项目测试文档——单元测试

一. 组件划分与测试案例

1. 用户账号管理

- **组件**：AccountService
- **功能**：注册、登录、更新资料、密码修改
- **测试函数参考表：**

标识符	名称	代码行
LLD_001_FUN_001	void testLoadUserByUsername_UserFound()	14
LLD_001_FUN_002	void testLoadUserByUsername_UserNotFound()	10
LLD_001_FUN_003	void testRegisterEmailAccount_Success()	19
LLD_001_FUN_004	void testRegisterEmailAccount_EmailAlreadyRegistered()	15
LLD_001_FUN_005	void testRegisterEmailAccount_UsernameAlreadyTaken()	16
LLD_001_FUN_006	void testRegisterEmailAccount_InvalidCode()	14
LLD_001_FUN_007	void testRegisterEmailAccount_MissingCode()	14
LLD_001_FUN_008	void testRegisterEmailVerifyCode_Success()	14
LLD_001_FUN_009	void testRegisterEmailVerifyCode_RequestLimit()	12

测试案例：

- 注册：确保当提供有效信息时用户可以成功注册（包括成功收到邮件信息）；无效信息（如重复的用户名或邮箱、不符合要求的密码）时拒绝注册。
- 登录：验证正确和错误的登录凭据处理。
- 更新资料：确保用户可以更新自己的资料，包括邮箱、密码和头像。
- 密码修改：验证用户在提供正确的旧密码时可以成功更改密码，并确保在旧密码错误时拒绝更改。

2. 帖子管理

- **组件**：TopicService
- **功能**：发帖、编辑帖子
- **测试函数参考表：**

标识符	名称	代码行
LLD_002_FUN_001	testListTypes_Success	10
LLD_002_FUN_002	testCreateTopic_Success	16
LLD_002_FUN_003	testUpdateTopic_Success	16
LLD_002_FUN_004	testGetTopic_Success	16

测试案例：

- 发帖：验证用户登录状态下可以成功发帖，并检查帖子存储在数据库中的正确性。
- 编辑帖子：验证用户可以编辑自己的帖子，且编辑内容正确更新到数据库。

3. 账户信息系统

- **组件**：AccountDetailsService, AccountPrivacyService
- **功能**：保存隐私、新建隐私、查找账户细节、保存账户细节
- **测试函数参考表：**

标识符	名称	代码行
LLD_003_FUN_001	testSavePrivacy_ExistingPrivacy	9

LLD_003_FUN_002	testSavePrivacy_NewPrivacy	9
LLD_003_FUN_003	testAccountPrivacy_ExistingPrivacy	9
LLD_003_FUN_004	testAccountPrivacy_NewPrivacy	9
LLD_003_FUN_005	testFindAccountDetailsById	9
LLD_003_FUN_006	testSaveAccountDetails_Success	22
LLD_003_FUN_007	testSaveAccountDetails_AccountExistsWithDifferentID	11

测试案例：

- 保存隐私：验证已登录用户能够成功更新其账户隐私设置，且更新后的设置在数据库中准确无误。
- 新建隐私：确保用户可以为账户添加新的隐私选项，并验证该设置被正确存储和应用到用户账户中。
- 查找账户细节：验证系统能够根据正确的账户ID返回完整的账户详细信息，并检查信息的准确性。
- 保存账户细节：测试用户在登录状态下能够成功保存或更新其账户的个人信息，同时验证更新的数据是否正确反映在数据库中。

4. 通知系统

- **组件**：NotificationService
- **功能**：生成通知、显示通知、删除通知
- **测试函数参考表：**

标识符	名称	代码行
LLD_004_FUN_001	testAddNotification	12
LLD_004_FUN_002	testFindUserNotification	22
LLD_004_FUN_003	testDeleteUserNotification	6

LLD_004_FUN_004	testDeleteUserAllNotification	5

测试案例：

- 生成通知：验证系统在触发特定事件（如评论或点赞）时自动生成通知。
- 显示通知：验证用户可以查看所有相关的通知。
- 删除通知：验证用户可以删除他们的通知记录。

5. 点赞和收藏功能

- **组件**：InteractionService
- **功能**：点赞帖子、取消点赞、收藏帖子、取消收藏
- **测试函数参考表：**

标识符	名称	代码行
LLD_005_FUN_001	testCreateComment_Success	16
LLD_005_FUN_002	testDeleteComment_Success	8
LLD_005_FUN_003	testInteract_Success	10

测试案例：

- 点赞帖子：验证用户可以对帖子进行点赞，且点赞计数更新正确。
- 取消点赞：验证用户可以取消点赞，且点赞计数正确更新。
- 收藏帖子：验证用户可以收藏帖子，且收藏状态正确记录在数据库。
- 取消收藏：验证用户可以取消收藏，且数据库状态正确更新。

6. 图片管理系统

- **组件**：ImageService
- **功能**：拉取图片、上传图片、上传头像
- **测试函数参考表：**

标识符	名称	代码行

LLD_006_FUN_001	testFetchImageFromMinio_Success	15
LLD_006_FUN_002	testUploadImage_Success	15
LLD_006_FUN_003	testUploadImage_LimitExceeded	13
LLD_006_FUN_004	testUploadAvatar_Success	19
LLD_006_FUN_005	testUploadAvatar_UpdateFailed	19

测试案例：

- 拉取图片：确保系统能够成功从MinIO中拉取指定的图片资源。
- 上传图片：验证用户能够成功上传图片至MinIO，并在数据库中记录相关信息。
- 上传头像：验证用户能成功上传并更新个人头像到MinIO，同时在用户资料中正确反映更新。

二. 测试方法

1. 保证所有的语句、分支被覆盖
2. 参考等价类划分方法
3. 参考边界值分析方法
4. 参考使用错误猜测方法
5. 测试语言都是基于java语言
6. 整个项目的测试都是基于springboot框架中提供的test脚本方法，编写单元测试类进行测试。
7. 单元测试采用Mockito工具来通过虚拟类模拟实际的类与方法，方便进行操作。

三. 测试环境

1. 现有一台笔记本电脑
2. 基于idea环境
3. 配备有springboot框架
4. 通过Maven来管理引入依赖（包括Mockito的依赖，junit单元测试依赖等等）

三. 测试对象

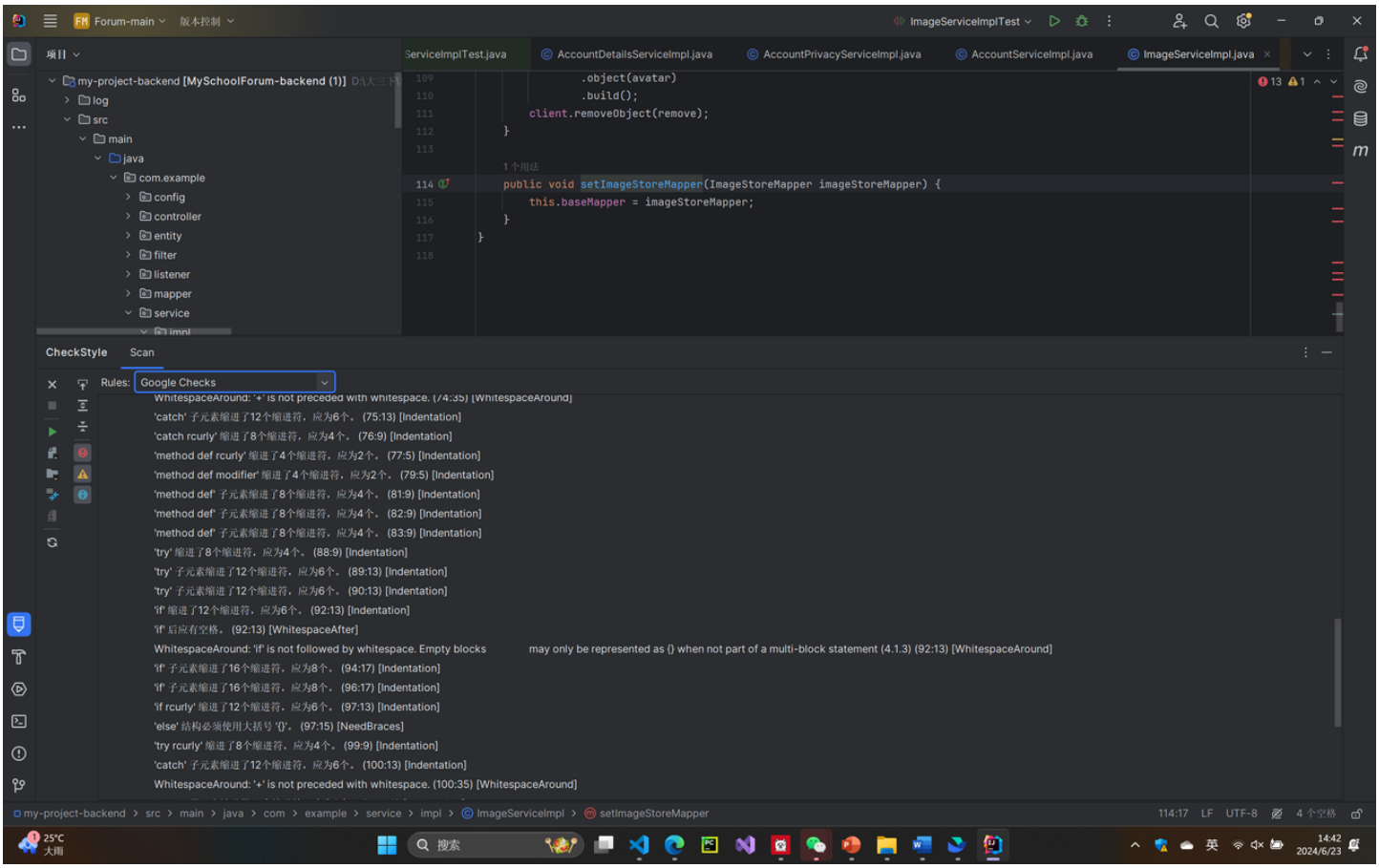
1. 用户账号管理系统
2. 帖子管理系统
3. 评论系统
4. 通知系统
5. 点赞和收藏系统

