# Unittest集成测试框架

## 一．Unittest简介

unittest的设计灵感最初来源于Junit以及其他语言中具有共同特征的单元框架。它支持自动化测试，在测试中使用setup（初始化）和teardown（关闭销毁）操作，组织测试用例为套件（批量运行），以及把测试和报告独立开来。python自带，非第三方库。

## 二．Unittest组成

unittest中最核心的四个概念是：**test case（测试用例）, test suite（测试套件）, test runner(测试运行), test fixture(测试夹具)。以下分别解释：**

1. **test fixture**

test fixture表示测试运行前需要做的准备工作以及结束后的清理工作。比如创建临时数据库、目录或启动一个服务器进程。包括测试前准备环境的搭建(setUp)，及测试后环境的还原(tearDown)两个方法。

1. test case

test case是单元测试中的最小个体。它检查特定输入的响应信息。unittest提供了一个基础类：TestCase，用来创建test case。

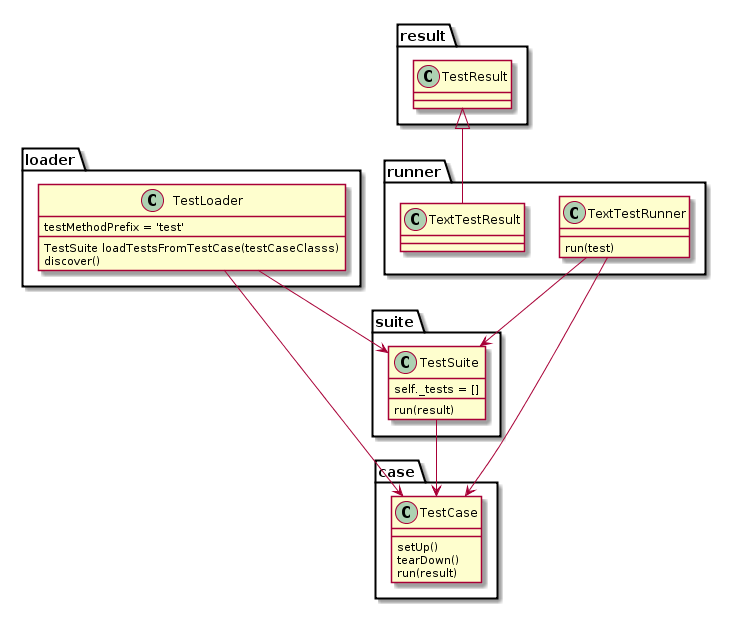
1. test suite

test suite是test case的合集，通常用test suite将test case汇总然后一起执行。

1. test runner

test runner是一个执行器，它可以执行case并提供结果给用户。它可以提供HTML图形界面、文本或者返回一个值表示测试结果。

## 三．Unittest执行流程



解释说明：

1. TestTextRunner做为程序执行的入口，是用来执行测试用例的，其中的run(test)会执行TestSuite/TestCase中的方法。
2. TextTestResult做为程序执行的出口，是用来保存TestTextRunner执行测试用例的结果，包括运行了多少测试用例，成功了多少，失败了多少等信息。
3. TextCase就是一个完整的测试用例，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)。
4. TestSuite是多个TextCase的集合。
5. TestLoader是用来加载TestCase到TestSuite中的。其中包含loadTestsFromName('模块名.TestCase名'), loadTestsFromNames(['模块名.TestCase名'])以及loadTestsFromTestCase(‘TestCase名’)

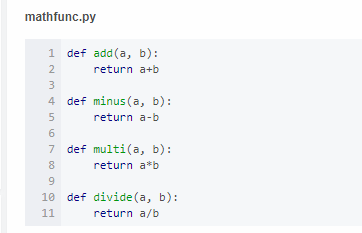
注意：1. TestLoader的方法加载的是类中的所有测试用例，并非单个用例。

2. 在写测试用例时，我们可以不写TestLoader，直接添加用例。这样的好处：是可以根据需求添加指定的用例。弊端：一个一个添加比较麻烦，不如使用TestLoader一次性全部将用例加载进suite中快捷。

