Unittest框架

- 讲师：潘sir

简介

- 单元测试框架：一段代码去测试另一段代码，相当于一个模版，我们在模版中填写我们的测试内容，执行测试，并生成测试报告
- 也可用作接口测试框架
- unittest (类似java的Junit)是python自带的，不需要重新搭建
- 怎么搭建框架？：直接import unittest 就相当于搭建完成了

框架使用

- 注意事项：
    1. 文件名不能包含中文
    2. def 都是test开头：如 testAdd1/testxxx

```
import unittest

# 继承
class Dcstest(unittest.TestCase):
    # 测试用例
    def testAdd1(self):
        # python自带的断言方法
        assert 1 + 1 == 3

    # pass
    def testAdd2(self):
        # 框架提供的断言方法
        self.assertEqual((1 + 3), 4)

if __name__ == '__main__':
    # pycharm使用：unittest.main() 即可
    # ipython解释器需要指定参数避免报错
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 写个接口用例

```
import unittest, requests


class Dcstest(unittest.TestCase):
    # 接口测试用例，正常用例
```

```
    def testCheckTotal(self):
        '''校验total值和实际数量是否一致'''
        url = 'http://127.0.0.1:5001/v1/signature'
        username = 'dcs'
        password = '123'
        res = requests.get(url, auth=(username, password))
        a = res.json()
        it = a["items"]
        # python断言
        assert len(it) == a["total"], "数量错误"
        # unittest断言
        self.assertTrue(len(it) == a["total"])

    def testCheckSize(self):
        '''校验page_size是否等于10000'''
        url = 'http://127.0.0.1:5001/v1/signature'
        username = 'dcs'
        password = '123'
        res = requests.get(url, auth=(username, password))
        a = res.json()
        it = a["page_size"]
        # unittest断言
        self.assertTrue(it == 10000)


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

```
testCheckSize (__main__.Dcstest)
校验page_size是否等于10000 ... ok
testCheckTotal (__main__.Dcstest)
校验total值和实际数量是否一致 ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.328s

OK
```

- 引入测试库

```
# from tsms import tsms_base
from tsms.tsms_base import Tsmstest
import unittest, requests
ts = Tsmstest()
class Addsign(unittest.TestCase):
```

```
    def testAddsign(self):
        """创建签名成功"""
        data = {"signature": "测试", "source": "深圳", "pics": []}
        ts.req_post('sign', data)
        assert ts.status_code == requests.codes.ok
        assert isinstance(ts.json["sign_id"], int)

    def testUserfail(self):
        """用户名错误"""
        data = {"signature": "测试", "source": "深圳", "pics": []}
        ts.req_post('sign', data, user='ddd')
        ts.check_fail(400, {'error': 'ER:0001', 'message': 'auth not pass'})

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 练习：手动编写两个用例

前置与后置

- 代码中重复的部分，可以作为前置条件，这样可以优化代码，也方便修改
- 用例前置与后置：每个用例执行前和执行后，都会执行

```
import unittest, requests
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
class Dcstest(unittest.TestCase):
    # 前置
    def setUp(self):
        logging.info("每条测试用例前都会执行")
        logging.info("数据准备工作，准备需要的数据")

    # 后置，-> 是注释，可以不写
    def tearDown(self) -> None:
        logging.info("每条测试用例之后都会执行")
        logging.info("数据清理工作，清理测试产生的垃圾数据")

    # 接口测试用例，正常用例
    def testCheckTotal(self):
        """用例1"""
        b = 1 + 1
        logging.info("我是执行操作")
        assert b==2

if __name__ == '__main__':
    # 执行本suite
```

```
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 练习1：测试删除签名接口，在执行删除前，先创建一个签名。

```
# from tsms import tsms_base
from tsms.tsms_base import Tsmstest
from tsms.tsms_decorator import logging
import unittest, requests
ts = Tsmstest()


class Delsign(unittest.TestCase):

    def setUp(self):
        # 创建一个随机签名
        ts.req_post("sign", ts.sign_data)

    def tearDown(self) -> None:
        pass

    def testDelsign(self):
        """删除签名成功"""
        ts.req_delete("sign", ts.json)
        assert ts.status_code == requests.codes.ok
        assert ts.text == 'ok'

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 练习2：添加后置条件，无论签名删除是否成功，都把数据清理干净

