Spring 5, JUnit 5를 이용한 테스트

- 강경미 (carami@nate.com)

Spring Test란

Spring Test는 스프링 프레임워크에서 제공하는 통합 테스트 및 스프링 MVC 통합 테스트를 지원합니다. Java 코드에 대해서 스프링 컨테이너 생성, 트랜잭션, DB와 연동 관리를 지원하며 웹 관련 Mock테스트를 위한 객체를 지원합니다.

실행 환경

- Junit 5, Spring Test , Mockito

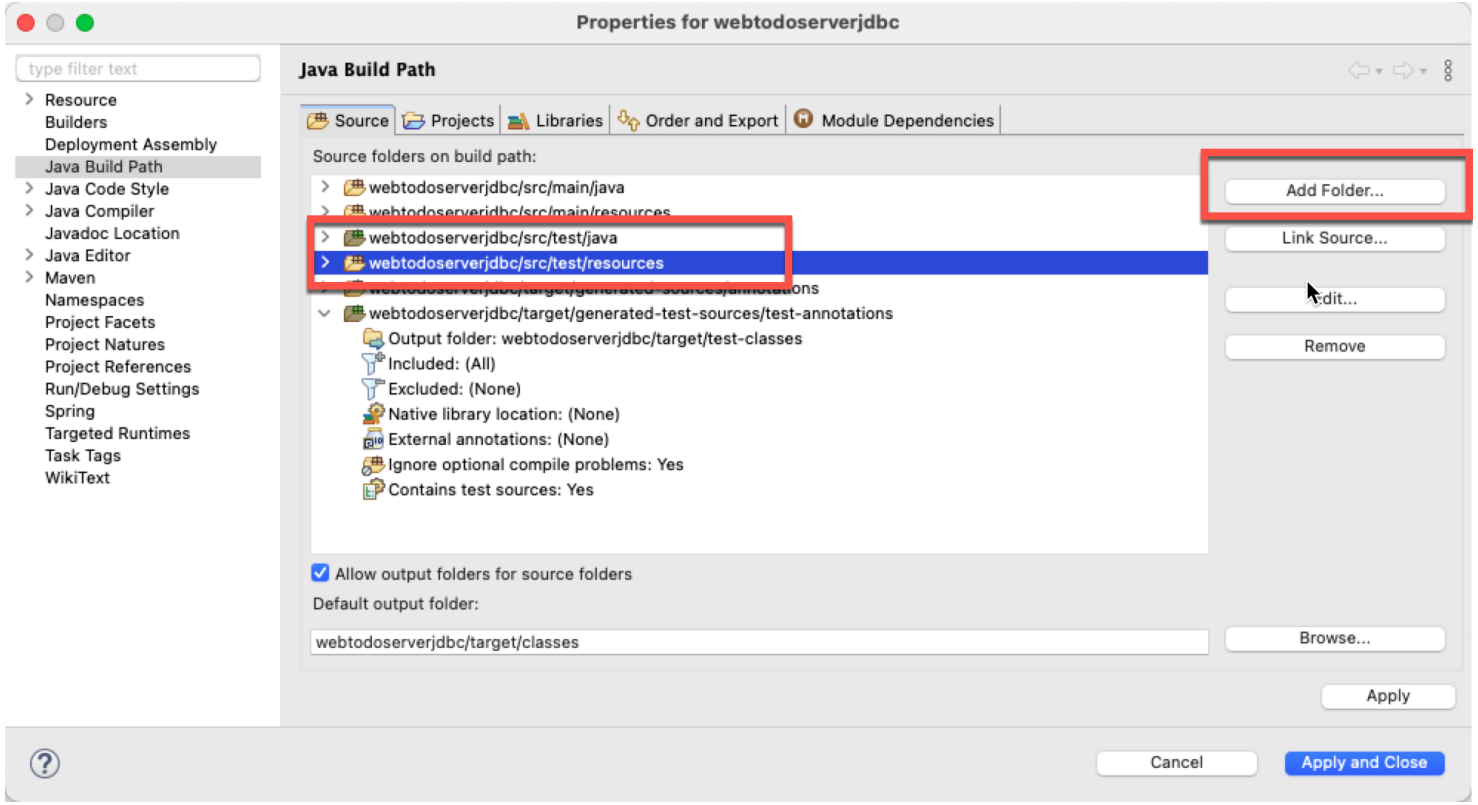
```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>5.3.23</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.8.1</version>
        <scope>test</scope>
    </dependency>
```

```
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>4.8.1</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest</artifactId>
        <version>2.2</version>
        <scope>test</scope>
    </dependency>
```

