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.

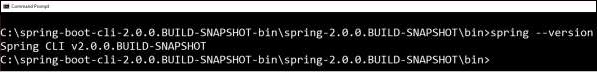
Phần này giải thích cho bạn các bước liên quan đến cài đặt thủ công Spring Boot CLI . Tìm hiểu thêm tại: https://docs.spring.io/springboot/ docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-springboot

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.

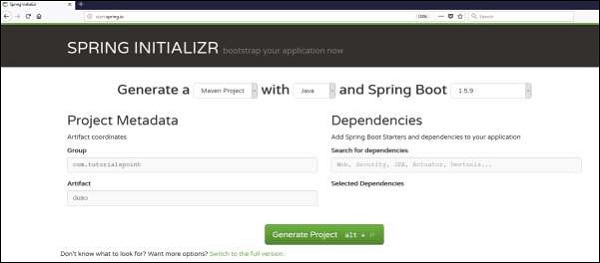
Bootstrapping

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";

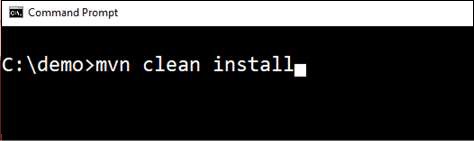
}

}

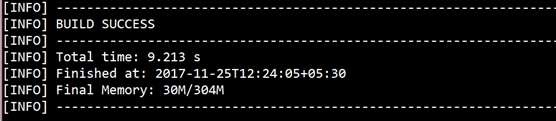
## Create an Executable JAR

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

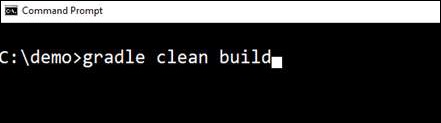
Sử dụng Maven command mvn clean install :



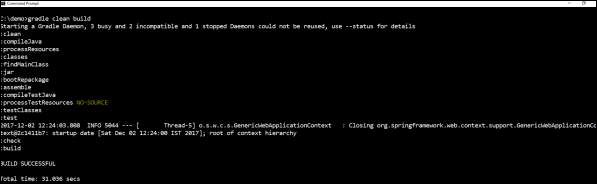
Sau khi chạy command , bạn có thể thấy tin nhắn BUILD SUCCESS tại command prompt như sau :



Sử dụng Gradle command gradle clean build như sau:



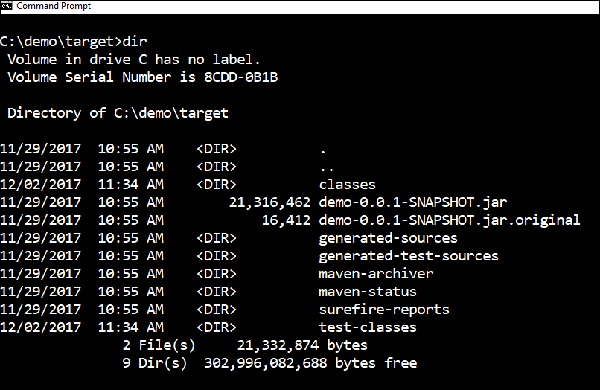
Sau khi chạy command , bạn có thể nhìn tin nhắn BUILD SUCCESSFUL trong command prompt như sau :



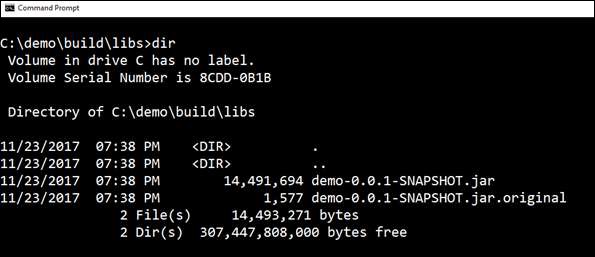
## Run Hello World with Java

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";

}

}

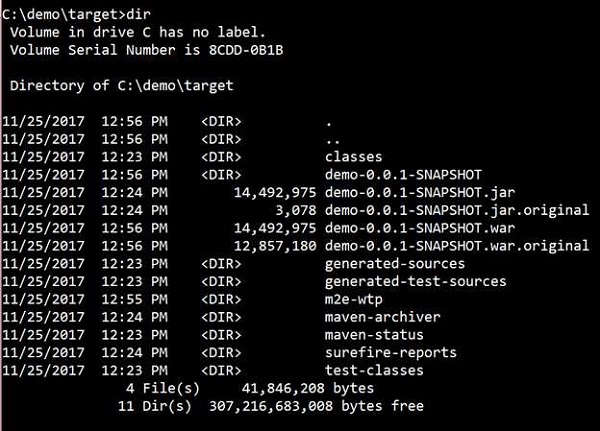
## Packaging your Application

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

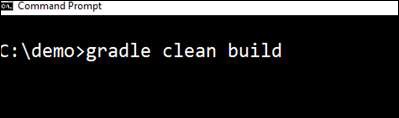
Với Maven , sử dụng command mvn package cho packaging ứng dụng của bạn , sau

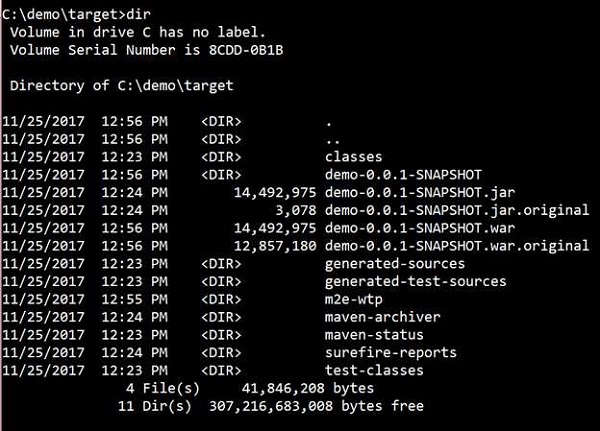
đó , WAR file sẽ được tạo và bạn có thể tìm thấy nó trong thư mục chỉ định





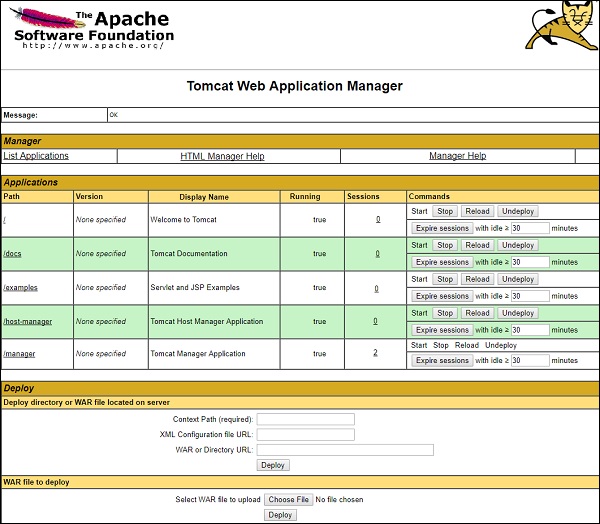
Với Gradle , sử dụng command gradle clean build cho packaging ứng dụng của bạn , sau đó WAR file sẽ được tạo và bạn có thể tìm nó trong thư mục build/libs





## 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

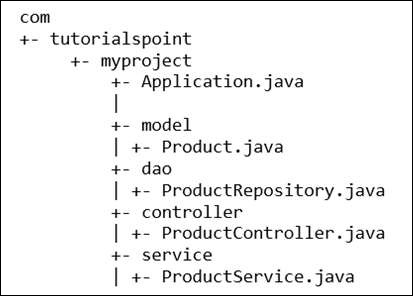
Một class không có bất kỳ khai báo package nào đc coi là một package mặc định .

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>

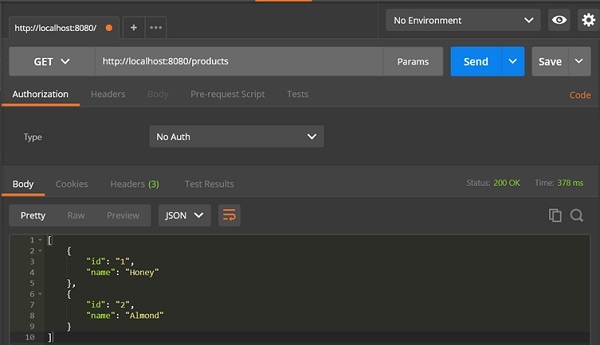
ứng dụng sẽ chạy trên cổng Tomcat 8080:

Started Application on Tomcat Port8080

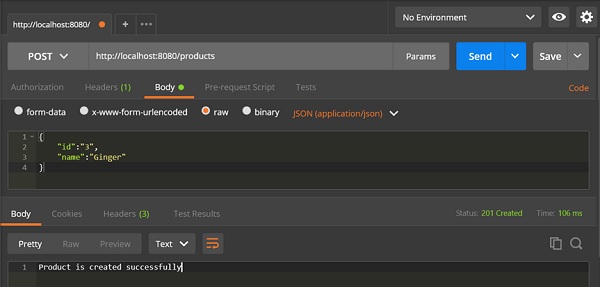
Và truy cập URL sử dụng ứng dụng POSTMAN và xem kết quả trả ra:

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

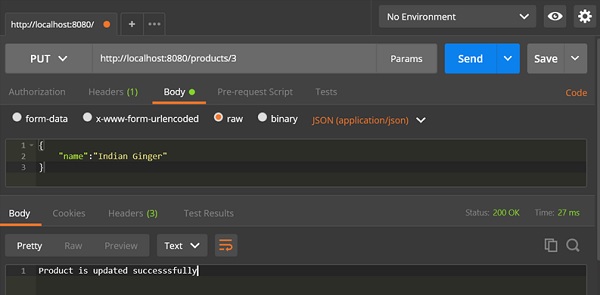
\



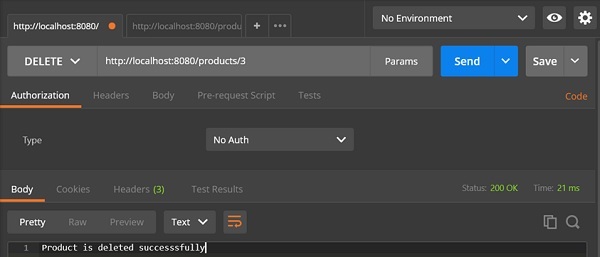
POST API URL: [**http://localhost:8080/products**](http://localhost:8080/products)



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()

}

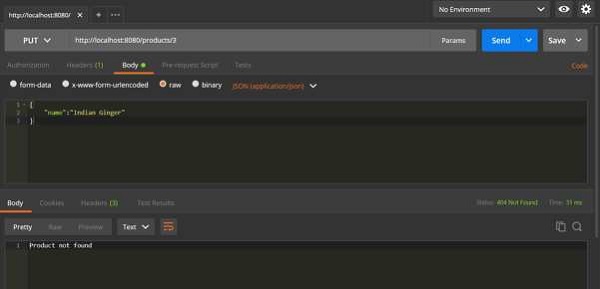
dependencies {

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()

}

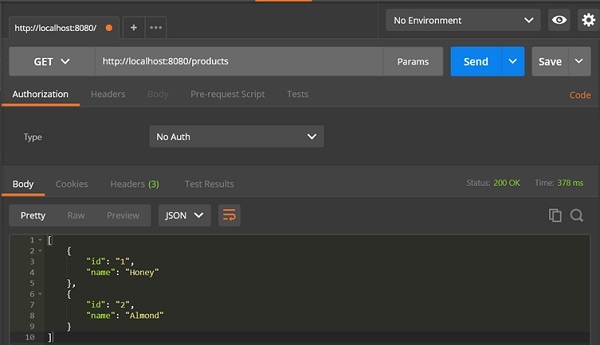
dependencies {

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

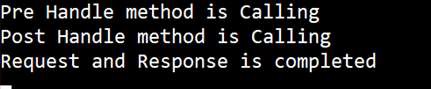
testCompile('org.springframework.boot:spring-boot-starter-test')

}

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 {

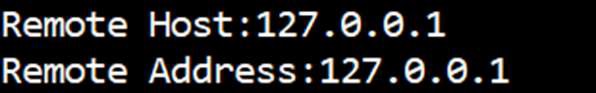
compile('org.springframework.boot:spring-boot-starter-web')

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 {

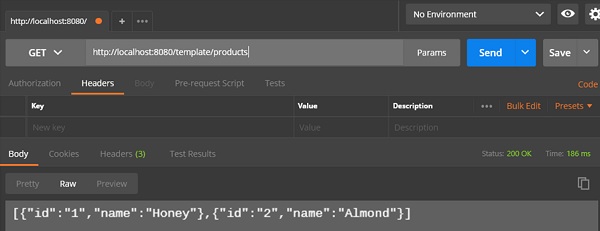
compile('org.springframework.boot:spring-boot-starter-web')

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

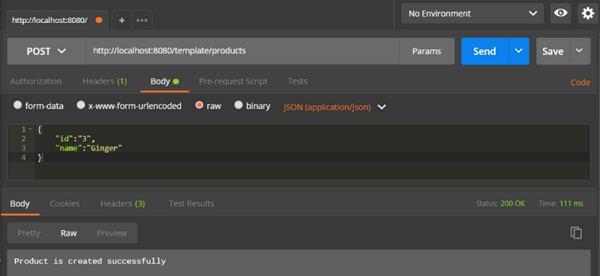
}

Kết quả :

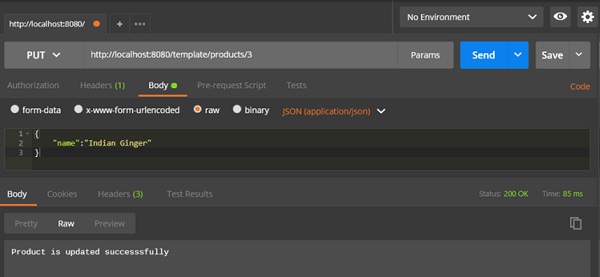
GET API:



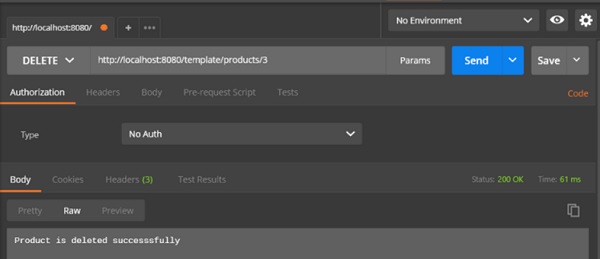
POST API:



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')

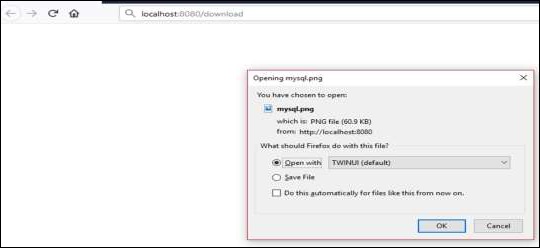
testCompile('org.springframework.boot:spring-boot-starter-test')

}

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) {

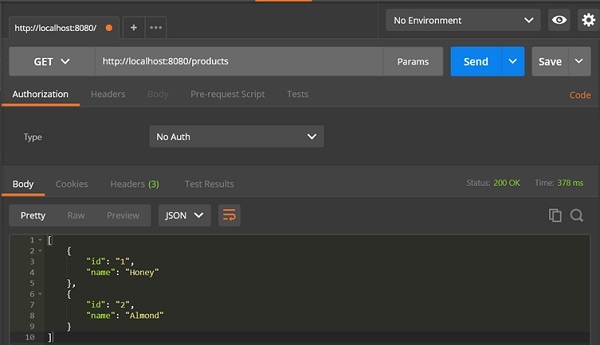
this.name = name;

}

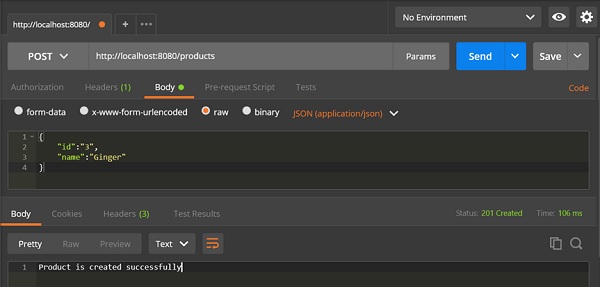
}

Kết quả:

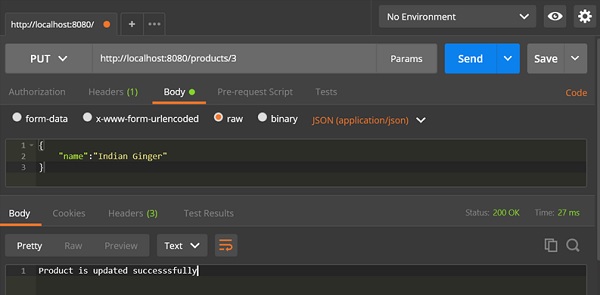
GET API:



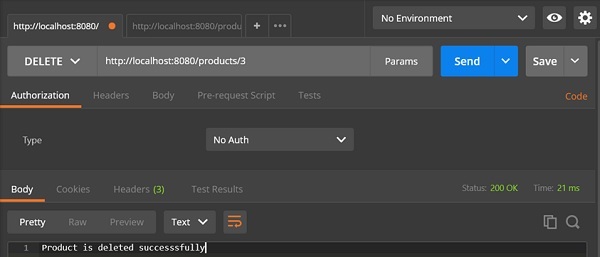
POST API:



PUT API:



DELETE API:



Chapter 19

Thymeleaf

Thymeleaf là một thư viện java được sử dụng để tạo một web application. Nó cung cấp một hỗ trợ tốt cho việc phục vụ XHTML/HTML5 trong web application.

