Unit Testing of Spring MVC Controllers: REST API

Petri Kainulainen July 26, 2013 138 comments

Spring Framework, Spring MVC, Spring MVC Test, Unit Testing

Spring MVC provides an easy way to create REST APIs. However, writing comprehensive and fast unit tests for these APIs has been troublesome. The release of the Spring MVC Test framework gave us the possibility to write unit tests which are readable, comprehensive and fast.

This blog post describes how we can write unit tests for a REST API by using the Spring MVC Test framework. During this blog post we will write unit tests for controller methods which provide CRUD functions for todo entries.

Let’s get started.

Getting The Required Dependencies with Maven

We can get the required testing dependencies by adding the following dependency declarations to our POM file:

- Hamcrest 1.3 (*hamcrest-all*). We use Hamcrest matchers when we are writing assertions for the responses.
- Junit 4.11. We need to exclude the *hamcrest-core* dependency because we already added the *hamcrest-all* dependency.
- Mockito 1.9.5 (*mockito-core*). We use Mockito as our mocking library.
- Spring Test 3.2.3.RELEASE
- JsonPath 0.8.1 (*json-path* and *json-path-assert*). We use JsonPath when we are writing assertions for JSON documents returned by our REST API.

The relevant dependency declarations looks as follows:

```xml
<dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-all</artifactId>
    <version>1.3</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <artifactId>hamcrest-core</artifactId>
            <groupId>org.hamcrest</groupId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>1.9.5</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>3.2.3.RELEASE</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>0.8.1</version>
```

```
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.jayway.jsonpath</groupId>
        <artifactId>json-path-assert</artifactId>
        <version>0.8.1</version>
        <scope>test</scope>
    </dependency>
```

Let’s move on and talk a bit about the configuration of our unit tests.

Configuring Our Unit Tests

The unit tests which we will write during this blog post use the web application context based configuration. This means that we configure the Spring MVC infrastructure by using either an application context configuration class or a XML configuration file.

Because the first part of this tutorial described the principles which we should follow when we are configuring the application context of our application, this issue is not discussed in this blog post.

However, there is one thing that we have to address here.

The application context configuration class (or file) which configures the web layer of our example application does not create an exception resolver bean. The *SimpleMappingExceptionResolver* class used in the earlier parts of this tutorial maps exception class name to the view which is rendered when the configured exception is thrown.

This makes sense if we are implementing a “normal” Spring MVC application. However, if we are implementing a REST API, we want to transform exceptions into HTTP status codes. This behavior is provided by the *ResponseStatusExceptionResolver* class which is enabled by default.

Our example application also has a custom exception handler class which is annotated with the @ControllerAdvice annotation. This class handles validation errors and application specific exceptions. We will talk more about this class later in this blog post.

Let’s move on and find out how we can write unit tests for our REST API.

Writing Unit Tests for a REST API

Before we can start writing unit tests for our REST API, we need to understand two things:

- We need to know what are the core components of the Spring MVC Test framework. These components are described in the second part of this tutorial.
- We need to know how we can write assertions for JSON documents by using JsonPath expressions. We can get this information by reading my blog post which describes how we can write clean assertions with JsonPath.

Next we will see the Spring MVC Test framework in action and write unit tests for the following controller methods:

- The first controller methods returns a list of todo entries.
- The second controller method returns the information of a single todo entry.
- The third controller method adds a new todo entry to the database and returns the added todo entry.

Get Todo Entries

The first controller method returns a list of todo entries which are found from the database. Let's start by taking a look at the implementation of this method.

Expected Behavior

The controller method which returns all todo entries stored to the database is implemented by following these steps:

1. It processes *GET* requests send to url '/api/todo'.
2. It gets a list of *Todo* objects by calling the *findAll()* method of the *TodoService* interface. This method returns all todo entries which are stored to the database. These todo entries are always returned in the same order.
3. It transforms the received list into a list of *TodoDTO* objects.

4. It returns the list which contains *TodoDTO* objects.

The relevant part of the *TodoController* class looks as follows:

```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@Controller
public class TodoController {

    private TodoService service;

    @RequestMapping(value = "/api/todo", method = RequestMethod.GET)
    @ResponseBody
    public List<TodoDTO> findAll() {
        List<Todo> models = service.findAll();
        return createDTOs(models);
    }

    private List<TodoDTO> createDTOs(List<Todo> models) {
        List<TodoDTO> dtos = new ArrayList<>();

        for (Todo model: models) {
            dtos.add(createDTO(model));
        }

        return dtos;
    }

    private TodoDTO createDTO(Todo model) {
        TodoDTO dto = new TodoDTO();

        dto.setId(model.getId());
        dto.setDescription(model.getDescription());
        dto.setTitle(model.getTitle());

        return dto;
    }
}
```

When a list of *TodoDTO* objects is returned, Spring MVC transforms this list into a JSON document which contains a collection of objects. The returned JSON document looks as follows:

```json
[
    {
        "id":1,
        "description":"Lorem ipsum",
        "title":"Foo"
    },
    {
        "id":2,
        "description":"Lorem ipsum",
        "title":"Bar"
    }
]
```

Let’s move on and write an unit test which ensures that this controller method is working as expected.

Test: Todo Entries Are Found

We can write an unit test for this controller method by following these steps:

1. Create the test data which is returned when the *findAll()* method of the *TodoService* interface is called. We create the test data by using a test data builder class.
2. Configure our mock object to return the created test data when its *findAll()* method is invoked.
3. Execute a *GET* request to url ‘/api/todo’.
4. Verify that the HTTP status code 200 is returned.
5. Verify that the content type of the response is ‘application/json’ and its character set is ‘UTF-8’.
6. Get the collection of todo entries by using the JsonPath expression *$* and ensure that that two todo entries are returned.
7. Get the *id*, *description*, and *title* of the first todo entry by using JsonPath expressions *$[0].id*, *$[0].description*, and *$[0].title*. Verify that the correct values are returned.
8. Get the *id*, *description*, and title of the second todo entry by using JsonPath expressions *$[1].id*, *$[1].description*, and *$[1].title*. Verify that the correct values are returned.
9. Verify that the *findAll()* method of the *TodoService* interface is called only once.
10. Ensure that no other methods of our mock object are called during the test.

The source code of our unit test looks as follows:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Arrays;

import static org.hamcrest.Matchers.*;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilder
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {TestContext.class, WebAppContext.class})
@WebAppConfiguration
public class TodoControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private TodoService todoServiceMock;

    //Add WebApplicationContext field here.

    //The setUp() method is omitted.

    @Test
    public void findAll_TodosFound_ShouldReturnFoundTodoEntries() throws Excepti
        Todo first = new TodoBuilder()
                .id(1L)
                .description("Lorem ipsum")
                .title("Foo")
                .build();
        Todo second = new TodoBuilder()
                .id(2L)
                .description("Lorem ipsum")
                .title("Bar")
                .build();

        when(todoServiceMock.findAll()).thenReturn(Arrays.asList(first, second))

        mockMvc.perform(get("/api/todo"))
                .andExpect(status().isOk())
                .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8)
                .andExpect(jsonPath("$", hasSize(2)))
                .andExpect(jsonPath("$[0].id", is(1)))
                .andExpect(jsonPath("$[0].description", is("Lorem ipsum")))
                .andExpect(jsonPath("$[0].title", is("Foo")))
                .andExpect(jsonPath("$[1].id", is(2)))
                .andExpect(jsonPath("$[1].description", is("Lorem ipsum")))
                .andExpect(jsonPath("$[1].title", is("Bar")));

        verify(todoServiceMock, times(1)).findAll();
        verifyNoMoreInteractions(todoServiceMock);
    }
}
```

Our unit test uses a constant called *APPLICATION_JSON_UTF8* which is declared in the *TestUtil* class. The value of that constant is a *MediaType* object which content type is 'application/json' and character set is 'UTF-8'.

The relevant part of the *TestUtil* class looks as follows:

```java
public class TestUtil {

    public static final MediaType APPLICATION_JSON_UTF8 = new MediaType(MediaType
                                                                        MediaType
                                                                        Charset.f
                                                                        );
}
```

Get Todo Entry

The second controller method which we have to test returns the information of a single todo entry. Let’s find out how this controller method is implemented.

Expected Behavior

The controller method which returns the information of a single todo entry is implemented by following these steps:

1. It processes *GET* requests send to url ‘/api/todo/{id}’. The *{id}* is a path variable which contains the *id* of the requested todo entry.
2. It obtains the requested todo entry by calling the *findById()* method of the *TodoService* interface and passes the *id* of the requested todo entry as a method parameter. This method returns the found todo entry. If no todo entry is found, this method throws a *TodoNotFoundException*.
3. It transforms the *Todo* object into a *TodoDTO* object.
4. It returns the created *TodoDTO* object.

The source code of our controller method looks as follows:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
public class TodoController {

    private TodoService service;

    @RequestMapping(value = "/api/todo/{id}", method = RequestMethod.GET)
    @ResponseBody
    public TodoDTO findById(@PathVariable("id") Long id) throws TodoNotFoundExce
        Todo found = service.findById(id);
        return createDTO(found);
    }

    private TodoDTO createDTO(Todo model) {
        TodoDTO dto = new TodoDTO();

        dto.setId(model.getId());
        dto.setDescription(model.getDescription());
        dto.setTitle(model.getTitle());

        return dto;
    }
}
```

The JSON document which is returned to the client looks as follows:

```
{
    "id":1,
    "description":"Lorem ipsum",
    "title":"Foo"
}
```

Our next question is:

> What happens when a TodoNotFoundException is thrown?

Our example application has an exception handler class which handles application specific exceptions thrown by our controller classes. This class has an exception handler method which is called when a *TodoNotFoundException* is thrown. The implementation of this method writes a new log message to the log file and ensures that the HTTP status code 404 is send back to the client.

