

LẬP TRÌNH JAVA

SpringMVC

*ThS. Dương Hữu Thành
Khoa CNTT, Đại học Mở Tp.HCM
thanh.dh@ou.edu.vn*

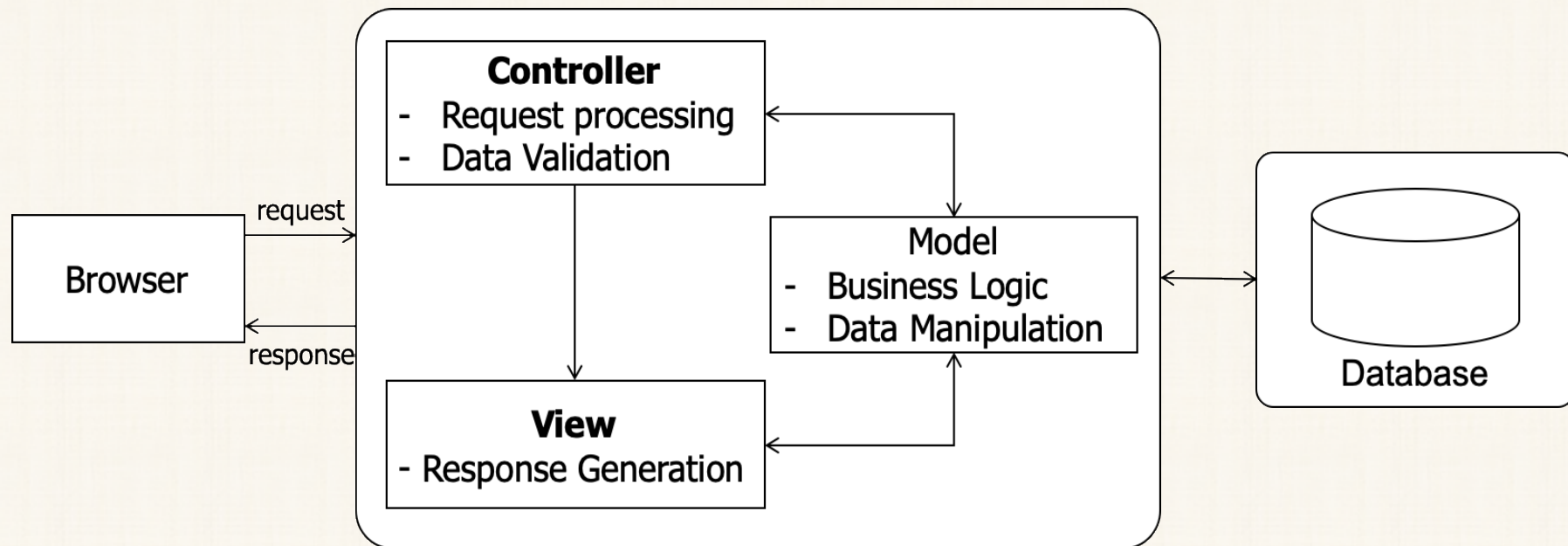




Nội dung chính

- 1. Giới thiệu SpringMVC**
- 2. Front Controller Design Pattern**
- 3. Controller**
- 4. Tag Libraries**
- 5. ViewResolver**
- 6. Hibernate Config**
- 7. Spring Tiles**

- Spring MVC là một framework mã nguồn mở dùng phát triển các ứng dụng Web theo mô hình MVC (Model-View-Controller).





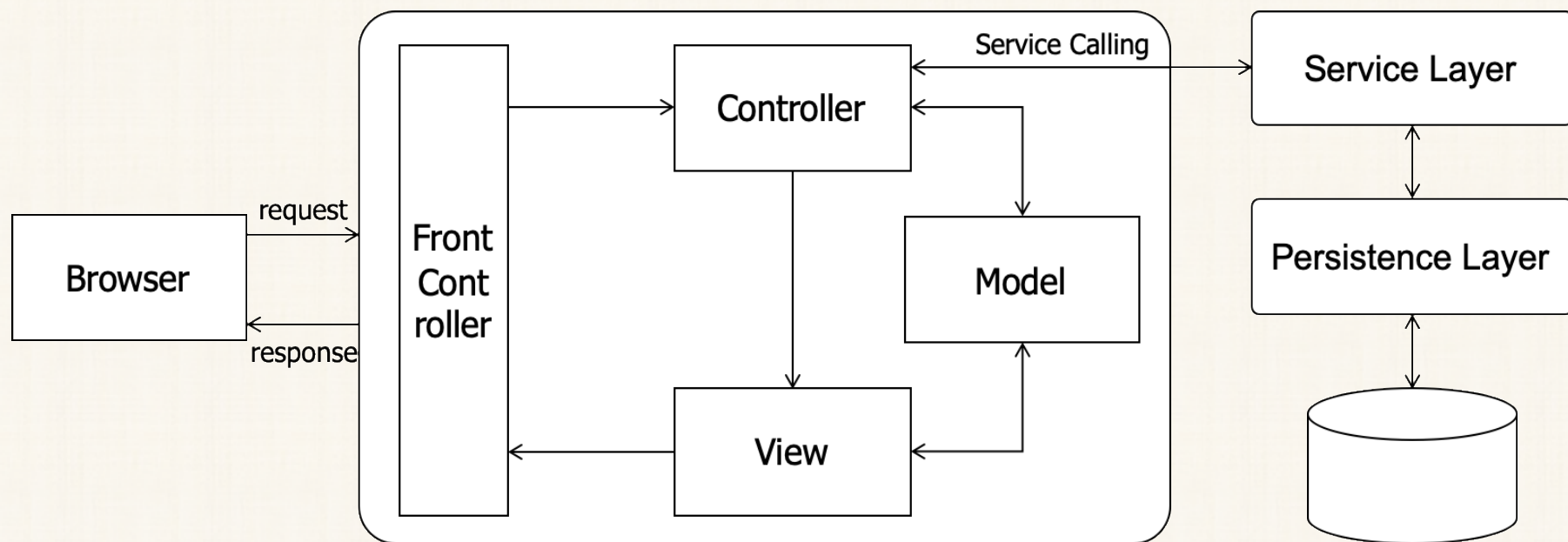
SpringMVC

- Spring MVC hiện thực tất cả các đặc trưng nổi bật của Spring Core như Inversion of Control, Dependency Injection.
- Phát triển các ứng dụng theo Spring MVC
 - models sẽ bao gồm các đối tượng domain được xử lý bởi tầng service và được lưu trữ bởi tầng persistence
 - view sử dụng JSP template được viết với JSTL (Java Standard Tag Library), ta cũng có thể định nghĩa các view là các tập tin pdf, excel hoặc các RESTful Web Service.



Front Controller Design Pattern

- Front Controller là điểm bắt đầu xử lý của tất cả các HTTP request, nó cũng là nơi khởi động vài thành phần quan trọng của framework.





Front Controller Design Pattern

- Front Controller nhận (intercept) request người dùng, thực hiện các chức năng chung, chuyển (dispatch) request đến controller tương ứng dựa trên cấu hình của ứng dụng Web và thông tin của HTTP request.
- Controller tương tác với tầng dịch vụ (Service Layer) thực hiện các logic nghiệp vụ (business logic) và lưu trữ (persistence logic).

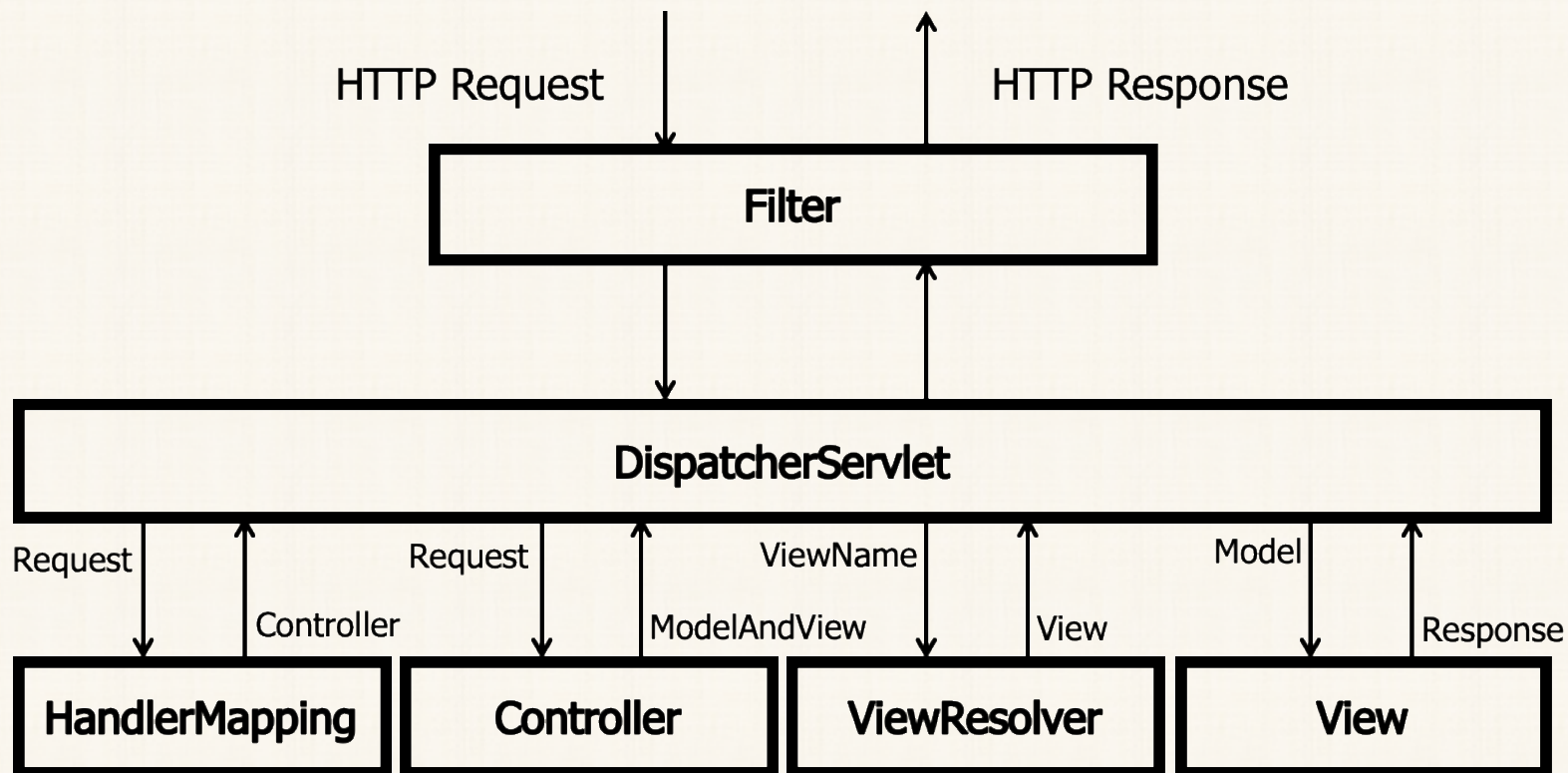


Front Controller Design Pattern

- Sau đó cập nhật model và view sẽ kết xuất dữ liệu của model cho View hiển thị và trả View đó về cho người dùng.
- Cuối cùng, Front Controller phản hồi đến client dưới dạng một View. Trong Spring MVC, **DispatcherServlet** làm việc như Front Controller.

