SPRING BOOT TUTORIAL:

Chapter 1.

-Spring boot là một framework dựa trên Java mã nguồn mở được phát triển bởi Pivotal Team và được sử dụng để xây dựng ứng dụng spring độc lập

- Micro Service là gì:

Micro Service là một kiến trúc cho phép các nhà phát triển phần mềm phát triển và triển khai các services độc lập , mỗi một service đang chạy có 1 tiến trình riêng và điều này mang lại mô hình “nhẹ” để hỗ trợ các ứng dụng xử lí .

* Ưu điểm:
* Micro Service mang lại các lợi ích sau cho nhà phát triển:
* Dễ dàng triển khai
* Dễ dàng mở rộng
* Tương thích với Containers
* Cấu hình đơn giản
* Thời gian phát triển ít hơn

Spring Boot là gì:

Spring boot cung cấp một nền tang tốt cho lập trình viên Java để phát triển một ứng dụng Spring độc lập mà bạn có thể “Run”. Bạn có thể bắt đầu với cấu hình đơn giản mà không cần thiết lập toàn bộ cấu hình Spring.

Ưu điểm

Spring boot cung cấp những ưu điểm sau cho các nhà phát triển

* Dễ hiểu và phát triển các ứng dụng spring
* Tăng năng suất
* Giảm thời gian phát triển ứng dụng

Mục tiêu

Spring boot được thiết kế với các mục tiêu sau :

* Để tránh cấu hình phức tạp XML trong Spring
* Để phát triển một ứng dụng Spring dễ dàng hơn
* Để giảm thiểu thời gian phát triển và chạy ứng dụng độc lập
* Cung cấp cách thức dễ dàng hơn để bắt đầu một ứng dụng

Vì sao nên chọn Spring Boot ?

Bạn có thể chọn Spring Boot vì các tính năng và lợi ích mà nó cung cấp sau đây:

* Nó cung cấp một cách linh hoạt để cấu hình Java Beans , cấu hình XML ,và đảm bảo Tương tác với Database
* Nó cung cấp một quá trình xử lý hang loạt mạnh mẽ và quản lý các Rest endpoints
* Trong Spring Boot , mọi thứ đều được cấu hình tự động
* Cung cấp ứng dụng Spring dựa trên các Annotation
* Quản lý các dependency
* Bao gồm tích hợp Servlet Container

Cách thức hoạt động ?

Spring Boot tự động cấu hình ứng dụng dựa trên các dependencies mà bạn đã them vào project bằng cách sử dụng @EnableAutoConfiguration annotation.

Ví dụ : nếu dữ liệu MySQL nằm trên đường dẫn của bạn nhưng bạn không cấu hình bất kể kết nối cơ sở dữ liệu nào , thì Spring Boot tự động cấu hình một bộ nhớ cơ sở dữ liệu

Điểm vào của ứng dụng Spring là lớp chứa @SpringBootApplication và phương thức chính.

Spring Boot tự động quét tất cả các components có trong project bằng cách sử dụng @ComponentScane annotation

Spring Boot Starter

Xử lý quản lý các dependency là một nhiệm vụ khó khan cho các dự dán lớn. Spring Boot giải quyết vấn đề này bằng cách cung cấp một bộ dependencies để thuận tiện cho các nhà phát triển .

Ví dụ : nếu bạn muốn sử dụng Spring và JPA để truy cấp database , điều đó là khả thi nếu bạn thêm “spring-boot-starter-data-jpa” dependency trong project

Lưu ý : toàn bộ các Spring Boot starter đều tuân theo một quy chuẩn đặt tên “spring-boot-starter-\*” trong đó \* chỉ ra rằng đó là một loại ứng dụng.

Ví dụ :

Theo dõi Spring Boot Starter sau đây kèm giải thích bên dưới:

Spring Boot Starter Actuator dependency được sử dụng để giám sát và quản lý ứng dụng của bạn : Code đc hiển thị như sau :

<dependency>

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

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

</dependency>

Spring Boot Starter Security dependency được sử dụng cho Spring Security :

<dependency>

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

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

</dependency>

Spring Boot Starter web dependency được sử dụng để viết một Rest endpoints :

<dependency>

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

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

</dependency>

Spring Boot Starter Thyme Leaf dependency được sử dụng để tạo 1 ứng dụng web

<dependency>

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

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

</dependency>

Spring Boot Starter Test dependency được sử dụng để tiết Test cases

<dependency>

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

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

</dependency>

Tự động Cấu Hình

Spring Boot Auto Configuration tự động cấu hình ứng dụng Spring dựa trên JAR dependencies bạn đã thêm vào project. Ví dụ:

Nếu dữ liệu MySQL là lớp đường dẫn của bạn nhưng bạn không cấu hình bất kể kết nối cơ sở dữ liệu nào , ngay sau đó Spring Boot tự động cấu hình 1 bộ nhớ cơ sở dữ liệu

Với mục đích trên , bạn cần thêm @EnableAutoConfiguration annotation hoặc @SpringBootApplication annotation vào main class file. Sau đó , ứng dụng Spring Boot sẽ tự động cấu hình.

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 Application

Điểm vào của ứng dụng Spring Boot là class chứa @SpringBootApplication annotation. Class nên có phương thức main để chạy ứng dụng Spring Boot.

@SpringBootApplication annotation bao gồm Auto-Configuration , Component Scan, và Spring Boot Configuration.

Nếu bạn thêm @SpringBootApplication annotation vào class , bạn không cần thêm @EnableAutoConfiguration , @ComponentScan và @SpringBootConfiguration annotation , @SpringBootApplication annotation chứa toàn bộ annotations trên .

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

ứng dụng Spring Boot quét toàn bộ beans và package khai báo khi ứng dụng khởi tạo , bạn nên thêm @ComponentScan annotation vào file class của bạn để quét các components đã thêm vào project.

Chapter 2.

Quick Start.

Cách tạo 1 ứng dụng spring boot bằng maven và gradle

Yêu cầu:

Hệ thống cần có yêu cầu tối thiểu:

* Java 7
* Maven 3.2
* Gradle 2.5

Spring Boot CLI

Spring Boot CLI là 1 command line tool và nó cho phép chúng chạy bằng Groovy scripts. Đây là cách dễ nhất để tạo một ứng dụng Spring Boot bằng cách sử dụng Spring Boot Command Line giao diện. Bạn có thể tạo , chạy và kiểm tra ứng dụng trong command prompt.

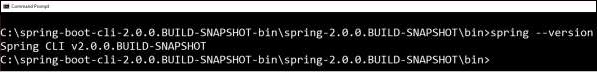
Phần này giải thích cho bạn các bước liên quan đến cài đặt thủ công Spring Boot CLI . Tìm hiểu thêm tại: https://docs.spring.io/springboot/ docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-springboot

Khởi tạo thủ công , bạn cần sử dụng 2 folder:

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

Sau khi download , giải nén file lưu trữ và làm theo các bước trong install.txt , không cần bắt buộc cài đặt môi trường.

Trong windows , hãy chuyển đến thư mục bin Spring Boot CLI trong command prompt và chạy lệnh spring –version để đảm bảo spring CLI đã được cài đặt đúng. Sau khi thực hiện lệnh, bạn có thể thấy phiên bản spring CLI :



Chạy Hello World với Groovy

Tạo một file groovy đơn giản với tên hello.groovy. lưu ý rằng trong ví dụ này , chúng tôi đã saved groovy file bên trong Spring Boot CLI bin directory. Chạy ứng dụng bằng cách sử dụng lệnh spring run hello.groovy :

Run Hello World with Groovy

Mỗi lần bạn chạy groovy file , các dependencies yêu cầu sẽ tự động download và sẽ khởi động ứng dụng trong Tomcat tại cổng 8080

Run Groovy File Tomcat Port

Khi tomcat khởi động , đi đến giao diện web và truy cập đường dẫn localhost:8080 và bạn có thể thấy output:



Chapter 3.

