1. Tìm hiểu về công cụ và cách tổ chức chương trình trong Java Servlet

Servlet chính là công nghệ được dùng để thiết lập ra các ứng dụng web. Servlet được xem là một API cung cấp các interface, lớp và cả các tài liệu. Servlet cũng là một thành phần website được lập trình viên triển khai trên máy chủ, phục vụ cho mục đích tạo các trang web động.

Công cụ hỗ trợ lập trình Web với Servlet:

- Eclipse: đây là một IDE cho phép phát triển các ứng dụng phần mềm bằng nhiều ngôn ngữ
- NetBeans: một IDE để phát triển các ứng dụng phần mềm trong Java, JavaScript, PHP, Python, C / C ++, … NetBeans cũng là một khuôn khổ nền tảng có thể được sử dụng để phát triển ứng dụng máy tính để bàn trong Java

Môi trường hỗ trợ lập trình ứng dụng web với Servlet:

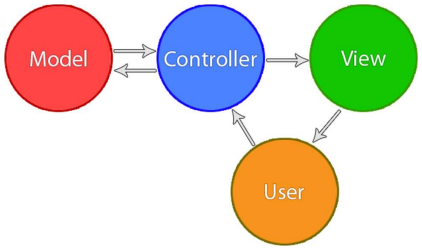
- Apache Tomcat (còn gọi là Jakarta Tomcat hoặc Tomcat): là một bộ chứa servlet (có nghĩa là một lớp Java hoạt động dưới sự nghiêm ngặt của API Servlet Java - một giao thức mà một lớp Java đáp ứng một yêu cầu http)
- Glassfish: Đây là một application server mã nguồn mở được phát triển dựa trên nền JavaEE hỗ trợ đầy đủ các tính năng cần thiết của JavaEE như web ứng dụng, Enterprise JavaBeans, JPA, JavaServer Faces, JMS, RMI, JavaServer Pages, Servlets, …

Servlet API chứa trong 4 gói:

- *javax.servlet*: chứa những lớp và interface mà định nghĩa quan hệ giữa một servlet và một servlet container
- *javax.servlet.http*: chứa những lớp và interface mà định nghĩa quan hệ giữa một HTTP servlet và một servelet container
- *javax.servlet.annotation*: chứa những chú thích đến những servlet, bộ lọc và listeners. Nó cũng chỉ định metadata cho những thành phần chú thích
- *javax.servlet.descriptor*: chứa những loại mà cung cấp truy cập hướng lập trình đến một thông tin cấu hình ứng dụng web

Mô hình MVC trong JAVA Servlet

MVC viết tắt của 3 từ đó là Model – View – Controller (MVC) là mẫu thiết kế nhằm mục tiêu chia tách phần Giao diện và Code để dễ quản lý, phát triển và bảo trì.



Mỗi phần có có một nhiệm vụ xử lý khác nhau, đối với trong mô hình MVC trong java nói riêng và mô hình MVC nói chung sẽ có:

- *Model*: Tương tác và truy xuất dữ liệu đến database (cơ sở dữ liệu)

- *View*: Giao diện mà người dùng có thể nhìn thấy, tuy nhiên thường view chỉ có một nhiệm vụ duy nhất là hiển thị dữ liệu. Trong Java thì các bạn nên giảm thiểu code Java vào file jsp nhé. Lý do tại sao mời các bạn đọc bài viết này.Trong Java web view chính là file jsp
- *Controller*: Nó có nhiệm vụ điều khiển tương tác giữa Model và View cũng như xử lý logic nghiệp vụ (Business). Có thể giải thích kỹ hơn nữa đối với trong Java thì controller lấy dữ liệu từ model sau đó gữi đến view. Trong Java web controller là file servlet

Viết một chương trình Servlet đơn giản

Cấu trúc danh mục ứng dụng

- Thư mục tên dự án ngoài cùng
- Bên trong có thư mục *WEB-INF*
- Bên trong thư mục *WEB-INF* có 2 thư mục con
    - *classes*: những lớp servlet và những lớp Java khác phải được đặt tại đây
    - *lib*: triển khai những file mà ứng dụng Servlet yêu cầu tại đây. File jar Servlet API không cần triển khai tại đây bởi vì Servlet container đã có bản sao này.

Một ứng dụng servlet/JSP thường có các trang JSP, tệp HTML, tệp hình ảnh và các tài nguyên khác. Chúng nên nằm trong thư mục ứng dụng và thường được sắp xếp trong các thư mục con. Chẳng hạn, tất cả các tệp hình ảnh có thể chuyển đến một thư mục hình ảnh, tất cả các trang JSP đến jsp, …

Expression Language (EL) trong JSP

JSP Epression 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 đơn giản cho JSP EL

Xác định một giá trị thuộc tính trong một thẻ JSP

```
<jsp:setProperty name="box" property="perimeter" value="100"/>
```

Xác định một Expression cho bất kỳ giá trị thuộc tính nào

Sử dụng các JSP EL bên trong Template Text cho một thẻ. Ví dụ, thẻ `<jsp:text>` chèn nội dung của nó bên trong phần thân của một JSP. Khai báo `<jsp:text>` sau chèn `<h1>Hello JSP!</h1>` vào trong JSP output:

```
<jsp:text>
<h1>Hello JSP!</h1>
</jsp:text>
```

Bao một JSP EL trong phần thân của một thẻ `<jsp:text>` hoặc bất kỳ thẻ khác) với cùng cú pháp ${} bạn sử dụng cho các thuộc tính

```
<jsp:text>
Box Perimeter is: 0
</jsp:text>
```

EL expression có thể sử dụng các dấu ngoặc đơn để nhóm các subexpression

Để vô hiệu hóa sự ước lượng của các EL expression, chúng ta xác định thuộc tính **isELIgnored** của page Directive như sau

```
<%@ page isELIgnored ="true|false" %>
```

Các giá trị hợp lệ của thuộc tính này là true hoặc false. Nếu nó là true, EL expression bị bỏ qua khi chúng xuất hiện trong các thuộc tính thẻ hoặc static text. Nếu nó là false, các EL expression được ước lượng bởi Container

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.

Hàm trong JSP EL

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:

```
${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.

Thư viện thẻ chuẩn - Standard Tag Library (JSTL) trong JSP

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

Cài đặt thư viện JSTL

Nếu bạn đang sử dụng Apache Tomcat container, thì bạn theo hai bước sau:

- Tải bản phân phối nhị phân từ Apache Standard Taglib và mở file nén đó.
- Để sử dụng Standard Taglib từ Jakarta Taglib, đơn giản bạn chỉ cần sao chép các JAR file trong thư mục 'lib' tới thư mục webapps\ROOT\WEB-INF\lib trong ứng dụng của bạn.

Để sử dụng bất kỳ thư viện nào, bạn phải bao một <taglib> tại trên cùng của mỗi JSP mà sử dụng thư viện đó.

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:

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

Nhóm Formatting Tags trong JSTL

Nhóm Formatting Tags trong JSTL được sử dụng để định dạng và hiển thị text, date, time và số ngôn ngữ trong Website. Sau đây là cú pháp cơ bản để include thư viện thẻ định dạng trong JSP:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/html/fmt" %>
```

Nhóm SQL Tags trong JSTL

Nhóm SQL Tags trong JSTL cung cấp các thẻ để tương tác với các Relational Database (RDBMSs), ví dụ như Oracle, MySQL, hoặc Microsoft SQL Server.

Đây là cú pháp để bao nhóm SQL Tags trong JSTL trong JSP:

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/html/sql" %>
```

Nhóm XML Tags trong JSTL

Nhóm XML Tags trong JSTL cung cấp một cách để tạo và thao tác các tài liệu XML. Sau đây là cú pháp để include nhóm XML Tags trong JSP:

```
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/html/xml" %>
```

Nhóm JSTL Functions

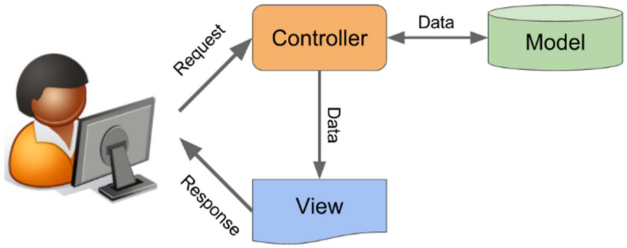
JSTL bao gồm một số hàm chuẩn, mà hầu hết là các hàm thao tác chuỗi phổ biến. Sau đây là cú pháp để bao nhóm JSTL Functions trong chương trình JSP:

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/html/functions" %>
```


3. Tìm hiểu về Lập trình JPA (Java Persistence API)

Khái niệm servlet: Servlet là một công nghệ lập trình Java dùng để tạo ra các ứng dụng web động. Servlet là một lớp Java được sử dụng để xử lý các yêu cầu HTTP từ phía máy khách và tạo ra các phản hồi tương ứng từ phía máy chủ. Servlet được chạy trên một máy chủ web và được quản lý bởi một container servlet.

