PHPUnit 手册

Sebastian Bergmann

PHPUnit 手册

Sebastian Bergmann

出版日期 此版本对应于 PHPUnit 6.5。最后更新于 2017-12-07。

目录

表格清单

范例清单

第 1 章 安装 PHPUnit

需求

PHPUnit 6.5 需要 PHP 7，强烈推荐使用最新版本的 PHP。

PHPUnit 需要使用 dom [http://php.net/manual/en/dom.setup.php] 和 json [http://php.net/manual/en/json.installation.php] 扩展，它们通常是默认启用的。

PHPUnit 还需要 pcre [http://php.net/manual/en/pcre.installation.php]、reflection [http://php.net/manual/en/reflection.installation.php]、spl [http://php.net/manual/en/spl.installation.php] 扩展。这些标准扩展默认启用，并且除非修改 PHP 的构建系统和 C 源代码，否则无法禁用它们。

代码覆盖率分析报告功能需要 Xdebug [http://xdebug.org/]（2.5.0以上）与 tokenizer [http://php.net/manual/en/tokenizer.installation.php] 扩展。生成 XML 格式的报告需要有 xmlwriter [http://php.net/manual/en/xmlwriter.installation.php] 扩展。

PHP 档案包 (PHAR)

要获取 PHPUnit，最简单的方法是下载 PHPUnit 的 PHP 档案包 (PHAR) [http://php.net/phar]，它将 PHPUnit 所需要的所有必要组件（以及某些可选组件）捆绑在单个文件中：

要使用 PHP档案包(PHAR)需要有 phar [http://php.net/manual/en/phar.installation.php] 扩展。

如果启用了 Suhosin [http://suhosin.org/] 扩展，需要在 `php.ini` 中允许执行 PHAR：

```
suhosin.executor.include.whitelist = phar
```

如果要全局安装 PHAR：

```
$ 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
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

也可以直接使用下载的 PHAR 文件：

```
$ wget https://phar.phpunit.de/phpunit-6.2.phar
$ php phpunit-6.2.phar --version
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

Windows

整体上说，在 Windows 下安装 PHAR 和手工在 Windows 下安装 Composer [https://getcomposer.org/doc/00-intro.md#installation-windows] 是一样的过程：

1. 为 PHP 的二进制可执行文件建立一个目录，例如 `C:\bin`
2. 将 **`;C:\bin`** 附加到 PATH 环境变量中（相关帮助 [http://stackoverflow.com/questions/6318156/adding-python-path-on-windows-7]）
3. 下载 https://phar.phpunit.de/phpunit-6.2.phar 并将文件保存到 `C:\bin\phpunit.phar`
4. 打开命令行（例如，按 **Windows**+**R** » 输入 **cmd** » **ENTER**）
5. 建立外包覆批处理脚本（最后得到 `C:\bin\phpunit.cmd`）：

```
C:\Users\username> cd C:\bin
C:\bin> echo @php "%~dp0phpunit.phar" %* > phpunit.cmd
C:\bin> exit
```

6. 新开一个命令行窗口，确认一下可以在任意路径下执行 PHPUnit:

```
C:\Users\username> phpunit --version
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

对于 Cygwin 或 MingW32 (例如 TortoiseGit) shell 环境，可以跳过第五步。 取而代之的是，把文件保存为 `phpunit`（没有 `.phar` 扩展名），然后用 **`chmod 775 phpunit`** 将其设为可执行。

校验 PHPUnit PHAR 发行包

由 PHPUnit 项目分发的所有官方代码发行包都由发行包管理器进行签名。在 phar.phpunit.de [https://phar.phpunit.de/] 上有 PGP 签名和 SHA1 散列值可用于校验。

下面的例子详细说明了如何对发行包进行校验。首先下载 `phpunit.phar` 和与之对应的单独 PGP 签名 `phpunit.phar.asc`:

```
wget https://phar.phpunit.de/phpunit.phar
wget https://phar.phpunit.de/phpunit.phar.asc
```

用单独的签名(`phpunit.phar`)对 PHPUnit 的 PHP 档案包(`phpunit.phar.asc`)进行校验:

```
gpg phpunit.phar.asc
gpg: Signature made Sat 19 Jul 2014 01:28:02 PM CEST using RSA key ID 6372C20A
gpg: Can't check signature: public key not found
```

在本地系统中没有发行包管理器的公钥(6372C20A)。为了能进行校验，必须从某个密钥服务器上取得发行包管理器的公钥。其中一个服务器是 `pgp.uni-mainz.de`。所有密钥服务器是链接在一起的，因此连接到任一密钥服务器都可以。

```
gpg --keyserver pgp.uni-mainz.de --recv-keys 0x4AA394086372C20A
gpg: requesting key 6372C20A from hkp server pgp.uni-mainz.de
gpg: key 6372C20A: public key "Sebastian Bergmann <sb@sebastian-bergmann.de>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

现在已经取得了条目名称为"Sebastian Bergmann <sb@sebastian-bergmann.de>"的公钥。不过无法检验这个密钥确实是由名叫 Sebastian Bergmann 的人创建的。但是可以先试着校验发行包的签名:

```
gpg phpunit.phar.asc
gpg: Signature made Sat 19 Jul 2014 01:28:02 PM CEST using RSA key ID 6372C20A
gpg: Good signature from "Sebastian Bergmann <sb@sebastian-bergmann.de>"
gpg:                 aka "Sebastian Bergmann <sebastian@php.net>"
gpg:                 aka "Sebastian Bergmann <sebastian@thephp.cc>"
gpg:                 aka "Sebastian Bergmann <sebastian@phpunit.de>"
gpg:                 aka "Sebastian Bergmann <sebastian.bergmann@thephp.cc>"
gpg:                 aka "[jpeg image of size 40635]"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: D840 6D0D 8294 7747 2937  7831 4AA3 9408 6372 C20A
```

此时，签名已经没问题了，但是这个公钥还不能信任。签名没问题意味着文件未被篡改。可是由于公钥加密系统的性质，还需要再校验密钥 6372C20A 确实是由真正的 Sebastian Bergmann 创建的。

任何攻击者都能创建公钥并将其上传到公钥服务器。他们可以建立一个带恶意的发行包，并用这个假密钥进行签名。这样，如果尝试对这个损坏了的发行包进行签名校验，由于密钥是“真”密钥，校验将成功完成。因此，需要对这个密钥的真实性进行校验。如何对公钥的真实性进行校验已经超出了本文档的范畴。

有个比较谨慎的做法是创建一个脚本来管理 PHPUnit 的安装，在运行测试套件之前校验 GnuPG 签名。例如：

```
#!/usr/bin/env bash
clean=1 # 是否在测试完成之后删除 phpunit.phar ?
aftercmd="php phpunit.phar --bootstrap bootstrap.php src/tests"
gpg --fingerprint D8406D0D82947747293778314AA394086372C20A
if [ $? -ne 0 ]; then
    echo -e "\033[33mDownloading PGP Public Key...\033[0m"
    gpg --recv-keys D8406D0D82947747293778314AA394086372C20A
    # Sebastian Bergmann <sb@sebastian-bergmann.de>
    gpg --fingerprint D8406D0D82947747293778314AA394086372C20A
    if [ $? -ne 0 ]; then
        echo -e "\033[31mCould not download PGP public key for verification\033[0m"
        exit
    fi
fi

if [ "$clean" -eq 1 ]; then
    # 如果存在就清理掉
    if [ -f phpunit.phar ]; then
        rm -f phpunit.phar
    fi
    if [ -f phpunit.phar.asc ]; then
        rm -f phpunit.phar.asc
    fi
fi

# 抓取最新的发行版和对应的签名
if [ ! -f phpunit.phar ]; then
    wget https://phar.phpunit.de/phpunit.phar
fi
if [ ! -f phpunit.phar.asc ]; then
    wget https://phar.phpunit.de/phpunit.phar.asc
fi

# 在运行前先校验
gpg --verify phpunit.phar.asc phpunit.phar
if [ $? -eq 0 ]; then
    echo
    echo -e "\033[33mBegin Unit Testing\033[0m"
    # 运行测试套件
    `$after_cmd`
    # 清理
    if [ "$clean" -eq 1 ]; then
        echo -e "\033[32mCleaning Up!\033[0m"
        rm -f phpunit.phar
        rm -f phpunit.phar.asc
    fi
else
    echo
    chmod -x phpunit.phar
    mv phpunit.phar /tmp/bad-phpunit.phar
    mv phpunit.phar.asc /tmp/bad-phpunit.phar.asc
    echo -e "\033[31mSignature did not match! PHPUnit has been moved to /tmp/bad-phpunit
    exit 1
fi
```

Composer

如果用 Composer [https://getcomposer.org/] 来管理项目的依赖关系，只要在项目的 `composer.json` 文件中简单地加上对 `phpunit/phpunit` 的依赖关系即可：

```
composer require --dev phpunit/phpunit ^6.2
```

可选的组件包

有以下可选组件包可用：

`PHP_Invoker`

一个工具类，可以用带有超时限制的方式调用可调用内容。当需要在严格模式下保证测试的超时限制时，这个组件包是必须的。

PHPUnit 的 PHAR 分发中已经包含了此组件包。可以用以下命令来经由 Composer 安装此组件包：

```
composer require --dev phpunit/php-invoker
```

`DbUnit`

移植到 PHP/PHPUnit 上的 DbUnit 用于提供对数据库交互测试的支持。

PHPUnit 的 PHAR 分发中已经包含了此组件包。可以用以下命令来经由 Composer 安装此组件包：

```
composer require --dev phpunit/dbunit
```

第 2 章 编写 PHPUnit 测试

例 2.1 “用 PHPUnit 测试数组操作”展示了如何用 PHPUnit 编写测试来对 PHP 数组操作进行测试。本例介绍了用 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()`（参见 附录 A, 断言）这样的断言方法用来对实际值与预期值的匹配做出断言。

例 2.1. 用 PHPUnit 测试数组操作

```
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    public function testPushAndPop()
    {
        $stack = [];
        $this->assertEquals(0, count($stack));

        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertEquals(1, count($stack));

        $this->assertEquals('foo', array_pop($stack));
        $this->assertEquals(0, count($stack));
    }
}
?>
```

> 当你想把一些东西写到 `print` 语句或者调试表达式中时，别这么做，将其写成一个测试来代替。
>
> —Martin Fowler

测试的依赖关系

> 单元测试主要是作为一种良好实践来编写的，它能帮助开发人员识别并修复 bug、重构代码，还可以看作被测软件单元的文档。要实现这些好处，理想的单元测试应当覆盖程序中所有可能的路径。一个单元测试通常覆盖一个函数或方法中的一个特定路径。但是，测试方法并不一定非要是一个封装良好的独立实体。测试方法之间经常有隐含的依赖关系暗藏在测试的实现方案中。
>
> —Adrian Kuhn et. al.

PHPUnit支持对测试方法之间的显式依赖关系进行声明。这种依赖关系并不是定义在测试方法的执行顺序中，而是允许生产者(producer)返回一个测试基境(fixture)的实例，并将此实例传递给依赖于它的消费者(consumer)们。

- 生产者(producer)，是能生成被测单元并将其作为返回值的测试方法。
- 消费者(consumer)，是依赖于一个或多个生产者及其返回值的测试方法。

例 2.2 “用 @depends 标注来表达依赖关系”展示了如何用 @depends 标注来表达测试方法之间的依赖关系。

例 2.2. 用 @depends 标注来表达依赖关系

```
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    public function testEmpty()
    {
        $stack = [];
        $this->assertEmpty($stack);

        return $stack;
    }

    /**
     * @depends testEmpty
     */
    public function testPush(array $stack)
    {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);

        return $stack;
    }

    /**
     * @depends testPush
     */
    public function testPop(array $stack)
    {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
?>
```

在上例中，第一个测试， testEmpty()，创建了一个新数组，并断言其为空。随后，此测试将此基境作为结果返回。第二个测试，testPush()，依赖于 testEmpty()，并将所依赖的测试之结果作为参数传入。最后，testPop() 依赖于 testPush()。

注意

默认情况下，生产者所产生的返回值将“原样”传递给相应的消费者。这意味着，如果生产者返回的是一个对象，那么传递给消费者的将是一个指向此对象的引用。如果需要传递对象的副本而非引用，则应当用 @depends clone 替代 @depends。

为了快速定位缺陷，我们希望把注意力集中于相关的失败测试上。这就是为什么当某个测试所依赖的测试失败时，PHPUnit 会跳过这个测试。通过利用测试之间的依赖关系，缺陷定位得到了改进，如例 2.3 “利用测试之间的依赖关系”中所示。

例 2.3. 利用测试之间的依赖关系

```
<?php
use PHPUnit\Framework\TestCase;

class DependencyFailureTest extends TestCase
{
    public function testOne()
```

```
    {
        $this->assertTrue(false);
    }

    /**
     * @depends testOne
     */
    public function testTwo()
    {
    }
}
?>
```

```
phpunit --verbose DependencyFailureTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

FS

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) DependencyFailureTest::testOne
Failed asserting that false is true.

/home/sb/DependencyFailureTest.php:6

There was 1 skipped test:

1) DependencyFailureTest::testTwo
This test depends on "DependencyFailureTest::testOne" to pass.


FAILURES!
Tests: 1, Assertions: 1, Failures: 1, Skipped: 1.
```

测试可以使用多个 @depends 标注。PHPUnit 不会更改测试的运行顺序，因此你需要自行保证某个测试所依赖的所有测试均出现于这个测试之前。

拥有多个 @depends 标注的测试，其第一个参数是第一个生产者提供的基境，第二个参数是第二个生产者提供的基境，以此类推。参见例 2.4 “有多重依赖的测试”

例 2.4. 有多重依赖的测试

```
<?php
use PHPUnit\Framework\TestCase;

class MultipleDependenciesTest extends TestCase
{
    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

    public function testProducerSecond()
    {
        $this->assertTrue(true);
        return 'second';
    }

    /**
     * @depends testProducerFirst
```

```
     * @depends testProducerSecond
     */
    public function testConsumer()
    {
        $this->assertEquals(
            ['first', 'second'],
            func_get_args()
        );
    }
}
?>
```

```
phpunit --verbose MultipleDependenciesTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

...

Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 3 assertions)
```

数据供给器

测试方法可以接受任意参数。这些参数由数据供给器方法（在 例 2.5 “使用返回数组的数组的数据供给器”中，是 additionProvider() 方法）提供。用 @dataProvider 标注来指定使用哪个数据供给器方法。

数据供给器方法必须声明为 public，其返回值要么是一个数组，其每个元素也是数组；要么是一个实现了 Iterator 接口的对象，在对它进行迭代时每步产生一个数组。每个数组都是测试数据集的一部分，将以它的内容作为参数来调用测试方法。

例 2.5. 使用返回数组的数组的数据供给器

```
<?php
use PHPUnit\Framework\TestCase;

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return [
            [0, 0, 0],
            [0, 1, 1],
            [1, 0, 1],
            [1, 1, 3]
        ];
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

...F
```

```
Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 (1, 1, 3)
Failed asserting that 2 matches expected 3.

/home/sb/DataTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

当使用到大量数据集时，最好逐个用字符串键名对其命名，避免用默认的数字键名。这样输出信息会更加详细些，其中将包含打断测试的数据集所对应的名称。

例 2.6. 使用带有命名数据集的数据供给器

```php
<?php
use PHPUnit\Framework\TestCase;

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return [
            'adding zeros'  => [0, 0, 0],
            'zero plus one' => [0, 1, 1],
            'one plus zero' => [1, 0, 1],
            'one plus one'  => [1, 1, 3]
        ];
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set "one plus one" (1, 1, 3)
Failed asserting that 2 matches expected 3.

/home/sb/DataTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

例 2.7. 使用返回迭代器对象的数据供给器

```php
<?php
```

```
use PHPUnit\Framework\TestCase;

require 'CsvFileIterator.php';

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertEquals($expected, $a + $b);
    }

    public function additionProvider()
    {
        return new CsvFileIterator('data.csv');
    }
}
?>
```

```
phpunit DataTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 ('1', '1', '3')
Failed asserting that 2 matches expected '3'.

/home/sb/DataTest.php:11

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

例 2.8. CsvFileIterator 类

```
<?php
use PHPUnit\Framework\TestCase;

class CsvFileIterator implements Iterator {
    protected $file;
    protected $key = 0;
    protected $current;

    public function __construct($file) {
        $this->file = fopen($file, 'r');
    }

    public function __destruct() {
        fclose($this->file);
    }

    public function rewind() {
        rewind($this->file);
        $this->current = fgetcsv($this->file);
        $this->key = 0;
    }

    public function valid() {
```

```
        return !feof($this->file);
    }

    public function key() {
        return $this->key;
    }

    public function current() {
        return $this->current;
    }

    public function next() {
        $this->current = fgetcsv($this->file);
        $this->key++;
    }
}
?>
```

如果测试同时从 @dataProvider 方法和一个或多个 @depends 测试接收数据，那么来自于数据供给器的参数将先于来自所依赖的测试的。来自于所依赖的测试的参数对于每个数据集都是一样的。参见例 2.9 “在同一个测试中组合使用 @depends 和 @dataProvider”

例 2.9. 在同一个测试中组合使用 @depends 和 @dataProvider

```
<?php
use PHPUnit\Framework\TestCase;

class DependencyAndDataProviderComboTest extends TestCase
{
    public function provider()
    {
        return [['provider1'], ['provider2']];
    }

    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

    public function testProducerSecond()
    {
        $this->assertTrue(true);
        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     * @dataProvider provider
     */
    public function testConsumer()
    {
        $this->assertEquals(
            ['provider1', 'first', 'second'],
            func_get_args()
        );
    }
}
?>
```

```
phpunit --verbose DependencyAndDataProviderComboTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.
```

```
...F

Time: 0 seconds, Memory: 3.50Mb

There was 1 failure:

1) DependencyAndDataProviderComboTest::testConsumer with data set #1 ('provider2')
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
Array (
-    0 => 'provider1'
+    0 => 'provider2'
1 => 'first'
2 => 'second'
)

/home/sb/DependencyAndDataProviderComboTest.php:31

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
```

注意

如果一个测试依赖于另外一个使用了数据供给器的测试，仅当被依赖的测试至少能在一组数据上成功时，依赖于它的测试才会运行。使用了数据供给器的测试，其运行结果是无法注入到依赖于此测试的其他测试中的。

注意

所有的数据供给器方法的执行都是在对 `setUpBeforeClass` 静态方法的调用和第一次对 `setUp` 方法的调用之前完成的。因此，无法在数据供给器中使用创建于这两个方法内的变量。这是必须的，这样 PHPUnit 才能计算测试的总数量。

对异常进行测试

例 2.10 “使用 expectException() 方法”展示了如何用 `@expectException` 标注来测试被测代码中是否抛出了异常。

例 2.10. 使用 expectException() 方法

```php
<?php
use PHPUnit\Framework\TestCase;

class ExceptionTest extends TestCase
{
    public function testException()
    {
        $this->expectException(InvalidArgumentException::class);
    }
}
?>
```

```
phpunit ExceptionTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb
```

```
There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

除了 expectException() 方法外，还有 expectExceptionCode()、expectExceptionMessage() 和 expectExceptionMessageRegExp() 方法可以用于为被测代码所抛出的异常建立预期。

或者，也可以用 @expectedException、@expectedExceptionCode、@expectedExceptionMessage 和 @expectedExceptionMessageRegExp 标注来为被测代码所抛出的异常建立预期。例 2.11 “使用 @expectedException 标注”展示了一个范例。

例 2.11. 使用 @expectedException 标注

```php
<?php
use PHPUnit\Framework\TestCase;

class ExceptionTest extends TestCase
{
    /**
     * @expectedException InvalidArgumentException
     */
    public function testException()
    {
    }
}
?>
```

```
phpunit ExceptionTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

对 PHP 错误进行测试

默认情况下，PHPUnit 将测试在执行中触发的 PHP 错误、警告、通知都转换为异常。利用这些异常，就可以，比如说，预期测试将触发 PHP 错误，如例 2.12 “用 @expectedException 来预期 PHP 错误”所示。

注意

PHP 的 error_reporting 运行时配置会对 PHPUnit 将哪些错误转换为异常有所限制。如果在这个特性上碰到问题，请确认 PHP 的配置中没有抑制想要测试的错误类型。

例 2.12. 用 @expectedException 来预期 PHP 错误

```
<?php
use PHPUnit\Framework\TestCase;

class ExpectedErrorTest extends TestCase
{
    /**
     * @expectedException PHPUnit\Framework\Error
     */
    public function testFailingInclude()
    {
        include 'not_existing_file.php';
    }
}
?>
```

```
phpunit -d error_reporting=2 ExpectedErrorTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.

