Table of Contents

接口测试课程

课程目标

- 能够根据接口API文档编写接口测试用例
- 能够使用Postman工具进行接口测试，并能够对大量接口用例进行管理、对接口响应结果进行断言、处理多接口的依赖及生成测试报告
- 能够使用Python+Requests封装的接口测试框架，实现接口对象封装、测试用例编写、测试数据管理及生成测试报告

课程大纲

章节	知识点	
第1章 接口测试基础	1.接口及接口测试概念 2.HTTP协议 3.接口规范	4.接口测试流程 5.项目环境说明
第2章 Postman实现接口测试	1.Postman介绍和安装 2.Postman基本用法 3.Postman高级用法	4.Postman测试报告 5.项目实战
第3章 数据库操作	1. 数据库介绍 2. 数据库基本操作	3. 数据库事务操作
第4章 代码实现接口测试	1. Requests库 2. 集成UnitTest	3. 接口测试框架开发 4. 项目实战
第5章 持续集成	1. 持续集成介绍 2. Git与Git代码托管平台 3. Jenkins	4. 持续集成之Postman 5. 持续集成之代码
第6章 接口测试扩展	1. 接口Mock测试	2. 接口测试总结

代码实现接口测试

目标

1. 掌握如何使用Requests库
2. 掌握如何使用UnitTest管理接口测试脚本
3. 掌握如何设计接口自动化测试框架

Requests库

目标

1. 了解什么是Requests库
2. 掌握如何使用Requests发送请求
3. 掌握如何使用Requests传递URL参数
4. 掌握如何使用Requests获取响应内容
5. 掌握如何使用Requests设置请求头
6. 掌握如何使用Requests操作Cookie和Session

---

1. 什么是Requests库？

1.1 介绍

Requests库是用Python编写的，基于urllib，采用Apache2 Licensed开源协议的HTTP库；
相比urllib库，Requests库更加方便，可以节约我们大量的工作，完全满足HTTP测试需求；

1.2 安装

```
pip install requests
```

---

2. 发送请求

> 常见的HTTP请求方式：GET、POST、PUT、DELETE

使用 requests 发送网络请求非常简单，只需要调用HTTP请求类型所对应的方法即可。

2.1 GET请求

```
# 导包
import requests

# 发送GET请求
response = requests.get("http://www.baidu.com")
```

请求方法的返回值response为 `Response` 对象，可以从这个对象中获取响应信息。比如： `response.text` 可以获取响应的文本内容

2.2 POST请求

```
response = requests.post(url, data=None, json=None)
    """
    :param url: 请求的URL
    :param data: (可选) 要发送到请求体中的字典、元组、字节或文件对象
    :param json: (可选) 要发送到请求体中的JSON数据
    :rtype: requests.Response
    """
```

案例一

需求：

1. 请求TPshop项目的登录接口，请求数据（username: 13088888888, password: 123456, verify_code: 1234）
2. 登录接口URL：http://localhost/index.php?m=Home&c=User&a=do_login

实现分析：

- 请求方式：POST
- 请求路径： `http://localhost/index.php?m=Home&c=User&a=do_login`
- 请求数据：请求体-表单提交（username: 13088888888, password: 123456, verify_code: 1234）

示例代码：

```
import requests

# 发送POST请求
data = {"username": "13088888888", "password": "123456", "verify_code": "1234"}
response = requests.post("http://localhost/index.php?m=Home&c=User&a=do_login", data=da
ta)

# 获取响应内容
print("text=", response.text)
```

案例二

需求：

1. 请求IHRM项目的登录接口，请求数据（ `{"mobile":"13800000002", "password":"123456"}` ）
2. 登录接口URL：http://182.92.81.159/api/sys/login

实现分析：

- 请求方式：POST
- 请求路径：`http://182.92.81.159/api/sys/login`
- 请求数据：请求体-JSON数据（`{"mobile":"13800000002", "password":"123456"}`）

示例代码：

```
import requests

# 发送POST请求
data = {"mobile": "13800000002", "password": "123456"}
response = requests.post("http://182.92.81.159/api/sys/login", json=data)

# 获取响应内容
print("text=", response.text)
```

2.3 其他请求类型

其他 HTTP 请求类型，比如：PUT、DELETE、HEAD 以及 OPTIONS。

```
import requests

response = requests.put("http://www.baidu.com", data={"key": "value"})
response = requests.delete("http://www.baidu.com")
response = requests.head("http://www.baidu.com")
response = requests.options("http://www.baidu.com")
```

