Spring MVC

Agenda

Objectives

At the end of this session, you will be able to:

- Understand MVC Architecture
- Understand Spring MVC Architecture and its Work flow
- Understand the various Spring MVC components
- Understand Spring Annotations
- Understand How to create a Spring Web Application

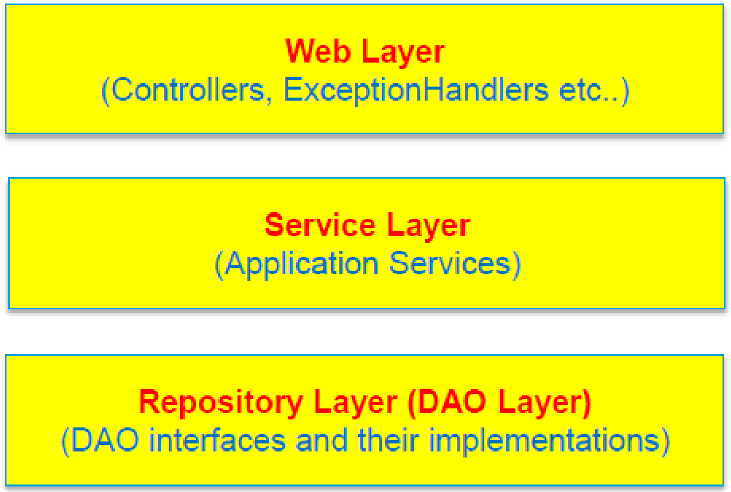
MVC Overview

MVC Overview

- MVC => Model-View-Controller
- This pattern clearly separates applications into
  - business (Model) logic,
  - presentation (View) logic and
  - navigation(Controller) logic
- Model
  - It represents the data that is transferred between the View and the controller or to its other business logic components (like services, dao)
- Controller
  - Handles navigation logic, handles the request, interacts with the service tier for business logic
  - It acts as an interface between the view and the model
- View
  - Interface to the client : What client sees and interacts with
  - Renders the response to the request
  - Pulls data from the model

MVC Overview - Separation Of Concern

- Separation Of Concern is a design principle for separating a computer application into distinct sections
- Each section addresses a separate concern
- Separation of concern helps us in identifying these layers and responsibilities of each layer
- A typical Web Application High Level Architecture may look like this


Web Layer
(Controllers, ExceptionHandlers etc..)
Service Layer
(Application Services)
Repository Layer (DAO Layer)
(DAO interfaces and their implementations)

Spring MVC

Spring MVC

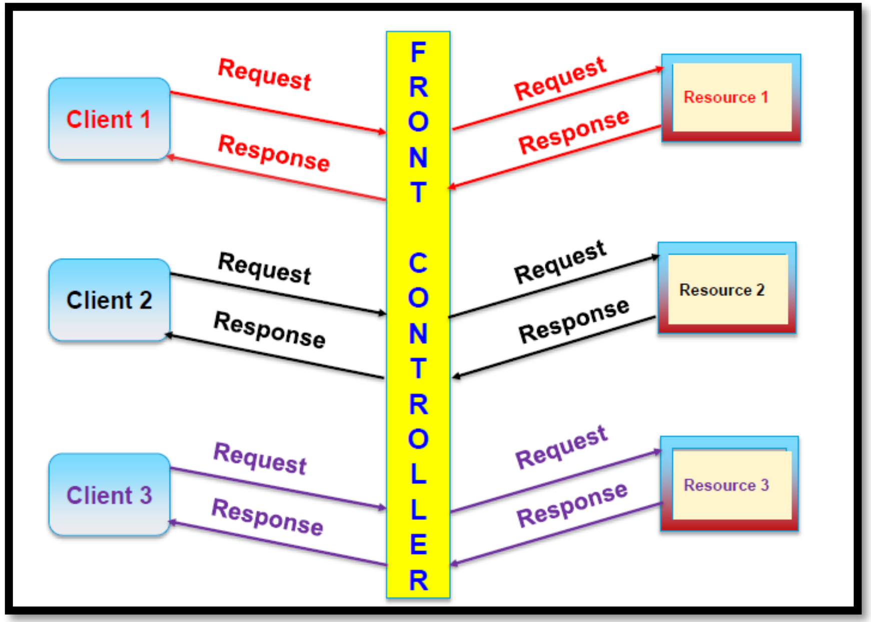
- **Spring MVC** is framework to develop **web application** based on MVC guidelines and with all Spring benefits like IOC