3. TestLoader不需要创建该类的实例; 该unittest模块提供了一个可以共享的实例unittest.defaultTestLoader。但是，使用子类或实例可以自定义一些可配置的属性。

## 四. Unittest简单示例

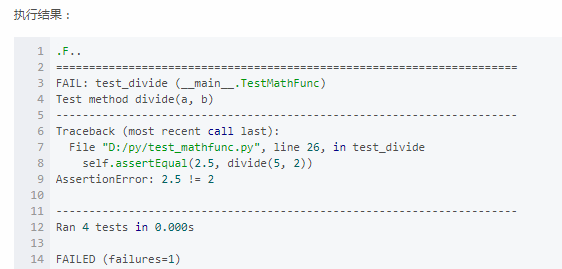
**文件mathfunc.py**



1. 简单Testcase测试用例：

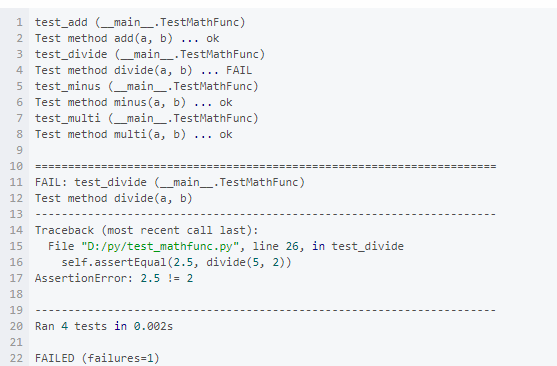


结果：



返回结果说明：

1. 在第一行给出了每一个用例执行的结果的标识，成功是 .，失败是 F，出错是 E，跳过是 S。从上面也可以看出，测试的执行跟方法的顺序没有关系，test\_divide写在了第4个，但是却是第2个执行的。
2. 每个测试方法均以 test 开头，否则是不被unittest识别的。
3. 在unittest.main(verbosity=2)中 verbosity 参数可以控制输出的错误报告的详细程度，默认是 1，如果设为 0，则不输出每一用例的执行结果，即没有上面的结果中的第1行；如果设为 2，则输出详细的执行结果，如下：

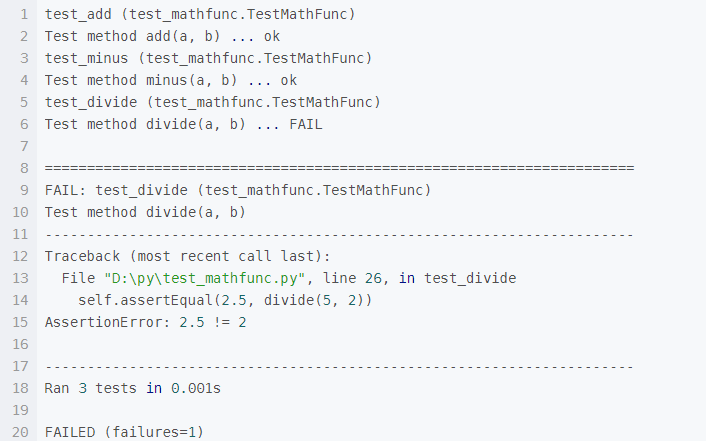


2.组织TestSuite

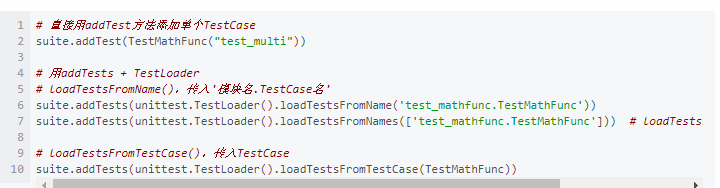
**添加到TestSuite中的case是会按照添加的顺序执行的**。



结果：



上面用了TestSuite的 addTests() 方法，并直接传入了TestCase列表，我们还可以：



注意：TestLoader的方法是无法对case进行排序的；

3.（1）将结果输出到文件中



执行此文件，可以看到，在同目录下生成了**UnittestTextReport.txt**，所有的执行报告均输出到了此文件中， txt格式的测试报告就会产生。

备注：文件写入的内容与控制台输出的内容一致

3.（2）将结果输出到HTML页面中

HTMLTestRunner是一个第三方的unittest HTML报告库，首先我们下载HTMLTestRunner.py，并放到当前目录下，或者你的’C:\Python27\Lib’下，就可以导入运行了**。**

