**目录**

[一、 什么是单元测试? 3](#_Toc16761)

[二、 为什么要单元测试？ 3](#_Toc31517)

[1、前言 3](#_Toc31192)

[2、现状 4](#_Toc14821)

[3、发现 4](#_Toc18292)

[4、剖析 5](#_Toc28244)

[5、重要性 7](#_Toc8444)

[6、应用 8](#_Toc4440)

[三、什么是PHPunit? 8](#_Toc27835)

[四、 怎么学习PHPunit? 9](#_Toc17627)

[1、安装xdebug 9](#_Toc28309)

[2、安装PHPunit 10](#_Toc18505)

[3、 编写第一个测试 12](#_Toc12018)

[1) 先创建一个项目 12](#_Toc30295)

[2) 写测试用例 13](#_Toc27133)

[3) 执行测试用例 14](#_Toc15802)

[4) 编写生产代码 16](#_Toc20963)

[5) 测试生产的代码 16](#_Toc21755)

[6) 完成类的自动加载 19](#_Toc1008)

[7) 数据供给器 19](#_Toc10633)

[4、 用xml配置编排测试套件 21](#_Toc20419)

[5、 代码覆盖率 23](#_Toc19638)

# 什么是单元测试?

单元测试是对单独的代码对象进行测试的过程，比如对函数、类、方法进行测试。单元测试可以使用任意一段已经写好的测试代码，也可以使用一些已经存在的测试框架，比如JUnit、PHPUnit或者Cantata++，单元测试框架提供了一系列共同、有用的功能来帮助人们编写自动化的检测单元，例如检查一个实际的值是否符合我们期望的值的断言。单元测试框架经常会包含每个测试的报告，以及给出你已经覆盖到的代码覆盖率。

# 为什么要单元测试？

## 1、前言

最近，在网上看到过一个调查，调查的内容是“程序员在项目开发中编写单元测试的情况”。当然，至于调查的结果，我想聪明的你已经可以猜到了。高达 58.3% 的比例，一般情况下不写单元测试，只有偶尔的情况才会写写。16.6% 的程序员从来都不写单元测试。只有很少的一部分程序员才会在自己的代码中进行单元测试，并保证方法测试通过。看到这些，你想到了什么？

## 2、现状

虽然，这个调查可能会有些片面性，但这也基本反应了国内程序员的开发现状，很少有程序员能够比较认真的去编写单元测试。而且，甚至有的程序员根本就不知道为什么要写单元测试（这一点让我很郁闷）。他们经常会说，公司里不是有测试人员嘛，测试应该是他们要做的事，我们的工作只是开发（这位仁兄肯定没有学过软件工程）。当然，这些并不是偶然的，正如佛经里边说的“因果循环”，有果必有因。那么，到底是什么原因，导致程序员对单元测试这么不感冒呢？

## 3、发现

通过与几个朋友的讨论，以及网上的调查，主要有这几种原因，导致程序员对单元测试很排斥，或许说很不以为意。

* 不知道怎么编写单元测试
* 项目没有要求，所以不编写
* 单元测试价值不高，完全是浪费时间
* 业务逻辑比较简单，不值得编写单元测试
* 不管怎样，集成测试将会抓住所有的 bug，用不着进行单元测试
* 在项目的前期还是尽量去编写单元测试，但是越到项目的后期就越失控
* 为了完成编码任务，没有足够的时间编写单元测试。编写单元测试会导致不能按时完成编码任务，导致项目延期

很显然，这几种原因归根结底，无外乎就是不了解单元测试，自认为很聪明，自己懒不想去测试，对项目的时间、进度把控不好。下面，我将一 一进行分析，剖析出程序员的开发心理，以此来给朋友们提个醒，最终聪明反被聪明误。

## 4、剖析

不知道怎么编写单元测试

这个问题在于，还没有接触过单元测试，同时，也没有体会过企业级的代码开发。不知道同时也不了解单元测试能带给你什么。设想一下，当你开发完一个功能模块的时候，你如何确定你的模块没有 bug 呢？如果涉及到具体的业务，你会执行 debug 模式，然后一点一点的深入到代码中去查看吗？如果你一直都是这样，那么你早就已经 OUT 了。赶快去了解一下单元测试的工具吧，你会收获很大的。