The relevant part of the *RestErrorHandler* class looks as follows:

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;

@ControllerAdvice
public class RestErrorHandler {

    private static final Logger LOGGER = LoggerFactory.getLogger(RestErrorHandle

    @ExceptionHandler(TodoNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public void handleTodoNotFoundException(TodoNotFoundException ex) {
        LOGGER.debug("handling 404 error on a todo entry");
    }
}
```

We have to write two unit tests for this controller method:

1. We have to write a test which ensures that our application is working properly when the todo entry is not found.
2. We have to write a test which verifies that the correct data is returned to the client when the todo entry is found.

Let's see how we can write these tests.

Test 1: Todo Entry Is Not Found

First, we must ensure that our application is working properly when a todo entry is not found. We can write an unit test which ensures this by following these steps:

1. Configure our mock object to throw a *TodoNotFoundException* when its *findById()* method is called and the *id* of the requested todo entry is 1L.
2. Execute a *GET* request to url '/api/todo/1'.
3. Verify that the HTTP status code 404 is returned.
4. Ensure that the *findById()* method of the *TodoService* interface is called only once by using the correct method parameter (1L).
5. Verify that no other methods of the *TodoService* interface are called during this test.

The source code of our unit test looks as follows:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;

import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilder
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {TestContext.class, WebAppContext.class})
@WebAppConfiguration
public class TodoControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private TodoService todoServiceMock;

    //Add WebApplicationContext field here.

    //The setUp() method is omitted.

    @Test
    public void findById_TodoEntryNotFound_ShouldReturnHttpStatusCode404() throw
        when(todoServiceMock.findById(1L)).thenThrow(new TodoNotFoundException("

        mockMvc.perform(get("/api/todo/{id}", 1L))
                .andExpect(status().isNotFound());

        verify(todoServiceMock, times(1)).findById(1L);
        verifyNoMoreInteractions(todoServiceMock);
    }
}
```

Test 2: Todo Entry Is Found

Second, we must write a test which ensures that the correct data is returned when the requested todo entry is found. We can write a test which ensures this by following these steps:

1. Create the *Todo* object which is returned when our service method is called. We create this object by using our test data builder.
2. Configure our mock object to return the created *Todo* object when its *findById()* method is called by using a method parameter 1L.
3. Execute a *GET* request to url '/api/todo/1'.
4. Verify that the HTTP status code 200 is returned.
5. Verify that the content type of the response is 'application/json' and its character set is 'UTF-8'.
6. Get the *id* of the todo entry by using the JsonPath expression *$.id* and verify that the *id* is 1.
7. Get the *description* of the todo entry by using the JsonPath expression *$.description* and verify that the *description* is "Lorem ipsum".
8. Get the *title* of the todo entry by using the JsonPath expression *$.title* and verify that the title is "Foo".
9. Ensure that the *findById()* method of the *TodoService* interface is called only once by using the correct method parameter (1L).
10. Verify that the other methods of our mock object are not called during the test.

The source code of our unit test looks as follows:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.Matchers.is;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilder
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.

@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration(classes = {TestContext.class, WebAppContext.class})
@WebAppConfiguration
public class TodoControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private TodoService todoServiceMock;

    //Add WebApplicationContext field here.

    //The setUp() method is omitted.

    @Test
    public void findById_TodoEntryFound_ShouldReturnFoundTodoEntry() throws Exce
        Todo found = new TodoBuilder()
                .id(1L)
                .description("Lorem ipsum")
                .title("Foo")
                .build();

        when(todoServiceMock.findById(1L)).thenReturn(found);

        mockMvc.perform(get("/api/todo/{id}", 1L))
                .andExpect(status().isOk())
                .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8)
                .andExpect(jsonPath("$.id", is(1)))
                .andExpect(jsonPath("$.description", is("Lorem ipsum")))
                .andExpect(jsonPath("$.title", is("Foo")));

        verify(todoServiceMock, times(1)).findById(1L);
        verifyNoMoreInteractions(todoServiceMock);
    }
}
```

Add New Todo Entry

The third controller method adds a new todo entry to the database and returns the information of the added todo entry. Let's move on and find out how it is implemented.

Expected Behavior

The controller method which adds new todo entries to the database is implemented by following these steps:

1. It processes *POST* requests send to url '/api/todo'.
2. It validates the *TodoDTO* object given as a method parameter. If the validation fails, a *MethodArgumentNotValidException* is thrown.
3. It Adds a new todo entry to the database by calling the *add()* method of the *TodoService* interface and passes the *TodoDTO* object as a method parameter. This method adds a new todo entry to the database and returns the added todo entry.
4. It transforms the created *Todo* object into a *TodoDTO* object.

5. It returns the *TodoDTO* object.

The source code of our controller method looks as follows:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@Controller
public class TodoController {

    private TodoService service;

    @RequestMapping(value = "/api/todo", method = RequestMethod.POST)
    @ResponseBody
    public TodoDTO add(@Valid @RequestBody TodoDTO dto) {
        Todo added = service.add(dto);
        return createDTO(added);
    }

    private TodoDTO createDTO(Todo model) {
        TodoDTO dto = new TodoDTO();

        dto.setId(model.getId());
        dto.setDescription(model.getDescription());
        dto.setTitle(model.getTitle());

        return dto;
    }
}
```

The *TodoDTO* class is a simple DTO class which source code looks as follows:

```
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;

public class TodoDTO {

    private Long id;

    @Length(max = 500)
    private String description;

    @NotEmpty
    @Length(max = 100)
    private String title;

    //Constructor and other methods are omitted.
}
```

As we can see, this class declares three validation constraints which are described in the following:

1. The maximum length of the *description* is 500 characters.

2. The *title* of a todo entry cannot be empty.

3. The maximum length of the *title* is 100 characters.

If the validation fails, our error handler component ensures that

1. The HTTP status code 400 is returned to the client.
2. The validation errors are returned to the client as a JSON document.

Because I have already written a blog post which describes how we can add validation to a REST API, the implementation of the error handler component is not discussed in this blog post.

However, we need to know what kind of a JSON document is returned to the client if the validation fails. This information is given in the following.

If the *title* and the *description* of the *TodoDTO* object are too long, the following JSON document is returned to the client:

```
{
    "fieldErrors":[
        {
            "path":"description",
            "message":"The maximum length of the description is 500 characters."
        },
        {
            "path":"title",
            "message":"The maximum length of the title is 100 characters."
        }
    ]
}
```

Note: Spring MVC does not guarantee the ordering of the field errors. In other words, the field errors are returned in random order. We have to take this into account when we are writing unit tests for this controller method.

On the other hand, if the validation does not fail, our controller method returns the following JSON document to the client:

```
{
    "id":1,
    "description":"description",
    "title":"todo"
}
```

We have to write two unit tests for this controller method:

1. We have to write a test which ensures that our application is working properly when the validation fails.
2. We have to write a test which ensures that our application is working properly when a new todo entry is added to the database.

Let's find out how we can write these tests.

Test 1: Validation Fails

Our first test ensures that our application is working properly when the validation of the added todo entry fails. We can write this test by following these steps:

1. Create a *title* which has 101 characters.
2. Create a *description* which has 501 characters.
3. Create a new *TodoDTO* object by using our test data builder. Set the *title* and the *description* of the object.
4. Execute a *POST* request to url '/api/todo'. Set the content type of the request to 'application/json'. Set the character set of the request to 'UTF-8'. Transform the created *TodoDTO* object into JSON bytes and send it in the body of the request.
5. Verify that the HTTP status code 400 is returned.
6. Verify that the content type of the response is 'application/json' and its content type is 'UTF-8'.
7. Fetch the field errors by using the JsonPath expression *$.fieldErrors* and ensure that two field errors are returned.
8. Fetch all available paths by using the JsonPath expression *$.fieldErrors[*].path* and ensure that field errors about the *title* and *description* fields are found.
9. Fetch all available error messages by using the JsonPath expression *$.fieldErrors[*].message* and ensure that error messages about the *title* and *description* fields are found.
10. Verify that the methods of our mock object are not called during our test.

The source code of our unit test looks as follows:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.Matchers.containsInAnyOrder;
import static org.hamcrest.Matchers.hasSize;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilder
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {TestContext.class, WebAppContext.class})
@WebAppConfiguration
public class TodoControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private TodoService todoServiceMock;

    //Add WebApplicationContext field here.

    //The setUp() method is omitted.

    @Test
    public void add_TitleAndDescriptionAreTooLong_ShouldReturnValidationErrorsFo
        String title = TestUtil.createStringWithLength(101);
        String description = TestUtil.createStringWithLength(501);

        TodoDTO dto = new TodoDTOBuilder()
                .description(description)
                .title(title)
                .build();

        mockMvc.perform(post("/api/todo")
                .contentType(TestUtil.APPLICATION_JSON_UTF8)
                .content(TestUtil.convertObjectToJsonBytes(dto))
        )
                .andExpect(status().isBadRequest())
                .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8)
                .andExpect(jsonPath("$.fieldErrors", hasSize(2)))
                .andExpect(jsonPath("$.fieldErrors[*].path", containsInAnyOrder(
                .andExpect(jsonPath("$.fieldErrors[*].message", containsInAnyOrd
                        "The maximum length of the description is 500 characters
                        "The maximum length of the title is 100 characters."
                )));

        verifyZeroInteractions(todoServiceMock);
    }
}
```

Our unit test uses two static methods of the *TestUtil* class. These methods are described in the following:

- The *createStringWithLength(int length)* method creates a new *String* object with the given length and returns the created object.

- The *convertObjectToJsonBytes(Object object)* method converts the object given as a method parameter into a JSON document and returns the content of that document as a *byte array*.

The source code of the *TestUtil* class looks as follows:

```java
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.http.MediaType;

import java.io.IOException;
import java.nio.charset.Charset;

public class TestUtil {

    public static final MediaType APPLICATION_JSON_UTF8 = new MediaType(MediaTyp

    public static byte[] convertObjectToJsonBytes(Object object) throws IOExcept
        ObjectMapper mapper = new ObjectMapper();
        mapper.setSerializationInclusion(JsonInclude.Include.NON_NULL);
        return mapper.writeValueAsBytes(object);
    }

    public static String createStringWithLength(int length) {
        StringBuilder builder = new StringBuilder();

        for (int index = 0; index < length; index++) {
            builder.append("a");
        }

        return builder.toString();
    }
}
```

Test 2: Todo Entry Is Added to The Database

The second unit test ensures that our controller is working properly when a new todo entry is added to the database. We can write this test by following these steps:

1. Create a new *TodoDTO* object by using our test data builder. Set “legal” values to the *title* and *description* fields.
2. Create a *Todo* object which is returned when the *add()* method of the *TodoService* interface is called.
3. Configure our mock object to return the created *Todo* object when its *add()* method is called and a *TodoDTO* object is given as a parameter.
4. Execute a *POST* request to url ‘/api/todo’. Set the content type of the request to ‘application/json’. Set the character set of the request to ‘UTF-8’. Transform the created *TodoDTO* object into JSON bytes and send it in the body of the request.

5. Verify that the HTTP status code 200 is returned.
6. Verify that the content type of the response is ‘application/json’ and its content type is ‘UTF-8’.
7. Get the *id* of the returned todo entry by using the JsonPath expression *$.id* and verify that the *id* is 1.
8. Get the *description* of the returned todo entry by using the JsonPath expression *$.description* and verify that the *description* is “description”.
9. Get the *title* of the returned todo entry by using the JsonPath expression *$.title* and ensure that the *title* is “title”.
10. Create an *ArgumentCaptor* object which can capture *TodoDTO* objects.
11. Verify that the *add()* method of the *TodoService* interface is called only once and capture the object given as a parameter.
12. Verify that the other methods of our mock object are not called during our test.
13. Verify that the *id* of the captured *TodoDTO* object is null.
14. Verify that the *description* of the captured *TodoDTO* object is “description”.
15. Verify that the *title* of the captured *TodoDTO* object is “title”.

The source code of our unit test looks as follows:

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.ArgumentCaptor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;

import static junit.framework.Assert.assertNull;
import static org.hamcrest.Matchers.is;
import static org.junit.Assert.assertThat;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilder
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {TestContext.class, WebAppContext.class})
@WebAppConfiguration
public class TodoControllerTest {

    private MockMvc mockMvc;

    @Autowired
    private TodoService todoServiceMock;

    //Add WebApplicationContext field here.

    //The setUp() method is omitted.

    @Test
    public void add_NewTodoEntry_ShouldAddTodoEntryAndReturnAddedEntry() throws E
        TodoDTO dto = new TodoDTOBuilder()
                .description("description")
                .title("title")
                .build();

        Todo added = new TodoBuilder()
                .id(1L)
                .description("description")
                .title("title")
                .build();

        when(todoServiceMock.add(any(TodoDTO.class))).thenReturn(added);

        mockMvc.perform(post("/api/todo")
                .contentType(TestUtil.APPLICATION_JSON_UTF8)
                .content(TestUtil.convertObjectToJsonBytes(dto))
        )
                .andExpect(status().isOk())
                .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
                .andExpect(jsonPath("$.id", is(1)))
                .andExpect(jsonPath("$.description", is("description")))
                .andExpect(jsonPath("$.title", is("title")));

        ArgumentCaptor<TodoDTO> dtoCaptor = ArgumentCaptor.forClass(TodoDTO.clas
        verify(todoServiceMock, times(1)).add(dtoCaptor.capture());
        verifyNoMoreInteractions(todoServiceMock);

        TodoDTO dtoArgument = dtoCaptor.getValue();
        assertNull(dtoArgument.getId());
        assertThat(dtoArgument.getDescription(), is("description"));
        assertThat(dtoArgument.getTitle(), is("title"));
    }
}
```

