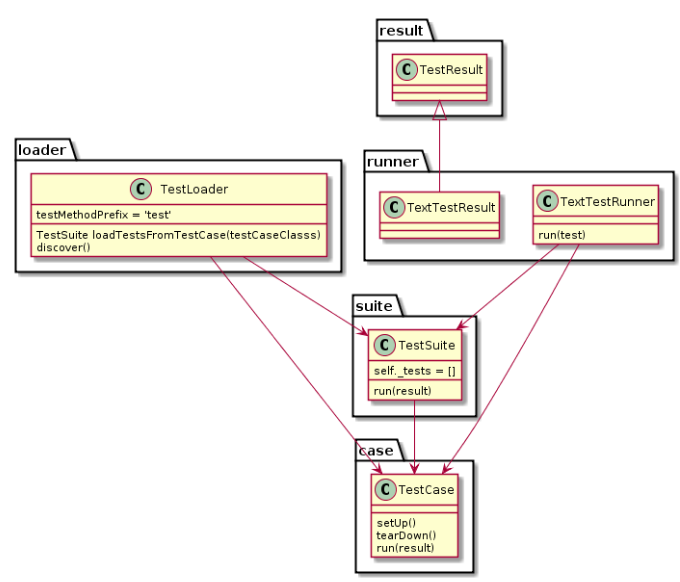
# 原理：

unittest中最核心的四个概念是：test case, test suite, test runner, test fixture。

# 流程：

unittest的流程：写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中，我们通过命令行或者unittest.main()执行时，main会调用TextTestRunner中的run来执行，或者我们可以直接通过TextTestRunner来执行用例



TestCase

TestCase的实例就是一个测试用例。。什么是测试用例呢？就是一个完整的测试流程，包括测试前准备环境的搭建(setUp)，执行测试代码(run)，以及测试后环境的还原(tearDown)。

**一个class继承unittest.TestCase即是一个TestCase，其中以 test 开头的方法在load时被加载为一个真正的TestCase。**

TestSuite

多个测试用例集合在一起，就是TestSuite，而且TestSuite也可以嵌套TestSuite。

TestLoader

是用来加载TestCase到TestSuite中的

其中有几个loadTestsFrom\_\_()方法，就是从各个地方寻找TestCase，创建它们的实例，然后add到TestSuite中，再返回一个TestSuite实例。

TextTestRunner

是来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的run(result)方法。 测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

test fixture

对一个测试用例环境的搭建和销毁，是一个fixture。

# 一、测试固件，TestFixture

setUp()

tearDwon()

#每个用例都执行

@classmethod

setUpCalss(cls)

tearDwonClass(cls)

#最开始和结束的时候执行

# 二、测试执行 main

if \_\_name\_\_='main':

unittest.main(verbosity=2)

# unittest.main()执行时，main会调用TextTestRunner中的run来执行

# 或者我们可以直接通过TextTestRunner来执行用例。

unittest.TextTestRunner(verbosite =2 ).run(suite)

unittest模块中包含main方法，main使用unittest.TestLoader类来自动查找和加载模块内的测试用例

verbosity默认是 1

如果设为 0，则不输出每一用例的执行结果

如果设为 2，则输出详细的执行结果

# 测试套件 TestSuite

## 按序：

suite = unittest.TestSuite()

suite.addTest(测试类1('测试方法名1'))

suite.addTest(测试类2('测试方法名2'))

unittest.TextTestRunner(verbosite =2 ).run(suite)

或者用“列表添加”

tests = [TestMathFunc("test\_add"), TestMathFunc("test\_minus")]

suite.addTests(tests)

## 按测试类makeSuite

把测试类中的用例组成TestSuite

suite = unittest.TestSuite (unittest.makeSuite(测试类))

unittest.TextTestRunner(verbosite =2 ).run(suite)

#makeSuite方法的参数是testCaseClass，也就是测试类

## 按测试类加载 TestLoader

suite =unittest.TestLoader().loadTestsFromTestCase(测试类)

unittest.TextTestRunner(verbosite =2 ).run(suite)

## 按测试模块执行

（如一个模块/py文件，有两个测试类）

suite =unittest.TestLoader ().loadTestsFromModule('模块名.py')

unittest.TextTestRunner(verbosite =2 ).run(suite)

## 优化测试套件

（把测试套件写到一个方法里）

P76

分离测试固件P77：把setUp和tearDown模块化，使其他模块不用每次单独写一遍

class InitTest(unittest.testcase):

    def setUp(self):

        浏览器初始化

    def tearDown(self):

        self.driver.quit()

from init import InitTest

class 测试类 (InitTest):

    def test\_XXX(self):

测试跳过：

@unittest.skip("I don't want to run this case.")

skip装饰器一共有三个

unittest.skip(reason) # skip无条件跳过

unittest.skipIf(condition, reason) # skipIf当condition为True时跳过

unittest.skipUnless(condition, reason) # skipUnless当condition为False时跳过

# 测试断言：

TestCase类中assert方法

assertEqual(a,b )

断言应该避免 if 和 try-except，防止结果为pass

批量执行测试用例：

如有两个测试文件，新建allTest.py：

def allTest():  
    suite= unittest.TestLoader().discover(

    start\_dir=os.path.dirname(\_\_file\_\_) , #测试模块路径

    pattern='test\_\*.py',   #获取所有以test开头的py文件

    top\_level\_dir=None #在调用的时候直接给默认值None  
 )

# 生成测试报告：

HTMLTestRunner.py放在lib下

with open('HTMLReport.html', 'w') as f:  
 runner = HTMLTestRunner(stream=f,  
 title='MathFunc Test Report',  
 description='generated by HTMLTestRunner.',  
 verbosity=2  
 )  
runner.run(suite)

# Coverage.py监控代码覆盖率

（pip安装）

coverage3 run allTest.py