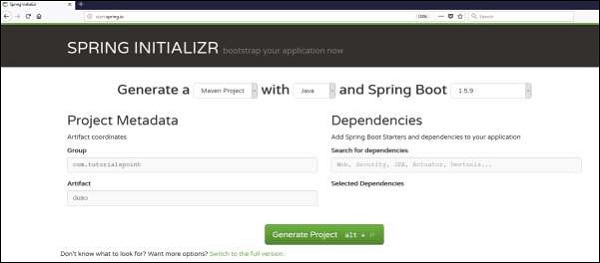
Bootstrapping

Chương này sẽ giải thích cho bạn cách thực hiện bootstrapping trên ứng dụng Spring Boot.

Spring Initializer

Một trong những cách để Bootstrapping ứng dụng Spring Boot là sử dụng Spring Initializer . để thực hiện việc này , bạn sẽ phải truy cập Spring Initializer trang web [www.start.spring.io](http://www.start.spring.io) và lựa chọn Build , phiên bản Spring Boot và nền tảng. Ngoài ra , bạn cần cung cấp một Group , Artifact và các dependencies bắt buộc để chạy ứng dụng.

Một ví dụ trong đó chúng tôi đã thêm dependency spring-boot-starter-web để viết Rest Endpoints.



Khi bạn cung cấp Group , Artifact , Dependencies , Build Project , nền tảng và phiên bản , click Generate Project button. Một file zip sẽ được tải xuống và giải nén.

Phần này sẽ giải thích ví dụ sử dụng cả Maven và Gradle.

MAVEN

Sau khi tải xuống project , giải nén file . file pom có dạng 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

Khi download project , giải nén file , build.gradle sẽ như sau:

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 số lượng Starters để thêm jars vào đường dẫn class , ví dụ :

Để viết một Rest Endpoint , chúng tôi cần thêm spring-boot-starter-web dependency vào đường dẫn class .

Main Method

Main method nên viết trong lớp Spring Boot Application. Class nên đc chú thích với @SpringBootApplication. Đây là entry point của spring boot application để khởi chạy. Bạn có thể tìm file lớp main trong thư mục: src/java/main với package mặc định.

Trong ví dụ này . file lớp main sẽ đc đặt tại thư mục: src/java/main với package mặc định: com.tutorialspoint.demo.

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);

}

}

Write a Rest Endpoint

Để viết 1 ví dụ Hello World Rest Endpoint trong Spring Boot Application main class , thực hiện theo các bước sau

* Đầu tiên , thêm @RestController annotation vào đầu class
* Giờ đây , viết a request URL method với @RequestMappiong annotation
* Sau đó , request URL method nên return chuỗi Hello World

Giờ đây , main ứng dụng Spring Boot sẽ trông như sau :

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";

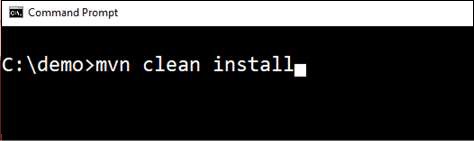
}

}

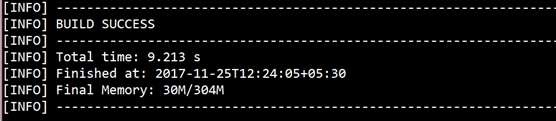
## Create an Executable JAR

Tạo 1 executable JAR file để chạy ứng dụng Spring Boot bằng cách sử dụng Maven và Gradle commands trong command prompt :

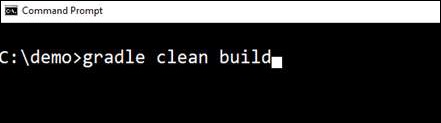
Sử dụng Maven command mvn clean install :



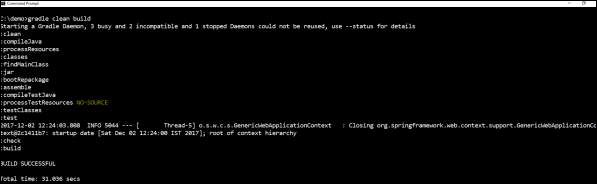
Sau khi chạy command , bạn có thể thấy tin nhắn BUILD SUCCESS tại command prompt như sau :



Sử dụng Gradle command gradle clean build như sau:



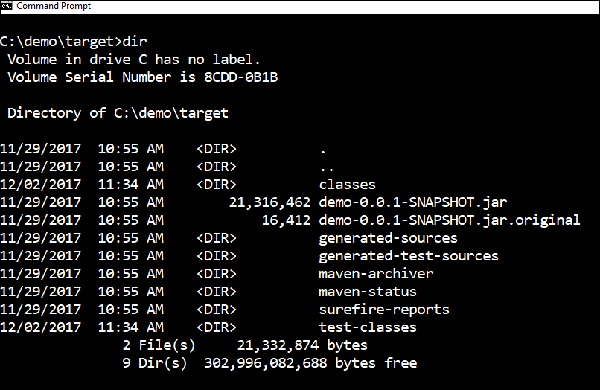
Sau khi chạy command , bạn có thể nhìn tin nhắn BUILD SUCCESSFUL trong command prompt như sau :



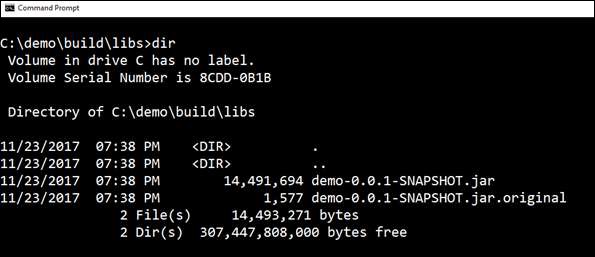
## Run Hello World with Java

khi bạn tạo một executable JAR file , bạn có thể tìm tìm thấy nó trong thư mục

Với Maven , bạn có thể tìm trong JAR file trông thư mục chỉ định :



Với Gradle , bạn có thể tìm thấy JAR file trong thư mục build/libs



Bây giờ , chạy JAR file sử dụng command java –jar<JARFILE>



Khi chạy JAR file , bạn có thể nhìn output dưới console window

Output in Console Window

Truy cập giao diện web theo đường dẫn <http://localhost:8080/>



Chapter 4.

Tomcat deployment

Bằng cách sử dụng ứng dụng Spring Boot , chúng ta có thể tạo một war file để deploy vào web server . Trong chap này , bạn sẽ học cách để tạo một file WAR và deploy ứng dụng Spring Boot lên Tomcat web server.

## Spring Boot Servlet Initializer

Cách triển khai là tạo ứng dụng Spring Boot @SpringBootApplication class kế thừa SpringBootServletInitializer class. Spring Boot Servlet Intializer class cho phép bạn cấu hình ứng dụng khi chúng đc chạy bằng cách sử dụng Servlet Container.

Chúng ta cần kế thừa class SpringBootServletInitializer để hỗ trợ triển khai WAR file.

