一、使用Mockito编码完成接口测试

之前我们是使用postman或RestfulTool插件来进行接口测试的，这里使用编码的方式来进行 接口测试。

一、说明

1.1背景（为什么要写代码做测试）

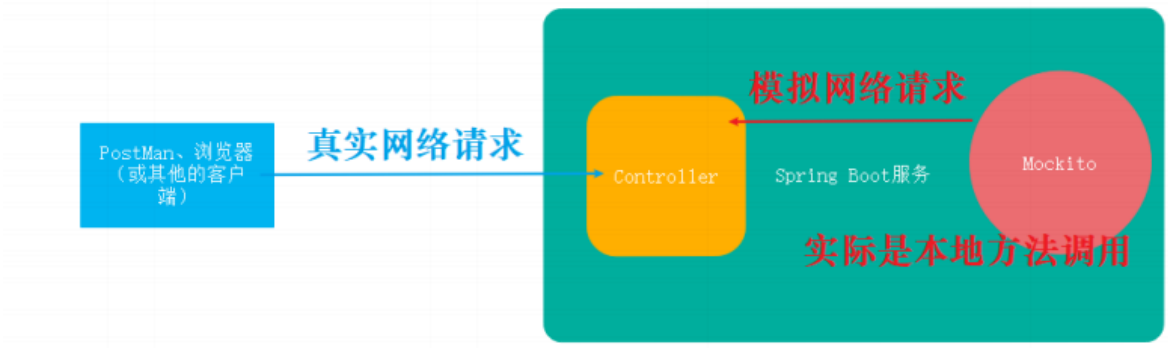
因为在做系统的自动化持续集成的时候，会要求自动的做单元测试， 只有所有的单元测试都跑通了，才能打包构建。比如：使用maven 在打包之前将所有的测试用例执行一遍。这里重点是自动化，所以postman这种工具很难插入到持续集成的自动化流程中去。

1.2回顾（Junit测试框架）

junit5	特点
@Test	声明一个测试方法
@BeforeAll	在当前类的所有测试方法之前执行。注解在【静态方法】上
@AfterAll	在当前类中的所有测试方法之后执行。注解在【静态方法】上
@BeforeEach	在每个测试方法之前执行。注解在【非静态方法】上
@AfterEach	在每个测试方法之后执行。注解在【非静态方法】
@ExtendWith(SpringExtension.class)	类class定义上

1.3Mockito测试框架

Mockito是GitHub上使用最广泛的Mock框架,并与JUnit结合使用.Mockito 框架可以创建和配置mock对象.使用Mockito简化了具有外部依赖的类的 测试开发。Mockito测试框架可以帮助我们模拟HTTP请求，从而达到在服 务端测试目的。因为其不会真的去发送HTTP请求，而是模拟HTTP请求内 容，从而节省了HTTP请求的网络传输，测试速度更快。



二、示例

2.1轻量级测试（不启动servlet容器和 Spring 上下文，自然也就无法实现依赖注入这就导致它在从控制层到持久层全流程测试中有很 大的局限性。）

前提：实体类头部有@Data @Builder @AllArgsConstructor @NoArgsConstructor这四个注解

- 1.打开BookController类，在BookController上 Alt+回车 选择 create Test。这里在弹出的界面 Generate勾选setUp/@Before和tearDown/@After，Member里勾选saveBook（book：Book）：AjaxResponse
- 2.类头上只保留@Slf4j注解
- 3.声明mockMvc并new一个对象

```
    private static MockMvc mockMvc;
    @BeforeAll
    static void setup(){
        mockMvc = MockMvcBuilders.standaloneSetup(new
BookController()).build();
     }
```

