前言

经过这几天对KidsGuardPro接口的调试，写了一套自动化脚本，目前可以正常进行注册、登录和获取订单信息接口的测试，现在把这几天调接口的工 作和遇到的困难记录一下。

postman传参拼接加密

在着手写脚本之前，还是先用postman来调通接口，看一下传参以及接口加密规则。在这要感谢一下秀芳提供的postman脚本，这省去了我自己去抓包看每个接口参数的时间。首先遇到的问题就是传参拼接加密的问题。

环境变量

KidsGuard Pro

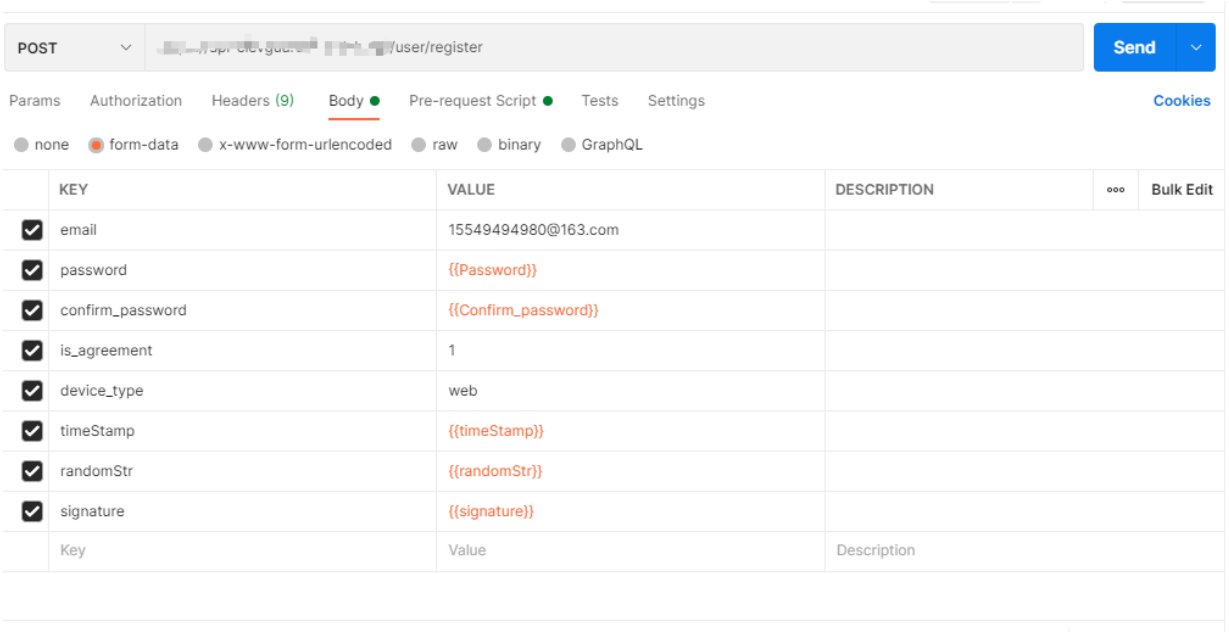
Save Share

	VARIABLE	INITIAL VALUE	CURRENT VALUE	Persist All	Reset All
☑	baseUrl	net	b.net		
☑	randomStr	65706237	14509111		
☑	signature	543859434665EBBC778E6898C1...	83154FE21C19E71B156797647FD1B941		
☑	timeStamp	1612184085	1616752904		
☑	token	4a55aec391a935b15c875e8dbf3...	4a55aec391a935b15c875e8dbf3b9840a4af3c8f81f0c9d5d1569e531f16466d		
☑	auth_token	s8mLTZGhIPF7P9Jwpy	s8mLTZGhIPF7P9Jwpy		
☑	Confirm_password	994dab4aa265429a66b591eda5e...	a3b4d0bf83802bc1096962016709a456		
☑	Password	994dab4aa265429a66b591eda5e...	a3b4d0bf83802bc1096962016709a456		
☑	confirm_password	59e30a1b7983de1003298a5a763...	59e30a1b7983de1003298a5a7630b367		

