

LẬP TRÌNH JAVA

SPRING FRAMEWORK

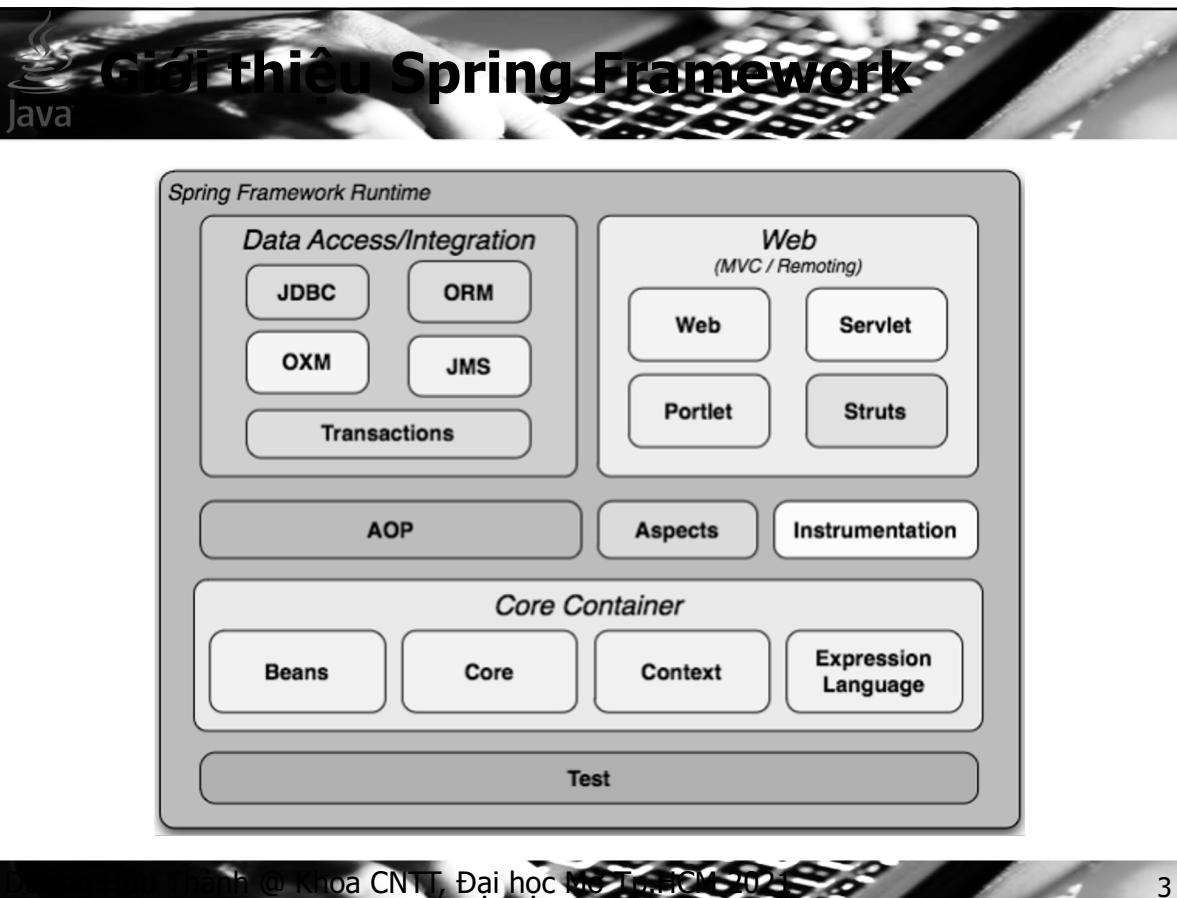
ThS. Dương Hữu Thành
Khoa CNTT, Đại học Mở Tp.HCM
thanh.dh@ou.edu.vn



1

 **Nội dung chính**

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



Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

3

Core Container

- Core module là thành phần quan trọng nhất của Spring Framework cung cấp các đặc trưng như IoC, DI.
- Bean module cung cấp BeanFactory là mẫu thiết kế Factory tổng quát phân tách sự phụ thuộc như khởi động, tạo đối tượng và việc truy cập vào các đối tượng từ logic chương trình.
 - Singleton tạo một thể hiện duy nhất của đối tượng.
 - Prototype (non-singleton) mỗi kết quả truy vấn sẽ tạo đối tượng mới.

