SPRING FRAMEWORK

Prerequisite

Core Java

OOP Concept

JDBC

Servlet JSP

HTML/CSS/JAVASCRIPT

MYSQL

What is SPRING FRAMEWORK

Dependency Injector Framework to make java application loosely coupled

IOC :- Inversion of Control Container

Easy development of J EE Application

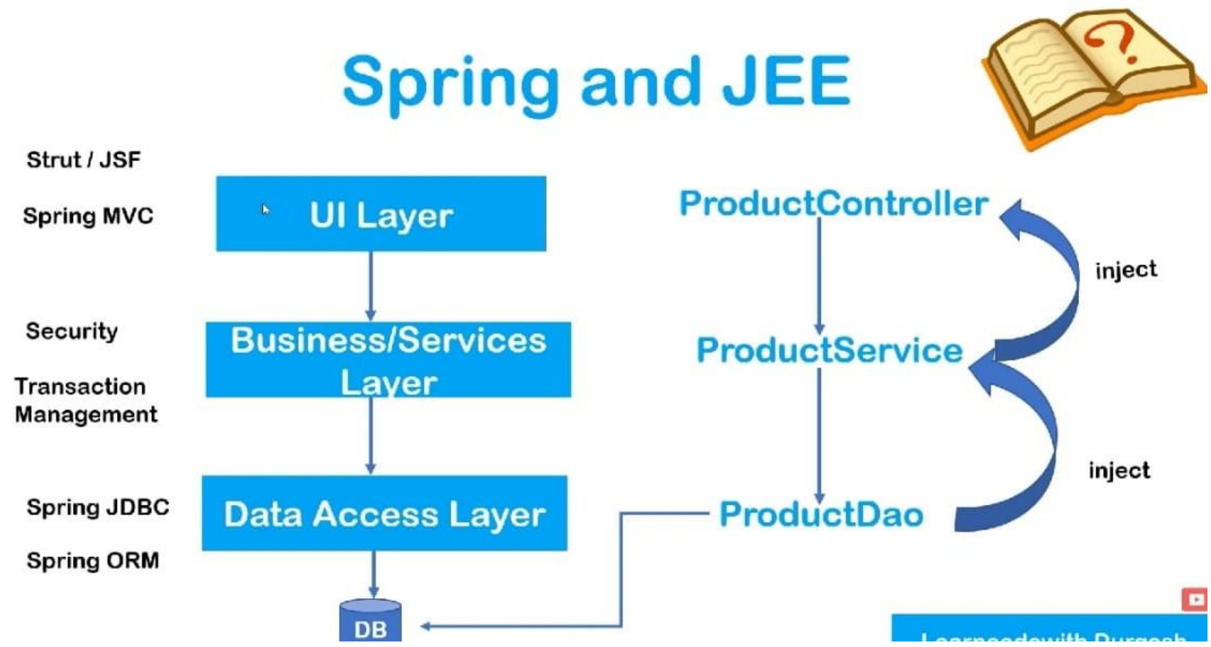
Rod Johnson in 2003

Dependency Inhection is a design pattern

Spring Framework

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts Hibernate Tapestry, EJB JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.