项目没有要求，所以不编写

这个问题反映出了一种现象，同时也是一种习惯。项目有没有要求，只能说明项目的管理上不严格，并不是程序员不编写单元测试的理由。他们在以往的开发中，并没有养成写单元测试的好习惯。可想而知，他们的代码质量，我就不敢恭维了。给个建议，尝试着写漂亮的代码，之所以因为漂亮，是指得健康、简洁、健壮。当然，完成漂亮的代码就离不开单元测试了。

单元测试价值不高，完全是浪费时间

这种说法其实是错误的。为什么这么说呢？在日常的开发中，代码的完工其实并不等于开发的完工。如果没有单元测试，那么如何保证代码能够正常运行呢？测试人员做的只是业务上的集成测试，也就是黑盒测试，对单个的方法是没有办法测试的，而且，测试出的 bug 的范围也会很广，根本不能确定 bug 的范围，还得去花时间来确定 bug 出在什么地方。难道这就不浪费时间了吗？甚至，这样的方式，时间浪费的会更多。

业务逻辑比较简单，不值得编写单元测试

所谓的业务逻辑比较简单，其实是相对的。当你对某一块业务逻辑很熟悉的时候，你自然会认为它很简单。然而，单元测试的必要性并不是仅仅在于测试代码的功能是否正确，还在于，当其他同事在了解你的业务的时候，能够很快的通过单元测试来熟悉代码的功能，甚至不用去读代码，就能够知道它做了哪些事情。因此，写单元测试不仅是解放了自己，更方便了别人。

项目前期还在尽量写测试，到了后期就失控了

这种问题的原因在于，对项目进度、项目中的技术点研究时间、人员的沟通、业务需求的熟悉程度等没有把控好。这个问题的出现并不是个人的问题，而是反映了项目管理中存在的问题。当然，个人的原因也存在，就是如何在有限的时间里，提高效率。这一点需要大家好好思考一下了。我的建议，多做计划，根据实际情况变更计划。多和项目组长、组成员进行沟通。及时反应项目中存在的问题。

为了完成编码任务，没有足够的时间编写单元测试

这个问题在于，程序员领取的任务较为复杂，或者自己的开发效率有待提高。其实，开发任务是包括编码和单元测试的。在领任务的时候，应该跟据自身的能力，跟组长或经理沟通好，以便留出一定的测试时间。当然，提高自己的编码效率也是很有必要的。至于如何提高开发效率，网上有很多这样的文章，这里就不再赘述了。

## 5、重要性

测试常常是程序员十分厌倦的一个活动。测试能给我们带来什么？了解这些是非常重要的，测试不可能保证一个程序是完全正确的，但是测试却可以增强我们对程序完整的信心，测试可以让我们相信程序做了我么期望它做的事情。测试能够使我们尽早的发现程序的 bug 和不足。

一个 bug 被隐藏的时间越长，修复这个 bug 的代价就越大。在《快速软件开发》一书中已引用了大量的研究数据指出：最后才修改一个 bug 的代价是在 bug 产生时修改它的代价的10倍。

当然，我们主要讨论的是单元测试。单元测试是一个方法层面上的测试，也是最细粒度的测试。用于测试一个类的每一个方法都已经满足了方法的功能要求。在开发中，对于自己开发的模块，只有在通过单元测试之后，才能提交到 SVN 库 或者 Git 库。

## 6、应用

正是由于测试在开发中的重要地位，才会在IT界刮起了 TDD 的旋风。TDD，也就是测试驱动开发模式。它旨在强调在开发功能代码之前，先编写测试代码。也就是说在明确要开发某个功能后，首先思考如何对这个功能进行测试，并完成测试代码的编写，然后编写相关的代码满足这些测试用例。然后循环进行添加其他功能，直到完成全部功能的开发。

# 三、什么是PHPunit?

PHPUnit是一个用PHP编程语言开发的开源软件，是一个单元测试框架。PHPUnit由Sebastian Bergmann创建，源于Kent Beck的SUnit，是xUnit家族的框架之一。本文将探索PHPUnit，特别介绍自动化单元测试的基本用法。你需要掌握PHP编程语言的基本知识才能继续。

作者：Kendrick Curtis，Stainless Software

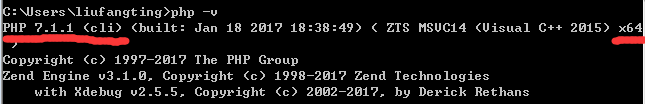
# 怎么学习PHPunit?

