对于订单模块的详细测试

1. 简介

在本次详细的测试设计中，我们选取了“订单管理”模块作为测试对象。我们测试的接口及场景如下：

1. 添加订单
2. 修改订单
3. 获取用户订单列表
4. 获取场馆指定日期的订单

测试内容涵盖了接口的基本功能、用户权限验证（如登录状态检查）、参数处理逻辑、页面重定向行为及接口返回值的正确性等方面，确保订单模块的核心功能在实际业务场景中的稳定性与可靠性。

2. 黑盒测试

2.1 等价类和边界类

2.1.1 添加订单

等价类划分

该模块的输入参数如下所示。

Parameter	Data Type	Description	Requirement
venueName	String	需要预定的场馆的场馆名字	目前只有六个，分别是`"2222"`，`"场馆2"`，`"场馆3"`，`"场馆4"`，`"场馆5"`，`"场馆6"`，只能从中选择。
startTime	String	预定的开始时间	必须是整点，格式为"yyyy-mm-dd hh:mm"，其中分钟数必须是0。开始时间不能晚于9点，结束时间不能早于19点。
hours	int	预定的时长	场馆20:00点关门，因此开始时间加预定时间不能晚于20点整。

request	HttpServletRequest	有效用户session，内含用户的信息。	必须含有有效用户信息，以Cookie的形式传入。

等价类

Parameter	Data Type	VALID	INVALID
venueName	String	`"2222"`, `"场馆2"`, `"场馆3"`, `"场馆4"`, `"场馆5"`, `"场馆6"` (1)	`""` (2) `null` (3) `"场馆1"` (4) `"其他场馆"` (5)
startTime	String	^\d{4}-\d{2}-\d{2} (09|1[0-8]):00$ (6)	`null` (7) `""` (8) `"2024/01/01 12:00"` (9) `"2024-01-01 08:00"` (10) `"2024-01-01 20:00"` (11) `"2024-01-01 12:30"` (12)
hours	int	^([1-9]|1[0-1])$ (13)	`0` (14) `-1` (15) `25` (16) `999` (17)
request	HttpServletRequest	含有效用户session(18)	`null` (19) `"wrong_session"` (20)

响应

- **正常情况下：**
  - **HTTP状态码**: 302 (重定向)
  - **响应类型**: 重定向响应
  - **重定向目标**: order_manage 页面
  - **响应体**: 空 (因为返回类型是 `void`)
- **异常情况下：**

- **HTTP状态码**: 500 (重定向)
- **响应类型**: 错误响应
- **响应体**: 返回Spring默认的错误页面或JSON错误信息

边界类划分

等价类	参数	边界值	描述
(1)	venueName	`"2222"`	数字类型场馆名
(1)	venueName	`"场馆2"`	最小编号场馆
(1)	venueName	`"场馆6"`	最大编号场馆
(2)	venueName	`""`	空字符串边界
(3)	venueName	`null`	空值边界
(4)	venueName	`"场馆1"`	相似但无效值
(5)	venueName	`"场馆7"`	超出范围编号
(5)	venueName	`"其他场馆"`	完全不同值
(6)	startTime	`"2024-01-01 09:00"`	最早营业时间边界
(6)	startTime	`"2024-01-01 19:00"`	最晚营业时间边界
(7)	startTime	`null`	空值边界
(8)	startTime	`""`	空字符串边界
(9)	startTime	`"2024/01/01 12:00"`	错误格式边界
(10)	startTime	`"2024-01-01 08:00"`	营业时间下界-1
(11)	startTime	`"2024-01-01 20:00"`	营业时间上界+1
(12)	startTime	`"2024-01-01 12:30"`	分钟非零边界
(13)	hours	`1`	最小有效小时数

(13)	hours	`24`	最大有效小时数
(14)	hours	`0`	小时数下界-1
(15)	hours	`-1`	负数边界
(16)	hours	`12`	小时数上界+1
(17)	hours	`999`	极大无效值
(18)	request	有效session对象	包含用户信息的session
(19)	request	`null`	空请求对象
(20)	request	空session对象	不包含用户信息的session

