Sumario

Spring Web MVC (Modelo Vista Controlador)

Spring Web MVC es el framework web original construido en la API de Servlet e incluido en Spring Framework desde el principio. El nombre formal "Spring Web MVC" proviene del nombre de su módulo fuente spring-webmvc, pero se conoce más comúnmente como "Spring MVC".
El framework Spring Web MVC proporciona una arquitectura Model-View-Controller (MVC) y componentes preparados que se pueden usar para desarrollar aplicaciones web flexibles y ligeramente acopladas. El patrón MVC da como resultado la separación de los diferentes aspectos de la aplicación (lógica de entrada, lógica comercial y lógica UI), a la vez que proporciona un acoplamiento flexible entre estos elementos.

● El Modelo, encapsula los datos de la aplicación y, en general, consistirá en POJO.
● La vista, es responsable de representar los datos del modelo y, en general, genera resultados HTML que el navegador del cliente puede interpretar.
● El Controlador, es responsable de procesar las solicitudes de los usuarios y crear un modelo apropiado y lo pasa a la vista para su representación.

Hay muchos beneficios de aprender a usar Spring MVC correctamente. Pero antes de comenzar en cómo usar Spring MVC, creo que es importante dedicar algo de tiempo a obtener una perspectiva de por qué debería usarlo de todos modos.

Eventualmente encontrarás páginas de guías de Spring y descargarás uno o dos proyectos... Este es un gran lugar para comenzar su viaje de conocimientos de Spring MVC. Sin embargo, si bien puedes tener una aplicación que funcione rápidamente, realmente no sabes cómo funciona la magia de este framework. Lograrás mucho con muy poco esfuerzo, pero no aprenderás tanto como necesites. En algún momento, y ese punto generalmente llega bastante rápido, deberá comenzar a personalizar un proyecto para satisfacer tus propios requerimientos.

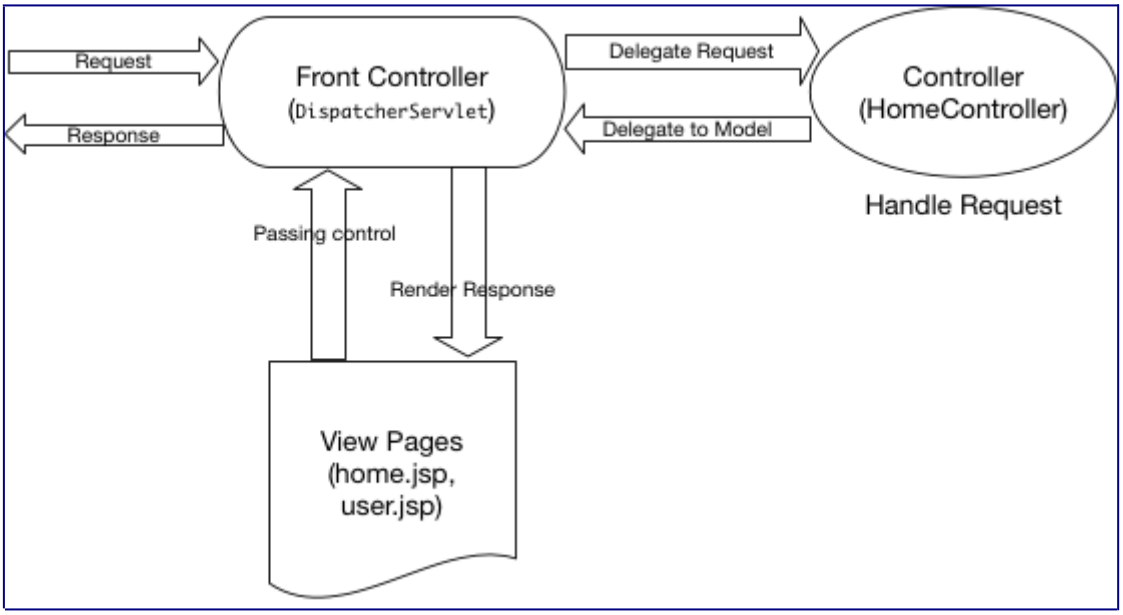
Por ejemplo, una aplicación va a tener sus propias necesidades de interfaz de usuario, sus propias necesidades de seguridad, necesidades de registro y una serie de otras cosas que simplemente no van a ser completamente manejadas para usted manualmente. E incluso si fueron manejados, probablemente se manejen de una manera parcial que no está resolviendo completamente sus necesidades. Debe tomar esa solución parcial, continuar desde donde termina Spring y agregar sus propios conocimientos a su proyecto.