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

4



Core Container

- Context module: ApplicationContext container sẽ nạp các định nghĩa Spring Bean và mốc nối (wire) chúng với nhau.
- SpEL (Spring Expression Language) là một ngôn ngữ biểu diễn (expression language) mạnh mẽ hỗ trợ các đặc trưng truy vấn và thao tác với một đối tượng lúc thực thi.



Application Context

- ClassPathXmlApplicationContext: từ tập tin XML trong classpath.
- FileSystemXmlApplicationContext: tập tin XML với đường dẫn tuyệt đối.
- XmlWebApplicationContext: nạp tập tin cấu hình XML với các định nghĩa của các bean từ vị trí chuẩn trong thư mục webapp, mặc định ở /WEB-INF/applicationContext.xml.
- AnnotationConfigApplicationContext: nạp các lớp Java được gắn annotation @Configuration thay vì sử dụng các tập tin XML.
- AnnotationConfigWebApplicationContext: được sử dụng tạo web application context bằng cách nạp các lớp Java gắn annotation @Configuration.



Viết chương trình đầu tiên

- Tạo một project maven, trong tập tin pom.xml bổ sung các dependencies sau:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.4</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.4</version>
</dependency>
```

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TP.HCM 2021

7

7



Viết chương trình đầu tiên

- Tạo tập tin HelloWorld.java trong com.dht.test

```
public class HelloWorld {
    private String message;

    // Các phương thức getter/setter của message
}
```

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TP.HCM 2021

8

8



Viết chương trình đầu tiên

- Tạo tập tin Beans.xml trong src/main/resources:

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/
       beans http://www.springframework.org/schema/beans/spring-
       beans-3.0.xsd">

    <bean id="helloWorld" class="com.dht.test.HelloWorld">
        <property name="message" value="Hello World!" />
    </bean>

</beans>
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

9

9



Viết chương trình đầu tiên

- Tạo lớp Tester trong com.dht.test và tạo phương thức main trong lớp.

```
ApplicationContext app
    = new ClassPathXmlApplicationContext("Beans.xml");

HelloWorld h = (HelloWorld) app.getBean("helloWorld");
System.out.println(h.getMessage());
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

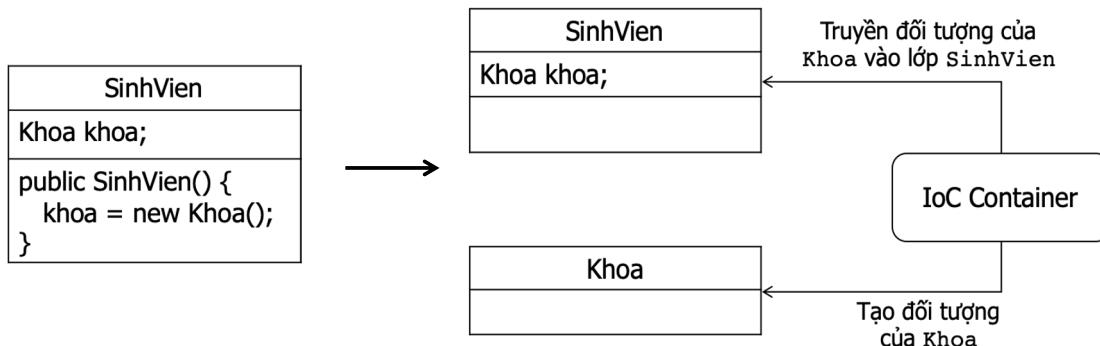
10

10



IoC Container

- Nguyên tắc cơ bản của IoC dựa trên nguyên tắc Hollywood: “Đừng gọi chúng tôi, chúng tôi sẽ gọi bạn” (Do not call us, we'll call you).



