Contents

[**Lý thuyết** 7](#_Toc137565297)

[**I.** **Servlet** 7](#_Toc137565298)

[**1.** **Tìm hiểu về công cụ và cách tổ chức chương trình trong Java Servlet** 7](#_Toc137565299)

[- **Khái niệm:** 7](#_Toc137565300)

[- **Kiến trúc:** 7](#_Toc137565301)

[- **Nhiệm vụ của hoạt động:** 7](#_Toc137565302)

[- **Công cụ sử dụng** 8](#_Toc137565303)

[- **Môi trường làm việc của Servlet** 8](#_Toc137565304)

[- **Servlet Container** 8](#_Toc137565305)

[- **API Servlet có bốn package Java:** 8](#_Toc137565306)

[- **Vòng đời của Servlet** 9](#_Toc137565307)

[- **Phương thức init()** 9](#_Toc137565308)

[- **Phương thức service()** 9](#_Toc137565309)

[- **Phương thức doGet()** 10](#_Toc137565310)

[- **Phương thức doPost()** 10](#_Toc137565311)

[- **Phương thức destroy()** 10](#_Toc137565312)

[- **Cấu trúc thư mục** 11](#_Toc137565313)

[- **Cách Servlet container và web server xử lý một request** 11](#_Toc137565314)

[- **Mô hình MVC:** 11](#_Toc137565315)

[**2.** **Tìm hiểu về JSP, EL và JSTL** 12](#_Toc137565316)

[**a)** **JSP** 12](#_Toc137565317)

[**b)** **Jsp standard tag library** 14](#_Toc137565318)

[**c)** **Khái niệm** 14](#_Toc137565319)

[**d)** **JSP Expression Language (EL)** 14](#_Toc137565320)

[e) **JavaServer Pages Standard Tag Library (JSTL)** 17](#_Toc137565321)

[**3.** **Tìm hiểu về lập trình JPA (Java Persistence API)** 19](#_Toc137565322)

[**a)** **Khái niệm JPA** 19](#_Toc137565323)

[**b)** **Object Relational Mapping (ORM)** 19](#_Toc137565324)

[**c)**  **Kiến trúc JPA** 20](#_Toc137565325)

[**d) Lợi ích của JPA** 22](#_Toc137565326)

[**e)** **Tại sao nên dùng JPA** 22](#_Toc137565327)

[**f)** **Các tính năng của JPA** 23](#_Toc137565328)

[**4. Tìm hiểu về Scope, Listener và Filter trong Java Web** 26](#_Toc137565329)

[- a) **Application Context/Scope** 26](#_Toc137565330)

[- b) **RequestScope** 26](#_Toc137565331)

[- c) **Session scope:** 26](#_Toc137565332)

[- d) **JSP page scope** 26](#_Toc137565333)

[- e) **Filter:** 27](#_Toc137565334)

[- f) **Sử dụng filter khi:** 27](#_Toc137565335)

[- g) **Các phương thức servlet filter** 27](#_Toc137565336)

[**5. Tìm hiểu về Layout và Đa ngữ (Internationalization - 118N)** 28](#_Toc137565337)

[**a)Đa ngữ (Internationalization – I18N)** 28](#_Toc137565338)

[**b)Bố cục trang Web bằng Thymeleaf Layout** 30](#_Toc137565339)

[- **Giới thiệu về Thymeleaf Layout:** 30](#_Toc137565340)

[- **Tính năng:** 30](#_Toc137565341)

[- **Ưu điểm:** 30](#_Toc137565342)

[- **Các khái niệm cơ bản của Thymeleaf Layout:** 31](#_Toc137565343)

[- **Khuôn mẫu(Thymeleaf Layout):** 31](#_Toc137565344)

[- **Các biểu thức đơn giản:** 31](#_Toc137565345)

[- **Sử dụng Thymeleaf Layout để tạo bố cục web** 31](#_Toc137565346)

[- **Định nghĩa khuôn mẫu** 31](#_Toc137565347)

[- **Định nghĩa mẫu con** 31](#_Toc137565348)

[- **Chèn các mẫu con vào các khuôn mẫu** 31](#_Toc137565349)

[- **Sử dụng thẻ Thymeleaf để truyền dữ liệu** 32](#_Toc137565350)

[- **Ưu điểm** 32](#_Toc137565351)

[- **Nhược điểm** 32](#_Toc137565352)

[**II. Spring Boot** 32](#_Toc137565353)

[**1.Tổng quan Spring Boot** 32](#_Toc137565354)

[**a)Spring Framework** 32](#_Toc137565355)

[**b)Spring MVC** 33](#_Toc137565356)

[**c)Spring Boot** 33](#_Toc137565357)

[**d)Spring Boot vs Spring Framework** 33](#_Toc137565358)

[**e)Spring Boot vs Spring MVC** 34](#_Toc137565359)

[**f)Môi trường & công cụ** 34](#_Toc137565360)

[**g)Tạo dự án Spring Boot** 34](#_Toc137565361)

[**h)Cấu trúc dự án sau khi tạo** 35](#_Toc137565362)

[**i)Tổ chức chương trình** 37](#_Toc137565363)

[**j)Annotations** 38](#_Toc137565364)

[**k)Cách chia sẻ dữ liệu giữa controller và view** 40](#_Toc137565365)

[**l)Thymeleaf Expression** 40](#_Toc137565366)

[**m)Lombok (Entity/Model)** 42](#_Toc137565367)

[**n)Data Validation** 43](#_Toc137565368)

[**o)Spring Boot Web API** 44](#_Toc137565369)

[**2. Khái niệm IOC, DI (Dependency Injection) và Spring Bean** 45](#_Toc137565370)

[**2.1. IOC:** 45](#_Toc137565371)

[**a) Khái niệm** 45](#_Toc137565372)

[**b)Thiết kế truyền thống - tham chiếu trực tiếp đến Dependency** 45](#_Toc137565373)

[**c)Thiết kế theo cách đảo ngược phụ thuộc Inverse Dependency** 45](#_Toc137565374)

[**2.2. DI (Dependency Injection)** 45](#_Toc137565375)

[**a)Khái niệm** 45](#_Toc137565376)

[**b)Các loại DI** 46](#_Toc137565377)

[**2.3. Spring Bean** 46](#_Toc137565378)

[**a)Khái niệm** 46](#_Toc137565379)

[**b)Các bean scope trong spring** 47](#_Toc137565380)

[**c)Vòng đời của Bean** 47](#_Toc137565381)

[**d)Làm sao để cung cấp configuration metadata cho Spring Container?** 47](#_Toc137565382)

[**e)Làm sao để định nghĩa Scope của Bean** 48](#_Toc137565383)

[**f)Bean autoWired là gì?** 48](#_Toc137565384)

[**g)Các mode auto wiring** 48](#_Toc137565385)

[**3. Spring Data JPA cơ bản và nâng cao** 48](#_Toc137565386)

[**a)** **Kiến thức cơ bản của JPA** 49](#_Toc137565387)

[**JPA là gì** 49](#_Toc137565388)

[**ORM là gì** 49](#_Toc137565389)

[**Lợi ích của JPA** 49](#_Toc137565390)

[**Tại sao nên dùng JPA** 49](#_Toc137565391)

[**Kiến trúc của JPA** 50](#_Toc137565392)

[**Entity**: 50](#_Toc137565393)

[**Primary Key:** 50](#_Toc137565394)

[**Foreign Key:** 50](#_Toc137565395)

[**Query**: 51](#_Toc137565396)

[**NativeQuery** 51](#_Toc137565397)

[**Criteria API Query** 51](#_Toc137565398)

[**b)** **Các bước để sử dụng JPA** 52](#_Toc137565399)

[c) Các tính năng nâng cao 56](#_Toc137565400)

[**4.Schedule Task, Interceptor và gửi mail** 58](#_Toc137565401)

[**Phần Interceptor:** 58](#_Toc137565402)

[**So sánh Interceptor và Filter** 58](#_Toc137565403)

[**Phần gửi email:** 59](#_Toc137565404)

[**Mime** 60](#_Toc137565405)

[**Cấu hình trong application.property** 60](#_Toc137565406)

[**Các hình thức gửi mail** 61](#_Toc137565407)

[**5. Lambda Expression trong Java** 61](#_Toc137565408)

[**a) Lambda Expression ( Biểu thức Lambda)** 61](#_Toc137565409)

[**b) Functional Interface** 61](#_Toc137565410)

[**c) Một số quy tắc khai báo Functional Interface** 61](#_Toc137565411)

[**d)** **Cú pháp Lambda Expression** 61](#_Toc137565412)

[**e)Lợi ích** 62](#_Toc137565413)

[**f)Lưu ý** 62](#_Toc137565414)

[**6.JSP Template Engine** 65](#_Toc137565415)

[**Khái niệm JSP (JavaServer Pages)** 65](#_Toc137565416)

[**Lý do nên sử dụng JSP** 65](#_Toc137565417)

[**Ưu điểm của JSP** 65](#_Toc137565418)

[**Thiết lập môi trường** 65](#_Toc137565419)

[**Kiến trúc** 66](#_Toc137565420)

[**Luồng hoạt động** 66](#_Toc137565421)

[**Vòng đời** 66](#_Toc137565422)

[**Cú pháp** 66](#_Toc137565423)

[**Scriptlet** 66](#_Toc137565424)

[**Khai báo** 67](#_Toc137565425)

[**Biểu thức** 67](#_Toc137565426)

[**Comment** 68](#_Toc137565427)

[**Directive** 68](#_Toc137565428)

[**Actions** 69](#_Toc137565429)

[**Implicit Objects** 70](#_Toc137565430)

[**Control-Flow Statements** 70](#_Toc137565431)

[**Lệnh điều kiện** 70](#_Toc137565432)

[**Vòng lặp** 71](#_Toc137565433)

[**Toán tử** 71](#_Toc137565434)

[**Keyword** 72](#_Toc137565435)

[**Client Request** 72](#_Toc137565436)

[**Object HttpServletRequest** 74](#_Toc137565437)

[**Servlet Response** 77](#_Toc137565438)

[**Object HttpServletResponse** 79](#_Toc137565439)

[**Form Processing** 81](#_Toc137565440)

[**Cookie Processing** 81](#_Toc137565441)

[**Servlet Cookies Methods** 82](#_Toc137565442)

[**Setting Cookies JSP** 83](#_Toc137565443)

[**Chuyển hướng trang** 84](#_Toc137565444)

[**Kết luận** 85](#_Toc137565445)

[**7.Java Stream** 85](#_Toc137565446)

[Java Stream 85](#_Toc137565447)

[Đặc điểm 85](#_Toc137565448)

[Các thao tác 86](#_Toc137565449)

[Các method trong stream: 86](#_Toc137565450)

[**Terminal Operations** 86](#_Toc137565451)

[- Reduce method 86](#_Toc137565452)

[**Intermediate Operations** 87](#_Toc137565453)

[**Luồng song song – Parallel Streams** 87](#_Toc137565454)

[**8.Đóng gói và triển khai Website trên Tomcat Server** 88](#_Toc137565455)

[**Khái niệm** 88](#_Toc137565456)

[**Khái niệm về đề tài** 88](#_Toc137565457)

[**Các yếu tố chính** 89](#_Toc137565458)

[**ƯU VÀ NHƯỢC ĐIỂM CỦA APACHE TOMCAT** 89](#_Toc137565459)

[NỘI DUNG CHÍNH 89](#_Toc137565460)

[TOMCAT SERVER 89](#_Toc137565461)

[CÁCH CÀI ĐẶT TOMCAT SERVER 90](#_Toc137565462)

[CÁCH CẤU HÌNH TOMCAT SERVER 90](#_Toc137565463)

[ĐÓNG GÓI WEBSITE THÀNH FILE WAR 90](#_Toc137565464)

[TRIỂN KHAI TRÊN TOMCAT SERVER 91](#_Toc137565465)

[TRIỂN KHAI BẰNG GIAO DIỆN WEB CỦA TOMCAT MANAGER 91](#_Toc137565466)

[**TRIỂN KHAI BẰNG CÁCH ĐẶT FILE WAR VÀO THƯ MỤC WEBAPPS CỦA TOMCAT** 91](#_Toc137565467)

[**TRIỂN KHAI BẰNG CÁCH SỬ DỤNG MAVEN PLUGIN** 92](#_Toc137565468)

[**QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER** 92](#_Toc137565469)

[**QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER** 92](#_Toc137565470)

[**QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER** 93](#_Toc137565471)

[TỔNG KẾT 93](#_Toc137565472)

[**II. Bài tập** 94](#_Toc137565473)

[**1. Servlet** 94](#_Toc137565474)

[**Database: Sql server** 94](#_Toc137565475)

[**2. Spring Boot** 95](#_Toc137565476)

**Soạn đề cương Java Nâng Cao**

# **Lý thuyết**

# **Servlet**

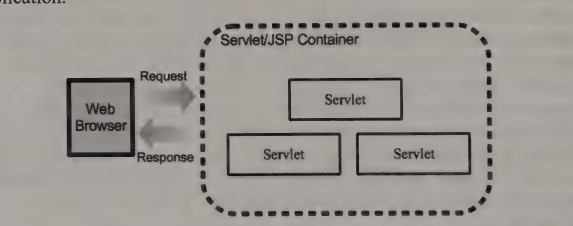
## **Tìm hiểu về công cụ và cách tổ chức chương trình trong Java Servlet**

### **Khái niệm:**

* Servlet là một công nghệ được sử dụng để tạo ra ứng dụng web.
* Servlet là một API cung cấp các interface và lớp bao gồm các tài liệu.
* Servlet là một thành phần web được triển khai trên máy chủ để tạo ra trang web động.

### **Kiến trúc:**

* Ứng dụng servlet bao gồm một hoặc nhiều servlets. JSP Page được biên dịch và chuyển từ servlet. Ứng dụng servlet chạy trong servlet container. Servlet container chuyển request từ người dùng sang ứng dụng servlet và ứng dụng servlet phản hồi tới người dùng.



### **Nhiệm vụ của hoạt động:**

* Đọc dữ liệu yêu cầu HTTP ẩn được gửi bởi khách hàng (trình duyệt). Điều này bao gồm các cookie, loại phương tiện truyền thông và các chương trình nén mà trình duyệt hiểu được, v.v.
* Xử lý dữ liệu và tạo ra các kết quả. Quá trình này có thể yêu cầu nói chuyện với một cơ sở dữ liệu, thực hiện một cuộc gọi RMI hoặc CORBA, gọi một dịch vụ Web, hoặc tính trực tiếp phản hồi.
* Gửi dữ liệu rõ ràng (tức là tài liệu) tới khách hàng (trình duyệt). Tài liệu này có thể được gửi bằng nhiều định dạng, bao gồm văn bản (HTML hoặc XML), nhị phân (hình ảnh GIF), Excel, v.v ...
* Gửi phản hồi HTTP ẩn cho khách hàng (trình duyệt). Điều này bao gồm nói với trình duyệt hoặc các trình khách khác loại tài liệu đang được trả về (ví dụ, HTML), thiết lập cookie và các tham số bộ nhớ đệm, và các tác vụ khác.

### **Công cụ sử dụng**

* là NetBeans và IntelliJ
* Apache Tomcat là hoạt động chung về tần suất hoạt động lập trình Java
* Xây dựng cách tổ chức chương trình
* Bộ chứa servlet/JSP là một máy chủ web đặc biệt có thể xử lý các servlet cũng như phục vụ các nội dung tĩnh.Servlet và JSP là hai trong vô số công nghệ được định nghĩa trong Phiên bản doanh nghiệp Java (EE). Các công nghệ Java EE khác bao gồm Java Dịch vụ tin nhắn (JMS), Enterprise JavaBeans (EJB), JavaServer Faces(JSF) và Tính bền bỉ của Java.

### **Môi trường làm việc của Servlet**

* Servlet chính là một lớp Java nên nó cần được thực thi trên máy ảo Java (gọi là JVM) thông qua một dịch vụ có tên là Servlet Engine.
* Theo đó, Servlet engine sẽ thực hiện tải lớp Servlet đầu tiên mà nó được yêu cầu hoặc tại thời điểm khi servlet engine bắt đầu. Tiếp đến, servlet sẽ ngừng tải để tập trung nguồn lực xử lý các yêu cầu khi Servlet engine bị dừng hoặc tắt.

### **Servlet Container**

* Servlet Container sử dụng Java để tự động tạo trang web ở phía máy chủ, xử lý các yêu cầu tính toán từ phía client và trả lại các kết quả theo request của client đó.
* Servlet Container thực chất là một phần của một web server tương tác với các Servlet.
* Và đúng như tên gọi Servlet Container giống như 1 cái thùng chứa tất cả các Servlet ở bên trong

### **API Servlet có bốn package Java:**

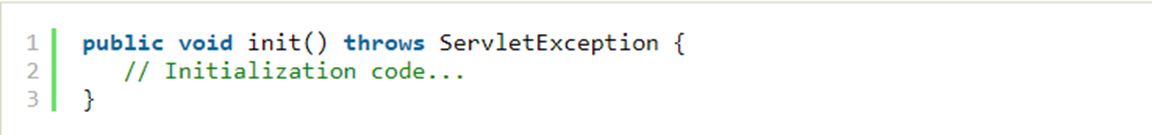
* javax.servlet: Chứa các class và interface xác định Contains giữa một servlet và một thùng chứa servlet.
* javax.servlet.http: Chứa các class và interface định nghĩa Contains giữa một HTTP servlet và một container servlet.
* javax.servlet.annotation: Chứa các chú thích để chú thích các servlet, bộ lọc và người nghe. Nó cũng chỉ định siêu dữ liệu cho chú thích các thành phần.
* javax.servlet.descriptor: Chứa các loại cung cấp có lập trìnhtruy cập vào thông tin cấu hình của ứng dụng web.

### **Vòng đời của Servlet**

* Một vòng đời của servlet có thể được định nghĩa là toàn bộ quá trình từ khi tạo ra đến khi hủy. Sau đây là tổng quan về vòng đời của servlet:
* Servlet được khởi tạo bằng cách gọi phương thức init ().
* Phương thức servlet service() được gọi để xử lý yêu cầu của khách hàng.
* Servlet được hủy bằng cách gọi phương thức destroy().
* Cuối cùng, servlet được thu thập bởi bộ sưu tập rác của JVM.

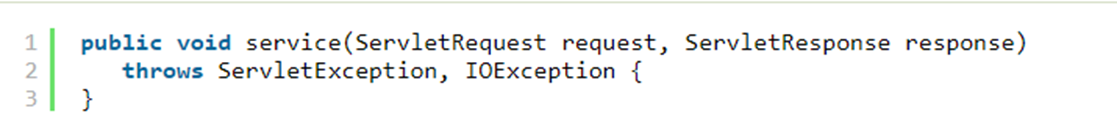
### **Phương thức init()**

* Phương thức init() được gọi chỉ một lần để khởi tạo servlet. Vì vậy, nó được sử dụng cho khởi tạo một lần, giống như với các phương thức init của applet.
* Servlet thường được tạo khi người dùng lần đầu tiên gọi một URL tương ứng với servlet, nhưng bạn cũng có thể chỉ định rằng servlet được tải khi máy chủ được khởi động lần đầu tiên.
* Khi một người dùng gọi một servlet, một thể hiện duy nhất của mỗi servlet sẽ được tạo ra, với mỗi yêu cầu của người dùng tạo ra một luồng mới được trao cho doGet hoặc doPost. Phương thức init() chỉ đơn giản tạo hoặc tải một số dữ liệu sẽ được sử dụng trong suốt vòng đời của servlet.

****

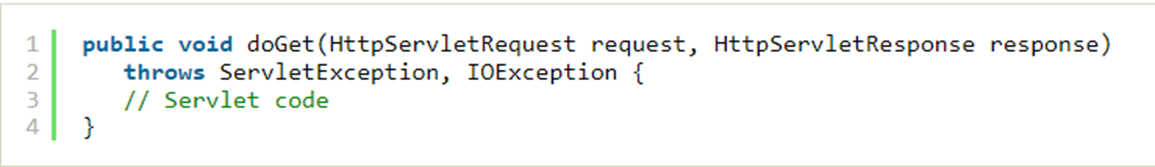
### **Phương thức service()**

* Phương thức service() là phương thức chính để thực hiện nhiệm vụ thực tế. Bộ chứa servlet (tức là web server) gọi phương thức service() để xử lý các yêu cầu đến từ khách hàng (trình duyệt) và trả về kết quả.
* Mỗi lần máy chủ nhận được một yêu cầu cho một servlet, máy chủ sẽ tạo ra một luồng mới và gọi phương thức service(). Phương thức service() kiểm tra kiểu yêu cầu HTTP (GET, POST, PUT, DELETE, v.v.) và gọi các phương thức doGet, doPost, doPut, doDelete, vv.

****

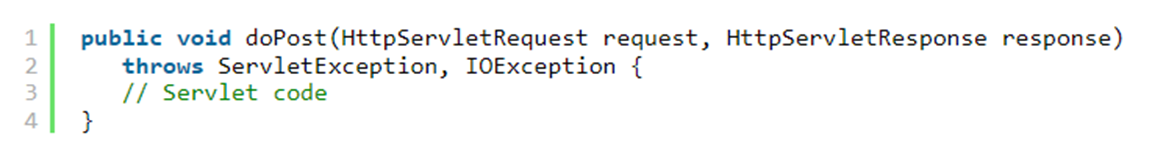
### **Phương thức doGet()**

* Một yêu cầu GET xuất phát từ một yêu cầu bình thường cho một URL hoặc từ một HTML Form mà không có METHOD chỉ định và nó phải được xử lý bằng phương thức doGet().

