TDD开发演示案例

小型电商系统案例

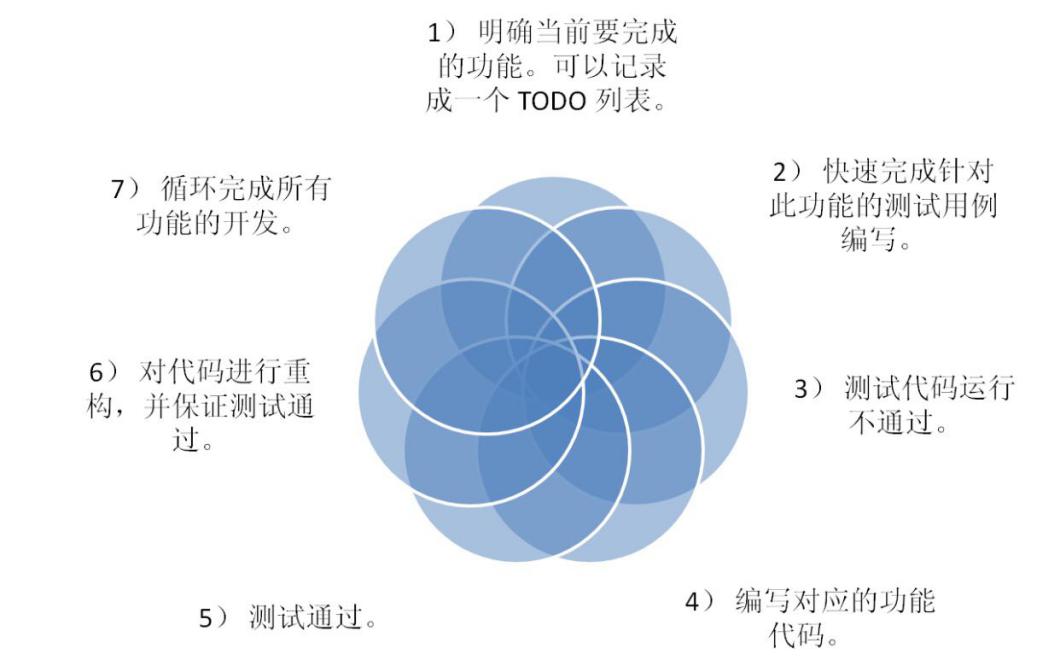
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档版本 | 版本说明 | 更新日期 | 作者 | 备注 |
| 1.0 | 创建 | 2018-12-30 | 方勇 |  |
|  |  |  |  |  |

# 概述

## 简介

通过一个小型的电商系统，来演示TDD开发过程

## TDD开发流程



## 技术环境

IDEA

SpringBoot 2.1.1.RELEASE

Junit4.12

myBatis

## 案例源码

[https://github.com/fangyong2/jvm](https://gitee.com/sujianfeng/lab-tdd)

# 功能需求

## 用户模块

用户信息：用户名、密码、年龄、累计充值金额、累计使用金额、当前余额、消费积分

模块功能：新增用户、查询用户

## 充值模块

用户充值

## 消费模块

用户购买

## 功能说明

积分规则：

当用户累计消费100元以下，每元1累积积分

当用户累计消费101元到1000元，每1元累计2积分

当用户累计消费1001元以上，每1元累计3积分

# 开发设计

## 技术与规范

RESTful规范

三层体系：controller、service、dao层

持久化框架：myBatis

数据库：mySql

## 数据库设计

用户表、充值记录表、消费记录表

## 模块与类定义

|  |  |  |  |
| --- | --- | --- | --- |
| 模块 | 表示层 | 逻辑层 | 持久层 |
| 用户模块 | UserController | UserService | UserDao |
| 充值模块 | RechargeController | RechargeService | RechargeDao |
| 消费模块 | ConsumeController | ConsumeService | ConsumeDao |
| 积分模块 | 无 | ScoreService | ScoreDao |

