Java EE Programming

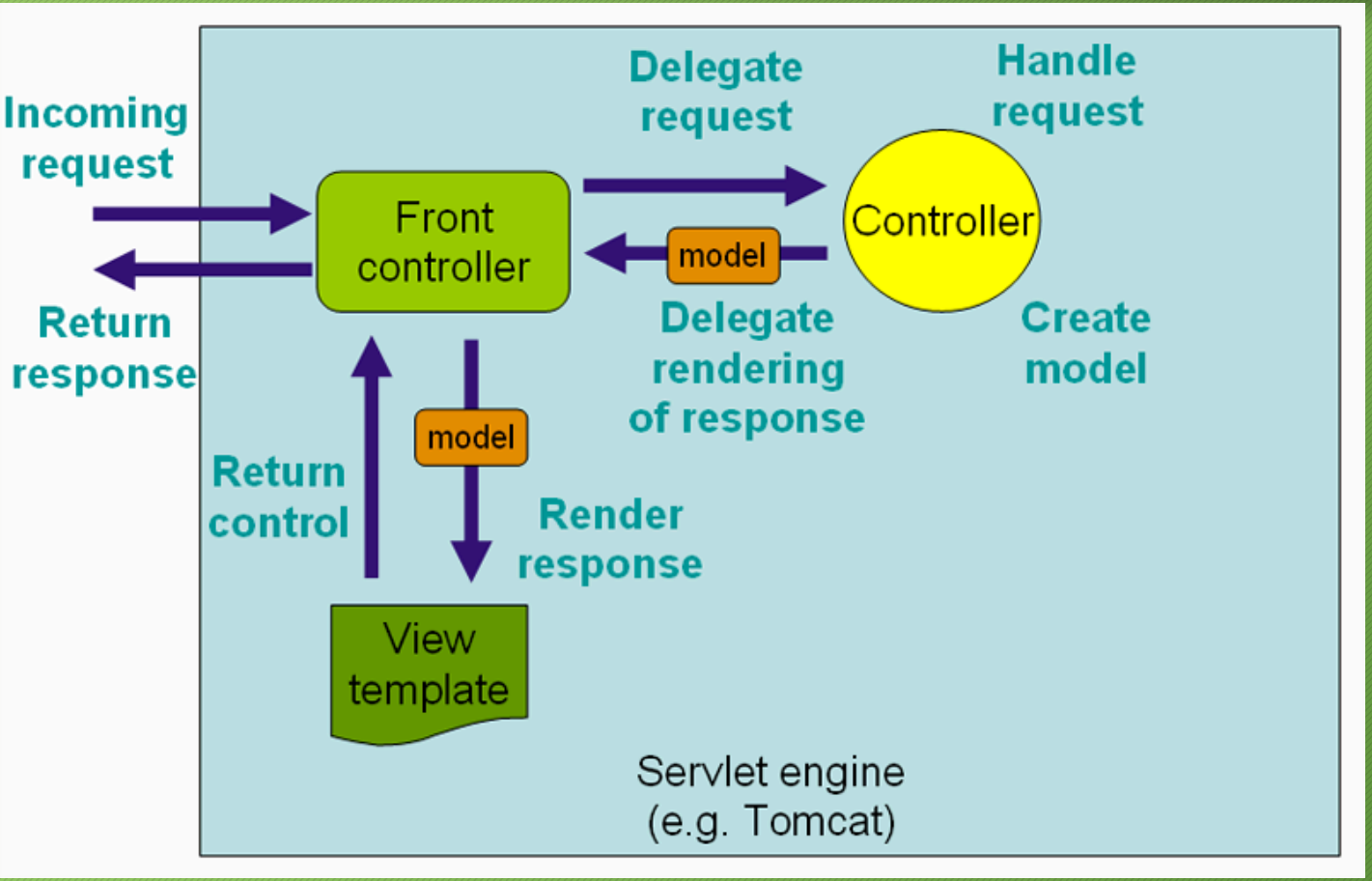
COMP 303

Lecture 4.1: Spring MVC

Spring MVC

- stands for Model View and Controller
- separates the business logic, presentation logic and data
- **Model** - The model represents data and the rules that govern access to and updates of this data.
- **View** - The view renders the contents of a model. It specifies exactly how the model data should be presented.
- **Controller** - The controller translates the user's interactions with the view into actions that the model will perform.