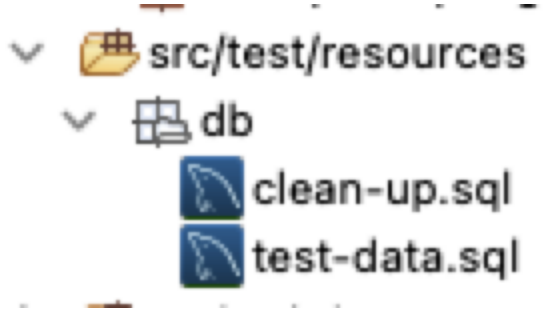
로그를 위한 라이브러리 추가

```
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
    </dependency>
```

- 다음과 같은 폴더를 작성하고 테스트 코드와 리소스 파일을 작성한다.
  - src/test/java
  - src/test/resources



Test시 테스트용 테이블, 데이터를 관리하기 위해 준비한다.



test-data.sql

```
drop table todo if exists;

create TABLE todo (
    id bigint generated by default as identity,
    todo VARCHAR(255),
    done boolean,
    primary key(id)
);


insert into todo (id, todo, done) values(1, 'hello', true);
insert into todo (id, todo, done) values(2, 'hi', true);
insert into todo (id, todo, done) values(3, 'smile', true);
```

clean-up.sql

```
drop table todo;
```

로그 설정 파일 작성

- test/resources 폴더에 logback.xml 파일로 작성

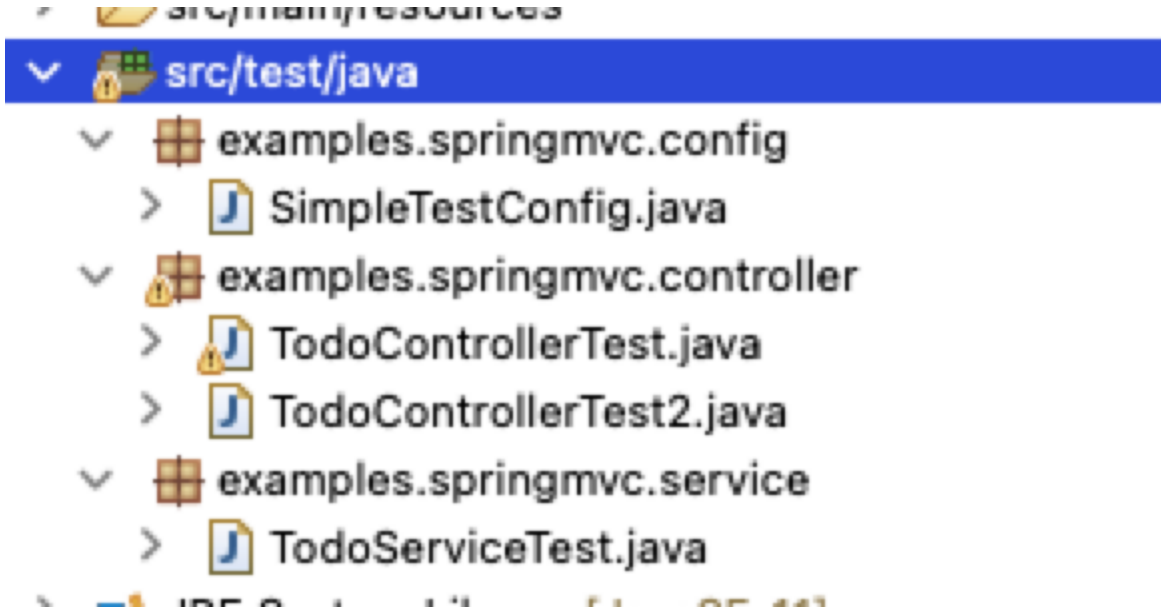
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <contextListener class="ch.qos.logback.classic.jul.LevelChangePropagator">
        <resetJUL>true</resetJUL>
    </contextListener>

