编号：

2020 - 2021学年第 二 学期



**实 验 报 告**

实验课程名称 软件架构与应用开发

专 业 班 级 计算机1803

学 生 姓 号 31801150

学 生 姓 名 张帅

实验指导教师 彭彬

一. 实验目的和要求

**实验要求：**

1.从零模仿实现L05对应的三个示例（ L05AopLog、 L05IOC、 L05Error）

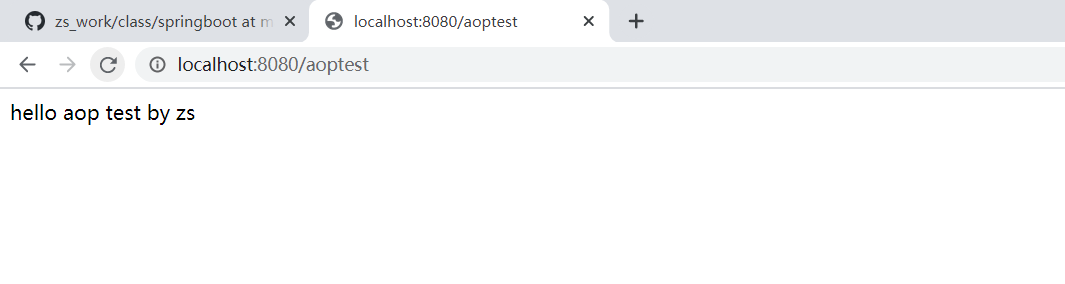
2. 利用IoC和AOP技术，模拟学生信息的增删改查询操作（学生类、增删改操作Service类使用IoC，使用AOP输出各种操作的调用日志），对无效的访问命令给出自定义错误提示，并编写测试类进行测试

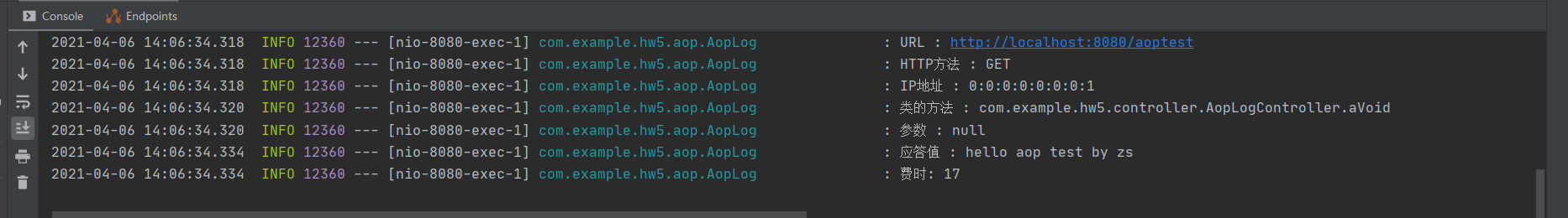
二. 实验内容

1.从零模仿实现L05对应的三个示例（ L05AopLog、 L05IOC、 L05Error）

示例1：

@Aspect  
  
@Component  
public class AopLog {  
 private Logger logger = LoggerFactory.*getLogger*(this.getClass());  
 ThreadLocal<Long> startTime = new ThreadLocal<>();  
 @Pointcut("execution(public \* com.example.hw5..\*.\*(..))")  
 public void aopWebLog() {  
 }  
  
 @Before("aopWebLog()")  
 public void doBefore(JoinPoint joinPoint) throws Throwable {  
 startTime.set(System.*currentTimeMillis*());  
  
 ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
 HttpServletRequest request = attributes.getRequest();  
 logger.info("URL : " + request.getRequestURL().toString());  
 logger.info("HTTP方法 : " + request.getMethod());  
 logger.info("IP地址 : " + request.getRemoteAddr());  
 logger.info("类的方法 : " + joinPoint.getSignature().getDeclaringTypeName() + "." + joinPoint.getSignature().getName());  
 logger.info("参数 : " + request.getQueryString());  
 }  
  
 @AfterReturning(pointcut = "aopWebLog()",returning = "retObject")  
 public void doAfterReturning(Object retObject) throws Throwable {  
  
 logger.info("应答值 : " + retObject);  
 logger.info("费时: " + (System.*currentTimeMillis*() - startTime.get()));  
 }  
  
  
 @AfterThrowing(pointcut = "aopWebLog()", throwing = "ex")  
 public void addAfterThrowingLogger(JoinPoint joinPoint, Exception ex) {  
 logger.error("执行 " + " 异常", ex);  
 }  
  