## 1、安装xdebug

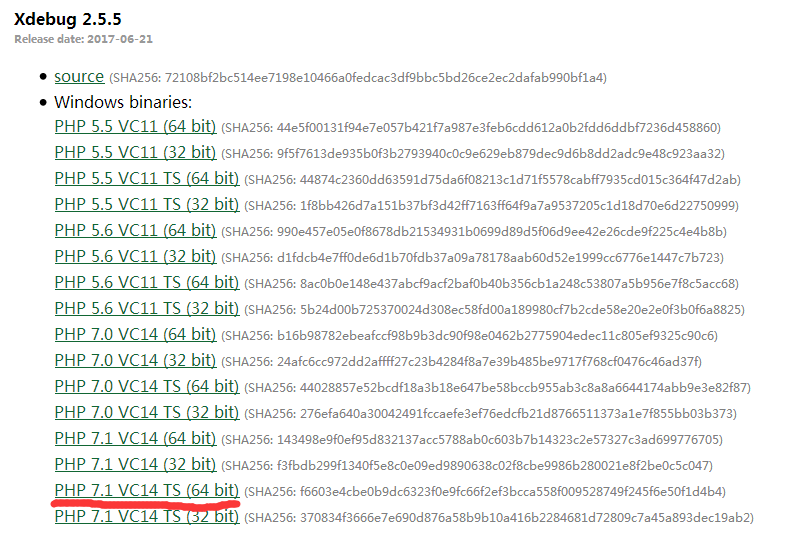
1）下载地址：

<https://xdebug.org/download.php>

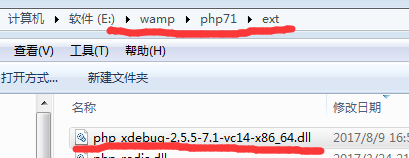
下载与PHP版本对应的xdebug版本,我的PHP版本是PHP7.1 x64



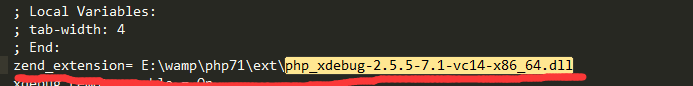
所以我这里选择下载 [PHP 7.1 VC14 TS (64 bit)](https://xdebug.org/files/php_xdebug-2.5.5-7.1-vc14-x86_64.dll)



2）下载完成之后解压，将php\_xdebug-2.5.5-7.1-vc14-x86\_64.dll，放到 php/ext 里面，我的路径是 E:\wamp\php71\ext 。



3）然后配置下php.ini，添加 php\_xdebug-2.5.5-7.1-vc14-x86\_64.dll



1. 重启Apache，运行cmd程序，在窗口中输入 php -m 查看xdebug 是否 安装成功

## 2、安装PHPunit

**Linux安装**

$ wget <https://phar.phpunit.de/phpunit-6.2.phar>

$ chmod +x phpunit-6.2.phar

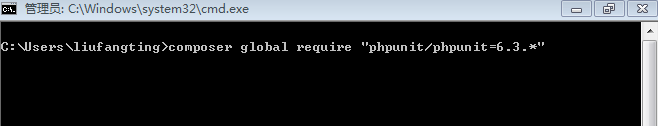
$ sudo mv phpunit-6.2.phar /usr/local/bin/phpunit