00:00/04:57

Inversion Of Control (IOC) and Dependency Injection

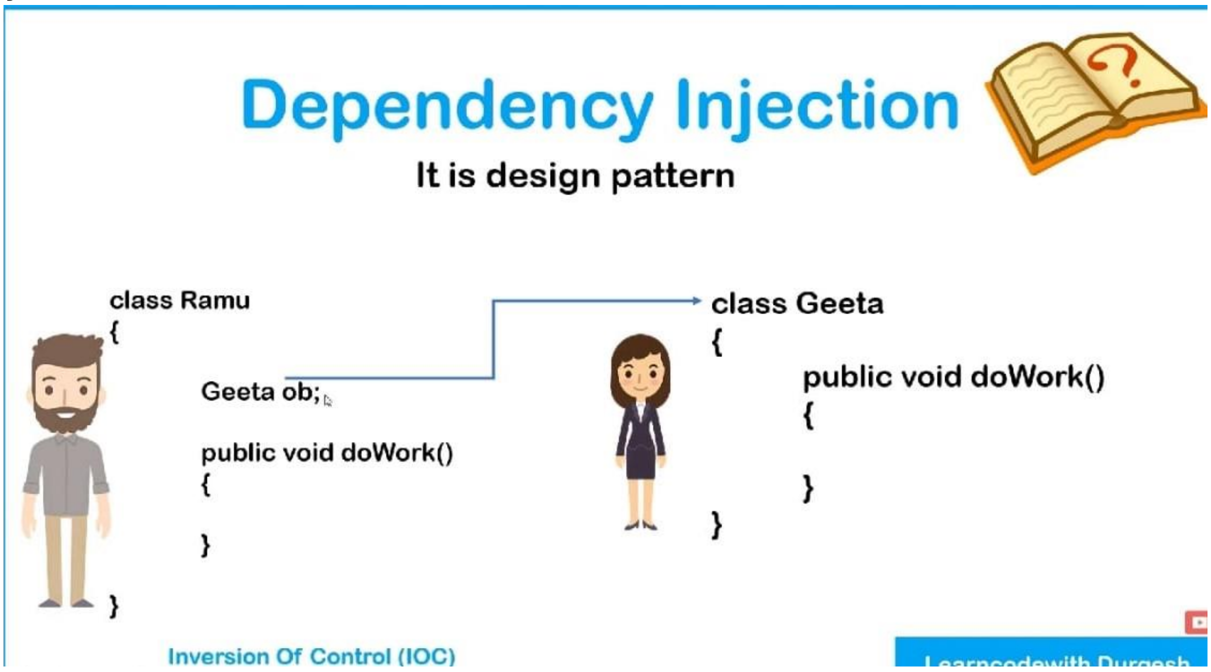
hese are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain. Let's understand this with the following code:

```
class Employee{
Address address;
Employee(){
address=new Address();
}
}
```

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

```
class Employee{
Address address;
Employee(Address address){
this.address=address;
```

