

# Introduction to Spring MVC Framework



Tuna Tore

# What is MVC?

- **MVC** stands for **Model View Controller (architecture)**

MCV đại diện cho ... (kiến trúc)

- It is a design pattern used in WEB based **JAVA Enterprise Applications**

nó là một design pattern sử dụng trong ứng dụng web java enterprise

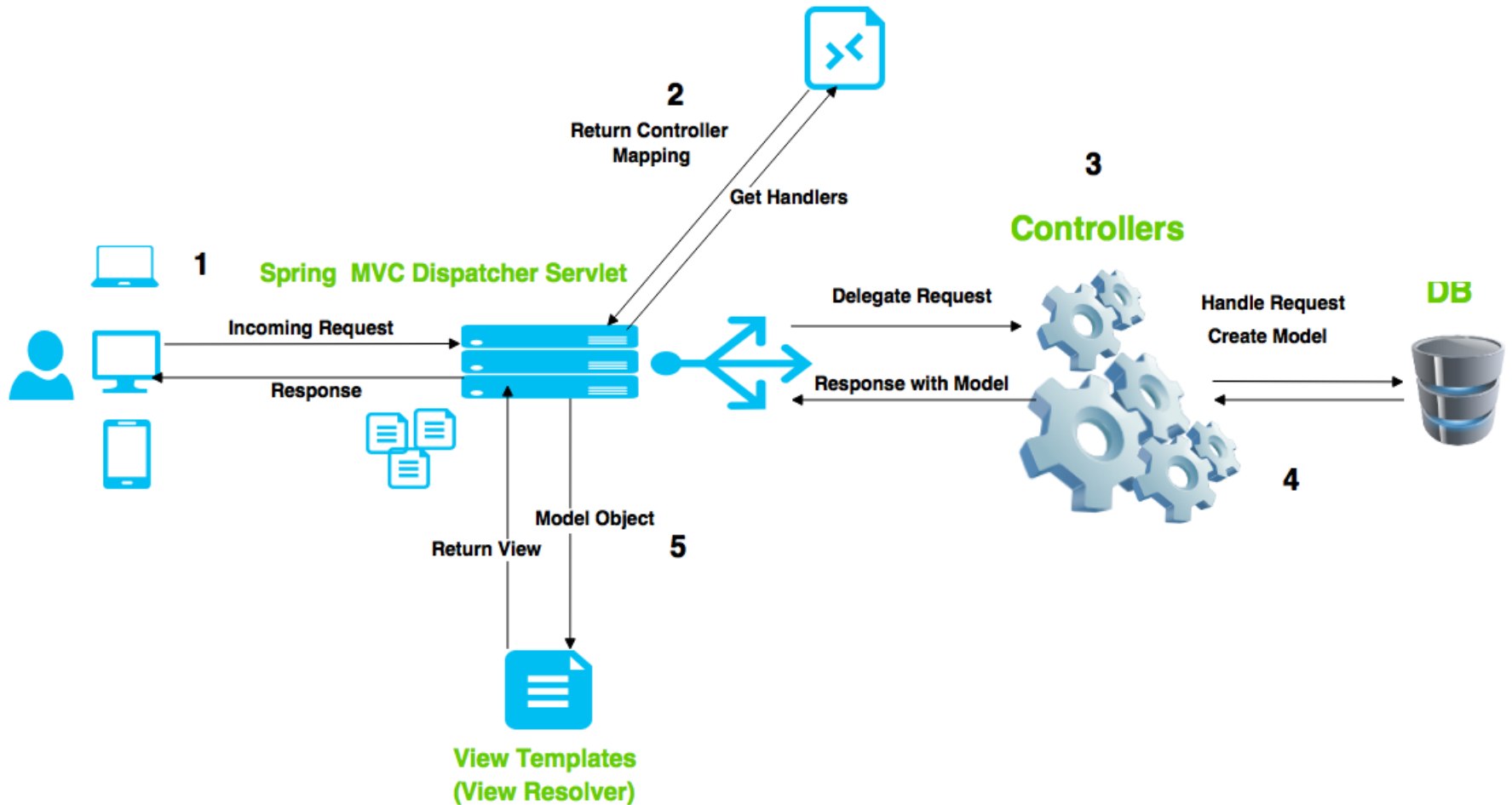
- **MVC pattern** is also implemented in other WEB frameworks as well for separating the model layer from view layer. By doing that UI programmers and back end developers can work together.
- **Model Layer** includes business specific logic
- **View Layer** is responsible for rendering(displaying) model objects inside the user interface using different view technologies (JSP, Facelets or Velocity) (browser)
- **Controller** receives user inputs and calls model objects based on handler mappings and pass model objects to views in order to display output inside the view layer.

