

执行方式一(执行所有的测试用例)

- 1 1.执行main()方法执行的特点:unittest.main()
- 2 注意:
- 3 1.是把所有的测试用例执行了一遍
- 4 2.执行测试用例的顺序控制不了,(按照测试用例名(方法名)的字母顺序执行的)

执行方式二(内部方法添加测试用例)(重点)

```
1 通过内部函数添加测试用例
2 步骤:
3 1.生成测试套件(也叫测试集合)
4     suitt = unittest.TestSuite()
5 2.把测试用例添加进测试集合,两种方式添加
6     1.suitt.addTest(类名("用例"))
7     2.suitt.addTests(map(类名,["用例1","用例2",...]))
8 3.生成测试结果对象,然后传递到run函数中
9     re = unittest.TestResult()
10    suitt.run(re)
11
12 #!/usr/bin/env python
13 # -*- coding:utf-8 -*-
14 #====#====#====#====
15 #Author:
16 #CreatDate:
17 #Version:
18 #====#====#====#====
19
20 import unittest
21
22 class Maker(unittest.TestCase):
23     def test_01(self):
24         print("test_01.....")
25
26     def test_02(self):
27         print("test_02.....")
28
29     def test_03(self):
30         print("test_03.....")
31
32     def test_04(self):
33         print("test_04.....")
34
35     def test_05(self):
36         print("test_05.....")
37
38 if __name__=='__main__':
39     # 1.
```

```

40     # 生成测试套件(也叫测试集合)
41     suitt = unittest.TestSuite()
42     # 2.
43     # 把测试用例添加进测试集合, 两种方式添加
44     # 1.
45     # suitt.addTest(Maker("test_03"))
46     # 2.
47     suitt.addTests(map(Maker, ["test_01", "test_03", "test_05"]))
48     # 3.
49     # 生成测试结果对象, 然后传递到run函数中
50     re = unittest.TestResult()
51     suitt.run(re)
52     #需要使用python中的IDLE执行,或直接终端执行这个文件

```

执行方式二(通过模块方法添加测试用例)(重点)

```

1  通过模块方法添加测试用例
2  1.如果测试用例的数量比较大,使用testsuite自带的方法加用例到集合,很麻烦
3  可以unittest中提供的testloader模块,提供了好多帮我们把测试用例加载到测试集合中的方法
4  2.步骤:
5      2.创建testloader的对象
6      3.使用testloader的对象中的loadTestsFromName函数添加测试用例,这个api,返回测试套件
7          1.可以添加整个模块的测试用例
8          2.也可以添加一个单元测试类中的所有测试用例(只能添加一个单元测试类)
9          3.也可以添加单元测试类中的某个测试用例(只能添加一个单元测试类中的一个测试用例)
10
11     4.生成测试结果对象,然后传递到run函数中
12         re = unittest.TestResult()
13         suitt.run(re)
14
15
16  模块代码:
17  #!/usr/bin/env python
18  # -*- coding:utf-8 -*-
19  #====#====#====#====
20  #Author:
21  #CreateDate:
22  #Version:
23  #====#====#====#====
24
25  import unittest
26
27  class Maker(unittest.TestCase):
28      def test_01(self):
29          print("test_01.....")
30
31      def test_02(self):
32          print("test_02.....")
33
34      def test_03(self):
35          print("test_03.....")
36

```

```

37     def test_04(self):
38         print("test_04.....")
39
40     def test_05(self):
41         print("test_05.....")
42
43
44 class Maker2(unittest.TestCase):
45     def test_01(self):
46         print("test_01...Maker2...")
47
48     def test_02(self):
49         print("test_02....Maker2..")
50
51     def test_03(self):
52         print("test_03...Maker2...")
53
54     def test_04(self):
55         print("test_04...Maker2...")
56
57     def test_05(self):
58         print("test_05...Maker2...")
59
60 业务代码:
61 #!/usr/bin/env python
62 # -*- coding:utf-8 -*-
63 #====#====#====#====
64 #Author:
65 #CreatDate:
66 #Version:
67 #====#====#====#====
68 import unittest
69 if __name__=='__main__':
70     #创建loader对象
71     loader=unittest.TestLoader()
72     #参数为模块名,返回测试套件,执行一个模块中所有的用例
73     suit=loader.loadTestsFromName("Maker2Tests")
74     #执行一个模块中的某个单元测试类中的所有用例
75     # suit=loader.loadTestsFromName("Maker2Tests.Maker")
76     #执行一个模块中的某个单元测试类中的某个用例
77     suit=loader.loadTestsFromName("Maker2Tests.Maker.test_05")
78
79     # 生成测试结果对象, 然后传递到run函数中
80     re = unittest.TestResult()
81     suit.run(re)
82
83

