



LẬP TRÌNH WEB (WEBPR330479)

# Spring Framework, Spring MVC, Spring Boot

THS. NGUYỄN HỮU TRUNG

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- [trungnh@hcmute.edu.vn](mailto:trungnh@hcmute.edu.vn)
- <https://www.youtube.com/@baigiai>

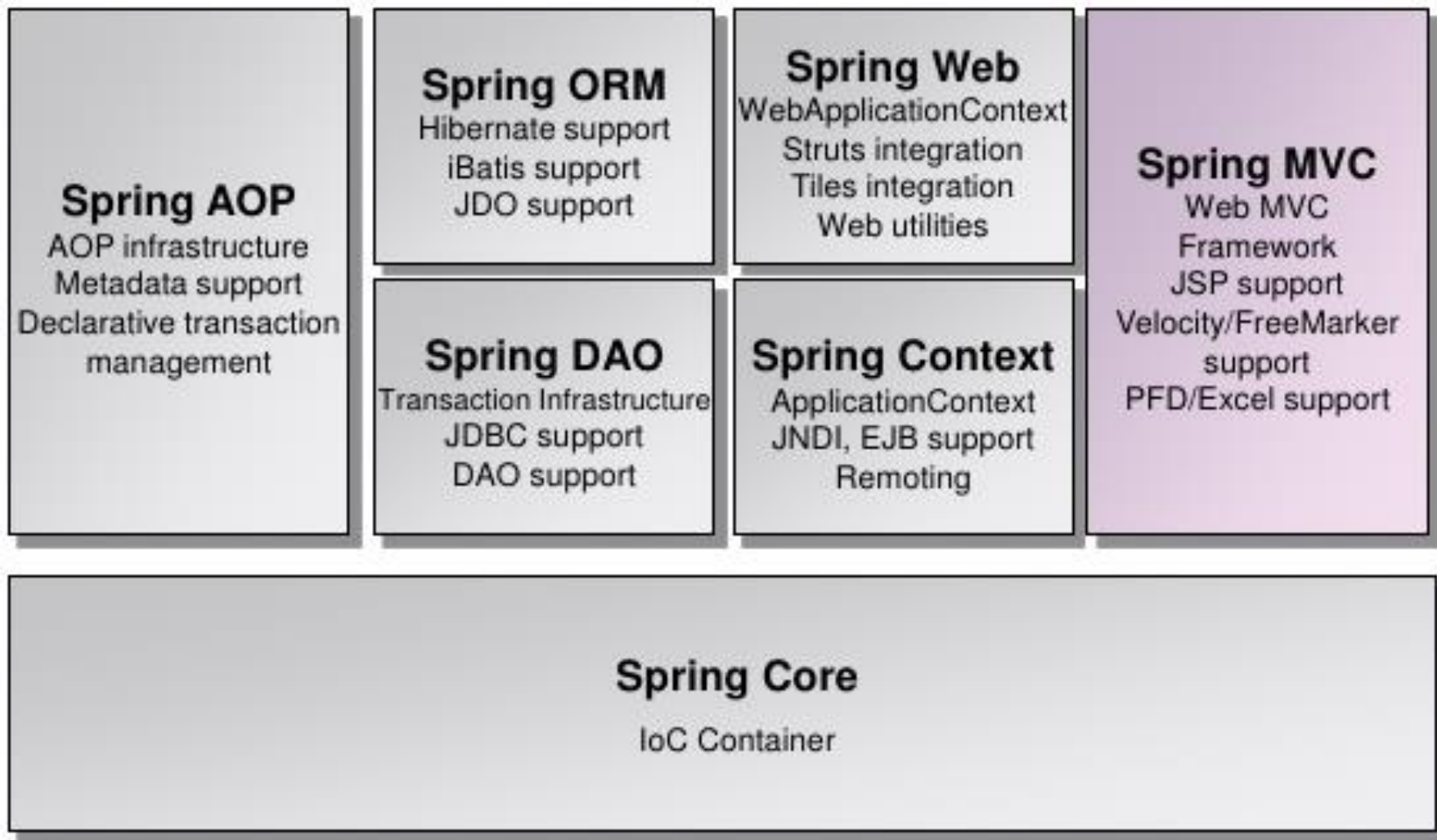


- Spring framework là nền tảng mã nguồn mở. Nó cung cấp cơ sở hạ tầng toàn diện để phát triển ứng dụng Java một cách mạnh mẽ, rất dễ dàng và nhanh chóng.
- Spring framework được tạo bởi Rod Johnson và bắt đầu được giới thiệu vào tháng 6 năm 2003.
- Spring là framework phát triển ứng dụng Java phổ biến nhất đối với doanh nghiệp.
- Spring Framework được hàng triệu nhà phát triển ứng dụng trên toàn thế giới sử dụng để tạo ra các sản phẩm phần mềm với hiệu suất cao, dễ dàng kiểm chứng, tái sử dụng mã.



# Kiến trúc Spring Framework

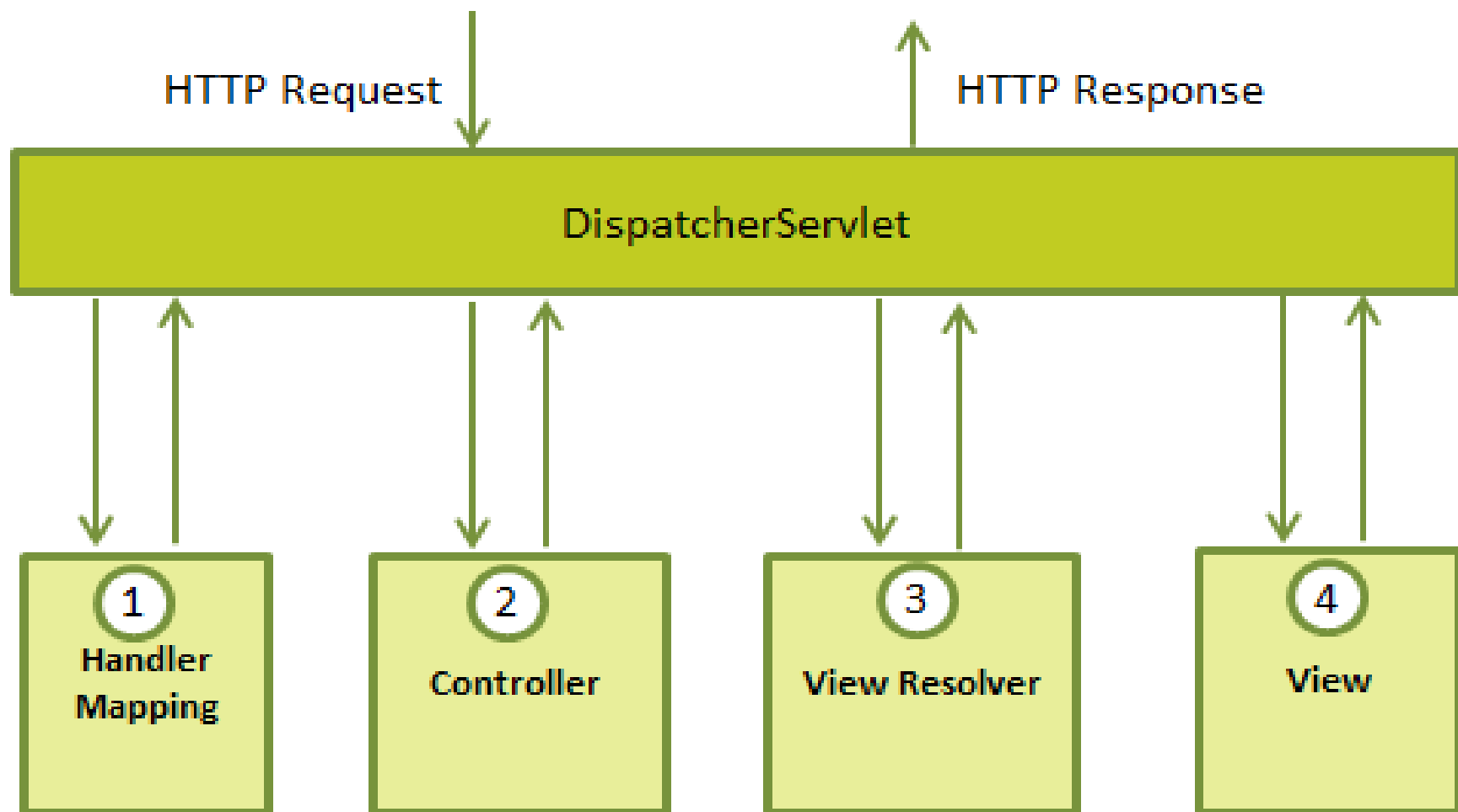
4



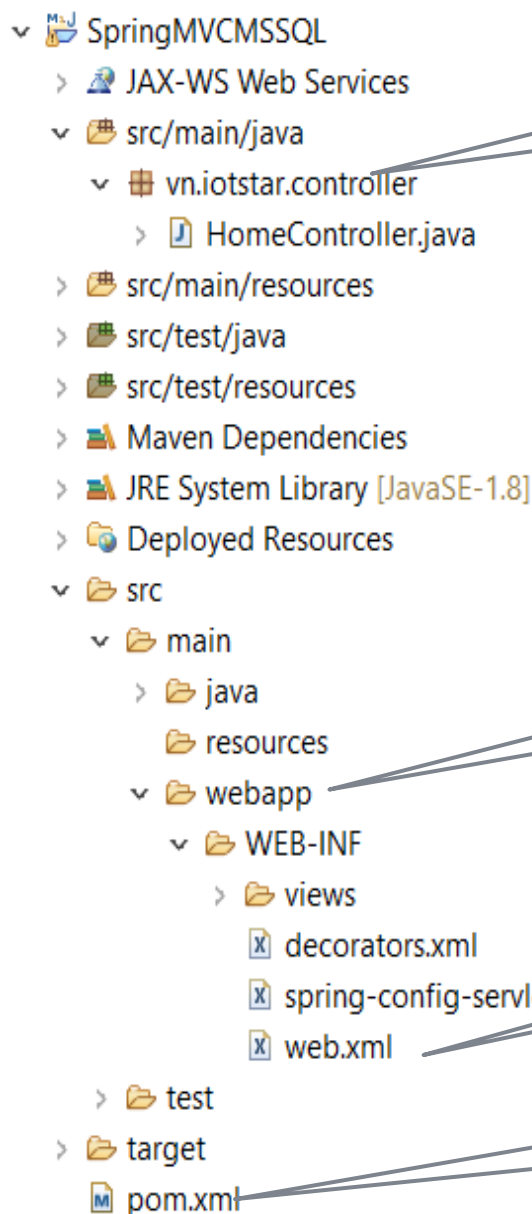
- Spring Core
  - ▣ Cung cấp nền tảng cơ bản của hệ thống ứng dụng Spring
- Spring AOP
  - ▣ Cung cấp nền tảng cho lập trình hướng khía cạnh
- Spring DAO
  - ▣ Cung cấp dụng cụ đối tượng truy xuất dữ liệu
- Spring Context
  - ▣ Cung cấp dịch vụ truy cập từ xa như JNDI, EJB...
- Spring MVC
  - ▣ Nền tảng ứng dụng web theo mô hình MVC
- Spring ORM
  - ▣ Cung cấp dịch vụ ánh xạ đối tượng quan hệ dữ liệu
- Spring Web
  - ▣ Cung cấp dịch vụ tích hợp các framework web khác

# Xử lý request trong Spring MVC

6



# Tổ chức dự án web với Maven Project



Các file mã nguồn Java đặt ở thư mục src

- Để dự án hoạt động theo Spring MVC cần
  - ▣ Các thư viện liên quan (\*.jar)
  - ▣ Cấu hình đúng (\*.xml)
  - ▣ Viết mã theo đúng qui ước