# What is **Dependency Injection**?

<https://toidicodedao.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/>

- This is also called as **IOC (Inversion of Control Principle)**
- It is a **software design pattern** which is really useful for designing loosely coupled software components là một dspartern mà hữu ích cho việc thiết kế các cặp thành phần phần mềm theo từng cặp
- You will have more **plug-gable** and testable code and objects tính dễ chỉnh sửa, test code và các đối tượng
- It is easy to **reuse** your code in other applications dễ dàng tái sử dụng trong ứng dụng khác
- The dependencies won't be hard coded inside all java objects/ classes instead they will be defined in **XML configuration files** or configuration classes (**Java Config**) giảm bớt sự phụ thuộc vào java code, thay vào đó là định nghĩa trong file cấu hình xml hoặc những class cấu hình java config
- Spring Container is responsible for injecting dependencies of objects chịu trách nhiệm cho các đối tượng phụ tuộc injecting
- There are two types of **Dependency injection** in Spring Framework **Setter Injection** and **Constructor Injection**

## Request Processing Workflow in Spring MVC

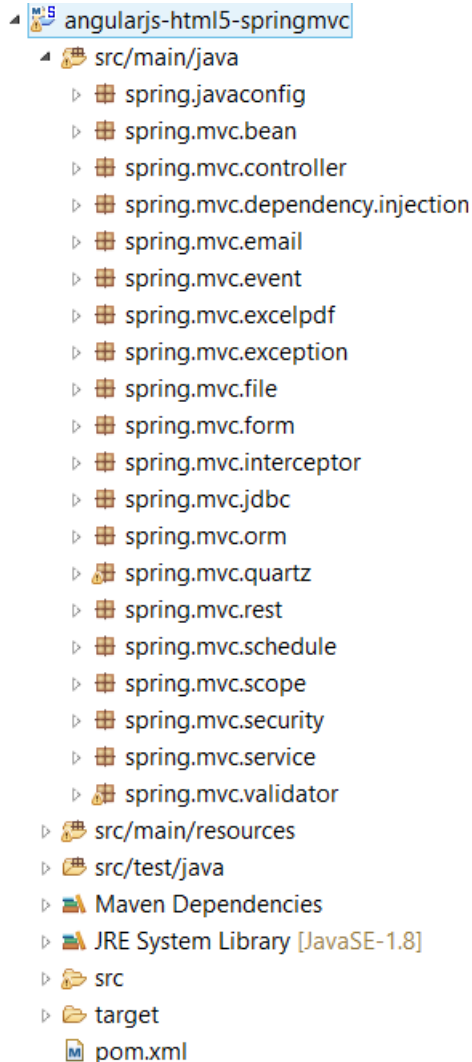


# The general information about web.xml and Java EE

- The **WEB-INF/web.xml** is the Web Application Deployment Descriptor of Java Enterprise Web applications/
- Inside **web.xml**, developers can define

Servlets,  
Welcome File List  
Filters,  
Context parameters,  
Error pages  
Security rights and etc.

