Mục lục

Giới thiệu 3

Quick start 9

Bootstrapping 12

Tomcat Deployment 21

Build System 32

Code Structure 33

Runner 35

Logging 43

Exception Handling 48

Interceptor 55

Servlet Filter 63

Tomcat Port Number 68

REST template 69

File Handling 80

Service Component 86

Thymeleaf 95

CORS Support 101

Internationalization 104

Scheduling 112

Enable HTTPS 116

Eureka Server 118

Service Registration with Eureka 122

Zuul Proxy Server and Routing 126

Spring Boot

Giới thiệu

Spring Boot là Java-based framework mã nguồn mở dùng để tạo micro Service. Được phát triển bởi Pivotal team, được xây dựng để hoạt động độc lập và sản phẩm được tạo ra được tính là một ứng dụng Spring. Phần này giới thiệu cho bạn sơ quát về Spring Boot và các khái niệm cơ bản của nó.

Micro Service là gì?

Micro service là một kỹ thuật phát triển phần mềm cho phép lập trình viên phát triển và tạo các service một cách độc lập. Mỗi running service có một tiến trình hoạt động độc lập, được tính là dạng mô hình nhẹ có thể hỗ trợ trong các ứng dụng kinh doanh thực tế.

Lợi ích của Micro service:

Dễ triển khai.

Có tính mở rộng.

Cấu hình tối thiểu.

Giảm thiểu giời gian sản xuất.

Spring Boot là gì?

Spring Boot là một platform tốt cho những lập trình viên Java sử dụng để phát triển độc lập một sản phẩm Spring application (có thể chạy thử). Bạn có thể bắt đầu với các cấu hình (configurations) tối thiểu nhất khi dùng Spring Boot mà không cần cài đặt bất cứ cấu hình nào khác của Spring.

Lợi ích:

Dễ để hiểu và phát triển Spring application.

Tăng năng suất.

Giảm thời gian phát triển.

Điểm nổi bật

Spring boot được xây dựng nhắm tới một số lợi ích:

Tránh việc cài đặt các cấu hình phức tập trong XML của Spring.

Tạo các sản phẩm Spring application một cách đơn giản hơn.

Giảm thiểu thời gian phát triển cũng như có thể chạy các ứng dụng một cách độc lập.

Giúp việc tiếp cận với ứng dụng đơn giản hơn.

Vì sao nên chọn Spring boot?

Spring boot cung cấp cho ta các lợi ích:

Đơn giản hóa quá trình cấu hình Java Beans, các cấu hình XML và Database Transactions.

Cung cấp trình đa xử lý mạnh mẽ và có thể quản lý REST endpoints.

Trong Spring boot mọi thứ đều được cấu hình tự động, không cần phải cấu hình thủ công nữa.

Cung cấp các @anotation cơ bản của Spring.

Dễ quản lý dependency.

Có bao hàm Embedded Servlet Container.

Cách hoạt động?

Spring Boot tự động cấu hình cho ứng dụng của bạn dựa trên các dependency mà bạn đã thêm vào trong project bằng cách sử dụng @EnableAutoConfiguration anotation. Ví dụ bạn thêm đường dẫn MySQL nhưng không cấu hình kết nối đến cơ sở dữ liệu, Spring boot sẽ tự động cấu hình cho bạn.

Có thể truy cập vào ứng dụng spring boot thông qua lớp có chứa @SpringBootApplication annotation hoặc ở phương thức main.

Spring Boot tự động quét tất cả các component trong project khi sử dụng @ComponentScan annotation.

Spring boot Starters

Quản lý dependency là một phần khó đối với các project lớn nhưng Spring boot đã cung cấp các dụng cụ hỗ trợ quản lý dependency cần thiết để giải quyết.

Ví dụ, nếu bạn muốn dùng Spring và JPA để kết nối cơ sở dữ liệu, bạn đơn giản chỉ cần thêm spring-boot-starter-data-jpa dependency trong project của bạn.

Luôn nhớ rằng mọi Spring boot starter đều có chung dạng là: spring-boot-starter-\* , với \*- là tên đại diện của ứng dụng bạn muốn sử dụng tương ứng.

Ví dụ

Giờ thì cùng xem một số các Spring boot Starter gồm chức năng và code cài đặt của chúng:

**Spring boot Starter Actuator dependency** giám sát và quản lý ứng dụng của bạn. Đây là phần code của nó-

<dependency>

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

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

</dependency>

**Spring boot Starter Security dependency** được dùng tương ứng cho Spring Security. Đây là phần code của nó-

<dependency>

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

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

</dependency>

**Spring boot Starter web dependency** dùng để viết một Rest Enpoints. Đây là phần code của nó-

<dependency>

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

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

</dependency>

**Spring boot Starter Thyme Leaf dependency** dùng để tạo ứng dụng web. Đây là phần code của nó-

<dependency>

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

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

</dependency>

**Spring boot Starter Test dependency** dùng để viết các Test case. Đây là phần code của nó-

<dependency>

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

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

</dependency>

Auto Configuration

Spring boot Auto Configuration tự động cấu hình ứng dụng Spring của bạn dựa trên các JAR dependency mà bạn đã thêm vào trong project. Ví dụ bạn thêm đường dẫn MySQL nhưng không cấu hình kết nối đến cơ sở dữ liệu, Spring boot sẽ tự động cấu hình cho bạn.

Để sử dụng chức năng này bạn cần thêm @EnableAutoConfiguration annotation hoặc @SpringBootApplication annotation vào file class main của bạn. Sau đó ứng dụng Spring boot sẽ tự động cấu hình tương ứng cho bạn.

Xem đoạn code mẫu sau sẽ giúp bạn dễ hình dung hơn-

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

Spring boot Appication

Điểm truy nhập vào Spring boot Application chính là các class có chứa @SpringBootApplicatin annotation (bạn nên để phương thức main ở trong class này). @SpringBootApplication annotation bao gồm cả Auto-Configuration, Component Scan và Spring Boot Configuration.

Có nghĩa là nếu bạn thêm @SpringBootApplication annotation vào 1 class thì bạn không cần thêm @EnableAutoConfiguration, @ComponentScan hay @SpringBootConfiguration annotation vào nữa vì @SpringBootApplication annotation đã bao hàm tất cả các annotation đó rồi.

Xem đoạn code sau để hiểu rõ hơn-

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);

}

}

Component Scan

Spring boot applicatio quét toàn bộ các bean và các gói khai báo khi ứng dụng của bạn được khởi tạo. Bạn cần thêm @Component annotation vào class file để quét các component mà bạn đã thêm vào project.

Xem đoạn code sau để hiểu rõ hơn-

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

@ComponentScan

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

Quick Start

Phần này hướng dẫn cho bạn cách tạo một ứng dụng Spring sử dụng Maven và Gradle.

Prerequisites

Bạn cần cài đặt tối thiểu các công cụ sau để tạo một ứng dụng Spring Boot-

Java 7

Maven 3.2

Gradle 2.5

Spring Boot CLI

Spring Boot CLI là một công cụ command line và nó cho phép chúng ta chạy các Groovy script.

Sử dụng Spring Boot Command Line Interface là cách đơn giản nhất để tạo một ứng dụng Spring Boot. Bạn có thể tạo , chạy và test ứng dụng trong command prompt.

Phần này hướng dẫn từng bước để cài đặt thủ công Spring Boot CLI.

Chi tiết hơn bạn có thể truy cập link:

[https://docs.spring.io/springboot/ docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-springboot](https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-spring-boot)

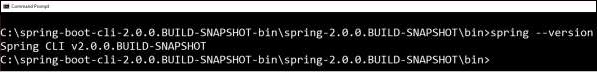
Bạn cũng có thể tải Spring CLI distribution từ Spring Software repository tại: <https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-manual-cli-installation>

Để cài đặt thủ công bạn cần 2 sử dụng 2 file sau-

* spring-boot-cli-2.0.0.BUILD-SNAPSHOT-bin.zip
* spring-boot-cli-2.0.0.BUILD-SNAPSHOT-bin.tar.gz

sau khi tải, lấy file unzip, làm theo các bước hướng dẫn được mô tả trong file install.txt (việc này không có bất cứ yêu cầu gì về cài đặt môi trường phát triển).

Trong Windows, truy cập trực tiếp vào Spring Boot CLI bin trong command prompt và sử dụng lệnh spring –version để đảm bảo CLI đã được cài đặt. Sau khi thực thi lệnh đó bạn có thể thấy được phiên bản của Spring client được thể hiện giống như sau-



### Tạo Hello World với Groovy

Tạo một file groovy đơn giản yêu cầu Rest Endpoint script và chạy file groovy với Spring boot CLI. Xem, thực hiện đoạn code sau để hiểu rõ hơn-

@Controller

class Example {

@RequestMapping("/")

@ResponseBody

public String hello() {

"Hello Spring Boot"

}

}

Lưu file groovy này với tên hello.groovy (lưu trong Spring Boot CLI bin). Sau đó chạy ứng dụng bằng cách sử dụng câu lệnh **spring run hello.groovy** như mẫu dưới đây-

Run Hello World with Groovy

Một khi bạn chạy groovy file, các dependency cần thiết sẽ được tải xuống và sẽ khởi động ứng dụng ở Tomcat cổng 8080 như hình dưới đây-

Run Groovy File Tomcat Port

Một khi Tomcat bắt đầu chạy, sử dụng trình duyệt web và vào đường dẫn URL <http://localhost:8080/> và bạn có thể thấy kết quả như sau:



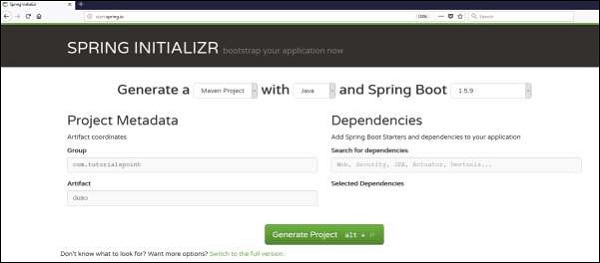
Bootstrapping

Phần này chỉ cho bạn cách sử dụng bootstrap trong ứng dụng Spring boot.

Spring intializer

Một trong các cách để Bootstrapping một ứng dụng Spring Boot là sử dụng Spring Initializer. Để làm được vậy, bạn sẽ truy cập vào trang web Spring Initializer [www.start.spring.io](https://start.spring.io/) và chọn Build bạn muốn, phiên bản Spring Boot và platform. Bạn còn phải cung cấp Group, Artifact và các dependency cần thiết để chạy ứng dụng.

Làm thử theo ví dụ dưới đây về cách thêm spring-boot-starter-web dependency để viết các REST point.



Sau khi bạn đã chọn Group, Artifact, Dependency, Build Project, Platform và phiên bản, click chọn nút Generate Project. File zip sẽ được tải và các file sẽ được giải nén.

Phần này chỉ cho bạn ví dụ bằng cách sử dụng cả Maven và Gradle.

Maven

Sau khi bạn tải project, giải nén file. File pom.xml của bạn sẽ nhiền như sau-

<?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/> <!-- lookup parent from repository -->

</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

Sau khi bạn tải project, giải nén file. File build.gradle file sẽ nhìn như thế này-

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')

}

Class Path Dependencies

Spring Boot cung cấp 1 số lượng lớn các Starter để thêm vào các jars trong class path. Ví dụ, để viết một Rest Endpoint, chúng ta cần thêm các spring-boot-starter-web dependency trong class path. Xem code dưới đây để hiểu rõ hơn-

### Maven dependency

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

### Gradle dependency

dependencies {

compile('org.springframework.boot:spring-boot-starter-web')

}

Main method

Phương thức chính nên là lớp Spring Boot Application. Lớp này nên được thêm @SpringBootApplication annotation. Đó sẽ là điểm truy cập, khởi chạy ứng dụng Spring Boot của bạn. Bạn có thể tìm file main class theo đường dẫn scr/java/main trong package mặc định.