Beans

- Sử dụng 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"
       xsi:schemaLocation =
"http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">

    <bean id = "helloWorld" class = "springdemo.HelloWorld">
        <property name = "message" value = "Hello World!" />
    </bean>

</beans>
```



Một số thuộc tính thẻ <bean>

- class: chỉ định lớp tạo bean.
- name: tên thuộc tính bean, hoặc sử dụng thuộc tính id với ý nghĩa tương tự.
- scope: phạm vi hoạt động đối tượng bean.
 - singleton (mặc định): tạo thể hiện bean duy nhất cho các lần sử dụng nó.
 - prototype: tạo thể hiện bean mới mỗi lần sử dụng.
 - request: tạo bean mới mỗi HTTP request.
 - session: tạo thể hiện bean mới mỗi HTTP session.



Một số thuộc tính thẻ <bean>

- init-method: thuộc tính này chỉ định phương thức được gọi khi khởi tạo bean.
- destroy-method: thuộc tính chỉ định phương thức được gọi trước khi bean bị huỷ.

```
<bean id = "helloWorld" class = "springdemo.HelloWorld"
      init-method="init" destroy-method="destroy" >
    <property name = "message" value = "Hello World!" />
</bean>
```



Các thẻ con thẻ <bean>

- Thẻ <property> sử dụng setter để inject
 - name: tên thuộc tính Java bean.
 - value: giá trị thiết lập cho thuộc tính Java bean.
 - ref: tham chiếu tới đối tượng bean khác.



Các thẻ con thẻ <bean>

- Thẻ <constructor-arg> sử dụng phương thức khởi tạo để inject.
 - index: chỉ số trong danh sách đối số của phương thức khởi tạo.
 - type: kiểu của đối số trong constructor.
 - value: giá trị là chuỗi chỉ định giá trị cho bean.
 - ref: tham chiếu tới bean khác.

```
<bean id = "helloWorld" class = "springdemo.HelloWorld">
    <constructor-arg value="Welcome " />
    <constructor-arg value="Thanh!" />
</bean>
```



Autowiring

- Autowiring cho phép ta không cần cung cấp chi tiết bean injection tường minh.
- Spring container có thể autowire quan hệ giữa các bean mà không cần sử dụng các thẻ <property> hay <constructor-arg>.
- Đặc trưng này được thể hiện thông qua thuộc tính autowire của thẻ <beans>, mặc định thuộc tính này vô hiệu (disabled).

```
<bean id="helloWorld" class="springdemo.HelloWorld"
      autowire = "autowire-type">
```



Autowiring

- Autowire-type
 - no: chế độ autowiring bị vô hiệu (mặc định).
 - byname: autowiring dựa trên name. Phương thức setter được sử dụng cho loại autowire này để inject một dependency.
 - byType: autowiring dựa trên kiểu dữ liệu.
 - constructor: tương tự byType nhưng sử dụng constructor để inject một dependency.
 - autodetect: Spring sẽ autowire bằng constructor, nếu không được Spring sẽ thử autowire bằng byType.



Cấu hình Java Collection

- Spring cung cấp 4 loại cấu hình Collections:
 - <list>: kết nối với thuộc tính kiểu danh sách (list).
 - <set>: kết nối với thuộc tính kiểu tập hợp (set), tức là không cho phép hai phần tử trùng nhau trong danh sách.
 - <map>: kết nối thuộc tính kiểu Collection mà mỗi phần tử là một cặp key/value, trong đó key và value có thể có kiểu dữ liệu bất kỳ.
 - <props>: tương tự map nhưng key và value có kiểu String.



Cấu hình Java Collection

- Ví dụ tạo tập tin CollectionDemo.java có nội dung như sau:

```
public class CollectionDemo {  
    private List demoList;  
    private Set demoSet;  
    private Map demoMap;  
    private Properties demoProperties;  
  
    // Các phương thức getter/setter của các thuộc tính  
}
```



```
<bean id="collectionDemo" class="springdemo.CollectionDemo">
    <property name="demoList">
        <list>
            <value>Apple</value>
            <value>Banana</value>
        </list>
    </property>
    <property name="demoSet">
        <set>
            <value>Apple</value>
            <value>Banana</value>
        </set>
    </property>
    <property name="demoMap">
        <map>
            <entry key="orange" value="Orange" />
            <entry key="lemon" value="Lemon" />
        </map>
    </property>
    <property name="demoProperties">
        <props>
            <prop key="blackberry">Blackberry</prop>
            <prop key="mango">Mango</prop>
        </props>
    </property>
</bean>
```

21

21



Cấu hình Java Collection

- Trong phương thức main() sử dụng đối tượng bean

```
ApplicationContext context
        = new
ClassPathXmlApplicationContext("Beans.xml");
CollectionDemo colDemo
        = (CollectionDemo)
context.getBean("collectionDemo");
System.out.println("List: " + colDemo.getDemoList());
System.out.println("Set: " + colDemo.getDemoSet());
System.out.println("Map: " + colDemo.getDemoMap());
System.out.println("Properties: " + colDemo.getDemoProperties());
```



Sử dụng annotation cho Beans

```
<?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>
```



Một số annotation quan trọng

- **@Required:** áp dụng cho các phương thức setter của các thuộc tính.