```

执行方式二(通过路径方式添加测试用例)(重点)

- 1 使用unittest.defaultTestLoader对象的discover方法加载用例, 可以将指定路径所有符合匹配规则 (pattern) 的文件中的单元测试用例一次性加载

```

2  第一个参数是一个目录，这个目录下可以有单元测试用例的文件（.py）
3  第二个参数是填文件名，可以通配
4  suitt = unittest.defaultTestLoader.discover(r"./Maker/", pattern="unit*.py")
5  说明：
6      1."unit*.py"指的是以unit开头，以.py结尾的文件
7      2..py中的单元测试用例要使用unittest框架写的测试用例
8
9  步骤：
10     1.生成测试套件(也叫测试集合)
11     2.使用discover方法批量添加
12     3.生成测试结果对象，然后传递到run函数中
13         re = unittest.TestResult()
14         suitt.run(re)
15
16  #!/usr/bin/env python
17  # -*- coding:utf-8 -*-
18  #====#====#====#====
19  #Author:
20  #CreateDate:
21  #Version:
22  #====#====#====#====
23
24  import unittest
25
26  if __name__=='__main__':
27      suit=unittest.defaultTestLoader.discover(r'./MyTestFile/',pattern="unit*.py")
28      print(suit.countTestCases())
29      # 生成测试结果对象，然后传递到run函数中
30      re = unittest.TestResult()
31      suit.run(re)
32
33
34

```

执行测试用例并生成执行报告

```

1  在前面测试用例、测试集合执行的时候都是用testsuite()的run()方法：suitt.run(result),如果要生成
   text文本形式的测试执行报告，可以使用TestRunner
2  将最后执行的：
3      re = unittest.TestResult()
4      suitt.run(re)
5  改为：
6  with open(r"./re.txt", "w", encoding="utf-8") as f:
7      runner = unittest.TextTestRunner(f, descriptions="单元测试报告执行", verbosity=5)
8      runner.run(suitt)
9  函数说明：
10 f:文件描述符
11 descriptions:用来标记是否输出测试用例的描述信息。布尔类型，没什么用

```

```
12 verbosity参数可以控制输出的错误报告的详细程度，只有3个取值：
13     0 (quiet)： 只显示执行的用例的总数和全局的执行结果。
14     1 (default)： 默认值，显示执行的用例的总数和全局的执行结果，并对每个用例的执行结果（成功T或失败F）有个标注。（测试用例中如果有和预计不同，会出现F）
15     2+ (verbose)： 显示执行的用例的总数和全局的执行结果，并输出每个用例的详细执行结果。
16
17
18 TextTestRunner()-->将结果能够以text文本形式展示的运行器
19
20 #使用TextTestRunner()运行器提供的run()方法运行测试集合
21
22 #如何产生一个文件流对象，如果打开一个文本文件，想着往里写入数据
23
24 #报告是以TextTestResult的形式展示的
25
26 #TextTestRunner是TestRunner的子类
27
28 #TextTestResult是TestResult的子类
29
30
31 如果没有生成1.txt,那么使用python的IDLE执行
32
```

断言

```
1 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：
3     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，会断言失败，并且输出msg消息
4     assertNotEqual(a,b,msg=""):就是判断a和b是否不相等
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
8     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
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
```

HTML格式展示执行报告(重点)

```
1 前面使用runner运行器是unittest自带的，效果不是很好，我们第三方开发的来用，可以以HTML格式展示结果。
2
3 HTMLTestRunner模块：就是一个第三方
4 1、下载该模块的python3版本
```

```

5 2、复制该文件到python的安装目录下/lib中或把HTMLTestRunner.py放到你的项目中
6 3、导包:from HTMLTestRunner import HTMLTestRunner
7
8 #!/usr/bin/env python
9 # -*- coding:utf-8 -*-
10 #====#====#====#====
11 #Author:
12 #CreatDate:
13 #Version:
14 #====#====#====#====
15 import unittest
16 from HTMLTestRunner import HTMLTestRunner
17 import time
18 if __name__=='__main__':
19     suite = unittest.defaultTestLoader.discover(r'./MyTestFile/', pattern="unit*.py")
20
21     filename="."+time.strftime("%Y-%m-%d %H_%M_%S")+"res.html"
22     with open(filename,"wb") as f:
23         runner=HTMLTestRunner(f,verbosity=2,title="单元测试报告",description="第一次运行结果")
24         runner.run(suite)
25
26
27

