金融项目第十天--课堂笔记

昨日回顾

（1）数据清理

- 封装连接数据库的基础方法类：导包、建立连接、建立游标、执行SQL语句、关闭游标、关闭连接
- 执行清除数据的SQL语句
    - 准备清除数据SQL语句
    - **确定清除sql的位置**
        - 放在第一个脚本的setupClass中
        - 放在最后一个脚本的teardownClass中
        - 单独定义一个测试类，只存放清除数据的脚本，将这个测试类添加到测试套件的最后边
    - SQL语句执行

（2）参数化

- 定义测试数据文件
    - 所有数据文件放在data目录下
    - 每个模块（接口）定义一个独立的测试数据文件
    - 编写测试数据文件
        - 测试数据文件的描述（给人看）
        - 请求参数（把所有的接口参数都定义在测试数据文件中）
        - 响应结果（把要断言的响应结果定义在测试数据文件中）
- 读取测试数据文件
    - **针对每个数据文件单独定义一个方法，读取数据（编写简单、方法多）**
    - 针对所有数据文件定义一个统一方法，读取数据（编写复杂，方法少）
        - 使用统一读取方法时，要求所有的测试数据文件必须定义为完全相同的数据格式
- 编写脚本使用测试数据文件中的参数
    - 引用读取的测试数据文件中的参数。
        - @parameterized.expand(读取数据文件的方法名(参数))
    - 使用数据文件中定义的参数来替换原脚本中的静态测试数据

（3）持续集成运行

- 通过github管理python自动化脚本
    - 初次上传代码
        - 注册github账号
        - 在pycharm配置github账号
        - 将项目代码同步到github服务器中
    - 更新代码
        - git-add：添加项目代码
        - git-commit：确定更新（写上注释）
        - git-push：将代码上传到github服务器上
- 通过jenkins在定时运行python自动化脚本
    - 添加Jenkins项目

- 配置Jenkins项目
  - 配置项目在github中的路径和对应的版本分支
  - 构建触发器（定时触发、代码更新时触发）
  - 构建命令：python run_suite.py（在jenkins项目工作目录下执行）
  - 构建后：
    - 配置测试报告的路径
    - 配置测试报告的模板格式、收件人、触发器