Bạn sẽ học chi tiết về thymleaf.

Thymeleaf Templates

Thymleaf chuyển đổi file của bạn thành định dạng XML. Nó chứa 6 kiểu của templates dưới đây:

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

Toàn bộ templates , ngoại trừ Legacy HTML5 , đang đề cập định dạng valid XML . Legacy HTML5 cho chúng ta render thẻ HTML5 trong trang web bao gồm các thẻ không đóng.

Web Application

Bạn có thể sử dụng Thymleaf templates để tạo một web application trong Spring Boot. Bạn sẽ phải theo dõi các bước sau đây để tạo một web application trong Spring Boot bằng cách sử dụng Thymleaf.

Sử dụng code dưới đây để tạo một @Controller class để điều hướng request URL sang HTML:

package com.tutorialspoint.demo.controller;

import org.springframework.stereotype.Controller;

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

@Controller

public class WebController {

@RequestMapping(value = "/index")

public String index() {

return "index";

}

}

Trong ví dụ , request URL là /index và control để điều hướng tới index.html. Lưu ý rằng file index.html nên được chứa trong thư mục templates và toàn bộ JS và CSS files nên được chứa trong thư mục static trong đường dẫn lớp. trong ví dụ sau đây, chúng ta đã sử dụng CSS file để thay đổi màu sắc đoạn text.

Bạn có thể sử dụng đoạn code và tạo một CSS trong folder riêng css và tên file là styles.css:

h4 {

color: red;

}

Code cho file index.html:

<!DOCTYPE html>

<html>

<head>

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

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

<title>Spring Boot Application</title>

</head>

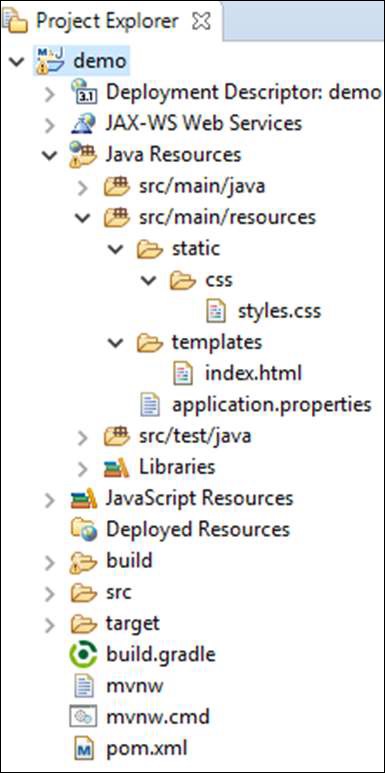
<body>

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

</body>

</html>

Cấu trúc project như sau :



Chúng ta cần thêm dependency Spring Boot Starter Thymleaf trong file cấu hình :

Maven :

<dependency>

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

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

</dependency>

Gradle:

compile group: 'org.springframework.boot', name: 'spring-boot-starter-thymeleaf'

kết quả:

<http://localhost:8080/index>



Chapter 20.

Consuming RESTful Web Services

Chúng ta sẽ bàn luận chi tiết về sử dụng RESTful Web Services bằng cách sử dụng Jquery AJAX.

Tạo một Spring Boot web application đơn giản và viết một controller class được sử dụng để điều hướng tới file HTML , sử dụng RESTful Web Services.

Chúng ta cần thêm Spring Boot starter Thymeleaf và Web dependency trong cấu hình:

Maven:

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

Gradle:

compile group: ‘org.springframework.boot’, name: ‘spring-boot-starter-thymeleaf’

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

code cho @Controller class:

@Controller

public class ViewController {

}

Bạn có thể định nghĩa Request URL methods để điều hướng tới HTML file như sau:

@RequestMapping(“/view-products”)

public String viewProducts() {

return “view-products”;

}

@RequestMapping(“/add-products”)

public String addProducts() {

return “add-products”;

}

API <http://localhost:9090/products> nên trả về JSON trong response:

[

{

"id": "1",

"name": "Honey"

},

{

"id": "2",

"name": "Almond"

}

]

Tạo một view-products.html chứa trong thư mục templates trong đường dẫn class:

Trong file HTML , chúng ta thêm thư viện Jquery và viết code để sử dụng RESTful web service khi tải trang:

<script>

$(document).ready(function(){

$.getJSON("http://localhost:9090/products", function(result){

$.each(result, function(key,value) {

$("#productsJson").append(value.id+" "+value.name+" ");

});

});

});

</script>

POST method với URL: <http://localhost:9090/products> nên chứa Request Body và Response body.

Code cho Request Body:

{

"id":"3",

"name":"Ginger"

}

Code cho Response body:

Product is created successfully

Tạo một file add-products.html chứa trong thư mục templates trong đường dẫn class:

Trong file HTML , chúng ta thêm thư viện Jquery và viết code để submits form tới RESTful web service khi click button:

<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script>

$(document).ready(function() {

$("button").click(function() {

var productmodel = {

id : "3",

name : "Ginger"

};

var requestJSON = JSON.stringify(productmodel);

$.ajax({

type : "POST",

url : "http://localhost:9090/products",

headers : {

"Content-Type" : "application/json"

},

data : requestJSON,

success : function(data) {

alert(data);

},

error : function(data) {

}

});

});

});

</script>

Code hoàn chỉnh:

Controller class:

package com.tutorialspoint.demo.controller;

import org.springframework.stereotype.Controller;

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

@Controller

public class ViewController {

@RequestMapping(“/view-products”)

public String viewProducts() {

return “view-products”;

}

@RequestMapping(“/add-products”)

public String addProducts() {

return “add-products”;

}

}

View-products.html:

<!DOCTYPE html>

<html>

<head>

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

<title>View Products</title>

<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script>

$(document).ready(function(){

$.getJSON("http://localhost:9090/products",function(result){

$.each(result, function(key,value) {

$("#productsJson").append(value.id+" "+value.name+" ");

});

});

});

</script>

</head>

<body>

<div id = "productsJson"> </div>

</body>

</html>

Add-product.html:

<!DOCTYPE html>

<html>

<head>

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

<title>Add Products</title>

<script src = "https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script>

$(document).ready(function() {

$("button").click(function() {

var productmodel = {

id : "3",

name : "Ginger"

};

var requestJSON = JSON.stringify(productmodel);

$.ajax({

type : "POST",

url : "http://localhost:9090/products",

headers : {

"Content-Type" : "application/json"

},

data : requestJSON,

success : function(data) {

alert(data);

},

error : function(data) {

}

});

});

});

</script>

</head>

<body>

<button>Click here to submit the form</button>

</body>

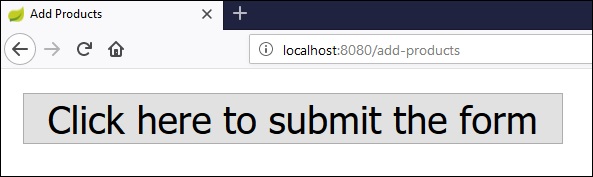
</html>

Kết quả:

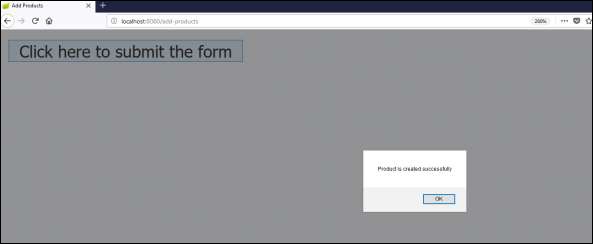
Localhost:8080/view-products



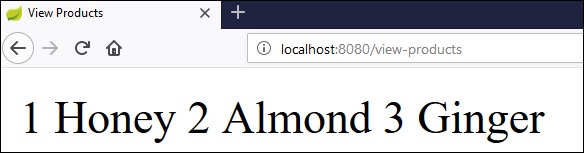
Localhost:8080/add-products



Click button:



Chạy lại url: localhost:8080/view-products:



Angular JS:

Để tích hợp API bằng cách sử dụng Angular JS , bạn có thể sử dụng ví dụ sau:

Sử dụng code sau đây để tạo Angular JS controller để tích hợp GET API – <http://localhost:9090/products>

angular.module('demo', [])

.controller('Hello', function($scope, $http) {

$http.get('http://localhost:9090/products').

then(function(response) {

$scope.products = response.data;

});

});

Sử dụng code dưới đây để tạo Angular JS controller để tích hợp POST API – <http://localhost:9090/products>

angular.module('demo', [])

.controller('Hello', function($scope, $http) {

$http.post('http://localhost:9090/products',data).

then(function(response) {

console.log("Product created successfully");

});

});

Lưu ý rằng: dữ liệu POST method đại diện Request body trong định dạng JSON để tạo một product.

Chapter 21

CORS Support.

Cross-Origin Resource Sharing (CORS) là một khái niệm bảo mật (security) cho phép hạn chế nguồn triển khai trên giao diện web. Nó ngăn chặn JavaScript code thực hiện hoặc tích hợp request chống lại các nguồn khác nhau.

Ví dụ : ứng dụng web của bạn đang chạy trên cổng 8080 và bằng việc sử dụng JavaScript , bạn đang cố tích hợp RESTful web services từ cổng 9090. Trong tình huống như vậy , bạn sẽ đối mặt với vấn đề bảo mật Cross-Origin Resource Sharing trên giao diện web.

Hai yêu cầu cần được xử lý với vấn đề này :

* RESTful web services nên đc hỗ trợ CORS.
* RESTful web service application nên được cho phép chấp nhận API từ cổng 8080.

Chúng ta sẽ học chi tiết về làm thế nào để kích hoạt Cross-Orgin Request cho một RESTful Web Service application.

Enable CORS in Controller Method

Chúng ta cần set orrgins cho RESTful web service bằng cách sử dụng @CrossOrigin annotation cho controller method. @CrossOrigin annotation hỗ trợ chỉ định REST API , và không cho toàn bộ ứng dụng.

@RequestMapping(value = "/products")

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

public ResponseEntity<Object> getProduct() {

return null;

}

GLOBAL CORS Configuration

Chúng ta cần định nghĩa cấu hình @Bean để set cấu hình CORS hỗ trợ toàn bộ cho ứng dụng Spring Boot của bạn.

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

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

}

};

}

Code để set cấu hình CORS toàn bộ trong main Spring Boot application :

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

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

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

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

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

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

}

};

}

}

Giờ đây , bạn có thể tạo một Spring Boot web application chạy trên cổng 8080 và RESTful web service application chạy trên cổng 9090. Để biết thêm chi tiết về triển khai RESTful Web Service , bạn có thể chuyển đến chuyên mục Consuming RESTful Web Services .

Chapter 22.

Internationalization

Internationalization là một tiến trình làm cho ứng dụng của bạn thích nghi với các ngôn ngữ và khu vực khác nhau mà không cần công nghệ thay đổi trên source code. Nói cách khác , internationalization là một sự linh hoạt của Localization.

Chúng ta sẽ học chi tiết về cách để triển khai internationalization trong Spring Boot.

Dependencies

Chúng ta cần spring boot starter web và spring boot starter thymelead dependency để phát triển một web application trong Spring Boot

Maven

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

Gradle

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

compile group: 'org.springframework.boot', name: 'spring-boot-starter-thymeleaf'

LocaleResolver

Chúng ta cần xác định Locale mặc định của ứng dụng , chúng ta cần thêm LocaleResolver bean trong Spring Boot application.

@Bean

public LocaleResolver localeResolver() {

SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();

sessionLocaleResolver.setDefaultLocale(Locale.US);

return sessionLocaleResolver;

}

LocaleChangeInterceptor

localeChangeInterceptor được sử dụng để thay đổi Locale mới dựa trên giá trị của tham số ngôn ngữ đã thêm vào một request

@Bean

public LocaleChangeInterceptor localeChangeInterceptor() {

LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();

localeChangeInterceptor.setParamName("language");

return localeChangeInterceptor;

}

Để có được tính năng này , chúng ta cần thêm LocaleChangeInterceptor vào interceptor registry của application. Cấu hình class nên kế thừa WebMvcConfigurerAdapter class và override phương thức addInterceptors()

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(localeChangeInterceptor());

}

Messages Sources

Spring Boot application mặc định sẽ lấy message sources từ thư mục src/main/resources chứa trong đường dẫn lớp.Mặc định tên file locale message nên là message.properties và nhiều files cho mỗi locale nên đặt tên theo mẫu

Messages\_XX.properties , XX là mã locale đại diện.

Toàn bộ message properties nên được sử dụng theo cặp key , values . Nếu bất kì properties không được tìm thấy trên locale, application sẽ sử dụng property mặc định từ messages.properties file.

Message.properties mặc định là:

welcome.text=Hi Welcome to Everyone

tiếng Pháp messages\_fr.properties:

welcome.text=Salut Bienvenue à tous

lưu ý – messages source file nên được lưu trữ theo định dạng UTF-8

HTML file

Trong file HTML , sử dụng syntax #{key} để hiển thị messages từ properties file.

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

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>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

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')

compile group: 'org.springframework.boot', name: 'spring-boot-starter-thymeleaf'

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

}

Controller:

package com.tutorialspoint.demo.controller;

import org.springframework.stereotype.Controller;

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

@Controller

public class ViewController {

@RequestMapping("/locale")

public String locale() {

return "locale";

}

}

Config internationalization:

package com.tutorialspoint.demo;

import java.util.Locale;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.LocaleResolver;

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

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

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

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

@Configuration

public class Internationalization extends WebMvcConfigurerAdapter {

@Bean

public LocaleResolver localeResolver() {

SessionLocaleResolver sessionLocaleResolver = new SessionLocaleResolver();

sessionLocaleResolver.setDefaultLocale(Locale.US);

return sessionLocaleResolver;

}

@Bean

public LocaleChangeInterceptor localeChangeInterceptor() {

LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();

localeChangeInterceptor.setParamName("language");

return localeChangeInterceptor;

}

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(localeChangeInterceptor());

}

}

Locale.html:

<!DOCTYPE html>

<html>

<head>

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

<title>Internationalization</title>

</head>

<body>

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

</body>

</html>

Kết quả:

<http://localhost:8080/locale>



http://localhost:8080/locale?language=fr



Chapter 23.

Scheduling

Scheduling là một tiến trình của việc thi hành các tasks trong một khoảng thời gian cụ thể. Spring Boot cung cấp một hỗ trợ tốt để viết một scheduler trên ứng dụng Spring.

Java Cron Expression

Java Cron expressions được sử dụng để cấu hình các trường hợp của CronTrigger

Một subclass của Trigger. Thông tin chi tiết về java cron expression bạn có thể tìm thấy link sau:

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

@EnableScheduling annotation được sử dụng để kick hoạt scheduler cho ứng dụng của bạn. Annotation này nên được thêm trong main Spring Boot application class.

@SpringBootApplication

@EnableScheduling

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

@Scheduled annotation được sử dụng để trigger scheduler cho 1 khoảng thời gian cụ thể.

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

public void cronJobSch() throws Exception {

}

theo dõi code đơn giản sau đây sẽ chỉ cách thi hành task với mỗi phút bắt đầu từ 9:00 AM và kết thúc vào 9:59 AM hàng ngày.

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

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

public void cronJobSch() {

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

Date now = new Date();

String strDate = sdf.format(now);

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

}

}

Kết quả: Application bắt đầu vào 09:03:23 và cứ mỗi một phút kể từ thời gian đó , cron job scheduler task lại được thi hành.

Fixed Rate:

Fixed Rate scheduler được sử dụng để thi hành các task tại thời gian cụ thể.

Nó không đợi việc hoàn thành task trước đó.

Giá trị nên là milliseconds. Code ví dụ sau đây:

@Scheduled(fixedRate = 1000)

public void fixedRateSch() {

}

Một ví dụ code cho việc thi hành một task với mỗi giây từ khi application khởi tạo:

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

@Scheduled(fixedRate = 1000)

public void fixedRateSch() {

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

Date now = new Date();

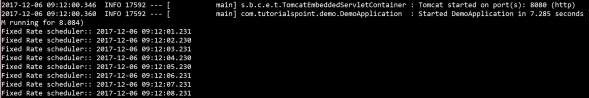
String strDate = sdf.format(now);

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

}

}

Kết quả: application đã bắt đầu vào 9:12:00 và mỗi giây fixedRate scheduler task được thực thi.



Fixed Delay

Fixed Delay scheduler được sử dụng để thực thi các tasks tại thời gian cụ thể. Nó nên đợi cho việc hoàn thành các task trước đó. Giá trị nên là milliseconds. Ví dụ Code sau đây:

@Scheduled(fixedDelay = 1000, initialDelay = 1000)

public void fixedDelaySch() {

}

initialDelay là thời gian sau đó task sẽ được thực thi lần đầu sau giá trị intial delay.

Một ví dụ việc thực thi task với mỗi giây sau 3 giây từ lúc application khởi chạy và được hoàn thành sau đây:

package com.tutorialspoint.demo.scheduler;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Scheduler {

@Scheduled(fixedDelay = 1000, initialDelay = 3000)

public void fixedDelaySch() {

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

Date now = new Date();

String strDate = sdf.format(now);

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

}

}

Chapter 25

Enabling HTTPS

Mặc định , Spring Boot application sử dụng HTTP cổng 8080 khi ứng dụng bắt đầu chạy.

Started Application on Tomcat port_8080

Bạn cần theo dõi các bước sau để cấu hình HTTPS và cổng 443 trong Spring Boot application:

* Lấy chứng chỉ SSL – tạo một chứng chỉ tự kí hoặc lấy chứng chỉ từ cơ quan cấp chứng chỉ
* Kích hoạt HTTPS và cổng 443

Self-Signed Certificate

Để tạo một chứng chỉ tự kí , môi trường Java Run Time đi kèm với công cụ tiện ích quản lý chứng chỉ.