Mô hình MVC



- Model(dữ liệu): Quản lí xử lí các dữ liệu
- View(giao diện): Nơi hiển thị dữ liệu cho người dùng
- Controller(bộ điều khiển): Điều khiển sự tương tác giữa 2 thành phần model và view

Khái niệm JPA(Java Persistence API):

- 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

Tổng quan về 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, …

Kiến trúc JPA

Kiến trúc JPA bao gồm ba thành phần chính là:

- Entity
- EntityManager
- EntityManagerFactory

Ngoài ra còn có, EntityTransaction, Persistence, Query

Entity

Khái niệm

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.

Entity Transaction

- 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.

Entity Manager: Entity Manager là một interface cung cấp các API cho việc tương tác với các Entity

Một số chức năng cơ bản của Entity Manager:

- 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

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

Application/Context scope

- Application scope được định nghĩa bởi javax.servlet.ServletContext interface
- Trong scope này data tồn tại và có thể truy cập từ các session, request khác nhau và chỉ bị hủy khi ứng dụng web bị shutdown
- Lấy đối tượng Application:
  - getServletContext()
  - getServletConfig().getServletContext()

Request scope

- Request scope bắt đầu 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 các servlet hoặc jsp cùng phục vụ trong 1 request. Ví dụ bạn gọi 1 servlet/jsp sau đó các servlet/jsp này lại gọi các servlet/jsp khác rồi mới trả về reponse
- Request object có sẵn trong JSP page như là 1 object ẩn . Bạn có thể set value cho 1 thuộc tính trong request object từ servlet và lấy nó ra ở JSP (Phải trong cùng 1 request)

Session scope

- 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
- 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 pages giống như một object ẩn
- Trong Servlet, bạn có thể lấy đối tượng object bằng cách gọi request.getSession()

JSP page scope

- Page scope 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

Listener

- Listener được tạo ra chủ yếu để lắng nghe các sự kiện (event) và phản hồi
- Listener trong Servlet: Context, Request, Attribute, Session, …

Cấu hình Listener (trong file web.xml)

```
<listener>
  <listener-class>
    listeners.AppContextListener
  </listener-class>
</listener>
```

FILTER

Filter (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.

Sử dụng Filter khi:

- Ghi lại tất cả các request đến
- Ghi nhật ký địa chỉ IP của máy tính từ các resquests nguồn
- Chuyển đổi
- Nén dữ liệu
- Mã hóa và giải mã
- Xác thực đầu vào, v.v.

Cấu hình Filter (trong file web.xml)

```
<filter>
 <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
 <filter-class>com.journaldev.servlet.filters.RequestLoggingFilter</filter-class> <!-- mandatory -->
 <init-param> <!-- optional -->
 <param-name>test</param-name>
 <param-value>testValue</param-value>
 </init-param>
</filter>
```

Authentication Filter

Chúng ta có thể thực hiện xác thực bằng Filter.

→ Thông qua ví dụ về xác thực khi đăng nhập

FilterConfig

Chúng ta có thể thực hiện xác thực bằng Filter.

→ Thông qua ví dụ về xác thực khi đăng nhập

5. Tìm hiểu về Layout và Đa ngữ (Internationalization - I18N)

Internationalization là tính năng của các ứng dụng web hiện đại, có khả năng hiển thị giao diện trên nhiều ngôn ngữ từ các quốc gia khác nhau. Ví dụ như nếu ứng dụng web có thể hiển thị tiếng Anh, tiếng Việt, tiếng Đức, tiếng Hàn, tiếng Nhật thì người dùng biết một trong các ngôn ngữ này, có thể chọn lựa ngôn ngữ mà họ rành nhất.

messageSource

messageSource là nơi chúng ta sẽ định nghĩa đường dẫn tới các tập tin chứa các chuỗi sẽ hiển thị trong ứng dụng web ở nhiều ngôn ngữ khác nhau, localeResolver sẽ xác định ngôn ngữ mà người dùng đang sử dụng là gì còn localeChangeInterceptor sẽ giúp detect việc người dùng có đang change ngôn ngữ mà họ muốn sử dụng không.

Spring cung cấp interface MessageSource để hiện thực mục đích của messageSource. Interface MessageSource này có một sub-interface là

HierarchicalMessageSource với 2 implementation là ReloadableResourceBundleMessageSource và ResourceBundleMessageSource.

ResourceBundleMessageSource implementation sử dụng ResourceBundle của Java, nó sẽ load hết tất cả các chuỗi được định nghĩa và không thể thay đổi nội dung của các chuỗi này trong suốt quá trình ứng dụng chạy. Còn ReloadableResourceBundleMessageSource implementation thì cũng tương tự như ResourceBundleMessageSource nhưng có thể cấu hình thêm cho nó tự reload lại nội dung của các chuỗi được định nghĩa sau một khoảng thời gian nào đó.

Để cấu hình MessageSource, định nghĩa các tập tin .properties chứa các chuỗi với nhiều ngôn ngữ khác nhau như khi làm việc với ResourceBundle trong Java.

Thêm mới 2 tập tin message.properties và message_vi.properties trong thư mục src/main/resources để định nghĩa cho ngôn ngữ mặc định của ví dụ là tiếng Anh, ngoài ra ứng dụng này còn hỗ trợ cả tiếng Việt.

Trong ứng dụng ví dụ ở trên, internationalization 2 câu là “Hello World” với “The time on the server is” nên sẽ định nghĩa nội dung của tập tin message.properties và message_vi.properties như sau:

message.properties

```
app.lang.vi=Vietnamese
app.lang.en=English
app.welcome=Hello world
app.time=The time on the server is
```

message_vi.properties

```
app.lang.vi=Ti\u1EBFng Vi\u1EC7t
app.lang.en=Ti\u1EBFng Anh
app.welcome=Xin chào
app.time=Gi\u1EDD c\u1EE7a server lúc này là
```

Sử dụng ReloadableResourceBundleMessageSource implementation và định nghĩa nó trong Spring container (tập tin src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml) như sau:

```
<beans:bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageS
ource">

<beans:property name="basename" value="classpath:i18n/messsage" />

</beans:bean>
```

Thuộc tính basename của class ReloadableResourceBundleMessageSource được dùng để khai báo đường dẫn tới các tập tin .properties.

localeResolver

localeResolver dùng để xác định ngôn ngữ mà người dùng đang sử dụng là gì để hiển thị cho đúng với ngôn ngữ đó. Spring sử dụng interface LocaleResolver để làm việc này.

Interface LocaleResolver có nhiều implementation khác nhau:

- AcceptHeaderLocaleResolver: resolver này sẽ detect ngôn ngữ mà người dùng đang sử dụng dựa vào header "accept-language" của HTTP request
- FixedLocaleResolver: dùng resolver này khi cố định locale cho ứng dụng
- SessionLocaleResolver: resolver này sử dụng attribute locale trong user session, nếu không có attribute này thì nó sẽ fallback sử dụng header "accept-language" của HTTP request như AcceptHeaderLocaleResolver

- CookieLocaleResolver: sử dụng locale được lưu trữ trong cookie của trình duyệt mà user đang sử dụng.

Sử dụng CookieLocaleResolver cho ví dụ:

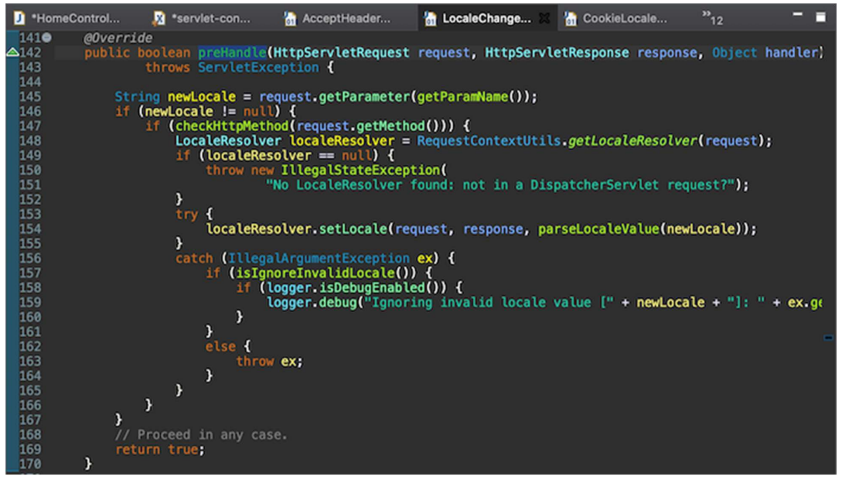
```
<beans:bean id="localeResolver"
class="org.springframework.web.servlet.i18n.CookieLocaleResolver"/>
```

localeChangeInterceptor

Khi người dùng muốn thay đổi ngôn ngữ hiển thị trên ứng dụng web, phải có cơ chế để xử lý yêu cầu này. LocaleChangeInterceptor là một class của Spring cho phép xử lý những yêu cầu này.