}



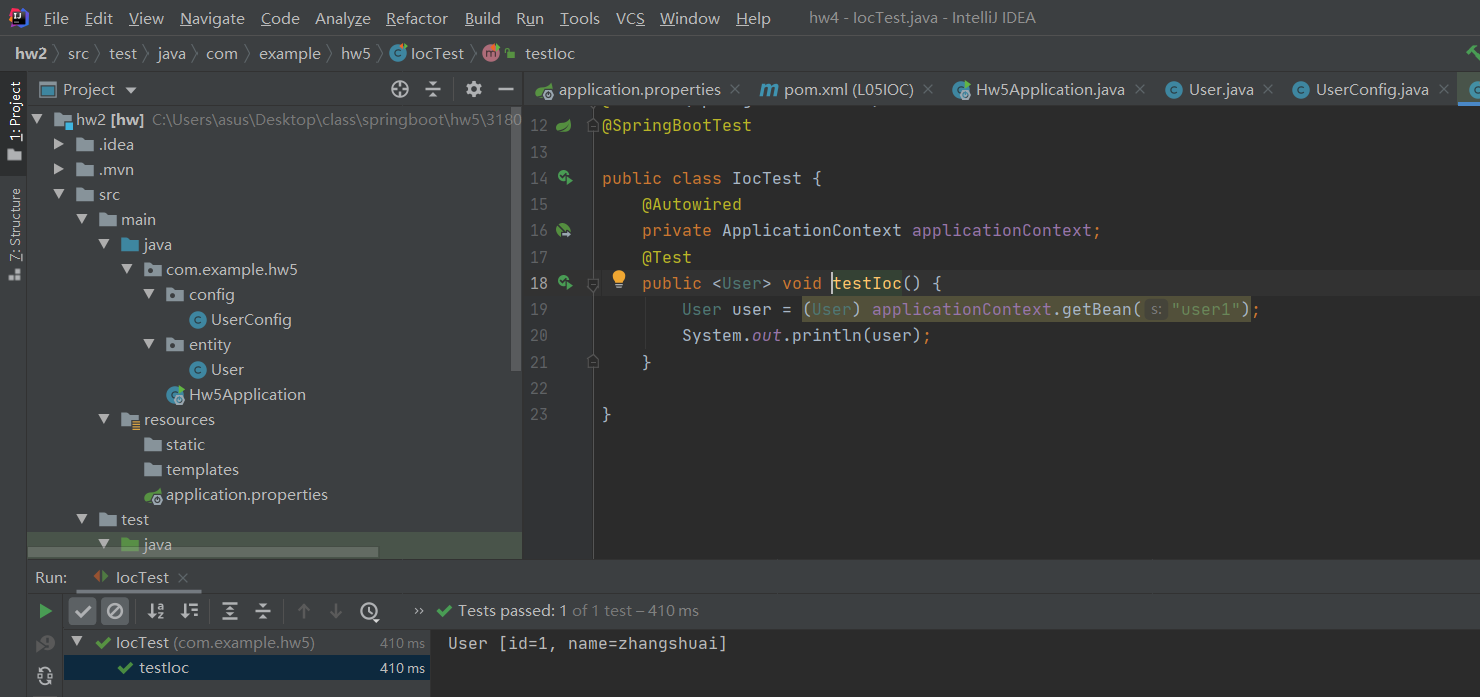


示例2：

@Configuration  
public class UserConfig {  
 @Bean("user1")  
 public User user() {  
 User user = new User();  
 user.setId(1);  
 user.setName("zhangshuai");  
 return user;  
 }  
  
}

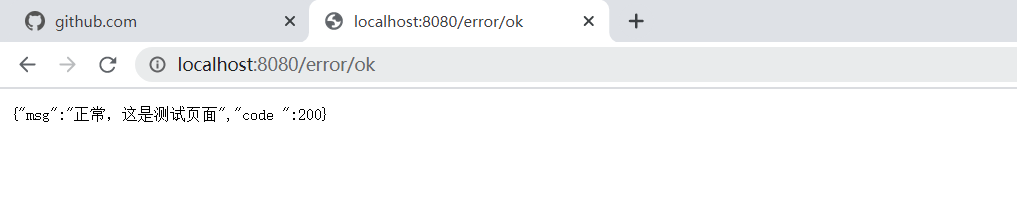
public class User implements Serializable {  
 private int id;  
 private String name;  
 @Override  
 public String toString() {  
 return "User [id=" + id + ", name=" + name + "]";  
 }   
 public int getId() {  
 return id;  
 }  
 public void setId(int id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 }

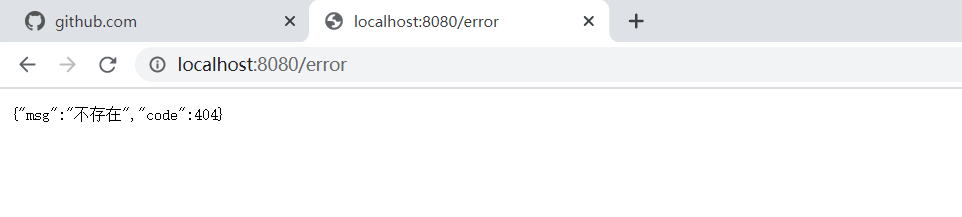
public class IocTest {  
 @Autowired  
 private ApplicationContext applicationContext;  
 @Test  
 public <User> void testIoc() {  
 User user = (User) applicationContext.getBean("user1");  
 System.*out*.println(user);  
 }  
  
}



示例三：

@RestController  
@RequestMapping("/error")  
public class TestErrorController implements ErrorController {  
  
 @Override  
 public String getErrorPath() {  
 return null;  
 }  
  
  
 @RequestMapping  
 public Map<String, Object> handleError() {  
 Map<String, Object> map = new HashMap<String, Object>();  
 map.put("code", 404);  
 map.put("msg", "不存在");  
 return map;  
 }  
  
 @RequestMapping("/ok")  
 @ResponseBody  
 public Map<String, Object> noError() {  
 Map<String, Object> map = new HashMap<String, Object>();  
 map.put("code ", 200);  
 map.put("msg", "正常，这是测试页面");  
  
 return map;  
 }  
  
  
}





2. 利用IoC和AOP技术，模拟学生信息的增删改查询操作（学生类、增删改操作Service类使用IoC，使用AOP输出各种操作的调用日志），对无效的访问命令给出自定义错误提示，并编写测试类进行测试使用