测试用例设计

TCs	Covered EP/BV	Input	Output
1	(1), (6), (13), (18)	`{"venueName": "2222", "startTime": "2024-06-15 14:00", "hours": 8, "request": "valid_session"}`	y
2	(1), (6), (13), (18)	`{"venueName": "场馆3", "startTime": "2024-08-20 09:00", "hours": 1, "request": "valid_session"}`	y
3	(1), (6), (13)	`{"venueName": "2222", "startTime": "2024-01-01 09:00", "hours": 1, "request": "valid_session"}`	y
4	(1), (6), (13)	`{"venueName": "场馆2", "startTime": "2024-01-01 19:00", "hours": 24, "request": "valid_session"}`	y
5	(1)	`{"venueName": "场馆6", "startTime": "2024-06-15 12:00", "hours": 8, "request": "valid_session"}`	y
6	(2)	`{"venueName": "", "startTime": "2024-06-15 14:00", "hours": 8, "request": "valid_session"}`	n
7	(3)	`{"venueName": null, "startTime": "2024-06-15 14:00", "hours": 8, "request": "valid_session"}`	n
8	(4)	`{"venueName": "场馆1", "startTime": "2024-06-15 14:00", "hours": 8, "request": "valid_session"}`	n
9	(5)	`{"venueName": "场馆7", "startTime": "2024-06-15 14:00", "hours": 8, "request": "valid_session"}`	n

10	(7)	`{"venueName": "2222", "startTime": null, "hours": 8, "request": "valid_session"}`	n
11	(8)	`{"venueName": "2222", "startTime": "", "hours": 8, "request": "valid_session"}`	n
12	(9)	`{"venueName": "2222", "startTime": "2024/01/01 12:00", "hours": 8, "request": "valid_session"}`	n
13	(9)	`{"venueName": "2222", "startTime": "2024-13-01 12:00", "hours": 8, "request": "valid_session"}`	n
14	(9)	`{"venueName": "2222", "startTime": "2024-01-32 12:00", "hours": 8, "request": "valid_session"}`	n
15	(10)	`{"venueName": "2222", "startTime": "2024-06-15 08:00", "hours": 8, "request": "valid_session"}`	n
16	(11)	`{"venueName": "2222", "startTime": "2024-06-15 20:00", "hours": 8, "request": "valid_session"}`	n
17	(12)	`{"venueName": "2222", "startTime": "2024-06-15 12:30", "hours": 8, "request": "valid_session"}`	n
18	(14)	`{"venueName": "2222", "startTime": "2024-06-15 14:00", "hours": 0, "request": "valid_session"}`	n
19	(16)	`{"venueName": "2222", "startTime": "2024-06-15 14:00", "hours": 25, "request": "valid_session"}`	n
20	(19)	`{"venueName": "2222", "startTime": "2024-06-15 14:00", "hours": 8, "request": null}`	n
21	(20)	`{"venueName": "2222", "startTime": "2024-06-15 14:00", "hours": 8, "request": "empty_session"}`	n

黑盒测试结果

经排查，问题分别是15 16 17 18 19。

2.1.2 修改订单

等价类划分

该模块的输入参数如下所示。

Parameter	Data Type	Description	Requirement
venueName	String	需要预定的场馆的场馆名字	目前只有六个，分别是 `"2222"`，`"场馆2"`，`"场馆3"`，`"场馆4"`，`"场馆5"`，`"场馆6"`，只能从中选择。
startTime	String	预定的开始时间	必须是整点，格式为"yyyy-mm-dd hh:mm"，其中分钟数必须是0。开始时间不能晚于9点，结束时间不能早于19点。
hours	int	预定的时长	场馆20:00点关门，因此开始时间加预定时间不能晚于20点整。
orderID	int	订单ID	订单的唯一标识
request	HttpServletRequest	有效用户session，内含用户的信息。	必须含有有效用户信息，以Cookie的形式传入。

等价类

Parameter	Data Type	VALID	INVALID
venueName	String	`"2222"`, `"场馆2"`, `"场馆3"`, `"场馆4"`, `"场馆5"`, `"场馆6"` (1)	`""` (2) `null` (3) `"场馆1"` (4) `"其他场馆"` (5)
startTime	String	^\d{4}-\d{2}-\d{2} (09|1[0-8]):00$ (6)	`null` (7) `""` (8) `"2025-06-04 20:00"` (9) `"2025-06-05 08:00"` (10) `"2025-06-05 20:00"` (11) `"2025-06-05 12:30"` (12)

hours	int	^([1-9]|1[0-1])$ (13)	`0` (14) `-1` (15) `12` (16) `999` (17)
orderID	int	有效的ID (18)	无效的ID (19) `-1` (20) `null` (21)
request	HttpServletRequest	含有效用户 session(22)	`null` (23) `"wrong_session"` (24)

响应

- **正常情况下：**
  - **HTTP状态码**: 302 (重定向)
  - **响应类型**: 重定向响应
  - **重定向目标**: order_edit 页面
  - **响应体**: 空 (因为返回类型是 `void`)
- **异常情况下：**
  - **HTTP状态码**: 500 (重定向)
  - **响应类型**: 错误响应
  - **响应体**: 返回Spring默认的JSON错误信息