## 实体类定义

### 用户：LabUser

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 中文名称 | 字段类型 | 备注 |
| id | 主键 | int |  |
| username | 用户名 | varchar(20) |  |
| password | 密码 | varchar(20) |  |
| age | 年龄 | int |  |
| totalAddMoney | 累计充值金额 | int |  |
| totalUseMoney | 累计消费金额 | int |  |
| remainMoney | 可用余额 | int |  |
| score | 消费积分 | int |  |
| createTime | 创建时间 | timestamp |  |
| updateTime | 更新时间 | timestamp |  |
| remark | 备注 | Varchar(500) |  |

### 充值记录：LabAddMoney

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 中文名称 | 字段类型 | 备注 |
| int | 主键 | int |  |
| userId | 用户id | varchar(20) |  |
| addMoney | 充值金额 | int |  |
| createTime | 创建时间 | timestamp |  |
| remark | 备注 | Varchar(500) |  |

### 购买记录：LabBuyGood

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 中文名称 | 字段类型 | 备注 |
| int | 主键 | int |  |
| userId | 用户id | varchar(20) |  |
| goodName | 购买商品 | varchar(20) |  |
| useMoney | 消费金额 | int |  |
| createTime | 创建时间 | timestamp |  |
| remark | 备注 | Varchar(500) |  |

## 接口方法定义

### 新增用户

|  |  |
| --- | --- |
| RESTful资源 | /user |
| http方法 | POST |
| 类名 | UserController |
| 方法名 | addUser |
| 输入 | LabUser: 用户对象 |
| 输出 | {  success: 是否成功,  message: 操作信息  } |
| 功能描述 | 新增一个用户 |

### 查询用户

|  |  |
| --- | --- |
| RESTful资源 | /user/{id} |
| http方法 | GET |
| 类名 | UserController |
| 方法名 | getUser |
| 输入 | id：用户id |
| 输出 | {  success: 是否成功,  message: 操作信息,  data: {  id: 用户id,  userName: 用户名称  }  } |
| 功能描述 | 获取一个用户信息 |

### 充值接口

|  |  |
| --- | --- |
| RESTful资源 | /recharge/addMoney |
| http方法 | POST |
| 类名 | RechargeController |
| 方法名 | addMoney |
| 输入 | id：用户id  addMoney: 充值金额 |
| 输出 | {  success: 是否成功,  message: 操作信息  } |
| 功能描述 | 充值 |

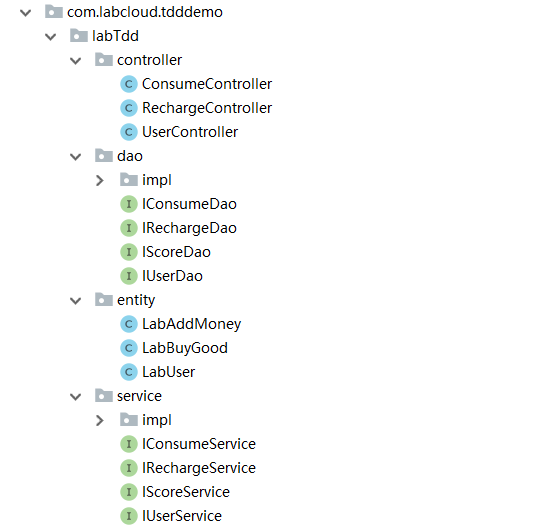
### 消费接口

|  |  |
| --- | --- |
| RESTful资源 | /consume/buyGood |
| http方法 | POST |
| 类名 | ConsumeController |
| 方法名 | buyGood |
| 输入 | id：用户id  goodName: 商品名称  useMoney: 消费金额 |
| 输出 | {  success: 是否成功,  message: 操作信息  } |
| 功能描述 | 购买一个指定商品 |

# TDD开发过程

## 利用模板自动生成测试和生成代码框架

开发标准化后，完全可以使用模板化生成框架代码



## Controller测试代码编写

先准备好一个ControllerTestBase基础类，主要是使用mockMvc方式模拟好web请求，并将rest请求和返回的结果检查统一写好，方便调用，可用通过源码网址（[https://github.com/fangyong2/jvm](https://gitee.com/sujianfeng/lab-tdd)