DispatcherServlet

- DispatcherServlet xử lý các HTTP Request và Response, nó quyết định phương thức nào của controller sẽ được thực thi khi nhận Request.





DispatcherServlet


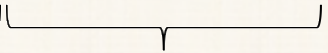
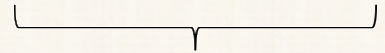
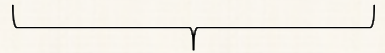
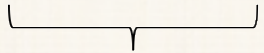
- Khi nhận được HTTP request, DispatcherServlet sẽ gọi Controller thích hợp dựa trên HandlerMapping.
- Controller nhận được request sẽ gọi phương thức thích hợp dựa trên phương thức request là POST hay GET.
- DispatcherServlet tìm view có sẵn cho request dựa trên ViewResolver, sau đó gửi dữ liệu đến view để kết xuất, hiển thị lên trình duyệt.



DispatcherServlet

- Trong Spring MVC, URL được chia làm 5 phần như bên dưới, khi người dùng thực hiện một request thì DispatcherServlet sẽ tìm kiếm phương thức trong controller phù hợp với phần Request Path.

`http://localhost:8080/SpringMVCDemo/dicts/search.htm?word=Love`

				
Scheme	Domain name	Application Name	Request Path	Request Params

- Lớp controller trong Spring sử dụng annotation **@Controller** hoặc **@RestController**.
- Khi một lớp gắn annotation là **@Controller** nhận một request, nó sẽ tìm kiếm phương thức xử lý thích hợp cho request đó thông qua annotation **@RequestMapping** chỉ định ánh xạ (mappings) giữa request với phương thức được gắn annotation này.



DispatcherServlet

- Phương thức xử lý request có thể chứa tùy ý các loại tham số sau:
 - `HttpServletRequest` hoặc `HttpServletResponse`.
 - Các tham số trên URL với annotation `@RequestParam`.
 - Các thuộc tính model với annotation `@ModelAttribute`.
 - Các giá trị cookie đính kèm trong request với annotation `@CookieValue`.
 - Map hoặc `ModelMap` để thêm các thuộc tính vào model.
 - `Errors` hoặc `BindingResult` để truy cập vào các kết buộc và kết quả kiểm tra (validation) cho đối tượng command.
 - `SessionStatus` để thông báo hoàn tất xử lý session.

- Sau khi xử lý xong, phương thức sẽ giao quyền điều khiển cho View thông qua giá trị trả về của phương thức.
- Phương thức xử lý có thể trả về giá trị kiểu String đại diện cho tên View hoặc void, trong trường hợp này View được chọn dựa trên tên phương thức hoặc tên controller.



DispatcherServlet

- **ViewResolver** dùng xác định các view được render để response cho một request từ client.

```
<bean id="viewResolver"  
class="org.springframework.web.servlet.view.InternalResource  
ViewResolver">  
  
    <property name="prefix" value="/WEB-INF/jsp/" />  
    <property name="suffix" value=".jsp" />  
  
</bean>
```



Web Application Context

- Trong các ứng dụng Spring, các đối tượng của ứng dụng tồn tại trong một container.
- Container dùng để tạo các đối tượng, kết hợp giữa các đối tượng và quản lý vòng đời các đối tượng, những đối tượng trong container gọi là Spring Managed beans.
- Container gọi là Application Context.



Web Application Context

- Container sử dụng **Dependency Injection (DI)** quản lý các đối tượng beans,
- Một thể hiện Application Context dùng tạo beans, kết hợp các beans thông qua cấu hình bean, và cung cấp beans khi có request từ client.
- Cấu hình bean được định nghĩa hoặc trong tập tin **XML**, hoặc **annotation** hoặc thông qua các **lớp Java**.



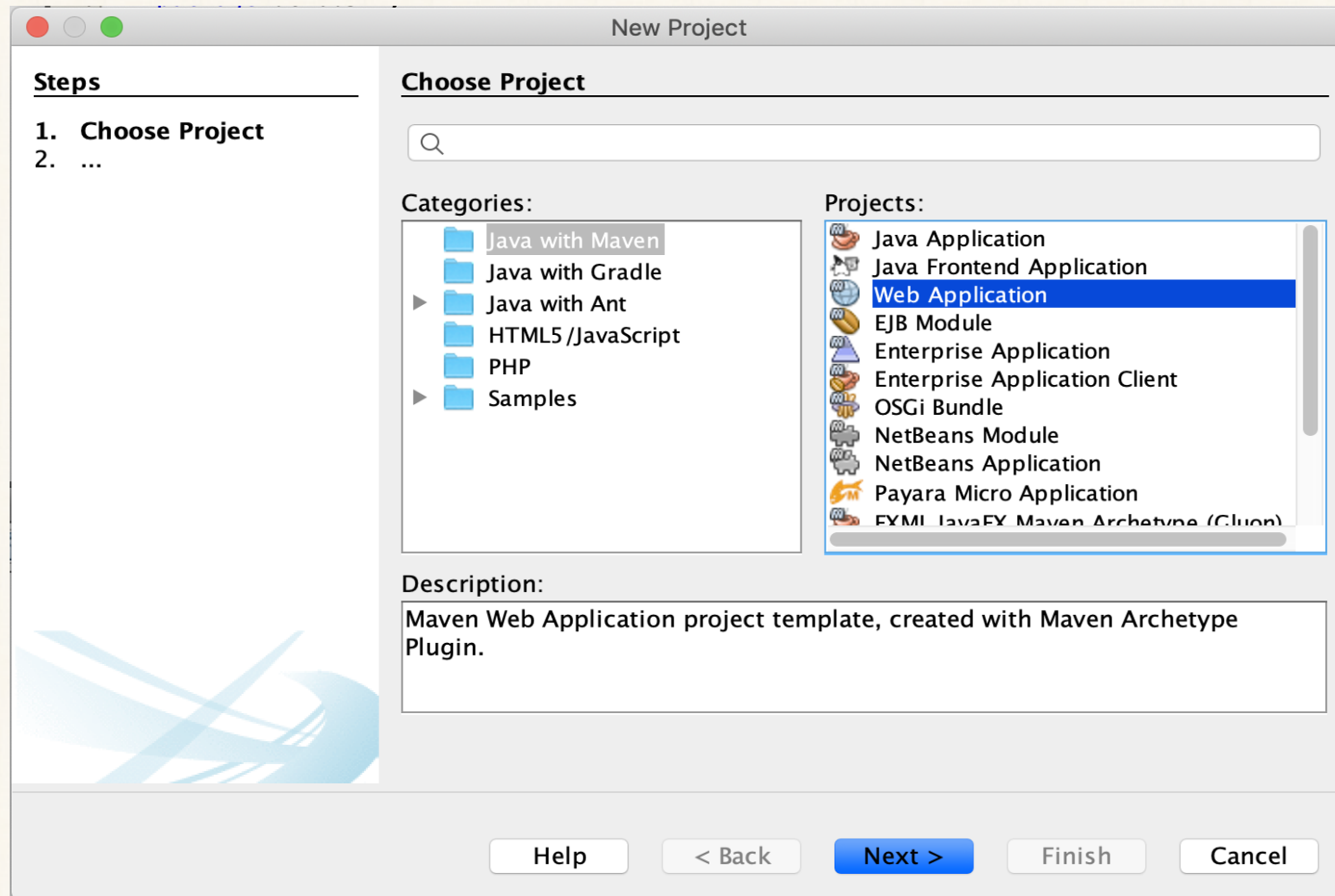
Chương trình đầu tiên SpringMVC

- **Cài bộ JDK (Java Development Toolkit).**
- **Cài đặt Apache Tomcat Server.**
- **Tạo project phát triển Web với Maven.**