```
# from tsms import tsms_base
from tsms.tsms_base import Tsmstest
from tsms.tsms_decorator import logging
import unittest, requests
ts = Tsmstest()


class Delsign(unittest.TestCase):

    def setUp(self):
        # 创建一个随机签名
        self.new_sign_id_dict = ts.req_post("sign", ts.sign_data)
```

```
    def tearDown(self) -> None:
        try:
            ts.req_delete("sign", self.new_sign_id_dict)
        except:
            logging.warn("删除失败")

    def testDelsign(self):
        """删除签名成功"""
        ts.req_delete("sign", ts.json)
        assert ts.status_code == requests.codes.ok
        assert ts.text == 'ok'

    def testDelNoExistSign(self):
        """删除不存在的签名"""
        ts.req_delete("sign", {"sign_id": 7})
        ts.check_fail(403, {'error': 'ER:0012', 'message': 'delete sign
fail'})


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 练习3：测试审核接口，编写前置：创建签名，后置：删除签名

```
# from tsms import tsms_base
from tsms.tsms_base import Tsmstest
from tsms.tsms_decorator import logging
import unittest, requests
ts = Tsmstest()


class Reviewsign(unittest.TestCase):

    def setUp(self):
        # 创建一个随机签名
        self.new_sign_id_dict = ts.req_post("sign", ts.sign_data)
        self.review_id = self.new_sign_id_dict["sign_id"]

    def tearDown(self) -> None:
        try:
            ts.req_delete("sign", self.new_sign_id_dict)
        except:
            logging.warn("删除失败")

    def testReviewSign(self):
        """删除签名成功"""
        ts.review("sign", self.review_id, "passed")
```

```
        assert ts.status_code == requests.codes.ok
        assert ts.text == 'ok'
        # 通过接口结果验证是否成功
        ts.tsms_get("sign")
        audit_status = ts.get_field("audit_status", id=self.review_id)
        assert audit_status == "passed"

    def testReviewFailSignid(self):
        """删除不存在的签名"""
        ts.review("sign", "100", "passed")
        ts.check_fail(400, {"error": "ER:0004", "message": "prams fail"} )

    def testUserfail(self):
        """删除不存在的签名"""
        ts.review("sign", self.review_id, "passed", user=ts.gen_ranstr(2,2))
        ts.check_fail(400, {"error": "ER:0003", "message": "please use root
user request"}  )


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 套件前置与后置：所有用例执行前，执行一次；所有用例执行完毕后，执行一次

```
from tsms.tsms_base import Tsmstest
from tsms.tsms_decorator import logging
import unittest, requests
ts = Tsmstest()


class CaptainTestCase(unittest.TestCase):
    # 用例前置
    def setUp(self) -> None:
        logging.info("用例前置")
        pass

    # 用例后置
    def tearDown(self) -> None:
        logging.info("用例后置")

    # 套件前置，所有用例前执行一次，需要用classmethod装饰(固定语法)
    @classmethod
    def setUpClass(cls) -> None:
        logging.info("所有用例前执行一次")

    # 套件后置
    @classmethod
```

```
    def tearDownClass(cls) -> None:
        logging.info("所有用例执行完成后执行一次")
        logging.info("关闭数据库连接")

    # 接口测试用例，正常用例
    def testCheckTotal(self):
        assert 1==1


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

```
2019-10-12 21:06:56,103 INFO     所有用例前执行一次
testCheckTotal (__main__.CaptainTestCase) ... 2019-10-12 21:06:56,104 INFO
用例前置
2019-10-12 21:06:56,104 INFO     用例后置
ok
2019-10-12 21:06:56,106 INFO     所有用例执行完成后执行一次
2019-10-12 21:06:56,106 INFO     关闭数据库连接

----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
```