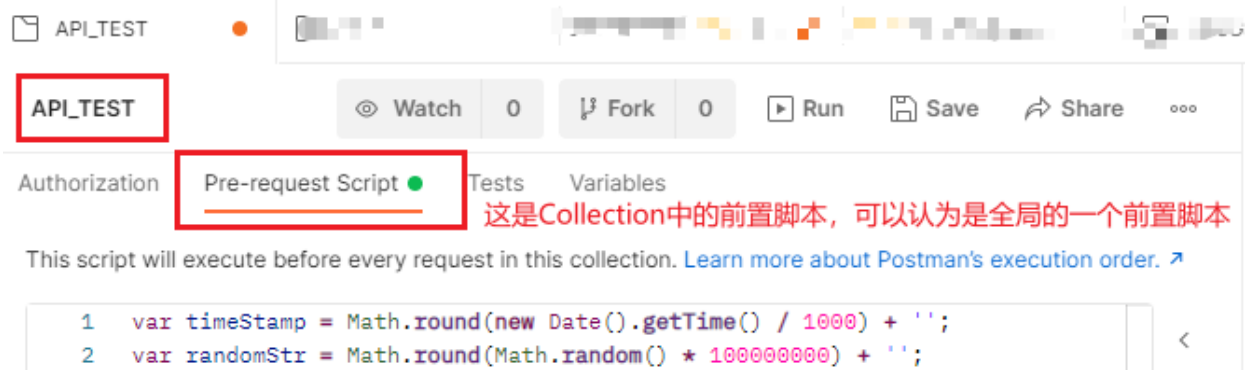
全局变量

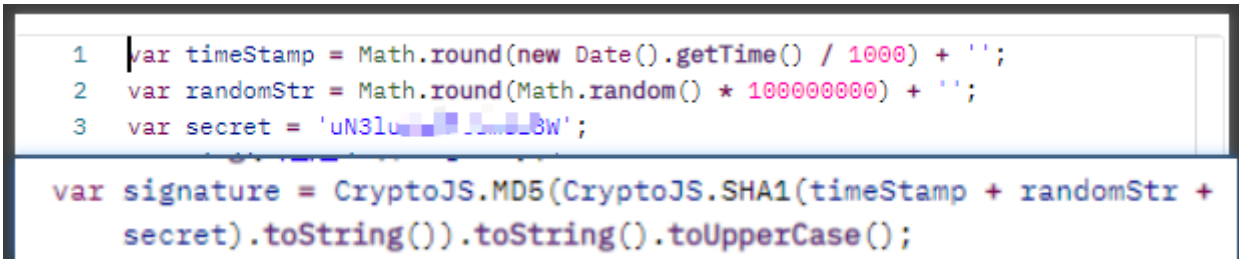
Globals Edit

VARIABLE	INITIAL VALUE	CURRENT VALUE
password	a3b4d0bf83802bc1096962016709a456	a3b4d0bf83802bc1096962016709a456
confirm_password	a3b4d0bf83802bc1096962016709a456	a3b4d0bf83802bc1096962016709a456
email		70lxf@Test.cn



如上图，导入的脚本是添加了环境变量和全局变量的，像password、confirm_password、timeStamp、randomStr、signature这些字段取得都是环境变量中的值,email字段取全局变量中的值。按照自己之前调接口的习惯，password、confirm_password、timeStamp、randomStr、signature取生产环境注册之后生成的字符串(抓包获得)，email随机写一个邮箱号。发送请求之后，第一次是成功的，看到接口有返回数据。再次点击发送，预期结果应该是报“邮箱已存在”之类的信息，实际结果报了鉴权错误。第一反应应该是传参有问题了，首先觉得是传的时间不对，就换了一个获取实时时间的方法，结果还是不行，看控制台台打印日志也没看出来，觉得应该有加密规则。问了一下秀芳，她和我说这些字段都要进行拼接再传。





