如何简单实现接口自动化测试（基于python）

一、简介

本文从一个简单的登录接口测试入手，一步步调整优化接口调用姿势，然后简单讨论了一下接口测试框架的要点，最后介绍了一下我们目前正在使用的接口测试框架pithy。期望读者可以通过本文对接口自动化测试有一个大致的了解。

二、引言

为什么要做接口自动化测试？

> 在当前互联网产品迭代频繁的背景下，回归测试的时间越来越少，很难在每个迭代都对所有功能做完整回归。但接口自动化测试因其实现简单、维护成本低，容易提高覆盖率等特点，越来越受重视。

为什么要自己写框架呢？

> 使用requets + unittest很容易实现接口自动化测试，而且requests的api已经非常人性化，非常简单，但通过封装以后（特别是针对公司内特定接口），再加上对一些常用工具的封装，可以进一步提高业务脚本编写效率。

三、环境准备

确保本机已安装python2.7以上版本，然后安装如下库：

```
pip install flask
pip install requests
```

后面我们会使用flask写一个用来测试的接口，使用requests去测试。

四、测试接口准备

下面使用flask实现两个http接口，一个登录，另外一个查询详情，但需要登录后才可以，新建一个demo.py文件（注意，不要使用windows记事本），把下面代码copy进去，然后保存、关闭。

接口代码

```
#!/usr/bin/python
# coding=utf-8
from flask import Flask, request, session, jsonify

USERNAME = 'admin'
PASSWORD = '123456'

app = Flask(__name__)
app.secret_key = 'pithy'


@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != USERNAME:
            error = 'Invalid username'
        elif request.form['password'] != PASSWORD:
            error = 'Invalid password'
        else:
            session['logged_in'] = True
            return jsonify({'code': 200, 'msg': 'success'})
    return jsonify({'code': 401, 'msg': error}), 401


@app.route('/info', methods=['get'])
def info():
    if not session.get('logged_in'):
        return jsonify({'code': 401, 'msg': 'please login !!'})
    return jsonify({'code': 200, 'msg': 'success', 'data':
'info'})

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


最后执行如下命令：

```
python demo.py
```

响应如下：

```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
```

大家可以看到服务已经起来了。

接口信息

登录接口

- 请求url

> /login

- 请求方法

> post

- 请求参数

参数名称	参数类型	参数说明
username	String	登录名称
password	String	登录密码

- 响应信息

参数名称	参数类型	参数说明
code	Integer	结果code
msg	String	结果信息

详情接口

- 请求url

> /info

- 请求方法

> get

- 请求cookies

参数名称	参数类型	参数说明
session	String	session

- 响应信息

参数名称	参数类型	参数说明
code	Integer	结果code
msg	String	结果信息
data	String	数据信息

五、编写接口测试

测试思路

- 使用requests [使用链接] 库模拟发送HTTP请求。
- 使用python标准库里unittest写测试case。

脚本实现

```
#!/usr/bin/python
# coding=utf-8
import requests
import unittest
```

```
class TestLogin(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.login_url = 'http://127.0.0.1:5000/login'
        cls.info_url = 'http://127.0.0.1:5000/info'
        cls.username = 'admin'
        cls.password = '123456'

    def test_login(self):
        """
        测试登录
        """
        data = {
            'username': self.username,
            'password': self.password
        }

        response = requests.post(self.login_url,
data=data).json()

        assert response['code'] == 200
        assert response['msg'] == 'success'

    def test_info(self):
        """
        测试info接口
        """

        data = {
            'username': self.username,
            'password': self.password
        }

        response_cookies = requests.post(self.login_url,
data=data).cookies
        session = response_cookies.get('session')
        assert session

        info_cookies = {
            'session': session
        }

        response = requests.get(self.info_url,
cookies=info_cookies).json()
        assert response['code'] == 200
        assert response['msg'] == 'success'
        assert response['data'] == 'info'
```

六、优化

封装接口调用

写完这个测试登录脚本，你或许会发现，在整个项目的测试过程，登录可能不止用到一次，如果每次都这么写，会不会太**冗余**了？ 对，确实太冗余了，下面做一下简单的封装，把登录接口的调用封装到一个方法里，把调用参数暴漏出来，示例脚本如下：

