Table of Contents

金融项目课程

本阶段是一个项目实战课程。测试项目是一个基于Java开发的金融类项目。在本阶段中，将重点对该项目进行系统测试和接口测试。

课程大纲

序号	章节	知识点
1	第一章 项目准备	1. 项目测试安排 2. 金融项目专业术语 3. 熟悉项目 4. 回顾项目测试流程
2	第二章 系统测试分析与设计	1. 项目需求评审 2. 项目测试计划 3. 系统测试用例的设计
3	第三章 接口测试的分析和设计	1. 接口测试的应用场景 2. 项目接口的特殊点 3. 编写接口测试用例
4	第四章 接口测试的执行	1. 接口测试准备—环境准备 2. 接口测试准备—数据准备 3. 手工执行接口测试 4. 编写自动化接口测试脚本 5. 测试数据清理 6. 执行自动化接口测试 7. 接口加解密
5	第五章 系统测试的执行和测试报告	1. 执行测试用例并提交缺陷 2. BUG定位 3. 软件测试报告
4	第六章 项目总结	1. 金融项目测试实战总结

课程目标

- 能够独立完成项目测试的整个流程
- 金融项目够完整介绍金融项目的测试经验

课程实施方式

- 以同学们自己实战为主，以小组合作形式展开测试
- 每组选举一个组长作为项目测试的负责人

项目准备

目标

1. 熟悉金融项目，为后续进行接口测试和系统测试做准备

项目测试安排

目标

1. 能够知道本阶段课程的整体实施安排
2. 同学们商议完成分组

1. 项目测试实战说明

- 同学们先完成分组，并选举测试实战的组长
- 组长负责安排测试计划（可与组员协商制定），关注测试进度，协调组内资源，确保测试目标按时达成
- 组员按照测试计划开展具体测试工作
- 组员每日需提交自己的工作进度日报，并说明工作中遇到的问题和困难
- 组长每日需提交团队整体的工作进度日报，并说明当前的问题和应对措施

2. 学生分组

分组建议

- 组员之间能够进行有效沟通
- 每个小组同学的工作能力分布均匀，强弱结合
- 组员之间能够方便交流（座位相近，或者调换座位）

选举组长

- 组内各位同学都有机会
- 同学们也可以选择之前的组长保持不变
- 组长要具备一定组织能力，同时也要有较高的测试水平

金融项目专业术语

目标

1．理解金融行业的专业术语

1. 金融项目专业术语

投资专业术语

- 债权人：指通过国家银行、合法金融机构等平台提供货币资金的企业或者个人。
- 借款人：指在信贷活动中以自身的信用或财产作保证，或者以第三者作为担保而从贷款人处借得货币资金的企事业单位或个人。
- 投资：指国家或企业以及个人，为了特定目的，与对方签订协议，促进社会发展，实现互惠互利，输送资金的过程。
- 投标：是一个投标招标的专业术语，是指投标人(卖方) 应招标人的邀请，根据招标通告或招标单所规定的条件，在规定的期限内，向招标人递盘的行为。在本项目中所指即发起的借贷关系的各种类型项目。
- 债权转让：债权人通过协议而将其债权全部或部分转移于第三人的行为。

本息专业术语

- 本金：贷款、存款或投资在计算利息之前的原始金额。
- 利息：借款人(债务人)因使用借入货币或资本而支付给贷款人(债权人)的报酬。
- 利率：利息率的简称，就是指一定期限内利息额与存款本金或贷款本金的比率。通常分为年利率、月利率和日利率三种。

还款专业术语

- 等额本息：指一种贷款的还款方式，指在还款期内，每月偿还同等数额的贷款(包括本金和利息)。
- 等额本金：指一种贷款的还款方式，是在还款期内把贷款数总额等分，每月偿还同等数额的本金和剩余贷款在该月所产生的利息。
- 提前还款：提前还款是指借款方在还款期未到之前即先行偿还贷款的行为。提前还款包括提前全部还款、提前部分还款且贷款期限不变、提前部分还款的同时缩短贷款期限三种情况。

熟悉项目

目标

1. 能够介绍出黑马安享智慧理财项目

1. 项目简介

业务特性

- 黑马安享智慧理财项目是一个P2P的金融平台，P2P金融又叫P2P信贷，P2P是 peer-to-peer 或 person-to-person 的简写，意思是个人对个人， 本项目采用国家政策允许的银行存管模式，为用户提供方便、快捷、 安心的P2P金融服务，可以为用户提供借款和投资功能的理财服务。