Spring MVC Framework



Docs.spring.io

Why need Spring MVC?

- A single request will result in multiple substantially different looking results.
- You have a large development team with different team members doing the Web development and the business logic.
- You perform complicated data processing, but have a relatively fixed layout.
- Implementing MVC with the builtin RequestDispatcher works very well for most simple and moderately complex applications
- MVC totally changes your overall system design
  - You can use MVC for individual requests

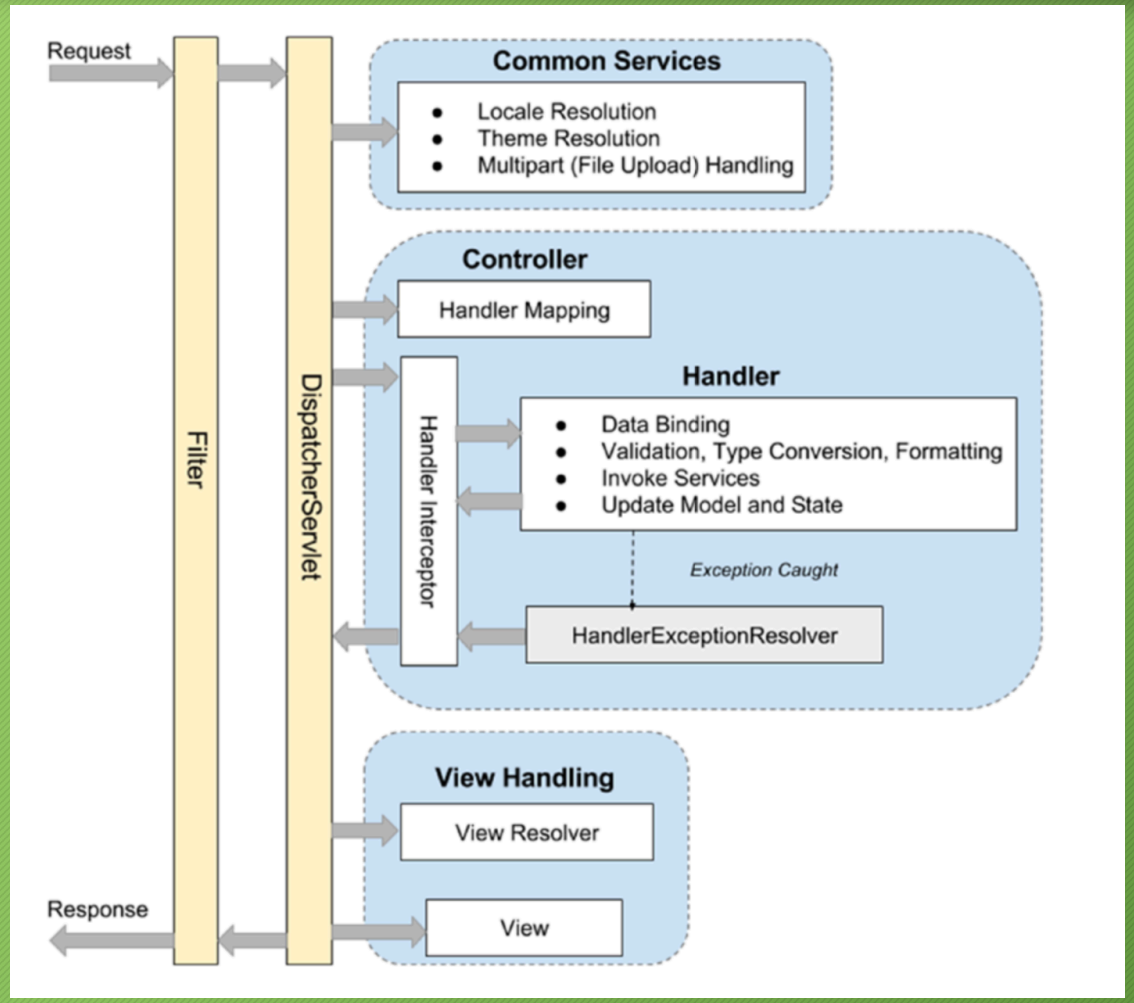
Spring MVC web support features

- Clear separation of roles - controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, and view resolver.
- Powerful and straightforward configuration of both framework and application classes.
- Reusable business code, no need for duplication.
- Customizable binding and validation.
- Flexible model transfer
- A simple powerful JSP tag library known as the Spring tag library.

Interaction among MVC Components

- The view registers as a listener on the model. Any changes to the underlying data of the model immediately result in a broadcast change notification, which the view receives.
- The controller is bound to the view. This typically means that any user actions that are performed on the view will invoke a registered listener method in the controller class.
- The controller is given a reference to the underlying model.
- The view calls the appropriate method on the controller.
- The controller accesses the model, possibly updating it in a way appropriate to the user's action.

Spring MVC Request Life Cycle