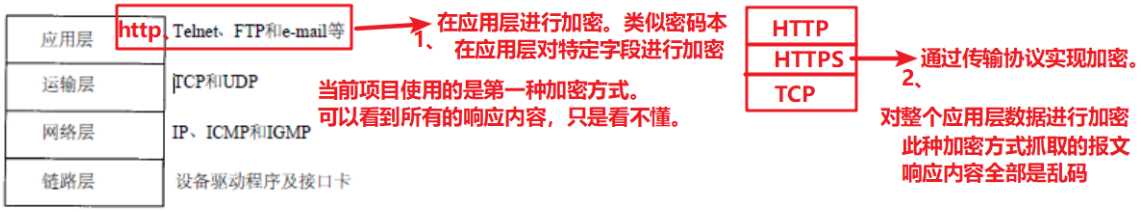
学习目标

- 执行系统测试用例，并提交bug
- 使用测试报告模板，编写测试报告
- 金融项目测试总结

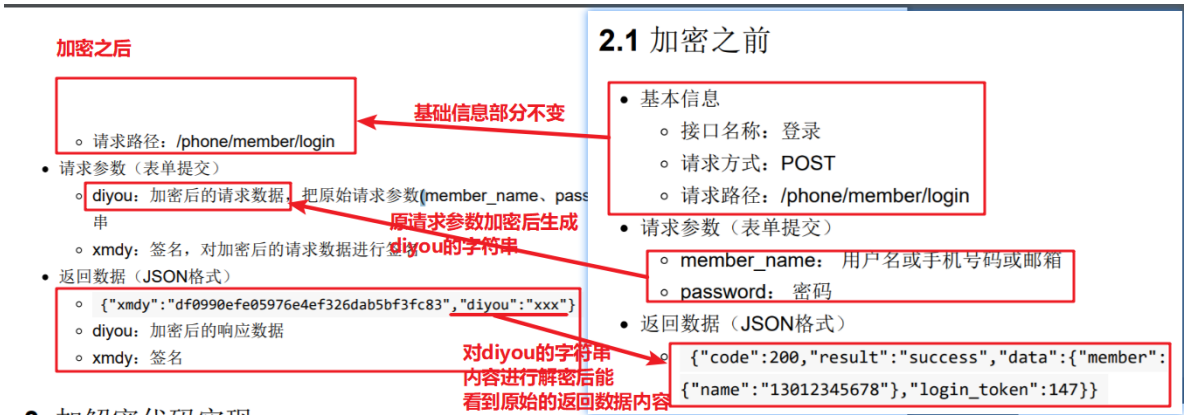
接口加解密：

介绍：

加解密的方法介绍：



P2P项目加密接口的介绍



加密的流程介绍

（了解即可）

只适用加密算法的问题：

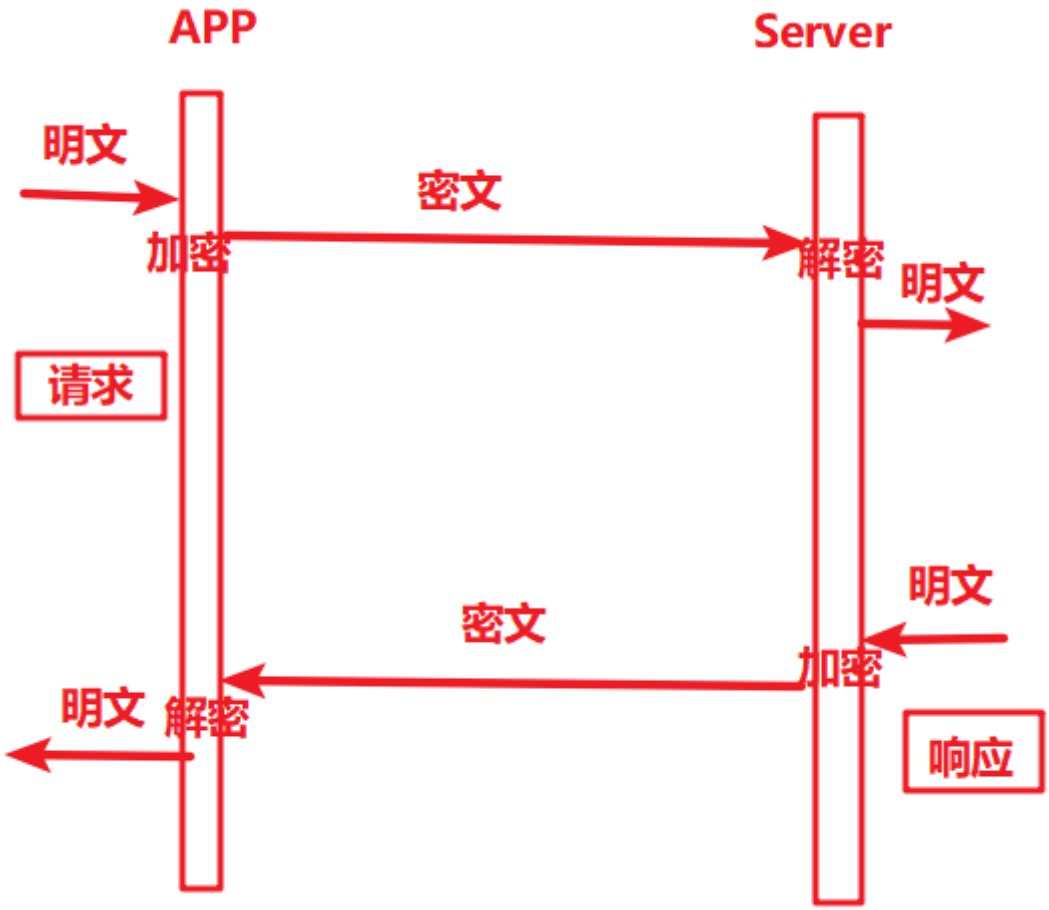
问题：加密算法都是公开的，如果有人穷举所有的加密算法来进行解析，加密数据就不安全