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 cần đề cập main class nên khởi chạy quá trình build file. Để thực hiện mục đích đó , bạn cần sử dụng đoạn code sau :

Với Maven , thêm class khởi chạy trong cấu hình file pom.xml

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

Với Gradle , thêm tên main class trong build.gradle

mainClassName="com.tutorialspoint.demo.DemoApplication"

## Update packaging JAR into WAR

Chúng ta phải update packaging JAR vào WAR sử dụng code dưới đây:

Với Maven , thêm packaging WAR vào file pom.xml

<packaging>war</packaging>

Với Gradle , thêm application plugin và war plugin trong build.gradle

apply plugin: ‘war’

apply plugin: ‘application’

Bây giờ , chúng ta hãy viết một ví dụ Rest Endpoint để trả về chuỗi “Hello World from Tomcat”. Để viết một Rest Endpoint , chúng ta nên thêm Spring Boot web starter dependency vào build file của chúng ta.

Với Maven :

<dependency>

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

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

</dependency>

Với Gradle:

dependencies {

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

}

Ví dụ đơn giản một Rest Endpoint trong Spring Boot Application class file:

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";

}

}

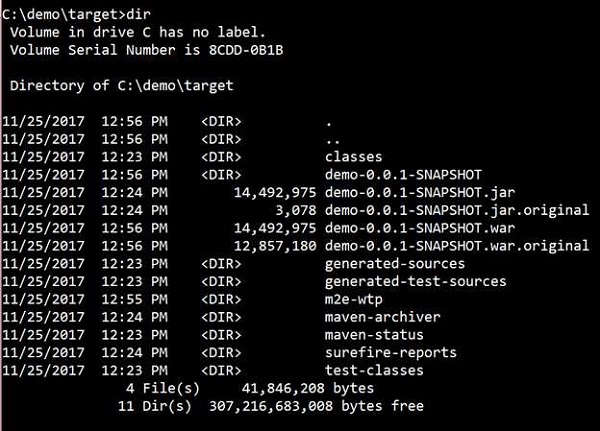
## Packaging your Application

Bây giờ , tạo một WAR file để deploy vào Tomcat server bằng cách sử dụng Maven và Gradle commands cho packaging ứng dụng của bạn

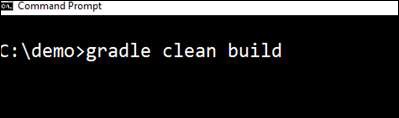
Với Maven , sử dụng command mvn package cho packaging ứng dụng của bạn , sau

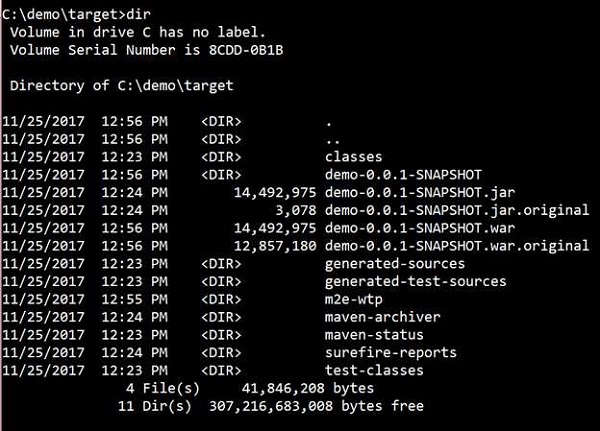
đó , WAR file sẽ được tạo và bạn có thể tìm thấy nó trong thư mục chỉ định





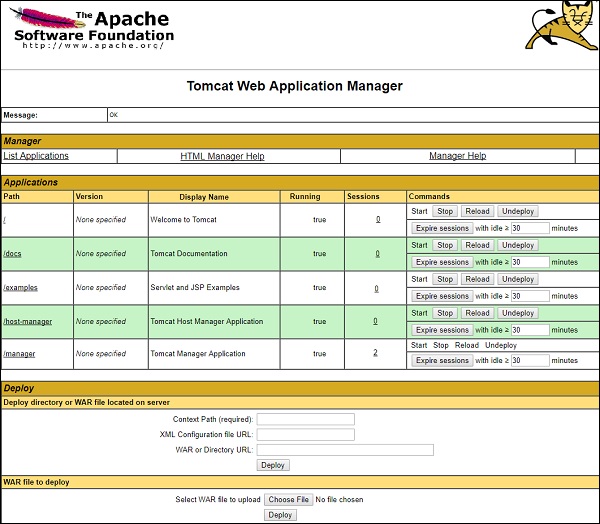
Với Gradle , sử dụng command gradle clean build cho packaging ứng dụng của bạn , sau đó WAR file sẽ được tạo và bạn có thể tìm nó trong thư mục build/libs





## Deploy into Tomcat

Bây giờ , chạy Tomcat Server và deploy WAR file trong thư mục webapps





Sau khi triển khai thành công , truy cập url [http://localhost:8080/<file](http://localhost:8080/%3cfile)-name>



Chapter 5.

Build Systems

Trong Spring Boot , lựa chọn hệ thống xây dựng là một công việc quan trọng , chúng ta nên lựa chọn Maven hoặc Gradle vì chúng hỗ trợ cung cấp trình quản lý dependency khá tốt , Spring không hỗ trợ tốt cho hệ thống xây dựng khác.

## Dependency Management

Spring Boot team cung cấp 1 danh sách các dependencies để hỗ trợ cho mỗi phiên bản Spring Boot phát hành , Bạn không cần cung cấp phiên bản cho các dependencies trong file cấu hình xâu dựng , Spring Boot tự động cấu hình các phiên bản dependencies dựa trên phiên bản phát hành. Ghi nhớ rằng , khi bạn cập nhật phiên bản Spring Boot , các dependencies sẽ tự động cập nhật theo.

Lưu ý – nếu bạn muốn chỉ định phiên bản cho dependency , bạn có thể chỉ định nó trong file cấu hình của bạn , tuy nhiên , Spring Boot team khuyến cáo rằng , điều đó là không cần thiết để chỉ định phiên bản cụ thể cho dependency.

## Maven Dependency

Dành cho việc cấu hình Maven , chúng ta nên kế thừa Spring Boot Starter Parent để quản lý Spring Boot Starters dependencie trong file pom.xml:

<parent>

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

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

<version>1.5.8.RELEASE</version>

</parent>

Chúng ta nên chỉ định phiên bản cho Spring Boot Parent Starter dependency . với các starter dependencies khác , chúng ta không cần chỉ định phiên bản Spring Boot.

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

## Gradle Dependency

Chúng ta có thể import Spring Boot Starters dependencies trực tiếp vào build.gradle file . Chúng ta không cần Spring Boot Start Parent dependency giống như Maven cho Gradle.

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

Tương tự , trong Gradle , chúng ta không cần chỉ định phiên bản Spring Boot cho dependencies , Spring Boot tự động cấu hình dependency dựa trên phiên bản của nó .

dependencies {

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

}

Chapter 6.

Code Structure

Spring Boot không có bất kì câu lệnh nào để làm việc , Tuy nhiên , có một số thực hành tốt nhất sau đây sẽ giúp đỡ chúng ta .

## Default package

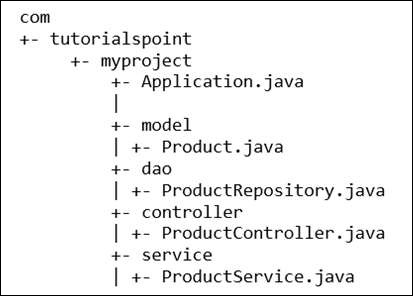
Một class không có bất kỳ khai báo package nào đc coi là một package mặc định .

Lưu ý , một khai báo package mặc định không được khuyến khích . Spring Boot sẽ gây ra các sự cố như trục trặc cấu hình tự động hoặc quét thành phần ,khi bạn sử dụng package mặc định.