Aquí es cuando una comprensión más profunda de Spring MVC es absolutamente necesaria. Hay muchos accesos directos y elecciones convenientes hechas para un proyecto de Spring. Se debe comprender dónde y cuáles son esos accesos directos para que pueda rehacerlos para satisfacer las necesidades únicas de proyecto.

La mayoría de las grandes organizaciones, como las compañías Fortune 100 y Fortune 500, invierten toneladas de dinero, seis, siete, incluso ocho o más cifras, en grandes proyectos de desarrollo de software. Con ese tipo de inversión, estos proyectos definitivamente van a utilizar frameworks de backend de escala empresarial como Spring, Spring MVC y todos sus relacionados. La Web funciona en todo el mundo, y la Web empresarial es lo que ejecuta el software utilizado o vendido por muchas de estas grandes organizaciones.

Spring MVC Example

Spring MVC is based on Model-View-Controller architecture. Below image shows Spring MVC architecture at a high level.

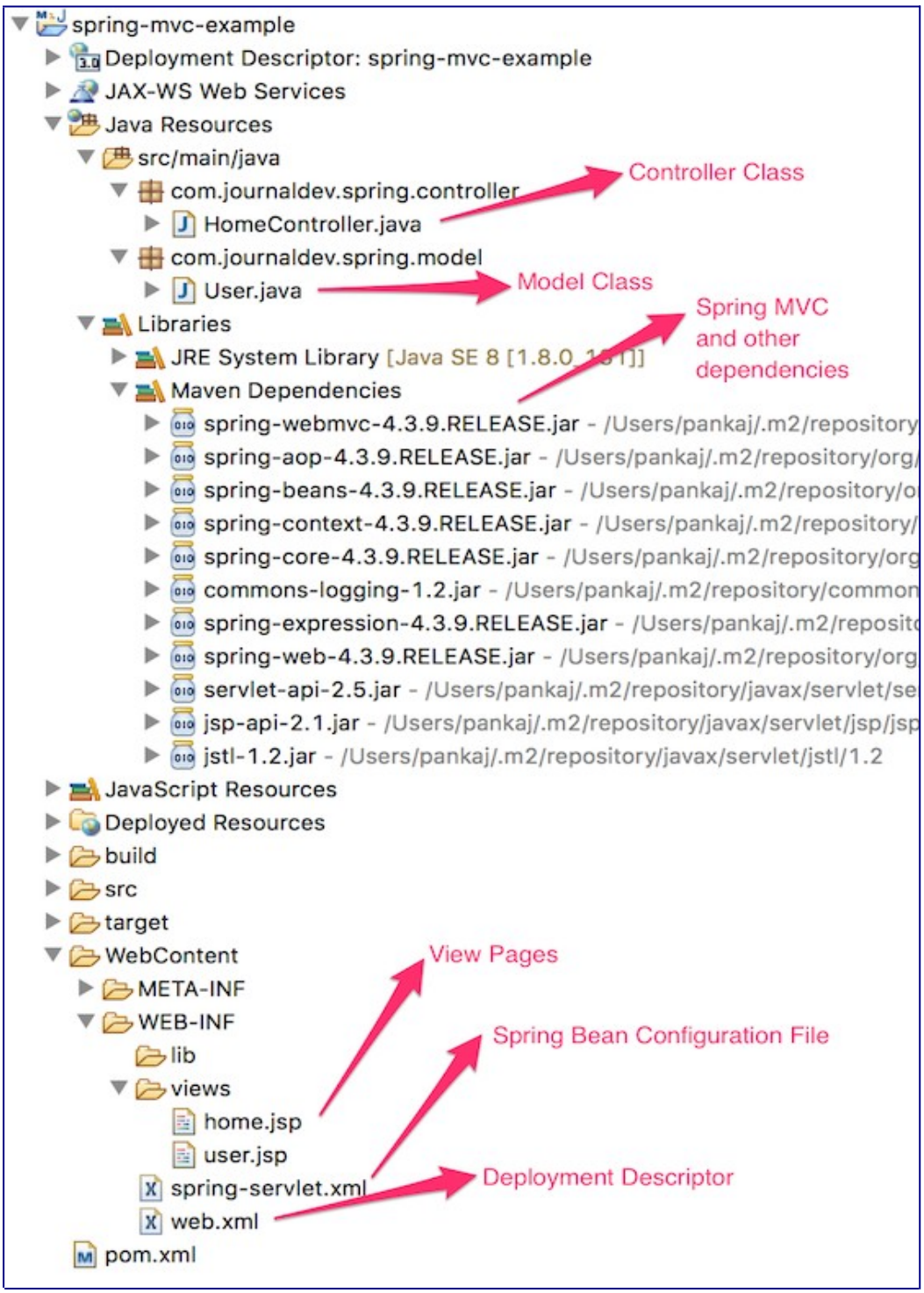


`DispatcherServlet` is the front controller class to take all requests and start processing them. We have to configure it in web.xml file. It's job is to pass request to appropriate controller class and send the response back when view pages have rendered the response page.
`HomeController.java` will be the single controller class in our spring mvc example application.
`home.jsp`, **`user.jsp`** are the view pages in our spring mvc hello world example application.

`User.java` will be the only model class we will have in our spring mvc example web application.

Spring MVC Example Hello World Eclipse Project

Below image shows our Spring MVC example project in Eclipse.



Let’s get started and create our project right from the scratch.

Spring MVC Dependencies to pom.xml

We need to add spring-web and spring-webmvc dependencies in pom.xml, also add servlet-api, jsp-api and jstl dependencies. Our final pom.xml file will be like below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.journaldev.spring.mvc</groupId>
        <artifactId>spring-mvc-example</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>war</packaging>
        <name>Spring MVC Example</name>
        <description>Spring MVC Hello World Example</description>

        <!-- Add Spring Web and MVC dependencies -->