If you wan to get up-to-date information about writing unit tests for Spring and Spring Boot web apps, you should take a look at my Test With Spring course:

CHECK IT OUT >>

Summary

We have now written unit tests for a REST API by using the Spring MVC Test framework. This tutorial has taught us four things:

- We learned to write unit tests for controller methods which read information from the database.
- We learned to write unit tests for controller methods which add information to the database.
- We learned how we can transform DTO objects into JSON bytes and send the result of the transformation in the body of the request.
- We learned how we can write assertions for JSON documents by using JsonPath expressions.

As always, you can get the example application of this blog post from Github. I recommend that you check it out because it has a lot of unit tests which were not covered in this blog post.

RELATED POSTS

WEEKLY /

Java Testing Weekly 43 / 2016

WEEKLY /

Java Testing Weekly 46 / 2016

WEEKLY /

Java Testing Weekly 16 / 2017

---

138 comments… add one

Lemrabet Link

August 22, 2013, 13:50

Hi Petri,

Thank you for this very useful

following step by step your recommendations

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath:rtls-management-test-
context.xml", "classpath:rtls-management-application.xml" })
@WebAppConfiguration
@TransactionConfiguration(defaultRollback = true, transactionManager =
"hibernatetransactionManager")
@Transactional
public class UserRestServiceTest {

Logger logger = Logger.getLogger(UserRestServiceTest.class.getName());
```

```
private MockMvc mockMvc;

@Autowired
private UserService userService;

@Autowired
private WebApplicationContext webApplicationContext;

@Before
public void setUp() {
// We have to reset our mock between tests because the mock objects
// are managed by the Spring container. If we would not reset them,
// stubbing and verified behavior would "leak" from one test to
another.
Mockito.reset(userService);

mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext)
.build();
}

private User addUser() {
logger.info("-> addUser");
User user = new User();
long id = 1;
user.setId(id);
user.setEnabled(true);
user.setFirstname("youness");
user.setUsername("admin");
user.setName("lemrabet");
user.setPassword("21232f297a57a5a743894a0e4a801fc3");
user.setEmail("youness.lemrabet@gmail.com");
logger.info(" findAllUsers");
User user = addUser();
// stubbing
when(userService.findAll()).thenReturn(Arrays.asList(user));
mockMvc.perform(get("/get/all"))
.andExpect(status().isOk())
.andExpect(
content().contentType(TestUtil.APPLICATION_JSON_UTF8))
.andExpect(jsonPath("$", hasSize(2)))
.andExpect(jsonPath("$[0].id", is(1)))
.andExpect(jsonPath("$[0].enabled", is(true)))
.andExpect(jsonPath("$[0].firstname", is("youness")))
.andExpect(jsonPath("$[0].username", is("admin")))
.andExpect(jsonPath("$[0].name", is("lemrabet")))
.andExpect(
jsonPath("$[0].password",
is("21232f297a57a5a743894a0e4a801fc3")))
.andExpect(
jsonPath("$[0].email", is("youness.lemrabet@gmail.com")));

verify(userService, times(1)).findAll();
verifyNoMoreInteractions(userService);

logger.info("<- findAllUsers");
}

}
```

I get the following error :

Failed tests:

findAllUsers(com.smartobjectsecurity.management.rest.user.UserRestServiceTest): Status expected: <200> but was: <404>

Thank you for your help

REPLY

Petri Link

August 22, 2013, 17:10

The status code 404 means that the tested controller method was not found. There are typically two reasons for this:

1. The request is send to wrong url (typo in the request mapping).
2. The controller class is not found during component scan. Check out that your component scan configuration is correct.

I hope that this answered to your question.

REPLY

Amishi Shah Link

November 8, 2014, 00:25

Hi Petri,

Thanks for the wonderful post. I have implemented a similar kind of test in my environment but I am getting the 404 error. I have checked for the URL as well as included the @ComponentScan(basePackages = {“somepackage”}) .

Do you recommend any other thing to be taken care of?

Thanks in anticipation.

Regards,
Amishi Shah

REPLY

Petri Link

November 9, 2014, 20:43

Hi,

Usually when you get a 404 response status, the reason is that the URL is not correct or the controller class is not found during component scan. You mentioned that you checked the URL and configured the Spring container to scan the correct package.

Did you remember to annotate the controller class with the `@Controller` or `@RestController` annotation?

REPLY

Sachin Link

April 10, 2017, 16:36

Hi Petri, I have proper component scan, right url and using @controller annotation for controller class, but still facing 404 issue, do you know any other possible reason.
But I am using @Path("/list") instead of @RequestMapping in controllers, will it make any difference?
Thanks

REPLY

Petri Link

April 10, 2017, 18:43

Well, to be honest, I have never used JAX-RS annotations in my Spring web applications even though it seems that you are able to do it if you use Spring Boot. In other words, I would use the `@RequestMapping` annotation.

REPLY

Sachin Link

April 12, 2017, 09:53

So you think this tutorial or any thing else will work with JAX-RS annotations :(?
Because now we can’t switch to @RequestMapping annotation.
Any help would be highly appreciated.

Petri Link

April 13, 2017, 08:52

Hi,

I did some digging and it seems that you cannot use Spring MVC Test if you use JAX-RS because Spring MVC Test framework uses its own mock implementation of the Servlet API (instead of deploying the application to a servlet container). That being said, you have still several other options:

- Use `RestTemplate` in your test class
- Use the Jersey Test Framework
- Use Rest Assured

Note that each one of these tools require that you deploy your application to a servlet container before you run your tests.

Lukman Link

October 20, 2013, 18:24

Hi
I was wondering how I could write a test for testing xml response instead of json . Any sample code would be appreciated.

I learnt from a lot from this tutorial thanks.

REPLY

Petri Link

October 21, 2013, 00:37

Hi Lukman,

The answer of this StackOverflow question answers to your question as well. Remember that you have to add XmlUnit to your *pom.xml* file (the correct scope for this dependency is `test`).

REPLY

Lukman Link

October 21, 2013, 08:17

Thanks Petri

REPLY

Chris Link

February 5, 2014, 23:43

hi Petri
I have the code from the blog. this is probably something silly but would like your input if you can.
Everything from the blog is almost the same except the integration test @ContextConfiguration which I changed to
@ContextConfiguration(locations = {“classpath:spring/root-Context.xml”, “classpath:spring/app/servlet-context.xml”})
as I made some changes to the files.

The integration test run and executes through the controller fine but fails with a response content type

Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 4.619 sec <<< FAILURE!
testController(my.tests.web.SomeControllerIntegrationTest) Time elapsed: 4.331 sec <<<
FAILURE!
java.lang.AssertionError: Content type expected: but was:
at org.springframework.test.util.AssertionErrors.fail(AssertionErrors.java:60)
at org.springframework.test.util.AssertionErrors.assertEquals(AssertionErrors.java:89)
at
org.springframework.test.web.servlet.result.ContentResultMatchers$1.match(ContentResultM
atchers.java:71)
at org.springframework.test.web.servlet.MockMvc$1.andExpect(MockMvc.java:141)
The mockMvc object is expecting Json in the response but it is not so. Thanks in advance

REPLY

Petri Link

February 6, 2014, 00:07

Hi Chris,

There are no silly questions! I assume that you have tested the controller method manually and verified that it returns a JSON document in the request body.

Are you using Spring 4? I remember that I was having a similar problem in a project which uses Spring 4 instead of Spring 3.2. I was able to solve it by setting the Accept request header:

```
mockMvc.perform(get("/api/srv/maincontractor")
        .accept(MediaType.APPLICATION_JSON)
)
//Add assertions here
```

Is it possible to see the request mapping of the tested controller method (I don't need to see the actual implementation)?

REPLY

Chris Link

February 6, 2014, 00:44

Hi Petri,

Yes. This is the controller method under test. It returns a list of terms in json format. If i do a mvn tomcat:run and test it using rest client I can see that it does.

```
@RequestMapping(value = "/terms/", method = RequestMethod.GET)
public ModelAndView listTerms() {…}
```

The version of spring are newer and so is the JSon mapper. I wonder why there is such a version to version compatibility issue with these both but this another animal.
3.2.0.RELEASE
1.9.2

REPLY

Chris Link

February 6, 2014, 01:19

Hi Petri

Thank you for your input previously. I think I see how I managed to screw this up. In my spring configuration. I am using ..

which I changed now to

.. turns out surprisingly that it actually does the conversion from pojo to json but renders it in plain text – which makes sense; so that it can be captured in a jsp etc.,

except it was not obvious.
Many thanks again.

REPLY

Petri Link

February 6, 2014, 20:14

Hi Chris,

If you want to simplify your code a bit (I assume that you create the `ModelAndView` object inside the controller method), you can change the controller method to look like this:

```
@RequestMapping(value = "/terms/", method = RequestMethod.GET)
@ResponseBody
public List<Terms> listTerms() {...}
```

This simplifies the code of your controller method and more importantly, it fixes your content type problem.

REPLY

Abhay Link

December 26, 2016, 14:38

Hi Petri,

Thanks for giving all information i need you help please tell me how i can write the junit test preparation on following code.