Lưu ý – Java khuyên khai báo package chuẩn quy ước đặt tên là tên miền đảo ngược . Ví dụ: com.tutorialspoint.myproject

## Typical Layout

Bố cục điển hình của ứng dụng Spring Boot được hiển thị như sau:



Application.java file sẽ khai báo main method với @SpringBootApplication như sau :

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);}

}

Chapter 7.

Dependency Injection

Trong Spring Boot , chúng ta có thể sử dụng Spring Framework để định nghĩa các beans của chúng ta và injection dependency . @ComponentScan annotation được sử dụng để tìm các beans và injected tương ứng với @Autowired annotation.

Nếu bạn đã theo mô hình dựng sẵn của Spring Boot , không cần phải chỉ định bất kể đối số nào với @ComponentScan annotation. Toàn bộ component class sẽ tự động đăng ký với Spring Beans.

Các bước ví dụ sau sẽ cung cấp ý tưởng về tự động wiring đối tượng Rest Template và tạo bean cho nó:

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

Code cho việc auto wired đối tượng Rest Template và tạo đối đượng Bean trong main Spring Boot Application

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();

}

}

Chapter 8

Runners.

Application Runner và Command Line Runner interface sẽ thực thi code sau khi Spring Boot Application được khởi chạy. Bạn có thể sử dụng giao diện để thực hiện bất kể hành động nào ngay lập tức sau khi application khởi chạy.

## Application Runner

Application Runner là một interface sử dụng để thực thi code sau khi Spring Boot application khởi chạy. Ví sau sẽ chỉ bạn cách implement Application Runner interface trên main class:

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");

}

}

Và kết quả Hello World From Application Runner

“Hello World from Application Runner” được thực thi ngay sau khi Tomcat bắt đầu.

## Command Line Runner

Command Line Runner là 1 interface . Nó được sử dụng để thực thi code sau khi Srping Boot application khởi chạy. ví dụ sau đây sẽ chỉ bản cách implement Command Line Runner interface trong main class:

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");

}

}

Và kết quả :

Command Line Runner

Chapter 9

Application Properties

Application Properties hỗ trợ chúng ta làm việc trên các môi trường khác nhau , Bạn sẽ đc học cách cấu hình và chỉ định properties tới a ứng dụng Spring Boot.

## Command Line Properties

Spring Boot application chuyển đổi command line properties sang môi trường Spring Boot properties. Command line properties được ưu tiên hơn các nguồn property khác. Mặc định , Spring Boot sử dụng cổng 8080 để chạy Tomcat. Chúng ta sẽ học cách thay đổi cổng bằng cách sử dụng command line properties.

Bước 1 – Sau khi tạo 1 executable JAR file , chạy nó bằng cách sử dụng java –jar<JARFILE> trong command

Bước 2 –Sử dụng command được đưa ra trong ảnh chụp bên dưới để thay đổi số cổng cho ứng dụng Spring Boot bằng cách sử dụng command line properties.



Lưu ý – bạn có thể cung cấp nhiều hơn 1 application properties bằng cách sử dụng dấu phân cách “-“

## Properties File

Properties files đc sử dụng để giữ N số lượng properties trong 1 file để chạy ứng dụng trong các môi trường khác nhau. Trong Spring Boot , properties đc giữ trong application.properties file chứa trong đường dẫn của lớp .

Application.properties file lưu trữ tại thư mục src/main/resources . Code ví dụ cho application.properties file:

server.port = 9090

spring.application.name = demoservice

Lưu ý – dựa vào code trên , ứng dụng Spring Boot demoservice đc khởi tạo trên cổng 9090

## YAML File

Spring Boot hỗ trợ YAML dựa trên cấu hình properties để chạy ứng dụng. Thay vì application.properties , chúng ta có thể sử dụng file application.yml . file YAML này nên được giữ bên trong đường dẫn của lớp. Ví dụ : application.yml đưa ra sau đây:

spring:

application:

name: demoservice

server:

port: 9090

## Externalized Properties

Thay vì giữ properties file trong đường dẫn lớp , chúng ta có thể giữ trong vị trí khác hoặc đg dẫn khác . Trong khi chạy JAR file , chúng ta có thể chỉ định đường dẫn properties file. Bạn có thể sử dụng theo command để chỉ định vị trí của properties file trong khi chạy JAR

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

Externalized Properties

## Use of @Value Annotation

@Value annotation được sử dụng để đọc môi trường hoặc giá trị thuộc tính application trong Java code. Cú pháp để đọc giá trị thuộc tính như sau :

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

Theo dõi ví dụ sau để đưa ra cú pháp đọc giá trị thuộc tính spring.application.name trong biến Java bằng cách sử dụng @Value annotation.

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

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 thuộc tính không được tìm thấy trong lúc chạy ứng dụng , Spring Boot throws ra một IIlegal Argument exception “**Could not resolve placeholder 'spring.application.name' in value "${spring.application.name}"**.

Để giải quyết vấn đề chỗ này , chúng ta có thể set giá trị mặc định cho thuộc tính sử dụng cú pháp sau đây:

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

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

## Spring Boot Active Profile

Spring Boot hỗ trợ properties khác nhau dựa trên Spring active profile. Ví dụ: chúng ta có thể giữ 2 file riêng biệt để phát triển và sản xuất để chạy ứng dụng Spring Boot .

### **Spring active profile in application.properties**

Chúng ta sẽ hiểu cách để có Spring active profile trong application.properties. Mặc định , application.properties sẽ được sử dụng để chạy ứng dụng Spring Boot. Nếu bạn muốn sử dụng profile dựa trên properties, chúng ta có thể giữ file properties riêng biệt cho mỗi profile dưới đây:

**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 file JAR , chúng ta cần chỉ định spring active profile dựa trên mỗi properties file. Mặc định , Spring Boot application sử dụng application.properties file. Command để cài đặt spring active profile là :



Bạn có thể xem tên active profile trên console log như sau:

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

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: dev

Bây giờ , Tomcat chạy xong trên công 9090:

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 cài đặt Production active profile như sau:



Bạn có thể nhìn tên actiove profile trên console log :

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

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: prod

Và Tomcat khởi chạy trên cổng 4431

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 for application.yml**

chúng ta hiểu cách giữ Spring active profile cho application.yml. Chúng ta có thể giữ Spring active profile properties trong 1 file application.yml . Không cần sử dụng file riêng biệt như application.properties.

Hãy xem 1 ví dụ code để giữ Spring active profiles trong application.yml file. Lưu ý rằng , dấu phân cách (--) được sử dụng cho mỗi profile riêng biệt trong application.yml file:

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 command cài đặt development active profile :



Bạn có thể nhìn tên active profile trên console log :

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

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: dev

Bây giờ , Tomcat bắt đầu trên cổng 9090 :

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

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

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

Command để set Production active profile :



Bạn có thể thấy tên active profile trên console log

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

main] com.tutorialspoint.demo.DemoApplication :

The following profiles are active: prod

Tomcat sẽ bắt đầu tại cổng 4431:

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

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

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

Chapter 10.

Logging

Spring Boot sử dụng Apache Commons logging cho toàn bộ logging nội bộ. Cấu hình mặc định của Spring Boot cung cấp hỗ trợ sử dụng Java Util Logging , Log4j2 , và Logback. Sử dụng chúng , chúng ta có thể cấu hình console logging cũng như file logging.

Nếu bạn đang sử dụng Spring Boot Starters , Logback sẽ cung cấp hỗ trợ cực tốt cho logging. Ngoài ra , Logback cũng cung cấp một hỗ trợ cực tốt cho Common Logging , Util Logging , Log4j , SLF4J.