角色和用户

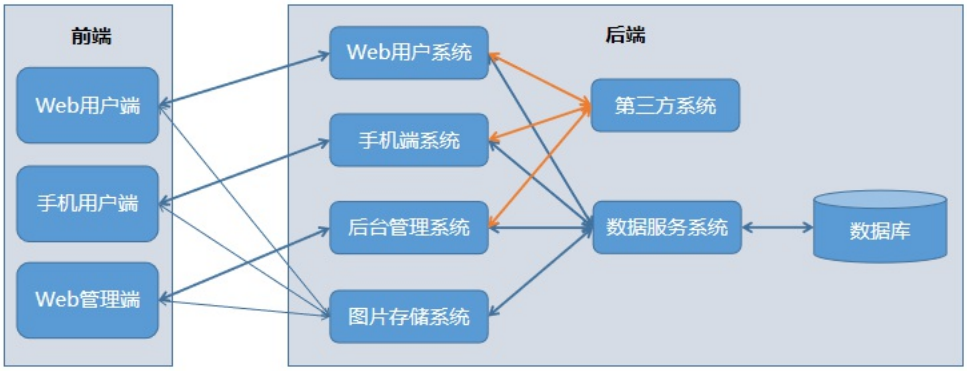
- 前台：借款人、投资人
- 后台：平台管理员

功能模块

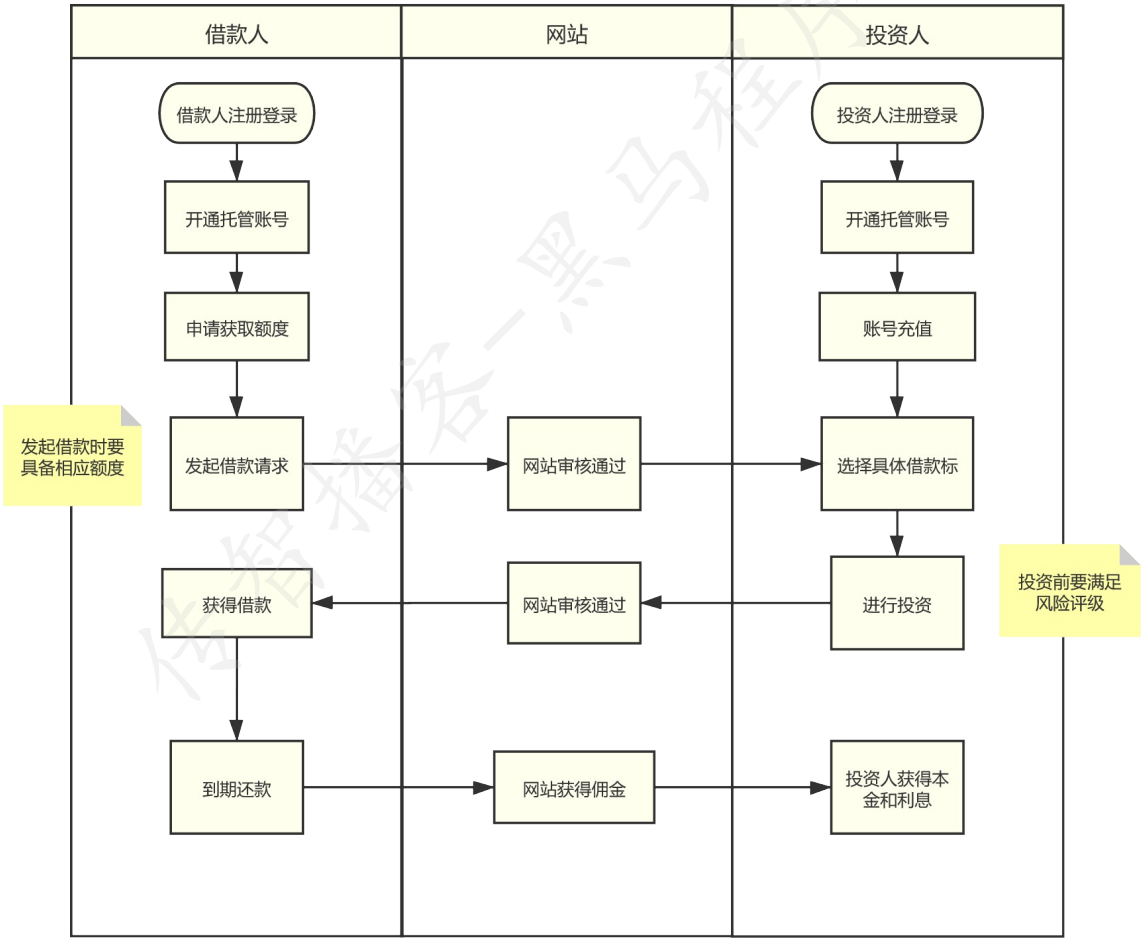
该金融项目包含三个子系统：

- web用户端
  - PC网站，供借款人和投资人使用
  - 网址：http://user-p2p-test.itheima.net
  - 功能模块：
    - 借款人：借款管理
    - 投资人：理财管理
    - 公共：登录、注册、会员中心、资金管理、账户管理、推广管理
- 手机用户端
  - APP和WAP，供借款人和投资人使用
  - 网址：http://mobile-p2p-test.itheima.net
  - 功能模块：
    - 登录、通用、借款、认证、用户中心、债权转让、推广、托管
- 后台管理系统
  - PC网站，内部运营管理系统
  - 网址：http://admin-p2p-test.itheima.net
  - 功能模块：
    - 登录、系统首页、借款管理、资金管理、认证管理、用户管理、内容管理、系统管理、统计管理、扩展管理

项目技术架构图



2. 项目核心业务流程



回顾项目测试流程

目标

1. 能够按照项目的测试流程开展测试工作

项目的测试流程

1. 需求评审
2. 制定测试计划和方案
3. 编写系统测试用例并评审
4. 编写接口测试用例并评审
5. 接口测试用例执行与BUG跟踪
6. 系统测试用例执行与BUG跟踪
7. 编写测试报告

系统测试分析与设计

目标

1．针对金融项目，制定整体项目测试阶段的测试计划
2．针对金融项目，编写系统测试用例并完成评审

项目需求评审

目标

1. 能够完成所测功能（借款和投资）的需求评审

1. 项目需求评审

需求评审前的准备

明确自己负责评审哪些需求

提前熟悉需求，记录对需求的疑问，为评审会议做准备

需求评审的目标

评审需求说明是否有遗漏

确保需求说明书清晰，准确，没有歧义

确保需求说明相互之间没有冲突，重复

测试计划

目标

1. 能够说出测试计划的核心内容
2. 能够编写简单的软件测试计划
3. 能够发现测试计划文档中的明显问题

1. 测试计划

前提说明

软件测试计划和软件测试方案有所区别

但是两者可以合并在一个文档体现：软件测试计划

软件测试计划的核心内容

测试的目标与范围

执行计划的角色与职责

任务的进度安排与资源分配

风险估计与应急计划

测试的准入/准出标准

软件测试方案的核心内容

测试策略

测试环境

测试工具

合并后的软件测试计划

测什么

谁来测

怎么测

2. 编写测试计划

软件测试计划编写步骤

提示：软件测试计划的进度安排要参照项目整体交付计划

1. 找到 / 整合 / 制作合适的文档模板

2. 准备测试计划的核心内容
   2.1 测试目标和范围
   2.2 评估测试工作量
   2.3 测试人员和时间
   2.4 测试方法和工具

3. 按照模板，编写软件测试计划文档

4. 测试团队内外，沟通，确认软件测试计划

注意：测试过程中，根据测试工作开展的实际情况及时调整测试计划

3. 评审测试计划

评审测试计划的过程

同学组内评审测试计划并记录评审问题

同学两组之间相互评审测试计划

老师挑选测试计划在班级范围内进行评审

评审测试计划的注意点

测试计划文档涵盖了核心内容

测试计划合理，可行，能够按时达到测试目标

测试方法正确，有效，能够指导具体测试工作

系统测试用例的设计

目标

1. 能够完成一般的功能测试工作
2. 能够编写合格的测试用例文档
3. 能够发现测试点中的明显问题
4. 能够发现测试用例中的明显问题

1. 功能测试的步骤

1. 对所测功能进行需求分析
2. 根据需求，整理测试点
3. 根据测试点，编写测试用例

2. 测试用例模板

用例ID　功能模块　优先级　用例标题　前置条件　测试数据　执行步骤　预期结果

3. 评审测试点

评审测试点的过程

同学组内评审测试点并记录评审问题

同学两组之间相互评审测试点

老师挑选测试点在班级范围内进行评审

评审测试点的注意点

测试点正确覆盖了所有的功能点，没有错误和遗漏

测试点描述简练清晰，可读性强

测试点能够指导测试用例的编写

测试点没有重复和冗余

4. 评审测试用例

评审测试用例的过程

同学组内评审测试用例并记录评审问题

同学两组之间相互评审测试用例

老师挑选测试用例在班级范围内进行评审

评审测试用例的注意点

用例覆盖了所有测试点

用例容易看懂，方便执行

用例没有重复和冗余

接口测试

目标

1. 能够针对金融项目的接口文档完成接口测试用例的编写

接口测试的应用场景

目标

1. 了解接口测试的应用场景

1. 接口测试的应用场景

接口测试用例的设计：

- 需求分析结束后，测试会根据需求编写系统测试用例，同时开发会根据需求来进行设计（包括前后端的接口和数据库设计等），设计完成后测试人员可以根据接口设计文档来编写接口的测试用例。

接口测试用例的执行：

接口测试的执行通常分为手工测试执行和自动化测试执行两种方式：

- 手工测试执行：
  - 在项目后端开发完成后（前端可能还没有开发完成），可以进行接口测试用例的手工执行，保证尽早的发现问题。
  - 如果项目的前后端同时开发完成，一般不会再对所有的接口用例进行手工执行，直接会进行系统测试用例的执行，只使用接口来辅助测试一些系统测试时无法模拟的异常场景。
- 自动化测试执行：
  - 为了防止开发修改代码时引入新的问题，通常使用接口自动化的方式来看护代码的质量。
  - 由于后端的需求变化比前端小，所以公司通常优先进行接口自动化，然后再考虑UI自动化。

项目接口的特殊点

目标

1. 熟悉金融项目接口的特殊点

1. 复杂的业务逻辑

- 金融项目涉及较多的专业术语
- 金融项目业务关联性比较强
- 金融项目涉及到财务知识

2. 依赖于第三方接口

- 第三方资金托管是指为P2P平台开发定制账户系统，提供系统外包运营服务；另一方面，为P2P平台提供支付和结算服务，帮助平台和用户实现充值、取现、资金划拨等；同时，保障用户资金由银行全程监管，投资人资金划入虚拟账户后，纳入汇付客户备付金管理体系。这一模式既满足了P2P平台为其客户提供各类基于投融资交易的支付服务需求，又确保了交易资金全程由银行监管，使得平台无法触碰资金，避免了资金池模式。