Spring MVC = Web Application + MVC Guidelines + Spring Benefits

- Spring has clear separation of Concern, where each role could be achieved by a specific object like
  - Dispatcher- Servlet, Handler-mapping, Controller, Model Object, Form Object, View Resolver, Validator etc.
- ***Spring supports different view technology*** like JSP, JSTL, Velocity, Thymeleaf, FreeMarker etc.
- Model transfer or ***data transfer is flexible***, it supports easy integration with any view technology
- Spring MVC's container ***WebApplicationContext*** by default gives all bean objects the scope of HttpSession (i.e. ***Bean Objects*** are ***session scoped objects***)

Front Controller Design

- Front controller design pattern is used to provide a centralized request handling mechanism
- This ensures that all requests will be handled by a single handler
- This can be used do the authentication, authorization, logging or tracking of request and then pass the requests to corresponding handlers

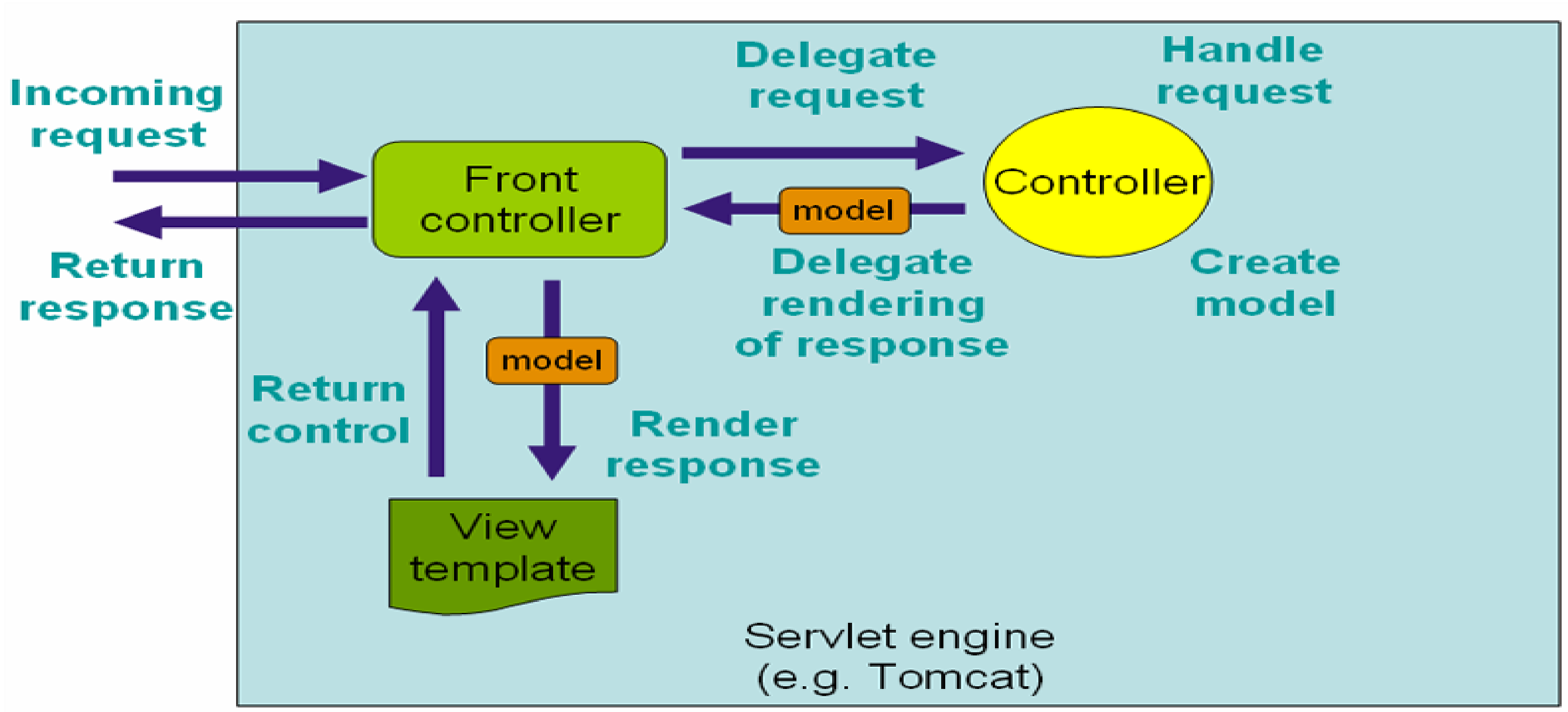


Work Flow in Spring MVC

And Lifecycle of a request in Spring MVC

Work Flow in Spring MVC

A High Level work flow of Spring MVC



Work Flow in Spring MVC

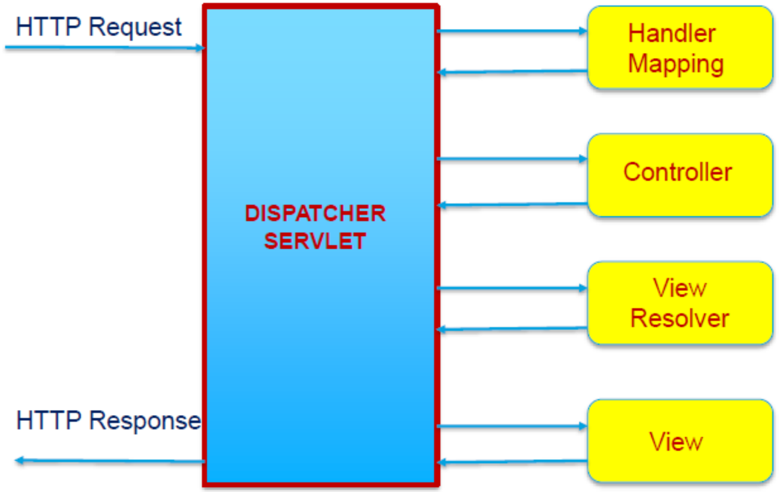
- Spring follows FrontController design pattern
- A servlet class called DispatcherServlet is an implementation of FrontController in Spring to which initially all requests are mapped
- Any request to the Spring MVC application passes through the FrontController (i.e. DispatcherServlet)