Trong ví dụ này, file main class được cất trong đường dẫn scr/java/main trong package mặc định com.tutorialspoint.demo. Xem code dưới đây để hiểu thêm-

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);

}

}

Viết một Rest Endpoint

Để viết 1 ví dụ Hello World Endpoint trong chính Spring Boot Application main class file, theo các bước dưới đây-

* Đầu tiên, thêm @RestController annotation ở đầu các class
* Tiếp theo, viết 1 phương thức Request URI bằng @RequestMapping annotation.
* Sau đó, phương thức Request URI sẽ trả về chuỗi Hello World.

Code lớp main Spring Boot Application file của bạn sẽ trông thế này-

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String hello() {

return "Hello World";

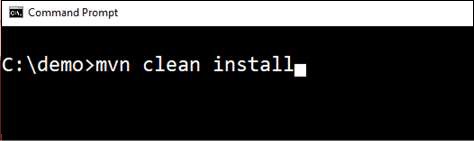
}

}

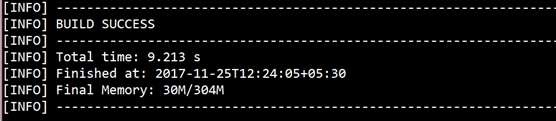
Tạo 1 Executable JAR

Cách tạo một file thực thi JAR để chạy ứng dụng Spring Boot sử dụng câu lệnh Maven và Gradle nhu sau:

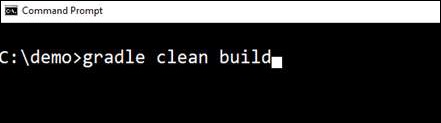
Dùng câu lệnh mvn clean install của Maven-



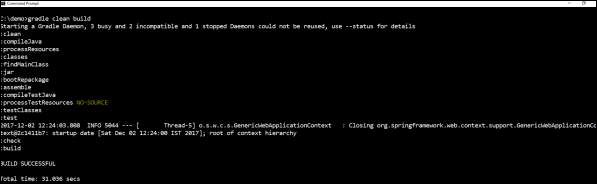
Sau khi chạy câu lệnh, bạn có thể thấy thông báo BUILD SUCCESS ở command prompt như vầy-



Dùng câu lệnh gradle clean build của Gradle-



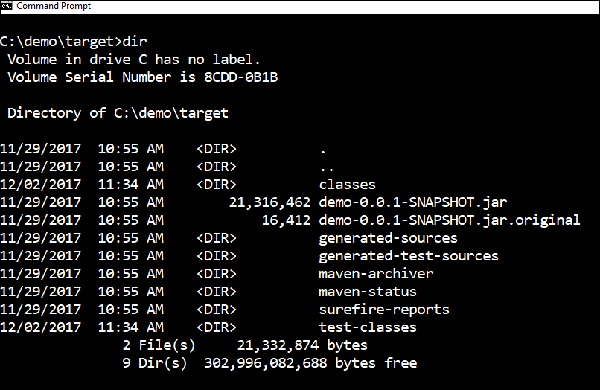
Khi chạy câu lệnh đó bạn sẽ thấy thông báo BUILD SUCCESS như vầy-



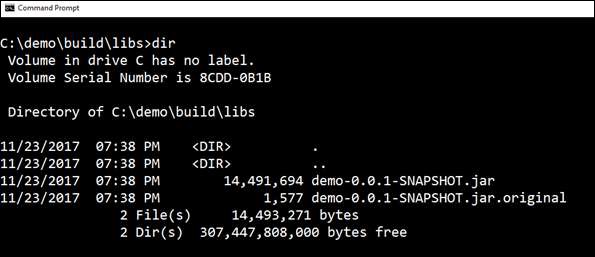
Chạy Hello World với Java

Sau khi bạn đã tạo một file thực thi JAR, bạn có thể tìm nó theo đường dẫn tương ứng.

Đối với Maven, bạn có thể tìm file JAR đó thông qua đường dẫn target như sau-



Còn đối với Gradle, bạn có thể tìm JAR file thông qua đường dẫn build/libs như vầy-



Tiếp theo, chạy file JAR bằng cách sử dụng command java –jar<JARFILE>. Theo ví dụ trên, tên file JAR được đặt là demo-0.0.1-SNAPSHOT.jar nên tương ứng ta có câu lệnh-



sau khi bạn chạy file JAR, bạn có thể thấy màn hình console hiển thị như vầy-

Output in Console Window

Theo như màn hình console hiển thị, Tomcat đã được khởi động ở cổng 8080 (http). Giờ bạn có thể sử dụng trình duyệt web và truy cập qua đường dẫn <http://8080/> và xem kết quả hiển thị sẽ như vầy-



Tomcat Deployment

­­Bằng cách sử dụng ứng dụng Spring Boot, chúng ta có thể tạo và triển khai 1 file WAR vào trong web server. Phần này sẽ hướng dẫn các bạn cách tạo 1 file WAR và triển khai ứng dụng Spring Boot trên Tomcat web server.

Spring Boot Servlet Initializer

Cách khởi tạo thông thường nhất là tạo một ứng dụng Spring Boot với 1 class chứa @SpringBootApplication extend (kế thừa) lớp SpringBootServletInitializer. Spring Boot Servlet Initializer cho phép bạn cấu hình ứng dụng khi nó được khởi chạy thông qua việc sử dụng Servlet Container.

Code Spring Boot Application class file để tạo file JAR như sau-

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);

}

}

Ta cần extend (kế thừa) SpringBootServletInitializer để nó hỗ trợ cho việc phát triển WAR file. Code Spring Boot Application class file sẽ nhìn như vầy-

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication

public class DemoApplication extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(DemoApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

Setting Main Class

Trong Spring Boot, chúng ta gọi hàm main muốn chạy vào trong build file. Với mục tiêu đó bạn có thể thực hiện bằng cách sau-

Đối với Maven, thêm lớp khởi tạo trong properties của pom.xml như vầy-

<start-class>com.tutorialspoint.demo.DemoApplication</start-class>

Đối với Gradle, thêm tên lớp main vào trong build.gradle như vầy-

mainClassName="com.tutorialspoint.demo.DemoApplication"

Update packaging JAR vào WAR

Chúng ta phải cập nhật packaging JAR vào WAR sử dụng các đoạn code như vầy:

Đối với Maven, thêm packaging dưới dạng WAR vào pom.xml như sau-

<packaging>war</packaging>

Đối với Gradle, thêm ứng dụng plugin và WAR plugin vào trong build.gradle như vầy-

apply plugin: ‘war’

apply plugin: ‘application’

Tiếp theo, thử viết một Rest Point đơn giản trả về “Hello World from Tomcat”. Trước tiên để viết 1 Rest point, chúng ta cần thêm Spring Boot web starter dependency vào build file của bạn.

Đối với Maven, thêm Spring Boot starter dependency vào file pom.xml như vầy-

<dependency>

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

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

</dependency>

Đối với Gradle, thêm Spring Boot starter dependency vào trong build.gradle bằng cách code như vầy-

dependencies {

compile('org.springframework.boot:spring-boot-starter-web')

}

Giờ hãy thử viết một Rest endpoint đơn giản trong Spring Boot Application class file đầy đủ-

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(DemoApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String hello() {

return "Hello World from Tomcat";

}

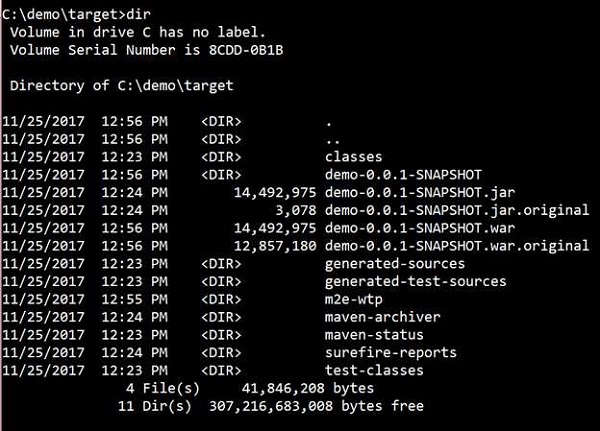
}

Packaging ứng dụng của bạn

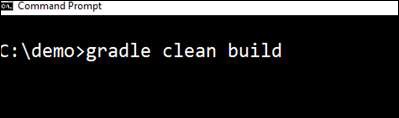
Tạo một file WAR để triển khai trên Tomcat server bằng cách sử dụng các câu lệnh Maven và Gradle để packaging (đóng gói) ứng dụng của bạn như sau:

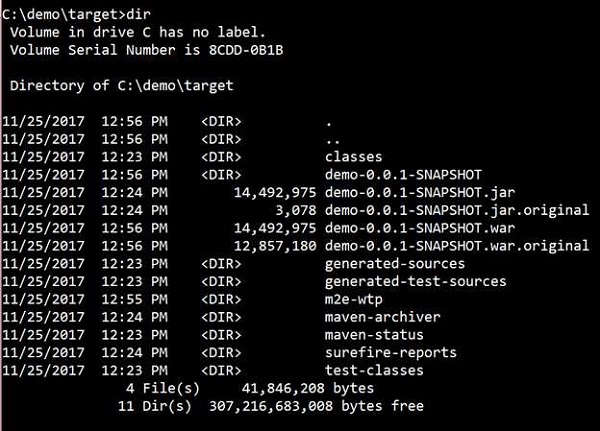
Đối với Maven, dùng câu lệnh mvn package để đóng gói ứng dụng của bạn. Sau đó, file WAR sẽ được tạo và bạn có thể tìm thấy nó trong đường dẫn target như vầy-





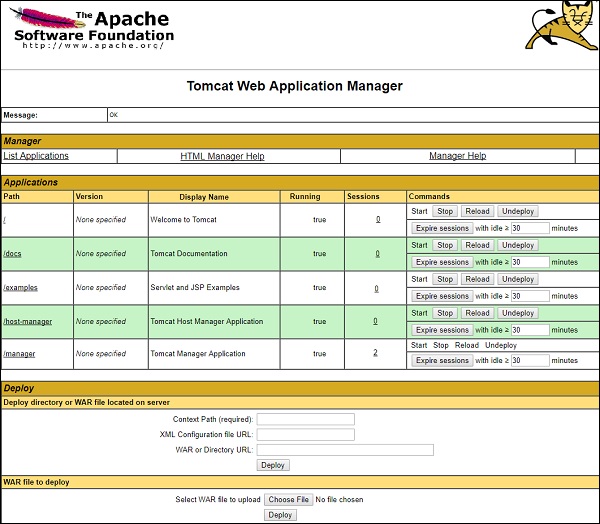
Đối với Gradle, dùng câu lệnh gradle clean build để đóng gói ứng dụng của bạn. Sau đó, file WAR của bạn sẽ được tạo và bạn có thể tìm nó trong đường dẫn build/libs. Làm theo hình dưới đây để hiểu rõ thêm-





Triển khai trong Tomcat

Bây giờ, chạy Tomcat Server và triển khai file WAR trong đường dẫn thư mục webapps theo các bước sau:





Sau khi triển khai thành công, truy cập theo đường dẫn URL trên trình duyệt: <http://localhost:8080/demo-0.0.1-SNAPSHOT/> kết quả thu được sẽ như hình dưới:



Code đầy đủ của cả ví dụ:

File 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>war</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/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<start-class>com.tutorialspoint.demo.DemoApplication</start-class>

</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>

File 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'

apply plugin: 'war'

apply plugin: 'application'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

mainClassName = "com.tutorialspoint.demo.DemoApplication"

repositories {

mavenCentral()

}

dependencies {

compile('org.springframework.boot:spring-boot-starter-web')

testCompile('org.springframework.boot:spring-boot-starter-test')

}

Hàm main của ứng dụng

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication extends SpringBootServletInitializer {

@Override

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(DemoApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String hello() {

return "Hello World from Tomcat";

}

}

Từ đoạn này chỉ dịch Maven không có Gradle

Build System

Việc chọn hệ thống, môi trường để xây dựng ứng dụng của bạn rất quan trọng, Maven và Gradle là 2 lựa chọn hỗ trợ Spring boot tốt nhất.

