小白学Flask第十五天| 重要环节---单元测试！

原创JAP君 Python进击者
2019-12-02原文

Flask系列文章：

小白学Flask第一天 | 我的第一个Flask程序

小白学Flask第二天| app对象的初始化和配置

小白学Flask第三天| 今天把视图函数的路由给讲清楚！

小白学Flask第四天| 把路由转换器玩的更牛逼

小白学Flask第五天 | 详解很重要的request对象

小白学Flask第六天| abort函数、自定义错误方法、视图函数的返回值

小白学Flask第七天| 讲讲cookie和session的操作

小白学Flask第八天| Flask上下文和请求钩子

小白学Flask第九天| 看看模板的那些事（一）

小白学Flask第十天| 宏、继承、包含、特殊变量

小白学Flask第十一天| flask-sqlalchemy数据库扩展包(一)

小白学Flask第十二天| flask-sqlalchemy数据库扩展包(二)

小白学Flask第十三天| 来谈谈数据库迁移、邮箱扩展的那些事！

小白学Flask第十四天 | 一文带你彻底了解蓝图是啥！

我用Flask写了一个图书作者管理项目(附完整代码)

主要内容：

1.为什么要测试？测试分为哪几种？

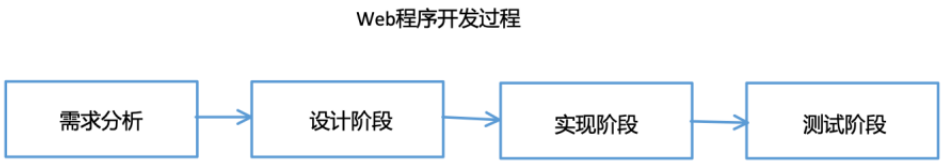
2.单元测试是个啥？

3.断言assert

4.简单的单元测试案例

为什么要测试？测试分为哪几种？

Web程序开发过程一般包括以下几个阶段：
需求分析，设计阶段，实现阶段，测试阶段



其中测试阶段通过人工或自动来运行测试某个系统的功能。目的是检验其是否满足需求，并得出特定的结果，以达到弄清楚预期结果和实际结果之间的差别的最终目的。

测试也是分种类的，测试从软件开发过程可以分为：**单元测试、集成测试、系统测试等**。

在众多的测试中，**与程序开发人员最密切的就是单元测试，因为单元测试是由开发人员进行的**，而其他测试都由专业的测试人员来完成。所以我们主要学习单元测试。

单元测试是个啥？

测试大家想必很耳熟，无非就是来检验我们的代码是否能够完成我们指定的任务或者如何才能更加高效的完成我们的任务。

但是单元测试大家可能有点陌生，举个简单的例子，**一部手机有许多零部件组成，在正式组装一部手机前，手机内部的各个零部件，CPU、内存、电池、摄像头等，都要进行测试，这就是单元测试。**

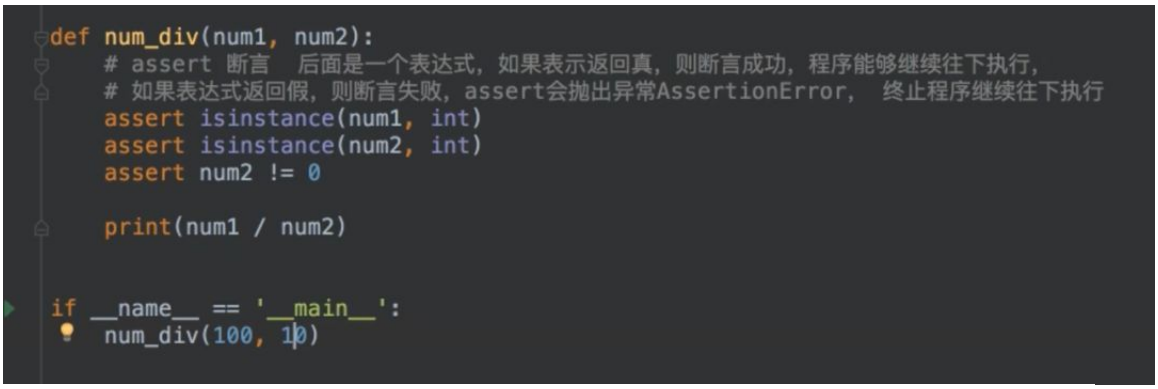
单元测试就是**开发者编写一小段代码，检验目标代码的功能是否符合预期。**
通常情况下，单元测试主要面向一些功能单一的模块进行。

断言assert

在Web开发过程中，单元测试实际上就是一些“断言”（assert）代码。

断言就是判断一个函数或对象的一个方法所产生的结果是否符合你期望的那个结果。python中assert断言是声明布尔值为真的判定，如果表达式为假会发生异常。单元测试中，一般使用assert来断言结果。

例如下面的代码：



如果num1或num2不为整数，或者num2等于0，那么就会报出AssertionError错误。

断言也是有一些常用的方法：

assertEqual	如果两个值相等，则pass
assertNotEqual	如果两个值不相等，则pass
assertTrue	判断bool值为True，则pass
assertFalse	判断bool值为False，则pass
assertIsNone	不存在，则pass
assertIsNotNone	存在，则pass