****

### **Phương thức doPost()**

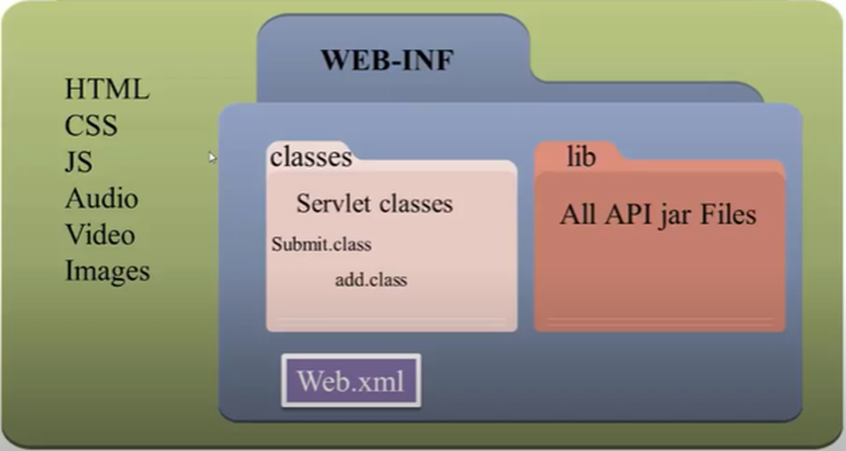
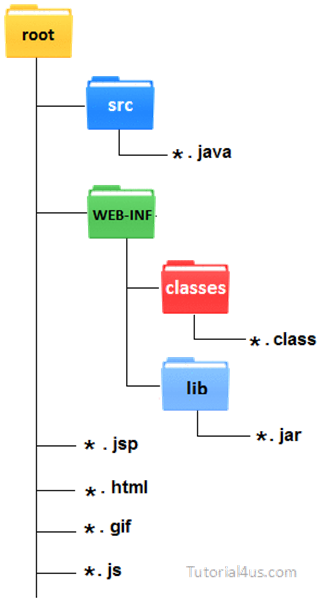
* Yêu cầu POST xuất phát từ một HTML Form cụ thể mà có METHOD là POST và nó phải được xử lý bởi phương thức doPost().

****

### **Phương thức destroy()**

* Phương thức destroy() chỉ được gọi một lần vào cuối chu kỳ sống của một servlet. Phương thức này cho phép servlet đóng kết nối cơ sở dữ liệu, chặn các luồng chạy gầm, viết các cookie hoặc đếm số lượt truy cập và thực hiện các hoạt động dọn dẹp khác.
* Sau khi phương thức destroy() được gọi, đối tượng servlet được đánh dấu để thu gom rác.
* ****

### **Cấu trúc thư mục**

****

### **Cách Servlet container và web server xử lý một request**

B1: Web Server sẽ nhận HTTP request

B2: Web Server chuyển tiếp yêu cầu tới Servlet Container

B3: Servlet được lấy ra một cách tự động và tải lên địa chỉ không gian của container, nếu nó nằm trong container.

B4: Container gọi init () method của servlet để khởi tạo (được gọi một lần khi servlet được load lần đầu tiên)

B5: Container gọi service () method của servlet để xử lý HTTP request, tức là đọc dữ liệu trong yêu cầu và hình thành một response

B6: Web server trả lại kết quả động đúng với vị trí yêu cầu

### **Mô hình MVC:**

MVC là viết tắt của cụm từ “Model-View-Controller“. Đây là mô hình thiết kế được sử dụng trong kỹ thuật phần mềm. MVC là một mẫu kiến trúc phần mềm để tạo lập giao diện người dùng trên máy tính. MVC chia thành ba phần được kết nối với nhau và mỗi thành phần đều có một nhiệm vụ riêng của nó và độc lập với các thành phần khác

* Model: Là bộ phận có chức năng lưu trữ toàn bộ dữ liệu của ứng dụng và là cầu nối giữa 2 thành phần bên dưới là View và Controller. Một model là dữ liệu được sử dụng bởi chương trình. Đây có thể là cơ sở dữ liệu, hoặc file XML bình thường hay một đối tượng đơn giản. Chẳng hạn như biểu tượng hay là một nhân vật trong game.
* View: Đây là phần giao diện (theme) dành cho người sử dụng. View là phương tiện hiển thị các đối tượng trong một ứng dụng. Chẳng hạn như hiển thị một cửa sổ, nút hay văn bản trong một cửa sổ khác. Nó bao gồm bất cứ thứ gì mà người dùng có thể nhìn thấy được.
* Controller: Là bộ phận có nhiệm vụ xử lý các yêu cầu người dùng đưa đến thông qua View. Một controller bao gồm cả Model lẫn View. Nó nhận input và thực hiện các update tương ứng.

## **Tìm hiểu về JSP, EL và JSTL**

### **JSP**

* JSP (JavaServer Pages) là một công nghệ phát triển web cho phép lập trình viên tạo các trang web động sử dụng Java và HTML. Trang JSP được xây dựng như các trang HTML thông thường, tuy nhiên nó cũng bao gồm các thẻ JSP để nhúng mã Java vào trong trang HTML, giúp các trang web trở nên động và tương tác với người dùng.
* Ví dụ:
* Sử dụng thẻ jsp:useBean để tạo một đối tượng JavaBean:

<jsp:useBean id="beanName" class="packageName.className"/>

* Sử dụng thẻ jsp:setProperty để gán giá trị cho một thuộc tính của đối tượng JavaBean:

<jsp:setProperty name="beanName" property="propertyName" value="propertyValue"/>

* Sử dụng thẻ jsp:getProperty để lấy giá trị của một thuộc tính của đối tượng JavaBean:

<jsp:getProperty name="beanName" property="propertyName"/>

* Sử dụng thẻ <% %> để thực hiện mã Java:

<% int x = 10; %>

* Sử dụng thẻ <%= %> để in ra giá trị của một biến Java:

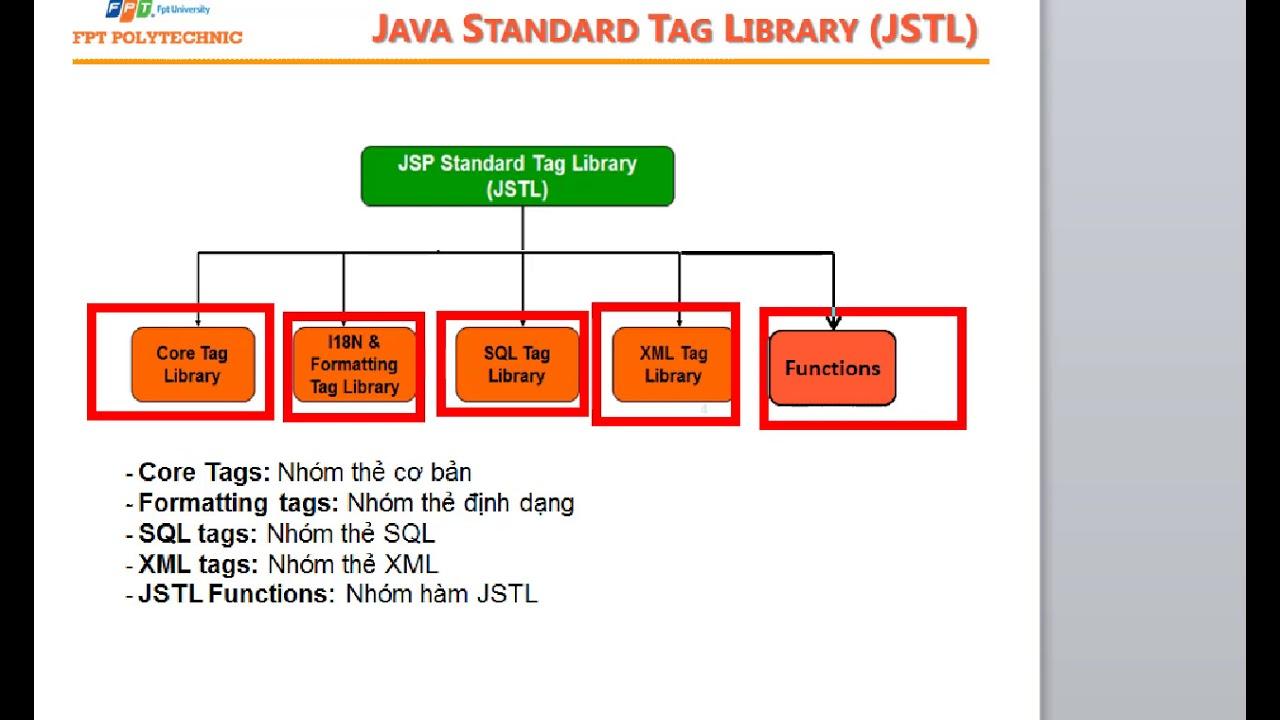
<% String message = "Hello World"; %>

<p><%= message %></p>

* Như vậy, JSP là một công nghệ phát triển web cho phép lập trình viên tạo các trang web động sử dụng Java và HTML. Các thẻ JSP cung cấp khả năng nhúng mã Java vào trong các trang HTML, giúp các trang web trở nên động và tương tác với người dùng.

|  |  |
| --- | --- |
| Ưu điểm | Mô tả |
| Extension to Servlet | JSP là mở rộng của Servlet . chúng ta có thể sử dụng tất cả các chức năng của Servlet ở trong JSP. Thêm nữa là chúng ta có thẻ sử dụng implicit objects, predefined tags, expression language and Custom tags trong JSP ,những thứ này là cho việc phát triển JSP dễ dàng |
| Easy to maintain | JSP có thể được dễ dàng quản lý vì chúng ta có thể dễ dàng chia việc xử lý business logic with the presentation logic.của chúng ta bằng presentation logic, trong Servlet chúng ta kết hợp xử lý business logic bằng presentation logic. |
| Fast Development: No need to recompile and redeploy | Nếu 1 trang JSP được bổ sung thêm (hay sửa đổi ) thì chúng ta không cần phải biên dịch lại hay deploy lại dự án , code servlet cần được cập nhật và biên dịch lại nếu chúng ta thay đổi giao diện ứng dụng |
| Less code than Servlet | Trong JSP chúng ta có thể sử dụng nhiều thẻ như là action tags, JSTL, custom tags, ...., điều này sẽ giảm mã chúng ta phải code hơn ngắn gọn ,hơn nữa có thể sử dụng EL, implicit objects, ... |

### **Jsp standard tag library**



### **Khái niệm**

* The JSP Standard Tag Library (JSTL) đại diện cho 1 tập thẻ (tags) để đơn giản hóa việc phát triển JSP

### **JSP Expression Language (EL)**

* **JSP Expression Language** (EL) giúp dễ dàng truy cập dữ liệu ứng dụng được lưu giữ trong các thành phần JavaBeans. JSP EL cho phép bạn tạo các Expression, gồm số học và logic. Bên trong một JSP EL, bạn có thể sử dụng các integer, các số floating point, string, các hằng có sẵn true hoặc false cho các giá trị Boolean, và null.

Cú pháp : ${expression}

* Toán tử cơ bản trong EL

JSP Expression Language (EL) hỗ trợ hầu hết các toán tử số học và logic được hỗ trợ bởi Java.

|  |  |
| --- | --- |
| **Toán tử** | **Miêu tả** |
| . | Truy cập một đặc tính của Bean hoặc Map entry |
| [] | Truy cập một phân tử mảng hoặc List |
| ( ) | Nhóm một subexpression để thay đổi thứ tự ước lượng |
| + | Phép cộng |
| - | Phép trừ hoặc phủ định một giá trị |
| \* | Phép nhân |
| / hoặc div | Phép chia |
| % hoặc mod | Phép chia lấy phần dư |
| == hoặc eq | Kiểm tra có bằng hay không |
| != hoặc ne | Kiểm tra tính không bằng |
| < hoặc lt | Kiểm tra tính nhỏ hơn |
| > hoặc gt | Kiểm tra tính lớn hơn |
| <= hoặc le | Kiểm tra tính nhỏ hơn hoặc bằng |
| >= hoặc ge | Kiểm tra tính lớn hơn hoặc bằng |
| && hoặc and | Phép AND logic |
| || hoặc or | Phép OR logic |
| ! hoặc not | Phần bù Boolean một ngôi |
| empty | Kiểm tra các giá trị biến rỗng |

- JSP EL cũng cho phép bạn sử dụng các hàm trong Expression. Những hàm này phải được định nghĩa trong các thư viện custom tag. Một sự sử dụng hàm có cú pháp sau:

${ns:func(param1, param2, ...)}

- Tại đây, ns là không gian tên của hàm đó, func là tên hàm và param1 là giá trị tham số đầu tiên. Ví dụ, hàm fn:length, mà là một phần của thư viện JSTL có thể được sử dụng như sau để nhận độ dài của chuỗi.

${fn:length("Get my length")}

- Để sử dụng một hàm từ bất kỳ thư viện thẻ nào (Standard hoặc Custom), bạn phải cài đặt thư viện đó trên Server và phải bao thư viện đó trong JSP bởi sử dụng <taglib> directive như đã giải thích trong chương JSTL.

- JSP Expression Language hỗ trợ các đối tượng ẩn sau:

|  |  |
| --- | --- |
| **Đối tượng ẩn** | **Miêu tả** |
| pageScope | Các biến scope từ phạm vi page |
| requestScope | Các biến scope từ phạm vi request |
| sessionScope | Các biến scope từ phạm vi session |
| applicationScope | Các biến scope từ phạm vi application |
| param | Các tham số request, dạng các chuỗi |
| paramValues | Các tham số request, dạng các tập hợp của các chuỗi |
| header | HTTP request header dạng các chuỗi |
| headerValues | HTTP request header dạng các tập hợp của các chuỗi |
| initParam | Các tham số khởi tạo context |
| cookie | Các giá trị Cookie |
| pageContext | Đối tượng JSP PageContext cho page hiện tại |

### e) **JavaServer Pages Standard Tag Library (JSTL)**

- **JavaServer Pages Standard Tag Library (JSTL)** là một tập hợp các thẻ JSP hữu ích, mà gói các tính năng lõi phổ biến tới các ứng dụng JSP.

- JSTL hỗ trợ tới các tác vụ phổ biến và có tính cấu trúc, ví dụ như các tính lặp và điều kiện, các thẻ để thao tác tài liệu XML, các thẻ đa ngôn ngữ, và các thẻ SQL. - - Nó cũng cung cấp Framework để tích hợp các Custom Tags với các thẻ JSTL.

- Các thẻ JSTL có thể được phân loại, theo tính năng của nó, thành các nhóm thư viện thẻ JSTL sau, mà có thể được sử dụng khi tạo một JSP page:

*Core Tags: Nhóm thẻ cơ bản*

*Formatting tags: Nhóm thẻ định dạng*

*SQL tags: Nhóm thẻ SQL*

*XML tags: Nhóm thẻ XML*

*JSTL Functions: Nhóm hàm JSTL*

* Nhóm Core Tags trong JSTL

Nhóm Core Tags là các thẻ JSTL được sử dụng phổ biến nhất. Sau đây là cú pháp đơn giản để bao thư viện JSTL Core trong JSP của bạn:

<%@ taglib prefix="c"

uri="http://java.sun.com/jsp/html/core" %>

Bảng dưới liệt kê các thẻ trong nhóm Core Tags trong JSTL:

|  |  |
| --- | --- |
| **Thẻ** | **Miêu tả** |
| Thẻ <c:out > trong JSTL | Giống <%= ... >, nhưng cho các Expression |
| Thẻ <c:set > trong JSTL | Thiết lập kết quả của một ước lượng Expression trong một 'scope' |
| Thẻ <c:remove > trong JSTL | Gỡ bỏ một biến mục tiêu (từ một biến scope cụ thể, nếu đã xác định) |
| Thẻ <c:catch> trong JSTL | Bắt bất kỳ Throwable mà xuất hiện trong thân của nó và trưng bày nó một cách tùy ý |
| Thẻ <c:if> trong JSTL | Thẻ điều kiện đơn giản, mà ước lượng phần thân của nó nếu điều kiện đã cho là true |
| Thẻ <c:choose> trong JSTL | Thẻ điều kiện đơn giản mà thiết lập một context cho các hoạt động điều kiện loại trừ, được đánh dấu bởi <when> và <otherwise> |
| Thẻ <c:when> trong JSTL | Thẻ phụ của <choose> mà include phần thân của nó nếu điều kiện được ước lượng là true |
| Thẻ <c:otherwise > trong JSTL | Thẻ phụ của <choose> mà theo sau thẻ <when> và chỉ chạy nếu tất cả điều kiện trước được ước lượng là 'false' |
| Thẻ <c:import> trong JSTL | Thu nhận một URL tuyệt đối hoặc quan hệ và trưng bày nội dung của nó tới hoặc trang đó, một String trong 'var', hoặc một Reader trong 'varReader' |
| Thẻ <c:forEach > trong JSTL | Thẻ lặp cơ bản, chấp nhận nhiều kiểu tập hợp khác nhau và hỗ trợ subsetting (chia tập con) và tính năng khác |
| Thẻ <c:forTokens> trong JSTL | Lặp qua các token, được phân biệt bởi các dấu phân tách (delimiter) đã cung cấp |
| Thẻ <c:param> trong JSTL | Thêm một parameter tới một URL của thẻ đang chứa 'import' |
| Thẻ <c:redirect > trong JSTL | Redirect tới một URL mới |
| Thẻ <c:url> trong JSTL | Tạo một URL với các tham số truy vấn tùy ý |

## **Tìm hiểu về lập trình JPA (Java Persistence API)**

### **Khái niệm JPA**

+ JPA là một đặc tả Java cho việc ánh xạ giữa các đối tượng Java với cơ sở dữ liệu quan hệ sử dụng công nghệ phổ biến là ORM (Object Relational Mapping).

+ Nói dễ hiểu hơn JPA là một công nghệ để thực hiện các thao tác với cơ sở dữ liệu quan hệ.

+ JPA cung cấp một cách tiếp cận trừu tượng hóa cho việc lưu trữ và truy xuất dữ liệu từ cơ sở dữ liệu.

+ JPA giúp cho việc phát triển ứng dụng Java trở nên dễ dàng và linh hoạt hơn..

+ JPA cũng cung cấp các tính năng quan trọng như quản lý định danh và quan hệ giữa các đối tượng, các chiến lược tải dữ liệu để giảm thiểu số lượng truy cập cơ sở dữ liệu, và đảm bảo tính nhất quán dữ liệu trong các giao dịch đa luồng. JPA được tích hợp với các framework như Spring và Hibernate, và được hỗ trợ bởi các hệ quản trị cơ sở dữ liệu phổ biến như MySQL, Oracle và PostgreSQL.

### **Object Relational Mapping (ORM)**

* ORM là viết tắt của Object Relational Mapping, nó chuyển đổi dữ liệu từ ngôn ngữ hướng đối tượng sang Database quan hệ và ngược lại. Ví dụ, trong Java nó được thực hiện với sự trợ giúp của Reflection và JDBC.
* ORM có khả năng xử lý các thao tác của nhiều loại cơ sở dữ liệu khác nhau một cách dễ dàng mà không quan tâm đến loại database sử dụng (SQL Server, MySQL, PostgreSQL, …) hay loại thao tác sử dụng (INSERT, UPDATE, DELETE, SELECT, …).
* Các ORM Framework có thể sử dụng cho JPA như: Hibernate, iBatis, Eclipse Link, OpenJPA, ….

### **c) Kiến trúc JPA**

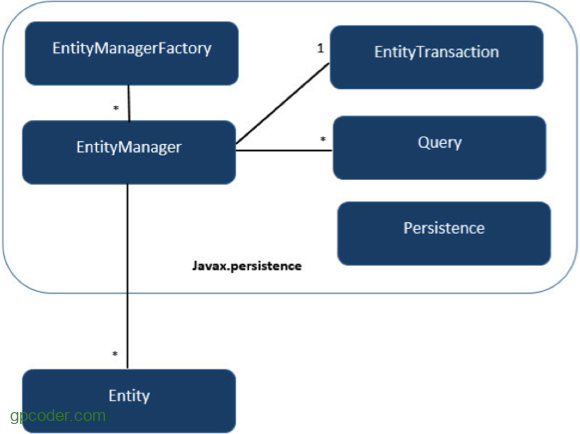
- Kiến trúc JPA bao gồm ba thành phần chính là:

+ Entity.

+ EntityManager.

+ EntityManagerFactory.

- Ngoài ra còn có, EntityTransaction, Persistence, Query.



**Entity:** Entity là các đối tượng thể hiện tương ứng 1 table trong cơ sở dữ liệu. Entity thường là các class POJO đơn giản, chỉ gồm các phương thức getter, setter.

Một số đặc điểm của một Entity:

• Entity có thể tương tác với cơ sở dữ liệu quan hệ.

• Entity được xác định thông qua một định danh (id), tương đương với khóa chính trong table của cơ sở dữ liệu quan hệ.

• Entity hỗ trợ transaction.

• Entity hỗ trợ kế thừa giống như những class Java khác.

**EntityManager**

EntityManager là một interface cung cấp các API cho việc tương tác với các Entity.

Một số chức năng cơ bản của EntityManager như:

* Persist: phương thức này dùng để lưu một thực thể mới tạo vào cơ sở dữ liệu.
* Merge: dùng để cập nhật trạng thái của entity vào cơ sở dữ liệu.
* Remove: xóa một instance của entity.

**JPA Entity Manager** là một thành phần quan trọng của Java Persistence API (JPA), được sử dụng để quản lý các đối tượng (entities) trong cơ sở dữ liệu quan hệ.

Khi chúng ta thao tác với JPA Entity Manager, chúng ta có thể thực hiện các hoạt động CRUD (Create, Read, Update, Delete) với các đối tượng entity. Điều này có nghĩa là chúng ta có thể tạo mới, đọc, cập nhật hoặc xóa các bản ghi trong cơ sở dữ liệu.

Một số đặc tính của JPA Entity Manager là:

Quản lý vòng đời của đối tượng (entity lifecycle management): Entity Manager theo dõi vòng đời của các đối tượng entity. Khi chúng ta tạo mới một đối tượng, nó được thêm vào trong quản lý của Entity Manager. Khi đối tượng đó được cập nhật hoặc xóa, Entity Manager cũng cập nhật trạng thái của nó trong quản lý.

Được sử dụng để tạo truy vấn (querying): Chúng ta có thể sử dụng Entity Manager để tạo và thực thi các truy vấn với cơ sở dữ liệu.

Quản lý transaction: Entity Manager cung cấp các phương thức để quản lý các transaction với cơ sở dữ liệu.

Cung cấp cơ chế caching: Entity Manager cung cấp cơ chế caching để tăng hiệu suất truy cập cơ sở dữ liệu. Khi một đối tượng entity được truy vấn, Entity Manager sẽ lưu trữ nó trong cache. Lần truy vấn tiếp theo đối tượng đó chỉ cần được truy vấn trong cache, giảm thiểu số lần truy vấn vào cơ sở dữ liệu.

Cho phép xử lý đồng thời (concurrency): Entity Manager cho phép xử lý đồng thời nhiều request truy vấn cùng lúc. Điều này giúp tăng hiệu suất ứng dụng.