    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{5} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="examples.springmvc" level="info"/>

    <logger name="org.springframework" level="off"/>

    <root level="info">
        <appender-ref ref="console" />
    </root>
</configuration>
```

로그 설정 파일 작성

- 로그는 debug, info, warn, error 순으로 심각하다. (로그레벨)
- 보통 아키텍트가 아키텍처를 정의할 때 어떤 경우에 어떤 로그레벨을 사용할지 결정한다.
- 위의 설정은 examples.springmvc로 시작하는 패키지의 info 이상의 로그를 남긴다는 의미다.
- info이상의 로그는 모두 console로 기록한다.
- console은 위의 appender로 정의 되었다. ConsoleAppender는 모니터에 출력을 하는 객체로 정해진 포맷형태로 출력을 한다.

Test를 위한 DataSource를 Java Config에서 작성한다.

- SimpleTestConfig.java



Test를 위한 DataSource를 Java Config에서 작성한다.

- SimpleTestConfig.java
  - 애플리케이션과 함께 실행되는 임베디드 H2를 실행하고 이를 이용해 DataSource를 생성한다.

```
package examples.springmvc.config;

import javax.sql.DataSource;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;
```

```
@Configuration
public class SimpleTestConfig {

	private static Logger logger = LoggerFactory.getLogger(SimpleTestConfig.class);

	@Bean(name = "dataSource")
	public DataSource dataSource() {
		try {
			EmbeddedDatabaseBuilder dbBuilder = new EmbeddedDatabaseBuilder();
			return dbBuilder.setType(EmbeddedDatabaseType.H2).build();
		} catch (Exception e) {
			logger.error("임베디드 DataSource 빈을 생성할 수 없습니다!", e);
			return null;
		}
	}

}
```

- `private static Logger logger = LoggerFactory.getLogger(SimpleTestConfig.class);`
    - 로그 객체를 선언하였다. getLogger()메소드 안에는 로그를 남길 클래스 정보를 넣어준다.
- 내장 H2데이터베이스를 실행하고, 접속할 수 있는 DataSource를 반환한다.
    - 주의할점은 H2가 실행되고 있으면 안된다. port충돌남.

```
EmbeddedDatabaseBuilder dbBuilder = new EmbeddedDatabaseBuilder();
return dbBuilder.setType(EmbeddedDatabaseType.H2).build();
```

JUnit 5 애노테이션

@Test

- 본 어노테이션을 붙이면 Test 메서드로 인식하고 테스트 한다. JUnit5 기준으로 접근제한자가 Default 여도 된다. (JUnit4 까지는 public이어야 했었다.)

@BeforeAll

- 본 어노테이션을 붙인 메서드는 해당 테스트 클래스를 초기화할 때 딱 한번 수행되는 메서드다. 메서드 시그니쳐는 static 으로 선언해야한다.

@BeforeEach

- 본 어노테이션을 붙인 메서드는 테스트 메서드 실행 이전에 수행된다.

JUnit 5 애노테이션

@AfterAll

- 본 어노테이션을 붙인 메서드는 해당 테스트 클래스 내 테스트 메서드를 모두 실행시킨 후 딱 한 번 수행되는 메서드다. 메서드 시그니쳐는 static 으로 선언해야한다.

@AfterEach

- 본 어노테이션을 붙인 메서드는 테스트 메서드 실행 이후에 수행된다.

@Disabled

- 본 어노테이션을 붙인 테스트 메서드는 무시된다.

org.springframework.test.context.jdbc.*