四. 静态测试

1. 集成开发环境：IntelliJ IDEA

2. 使用的开源工具：CheckStyle

3. 测试结果



五.用例分析与设计

1.用户账号管理系统

1.1 testLoadUserByUsername的测试分析与设计

我们对此设计了两个函数分别测试找得到用户后添加成功的情况以及找不到用户添加失败两种情况。

1.1.1 标识符定义

UT_TD_001_001

1.1.2 被测特性

输入的用户名在数据库中不存在，显示登录失败

输入的用户名正确，则登录成功

1.1.3 测试方法

对于输入的账户信息的等价类划分可以考虑空和非空的两种情况，对于非空的情况，又可以划分为账户信息能找得到和找不到两种情况

1.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_001_001_001	输入的用户信息不存在数据库	低
UT_TC_001_001_002	输入的用户信息存在于数据库	高

1.1.5 测试通过/失败标识

所有的测试用例都被执行同时不存在错误。

1.1.6 用例设计

测试项编号	UT_TC_001_001_001	
优先级	低	
测试项描述	输入的用户信息不存在数据库	
前置条件	用户登录主界面，输入账户信息进行查询	
用例序号	输入	期望结果额
001	username = "nonexistentuser"	返回false，User Not Found的详细信息

测试项编号	UT_TC_001_001_002
优先级	高
测试项描述	输入的用户信息存在于数据库之中

前置条件	用户登录主界面，输入账户信息进行查询	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode("123456");
		返回true，给出查询成功的返回信息

1.2 testRegisterEmailAccount的测试分析与设计

1.2.1 标识符定义

UT_TD_002_001

1.2.2 被测特性

想要注册的邮箱已经存在于数据库中时，提示邮箱已经存在注册失败信息。

输入的用户名已经被其他人注册过了，也就是数据库中存在该用户名信息，显示注册失败。

输入的验证码是错误的，提示验证码错误。

输入的验证码是空置，提示没有验证码信息。

当所有信息合理的时候，显示注册成功。

1.2.3 测试方法

对于输入的注册信息的等价类划分可以考虑空和非空的两种情况，对于非空的情况，又可以划分为验证码信息错误，验证码信息缺失，邮箱信息错误，用户名信息错误四种等价类。

1.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_001_002_001	邮箱信息已经被注册过	低
UT_TC_001_002_002	用户名信息已经被注册过	低
UT_TC_001_002_003	验证码信息错误	低
UT_TC_001_002_004	验证码信息丢失	低
UT_TC_001_002_005	注册信息合理	高

1.2.5 测试通过/失败标识

所有的测试用例都被执行同时不存在错误。

1.2.6 用例设计

测试项编号	UT_TC_001_002_001	
优先级	低	
测试项描述	邮箱信息已经被注册过	
前置条件	用户登录主界面，输入账户信息进行注册	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode("123456");
		返回false，以及邮箱信息已经被注册过的详细信息

测试项编号	UT_TC_001_002_002	
优先级	低	
测试项描述	用户名信息已经被注册过	
前置条件	用户登录主界面，输入账户信息进行注册	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode("123456");
		返回false，以及用户信息已经被注册过的详细信息

测试项编号	UT_TC_001_002_003	
优先级	低	
测试项描述	验证码信息错误	
前置条件	用户登录主界面，输入账户信息进行注册	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode("123456");
		返回false，验证码信息错误

测试项编号	UT_TC_001_002_004	
优先级	低	
测试项描述	验证码信息丢失	
前置条件	用户登录主界面，输入账户信息进行注册	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode(null);
		返回false，验证码信息丢失

测试项编号	UT_TC_001_002_005	
优先级	高	
测试项描述	注册信息正确，注册成功	
前置条件	用户登录主界面，输入账户信息进行注册	
用例序号	输入	期望结果额

001	info.setEmail("test@example.com");
	info.setUsername("testuser");
	info.setPassword("password");
	info.setCode("123456");
		返回true，以及注册成功的详细信息

1.3 testRegisterEmailVerifyCode的测试分析与设计

1.3.1 标识符定义

UT_TD_003_001

1.3.2 被测特性

注册时为邮箱发送验证码信息，当请求人数较少时请求成功。

当请求人数很多时，请求失败。

1.3.3 测试方法

将邮箱注册划分为注册成功和注册失败两个等价类。

1.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_001_003_001	验证码发送失败	低
UT_TC_001_003_002	验证码发送成功	高

1.3.5 测试通过/失败标识

所有的测试用例都被执行同时不存在错误。

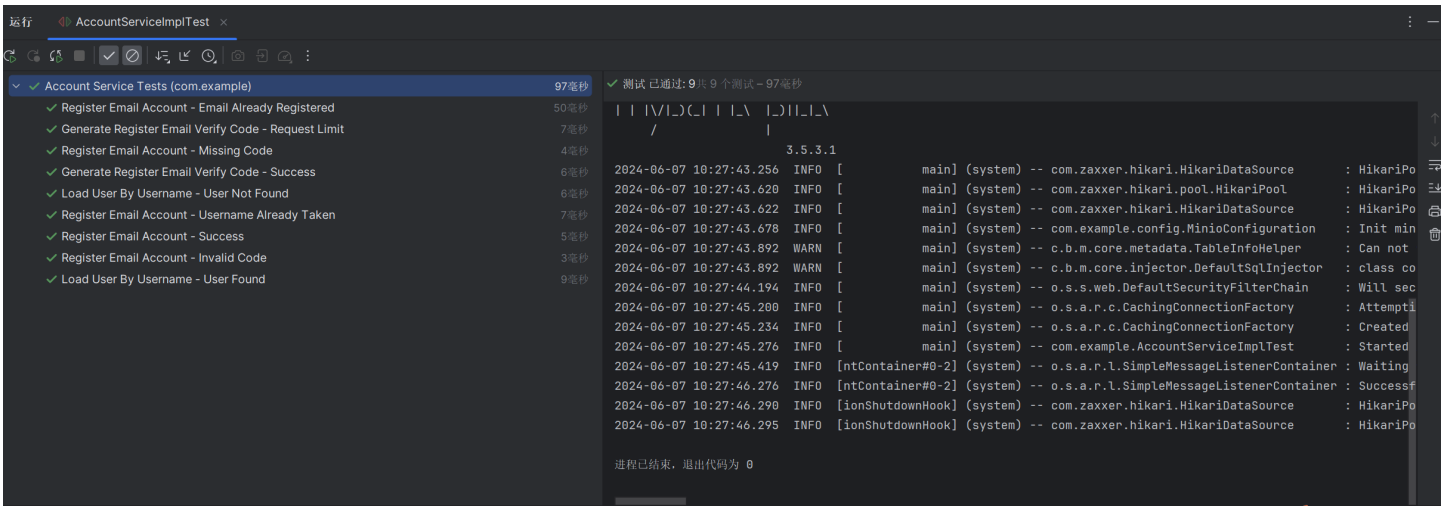
1.3.6 用例设计

测试项编号	UT_TC_001_003_001	
优先级	低	
测试项描述	验证码发送失败	
前置条件	用户登录主界面，输入账户信息，点击发送验证码	
用例序号	输入	期望结果额
001	String type = "register"; String email = "test@example.com"; String address = "127.0.0.1";	返回false，显示请求失败