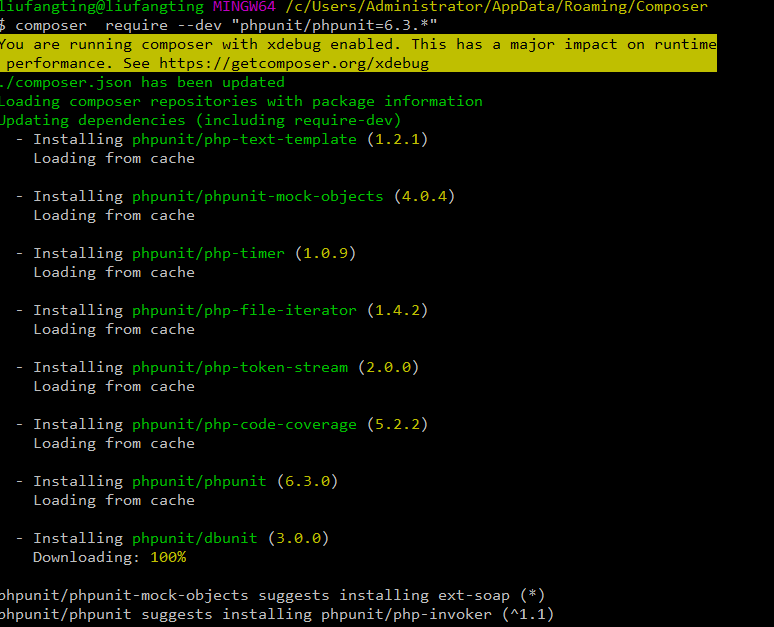
$ phpunit --version

**Window安装**

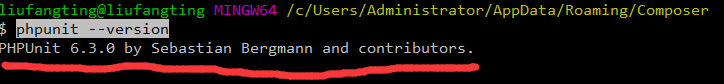
composer global require "phpunit/phpunit=6.3.\*"

打开cmd程序





安装完成后,在窗口输入phpunit --version 查看phpunit 是否安装成功。



## 编写第一个测试

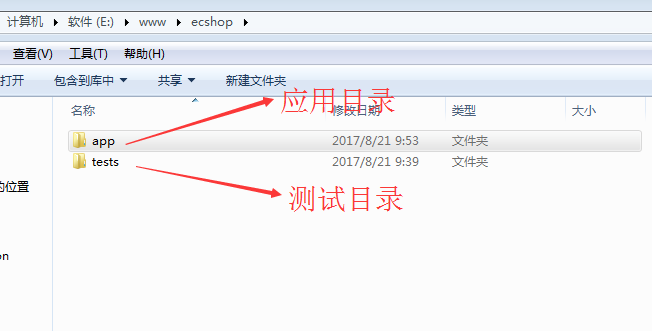
**PHPUnit 编写测试的基本惯例与步骤**

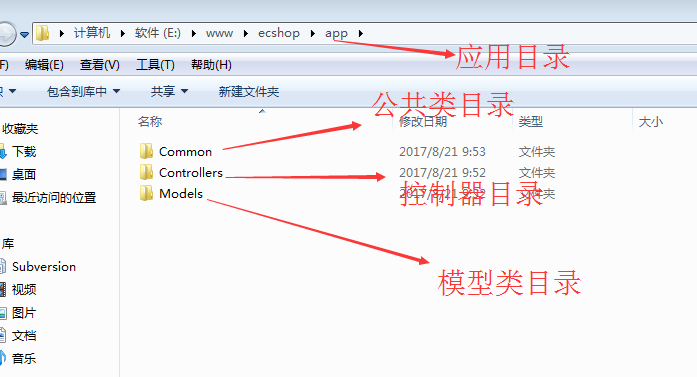
1. 针对类 Class 的测试写在类 ClassTest中。
2. ClassTest（通常）继承自 PHPUnit\Framework\TestCase。
3. 测试都是命名为 test\* 的公用方法。也可以在方法的文档注释块(docblock)中使用 @test 标注将其标记为测试方法
4. 在测试方法内，类似于 assertEquals()这样的断言方法用来对实际值与预期值的匹配做出断言,需要查看其它断言的方法可以查看这个链接地址：