拼接规则如上图：

timeStamp：获取当前最新时间

randomStr：随机获取0-100000000范围内的数值

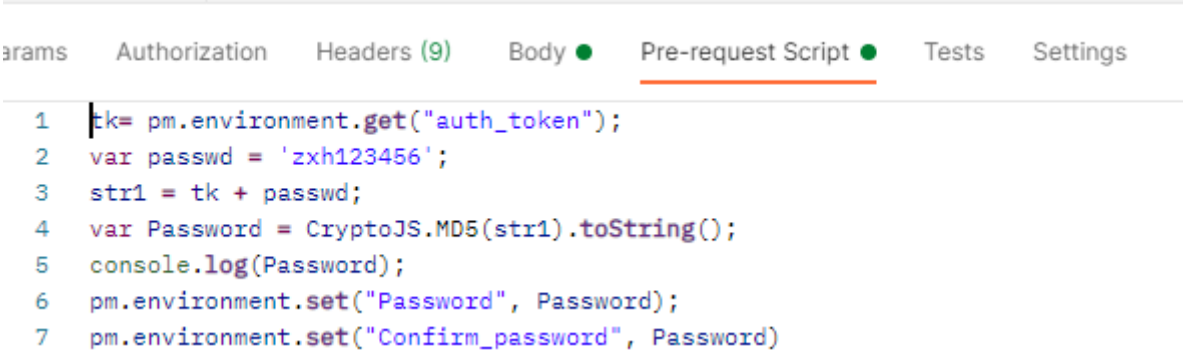
secret：内部自定义的key

signature：这个拼接规则比较复杂。

- 先拼接上面三个参数
- 再进行SHA1加密，加密之后转换成string类型
- 再进行MD5加密
- 转换成string类型
- 结果全部转换为大写

经过以上加密规则操作之后，把获得的数值都存进环境变量中，这样发送请求时传参就可以实时取环境变量中的值。(什么是环境变量以及环境变量如何设置 https://blog.csdn.net/weixin_29092787/article/details/112586897)

除了以上三个字段有经过加密算法处理，我们看到password、confirm_password也取得是环境变量中的值，这里我们在单个接口的Pre-request Script中来对密码进行加密（为什么不像上面三个字段那样放在全局的前置脚本中操作呢？因为上面三个字段是每个接口都会传的，而密码字段不是所有接口都要传的，放在单个接口的前置脚本中加密更加合理）。



可以看到密码的加密规则比较简单

- 先拼接auth_token+自己定义的密码（其中auth_token的值为内部自定义的固定值，我们放在环境变量中，也可以写死）
- 进行MD5加密
- 转换成string类型