AOP输出操作：

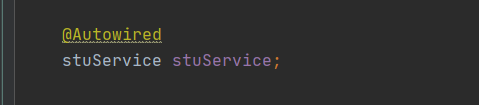
package com.example.hw5.aop;  
  
//import org.apache.commons.lang3.builder.ToStringBuilder;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.\*;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Component;  
import org.springframework.web.context.request.RequestContextHolder;  
import org.springframework.web.context.request.ServletRequestAttributes;  
  
import javax.servlet.http.HttpServletRequest;  
@Aspect  
@Component  
public class AopLog {  
 private Logger logger = LoggerFactory.*getLogger*(this.getClass());  
 //线程局部的变量,解决多线程中相同变量的访问冲突问题。  
 ThreadLocal<Long> startTime = new ThreadLocal<>();  
//定义切点  
 @Pointcut("execution(public \* com.example.hw5..\*.\*(..))")  
 public void aopWebLog() {  
 }  
  
 @Before("aopWebLog()")  
 public void doBefore(JoinPoint joinPoint) throws Throwable {  
 startTime.set(System.*currentTimeMillis*());  
 // 接收到请求，记录请求内容  
 ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
 HttpServletRequest request = attributes.getRequest();  
  
 // 记录下请求内容  
 logger.info("URL : " + request.getRequestURL().toString());  
 logger.info("HTTP方法 : " + request.getMethod());  
 logger.info("IP地址 : " + request.getRemoteAddr());  
 logger.info("类的方法 : " + joinPoint.getSignature().getDeclaringTypeName() + "." + joinPoint.getSignature().getName());  
 logger.info("参数 : " + request.getQueryString());  
 }  
  
 @AfterReturning(pointcut = "aopWebLog()",returning = "retObject")  
 public void doAfterReturning(Object retObject) throws Throwable {  
 // 处理完请求，返回内容  
 logger.info("应答值 : " + retObject);  
 logger.info("费时: " + (System.*currentTimeMillis*() - startTime.get()));  
 }  
  
 //抛出异常后通知（After throwing advice） ： 在方法抛出异常退出时执行的通知。  
 @AfterThrowing(pointcut = "aopWebLog()", throwing = "ex")  
 public void addAfterThrowingLogger(JoinPoint joinPoint, Exception ex) {  
 logger.error("执行 " + " 异常", ex);  
 }  
  
}



IOC Service层：  
public interface stuService {  
  
 ArrayList<stuEntity> findall();  
  
 void add(stuEntity stu);  
  
 void remove(int id);  
  
 stuEntity findbyid(int id);  
  
 void update(stuEntity stu);  
}

@Service  
public class stuServiceImpl implements stuService {  
  
 @Autowired  
 private MongoTemplate mongoTemplate;  
  
 @Autowired  
 stuDAO stuDAO;  
  
 @Override  
 public ArrayList<stuEntity> findall() {  
 return (ArrayList<stuEntity>) stuDAO.findAll();  
 }  
  
 @Override  
 public void add(stuEntity stu) {  
 stuDAO.save(stu);  
 }  
  
 @Override  
 public void remove(int id) {  
 Query query = new Query(Criteria.*where*("id").is(id));//查找其id字段的值与传入参数id相等的数据  
 mongoTemplate.remove(query, stuEntity.class);  
 }  
  
 @Override  
 public stuEntity findbyid(int id) {  
 Query query = new Query(Criteria.*where*("id").is(id));  
 stuEntity stuEntity = mongoTemplate.findOne(query,stuEntity.class);  
 return stuEntity;  
  
 }  
  
 @Override  
 public void update(stuEntity stu) {  
 Query query = new Query(Criteria.*where*("id").is(stu.getId()));  
 Update update = new Update().set("name",stu.getName()).set("id",stu.getId()).set("email",stu.getEmail()).set("number",  
 stu.getNumber()).set("sex",stu.getSex()).set("age",stu.getAge());  
 mongoTemplate.updateFirst(query,update,stuEntity.class);  
  