Quản lý Dependency

Spring boot hỗ trợ các dependency theo phiên bản của Spring Boot cập nhật thường xuyên. Bạn không cần khai báo phiên bản của Spring Boot trong file cấu hình, Spring boot sẽ tự động cài đặt dependency khi có cập nhật mới nên bạn chỉ cần cập nhật phiên bản Spring Boot mới là được.

Maven Dependency

Đối với cấu hình của Maven bạn nên thừa kế Spring Boot Starter parent để có thể sử dụng các dependency của lớp này, bằng cách cấu hình trong file pom.xml như vầy-

<parent>

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

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

<version>1.5.8.RELEASE</version>

</parent>

Với Spring Boot Parent Starter dependency bạn nên khai báo phiên bản của chúng, còn với các Starter dependency khác thì không cần. Ví dụ với Spring Boot Starter web:

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

Cấu trúc code

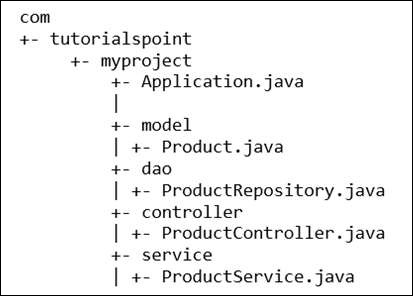
Spring boot không có cấu trúc mẫu để lựa chọn. Ta có thể tự tạo 1 số thói quen như sau:

Package mặc định

Một lớp không chưa package nào thì có thể xem là 1 package mặc định. Không nên khai báo package mặc định vì có thể gây sự cố trong quá trình Auto configuration hay component scan.

Lưu ý: thường thì tên package sẽ là tên miền bạn muốn đặt đảo ngược. vd: com.tutorialspoint.myproject

Bố cục điển hình của 1 ứng dụng Spring Boot:



File Application.java nên được khai báo cùng với hàm main và @SpringBootApplication annotation.

package com.tutorialspoint.myproject;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {SpringApplication.run(Application.class, args);}

}

Bean và Dependency Injection

Trong Spring Boot có thể dùng Spring framework để xác định các bean và DI của chúng. Dùng @ComponentScan để tìm bean và @Autowired để xác định DI tương ứng.

package com.tutorialspoint.demo;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestTemplate;

@SpringBootApplication

public class DemoApplication {

@Autowired

RestTemplate restTemplate;

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

}

Runners

Application runner và Command line runner là các giao diện giúp bạn thực thi lệnh code khi ứng dụng Spring đã được bắt đầu chạy, nhờ đó bạn có thể thực thi hành động mong muốn song song với ứng dụng đang chạy.

Application runner

Cách kế thừa AR trong hàm main để sử dụng:

package com.tutorialspoint.demo;

import org.springframework.boot.ApplicationArguments;

import org.springframework.boot.ApplicationRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication implements ApplicationRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(ApplicationArguments arg0) throws Exception {

System.out.println("Hello World from Application Runner");

}

}

Command line runner

Cách kế thừa CLN trong hàm main để sử dụng:

package com.tutorialspoint.demo;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(String... arg0) throws Exception {

System.out.println("Hello world from Command Line Runner");

}

}

Application properties

SB cho hỗ trợ làm việc trên nhiều môi trường, phần này chỉ cách cấu hình và xác định các properties cho ứng dụng SB của bạn.

## Command Line Properties

Cách đổi số của cổng:

B1: tạo 1 file thực thi JAR, chạy bằng lệnh java –jar<tên file jar>

B2: dùng câu lệnh dưới đây để đổi số của cổng-



## Properties File

Các file Properties thường được dùng để giữ nhiều thuộc tính để chạy ứng dụng trong 1 file duy nhất trên nhiều môi trường khác nhau. Với SB các properties được giữ trong file application.properties với đường dẫn scr/main.resource . code mẫu của 1 file application.properties-

server.port = 9090

spring.application.name = demoservice

## YAML File

SB cũng hỗ trợ YAML dựa trên cấu hình propreties để chạy ứng dụng, ta có thể dùng filee application.yaml thay cho file application.properties. mẫu-

spring:

application:

name: demoservice

server:

port: 9090

## Externalized Properties

Thay vì cất file properties như bình thường ta có thể cất chúng ở các chỗ khác. Ta có thể thay đổi đường dẫn trong quá trình chạy JAR file bằng câu lệnh-

-Dspring.config.location = C:\application.properties

Externalized Properties

## Use of @Value Annotation

@Value annotation dùng để đọc môi trường hay property của ứng dụng trong code Java.

Lệnh là:

@Value("${property\_key\_name}")

Ví dụ lệnh để đọc property của file spring.application.name bằng cách dùng @Value annotation-.

@Value("${spring.application.name}")

Code ví dụ:

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication {

@Value("${spring.application.name}")

private String name;

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String name() {

return name;

}

}

**Lưu ý:** nếu không tìm thấy property khi chạy ứng dụng , SB sẽ coi illegal Argument exception là Could no resolve placeholder ‘spring.application.name in value “${spring.application.name}”.

**Có thể giải quyết việc này bằng cách đặt giá trị mặc định cho property bằng câu lệnh:**

@Value("${property\_key\_name:default\_value}")

@Value("${spring.application.name:demoservice}")

## Spring Boot Active Profile

## Spring Boot hỗ trợ các thuộc tính khác nhau dựa trên hồ sơ hoạt động của Spring. Ví dụ: chúng ta có thể giữ hai tệp riêng biệt để phát triển và sản xuất để chạy ứng dụng Spring Boot.

## Spring active profile trong application.properties

Hãy cùng tìm hiểu làm thế nào để có Spring active profile trong application.properations. Theo mặc định, application. properties sẽ được sử dụng để chạy ứng dụng Spring Boot. Nếu bạn muốn sử dụng profile dựa trên các property, chúng ta có thể giữ tệp properties riêng cho từng profile như được hiển thị bên dưới –

**application.properties**

server.port = 8080

spring.application.name = demoservice

**application-dev.properties**

server.port = 9090

spring.application.name = demoservice

**application-prod.properties**

server.port = 4431

spring.application.name = demoservice

Trong khi chạy tệp JAR, chúng ta cần chỉ định spring active profile dựa trên mỗi tệp thuộc tính (properties). Theo mặc định, ứng dụng Spring Boot sử dụng tệp application.properies. Lệnh để thiết lập hồ sơ hoạt động mùa xuân được hiển thị dưới đây −



Bạn có thể thấy tên active profile trên console log điều khiển như dưới đây −

2017-11-26 08:13:16.322 INFO 14028 --- [

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: dev

Tomcat đã được chạy trên cổng 9090 (http) −

2017-11-26 08:13:20.185 INFO 14028 --- [

main] s.b.c.e.t.TomcatEmbeddedServletContainer :

Tomcat started on port(s): 9090 (http)

Bạn có thể đặt Production active profile như vầy −



Bạn có thể thấy tên active profile trên console log như vầy −

2017-11-26 08:13:16.322 INFO 14028 --- [

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: prod

Tomcat chạy trên cổng 4431 (http) sẽ như vầy −

2017-11-26 08:13:20.185 INFO 14028 --- [

main] s.b.c.e.t.TomcatEmbeddedServletContainer :

Tomcat started on port(s): 4431 (http)

### Spring active profile cho application.yml

Hãy cùng tìm hiểu làm thế nào để giữ Spring active profile cho application.yml. Chúng ta có thể giữ các thuộc tính Spring active profile trong tệp application.yml. Không cần sử dụng tệp riêng biệt như application.properies. Sau đây là một đoạn code mẫu để giữ các Spring active profiles trong tệp application.yml.

Lưu ý rằng dấu phân cách (---) được sử dụng để phân tách từng cấu hình trong tệp application.yml.

spring:

application:

name: demoservice

server:

port: 8080

---

spring:

profiles: dev

application:

name: demoservice

server:

port: 9090

---

spring:

profiles: prod

application:

name: demoservice

server:

port: 4431

Để ra lệnh thiết lập development active profile được đưa ra dưới đây −



Bạn có thể thấy tên active profile trên console log như vầy −

2017-11-26 08:41:37.202 INFO 14104 --- [

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: dev

Tomcat đã được chạy trên cổng 9090 (http) −

2017-11-26 08:41:46.650 INFO 14104 --- [

main] s.b.c.e.t.TomcatEmbeddedServletContainer :

Tomcat started on port(s): 9090 (http)

Lệnh để đặt Production active profile −



Bạn có thể thấy tên active profile trên console log như vầy −

2017-11-26 08:43:10.743 INFO 13400 --- [

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: prod

Điều này sẽ khởi chạy Tomcat trên cổng 4431 (http) như vầy:

2017-11-26 08:43:14.473 INFO 13400 --- [

main] s.b.c.e.t.TomcatEmbeddedServletContainer :

Tomcat started on port(s): 4431 (http)

Logging

Spring Boot sử dụng Apache Commons logging cho tất cả các bản ghi nội bộ. Các cấu hình mặc định của Spring Boot hỗ trợ cho việc sử dụng Java Util Logging, Log4j2 và Logback. Sử dụng chúng, chúng ta có thể cấu hình console logging và file logging.

Nếu bạn đang sử dụng Spring Boot Starters, Logback sẽ hỗ trợ tốt cho việc logging. Bên cạnh đó, Logback cũng hỗ trợ tốt cho Common logging , Util logging, Log4J và SLF4J.

Log format

Format mặc định của log trong SB:



Cung cấp cho ta các thông tin:

Ngày giờ

Log level: INFO ERROR or WARN

Process ID

Thread name: trong ngoặc vuông

Logger name biểu thị tên class nguồn

The log message

## Console Log Output

## Các log mặc định sẽ hiển thị trên màn hình console. Theo mặc định, INFO ERROR WARN là các log sẽ được lưu vào log file.

## Nếu dùng debug level log, hãy thêm debug flag lúc bắt đầu chạy ứng dụng bằng lệnh:

java –jar demo.jar --debug

thêm chế độ debug bằng lệnh:

debug = true

## File Log Output

## Mặc định các logging chỉ hiện trên cửa sổ console không lưu trong file, nếu muốn lưu trong file, cần cài đặt property trong file logging.file hoặc logging.path trong file application.properties .

## Chỉ định địa chỉ file log đó bằng lệnh mẫu như sau-

logging.path = /var/tmp/

## chú ý tên file sẽ được lưu là spring.log.

## có thể tự đặt tên nếu muốn bằng lệnh:

logging.file = /var/tmp/mylog.log

## tất cả các file log sẽ tự động rotate khi đạt kích cỡ 10MB

Log levels

SB hỗ trợ tất cả các logger level như:

 “TRACE”, “DEBUG”, “INFO”, “WARN”, “ERROR”, “FATAL”, “OFF”

Có thể tự đặt Root logger trong application.properties như ví dụ này:

logging.level.root = WARN

lưu ý logback không hỗ trợ FATAL level log, nó được tính như ERROR level log.

Cấu hình LogBack

Có thể cấu hình LogBack thủ công bằng XML. Những cấu hình này sẽ được lưu trong logback.xml.

Ví dụ cấu hình root level trong xml-

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

<configuration>

<root level = "INFO">

</root>

</configuration>

Cấu hình cả console appender-

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

<configuration>

<appender name = "STDOUT" class = "ch.qos.logback.core.ConsoleAppender"></appender>

<root level = "INFO">

<appender-ref ref = "STDOUT"/>

</root>

</configuration>

Và cấu hình cả file appender-

Lưu ý: phải đặt log file trong file appender.