Công cụ tiện ích này được sử dụng để tạo một chúng chỉ tự kí. Code đc show dưới đây:

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

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]:

What is the name of your organizational unit?

[Unknown]:

What is the name of your organization?

[Unknown]:

What is the name of your City or Locality?

[Unknown]:

What is the name of your State or Province?

[Unknown]:

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

[Unknown]:

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

[no]: yes

Code sẽ generate một file lưu trữ key PKCS12 hay là keystore.p12 và tên bí danh chứng chỉ là tomcat.

Configure HTTPS

Chúng ta cần cung cấp cổng server là 443 , đường dẫn file key-store , password key-store , kiểu key-store , và bí danh key vào application.properties. Quan sat code đưa ra sau đây:

server.port: 443

server.ssl.key-store: keystore.p12

server.ssl.key-store-password: springboot

server.ssl.keyStoreType: PKCS12

server.ssl.keyAlias: tomcat

bạn có thể sử dụng code sau đây nếu bạn đang sử dụng YAML properties dưới dạng application.yml:

server:

port: 443

ssl:

key-store: keystore.p12

key-store-password: springboot

keyStoreType: PKCS12

keyAlias: tomcat

kết quả:

Started Application on Tomcat Port 443

Chapter 26

Eureka Server

Eureka Server là một application lưu trữ thông tin về toàn cliend-service applications. Mỗi một Micro service sẽ đăng kí vào Eureka server và Eureka server biết toàn bộ client applications đang chạy trên mỗi cổng và địa chỉ IP.

Eureka Server được biết đến như là Discovery Server.

Chúng ta sẽ học chi tiết về cách để build một Eureka server.

Building a Eureka Server

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

@EnableEurekaServer annotation được sử dụng để làm Spring Boot application hoạt động như một Eureka Server.

Code cho main Spring Boot application:

package com.tutorialspoint.eurekaserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableEurekaServer

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

}

}

Add dependency được thêm vào cấu hình :

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-eureka-server')

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>eurekaserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eurekaserver</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Gradle:

buildscript {

ext {

springBootVersion = '1.5.9.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()

}

ext {

springCloudVersion = 'Edgware.RELEASE'

}

dependencies {

compile('org.springframework.cloud:spring-cloud-starter-eureka-server')

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

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"

}

}

Mặc định , Eureka Server đăng ký vào discovery. Bạn nên thêm cấu hình dưới đây vào file application.properties hoặc file application.yml:

Application.properties:

eureka.client.registerWithEureka = false

eureka.client.fetchRegistry = false

server.port = 8761

application.yml:

eureka:

client:

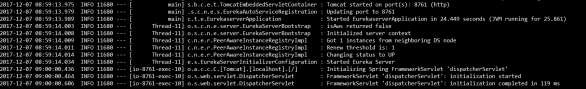
registerWithEureka: false

fetchRegistry: false

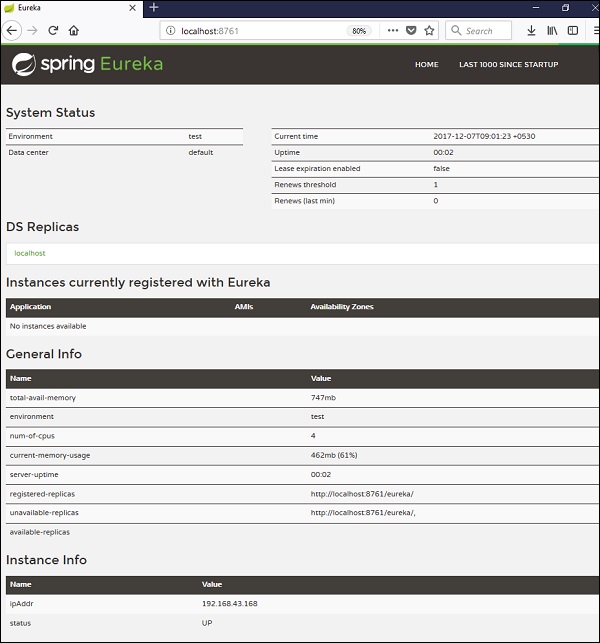
server:

port: 8761

kết quả:



<http://localhost:8761> trong giao diện web , bạn có thể tìm thấy Eureka Server chạy trên cổng 8761:



Chapter 27:

Service Registration with Eureka

Bạn sẽ học chi tiết về cách đăng ký Spring Boot Micro service application trong Eureka Server. Trước khi đăng ký application , hãy chắc chắn răng Eureka Server đang chạy trên cổng 8761 hoặc build lần đầu Eureka Server và chạy nó. Để biết thêm thông tin cho việc building Eureka server , bạn có thể chuyển tới chương trước.

Đầu tiên , bạn cần thêm dependencies trong cấu hình build để đăng ký microservice với Eureka server.

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-eureka')

chúng ta cần thêm @EnableEurekaClient annotation trong main Spring Boot application class. @EnableEurekaClient annotation tạo ứng dụng Spring Boot hoạt động như một Eureka client.

Main Spring Boot application:

package com.tutorialspoint.eurekaclient;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableEurekaClient

public class EurekaclientApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaclientApplication.class, args);

}

}

Để đăng ký với Spring Boot application vào Eureka Server , chúng ta cần thêm cấu hình sau vào trong application.properties file hoặc application.yml file và chỉ định Eureka Server URL trong cấu hình :

Application.yml:

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka

instance:

preferIpAddress: true

spring:

application:

name: eurekaclient

application.properties:

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

eureka.client.instance.preferIpAddress = true

spring.application.name = eurekaclient

Bây giờ , thêm Rest Endpoint để trả về String trong main Spring Boot application và Spring Boot Starter web dependency trong cấu hình build.

package com.tutorialspoint.eurekaclient;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

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

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

@SpringBootApplication

@EnableEurekaClient

@RestController

public class EurekaclientApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaclientApplication.class, args);

}

@RequestMapping(value = "/")

public String home() {

return "Eureka Client application";

}

}

Toàn bộ cấu hình file như sau:

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>eurekaclient</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eurekaclient</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

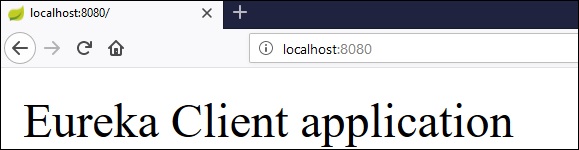
</projecta>

Kết quả:

<http://localhost:8761/>



<http://localhost:8080/>



Chapter 28.

Zuul Proxy Server and Routing

Zuul Server là một gateway application để xử lý toàn bộ request và định tuyết động (điều hướng) của ứng dụng microservice. Zuul Server được biết đến là Edge Server.

Ví dụ , /api/user được map tới user service và /api/products được map tới product service và Zuul Server định tuyến động request tới application backend tương ứng.

Chúng ta sẽ học chi tiết về cách tạo Zuul Server application trong Spring Boot.

Creating Zuul Server application

Zuul Server được đóng gọi trong Spring Cloud dependency.

Thêm @EnableZuulProxy annotation trên main Spring Boot application. @EnableZuulProzy annotation được sử dụng để tạo Spring Boot application hoạt động như một Zuul Prozy server.

package com.tutorialspoint.zuulserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableZuulProxy

public class ZuulserverApplication {

public static void main(String[] args) {

SpringApplication.run(ZuulserverApplication.class, args);

}

}

Bạn sẽ phải thêm Spring Cloud Stater Zuul dependency trong cấu hình build:

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-zuul')

định tuyến Zuul , thêm properties vào application.properties hoặc application.yml

spring.application.name = zuulserver

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

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

server.port = 8111

có nghĩa rằng , http gọi tới /api/demo để chuyển tới products service. Ví dụ , /api/demo/products được điều hướng tới /products

file yaml:

server:

port: 8111

spring:

application:

name: zuulserver

zuul:

routes:

products:

path: /api/demo/\*\*

url: http://localhost:8080/

lưu ý - <http://localhost:8080/> application nên đã được chạy trước khi định tuyến Zuul Proxy.

Cấu hình hoàn chỉnh:

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>zuulserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>zuulserver</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Gradle:

buildscript {

ext {

springBootVersion = '1.5.9.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()

}

ext {

springCloudVersion = 'Edgware.RELEASE'

}

dependencies {

compile('org.springframework.cloud:spring-cloud-starter-zuul')

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

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"

}

}

Kết quả:

Started Application on Tomcat Port_8111

<http://localhost:8111/api/demo/products> trong giao diện web và bạn có thể thấy output /products REST Endpoint:



Chapter 29.

Spring Boot – Spring Cloud Configuration Server

Spring Cloud Configuration Server là một ứng dụng tập trung quản lý toàn bộ application liên quan thuộc tính cấu hình. Bạn sẽ học chỉ tiết về cách để tạo Spring Cloud Configuration server.

Creating Spring Cloud Configuration Server

Bây giờ , thêm Spring Cloud Config server dependency trong cấu hình build:

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-config-server')

Bây giờ , thêm @EnableConfigServer annotation trong main Spring Boot application class. @EnableConfigServer annotation tạo Spring Boot application hoạt động như một Configuration Server.

Main Spring Boot application:

package com.tutorialspoint.configserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableConfigServer

public class ConfigserverApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigserverApplication.class, args);

}

}

Bây giờ , thêm cấu hình dưới đây vào properties file và thay thế application.properties thành bootstrap.properties file

server.port = 8888

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

SPRING\_PROFILES\_ACTIVE=native

Configuration Server chạy trên Tomcat cổng 8888 và thuộc tính configuration application được load từ địa điểm tìm kiếm.

Bây giờ , trong <file:///C:/configprop/> nơi application.properties file – của client application. Ví dụ , client application của bạn tên là config-client , và đổi tên application.properties file thành config-client.properties và lưu trữ properties file trên đường dẫn <file:///C:/configprop/>

Code cho config-client properties file là:

welcome.message = Welcome to Spring cloud config server

code hoàn chỉnh:

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>configserver</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>configserver</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

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

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

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

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

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

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

Gradle:

<scope>import</scope>

</dependency>

</dependencies>

buildscript {

ext {

springBootVersion = '1.5.9.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()

}

ext {

springCloudVersion = 'Edgware.RELEASE'

}

dependencies {

compile('org.springframework.cloud:spring-cloud-config-server')

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

}

dependencyManagement {

imports {

mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"

}

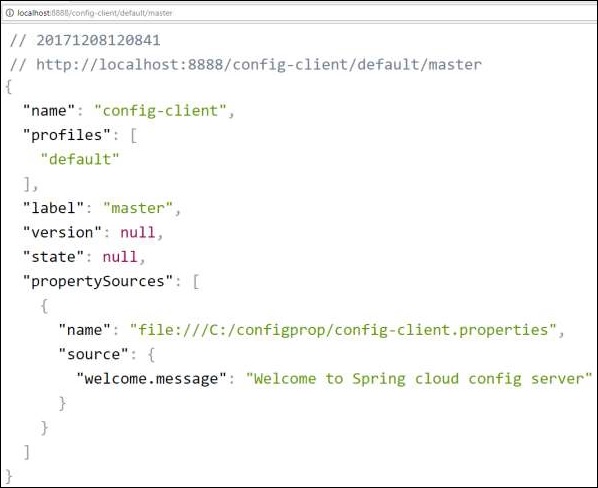
}

Kết quả Tomcat cổng 8888:

Tomcat Port 8888 Output

URL: <http://localhost:8888/config-cliend/default/master> trên giao diện web:

Bạn có thể thấy thuộc tính config-client application configuration properties:



Chapter 30

Spring Boot – Cloud Configuration Client

Một vài applications có thể cần cấu hình properties , cấu hình properties có thể cần một sự thay đổi và các developers có thể cần tắt hoặc khởi động lại application để thực thi chúng.Tuy nhiên , điều này có thể dẫn đến downtime “ thời gian chết” trong việc phát triển và cần thiết phải khởi động lại ứng dụng. Spring Cloud Configuration Server cho phép các developers load cấu hình properties mới mà không cần việc khởi động lại application và không cần bất kể downtime nào.

Working with Spring Cloud Configuration Server

Đầu tiên, tạo project Spring Boot và chọn Spring Cloud Config Client dependency. Bây giờ , thêm Spring Cloud Starter Config dependency trong file cấu hình của bạn.

Maven :

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-config')

Bây giờ , bạn cần thêm @RefreshScope annotation vào main Spring Boot application . @RefreshScope annotation được sử dụng để load lại giá trị configuration properties từ config server.

package com.example.configclient;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.context.config.annotation.RefreshScope;

@SpringBootApplication

@RefreshScope

public class ConfigclientApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigclientApplication.class, args);

}

}

Thêm config server URL trong application.properties và cung cấp tên application.

Lưu ý – <http://localhost:888> cấu hình server nên chạy trước khi bắt đầu config client application.

spring.application.name = config-client

spring.cloud.config.uri = http://localhost:8888

code viết một REST Endpoint đơn giản để đọc tin nhắn welcome từ configuration server :

package com.example.configclient;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.context.config.annotation.RefreshScope;

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

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

@SpringBootApplication

@RefreshScope

@RestController

public class ConfigclientApplication {

@Value("${welcome.message}")

String welcomeText;

public static void main(String[] args) {

SpringApplication.run(ConfigclientApplication.class, args);

}

@RequestMapping(value = "/")

public String welcomeText() {

return welcomeText;

}

}

Kết quả:

Bạn có thể nhìn trên cửa sổ console , config-client application được nạp cấu hình từ <http://localhost:8888>

2017-12-08 12:41:57.682 INFO 1104 --- [

main] c.c.c.ConfigServicePropertySourceLocator :

Fetching config from server at: http://localhost:8888

<http://localhost:8080/> tin nhắn welcome được tải lên từ configuration server.



Bây giờ , thay đổi giá trị property trên configuration server và chạy actuator Endpoint POST URL <http://localhost:8080/refresh> và xem giá trị configuration property mới trong URL <http://localhost:8080/>

Chapter 31

Spring Boot – Actuator

Spring Boot Actuator cung cấp endpoints bảo mật để theo dõi và quản lý Spring Boot application. Mặc định , toàn bộ actuator endpoints được bảo mật. Bạn sẽ học chi tiết về cách để kích hoạt Spring Boot actuator cho application của bạn.

Enabling Spring Boot Actuator

Để kích hoạt enable Spring Boot actuator endpoints cho Spring Boot application, chúng ta cần thêm Spring Boot Starter actuator dependency trong cấu hình build của bạn.

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile group: 'org.springframework.boot', name: 'spring-boot-starter-actuator'

trong application.properties chúng ta cần disable security cho actuator endpoint.

management.security.enabled = false

YAML:

management:

security:

enabled: false

nếu bạn muốn sử dụng số cổng riêng biệt cho truy cập Spring Boot actuator endpoints , thêm quản lý số cổng trong application.properties:

management.port = 9000

YAML:

management:

port: 9000

kết quả chạy cổng 8080, lưu ý rằng nếu bạn chỉ định số cổng quản lý , sau đó cùng application đó sẽ chạy trên 2 cổng khác nhau:

Startedc Application on Tomcat Port

Một số quan trọng của Spring Boot Actuator endpoints đc đưa ra dưới đây , bạn có thể bắt đầu với chúng trong giao diện web của bạn và theo dõi hành vi application của bạn.

/metrics : Để xem các số liệu application như bộ nhớ được sử dụng , bộ nhớ trống , luồng , các classes , thời gian hoạt động hệ thống , v..v

/env : Để xem danh sách các biến môi trường được sử dụng trong application

/beans : Để xem các Spring beans và kiểu của chúng , phạm vi và dependency

/health : Để xem tình trạng ứng dụng

/info : Để xem thông tin về Spring Boot application

/trace : Để xem list các Traces của Rest endpoints

Chapter 32

Spring Boot – Admin Server

Theo dõi application của bạn bằng cách sử dụng Spring Boot Actuator Endpoint hơi khó. Bởi vì , nếu bạn có “n” số lượng applications , mỗi application sẽ có một endpoint actuator riêng biệt , do vậy làm cho việc theo dõi khó khăn. Spring Boot Admin Server là một application được sử dụng để quản lý và theo dõi Microservice application.

Để xử lý những tình huống như vậy , CodeCentric Team cung cấp Spring Boot Admin UI để quản lý và theo dõi toàn bộ Spring Boot application Actuator endpoints tại một nơi.

Để xây dựng một Spring Boot Admin Server chúng ta cần thêm các dependencies dưới đây vào file cấu hình.

Maven:

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-server</artifactId>

<version>1.5.5</version>

</dependency>

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-server-ui</artifactId>

<version>1.5.5</version>

</dependency>

Gradle:

compile group: 'de.codecentric', name: 'spring-boot-admin-server', version: '1.5.5'

compile group: 'de.codecentric', name: 'spring-boot-admin-server-ui', version: '1.5.5'

Thêm @EnableAdminServer annotation trong main Spring Boot application class. @EnableAdminServer annotation được sử dụng để làm cho ứng dụng của bạn là Admin Server để theo dõi toàn bộ microservices khác.

package com.tutorialspoint.adminserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import de.codecentric.boot.admin.config.EnableAdminServer;

@SpringBootApplication

@EnableAdminServer

public class AdminserverApplication {

public static void main(String[] args) {

SpringApplication.run(AdminserverApplication.class, args);

}

}

Bây giờ , định nghĩa cổng sever và tên application trong application.properties:

server.port = 9090

spring.application.name = adminserver

YAML:

server:

port: 9090

spring:

application:

name: adminserver

kết quả: <http://localhost:9090>



Chapter 32

Spring Boot - Admin Client

Cho việc theo dõi và quản lý microservice application thông qua Spring Boot Admin Server , bạn nên thêm Spring Boot Admin starter client dependency và chỉ ra Admin Servcer URL trong application.properties

Lưu ý – theo dõi một application , bạn nên kích hoạt Spring Boot Actuator Endpoints cho Microservice application.