```
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.jdbc.SqlConfig;
import org.springframework.test.context.jdbc.SqlGroup;
```

@Sql annotation 이란?

- SQL 스크립트 혹은 쿼리를 실행시킨다. 주로 테스트 클래스, 메소드에 사용된다. 쉽게 말해 테스트 환경에서 데이터를 CRUD 할 수 있는 방법을 제공한다.

@SqlGroup 이란?

- @Sql 을 여러개 그룹화 해서 사용할 수 있다.

TodoServiceTest

```
package examples.springmvc.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;

import java.util.List;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ActiveProfiles;
import org.springframework.test.context.jdbc.Sql;
import org.springframework.test.context.jdbc.SqlConfig;
import org.springframework.test.context.jdbc.SqlGroup;
import org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

import examples.springmvc.config.ApplicationConfig;
import examples.springmvc.config.SimpleTestConfig;
import examples.springmvc.domain.Todo;
```

```
// 임베디드 DB를 사용하기 위한 Config클래스와 Service, Repository, 트랜잭션 설정을 한 ApplicationConfig.class를 읽어들니다.
@SpringJUnitConfig(classes = {SimpleTestConfig.class, ApplicationConfig.class})
@DisplayName("Integration SingerService Test")
@ActiveProfiles("test")
public class TodoServiceTest {

        private static Logger logger = LoggerFactory.getLogger(TodoServiceTest.class);

    // test할 객체를 주입받는다.
        @Autowired
        TodoService todoService;

    // JUNIT5 라이프싸이클과 관련된 애노테이션
        @BeforeAll
        static void setUp() {
                logger.info("--> @BeforeAll - 이 클래스의 모든 테스트 메소드를 실행하기 전에 실행합니다.");
        }

        @AfterAll
        static void tearDown(){
                logger.info("--> @AfterAll - 이 클래스의 모든 테스트 메소드를 실행한 다음에 실행합니다.");
        }

        @BeforeEach
        void init() {
                logger.info("--> @BeforeEach - 이 클래스의 각 테스트 메소드 실행하기 전에 실행합니다.");
        }

        @AfterEach
        void dispose() {
                logger.info("--> @AfterEach - 이 클래스의 각 테스트 메소드 실행한 다음에 실행합니다.");
        }
```

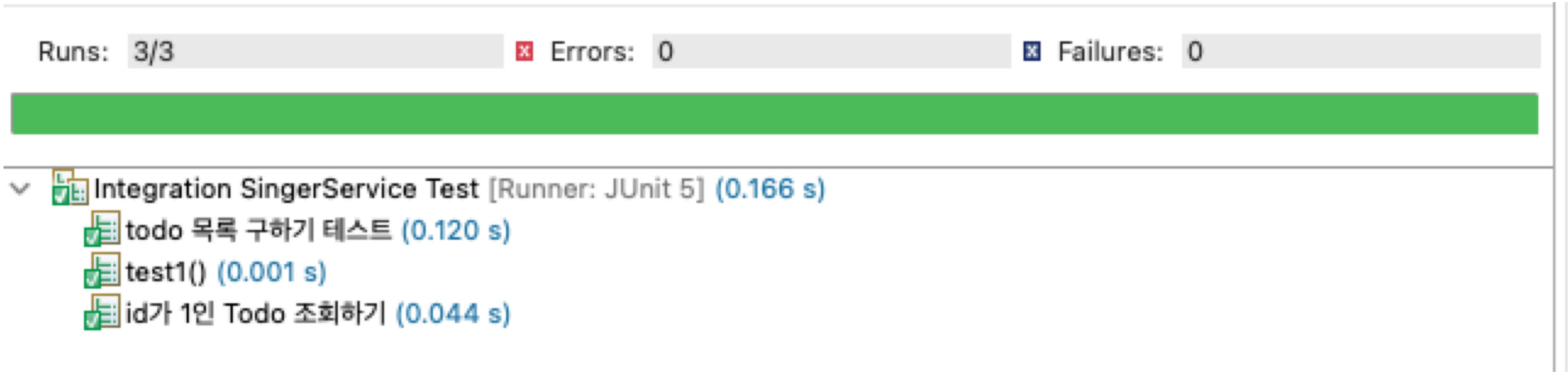
```
    @Test
    public void test1() {
            logger.info("test1");
    }