边界类划分

等价类	参数	边界值	描述
(1)	venueName	`"2222"`	数字类型场馆名
(1)	venueName	`"场馆2"`	最小编号场馆
(1)	venueName	`"场馆6"`	最大编号场馆
(2)	venueName	`""`	空字符串边界
(3)	venueName	`null`	空值边界
(4)	venueName	`"场馆1"`	相似但无效值
(5)	venueName	`"场馆7"`	超出范围编号
(6)	startTime	`"2025-06-05 09:00"`	最早营业时间边界

(6)	startTime	`"2025-06-05 19:00"`	最晚营业时间边界
(7)	startTime	`null`	空值边界
(8)	startTime	`""`	空字符串边界
(9)	startTime	`"2025-06-04 20:00"`	过去时间边界
(10)	startTime	`"2025-06-05 08:00"`	营业时间下界-1
(11)	startTime	`"2025-06-05 20:00"`	营业时间上界+1
(12)	startTime	`"2025-06-05 12:30"`	分钟非零边界
(13)	hours	`1`	最小有效小时数
(13)	hours	`11`	最大有效小时数(不超营业时间)
(14)	hours	`0`	小时数下界-1
(15)	hours	`-1`	负数边界
(16)	hours	`12`	超过营业时间边界
(17)	hours	`999`	极大无效值
(18)	orderID	`1`	最小有效ID
(18)	orderID	`2147483647`	最大有效ID
(19)	orderID	`0`	ID下界-1
(20)	orderID	`-1`	负数ID边界
(21)	orderID	`null`	空ID边界
(22)	request	有效session对象	包含用户信息的session
(23)	request	`null`	空请求对象
(24)	request	空session对象	不包含用户信息的session

测试用例设计

TCs	Covered EP/BV	Input	Output
22	(1), (6), (13), (18), (22)	`{"venueName": "场馆4", "startTime": "2025-06-10 12:00", "hours": 8, "orderID": 49, "request": "valid_session"}`	y
23	(1), (6), (13), (18)	`{"venueName": "2222", "startTime": "2025-06-05 09:00", "hours": 1, "orderID": 1, "request": "valid_session"}`	n
24	(1), (6), (13), (18)	`{"venueName": "场馆6", "startTime": "2025-06-06 19:00", "hours": 1, "orderID": 2147483647, "request": "valid_session"}`	n
25	(13)	`{"venueName": "场馆4", "startTime": "2025-06-10 09:00", "hours": 11, "orderID": 49, "request": "valid_session"}`	y
26	(2)	`{"venueName": "", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": 100, "request": "valid_session"}`	n
27	(4)	`{"venueName": "场馆1", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": 100, "request": "valid_session"}`	n
28	(7)	`{"venueName": "场馆4", "startTime": null, "hours": 4, "orderID": 100, "request": "valid_session"}`	n
29	(9)	`{"venueName": "场馆4", "startTime": "2025-06-04 21:00", "hours": 4, "orderID": 100, "request": "valid_session"}`	n
30	(10)	`{"venueName": "场馆4", "startTime": "2025-06-10 08:00", "hours": 4, "orderID": 100, "request": "valid_session"}`	n
31	(11)	`{"venueName": "场馆4", "startTime": "2025-06-10 20:00", "hours": 1,	n

		`"orderID": 100, "request": "valid_session"}`	
32	(12)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:30", "hours": 3, "orderID": 100, "request": "valid_session"}`	n
33	(14)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:00", "hours": 0, "orderID": 100, "request": "valid_session"}`	n
34	(16)	`{"venueName": "场馆4", "startTime": "2025-06-10 09:00", "hours": 12, "orderID": 100, "request": "valid_session"}`	n
35	(16)	`{"venueName": "场馆4", "startTime": "2025-06-10 18:00", "hours": 3, "orderID": 100, "request": "valid_session"}`	n
36	(19)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": 0, "request": "valid_session"}`	y
37	(20)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": -1, "request": "valid_session"}`	n
38	(23)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": 100, "request": null}`	n
39	(24)	`{"venueName": "场馆4", "startTime": "2025-06-10 16:00", "hours": 4, "orderID": 100, "request": "empty_session"}`	n

黑盒测试结果

同理。经排查，测试错误的内容为30 31 32 34 35。

2.1.3 获取用户订单列表

等价类划分

该模块的输入参数如下所示。

Parameter	Data Type	Description	Requirement
venueName	String	需要预定的场馆的场馆名字	目前只有六个，分别是 `"2222"`，`"场馆2"`，`"场馆3"`，`"场馆4"`，`"场馆5"`，`"场馆6"`，只能从中选择。
date	String	需要查看的用户预定情况的日期	格式为"yyyy-mm-dd"