- 4.在@Test下编写测试代码

```
 @Test
    void saveBook() throws Exception{
     String book = "{\n" +
             "  \"id\": 1,\n" +
             "  \"author\": \"szz\",\n" +
             "  \"title\": \"Spring Boot从入门到精通\",\n" +
             "  \"content\": \"Spring Boot从入门到精通\",\n" +
             "  \"createTime\": \"2021-03-07 21:54:10\",\n" +
             "  \"readers\": [{\"name\": \"aaa\",\"age\": 20},{\"name\":
\"bbb\",\"age\": 19}]\n" +
             "}";
     MvcResult result = mockMvc.perform(
             MockMvcRequestBuilders
                     .request(HttpMethod.POST,
                             "/api/v1/books")
                     .contentType("application/json")
                     .content(book)
     )
             .andExpect(MockMvcResultMatchers.status().isOk())


.andExpect(MockMvcResultMatchers.jsonPath("$.data.author").value("szz"))


.andExpect(MockMvcResultMatchers.jsonPath("$.data.readers[0].age").value(20))
             .andDo(print())
             .andReturn();
     result.getResponse().setCharacterEncoding("UTF-8");
     log.info(result.getResponse().getContentAsString());
    }
```

- 5@Test旁边的小运行按钮运行一下

2.2真实servlet容器环境下的测试

上面的测试不支持@Resource和 @AutoWired注解

- 1.在测试类上面额外加上这样两个注 解，并且mockMvc对象使用@Resource自动注入，删掉Before注解及 setUp函数。

```
@AutoConfigureMockMvc
@SpringBootTest
@ExtendWith(SpringExtension.class)
```

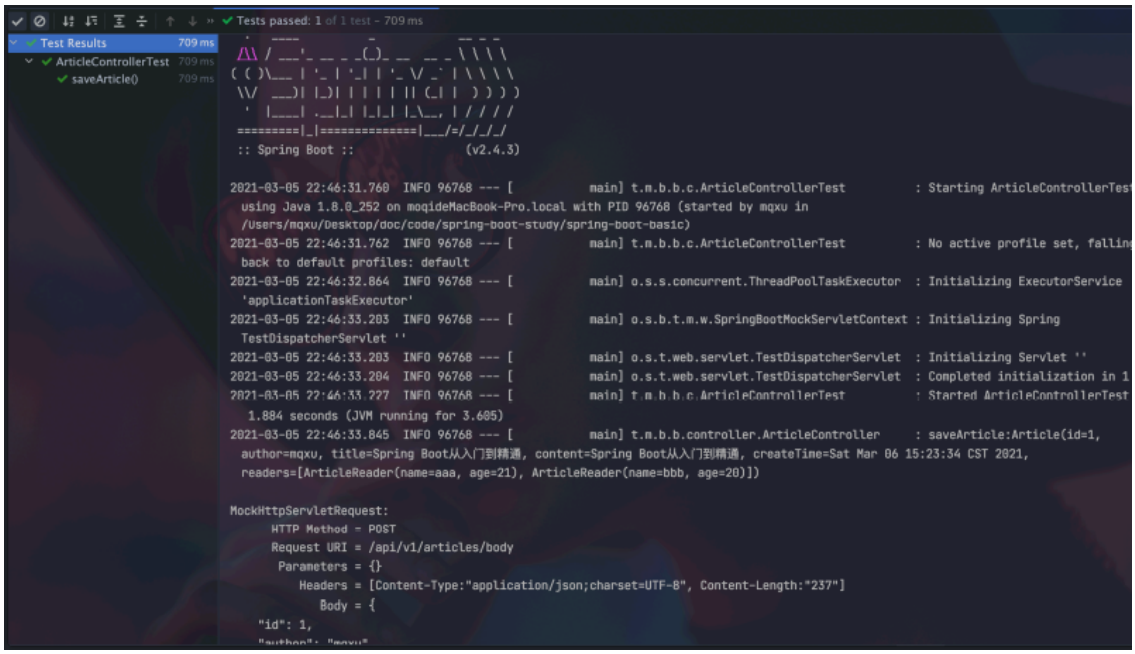
```
@Slf4j
@AutoConfigureMockMvc
@SpringBootTest
@ExtendWith(SpringExtension.class)
class ArticleControllerTest {

    //mock对象
    @Resource
    private MockMvc mockMvc;

    //在所有测试方法执行之前进行mock对象初始化
    //@BeforeAll
    //static void setUp() {
    //    mockMvc = MockMvcBuilders.standaloneSetup(new ArticleController()).build();
    //}
```