# The general information about the sample web application



- **Apache Maven Project**
- **Pivotal tc server**
- **Java Spring MVC 4.x version**

# What is the concept of DispatcherServlet and how do you configure it?

được cấu hình để gửi http request đi và nhận về kết quả từ trình duyệt

- **DispatcherServlet** is configured in order to dispatch incoming HTTP request to handlers and returns response to browsers
- **Spring MVC** is designed around **DispatcherServlet**
- **DispatcherServlet** is configured in web.xml file

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/classes/mvc-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

# What is the concept of DispatcherServlet and how do you configure it?

- **DispatcherServlet** is configured in order to dispatch incoming HTTP request to handlers and returns response to browsers
- **Spring MVC** is designed around **DispatcherServlet**
- **DispatcherServlet** is configured in web.xml file

```
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/classes/mvc-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```



# What is **ContextLoaderListener** and how do you configure it in Spring MVC?

- **ContextLoaderListener** is responsible for creating a root application context in Java Enterprise Web application inside **ServletContext**
- Developers can use different controller and view layers (For example Struts, JSF or Spring MVC can be used) however **Spring Framework** will be responsible for managing services layer or components defined **Context-LoaderListener** configuration files
- You can set different configuration XML files in web.xml

<http://docs.spring.io/spring/docs/2.5.x/reference/mvc.html#mvc-servlet>

# How do you configure controllers in Spring MVC?

- By extending **AbstractController** class and implementing **handleRequestInternal** method
- By using **@Controller** or **@RestController** annotation on class level
- Controller with bean name **URL mapping**
- Controller with **class name** mapping
- Controller with **XML config with mappings**

# How is an incoming request mapped to a mapped to a method in Spring MVC?

- **@RequestMapping** is used in order to map incoming request to a controller using URL matching
- **@RequestParam** and **@PathVariable** is used to get parameters or variable from HTTP request

```
@RequestMapping(value = "/jdbcDelete/user/{iduser}",  
method=RequestMethod.GET)  
public ModelAndView jdbcDelete(@PathVariable(value="iduser")  
int iduser) {
```

```
@RequestMapping(value="/register", method=RequestMethod.POST)  
public ModelAndView register(@RequestParam(required=true)  
String email,@RequestParam(required=true) String password)
```

## What is a **View in Spring MVC**? How is the **InternalResourceViewResolver** configured?

- **Java Spring MVC** can be configured with different view technologies such as **Java Server Faces**, **JSP**, **Velocity** etc.
- **Views** are responsible for displaying the content of Model objects on browsers as response output
- **InternalResourceViewResolver** is used to **resolve the correct view** based on suffix and prefixes so the correct output (view) is resolved based on strings

## Return types from Controllers

You can return the followings from Controllers

- **ModelAndView**
- **Model**
- **Map**
- **View**
- **String**
- **void**
- **HttpHeaders and others**

## Scopes in Spring MVC

- Default Scope in Spring MVC is **request**
- **Session scope** can be used in order to save the state of beans in web based applications
- Each time a request send through HTTP a new bean will be created with request scope

# Application Event Handling

- **Developers can handle application events by implementing**

`ApplicationListener<ApplicationEvent>`

- **Customs events can be published by using application context and extending**

`ApplicationEvent` class

# Spring MVC File Upload

- In order to upload a file with **Java Spring MVC**

```
<!-- File Upload bean config-->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartRe
solver">
<!-- set the maximum file size in bytes -->
<property name="maxUploadSize" value="1000000"/>
</bean>
```

@Controller