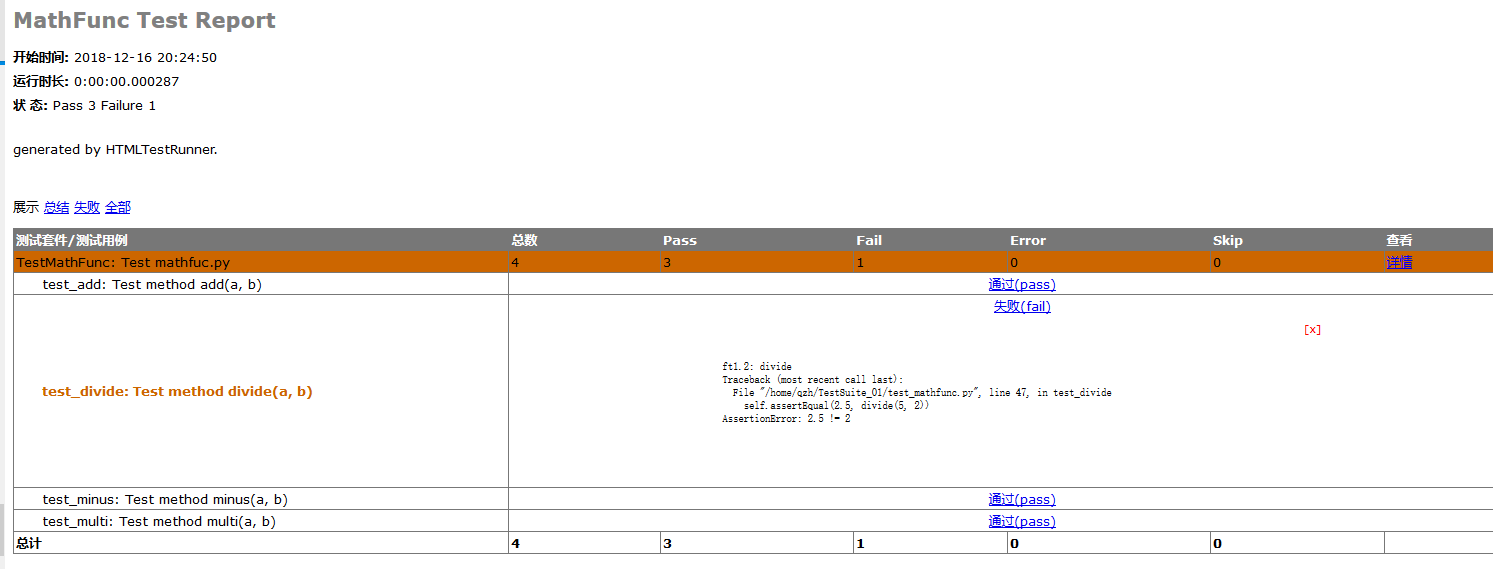
**案例：**



结果：



生成HTMLReport.html报告：



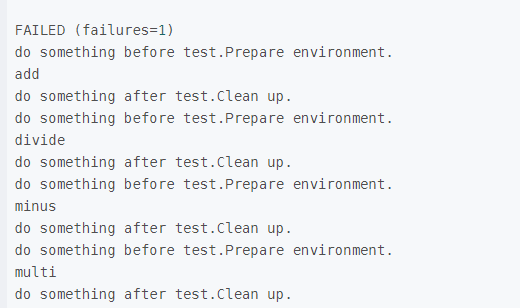
4.test fixture之setUp() tearDown()

