测试对象：MessageController类

测试人员：刘畅

测试日期：2020/6/23

# 一、测试对象描述

该类作为Controller层的一个类，主要功能为接受前端传来的数据，经过处理后传给MessageService和MessageVoService进行处理，得到它们返回的数据后经过处理再返回给前端，或者跳转到对应的界面。

为了方便测试，我们使用mockmvc对前端进行模拟，使用mockBean对MessageService和MessageVoService进行模拟，将mockBean作为待测对象的桩来进行测试。

该类需要测试的方法如下：

|  |  |  |
| --- | --- | --- |
| 方法标识符 | 方法名 | 测试关注点 |
| Unit\_003\_Function\_001 | String message\_list(Model model,HttpServletRequest request) | 是否能正确断言，是否能正确响应，是否能调用正确的函数，是否返回正确的界面 |
| Unit\_003\_Fuction\_002 | List<MessageVo> message\_list(@RequestParam(value = "page",defaultValue = "1")int page) | 是否能正确响应，是否能正确调用对应service层函数 |
| Unit\_003\_Fuction\_003 | List<MessageVo> user\_message\_list(@RequestParam(value = "page",defaultValue = "1")int page,HttpServletRequest request) | 是否能正确触发断言、是否能正确响应，是否能调用正确的函数 |
| Unit\_003\_Function\_004 | void sendMessage(String userID, String content, HttpServletResponse response) | 是否能调用正确的函数，是否能返回正确的界面 |
| Unit\_003\_Fuction\_005 | boolean modifyMessage(int messageID,String content, HttpServletResponse response) | 是否能调用正确的函数，是否能返回正确的值 |
| Unit\_003\_Fuction\_006 | boolean delMessage(int messageID) | 是否能调用正确的函数，是否能返回正确的值 |

# 二、测试用例设计

## 2.1、Unit\_003\_Fuction\_001

待测试的方法为MessageController类中的String message\_list(Model model,HttpServletRequest request)方法。

该方法接收到前端传来的get请求后，经过调用messageService和messageVoService获取对应的数据，然后对请求中的session进行判断，user存在即返回信息和返回message\_list界面，否则断言“请登录”。

为了保证分支覆盖，我们设计了两个测试用例分别测试user是否存在的情况。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *fail\_to\_return\_message\_list()* | *get*("/message\_list")) | *assertThrows*(NestedServletException.class,()->mockMvc.perform(*get*("/message\_list")),"请登录！");通过 |  |
| 2 | *succeed\_to\_return\_message\_list()* | *get("/message\_list").sessionAttr("user",user)* | *perform.andExpect(status().isOk()); assertModelAttributeAvailable(mv,"total"); assertModelAttributeAvailable(mv,"user\_total"); verify(messageService).findPassState(message\_pageable); verify(messageVoService).returnVo(any()); verify(messageService).findByUser(userID,user\_message\_pageable); assertEquals(mv.getViewName(),"message\_list");全部通过* |  |

## 2.2、Unit\_003\_Fuction\_002

待测试的方法为MessageController类中的List<MessageVo> message\_list(@RequestParam(value = "page",defaultValue = "1")int page)方法。 该方法接收到前端传来的get请求后，调用messageService中的函数查询对应的数据，然后使用messageVoService中的函数对数据规格进行处理，最后返回处理后的结果数据。

为了测试是否能正确响应和正确调用service层的函数，我们设计如下单个测试用例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *return\_pass\_state\_message\_list()* | *get*("/message/getMessageList") | perform.andExpect(*status*().isOk()); *verify*(messageService).findPassState(*any*()); *verify*(messageVoService).returnVo(*any*());全部通过 |  |

## 2.3、Unit\_003\_Fuction\_003

待测试的方法为MessageController类中的List<MessageVo> user\_message\_list(@RequestParam(value = "page",defaultValue = "1")int page,HttpServletRequest request)方法。

该方法接收到前端传来的get请求后，获取请求中session中的user，对user进行判断，若user为空则断言“请登录！”，否则调用messageSevice和messageVoService对应的函数获得用户的消息，进行规格处理后返回。

为了保证分支覆盖，我们设计如下两个测试用例：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *fail\_to\_return\_user\_message\_list()* | *get*("/message/findUserList")) | *assertThrows*(NestedServletException.class,()->mockMvc.perform(*get*("/message/findUserList")),"请登录！");通过 |  |
| 2 | *succeed\_to\_return\_user\_message\_list()* | *get(*"/message/findUserList")*.sessionAttr("*user",user) | perform.andExpect(status().isOk()); verify(messageVoService).returnVo(any()); verify(messageService).findByUser(userID,user\_message\_pageable);通过 |  |

用例1覆盖判断语句为真的情况

用例2覆盖判断语句为假的情况

## 2.4、Unit\_003\_Fuction\_004

待测试的方法为MessageController类中的void sendMessage(String userID, String content, HttpServletResponse response)方法。

该方法在接收到前端传来的post请求和userID、content参数后，调用MessageService中的create函数创建message，最后返回到message\_list界面。

为了测试是否正确调用create函数和返回到正确的界面，我们设计了如下单个测试用例。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *user\_add\_new\_message()* | *post*("/sendMessage").param("userID","user").param("content","this is content") | andExpect(*redirectedUrl*("/message\_list")); *verify*(messageService).create(*any*());通过 |  |

## 2.5、Unit\_003\_Fuction\_005

待测试的方法为MessageController类中的boolean modifyMessage(int messageID,String content, HttpServletResponse response)方法。

该方法在接收到前端传来的post请求和messageID、userID、content参数后，调用MessageService中的update函数修改message，最后返回true。

为了判断是否正确调用update函数和返回是否正确，我们设计如下单个测试用例。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *user\_modify\_message()* | *post*("/modifyMessage.do").param("messageID","1").param("userID","user").param("content","this is content") | andExpect(*content*().string("true")); *verify*(messageService).update(*any*());通过 |  |

## 2.6、Unit\_003\_Fuction\_006

待测试的方法为MessageController类中的boolean delMessage(int messageID)方法。

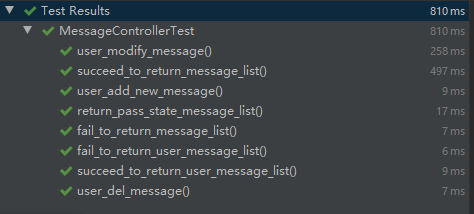
该方法在接收到前端传来的post请求和messageID参数后，调用MessageService中的delById函数删除message，最后返回true。

为判断是否正确调用delById函数和返回值是否正确，我们设计如下单个测试用例。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | *user\_del\_message()* | *post*("/delMessage.do").param("messageID","1") | perform.andExpect(*content*().string("true")); *verify*(messageService).delById(*anyInt*());通过 |  |

# 三、测试用例的执行与结果

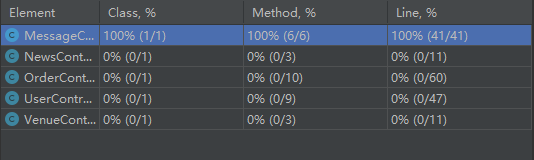
测试用例结果执行如下：



从图中可以看出，全部的8个测试用例均执行通过。

# 四、测试结果分析

上述通过的测试用例对于MessageController类的覆盖度如下：



从该图可得知，测试对于MessageController类的方法覆盖度（Method）达到100%，对其语句覆盖度（Line）也达到100%，经测试经理评审，该单元通过了单元测试。