5. }
6. }



Dependency Injection

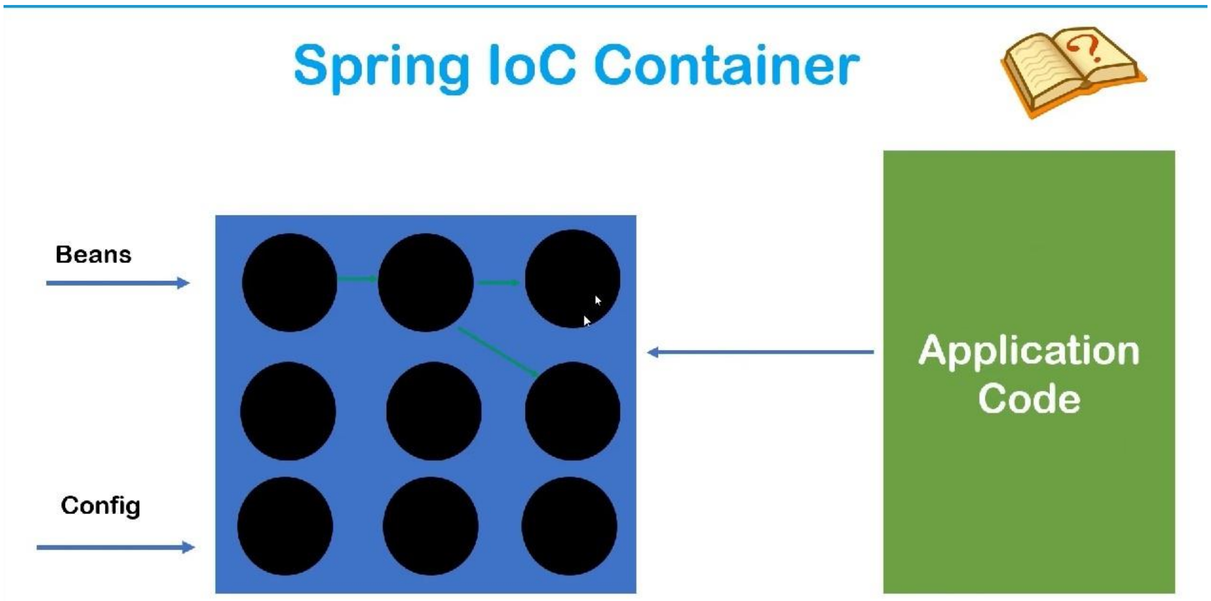
It is design pattern

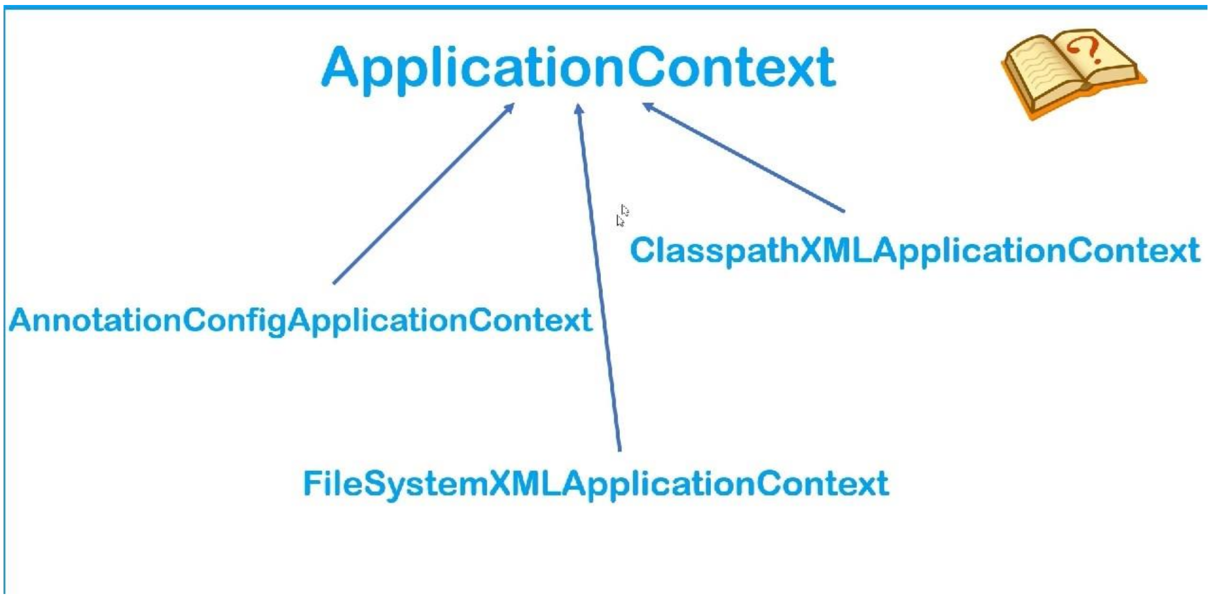
```
class Ramu
{
        Geeta ob;

        public void doWork()
        {

        }
}

class Geeta
{
        public void doWork()
        {

        }
}
```

Inversion Of Control (IOC)

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by Class path, XML file or annotation.




ApplicationContext
ClasspathXMLApplicationContext
AnnotationConfigApplicationContext
FileSystemXMLApplicationContext

Advantage of Dependency Injection

- makes the code loosely coupled so easy to maintain
- makes the code easy to test

Advantages of Spring Framework

There are many advantages of Spring Framework. They are as follows:

1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of Jdbc Template, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

4) Lightweight

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

5) Fast Development

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as JMS

, JDBC

, JPA and JTA.