3. 接口加解密

- 手机端的接口数据进行了加密处理
- 发送请求之前对请求参数进行加密
- 得到接口响应数据后需要进行解密

编写接口测试用例

目标

1．熟悉待测功能的接口
2．掌握如何编写测试用例

1. 待测功能分析

重点测试投资流程相关的接口，接口梳理如下：

1. 注册
2. 登录
3. 开通托管账号
4. 充值
5. 投资产品列表
6. 投资产品详情
7. 投资
8. 我的投资列表

2.接口用例设计的方法与思路

- 单接口测试
    - 正向功能：(通过性测试) 仅必填参数 全部参数 参数组合
    - 反向测试：(异常测试)
        - 参数异常：少参、多参
        - 数据异常：数据为空、长度不符、类型不符、错误数据
        - 业务数据异常：结合业务功能考虑输出的各种异常返回情况
- 多接口测试：业务场景功能测试（站在用户角度考虑常用的使用场景） 接口之间数据依赖

3. 编写接口测试用例

ID	模块	用例名称	接口名称	前置条件	请求URL	请求类型	请求头	请求参数类型	请求参数	预期结果	测试结果
001	登录注册	注册成功	获取图片验证码		/common/public/verifycode1/{r}	GET		url	0.782847146	获取成功，状态码：200，返回数据：图片	

			获取短信验证码		/member/public/sendSms	POST	{"Cookie":"JSESSIONID=99679F2B755D2C7595CBEEB9D9215DD0"}	form	{
									"phone": "13812345678",
									"imgVerifyCode": "8888",
									"type": "reg"
									}
										获取成功，状态码：200，返回数据：{"status":200,"description":"短信发送成功"}	

			注册		/member/public/reg	POST	{"Cookie":"JSESSIONID=99679F2B755D2C7595CBEEB9D9215DD0"}	form	{
									"phone": "13812345678",
									"password": "test123",
									"verifycode": "8888",
									"phone_code": "200201",
									"dy_server": "on",
									"invite_phone": ""
									}
										注册成功，状态码：200，返回数据：{"status":200,"description":"注册成功"}	

接口测试

目标

1. 能够对金融项目完成接口测试的工作

接口测试环境准备

目标

1. 了解进行项目部署
2. 熟悉Mock第三方接口的开发流程

1. 项目部署（了解）

- 安装应用服务器和数据库服务器
- 部署项目代码，并进行相关的配置

2. Mock接口服务（了解)

使用Python + Flask搭建web服务，通过Requests实现回调接口的调用。

Mock接口列表：

- 开户
- 充值
- 查询余额
- 主动投标
- ...

示例代码：

```
@app.route("/muser/publicRequests", methods=["GET", "POST"])
def mock():
    print("request param===", request.form)
    CmdId = request.form.get("CmdId")
    # 开户
    if CmdId == "UserRegister":
        return userRegister()
    # 充值
    elif CmdId == "NetSave":
        return NetSave()
    # 查询余额
    elif CmdId == "QueryBalanceBg":
        return QueryBalanceBg()
    # 主动投标
    elif CmdId == "InitiativeTender":
        return InitiativeTender()
    else:
```

```
        print("未知操作！！！")
        return "OK"

# 开户
def userRegister():
    # 获取请求参数
    Version = request.form.get("Version")
    CmdId = request.form.get("CmdId")
    MerCustId = request.form.get("MerCustId")
    BgRetUrl = request.form.get("BgRetUrl")
    RetUrl = request.form.get("RetUrl")
    UsrId = request.form.get("UsrId")
    UsrName = request.form.get("UsrName")
    IdType = request.form.get("IdType")
    IdNo = request.form.get("IdNo")
    UsrMp = request.form.get("UsrMp")
    UsrEmail = request.form.get("UsrEmail")
    MerPriv = request.form.get("MerPriv")
    ChkValue = request.form.get("ChkValue")
    CharSet = request.form.get("CharSet")

    # 构造回调请求数据
    # 构造用户账号，格式：6000060011481418
    UsrCustId = "60{}".format(time.strftime("%Y%m%d%H%M%S"))
    print("UsrCustId==========", UsrCustId)
    data = {
        "UsrCustId": UsrCustId,
        "BgRetUrl": BgRetUrl,
        "UsrName": UsrName,
        "IdType": IdType,
        "MerPriv": MerPriv,
        "RetUrl": RetUrl,
        "UsrMp": UsrMp,
        "TrxId": "406544084846401438",
        "UsrId": UsrId,
        "RespCode": "000",
        "RespDesc": "成功",
        "IdNo": IdNo,
        "ChkValue": ChkValue,
        "MerCustId": MerCustId,
        "UsrEmail": UsrEmail,
        "CmdId": CmdId,
    }
    response = requests.post(RetUrl, data=data)
    print("response.txt===", response.text)

    # 保存用户信息到数据库
    conn = DBUtil.get_conn()
    conn.autocommit(True)
```

```
    cursor = DBUtil.get_cursor()
    sql = "insert into p2p_account(account, create_time) values (%s, %s)"
    cursor.execute(sql, (UsrCustId, time.strftime("%Y-%m-%d %H:%M:%S")))
    print("new id=====", cursor.lastrowid)
    DBUtil.close_cursor(cursor)
    DBUtil.close_conn()

    return "UserRegister OK"
```

测试数据准备

目标

1. 知道如何准备测试数据

1. 测试数据准备方式

- 手动操作系统进行构造
  - 要求对应功能已经实现
  - 效率比较低
  - 适合不需要频繁构造的数据
- 调用其它接口构造
  - 依赖数据准备接口的正确性
  - 接口用例中各个API的耦合度比较高
- 直接操作数据库
  - 比较灵活
  - 对数据库的表结构熟悉程度要求比较高（某一个接口的数据准备可能涉及到多张表操作）
  - 数据库表结构发生变化了，可能会导致之前的用例执行失败

> 在工作中，需要根据实际情况来选择数据准备的方法。
> 如果对数据库操作比较方便，可以直接操作数据库；
> 对于逻辑比较复杂，需要操作数据库多张表的情况，可以考虑采用调用其它API的方式准备数据；
> 对于只需要准备一次，并且对应功能已经实现的情况，可以直接手动操作系统来准备。

2. 测试数据准备实践

2.1 需要准备的测试数据

- 借款人基础数据
- 借款数据

2.2 手动构造借款人基础数据

操作步骤：

1. 打开用户系统
2. 注册借款人账号
3. 开通资金托管账号
4. 申请额度，管理员后台审核通过

2.3 直接操作数据库构造借款数据

借款相关表：

- fn_loan：借款标主表
- fn_loan_info：借款标明细表
- fn_loan_amount：借款额度表
- fn_loan_amount_log：额度变化日志表

手工执行接口测试

目标

```
1. 掌握如何通过JMeter手工执行接口测试
2. 掌握对失败用例进行问题分析
```

1.手工执行接口测试的场景

- 测试目的：
  - 保证尽早的发现问题
- 测试时机：
  - 在项目后端开发完成后（前端可能还没有开发完成），可以进行接口测试用例的手工执行
  - 注意：如果项目的前后端同时开发完成，一般不会再对所有的接口用例进行手工执行，直接会进行系统测试用例的执行，只使用接口来辅助测试一些系统测试时无法模拟的异常场景。
- 测试依据：
  - 手工执行接口测试的用例是根据接口设计文档来设计