Web Module

- A web module has a specific structure. The top-level directory of a web module is the **document root** of the application.
- The document root contains a subdirectory named WEB-INF, which can contain the following files and directories:
  - **classes**: A directory that contains server-side classes: java classes, enterprise bean class files, utility classes, and JavaBeans components
  - **lib**: A directory that contains JAR files that contain enterprise beans, and JAR archives of libraries called by server-side classes
  - Deployment descriptors, such as **web.xml** (the web application deployment descriptor) and dispatcher-servlet.xml (servlet file)
- You can also create application-specific subdirectories (that is, package directories) in either the document root or the WEB-INF/views/ directory.

web.xml


```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">

  <display-name>SpringMVC</display-name>
        <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
         <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

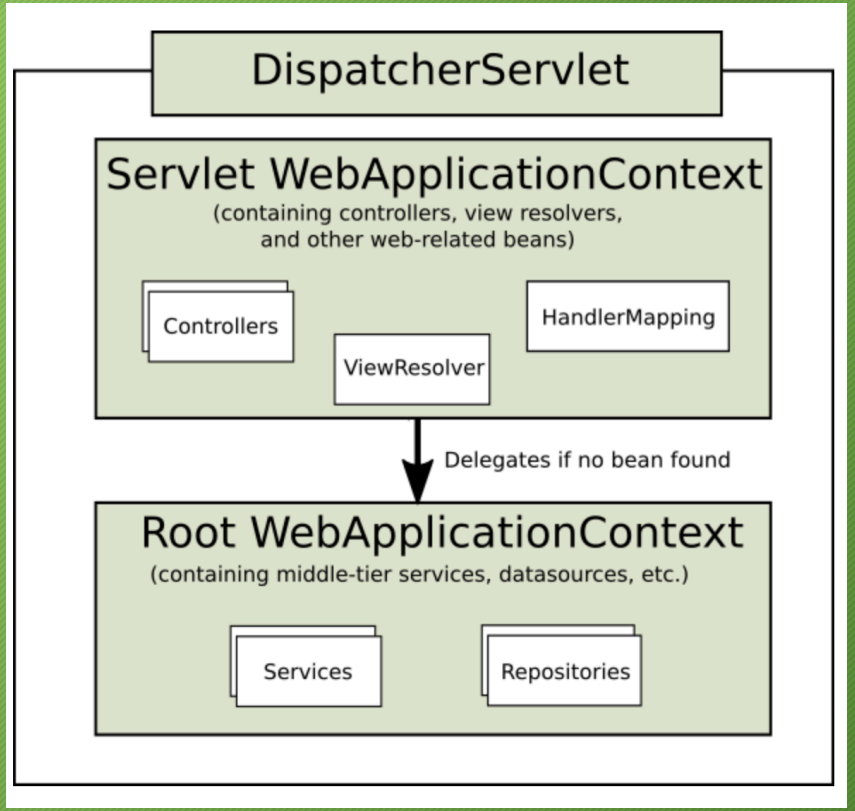
Configuration Files in Spring MVC

- POM.xml
- web.xml
- dispatcher-servlet.xml

Dispatcher Servlet

- provides a shared algorithm for request processing.
- It declared and mapped according to the Servlet specification in **web.xml**

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

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



Spring.io

Web Controller

- In Spring MVC, HTTP requests are handled by a controller using the **@Controller** annotation.
- Controller is concise and simple, acts as a stereotype for the annotated class.