        <dependencies>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
                        <version>4.3.9.RELEASE</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-web</artifactId>
                        <version>4.3.9.RELEASE</version>
                </dependency>
                <!-- Servlet -->
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>servlet-api</artifactId>
                        <version>2.5</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet.jsp</groupId>
                        <artifactId>jsp-api</artifactId>
                        <version>2.1</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>jstl</artifactId>
                        <version>1.2</version>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.6.1</version>
                                <configuration>
                                        <source>1.8</source>
                                        <target>1.8</target>
                                </configuration>
                        </plugin>
                        <plugin>
                                <artifactId>maven-war-plugin</artifactId>
                                <version>3.0.0</version>
                                <configuration>

<warSourceDirectory>WebContent</warSourceDirectory>
                                </configuration>
                        </plugin>
                </plugins>
                <finalName>${project.artifactId}</finalName> <!-- added to
remove Version from WAR file -->
        </build>
```

```
</project>
```

Notice the **`finalName`** configuration in build, so that our **WAR** file name doesn't have version details.

Spring MVC DispatcherServlet as Front Controller

We have to add Spring MVC framework to our web application, for that we need to configure **`DispatcherServlet`** in **web.xml** as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>spring-mvc-example</display-name>

        <!-- Add Spring MVC DispatcherServlet as front controller -->
        <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring-servlet.xml</param-value>
                </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

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

 </web-app>
```

`contextConfigLocation` init-param is used to provide the location of spring bean configuration file.