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

<configuration>

<appender name = "FILE" class = "ch.qos.logback.core.FileAppender">

<File>/var/tmp/mylog.log</File>

</appender>

<root level = "INFO">

<appender-ref ref = "FILE"/>

</root>

</configuration>

Cách định nghĩa Log pattern:

Lệnh-

<pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>

Code vd-

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

<configuration>

<appender name = "STDOUT" class = "ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>

</encoder>

</appender>

<appender name = "FILE" class = "ch.qos.logback.core.FileAppender">

<File>/var/tmp/mylog.log</File>

<encoder>

<pattern>[%d{yyyy-MM-dd'T'HH:mm:ss.sss'Z'}] [%C] [%t] [%L] [%-5p] %m%n</pattern>

</encoder>

</appender>

<root level = "INFO">

<appender-ref ref = "FILE"/>

<appender-ref ref = "STDOUT"/>

</root>

</configuration>

Cách thêm slf4j Logger vào file main class SB:

package com.tutorialspoint.demo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

private static final Logger logger = LoggerFactory.getLogger(DemoApplication.class);

public static void main(String[] args) {

logger.info("this is a info message");

logger.warn("this is a warn message");

logger.error("this is a error message");

SpringApplication.run(DemoApplication.class, args);

}

}

Kết quả chạy của màn hình console-

Logger Console Window

Kết quả ghi lại trong log file-

Log Output

Exception Handling

Can thiệp exception và errors trong APIs rồi phản hồi lại về client là việc cần cho các ứng dụng enterprise. Chương này hướng dẫn cách can thiệp vào các exception.

## Controller Advice

@ControllerAdvice là annotation can thiệp exception 1 cách toàn cục (globally).

## Exception Handler

@ExceptionHandler là annotation dùng để định rõ các exception rồi gửi phản hồi tự chọn đến client.

Vd-

package com.tutorialspoint.demo.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;

@ControllerAdvice

public class ProductExceptionController {

}

Chỉ định 1class kế thừa RuntimeException-

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Có thể chỉ định phương thức @ExceptionHandler để can thiệp vào exception như sau-.phương thức này nên được dùng để viết Controller Advice class file.

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

}

Sau đó, dùng đoạn code này để ném exception vào từ API-

@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)

public ResponseEntity<Object> updateProduct() {

throw new ProductNotfoundException();

}

Code đầy đủ-

Trong ví dụ này, ta dùng PUT API để cập nhật product. Trong quá trình cập nhật nếu không tìm thấy product thì trả về thông báo phản hồi: “Product not found”.

Chú ý: ProductNotFoundException exception class nên thừa kế RuntimeException.

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Code Controller Advice class để can thiệp vào exception bậc toàn cục (globally)-. Có thể định nghĩa bất cứ phương thức Exception Handler nào trong class file này.

package com.tutorialspoint.demo.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice

public class ProductExceptionController {

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

return new ResponseEntity<>("Product not found", HttpStatus.NOT\_FOUND);

}

}

Product Service API controller file dưới đây dùng để cập nhật Product. Nếu không tìm thấy Product thì nó vứt vào ProductNotFoundException class.

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/{id}", method = RequestMethod.PUT)

public ResponseEntity<Object> updateProduct(@PathVariable("id") String id, @RequestBody Product product) {

if(!productRepo.containsKey(id))throw new ProductNotfoundException();

productRepo.remove(id);

product.setId(id);

productRepo.put(id, product);

return new ResponseEntity<>("Product is updated successfully", HttpStatus.OK);

}

}

Code hàm main của ứng dụng SB vd:

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);

}

}

Code cho POJO class của 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;

}

}

Code của file pom.xml (Maven)-

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

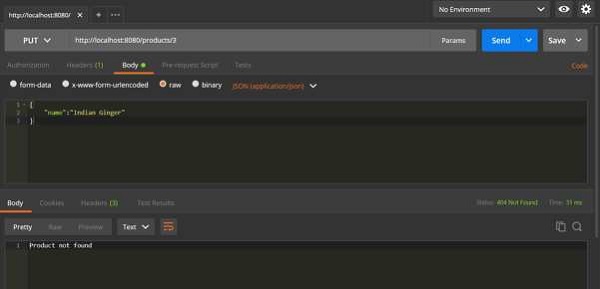
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

Exception Handling Tomcat Application Startded

Truy cập đường dẫn URL dưới đây trong ứng dụng POSTMAN và bạn sẽ thấy kết quả như vầy-

Update URL: http://localhost:8080/products/3



Interceptor

Có thể dùng Interceptor trong SB để thao tác trong các hoàn cảnh:

* Trước khi gửi yêu cầu đến controller
* Trước khi gửi phản hồi cho client

Để dùng 1 interceptor, bạn cần tạo @Component class để hỗ trợ nó và cần thừa kế HandlerInterceptor interface.

3 phương thức bạn cần nhớ khi sử dụng Interceptor:

1. preHandle() – dùng để thực hiện các thao tác trước khi gửi phản hồi đến controller. Phương thức này nên trả về true để phản hồi lại client
2. postHandle() – dùng để thực hiện các thao tác trước khi gửi phản hồi về client.
3. afterCompetition() – dùng để thực hiện các thao tác sau khi hoàn thành việc yêu cầu và phản hồi.

code vd-

@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 {}

}

Phải khai báo Interceptor với InterceptorRegistry bằng cách sử dụng WebMvcConfigureAdapter như vầy-

@Component

public class ProductServiceInterceptorAppConfig extends WebMvcConfigurerAdapter {

@Autowired

ProductServiceInterceptor productServiceInterceptor;

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(productServiceInterceptor);

}

}

Thử dùng GET products API, code dưới dây là Interceptor 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");

}

}

Code cho Application class file để khai áo Interceptor vào trong Interceptor Registry – 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);

}

}

Code cho Controller class file 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);

}

}

Code của POJO class cho Product.java:

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;

}

}

Code hàm main của ứng dụng SB class file DemoApplication.java-

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);

}

}

Code file pom.xml (Maven) -

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh -

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

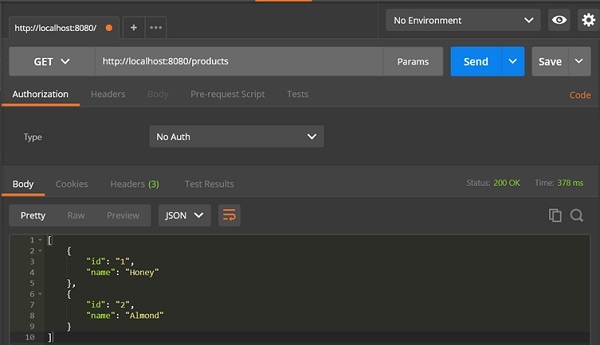
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

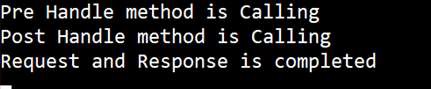
Started Application on Tomcat Port 8080

Truy cập đường dẫn URL dưới đây trong ứng dụng POSTMAN và bạn sẽ thấy kết quả như vầy-

GET API: **http://localhost:8080/products**



Kết quả hiển thị trên cửa sổ console, bạn có thể thấy lệnh in System.out.println đã được thê vào trong Interceptor và hiển thị ra như hình-



Servlet Filter

1 Filter là 1 đối tượng dùng để chặn yêu cầu và phản hồi từ HTTP trong ứng dụng của bạn. Bằng cách sử dụng Filter ta có thể thực hiện 2 thao tác tại 2 trường hợp-

* Trước khi gửi các yêu cầu đến controller
* Trước khi gửi 1 phản hòi đến client

Code ví dụ về 1 class thừa kế Servlet Filter có @Component annotation.

@Component

public class SimpleFilter implements Filter {

@Override

public void destroy() {}

@Override

public void doFilter

(ServletRequest request, ServletResponse response, FilterChain filterchain)

throws IOException, ServletException {}

@Override

public void init(FilterConfig filterconfig) throws ServletException {}

}

Ví dụ sau là code cho việc đọc remote host và remote address từ đối tượng ServletRequest trước khi gửi yêu cầu đến controller.

Trong phương thức doFilter(), ta đã thêm lệnh in System.out.println để in ra remote host và remote address.

package com.tutorialspoint.demo;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import org.springframework.stereotype.Component;

@Component

public class SimpleFilter implements Filter {

@Override

public void destroy() {}

@Override

public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterchain)

throws IOException, ServletException {

System.out.println("Remote Host:"+request.getRemoteHost());

System.out.println("Remote Address:"+request.getRemoteAddr());

filterchain.doFilter(request, response);

}

@Override

public void init(FilterConfig filterconfig) throws ServletException {}

}

Trong hàm main, chúng ta đã thêm REST point đơn giản trả về chuỗi “Hello World”.

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@RequestMapping(value = "/")

public String hello() {

return "Hello World";

}

}

Code cho file pom.xml (Maven) -

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

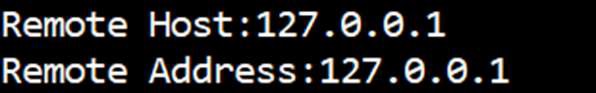
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

truy cập đường dẫn URL <http://localhost:8080/> sẽ thấy dòng hiển thị Hello World như vầy



Còn trong console log sẽ hiển thị Remote host và Remote address như vầy-



Tomcat Port Number

Bạn có thể chạy ứng dụng của mình trên nhiều cổng (port) khác nhau với SB. Nhớ răng cổng mặc định là 8080.

## Custom Port

Trong file application.properties, chúng ta có thể đặt số của cổng tùy ý cho đặc tính server.port

server.port = 9090

trong file application.yml có cách tương tự:

server:

port: 9090

## Random Port

Trong file **application.properties** , chúng ta có thể đặt số của cổng là ngẫu nhiên bằng lệnh:

server.port = 0

trong file application.yml có cách tương tự:

server:

port: 0

**Note** − If the **server.port** number is 0 while starting the Spring Boot application, Tomcat uses the random port number.

REST TEMPLATE

Rest template được dùng để tạo các ứng dụng sử dụng RESTful Web Service. Bạn có thể dùng phương thức exchange() để sử dụng các web service cho tất cả các phương thức HTTP.

Code ví dụ tạo Bean cho Rest Template để auto wiring Rest Template object-

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestTemplate;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

}

GET

**Sử dụng GET API bằng cách dùng RestTemplate – phương thức exchange()**

Giả sử URL [**http://localhost:8080/products**](http://localhost:8080/products) trả về JSON tương ứng, chúng ta sẽ dùng phản hồi API đó bằng cách sử dụng Rest Template như code sau-

[

{

"id": "1",

"name": "Honey"

},

{

"id": "2",

"name": "Almond"

}

]

Các bước:

* Autowired the Rest Template Object.
* Dùng HttpHeaders để đặt Request Headers.
* Dùng HttpEntity gói các đối tượng request.
* Cung cấp kiểu URL, HttpMethod, and Return cho phương thức Exchange() .

@RestController

public class ConsumeWebService {

@Autowired

RestTemplate restTemplate;

@RequestMapping(value = "/template/products")

public String getProductList() {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity <String> entity = new HttpEntity<String>(headers);

return restTemplate.exchange("

http://localhost:8080/products", HttpMethod.GET, entity, String.class).getBody();

}

}

POST

**Sử dụng POST API bằng cách dùng RestTemplate – phương thức exchange()**

Giả sử URL [**http://localhost:8080/products**](http://localhost:8080/products) trả về phản hôi như hình dưới, chúng ta sẽ dùng phản hồi API đó bằng cách sử dụng Rest Template. Code cho phần thân Request

{

"id":"3",

"name":"Ginger"

}

Code của phần thân Response

Product is created successfully

Các bước để có thể sử dụng API

* Autowired the Rest Template Object.
* Dùng HttpHeaders để đặt Request Headers.
* Dùng HttpEntity gói các đối tượng request. ở đây, chúng ta gói Product object lại để gửi nó đến phần thân Request.
* Cung cấp kiểu URL, HttpMethod, and Return cho phương thức Exchange() .