Time: 0 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

PHPUnit\Framework\Error\Notice 和 PHPUnit\Framework\Error\Warning 分别代表 PHP 通知与 PHP 警告。

注意

对异常进行测试是越明确越好的。对太笼统的类进行测试有可能导致不良副作用。因此，不再允许用 @expectedException 或 setExpectedException() 对 Exception 类进行测试。

如果测试依靠会触发错误的 PHP 函数，例如 fopen ，有时候在测试中使用错误抑制符会很有用。通过抑制住错误通知，就能对返回值进行检查，否则错误通知将会导致抛出 PHPUnit\Framework\Error\Notice。

例 2.13. 对会引发PHP 错误的代码的返回值进行测试

```
<?php
use PHPUnit\Framework\TestCase;

class ErrorSuppressionTest extends TestCase
{
    public function testFileWriting() {
        $writer = new FileWriter;
        $this->assertFalse(@$writer->write('/is-not-writeable/file', 'stuff'));
    }
}
class FileWriter
{
    public function write($file, $content) {
        $file = fopen($file, 'w');
        if($file == false) {
            return false;
        }
        // ...
    }
}
```

```
?>
```

```
phpunit ErrorSuppressionTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

如果不使用错误抑制符，此测试将会失败，并报告 `fopen(/is-not-writeable/file): failed to open stream: No such file or directory`。

对输出进行测试

有时候，想要断言（比如说）某方法的运行过程中生成了预期的输出（例如，通过 `echo` 或 `print`）。`PHPUnit\Framework\TestCase` 类使用 PHP 的 输出缓冲 [http://www.php.net/manual/en/ref.outcontrol.php] 特性来为此提供必要的功能支持。

例 2.14 “对函数或方法的输出进行测试”展示了如何用 `expectOutputString()` 方法来设定所预期的输出。如果没有产生预期的输出，测试将计为失败。

例 2.14. 对函数或方法的输出进行测试

```
<?php
use PHPUnit\Framework\TestCase;

class OutputTest extends TestCase
{
    public function testExpectFooActualFoo()
    {
        $this->expectOutputString('foo');
        print 'foo';
    }

    public function testExpectBarActualBaz()
    {
        $this->expectOutputString('bar');
        print 'baz';
    }
}
?>
```

```
phpunit OutputTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) OutputTest::testExpectBarActualBaz
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
```

```
+'baz'


FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

表 2.1 “用于对输出进行测试的方法”中列举了用于对输出进行测试的各种方法。

表 2.1. 用于对输出进行测试的方法

方法	含义
`void expectOutputRegex(string $regularExpression)`	设置输出预期为输出应当匹配正则表达式 `$regularExpression`。
`void expectOutputString(string $expectedString)`	设置输出预期为输出应当与 `$expectedString` 字符串相等。
`bool setOutputCallback(callable $callback)`	设置回调函数，用来做诸如将实际输出规范化之类的动作。
`string getActualOutput()`	获取实际输出。

注意

在严格模式下，本身产生输出的测试将会失败。

错误相关信息的输出

当有测试失败时，PHPUnit 全力提供尽可能多的有助于找出问题所在的上下文信息。

例 2.15. 数组比较失败时生成的错误相关信息输出

```php
<?php
use PHPUnit\Framework\TestCase;

class ArrayDiffTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [1, 2,  3, 4, 5, 6],
            [1, 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit ArrayDiffTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArrayDiffTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
```

```
     0 => 1
     1 => 2
-    2 => 3
+    2 => 33
     3 => 4
     4 => 5
     5 => 6
 )

/home/sb/ArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

在这个例子中，数组中只有一个值不同，但其他值也都同时显示出来，以提供关于错误发生的位置的上下文信息。

当生成的输出很长而难以阅读时，PHPUnit 将对其进行分割，并在每个差异附近提供少数几行上下文信息。

例 2.16. 长数组比较失败时生成的错误相关信息输出

```php
<?php
use PHPUnit\Framework\TestCase;

class LongArrayDiffTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2,  3, 4, 5, 6],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit LongArrayDiffTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) LongArrayDiffTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
     13 => 2
-    14 => 3
+    14 => 33
     15 => 4
     16 => 5
     17 => 6
 )


/home/sb/LongArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

边缘情况

当比较失败时，PHPUnit 为输入值建立文本表示，然后以此进行对比。这种实现导致在差异指示中显示出来的问题可能比实际上存在的多。

这种情况只出现在对数组或者对象使用 assertEquals 或其他“弱”比较函数时。

例 2.17. 当使用弱比较时在生成的差异结果中出现的边缘情况

```php
<?php
use PHPUnit\Framework\TestCase;

class ArrayWeakComparisonTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [1, 2, 3, 4, 5, 6],
            ['1', 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
phpunit ArrayWeakComparisonTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArrayWeakComparisonTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
-    0 => 1
+    0 => '1'
     1 => 2
-    2 => 3
+    2 => 33
     3 => 4
     4 => 5
     5 => 6
 )


/home/sb/ArrayWeakComparisonTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

在这个例子中，第一个索引项中的 1 and '1' 在报告中被视为不同，虽然 assertEquals 认为这两个值是匹配的。

第 3 章 命令行测试执行器

PHPUnit 命令行测试执行器可通过 phpunit 命令调用。下面的代码展示了如何用 PHPUnit 命令行测试执行器来运行测试：

```
phpunit ArrayTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

..

Time: 0 seconds


OK (2 tests, 2 assertions)
```

上面这个调用例子中，PHPUnit 命令行测试执行器将在当前工作目录中寻找 ArrayTest.php 源文件并加载之。而在此源文件中应当能找到 ArrayTest 测试用例类，此类中的测试将被执行。

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：

. 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（参见第 6 章 有风险的测试）。

S 当测试被跳过时输出（参见第 7 章 未完成的测试与跳过的测试）。

I 当测试被标记为不完整或未实现时输出（参见第 7 章 未完成的测试与跳过的测试）。

PHPUnit 区分 败*(failure)*与错误*(error)*。失败指的是被违背了的 PHPUnit 断言，例如一个失败的 assertEquals() 调用。错误指的是意料之外的异常(exception)或 PHP 错误。这种差异已被证明在某些时候是非常有用的，因为错误往往比失败更容易修复。如果得到了一个非常长的问题列表，那么最好先对付错误，当错误全部修复了之后再试一次瞧瞧还有没有失败。

命令行选项

让我们来瞧瞧以下代码中命令行测试运行器的各种选项：

```
phpunit --help
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

Usage: phpunit [options] UnitTest [UnitTest.php]
       phpunit [options] <directory>

Code Coverage Options:

  --coverage-clover <file>    Generate code coverage report in Clover XML format.
  --coverage-crap4j <file>    Generate code coverage report in Crap4J XML format.
  --coverage-html <dir>       Generate code coverage report in HTML format.
  --coverage-php <file>       Export PHP_CodeCoverage object to file.
  --coverage-text=<file>      Generate code coverage report in text format.
                              Default: Standard output.
  --coverage-xml <dir>        Generate code coverage report in PHPUnit XML format.
  --whitelist <dir>           Whitelist <dir> for code coverage analysis.
  --disable-coverage-ignore   Disable annotations for ignoring code coverage.

Logging Options:
```

```
  --log-junit <file>          Log test execution in JUnit XML format to file.
  --log-teamcity <file>       Log test execution in TeamCity format to file.
  --testdox-html <file>       Write agile documentation in HTML format to file.
  --testdox-text <file>       Write agile documentation in Text format to file.
  --testdox-xml <file>        Write agile documentation in XML format to file.
  --reverse-list              Print defects in reverse order

Test Selection Options:

  --filter <pattern>          Filter which tests to run.
  --testsuite <name,...>      Filter which testsuite to run.
  --group ...                 Only runs tests from the specified group(s).
  --exclude-group ...         Exclude tests from the specified group(s).
  --list-groups               List available test groups.
  --list-suites               List available test suites.
  --test-suffix ...           Only search for test in files with specified
                              suffix(es). Default: Test.php,.phpt

Test Execution Options:

  --dont-report-useless-tests Do not report tests that do not test anything.
  --strict-coverage           Be strict about @covers annotation usage.
  --strict-global-state       Be strict about changes to global state
  --disallow-test-output      Be strict about output during tests.
  --disallow-resource-usage   Be strict about resource usage during small tests.
  --enforce-time-limit        Enforce time limit based on test size.
  --disallow-todo-tests       Disallow @todo-annotated tests.

  --process-isolation         Run each test in a separate PHP process.
  --globals-backup            Backup and restore $GLOBALS for each test.
  --static-backup             Backup and restore static attributes for each test.

  --colors=<flag>             Use colors in output ("never", "auto" or "always").
  --columns <n>               Number of columns to use for progress output.
  --columns max               Use maximum number of columns for progress output.
  --stderr                    Write to STDERR instead of STDOUT.
  --stop-on-error             Stop execution upon first error.
  --stop-on-failure           Stop execution upon first error or failure.
  --stop-on-warning           Stop execution upon first warning.
  --stop-on-risky             Stop execution upon first risky test.
  --stop-on-skipped           Stop execution upon first skipped test.
  --stop-on-incomplete        Stop execution upon first incomplete test.
  --fail-on-warning           Treat tests with warnings as failures.
  --fail-on-risky             Treat risky tests as failures.
  -v|--verbose                Output more verbose information.
  --debug                     Display debugging information.

  --loader <loader>           TestSuiteLoader implementation to use.
  --repeat <times>            Runs the test(s) repeatedly.
  --teamcity                  Report test execution progress in TeamCity format.
  --testdox                   Report test execution progress in TestDox format.
  --testdox-group             Only include tests from the specified group(s).
  --testdox-exclude-group     Exclude tests from the specified group(s).
  --printer <printer>         TestListener implementation to use.

Configuration Options:

  --bootstrap <file>          A "bootstrap" PHP file that is run before the tests.
  -c|--configuration <file>   Read configuration from XML file.
  --no-configuration          Ignore default configuration file (phpunit.xml).
  --no-coverage               Ignore code coverage configuration.
  --no-extensions             Do not load PHPUnit extensions.
  --include-path <path(s)>    Prepend PHP's include_path with given path(s).
```

```
  -d key[=value]              Sets a php.ini value.
  --generate-configuration    Generate configuration file with suggested settings.

Miscellaneous Options:

  -h|--help                   Prints this usage information.
  --version                   Prints the version and exits.
  --atleast-version <min>     Checks that version is greater than min and exits.
```

`phpunit UnitTest`	运行由 `UnitTest` 类提供的测试。这个类应当在 `UnitTest.php` 源文件中声明。 `UnitTest` 这个类必须满足以下二个条件之一：要么它继承自 `PHPUnit\Framework\TestCase`；要么它提供 `public static suite()` 方法，这个方法返回一个 `PHPUnit_Framework_Test` 对象，比如，一个 `PHPUnit_Framework_TestSuite` 类的实例。
`phpunit UnitTest UnitTest.php`	运行由 `UnitTest` 类提供的测试。这个类应当在指定的源文件中声明。
`--coverage-clover`	为运行的测试生成带有代码覆盖率信息的 XML 格式的日志文件。更多细节请参见第 13 章 *Logging*（日志记录）。 请注意，此功能仅当安装了 tokenizer 和 Xdebug 这两个 PHP 扩展后才可用。
`--coverage-crap4j`	生成 Crap4j 格式的代码覆盖率报告。更多细节请参见第 11 章 代码覆盖率分析。 请注意，此功能仅当安装了 tokenizer 和 Xdebug 这两个 PHP 扩展后才可用。
`--coverage-html`	生成 HTML 格式的代码覆盖率报告。更多细节请参见第 11 章 代码覆盖率分析。 请注意，此功能仅当安装了 tokenizer 和 Xdebug 这两个 PHP 扩展后才可用。
`--coverage-php`	生成一个序列化后的 PHP_CodeCoverage 对象，此对象含有代码覆盖率信息。 请注意，此功能仅当安装了 tokenizer 和 Xdebug 这两个 PHP 扩展后才可用。
`--coverage-text`	为运行的测试以人们可读的格式生成带有代码覆盖率信息的日志文件或命令行输出。更多细节请参见 第 13 章 *Logging*（日志记录）。 请注意，此功能仅当安装了 tokenizer 和 Xdebug 这两个 PHP 扩展后才可用。
`--log-junit`	为运行的测试生成 JUnit XML 格式的日志文件。更多细节请参见 第 13 章 *Logging*（日志记录）。
`--testdox-html` 和 `--testdox-text`	为运行的测试以 HTML 或纯文本格式生成敏捷文档。更多细节请参见 第 12 章 测试的其他用途。
`--filter`	只运行名称与给定模式匹配的测试。如果模式未闭合包裹于分隔符，PHPUnit 将用 / 分隔符对其进行闭合包裹。 测试名称将以以下格式之一进行匹配：

`TestNamespace\TestCaseClass::testMethod`	默认的测试名称格式等价于在测试方法内使用 `__METHOD__` 魔术常量。
`TestNamespace\TestCaseClass::testMethod with data set #0`	当测试拥有数据供给器时，数据的每轮迭代都会将其当前索引附加在默认测试名称结尾处。
`TestNamespace\TestCaseClass::testMethod with data set "my named data"`	当测试拥有使用命名数据集的数据供给器时，数据的每轮迭代都会将当前名称附加在默认测试名称结尾处。命名数据集的例子参见例 3.1 “命名数据集”。

例 3.1. 命名数据集

```
<?php
use PHPUnit\Framework\TestCase;

namespace TestNamespace;

class TestCaseClass extends Tes
{
    /**
     * @dataProvider provider
     */
    public function testMethod(
    {
        $this->assertTrue($data
    }

    public function provider()
    {
        return [
            'my named data' =>
            'my data'       =>
        ];
    }
}
?>
```

`/path/to/my/test.phpt`	对于 PHPT 测试，其测试名称是文件系统路径。

有效的过滤器模式例子参见例 3.2 “过滤器模式例子”。

例 3.2. 过滤器模式例子

- `--filter 'TestNamespace\\TestCaseClass::testMethod'`
- `--filter 'TestNamespace\\TestCaseClass'`

- `--filter TestNamespace`
- `--filter TestCaseClass`
- `--filter testMethod`
- `--filter '/::testMethod .*"my named data"/'`
- `--filter '/::testMethod .*#5$/'`
- `--filter '/::testMethod .*#(5|6|7)$/'`

在匹配数据供给器时有一些额外的快捷方式，参见例 3.3 “过滤器的快捷方式”。

例 3.3. 过滤器的快捷方式

- `--filter 'testMethod#2'`
- `--filter 'testMethod#2-4'`
- `--filter '#2'`
- `--filter '#2-4'`
- `--filter 'testMethod@my named data'`
- `--filter 'testMethod@my.*data'`
- `--filter '@my named data'`
- `--filter '@my.*data'`

`--testsuite` 只运行名称与给定模式匹配的测试套件。

`--group` 只运行来自指定分组（可以多个）的测试。可以用 `@group` 标注为测试标记其所属的分组。

`@author` 标注是 `@group` 的一个别名，允许按作者来筛选测试。

`--exclude-group` 排除来自指定分组（可以多个）的测试。可以用 `@group` 标注为测试标记其所属的分组。

`--list-groups` 列出所有有效的测试分组。

`--test-suffix` 只查找文件名以指定后缀（可以多个）结尾的测试文件。

`--report-useless-tests` 更严格对待事实上不测试任何内容的测试。详情参见 第 6 章 有风险的测试。

`--strict-coverage` 更严格对待意外的代码覆盖。详情参见 第 6 章 有风险的测试。

`--strict-global-state` 更严格对待全局状态篡改。详情参见 第 6 章 有风险的测试。

`--disallow-test-output` 更严格对待测试执行期间产生的输出。详情参见第 6 章 有风险的测试。

--disallow-todo-tests	不执行文档注释块中含有 @todo 标注的测试。
--enforce-time-limit	根据测试规模对其加上执行时长限制。详情参见第 6 章有风险的测试。
--process-isolation	每个测试都在独立的PHP进程中运行。
--no-globals-backup	不要备份并还原 $GLOBALS。更多细节请参见“全局状态”一节。
--static-backup	备份并还原用户定义的类中的静态属性。更多细节请参见“全局状态”一节。
--colors	使用彩色输出。Windows下，用 ANSICON [https://github.com/adoxa/ansicon] 或 ConEmu [https://github.com/Maximus5/ConEmu]。 本选项有三个可能的值： • never: 完全不使用彩色输出。当未使用 --colors 选项时，这是默认值。 • auto: 如果当前终端不支持彩色、或者输出被管道输出至其他命令、或输出被重定向至文件时，不使用彩色输出，其余情况使用彩色。 • always: 总是使用彩色输出，即使当前终端不支持彩色、输出被管道输出至其他命令、或输出被重定向至文件。 当使用了 --colors 选项但未指定任何值时，将选择 auto 做为其值。
--columns	定义输出所使用的列数。如果将其值定义为 max，则使用当前终端所支持的最大列数。
--stderr	选择输出到 STDERR 而非 STDOUT.
--stop-on-error	首次错误出现后停止执行。
--stop-on-failure	首次错误或失败出现后停止执行。
--stop-on-risky	首次碰到有风险的测试时停止执行。
--stop-on-skipped	首次碰到跳过的测试时停止执行。
--stop-on-incomplete	首次碰到不完整的测试时停止执行。
--verbose	输出更详尽的信息，例如不完整或者跳过的测试的名称。
--debug	输出调试信息，例如当一个测试开始执行时输出其名称。
--loader	指定要使用的 PHPUnit_Runner_TestSuiteLoader 实现。 标准的测试套件加载器将在当前工作目录和 PHP 的 include_path 配置指令中指定的每个目录内查找源文件。诸如 Project_Package_Class 这样的类名对应的源文件名为 Project/Package/Class.php。
--repeat	将测试重复运行指定次数。

`--testdox`	将测试进度以敏捷文档方式报告。更多细节请参见第 12 章 测试的其他用途。
`--printer`	指定要使用的结果输出器(printer)。输出器类必须扩展 `PHPUnit_Util_Printer` 并且实现 `PHPUnit\Framework\TestListener` 接口。
`--bootstrap`	在测试前先运行一个 "bootstrap" PHP 文件。
`--configuration`, `-c`	从 XML 文件中读取配置信息。更多细节请参见附录 C, *XML* 配置文件。 如果 `phpunit.xml` 或 `phpunit.xml.dist`（按此顺序）存在于当前工作目录并且未使用 `--configuration`，将自动从此文件中读取配置。
`--no-configuration`	忽略当前工作目录下的 `phpunit.xml` 与 `phpunit.xml.dist`。
`--include-path`	向 PHP 的 `include_path` 开头添加指定路径（可以多个）。
`-d`	设置指定的 PHP 配置选项的值。

注意

请注意，从 4.8 开始，选项不能放在参数之后。

第 4 章 基境(fixture)

在编写测试时，最费时的部分之一是编写代码来将整个场景设置成某个已知的状态，并在测试结束后将其复原到初始状态。这个已知的状态称为测试的 基境*(fixture)*。

在例 2.1 “用 PHPUnit 测试数组操作”中，基境十分简单，就是存储在 `$stack` 变量中的数组。然而，绝大多数时候基境均远比一个简单数组要复杂，用于建立基境的代码量也会随之增长。测试的真正内容就被淹没于建立基境带来的干扰中。当编写多个需要类似基境的测试时这个问题就变得更糟糕了。如果没有来自于测试框架的帮助，就不得不在写每一个测试时都将建立基境的代码重复一次。

PHPUnit 支持共享建立基境的代码。在运行某个测试方法前，会调用一个名叫 `setUp()` 的模板方法。`setUp()` 是创建测试所用对象的地方。当测试方法运行结束后，不管是成功还是失败，都会调用另外一个名叫 `tearDown()` 的模板方法。`tearDown()` 是清理测试所用对象的地方。

在例 2.2 “用 `@depends` 标注来表达依赖关系”中，我们在测试之间运用生产者-消费者关系来共享基境。这并非总是预期的方式，甚至有时是不可能的。例 4.1 “用 setUp() 建立栈的基境”展示了另外一个编写测试 `StackTest` 的方式。在这个方式中，不再重用基境本身，而是重用建立基境的代码。首先声明一个实例变量，`$stack`，用来替代方法内的局部变量。然后把 `array` 基境的建立放到 `setUp()` 方法中。最后，从测试方法中去除冗余代码，在 `assertEquals()` 断言方法中使用新引入的实例变量 `$this->stack`替代方法内的局部变量 `$stack`。

例 4.1. 用 setUp() 建立栈的基境

```
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    protected $stack;

    protected function setUp()
    {
        $this->stack = [];
    }

    public function testEmpty()
    {
        $this->assertTrue(empty($this->stack));
    }

    public function testPush()
    {
        array_push($this->stack, 'foo');
        $this->assertEquals('foo', $this->stack[count($this->stack)-1]);
        $this->assertFalse(empty($this->stack));
    }

    public function testPop()
    {
        array_push($this->stack, 'foo');
        $this->assertEquals('foo', array_pop($this->stack));
        $this->assertTrue(empty($this->stack));
    }
}
?>
```

测试类的每个测试方法都会运行一次 `setUp()` 和 `tearDown()` 模板方法（同时，每个测试方法都是在一个全新的测试类实例上运行的）。

另外，setUpBeforeClass() 与 tearDownAfterClass() 模板方法将分别在测试用例类的第一个测试运行之前和测试用例类的最后一个测试运行之后调用。

下面这个例子中展示了测试用例类中所有可用的模板方法。

例 4.2. 展示所有可用模板方法的例子