测试项编号	UT_TC_001_003_002	
优先级	高	
测试项描述	验证码发送成功	
前置条件	用户登录主界面，输入账户信息，点击发送验证码	
用例序号	输入	期望结果额

001	String type = "register"; String email = "test@example.com"; String address = "127.0.0.1";	返回true，显示已发送验证码到邮箱中

1.4 测试结果



2.帖子管理系统

2.1 testListTypes的测试分析与设计

此测试针对的是查询话题类型列表功能的成功场景，确保当调用相关服务方法时，能够正确地从数据库中获取到话题类型列表，并且返回的列表不为空且包含预期数量的元素。

2.1.1 标识符定义

UT_TD_002_001

2.1.2 被测特性

- 数据库中存在话题类型记录时，能够成功查询并返回这些记录。
- 查询过程调用了正确的数据访问层方法，并处理返回结果正确。

2.1.3 测试方法

- 使用Mockito框架模拟 `topicTypeMapper.selectList()` 方法的行为，使其返回一个包含预设话题类型的集合。
- 验证 `topicService.listTypes()` 方法调用后返回的列表非空，并且其大小与预期相符。
- 确认 `topicTypeMapper.selectList()` 方法被正确调用了一次，以验证服务层正确委托给了数据访问层。

2.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_002_001_001	数据库存在话题类型记录时，查询方法应成功返回这些记录	高

2.1.5 测试通过/失败标识

测试通过的条件是：所有预期的方法调用均按计划发生，返回的话题类型列表非空，且列表中的元素数量与预期一致。

2.1.6 用例设计

测试项编号	UT_TC_002_001_001	
优先级	高	
测试项描述	当数据库中预设了话题类型记录时，调用 `topicService.listTypes()` 方法能够成功获取这些记录并返回。	
前置条件	系统已配置好Mock对象，模拟 `topicTypeMapper.selectList()` 方法返回一个包含至少一个 `TopicType` 对象的列表。	
用例序号	输入	期望结果额
001		• `topicService.listTypes()` 方法调用后返回的 `topicTypes` 列表不为null。 • 列表 `topicTypes` 的大小为1，与预期的模拟返回值一致。 • 确认 `topicTypeMapper.selectList(any())` 被正确调用了一次，证明服务层逻辑正确执行了数据访问操作。

2.2 testCreateTopic的测试分析与设计

此测试案例旨在验证创建话题功能在一切正常（包括限流检查通过、数据库插入成功及缓存清理）的情况下能够成功执行，并且最终不返回任何错误信息。

2.2.1 标识符定义

UT_TD_002_002

2.2.2 被测特性

- 限流检查允许创建新话题。

- 数据库插入新话题记录成功。
- 话题类型查询成功返回预期数据。
- 创建话题后触发缓存清理机制。

2.2.3 测试方法

- 使用Mockito模拟 `flowUtils.limitPeriodCounterCheck()`、`topicMapper.insert()` 和 `topicTypeMapper.selectList()` 方法，使它们分别返回预设的成功结果或数据。
- 构造 `TopicCreateVO` 对象作为创建话题的参数，模拟用户提供的标题、内容和类型信息。
- 调用 `topicService.createTopic()` 方法，并验证它在预期条件下执行后的副作用，如数据库插入调用、缓存清理调用等。

2.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_002_002_001	在满足限流条件及数据库插入成功时，创建话题操作应执行无误	高

2.2.5 测试通过/失败标识

测试通过的条件为：`topicService.createTopic()` 调用后不返回错误信息（即返回null），并且相关的Mock方法被正确调用。

2.2.6 用例设计

测试项编号	UT_TC_002_002_001	
优先级	高	
测试项描述	当系统未达到创建话题的频率限制，且数据库操作与话题类型查询均成功时，`topicService.createTopic()` 应当能够无错误地完成话题创建。	
前置条件	通过Mockito配置了 `flowUtils.limitPeriodCounterCheck()`、`topicMapper.insert()` 和 `topicTypeMapper.selectList()` 方法的返回行为。	
用例序号	输入	期望结果额
001	Title = "New Title"; Content = "New Content"; Type = 1; ID = 1;	• 方法调用后返回值为null，表示没有错误信息。

		• 确认 `topicMapper.insert(any(Topic.class))` 被调用一次，表明话题创建记录尝试被插入到数据库。 • 确认 `cacheUtils.deleteCachePattern(anyString())` 被调用，验证了在创建话题后，相关的缓存清理逻辑被执行。

2.3 testUpdateTopic的测试分析与设计

本测试案例专注于验证更新话题功能在正常流程下（即话题类型查询成功及数据库更新操作成功）能够顺利执行，并且最终无错误返回。

2.3.1 标识符定义

UT_TD_002_003

2.3.2 被测特性

- 成功查询到话题类型信息。
- 更新数据库中指定话题记录的操作成功执行。
- 更新操作后不返回错误信息。

2.3.3 测试方法

- 利用Mockito模拟 `topicTypeMapper.selectList()` 方法，使其返回一个包含预期话题类型的列表。
- 设置 `topicMapper.update()` 方法的返回值为1，模拟话题记录更新成功。
- 构建 `TopicUpdateVO` 对象，模拟用户提交的更新信息，包括话题ID、新标题、新内容及新类型。
- 执行 `topicService.updateTopic()` 方法并验证其执行效果及后续影响。

2.3.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_002_003_001	数据库更新操作成功且返回正确响应时，更新话题功能应执行无误	高

2.3.5 测试通过/失败标识

测试通过的条件为：更新操作执行后返回null，表明没有错误；同时，`topicMapper.update()`被正确调用一次。

2.3.6 用例设计

测试项编号	UT_TC_002_003_001	
优先级	高	
测试项描述	在话题类型查询成功且数据库更新操作预期执行时，`topicService.updateTopic()`应能无误地完成话题信息的更新。	
前置条件	通过Mockito配置了`topicTypeMapper.selectList()`和`topicMapper.update()`方法的返回行为。	
用例序号	输入	期望结果额
001	Title = "Updated Title"; Content = "Updated Content"; Type = 1; ID = 1;	• 方法调用后返回值为null，表示更新操作没有遇到错误。 • 确认`topicMapper.update(any(), any(UpdateWrapper.class))`被调用一次，确认数据库更新操作被执行。

2.4 testGetTopic的测试分析与设计

本测试案例关注于获取话题详情功能的成功场景，确保在所有依赖服务和数据访问正常响应时，能够正确获取话题详情并封装为`TopicDetailVO`对象。

2.4.1 标识符定义

UT_TD_002_004

2.4.2 被测特性

- 通过话题ID成功查询到话题信息。
- Redis操作及账户相关信息查询均返回预期数据。
- 话题详情封装逻辑正确无误。

2.4.3 测试方法

- 使用Mockito模拟`topicMapper.selectById()`方法，返回预设的话题对象。
- 对`stringRedisTemplate.opsForHash()`返回一个模拟的`HashOperations`实例，模拟Redis操作。

- 分别设置 `accountMapper.selectById()` 、`accountDetailsMapper.selectById()` 和 `accountPrivacyMapper.selectById()` 方法的返回值，以模拟账户及其详细信息、隐私信息的查询成功。
- 调用 `topicService.getTopic()` 方法，并验证其返回的 `TopicDetailVO` 对象的有效性。

2.4.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_002_004_001	正确查询并封装话题详情信息，包括从数据库及Redis获取的数据	高