签名作用：

- 与原始数据混淆在一起，防止有人逆向解密后，直接看到原始数据的文件
- 防止数据在传输过程中被篡改，通过签名在收到数据后进行核对的工作

加解密接口测试时的要点：

核心就是：编写出与应用服务器相匹配的加密和解密的方法



加解密接口测试：

加解密的接口测试：

- 找开发来要加解密的方法（因为项目代码中一定有对应的加解密方法）

关键方法使用：

- get_diyou(data)： data就是原始的请求参数
- get_xmdy(data)：data是get_diyou()方法返回的结果
- decrypt_data(data)： data是响应结果中diyou字段对应的值

```python
# 加解密工具类
class EncryptUtil:
    # 发送请求时，加密密码
    SEND_AES_KEY = ";3jm$>/p-ED^cVz_j~.KV&V)k9jn,UAH"
    # 发送请求时，签名密钥
    SEND_SIGN_KEY = "DY34fdgsWET@#$%wg#@4fgd345sg"
    # 接收数据时，解密密钥
    RECEIVE_AES_KEY = "54Ms5bkE6UEdyrRviJO![OR]g+i79x]k"

    @staticmethod
    def padding_pkcs5(value):
        BS = AES.block_size
        return str.encode(value + (BS - len(value) % BS) * chr(BS - len(value) %
BS))

    # 替换空字符
    @staticmethod
    def replace_blank(str_data):
        str_data = re.compile("\t|\r|\n").sub("", str_data)
        print("replace_blank str_data=", str_data)
        return str_data

    @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)

        # 加密
        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

    @staticmethod
    def md5value(data):
        print("md5value data=", data)
        md5 = hashlib.md5()
        md5.update(data.encode())
        return md5.hexdigest()

    @staticmethod
    def get_diyou(data):
        # 把字典转换为JSON字符串
        if isinstance(data, dict):
            data = json.dumps(data)
        aes_encrypt_data = EncryptUtil.aes_encrypt(EncryptUtil.SEND_AES_KEY,
data)
        return EncryptUtil.replace_blank(aes_encrypt_data)

    @staticmethod
    def get_xmdy(data):
        return EncryptUtil.md5value(
            EncryptUtil.SEND_SIGN_KEY + EncryptUtil.replace_blank(data) +
EncryptUtil.SEND_SIGN_KEY)

    @staticmethod
    def decrypt_data(data):
        return EncryptUtil.aes_decrypt(EncryptUtil.RECEIVE_AES_KEY, data)
```

编写加解密接口的测试脚本：

步骤：

1、准备参数

2、对参数进行加密

3、发送加密后的参数请求

4、接收响应，并对响应进行解密

5、对解密后的数据进行断言

```
 def test01_index(self):
        #1、请求参数
        get_index_url = app.MOBILE_URL + "/phone/index/index"
        req_data = {}
        #2、对参数进行加密
        diyou = EncryptUtil.get_diyou(req_data)
        xmdy = EncryptUtil.get_xmdy(diyou)
```

```
        #3、发送请求
        req_param = {"diyou":diyou,"xmdy":xmdy}
        response = requests.post(get_index_url,data=req_param)
        logging.info("response = {}".format(response.json()))
        #4、接收响应数据，并对响应数据进行解密
        diyou_data = response.json().get("diyou")
        decryted_data = EncryptUtil.decrypt_data(diyou_data)
        #5、对解密后的响应数据进行断言
        #将json格式转换为字典
        data = json.loads(decryted_data)
        self.assertEqual(200,data.get("code"))
        self.assertEqual("success",data.get("result"))
```