 }  
  
  
}

注入service层：

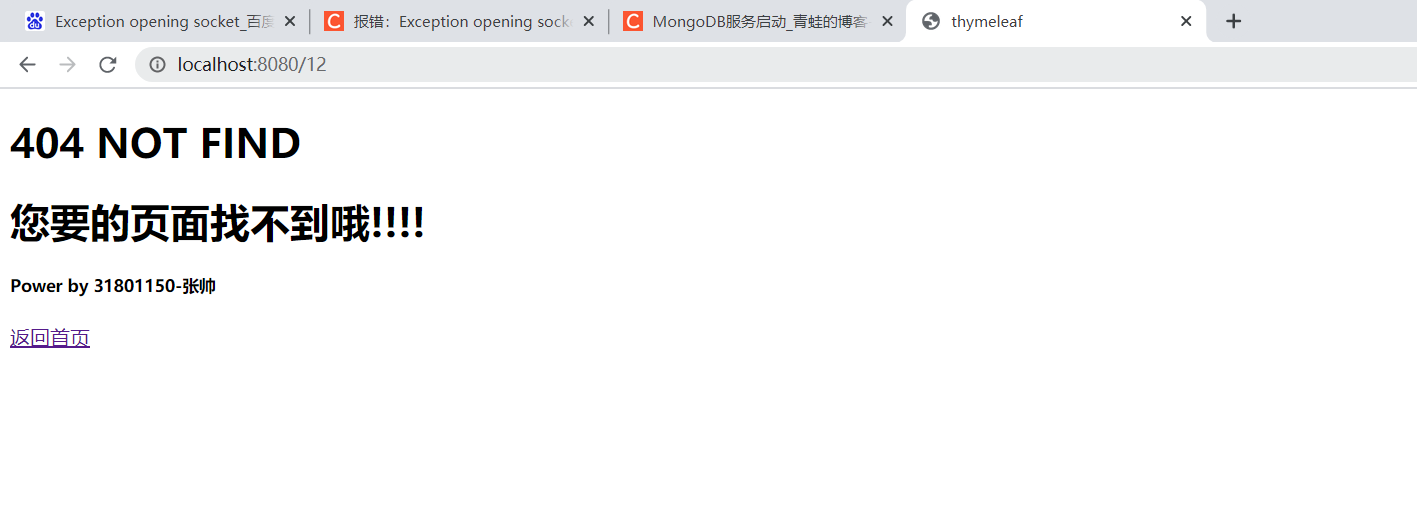


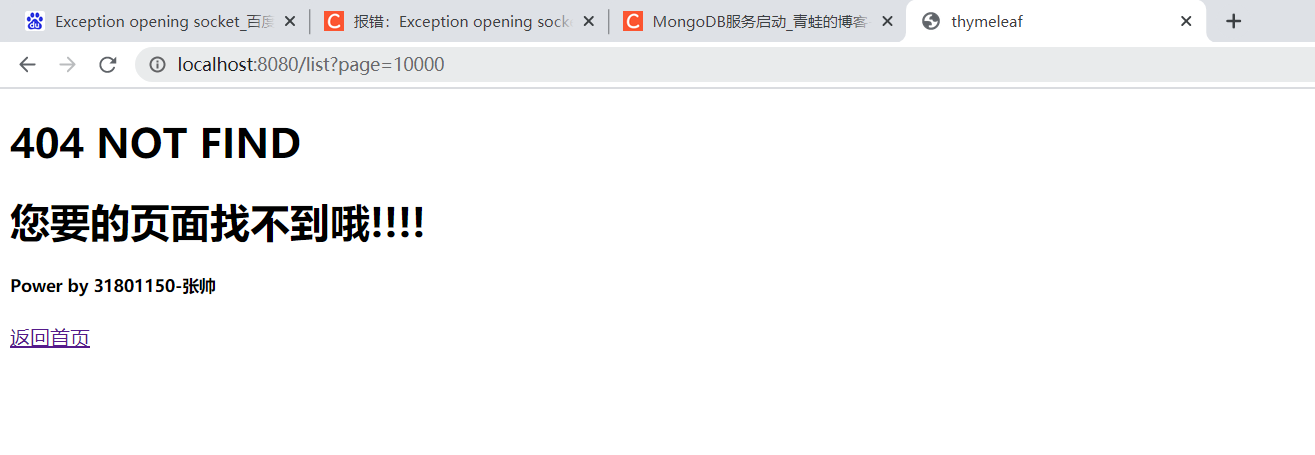
自定义错误提示：

@Controller  
@RequestMapping(value = "/error")  
public class errorController implements ErrorController {  
  
  
 @Override  
 public String getErrorPath() {  
 return null;  
 }  
  
  
 @RequestMapping  
 String error(){  
  
 return "error";  
 }  
}

<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>thymeleaf</title>  
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
</head>  
<style type="text/css"> .warn{color:red} </style>  
<body>  
  
<h1>404 NOT FIND</h1>  
<h1>您要的页面找不到哦!!!!</h1>  
<h5>Power by 31801150-张帅</h5>  
<a th:href="@{'/'}">返回首页</a>  
  