Class LocaleChangeInterceptor hiện thực interface HandlerInterceptor cho phép chặn tất cả request của user để thêm logic xử lý. Class LocaleChangeInterceptor sẽ lấy giá trị của một request parameter cấu hình để xử lý request change ngôn ngữ của người dùng.

Code của phương thức preHandle() trong class LocaleChangeInterceptor để thấy điều này:



6. Bố cục trang Web bằng Thymeleaf Layout

Thymeleaf

Thymeleaf là một Java Template Engine. Có nhiệm vụ xử lý và generate ra các file HTML, XML, …

Các file HMTL do Thymeleaf tạo ra là nhờ kết hợp dữ liệu và template + quy tắc để sinh ra một file HTML chứa đầy đủ thông tin.

Việc của bạn là cung cấp dữ liệu và quy định template như nào, còn việc dùng các thông tin đó để render ra HTML sẽ do Thymeleaf giải quyết.

Cú pháp

Cú pháp của Thymeleaf sẽ là một attributes (Thuộc tính) của thẻ HTML và bắt đầu bằng chữ th: .

Với cách tiếp cận này, bạn sẽ chỉ cần sử dụng các thẻ HTML cơ bản đã biết mà không cần bổ sung thêm syntax hay thẻ mới như JSP truyền thống.

Ví dụ:

Để truyền dữ liệu từ biến name trong Java vào một thẻ H1 của HTML.

```
<h1 th:text="${name}"></h1>
```

Chúng ta viết thẻ H1 như bình thường, nhưng không chứa bất cứ text nào trong thẻ. Mà sử dụng cú pháp th:text="${name}" để Thymeleaf lấy thông tin từ biến name và đưa vào thẻ H1.

Thuộc tính th:text biến mất và giá trị biến name được đưa vào trong thẻ H1.

Đó là cách Thymeleaf hoạt động.

Model & View Trong Spring Boot

Model là đối tượng lưu giữ thông tin và được sử dụng bởi Template Engine để generate ra webpage. Có thể hiểu nó là Context của Thymeleaf

Model lưu giữ thông tin dưới dạng key-value.

Trong template thymeleaf, để lấy các thông tin trong Model. bạn sẽ sử dụng Thymeleaf Standard Expression.

1. ${...}: Giá trị của một biến
2. *{...}: Giá trị của một biến được chỉ định (sẽ giải thích ở dưới)

Ngoài ra, để lấy thông tin đặc biệt hơn:

1. #{...}: Lấy message
2. @{...}: Lấy đường dẫn URL dựa theo context của server

${...} - Variables Expressions

Trên Controller bạn đưa vào một số giá trị:

```
model.addAttribute("today", "Monday");
```

Để lấy giá trị của biến today, tôi sử dụng ${...}


```
<p>Today is: <span th:text="${today}"></span>.</p>
```

Đoạn expression trên tương đương với:

```
ctx.getVariable("today");
```

**{...} - Variables Expressions on selections*

Dấu * còn gọi là asterisk syntax. Chức năng của nó giống với ${...} là lấy giá trị của một biến.

Điểm khác biệt là nó sẽ lấy ra giá trị của một biến cho trước bởi th:object

Còn ${...} sẽ lấy ra giá trị cục bộ trong Context hay Model.

Vậy đoạn code ở trên tương đương với:

```
<div>
<p>Name: <span th:text="${session.user.firstName}"></span>.</p>
<p>Surname: <span th:text="${session.user.lastName}"></span>.</p>
</div>
```

#{...} - Message Expression

Ví dụ, trong file config .properties của tôi có một message chào người dùng bằng nhiều ngôn ngữ.

```
home.welcome=¡Bienvenido a nuestra tienda de comestibles!
```

Thì cách lấy nó ra nhanh nhất là:

```
<p th:utext="#{home.welcome}">Xin chào các bạn!</p>
```

Đoạn text tiếng việt bên trong thẻ p sẽ bị thay thế bởi thymeleaf khi render #{home.welcome}.

@{...} - URL Expression

@{...} xử lý và trả ra giá trị URL theo context của máy chủ cho chúng ta.

```
<!-- tương đương với 'http://localhost:8080/order/details?orderId=3' -->
<a href="details.html"
   th:href="@{http://localhost:8080/order/details(orderId=${o.id})}">view</a>

<!-- tương đương '/order/details?orderId=3' -->
<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- tương dương '/gtvg/order/3/details' -->
<a href="details.html"
th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>
```

Nếu bắt dầu bằng dấu / thì nó sẽ là Relative URL và sẽ tương ứng theo context của máy chủ của bạn.

7. Cách tổ chức và chia sẻ dữ liệu của Spring Boot

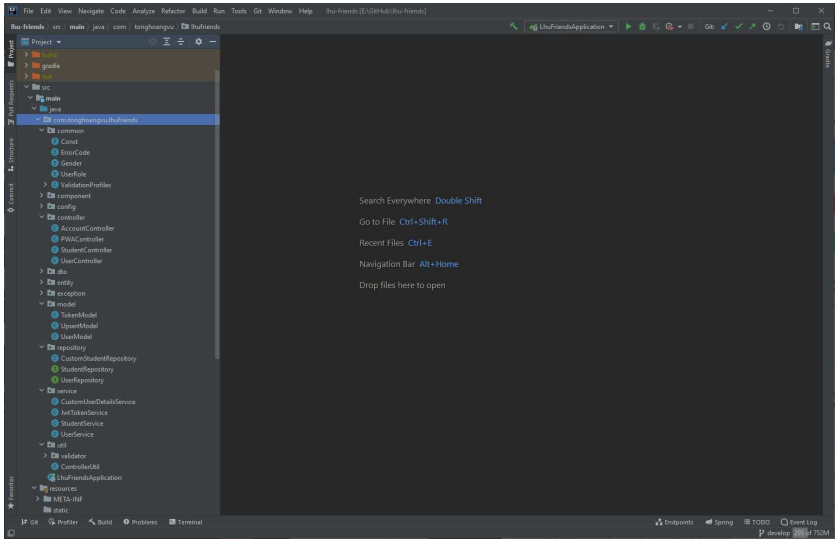
Cấu trúc chung của ứng dụng

Dù cho project được tạo với Maven hay Gradle thì cấu trúc chung vẫn tương tự nhau, do tuân theo một template có sẵn (tên là Archetype):

- Thư mục gốc chứa các file linh tinh như pom.xml (của Maven), build.gradle và các file khác như .gitignore,... dùng để cấu hình dự án
- Thư mục .mvn hoặc .gradle là thư mục riêng của Maven và Gradle, đừng nên đụng tới hay exclude nó ra khỏi source code
- Code được chứa trong thư mục src
- Thư mục build ra chứa các file class, file JAR. Với Maven là target còn Gradle là build

Như đã nói ở trên thì source code chứa trong thư mục src.

Chi tiết cấu trúc source code

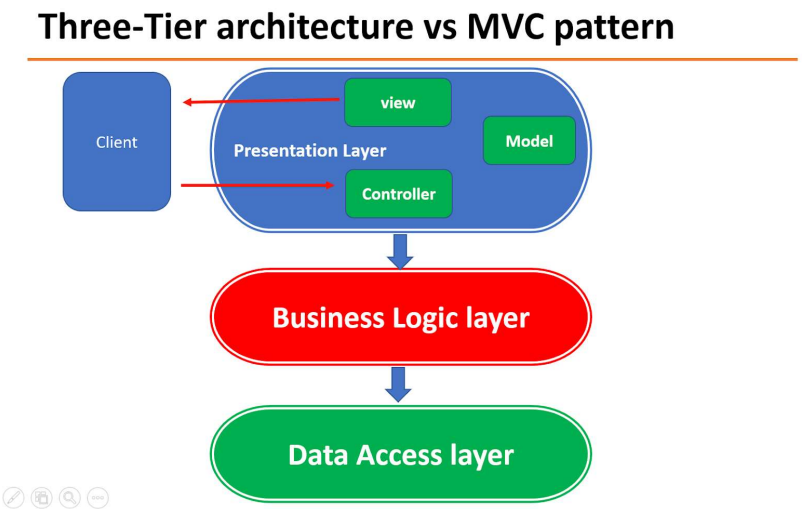


Như hình, thư mục chính cần quan tâm là src/main/java/<tên package>. Mọi code java đều nằm trong này:

- Tên package chính được đặt dạng ngược với tên miền. Ví dụ như tonghoangvu.com thì đặt thành com.tonghoangvu. Cộng thêm tên project nữa
- Có các package con, mỗi package đại diện cho các class thuộc layer cụ thể (ví dụ như service, controller,...)
- Thư mục resources chứa các tài nguyên của ứng dụng như hình ảnh, static file, properties file, …

Ngoài ra còn có src/test dùng để chứa các test class, dùng cho unit test.

