**Giới thiệu**

Viết Test là một phần quan trọng trong việc xây dựng tất cả ứng dụng chứ không riêng gì **Spring Boot**, nên tất nhiên chúng ta không thể bỏ phần này ra khỏi series được rồi.

Hôm nay chúng ta sẽ tìm hiểu cách để viết TestCase trong **Spring Boot**.

Yêu cầu tối thiếu để đi tiếp phần này đó là bạn phải có nền tảng với JUnit và Mockito.

Tôi nghĩ JUnit thì chắc ai cũng biết rồi, nhưng Mockito có lẽ sẽ có bạn lạ lẫm, nếu bạn chưa nghe tới nó bao giờ thì cũng đừng lo, có thể đọc viết [hướng dẫn Mockito chi tiết](https://loda.me/test-huong-dan-toan-tap-mockito-loda1576641016810) bởi luda :v

**Vấn đề Test + Spring**

Chúng ta đều biết rằng **Spring Boot** sẽ phải tạo Context và tìm kiếm các Bean và nhét vào đó. Sau tất cả các bước config và khởi tạo thì chúng ta sử dụng @Autowired để lấy đối tượng ra sử dụng.

Vấn đề đầu tiên bạn nghĩ tới khi viết Test sẽ là làm sao @Autowired bean vào class Test được và làm sao cho JUnit hiểu @Autowired là gì.

Hướng giải quyết là tích hợp Spring vào với JUnit.

**@RunWith(SpringRunner.class)**

**Spring Boot** đã thiết kế ra lớp SpringRunner, sẽ giúp chúng ta tích hợp **Spring + JUnit**.

Để test trong Spring, trong mọi class Test chúng ta sẽ thêm @RunWith(SpringRunner.class) lên đầu Class Test đó.

@RunWith(SpringRunner.class)

public class TodoServiceTest {

...

}

Khi bạn chạy TodoServiceTest nó sẽ tạo ra một Context riêng để chứa bean trong đó, vậy là chúng ta có thể @Autowired thoải mái trong nội hàm Class này.

Vấn đề tiếp theo là làm sao đưa Bean vào trong Context.

Có 2 cách

1. @SpringBootTest
2. @TestConfiguration

**@SpringBootTest**

@SpringBootTest sẽ đi tìm kiếm class có gắn @SpringBootApplication và từ đó đi tìm toàn bộ Bean và nạp vào Context.

Cái này chỉ nên sử dụng trong trường hợp muốn Integration Tests, vì nó sẽ tạo toàn bộ Bean, không khác gì bạn chạy cả cái SpringApplication.run(App.class, args);, rất tốn thời gian mà rất nhiều Bean thừa thãi, không cần dùng đến cũng khởi tạo.

@RunWith(SpringRunner.class)

@SpringBootTest

public class TodoServiceTest {

/\*\*

\* Cách này sử dụng @SpringBootTest

\* Nó sẽ load toàn bộ Bean trong app của bạn lên,

\* Giống với việc chạy App.java vậy

\*/

@Autowired

private TodoService todoService;

}

**@TestConfiguration**

@TestConfiguration giống với @Configuration, chúng ta tự định nghĩa ra Bean.

Các Bean được tạo bởi @TestConfiguration chỉ tồn tại trong môi trường Test. Rất phụ hợp với việc viết UnitTest.

Class Test nào, cần Bean gì, thì tự tạo ra trong @TestConfiguration

@RunWith(SpringRunner.class)

public class TodoServiceTest2 {

/\*\*

\* Cách này sử dụng @TestConfiguration

\* Nó chỉ tạo ra Bean trong Context test này mà thôi

\* Tiết kiệm thời gian hơn khi sử dụng @SpringBootTest (vì phải load hết Bean lên mà không dùng đến)

\*/

@TestConfiguration

public static class TodoServiceTest2Configuration{

/\*

Tạo ra trong Context một Bean TodoService

\*/

@Bean

TodoService todoService(){

return new TodoService();

}

}

@Autowired

private TodoService todoService;

**@MockBean**

**Spring** hỗ trợ mock với annotation @MockBean, chúng ta có thể mock lấy ra một Bean "giả" mà không thèm để ý tới thằng Bean "thật". (Kể cả thằng "thật" có ở trong Context đi nữa, cũng không quan tâm).