## Log Format

Mặc định định dạng Spring Boot Log :



Chúng cung cấp thông tin như sau:

* Ngày và Thời gian của log
* Cấp độ Log như INFO , ERROR , WARN
* Process ID (ID tiến trình)
* Dấu phân cách (---) cung cấp thông tin:
* Thread name : tên luồng được đặt trong dấu []
* Logger Name : tên logger hiển thị tên Source class
* Log message : tin nhắn log

## Console Log Output

Tin nhắn log mặc định sẽ in trên cửa sổ console. Mặc định , “INFO” , “ERROR” , “WARN” tin nhắn log sẽ in trong file log .

Nếu bạn có kích hoạt cấp độ log debug , thêm cờ debug trên việc khởi chạy ứng dụng của bạn sử dụng command :

java –jar demo.jar --debug

bạn cũng có thể thêm chế độ debug vào application.properties file

debug = true

## File Log Output

Mặc định , toàn bộ logs sẽ in trên cửa sổ console không phải file. Nếu bạn muốn in log trong a file , bạn cần cài đặt thuộc tính logging.file hoặc logging.path trong application.properties file.

Bạn có thể chỉ định đường dẫn log file sử dụng thuộc tính sau : (Lưu ý rằng tên log file là spring.log)

logging.path = /var/tmp/

Bạn có thể chỉ định mỗi tên log file sử dụng thuộc tính sau:

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

Lưu ý: file sẽ tự động thay phiên nhau sau khi đạt kích thước 10MB

## Log Levels

Spring Boot hỗ trợ toàn bộ cấp độ logger như là “TRACE” , “DEBUG” ,”INFO” , “WARN” , “ERROR”, “FATAL” , “OFF”. Bạn có thể định nghĩa Root logger trong application.properties file như sau:

logging.level.root = WARN

Logback không hỗ trợ cấp độ log “FATAL” . Nó sẽ ánh xạ tới “ERROR”

## Configure Logback

Logback hỗ trợ XML dựa trên cấu hình để xử lý cấu hình Spring Boot Log . Cấu hình Logging chi tiết được cấu hình trong file logback.xml. file logback.xml nên được đặt trong đường dẫn lớp.

Bạn có thể cấu hình cấp độ log “ROOT” trong logback.xml file sử dụng code dưới đây:

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

<configuration>

<root level = "INFO">

</root>

</configuration>

Bạn có thể cấu hình giao diện console (Console Appender) trong Logback.xml file dưới đây:

<?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>

Bạn có thể cấu hình giao diện file trong logback.xml file sử dụng code dưới đây. Chú ý rằng : bạn nên chỉ định đường dẫn file Log bên trong giao diện file (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>

Bạn có thể định nghĩa quy ước Log (Log pattern) trong logback.xml file sử dụng code dưới đây. Bạn cũng có thể định nghĩa cài đặt hỗ trợ quy ước log bên trong giao diện hoặc file giao diện log sử dụng code dưới đây:

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

Code hoàn thành logback.xml file dưới đây: bạn phải đặt nó trong đường dẫn lớp.

<?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>

Code đưa ra dưới đây sẽ hướng dẫn cách thêm slf4j logger trong Spring Boot main class file :

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);

}

}

Output bạn có thể thấy trong cửa sổ console :

Logger Console Window

Out put bạn có thể nhìn trong file log là :

Log Output

Chapter 11.

Building RESTFul Web Services

Spring Boot cung cấp 1 hỗ trợ rất tốt cho việc xây dựng RESTful Web Services cho ứng dụng doanh nghiệp. Sau đây sẽ giải thích chi tiết về xây dựng RESTful Web Services sử dụng Spring Boot

Lưu ý – Việc xây dựng một RESTful Web Services , chúng ta cần phải thêm Spring Boot Starter Web dependency vào file cấu hình để build.

Nếu bạn là một người sử dụng Maven , thực hiện code sau đây để thêm dependency vào file pom.xml:

<dependency>

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

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

</dependency>

Nếu bạn là một ng sử dụng Gradle , thực hiện code sau đây để thêm dependency vào file build.gradle

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

code hoàn thành việc build file cấu hình 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>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Code hoàn thành việc build cấu hình file Gradle Build – build.gradle:

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

dependencies {

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

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

}

Trước khi bạn tiến hành xây dựng một RESTful web service , lời khuyên bạn phải có kiến thức về các annotations dưới đây:

## Rest Controller

@RestController annotation được sử dụng để định nghĩa RESTful web service.

Nó thực hiện JSON , XML và tự tạo phản hồi (response)

Cú pháp sau đây :

@RestController

public class ProductServiceController {

}

Request Mapping

@RequestMapping annotation được sử dụng để định nghĩa Request URL chấp nhận truy cập REST Endpoints . Chúng ta có thể định nghĩa Request method để consume và produce đối tượng. Request method mặc định là GET.

@RequestMapping(value = "/products")

public ResponseEntity<Object> getProducts() { }

Request Body

@RequestBody annotation được sử dụng để định nghĩa kiểu content của request body:

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

}

Path Variable

@PathVariable annotation được sử dụng để định nghĩa bất kì hoặc một request URL động.

Path variable trong request URL được định nghĩa trong dấu ngoặc {}

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

}

Request Parameter

@RequestParam annotation được sử dụng để đọc tham số request từ Request URL. Mặc định , nó là 1 tham số bắt buộc. Chúng ta cũng có thể cài đặt giá trị mặc định cho tham số request dưới đây:

public ResponseEntity<Object> getProduct(

@RequestParam(value = "name", required = false, defaultValue = "honey") String name) {

}

GET API

HTTP request method mặc định là GET . method không bắt bất kì một Request Body nào . Bạn có thể gửi tham số request và path variables để định nghĩa URL bất kì hoặc URL động

Code đơn giản để định nghĩa HTTP GET request method được viết dưới đây. Trong ví dụ này , Chúng ta đã sử dụng HashMap để lưu trữ Product . Lưu ý rằng Chúng ta sử dụng 1 POJO class như là một Product để lưu trữ.

Request URL là /products và nó sẽ trả về một list các products from HashMap repository. Controller class file được đưa ra dưới đây có chứa GET method REST Endpoint.

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.RequestMapping;

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

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);

}

}

## POST API

HTTP POST request được sử dụng để tạo một resource. Method chứa Request Body. Chúng ta có thể gửi tham số request và path variables để định nghĩa URL bất kì hoặc URL động.

Theo dõi ví dụ sau đây , code đơn giản để định nghĩa HTTP POST request method. Trong ví dụ này , chúng ta sử dựng HashMap để lưu trữ Product, nơi product là một POJO class

Request URL là /products và nó sẽ trả về String sau khi lưu trữ product vào HashMap repository

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.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;

@RestController

public class ProductServiceController {

private static Map<String, Product> productRepo = new HashMap<>();

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

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

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

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

}

}

PUT API

HTTP PUT request được sử dụng để cập nhật nguồn đã tồn tại. Method này chứa một Request Body. Chúng ta có thể gửi tham số request và path variables để định nghĩa URL bất kì or URL động.

Ví dụ dưới đây sẽ chỉ cách để định nghĩa HTTP PUT request method.

Trong ví dụ này , chúng ta sử dụng HashMap để cập nhật Product đã tồn tại.

Request URL là /products/{id} sẽ trả về String sau khi product trong một HashMap repository được thay đổi.

Lưu ý rằng , chúng ta sử dụng Path variable {id} để định nghĩa products ID cần được update.