Chương trình đầu tiên SpringMVC

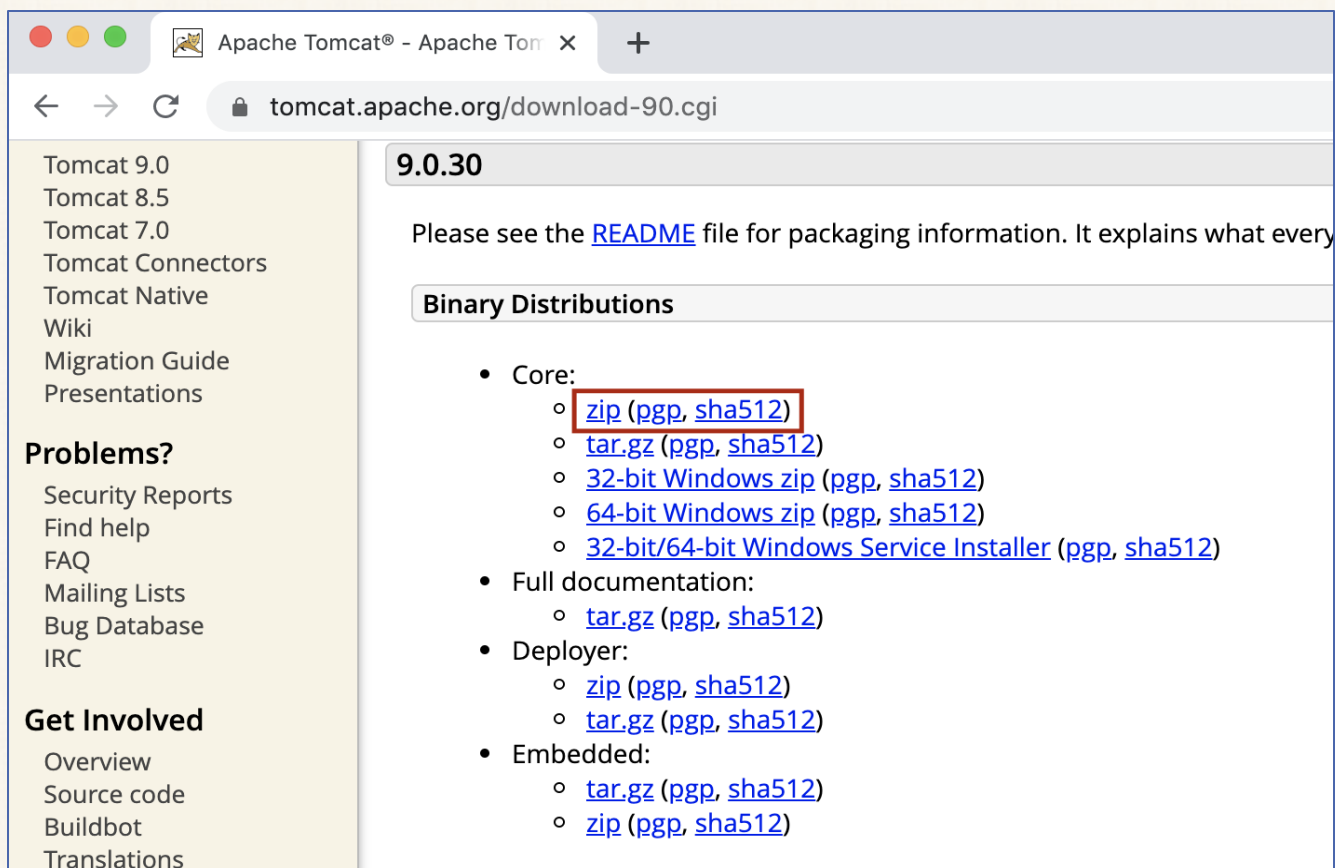
▪ Tạo project web maven





Chương trình đầu tiên SpringMVC

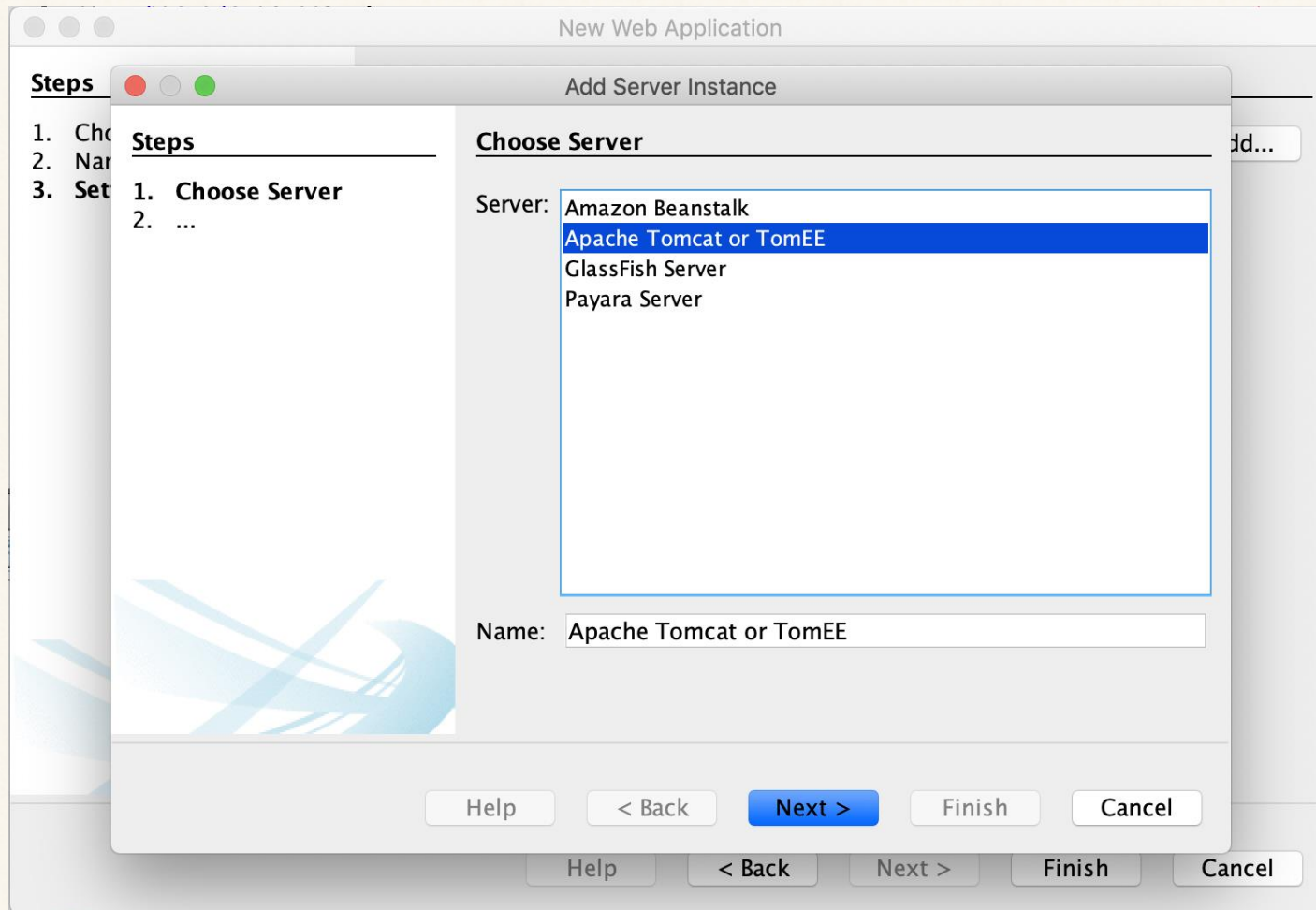
- Apache Tomcat là một Java Web Server phổ biến, tải Tomcat tại <http://tomcat.apache.org/>.





Chương trình đầu tiên SpringMVC

▪ Thiết lập Tomcat Server





Chương trình đầu tiên SpringMVC

▪ Thiết lập Tomcat Server

New Web Application

Steps Settings Add Server Instance

Steps

1. Choose Server
2. **Installation and Login Details**

Installation and Login Details

Specify the Server Location (Catalina Home) and login details

Server Location: /duonghuuthanh/Downloads/apache-tomcat-9.0.30 Browse ...

☐ Use Private Configuration Folder (Catalina Base)

Catalina Base: Browse ...

Enter the credentials of an existing user in the manager or manager-script role

Username: root

Password:

☒ Create user if it does not exist

Help < Back Next > Finish Cancel



Chương trình đầu tiên SpringMVC

- Điền thông tin tên project

New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Settings

Name and Location

Project Name:

Project Location:

Project Folder:

Artifact Id:

Group Id:

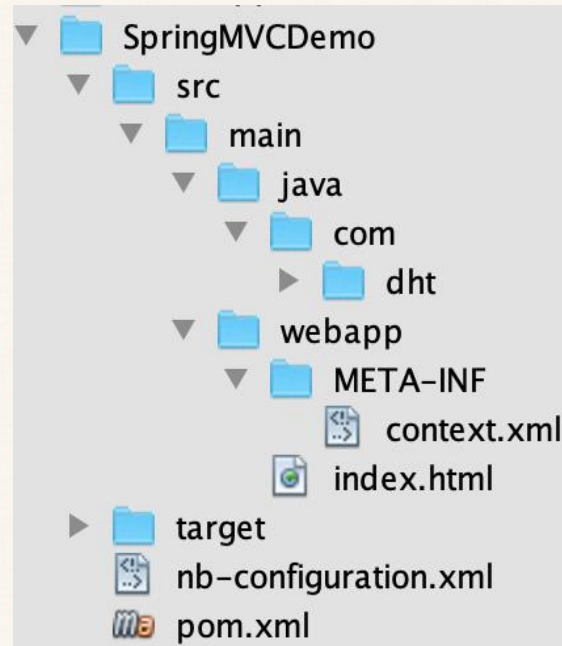
Version:

Package: (Optional)



Chương trình đầu tiên SpringMVC

- Cấu trúc project được tạo





Chương trình đầu tiên SpringMVC

- Thêm các dependencies vào pom.xml

```
<dependencies>
  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>7.0</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.2.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
```




Chương trình đầu tiên SpringMVC

- Trong thư mục webapp, tạo

/WEB-INF/applicationContext.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-
3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
3.0.xsd">
  <context:annotation-config />
</beans>
```



Chương trình đầu tiên SpringMVC

```
@Configuration
```

```
@EnableWebMvc
```

```
@ComponentScan(basePackages = "com.dht.springmvcdemo")
```

```
public class WebApplicationContextConfig implements  
WebMvcConfigurer {
```

```
    @Override
```

```
    public void configureDefaultServletHandling(  
        DefaultServletHandlerConfigurer configurer) {  
        configurer.enable();
```

```
    }
```

```
    @Bean
```

```
    public InternalResourceViewResolver  
        getInternalResourceViewResolver() {  
        InternalResourceViewResolver resolver  
            = new InternalResourceViewResolver();  
        resolver.setViewClass(JstlView.class);  
        resolver.setPrefix("/WEB-INF/jsp/");  
        resolver.setSuffix(".jsp");  
  
        return resolver;  
    }  
}
```



Chương trình đầu tiên SpringMVC

```
public class DispatcherServletInitializer
    extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] {
            WebApplicationContextConfig.class
        };
    }
    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```



Chương trình đầu tiên SpringMVC

- Tạo thư mục WEB-INF/jsp trong thư mục webapp, trong thư mục jsp tạo tập tin welcome.jsp như sau:

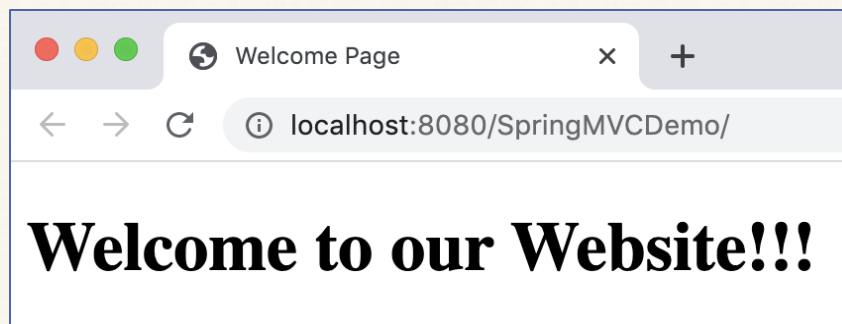
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome Page</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>
</html>
```




Chương trình đầu tiên SpringMVC

▪ Controller

```
@Controller
public class HomeController {
    @RequestMapping(value = "/")
    public String index(Model model) {
        model.addAttribute("message",
            "Welcome to our Website!!!");
        return "welcome";
    }
}
```





Controller

- Để định nghĩa controller, ta chỉ cần có các lớp Java kết hợp annotation **@Controller**.
- Các phương thức trong controller thường được gắn annotation **@RequestMapping** chỉ định đường dẫn URL sẽ ánh xạ phương thức đang viết.
- Ta cũng có thể sử dụng @RequestMapping cho lớp controller, khi đó Spring MVC sẽ xét các giá trị của @RequestMapping ở cấp lớp trước khi ánh xạ phần còn lại của URL vào phương thức xử lý.



Controller

- Mỗi lớp controller được phép chỉ định một **phương thức ánh xạ mặc định** (default mapping method), nó đơn giản là phương thức **không cần chỉ định** đường dẫn URL cho thuộc tính value của @RequestMapping, phương thức này được xem là phương thức ánh xạ mặc định cho lớp controller.



Controller

- @PathVariable
 - Để lấy giá trị tham số truyền trên đường dẫn URL của request sử dụng annotation @PathVariable.