@RestController

public class ConsumeWebService {

@Autowired

RestTemplate restTemplate;

@RequestMapping(value = "/template/products", method = RequestMethod.POST)

public String createProducts(@RequestBody Product product) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);

return restTemplate.exchange(

"http://localhost:8080/products", HttpMethod.POST, entity, String.class).getBody();

}

}

PUT

**Sử dụng PUT API bằng cách dùng RestTemplate – phương thức exchange()**

Giả sử URL [**http://localhost:8080/products**](http://localhost:8080/products)**/3** trả về phản hôi như hình dưới, chúng ta sẽ dùng phản hồi API đó bằng cách sử dụng Rest Template. Code cho phần thân Request

{

"name":"Indian Ginger"

}

Code của phần thân Response

Product is updated successfully

Các bước để có thể sử dụng API

* Autowired the Rest Template Object.
* Dùng HttpHeaders để đặt Request Headers.
* Dùng HttpEntity gói các đối tượng request. ở đây, chúng ta gói Product object lại để gửi nó đến phần thân Request.
* Cung cấp kiểu URL, HttpMethod, and Return cho phương thức Exchange() .

@RestController

public class ConsumeWebService {

@Autowired

RestTemplate restTemplate;

@RequestMapping(value = "/template/products/{id}", method = RequestMethod.PUT)

public String updateProduct(@PathVariable("id") String id, @RequestBody Product product) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);

return restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.PUT, entity, String.class).getBody();

}

}

DELETE

**Sử dụng PUT API bằng cách dùng RestTemplate – phương thức exchange()**

Giả sử URL [**http://localhost:8080/products**](http://localhost:8080/products)**/3** trả về phản hôi như hình dưới, chúng ta sẽ dùng phản hồi API đó bằng cách sử dụng Rest Template. Code cho phần thân Request

Product is deleted successfully

Các bước để có thể sử dụng API

* Autowired the Rest Template Object.
* Dùng HttpHeaders để đặt Request Headers.
* Dùng HttpEntity gói các đối tượng requestCung cấp kiểu URL, HttpMethod, and Return cho phương thức Exchange() .

@RestController

public class ConsumeWebService {

@Autowired

RestTemplate restTemplate;

@RequestMapping(value = "/template/products/{id}", method = RequestMethod.DELETE)

public String deleteProduct(@PathVariable("id") String id) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(headers);

return restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.DELETE, entity, String.class).getBody();

}

}

Code hoàn chỉnh của file Rest Template Controller class

package com.tutorialspoint.demo.controller;

import java.util.Arrays;

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

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.MediaType;

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 org.springframework.web.client.RestTemplate;

import com.tutorialspoint.demo.model.Product;

@RestController

public class ConsumeWebService {

@Autowired

RestTemplate restTemplate;

@RequestMapping(value = "/template/products")

public String getProductList() {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<String> entity = new HttpEntity<String>(headers);

return restTemplate.exchange(

"http://localhost:8080/products", HttpMethod.GET, entity, String.class).getBody();

}

@RequestMapping(value = "/template/products", method = RequestMethod.POST)

public String createProducts(@RequestBody Product product) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);

return restTemplate.exchange(

"http://localhost:8080/products", HttpMethod.POST, entity, String.class).getBody();

}

@RequestMapping(value = "/template/products/{id}", method = RequestMethod.PUT)

public String updateProduct(@PathVariable("id") String id, @RequestBody Product product) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(product,headers);

return restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.PUT, entity, String.class).getBody();

}

@RequestMapping(value = "/template/products/{id}", method = RequestMethod.DELETE)

public String deleteProduct(@PathVariable("id") String id) {

HttpHeaders headers = new HttpHeaders();

headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));

HttpEntity<Product> entity = new HttpEntity<Product>(headers);

return restTemplate.exchange(

"http://localhost:8080/products/"+id, HttpMethod.DELETE, entity, String.class).getBody();

}

}

Code của Spring Boot Application Class – DemoApplication.java −

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);

}

}

Code file pom.xml (Maven) −

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

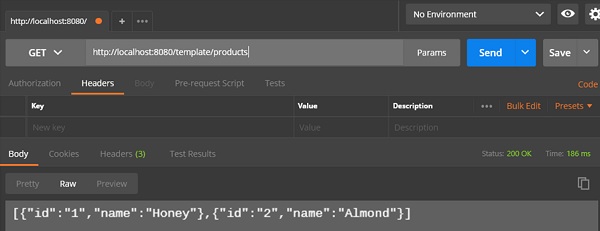
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

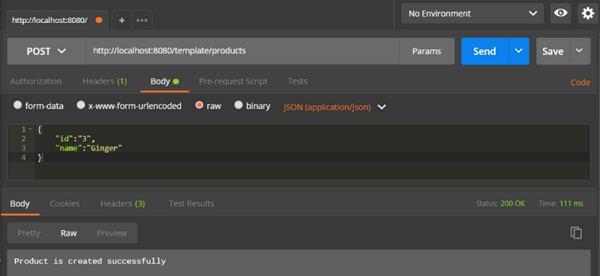
Started Application on Tomcat Port_8080

Truy cập vào đường dẫn URL trong ứng dụng POSTMAN để thấy kết quả.

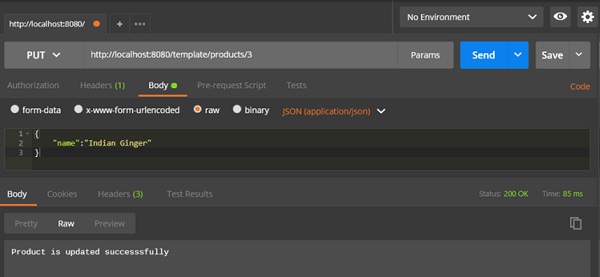
GET Products by Rest Template − **http://localhost:8080/template/products**



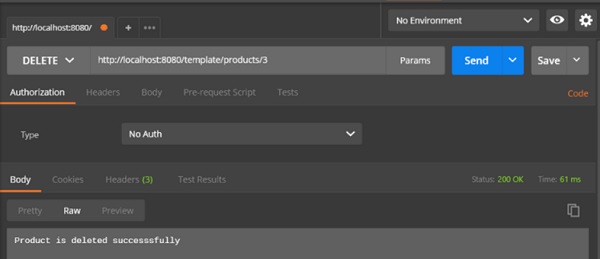
Create Products POST − **http://localhost:8080/template/products**



Update Product PUT − **http://localhost:8080/template/products/3**



Delete Product − **http://localhost:8080/template/products/3**



File Handling

## Trong chương này, bạn sẽ tìm hiểu cách tải lên và tải xuống tệp bằng cách sử dụng dịch vụ web.

## File Upload

Để tải lên một tệp, bạn có thể sử dụng MultipartFile làm Request Parameter và API này sẽ tiêu thụ Multi-Part form data value. Quan sát code được đưa ra dưới đây -

@RequestMapping(value = "/upload", method = RequestMethod.POST, consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public String fileUpload(@RequestParam("file") MultipartFile file) {

return null;

}

Code đầy đủ dưới đây −

package com.tutorialspoint.demo.controller;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import org.springframework.http.MediaType;

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

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.multipart.MultipartFile;

@RestController

public class FileUploadController {

@RequestMapping(value = "/upload", method = RequestMethod.POST,

consumes = MediaType.MULTIPART\_FORM\_DATA\_VALUE)

public String fileUpload(@RequestParam("file") MultipartFile file) throws IOException {

File convertFile = new File("/var/tmp/"+file.getOriginalFilename());

convertFile.createNewFile();

FileOutputStream fout = new FileOutputStream(convertFile);

fout.write(file.getBytes());

fout.close();

return "File is upload successfully";

}

}

## File Download

Bạn nên sử dụng InputStreamResource khi muốn tải xuống 1file. Chúng ta cần thiết lập HttpHeader Content-Disposition và cần chỉ định kiểu phản hồi Media của ứng dụng.

**Lưu ý - Trong ví dụ sau, file nên có sẵn trên đường dẫn được chỉ định nơi ứng dụng đang chạy.**.

@RequestMapping(value = "/download", method = RequestMethod.GET)

public ResponseEntity<Object> downloadFile() throws IOException {

String filename = "/var/tmp/mysql.png";

File file = new File(filename);

InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

HttpHeaders headers = new HttpHeaders();

headers.add("Content-Disposition", String.format("attachment; filename=\"%s\"", file.getName()));

headers.add("Cache-Control", "no-cache, no-store, must-revalidate");

headers.add("Pragma", "no-cache");

headers.add("Expires", "0");

ResponseEntity<Object>

responseEntity = ResponseEntity.ok().headers(headers).contentLength(file.length()).contentType(

MediaType.parseMediaType("application/txt")).body(resource);

return responseEntity;

}

Code đầy đủ dưới đây −

package com.tutorialspoint.demo.controller;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import org.springframework.core.io.InputStreamResource;

import org.springframework.http.HttpHeaders;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

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

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class FileDownloadController {

@RequestMapping(value = "/download", method = RequestMethod.GET)

public ResponseEntity<Object> downloadFile() throws IOException {

String filename = "/var/tmp/mysql.png";

File file = new File(filename);

InputStreamResource resource = new InputStreamResource(new FileInputStream(file));

HttpHeaders headers = new HttpHeaders();

headers.add("Content-Disposition", String.format("attachment; filename=\"%s\"", file.getName()));

headers.add("Cache-Control", "no-cache, no-store, must-revalidate");

headers.add("Pragma", "no-cache");

headers.add("Expires", "0");

ResponseEntity<Object>

responseEntity = ResponseEntity.ok().headers(headers).contentLength(

file.length()).contentType(MediaType.parseMediaType("application/txt")).body(resource);

return responseEntity;

}

}

Code hàm main của ứng dụng SB −

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);

}

}

Code file pom.xml (Maven) −

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

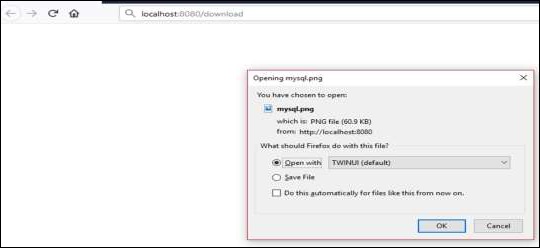
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

POSTMAN Application

Truy cập vào đường dẫn URL trong ứng dụng POSTMAN kết quả hiển thị như bên dưới −

Tải file lên − **http://localhost:8080/upload**



Tải file xuống − **http://localhost:8080/upload**

Service Component

Service Components là class file có chứa @Service annotation. Các class file này được sử dụng để ghi các business logic trên một layer riêng biệt, tách biệt với @RestControll class file. Logic để tạo 1 service component class file được thể hiện như sau −

public interface ProductService {

}

Class thừa kế Interface có @Service annotation sẽ như vầy-

@Service

public class ProductServiceImpl implements ProductService {

}

Theo như trong hướng dẫn này, chúng ta đang sử dụng (các) Product Service API để lưu trữ, truy xuất, cập nhật và xóa các product (sản phẩm). Chúng ta đã viết business logic trong chính @RestControll class file. Bây giờ, chúng ta sẽ chuyển mã business logic từ controller sang service component.Bạn có thể tạo 1 Interface chứa các phương thức - thêm, - sửa, - nhận và - xóa bằng cách sử dụng code như dưới đây −

package com.tutorialspoint.demo.service;

import java.util.Collection;

import com.tutorialspoint.demo.model.Product;

public interface ProductService {

public abstract void createProduct(Product product);

public abstract void updateProduct(String id, Product product);

public abstract void deleteProduct(String id);

public abstract Collection<Product> getProducts();