```
<?php
use PHPUnit\Framework\TestCase;

class TemplateMethodsTest extends TestCase
{
    public static function setUpBeforeClass()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    protected function setUp()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    protected function assertPreConditions()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    public function testOne()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
        $this->assertTrue(true);
    }

    public function testTwo()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
        $this->assertTrue(false);
    }

    protected function assertPostConditions()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    protected function tearDown()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    public static function tearDownAfterClass()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

    protected function onNotSuccessfulTest(Exception $e)
    {
        fwrite(STDOUT, __METHOD__ . "\n");
        throw $e;
    }
}
?>
```

```
phpunit TemplateMethodsTest
```

```
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

TemplateMethodsTest::setUpBeforeClass
TemplateMethodsTest::setUp
TemplateMethodsTest::assertPreConditions
TemplateMethodsTest::testOne
TemplateMethodsTest::assertPostConditions
TemplateMethodsTest::tearDown
.TemplateMethodsTest::setUp
TemplateMethodsTest::assertPreConditions
TemplateMethodsTest::testTwo
TemplateMethodsTest::tearDown
TemplateMethodsTest::onNotSuccessfulTest
FTemplateMethodsTest::tearDownAfterClass


Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) TemplateMethodsTest::testTwo
Failed asserting that <boolean:false> is true.
/home/sb/TemplateMethodsTest.php:30

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

setUp() 多 tearDown() 少

理论上说，setUp() 和 tearDown() 是精确对称的，但是实践中并非如此。实际上，只有在 setUp() 中分配了诸如文件或套接字之类的外部资源时才需要实现 tearDown()。如果 setUp() 中只创建纯 PHP 对象，通常可以略过 tearDown()。不过，如果在 setUp() 中创建了大量对象，你可能想要在 tearDown() 中 unset() 指向这些对象的变量，这样它们就可以被垃圾回收机制回收掉。对测试用例对象的垃圾回收动作则是不可预知的。

变体

如果两个基境建立工作略有不同的测试该怎么办？有两种可能：

- 如果两个 setUp() 代码仅有微小差异，把有差异的代码内容从 setUp() 移到测试方法内。
- 如果两个 setUp() 是确实不一样，那么需要另外一个测试用例类。参考基境建立工作的不同之处来命名这个类。

基境共享

有几个好的理由来在测试之间共享基境，但是大部分情况下，在测试之间共享基境的需求都源于某个未解决的设计问题。

一个有实际意义的多测试间共享基境的例子是数据库连接：只登录数据库一次，然后重用此连接，而不是每个测试都建立一个新的数据库连接。这样能加快测试的运行。

例 4.3 “在同一个测试套件内的不同测试之间共享基境”用 setUpBeforeClass() 和 tearDownAfterClass() 模板方法来分别在测试用例类的第一个测试之前和最后一个测试之后连接与断开数据库。

例 4.3. 在同一个测试套件内的不同测试之间共享基境

```
<?php
```

```
use PHPUnit\Framework\TestCase;

class DatabaseTest extends TestCase
{
    protected static $dbh;

    public static function setUpBeforeClass()
    {
        self::$dbh = new PDO('sqlite::memory:');
    }

    public static function tearDownAfterClass()
    {
        self::$dbh = null;
    }
}
?>
```

需要反复强调的是：在测试之间共享基境会降低测试的价值。潜在的设计问题是对象之间并非松散耦合。如果解决掉潜在的设计问题并使用桩件(stub)（参见第 9 章 测试替身）来编写测试，就能达成更好的结果，而不是在测试之间产生运行时依赖并错过改进设计的机会。

全局状态

使用单件(singleton)的代码很难测试。 [http://googletesting.blogspot.com/2008/05/tott-using-dependancy-injection-to.html]使用全局变量的代码也一样。通常情况下，欲测代码和全局变量之间会强烈耦合，并且其创建无法控制。另外一个问题是，一个测试对全局变量的改变可能会破坏另外一个测试。

在 PHP 中，全局变量是这样运作的：

- 全局变量 `$foo = 'bar';` 实际上是存储为 `$GLOBALS['foo'] = 'bar';` 的。
- `$GLOBALS`这个变量是一种被称为超全局变量的变量。
- 超全局变量是一种在任何变量作用域中都总是可用的内建变量。
- 在函数或者方法的变量作用域中，要访问全局变量 `$foo`，可以直接访问 `$GLOBALS['foo']`，或者用 `global $foo;` 来创建一个引用全局变量的局部变量。

除了全局变量，类的静态属性也是一种全局状态。

默认情况下，PHPUnit 用一种更改全局变量与超全局变量 (`$GLOBALS`、`$_ENV`、`$_POST`、`$_GET`、`$_COOKIE`、`$_SERVER`、`$_FILES`、`$_REQUEST`) 不会影响到其他测试的方式来运行所有测试。同时，还可以选择将这种隔离扩展到类的静态属性。

注意

对全局变量和类的静态属性的备份与还原操作使用了 `serialize()` 与 `unserialize()`。

某些类的实例对象（比如 PDO）无法序列化，因此如果把这样一个对象存放在比如说 `$GLOBALS` 数组内时，备份操作就会出问题。

在“@backupGlobals”一节中所讨论的 `@backupGlobals` 标注可以用来控制对全局变量的备份与还原操作。另外，还可以提供一个全局变量的黑名单，黑名单中的全局变量将被排除于备份与还原操作之外，就像这样：

```
class MyTest extends TestCase
{
```

```
    protected $backupGlobalsBlacklist = ['globalVariable'];

    // ...
}
```

注意

在方法（例如 setUp() 方法）内对 $backupGlobalsBlacklist 属性进行设置是无效的。

在 “@backupStaticAttributes”一节 中提到的 @backupStaticAttributes 标注可以用于在每个测试之前备份所有已声明类的静态属性值并在其后恢复。

它所处理的并不只是测试类自身，而是在测试开始时已声明的所有类。它只作用于静态类属性，不作用于函数内声明的静态变量。

注意

只有启用了 @backupStaticAttributes 的测试方法才会在方法之前执行此操作。如果在此之前运行的某个没有启用 @backupStaticAttributes 的测试方法改变了静态属性的值，那么被备份及还原的将会是这个改变后的值——而非初始声明时提供的默认值。PHP 并不额外记录任何静态变量的声明时提供的初始默认值。

同样的情况也发生于测试内部新加载/声明的类的静态属性上。它们也无法在测试结束之后复原为声明时提供的原始默认值，因为无从得知这些默认值。这些被修改过的值会泄漏到后继测试中。

对单元测试而言，推荐在 setUp() 中显式的重置测试中使用到的静态属性（最好同时在 tearDown() 中执行重置，这样就保证不会影响到后继的测试）。

可以提供黑名单来将静态属性从备份与还原操作中排除出去：

```
class MyTest extends TestCase
{
    protected $backupStaticAttributesBlacklist = [
        'className' => ['attributeName']
    ];

    // ...
}
```

注意

在方法（例如 setUp() ）内对 $backupStaticAttributesBlacklist 属性进行设置是无效的。

第 5 章 组织测试

PHPUnit 的目标之一是测试应当可组合：我们希望能将任意数量的测试以任意组合方式运行，例如，整个项目的所有测试，或者项目中的某个组件内的所有类的测试，又或者仅仅某单个类的测试。

PHPUnit 支持好几种不同的方式来组织测试以及将它们编排组合成测试套件。本章介绍了最常用的方法。

用文件系统来编排测试套件

编排测试套件的各种方式中，最简单的大概就是把所有测试用例源文件放在一个测试目录中。通过对测试目录进行递归遍历，PHPUnit 能自动发现并运行测试。

现在来看看 sebastianbergmann/money [http://github.com/sebastianbergmann/money/] 这个库的测试套件。在这个项目的目录结构中，可以看到 `tests` 目录下的测试用例类镜像了 `src` 目录下被测系统(SUT, System Under Test)的包(package)与类(class)的结构：

```
src                                 tests
`-- Currency.php                    `-- CurrencyTest.php
`-- IntlFormatter.php               `-- IntlFormatterTest.php
`-- Money.php                       `-- MoneyTest.php
`-- autoload.php
```

要运行这个库的全部测试，只要将 PHPUnit 命令行测试执行器指向测试目录即可：

```
phpunit --bootstrap src/autoload.php tests
PHPUnit 6.5.0 by Sebastian Bergmann.

.................................

Time: 636 ms, Memory: 3.50Mb

OK (33 tests, 52 assertions)
```

注意

当 PHPUnit 命令行测试执行器指向一个目录时，它会在目录下查找 `*Test.php` 文件。

如果只想运行在 `CurrencyTest` 文件中的 `tests/CurrencyTest.php` 测试用例类中声明的测试，可以使用如下命令：

```
phpunit --bootstrap src/autoload.php tests/CurrencyTest
PHPUnit 6.5.0 by Sebastian Bergmann.

........

Time: 280 ms, Memory: 2.75Mb

OK (8 tests, 8 assertions)
```

如果想要对运行哪些测试有更细粒度的控制，可以使用 `--filter` 选项：

```
phpunit --bootstrap src/autoload.php --filter testObjectCanBeConstructedForValidConstruc
PHPUnit 6.5.0 by Sebastian Bergmann.

..
```

```
Time: 167 ms, Memory: 3.00Mb

OK (2 test, 2 assertions)
```

注意

这种方法的缺点是无法控制测试的运行顺序。这可能导致测试的依赖关系方面的问题，参见“测试的依赖关系”一节。在下一节中，可以看到如何用 XML 配置文件来明确指定测试的执行顺序。

用 XML 配置来编排测试套件

PHPUnit的 XML 配置文件（附录 C, *XML* 配置文件）也可以用于编排测试套件。例 5.1 “用 XML 配置来编排测试套件”展示了一个最小化的 `phpunit.xml` 例子，它将在递归遍历 `tests` 时添加所有在 `*Test.php` 文件中找到的 `*Test` 类。

例 5.1. 用 XML 配置来编排测试套件

```
<phpunit bootstrap="src/autoload.php">
  <testsuites>
    <testsuite name="money">
      <directory>tests</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

如果 `phpunit.xml` 或 `phpunit.xml.dist`（按此顺序）存在于当前工作目录并且未使用 `--configuration`，将自动从此文件中读取配置。

可以明确指定测试的执行顺序：

例 5.2. 用 XML 配置来编排测试套件

```
<phpunit bootstrap="src/autoload.php">
  <testsuites>
    <testsuite name="money">
      <file>tests/IntlFormatterTest.php</file>
      <file>tests/MoneyTest.php</file>
      <file>tests/CurrencyTest.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

第 6 章 有风险的测试

在执行测试时，PHPUnit 可以进行一些额外的检查，见下文。

无用测试

PHPUnit 可以更严格对待事实上不测试任何内容的测试。此项检查可以用命令行选项 `--report-useless-tests` 或在 PHPUnit 的 XML 配置文件中设置 `beStrictAboutTestsThatDoNotTestAnything="true"` 来启用。

在启用本项检查后，如果某个测试未进行任何断言，它将被标记为有风险。仿件对象中的预期和诸如 `@expectedException` 这样的标注同样视为断言。

意外的代码覆盖

PHPUnit 可以更严格对待意外的代码覆盖。此项检查可以用命令行选项 `--strict-coverage` 或在 PHPUnit 的 XML 配置文件中设置 `checkForUnintentionallyCoveredCode="true"` 来启用。

在启用本项检查后，如果某个带有 `@covers` 标注的测试执行了未在 `@covers` 或 `@uses` 标注中列出的代码，它将被标记为有风险。

测试执行期间产生的输出

PHPUnit 可以更严格对待测试执行期间产生的输出。 此项检查可以用命令行选项 `--disallow-test-output` 或在 PHPUnit 的 XML 配置文件中设置 `beStrictAboutOutputDuringTests="true"` 来启用。

在启用本项检查后，如果某个测试产生了输出，例如，在测试代码或被测代码中调用了 `print`，它将被标记为有风险。

测试执行时长的超时限制

如果安装了 `PHP_Invoker` 包并且 `pcntl` 扩展可用，那么可以对测试的执行时长进行限制。此时间限制可以用命令行选项 `--enforce-time-limit` 或在 PHPUnit 的 XML 配置文件中设置 `beStrictAboutTestSize="true"` 来启用。

带有 `@large` 标注的测试如果执行时间超过60秒将视为失败。此超时限制可以通过XML配置文件中的 `timeoutForLargeTests` 属性进行配置。

带有 `@medium` 标注的测试如果执行时间超过10秒将视为失败。此超时限制可以通过XML配置文件中的 `timeoutForMediumTests` 属性进行配置。

没有 `@medium` 或 `@large` 标注的测试都将视同为带有 `@small`标注，这类测试如果执行时间超过1秒将视为失败。此超时限制可以通过XML配置文件中的 `timeoutForSmallTests` 属性进行配置。

全局状态篡改

PHPUnit 可以更严格对待篡改全局状态的测试。此项检查可以用命令行选项 `--strict-global-state` 或在 PHPUnit 的 XML 配置文件中设置 `beStrictAboutChangesToGlobalState="true"` 来启用。

第 7 章 未完成的测试与跳过的测试

未完成的测试

开始写新的测试用例类时，可能想从写下空测试方法开始，比如：

```
public function testSomething()
{
}
```

以此来跟踪需要编写的测试。空测试的问题是 PHPUnit 框架会将它们解读为成功。这种错误解读导致错误报告变得毫无用处——无法分辨出测试是真的成功了还是根本就未编写实现。在未实现的测试中调用 `$this->fail()` 同样没啥帮助，因为测试将被解读为失败。这和将未实现的测试解读为成功是一样的错误。

假如把成功的测试视为绿灯、测试失败视为红灯，那么还额外需要黄灯来将测试标记为未完成或尚未实现。`PHPUnit_Framework_IncompleteTest` 是一个标记接口，用于将测试方法抛出的异常标记为测试未完成或目前尚未实现而导致的结果。`PHPUnit_Framework_IncompleteTestError` 是这个接口的标准实现。

例 7.1 “将测试标记为未完成”展示了一个测试用例类 `SampleTest`，它有一个测试方法 `testSomething()`。通过在测试方法中调用便捷方法 `markTestIncomplete()`（会自动抛出一个 `PHPUnit_Framework_IncompleteTestError` 异常）将这个测试标记为未完成。

例 7.1. 将测试标记为未完成

```
<?php
use PHPUnit\Framework\TestCase;

class SampleTest extends TestCase
{
    public function testSomething()
    {
        // 可选：如果愿意，在这里随便测试点什么。
        $this->assertTrue(true, '这应该已经是能正常工作的。');

        // 在这里停止，并将此测试标记为未完成。
        $this->markTestIncomplete(
          '此测试目前尚未实现。'
        );
    }
}
?>
```

在 PHPUnit 命令行测试执行器的输出中，未完成的测试记为 `I`，如下例所示：

```
phpunit --verbose SampleTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

I

Time: 0 seconds, Memory: 3.95Mb

There was 1 incomplete test:

1) SampleTest::testSomething
This test has not been implemented yet.
```

```
/home/sb/SampleTest.php:12
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 1, Incomplete: 1.
```

表 7.1 “用于未完成的测试的 API”列举了用于将测试标记为未完成的 API。

表 7.1. 用于未完成的测试的 API

方法	含义
`void markTestIncomplete()`	将当前测试标记为未完成。
`void markTestIncomplete(string $message)`	将当前测试标记为未完成，并用 `$message` 作为说明信息。

跳过测试

并非所有测试都能在任何环境中运行。比如说，考虑这样一种情况：一个数据库抽象层，针对其所支持的各种数据库系统有多个不同的驱动程序。针对 MySQL 驱动程序的测试当然只在 MySQL 服务器可用才能运行。

例 7.2 “跳过某个测试” 展示了一个测试用例类 `DatabaseTest`，它有一个测试方法 `testConnection()`。在测试用例类的 `setUp()`模板方法中，检查了 MySQLi 扩展是否可用，并且在扩展不可用时用 `markTestSkipped()` 方法来跳过此测试。

例 7.2. 跳过某个测试

```
<?php
use PHPUnit\Framework\TestCase;

class DatabaseTest extends TestCase
{
    protected function setUp()
    {
        if (!extension_loaded('mysqli')) {
            $this->markTestSkipped(
              'MySQLi 扩展不可用。'
            );
        }
    }

    public function testConnection()
    {
        // ...
    }
}
?>
```

在 PHPUnit 命令行测试执行器的输出中，被跳过的测试记为 S，如下例所示：

```
phpunit --verbose DatabaseTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

S

Time: 0 seconds, Memory: 3.95Mb

There was 1 skipped test:

1) DatabaseTest::testConnection
```

```
The MySQLi extension is not available.

/home/sb/DatabaseTest.php:9
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Skipped: 1.
```

表 7.2 “用于跳过测试的 API”列举了用于跳过测试的 API。

表 7.2. 用于跳过测试的 API

方法	含义
`void markTestSkipped()`	将当前测试标记为已跳过。
`void markTestSkipped(string $message)`	将当前测试标记为已跳过，并用 `$message` 作为说明信息。

用 @requires 来跳过测试

除了上述方法，还可以用 `@requires` 标注来表达测试用例的一些常见前提条件。

表 7.3. 可能的 @requires 用法

类型	可能的值	范例	其他范例
`PHP`	任何 PHP 版本标识符	@requires PHP 5.3.3	@requires PHP 7.1-dev
`PHPUnit`	任何 PHPUnit 版本标识符	@requires PHPUnit 3.6.3	@requires PHPUnit 4.6
`OS`	用来对 PHP_OS [http://php.net/manual/en/reserved.constants.php#constant.php-os] 进行匹配的正则表达式	@requires OS Linux	@requires OS WIN32|WINNT
`function`	任何对 function_exists [http://php.net/function_exists] 而言有效的参数	@requires function imap_open	@requires function ReflectionMethod::setAccessible
`extension`	任何扩展模块名，可以附带有版本标识符	@requires extension mysqli	@requires extension redis 2.2.0

例 7.3. 用 @requires 来跳过测试

```
<?php
use PHPUnit\Framework\TestCase;

/**
 * @requires extension mysqli
 */
class DatabaseTest extends TestCase
{
    /**
     * @requires PHP 5.3
     */
    public function testConnection()
    {
        // 测试要求有 mysqli 扩展，并且 PHP >= 5.3
    }
```

```
    // ... 所有其他要求有 mysqli 扩展的测试
}
?>
```

如果使用了某种在特定版本的 PHP 下无法编译的语法，请在此章节内查找 XML 配置信息中关于版本依赖的信息：“测试套件”一节

第 8 章 数据库测试

在各种编程语言中，许多入门与中级的单元测试范例都暗示着这样一种信息：很容易用简单的测试来对应用程序的逻辑进行测试。但是对于以数据库为中心的应用程序而言，这与现实相去甚远。一旦开始使用诸如 Wordpress、TYPO3、或 Symfony（配合 Doctrine 或 Propel）之类的东西，就很容易在用 PHPUnit 时碰到超多问题：正是由于这些库和数据库之间实在耦合的太紧密了。

> **注意**
>
> 请确保已经安装了 PHP 扩展模块 pdo 和与数据库对应的特定扩展，比如 pdo_mysql。否则以下范例是无法运行的。

你大概会在日常工作面对的项目中经历这一幕。你打算把你那或生疏或纯熟的 PHPUnit 技能用到工作中去，结果被以下问题之一卡住了：

1. 待测方法执行了一个相当大的 JOIN 操作，并且得到的数据用于计算某些重要的结果。
2. 业务逻辑中混合执行了 SELECT、INSERT、UPDATE 和 DELETE 语句。
3. 为了给待测方法建立合理的初始数据，需要在两个以上（可能远超过）表里设置测试数据。

DbUnit 扩展大大简化了为测试设置数据库的操作，并且可以在对数据执行了一系列操作之后验证数据库的内容。

数据库测试所支持的供应商

DbUnit 目前支持 MySQL、PostgreSQL、Oracle 和 SQLite。通过集成 Zend Framework [http://framework.zend.com] 或 Doctrine 2 [http://www.doctrine-project.org]，也可以访问其他数据库系统，比如 IBM DB2 或者 Microsoft SQL Server。

数据库测试的难点

为什么所有单元测试的范例都不包含数据库交互？这里有个很好的理由：这类测试的建立和维护都很复杂。对数据库进行测试时，需要考虑以下这些变数：

- 数据库和表
- 向表中插入测试所需要的行
- 测试运行完毕后验证数据库的状态
- 每个新测试都要清理数据库

许多数据库 API，比如 PDO、MySQLi 或者 OCI8，都十分繁琐且书写起来十分冗长，因此，手工进行这些步骤绝对是噩梦。

测试代码应当尽可能简短精确，这有若干原因：

- 你不希望因为生产代码的小变更而需要对测试代码进行数量可观的修改。
- 你希望在哪怕好几个月以后也能轻松地阅读并理解测试代码。

另外，必须认识到，对于代码而言，本质上来说数据库是全局输入变量。测试套件中的两个不同的测试可能是运行在同一个数据库上的，并且可能把数据重用好多次。一个测试中出现

的失败很容易影响到后继测试的结果，从而让整个测试过程变得非常艰难。前面提到的清理步骤对于解决“数据库是全局输入”的问题是非常重要的。

DbUnit 以一种优雅的方式来帮助简化数据库测试中的所有这些问题。

PHPUnit 无法帮你解决的问题是，相对于不使用数据的测试而言，数据库测试是非常慢的。随着数据库交互规模的增大，运行测试可能需要耗费可观的时间。然而，只要保持每个测试所使用的数据量较小并且尽可能用非数据库测试来对代码进行测试，即使很大的测试套件也能轻松在一分钟内跑完。