```
public class FileUploadController {
@RequestMapping(value="/uploadFile", method=RequestMethod.POST)
public @ResponseBody String
handleFileUpload(@RequestParam("file") MultipartFile file){
```



# Quartz Scheduling Framework and Spring Schedules

- **Quartz** is an alternative to **Spring Schedules** and has more functionality
- In order to use Quartz; **Triggers, Jobs, Tasks and Scheduler** have to be defined in XML configuration
- Spring Schedules can be configured using **@Scheduled** annotation
- **@Scheduled(cron="0/30 \* \* \* \* ?")** or **@Scheduled(fixedDelay = 10000)** can be applied to schedules

# Logging in Spring MVC

- **LOGBack** framework will be used in order to log outputs
- LOGBack is a significantly improved version of log4j logging framework
- Both log4j and logback were founded with the same developer
- It is easy to switch logging technologies by using LOG-Back
- The configuration is done through **logback.xml**
- **By default Spring Framework use commons-logging so dependencies should be excluded in pom.xml**

# Apache Maven

- **Apache Maven** is a build automation and dependency management tool
- Apache Maven is configured using **pom.xml** file
- Maven is integrated into **Eclipse or STS**

# JPA and Hibernate ORM

- **JPA** stands for Java Persistence API
- **Hibernate ORM** is an implementation of JPA
- ORM stands for **Object Relational Mapping**
- Hibernate ORM uses **@Entity** annotation to manage classes in ORM framework
- **@Table**(name="USER") is used to map Java Objects to Database Tables
- **@Id** and **@Column**(name="USERNAME") annotations can be used on field levels.
- **@PersistenceContext** has to be used on EntityManager

# JDBC and JdbcTemplate

- **JDBC** stands for Java Database Connectivity
- It is an application programming interface API for connecting different databases
- Spring MVC uses **jdbcTemplate** in order to run queries on databases
- **JdbcTemplate** simplifies database access code
- **JdbcTemplate** handles database related exceptions and throws **DataAccessException**

# PDF and Excel Documents

- In order to return **PDF and Excel Documents** `org.springframework.web.servlet.view.XmlViewResolver` has to be configured in Spring MVC Framework
- Documents have to extend related classes named as `AbstractExcelView` and **`AbstractPdfView`**
- **Apache POI** is used to generate PDFs inside sample applications
- **Itext** library is used to generate Excel documents

# Spring MVC Java Config

- In order to configure **Spring MVC** with JavaConfig;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = {"spring.javaconfig"})

**public class** JavaConfig **extends** WebMvcConfigurerAdapter {

- You don't need **web.xml** file with the new specification  
Servlet API 3.0+

**public class** WebInitializer **implements**

WebApplicationInitializer {

@Override

**public void** onStartup(ServletContext servletContext)

**throws** ServletException {

# Spring MVC Email

- **Spring MVC** can send emails to users using Java Mail API
- Inside the sample application **Velocity Email Template** is used in order to send customized emails
- Velocity is configured in configuration files and template locations should be set

```
<!-- Velocity Email Template Config Bean -->
<bean id="velocityEngine"
class="org.springframework.ui.velocity.VelocityEngineFactoryBean">
  <property name="resourceLoaderPath" value="/WEB-INF/email-templates/" />
</bean>
```



# Spring Security

- In order to activate Spring Security in Spring MVC applications, springSecurityFilterChain has to be configured in web.xml file with **DelegatingFilterProxy**
- **Spring Security** annotations and custom tags can be used in order to define java method level security

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
```

# Spring Exception Handling

- Inside **Spring MVC**, users can define a exception handling class by implementing

```
@Component
public class ExceptionHandler implements HandlerExceptionResolver{

    private static final Logger logger =
        LoggerFactory.getLogger(ExceptionHandler.class);

    @Override
    public ModelAndView resolveException(HttpServletRequest
request, HttpServletResponse response, Object object, Exception
exception) {

        logger.error("Error: ", exception);
        return new
        ModelAndView("error/exception", "exception", "ExceptionHandler
message: " + exception.toString());
    }
}
```

# Spring REST with RestTemplate

- **@RestController** will add **@ResponseBody** annotations to all methods inside a class
- **RestTemplate** is used to access Rest Based Web Services
- **@PathVariable** is used in order to get variables from URL
- **ResponseEntity** class is used to map response to Java Objects