Table of Contents

接口测试课程

接口自动化测试

目标

1. 理解接口自动化测试的概念
2. 掌握如何使用Requests库
3. 掌握如何使用UnitTest管理接口测试脚本
4. 掌握接口自动化测试框架的结构
5. 掌握如何搭建接口自动化测试框架
6. 了解Mock的概念和作用
7. 掌握如何使用Python中的Mock模块实现mock测试

接口自动化测试介绍

目标

1．理解接口自动化测试的概念
2．了解接口自动化测试的实现方式

---

1. 接口自动化测试

1.1 概念

> 接口测试：是对系统或组件之间的接口进行测试，主要是校验数据的交换、传递和控制管理过程，以及相互逻辑依赖关系。
>
> 自动化测试：是把以人为驱动的测试行为转化为机器执行的一种过程。

接口自动化测试：是让程序或工具代替人工自动的完成对接口进行测试的一种过程。

1.2 实现方式

- 使用接口测试工具来实现，比如：JMeter
- 通过编写代码来实现

1.3 接口测试工具的不足

1. 测试数据不好控制
2. 不方便测试加密接口
3. 扩展能力不足

Requests库

目标

```
1. 掌握如何使用Requests库
```

---

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、HEAD、OPTIONS

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

```
import requests

response = requests.get("http://www.baidu.com")
```

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。

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
    """
```

```
import requests

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

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

> https://www.baidu.com/s?wd=python&ie=utf-8&rqlang=cn&tn=baiduhome_pg

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。

```
import requests

response = requests.get("http://www.baidu.com", params="kw=python")
print(response.url)  # http://www.baidu.com/?kw=python

params = {"k1": "v1", "k2": ["v2", "v3"]}
response = requests.get("http://www.baidu.com", params=params)
print(response.url)  # http://www.baidu.com/?k1=v1&k2=v2&k2=v3
```

---

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
```

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。

```
response = requests.get("http://www.baidu.com")
json_data = response.json()
```

如果 JSON 解码失败，response.json() 就会抛出一个异常

---

5. 定制请求头

> 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("http://www.baidu.com", headers=headers)
```

---

6. Cookie

获取响应信息中的cookie数据：

```
response = requests.get("http://www.baidu.com")
print(response.cookies)
```

发送请求时添加cookie数据，可以使用 cookies 参数：

```
requests.get("http://www.baidu.com", cookies={"c1": "v1"})
```

6.1 案例

需求

```
使用requests库调用TPshop登录功能的相关接口，完成登录操作，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

在 requests 里，session对象是一个非常常用的对象，这个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。

7.1 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，登录成功后获取‘我的订单’页面的数据
2).使用Session对象来实现
```

示例代码

```python
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 需求

使用TPShop项目完成对登录功能的接口测试

1.2 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	备注
001	登录	登录成功	获取验证码	http://localhost/index.php?m=Hor	GET			获取到验证码图片		
			登录	http://localhost/index.php?m=Hor	POST	form	"username": "13012345678", "password": "123456", "verify_code": "8888"	登录成功		
002	登录	账号不存在	获取验证码	http://localhost/index.php?m=Hor	GET			获取到验证码图片		
			登录	http://localhost/index.php?m=Hor	POST	form	"username": "13088888888", "password": "123456", "verify_code": "8888"	账号不存在		
003	登录	密码错误	获取验证码	http://localhost/index.php?m=Hor	GET			获取到验证码图片		
			登录	http://localhost/index.php?m=Hor	POST	form	"username": "13012345678", "password": "error", "verify_code": "8888"	密码错误		

1.3 代码实现

```
import unittest

import requests
from requests import Session


class TestLogin(unittest.TestCase):