```
import org.springframework.beans.factory.annotation.Required;
public class Category {
    private String name;
    public String getName() {
        return name;
    }
    @Required
    public void setName(String name) {
        this.name = name;
    }
}
```



Một số annotation quan trọng

- **@Autowired:** áp dụng cho thuộc tính, phương thức khởi tạo, phương thức setter.

```
import org.springframework.beans.factory.annotation.Autowired;
public class Product {
    private String name;
    @Autowired
    private Category category;
}
```

```
<bean id="category" class="springdemo.Category">
    <property name="name" value="Mobile" />
</bean>
<bean id="product" class="springdemo.Product">
    <property name="name" value="iPhone 7 Plus" />
</bean>
```



Một số annotation quan trọng

- **@Autowired**

```
ApplicationContext context
    = new ClassPathXmlApplicationContext("Beans.xml");

Product p = (Product) context.getBean("product");
System.out.printf("%s - %s\n",
    p.getName(),
    p.getCategory().getName());
```



Một số annotation quan trọng

- @Qualifier: khi có nhiều đối tượng beans cùng kiểu, thì sử dụng annotation kết hợp với @Autowired để chỉ định mốc nối (wired) cụ thể đến đối tượng beans nào.

```
<bean id="category" class="springdemo.Category">
    <property name="name" value="Mobile" />
</bean>
<bean id="cate" class="springdemo.Category">
    <property name="name" value="Tablet" />
</bean>
```



Một số annotation quan trọng

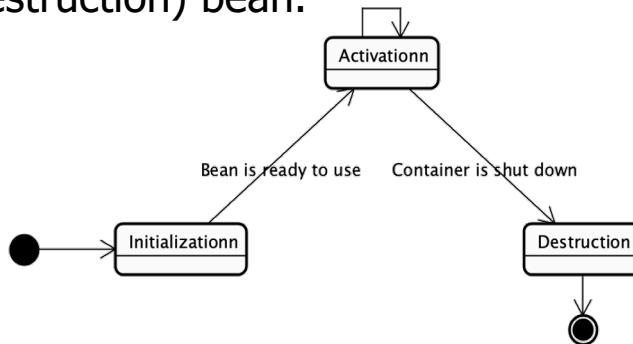
- @Qualifier

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
public class Product {
    private String name;
    @Autowired
    @Qualifier("cate")
    private Category category;
}
```



Vòng đời Spring Bean

- BeanFactory quản lý vòng đời của các beans được tạo thông qua Spring IoC Container.
- Vòng đời hoạt động của bean bao gồm nhiều hàm callback thực hiện trong hai thời điểm: sau khi khởi động (post-initialization) và trước khi huỷ (pre-destruction) bean.



Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

29

29



Vòng đời Spring Bean

- **Khởi tạo bean (Initialization)**
 - Bean container tìm thấy định nghĩa bean trong tập tin cấu hình và tạo thể hiện của bean.
- **Sử dụng bean (Activation)**
 - Bean đã được khởi tạo và dependency đã được inject, bean sẵn sàng được sử dụng trong ứng dụng
- **Huỷ bean (Deletion)**
 - Nếu lớp Bean hiện thực giao diện DisposableBean thì sẽ gọi phương thức destroy().
 - Nếu bean có khai báo thuộc tính destroy-method thì phương thức sẽ được gọi.