```
@RequestMapping(value="/search-calendars/{resourceTypeCode}",
method=RequestMethod.GET , produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<List> searchResource( @PathVariable String resourceTypeCode,
@RequestParam(value = "resourceId", required = false) Integer resourceId) throws
SunshineException {
if(StringUtils.isEmpty(resourceTypeCode)){
throw new SunshineException("Resource code should not be Empty");
}
SearchCalendarDTO resourceTypeDTO = new
SearchCalendarDTO(resourceTypeCode,resourceId );
List searchlist = calendarService.searchCalendars(resourceTypeDTO);
return new ResponseEntity(searchlist, HttpStatus.OK);
}
}
```

Thanks,

Abhay

REPLY

Ahmad Link

March 6, 2014, 13:00

Hi Petri,

First, I want to thank you for this great tutorial, it helps me a lot.

I found an weird issue in bean validation, spring mvc controllers' unit tests.
To reproduce you juste have to upgrade your dependency for the jsp api to the lastest version and execute your first test in TodoControllerTest
(`add_EmptyTodoEntry_ShouldReturnValidationErrorForTitle()`)
Upgrading that dependency will make this test fail, because I don't know how but it kinda bypass bean validation. So you should get a 201 instead of 400.

Can you confirm this? Is it a known issue?

Thanks in advance for your time.

REPLY

Petri Link

March 10, 2014, 19:34

Hi Ahmad,

actually I couldn't reproduce this issue. Did you upgrade any other dependencies to newer versions?

REPLY

Ahmad Link

March 21, 2014, 11:51

Hi Petri,

Thanks for your reply, yes actually I upgraded other dependencies (sorry I forgot to mentionned):

– Spring: 4.0.2.RELEASE

– Hibernate Validator: 5.1.0.Final

– Servlet Api: 3.1

If I remember well, I fixed the issue, by adding a dependency to javaee-web-api: 7.0.

REPLY

Ahmad Link

March 21, 2014, 11:53

Petri,

If you still can't reproduce it, I can send you a pull request with the pom.xml if you want.

Thanks for your time.

REPLY

Petri Link

March 22, 2014, 14:37

Hi Ahmad,

actually I happened to run into this issue when I updated Hibernate Validator to version 5.1.0.Final. I was able to solve this issue by using Hibernate Validator 5.0.3.Final. I suspect that Spring Framework 4 doesn't support Hibernate Validator 5.1.0.Final but I haven't been able to confirm this.

REPLY

Angelos Link

June 6, 2014, 18:21

thank you man for this, I was getting very frustrated!!

Petri Link

June 6, 2014, 20:24

You are welcome! I am happy to hear that I was able to help you out.

Magnus Link

June 4, 2014, 15:02

Petri, these examples are really nice! I particularly like how clean and fast the unit test runs of your controller's.

Do you do the same type of setup in Spring 4?

If you have a simple setup and use SpringBoot org.springframework.boot.SpringApplication.Application to bootstrap the application, is there anything significant that would change with your solution?

Also, I thought I saw that spring-mvc-test was added into the Spring framework in Spring 3.2 which of course was released later after your blog post about this. I assume you would be using that instead of spring-mvc-test separately or did they make any changes that creates overhead that slows down the unit tests significantly?

Thanks,
Magnus

REPLY

Petri Link

June 4, 2014, 20:15

Actually the example application of this blog post uses Spring Framework 3.2.X and Spring MVC Test framework. The standalone project is called spring-test-mvc, and it is used in my blog posts which talk about the integration testing of Spring MVC Applications. There really isn't any reason to use it anymore (unless you have to use Spring 3.1).

I use the same setup for testing web applications which use Spring Framework 4, but I haven't used Spring Boot yet so I cannot answer to your question. I am planning to write a Spring Boot tutorial in the future, and I will definitely address this issue in that tutorial.

REPLY

Jaxox Link

June 24, 2014, 21:31

Hi Petri

Thanks for the tutorial.

I am having some issue to get it start. I have post the question to the Stackoverflow.

http://stackoverflow.com/questions/24393684/constructor-threw-exception-nested-exception-is-java-lang-noclassdeffounderror

let me know if you need more info, thanks.

REPLY

Petri Link

June 24, 2014, 22:20

Hi,

It seems that you already got the correct answer to your StackOverflow question. Follow the instructions given in that answer and you should be able to solve your problem.

REPLY

Paul Statham Link

August 20, 2014, 13:12

Hi Petri,

I have a question about the your object creation. In the controller test class you build your test data using a builder, but when you're converting your data with createDTO you don't use a builder here. Is there a reason for this?

Thanks,
Paul

REPLY

Petri Link

August 20, 2014, 13:48

Thank you for asking such a good question.

I cannot remember what my reason was when I made the decision to use the builder pattern only in my tests (I probably didn't have any reason for this), but nowadays I follow these "rules":

- If the DTO is read-only, I will use a builder pattern and mark all its field as `final`. Also, I won't add setters to this DTO. This is handy if I want to just transform read-only information without exposing the internal data model of my application to the outside world.
- If the information of the DTO can be modified (e.g. a form object or an object that is read from the request body), I will not use a builder pattern in my application. The reason for this is that Spring provides the object to my controller method when it is processing a request. When I return information back to the client of my API, I use a library to map entities into data transfer objects so that I can eliminate unnecessary "boiletplate" code (check out Dozer, jTransfo, and ModelMapper). However, I do use builder pattern in my tests because this way I can write tests that speak the language that is understood by domain experts.

REPLY

Akshay Link

September 28, 2014, 20:23

Hi Petri,

Nice tutorial.
When I tried to use : https://github.com/pkainulainen/spring-mvc-test-examples/tree/master/controllers-unittest I found that TodoBuilder class was missing.

Can you guide me what went wrong? (Version I have used : https://github.com/pkainulainen/spring-mvc-test-examples/commit/93ea56b879a0af64f862c935de188cf4860a9dd0)

Thanks,
Akshay

REPLY

Petri Link

September 28, 2014, 20:30

Hi Akshay,

Unfortunately I am not sure what is going on because I can find the `TodoBuilder` class. :(

REPLY

damn Link

October 7, 2014, 09:42

without TodoBuilder this are piece of shit…

REPLY

Petri Link

October 7, 2014, 10:23

Hi,

thanks for the feedback! I left a few trivial classes out from this blog post since I assumed that those who are interested in them, will read them on Github.

Anyway, if you want to get the source code of those test data builder classes, just click the links below:

- The source code of the `TodoBuilder` class
- The source code of the `TodoDTOBuilder` class.

REPLY

Ramakrishna Link

November 29, 2014, 23:20

Hi Petri, Superb Tutorial !!
I have a question.
While verifying the response, we are doing an inline compare of values like:
.andExpect(jsonPath("$[0].id", is(1)))
.andExpect(jsonPath("$[0].description", is("Lorem ipsum")))
.andExpect(jsonPath("$[0].title", is("Foo")))

is it possible to collect this response or parts of response into an object. So that I can write a seperate method to pass on the json response to a separate method for asserting values

REPLY

Ramakrishna Link

November 29, 2014, 23:43

Okay, I have figured this out.

```
MvcResult result = mockMvc.perform(post("/admin/state/getById/" + state.getId()))
.andExpect(status().isOk())
.andExpect(content().contentType(TestUtil.APPLICATION_JSON))
.andDo(print())
.andReturn();

MockHttpServletResponse response = result.getResponse();
String reponseJSON = response.getContentAsString();
```

this responseJSON now is a string representation of the returned JSON,
Now we can use it in whatever way we want, eg: using Jackson convert it into the relevant DTO and compare values.

REPLY

Petri Link

November 30, 2014, 15:58

Hi,

I am happy to hear that you were able to solve your problem. By the way, is there some reason why you want to do this? Do you like to write assertions for real objects instead of using jsonpath?

REPLY

Abrar Link

December 29, 2014, 15:27

Hi petri, thanks for nice tutorial..

i have a very simple controller which return JSON array here is my test class to test that controller :

```
@Test
public void findAllObjects() throws Exception {
    Components first = new ExBuilder()
            .cname("asdfasf")
            .mdesc("asdcb")
            .cdesc("asdfa")
            .ccode("asdf")
            .unitrateusd(24)
            .build();

    when(exampledao.list()).thenReturn(Arrays.asList(first));

    mockMvc.perform(get("/getdata"))
        .andExpect(status().isOk())
        .andExpect(content().contentType(TestUtil.APPLICATION_JSON))
        .andExpect(jsonPath("$", hasSize(0)))
        .andExpect(jsonPath("$", hasSize(1)))
        .andExpect(jsonPath("$[0].cname", is("asdfasf")))
        .andExpect(jsonPath("$[0].mdesc", is("asdcb")))
        .andExpect(jsonPath("$[0].cdesc", is("asdfa")))
        .andExpect(jsonPath("$[0].ccode", is("asdf")))
        .andExpect(jsonPath("$[0].unitrateusd", is(24)));


    verify(exampledao, times(1)).list();
    verifyNoMoreInteractions(exampledao);
}
```

here is my java controller which i am trying to test:

```
@RequestMapping(value="/getdata" , method=RequestMethod.GET)
public List listContact(ModelAndView model) throws IOException {
    System.out.println("dfadfajfajfa");

    List listContact;
    listContact= exampeldao.list();

    System.out.println("hiii i made a call  but i dnt have any data.....");

    return listContact;
}
```

i could not able to call list() method, which is defined in another class.

when i run the test i am getting SecurityException:org.hamcrest.Matchers signature information does not match signature information of other classes in same package.

can u help me to solve this issue..

thank you

REPLY

Petri Link

December 29, 2014, 17:11

First, your test looks fine to me and it should work (assuming that you have annotated your controller with the `@RestController` annotation).

Are you trying to run your unit tests by using Eclipse? The reason why I am asking this is that I found this bug report and one commenter suggests that:

> The hamcrest-library-1.3.jar in maven dependencies had collision with Eclipse embedded Hamcrest.

He also found a solution to this problem (but unfortunately it sounds like a hack):

> So I found alternative simple solution: rename the file $ECLIPSE_HOME\plugins\org.hamcrest.core_1.3.0.v201303031735.jar to something like *.bak or remove the file.
>
> Eclipse ignores the embedded Hamcrest :) and Maven dependencies will be used instead.

Unfortunately I cannot provide you any further advice since I haven't used Eclipse for several years.

On the other hand, if you this happened when you tried to run your unit tests by using Maven, let me know and I will take a closer look at this problem.

REPLY

Abrar Link

December 30, 2014, 07:29

hiiii petri…
when i am trying to call any uri of controller i am getting some security error:
class:org.hamcrest.Matchers signature information does not match signature information of other classes in same package.
can u tell me why i am getting this error
Thankl You

REPLY

Petri Link

December 30, 2014, 08:41

Hi,

I answered to this question in this comment.

REPLY

Abrar Link

January 2, 2015, 15:00

Thanks for instant response…
i am able to solve that security issue error, but my main issue is i am not able to execute this "listContact= exampeldao.list();" statement from my test code. If i define the list() inside the controller class i am able to get the data. but if i define list() in some other class and if i try to access via instance , i could not able to do. can u tell me where probably i am making mistake.

Thank You

REPLY

Petri Link

January 2, 2015, 15:37

Do you mean that you cannot create a mock object that returns something when the `list()` method of `ExampleDao` object is invoked?

If so (and you want to get good advice), I have to see the source code of the tested method and the source code of your unit test.

If I have to guess, I would say that you haven't configured the mock object correctly or the `list()` method of the `ExampleDao` object is not invoked during your unit test.

REPLY

Abrar Link

January 3, 2015, 13:48

hi petri…

This is my list() method