```
@Controller
public class EmployeeController {
 @RequestMapping("/welcome")
public ModelAndView printMessage(
}
```

- @RequestMaping annotation ensures that HTTP GET requests to /welcome are mapped to the printMessage() method.

ModelAndView

- Holder for both Model and View in the web MVC framework.
- This class merely holds both to make it possible for a controller to return both model and view in a single return value.
- Represents a model and view returned by a handler, to be resolved by a DispatcherServlet.
- The view can take the form of a String view name which will need to be resolved by a ViewResolver object;
- alternatively a View object can be specified directly.
- The model is a Map, allowing the use of multiple objects keyed by name.

Request Mapping

- **@RequestMapping** annotation is used to map requests to controllers methods.

```
@RequestMapping("/welcome")
public ModelAndView printMessage() ……..
```

- HTTP method specific shortcut variants of @RequestMapping:
  - @GetMapping
  - @PostMapping
  - @PutMapping
  - @DeleteMapping
  - @PatchMapping
- By default **@RequestMapping** matches to all HTTP methods.

Request Param

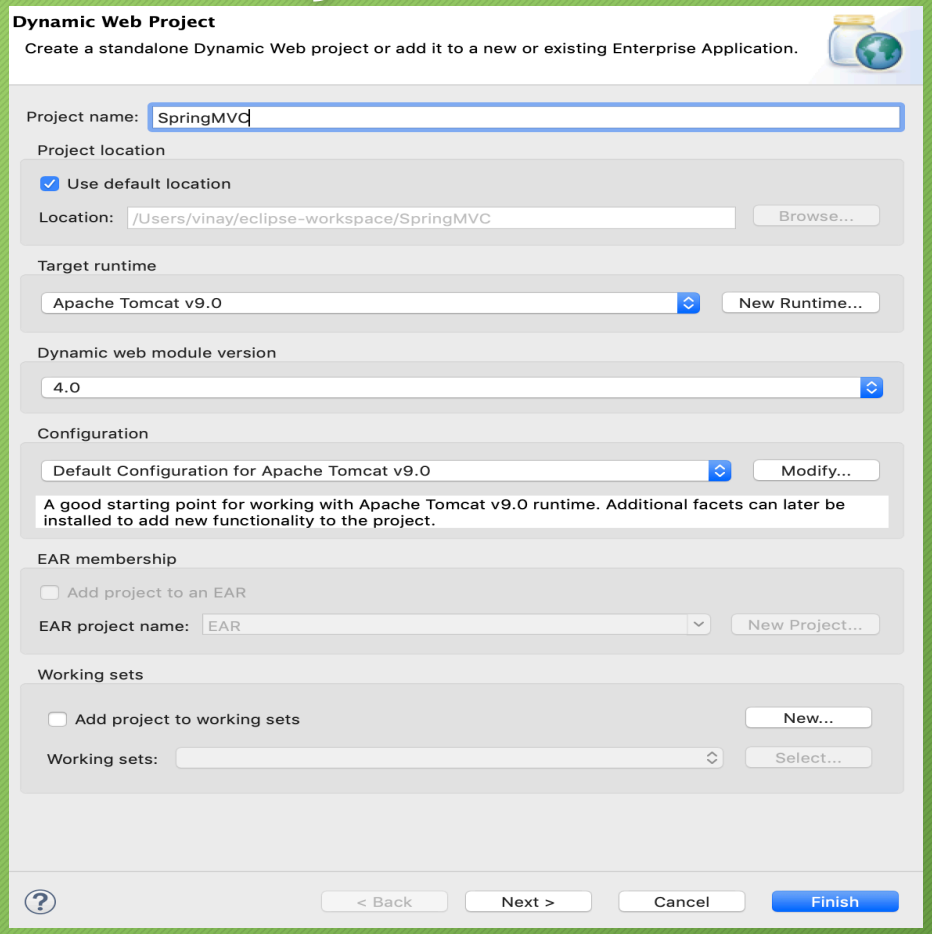
- **@RequestParam** annotation to bind Servlet request parameters to a method argument in a controller.