以 Doctrine 2 [http://www.doctrine-project.org] 为例，此项目的测试套件目前包含了大约1000个测试，其中将近一半访问了数据库。但是在一台安装了MySQL的普通的台式机上，整个测试套件依然能在15秒钟内跑完。

数据库测试的四个阶段

Gerard Meszaros 在他的书《xUnit 测试模式》中列出了单元测试的四个阶段：

1. 建立基境(fixture)
2. 执行被测系统
3. 验证结果
4. 拆除基境(fixture)

> 什么是基境(fixture)？
>
> 基境(fixture)是对开始执行某个测试时应用程序和数据库所处初始状态的描述。

对数据库进行测试至少要处理建立与拆除的步骤，在其中完成清理工作，并将所需的基境数据写入表内。因而，对于数据库扩展模块而言，在数据库测试中有很好的理由将这四个步骤还原成类似下面这样的工作流程，这个流程对于每个测试都会完整执行：

1. 清理数据库

由于总是会有某个测试运行在并不确定表中是否有数据的数据库上，PHPUnit 在所有指定表上执行 TRUNCATE 操作来把它们清空。

2. 建立基境

PHPUnit 随后将迭代所有指定的基境数据行并将其插入到对应的表里。

3–5. 运行测试、验证结果、并拆除基境

在所有数据库都完成重置并加载好初始状态后，PHPUnit 才会执行实际的测试。这个部分的测试代码完全不需要数据库扩展模块的参与，可以随意测试任何想要测试的内容。

在测试中，验证的目的可以使用一个名为 `assertDataSetsEqual()` 的特殊断言来实现。当然，这完全是可选的。这个特性将在“数据库断言”一节中进行解说。

PHPUnit 数据库测试用例的配置

一般而言，使用 PHPUnit 时，测试用例都是按如下方式扩展自 `PHPUnit\Framework\TestCase` 类：

```
<?php
```

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    public function testCalculate()
    {
        $this->assertEquals(2, 1 + 1);
    }
}
?>
```

如果测试代码用到了数据库扩展模块，那么建立的过程就会更复杂一些，需要扩展另一个抽象 TestCase 类，它要求实现两个抽象方法，getConnection() 和 getDataSet()：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class MyGuestbookTest extends TestCase
{
    use TestCaseTrait;

    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $pdo = new PDO('sqlite::memory:');
        return $this->createDefaultDBConnection($pdo, ':memory:');
    }

    /**
     * @return PHPUnit_Extensions_Database_DataSet_IDataSet
     */
    public function getDataSet()
    {
        return $this->createFlatXMLDataSet(dirname(__FILE__).'/_files/guestbook-seed.xml
    }
}
?>
```

实现 getConnection()

为了让清理与载入基境的功能正常运作，PHPUnit 数据库扩展模块需要用 PDO 库来实现跨供应商抽象访问数据库连接。重要的是要注意到，使用 PHPUnit 的数据库扩展模块并不要求应用程序本身基于PDO，PDO连接仅仅用于清理和建立基境。

在之前的例子里，我们在内存中创建 Sqlite 数据库并建立了连接，将此连接传递给 createDefaultDBConnection 方法，这个方法将 PDO 实例和第二参数（数据库名）包装在一个非常简单的数据库连接抽象层中，这个抽象层的类型是 PHPUnit_Extensions_Database_DB_IDatabaseConnection。

“使用数据库连接”一节解说了这个接口的API以及如何充分利用它们。

实现 getDataSet()

getDataSet() 方法定义了在每个测试执行之前的数据库初始状态应该是什么样。数据库的状态通过由 PHPUnit_Extensions_Database_DataSet_IDataSet 所代表的 DataSet（数据集）和由 PHPUnit_Extensions_Database_DataSet_IDataTable所代表的 DataTable（数据表）这两个概念进行抽象。下一节将详细讲述这些概念是如何运作的以及在数据库测试中使用它们有什么好处。

对于具体实现，只需要知道 setUp() 中会调用一次 getDataSet() 方法来接收基境数据集并将其插入数据库。在范例中使用了工厂方法 createFlatXMLDataSet($filename)，它代表一个用 XML 表示的数据集。

数据库构架(DDL)怎么办？

PHPUnit 假设在测试运行之前数据库以及其中的所有表(table)、触发器(trigger)、序列(Sequence)和视图(view)都已经创建好。这意味着开发者必须在运行测试套件之前确保数据库已经正确建立。

有几种方法来达成这个数据库测试的先决条件。

1. 如果使用的是持久化数据库(不是 Sqlite Memory)，可以很轻松地用 phpMyAdmin（针对 MySQL）之类的工具来一次性建立数据库，并在每个测试中复用这个数据库。

2. 如果使用的是诸如 Doctrine 2 [http://www.doctrine-project.org] 或 Propel [http://www.propelorm.org/] 这样的库，可以用它们的API来在测试运行前一次性建立所需的数据库。可以利用 PHPUnit 的引导和配置 [textui.html] 功能来在每次测试运行时执行这些代码。

小建议：使用你自己的抽象数据库 TestCase 类

从前面的实现范例中容易发现 getConnection() 方法是相当稳定的，可以在不同的数据库测试用例中重用。另外，为了保持测试的性能良好和数据库的开销较低，可以对代码进行一点重构，来为应用程序形成一个通用的抽象测试用例，同时依然可以为每个具体测试用例指定不同的数据基境：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

abstract class MyApp_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // only instantiate pdo once for test clean-up/fixture load
    static private $pdo = null;

    // only instantiate PHPUnit_Extensions_Database_DB_IDatabaseConnection once per test
    private $conn = null;

    final public function getConnection()
    {
        if ($this->conn === null) {
            if (self::$pdo == null) {
                self::$pdo = new PDO('sqlite::memory:');
            }
            $this->conn = $this->createDefaultDBConnection(self::$pdo, ':memory:');
        }

        return $this->conn;
    }
}
?>
```

这个例子里，数据库连接信息硬编码在 PDO 连接里了。PHPUnit 有另外一个绝妙的特性，可以让这个 TestCase 类更加通用。通过 XML 配置 [appendixes.configuration.html#appendixes.configuration.php-ini-constants-variables] 可以为每个测试单独配置数据库连接信息。首先，在应用程序的 tests/ 目录下创建 “phpunit.xml” 文件，内容大体是这样：

```
<?xml version="1.0" encoding="UTF-8" ?>
<phpunit>
    <php>
        <var name="DB_DSN" value="mysql:dbname=myguestbook;host=localhost" />
        <var name="DB_USER" value="user" />
        <var name="DB_PASSWD" value="passwd" />
        <var name="DB_DBNAME" value="myguestbook" />
    </php>
</phpunit>
```

现在可以修改 TestCase 类了，像这样：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

abstract class Generic_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // only instantiate pdo once for test clean-up/fixture load
    static private $pdo = null;

    // only instantiate PHPUnit_Extensions_Database_DB_IDatabaseConnection once per test
    private $conn = null;

    final public function getConnection()
    {
        if ($this->conn === null) {
            if (self::$pdo == null) {
                self::$pdo = new PDO( $GLOBALS['DB_DSN'], $GLOBALS['DB_USER'], $GLOBALS[
            }
            $this->conn = $this->createDefaultDBConnection(self::$pdo, $GLOBALS['DB_DBNA
        }

        return $this->conn;
    }
}
?>
```

现在可以从命令行界面以不同的配置来运行数据库测试套件了：

```
user@desktop> phpunit --configuration developer-a.xml MyTests/
user@desktop> phpunit --configuration developer-b.xml MyTests/
```

在开发机上进行开发时能够轻松的针对不同的目标数据库来运行数据库测试显得非常重要。如果多个开发人员在同一个数据库连接上运行数据库测试，很容易因为竞态而导致测试失败。

理解 DataSet（数据集）和 DataTable（数据表）

PHPUnit 的数据库扩展模块的核心概念是 DataSet（数据集）和 DataTable（数据表）。为了掌握如何使用 PHPUnit 进行测试，需要试着去了解这些简单的概念。DataSet（数据集）和 DataTable（数据表）是围绕着数据库表、行、列的抽象层。通过一套简单的API，底层数据库内容被隐藏在对象结构之下，同时，这个对象结构也可以用其他非数据库数据源来实现。

为了能比较实际内容和预期内容，这种抽象是必须的。预期内容可以用诸如 XML、 YAML、CSV 文件或者 PHP 数组等方式来表达。DataSet 和 DataTable 接口以语义相似的方式模拟关系数据库存储，从而能够对这些概念上完全不同的数据源进行比较。

在测试中，数据库断言的工作流由以下三个简单的步骤组成：

- 用表名称来指定数据库中的一个或多个表（实际上是指定了一个数据集）
- 用你喜欢的格式（YAML、XML等等）来指定预期数据集
- 断言这两个数据集陈述是彼此相等的。

在 PHPUnit 的数据库扩展中，断言并非唯一使用 DataSet 和 DataTable 的情形。就像上一节中所展示的那样，它们也用于描述数据库的初始内容。数据库 TestCase 类强制要求定义一个基境数据集，随后用它来：

- 根据此数据集所指定的所有表名，将数据库中对应表内的行全部删除。
- 将数据集内数据表中的所有行写入数据库。

可用的各种实现

有三种不同类型的 DataSet/DataTable：

- 基于文件的 DataSet 和 DataTable
- 基于查询的 DataSet 和 DataTable
- 筛选与组合 DataSet 和 DataTable

基于文件的数据集和表一般用于初始化基境或描述数据库的预期状态。

Flat XML DataSet（平直 XML 数据集）

最常见的一种数据集名叫 Flat XML。这是一种非常简单的 XML 格式，根节点为 <dataset>，根节点下的每个标签就代表数据库中的一行数据。标签的名称就等于表名，而每个属性代表一个列。一个简单的留言本应用程序的例子大致上可能是这样：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="I like it!" user="nancy" created="2010-04-26 12:14:20" />
</dataset>
```

显然，这非常易于编写。在这里，<guestbook> 是表名，这个表内有两行记录，每行有四个列：“id”、“content”、“user” 和 “created”，以及各自的值。

不过，这种简单性是有代价的。

从上面这个例子里不太容易看出该如何指定一个空表。其实可以插入一个没有属性值的标签，以空表的名字作为标签名。空的 guestbook 表所对应的 Flat XML 文件大致上可能是这样：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook />
</dataset>
```

在 Flat XML DataSet 中，要处理 NULL 值会非常烦。在几乎所有数据库中（Oracle 是个例外），NULL 值和空字符串值是有区别的，这一点在 Flat XML 格式中很难表述。可以在数据行的表述中省略掉对应的属性来表示NULL值。假定上面这个留言本通过在 user 列使用 NULL 值的方式来允许匿名留言，那么 guestbook 表的内容可能是这样：

```
<?xml version="1.0" ?>
<dataset>
```

```
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="I like it!" created="2010-04-26 12:14:20" />
</dataset>
```

在这个例子里第二个条目是匿名发表的。但是这为列的识别带来了一个非常严重的问题。在数据集相等断言的判定过程中，每个数据集都需要指明每个表拥有哪些列。如果有一个列在数据表的所有行里其值都是 NULL，那么数据库扩展模块又该从何得知表中包含这个列呢？

在这里，Flat XML DataSet 做了一个关键假设：一个表的列信息由此表第一行的属性定义决定。在上面这个例子里，这意味着 guestbook 有 “id”、“content”、“user” 和 “created” 这几个列。第二行中 “user” 列没有定义，因此将向数据库中插入 NULL 值。

如果从数据集中删掉第一行，因为没有指定 “user”，guestbook 表拥有的列就只剩下 “id”、“content” 和 “created”。

要在有 NULL 值的情况下有效地使用 Flat XML Dataset，就必须保证每个表的第一行不包含 NULL 值，只有后继的那些行才能省略属性。这就有点棘手，因为数据行的排列顺序也是数据断言的一个相关因素。

反过来，如果在 Flat XML Dataset 中只指明了实际表中所有列的某个子集，那么所有省略掉的列都会设为它们的的默认值。如果某个省略掉的列的定义是 “NOT NULL DEFAULT NULL”，就会出现错误。

总的来说，建议只在不需要 NULL 值的情况下使用 Flat XML Dataset。

可以在数据库 TestCase 中调用 `createFlatXmlDataSet($filename)` 方法来创建 Flat XML Dataset 实例：

```php
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class MyTestCase extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        return $this->createFlatXmlDataSet('myFlatXmlFixture.xml');
    }
}
?>
```

XML DataSet（XML 数据集）

有另外一种更加结构化的 XML DataSet，它写起来有点冗长，但是规避了 Flat XML DataSet 所存在的 NULL 问题。在根节点 `<dataset>` 内，可以指定 `<table>`、`<column>`、`<row>`、`<value>` 和 `<null />` 标签。和上面用 Flat XML 所定义的留言本数据集等价的 XML DataSet 如下：

```
<?xml version="1.0" ?>
<dataset>
    <table name="guestbook">
        <column>id</column>
        <column>content</column>
        <column>user</column>
        <column>created</column>
        <row>
            <value>1</value>
            <value>Hello buddy!</value>
            <value>joe</value>
```

```
            <value>2010-04-24 17:15:23</value>
        </row>
        <row>
            <value>2</value>
            <value>I like it!</value>
            <null />
            <value>2010-04-26 12:14:20</value>
        </row>
    </table>
</dataset>
```

所定义的每个 <table> 都有一个名称，并且必须有对所有列及其名称的定义。其下可以包含零个或任意正整数个 <row> 元素。没有定义 <row> 意味着这是个空表。<value> 和 <null /> 标签必须按照之前给定 <column> 元素的顺序来指定。<null /> 标签显然意味着这个值为 NULL。

可以在数据库 TestCase 中调用 createXmlDataSet($filename) 方法来创建 XML DataSet 实例：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class MyTestCase extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        return $this->createXMLDataSet('myXmlFixture.xml');
    }
}
?>
```

MySQL XML DataSet（MySQL XML 数据集）

这种新的 XML 格式是 MySQL 数据库服务器 [http://www.mysql.com]专用的。PHPUnit 3.5 加入了对这种格式的支持。可以用 mysqldump [http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html] 工具来生成这种格式的文件。与同样为 mysqldump 所支持的 CSV 数据集不同，这种 XML 格式可以在单个文件中包含多个表的数据。要生成这种格式的文件，可以这样调用 mysqldump：

```
mysqldump --xml -t -u [username] --password=[password] [database] > /path/to/file.xml
```

可以在数据库 TestCase 中调用 createMySQLXMLDataSet($filename) 方法来使用这个文件：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class MyTestCase extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        return $this->createMySQLXMLDataSet('/path/to/file.xml');
    }
}
?>
```

YAML DataSet（YAML 数据集）

也可以用 YAML DataSet 来写这个留言本的例子：

```
guestbook:
  -
    id: 1
    content: "Hello buddy!"
    user: "joe"
    created: 2010-04-24 17:15:23
  -
    id: 2
    content: "I like it!"
    user:
    created: 2010-04-26 12:14:20
```

简单方便，同时还解决了和它类似的 FLat XML DataSet 所具有的 NULL 问题。在 YAML 中，只有列名而没有指定值就表示 NULL。空白字符串则这样指定：`column1: ""`。

目前，数据库 TestCase 中没有 YAML DataSet 的工厂方法，因此需要手工进行实例化：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;
use PHPUnit\DbUnit\DataSet\YamlDataSet;

class YamlGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
        return new YamlDataSet(dirname(__FILE__)."/_files/guestbook.yml");
    }
}
?>
```

CSV DataSet（CSV 数据集）

另外一种基于文件的 DataSet 是基于 CSV 文件的。数据集中的每个表用一个单独的 CSV 文件表示。对于留言本的例子，可以这样定义 guestbook-table.csv 文件：

```
id,content,user,created
1,"Hello buddy!","joe","2010-04-24 17:15:23"
2,"I like it!","nancy","2010-04-26 12:14:20"
```

用 Excel 或者 OpenOffice 来对这种格式进行编辑是非常方便的，但是在 CSV DataSet 中无法指定 NULL 值。给出一个空白列的结果是往这个列中插入数据库的默认空值。

可以这样创建 CSV DataSet：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;
use PHPUnit\DbUnit\DataSet\CsvDataSet;

class CsvGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
```

```
        $dataSet = new CsvDataSet();
        $dataSet->addTable('guestbook', dirname(__FILE__)."/_files/guestbook.csv");
        return $dataSet;
    }
}
?>
```

Array DataSe（数组数据集）

在 PHPUnit 的数据库扩展中，（尚）没有基于数组的 DataSet，不过很容易自行实现之。留言本的例子大致是这样：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class ArrayGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
        return new MyApp_DbUnit_ArrayDataSet(
            [
                'guestbook' => [
                    [
                        'id' => 1,
                        'content' => 'Hello buddy!',
                        'user' => 'joe',
                        'created' => '2010-04-24 17:15:23'
                    ],
                    [
                        'id' => 2,
                        'content' => 'I like it!',
                        'user' => null,
                        'created' => '2010-04-26 12:14:20'
                    ],
                ],
            ]
        );
    }
}
?>
```

PHP 版本的 DataSet 相比于所有其他基于文件的 DataSet 相比有很明显的优点：

- PHP 数组显然可以处理 NULL 值。
- 不需要为断言提供任何额外文件，可以直接在 TestCase 中指定。

对于这种 DataSet 而言，和平直 XML、CSV、YAML DataSet 一样，表的列名信息由第一个指定的行的键名定义。在上面这个例子里，就是 “id”、“content”、“user” 和 “created”。

这个数组 DataSet 类的实现是非常简单直接的：

```
<?php
class MyApp_DbUnit_ArrayDataSet extends PHPUnit_Extensions_Database_DataSet_AbstractData
{
    /**
     * @var array
     */
    protected $tables = [];
```

```
    /**
     * @param array $data
     */
    public function __construct(array $data)
    {
        foreach ($data AS $tableName => $rows) {
            $columns = [];
            if (isset($rows[0])) {
                $columns = array_keys($rows[0]);
            }

            $metaData = new PHPUnit_Extensions_Database_DataSet_DefaultTableMetaData($ta
            $table = new PHPUnit_Extensions_Database_DataSet_DefaultTable($metaData);

            foreach ($rows AS $row) {
                $table->addRow($row);
            }
            $this->tables[$tableName] = $table;
        }
    }

    protected function createIterator($reverse = false)
    {
        return new PHPUnit_Extensions_Database_DataSet_DefaultTableIterator($this->table
    }

    public function getTable($tableName)
    {
        if (!isset($this->tables[$tableName])) {
            throw new InvalidArgumentException("$tableName is not a table in the current
        }

        return $this->tables[$tableName];
    }
}
?>
```

Query (SQL) DataSet（查询(SQL)数据集）

对于数据库断言，不仅需要有基于文件的 DataSet，同时也需要有一种内含数据库实际内容的基于查询/SQL 的 DataSet。Query DataSet 在此闪亮登场：

```
<?php
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
$ds->addTable('guestbook');
?>
```

单纯以名称来添加表是一种隐式地用以下查询来定义 DataTable 的方法：

```
<?php
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
$ds->addTable('guestbook', 'SELECT * FROM guestbook');
?>
```

可以在这种用法中为你的表任意指定查询，例如限定行、列，或者加上 ORDER BY 子句：

```
<?php
$ds = new PHPUnit_Extensions_Database_DataSet_QueryDataSet($this->getConnection());
$ds->addTable('guestbook', 'SELECT id, content FROM guestbook ORDER BY created DESC');
?>
```

在关于数据库断言的那一节中有更多关于如何使用 Query DataSet 的细节。

Database (DB) Dataset（数据库数据集）

通过访问测试所使用的数据库连接，可以自动创建包含数据库所有表以及其内容的 DataSet。所使用的数据库由数据库连接工厂方法的第二个参数指定。

可以像 `testGuestbook()` 中那样创建整个数据库所对应的 DataSet，或者像 `testFilteredGuestbook()` 方法中那样用一个白名单来将 DataSet 限制在若干表名的集合上。

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class MySqlGuestbookTest extends TestCase
{
    use TestCaseTrait;

    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $database = 'my_database';
        $user = 'my_user';
        $password = 'my_password';
        $pdo = new PDO('mysql:...', $user, $password);
        return $this->createDefaultDBConnection($pdo, $database);
    }

    public function testGuestbook()
    {
        $dataSet = $this->getConnection()->createDataSet();
        // ...
    }

    public function testFilteredGuestbook()
    {
        $tableNames = ['guestbook'];
        $dataSet = $this->getConnection()->createDataSet($tableNames);
        // ...
    }
}
?>
```

Replacement DataSet（替换数据集）

前面谈到了 Flat XML 和 CSV DataSet 所存在的 NULL 问题，不过有一种稍微有点复杂的解决方法可以让这两种数据集都能正常处理 NULL。

Replacement DataSet 是已有数据集的修饰器(decorator)，能够将数据集中任意列的值替换为其他替代值。为了让留言本的例子能够处理 NULL 值，首先指定类似这样的文件：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="I like it!" user="##NULL##" created="2010-04-26 12:14:20"
</dataset>
```

然后将 Flat XML DataSet 包装在 Replacement DataSet 中：