Được quản lý bởi container: Entity Manager được quản lý bởi container và có thể được sử dụng trong môi trường enterprise.

**EntityManagerFactory**

EntityManagerFactory được dùng để tạo ra một instance của EntityManager.

**Persistence**

Một Persistence định nghĩa một tập hợp các Entity class được quản lý bởi 1 instance của EntityManager trong ứng dụng.

Persistence (javax.persistence.Persistence) class bao gồm các phương thức static để lấy instance của EntityManagerFactory.

**EntityTransaction**

Một Transaction là một tập hợp các thao tác trong đó tất cả các thao tác phải được thực hiện thành công hoặc tất cả thất bại.

Một database transaction bao gồm một tập hợp các câu lệnh SQL được committed hoặc rolled back trong một unit.

EntityTransaction có quan hệ 1-1 với EntityManager. Bất kỳ thao tác nào được bắt đầu thông qua đối tượng EntityManager đều được đặt trong một Transaction. Đối tượng EntityManager giúp tạo EntityTransaction.

**Query**

Đây là một interface, được mỗi nhà cung cấp JPA implement để có được các đối tượng quan hệ đáp ứng các tiêu chí (criteria) truy vấn

### **d) Lợi ích của JPA**

* Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu).
* Không phụ thuộc vào các framework ORM.
* Có nhiều nhà cung cấp hỗ trợ cài đặt JPA.
* Dữ liệu có thể được lưu trữ thông qua việc ORM.

### **Tại sao nên dùng JPA**

* Viết ít code hơn.
* Performance tốt.
* Độc lập về database.
* Không phải làm việc với SQL.
* Hỗ trợ cấu hình triển khai bằng annotation và xml.
* Có nhiều framework ORM miễn phí hỗ trợ có thể dùng để phát triển nhiều loại ứng dụng khác nhau.
* JPA là một đặc tả đã được chuẩn hóa và là một thành phần trong đặc tả EJB 3.
* Dễ dàng chuyển từ một ORM này sang một ORM khác. Ví dụ từ iBatis sang Hibernate.

### **Các tính năng của JPA**

Một số tính năng của JPA:

* JPA hỗ trợ plugable, tức là có thể sử dụng nhiều 3rd khác nhau như Hibernate hay MyBatis.
* Hỗ trợ cấu hình thông qua annotation và xml.
* Giảm bớt số lớp yêu cầu cho việc phát triển persistence.
* Không cần phải viết các mô tả triển khai trong XML. Các Annotation dựa trên metadata đã hỗ trợ trong các ứng dụng JPA.
* Đã chuẩn hóa ORM và dễ dàng phát triển hơn.
* JPA hỗ trợ truy vấn động và tĩnh.
* Nhiều IDE hỗ trợ phát triển ứng dụng JPA và có thể tự động sinh code ánh xạ từ cơ sở dữ liệu thành các entity và ngược lại.
* One-To-One Mapping: thêm @OneToOne annotation trước property
* One-To-Many Mapping: thêm @OneToMany(targetEntity = className.class) trước property kiểu List của entity đầu 1, sẽ tạo ra một bảng trung gian đại diện mapping cho 2 entity
* Many-To-One Mapping: thêm @ManyToOne trước property kiểu của entity đầu 1
* Many-To-Many Mapping: thêm @ManyToMany(targetEntity = className.class) trước property kiểu List cho cả 2 entity, JPA sẽ tự tạo ra bảng trung gian đại diện cho quan hệ Many-To-Many của 2 entity
* JPA Cascading Operations:

|  |  |
| --- | --- |
| **Cascade Operations** | **Description** |
| PERSIST | In this cascade operation, if the parent entity is persisted then all its related entity will also be persisted. |
| MERGE | In this cascade operation, if the parent entity is merged then all its related entity will also be merged. |
| DETACH | In this cascade operation, if the parent entity is detached then all its related entity will also be detached. |
| REFRESH | In this cascade operation, if the parent entity is refreshed then all its related entity will also be refreshed. |
| REMOVE | In this cascade operation, if the parent entity is removed then all its related entity will also be removed. |
| ALL | In this case, all the above cascade operations can be applied to the entities related to parent entity |

* Ví dụ định nghĩa cascading operations: @OneToOne(cascade=CascadeType.Persist)
* JPQL: là query language thực hiện các thao tác với cơ sở dữ liệu dựa trên các entity
* Creating Query in JPQL: createQuery(string name): là phương thức của EntityManager interface dùng để tạo một instance cua Query interface để thực thi câu lệnh SQL

Query query = em.createQuery("Select s.s\_name from StudentEntity s");

* createNamedQuery(String name): là phương thức của EntityManager interface dùng để tạo một instance của Query interface cho phép thực thi câu lệnh thông qua tên query
* Có thể định nghĩa queries trong khi định nghĩa entity class theo dạng annotation:

@NamedQuery(name = "find name" , query = "Select s from StudentEntity s")

* Một số phương thức thực thi query của Query Interface:
  + int executeUpdate() - This method executes the update and delete operation.
  + int getFirstResult() - This method returns the first positioned result the query object was set to retrieve.
  + int getMaxResults() - This method returns the maximum number of results the query object was set to retrieve.
  + java.util.List getResultList() - This method returns the list of results as an untyped list.
  + Query setFirstResult(int startPosition) - This method assigns the position of first result to retrieve.
  + Query setMaxResults(int maxResult) - This method assigns the maximum numbers of result to retrieve.
* JPA Criteria Api: là cách thường dùng nhất để truy vấn dữ liệu
* Tạo Criteria Query:
  + Tạo object CriteriaBuilder interface bằng cách gọi hàm getCriteriaBuilder() của EntityManager

EntityManager em = emf.createEntityManager();

     CriteriaBuilder cb=em.getCriteriaBuilder();

* + Tạo CriteriaQuery thông qua tạo query object

CriteriaQuery<StudentEntity> cq=cb.createQuery(StudentEntity.**class**);

* + Gọi từ CriteriaQuery để thiết lập query root

Root<StudentEntity> stud=cq.from(StudentEntity.**class**);

* + Gọi hàm select của CriteriaQuery để xác định type của kết quả truy vấn:

CriteriaQuery<StudentEntity> select = cq.select(stud);

* + Tạo query:

Query q = em.createQuery(select);

* + List<StudentEntity> list = q.getResultList();
* Các hàm của Criteria API Query:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clause** | **Criteria API Interface** | **Methods** | **Chức năng** |  |
| SELECT | CriteriaQuery | select() | Lấy dữ liệu từ database, thực hiện thông qua 1 biểu thức hoặc nhiều biểu thức | Select()  MultiSelect() |
| FROM | AbstractQuery | from() |  |  |
| WHERE | AbstractQuery | where() | Thêm điều kiện cho truy vấn |  |
| ORDER BY | CriteriaQuery | orderBy() | Sắp xếp dữ liệu tăng dần hoặc giảm dần | orderBy() |
| GROUP BY | AbstractQuery | groupBy() |  |  |
| HAVING | AbstractQuery | having() |  |  |

## **4. Tìm hiểu về Scope, Listener và Filter trong Java Web**

### - a) **Application Context/Scope**

- Application scope được định nghĩa bởi javax.serverlet.ServletContext interface

- Trong scope này data đã tồn tại và có thể truy cập từ các session , request khác nhau và chỉ bị huỷ khi ứng dụng web bị shutdown

- getServletContext()

- getServletConfig().getServlet.Context()

### - b) **RequestScope**

- Request scope bắt đàu ngay khi một HTTP request được gửi tới server và kết thúc khi server trả về một HTTP response

- Các tham số/ thuộc tính trong một Request scope có thể được truy cập từ cá servlet hoặc jsp cùng phục vụ trong 1 request.

- Request object có sẵn trong JSP page như là 1 object ẩn, ta có thể set value cho 1 thuộc tính trong request object từ servlet và lấy nó ra ở JSP

### - c) **Session scope:**

- Một session scope bắt đầu khi một client thành lập kết nối ứng dụng web cho tới khi hết thời gian timeout hoặc browser bị đóng

- Một session scope bắt đầu khi một client thành lập kết nối với ứng dụng web cho tới khi hết thời gian timeout hoặc browser bị đóng

- Session object sẵn dùng trong JSP page như một object ẩn

- Trong servlet có thể lấy đối tượng object bằng cách gọi request.getSession().

### - d) **JSP page scope**

- Pagescope giới hạn bởi phạm vi và thời gian tồn tại của các thuộc tính trong cùng 1 page nơi mà nó được tạo

- Nó sẵn dùng trong một JSP page giống như một object ẩn.

### - e) **Filter:**

- Filter hay còn gọi là bộ lọc: là một đối tượng được gọi trong quá trình tiền xử lý và hậu xử lý của một yêu cầu

- Nó dễ kết hợp vào hệ thống, được định nghĩa trong file web.xml. Khi cần có thể thêm hoặc xóa filter mà không ảnh hưởng đến servlet

### - f) **Sử dụng filter khi:**

- Ghi lại tất cả các request đến

- Ghi nhật ký địa chỉ IP của máy tính từ các request nguồn

- Chuyển đổi, nén dữ liệu

- Mã hoá và giải mã, xác thực đầu vào.

### - g) **Các phương thức servlet filter**

- Phương thức: public void doFilter: được gọi bởi container khi mỗi cặp request/response được truyền thông qua chain

- Phương thức: public void Init(FilterConfig filterConfig): phương thức này được triệu hồi bởi web container để chỉ một filter mà đang được đặt bên trong service

- Phương thức public void destroy(): được gọi bởi web container để chỉ một filter bị huỷ khỏi service

- Bổ sung: thêm mô tả của listener

|  |  |
| --- | --- |
| **Listener Interface** | **Description** |
| javax.servlet.AsyncListener | Listener sẽ cảnh báo nếu một hành động không đồng bộ bắt đầu trên ServletRequest mà Listener được đính kèm đã hoàn thành, hết thời gian chờ hoặc không thành công |
| javax.servlet.ServletContextListener | Interface để nhận các sự kiện thông báo thay đổi vòng đời của ServletContext |
| javax.servlet.ServletContextAttributeListener | Interface nhận sự kiện thông báo thay đổi thuộc tính của ServletContext. |
| javax.servlet.ServletRequestListener | Interface nhận thông báo về các yêu cầu đầu vào và đầu ra thoát khỏi phạm vi của ứng dụng web |
| javax.servlet.ServletRequestAttributeListener | Interface nhận vào sự kiện thông báo thay đổi thuộc tính ServletRequest. |
| javax.servlet.http.HttpSessionListener | Interface nhận các sự kiện thông báo thay đổi vòng đời của HttpSession |
| javax.servlet.http.HttpSessionBindingListener | Khi một đối tượng bị ràng buộc hoặc giải phóng khỏi phiên, nó sẽ được cảnh báo bởi javax.servlet.http.HttpSessionBindingListener. |
| javax.servlet.http.HttpSessionAttributeListener | Interface nhận sự kiện thông báo thay đổi thuộc tính HttpSession |
| javax.servlet.http.HttpSessionActivationListener | Các đối tượng được kết nối với javax.servlet.http.HttpSessionActivationListener |

## **5. Tìm hiểu về Layout và Đa ngữ (Internationalization - 118N)**

### **a)Đa ngữ (Internationalization – I18N)**

* Internationalisation: là việc kích hoạt 1 trang web có khả năng cung cấp nhiều phiên bản với các ngôn ngữ khác nhau.
* Localisation (L10N) là việc thêm các resource tới một website để thích nghi với một khu vực địa lý – văn hóa cụ thể.
* Những khu vực địa lý – văn hóa đó được gọi là các **locale**

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | **String getCountry()**  Phương thức này trả về mã hóa country/region trong dạng chữ hoa cho Locale này trong định dạng ISO 3166 2-letter |
| 2 | **String getDisplayCountry()**  Phương thức này trả về một tên của country của locale đó mà thích hợp để hiển thị cho người dùng |
| 3 | **String getLanguage()**  Phương thức này trả về mã hóa ngôn ngữ trong dạnh chữ thường cho Locale này trong định dạng ISO 639 |
| 4 | **String getDisplayLanguage()**  Phương thức này trả về một tên ngôn ngữ của locale đó mà thích hợp hiển thị cho người dùng |
| 5 | **String getISO3Country()**  Phương thức này trả về một tên 3 ký tự viết tắt cho country của Locale này |
| 6 | **String getISO3Language()**  Phương thức này trả về một tên 3 ký tự viết tắt cho language của Locale này |

+ Thực chất thì I18n là ruby gem được hỗ trợ có sẵn kể từ rails 2.2, vì thế chúng ta chỉ việc thiết lập một vài bước đơn giản để sử dụng chứ không cần cài đặt nữa.

+ Mặc định app rails sẽ hỗ trợ ngôn ngữ English nhưng chúng ta có thể dễ dàng mở rộng qua các thứ tiếng khác như Lào, Campuchia, Thái, ...bla..bla.. mà bạn muốn có trong app của mình

* Ưu điểm của đa ngữ i18n trong Java Servlet: Hỗ trợ nhiều ngôn ngữ: Giúp ứng dụng dễ dàng hỗ trợ nhiều ngôn ngữ, phù hợp với người dùng đa dạng từ nhiều quốc gia. Tách biệt nội dung và mã nguồn: Nội dung được lưu trữ riêng biệt trong các tệp properties, giúp dễ dàng cập nhật và quản lý nội dung mà không ảnh hưởng đến mã nguồn chính. Tận dụng các tiện ích của Java: Sử dụng các class như ResourceBundle và Locale giúp việc xử lý đa ngữ dễ dàng và hiệu quả hơn. Nhược điểm của đa ngữ i18n trong Java Servlet:
* Phức tạp hóa quá trình phát triển: Việc hỗ trợ đa ngữ có thể làm tăng độ phức tạp của mã nguồn và quá trình phát triển. Bảo trì và cập nhật: Việc duy trì và cập nhật nội dung đa ngữ trong các tệp properties có thể gây khó khăn, đặc biệt khi số lượng ngôn ngữ hỗ trợ tăng lên.
* Thư viện:

Dynamic Internationalization - I18N **java.util.Locale request.getLocale()**

## **b)Bố cục trang Web bằng Thymeleaf Layout**

### - **Giới thiệu về Thymeleaf Layout:**

- Là thư viện mở rộng của Thymeleaf giúp cho xử lý layout trong ứng dụng web Java

- Hỗ trợ modularity và tái sử dụng fragment HTML, truyền tham số và kế thừa layout

- Sử dụng Thymeleaf Layout giúp xây dựng trang web nhanh chóng dễ dàng và hiệu quả.

### - **Tính năng:**

- Cung cấp tính năng layout để tái sử dụng mã HTML giữa các trang

- Cung cấp tính năng thay thế các phần tử trong bảng bằng cách sử dụng Thymeleaf

- Hỗ trợ tính năng kế thừa layout giúp cho việc tạo ra các trang có cấu trúc giống nhau

### - **Ưu điểm:**

- Tái sử dụng code

- Tiết kiệm thời gian , linh hoạt, dễ bảo trì

### - **Các khái niệm cơ bản của Thymeleaf Layout:**

- Khuôn mẫu (template layout) là layout chung cho toàn bộ trang web , giúp tái sử dụng mã nguồn HTML

- Mẫu con: là phần tử HTML được sử dụng trong khuôn mẫu hoặc trang con giúp tái sử dụng mã HTML

### - **Khuôn mẫu(Thymeleaf Layout):**

Là layout chung cho toàn bộ trang web, giúp tái sử dụng mã HTML.

### - **Các biểu thức đơn giản:**

+ Hiển thị giá trị của biến: th:text=^${name}^

+ Điều kiện hiển thị: th:if=${status == ‘active’}

+ Lặp lại danh sách th:each=”item:${item}”

+ Thao tác với thuộc tính HTML: th:attr=”href=@{/home}”

+ Gọi phương thức: th:onclick=”|submitForm{‘${param}’}|”

### - **Sử dụng Thymeleaf Layout để tạo bố cục web**

### - **Định nghĩa khuôn mẫu**

- + Tệp HTML layout.html chứa khuôn mẫu cho toàn bộ trang web

- + Sử dụng thể th:replace để đặt các mẫu con vào các vị trí tương ứng.

- + Sử dụng thẻ th:Fragment để định nghĩa các phần từ trong khuôn mẫu mà có thể sử dụng ở nhiều trang con.

### - **Định nghĩa mẫu con**

+ Tệp HTML header.html và footer.html chứa các mẫu con phần đầu và cuối của trang web.

+ Sử dụng thẻ th:fragment để đặt tên cho mẫu con để có thể đặt nó vào vị trí tương ứng trong khuôn mẫu.

### - **Chèn các mẫu con vào các khuôn mẫu**

+ Tệp HTML index.html chứa nội dung chính của trang web và sử dụng khuôn mẫu từ tệp layout.html

+ Sử dụng thẻ th:insert hoặc th:replace để đặt mẫu con vào các vị trí tương ứng trong khuôn mẫu

### - **Sử dụng thẻ Thymeleaf để truyền dữ liệu**

+ Tệp HTML index.html, có thể sử dụng các biểu thức Thymeleaf để truyền giá trị vào các thuộc tính HTML hoặc phần tử HTML

### - **Ưu điểm**

+ Cho phép lập trình viên tái sử dụng các thymeleaf layout

+ Dễ dàng định nghĩa và sử dụng các fragment

+ Khả năng tùy chỉnh mạnh mẽ và hỗ trợ các tính năng tiên tiến

### - **Nhược điểm**

**+** Tốc độ xử lý chậm hơn so với các giải pháp khác như JSP hoặc Thymeleaf thường

**+** Cần có kiến thức và kinh nghiệm về Thymeleaf Layout để sử dụng và tùy chỉnh các tính năng của nó

**+** Việc tạo các template layout phức tạp và phù hợp với yêu cầu của ứng dụng có thể đòi hỏi nhiều thời gian và công sức hơn

# **II. Spring Boot**

## **1.Tổng quan Spring Boot**

### **a)Spring Framework**

* Spring là 1 framework mã nguồn mở, cung cấp cấu trúc để phát triển các ứng dụng java có hiệu năng cao
* Spring tổ chức theo module nên dễ kiểm thử và tái sử dụng lại code
* Tính năng cốt lõi:
* Phát triển Java Desktop, ứng dụng mobile và web.
* Mục tiêu chính của Spring là giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn dựa trên mô hình sử dụng POJO (Plain Old Java Object)

### **b)Spring MVC**

* Spring MVC là một framework dựa trên Spring, được thiết kế dành cho xây dựng ứng dụng nền tảng web.
* Spring MVC dựa trên mô hình MVC (Model-View-Controller)
* **Ưu điểm:**
* Các tầng trong Spring MVC độc lập nên việc unit test trở nên dễ dàng hơn
* Phần view có thể tích hợp nhiều Framework về UI như JSF, Thymeleaf
* Hỗ trợ cả Annotation và XML config giúp việc phát triển code nhanh hơn và clean hơn
* Cung cấp việc phân chia một cách rõ ràng, linh hoạt giữa controller, service, data access layer
* **Nhược điểm:** Khi sử dụng Spring MVC, chúng ta cần cấu hình
  + Component scan
  + Dispatcher Servlet
  + View resolver
  + Web jars

### **c)Spring Boot**

* Spring Boot là được phát triển dựa trên Spring, cung cấp nền tảng cho phép phát triển các ứng dụng độc lập nhanh chóng với cấu hình tối thiểu mà không cần thiết lập toàn bộ cấu hình Spring.
* Được các lập trình viên yêu thích vì môi trường sử dụng nhanh chóng, cho phép phát triển trực tiếp vào logic thay vì lựa chọn với việc cấu hình và thiết lập
* Spring boot là một framework dựa trên microservice
* Tính năng chính của Spring Boot
* Tránh cấu hình nặng của XML trong Spring
* Cung cấp khả năng bảo trì và tạo các REST Endpoint dễ dàng
* Bao gồm máy chủ Tomcat nhúng
* Triển khai dễ dàng, các tệp tin war và jar có thể dễ dàng triển khai máy chủ Tomcat.