```
public class ExampleDaoImpl  implements Example {

    @Autowired
    DataSource dataSource;

    @Autowired
    Example exampledao;

    public List list() {
        //removed implementation as irrelevant
    }
}
```

This method is declared in "Example" interface and which is implemented by ExampleDao class.

Here is my controller:

```
@RequestMapping(value="/getdata" , method=RequestMethod.GET)
public List listContact(ModelAndView model) throws IOException{
    System.out.println("checking for unit test");
    List listContact;
    listContact= exampledao.list1();
    System.out.println("i dont have any data....");
    return listContact;

}
```

here is my complete Testclass:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={
    "classpath:/rest-servlet.xml",
    "classpath:/test-context.xml"
})
@WebAppConfiguration
public class RestTest {

    private MockMvc mockMvc;

    @Autowired
    private Example exampledao;

    @Autowired
    private WebApplicationContext webApplicationContext;

    @Before
    public void setUp() {
        Mockito.reset(exampledao);
        mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationC
                        .build();
    }

    @Test
    public void findAllObjects() throws Exception {
        Components first = new TodoBuilder()
            .cname("asdfasf")
            .mdesc("asdcb")
            .cdesc("asdfa")
            .ccode("asdfasf")
            .unitrateusd(24)
            .build();

        when(exampledao.list1()).thenReturn(Arrays.asList(first));

        mockMvc.perform(get("/getdata"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(TestUtil.APPLICATION_JS
            .andExpect(jsonPath("$", hasSize(1)))
            .andExpect(jsonPath("$[0].cname", is("asdfasf")))
            .andExpect(jsonPath("$[0].mdesc", is("asdcb")))
            .andExpect(jsonPath("$[0].cdesc", is("asdfa")))
            .andExpect(jsonPath("$[0].ccode", is("asdfasf")))
            .andExpect(jsonPath("$[0].unitrateusd", is(24)));

        verify(exampledao, times(1)).list1();
        verifyNoMoreInteractions(exampledao);
    }
}
```

And here is my configuration files:

test-context.xml:

i don't no where i m committing mistake :-) help me to fix this issue.. :-)

Thank You

Petri Link

January 5, 2015, 20:41

Wordpress doesn't allow you to paste XML in comments => Wordpress removed the content of your configuration files. You should paste those files to Pastebin.com and add the link to your comment.

Anyway, the only problem I can see is that the `ExampleDaoImpl` class has a method called `list()` but your test mocks the `list1()` method.

Also, it would be helpful if you could clarify your problem a bit. Do you mean that your test fails because:

1. The `list1()` method returns an empty list?
2. The `list1()` method is not invoked at all.

Abrar Link

January 6, 2015, 09:24

i could not able to invoke list1() method at all…
here is my link of configuration files…
http://pastebin.com/bjWLDJPA
http://pastebin.com/psN6Zb5J

Thank You

Petri Link

January 6, 2015, 18:44

I took a quick look at your configuration files and I have no idea why you don't get an exception because both configuration files configure the `exampleDao` bean.

If you use the web application context based configuration, you have to split the application context configuration files of your application so that you can use only some of them in your unit tests (get more information about this).

In other words, you have two options:

1. You can split your configuration file into multiple configuration files and configure only the mock bean in the test specific configuration file. After you have done this, you can configure your unit tests by using the correct application context configuration files.
2. You can configure the web layer and the required mock beans in the *test-config.xml* file and configure your unit tests by using only that file.

I hope that this answered to your question.

Patrick Link

January 9, 2015, 21:52

Hi Petri,

instead of excluding hamcrest-core from junit and adding hamcrest-all as a dependency you could just add hamcrest-library as a dependency.

Regards,
Patrick

REPLY

Petri Link

January 9, 2015, 22:37

Hi Patrick,

Good catch! I am going to rewrite my Spring MVC Test tutorial during this year (2015), and I will include this fix to my new tutorial.

Thank you for pointing this out!

REPLY

Sachin Link

April 10, 2017, 16:59

Have you rewrite this tutorial, if yes please share the url.

Thanks

REPLY

Petri Link

April 10, 2017, 18:45

Hi,

I was supposed to rewrite this tutorial in 2015, but I decided to concentrate on other tutorials because this one was good enough.

REPLY

Patrick Link

January 10, 2015, 00:29

Hi again,

just another Point: are you always programming your Builders like the TodoBuilder? I mean the fact that you are not able to create multiple Todo instances with one Builder instance (build only returns the reference to model instance).

I prefer to copy all fields of the Todo model to the builder class and create a new Todo instance with every build method call. This is especially useful when you want to create multiple instances with e.g. only one difference.

What do you think about it?

Regards,
Patrick

REPLY

Petri Link
January 10, 2015, 18:09

Hi,

Actually I don't use the approach that is used in this blog post anymore. I "copy" the fields of the constructed object to the test data builder class (and of course to the "real" builder), and create a new object in the `build()` method.

My main reason for doing this was that often objects have a many mandatory properties, and the "real" builder class verifies that everyone of them is "valid". If these fields are not valid, it will throw the correct `RuntimeException`. When I copy these fields to the test data builder class, I also set default values to these mandatory fields (e.g. if the field is a `String` field, I use string "NOT_IMPORTANT"). This way I can set only those fields that are relevant for the test case.

This is one of those things that will change when I update my Spring MVC Test tutorial => don't change your way of doing things since it is better than the approach described here.

REPLY

Dinda Link

June 9, 2015, 10:14

Hi Petri,

First of all, your tutorial is really great. I am new to unit testing (or to spring in general) and I learned a lot from this tutorial.

I was trying to test a controller my self, using the stand alone setup (because I hate xml), but then there's an error in the .andExpect(jsonPath("$.var3", is("123456"))) line. The error said "json can not be null" and it throws IllegalArgumentException.
And I've tried almost everything and can't figure out why it happened. Does it have anything to do with me not using xml? or what

Here's the controller I want to test

```
@RestController
@RequestMapping("/test")
public class MyController {

    @Autowired
    MyService myService;

    @RequestMapping(
            value={"/dosomething"},
            method=RequestMethod.POST
    )
    @ResponseBody
    public MyResponse doSomething(@RequestBody MyRequest request) {
        return myService.doSomething(request);
    }
    //another methods here
}
```

Here’s my test controller

```
public class MyControllerTest {

    @Mock
    private MyService myService;

    @InjectMocks
    private MyController myController;

    private MockMvc mockMvc;

    public MyControllerTest() {
    }

    @Before
    public void setUp() {
        MockitoAnnotations.initMocks(this);
        mockMvc = MockMvcBuilders.standaloneSetup(myController).build();
    }

    @Test
    public void testDoSomething() throws Exception {
        System.out.println("do something");
```

```
            MyRequest myRequest = new MyRequest();
            myRequest.var1("abc");
            myRequest.var2("def");

            MyResponse myResponse = new MyResponse();
            myResponse.var3("123456");
            myResponse.var4("qwerty");
            //in reality (when tested from postman), the var4 value is and supposed to

            when(myService.doSomething(myRequest)).thenReturn(myResponse);

            mockMvc.perform(post("/test/dosomething")
                    .content(new ObjectMapper().writeValueAsBytes(myRequest))
                    .contentType(MediaType.APPLICATION_JSON)
            )
                    .andExpect(status().isOk())
                    .andExpect(jsonPath("$.var3", is("123456")))
                    ;
        }
    }
```

Thank you

REPLY

Petri Link

June 9, 2015, 18:59

Hi,

Replace this line:

```
when(myService.doSomething(myRequest)).thenReturn(myResponse);
```

With this line:

```
when(myService.doSomething(isA(MyRequest.class))).thenReturn(myResponse);
```

The problem is that the `MyRequest` object that is passed to the `doSomething()` method of the `MyService` class is not the same object which you create in your test method.

The reason for this is that Spring MVC creates a new `MyRequest` object when it resolves the method parameters of your controller method (and reads the field values from the request body).

Did this solve your problem?

REPLY

Dinda Link

June 16, 2015, 14:09

OMG, thank you!
That did solve my problem. And apparently, it was that simple.

I have some other questions if you don't mind.
I was wondering how to pass parameter (Object) annotated with @ModelAttribute when unit testing a method in Controller class.
For example, I have this GET method

public Response doSomething(@ModelAttribute Request request).
How should I unit test that kind of method? What should I do in the mockMvc.perform(get("/test/something")) ?

Thank you.

REPLY

Petri Link

June 16, 2015, 21:08

Thank you for your kind words. I really appreciate them.

> I was wondering how to pass parameter (Object) annotated with @ModelAttribute when unit testing a method in Controller class.

It depends.

If you want to write a unit test for a controller method that processes form submissions, you should read my blog post that describes how you can write unit tests for "normal" Spring MVC controllers.

If you want to write a unit test that passes "other" objects to your controller method, the answer to your question depends from the way you "initialize" the `@ModelAttribute` object (see Using @ModelAttribute on a method argument for more details). Could you provide an example that provides more details about your problem?

REPLY

candy Link

July 8, 2015, 08:07

Hi Petri,
Your tutorial is good and thank you for your tutorials, I am new to this MVC testing and I learned a lot from this tutorial.I have some doubts.

In this tutorial u explained about one @pathvariable only
mockMvc.perform(get("/api/todo/{id}", 1L)) ok fine,how to pass multiple @pathvariables in url ,for ex /add/{id}/{page}

Thank you

REPLY

Petri Link

July 8, 2015, 09:59

Hi,

> how to pass multiple @pathvariables in url

You can simply pass the url template and variables as method parameters to the factory method that creates the used `RequestBuilder` object (see the Javadoc of the `MockMvcRequestBuilders` class).

For example, if you want to send get a GET request by using url template: '/api/person/{personId}/todo/{todoId}' when the `personId` is `1L` and the `todoId` is `99L`, you have to use the following code:

```
mockMvc.perform(get("/api/person/{personId}/todo/{todoId}", 1L, 99L));
```

As long as you remember to pass the path variable values in the same order than the path variables, you should be fine. If you have any further questions, don't hesitate to ask them.

REPLY

candy Link

July 8, 2015, 11:59

yes,its working fine.Thank you

I have one more questions,In my url am passing pathvariables ,HttpSession and Model for form values am clear about how to pass pathvariables and HttpSession but how to pass that Model.

Am passing those all form values in model attribute but still am getting Null Pointer expection

REPLY

Petri Link

July 8, 2015, 12:13

Hi,

If you want to set the property values of a form object (a controller method parameter that is annotated with the `@ModelAttribute` annotation), you should set the property values by using the `param()` method of the `MockHttpServletRequestBuilder` class.

I have written a blog post that describes how you can write unit tests for controller methods which processes form submissions. It describes how you can use this method in your unit tests.