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

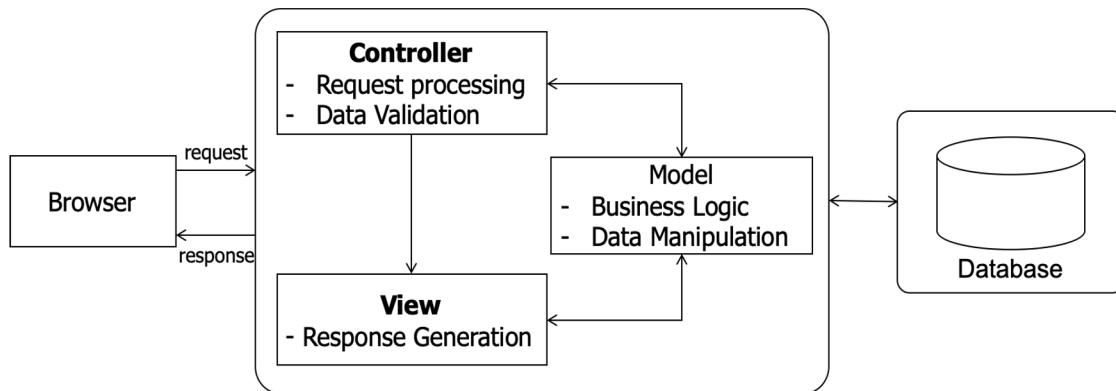
30

30



Spring MVC

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



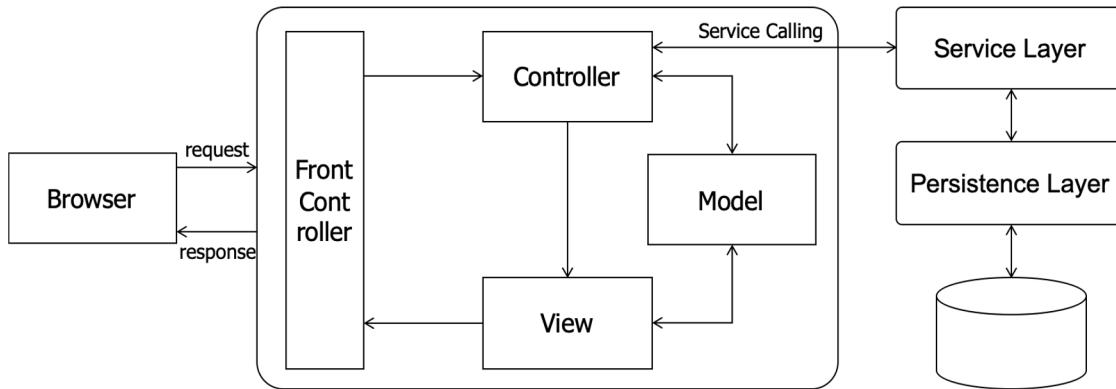
Spring MVC

- 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).



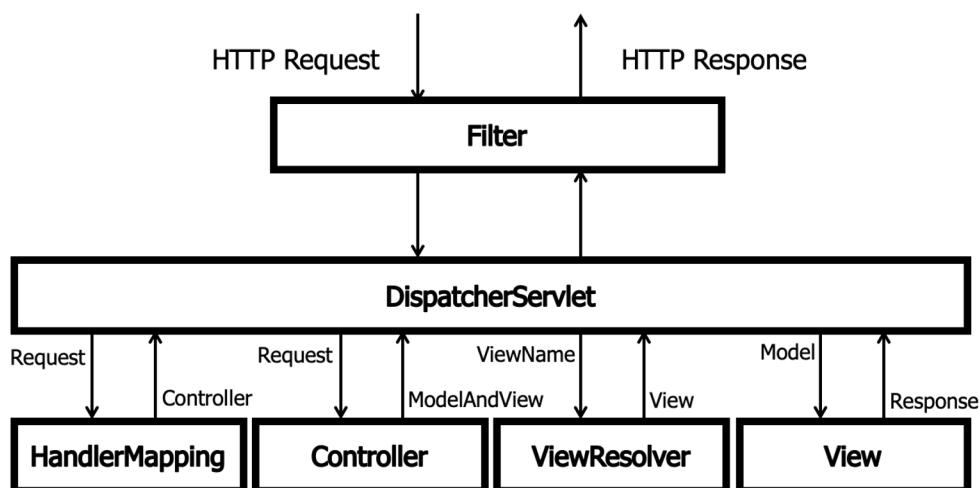
Front Controller Design Pattern

- 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



DispatcherServlet

- 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.