添加注解

注入 MockMvc 实例

删掉

- 启动测试一下，发现与之前的区别



该测试方法真实的启动了一个 tomcat容器、以及Spring 上下文，所以我们可以进行依赖注入（@Resource）。实现的效果和使用MockMvcBuilders构建MockMVC对 象的效果是一样的，但是有一个非常明显的缺点：每次做一个接口测试， 都会真实的启动一次servlet容器，Spring上下文加载项目里面定义的所有 的Bean，导致执行过程很缓慢。

2.3关于一下注解

2.3.1@SpringBootTest 注解

是用来创建Spring的上下文ApplicationContext，保证测试在上下文环境 里运行。单独使用@SpringBootTest不会启动servlet容器。所以只是使用 SpringBootTest 注解，不可以使用@Resource和@Autowired等注解 进行bean的依赖注入。（准确的说是可以使用，但被注解的bean为 null）。

2.3.2 @ExtendWith(@RunWith注解)

- RunWith方法为我们构造了一个的Servlet容器运行运行环境，并在此 环境下测试。然而为什么要构建servlet容器？因为使用了依赖注入， 注入了MockMvc对象，而在上一个例子里面是我们自己new的。
- 而@AutoConfigureMockMvc注解，该注解表示mockMvc对象由 spring 依赖注入构建，你只负责使用就可以了。这种写法是为了让测 试在servlet容器环境下执行。

  简单的说：如果你单元测试代码使用了“依赖注入@Resource”就必须加 上@ExtendWith，如果你不是手动new MockMvc对象就加上 @AutoConfigureMockMvc

  实际上@SpringBootTest 注解注解已经包含了 @ExtendWith注 解，如果使用了前者，可以忽略后者！

2.3.3@Transactional

该注解加在方法上可以使单元测试进行事务回滚，以保证数据库表中没有 因测试造成的垃圾数据，因此保证单元测试可以反复执行； 但是不建议这么做，使用该注解会破坏测试真实性。请参考这篇文章详细理解：

不要在 Spring Boot 集成测试中使用 @Transactiona

http://www.zimug.com/java/spring/%e4%b8%8d%e8%a6%81%e5%9c%a8spring%e5%8d%95%e5%85%83%e6%b5%8b%e8%af%95%e4%b8%ad%e4%bd%bf%e7%94%a8-transactional%e6%b3%a8%e8%a7%a3/.html

二、使用Swagger2构建API文档

一、背景（为什么要发布API接口文档)

当下很多公司都采取前后端分离的开发模式，前端和后端的工作由不同的 工程师完成。在这种开发模式下，维护一份及时更新且完整的API 文档将 会极大的提高我们的工作效率。传统意义上的文档都是后端开发人员使用 word编写的，相信大家也都知道这种方式很难保证文档的及时性，这种 文档久而久之也就会失去其参考意义，反而还会加大我们的沟通成本。而 Swagger 给我们提供了一个全新的维护 API 文档的方式，下面我们就来 了解一下它的优点：

- 代码变，文档变。只需要少量的注解，Swagger 就可以根据代码自动 生成 API 文档，很好的保证了文档的时效性。
- 代码变，文档变。只需要少量的注解，Swagger 就可以根据代码自动 生成 API 文档，很好的保证了文档的时效性。
- 跨语言性，支持 40 多种语言。
- Swagger UI 呈现出来的是一份可交互式的 API 文档，我们可以直接 在文档页面尝试 API 的调用，省去了准备复杂的调用参数的过程。
- Swagger UI 呈现出来的是一份可交互式的 API 文档，我们可以直接 在文档页面尝试 API 的调用，省去了准备复杂的调用参数的过程。

二、整合swagger2生成文档

当我们在使用Spring MVC写接口的时候，为了生成API文档，为了方便整 合Swagger，都是用这个SpringFox的这套封装。但是，自从2.9.2版本更 新之后，就一直没有什么动静，也没有更上Spring Boot的大潮流。 现在SpringFox出了一个starter，看了一下功能，虽然还不完美，但相较 于之前我们自己的轮子来说还是好蛮多的。来看看这个版本有些什么亮 点：