等价类

Parameter	Data Type	VALID	INVALID
venueName	String	`"2222"`，`"场馆2"`，`"场馆3"`，`"场馆4"`，`"场馆5"`，`"场馆6"` (1)	`""` (2) `null` (3) `"场馆1"` (4) `"其他场馆"` (5)
date	String	^\\d{4}-\\d{2}-\\d{2}$ (5)	`""` (6) `null` (7) `"2024/01/01"` (8) `"13-01-2024"` (9) `"2024-02-30"` (10)

响应

- **正常情况下：**
  - **HTTP状态码**: 200
  - **响应类型**: 正确响应

- **响应体**: 包含场馆信息和预定信息的JSON

- **异常情况下：**
  - **HTTP状态码**: 500 (重定向)
  - **响应类型**: 错误响应
  - **响应体**: 返回Spring默认的错误页面或JSON错误信息

边界类划分

等价类	参数	边界值	描述
(1)	venueName	`"2222"`	数字类型场馆名
(1)	venueName	`"场馆2"`	最小编号场馆
(1)	venueName	`"场馆6"`	最大编号场馆
(2)	venueName	`""`	空字符串边界
(3)	venueName	`null`	空值边界
(4)	venueName	`"场馆1"`	相似但无效值
(5)	venueName	`"场馆7"`	超出范围编号
(5)	date	`"1900-01-01"`	最早有效日期
(5)	date	`"9999-12-31"`	最晚有效日期
(6)	date	`""`	空字符串边界
(7)	date	`null`	空值边界
(8)	date	`"2024/01/01"`	错误格式边界
(9)	date	`"13-01-2024"`	错误格式边界
(10)	date	`"2024-02-30"`	无效日期边界
(10)	date	`"2024-00-01"`	月份下界无效
(10)	date	`"2024-13-01"`	月份上界无效
(10)	date	`"2024-01-00"`	日期下界无效
(10)	date	`"2024-01-32"`	日期上界无效

测试用例设计

TCs	Covered EP/BV	Input	Output
37	(1), (5)	`{"venueName": "场馆5", "date": "2024-09-15"}`	y
38	(1), (5)	`{"venueName": "2222", "date": "1900-01-01"}`	y
39	(1), (5)	`{"venueName": "场馆2", "date": "9999-12-31"}`	y
40	(1)	`{"venueName": "场馆6", "date": "2024-09-15"}`	y
41	(2)	`{"venueName": "", "date": "2024-09-15"}`	n
42	(4)	`{"venueName": "场馆1", "date": "2024-09-15"}`	n
43	(5)	`{"venueName": "场馆7", "date": "2024-09-15"}`	n
44	(6)	`{"venueName": "场馆5", "date": ""}`	n
45	(7)	`{"venueName": "场馆5", "date": null}`	n
46	(10)	`{"venueName": "场馆5", "date": "2024-00-01"}`	n
47	(10)	`{"venueName": "场馆5", "date": "2024-13-01"}`	n
48	(10)	`{"venueName": "场馆5", "date": "2024-01-32"}`	n
49	(10)	`{"venueName": "场馆5", "date": "2024-02-`	n

		`30"}`	

黑盒测试结果

错误的测试用例序号为49

2.1.4 获取场馆预订日期的订单

等价类划分

该模块的输入参数如下所示。

Parameter	Data Type	Description	Requirement
page	int	需要查询的页数	需要正确有效的页数
request	HttpServletRequest	有效用户session，内含用户的信息。	必须含有有效用户信息，以Cookie的形式传入。

等价类

Parameter	Data Type	VALID	INVALID
page	int	`1`, `2`, `10`, `100` (1)	`0` (2) `-1` (3) `null` (4)

request	HttpServletRequest	含有效用户 session (5)	`null` (6) `"empty_session"` (7)

响应

- **正常情况下：**
  - **HTTP状态码**: 200
  - **响应类型**: 正确响应
  - **响应体**: 包含订单信息的JSON List
- **异常情况下：**
  - **HTTP状态码**: 500 (重定向)
  - **响应类型**: 错误响应
  - **响应体**: 返回Spring默认的错误页面或JSON错误信息

边界类划分

等价类	参数	边界值	描述
(1)	page	`1`	最小有效页码
(1)	page	`2147483647`	最大有效页码
(2)	page	`0`	页码下界-1
(3)	page	`-1`	负数页码边界
(4)	page	`null`	空页码边界
(5)	request	有效session对象	包含用户信息的session
(6)	request	`null`	空请求对象
(7)	request	空session对象	不包含用户信息的session

测试用例设计

TCs	Covered EP/BV	Input	Output
50	(1), (5)	`{"page": 5, "request": "valid_session"}`	y

51	(1)	`{"page": 1, "request": "valid_session"}`	y
52	(1)	`{"page": 2147483647, "request": "valid_session"}`	y
53	(2)	`{"page": 0, "request": "valid_session"}`	n
54	(3)	`{"page": -1, "request": "valid_session"}`	n
55	(4)	`{"page": null, "request": "valid_session"}`	y
56	(6)	`{"page": 5, "request": null}`	n
57	(7)	`{"page": 5, "request": "empty_session"}`	n