Các file jsp, ảnh, scripts, styles... đặt ở webapp

File web.xml là file cấu hình ứng dụng web

Chứa cấu hình các file thư viện

# Tổ chức dự án Spring MVC với Maven Project

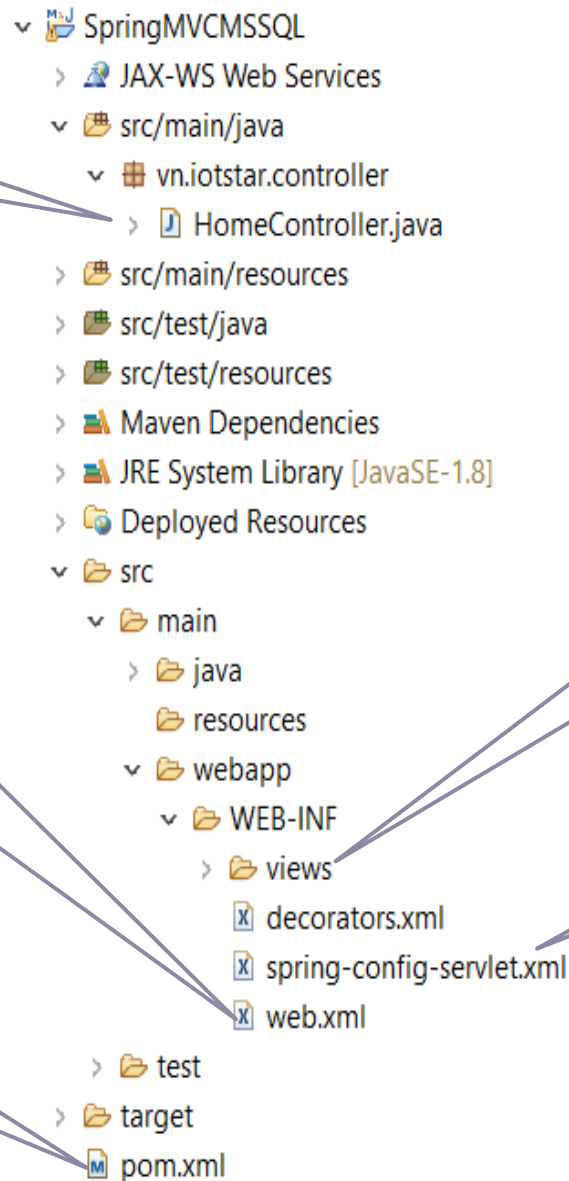
Controller

Cấu hình web

Thư viện

View

Cấu hình Spring MVC

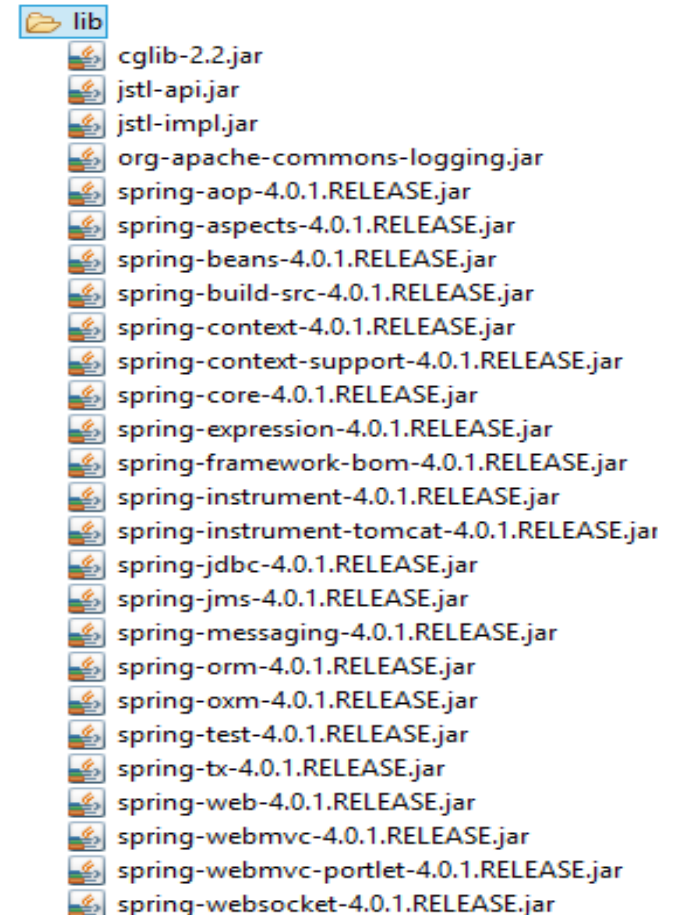




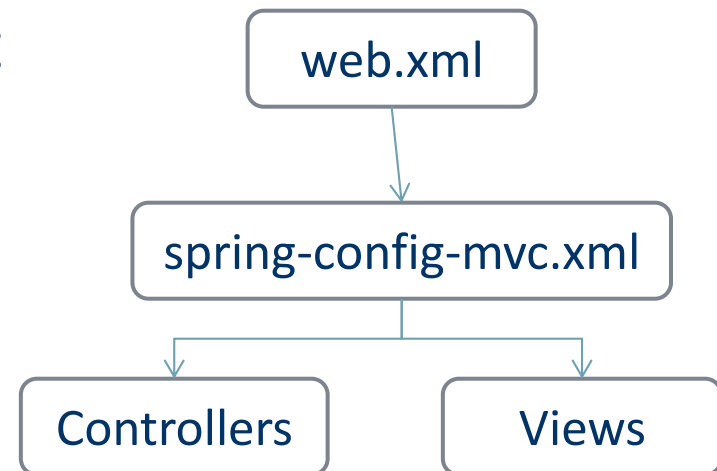
- Thư viện cần thiết cho ứng dụng web nói chung và Spring MVC nói riêng phải được đặt trong thư mục **/WEB-INF/lib** đối với **Dynamic Web**, còn đối với **Maven Web** thì cấu hình trong file **pom.xml**.

## ▼ Maven Dependencies

- > junit-3.8.1.jar - E:\M2\junit\junit\3.8.1
- > javax.servlet-api-4.0.1.jar - E:\M2\javax\servlet\javax.servlet-api\4.0.1
- > spring-webmvc-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-webmvc\4.3.30.RELEASE.jar
- > spring-aop-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-aop\4.3.30.RELEASE.jar
- > spring-beans-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-beans\4.3.30.RELEASE.jar
- > spring-context-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-context\4.3.30.RELEASE.jar
- > spring-core-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-core\4.3.30.RELEASE.jar
- > commons-logging-1.2.jar - E:\M2\commons-logging\commons-logging\1.2
- > spring-expression-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-expression\4.3.30.RELEASE.jar
- > spring-web-4.3.30.RELEASE.jar - E:\M2\org\springframework\spring-web\4.3.30.RELEASE.jar
- > jstl-1.2.jar - E:\M2\jstl\jstl\1.2
- > jsp-api-2.0.jar - E:\M2\javax\servlet\jsp\jsp-api\2.0
- > servlet-api-2.4.jar - E:\M2\javax\servlet\servlet-api\2.4
- > mysql-connector-java-8.0.13.jar - E:\M2\mysql\mysql-connector-java\8.0.13
- > protobuf-java-3.6.1.jar - E:\M2\com\google\protobuf\protobuf-java\3.6.1
- > mssql-jdbc-9.4.0.jre8.jar - E:\M2\com\microsoft\sqlserver\mssql-jdbc\9.4.0.jre8
- > sitemesh-2.4.2.jar - E:\M2\opensymphony\sitemesh\sitemesh\2.4.2



- **web.xml** là file cấu hình ứng dụng web
  - ▣ Khai báo **DispatcherServlet**
    - Tiếp nhận và điều phối yêu cầu từ người dùng
  - ▣ Khai báo **CharacterEncodingFilter**
    - Xử lý chế độ mã hóa ký tự
  - ▣ Khai báo **spring-config-mvc.xml**
    - Cấu hình Spring MVC
- **spring-config-mvc.xml** là file cấu hình Spring MVC
  - ▣ Cấu hình ứng dụng Spring MVC
  - ▣ Khai báo **Controller**
  - ▣ Khai báo **ViewResolver**



# Cấu hình ứng dụng web trong web.xml

11

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://JAVA.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
<display-name>Shopping Spring MVC</display-name>
<servlet>
<servlet-name>dispatcher</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/configs/*_xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>dispatcher</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
```

Khai báo  
**DispatcherServlet**

Sử dụng dấu \* để chỉ ra rằng tất cả các file xml đặt vào thư mục **/WEB-INF/configs** đều được xem như là file cấu hình Spring và được nạp vào ứng dụng

Tất cả các URL đều được DispatcherServlet tiếp nhận và xử lý

# Khai báo CharacterEncodingFilter

12

```
<filter>
  <filter-name>utf8</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>utf8</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

CharacterEncodingFilter cho phép ứng dụng web làm việc với utf-8 (tiếng Việt)

# Cấu trúc file cấu hình Spring-config-mvc.xml

13

```
<?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"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd">
```

```
<!-- Nội dung khai báo cấu hình Spring -->
```

```
</beans>
```

Các namespace và schema qui định cú pháp trong file cấu hình

# Spring-config-mvc.xml

14

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

```
<beans ...>
```

```
<!-- Cấu hình Spring MVC Annotation -->
```

```
<context:annotation-config />
```

```
<mvc:annotation-driven/>
```

Cho phép sử dụng Annotation trong ứng dụng Spring

```
<!-- Cấu hình ViewResolver -->
```

```
<bean id="viewResolver"
```

```
  p:prefix="/WEB-INF/views/" p:suffix=".jsp"
```

```
  class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
```

View = prefix + viewname + suffix

```
<!-- Cấu hình package chứa các controller -->
```

```
<context:component-scan base-package="poly.controller"/>
```

```
</beans>
```

```
<context:component-scan base-package="vn.iotstar"/>
```

Chỉ rõ gói chứa các Controller. Sử dụng dấu phẩy để phân cách các gói

```
package vn.iotstar.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

    @RequestMapping(value={"/", "/trang-chu"})
    public String Index() {
        return "Hello";
    }
}
```

Chú thích lớp  
Controller

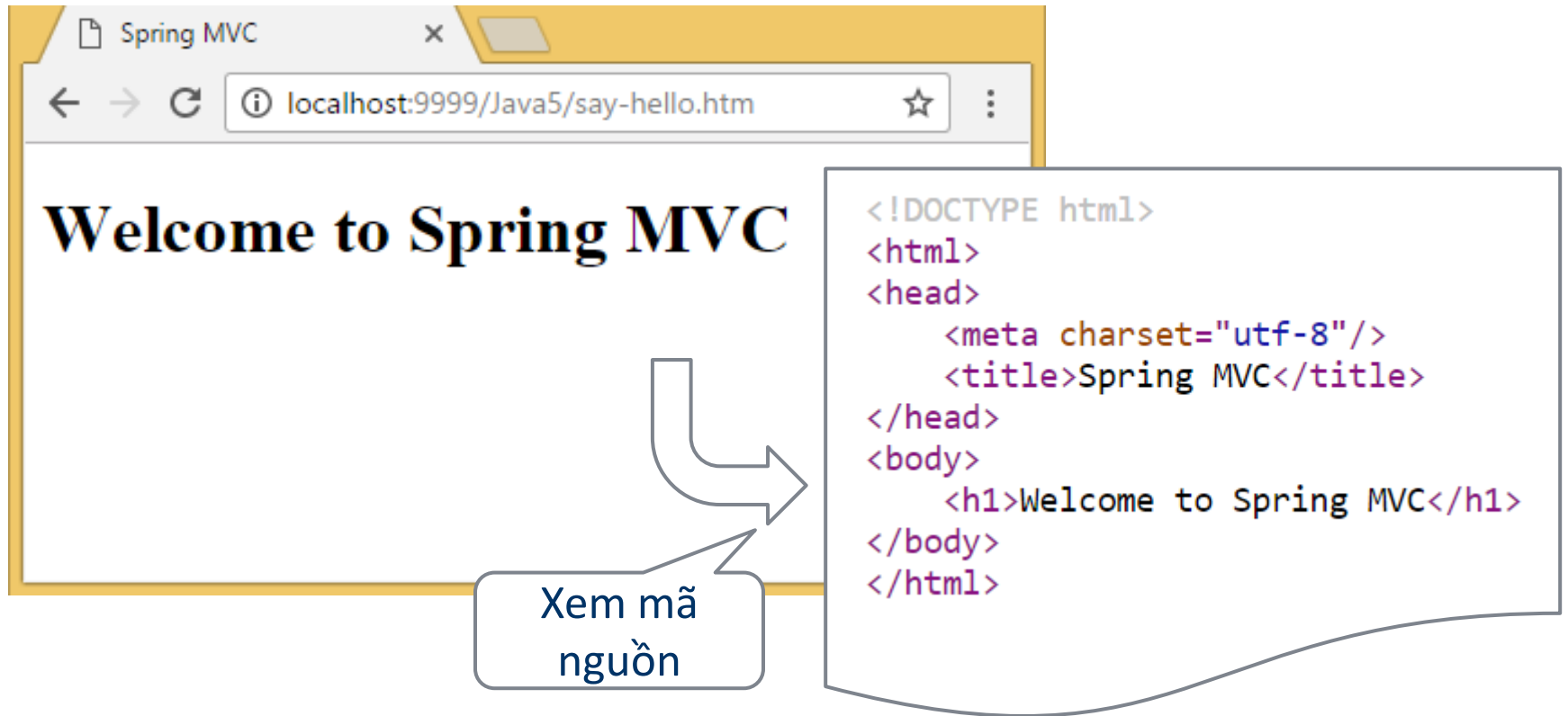
Tên giao dịch (URL)