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.model.Product;

@RestController

public class ProductServiceController {

private static Map<String, Product> productRepo = new HashMap<>();

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

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

productRepo.remove(id);

product.setId(id);

productRepo.put(id, product);

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

}

}

DELETE API

HTTP Delete request được sử dụng để xóa nguồn đã tồn tại. Phương thức này không chứa bất kể Request Body nào. Chúng ta có thể gửi tham số request và path variables để định nghĩa URL bất kì hoặc URL động.

Ví dụ dưới đây sẽ chỉ cách định nghĩa HTTP DELETE request method. Trong ví dụ này , chúng ta sử dụng HashMap để xóa product đã tồn tại như là 1 POJO class.

Request URL là /products/{id} và sẽ trả về String nếu xóa thành công một product từ HashMap repository. Chúng ta sử dụng Path variable {id} để định nghĩa ID product cần được xóa.

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.model.Product;

@RestController

public class ProductServiceController {

private static Map<String, Product> productRepo = new HashMap<>();

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

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

productRepo.remove(id);

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

}

Code đầy đủ:

Spring Boot main application – 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);

}

}

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;

}

}

Rest Controller – 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.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.DELETE)

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

productRepo.remove(id);

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

}

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

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

productRepo.remove(id);

product.setId(id);

productRepo.put(id, product);

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

}

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

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

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

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

}

@RequestMapping(value = "/products")

public ResponseEntity<Object> getProduct() {

return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);

}

}

Bạn có thể tạo một executable JAR file và chạy Spring Boot application bằng cách sử dụng Commands của Maven hoặc Gradle dưới đây .

Dành cho Maven :

mvn clean install

Sau khi “BUILD SUCCESS” , bạn có thể tìm JAR file trong thư mục chỉ định.

Dành cho Gradle:

gradle clean build

Sau khi “BUILD SUCCESS” , bạn có thể tìm JAR file trong thư mục build/libs

Bạn có thể chạy JAR file bằng cách sử dụng command:

java –jar <JARFILE>

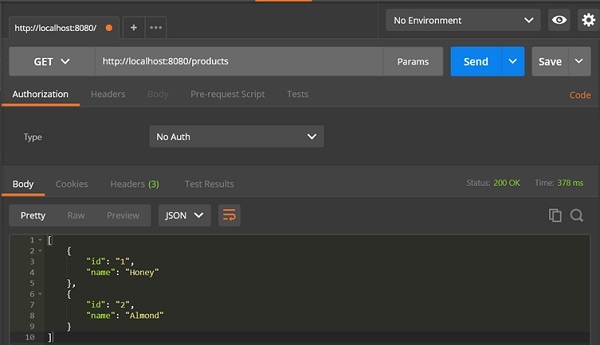
ứng dụng sẽ chạy trên cổng Tomcat 8080:

Started Application on Tomcat Port8080

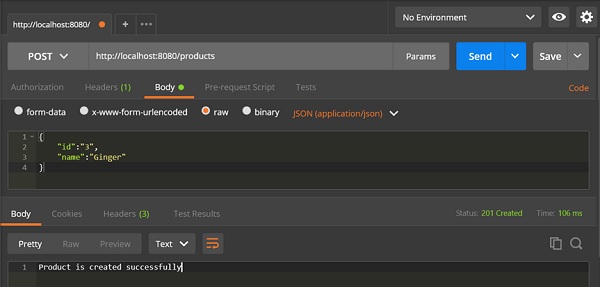
Và truy cập URL sử dụng ứng dụng POSTMAN và xem kết quả trả ra:

GET API URL : [**http://localhost:8080/products**](http://localhost:8080/products)

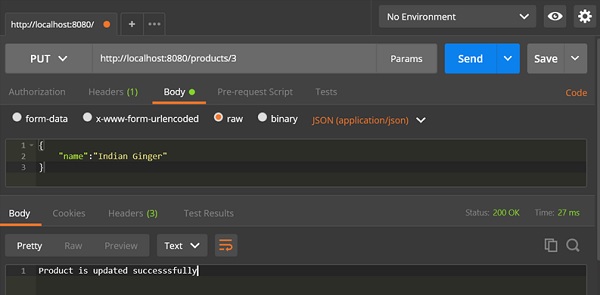
\



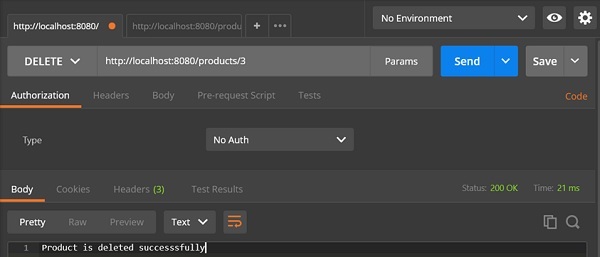
POST API URL: [**http://localhost:8080/products**](http://localhost:8080/products)



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



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



Chapter 12.

EXCEPTION HANDLING

Việc xử lý ngoại lệ exception và lỗi errors trong APIs và gửi phản hồi thích hợp tới client sẽ tốt cho các ứng dụng doanh nghiệp. Chúng ta sẽ học cách xử lý exceptions trong Spring Boot.

Trước khi tiến hành với việc xử lý exception , chúng ta sẽ cùng hiểu về các annotations sau đây.

Controller Advice

@ControllerAdvice là một annotation để xử lý ngoại lệ exceptions toàn cục.

Exception Handler

@ExceptiopnHandler là một annotation được sử dụng để xử lý chỉ định các ngoại lệ exceptions và gửi phản hồi bất kì tới client.

Bạn có thể sử dụng code dưới đây để tạo @ControllerAdvice class để xử lý exceptions cục bộ.

package com.tutorialspoint.demo.exception;

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

@ControllerAdvice

public class ProductExceptionController {

}

Định nghĩa một class kế thừa RuntimeException class.

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Bạn có thể định nghĩa @ExceptionHandler method để xử lý các ngoại lệ exceptions. Phương thức này nên được sử dụng cho việc viết một ControllerAdvice class.

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

}

Sử dụng code đưa ra sau đây để ném ra ngoại lệ từ API

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

public ResponseEntity<Object> updateProduct() {

throw new ProductNotfoundException();

}

Code đầy đủ để xử lý exception được đưa ra sau đây . Trong ví dụ này , chúng ta sử dụng PUT API để cập nhật product . Trong khi cập nhật product , nếu product không tìm thấy , và trả ra một phản hồi tin nhắn báo lỗi ví dụ “Product not found”

Lưu ý rằng , ProductNotFoundException exception class nên được kế thừa RuntimeException.

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Controller Advice class để xử lý exception cục bộ được đưa ra sau đây. Chúng ta có thể định nghĩa bất kì Exception Handler method trong class 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 được đưa ra sau đây để cập nhật product. Nếu product không tìm thấy , ngay sau đó nó sẽ ném một ngoại lệ ProductNotFoundException.

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 cho main Spring Boot application class.

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 cho 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 cho Maven build – pom.xml

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

<project xmlns = "http://maven.apache.org/POM/4.0.0"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.tutorialspoint</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>demo</name>

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

<parent>

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

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

<version>1.5.8.RELEASE</version>

<relativePath/>

</parent>

<properties>

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Cho Gradle Build

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()

}

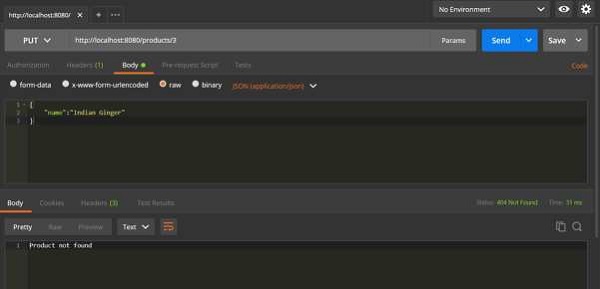
dependencies {

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

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

}

Kết quả :



Chapter 13.