```
<?php
use PHPUnit\Framework\TestCase;
```

```
use PHPUnit\DbUnit\TestCaseTrait;

class ReplacementTest extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        $ds = $this->createFlatXmlDataSet('myFlatXmlFixture.xml');
        $rds = new PHPUnit_Extensions_Database_DataSet_ReplacementDataSet($ds);
        $rds->addFullReplacement('##NULL##', null);
        return $rds;
    }
}
?>
```

DataSet Filter（数据集筛选器）

如果有一个非常大的基境文件，可以用数据集筛选器来为需要包含在子数据集中的表和列指定白/黑名单。与 DB DataSet 联用来对数据集中的列进行筛选尤其方便。

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class DataSetFilterTest extends TestCase
{
    use TestCaseTrait;

    public function testIncludeFilteredGuestbook()
    {
        $tableNames = ['guestbook'];
        $dataSet = $this->getConnection()->createDataSet();

        $filterDataSet = new PHPUnit_Extensions_Database_DataSet_DataSetFilter($dataSet)
        $filterDataSet->addIncludeTables(['guestbook']);
        $filterDataSet->setIncludeColumnsForTable('guestbook', ['id', 'content']);
        // ..
    }

    public function testExcludeFilteredGuestbook()
    {
        $tableNames = ['guestbook'];
        $dataSet = $this->getConnection()->createDataSet();

        $filterDataSet = new PHPUnit_Extensions_Database_DataSet_DataSetFilter($dataSet)
        $filterDataSet->addExcludeTables(['foo', 'bar', 'baz']); // 只保留 guestbook 表！
        $filterDataSet->setExcludeColumnsForTable('guestbook', ['user', 'created']);
        // ..
    }
}
?>
```

> 注意：不能对同一个表同时应用排除与包含两种列筛选器，只能分别应用于不同的表。另外，表的白名单和黑名单也只能选择其一，不能二者同时使用。

Composite DataSet（组合数据集）

Composite DataSet 能将多个已存在的数据集聚合成单个数据集，因此非常有用。如果多个数据集中存在同样的表，其中的数据行将按照指定的顺序进行追加。例如，假设有两个数据集，*fixture1.xml*：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23" />
</dataset>
```

和 *fixture2.xml*：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="2" content="I like it!" user="##NULL##" created="2010-04-26 12:14:20"
</dataset>
```

通过 Composite DataSet 可以把这两个基境文件聚合在一起：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class CompositeTest extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        $ds1 = $this->createFlatXmlDataSet('fixture1.xml');
        $ds2 = $this->createFlatXmlDataSet('fixture2.xml');

        $compositeDs = new PHPUnit_Extensions_Database_DataSet_CompositeDataSet();
        $compositeDs->addDataSet($ds1);
        $compositeDs->addDataSet($ds2);

        return $compositeDs;
    }
}
?>
```

当心外键

在建立基境的过程中， PHPUnit 的数据库扩展模块按照基境中所指定的顺序将数据行插入到数据库内。假如数据库中使用了外键，这就意味着必须指定好表的顺序，以避免外键约束失败。

实现自有的 DataSet/DataTable

为了理解 DataSet 和 DataTable 的内部实现，让我们来看看 DataSet 的接口。如果没打算自行实现 DataSet 或者 DataTable，可以直接跳过这一部分。

```
<?php
interface PHPUnit_Extensions_Database_DataSet_IDataSet extends IteratorAggregate
{
    public function getTableNames();
    public function getTableMetaData($tableName);
    public function getTable($tableName);
    public function assertEquals(PHPUnit_Extensions_Database_DataSet_IDataSet $other);

    public function getReverseIterator();
}
?>
```

这些 public 接口在数据库 TestCase 中 `assertDataSetsEqual()` 断言内使用，用以检测数据集是否相等。IDataSet 中继承自 `IteratorAggregate` 接口的 `getIterator()` 方法用

于对数据集中的所有表进行迭代。逆序迭代器让 PHPUnit 能够按照与创建时相反的顺序对所有表执行 TRUNCATE 操作，以此来保证满足外键约束。

根据具体实现的不同，要采取不同的方法来将表实例添加到数据集中。例如，在所有基于文件的数据集中，表都是在构造过程中直接从源文件生成并加入数据集中，比如 `YamlDataSet`、`XmlDataSet` 和 `FlatXmlDataSet`均是如此。

数据表则由以下接口表示：

```
<?php
interface PHPUnit_Extensions_Database_DataSet_ITable
{
    public function getTableMetaData();
    public function getRowCount();
    public function getValue($row, $column);
    public function getRow($row);
    public function assertEquals(PHPUnit_Extensions_Database_DataSet_ITable $other);
}
?>
```

除了 `getTableMetaData()` 方法之外，这个接口是一目了然的。数据库扩展模块中的各种断言（将于下一章中介绍）用到了所有这些方法，因此它们全部都是必需的。`getTableMetaData()` 方法需要返回一个实现了 `PHPUnit_Extensions_Database_DataSet_ITableMetaData` 接口的描述表结构的对象。这个对象包含如下信息：

- 表的名称
- 表的列名数组，按照列在结果集中出现的顺序排列。
- 构成主键的列的数组。

这个接口还包含有检验两个表的元数据实例是否彼此相等的断言，供数据集相等断言使用。

数据库连接 API

由数据库 TestCase 中的 `getConnection()` 方法所返回的连接接口有三个很有意思的方法：

```
<?php
interface PHPUnit_Extensions_Database_DB_IDatabaseConnection
{
    public function createDataSet(Array $tableNames = NULL);
    public function createQueryTable($resultName, $sql);
    public function getRowCount($tableName, $whereClause = NULL);

    // ...
}
?>
```

1. `createDataSet()` 方法创建一个在数据集实现一节描述过的 Database (DB) DataSet（数据库数据集）。

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class ConnectionTest extends TestCase
{
    use TestCaseTrait;
```

```
    public function testCreateDataSet()
    {
        $tableNames = ['guestbook'];
        $dataSet = $this->getConnection()->createDataSet();
    }
}
?>
```

2. `createQueryTable()` 方法用于创建 QueryTable 的实例，需要为其指定结果名称和所使用的 SQL 查询。当涉及到结果/表的断言（如后面关于数据库断言 API 那一节所示）时，这个方法会很方便。

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class ConnectionTest extends TestCase
{
    use TestCaseTrait;

    public function testCreateQueryTable()
    {
        $tableNames = ['guestbook'];
        $queryTable = $this->getConnection()->createQueryTable('guestbook', 'SELECT *
    }
}
?>
```

3. `getRowCount()` 方法提供了一种方便的方式来取得表中的行数，并且还可以选择附加一个 WHERE 子句来在计数前对数据行进行过滤。它可以和一个简单的相等断言合用：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class ConnectionTest extends TestCase
{
    use TestCaseTrait;

    public function testGetRowCount()
    {
        $this->assertEquals(2, $this->getConnection()->getRowCount('guestbook'));
    }
}
?>
```

数据库断言 API

作为测试工具，数据库扩展模块理所当然会提供一些断言，可以用来验证数据库的当前状态、表的当前状态、表中数据行的数量。本节将详细描述这部分功能：

对表中数据行的数量作出断言

很多时候，确认表中是否包含特定数量的数据行是非常有帮助的。可以轻松做到这一点，不需要任何额外的使用连接 API 的粘合剂代码。比如说，在往留言本中插入一个新行之后，想要确认在表中除了之前的例子中一直都有的两行之外还有第三行：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;
```

```
class GuestbookTest extends TestCase
{
    use TestCaseTrait;

    public function testAddEntry()
    {
        $this->assertEquals(2, $this->getConnection()->getRowCount('guestbook'), "Pre-Co

        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Hello world!");

        $this->assertEquals(3, $this->getConnection()->getRowCount('guestbook'), "Insert
    }
}
?>
```

对表的状态作出断言

前面的这个断言很有帮助，但是肯定还想要检验表的实际内容，好核实是否所有值都写到了正确的列中。可以通过表断言来做到这一点。

为此，先定义一个 QueryTable 实例，从表名称和 SQL 查询派生出其内容，随后将其与一个基于文件/数组的数据集进行比较：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class GuestbookTest extends TestCase
{
    use TestCaseTrait;

    public function testAddEntry()
    {
        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Hello world!");

        $queryTable = $this->getConnection()->createQueryTable(
            'guestbook', 'SELECT * FROM guestbook'
        );
        $expectedTable = $this->createFlatXmlDataSet("expectedBook.xml")
                              ->getTable("guestbook");
        $this->assertTablesEqual($expectedTable, $queryTable);
    }
}
?>
```

现在需要为这个断言编写Flat XML 文件 *expectedBook.xml*：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="I like it!" user="nancy" created="2010-04-26 12:14:20" />
    <guestbook id="3" content="Hello world!" user="suzy" created="2010-05-01 21:47:08" /
</dataset>
```

在整个时间长河中，只有特定的一秒钟内这个断言可以通过评定，在 *2010–05–01 21:47:08*。在数据库测试中，日期构成了一个特殊的问题。可以从这个断言中省略 “created” 列来规避失败。

为了让断言能得以通过， Flat XML 文件 *expectedBook.xml* 需要调整成大致类似这样：

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" />
    <guestbook id="2" content="I like it!" user="nancy" />
    <guestbook id="3" content="Hello world!" user="suzy" />
</dataset>
```

还得修正一下 QueryTable 的调用：

```
<?php
$queryTable = $this->getConnection()->createQueryTable(
    'guestbook', 'SELECT id, content, user FROM guestbook'
);
?>
```

对查询的结果作出断言

利用　QueryTable，也可以对复杂查询的结果作出断言，只需要指定查询以及结果名称，并随后将其与某个数据集进行比较：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class ComplexQueryTest extends TestCase
{
    use TestCaseTrait;

    public function testComplexQuery()
    {
        $queryTable = $this->getConnection()->createQueryTable(
            'myComplexQuery', 'SELECT complexQuery...'
        );
        $expectedTable = $this->createFlatXmlDataSet("complexQueryAssertion.xml")
                              ->getTable("myComplexQuery");
        $this->assertTablesEqual($expectedTable, $queryTable);
    }
}
?>
```

对多个表的状态作出断言

当然可以一次性对多个表的状态作出断言，并将查询数据集与基于文件的数据集进行比较。有两种不同的方式来进行数据集断言。

1. 可以从自数据库连接建立数据库数据集，并将其与基于文件的数据集进行比较。

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class DataSetAssertionsTest extends TestCase
{
    use TestCaseTrait;

    public function testCreateDataSetAssertion()
    {
        $dataSet = $this->getConnection()->createDataSet(['guestbook']);
        $expectedDataSet = $this->createFlatXmlDataSet('guestbook.xml');
        $this->assertDataSetsEqual($expectedDataSet, $dataSet);
    }
```

```
}
?>
```

2. 也可以自行构造数据集：

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\DbUnit\TestCaseTrait;

class DataSetAssertionsTest extends TestCase
{
    use TestCaseTrait;

    public function testManualDataSetAssertion()
    {
        $dataSet = new PHPUnit_Extensions_Database_DataSet_QueryDataSet();
        $dataSet->addTable('guestbook', 'SELECT id, content, user FROM guestbook'); //
        $expectedDataSet = $this->createFlatXmlDataSet('guestbook.xml');

        $this->assertDataSetsEqual($expectedDataSet, $dataSet);
    }
}
?>
```

常见问题（FAQ）

PHPUnit 会为每个测试（重新）创建数据库吗？

不，PHPUnit 要求在测试套件开始时所有数据库对象必须全部可用。数据库、表、序列、触发器还有视图，必须全部在运行测试套件之前创建好。

Doctrine 2 [http://www.doctrine-project.org] 或 eZ Components [http://www.ezcomponents.org] 拥有强力的工具，可以按预定义的数据结构创建数据库，但是这些都必须和 PHPUnit 扩展模块对接之后才能自动在整个测试套件运行之前重新创建数据库。

由于每个测试都会彻底清空数据库，因此无须为每个测试重新创建数据库。持久可用的数据库同样能够完美工作。

为了让数据库扩展模块正常工作，需要在应用程序中使用 PDO 吗？

不，只在基境的清理与建立阶段还有断言检定时用到PDO。在你的自有代码中，可以使用任意数据库抽象。

如果看到 “Too much Connections” 错误该怎么办？

如果没有对 TestCase 中 `getConnection()` 方法所创建 PDO 实例进行缓存，那么每个数据库测试都会增加一个或多个数据库连接。MySQL的默认配置只允许100个并发连接，其他供应商的数据库也都有各自的最大连接限制。

子章节 “使用你自己的抽象数据库 TestCase 类”展示了如何通过在所有测试中使用单个PDO实例缓存来防止发生此错误。

Flat XML / CSV 数据集中如何处理 NULL？

别这么干。应当改用 XML 或者 YAML 数据集。

第 9 章 测试替身

Gerard Meszaros 在 [Meszaros2007] 中介绍了测试替身的概念：

> 有时候对被测系统(SUT)进行测试是很困难的，因为它依赖于其他无法在测试环境中使用的组件。这有可能是因为这些组件不可用，它们不会返回测试所需要的结果，或者执行它们会有不良副作用。在其他情况下，我们的测试策略要求对被测系统的内部行为有更多控制或更多可见性。
>
> 如果在编写测试时无法使用（或选择不使用）实际的依赖组件(DOC)，可以用测试替身来代替。测试替身不需要和真正的依赖组件有完全一样的的行为方式；他只需要提供和真正的组件同样的 API 即可，这样被测系统就会以为它是真正的组件！
>
> —Gerard Meszaros

PHPUnit 提供的 `createMock($type)` 和 `getMockBuilder($type)` 方法可以在测试中用来自动生成对象，此对象可以充当任意指定原版类型（接口或类名）的测试替身。在任何预期或要求使用原版类的实例对象的上下文中都可以使用这个测试替身对象来代替。

`createMock($type)` 方法直接返回指定类型（接口或类）的测试替身对象实例。此测试替身的创建使用了最佳实践的默认值（不执行原始类的 `__construct()` 和 `__clone()` 方法，且不对传递给测试替身的方法的参数进行克隆）。如果这些默认值非你所需，可以用 `getMockBuilder($type)` 方法并使用流畅式接口来定制测试替身的生成过程。

在默认情况下，原版类的所有方法都会被替换为只会返回 `null` 的伪实现（其中不会调用原版方法）。使用诸如 `will($this->returnValue())` 之类的方法可以对这些伪实现在被调用时应当返回什么值做出配置。

局限性：final、private、与 static 方法

请注意，`final`、`private` 和 `static` 方法无法对其进行上桩(stub)或模仿(mock)。PHPUnit 的测试替身功能将会忽略它们，并维持它们的原始行为。

Stubs（桩件）

将对象替换为（可选地）返回配置好的返回值的测试替身的实践方法称为上桩*(stubbing)*。可以用桩件*(stub)*来“替换掉被测系统所依赖的实际组件，这样测试就有了对被测系统的间接输入的控制点。这使得测试能强制安排被测系统的执行路径，否则被测系统可能无法执行”。

例 9.2 “对某个方法的调用上桩，返回固定值”展示了如何对方法的调用上桩以及如何设定返回值。首先用 `PHPUnit\Framework\TestCase` 类提供的 `createMock()` 方法来建立一个桩件对象，它表面看起来像是 `SomeClass`类（例 9.1 “需要对其上桩的类”）的实例。随后用 PHPUnit 提供的 流畅式接口 [http://martinfowler.com/bliki/FluentInterface.html]来指定桩件的行为。本质上，这意味着不需要建立多个临时对象然后再把它们捆到一起。取而代之的是范例中所示的链式方法调用。这使得代码更加易读并更加“流畅”。

例 9.1. 需要对其上桩的类

```
<?php
use PHPUnit\Framework\TestCase;

class SomeClass
{
    public function doSomething()
    {
        // 随便做点什么。
    }
```

```
}
?>
```

例 9.2. 对某个方法的调用上桩，返回固定值

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testStub()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->willReturn('foo');

        // 现在调用 $stub->doSomething() 将返回 'foo'。
        $this->assertEquals('foo', $stub->doSomething());
    }
}
?>
```

局限性：名字为“method”的方法

仅当原始类中不包含名字为“method”的方法时，以上范例才能正常运行。

如果原始类包含名为“method”的方法，就必须用 `$stub->expects($this->any())->method('doSomething')->willReturn('foo');`。

“在幕后”，当使用了 `createMock()` 方法时， PHPUnit 自动生成了一个新的 PHP 类来实现想要的行为。

例 9.3 “使用可用于配置生成的测试替身类的仿件生成器 API”这个例子展示了如何用仿件生成器的流畅式接口来配置测试替身的生成。这个测试替身的默认配置用的是和 `createMock()` 相同的最佳实践。

例 9.3. 使用可用于配置生成的测试替身类的仿件生成器 API

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testStub()
    {
        // 为 SomeClass 类建立桩件。
        $stub = $this->getMockBuilder($originalClassName)
                     ->disableOriginalConstructor()
                     ->disableOriginalClone()
                     ->disableArgumentCloning()
                     ->disallowMockingUnknownTypes()
                     ->getMock();

        // 配置桩件。
        $stub->method('doSomething')
             ->willReturn('foo');

        // 现在调用 $stub->doSomething() 将返回 'foo'。
```

```
        $this->assertEquals('foo', $stub->doSomething());
    }
}
?>
```

在之前的例子中，用 `willReturn($value)` 返回简单值。这个简短的语法相当于 `will($this->returnValue($value))`。而在这个长点的语法中，可以使用变量，从而实现更复杂的上桩行为。

有时想要将（未改变的）方法调用时所使用的参数之一作为桩件的方法的调用结果来返回。例 9.4 “对某个方法的调用上桩，返回参数之一”展示了如何用 `returnArgument()` 代替 `returnValue()` 来做到这点。

例 9.4. 对某个方法的调用上桩，返回参数之一

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testReturnArgumentStub()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->returnArgument(0));

        // $stub->doSomething('foo') 返回 'foo'
        $this->assertEquals('foo', $stub->doSomething('foo'));

        // $stub->doSomething('bar') 返回 'bar'
        $this->assertEquals('bar', $stub->doSomething('bar'));
    }
}
?>
```

在用流畅式接口进行测试时，让某个已上桩的方法返回对桩件对象的引用有时会很有用。例 9.5 “对方法的调用上桩，返回对桩件对象的引用”展示了如何用 `returnSelf()` 来做到这点。

例 9.5. 对方法的调用上桩，返回对桩件对象的引用

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testReturnSelf()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->returnSelf());

        // $stub->doSomething() 返回 $stub
        $this->assertSame($stub, $stub->doSomething());
    }
```

```
}
?>
```

有时候，上桩的方法需要根据预定义的参数清单来返回不同的值。可以用 `returnValueMap()` 方法将参数和相应的返回值关联起来建立映射。范例参见例 9.6 “对方法的调用上桩，按照映射确定返回值”。

例 9.6. 对方法的调用上桩，按照映射确定返回值

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testReturnValueMapStub()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 创建从参数到返回值的映射。
        $map = [
            ['a', 'b', 'c', 'd'],
            ['e', 'f', 'g', 'h']
        ];

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->returnValueMap($map));

        // $stub->doSomething() 根据提供的参数返回不同的值。
        $this->assertEquals('d', $stub->doSomething('a', 'b', 'c'));
        $this->assertEquals('h', $stub->doSomething('e', 'f', 'g'));
    }
}
?>
```

如果上桩的方法需要返回计算得到的值而不是固定值（参见 `returnValue()`）或某个（未改变的）参数（参见 `returnArgument()`），可以用 `returnCallback()` 来让上桩的方法返回回调函数或方法的结果。范例参见例 9.7 “对方法的调用上桩，由回调生成返回值”。

例 9.7. 对方法的调用上桩，由回调生成返回值

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testReturnCallbackStub()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->returnCallback('str_rot13'));

        // $stub->doSomething($argument) 返回 str_rot13($argument)
        $this->assertEquals('fbzrguvat', $stub->doSomething('something'));
    }
}
```

```
?>
```

相比于建立回调方法，有一个更简单的选择是直接给出期望返回值的列表。可以用 onConsecutiveCalls() 方法来做到这个。范例参见 例 9.8 “对方法的调用上桩，按照指定顺序返回列表中的值”。

例 9.8. 对方法的调用上桩，按照指定顺序返回列表中的值

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testOnConsecutiveCallsStub()
    {
        // 为 SomeClass 类创建桩件。
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->onConsecutiveCalls(2, 3, 5, 7));

        // $stub->doSomething() 每次返回值都不同
        $this->assertEquals(2, $stub->doSomething());
        $this->assertEquals(3, $stub->doSomething());
        $this->assertEquals(5, $stub->doSomething());
    }
}
?>
```

除了返回一个值之外，上桩的方法还能抛出一个异常。例 9.9 “对方法的调用上桩，抛出异常”展示了如何用 throwException() 做到这点。

例 9.9. 对方法的调用上桩，抛出异常

```
<?php
use PHPUnit\Framework\TestCase;

class StubTest extends TestCase
{
    public function testThrowExceptionStub()
    {
        // 为 SomeClass 类创建桩件
        $stub = $this->createMock(SomeClass::class);

        // 配置桩件。
        $stub->method('doSomething')
             ->will($this->throwException(new Exception));

        // $stub->doSomething() 抛出异常
        $stub->doSomething();
    }
}
?>
```

