on your configuration decide the which configuration (JSP, Velocity, PDF etc.) to be invoked.  
**9.** After choosing view technology prepare the view and return the response back to the client.



## Spring MVC Framework- Initialization



**MultipartResolver:**Interface to handle the file uploads  
**LocaleResolver:**Helps to resolve the locale from the request  
**ThemeResolver:**Resolve a theme for a request(CSS)  
**HandlerMapping:**Maps the Request to Handlers (Controllers)  
**HandlerAdapter:**Plugs the other frameworks handlers  
**HandlerExceptionResolver:**Mapping of the exceptions to handlers and views  
**ViewResolver:**Maps the view names to view instances

All the above mentioned components ie. **HandlerMapping**, **Controller**and **ViewResolver**are parts of ***WebApplicationContext***which is an extension of the plain ***ApplicationContext***with some extra features necessary for web applications.

## Required Spring MVC Configuration

You need to map requests that you want the **DispatcherServlet**to handle, by using a URL mapping in the ***web.xml*** file. The following is an example to show declaration and mapping for ***spring3*** **DispatcherServlet**example:

**Step 1:- Configure the web.xml with DispatcherServlet and details of the application context file** location.

<web-app id="WebApp\_ID" version="2.4" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/j2ee" 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>spring3</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring3</servlet-name>

<url-pattern>\*.\*</url-pattern>

</servlet-mapping>

</web-app>

**Step 2:- Configure the contextConfigLocation for the application context to be loaded.**

<web-app id="WebApp\_ID" version="2.4" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/j2ee" 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>spring3</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring3</servlet-name>

<url-pattern>\*.\*</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name><param-value>/WEB-INF/spring3-servlet.xml</param-value></context-param>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

</web-app>

**Step 3:- Configure the spring3-servlet.xml**

<beans xmlns:context="http://www.springframework.org/schema/context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.springframework.org/schema/beans" 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.dineshonjava">

</context:component-scan>

<context:annotation-config></context:annotation-config>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/"></property>

<property name="suffix" value=".jsp"></property>

</bean>

</beans>

Following are the important points about **spring3-servlet.xml** file:

* The ***[servlet-name]-servlet.xml*** file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.
* The ***<context:component-scan…>*** tag will be use to activate Spring MVC annotation scanning capability which allows to make use of annotations like ***@Controller*** and ***@RequestMapping*** etc.
* The ***InternalResourceViewResolver***will have rules defined to resolve the view names. As per the above defined rule, a logical view named **hello** is delegated to a view implementation located at */WEB-INF/jsp/hello.jsp* .

***@Controller:***

* Used at the class level
* Tells the spring framework that the marked class acts as a controller.

@Controller

public class EmployeeController{

}

***@RequestMapping:***

* Can be used at the class level and method level in controllers.
* ***Argu***ments:-  
  –**URL[]**  
  –**HTTP Methods[]**-GET, POST, DELETE, TRACE, OPTIONS, HEAD, PUTS. Defaults method supported is GET  
  **-params[]-used**to check if a request parameter matches with a value and only if the conditions passes the method or controller processes the request.  
  (eg. ***@RequestMapping params=”myName=guest” )***  
  **-headers[]-**used to check if a request header matches with a value and only if the condition passes the method or controller processes the request  
  ***(eg. @RequestMapping headers=”myheader=guestHadder”)***

Next section will show you how to create your actual components ie. Controller, Model and View.

Defining a Controller

DispatcherServlet delegates the request to the controllers to execute the functionality specific to it. The @Controller annotation indicates that a particular class serves the role of a controller. The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

@Controller

@RequestMapping("/employee/\*")

public class EmployeeController{

@RequestMapping("add", method = RequestMethod.POST)

public String createEmployee(){}//appcontext/employee/add

@RequestMapping("delete", method = RequestMethod.GET)

public String deleteEmployee(){}//appcontext/employee/delete

@RequestMapping("details")//By default the method is GET

public String getEmployeeDetails(){}//- /appcontext/employee/details

}

You can write above controller in another form where you can add additional attributes in @RequestMapping as follows:

@Controller

public class EmployeeController{

@RequestMapping("add", method = RequestMethod.POST,value=/employee)

public String createEmployee(){}//appcontext/employee/add

@RequestMapping("delete", method = RequestMethod.GET,value=/employee)

public String deleteEmployee(){}//appcontext/employee/delete

@RequestMapping("details",value=/employee)//By default the method is GET

public String getEmployeeDetails(){}//- /appcontext/employee/details

}

The **value** attribute indicates the URL to which the handler method is mapped and the **method** attribute defines the service method to handle HTTP GET request. There are following important points to be noted about the controller defined above:

* You will defined required business logic inside a service method. You can call another methods inside this method as per requirement.
* Based on the business logic defined, you will create a **model** within this method. You can setter different model attributes and these attributes will be accessed by the view to present the final result. This example creates a model with its attribute “message”.
* A defined service method can return a String which contains the name of the **view** to be used to render the model.

**Creating JSP Views**

Spring MVC supports many types of views for different presentation technologies. These include – JSPs, HTML, PDF, Excel worksheets, XML, Velocity templates, XSLT, JSON, Atom and RSS feeds, JasperReports etc. But most commonly we use JSP templates written with JSTL. So let us write a simple employee view in **/WEB-INF/emp/employee.jsp**:

<html>

<head>

<title>Hello Spring MVC</title>

</head>

<body>

<h2>

${name}</h2>

</body>

</html>

Here **${name}** is the attribute which we have setup inside the Controller. You can have multiple attributes to be displayed inside your view.