INTERCEPTOR

Bạn có thể sử dụng Interceptor trong Spring Boot để thực hiện các hoạt động sau đây:

* Trước khi gửi request tới controller
* Trước khi gửi phản hồi tới client

Ví dụ: Bạn có thể sử dụng một interceptor để thêm request header trước khi gửi request tới controller và thêm response header trước khi gửi phản hồi tới client.

Để làm việc với interceptor , bạn cần tạo @Component class hỗ trợ nó và nó nên kế thừa HandlerInterceptor interface.

Các bước sau đây có 3 phương thức bạn nên biết trong khi làm việc với Interceptors:

* **preHandle()** method – được sử dụng để thực hiện các hoạt động trước khi gửi request tới controller . phương thức này nên trả về true để trả về phản hồi tới client.
* **postHandle()** method – được sử dụng để thực hiện các hoạt động trước khi gửi response tới client.
* **afterCompletion()** method – được sử dụng để thực hiện các hoạt động sau khi hoàn thành request và response

Code dưới đây :

@Component

public class ProductServiceInterceptor implements HandlerInterceptor {

@Override

public boolean preHandle(

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

return true;

}

@Override

public void postHandle(

HttpServletRequest request, HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {}

@Override

public void afterCompletion(HttpServletRequest request, HttpServletResponse response,

Object handler, Exception exception) throws Exception {}

}

Bạn sẽ phải đăng ký interceptor với InterceptorRegistry bằng cách sử dụng WebMvcConfigurerAdapter.

@Component

public class ProductServiceInterceptorAppConfig extends WebMvcConfigurerAdapter {

@Autowired

ProductServiceInterceptor productServiceInterceptor;

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(productServiceInterceptor);

}

}

Trong ví dụ dưới đây , chúng ta sẽ truy cập GET products API và đưa ra ouput như sau:

Code cho 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 Configuration class để đăng ký Interceptor tới 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 cho 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 cho main Spring Boot application 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);

}

}

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>

Gradle:

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

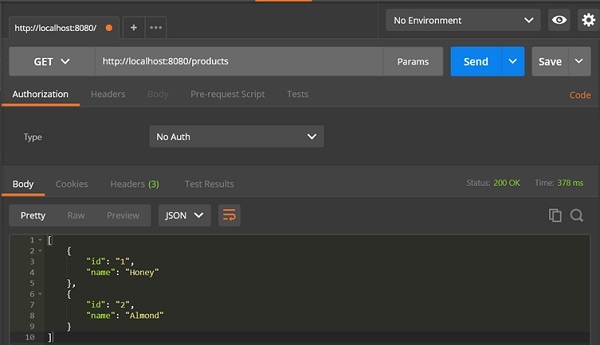
dependencies {

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

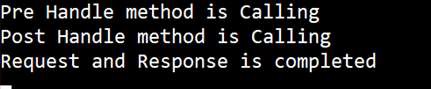
testCompile('org.springframework.boot:spring-boot-starter-test')

}

Kết quả :



Console log:



Chapter 14.

Servlet Filter

Một lớp lọc filter là một đối tượng được dùng để intercept HTTP requests và responses của ứng dụng. Bằng cách sử dụng filter , chúng ta có thể thực hiện 2 hành động sau tại 2 trường hợp sau :

* trước khi gửi request tới controller
* trước khi gửi response tới client

Code sau đây là ví dụ cho một Servlet Filter implementation class với @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 đây sẽ cho thấy code để đọc từ máy chủ khác và địa chỉ khác từ đối tượng ServletRequest trước khi gửi request tới controller.

Trong doFilter() method , chúng ta có thể thêm phần tử system.out.println để hiển thị máy chủ khác và địa chỉ khác.

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 Spring Boot main application class , chúng ta thêm REST endpoint đơn giản với kết quả trả về là 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 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>

Code cho Gradle:

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

dependencies {

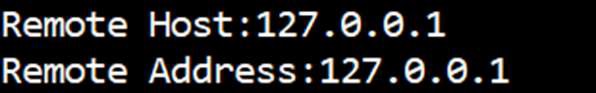
compile('org.springframework.boot:spring-boot-starter-web')

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

}

Kết quả: 

Console log:



Chapter 15.

Tomcat Port Number

Spring Boot cho phép bạn chạy cùng một ứng dụng nhiều lần trên cùng một số cổng khác nhau. Bạn sẽ học chi tiết về điều này. Lưu ý rằng: cổng mặc định là 8080.

Custom Port

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

server.port = 9090

trong application.yml :

server:

port: 9090

Random Port

Trong application.properties file , chúng ta có thể cài đặt ngẫu nhiên số port cho thuộc tính server.port

server.port = 0

trong application.yml :

server:

port: 0

Lưu ý : nếu số server.port là 0 trong khi khởi chạy Spring Boot application , Tomcat sử dụng số port ngẫu nhiên.

Chapter 16.

Rest Template

Rest Template được sử dụng để tạo applications với tích hợp RESTful Web Services. Bạn có thể sử dụng exchange() method để tích hợp web services cho toàn bộ HTTP methods . Code sau đây sẽ chỉ ra cách để tạo Bean cho Rest Template để auto wiring đối tượng Rest Template.

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

Tích hợp GET API bằng cách sử dụng RestTemplaete – exchange() method

Giả sử URL: <http://localhost:8080/products> trả về JSON và chúng ta sẽ tích hợp nó vào API response bằng cách sử dụng Rest Template như sau:

[

{

"id": "1",

"name": "Honey"

},

{

"id": "2",

"name": "Almond"

}

]

Bạn sẽ phải làm theo những điểm đưa ra sau đây để tích hợp API:

* Autowired đối tượng Rest Template
* Sử dụng HttpHeaders để cài đặt Request Headers
* Sử dụng HttpEntity để chuyển đổi request object
* Cung cấp URL , HttpMethod và Return type cho Exchange() method.

@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

Tích hợp POST API bằng cách sử dụng RestTemplate – exchange() method

Giả sử URL này <http://localhost:8080/products> trả về response sau đây , chúng ta sẽ tích hợp API response bằng cách sử dụng Rest Template.

Code cho Request Body:

{

"id":"3",

"name":"Ginger"

}

Code cho Response body:

Product is created successfully

Bạn sẽ phải làm theo các điều sau đây để tích hợp API :

* Autowired đối tượng Rest Template
* Sử dụng HttpHeaders để set Request Headers
* Sử dụng HttpEntity để chuyển đổi request object. Chúng ta có thể chuyển đổi Product object để gửi nó tới request body.
* Cung cấp URL , HttpMethod , và kiểu trả về cho exchange() method.
* @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

Tích hợp PUT API bằng cách sử dụng RestTemplate – exchange() method

Giả sử URL : <http://localhost:8080/products/3> trả về response dưới đây và chúng ta sẽ tích hợp API response bằng cách sử dụng Rest Template

Code cho Request Body:

{

"name":"Indian Ginger"

}

Code cho Response body:

Product is updated successfully

Bạn sẽ phải làm theo các điều sau đây để tích hợp API:

* Autowired Rest Template Object.
* Sử dụng HttpHeaders để set Request Headers
* Sử dụng HttpEntity để chuyển đổi request object. Chúng ta chuyển đổi Product object để gửi nó tới request body.
* Cung cấp URL , HttpMethod và Return type cho exchange() method.