- DispatcherServlet provided by the Spring API which in turn sends calls to the respective handler method mapping to the request
- Specific Requests are handled as methods and these handler methods are mapped to the request using *@RequestMapping* Annotation
- The class containing the handler methods are called as controller written as POJO class Annotated with *@Controller*
- Request routing is completely controlled by the Front Controller i.e. DispatcherServlet in Spring

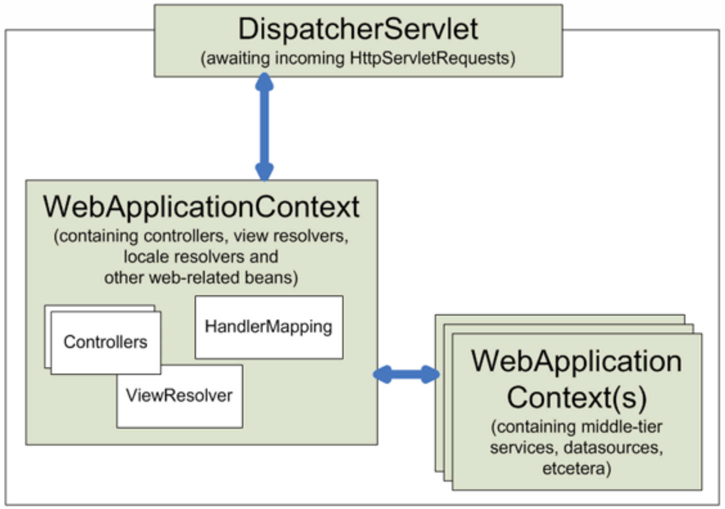
Lifecycle of a request in Spring MVC

- A request leaves the browser asking for a particular URL and optionally with request parameters
- The request is first examined by the DispatcherServlet
- DispatcherServlet consults handler-mappings defined in a configuration file (Spring Configuration file)
- It selects an appropriate controller and delegates to it to handle the request
- The controller applies appropriate logic to process the request which may in most times results in some information or data to be return back to the page
- This information or data in called the model
- Model is associated with the logical name of the result page or rendering entity called view
- The whole (model data and result page's logical name) is returned as a ModelAndView object along with the request back to DispatcherServlet

Lifecycle of a request in Spring MVC (Contd.).

- DispatcherServlet then consults the logical view name with a view resolving object to determine the actual view
- DispatcherServlet delivers the model and request to the view implementation which renders an output and sends it back to the browser.




DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)

Core Components of Spring MVC

And Lifecycle of a request in Spring MVC

Core Components of Spring MVC

- DispatcherServlet
  - *Spring's Front Controller* implementation
  - Any request that comes to the application is first interfaced with the DispatcherServlet
- Controller
  - *User created component* for handling requests
  - Encapsulates navigation logic
  - Delegates requests to the service objects for business logic implementation
  - Handler method returns the Model object and the logical view name to render it
- Model
  - Data Object that is transmitted between the View, Business logic and Controller
- ViewResolver
  - Used to map logical View names to actual View implementations

Core Components of Spring MVC

- View
    - Responsible for rendering output
    - Can be designed using any view technologies like simple JSP pages
- ModelAndView
  - The **ModelAndView class** is simply a container to hold the model data to be transported and the logical view name
  - The Object is created and returned by the handler method of the Controller
  - ModelAndView helps in returning both the Model and the view in one return by the controller

First Spring MVC Application

And Lifecycle of a request in Spring MVC

First Spring MVC Application

- Let us understand how to create a simple Spring MVC application
- First we’ll create the app to request for the simple page
- Next we’ll enhance the same to display our dynamic message on the rendered page

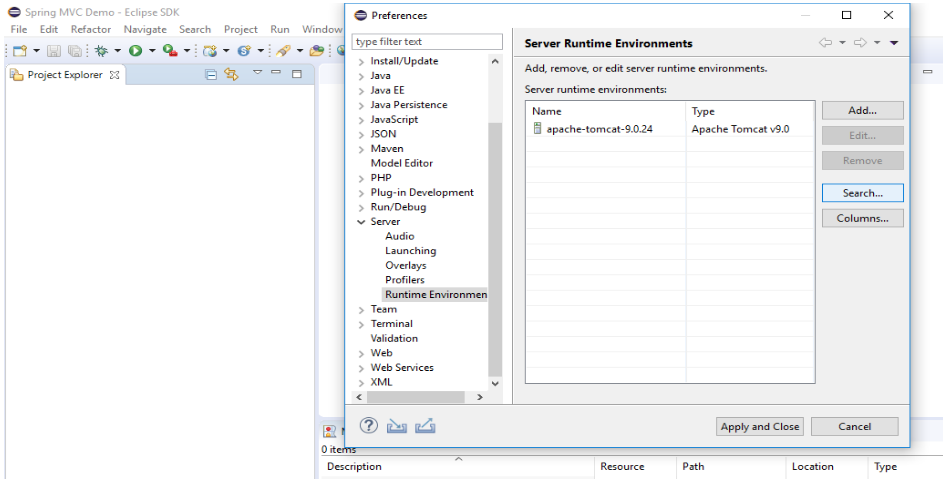
- What is needed?
  - 2 JSP Pages, index page and welcome page
  - Simple Java class to represent user defined controller (POJO)
  - Configuration details to instruct handler mapping
  - And Spring jars for bringing all together

- We would create Maven Projects to take care of the Spring dependencies
  - i.e. Required Spring jars and its compatibilities between them

First Spring MVC Application

- Let us set up the eclipse environment with WebServer (Apache Tomcat Server 9)
- Download Apache Tomcat server 9 and unzip the same

  https://tomcat.apache.org/download-90.cgi
- Add the tomcat server to eclipse from Windows Preferences
  - Server -> Runtime Environments -> search



Server Runtime Environments

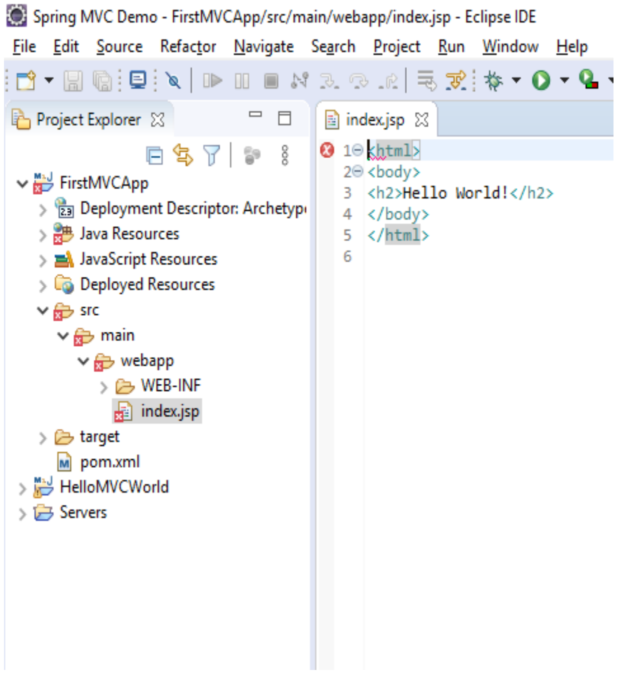
Add, remove, or edit server runtime environments.