- 练习：前置登录tsms网站，登录完成之后，保存token；因为只需要登录一次，即可重复使用token获取页面数据

```
from tsms.tsms_base import Tsmstest
from tsms.tsms_decorator import logging
import unittest, requests, re
ts = Tsmstest()


class CaptainTestCase(unittest.TestCase):
    # 用例前置
    def setUp(self) -> None:
        logging.info("用例前置")
        pass

    # 用例后置
    def tearDown(self) -> None:
        logging.info("用例后置")

    # 套件前置，所有用例前执行一次，需要用classmethod装饰(固定语法)
```

```
    @classmethod
    def setUpClass(cls) -> None:
        """前置完成登录"""
        cls.s = requests.session()
        url = 'http://127.0.0.1:5001/login'
        data = {
            "username": "dcs",
            "password": "123",
        }
        r = cls.s.get(url)
        csrf_token = re.findall(r'csrf_token.*?value="(.*?)">-', r.text)
        data["csrf_token"] = csrf_token
        r2 = cls.s.post(url, data=data)

    # 套件后置
    @classmethod
    def tearDownClass(cls) -> None:
        logging.info("所有用例执行完成后执行一次")

    # 接口测试用例，正常用例
    def testAddsign1(self):
        sign_data = ts.sign_data
        ts.req_post("sign", sign_data)
        # 打开页面
        a = self.s.get("http://127.0.0.1:5001/user/dcs/sign")
        logging.info("当前的签名内容为: {}".format(sign_data["signature"] ))
        # 断言随机的签名内容在前端页面可查
        assert sign_data["signature"] in a.text

    def testAddsign2(self):
        sign_data = {"signature": "hellokitty", "source": "深圳", "pics": []}
        ts.req_post("sign", sign_data)
        # 打开页面
        a = self.s.get("http://127.0.0.1:5001/user/dcs/sign")
        # 断言随机的签名内容在前端页面可查
        assert "hellokitty" in a.text

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

断言

```
import unittest
class Test(unittest.TestCase):
    def test01(self):
```

```
        '''判断 a == b '''
        a = 1
        b = 1
        self.assertEqual(a, b)

    def test02(self):
        '''判断 a in b '''
        a = "hello"
        b = "hello world!"
        self.assertIn(a, b)

    def test03(self):
        '''判断 a is True '''
        a = True
        self.assertTrue(a)

    def test04(self):
        '''失败案例'''
        a = "中文"
        b = "dsc"
        self.assertEqual(a, b)

if __name__ == "__main__":
    unittest.main(argv=['ignored', '-v'], exit=False)
```

汇总

1. assertEqual(self, first, second, msg=None)：判断两个参数相等：first == second
2. assertNotEqual(self, first, second, msg=None)：判断两个参数不相等：first ！ = second
3. assertIn(self, member, container, msg=None)：判断是字符串是否包含：member in container
4. assertNotIn(self, member, container, msg=None)：判断是字符串是否不包含：member not in container
5. assertTrue(self, expr, msg=None)：判断是否为真：expr is True
6. assertFalse(self, expr, msg=None)：判断是否为假：expr is False
7. assertIsNone(self, obj, msg=None)：判断是否为None：obj is None
8. assertIsNotNone(self, obj, msg=None)：判断是否不为None：obj is not None

pycharm执行

1. pycharm指定脚本运行方式：preferences -> Tools -> Python Intergrated Tools -> 选择自己的项目 -> Default Test runner：选择unittest
2. 注意：先选择运行方式，再新建用例文件
3. 运行方式：
    - 点击左边的绿色三角箭头，即可执行对应用例
    - 在合适的位置，右击，选择：Run ...
        1. 右击 def 那一行，选择：Run ...，执行单条

2. 右击 def 下面部分，选择：Run ...，执行全部用例
- 直接执行脚本，通过：unittest.main() 运行全部

项目设计

目录结构

APITestPro/ # 项目文件 ├── cases # 测试用例 | └── **init**.py ├── common # 公共模块(方法) | └── **init**.py ├── report # 测试报告 └── run_all_cases.py # 执行脚本

补充说明

- 写自动化就是把功能测试用例翻译成python代码，所以自动化难点不是在写用例，而是在翻译，当然前提是功能测试用例要会写
- 测试用例设计需要注意，用例之间要相互独立，不要相互依赖