Tên view

```
<%@ page pageEncoding="utf-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Spring MVC</title>
</head>
<body>
    <h1>Welcome to Spring MVC</h1>
</body>
</html>
```



- Chạy index.jsp sau đó nhập lại url như sau
  - ▣ <http://localhost:8080/HelloSpring/say-hello.htm>
- Sau đây là kết quả phản hồi

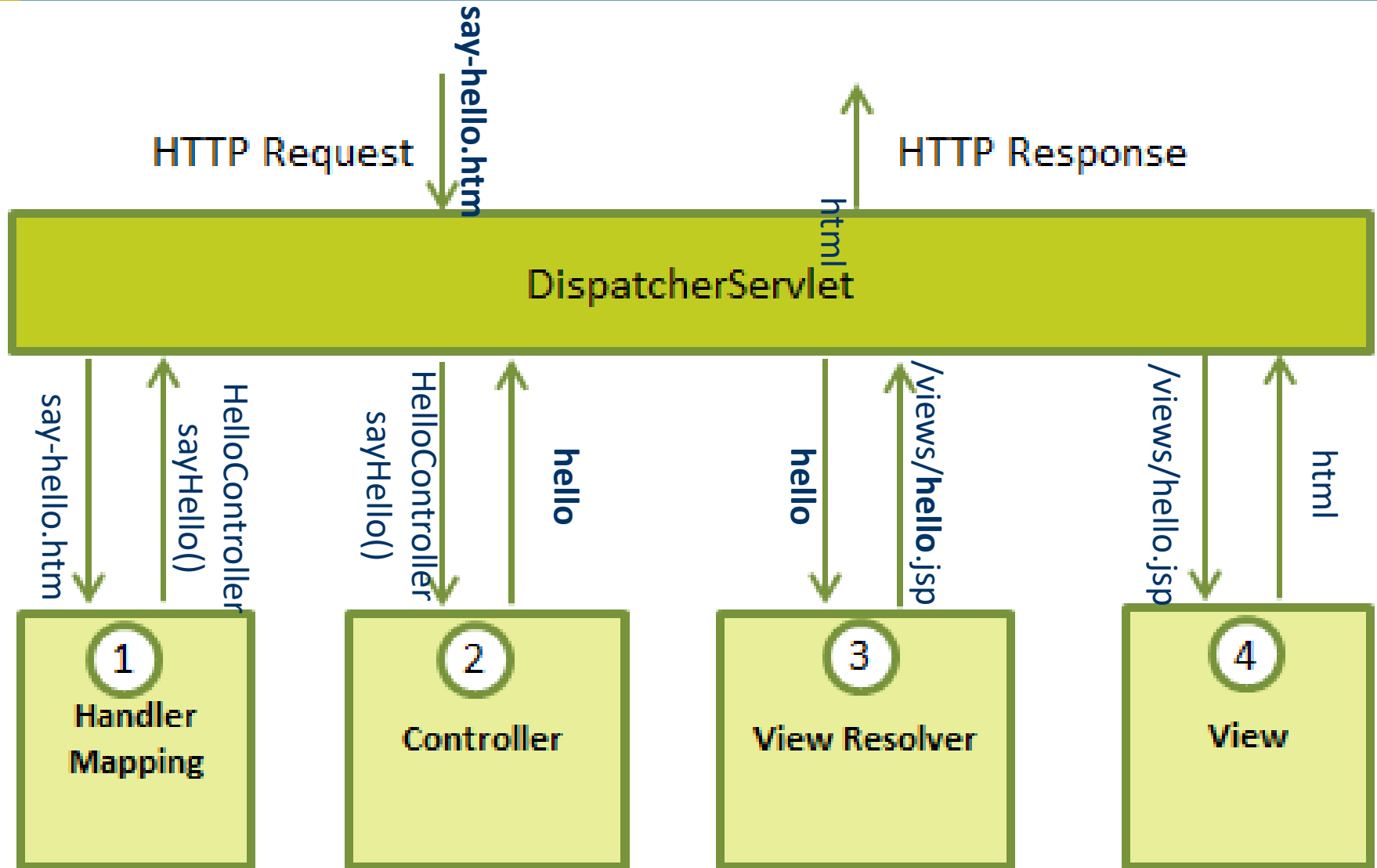


The image shows a web browser window with the title 'Spring MVC'. The address bar displays 'localhost:9999/Java5/say-hello.htm'. The main content area shows 'Welcome to Spring MVC'. A callout box labeled 'Xem mã nguồn' (View source) points to a code block containing the HTML source code for the page.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>Spring MVC</title>
</head>
<body>
  <h1>Welcome to Spring MVC</h1>
</body>
</html>
```

# Quy trình xử lý say-hello.htm

18



# Lưu ý ViewResolver

```
@Controller  
public class HelloController {  
    @RequestMapping("say-hello")  
    public String sayHello() {  
        return "hello";  
    }  
}
```

```
<bean id="viewResolver"  
    p:prefix="/WEB-INF/views/" p:suffix=".jsp"  
    class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
```

prefix + view + suffix  
/WEB-INF/views/hello.jsp

- DispatcherServlet sẽ nhận request với URL kết thúc .htm
  1. Chuyển **say-hello.htm** cho Handler Mapping và sẽ nhận được **sayHello()** của **HelloController** (do phương thức này được map với tên say-hello)
  2. Gọi sayHello() của HelloController và nhận được “**hello**” (do phương thức này return “hello”)
  3. Chuyển “**hello**” cho ViewResolver và nhận được “**/WEB-INF/views/hello.jsp**” (do ghép nối prefix + hello + suffix)
  4. Gọi hello.jsp và nhận kết quả HTML sau cùng là phản hồi cho người dùng

- Trong lập trình Servlet/JSP chúng ta đã được làm việc với các thành phần web sau
  - ▣ HttpServletRequest
    - Gói dữ liệu gửi từ client và chia sẻ cho nhiều Servlet/JSP hoạt động trên một request
  - ▣ HttpServletResponse
    - Gói dữ liệu chuyển về client
  - ▣ HttpSession
    - Phạm vi chia sẻ dữ liệu theo từng phiên làm việc khác nhau
  - ▣ ServletContext
    - Phạm vi chia sẻ dữ liệu trên toàn ứng dụng

- Trong Spring MVC bạn có thể truy xuất các đối tượng web một cách dễ dàng bằng cách định nghĩa chúng như những đối số của action method hoặc sử dụng **@Autowire**.

Đối với ServletContext bạn sử dụng **@Autowired** để tham chiếu đến

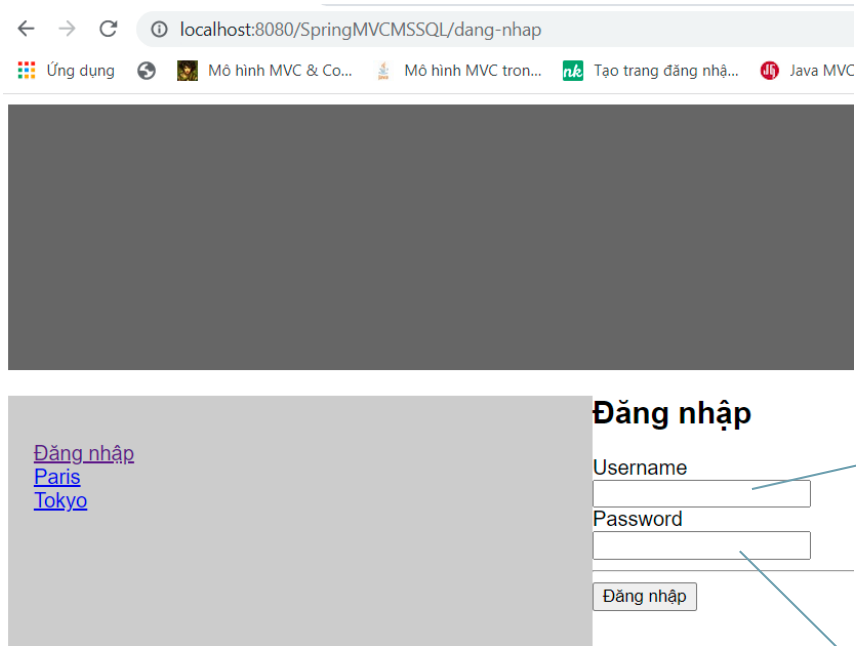
Khi bạn muốn làm việc với đối tượng nào bạn chỉ việc khai báo đối tượng đó như đối số của action method.

```
@Controller
public class UserController {
    @Autowired
    ServletContext application;

    @RequestMapping("say-hello")
    public String sayHello(
        HttpServletRequest request,
        HttpServletResponse response,
        HttpSession session) {
        System.out.println("index");
        return "user";
    }
}
```

# Ví dụ: Trang đăng nhập

23



## Đăng nhập

Sai username hoặc password

Username

Password

Đăng nhập

## Thông tin người dùng

UserName: trung

Password: 123

# Xây dựng UserController

24

@Controller

```
public class LoginController {
```

```
    @RequestMapping(value = {"/dang-nhap"})
```

```
    public String showForm() {
```

```
        return "clients/login";
```

```
    }
```

```
    @RequestMapping(value={"/dang-nhap"}, method=RequestMethod.POST)
```

```
    public String login(HttpServletRequest req) {
```

```
        String userName = req.getParameter("username");
```

```
        String passWord = req.getParameter("password");
```

```
        if (userName.equals("trung") && passWord.equals("123")){
```

```
            req.setAttribute("uid", userName);
```

```
            req.setAttribute("pwd", passWord);
```

```
            return "clients/info";
```

```
        }else {
```

```
            req.setAttribute("message", "Sai username hoặc password");
```

```
            return "clients/login";
```

```
        }}}
```

View này chứa form

Sử dụng request để nhận tham số và chia sẻ dữ liệu

Nhận tham số

Chia sẻ dữ liệu

View này hiển thị thông tin user



# Xây dựng các view

25

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<h2>Đăng nhập</h2>
${message}
<form action="dang-nhap" method="POST">
<div>Username</div>
<input name="username"/>
<div>Password</div>
<input name="password"/>
<hr>
<button>Đăng nhập</button>
</form>
```

Hiển thị dữ liệu  
truyền từ action

**clients/login.jsp**

```
<h3>USER INFO</h3>
<ul>
    <li>User Name: ${uid}</li>
    <li>Password: ${pwd}</li>
</ul>
```