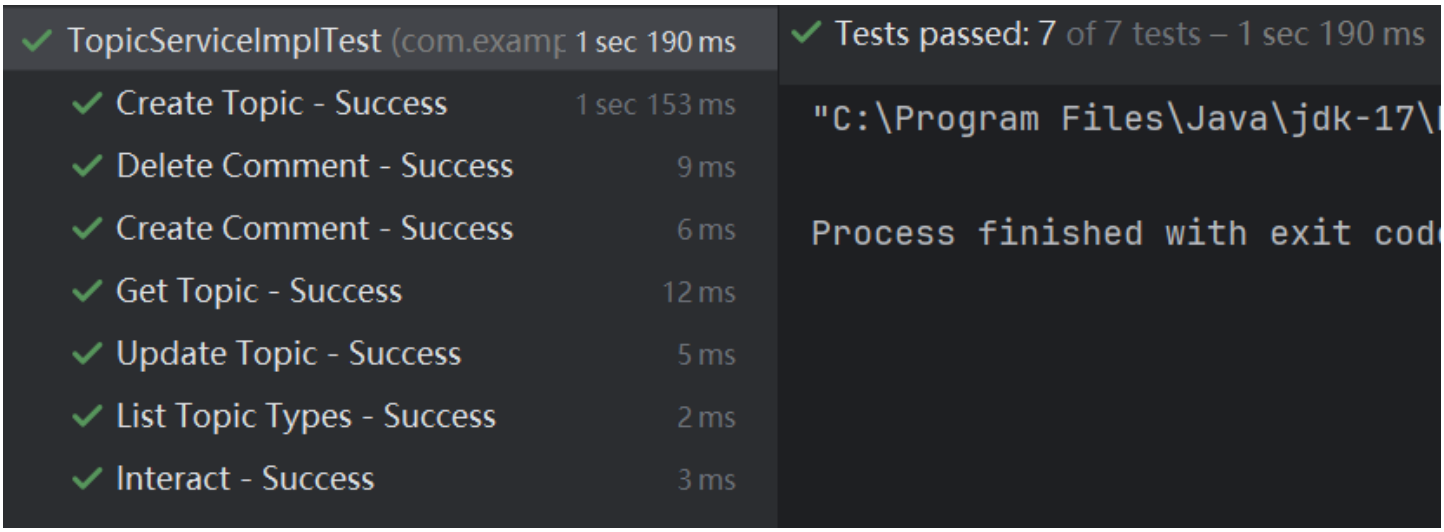
2.4.5 测试通过/失败标识

测试通过的条件是： `topicService.getTopic()` 返回的 `TopicDetailVO` 不为null，且其中的标题字段与预期话题对象的标题相匹配；所有预期的Mock方法被正确调用。

2.4.6 用例设计

测试项编号	UT_TC_002_004_001	
优先级	高	
测试项描述	在所有依赖服务（如数据库查询、Redis操作）均响应成功的情况下，验证 `topicService.getTopic()` 方法能否正确获取并封装话题详情信息。	
前置条件	通过Mockito配置了 `topicMapper.selectById()` 、`stringRedisTemplate.opsForHash()` 、 `accountMapper.selectById()` 、`accountDetailsMapper.selectById()` 及 `accountPrivacyMapper.selectById()` 方法的返回行为。	
用例序号	输入	期望结果额
001	Tid = 1; Uid = 1;	• 返回的 `TopicDetailVO` 对象不为null。 • `TopicDetailVO` 对象的标题属性与预设话题对象的标题属性相等。 • 确认 `topicMapper.selectById(anyInt())` 被调用一次，验证了数据库查询逻辑的执行。

2.5 测试结果



3.账户信息系统

3.1 testSavePrivacy的测试分析与设计

这两个测试用例分别检验了在不同场景下保存用户账户隐私设置的功能：一种是隐私设置已存在的情况（`testSavePrivacy_ExistingPrivacy`），另一种是隐私设置尚不存在的情况（`testSavePrivacy_NewPrivacy`）。

3.1.1 标识符定义

UT_TD_003_001

3.1.2 被测特性

- 系统应能准确识别给定账户ID已存在的隐私记录。
- 保存新的隐私设置时，应对现有记录进行更新，避免产生重复记录。
- 如果给定账户ID没有找到相应的隐私记录，系统应创建一个新的记录。
- 保存操作应触发向数据库插入一条新的 `AccountPrivacy` 记录。

3.1.3 测试方法

两个测试均利用模拟行为来模拟数据库交互：

- 已有隐私设置： `accountPrivacyMapper.selectById()` 模拟返回非空的 `AccountPrivacy` 对象。
- 新建隐私设置： `accountPrivacyMapper.selectById()` 模拟返回 `null`。随后调用 `accountPrivacyService.savePrivacy()` 方法，并通过以下方式验证：
- 验证 `accountPrivacyMapper.selectById(1)` 被正确调用，以判断记录是否存在。
- 使用 `verify` 方法确保在两种情况下 `accountPrivacyMapper.insertOrUpdate(any(AccountPrivacy.class))` 均被恰当地调用了一次，表明无论是更新还是插入操作都按预期执行。

3.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_003_001_001	当隐私设置已存在时，更新隐私设置信息并验证更新操作	低
UT_TC_003_001_002	当隐私设置不存在时，插入新的隐私设置信息并验证插入操作	高

3.1.5 测试通过/失败标识

- 测试通过：对于每个测试用例，如果隐私设置的保存操作（更新或插入）执行成功，且相应的方法调用验证无误，则视为测试通过。
- 测试失败：如果预期的数据库操作未执行（如未更新已存在的隐私设置或未插入新设置），或者方法调用验证失败，则视为测试失败。

3.1.6 用例设计

测试项编号	UT_TC_003_001_001	
优先级	低	
测试项描述	确保当用户已存在隐私设置时，调用 `savePrivacy` 方法能够正确更新这些设置。	
前置条件	数据库中已存在该用户的隐私设置记录。	
用例序号	输入	期望结果额
001	ID = 1; privacySaveVO;	• 数据库中的隐私设置记录被正确更新。 • `accountPrivacyMapper.selectById(1)` 被调用验证原有记录存在。 • `accountPrivacyMapper.insertOrUpdate()` 被调用一次，用于更新操作。

测试项编号	UT_TC_003_001_002
优先级	高
测试项描述	验证在没有预先存在的隐私设置情况下，`savePrivacy` 方法能成功创建新的隐私设置记录。

前置条件	数据库中没有该用户的隐私设置记录。	
用例序号	输入	期望结果额
001	ID = 1; privacySaveVO;	• 数据库中新增了该用户的隐私设置记录。 • `accountPrivacyMapper.selectById(1)` 被调用返回 `null`，确认原记录不存在。 • `accountPrivacyMapper.insertOrUpdate()` 被调用一次，用于插入新记录。

3.2 testAccountPrivacy的测试分析与设计

设计了两个测试用例，分别测试存在和不存在的情况，验证系统能够正确查询并返回数据库中已存在的用户隐私设置。

3.2.1 标识符定义

UT_TD_003_002

3.2.2 被测特性

- 已有隐私设置查询： 验证系统能够正确查询并返回数据库中已存在的用户隐私设置。
- 新隐私设置处理： 验证在数据库中未找到用户隐私设置时，系统如何处理并返回默认或新建的隐私设置。

3.2.3 测试方法

- 使用Mockito框架模拟 `accountPrivacyMapper.selectById()` 方法，分别返回已存在的隐私设置对象和 `null`，以模拟两种不同的查询结果场景。
- 直接调用 `accountPrivacyService.accountPrivacy()` 方法，并验证其返回结果。

3.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_003_002_001	确保服务层能够正确处理数据库中已有的隐私设置记录，并将其准确返回。	低
UT_TC_003_002_002	验证在没有查到用户隐私设置时，服务层是否提供了合理的默认处理逻辑或创建新记录的逻辑。	高

3.2.5 测试通过/失败标识

- 通过： 当查询已有隐私设置时，返回的结果与预期的隐私设置对象相匹配；在没有隐私设置记录时，返回的对象至少包含基本的ID信息且非空。
- 失败： 查询已有隐私设置时返回 `null` 或内容不符；查询不到记录时返回 `null` 或未正确初始化新对象。

3.2.6 用例设计

测试项编号	UT_TC_003_002_001	
优先级	低	
测试项描述	验证当用户隐私设置已存在时，服务能正确返回该用户的隐私设置信息。	
前置条件	数据库中存在ID为1的账户隐私设置记录。	
用例序号	输入	期望结果额
001	ID = 1;	• 返回的 `AccountPrivacy` 对象不为 `null` ，且与预期的 `accountPrivacy` 对象相等。

测试项编号	UT_TC_003_002_002	
优先级	高	
测试项描述	在数据库中没有查到指定用户的隐私设置时，验证服务是否能够返回一个默认初始化或新创建的隐私设置对象。	
前置条件	数据库中不存在ID为1的账户隐私设置记录。	
用例序号	输入	期望结果额
001	ID = 1;	• 返回的 `AccountPrivacy` 对象不为 `null` ，且其ID属性为1，表明服务可能采取了默认创建或返回默认设置的策略。

3.3 testFindAccountDetailsById的测试分析与设计

设计测试用例验证系统能够根据账户ID从数据库中准确查询并返回账户的详细信息记录。

3.3.1 标识符定义

UT_TD_003_003

3.3.2 被测特性

- 查询账户详细信息： 验证系统能够根据账户ID从数据库中准确查询并返回账户的详细信息记录。

3.3.3 测试方法

- 利用Mockito框架模拟 `accountDetailsMapper.selectById()` 方法，使其返回预设的 `accountDetails` 对象，模拟数据库查询操作。
- 调用 `accountDetailsService.findAccountDetailsById()` 方法，并检查其返回结果。

3.3.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_003_003_001	确保服务层方法 `findAccountDetailsById()` 在接收到有效的账户ID时，能够正确委托给数据访问层执行查询操作，并将查询到的账户详细信息对象返回。	高