简单的单元测试案例

这里给大家写了一个简单的登录案例，大致情况是测试登录过程中有无填写账号密码、账号密码有无错误，具体大家可以看代码。

login.py

```
# coding:utf-8
```

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/login", methods=["POST"])
def login():
    # 接收参数
    username = request.form.get("username")
    password = request.form.get("password")

    # 参数判断
    if not all([username, password]):
        resp = {
            "code":1,
            "message":"invalid params"
        }
        return jsonify(resp)

    if username == "admin" and password =="javaandpython":
        resp = {
            "code": 0,
            "message": "login success"
        }
```

```
        return jsonify(resp)
    else:
        resp = {
            "code": 2,
            "message": "login failed"
        }

        return jsonify(resp)

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

test.py

```
import unittest
from login import app
import json

class LoginTest(unittest.TestCase):
    '''构造单元测试'''
    def test_user_pass_isempty(self):
        '''测试用户和密码是否完整'''
        # 创建进行web请求的客户端，使用flask提供的
        client = app.test_client()
```

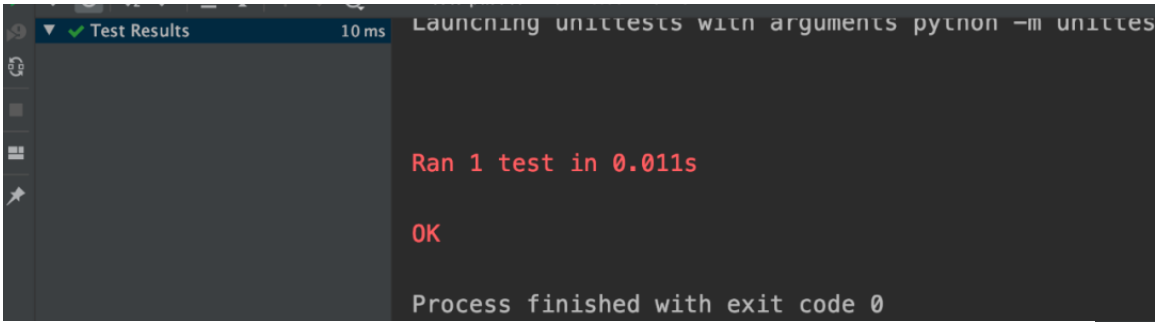
```
        # 利用client客户端模拟发送web请求
        ret = client.post("/login", data={}) # 用户名和密码都为空

        # ret是视图返回的响应对象，data属性是响应体的数据
        resp = ret.data
        # 解析json
        resp = json.loads(resp)

        # 开始进行断言
        self.assertIn("code", resp)
        self.assertEqual(resp["code"], 1)

if __name__ == '__main__':
    # 直接通过下面方法进行测试
    unittest.main()
```

我们可以直接通过运行test.py文件来测试login方法是否编写正确。



例如上图就说明整个测试是成功的。

大家可能注意到，我们上面的代码只是测试了用户名和密码都为空的情况，作为单元测试，我们需要把其他几种情况都列举出来，如密码为空，用户名不为空。这样才能够体现出测试的完整性。

在测试类中，有两个固定的方法：

```
import unittest
class TestClass(unittest.TestCase):


    #该方法会首先执行，方法名为固定写法
    def setUp(self):
        pass


    #该方法会在测试代码执行完后执行，方法名为固定写法
    def tearDown(self):
        pass
```

还是按照上面登录的例子来看

```
import unittest
from login import app
import json


class LoginTest(unittest.TestCase):
    '''构造单元测试'''
```

```
def setUp(self):
    # 设置flask工作在测试模式下
    app.testing = True

    # 创建进行web请求的客户端，使用flask提供的
    self. client = app.test_client()

def test_user_pass_isempty(self):
    '''测试用户和密码是否完整'''
    # 利用client客户端模拟发送web请求
    ret = self.client.post("/login", data={})

    # ret是视图返回的响应对象，data属性是响应体的数据
    resp = ret.data

    # 解析json
    resp = json.loads(resp)

    # 开始进行断言
    self.assertIn("code", resp)
    self.assertEqual(resp["code"], 1)

def test_user_pass_isright(self):
    '''测试用户名或密码是否正确'''
```

```
        # 利用client客户端模拟发送web请求
        ret = self.client.post("/login", data={"username" :
"admin"})
        # ret是视图返回的响应对象，data属性是响应体的数据
        resp = ret.data

        # 解析json
        resp = json.loads(resp)

        # 开始进行断言
        self.assertIn("code", resp)
        self.assertEqual(resp["code"], 1)

if __name__ == '__main__':
    # 直接通过下面方法进行测试
    unittest.main()
```

可以从上面的代码中发现，setUp里面的所写的是多个测试函数中可能重复出现的代码。

本Flask系列文章终于接近了尾声，有关于flask的基本知识在这十五天文章里面都写了出来，非常感谢大家的支持。在这个系列的文章写作中，收到了很多读者朋友的认可和支持，所以我才有动力把它给完成。

除了这十五篇flask文章，日后还会有一些补充的知识，也可能会带着大家写一个Flask项目，所以转发、在看！继续给我动力！！！

精选留言

暂无...