Spring mvc

Spring mvc là một phần hoặc module của spring framework (có thể gọi là framework web ) . spring framework là hệ thống linh hoạt có nhiều module và nhiều dự án con, mỗi module có mục tiêu cụ thể.

Spring mvc là một trong những module đó có trách nhiệm phát triển úng dụng web theo kiến trúc Model-View-Controller.

Ngoài Spring mvc thì Spirng framework gồm nhiều module khác như Spring core,Spring data,Spring AOP…..

Kiến trúc của Spring mc :  
Model :

-Tầng này đại diện cho dữ liệu và logic xử lí dữ liệu trong ứng dụng.

-Nó là nới lưu trữ thông tin, thực hiện thao tác truy cập dữ liệu, và định nghĩa các quy tắc logic .

Ví dụ :

public class Book {

private String title;

private String author;

// Các getters và setters

}

View :

-Giao diện(view ) đại diện cho phần hiển thị của ứng dụng .

-Nó chịu trách nhiệu hiển thị dữ liệu của mô hình và cung cấp giao diện ngươi dùng để tương tác

-View không chứa logic xử lý

Ví dụ :

<!DOCTYPE html>

<html>

<head>

<title>Danh sách Sách</title>

</head>

<body>

<h1>Danh sách Sách</h1>

<ul>

<!-- Danh sách sách sẽ được hiển thị ở đây -->

</ul>

</body>

</html>

Controller :

-Bộ điều khiển chịu trách nhiệm xử lý yêu cầu từ người dùng và tương tác với mô hình và giao diện để xử lý logic

- Nó là trung tâm quyết định và định hình luồng điều khiển của úng dụng.

-Nhận thông tin từ giao diện, xử lý nó (thường gọi các method của mô hình ) và sao đó chọn view phù hợp để hiển thị kết quả

Ví dụ :

public class BookController {

private List<Book> books; // Danh sách sách

public BookController(List<Book> books) {

this.books = books;

}

public List<Book> getAllBooks() {

return books;

}

}

public class Main {

public static void main(String[] args) {

// Khởi tạo danh sách sách

List<Book> books = new ArrayList<>();

books.add(new Book("Clean Code", "Robert C. Martin"));

books.add(new Book("Design Patterns", "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides"));

// Khởi tạo Bộ điều khiển

BookController controller = new BookController(books);

// Nhận yêu cầu từ Giao diện và hiển thị danh sách sách

List<Book> displayedBooks = controller.getAllBooks();

// Hiển thị danh sách sách trong Giao diện

for (Book book : displayedBooks) {

System.out.println(book.getTitle() + " by " + book.getAuthor());

}

}

}

Luồng chạy chi tiết trong kiến trúc MVC :3

1. Người dùng tương tác với giao diện View và bấm nút đăng nhập trên trang web gửi yêu cầu đăng nhập đến máy chủ hành động là http Request đến disparcherServlet

Giao diện view chịu trách nhiệm thu thấp thông tin đăng nhập từ người dùng, chẳng hạn như tên đăng nhập và mật khẩu.

2. Controller xử lý yêu cầu :

Bộ điều hiển nhận yêu cầu đăng nhập từ giao diện ()

Nó xử lý yêu cầu bằng cách gọi một phương thức trong service hoặc thành phần xử lý logic đăng nhập

Bộ điều khiển có thể gọi các phương thức trên các đối tượng model để thực hiện logic xử lí, chẳng hạn như kiểm tra tên đăng nhập

3.Model :

Mô hình model chứa các đối tượng logic liên quan đến dữ liệu và xử lí dữ liệu.

Trong trường hợp đăng nhập, Mô hình có thể bao gồm đối tượng user để đại diện cho thông tin người dùng

Mô hình thực hiện kiểm tra đăng nhập bằng cách so sánh thông tin đăng nhập được cấp cho người dùng trong database

4.Controller

Bộ điều khiển nhận kết quả từ mô hình.

Nếu đăng nhập thành công, nó có thể hành động các hành động tiếp theo, chẳng hanhjh như chuyển hướng người dùng đến trang chính của úng dụng hoặc cung cấp dữ liệu người dùng đăng nhập cho giao diện hiển thị

Nếu đăng nhập không thành công, Bộ điều khiển có thể chuyển hướng người dùng đến trang đăng nhập lại hoặc hiển thị thông báo lỗi.

5.View :

Giao diện nhận kết quả từ bộ điều khiển và hiển thị thông tin tương ứng cho người dùng, chẳng hạn nhưu trang chính hoặc thông báo lỗi

Một ví dụ về nhiệu vụ của 3 MVC

Để minh họa xử lý logic của Model trong Spring MVC, hãy xem xét một ví dụ đơn giản về việc xác thực người dùng trong quá trình đăng nhập. Trong ví dụ này, chúng ta sẽ có một `UserService` để thực hiện xác thực và một `User` đại diện cho thông tin người dùng:

### 1. \*\*Đối tượng User (Model):\*\*

```java

public class User {

private String username;

private String password;

// Getters và setters

}

```

### 2. \*\*UserService (Model):\*\*

```java

public class UserService {

private List<User> users; // Giả sử dữ liệu người dùng được lưu trữ trong danh sách

public UserService(List<User> users) {

this.users = users;

}

public User authenticateUser(String username, String password) {

for (User user : users) {

if (user.getUsername().equals(username) && user.getPassword().equals(password)) {

return user; // Trả về đối tượng User nếu xác thực thành công

}

}

return null; // Trả về null nếu xác thực không thành công

}

}

```

### 3. \*\*Controller (Bộ điều khiển):\*\*

```java

@Controller

public class LoginController {

@Autowired

private UserService userService; // Sử dụng Dependency Injection để có thể sử dụng UserService

@RequestMapping("/login")

public String login(@RequestParam String username, @RequestParam String password, Model model) {

// Gọi phương thức authenticateUser từ UserService để xác thực người dùng

User authenticatedUser = userService.authenticateUser(username, password);

if (authenticatedUser != null) {

// Nếu xác thực thành công, chuyển hướng đến trang chính và truyền thông tin người dùng

model.addAttribute("user", authenticatedUser);

return "home";

} else {

// Nếu xác thực không thành công, hiển thị trang đăng nhập lại với thông báo lỗi

model.addAttribute("error", "Invalid username or password");

return "login";

}

}

}

```

Trong ví dụ này:

- `User` là đối tượng Model đại diện cho thông tin người dùng.

- `UserService` là một thành phần Model chịu trách nhiệm xử lý logic đăng nhập và kiểm tra xác thực.

- `LoginController` là một Bộ điều khiển, chịu trách nhiệm xử lý yêu cầu đăng nhập từ người dùng. Nó sử dụng `UserService` để xác thực người dùng và thực hiện các hành động tương ứng.

Các tầng Model và Controller được tách biệt, giúp mã nguồn dễ bảo trì và tái sử dụng.

Tại sao chúng ta nên cấu hình config bằng file java mà không cấu hình config bằng file xml

Cấu hình Java (Java Config) trong Spring MVC là một cách để cấu hình ứng dụng Spring bằng cách sử dụng mã Java thay vì sử dụng cấu hình XML truyền thống. Có một số lý do mà bạn có thể muốn sử dụng cấu hình Java trong Spring MVC:

1. \*\*Sử dụng mã Java để cấu hình\*\*: Cấu hình Java cho phép bạn sử dụng mã Java thay vì XML để định nghĩa các bean và cấu hình ứng dụng. Điều này có thể làm cho mã của bạn trở nên rõ ràng hơn và dễ đọc hơn so với XML, đặc biệt là đối với các ứng dụng lớn và phức tạp.

2. \*\*Kiểm soát linh hoạt\*\*: Cấu hình Java cho phép bạn sử dụng các tính năng của ngôn ngữ Java như biến, điều kiện, vòng lặp, và các thư viện mã nguồn mở để cấu hình ứng dụng. Điều này giúp bạn kiểm soát ứng dụng một cách linh hoạt và thực tomhiện các tùy chỉnh phức tạp hơn.

3. \*\*Kiểm soát qua mã nguồn\*\*: Cấu hình Java giúp bạn kiểm soát toàn bộ ứng dụng qua mã nguồn, điều này có thể hữu ích trong việc quản lý mã nguồn và tránh phải thao tác trên các tệp XML hoặc tệp cấu hình ngoài.

4. \*\*Kiểm soát qua các IDE\*\*: Cấu hình Java thường được hỗ trợ tốt hơn trong các môi trường phát triển tích hợp (IDE) như IntelliJ IDEA, Eclipse, và NetBeans. Điều này giúp bạn dễ dàng tìm kiếm, xem và chỉnh sửa cấu hình ứng dụng trong môi trường phát triển.

5. \*\*Kiểm soát hơn về lỗi tại thời điểm biên dịch\*\*: Sử dụng cấu hình Java có thể giúp bạn phát hiện lỗi trong quá trình biên dịch mã nguồn của bạn, chứ không phải chờ đến thời gian chạy ứng dụng để phát hiện lỗi trong tệp XML hoặc các cấu hình khác.

Tuy nhiên, quyết định sử dụng cấu hình Java hoặc XML trong Spring MVC phụ thuộc vào sự ưa thích cá nhân và cách bạn hoặc nhóm của bạn quản lý dự án. Cả hai phương pháp đều có thể sử dụng để xây dựng các ứng dụng Spring MVC chất lượng và có tính linh hoạt cao.

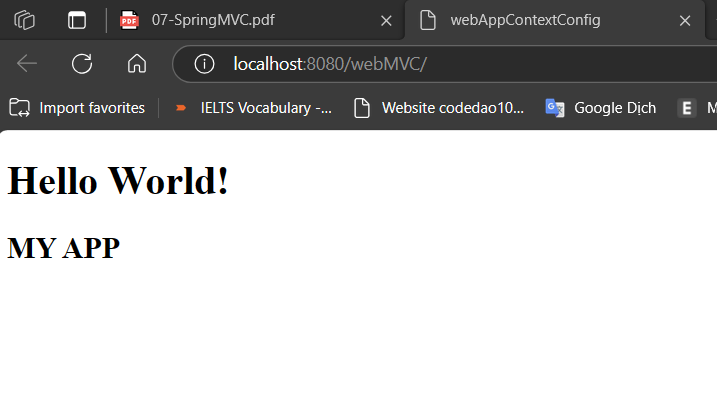
Cấu hình để cho tomcat đọc được

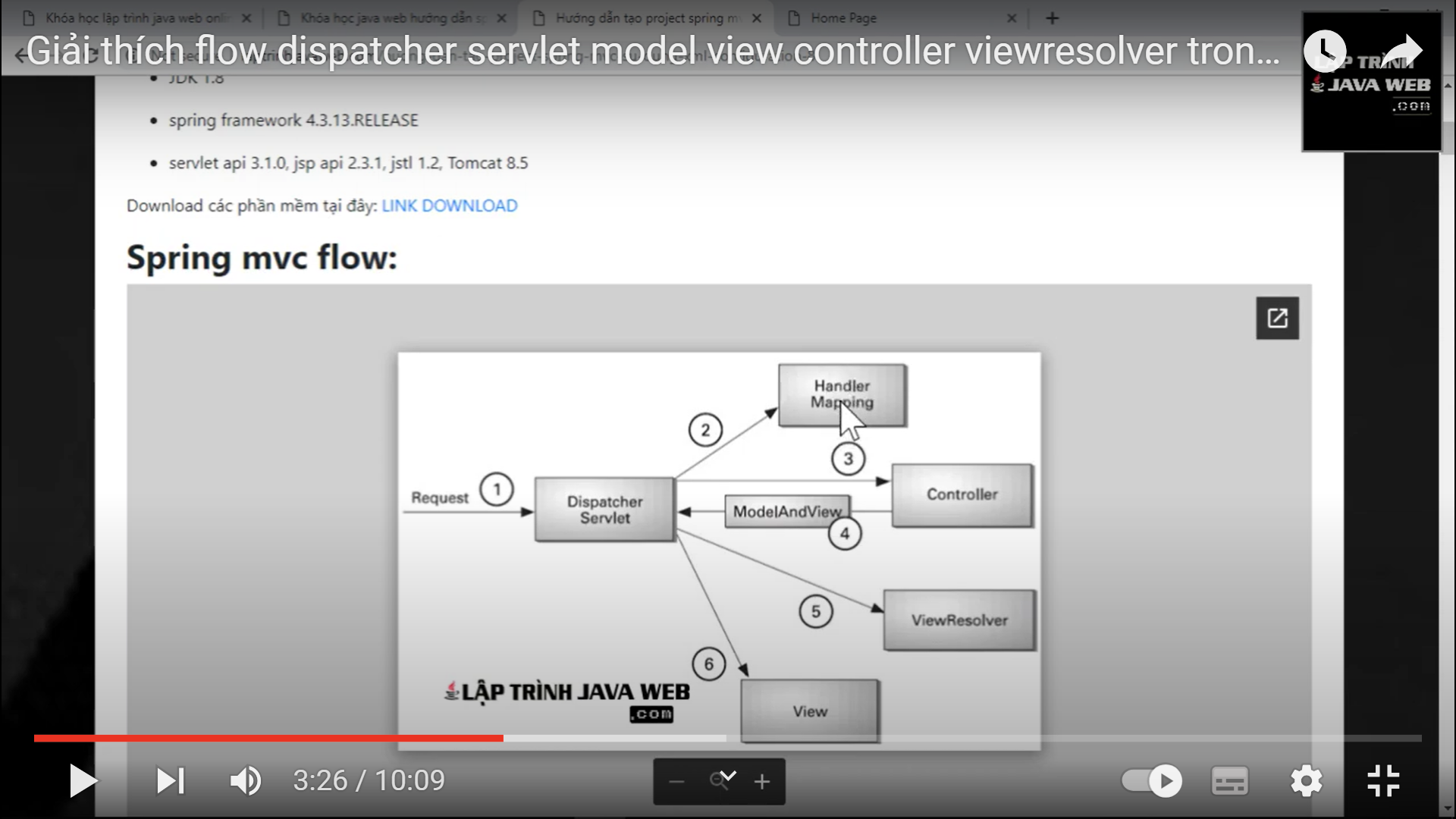
Tại sao trong kênh laptrinhjavaweb lại cấu hình config bằng file xml mà không cấu hình bằng file java

Giải thích dispathcherservlet

Ví dụ khi truy cập vào controller có chứa trang chủ và nó sẽ request trang chủ đến dispathservlet và dispathservlet sẽ đi tìm các handler mapping tức là những requestMapping : getMapping, postMapping,PutMapping…. Và handler sẽ vào đúng package controller để và nó sẽ tìm những class trong gói bắng đầu bằng @Controller và nếu tìm thấy giống controller request đên thì kế tìm chúng sẽ tìm requestMapping hoặc putMapping PostMapping có dấu “/” giống như hình. Sau khi đã xác định đươc controller nào thì xem nó trả về một cái tên file là index.jsp lúc đó trả modelandview cho servlet

Nhờ dấu / mà vào controller và vào getmapping(“/”) mà phân giải được index và xuất ra hello world





Tại sao chúng ta nên dùng jsp thay cho html trong spring MVC

Trong Spring MVC, JSP (JavaServer Pages) thường được sử dụng để xây dựng giao diện người dùng của ứng dụng web. Dưới đây là một số lý do tại sao chúng ta thường sử dụng JSP trong Spring MVC:

1. \*\*Tích hợp dễ dàng với Spring\*\*: Spring Framework đã được thiết kế để tích hợp chặt chẽ với JSP. Điều này có nghĩa rằng việc sử dụng JSP trong Spring MVC thường rất dễ dàng và không đòi hỏi nhiều cấu hình phức tạp.

2. \*\*Cú pháp gần giống HTML\*\*: JSP sử dụng cú pháp gần giống với HTML, điều này giúp cho các nhà phát triển web quen thuộc với việc viết mã HTML có thể nhanh chóng chuyển sang sử dụng JSP.

3. \*\*Hỗ trợ sử dụng biến Java\*\*: JSP cho phép sử dụng biến Java và các biểu thức Java bên trong mã HTML. Điều này giúp trong việc truy cập và xử lý dữ liệu từ các đối tượng Java được truyền từ Spring Controller đến JSP.

4. \*\*Thư viện Tag hữu ích\*\*: JSP cung cấp một loạt các thẻ (tags) tiện ích như JSTL (JSP Standard Tag Library) và Spring Tag Library, giúp làm cho việc hiển thị và xử lý dữ liệu trở nên dễ dàng hơn.

5. \*\*Hỗ trợ quản lý phiên\*\*: JSP cho phép quản lý phiên (session management) dễ dàng, điều này quan trọng khi bạn cần theo dõi trạng thái của người dùng trong các ứng dụng web.

6. \*\*Khả năng tạo các tệp view động\*\*: JSP có thể được sử dụng để tạo các trang view động, tức là các trang web có nội dung thay đổi dựa trên dữ liệu được truyền từ Spring Controller.

Tuy nhiên, cũng cần lưu ý rằng trong các ứng dụng Spring MVC hiện đại, có nhiều sự lựa chọn khác cho việc tạo giao diện người dùng, như sử dụng Thymeleaf, FreeMarker, hoặc React/Angular/Vue.js cho phần giao diện người dùng phía máy khách. Sự lựa chọn của công nghệ giao diện phụ thuộc vào yêu cầu cụ thể của dự án và sở thích của nhóm phát triển.

TomCat hoạt động như thế nào ?

Tomcat là một máy chủ ứng dụng web (Web Application Server) phổ biến, và nó thường được sử dụng để triển khai ứng dụng Spring MVC. Dưới đây là cách Tomcat hoạt động trong mô hình ứng dụng Spring MVC:

1. \*\*Triển khai ứng dụng\*\*: Đầu tiên, bạn phải triển khai ứng dụng Spring MVC của mình. Điều này bao gồm việc đóng gói ứng dụng thành một tệp WAR (Web Application Archive) và sao chép nó vào thư mục webapps của máy chủ Tomcat.

2. \*\*Khởi động Tomcat\*\*: Bạn khởi động máy chủ Tomcat bằng cách chạy tệp thực thi hoặc sử dụng các công cụ quản lý như Apache Tomcat Manager. Khi Tomcat khởi động, nó sẽ tìm và triển khai các ứng dụng có sẵn trong thư mục webapps.

3. \*\*Quản lý yêu cầu và phản hồi\*\*: Khi một yêu cầu HTTP đến từ một trình duyệt hoặc ứng dụng khách hàng, Tomcat sẽ nhận yêu cầu này và xác định ứng dụng Spring MVC nào phải xử lý nó dựa trên URL và cấu hình.

4. \*\*Dispatcher Servlet\*\*: Trong ứng dụng Spring MVC, có một Servlet quan trọng được gọi là `DispatcherServlet`. DispatcherServlet là thành phần trung tâm trong xử lý yêu cầu và phản hồi. Khi một yêu cầu đến, DispatcherServlet sẽ tiếp nhận nó và xác định Controller phù hợp để xử lý yêu cầu dựa trên cấu hình của ứng dụng và URL.

5. \*\*Controller\*\*: Controller là một thành phần quan trọng trong Spring MVC. Nó xử lý yêu cầu, thực hiện logic kinh doanh và chuyển dữ liệu cho View.

6. \*\*View Resolver\*\*: View Resolver là một phần của Spring MVC được sử dụng để tìm và chọn View phù hợp để hiển thị dữ liệu cho người dùng. Thường, View Resolver sẽ tìm các tệp JSP hoặc Thymeleaf templates để tạo trang HTML cho người dùng.

7. \*\*Hiển thị kết quả\*\*: View sẽ được hiển thị dưới dạng HTML và gửi lại cho máy chủ Tomcat. Tomcat sau đó gửi phản hồi HTML này đến trình duyệt của người dùng.

8. \*\*Kết thúc xử lý\*\*: Sau khi phản hồi đã được gửi, Tomcat sẽ kết thúc xử lý cho yêu cầu và sẵn sàng xử lý các yêu cầu mới từ trình duyệt hoặc ứng dụng khách hàng khác.

Như vậy, Tomcat hoạt động như một máy chủ ứng dụng web chịu trách nhiệm quản lý việc xử lý yêu cầu và phản hồi trong ứng dụng Spring MVC, cung cấp môi trường để triển khai và chạy ứng dụng web Spring MVC của bạn. deploy file war lên server Tomcat. File war này được đóng gói theo cơ chế maven

Giải thích DI :

Khi lớp A phụ thuộc vào B

Thì B là dependence của A

B thay đổi thì A cũng thay đổi theo

Nguyên tắc của DI là không cho một lớp phụ thuộc vào một lớp thường mà phải phụ thuộc vào một lớp trừu tượng.

Dependence injection (tiêm sự phụ thuộc) : để cho một lớp phụ thuôc vào một lớp bậc cao để dễ sửa mở rộng và tùy biến hơn.

Ví dụ một lớp xe máy có động cơ trung quốc đang chạy trong hàm main thì bỗng dưng nó bị hư thì cách khắc phục là tạo thêm một interface động cơ tạo thêm lớp động cơ mỹ hay đức gì đó … imlement lại interface dongco thì vẫn chạy bình thường lúc này đã giảm được sự phụ thuộc vào trung quốc và bây giờ nếu muốn chúng ta muốn xe chạy động cơ gì cứ tạo ra rồi tiêm vào xe máy ở hàm main.

Như vầy gọi là contructor injection :

public class OrderService {

private PaymentProcessor paymentProcessor;

// Constructor injection

public OrderService(PaymentProcessor paymentProcessor) {

this.paymentProcessor = paymentProcessor;

}

youtube.com/watch?v=HNnUUOUnUfo&t=42s

Giải thich Bean trong spring mvc

Bean là một đối tương trong java được spring ioc container quản lý

Có 2 cách gắn bean vào trong biến @autowired và inject sau đó spring sẽ giúp ta tìm bean tương ứng

Bean sẽ phân giải file .jsp r.setViewClass(JstlView.class); đang dùng một đối tượng của lớp InternalResourceViewResolver để sử dụng lớp jstlView để xử lý cụ thể các thẻ jsp cho các trang web

JstlView là một lớp xử lý view cụ thể trong Spring, được sử dụng khi bạn muốn sử dụng thư viện JSTL (JavaServer Pages Standard Tag Library) để xử lý các thẻ JSP trong trang web.

r.setPrefix("/WEB-INF/pages/"); set đường dường

r.setSuffix(".jsp"); spring sẽ tìm tệp .jsp này và trả nó về một view

Dòng return r; có ý nghĩa rằng phương thức internalResourceViewResolver trả về đối tượng InternalResourceViewResolver (biến r) đã được cấu hình. Điều này cho phép Spring Container quản lý và sử dụng bean này trong toàn bộ ứng dụng.

