

Python+Requests+Unittest接口测试框架项目实战

1 拉勾教育项目实战准备工作

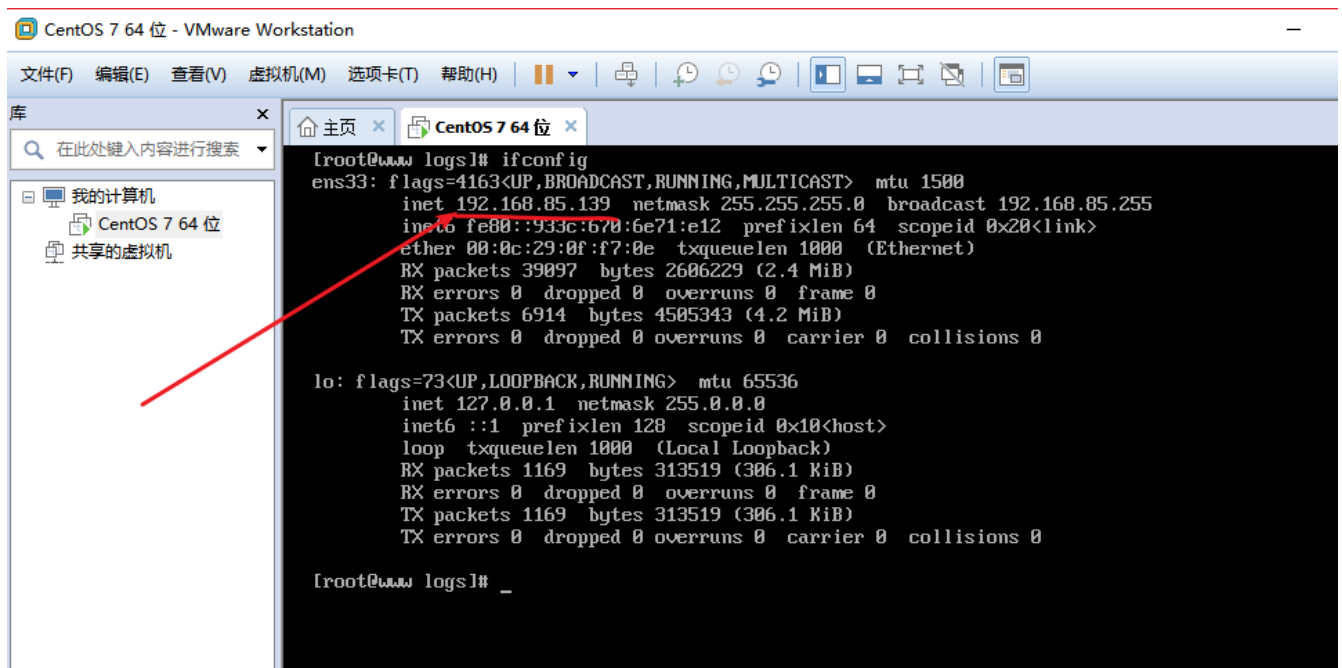
1.1 接口测试流程

接口测试一般在功能测试之前进行；

- 需求分析
- 接口文档分析
- 设计接口测试用例
- 执行接口测试
 - 搭建环境
 - Postman工具执行、Jmeter执行、**Python代码执行**
- 提交跟踪和管理缺陷
- 输出接口测试报告
- 自动化接口测试和持续集成

1.2 环境准备

使用之前用虚拟机vmware搭建的拉勾教育项目环境



服务器IP地址：192.168.85.139

1.3 获取接口文档

拉勾教育接口文档.md - Typora

文件(F) 编辑(E) 段落(P) 格式(O) 视图(V) 主题(T) 帮助(H)

文件

大纲

查找

×

拉勾教育接口文档

用户管理

- 查询用户列表
- 搜索用户
- 查询用户权限
- 登陆
- 冻结/解冻用户
- 修改用户角色权限

课程管理

- 查询所有课程
- 根据课程ID查询课程
- 新增课程
- 修改课程
- 更新课程状态
- 查询课程章节
- 新增课程章节
- 修改课程章节
- 更新课程章节状态