Đầu tiên , thêm Spring Boot Admin starter client dependency và Spring Boot starter actuator dependency trong cấu hình build của bạn.

Maven:

<dependency>

<groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-starter-client</artifactId>

<version>1.5.5</version>

</dependency>

<dependency>

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

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

</dependency>

Gradle:

compile group: 'de.codecentric', name: 'spring-boot-admin-starter-client', version: '1.5.5'

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

bây giờ , thêm Spring Boot Admin Server URL vào application.properties:

cho việc cấu hình users , thêm properties vào application.properties:

spring.boot.admin.url = http://localhost:9090/

YAML:

spring:

boot:

admin:

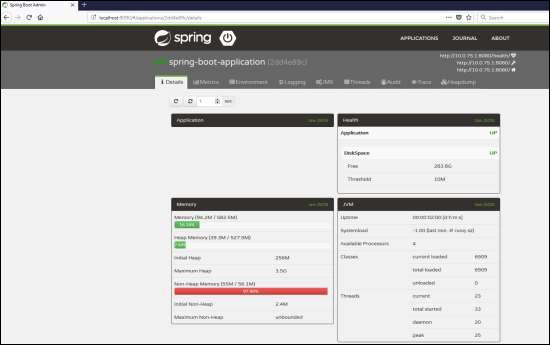
url: http://localhost:9000/

kết quả:

<http://localhost:9090/> từ giao diện web và thấy Spring Boot application được đăng ký với Spring Boot Admin Server



Click Details để xem các actuator endpoints trong Admin Server UI



Chapter 33.

Enabling Swagger2

Swagger2 là một dự án mã nguồn mở được sử dụng để generate tài liệu REST API cho RESTful web services. Nó cung cấp một giao diện user để truy cập RESTful web servicers thông qua giao diện web.

Để kích hoạt Swagger2 trong Spring Boot application , bạn cần thêm dependencies trong cấu hình build:

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

Cho Gradle:

compile group: 'io.springfox', name: 'springfox-swagger2', version: '2.7.0'

compile group: 'io.springfox', name: 'springfox-swagger-ui', version: '2.7.0'

Bây giờ , thêm @EnableSwagger2 annotation trong main Spring Boot application.

@EnableSwagger2 annotation được sử dụng để kích hoạt Swagger2 cho Spring Boot application của bạn.

Code main Spring Boot application:

package com.tutorialspoint.swaggerdemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication

@EnableSwagger2

public class SwaggerDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SwaggerDemoApplication.class, args);

}

}

Tiếp theo , tạo một Docket Bean để configure Swagger2 cho Spring Boot application của bạn , chúng ta cần định nghĩa base package để cấu hình REST API(s) cho Swagger2

@Bean

public Docket productApi() {

return new Docket(DocumentationType.SWAGGER\_2).select() .apis(RequestHandlerSelectors.basePackage("com.tutorialspoint.swaggerdemo")).build();

}

Bây giờ , thêm bean trong main Spring Boot application class :

package com.tutorialspoint.swaggerdemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication

@EnableSwagger2

public class SwaggerDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SwaggerDemoApplication.class, args);

}

@Bean

public Docket productApi() {

return new Docket(DocumentationType.SWAGGER\_2).select()

.apis(RequestHandlerSelectors.basePackage("com.tutorialspoint.swaggerdemo")).build();

}

}

Bây giờ thêm Spring Boot Starter Web dependency trong cấu hình build để viết một REST Endpoints:

Maven:

<dependency>

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

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

</dependency>

Gradle:

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

code build 2 ví dụ RESTful web services GET và POST trong Rest Controller:

package com.tutorialspoint.swaggerdemo;

import java.util.ArrayList;

import java.util.List;

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

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

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

@RestController

public class SwaggerAPIController {

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

public List<String> getProducts() {

List<String> productsList = new ArrayList<>();

productsList.add("Honey");

productsList.add("Almond");

return productsList;

}

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

public String createProduct() {

return "Product is saved successfully";

}

}

Kết quả:

<http://localhost:8080/swagger-ui.html>



Chapter 34

Spring Boot – Creating Docker Image

Docker là một service quản lí container giúp dễ dàng xây dựng và triển khai “deployment” . tìm hiểu về Docker: <https://www.tutorialspoint.com/docker/index.htm>

Bạn sẽ học cách tạo một Docker image bằng cách sử dụng Maven và Gradle dependencies cho Spring Boot application.

Create Dockerfile:

Đầu tiên , tạo một file với tên là Dockerfile chứa trọng thư viện src/main/docker với chủ đề dưới đây , lưu ý rằng – file này rất quan trọng để tạo một Docker image

FROM java:8

VOLUME /tmp

ADD dockerapp-0.0.1-SNAPSHOT.jar app.jar

RUN bash -c 'touch /app.jar'

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]

Maven:

<properties>

<docker.image.prefix>spring-boot-tutorialspoint</docker.image.prefix>

</properties>

<build>

<plugins>

<plugin>

<groupId>com.spotify</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>1.0.0</version>

<configuration>

<imageName>${docker.image.prefix}/${project.artifactId}</imageName>

<dockerDirectory>src/main/docker</dockerDirectory>

<resources>

<resource>

<directory>${project.build.directory}</directory>

<include>${project.build.finalName}.jar</include>

</resource>

</resources>

</configuration>

</plugin>

<plugin>

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

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

</plugin>

</plugins>

</build>

Code hoàn chỉnh:

<?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>dockerapp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>dockerapp</name>

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

<parent>

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

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

<version>1.5.9.RELEASE</version>

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

</parent>

<properties>

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

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

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

<docker.image.prefix>spring-boot-tutorialspoint</docker.image.prefix>

</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>com.spotify</groupId>

<artifactId>docker-maven-plugin</artifactId>

<version>1.0.0</version>

<configuration>

<imageName>${docker.image.prefix}/${project.artifactId}</imageName>

<dockerDirectory>src/main/docker</dockerDirectory>

<resources>

<resource>

<directory>${project.build.directory}</directory>

<include>${project.build.finalName}.jar</include>

</resource>

</resources>

</configuration>

</plugin>

<plugin>

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

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

</plugin>

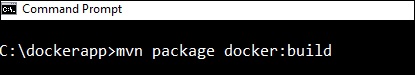
</plugins>

</build>

</project>

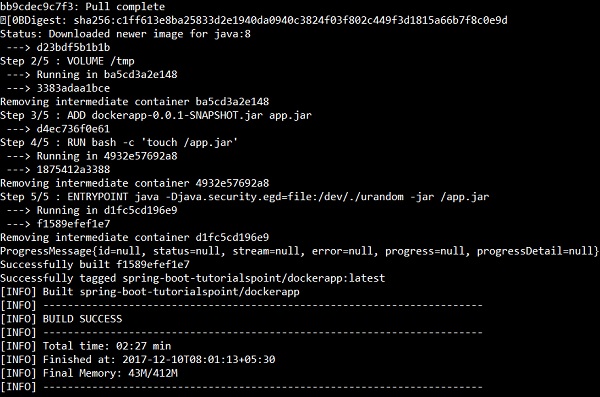
Bạn có thể chạy trên application bằng cách sử dụng Maven command :

-mvn package docker: build



Lưu ý – kích hoạt Expose daemon trên tcp://localhost:2375 không có TLS

Sau khi build thành công , bạn có thể thấy output trên console:



Giờ đây , thấy Docker images bằng cách sử dụng command docker images và nhìn thông tin image trên console.



Gradle tương tự : tìm hiểu trên link <https://www.tutorialspoint.com/spring_boot/spring_boot_creating_docker_image.htm>

Chapter 35.

Spring Boot – Tracing Micro Service Logs

Hầu hết các developers phải đối mặt với khó khăn của việc theo dõi logs nếu có bất kỳ vấn đề nào xảy ra. Điều này có thể được giải quyết bằng Spring Cloud Sleuth và ZipKin server cho Spring Boot application.

Spring Cloud Sleuth

Spring cloud Sleuth logs được in theo định dạng sau :

[application-name,traceid,spanid,zipkin-export]

Với :

* Application-name = tên của application
* Traceid = mỗi một request và response , traceid là giống nhau khi việc gọi cùng 1 service hoặc 1 service tới service khác.
* Spandid = Spanid được in cùng với Traceid , Spandid thì khác nhau với mỗi request và response gọi 1 service tới một service khác.
* Zipkin-export = mặc định là false , nếu nó true , logs sẽ được xuất sang Zipkin server.

Bây giờ , thêm Spring Cloud Starter Sleuth dependency trong cấu hình build của bạn.

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-sleuth')

Bây giờ , thêm logs vào Spring Boot application Rest Controller class:

package com.tutorialspoint.sleuthapp;

import java.util.logging.Level;

import java.util.logging.Logger;

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 SleuthappApplication {

private static final Logger LOG = Logger.getLogger(SleuthappApplication.class.getName());

public static void main(String[] args) {

SpringApplication.run(SleuthappApplication.class, args);

}

@RequestMapping("/")

public String index() {

LOG.log(Level.INFO, "Index API is calling");

return "Welcome Sleuth!";

}

}

Bây giờ , thêm application name trong application.properties:

spring.application.name = tracinglogs

kết quả:

<http://localhost:8080/>



Bạn có thể nhìn logs dưới đây trên cửa sổ console. Quan sát log được in theo định dạng :

Log is Printed

Zipkin Server

Zipkin là một application để theo dõi và quản lý Spring Cloud Sleuth logs của Spring Boot application của bạn. Để build một Zipkin server , chúng ta cần thêm Zipkin UI và Zipkin Server dependencies trong cấu hình build của bạn:

Maven:

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-server</artifactId>

</dependency>

<dependency>

<groupId>io.zipkin.java</groupId>

<artifactId>zipkin-autoconfigure-ui</artifactId>

</dependency>

Gradle:

compile('io.zipkin.java:zipkin-autoconfigure-ui')

compile('io.zipkin.java:zipkin-server')

Bây giờ , cấu hình cổng server = 9411 trong application.properties

Thêm thuộc tính trong application.properties:

server.port = 9411

YAML:

server:

port: 9411

Thêm @EnableZipkinServer annotation trong main Spring Boot application class. @EnableZipkinServer annotation được sử dụng để kích hoạt application của bạn hoạt động như một Zipkin server.

package com.tutorialspoint.zipkinapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import zipkin.server.EnableZipkinServer;

@SpringBootApplication

@EnableZipkinServer

public class ZipkinappApplication {

public static void main(String[] args) {

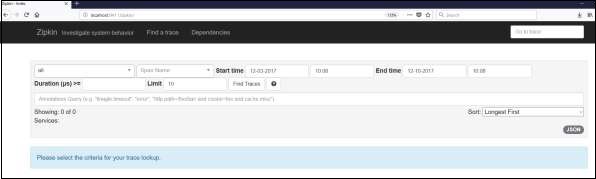
SpringApplication.run(ZipkinappApplication.class, args);

}

}

Kết quả:

<http://localhost:9411/zipkin/>



Sau đó , thêm dependency trong client service application và point out Zipkin Server URL để theo dõi nhật kí microservice thông qua Zipkin UI

Thêm Spring Cloud Starter Zipkin dependency trong cấu hình build của bạn:

Maven:

<dependency>

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

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-sleuth-zipkin')

Thêm Always Sampler Bean trong Spring Boot application để xuất nhật kí sang Zipkin server.

@Bean

public AlwaysSampler defaultSampler() {

return new AlwaysSampler();

}

Nếu bạn thêm AlwaysSampler Bean , sau đó tự động Spring Sleuth Zipkin Export option sẽ thay đổi từ false sang true.

Tiếp theo , cấu hình Zipkin Server dựa trên URL trong client service application.properties file.

spring.zipkin.baseUrl = http://localhost:9411/zipkin/

sau đó cung cấp trace id và tìm traces trong Zipkin UI.

http://localhost:9411/zipkin/traces/{traceid}

Chapter 36.

Spring Boot – Flyway Database

Flyway là một phiên bản ứng dụng điều khiển để phát triển Database scheme của bạn một cách dễ dàng và đáng tin cậy trên toàn bộ các trường hợp phiên bản của bạn. Để học nhiều hơn về Flyway , bạn có thể sử dụng link: - [www.flywaydb.org](http://www.flywaydb.org)

Rất nhiều dự án phần mềm sử dụng cơ sở dữ liệu quan hệ. Điều này là bắt buộc cho việc xử lý di chuyển cơ sở dữ liệu , còn thường được gọi là di chuyển lược đồ.

Bạn sẽ học chi tiết về cách cấu hình Flyway database trong Spring Boot application.

Configuring Flyway Database

Đầu tiên , tạo dự án Spring Boot có các dependencies:

* Spring Boot Starter Web
* Flyway
* MySQL
* JDBC

Maven:

<dependency>

<groupId>org.flywaydb</groupId>

<artifactId>flyway-core</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

Gradle:

compile('org.flywaydb:flyway-core')

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

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

compile('mysql:mysql-connector-java')

Trong application properties , chúng ta cần cấu hình database properties cho việc tạo một DataSource và flyway properties , chúng ta cần cấu hình trong application.properties.

spring.application.name = flywayapp

spring.datasource.driverClassName = com.mysql.jdbc.Driver

spring.datasource.url = jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true

spring.datasource.username = root

spring.datasource.password = root

spring.datasource.testOnBorrow = true

spring.datasource.testWhileIdle = true

spring.datasource.timeBetweenEvictionRunsMillis = 60000

spring.datasource.minEvictableIdleTimeMillis = 30000

spring.datasource.validationQuery = SELECT 1

spring.datasource.max-active = 15

spring.datasource.max-idle = 10

spring.datasource.max-wait = 8000

flyway.url = jdbc:mysql://localhost:3306/mysql

flyway.schemas = USERSERVICE

flyway.user = root

flyway.password = root

YAML:

spring:

application:

name: flywayapp

datasource:

driverClassName: com.mysql.jdbc.Driver

url: "jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true"

password: "root"

username: "root"

testOnBorrow: true

testWhileIdle: true

timeBetweenEvictionRunsMillis: 60000

minEvictableIdleTimeMillis: 30000

validationQuery: SELECT 1

max-active: 15

max-idle: 10

max-wait: 8000

flyway:

url: jdbc:mysql://localhost:3306/mysql

schemas: USERSERVICE

user: "root"

password: "root"

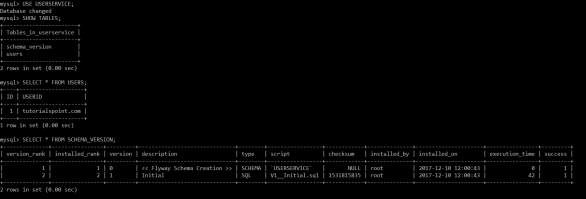
bây giờ , tạo một file SQL chứa trong thư mục src/main/resources/db/migration.

Tên file SQL là “V1\_\_Initial.sql”

CREATE TABLE USERS (ID INT AUTO\_INCREMENT PRIMARY KEY, USERID VARCHAR(45));

INSERT INTO USERS (ID, USERID) VALUES (1, 'tutorialspoint.com');

Kết quả:



Chapter 37.

Spring Boot – Sending Email

Bằng cách sử dụng Spring Boot RESTful web service , bạn có thể gửi email với Gmail Transport Layer Security. Chúng ta sẽ học chi tiết cách sử dụng tính năng này.

Đầu tiên , chúng ta cần thêm Spring Boot Starter Mail dependency trong cấu hình build.

Maven:

<dependency>

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

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

</dependency>

Gradle:

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

code cho main Spring Boot application class:

package com.tutorialspoint.emailapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EmailappApplication {

public static void main(String[] args) {

SpringApplication.run(EmailappApplication.class, args);

}

}

Bạn có thể viết một Rest API đơn giản để gửi email trong Rest Controller class:

package com.tutorialspoint.emailapp;

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

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

@RestController

public class EmailController {

@RequestMapping(value = "/sendemail")

public String sendEmail() {

return "Email sent successfully";

}

}

Bạn có thể viết một phương thức để gửi email với tập tin đính kèm. Định nghĩa mail.smtp properties và sử dụng PasswordAuthentication.

private void sendmail() throws AddressException, MessagingException, IOException {

Properties props = new Properties();

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

props.put("mail.smtp.host", "smtp.gmail.com");

props.put("mail.smtp.port", "587");

Session session = Session.getInstance(props, new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication("tutorialspoint@gmail.com", "<your password>");

}

});

Message msg = new MimeMessage(session);

msg.setFrom(new InternetAddress("tutorialspoint@gmail.com", false));

msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse("tutorialspoint@gmail.com"));

msg.setSubject("Tutorials point email");

msg.setContent("Tutorials point email", "text/html");

msg.setSentDate(new Date());

MimeBodyPart messageBodyPart = new MimeBodyPart();

messageBodyPart.setContent("Tutorials point email", "text/html");

Multipart multipart = new MimeMultipart();

multipart.addBodyPart(messageBodyPart);

MimeBodyPart attachPart = new MimeBodyPart();

attachPart.attachFile("/var/tmp/image19.png");

multipart.addBodyPart(attachPart);

msg.setContent(multipart);

Transport.send(msg);

}

Bây giờ , gọi phương thức sendmail() từ Rest API :

@RequestMapping(value = "/sendemail")

public String sendEmail() throws AddressException, MessagingException, IOException {

sendmail();

return "Email sent successfully";

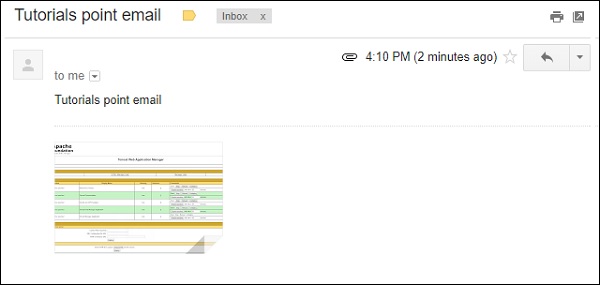
}

Lưu ý – vui lòng bật cho phép các ứng dụng kém an toàn hơn trong cài đặt tài khoản Gmail trước khi gửi một email.