Spring MVC Example Bean Configuration File

Next step is to create spring bean configuration file `spring-servlet.xml` as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                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">
```

```
        <!-- DispatcherServlet Context: defines this servlet's request-
processing
                infrastructure -->

        <!-- Enables the Spring MVC @Controller programming model -->
        <annotation-driven />
        <context:component-scan base-package="com.journaldev.spring" />

        <!-- Resolves views selected for rendering by @Controllers to .jsp
resources
                in the /WEB-INF/views directory -->
        <beans:bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                <beans:property name="prefix" value="/WEB-INF/views/" />
                <beans:property name="suffix" value=".jsp" />
        </beans:bean>

</beans:beans>
```

There are three important configurations.

1. **`annotation-driven`** tells DispatcherServlet to look for Controller classes using `@Controller` annotation.
2. **`context`**`:component-scan` tells DispatcherServlet where to look for controller classes.
3. **`InternalResourceViewResolver`** bean configuration to specify location of view pages and suffix used. Controller class methods return name of the view page and then suffix is added to figure out the view page to use for rendering the response.

Spring MVC Controller Class

We have a single controller class to respond for two URIs – “/” for home page and “/user” for user page.

```
package com.journaldev.spring.controller;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.journaldev.spring.model.User;

@Controller
public class HomeController {

        /**
         * Simply selects the home view to render by returning its name.
         */
        @RequestMapping(value = "/", method = RequestMethod.GET)
        public String home(Locale locale, Model model) {
                System.out.println("Home Page Requested, locale = " + locale);
                Date date = new Date();
```

```
                DateFormat dateFormat =
DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

                String formattedDate = dateFormat.format(date);

                model.addAttribute("serverTime", formattedDate);

                return "home";
        }

        @RequestMapping(value = "/user", method = RequestMethod.POST)
        public String user(@Validated User user, Model model) {
                System.out.println("User Page Requested");
                model.addAttribute("userName", user.getUserName());
                return "user";
        }
}
```

Note that for simplicity, I have not used any logging framework such as log4j.

Spring MVC Model Class

We have a simple model class with a single variable and it's getter-setter methods. It's a simple POJO class.

```
package com.journaldev.spring.model;

public class User {
        private String userName;

        public String getUserName() {
                return userName;
        }

        public void setUserName(String userName) {
                this.userName = userName;
        }
}
```

Spring MVC View Pages

We have two view pages as defined below.

`home.jsp`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page session="false"%>
<html>
<head>
<title>Home</title>
</head>
<body>
        <h1>Hello world!</h1>

        <P>The time on the server is ${serverTime}.</p>

        <form action="user" method="post">
                <input type="text" name="userName"><br> <input
```

```
                        type="submit" value="Login">
        </form>
</body>
</html>
```