DispatcherServlet

- 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

- View Resolver 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 Spring MVC

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

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

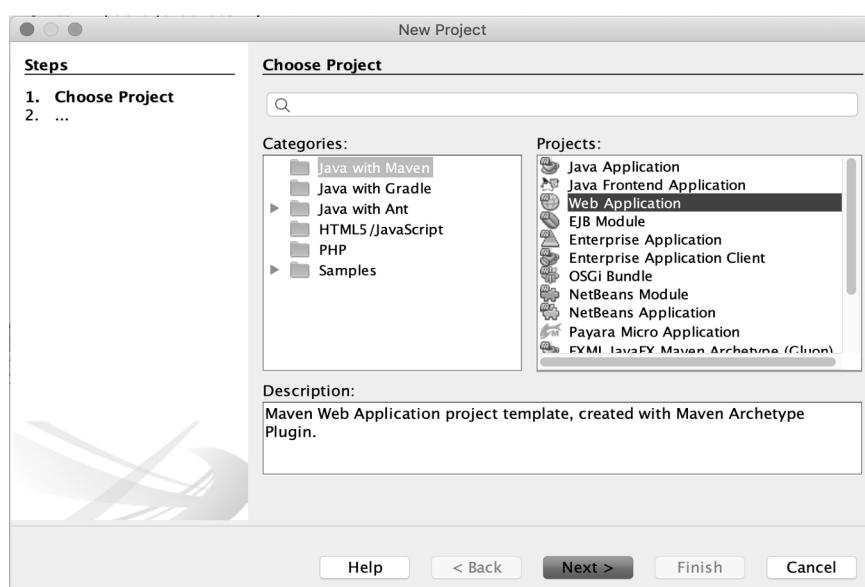
45

45



Chương trình đầu tiên Spring MVC

- Tạo project web maven



Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

46

46



Chương trình đầu tiên Spring MVC

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

The screenshot shows the Apache Tomcat download page at tomcat.apache.org/download-90.cgi. The page header includes the Java logo and the title "Chương trình đầu tiên Spring MVC". On the left, there's a sidebar with links for Tomcat versions (9.0, 8.5, 7.0), connectors, native, wiki, migration guide, and presentations. Below that is a "Problems?" section with security reports, help, FAQ, mailing lists, bug database, and IRC. At the bottom is a "Get Involved" section with overview, source code, buildbot, and translations. The main content area is titled "9.0.30" and contains a note: "Please see the [README](#) file for packaging information. It explains what every". Below this is a "Binary Distributions" section with a bulleted list of download options:

- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
 - [tar.gz \(pgp, sha512\)](#)
- Deployer:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
- Embedded:
 - [tar.gz \(pgp, sha512\)](#)
 - [zip \(pgp, sha512\)](#)

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

47

47



Chương trình đầu tiên Spring MVC

- Thiết lập Tomcat Server

The screenshot shows the "New Web Application" wizard in the GlassFish Admin Console. The title bar says "New Web Application". The left sidebar shows "Steps" with "1. Choose Server" highlighted. The main panel is titled "Add Server Instance" and has a "Choose Server" dropdown. The dropdown list shows "Amazon Beanstalk", "Apache Tomcat or TomEE" (which is selected and highlighted in blue), "GlassFish Server", and "Payara Server". Below the dropdown is a "Name:" field containing "Apache Tomcat or TomEE". At the bottom are "Help", "< Back", "Next >", "Finish", and "Cancel" buttons.

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

48

48



Chương trình đầu tiên Spring MVC

▪ 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

Use Private Configuration Folder (Catalina Base)

Catalina Base:

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

49

49



Chương trình đầu tiên Spring MVC

▪ Đienia thông tin tên project

New Web Application

Steps Name and Location Add Server Instance

Steps

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

Name and Location

Project Name: SpringMVCdemo

Project Location: /Users/duonghuuthanh/NetBeansProjects

Project Folder: onghuuthanh/NetBeansProjects/SpringMVCdemo

Artifact Id: SpringMVCdemo

Group Id: com.dht

Version: 1.0-SNAPSHOT

Package: com.dht.springmvcdemo (Optional)