}

Đoạn code mẫu sau cho phép bạn tạo một class kế thừa ProductService interface (có @Service annotation) và viết business logic để lưu trữ, truy xuất, xóa và cập nhật product.

package com.tutorialspoint.demo.service;

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

import org.springframework.stereotype.Service;

import com.tutorialspoint.demo.model.Product;

@Service

public class ProductServiceImpl implements ProductService {

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);

}

@Override

public void createProduct(Product product) {

productRepo.put(product.getId(), product);

}

@Override

public void updateProduct(String id, Product product) {

productRepo.remove(id);

product.setId(id);

productRepo.put(id, product);

}

@Override

public void deleteProduct(String id) {

productRepo.remove(id);

}

@Override

public Collection<Product> getProducts() {

return productRepo.values();

}

}

Đoạn code ở đây hiển thị Rest Controller class file, ở đây chúng tôi @Autowired ProductService interface và đã gọi đến các phương thức.

package com.tutorialspoint.demo.controller;

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

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.model.Product;

import com.tutorialspoint.demo.service.ProductService;

@RestController

public class ProductServiceController {

@Autowired

ProductService productService;

@RequestMapping(value = "/products")

public ResponseEntity<Object> getProduct() {

return new ResponseEntity<>(productService.getProducts(), HttpStatus.OK);

}

@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)

public ResponseEntity<Object>

updateProduct(@PathVariable("id") String id, @RequestBody Product product) {

productService.updateProduct(id, product);

return new ResponseEntity<>("Product is updated successsfully", HttpStatus.OK);

}

@RequestMapping(value = "/products/{id}", method = RequestMethod.DELETE)

public ResponseEntity<Object> delete(@PathVariable("id") String id) {

productService.deleteProduct(id);

return new ResponseEntity<>("Product is deleted successsfully", HttpStatus.OK);

}

@RequestMapping(value = "/products", method = RequestMethod.POST)

public ResponseEntity<Object> createProduct(@RequestBody Product product) {

productService.createProduct(product);

return new ResponseEntity<>("Product is created successfully", HttpStatus.CREATED);

}

}

Code cho POJO class – Product.java −

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 của ứng dụng SB−

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);

}

}

Code file pom.xml (Maven) −

<?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>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

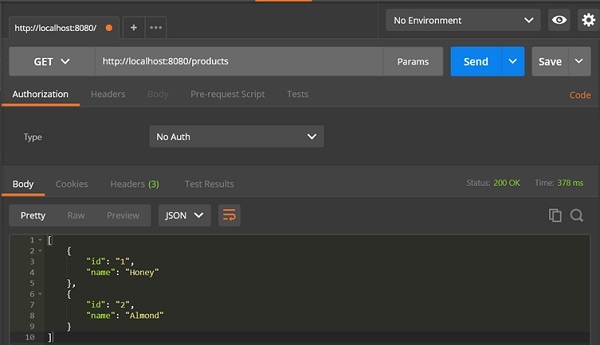
java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

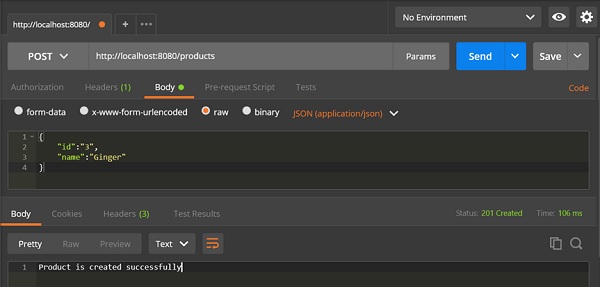
BUILD SUCCESSFUL

Truy cập đường dẫn URL trong ứng dụng POSTMAN bạn sẽ thu được kết quả đầu ra như sau −

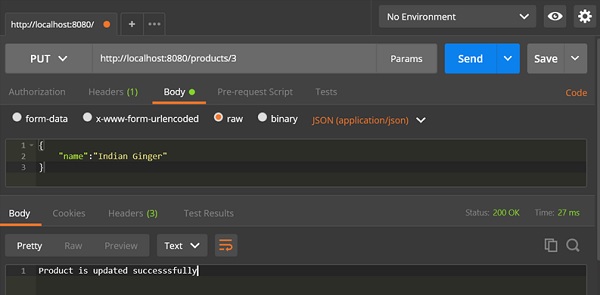
GET API URL − **http://localhost:8080/products**



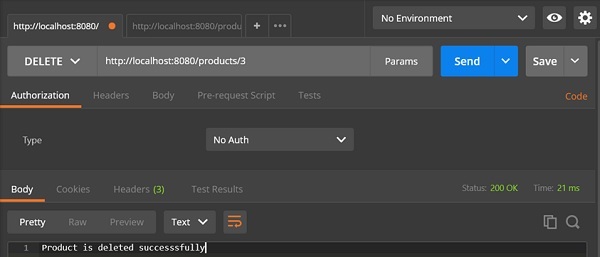
POST API URL − **http://localhost:8080/products**



PUT API URL − [**http://localhost:8080/products/3**](http://localhost:8080/products/3)



DELETE API URL is − **http://localhost:8080/products/3**



Thymeleaf

Thymeleaf là một thư viện dựa trên Java được sử dụng để tạo một ứng dụng web. Nó cung cấp các hỗ trợ tốt phục vụ XHTML / HTML5 trong các ứng dụng web. Trong chương này, chúng ta sẽ tìm hiểu chi tiết về Thymeleaf.

Thymeleaf Templates

Thymeleaf chuyển đổi các tệp của bạn thành các tệp XML được định dạng khá tốt. Nó chứa 6 loại mẫu như được đưa ra dưới đây −

* XML
* Valid XML
* XHTML
* Valid XHTML
* HTML5
* Legacy HTML5

Tất cả các mẫu, ngoại trừ HTML5 Legacy, đều được triển khai thành các tệp XML được định dạng. HTML5 Legacy cho phép chúng ta hiển thị các thẻ HTML5 trong trang web bao gồm các thẻ không đóng.

Web Application

Bạn có thể sử dụng các mẫu Thymeleaf để tạo một ứng dụng web trong Spring Boot. Thử tạo 1 ứng dụng mẫu qua các bước sau.

Sử dụng đoạn code sau để tạo file class @Controll nhằm chuyển hướng Request URI sang tệp HTML −

package com.tutorialspoint.demo.controller;

import org.springframework.stereotype.Controller;

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

@Controller

public class WebController {

@RequestMapping(value = "/index")

public String index() {

return "index";

}

}

Trong ví dụ trên, Request URI là / index và sự điều khiển được chuyển hướng vào tệp index.html. Lưu ý rằng tệp index.html nên được đặt trong templates và tất cả các tệp JS và CSS nên được đặt trong static. Trong ví dụ hiển thị, đã sử dụng tệp CSS để thay đổi màu của văn bản.

Bạn có thể sử dụng đoạn code sau và tạo một tệp CSS trong css thư mục riêng và đặt tên tệp là Styles.css −

h4 {

color: red;

}

Code cho tệp index.html được đưa ra dưới đây −

<!DOCTYPE html>

<html>

<head>

<meta charset = "ISO-8859-1" />

<link href = "css/styles.css" rel = "stylesheet"/>

<title>Spring Boot Application</title>

</head>

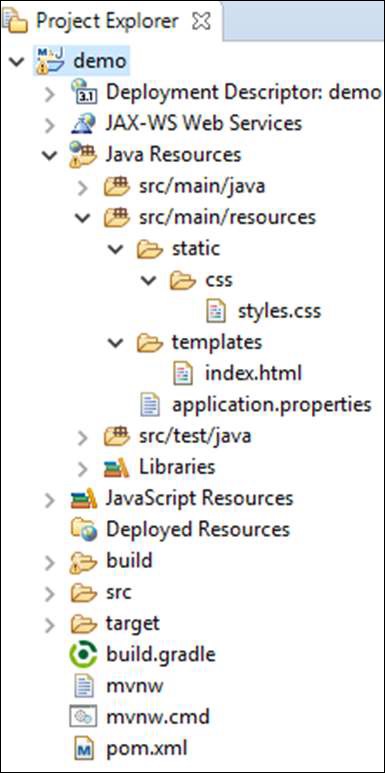
<body>

<h4>Welcome to Thymeleaf Spring Boot web application</h4>

</body>

</html>

Cấu trúc file tổng thể −



Nhớ hãy thêm Spring Boot Starter Thymeleaf dependency trong file cấu hình.

Maven pom.xml file −

<dependency>

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

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

</dependency>

Code cho main Spring Boot application class file −

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);

}

}

Code Maven – 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>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SB bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

Started Application on Tomcat Port_8080

Link URL xem kết quả −

**http://localhost:8080/index**



CORS Support

Chia sẻ tài nguyên chéo (CORS) là một khái niệm bảo mật cho phép hạn chế các tài nguyên được triển khai trong các trình duyệt web. Nó ngăn chặn mã JavaScript sản xuất hoặc tiêu thụ các yêu cầu đối với nguồn gốc khác nhau.

Ví dụ: ứng dụng web của bạn đang chạy trên cổng 8080 và bằng cách sử dụng JavaScript, bạn đang cố gắng tiêu thụ các dịch vụ web RESTful từ cổng 9090. Trong các tình huống như vậy, bạn sẽ phải đối mặt với vấn đề bảo mật Chia sẻ tài nguyên nguồn gốc chéo trên các trình duyệt web của mình.

Cần có hai yêu cầu để xử lý vấn đề này −

• Các dịch vụ web RESTful nên hỗ trợ Chia sẻ tài nguyên gốc chéo.

• Ứng dụng dịch vụ web RESTful sẽ cho phép truy cập (các) API từ cổng 8080.

Trong chương này, chúng ta sẽ tìm hiểu chi tiết về Cách đưa vào sử dụng chức năng Yêu cầu nguồn gốc chéo cho 1 ứng dụng RESTful Web Service..

## Enable CORS in Controller Method

Chúng ta cần đặt nguồn gốc cho dịch vụ web RESTful sử dụng @CrossOrigin annotation cho phương thức điều khiển. Chú thích @CrossOrigin này hỗ trợ API REST cụ thể và không dành cho toàn bộ ứng dụng.

@RequestMapping(value = "/products")

@CrossOrigin(origins = "http://localhost:8080")

public ResponseEntity<Object> getProduct() {

return null;

}

## Global CORS Configuration

Chúng ta cần xác định cấu hình @Bean được hiển thị để đặt hỗ trợ cấu hình CORS trên toàn cục cho ứng dụng Spring Boot của bạn.

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/products").allowedOrigins("http://localhost:9000");

}

};

}

Code để đặt cấu hình CORS trên toàn cầu trong ứng dụng Spring Boot chính được đưa ra dưới đây.

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/products").allowedOrigins("http://localhost:8080");

}

};

}

}

Giờ đây, bạn có thể tạo một ứng dụng web Spring Boot chạy trên cổng 8080 và ứng dụng RESTful web service của bạn có thể chạy trên cổng 9090.

Internationalization

Internationalation giúp ứng dụng của bạn có thể chuyển đổi phù hơp sang các ngôn ngữ khác mà không bị thay đổi về source code.

Chương này chỉ bạn cách thừa kế sử dụng Internationalization.

## Dependencies

Chúng ta cần Spring Boot Starter Web và Spring Boot Starter Thymeleaf dependency để triển khai 1 ứng dụng trong Spring Boot.

### Maven

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

## LocaleResolver

Chúng ta cần khai báo vị trí mặc định - Locale của ứng dụng. Thêm 1 bean là LocaleResolver bean vào ứng dụng Spring Boot của bạn để giải quyết vị trí mặc định.