Hiển thị dữ liệu  
truyền từ action

**clients/info.jsp**

- Bạn có thể sử dụng `request.setAttribute(name, value)` để truyền dữ liệu cho View
- Trong Spring MVC bạn có phương án khác chuẩn tắc hơn là sử dụng `ModelMap` làm đối số action method thay vì sử dụng `HttpServletRequest`

```
@Controller
```

```
public class HomeController {
```

```
@RequestMapping(value = { "/home" }, method = RequestMethod.GET)
```

```
public String Index(HttpServletRequest req) {
```

```
req.setAttribute("name", "Nguyễn Trung");
```

```
return "clients/index";
```

```
}
```

Trong JSP bạn có thể sử dụng `<%=request.getAttribute("name")%>` để truy xuất hoặc có thể sử dụng biểu thức EL `${name}` để truy xuất

```
@RequestMapping(value = { "/", "/trang-chu" }, method = RequestMethod.GET)
```

```
public String Index2(ModelMap model) {
```

```
model.addAttribute("name", "Nguyễn Hữu Trung");
```

```
return "clients/index";
```

```
}}
```

- Hiệu chỉnh action login theo hướng dẫn sau
  - ▣ Thêm đối số ModelMap model
  - ▣ Thay request.setAttribute() bằng model.addAttribute()

```
@RequestMapping(value={"/dang-  
nhap"},method=RequestMethod.POST)  
public String login(ModelMap model, HttpServletRequest req) {  
String userName = req.getParameter("username");  
String passWord = req.getParameter("password");  
if (userName.equals("trung") && passWord.equals("123")){  
model.addAttribute("uid", userName);  
model.addAttribute("pwd", passWord);  
return "clients/info";  
}else {  
model.addAttribute("message","Sai username hoặc password");  
return "clients/login";  
}}
```



## LẬP TRÌNH WEB (WEBPR330479)

Bean & DI trong Spring



THS. NGUYỄN HỮU TRUNG

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- [trungnh@hcmute.edu.vn](mailto:trungnh@hcmute.edu.vn)
- <https://www.youtube.com/@baigiai>



- ⦿ Hiểu Dependence Injection(DI) là gì?
- ⦿ Xây dựng và sử dụng Bean
- ⦿ Sử dụng @Autowire và @Qualifier
- ⦿ Sử dụng bean CommonsMultipartResolver để upload file lên server
- ⦿ Sử dụng bean JavaMailSender để gửi email
- ⦿ Xây dựng bean gửi email

- Giả sử chúng ta có lớp Company nắm giữ thông tin về doanh nghiệp như tên công ty, khẩu hiệu và logo. Trong website chúng ta muốn sử dụng lớp này để làm việc về thông tin doanh nghiệp.
- Rõ ràng các lớp trong website phụ thuộc vào lớp Company. Vì vậy khi chúng ta muốn thay đổi thông tin của doanh nghiệp thì phải hiệu chỉnh lại mã các lớp trong website và dịch lại ứng dụng
- Vấn đề đặt ra là làm thế nào để thay đổi thông tin doanh nghiệp mà không phải hiệu chỉnh lại mã của website.

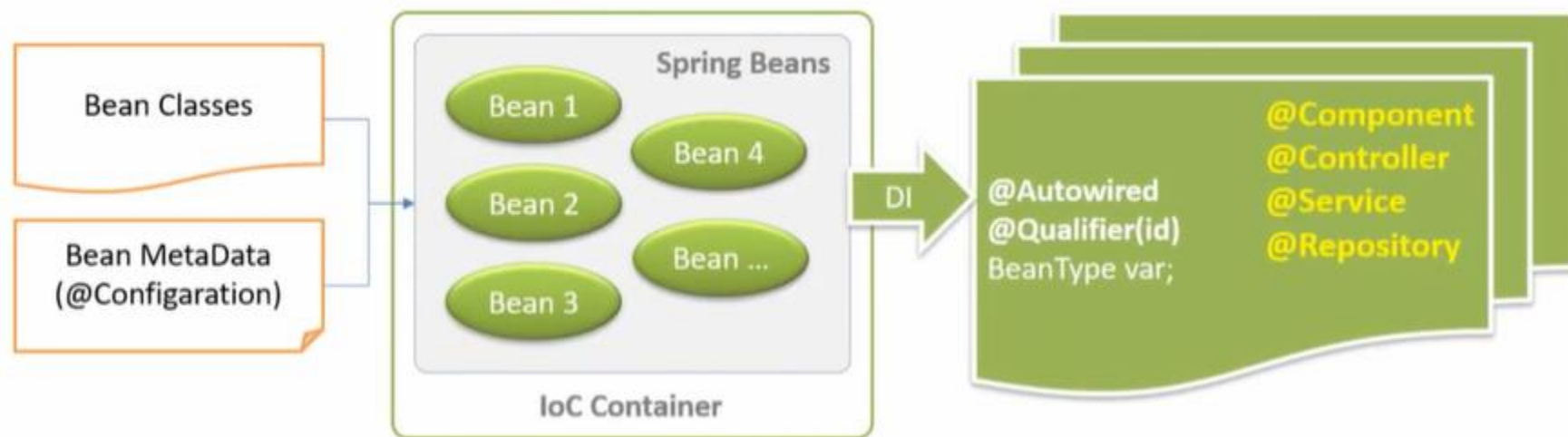


- DI là cách truyền một module vào một module khác thông qua cấu hình XML hay viết mã dưới sự hỗ trợ của DI container
- Spring framework có trang bị DI container nên có thể thực hiện DI một cách dễ dàng
- DI được dùng để làm giảm sự phụ thuộc giữa các module, dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.

- Để cụ thể hóa DI chúng ta xét lớp bean Company gồm 3 thuộc tính
  - ▣ Name: tên công ty
  - ▣ Slogan: khẩu hiệu
  - ▣ Logo: ảnh logo

```
public class Company {  
    private String name;  
    private String slogan;  
    private String logo;  
  
    getters/setters  
}
```

- ❑ Spring Bean là các bean được Spring quản lý gồm:
  - ❑ Bean hệ thống: `HttpServletRequest`, `HttpServletResponse`, `HttpSession`, `ServletContext`,...
  - ❑ Bean do người dùng nạp vào



- ❑ Spring Beans = <Built-in beans> + User-defined beans>
- ❑ DI(tiêm) = liên kết đến bean cần thiết
- ❑ IoC Container (Inversion of Control = “Đảo ngược điều khiển”) là engine làm nhiệm vụ nạp bean từ bên ngoài vào hệ thống nhằm cho phép điều khiển theo mã tùy biến của bean bên ngoài.

- Có thể yêu cầu IoC Container nạp bean bằng 2 cách:
  - ▣ Cách 1: tạo lớp có đính kèm các annotation sau:
    - @Controller
    - @Component
    - @Service
    - @Repository
  - ▣ Cách 2: tạo lớp bất kỳ và viết phương thức bean kèm @Bean khai báo trong file cấu hình @Configuration
- Khi khởi động ứng dụng, spring sẽ tìm kiếm các annotation đã được định nghĩa trên các lớp và các phương thức cùng với các @Scope để tạo và nạp các đối tượng vào thời điểm phù hợp

- Mong muốn tạo một đối tượng từ `Company` chứa thông tin của một doanh nghiệp và được sử dụng trong website nhưng khi thay đổi thông tin sang doanh nghiệp khác thì không phải dịch lại website
- Để đạt được mong muốn trên bạn cần khai báo bean trong file cấu hình của Spring. DI container sẽ tạo đối tượng khi khởi khởi động.

```
<bean id="iotstar" class="vn.iotstar.bean.Company">  
  <property name="name" value="LTWeb JAVA"></property>  
  <property name="slogan" value="Học thực - Giá trị thực"/>  
  <property name="logo"  
    value="templates/logo/iotstar.png"/>  
</bean>
```

- @Autowired
  - ▣ Dựa vào kiểu dữ liệu của biến để tìm kiếm và liên kết đến Spring Bean có kiểu phù hợp
  - ▣ Nếu nhiều hơn 01 Spring Bean có kiểu phù hợp thì hệ thống sẽ báo lỗi
- Giải quyết xung đột
  - ▣ Cách 1: Cấu hình @Bean với @Primary để báo cho hệ thống biết đó là bean chính
  - ▣ Cách 2: đặt id cho bean và sử dụng @Qualifier(id) bên cạnh @Autowired

@Controller

@RequestMapping("/home/")

**public class** HomeController {

**@Autowired**

Company company;

Bean đã được tiêm vào và sẵn sàng phục vụ các action trong Controller

@RequestMapping("index")

**public** String index(ModelMap model) {

model.addAttribute("company", company);

**return** "home/index";

}

}

Sử dụng bean đã tiêm vào

- View index.jsp được thiết kế để hiển thị thông tin doanh nghiệp.

```
<body>  
    <h1>${company.name}</h1>  
      
    <div>${company.slogan}</div>  
    <hr>  
</body>
```

**@Autowired**  
**Account account;**

Liên kết với bean này

```
@Configuration
public class HelloConfig {
    @Bean
    public Account getAccount1() {
        return new Account();
    }

    @Primary @Bean
    public Account getAccount2() {
        return new Account();
    }
}
```

Bean theo kiểu và id.

**@Autowired**  
**@Qualifier("bean1")**  
**Account account;**

Liên kết với bean này

```
@Configuration
public class HelloConfig {
    @Bean("bean1")
    public Account getAccount1() {
        return new Account();
    }

    @Bean("bean2")
    public Account getAccount2() {
        return new Account();
    }
}
```



- Tiêm vào Field
  - ▣ @Autowired
  - ▣ @Qualifier("bean1")

Account account;
- Tiêm vào setter
  - ▣ @Autowired

Public void setAccount(@Qualifier("bean1") Account account){}
- Tiêm vào constructor
  - ▣ @Autowired

Public MyController(@Qualifier("bean1") Account account){}

- **@Autowired** được sử dụng để tiêm bean vào Controller dưới 3 hình thức sau
  - ▣ Tiêm vào **field**
  - ▣ Tiêm thông qua **constructor**
  - ▣ Tiêm thông qua **setter**

```
@Controller
@RequestMapping("/home/")
public class HomeController {
    @Autowired
    Company company;
```

Tiêm vào field

```
@Controller
@RequestMapping("/home/")
public class HomeController {
    Company company;
    @Autowired
    public HomeController(Company company) {
        this.company = company;
    }
```

Tiêm thông qua constructor

```
@Controller
@RequestMapping("/home/")
public class HomeController {
    Company company;
    @Autowired
    public void setCompany(Company company) {
        this.company = company;
    }
```

Tiêm thông qua phương thức setter

- Bằng cách nào để DI container nhận biết được bean nào để truyền vào cho Controller khi sử dụng **@Autowired**?
- **@Autowired** sẽ nhận biết bean thông qua **kiểu dữ liệu**.