<https://phpunit.de/manual/current/zh_cn/appendixes.assertions.html>

### 1) 先创建一个项目

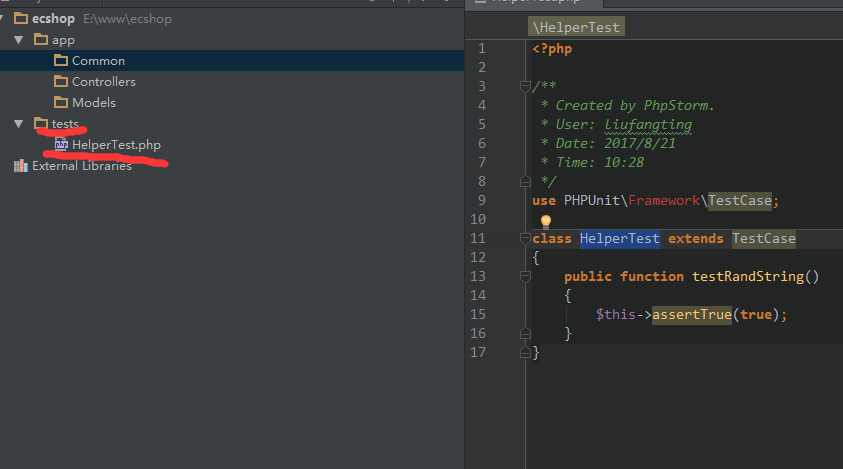
目录说明





### 2) 写测试用例

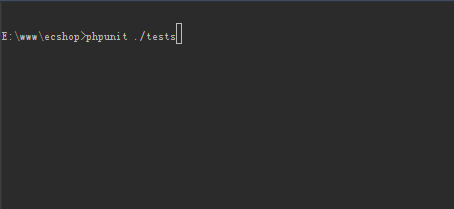
现在咱们在tests目录里面写一个测试类,名字叫HelperTest,在这个HelperTest类里面定义了一个testRandString方法



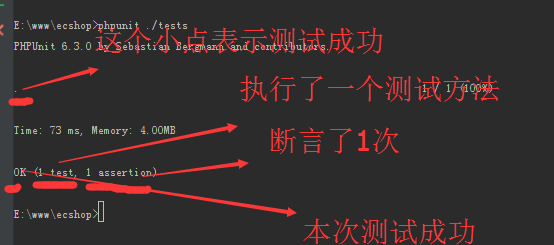


### 3) 执行测试用例

打开Terminal,用**PHPUnit 命令行测试执行器来执**行咱们刚刚写的测试用例



运行的结果:



对于每个测试的运行，PHPUnit 命令行工具输出一个字符来指示进展，比如说上面截图里面的小点 . 表示测试成功时候的进展。

. 当测试成功时输出。

F 当测试方法运行过程中一个断言失败时输出。

E 当测试方法运行过程中产生一个错误时输出。

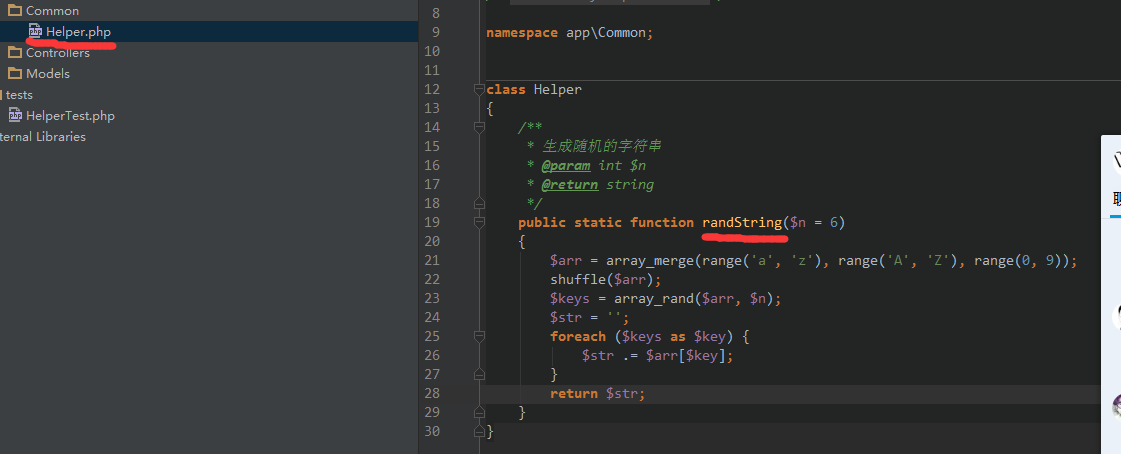
R 当测试被标记为有风险时输出。

S 当测试被跳过时输出。

I 当测试被标记为不完整或未实现时输出。

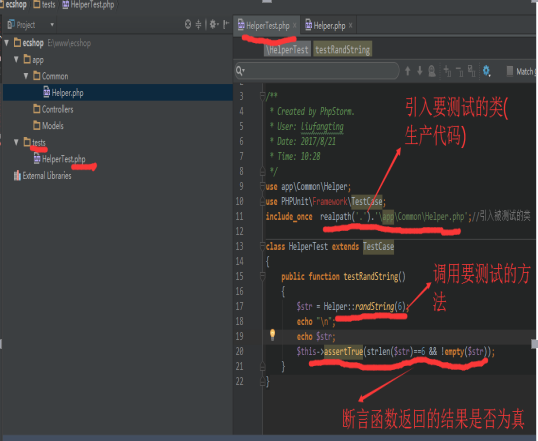
### 4) 编写生产代码

刚才在HelperTest类里面写了testRandString() 这个方法, 主要是想测试一个能随机字符串生成的方法 randString(),现在就在Common目录下定义一个Helper 类，并定义一个randString()方法

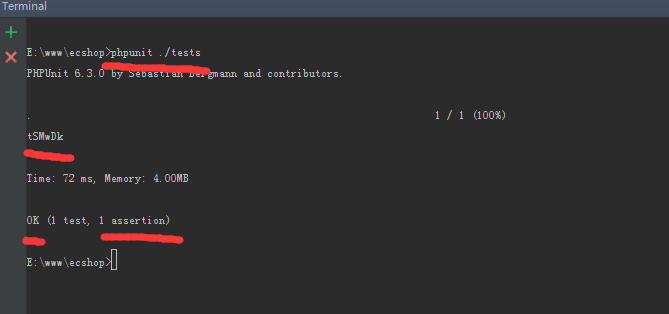


### 5) 测试生产的代码

在HelperTest.php 文件中引入Helper.php文件,然后执行HelperTest.php文件

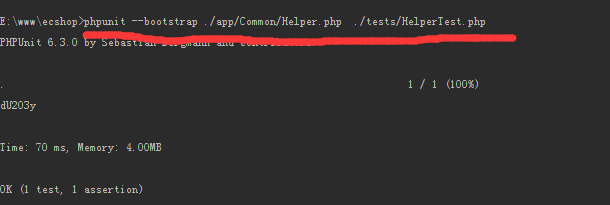


执行的结果如下:



思考一个问题，在HelperTest.php 文件中，每次调用app中的方法的时候都要include，这样会不会很麻烦。可以这么干

phpunit —bootstrap [源文件] 测试文件

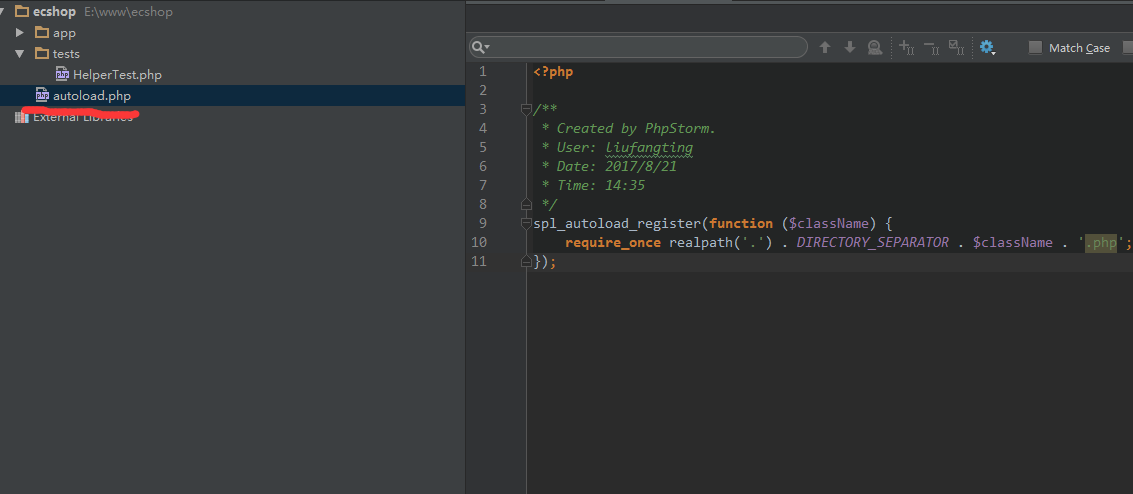


但是有个问题,如果要HelperTest.php 要引入多个类,这就比较尴尬了。

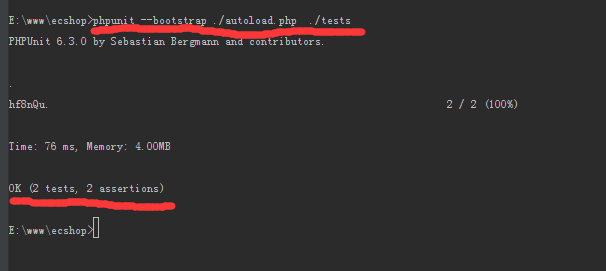


所以需要引入某各类的时候，希望它能自动加载,下面就看看如何完成自动加载。

### 6) 完成类的自动加载



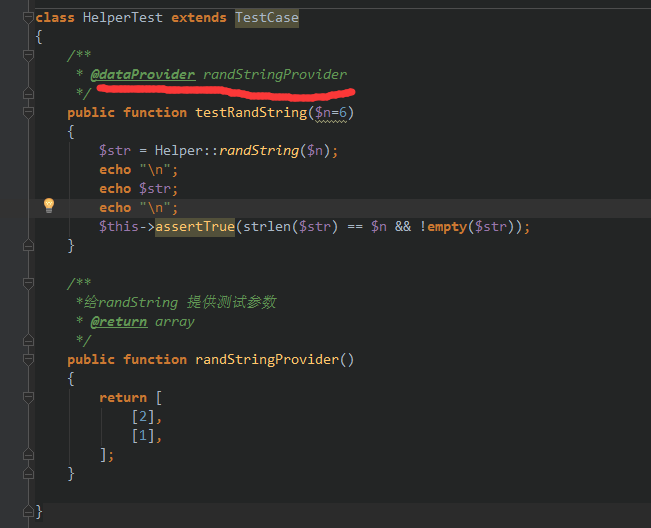
测试结果



### 7) 数据供给器

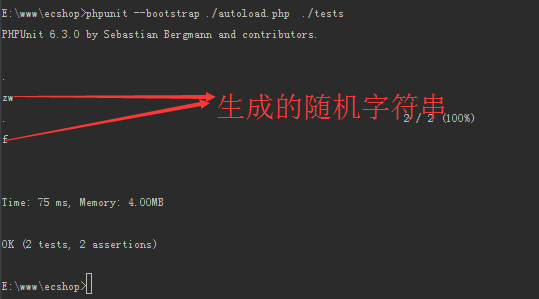
**定义:**

**数据供给器给要测试的方法提供数据支持**



用 @dataProvider 标注来指定使用哪个数据供给器方法

测试结果

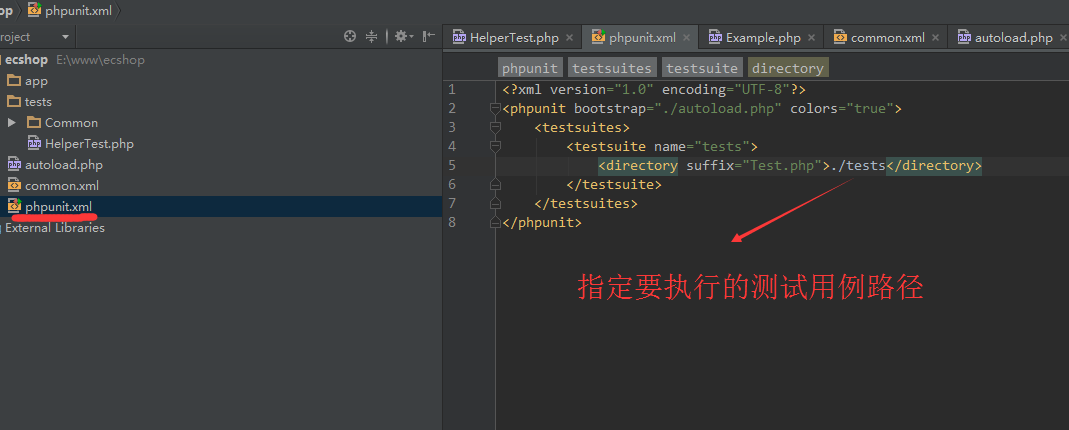


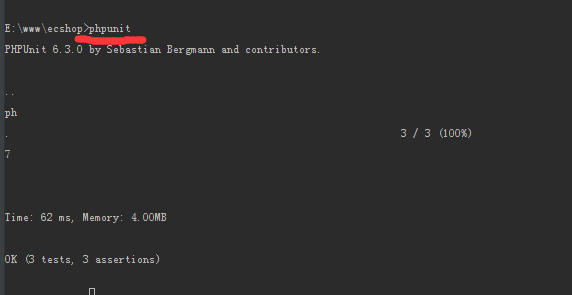
## 用xml配置编排测试套件

之前运行测试用例是这么执行的phpunit --bootstrap ./autoload.php ./tests .

现在想直接输入phpunit (不加任何参数)来执行测试用例