3.3.5 测试通过/失败标识

- 通过标准： 调用 `findAccountDetailsById()` 后返回的账户详细信息对象非空，并且与预期的 `accountDetails` 对象完全一致。
- 失败情况： 返回结果为 `null` ，或返回的账户详细信息与预期不符。

3.3.6 用例设计

测试项编号	UT_TC_003_003_001	
优先级	高	
测试项描述	验证当请求查询的账户ID对应有记录时，服务能正确返回该账户的详细信息。	
前置条件	数据库中存在ID为1的账户详细信息记录。	
用例序号	输入	期望结果额
001	ID = 1;	• 返回的 `AccountDetails` 对象不为 `null` ，且与预期的 `accountDetails` 对象完全相等，表明查询操作成功执行并返回了正确的数据。

3.4 testSaveAccountDetails的测试分析与设计

设计了两个测试用例，验证在账户名或邮箱未被其他账户占用时，能够成功保存或更新账户详细信息。

3.4.1 标识符定义

UT_TD_003_004

3.4.2 被测特性

- 成功保存账户详细信息： 验证在账户名或邮箱未被其他账户占用时，能够成功保存或更新账户详细信息。
- 账户名或邮箱冲突： 检查当尝试保存的账户名或邮箱已被其他不同ID的账户使用时，保存操作应失败。

3.4.3 测试方法

- 使用Mockito模拟 `accountService.findAccountByNameOrEmail()` 、 `accountService.update()` 链式调用、 `accountDetailsMapper.insert()` 、 `accountDetailsMapper.updateById()` 等方法，构造不同的测试场景。
- 直接调用 `accountDetailsService.saveAccountDetails()` 方法，通过断言和Mockito的 `verify` 方法验证预期的行为。

3.4.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_003_004_001	确保在无冲突情况下，账户详细信息和服务层的账户信息都能正确更新。	低
UT_TC_003_004_002	验证当尝试保存的账户名或邮箱与数据库中其他ID的账户相同，操作应被阻止。	高

3.4.5 测试通过/失败标识

- 成功保存： 操作返回 `true` ，且所有预期的Mock方法被调用正确次数。
- 账户已存在（不同ID）： 操作返回 `false` ，且相关更新和插入方法未被调用。

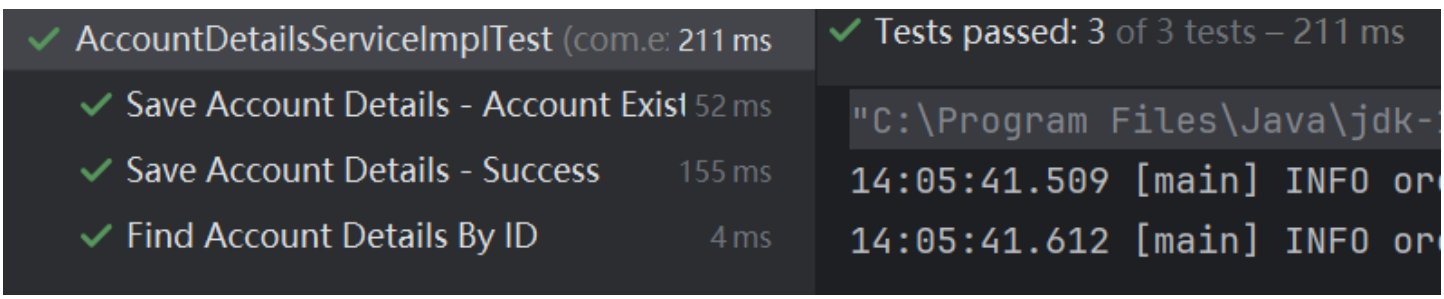
3.4.6 用例设计

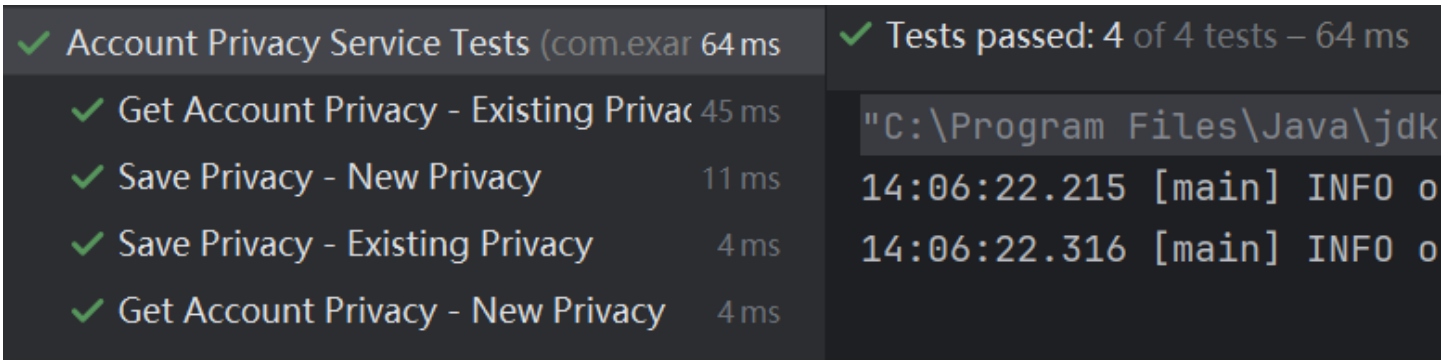
测试项编号	UT_TC_003_004_001
优先级	低
测试项描述	在账户名或邮箱未被占用时，确保账户详细信息能够被正确保存或更新。

前置条件	数据库查询返回账户名或邮箱未被使用。	
用例序号	输入	期望结果额
001	ID = 1; detailsSaveVO;	• 方法返回 `true` ，表示保存成功。 • 验证更新账户信息的Mock方法被正确调用。 • 验证 `accountDetailsMapper.insertOrUpdate()` 被调用，表示账户详细信息被正确处理。

测试项编号	UT_TC_003_004_002	
优先级	高	
测试项描述	当尝试保存的账户名或邮箱与数据库中不同ID的账户相同，保存操作应被拒绝。	
前置条件	数据库中有账户名或邮箱与输入相匹配，但ID不同。	
用例序号	输入	期望结果额
001	ID = 2; Username = "otheruser"; Password = "password"; Email = "other@example.com"; Role = "USER"; Avatar = "avatar.png"; Date = Date();	• 方法返回 `false` ，表示保存失败。 • 确认账户信息更新和账户详细信息插入或更新方法均未被调用，验证操作被合理阻止。

3.5 测试结果





4.通知系统

4.1 testAddNotification的测试分析与设计

设计了一个测试用例用来验证系统能够正确添加新的通知信息至数据库，包括通知接收者ID、标题、内容、类型及链接地址。

4.1.1 标识符定义

UT_TD_004_001

4.1.2 被测特性

- 添加通知功能： 验证系统能够正确添加新的通知信息至数据库，包括通知接收者ID、标题、内容、类型及链接地址。

4.1.3 测试方法

- 使用Mockito框架创建一个 `Notification` 对象模拟待添加的通知数据。
- 调用 `notificationService.addNotification()` 方法，传入与模拟对象相同的数据。
- 验证 `notificationMapper.insert()` 方法是否被正确调用，并传入了预期的参数。

4.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_004_001_001	确保服务层的 `addNotification` 方法能够接收指定参数，构建 `Notification` 实体，并委托给数据访问层执行插入操作。	高

4.1.5 测试通过/失败标识

- 通过： 当 `notificationMapper.insert()` 被正确调用，并且传入的参数与预期的模拟通知对象相等。
- 失败： 如果 `notificationMapper.insert()` 未被调用，或传入的参数不匹配预期值。

4.1.6 用例设计

测试项编号	UT_TC_004_001_001	
优先级	高	
测试项描述	验证添加通知功能在接收到完整且有效参数时，能够成功将通知信息添加到数据库。	
前置条件	数据库连接正常，通知服务及映射器配置无误。	
用例序号	输入	期望结果额
001	Uid = 1; Title = "Maintenance"; Content = "System will be down"; Type = "Alert"; Url = "http://example.com/maintenance";	• `notificationMapper.insert()` 被验证调用了一次，且传入的参数与创建的 `mockNotification` 对象完全匹配，包括所有字段值均一致。

4.2 testFindUserNotification的测试分析与设计

设计了一个测试用例，验证系统能够根据用户ID从数据库中查询并返回该用户的全部通知信息列表。

4.2.1 标识符定义

UT_TD_004_002

4.2.2 被测特性

- 查询用户通知： 验证系统能够根据用户ID从数据库中查询并返回该用户的全部通知信息列表。

4.2.3 测试方法

- 使用Mockito框架构造一个包含单个 `Notification` 对象的列表，模拟数据库查询结果。
- 配置 `notificationMapper.selectList()` 方法的返回值为上述模拟列表。
- 调用 `notificationService.findUserNotification()` 方法，并检查其返回的 `NotificationVO` 列表。