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

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

```
    def get_login_verify_code(self):
        url = "http://localhost/index.php?m=Home&c=User&a=verify"
        response = self.session.get(url)

        # 判断是否为图片类型
        content_type = response.headers.get("Content-Type")
        self.assertIn("image", content_type)

    def test_login_success(self):
        # 获取验证码
        self.get_login_verify_code()

        # 发送请求
        url = "http://localhost/index.php?m=Home&c=User&a=do_login"
        data = {
            "username": "13012345678",
            "password": "123456",
            "verify_code": "8888",
        }
        response = self.session.post(url, data=data)

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
        json_data = response.json()
        self.assertEqual(1, json_data.get("status"))
        self.assertIn("登陆成功", json_data.get("msg"))

    def test_login_username_not_exist(self):
        # 获取验证码
        self.get_login_verify_code()

        # 发送请求
        url = "http://localhost/index.php?m=Home&c=User&a=do_login"
        data = {
            "username": "13088888888",
            "password": "123456",
            "verify_code": "8888",
        }
        response = self.session.post(url, data=data)

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
        json_data = response.json()
```

```
        self.assertEqual(-1, json_data.get("status"))
        self.assertIn("账号不存在", json_data.get("msg"))

    def test_login_password_is_error(self):
        # 获取验证码
        self.get_login_verify_code()

        # 发送请求
        url = "http://localhost/index.php?m=Home&c=User&a=do_login"
        data = {
            "username": "13012345678",
            "password": "error",
            "verify_code": "8888",
        }
        response = self.session.post(url, data=data)

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
        json_data = response.json()
        self.assertEqual(-2, json_data.get("status"))
        self.assertIn("密码错误", json_data.get("msg"))
```

接口测试框架开发

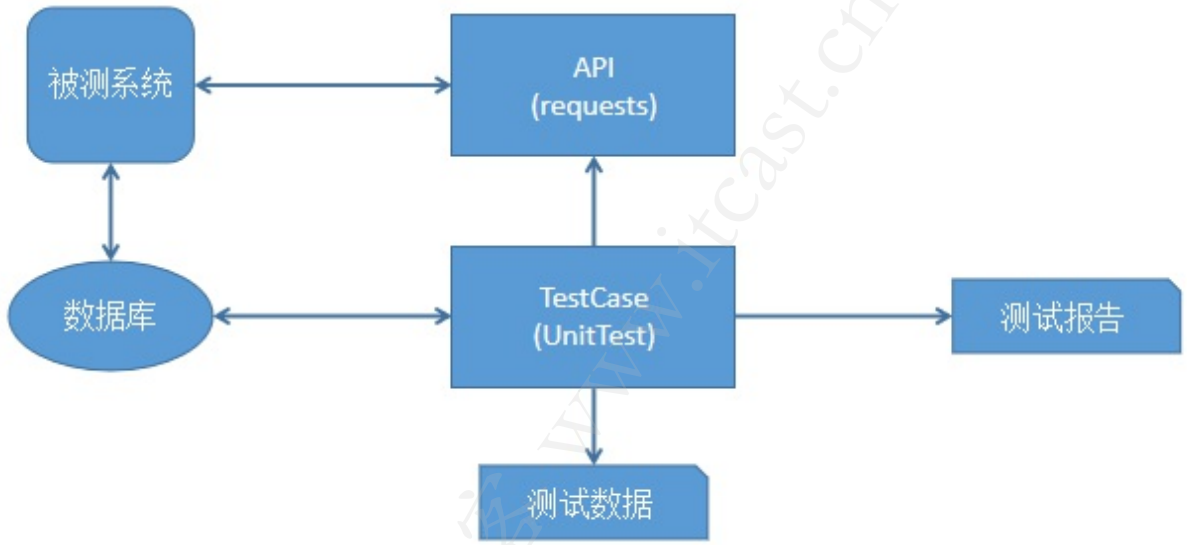
目标

1. 掌握接口自动化测试框架的结构
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

接口自动化测试框架的结构如图：



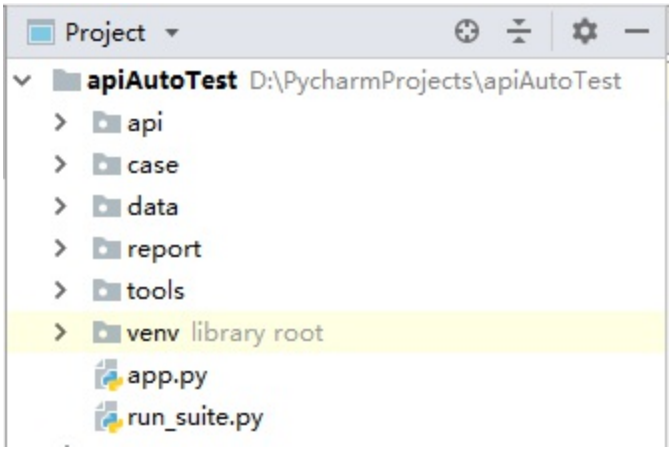
接口自动化测试框架的结构说明：

- 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

接口自动化测试框架目录结构如图：



这里将接口自动化测试框架命名为apiAutoTest，各个目录与文件的作用说明：

- api：定义封装被测系统的接口
- case：定义测试用例
- data：存放测试数据
- report：存放生成的测试报告
- tools：存放第三方的文件
- app.py：定义项目的配置信息
- 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，方便测试用例的调用，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：

```
# 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，并使用单元测试框架UnitTest管理测试用例。
定义登录功能的测试用例，示例代码：