```
@RequestMapping(value = "/list/{word}")
public String details(ModelMap model,
    @PathVariable(value = "word") String word) {
    String message = dicts.get(word);
    if (message == null)
        message = "Không có từ này!!!";
    model.addAttribute("message", message);
    return "dicts-detail";
}
```




- @RequestParam
 - Để lấy giá trị các tham số được truyền thông qua các tham số của HTTP GET.

```
@RequestMapping(value = "/search")
public String list(ModelMap model,
    @RequestParam(value = "word") String word) {
    Map<String, String> res = new HashMap<>();
    String des = dicts.get(word);
    if (des != null)
        res.put(word, des);
    model.addAttribute("words", res);
    return "dicts-list";
}
```



Controller

- Mặc định `@RequestParam` bắt buộc phải truyền tham số, ta có thể cho nó thành tùy chọn sử dụng thuộc tính `required=false`.
 - `@RequestParam(required=false, defaultValue="")`
- Lấy nhiều request param cùng lúc

```
@GetMapping("/test")
public String test(@RequestParam Map<String,String> allParams) {
    return "Parameters are " + allParams.entrySet();
}
```



Tag Libraries

- JavaServer Page (JSP) là công nghệ cho phép nhúng mã nguồn Java vào các trang HTML, mã nguồn Java được chèn giữa cặp dấu `<% %>` hoặc thông qua các thẻ của JSTL (JavaServer Pages Standard Tag Library).
- JSTL là thư viện các thẻ chuẩn được cung cấp bởi Oracle. Để sử dụng thư viện JSTL trong các trang JSP cần chỉ định nó thông qua **taglib** (taglib directives).



Tag Libraries

- Taglib khai báo các trang JSP sử dụng tập các thẻ thư viện của JSTL và **chỉ định vị trí** của thư viện bằng thuộc tính **uri**, thuộc tính **prefix** chỉ tiền tố khi sử dụng các thẻ trong thư viện chỉ định.

```
<%@ taglib prefix="c"
        uri="http://java.sun.com/jsp/jstl/core"%>
```




Tag Libraries

- Spring MVC cũng cung cấp thư viện thẻ riêng giúp cho việc phát triển các view JSP được dễ dàng hơn, để sử dụng các thư viện này ta

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form" %>

<%@ taglib prefix="spring"
    uri="http://www.springframework.org/tags" %>
```



Tag Libraries

▪ Ví dụ sử dụng modelAttribute

```
@RequestMapping(value = "/add")
public String addWordView(ModelMap model) {
    Word w = new Word();
    model.addAttribute("word", w);
    return "dicts-add-word";
}

@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
    @ModelAttribute(value = "word") Word newWord) {
    if (dicts.get(newWord.getWord()) == null) {
        dicts.put(newWord.getWord(), newWord.getDescription());
        return "redirect:/dicts/list";
    } else {
        model.addAttribute("message", "Từ đã tồn tại!!!");
        return "dicts-add-word";
    }
}
```



Tag Libraries

▪ Tập tin jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form" %>
<html>
  <head>
    <meta charset="UTF-8">
    <title>My Dictionary</title>
  </head>
  <body>
    <form:form method="POST" modelAttribute="word">
      <form:input id="wordId" path="word" />
      <form:input id="desId" path="description" />
      <input type="submit" value="Thêm từ" />
    </form:form>
  </body>
</html>
```

- Trong thẻ `<form:form>` ngoài việc khai báo thuộc tính `method` là `POST`, thì một thuộc tính quan trọng khác được khai báo là **`modelAttribute`** có giá trị là `"word"`, đây là tên thuộc tính dùng lưu trữ đối tượng `Word` mới được tạo trong phương thức `addWordView()` (đối tượng này được gọi là **`Backing Bean`** trong Spring MVC). Trong các thẻ `<form:input>` bên trong `<form:form>` có thuộc tính quan trọng là **`path`**, giá trị của thuộc tính này là tên trường của đối tượng `Backing Bean`, nên giá trị được nhập vào form này sẽ được kết buộc vào trường tương ứng trong Bean.



Tag Libraries

- Trong các phần trước, ta đã sử dụng `@ModelAttribute` để kết buộc tham số của phương thức trong controller sử dụng trong các Web View.
- Trong minh họa này sử dụng `@ModelAttribute` cho phương thức trong controller, các phương thức được gắn annotation này sẽ được gọi trước tất cả các phương thức `RequestMapping` khác, mục đích của ta là thêm các **thuộc tính chung** vào model trước khi gọi các phương thức `RequestMapping`



Tag Libraries

```
@Controller
@ControllerAdvice
public class CommonController {
    // ...

    @ModelAttribute
    public void addAttributes(Model model) {
        model.addAttribute("message", "common message");
    }
    // ...
}
```

Một số thẻ thông dụng

- Một số thẻ thông dụng
 - `<c:set>`
 - `<c:out>`
 - `<c:if test= var= scope=>`

```
<c:set var = "salary"
      scope = "session" value = "${2000*2}"/>

<c:if test = "${salary > 2000}">
  <p>My salary is:  <c:out value = "${salary}"/><p>
</c:if>
```

Một số thẻ thông dụng

- `<c:forEach items= var= begin= end=>`

```
<c:forEach var="i" begin="1" end="5">  
    Item <c:out value = "${i}"/><p>  
</c:forEach>
```

- `<c:forTokens>`

```
<c:forTokens items = "Apple,Banana,Lemon"  
    delims = ", " var = "fruit">  
    <c:out value = "${fruit}"/><p>  
</c:forTokens>
```

Một số thẻ thông dụng

- `<c:choose></c:choose>`
- `<c:when test=></c:when>`
- `<c:otherwise></c:otherwise>`

```
<c:choose>  
  <c:when test=></c:when>  
  <c:otherwise></c:otherwise>  
</c:choose>
```



Một số thẻ thông dụng

- `<c:url value= var=>`: tạo URL với query params
- `<c:param name= value=>`
- `<c:import>`

```
<c:url value = "/index.jsp" var = "myURL">  
  <c:param name = "firstName" value = "Thanh"/>  
  <c:param name = "lastName" value = "Duong"/>  
</c:url>  
<c:import url = "${myURL}"/>a
```




WebDataBinder

- WebDataBinder dùng lấy dữ liệu từ đối tượng `HttpServletRequest`, chuyển nó thành định dạng dữ liệu thích hợp, nạp nó vào đối tượng Backing Bean và kiểm tra dữ liệu (validate).
- Để điều chỉnh cách thức kết buộc dữ liệu (data binding), ta cần khởi động và cấu hình đối tượng WebDataBinder trong controller.
- Annotation `@InitBinder` dùng để chỉ định phương thức khởi động WebDataBinder.

- Trong controller

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    binder.setAllowedFields("word", "description");
}
```

- Phương thức action trong controller

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
    @ModelAttribute(value = "word") Word newWord,
    BindingResult result) {
    if (result.getSuppressedFields().length > 0)
        throw new RuntimeException("disallowed fields!!!");
    ...
}
```



Properties File

- Trong ví dụ trên các nhãn hiển thị trên trang web đều là hard-code trực tiếp từ trong tập tin .jsp,
- Điều này thiếu linh hoạt khi ta cần chỉnh sửa nội dung hiển thị trang web, cũng như khi cần phát triển trang web đa ngôn ngữ.

```
<%@ taglib prefix="spring"
uri="http://www.springframework.org/tags" %>
<form:form method="post" modelAttribute="word">
  <spring:message code="label.word" />
  <form:input id="wordId" path="word" /> <br />
  <input type="submit" value="Thêm từ" />
</table>
<form:form>
```



Properties File

- Thẻ `<spring:message>` chỉ định văn bản từ ngoài được điền vào khi chương trình thực thi, để sử dụng thẻ này ta cần thêm thư viện Spring Tag.

```
<%@ taglib prefix="spring"
      uri="http://www.springframework.org/tags" %>
```

- Trong thư mục `src/main/resources` tạo tập tin `messages.properties` có nội dung:

```
label.word=Từ mới (tiếng Anh)
label.description=Nghĩa của từ (tiếng Việt)
```



Properties File

- Để kết nối thông tin từ tập tin properties và trên JSP view, ta cần cấu hình Bean cho lớp ResourceBundleMessageSource với tên **messageSource**, trong đó thuộc tính basename chỉ định giá trị là tên của tập tin property.

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource resource
        = new ResourceBundleMessageSource();
    resource.setBasename("messages") ;
    // resource.setBasenames("messages1", "messages2");
    return resource;
}
```




View Resolver

- Redirect là kỹ thuật chuyển người dùng đến một trang khác với trang web đang request.
- Kỹ thuật này thường được sử dụng sau khi submit một web form để hạn chế người dùng submit lại form tương tự khi bấm nút Back hoặc Refresh trên trình duyệt.
- Để sử dụng RedirectView để xử lý chuyển trang trong controller, ta chỉ cần trả về chuỗi URL với phần tiền tố (prefix) chuyển trang, có hai tiền tố được sử dụng để chuyển trang: forward và redirect.

- Ví dụ

```
@RequestMapping(value = "/hello2")
public String hello2(ModelMap model) {
    return "hello";
}

@RequestMapping(value = "/hello1")
public String hello1(Model model) {
    model.addAttribute("message",
        "Welcome to our Website!!!");
    return "forward:/hello2";
}
```



View Resolver

- **forward**: Spring chuyển request hiện tại đến một phương thức request mapping khác dựa trên đường dẫn sau tiền tố forward, request được chuyển tới **vẫn là request gốc ban đầu**, nên những giá trị được đặt vào model khi bắt đầu request vẫn còn giá trị.
- **redirect**: Spring sẽ tạo một **request mới**, nên những giá trị đặt vào model khi bắt đầu request hiện tại sẽ mất đi.



View Resolver

- Static Resource

- Tạo thư mục `src/main/webapp/resources/images` và sao chép tập tin ảnh nào đó có tên `java.jpg` vào thư mục này.
- Ghi đè phương thức `addResourceHandlers()` trong `WebApplicationContextConfig.java` để chỉ định vị trí chứa các tài nguyên.