权限管理

- 角色列表

用户管理

查询用户列表

基本信息

- 请求方法: POST
- 资源路径: /user/findAllUserByPage
- 接口描述: 根据分页查询多个用户组成的列表信息

请求参数

请求头

参数名	参数类型	是否必填	示例	备注
Authorization	TOKEN	是		用户令牌
Content-Type	application/json	是		

查询参数

参数名	参数类型	是否必填	示例	备注

7296 词

1.4 设计接口测试用例

接口测试用例.xlsx

开始 插入 页面布局 公式 数据 审阅 视图 开发工具 会员专享 智能工具箱

查找命令、搜索模板

未同步 协作 分享

粘贴 复制 格式刷 B I U 田 背景色 前景色 边框 合并居中 自动换行 常规 条件格式 表格样式 求和 筛选 排序 填充 单元格 行和列

A2 fx 1

	A	B	C	D	E	F	G	H	I
1	编号	标题	模块	接口名称	请求方法	URL	请求头	请求数据	请求数据类型
2	1	登录成功	拉勾教育	登录	POST	/ssm_web/user/login		查询参数: phone=15321919666 password=123456	x-www-form-
3	2	密码错误	拉勾教育	登录	POST	/ssm_web/user/login		查询参数: phone=15321919666 password=error	x-www-form-
4	3	账号不存在	拉勾教育	登录	POST	/ssm_web/user/login		查询参数: phone=15121919666 password=123456	x-www-form-

单接口测试 业务场景测试 Sheet3

1.5 初始化接口测试框架

除了从零开始的项目，实际工作都会拿到一个已有的框架，在这个基础上继续编写代码即可。

本课程主要演示从零开始搭建一个接口测试框架项目

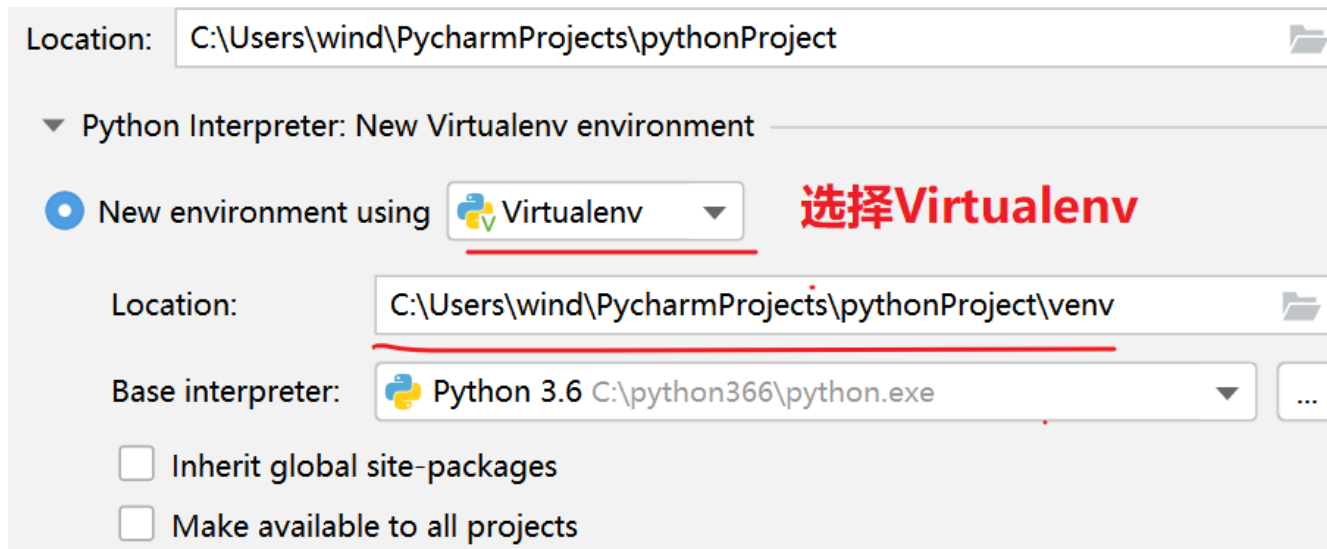
1.5.1 新建接口测试框架目录结构

virtualenv(venv)虚拟环境介绍

Python代码能够被运行的本质，是因为我们安装了Python运行环境

但是在实际项目中，为了让Python代码能够在任何操作系统中运行，我们需要把Python运行环境整合到项目中进行集中管理。所以开发发明了虚拟环境，用来单独管理该项目的python文件。