Tổ chức source code theo mô hình 3 lớp



Tương ứng với từng thành phần của mô hình 3 lớp, thì chúng ta sẽ có các thư mục và cách đặt tên tương ứng:

- Controller layer: đặt trong controller, các class là controller sẽ có hậu tố Controller (ví dụ UserController, AuthController,...)
- Service layer: đặt trong service, các class có hậu tố là Service và thường tương ứng với controller (ví dụ UserService, …)
- Data access layer: ca này khó, bởi vì layer này bao gồm repository (đặt trong repository và hậu tố tương tự), DTO, model, entity... chi tiết mình sẽ nói ở các bài sau

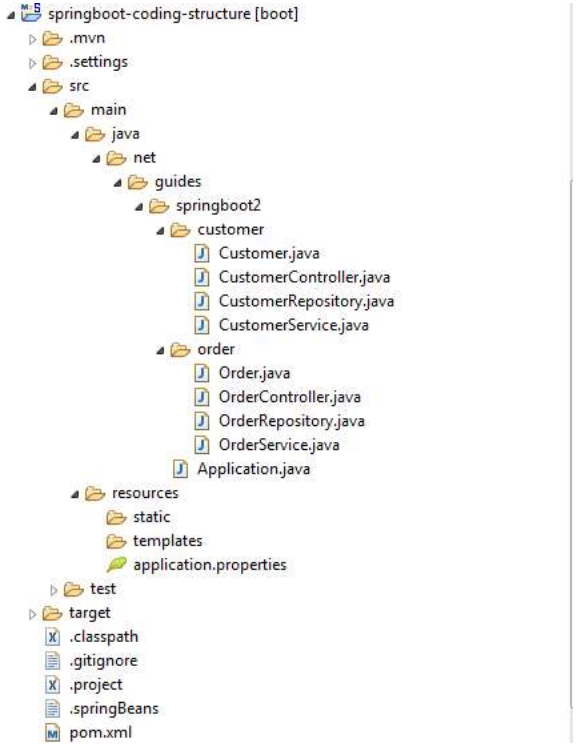
Ngoài ra, với các loại khác thì:

- util package chứa các lớp util (xử lý linh tinh), ví dụ như convert end date, tính toán đơn giản, …
- common package chứa các class định nghĩa như enum, interface, class dùng chung và đơn giản
- exception package chứa các class có nhiệm vụ xử lý exception trong Spring Boot
- component chứa các bean được định nghĩa còn lại nhưng không thuộc layer nào.

Xong, hầu hết ứng dụng Spring Boot có tới 80% cấu trúc tương tự. Khác biệt là rất nhỏ, chỉ đơn giản nằm trong cách đặt tên thôi.

Tổ chức code theo tính năng

Cách này có hơi khác với cách tổ chức theo mô hình 3 lớp ở trên. Cụ thể thay vì chia thành các package dựa theo layer, thì cách này chia theo tính năng. Nghĩa là mỗi tính năng, ví dụ user package thì sẽ chứa nào là UserController, UserService, …



8. Khái niệm IoC, DI (Dependency Injection) và Spring Bean

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 thục 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).

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) mà 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 điện 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)

Dependency injection (DI) nó là một hình thức cụ thể của Inverse of Control (Dependency Inverse) đã nói ở trên. DI thiết kế sao cho các dependency (phụ thuộc) của một đối tượng CÓ THỂ được đưa vào, tiêm vào đối tượng đó (Injection) khi nó cần tới (khi đối tượng khởi tạo). Cụ thể cần làm:

Xây dựng các lớp (dịch vụ) có sự phụ thuộc nhau một cách lỏng lẻo, và dependency có thể tiêm vào đối tượng (injection) - thường qua phương thức khởi tạo constructor, property, setter

Xây dựng được một thư viện có thể tự động tạo ra các đối tượng, các dependency tiêm vào đối tượng đó, thường là áp dụng kỹ thuật Reflection của C# (xem thêm lớp type): Thường là thư viện này quá phức tạp để tự phát triển nên có thể sử dụng các thư viện có sẵn.

Spring Bean

Trong Spring, các đối tượng tạo thành xương sống của ứng dụng của bạn và được quản lý bởi Spring IoC container được gọi là các bean. Bean là một đối tượng được khởi tạo, lắp ráp và quản lý bởi bộ chứa Spring IoC.

Trong Spring Boot, bạn có thể khai báo các Spring Bean bằng cách sử dụng các annotation như @Component, @Service, @Repository, @Controller, hoặc @Configuration. Sau đó, Spring Boot sẽ tự động quản lý việc khởi tạo và phân phối các Bean này đến các thành phần khác trong ứng dụng của bạn.

9. Spring Data JPA cơ bản và nâng cao

Giới thiệu

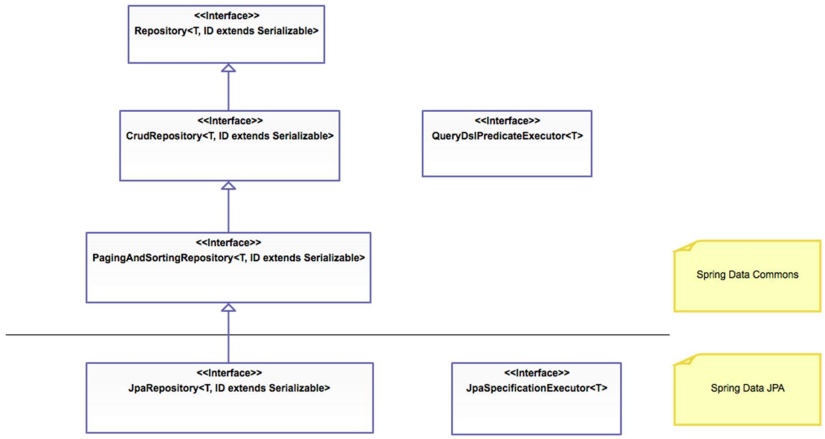
Việc xây dựng và triển khai tầng truy cập dữ liệu trong các ứng dụng đôi khi gây ra cho lập trình viên khá nhiều phiền toái do sự cồng kềnh và phức tạp của nó. Dù hiện nay đã có nhiều thư viện và framework hỗ trợ nhưng chúng ta vẫn không tránh khỏi việc phải viết lại quá nhiều mã viết sẵn trong các thư viện, và code của chúng ta cũng sẽ ngày một trở nên "rối loạn" hơn. Để giảm thiểu ảnh hưởng từ vấn đề trên, Spring cung cấp cho chúng ta giải pháp đó là các abstract reposity, chúng được thiết kế dựa trên các interface, giảm đáng kể các mã viết sẵn và ảnh hưởng từ việc phải thực thi quá nhiều các lớp trong tầng truy cập dữ liệu.

Core concepts

Phần cơ bản và cốt lõi nhất của Pring Data abstract repository đó là một interface có tên là Repository, nó nhận vào một Entity class tương ứng với java class đại diện cho một bảng trong database của chúng ta, và kiểu dữ liệu của trường ID trong bảng đó. Interface này cung cấp một số function xoay quanh việc CRUD thực thể mà nó quản lí.

Spring Data cũng cung cấp cho chúng ta một số interface với các tính năng phù hợp với từng mục đích, và chúng đều kế thừa từ interface Repository ở trên.

Một số interface Spring Data cung cấp sẵn cho chúng ta theo sơ đồ cây:



Query methods

Để tạo dựng được một abstract repository với Spring Data cơ bản sẽ gồm 4 bước sau:

- Khai báo một interface mở rộng từ một trong số các interface có sẵn tùy theo mục đích
- Khai báo các phương thức truy vấn trong interface
- Cấu hình cho Spring biết đây là một repository và khởi tạo nó
- Inject bean của repository và sử dụng

Định nghĩa repository interfaces

Ở bước đầu tiên khi tạo một repository, bạn sẽ phải định nghĩa một entity class để repository đó nhận biết và làm việc với nó. Repository sẽ nhận vào entity class và kiểu dữ liệu của ID trong entity class, sau đó, chúng ta sẽ có các CRUD method của Repository phù hợp với entity class.

