#### ddt

```
1
 2 #!/usr/bin/env python
 3 # -*- coding:utf-8 -*-
4 #===#===#===#===
5 #Author:
   #CreatDate:
   #version:
8 #====#===#===
9
10 import unittest
   import ddt
11
12
13 test_data=['hello','world']
14
15
   @ddt.ddt#代表下面这个单元测试支持ddt驱动
   class Testmaker(unittest.TestCase):
16
17
       @ddt.data(1,2)
18
19
       def test_ddt(self,v):#这个用例会执行2遍
20
           print("test_ddt")
           print(v)#1 2
21
22
23
       (0, 1, 2, 3), [3, 4, 5]
24
       @ddt.unpack
25
       def test_ddt2(self,v1,v2,v3):#data参数中的序列有多少个数,那么这里的参数个数就必须是多少个
26
           print("test_ddt2")
27
           print(v1)
28
           print(v2)
29
30
       @ddt.data(test_data)#参数是列表时
31
       @ddt.unpack
32
       def test_ddt3(self,v1,v2):
33
           print("test_ddt3")
34
           print(v1)
35
           print(v2)
36
37
       @ddt.file_data("mydata.yaml")#参数是文件名
       def test_ddt4(self,txt):#执行了4次,因为mydata.yaml中有4个数据
38
39
           print("test_ddt4")
40
           print(txt)
41
42
   if __name__=='__main___':
       unittest.main()
43
44
45
46
47
```

# unittest使用ddt时,如果使用测试套件执行用例时,会出现添加不了测试用例的问题

```
1 问题: #注意:使用了ddt加载数据后,会改变测试用例的名字,导致添加进测试套件中出错
   #!/usr/bin/env python
3 # -*- coding:utf-8 -*-
4 #====#===#====#
   #Author:
6 #CreatDate:
   #Version:
8 #====#===#===
9
10 import unittest
11 import ddt
12
13 test_data=['hello','world']
14
15
   @ddt.ddt#代表下面这个单元测试支持ddt驱动
   class Testmaker(unittest.TestCase):
16
17
18
       @ddt.data(1,2)
19
       def test_ddt(self,v):#这个用例会执行2遍
           print("test_ddt")
20
21
           print(v)#1 2
22
23
       (0, 1, 2, 3), [3, 4, 5]
24
       @ddt.unpack
25
       def test_ddt2(self,v1,v2,v3):#data参数中的序列有多少个数,那么这里的参数个数就必须是多少个
26
           print("test_ddt2")
27
           print(v1)
28
           print(v2)
29
       @ddt.data(test_data)#参数是列表时
30
31
       @ddt.unpack
       def test_ddt3(self,v1,v2):
32
33
           print("test_ddt3")
           print(v1)
34
35
           print(v2)
36
       @ddt.file_data("mydata.yam1")#参数是文件名
37
38
       def test_ddt4(self,txt):#执行了4次,因为mydata.yaml中有4个数据
           print("test_ddt4")
39
40
           print(txt)
41
   if __name__=='__main___':
42
43
       #注意:使用了ddt加载数据后,会改变测试用例的名字,导致添加进测试套件中出错
44
       # 生成测试套件(也叫测试集合)
45
       suitt = unittest.TestSuite()
46
       suitt.addTests(map(Testmaker, ["test_ddt", "test_ddt3", "test_ddt4"]))
47
48
       re = unittest.TestResult()
49
       suitt.run(re)
```

```
50
51 解决方法:
52
   第一种方式,使用路径添加模块方式生成测试套件,这个方式不需要测试用例的名字
53 | if __name__=='__main__':
54
       suit = unittest.defaultTestLoader.discover(r'./', pattern="03unit.py")
55
       re = unittest.TestResult()
56
       suit.run(re)
57
58 第二种方式:
59
   if __name__=='__main___':
       suit = unittest.TestSuite()
60
61
       #获取ddt驱动之后的测试用例名字
62
       mylist=unittest.TestLoader().getTestCaseNames(Testmaker)
63
       print(mylist)
64
       suit.addTests(map(Testmaker,mylist))
65
       re = unittest.TestResult()
66
       suit.run(re)
```