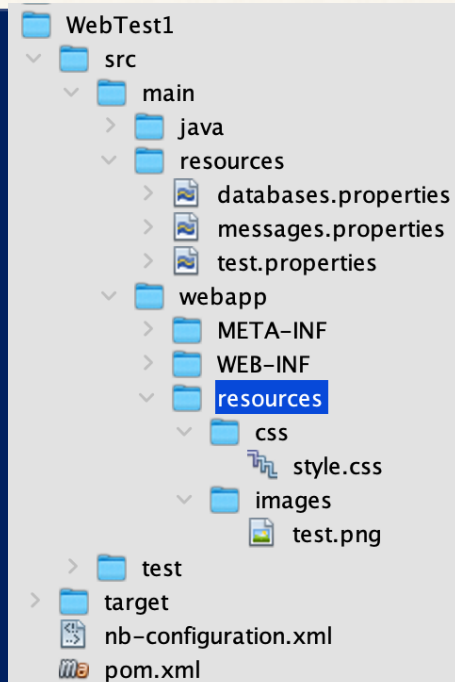
```
@Override
public void addResourceHandlers(
    ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/css/**")
        .addResourceLocations("/resources/css/");
    registry.addResourceHandler("/img/**")
        .addResourceLocations("/resources/images/");
}
```



View Resolver

- Sử dụng static resource trong jsp

```
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <link href="<c:url value="/css/style.css"/>"
              rel="stylesheet" />
    </head>
    <body>
        " />
    </body>
</html>
```





CommonsMultipartResolver

- **Multipart request** là một loại HTTP Request để gửi các tập tin và dữ liệu đến server.
- Lớp **CommonsMultipartResolver** quyết định một request có được phép chứa nội dung multipart và chuyển HTTP request thành các các tham số và tập tin multipart hay không.

```
@Bean
public CommonsMultipartResolver multipartResolver() {
    CommonsMultipartResolver resolver
        = new CommonsMultipartResolver();
    resolver.setDefaultEncoding("UTF-8");

    return resolver;
}
```



CommonsMultipartResolver

- Các dependency cần thiết

```
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.6</version>
</dependency>
```



CommonsMultipartResolver

- Bổ sung thuộc tính trong pojo

```
private MultipartFile img;
```

- Tập tin jsp

```
<form:form method="POST" modelAttribute="word"
           enctype="multipart/form-data">
  <table>
    ...
    <tr>
      <td><spring:message code="label.image" /></td>
      <td><form:input id="imageId" path="img"
                    type="file" /></td>
    </tr>
    ...
  </table>
</form:form>
```



CommonsMultipartResolver

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String addWordProcess(ModelMap model,
                             @ModelAttribute(value = "word") Word newWord,
                             HttpServletRequest request) {
    ...
    MultipartFile img = newWord.getImg();
    String rootDir = request.getSession()
                           .getServletContext().getRealPath("/");
    if (img != null && !img.isEmpty()) {
        try {
            img.transferTo(new File(rootDir + "resources/images/"
                                   + newWord.getWord() + ".png"));
        } catch (IOException | IllegalStateException ex) {
            System.err.println(ex.getMessage());
        }
    }
    ...
}
```



Upload lên cloudinary

- Đầu tiên đăng ký người dùng tại trang bên dưới để nhận các thông tin cloud name, API key, API secret: <https://cloudinary.com/users/register/free>
- Thêm dependency sau vào pom.xml

```
<dependency>
  <groupId>com.cloudinary</groupId>
  <artifactId>cloudinary-http44</artifactId>
  <version>1.29.0</version>
</dependency>
<dependency>
  <groupId>com.cloudinary</groupId>
  <artifactId>cloudinary-taglib</artifactId>
  <version>1.29.0</version>
</dependency>
```


Upload lên clouldinary

- Tạo bean

```
@Bean
public Cloudinary cloudinary() {
    Cloudinary cloudinary
        = new Cloudinary(ObjectUtils.asMap(
            "cloud_name", "...",
            "api_key", "...",
            "api_secret", "...",
            "secure", true));

    return cloudinary;
}
```



Upload lên clouldinary

- Xử lý upload trong controller

```
@Autowired
private Clouldinary clouldinary;

@RequestMapping(path="/upload", method = RequestMethod.POST)
public String upload(@ModelAttribute("user") User user) {
    try {
        clouldinary.uploader().upload(user.getAvatar().getBytes(),
            ObjectUtils.asMap("resource_type", "auto"));
    } catch (IOException ex) {
        System.out.println(ex.getMessage());
    }

    return "index";
}
```



Upload lên clouldinary

- Lớp User

```
public class User {  
    private MultipartFile avatar;  
}
```

- Form upload

```
<c:url value="/upload" var="action" />  
<form:form action="${action}"  
            method="POST" modelAttribute="user"  
            enctype="multipart/form-data">  
    <form:input id="imageId" path="avatar" type="file" />  
    <input type="submit" value="upload" />  
</form:form>
```



Hibernate Config

- Sử dụng dependency

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.4.10.Final</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.18</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>5.2.3.RELEASE</version>
</dependency>
```



Hibernate Config

- Tạo thư mục resources trong thư mục src/main, trong thư mục này tạo tập tin database.properties có nội dung như sau:

```
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.showSql=true
hibernate.connection.driverClass=com.mysql.cj.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/saledb
hibernate.connection.username=root
hibernate.connection.password=12345678
```

- Trong gói com.dht.config chứa các tập tin cấu hình bằng mã nguồn Java.
- Tạo tập tin HibernateConfig.java.



Hibernate Config

```
@Configuration
```

```
@PropertySource("classpath:database.properties")
```

```
public class HibernateConfig {
```

```
    @Autowired
```

```
    private Environment env;
```

```
    @Bean
```

```
    public LocalSessionFactoryBean getSessionFactory() {
```

```
        LocalSessionFactoryBean sessionFactory
```

```
            = new LocalSessionFactoryBean();
```

```
        sessionFactory.setPackagesToScan(new String[] {
```

```
            "com.dht.model"
```

```
        });
```

```
        sessionFactory.setDataSource(dataSource());
```

```
        sessionFactory.setHibernateProperties(hibernateProperties());
```

```
        return sessionFactory;
```

```
    }
```

```
    @Bean
```

```
    public DataSource dataSource() {}
```

```
    private Properties hibernateProperties() {}
```

```
}
```



Hibernate Config

```
@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource
        = new DriverManagerDataSource();
    dataSource.setDriverClassName(
        env.getProperty("hibernate.connection.driverClass"));
    dataSource.setUrl(env.getProperty("hibernate.connection.url"));
    dataSource.setUsername(
        env.getProperty("hibernate.connection.username"));
    dataSource.setPassword(
        env.getProperty("hibernate.connection.password"));
    return dataSource;
}

private Properties hibernateProperties() {
    Properties props = new Properties();
    props.put(DIALECT, env.getProperty("hibernate.dialect"));
    props.put(SHOW_SQL, env.getProperty("hibernate.showSql"));
    return props;
}
```



Hibernate Config

- @Bean dataSource(): việc tạo kết nối đến cơ sở dữ liệu tốn nhiều thời gian, đặc biệt trong môi trường mạng, nên rất cần thiết cho **việc tái sử dụng**, cũng như chia sẻ sử dụng các kết nối đã mở (**connection pool**). Việc tạo Bean dataSource có nhiệm vụ **tối ưu** việc sử dụng các kết nối này.
- @Bean sessionFactory sử dụng LocalSessionFactoryBean tạo sessionFactory.



Hibernate Config

- Quản lý giao tác là kỹ thuật lập trình quan trọng trong phát triển các ứng dụng thương mại để đảm bảo tính **nhất quán** và **toàn vẹn dữ liệu**.
- **HibernateTransactionManager** kết buộc Session từ một SessionFactory vào một thread, cho phép một Session cho mỗi SessionFactory.



Hibernate Config

- Thêm phương thức trong **HibernateConfig**

```
@Bean
public HibernateTransactionManager transactionManager() {
    HibernateTransactionManager transactionManager
        = new HibernateTransactionManager();
    transactionManager.setSessionFactory(
        getSessionFactory().getObject());
    return transactionManager;
}
```

- Tập tin DispatcherServletInitializer.java:

```
@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] {
        HibernateConfig.class
    };
}
```




Hibernate Config

- Tập tin WebApplicationContextConfig.java

```
@Configuration
@EnableWebMvc
@EnableTransactionManagement
@ComponentScan(basePackages = "com.dht.controller")
public class WebApplicationContextConfig
    implements WebMvcConfigurer {

}
```

- **@EnableTransactionManagement** cho phép khả năng sử dụng **quản lý giao tác** thông qua annotation của Spring.

Template with Tiles

- Apache Tiles là một framework mã nguồn mở giúp **tái sử dụng tối đa** khi xây dựng các front-end template.
- Tiles cho phép lập trình viên định nghĩa các phần con (tiles) để lắp ráp thành một trang web hoàn chỉnh khi ứng dụng thực thi, những phần con này có các tham số với giá trị có thể thay đổi khi chương trình thực thi.

```
<dependency>  
  <groupId>org.apache.tiles</groupId>  
  <artifactId>tiles-extras</artifactId>  
  <version>3.0.8</version>  
</dependency>
```

Template with Tiles

- Tạo thư mục **WEB-INF/**, trong thư mục này tạo tập tin **tiles.xml** như bên dưới.
- Tập tin **tiles.xml** là tập tin rất quan trọng trong phát triển ứng dụng dựa trên Apache Tiles.
- Mỗi định nghĩa là thẻ **<definition>**
 - Thuộc tính được chỉ định bằng thẻ **<put-attribute>** bên trong **<definition>**.
 - Giá trị của các thuộc tính này chèn vào template bằng **<tiles:insertAttribute name="">**.
- Định nghĩa kế thừa một định nghĩa khác thông qua thuộc tính **extends** trong thẻ **<definition>**



Template with Tiles

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC "-//Apache
    Software Foundation//DTD Tiles Configuration 3.0//EN"
    "http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
<tiles-definitions>
  <definition name="baseLayout"
    template="/WEB-INF/layout/base.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header"
      value="/WEB-INF/layout/header.jsp" />
    <put-attribute name="content" value="" />
    <put-attribute name="footer"
      value="/WEB-INF/layout/footer.jsp" />
  </definition>
  <definition name="index" extends="baseLayout">
    <put-attribute name="title" value="Trang chủ" />
    <put-attribute name="content"
      value="/WEB-INF/welcome.jsp" />
  </definition>
</tiles-definitions>
```



Template with Tiles

- header.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
  <ul class="navbar-nav">
    <li class="nav-item active">
      <a class="nav-link" href="#">Trang chủ</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="#">Danh mục</a>
    </li>
  </ul>
</nav>
```




Template with Tiles

- footer.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<section>
    <div class="jumbotron">
        <h1>Apache Tiles</h1>
        <p>SpringMVC DemoApp...</p>
    </div>
</section>
```

- index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<h1 class="text-center text-danger">TRANG CHỦ</h1>
```