Định nghĩa các query methods

Query lookup strategies

Tiếp theo, chúng ta sẽ cùng thảo luận về định nghĩa của các query methods. Đại khái sẽ có 2 cách để repository có thể nhận biết câu lệnh query tới database mà bạn muốn thông qua method name. Cách thứ nhất đó là lấy trực tiếp câu truy vấn từ tên phương thức trong repository. Cách thứ hai sẽ sử dụng thêm một số tùy chọn bổ sung từ Spring Data Framework để tạo câu truy vấn. Bạn có thể lựa chọn một trong 2 cách tùy thuộc vào mục đích thực tế, tuy nhiên Spring cũng đưa ra một số chiến lược để giúp bạn thực hiện điều đó, nó được gọi là qvc uery-lookup-strategy gồm 3 tùy chọn sau đây:

- **CREATE** - Nếu tùy chọn này được sử dụng thì Spring framework sẽ cố gắng tự động tạo một truy vấn từ tên phương thức truy vấn
- **USE_DECLARED_QUERY** - Đối với tùy chọn này Spring framework cố gắng tìm một truy vấn đã khai báo, có thể bằng các anotation hoặc khái báo bởi các phương tiện khác. Nếu repository không tìm được nó sẽ báo lỗi khi runtime
- **CREATE_IF_NOT_FOUND** (default) - Đây là tùy chọn mặc định và nó kết hợp CREATE và USE_DECLARED_QUERY. Nó tìm kiếm một truy vấn đã khai báo trước và nếu không tìm thấy truy vấn đã khai báo nào, nó sẽ tạo ra một truy vấn dựa trên tên phương thức tùy chỉnh

Query creation

Spring giúp chúng ta tạo ra các query method từ chính tên của các phương thức có trong repository, bắt đầu là với các tiền tố đại diện cho mục đích của câu lệnh query sẽ được thực hiện. Ví dụ như findBy, find, readBy, read, getBy, ..., và phần còn lại sẽ đại hiện cho các variable có trong entity class, chúng ta có thể nối chúng với các từ khóa như And, Or, ... như sau:

Do sự trợ giúp của các từ khóa, method name của chúng ta trở nên rõ ràng và dễ hiểu hơn, tuy nhiên trong một số trường hợp chúng ta muốn một biểu thức vẫn rõ nghĩa như vậy nhưng không thể sử dụng các từ khóa như đã kể trên, vậy làm sao để Spring có thể hiểu mục đích của chúng ta để tạo ra các query đúng với mục đích nhất? Chúng ta cùng đi đến ví dụ tiếp theo để cùng làm rõ cách Spring phân tích các biểu thức thuộc tính.

Query data

Spring cung cấp cho chúng ta anotation @Repository giúp chú thích class trở thành một repository, qua đó Spring có thể nhận biết vào tạo ra một repository bean cho chúng ta khi khởi chạy ứng dụng. Khi đó chúng ta chỉ cần inject bean đó vào và sử dụng.

10. Schedule Task, Interceptor và gửi mail

Schedule Task

Đôi khi viết chương trình chúng ta sẽ gặp những tình hướng thực hiện các chức năng chạy theo 1 lịch cố định: ví dụ cứ 5 phút gửi request 1 lần, cứ vào 23h đêm hàng ngày thì thực hiện chạy chức năng backup data, cứ 7h sáng chủ nhật hàng tuần thì tính toán và gửi báo cáo…

Sử dụng TimerTask của Java, các thư viện như Quartz… Trong spring, tích hợp sẵn cho chúng ta.

Cấu hình Scheduling Tasks trong Spring

Bật chức năng schedule trong Spring:

Với trường hợp cấu hình bằng annotation hay Spring Boot ta sử dụng annotation `@EnableScheduling`

```
@Configuration
@EnableScheduling
public class SpringConfig {
...
}


@SpringBootApplication
@EnableScheduling
public class Application {
...
}
```

Với trường hợp cấu hình bằng xml ta sử dụng thẻ sau:

```
<task:annotation-driven>
```

*Tạo task/job chạy theo lịch với annotation **@Schedule***

Tạo schedule task với **`fixedDelay`**

```
@Scheduled(fixedDelay = 1000)
public void scheduleFixedDelayTask() throws InterruptedException {
System.out.println("Task1 - " + new Date());
}
```

- Cứ sau khoảng thời gian `fixedDelay` thì nó lại chạy một lần, ví dụ ở trên thì cứ sau 1000ms (1 giây) thì nó lại chạy method `scheduleFixedDelayTask` một lần
- Với `fixedDelay` thì chỉ khi nào task trước đó thực hiện xong thì nó mới chạy tiếp task đó lại lần nữa. Ví dụ sau 1 giây mà method `scheduleFixedDelayTask` chưa chạy xong thì nó sẽ chờ cho tới khi nào xong mới chạy lại lần tiếp theo

Tạo schedule task với **`fixedRate`**

```
@Scheduled(fixedRate = 1000)
public void scheduleFixedRateTask() throws InterruptedException {
System.out.println("Task2 - " + new Date());
}
```

- `fixedRate` thì giống với `fixedDelay`, tuy nhiên cứ sau khoảng thời gian `fixedRate` thì nó chạy tiếp 1 lần nữa mà không cần quan tâm lần chạy trước đã hoàn thành chưa.
- Ví dụ sau 1s mà method `scheduleFixedRateTask` chưa thực hiện xong thì nó vẫn chạy lần tiếp theo.

Tạo schedule với **`cron`**

Với cron ta sẽ sử dụng cron expression để định nghĩa lịch chạy.

Bằng cron ta có thể định nghĩa thời gian chạy theo giờ, phút, giây, ngày tháng năm, trong khoảng thời gian nào… do đó việc đặt lịch linh hoạt hơn so với `fixedDelay` và `fixedRate` rất nhiều

Ví dụ: từ giây thứ 5 đến giây thứ 10 trong khoảng thời gian 12h-14h các ngày từ thứ 2 đến thứ 6, cứ 1 giây lặp lại một lần

```
@Scheduled(cron = "5-10/1 * 12-14 * * MON-FRI")
public void scheduleTaskUsingCronExpression() throws InterruptedException {
System.out.println("Task3 - " + new Date());
}
```

Ngoài việc sử dụng annotation @Scheduled bạn cũng có thể cấu hình trong file xml như sau:

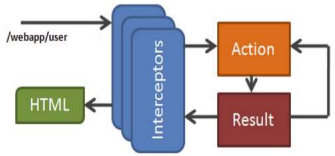
```
<beans>
<bean name="beanA" class="your_class"/>
<bean name="beanB" class="your_class"/>
<bean name="beanC" class="your_class"/>
</beans>

<!-- Configure parameters -->
<task:scheduled-tasks scheduler="myScheduler">
<task:scheduled ref="beanA" method="scheduleFixedDelayTask"
fixed-delay="1000" initial-delay="1000" />
<task:scheduled ref="beanB" method="scheduleFixedRateTask"
fixed-rate="1000" />
<task:scheduled ref="beanC" method="scheduleTaskUsingCronExpression"
cron="*/5 * * * * MON-FRI" />
</task:scheduled-tasks>
```

Interceptor

Trong **Spring**, khi một request được gửi đến controller, trước khi request được xử lý bởi **Controller**, nó phải vượt qua các **Interceptor** (0 hoặc nhiều).

Spring Interceptor là một khái niệm khá giống với **Servlet Filter**.

Spring Interceptor chỉ áp dụng đối với các request đang được gửi đến một **Controller**.



Bạn có thể sử dụng **Interceptor** trong Spring Boot để làm một số việc như:

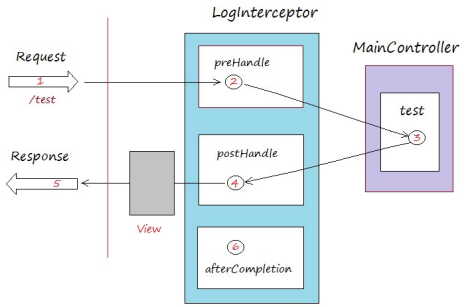
- Trước khi gửi request tới Controller
- Trước khi gửi response tới Client
- Ghi lại Log.

Lớp **Interceptor** của bạn cần phải thực hiện interface **org.springframework.web.servlet.HandlerInterceptor** hoặc mở rộng từ lớp **org.springframework.web.servlet.handler.HandlerInterceptorAdapter**.

Với ví dụ này, bạn có thể sử dụng 1 interceptor để thêm request header trước khi gửi request tới Controller và thêm response header trước khi gửi tới Client.

Để làm việc với Interceptor, bạn cần tạo class **@Component** và được implement từ interface **HandlerInterceptor**.

Sau đây là 3 method bạn nên biết khi làm việc với Interceptor:



- **preHandle()** method: method này sử dụng để thực hiện các operations trước khi gửi request tới Controller. Method này trả về true rồi trả response cho Client
- **postHandle()** method: method này sử dụng để thực hiện các operations trước khi gửi request tới Client
- **afterCompletion()** method: method này được sử dụng để thực hiện các operations sau khi hoàn thành việc gửi request và response.

Quan sát đoạn code dưới đây để hiểu rõ hơn.

```
@Component
public class ProductServiceInterceptor implements HandlerInterceptor {
   @Override
   public boolean preHandle(
```



```
      HttpServletRequest request, HttpServletResponse response, Object handler)
throws Exception {

      return true;
   }
   @Override
   public void postHandle(
      HttpServletRequest request, HttpServletResponse response, Object handler,
      ModelAndView modelAndView) throws Exception {}

   @Override
   public void afterCompletion(HttpServletRequest request, HttpServletResponse
response,
      Object handler, Exception exception) throws Exception {}
}
```

Bạn sẽ phải đăng ký Interceptor này với **InterceptorRegistry** bằng cách sử dụng **WebMvcConfigurerAdapter** như hình dưới đây.

```
@Component
public class ProductServiceInterceptorAppConfig extends WebMvcConfigurerAdapter
{
   @Autowired
   ProductServiceInterceptor productServiceInterceptor;

   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(productServiceInterceptor);
   }
}
```

Trong ví dụ dưới đây, chúng ta sẽ GET ra API của product.