另外，也可以自行编写桩件，并在此过程中改善设计。在系统中被广泛使用的资源是通过单个外观(facade)来访问的，因此很容易就能用桩件替换掉资源。例如，将散落在代码各处的对数据库的直接调用替换为单个 Database 对象，这个对象实现了 IDatabase 接口。接下来，就可以创建实现了 IDatabase 的桩件并在测试中使用之。甚至可以创建一个选项来控制是用桩件还是用真实数据库来运行测试，这样测试就既能在开发过程中用作本地测试，又能在实际数据库环境中进行集成测试。

需要上桩的功能往往集中在同一个对象中，这就改善了内聚度。将功能通过单一且一致的接口呈现出来，就降低了这部分与系统其他部分之间的耦合度。

仿件对象(Mock Object)

将对象替换为能验证预期行为（例如断言某个方法必会被调用）的测试替身的实践方法称为模仿*(mocking)*。

可以用 仿件对象*(mock object)*“作为观察点来核实被测试系统在测试中的间接输出。通常，仿件对象还需要包括桩件的功能，因为如果测试尚未失败则仿件对象需要向被测系统返回一些值，但是其重点还是在对间接输出的核实上。因此，仿件对象远不止是桩件加断言，它是以一种从根本上完全不同的方式来使用的”(Gerard Meszaros)。

局限性：对预期的自动校验

PHPUnit只会对在某个测试的作用域内生成的仿件对象进行自动校验。诸如在数据供给器内生成或用@depends 标注注入测试的仿件对象，PHPUnit并不会自动对其进行校验。

这有个例子：假设需要测试的当前方法，在例子中是 `update()`，确实在一个观察着另外一个对象的对象中上被调用了。例 9.10 “被测系统(SUT)中 Subject 与 Observer 类的代码”展示了被测系统(SUT)中 `Subject` 和 `Observer` 两个类的代码。

例 9.10. 被测系统(SUT)中 Subject 与 Observer 类的代码

```
<?php
use PHPUnit\Framework\TestCase;

class Subject
{
    protected $observers = [];
    protected $name;

    public function __construct($name)
    {
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }

    public function attach(Observer $observer)
    {
        $this->observers[] = $observer;
    }

    public function doSomething()
    {
        // 做点什么
        // ...

        // 通知观察者发生了些什么
        $this->notify('something');
    }

    public function doSomethingBad()
    {
        foreach ($this->observers as $observer) {
            $observer->reportError(42, 'Something bad happened', $this);
```

```
        }
    }

    protected function notify($argument)
    {
        foreach ($this->observers as $observer) {
            $observer->update($argument);
        }
    }

    // 其他方法。
}

class Observer
{
    public function update($argument)
    {
        // 做点什么。
    }

    public function reportError($errorCode, $errorMessage, Subject $subject)
    {
        // 做点什么。
    }

    // 其他方法。
}
?>
```

例 9.11 “测试某个方法会以特定参数被调用一次”展示了如何用仿件对象来测试 Subject 和 Observer 对象之间的互动。

首先用 PHPUnit\Framework\TestCase 类提供的 getMockBuilder() 方法建立 Observer 的仿件对象。由于给出了一个数组做为 getMock() 方法的第二（可选）参数，Observer 类只有 update() 方法会被替换为仿实现。

由于关注的是检验某个方法是否被调用，以及调用时具体所使用的参数，因此引入 expects() 与 with() 方法来指明此交互应该是什么样的。

例 9.11. 测试某个方法会以特定参数被调用一次

```
<?php
use PHPUnit\Framework\TestCase;

class SubjectTest extends TestCase
{
    public function testObserversAreUpdated()
    {
        // 为 Observer 类建立仿件对象，只模仿 update() 方法。
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['update'])
                         ->getMock();

        // 建立预期状况：update() 方法将会被调用一次，
        // 并且将以字符串 'something' 为参数。
        $observer->expects($this->once())
                 ->method('update')
                 ->with($this->equalTo('something'));

        // 创建 Subject 对象，并将模仿的 Observer 对象连接其上。
        $subject = new Subject('My subject');
        $subject->attach($observer);
```

```
        // 在 $subject 对象上调用 doSomething() 方法，
        // 预期将以字符串 'something' 为参数调用
        // Observer 仿件对象的 update() 方法。
        $subject->doSomething();
    }
}
?>
```

with() 方法可以携带任何数量的参数，对应于被模仿的方法的参数数量。可以对方法的参数指定更加高等的约束而不仅是简单的匹配。

例 9.12. 测试某个方法将会以特定数量的参数进行调用，并且对各个参数以多种方式进行约束

```
<?php
use PHPUnit\Framework\TestCase;

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // 为 Observer 类建立仿件，对 reportError() 方法进行模仿
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['reportError'])
                         ->getMock();

        $observer->expects($this->once())
                 ->method('reportError')
                 ->with(
                       $this->greaterThan(0),
                       $this->stringContains('Something'),
                       $this->anything()
                   );

        $subject = new Subject('My subject');
        $subject->attach($observer);

        // doSomethingBad() 方法应当会通过（observer的）reportError()方法
        // 向 observer 报告错误。
        $subject->doSomethingBad();
    }
}
?>
```

withConsecutive() 方法可以接受任意多个数组作为参数，具体数量取决于欲测试的调用。每个数组都都是对被仿方法的相应参数的一组约束，就像 with() 中那样。

例 9.13. 测试某个方法将会以特定参数被调用二次

```
<?php
use PHPUnit\Framework\TestCase;

class FooTest extends TestCase
{
    public function testFunctionCalledTwoTimesWithSpecificArguments()
    {
        $mock = $this->getMockBuilder(stdClass::class)
                     ->setMethods(['set'])
                     ->getMock();

        $mock->expects($this->exactly(2))
             ->method('set')
```

```
            ->withConsecutive(
                [$this->equalTo('foo'), $this->greaterThan(0)],
                [$this->equalTo('bar'), $this->greaterThan(0)]
            );

        $mock->set('foo', 21);
        $mock->set('bar', 48);
    }
}
?>
```

callback() 约束用来进行更加复杂的参数校验。此约束的唯一参数是一个 PHP 回调项 (callback)。此 PHP 回调项接受需要校验的参数作为其唯一参数，并应当在参数通过校验时返回 true，否则返回 false。

例 9.14. 更加复杂的参数校验

```
<?php
use PHPUnit\Framework\TestCase;

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // 为 Observer 类建立仿件，模仿 reportError() 方法
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['reportError'])
                         ->getMock();

        $observer->expects($this->once())
                 ->method('reportError')
                 ->with($this->greaterThan(0),
                        $this->stringContains('Something'),
                        $this->callback(function($subject){
                          return is_callable([$subject, 'getName']) &&
                                 $subject->getName() == 'My subject';
                        }));

        $subject = new Subject('My subject');
        $subject->attach($observer);

        // doSomethingBad() 方法应当会通过（observer的）reportError()方法
        // 向 observer 报告错误。
        $subject->doSomethingBad();
    }
}
?>
```

例 9.15. 测试某个方法将会被调用一次，并且以某个特定对象作为参数。

```
<?php
use PHPUnit\Framework\TestCase;

class FooTest extends TestCase
{
    public function testIdenticalObjectPassed()
    {
        $expectedObject = new stdClass;

        $mock = $this->getMockBuilder(stdClass::class)
                     ->setMethods(['foo'])
                     ->getMock();
```

```
        $mock->expects($this->once())
             ->method('foo')
             ->with($this->identicalTo($expectedObject));

        $mock->foo($expectedObject);
    }
}
?>
```

例 9.16. 创建仿件对象时启用参数克隆

```
<?php
use PHPUnit\Framework\TestCase;

class FooTest extends TestCase
{
    public function testIdenticalObjectPassed()
    {
        $cloneArguments = true;

        $mock = $this->getMockBuilder(stdClass::class)
                     ->enableArgumentCloning()
                     ->getMock();

        // 现在仿件将对参数进行克隆，因此 identicalTo 约束将会失败。
    }
}
?>
```

表 A.1 “约束条件”列出了可以应用于方法参数的各种约束，表 9.1 “匹配器”列出了可以用于指定调用次数的各种匹配器。

表 9.1. 匹配器

匹配器	含义
`PHPUnit_ Framework_ MockObject_ Matcher_ AnyInvokedCount any()`	返回一个匹配器，当被评定的方法执行0次或更多次（即任意次数）时匹配成功。
`PHPUnit_ Framework_ MockObject_ Matcher_ InvokedCount never()`	返回一个匹配器，当被评定的方法从未执行时匹配成功。
`PHPUnit_ Framework_ MockObject_ Matcher_ InvokedAtLeastOnce atLeastOnce()`	返回一个匹配器，当被评定的方法执行至少一次时匹配成功。
`PHPUnit_ Framework_ MockObject_ Matcher_ InvokedCount once()`	返回一个匹配器，当被评定的方法执行恰好一次时匹配成功。
`PHPUnit_ Framework_ MockObject_ Matcher_ InvokedCount exactly(int $count)`	返回一个匹配器，当被评定的方法执行恰好 `$count` 次时匹配成功。
`PHPUnit_ Framework_ MockObject_ Matcher_ InvokedAtIndex at(int $index)`	返回一个匹配器，当被评定的方法是第 `$index` 个执行的方法时匹配成功。

注意

`at()` 匹配器的 `$index` 参数指的是对给定仿件对象的所有方法的调用的索引，从零开始。使用这个匹配器要谨慎，因为它可能导致测试由于与具体的实现细节过分紧密绑定而变得脆弱。

如一开始提到的，如果 createMock() 方法在生成测试替身时所使用的默认值不符合你的要求，则可以通过 getMockBuilder($type) 方法来用流畅式接口定制测试替身的生成过程。以下是仿件生成器所提供的方法列表：

- setMethods(array $methods) 可以在仿件生成器对象上调用，来指定哪些方法将被替换为可配置的测试替身。其他方法的行为不会有所改变。如果调用 setMethods(null)，那么没有方法会被替换。
- setConstructorArgs(array $args) 可用于向原版类的构造函数（默认情况下不会被替换为伪实现）提供参数数组。
- setMockClassName($name) 可用于指定生成的测试替身类的类名。
- disableOriginalConstructor() 参数可用于禁用对原版类的构造方法的调用。
- disableOriginalClone() 可用于禁用对原版类的克隆方法的调用。
- disableAutoload()可用于在测试替身类的生成期间禁用 __autoload()。

Prophecy

Prophecy [https://github.com/phpspec/prophecy] 是个“极为自我却又非常强大且灵活的 PHP 对象模仿框架。虽然一开始是为了满足 phpspec2 的需要而建立的，但它足够灵活，可以用最小代价用于任何测试框架内。”

PHPUnit 对用 Prophecy 建立测试替身提供了内建支持。例 9.17 “测试某个方法会以特定参数被调用一次”展示了例 9.11 “测试某个方法会以特定参数被调用一次”中展示的测试应该如何用 Prophecy 的的预言式理念方式来达到同样的效果：

例 9.17. 测试某个方法会以特定参数被调用一次

```
<?php
use PHPUnit\Framework\TestCase;

class SubjectTest extends TestCase
{
    public function testObserversAreUpdated()
    {
        $subject = new Subject('My subject');

        // 为 Observer 类建立预言(prophecy)。
        $observer = $this->prophesize(Observer::class);

        // 建立预期状况：update() 方法将会被调用一次，
        // 并且将以字符串 'something' 为参数。
        $observer->update('something')->shouldBeCalled();

        // 揭示预言，并将仿件对象链接到主体上。
        $subject->attach($observer->reveal());

        // 在 $subject 对象上调用 doSomething() 方法，
        // 预期将以字符串 'something' 为参数调用
        // Observer 仿件对象的 update() 方法。
        $subject->doSomething();
    }
}
?>
```

更多关于如何用这个测试替身框架来创建、配置及使用桩件、谍件、仿件的细节，请参考 Prophecy 的 文档 [https://github.com/phpspec/prophecy#how-to-use-it]。

对特质(Trait)与抽象类进行模仿

`getMockForTrait()` 方法返回一个使用了特定特质(trait)的仿件对象。给定特质的所有抽象方法将都被模仿。这样就能对特质的具体方法进行测试。

例 9.18. 对特质的具体方法进行测试

```
<?php
use PHPUnit\Framework\TestCase;

trait AbstractTrait
{
    public function concreteMethod()
    {
        return $this->abstractMethod();
    }

    public abstract function abstractMethod();
}

class TraitClassTest extends TestCase
{
    public function testConcreteMethod()
    {
        $mock = $this->getMockForTrait(AbstractTrait::class);

        $mock->expects($this->any())
             ->method('abstractMethod')
             ->will($this->returnValue(true));

        $this->assertTrue($mock->concreteMethod());
    }
}
?>
```

`getMockForAbstractClass()` 方法返回一个抽象类的仿件对象。给定抽象类的所有抽象方法将都被模仿。这样就能对抽象类的具体方法进行测试。

例 9.19. 对抽象类的具体方法进行测试

```
<?php
use PHPUnit\Framework\TestCase;

abstract class AbstractClass
{
    public function concreteMethod()
    {
        return $this->abstractMethod();
    }

    public abstract function abstractMethod();
}

class AbstractClassTest extends TestCase
{
    public function testConcreteMethod()
    {
        $stub = $this->getMockForAbstractClass(AbstractClass::class);

        $stub->expects($this->any())
             ->method('abstractMethod')
             ->will($this->returnValue(true));
```

```
        $this->assertTrue($stub->concreteMethod());
    }
}
?>
```

对 Web 服务(Web Services)进行上桩或模仿

当应用程序需要和 web 服务进行交互时，会想要在不与 web 服务进行实际交互的情况下对其进行测试。为了简单地对 web 服务进行上桩或模仿，可以像使用 `getMock()`（见上文）那样使用 `getMockFromWsdl()`。唯一的区别是 `getMockFromWsdl()` 所返回的桩件或者仿件是基于以 WSDL 描述的 web 服务，而 `getMock()` 返回的桩件或者仿件是基于 PHP 类或接口的。

例 9.20 “对 web 服务上桩”展示了如何用 `getMockFromWsdl()` 来对（例如）`GoogleSearch.wsdl` 中描述的 web 服务上桩。

例 9.20. 对 web 服务上桩

```
<?php
use PHPUnit\Framework\TestCase;

class GoogleTest extends TestCase
{
    public function testSearch()
    {
        $googleSearch = $this->getMockFromWsdl(
          'GoogleSearch.wsdl', 'GoogleSearch'
        );

        $directoryCategory = new stdClass;
        $directoryCategory->fullViewableName = '';
        $directoryCategory->specialEncoding = '';

        $element = new stdClass;
        $element->summary = '';
        $element->URL = 'https://phpunit.de/';
        $element->snippet = '...';
        $element->title = '<b>PHPUnit</b>';
        $element->cachedSize = '11k';
        $element->relatedInformationPresent = true;
        $element->hostName = 'phpunit.de';
        $element->directoryCategory = $directoryCategory;
        $element->directoryTitle = '';

        $result = new stdClass;
        $result->documentFiltering = false;
        $result->searchComments = '';
        $result->estimatedTotalResultsCount = 3.9000;
        $result->estimateIsExact = false;
        $result->resultElements = [$element];
        $result->searchQuery = 'PHPUnit';
        $result->startIndex = 1;
        $result->endIndex = 1;
        $result->searchTips = '';
        $result->directoryCategories = [];
        $result->searchTime = 0.248822;

        $googleSearch->expects($this->any())
                     ->method('doGoogleSearch')
                     ->will($this->returnValue($result));
```

```
        /**
         * $googleSearch->doGoogleSearch() 将会返回上桩的结果，
         * web 服务的 doGoogleSearch() 方法不会被调用。
         */
        $this->assertEquals(
          $result,
          $googleSearch->doGoogleSearch(
            '000000000000000000000000000000000',
            'PHPUnit',
            0,
            1,
            false,
            '',
            false,
            '',
            '',
            ''
          )
        );
    }
}
?>
```

对文件系统进行模仿

vfsStream [https://github.com/mikey179/vfsStream] 是对虚拟文件系统 [http://en.wikipedia.org/wiki/Virtual_file_system] 的 流包覆器(stream wrapper) [http://www.php.net/streams]，可以用于模仿真实文件系统，在单元测试中可能会有所助益。

如果使用 Composer [https://getcomposer.org/] 来管理项目的依赖关系，那么只需简单的在项目的 `composer.json` 文件中加一条对 `mikey179/vfsStream` 的依赖关系即可。以下是一个最小化的 `composer.json`文件例子，只定义了一条对 PHPUnit 4.6 与 vfsStream 的开发时(development-time)依赖：

```
{
    "require-dev": {
        "phpunit/phpunit": "~4.6",
        "mikey179/vfsStream": "~1"
    }
}
```

例 9.21 “一个与文件系统交互的类”展示了一个与文件系统交互的类。

例 9.21. 一个与文件系统交互的类

```
<?php
use PHPUnit\Framework\TestCase;

class Example
{
    protected $id;
    protected $directory;

    public function __construct($id)
    {
        $this->id = $id;
    }

    public function setDirectory($directory)
    {
        $this->directory = $directory . DIRECTORY_SEPARATOR . $this->id;
```

```
        if (!file_exists($this->directory)) {
            mkdir($this->directory, 0700, true);
        }
    }
}?>
```

如果不使用诸如 vfsStream 这样的虚拟文件系统，就无法在隔离外部影响的情况下对 `setDirectory()` 方法进行测试（参见 例 9.22 “对一个与文件系统交互的类进行测试”）。

例 9.22. 对一个与文件系统交互的类进行测试

```
<?php
use PHPUnit\Framework\TestCase;

class ExampleTest extends TestCase
{
    protected function setUp()
    {
        if (file_exists(dirname(__FILE__) . '/id')) {
            rmdir(dirname(__FILE__) . '/id');
        }
    }

    public function testDirectoryIsCreated()
    {
        $example = new Example('id');
        $this->assertFalse(file_exists(dirname(__FILE__) . '/id'));

        $example->setDirectory(dirname(__FILE__));
        $this->assertTrue(file_exists(dirname(__FILE__) . '/id'));
    }

    protected function tearDown()
    {
        if (file_exists(dirname(__FILE__) . '/id')) {
            rmdir(dirname(__FILE__) . '/id');
        }
    }
}
?>
```

上面的方法有几个缺点：

- 和任何其他外部资源一样，文件系统可能会间歇性的出现一些问题，这使得和它交互的测试变得不可靠。
- 在 `setUp()` 和 `tearDown()` 方法中，必须确保这个目录在测试前和测试后均不存在。
- 如果测试在 `tearDown()` 方法被调用之前就终止了，这个目录就会遗留在文件系统中。

例 9.23 “在对与文件系统交互的类进行的测试中模仿文件系统”展示了如何在对与文件系统交互的类进行的测试中使用 vfsStream 来模仿文件系统。

例 9.23. 在对与文件系统交互的类进行的测试中模仿文件系统

```
<?php
use PHPUnit\Framework\TestCase;

class ExampleTest extends TestCase
{
```

```
    public function setUp()
    {
        vfsStreamWrapper::register();
        vfsStreamWrapper::setRoot(new vfsStreamDirectory('exampleDir'));
    }

    public function testDirectoryIsCreated()
    {
        $example = new Example('id');
        $this->assertFalse(vfsStreamWrapper::getRoot()->hasChild('id'));

        $example->setDirectory(vfsStream::url('exampleDir'));
        $this->assertTrue(vfsStreamWrapper::getRoot()->hasChild('id'));
    }
}
?>
```

这有几个优点：

- 测试本身更加简洁。
- vfsStream 让开发者能够完全控制被测代码所处的文件系统环境。
- 由于文件系统操作不再对真实文件系统进行操作，`tearDown()` 方法中的清理操作不再需要了。

第 10 章 测试实践

> 你总能编写更多测试。但是很快就会发现，在所有想得出来的测试中只有很小一部分是真正有用的。需要编写的是那些觉得能运作但却失败或觉得必将失败但却成功的测试。另外一种思考方式是从成本/收益的关系上去考量。需要编写的是能够给出反馈信息的测试。
>
> —Erich Gamma

在开发过程中

当需要对软件的内部结构进行更改时，你实际上是要在不影响其可见行为的情况下让它更加容易理解、更加易于修改，测试套件对于安全地进行这些所谓的重构 [http://martinfowler.com/bliki/DefinitionOfRefactoring.html]而言是非常宝贵的。否则，你可能在重组过程中将系统搞坏而不自知。

在使用单元测试来确认重构的转换步骤中确实保持原有行为并且没有引入错误时，以下情况有助于改进项目的编码与设计：

1. 所有单元测试均正确运行。
2. 代码传达其设计原则。
3. 代码没有冗余。
4. 代码所包含的类和方法的数量降至最低。

当需要向系统内添加新的功能时，首先为其编写测试。然后，当测试能够正常运行就标志着开发完成了。下一章将详细讨论这种做法。

在调试过程中

当看到缺陷报告时，你可能会有尽快修复错误的冲动。经验表明，这种冲动不是好事，因为修复一个缺陷时很可能导致另外一个缺陷。

下列操作可以帮你压住冲动：

1. 确认能够重现此缺陷。
2. 在代码中寻找此缺陷的最小规模表达。例如，如果在输出中有一个数字看起来不对，那么就寻找算出此数字的那个对象。
3. 编写一个目前会失败而缺陷修复后将会成功的自动测试。
4. 修复缺陷。