user.jsp

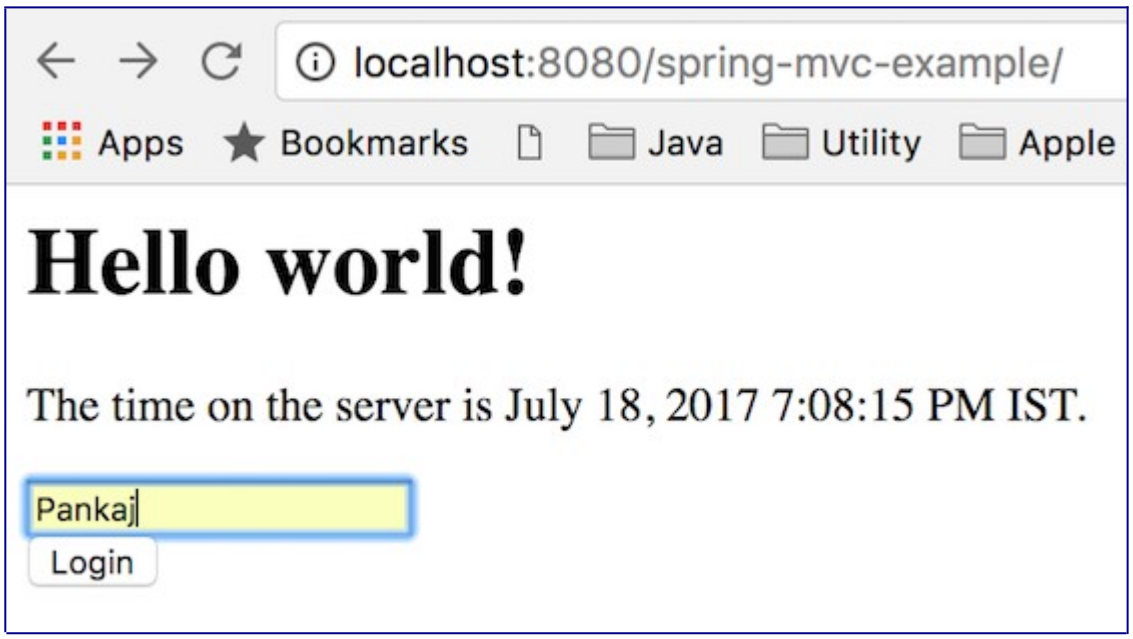
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>User Home Page</title>
</head>
<body>
<h3>Hi ${userName}</h3>
</body>
</html>
```

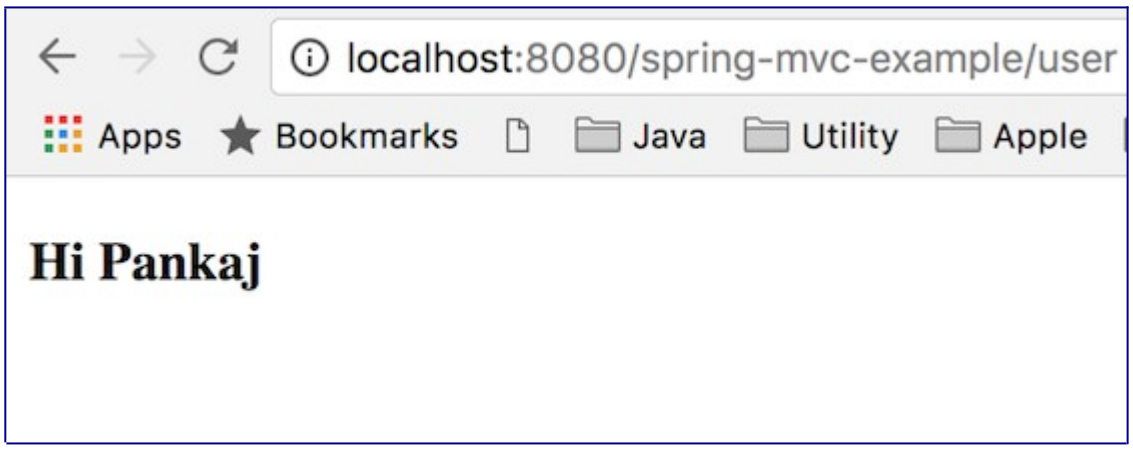
Notice that Spring MVC takes care of mapping form variables to model class variables, that's why we have same variable name in both places.

That's it, our spring mvc example project is ready to de deployed and test.

Spring MVC Example Test

Once the spring mvc project is deployed, we can access the home page at `http://localhost:8080/spring-mvc-example/`. Change the tomcat port and context-root accordingly.





That’s all for Spring MVC example, I have tried to keep it as simple as possible. But still if you face any issues then please let me know through comments and I will try to help you out. You can download the final spring mvc example project from below link.

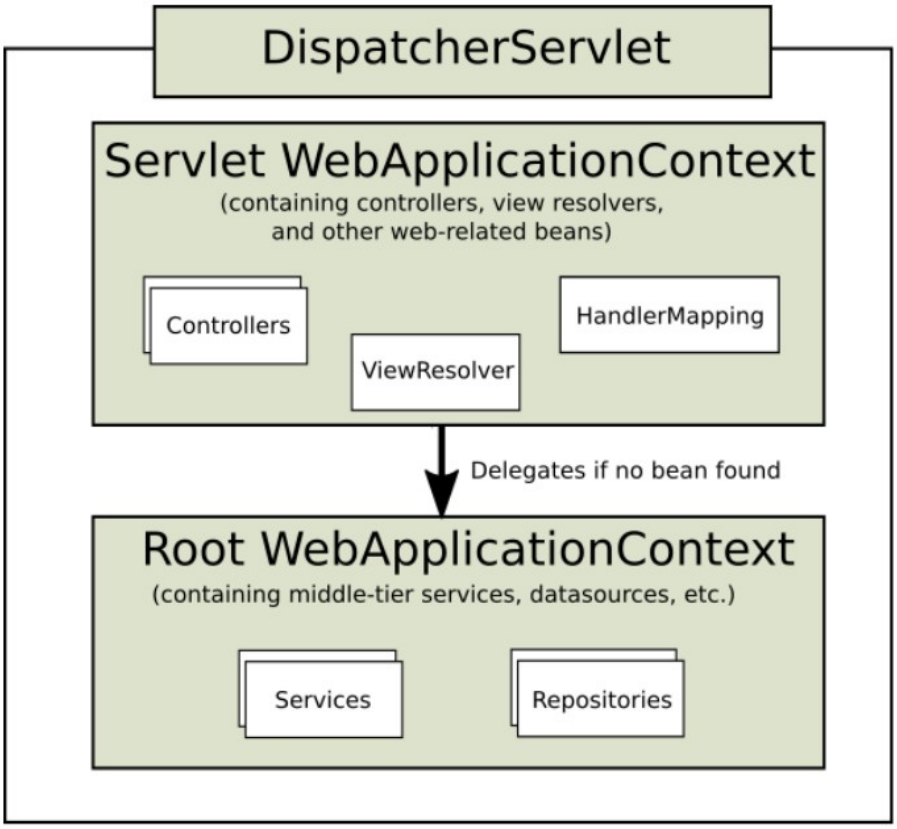
Arquitectura Spring MVC

DispatcherServlet

Spring MVC, al igual que muchos otros frameworks web, está diseñado en torno al patrón del controlador frontal, donde un Servlet central, DispatcherServlet, proporciona un algoritmo compartido para el procesamiento de solicitudes mientras que el trabajo real se realiza mediante componentes delegados configurables. Este modelo es flexible y admite diversos flujos de trabajo.
El DispatcherServlet, como cualquier Servlet, debe declararse y correlacionarse de acuerdo con la especificación de Servlet mediante la configuración de Java o en web.xml. A su vez, DispatcherServlet utiliza la configuración de Spring para descubrir los componentes de delegado que necesita para la asignación de solicitudes, la resolución de vistas, el manejo de excepciones y más.

Jerarquía de Contextos

DispatcherServlet espera un WebApplicationContext, una extensión de un ApplicationContext simple, para su propia configuración. WebApplicationContext tiene un enlace al ServletContext y Servlet al que está asociado. También está vinculado al ServletContext de manera que las aplicaciones pueden usar métodos estáticos en RequestContextUtils para buscar el WebApplicationContext si necesitan acceder a él.
Para muchas aplicaciones tener un solo WebApplicationContext es simple y suficiente. También es posible tener una jerarquía de contexto donde un WebApplicationContext raíz se comparte a través de múltiples instancias de DispatcherServlet (u otras instancias del Servlet), cada una con su propia configuración secundaria de WebApplicationContext.