Class ProductServiceInterceptor.java.

```
package com.tutorialspoint.demo.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
```

```
import org.springframework.web.servlet.ModelAndView;

@Component
public class ProductServiceInterceptor implements HandlerInterceptor {
   @Override
   public boolean preHandle
      (HttpServletRequest request, HttpServletResponse response, Object handler)
      throws Exception {

      System.out.println("Pre Handle method is Calling");
      return true;
   }
   @Override
   public void postHandle(HttpServletRequest request, HttpServletResponse
response,
      Object handler, ModelAndView modelAndView) throws Exception {

      System.out.println("Post Handle method is Calling");
   }
   @Override
   public void afterCompletion
      (HttpServletRequest request, HttpServletResponse response, Object
      handler, Exception exception) throws Exception {

      System.out.println("Request and Response is completed");
   }
}
```

File class Application Configuration để đăng ký Interceptor vào Interceptor Register – ProductServiceInterceptorAppConfig.java.

```
package com.tutorialspoint.demo.interceptor;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
```



```
@Component
public class ProductServiceInterceptorAppConfig extends WebMvcConfigurerAdapter
{
   @Autowired
   ProductServiceInterceptor productServiceInterceptor;

   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(productServiceInterceptor);
   }
}
```

File class ProductServiceController.java.

```
package com.tutorialspoint.demo.controller;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import com.tutorialspoint.demo.exception.ProductNotfoundException;
import com.tutorialspoint.demo.model.Product;

@RestController
public class ProductServiceController {
   private static Map<String, Product> productRepo = new HashMap<>();
   static {
      Product honey = new Product();
      honey.setId("1");
      honey.setName("Honey");
      productRepo.put(honey.getId(), honey);
      Product almond = new Product();
```

```
      almond.setId("2");
      almond.setName("Almond");
      productRepo.put(almond.getId(), almond);
   }
   @RequestMapping(value = "/products")
   public ResponseEntity<Object> getProduct() {
      return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);
   }
}
```

Model Product.

```
package com.tutorialspoint.demo.model;

public class Product {
   private String id;
   private String name;

   public String getId() {
      return id;
   }
   public void setId(String id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

Hàm main Spring Boot.

```
package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```



```
@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Maven build – **pom.xml**.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<project xmlns = "http://maven.apache.org/POM/4.0.0" xmlns:xsi = "
   http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">

   <modelVersion>4.0.0</modelVersion>
   <groupId>com.tutorialspoint</groupId>
   <artifactId>demo</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>demo</name>
   <description>Demo project for Spring Boot</description>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.8.RELEASE</version>
      <relativePath/>
   </parent>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
      <java.version>1.8</java.version>
   </properties>

   <dependencies>
      <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
         </dependency>

         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
         </dependency>
      </dependencies>

      <build>
         <plugins>
            <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
         </plugins>
      </build>
   </project>
```

Gradle Build – **build.gradle**.

```
   buildscript {
      ext {
         springBootVersion = '1.5.8.RELEASE'
      }
      repositories {
         mavenCentral()
      }
      dependencies {
         classpath("org.springframework.boot:spring-boot-gradle-
plugin:${springBootVersion}")
      }
   }
   apply plugin: 'java'
   apply plugin: 'eclipse'
   apply plugin: 'org.springframework.boot'
```



```
   group = 'com.tutorialspoint'
   version = '0.0.1-SNAPSHOT'
   sourceCompatibility = 1.8

   repositories {
      mavenCentral()
   }
   dependencies {
      compile('org.springframework.boot:spring-boot-starter-web')
      testCompile('org.springframework.boot:spring-boot-starter-test')
   }
```

Bạn có thể tạo file JAR và run Spring Boot bằng cách sử dụng lệnh của Maven hoặc Gradle.

Đối với Maven:

```
mvn clean install
```

Sau khi "BUILD SUCCESS", bạn có thể tìm thấy file JAR trong thư mục đích.

Còn đối với Gradle:

Sau khi "BUILD SUCCESS", bạn có thể tìm thấy file JAR trong thư mục build/libs.

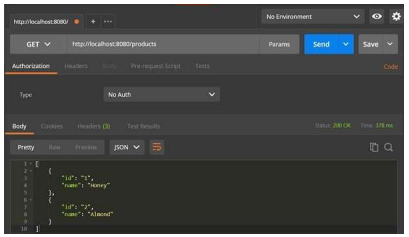
Run file JAR:

```
java –jar <JARFILE>
```

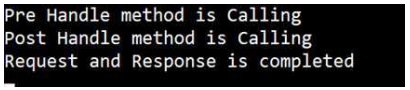
Thao tác này sẽ khởi động cổng ứng dụng Tomcat 8080 như hình dưới.

Started Application on Tomcat Port 8080

Paste link **http://localhost:8080/products** vào POSTMAN, chọn GET rồi gửi request.



Trong Console window, bạn có thể thêm các câu lệnh System.out.println trong Interceptor để được hiển thị như hình dưới nhé.



Spring Email

Spring Email là gì?

Giới thiệu về Spring Email

Khái niệm: Spring Email là một thư viện trong Spring Framework, cung cấp cho các nhà phát triển các lớp và công cụ hỗ trợ để tạo và gửi email trong ứng dụng Spring Boot Java. Thư viện này sử dụng các giao thức SMTP và MIME để tạo và gửi các email, cho phép chúng ta đính kèm các tệp đính kèm, tùy chỉnh nội dung email và gửi email đến nhiều người nhận cùng lúc.

Tính năng: Một số tính năng cơ bản của Spring Email:

- Xây dựng nội dung email: Spring Email cung cấp cho chúng ta lớp MimeMessageHelper để dễ dàng tạo nội dung email, cho phép đính kèm tệp tin, chèn hình ảnh, định dạng nội dung email theo HTML hoặc văn bản thuần túy
- Gửi email đến nhiều người nhận: Chúng ta có thể gửi email đến nhiều người nhận cùng lúc thông qua việc chỉ định danh sách địa chỉ email của các người nhận
- Tính năng đa luồng: Spring Email hỗ trợ tính năng đa luồng để xử lý các nhiệm vụ gửi email một cách hiệu quả và không làm chậm hoạt động của ứng dụng
- Tính năng tùy chỉnh cao: Spring Email cho phép chúng ta tùy chỉnh các thông tin về SMTP server, cấu hình TLS/SSL, xác thực, thời gian chờ kết nối, các thông tin xác thực tài khoản email, v.v. để phù hợp với yêu cầu của ứng dụng.

Ứng dụng: Spring Email được sử dụng rộng rãi trong các ứng dụng web để gửi các email thông báo đến người dùng, như thông báo đặt hàng, thông báo kích hoạt tài khoản, v.v. Ngoài ra, nó cũng được sử dụng trong các ứng dụng quản lý email như các ứng dụng webmail hoặc các công cụ quản lý email khác. Với tính năng tùy chỉnh cao, Spring Email cũng có thể được sử dụng để gửi email tùy chỉnh trong các ứng dụng phát triển khác như ứng dụng máy chủ hoặc ứng dụng máy tính cá nhân.

Tầm quan trọng của Spring Email trong ứng dụng web?

Với tầm quan trọng của email trong giao tiếp trực tuyến hiện nay, Spring Email có tầm quan trọng rất lớn trong ứng dụng web. Dưới đây là một số tầm quan trọng của Spring Email trong ứng dụng web:

Gửi thông báo cho người dùng: Trong ứng dụng web, chúng ta thường cần gửi email thông báo cho người dùng để cập nhật về các sự kiện quan trọng, ví dụ như đăng ký tài khoản, kích hoạt tài khoản, thông báo đặt hàng, v.v. Spring Email cung cấp các công cụ để tạo và gửi các email thông báo này một cách dễ dàng và nhanh chóng.

Gửi email xác nhận: Khi người dùng đăng ký tài khoản hoặc thực hiện một hành động quan trọng, chúng ta thường cần gửi email xác nhận để đảm bảo tính xác thực và bảo mật cho ứng dụng. Spring Email cung cấp các công cụ để tạo và gửi email xác nhận một cách dễ dàng và đáng tin cậy.

Gửi email marketing: Trong ứng dụng web, chúng ta có thể sử dụng email để quảng cáo sản phẩm hoặc dịch vụ đến khách hàng. Spring Email cung cấp các công cụ để tạo và gửi các email marketing này một cách dễ dàng và tiện lợi.