### **d)Spring Boot vs Spring Framework**

|  |  |
| --- | --- |
| **Spring Boot** | **Spring Framework** |
| * Được sử dụng rộng rãi để phát triển các REST APIs * Rút gọn mã và cung cấp cách dễ dàng để phát triển Web Application * Tính năng Autoconfiguration tự động cấu hình các lớp dựa trên yêu cầu * Tạo một ứng dụng độc lập ít cấu hình hơn * Cung cấp máy chủ nhúng như Tomcat | * Là một framework Java EE sử dụng rộng rãi để xây dựng các ứng dụng * Nó nhằm mục đích đơn giản việc phát triển Java EE * Tính năng chính là Dependency Injection * Làm mọi thứ đơn giản hơn bằng cách cho phép phát triển loosely coupled application * Cần thiết lập server một cách rõ ràng |

### **e)Spring Boot vs Spring MVC**

|  |  |
| --- | --- |
| **Spring MVC** | **Spring Boot** |
| * Model View Controller-based web framework dựa trên Spring * Cung cấp tính năng sẵn sàng sử dụng để xây dựng một ứng dụng web * Cần nhiều file mô tả triển khai * Chỉ định từ dependency riêng biệt * Cần nhiều thời gian để đạt được điều tương tự | * Là một module của Spring để đóng gói ứng dụng * Cung cấp cấu hình mặc định để xây dựng Spring-powerd framework * Không yêu cầu đối với các file mô tả khi triển khai * Kết hợp các dependencies lại với nhau trong một đơn vị duy nhất * Giảm thời gian phát triển và tăng năng suất |

### **f)Môi trường & công cụ**

* **Công cụ:** Eclipse, Intellij, Netbeans, VsCode, Intellij IDEA, …
* **Extensions VS code:** Extension Pack For Java, Debugger For Java, Spring Boot Extension Pack, Spring Boot Dashboard, Spring Boot Tools, …

### **g)Tạo dự án Spring Boot**

* Có rất nhiều cách để tạo dự án spring boot, nhưng phổ biến nhất là:
* Sử dụng trang Spring Initializr
* Sử dụng Spring Initializr hỗ trợ cho các IDE Spring Tool Suite(STS), IntelliJ IDEA Ultimate and IntelliJ IDEA Community Edition(Limited Configuration options), Netbeans and VSCode
* Quá trình tạo:
* Project: Lựa chọn package manager cho project là Gradle hay Maven
* Language: ngôn ngữ sử dụng cho project
* Spring Boot: phiên bản sử dụng cho project
* Project Metadata
  + Group: thông tin về nhóm, group id nên dùng đầy đủ theo tên domain (ex: com.example)
  + Artifact: thông tin về ID Project
  + Name: Tên project, viết thường, phân cách bởi dấu “-”
  + Description: Thông tin mô tả về project
  + Package Name: Tên package sau khi khởi tạo
  + Packaging: packaging type (Jar or War)
  + Java: phiên bản java tương thích với máy
  + Dependencies:
    - Spring Boot Devtools: Tính năng hỗ trợ LiveReload giúp khởi động lại ứng dụng nhanh hơn, giảm thời gian chờ sau mỗi lần chỉnh sửa source code.
    - Spring Web: Cung cấp các thư viện để phát triển ứng dụng Spring MVC Web application như spring-webmvc, jackson-json, validation-api and tomcat.; Spring Rest Api. Ngoài ra, còn cấu hình các bean chung như DispatcherServlet, Resource Handlers, MessageSource…
    - Lombok: Hỗ trợ code ngắn gọn bằng cách rút gọn một số boiler-plate code qua annotation như Getter, Setter, Constructor, …

### **h)Cấu trúc dự án sau khi tạo**

│ .gitignore

│ mvnw

│ mvnw.cmd

│ pom.xml

├───.mvn: **thư mục maven tạo ra**

├───src: **thư mục chứa source code chính**

│ ├───main

│ │ ├───java

│ │ │ └───com

│ │ │ └───ltp

│ │ │ └───gradesubmission

│ │ │ │ GradeSubmissionApplication.java

│ │ │ │

│ │ │ ├───constants

│ │ │ │ Constants.java

│ │ │ │

│ │ │ ├───DTO

│ │ │ ├───entity

│ │ │ │ Grade.java

│ │ │ │

│ │ │ ├───exception

│ │ │ ├───repository

│ │ │ │ GradeRepository.java

│ │ │ │

│ │ │ ├───service

│ │ │ │ GradeService.java

│ │ │ │ GradeServiceImpl.java

│ │ │ │

│ │ │ ├───validation

│ │ │ │ Score.java

│ │ │ │ ScoreValidator.java

│ │ │ │

│ │ │ └───web

│ │ │ └───controller

│ │ │ GradeController.java

│ │ │

│ │ └─── resources: **thư mục chứa resources**

│ │ │ application.properties: **tệp chứa các thuộc tính toàn ứng**

**dụng. Spring đọc các thuộc tính được xác định trong tệp này**

**định cấu hình ứng dụng như: port, database, connection string,...**

│ │ ├───static: **thư mục chứa các file tĩnh như css, js, images**

│ │ │ form-stylesheet.css

│ │ │ grades-stylesheet.css

│ │ └───templates: **thư mục chứa các file html được render**

**bởi Spring theo server -side**

│ │ form.html

│ │ grades.html

│ └───test

└───target: **thư mục build ra các file class và jar của maven**

* **Theo Gradle:**

│ .gitignore

│ build.gradle

│ gradlew

│ gradlew.bat

│ HELP.md

│ settings.gradle

├───gradle

└───src

├───main

│ ├───java

│ │ └───com

│ │ └───example

│ │ └───demo

│ │ DemoApplication.java

│ │

│ └───resources

│ │ application.properties

│ ├───static

│ └───templates

└───test

### **i)Tổ chức chương trình**

* **Tổ chức theo mô hình 3 tầng:**
* Presentation Layer (Web Layer): được tổ chức theo mô hình MVC, @Controller (controller): nhận dữ liệu từ Service Layer, xử lý và trả về kết quả cho người dùng
* Business Logic Layer (Service Layer): @Service: Đảm nhận xử lý các logic nghiệp vụ, trả về kết quả cho Web Layer dưới dạng DTO
* Data Access Layer (Persistence Layer, Repository Layer): Repository or DAO, @Repository (annotation): là lớp duy nhất có thể truy cập vào database thực hiện CRUD dữ liệu
* **Mô hình MVC (Spring Boot Web App):** cho tầng Presentation Layer trong mô hình 3 tầng
* View: những gì mà người dùng nhìn thấy
* Model: data cần bởi view
* Controller: kết hợp Model và View với nhau
* **Ngoài ra còn có các thư mục:**
* DTOs (data transfer object): mục đích đóng gói, chuyển đổi dữ liệu giữa server - client, giảm bớt các thông tin không cần thiết
* Entities (Domain Model): đại diện cho các đối tượng tương ứng với table trong DB

### **j)Annotations**

* **Annotations:** Spring Boot sử dụng annotations để cung cấp thông tin bổ sung về một program.
* Để phục vụ cho kiến trúc ở trên, Spring Boot tạo ra 3 annotation là **@Controller** vs **@Service** vs **@Repository** để chúng ta có thể đánh dấu các tầng với nhau

@**Repository**

public interface ContactRepository extends CrudRepository<Contact, Long> {

}

@**Service**

public class ContactServiceImpl implements ContactService {

@Autowired

private ContactRepository contactRepository;

@Override

public Contact getContactById(Long id) {

Optional<Contact> contact = contactRepository.findById(id);

return unwrapContact(contact, id);

}

}

@**Controller**

public class GradeController {

@Autowired

private GradeService gradeService;

@GetMapping("/")

public String getForm(

Model model,

@RequestParam(required = false) Long id) {

Grade grade;

if (id != null) {

grade = gradeService.getGrade(id);

} else {

grade = new Grade();

}

model.addAttribute("grade", grade);

return "form";

}

}

* Ngoài ra, còn có các annotations:

|  |  |
| --- | --- |
| Annotation | Chức năng |
| @Controller | Đánh dấu 1 lớp là trình xử lý yêu cầu |
| @RequestMapping | Sử dụng để ánh xạ các yêu cầu của web |
| @GetMapping(“path”) | Ánh xạ yêu cầu HTTP GET, sử dụng để tạo một Endpoint |
| @PostMapping(“path”) | Ánh xạ các yêu cầu của HTTP POST, sử dụng để tạo một Endpoint |
| @PutMapping | Ánh xạ yêu cầu HTTP PUT, sử dụng tạo Endpoint create hoặc update |
| @DeleteMapping | Ánh xạ yêu cầu HTTP DELETE, sử dụng tạo Endpoint delete |
| @RequestParam | Được sử dụng để trích xuất các tham số từ URL, phù hợp cho ứng dụng web, có thể chỉ định giá trị mặc định nếu tham số không có trong URL |
| @PathVariable | Sử dụng để trích xuất các giá trị từ URL, phù hợp cho dịch vụ web RESTful, trong đó URL chứa biến đường dẫn |
| @Autowired | Báo cho Spring biết tự động tìm và inject bean phù hợp vào vị trí đặt annotation |

### **k)Cách chia sẻ dữ liệu giữa controller và view**

* **Controller đến View**

@GetMapping("/")

public String getForm(

Model model, **// Model parameters đóng vai trò như thùng chứa lưu giữ dữ liệu**

@RequestParam(required = false) Long id) {

Grade grade;

if (id != null) {

grade = gradeService.getGrade(id);

} else {

grade = new Grade();

}

model.addAttribute("grade", grade); **// Sử dụng addAttribute để thêm dữ liệu vào thùng chứa model, view có thể truy cập value thông qua tên key (ở đây là grade)**

return "form";

}

* **View đến Controller**

@PostMapping("/handleSubmit")

public String submitForm(

**Grade grade, // Controller sẽ nhận đối tượng grade từ view khi submit form, ban đầu submitForm sẽ gán grade = new Grade(), sau đó sẽ gán grade bằng value từ view**

) {

gradeService.saveGrade(grade);

**return "redirect:/grades"; // Redirect đến RequestMapping tương ứng với path**

}

### **l)Thymeleaf Expression**

* Cách phổ biến để truy cập dữ liệu (trong **View**) gửi từ **Controller** là sử dụng **Thymeleaf**
* Thêm dependency trong pom.xml (Maven)

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

* Một số **Thymeleaf Expression**:

|  |  |
| --- | --- |
| Cú pháp | Chức năng |
| ${...} | Thực hiện trên một attribute model theo một cách nào đó |
| \*{field} | Chọn một trường từ một đối tượng được liên kết trước đó |
| @{expression} | Tạo URL tương đối dựa trên biểu thức |
| #{expression} | Lấy thông tin từ tệp tin ngôn ngữ (properties file) |
| <th:block> | Tạo 1 khối chứa các phần tử HTML |
| <th:fragment> | Định nghĩa một đoạn mã có thể được sử dụng lại |
| th:field | Liên kết một trường biểu mẫu với một trường đối tượng |
| "${#class.method(target, other params)}" | Thực thi một phương thức utility trên một thuộc tính mô hình |
| th:text | Hiển thị kết quả của một biểu thức dưới dạng văn bản |
| th:utext | Thêm nội dung văn bản không được xử lý HTML vào một phần tử HTML |
| th:if, th:unless | Hiển thị một phần tử nếu một điều kiện if/else |
| th:unless | Hiển thị một phần tử nếu một điều kiện là sai |
| th:switch - th:case | Hiển thị trường hợp phù hợp |
| th:each=”entry: ${collection}” | Vòng lặp tạo ra nhiều phần tử html, như được nhập trong một collection |
| th:object | Liên kết một phần tử html với một đối tượng |
| th:with | Định nghĩa một biến mới trong môi trường hiện tại |
| th:action | Chỉ định một Endpoint - sử dụng biểu thức Link - nơi dữ liệu mẫu được gửi |
| th:href, th:src | Sử dụng Link để chỉ định một Endpoint Url hoặc dẫn đến tài nguyên bên ngoài, th:src đặt giá trị cho thuộc tính src |
| th:attr | Đặt giá trị thuộc tính HTML |
| th:class, th:style | Đặt giá trị cho thuộc tính class và style |

* Cấu hình thymeleaf: (file application.properties)

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

### **m)Lombok (Entity/Model)**

* Là một thư viện Java giúp sinh các mã getter và setter tự động, hỗ trợ sinh các hàm khởi tạo (constructor) với tham số, hoặc không có tham số
* Annotation**:**
  + @Getter và @Setter: để thay thế các phương thức getter và setter
  + @NoAgrsConstructor: thay thế phương thức khởi tạo động không có tham số
  + @AllArgsConstructor: thay thế phương thức khởi tạo động có tất cả các tham số
  + **Sử dụng lombok**
* Để sử dụng Lombok cần add thêm thư viện lombok và cài thêm extension lombok trong editor để không báo lỗi: **Lombok Annotations Support for VScode**

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.26</version>

<scope>provided</scope>

</dependency>

### **n)Data Validation**

* Để sử dụng ta cần cài thêm thư viện

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

|  |  |
| --- | --- |
| Cú pháp | Chức năng |
| @NotBlank | Ít nhất một ký tự không phải khoảng trắng |
| @Min | Không thể nhỏ hơn mức tối thiểu |
| @Max | Không thể vượt qua mức tối đa |
| @NotEmpty | Không thể để giá trị null hoặc rỗng |
| @Email | Định dạng địa chỉ email |
| @Size | Kích thước phải nằm giữa |
| @AssertTrue | Phải bằng True |
| @Past | Trong quá khứ |
| @Future | Trong tương lai |

### **o)Spring Boot Web API**

* **Annotation**

|  |  |
| --- | --- |
| Cú pháp |  |
| @ResponseBody | Chuyển 1 đối tượng trả về thành JSON |
| @RequestBody | Chuyển 1 đối tượng nhận được thành JSON |
| @RestController | Kết hợp @Controller + @ResponseBody |
| @ResponseEntity | Cho phép trả về dữ liệu và status code |
| @RequestParam | Sử dụng để ánh xạ các yêu cầu của web thông qua param |
| @PathVariable | Trích xuất một biến từ đường dẫn được cung cấp |
| @JsonFormat | Cấu hình các thuộc tính được trả về với pattern và shape  VD: @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy hh:mm:ss") |
| @Valid | Validate field |

* Cấu hình Open API
* Open Api là một định dạng chuẩn tiêu chuẩn để giúp xác định các khả năng của Rest API, giúp người dùng hiểu cách sử dụng API

## **2. Khái niệm IOC, DI (Dependency Injection) và Spring Bean**

## **2.1. IOC:**

### **a) Khái niệm**

**-** Inversion of Control (IoC - Đảo ngược điều khiển) là một nguyên lý thiết kế trong công nghệ phần mềm trong đó các thành phần nó dựa vào để làm việc bị đảo ngược quyền điều khiển khi so sánh với lập trình hướng thủ tục truyền thống.

- Khi áp dụng cho các đối tượng lớp (dịch vụ) có thể gọi nó là Dependency inversion (đảo ngược phụ thuộc), để diễn giải trước tiên cần nắm rõ khái niệm Dependency (phụ thuộc)

### **b)Thiết kế truyền thống - tham chiếu trực tiếp đến Dependency**

* Có lớp class A có sử dụng một chức năng (gọi hàm nào đó) của class B, lớp class B lại tham chiếu và gọi các chức năng có trong class C. Ta thấy class A dựa vào class B để hoạt động, class B dựa vào class C. Nếu vậy khi thiết kế theo cách thông thường, viết code thì class A có tham chiếu trực tiếp (cứng) đến class B và trong class B có tham chiếu đến class C

### **c)Thiết kế theo cách đảo ngược phụ thuộc Inverse Dependency**

* Cách viết code này, ở thời điểm thực thi thì class A vẫn gọi được hàm có class B, class B vẫn gọi hàm có class C nghĩa là kết quả không đổi. Tuy nhiên, khi thiết kế ở thời điểm viết code (trong code) class A không tham chiếu trực tiếp đến class B mà nó lại sử dụng interface (hoặc lớp abstract) là classB triển khai. Điều này dẫn tới sự phụ thuộc lỏng lẻo giữa classA và classB
* Khi thực thi, classB có thể được thay thế bởi bất kỳ lớp nào triển khai từ giao diện interface B, classB cụ thể mà classA sử dụng được quyết định và điều khiển bởi interface B (điều này có nghĩa tại sao gọi là đảo ngược phụ thuộc)

## **2.2. DI (Dependency Injection)**

### **a)Khái niệm**

* Dependency injection (DI) là một kỹ thuật lập trình giúp tách một class độc lập với các biến phụ thuộc. Với lập trình hướng đối tượng, chúng ta hầu như luôn phải làm việc với rất nhiều class trong một chương trình. Các class được liên kết với nhau theo một mối quan hệ nào đó. Dependency là một loại quan hệ giữa 2 class mà trong đó một class hoạt động độc lập và class còn lại phụ thuộc bởi class kia.

### **b)Các loại DI**

* Dependency Injection là một Design Pattern hay hiểu đơn giản là một phương pháp lập trình Là một thiết kế để hiệu quả cao hơn khi code
* Vấn đề:
* Các class không nên phụ thuộc vào các kế thừa cấp thấp
* Nên phụ thuộc vào Abstraction (lớp trừu tượng) DI là việc các Object nên phụ thuộc vào các Abstract class và thể hiện chi tiết của nó sẽ được Inject vào đối tượng lúc Runtime
* Có 3 cách để Inject Dependency vào một đối tượng:
* Constructor Injection
* Setter Injection
* Interface Injection
* Thông thường, chúng ta chỉ thường gặp ba loại Dependency Injection sau:
* Constructor injection: Các dependency (biến phụ thuộc) được cung cấp thông qua constructor (hàm tạo lớp).
* Setter injection: Các dependency (biến phụ thuộc) sẽ được truyền vào 1 class thông qua các setter method (hàm setter).
* Interface injection: Dependency sẽ cung cấp một Interface, trong đó có chứa hàm có tên là Inject. Các client phải triển khai một Interface mà có một setter method dành cho việc nhận dependency và truyền nó vào class thông qua việc gọi hàm Inject của Interface đó.

## **2.3. Spring Bean**

### **a)Khái niệm**

* Spring Beans chính là những Java Object mà từ đó tạo nên khung sườn của một Spring application.Chúng được cài đặt, lắp ráp và quản lý bởi Spring IoC container. Những bean này được tạo ra bởi configuration metadata được cung cấp từ container, ví dụ, trong tag nằm trong file XML.
* Các bean được define trong spring framework là singleton bean. Có một thuộc tính trong bean với tên là “singleton” nếu được gán giá trị là true thì bean đó sẽ trở thành singleton, nếu là false thì bean đó sẽ trở thành prototype bean. Mặc định nếu không được định nghĩa giá trị của nó sẽ là true. Vì thế tất cả các bean trong spring framework mặc định sẽ là singleton bean.

### **b)Các bean scope trong spring**

* Có 5 scope được định nghĩa cho Spring Bean:
* **Singleton:** Chỉ duy nhất một thể hiện của bean sẽ được tạo cho mỗi container. Đây là scope mặc định cho spring bean. Khi sử dụng scope này cần chắc chắn rằng các bean không có các biến/thuộc tính được share.
* **Prototype:** Một thể hiện của bean sẽ được tạo cho mỗi lần được yêu cầu(request)
* **Request:** giống với prototype scope, tuy nhiên nó dùng cho ứng dụng web, một thể hiện của bean sẽ được tạo cho mỗi HTTP request.
* **Session:** Mỗi thể hiện của bean sẽ được tạo cho mỗi HTTP Session
* **Global-Session:** Được sử dụng để tạo global sesion bean cho các ứng dụng Portlet.
* Trong 5 scope trên thì 3 scope cuối chỉ dùng trong ứng dụng web.

### **c)Vòng đời của Bean**

* Vòng đời của bean bao gồm các bước sau:
* **Bean Definition :** khởi tạo bean thông qua sử dụng Annotation hoặc XML
* **Bean Instantiation :** Spring khởi tạo các đối tượng Bean giống như khởi tạo đối tượng Java thông thường và đưa nó vào ApplicationContext
* **Populating Bean properties :** Spring thực hiện scan các bean thực thi các Aware interfaces và thực hiện set các giá vào các property như id, scope và giá trị mặc định như khai báo của bean đấy
* **Pre-Initialization :** Các phương thức postProcessBeforeInitialization() bắt đầu thực thi và @PostConstruct thực thi sau ngay nó
* **AfterPropertiesSet** : Spring thực thi các phương thức afterPropertiesSet() của beans mà có implement InitializingBean

### **d)Làm sao để cung cấp configuration metadata cho Spring Container?**

* Có ba cách để cũng cấp configuration metadata cho Spring container.
* Thông qua XML configuration file
* Thông qua Annotation-based configuration
* Thông qua Java-based configuration

### **e)Làm sao để định nghĩa Scope của Bean**

* Khi định nghĩa một bean trong Spring, chúng ta còn phải định nghĩa scope của bean. Việc định nghĩa scope có thể thực hiện thông qua việc sử dụng thuộc tính tên là “scope” khi định nghĩa. Lấy ví dụ, khi bean phải tạo mới mỗi lần cần sử dụng, thuộc tính scope sẽ là “prototype”. Mặt khác, khi bean luôn luôn trả về một instance giống nhau khi sử dụng, thuộc tính scope sẽ là “singleton”.

### **f)Bean autoWired là gì?**

* Spring container có khả năng autowired quan hệ giữa các bean có mối quan hệ hợp tác với nhau. Spring sẽ giải quyết các mối quan hệ hợp tác bằng cách xem xét nội dung của BeanFactory với các tag và .

### **g)Các mode auto wiring**

* Autowiring có năm mode được sử dụng để hướng dẫn Spring container làm sao autowiring giải quyết dependency injection.
* **no:** đây là default setting, bean tham chiếu sẽ được reference rõ ràng khi khai báo.
* **byName:** khi autowiring byName, Spring container sẽ cố gắng match giá trị được khai báo trong bean với tên tương tự trong configuration file.
* **byType:** khi autowiring bằng data type, Spring container sẽ cố gắng match chính xác với tên của bean trong configuration file. Nếu có nhiều hơn một, fatal exception sẽ được throw ra.
* **constructor:** mode này tương tự với byType, nhưng type sẽ apply lên các constructor argument. Nếu không có kết quả nào phù hợp, thì fata exception cũng được throw ra.
* **autodetect:** Spring sẽ cố gắng wire bằng cách sử dụng constructor, nếu không có nó sẽ fallback xuống sử dụng byType.

## **3. Spring Data JPA cơ bản và nâng cao**

* Spring Data là một module của Spring Framework. Mục đích của Spring Data JPA là giảm thiểu việc thực hiện quá nhiều bước để có thể implement được JPA. Spring Data JPA là một phần của Spring Data và nó hỗ trợ Hibernate 5, OpenJPA 2.4 và EclipseLink 2.6.1.

### **Kiến thức cơ bản của JPA**

### **JPA là gì**

* Là một đặc tả Java cho việc ánh xạ giữa các đối tượng Java với cơ sở dữ liệu quan hệ sử dụng công nghệ phổ biến là ORM (Object Relational Mapping).
* JPA cung cấp đầy đủ các công cụ cho phép thao tác với cơ sở dữ liệu một cách đơn giản và nhanh chóng.
* JPA hoạt động như một cầu nối giữa các table/ các mối quan hệ giữa các table trong database và các class/ mối quan hệ giữa các object. các loại JPA cơ bản
* JPA cũng cung cấp các tính năng quan trọng như quản lý định danh và quan hệ giữa các đối tượng, các chiến lược tải dữ liệu để tải dữ liệu giảm thiểu số lượng truy cập cơ sở dữ liệu, và đảm bảo tính nhất quán dữ liệu trong các giao dịch đa luồng. JPA được tích hợp với các framework như Spring và Hibernate và được hỗ trợ bởi các hệ quản trị cơ sở dữ liệu phổ biến như MySQL, Oracle và PostgreSQL

### **ORM là gì**

* ORM là viết tắt của “Object Relational Mapping”, là quá trình chuyển đổi dữ liệu từ ngôn ngữ hướng đối tượng sang Database quan hệ và ngược lại.
* ORM có khả năng xử lý các thao tác của nhiều loại cơ sở dữ liệu khác nhau một cách dễ dàng mà không quan tâm đến loại Database sử dụng hay loại thao tác sử dụng.