---

3. 传递URL参数

如果需要在URL的查询字符串中传递数据，可以使用params参数来定义，params传递的参数可以是字符串或字典。

```
response = requests.get(url, params=None)

response = requests.post(url, params=None, data=None)
```

案例

需求：

1. 访问TPshop搜索商品的接口，通过查询字符串的方式传递搜索的关键字`iPhone`，并查看响应数据

2. 请求路径格式为：`http://localhost/Home/Goods/search.html?q=iPhone`

实现分析：

- 请求方式：GET
- 请求路径：`http://localhost/Home/Goods/search.html`
- 传参方式：查询字符串（q=iPhone）

示例代码：

```
import requests

# 方式一：直接定义在URL中
# response = requests.get("http://localhost/Home/Goods/search.html?q=iPhone")

# 方式二：传递字符串类型的参数
# response = requests.get("http://localhost/Home/Goods/search.html", params="q=iPhone")

# 方式三：传递字典类型的参数
response = requests.get("http://localhost/Home/Goods/search.html", params={"q": "iPhone"
})

# 获取响应结果
print("url=", response.url)
```

4. 响应内容

请求方法的返回值*response*为 Response 对象，我们可以从这个对象中获取所有我们想要的响应信息。

```
response.status_code 状态码
response.url         请求url
response.encoding    查看响应头部字符编码
response.headers     头信息
response.cookies     cookie信息
response.text        文本形式的响应内容
response.content     字节形式的响应内容
response.json()      JSON形式的响应内容
```

案例

```
1). 访问百度首页的接口`http://www.baidu.com`，获取以下响应数据
2). 获取响应状态码
3). 获取请求URL
```

```
4). 获取响应字符编码
5). 获取响应头数据
6). 获取响应的cookie数据
7). 获取文本形式的响应内容
8). 获取字节形式的响应内容
```

4.1 JSON 响应内容

如果请求响应的内容为JSON格式的数据，则可以直接调用 `response.json()` 方法获取数据，因为requests中内置了JSON解码器，帮助我们处理JSON数据。

```
json_data = response.json()
```

注意：如果 JSON 解码失败，response.json() 就会抛出一个异常

案例

```
1). 访问查询天气信息的接口，并获取JSON响应数据
2). 接口地址：http://www.weather.com.cn/data/sk/101010100.html
```

---

5. 设置请求头

> 思考：发送HTTP请求时，传递参数的方式有哪些？

如果需要为请求添加请求头数据，只需要传递一个字典类型的数据给 headers 参数就可以了

```
headers = {"area": "010"}
response = requests.get(url, headers=headers)
```

案例

需求：

1. 请求IHRM项目的登录接口，URL： `http://182.92.81.159/api/sys/login`
2. 请求头： `Content-Type: application/json`
3. 请求体： `{"mobile":"13800000002", "password":"123456"}`

示例代码：

```
import requests

# 发送请求
url = "http://182.92.81.159/api/sys/login"
```

```
login_data = {"mobile": "13800000002", "password": "123456"}
headers = {"Content-Type": "application/json"}
response = requests.post(url, json=login_data, headers=headers)

# 获取响应内容
print("json data=", response.json())
```

---

6. Cookie [了解]

获取响应信息中的cookie数据：

```
cookies = response.cookies
```

发送请求时添加cookie数据，可以使用 cookies 参数：

```
requests.get(url, cookies={"c1": "v1"})
```

案例

需求：

1. 使用requests库调用TPshop登录功能的相关接口，完成登录操作
2. 登录成功后获取‘我的订单’页面的数据

案例实现分析：

- 相关接口：
  - 获取验证码：http://localhost/index.php?m=Home&c=User&a=verify
  - 登录：http://localhost/index.php?m=Home&c=User&a=do_login
  - 我的订单：http://localhost/Home/Order/order_list.html
- 获取cookie数据：
  - response.cookies
- 添加cookie数据：
  - requests.post(url, cookies={"c1": "v1"})

---

7. Session

session对象代表一次用户会话：从客户端浏览器连接服务器开始，到客户端浏览器与服务器断开

会话能让我们在跨请求时候保持某些参数，比如在同一个 session 实例发出的所有请求之间保持 cookie

创建**session**对象

```
session = requests.Session()
```

得到session对象后，就可以调用该对象中的方法来发送请求。

案例

需求：

1. 使用requests库调用TPshop登录功能的相关接口，完成登录操作
2. 登录成功后获取‘我的订单’页面的数据
3. 要求：使用Session对象来实现