比如每个testcase在执行之前以及结束之后需要初始化环境，需要使用setUp(),teardown()方法。

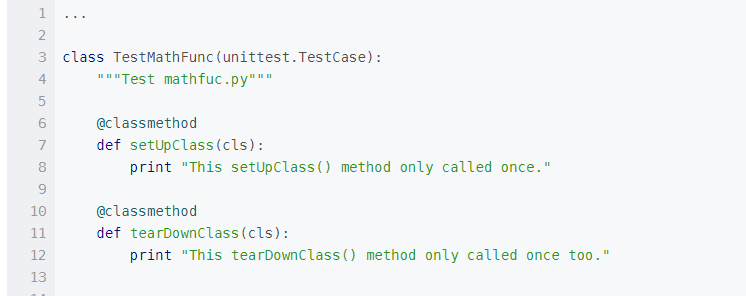
例如：



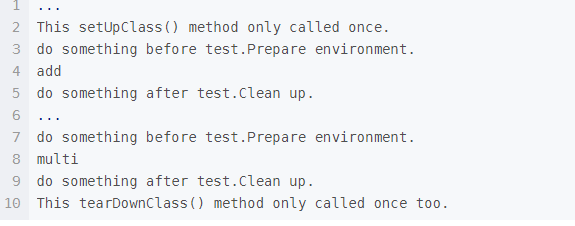
结果：



可以看到setUp和tearDown在每次执行case前后都执行了一次。如果想要在所有case执行之前准备一次环境，并在所有case执行结束之后再清理环境，我们可以用 setUpClass() 与 tearDownClass():



结果：



可以看到setUpClass以及tearDownClass均只执行了一次。

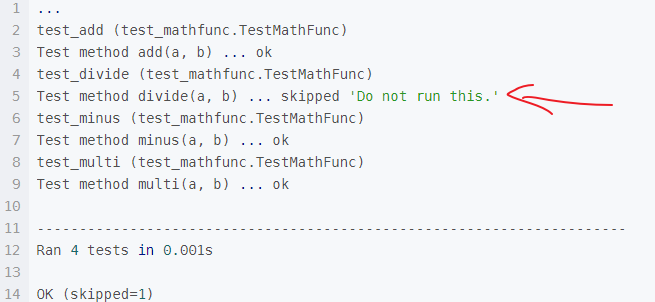
1. 跳过某个Case

skip装饰器一共有三个 unittest.skip(reason)、unittest.skipIf(condition, reason)、unittest.skipUnless(condition, reason)，skip无条件跳过，skipIf当condition为True时跳过，skipUnless当condition为False时跳过。

案例：

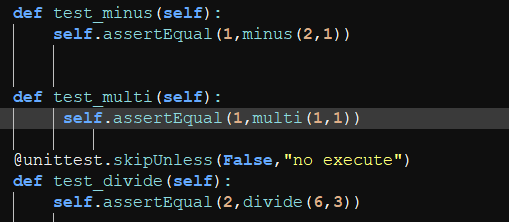


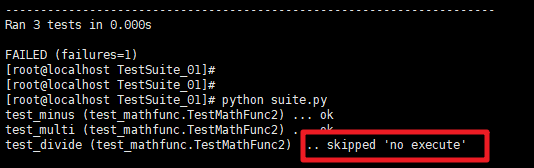
结果：



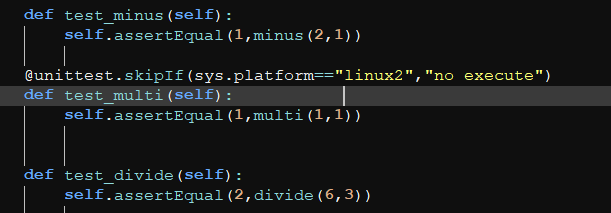
可以看到总的test数量还是4个，但divide()方法被skip了。

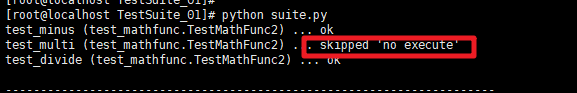
unittest.skipUnless(condition, reason)案例：



