Spring boot MockMVC example

By Lokesh Gupta | Filed Under: Spring Boot Test

Learn to use **Spring MockMVC** to perform **integration testing** of Spring webmvc controllers. **MockMVC** class is part of Spring MVC test framework which helps in testing the controllers explicitly starting a Servlet container.

In this MockMVC tutorial, we will use it along with Spring boot's **WebMvcTest** class to execute Junit testcases which tests REST controller methods written for **Spring boot 2 hateoas example**.

1. Maven Dependencies

The `spring-boot-starter-test` dependency includes all required dependencies to create and execute tests.

pom.xml

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

If not using spring boot then include following dependencies.

pom.xml

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
```

- Using `EmployeeRESTController.class` as parameter, we are asking to initialize only one web controller and you need to provide remaining dependencies required using `Mock` objects.

3. Spring boot MockMvc tests example

Let’s create some JUnit tests which will test different HTTP methods in controller class.

3.1. HTTP GET

The HTTP APIs are defined in controller are given below. In given tests, we are testing two GET apis – one without path parameter and another with path parameter.

EmployeeRESTController.java

```
@GetMapping(value = "/employees")
public EmployeeListVO getAllEmployees()
{
    //code
}

@GetMapping(value = "/employees/{id}")
public ResponseEntity<EmployeeVO> getEmployeeById (@PathVariable("id") int id)
{
    //code
}
```

And corresponding tests for the methods are given below. These tests hit the APIs, pass the path parameters using **MockMvcRequestBuilders** and verify the status response codes and response content using **MockMvcResultMatchers** and **MockMvcResultHandlers**.

TestEmployeeRESTController.java

```
@Autowired
private MockMvc mvc;
```

```
    <version>{version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>{version}</version>
    <scope>test</scope>
</dependency>
```

2. Test Class Configuration

A JUnit test class to test the Spring MVC controller request and responses, we can use below given configuration.

Controller Test

```
@RunWith(SpringRunner.class)
@WebMvcTest(EmployeeRESTController.class)
public class TestEmployeeRESTController {

  @Autowired
  private MockMvc mvc;

}
```

- `SpringRunner` is an alias for the `SpringJUnit4ClassRunner`. It is a custom extension of JUnit's BlockJUnit4ClassRunner which provides functionality of the Spring TestContext Framework to standard JUnit tests by means of the TestContextManager and associated support classes and annotations.
- `@WebMvcTest` annotation is used for Spring MVC tests. It disables full auto-configuration and instead apply only configuration relevant to MVC tests.
- The `WebMvcTest` annotation auto-configure `MockMvc` instance as well.

```
@Test
public void getAllEmployeesAPI() throws Exception
{
  mvc.perform( MockMvcRequestBuilders
      .get("/employees")
      .accept(MediaType.APPLICATION_JSON))
      .andDo(print())
      .andExpect(status().isOk())
      .andExpect(MockMvcResultMatchers.jsonPath("$.employees").exists())
      .andExpect(MockMvcResultMatchers.jsonPath("$.employees[*].employeeId").isNot
}

@Test
public void getEmployeeByIdAPI() throws Exception
{
  mvc.perform( MockMvcRequestBuilders
      .get("/employees/{id}", 1)
      .accept(MediaType.APPLICATION_JSON))
      .andDo(print())
      .andExpect(status().isOk())
      .andExpect(MockMvcResultMatchers.jsonPath("$.employeeId").value(1));
}
```

3.2. HTTP POST

The HTTP POST API is defined in controller is:

```
EmployeeRESTController.java
```

```
@PostMapping(value = "/employees")
public ResponseEntity<EmployeeVO> addEmployee (@Valid @RequestBody EmployeeVO empl
{
    //code
    return new ResponseEntity<EmployeeVO>(employee, HttpStatus.CREATED);
}
```

And corresponding spring mockmvc test for post json request is as follows:

```
TestEmployeeRESTController.java
```

```
@Autowired
private MockMvc mvc;

@Test
public void createEmployeeAPI() throws Exception
{
  mvc.perform( MockMvcRequestBuilders
      .post("/employees")
      .content(asJsonString(new EmployeeVO(null, "firstName4", "lastName4", "email
      .contentType(MediaType.APPLICATION_JSON)
      .accept(MediaType.APPLICATION_JSON))
      .andExpect(status().isCreated())
      .andExpect(MockMvcResultMatchers.jsonPath("$.employeeId").exists());
}

public static String asJsonString(final Object obj) {
    try {
        return new ObjectMapper().writeValueAsString(obj);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

3.3. HTTP PUT

The HTTP API is defined in controller is:

```
EmployeeRESTController.java
```

```
@PutMapping(value = "/employees/{id}")
public ResponseEntity<EmployeeVO> updateEmployee (@PathVariable("id") int id, @Val
{
    //code
    return new ResponseEntity<EmployeeVO>(emp, HttpStatus.OK);
}
```

And corresponding tests for the methods are:

```
TestEmployeeRESTController.java
```

```
@Test
public void updateEmployeeAPI() throws Exception
{
  mvc.perform( MockMvcRequestBuilders
      .put("/employees/{id}", 2)
      .content(asJsonString(new EmployeeVO(2, "firstName2", "lastName2", "email2@m
      .contentType(MediaType.APPLICATION_JSON)
      .accept(MediaType.APPLICATION_JSON))
      .andExpect(status().isOk())
      .andExpect(MockMvcResultMatchers.jsonPath("$.firstName").value("firstName2")
      .andExpect(MockMvcResultMatchers.jsonPath("$.lastName").value("lastName2"))
      .andExpect(MockMvcResultMatchers.jsonPath("$.email").value("email2@mail.com"
}
```

