测试对象：AdminVenueController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，主要功能为接受管理员界面前端传来的数据，经过处理后传给VenueService层进行对数据库的增删改查，得到VenueService层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VenueService进行模拟，将该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\_004\_Function\_001 | Venue findByVenueID(int id) | 是否能正确响应，调用正确的方法和返回到正确的界面 |
| Unit\_004\_Fuction\_002 | Vunue findByVenueName(String venueName) | 是否能正确响应，调用正确的方法、传递model有效的值和返回到正确的界面 |
| Unit\_004\_Fuction\_003 | Page<Venue> findAll(pageable pageable) | 是否能正确响应并返回到正确的界面 |
| Unit\_004\_Function\_004 | List<Venue> findAll(@RequestParam(value = "page",defaultValue = "1")int page) | 正确响应和调用正确的service层函数 |
| Unit\_004\_Fuction\_005 | void addVenue(String venueName, String address, String description, int price, MultipartFile picture, String open\_time, String close\_time, HttpServletRequest request, HttpServletResponse response) | 是否正确响应、是否调用正确的函数和是否返回正确的界面 |
| Unit\_004\_Fuction\_006 | void modifyVenue(int venueID,String venueName, String address, String description, int price, MultipartFile picture, String open\_time,String close\_time, HttpServletRequest request, HttpServletResponse response) | 是否正确响应、是否调用正确的函数和是否返回正确的界面 |
| Unit\_004\_Fuction\_007 | boolean delVenue(int venueID) | 是否正确响应，是否调用Service层正确的函数和是否返回正确的结果 |
| Unit\_004\_Fuction\_008 | boolean checkVenueName(String venueName) | 是否正确响应、是否调用正确的函数和是否返回正确的结果 |

# 二、测试用例设计

## 2.1、Unit\_004\_Fuction\_001

待测试的方法为AdminVenueController类中的Venue findByVenueID(int id)方法。

该方法接受前端传来的get请求后，调用venueService种findAll函数获取所有的venue信息，设定给model后返回到admin/venue\_manage界面

为了测试该方法是否能正确响应，调用正确的方法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 | return\_venue\_manage\_html() | *get*("/venue\_manage") | perform.andExpect(*status*().isOk()); *assertModelAttributeAvailable*(mv,"total"); *verify*(venueService).findAll(pageable); *assertEquals*(mv.getViewName(),"admin/venue\_manage");通过 |  |

## 2.2、Unit\_004\_Fuction\_002

待测试的方法为AdminVenueController类中的String editVenue(Model model,int venueID)方法。

该方法在接受到前端传来的get请求后，通过venueService的findByID方法查询到需要修改的venue信息，传递给model后返回至/admin/venue\_edit界面。

为了测试该方法是否能正确响应，调用正确的方法、传递model有效的值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 | return\_edit\_venue\_html() | *get*("/venue\_edit") | perform.andExpect(*status*().isOk()); *assertModelAttributeAvailable*(mv,"venue"); *verify*(venueService).findByVenueID(venueID); *assertEquals*(mv.getViewName(),"/admin/venue\_edit");通过 |  |

## 2.3、Unit\_004\_Fuction\_003

待测试的方法为AdminVenueController类中的String venue\_add()方法。

该方法在接受到前端传来的get请求后直接返回至/admin/venue\_edit界面。

为了测试该方法是否能正确响应并返回到正确的界面，我们设计了如下测试用例：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | return\_add\_venue\_html() | *get*("/venue\_add") | perform.andExpect(*status*().isOk()); *assertEquals*(mv.getViewName(),"/admin/venue\_add");通过 |  |

## 2.4、Unit\_004\_Fuction\_004

待测试的方法为AdminVenueController类中的List<Venue> getVenueList(@RequestParam(value = "page",defaultValue = "1")int page)方法。

该方法在接受到前端传来的get请求后，创建pageable变量存储信息，调用venueService种的findAll函数后返回所有的Venue信息。

为了测试该方法是否正确响应和调用正确的service层函数，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\_venue\_list() | *get*("/venueList.do").param("page","1") | perform.andExpect(*status*().isOk()); *verify*(venueService).findAll(pageable);通过 |  |

## 2.5、Unit\_004\_Fuction\_005

待测试的方法为AdminVenueController类中的void addVenue(String venueName, String address, String description, int price, MultipartFile picture, String open\_time,String close\_time,HttpServletRequest request, HttpServletResponse response)方法。

该方法在接收到前端传来的http请求后，对其picture进行判断并处理，再调用venueService种的create函数，判断是否创建成功，并根据创建结果返回到对应的界面。