简单的说，虚拟环境就是一个新的Python环境，只不过在当前项目运行代码时，项目会优先使用内部的虚拟环境Python来运行程序。





The image shows the 'New Virtualenv Environment' dialog box in PyCharm. The 'Location' field is set to 'C:\Users\wind\PycharmProjects\pythonProject'. The 'Python Interpreter' section is expanded, showing 'New environment using' with a dropdown menu set to 'Virtualenv'. A red line underlines the 'Virtualenv' option, and the text '选择Virtualenv' (Select Virtualenv) is written in red next to it. Below this, the 'Location' field is set to 'C:\Users\wind\PycharmProjects\pythonProject\venv'. The 'Base interpreter' field is set to 'Python 3.6 C:\python366\python.exe'. There are two checkboxes at the bottom: 'Inherit global site-packages' and 'Make available to all projects', both of which are currently unchecked.

Location: C:\Users\wind\PycharmProjects\pythonProject

▼ Python Interpreter: New Virtualenv environment

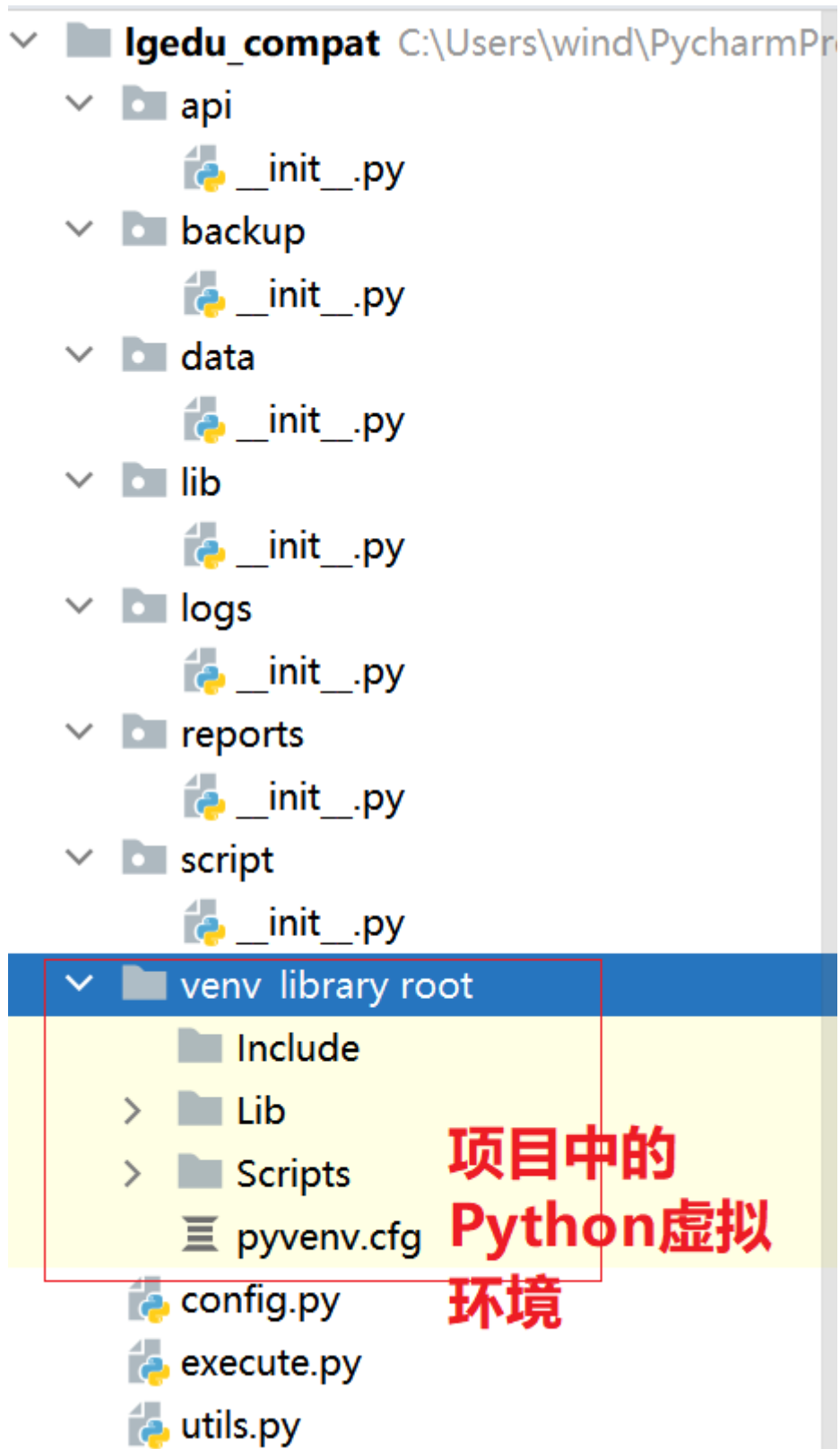
● New environment using  Virtualenv  **选择Virtualenv**