### **Lợi ích của JPA**

* Đơn giản hóa công nghệ cho tầng persistence (tầng dữ liệu).
* Không phụ thuộc vào các framework ORM.
* Có nhiều nhà cung cấp hỗ trợ cài đặt JPA.
* Dữ liệu có thể được lưu trữ thông qua việc ORM.

### **Tại sao nên dùng JPA**

* Viết ít code hơn.
* Performance tốt.
* Độc lập về database.
* Không phải làm việc với SQL.
* Hỗ trợ cấu hình triển khai bằng annotation và xml.
* Có nhiều framework ORM miễn phí hỗ trợ có thể dùng để phát triển nhiều loại ứng dụng khác nhau.
* JPA là một đặc tả đã được chuẩn hóa và là một thành phần trong đặc tả EJB 3.
* Dễ dàng chuyển từ một ORM này sang một ORM khác. Ví dụ từ iBatis sang Hibernate.

### **Kiến trúc của JPA**

* JPA sử dụng metadata để ánh xạ các đối tượng persistence với các bảng trong cơ sở dữ liệu. JPA hỗ trợ SQL như là một ngôn ngữ truy vấn để dễ dàng xử lý các truy vấn cơ sở dữ liệu. Ngôn ngữ truy vấn JPA có thể dùng thực thi truy vấn tĩnh và truy vấn động.
* JPA bao gồm ba thành phần chính là: Entity, EntityManager, và EntityManagerFactory. Ngoài ra còn có, EntityTransaction, Persistence, Query.

### **Entity**:

* đối tượng java mà Spring Boot JPA có thể quản lý
* @Table: Chỉ định bảng cho thực thể được chú thích
* Name: Tên bảng
* uniqueConstraints: Các ràng buộc duy nhất được đặt trên bảng
* @Column: ánh xạ một trường vào một cột trong bảng
* Name: Tên cột
* nullable: boolean xác định xem cột có thể chấp nhận giá trị rỗng hay không
* unique: boolean xác định xem cột có từ chối trùng lặp hay không

### **Primary Key:**

* cột xác định duy nhất từng bản ghi bên trong một bảng
* @Id: áp dụng trên trường ánh xạ tới khóa chính

### **Foreign Key:**

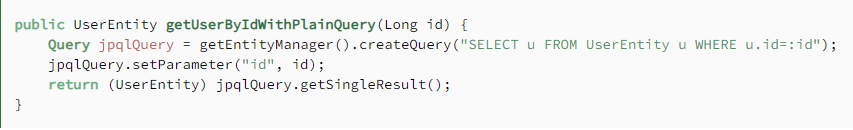
* tham chiếu khóa chính của bảng khác
* Khi áp dụng, hãy đặt cột khóa ngoại trong bảng không thể tồn tại nếu không có cột kia
* @JoinColumn: chỉ định một cột để tham gia liên kết thực thể
* Name: tên của cột
* referencedColumnName: tên của cột được tham chiếu bởi cột khóa ngoại
* @ManyToOne: biểu diễn mối quan hệ nhiều-một
* @OneToMany: biểu diễn mối quan hệ 1-nhiều

Có ba loại Query JPA cơ bản:

* Query, được viết bằng cú pháp Java Persistence Query Language (JPQL)
* NativeQuery, được viết bằng cú pháp SQL đơn giản
* Criteria API Query, được xây dựng theo chương trình thông qua các phương pháp khác nhau

### **Query**:

Có cú pháp tương tự như SQL và thường được sử dụng để thực hiện các thao tác CRUD:

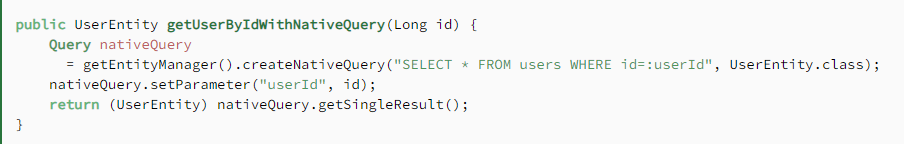


Query này truy xuất bản ghi phù hợp từ bảng người dùng và cũng ánh xạ nó tới đối tượng UserEntity.

Có 2 loại Query phụ bổ sung: TypedQuery và NamedQuery

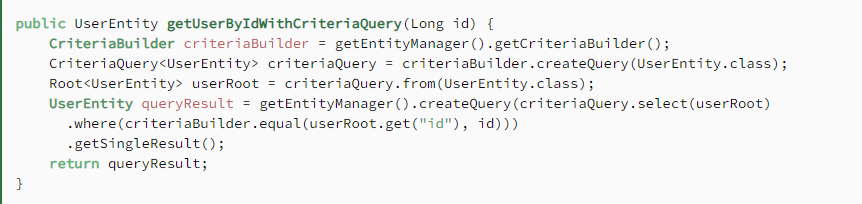
### **NativeQuery**

NativeQuery chỉ đơn giản là một truy vấn SQL. Những điều này cho phép chúng ta giải phóng toàn bộ sức mạnh của cơ sở dữ liệu của mình, vì chúng ta có thể sử dụng các tính năng độc quyền không có sẵn trong cú pháp hạn chế của JPQL.

Điều này đi kèm với một tổn thất. Mất tính di động cơ sở dữ liệu của ứng dụng với NativeQuery vì nhà cung cấp JPA không thể trừu tượng hóa các chi tiết cụ thể từ việc triển khai cơ sở dữ liệu hoặc nhà cung cấp nữa.

### **Criteria API Query**

Là các truy vấn an toàn kiểu, được xây dựng theo chương trình – hơi giống với các truy vấn JPQL theo cú pháp:



Việc sử dụng các Criteria API Query có thể gây khó khăn ngay từ đầu, nhưng chúng có thể là một lựa chọn tuyệt vời khi chúng ta cần thêm các phần tử truy vấn động hoặc khi được kết hợp với Siêu mô hình JPA.

### **Các bước để sử dụng JPA**

**Bước 1:** Thêm dependency vào file pom.xml:

* Đầu tiên bạn muốn sử dụng được Spring Data JPA thì bạn cần phải thêm dependency của Spring Data JPA vào file pom.xml
* Dependency spring-boot-starter-data-jpa cung cấp một số dependency liên quan đến JPA, bao gồm Hibernate, JDBC và các dependency khác.
* Lưu ý rằng bạn cũng cần phải thêm các dependency liên quan đến cơ sở dữ liệu mà bạn đang sử dụng (ví dụ như MySQL connector).

Bước 2: Cấu hình kết nối cơ sở dữ liệu trong file application.properties.

* Spring Boot cung cấp một cách để cấu hình database thông qua file application.properties hoặc application.yml. Bạn có thể cấu hình database bằng cách thêm các thuộc tính trong file này. Ví dụ:

spring.datasource.url=jdbc:mysql://localhost:3306/exampledb

spring.datasource.username=dbuser

spring.datasource.password=dbpass

spring.jpa.hibernate.ddl-auto=update

Trong đó:

* spring.datasource.url: địa chỉ kết nối tới database.
* spring.datasource.username: tên đăng nhập để truy cập vào database.
* spring.datasource.password: mật khẩu để truy cập vào database.
* spring.datasource.driver-class-name: tên của driver để kết nối tới database. Trong trường hợp này, chúng ta sử dụng driver của MySQL.

**Bước 3**: Tạo các Entity để ánh xạ đến các bảng trong cơ sở dữ liệu.

* Một entity class là một lớp Java đại diện cho một bảng trong database. Trong entity class, các trường của lớp tương ứng với các cột trong bảng. Bạn có thể sử dụng annotation để chỉ định các thuộc tính của entity class.
* Ví dụ, để tạo một entity class để ánh xạ với bảng users trong database, bạn có thể sử dụng đoạn mã sau:

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "name")

private String name;

@Column(name = "age")

private int age;

// Getters and setters

}

* @Entity: annotation này được sử dụng để đánh dấu lớp là một entity class.
* @Table(name = "users"): annotation này được sử dụng để chỉ định tên của bảng trong database.
* @Id: annotation này được sử dụng để chỉ định trường id của entity class.
* @GeneratedValue(strategy = GenerationType.AUTO): annotation này được sử dụng để chỉ định cách sinh ra giá trị cho trường id. Trong trường hợp này, giá trị của trường id sẽ được sinh tự động bởi hệ thống.

**Bước 4**: Tạo Repository interface để thao tác với database.

* Repository interface là một interface trong Spring Data JPA được sử dụng để thao tác với database. Spring Data JPA sẽ tạo ra các method để thao tác với database dựa trên tên method và kiểu dữ liệu của tham số truyền vào.
* Ví dụ, để tạo một repository để thao tác với entity class User, bạn có thể sử dụng đoạn mã sau:

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByLastName(String lastName);

User findByEmail(String email);

}

Trong đó:

* @Repository: annotation này được sử dụng để đánh dấu interface là một repository.
* JpaRepository: đây là một interface trong Spring Data JPA cung cấp các method để thao tác với database. Interface này có hai tham số: kiểu entity và kiểu của trường id của entity.
* List<User> findByLastName(String lastName): method này được tạo ra tự động bởi Spring Data JPA để tìm kiếm các user có lastName bằng với tham số truyền vào.
* User findByEmail(String email): method này được tạo ra tự động bởi Spring Data JPA để tìm kiếm một user có email bằng với tham số truyền vào.

**Bước 5**: Sử dụng Repository để thao tác với database.

Sau khi đã tạo Repository interface, bạn có thể sử dụng nó để thao tác với database trong ứng dụng. Ví dụ, để tạo một user mới, bạn có thể sử dụng đoạn mã sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @Service  public class UserService {    @Autowired  private UserRepository userRepository;    public User createUser(User user) {  return userRepository.save(user);  }  } |

Trong đó:

* @Service: annotation này được sử dụng để đánh dấu lớp là một service.
* @Autowired: annotation này được sử dụng để chú thích cho Spring biết rằng nó phải inject một instance của UserRepository vào biến userRepository.
* userRepository.save(user): method này được tạo ra tự động bởi Spring Data JPA để lưu một user mới vào database.

**Bước 6**: Sử dụng các phương thức được cung cấp bởi Spring Data JPA để thực hiện các thao tác CRUD với cơ sở dữ liệu:

* ví dụ:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public List<User> getAllUsers() {

return userRepository.findAll();

}

public User createUser(User newUser) {

return userRepository.save(newUser);

}

public User updateUser(User userToUpdate) {

return userRepository.save(userToUpdate);

}

public void deleteUser(Long userId) {

userRepository.deleteById(userId);

}

}

Lưu ý: @Autowired để Spring Boot inject đối tượng UserRepository vào UserService.

Trong đó, getAllUsers() trả về danh sách tất cả các user, createUser() tạo một user mới, updateUser() cập nhật thông tin của một user, và deleteUser() xóa một user theo id.

### Các tính năng nâng cao

* Spring Data JPA cung cấp các tính năng nâng cao như lọc dữ liệu, phân trang, đặt tên các truy vấn tùy chỉnh và sử dụng các annotation để định nghĩa các mối quan hệ giữa các đối tượng.
* **Pagination**: Spring Data JPA cung cấp tính năng phân trang dữ liệu giúp cho việc hiển thị các bản ghi trên các trang web trở nên dễ dàng hơn. Chúng ta có thể sử dụng phương thức findAll của JpaRepository và truyền tham số PageRequest để xác định số trang và số lượng bản ghi trên mỗi trang.

Ví dụ:

public Page<User> getAllUsers(int pageNumber, int pageSize) {

Pageable pageable = PageRequest.of(pageNumber, pageSize);

return userRepository.findAll(pageable);

}

* Trong đó, getUsersPage() trả về một trang dữ liệu của các **user**, với **pageNumber** là số trang cần lấy và **pageSize** là số lượng user trên mỗi trang.
* **Sorting**: Spring Data JPA cho phép sắp xếp các bản ghi theo một trường cụ thể hoặc nhiều trường. Chúng ta có thể sử dụng phương thức findAll của JpaRepository và truyền tham số Sort để xác định thứ tự sắp xếp và trường sắp xếp.

Ví dụ:

public List<User> getAllUsersSorted() {

Sort sort = Sort.by(Sort.Direction.ASC, "name", "age");

return userRepository.findAll(sort);

}

* **Query Methods**: Spring Data JPA cung cấp các phương thức truy vấn dựa trên các quy ước tên phương thức. Chúng ta có thể đặt tên phương thức với các tiền tố và hậu tố được định nghĩa sẵn như findFirstBy, findTopBy, findBy, countBy và nhiều hơn nữa.

Ví dụ:

public List<User> findByAge(int age);

public User findFirstByOrderByName();

public long countByName(String name);

* **Native Queries**: Spring Data JPA cũng hỗ trợ việc viết các truy vấn tùy chỉnh sử dụng ngôn ngữ SQL hoặc các ngôn ngữ truy vấn khác. Chúng ta có thể sử dụng Annotation @Query để định nghĩa các truy vấn tùy chỉnh.

Ví dụ:

@Query(value = "SELECT \* FROM users WHERE age > :age", nativeQuery = true)

public List<User> findAllByAgeGreaterThan(int age);

* **Specifications**: Specifications là một tính năng nâng cao của Spring Data JPA cho phép xây dựng các câu truy vấn phức tạp sử dụng các điều kiện và phép toán logic. Chúng ta có thể sử dụng các API của Criteria API để xây dựng các điều kiện và sử dụng Specification API để kết hợp các điều kiện lại với nhau.

Ví dụ:

public static Specification<User> hasAgeGreaterThan(int age) {

return (user, cq, cb) -> cb.greaterThan(user.get("age"), age);

}

public List<User> findAll(Specification<User> spec)

## **4.Schedule Task, Interceptor và gửi mail**

### **Phần Interceptor:**

- Interceptor là một tính năng mạnh mẽ trong Spring Boot Framework.

- Nó cho phép các phần tử trong ứng dụng Spring Boot thực hiện các hành động cụ thể trước hoặc sau khi các yêu cầu HTTP được xử lý bởi các phương thức xử lý của ứng dụng.

- **Chức năng:**

· Xác thực và phân quyền truy cập

· Kiểm tra và sửa đổi thông tin yêu cầu HTTP

· Ghi log các yêu cầu và phản hồi

· Nén phản hồi

· Thêm thông tin phiên bản, tên miền vào tiêu đề yêu cầu

- **Ưu điểm**

· Tiết kiệm thời gian: Interceptor trong Spring Boot giúp tiết kiệm thời gian bằng cách cho phép xử lý chung các yêu cầu HTTP chung trên toàn bộ ứng dụng, thay vì phải xử lý riêng lẻ cho từng controller hoặc handler.

· Tính linh hoạt: Interceptor trong Spring Boot cho phép xác định các tác vụ được thực hiện trước khi controller hoặc handler được thực thi, giúp cho việc xử lý yêu cầu HTTP trở nên linh hoạt hơn

· Tăng tính bảo mật: Interceptor trong Spring Boot có thể được sử dụng để kiểm soát truy cập và giám sát các hoạt động độc hại, giả mạo hoặc không hợp lệ.

- **Nhược điểm**

· Khó khăn trong việc xử lý lỗi: Interceptor trong Spring Boot không thể trả về kết quả cho client trực tiếp, và do đó việc xử lý lỗi có thể trở nên khó khăn hơn.

· Khó khăn trong việc debug: Vì interceptor được sử dụng để xử lý các yêu cầu HTTP chung, việc debug và xử lý lỗi có thể trở nên phức tạp hơn nếu không được xử lý đúng cách.

### **So sánh Interceptor và Filter**

· **Giống nhau:**

· Hai tính năng quan trọng trong Spring boot được sử dụng để xử lý các yêu cầu HTTP trước khi chúng được xử lý bởi một controller hoặc handler

· **Khác nhau:**

· Vị trí trong chuỗi xử lý: Filter được gọi đầu tiên trong chuỗi xử lý khi một yêu cầu HTTP đến, còn Interceptor được gọi sau Filter và trước khi yêu cầu được xử lý bởi controller hoặc handler.

· Phạm vi ảnh hưởng: Filter áp dụng trên toàn bộ ứng dụng, trong khi Interceptor chỉ áp dụng trên một số requestmapping hoặc handler mapping cụ thể.

· Tính năng: Filter cho phép chúng ta thực hiện các xử lý bổ sung trên request và response, còn Interceptor cho phép kiểm soát truy cập và giám sát các hoạt động độc hại, giả mạo hoặc không hợp lệ trên hệ thống hoặc mạng máy tính.

· Cách đăng ký: Để đăng ký một Filter, chúng ta cần đăng ký nó trong tệp web.xml hoặc sử dụng annotation @WebFilter. Để đăng ký một Interceptor, chúng ta cần tạo một class implement interface HandlerInterceptor và đăng ký nó trong file cấu hình Spring hoặc bằng annotation @Interceptor.

· **Cấu hình Interceptor**

· Để sử dụng Interceptor, bạn cần phải tạo một lớp triển khai của interface HandlerInterceptor của Spring Boot, với ba phương thức sau:

· preHandle(): Phương thức này được gọi trước khi xử lý yêu cầu.

· postHandle(): Phương thức này được gọi sau khi xử lý yêu cầu và trước khi phản hồi được trả về.

· afterCompletion(): Phương thức này được gọi sau khi phản hồi đã được trả về cho người dùng.

### **Phần gửi email:**

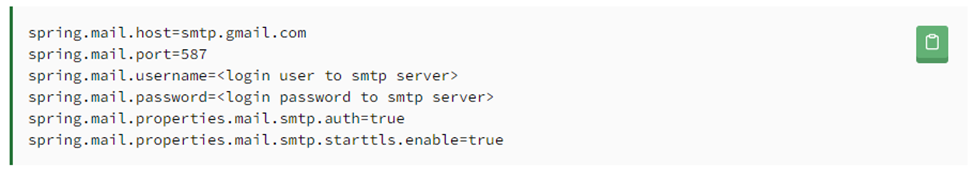
Spring Email:

* Spring Mail là một trong những tính năng quan trọng của Spring Framework, cung cấp các lớp và giao diện cho phép ứng dụng gửi và nhận email một cách dễ dàng. Spring Mail cho phép ứng dụng tạo và gửi email qua SMTP (Simple Mail Transfer Protocol) hoặc gửi email qua các dịch vụ email khác như Gmail, Yahoo Mail và Outlook.
* JavaMailSender: là giao diện chính cho phép tạo và gửi email trong Spring Mail. Nó cung cấp các phương thức để thiết lập các thông tin cần thiết cho việc gửi email, chẳng hạn như tên máy chủ SMTP, tài khoản người gửi, mật khẩu và các thuộc tính khác của email như tiêu đề, nội dung, tệp đính kèm,..
* MimeMessageHelper: là lớp hỗ trợ cho việc tạo và cấu hình các thuộc tính của email như tiêu đề, nội dung, tệp đính kèm
* JavaMailSenderImpl: là lớp cài đặt cho giao diện JavaMailSender, cung cấp các phương thức để gửi email.
* MailMessage :Là một interface đại diện cho một tin nhắn (message) đơn giản. Nó bao gồm các thông tin cơ bản của một email như người gửi, người nhận, tiêu đề (subject) và nội tin nhắn.
* SimpleMailMessage :Đây là một lớp thực hiện (implements) interface MailMessage, được sử dụng để tạo một tin nhắn (message) đơn giản.
* MimeMailMessage:Đây là một lớp thực hiện interface MailMessage, được sử dụng để tạo ra một tin nhắn hỗ trợ MIME.
* MailSender :Đây là một interface ở mức cao nhất (top-level), nó cung cấp các chức năng để gửi một email đơn giản
* MimeMessagePreparator :Interface này cung cấp phương thức callback được gọi khi chuẩn bị một tin nhắn MIME.

### **Mime**

* MIME (Multipurpose Internet Mail Extensions) là một tiêu chuẩn để mã hóa và truyền tải các thông điệp email có định dạng đa phương tiện. Với MIME, các thông điệp email không chỉ chứa văn bản đơn giản, mà còn có thể chứa các tệp đính kèm, hình ảnh, âm thanh, video, các tài liệu định dạng HTML, CSS, JavaScript, và các định dạng khác.

### **Cấu hình trong application.property**

****

* spring.mail.host: Đây là địa chỉ máy chủ SMTP của nhà cung cấp dịch vụ email. spring.mail.port: Đây là cổng kết nối đến máy chủ SMTP.
* spring.mail.username: Đây là tên người dùng của tài khoản email được sử dụng để gửi email.
* spring.mail.password: Đây là mật khẩu của tài khoản email được sử dụng để gửi email.
* spring.mail.properties.mail.smtp.auth: Đây là thuộc tính cấu hình để bật xác thực SMTP. Nếu giá trị của nó là true, ứng dụng sẽ thực hiện xác thực với máy chủ SMTP để gửi email.
* spring.mail.properties.mail.smtp.starttls.enable: Đây là thuộc tính cấu hình để bật chức năng bảo mật STARTTLS. STARTTLS là một cơ chế để bảo vệ email trong quá trình truyền qua mạng bằng cách mã hóa nó. Nếu giá trị của nó là true, ứng dụng sẽ sử dụng chức năng bảo mật STARTTLS khi gửi email.

### **Các hình thức gửi mail**

* gửi mail thông thường
* gửi mail tệp đính kèm
* gửi mail định dạng HTML

## **5. Lambda Expression trong Java**

### **a) Lambda Expression ( Biểu thức Lambda)**

* Là một tính năng mới và quan trọng được thêm vào Java SE 8.
* Cung cấp một cách ngắn gọn và rõ ràng để biểu diễn một phương thức của interface sử dụng một biểu thức
* Lambda expression thực thi phương thức của một Functional Interface. Giúp không cần định nghĩa phương thức lại mà chỉ cần viết code để thực thi phương thức

### **b) Functional Interface**

* Là một interface mà chỉ có một phương thức abstract duy nhất.
* Java cung cấp một annotation @FunctionalInterface, được sử dụng để chỉ định một interface là Functional interface.