2. 手工执行接口测试方式

在工作中通常会使用Jmeter或者Postman工具来手工执行接口测试用例

在本项目中我们选择Jmeter的方式来手工执行

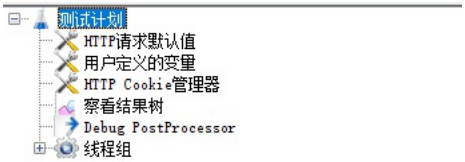
3. 基于JMeter测试

3.1 常用测试元件

1. 取样器-HTTP请求
2. 配置元件-HTTP请求默认值
3. 配置元件-用户定义的变量
4. 配置元件-HTTP Cookie管理器
5. 后置处理器-JSON提取器
6. 后置处理器-正则表达式提取器
7. 断言-响应断言
8. 断言-JSON断言
9. 监听器-察看结果树

3.2 初始化工作

1. 创建测试用例结构
2. 设置HTTP请求默认值



3.3 实现测试用例

根据编写的测试用例文档，使用JMeter实现测试用例

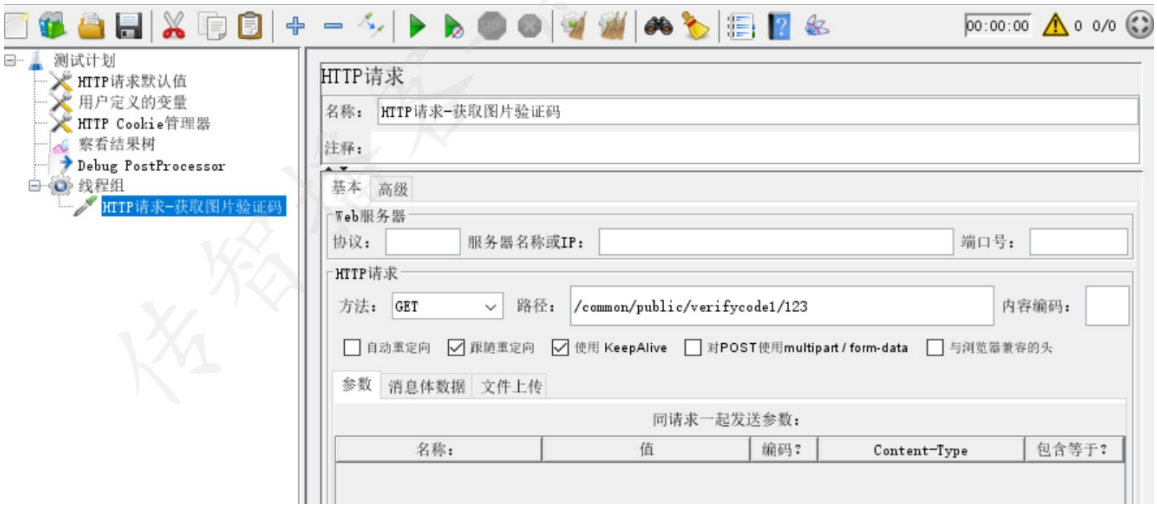
注册成功

操作步骤：

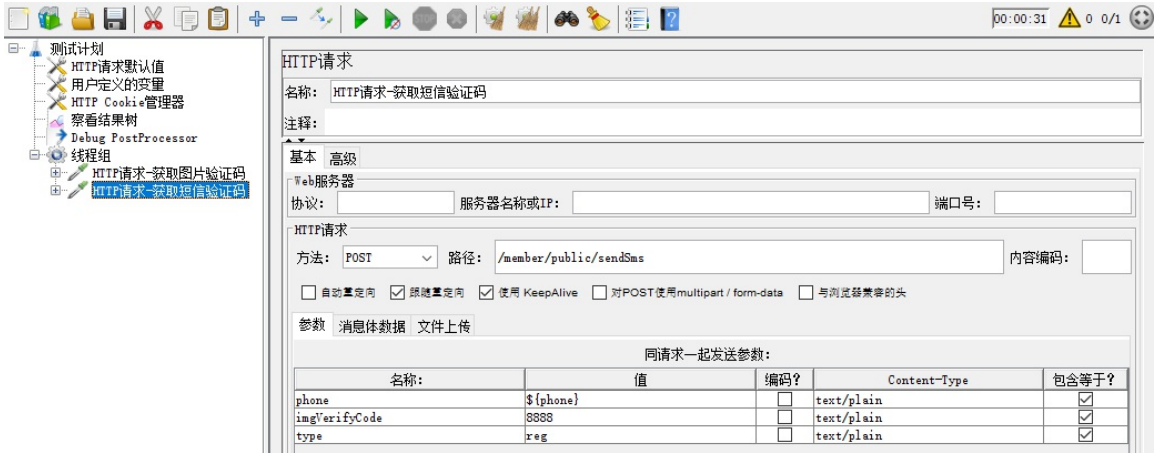
1. 在线程组下，添加取样器‘HTTP请求-获取图片验证码’，并填写请求数据
2. 在线程组下，添加取样器‘HTTP请求-获取短信验证码’，并填写请求数据
3. 在线程组下，添加取样器‘HTTP请求-注册’，并填写请求数据
4. 发送请求，调试脚本

实现截图：

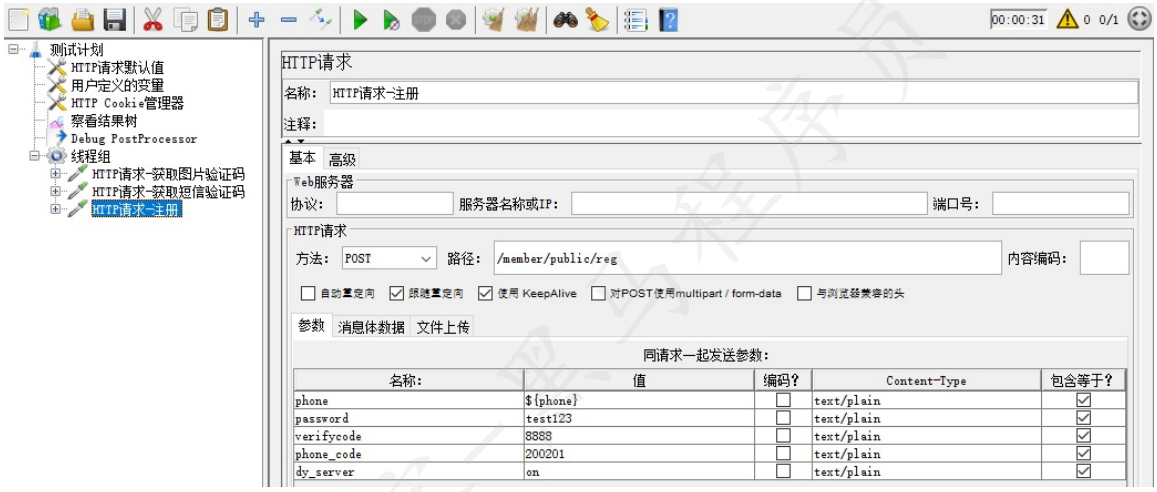
1. 添加‘HTTP请求-获取图片验证码’的请求



2. 添加‘HTTP请求-获取图片验证码’的请求



3. 添加‘HTTP请求-获取图片验证码’的请求



4. 失败用例问题分析

- 针对执行失败的测试用例，分析异常信息并进行分析判断
- 脚本问题：调试脚本，再次进行测试
- 系统bug：提交缺陷，并进行跟踪管理

编写自动化接口测试脚本

目标

```
1. 掌握如何通过JMeter实现接口自动化测试并生成测试报告
2. 掌握如何使用代码实现接口自动化测试并生成测试报告
```

1.自动化执行接口测试的场景

- 测试目的：防止开发修改代码时引入新的问题
- 测试时机：
  - 开发进行系统测试转测前，可以先进行接口自动化脚本的编写
  - 开发进行系统测试转测后，优先进行系统测试用例的执行，再进行接口自动化脚本的编写。
  - 注意：编写接口自动化脚本和系统测试用例执行没有明确的先后顺序，在项目中系统测试用例执行的优先级更高。