```
@RequestParam(value = "name", required = false, defaultValue = "Hi") String name)
```

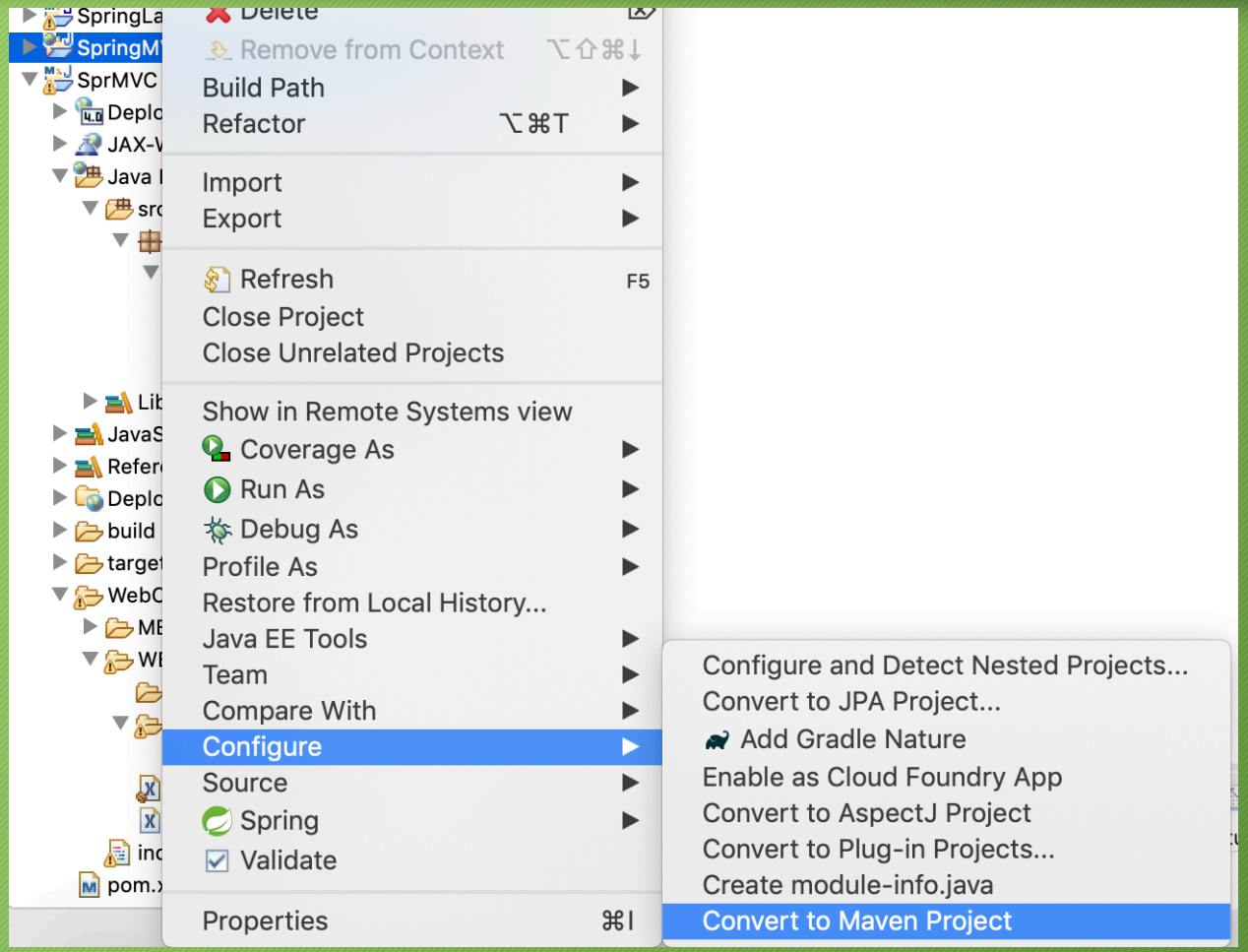
- By default, method parameters that use this annotation are required.
- But by setting the **@RequestParam** annotation's **required** flag to **false**, make method parameter is optional and give a default value.
- Multiple parameters with same name is possible by declaring the argument type as an array or a list.

Creating a a Spring MVC App – step 1

- Create a Dynamic Web Project



Step 2: Convert into Maven Project



Step 3: Configure the POM.xml file

*SpringMVC/pom.xml

SpringMVC/pom.xml