</body>

找不到页面的提示：

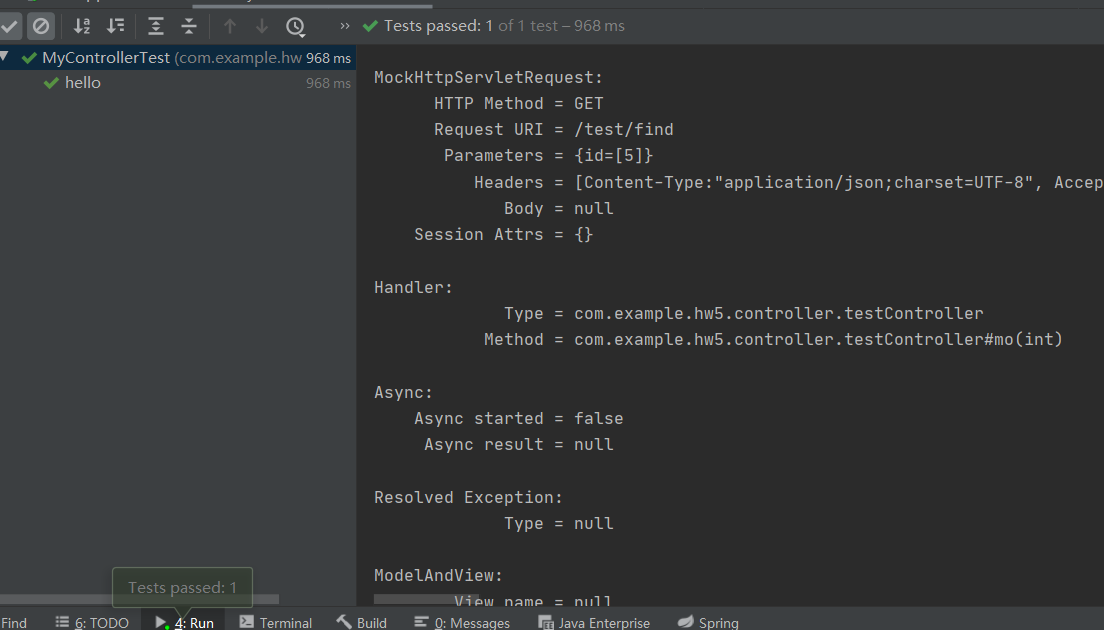


携带参数有误的提示：（page=10000参数错误，没有这么多页）  


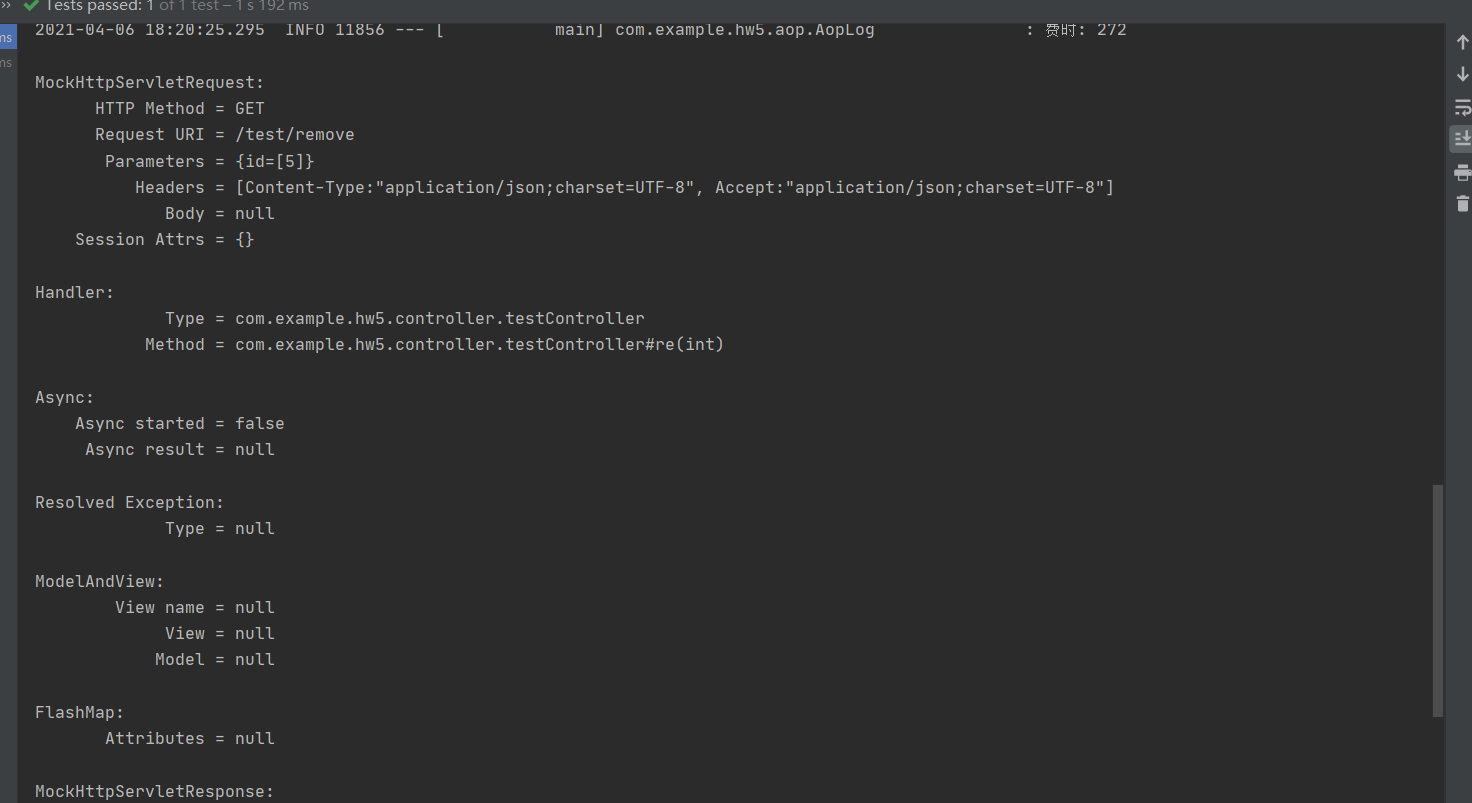
编写单元测试：

按照id查找：

@Test  
public void find() throws Exception {  
 // 得到MvcResult自定义验证  
 // 执行请求  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*get*("/test/find")  
 .contentType(MediaType.*APPLICATION\_JSON\_UTF8*)  
 //传入参数  
 .param("id", String.*valueOf*(5))  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("{\"id\":5,\"name\":\"test\",\"age\":20,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"}"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 Assert.*assertEquals*("{\"id\":5,\"name\":\"test\",\"age\":20,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"}",content);  
}