7) Declarative support

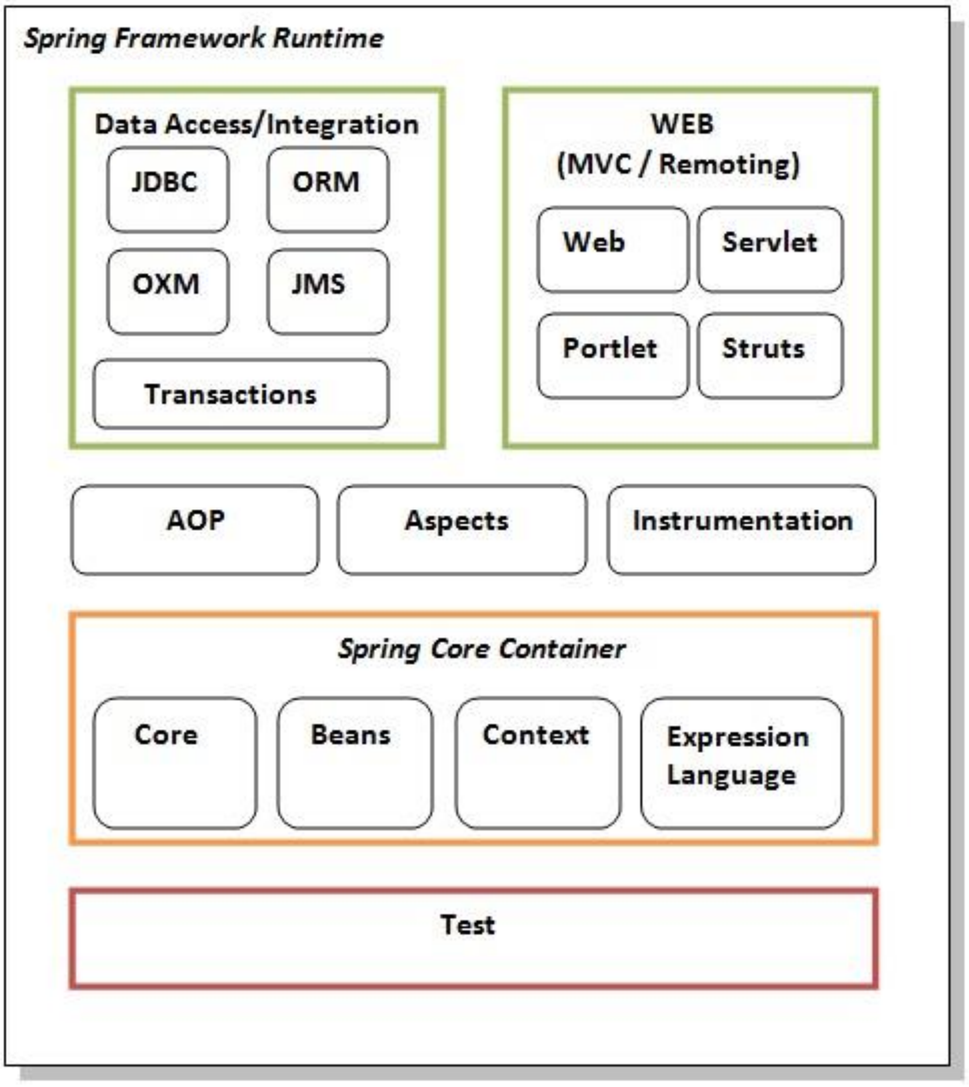
It provides declarative support for caching, validation, transactions and formatting.

Spring Modules

1. **Spring Modules**
2. **Test**
3. **Spring Core Container**
4. **AOP, Aspects and Instrumentation**
5. **Data Access / Integration**

6. Web

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



Test

This layer provides support of testing with JUnit and TestNG.

Spring Core Container

The Spring Core container contains core, beans, context and expression language (EL) modules.

Core and Beans

These modules provide IOC and Dependency Injection features.

Context

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

Expression Language

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

AOP, Aspects and Instrumentation

These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code.

The aspects module provides support to integration with AspectJ.

The instrumentation module provides support to class instrumentation and classloader implementations.

Data Access / Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

Web

This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

Spring IOC Container

It is program responsible for object creation holding them in memory and injection.