）获取代码。

从这里大家可用看出，编写测试代码只要根据开发设计好的文档进行编写，基本上不怎么思考，从这里我们应该可用体会到，TDD是以需求为中心进行编写单元测试代码。

一个模块对应单元测试类，一个接口对应一个单元测试方法：

### UserControllerKotlinTest 用户测试

|  |
| --- |
| **public class** UserControllerTest **extends** ControllerTestBase {   @Override  **public void** beforeTest() {  *//测试前环境准备  //初始化session等* }   @Override  **public void** afterTest() {  *//测试后恢复现场* }   @Test  **public void** addUser() **throws** Exception {  post(**"/user"**,  **new** HashMap<String, String>(){{put(**"username"**, **"张三"**); put(**"password"**, **"123"**);}},  **new** HashMap<String, Object>(){{put(**"$.success"**, **true**); put(**"$.message"**, **"新增成功!"**);}}  );  }   @Test  **public void** getUser() **throws** Exception {  get(**"/getUser"**,  **new** HashMap<String, String>(){{put(**"id"**, **"1"**);}},  **new** HashMap<String, Object>(){{put(**"$.success"**, **true**);}}  );  } } |

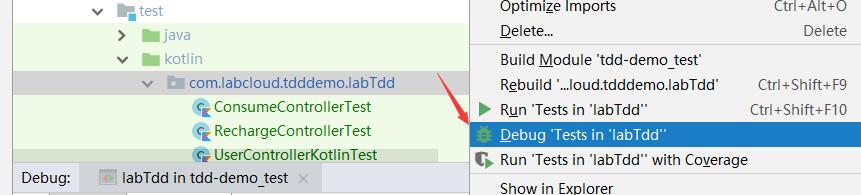
### RechargeControllerTest 充值测试

|  |
| --- |
| **public class** RechargeControllerTest **extends** ControllerTestBase {  @Override  **public void** beforeTest() {  *//测试前环境准备  //初始化session等* }   @Override  **public void** afterTest() {  *//测试后恢复现场* }   */\*\*  \* 充值  \** ***@throws*** *Exception  \*/* @Test  **public void** addMoney() **throws** Exception {  post(**"/addMoney"**,  **new** HashMap<String, String>(){{put(**"id"**, **"1"**); put(**"addMoney"**, **"1000"**);}},  **new** HashMap<String, Object>(){{put(**"$.success"**, **true**);}}  );  } } |

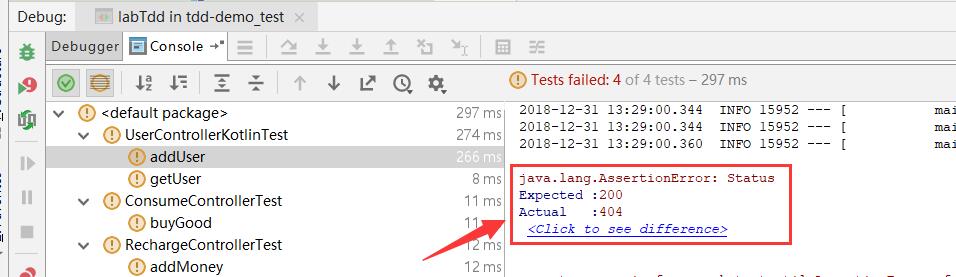
### ConsumeControllerTest 消费测试

|  |
| --- |
| **public class** ConsumeControllerTest **extends** ControllerTestBase {  @Override  **public void** beforeTest() {  *//测试前环境准备  //初始化session等* }   @Override  **public void** afterTest() {  *//测试后恢复现场* }   */\*\*  \* 消费测试  \** ***@throws*** *Exception  \*/* @Test  **public void** buyGood() **throws** Exception {  post(**"/buyGood"**,  **new** HashMap<String, String>(){{put(**"id"**, **"1"**); put(**"useMoney"**, **"1000"**); put(**"goodName"**, **"apple"**);}},  **new** HashMap<String, Object>(){{put(**"$.success"**, **true**);}}  );  } } |