示例代码：

```
import requests

# 获取验证码
session = requests.Session()
response = session.get("http://localhost/index.php?m=Home&c=User&a=verify")
print(response.cookies)

# 登录
login_data = {"username": "13012345678", "password": "123456", "verify_code": "8888"}
response = session.post("http://localhost/index.php?m=Home&c=User&a=do_login", data=log
in_data)
print(response.cookies)
print("login response data=", response.json())

# 我的订单
response = session.get("http://localhost/Home/Order/order_list.html")
print(response.text)
```

集成UnitTest

目标

1. 掌握如何使用UnitTest管理接口测试脚本

---

1. 集成UnitTest

将接口测试脚本集成到UnitTest单元测试框架中，利用UnitTest的功能来运行接口测试用例。

1.1 使用UnitTest框架的目的

1. 方便管理和维护多个测试用例
2. 提供丰富的断言方法
3. 能够生成测试报告

---

2. 案例

2.1 需求

使用TPShop项目完成对登录功能的接口测试

2.2 用例设计

ID	模块	用例名称	接口名称	请求URL	请求类型	请求参数类型	请求参数	预期结果	测试结果
001	登录	登录成功	获取验证码	http://localhost/index.php?m=Home&c=User&a=verify	GET			获取到验证码图片 响应头：Content-Type=image/png	
			登录	http://localhost/index.php?m=Home&c=User&a=do_login	POST	form	"username": "13012345678", "password": "123456", "verify_code": "8888"	登录成功，状态码：200 返回数据：{"status": 1, "msg": "登陆成功", "result": {}}	
002	登录	账号不存在	获取验证码	http://localhost/index.php?m=Home&c=User&a=verify	GET			获取到验证码图片 响应头：Content-Type=image/png	
			登录	http://localhost/index.php?m=Home&c=User&a=do_login	POST	form	"username": "13088888888", "password": "123456", "verify_code": "8888"	账号不存在，状态码：200 返回数据：{"status": -1, "msg": "账号不存在!"}	
003	登录	密码错误	获取验证码	http://localhost/index.php?m=Home&c=User&a=verify	GET			获取到验证码图片 响应头：Content-Type=image/png	
			登录	http://localhost/index.php?m=Home&c=User&a=do_login	POST	form	"username": "13012345678", "password": "error", "verify_code": "8888"	密码错误，状态码：200 返回数据：{"status": -2, "msg": "密码错误!"}	

接口URL：

- 获取验证码： `http://localhost/index.php?m=Home&c=User&a=verify`
- 登录： `http://localhost/index.php?m=Home&c=User&a=do_login`

2.3 示例代码

```
import requests
import unittest


class TestLogin(unittest.TestCase):
    def setUp(self):
        self.session = requests.Session()

        self.verify_url = "http://localhost/index.php?m=Home&c=User&a=verify"
        self.login_url = "http://localhost/index.php?m=Home&c=User&a=do_login"

    def tearDown(self):
        self.session.close()

    def test_login_success(self):
        # 获取验证码
        response = self.session.get(self.verify_url)
        print("type=", response.headers.get("Content-Type"))
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        data = {"username": "13012345678", "password": "123456", "verify_code": "8888"}
        response = self.session.post(self.login_url, data=data)
        result = response.json()
        print("login response data=", result)

        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(1, result.get("status"))
        self.assertEqual("登陆成功", result.get("msg"))

    def test_login_username_is_not_exist(self):
        # 获取验证码
        response = self.session.get(self.verify_url)
        print("type=", response.headers.get("Content-Type"))
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        data = {"username": "13088889999", "password": "123456", "verify_code": "8888"}
        response = self.session.post(self.login_url, data=data)
        result = response.json()
        print("login response data=", result)
```

```
        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(-1, result.get("status"))
        self.assertIn("账号不存在", result.get("msg"))

    def test_login_pwd_is_error(self):
        # 获取验证码
        response = self.session.get(self.verify_url)
        print("type=", response.headers.get("Content-Type"))
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        data = {"username": "13012345678", "password": "error", "verify_code": "8888"}
        response = self.session.post(self.login_url, data=data)
        result = response.json()
        print("login response data=", result)

        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(-2, result.get("status"))
        self.assertIn("密码错误", result.get("msg"))
```

