**Spring boot**

Open source Java-based framework used to create a Micro Service

**Micro Service**

Is an architecture that allows the developers to develop and deploy services independently

Advantages:

Easy deployment

Simple scalability

Compatible with Containers

Minimum configuration

Lesser production time

**Spring boot**

**Advantages**:

Easy to understand and develop spring applications

Increases productivity

Reduces the development time

**Goals**:

To avoid complex XML configuration in Spring

To develop a production ready Spring applications in an easier way

To reduce the development time and run the application independently

Offer an easier way of getting started with the application

**Benefits**:

It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.

It provides a powerful batch processing and manages REST endpoints.

In Spring Boot, everything is auto configured; no manual configurations are needed.

It offers annotation-based spring application

Eases dependency management

It includes Embedded Servlet Container

Spring Boot **Starter Actuator** dependency is used to monitor and manage your application

<dependency>

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

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

</dependency>

Automatically configured:

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);

}

}

@SpringBootApplication annotation includes Auto- Configuration, Component Scan, and Spring Boot Configuration

Entry point:

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);

}

}

Scan all the beans and package declarations:

import org.springframework.boot.SpringApplication;

import org.springframework.context.annotation.ComponentScan;

**@ComponentScan**

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

///////////////////////

**Instalar maven**

<https://maven.apache.org/download.cgi>

Bajar el zip de link

Agregar en variables del sistema, en variables de usuario agregando bin

Crear la variable M2\_HOME con la ruta sin bin

En la consola > mvn –version

**Instalar gradle**

<https://gradle.org/releases/>

Descargar el complete

Poner en Archivos de programa, copiar la ruta y pegarla en path de las variables del sistema, en variables de usuario agregando \bin

En la consola gradle –version

**Instalar Spring Boot CLI**

<https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-installing-the-cli> > en Manual Installation está el zip

Pegar descomprimido en Archivos de programa

Crear la variable del sistema, en path agregando bin y una sin bin con el nombre SPRING\_HOME

En la consola spring –version

Abrir un editor de texto

@Controller

class Example {

@RequestMapping("/")

@ResponseBody

public String hello() {

"Hello Spring Boot"

}

}

grabarlo con .groovy dentro de la ruta donde está el spring boot cli, en bin

Abrir el cmd en la ruta donde está el archivo

spring run nombreArchivo.groovy

/////////////////////////////////////////

bootstrapping: empezar algo con pocos recursos

Ir a

<https://start.spring.io/>

Group: com.tutorialspoint

Artifact: demo

Dependencies >> ADD DEPENDENCIES

JPA

Actuator

Web

Descargar el proyecto, descomprimirlo

For writing a Rest Endpoint, we need to add the spring-boot-starter-web dependency in our class path

Write a Rest Endpoint:

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";

}

}

You can find the main class file under src/java/main

Add the @RestController annotation at the top of the class

Write a Request URI method with @RequestMapping annotation

The Request URI method should return the Hello World string

En el cmd > mvn clean install

En el cmd > gradle clean build

Once you have created an executable JAR file, you can find it under the following directories:

Maven en: target

Gradle en: build/libs

Ir al directorio y:

java –jar demo-0.0.1-SNAPSHOT.jar

Abrir el navegador en:

http://localhost:8080/

/////////////////////////////////

Support WAR file deployment:

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);

}

}

We need to mention the main class that should start in the build file

Maven:

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

Gradle, add in build.gradle:

mainClassName="com.tutorialspoint.demo.DemoApplication"

We have to update the packaging JAR into WAR:

Maven:

<packaging>war</packaging>

Gradle:

apply plugin: ‘war’

apply plugin: ‘application’

To write a Rest Endpoint, we need to add the Spring Boot web starter dependency into our build file:

<dependency>

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

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

</dependency>

dependencies {

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

}

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:

mvn package

Your WAR file will be created and you can find it under build/libs directory:

gradle clean build

Tomcat > webapps

Web browser <http://localhost:8080/demo-0.0.1-SNAPSHOT/>

///////////////////////////////

Spring Boot automatically configures the dependencies version based on the release. Remember that when you upgrade the Spring Boot version, dependencies also will upgrade automatically

For Maven configuration, we should inherit the Spring Boot Starter parent project to manage the Spring Boot Starters dependencies:

<parent>

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

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

<version>1.5.8.RELEASE</version>

</parent>

We should specify the version number for Spring Boot Parent Starter dependency. Then for other starter dependencies:

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

We can import the Spring Boot Starters dependencies directly:

buildscript {

ext {

springBootVersion = '1.5.8.RELEASE'

}

repositories {

mavenCentral()

}

dependencies {

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

}

}

We need not specify the Spring Boot version number for dependencies:

dependencies {

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

}

///////////////////////////////////////////////////////////////////////////

A class that does not have any package declaration is considered as a default package

A default package declaration is not recommended

Naming convention for package declaration is reversed domain name. For example − com.tutorialspoint.myproject

com -> tutorialspoint -> myproject

myproject -> Application.java, model, dao, controller, service

main method goes in Applicacion.java

///////////////////////////////

The @ComponentScan annotation is used to find beans and the corresponding injected with @Autowired annotation.

auto wired Rest Template object and Bean creation object in main Spring Boot Application class file:

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();

}

}

///////////////////////////////

Application Runner and Command Line Runner interfaces lets you to execute the code after the Spring Boot application is started.

@Override

public void run(ApplicationArguments arg0) throws Exception {

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

}

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

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

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

}

}

///////////////////////////////

**Application Properties**

Command line properties take precedence over the other property sources

Change the port number:

java –jar demo-0.0.1-SNAPSHOT.jar –server.port=9090

You can provide more than one application properties by using the delimiter –

Properties are kept in the application.properties file under the classpath In the src/main/resources directory

server.port = 9090

spring.application.name = demoservice

application.yml file this YAML file also should be kept inside the classpath

spring:

application:

name: demoservice

server:

port: 9090

Specify the location of properties file while running the JAR:

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

Read the environment or application property value:

@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;

}

}

To resolve the placeholder issue, we can set the default value for the property using the syntax given below:

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

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

Spring Boot supports different properties based on the Spring active profile

If you want to use profile based properties, we can keep separate properties file for each profile as shown below:

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

While running the JAR file, we need to specify the spring active profile based on each properties file.

By default, Spring Boot application uses the application.properties file.

The command to set the spring active profile is shown below:

java –jar demo-0.0.1-SNAPSHOT.jar –spring-profiles.active=dev

Set the Production active profile:

java –jar demo-0.0.1-SNAPSHOT.jar –spring-profiles.active=prod

To keep the Spring active profiles in application.yml file. Note that the delimiter (---) is used to separate each profile in 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

Set development active profile:

java –jar demo-0.0.1-SNAPSHOT.jar –spring-profiles.active=dev

/////////////////////////////////

**Logging**

Spring Boot uses Apache Commons logging for all internal logging. Spring Boot’s default configurations provides a support for the use of Java Util Logging, Log4j2, and Logback

Date and Time of the log

Log level shows INFO, ERROR or WARN

Process ID

The --- which is a separator

Thread name is enclosed within the square brackets []

Logger Name that shows the Source class name

The Log message

By default, “INFO”, “ERROR” and “WARN” log messages will print in the log file

Enable the debug level log:

java –jar demo.jar –debug

Add the debug mode to your application.properties file:

debug = true

If you want to print the logs in a file, you need to set the property logging.file or logging.path in the application.properties file

Specify the log file path:

logging.path = /var/tmp/

Specify the own log file name:

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

Files will rotate automatically after reaching the size 10 MB

Define Root logger:

logging.level.root = WARN

Logback does not support “FATAL” level log. It is mapped to the “ERROR” level log

Logging configuration details are configured in logback.xml file. The logback.xml file should be placed under the classpath

Configure the ROOT level log in Logback.xml:

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

<configuration>

<root level = "INFO">

</root>

</configuration>

Configure the console appender in Logback.xml:

<?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>

Configure the file appender in Logback.xml:

<?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>

Define the Log pattern in logback.xml, also define the set of supported log patterns inside the console or file log appender:

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

Place this in the class path:

<?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>

Add the slf4j logger in 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);

}

}

////////////////////////////////

**Building RESTful Web Services**

Need to add the Spring Boot Starter Web dependency into the build configuration file

<dependency>

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

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

</dependency>

build.gradle file:

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

<?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>

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')

}

Define the RESTful web services. It serves JSON, XML and custom response:

@RestController

public class ProductServiceController {

}

Define the Request URI to access the REST Endpoints. We can define Request method to consume and produce object. The default request method is GET:

@RequestMapping(value = "/products")

public ResponseEntity<Object> getProducts() { }

Define the request body content type:

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

}

Define the custom or dynamic request URI:

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

}

Read the request parameters from the Request URL. By default, it is a required parameter. We can also set default value for request parameters:

public ResponseEntity<Object> getProduct(

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

}

The default HTTP request method is GET. This method does not require any Request Body. You can send request parameters and path variables to define the custom or dynamic URL

We used HashMap to store the Product. Note that we used a POJO class as the product to be stored.

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);

}

}

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);

}

}

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);

}

}

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);

}

}

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);

}

}

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;

}

}

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);

}

}

Luego:

mvn clean install

gradle clean build

Localizar el jar y a continuación:

java –jar <JARFILE>

En postman, get:

<http://localhost:8080/products>

post:

{

“name”:”Indian Ginger”

}

Delete:

<http://localhost:8080/products/3>

///////////////////////////////

**Exception Handling**

**@ControllerAdvice** is an annotation, to handle the exceptions globally

Handle the specific exceptions and sending the custom responses to the client:

package com.tutorialspoint.demo.exception;

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

**@ControllerAdvice**

public class ProductExceptionController {

}

Define a class that extends the RuntimeException class:

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Writing the Controller Advice class file:

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

}

Throw the exception from the API:

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

public ResponseEntity<Object> updateProduct() {

throw new ProductNotfoundException();

}

The ProductNotFoundException exception class should extend the **RuntimeExceptio**n:

package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends **RuntimeException** {

private static final long serialVersionUID = 1L;

}

The Controller Advice class to handle the exception globally is given below. We can define any Exception Handler methods in this class file:

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);

}

}

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);

}

}

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);

}

}

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;

}

}

<?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>

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')

}

//////////////////////////

**Interceptor**

To perform operations:

Before sending the request to the controller

Before sending the response to the client

To work with interceptor, you need to create @Component class that supports it and it should implement the HandlerInterceptor interface

preHandle() method − to perform operations before sending the request to the controller. This method should return true to return the response to the client

postHandle() method − to perform operations before sending the response to the client

afterCompletion() method − to perform operations after completing the request and response

@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 {}

}

You will have to register this Interceptor with InterceptorRegistry by using WebMvcConfigurerAdapter as shown below:

@Component

public class ProductServiceInterceptorAppConfig extends WebMvcConfigurerAdapter {

@Autowired

ProductServiceInterceptor productServiceInterceptor;

@Override

public void addInterceptors(InterceptorRegistry registry) {

registry.addInterceptor(productServiceInterceptor);

}

}

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");

}

}

The code for Application Configuration class file to register the Interceptor into Interceptor Registry – ProductServiceInterceptorAppConfig.java is given below:

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);

}

}

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);

}

}

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;

}

}

////////////////////////