3.4. HTTP DELETE

The HTTP API is defined in controller is:

```
EmployeeRESTController.java
```

```
@DeleteMapping(value = "/employees/{id}")
public ResponseEntity<HttpStatus> removeEmployee (@PathVariable("id") int id)
{
    //code
    return new ResponseEntity<HttpStatus>(HttpStatus.ACCEPTED);
}
```

And corresponding tests for the methods are:

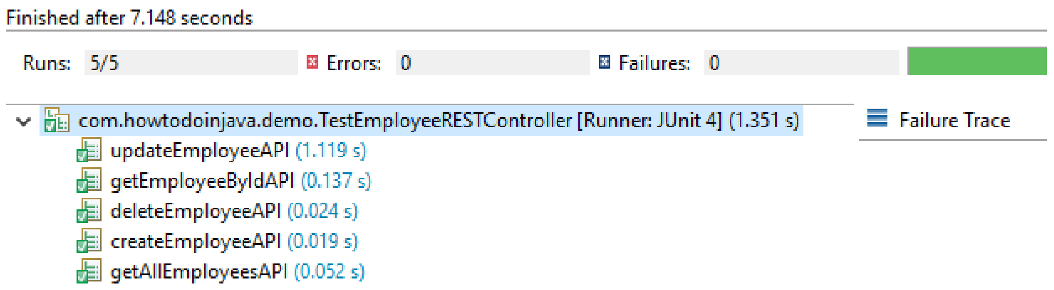
```
TestEmployeeRESTController.java
```

```
@Test
```

```
public void deleteEmployeeAPI() throws Exception
{
  mvc.perform( MockMvcRequestBuilders.delete("/employees/{id}", 1) )
        .andExpect(status().isAccepted());
}
```

3.5. Test Results

When we run above tests, we get the test results as below.



```
Finished after 7.148 seconds
Runs: 5/5    Errors: 0    Failures: 0

com.howtodoinjava.demo.TestEmployeeRESTController [Runner: JUnit 4] (1.351 s)
    updateEmployeeAPI (1.119 s)
    getEmployeeByIdAPI (0.137 s)
    deleteEmployeeAPI (0.024 s)
    createEmployeeAPI (0.019 s)
    getAllEmployeesAPI (0.052 s)
```

Test Results

4. Summary

In this **spring boot integration test example**, we learned to write *Spring MVC integration tests* using MockMvc class. He learned to write JUNit tests for HTTP GET, POST, PUT and DELETE APIs.

We also looked to verify API response status and response body, pass HTTP headers, and request path parameters as well.

Happy Learning !!

Read More:

SO Thread

About Lokesh Gupta

A family guy with fun loving nature. Love computers, programming and solving everyday problems. Find me on Facebook and Twitter.

Feedback, Discussion and Comments

Priynka

October 14, 2019

asJsonString is giving errror it will give MIME does not contain '/'

Reply

Parshuram Patil

September 16, 2019

For one of my POST API I am geting such response, whats the possible cause??

MockHttpServletResponse:
Status = 200

Error message = null
Headers = []
Content type = null
Body =
Forwarded URL = null
Redirected URL = null
Cookies = []

Reply

Raj

July 12, 2019

org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.ClassCastException: org.springframework.web.servlet.resource.ResourceHttpRequestHandler cannot be cast to org.springframework.web.method.HandlerMethod

I am getting this error, any idea what is wrong?

Reply

chandrajeet

July 18, 2019

Do you have any solution? I am getting the same exception

Reply

Baiju damodar

May 6, 2019

Hey , can we mock methods using MockMVC , if no How can we mock a method with the similar approach ?? Pleas do a session on that as well . A typical example of a method call "addValues(int a, int b)" whichj the adds , subtracts , multiplies, and divides a with b , and combines the entire result as one string and returns it.

Reply

adnsbd

April 10, 2019

Do you need to running service first to do Spring boot MockMvc tests ?

I got assertion error on line "status().isOk()"

Reply

sobhan

January 30, 2019

asJsonString is giving errror

Reply

Lokesh Gupta

January 30, 2019

What error you are facing. I hope you have defined the method as given in post :

```java
public static String asJsonString(final Object obj) {
    try {
        return new ObjectMapper().writeValueAsString(obj);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Reply

Ask Questions & Share Feedback

Please do not submit a comment only to say "Thank you".

Comment

*Want to Post Code Snippets or XML content? Please use [java] ... [/java] tags otherwise code may not appear partially or even fully. e.g.

```
[java]
public static void main (String[] args) {
...
```

}
[/java]

Name *

Email *

Website

POST COMMENT

Meta Links

Advertise
Contact Us
Privacy policy
About Me

Recommended Reading

10 Life Lessons
Secure Hash Algorithms
How Web Servers work?
How Java I/O Works Internally?
Best Way to Learn Java
Java Best Practices Guide
Microservices Tutorial

REST API Tutorial
How to Start New Blog