黑盒测试结果

所有用例均通过。

2.2 决策表

2.2.1 场景

当用户需要修改自己的预约订单时，他需要输入一些信息，为了更清楚地解释决策表的使用，我们决定使用以测试场景“用户修改预约订单”为例进行详细讲解

网页前端发往后端信息：

2.2.2 条件组合表

我们构建了初始条件表，其中有有效或无效，条件组合共有64条，接下来我们整合条件组合结果相同，最终决策表如下：

Decision Table								
Modify Order								
Condition	1	2	3	4	5	6	7	8
orderID Valid? (Exists in DB & Belongs to User)	T	T	T	T	F	F	F	F
orderID Provided? (Not Null)	T	T	T	T	T	T	F	F
date Valid? (Format, Future & Available)	T	T	F	--	T	F	T	F
hours Valid? (Positive Integer & Within Range)	T	F	T	--	T	F	T	F
startTime Valid? (Format & Available)	T	T	F	--	T	F	T	F
venueName Valid? (Exists in DB & Available)	T	T	F	--	T	F	T	F
Action								
Error Code	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Message	修改成功	hours 无效或超出范围	date 或 startTime 无效	缺少修改参数	订单ID不存在或无权限	订单ID不存在或无权限	订单ID缺失	订单ID缺失
Modify Successfully	T	F	F	F	F	F	F	F

2.2.3 使用apifox测试结果

依据上面决策表分析出的用例，借助apifox进行测试的结果如下

经排查第三个条件组合未通过

3. 白盒测试

3.1 测试目标

本节专门介绍对订单预约模块中与后端服务相对应的一个类及其内部方法进行白盒测试的过程和结果。

由于白盒测试基于代码结构和内部逻辑，因此通过检查测试用例是否全面覆盖代码来评估测试的质量和完整性。测试的目标是实现几乎完全的类覆盖、语句覆盖、条件覆盖和甚至语句覆盖。