Server runtime environments:

Name	Type
apache-tomcat-9.0.24	Apache Tomcat v9.0

First Spring MVC Application

- Create a **Maven Project**
  - On first page choose *Use default workspace* and click Next
  - Under Archetype choose *maven-archetype-webapp* and click Next
  - Fill the Maven Project details
    - Group ID :: com.wipro
    - Artifact ID :: FirstMVCApp
    - Click finish
  - It takes little time and *generates Spring Web Project*
  - The initial error is because the build path is not aware of servlets and JSP APIs
  - To set it Right click on project -> Properties -> Java Build Path -> Add Library
  - Server Runtime -> Apache Tomcat -> Finish


Spring MVC Demo - FirstMVCApp/src/main/webapp/index.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.jsp
FirstMVCApp
Deployment Descriptor: Archetyp
Java Resources
JavaScript Resources
Deployed Resources
src
main
webapp
WEB-INF
index.jsp
target
pom.xml
HelloMVCWorld
Servers

```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

First Spring MVC Application

- Next set the *dependencies in the POM* file
  - Open POM.xml file
  - Add the following spring dependencies under the tag <dependencies>

```xml
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-core</artifactId>
	<version>5.2.1.RELEASE</version>
</dependency>

<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-web</artifactId>
	<version>5.2.1.RELEASE</version>
</dependency>

<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-webmvc</artifactId>
	<version>5.2.1.RELEASE</version>
</dependency>
```

First Spring MVC Application

- Configure **DispatcherServlet** in web.xml and establish URL mappings
- Edit Web.xml file to include the below script
- This acts as the front controller
- In url-pattern we have just the */*(slash) to indicate any request to this app should be directed to the dispatcher

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>dispatcher</servlet-name>
      <url-pattern>/</url-pattern>
</servlet-mapping>
```

First Spring MVC Application

- Next create the **Spring configuration** metadata in a configuration file named <servlet-name>-servlet.xml in the WEB-INF directory of your web application
- i.e. File name has to be the dispatcher servlet name given in the web.xml file post fixed with -servlet.xml
- In our case its dispatcher so the file name would be dispatcher-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"
  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">

  </beans>
```

- Note here we have added the spring-bean.xsd and spring-context.xsd

First Spring MVC Application

- In the Spring configuration file in between the <beans> tag we add the following script

```
<context:component-scan base-package="com.wipro" />
```

- When the request is received by the DispatcherServlet, it checks the Spring configuration file (i.e. dispatcher-servlet.xml) for a mapping of the specified URL to a user defined controller bean
- <contect:component-scan>is used to indicate the location were the required components like User defined Controller beans and Service beans can be looked up for

First Spring MVC Application

- Next script to be added in the Spring configuration file is

```
<!-- Including the bean for View Resolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     <property name="prefix">
          <value>/WEB-INF/views/</value>
     </property>
     <property name="suffix">
          <value>.jsp</value>
     </property>
</bean>
```

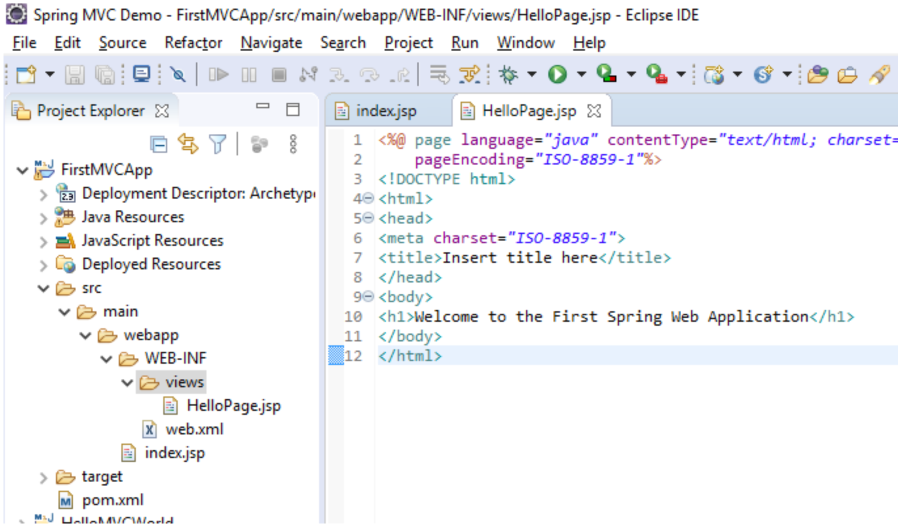
- Logical View name returned by the handler methods is resolved by the view resolver bean class called **InternalResourceViewResolver**
- DispatcherServlet looks for a **view resolver** to resolve the logical view name that it got from the ModelAndView object
- In our case all views except index.jsp has to be created under the *prefix value*

First Spring MVC Application

- Add the following code in Index.jsp to request for a new page called *HelloPage.jsp*