```
<project xm                                .apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:sc
  <modelVersion>4.0.0</modelVersion>
  <groupId>SpringMVC</groupId>
  <artifactId>SpringMVC</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>SpringMVC</name>
  <description>Spring MVC Example</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- Spring dependencies -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.1.9.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>5.1.9.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.1.9.RELEASE</version>
        </dependency>

    </dependencies>

  <build>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Step 4: Define and configure dispatcher-servlet.xml file

dispatcher-servlet.xml

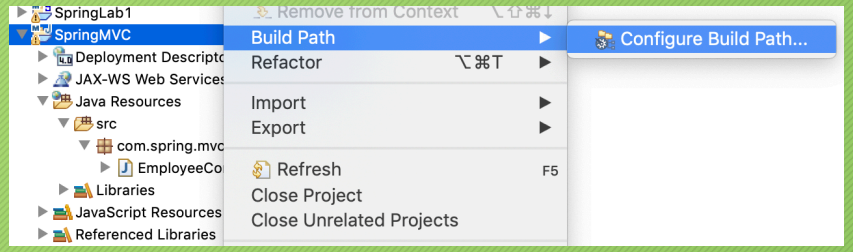
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:context="http://www.springframework.org/schema/context"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="
6  http://www.springframework.org/schema/beans
7  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8  http://www.springframework.org/schema/context
9  http://www.springframework.org/schema/context/spring-context-3.0.xsd">
10