Needs beans classes, config information (dependencies)

Application code uses the beans

Application Context (Interface extending Bean Factory Class)

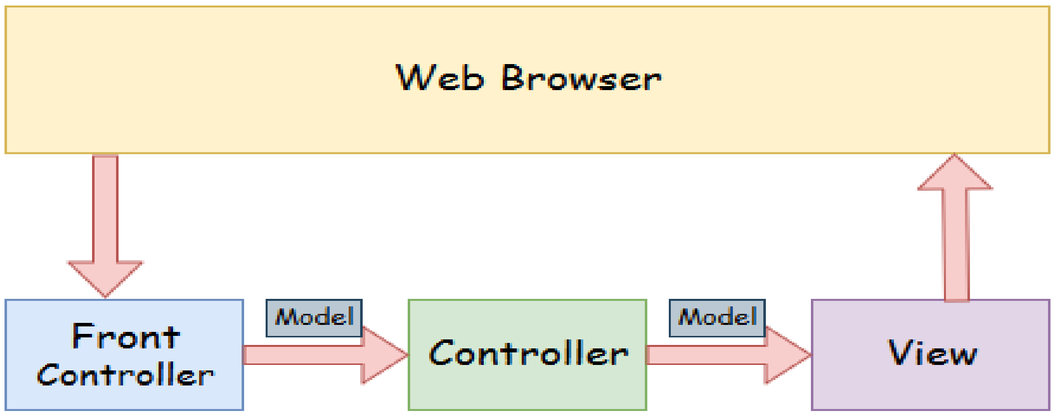
SPRING MVC

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection. • It is build on the top of Servlet API. (works on Servlet). We need Servlet class for Spring MVC.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views

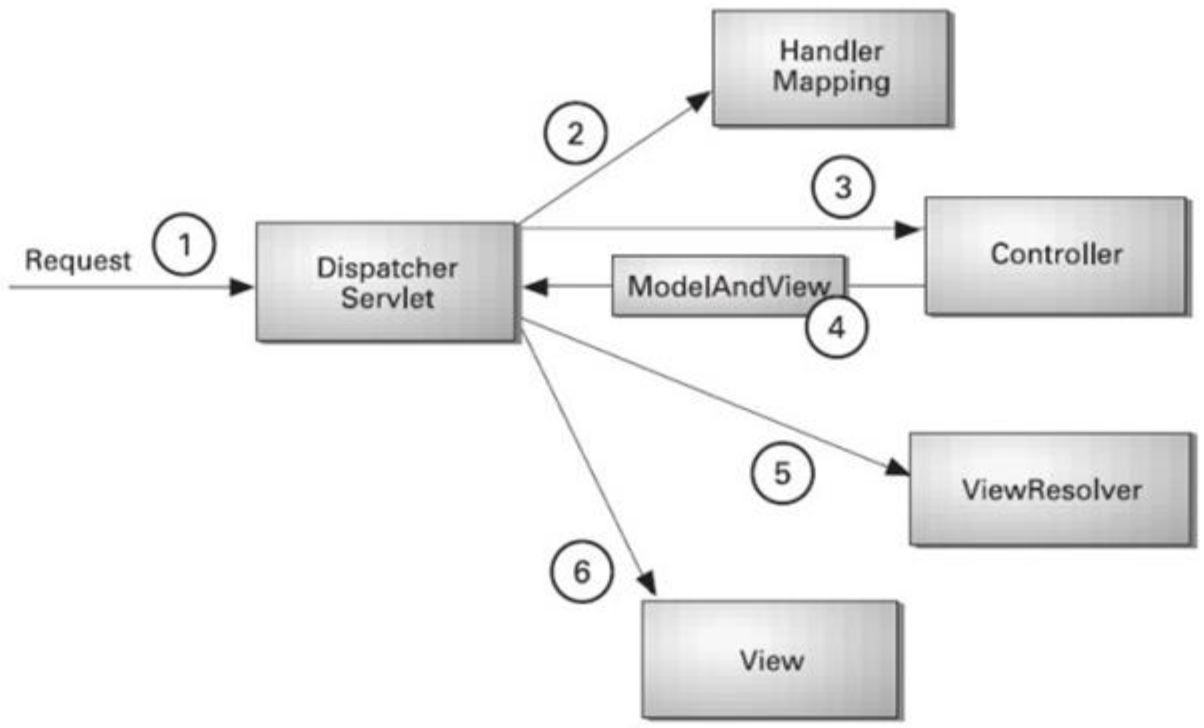
- Model – Data
- View – The final Result (Presentation of data to user) (JSP/HTML)
- Controller – Control the program :- Interface between model and view for processing the request. (Servlet)