```
#!/usr/bin/python
# coding=utf-8
import requests
import unittest
try:
    from urlparse import urljoin
except ImportError:
    from urllib.parse import urljoin


class DemoApi(object):

    def __init__(self, base_url):
        self.base_url = base_url

    def login(self, username, password):
        """
        登录接口
        :param username: 用户名
        :param password: 密码
        """
        url = urljoin(self.base_url, 'login')
        data = {
            'username': username,
            'password': password
        }

        return requests.post(url, data=data).json()

    def get_cookies(self, username, password):
        """
        获取登录cookies
        """
        url = urljoin(self.base_url, 'login')
        data = {
            'username': username,
            'password': password
        }

        return requests.post(url, data=data).cookies
```

```
    def info(self, cookies):
        """
        详情接口
        """
        url = urljoin(self.base_url, 'info')
        return requests.get(url, cookies=cookies).json()


class TestLogin(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        cls.base_url = 'http://127.0.0.1:5000'
        cls.username = 'admin'
        cls.password = '123456'
        cls.app = DemoApi(cls.base_url)

    def test_login(self):
        """
        测试登录
        """
        response = self.app.login(self.username, self.password)
        assert response['code'] == 200
        assert response['msg'] == 'success'

    def test_info(self):
        """
        测试获取详情信息
        """
        cookies = self.app.get_cookies(self.username,
self.password)
        response = self.app.info(cookies)
        assert response['code'] == 200
        assert response['msg'] == 'success'
        assert response['data'] == 'info'
```


OK，在这一个版本中，我们不但在把登录接口的调用封装成了一个实例方法，实现了复用，而且还把host（self.base_url）提取了出来，但问题又来了，登录之后，登录接口的http响应会把**session**以 cookie的形式set到客户端，之后的接口都会使用此session去请求，还有，就是在接口调用过程中，希望可以把**日志**打印出来，以便调试或者出错时查看。

好吧，我们再来改一版。

保持cookies&增加log信息

使用requests库里的同一个Session对象(它也会在同一个Session 实例发出的所有请求之间保持 cookie)，即可解决上面的问题，示例代码如下：

```python
#!/usr/bin/python
# coding=utf-8
import unittest
from pprint import pprint
from requests.sessions import Session
try:
    from urlparse import urljoin
except ImportError:
    from urllib.parse import urljoin


class DemoApi(object):

    def __init__(self, base_url):
        self.base_url = base_url
        # 创建session实例
        self.session = Session()

    def login(self, username, password):
        """
        登录接口
        :param username: 用户名
        :param password: 密码
        """
        url = urljoin(self.base_url, 'login')
        data = {
            'username': username,
            'password': password
        }

        response = self.session.post(url, data=data).json()
        print('\n****************************************')
        print(u'\n1、请求url: \n%s' % url)
        print(u'\n2、请求头信息:')
        pprint(self.session.headers)
        print(u'\n3、请求参数:')
        pprint(data)
        print(u'\n4、响应:')
        pprint(response)
        return response

    def info(self):
        """
        详情接口
        """
        url = urljoin(self.base_url, 'info')
        response = self.session.get(url).json()

        print('\n****************************************')
        print(u'\n1、请求url: \n%s' % url)
        print(u'\n2、请求头信息:')
```

```
            pprint(self.session.headers)
            print(u'\n3、请求cookies:')
            pprint(dict(self.session.cookies))
            print(u'\n4、响应:')
            pprint(response)
            return response


    class TestLogin(unittest.TestCase):

        @classmethod
        def setUpClass(cls):
            cls.base_url = 'http://127.0.0.1:5000'
            cls.username = 'admin'
            cls.password = '123456'
            cls.app = DemoApi(cls.base_url)

        def test_login(self):
            """
            测试登录
            """
            response = self.app.login(self.username, self.password)
            assert response['code'] == 200
            assert response['msg'] == 'success'

        def test_info(self):
            """
            测试获取详情信息
            """
            self.app.login(self.username, self.password)
            response = self.app.info()
            assert response['code'] == 200
            assert response['msg'] == 'success'
            assert response['data'] == 'info'
```


大功告成，我们把多个相关接口调用封装到一个类中，使用同一个requests Session实例来保持cookies，并且在调用过程中打印出了日志，我们所有目标都实现了，但再看下脚本，又会感觉不太舒服，在每个方法里，都要写一遍print 1、2、3... 要拼url、还要很多细节等等，但其实我们**真正需要做的只是拼出关键的参数**（url参数、body参数或者传入headers信息），可不可以只需定义必须的信息，然后把其它共性的东西都封装起来呢，统一放到一个地方去管理？

封装重复操作

来，我们再整理一下我们的需求:

- 首先，不想去重复做拼接url的操作。

- 然后，不想每次都去手工打印日志。
- 不想和requests session打交道。
- 只想定义好参数就直接调用。

我们先看一下实现后，脚本可能是什么样:

```
class DemoApi(object):

    def __init__(self, base_url):
        self.base_url = base_url

    @request(url='login', method='post')
    def login(self, username, password):
        """
        登录接口
        """
        data = {
            'username': username,
            'password': password
        }

        return {'data': data}

    @request(url='info', method='get')
    def info(self):
        """
        详情接口
        """
        pass
```


调用登录接口的日志：

```
************************************************************
1、接口描述
登录接口

2、请求url
http://127.0.0.1:5000/login

3、请求方法
post

4、请求headers
{
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "keep-alive",
```

```
    "User-Agent": "python-requests/2.7.0 CPython/2.7.10
Darwin/16.4.0"
}

5、body参数
{
    "password": "123456",
    "username": "admin"
}

6、响应结果
{
    "code": 200,
    "msg": "success"
}
```

在这里，我们使用python的装饰器功能，把公共特性封装到装饰器中去实现。现在感觉好多了，没什么多余的东西了，我们可以专注于关键参数的构造，剩下的就是如何去实现这个装饰器了，我们先理一下思路：

> - 获取装饰器参数
> - 获取函数/方法参数
> - 把装饰器和函数定义的参数合并
> - 拼接url
> - 处理requests session，有则使用，无则新生成一个
> - 组装所有参数，发送http请求并打印日志

因篇幅限制，源码不再列出，有兴趣的同学可以查看已经实现的源代码。

源代码查看地址：https://github.com/yuyu1987/pithy-test/blob/master/pithy/api.py

七、扩展

http接口请求的姿势我们定义好了，我们还可以做些什么呢？

> - [x] 非HTTP协议接口
> - [x] 测试用例编写
> - [x] 配置文件管理
> - [x] 测试数据管理
> - [x] 工具类编写
> - [x] 测试报告生成
> - [x] 持续集成
> - [x] 等等等等

需要做的还是挺多的，要做什么不要做什么，或者先做哪个，我觉得可以根据以下几点去判断：

- 是否有利于提高团队生产效率？
- 是否有利于提高测试质量？
- 有没有现成的轮子可以用？

下面就几项主要的点进行一下说明，限于篇幅，不再展开了。

测试报告

这个应该是大家最关心的了，毕竟这是测试工作的产出；

目前python的主流单元测试框均有report插件，因此不建议自己再编写，除非有特殊需求的。

- pytest：推荐使用pytest-html和allure pytest。
- unittest：推荐使用HTMLTestRunner。

持续集成

持续集成推荐使用Jenkins，运行环境、定时任务、触发运行、邮件发送等一系列功能均可以在Jenkins上实现。

测试用例编写

推荐遵守如下规则：

- **原子性**：每个用例保持独立，彼此不耦合，以降低干扰。
- **专一性**：一个用例应该专注于验证一件事情，而不是做很多事情，一个测试点不要重复验证。
- **稳定性**：绝大多数用例应该是非常稳定的，也就是说不会经常因为除环境以外的因素挂掉，因为如果在一个测试项目中有很多不稳定的用例的话，测试结果就不能很好的反应项目质量。
- **分类清晰**：有相关性的用例应写到一个模块或一个测试类里，这样做即方便维护，又提高了报告的可读性。

测试工具类

这个可以根据项目情况去做，力求简化一些类库的使用，数据库访问、日期时间、序列化与反序列化等数据处理，或者封装一些常用操作，如随机生成订单号等等，以提高脚本编写效率。

测试数据管理

常见的方式有写在代码里、写在配置文件里(xml、yaml、json、.py、excel等)、写在数据库里等，该处没有什么好推荐的，建议根据个人喜好，怎么方便怎么来就可以。

八、pithy测试框架介绍

pithy意为简洁有力的，意在简化自动化接口测试，提高测试效率。

- **项目地址：点击查看**
- **帮助文档：点击查看**

目前实现的功能如下：

> - 一键生成测试项目
> - http client封装
> - thrift接口封装
> - 简化配置文件使用
> - 优化JSON、日期等工具使用

编写测试用例推荐使用pytest，pytest提供了很多测试工具以及插件，可以满足大部分测试需求。

安装