4.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_004_002_001	确保服务层方法能够正确处理查询请求，将查询到的通知信息转换为 `NotificationVO` 对象列表，并返回给调用	高

	方。	

4.2.5 测试通过/失败标识

- 通过： 返回的 `NotificationVO` 列表非空，且其大小、内容与预期的模拟数据一致。
- 失败： 返回列表为空，大小不符，或列表内容与预期不符。

4.2.6 用例设计

测试项编号	UT_TC_004_002_001	
优先级	高	
测试项描述	验证在用户有相关通知记录时，服务能返回正确的通知列表。	
前置条件	数据库中存在与用户ID关联的通知记录。	
用例序号	输入	期望结果额
001	ID = 1;	• 返回的 `NotificationVO` 列表非空。 • 列表大小为1，与设置的模拟数据相符。 • 列表中第一个元素的标题为"New Update"，验证数据内容正确性。

4.3 testDeleteUserNotification的测试分析与设计

设计测试用例验证系统能够根据用户ID和通知ID删除指定的通知记录。

4.3.1 标识符定义

UT_TD_004_003

4.3.2 被测特性

- 删除用户通知： 验证系统能够根据用户ID和通知ID删除指定的通知记录。

4.3.3 测试方法

- 直接调用 `notificationService.deleteUserNotification()` 方法，传入用户ID和通知ID作为参数。
- 使用Mockito验证 `notificationMapper.delete()` 方法是否被正确调用，以确认删除操作的执行。

4.3.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_004_003_001	确认服务层方法能够正确构建删除请求，并委派给数据访问层执行通知记录的删除操作。	高

4.3.5 测试通过/失败标识

- 通过： 当调用 `notificationMapper.delete()` 方法的验证成功，表明删除操作已按预期执行。
- 失败： 如果 `notificationMapper.delete()` 方法的调用验证未通过，表明删除操作未被执行或执行方式有误。

4.3.6 用例设计

测试项编号	UT_TC_004_003_001	
优先级	高	
测试项描述	验证在指定用户ID和通知ID下，通知能够被成功删除。	
前置条件	数据库中存在指定用户ID和通知ID对应的记录。	
用例序号	输入	期望结果额
001	ID = 1; UID = 1;	• 无需直接验证返回结果，因为该操作通常不返回具体值或返回通用的成功标志。 • 确认 `notificationMapper.delete(any(QueryWrapper.class))` 被调用，表明删除逻辑被执行。

4.4 testDeleteUserAllNotification的测试分析与设计

设计了测试用例验证系统能够根据用户ID删除该用户的所有通知记录。

4.4.1 标识符定义

UT_TD_004_004

4.4.2 被测特性

- 删除用户所有通知： 验证系统能够根据用户ID删除该用户的所有通知记录。

4.4.3 测试方法

- 调用 `notificationService.deleteUserAllNotification()` 方法，传入用户ID作为参数。
- 使用Mockito验证 `notificationMapper.delete()` 方法是否被调用，以确认批量删除操作的执行。

4.4.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_004_004_001	确保服务层能够针对特定用户ID生成正确的删除请求，并成功委托给数据访问层清除该用户的所有通知记录。	高

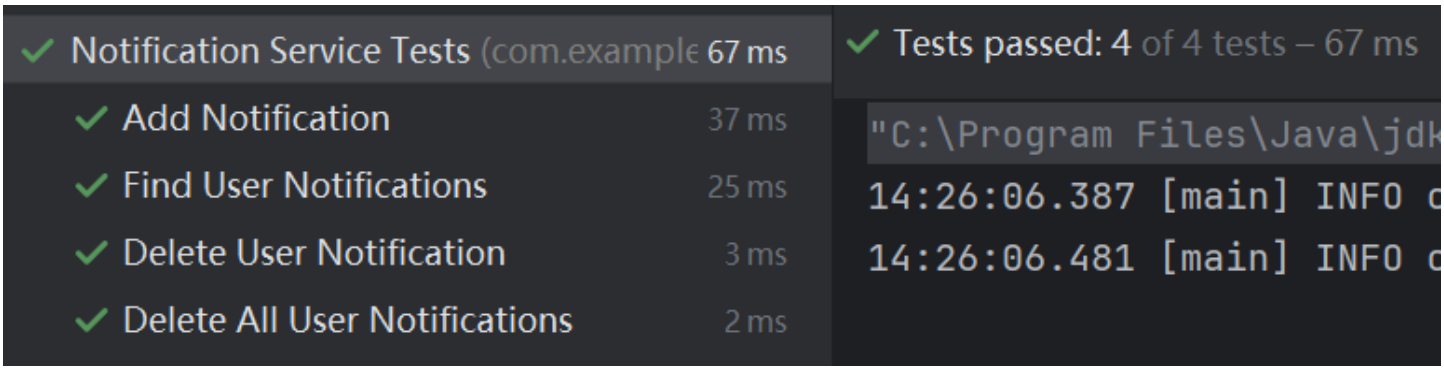
4.4.5 测试通过/失败标识

- 通过： 当调用 `notificationMapper.delete()` 方法的验证成功，意味着所有相关通知记录已被删除。
- 失败： 如果 `notificationMapper.delete()` 方法未被调用或调用条件不满足预期，则测试视为失败。

4.4.6 用例设计

测试项编号	UT_TC_004_004_001	
优先级	高	
测试项描述	验证指定用户的所有通知能够被一次性成功删除。	
前置条件	数据库中存在该用户的多个通知记录。	
用例序号	输入	期望结果额
001	ID = 1;	• 不直接检查返回结果，因为批量删除操作可能不返回具体数据变化，或返回通用的成功指示。 • 确认 `notificationMapper.delete(any(QueryWrapper.class))` 被调用，验证批量删除逻辑已执行，意在清除指定用户的全部通知记录。

4.5 测试结果



5.点赞和收藏系统

5.1 testCreateComment的测试分析与设计

本测试用例旨在验证在正常情况下，创建评论功能能够成功执行，涵盖了评论频率检查、主题和账户信息查询、评论内容的构造、数据库插入操作以及通知服务的调用等关键环节。

5.1.1 标识符定义

UT_TD_005_001

5.1.2 被测特性

1. 评论频率控制： 确保用户在指定周期内的评论次数未超过限制。
2. 主题与账户验证： 验证评论所关联的主题和账户信息能够正确查询。
3. 评论创建逻辑： 包括评论内容的构造、存储逻辑及与之相关的数据库操作。
4. 通知机制： 在评论创建成功后，系统应向相关人员发送通知。

5.1.3 测试方法

- 使用Mockito框架对依赖的服务和数据库操作进行模拟，以控制测试环境并验证预期行为。
- 通过直接调用 `topicService.createComment` 方法，传入用户ID和评论信息，来触发评论创建流程。

5.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_005_001_001	1. 评论频率检查： 模拟频率检查逻辑返回允许创建评论的结果。 2. 数据库查询： 模拟主题和账户查询成功。 3. 评论创建请求： 构建有效的评论创建请求，包含目标主题ID和评论内容。	高

	4. 预期结果验证： 验证评论创建后不返回错误信息，数据库插入操作被正确调用，且至少有一次通知发送尝试。	

5.1.5 测试通过/失败标识

- 通过： `topicService.createComment` 调用后返回 `null` ，表示无错误发生；数据库插入方法和通知服务的调用均被验证。
- 失败： 如果返回结果非 `null` ，数据库插入未被调用，或者通知服务调用验证失败。

5.1.6 用例设计

测试项编号	UT_TC_005_001_001	
优先级	高	
测试项描述	验证在所有前置条件满足的情况下，用户能够成功创建一条评论，包括通过评论频率检查、正确查询到主题和账户信息、成功构造评论内容并插入数据库，以及触发通知服务发送通知。	
前置条件	1. 系统环境： 测试环境配置完毕，数据库服务、流控工具（flowUtils）、主题信息服务（topicMapper）、账户信息服务（accountMapper）、评论数据库操作（topicCommentMapper）以及通知服务（notificationService）均运行正常。 2. 用户权限： 测试用户具备创建评论的权限。 3. 模拟数据： `flowUtils` 的频率检查逻辑已模拟为允许评论，且存在有效的主题ID和账户ID可供查询。	
用例序号	输入	期望结果额