  这里强调的是接口自动化脚本的编写时机，因为自动化脚本一旦编写并调试完成后，会定时、自动地一直运行下去，因此脚本编写完成后的所有项目时间中都会进行自动化脚本的执行工作。
- 测试依据
  - 自动化执行接口测试的用例可以根据接口设计文档来设计
  - 自动化执行接口测试的用例也可以根据环境抓包来设计（在实际环境进行业务操作并抓包来分析关键接口）

2. 自动化接口测试实现方式

在本项目中会采用以下两种方式进行自动化接口测试：

- 基于工具测试：JMeter
- 基于代码测试：Python + Requests

3. 基于JMeter的接口自动化测试

3.1 Jmeter手工测试脚本进行自动化调优

- 参数化
- 断言
- 输出测试报告

3.2 实现测试用例

根据编写的测试用例文档，使用JMeter实现测试用例

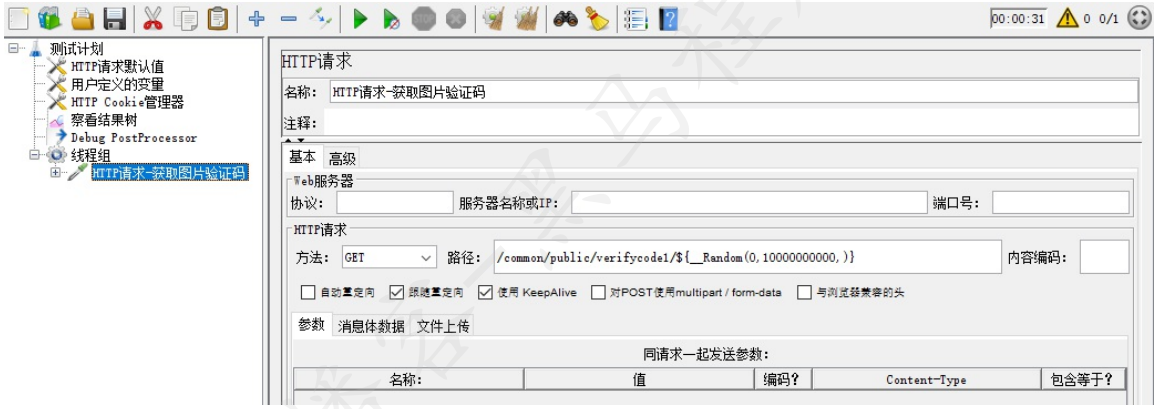
注册成功

操作步骤：

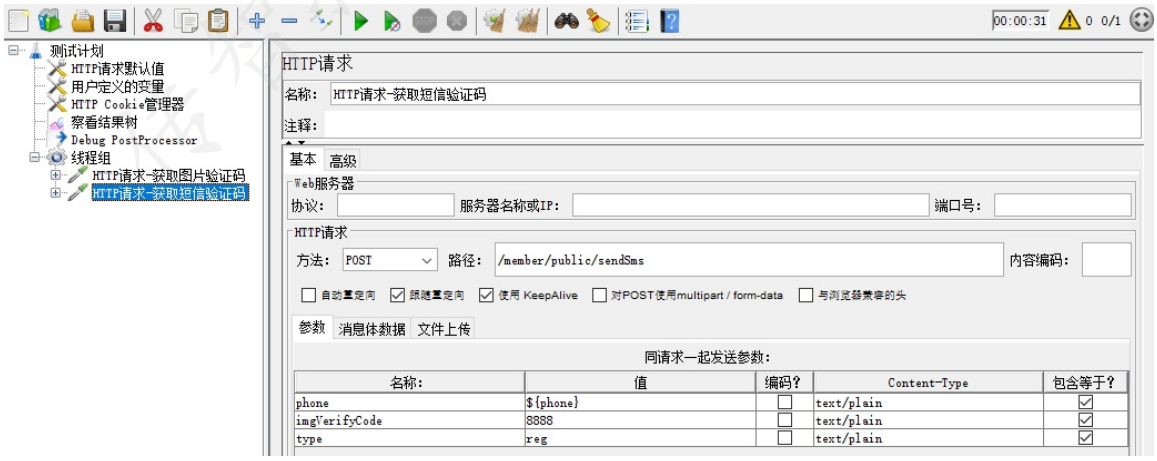
1. 在线程组下，添加取样器‘HTTP请求-获取图片验证码’，并填写请求数据
2. 在线程组下，添加取样器‘HTTP请求-获取短信验证码’，并填写请求数据
3. 在线程组下，添加取样器‘HTTP请求-注册’，并填写请求数据
4. 在取样器下，添加‘响应断言’断言响应状态码
5. 在取样器下，添加‘JSON断言’断言status和description
6. 发送请求，调试脚本