```
pip install pithy-test
pip install pytest
```

使用

一键生成测试项目

```
>>>  pithy-cli init
请选择项目类型,输入api或者app: api
请输入项目名称,如pithy-api-test: pithy-api-test
```

```
开始创建pithy-api-test项目
开始渲染...
生成 api/.gitignore                    [√]
生成 api/apis/__init__.py              [√]
生成 api/apis/pithy_api.py             [√]
生成 api/cfg.yaml                      [√]
生成 api/db/__init__.py                [√]
生成 api/db/pithy_db.py                [√]
生成 api/README.MD                     [√]
生成 api/requirements.txt              [√]
生成 api/test_suites/__init__.py       [√]
生成 api/test_suites/test_login.py     [√]
生成 api/utils/__init__.py             [√]
生成成功,请使用编辑器打开该项目
```

生成项目树：

```
>>> tree pithy-api-test
pithy-api-test
├── README.MD
├── apis
│   ├── __init__.py
│   └── pithy_api.py
├── cfg.yaml
├── db
│   ├── __init__.py
│   └── pithy_db.py
├── requirements.txt
├── test_suites
│   ├── __init__.py
│   └── test_login.py
└── utils
    └── __init__.py

4 directories, 10 files
```


调用HTTP登录接口示例

```
from pithy import request

@request(url='http://httpbin.org/post', method='post')
def post(self, key1='value1'):
    """
    post method
    """
    data = {
        'key1': key1
    }
    return dict(data=data)
```

```
# 使用
response = post('test').to_json()      # 解析json字符,输出为字典
response = post('test').json           # 解析json字符,输出为字典
response = post('test').to_content()   # 输出为字符串
response = post('test').content        # 输出为字符串
response = post('test').get_cookie()   # 输出cookie对象
response = post('test').cookie         # 输出cookie对象

# 结果取值, 假设此处response = {'a': 1, 'b': { 'c': [1, 2, 3, 4]}}
response = post('13111111111', '123abc').json

print response.b.c    # 通过点号取值,结果为[1, 2, 3, 4]

print response('$.a') # 通过object path取值,结果为1

for i in response('$..c[@>3]'): # 通过object path取值,结果为选中c字
典里大于3的元素
    print i
```

优化JSON、字典使用


```
# 1、操作JSON的KEY
from pithy import JSONProcessor
dict_data = {'a': 1, 'b': {'a': [1, 2, 3, 4]}}
json_data = json.dumps(dict_data)
result = JSONProcessor(json_data)
print result.a      # 结果：1
print result.b.a    # 结果：[1, 2, 3, 4]

# 2、操作字典的KEY
dict_data = {'a': 1, 'b': {'a': [1, 2, 3, 4]}}
result = JSONProcessor(dict_data)
print result.a      # 1
print result.b.a    # [1, 2, 3, 4]

# 3、object path取值
raw_dict = {
    'key1':{
        'key2':{
            'key3': [1, 2, 3, 4, 5, 6, 7, 8]
        }
    }
}

jp = JSONProcessor(raw_dict)
for i in jp('$..key3[@>3]'):
    print i

# 4、其它用法
```

```
dict_1 = {'a': 'a'}
json_1 = '{"b": "b"}'
jp = JSONProcessor(dict_1, json_1, c='c')
print(jp)
```

更多使用方法

点击查看

九、总结

在本文中，我们以提高脚本开发效率为前提，一步一步打造了一个简易的测试框架，但因水平所限，并未涉及测试数据初始化清理、测试中如何MOCK等话题，前路依然任重而道远，希望给大家一个启发，不足之处还望多多指点，非常感谢。

作者简介

孙彦辉，饿了么软件测试工程师，主要负责大物流蜂鸟商家版的测试工作。

参考：

- requests：http://www.python-requests.org/en/master/
- thriftpy：http://thriftpy.readthedocs.io/en/latest/
- objectpath：http://objectpath.org/
- pytest：https://docs.pytest.org