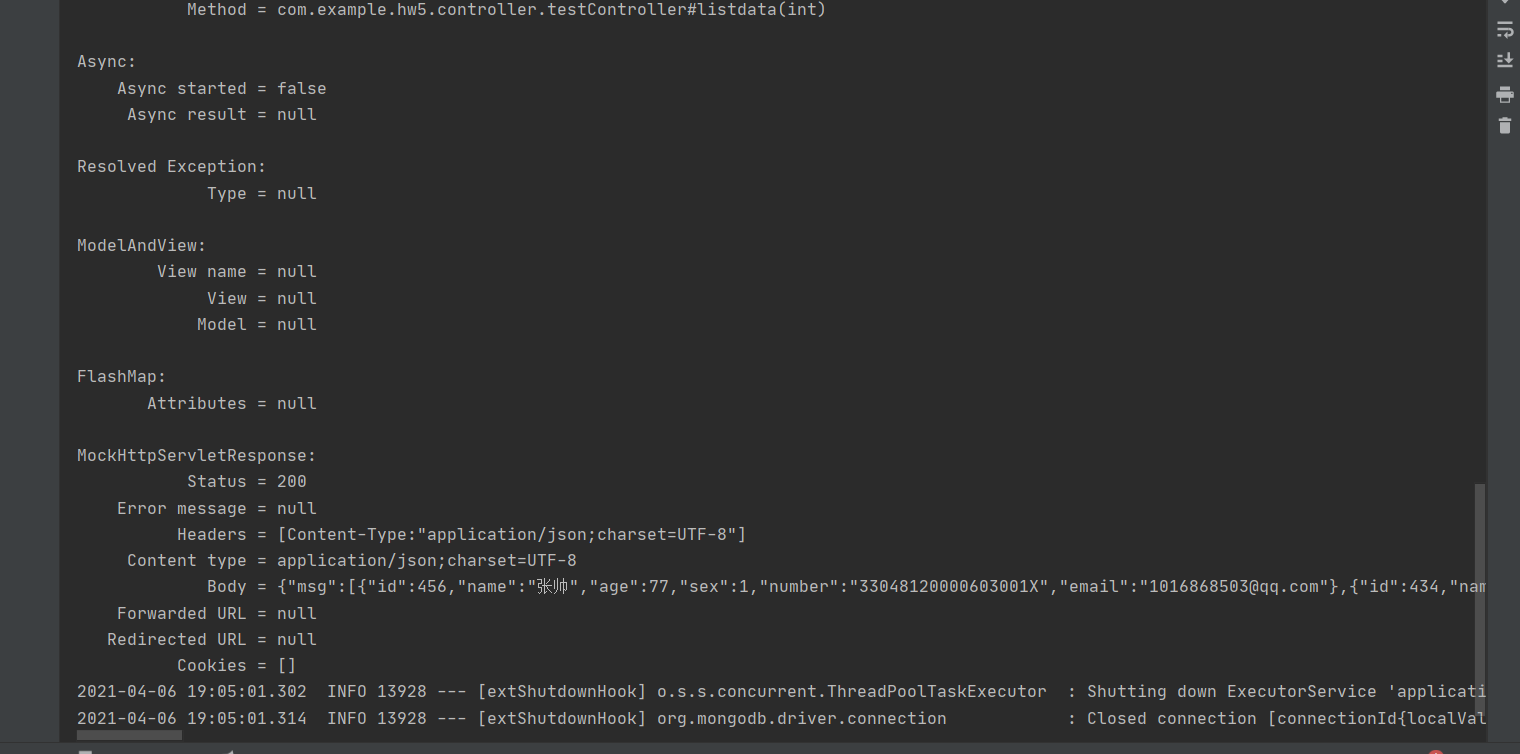
按照id删除：

@Test  
public void remove() throws Exception {  
 // 得到MvcResult自定义验证  
 // 执行请求  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*get*("/test/remove")  
 .contentType(MediaType.*APPLICATION\_JSON\_UTF8*)  
 //传入参数  
 .param("id", String.*valueOf*(5))  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("remove success id = 5"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 Assert.*assertEquals*("remove success id = 5",content);  
}