001	UID = 2;
	AddCommentVO{
	TID = 1;
	Content = "Comment Content";
	};
		1. 操作结果： 调用 `topicService.createComment` 方法后，预期返回值为 `null` ，表示没有错误发生。
		2. 数据库验证： 确认数据库中新增了一条与输入相对应的评论记录，表明 `topicCommentMapper.insert` 方法被正确调用并执行了插入操作。
		3. 通知验证： 至少有一次通知发送操作被记录，验证通过 `verify(notificationService, atLeastOnce()).addNotification(...)` ，表明在评论创建成功后，系统按照预期尝试发送了通知给相关用户或管理员。

5.2 testDeleteComment的测试分析与设计

本测试用例旨在验证 `topicService` 中的 `deleteComment` 方法在成功删除评论时的行为，当调用此方法并传入指定的主题ID和评论ID时，预期数据库中的对应评论会被正确删除。

5.2.1 标识符定义

UT_TD_005_002

5.2.2 被测特性

- `topicService.deleteComment` 方法的成功执行路径，确保能够正确删除指定评论。
- `topicCommentMapper.delete` 方法被正确调用并返回预期的结果。

5.2.3 测试方法

- 使用Mockito框架模拟 `topicCommentMapper.delete` 方法的行为，设置其在接收到任何 `QueryWrapper` 参数时返回1，表示删除成功。
- 调用 `topicService.deleteComment` 方法，传入主题ID为1和评论ID为1作为参数。
- 验证 `topicCommentMapper.delete` 方法确实被调用了一次，并且接收到的是任意类型的 `QueryWrapper` 对象。

5.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_005_002_001	1. 模拟行为验证： 确认 `topicCommentMapper.delete` 方法被正确配置为在接收到任何查询包装器时返回删除成功的指示。 2. 功能执行验证： 验证 `topicService.deleteComment` 方法在给定有效参数时，能够触发预期的数据库删除操作。 3. 调用验证： 检查 `topicCommentMapper.delete` 的实际调用情况，确保它按照预期被调用且参数符合要求。	高

5.2.5 测试通过/失败标识

- 通过条件： `topicCommentMapper.delete` 方法被正确调用一次，并且返回值为1，表明删除操作成功执行。
- 失败条件： 如果 `topicCommentMapper.delete` 未被调用，或者调用次数不为1，以及返回值不是预期的1，则测试失败。

5.2.6 用例设计

测试项编号	UT_TC_005_002_001	
优先级	高	
测试项描述	验证成功删除评论的功能。	
前置条件	• `topicCommentMapper` 已被Mockito框架成功模拟。 • 数据库中存在相应主题ID和评论ID对应的记录（此信息在此测试用例中通过模拟实现，实际数据库状态不影响测试结果）。	
用例序号	输入	期望结果额
001	ID = 1; UID = 1;	• `topicCommentMapper.delete` 被调用，且传入了任意 `QueryWrapper` 对象作为参数。 • 方法返回值为1，表示删除操作成功。

5.3 testInteract的测试分析与设计

该测试用例旨在验证 `topicService` 中的 `interact` 方法在成功处理用户交互（如点赞、评论等）时的行为。特别地，当启用某种特定逻辑（本例中由 `true` 控制的逻辑）并传入一个有效的 `Interact` 对象时，确保能够正确与Redis进行交互，这里主要关注 `stringRedisTemplate.opsForHash().put` 方法的调用。

5.3.1 标识符定义

UT_TD_005_003

5.3.2 被测特性

- `topicService.interact` 方法在特定逻辑条件下的执行流程。
- Redis操作，特别是通过 `stringRedisTemplate.opsForHash().put` 方法存储交互信息的能力。

5.3.3 测试方法

- 使用Mockito框架模拟 `stringRedisTemplate.opsForHash()` 的行为，使其返回一个可验证的操作对象（ `hashOperations` ）。
- 调用 `topicService.interact` 方法，传入一个构造好的 `Interact` 对象表示用户对某个主题的交互行为，并设定一个布尔标志为 `true` ，这可能控制了某些特定的业务逻辑。

- 验证 `stringRedisTemplate.opsForHash().put` 方法是否被正确调用，以确认交互信息是否尝试被存储到Redis中。

5.3.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_005_003_001	1. 模拟Redis操作： 确保 `stringRedisTemplate.opsForHash()` 返回一个可验证的对象，用于后续的调用验证。 2. 功能执行验证： 检查在特定逻辑条件下，`topicService.interact` 能否正确触发与Redis的交互操作。 3. Redis操作调用验证： 验证是否尝试将交互数据通过 `put` 方法存入Redis的哈希结构中。	高

5.3.5 测试通过/失败标识

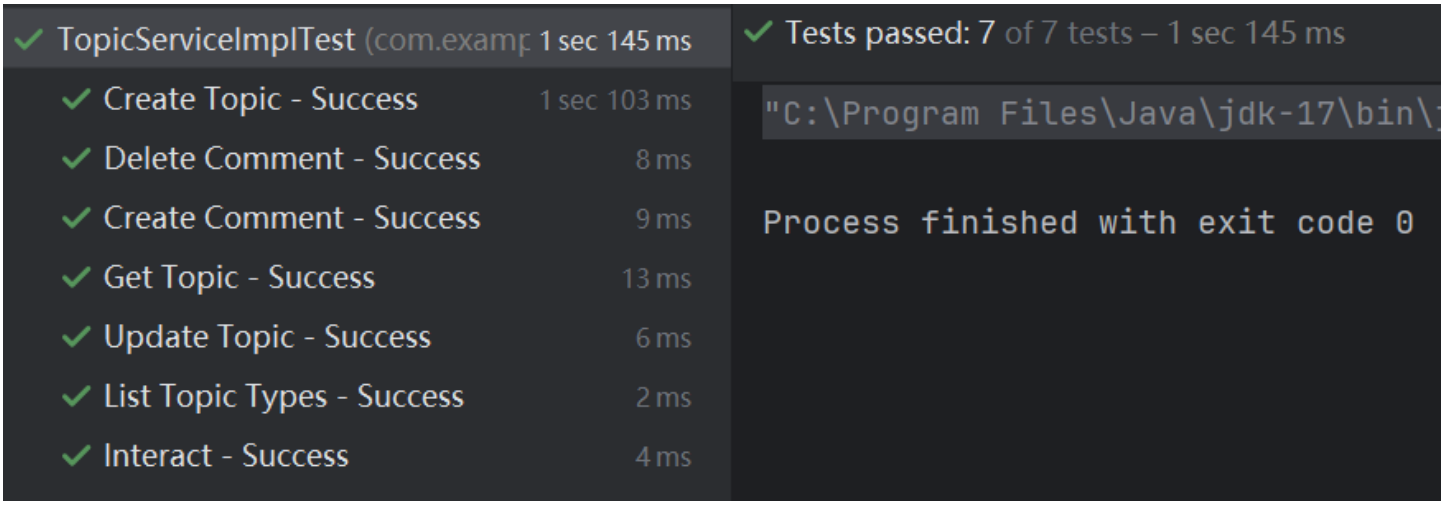
- 通过条件： `stringRedisTemplate.opsForHash().put` 方法被正确调用一次，意味着交互信息尝试被保存。
- 失败条件： 如果该方法没有被调用，或者调用次数不符合预期，则测试视为失败。

5.3.6 用例设计

测试项编号	UT_TC_005_003_001	
优先级	高	
测试项描述	验证成功执行用户交互逻辑并正确与Redis交互。	
前置条件	• `stringRedisTemplate` 已被Mockito框架成功模拟，其 `.opsForHash()` 方法返回一个可验证的 `hashOperations` 对象。 • 准备一个有效的 `Interact` 对象，包含交互的主体ID、目标ID、时间戳及类型（本例为"like"）。	
用例序号	输入	期望结果额
001	Interact{ TID = 1; UID = 1; Date(); Type = "like";	• `stringRedisTemplate.opsForHash().put` 被调用，意在将用户的交互信息（至少包括主体ID、目标ID及类型）存入Redis的哈希结构中，参数分别为任意字符串（可能是键、字段和值），表明交互数据处理逻辑被执行。

	}; State = true;	

5.4 测试结果



6.图片管理系统

标识符	名称	代码行
LLD_006_FUN_001	testFetchImageFromMinio_Success	15
LLD_006_FUN_002	testUploadImage_Success	15
LLD_006_FUN_003	testUploadImage_LimitExceeded	13
LLD_006_FUN_004	testUploadAvatar_Success	19
LLD_006_FUN_005	testUploadAvatar_UpdateFailed	19

6.1 testFetchImageFromMinio的测试分析与设计

该测试用例旨在验证 `imageService.fetchImageFromMinio` 方法在从Minio成功获取图片数据并写入输出流时的行为。通过模拟客户端调用和响应，确保方法能正确处理Minio中的对象读取操作。