### 运行单元测试



几个接口测试都失败了，很显然这是正确的反馈，我们的产品代码都还没写，接下来开始准备编写产品代码。



## Controller产品代码编写

### UserController 用户

TDD就是先写测试代码后写产品代码，那么我们就可用根据测试代码编写产品代码了。

注意：这里只是简单写了下Controller的产品代码，目前是为了先让他测试通过（当然也可以先不管它测试是否通过，后面再测试也行，看个人习惯），后面再补充业务代码。这样写的意义在于：保证接口的输入输出是符合功能需求的。

一个单位测试类对应一个产品代码类，一个测试方法对应一个产品代码方法：

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-30 19:07  \*/* @RestController **public class** UserController {   @PostMapping(**"/user"**)  **public** Map<String, Object> addUser(LabUser labUser){  Map<String, Object> result = **new** HashMap<>();   result.put(**"success"**, **true**);  result.put(**"message"**, **"新增成功!"**);  **return** result;  }   @GetMapping(**"/getUser"**)  **public** Map<String, Object> getUser(**int** id){  Map<String, Object> result = **new** HashMap<>();   result.put(**"success"**, **true**);  **return** result;  } |

### RechargeController 充值

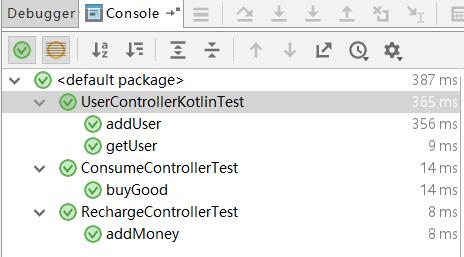
|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 13:33  \*/*  @RestController **public class** RechargeController {   @PostMapping(**"/addMoney"**)  **public** Map<String, Object> addMoney(**int** id, **int** addMoney){  Map<String, Object> result = **new** HashMap<>();    result.put(**"success"**, **true**);  result.put(**"message"**, **"充值成功!"**);  **return** result;  } } |

### ConsumeController消费

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 13:33  \*/*  @RestController **public class** ConsumeController {   @PostMapping(**"/buyGood"**)  **public** Map<String, Object> buyGood(**int** id, **int** useMoney, String goodName){  Map<String, Object> result = **new** HashMap<>();   result.put(**"success"**, **true**);  result.put(**"message"**, **"消费成功!"**);  **return** result;  } } |

### 运行UserControllerKotlinTest

从上面代码看出，我们看出，只是写了接口的基本实现，写完，我们再来测试下：



Ok，那么我们基本实现了controller层的代码编写，并且通过单元测试保证了接口的基本功能是没有问题的，那么接下来我们可用进行service层代码的编写。

## Servcice层测试代码编写

### UserServiceTest用户测试

**由于UserService依赖UserDao，需要使用mock把UserDao隔离出去：**

**userDao** = *mock*(IUserDao.**class**);

**当访问userDao时，根据模拟返回值：**

*when*(**userDao**.addUser(**opResult**, labUser)).thenReturn(1);

*when*(**userDao**.getUser(**opResult**, 1)).thenReturn(labUserReturn);

**使用反射工具类，将userDao注入到userService的私有属性中:**

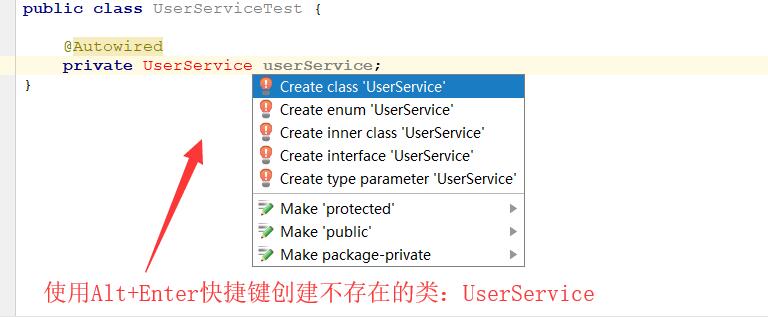
ReflectionTddUtils.*setFieldValue*(**userService**, **"userDao"**, **userDao**);