```

邮件的自动化

```

1 通过python我们可以自动发送报告给负责人
2 Python对SMTP支持有smtpplib和email两个模块，email负责构造邮件，smtpplib负责发送邮件
3 1.把html发送为正文
4 2.把html发送为附件
5
6 需要把发送邮件的邮箱的smtp/pop3的服务开通
7
8 发送为正文:
9 #!/usr/bin/env python
10 # -*- coding:utf-8 -*-
11 #====#====#====#====
12 #Author:
13 #CreatDate:
14 #Version:
15 #====#====#====#====
16 #构建邮件内容的
17 from email.mime.text import MIMEText
18 #构建邮件头部信息的
19 from email.header import Header
20 #构建发件人
21 from email.utils import formataddr
22 #创建发送邮件对象
23 import smtpplib

```

```

24 #把测试报告发送为正文
25
26 #读取报告的内容
27 htmlmsg=open('2023-07-05 20_53_41res.html','rb').read()
28 #构建邮件正文
29 msg=MIMEText(htmlmsg,'html','utf-8')
30 #头部信息
31 msg['Subject']=Header("python邮件自动化",'utf-8')
32 #发件人信息
33 msg['From']=formataddr(['张三202305','76754438@qq.com'])
34 #收件人
35 msg['To']="lzs8407@163.com"
36
37 #构建SMTP对象
38 smtp=smtplib.SMTP()
39 #连接发送邮件的邮箱服务器
40 smtp.connect("smtp.qq.com")
41 #登录
42 smtp.login("76754438@qq.com","uomuthspmlfghbcg")
43 #发送
44 smtp.sendmail("76754438@qq.com","lzs8407@163.com",msg.as_string())
45 #退出
46 smtp.quit()
47
48
49 发送为附件
50 #!/usr/bin/env python
51 # -*- coding:utf-8 -*-
52 #====#====#====#====
53 #Author:
54 #CreatDate:
55 #Version:
56 #====#====#====#====
57 #构建邮件内容的
58 from email.mime.text import MIMEText
59 #构建邮件头部信息的
60 from email.header import Header
61 #构建发件人
62 from email.utils import formataddr
63 from email.mime.multipart import MIMEMultipart
64 #创建发送邮件对象
65 import smtplib
66 #发送附件的对象
67 MYmime=MIMEMultipart()
68 #读取报告的内容
69 htmlmsg=open('2023-07-05 20_53_41res.html','rb').read()
70 #构建邮件正文
71 msg=MIMEText(htmlmsg,'base64','utf-8')
72 msg['Content-Type']="application/octet-stream"
73 msg['Content-Disposition']='attachment;filename=1.html'
74 #头部信息
75 MYmime['Subject']=Header("python邮件自动化",'utf-8')
76 #发件人信息

```

```

77 MYmime['From']=formataddr(['张三20230888','76754438@qq.com'])
78 #收件人
79 MYmime['To']="lzs8407@163.com"
80 MYmime.attach(msg)
81
82
83 #构建SMTP对象
84 smtp=smtpplib.SMTP()
85 #连接发送邮件的邮箱服务器
86 smtp.connect("smtp.qq.com")
87 #登录
88 smtp.login("76754438@qq.com","uomuthspmlfgbhcg")
89 #发送
90 smtp.sendmail("76754438@qq.com","lzs8407@163.com",MYmime.as_string())
91 #退出
92 smtp.quit()
93
94

```

ddt

```

1 ddt是“Data-Driven Tests”的缩写，是unittest中实现数据驱动的主要方式之一，它主要包括如下的装饰器
2
3 @ddt
4 标记测试类，支持DDT数据驱动
5
6 @data
7 标记测试用例，传递参数
8
9 @unpack
10 当@data中的参数是元组、列表时，用于分割序列中的元素
11
12 @file_data
13 标记测试用例，传递文件，支持yaml和json文件
14
15 #!/usr/bin/env python
16 # -*- coding:utf-8 -*-
17 #====#====#====#====
18 #Author:
19 #CreatDate:
20 #Version:
21 #====#====#====#====
22
23 import unittest
24 import ddt
25
26 test_data=['hello','world']
27
28 @ddt.ddt#代表下面这个单元测试支持ddt驱动
29 class Testmaker(unittest.TestCase):
30
31     @ddt.data(1,2)

```



```
32     def test_ddt(self,v):#这个用例会执行2遍
33         print("test_ddt")
34         print(v)#1 2
35
36     @ddt.data((1,2,3),[3,4,5])
37     @ddt.unpack
38     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
39         print("test_ddt2")
40         print(v1)
41         print(v2)
42
43     #代码为完成
44
45 if __name__=='__main__':
46     unittest.main()
47
48
```

##