实现截图：

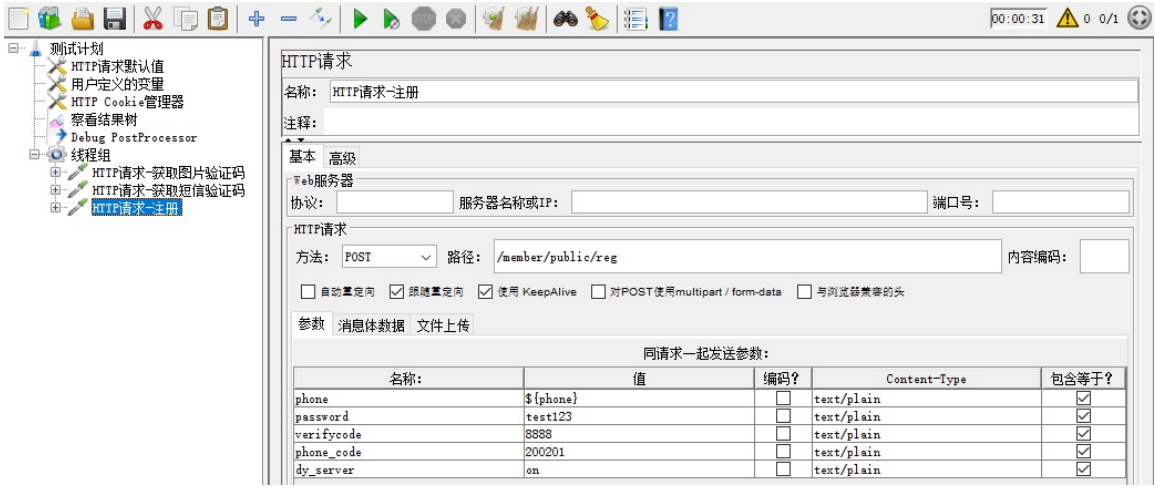
1. 添加‘HTTP请求-获取图片验证码’的请求



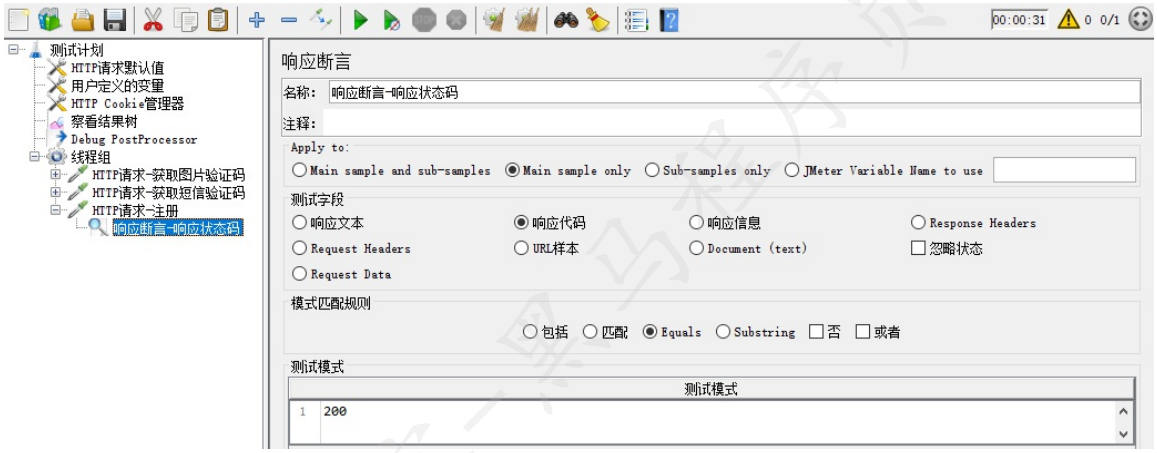
2. 添加‘HTTP请求-获取图片验证码’的请求



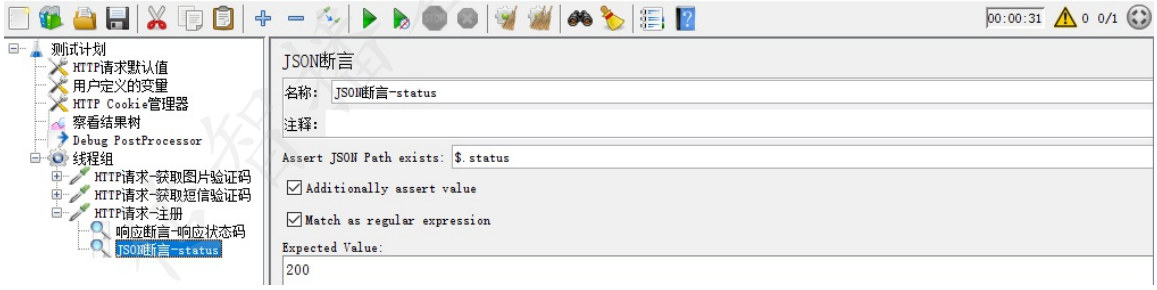
3. 添加‘HTTP请求-获取图片验证码’的请求



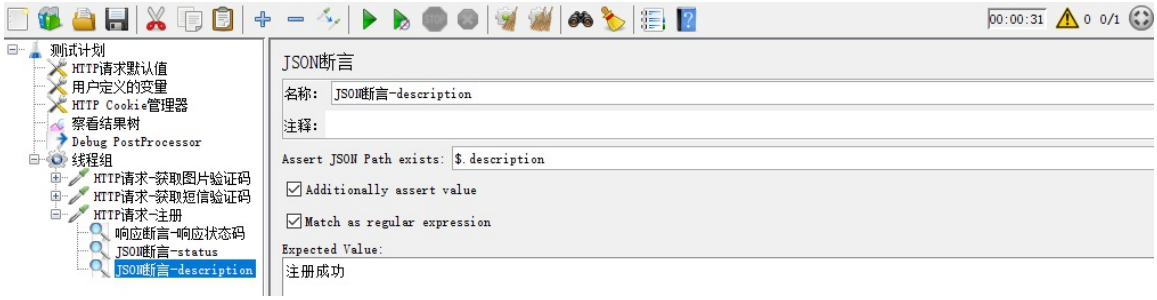
4. 添加‘响应断言-响应状态码’



5. 添加‘JSON断言-status’



6. 添加‘JSON断言-description’



3.3 生成测试报告

```
jmeter -n -t [jmx file] -l [result file] -e -o [html report folder]

eg: jmeter -n -t hello.jmx -l result.jtl -e -o ./report
```

参数描述：

- -n：非GUI模式执行JMeter
- -t [jmx file]：测试计划保存的路径及.jmx文件名，路径可以是相对路径也可以是绝对路径
- -l [result file]：保存生成测试结果的文件，jtl文件格式
- -e：测试结束后，生成测试报告
- -o [html report folder]：存放生成测试报告的路径，路径可以是相对路径也可以是绝对路径

注意：result.jtl和report会自动生成，如果在执行命令时result.jtl和report已存在，必须用先删除，否则在运行命令时就会报错

4. 基于代码测试

4.1 项目搭建

新建项目

项目名称：apiTestP2P

创建目录结构

```
apiTestP2P
├── api
├── script
├── data
├── report
├── lib
├── log
├── app.py
├── utils.py
└── run_suite.py
```

安装依赖包

- 安装 `requests` 包
- 安装 `parameterized` 包
- 安装 `pymysql` 包
- 添加 `HTMLTestRunner`

4.2 初始化日志配置

使用Python中的 `logging` 日志模块来收集日志，把日志信息输出到控制台和日志文件中

```
# app.py
# 初始化日志配置
def init_log_config():
    # 创建日志器
    logger = logging.getLogger()
    logger.setLevel(logging.INFO)

    # 创建控制台处理器
    sh = logging.StreamHandler()

    # 创建文件处理器
    log_path = BASE_DIR + "/log/p2p.log"
    fh = logging.handlers.TimedRotatingFileHandler(log_path, when="midnight", interval=1
,
                                                   backupCount=7, encoding="UTF-8")

    # 创建格式化器
    f='%(asctime)s %(levelname)s [%(name)s] [%(filename)s(%(funcName)s:%(lineno)d)] - %
(message)s'
    formatter = logging.Formatter(f)

    # 把格式化器添加到处理器中
    sh.setFormatter(formatter)
    fh.setFormatter(formatter)

    # 把处理器添加到日志器中
    logger.addHandler(sh)
    logger.addHandler(fh)
```

4.3 编写代码

封装接口类

根据用例分析待测功能，按功能模块定义接口类

```
登录模块：login.py
认证模块：approve.py
托管模块：trust.py
投资模块：tender.py
```

编写测试脚本

1. 定义测试脚本文件

```
登录模块：test_login.py
认证模块：test_approve.py
托管模块：test_trust.py
投资模块：test_tender.py
```

2. 使用unittest管理测试脚本

执行测试脚本

1. 使用unittest执行测试脚本
2. 调试代码

4.4 数据驱动

定义数据文件

1. 定义存放测试数据的目录，目录名称：data
2. 分模块定义数据文件
3. 根据业务编写用例数据

测试数据参数化

修改测试脚本，使用 `parameterized` 实现参数化

4.5 生成测试报告

使用 `HTMLTestRunner` 生成测试报告，并分析测试结果

```
report_file = "./report/report{}.html".format(time.strftime("%Y%m%d-%H%M%S"))
with open(report_file, "wb") as f:
    runner = HTMLTestRunner(f, title="P2P接口自动化测试报告", description="V1.0")
    runner.run(suite)
```

测试数据清理

目标

1. 知道如何清理测试数据

1. 测试数据清理

1.1 为什么要清理测试数据？

- 还原到测试前的环境
- 实现测试用例多次重复执行
- 减少对其它测试用例的数据影响

1.2 如何清理测试数据

- 调用删除接口进行清理
- 直接操作数据库进行删除

2. 测试数据清理实践

2.1 直接操作数据库删除用户信息

用户信息相关表：

- mb_member：用户主表
- mb_member_info：用户信息表
- mb_member_login_log：用户登录日志表
- mb_member_register_log：用户注册日志表