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 14:16  \*/*  **public class** UserServiceTest {   **private static** UserService *userService*;  **private static** RestResult *opResult* = RestResult.*create*();  **private static** IUserDao *userDao*;  @BeforeClass  **public void** beforeTest() **throws** NoSuchFieldException, IllegalAccessException {  **userDao** = *mock*(IUserDao.**class**);  **userService** = **new** UserService();  ReflectionTddUtils.*setFieldValue*(**userService**, **"userDao"**, **userDao**);  }   */\*\*  \* 创建用户测试  \*/* @Test  **public void** addUser(){  LabUser labUser = **new** LabUser();  labUser.setUsername(**"张三"**);  labUser.setAge(12);  *when*(**userDao**.addUser(**opResult**, labUser)).thenReturn(1);  **int** update = **userService**.addUser(**opResult**, labUser);  *assertThat*(update, *equalTo*(1));  }   */\*\*  \* 查询用户测试  \*/* @Test  **public void** getUser(){  LabUser labUserReturn = **new** LabUser();  labUserReturn.setId(1);  *when*(**userDao**.getUser(**opResult**, 1)).thenReturn(labUserReturn);  LabUser labUser = **userService**.getUser(**opResult**, 1);  *assertThat*(labUser.getId(), *equalTo*(1));  } } |
|  |

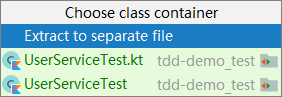
### UserService产品代码编写

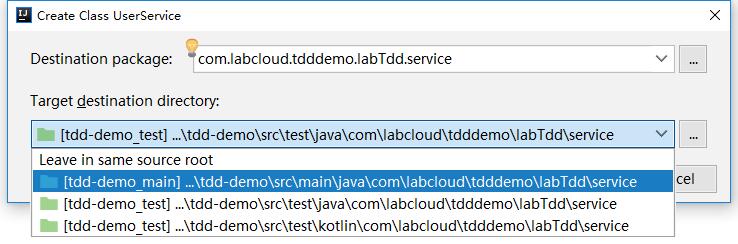
**快速创建一个空的UserService代码方法：**

Service层单元测试，需要先把产品的service注入进来，那么先写好类名，然后使用alt+enter快捷键创建类：



选择类所在jar包位置





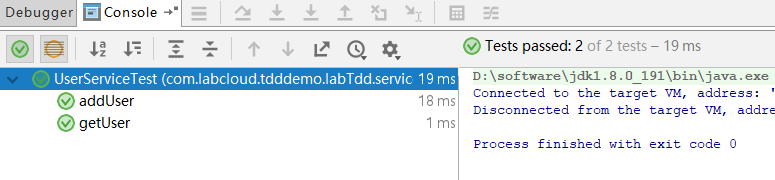
创建好一个空的UserService类

|  |
| --- |
| **public class** UserService { } |

然后用同样的方法，补齐UserService的代码：

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 13:58  \*/*  @Service **public class** UserService **implements** IUserService {   @Autowired  **private** IUserDao **userDao**;   @Override  **public int** addUser(RestResult opResult, LabUser labUser) {  **return userDao**.addUser(opResult, labUser);  }   @Override  **public** LabUser getUser(RestResult opResult, **int** id) {  **return userDao**.getUser(opResult, id);  } } |