经过以上加密规则操作之后，把获得的数值都存进环境变量中，这样发送请求时传参就可以实时取环境变量中的值。

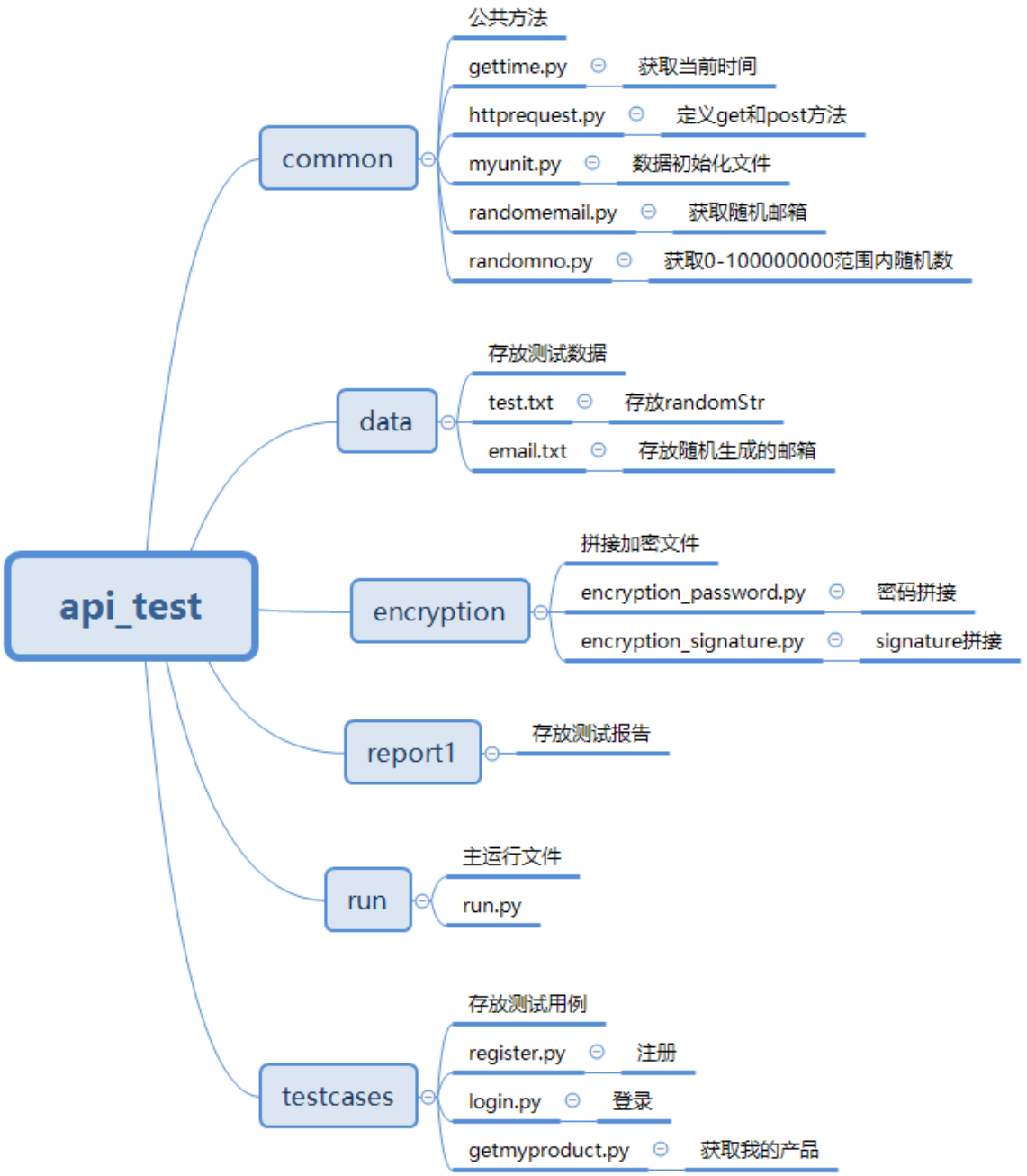
对以上参数进行处理之后，再次点击发送请求，就可以返回成功了。

以上只着重写了传参的拼接加密规则，postman的具体用法不再赘述。

python3+requests

现在写的这套脚本层级比较简单，只进行了部分公共方法的封装，也没有把持续集成应用上去，后续有时间会持续优化。

- **层级目录**



具体思路：

拿注册这个接口来说

```
'''
注册接口
'''
import unittest
from common.httprequest import http_request
from encryption.encryption_signature import get_str_sha1_md5_str
from encryption.encryption_password import get_str_password_md5
from common.gettime import timeStamp
from common.myunit import StartEnd
from common.randomemail import eamil


class Testregister(StartEnd):
    register = 'https://[redacted]/user/register'
    # 随机邮箱注册
    def test_001_randomEmail(self):
        register_url = self.register
        save = open('C:/Users/2021/PycharmProjects/api_test/data/test.txt', 'r')
        random1 = save.read()
        save.close()
        register_data = {'email': eamil(),
                         'password': get_str_password_md5(),
                         'confirm_password': get_str_password_md5(),
                         'is_agreement': '1',
                         'device_type': 'web',
                         'timeStamp': timeStamp(),
                         'randomStr': str(random1),
                         'signature': get_str_sha1_md5_str()}
        login_msg = http_request.http_post(register_url, register_data)
```

外部依赖包导入

请求URL

取存在本地的randomStr

随机邮箱

拼接的密码

最新时间

拼接的signature

post方法请求

```
try:
    self.assertEqual('注册并激活成功!', login_msg.json()['msg'])
    print('test_001_randomEmail 测试通过')
except Exception as e:
    print('test_001_randomEmail 测试不通过')
    raise e
```

断言

1. 用拼接字符串的方法以及生成随机数的方法，生成一个随机邮箱

```
'''
生成随机邮箱
'''

import random
import string

def eamil():

    a = random.randint(0, 999)
    b = random.choice(string.ascii_letters)
    randomemail = str(a)+str(b)+"zxh@Test.cn"
    print("随机生成的邮箱号为:",randomemail)
    with open("C:/Users/2021/PycharmProjects/api_test/data/email.txt", "wt") as out_file:
        out_file.write(randomemail)
    ...
    return randomemail
```

随机取0-999内的数值

随机取一个字母

进行拼接

把生成的邮箱存在本地

2.密码的拼接

```
'''
密码加密
'''

import hashlib

def get_str_password_md5():
    auth_token ="[redacted]py"
    passwd = "zxh123456"
    str1 = auth_token + passwd  拼接成字符串
    hash_md5 = hashlib.md5(str1.encode('utf-8'))  进行MD5加密
    encrypts2 = str(hash_md5.hexdigest())  转换成string类型
    password = encrypts2
    return password
```