El WebApplicationContext raíz normalmente contiene beans de infraestructura, como repositorios de datos y servicios comerciales, que deben compartirse en varias instancias de Servlet. Esos beans se heredan de manera efectiva y se pueden anular(es decir, volver a declarar) en el WebApplicationContext hijo específico del servlet, que normalmente contiene beans locales al Servlet determinado:



Lo que realmente queremos hacer como desarrolladores de una aplicación web es abstraer las siguientes tareas tediosas y repetitivas y centrarnos en la lógica empresarial útil:

● Asignar una solicitud HTTP a un determinado método de procesamiento.

● Análisis de datos de solicitud HTTP y encabezados en objetos de transferencia de datos (DTO) u objetos de dominio

● Interacción modelo-vista-controlador.

● Generación de respuestas de DTOs, objetos de dominio, etc.

Spring DispatcherServlet proporciona exactamente eso. Es el corazón de Spring Web MVC; este componente central recibe todas las solicitudes a su aplicación.
Como se puede ver, DispatcherServlet es muy extensible. Por ejemplo, le permite conectar diferentes adaptadores existentes o nuevos para muchas tareas:

● Asignar una solicitud a una clase o método que debe manejarlo (implementaciones de HandlerMappinginterface).

● Manejar una solicitud usando un patrón específico, como un servlet regular, un flujo de trabajo MVC más complejo, o simplemente un método en un bean POJO (implementaciones de la interfaz HandlerAdapter).

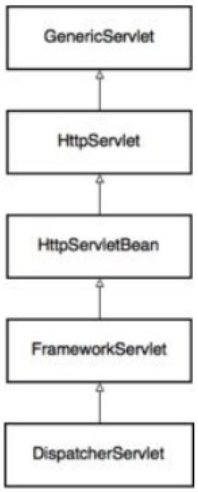
● Resuelva las vistas por nombre, lo que le permite usar diferentes motores de plantillas, XML, XSLT o cualquier otra tecnología de visualización (implementaciones de la interfaz ViewResolver).

● Analizar peticiones de varias partes utilizando la implementación de carga de archivos de Apache Commons por defecto o escribiendo su propio MultipartResolver.

● Resolver la configuración regional con cualquier implementación de LocaleResolver, incluidas cookies, sesiones, aceptar el encabezado HTTP o cualquier otra forma de determinar la configuración regional que el usuario espera.

Procesamiento de una solicitud HTTP

Primero, tracemos el procesamiento de solicitudes HTTP simples a un método en su capa de controlador y de vuelta al navegador / cliente.
El DispatcherServlet tiene una jerarquía de herencia larga; vale la pena entender estos aspectos individuales uno por uno, de arriba hacia abajo. Los métodos de procesamiento de solicitudes nos interesarán más.