Quản lý email: Nếu ứng dụng web cần tích hợp chức năng quản lý email như các ứng dụng webmail hoặc các công cụ quản lý email khác, Spring Email là một lựa chọn tuyệt vời để hỗ trợ việc

này. Chúng ta có thể sử dụng Spring Email để tạo và gửi email, cũng như quản lý các email đã gửi hoặc nhận.

→ Với các tính năng tiên tiến và tùy chỉnh cao, Spring Email giúp các nhà phát triển ứng dụng web xây dựng và triển khai các tính năng email một cách dễ dàng và hiệu quả.

Cài đặt và cấu hình môi trường để sử dụng Spring Email

Bước 1: Tạo project.

Đầu tiên các bạn cần tạo mới một dự án mới, bạn có thể tham khảo các bước tạo project trên phần mềm Eclipse qua các bài trước.

Hoặc bạn có thể tạo project trên trang chủ Spring Boot Initializr hoặc sử dụng các IDE như Eclipse, IntelliJ IDEA, hoặc NetBeans để tạo project.

Bước 2: Thêm dependency Spring Email

Thêm dependency của Spring Email vào file pom.xml để có thể sử dụng được Spring Email:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Bước 2: Cấu hình file

Cấu hình thông tin kết nối SMTP server trong file **application.properties** hoặc **application.yml**.

Ví dụ:

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=example@gmail.com
spring.mail.password=examplepassword
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Lưu ý: Chúng ta có thể thay đổi thông tin kết nối SMTP server tùy theo nhu cầu sử dụng.

Sử dụng Spring Email

Tạo một đối tượng EmailSender và cấu hình nó trong Spring Boot

Để cấu hình EmailSender trong Spring Boot thì chúng ta cần add dependency và cấu hình thông tin kết nối như phần trên mình vừa hướng dẫn.

Bước 1: Tạo đối tượng EmailSender

Để tạo đối tượng EmailSender, bạn có thể sử dụng @Autowired để inject dependency của EmailSender vào trong class của bạn.

Ví dụ:

```
@Service
public class MyEmailService {

    @Autowired
    private JavaMailSender javaMailSender;

    // ...
}
```

Bước 2: Sử dụng đối tượng EmailSender để gửi email

Sau khi tạo đối tượng EmailSender, bạn có thể sử dụng các phương thức của nó để gửi email.

Ví dụ:

```
@Service
public class MyEmailService {

    @Autowired
    private JavaMailSender javaMailSender;

    public void sendEmail(String to, String subject, String text) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(text);
        javaMailSender.send(message);
    }
}
```

Trong ví dụ trên, chúng ta đã tạo một đối tượng SimpleMailMessage để lưu trữ các thông tin cần thiết để gửi email, sau đó sử dụng phương thức javaMailSender.send() để gửi email.

Tạo một đối tượng MimeMessagePreparator để thiết lập nội dung email

Để tạo một đối tượng **MimeMessagePreparator** trong Spring Boot để thiết lập nội dung email, bạn có thể sử dụng đối tượng MimeMessageHelper để tạo email có định dạng **MIME**. MimeMessageHelper cung cấp các phương thức để thiết lập các thuộc tính của email như địa chỉ email người gửi, địa chỉ email người nhận, tiêu đề email và nội dung email.

Ví dụ, để tạo một đối tượng MimeMessagePreparator với nội dung email là một chuỗi HTML, bạn có thể sử dụng đoạn mã sau:


```
MimeMessagePreparator messagePreparator = mimeMessage -> {
    MimeMessageHelper messageHelper = new MimeMessageHelper(mimeMessage, true, "UTF-8");
    messageHelper.setFrom("sender@example.com");
    messageHelper.setTo("recipient@example.com");
    messageHelper.setSubject("Test email");
    String htmlContent = "<html><body><h1>This is a test email</h1></body></html>";
    messageHelper.setText(htmlContent, true);
};
```

Lưu ý rằng trong ví dụ trên, chúng ta đã sử dụng charset là UTF-8 để đảm bảo rằng email có thể hiển thị đúng các ký tự Unicode.

Sau khi tạo đối tượng MimeMessagePreparator, bạn có thể sử dụng nó để gửi email bằng cách sử dụng phương thức send() của JavaMailSender:

```
javaMailSender.send(messagePreparator);
```

Khi gửi email, JavaMailSender sẽ sử dụng thông tin cấu hình được cung cấp trong file **application.properties** hoặc **application.yml** để thiết lập kết nối SMTP và gửi email.

11. Lambda Expression trong Java

Lambda Expression

- 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

Functional Interface

- Functional interface là một interface mà chỉ có một phương thức abstract duy nhất
- Java cung cấp một anotation @FunctionalInterface, được sử dụng để chỉ định một interface là Functional interface

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

Lợi ích

- Thực thi Functional interface.
- Viết ít code hơn.

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.

```
@FunctionalInterface
public interface Foo {
    String method(String string);
}
public String add(String string, Foo foo) {
    return foo.method(string);
}
Foo foo = parameter -> parameter + " from lambda"; String result =
useFoo.add("Message ", foo);
```

☺

```
public String add(String string, Function<String, String> fn) {
```



```
    return fn.apply(string);
}
Function<String, String> fn = parameter -> parameter + " from lambda";
String result = useFoo.add("Message ", fn);
```

@FunctinonalInterface ANOTATION

- Khi sử dụng anotation @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

```
public interface Foo {
    String method();
}
```

☺

```
@FunctionalInterface
public interface Foo {
    String method();
}
```

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.

```
@FunctionalInterface
public interface Foo {
    String method(String string);
    default void defaultMethod() {}
}
@FunctionalInterface
public interface FooExtended extends Baz, Bar {}
@FunctionalInterface
public interface Baz {
    String method(String string);
    default String defaultBaz() {} }
```

☺

```
@FunctionalInterface
```

```
public interface Bar {
    String method(String string);
    default String defaultBar() {} }
```

Lỗi Default Method trong Functional Interface

Lỗi: interface FooExtended inherits unrelated defaults for defaultCommon() from types Baz and Bar...

```
@FunctionalInterface
public interface Baz {
    String method(String string);
    default String defaultBaz() {}
    default String defaultCommon(){}
}
@FunctionalInterface
public interface Bar {
    String method(String string);
    default String defaultBar() {}
    default String defaultCommon(){}
}
```

☺

```
@FunctionalInterface
public interface FooExtended extends Baz, Bar {
    @Override
    default String defaultCommon() {
        return Bar.super.defaultCommon();
    }
}
```

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

```
private String value = "Enclosing scope value";
public String scopeExperiment() {
    Foo fooIC = new Foo() {
        String value = "Inner class value";
        @Override public String method(String string) {
            return this.value;
        }
};
String resultIC = fooIC.method("");
Foo fooLambda = parameter -> {
    String value = "Lambda value";
    return this.value;
};
String resultLambda = fooLambda.method("");
return "Results: resultIC = " + resultIC + ", resultLambda = " +
resultLambda; }
```

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.

```
Foo foo = parameter -> {
    String result = "Something " + parameter;
    //many lines of code
    return result;
};
```

☺

```
Foo foo = parameter -> buildString(parameter);
private String buildString(String parameter) {
```

```
    String result = "Something " + parameter;
    //many lines of code
    return result;
}
```

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.

```
(String a, String b) -> a.toLowerCase() + b.toLowerCase();
```

☺

```
(a, b) -> a.toLowerCase() + b.toLowerCase();
```

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

```
(a) -> a.toLowerCase();
```

☺

```
a -> a.toLowerCase();
```

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.

```
a -> {
    return a.toLowerCase()
};
```

☺

```
a -> a.toLowerCase();
```

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

```
public void method() {
    String localVariable = "Local";
    Foo foo = parameter -> {
        String localVariable = parameter;
        return localVariable;
    };
}
```

☺

```
Lỗi: Variable 'localVariable' is already defined in the scope.
```

Tránh lỗi "lạ" xảy ra

- Mô hình "hiệu quả cuối cùng" giúp ích rất nhiều ở đây, nhưng không phải trong mọi trường hợp. Lambdas không thể thay đổi giá trị của một đối tượng khỏi phạm vi kèm theo
- Đoạn mã dưới đây là hợp lệ, vì biến total vẫn là "effectively final", nhưng liệu đối tượng mà nó tham chiếu có cùng trạng thái sau khi thực thi lambda không? KHÔNG!

```
int[] total = new int[1];
Runnable r = () -> total[0]++;
r.run();
```

12. JSP Template Engine

JSP Là Gì?