- Spring 5，Webflux 支持（仅请求映射支持，尚不支持功能端点）
- Spring 5，Webflux 支持（仅请求映射支持，尚不支持功能端点）
- Spring Boot 支持 springfox-boot-starter 依赖性（零配置，自动配 置支持）
- 具有自动完成功能的文档化配置属性
- 更好的规范兼容性
- 支持 OpenApi 3.0.3
- 几乎零依赖性（唯一需要的库是 spring-plugin、pswagger-core）
- 现有的 swagger2 注释将继续有效，并丰富 open API 3.0 规范

  对于这次的更新，我觉得比较突出的几点：Webflux的支持，目前的轮子 就没有做到；对OpenApi 3的支持；以及对Swagger 2的兼容（可以比较 方便的做升级了）。

1.加依赖（可能时间会长，有耐心）

```
<dependency>
 <groupId>io.springfox</groupId>
 <artifactId>springfox-boot-starter</artifactId>
 <version>3.0.0</version>
<dependency>
```

2.在SpringBootBasicApplication类（启动主类）加注解

```
@SpringBootApplication
@EnableOpenApi
public class SpringBootBasicApplication {
 public static void main(String[] args) {

SpringApplication.run(SpringBootBasicApplication.class,
args);
 }
}
```

3.书写swagger注解

约定大于配置

- **接口层面常用注解：类级别注解@Api，方法级别注解@ApiOperation，参数级别注解@ApiParam**
- **实体类层面常用注解：类级别@ApiModel,属性级别@ApiModeProperty**

Bookcontroller类

```
package top.szz.boot.basic.controller;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;
import top.szz.boot.basic.controller.dto.AjaxResponse;
import top.szz.boot.basic.controller.dto.Param;
import top.szz.boot.basic.entity.Book;
import top.szz.boot.basic.entity.BookReader;

import java.util.Arrays;
import java.util.Date;
import java.util.List;


/**
 * @author 囧态汗
 * @date 2021/3/4
 * @description BookController
 */
@RestController
@RequestMapping(value = "/api/v1/books")
@Slf4j
@Api(tags = "书管理接口")
public class BookController {

    @ApiOperation("获取全部书本")
    @GetMapping("all")
    public AjaxResponse selectBooks(){
        BookReader[] readers ={
                BookReader.builder()
                .name("aaa")
                .age(20)
                .build(),
                BookReader.builder()
```

```
                .name("bbb")
                .age(19)
                .build(),

        };
        List<BookReader> readerList = Arrays.asList(readers);
        Book book1 = Book.builder()
                .id(123)
                .author("szz")
                .title("SpringBoot")
                .content("SpringBoot从入门到精通")
                .createdTime(new Date())
                .readers(readerList)
                .build();

        Book book2 = Book.builder()
                .id(456)
                .author("szz")
                .title("Vue.js")
                .content("Vue.js从入门到精通")
                .createdTime(new Date())
                .readers(readerList)
                .build();
        Book[] books = {book1,book2};
        List<Book> bookList =Arrays.asList(books);
        return AjaxResponse.success(bookList);
    }
    @ApiOperation("URL传参，根据id获取书本")
    @GetMapping("{id}")
    public AjaxResponse getBook(@PathVariable int id){
        Book book = Book.builder()
                .id(id)
                .author("szz")
                .title("Java")
                .content("Java")
                .createdTime(new Date())
                .build();
        return AjaxResponse.success(book);
    }
    @ApiOperation("增加书本")
    @PostMapping()
    public AjaxResponse saveBook(@RequestBody Book book){
        log.info("saveBook:" + book);
        return AjaxResponse.success(book);
    }

//    @PutMapping()
//    public AjaxResponse updateBook(@RequestParam int id,@RequestParam
String title){
//        Book book = Book.builder()
//                .id(111)
//                .author("szz")
//                .title("Java")
//                .content("Java")
//                .createdTime(new Date())
//                .build();
//        log.info("book:" + book);
//
```

```
//        book.setId(id);
//        book.setTitle(title);
//
//        log.info("book:" + book);
//        return AjaxResponse.success(book);
//    }
    @ApiOperation("修改书本")
    @PutMapping()
    public AjaxResponse updateBook(@RequestBody Book book){
        Book book1 = Book.builder()
                .id(111)
                .author("szz")
                .title("Java")
                .content("Java")
                .createdTime(new Date())
                .build();
        log.info("book:" + book1);

        book1.setId(book.getId());
        book1.setTitle(book.getTitle());

        log.info("book:" + book1);
        return AjaxResponse.success(book1);
    }

//    @DeleteMapping("{id}")
//    public AjaxResponse deleteBook(@PathVariable int id){
//        log.info("id:"+id);
//        return AjaxResponse.success();
//    }


//    @DeleteMapping()
//    public AjaxResponse deleteBook(@RequestParam(value = "id",defaultValue
= "111")int idd,@RequestParam("title")String ti){
//        log.info("id:" + idd);
//        log.info("title:"+ti);
//        return AjaxResponse.success();
//    }
//

//       @DeleteMapping()
//       public AjaxResponse deleteBook(int id,String title){
//          log.info("id:" + id);
//          log.info("title:"+title);
//          return AjaxResponse.success();
//    }

//        @DeleteMapping()
//        public AjaxResponse deleteBook(@RequestParam("id") int
idd,@RequestParam("title") String ti){
//            log.info("id:" + idd);
//            log.info("title:"+ti);
//            return AjaxResponse.success();
//}

    @ApiOperation("删除书本")
    @DeleteMapping()
```

```
    public AjaxResponse deleteBook(@RequestBody Param param){
        log.info("id:" + param.getId());
        log.info("title:"+param.getTitle());
        return AjaxResponse.success(param);
    }

}
```