// 테스트가 실행될때 test-data.sql이 실행되고 테스트가 종료되면 clean-up.sql이 실행된다.
    @Test
    @DisplayName("todo 목록 구하기 테스트")
    @SqlGroup({
                @Sql(value = "classpath:db/test-data.sql",
                            config = @SqlConfig(encoding = "utf-8", separator = ";", commentPrefix = "--"),
                            executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
                @Sql(value = "classpath:db/clean-up.sql",
                            config = @SqlConfig(encoding = "utf-8", separator = ";", commentPrefix = "--"),
                            executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD),
    })
    public void findAll() {
            logger.info("findAll");
            List<Todo> result = todoService.getTotos();
            assertNotNull(result); // result가 Null이 아니면 성공
            assertEquals(3, result.size()); // result결과가 3건이면 성공
    }
```

```
        @Test
        @DisplayName("id가 1인 Todo 조회하기")
        @SqlGroup({
                        @Sql(value = "classpath:db/test-data.sql",
                                        config = @SqlConfig(encoding = "utf-8", separator = ";", commentPrefix = "--"),
                                        executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD),
                        @Sql(value = "classpath:db/clean-up.sql",
                                        config = @SqlConfig(encoding = "utf-8", separator = ";", commentPrefix = "--"),
                                        executionPhase = Sql.ExecutionPhase.AFTER_TEST_METHOD),
        })
        public void testFindByFirstNameAndLastNameOne() throws Exception {
                logger.info("testFindByFirstNameAndLastNameOne");
                Todo todo = todoService.getToto(1L);
                assertNotNull(todo);
        }