我们使用CodeCov和Jacoco来监控代码覆盖率的变化。

3.2 测试用例

3.2.1 分析代码结构

通过人工分析每个方法的代码，可以识别出潜在的分支条件和关键语句，对后续的测试设计具有指导性。

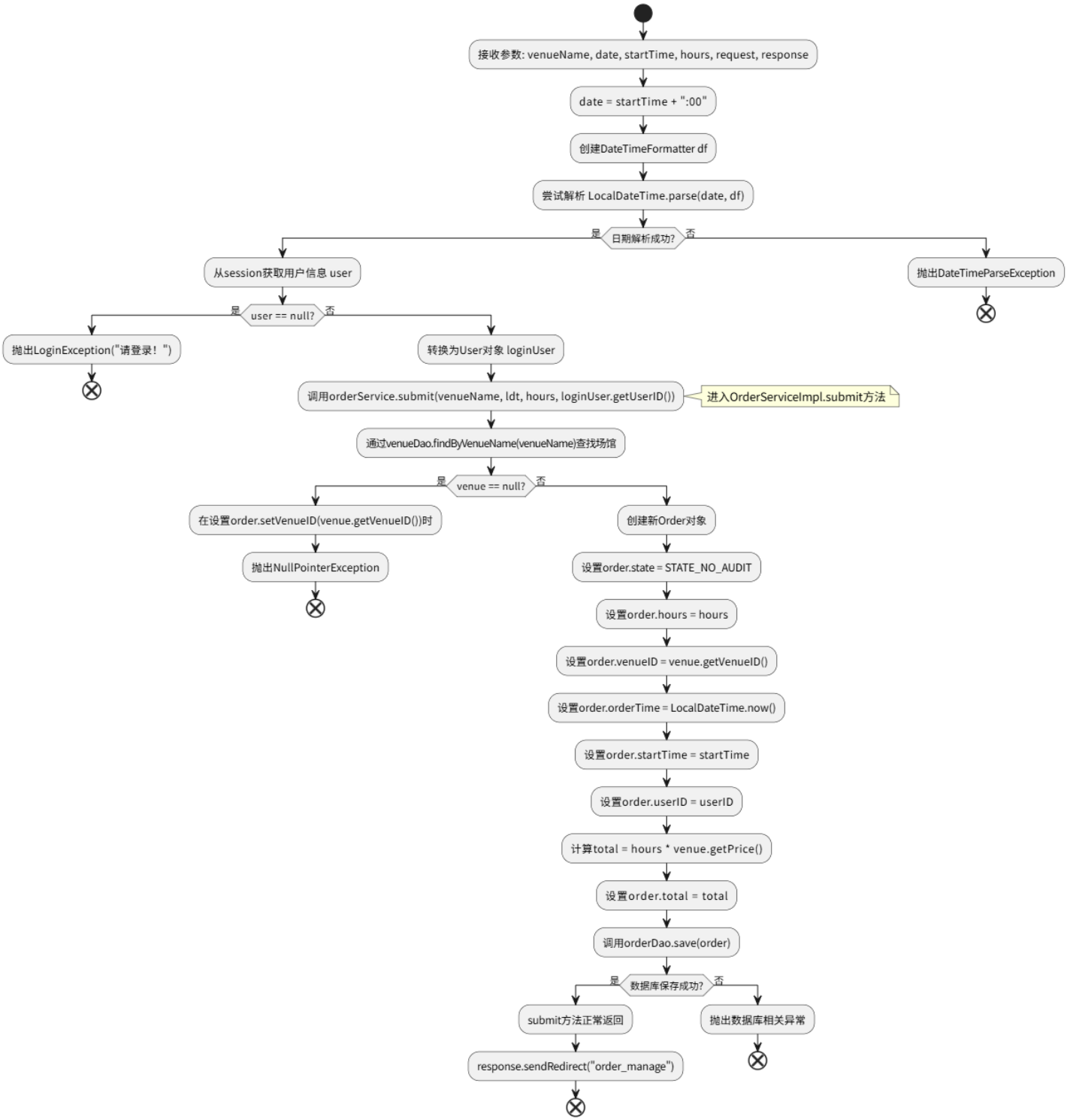
方法表

函数名	描述	测试用例是否覆盖	函数内是否有分支
order_manage	显示用户订单管理页面，获取用户订单列表并分页	yes	yes

	显示		
order_place(Model model, int venueID)	根据场馆ID显示订单预订页面	yes	no
order_place(Model model)	显示订单预订页面（无参数版本）	yes	no
order_list	获取用户订单列表（Ajax请求），支持分页	yes	yes
addOrder	添加新订单，处理预订信息并重定向	yes	yes
finishOrder	完成订单状态更新	yes	no
editOrder	显示订单编辑页面，获取订单和场馆信息	yes	no
modifyOrder	修改订单信息，更新订单详情	yes	yes
delOrder	删除指定订单	yes	no
getOrder	获取指定场馆和日期的订单信息	yes	yes

流程图

共有10种方法需要测试。我仅展示其中一个分支较多的示例流程图。



我们可以看到该函数对每个参数都进行了合法性验证，并考虑了外键被攻破的潜在问题。并且，我们可以发现它的布局比较宽泛，只要参数中有一个无效，函数就会停止，所以组合无效的情况不会让我们太纠结。

3.2.2 测试用例

我们通过方法表和流程图对代码结构进行分析后，构造了如下28个功能性测试的用例：

测试套件ID	#1-1
目标	测试用户未登录时访问订单管理页面的异常处理

要测试的特性	GET /order_manage 请求，Session中无user属性，应抛出LoginException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	4.2, 5.1

测试套件ID	#1-2
目标	测试用户登录后成功访问订单管理页面
要测试的特性	GET /order_manage 请求，Session中包含user对象，返回订单管理页面并分页显示订单
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.6, 3.3

测试套件ID	#1-3
目标	测试用户无订单时的订单管理页面显示
要测试的特性	GET /order_manage 请求，用户已登录但无任何订单，返回空订单列表
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	3.1, 3.3

测试套件ID	#1-4
目标	测试通过场馆详情页面访问订单下单页面
要测试的特性	GET /order_place.do?venueID=1，返回特定场馆的订单下单页面
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.7

测试套件ID	#1-5
目标	测试访问不存在场馆的订单下单页面
要测试的特性	GET /order_place.do?venueID=999，场馆不存在时应抛出异常
测试工具	MockMvc, JUnit 5, Mockito

风险覆盖	1.7, 5.1

测试套件ID	#1-6
目标	测试从顶部导航访问订单下单页面
要测试的特性	GET /order_place，正常返回订单下单页面
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	3.1

测试套件ID	#1-7
目标	测试未登录用户获取订单列表
要测试的特性	GET /getOrderList.do，Session中无user属性，应抛出LoginException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	4.2, 5.1

测试套件ID	#1-8
目标	测试已登录用户获取订单列表
要测试的特性	GET /getOrderList.do?page=1，Session中包含user对象，返回JSON格式的订单列表
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.6, 3.3

测试套件ID	#1-9
目标	测试未登录用户添加订单
要测试的特性	POST /addOrder.do，Session中无user属性，应抛出LoginException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	4.2, 5.1