AjaxResponse类

```
package top.szz.boot.basic.controller.dto;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @author 囧态汗
 * @date 2021/3/4
 * @description AjaxResponse
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
@ApiModel("统一响应结果")
public class AjaxResponse {
    /**
     * 请求响应状态码（200、400、500）
     */
    @ApiModelProperty("请求响应状态码")
    private Integer code;
    /**
     * 请求结果描述信息
     */
    @ApiModelProperty("请求结果描述信息")
    private String message;
    /**
     * 请求返回数据
     */
    @ApiModelProperty("请求返回数据")
    private Object data;

    public static AjaxResponse success(){
        AjaxResponse ajaxResponse =new AjaxResponse();
        ajaxResponse.setCode(200);
        ajaxResponse.setMessage("请求响应成功");
        return ajaxResponse;
    }
    public static AjaxResponse success(Object obj){
        AjaxResponse ajaxResponse =new AjaxResponse();
        ajaxResponse.setCode(200);
        ajaxResponse.setMessage("请求响应成功");
        ajaxResponse.setData(obj);
        return ajaxResponse;
```

```
    }
    public static AjaxResponse success(Object obj, String
            message) {
        AjaxResponse ajaxResponse = new AjaxResponse();
        ajaxResponse.setCode(200);
        ajaxResponse.setMessage(message);
        ajaxResponse.setData(obj);
        return ajaxResponse;
    }
}
```

Book类

```
package top.szz.boot.basic.entity;

import com.fasterxml.jackson.annotation.*;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;
import java.util.List;

/**
 * @author 囧态汗
 * @date 2021/3/4
 * @description Book
 */
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor

//@JsonPropertyOrder(value = {"content","title"})
@ApiModel("书本基本信息")
public class Book {
 //   @JsonIgnore
    @ApiModelProperty("id")
    private Integer id;

   // @JsonProperty("name")
    @ApiModelProperty("作者")
    private String author;

    @ApiModelProperty("标题")
    private String title;

    @ApiModelProperty("内容")
    private String content;

    @ApiModelProperty("创建时间")
    @JsonInclude(JsonInclude.Include.NON_NULL)
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss",timezone = "GMT+8")
```

```
    private Date createdTime;

    @ApiModelProperty("读者列表")
    private List<BookReader> readers;
}
```

4.在浏览器启动服务

http://localhost:8080/swagger-ui/index.html

每个接口都可测试，进行try