寻找缺陷的最小可靠重现使你有机会去真正检查缺陷的原因。当修复了缺陷之后，所编写的测试则有助于提高缺陷真正被修复的几率，因为新加入的测试降低了未来修改代码时又破坏此修复的可能性。而之前所编写的所有测试则降低了在不经意间导致其他问题的可能性。

进行单元测试带来了很多好处：

- 进行测试让代码的作者和评审者对补丁能够产生正确的结果有信心。
- 编写测试用例对开发者而言是一种很好的发现边缘情况的原动力。
- 进行测试提供了一种良好的方法来快速捕捉退步(Regression)，并且能用来保证退步不会重复出现。

- 单元测试就如何使用 API 提供了可正常工作的范例，能够大大帮助文档编制工作。

总之，进行集成单元测试降低了任何修改的成本与风险。这使得项目能够更快并且更有信心地进行[...]重大架构改良[...]。

—Benjamin Smedberg

第 11 章 代码覆盖率分析

> 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。
>
> —Wikipedia

在本章中，你将学到 PHPUnit 中与代码覆盖率相关的一切功能。通过这部分功能，能够了解在测试运行过程中执行了生产代码的哪些部分。它使用了 PHP_CodeCoverage [https://github.com/sebastianbergmann/php-code-coverage] 组件，而这个组件又使用了 PHP 的 Xdebug [http://xdebug.org/] 扩展所提供的代码覆盖率功能。

注意

Xdebug 不随 PHPUnit 分发。如果在运行测试时收到了 Xdebug 扩展未加载的提示，就意味着 Xdebug 未安装或者未正确配置。在使用 PHPUnit 的代码覆盖率分析功能之前，需要阅读下 Xdebug 安装指南 [http://xdebug.org/docs/install]。

PHPUnit 可以生成基于 HTML 的代码覆盖率报告，同时也能生成好几种（Clover、Crap4J、PHPUnit）基于XML的代码覆盖率信息记录文件。代码覆盖率信息也能以文本格式提供（同时可以输出到STDOUT）或以PHP代码格式输出以供进一步处理。

第 3 章 命令行测试执行器中列出了各种控制代码覆盖率功能的命令行参数供参考，同时“Logging（日志记录）”一节中可以找到其他相关的配置信息。

用于代码覆盖率的软件衡量标准

目前存在多种软件衡量标准用于衡量代码覆盖率：

行覆盖率*(Line Coverage)*	行覆盖率*(Line Coverage)*按单个可执行行是否已执行到进行计量。
函数与方法覆盖率*(Function and Method Coverage)*	函数与方法覆盖率*(Function and Method Coverage)*按单个函数或方法是否已调用到进行计量。仅当函数或方法的所有可执行行全部已覆盖时 PHP_CodeCoverage 才将其视为已覆盖。
类与特质覆盖率*(Class and Trait Coverage)*	类与特质覆盖率*(Class and Trait Coverage)*按单个类或特质的所有方法是否全部已覆盖进行计量。仅当一个类或性状的所有方法全部已覆盖时 PHP_CodeCoverage 才将其视为已覆盖。
Opcode 覆盖率*(Opcode Coverage)*	*Opcode* 覆盖率按函数或方法对应的每条 opcode 在运行测试套件时是否执行到进行计量。一行（PHP的）代码通常会编译得到多条 opcode。进行行覆盖率计量时，只要其中任何一条 opcode 被执行就视为此行已覆盖。
分支覆盖率*(Branch Coverage)*	分支覆盖率*(Branch Coverage)*按控制结构的分支进行计量。测试套件运行时每个控制结构的布尔表达式求值为 `true` 和 `false` 各自计为一个分支。
路径覆盖率*(Path Coverage)*	路径覆盖率*(Path Coverage)*按测试套件运行时函数或者方法内部所经历的执行路径进行计量。一个执行路径指的是从进入函数或方法一直到离开的过程中经过各个分支的特定序列。
变更风险反模式*(CRAP)*指数*(Change Risk Anti-Patterns (CRAP) Index)*	变更风险反模式*(CRAP)*指数*(Change Risk Anti-Patterns (CRAP) Index)*是基于代码单元的圈复杂度(cyclomatic complexity)与代码覆盖率计算得出的。不太复杂并具有恰

当测试覆盖率的代码将得出较低的CRAP指数。可以通过编写测试或重构代码来降低其复杂性的方式来降低CRAP指数。

注意

目前 PHP_CodeCoverage 尚不支持 *Opcode*覆盖率、分支覆盖率 及 路径覆盖率。

将文件列入白名单

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。可以用命令行选项 `--whitelist` 或通过配置文件（参见“Whitelisting Files for Code Coverage”一节）来完成。

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（参见“Whitelisting Files for Code Coverage”一节）。这样可以把完全没有测试到的文件也一并包含到报告中。如果需要知道这些未被覆盖文件中有哪些行是可执行的，需要同时在 PHPUnit 的配置信息中设置 `processUncoveredFilesFromWhitelist="true"`（参见“Whitelisting Files for Code Coverage”一节）。

注意

请注意，当设置了 `processUncoveredFilesFromWhitelist="true"` 时将对源代码文件进行载入，这在某些情况下可能导致问题，比如，源代码文件包含有处于类或者函数作用域之外的代码。

略过代码块

有时，一些代码块是无法对其进行测试的，因此希望在代码覆盖率分析中忽略它们。在 PHPUnit 中可以用 `@codeCoverageIgnore`、`@codeCoverageIgnoreStart` 与 `@codeCoverageIgnoreEnd` 标注来做到这点，如例 11.1 “使用 `@codeCoverageIgnore`、`@codeCoverageIgnoreStart` 与 `@codeCoverageIgnoreEnd` 标注”中所示。

例 11.1. 使用 `@codeCoverageIgnore`、`@codeCoverageIgnoreStart` 与 `@codeCoverageIgnoreEnd` 标注

```
<?php
use PHPUnit\Framework\TestCase;

/**
 * @codeCoverageIgnore
 */
class Foo
{
    public function bar()
    {
    }
}

class Bar
{
    /**
     * @codeCoverageIgnore
     */
    public function foo()
    {
```

```
    }
}

if (false) {
    // @codeCoverageIgnoreStart
    print '*';
    // @codeCoverageIgnoreEnd
}

exit; // @codeCoverageIgnore
?>
```

代码中被忽略掉的行（用标注标记为忽略）将会计为已执行（如果它们是可执行的），并且不会在代码覆盖情况中被高亮标记。

指明要覆盖的方法

@covers 标注（参见 表 B.1 “用于指明测试覆盖哪些方法的标注”）可以用在测试代码中来指明测试方法想要对哪些方法进行测试。如果提供了这个信息，则只有指定方法的代码覆盖率信息会被统计。 例 11.2 “在测试中指明欲覆盖哪些方法”展示了一个例子。

例 11.2. 在测试中指明欲覆盖哪些方法

```
<?php
use PHPUnit\Framework\TestCase;

class BankAccountTest extends TestCase
{
    protected $ba;

    protected function setUp()
    {
        $this->ba = new BankAccount;
    }

    /**
     * @covers BankAccount::getBalance
     */
    public function testBalanceIsInitiallyZero()
    {
        $this->assertEquals(0, $this->ba->getBalance());
    }

    /**
     * @covers BankAccount::withdrawMoney
     */
    public function testBalanceCannotBecomeNegative()
    {
        try {
            $this->ba->withdrawMoney(1);
        }

        catch (BankAccountException $e) {
            $this->assertEquals(0, $this->ba->getBalance());

            return;
        }

        $this->fail();
    }

    /**
```

```
     * @covers BankAccount::depositMoney
     */
    public function testBalanceCannotBecomeNegative2()
    {
        try {
            $this->ba->depositMoney(-1);
        }

        catch (BankAccountException $e) {
            $this->assertEquals(0, $this->ba->getBalance());

            return;
        }

        $this->fail();
    }

    /**
     * @covers BankAccount::getBalance
     * @covers BankAccount::depositMoney
     * @covers BankAccount::withdrawMoney
     */
    public function testDepositWithdrawMoney()
    {
        $this->assertEquals(0, $this->ba->getBalance());
        $this->ba->depositMoney(1);
        $this->assertEquals(1, $this->ba->getBalance());
        $this->ba->withdrawMoney(1);
        $this->assertEquals(0, $this->ba->getBalance());
    }
}
?>
```

同时，可以用 @coversNothing 标注来指明一个测试不覆盖任何方法（参见“@coversNothing”一节）。这可以在编写集成测试时用于确保代码覆盖全部来自单元测试。

例 11.3. 指明测试不欲覆盖任何方法

```
<?php
use PHPUnit\Framework\TestCase;

class GuestbookIntegrationTest extends PHPUnit_Extensions_Database_TestCase
{
    /**
     * @coversNothing
     */
    public function testAddEntry()
    {
        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Hello world!");

        $queryTable = $this->getConnection()->createQueryTable(
            'guestbook', 'SELECT * FROM guestbook'
        );

        $expectedTable = $this->createFlatXmlDataSet("expectedBook.xml")
                              ->getTable("guestbook");

        $this->assertTablesEqual($expectedTable, $queryTable);
    }
}
?>
```

边缘情况

本节中展示了一些值得注意的边缘情况，在这些边缘情况中可能出现令人迷惑的代码覆盖率信息。

例 11.4.

```
<?php
use PHPUnit\Framework\TestCase;

// 因为覆盖率是“基于行”而不是基于语句的，
// 每行只会有一种覆盖状态
if (false) this_function_call_shows_up_as_covered();

// 由于代码覆盖率的内部工作方式，这两行显得很特殊。
// 这一行会显示为非可执行
if (false)
    // 这一行会显示为已覆盖，
    // 实际上是上一行的 if 语句的覆盖信息显示在这了！
    will_also_show_up_as_covered();

// 要避免这种情况，必须使用大括号
if (false) {
    this_call_will_never_show_up_as_covered();
}
?>
```

第 12 章 测试的其他用途

一旦习惯了编写自动测试，就可能会发现测试的更多用途。这有一些例子。

敏捷文档

通常，在使用了诸如极限编程之类的敏捷流程的项目中，文档往往无法跟上项目设计与代码的频繁变更。极限编程要求群体代码所有权*(collective code ownership)*，因此所有开发者都需要知道整个系统是如何工作的。如果你足够训练有素，为测试使用了“能说明问题的名称(speaking names)”来描述各个类应当干什么，那么就可以用 PHPUnit 的 TestDox 功能来基于项目的测试生成项目的自动文档。这个文档能够就项目中的各个类应当起什么作用给开发者一份概述。

PHPUnit 的 TestDox 功能着眼于测试类及其所有测试方法的名称，将它们驼峰式大小写（camel case）拼写的 PHP 名称转换为句子：`testBalanceIsInitiallyZero()` 转化为 "Balance is initially zero（初始结余为零）"。如果有多个测试方法的名字互相之间的差异只是一个或多个数字的后缀，例如 `testBalanceCannotBecomeNegative()` 和 `testBalanceCannotBecomeNegative2()`，假如所有这些测试都成功，句子"Balance cannot become negative（结余不能变为负数）"只会出现一次。

来看一下从 `BankAccount` 类生成的敏捷文档：

```
phpunit --testdox BankAccountTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

BankAccount
 [x] Balance is initially zero
 [x] Balance cannot become negative
```

另外，敏捷文档也可以以 HTML 或纯文本格式生成，并写入文件中，用 `--testdox-html` 和 `--testdox-text` 参数即可。

敏捷文档可以用于将对项目所使用的外部包所做出的假设文档化。使用外部包，你就暴露于这个包的行为与你所预期的不同的风险中，并且包的未来版本可能在你所不知道的情况下有微妙的改变并破坏你的代码。每次做出假设时就编写一个对应的测试可以处理这些风险。如果测试成功，那么假设就有效。如果所有的假设都通过测试来文档化，外部包在未来发布新版本就不会引起忧虑：如果测试成功，那么系统就应当能继续正常运作。

跨团队测试

一旦用测试将假设文档化，你就拥有了测试。包的提供者——你做假设的对象——对你的测试一无所知。如果打算与包的提供者有更亲密的关系，可以用测试来沟通与协调你的活动。

当你愿意和包的提供着协调你的活动时，你们可以共同编写测试。通过这样的方式，测试能够展现出尽可能多的假设。隐藏的假设是在给合作判死刑。利用测试，你精确的将对所提供的包的预期文档化。提供者在所有测试顺利运行时就知道包已经完整了。

通过使用桩件（参见本书前面关于“仿件对象”的那一章），你可以更好的与供应商解耦：供应商的工作就是让测试能够运行于包的实际实现上；你的工作则是让测试能够运行于你自己的代码上。在你拿到包的实际实现前，使用桩件对象。通过这种方式，两个团队可以互相独立的进行开发。

第 13 章 Logging（日志记录）

PHPUnit 可以生成几种类型的日志文件。

测试结果 (XML)

PHPUnit 所生成的测试结果 XML 日志文件是基于 JUnit task for Apache Ant [http://ant.apache.org/manual/Tasks/junit.html] 所使用的 XML 日志的。下面的例子展示了 `ArrayTest` 中的测试所生成的 XML 日志文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
  <testsuite name="ArrayTest"
             file="/home/sb/ArrayTest.php"
             tests="2"
             assertions="2"
             failures="0"
             errors="0"
             time="0.016030">
    <testcase name="testNewArrayIsEmpty"
              class="ArrayTest"
              file="/home/sb/ArrayTest.php"
              line="6"
              assertions="1"
              time="0.008044"/>
    <testcase name="testArrayContainsAnElement"
              class="ArrayTest"
              file="/home/sb/ArrayTest.php"
              line="15"
              assertions="1"
              time="0.007986"/>
  </testsuite>
</testsuites>
```

以下 XML 日志文件是由名为 `FailureErrorTest` 的测试用例类中的两个测试 `testFailure` 和 `testError` 所生成的，展示了失败和错误是如何表示的：

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuites>
  <testsuite name="FailureErrorTest"
             file="/home/sb/FailureErrorTest.php"
             tests="2"
             assertions="1"
             failures="1"
             errors="1"
             time="0.019744">
    <testcase name="testFailure"
              class="FailureErrorTest"
              file="/home/sb/FailureErrorTest.php"
              line="6"
              assertions="1"
              time="0.011456">
      <failure type="PHPUnit_Framework_ExpectationFailedException">
testFailure(FailureErrorTest)
Failed asserting that &lt;integer:2&gt; matches expected value &lt;integer:1&gt;.

/home/sb/FailureErrorTest.php:8
</failure>
    </testcase>
    <testcase name="testError"
              class="FailureErrorTest"
```

```
              file="/home/sb/FailureErrorTest.php"
              line="11"
              assertions="0"
              time="0.008288">
      <error type="Exception">testError(FailureErrorTest)
Exception:

/home/sb/FailureErrorTest.php:13
</error>
    </testcase>
  </testsuite>
</testsuites>
```

代码覆盖率 (XML)

PHPUnit 所生成的 XML 格式代码覆盖率信息日志记录不严格地基于 Clover [http://www.atlassian.com/software/clover/]. 所使用的 XML 日志的。下面的例子展示了 `BankAccountTest` 中的测试所生成的 XML 日志文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<coverage generated="1184835473" phpunit="3.6.0">
  <project name="BankAccountTest" timestamp="1184835473">
    <file name="/home/sb/BankAccount.php">
      <class name="BankAccountException">
        <metrics methods="0" coveredmethods="0" statements="0"
                 coveredstatements="0" elements="0" coveredelements="0"/>
      </class>
      <class name="BankAccount">
        <metrics methods="4" coveredmethods="4" statements="13"
                 coveredstatements="5" elements="17" coveredelements="9"/>
      </class>
      <line num="77" type="method" count="3"/>
      <line num="79" type="stmt" count="3"/>
      <line num="89" type="method" count="2"/>
      <line num="91" type="stmt" count="2"/>
      <line num="92" type="stmt" count="0"/>
      <line num="93" type="stmt" count="0"/>
      <line num="94" type="stmt" count="2"/>
      <line num="96" type="stmt" count="0"/>
      <line num="105" type="method" count="1"/>
      <line num="107" type="stmt" count="1"/>
      <line num="109" type="stmt" count="0"/>
      <line num="119" type="method" count="1"/>
      <line num="121" type="stmt" count="1"/>
      <line num="123" type="stmt" count="0"/>
      <metrics loc="126" ncloc="37" classes="2" methods="4" coveredmethods="4"
               statements="13" coveredstatements="5" elements="17"
               coveredelements="9"/>
    </file>
    <metrics files="1" loc="126" ncloc="37" classes="2" methods="4"
             coveredmethods="4" statements="13" coveredstatements="5"
             elements="17" coveredelements="9"/>
  </project>
</coverage>
```

代码覆盖率 (TEXT)

以易于常人了解(human-readable)的格式生成代码覆盖率，输出到命令行或保存成文本文件。这个输出格式旨在为工作于少量类时提供快捷的覆盖情况概览。对于更大的项目，这个输出有助于对项目的覆盖情况有一个快速的概览，或者配合 `--filter` 功能使用也会很有用。若从命令行调用并且写入到 `php://stdout`, `--colors` 设置会非常好用。从命令行调用

时，写入到标准输出是默认选项。默认情况下，只会显示至少有一行被覆盖的文件。这只能通过 XML 配置选项 `showUncoveredFiles` 来改变。参见 “Logging（日志记录）”一节。默认情况下，在详细报告中会显示所有文件以及它们的覆盖情况。这可以通过 XML 配置选项 `showOnlySummary` 来改变。.

第 14 章 扩展 PHPUnit

可以用多种方式对 PHPUnit 进行扩展，使编写测试更容易，以及对运行测试所得到的反馈进行定制。扩展 PHPUnit 时，一般从这些点入手：

PHPUnit\Framework\TestCase 的子类

将自定义的断言和工具方法写在 PHPUnit\Framework\TestCase 的一个抽象子类中，然后从这个抽象子类派生你的测试用例类。这是扩展 PHPUnit 的最容易的方法。

编写自定义断言

编写自定义断言时，最佳实践是遵循 PHPUnit 自有断言的实现方式。正如 例 14.1 “PHPUnit_Framework_Assert 类的 assertTrue() 与 isTrue() 方法”中所示，assertTrue() 方法只是对 isTrue() 和 assertThat() 方法的外包覆：isTrue() 创建了一个匹配器对象，将其传递给 assertThat() 进行评定。

例 14.1. PHPUnit_Framework_Assert 类的 assertTrue() 与 isTrue() 方法

```
<?php
use PHPUnit\Framework\TestCase;

abstract class PHPUnit_Framework_Assert
{
    // ...

    /**
     * 断言某个条件为真。
     *
     * @param  boolean $condition
     * @param  string  $message
     * @throws PHPUnit_Framework_AssertionFailedError
     */
    public static function assertTrue($condition, $message = '')
    {
        self::assertThat($condition, self::isTrue(), $message);
    }

    // ...