### 运行UserServiceTest



### RechargeServiceTest充值测试代码

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 20:56  \*/* **public class** RechargeServiceTest {   **private static** IRechargeService *rechargeService*;  **private static** IRechargeDao *rechargeDao*;  **private static** RestResult *opResult* = RestResult.*create*();   @BeforeClass  **public static void** beforeTest() **throws** NoSuchFieldException, IllegalAccessException {  *rechargeDao* = *mock*(IRechargeDao.**class**);  *rechargeService* = **new** RechargeService();  ReflectionTddUtils.*setFieldValue*(*rechargeService*, **"rechargeDao"**, *rechargeDao*);  }   @Test  **public void** addMoney(){  *when*(*rechargeDao*.addMoney(*opResult*, 1, 1000)).thenReturn(1);  **int** update = *rechargeService*.addMoney(*opResult*, 1, 1000);  *assertThat*(update, *equalTo*(1));  } } |

### RechargeService充值产品代码

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 17:18  \*/* @Service **public class** RechargeService **implements** IRechargeService {   @Autowired  **private** IRechargeDao **rechargeDao**;   @Override  **public int** addMoney(RestResult opResult, **int** id, **int** addMoney) {  **return rechargeDao**.addMoney(opResult, id, addMoney);  } } |

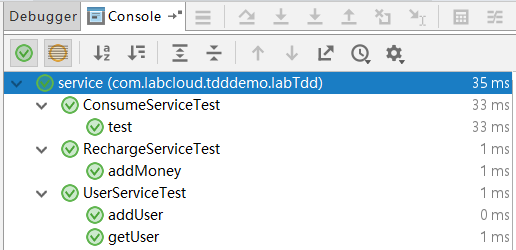
### ConsumeServiceTest消费测试代码

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 21:03  \*/* **public class** ConsumeServiceTest {   **private static** IConsumeService *consumeService*;  **private static** IConsumeDao *consumeDao*;  **private static** RestResult *restResult* = RestResult.*create*();    @BeforeClass  **public static void** beforeClass() **throws** NoSuchFieldException, IllegalAccessException {  *consumeService* = **new** ConsumeService();  *consumeDao* = *mock*(IConsumeDao.**class**);  ReflectionTddUtils.*setFieldValue*(*consumeService*, **"consumeDao"**, *consumeDao*);  }   @Test  **public void** test(){   *when*(*consumeDao*.buyGood(*restResult*, 1, 3000, **"apple"**)).thenReturn(1);  **int** update = *consumeService*.buyGood(*restResult*, 1, 3000, **"apple"**);  *assertThat*(update, *equalTo*(1));  } } |

### ConsumeService消费产品代码

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 17:19  \*/* @Service **public class** ConsumeService **implements** IConsumeService {   @Autowired  **private** IConsumeDao **consumeDao**;   @Override  **public int** buyGood(RestResult opResult, **int** id, **int** useMoney, String goodName) {  **return consumeDao**.buyGood(opResult, id, useMoney, goodName);  } } |

### 全部测试service的测试代码



## Dao层测试代码编写

Dao层测试由于依赖*sqlSessionTemplate* ，那么也需要把*sqlSessionTemplate* 进行隔离：

*sqlSessionTemplate* = *mock*(SqlSessionTemplate.**class**);