@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

Tích hợp DELETE API bằng cách sử dụng RestTemplate – exchange() method

Giả sử URL: <http://localhost:8080/products/3> trả về response dưới đây và chúng ta sẽ tích hợp API response bằng cách sử dụng Rest Template:

Code Response body

Product is deleted successfully

Bạn sẽ phải làm theo điểm sau để tích hợp API:

* Autowired Rest Template Object
* Sử dụng HttpHeaders để set Request Headers
* Sử dụng HttpEntity để chuyển đổi request object
* Cung cấp URL , HttpMethod và kiểu trả về cho exchange() method

@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();

}

}

Rest Template Controller class hoàn chỉnh:

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();

}

}

Spring Application:

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

Maven :

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

<project xmlns = "http://maven.apache.org/POM/4.0.0"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.tutorialspoint</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>demo</name>

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

<parent>

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

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

<version>1.5.8.RELEASE</version>

<relativePath/>

</parent>

<properties>

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Gradle:

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 {

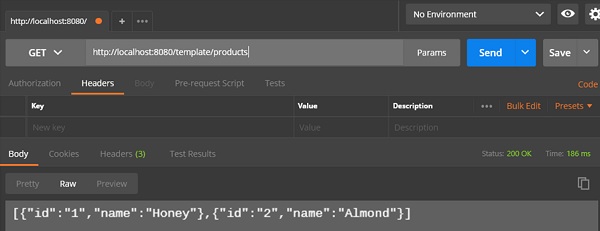
compile('org.springframework.boot:spring-boot-starter-web')

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

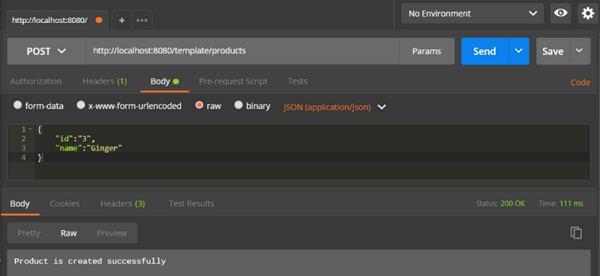
}

Kết quả :

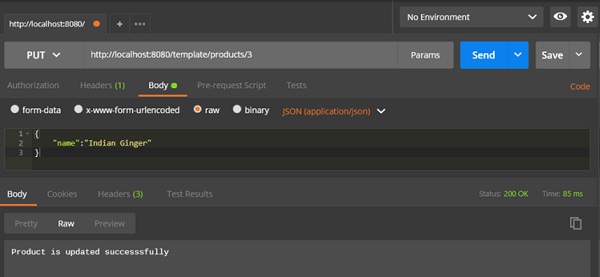
GET API:



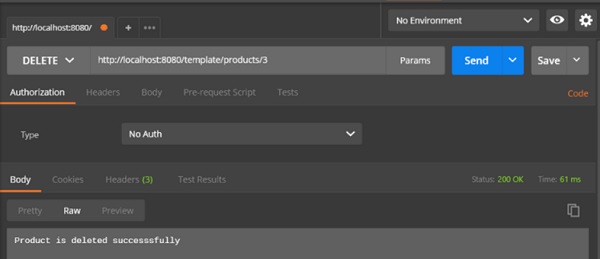
POST API:



PUT API:



DELETE API:



Chapter 17.

File Handling

Bạn sẽ học cách upload và download file bằng cách sử dụng web service.

File Upload

Dành cho việc uploading một file , bạn có thể sử MultipartFile như một Request Parameter và API này nên được tích hợp Multi-Part từ giá trị dữ liệu . Quan sát code dưới đây:

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

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

return null;

}

Code hoàn chỉnh với dí dụ 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

Dành cho download file , bạn nên sử dựng InputStreamResource cho việc download một file. Chúng ta cần set HttpHeader Content-Disposition trong response và cần chỉ định response Media Type của ứng dụng.

Lưu ý – trong ví dụ sau đây , file nên khả dụng 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 hoàn chỉnh:

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;

}

}

Main Spring Boot application:

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 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>

Code cho Gradle :

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

group = 'com.tutorialspoint'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

dependencies {

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

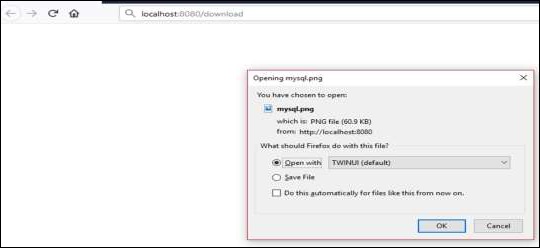
testCompile('org.springframework.boot:spring-boot-starter-test')

}

Kết quả:

[**http://localhost:8080/upload**](http://localhost:8080/upload)

[**http://localhost:8080/download**](http://localhost:8080/download)



Chapter 18.

Service Components

Service Components là class chứa các @Service annotation. Những class được sử dụng để viết business logic trong các tầng khác nhau , độc lập từ @RestController class . Logc cho việc khởi tạo một thành phần service class được chỉ ra dưới đây:

public interface ProductService {

}

Class được kế thừa interface với @Service annotation được chỉ ra dưới đây:

@Service

public class ProductServiceImpl implements ProductService {

}

Quan sát hướng dẫn này , chúng ta đang sử dụng Product Service API(s) để lưu trữ , truy xuất , cập nhật , xóa products. Chúng ta đã viết business logic trong @RestController class. Bây giờ , chúng ta sẽ đi đến business logic code từ controller tới thành phần service.

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();

}

Code dưới đây sẽ tạo một class implements ProductService interface với @Service annotation và viết business logic để lưu trữ , thêm mới , 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();

}

}

Code sau đây sẽ show Rest Controller class , @Autorwired ProductService interface và gọi method.

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

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) {

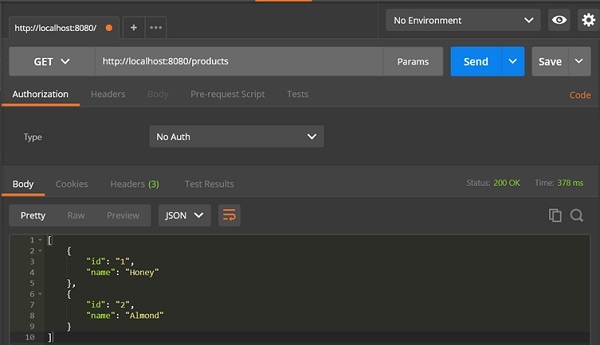
this.name = name;

}

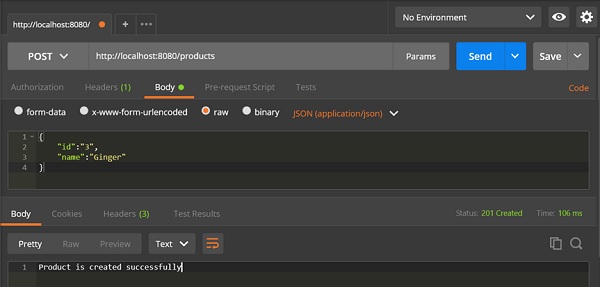
}

Kết quả:

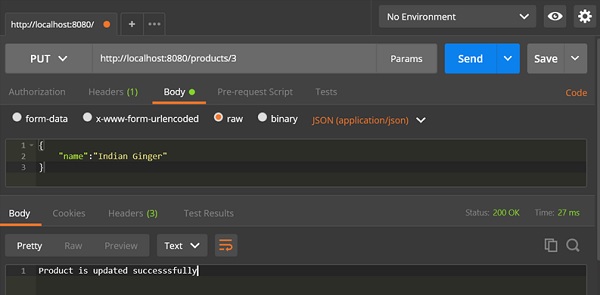
GET API:



POST API:



PUT API:



DELETE API:

