Home Subscrib

Spring Boot ResponseEntity

Spring Boot ResponseEntity tutorial shows how to use `ResponseEntity` in a Spring application.

Tweet

Spring is a popular Java application framework and *Spring Boot* is an evolution of Spring that helps create stand-alone, production-grade Spring based applications easily.

ResponseEntity

`ResponseEntity` represents an HTTP response, including headers, body, and status. While `@ResponseBoc` puts the return value into the body of the response, `ResponseEntity` also allows us to add headers and status code.

Spring Boot ResponseEntity example

In the following application, we demonstrate the usage of `ResponseEntity`. The application has two methods: one method uses `ResponseEntity` to create an HTTP response, the other one `@ResponseBody`.

```
pom.xml
src
├───main
│   ├───java
│   │   └───com
│   │       └───zetcode
│   │           │   Application.java
│   │           ├───controller
│   │           │       MyController.java
│   │           └───model
│   │                   Country.java
│   └───resources
│       └───static
│               index.html
└───test
    └───java
```

This is the project structure of the Spring application.

`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.zetcode</groupId>
    <artifactId>responseentityex</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
    </parent>

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

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

</project>
```

This is the Maven `pom.xml` file. The `spring-boot-starter-parent` is a parent POM providing dependen and plugin management for applications built with Maven. The `spring-boot-starter-web` is a dependency for creating Spring Boot web applications using Spring MVC. The `spring-boot-maven-plug` packages Spring applications into executable JAR or WAR archives.

`com/zetcode/model/Country.java`

```
package com.zetcode.model;

public class Country {
    private String name;
    private int population;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPopulation() {
        return population;
    }

    public void setPopulation(int population) {
        this.population = population;
    }
}
```

This is the `Country` bean. It has two attributes: `name` and `population`.

com/zetcode/controller/MyController.java

```
package com.zetcode.controller;

import com.zetcode.bean.Country;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyController {

    @RequestMapping(value = "/getCountry")
    public ResponseEntity<Country> getCountry() {

        var c = new Country();
        c.setName("France");
        c.setPopulation(66984000);

        var headers = new HttpHeaders();
        headers.add("Responded", "MyController");

        return ResponseEntity.accepted().headers(headers).body(c);
    }

    @RequestMapping(value = "/getCountry2")
    @ResponseBody
    public Country getCountry2() {

        var c = new Country();
        c.setName("France");
        c.setPopulation(66984000);

        return c;
    }
}
```

The controller contains two methods. The first one uses `ResponseEntity`, the second one `@ResponseBody`

```
@RequestMapping(value = "/getCountry")
public ResponseEntity<Country> getCountry() {
```

The `getCountry()` method is mapped to the `getCountry` URL pattern; it returns a `ResponseEntity` of ty `Country`.

```
var c = new Country();
c.setName("France");
c.setPopulation(66984000);
```

We create a `Country` bean; this bean is returned in the response.

```
var headers = new HttpHeaders();
headers.add("Responded", "MyController");
```

We create an instance of `HttpHeaders` and add a new header value.

```
return ResponseEntity.accepted().headers(headers).body(c);
```

A `ResponseEntity` is returned. We give `ResponseEntity` a custom status code, headers, and a body.

```
@RequestMapping(value = "/getCountry2")
@ResponseBody
public Country getCountry2() {

    var c = new Country();
    c.setName("France");
    c.setPopulation(66984000);

    return c;
}
```

With `@ResponseBody`, only the body is returned. The headers and status code are provided by Spring.

`resources/static/index.html`

```
<!DOCTYPE html>
<html>
    <head>
        <title>Home page</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <p>
            <a href="getCountry">Get country 1</a>
        </p>

        <p>
            <a href="getCountry2">Get country 2</a>
        </p>

    </body>
</html>
```

This is the home page. It contains two links.

com/zetcode/Application.java

```
package com.zetcode;

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);
    }
}
```

`Application` is the entry point which sets up Spring Boot application.

```
$ curl localhost:8080/getCountry -I
HTTP/1.1 202
Responded: MyController
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 17 Jan 2019 21:40:49 GMT
```

When calling the first method, we can see the chosen 202 status code and the custom header value.

In this tutorial, we have shown how to use `ResponseEntity` in a Spring application. You might also be interested in the related tutorials: Spring Boot @ResponseStatus tutorial, Spring Boot @ExceptionHandle tutorial, Spring Boot upload file, Spring Boot @PathVariable tutorial, Spring Boot @RequestParam tutoria Spring Boot REST H2 tutorial, Standalone Spring applications, Java tutorial.

Home Top of Page

ZetCode last modified January 17, 2019 © 2007 - 2019 Jan Bodnar Follow on Facebook