Spring Web Model-View-Controller



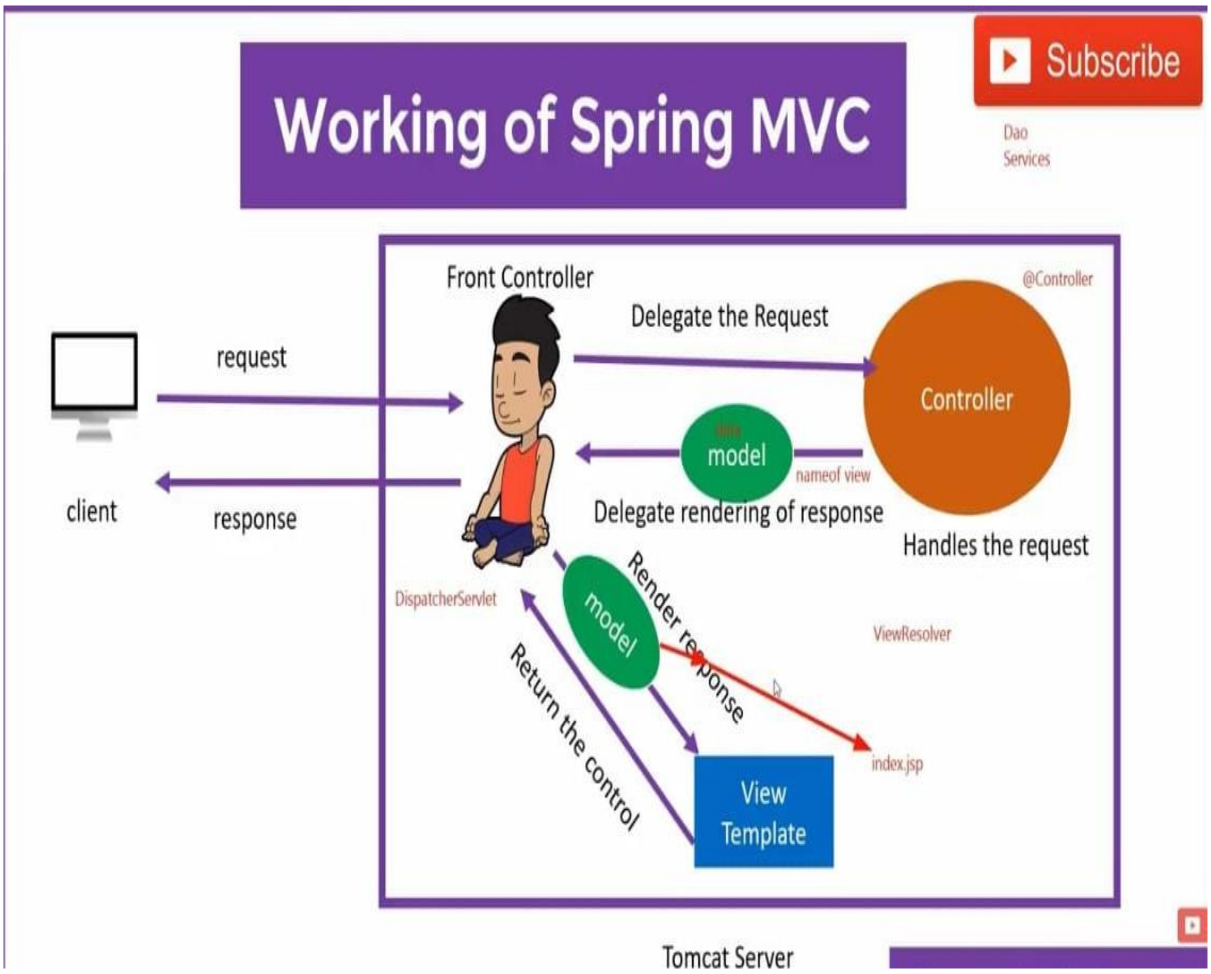
- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC



- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
- The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.

- The controller returns an object of ModelAndView.
- The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.



WHY SPRING MVC

- Supports Model View Controller Pattern
- Provides Powerful Configuration
- Implements basic features of core Spring Framework like Inversion of Control, Dependency and Injection.
- It uses loose coupling with core Spring Framework.
- It is Flexible, easy to test and has many features.
- Works on Client Server Architecture
- Way to design code.
- For more Scalability and Organised Formation of Project.

Advantages of Spring MVC Framework

Let's see some of the advantages of Spring MVC Framework:-

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

ECLIPSE IDE CONFIGURATION FOR SPRING MVC

- Create Maven Project for Spring
- Tomcat Configuration with Eclipse
- Add Maven Dependency for Spring in Eclipse

CONFIGURING SPRING MVC IN ECLIPSE

- Cofigure Dispatcher Servlet in web.xml
- Create Spring Configuration File
- Configure View Resolver
- Create Controller

- Create View to show Result

  Transferring Data from Controller to View
  1. Model
  2. Model and View

```
Public String about (Model m)
m.add Attribute(“name”, “MIT”);
List <String> friends = new ArrayList<String>();
return index;
```

```
Public ModelAndView about ()
M= new ModelAndView();
m.addObject(“name”, “Uttam”)
LocalDateTime now = LocalDateTime.now();
m.addObject(time,now)
m.setViewName(“about”);
return m;
```

Transferring Data from view to Controller
1. HttpServletRequest Object
2. @RequestParam
3. @Model

Modelling Data between View- Controller- View

There are three methods to implement this

1. **Using HttpServlet Request**
2. **Using @ RequestParam Annotation**
3. **Using @ Model Attribute**

METHOD 1 :- Using HttpServlet Request

web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>

  </servlet-mapping>
</web-app>
```

spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
     xmlns:p="http://www.springframework.org/schema/p"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
        <context:component-scan base-
package="springmvc.controller"></context:component-scan>
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResol
ver" name="viewResolver">
        <property name="prefix" value= "/WEB-INF/views/"></property>
        <property name="suffix" value= ".jsp"></property>
        </bean>
        </beans>
```

register.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>
<h1>Fill this form for Registration</h1>
<form action = "processform" method="post">
Enter Your Name:<input type="text" name="uname"></br>
Enter Your Email:<input type="text" name="email"></br>
Enter Your Password:<input type="password" name="pwd"></br>
<input type="submit" value="Register">
</form>

</body>
</html>
```

RegisterController.java

```
package springmvc.controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
@Controller
public class RegisterController {
        @RequestMapping("/register")
public String register() {

        return("register");
}
        @RequestMapping(path = "/processform",
method=RequestMethod.POST)
        public String process(HttpServletRequest request, Model m)
        {
                String username = request.getParameter("name");
                String email = request.getParameter("email");
                String password = request.getParameter("pwd");
                System.out.println("Username is : - "+username);
                System.out.println("Email is : - "+email);
                System.out.println("Password is : - "+password);
                m.addAttribute("name",username);
                m.addAttribute("email",email);
                m.addAttribute("password",password);
                return("success");