Kết quả:

<http://localhost:8080/sendemail>





Chapter 38.

Spring Boot – Hystrix

Hystrix là một thư viện từ Netflix. Hystrix cô lập các điểm truy cập giữa các services, ngăn chặn việc xếp tầng thất bại trên chúng và cũng cấp các tùy chọn dự phòng.

Ví dụ , khi bạn đang gọi tới ứng dụng bên thứ 3 , nó sẽ mất nhiều thời gian để gửi phản hồi , vì thế tại thời điểm đó , control chuyển sang phương thức dự phòng và trả về phản hồi tùy ý tới application của bạn.

Bạn sẽ học cách triển khai Hystrix trong một Spring Boot application.

Đầu tiên , bạn cần thêm Spring Cloud Starter Hystrix dependency trong cấu hình build.

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile('org.springframework.cloud:spring-cloud-starter-hystrix')

bây giờ , thêm @EnableHystrix annotation trong main Spring Boot application class. @EnableHystrix annotation được sử dụng để kích hoạt các chức năng Hystrix trong Spring Boot application.

Main Spring Boot application:

package com.tutorialspoint.hystrixapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.hystrix.EnableHystrix;

@SpringBootApplication

@EnableHystrix

public class HystrixappApplication {

public static void main(String[] args) {

SpringApplication.run(HystrixappApplication.class, args);

}

}

Viết một Rest Controller đơn giản và trả về chuỗi String sau 3 giây từ lúc thời điểm gửi request:

@RequestMapping(value = "/")

public String hello() throws InterruptedException {

Thread.sleep(3000);

return "Welcome Hystrix";

}

Bây giờ , thêm @HystrixCommand và @HystrixProperty cho Rest API và định nghĩa timeout trong giá trị milliseconds:

@HystrixCommand(fallbackMethod = "fallback\_hello", commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "1000")

})

Tiếp theo , định nghĩa phương thức dự phòng nếu request mất quá nhiều thời gian để phản hồi:

private String fallback\_hello() {

return "Request fails. It takes long time to response";

}

Rest Controller hoàn chỉnh chứa REST API và Hystrix properties:

@RequestMapping(value = "/")

@HystrixCommand(fallbackMethod = "fallback\_hello", commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "1000")

})

public String hello() throws InterruptedException {

Thread.sleep(3000);

return "Welcome Hystrix";

}

private String fallback\_hello() {

return "Request fails. It takes long time to response";

}

trong ví dụ này , REST API được viết trong main Spring Boot application:

package com.tutorialspoint.hystrixapp;

import org.springframework.boot.SpringApplication;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.hystrix.EnableHystrix;

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

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

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@SpringBootApplication

@EnableHystrix

@RestController

public class HystrixappApplication {

public static void main(String[] args) {

SpringApplication.run(HystrixappApplication.class, args);

}

@RequestMapping(value = "/")

@HystrixCommand(fallbackMethod = "fallback\_hello", commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "1000")

})

public String hello() throws InterruptedException {

Thread.sleep(3000);

return "Welcome Hystrix";

}

private String fallback\_hello() {

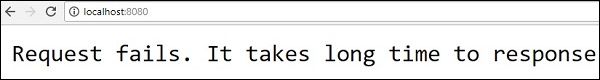
return "Request fails. It takes long time to response";

}

}

Kết quả :

<http://localhost:8080/> từ giao diện web và nhận phản hồi từ Hystrix. API mất 3 giây để phản hồi nhưng Hystrix timout là 1 giây :



Chapter 39.

Spring Boot – Web Socket

Chúng ta tìm hiểu cách xây dựng một ứng dụng web tương tác bằng cách sử dụng Spring Boot với Web sockets.

Để xây dựng một ứng dụng web tương tác trong Spring Boot với Web socket, bạn cần thêm dependencies sau đây:

Maven:

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>webjars-locator</artifactId>

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>sockjs-client</artifactId>

<version>1.0.2</version>

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>stomp-websocket</artifactId>

<version>2.3.3</version>

</dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>bootstrap</artifactId>

<version>3.3.7</version> </dependency>

<dependency>

<groupId>org.webjars</groupId>

<artifactId>jquery</artifactId>

<version>3.1.0</version>

</dependency>

Gradle:

compile("org.springframework.boot:spring-boot-starter-websocket")

compile("org.webjars:webjars-locator")

compile("org.webjars:sockjs-client:1.0.2")

compile("org.webjars:stomp-websocket:2.3.3")

compile("org.webjars:bootstrap:3.3.7")

compile("org.webjars:jquery:3.1.0")

Tạo một controller xử lý Message để hoạt động với STOMP messaging. STOMP messages có thể định tuyến tới @Controller class. Ví dụ , GreetingController được map tới xử lý messages đến đích “/hello”.

package com.tutorialspoint.websocketapp;

import org.springframework.messaging.handler.annotation.MessageMapping;

import org.springframework.messaging.handler.annotation.SendTo;

import org.springframework.stereotype.Controller;

@Controller

public class GreetingController {

@MessageMapping("/hello")

@SendTo("/topic/greetings")

public Greeting greeting(HelloMessage message) throws Exception {

Thread.sleep(1000); // simulated delay

return new Greeting("Hello, " + message.getName() + "!");

}

}

Bây giờ , cấu hình Spring cho STOMP messaging. Viết một WebSocketConfig class kế thừa AbstractWebSocketMessageBrokerConfigurer class :

package com.tutorialspoint.websocketapp;

import org.springframework.context.annotation.Configuration;

import org.springframework.messaging.simp.config.MessageBrokerRegistry;

import org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBrokerConfigurer;

import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;

import org.springframework.web.socket.config.annotation.StompEndpointRegistry;

@Configuration

@EnableWebSocketMessageBroker

public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

@Override

public void configureMessageBroker(MessageBrokerRegistry config) {

config.enableSimpleBroker("/topic");

config.setApplicationDestinationPrefixes("/app");

}

@Override

public void registerStompEndpoints(StompEndpointRegistry registry) {

registry.addEndpoint("/tutorialspoint-websocket").withSockJS();

}

}

@EnableWebSocketMessageBroker annotation được sử dụng để cấu hình Web socket message broker để tạo STOMP endpoints.

Bạn có thể tạo một giao diện client chứa trọng src/main/resources/static/index.html:

<!DOCTYPE html>

<html>

<head>

<title>Hello WebSocket</title>

<link href = "/webjars/bootstrap/css/bootstrap.min.css" rel = "stylesheet">

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

<script src = "/webjars/jquery/jquery.min.js"></script>

<script src = "/webjars/sockjs-client/sockjs.min.js"></script>

<script src = "/webjars/stomp-websocket/stomp.min.js"></script>

<script src = "/app.js"></script>

</head>

<body>

<noscript>

<h2 style = "color: #ff0000">

Seems your browser doesn't support Javascript! Websocket relies on Javascript being

enabled. Please enable Javascript and reload this page!

</h2>

</noscript>

<div id = "main-content" class = "container">

<div class = "row">

<div class = "col-md-6">

<form class = "form-inline">

<div class = "form-group">

<label for = "connect">WebSocket connection:</label>

<button id = "connect" class = "btn btn-default" type = "submit">Connect</button>

<button id = "disconnect" class = "btn btn-default" type = "submit" disabled = "disabled">Disconnect

</button>

</div>

</form>

</div>

<div class = "col-md-6">

<form class = "form-inline">

<div class = "form-group">

<label for = "name">What is your name?</label>

<input type = "text" id = "name" class = "form-control" placeholder = "Your name here...">

</div>

<button id = "send" class = "btn btn-default" type = "submit">Send</button>

</form>

</div>

</div>

<div class = "row">

<div class = "col-md-12">

<table id = "conversation" class = "table table-striped">

<thead>

<tr>

<th>Greetings</th>

</tr>

</thead>

<tbody id = "greetings"></tbody>

</table>

</div>

</div>

</div>

</body>

</html>

Tạo một app.js để tích hợp và tạo tin nhắn bằng cách sử dụng STOMP:

var stompClient = null;

function setConnected(connected) {

$("#connect").prop("disabled", connected);

$("#disconnect").prop("disabled", !connected);

if (connected) {

$("#conversation").show();

} else {

$("#conversation").hide();

}

$("#greetings").html("");

}

function connect() {

var socket = new SockJS('/tutorialspoint-websocket');

stompClient = Stomp.over(socket);

stompClient.connect({}, function (frame) {

setConnected(true);

console.log('Connected: ' + frame);

stompClient.subscribe('/topic/greetings', function (greeting) {

showGreeting(JSON.parse(greeting.body).content);

});

});

}

function disconnect() {

if (stompClient !== null) {

stompClient.disconnect();

}

setConnected(false);

console.log("Disconnected");

}

function sendName() {

stompClient.send("/app/hello", {}, JSON.stringify({'name': $("#name").val()}));

}

function showGreeting(message) {

$("#greetings").append("<tr><td>" + message + "</td></tr>");

}

$(function () {

$( "form" ).on('submit', function (e) {e.preventDefault();});

$( "#connect" ).click(function() { connect(); });

$( "#disconnect" ).click(function() { disconnect(); });

$( "#send" ).click(function() { sendName(); });

});

Code cho main Spring Boot application:

package com.tutorialspoint.websocketapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class WebsocketappApplication {

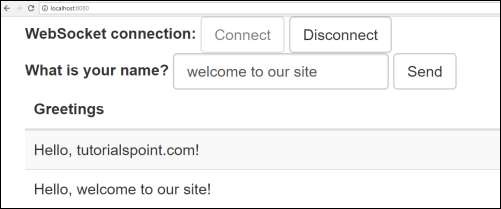
public static void main(String[] args) {

SpringApplication.run(WebsocketappApplication.class, args);

}

}

Kết quả: <http://localhost:8080/> trong giao diện web và kết nối đến web socket và gửi một greeting và nhận lại message.



Chapter 40 – Batch Service

Bath Service là một tiến trình thực thi nhiều hơn một câu lệnh trong một single task . Bạn sẽ học cách tạo một batch service trong Spring Boot application.

Chúng ta sẽ xem xét 1 ví dụ nơi chúng ta sẽ lưu trữ nội dung CSV file trong HSQLDB.

Để tạo chương trình Batch Service , chúng ta cần thêm Spring Boot Starter Batch dependency và HSQLDB dependency trong cấu hình build.

Maven:

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

</dependency>

Gradle:

compile("org.springframework.boot:spring-boot-starter-batch")

compile("org.hsqldb:hsqldb")

tiếp theo , viết một câu lệnh SQL cho HSQLDB chứa trong lớp đường dẫn thư mục - **request\_fail\_hystrix\_timeout**

DROP TABLE USERS IF EXISTS;

CREATE TABLE USERS (

user\_id BIGINT IDENTITY NOT NULL PRIMARY KEY,

first\_name VARCHAR(20),

last\_name VARCHAR(20)

);

Tạo một POJO class cho USERS:

package com.tutorialspoint.batchservicedemo;

public class User {

private String lastName;

private String firstName;

public User() {

}

public User(String firstName, String lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

@Override

public String toString() {

return "firstName: " + firstName + ", lastName: " + lastName;

}

}

Tạo một bộ xử lý trung gian để thực hiện các hành động sau khi đọc dữ liệu từ CSV file và trước khi viết dữ liệu vào SQL:

package com.tutorialspoint.batchservicedemo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.batch.item.ItemProcessor;

public class UserItemProcessor implements ItemProcessor<User, User> {

private static final Logger log = LoggerFactory.getLogger(UserItemProcessor.class);

@Override

public User process(final User user) throws Exception {

final String firstName = user.getFirstName().toUpperCase();

final String lastName = user.getLastName().toUpperCase();

final User transformedPerson = new User(firstName, lastName);

log.info("Converting (" + user + ") into (" + transformedPerson + ")");

return transformedPerson;

}

}

Tạo một cấu hình Batch để đọc dữ liệu từ CSV và viết vào trong SQL , chúng ta cần thêm @EnableBatchProcessing annotation trong configuration. @EnableBatchProcessing annotation được sử dụng để kích hoạt hành động batch cho Spring Boot application.

package com.tutorialspoint.batchservicedemo;

import javax.sql.DataSource;

import org.springframework.batch.core.Job;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;

import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;

import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;

import org.springframework.batch.core.launch.support.RunIdIncrementer;

import org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider;

import org.springframework.batch.item.database.JdbcBatchItemWriter;

import org.springframework.batch.item.file.FlatFileItemReader;

import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;

import org.springframework.batch.item.file.mapping.DefaultLineMapper;

import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.io.ClassPathResource;

@Configuration

@EnableBatchProcessing

public class BatchConfiguration {

@Autowired

public JobBuilderFactory jobBuilderFactory;

@Autowired

public StepBuilderFactory stepBuilderFactory;

@Autowired

public DataSource dataSource;

@Bean

public FlatFileItemReader<User> reader() {

FlatFileItemReader<User> reader = new FlatFileItemReader<User>();

reader.setResource(new ClassPathResource("file.csv"));

reader.setLineMapper(new DefaultLineMapper<User>() {

{

setLineTokenizer(new DelimitedLineTokenizer() {

{

setNames(new String[] { "firstName", "lastName" });

}

});

setFieldSetMapper(new BeanWrapperFieldSetMapper<User>() {

{

setTargetType(User.class);

}

});

}

});

return reader;

}

@Bean

public UserItemProcessor processor() {

return new UserItemProcessor();

}

@Bean

public JdbcBatchItemWriter<User> writer() {

JdbcBatchItemWriter<User> writer = new JdbcBatchItemWriter<User>();

writer.setItemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<User>());

writer.setSql("INSERT INTO USERS (first\_name, last\_name) VALUES (:firstName, :lastName)");

writer.setDataSource(dataSource);

return writer;

}

@Bean

public Job importUserJob(JobCompletionNotificationListener listener) {

return jobBuilderFactory.get("importUserJob").incrementer(

new RunIdIncrementer()).listener(listener).flow(step1()).end().build();

}

@Bean

public Step step1() {

return stepBuilderFactory.get("step1").<User, User>chunk(10).reader(reader()).processor(processor()).writer(writer()).build();

}

}

Phương thức reader() được sử dụng để đọc dữ liệu từ CSV file và phương thức writer() được sử dụng để viết dữ liệu trong SQL.

Tiếp theo , chúng ta phải viết một Job Completion Notification Listener class được sử dụng để thông báo sau khi Job hoàn thành.

package com.tutorialspoint.batchservicedemo;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.batch.core.BatchStatus;

import org.springframework.batch.core.JobExecution;

import org.springframework.batch.core.listener.JobExecutionListenerSupport;

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

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Component;

@Component

public class JobCompletionNotificationListener extends JobExecutionListenerSupport {

private static final Logger log = LoggerFactory.getLogger(JobCompletionNotificationListener.class);

private final JdbcTemplate jdbcTemplate;

@Autowired

public JobCompletionNotificationListener(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

@Override

public void afterJob(JobExecution jobExecution) {

if (jobExecution.getStatus() == BatchStatus.COMPLETED) {

log.info("!!! JOB FINISHED !! It's time to verify the results!!");

List<User> results = jdbcTemplate.query(

"SELECT first\_name, last\_name FROM USERS", new RowMapper<User>() {

@Override

public User mapRow(ResultSet rs, int row) throws SQLException {

return new User(rs.getString(1), rs.getString(2));

}

});

for (User person : results) {

log.info("Found <" + person + "> in the database.");

}

}

}

}

Chapter 41.

Spring Boot – Apache Kafka

Apache Kafka là một dự án mã nguồn mở được sử dụng để xuất bản và đăng ký messages dựa trên hệ thống nhắn tin chịu lỗi “the fault-tolerant messaging system”. Nó nhanh , có thể mở rộng và phân phối theo thiết kế.Nếu bạn là một người mới biết Kafka hoặc muốn hiểu biết tốt hơn về nó , hãy theo dõi link sau :

[www.tutorialspoint.com/apache\_kafka](https://www.tutorialspoint.com/apache_kafka/index.htm)

chúng ta sẽ học cách triển khai Apache Kafka trong Spring Boot application.

Đầu tiên , chúng ta cần thêm Spring Kafka dependency trong cấu hình build.