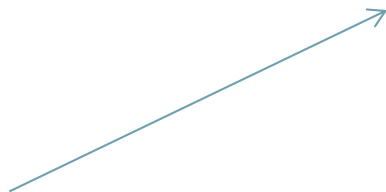
```
@Controller  
@RequestMapping("/home/")  
public class HomeController {  
    @Autowired  
    Company company;  
}
```

```
<bean id="iotstar" class="vn.iotstar.bean.Company">  
    <property name="name" value="LTWeb JAVA"></property>  
    <property name="slogan" value="Học thực - Giá trị thực"/>  
    <property name="Logo" value="templates/logo/iotstar.png"/>  
</bean>
```

- Khi có nhiều bean cùng kiểu dữ liệu thì **@Autowired** không là chưa đủ để xác định bean nào được truyền vào mà cần phải có thêm **@Qualifier** để nhận biết qua id

@Controller

```
public class HomeController {  
    @Autowired @Qualifier("iotstar")
```



```
<bean id="iotstar" class="vn.iotstar.bean.Company">  
    <property name="name" value="LTWeb JAVA"></property>  
    <property name="slogan" value="Học thực - Giá trị thực"/>  
    <property name="logo" value="templates/logo/iotstar.png"/>  
</bean>
```

- Lớp bean được chú thích bởi **@Component** hoặc **@Service**, **@Repository** sẽ tự khai báo mà bạn không cần phải khai báo bằng tay vào file cấu hình.
- Tuy nhiên bạn cần phải khai báo package chứa bean vào  
`<context:component-scan`  
`base-package="vn.iotstar.controller, vn.iotstar.components"/>`

Sử dụng **dấu phẩy** để  
phân cách các  
package.

```
package vn.iotstar.components;
import org.springframework.stereotype.Component:
@Component("mailer")
public class Mailer {
public void send (String from, String to, String subject, String body) {
}
}
@Controller
@RequestMapping("/mailer/")
public class MailerController {
@Autowired
Mailer mailer;
@RequestMapping("send")
public String send(ModelMap model) {
String from="trungnh@hcmute.edu.vn";
String to="trungnhspkt@gmail.com";
String subject = "Hello";
String body="Test mail hệ thống";
mailer.send(from, to, subject, body);
return "mail/form";
}
}
```

Bean tự khai báo với id là  
mailer

```
@Service
public class CookieService{
    @Autowired HttpServletRequest request;
    @Autowired HttpServletResponse response;
    public Cookie create(String name, String value, int days) {
        Cookie cookie = new Cookie(name, value);
        cookie.setMaxAge(days * 60 * 60);
        cookie.setPath("/");
        return cookie;
    }
    public Cookie get(String name) {
        Cookie[] cookies = request.getCookies();
        if(cookies != null)
            for(Cookie cookie: cookies)
                if(cookie.getName().equalsIgnoreCase(name)) return cookie;
        return null;
    }
}
```

**@Service**

```
public class SessionService{  
    @Autowired  
    HttpSession session;  
    public void setAttribute(String name, Object value) {  
        session.setAttribute(name, value);  
    }  
    public <T> T getAttribute(String name) {  
        return (T) session.getAttribute(name);  
    }  
    public void removeAttribute(String name) {  
        session.removeAttribute(name);  
    }  
}
```



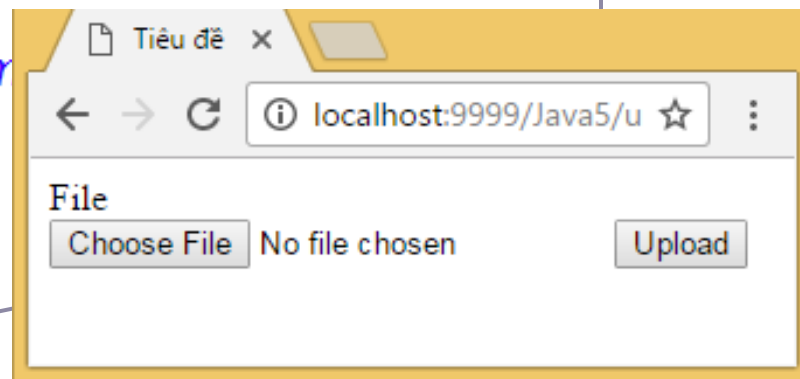
- Upload file là một chức năng quan trọng trong ứng dụng web
- Các ứng dụng thường gặp
  - ▣ Gửi mail có kèm file
  - ▣ Upload hình đại diện trên facebook, gmail...
  - ▣ Upload video lên Youtube
  - ▣ Nộp hồ sơ xin việc
  - ▣ Nộp bài học lên LMS
  - ▣ ...

- Để upload file, trước hết bạn cần khai báo bean **CommonsMultipartResolver** vào file cấu hình

```
<bean id="multipartResolver"  
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  
<!-- maxUploadSize = 20MB -->  
<property name="maxUploadSize" value="20971520"></property>  
</bean>
```

- Mặc định tổng kích thước file là 2MB. Bạn có thể cấu hình thuộc tính **maxUploadSize** để thay đổi thông số này
- Thư viện cần thiết
  - **commons-fileupload-1.2.2.jar**
  - **commons-io-1.3.2.jar**

```
${message}  
<form action="uploader/upload.htm"  
      method="post" enctype="multipart/form-data">  
  <div>File</div>  
  <input type="file" name="image">  
  <button>Upload</button>  
</form>
```



- Form upload file bắt buộc các thuộc tính
  - ▣ **method="POST"**
  - ▣ **enctype="multipart/form-data"**

```
@RequestMapping("upload")
public String upload(ModelMap model, @RequestParam("image") MultipartFile image) {
    if(image.isEmpty()){
        model.addAttribute("message", "Vui lòng chọn file !");
    }
    else{
        try {
            String path = context.getRealPath("/images/" + image.getOriginalFilename());
            image.transferTo(new File(path));

            model.addAttribute("name", image.getOriginalFilename());
            model.addAttribute("type", image.getContentType());
            model.addAttribute("size", image.getSize());
            return "uploader/success";
        }
        catch (Exception e) {
            model.addAttribute("message", "Lỗi lưu file !");
        }
    }
    return "uploader/form";
}
```

**@Service**

```
public class UploadService {  
    @Autowired  
    ServletContext app;  
    public File save(MultipartFile file, String folder) {  
        File dir = new File(app.getRealPath(folder));  
        if(!dir.exists()) dir.mkdirs();  
        try {  
            File saveFile = new File(dir, file.getOriginalFilename());  
            file.transferTo(saveFile);  
            return saveFile;  
        } catch (Exception e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

Phương thức	Công dụng
<code>isEmpty()</code>	Kiểm tra xem có file upload không
<code>getOriginalFilename()</code>	Lấy tên file gốc
<code>transferTo(File)</code>	Chuyển file đến đường dẫn mới
<code>getContentType()</code>	Lấy kiểu file
<code>getSize()</code>	Lấy kích thước file
<code>getBytes()</code>	Lấy nội dung file
<code>getInputStream()</code>	Lấy luồng dữ liệu để đọc file

```
  
<ul>  
  <li>File Name: ${name}</li>  
  <li>File Size: ${size}</li>  
  <li>File Type: ${type}</li>  
</ul>
```

- Chức năng gửi email đóng vai trò vô cùng quan trọng trong ứng dụng web

- Email kích hoạt tài khoản

*Thông thường sau khi đăng ký thành viên thành công hệ thống sẽ gửi cho chúng ta một email chào và có liên kết để kích hoạt tài khoản.*

- Đơn đặt hàng

*Sau khi đặt hàng chúng ta cũng nhận được email báo đơn hàng*

- Quên mật khẩu

*Mật khẩu sẽ được gửi qua email nếu chúng ta cung cấp thông tin hợp lệ*

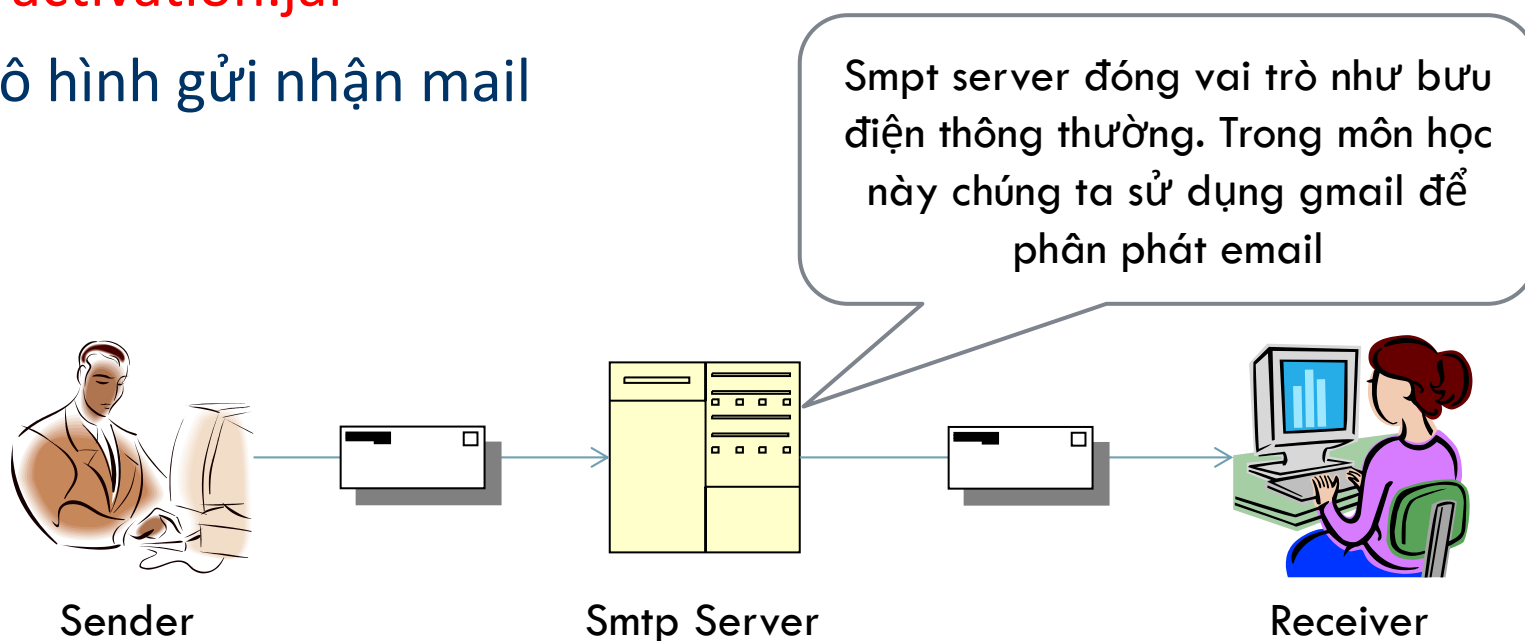
- Gửi thông tin cho bạn bè

*Khi xem hàng hóa trên internet nếu thấy hàng hóa đó phù hợp với bạn mình thì có thể gửi thông tin hàng hóa đó cho bạn của mình.*

- ...



- Spring cung cấp bean JavaMailSender giúp thực hiện chức năng gửi email rất thuận tiện.
- Thư viện cần thiết cho bean này gồm
  - ▣ **mail.jar**
  - ▣ **activation.jar**
- Mô hình gửi nhận mail



- Khai báo bean JavaMailSender có cấu hình để gửi email thông qua Gmail như sau

```
<!-- Spring Mail Sender -->
<bean id="mailSender"
class="org.springframework.mail.javamail.JavaMailSenderImpl">
<property name="host" value="smtp.gmail.com" />
<property name="port" value="465" />
<property name="username" value="trungnhspkt@gmail.com" />
<property name="password" value="" />
<property name="defaultEncoding" value="UTF-8" />
<property name="javaMailProperties">
<props>
<prop key="mail.smtp.auth">true</prop>
<prop key="mail.smtp.socketFactory.class">
javax.net.ssl.SSLSocketFactory
</prop>
<prop key="mail.smtp.socketFactory.port">465</prop>
<prop key="mail.debug">true</prop>
<prop key="mail.smtp.starttls.enable">true</prop>
</props>
</property>
</bean>
```

Tài khoản Smtplib được sử dụng để phát mail đến người nhận

- Bạn phải đăng ký 1 tài khoản Gmail thông thường sau đó đăng nhập vào gmail và tiến hành kích hoạt thông qua liên kết sau

<https://www.google.com/settings/security/lesssecureapps>

## Less secure apps

Some apps and devices use less secure sign-in technology, which makes your account more vulnerable. You can **turn off** access for these apps, which we recommend, or **turn on** access if you want to use them despite the risks. [Learn more](#)

Access for less secure apps

☐ Turn off

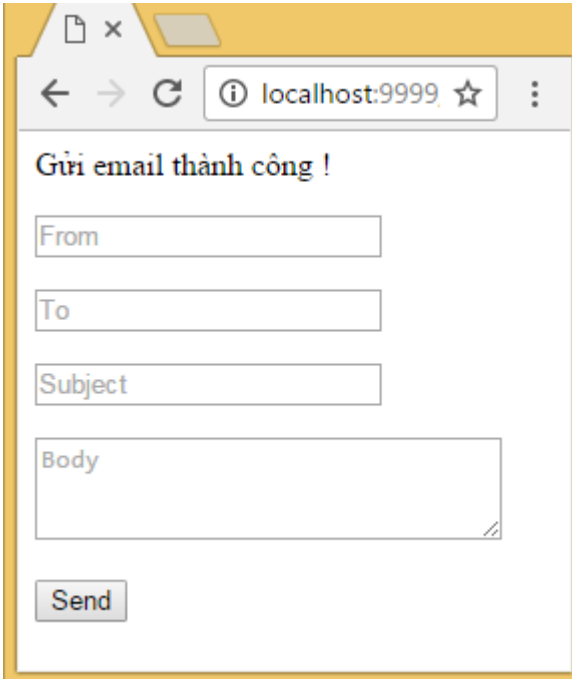
☒ Turn on

Chọn mục 'Turn on' để kích hoạt tài khoản đang đăng nhập trước khi sử dụng nó để phân phát email qua ứng dụng

# Form gửi mail

60

```
<h2>Send Email Form</h2>
${error}
<form:form action="send.htm" method="post"
enctype="multipart/form-data" modelAttribute="mail">
<div>From:</div>
<form:input path="from" />
<div>To:</div>
<form:input path="to" />
<div>Subject:</div>
<form:input path="subject" />
<div>Body:</div>
<form:textarea path="body" rows="3" cols="30"/>
<div>Attachment File:</div>
<input name="attachment" type="file">
<hr>
<input type="submit" value="Send">
</form:form>
```



The screenshot shows a web browser window with the address bar displaying 'localhost:9999'. The page content includes a success message 'Gửi email thành công !' (Email sent successfully!). Below the message are four input fields: 'From', 'To', 'Subject', and 'Body'. The 'Body' field is a text area. At the bottom of the form is a 'Send' button.