编写测试脚本的优化：

- 先定义发送加密请求的基础方法：

```
def encryted_Request(url,req_data):
    # 对数据进行加密
    diyou = EncryptUtil.get_diyou(req_data)
    xmdy = EncryptUtil.get_xmdy(diyou)
    # 发送请求
    req_param = {"diyou": diyou, "xmdy": xmdy}
    response = requests.post(url, data=req_param)
    # 接收响应并解密
    diyou_data = response.json().get("diyou")
    data = EncryptUtil.decrypt_data(diyou_data)
    result = json.loads(data)
    # 返回解密后结果
    return result
```

- 写测试脚本：在测试脚本中传递参数，调用封装的方法，并进行断言

```
    def test03_login(self):
        #准备参数
        login_url = app.MOBILE_URL + "/phone/member/login"
        req_data = {"member_name": "13012345678", "password": "test123"}
        #调用封装的发送加密数据的接口
        data = encryted_Request(login_url,req_data)
        #对结果进行断言
        self.assertEqual(200,data.get("code"))
        self.assertEqual("success",data.get("result"))
```

系统测试执行要点：

正常情况下公司中系统测试分为多轮进行：

- 第一轮通常是需要执行所有的系统测试用例 （必须有）
    - 执行过程中的所有缺陷需要及时提交缺陷报告
- 第二轮通常是根据特定的策略来选择部分的系统测试用例进行执行
    - 特定的策略通常由测试主管来制定

    - 策略一般有：优先级（高、中）、模块（风险高的模块）、根据用例的类型（功能、稳定性、兼容性）。。。。。
- 第三轮通常是发散测试
    - 常见的形式是：组内交叉测试（正常情况下A负责模块1，B负责模块2；在交叉测试时，由B测试模块1，A测试模块2，而且不需要按照用例来执行）
- 最后为了保险起见，发布前有时还会针对所有Level 0的用例，再进行一轮测试，保证发布的系统基本功能可用

bug定位的介绍:

BUG定位的要求：

- 基本要求：定位出bug是前端bug还是后端bug
- 更高要求：bug所属模块、bug的代码级别（不要求）

BUG定位的价值

- 找到BUG的本质(找到必现路径)
- 提升开发修复BUG的效率
- 提升自身的逻辑思维与技术能力

bug定位的时间安排：

- 优先需要进行所有测试用例的执行，找到bug，并提交bug
- 在不耽误执行进度的前提下，可以花费时间来尽可能定位出bug

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

```
例如：玩CF游戏，从高处跳下来，概率出现不掉血的情况 —— bug
定位：
（1）分析问题可能的原因：高度、重量、地形、姿势。。。。。
（2）针对每一个维度分别进行测试分析。将其他因素固定，只改变其中一个因素
（3）最终确定会产生bug的影响因素
```

技术手段

- 查看数据库
- 抓包分析
- 查看日志

```
例如：界面上添加用户，在用户列表中未看到对应的用户信息
借助与技术手段来定位：
（1）查看数据库中是否有对应的用户记录；
  - 有则说明，添加用户的操作成功，查看用户的操作失败
  - 没有则说明，添加用户的操作没有成功
（2）针对第一步确定下来失败的业务操作进行抓包，观察所有的请求和响应数据，同时抓包服务器的日志
```

- 先看响应码，如果响应中有4xx错误，通常是客户端请求有问题；如果响应中有5xx错误，通常就是服务器端有问题（大概率是bug）；如果响应是2xx，则需要进一步的分析。
- 当响应为200时，
  - 检查响应内容正确前端页面显示不出来，响应字段和前端定义的字段是否匹配，找前端人员来确定
  - 检查响应内容不正确，检查请求的参数是否正确；
    - 如果请求参数不正确，就是前端代码的问题；
    - 如果请求参数正确，但响应不正确，结合服务器日志来进行问题的定位（大概率就是后端代码的问题）

备注：服务器日志需要与开发人员确定所在的位置（可以写入日志文件，也可能直接打印在命令行窗口）

回顾-软件测试报告的目标

- 对一段的时间工作进行总结

回顾-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

编写软件测试报告

1. 软件测试报告模板
2. 收集，汇总材料
3. 按照模板文档填充测试报告的内容

评审软件测试报告

- 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

金融项目测试经验

（1）项目的测试过程是什么?