3.获取最新时间：通用方法，可以查看具体脚本

4.获取随机字符串（randomStr）

这个地方要说一下，踩了很长时间的坑。一开始的randomStr我写了一个随机获取0-10000000范围内数值的方法，传参的时候调用这个方法生成的随机数，当我把所有参数都配置好，运行文件之后，报错鉴权有误。第一反应是timeStamp和拼接signature是的timeStamp不一致导致的错误，随后把对应结果打印出来，发现是randomStr和拼接signature是的randomStr不一致。想了一下，在我调用注册接口时，会分别调两次生成随机数的方法，一次是直接传randomStr时调用，另外一次是传拼接signature时调用，会生成两个不同的数值，要是能保持一致，无异于买彩票的几率。

解决思路，先生成随机数直接存到本地，再要取随机数的地方直接在本地文件中拿，这样就能保证拿到的数值一致。

```
'''
生成随机数存到本地
'''
import random


def random1():
    randomStr = random.randint(0, 100000000) 生成随机数
    with open("C:/Users/2021/PycharmProjects/api_test/data/test.txt", "wt") as out_file:
        out_file.write(str(randomStr))    存到本地
    # open('../data/test.txt', 'w').write(str(randomStr))
    # open('../data/test.txt', 'w').close()
    return randomStr
```

```
class Testregister(StartEnd):
    register = 'https://[redacted]net/user/register'
    random1()    调用方法生成随机数
    # 随机邮箱注册
```

5.signature拼接，按照之前的拼接规则来进行，可查看具体脚本。

熟悉了注册接口的传参形式，登录接口就好写了。这里我把注册接口生成的邮箱也存到了本地，形成这样一个操作步骤，注册成功之后直接用注册的邮箱去登录，再进行断言。

获取我的产品这个接口，有一个我们常说的接口参数关联操作。因为这个接口传参的时候，头部要携带登录之后的生成的token，所以我先写了一个获取token的方法。

```
'''
登录生成token
'''
from common.httprequest import http_request
from encryption.encryption_signature import get_str_sha1_md5_str
from common.gettime import timeStamp



def getToken():  # 获取token函数
    url = "https://[redacted]/user/signin"
    save = open('C:/Users/2021/PycharmProjects/api_test/data/test.txt', 'r')
    random1 = save.read()
    save.close()
    login_data = {'email': '[redacted]yfone.cn',#该账号存在订单，方便后续查询
                      'password': '123456a',
                      'device_type': 'web',
                      'randomStr': str(random1),
                      'signature': get_str_sha1_md5_str(),
                      'timeStamp': timeStamp()}
    r = http_request.http_post(url, login_data)  # 发送post请求
    token = r.json()["data"]["token"]
    with open("C:/Users/2021/PycharmProjects/api_test/data/token.txt", "wt") as out_file:
        out_file.write(str(token))
    return (r.json()["data"]["token"])  # 将获取的token返回
```

获取我的产品这个接口传参时，headers中直接调用这个方法。

```
'''
获取我购买的产品
'''

import unittest
from common.httprequest import http_request
from encryption.encryption_signature import get_str_sha1_md5_str
from common.gettime import timeStamp
from config.gettoken import getToken

class GetMyproduct(unittest.TestCase):
    myproduct = "https://[redacted]/user/user/getMyProduct"
    headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Monimaster-Token': getToken(),
        'Monimaster-Device-Type': 'web'
    }
```

以上都是对单个接口的介绍，如何实现同时调用这些接口，这里我写的run.py文件就是来运行所有测试用例的。

```python
#生成测试报告
import time
import HTMLTestRunner
import unittest
from testcases.register import Testregister
from testcases.login import TestLogin
from testcases.getmyproduct import GetMyproduct

def test_report():
    suite= unittest.TestSuite()
    test1=unittest.TestLoader().loadTestsFromTestCase(Testregister)
    suite.addTests(test1)
    test2 = unittest.TestLoader().loadTestsFromTestCase(TestLogin)
    suite.addTests(test2)
    test3 = unittest.TestLoader().loadTestsFromTestCase(GetMyproduct)
    suite.addTests(test3)

    now=time.strftime('%Y-%m-%d %H %M %S') #获取当前时间
    filename='..//report1/'+now+'._report.html'
    fp=open(filename,'wb')
    runner = HTMLTestRunner.HTMLTestRunner\
        (stream=fp, title='接口测试报告', description='测试用例执行情况')
    runner.run(suite)

if __name__ == '__main__':
    test_report()
```