为了保证分支覆盖，我们设计了4个测试用例来分别测试picture是否存在的情况和创建是否成功的情况。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | add\_venue\_success\_and\_picture\_is\_empty() | MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_manage")); *verify*(venueService).create(*any*());  通过 |  |
| 2 | add\_venue\_success\_and\_picture\_exists() | MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_manage")); *verify*(venueService).create(*any*());  通过 |  |
| 3 | add\_venue\_failed\_and\_picture\_is\_empty() | MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_add")); *verify*(venueService).create(*any*());  通过 |  |
| 4 | add\_venue\_failed\_and\_picture\_exists() | MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_add")); *verify*(venueService).create(*any*());  通过 |  |

测试用例1覆盖了条件语句一假、条件语句二真

测试用例2覆盖了条件语句一真，条件语句二真

测试用例3覆盖了条件语句一假，条件语句二假

测试用例4覆盖了条件语句一真，条件语句二假

## 2.6、Unit\_004\_Fuction\_006

待测试的方法为AdminVenueController类中的modifyVenue(int venueID,String venueName, String address, String description, int price, MultipartFile picture, String open\_time,String close\_time,HttpServletRequest request, HttpServletResponse response)方法。

该方法在接受前端http请求后，判断请求中的picture是否存在并对其进行处理，调用venueService中的findByVenueID函数和update函数对Venue信息进行处理后返回到venue\_manage界面。

为了实现分支覆盖，我们设计了两个测试用例分别测试picture是否为空的情况，每个测试用例均需测试是否正确响应、是否调用正确的函数和是否返回正确的界面。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | modify\_venue\_and\_picture\_is\_empty() | MockMultipartFile mockMultipartFile = new MockMultipartFile("picture","","picture", "".getBytes());  MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_manage")); *verify*(venueService).findByVenueID(*anyInt*()); *verify*(venueService).update(*any*());  通过 |  |
| 2 | modify\_venue\_and\_picture\_exists() | MockMultipartFile mockMultipartFile = new MockMultipartFile("picture","1.bmp","picture", "1.bmp".getBytes());  MockMvcRequestBuilders.*multipart*("/addVenue.do").file(mockMultipartFile).param("venueName","venue").param("address","this is address").param("description","this is description").param("price","100").param("open\_time","08:00").param("close\_time","18:00"); | perform.andExpect(*redirectedUrl*("venue\_manage")); *verify*(venueService).findByVenueID(*anyInt*()); *verify*(venueService).update(*any*());  通过 |  |

## 2.7、Unit\_004\_Fuction\_007

待测试的方法为AdminVenueController类中的boolean delVenue(int venueID)方法。

该方法在获取前端传来的post请求后，根据请求中的venueID，使用venueService中的delByID函数，返回true。

为了测试该方法是否正确响应，是否调用Service层正确的函数和是否返回正确的结果，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 | admin\_del\_venue() | *post*("/delVenue.do").param("venueID","1") | perform.andExpect(*content*().string("true")); perform.andExpect(*status*().isOk()); *verify*(venueService).delById(*anyInt*());通过 |  |

## 2.8、Unit\_004\_Fuction\_008

待测试的方法为AdminVenueController类中的boolean checkVenueName(String venueName)方法。

该方法在获取前端传来的post请求后，根据请求中的venueName，使用venueService中的countVenueName函数，根据结果返回true/false。

为了实现分支覆盖，我们设计了两个测试用例分别测试返回结果为true或者false的情况，每个测试用例均需测试是否正确响应、是否调用正确的函数和是否返回正确的结果。

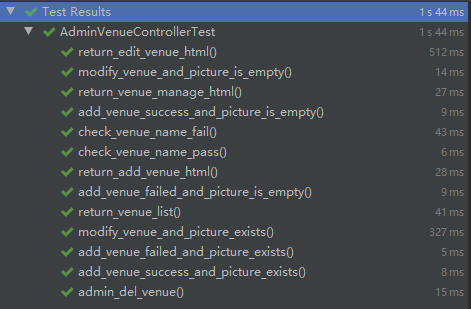
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 驱动函数 | 输入变量 | 预期结果 | 执行结果 |
| 1 | check\_venue\_name\_pass() | *post*("/checkVenueName.do").param("venueName","venue") | andExpect(*status*().isOk())  andExpect(*content*().string("true"))  *verify*(venueService).countVenueName("venue")通过 |  |
| 2 | check\_venue\_name\_fail() | *post*("/checkVenueName.do").param("venueName","venue") | andExpect(*status*().isOk())  andExpect(*content*().string("false"))  *verify*(venueService).countVenueName("venue")通过 |  |

测试用例1覆盖了结果为true的情况

测试用例2覆盖了结果为false的情况

# 三、测试用例的执行与结果

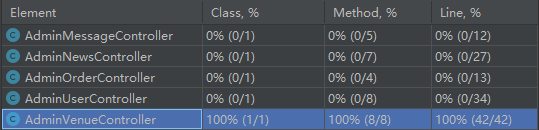
测试用例结果执行如下：



从图中可以看出，全部的13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AdminVenueController类的覆盖度如下：



从该图可得知，测试对于AdminVenueController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。