清除注册数据：

```
@classmethod
def tearDownClass(cls) -> None:
    # 清除注册数据
    conn, cursor = None, None
    try:
        conn = DBUtil.get_conn(DBUtil.DB_MEMBER)
        conn.autocommit(True)
        cursor = conn.cursor()
        # 用户主表
```

```
        sql = "delete from mb_member where name=%s"
        cursor.execute(sql, (TestLogin.reg_phone,))
        # 用户信息表
        sql = "delete from mb_member_info where member_name=%s"
        cursor.execute(sql, (TestLogin.reg_phone,))
        # 用户注册日志表
        sql = "delete from mb_member_register_log where member_name=%s"
        cursor.execute(sql, (TestLogin.reg_phone,))
        conn.commit()
    except Exception as e:
        conn.rollback()
        traceback.print_exc()
    finally:
        DBUtil.close(cursor, conn)
```

执行接口测试

目标

1. 掌握如何执行接口测试

1. 执行接口测试

基于代码测试

- 把测试用例封装成测试套件执行
- 设置有依赖关系的接口执行顺序

基于**JMeter**测试

- 批量执行测试用例

2.定时执行接口测试

- 配置Jenkins的持续集成工具
- 在Jenkins的Builds组件中，添加代码测试套件的执行命令，或者Jmeter脚本的批处理运行命令
- 完成接口自动化脚本的定时、自动运行

接口加解密

目标

1. 知道项目中接口加解密的实现流程
2. 了解加解密的代码实现
3. 知道JMeter如何实现接口加解密测试
4. 知道基于Requests如何实现接口加解密测试

1. APP接口加密介绍

1.1 加密方式

- AES：(Advanced Encryption Standard)密码学中的高级加密标准，是美国联邦政府采用的一种区块加密标准。
- 这个标准用来替代原先的DES（Data Encryption Standard），已经被多方分析且广为全世界所使用。

2. 项目接口加解密实现分析

以登录接口为例，介绍接口加解密的实现

2.1 加密之前

- 基本信息
  - 接口名称：登录
  - 请求方式：POST
  - 请求路径：/phone/member/login
- 请求参数（表单提交）
  - member_name： 用户名或手机号码或邮箱
  - password： 密码
- 返回数据（JSON格式）
  - `{"code":200,"result":"success","data":{"member":{"name":"13012345678"},"login_token":147}}`

2.2 加密之后

- 基本信息
  - 接口名称：登录
  - 请求方式：POST

    - 请求路径：/phone/member/login
- 请求参数（表单提交）
    - diyou：加密后的请求数据，把原始请求参数(member_name、password)进行加密得到的字符串
    - xmdy：签名，对加密后的请求数据进行签名
- 返回数据（JSON格式）
    - `{"xmdy":"df0990efe05976e4ef326dab5bf3fc83","diyou":"xxx"}`
    - diyou：加密后的响应数据
    - xmdy：签名

3. 加解密代码实现

根据加密方式，结合开发人员提供的源码，封装自己的加解密方法，方便在接口测试中使用...

3.1 Java实现方式

核心代码展示：

```
/**
 * 加密
 *
 * @param key      密钥
 * @param content 待加密数据
 * @return 加密后的数据
 */
public static String aesEncrypt(String key, String content) {
    try {
        System.out.println("aesEncrypt key=" + key);
        System.out.println("aesEncrypt content=" + content);

        content = new BASE64Encoder().encodeBuffer(content.getBytes());
        content = replaceBlank(content);

        SecretKey secretKey = new SecretKeySpec(key.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] aes_byte = cipher.doFinal(content.getBytes(StandardCharsets.UTF_8));
        return new BASE64Encoder().encode(aes_byte);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

/**
 * 解密
```

```
     *
     * @param key     密钥
     * @param content 待解密数据
     * @return 解密之后的数据
     */
    public static String aesDecrypt(String key, String content) {
        try {
            // 根据字节数组生成AES密钥
            SecretKey secretKey = new SecretKeySpec(key.getBytes(), "AES");
            // 根据指定算法AES生成密码器
            Cipher cipher = Cipher.getInstance("AES");
            // 初始化密码器
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            // 将加密并编码后的内容解码成字节数组
            byte[] byte_content = new BASE64Decoder().decodeBuffer(content);
            // 解密
            byte[] byte_decode = cipher.doFinal(byte_content);

            // 转换为字符串
            String aes_decode = new String(byte_decode, StandardCharsets.UTF_8);
            String aes_decode2 = new String(new BASE64Decoder().decodeBuffer(aes_decode));
            return unicodeToString(aes_decode2);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

3.2 Python实现方式

python在Windows下使用AES时要安装 `pycryptodome` 模块，可以使用PIP进行安装：

```
pip install pycryptodome
```

核心代码展示：

```
@staticmethod
def aes_encrypt(key, data):
    """
    AES加密
    :param key: 密钥
    :param data: 待加密数据
    :return: 加密后数据
    """
    data = base64.encodebytes(data.encode()).decode()
    # 替换特殊字符
    data = EncryptUtil.replace_blank(data)
```

```
    print("data=", data)

    # 初始化加密器
    aes = AES.new(key.encode(), AES.MODE_ECB)

    # 解密
    padding_value = EncryptUtil.padding_pkcs5(data)
    encrypt_aes = aes.encrypt(padding_value)

    # 用base64转成字符串形式
    encrypted_text = base64.encodebytes(encrypt_aes).decode()
    return encrypted_text

@staticmethod
def aes_decrypt(key, data):
    """
    AES解密
    :param key: 密钥
    :param data: 待解密数据
    :return: 解密后数据
    """
    # 初始化加密器
    aes = AES.new(key.encode(), AES.MODE_ECB)
    # 优先逆向解密base64成bytes
    base64_decrypted = base64.decodebytes(data.encode())

    # 执行解密
    decrypted_bytes = base64.decodebytes(aes.decrypt(base64_decrypted))
    # 转换为字符串
    decrypted_text = str(decrypted_bytes, encoding="utf-8")

    # 把Unicode转成中文
    result = decrypted_text.encode().decode("unicode_escape")
    return result
```

4. Python+Requests实现加解密测试

4.1 实现步骤

1. 构造测试数据
2. 对请求参数进行加密
3. 发送请求
4. 对响应数据进行解密
5. 断言

4.2 示例代码

```python
import json
import unittest
import requests
from utils import EncryptUtil

class TestMobile(unittest.TestCase):

    def test_login(self):
        """登录"""
        url = "http://mobile-p2p-test.itheima.net/phone/member/login"

        # 请求参数
        req_data = {
            "member_name": "13012345678",
            "password": "test123"
        }

        # 对请求参数进行加密，并获取签名
        diyou = EncryptUtil.get_diyou(req_data)
        xmdy = EncryptUtil.get_xmdy(diyou)

        # 发送请求
        r = requests.post(url, data={"diyou": diyou, "xmdy": xmdy})
        print("r.text==", r.text)

        # 对响应数据进行解密
        json_data = r.json()
        diyou_data = json_data.get("diyou")
        decrypted_data = EncryptUtil.decrypt_data(diyou_data)
        print("decrypted_data=", decrypted_data)

        # 断言
        json_data = json.loads(decrypted_data)
        self.assertEqual(200, json_data.get("code"))
        self.assertEqual("success", json_data.get("result"))
```

5. JMeter实现加解密测试

5.1 实现步骤

1. 在Java项目中封装加解密方法，并导出jar包
2. 在JMeter中引入依赖的jar包（包括：加解密jar包、以及其他依赖的jar包）
   - 把jar包放到JMeter的lib\ext目录下
   - 在测试计划-->添加目录或jar包到ClassPath，添加需要调用的jar包
3. 在‘HTTP请求’取样器下添加前置处理器‘BeanShell PreProcessor’，并编写对请求数据进行加密的代码

4. 在‘HTTP请求’取样器下添加后置处理器‘BeanShell PostProcessor’，并编写对响应数据进行解密代码
5. 在‘HTTP请求’取样器下添加断言‘BeanShell断言’，对返回数据进行断言

5.2 前置处理脚本（实现请求数据加密）