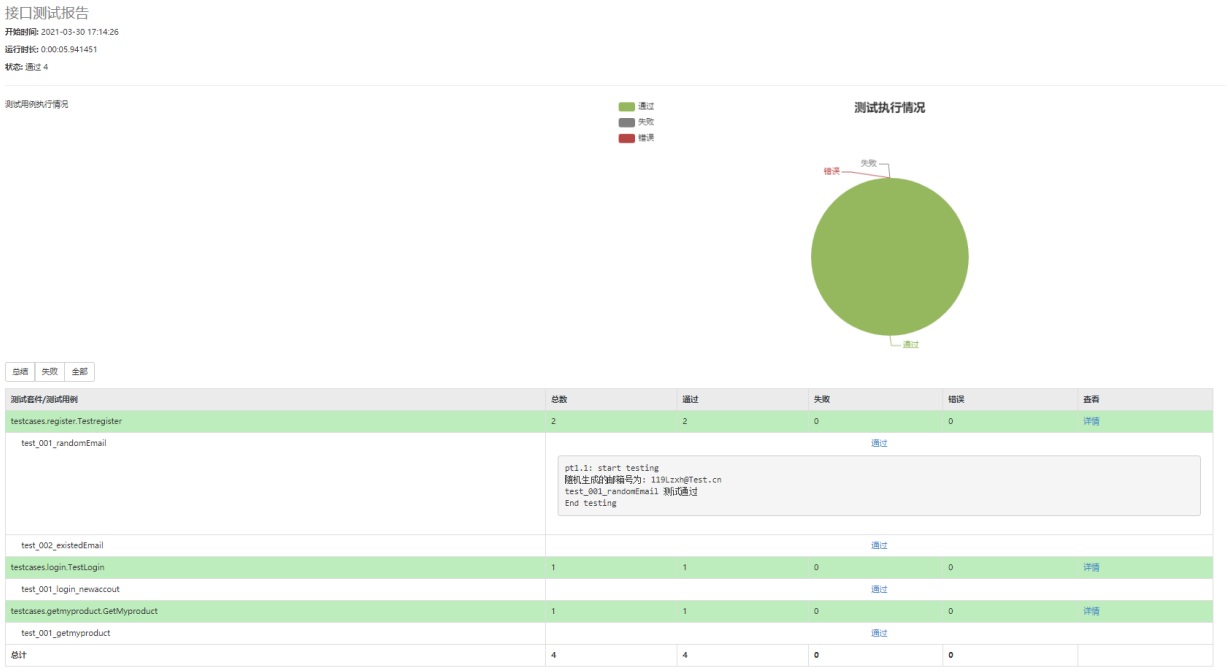
导入测试用例

这里的思路是获取当前最新时间，拼接成最新测试报告存到report1目录下

运行所有测试用例

最终效果：

- 测试通过：



- 测试失败：

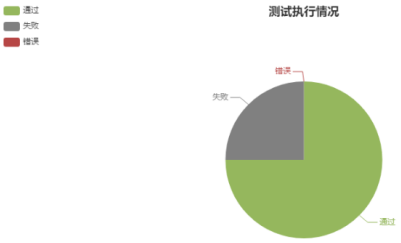
接口测试报告

开始时间: 2021-03-30 17:13:49

运行时长: 0:00:07.940303

状态: 通过 3 失败 1

测试用例执行情况



总结 失败 全部

测试套件/测试用例	总数	通过	失败	错误	查看
testcases.register.Testregister	2	2	0	0	详情
testcases.login.TestLogin	1	1	0	0	详情
testcases.getmyproduct.GetMyproduct	1	0	1	0	详情
test_001_getmyproduct	失败				

```
ft3.1: test_001_getmyproduct 测试不通过
Traceback (most recent call last):
  File "C:\Users\2021\PycharmProjects\api_test\testcases\getmyproduct.py", line 37, in test_001_getmyproduct
    raise e
  File "C:\Users\2021\PycharmProjects\api_test\testcases\getmyproduct.py", line 31, in test_001_getmyproduct
    self.assertEqual('success', product_msg.json()['msg'])
AssertionError: 'success' != 'Error!The authentication failed'
- success
+ Error!The authentication failed
```

总计	4	3	1	0	