Maven:

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

<version>2.1.0.RELEASE</version>

</dependency>

Gradle:

compile group: 'org.springframework.kafka', name: 'spring-kafka', version: '2.1.0.RELEASE'

Producing Messages

Để tạo messages tới Apache Kafka , chúng ta cần định nghĩa Configuration class cho Producer configuration như sau:

package com.tutorialspoint.kafkademo;

import java.util.HashMap;

import java.util.Map;

import org.apache.kafka.clients.producer.ProducerConfig;

import org.apache.kafka.common.serialization.StringSerializer;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.kafka.core.DefaultKafkaProducerFactory;

import org.springframework.kafka.core.KafkaTemplate;

import org.springframework.kafka.core.ProducerFactory;

@Configuration

public class KafkaProducerConfig {

@Bean

public ProducerFactory<String, String> producerFactory() {

Map<String, Object> configProps = new HashMap<>();

configProps.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

configProps.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class);

configProps.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class);

return new DefaultKafkaProducerFactory<>(configProps);

}

@Bean

public KafkaTemplate<String, String> kafkaTemplate() {

return new KafkaTemplate<>(producerFactory());

}

}

Để xuất bản một tin nhắn , Autowire đối tượng Kafka Template và tạo message dưới đây:

@Autowired

private KafkaTemplate<String, String> kafkaTemplate;

public void sendMessage(String msg) {

kafkaTemplate.send(topicName, msg);

}

Consuming a Message

Để tích hợp messages , chúng ta cần viết một Consumer configuration class dưới đây:

package com.tutorialspoint.kafkademo;

import java.util.HashMap;

import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerConfig;

import org.apache.kafka.common.serialization.StringDeserializer;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.kafka.annotation.EnableKafka;

import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;

import org.springframework.kafka.core.ConsumerFactory;

import org.springframework.kafka.core.DefaultKafkaConsumerFactory;

@EnableKafka

@Configuration

public class KafkaConsumerConfig {

@Bean

public ConsumerFactory<String, String> consumerFactory() {

Map<String, Object> props = new HashMap<>();

props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:2181");

props.put(ConsumerConfig.GROUP\_ID\_CONFIG, "group-id");

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class);

props.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class);

return new DefaultKafkaConsumerFactory<>(props);

}

@Bean

public ConcurrentKafkaListenerContainerFactory<String, String> kafkaListenerContainerFactory() {

ConcurrentKafkaListenerContainerFactory<String, String>

factory = new ConcurrentKafkaListenerContainerFactory<>();

factory.setConsumerFactory(consumerFactory());

return factory;

}

}

Tiếp theo , viết một Listener để listen tới messages.

@KafkaListener(topics = "tutorialspoint", groupId = "group-id")

public void listen(String message) {

System.out.println("Received Messasge in group - group-id: " + message);

}

Gọi phương thức sendMessage() từ ApplicationRunner class chạy phương thức từ main Spring Boot application class và tích hợp message.

Main Spring Boot application.

package com.tutorialspoint.kafkademo;

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

import org.springframework.boot.ApplicationArguments;

import org.springframework.boot.ApplicationRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.kafka.annotation.KafkaListener;

import org.springframework.kafka.core.KafkaTemplate;

@SpringBootApplication

public class KafkaDemoApplication implements ApplicationRunner {

@Autowired

private KafkaTemplate<String, String> kafkaTemplate;

public void sendMessage(String msg) {

kafkaTemplate.send("tutorialspoint", msg);

}

public static void main(String[] args) {

SpringApplication.run(KafkaDemoApplication.class, args);

}

@KafkaListener(topics = "tutorialspoint", groupId = "group-id")

public void listen(String message) {

System.out.println("Received Messasge in group - group-id: " + message);

}

@Override

public void run(ApplicationArguments args) throws Exception {

sendMessage("Hi Welcome to Spring For Apache Kafka");

}

}

Install Apache Kafka:

Chạy cmd

.\bin\windows\zookeeper-server-start.bat.\config\zookeeper.properties

và

.\bin\windows\kafka-server-start.bat .\config\server.properties

Apache Kafka

<https://www.youtube.com/watch?v=EUzH9khPYgs>

Spring Kafka

<https://www.youtube.com/watch?v=80ngRl7RhCw&t=472s>

Chapter 42

Spring Boot – Twilio

Twilio là một ứng dụng bên thứ ba được sử dụng để gửi SMS và tạo gọi thoại từ application của chúng ta.Nó cho phép chúng ta gửi SMS và thực hiện các cuộc gọi thoại theo chương trình.

Bạn sẽ học cách để triển khai việc gửi SMS và tạo gọi thoại bằng cách sử dụng Spring Boot với Twilio.

Lưu ý – chúng ta sử dụng tài khoản Trail trong Twilio để gửi SMS và tạo gọi thoại. bạn có thể tìm hiểu thêm về Twilio tại [www.twilio.com](http://www.twilio.com)

Đầu tiên , chúng ta cần thêm Twilio dependency trong cấu hình build:

Maven:

<dependency>

<groupId>com.twilio.sdk</groupId>

<artifactId>twilio</artifactId>

<version>7.16.1</version>

</dependency>

Gradle:

compile group: "com.twilio.sdk", name:"twilio", version: "7.16.1"

khởi tạo tài khoản Twilio với ACCOUNT\_SID và AUTH\_ID trong khối static :

static {

Twilio.init(ACCOUNT\_SID, AUTH\_ID);

}

Sending SMS

Để gửi SMS , chúng ta cần cung cấp số điện thoại gửi và số điện thoại nhận tới phương thức Message.create() .Nội dung Message chúng ta cần cung cấp cho phương thức Message.creator() :

Message.creator(new PhoneNumber("to-number"), new PhoneNumber("from-number"),

"Message from Spring Boot Application").create();

Main Spring Boot application:

package com.tutorialspoint.smsdemo;

import org.springframework.boot.ApplicationArguments;

import org.springframework.boot.ApplicationRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.twilio.Twilio;

import com.twilio.rest.api.v2010.account.Message;

import com.twilio.type.PhoneNumber;

@SpringBootApplication

public class SmsdemoApplication implements ApplicationRunner {

private final static String ACCOUNT\_SID = "<your-account-sid>";

private final static String AUTH\_ID = "<your-auth-id>";

static {

Twilio.init(ACCOUNT\_SID, AUTH\_ID);

}

public static void main(String[] args) {

SpringApplication.run(SmsdemoApplication.class, args);

}

@Override

public void run(ApplicationArguments arg0) throws Exception {

Message.creator(new PhoneNumber("to-number"), new PhoneNumber("from-number"),

"Message from Spring Boot Application").create();

}

}

Kết quả: bạn sẽ nhận được SMS tới số điện thoại nhận :

Message đã nhận từ số điện thoại nhận:

Sent from your Twilio trail account

- Message from Spring Boot Application

Lưu ý: - trong ví dụ này , chúng tôi sử dụng tài khoản Trail , vì thế bạn nên xác minh số điện thoại trước khi gửi SMS.

Voice Calls:

Để tạo voice calls bằng cách sử dụng Twilio , chúng ta cần gọi phương thức Call.creator() . Chúng ta cần cung cấp số điện thoại nhận và số điện thoại gửi và đoạn voice:

Call.creator(new PhoneNumber("<to-number>"), new PhoneNumber("<from-number>"),

new URI("http://demo.twilio.com/docs/voice.xml")).create();

main Spring Boot application:

package com.tutorialspoint.smsdemo;

import java.net.URI;

import org.springframework.boot.ApplicationArguments;

import org.springframework.boot.ApplicationRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.twilio.Twilio;

import com.twilio.rest.api.v2010.account.Call;

import com.twilio.type.PhoneNumber;

@SpringBootApplication

public class SmsdemoApplication implements ApplicationRunner {

private final static String ACCOUNT\_SID = "<ACCOUNT-SID>";

private final static String AUTH\_ID = "AUTH-ID";

static {

Twilio.init(ACCOUNT\_SID, AUTH\_ID);

}

public static void main(String[] args) {

SpringApplication.run(SmsdemoApplication.class, args);

}

@Override

public void run(ApplicationArguments arg0) throws Exception {

Call.creator(new PhoneNumber("<to-number>"), new PhoneNumber("<from-number>"),

new URI("http://demo.twilio.com/docs/voice.xml")).create();

}

}

Kết quả: bạn sẽ nhận được gọi thoại tới số điện thoại nhận từ Twilio

<https://demo.twilio.com/docs/voice.xml>

lưu ý – trong ví dụ này , chúng tôi sử dụng tài khoản Trail , vì thế bạn nên xác minh số điện thoại trước khi tạo gọi thoại.

<https://www.youtube.com/watch?v=OuBttmaPlhM>

Chapter 43.

Spring Boot – Unit Test Cases

Unit Testing là một trong những testing được hoàn thành bởi các developers để chắc chắn chức năng đơn vị hoặc các thành phần riêng lẻ hoạt động tốt.

Chúng ta sẽ học cách viết một unit test case bằng cách sử dụng Mockito và Web Controller.

Mockito.

Cho việc injecting Mockito Mocks tới Spring Beans , chúng ta cần thêm Mockito-core dependency trong cấu hình build.

Maven:

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>2.13.0</version>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

Gradle:

compile group: 'org.mockito', name: 'mockito-core', version: '2.13.0'

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

code viết cho Service class chứa một phương thức trả về giá trị String :

package com.tutorialspoint.mockitodemo;

import org.springframework.stereotype.Service;

@Service

public class ProductService {

public String getProductName() {

return "Honey";

}

}

Bây giờ , inject ProductService class trong Service khác:

package com.tutorialspoint.mockitodemo;

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

import org.springframework.stereotype.Service;

@Service

public class OrderService {

@Autowired

ProductService productService;

public OrderService(ProductService productService) {

this.productService = productService;

}

public String getProductName() {

return productService.getProductName();

}

}

Main Spring Boot application:

package com.tutorialspoint.mockitodemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MockitoDemoApplication {

public static void main(String[] args) {

SpringApplication.run(MockitoDemoApplication.class, args);

}

}

Sau đó , cấu hình Application context cho tests . @Profile(“test”) annotation được sử dụng để cấu hình class khi Test cases chạy:

package com.tutorialspoint.mockitodemo;

import org.mockito.Mockito;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.context.annotation.Profile;

@Profile("test")

@Configuration

public class ProductServiceTestConfiguration {

@Bean

@Primary

public ProductService productService() {

return Mockito.mock(ProductService.class);

}

}

Bây giờ , bạn có thể viết một Unit Test case cho Order Service chứa trong package

Src/test/resources :

package com.tutorialspoint.mockitodemo;

import org.junit.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.ActiveProfiles;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@SpringBootTest

@ActiveProfiles("test")

@RunWith(SpringJUnit4ClassRunner.class)

public class MockitoDemoApplicationTests {

@Autowired

private OrderService orderService;

@Autowired

private ProductService productService;

@Test

public void whenUserIdIsProvided\_thenRetrievedNameIsCorrect() {

Mockito.when(productService.getProductName()).thenReturn("Mock Product Name");

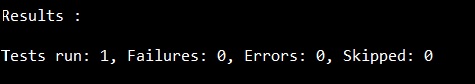
String testName = orderService.getProductName();

Assert.assertEquals("Mock Product Name", testName);

}

}

Kết quả:



Gradle:



Chapter 44

Spring Boot – Rest Controller Unit Test

Spring Boot cung cấp một cách dễ dàng để viết một Unit Test cho Rest Controller. Với sự giúp đỡ của SpringJUnit4ClassRunner và MockMvc , chúng ta có thể tạo một web application context để viết Unit Test cho Rest Controller.

Unit Tests nên được viết trong thư mục src/test/java và đường dẫn resources cho việc viết một test nên được đặt trong thư mục src/test/resources.

Cho việc viết một Unit Test , chúng ta cần thêm Spring Boot Starter Test dependency trong cấu hình build:

<dependency>

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

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

<scope>test</scope>

</dependency>

Gradle:

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

Trước khi viết một Test case , chúng ta nên build RESTful web services trước.

Để biết thêm thông tin việc build RESTful web services , xem lại bài viết RESTful web services:

Writing a Unit Test for REST Controller.

Trong mục này , chúng ta sẽ học cách viết một Unit Test cho REST Controller:

Đầu tiên , chúng ta cần tạo Abstract class được sử dụng để tạo web application context bằng cách sử dụng MockMvc và định nghĩa mapToJson() và mapFromJson() để convert đối tượng Java thành chuỗi JSON và convert chuỗi JSON thành đối tượng JAVA.

package com.tutorialspoint.demo;

import java.io.IOException;

import org.junit.runner.RunWith;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import org.springframework.test.context.web.WebAppConfiguration;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.web.context.WebApplicationContext;

import com.fasterxml.jackson.core.JsonParseException;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

@RunWith(SpringJUnit4ClassRunner.class)

@SpringBootTest(classes = DemoApplication.class)

@WebAppConfiguration

public abstract class AbstractTest {

protected MockMvc mvc;

@Autowired

WebApplicationContext webApplicationContext;

protected void setUp() {

mvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();

}

protected String mapToJson(Object obj) throws JsonProcessingException {

ObjectMapper objectMapper = new ObjectMapper();

return objectMapper.writeValueAsString(obj);

}

protected <T> T mapFromJson(String json, Class<T> clazz)

throws JsonParseException, JsonMappingException, IOException {

ObjectMapper objectMapper = new ObjectMapper();

return objectMapper.readValue(json, clazz);

}

}

Tiếp theo, viết một class kế thừa AbstractTest class và viết một Unit Test cho mỗi phương thức GET , POST, PUT, DELETE.

Code cho GET API Test case dưới đây: API này để xem danh sách các sản phẩm “products”:

@Test

public void getProductsList() throws Exception {

String uri = "/products";

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)

.accept(MediaType.APPLICATION\_JSON\_VALUE)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

Product[] productlist = super.mapFromJson(content, Product[].class);

assertTrue(productlist.length > 0);

}

Code cho POST API Test case được đưa ra dưới đây. API này để tạo một product:

@Test

public void createProduct() throws Exception {

String uri = "/products";

Product product = new Product();

product.setId("3");

product.setName("Ginger");

String inputJson = super.mapToJson(product);

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.post(uri)

.contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(201, status);

String content = mvcResult.getResponse().getContentAsString();

assertEquals(content, "Product is created successfully");

}

Code cho PUT API Test case được đưa ra dưới đây. API này sẽ update product đã tồn tại:

@Test

public void updateProduct() throws Exception {

String uri = "/products/2";

Product product = new Product();

product.setName("Lemon");

String inputJson = super.mapToJson(product);

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.put(uri)

.contentType(MediaType.APPLICATION\_JSON\_VALUE).content(inputJson)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

assertEquals(content, "Product is updated successsfully");

}

Code cho Delete API Test case được đưa ra dưới đây . API này sẽ xóa các product tồn tại:

@Test

public void deleteProduct() throws Exception {

String uri = "/products/2";

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.delete(uri)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

assertEquals(content, "Product is deleted successsfully");

}

Code đầy đủ : full Controller Test:

package com.tutorialspoint.demo;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertTrue;

import org.junit.Before;

import org.junit.Test;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import com.tutorialspoint.demo.model.Product;

public class ProductServiceControllerTest extends AbstractTest {

@Override

@Before

public void setUp() {

super.setUp();

}

@Test

public void getProductsList() throws Exception {

String uri = "/products";

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)

.accept(MediaType.APPLICATION\_JSON\_VALUE)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

Product[] productlist = super.mapFromJson(content, Product[].class);

assertTrue(productlist.length > 0);

}

@Test

public void createProduct() throws Exception {

String uri = "/products";

Product product = new Product();

product.setId("3");

product.setName("Ginger");

String inputJson = super.mapToJson(product);

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.post(uri)

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.content(inputJson)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(201, status);

String content = mvcResult.getResponse().getContentAsString();

assertEquals(content, "Product is created successfully");

}

@Test

public void updateProduct() throws Exception {

String uri = "/products/2";

Product product = new Product();

product.setName("Lemon");

String inputJson = super.mapToJson(product);

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.put(uri)

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.content(inputJson)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

assertEquals(content, "Product is updated successsfully");

}

@Test

public void deleteProduct() throws Exception {

String uri = "/products/2";

MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.delete(uri)).andReturn();

int status = mvcResult.getResponse().getStatus();

assertEquals(200, status);

String content = mvcResult.getResponse().getContentAsString();

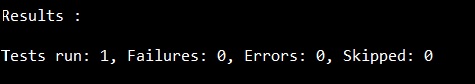
assertEquals(content, "Product is deleted successsfully");

}

}

Kết quả:

Maven:



Gradle:



Chapter 45.

Spring Boot – Database Handling

Spring Boot cung cấp một hỗ trợ tốt để tạo một DataSource cho Database. Chúng ta ko cần viết bất kể dòng code phụ nào để tạo một DataSource trong Spring Boot. Chỉ cần thêm dependencites và thực hiện chi tiết cấu hình đủ để tạo một DataSource và kết nối Database.

Chúng ta sẽ học cách sử dụng Spring Boot JDBC driver connection để kết nối database.

Đầu tiên , chúng ta cần thêm Spring Boot Starter JDBC dependency trong cấu hình build .

Maven:

<dependency>

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

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

</dependency>

Gradle:

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

Connect to H2 Database:

Để kết nối tới H2 database , chúng ta cần thêm H2 database dependency trong cấu hình build:

Maven:

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

Gradle:

compile('com.h2database:h2')

Chúng ta cần tạo file schema.sql và data.sql chứa trong đường dẫn thư mục src/main/resources để kết nối H2 database.

