Spring Boot Jersey

last modified July 6, 2020

Spring Boot Jersey tutorial shows how to set up a simple RESTFul application with Jersey in a Spring Boot application. Jersey is an alternative to Spring RESTFul applications created with @RestController.

Spring is a popular Java application framework for creating enterprise applications. *Spring Boot* is the next step in evolution of Spring framework. It helps create stand-alone, production-grade Spring based applications with minimal effort. It promotes using the *convention over configuration*principle over XML configurations.

RESTFul application

A RESTFul application follows the REST architectural style, which is used for designing networked applications. RESTful applications generate HTTP requests performing CRUD (Create/Read/Update/Delete) operations on resources. RESTFul applications typically return data in JSON or XML format.

JAX-RS

Java API for RESTful Web Services (JAX-RS) is a Java programming language API specification that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern. JAX-RS uses annotations to simplify the development and deployment of web service clients and endpoints. JAX-RS is an official part of Java EE.

Jersey

Jersey is an open source framework for developing RESTful Web Services in Java. It is a reference implementation of the Java API for RESTful Web Services (JAX-RS) specification.

Spring Boot Jersey example

The following application is a simpe Spring Boot RESTful application created with Jersey.

```
$ tree
.
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── com
    │   │       └── zetcode
    │   │           ├── Application.java
    │   │           ├── config
    │   │           │   └── JerseyConfig.java
```

```
    │   │           └── endpoint
    │   │               ├── HelloEndpoint.java
    │   │               └── ReverseReturnEndpoint.java
    │   └── resources
    └── test
        └── java
            └── com
                └── zetcode
                    └── ApplicationTests.java
```

This is the project structure.

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>SpringBootJersey</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.9.RELEASE</version>
    </parent>

    <dependencies>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jersey</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>
```

This is the Maven build file. Spring Boot starters are a set of convenient dependency descriptors which greatly simplify Maven configuration. The spring-boot-starter-parent has some common configurations for a Spring Boot application. The spring-boot-starter-jersey is a starter for building RESTful web applications using JAX-RS and Jersey. It is an alternative to spring-boot-starter-web. The spring-boot-starter-test is a starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito.

The spring-boot-maven-plugin provides Spring Boot support in Maven, allowing us to package executable JAR or WAR archives. Its spring-boot:run goal runs the Spring Boot application.

application.yml

```
server:
    port: 8086
    context-path: /api

spring:
    main:
        banner-mode: "off"

logging:
    level:
        org:
            springframework: ERROR
```

In the application.yml file we write various configuration settings of a Spring Boot application. We set the port and the context path. With the banner-mode property we turn off the Spring banner.

We set the logging level for spring framework to ERROR. The application.yml file is located in the in the src/main/resources directory.

JerseyConfig.java

```
package com.zetcode.config;

import com.zetcode.endpoint.HelloService;
import com.zetcode.endpoint.ReverseService;
import org.glassfish.jersey.server.ResourceConfig;
import org.springframework.context.annotation.Configuration;

@Configuration
public class JerseyConfig extends ResourceConfig {
```

```
    public JerseyConfig() {

        register(HelloService.class);
        register(ReverseService.class);
    }
}
```

JerseyConfig registers two service classes.

HelloService.java

```
package com.zetcode.service;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import org.springframework.stereotype.Service;

@Service
@Path("/hello")
public class HelloService {

    @GET
    @Produces("text/plain")
    public String hello() {
        return "Hello from Spring";
    }
}
```

This is the HelloService. The @Path annotation defines the URL to which the service class will respond. HelloService is annotated also with Spring's @Service for autodetection. Our service method simply returns "Hello from Spring" message.

HelloService.java

```
package com.zetcode.service;

import javax.validation.constraints.NotNull;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import org.springframework.stereotype.Service;

@Service
@Path("/reverse")
public class ReverseService {

    @GET
    @Produces("text/plain")
    public String reverse(@QueryParam("data") @NotNull String data) {
```

```
            return new StringBuilder(data).reverse().toString();
        }
    }
```

The reverse() service method returns a string which is reversed. It accepts one parameter, which cannot be null. @QueryParam binds the value(s) of a HTTP query parameter to a resource method parameter.

ApplicationTests.java

```
package com.zetcode;

import static org.assertj.core.api.Assertions.assertThat;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class ApplicationTests {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void hello() {
        ResponseEntity<String> entity = this.restTemplate.getForEntity("/hello",
                String.class);
        assertThat(entity.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(entity.getBody()).isEqualTo("Hello from Spring");
    }

    @Test
    public void reverse() {
        ResponseEntity<String> entity = this.restTemplate
                .getForEntity("/reverse?data=regit", String.class);
        assertThat(entity.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(entity.getBody()).isEqualTo("tiger");
    }

    @Test
    public void validation() {
        ResponseEntity<String> entity = this.restTemplate.getForEntity("/reverse",
                String.class);
        assertThat(entity.getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
    }
}
```

In the ApplicationTests, we test the two endpoints.

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);
    }
}
```

The Application sets up the Spring Boot application. The @SpringBootApplication enables auto-configuration and component scanning.

```
$ mvn spring-boot:run
```

With mvn spring-boot:run command, we run the application. The application is deployed on embedded Tomcat server.

```
$ curl localhost:8086/api/hello
Hello from Spring
```

With the curl command, we connect to the hello endpoint.

```
$ curl localhost:8086/api/reverse?data=summer
remmus
```

The summer's characters are reversed.

In this tutorial, we have created a simple RESTFul application in Spring Boot with Jersey, which is the reference implementation of the JAX-RS specification.