```
<a href="Hello">Click here </a>
```

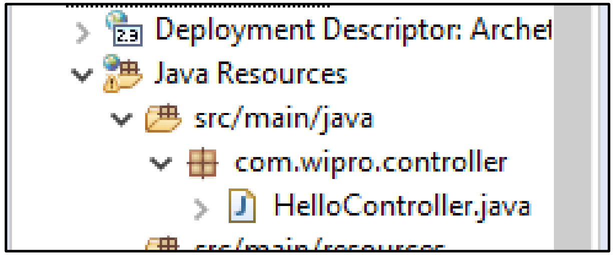
- Create a new page called HelloPage.jsp under WEB-INF/views


Spring MVC Demo - FirstMVCApp/src/main/webapp/WEB-INF/views/HelloPage.jsp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
index.jsp
HelloPage.jsp
FirstMVCApp
Deployment Descriptor: Archetyp
Java Resources
JavaScript Resources
Deployed Resources
src
main
webapp
WEB-INF
views
HelloPage.jsp
web.xml
index.jsp
target
pom.xml
HelloMVCWorld

```
<%@ page language="java" contentType="text/html; charset=
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to the First Spring Web Application</h1>
</body>
</html>
```

First Spring MVC Application

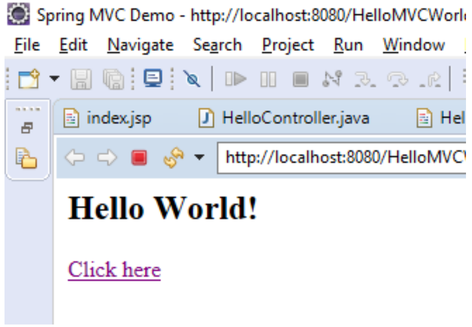
- Let us create the Controller to handle the Hello request and Run the Application



Deployment Descriptor: Archet
Java Resources
src/main/java
com.wipro.controller
HelloController.java
src/main/resources