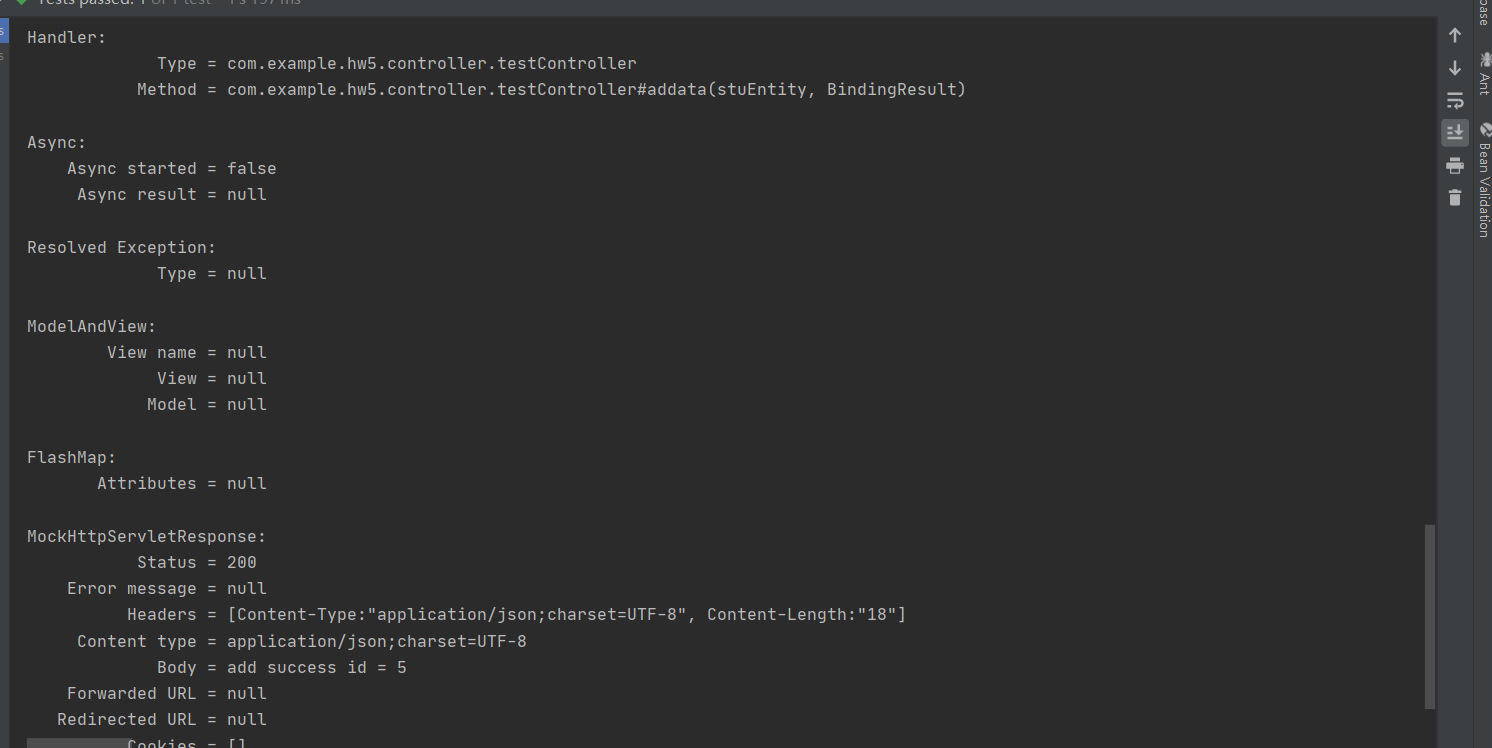
分页查询：

@Test  
public void list() throws Exception {  
 // 得到MvcResult自定义验证  
 // 执行请求  
  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*get*("/test/list")  
 .contentType(MediaType.*APPLICATION\_JSON\_UTF8*)  
 //传入参数  
 .param("page", String.*valueOf*(0))  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("{\"msg\":[{\"id\":456,\"name\":\"张帅\",\"age\":77,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"},{\"id\":434,\"name\":\"1212\",\"age\":12,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"},{\"id\":333,\"name\":\"admin\",\"age\":39,\"sex\":0,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"}],\"flag\":1,\"index\":0,\"sum\":0}"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 Assert.*assertEquals*("{\"msg\":[{\"id\":456,\"name\":\"张帅\",\"age\":77,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"},{\"id\":434,\"name\":\"1212\",\"age\":12,\"sex\":1,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"},{\"id\":333,\"name\":\"admin\",\"age\":39,\"sex\":0,\"number\":\"33048120000603001X\",\"email\":\"1016868503@qq.com\"}],\"flag\":1,\"index\":0,\"sum\":0}",content);  
}