11     <context:component-scan base-package="com.spring.mvc" />
12
13     <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
14         <property name="prefix">
15             <value>/WEB-INF/views/</value>
16         </property>
17         <property name="suffix">
18             <value>.jsp</value>
19         </property>
20     </bean>
21 </beans>
```

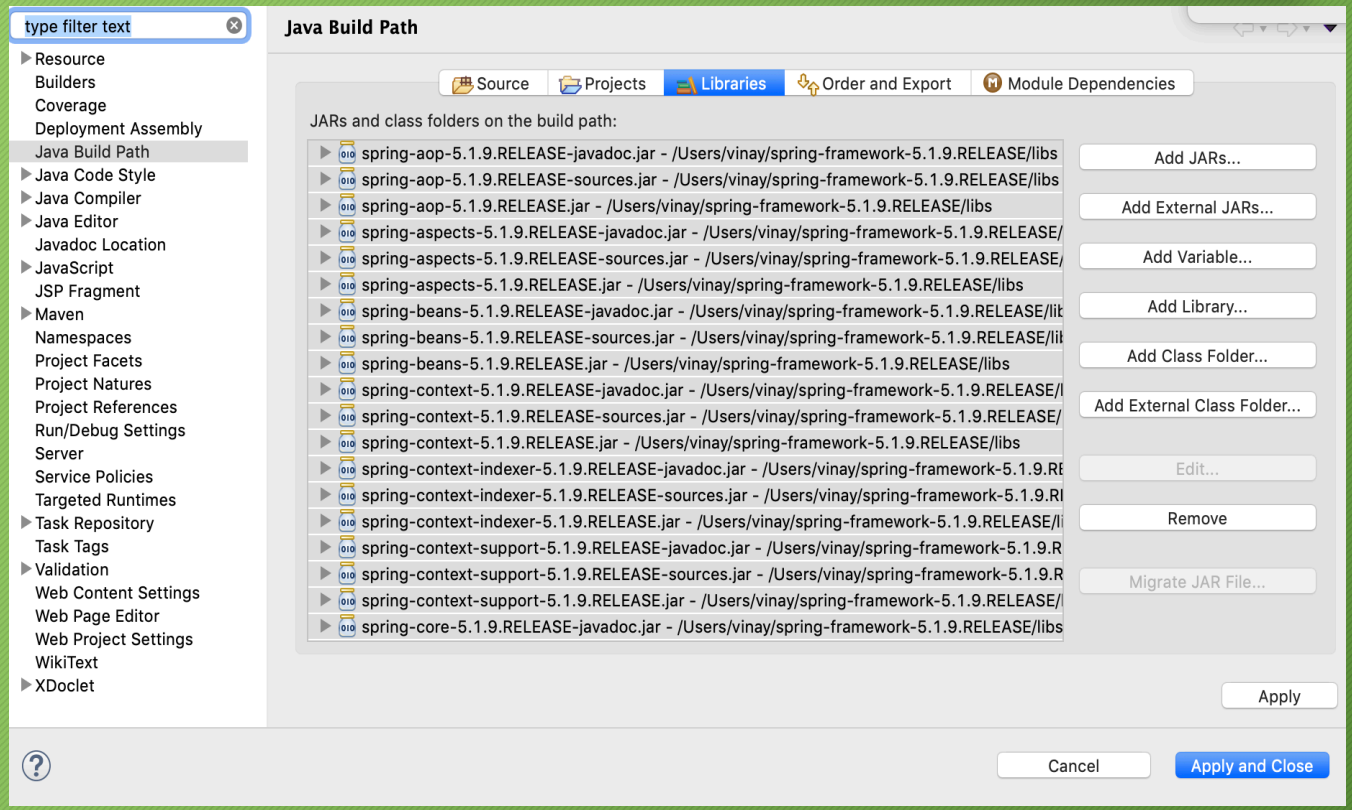
Step 5: Configure the web.xml file

*web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:s
  <display-name>SpringMVC</display-name>

  <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

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

    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

</web-app>
```

Step 6: Add Spring framework jar files





Step 7: Define the Controller file


```
@Controller
public class EmployeeController {
	String message = "Welcome to Spring MVC";
	@RequestMapping("/welcome")
	public ModelAndView printMessage(
	@RequestParam(value = "name", required = false, defaultValue = "Hi") String name) {
	ModelAndView mview = new ModelAndView("empview");
	mview.addObject("empMessage", message);
	mview.addObject("empName", name);
	return mview;
}
```