- JSP(JavaServerPages): là một công nghệ để phát triển các trang web
- Thành phần JavaServer Pages 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

Tại Sao 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 dựng 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

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

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 máy Windows::

%CATALINA_HOME%\bin\startup.bat

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

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

Form Processing

Các phương pháp trong xử lý biểu mẫu

- Phương thức GET
- Phương pháp POST

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 Cookies Processing

Cấu trúc cookie

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

Setting Cookies JSP

- Bước 1: Tạo đối tượng Cookie
- Bước 2: Set maximum age
- Bước 3: Gửi Cookie vào header phản hồi HTTP

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()

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()

13. Thymeleaf Template

Khái niệm

Thymeleaf là một công cụ mạnh mẽ và linh hoạt được sử dụng để phát triển giao diện người dùng (UI) trong các ứng dụng web Java.

Nó là một công nghệ template engine, cho phép xây dựng các template HTML, XML hoặc văn bản thông qua việc kết hợp các mã thymeleaf vào trong các file template này.

Thymeleaf được tích hợp vào ứng dụng web Java thông qua một template engine. Nhiệm vụ của template engine là xử lý các template và thực hiện thay thế các đoạn mã thymeleaf bằng dữ liệu thực tế trong quá trình render.

Thymeleaf templates là một phần quan trọng trong việc phát triển giao diện người dùng trong các ứng dụng web Java.

Ưu điểm

- Cú pháp dễ đọc và hiểu
- Tích hợp tốt với Spring Framework
- Linh hoạt và mạnh mẽ
- Render trực tiếp và xem trước
- Hỗ trợ quốc tế hóa
- Cộng đồng phát triển và tài liệu phong phú

Nhược điểm

- Hiệu suất chậm hơn so với các công nghệ template khác
- Phụ thuộc vào Spring Framework
- Đòi hỏi kiến thức HTML và CSS
- Khả năng mở rộng có hạn
- Khó khăn trong việc kiểm tra và gỡ lỗi

Đặc điểm chính

- Tích hợp dữ liệu
- Xử lý sự kiện
- Lặp lại dữ liệu
- Quản lý layout template
- Tích hợp với Spring Framework

Cú pháp

- Thymeleaf sử dụng cú pháp đơn giản và dễ hiểu, tương tự với cú pháp của HTML.
- Cú pháp của Thymeleaf sẽ là một attributes của thẻ HTML và bắt đầu bằng chữ "th:"
- Với cách tiếp cận này, chỉ cần sử dụng các thẻ HTML cơ bản đã biết mà không cần bổ sung thêm syntax hay thẻ mới như JSP truyền thống

Tích hợp Spring Framework

Thymeleaf được thiết kế đặc biệt để tích hợp tốt với Spring Framework và cung cấp tích hợp sẵn với nhiều tính năng của Spring:

- Dependency
- Cấu hình Thymeleaf
- Sử dụng Thymeleaf trong Spring MVC
- Sử dụng Thymeleaf trong templates

Dependency

Cần thêm các dependency cần thiết vào file pom.xml để sử dụng Thymeleaf và Spring Framework.

Cấu hình Thymeleaf:

Cần cấu hình Thymeleaf trong file cấu hình của Spring (thông thường là file application.properties hoặc application.yml).

Sử dụng Thymeleaf trong Spring MVC:

Có thể trả về tên template và dữ liệu cho template trong model của Spring.

Sử dụng Thymeleaf trong template:

Có thể sử dụng cú pháp Thymeleaf để truy cập và hiển thị dữ liệu được truyền từ controller.

Tích hợp CSS

- Đảm bảo rằng đã định nghĩa các tệp CSS và đặt chúng trong thư mục tài nguyên của dự án (thông thường là thư mục resources/static/css/).
- Trong template Thymeleaf, sử dụng thuộc tính *th:href* để chỉ định đường dẫn đến tệp CSS và thuộc tính rel để chỉ định kiểu liên kết.

Tích hợp JS

- Đảm bảo rằng bạn đã định nghĩa các tệp JavaScript của bạn và đặt chúng trong thư mục tài nguyên của dự án (thông thường là thư mục resources/static/js/).
- Trong template Thymeleaf, sử dụng thuộc tính th:src để chỉ định đường dẫn đến tệp JavaScript và thuộc tính type để chỉ định kiểu tệp.

Truyền dữ liệu từ Controller vào Template

Để truyền dữ liệu từ Controller vào Template trong Thymeleaf, có thể sử dụng đối tượng Model để chứa và truyền dữ liệu.

Hiển thị dữ liệu trong Template

Trong Template Thymeleaf, có thể truy cập và hiển thị dữ liệu từ Model bằng cách sử dụng cú pháp Thymeleaf.

Binding dữ liệu đối tượng

Ngoài việc binding dữ liệu cơ bản, Thymeleaf cũng hỗ trợ binding dữ liệu từ đối tượng (object). Có thể truy cập các thuộc tính và phương thức của đối tượng trong template.

Xử lý sự kiện

Trong Thymeleaf, có thể xử lý sự kiện trên các thành phần HTML bằng cách sử dụng các thuộc tính và các cú pháp cung cấp bởi Thymeleaf.

Template Layouts

Trong Thymeleaf, template layout được sử dụng để tạo cấu trúc và giao diện chung cho nhiều trang trong ứng dụng web. Template layout cho phép xác định các phần chung như header, footer, thanh điều hướng và áp dụng chúng cho các trang cụ thể. Điều này giúp giảm sự trùng lặp và quản lý giao diện một cách hiệu quả.

So sánh Thymeleaf Template và JSP Template Engine

Tùy thuộc vào yêu cầu và môi trường phát triển, cả Thymeleaf và JSP đều có ưu điểm và giới hạn riêng. Thymeleaf thường được ưa chuộng hơn trong các dự án phát triển web Java hiện đại nhờ cú pháp HTML dễ đọc, tích hợp tốt với HTML và khả năng xử lý trên máy chủ linh hoạt. Việc chọn template engine phụ thuộc vào nhiều yếu tố như kỹ năng của nhóm phát triển, yêu cầu dự án, và sự tương thích với công nghệ hiện có.

Điểm hay của Thymeleaf Template

- Cú pháp thuần HTML
- Tích hợp tốt với HTML
- Xử lý tại máy chủ (Server-side rendering)
- Hỗ trợ tích hợp tốt
- Công cụ mạnh mẽ
- Hiệu suất và caching

14. 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

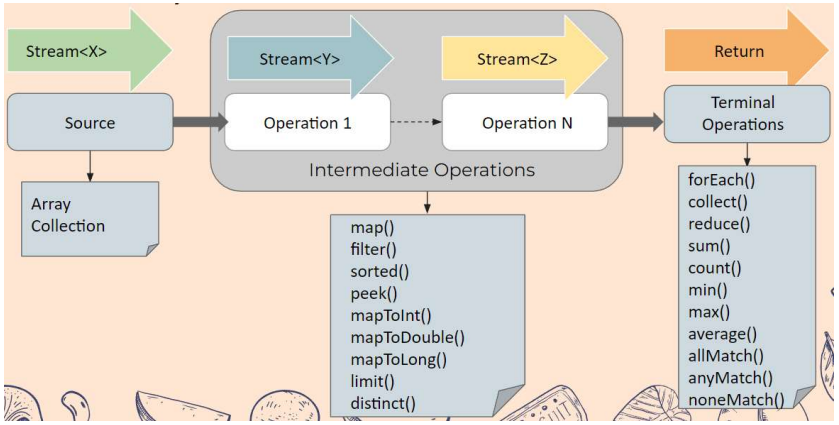
Đặc điểm:

- 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:

- forEach(): duyệt
- filter(): lọc
- map(), mapToDouble(), mapToInt(), mapToLong(): chuyển đổi
- reduce(): tích lũy
- sum(), count(), min(), max(), average(): tổng hợp
- allMatch(), anyMatch(), noneMatch(): kiểm tra

Thao tác xử lý:



Trong Java 8, Collection interface được hỗ trợ 2 phương thức để tạo ra Stream bao gồm:

- stream(): trả về một stream sẽ được xử lý theo tuần tự

- parallelStream(): trả về một Stream song song, các xử lý sau đó sẽ thực hiện song song → tăng tốc độ xử lý

Các method trong Stream:

- Terminal Operations
  - Collect method
  - forEach method
  - Reduce method
  - Max, Min method
- Intermediate Operations
  - Distinct method
  - Map method
  - Filter method
  - Sorted method
  - Limit method
  - Skip method

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(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

Min, Max 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).

.filter()

- 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)

.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

Distinct

- Trả về một stream với các phần tử không trùng lặp

Sorted

- Giúp sắp xếp các phần tử theo một thứ tự xác định

Limit

- 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

- 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

15. Đóng gói và triển khai Website trên Tomcat Server

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® là một phần mềm mã nguồn mở thực hiện cài đặt công nghệ Java Servlet, JavaServer Pages, Java Expression Languate 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 phí.

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 điểm của apache tomcat

- Miễn phí
- Dễ dàng cài đặt và triển khai
- Linh hoạt
- Tiết kiệm tài nguyên
- Được cộng đồng hỗ trợ

Nhược điểm của apache tomcat

- 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

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 TomcatManager 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
- 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
- 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 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