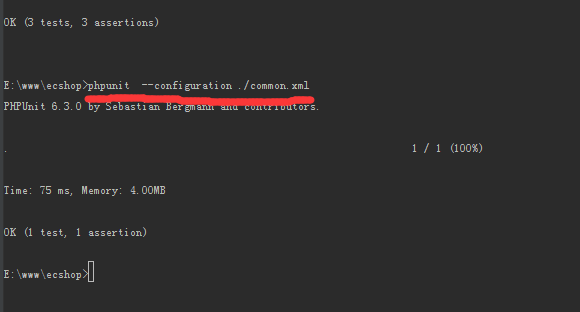
Xml文件如下图所示





**注意**：**如果你直接用 phpunit 命令不加 --configuration 来指定对于的xml文件,phpunit 默认去当前路径查找有没有叫phpunit.xml配置文件！！！**

如果需要指定xml配置文件 --configuration 后面跟上xml路径即可,看下图

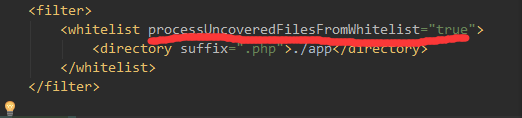


## 代码覆盖率

定义：

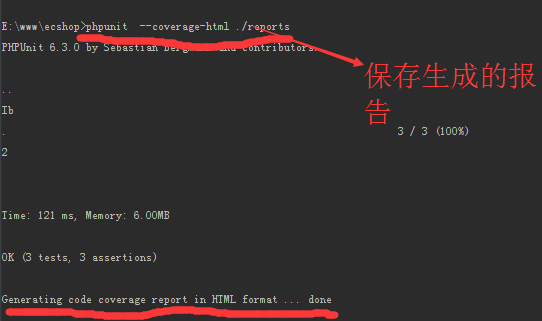
计算机科学中所说的代码覆盖率是一种用于衡量特定测试套件对程序源代码测试程度的指标。拥有高代码覆盖率的程序相较于低代码低概率的程序而言测试的更加彻底、包含软件 bug 的可能性更低

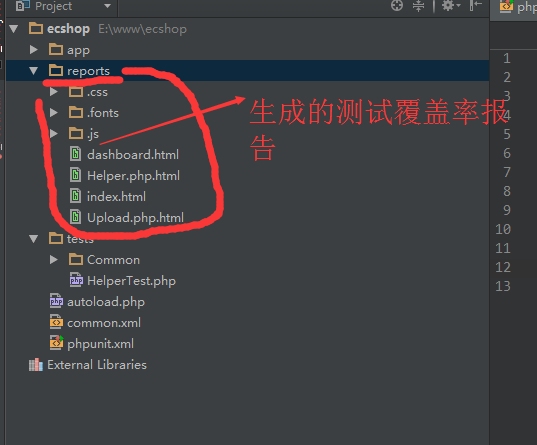
注意：生成测试报告，为了告诉 PHPUnit 哪些源代码文件要包含在代码覆盖率报告中，必须配置白名单，可以在 PHPUnit 的配置信息中设置 addUncoveredFilesFromWhitelist="true" 来将白名单中包含的所有文件全部加入到代码覆盖率报告中



下面开始生成测试报告

phpunit --coverage-html ./reports





浏览测试报告，效果如下