测试套件ID	#1-10
目标	测试已登录用户成功添加订单
要测试的特性	POST /addOrder.do，参数：venueName="Test Venue", date="2025-12-22", startTime="2025-12-22 11:00", hours="1"
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.4, 1.5, 5.5

测试套件ID	#1-11
目标	测试添加订单时预订小时数边界值验证
要测试的特性	POST /addOrder.do，hours参数为"0"和"-1"，应抛出InvalidParameterException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	3.2, 5.1

测试套件ID	#1-12
目标	测试添加订单时日期时间格式错误处理
要测试的特性	POST /addOrder.do，date参数为"invalid-date"，应抛出DateTimeParseException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	3.2, 5.1

测试套件ID	#1-13
目标	测试添加订单时服务层异常处理
要测试的特性	POST /addOrder.do，Service层抛出RuntimeException("场馆已被预订或服务异常")
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	5.1, 5.5

测试套件ID	#1-14

目标	测试添加订单时场馆不存在异常处理
要测试的特性	POST /addOrder.do，venueName="NonExistent Venue"，Service层抛出IllegalArgumentException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.7, 5.1

测试套件ID	#1-15
目标	测试用户完成订单功能
要测试的特性	POST /finishOrder.do?orderID=1，正常完成订单操作
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.10, 5.4

测试套件ID	#1-16
目标	测试完成不存在订单的异常处理
要测试的特性	POST /finishOrder.do?orderID=999，Service层抛出RuntimeException("订单不存在")
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	5.1, 5.4

测试套件ID	#1-17
目标	测试访问修改订单页面
要测试的特性	GET /modifyOrder.do?orderID=1，返回订单修改页面，包含订单和场馆信息
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.5, 1.7

测试套件ID	#1-18
目标	测试访问不存在订单的修改页面
要测试的特性	GET /modifyOrder.do?orderID=999，订单不存在时应抛出NullPointerException

测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	5.1

测试套件ID	#1-19
目标	测试订单对应场馆不存在时的修改页面
要测试的特性	GET /modifyOrder.do?orderID=1，订单存在但场馆不存在，应抛出NullPointerException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.7, 5.1

测试套件ID	#1-20
目标	测试未登录用户修改订单
要测试的特性	POST /modifyOrder，Session中无user属性，应抛出LoginException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	4.2, 5.1

测试套件ID	#1-21
目标	测试已登录用户成功修改订单
要测试的特性	POST /modifyOrder，参数：venueName="New Venue", date="2025-12-25", startTime="2025-12-22 10:00", hours="2", orderID="1"
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.5, 5.4

测试套件ID	#1-22
目标	测试修改订单时小时数边界值验证
要测试的特性	POST /modifyOrder，hours参数为"0"和"-1"，应抛出InvalidParameterException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	3.2, 5.1

测试套件ID	#1-23
目标	测试修改订单时服务层权限异常
要测试的特性	POST /modifyOrder，Service层抛出IllegalArgumentException("订单ID不存在")
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	4.3, 5.1

测试套件ID	#1-24
目标	测试用户删除订单功能
要测试的特性	POST /delOrder.do?orderID=1，正常删除订单，返回"true"
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.8

测试套件ID	#1-25
目标	测试删除不存在订单的异常处理
要测试的特性	POST /delOrder.do?orderID=999，Service层抛出RuntimeException("订单不存在")
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.8, 5.1

测试套件ID	#1-26
目标	测试获取特定日期场馆订单列表
要测试的特性	GET /order/getOrderList.do?venueName=Test Venue&date=2019-12-22，返回JSON格式的VenueOrder对象
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.4, 1.6

测试套件ID	#1-27

目标	测试获取特定日期无订单的场馆列表
要测试的特性	GET /order/getOrderList.do?venueName=Test Venue&date=2019-12-22，场馆存在但当日无订单，返回空订单数组
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.6

测试套件ID	#1-28
目标	测试查询不存在场馆的特定日期订单
要测试的特性	GET /order/getOrderList.do?venueName=NonExistent Venue&date=2019-12-22，场馆不存在时应抛出NullPointerException
测试工具	MockMvc, JUnit 5, Mockito
风险覆盖	1.7, 5.1