```
@Controller
public class EmailController {

    @Autowired
    ServletContext context;

    @Autowired
    JavaMailSender mailSender;

    @RequestMapping(value = "input", method =
    RequestMethod.GET)
    public String showForm(ModelMap model) {
        model.addAttribute("mail", new EmailInfo());
        return "AttachEmailInput";
    }
}
```

Tiêm bean vào để sử dụng

```
@RequestMapping(value = "send", method =  
RequestMethod.POST)  
public String sendWithAttach(ModelMap model,  
@ModelAttribute("mail") EmailInfo mailInfo,  
@RequestParam("attachment") MultipartFile file) {  
try {
```

```
MimeMessage message =  
mailSender.createMimeMessage();  
MimeMessageHelper helper = new  
MimeMessageHelper(message, true);  
helper.setFrom(mailInfo.getFrom());  
helper.setTo(mailInfo.getTo());  
helper.setReplyTo(mailInfo.getFrom());  
helper.setSubject(mailInfo.getSubject());  
helper.setText(mailInfo.getBody(), true);
```

Tạo một email

```
if (!file.isEmpty()) {  
    String imageUrl = "uploads/" + file.getOriginalFilename();  
    String absolutePath = context.getRealPath(imageUrl);  
    File uploadFile = new File(absolutePath);  
    file.transferTo(uploadFile);
```

```
    helper.addAttachment(uploadFile.getName(), uploadFile);
```

```
    model.addAttribute("imageUrl", imageUrl);  
}
```

Gửi email

```
mailSender.send(message);  
} catch (Exception ex) {  
    model.addAttribute("error", ex.getMessage());  
    return "AttachEmailInput";  
}  
return "AttachEmailSuccess";  
}
```

- Trước hết phải upload file
  - ▣ `<form action="mailer/send.htm"`  
`method="post" enctype="multipart/form-data">`
  - ▣ `public String send(...`  
`@RequestParam("attach") MultipartFile attach)`
- Sau đó đính kèm file với phương thức `addAttachment(name, file)`  
`String fileName = attach.getOriginalFilename();`  
`String path = context.getRealPath("/images/" + fileName);`  
`helper.addAttachment(fileName, new File(path));`



## JavaMailSender

Phương thức	Công dụng
createMimeMessage()	Tạo mail
Send(mail)	Gửi mail

## MimeMessageHelper

Phương thức	Công dụng
setFrom(email, name)	Cấp thông tin người gửi
setTo(email)	Email người nhận
setCc(emails)	Danh sách email cùng nhận
setBcc(emails)	Danh sách email cùng nhận ẩn danh
setReplyTo(email, name)	Cấp thông tin người nhận phản hồi
setSubject(subject)	Tiêu đề email
setText(body, isHtml)	Nội dung email
addAttachment(name, file)	File đính kèm

```
package vn.iotstar.model;
public class EmailInfo {
    private String from, to, subject, body;
    public String getFrom() {return from;}
    public void setFrom(String from) {
        this.from = from;}
    public String getTo() {return to;}
    public void setTo(String to) {
        this.to = to;}
    public String getSubject() {
        return subject;}
    public void setSubject(String subject) {
        this.subject = subject;}
    public String getBody() {return body;}
    public void setBody(String body) {
        this.body = body;}
    }
}
```



LẬP TRÌNH WEB (WEBPR330479)

# Controller trong Spring MVC

THS. NGUYỄN HỮU TRUNG

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- [trungnh@hcmute.edu.vn](mailto:trungnh@hcmute.edu.vn)
- <https://www.youtube.com/@baigiai>



- ◉ Sử dụng thành thạo @RequestMapping
  - ◉ Ánh xạ nhiều action
  - ◉ Ánh xạ phân biệt POST|GET
  - ◉ Ánh xạ phân biệt tham số
- ◉ Nắm vững phương pháp nhận tham số
  - ◉ Sử dụng HttpServletRequest
  - ◉ Sử dụng @RequestParam
  - ◉ Sử dụng JavaBean
  - ◉ Sử dụng @PathVariable để nhận dữ liệu từ URL
- ◉ Sử dụng @CookieValue để nhận cookie
- ◉ Hiểu rõ kết quả của phương thức action

- Annotation **@RequestMapping** được sử dụng để ánh xạ một action đến một phương thức action trong Controller

```
@RequestMapping(" /home")  
public String Index(HttpServletRequest req) {  
    req.setAttribute("name", "Nguyễn Hữu Trung");  
    return "clients/index";  
}
```

- Khi người dùng đưa ra yêu cầu **index.htm** thì phương thức action **Index(HttpServletRequest req)** sẽ thực hiện
- Trong một lớp **@Controller** có thể chứa nhiều phương thức action.

# @RequestMapping (2)

71

- @RequestMapping("/home") là cách viết thu gọn của @RequestMapping(value="/home")
- @RequestMapping() có thể được sử dụng để đặt trên lớp **Controller** để ánh xạ chung cho nhiều action method

```
@Controller
@RequestMapping("/home/")
public class HomeController {
    @RequestMapping("index")
    public String index() {
        return "home/index";
    }
    @RequestMapping("about")
    public String about() {
        return "home/about";
    }
}
```

← home/index.htm

← home/about.htm

# @RequestMapping (3)

72

```
@Controller
public class HomeController {
    @RequestMapping("/home/index")
    public String index() {
        return "home/index";
    }
    @RequestMapping("/home/about")
    public String about() {
        return "home/about";
    }
}
```

**home/index.htm**

**home/about.htm**

```
@Controller
@RequestMapping("/home/")
public class HomeController {
    @RequestMapping("index")
    public String index() {
        return "home/index";
    }
    @RequestMapping("about")
    public String about() {
        return "home/about";
    }
}
```

- Hai cách ánh xạ này hoàn toàn tương đương nhau



- Trong Servlet khi yêu cầu từ người dùng gửi đến server với phương thức web là GET thì phương thức doGet() của Servlet được thực hiện, ngược lại nếu phương thức web là POST thì doPost() được thực hiện
- Chú ý:
  - Trường hợp POST duy nhất là khi bạn submit một form có thuộc tính method="POST".
  - Các trường hợp GET thường gặp
    - Nhập url vào ô địa chỉ của trình duyệt web
    - Nhấp vào liên kết
    - Submit form với method="GET"

- Trong Spring MVC phân biệt POST|GET thông qua tham số method của phương thức action

```
@RequestMapping(value="login", method=RequestMethod.GET)  
public String login() {  
    return "user/login";  
}
```

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(ModelMap model, HttpServletRequest request) {  
    return "user/login";  
}
```

- Như vậy khi yêu cầu user/login.htm được gửi đến server, Spring MVC sẽ gọi phương thức login() nào là tùy thuộc vào phương thức web **GET** hay **POST**

- Thông thường GET là để vào giao diện còn POST được sử dụng để xử lý các nút chức năng

```
@RequestMapping(value = {"/dang-nhap"}, method=RequestMethod.GET)  
public String showForm() {  
    return "clients/login";  
}
```

```
@RequestMapping(value={"/dang-nhap"}, method=RequestMethod.POST)  
public String login(ModelMap model, HttpServletRequest req) {  
    String userName = req.getParameter("username");  
    String passWord = req.getParameter("password");  
    if (userName.equals("trung") && passWord.equals("123")){  
        model.addAttribute("uid", userName);  
        model.addAttribute("pwd", passWord);  
        return "clients/info";  
    } else {  
        model.addAttribute("message", "Sai username hoặc password");  
        return "clients/login";  
    }  
}
```

- Trong Spring MVC không những hỗ trợ gọi phương thức action phân biệt theo phương thức web mà còn cho phép phân biệt theo tham số truyền theo

```
@RequestMapping(value = { "/home"}, method = RequestMethod.GET,  
params = "id")  
public String Index(HttpServletRequest req) {  
    req.setAttribute("name", "Nguyễn Trung");  
    return "clients/index";  
}
```

**index.htm?id**

- Với định nghĩa này khi gọi **index.htm** phải có tham số **id** thì phương thức **Index()** mới được thực hiện

- Khi yêu cầu student.htm thì phương thức nào sẽ thực hiện?

```
@Controller
@RequestMapping("student")
public class StudentController {
    @RequestMapping()
    public String index(ModelMap model) {...}

    @RequestMapping(params="btnInsert")
    public String insert(ModelMap model) {...}

    @RequestMapping(params="btnUpdate")
    public String update(ModelMap model) {...}

    @RequestMapping(params="btnDelete")
    public String delete(ModelMap model) {...}

    @RequestMapping(params="lnkEdit")
    public String edit(ModelMap model) {...}
}
```

- Nếu có tham số **btnInsert**, **btnUpdate**, **btnDelete** hoặc **lnkEdit** thì các phương thức **insert()**, **update()**, **delete()** hoặc **edit()** sẽ thực hiện
- Nếu không có các tham số trên thì **index()** sẽ thực hiện
- Nếu có nhiều hơn 1 tham số trên thì sẽ báo lỗi

- [Thêm]=>insert()
- [Cập nhật]=>update()
- [Xóa]=>delete()
- [Nhập lại]=>index()
- [Sửa]=>edit()

<a href="student/index.htm?lnkEdit">Sửa</a>

- Tham số là dữ liệu truyền đến server khi có yêu cầu từ người dùng dưới dạng các trường của form hoặc chuỗi truy vấn của liên kết

- Ví dụ

- ▣ Khi nhấp vào liên kết sau thì các tham số **mark** và **name** sẽ được truyền đến phương thức action

`<a href="index.htm?mark=5&name=Trung">Xin chào</a>`

- ▣ Khi nhấp vào nút Hello của form sau thì các tham số **mark** và **name** sẽ được truyền đến phương thức action

`<form action="index.htm">`

`<input name="mark">`

`<input name="name">`

`<button>OK</button>`

`</form>`

- Spring MVC cung cấp các phương pháp nhận tham số sau đây
  - ▣ Sử dụng **HttpServletRequest** tương tự Servlet
  - ▣ Sử dụng **@RequestParam**
  - ▣ Sử dụng **JavaBean**
  - ▣ Sử dụng **@PathVariable** để nhận một phần trên URL

- Chỉ cần thêm đối số HttpServletRequest vào phương thức action là có thể nhận được tham số người dùng như Servlet