Location: C:\Users\wind\PycharmProjects\pythonProject\venv

Base interpreter:  Python 3.6 C:\python366\python.exe  ...

☐ Inherit global site-packages

☐ Make available to all projects

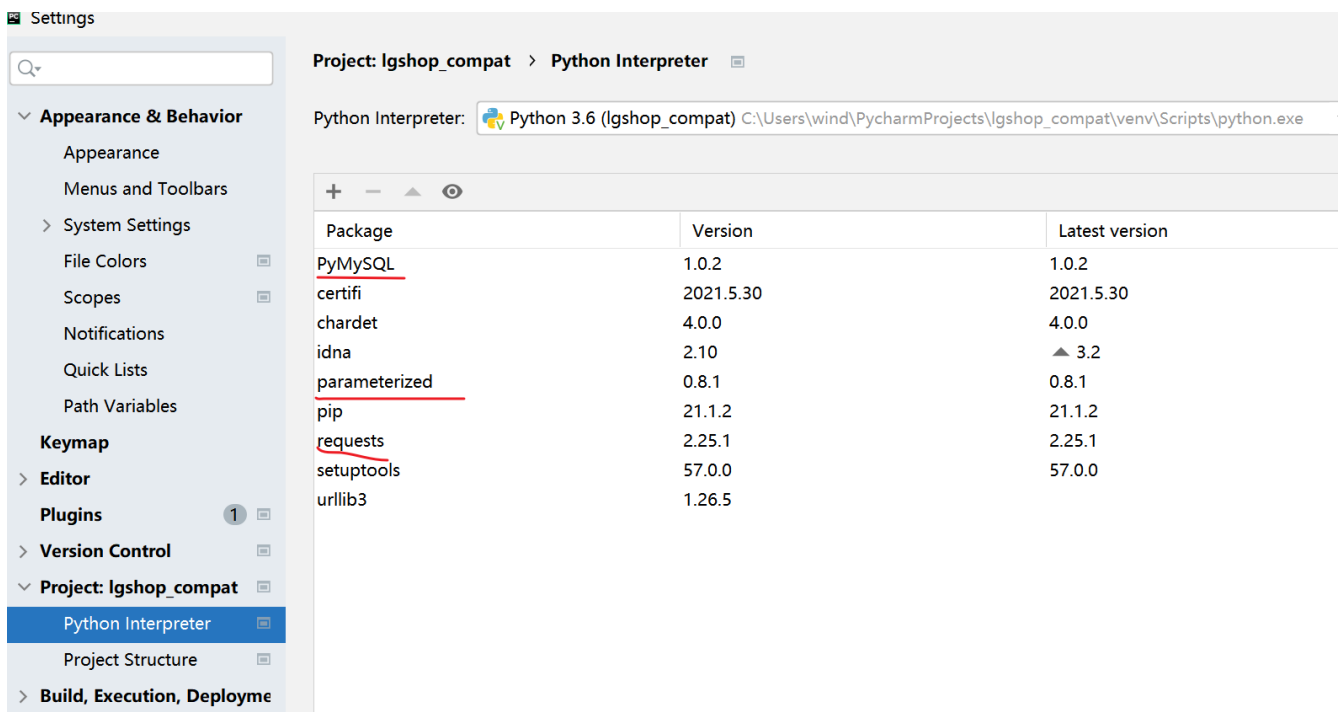


虚拟环境运行代码细节分析



1.5.2 安装需要用到的依赖包

- requests
- pymysql
- parameterized
- jsonpath
- HTMLTestRunner_PY3



后期还需要新的依赖包时，可以继续安装

1.5.3 配置公有属性

在config.py中设置公有的配置属性

- BASE_URL：拉勾教育项目基础URL

因为URL的域名、端口经常变更，所以独立出来，提高代码维护性

- BASE_PATH: 拉勾教育项目的绝对路径；该代码始终能获取到当前项目的绝对路径。
生成日志、报告、备份文件时都需要使用BASE_PATH
- BASE_HEADERS: 拉勾教育项目公有请求头

```
config.py x
1 import os.path
2
3 BASE_URL="http://192.168.85.139:8080"
4 BASE_PATH=os.path.dirname(os.path.abspath(__file__))
5 BASE_HEADERS={"Content-Type":"application/json", "Authorization":""}
6 |
```