@RunWith(SpringRunner.class)

public class TodoServiceTest2 {

/\*\*

\* Đối tượng TodoRepository sẽ được mock, chứ không phải bean trong context

\*/

@MockBean

TodoRepository todoRepository;

**Demo 1**

Chúng ta sẽ học cách sử dụng các Annotation ở trên.

**Cài đặt**

*pom.xml*

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<packaging>pom</packaging>

<parent>

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

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

<version>2.0.5.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>me.loda.spring</groupId>

<artifactId>spring-boot-learning</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-boot-learning</name>

<description>Everything about Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--spring mvc, rest-->

<dependency>

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

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

</dependency>

<!--for test-->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Cấu trúc thư mục:



**Tạo Model, Service, Repository**

Chúng ta sử dụng [Lombok](https://loda.me/general-huong-dan-su-dung-lombok-giup-code-java-nhanh-hon-69-loda1552789752787) cho tiện nhé.

*Todo.java*

@Data

@AllArgsConstructor

public class Todo {

private int id;

private String title;

private String detail;

}

Vì phục vụ mục đích demo, chúng ta sẽ không cần sử dụng tới **Spring JPA** mà sẽ tự viết.

*TodoRepository.java*

public interface TodoRepository {

List<Todo> findAll();

Todo findById(int id);

}

*TodoService.java*

@Service

public class TodoService {

@Autowired

private TodoRepository todoRepository;

public int countTodo(){

return todoRepository.findAll().size();

}

public Todo getTodo(int id){

return todoRepository.findById(id);

}

public List<Todo> getAll(){

return todoRepository.findAll();

}

}

*App.java*

@SpringBootApplication

public class App {

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

}

Rất đơn giản.

**Test bằng @SpringBootTest**

Chúng ta mock TodoRepository và giả lập cho nó trả ra một List<Todo> gồm 10 phần tử.

@RunWith(SpringRunner.class)

@SpringBootTest

public class TodoServiceTest {

/\*\*

\* Cách này sử dụng @SpringBootTest

\* Nó sẽ load toàn bộ Bean trong app của bạn lên,

\* Giống với việc chạy App.java vậy

\*/

/\*\*

\* Đối tượng TodoRepository sẽ được mock, chứ không phải bean trong context

\*/

@MockBean

TodoRepository todoRepository;

@Autowired

private TodoService todoService;

@Before

public void setUp() {

Mockito.when(todoRepository.findAll())

.thenReturn(IntStream.range(0, 10)

.mapToObj(i -> new Todo(i, "title-" + i, "detail-" + i))

.collect(Collectors.toList()));

}

@Test

public void testCount() {

Assert.assertEquals(10, todoService.countTodo());

}

}

Bạn sẽ thấy Test chạy thành công, nhưng sẽ mất thời gian vì khởi tạo Context quá lâu do @SpringBootTest phải load hết tất cả bean lên.

**Test bằng @TestConfiguration**

package me.loda.spring.testinginspringboot;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* For Vietnamese readers:

\* Các bạn thân mến, mình rất vui nếu project này giúp

\* ích được cho các bạn trong việc học tập và công việc. Nếu

\* bạn sử dụng lại toàn bộ hoặc một phần source code xin để

\* lại dường dẫn tới github hoặc tên tác giá.

\* Xin cảm ơn!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.util.stream.Collectors;

import java.util.stream.IntStream;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.Mockito;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.TestConfiguration;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.context.annotation.Bean;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

public class TodoServiceTest2 {

/\*\*

\* Cách này sử dụng @TestConfiguration

\* Nó chỉ tạo ra Bean trong Context test này mà thôi

\* Tiết kiệm thời gian hơn khi sử dụng @SpringBootTest (vì phải load hết Bean lên mà không dùng đến)

\*/

@TestConfiguration

public static class TodoServiceTest2Configuration{

/\*

Tạo ra trong Context một Bean TodoService

\*/

@Bean

TodoService todoService(){

return new TodoService();

}

}

@MockBean

TodoRepository todoRepository;

@Autowired

private TodoService todoService;

@Before

public void setUp() {

Mockito.when(todoRepository.findAll())

.thenReturn(IntStream.range(0, 10)

.mapToObj(i -> new Todo(i, "title-" + i, "detail-" + i))

.collect(Collectors.toList()));

}

@Test

public void testCount() {

Assert.assertEquals(10, todoService.countTodo());

}

}

**Vấn đề Test + Spring Boot 2**

Ở trên chúng ta đã xử lý xong vấn đề dependency injection của **JUnit** + **Spring Boot** rồi.