Comprender la solicitud HTTP, tanto localmente durante el desarrollo estándar, como de forma remota, es una parte fundamental para comprender la arquitectura MVC.

GenericServletGenericServlet es una parte de la especificación de Servlet que no se enfoca directamente en HTTP. Define el método service () que recibe las solicitudes entrantes y produce respuestas.

HttpServlet

La clase HttpServlet es, como su nombre lo sugiere, la implementación Servlet centrada en HTTP, también definida por la especificación. En términos más prácticos, HttpServlet es una clase abstracta con una implementación de método service () que divide las solicitudes por el tipo de método HTTP.

HttpServletBean

A continuación, HttpServletBean es la primera clase consciente de Spring en la jerarquía. Inyecta las propiedades del bean utilizando los valores de servlet init-param recibidos de web.xml o de WebApplicationInitializer. En el caso de las solicitudes a su aplicación, se solicitan los métodos doGet (), doPost (), etc. para esas solicitudes HTTP específicas.

FrameworkServlet

FrameworkServlet integra la funcionalidad Servlet con un contexto de aplicación web, implementando la interfaz ApplicationContextAware. Pero también es capaz de crear un contexto de aplicación web por sí mismo.

Como ya vio, la superclase HttpServletBean inyecta init-params como propiedades de bean. Entonces, si se proporciona un nombre de clase de contexto en el contextClass init-param del servlet, entonces se creará una instancia de esta clasecomo contexto de aplicación.
De lo contrario, se usará una clase XmlWebApplicationContext predeterminada.

DispatcherServlet: Unificar el procesamiento de solicitudes

La implementación HttpServlet.service (), que enruta las solicitudes por el tipo de verbo HTTP, tiene mucho sentido en el contexto de los servlets de bajo nivel. Sin embargo, en el nivel de abstracción Spring MVC, el tipo de método es solo uno de los parámetros que se pueden usar para asignar la solicitud a su controlador. Y entonces, la otra función principal de la clase FrameworkServlet es unir la lógica de manejo a un único método processRequest (), que a su vez llama al método doService ():

DispatcherServlet: Enriquecer la solicitud

Finalmente, el DispatcherServlet implementa el método doService (). Aquí, se agrega a la solicitud algunos objetos útiles que pueden ser útiles en la línea de procesamiento: contexto de la aplicación web, resolver el entorno local, resolver el tema, fuente del tema, etc.