后期还会添加更多属性到config.py中

1.5.4 初始化日志配置

在utils.py编写初始化日志的代码

```
#utils.py
# 配置logging
def init_logging():
    # 实例化日志器
    logger = logging.getLogger()
    # 设置日志等级
    logger.setLevel(logging.INFO)
    # 获取日志处理器
    # 控制台处理器
    sh = logging.StreamHandler()
    # 文件处理器 os.path.dirname(os.path.abspath(__file__)) 是获取当前utils.py的父级目录
    fh =
    logging.handlers.TimedRotatingFileHandler(os.path.dirname(os.path.abspath(__file__)) +
    "/logs/lagou.log",
                                           when='h',
                                           interval=24,
                                           backupCount=3,
                                           encoding="utf-8")

    # 设置日志格式
    fmt = "%(asctime)s [%(levelname)s] [%(funcName)s %(lineno)d] %(message)s"
    formatter = logging.Formatter(fmt)
    # 将格式添加到处理器
    sh.setFormatter(formatter)
    fh.setFormatter(formatter)
    # 将处理器添加到日志器
    logger.addHandler(sh)
    logger.addHandler(fh)
    # 返回日志器
    return logger
```

在 `api.__init__.py` 中初始化日志

```
#api.__init__.py
from utils import init_logging
import logging

# 初始化日志配置
init_logging()

# 调试打印日志
logging.info("Test日志打印")
```

2 封装拉勾教育系统接口

2.1 封装登录接口

```
import config

class UserApi:
    def __init__(self):
        self.url_login = config.BASE_URL + "/ssm_web/user/login"

    # 封装登录接口
    def login(self, session, phone, password):
        return session.post(url=self.url_login, params={"phone": phone, "password": password})
```

2.2 封装课程管理模块接口

```
import config

class CourseApi:
    def __init__(self):
        self.url_find_all_course = config.BASE_URL + "/ssm_web/course/findAllCourse"
        self.url_add_course = config.BASE_URL + "/ssm_web/course/saveOrUpdateCourse"
        self.url_modify_course = config.BASE_URL + "/ssm_web/course/saveOrUpdateCourse"
        self.url_update_status = config.BASE_URL + "/ssm_web/course/updateCourseStatus"

    # 封装查询所有课程
    def query_course(self, session, requests_body, headers):
        return session.post(url=self.url_find_all_course, json=requests_body, headers=headers)
```

```
# 封装新增课程
def add_course(self, session, requests_body, headers):
    return session.post(url=self.url_add_course, json=requests_body, headers=headers)

# 封装修改课程
def modify_course(self, session, requests_body, headers):
    return session.post(url=self.url_modify_course, json=requests_body,
headers=headers)

# 更新课程状态
def update_course_status(self, session, query_params, headers):
    return session.get(url=self.url_update_status, params=query_params,
headers=headers)
```

3 实现拉勾教育接口测试用例

3.1 实现登录接口测试用例

```
import unittest, requests
import logging
from api.edu_user import UserApi

class TestLogin(unittest.TestCase):
    session = None
    @classmethod
    def setUpClass(cls):
        # 实例化session
        cls.session = requests.Session()
        # 实例化edu_user
        cls.user_api = UserApi()

    @classmethod
    def tearDownClass(cls):
        # 关闭session
        if cls.session:
            cls.session.close()

    def test01_login_success(self):
        # 登录
        response = self.user_api.login(self.session, '15321919666', '123456')
        # 打印
        logging.info(f"登录成功用例登录的结果为: {response.json()}")
        # 断言
        self.assertEqual("响应成功", response.json().get("message"))
        self.assertEqual(200, response.status_code)
```