@Bean

public LocaleResolver localeResolver() {

SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();

sessionLocaleResolver.setDefaultLocale(Locale.US);

return sessionLocaleResolver;

}

## LocaleChangeInterceptor

LocaleChangeInterceptor dùng để đổi vị trí ngôn ngữ tương ứng tương ứng với vị trí ngôn ngữ yêu cầu đổi.

@Bean

public LocaleChangeInterceptor localeChangeInterceptor() {

LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();

localeChangeInterceptor.setParamName("language");

return localeChangeInterceptor;

}

Để có được hiệu ứng này, chúng ta cần thêm LocaleChangeInterceptor vào registry interceptor của ứng dụng. Lớp cấu hình sẽ extend lớp WebMvcConfigurerAdapter và ghi đè phương thức addInterceptors ().

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(localeChangeInterceptor());

}

## Messages Sources

Ứng dụng Spring Boot theo mặc định lấy các nguồn thông báo từ thư mục **src / main / resource** dưới đường dẫn lớp. Tên file tin nhắn cục bộ mặc định phải là **message.properies** và các tệp cho mỗi miền sẽ đặt tên là **message\_XX.properies**. Mã XX XX thể hiện mã địa phương.Tất cả các thuộc tính thông báo nên được sử dụng làm giá trị khóa cặp. Nếu bất kỳ thuộc tính nào không được tìm thấy trên miền địa phương, ứng dụng sẽ sử dụng thuộc tính mặc định từ tệp tin.properations.

messages.properties mặc định sẽ được hiển thị như sau −

welcome.text=Hi Welcome to Everyone

ngôn ngữ tiếng Pháp messages\_fr.properties sẽ hiển thị như sau −

welcome.text=Salut Bienvenue à tous

**Note** − File message nguồn nên được lưu dưới dạng định dạng tập tin UTF-8.

## HTML file

Trong file HTML , dùng syntax **#{key}** để biển thị tin nhắn từ properties file.

<h1 th:text = "#{welcome.text}"></h1>

Code đầy đủ như sau-

### Maven – 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>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

}

Main Spring Boot application class file như sau −

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);

}

}

controller class file như sau −

package com.tutorialspoint.demo.controller;

import org.springframework.stereotype.Controller;

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

@Controller

public class ViewController {

@RequestMapping("/locale")

public String locale() {

return "locale";

}

}

Configuration class để hỗ trợ cho Internationalization

package com.tutorialspoint.demo;

import java.util.Locale;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.LocaleResolver;

import org.springframework.web.servlet.config.annotation.InterceptorRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;

import org.springframework.web.servlet.i18n.SessionLocaleResolver;

@Configuration

public class Internationalization extends WebMvcConfigurerAdapter {

@Bean

public LocaleResolver localeResolver() {

SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();

sessionLocaleResolver.setDefaultLocale(Locale.US);

return sessionLocaleResolver;

}

@Bean

public LocaleChangeInterceptor localeChangeInterceptor() {

LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();

localeChangeInterceptor.setParamName("language");

return localeChangeInterceptor;

}

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(localeChangeInterceptor());

}

}

Message sources – messages.properties hiển thị như sau −

welcome.text = Hi Welcome to Everyone

Message sources – message\_fr.properties hiển thị như sau −

welcome.text = Salut Bienvenue à tous

file HTML locale.html nên được đặt dưới thư mục mẫu trên đường dẫn các lớp như được hiển thị −

<!DOCTYPE html>

<html>

<head>

<meta charset = "ISO-8859-1"/>

<title>Internationalization</title>

</head>

<body>

<h1 th:text = "#{welcome.text}"></h1>

</body>

</html>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SpringBoot bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 như vầy-

Started Application on Tomcat Port_8080

Truy nhập URL **http://localhost:8080/locale** trên trình duyệt sẽ thấy đầu ra −



URL **http://localhost:8080/locale?language=fr** sẽ như vầy −



# Scheduling

Scheduling là một chuỗi quá trình thực hiện các nhiệm vụ trong khoảng thời gian cụ thể. Spring Boot cung cấp hỗ trợ tốt để tạo lịch trình trên các ứng dụng Spring.

## Java Cron Expression

Java Cron expressions được sử dụng để định cấu hình các thể hiện của CronTrigger, một lớp con của org.quartz.Trigger. Để biết thêm thông tin về Java cron expression, bạn có thể tham khảo liên kết này −

<https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm>

@EnableScheduling annotation được sử dụng để kích hoạt lịch trình cho ứng dụng của bạn. Chú thích này nên được thêm vào file class main của ứng dụng Spring Boot.

@SpringBootApplication

@EnableScheduling

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

@Scheduled annotation được sử dụng để kích hoạt bộ lập lịch trong một khoảng thời gian cụ thể.

@Scheduled(cron = "0 \* 9 \* \* ?")

public void cronJobSch() throws Exception {

}

Sau đây là code mẫu hướng dẫn cách khởi chạy tác vụ mỗi phút bắt đầu lúc 9:00 sáng và kết thúc lúc 9:59 sáng, mỗi ngày

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

@Scheduled(cron = "0 \* 9 \* \* ?")

public void cronJobSch() {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");

Date now = new Date();

String strDate = sdf.format(now);

System.out.println("Java cron job expression:: " + strDate);

}

}

Ảnh chụp màn hình sau đây cho thấy ứng dụng đã bắt đầu lúc 09:03:23 và cứ sau một phút kể từ thời điểm đó, tác vụ lập lịch công việc cron đã thực thi.

cron job scheduler

## Fixed Rate

Fixed Rate scheduler được sử dụng để thực hiện các tác vụ tại thời điểm cụ thể. Nó không chờ đợi để hoàn thành nhiệm vụ trước đó. Các giá trị nên được tính bằng mili giây. Code mẫu được hiển thị ở đây −

@Scheduled(fixedRate = 1000)

public void fixedRateSch() {

}

Code mẫu để thực thi một tác vụ mỗi giây từ khi khởi động ứng dụng được hiển thị ở đây −

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

@Scheduled(fixedRate = 1000)

public void fixedRateSch() {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");

Date now = new Date();

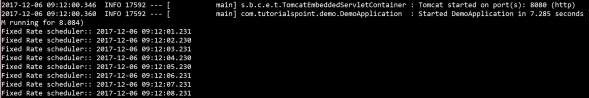
String strDate = sdf.format(now);

System.out.println("Fixed Rate scheduler:: " + strDate);

}

}

Quan sát ảnh chụp màn hình sau đây cho thấy ứng dụng đã bắt đầu lúc 09:12:00 và sau đó mỗi giây, các tác vụ fixed rate scheduler đã được thực hiện.



## Fixed Delay

Fixed Delay scheduler được sử dụng để thực hiện các tác vụ tại một thời điểm cụ thể. Nó sẽ chờ hoàn thành nhiệm vụ trước đó. Các giá trị nên được tính bằng mili giây. Code mẫu được hiển thị ở đây −

@Scheduled(fixedDelay = 1000, initialDelay = 1000)

public void fixedDelaySch() {

}

Ở đây, initDelay là thời gian sau đó tác vụ sẽ được thực hiện lần đầu tiên sau khoảng thời gian initDelay.

Một ví dụ để thực thi tác vụ cho mỗi giây sau 3 giây kể từ khi khởi động ứng dụng đã hoàn thành được hiển thị bên dưới −

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

@Scheduled(fixedDelay = 1000, initialDelay = 3000)

public void fixedDelaySch() {

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");

Date now = new Date();

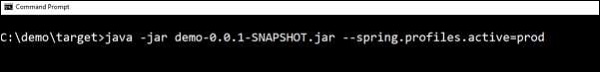
String strDate = sdf.format(now);

System.out.println("Fixed Delay scheduler:: " + strDate);

}

}

Quan sát ảnh chụp màn hình sau đây cho thấy ứng dụng đã bắt đầu lúc 09:18:39 và cứ sau 3 giây, tác vụ fixed delay scheduler đã thực hiện trên mỗi giây.



# Enabling HTTPS

Theo mặc định, ứng dụng Spring Boot sử dụng cổng HTTP 8080 khi ứng dụng khởi động.

Started Application on Tomcat port_8080

Bạn cần làm theo các bước được đưa ra dưới đây để cấu hình HTTPS và cổng 443 trong ứng dụng Spring Boot −

* Lấy SSL certificate - tự tạo certificate hoặc lấy từ một Certificate Authority.
* Kích hoạt HTTPS và cổng 443

Self-Signed Certificate (tự tạo 1 certificate)

Để tự tạo một self-signed certificate, cần môi trường Java Run Time đi kèm với công cụ khóa tiện ích quản lý chứng chỉ. Công cụ tiện ích này được sử dụng để tạo Self-Signed certificate. Nó được hiển thị trong code được đưa ra ở đây −

keytool -genkey -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]:

What is the name of your organizational unit?

[Unknown]:

What is the name of your organization?

[Unknown]:

What is the name of your City or Locality?

[Unknown]:

What is the name of your State or Province?

[Unknown]:

What is the two-letter country code for this unit?

[Unknown]:

Is CN = Unknown, OU=Unknown, O = Unknown, L = Unknown, ST = Unknown, C = Unknown correct?

[no]: yes

Code này sẽ generate một file keystore PKCS12 có tên là keystore.p12 với tên certifacate bí danh (alias certificate name) là tomcat.

Configure HTTPS

Chúng ta cần cung cấp cổng server như 443, đường dẫn tệp lưu trữ khóa, mật khẩu lưu trữ khóa, loại keystore và tên bí danh chính (alias name) vào tệp application.properations. Quan sát code được đưa ra ở đây −

server.port: 443

server.ssl.key-store: keystore.p12

server.ssl.key-store-password: springboot

server.ssl.keyStoreType: PKCS12

server.ssl.keyAlias: tomcat

Bạn có thể sử dụng code sau nếu bạn đang sử dụng các thuộc tính YAML sử dụng bên dưới application.yml −

server:

port: 443

ssl:

key-store: keystore.p12

key-store-password: springboot

keyStoreType: PKCS12

keyAlias: tomcat

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SpringBoot bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 443 như vầy-

Started Application on Tomcat Port_8080

# Eureka Server

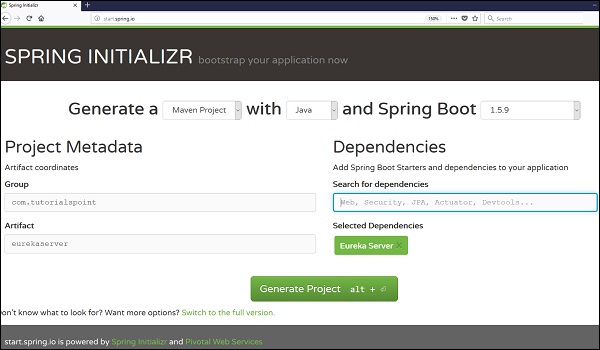
Eureka Server là một ứng dụng chứa thông tin về tất cả các ứng dụng dịch vụ khách hàng. Mỗi Micro service sẽ đăng ký vào máy chủ Eureka và máy chủ Eureka biết tất cả các ứng dụng khách đang chạy trên mỗi cổng và địa chỉ IP. Eureka Server còn được gọi là Discovery Server.

Trong chương này, chúng ta sẽ tìm hiểu chi tiết về Cách xây dựng Eureka server.

## Xây dụng một Eureka Server

Eureka Server đi kèm với gói Spring Cloud. Chúng ta cần phát triển Eureka server và chạy nó trên cổng mặc định 8761.