Además, el método doService () prepara mapas flash de entrada y salida.
Flashmap es básicamente un patrón para pasar parámetros de una solicitud a otra solicitud que sigue inmediatamente. Esto puede ser muy útil durante los redireccionamientos.

DispatcherServlet: Envío de la solicitud

El objetivo principal del método dispatch () es encontrar un controlador apropiado para la solicitud y darle los parámetros de solicitud / respuesta. El controlador es básicamente cualquier tipo de objeto y no está limitado a una interfaz específica. Esto también significa que Spring necesita encontrar un adaptador para este controlador que sepa cómo "hablar" con el controlador.

Para encontrar el controlador que coincida con la solicitud, Spring realiza las implementaciones registradas de la interfaz HandlerMapping.

Manejo de la solicitud (Request)

Ahora que Spring determinó el controlador para la solicitud y el adaptador para el controlador, es hora de manejar finalmente la solicitud. Aquí está la firma del método HandlerAdapter.handle (). Es importante tener en cuenta que el controlador tiene una opción en cómo manejar la solicitud.
Escriba los datos en el objeto de respuesta por sí mismo.
Devuelve un objeto ModelAndView para ser renderizado por el DispatcherServlet.
Hay varios tipos de controladores disponibles.
Así es como SimpleControllerHandlerAdapter procesa una instancia del controlador Spring MVC (no la confunda con un POJO con anotación de controlador @).

Representando la vista

Por ahora, Spring ha procesado la solicitud HTTP y ha recibido un objeto ModelAndView, por lo que debe representar la página HTML que el usuario verá en el navegador. Lo hace en función del modelo y la vista seleccionada encapsulada en el objeto ModelAndView.
También tenga en cuenta que podría representar un objeto JSON o XML, o cualquier otro formato de datos que se pueda transferir a través del protocolo HTTP.
Vamos a tocar más sobre eso en la próxima sección centrada en REST aquí.
Regresemos al DispatcherServlet. El método render () primero establece la configuración regional de la respuesta utilizando la instancia de LocaleResolver proporcionada. Supongamos que su navegador moderno establece correctamente el encabezado Accept y que AcceptHeaderLocaleResolver se usa de forma predeterminada.
Durante el procesamiento, el objeto ModelAndView ya podría contener una referencia a una vista seleccionada, o solo un nombre de vista, o nada si el controlador confiaba en una vista predeterminada.