3.2 实现课程管理模块业务场景接口测试

```
import unittest, requests
import logging

import jsonpath

import config
from api.edu_course import CourseApi

class TestCourse(unittest.TestCase):
    session = None

    @classmethod
    def setUpClass(cls):
        # 实例化session
        cls.session = requests.Session()
        # 实例化edu_user
        cls.course_api = CourseApi()

    @classmethod
    def tearDownClass(cls):
        # 关闭session
        if cls.session:
            cls.session.close()

    def test01_course_manager(self):
        # 查询课程
        response = self.course_api.query_course(self.session, {}, config.BASE_HEADERS)
        logging.info(f"查询所有课程结果为: {response.json()}")

        # 断言
        self.assertEqual("响应成功", response.json().get("message"))
        self.assertEqual(200, response.status_code)
        # 断言id为15的brief是海量大数据课程
        self.assertEqual("大数据云计算", jsonpath.jsonpath(response.json(), "$.content[?(@.id==15)].courseName")[0] )

        # 添加课程
        add_data = {"courseName": "测试开发技术巅峰之路", "brif": "从小白走向测试开发",
"previewFirstField": "测试开发"}
        response = self.course_api.add_course(self.session, add_data,
config.BASE_HEADERS)
        logging.info(f"添加课程结果为: {response.json()}")
        # 断言
        self.assertEqual("响应成功", response.json().get("message"))
        self.assertEqual(200, response.status_code)
        # 修改课程
        modify_data = {"id": "27", "courseName": "一拳超人成为了测试开发"}
        response = self.course_api.modify_course(self.session, modify_data,
config.BASE_HEADERS)
```

```

logging.info(f"修改课程结果为: {response.json()}")
# 断言
self.assertEqual("响应成功", response.json().get("message"))
self.assertEqual(200, response.status_code)

# 更新课程状态
params = {"status": 1, "id": "27"}
response = self.course_api.update_course_status(self.session, params,
config.BASE_HEADERS)
logging.info(f"更新课程状态为1, 结果为: {response.json()}")
# 断言
self.assertEqual("响应成功", response.json().get("message"))
self.assertEqual(200, response.status_code)

```

3.3 JsonPath提取数据并断言

断言查询所有课程接口中，返回的json响应数据里面，id为15的courseName是大数据云计算

```

self.assertEqual("大数据云计算", jsonpath.jsonpath(response.json(), "$.content[?
(@.id==15)].courseName")[0] )

```

4 实现数据驱动测试

4.1 实现登录接口数据驱动测试

设计数据文件

```

[
  {
    "case_name": "登录成功",
    "phone": "15321919666",
    "password": "123456",
    "response_code": 200,
    "message": "响应成功"
  },
  {
    "case_name": "密码错误",
    "phone": "15321919666",
    "password": "1234567",
    "response_code": 200,
    "message": "用户名密码错误"
  },
  {
    "case_name": "手机号码没有注册",
    "phone": "15421919666",

```

```

        "password": "123456",
        "response_code": 200,
        "message": "用户名密码错误"
    },
    {
        "case_name": "密码为空",
        "phone": "15321919666",
        "password": "",
        "response_code": 200,
        "message": "用户名密码错误"
    },
    {
        "case_name": "手机号码为空",
        "phone": "",
        "password": "123456",
        "response_code": 200,
        "message": "用户名密码错误"
    }
]

```

编写读取数据文件的方法

```

def read_json_data(filename):
    """
    :param filename: 数据文件的路径和名称
    :return:
    """
    with open(filename, mode='r', encoding="utf-8") as f: # 是一个IO类型不是JSON数据，所以无法处理
        jsonData = json.load(f) # 转化io对象f为jsonData对象

        result_list = []
        for case_data in jsonData:
            result_list.append(tuple(case_data.values())) # 转化json文件中的数据为列表元组数据，这个数据是parameterized工具所需要的
            print(result_list)

        return result_list # 需要返回结果给parameterized.expand()

```

实现参数化和数据驱动

```

import unittest, requests
import logging

import config
from api.edu_user import UserApi
from utils import read_json_data
from parameterized import parameterized

```

```

class TestLogin(unittest.TestCase):
    session = None
    @classmethod
    def setUpClass(cls):
        # 实例化session
        cls.session = requests.Session()
        # 实例化edu_user
        cls.user_api = UserApi()

    @classmethod
    def tearDownClass(cls):
        # 关闭session
        if cls.session:
            cls.session.close()
    @parameterized.expand(read_json_data(config.BASE_PATH + "/data/login_data.json"))
    def test01_login_success(self, case_name, phone, password, response_code, message):
        # 登录
        response = self.user_api.login(self.session, phone, password)
        # 打印
        logging.info(f"{case_name}用例登录的结果为: {response.json()}")
        # 断言
        self.assertEqual(message, response.json().get("message"))
        self.assertEqual(response_code, response.status_code)