Let me know if this solved your problem. :)

REPLY

Anonymous Link

July 8, 2015, 14:15

Hi,

thanks for that blog post thats very helpfull .

how to test private methods?

Thanks

Petri Link

July 8, 2015, 17:57

Hi,

You cannot (and should not) test private methods. You should only test your public API. If the tested methods use private methods (and often they do), the private methods will be “tested” as well.

On the other hand, if you have to test the functionality of a private method, you should move it to another class and make it public. Or you could make it protected (this is useful if you are working with legacy code).

candy Link

July 9, 2015, 09:44

Hi,

ok thank you.

Petri Link

July 9, 2015, 09:59

You are welcome!

---

candy Link

July 9, 2015, 13:12

Hi
Am trying to get userID form session but am getting null.

Thank you

REPLY

Petri Link

July 9, 2015, 16:24

Unfortunately I cannot know what is wrong. However, because you get `null`, it is very likely that the userID is not found from the session.

REPLY

Candy Link

July 9, 2015, 15:23

Hi,

How to test Spring MVC with tiles

Thank you

REPLY

Petri Link

July 9, 2015, 16:30

I have never used Tiles, but I assume that you can write controller tests by using the techniques described in this blog post. There seems to be some differences though.

REPLY

Candy Link

July 10, 2015, 08:29

Ok, Thank you

REPLY

Petri Link

July 10, 2015, 10:48

You are welcome.

REPLY

Candy Link

July 10, 2015, 12:26

Hi,

In my controller i have HttpSession session as parameter and my using this session to get userid.
my mockMVC my passing mockSession as parameter but am getting null how to get userid using this seccion.

Thank you

REPLY

Petri Link

July 10, 2015, 12:33

Hi,

Another reader had the exact same problem. Check out this thread.

REPLY

candy Link

July 14, 2015, 12:47

Hi,

This the way to get userid from session.

```
UserBean userBean = new UserBean();
userBean.setUserId("111");
this.mockMvc.perform(post("/getuser")
        .sessionAttr("userbean",userBean))
```

Thanks

REPLY

Petri Link

July 14, 2015, 12:52

You are welcome! I am happy to hear that you were able to solve your problem.

REPLY

Candy Link

July 17, 2015, 10:33

Hi,
I am getting java.lang.ClassNotFoundException: org.springframework.test.web.server.MockMvc exception while building the maven project.
could you please help to resolve this issue.

Thank you

Petri Link

July 17, 2015, 12:53

Which Spring version are you using? The reason why ask this is that I noticed that the error message has the "old" package of the `MockMvc` class.

If you are writing tests for an application that uses Spring 3.1.X, this package is correct. If this is the case, you need to ensure that you have added the standalone spring-test-mvc dependency to your *pom.xml* file.

If you are writing tests for an application that uses Spring 3.2.X or never, you should add the sprint-test library to your *pom.xml* file. Also, remember that in this case you must remove the spring-test-mvc dependency from your POM file.

Candy Link

July 17, 2015, 14:13

I am using spring 4.0.1.RELEASE

Thank you

Petri Link

July 17, 2015, 17:38

You have to ensure that your *pom.xml* contains ONLY the spring-test dependency. If your *pom.xml* contains the spring-test-mvc dependency, you need to remove it. Also, you need to fix the imports because the package of the `MockMvc` class has changed.

Candy Link

July 21, 2015, 09:50

Hi

Now its working fine.

Thank you

Petri Link

July 21, 2015, 11:52

You are welcome.

Candy Link

July 21, 2015, 12:14

Hi,

I am getting "The method when(T) in the type Mockito is not applicable for the arguments (void)" while testing void methods.

Thanks

REPLY

Candy Link

July 21, 2015, 13:24

Hi,

One more doubt whether tomcat is running or not while executing junit testcases.

Thanks

REPLY

Petri Link

July 21, 2015, 22:00

No. Tomcat is not running when you run unit tests that use the Spring MVC Test framework.

REPLY

Petri Link

July 21, 2015, 22:03

The Mockito documentation explains how you can mock `void` methods.

REPLY

Candy Link

July 22, 2015, 07:39

Ok Thank you very much Petri.

REPLY

Paul Link

November 16, 2015, 21:10

Thanks for the insightful and thorough tutorial . However, it appears that for several of the examples, the controller logic is not actually being tested. Consider the line:
when(todoServiceMock.findAll()).thenReturn(Arrays.asList(first, second));
Seems like it would be a more true unit test if the call service.findAll(); was mocked. In this way, we only mock the database service and actually execute the rest of the logic in findAll (which in this particular case is pretty minimal). Thoughts?

REPLY

Petri Link

November 18, 2015, 23:10

Hi Paul,

Thank your for your kind words. I really appreciate them. Also, thank you for writing an interesting comment. I will answer to it below:

> However, it appears that for several of the examples, the controller logic is not actually being tested. Consider the line:
> when(todoServiceMock.findAll()).thenReturn(Arrays.asList(first, second));
> Seems like it would be a more true unit test if the call service.findAll(); was mocked.

Do you mean that the service logic is not tested? The reason why I ask this is that these unit tests isolate the tested code by replacing its dependencies with mock objects. In other words, they test the controller logic but not the service logic (because the service is a mock object).

> Seems like it would be a more true unit test if the call service.findAll(); was mocked. In this way, we only mock the database service and actually execute the rest of the logic in findAll (which in this particular case is pretty minimal). Thoughts?

I assume that you want to mock the repository's `findAll()` method? This is entirely possible. It just means that you want to use a larger than unit I did. Typically I use small units and mock the external dependencies for two reasons:

- I think that these tests are easier to write and read.
- If my unit test fails, I know the exact location of the problem.

However, I am saying that this is the one and only way of doing things. It is just an approach that has served me well. If you think that a bigger unit size helps you to write better tests, you should give it a shot. You will notice pretty soon if it is a better approach than using small units. In fact, this example is so simple that the unit size probably don't matter at all because there really isn't a lot of logic.

P.S. Naturally I write integration and end-to-end tests as well.

P.P.S. You might want to check out my Writing Clean Tests tutorial. It should give you something to think about.

REPLY

Will Link

February 2, 2016, 00:05

Hi
Great tutorial.. thanks very much.
I am having an issue getting it working though and would be grateful for any advice..
I added the jsonPath 0.8.1 jar file to my class path manually through eclipse and now when running my test I get the following error.. On researching it suggest this jar file was compiled with a different jdk version, I am using java 5, could this be the problem?

```
java.lang.UnsupportedClassVersionError: Bad version number in .class file
    at java.lang.ClassLoader.defineClass1(Native Method)
    at java.lang.ClassLoader.defineClass(ClassLoader.java:620)
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:124)
```

Update: I removed the irrelevant part of the stack trace – Petri

REPLY

Petri Link

February 2, 2016, 19:01

Hi,

The problem is that your JDK is too old. The latest version of JsonPath (2.1.0) requires JDK 1.6, and I assume that the version 0.8.1 requires it as well (otherwise you wouldn't get this error).

REPLY

Will Link

February 2, 2016, 20:36

Thanks for the reply. I though that may be the problem. Unfortunately I cannot upgrade my java version as the production server is not compatible. Do you recommend an alternative library I could use to make assertions about the returned json response. Thanks.

REPLY

Will Link

February 2, 2016, 21:08

Maybe I could use jackson object mapper to make assertions like..

```
MyClass myClass = new MyClass();
    myClass.setProp1("property1");
    when(myService.method1("property1")).thenReturn(myClass);
    String urlTemplate = "/myUrl/property1";

ResultActions results = mockMvc.perform(get(urlTemplate, new Object()))
        .andExpect(status().isOk())
        .andExpect(content().contentType("application/json;charset=UTF-8
MyClass response =
        mapper.readValue(results.andReturn().getResponse()
            .getContentAsString(), MyClass.class);

assertEquals("property1" response.getProp1());
```

REPLY

Petri Link

February 3, 2016, 22:47

Hi Will,

that is definitely one way to solve your problem. However, you can also “hide” these implementation details by creating a custom Hamcrest matcher. After you have created your custom matcher, you can use it for writing assertions for the returned JSON.

For example, your test case could look like this:

```
mockMvc.perform(get("/foo/bar"))
    .andExpect(content().string(hasJsonProperty("id", 1L)));
```

Adel Sassi Link

February 26, 2016, 12:33

Hello, thank you so much for your tutorial.

I have encountered an exception when makin unit test using mock:

REPLY

Adel Sassi Link

February 26, 2016, 12:36

Error: java.lang.AssertionError: Status expected: but was:

Contoller:

@RestController

@RequestMapping(value = “Univ”)

@ComponentScan(basePackages = {“com.back.controller”})

public class UniversityController {

@Autowired

private UniversityService universityService;

@RequestMapping(value = “/university/Hello”, method = RequestMethod.GET , produces = “application/json”)

public String sayHello() throws UniversityException {

```
return universityService.getName();
}
}
```

Class for test:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
“file:WebContent/WEB-INF/spring-config.xml”,”file:WebContent/WEB-
INF/AnnotationsDriven.xml”})
@WebAppConfiguration
public class TodoControllerTest {
@Autowired
@Qualifier(“universityForTest”)
private UniversityService universityServiceMock;
@Autowired
private WebApplicationContext webApplicationContext;
private MockMvc mockMvc;
@Before
public void setUp() {
Mockito.reset(universityServiceMock);
mockMvc =
MockMvcBuilders.webAppContextSetup(webApplicationContext).dispatchOptions(true)
.build();
}
@Test
public void tester() throws Exception {
when(universityServiceMock.getName()).thenReturn(“Hello Mock!!”);
mockMvc.perform(get(“/Univ/university/Hello”)).andExpect(status().isOk());
}
}
```

REPLY

Petri Link

February 26, 2016, 12:55

Hi,

Your unit test fails because your controller method doesn’t return HTTP response status 200 (OK). Replace the following code: `.andExpect(status().isOk())` with: `andDo(print())`. This prints the request and response into `System.out` and should help you to solve your problem.

REPLY

Vishal Link

April 15, 2016, 18:00

The last example in this blog illustrates a test case for ToDo object entry into the database. You have already verified returned object success using jsonPath. I do not understand why did we again retrieve the ToDo object using ArgumentCaptor and then again verified ID, Description and other fields. Is it not duplicated? I am sure there must be a reason. Can you please explain.

REPLY

Petri Link

April 16, 2016, 15:56

Good question. Actually I have to I have to admit that the test question sucks because it can fail for more than one reason (it has other flaws as well). If I would test that controller method now, I would write several unit tests for it. If you think about the requirements of the tested method, it is a somewhat clear that:

- The information of the saved todo entry must be returned as JSON.
- The correct information must be passed forward to the service method (that saves it).

Now, if you don’t capture the `TodoDTO` object that is passed to the service method, you cannot ensure that the correct information is passed to the service method.

REPLY

Bharat Link

April 17, 2016, 01:44

Hello,
Thanks for the very nice overview. Helped me a lot with understanding the test framework.

Question:
When I configure as you outlined here and run the test, I keep getting the below error. Any help with how I should troubleshoot it would be helpful. Thank you.