3.2.3 测试脚本

1. **设置项目**：如果我们手动运行单元测试，可以通过Maven test进行测试；CI/CD时，通过run workfile来进行自动化测试
2. **添加代码覆盖率插件**：Jacoco 是一个流行的代码覆盖率报告工具。我们可以使用它以任意粒度查看覆盖率信息。它还可以与 CodeCov 集成，在 GitHub 等 CI/CD 平台上自动生成详细的报告。
3. **创建测试类**：为我们的测试用例创建一个新的 Java 类。该类应添加 `@SpringBootTest` 注解，以便在 SpringCloud 环境下进行测试。
4. **编写测试方法**：在测试类中，为每个要测试的功能或单元编写独立的测试方法。在每个测试方法前添加 `@Test` 注解。确保方法名称具有描述性，并能反映测试的目的。
5. **准备测试数据**：在每个测试方法执行前，使用 `@Before` 或 `@BeforeEach` 注解设置必要的测试数据或测试环境。这可以确保每个测试用例的环境一致。
6. **执行断言**：在每个测试方法中，执行必要的操作，并使用 JUnit（如 `assertEquals`、`assertTrue` 等）和 WebMvc 的断言方法（如 `ResultActions` 对象的 `andExpect`）进行断言。将实际结果或响应与预期结果或响应进行比较，以判断被测试功能是否正常。
7. **处理异常**：如果测试的代码逻辑涉及 WebMvc，可以使用 `andExpect` 来预期 40x 或 50x 的响应。否则，可以简单地为预期异常添加注解。
8. **运行测试**：只需点击根项目 Maven 面板中的测试选项，即可轻松运行所有测试套件。

9. **分析测试结果**：测试完成后，报告将生成在 `target/site` 目录下，我们可以点击 `index.html` 查看报告。如果测试是在 CI/CD 环境下进行的，可以在 Pull Request 页面查看报告。

3.3 测试执行与结果

3.3.1 运行测试代码

3.3.2 覆盖率

meethere > com.meethere.controller.user

com.meethere.controller.user

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
OrderController		100%		100%	0	15	0	61	0	11	0	1
MessageController		100%		100%	0	9	0	41	0	7	0	1
UserController		100%		100%	0	16	0	48	0	10	0	1
VenueController		100%		n/a	0	4	0	12	0	4	0	1
NewsController		100%		n/a	0	4	0	12	0	4	0	1
Total	0 of 778	100%	0 of 24	100%	0	48	0	174	0	36	0	5

3.4 总结

3.4.1 确定覆盖范围

完成上述分析后，我们发现当前的测试用例覆盖了以下层次（按从低到高的顺序）：

1. 指令覆盖率：每个条件的真分支和假分支至少被测试一次。
2. 路径覆盖率：程序的所有可能路径至少被测试一次。

3.4.2 结论

在详细测试设计阶段，我们通过手动检查代码并创建带有分支方法的流程图，清晰地理解代码中的逻辑路径和潜在的分支条件。这一过程确保了我们的测试用例能够涵盖所有可能的分支情况，从而验证

代码的准确性和连贯性。

通过编写测试代码并使用JUnit、WebMvc等工具进行测试执行，我们确保测试用例涵盖了被测方法中的所有语句和分支。这种严谨的方法使得在实际测试中实现了100%的类覆盖率和90%的语句覆盖率。通过这一全面的测试过程，我们能够识别出潜在的问题和漏洞，从而增强我们对代码质量和可靠性的信心。

3.4.3 改进

我们已经确定，尽管测试用例可能无法实现条件组合覆盖，但进一步优化并非必要。这是因为系统代码设计的特点，即‘n’个错误输入条件实际上相当于一个错误输入条件。当遇到初始错误输入时，程序会立即停止验证并报告错误。值得注意的是，无论正确或错误条件的组合如何，程序的响应行为都不会改变。因此，在白盒测试阶段，我们决定不调整测试用例。

4. 测试结果和分析

在黑盒测试的过程中，我们发现：

1. 该模块的正确用例流程通过率高，表明其能够成功实现核心功能。
2. 当前端表单中存在空字段或无效输入时，系统无法拒绝提交，并提供相应的错误信息。仅当当表单输入全部有效且提交成功时，系统会提供合理的反馈
3. 后端系统缺乏有效的输入验证和异常处理机制，未能对某些无效输入或异常情况提供必要的错误拦截、处理及适当的响应消息，尤其是对时间范围的限制模块。

在白盒测试的阶段，我们发现：

1. **功能测试：** `OrderController.java` 文件的功能测试覆盖率达到 100%，表明所有代码行和分支都被测试执行过。然而，在执行的 28 个功能测试用例中，有 5 个失败，这表明尽管覆盖率高，但仍有特定功能场景的预期行为与实际测试结果不符。
2. **边界测试：** Jacoco 报告显示 `OrderController` 的所有分支都被覆盖，这意味着在边界条件下的代码路径已被触及。然而，测试结果中存在的失败用例 暗示了某些边界输入的处理可能未能满足预期的正确性或鲁棒性。
3. **异常处理测试：** `OrderController.java` 的代码覆盖率显示，与异常处理相关的代码路径（如抛出异常或捕获异常的逻辑）已被测试执行。尽管如此，测试中仍有失败项，这可能意味着在模拟特定的错误参数、未预见的意外情况或异常传递机制时，其处理方式未能完全符合测试用例的预期。