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(HttpServletRequest request) {  
    String id = request.getParameter("id");  
    String pw = request.getParameter("password");  
    // ...  
    return "login";  
}
```



# Sử dụng @RequestParam (1)

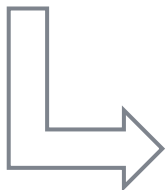
81

- Sử dụng **@RequestParam** thể hiện tính chuyên nghiệp hơn và có thể chuyển đổi tự động sang kiểu mong muốn.
- Ví dụ sau được sử dụng để nhận các tham số có tên là id và password

```
@RequestMapping(value="login", method=RequestMethod.POST)  
public String login(@RequestParam("id") String id,  
                    @RequestParam("password") String password) {  
    // ...  
    return "login";  
}
```

- **@RequestParam**(value, defaultValue, required) là dạng đầy đủ với ý nghĩa của các tham số:
  - ▣ value: chỉ ra tên tham số muốn nhận
  - ▣ defaultValue: là giá trị mặc định của tham số khi tham số không tồn tại
  - ▣ required: tham số có bắt buộc hay không
- Ví dụ với khai báo nhận tham số sau
  - ▣ **@RequestParam**(value="tuoi", defaultValue="40", required=false) **Integer age**
    - Tên tham số là **tuoi** sẽ được nhận vào đối số là **age**
    - Nếu không có tham số thì giá trị của **age** là 40
    - Tham số **tuoi** là không bắt buộc

```
@RequestMapping(value={"/dang-nhap"},method=RequestMethod.POST)
public String login(ModelMap model, HttpServletRequest req) {
    String userName = req.getParameter("username");
    String passWord = req.getParameter("password");
    if (userName.equals("trung") && passWord.equals("123")){
        model.addAttribute("uid", userName);
        model.addAttribute("pwd", passWord);
        return "clients/info";
    }else {model.addAttribute("message","Sai username hoặc password");
    return "clients/login";
    }
}
```



```
@RequestMapping(value={"/dang-nhap"},method=RequestMethod.POST)
public String login(ModelMap model, @RequestParam("username")String
    userName,
    @RequestParam("password")String passWord) {
    if (userName.equals("trung") && passWord.equals("123")){
        model.addAttribute("uid", userName);
        model.addAttribute("pwd", passWord);
        return "clients/info";
    }else {model.addAttribute("message","Sai username hoặc password");
    return "clients/login";
    }}
}
```

- Lớp JavaBean là lớp thỏa mãn các qui ước sau
  - ▣ Phải được định nghĩa là public
  - ▣ Phải có constructor không tham số
  - ▣ Đọc ghi dữ liệu thông qua getter/setter

```
package vn.iotstar.model;  
public class UserModel {  
    private String userName;  
    private String passWord;  
    public String getUsername() {return userName;}  
    public void setUsername(String userName) {  
        this.userName = userName;  
    }  
    public String getPassword() {return passWord;}  
    public void setPassword(String passWord) {  
        this.passWord = passWord;  
    }  
}
```

- Thuộc tính của bean được xác định từ các getter và setter bằng cách
  - ▣ Bỏ get và set và đổi ký tự đầu tiên của phần còn lại sang ký tự thường
- Ví dụ lớp UserModel có 2 thuộc tính cho phép đọc/ghi là userName và passWord
  - ▣ Thuộc tính userName được xác định từ **getUserName()** và **setUserName()**
  - ▣ Thuộc tính password được xác định từ **getPassWord()** và **setPassWord()**
- *Chú ý quan trọng: các trường dữ liệu không phải là thuộc tính của bean*

- Spring MVC cho phép sử dụng JavaBean để nhận các tham số **cùng tên** với các thuộc tính của bean.

```
@RequestMapping(value={"/dang-nhap"},method=RequestMethod.POST)  
public String login(ModelMap model, UserModel user) {  
    if (user.getUserName().equals("trung") &&  
        user.getPassWord().equals("123")){  
        model.addAttribute("uid", user.getUserName());  
        model.addAttribute("pwd", user.getPassWord());  
        return "clients/info";  
    }else {  
        model.addAttribute("message","Sai username hoặc password");  
        return "clients/login";  
    }  
}
```

- Với ví dụ này thì các thuộc tính username và password của đối số user sẽ nhận các giá trị từ các tham số cùng tên là username và password.

# Sử dụng @PathVariable

87

- Spring MVC cho phép nhận một phần dữ liệu từ đường dẫn URL
- Ví dụ action edit() sau đây sẽ lấy được tên sinh viên từ URL student/**Nguyễn Hữu Trung**.htm

```
@Controller
@RequestMapping("/student")
public class StudentController {
    @RequestMapping(value="/{name}", params="lnkEdit")
    public String edit(ModelMap model, @PathVariable("name") String name) {
        ...
        return "student";
    }
}
```

`<a href="student/Nguyễn Hữu Trung.htm?lnkEdit">Sửa</a>`

- Trong Servlet bạn có thể nhận cookie thông qua `HttpServletRequest`. Phương pháp này viết mã khá dài dòng, phức tạp.
- Trong Spring MVC bạn có thể sử dụng **@CookieValue** để nhận dữ liệu từ cookie

```
@RequestMapping(value = {"/", "/trang-chu" }, method =  
RequestMethod.GET)  
public String Index3(ModelMap model, @CookieValue("username")  
String userName) {  
    model.addAttribute("name", "Nguyễn Hữu Trung");  
    return "clients/index";  
}
```

- Ví dụ này cho phép sử dụng đối số **userName** để nhận giá trị của cookie có tên là **username**



- @CookieValue(**value**, **defaultValue**, **required**) có 3 tham số và ý nghĩa như sau
  - ▣ Value: tên cookie muốn nhận dữ liệu
  - ▣ defaultValue: giá trị mặc định của cookie
  - ▣ Required: có bắt buộc cookie username có tồn tại hay không
- Ví dụ
  - ▣ @CookieValue(value="username",       defaultValue="trung",  
required=false) String userName
    - Sử dụng đối số username để nhận giá trị của cookie có tên là username
    - Nếu cookie không tồn tại thì giá trị của userName là trung
    - Cookie này cho phép không tồn tại

- Return của phương thức action không đơn thuần phải là tên của view mà có thể là 1 trong 3 trường hợp sau
  - ▣ Tên view => ViewResolver sẽ xử lý để xác định view
    - return “<tên view>”
  - ▣ Nội dung => được trả trực tiếp về client mà không qua ViewResolver. Trường hợp này phương thức action phải được chú thích bởi **@ResponseBody**
    - return “<Nội dung>”
  - ▣ Lời gọi một action khác
    - return “**redirect:/<action>**”

# Đầu ra của phương thức action

91

```
@RequestMapping(value = { "/home"})  
public String Index() {  
return "clients/index";  
}
```

→ **/WEB-INF/views/clients/index.jsp**

```
@ResponseBody  
@RequestMapping(value = { "/home"}, method = RequestMethod.GET)  
public String Index() {  
return "Nguyễn Hữu Trung";  
}
```

→ **Nguyễn Hữu Trung**

Trường hợp này rất hữu ích cho tương tác JSON, JavaScript, XML...

```
@RequestMapping(value = { "/home"}, method = RequestMethod.GET)  
public String Index() {  
return "redirect:/clients/index.htm"; → @RequestMapping("/clients/index")  
}
```

## LẬP TRÌNH WEB (WEBPR330479)

### Form trong Spring



THS. NGUYỄN HỮU TRUNG

- Ths. Nguyễn Hữu Trung
- Khoa Công Nghệ Thông Tin
- Trường Đại học Sư Phạm Kỹ Thuật TP.HCM
- 090.861.7108
- [trungnh@hcmute.edu.vn](mailto:trungnh@hcmute.edu.vn)
- <https://www.youtube.com/@baigiai>

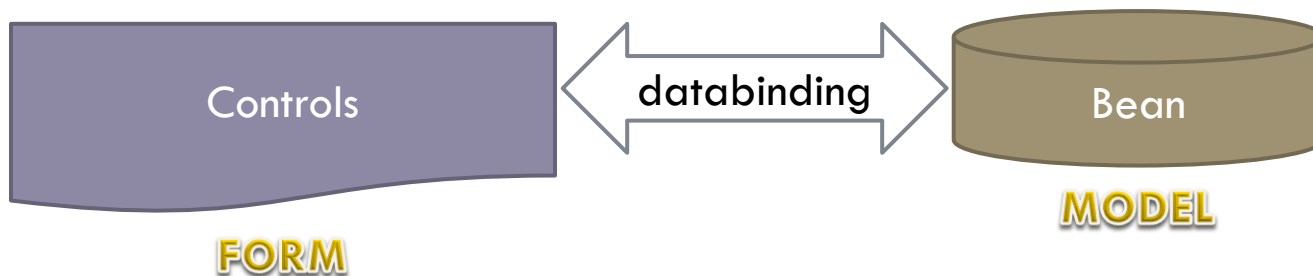


- ⦿ Hiểu cơ chế Databinding
- ⦿ Xây dựng form trong Spring
- ⦿ @ModelAttribute

# Giới thiệu Databinding?

95

- Databinding là sự kết nối dữ liệu của bean đặt trong model đến các điều khiển trên form.
- Khi thay đổi dữ liệu trong bean thì dữ liệu trên các điều khiển cũng thay đổi theo.
- Ràng buộc dữ liệu có thể là 1 chiều hoặc 2 chiều
  - ▣ Chiều lên: *chuyển dữ liệu từ các điều khiển vào các thuộc tính của bean*
  - ▣ Chiều về: *hiển thị dữ liệu từ các thuộc tính của bean lên các điều khiển của form*



# Databinding với các thẻ HTML?

96

- Bạn có thể buộc dữ liệu từ các thuộc tính của bean vào các điều khiển HTML bằng cách sử dụng biểu thức EL

```
<form action="login.htm" method="post">
  <div>User Name:</div>
  <input name="id" value="${user.id}">

  <div>Password:</div>
  <input name="password" value="${user.password}">

  <hr>
  <button>Login</button>
</form>
```



- Dù chúng ta hoàn toàn có thể buộc dữ liệu từ bean trong model lên form với EL nhưng gặp phải một số hạn chế sau:
  - ▣ Phải viết mã trên giao diện, dài dòng, khó quản lý
  - ▣ Đổ dữ liệu vào các List Control trở nên phức tạp và khó khăn
    - Combox
    - Listbox
    - Radiobuttons
    - Checkboxes
  - ▣ Kiểm và thông báo lỗi

- Spring MVC cung cấp thư viện thẻ giúp việc buộc dữ liệu từ bean vào các điều khiển trở nên dễ dàng hơn

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

- Sau khi khai báo thư viện thẻ ngay đầu trang JSP, chúng ta có thể tạo form và ràng buộc dữ liệu