        }
}
```

success.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>

<%
String name= (String)request.getAttribute("name");
String email = (String)request.getAttribute("email");
String password = (String)request.getAttribute("password");
%>
<h1>My name is: <%=name %></h1>
<h1>My email is: <%=email %></h1>
<h1>My password is: <%=password %></h1>
```

```
</body>
</html>
```

METHOD 2 :- Using @RequestParam Annotation

web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>

  </servlet-mapping>
</web-app>
```

spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
     xmlns:p="http://www.springframework.org/schema/p"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
        <context:component-scan base-
package="springmvc.controller"></context:component-scan>
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResol
ver" name="viewResolver">
        <property name="prefix" value= "/WEB-INF/views/"></property>
        <property name="suffix" value= ".jsp"></property>


        </bean>
        </beans>
```

register.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>
<h1>Fill this form for Registration</h1>
<form action = "processform" method="post">
Enter Your Name:<input type="text" name="uname"></br>
Enter Your Email:<input type="text" name="email"></br>
Enter Your Password:<input type="password" name="pwd"></br>
<input type="submit" value="Register">
</form>

</body>
</html>
```

RegisterController.java

```
package springmvc.controller;

import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
@Controller
public class RegisterController {
        @RequestMapping("/register")
public String register() {

        return("register");
}
        @RequestMapping(path = "/processform",
method=RequestMethod.POST)
        public String process(@RequestParam("uname") String username,
                        @RequestParam("email") String email,
                        @RequestParam("pwd") String password, Model m)
        {

                System.out.println("Username is : - "+username);
                System.out.println("Email is : - "+email);
                System.out.println("Password is : - "+password);
                m.addAttribute("name",username);
                m.addAttribute("email",email);
                m.addAttribute("password",password);
                return("success");

        }
}
```

success.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>
```

```
<%
String name= (String)request.getAttribute("name");
String email = (String)request.getAttribute("email");
String password = (String)request.getAttribute("password");
%>
<h1>My name is: <%=name %></h1>
<h1>My email is: <%=email %></h1>
<h1>My password is: <%=password %></h1>

</body>
</html>
```

METHOD 3 :- Using @Model Annotation

web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>

  </servlet-mapping>
</web-app>
```

spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
     xmlns:p="http://www.springframework.org/schema/p"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
        <context:component-scan base-
package="springmvc.controller"></context:component-scan>
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResol
ver" name="viewResolver">
        <property name="prefix" value= "/WEB-INF/views/"></property>
        <property name="suffix" value= ".jsp"></property>


        </bean>
        </beans>
```

register.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>
<h1>Fill this form for Registration</h1>
<form action = "processform" method="post">
Enter Your Name:<input type="text" name="uname"></br>
Enter Your Email:<input type="text" name="email"></br>
Enter Your Password:<input type="password" name="pwd"></br>
<input type="submit" value="Register">
</form>
```

```
</body>
</html>
```

User.java

```
package springmvc.model;

public class User {
private String uname;
private String email;
private String pwd;
public String getUname() {
	return uname;
}
public void setUname(String uname) {
	this.uname = uname;
}
public String getEmail() {
	return email;
}
public void setEmail(String email) {
	this.email = email;
}
public String getPwd() {
	return pwd;
}
public void setPwd(String pwd) {
	this.pwd = pwd;
}
@Override
public String toString() {
	return "User [uname=" + uname + ", email=" + email + ", pwd=" +
pwd + "]";
}

}
```

RegisterController.java

```
package springmvc.controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import springmvc.model.User;
@Controller
public class RegisterController {
	@RequestMapping("/register")
public String register() {

	return("register");
}

@RequestMapping(path = "/processform",
method=RequestMethod.POST)
	public String process(@ModelAttribute User user, Model m)
	{
System.out.println(user);

		return("success");

	}


}
```

success.jsp

```
<%@ page isELIgnored="false" %>

<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="ISO-8859-1">
<title>Register  Page</title>
</head>
<body>

<h1>My name is: ${user.uname}</h1>
<h1>My email is: ${user.email}</h1>
<h1>My password is: ${user.pwd}</h1>

</body>
</html>
```