Schema.sql file:

CREATE TABLE PRODUCT (ID INT PRIMARY KEY, PRODUCT\_NAME VARCHAR(25));

Data.sql file:

INSERT INTO PRODUCT (ID,PRODUCT\_NAME) VALUES (1,'Honey');

INSERT INTO PRODUCT (ID,PRODUCT\_NAME) VALUES (2,'Almond');

Connect MySQL

Để kết nối MySQL database , chúng ta cần thêm MySQL dependency trong cấu hình build:

Maven:

<dependency>

<groupId>mysql</groupId>

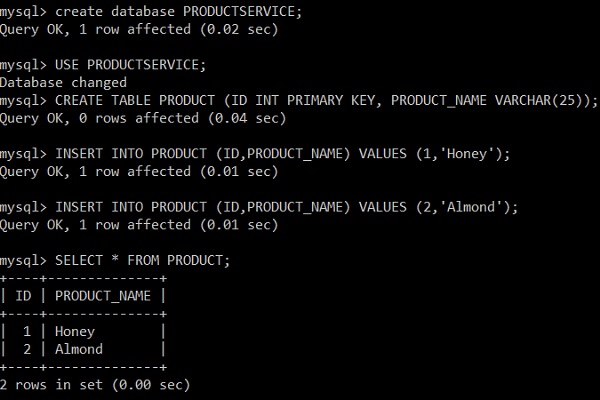
<artifactId>mysql-connector-java</artifactId>

</dependency>

Gradle:

compile('mysql:mysql-connector-java')

Bây giờ , tạo database và bảng trong MySQL



Cấu hình: thêm properties trong application.properties:

spring.datasource.driverClassName = com.mysql.jdbc.Driver

spring.datasource.url = jdbc:mysql://localhost:3306/PRODUCTSERVICE?autoreconnect = true

spring.datasource.username = root

spring.datasource.password = root

spring.datasource.testOnBorrow = true

spring.datasource.testWhileIdle = true

spring.datasource.timeBetweenEvictionRunsMillis = 60000

spring.datasource.minEvictableIdleTimeMillis = 30000

spring.datasource.validationQuery = SELECT 1

spring.datasource.max-active = 15

spring.datasource.max-idle = 10

spring.datasource.max-wait = 8000

YAML:

spring:

datasource:

driverClassName: com.mysql.jdbc.Driver

url: "jdbc:mysql://localhost:3306/PRODUCTSERVICE?autoreconnect=true"

username: "root"

password: "root"

testOnBorrow: true

testWhileIdle: true

timeBetweenEvictionRunsMillis: 60000

minEvictableIdleTimeMillis: 30000

validationQuery: SELECT 1

max-active: 15

max-idle: 10

max-wait: 8000

Connect Redis

Redis là một database mã nguồn mở được sử dụng để lưu trữ cấu trúc dữ liệu trong bộ nhớ. Để kết nối tới Redis database trong Spring Boot application, chúng ta cần thêm Redis dependency trong cấu hình build:

Maven:

<dependency>

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

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

</dependency>

Gradle:

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

Cho việc kết nối Redis , chúng ta cần sử dụng RedisTemplate. RedisTemplate , chúng ta cần cung cấp JedisConnectionFactory chi tiết:

@Bean

JedisConnectionFactory jedisConnectionFactory() {

JedisConnectionFactory jedisConFactory = new JedisConnectionFactory();

jedisConFactory.setHostName("localhost");

jedisConFactory.setPort(6000);

jedisConFactory.setUsePool(true);

return jedisConFactory;

}

@Bean

public RedisTemplate<String, Object> redisTemplate() {

RedisTemplate<String, Object> template = new RedisTemplate<>();

template.setConnectionFactory(jedisConnectionFactory());

template.setKeySerializer(new StringRedisSerializer());

template.setHashKeySerializer(new StringRedisSerializer());

template.setHashValueSerializer(new StringRedisSerializer());

template.setValueSerializer(new StringRedisSerializer());

return template;

}

Bây giờ , autowire RedisTemplate class và access dữ liệu từ Redis database:

@Autowired

RedisTemplate<String, Object> redis;

Map<Object,Object> datalist = redis.opsForHash().entries(“Redis\_code\_index\_key”);

JDBCTemplate:

Để chấp nhận Relational Database bằng cách sử dụng JdbcTemplate trong Spring Boot application, chúng ta cần thêm Spring Boot Starter JDBC dependency trong cấu hình build:

Sau đó , nếu bạn @Autowired JdbcTemplate class , Spring Boot tự động kết nối Database và sets Datasource cho đối tượng JdbcTemplate.

@Autowired

JdbcTemplate jdbcTemplate;

Collection<Map<String, Object>> rows = jdbc.queryForList("SELECT QUERY");

@Repository annotation nên được thêm vào class . @Repository annotation được sử dụng để tạo database repository cho Spring Boot application.

@Repository

public class ProductServiceDAO {

}

Multiple DataSource

Chúng ta có thể giữ “n” số Datasources trong một Spring Boot application duy nhất. Ví dụ được đưa ra dưới đây sẽ chỉ cách tạo nhiều hơn 1 datasource trong Spring Boot application. Bây giờ , thêm cấu hình chi tiết 2 datasource trong application.properties file:

Cho việc cấu hình application.properties:

spring.dbProductService.driverClassName = com.mysql.jdbc.Driver

spring.dbProductService.url = jdbc:mysql://localhost:3306/PRODUCTSERVICE?autoreconnect = true

spring.dbProductService.username = root

spring.dbProductService.password = root

spring.dbProductService.testOnBorrow = true

spring.dbProductService.testWhileIdle = true

spring.dbProductService.timeBetweenEvictionRunsMillis = 60000

spring.dbProductService.minEvictableIdleTimeMillis = 30000

spring.dbProductService.validationQuery = SELECT 1

spring.dbProductService.max-active = 15

spring.dbProductService.max-idle = 10

spring.dbProductService.max-wait = 8000

spring.dbUserService.driverClassName = com.mysql.jdbc.Driver

spring.dbUserService.url = jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect = true

spring.dbUserService.username = root

spring.dbUserService.password = root

spring.dbUserService.testOnBorrow = true

spring.dbUserService.testWhileIdle = true

spring.dbUserService.timeBetweenEvictionRunsMillis = 60000

spring.dbUserService.minEvictableIdleTimeMillis = 30000

spring.dbUserService.validationQuery = SELECT 1

spring.dbUserService.max-active = 15

spring.dbUserService.max-idle = 10

spring.dbUserService.max-wait = 8000

YAML:

spring:

dbProductService:

driverClassName: com.mysql.jdbc.Driver

url: "jdbc:mysql://localhost:3306/PRODUCTSERVICE?autoreconnect=true"

password: "root"

username: "root"

testOnBorrow: true

testWhileIdle: true

timeBetweenEvictionRunsMillis: 60000

minEvictableIdleTimeMillis: 30000

validationQuery: SELECT 1

max-active: 15

max-idle: 10

max-wait: 8000

dbUserService:

driverClassName: com.mysql.jdbc.Driver

url: "jdbc:mysql://localhost:3306/USERSERVICE?autoreconnect=true"

password: "root"

username: "root"

testOnBorrow: true

testWhileIdle: true

timeBetweenEvictionRunsMillis: 60000

minEvictableIdleTimeMillis: 30000

validationQuery: SELECT 1

max-active: 15

max-idle: 10

max-wait: 8000

Bây giờ , tạo một Configuration class để tạo một DataSource và JdbcTemplate cho multiple datasources:

import javax.sql.DataSource;

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

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

import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Primary;

import org.springframework.jdbc.core.JdbcTemplate;

@Configuration

public class DatabaseConfig {

@Bean(name = "dbProductService")

@ConfigurationProperties(prefix = "spring.dbProductService")

@Primary

public DataSource createProductServiceDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "dbUserService")

@ConfigurationProperties(prefix = "spring.dbUserService")

public DataSource createUserServiceDataSource() {

return DataSourceBuilder.create().build();

}

@Bean(name = "jdbcProductService")

@Autowired

public JdbcTemplate createJdbcTemplate\_ProductService(@Qualifier("dbProductService") DataSource productServiceDS) {

return new JdbcTemplate(productServiceDS);

}

@Bean(name = "jdbcUserService")

@Autowired

public JdbcTemplate createJdbcTemplate\_UserService(@Qualifier("dbUserService") DataSource userServiceDS) {

return new JdbcTemplate(userServiceDS);

}

}

Sau đó , autowire đối tượng JDBCTemplate bằng cách sử dụng @Qualifier annotation:

@Qualifier("jdbcProductService")

@Autowired

JdbcTemplate jdbcTemplate;

@Qualifier("jdbcUserService")

@Autowired

JdbcTemplate jdbcTemplate;

Chapter 46.

Spring Boot – Securing Web Applications

Nếu một dependency Spring Boot Security được thêm vào đường dẫn , Spring Boot application tự động bắt buộc Basic Authentication cho toàn bộ HTTP Endpoints. Điểm Endpoint “/” và “/home” không bắt buộc bất kỳ authentication nào. Toàn bộ Endpoints khác bắt buộc authentication.

Dành cho việc thêm Spring Boot Security cho Spring Boot application, chúng ta cần thêm Spring Boot Starter Security dependency trong cấu hình build.

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile("org.springframework.boot:spring-boot-starter-security")

Securing a Web application:

Đầu tiên , tạo một web application chưa xác thực bằng cách sử dụng Thymeleaf templates.

Sau đó , tạo một home.html chứa trong thư mục src/main/resources/templates

<!DOCTYPE html>

<html xmlns = "http://www.w3.org/1999/xhtml"

xmlns:th = "http://www.thymeleaf.org"

xmlns:sec = "http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>

<title>Spring Security Example</title>

</head>

<body>

<h1>Welcome!</h1>

<p>Click <a th:href = "@{/hello}">here</a> to see a greeting.</p>

</body>

</html>

/hello định nghĩa trong HTML file bằng cách sử dụng thymeleaf templates.

Tạo hello.html chứa trong thư mục src/main/resources/templates

<!DOCTYPE html>

<html xmlns = "http://www.w3.org/1999/xhtml"

xmlns:th = "http://www.thymeleaf.org"

xmlns:sec = "http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>

<title>Hello World!</title>

</head>

<body>

<h1>Hello world!</h1>

</body>

</html>

Bây giờ , chúng ta cần setup Spring MVC – View Controller cho home và hello views

Tạo một MVC configuration file kế thừa WebMvcConfigurerAdapter.

package com.tutorialspoint.websecuritydemo;

import org.springframework.context.annotation.Configuration;

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

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

@Configuration

public class MvcConfig extends WebMvcConfigurerAdapter {

@Override

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/home").setViewName("home");

registry.addViewController("/").setViewName("home");

registry.addViewController("/hello").setViewName("hello");

registry.addViewController("/login").setViewName("login");

}

}

Bây giờ , thêm Spring Boot Starter security dependency để cấu hình build:

Maven:

<dependency>

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

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

</dependency>

Gradle:

compile("org.springframework.boot:spring-boot-starter-security")

Bây giờ , tạo một Web Security Configuration file , được sử dụng để bảo mất application của bạn để cho phép truy cập HTTP Endpoints bằng cách sử dụng basic authentication.

package com.tutorialspoint.websecuritydemo;

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

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/", "/home").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.permitAll();

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.inMemoryAuthentication()

.withUser("user").password("password").roles("USER");

}

}

Bây giờ , tạo login.html chứa trong thư mục src/main/resources để chấp nhận user truy cập HTTP Endpoint trên màn hình login:

<!DOCTYPE html>

<html xmlns = "http://www.w3.org/1999/xhtml" xmlns:th = "http://www.thymeleaf.org"

xmlns:sec = "http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>

<title>Spring Security Example </title>

</head>

<body>

<div th:if = "${param.error}">

Invalid username and password.

</div>

<div th:if = "${param.logout}">

You have been logged out.

</div>

<form th:action = "@{/login}" method = "post">

<div>

<label> User Name : <input type = "text" name = "username"/> </label>

</div>

<div>

<label> Password: <input type = "password" name = "password"/> </label>

</div>

<div>

<input type = "submit" value = "Sign In"/>

</div>

</form>

</body>

</html>

Cuối cùng , cập nhật hello.html để cho phép user Sign-out từ appication và màn hình với username hiện tại:

<!DOCTYPE html>

<html xmlns = "http://www.w3.org/1999/xhtml" xmlns:th = "http://www.thymeleaf.org"

xmlns:sec = "http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">

<head>

<title>Hello World!</title>

</head>

<body>

<h1 th:inline = "text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>

<form th:action = "@{/logout}" method = "post">

<input type = "submit" value = "Sign Out"/>

</form>

</body>

</html>

Code main:

package com.tutorialspoint.websecuritydemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class WebsecurityDemoApplication {

public static void main(String[] args) {

SpringApplication.run(WebsecurityDemoApplication.class, args);

}

}

Chapter 47.

Spring Boot – Oauth2 with JWT

Chúng ta sẽ học chi tiết về cơ chế Spring Boot Security và Oauth2 with JWT.

Authorization Server

Authorization Server là một thành phần kiến trúc tối cao cho Web API Security.

Authorization Server đóng vai trò điểm xác thực ủy quyền tập trung cho phép ứng dụng của bạn và HTTP Endpoints xác định tính năng của application của bạn.

Resource Server

Resource Server là một ứng dụng cung cấp token truy cập tới clients để truy cập Resource Server HTTP Endpoints. Nó là một collection của thư viện chứa HTTP Endpoints , static resources và Dynamic web pages.

Oauth2

Oauth2 là một framework authorization kích hoạt ứng dụng Web Security để truy cập resources từ client. Để build một ứng dụng Oauth2 , chúng ta cần tập trung vào Grant Type (Authorization code) mã ủy quyền , Client ID và Client secret.

JWT Token

JWT Token là một JSON Web Token , được sử dụng để đại diện cho các yêu cầu được bảo mật giữa 2 bên. Bạn có thể tìm hiểu thêm về JWT token tại [www.jwt.io](http://www.jwt.io)

Chúng ta sẽ build một Oauth2 application kích hoạt sử dụng Authorization Server, Resource Server với trợ giúp của JWT Token.

Bạn có thể sử dụng các bước dưới đây để triển khai Spring Boot Security với JWT token bằng cách truy cập database.

Đầu tiên , chúng ta cần thêm dependencies sau đây:

Maven:

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework.security.oauth</groupId>

<artifactId>spring-security-oauth2</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-jwt</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

Gradle:

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

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

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

testCompile('org.springframework.security:spring-security-test')

compile("org.springframework.security.oauth:spring-security-oauth2")

compile('org.springframework.security:spring-security-jwt')

compile("org.springframework.boot:spring-boot-starter-jdbc")

compile("com.h2database:h2:1.4.191")

Spring Boot Starter Security – triển khai Spring Security

Spring Security Oauth2 – triển khai kết cấu Oauth2 để kích hoạt Authorization Server và Resource Server

Spring Security JWT – Generates JWT Token cho Web security

Spring Boot Starter JDBC – truy cập database để đảm bảo user có sẵn hay không

Spring Boot Starter Web – Writes HTTP endpoints

H2 Database – lưu trữ thông tin user cho authentication và authorization

Bây giờ , trong main Spring Boot application , thêm @EnableAuthorizationServer và @EnableResourceServer annotation để đóng vai trò như một Auth server và Resource Server trong cùng một application.

Bạn có thể sử dụng code dưới đây để viết một HTTP endpoint đơn giản để truy cập API với Spring Security bằng cách sử dụng JWT Token.

package com.tutorialspoint.websecurityapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

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

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

@SpringBootApplication

@EnableAuthorizationServer

@EnableResourceServer

@RestController

public class WebsecurityappApplication {

public static void main(String[] args) {

SpringApplication.run(WebsecurityappApplication.class, args);

}

@RequestMapping(value = "/products")

public String getProductName() {

return "Honey";

}

}

Sử dụng code dưới đây để định nghĩa POJO class để lưu trữ thông tin User cho authentication.

package com.tutorialspoint.websecurityapp;

import java.util.ArrayList;

import java.util.Collection;

import org.springframework.security.core.GrantedAuthority;

public class UserEntity {

private String username;

private String password;

private Collection<GrantedAuthority> grantedAuthoritiesList = new ArrayList<>();

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public Collection<GrantedAuthority> getGrantedAuthoritiesList() {

return grantedAuthoritiesList;

}

public void setGrantedAuthoritiesList(Collection<GrantedAuthority> grantedAuthoritiesList) {

this.grantedAuthoritiesList = grantedAuthoritiesList;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

}

Bây giờ , sử dụng code dưới đây và định nghĩa CustomUser class kế thừa User class của thư viện security cho Spring Boot authentication.

package com.tutorialspoint.websecurityapp;

import org.springframework.security.core.userdetails.User;

public class CustomUser extends User {

private static final long serialVersionUID = 1L;

public CustomUser(UserEntity user) {

super(user.getUsername(), user.getPassword(), user.getGrantedAuthoritiesList());

}

}

Bạn có thể tạo @Repository class để đọc thông tin User từ database và gửi nó tới CustomUser service và thêm Granted Authority (quyền truy cập) ROLE\_SYSTEMADMIN

package com.tutorialspoint.websecurityapp;

import java.sql.ResultSet;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

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

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.stereotype.Repository;

@Repository

public class OAuthDao {

@Autowired

private JdbcTemplate jdbcTemplate;

public UserEntity getUserDetails(String username) {

Collection<GrantedAuthority> grantedAuthoritiesList = new ArrayList<>();

String userSQLQuery = "SELECT \* FROM USERS WHERE USERNAME=?";

List<UserEntity> list = jdbcTemplate.query(userSQLQuery, new String[] { username },

(ResultSet rs, int rowNum) -> {

UserEntity user = new UserEntity();

user.setUsername(username);

user.setPassword(rs.getString("PASSWORD"));

return user;

});

if (list.size() > 0) {

GrantedAuthority grantedAuthority = new SimpleGrantedAuthority("ROLE\_SYSTEMADMIN");

grantedAuthoritiesList.add(grantedAuthority);

list.get(0).setGrantedAuthoritiesList(grantedAuthoritiesList);

return list.get(0);

}

return null;

}

}

Bạn có thể tạo một CustomUserDetail Service class kế thừa UserDetailsService để gọi DAO repository class:

package com.tutorialspoint.websecurityapp;

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

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

@Service

public class CustomDetailsService implements UserDetailsService {

@Autowired

OAuthDao oauthDao;