### **c) Một số quy tắc khai báo Functional Interface**

* Một Functional Interface hợp lệ chỉ có duy nhất một method trừu tượng.
* Một Functional Interface có thể có các phương thức của lớp java.lang.Object
* Phương thức default and static không phá vỡ quy tắc của Functional interface.
* Một Functional Interface có thể kế thừa từ một Functional Interface khác chỉ khi nó không có bất kỳ phương thức trừu tượng nào
* Các Functional Interface đã được định nghĩa trong Java 8 – Java Predefined-Functional Interfaces được đặt trong gói java.util.function

### **Cú pháp Lambda Expression**

(argument-list) -> {body}

* argument-list: danh sách tham số
* Dấu “->”: được sử dụng để nối danh sách tham số và phần thân của biểu
* thức
* body: chứa biểu thức và những câu lệnh

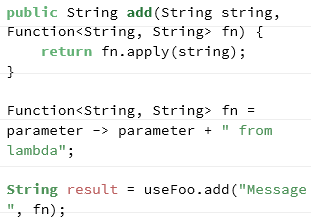
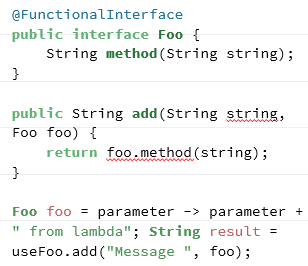
### **e)Lợi ích**

* Thực thi Functional Interface
* Viết ít code hơn

### **f)Lưu ý**

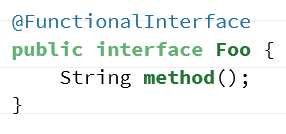
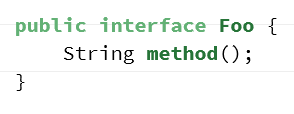
**STANDARD FUNCTIONAL INTERFACE**

* Functional interface, được tổng hợp trong gói java.util.function, cung cấp khá nhiều những target type cho biểu thức Lambda. Với mỗi interface trong đó là Generic và Abstract.



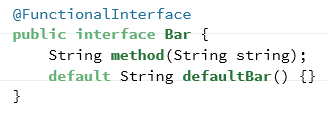
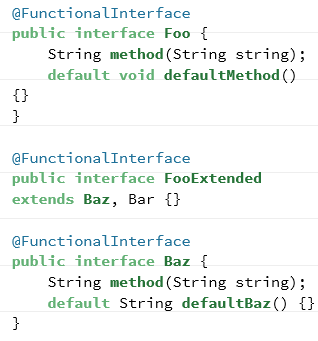
**@FunctinonalInterface ANNOTATION**

* Khi sử dụng annotation @FunctionalInterface, trình biên dịch sẽ thông báo lỗi nếu như có sự thay đổi cấu trúc của một Functional interface.
* Giúp cho các nhà phát triển khác dễ hiểu kiến trúc ứng dụng.



KHÔNG LẠM DỤNG DEFAULT METHOD TRONG FUNCTIONAL INTERFACE

* Có thể thêm phương thức với từ khóa “default” vào Functional interface. Điều này là chấp nhận được, miễn là Functional interface vẫn chỉ chứa duy nhất 1 abstract method.



ĐỪNG CHO RẰNG LAMBDA EXPRESSION LÀ INNER CLASS

* Một điều khác nhau cần lưu ý nhất là: PHẠM VI.
* Khi sử dụng một inner class, nó sẽ tạo ra một phạm vi mới. Có thể ẩn các biến cục bộ khỏi phạm vi kèm theo bằng cách khởi tạo các biến cục bộ mới có cùng tên. Cũng có thể sử dụng từ khóa this bên trong inner class để tham chiếu đến instance của nó.
* Tuy nhiên, biểu thức lambda hoạt động với phạm vi kèm theo. Không thể ẩn các biến khỏi phạm vi kèm theo bên trong phần thân của lambda. Trong trường hợp này, từ khóa this là tham chiếu đến một instance kèm theo.

GIỮ LAMBDA EXPRESSION NGẮN GỌN VÀ DỄ HIỂU

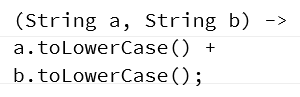
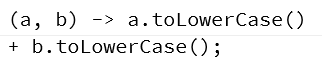
* Nếu có thể, nên viết biểu thức lambda trên 1 dòng thay vì dùng một khối code lớn.
* Chủ yếu là về phong cách code, không ảnh hưởng đến hiệu suất.

TRÁNH VIẾT CÁC KHỐI CODE TRONG THÂN LAMBDA EXPRESSION

* Nếu có khối lượng lớn code, có thể tách phần xử lý ra một hàm riêng, sau đó gọi hàm trong phần thân của Lambda expression.

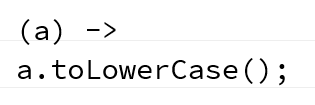
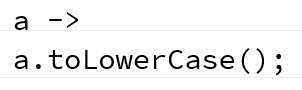
TRÁNH CHỈ ĐỊNH KIỂU CỤ THỂ TRONG LAMBDA EXPRESSION

* Compiler (trình biên dịch) trong hầu hết các trường hợp đều có khả năng tự suy ra kiểu. Vì vậy có thể bỏ qua việc xác định kiểu

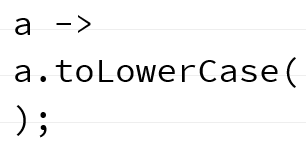
TRÁNH SỬ DỤNG DẤU ()

* Tránh sử dụng cặp dấu () nếu chỉ có 1 tham số duy nhất.
* Điều này giúp code ngắn gọn hơn đôi chút.

TRÁNH SỬ DỤNG MỆNH ĐỀ RETURN VÀ DẤU {}

* Tránh sử dụng từ khóa return nếu chỉ có 1 tham số duy nhất.
* Tránh sử dụng cặp dấu ngoặc nhọn nếu chỉ có 1 câu lệnh hoặc 1 biểu thức.
* Điều này giúp code ngắn gọn hơn đôi chút.

CẨN THẬN KHI SỬ DỤNG BIẾN FINAL

* Việc truy cập một biến không phải là final bên trong các biểu thức lambda sẽ gây ra lỗi thời gian biên dịch, **nhưng điều đó không có nghĩa là chúng ta nên đánh dấu mọi biến mục tiêu là *biến final.***
* Theo khái niệm “ **effectively final**”, trình biên dịch coi mọi biến là *biến final*  miễn là nó chỉ được gán một lần.
* Sẽ an toàn khi sử dụng các biến như vậy bên trong lambda vì trình biên dịch sẽ kiểm soát trạng thái của chúng và gây ra lỗi thời gian biên dịch ngay sau bất kỳ nỗ lực nào để thay đổi chúng.
* Ví dụ: đoạn mã sau sẽ không biên dịch

1. Tổng kết

* Bản chất của Lambda Expression - Biểu thức Lambda trong java là tạo ra một đối tượng nặc danh cho một Functional Interface, và cung cấp thân hàm cho phương thức trừu tượng duy nhất của Functional Interface.
* Functional Interface trong Java là một interface chỉ được chứa một hàm trừu tượng duy nhất

## **6.JSP Template Engine**

### **Khái niệm JSP (JavaServer Pages)**

* Là một công nghệ để phát triển các trang web
* Thành phần JSP là một loại Java Servlet
* Sử dụng JSP, có thể thu thập thông tin đầu vào từ người dùng
* Thẻ JSP có thể được sử dụng cho nhiều mục đích khác nhau

### **Lý do nên sử dụng JSP**

* Hiệu suất tốt
* Được biên dịch trước khi được xử lý
* Sử dụng JSP có thể thu thập thông tin đầu vào từ người dùng
* Các trang JavaServer được xây dụng trên API Java Servlets
* Các trang JSP có thể được sử dụng kết hợp với các servlet xử lý logic nghiệp vụ

### **Ưu điểm của JSP**

* So với Active Server Pages (ASP): phần động được viết bằng Java, không phải Visual Basic
* So với Pure Servlet: thuận tiện hơn khi viết (và sửa đổi!)
* So với phía máy chủ bao gồm SSI: SSI thực sự chỉ dành cho các hành động đơn giản
* So với Javascript: JavaScript có thể tạo HTML động trên máy khách nhưng khó có thể tương tác với máy chủ web
* So với HTML tĩnh: HTML thông thường không thể chứa thông tin động.

### **Thiết lập môi trường**

1. **Thiết lập Java Development Kit**

* **Bước 1:** Tải SDK từ trang Java của Oracle
* **Bước 2:** Thiết lập các biến môi trường PATH và JAVA\_HOME

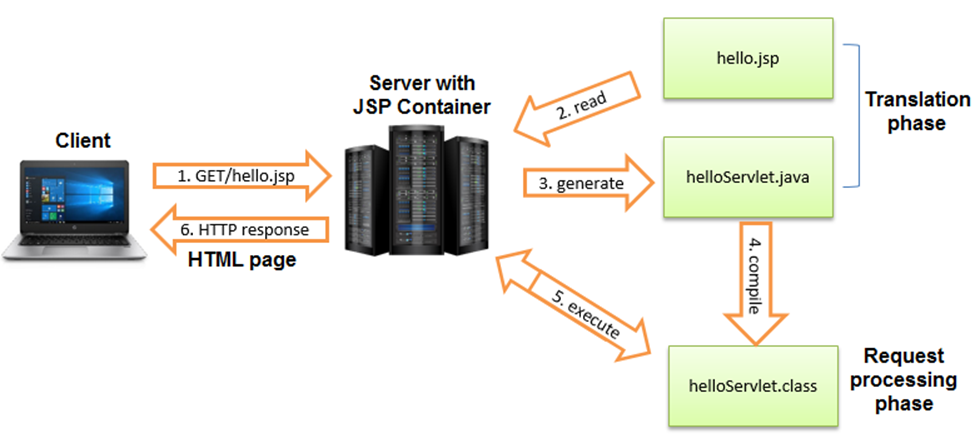
1. **Thiết lập Web Server: Tomcat**

* **Bước 1:** Tải xuống phiên bản mới nhất của Tomcat
* **Bước 2:** Tạo CATALINA\_HOME biến môi trường trỏ đến các vị trí này Tomcat có thể được khởi động bằng cách thực hiện lệnh sau trên Windows:

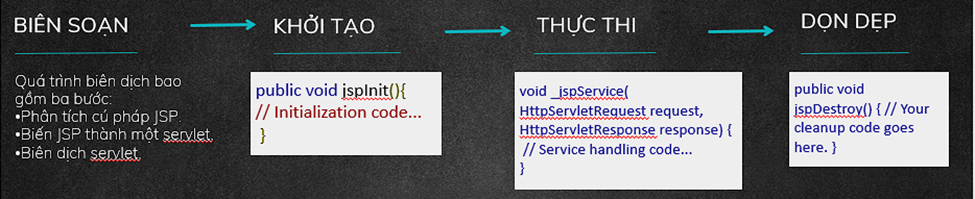
*%CATALINA\_HOME%\bin\startup.bat*

### **Kiến trúc**

### **Luồng hoạt động**

****

### **Vòng đời**

****

### **Cú pháp**

### **Scriptlet**

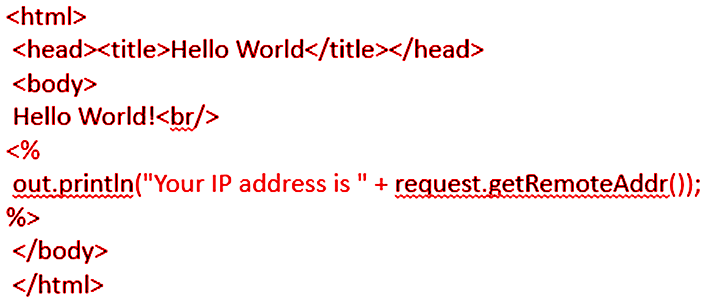
* **Cú pháp: *<% code fragment %>***
* **Viết XML:**

***<jsp:scriptlet>***

***code fragment***

***</jsp:scriptlet>***

* **Ví dụ:**

****

### **Khai báo**

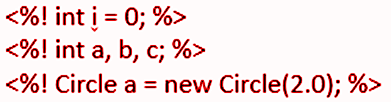
* **Cú pháp: *<%! declaration; [ declaration; ]+ ... %>***
* **Viết XML:**

***<jsp:declaration>***

***code fragment***

***</jsp: declaration >***

* **Ví dụ:**

****

### **Biểu thức**

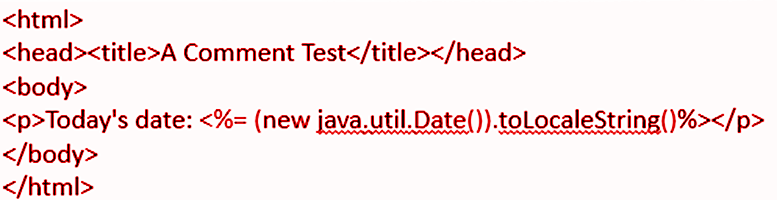
* **Cú pháp: *<%= expression %>***
* **Viết XML:**

***<jsp: expression >***

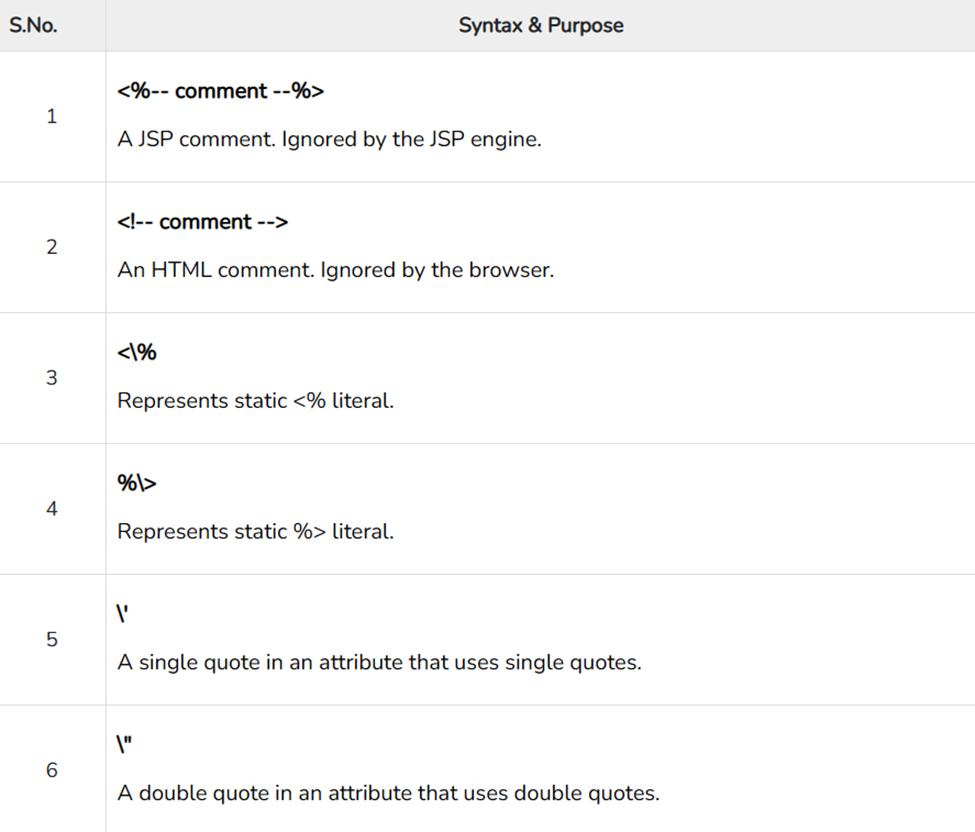
***expression***

***</jsp: expression >***

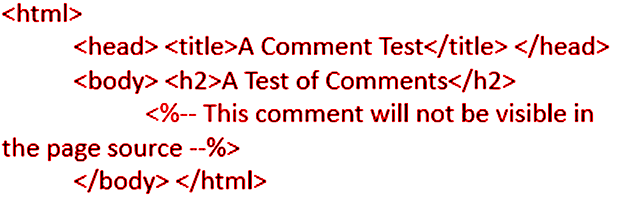
* **Ví dụ:**

****

### **Comment**

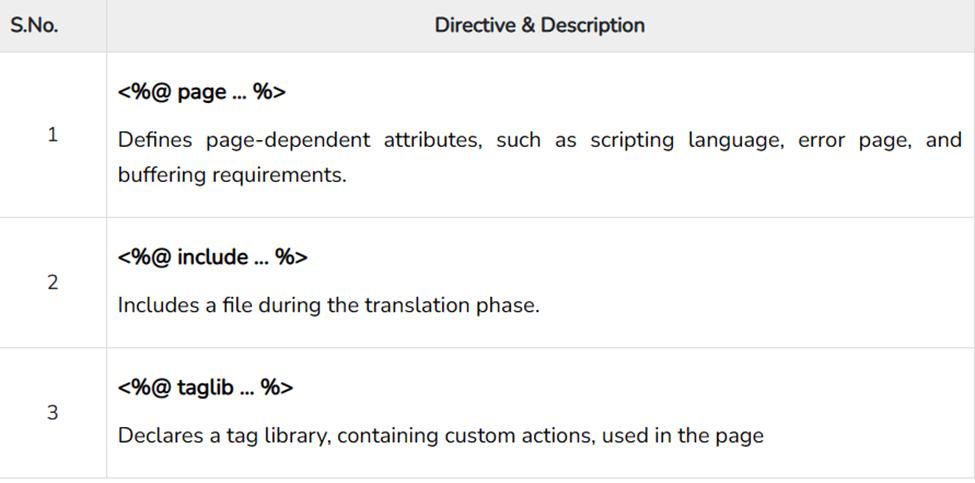
****

* **Cú pháp: *<%-- This is comment --%>***
* **Ví dụ:**

****

### **Directive**

* **Cú pháp: *<%@ directive attribute="value" %>***

****

### **Actions**

* **Cú pháp: *<jsp:action\_name attribute="value" />***

|  |  |
| --- | --- |
| **Syntax** | **Purpose** |
| **jsp:include** | Bao gồm 1 tệp tại thời điểm trang được yêu cầu |
| **jsp:useBean** | Tìm hoặc khởi tạo 1 JavaBean |
| **jsp:setProperty** | Đặt thuộc tính của JavaBean |
| **jsp:getProperty** | Chèn thuộc tính của 1 JavaBean vào đầu ra |
| **jsp:forward** | Chuyển tiếp người yêu cầu đến 1 trang mới |
| **jsp:plugin** | Tạo mã dành riêng cho trình duyệt tạo thẻ OBJECT hoặc EMBED cho plugin Java |
| **jsp:element** | Định nghĩa động các phần tử XML |
| **jsp:attribute** | Định nghĩa thuộc tính của phần tử XML được xác định động |
| **jsp:body** | Định nghĩa phần thân của phần tử XML được xác định động |
| **jsp:text** | Được sử dụng để viết văn bản mẫu trong các trang và tài liệu JSP |

### **Implicit Objects**

|  |  |
| --- | --- |
| **Object** | **Description** |
| **request** | Đây là đối tượng HttpServletRequest được liên kết với request |
| **response** | Đây là đối tượng HttpServletReponse được liên kết với phản hồi cho client |
| **out** | Đây là đối tượng PrintWriter được sử dụng để gửi output đến client |
| **session** | Đây là đối tượng HttpSession được liên kết với request |
| **application** | Đây là đối tượng ServletContext được liên kết với ngữ cảnh ứng dụng |
| **config** | Đây là đối tượng ServletConfig được liên kết với trang |
| **pageContent** | Điều này gói gọn việc sử dụng các tính năng dành riêng cho máy chủ như JspWriters hiệu suất cao hơn |
| **page** | Đây chỉ đơn giản là một từ đồng nghĩa với điều này, và được sử dụng để gọi các phương thức được xác định bởi lớp servlet đã dịch |
| **Exception** | Đối tượng Exception cho phép dữ liệu ngoại lệ được truy cập bới JSP được chỉ định |

### **Control-Flow Statements**

Bạn có thể sử dụng tất cả các API và các khối lệnh của Java trong lập trình JSP của mình bao gồm các câu lệnh điều kiện, vòng lặp, v.v.

### **Lệnh điều kiện**

|  |  |
| --- | --- |
| **IF-ELSE** | **SWITCH CASE** |
|  |  |

### **Vòng lặp**

|  |  |
| --- | --- |
| **FOR** | **WHILE** |
|  |  |

### **Toán tử**

|  |  |
| --- | --- |
| **Loại** | **Toán tử** |
| **Postfix** | () [] . (dot operator) |
| **Unary** | ++ -- ! ~ |
| **Multiplicative** | \* / % |
| **Additive** | + - |
| **Shift** | >> >>> << |
| **Relational** | > >= < <= |
| **Equality** | == != |
| **Bitwise AND** | & |
| **Bitwise XOR** | ^ |
| **Bitwise OR** | | |
| **Logical AND** | && |
| **Logical OR** | || |
| **Conditional** | ?: |
| **Assignment** | = += -= \*= /= %= >>= <<= &= ^= |= |
| **Comma** | , |

### **Keyword**

Ngôn ngữ biểu thức JSP:

|  |  |
| --- | --- |
| **Boolean** | true và false |
| **Số nguyên** | như trong Java |
| **Dấu phẩy động** | như trong Java |
| **Chuỗi** | với dấu ngoặc kép và dấu ngoặc kép; " được thoát là \", ' được thoát là \', và \ được thoát là \\. |
| **Null** | null |

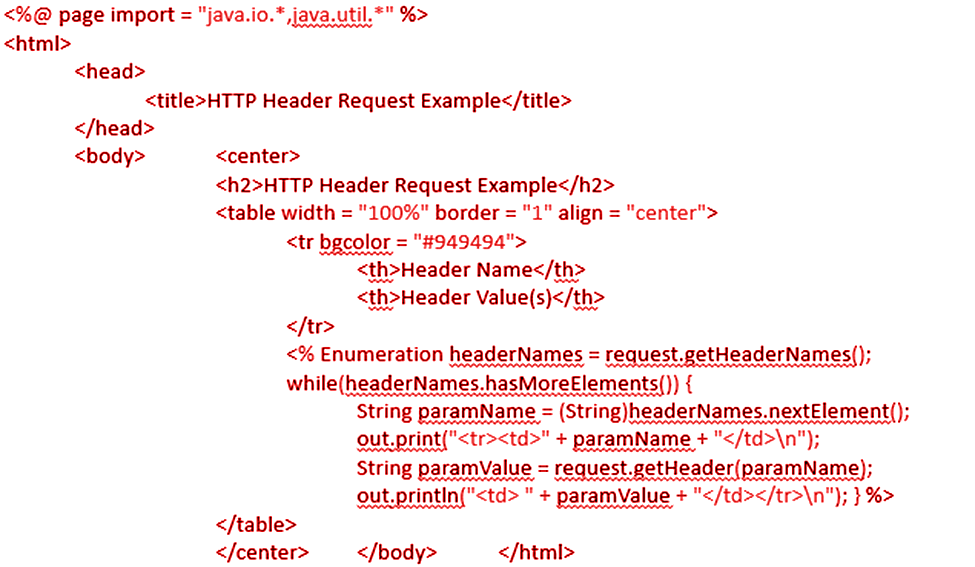
### **Client Request**

|  |  |
| --- | --- |
| **Header** | **Description** |
| **Accept** | Tiêu đề này chỉ định các loại MIME mà trình duyệt hoặc các máy khách khác có thể xử lý. Giá trị của Image/png hoặc image/jpeg là hai khả năng phổ biến nhất |
| **Accept-Charset** | Tiêu đề này chỉ định các bộ ký tự mà trình duyệt có thể sử dụng để hiển thị thông tin.  Ví dụ: ISO-8859-1 |
| **Accept-Encoding** | Tiêu đề này chỉ định các loại mã hóa mà trình duyệt biết cách xử lý. Giá trị của gzip hoặc nén là 2 khả năng phổ biến nhất |
| **Accept-Language** | Tiêu đề này chỉ định ngôn ngữ ưa thích của máy khách trong trường hợp servlet có thể tạo ra kết quả bằng nhiều ngôn ngữ.  Ví dụ: en, en-us, ru, … |
| **Authorization** | Tiêu đề này được khách hàng sử dụng để nhận dạng bản thân khi truy cập các trang web được bảo vệ bằng bảo mật |
| **Connection** | Tiêu đề này cho biết liệu máy khách có thể xử lý các kết nối HTTP liên tục hay không. Kết nối liên tục cho phép máy khách hoặc trình duyệt khác truy xuất nhiều tệp với một yêu cầu duy nhất. Giá trị của Keep-Active có nghĩa là nên sử dụng acsc kết nối liên tục |
| **Connection-Length** | Tiêu đề này chỉ áp dụng cho các yêu cầu POST và cung cấp kích thước của dữ liệu POST tính bằng byte |
| **Cookie** | Tiêu đề này trả lại cookie cho các máy chủ đã gửi chúng trước đó đến trình duyệt |
| **Host** | Tiêu đề này chỉ định máy chủ lưu trữ và cổng như được đưa ra trong URL gốc |
| **If-Modified-Since** | Tiêu đề này chỉ ra rằng khách hàng chỉ muốn trang nếu nó đã được thay đổi sau ngày được chỉ định. Máy chủ gửi mã 304 có nghĩa là tiêu đề không sửa đổi nếu không có kết quả mới hơn |
| **If-Unmodified-Since** | Tiêu đề này là đảo ngược của If-Modified-Since. Nó chỉ định rằng hoạt động sẽ chỉ thành công nếu tài liệu cũ hơn ngày được chỉ định |
| **Referer** | Tiêu đề này cho biết URL của các trang web giới thiệu.  Ví dụ: nếu bạn đang ở trang web 1 và nhấp vào liên kết đến trang web 2, URL của trang web 1 được bao gồm trong tiêu đề. Giới thiệu khi trình duyệt yêu cầu trang web 2 |
| **User-Agent** | Tiêu đề này xác định trình duyệt hoặc ứng dụng khách khác đưa ra yêu cầu và có thể được sử dụng để trả về nội dung khác nhau cho các loại trình duyệt khác nhau |

### **Object HttpServletRequest**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **Cookie[] getCookie()** | Trả về 1 mảng chứa tất cả các đối tượng Cookie mà máy khách đã gửi cùng với yêu cầu này |
| **Enumeration getAttributeNames()** | Trả về 1 liệt kê chứa tên của các thuộc tính có sẵn cho yêu cầu này |
| **Enumeration**  **getHeaderNames()** | Trả về 1 bảng liệt kê tất cả các tên đầu đề mà yêu cầu này chứa |
| **Enumaration**  **getParameterNames()** | Trả về 1 bảng liệt kê các đối tượng chuỗi chứa tên của các tham số chứa trong yêu cầu này |
| **HttpSession**  **getSession()** | Trả về phiên hện tại được liên kết với yêu cầu này hoặc nếu yêu cầu không có phiên, hãy tạo 1 phiên |
| **HttpSession**  **getSession(boolean create)** | Trả về HttpSession hiện tại được liên kết với yêu cầu này hoặc, nếu không có phiên hiện tại và tạo là true, trả về 1 phiên mới |
| **Locale getLocale()** | Trả về ngôn ngữ ưa thích mà máy khách sẽ chấp nhận nội dung, dựa trên tiêu đề Chấp nhận ngôn ngữ |
| **Object**  **getAttribute(String name)** | Trả về giá trị của thuộc tính đã đặt tên là 1 Đối tượng, hoặc null nếu không có thuộc tính nào có tên đã cho tồn tại |
| **ServletInputStream**  **getInputStream()** | Truy xuất phần thân của yêu cầu dưới dạng dữ liệu nhị phân bằng cách sử dụng ServletInputStream |
| **String**  **getAuthType()** | Trả về tên của lược đồ xác thực được sử dụng để bảo vệ servlet  Ví dụ: “BASIS” hoặc “SSL” hoặc null nếu JSP không được bảo vệ |
| **String**  **getCharacterEncoding()** | Trả về tên của mã hóa ký tự được sử dụng trong nội dung của yêu cầu này |
| **String**  **getContentType()** | Trả về kiểu MIME của nội dung yêu cầu hoặc null nếu không xác định được kiểu |
| **String**  **getContextPath()** | Trả về phần URI yêu cầu cho biết ngữ cảnh của yêu cầu |
| **String**  **getHeader(String name)** | Trả về giá trị của tiêu đề yêu cầu đã xác định dưới dạng chuỗi |
| **String**  **getMethod()** | Trả về tên của phương thức HTTP mà yêu cầu này được thực hiện  Ví dụ: GET, POST hoặc PUT |
| **String**  **getParameter(String name)** | Trả về giá trị của tham số request dưới dạng String hoặc null nếu tham số không tồn tại |
| **String**  **getPathInfo()** | Trả về bất ký thông tin đường dẫn bổ sung nào được liên kết với URL mà máy khách đã gửi khi thực hiện yêu cầu này |
| **String**  **getProtocol()** | Trả về tên và phiên bản của giao thức mà yêu cầu sử dụng |
| **String**  **getQueryString()** | Trả về chuỗi truy vấn chứa trong URL yêu cầu sau đường dẫn |
| **String**  **getRemoteAddr()** | Trả về địa chỉ Giao thức Internet(IP) của máy khách đã gửi yêu cầu |
| **String**  **getRemoteHost()** | Trả về tên đủ điều kiện của máy khách gửi yêu cầu |
| **String**  **getRemoteUser()** | Trả về thông tin đăng nhập của người dùng thực hiện yêu cầu này, nếu người dùng đã được xác thực hoặc null nếu người dùng chưa được xác thực |
| **String**  **getRequestURI()** | Trả về 1 phân URL của yêu cầu này từ tên giao thức lên đến chuỗi truy vấn trong dòng đầu tiên của yêu cầu HTTP |
| **String**  **getRequestedSessionId()** | Trả về ID phiên do máy khách chỉ định |
| **String**  **getServletPath()** | Trả về phần URL của yêu cầu này gọi JSP |
| **String[]**  **getParameterValues(string name)** | Trả về 1 mảng các đối tượng Chuỗi chứa tất cả các giá trị mà tham số yêu cầu đã cho có, hoặc null nếu tham số không tồn tại |
| **boolean isSecure()** | Trả về boolean cho biết liệu yêu cầu này có được thực hiện bằng kênh bảo mật hay không, chẳng hạn như HTTPS |
| **int getContentLength()** | Trả về độ dài, tính bằng byte, của nội dung yêu cầu và được cung cấp bởi luồng đầu vào hoặc -1 nếu không xác định độ dài |
| **int getIntHeader(String name)** | Trả về giá trị của tiêu đề yêu cầu đã được xác định dưới dạng int |
| **int getServletPort()** | Trả về số cổng mà yêu cầu này đã được nhận |

**Ví dụ:**

****

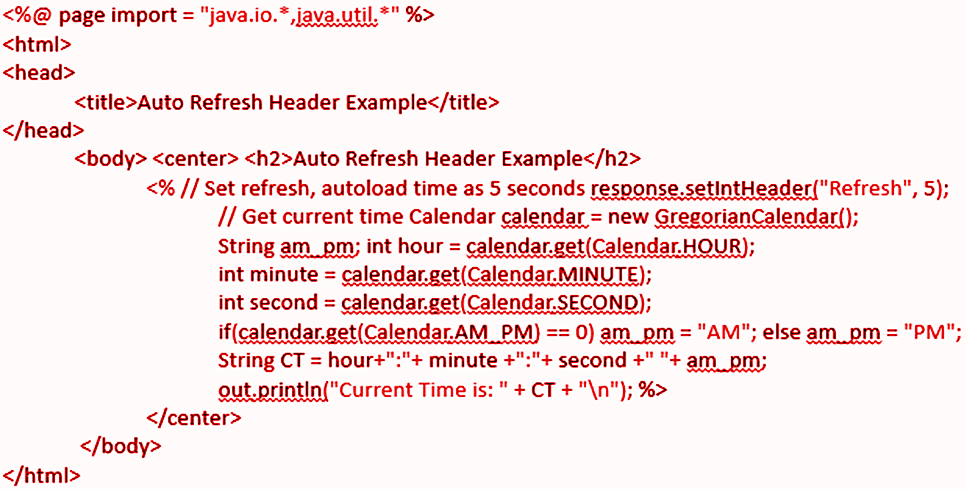
### **Servlet Response**

|  |  |
| --- | --- |
| **Header** | **Description** |
| **Allow** | Chỉ định các phương thức yêu cầu (GET, POST, v.v) mà máy chủ hỗ trợ |
| **Cache-Control** | Tiêu đề này chỉ định các trường hợp mà tài liệu phản hồi có thể lưu trữ một cách an toàn. Nó có thể có các giá trị công khai, riêng tư hoặc không có bộ nhớ cache… Public có nghĩa là tài liệu có thể lưu vào bộ nhớ cache. Private có nghĩa là tài liệu chỉ dành cho 1 người dùng và chỉ có thể lưu trữ trong bộ nhớ cache riêng tư( không chia sẻ) và không có cache có nghĩa là tài liệu không bao giờ được lưu trữ vào bộ nhớ cache |
| **Connection** | Tiêu đề này hướng dẫn trình duyệt có sử dụng các kết nối HTTP liên tục hay không. Giá trị close hướng dẫn trình duyệt không sử dụng các kết nối HTTP liên tục và giữ các phương tiện tồn tại bằng cách sử dụng các kết nối liên tục |
| **Content-Disposition** | Tiêu đề này cho phép bạn yêu cầu trình duyệt yêu cầu người dùng lưu phản hồi vào đĩa trong một tệp có tên đã cho |
| **Content-Encoding** | Tiêu đề này chỉ định các trang được mã hóa trong quá trình truyền |
| **Content-Language** | Tiêu đề này biểu thị ngôn ngữ mà tài liệu được viết.  Ví dụ: en, en-us, ru, … |
| **Content-Length** | Tiêu đề này cho biết số byte trong phản hồi. Thông tin này chỉ cần thiết nếu trình duyệt đang sử dụng kết nối HTTP liên tục (giữ sống) |
| **Content-Type** | Tiêu đề này cung cấp loại MIME (Multipurpose Internet Mail Extension) của tài liệu phản hồi |
| **Expires** | Chỉ định thời gian mà nội sẽ bị coi là lỗi thời và do đó không còn được lưu vào bộ nhớ cache nữa. |
| **Last-Modified** | Cho biết khi nào tài liệu được thay đổi lần cuối. Sau đó, máy khách có thể Lưu trữ tài liệu và cung cấp ngày bằng tiêu đề yêu cầu IF-Modified-since trong các yêu cầu |
| **Location** | Tiêu đề này phải được bao gồm trong tất cả các phản hồi có mã trạng thái trong những năm 300. Thao tác này sẽ thông báo cho trình duyệt về địa chỉ tài liệu. Trình duyệt tự động kết nối lại với vị trí này và truy xuất tài liệu mới. |
| **Refresh** | Tiêu đề này chỉ định thời gian trình duyệt sẽ yêu cầu 1 trang cập nhật. Bạn có thể chỉ định thời gian theo số giây sau đó 1 trang sẽ được làm mới |
| **Retry-After** | có thể được sử dụng cùng phản hồi 503(service Unavailable) để khách hàng biết bao lâu nó có thể lặp lại yêu cầu của mình |
| **Set-Cookie** | Tiêu đề này chỉ định 1 cookie được liên kết với trang |

### **Object HttpServletResponse**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **String encodeRedirectURL(String url)** | Mã hóa URL được chỉ định để sử dụng trong phương thức sendRedirect hoặc nếu không cần mã hóa, trả về URL không thay đổi. |
| **String encodeURL(String url)** | Mã hóa URL được chỉ định bằng cách bao gồm ID phiên trong đó hoặc nếu không cần mã hóa, trả về URL không thay đổi. |
| **boolean containsHeader(String name)** | Trả về Boolean, cho biết tiêu đề phản hồi đã đặt tên đã được đặt chưa. |
| **boolean isCommitted()** | Trả về boolean cho biết phản hồi đã được cam kết hay chưa |
| **void addCookie(Cookie cookie)** | Thêm cookie được chỉ định vào phản hồi |
| **void addDateHeader(String name, long date)** | Thêm tiêu đề phản hồi với tên và ngày-giá trị đã cho |
| **void addHeader(String name, String value)** | Thêm tiêu đề phản hồi với tên và giá trị đã cho |
| **void addintHeader(String name, int value)** | Thêm tiêu đề phản hồi với tên và giá trị số nguyên đã cho |
| **void flushBuffer()** | Buộc bất kì nội dung nào trong bộ đệm phải được ghi cho khách hàng |
| **void reset()** | Xóa mọi dữ liệu tồn tại trong bộ đệm cũng như mã trạng thái và tiêu đề |
| **void resetBuffer()** | Xóa nội dung của bộ đệm cơ bản trong phản hồi mà không xóa tiêu đề hoặc mã trạng thái |
| **void sendError(int sc)** | Gửi phản hồi lỗi cho máy khách bằng cách sử dụng mã trạng thái được chỉ định và xóa bộ nhớ đệm |
| **void sendError(int sc, String msg)** | Gửi phản hồi lỗi cho máy khách bằng cách sử dụng trạng thái được chỉ định |
| **void sendRedirect(String location)** | Gửi phản hồi chuyển hướng tạm thời đến máy khách bằng cách sử dụng URL vị trí chuyển hướng được chỉ định |
| **void setBufferSize(int size)** | Đặt kích thước bộ đệm ưa thích cho phần thân của phản hồi |
| **void setCharacterEncoding(String charset)** | Đặt mã hóa ký tự (bộ ký tự MIME) của phản hồi được gửi đến máy khách, ví dụ: thành UTF-8 |
| **void setContentLength(int len)** | Đặt độ dài của nội dung trong phản hồi trong HTTP servlet; phương pháp này cũng đặt tiêu đề HTTP Content-length |
| **void setContentType(String type)** | Đặt loại nội dung của phản hồi được gửi đến máy khách, nếu phản hồi chưa được cam kết |
| **void setDateHeader(String name, long date)** | Đặt tiêu đề phản hồi với tên và giá trị ngày đã cho |
| **void setHeader(String name, String value)** | Đặt tiêu đề phản hồi với tên và giá trị đã cho |
| **void setintHeader(String name, int value)** | Đặt tiêu đề phản hồi với tên và giá trị số nguyên đã cho |
| **void setLocale(Locale loc)** | Đặt ngôn ngữ của phản hồi, nếu phản hồi chưa được cam kết |
| **void setStatus(int sc)** | Đặt mã trạng thái của phản hồi này |

**Ví dụ**

****

### **Form Processing**

1. **Các phương pháp trong xử lý biểu mẫu**

* Phương thức GET
* Phương thức POST

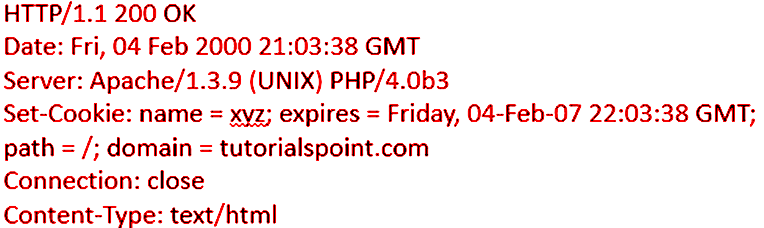
1. **Đọc dữ liệu biểu mẫu bằng JSP**

* **getParameter()** - Bạn gọi phương thức request.getParameter() để lấy giá trị của một tham số form.
* **getParameterValues()** − Gọi phương thức này nếu tham số xuất hiện nhiều lần và trả về nhiều giá trị, ví dụ hộp kiểm.
* **getParameterNames()** − Gọi phương thức này nếu bạn muốn có một danh sách đầy đủ tất cả các tham số trong yêu cầu hiện tại.
* **getInputStream()** − Gọi phương thức này để đọc luồng dữ liệu nhị phân đến từ máy khách.

### **Cookie Processing**

1. **Cấu trúc Cookie**

* Cookie thường được đặt trong tiêu đề HTTP
* JSP đặt cookie có thể gửi các tiêu đề

****

### **Servlet Cookies Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void setDomain(String pattern)** | Phương pháp này đặt tên miền mà cookie áp dụng; ví dụ: tutorialspoint.com |
| **public string getDomain()** | Phương pháp này lấy tên miền mà cookie áp dụng; ví dụ: tutorialspoint.com |
| **public void setMaxAge(int expiry)** | Phương pháp này đặt thời gian (tih1 bằng giây) sẽ trôi qua trước khi cookie hết hạn. Nếu bạn không đặt tùy chọn này, cookie sẽ chỉ tồn tại cho phiên bản hiện tại. |
| **public int getMaxAge()** | Phương pháp này trả về tuổi tối đa của cookie, được chỉ định bằng giây. Theo mặc định, -1 cho biết cookie sẽ tồn tại cho đến khi trình duyệt tắt |
| **public String getName()** | Phương thức này trả về tên của cookie, tên không thể thay đổi sau khi tạo. |
| **public void setValue(String newValue)** | Phương thức này đặt giá trị được liên kết với cookie |
| **pubic String getValue()** | Phương thức này nhận được giá trị được liên kết với cookie |
| **public void setPath(String uri)** | Phương pháp này đặt đường dẫn mà cookie này áp dụng. Nếu bạn không chỉ định đường dẫn, cookie sẽ được trả về cho tất cả các URL trong cùng thư mục với trang hiện tại cũng như tất cả các thư mục con. |
| **public String getPath()** | Phương thức này lấy đường dẫn mà cookie này áp dụng |
| **public void setSecure(boolean flag)** | Phương pháp này đặt giá trị boolean cho biết liệu cookie chỉ nên được gửi qua các kết nối được mã hóa (tức là SSL) |
| **public void setComment(String purpose)** | Phương pháp này chỉ định nhận xét mô tả mục đích của cookie. Nhận xét này rất hữu ích nếu trình duyệt trình bày cookie cho người dùng |
| **public string getComment()** | Phương thức này trả về nhận xét mô tả mục đích của cookie này hoặc null nếu cookie không có nhận xét. |

### **Setting Cookies JSP**

1. **Các bước**

* **Bước 1:** Tạo đối tượng Cookie

***Cookie cookie = new Cookie("key","value");***

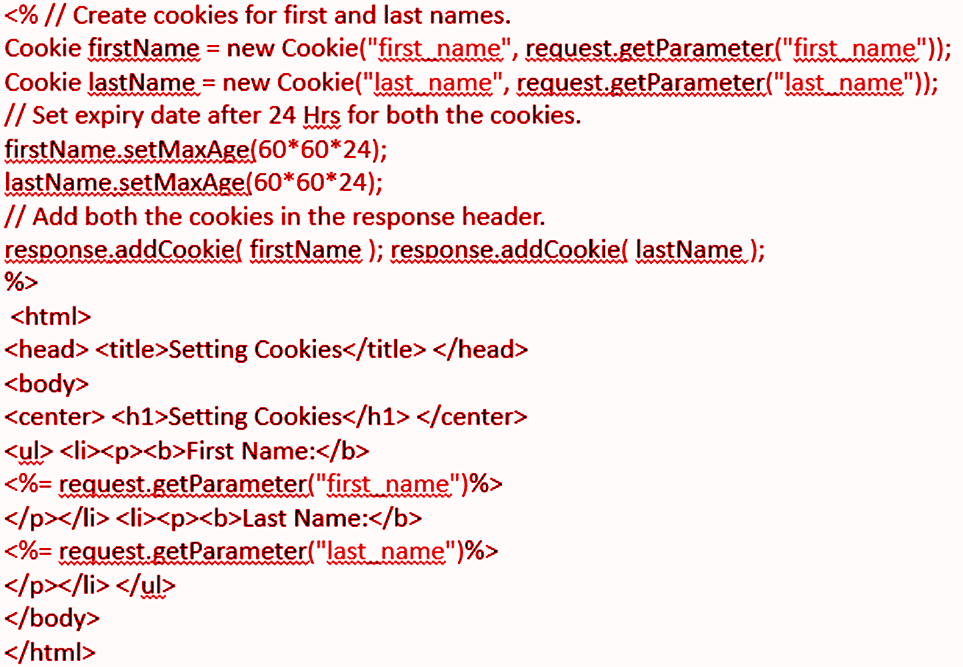
* **Bước 2:** Set maximum age

***cookie.setMaxAge(60\*60\*24);***

* **Bước 3:** Gửi Cookie vào header phản hồi HTTP

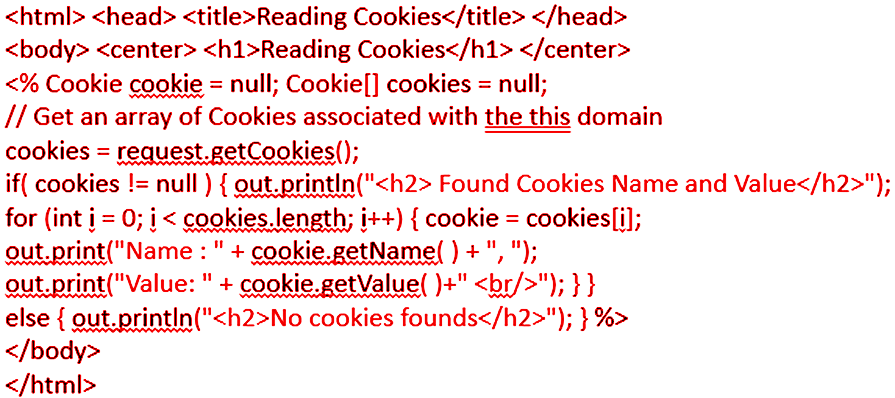
***response.addCookie(cookie);***

1. **Ví dụ**

****

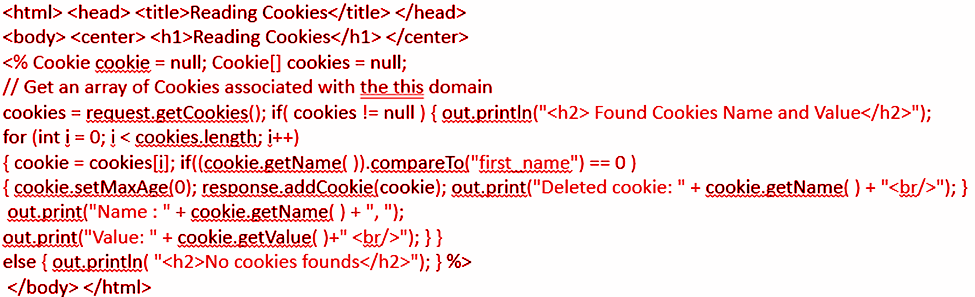
1. **Đọc Cookie với JSP**

* Tạo một mảng các đối tượng javax.servlet.http.Cookie
* Sử dụng mảng và sử dụng các phương thức getName() và getValue()

****

1. **Xóa cookie với JSP**

* Đọc một cookie đã có sẵn và lưu trữ nó trong đối tượng Cookie.
* Đặt tuổi cookie bằng 0 bằng cách sử dụng phương thức setMaxAge() để xóa cookie hiện có.
* Thêm cookie này trở lại tiêu đề phản hồi.