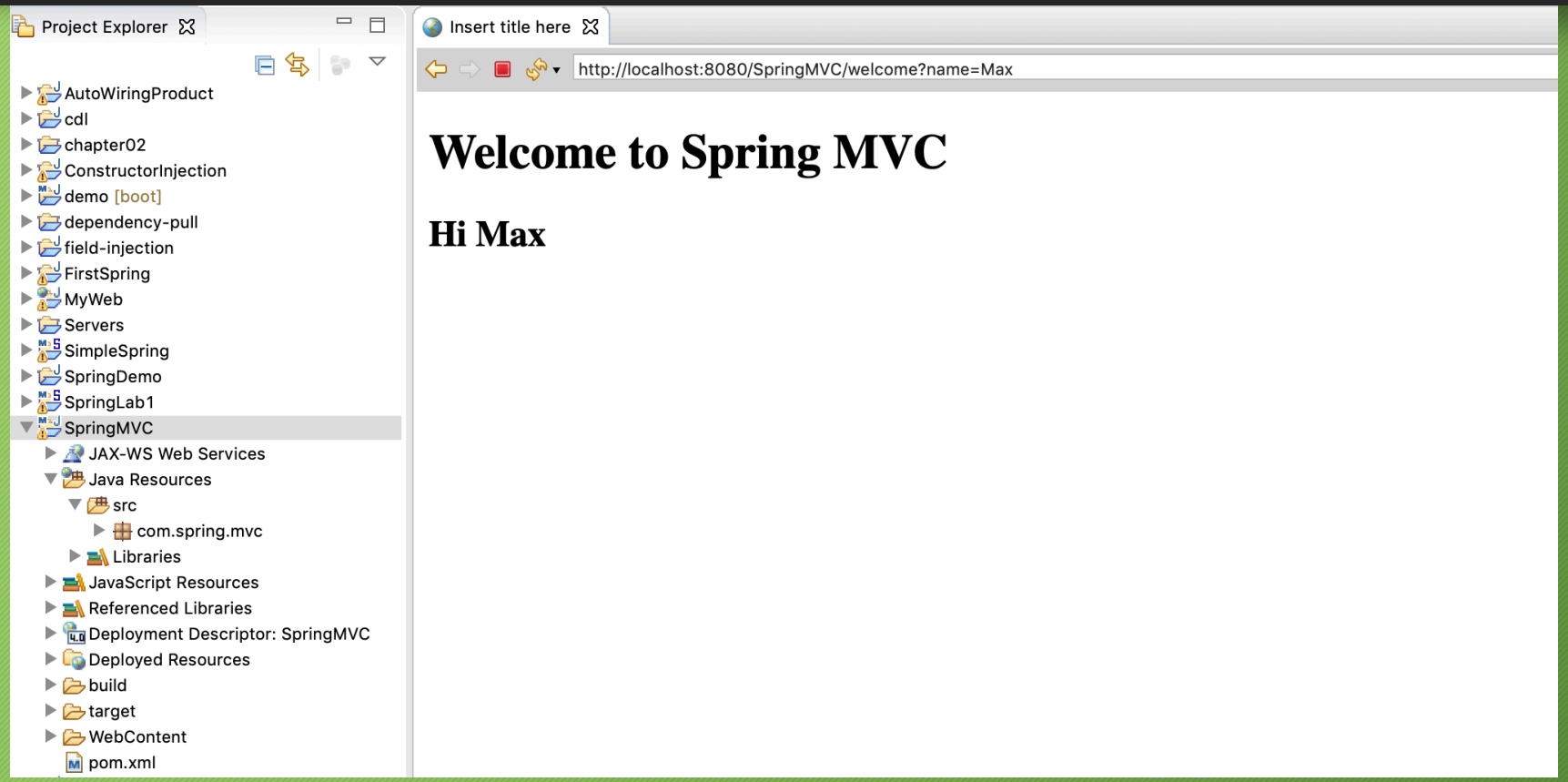
Step 8: Create index.jsp file


```
<%@ page language="java" contentType="text/html;
charset=UTF8"    pageEncoding="UTF-8"%>
<!DOCTYPE html><html>
<head>
<meta charset="UTF-8">
<title>Employee Home Page</title></head>
<body>
   <h2><a href="welcome?name=Max">Get Welcome Message </a> </h2>
 </body>
</html>
```

Step 9: Create empview.jsp file under views folder


```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html><html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1> ${empMessage} </h1>
        <h2> Hi ${empName}</h2>
</body>
</html>
```

Step 10: Run SpringMVC Project on Server



Questions?

References

- Text Book:
- Pro Spring 5 by Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho, 2019
  - Chapter 16
- Online Resource:
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html
- https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html