Template with Tiles

- base.jsp

```
<%@ taglib prefix="tiles"
        uri="http://tiles.apache.org/tags-tiles" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="bootstrap.min.css" />
    <title><tiles:insertAttribute name="title" /></title>
  </head>
  <body>
    <tiles:insertAttribute name="header" />
    <tiles:insertAttribute name="content" />
    <tiles:insertAttribute name="footer" />
  </body>
</html>
```



Template with Tiles

@Configuration

```
public class TilesConfig {
    @Bean
    public UrlBasedViewResolver viewResolver() {
        UrlBasedViewResolver viewResolver
            = new UrlBasedViewResolver();
        viewResolver.setViewClass(TilesView.class);
        viewResolver.setOrder(-2);
        return viewResolver;
    }

    @Bean
    public TilesConfigurer tilesConfigurer() {
        TilesConfigurer configurer = new TilesConfigurer();
        configurer.setDefinitions("/WEB-INF/tiles.xml");
        configurer.setCheckRefresh(true);
        return configurer;
    }
}
```

- Chỉ định thông tin beans

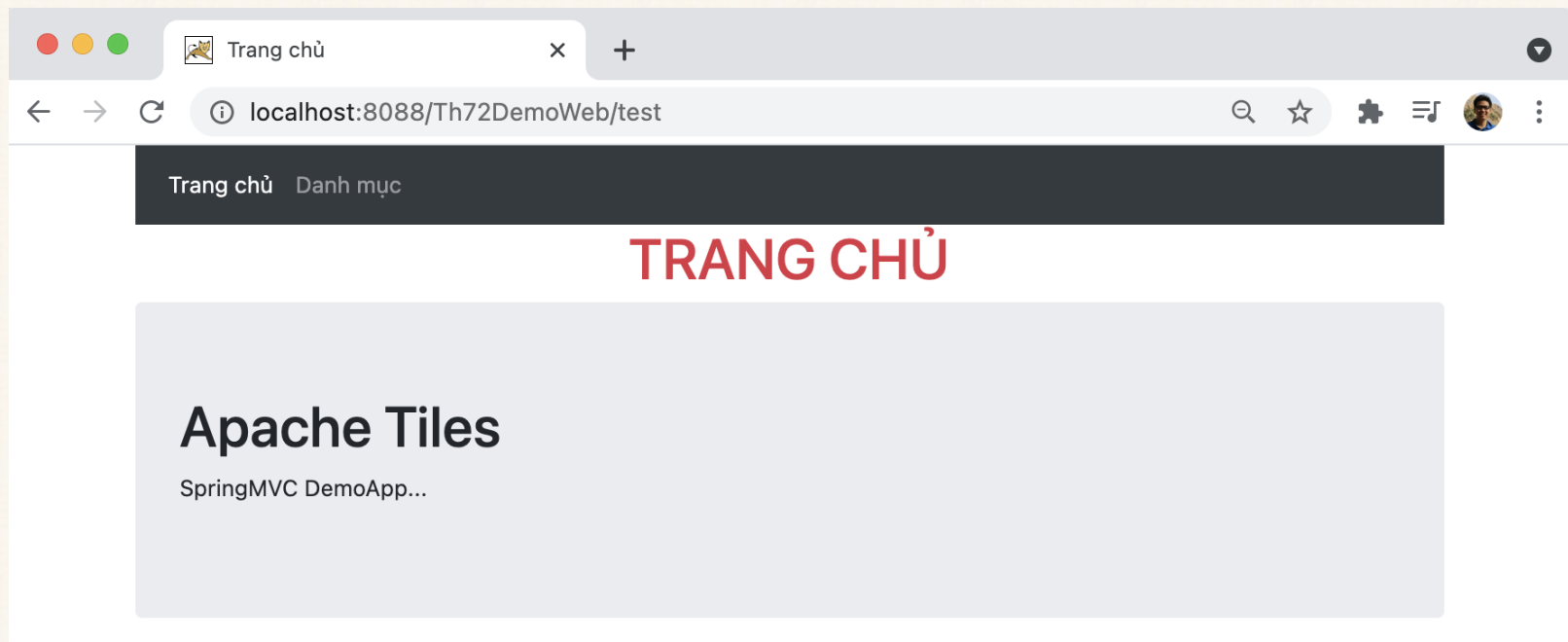
```
@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] {
        ..., TilesConfig.class
    };
}
```

- Trong controller

```
@Controller
public class HomeController {
    @RequestMapping("/test")
    public String test(Model model) {
        return "index";
    }
}
```

Template with Tiles

- Truy cập
 - <http://localhost:8088/WebAppName/test>





Bean Validation

- **Bean Validation** cho phép mô tả các ràng buộc trên các đối tượng thông qua các annotation.
- Ta sử dụng Hibernate Validator để kiểm tra một số dữ liệu đầu vào của chức năng thêm sản phẩm.
- Dependency hibernate-validator

```
<dependency>  
  <groupId>org.hibernate.validator</groupId>  
  <artifactId>hibernate-validator</artifactId>  
  <version>6.1.1.Final</version>  
</dependency>
```



Bean Validation

- Trong tập tin pojo (persistent class) thiết lập các ràng buộc thông qua annotation.

```
@Entity
@Table(name = "product")
public class Product implements Serializable {
    @Size(min=10, max=50,
        message="{product.name.sizeMsg}")
    private String name;
    @NotNull(message="{product.price.notNullMsg}")
    @Min(value=100000, message="{product.price.minMsg}")
    @Max(value=1000000000, message="{product.price.maxMsg}")
    private BigDecimal price;
    @ManyToOne
    @JoinColumn(name="category_id")
    @NotNull(message="{product.category.notNullMsg}")
    private Category category;
}
```



Bean Validation

- Gói **javax.validation.constraints** chứa annotation dùng để thiết lập kiểm tra dữ liệu như @Size, @Max, @Min, @NotNull.
- Thuộc tính message của mỗi annotation chỉ định nội dung lỗi sẽ được hiển thị khi dữ liệu vi phạm ràng buộc chỉ định và các nội dung này được cấu hình lấy từ **messages.properties**.

```
product.price.notNullMsg=Phải có giá sản phẩm  
product.price.minMsg=Giá sản phẩm tối thiểu 100.000 VNĐ  
product.price.maxMsg=Giá sản phẩm tối đa 1 tỷ VNĐ  
product.name.sizeMsg=Tên sản phẩm tối thiểu 10 và tối đa 50 ký tự  
product.category.notNullMsg=Phải chọn danh mục cho sản phẩm  
product.image.notNullMsg=Phải có ảnh đại diện sản phẩm
```



Bean Validation

- Chỉnh sửa controller tương ứng

```
import javax.validation.Valid;
...
@PostMapping(value = "/products/add")
public String addProductProcess(Model model,
    @ModelAttribute(value = "product") @Valid Product product,
    BindingResult result, HttpServletRequest request) {
    if(result.hasErrors()) {
        ...
        return "add-product";
    }
    ...
    return "redirect:/";
}
```



Bean Validation

- Hiển thị thông tin lỗi tại view

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form" %>

<form:form action="${action}" modelAttribute="product"
        method="post" enctype="multipart/form-data" >
    <form:errors path="*" element="div" />
    ...
    <div class="form-group">
        <form:input id="priceId" path="price" />
        <form:errors path="price" cssClass="text-danger" />
    </div>
    <div class="form-group">
        <form:button class="pull-right">Submit</form:button>
    </div>
</form:form>
```




Bean Validation

- Các thẻ `<form:errors>`, trong đó thuộc tính `path` chỉ định tên thuộc tính của đối tượng model khi dữ liệu cung cấp cho nó vi phạm ràng buộc.
- Để dòng `<form:errors>` đầu tiên thuộc tính `path` có giá trị là `*` chỉ định hiển thị lỗi tất cả các trường nếu có.



Bean Validation

- Thiết lập cấu hình cho phép kiểm tra dữ liệu, bổ sung cấu hình LocalValidatorFactoryBean

```
@Bean(name = "validator")
public LocalValidatorFactoryBean validator() {
    LocalValidatorFactoryBean bean
        = new LocalValidatorFactoryBean();
    bean.setValidationMessageSource(messageSource());
    return bean;
}

@Override
public Validator getValidator(){
    return validator();
}

@Override
public void addFormatters(FormatterRegistry registry) {
    registry.addFormatter(new CategoryFormatter());
}
```



Bean Validation

- Lớp CategoryFormatter

```
public class CategoryFormatter implements Formatter<Category> {  
    @Override  
    public String print(Category obj, Locale locale) {  
        return String.valueOf(obj.getId());  
    }  
    @Override  
    public Category parse(String text, Locale locale)  
        throws ParseException {  
        Category c = new Category();  
        c.setId(Integer.parseInt(text));  
  
        return c;  
    }  
}
```



Bean Validation

Thêm sản phẩm

localhost:8080/saleweb/products/add

THÊM SẢN PHẨM

Tên sản phẩm tối thiểu 10 và tối đa 50 ký tự
Mô tả chi tiết tối đa 255 ký tự
Phải có giá sản phẩm
Nhà sản xuất tối đa 50 ký tự

Danh mục sản phẩm

Điện thoại thông minh

Tên sản phẩm

Test

Tên sản phẩm tối thiểu 10 và tối đa 50 ký tự

Giá sản phẩm

Phải có giá sản phẩm

Nhà sản xuất

aa

Nhà sản xuất tối đa 50 ký tự

Mô tả chi tiết sản phẩm

aa
aa

Mô tả chi tiết tối đa 255 ký tự

Ảnh sản phẩm

Choose File No file chosen

THÊM SẢN PHẨM

Spring Web MVC © 2020.



Bean Validation

- Tự tạo một bean validation

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
@Constraint(validatedBy = ProductNameValidator.class)
@Documented
public @interface ProductName {
    String message() default "";
    Class<?>[] groups() default {};
    public abstract Class<? extends Payload>[] payload() default {};
}
```




Bean Validation

```
public class ProductNameValidator
    implements ConstraintValidator<ProductName, String> {
    @Override
    public void initialize(ProductName constraintAnnotation) {
    }
    @Override
    public boolean isValid(String value,
        ConstraintValidatorContext context) {
        try {
            return productService.checkProductName(value);
        } catch (NoResultException ex) {
            return false;
        }
    }
}
```



Bean Validation

- Gắn bean validation vừa tạo vào persistence class.

```
@Entity
@Table(name = "product")
public class Product implements Serializable {
    // ...
    @Column(name = "name")
    @Size(min=10, max=50, message="{product.name.error.sizeMsg}")
    @ProductName(message="{product.name.error.productNameMsg}")
    private String name;
    //
}
```



Bean Validation

Thêm sản phẩm

localhost:8080/saleweb/products/add

Giỏ hàng

THÊM SẢN PHẨM

Tên sản phẩm này đã tồn tại

Danh mục sản phẩm
Điện thoại thông minh

Tên sản phẩm
Galaxy Note 10
Tên sản phẩm này đã tồn tại

Giá sản phẩm
10000000

Nhà sản xuất
Samsung

Mô tả chi tiết sản phẩm
256GB

Ảnh sản phẩm
Choose File No file chosen

THÊM SẢN PHẨM

Spring Web MVC © 2020.