```
log.info("==============================" + sampler.getName() + "======================
========");

import cn.itcast.p2p.EncryptUtil;
import org.apache.jmeter.config.*;

// 获取请求参数
TreeMap map = new TreeMap();
Arguments args = sampler.getArguments();
for(int i=0; i<args.getArgumentCount(); i++){
    Argument arg = args.getArgument(i);
    String name = arg.getName();
    String value = arg.getValue();
    log.info("name="+name+"  value="+value);
    map.put(name, value);
}
// 加密
String diyou = EncryptUtil.getDiyou(map);
String xmdy = EncryptUtil.getXmdy(diyou);
// 添加请求参数
sampler.addArgument("diyou", diyou);
sampler.addArgument("xmdy", xmdy);
```

5.3 后置处理脚本（实现响应数据解密）

```
import cn.itcast.p2p.EncryptUtil;
import org.json.*;

// 获取响应数据
String responseStr = prev.getResponseDataAsString();
log.info("responseStr=" + responseStr);
JSONObject responseJson = new JSONObject(responseStr);
// 获取响应数据diyou
String diyou_data = responseJson.getString("diyou");
log.info("diyou_data="+diyou_data);
// 解密
String jsonDataStr = EncryptUtil.decryptDiyou(diyou_data);
log.info("jsonDataStr=="+jsonDataStr);
// 添加到变量列表中
vars.put("jsonDataStr", jsonDataStr);
```

5.4 BeanShell断言

```
import org.json.*;

// 获取解密之后的响应数据
String jsonDataStr = vars.get("jsonDataStr");
JSONObject jsonData = new JSONObject(jsonDataStr);

// 断言-code
if(jsonData.getInt("code") != 200){
    Failure = true;
    FailureMessage = "code的返回值有误";
}
```

扩展

目标

```
1. 掌握如何使用BeautifulSoup解析HTML文档
```

1. BeautifulSoup

1.1 BeautifulSoup介绍

- Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库.
- 它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式.
- Beautiful Soup会帮你节省数小时甚至数天的工作时间.
- Beautiful Soup 3目前已经停止开发，推荐在项目中使用Beautiful Soup 4，简称BS4.

1.2 安装 BeautifulSoup

使用PIP工具可以直接进行安装：

```
pip install beautifulsoup4
```

> 注意：包的名称为 `beautifulsoup4` 而不是 `BeautifulSoup` ，BeautifulSoup是Beautiful Soup3的发布版本

1.3 如何使用

将一段文档传入BeautifulSoup的构造方法,就能得到一个文档的对象, 可以传入一段字符串或一个文件句柄。

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(open("index.html"), "html.parser")

soup = BeautifulSoup("<html>data</html>", "html.parser")
```

参数说明：

- "html.parser"：Python标准库中的HTML解析器。BeautifulSoup还支持一些第三方的解析器，如：lxml、html5lib等等， 这些第三方解析器需要额外安装。

1.4 基本用法

通过BeautifulSoup对象可以获取文档元素的所有信息，下面我们介绍一些最基本、最常用的方法。

```
from bs4 import BeautifulSoup

html = """
<html>
<head><title>黑马程序员</title></head>
<body>
    <p id="test01">软件测试</p>
    <p id="test02">2020年</p>
    <a href="/api.html">接口测试</a>
    <a href="/web.html">Web自动化测试</a>
    <a href="/app.html">APP自动化测试</a>
</body>
</html>
"""
soup = BeautifulSoup(html, "html.parser")

print(soup.title)  # 获取title标签
print(soup.title.name)  # 获取title标签的名称
print(soup.title.string)  # 获取title标签的文本内容

print(soup.p)  # 获取第一个p标签
print(soup.p["id"])  # 获取第一个p标签的id属性值

print(soup.find_all("a"))  # 获取所有的a标签

# 获取所有的a标签，并遍历打印a标签的href属性值和文本内容
for a in soup.find_all("a"):
    print("href={} text={}".format(a["href"], a.string))
    print("href={} text={}".format(a.get("href"), a.get_text()))
```

系统测试执行并编写测试报告

目标

1. 能够对金融项目完成系统测试用例的执行工作
2. 能编写系统测试报告

执行测试用例并提交缺陷

目标

1. 执行用例，并能够按要求提交缺陷报告

1. 软件缺陷报告模板

基本内容

缺陷标题　前置条件　复现步骤　实际结果　期望结果

其他信息

缺陷ID　严重程度　优先级　缺陷类型　缺陷状态

2.评审缺陷报告的注意点

软件缺陷能够重现

缺陷报告完整清晰，容易理解

确保一个缺陷报告只描述一个BUG

BUG定位

目标

1. 能够理解BUG定位的价值
2. 能够说出BUG定位的技巧

1. BUG定位的引入

BUG示例

BUG描述

【BUG标题】12位的手机号码也能注册成功

【前置条件】手机号码未注册

【复现步骤】1.进入前台用户注册页面；2.填写12位手机号码；3.正确填写其他信息，点击注册

【结果】注册成功，自动登录

【期望】注册失败，给出对应提示

BUG定位的价值

1.找到BUG的本质（必现路径）

2.提升开发修复BUG的效率

3.提升自身的逻辑思维与技术能力

2. BUG定位的技巧

逻辑分析

分析所有可能，逐个排查

找到最短复现路径

技术手段

查看数据库

抓包分析

查看服务器日志

软件测试报告

目标

1. 能够说出软件测试报告的核心内容
2. 能够编写简单的软件测试报告
3. 能够发现软件测试报告中的明显问题

1. 软件测试报告

软件测试报告的目标

汇报近期的测试工作，体现工作过程和成果

评估当前软件质量，为团队决策提供依据

改进自身测试工作，持续提升测试效率和质量

软件测试报告的核心内容

测试工作的经过与结果

缺陷汇总与分析

软件上线风险

测试工作总结与改进

2. 编写软件测试报告

软件测试报告的编写步骤

1. 找到 / 整合 / 制作合适的文档模板

2. 收集，汇总材料
   2.2 每位团队成员的工作记录和成果
   2.3 测试用例的数量和执行结果
   2.4 BUG的状态和数量
   2.5 团队成员的反思与改进建议等

3．按照模板，编写软件测试报告文档

3. 评审软件测试报告

评审软件测试报告的过程

同学组内评审测试报告并记录评审问题

同学两组之间相互评审测试报告

老师挑选测试报告在班级范围内进行评审

评审软件测试报告的注意点

测试报告文档涵盖了核心内容

测试报告准确有效，达到测试报告的目标

项目总结

目标

1. 能够对金融项目的系统测试和接口测试进行总结和回顾

金融项目测试实战总结

目标

1. 能够完整介绍金融项目的测试经验

1. 金融项目测试经验的阐述

- 项目的介绍
  - 项目是做什么的？
  - 项目给谁用的？
  - 项目包含哪些主要功能？
  - 项目的技术架构是什么？
  - 项目的核心业务流程是什么？
- 项目的测试过程是什么？
- 如何测试自己负责的功能模块？
- 接口测试用例如何设计？
- 接口测试的时机/目的是什么？
- 如何准备接口自动化测试过程中的数据？
- 如何使用Jmeter编写接口自动化脚本？
- 如何使用代码编写接口自动化脚本？
- 如何保证接口自动化测试脚本的稳定性？
- 项目中涉及到第三方接口时如何进行接口测试？
- 项目测试过程中发现的印象深刻的BUG
- 项目测试过程中遇到的问题