#### 断言

```
断言:一个自动化测试用例,测试步骤、测试的断言缺一不可
2
   unittest中提供的断言方法有:
       assertEqual(a,b,msg=""):就是判断a和b是否相等,如果相等,则断言成功,如果不相等,会断言失
3
   败,并且输出msg消息
       assertNotEqual(a,b,msg=""): 就是判断a和b是否不相等
4
5
       assertTrue(a):就是判断a是否为True这个bool值
6
       assertFalse(a):就是判断a是否为False这个bool值
7
       assertIs(a,b,msg=""):判断a和b的内存地址是否相等,如果相等则身份一致
       assertIsNot(a,b,msg):判断a和b的内存地址是否不相等,如果像等了,返回false,断言失败
8
9
       assertIsNone(a):判断对象a是不是空指针(没有指向堆内存中空间),如果是则断言成功
10
       assertIsNotNone(a):判断对象a是不是空指针,如果是,则断言失败
11
       assertIn(a,b):判断a是不是b的成员,如果是则断言成功
12
       assertNotIn((a,b):判断a是不是b的成员,如果不是则断言成功
13
       assertIsInstance(a,b):判断a是b的一个实例对象
14
       assertIsNotInstance(a,b):判断a不是b的一个实例对象
15
16 #!/usr/bin/env python
17 # -*- coding:utf-8 -*-
18 #===#===#===#
19
   #Author:
20 #CreatDate:
21 #Version:
22 #===#===#===
23 #场景:百度首页中点击新闻链接,测试新闻页面打开,并且正确,正确的话就是url是
   https://news.baidu.com/,
24 #而且浏览器有2个句柄
25
26 | import unittest
   from selenium import webdriver
27
28
   import time
```

```
29
    class Maker(unittest.TestCase):
30
31
        def setUp(self):
            self.dr=webdriver.Firefox()
32
33
            self.dr.get("https://www.baidu.com")
34
            time.sleep(2)
35
36
        def tearDown(self):
            time.sleep(3)
37
            self.dr.quit()
38
39
        def test_urlnew(self):
40
41
            self.dr.find_element_by_link_text('新闻').click()
42
            time.sleep(2)
            #获取浏览器的句柄个数
43
            mylist=self.dr.window_handles
44
45
            # 预期结果
46
            n=2
47
            #测试结果
            n2=len(mylist)
48
            self.assertEqual(n,n2,"测试不通过1")
49
50
            self.dr.switch_to.window(mylist[1])
51
            #预期url
52
            myurl="https://news.baidu.com/"
            #测试结果
53
            myurl2=self.dr.current_url
54
55
            self.assertEqual(myurl, myurl2, '测试不通过2')
56
57
    if __name__=='__main___':
58
59
        unittest.main()
60
61
62
```

#### POM模式介绍

9

```
虽然unittest单元测试框架在一定层度上解决了大量用例(100条、500条甚至几千条用例)所带来的用例管理
  和执行上的麻烦。
2
3
  但是UI的元素属性变化带来的麻烦,就有心无力了,比如涉及到500条case的页面元素属性变化,现有脚本的定
  位出现问题,那岂不是要改500遍,那是不是会改到心碎。这种怎么解决呢?
4
  使用POM模式
5
  POM设计模式,即Page Object Model,这也是是目前最为经典的一种设计思想,用大白话说就是:将页面UI
  元素对象、逻辑、业务、数据等分离开来,使得代码逻辑更加清晰,复用性,可维护性更高的一种方法。
6
7
  POM模式的优点
8
  让UI自动化更早介入项目中,可项目开发完再进行元素定位的适配与调试
```