```
org.mockito.exceptions.misusing.NotAMockException: Argument should be a mock, but is:
class com.sun.proxy.$Proxy36
at
com.travenion.controllers.customer.CustomerControllerTest.setup(CustomerControllerTest.java
:50)
```

Update: I removed the irrelevant part of the stacktrace – Petri

REPLY

Bharat Link

April 17, 2016, 03:16

To add to the above, I was able to get it to work with StandaloneSetup config, but when using webApplicationContext based configuration, I keep running into the above issue. And using WebApplicationContext, I was not even able to inject mocks manually. If I tried injecting it manually, the service is hitting the real object instead of mock object…hence it is going all the way to the database instead of using my test data.

REPLY

Petri Link

April 17, 2016, 17:41

The problem is that a real object is injected into your controller instead of a mock object. If you want to use the web application context based setup, you should read this blog post. However, I recommend that you use the standalone setup in your unit tests because it's a lot easier to maintain if you have a lot services that need to be mocked.

REPLY

Bharat Link

April 18, 2016, 00:41

Thanks Petri! Yes, I did follow that blog already and that's where I picked up the standalone setup from. I will continue using the standalone setup. I appreciate your feedback.

REPLY

Petri Link

April 18, 2016, 18:29

You are welcome. It is always fun to help other people.

REPLY

Rahul Singh Link

May 26, 2016, 20:23

Hi Petri I have followed your youtube tutorial and I have a very basic problem as far as I know I am doing everything write but still getting assertion error Content type not set, I have asked this question on stack overflow and other sites no help yet and It's kinda blocker for me right now

So here is my problem

this is the setup method

```
@Before
public void init() {
MockitoAnnotations.initMocks(this);
ReflectionTestUtils.setField(restController, “luceneSearchEnabled”, true);
mockMvc = standaloneSetup(restController).build();
}
```

This is my test method:

```
@Test
public void pmmSearchContentTypeTest() throws Exception {
mockMvc
.perform(get(“/api/v1/pmm”).contentType(MediaType.APPLICATION_JSON))
.andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON_VALUE)
.andReturn();
}
```

This is my search method where I am setting content type:

```
@RequestMapping(value = “/api/v1/pmm”, method = RequestMethod.GET, produces =
{MediaType.APPLICATION_JSON_VALUE})
@ResponseBody
public String pmmSearch() { … }
```

REPLY

Rahul Singh Link

May 26, 2016, 20:24

Also I have checked this manually from browser and the content-type is getting set correctly

REPLY

Petri Link

May 27, 2016, 09:10

Hi,

Typically when you get that error it means that the controller method threw an exception. Have you tried to print the sent request and the received response?

REPLY

Rahul Singh Link

June 22, 2016, 20:42

I was able to figure that out I had to make a real oblect in standalone setup and I was using incomplete url

Thanks for your help though

REPLY

Petri Link

June 23, 2016, 23:18

Hi,

You are welcome! It is good to hear that you were able to solve your problem.

Sameedha Link

June 6, 2016, 23:08

Hi Petri,

Thanks for the awesome tutorial. It really helped !!
But, I am facing this weird issue while running JUnit test .

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'com.borrowlenses.services.junits.OrderServiceControllerTest': Injection of autowired dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.borrowlenses.controller.OrdersRestController com.borrowlenses.services.junits.OrderServiceControllerTest.mockOrdersRestController; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.borrowlenses.controller.OrdersRestController] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}
at
org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessPropertyValues(AutowiredAnnotationBeanPostProcessor.java:292)

I have added the correct context configuration location. Can you help me to identify the cause of this problem ?

Thanks in Advance :)

REPLY

Petri Link

June 7, 2016, 19:59

Hi,

It seems that the Spring container cannot find the `OrdersRestController` bean. Unfortunately it's impossible to say what causes this because I cannot run your code. However, take a look at this blog post. It identifies the situations when the `NoSuchBeanDefinitionException` is thrown and explains how you can solve them.

REPLY

Arun Link

October 21, 2016, 01:05

Hi Petri,

If our controller methods have Method Security(@PreAuthorize) and checks for roles. How do we write Unit tests for the controller methods ? Should we create mock for spring security ? if so, how do we do ?

REPLY

c4rb0n Link

March 24, 2017, 18:40

Hello Petri,

Thanks for your tutorials, it really helps to explore Spring Test.

I have a question regarding exploring the operation results.
Is that anyhow possible to iterate through the returned ToDo lists instead of:

```
mockMvc.perform(get("/api/todo"))
.andExpect(status().isOk())
.andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
.andExpect(jsonPath("$", hasSize(2)))
```

```
.andExpect(jsonPath("$[0].id", is(1)))
.andExpect(jsonPath("$[0].description", is("Lorem ipsum")))
.andExpect(jsonPath("$[0].title", is("Foo")))
.andExpect(jsonPath("$[1].id", is(2)))
.andExpect(jsonPath("$[1].description", is("Lorem ipsum")))
.andExpect(jsonPath("$[1].title", is("Bar")));
```

???

REPLY

Petri Link

March 27, 2017, 21:18

Hi,

You can invoke the `andReturn()` method of the `ResultActions` interface. This method returns a `MvcResult` object that allows you to access the result of an invoked request.

On the other hand, if you want to write a test that returns a list of objects, you can simply invoke the controller method without using the Spring MVC Test framework.

REPLY

---

channing Link

May 7, 2017, 11:30

Hi Petri,
I am a new starter for junit test of Rest api. I have two problems when I tried to implement your code.
The first one:
When I implement the junit test for add(@Valid@RequestBody TodoDTO todoDto),it throws "java.lang.AssertError: Content type not set", and I don't know how to solve it. I have read your

previous blogs, and every unit test is ok, and only this one has problem. The detailed information is below:

The segment of ToDoController.java:

```
@RequestMapping(value="/api/todo",method=RequestMethod.POST)
@ResponseBody
public TodoDTO add(@Valid@RequestBody TodoDTO todoDto){
    Todo added = service.add(todoDto);
    return createDTO(added);
}

private TodoDTO createDTO(Todo model) {
    TodoDTO dto = new TodoDTO();
    dto.setId(model.getId());
    dto.setDescription(model.getDescription());
    dto.setTitle(model.getTitle());
    return dto;
}
```

The segment of TodoControllerTest.java:

```
@Test
public void testAddJson_Fail() throws Exception{
    String title = TestUtil.createStringWithLength(101);
    String description = TestUtil.createStringWithLength(501);
    TodoDTO first = new TodoDTOBuilder()
        .description(description)
        .title(title)
```

```
            .build();

        mockMvc.perform(post("/api/todo")
            .contentType(TestUtil.APPLICATION_JSON_UTF8)
            .content(TestUtil.convertObjectToJsonBytes(first)))
            .andExpect(status().isBadRequest())
            .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
            .andExpect(jsonPath("$.fieldErrors", hasSize(2)))
            .andExpect(jsonPath("$.fieldErrors[*].path", containsInAnyOrder("title", "d
            .andExpect(jsonPath("$.fieldErrors[*].message", containsInAnyOrder(
                   "The maximum length of the description is 500 characters.",
                   "The maximum length of the title is 100 characters."
        )));

        verifyZeroInteractions(todoServiceMock);
    }
```

The second one:
because I have encounter above problem, so I comment the ” .andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))”, but it throws another problem, and it is “java.lang.IllegalArgumentException:json can not be null or empty”, I followed you every step, and I don’t why it appears the problem.The detailed information is below:
The segment of ToDoControllerTest:

```
    @Test
    public void testAddJson_Fail() throws Exception{
        String title = TestUtil.createStringWithLength(101);
        String description = TestUtil.createStringWithLength(501);
        TodoDTO first = new TodoDTOBuilder()
            .description(description)
            .title(title)
            .build();

    mockMvc.perform(post("/api/todo")
            .contentType(TestUtil.APPLICATION_JSON_UTF8)
            .content(TestUtil.convertObjectToJsonBytes(first)))
            .andExpect(status().isBadRequest())
            //.andExpect(content().contentType(TestUtil.APPLICATION_JSON_UTF8))
            .andExpect(jsonPath("$.fieldErrors", hasSize(2)))
            .andExpect(jsonPath("$.fieldErrors[*].path", containsInAnyOrder("title", "d
            .andExpect(jsonPath("$.fieldErrors[*].message", containsInAnyOrder(
                    "The maximum length of the description is 500 characters.",
                    "The maximum length of the title is 100 characters."
    )));

    verifyZeroInteractions(todoServiceMock);
}
```

I have added the FieldErrorDTO.java, ValidationErrorDTO.java, RestErrorHandler.java to my project according to your “Spring From the Trenches: Adding Validation to a REST API” blog. I don’t know whether I have used it correctly.
I don’t know if I have explained my problem clearly, if it doesn’t make any sense, could you send me an email, and I can send you my project. My address is “igdnss@126.com”.
Thank you in advance.

REPLY

Petri Link

May 8, 2017, 22:07

Hi,

If you get the error: *java.lang.AssertError: Content type not set*, the problem (most likely) is that the tested code threw an exception. What log level do you use for the `org.springframework` loggers? I ask this because if you set this log level to `DEBUG`, you should be able to find the exception from log file after you have run the failed test.

REPLY

Dan Link

July 6, 2017, 04:11

I am confused about what it means when you are doing
when(todoServiceMock.add(any(TodoDTO.class))).thenReturn(added);
Is this is saying "when this method is called in my Service class, rather than call the actual add method, return the test model obj you created earlier? If so, then I am confused how this tests the Service or the Controller.
Does the MockMVC run an instance of your server behind the scenes, processing the HTTP requests? And if so, does it automatically bind your MockService with a service instance in that particular controller?

REPLY

Petri Link

July 6, 2017, 10:49

Hi,

You are right. That particular line configures the object that is returned when the `add()` method is invoked and a `TodoDTO` object is given as a method parameter.

> If so, then I am confused how this tests the Service or the Controller.

This test doesn't test the service class. The service object is simply replaced with a test double that is created by using Mockito.

This is a quite old tutorial and the examples use web application context based setup. In other words, our test double is a Spring bean that is created in the application context configuration class which configures the application context of our unit tests. Nowadays I use the standalone setup when I write unit tests for Spring MVC controllers mainly because the setup code of a single test case is easier read (IMO).

> Does the MockMVC run an instance of your server behind the scenes, processing the HTTP requests?

The Spring MVC Test framework doesn't use a real server. It's build on the top of the Servlet API mock objects which are provided by the spring-test module, and it uses the `DispatcherServlet` class that provides full support for Spring MVC runtime behavior.

> And if so, does it automatically bind your MockService with a service instance in that particular controller?

No. The mock service is a Spring bean that is injected to the tested controller by the Spring container. That being said, if you use the standalone configuration, you have to create tested controller object yourself (and inject all required dependencies manually).

If you have any additional questions, don't hesitate to ask them!

REPLY

Sanjib Link

August 22, 2017, 11:25

Hi petri,
I have been following along your tutorials and i am stuck while returning from the controllers.
It gives

MockHttpServletRequest:
HTTP Method = GET