@Override

protected Class<?>[] getServletConfigClasses() {

return new Class[]{

WebAppContextConfig.class //

};

}

Khi mã của bạn được chạy, mối quan hệ giữa hai lớp `DispatcherServletlnit` và `WebAppContextConfig` xuất hiện trong quá trình cấu hình và khởi tạo ứng dụng web Spring. Dưới đây là cách mối quan hệ này hoạt động khi ứng dụng được khởi động:

1. \*\*DispatcherServletlnit\*\*:

- `DispatcherServletlnit` là một lớp kế thừa từ `AbstractAnnotationConfigDispatcherServletInitializer`, và nhiệm vụ chính của nó là cấu hình và khởi tạo Servlet Dispatcher (DispatcherServlet).

- Trong phương thức `getServletConfigClasses()`, nó chỉ định rằng cấu hình của DispatcherServlet sẽ được lấy từ lớp `WebAppContextConfig`. Điều này có nghĩa là DispatcherServlet sẽ sử dụng cấu hình được định nghĩa trong `WebAppContextConfig` để xử lý các yêu cầu web và hiển thị các trang web.

- Đồng thời, nó định nghĩa cấu hình cho `ServletMappings`, trong trường hợp này là `"/"`, điều này có nghĩa là DispatcherServlet sẽ xử lý tất cả các yêu cầu đến root của ứng dụng.

2. \*\*WebAppContextConfig\*\*:

- `WebAppContextConfig` là một lớp cấu hình chính cho ứng dụng web Spring.

- Nó được đánh dấu bằng `@Configuration` để cho biết rằng nó chứa cấu hình Spring.

- Bằng cách sử dụng `@EnableWebMvc`, nó kích hoạt cấu hình Spring MVC cho ứng dụng.

- Trong phương thức `configureDefaultServletHandling()`, nó cho phép quyền truy cập các tệp tĩnh bằng cách cấu hình `DefaultServletHandlerConfigurer

- Phương thức `internalResourceViewResolver()` định nghĩa một bean `InternalResourceViewResolver`, cấu hình cách xử lý view (trang web) trong ứng dụng. Bean này được sử dụng bởi DispatcherServlet để phân giải các trang JSP và hiển thị chúng.

Khi ứng dụng được khởi động, Spring Container sẽ quản lý và kết nối các cấu hình và bean này một cách tự động. DispatcherServlet sẽ sử dụng cấu hình và bean từ `WebAppContextConfig` để xử lý các yêu cầu và hiển thị các trang web trong ứng dụng của bạn.

Nhờ

Giải thích @EnableTransactionManagement

Giao tác trong Spring: Trong mô hình ứng dụng Spring, giao tác là một quá trình thực hiện một loạt các thao tác cơ sở dữ liệu (ví dụ: INSERT, UPDATE, DELETE) một cách an toàn. Nếu tất cả các thao tác cơ sở dữ liệu thành công, giao tác được commit (lưu trữ), ngược lại, nếu có bất kỳ lỗi nào xảy ra, giao tác sẽ được rollback (hủy bỏ).

@EnableTransactionManagement: Annotation này được đặt trên một lớp cấu hình (configuration class) trong ứng dụng Spring. Nó có nhiệm vụ bật và cấu hình quản lý giao tác cho ứng dụng. Khi bạn sử dụng @EnableTransactionManagement, Spring sẽ tự động cấu hình một PlatformTransactionManager để quản lý giao tác.

Giải thích ORM trong spring mvc

Giúp tạo ra những bảng trong csdl từ các class như sinh viên và nhân viên …Truy vấn dữ liệu thông qua các logic hướng đối tượng, không trực tiếp viết các truy vấn SQL.

Vd : có một class SV trong gói pojo và sử dụng ORM để tạo ra một bảng trong mysql mà không cần viết code sql để truy vấn. Orm truy vấn nhanh hơn.

HQL : truy vấn dựa trên đối tượng

Vd : Session s = factory.getObject().getCurrentSession();

Query q = s.createQuery("FROM Product");

model.addAttribute("products", q.getResultList());