Spring Validation

- Spring cũng cung cấp một cơ chế cổ điển cho phép kiểm tra các dữ liệu đầu vào gọi là Spring Validation. So với Bean Validation thì Spring Validation linh hoạt và dễ mở rộng hơn.
- Trong gói `com.dht.validator` tạo lớp `PriceValidator` **hiện thực giao diện `Validator`**, giao diện này có hai phương thức quan trọng:
 - **`supports()`** cho biết validator có được phép kiểm tra lớp chỉ định không.
 - **`validate()`** là phương thức được gọi để kiểm tra dữ liệu của đối tượng lớp.



Spring Validation

- Tạo lớp Validator

@Component

```
public class PriceValidator implements Validator {  
    @Override  
    public boolean supports(Class<?> clazz) {  
        return Product.class.isAssignableFrom(clazz);  
    }  
    @Override  
    public void validate(Object target, Errors errors) {  
        Product product = (Product) target;  
  
        if (product.getPrice().compareTo(new BigDecimal(5000000)) < 0)  
            errors.rejectValue("price",  
                                "product.price.priceValidatorMsg");  
    }  
}
```




Spring Validation

- Validate trong controller

```
@Controller
public class ProductController {
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private ProductService productService;
    @Autowired
    private PriceValidator priceValidator;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.setValidator(priceValidator);
    }
    // ...
}
```



Spring Validation

Thêm sản phẩm

localhost:8080/saleweb/products/add

Giỏ hàng

THÊM SẢN PHẨM

Máy tính xách tay phải có giá ít nhất là 5 triệu

Danh mục sản phẩm
Điện thoại thông minh

Tên sản phẩm
Test12345

Giá sản phẩm
4900000
Máy tính xách tay phải có giá ít nhất là 5 triệu

Nhà sản xuất
Dell

Mô tả chi tiết sản phẩm
2020

Ảnh sản phẩm
Choose File No file chosen

THÊM SẢN PHẨM

Spring Web MVC © 2020.



Spring Validation

- Vì ta đã sử dụng Spring Validator để kiểm tra dữ liệu, nên những validator đã thiết lập trước đó dựa trên Bean Validator sẽ không còn tác dụng, Spring MVC sẽ **bỏ qua** các annotation như @Min, @Max, @Size.
- Để có thể kết hợp Spring Validation và Bean Validation ta có thể tạo lớp WebAppValidator.

@Component

```
public class WebAppValidator implements Validator {
```

```
    @Autowired
```

```
    private javax.validation.Validator beanValidator;
```

```
    private Set<Validator> springValidators = new HashSet<>();
```

```
    @Override
```

```
    public boolean supports(Class<?> clazz) {
```

```
        return Product.class.isAssignableFrom(clazz);
```

```
    }
```

```
    @Override
```

```
    public void validate(Object target, Errors errors) {
```

```
        Set<ConstraintViolation<Object>> constraintViolations  
            = beanValidator.validate(target);
```

```
        for (ConstraintViolation<Object> obj: constraintViolations)  
            errors.rejectValue(obj.getPropertyPath().toString(),  
                               obj.getMessageTemplate(), obj.getMessage());
```

```
        for (Validator obj: springValidators)
```

```
            obj.validate(target, errors);
```

```
    }
```

```
    public void setSpringValidators(
```

```
        Set<Validator> springValidators) {
```

```
        this.springValidators = springValidators;
```

```
    }
```

```
}
```



Spring Validation

- Trong lớp `WebApplicationContextConfig` tạo một Beans cho `WebAppValidator` như sau:

```
@Bean
public WebAppValidator productValidator() {
    Set<Validator> springValidators = new HashSet<>();
    springValidators.add(new PriceValidator());

    WebAppValidator validator = new WebAppValidator();
    validator.setSpringValidators(springValidators);

    return validator;
}
```




Spring Validation

- Trong controller thay thuộc tính priceValidator bằng productValidator:

```
@Controller
public class ProductController {
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private ProductService productService;
    @Autowired
    private WebAppValidator productValidator;

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.setValidator(productValidator);
    }
    // ...
}
```



Spring Security

- Spring Security quan tâm đến các đối tượng `HttpRequest` và `HttpResponse`, một request có thể thực hiện thông qua trình duyệt Web, Web Service, HTTP client hoặc thực hiện bằng Ajax.
- Spring Security cung cấp các **Servlet Filter** xây dựng sẵn và chỉ cần cấu hình các filter thích hợp cho ứng dụng Web để **kiểm tra các HTTP request** trước khi thực hiện công việc nào đó.

- Các dependencies

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>6.3.4</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>6.3.4</version>
</dependency>
```

```

@Entity
@Table(name = "user")
public class User implements Serializable {
    private static final long serialVersionUID = 3L;
    public static final String USER = "ROLE_USER";
    public static final String ADMIN = "ROLE_ADMIN";
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    ...
    @Pattern(regex = "^[A-Za-z0-9+_.-]+@(.+)$",
        message = "{user.email.error.invalidMsg}")
    private String email;
    @Column(name = "user_role")
    private String userRole;
    @Size(min = 1, max = 45, message = "{user.username.sizeMsg}")
    private String username;
    @NotEmpty(message = "{user.password.sizeMsg}")
    private String password;
    @Transient
    private String confirmPassword;
    // Các phương thức getter/setter
}

```



▪ Tạo PassValidator

@Component

```
public class PassValidator implements Validator {  
    @Override  
    public boolean supports(Class<?> clazz) {  
        return User.class.isAssignableFrom(clazz);  
    }  
    @Override  
    public void validate(Object target, Errors errors) {  
        User u = (User) target;  
        if (!u.getPassword().trim()  
            .equals(u.getConfirmPassword().trim()))  
            errors.rejectValue("password",  
                                "user.password.error.notMatchMsg");  
    }  
}
```




Spring Security

- Sửa phương thức trong supports() lớp `com.dht.validator.WebAppValidator` như sau:

```
@Override
public boolean supports(Class<?> clazz) {
    return Product.class.isAssignableFrom(clazz) ||
           User.class.isAssignableFrom(clazz);
}
```



Spring Security

- Thêm Bean userValidator

```
@Bean
public WebAppValidator userValidator() {
    Set<Validator> springValidators = new HashSet<>();
    springValidators.add(new PassValidator());

    WebAppValidator validator = new WebAppValidator();
    validator.setSpringValidators(springValidators);

    return validator;
}
```



Spring Security

- `com.dht.repository.UserRepository`

```
public interface UserRepository {  
    void addUser(User user);  
    List<User> getUsers(String username);  
}
```

- `com.dht.repository.impl.UserRepositoryImpl`

```
@Repository  
public class UserRepositoryImpl implements UserRepository {  
    @Autowired  
    private SessionFactory sessionFactory;  
    @Override  
    public void addUser(User user) {  
        sessionFactory.getCurrentSession().save(user);  
    }  
    @Override  
    public List<User> getUsers(String username) {  
        ...  
    }  
}
```



Spring Security

- com.dht.service.UserService

```
import org.springframework.security.core
        .userdetails.UserDetailsService;

public interface UserService extends UserDetailsService {
    void addUser(User user);
    User getUserByUsername(String username);
}
```

- com.dht.service.impl.UserServiceImpl



```
@Service ("userDetailsService")
public class UserServiceImpl implements UserService {
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    @Transactional
    public void addUser(User user) {
        user.setPassword(
            bCryptPasswordEncoder.encode(user.getPassword()));
        userRepository.addUser(user);
    }
    @Override
    @Transactional(readOnly = true)
    public User getUserByUsername(String username) {
        return userRepository getUsers(username).get(0);
    }
    ...
}
```




Spring Security

```
Service("userDetailsService")
```

```
public class UserServiceImpl implements UserService {  
    ...  
    @Override  
    @Transactional(readOnly = true)  
    public UserDetails loadUserByUsername(String username)  
        throws UsernameNotFoundException {  
        List<User> users = userRepository getUsers(username);  
        if (users.isEmpty())  
            throw new UsernameNotFoundException("Không tồn tại!");  
        User u = users.get(0);  
        Set<GrantedAuthority> authorities = new HashSet<>();  
        authorities.add(new SimpleGrantedAuthority(u.getUserRole()));  
  
        return new org.springframework.security.core.userdetails.User(  
            u.getUsername(), u.getPassword(), authorities);  
    }  
}
```



@Controller

```
public class LoginController {
```

```
    @Autowired private UserService userService;
```

```
    @Autowired private WebAppValidator userValidator;
```

```
    @InitBinder
```

```
    public void initBinder(WebDataBinder binder) {
```

```
        binder.setValidator(userValidator);
```

```
    }
```

```
    @GetMapping(value = "/register")
```

```
    public String registerView(Model model) {
```

```
        model.addAttribute("user", new User());
```

```
        return "register";
```

```
    }
```

```
    @PostMapping(value = "/register")
```

```
    public String registerProcess(
```

```
        @ModelAttribute(name = "user") @Valid User user,
```

```
        BindingResult result) {
```

```
        if (result.hasErrors())
```

```
            return "register";
```

```
        userService.addUser(user);
```

```
        return "redirect:/login";
```

```
    }
```

```
}
```



Spring Security

```
@Configuration
@EnableWebSecurity
@EnableTransactionManagement
@ComponentScan(basePackages = "...")
public class SpringSecurityConfig {
    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public HandlerMappingIntrospector
        mvcHandlerMappingIntrospector() {
        return new HandlerMappingIntrospector();
    }

    ...
}
```



Spring Security

```
public class SpringSecurityConfig {
    ...
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
    Exception {
        http.csrf(c -> c.disable()).authorizeHttpRequests(requests
            -> requests.requestMatchers("/", "/home").permitAll()
            .requestMatchers(HttpMethod.GET, "/api/products").hasRole("ADMIN")
            .requestMatchers(HttpMethod.GET,
"/api/products/**").hasAnyRole("USER", "ADMIN")
                .anyRequest().authenticated())
            .formLogin(form -> form.loginPage("/login")
                .loginProcessingUrl("/login")
                .defaultSuccessUrl("/", true)
                .failureUrl("/login?error=true").permitAll())
            .logout(logout ->
logout.logoutSuccessUrl("/login").permitAll());

        return http.build();
    }
}
```



Spring Security

- SecurityWebApplicationInitializer.java