}
```

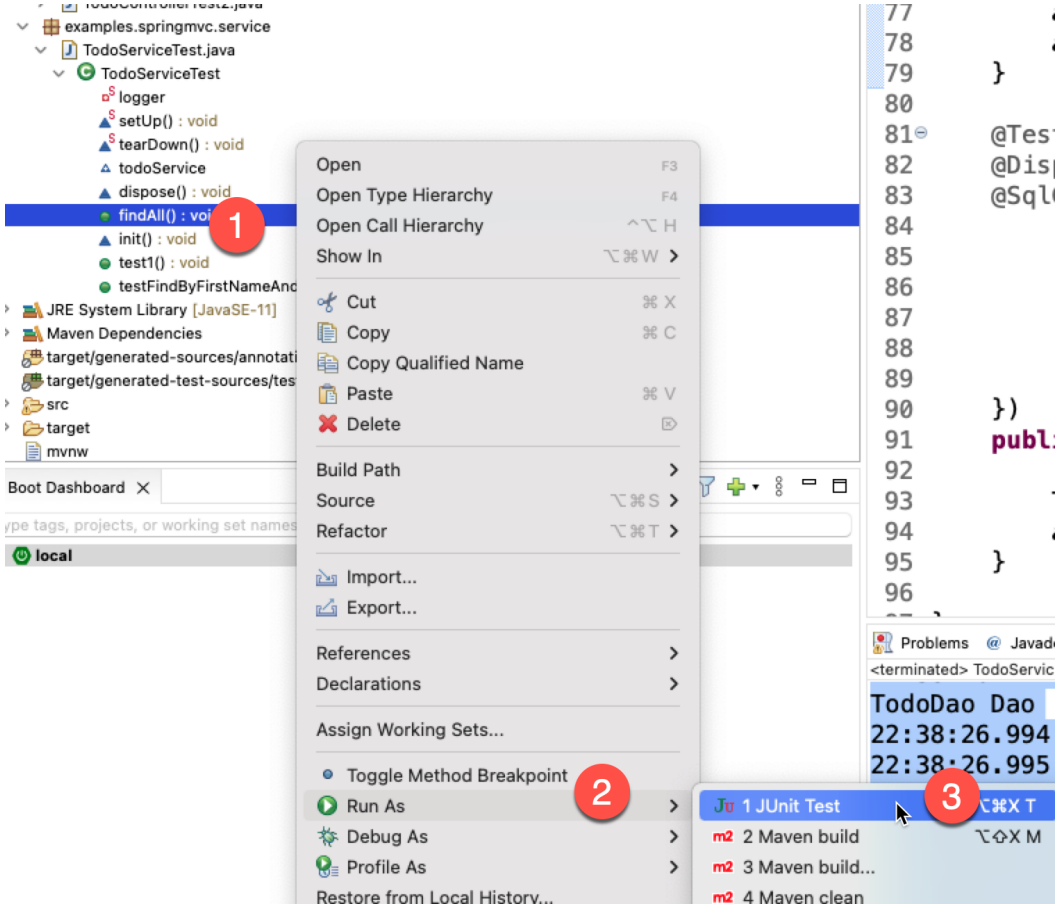
실행 결과

```
22:38:26.711 [main] INFO  e.s.s.TodoServiceTest - --> @BeforeAll - 이 클래스의 모든 테스트 메소드를 실행하기 전에 실행합니다.
TodoDao Dao
22:38:26.994 [main] INFO  e.s.s.TodoServiceTest - --> @BeforeEach - 이 클래스의 각 테스트 메소드 실행하기 전에 실행합니다.
22:38:26.995 [main] INFO  e.s.s.TodoServiceTest - findAll
22:38:27.030 [main] INFO  e.s.s.TodoServiceTest - --> @AfterEach - 이 클래스의 각 테스트 메소드 실행한 다음에 실행합니다.
22:38:27.044 [main] INFO  e.s.s.TodoServiceTest - --> @BeforeEach - 이 클래스의 각 테스트 메소드 실행하기 전에 실행합니다.
22:38:27.044 [main] INFO  e.s.s.TodoServiceTest - test1
22:38:27.044 [main] INFO  e.s.s.TodoServiceTest - --> @AfterEach - 이 클래스의 각 테스트 메소드 실행한 다음에 실행합니다.
22:38:27.068 [main] INFO  e.s.s.TodoServiceTest - --> @BeforeEach - 이 클래스의 각 테스트 메소드 실행하기 전에 실행합니다.
22:38:27.068 [main] INFO  e.s.s.TodoServiceTest - testFindByFirstNameAndLastNameOne
22:38:27.077 [main] INFO  e.s.s.TodoServiceTest - --> @AfterEach - 이 클래스의 각 테스트 메소드 실행한 다음에 실행합니다.
22:38:27.090 [main] INFO  e.s.s.TodoServiceTest - --> @AfterAll - 이 클래스의 모든 테스트 메소드를 실행한 다음에 실행합니다.
```

실행결과


Runs: 3/3
Errors: 0
Failures: 0
Integration SingerService Test [Runner: JUnit 5] (0.166 s)
todo 목록 구하기 테스트 (0.120 s)
test1() (0.001 s)
id가 1인 Todo 조회하기 (0.044 s)

특정 메소드만 테스트하기



Controller 테스트하기

MockMvc

- MockMvc는 웹 어플리케이션을 서버에 배포하지 않고도 스프링 MVC의 동작을 재현할 수 있는 클래스이다. 테스트는 MockMvc를 이용해 테스트용으로 확장된 DispatcherServlet으로 요청을 보낸다. 그리고 DispatcheServlet은 요청을 받아 매핑 정보를 보고 이에 맞는 핸들러 메소드를 호출한다. 최종적으로 테스트 메소드는 MockMvc가 반환하는 실행 결과를 받아 실행 결과가 맞는지 검증한다.
- MockMvcBuilders.standaloneSetup()
    - MockMvc 인스턴스를 생성해주는 메소드이다. standaloneSetup은 테스트에서 사용할 컨트롤러 하나만 지정해서 생성한다.

- .perform()
    - 괄호 내에 있는 문자열을 URL로 요청을 보내는 메소드이다.
- .andExpect()
    - 괄호 내에 있는 메소드들을 이용해 어떤 결과를 예측하는지 작성하는 메소드이다. Status(), content() 등 내부에서 사용할 수 있는 다양한 메소드가 있다.

Mockito

- Controller를 테스트 할때 Controller가 사용하는 Service는 잘 돌아간다고 가정하고 테스트하고 싶은 경우가 있다. 오픈 소스인 Mockito는 내가 원하는 값을 반환하는 가짜 객체를 생성할 수 있도록 해준다.

Mockito

- 이병헌의 유명한 대사. 모히또가서 몰디브 한잔 할때의 그....
- @Mock
    - mock 객체를 생성한다. 말그대로 가짜 객체가 바이트 코드 조작으로 만들어진다. 이 객체에 녹음기 녹음하듯이 원하는 형태로 동작하도록 할 수 있다.
- @InjectMocks
    - @InjectMocks라는 어노테이션이 존재하는데, @Mock이 붙은 목객체를 @InjectMocks이 붙은 객체에 주입시킬 수 있다.실무에서는 @InjectMocks(Service) @Mock(DAO) 이런식으로 Service테스트 목객체에 DAO 목객체를 주입시켜 사용한다.

Controller 테스트하기