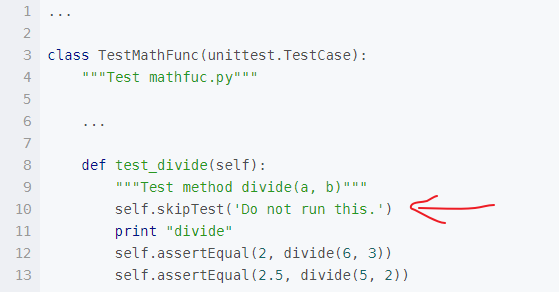


unittest.skipIf(condition, reason)案例：

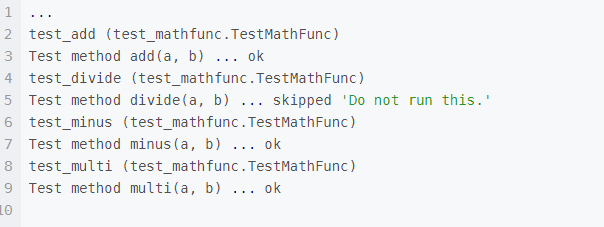




除了以上三个方法可以跳过case外，也可以用TestCase.skipTest()方法。如下图：



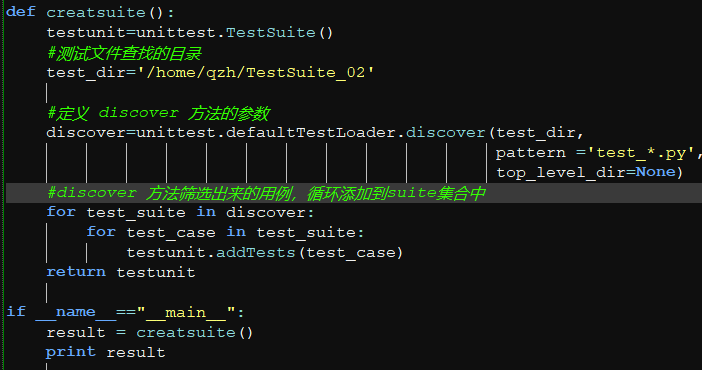
结果：



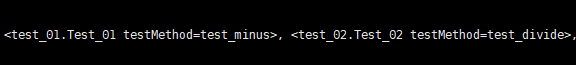
效果跟上面的装饰器一样，跳过了divide方法。

1. unittest.TestLoader.discover方法匹配目录下的用例

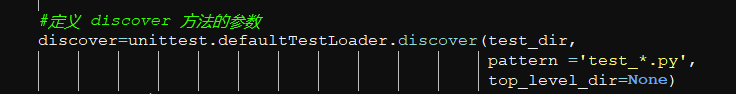
假如现在目录下存在两个测试用例，Test\_01.py和Test\_02.py，如果用addTests的方法添加用例到suite集合中，有些麻烦，这时候可以采取TestLoader()这个类。



结果：



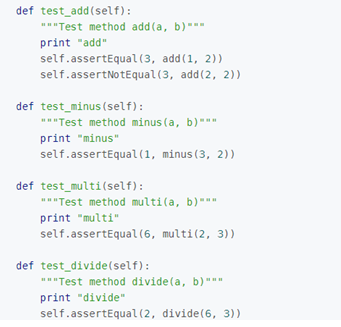
**注意：**



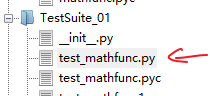
discover方法,test\_dir是要寻找的目录路径，pattern是查找条件，即在指定的目录下查找以"test"开头的测试用例，top\_level\_dir是项目的顶级目录（默认为开始目录）(事实上，这个查找是不区分大小写的，即Test开头的也能找到)