```

5 生成测试报告

```

# 导包
import time
import unittest

import lib.HTMLTestRunner_PY3
from script.test_login import TestLogin
from script.test_course import TestCourse
# 创建测试套件
suite = unittest.TestSuite()
# 将测试用例添加到测试套件
suite.addTests(unittest.makeSuite(TestLogin))
suite.addTests(unittest.makeSuite(TestCourse))
# 使用HTMLTestRunner运行测试用例，生成测试报告
with open("./reports/report{}.html".format(time.strftime('%Y%m%d %H%M%S')), 'wb') as f: #
    打开要生成的测试报告文件
    # 实例化runner
    runner = lib.HTMLTestRunner_PY3.HTMLTestRunner(f)
    # 使用runner运行测试套件，得到测试结果
    result = runner.run(suite)
    # 使用runner生成测试报告

```

```
runner.generateReport(suite, result)
```

6 登录态管理

6.1 手动管理令牌

将获取到的令牌手动填写到config.HEADERS的值中

内部接口引用config.HEADERS即可

6.2 自动管理令牌

在utils.py中编写登录接口方法，登录成功后自动获取令牌并设置到config.HEADERS中

内部接口引用config.HEADERS即可

7 多线程执行测试用例

由于Unittest框架不支持多线程，所以修改Unittest框架实现多线程运行测试用例的成本太高，这里我们不考虑。我们可以在HTMLTestRunner的基础上，使用多线程运行测试用例

HTMLTestRunner内部是根据运行测试用例后的TestResult对象来生成测试报告的，

所以使用多线程运行时，我们可以收集多个HTMLTestRunner的结果对象，然后整合成一个结果，最后生成测试报告。

多线程运行测试用例，得到多个result对象；可以把result对象添加到一个列表，然后通过遍历列表来实现整合测试报告，得到多线程执行测试用例的结果

8 服务端备份/还原/初始化

8.1 封装备份/还原/初始化的方法

```
class DataManager:
    """
    管理被测试服务器的数据，一般先备份、在初始化数据、测试结束后，还原数据
    """

    def __init__(self):
        pass

    @classmethod
    def init_data(self):
        """
        初始化数据到测试之前的状态
        :return:
        """
        print("初始化测试数据...")
        command = os.path.abspath(config.MYSQL_HOME + "/bin/mysql")
        print(command)
        test_data = config.BASE_DIR + "/backup/test_data.sql"
        sql = os.system(
            f"{command} -h {config.MYSQL_HOST} -u root -proot ssm_lagou_edu <
{test_data}")
        print("初始化测试数据结束...")
```

```

    @classmethod
    def back_data(self):
        """
        备份现有数据
        :return:
        """
        print("开始备份数据...")
        command = os.path.abspath(config.MYSQL_HOME + "/bin/mysqldump")
        print(command)
        backup = config.BASE_DIR + "/backup/backdb.sql"
        sql = os.system(
            f"{command} -u root -h {config.MYSQL_HOST} -proot ssm_lagou_edu course>
{backup}")
        print(f"备份数据结束，备份的数据文件名称：{backup}")

    @classmethod
    def recovery_data(self):
        """
        恢复数据
        :return:
        """
        print("开始恢复数据...")
        command = os.path.abspath(config.MYSQL_HOME + "/bin/mysql")
        print(command)
        test_data = config.BASE_DIR + "/backup/test_data.sql"
        sql = os.system(f"{command} -h {config.MYSQL_HOST} -u root -proot ssm_lagou_edu <
{test_data}")
        print("恢复数据结束...")