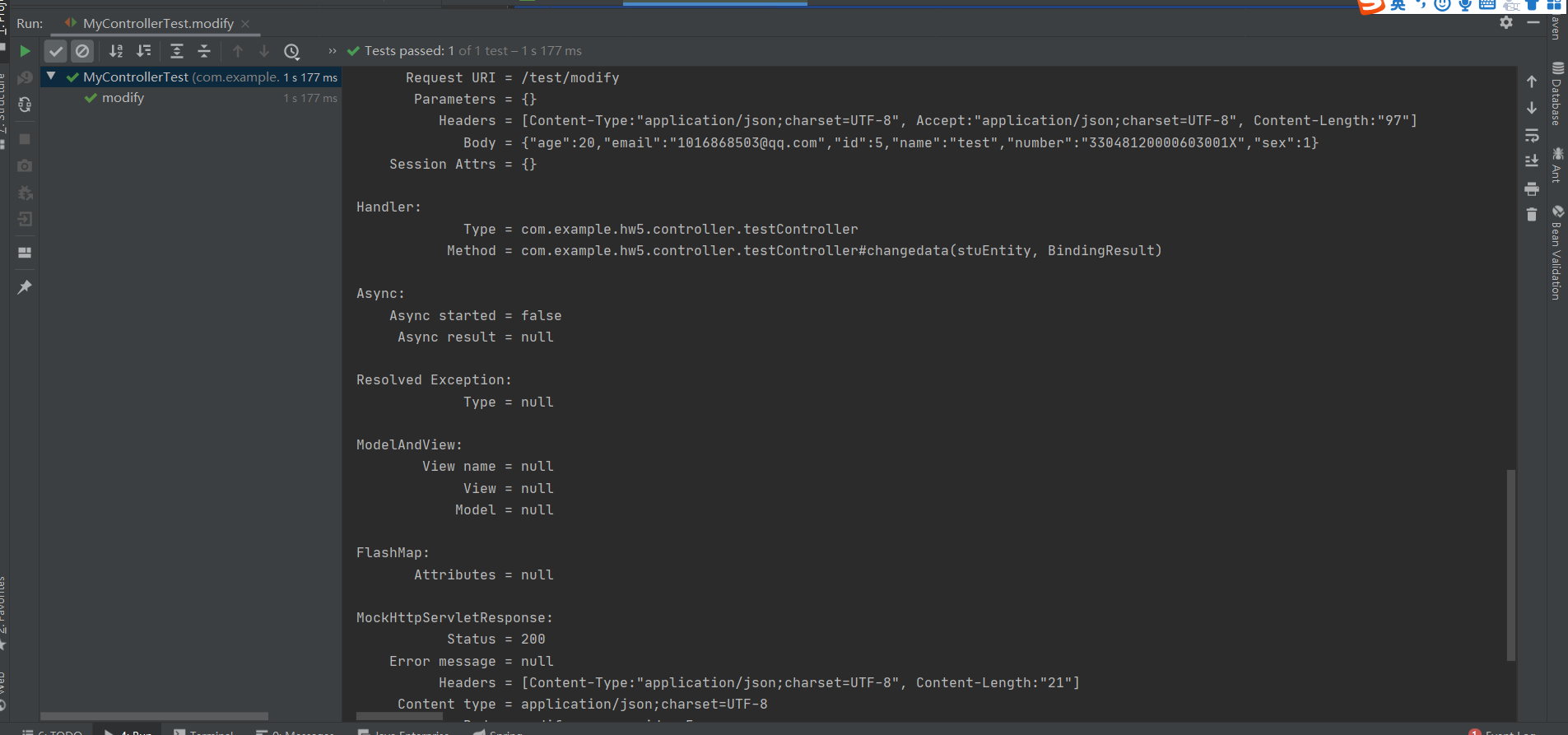


添加一个：

@Test  
public void add() throws Exception {  
  
  
 stuEntity stu =new stuEntity(5,"test",20,1,"33048120000603001X","1016868503@qq.com");  
 String requestJson = JSONObject.*toJSONString*(stu);  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*post*("/test/add")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 //传入参数  
 .content(requestJson)  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("add success id = 5"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 *assertEquals*("add success id = 5",content);  
}

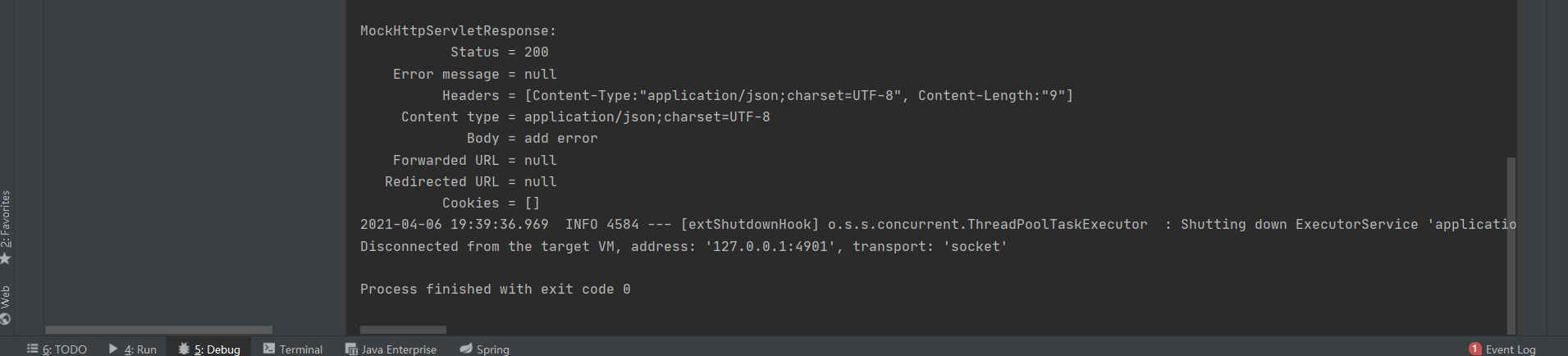


修改一个：  
@Test  
public void modify() throws Exception {  
  
  
 stuEntity stu =new stuEntity(5,"test",20,1,"33048120000603001X","1016868503@qq.com");  
 String requestJson = JSONObject.*toJSONString*(stu);  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*post*("/test/modify")  
 .contentType(MediaType.*APPLICATION\_JSON\_UTF8*)  
 //传入参数  
 .content(requestJson)  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("modify success id = 5"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 Assert.*assertEquals*("modify success id = 5",content);  
}



错误数据测试：：

@Test  
public void erroradd() throws Exception {  
  
  
 stuEntity stu =new stuEntity(1212,"test",50000,1,"33048120000603001X","1016868503@qq.com");  
 String requestJson = JSONObject.*toJSONString*(stu);  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*post*("/test/add")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 //传入参数  
 .content(requestJson)  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andExpect(MockMvcResultMatchers.*content*().string("add error"))  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
 //得到返回代码  
 int status=mvcResult.getResponse().getStatus();  
 //得到返回结果  
 String content=mvcResult.getResponse().getContentAsString();  
 //断言，判断返回代码是否正确  
 //Assert.assertEquals(200,status);  
 //断言，判断返回的值是否正确  
 *assertEquals*("add error",content);  
}



找不到页面测试：

@Test  
public void error() throws Exception {  
 MvcResult mvcResult= mockMvc.perform(MockMvcRequestBuilders.*get*("/1212")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 //传入参数  
 // .accept(MediaType.TEXT\_HTML\_VALUE))  
 //接收的类型  
 .accept(MediaType.*APPLICATION\_JSON\_UTF8*))  
 //等同于Assert.assertEquals(200,status);  
 //判断接收到的状态是否是200  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound())  
 //等同于 Assert.assertEquals("hello longzhonghua",content);  
 .andDo(MockMvcResultHandlers.*print*())  
 //返回MvcResult  
 .andReturn();  
   
   
}