Vấn đề tiếp theo là cái **Controller**. Đúng vậy, chúng ta tạo ra hàng chục Rest API đón request người dùng, vậy làm sao để test nó.

Nếu **Controller** không được test thì là một lỗ hổng cực lớn, vì nó là đầu ra chính của chương trình, nó sai -> chương trình không có giá trị.

Tuy nhiên, chúng ta không muốn khởi động cả Tomcat Server + Database để test 1 API. Nó rất tốn tài nguyên và thời gian.

**@WebMvcTest**

**Spring Boot** hỗ trợ test **Controller** mà không cần khởi động Tomcat Server bằng annotation @WebMvcTest.

Tất nhiên là nếu không khởi động Server, thì phải có một phương thức khác giả lập, vâng, lại là **Mock**.

Bây giờ, khi muốn test một Controller nào đó, chúng ta làm như sau:

@RunWith(SpringRunner.class)

// Bạn cần cung cấp lớp Controller cho @WebMvcTest

@WebMvcTest(TodoRestController.class)

public class TodoRestControllerTest {

/\*\*

\* Đối tượng MockMvc do Spring cung cấp

\* Có tác dụng giả lập request, thay thế việc khởi động Server

\*/

@Autowired

private MockMvc mvc;

}

**Demo 2**

Chúng ta vẫn sử dụng tiếp Project đã tạo ở **# Demo 1**

**Tạo Controller**

*TodoRestController.java*

@RestController

@RequestMapping("/api/v1")

public class TodoRestController {

@Autowired

TodoService todoService;

@GetMapping("/todo")

public List<Todo> findAll(){

return todoService.getAll();

}

}

**Tạo Test Controller**

import static org.hamcrest.CoreMatchers.is;

import static org.hamcrest.Matchers.hasSize;

import static org.mockito.BDDMockito.given;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.boot.test.mock.mockito.MockBean;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

@RunWith(SpringRunner.class)

// Bạn cần cung cấp lớp Controller cho @WebMvcTest

@WebMvcTest(TodoRestController.class)

public class TodoRestControllerTest {

/\*\*

\* Đối tượng MockMvc do Spring cung cấp

\* Có tác dụng giả lập request, thay thế việc khởi động Server

\*/

@Autowired

private MockMvc mvc;

@MockBean

private TodoService todoService;

@Test

public void testFindAll() throws Exception {

// Tạo ra một List<Todo> 10 phần tử

List<Todo> allTodos = IntStream.range(0, 10)

.mapToObj(i -> new Todo(i, "title-" + i, "detail-" + i))

.collect(Collectors.toList());

// giả lập todoService trả về List mong muốn

given(todoService.getAll()).willReturn(allTodos);

mvc.perform(get("/api/v1/todo").contentType(MediaType.APPLICATION\_JSON)) // Thực hiện GET REQUEST

.andExpect(status().isOk()) // Mong muốn Server trả về status 200

.andExpect(jsonPath("$", hasSize(10))) // Hi vọng server trả về List độ dài 10

.andExpect(jsonPath("$[0].id", is(0))) // Hi vọng phần tử trả về đầu tiên có id = 0

.andExpect(jsonPath("$[0].title", is("title-0"))) // Hi vọng phần tử trả về đầu tiên có title = "title-0"

.andExpect(jsonPath("$[0].detail", is("detail-0")));

}

}

Chạy thử và trải nghiệm bạn nhé :3

**Kết**

Test là một phần quan trọng trong hệ thống, hi vọng các đọc kĩ và thực hành để nắm chắc.