接口测试框架开发

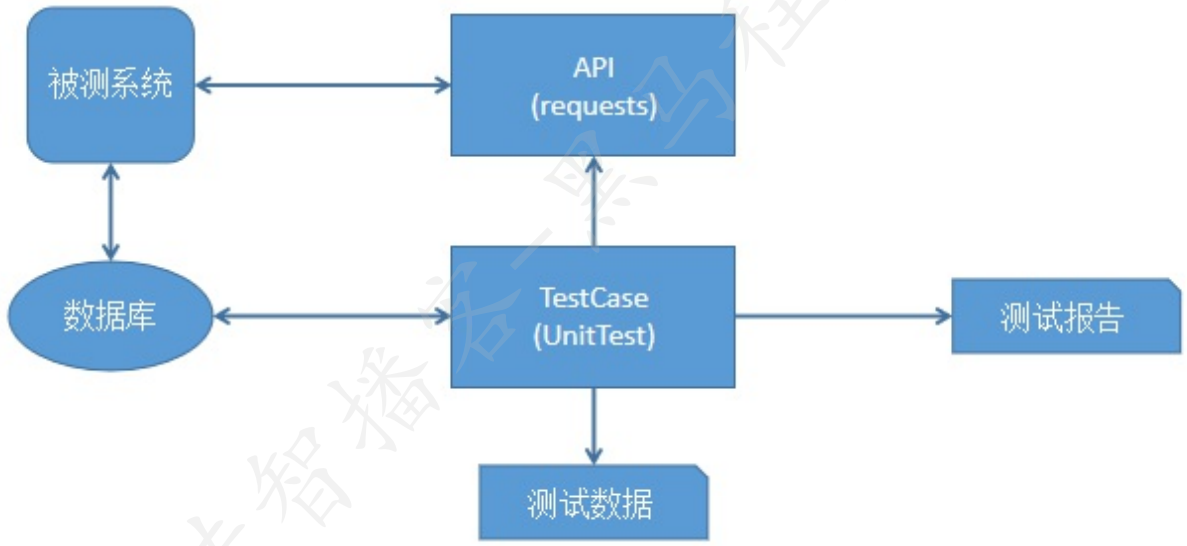
目标

1. 掌握接口测试框架的结构
2. 掌握如何封装被测系统的接口
3. 掌握如何定义接口测试用例
4. 掌握如何集成测试报告

---

1. 框架结构

接口测试框架的结构如图：



接口测试框架的结构说明：

- API 用于封装被测系统的接口
- TestCase 将一个或者多个接口封装成测试用例，并使用UnitTest管理测试用例
- TestCase 可以调用数据库进行数据的校验
- 为了方便维护测试数据，可以把测试脚本和测试数据分离开
- 通过UnitTest断言接口返回的数据，并生成测试报告

---

2. 框架目录结构

接口测试框架目录结构：

```
apiTestFramework    # 项目名称
├── api             # 定义封装被测系统的接口
├── script          # 定义测试用例脚本
├── data            # 存放测试数据
├── report          # 存放生成的测试报告
├── lib             # 存放第三方的文件
├── app.py          # 定义项目的配置信息
├── utils.py        # 定义工具类
└── run_suite.py    # 执行测试套件的入口
```

3. 封装被测系统的接口

按照功能模块定义封装被测系统的接口，方便测试脚本的调用，并且能够到达代码的复用。

对登录功能的相关接口进行封装，示例代码：

```python
# api/login.py
class LoginApi:

    def __init__(self):
        self.verify_code_url = "http://localhost/index.php?m=Home&c=User&a=verify"
        self.login_url = "http://localhost/index.php?m=Home&c=User&a=do_login"

    # 获取验证码
    def get_login_verify_code(self, session):
        return session.get(self.verify_code_url)

    # 登录
    def login(self, session, username, password, verify_code):
        # 发送请求
        data = {
            "username": username,
            "password": password,
            "verify_code": verify_code,
        }
        return session.post(self.login_url, data=data)
```