- 参与需求评审，进行需求分析
- 写测试计划，并评审
- 编写测试点和系统测试用例，并评审
- 编写接口测试用例，并评审
- 执行接口测试用例（手工和自动化执行）
- 执行系统测试用例
- 编写测试报告，并评审

（2）如何测试自己负责的功能模块？（基本问题）

从功能、性能、易用性、兼容性、。。。。。各个维度进行测试

功能测试时，首先针对每个页面中的功能点设计测试用例；如：。。。。。

然后再针对业务流程设计测试用例；如。。。。。。。

性能测试时，如：。。。。。

易用性：如：。。。。。

兼容性：如浏览器、操作系统、分辨率。。。。。。。。

（3）接口测试用例如何设计？

- 单接口的设计
  - 正向（成功）：必填参数、全部参数
  - 反向（失败）：
    - 参数错误：多参、少参（优先级低）
    - 参数数据错误：数据为空，长度范围错误、类型错误（优先级中）
    - 业务数据错误：基于业务功能返回错误异常（参考API中定义的不同的错误描述）——（优先级高）
- 多接口的设计：按照业务流程来进行测试
  - 整理出系统的业务流程
  - 把业务流程中每个动作对应接口整理出来
  - 按照业务流程的顺序进行测试

（4） 接口测试的时机/目的是什么？

- 手工执行接口测试
  - 时机：后端开发完成，但是前段没有开发完成
  - 目的：尽早发现bug
- 自动化接口测试
  - 时机：在项目测试全过程
  - 目的：保证项目质量，避免开发修改代码引入新问题

（5）如何准备接口自动化测试过程中的数据？

- 手工构造：只需要一次构造，可以长时间使用的数据
- 接口方式构造：需要多次构造，且数据相对比较复杂的情况
- 数据库方式：需要多次构造，但数据涉及表结构比较简单（数据涉及的表结构不超过两张）

（6）如何使用Jmeter编写接口自动化脚本？ —— 案例

- 会编写接口自动化脚本
  - 每个测试用例对应Jmeter中的线程组
  - 使用HTTP取样器发送请求
  - 补充用法：数据定义、响应结果的数据关联、响应断言。。。。。
- 对自动化数据需要能自动化清除
  - 配置JDBC连接池
  - 通过JDBC Request请求，执行清除数据SQL语句
- Jmeter脚本部署到持续集成

（7）如何使用代码编写接口自动化脚本？ —— 案例

- 先搭建框架（api、script、log、report、utlis、app）

- 定义日志初始化配置 和 运行脚本的套件
- 编写脚本时
    - 先定义API接口类
    - 在测试脚本中调用API的接口类

（8）如何保证接口自动化测试脚本的稳定性?

- 跑完脚本后清除测试数据
- 断言时通过连接数据库的方式获取实际的数据，而不使用固定数据

（9）项目中涉及到第三方接口时如何进行接口测试?

mock技术，模拟第三方系统接收请求，并返回响应

（10）项目测试过程中发现的印象深刻的BUG

- 发现/定位困难，如概率性bug
- 有技术含量的bug，如接口、白盒测试，通过代码分析发现的bug（不是手工容易构造的bug）
- bug沟通过程中，比较曲折，并最终解决。

以上是印象深刻的bug的思路，结合自己的项目来准备

（11）项目测试过程中遇到的问题

话外音：通过这个问题学习到了什么（跟第10题的答案基本一致）