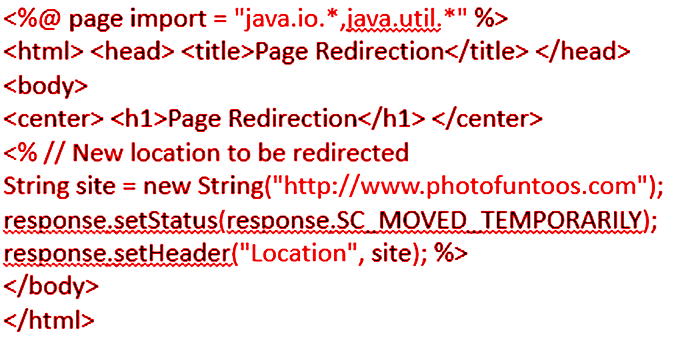
****

### **Chuyển hướng trang**

* Chuyển hướng trang thường được sử dụng khi một tài liệu di chuyển đến một vị trí mới
* Sử dụng phương thức sendRedirect()

*public void response.sendRedirect(String location) throws IOException*

* Ví dụ:

****

### **Kết luận**

JSP là một template hiệu quả trong lập trình web nhưng cần chú ý :

* Không dễ để truy cập cơ sở dữ liệu.
* Rất khó để theo dõi.
* Thời gian cần thiết để biên dịch JSP lâu hơn.

## **7.Java Stream**

### Java Stream

● Là một đối tượng mới của Java được giới thiệu từ phiên bản Java 8, giúp cho việc thao tác trên collection và array trở nên dễ dàng và tối ưu.

● Định nghĩa đơn giản Stream là một wrapper(bao bọc) của collection và array.

● Cung cấp các phương thức tăng cường xử lý các phần tử bên trong nó với kỹ thuật lập trình Lambda.

### Đặc điểm

● Stream hỗ trợ hoàn hảo cho Lambda Expression.

● Stream không chứa các element của collection hay array.

● Stream là immutable object.

● Stream không dùng lại được, nghĩa là một khi đã sử dụng nó xong, không thể gọi lại để sử dụng lần nữa.

● Không thể dùng index để access các element trong Stream.

● Stream hỗ trợ thao tác song song các element trong collection hay array.

● Stream hỗ trợ thao tác lazy, khi cần thì thao tác mới được thực hiện.

### Các thao tác

* Duyệt:

· forEach() : method: Dùng để duyệt qua mọi phần tử trong stream.

* Tích lũy

· Reduce(): method: Reduce() method với 1 trong 2 tham số truyền vào là method reference, dùng để kết hợp các phần tử thành một giá trị đơn cùng kiểu với dữ liệu ban đầu.

* Lọc

· Filter(): Dùng để lọc và xóa bỏ các phần tử với điều kiện do người dùng định nghĩa.

* Tổng hợp

· sum(), count(), min(), max(), average()

* Chuyển đổi

· map(): Map method được sử dụng để trả về một stream mà ở đó các phần tử đã được thay đổi theo cách người dùng tự định nghĩa.

· mapToDouble()

· mapToInt()

· mapToLong()

* Kiểm tra

· allMatch()

· anyMatch()

· noneMatch()

### Các method trong stream:

### **Terminal Operations**

* Collect method:
* Là một trong những phương thức xử lý tiêu biểu của interface Stream.
* Dùng để trả về kết quả của stream dưới dạng List hoặc Set.
* Phương thức collect() giúp thu thập kết quả Stream sang một Collection.
* forEach method (.forEach(T -> {...})):
* Stream<T>.forEach(item -> {...}) được sử dụng để duyệt các phần tử trong Stream.

### - Reduce method

* Là phương thức xử lý nó giúp làm giảm các phần tử của stream về một giá trị đơn lẻ.
* Dùng để kết hợp các phần tử thành một giá trị đơn cùng kiểu với dữ liệu ban đầu.
* Phương thức reduce() kết hợp các phần tử luồng thành một bằng cách sử dụng một BinaryOperator.
* Max, Min method
* Trả về giá trị bé nhất hoặc lớn nhất trong các phần tử.
* Chấp nhận đối số là một Comparator sao cho các item trong stream có thể được so sánh với nhau để tìm tối thiểu (min) hoặc tối đa (max).

### **Intermediate Operations**

* Distinct method:
* Trả về một stream với các phần tử không trùng lặp.
* Map method (.map()):
* Là phương thức mạnh hỗ trợ việc map từng phần tử của danh sách với đối tượng khác.
* Được sử dụng để trả về một stream mà ở đó các phần tử đã được thay đổi theo cách người dùng tự định nghĩa.
* Stream map() giúp ánh xạ các phần tử tới các kết quả tương ứng.

Filter method:

* Phương thức filter () là một hoạt động trung gian của giao diện Luồng cho phép chúng ta lọc các phần tử của luồng phù hợp với một vị trí nhất định.
* Dùng để lọc và xóa bỏ các phần tử với điều kiện do người dùng định nghĩa.

Stream<T> filter(Predicate<? super T> predicate)

* Sorted method
* Giúp sắp xếp các phần tử theo một thứ tự xác định.

- Limit method

* Được sử dụng để loại bỏ các phần tử n đầu tiên của Stream.
* Với tham số đầu vào là số nguyên không âm n nó sẽ trả về một stream chứa n phần tử đầu tiên.

· Skip method

* Được sử dụng để cắt giảm kích thước của Stream.
* Với tham số truyền vào là số nguyên không âm n nó sẽ trả về các phần tử còn lại đằng sau n phần tử đầu tiên.

### **Luồng song song – Parallel Streams**

- Là một sự thay thế của stream để phục vụ cho việc xử lý song song các phần tử. Kết quả của đoạn code sau thể hiện sự khác nhau giữa Stream và Parallel Stream.

Chú ý

- Thực chất các intermediate method không hoạt động khi gọi đến mà chỉ thực thi khi có một terminal method kết thúc stream của nó.

- Một luồng hay một stream có thể không có hoặc có nhiều intermediate method kết hợp với nhau .

Stream được sử dụng để trả về kết quả các phần tử sau khi được xử lý thông qua các method mà không làm thay đổi giá trị các phần tử gốc.

## **8.Đóng gói và triển khai Website trên Tomcat Server**

### **Khái niệm**

**Khái niệm Apache**

- Apache là một web server mã nguồn mở, được sử dụng để chạy và quản lý các ứng dụng web. Trong Java web, Apache thường được sử dụng để kết nối các ứng dụng web với máy chủ và hỗ trợ các công nghệ như Servlet và JSP. Ngoài ra, Apache còn cung cấp các tính năng bảo mật, quản lý phiên và quản lý tài nguyên để giúp các ứng dụng web hoạt động hiệu quả hơn.

**Khái niệm Tomcat**

- Tomcat là một máy chủ ứng dụng web miễn phí và mã nguồn mở được phát triển bởi Apache Software Foundation. Nó được sử dụng để triển khai các ứng dụng web Java, các trang web tĩnh và các trang web động. Tomcat cung cấp môi trường thực thi cho các ứng dụng web được viết bằng các ngôn ngữ lập trình Java như Servlets, JSP và WebSocket. Nó cũng hỗ trợ các chuẩn web như HTTP và SSL. Tomcat là một trong những máy chủ web phổ biến nhất được sử dụng trong các dự án Java web.

**Khái niệm Apache Tomcat**

-  [Apache Tomcat®](http://tomcat.apache.org/)  là một phần mềm mã nguồn mở thực hiện cài đặt công nghệ Java Servlet, JavaServer Pages, Java Expression Language và Java WebSocket.

- Tomcat là một ứng dụng máy chủ gọn nhẹ, thường dùng để deploy các ứng dụng Java Web. Nó được phát triển bởi Apache và hoàn toàn miễn ph

### **Khái niệm về đề tài**

* Đóng gói và triển khai Website trên Tomcat Server là quá trình đóng gói các tập tin của website vào một file JAR hoặc WAR và triển khai nó trên máy chủ Tomcat Server.
* Quá trình này đảm bảo rằng website của bạn có thể được chạy trên Tomcat Server một cách thích hợp và được truy cập từ các trình duyệt web.
* Để thực hiện quá trình này, bạn cần phải cấu hình Tomcat Server và cài đặt các công cụ và thư viện cần thiết để triển khai website của mình.
* Sau khi triển khai, bạn có thể truy cập vào trang web của mình bằng cách sử dụng địa chỉ IP hoặc tên miền của máy chủ Tomcat Server.

### **Các yếu tố chính**

* Bảo mật: Việc đóng gói và triển khai giúp bảo vệ mã nguồn của website, tránh việc lộ thông tin quan trọng và giảm rủi ro bị tấn công.
* Quản lý: Quá trình triển khai đơn giản và tối ưu hóa hiệu suất, giúp quản lý website dễ dàng hơn.
* Tương thích: Tomcat Server hỗ trợ nhiều ngôn ngữ lập trình và nền tảng khác nhau, đảm bảo tương thích và ổn định cho website.

### **ƯU VÀ NHƯỢC ĐIỂM CỦA APACHE TOMCAT**

**Ưu điểm**

● Miễn phí

● Dễ dàng cài đặt và triển khai

● Linh hoạt

● Tiết kiệm tài nguyên

● Được cộng đồng hỗ trợ

**Nhược điểm**

● Yêu cầu kiến thức kỹ thuật

● Khó khăn trong việc quản lý ứng dụng

● Khó khăn trong việc xử lý các vấn đề bảo mật

### NỘI DUNG CHÍNH

### TOMCAT SERVER

● Tomcat là một máy chủ web Java được phát triển bởi Apache Software Foundation. Đó là một trong những máy chủ web phổ biến nhất để chạy các ứng dụng web Java.

● Cách cài đặt Tomcat Server gồm có 4 cách và cấu hình Tomcat Server gồm có 5 cách

### CÁCH CÀI ĐẶT TOMCAT SERVER

● Bước 1: Tải xuống Tomcat từ trang chủ của Apache.

● Bước 2: Giải nén tệp zip vào thư mục của bạn và đặt đường dẫn trong biến môi trường "CATALINA\_HOME".

● Bước 3: Mở cửa sổ dòng lệnh hoặc terminal và chạy tệp "startup.bat" (Windows) hoặc "startup.sh" (Linux) trong thư mục "bin" của Tomcat.

● Bước 4: Kiểm tra xem Tomcat đã được khởi động thành công hay không bằng cách truy cập địa chỉ "http://localhost:8080/" trên trình duyệt web của bạn. Nếu bạn thấy trang chào mừng Tomcat, điều đó có nghĩa là Tomcat đã được cài đặt và chạy thành công.

### CÁCH CẤU HÌNH TOMCAT SERVER

● Bước 1: Để cấu hình Tomcat Server, bạn cần truy cập vào tệp "server.xml" trong thư mục "conf" của Tomcat.

● Bước 2: Cấu hình các giá trị cơ bản như cổng và tên máy chủ.

● Bước 3: Cấu hình các nguồn dữ liệu, bao gồm xác thực và quản lý người dùng.

● Bước 4: Cấu hình các ứng dụng web, bao gồm đường dẫn và các thiết lập khác.

● Bước 5: Lưu tệp "server.xml" và khởi động lại Tomcat Server để các thay đổi cấu hình có hiệu lực.

### ĐÓNG GÓI WEBSITE THÀNH FILE WAR

● Xác định các tài nguyên và mã nguồn của ứng dụng web cần được đóng gói. Bạn có thể sử dụng một IDE như Eclipse, IntelliJ hoặc NetBeans để tạo và quản lý ứng dụng web của mình.

● Tạo một file WAR mới trong IDE bằng cách chọn File > New > Project, chọn loại dự án "Web Application" và nhập thông tin cần thiết như tên dự án, đường dẫn, mô tả, v.v.

● Thêm các tài nguyên và mã nguồn của ứng dụng web vào dự án mới.

● Cấu hình file web.xml để định cấu hình ứng dụng web, bao gồm các servlet, filter, listener, mô tả và các thông tin khác. File web.xml thường được đặt trong thư mục WEB-INF của ứng dụng web.

● Cấu hình các thư viện phụ thuộc (dependencies) cho ứng dụng web bằng cách thêm các file .jar vào thư mục lib của ứng dụng web. Bạn cũng có thể sử dụng Maven hoặc Gradle để quản lý các phụ thuộc.

● Sử dụng công cụ đóng gói (packaging tool) như Apache Ant hoặc Maven để đóng gói ứng dụng web thành file WAR. Trong Maven, bạn có thể sử dụng Plugin maven-war-plugin để đóng gói ứng dụng web. Plugin này cung cấp các tùy chọn cấu hình để tùy chỉnh quá trình đóng gói, bao gồm tên file WAR, đường dẫn, các tài nguyên cần được đóng gói, v.v.

● Sau khi đóng gói hoàn tất, bạn có thể sử dụng file WAR để triển khai ứng dụng web trên một máy chủ web như Apache Tomcat, Jetty, v.v.

### TRIỂN KHAI TRÊN TOMCAT SERVER

* Triển khai bằng giao diện web của Tomcat Manager
* Triển khai bằng cách đặt file WAR vào thư mục webapps của Tomcat
* Triển khai bằng cách sử dụng Maven plugin
* Tùy vào mục đích sử dụng và yêu cầu của từng dự án, bạn có thể sử dụng cách triển khai phức tạp hơn như sử dụng Docker hoặc Kubernetes để triển khai ứng dụng trên môi trường sản phẩm.

### TRIỂN KHAI BẰNG GIAO DIỆN WEB CỦA TOMCAT MANAGER

* Bước 1: Đăng nhập vào giao diện quản trị web của Tomcat Manager tại URL https://:/manager/html (trong đó là địa chỉ IP hoặc tên miền của server Tomcat, là cổng mà Tomcat đang lắng nghe).
* Bước 2: Chọn mục "Deploy" trên thanh menu và click vào nút "Choose File" để chọn file WAR muốn triển khai.
* Bước 3: Click nút "Deploy" để bắt đầu quá trình triển khai.

●

### **TRIỂN KHAI BẰNG CÁCH ĐẶT FILE WAR VÀO THƯ MỤC WEBAPPS CỦA TOMCAT**

● Bước 1: Copy file WAR vào thư mục webapps của Tomcat (thư mục này thường được đặt tại đường dẫn /opt/tomcat/webapps trên Linux hoặc C:\Program Files\Apache Software Foundation\Tomcat X.X\webapps trên Windows, trong đó X.X là phiên bản của Tomcat).

● Bước 2: Khởi động lại Tomcat để cập nhật danh sách ứng dụng.

● Bước 3: Kiểm tra xem ứng dụng đã được triển khai bằng cách truy cập vào URL http://:/ (trong đó là tên của ứng dụng, thường là tên của file WAR mà bạn triển khai).

●

### **TRIỂN KHAI BẰNG CÁCH SỬ DỤNG MAVEN PLUGIN**

Bước 1: Thêm dependency của Maven plugin vào file pom.xml:

```

org.codehaus.mojo

tomcat-maven-plugin

2.2

```

Bước 2: Cấu hình plugin trong file pom.xml:

```

org.codehaus.mojo

tomcat-maven-plugin

2.2

http://:/manager/text

tomcat

/myapp

```

Bước 3: Chạy lệnh sau để triển khai ứng dụng:

```

mvn tomcat:deploy

```

### **QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER**

Tomcat Manager: Là một ứng dụng Web được tích hợp sẵn trong Tomcat, cho phép quản trị viên quản lý các ứng dụng trên Tomcat Server từ xa thông qua giao diện đơn giản. Tomcat Manager cung cấp các chức năng như deploy, undeploy và reload các ứng dụng, theo dõi trạng thái của các ứng dụng và xem các thông tin về lỗi trong quá trình triển khai. Người dùng cần đăng nhập bằng thông tin đăng nhập được cấp để truy cập vào Tomcat Manager.

### **QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER**

JMX (Java Management Extensions): Là một công nghệ quản lý ứng dụng Java tiêu chuẩn, cho phép quản trị viên quản lý các ứng dụng trên Tomcat Server từ xa thông qua các công cụ quản lý JMX như JConsole hoặc VisualVM. JMX cung cấp các giao diện quản lý tương tác để quản lý và giám sát các ứng dụng trên Tomcat Server, bao gồm quản lý bộ nhớ, luồng và các chỉ số hiệu suất khác.

### **QUẢN LÝ ỨNG DỤNG TRÊN TOMCAT SERVER**

Cả hai công cụ này đều cung cấp các tính năng quản lý mạnh mẽ và hữu ích để quản lý ứng dụng trên Tomcat Server. Tuy nhiên, Tomcat Manager dễ sử dụng hơn và phù hợp cho người mới bắt đầu trong lĩnh vực quản lý ứng dụng, trong khi JMX phù hợp cho những người có kinh nghiệm với các công nghệ quản lý ứng dụng Java.

Ưu nhược điểm

**Ưu điểm:**

● Bài học giúp người học hiểu được cách đóng gói và triển khai website trên Tomcat server, là một trong những kỹ năng quan trọng trong lập trình web.

● Bài học cung cấp cho người học kiến thức về cách cấu hình và quản lý Tomcat server, từ đó giúp người học có thể tự tin triển khai website của mình trên môi trường thực tế.

● Bài học giúp người học nắm được quy trình triển khai ứng dụng web bằng Eclipse IDE, giúp tăng tính thực tiễn và sự tiện lợi cho việc triển khai ứng dụng.

**Nhược điểm:**

● Bài học chỉ tập trung vào Tomcat server và sử dụng Eclipse IDE để triển khai ứng dụng, không đề cập đến các công cụ và phương pháp khác như Docker, Kubernetes, Jenkins... để triển khai và quản lý ứng dụng trên môi trường thực tế.

● Bài học không đề cập đến các vấn đề liên quan đến bảo mật và tối ưu hóa hiệu suất của ứng dụng khi triển khai trên Tomcat server.

### TỔNG KẾT

● Đóng gói web application: Ta cần tạo file WAR (Web Application Archive) bằng cách đóng gói tất cả các tài nguyên (file HTML, CSS, JavaScript, JSP, Servlet, thư viện, hình ảnh, font chữ, ...) vào trong thư mục WEB-INF và nén thư mục này thành file WAR.

● Triển khai web application trên Tomcat server: Ta cần cài đặt Tomcat server và sao chép file WAR vào thư mục webapps của Tomcat. Sau đó, ta khởi động Tomcat và truy cập vào địa chỉ http://localhost:8080/ten-web-app để kiểm tra xem web application có hoạt động đúng không.

● Cấu hình file web.xml: Đây là file cấu hình cho web application, cho phép ta thiết lập các Servlet, Filter, Listener, các tham số cho ứng dụng, ...

● Sử dụng JNDI và database: Ta có thể cấu hình kết nối tới database thông qua JNDI (Java Naming and Directory Interface) để tránh lộ thông tin cấu hình kết nối tới client. Ta cần tạo file context.xml trong thư mục META-INF và cấu hình kết nối tới database trong file này.

● Quản lý session và cookie: Session được sử dụng để lưu trữ thông tin giữa các request và response, giúp ứng dụng giữ được trạng thái của người dùng. Cookie là một cách để lưu trữ thông tin trên máy khách và được sử dụng để lưu trữ thông tin như đăng nhập, ngôn ngữ, ...

● Bảo mật web application: Ta cần phải bảo vệ web application khỏi các cuộc tấn công từ bên ngoài. Các kỹ thuật bảo mật như mã hóa SSL (Secure Sockets Layer), mã hóa cookie, tạo mật khẩu an toàn, sử dụng các framework bảo mật như Spring Security, ... cần được áp dụng để đảm bảo an toàn cho ứng dụng.

# **II. Bài tập**

## **1. Servlet**

### **Database: Sql server**

* Cấu hình trong file application.properties:

server.port=9090

spring.datasource.url=jdbc:sqlserver://localhost:1233;encrypt=false;databaseName=contact;integratedSecurity=false;

spring.datasource.username=long

spring.datasource.password=123

spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver

spring.jpa.show-sql=true

spring.jpa.hibernate.dialect=org.hibernate.dialect.SQLServer2012Dialect

spring.jpa.hibernate.ddl-auto = create-drop

* Thêm dependency:

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>mssql-jdbc</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

## **2. Spring Boot**