**五．书写规范注意**

1. 每个TestCase中的测试方法均以 test 开头，否则是不被unittest识别的。

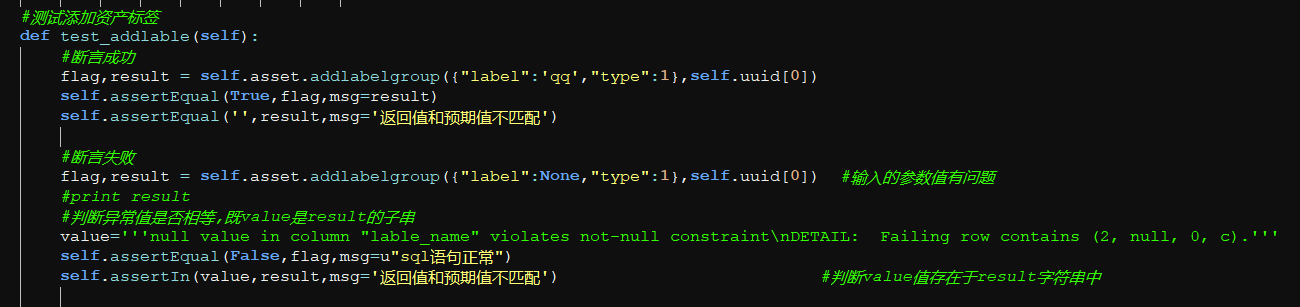


1. 测试文件命名：test\_描述.py



1. 提示：

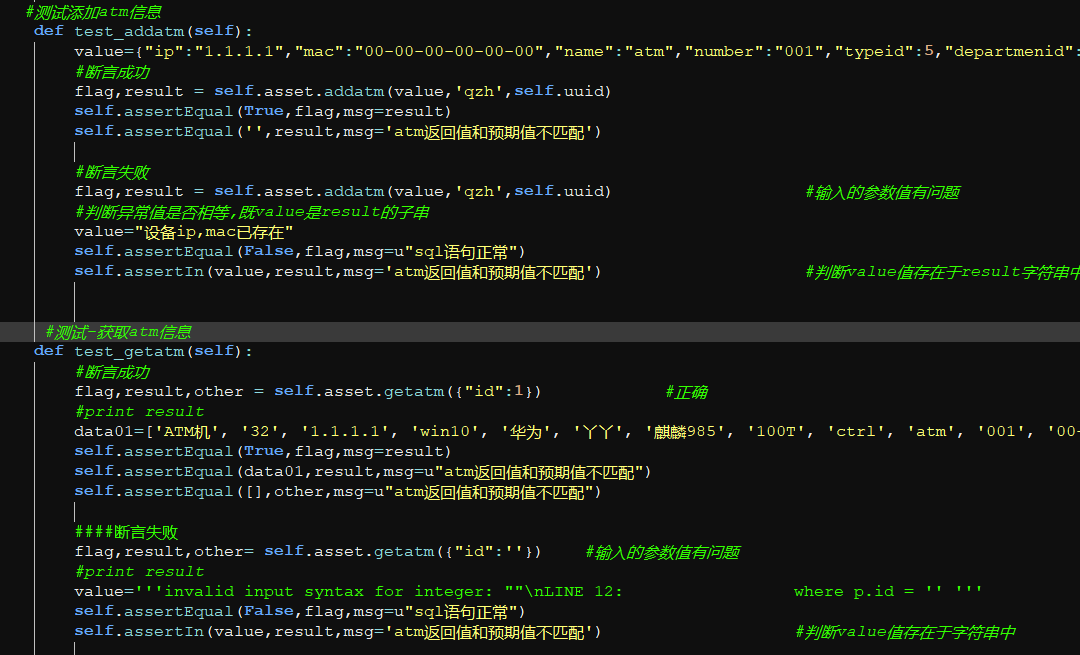
我们写的测试用例针对一个接口，要有正确的断言，以及发生错误的异常断言。



1. 书写的测试用例文件一定要自闭合

例如：当我们写了一个添加接口的测试用例文件，这时，我们还要写一个查询测试用例，以确保添加接口确实把数据添加进去了。

如下图所示：



**六．常用的TestCase断言**



**七．命令行模式**

(1).执行test\_mathfunc.py 文件的所有测试用例：

python –m unittest test\_mathfunc

(2). 执行test\_mathfunc.py文件的TestMathFunc类的所有测试用例：

python –m unittest test\_mathfunc. TestMathFunc

(3). 执行test\_mathfunc.py文件的TestMathFunc类的test\_add用例：

Python -m unittest test\_mathfunc.TestMathFunc.test\_add

**八．Coverage覆盖率**

（1） Coverage.py 是一个用来测试代码覆盖率的 Python 第三方库, coverage.py 库支持 Python 2.6 或者更高的版本，还兼容 Python 3 的最新版.