6.1.1 标识符定义

UT_TD_006_001

6.1.2 被测特性

- `imageService.fetchImageFromMinio` 方法正确读取Minio中指定图像文件并写入输出流的能力。
- Minio客户端 (`client`) 的 `getObject` 方法调用及其响应处理逻辑。

6.1.3 测试方法

- 使用Mockito框架创建 `OutputStream` 和 `GetObjectResponse` 的模拟对象。
- 配置Minio客户端模拟对象，使其在调用 `getObject` 方法时返回预设的 `GetObjectResponse` 。
- 调用 `imageService.fetchImageFromMinio` 方法，传入模拟的输出流和图片名称。
- 使用Mockito的 `verify` 方法来检查Minio客户端的 `getObject` 方法是否被正确调用。
- 异常处理确保在整个测试过程中没有未预期的异常抛出。

6.1.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_006_001_001	1. Minio客户端行为模拟： 确保当调用 `client.getObject` 方法时，返回一个预设的 `GetObjectResponse` 对象，其中包含了名为"image.jpg"的图像文件的数据。 2. 方法执行与输出验证： 验证 `imageService.fetchImageFromMinio` 方法是否能够正确处理返回的响应，并将图像数据写入到提供的输出流中。 3. 异常监控： 测试过程中应无异常抛出，确保方法执行的健壮性。	高

6.1.5 测试通过/失败标识

- 通过条件： `client.getObject` 方法被正确调用，且整个方法执行过程中没有异常发生，表明图像数据成功从Minio取出并写入输出流。
- 失败条件： 如果 `client.getObject` 没有被调用，或者过程中抛出了任何异常，则测试视为失败。

6.1.6 用例设计

测试项编号	UT_TC_006_001_001

优先级	高	
测试项描述	验证从Minio成功获取图像数据并写入输出流的功能。	
前置条件	• `client` 对象（Minio客户端）已通过Mockito框架进行了模拟设置。 • 创建了一个模拟的 `OutputStream` 用于接收图像数据。 • 准备了一个 `GetObjectResponse` 对象，模拟了从Minio获取到的图像文件响应。	
用例序号	输入	期望结果额
001	Object = "image.jpg";	• `client.getObject` 被正确调用，表明尝试从Minio获取指定图像文件。 • 图像数据（"image data"的字节形式）应被写入到提供的输出流中，尽管直接验证这一点在当前测试代码中未直接体现，但通过方法调用的验证间接推断。

6.2 testUploadImage的测试分析与设计

这两个测试用例分别针对 `imageService.uploadImage` 方法的两种不同场景进行验证：“上传成功”和“超出限制”。

1. Upload Image - Success: 验证在一切正常的情况下，即流量限制检查通过并且数据库插入操作成功，上传图片功能能否正确执行。
2. Upload Image - Limit Exceeded: 验证当图片上传请求因流量限制检查未通过时，服务是否能够正确阻止图片上传，并不执行进一步的存储操作。

6.2.1 标识符定义

UT_TD_006_002

6.2.2 被测特性

- 上传成功场景:
  - 流量限制检查逻辑的有效性。
  - 图片存储至数据库及对象存储（如Minio）的成功处理。
- 超出限制场景:
  - 流量限制逻辑的执行，确保在限制条件下阻止上传。
  - 在限制条件下不执行数据库插入及对象存储操作。

6.2.3 测试方法

- 使用Mockito对 `flowUtils.limitPeriodCounterCheck` 和 `imageStoreMapper.insert` 方法进行模拟，控制它们的返回值或行为。
- 对于 `client.putObject` 也进行模拟验证或检查调用情况，根据测试场景需要。
- 调用 `imageService.uploadImage` 方法并检查返回结果及各依赖方法的调用情况。

6.2.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_006_002_001	• 流量限制检查失败（返回 `false`）。 • 预期结果：方法返回 `null`，且 `client.putObject` 未被调用，表示未尝试向对象存储上传图片。	低
UT_TC_006_002_002	• 确保流量限制检查通过（返回 `true`）。 • 数据库插入操作模拟返回1，代表成功。 • 预期结果：方法返回非空字符串（通常为图片的存储路径或标识），并且 `client.putObject` 及 `imageStoreMapper.insert` 被调用。	高

6.2.5 测试通过/失败标识

- Upload Image - Success:
  - 通过条件: 返回结果非空， `flowUtils.limitPeriodCounterCheck` 和 `imageStoreMapper.insert` 被正确调用， `client.putObject` 也被调用。
  - 失败条件: 返回结果为空，或相关方法调用情况不符合预期。
- Upload Image - Limit Exceeded:
  - 通过条件: 返回结果为 `null` ， `client.putObject` 从未被调用。
  - 失败条件: 返回非空结果，或 `client.putObject` 被错误地调用了。

6.2.6 用例设计

测试项编号	UT_TC_006_002_001	
优先级	低	
测试项描述	验证图片上传功能在流量限制超时时的处理逻辑。	
前置条件	流量限制检查故意设置为失败。	
用例序号	输入	期望结果额

001	ID = 1; mockFile;	返回 `null`，且不触发对象存储上传操作，证明上传请求被合理拒绝。

测试项编号	UT_TC_006_002_002	
优先级	高	
测试项描述	验证图片上传功能在所有前置条件满足时的成功执行。	
前置条件	流量限制检查通过，数据库插入操作预期成功。	
用例序号	输入	期望结果额
001	ID = 1; mockFile;	返回非空字符串，代表上传成功。

6.3 testUploadAvatar的测试分析与设计

这两个测试用例旨在验证 `imageService.uploadAvatar` 方法在上传用户头像时的两种不同场景：上传成功与更新用户信息失败。

1. Upload Avatar - Success: 验证在上传新头像成功后，旧头像被移除，用户账户信息中的头像路径得到更新，且新头像被成功存储至对象存储服务。
2. Upload Avatar - Update Failed: 验证当尝试更新用户账户信息失败时，虽然新头像被上传至对象存储服务，但旧头像保持不变，且最终返回 `null` 表示上传失败。

6.3.1 标识符定义

UT_TD_006_003

6.3.2 被测特性

- 上传成功场景:
  - 用户账户信息的正确更新，包括旧头像的移除与新头像路径的设置。
  - 新头像文件上传至对象存储服务。
  - 旧头像从对象存储服务中移除。
- 更新失败场景:
  - 在用户账户信息更新失败时，确保旧头像不被删除。
  - 尽管上传操作执行，但由于更新失败，最终结果应为失败标志（返回 `null`）。

6.3.3 测试方法

- 使用Mockito模拟 `accountMapper.selectById` 和 `accountMapper.update` 方法，控制它们的返回值。
- 模拟对象存储服务客户端(`client`)的 `putObject` 和 `removeObject` 方法的调用情况。
- 调用 `imageService.uploadAvatar` 方法并检查返回结果及各依赖方法的调用情况。

6.3.4 测试项标识

测试项标识符	测试项描述	优先级
UT_TC_006_003_001	• 同样确保能获取用户账户信息，但模拟更新操作失败（返回0）。 • 预期结果：方法返回 `null` ， `client.putObject` 被调用以上传新头像，但 `client.removeObject` 不应被调用，以防止在用户信息未更新情况下删除旧头像。	低
UT_TC_006_003_002	• 确保能正确从数据库获取用户账户信息。 • 更新账户信息模拟成功（返回1）。 • 预期结果：方法返回非空字符串（新头像的存储路径），且 `client.putObject` 、 `client.removeObject` 及 `accountMapper.update` 均被正确调用。	高

6.3.5 测试通过/失败标识

- Upload Avatar - Success:
  - 通过条件: 返回结果非空，所有预期的数据库更新和对象存储操作均被调用。
  - 失败条件: 返回结果为空，或任何预期的操作未被正确执行。
- Upload Avatar - Update Failed:
  - 通过条件: 返回结果为 `null` ，仅 `client.putObject` 被调用，而 `client.removeObject` 未被调用。
  - 失败条件: 返回非空结果，或 `client.removeObject` 被错误地调用。

6.3.6 用例设计

测试项编号	UT_TC_006_003_001
优先级	低
测试项描述	验证在更新用户信息步骤失败时，上传头像操作的回退逻辑

前置条件	能够从数据库获取到用户信息，预期更新操作失败。	
用例序号	输入	期望结果额
001	ID = 1; mockFile;	返回null，且不删除旧头像，确保数据一致性。

测试项编号	UT_TC_006_003_002	
优先级	高	
测试项描述	验证用户头像上传并在所有步骤成功执行时的功能完整性。	
前置条件	能够从数据库获取到用户信息，预期更新操作成功。	
用例序号	输入	期望结果额
001	ID = 1; mockFile;	返回新头像的存储路径，表明上传及更新流程成功

6.4 测试结果