### UserDaoTest 用户dao测试

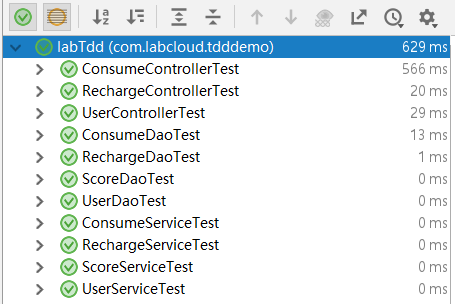
|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 21:14  \*/* **public class** UserDaoTest {   **private static** SqlSessionTemplate *sqlSessionTemplate*;  **private static** IUserDao *userDao*;  **private static** RestResult *restResult* = RestResult.*create*();   @BeforeClass  **public static void** beforeClass() **throws** NoSuchFieldException, IllegalAccessException {  *sqlSessionTemplate* = *mock*(SqlSessionTemplate.**class**);  *userDao* = **new** UserDao();  ReflectionTddUtils.*setFieldValue*(*userDao*, **"sqlSessionTemplate"**, *sqlSessionTemplate*);  }   @Test  **public void** space(){  *assertThat*(*userDao*.space(), *equalTo*(**"UserDao"**));  }   @Test  **public void** addUser(){  LabUser labUser = **new** LabUser();  labUser.setId(1);  labUser.setUsername(**"张三"**);  *when*(*sqlSessionTemplate*.insert(*userDao*.space() + **".insertUser"**, labUser)).thenReturn(1);  **int** update = *userDao*.addUser(*restResult*, labUser);  *assertThat*(update, *equalTo*(1));  }   @Test  **public void** getUser(){  Map<Object, Object> params = **new** HashMap<>();  params.put(**"condition"**, **" and id = 1"**);  LabUser labUser = **new** LabUser();  labUser.setId(1);  labUser.setUsername(**"张三"**);  List<Object> rows = **new** ArrayList<>();  rows.add(labUser);  *when*(*sqlSessionTemplate*.selectList(*userDao*.space() + **".queryLabUsers"**, params)).thenReturn(rows);  LabUser labUserReturn = *userDao*.getUser(*restResult*, 1);  *assertThat*(labUserReturn.getId(), *equalTo*(labUser.getId()));  }    @Test  **public void** queryLabUsers(){  Map<Object, Object> params = **new** HashMap<>();  params.put(**"condition"**, **" and id = 1"**);  LabUser labUser = **new** LabUser();  labUser.setId(1);  labUser.setUsername(**"张三"**);  List<Object> rows = **new** ArrayList<>();  rows.add(labUser);  *when*(*sqlSessionTemplate*.selectList(*userDao*.space() + **".queryLabUsers"**, params)).thenReturn(rows);  List<LabUser> rowsReturn = *userDao*.queryLabUsers(*restResult*, **" and id = 1"**);  *assertThat*(rowsReturn.get(0).getId(), *equalTo*(labUser.getId()));  } } |

### UserDao用户dao产品代码

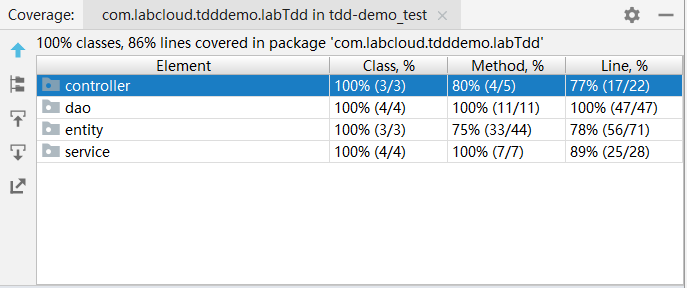
|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 17:19  \*/* @Repository **public class** UserDao **implements** IUserDao {   @Autowired  **private** SqlSessionTemplate **sqlSessionTemplate**;  @Override  **public** String space(){  **return** StringUtilsEx.*rightStr*(**this**.getClass().getName(), **"."**);  }   */\*\*  \* 新增用户  \** ***@param opResult*** *\** ***@param labUser*** *\** ***@return*** *\*/* @Override  **public int** addUser(RestResult opResult, LabUser labUser) {  **return sqlSessionTemplate**.insert(space() + **".insertUser"**, labUser);  }   */\*\*  \* 查询用户  \** ***@param opResult*** *\** ***@param id*** *\** ***@return*** *\*/* @Override  **public** LabUser getUser(RestResult opResult, **int** id) {  List<LabUser> rows = queryLabUsers(opResult, String.*format*(**" and id = %s"**, id));  **return** rows.size() > 0 ? rows.get(0) : **null**;  }   */\*\*  \* 根据条件查询多个用户信息  \** ***@param condition*** *\** ***@return*** *\*/* **public** List<LabUser> queryLabUsers(RestResult restResult, String condition){  Map<Object, Object> params = **new** HashMap<>();  params.put(**"condition"**, condition);  List<LabUser> rows = **sqlSessionTemplate**.selectList(space() + **".queryLabUsers"**, params);  **return** rows;  }   */\*\*  \* 用户余额变更  \** ***@param restResult*** *\** ***@param userId*** *\** ***@param money*** *\*/* **public int** userMoneyUpdate(RestResult restResult, **int** userId, **int** money){  Map<String, Object> params = **new** HashMap<>();  params.put(**"userId"**, userId);  params.put(**"money"**, money);  params.put(**"totalAddMoney"**, money > 0 ? money : 0);  params.put(**"totalUseMoney"**, money < 0 ? money : 0);  **int** update = **sqlSessionTemplate**.update(space() + **".updateUserMoney"**, params);  **if** (update == 0){  LabUser labUser = **new** LabUser();  labUser.setId(userId);  labUser.setTotalAddMoney(0);  labUser.setTotalAddMoney(0);  labUser.setScore(0);  labUser.setRemainMoney(money);  update = addUser(restResult, labUser);  }  **return** update;  } } |