```
import org.springframework.security.web.context
    .AbstractSecurityWebApplicationInitializer;

public class SecurityWebApplicationInitializer
    extends AbstractSecurityWebApplicationInitializer {

}
```




- Bổ sung dependency:

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  <version>3.1.2.RELEASE</version>
</dependency>
```

```
@Bean
public SpringTemplateEngine templateEngine() {
    SpringTemplateEngine e = new SpringTemplateEngine();
    e.setTemplateResolver(templateResolver());
    e.addDialect(new SpringSecurityDialect());

    return e;
}
```

- Sử dụng một số thẻ:

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  <version>3.1.2.RELEASE</version>
</dependency>
```

```
<div sec:authorize="isAuthenticated()">
  <p sec:authentication="name"></p>
  <p sec:authentication="authorities"></p>
</div>
```




- Trang login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@ taglib prefix="spring"
           uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c"
           uri="http://java.sun.com/jsp/jstl/core" %>

<div class="alert alert-danger">
    <c:if test="${param.error != null}">
        <spring:message code="user.login.error1" />
    </c:if>
    <c:if test="${param.accessDenied != null}">
        <spring:message code="user.login.error2" />
    </c:if>
</div>

...
```

`<spring:url value="/login" var="action" />`

`<form action="${action}" method="post" >`

`<div class="form-group">`

`<label for="usernameId">`

`<spring:message code="user.username" />`

`</label>`

`<input name="username" id="usernameId"
 class="form-control" />`

`</div>`

`<div class="form-group">`

`<label for="passwordId">`

`<spring:message code="user.password" />`

`</label>`

`<input id="passwordId" name="password"
 class="form-control" type="password" />`

`</div>`

`<div class="form-group">`

`<input type="submit"`

`value="<spring:message code="user.login" />" />`

`</div>`

`</form>`



Spring Security

```
<ul>
  <c:choose>
    <c:when test="{pageContext.request.userPrincipal.name == null}">
      <li>
        <a href="{c:url value='/register' }">Register</a>
      </li>
      <li>
        <a href="{c:url value='/login' }">Login</a>
      </li>
    </c:when>
    <c:when test="{pageContext.request.userPrincipal.name != null}">
      <li>
        <a href="#">{pageContext.request.userPrincipal.name}</a>
      </li>
      <li>
        <a href="{c:url value='/logout' }">Logout</a>
      </li>
    </c:when>
  </c:choose>
</ul>
```



Xử lý sau khi đăng nhập thành công

- Lớp xử lý sau khi đăng nhập thành công phải hiện thực interface AuthenticationSuccessHandler

```
public class LoginSuccessHandler
    implements AuthenticationSuccessHandler {
    @Override
    public void onAuthenticationSuccess (HttpServletRequest hsr,
        HttpServletResponse response, Authentication a)
        throws IOException, ServletException {
        System.out.print("Do something!");
        response.sendRedirect("/");
    }
}
```



Xử lý sau khi đăng nhập thành công

- Khai báo trong cấu hình loginForm của Spring Security

```
http.formLogin().successHandler(this.loginSuccessHandler);
```

- loginSuccessHandler là một instance của lớp trên.



Lấy thông tin current user

- Lấy thông tin user trong @Controller: sử dụng đối số kiểu Principle hoặc đối số kiểu Authentication có trực tiếp trong các phương thức của controller.

```
@Controller
public class SecurityController {
    @RequestMapping("/user")
    public String currentUser_name(Principal principal) {
        return principal.getName();
    }
}
```



Lấy thông tin current user

- Lấy thông tin user trong một Bean

```
Authentication authentication  
    = SecurityContextHolder.getContext().getAuthentication();  
  
String currentPrincipalName = authentication.getName();
```




Spring Security Tag Libraries

- Thêm dependency

```
<dependency>  
  <groupId>org.springframework.security</groupId>  
  <artifactId>spring-security-taglibs</artifactId>  
  <version>5.5.1</version>  
</dependency>
```

- Import taglibs

```
<%@ taglib prefix="sec"  
uri="http://www.springframework.org/security/tags" %>
```



Spring Security Tag Libraries

- Kiểm tra request được chứng thực chưa

```
<sec:authorize access="hasRole('ADMIN')">  
    ADMIN USER  
</sec:authorize>
```

- Truy cập các thông tin chứng thực được lưu trong Spring Context

```
<sec:authentication property="principal.username">
```



Tạo Rest API

- REST (REpresentational State Transfer) được đề xuất vào năm 2000 bởi Roy Fielding, nó có ảnh hưởng quan trọng trong phát triển các ứng dụng Web hiện đại.
- RESTful Web Service là giải pháp thông dụng nhất để xây dựng Web Service trong các ứng dụng Web.



Tạo Rest API

- Mọi thứ trong REST được xem là một tài nguyên được xác định bởi URI, URI được sử dụng kết nối client và server để trao đổi tài nguyên theo định dạng HTML, JSON, XML, ... để có thể trao đổi dữ liệu, REST dựa trên các phương thức của giao thức HTTP như GET, POST, PUT và DELETE.
- Thông thường, các ứng dụng Web Services dựa trên REST sẽ trả về dữ liệu theo hai định dạng chính là JSON hoặc XML.



Tạo Rest API

```
@RestController
public class ApiProductController {
    @Autowired
    private ProductService productService;

    @GetMapping("/api/product")
    public ResponseEntity<List<Product>> getProducts() {
        return new ResponseEntity<>(
            this.productService.getProducts(),
            HttpStatus.OK);
    }
}
```

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.2</version>
</dependency>
```




Tạo Rest API

- @JsonGetter đánh dấu một phương thức là getter trong quá trình serialization.
- @JsonValue chỉ định phương thức duy nhất để jackson sử dụng serialize đối tượng.
- @JsonRootName (class level) chỉ định một tên key (của Map) chứa dữ liệu response.
- @JsonIgnoreProperties (class level) chỉ định các thuộc tính sẽ bỏ qua khi serialize và deserialize.



Tạo Rest API

- @JsonIgnore
- @JsonInclude(Include.NON_NULL)
- @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy hh:mm:ss")
- @JsonProperty(value=): chỉ định tên khoá cho thuộc tính khi serialize



Tạo Rest API

GET http://localhost:8088/Th72Dem... + ... No Environment

Untitled Request BUILD

GET http://localhost:8088/Th72DemoWeb/api/product Send Save

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
-----	-------	-------------	-----	-----------

Body Cookies Headers (11) Test Results 200 OK 75 ms 4.75 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "name": "iPhone 7 Plus",
5     "description": "32GB",
6     "price": 11000000,
7     "active": true,
8     "image": "/images/uploads/iphone-7-plus.png",
9     "category": {
10      "id": 1,
11      "name": "Điện thoại thông minh"
12    }
13  },
14  {
15    "id": 2,
```



Tạo Rest API

```
@RestController
public class ApiProductController {
    @Autowired
    private ProductService productService;

    @PostMapping("/{productId}")
    @ResponseStatus(value = HttpStatus.CREATED)
    public void addOrUpdate(HttpSession session,
        @PathVariable(value = "productId") int productId) {
        Product product
            = this.productService.getProductById(productId);
        this.productService.addOrUpdateProduct(product);
    }
}
```



SpringMVC + Hibernate

- **Domain layer** chứa các domain model đại diện cho các loại lưu trữ dữ liệu dựa trên các yêu cầu logic nghiệp vụ.
- Ví dụ tạo gói com.dht.model, trong gói này lần lượt tạo các lớp Category.java.

```
@Entity
@Table(name="category")
public class Category implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name;
}
```




SpringMVC + Hibernate

- Việc xử lý truy vấn dữ liệu được tách thành một tầng riêng giúp việc tái sử dụng logic xử lý tương tác dữ liệu hiệu quả hơn ở các controller và các tầng khác. Các công việc này được thực hiện ở tầng **Persistence Layer**.
- Đối tượng repository có nhiệm vụ thực hiện các thao tác CRUD trên các đối tượng domain. @Repository là chỉ định lớp Repository.



SpringMVC + Hibernate

- **Persistence layer** chứa các đối tượng repository để truy cập vào các đối tượng domain, đối tượng repository gửi các câu truy vấn tới data source của dữ liệu, ánh xạ (map) dữ liệu từ data source đến đối tượng domain, và cuối cùng nó lưu trữ bền vững (persist) sự thay đổi của đối tượng domain xuống data source.



SpringMVC + Hibernate

- Tạo com.dht.repository

```
public interface CategoryRepository {  
    List<Category> getCategories();  
}
```



SpringMVC + Hibernate

- com.dht.repository.impl

```
@Repository
public class CategoryRepositoryImpl
    implements CategoryRepository {

    @Autowired
    private LocalSessionFactoryBean sessionFactory;

    @Override
    @Transactional
    public List<Category> getCategories() {
        Session session = this.sessionFactory
            .getObject()
            .getCurrentSession();
        Query q = session.createQuery("FROM Category");

        return q.getResultList();
    }
}
```



SpringMVC + Hibernate

- **Service Layer** chứa các xử lý nghiệp vụ phức tạp tương tác với cơ sở dữ liệu, bao gồm nhiều thao tác CRUD, thực hiện trên nhiều đối tượng repository.
- Tạo com.dht.service

```
public interface CategoryService {  
    List<Category> getCategories();  
}
```




SpringMVC + Hibernate

- Tạo com.dht.service.impl

```
@Service
public class CategoryServiceImpl implements CategoryService {
    @Autowired
    private CategoryRepository categoryRepository;

    @Override
    public List<Category> getCategories() {
        return this.categoryRepository.getCategories();
    }
}
```



SpringMVC + Hibernate

- Sử dụng trong controller

```
@Controller
public class HomeController {
    @Autowired
    private CategoryService categoryService;

    @RequestMapping("/test")
    public String test(Model model) {
        model.addAttribute("categories",
                           this.categoryService
                               .getCategories());

        return "index";
    }
}
```

- CORS (Cross-Origin Resource Sharing) cho phép giao tiếp giữa các domain khác nhau.

```
@Controller
@CrossOrigin(origins="http://example.com")
public class ProductController {
    @CrossOrigin
    @GetMapping("/products")
    public String list() {
        // ...
    }
}
```

- Khai báo CORS toàn cục

```
@Configuration
@EnableWebMvc
public class WebApplicationContextConfig
    implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry){
        registry.addMapping("/**");
    }
}
```

- Sử dụng CORS với Spring Security

```
@EnableWebSecurity
public class WebSecurityConfig {

    @Bean
    public SecurityFilterChain
filterChain(HttpSecurity http) throws Exception {
        http.cors().and()...
    }
}
```


Q&A