```
package examples.springmvc.controller;

// static 메소드를 static import 하고 있다.
import static org.hamcrest.CoreMatchers.containsString;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

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

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import com.fasterxml.jackson.databind.ObjectMapper;

import examples.springmvc.config.ApplicationConfig;
import examples.springmvc.config.MvcConfig;
import examples.springmvc.domain.Todo;
import examples.springmvc.service.TodoService;
```

```
// MvcMock을 사용하려면 아래와 같은 애노테이션이 설정되야 한다.
// 그렇지 않으면 웹 환경이 구성 안된다.
@ExtendWith(SpringExtension.class)
@WebAppConfiguration
@ContextConfiguration(classes = {MvcConfig.class, ApplicationConfig.class})
public class TodoControllerTest2 {

    // TodoService가 반환할 값
    private final List<Todo> todos = new ArrayList<>();

    // 가짜 객체가 주입될 Controller를 선언한다.
    @InjectMocks
    private TodoController todoController;

    // 가짜 객체
    @Mock
    private TodoService todoService;

    private MockMvc mockMvc;
```

```
    // 테스트 코드가 실행되기 전에 Mockito환경을 초기화하고
    // MockMvc를 초기화 한다.
    @BeforeEach
    public void initTodos() {
        Todo todo = new Todo();
        todo.setId(1l);
        todo.setTodo("haha");
        todo.setDone(false);
        todos.add(todo);

        // 해당 코드를 호출하지 않으면 @Mock, @InjectMock이 동작하지 않는다.
        MockitoAnnotations.openMocks(this);
        // todoController를 테스트 하기 위한 mockMvc를 선언한다.
        this.mockMvc = MockMvcBuilders.standaloneSetup(todoController).build();
    }
```

```
    @Test
    public void testList() throws Exception {
        ObjectMapper objectMapper = new ObjectMapper();

        // tdoService목객체의 getTodos()메소드를 호출하면
        // todos가 리턴되도록 설정한다.
        when(todoService.getTotos()).thenReturn(todos);

        // mockMvc에게 GET방식으로 api/todos를 호출한다.
        // 200 OK가 나오고, todos를 JSON으로 변환한 값과 api의 응답값이 같을 경우 성공하도록 한다.
        // get, status, content 등은 static import되어 있다. 초보자들은 이 부분이 힘들수 있다.
        this.mockMvc.perform(get("/api/todos"))
        .andExpect(status().isOk())
        .andExpect(content().string(containsString(objectMapper.writeValueAsString(todos))))
        .andDo(print());

        // Controller에서 목 객체인 todoService의 getTodos()를 한번 호출했는지 확인한다.
        //1번 호출되지 않았다면 Controller는 이상하게(?)동작했다는 의미이다.
        verify(todoService, times(1)).getTotos();
    }


}
```

감사합니다.