```
@Controller
public class HelloController {

    @RequestMapping("/Hello")
    public String helloMethod() {
    return "HelloPage";
    }
}
```

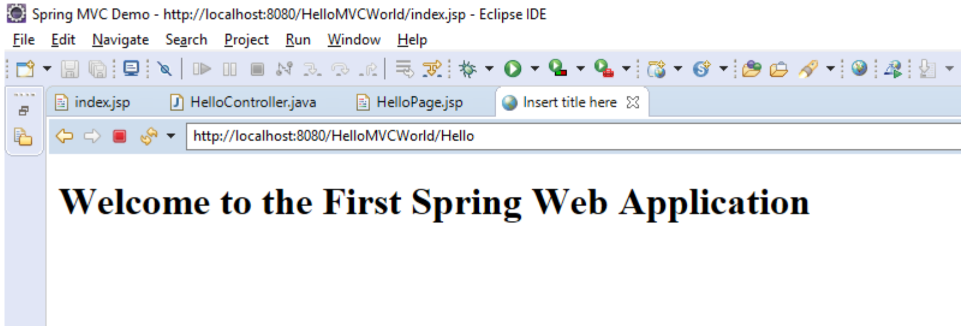


Spring MVC Demo - http://localhost:8080/HelloMVCWorl

http://localhost:8080/HelloMVC

Hello World!

Click here



Spring MVC Demo - http://localhost:8080/HelloMVCWorld/index.jsp - Eclipse IDE

http://localhost:8080/HelloMVCWorld/Hello

Welcome to the First Spring Web Application

Spring Annotations

And Lifecycle of a request in Spring MVC

Spring Annotations

- Spring has adopted Annotations in a very effective way like to describe the bean configuration, bean wiring etc.
- Some Spring Core Annotations are
- **@Autowired** – used for Dependency Injection, It injects the object dependency implicitly
- **@Configuration** – used for defining Spring Configuration as a java class instead of XML
- **@Bean** – used for defining bean class in Spring Configuration class

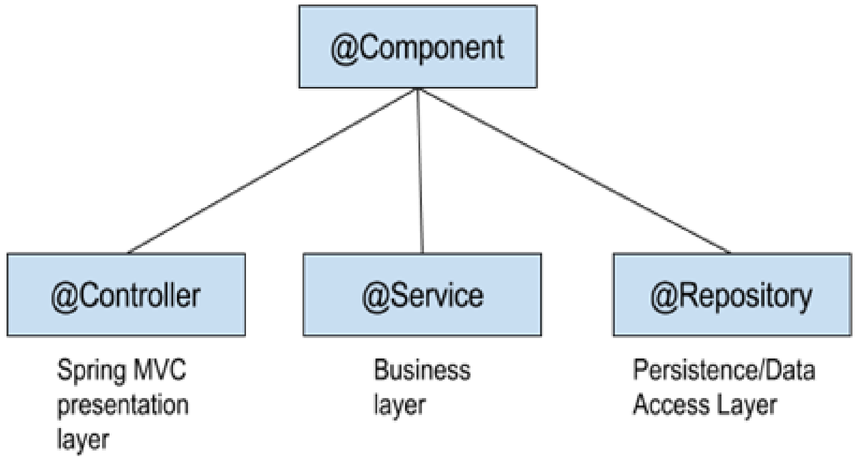
- All **beans** in Spring are Components and they are annotated as **@Component**
- Based on the **special purpose** of the bean there are specific Annotations to mark them like **@Controller, @Service or @Repository** which are *inherited from @Component*
- These are classified as Spring Stereotype Annotations
- These Annotations help in auto detect of the bean classes using classpath specified by @ComponentScan or <context:component-scan>

Spring Annotations

@Controller
Represents the class as a Controller class
This is part of Spring MVC Application

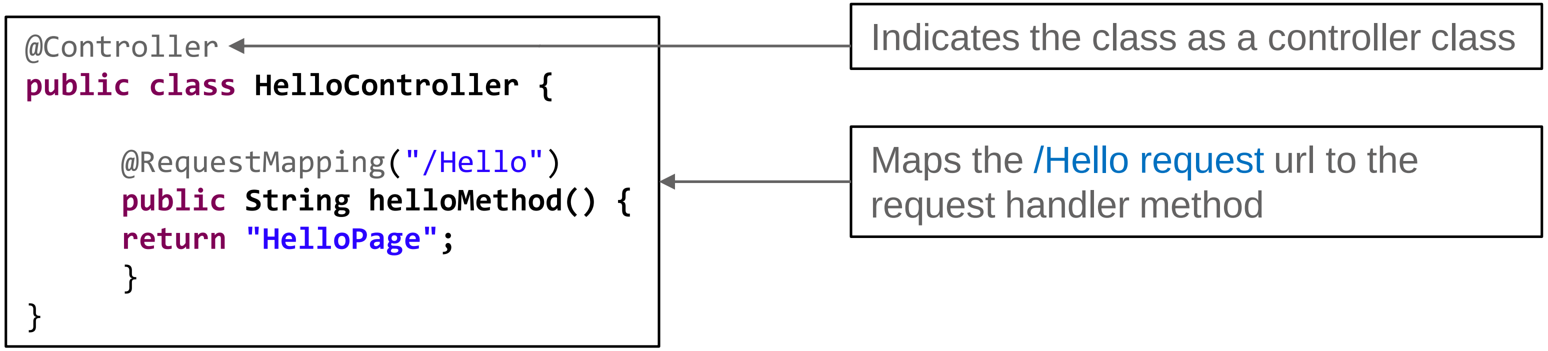
@Service
Represents the class as a Business Layer Class which contains business logic of the application

@Repository
Represents the class as a Database/Persistence Layer Class



Spring MVC Annotations

- Some Important Annotations of Spring MVC are
  - @Controller – To create Controller beans, thus the class doesn't have to explicitly implement the basic Controller Interface or inherit any specific Controller classes
  - This removes the direct dependency on the Servlets
  - @RequestMapping – maps the request URL to the method which handles it
  - This gave way to have more flexible method signatures for handler methods
  - @ModelAttribute – to bind the Model object as method attribute or return object