@Override

public CustomUser loadUserByUsername(final String username) throws UsernameNotFoundException {

UserEntity userEntity = null;

try {

userEntity = oauthDao.getUserDetails(username);

CustomUser customUser = new CustomUser(userEntity);

return customUser;

} catch (Exception e) {

e.printStackTrace();

throw new UsernameNotFoundException("User " + username + " was not found in the database");

}

}

}

Tiếp theo , tạo một @Configuration class để kích hoạt Web Security và định nghĩa Password encoder (BcryptPasswordEncoder) , và định nghĩa AuthenticationManager bean. Security configuration class nên kế thừa WebSecurityConfigurerAdapter :

package com.tutorialspoint.websecurityapp;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

private CustomDetailsService customDetailsService;

@Bean

public PasswordEncoder encoder() {

return new BCryptPasswordEncoder();

}

@Override

@Autowired

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(customDetailsService).passwordEncoder(encoder());

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().anyRequest().authenticated().and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.NEVER);

}

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring();

}

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

Bây giờ , định nghĩa Oauth2 configuration class để thêm Client ID , Client Secret , định nghĩa JwtAccessTokenConverter , Private key và public key cho token signer key và verifier key , và cấu hình ClientDetailsServiceConfigurer cho giới hạn Token validity.

package com.tutorialspoint.websecurityapp;

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

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;

import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;

import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;

import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;

import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

@Configuration

public class OAuth2Config extends AuthorizationServerConfigurerAdapter {

private String clientid = "tutorialspoint";

private String clientSecret = "my-secret-key";

private String privateKey = "private key";

private String publicKey = "public key";

@Autowired

@Qualifier("authenticationManagerBean")

private AuthenticationManager authenticationManager;

@Bean

public JwtAccessTokenConverter tokenEnhancer() {

JwtAccessTokenConverter converter = new JwtAccessTokenConverter();

converter.setSigningKey(privateKey);

converter.setVerifierKey(publicKey);

return converter;

}

@Bean

public JwtTokenStore tokenStore() {

return new JwtTokenStore(tokenEnhancer());

}

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.authenticationManager(authenticationManager).tokenStore(tokenStore())

.accessTokenConverter(tokenEnhancer());

}

@Override

public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {

security.tokenKeyAccess("permitAll()").checkTokenAccess("isAuthenticated()");

}

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory().withClient(clientid).secret(clientSecret).scopes("read", "write")

.authorizedGrantTypes("password", "refresh\_token").accessTokenValiditySeconds(20000)

.refreshTokenValiditySeconds(20000);

}

}

Bây giờ , tạo một Private key và public key bằng cách sử dụng openssl

Bạn có thể sử dụng câu lệnh dưới đây để generating private key:

openssl genrsa -out jwt.pem 2048

openssl rsa -in jwt.pem

bạn có thể sử dụng public key generation sử dụng câu lệnh sau:

openssl rsa -in jwt.pem -pubout

cho phiên bản Spring Boot 1.5 trở đi , thêm thuộc tính sau trong application.properties để định nghĩa Oauth2 Resource filter:

security.oauth2.resource.filter-order=3

YAML:

security:

oauth2:

resource:

filter-order: 3

Bây giờ , tạo schema.sql và data.sql file chứa trong đường dẫn resources src/main/resources/directory để kết nối application tới H2 database.

Schema.sql:

CREATE TABLE USERS (ID INT PRIMARY KEY, USERNAME VARCHAR(45), PASSWORD VARCHAR(60));

Data.sql:

INSERT INTO USERS (ID, USERNAME,PASSWORD) VALUES (

1, 'tutorialspoint@gmail.com','$2a$08$fL7u5xcvsZl78su29x1ti.dxI.9rYO8t0q5wk2ROJ.1cdR53bmaVG');

INSERT INTO USERS (ID, USERNAME,PASSWORD) VALUES (

2, 'myemail@gmail.com','$2a$08$fL7u5xcvsZl78su29x1ti.dxI.9rYO8t0q5wk2ROJ.1cdR53bmaVG');

Lưu ý – password nên được lưu trữ dưới định dạng Bcrypt Encoder trong bảng database:

Kết quả:

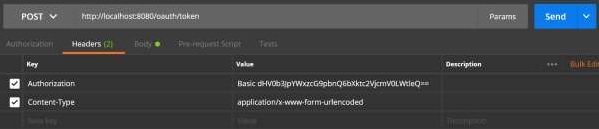
POST method URL trong Postman để lấy OAUTH2 token:

<http://localhost:8080/oauth/token>

thêm Request Headers:

Authorization – Basic Auth với Client id và Client secret

Content Type – application/x-www-form-urlencoded

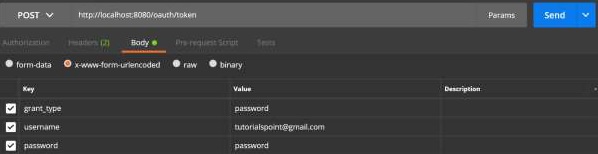


Thêm Request Parameters:

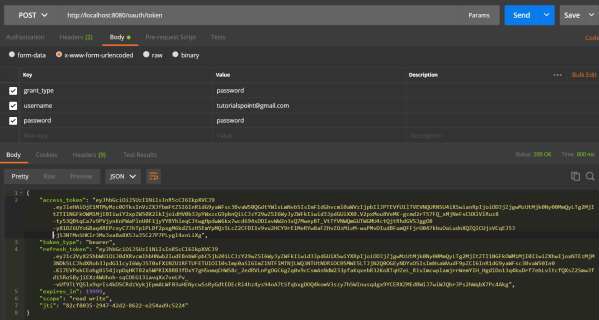
Grant\_type = password

Username = username của bạn

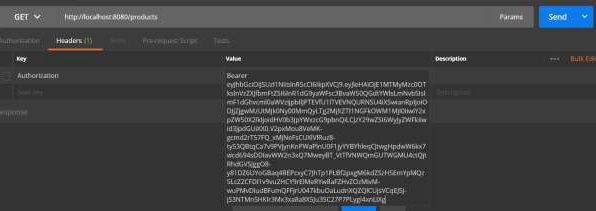
Password = password của bạn



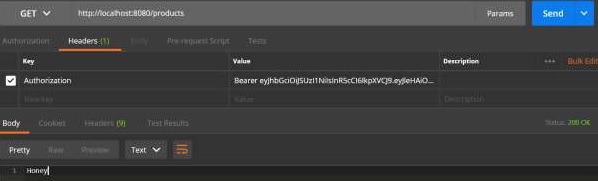
Truy cập API và lấy access\_token:



Truy cập Resource Server API với Bearer + access token trong Request Header:



Kết quả:



Chapter 48.

Spring Boot – Google Cloud Platform

Google Cloud Platform (nền tảng đám mây của Google) cung cấp một dịch vụ điện toán đám mây chạy trên Spring Boot application trong môi trường cloud (đám mây). Chúng ta sẽ học cách deploy Spring Boot application trong nền tảng công nghệ ứng dụng GCP:

Đầu tiên , download thư viện Gradle build cho Spring Boot application từ Spring Initializer page:

Trong build.gradle file , thêm Google Cloud appengine plugin và appengine vào đường dẫn dependency:

buildscript {

ext {

springBootVersion = '1.5.9.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

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

classpath 'com.google.cloud.tools:appengine-gradle-plugin:1.3.3'

}

}

apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'org.springframework.boot'

apply plugin: 'com.google.cloud.tools.appengine'

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')

}

Bây giờ , viết một HTTP Endpoint đơn giản và trả về một chuỗi thành công:

package com.tutorialspoint.appenginedemo;

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 AppengineDemoApplication {

public static void main(String[] args) {

SpringApplication.run(AppengineDemoApplication.class, args);

}

@RequestMapping(value = "/")

public String success() {

return "APP Engine deployment success";

}

}

Tiếp theo , thêm app.yml file chứa trong src/main/appengine directory:

runtime: java

env: flex

handlers:

- url: /.\*

script: this field is required, but ignored

bây giờ , đến Google Cloud console và chọn Activate Google Cloud shell tại đầu trang:

di chuyển source files và Gradle file trong thư mục home của google cloud machine của bạn bằng cách sử dụng google cloud shell:

bây giờ , thực thi câu lệnh Gradle appengineDeplot và nó sẽ deploy application của bạn lên Google Cloud appengine

lưu ý – GCP nên được bật thanh toán và trước khi deploy application của bạn lên appengine, bạn nên tạo nền tảng appengine trong GCP.

Sẽ mất vài phút để deploy application của bạn tới nền tàng GCP appengine.

Sau khi build successful , bạn có thể nhìn thấy Service URL trong cửa sổ console:

Và output:

App Engine Development Success

Google Cloud SQL:

Để kết nối tới Google Cloud SQL tới Spring Boot application , bạn nên thêm thuộc tính dưới đây trong application.properties:

jdbc:mysql://google/<DATABASE-NAME>?cloudSqlInstance = <GOOGLE\_CLOUD\_SQL\_INSTANCE\_NAME> &socketFactory = com.google.cloud.sql.mysql.SocketFactory&user = <USERNAME>&password = <PASSWORD>

lưu ý – Spring Boot application và Google Cloud SQL nên cùng một GCP project.

Application.properties:

spring.dbProductService.driverClassName = com.mysql.jdbc.Driver

spring.dbProductService.url = jdbc:mysql://google/PRODUCTSERVICE?cloudSqlInstance = springboot-gcp-cloudsql:asia-northeast1:springboot-gcp-cloudsql-instance&socketFactory = com.google.cloud.sql.mysql.SocketFactory&user = root&password = rootspring.dbProductService.username = root

spring.dbProductService.password = root

spring.dbProductService.testOnBorrow = true

spring.dbProductService.testWhileIdle = true

spring.dbProductService.timeBetweenEvictionRunsMillis = 60000

spring.dbProductService.minEvictableIdleTimeMillis = 30000

spring.dbProductService.validationQuery = SELECT 1

spring.dbProductService.max-active = 15

spring.dbProductService.max-idle = 10

spring.dbProductService.max-wait = 8000

YAML:

spring:

datasource:

driverClassName: com.mysql.jdbc.Driver

url: "jdbc:mysql://google/PRODUCTSERVICE?cloudSqlInstance=springboot-gcp-cloudsql:asia-northeast1:springboot-gcp-cloudsql-instance&socketFactory=com.google.cloud.sql.mysql.SocketFactory&user=root&password=root"

password: "root"

username: "root"

testOnBorrow: true

testWhileIdle: true

validationQuery: SELECT 1

max-active: 15

max-idle: 10

max-wait: 8000

Chapter 49

Spring Boot – Google Oauth2 Sign-in

Chúng ta sẽ học về cách thêm Google Oauth2 Sign-in bằng cách sử dụng Spring Boot application với Gradle build:

Đầu tiên , thêm Spring Boot Oauth2 security dependency trong cấu hình 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.projects'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = 1.8

repositories {

mavenCentral()

}

dependencies {

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

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

compile('org.springframework.security.oauth:spring-security-oauth2')

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

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

}

Bây giờ , thêm HTTP Endpoint để đọc User Principal từ Google sau khi authenticating từ Spring Boot trong main Spring Boot application class:

package com.tutorialspoint.projects.googleservice;

import java.security.Principal;

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 GoogleserviceApplication {

public static void main(String[] args) {

SpringApplication.run(GoogleserviceApplication.class, args);

}

@RequestMapping(value = "/user")

public Principal user(Principal principal) {

return principal;

}

}

Bây giờ , viết một Configuration để kích hoạt Oauth2SSO cho web security và xóa authentication cho index.html:

package com.tutorialspoint.projects.googleservice;

import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableOAuth2Sso

public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf()

.disable()

.antMatcher("/\*\*")

.authorizeRequests()

.antMatchers("/", "/index.html")

.permitAll()

.anyRequest()

.authenticated();

}

}

Tiếp theo , thêm index.html file chứa trong static resources và thêm link để redirect tới user HTTP Endpoint để đọc Google user Principal:

<!DOCTYPE html>

<html>

<head>

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

<title>Insert title here</title>

</head>

<body>

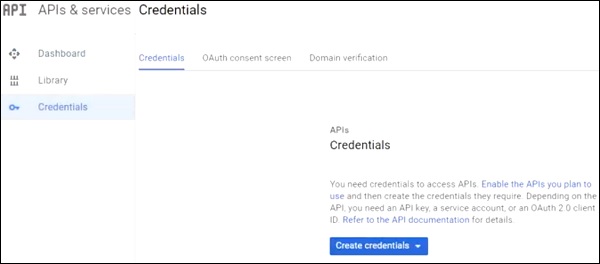
<a href = "user">Click here to Google Login</a>

</body>

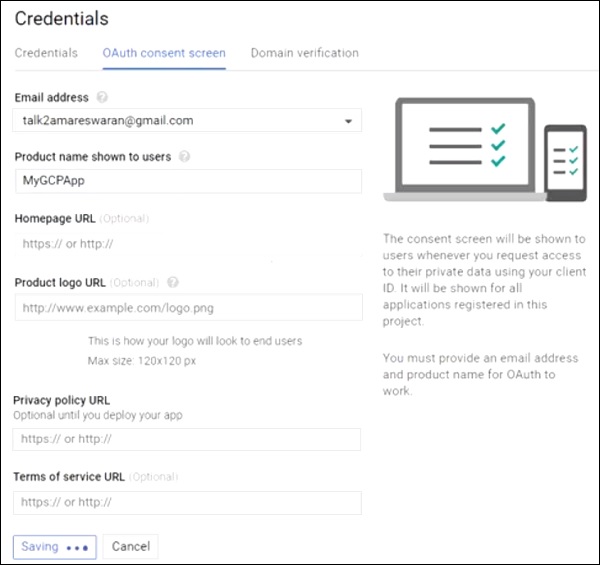
</html>

Lưu ý – trong Google Cloud console – Enable Gmail Services , Analytics Services và Google+ service API

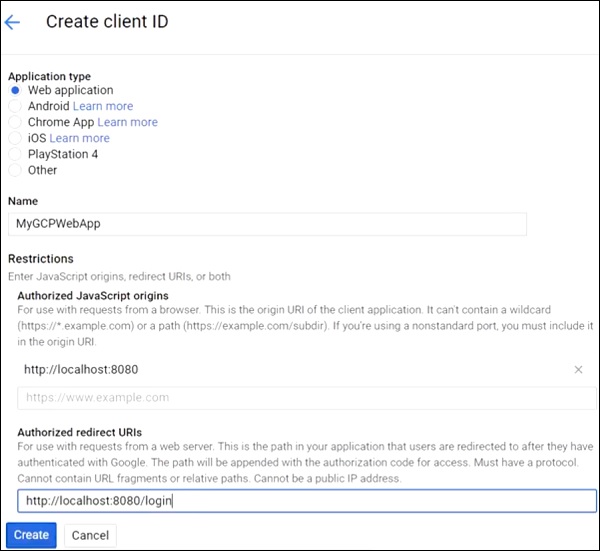
Sau đó , đi tới phần Credentials và tạo một credentials và chọn Oauth Client ID



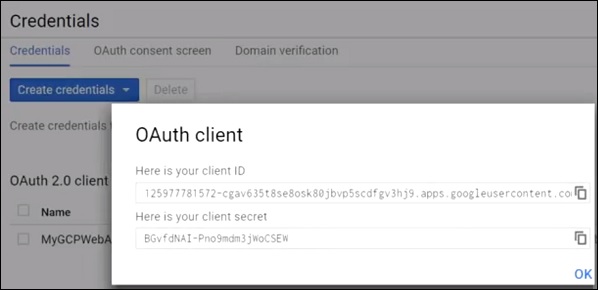
Tiếp theo , cung cấp Product Name trong Oauth2 consent screen:



Tiếp theo , chọn Application Type là Web application , cung cấp Authorized javascript origins và Authorized redirect URLs



Bây giờ , Oauth2 Client Id và Client Secret được tạo:



Tiếp theo , thêm Client Id và Client Secret trong application.properties:

security.oauth2.client.clientId = <CLIENT\_ID>

security.oauth2.client.clientSecret = <CLIENT\_SECRET>

security.oauth2.client.accessTokenUri = https://www.googleapis.com/oauth2/v3/token

security.oauth2.client.userAuthorizationUri = https://accounts.google.com/o/oauth2/auth

security.oauth2.client.tokenName = oauth\_token

security.oauth2.client.authenticationScheme = query

security.oauth2.client.clientAuthenticationScheme = form

security.oauth2.client.scope = profile email

security.oauth2.resource.userInfoUri = https://www.googleapis.com/userinfo/v2/me

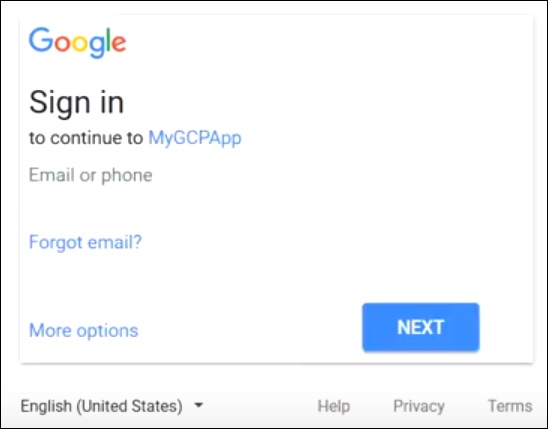
security.oauth2.resource.preferTokenInfo = false

kết quả:

<http://localhost:8080/> :

Google Login link

Nó sẽ redirect tới Google login screen và cung cấp một Gmail login details:



Nếu login thành công , chúng ta sẽ nhận được đối tượng Principal của Gmail user.