```
Request URI = /users
Parameters = {}
Headers = {Accept=[application/json]}

Handler:
Type = com.firsthelpfinancial.restapp.controller.UserController
Method = public java.util.List com.firsthelpfinancial.restapp.controller.UserController.findAll()

Async:
Was async started = false
Async result = null

Resolved Exception:
Type = org.springframework.web.util.NestedServletException

ModelAndView:
View name = error/error
View = null
Attribute = exception
value = org.springframework.web.util.NestedServletException: Handler dispatch failed; nested
exception is java.lang.AbstractMethodError:
org.springframework.mock.web.MockHttpServletResponse.getHeaders(Ljava/lang/String;)Ljav
a/util/Collection;

FlashMap:

MockHttpServletResponse:
Status = 500
Error message = null
Headers = {}
Content type = null
Body =
Forwarded URL = /WEB-INF/jsp/error/error.jsp
Redirected URL = null
Cookies = []
```

The stack trace prints:

```
13:45:52.694 [main] DEBUG
org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver –
Resolving exception from handler [public java.util.List
com.firsthelpfinancial.restapp.controller.UserController.findAll()]:
org.springframework.web.util.NestedServletException: Handler dispatch failed; nested
exception is java.lang.AbstractMethodError:
org.springframework.mock.web.MockHttpServletResponse.getHeaders(Ljava/lang/String;)Ljav
a/util/Collection;
13:45:52.694 [main] DEBUG
org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver – Resolving
exception from handler [public java.util.List
com.firsthelpfinancial.restapp.controller.UserController.findAll()]:
org.springframework.web.util.NestedServletException: Handler dispatch failed; nested
exception is java.lang.AbstractMethodError:
org.springframework.mock.web.MockHttpServletResponse.getHeaders(Ljava/lang/String;)Ljav
a/util/Collection;
13:45:52.695 [main] DEBUG
org.springframework.web.servlet.handler.SimpleMappingExceptionResolver – Resolving
exception from handler [public java.util.List
com.firsthelpfinancial.restapp.controller.UserController.findAll()]:
org.springframework.web.util.NestedServletException: Handler dispatch failed; nested
exception is java.lang.AbstractMethodError:
org.springframework.mock.web.MockHttpServletResponse.getHeaders(Ljava/lang/String;)Ljav
a/util/Collection;
```

any help is greatly appreciated.

REPLY

Petri Link

August 22, 2017, 18:41

Hi,

This look pretty interesting. In other words, I have never seen this exception before. That being said, it might be caused by incompatible Spring Test and Spring Framework versions. I would check that you use the same Spring Test and Spring Framework version.

REPLY

Sanjib Link

August 23, 2017, 10:43

Yeah, that was it. It was due to the version mismatch as you have pointed out. Thank you very much.

REPLY

Petri Link

August 23, 2017, 20:33

You are welcome!

REPLY

sreenu Link

September 22, 2017, 11:51

hi preti,
i need your help . i wan know junit testing code in rest controller .how to test rest controller methods.

REPLY

Petri Link

September 22, 2017, 18:50

Hi,

This blog post describes how you can write unit tests for a REST API by using JUnit and Spring MVC Test framework. That's why I am a bit confused. Is this post hard to understand? If so, could you identify the sections which are unclear to you?

REPLY

Yura Link

October 1, 2017, 22:35

Hi, Petri
Thank you for the post and information.
Have an issue.
Trying to run unit tests with your reccomendations, but getting such error.

```
java.lang.AssertionError: JSON path “$”
Expected: a collection with size
but: collection size was
at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
at
org.springframework.test.util.JsonPathExpectationsHelper.assertValue(JsonPathExpectationsH
elper.java:74)
at
org.springframework.test.web.servlet.result.JsonPathResultMatchers$1.match(JsonPathResult
Matchers.java:86)
at org.springframework.test.web.servlet.MockMvc$1.andExpect(MockMvc.java:171)
at
com.softserve.edu.Resources.controller.LookUpControllerTest.testLoadResourceTypes(LookU
pControllerTest.java:56)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:44)
at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:271)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:70)
at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:50)
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:238)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
at org.eclipse.jdt.internal.junit4.runner.JUnit4TestReference.run(JUnit4TestReference.java:86)
at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:459)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:678)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:382)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:192)
```

This is my TestClass

```
public class LookUpControllerTest {

    @InjectMocks
    LookUpController lookUpController;

    @Mock
    ResourceTypeService resourceTypeService;

    MockMvc mockMvc;
    @Before
    public void setUp() throws Exception {
        MockitoAnnotations.initMocks(this);
        mockMvc = MockMvcBuilders.standaloneSetup(lookUpController).build();
    }

    @Test
    public void testLoadResourceTypes() throws Exception{
        List  resourceTypes = new ArrayList();
        ResourceType resType1 = new ResourceType();
        resType1.setId(1L);
        resType1.setTypeName("Cars");
        resType1.setTableName("cars");
        ResourceType resType2 = new ResourceType();
        resType1.setId(2L);
        resType1.setTypeName("Building");
        resType1.setTableName("building");

        when(resourceTypeService.getInstances()).thenReturn(resourceTypes);
        mockMvc.perform(get("/lookUp/resourceTypes"))
                .andExpect(status().isOk())
                .andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8_VA
                .andExpect(jsonPath("$", hasSize(2)))
                .andExpect(jsonPath("$[0].id", is(1L)))
                .andExpect(jsonPath("$[0].typeName", is("Cars")))
                .andExpect(jsonPath("$[0].tableName", is("cars")))
                .andExpect(jsonPath("$[1].id", is(2L)))
                .andExpect(jsonPath("$[1].typeName", is("Building")))
                .andExpect(jsonPath("$[1].tableName", is("building")));

        verify(resourceTypeService, times(1)).getInstances();
        verifyNoMoreInteractions(resourceTypeService);
    }
}
```

What could be wrong?

REPLY

Yura Link

October 1, 2017, 22:49

it was simple mistake). just forgot to add object to List.

But still after test running, gets an error:

ava.lang.AssertionError: JSON path “$[0].id”

Expected: is

but: was

why it changed the objects order of list?

REPLY

Petri Link

October 2, 2017, 00:42

Hi,

If you are writing assertions for numbers by using JsonPath and Hamcrest, you should use integers instead of long values. Of course, I cannot be 100% sure that this is the root cause of your problem because your comment doesn’t reveal the expected and actual values of the `id` property.

Thus, if this doesn’t help you, let me know.

REPLY

Yura Link

October 3, 2017, 10:24

i changed to integers. but still.

Java.lang.AssertionError: JSON path “$[0].id”

Expected: is 1

but: was 2

why it changed the objects order of list?

REPLY

Petri Link

October 7, 2017, 10:05

It’s kind of hard to say for sure because I cannot debug the code, but I can say that I haven’t seen similar behavior in my own tests. In other words, I suspect that the tested code changes the order of the returned objects.

I would take run the test by using a debugger and find out when the order of the objects changes. After you have figured out the reason for this behavior, it should be easy to fix it.

By the way, I just noticed that your test code doesn’t add the `ResourceType` objects to the returned list. Are you sure that you are adding them to the list by using the order that is expected by your test case?

Yura Link

October 3, 2017, 10:33

Have another issue:

Here is my another controller method with DTO class static method usage:

```
RequestMapping(value = "/lookup/resourceTypes", method = RequestMethod.GET)
public List findResourcesTypes() {
    return DtoUtilMapper.resTypesToResTypesDTO(resourceTypeService.getInstances());
}
```

DTO Class:

```
    public class DtoUtilMapper {

        public static List resTypesToResTypesDTO (List resTypes){
            List  resTypeDTOs = new ArrayList();
            for (ResourceType resType : resTypes) {
                resTypeDTOs.add(new ResourceTypeDTO(resType.getId(),
                        resType.getTableName(),
                        resType.getTypeName()
                ));
            }


            return resTypeDTOs;
        }
    }
```

And when i test this method with the same test as previous, i`ve got:
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.NullPointerException
at
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:9
82)

There were more erorrs in a trace: could send it if needed.
and one more caused:

```
Caused by: java.lang.NullPointerException
at com.softserve.edu.Resources.dto.ResourceTypeDTO.(ResourceTypeDTO.java:27)
at
com.softserve.edu.Resources.dto.DtoUtilMapper.resTypesToResTypesDTO(DtoUtilMapper.java
:15)
33 more errors
```

what could be the reason?

REPLY

Petri Link

October 7, 2017, 10:21

You are trying to use an object reference that has a `null` value. Again, the easiest way to figure why this is happens is to run your test by using a debugger. Put a breakpoint to the `DtoUtilMapper` class (just before the row that rows the exception) and take a look at the variable values. This should reveal why your test throws the `NullPointerException`.

REPLY

Mohd Furkan Link

December 4, 2017, 09:26

I learnt a lot from this tutorial thanks.

REPLY

Petri Link

December 6, 2017, 16:38

You are welcome!

REPLY

Nic Link

January 6, 2019, 21:43

Thank a lot Petri for clean and detailed example

REPLY

Petri Link

January 10, 2019, 20:34

You are welcome. I am happy to hear that this blog post was useful to you.

REPLY

Leave a Comment

Name Name (required)

Comment

Save my name, email, and website in this browser for the next time I comment.

SUBMIT

PREVIOUS POST: WHAT KIND OF A SPRING TESTING BOOK WOULD YOU LOVE TO READ?

NEXT POST: SCREENCAST: UNIT TESTING OF SPRING MVC CONTROLLERS – REST API

NEVER MISS A BLOG POST

Subscribe my email newsletter AND you will get an email when I publish a new blog post

Your Email

Subscribe Now

I will never sell, rent, or share your email address.

WRITE BETTER TESTS

Test With Spring Course

Java Testing Weekly

JUnit 5 Tutorial

Spring MVC Test Tutorial

TestProject Tutorial

WireMock Tutorial

Writing Clean Tests

Writing Tests for Data Access Code

MASTER SPRING FRAMEWORK

Spring Data JPA Tutorial

Spring Data Solr Tutorial

Spring From the Trenches

Spring MVC Test Tutorial

Spring Social Tutorial

Using jOOQ with Spring

BUILD YOUR APPLICATION

Getting Started With Gradle

Maven Tutorial

FIND THE BEST TUTORIALS

JUnit 5 - The Ultimate Resource

Spring Batch - The Ultimate Resource

SEARCH

enter search term and press enter

FROM THE BLOG

Recent Popular Favorites

Java Testing Weekly 6 / 2019

Adding Input and Output Parameters to TestProject Actions

Java Testing Weekly 5 / 2019

Java Testing Weekly 4 / 2019

Writing Addons With TestProject

HOW TO WRITE BETTER TESTS?

If you are struggling to write automated tests that **embrace change**, you should find out how my testing course can help you to **write tests for Spring and Spring Boot applications**.