```
@Controller
public class HelloController {

      @RequestMapping("/Hello")
      public String helloMethod() {
      return "HelloPage";
      }
}
```

Indicates the class as a controller class

Maps the /Hello request url to the request handler method

Spring MVC Application 2

And Lifecycle of a request in Spring MVC

Spring MVC Application 2

- Let us modify the previous demo to return a “Hello World” message back from the Controller

```
@RequestMapping("/Hello")
public String helloMethod(){
return "HelloPage";
}
```

- This method only returns a String which is the logical view name
- To return both data (Hello World message) and the logical View name we use the object of a Class called **ModelAndView**

- Edit the index.jsp to add new request

```
1. <a href="Hello">Welcome Demo </a>
2. <a href="HelloWorld">Demo with data </a>
```

- Edit the controller class to add new method for the HelloWorld request

Spring MVC Application 2

- As we have seen The **ModelAndView class** is a container to hold both the model data and the logical view name
- After Spring MVC 2.5 with Annotations creating a handleRequest method is very simple with combination of *@RequestMapping and ModelAndView*

```
@RequestMapping("/HelloWorld")
public ModelAndView sayHello() {
String msg = "Hello World";
ModelAndView mv = new
ModelAndView("FirstPage","myMessage", msg);
return mv;
}
```

- Here @RequestMapping is mapped to the URL “/HelloWorld”
- Return type of the method is ModelAndView object
- Which binds the model data named myMessage with value from String called msg and the redirected page called FirstPage for the view resolver

Spring MVC Application 2

- To Present the model data on to the view i.e. in JSP Page
- Here we are expressing the Spring Model data on the JSP page as Expression Language
  - By giving the variables in between curly braces {} refixed with $ sign i.e. pattern ${..}

```
<h2>This is the message from Model :: ${myMessage} </h2>
```

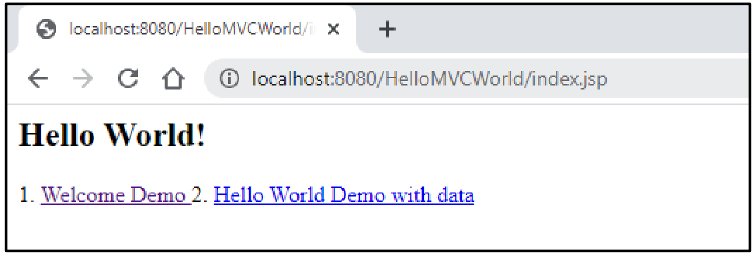
- With Servlet version 2.4 and above EL evaluation is by default actvated
- If in case JSP does not evaluate the EL we can activate it by using the Page directive property isELIgnored

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

- Setting the isELIgnored property to false activates the EL

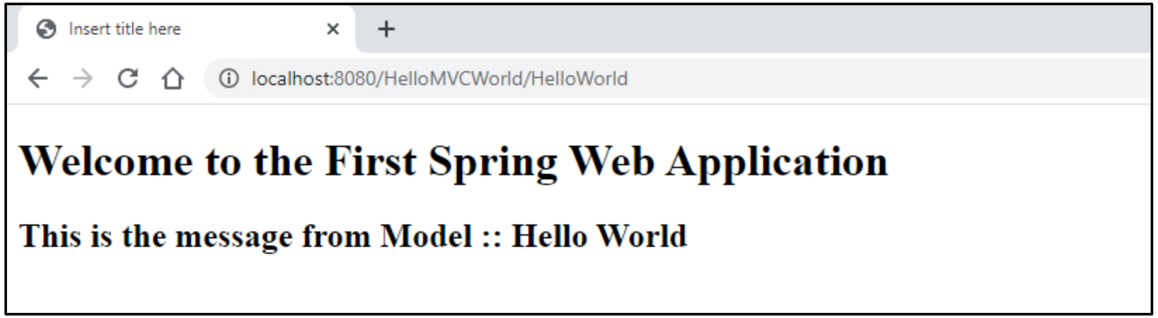
Spring MVC Application 2

- Executed Result of the Application


localhost:8080/HelloMVCWorld/i
localhost:8080/HelloMVCWorld/index.jsp

Hello World!

1. Welcome Demo 2. Hello World Demo with data



Welcome to the First Spring Web Application

This is the message from Model :: Hello World

Summary

- In this module, we have learnt :

  - **MVC Architecture**
  - **Spring MVC understanding**
  - **Core Components of Spring MVC**
  - **Flow of Request in Spring MVC based Web Application**
  - **How to develop a Web Application using Spring MVC**

Thank you