10 如果UI页面元素更改,测试脚本不需要更改,只需要更改页面对象中的某些代码就可以

POM 将页面元素定位和业务操作流程分开,分离了测试对象和测试脚本

```
11
   POM能让我们的测试代码变得可读性更好, 高可维护性, 高复用性,
   可多人共同维护开发脚本,利于团队协作
12
13
14
   ddt读取yam1文件内容
15
   1. 每组数据用'-'隔开,一个-就是一组数据
16
   2.读取yam1文件必须是test开头的函数
17
18
   要独立创建项目,不然导入模块会出问题
19
20
   项目下有4个目录:
21
   base目录,里面的文件都是写selenium相关的代码,如元素定位,操作元素等
22
   data目录,里面是存储数据的文件
23
   cases目录,里面是业务代码,就是在单元测试类中写测试用例
24
   page_Object目录:这里是封装业务代码的
25
26
   pom模式是把selenium的api,数据,页面元素的属性,业务这些分离开来,
27
   如果seleniium的api改变了,只需修改base里的代码
28
   如果数据变了,只需修改data里的内容
29
   如果页面元素的属性变了,只需修改page_Object里的代码
30
   如果业务代码变了,只需修改cases里的内容
31
32 base:
33
   #!/usr/bin/env python
34 # -*- coding:utf-8 -*-
35
   #====#====#====
36 #Author:
37 #CreatDate:
38
   #version:
39
   #====#====#====
   #页面操作或定位元素等代码都写到这
40
41
   from selenium import webdriver
42
   class BasePage():
43
       def __init__(self,url):
44
          self.dr=webdriver.Firefox()
45
          self.dr.get(url)
46
47
       #*lcator是(By.ID,"kw")
48
       def find_element(self,*lcator):
49
          e=self.dr.find_element(lcator)
50
          return e
51
       def mysend_keys(self,lcator,value,is_clear=True):
52
53
          e=self.dr.find_element(*lcator)
54
          if is_clear:
55
              e.clear()
56
              e.send_keys(value)
57
          else:
58
              e.send_keys(value)
59
60
61
   page_Object目录:
   #!/usr/bin/env python
62
   # -*- coding:utf-8 -*-
63
```

```
64 #====#===#===
65
     #Author:
66
    #CreatDate:
    #Version:
67
68
    #====#====#====
     from base.base_page import BasePage
69
70
    from selenium.webdriver.common.by import By
 71
     class Seach_page(BasePage):
         def __init__(self,url):
72
73
             BasePage.__init__(self,url)
74
             self.name_locat=(By.ID,'kw')
75
 76
         def input_name(self,value):
             self.mysend_keys(self.name_locat,value)
77
78
79
     cases目录:
    #!/usr/bin/env python
80
81
     # -*- coding:utf-8 -*-
     #====#====#====
82
83
     #Author:
84
     #CreatDate:
85 #Version:
86
     #====#====#====
87
     from page_Object.seach_page import Seach_page
88
     import unittest
89
90
     import ddt
91
92
     @ddt.ddt
93
     class Maker(unittest.TestCase):
94
         @classmethod
95
         def setUpClass(cls):
             cls.res=Seach_page("https://www.baidu.com")
96
97
         @ddt.file_data("../data/mydata.yaml")
98
         def test_res(self,txt):
99
100
             self.res.input_name(txt)
101
     if __name__=='__main___':
102
103
         unittest.main()
```

# pytest的介绍

```
      1
      pytest是Python的一款单元测试框架,一种全新的框架思维来管理和规范我们的测试脚本,从而实现高类聚低耦合的理念

      2

      3
      优势:

      4
      1.通用性

      5
      2.丰富的第三方库

      6
      3.入门快速

      7
      4.精美的报告

      8
      5.定制性强
```

## pytest的安装

```
1安装方式一:21.pycharm上创建一个文件,输入import pytest32.鼠标点击pytest,出现小灯泡,点击小灯泡进行下载4安装方式二:61.pycharm上终端,输入:pip install -U pytest7安装方式三:8安装方式三:91.pycharm上的菜单-文件-设置-项目xxx-project Interpreter中下载
```

## pytest的约束

```
1.所有的单测文件名都需要满足test_*.py格式或*_test.py格式。
2.在单测文件中,测试类以Test开头,并且不能带有 init 方法(注意:定义class时,需要以Test开头,不然 pytest是不会去运行该class的)
3.在单测类中,可以包含一个或多个test_开头的函数。
此时,在执行pytest命令时,会自动从当前目录及子目录中寻找符合上述约束的测试函数来执行。
5
```

## pytest的helloworld

```
1 目的测试一个加法
2 #!/usr/bin/env python
3 # -*- coding:utf-8 -*-
   #====#====#====
5 #Author:
6 #CreatDate:
   #version:
8 #====#====#====
9
10 | import pytest
11
12 def test_01():
       print("hello pytest")
13
14
15
      a=10+20
16
       b = 30
       assert a==b
17
18
19 if __name__=='__main___':
20
       #如果使用下面的方式,文件名不受规则限制
21
       pytest.main(["01hellopytest.py","-s"])
22
```

## pytest的命令

```
import pytest
##执行别的文件中所有用例

#pytest.main(["./01hello_pytest.py","-s"])
##查看pytest的版本

#pytest.main(["--version"])
##查看内置函数参数

#pytest.main(["--fixtures"])
#查看帮助信息
#pytest.main(['--help'])
```

# pytest的执行用例

```
1 #!/usr/bin/env python
2 # -*- coding:utf-8 -*-
3 #====#===#====
4 #Author:
5 #CreatDate:
6 #Version:
7 #====#===#====
8 import pytest
9 #1.执行某个目录中所有的用例
10 # pytest.main(["./mycasetest/"])
11 #2.执行某个目录中某个模块的所有用例
# pytest.main(["./mycasetest/test_02.py"])
13 #3.执行某个目录中某个模块的某个用例
# pytest.main(["./mycasetest/test_02.py::test_01"])
15 #4.执行某个目录中某个模块的某几个用例,用例的上面需要标记@pytest.mark.slow,slow自己取的名字
pytest.main(["./mycasetest/test_02.py::test_01","-m","slow"])
17 代码未完成
18
```