4. 定义接口测试用例

将api模块中的一个或多个接口封装成一个测试用例，并使用测试框架UnitTest管理测试用例。

定义登录功能的测试用例，示例代码：

```python
import unittest

import requests
from requests import Session

from api.login import LoginApi


class TestLogin(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.login_api = LoginApi()

    def setUp(self):
        self.session = Session()

    def tearDown(self):
        self.session.close()

    # 登录成功
    def test_login_success(self):
        # 获取验证码
        response = self.login_api.get_login_verify_code(self.session)
        # 判断是否为图片类型
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        response = self.login_api.login(self.session, "13012345678", "123456", "8888")
        result = response.json()
        print("login response data=", result)

        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(1, result.get("status"))
        self.assertEqual("登陆成功", result.get("msg"))

    # 账号不存在
    def test_login_username_not_exist(self):
        # 获取验证码
        response = self.login_api.get_login_verify_code(self.session)
        # 判断是否为图片类型
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        response = self.login_api.login(self.session, "13088888888", "123456", "8888")
        result = response.json()
        print("login response data=", result)
```

```
        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(-1, result.get("status"))
        self.assertIn("账号不存在", result.get("msg"))

    # 密码错误
    def test_login_password_is_error(self):
        # 获取验证码
        response = self.login_api.get_login_verify_code(self.session)
        # 判断是否为图片类型
        self.assertIn("image", response.headers.get("Content-Type"))

        # 登录
        response = self.login_api.login(self.session, "13012345678", "error", "8888")
        result = response.json()
        print("login response data=", result)

        # 断言
        self.assertEqual(200, response.status_code)
        self.assertEqual(-2, result.get("status"))
        self.assertIn("密码错误", result.get("msg"))
```

5. 集成测试报告

使用HTMLTestRunner生成HTML格式的测试报告

```
import time
import unittest

from script.test_login import TestLogin
from tools.HTMLTestRunner import HTMLTestRunner

suite = unittest.TestSuite()
suite.addTest(unittest.makeSuite(TestLogin))

# 测试报告文件路径
report_file = "./report/report{}.html".format(time.strftime("%Y%m%d-%H%M%S"))
with open(report_file, "wb") as f:
    # 创建HTMLTestRunner运行器
    runner = HTMLTestRunner(f, title="TPshop接口自动化测试报告", description="V1.0")

    # 运行测试套件
    runner.run(suite)
```

项目实战

目标

1. 掌握如何通过代码实现接口测试

---

1. 项目搭建

1.1 新建项目

项目名称：apiTestIHRM

1.2 创建目录结构

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

1.3 安装依赖包

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

---

2. 编写代码

2.1 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
员工模块：employee.py
```

2.2 编写测试脚本

1. 定义测试脚本文件

   ```
   登录模块：test_login.py
   员工模块：test_employee.py
   ```

2. 使用unittest管理测试脚本

2.3 执行测试脚本

1. 使用unittest执行测试脚本
2. 调试代码

---

3. 数据库数据校验

3.1 用例场景

调用修改员工信息的接口后，校验用户名是否更新到了数据库中。

3.2 表结构

```sql
CREATE TABLE `bs_user` (
  `id` varchar(40) NOT NULL COMMENT 'ID',
  `mobile` varchar(40) NOT NULL COMMENT '手机号码',
  `username` varchar(255) NOT NULL COMMENT '用户名称',
  `password` varchar(255) DEFAULT NULL COMMENT '密码',
  `enable_state` int(2) DEFAULT '1' COMMENT '启用状态 0是禁用，1是启用',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `department_id` varchar(40) DEFAULT NULL COMMENT '部门ID',
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_user_phone` (`mobile`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='员工表';
```

3.3 示例代码

```python
# 修改员工
def test03_update_emp(self):
```

```
        # 测试数据
        emp_id = TestEmployee.employee_id
        username = "tom-new"

        # 调用接口
        response = self.employee_api.update_emp(emp_id, username)
        result = response.json()
        logging.info("update emp data={}".format(result))

        # 断言
        utils.common_assert(self, response, 200, True, 10000, "操作成功")

        # 数据库数据校验
        conn = pymysql.connect("182.92.81.159", "readuser", "pwd123", "ihrm")
        cursor = conn.cursor()
        sql = "select username from bs_user where id=%s"
        cursor.execute(sql, emp_id)
        data = cursor.fetchone()
        db_username = data[0]
        cursor.close()
        conn.close()
        self.assertEqual(username, db_username)
```

4. 测试数据参数化

4.1 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

```
登录模块：login.json
员工模块：employee.json
```

3. 根据业务编写用例数据

4.2 引入参数化

修改测试脚本，使用 `parameterized` 实现参数化

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="IHRM接口自动化测试报告", description="V1.0")
    runner.run(suite)
```