```
<form:form action="login.htm" modelAttribute="user">
  <div>User Name:</div>
  <form:input path="id"/>
  <div>Password:</div>
  <form:input path="password"/>
  <hr>
  <button>Login</button>
</form:form>
```

Thẻ trong thư  
viện form

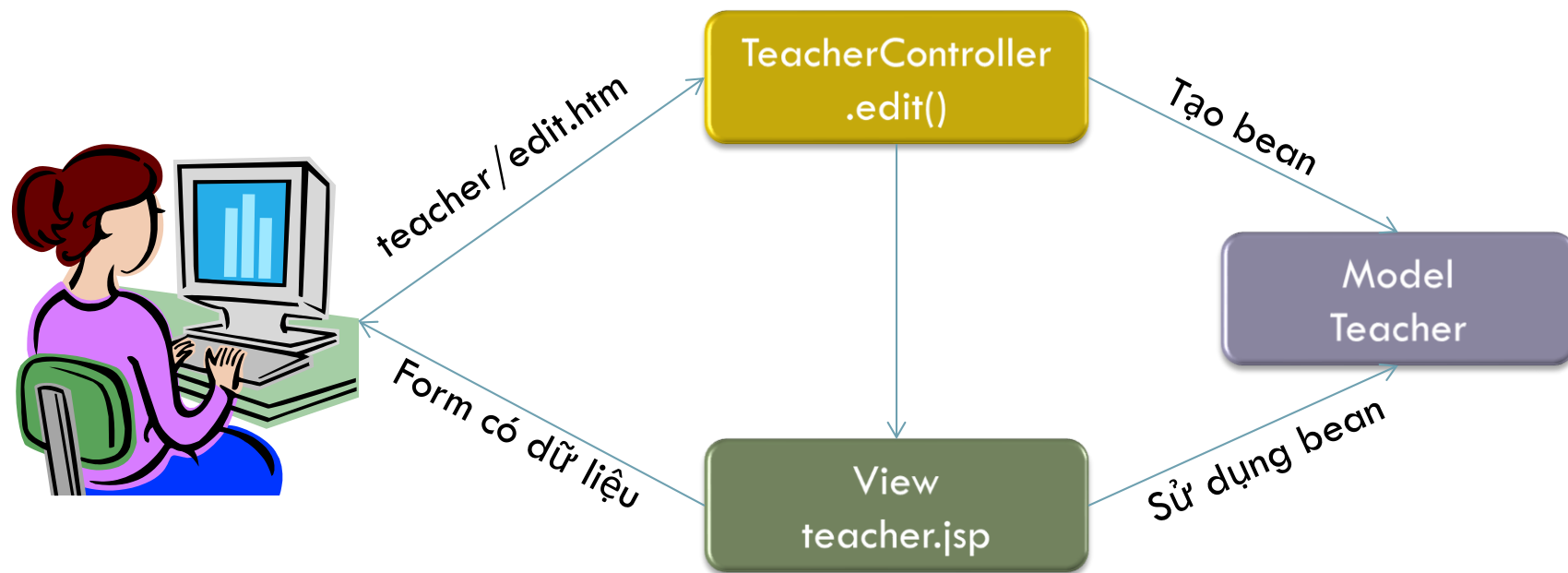
Tên của bean  
đặt trong model

Tên thuộc tính  
của bean user

# Ưu điểm của form Spring?

99

- Cung cấp cơ chế buộc dữ liệu lên các điều khiển
- Form đơn giản, rõ ràng, dễ hiểu
- Khi thay đổi dữ liệu trong bean thì dữ liệu trên các điều khiển cũng thay đổi theo.
- Cấp dữ liệu vào các List Control trở nên rất đơn giản
- Kiểm và hiển thị lỗi một cách dễ dàng



- Người sử dụng yêu cầu `teacher/edit.htm`
- Phương thức `edit()` tạo bean và đặt vào model
- View chứa form buộc dữ liệu từ bean trong model lên các điều khiển của form

```
package vn.iotstar.model;
public class Teacher {
    private String name;
    private int age;
    private String depart;

    public Teacher() {}
    public Teacher(String name, int age, String depart) {
        super();
        this.name = name;
        this.age = age;
        this.depart = depart;
    }

    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    public int getAge() {return age;}
    public void setAge(int age) {this.age = age;}
    public String getDepart() {return depart;}
    public void setDepart(String depart) {this.depart = depart;}
}
```

Trường chứa dữ  
liệu

Các constructor

Các phương thức  
getter/setter

```
@Controller
@RequestMapping("/teacher/")
public class TeacherController {
    @RequestMapping("edit")
    public String edit(ModelMap model) {
        Teacher gv = new Teacher("Nguyễn Hữu Trung", 40, "CNTT");
        model.addAttribute("Teacher", gv);
        return "clients/teacher";
    }
}
```

Trong model có  
bean Teacher

- Khi gọi **teacher/edit.htm** thì phương thức action **edit()** sẽ chạy. **edit()** tạo một đối tượng **gv** và đặt vào **model** với tên là **Teacher** để chuyển sang view **teacher.jsp**

# Thiết kế form có ràng buộc dữ liệu

103

- View teacher.jsp chứa form buộc các thuộc tính của bean vào các điều khiển

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

```
<frm:form action="../../teacher/update.htm" modelAttribute="Teacher">
<div>Họ tên</div>
<frm:input path="name"/>
<div>Tuổi</div>
<frm:input path="age"/>
<div>Đơn vị</div>
<frm:select path="depart">
<frm:option value="CNTT">Khoa CNTT</frm:option>
<frm:option value="CKM">Khoa CKM</frm:option>
</frm:select>
<hr>
<button>Cập nhật</button>
</frm:form>
```

**Các thuộc tính của bean**

Bean buộc dữ liệu  
lên các điều khiển

Các thuộc tính của bean TeacherModel được buộc với các điều khiển của form. Như vậy khi muốn thay đổi dữ liệu trên form bạn chỉ cần thay đổi bean trong model

- Form sẽ submit dữ liệu đến action “**update.htm**”. Bạn cần bổ sung phương thức action `update()` vào `TeacherController` để xử lý nút Update.

```
@RequestMapping("update")
public String update(@ModelAttribute("Teacher") Teacher teacher)
{
    return "clients/teacher";
}
```

- Dữ liệu form được chuyển vào các thuộc tính của đối số action `teacher`.
- **@ModelAttribute("Teacher")** sẽ bổ sung một attribute có tên là `Teacher` có giá trị là đối số `teacher` vào model. Attribute này sẽ buộc dữ liệu lên các điều khiển khi quay trở lại form



# Các điều khiển form của Spring

105

Điều khiển Spring	Sinh ra điều khiển HTML
<form:form>	<form>
<form:input/>	<input type="text"/>
<form:textarea/>	<textarea/>
<form:checkbox/>	<input type='checkbox'/>
<form:radiobutton/>	<input type='radio'/>
<form:hidden/>	<input type='hidden'/>
<form:password/>	<input type='password'/>
<form:button/>	<button/>
<form:select/>	<select/>
<form:radiobuttons/>	Nhóm radio
<form:checkboxes/>	Nhóm checkbox

Đây là các List Control cần được cấp dữ liệu từ Collection, Array hoặc Map

## Sửa thông tin

Họ tên

Tuổi

Đơn vị

**<form:select>**

## Sửa thông tin

Họ tên

Nguyễn Hữu Trung

Tuổi

42

Đơn vị

Khoa CNTT ▾

Cập nhật

Đổi từ nhập dữ liệu sang chọn mục trong ComboBox

- Để đạt được điều mong muốn trên thì chúng ta cần thay đổi
  - ▣ TeacherController: phải cung cấp dữ liệu dạng Array, Collection hoặc Map vào model
  - ▣ teacher.jsp: phải thay điều khiển và đổ dữ liệu vào

```
//đổ dữ liệu vào combobox  
@ModelAttribute("departs")  
public String[] getDeparts() {  
    String[] departs = {"Khoa CNTT", "Khoa  
CKM", "Khoa Điện-Điện tử"};  
    return departs;  
}  
String[]
```

```
<div>Đơn vị</div>  
<frm:select path="depart"  
items="${departs}">
```

## Sửa thông tin

Họ tên  
Nguyễn Hữu Trung

Tuổi  
42

Đơn vị  
Khoa CNTT  
Khoa CNTT  
Khoa CKM  
Khoa Điện-Điện tử

- Thay đổi TeacherController
  - ▣ Bổ sung phương thức getDeparts().
  - ▣ **@ModelAttribute("departs")** sẽ đặt kết quả của phương thức này vào trong Model với tên là **departs**. Dữ liệu này được sử dụng để đổ vào **ComboBox**
- Thay đổi view (teacher.jsp)
  - ▣ Thay `<frm:input path="depart">` bằng `<frm:select path="depart" items="{departs}">`.
  - ▣ Thuộc tính items chỉ ra dữ liệu (Collection, Map hay mảng) đặt trong Model để đổ vào ComboBox

- Trong Spring MVC @ModelAttribute được sử dụng để bổ sung attribute vào model trong 2 trường hợp:
  - @ModelAttribute(name) argument
    - Sẽ bổ sung attribute có tên là name và có giá trị là giá trị của đối số phương thức action
    - Tương đương: `model.addAttribute(name, argument)`
  - @ModelAttribute(name) method
    - Sẽ bổ sung attribute có tên là name và có giá trị là kết quả của phương thức
    - Tương đương: `model.addAttribute(name, method())`
- Trong view bạn có thể sử dụng nó như một attribute bình thường: buộc vào form, sử dụng EL, đổ vào ListControl

```
//đổ dữ liệu vào combobox  
@ModelAttribute("departs")  
public String[] getDeparts() {  
    List<String> departs = new ArrayList<>();  
    departs.add("Khoa CNTT");  
    departs.add("Khoa CKM");  
    departs.add("Khoa Điện - Điện tử");  
    return departs;  
}
```

**List<String>**

## Sửa thông tin

Họ tên  
Nguyễn Hữu Trung

Tuổi  
42

Đơn vị  
Khoa CNTT  
Khoa CKM  
Khoa Điện-Điện tử

```
<div>Đơn vị</div>  
<frm:select path="depart"  
items="${departs}">
```

```
//đổ dữ liệu vào combobox  
@ModelAttribute("departs")  
public String[] getDeparts() {  
    Map<String,String> departs = new HashMap<>();  
    departs.put("CNTT", "Khoa CNTT");  
    departs.put("CKM", "Khoa CKM");  
    departs.put("D-DT", "Khoa Điện - Điện tử");  
    return departs;  
}
```

**Map<String,String>**

Sửa thông tin

Họ tên  
Nguyễn Hữu Trung  
Tuổi  
42  
Đơn vị  
Khoa CNTT  
Khoa CKM  
Khoa Điện-Điện tử

```
<div>Đơn vị</div>  
<frm:select path="depart"  
items="${departs}">
```



//đổ dữ liệu vào combobox

@ModelAttribute("departs")

```
public List<Department> getDeparts() {  
    List<Department> departs = new ArrayList<>();  
    departs.add(new Department("CNTT", "Khoa CNTT"));  
    departs.add(new Department("CKM", "Khoa CKM"));  
    departs.add(new Department("D-DT", "Khoa Điện - Điện tử"));  
    return departs;  
}
```

**List<Department>**

```
public class Department {  
    private String dID;  
    private String dname;  
    public Department() {}  
    public Department(String dID, String dname) {  
        super();  
        this.dID = dID;  
        this.dname = dname;  
    }  
    public String getdID() {return dID;}  
    public void setdID(String dID) {this.dID = dID;}  
    public String getDname() {return dname;}  
    public void setDname(String dname) {this.dname = dname;}  
}
```

```
<frm:select path="depart"  
    items="${departs}"  
    itemLabel = "dname" itemvalue="dID">  
</frm:select>
```

## Sửa thông tin

Họ tên

Tuổi

Đơn vị

- `<frm:select`                      `path="property"`                      **`items="{items}"`**  
**`itemValue="prop1" itemLabel="prop2">`**
  - ▣ `items`: chỉ ra tập dữ liệu đổ vào ComboBox
  - ▣ `itemValue` và `itemLabel` chỉ được sử dụng khi tập `items` là `Collection<Bean>`
    - `itemValue`: chỉ ra tên thuộc tính để làm giá trị
    - `itemLabel`: chỉ ra tên thuộc tính để làm nhãn (nhìn thấy)
- List Control khác có cùng cú pháp với `select`
  - ▣ `<frm:radiobuttons`                      `path="property"`                      **`items="{items}"`**  
**`itemValue="prop1" itemLabel="prop2">`**
  - ▣ `<frm:checkboxes`                      `path="property"`                      **`items="{items}"`**  
**`itemValue="prop1" itemLabel="prop2">`**
- Đổ dữ liệu vào các List Control là như nhau

- Thẻ Spring `<frm:tag>` có một số thuộc tính thường dùng sau:
  - ▣ `cssClass`: thay cho thuộc tính class trong HTML
  - ▣ `disabled`: thay cho thuộc tính disabled trong HTML
  - ▣ `readonly`: thay cho thuộc tính readonly trong HTML
  - ▣ `cssErrorClass`: cho ra class định dạng thông báo lỗi
- Ví dụ:
  - ▣ `<frm:input path="id" readonly="true"/>`
  - ▣ `<frm:input path="name" cssClass="form-control">`