其他三个dao代码类似，就不列出来了，完整代码在此（[https://github.com/fangyong2/jvm](https://gitee.com/sujianfeng/lab-tdd)）

## 全部单元测试效果



## 查看测试覆盖率



# 集成测试

集成测试整个系统所有模块联合起来进行测试，集成测试一定是要在单元测试全部通过后才进行测试。

|  |
| --- |
| */\*\*  \** ***@Author*** *方勇  \** ***@create*** *2018-12-31 23:52  \* 集成测试  \*/* @Transactional **public class** labTddIntegrationTest **extends** ControllerTestBase {    @Autowired  **private** IUserService **userService**;  @Autowired  **private** IConsumeService **consumeService**;  @Autowired  **private** IRechargeService **rechargeService**;  @Autowired  **private** IScoreService **scoreService**;   **private** LabUser **labUser**;    @Override  **public void** beforeTest() {  RestResult restResult = RestResult.*create*();  *//新增一个用户* **userService**.addUser(restResult, **new** LabUser(0, **"张三"**));  *//取出这个用户数据* List<LabUser> labUsers = **userService**.queryLabUsers(restResult, **""**);  **labUser** = labUsers.get(0);  }   **private void** assertUserInfo(LabUser labUserTmp, **int** totalAddMoney, **int** totalUseMoney, **int** remainMoney, **int** scoreMoney){  *assertThat*(**"用户累计充值金额存储错误!"**, labUserTmp.getTotalAddMoney(), *equalTo*(totalAddMoney));  *assertThat*(**"用户累计消费金额存储错误!"**, labUserTmp.getTotalUseMoney(), *equalTo*(totalUseMoney));  *assertThat*(**"用户可用金额存储错误!"**, labUserTmp.getRemainMoney(), *equalTo*(remainMoney));  *assertThat*(**"用户可用积分存储错误!"**, labUserTmp.getScore(), *equalTo*(scoreMoney));  }   @Override  **public void** afterTest() {  }   @Test  **public void** test(){  LabUser labUserTmp = **null**;  RestResult restResult = RestResult.*create*();   *//充值1000* **rechargeService**.addMoney(restResult, **labUser**.getId(), 1000);  labUserTmp = **userService**.getUser(restResult, **labUser**.getId());  assertUserInfo(labUserTmp, 1000, 0, 1000, 0);   *//再充值12000* **rechargeService**.addMoney(restResult, **labUser**.getId(), 12000);  labUserTmp = **userService**.getUser(restResult, **labUser**.getId());  assertUserInfo(labUserTmp, 13000, 0, 13000, 0);   *//消费5500* **consumeService**.buyGood(restResult, **labUser**.getId(), 5500, **"apple"**);  labUserTmp = **userService**.getUser(restResult, **labUser**.getId());  assertUserInfo(labUserTmp, 13000, 5500, 13000 - 5500, 100 + 900 \* 2 + (5500 - 1000) \* 3);   } } |