安装coverage: pip install coverage

（2）命令：coverage run + 待测试文件名称 (运行测试文件，同时在当前目录生成.coverage文件)

coverage report (读取.coverage文件中的内容，并输出到控制台)

coverage html (读取.coverage文件中的内容，生成html报告)

备注：特别声明针对coverage html命令常用的几个参数 ：

-d +目录 (指定生成的html文件存储在指定目录下) ，

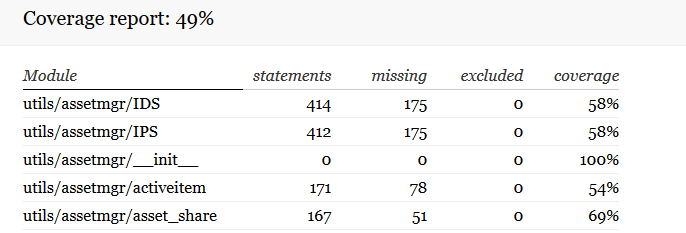
--include=指定包含的文件或某目录下的所有文件，

--omit=指定排除的文件或排除目录下的文件。

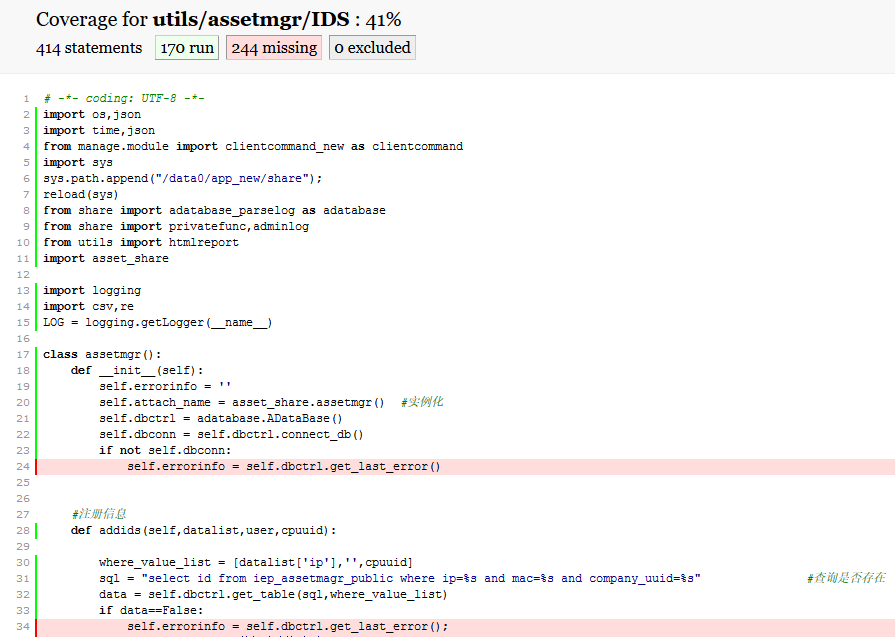
案例如下： (1) coverage run suite文件名

(2) coverage html --include=utils/assetmgr/\* -d /root/htmcov/

（3）示例报告：



具体信息：



解释说明：



Statements：文件内容的总行数

Missing： 错过的行数

excluded：不包括在内的行数

coverage：单文件覆盖率

备注：coverage 必须查找源码py文件，不可以是pyc文件。

官方文档链接：https://cloud.tencent.com/developer/section/1367592#stage-100049367