```

8.2 执行用例之前备份/初始化/还原

```

import unittest, requests
import logging

import jsonpath

import config
from api.edu_course import CourseApi
from utils import DataManager

class TestCourse(unittest.TestCase):
    session = None
    # 实例化Datamanager
    data_manager = DataManager()
    @classmethod
    def setUpClass(cls):
        # 实例化session
        cls.session = requests.Session()
        # 实例化edu_user
        cls.course_api = CourseApi()

```

```

# 备份数据
cls.data_manager.back_data()
# 初始化数据
cls.data_manager.init_data()

@classmethod
def tearDownClass(cls):
    # 关闭session
    if cls.session:
        cls.session.close()
    # 还原数据
    cls.data_manager.recovery_data()

def test01_course_manager(self):
    # 查询课程
    response = self.course_api.query_course(self.session, {}, config.BASE_HEADERS)
    logging.info(f"查询所有课程结果为: {response.json()}")

    # 断言
    self.assertEqual("响应成功", response.json().get("message"))
    self.assertEqual(200, response.status_code)
    # 断言id为15的brief是海量大数据课程
    self.assertEqual("大数据云计算", jsonpath.jsonpath(response.json(), "$.content[?(@.id==15)].courseName")[0] )
    # 添加课程
    add_data = {"courseName": "测试开发技术巅峰之路", "brif": "从小白走向测试开发",
"previewFirstField": "测试开发"}
    response = self.course_api.add_course(self.session, add_data,
config.BASE_HEADERS)
    logging.info(f"添加课程结果为: {response.json()}")
    # 断言
    self.assertEqual("响应成功", response.json().get("message"))
    self.assertEqual(200, response.status_code)
    # 修改课程
    modify_data = {"id": "27", "courseName": "一拳超人成为了测试开发"}
    response = self.course_api.modify_course(self.session, modify_data,
config.BASE_HEADERS)
    logging.info(f"修改课程结果为: {response.json()}")
    # 断言
    self.assertEqual("响应成功", response.json().get("message"))
    self.assertEqual(200, response.status_code)

    # 更新课程状态
    params = {"status": 1, "id": "27"}
    response = self.course_api.update_course_status(self.session, params,
config.BASE_HEADERS)
    logging.info(f"更新课程状态为1, 结果为: {response.json()}")
    # 断言
    self.assertEqual("响应成功", response.json().get("message"))
    self.assertEqual(200, response.status_code)

```


9 【扩展】Flask 实现Mock测试

9.1 Mock测试的概念

把一些难以测试的对象，通过虚拟的方式完成测试的过程，就叫做Mock测试。

在互联网行业，一些公司也把Mock测试叫做挡板。

Mock测试的优点：

- 解除接口的依赖
- 替换速度较慢的接口
- 模拟的难以产生的异常信息
- 辅助测试人员更早介入测试

Mock测试的缺点：

- 由于Mock构造了虚假的数据，所以测试结果并不是全部准确的，会出现一部分链路没有测试到的场景。
所以实现Mock测试后，还需要在正常的环境中，重复验证。

9.2 Mock的实现方法

- 代码实现
- 工具实现

一般的Mock都是通过代码来实现。实现Mock时，需要清楚代码的内部逻辑，才能实现。

所以一般都是由开发人员编写Mock代码。

9.3 Flask实现替换拉勾教育登录接口

场景一：现在假设拉勾教育还有上线，但是接口文档已经有了。

那么我们怎么提高工作效率，让我们在上线之前就能够编写测试代码，实现接口测试的代码编写？

```
# 导包
from flask import Flask, request

# 实例化flask对象
app = Flask(__name__)

# 实现拉勾教育登录接口
@app.route("/ssm_web/user/login", methods=['POST'])
def login():
```

```
# 完成登录, 并返回响应数据
# 获取查询参数
params = request.args
if not params.get("phone") == "15321919666":
    return '{"success": True,"state": 1,"message": "用户名密码错误","content": None}'
if not params.get("password") == "123456":
    return '{"success": True,"state": 1,"message": "用户名密码错误","content": None}'

response_data = {
    "success": True,
    "state": 1,
    "message": "响应成功",
    "content": {
        "access_token": "e26d5ba4-24a4-455c-b683-8a11bc78053c",
        "user_id": 100030011,
        "user": {
            "id": 100030011,
            "name": "15321919666",
            "portrait": "https://edu-lagou.oss-cn-
beijing.aliyuncs.com/images/2020/06/28/15933251448762927.png",
            "phone": "15321919666",
            "password": "123456",
            "reg_ip": None,
            "account_non_expired": None,
            "credentials_non_expired": None,
            "account_non_locked": None,
            "status": "DISABLE",
            "is_del": None,
            "createTime": 1594347555000,
            "updateTime": 1594347555000
        }
    }
}

return response_data

if __name__ == '__main__':
    app.run()
```