Truy cập trang chủ Spring Initializer <https://start.spring.io/> và tải Spring Boot project with Eureka server dependency như ảnh chụp dưới đây −



Sau khi tải xuống project trong file main Spring Boot Application class, chúng ta cần thêm annotation @EnableEurekaServer. Chú thích @EnableEurekaServer được sử dụng để làm cho ứng dụng Spring Boot của bạn hoạt động như một Eureka Server.

Code cho file main Spring Boot application class−

package com.tutorialspoint.eurekaserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

}

}

Đảm bảo Spring cloud Eureka server dependency được thêm vào trong file build cấu hình của bạn. Code cho Maven dependency được hiển thị bên dưới −

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

File build cấu hình hoàn chỉnh được đưa ra dưới đây −

**Maven 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>eurekaserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eurekaserver</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Edgware.RELEASE</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

File application.yml như sau−

eureka:

client:

registerWithEureka: false

fetchRegistry: false

server:

port: 8761

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SpringBoot bằng Maven commands bằng lệnh-

mvn clean install

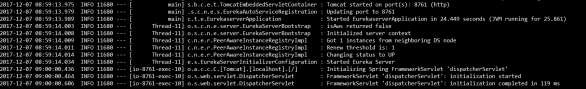
sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

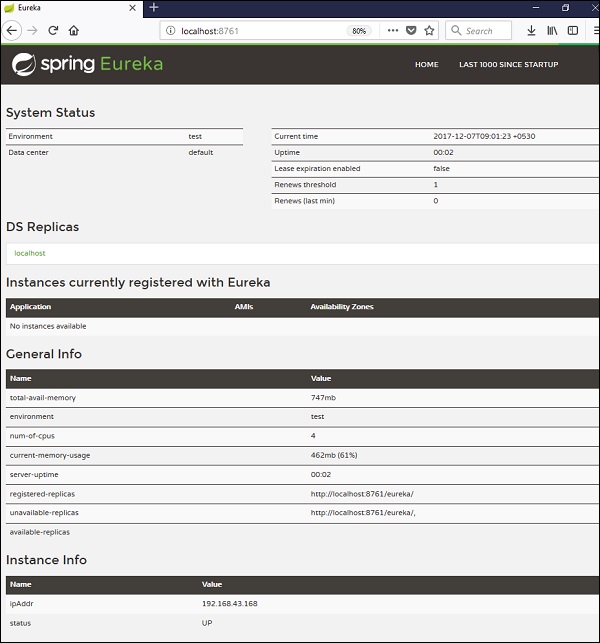
Có thể chạy file JAR bằng cách sử dụng lệnh-

java –jar <JARFILE>

chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8761 như vầy−



Truy cập URL **http://localhost:8761/** trên trình duyệt, bạn có thể thấy Eureka Server đang chạy trên cổng 8761 như hình sau −



# Service Registration with Eureka

Trong chương này, bạn sẽ tìm hiểu chi tiết về Cách khai báo ứng dụng Spring Boot Micro Service vào Eureka Server. Trước khi thực hiện, đảm bảo rằng Eureka Server đang chạy trên cổng 8761 hoặc trước tiên hãy xây dựng Eureka Server và chạy nó. Để biết thêm thông tin về việc xây dựng Eureka Server, bạn có thể tham khảo chương trước.

Trước tiên, bạn cần thêm các dependency sau vào tệp cấu hình bản dựng của chúng tôi để đăng ký microservice với Eureka server.

Người dùng Maven có thể thêm các dependency sau vào tệp **pom.xml** −

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

Bây giờ, chúng ta cần thêm chú thích @EnableEurekaClient trong file main class của ứng dụng Spring Boot. Chú thích @EnableEurekaClient làm cho ứng dụng Spring Boot của bạn hoạt động như một ứng dụng client Eureka.

Code hàm main của Spring Boot sẽ như sau −

package com.tutorialspoint.eurekaclient;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

public class EurekaclientApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaclientApplication.class, args);

}

}

Để đăng ký ứng dụng Spring Boot vào Eureka Server, chúng ta cần thêm cấu hình sau vào tệp application.properations hoặc tệp application.yml và chỉ định URL Eureka Server trong cấu hình của chúng ta.

Code cho file application.yml như sau −

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka

instance:

preferIpAddress: true

spring:

application:

name: eurekaclient

Code cho file application.properties như sau −

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

eureka.client.instance.preferIpAddress = true

spring.application.name = eurekaclient

Bây giờ, thêm Rest Endpoint để trả về Chuỗi trong hàm main ứng dụng Spring Boot và Spring Boot Starter web dependency trong tệp cấu hình . Như code được đưa ra dưới đây −

package com.tutorialspoint.eurekaclient;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

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

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@EnableEurekaClient

@RestController

public class EurekaclientApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaclientApplication.class, args);

}

@RequestMapping(value = "/")

public String home() {

return "Eureka Client application";

}

}

File cấu hình đầy đủ.

**For Maven user - 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>eurekaclient</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eurekaclient</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Edgware.RELEASE</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

<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>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</projecta>

Có thể tạo 1 file thực thi JAR và chạy ứng dụng SpringBoot bằng Maven commands bằng lệnh-

mvn clean install

sau khi “BUILD SUCCESS” bạn có thể tìm file JAR dưới đường dẫn chỉ định.

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

Có thể chạy file JAR bằng cách sử dụng lệnh-

java –jar <JARFILE>

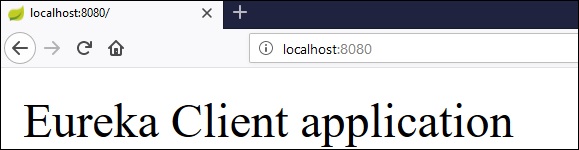
chạy lệnh này sẽ khởi chạy ứng dụng trên Tomcat cổng 8080 và ứng dụng Eureka Client trên Eureka Server như vầy−

Started Application on Tomcat Port

Truy nhập URL http://localhost:8761/ trên trình duyệt bạn sẽ thấy được ứng dụng Eureka Client đã được đăng ký trên Eureka Server.



Truy nhập URL **http://localhost:8080/** trên trình duyệt để xem đầu ra Rest Endpoint output.



# Zuul Proxy Server and Routing

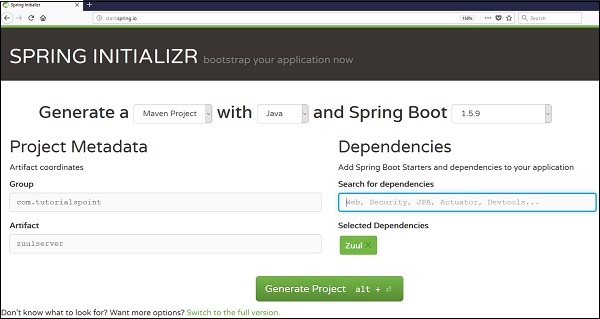
Zuul Server is a gateway application that handles all the requests and does the dynamic routing of microservice applications. The Zuul Server is also known as Edge Server.

For Example, **/api/user** is mapped to the user service and /api/products is mapped to the product service and Zuul Server dynamically routes the requests to the respective backend application.

In this chapter, we are going to see in detail how to create Zuul Server application in Spring Boot.

## Creating Zuul Server Application

The Zuul Server is bundled with Spring Cloud dependency. You can download the Spring Boot project from Spring Initializer page <https://start.spring.io/> and choose the Zuul Server dependency.



Add the @EnableZuulProxy annotation on your main Spring Boot application. The @EnableZuulProxy annotation is used to make your Spring Boot application act as a Zuul Proxy server.

package com.tutorialspoint.zuulserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication

@EnableZuulProxy

public class ZuulserverApplication {

public static void main(String[] args) {

SpringApplication.run(ZuulserverApplication.class, args);

}

}

You will have to add the Spring Cloud Starter Zuul dependency in our build configuration file.

Maven users will have to add the following dependency in your **pom.xml** file −

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

For Zuul routing, add the below properties in your application.properties file or application.yml file.

spring.application.name = zuulserver

zuul.routes.products.path = /api/demo/\*\*

zuul.routes.products.url = http://localhost:8080/

server.port = 8111

This means that http calls to **/api/demo/** get forwarded to the products service. For example, **/api/demo/products** is forwarded to **/products**.

yaml file users can use the application.yml file shown below −

server:

port: 8111

spring:

application:

name: zuulserver

zuul:

routes:

products:

path: /api/demo/\*\*

url: http://localhost:8080/

**Note** − The **http://localhost:8080/** application should already be running before routing via Zuul Proxy.

The complete build configuration file is given below.

Maven users can use the pom.xml file given below −

<?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>zuulserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>zuulserver</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Edgware.RELEASE</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

You can create an executable JAR file, and run the Spring Boot application by using the Maven or Gradle commands given below −

For Maven, you can use the command given below −

mvn clean install

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

For Gradle, you can use the command given below −

gradle clean build

After “BUILD SUCCESSFUL”, you can find the JAR file under the build/libs directory.

Now, run the JAR file by using the command shown below −

java –jar <JARFILE>

You can find the application has started on the Tomcat port 8111 as shown here.

Started Application on Tomcat Port_8111

Now, hit the URL **http://localhost:8111/api/demo/products** in your web browser and you can see the output of **/products** REST Endpoint as shown below −

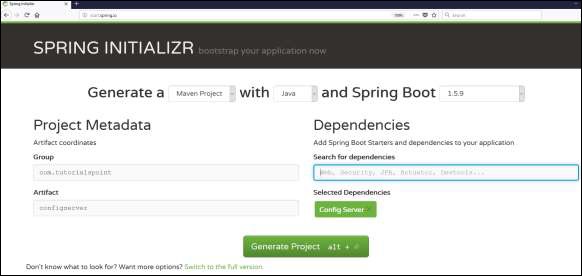


# Cloud Configuration Server

Spring Cloud Configuration Server is a centralized application that manages all the application related configuration properties. In this chapter, you will learn in detail about how to create Spring Cloud Configuration server.

## Creating Spring Cloud Configuration Server

First, download the Spring Boot project from the Spring Initializer page and choose the Spring Cloud Config Server dependency. Observe the screenshot given below −



Now, add the Spring Cloud Config server dependency in your build configuration file as explained below −

Maven users can add the below dependency into the pom.xml file.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

Now, add the @EnableConfigServer annotation in your main Spring Boot application class file. The @EnableConfigServer annotation makes your Spring Boot application act as a Configuration Server.

The main Spring Boot application class file is given below −

package com.tutorialspoint.configserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication

@EnableConfigServer

public class ConfigserverApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigserverApplication.class, args);

}

}

Now, add the below configuration to your properties file and replace the application.properties file into bootstrap.properties file. Observe the code given below −

server.port = 8888

spring.cloud.config.server.native.searchLocations=file:///C:/configprop/

SPRING\_PROFILES\_ACTIVE=native

Configuration Server runs on the Tomcat port 8888 and application configuration properties are loaded from native search locations.

Now, in **file:///C:/configprop/**, place your client application - application.properties file. For example, your client application name is **config-client**, then rename your application.properties file as **config-client.properties** and place the properties file on the path **file:///C:/configprop/**.

The code for config-client properties file is given below −

welcome.message = Welcome to Spring cloud config server

The complete build configuration file is given below −

Maven users can use **pom.xml** given below −

<?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>configserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>configserver</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.9.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Edgware.RELEASE</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Now, create an executable JAR file, and run the Spring Boot application by using the following Maven or Gradle commands −

For Maven, use the command given below −

mvn clean install

After “BUILD SUCCESS”, you can find the JAR file under the target directory.

For Gradle, use the command given below −

gradle clean build

After “BUILD SUCCESSFUL”, you can find the JAR file under the build/libs directory.

Run the JAR file by using the following command −

java –jar <JARFILE>

Now, the application has started on the Tomcat port 8888 as shown here −

Tomcat Port 8888 Output

Now hit the URL **http://localhost:8888/config-client/default/master** on your web browser and you can see your config-client application configuration properties as shown here.