Help < Back Next > Finish Cancel

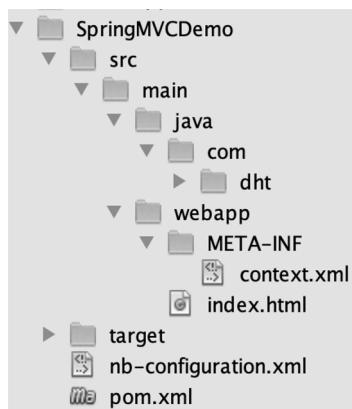
50

50



Chương trình đầu tiên Spring MVC

- Cấu trúc project được tạo



Chương trình đầu tiên Spring MVC

- Thêm các dependencies vào pom.xml

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javamee-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 Spring MVC

```
@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;  
    }  
}
```

53

Chương trình đầu tiên Spring MVC

```
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 Spring MVC

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

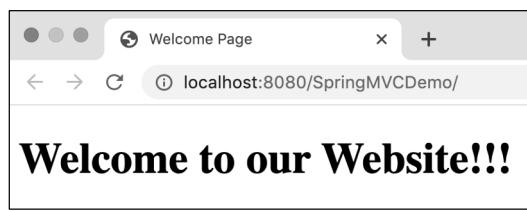
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Welcome Page</title>
    </head>
    <body>
        <h1>${message}</h1>
    </body>
</html>
```



Chương trình đầu tiên Spring MVC

- Controller

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





Controller

Java

- 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 với 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

Java

- 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 đó.



- @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";
}
```



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) {
        dict.put(newWord.getWord(), newWord.getDescription());
        return "redirect:/dicts/list";
    } else {
        model.addAttribute("message", "Từ đã tồn tại!!!!");
        return "dicts-add-word";
    }
}
```



▪ 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.



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 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, cung 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.



View Resolver

▪ 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

Java

▪ 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/");  
}
```

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

79

79



View Resolver

Java

▪ 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=""  
            rel="stylesheet" />  
    </head>  
    <body>  
          
    <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>
```



View Resolver

Java

- 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>
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

83

83



View Resolver

Java

```
@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());
        }
    }
    ...
}
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

84

84



Bean Validation

- Java 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 mới 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

- Chính 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

- Spring Validation



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() {}
}
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

95

95



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;
}
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

96

96



Hibernate Config

- @Bean dataSource(): việc tạo kết nối đến cơ sở dữ liệu tồn tại 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 getSessionFactory sử dụng LocalSessionFactoryBean tạo SessionFactory.



Hibernate Config

- Thêm phương thức trong **HibernateConfig**

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



Hibernate Config

- Tập tin WebApplicationContextConfig.java

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

- Tập tin DispatcherServletInitializer.java:

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



Hibernate Config

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



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/template/base.jsp">
        <put-attribute name="title" value="" />
        <put-attribute name="header"
            value="/WEB-INF/layout/template/header.jsp" />
        <put-attribute name="content" value="" />
        <put-attribute name="footer"
            value="/WEB-INF/layout/template/footer.jsp" />
    </definition>
    <definition name="index" extends="baseLayout">
        <put-attribute name="title" value="Trang chủ" />
        <put-attribute name="content"
            value="/WEB-INF/jsp/index.jsp" />
    </definition>
</tiles-definitions>
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

103

103



Template with Tiles

▪ Tập tin 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">
        <title><tiles:insertAttribute name="title" /></title>
    </head>
    <body>
        <tiles:insertAttribute name="header" />
        <tiles:insertAttribute name="content" />
        <tiles:insertAttribute name="footer" />
    </body>
</html>
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

104

104



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/layout/tiles.xml");  
        configurer.setCheckRefresh(true);  
        return configurer;  
    }  
}
```

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

105

105



Template with Tiles

▪ Chỉ định thông tin beans

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

Đỗ Văn Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

106

106



Spring Security

- Spring Security quan tâm đến các đối tượng HttpServletRequest và HttpServletResponse, 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



Spring Security

- Các dependencies

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



Spring Security

Java

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

109

109



Tạo Rest API

Java

Đỗ Minh Thành @ Khoa CNTT, Đại học Mở TPHCM 2021

110

110



SpringMVC + Hibermate

- 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.saleweb.model trong thư mục src/main/java, trong gói này lần lượt tạo các lớp Category.java, Product.java.



SpringMVC + Hibermate

- 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 + Hibermate

- 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 + Hibermate

- 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.



- 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.

Q&A