```
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")

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
```

```
        json_data = response.json()
        self.assertEqual(1, json_data.get("status"))
        self.assertIn("登陆成功", json_data.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")

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
        json_data = response.json()
        self.assertEqual(-1, json_data.get("status"))
        self.assertIn("账号不存在", json_data.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")

        # 断言状态码
        self.assertEqual(requests.codes.ok, response.status_code)

        # 断言响应数据
        print(response.json())
        json_data = response.json()
        self.assertEqual(-2, json_data.get("status"))
        self.assertIn("密码错误", json_data.get("msg"))
```

5. 集成测试报告

使用HTMLTestRunner生成HTML格式的测试报告

```python
import time
import unittest

from case.test_login import TestLogin
from case.test_order import TestOrder
from case.test_register import TestRegister
from tools.HTMLTestRunner import HTMLTestRunner

suite = unittest.TestSuite()
suite.addTest(TestRegister("test_register"))
suite.addTest(unittest.makeSuite(TestLogin))
suite.addTest(TestOrder("test_order"))

# 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)
```

Mock测试

目标

1. 了解Mock的概念和作用
2. 了解Mock的实现方式
3. 掌握如何使用Python中的Mock模块实现mock测试

---

1. 什么是Mock？

Mock是模拟的意思，在软件测试领域，对于某些不容易构造或者不容易获取的对象，可以通过某种技术手段虚拟出一个测试对象， 返回预先设计的结果。也就是说对于任意被测试的对象，可以根据具体测试场景的需要，返回特定的结果。

Mock测试：在测试过程中，对于某些不容易构造或者不容易获取的对象，可以用一个虚拟的对象来代替的测试方法。

2. Mock的作用

- 可以用来解除测试对象对外部服务的依赖（比如数据库，第三方接口等），使得测试用例可以独立运行。
- 替换外部服务调用或一些速度较慢的操作，提升测试用例的运行速度。
- 模拟异常逻辑：一些异常的逻辑往往在正常测试中是很难触发的，通过Mock可以人为的控制触发异常逻辑。

3. 接口Mock的实现方式

关于接口Mock的实现方式，可以分为白盒和黑盒两种：

- 白盒：手动构造mock对象，比如：可以自己写某个接口方法的实现，根据需要编写返回值
- 黑盒：Mock方案和程序使用的语言无关，比如：搭建一个Mock服务器

4. Python Mock

Mock是Python中一个用于支持的测试的库，它的主要功能是使用mock对象替代掉指定的Python对象，以达到模拟对象的行为。

4.1 Mock的安装和导入

在Python 3.3以前的版本中，需要另外安装mock模块，可以使用pip命令来安装：

```
pip install mock
```

然后在代码中就可以直接import进来：

```
import mock
```

从Python 3.3开始，mock模块已经被合并到标准库中，被命名为unittest.mock，可以直接import进来使用：

```
from unittest import mock
```

4.2 基本用法

Mock对象是mock模块中最重要的概念，就是通过unittest.mock.Mock类创建的实例，这个类的实例可以用来替换其他的Python对象，来达到模拟的效果。 Mock类的定义如下：

```
class Mock(name=None, return_value=DEFAULT, side_effect=None, wraps=None,
        spec=None, spec_set=None, unsafe=False)
```

Mock对象的基本步骤：

1. 找到要替换的对象（可以是一个类、类的实例、函数）
2. 实例化Mock类得到一个mock对象，并且设置mock对象的行为（比如被调用的时候返回什么值，被访问成员的时候返回什么值）
3. 使用这个mock对象替换掉要替换的对象
4. 编写测试代码

```
import unittest
from unittest import mock

def add(x, y):
    print("该方法还未写完...")
    return 0

class TestAdd(unittest.TestCase):

    def test_add01(self):
        # mock
        add = mock.Mock(return_value=3)

        result = add(1, 2)
        print("result=", result)
```

```
        self.assertEqual(3, result)
```