    /**
     * 返回一个 PHPUnit_Framework_Constraint_IsTrue 匹配器对象
     *
     * @return PHPUnit_Framework_Constraint_IsTrue
     * @since  Method available since Release 3.3.0
     */
    public static function isTrue()
    {
        return new PHPUnit_Framework_Constraint_IsTrue;
    }

    // ...
}?>
```

例 14.2 “PHPUnit_Framework_Constraint_IsTrue 类”展示了 PHPUnit_Framework_Constraint_IsTrue 是如何扩展针对匹配器对象（或约束）的抽象基类 PHPUnit_Framework_Constraint 的。

例 14.2. PHPUnit_Framework_Constraint_IsTrue 类

```
<?php
use PHPUnit\Framework\TestCase;

class PHPUnit_Framework_Constraint_IsTrue extends PHPUnit_Framework_Constraint
{
    /**
     * 对参数 $other 进行约束评定。如果符合约束，
     * 返回 TRUE，否则返回 FALSE。
     *
     * @param mixed $other Value or object to evaluate.
     * @return bool
     */
    public function matches($other)
    {
        return $other === true;
    }

    /**
     * 返回代表此约束的字符串。
     *
     * @return string
     */
    public function toString()
    {
        return 'is true';
    }
}?>
```

在实现 assertTrue() 和 isTrue() 方法及 PHPUnit_Framework_Constraint_IsTrue 类时所付出的努力带来了一些好处，assertThat() 能够自动负责起断言的评定与任务簿记（例如为了统计目的而对其进行计数）工作。此外，isTrue() 方法还可以在配置仿件对象时用来作为匹配器。

实现 PHPUnit\Framework\TestListener

例 14.3 “简单的测试监听器”展示了 PHPUnit\Framework\TestListener 接口的一个简单实现。

例 14.3. 简单的测试监听器

```
<?php
use PHPUnit\Framework\TestCase;
use PHPUnit\Framework\TestListener;

class SimpleTestListener implements TestListener
{
    public function addError(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Error while running test '%s'.\n", $test->getName());
    }

    public function addFailure(PHPUnit_Framework_Test $test, PHPUnit_Framework_Assertion
    {
        printf("Test '%s' failed.\n", $test->getName());
    }

    public function addIncompleteTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Test '%s' is incomplete.\n", $test->getName());
```

```
    }

    public function addRiskyTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Test '%s' is deemed risky.\n", $test->getName());
    }

    public function addSkippedTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Test '%s' has been skipped.\n", $test->getName());
    }

    public function startTest(PHPUnit_Framework_Test $test)
    {
        printf("Test '%s' started.\n", $test->getName());
    }

    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        printf("Test '%s' ended.\n", $test->getName());
    }

    public function startTestSuite(PHPUnit_Framework_TestSuite $suite)
    {
        printf("TestSuite '%s' started.\n", $suite->getName());
    }

    public function endTestSuite(PHPUnit_Framework_TestSuite $suite)
    {
        printf("TestSuite '%s' ended.\n", $suite->getName());
    }
}
?>
```

例 14.4 “使用测试监听器基类”展示了如何从抽象类 `PHPUnit_Framework_BaseTestListener` 派生子类，这个抽象类为所有接口方法提供了空白实现，这样你就只需要指定那些在你的使用情境下有意义的接口方法。

例 14.4. 使用测试监听器基类

```
<?php
use PHPUnit\Framework\TestCase;

class ShortTestListener extends PHPUnit_Framework_BaseTestListener
{
    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        printf("Test '%s' ended.\n", $test->getName());
    }
}
?>
```

在“测试监听器”一节中可以看到如何配置 PHPUnit 来将测试监听器附加到测试执行过程上。

从 PHPUnit_Extensions_TestDecorator 派生子类

可以将测试用例或者测试套件包装在 `PHPUnit_Extensions_TestDecorator` 的子类中并运用 Decorator（修饰器）设计模式来在测试运行前后执行一些动作。

PHPUnit 了包含了一个具体的测试修饰器：PHPUnit_Extensions_RepeatedTest。它用于重复运行某个测试，并且只在全部循环中都成功时计为成功。

例 14.5 “RepeatedTest 修饰器”展示了测试修饰器 PHPUnit_Extensions_RepeatedTest 的一个删减版本，用以说明如何编写你自己的测试修饰器。

例 14.5. RepeatedTest 修饰器

```
<?php
use PHPUnit\Framework\TestCase;

require_once 'PHPUnit/Extensions/TestDecorator.php';

class PHPUnit_Extensions_RepeatedTest extends PHPUnit_Extensions_TestDecorator
{
    private $timesRepeat = 1;

    public function __construct(PHPUnit_Framework_Test $test, $timesRepeat = 1)
    {
        parent::__construct($test);

        if (is_integer($timesRepeat) &&
            $timesRepeat >= 0) {
            $this->timesRepeat = $timesRepeat;
        }
    }

    public function count()
    {
        return $this->timesRepeat * $this->test->count();
    }

    public function run(PHPUnit_Framework_TestResult $result = null)
    {
        if ($result === null) {
            $result = $this->createResult();
        }

        for ($i = 0; $i < $this->timesRepeat && !$result->shouldStop(); $i++) {
            $this->test->run($result);
        }

        return $result;
    }
}
?>
```

实现 PHPUnit_Framework_Test

PHPUnit_Framework_Test 接口是比较狭义的，十分容易实现。举例来说，你可以自行为 PHPUnit_Framework_Test 编写一个类似于 PHPUnit\Framework\TestCase 的实现来运行数据驱动测试。

例 14.6 “一个数据驱动的测试”展示了一个数据驱动的测试用例类，对来自 CSV 文件内的值进行比较。这个文件内的每个行看起来类似于 foo;bar，第一个值是期望值，第二个值则是实际值。

例 14.6. 一个数据驱动的测试

```
<?php
use PHPUnit\Framework\TestCase;
```

```
class DataDrivenTest implements PHPUnit_Framework_Test
{
    private $lines;

    public function __construct($dataFile)
    {
        $this->lines = file($dataFile);
    }

    public function count()
    {
        return 1;
    }

    public function run(PHPUnit_Framework_TestResult $result = null)
    {
        if ($result === null) {
            $result = new PHPUnit_Framework_TestResult;
        }

        foreach ($this->lines as $line) {
            $result->startTest($this);
            PHP_Timer::start();
            $stopTime = null;

            list($expected, $actual) = explode(';', $line);

            try {
                PHPUnit_Framework_Assert::assertEquals(
                  trim($expected), trim($actual)
                );
            }

            catch (PHPUnit_Framework_AssertionFailedError $e) {
                $stopTime = PHP_Timer::stop();
                $result->addFailure($this, $e, $stopTime);
            }

            catch (Exception $e) {
                $stopTime = PHP_Timer::stop();
                $result->addError($this, $e, $stopTime);
            }

            if ($stopTime === null) {
                $stopTime = PHP_Timer::stop();
            }

            $result->endTest($this, $stopTime);
        }

        return $result;
    }
}

$test = new DataDrivenTest('data_file.csv');
$result = PHPUnit_TextUI_TestRunner::run($test);
?>
```

```
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds
```

```
There was 1 failure:

1) DataDrivenTest
Failed asserting that two strings are equal.
expected string <bar>
difference      <  x>
got string      <baz>
/home/sb/DataDrivenTest.php:32
/home/sb/DataDrivenTest.php:53

FAILURES!
Tests: 2, Failures: 1.
```

附录 A. 断言

本附录列举可用的各种断言方法。

断言方法的用法：静态 vs. 非静态

PHPUnit 的各个断言是在 PHPUnit\Framework\Assert 中实现的。PHPUnit\Framework\TestCase 则继承于 PHPUnit\Framework\Assert。

各个断言方法均声明为 static，可以从任何上下文以类似于 PHPUnit\Framework\Assert::assertTrue() 的方式调用，或者也可以用类似于 $this->assertTrue() 或 self::assertTrue() 的方式在扩展自 PHPUnit\Framework\TestCase 的类内调用。

实际上，只要（手工）包含了 PHPUnit 中的 src/Framework/Assert/Functions.php 源码文件，甚至可以在任何上下文中（甚至包括扩展自 PHPUnit\Framework\TestCase 的类中）以诸如 assertTrue() 这样的方式来调用全局函数封装。

有个常见的疑问——对于那些 PHPUnit 的新手尤甚——是究竟应该用诸如 $this->assertTrue() 还是诸如 self::assertTrue() 这样的形式来调用断言才是“正确的方式”？简而言之：没有正确方式。同时，也没有错误方式。这基本上是个人喜好问题。

对于大多数人而言，由于测试方法是在测试对象上调用，因此用 $this->assertTrue() 会“觉的更正确”。然而请记住断言方法是声明为 static 的，这使其可以（重）用于测试对象的作用域之外。最后，全局函数封装让开发者能再少打一些字（用 assertTrue() 代替 $this->assertTrue() 或者 self::assertTrue()）。

assertArrayHasKey()

```
assertArrayHasKey(mixed $key, array $array[, string $message = ''])
```

当 $array 不包含 $key 时报告错误，错误讯息由 $message 指定。

assertArrayNotHasKey() 是与之相反的断言，接受相同的参数。

例 A.1. assertArrayHasKey() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class ArrayHasKeyTest extends TestCase
{
    public function testFailure()
    {
        $this->assertArrayHasKey('foo', ['bar' => 'baz']);
    }
}
?>
```

```
phpunit ArrayHasKeyTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:
```

```
1) ArrayHasKeyTest::testFailure
Failed asserting that an array has the key 'foo'.

/home/sb/ArrayHasKeyTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertClassHasAttribute()

```
assertClassHasAttribute(string $attributeName, string $className[,
string $message = ''])
```

当 `$className::attributeName` 不存在时报告错误，错误讯息由 `$message` 指定。

`assertClassNotHasAttribute()` 是与之相反的断言，接受相同的参数。

例 A.2. assertClassHasAttribute() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class ClassHasAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertClassHasAttribute('foo', stdClass::class);
    }
}
?>
```

```
phpunit ClassHasAttributeTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClassHasAttributeTest::testFailure
Failed asserting that class "stdClass" has attribute "foo".

/home/sb/ClassHasAttributeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertArraySubset()

```
assertArraySubset(array $subset, array $array[, bool $strict = '',
string $message = ''])
```

当 `$array` 不包含 `$subset` 时报告错误，错误讯息由 `$message` 指定。

`$strict` 是一个标志，用于表明是否需要对数组中的对象进行全等判定。

例 A.3. assertArraySubset() 的用法

```php
<?php
```

```php
use PHPUnit\Framework\TestCase;

class ArraySubsetTest extends TestCase
{
    public function testFailure()
    {
        $this->assertArraySubset(['config' => ['key-a', 'key-b']], ['config' => ['key-a'
    }
}
?>
```

```
phpunit ArrayHasKeyTest
PHPUnit 4.4.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) Epilog\EpilogTest::testNoFollowOption
Failed asserting that an array has the subset Array &0 (
    'config' => Array &1 (
        0 => 'key-a'
        1 => 'key-b'
    )
).

/home/sb/ArraySubsetTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertClassHasStaticAttribute()

```
assertClassHasStaticAttribute(string $attributeName, string
$className[, string $message = ''])
```

当 `$className::attributeName` 不存在时报告错误，错误讯息由 `$message` 指定。

`assertClassNotHasStaticAttribute()` 是与之相反的断言，接受相同的参数。

例 A.4. assertClassHasStaticAttribute() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class ClassHasStaticAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertClassHasStaticAttribute('foo', stdClass::class);
    }
}
?>
```

```
phpunit ClassHasStaticAttributeTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F
```

```
Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClassHasStaticAttributeTest::testFailure
Failed asserting that class "stdClass" has static attribute "foo".

/home/sb/ClassHasStaticAttributeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertContains()

`assertContains(mixed $needle, Iterator|array $haystack[, string $message = ''])`

当 `$needle` 不是 `$haystack`的元素时报告错误，错误讯息由 `$message` 指定。

`assertNotContains()` 是与之相反的断言，接受相同的参数。

`assertAttributeContains()` 和 `assertAttributeNotContains()` 是便捷包装(convenience wrapper)，以某个类或对象的 `public`、`protected` 或 `private` 属性为搜索范围。

例 A.5. assertContains() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains(4, [1, 2, 3]);
    }
}
?>
```

```
phpunit ContainsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsTest::testFailure
Failed asserting that an array contains 4.

/home/sb/ContainsTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

`assertContains(string $needle, string $haystack[, string $message = '', boolean $ignoreCase = false])`

当 `$needle` 不是 `$haystack` 的子字符串时报告错误，错误讯息由 `$message` 指定。

如果 $ignoreCase 为 true，测试将按大小写不敏感的方式进行。

例 A.6. assertContains() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains('baz', 'foobar');
    }
}
?>
```

```
phpunit ContainsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsTest::testFailure
Failed asserting that 'foobar' contains "baz".

/home/sb/ContainsTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

例 A.7. 带有 $ignoreCase 参数的 assertContains() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains('foo', 'FooBar');
    }

    public function testOK()
    {
        $this->assertContains('foo', 'FooBar', '', true);
    }
}
?>
```

```
phpunit ContainsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F.

Time: 0 seconds, Memory: 2.75Mb

There was 1 failure:
```

```
1) ContainsTest::testFailure
Failed asserting that 'FooBar' contains "foo".

/home/sb/ContainsTest.php:6

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

assertContainsOnly()

`assertContainsOnly(string $type, Iterator|array $haystack[, boolean $isNativeType = null, string $message = ''])`

当 `$haystack` 并非仅包含类型为 `$type` 的变量时报告错误，错误讯息由 `$message` 指定。

`$isNativeType` 是一个标志，用来表明 `$type` 是否是原生 PHP 类型。

`assertNotContainsOnly()` 是与之相反的断言，并接受相同的参数。

`assertAttributeContainsOnly()` 和 `assertAttributeNotContainsOnly()` 是便捷包装(convenience wrapper)，以某个类或对象的 `public`、`protected` 或 `private` 属性为搜索范围。

例 A.8. assertContainsOnly() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class ContainsOnlyTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnly('string', ['1', '2', 3]);
    }
}
?>
```

```
phpunit ContainsOnlyTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsOnlyTest::testFailure
Failed asserting that Array (
    0 => '1'
    1 => '2'
    2 => 3
) contains only values of type "string".

/home/sb/ContainsOnlyTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertContainsOnlyInstancesOf()

```
assertContainsOnlyInstancesOf(string $classname, Traversable|array
$haystack[, string $message = ''])
```

当 $haystack 并非仅包含类 $classname 的实例时报告错误，错误讯息由 $message 指定。

例 A.9. assertContainsOnlyInstancesOf() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class ContainsOnlyInstancesOfTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnlyInstancesOf(
            Foo::class,
            [new Foo, new Bar, new Foo]
        );
    }
}
?>
```

```
phpunit ContainsOnlyInstancesOfTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsOnlyInstancesOfTest::testFailure
Failed asserting that Array ([0]=> Bar Object(...)) is an instance of class "Foo".

/home/sb/ContainsOnlyInstancesOfTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertCount()

```
assertCount($expectedCount, $haystack[, string $message = ''])
```

当 $haystack 中的元素数量不是 $expectedCount 时报告错误，错误讯息由 $message 指定。

assertNotCount() 是与之相反的断言，接受相同的参数。

例 A.10. assertCount() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class CountTest extends TestCase
{
    public function testFailure()
    {
        $this->assertCount(0, ['foo']);
    }
```

```
}
?>
```

```
phpunit CountTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) CountTest::testFailure
Failed asserting that actual size 1 matches expected size 0.

/home/sb/CountTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertDirectoryExists()

`assertDirectoryExists(string $directory[, string $message = ''])`

当 `$directory` 所指定的目录不存在时报告错误，错误讯息由 `$message` 指定。

`assertDirectoryNotExists()` 是与之相反的断言，并接受相同的参数。

例 A.11. assertDirectoryExists() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class DirectoryExistsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryExists('/path/to/directory');
    }
}
?>
```

```
phpunit DirectoryExistsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryExistsTest::testFailure
Failed asserting that directory "/path/to/directory" exists.

/home/sb/DirectoryExistsTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertDirectoryIsReadable()

```
assertDirectoryIsReadable(string $directory[, string $message = ''])
```

当 $directory 所指定的目录不是个目录或不可读时报告错误，错误讯息由 $message 指定。

assertDirectoryNotIsReadable() 是与之相反的断言，并接受相同的参数。

例 A.12. assertDirectoryIsReadable() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class DirectoryIsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryIsReadable('/path/to/directory');
    }
}
?>
```

```
phpunit DirectoryIsReadableTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryIsReadableTest::testFailure
Failed asserting that "/path/to/directory" is readable.

/home/sb/DirectoryIsReadableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertDirectoryIsWritable()

```
assertDirectoryIsWritable(string $directory[, string $message = ''])
```

当 $directory 所指定的目录不是个目录或不可写时报告错误，错误讯息由 $message 指定。

assertDirectoryNotIsWritable() 是与之相反的断言，并接受相同的参数。

例 A.13. assertDirectoryIsWritable() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class DirectoryIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryIsWritable('/path/to/directory');
    }
}
```

```
?>
```

```
phpunit DirectoryIsWritableTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryIsWritableTest::testFailure
Failed asserting that "/path/to/directory" is writable.

/home/sb/DirectoryIsWritableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertEmpty()

`assertEmpty(mixed $actual[, string $message = ''])`

当 `$actual` 非空时报告错误，错误讯息由 `$message` 指定。

`assertNotEmpty()` 是与之相反的断言，接受相同的参数。

`assertAttributeEmpty()` 和 `assertAttributeNotEmpty()` 是便捷包装(convenience wrapper)，可以应用于某个类或对象的某个 `public`、`protected` 或 `private` 属性。

例 A.14. assertEmpty() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class EmptyTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEmpty(['foo']);
    }
}
?>
```

```
phpunit EmptyTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) EmptyTest::testFailure
Failed asserting that an array is empty.

/home/sb/EmptyTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertEqualXMLStructure()

```
assertEqualXMLStructure(DOMElement $expectedElement, DOMElement
$actualElement[, boolean $checkAttributes = false, string $message
= ''])
```

当 `$actualElement` 中 DOMElement 的 XML 结构与 `$expectedElement` 中 DOMElement 的 XML 结构不相同时报告错误，错误讯息由 `$message` 指定。

例 A.15. assertEqualXMLStructure() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class EqualXMLStructureTest extends TestCase
{
    public function testFailureWithDifferentNodeNames()
    {
        $expected = new DOMElement('foo');
        $actual = new DOMElement('bar');

        $this->assertEqualXMLStructure($expected, $actual);
    }

    public function testFailureWithDifferentNodeAttributes()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo bar="true" />');

        $actual = new DOMDocument;
        $actual->loadXML('<foo/>');

        $this->assertEqualXMLStructure(
          $expected->firstChild, $actual->firstChild, true
        );
    }

    public function testFailureWithDifferentChildrenCount()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo><bar/><bar/><bar/></foo>');

        $actual = new DOMDocument;
        $actual->loadXML('<foo><bar/></foo>');

        $this->assertEqualXMLStructure(
          $expected->firstChild, $actual->firstChild
        );
    }

    public function testFailureWithDifferentChildren()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo><bar/><bar/><bar/></foo>');

        $actual = new DOMDocument;
        $actual->loadXML('<foo><baz/><baz/><baz/></foo>');

        $this->assertEqualXMLStructure(
          $expected->firstChild, $actual->firstChild
        );
    }
}
```

```
?>
```

```
phpunit EqualXMLStructureTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

FFFF

Time: 0 seconds, Memory: 5.75Mb

There were 4 failures:

1) EqualXMLStructureTest::testFailureWithDifferentNodeNames
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'foo'
+'bar'

/home/sb/EqualXMLStructureTest.php:9

2) EqualXMLStructureTest::testFailureWithDifferentNodeAttributes
Number of attributes on node "foo" does not match
Failed asserting that 0 matches expected 1.

/home/sb/EqualXMLStructureTest.php:22

3) EqualXMLStructureTest::testFailureWithDifferentChildrenCount
Number of child nodes of "foo" differs
Failed asserting that 1 matches expected 3.

/home/sb/EqualXMLStructureTest.php:35

4) EqualXMLStructureTest::testFailureWithDifferentChildren
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/EqualXMLStructureTest.php:48

FAILURES!
Tests: 4, Assertions: 8, Failures: 4.
```

assertEquals()

`assertEquals(mixed $expected, mixed $actual[, string $message = ''])`

当两个变量 `$expected` 和 `$actual` 不相等时报告错误，错误讯息由 `$message` 指定。

`assertNotEquals()` 是与之相反的断言，接受相同的参数。

`assertAttributeEquals()` 和 `assertAttributeNotEquals()` 是便捷包装 (convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.16. assertEquals() 的用法

```
<?php
use PHPUnit\Framework\TestCase;
```

```
class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEquals(1, 0);
    }

    public function testFailure2()
    {
        $this->assertEquals('bar', 'baz');
    }

    public function testFailure3()
    {
        $this->assertEquals("foo\nbar\nbaz\n", "foo\nbah\nbaz\n");
    }
}
?>
```

```
phpunit EqualsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

FFF

Time: 0 seconds, Memory: 5.25Mb

There were 3 failures:

1) EqualsTest::testFailure
Failed asserting that 0 matches expected 1.

/home/sb/EqualsTest.php:6

2) EqualsTest::testFailure2
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/EqualsTest.php:11

3) EqualsTest::testFailure3
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
 'foo
-bar
+bah
 baz
 '

/home/sb/EqualsTest.php:16

FAILURES!
Tests: 3, Assertions: 3, Failures: 3.
```

如果 $expected 和 $actual 是某些特定的类型，将使用更加专门的比较方式，参阅下文。

assertEquals(float $expected, float $actual[, string $message = '', float $delta = 0])

当两个浮点数 $expected 和 $actual 之间的差值（的绝对值）大于 $delta 时报告错误，错误讯息由 $message 指定。

关于为什么 $delta 参数是必须的，请阅读《关于浮点运算，每一位计算机科学从业人员都应该知道的事实 [http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html]》。

例 A.17. 将assertEquals()用于浮点数时的用法

```
<?php
use PHPUnit\Framework\TestCase;

class EqualsTest extends TestCase
{
    public function testSuccess()
    {
        $this->assertEquals(1.0, 1.1, '', 0.2);
    }

    public function testFailure()
    {
        $this->assertEquals(1.0, 1.1);
    }
}
?>
```

```
phpunit EqualsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that 1.1 matches expected 1.0.

/home/sb/EqualsTest.php:11

FAILURES!
Tests: 2, Assertions: 2, Failures: 1.
```

assertEquals(DOMDocument $expected, DOMDocument $actual[, string $message = ''])

当 $expected 和 $actual 这两个 DOMDocument 对象所表示的 XML 文档对应的无注释规范形式不相同时报告错误，错误讯息由 $message 指定。

例 A.18. assertEquals()应用于 DOMDocument 对象时的用法

```
<?php
use PHPUnit\Framework\TestCase;

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo><bar/></foo>');

        $actual = new DOMDocument;
        $actual->loadXML('<bar><foo/></bar>');
```

```
        $this->assertEquals($expected, $actual);
    }
}
?>
```

```
phpunit EqualsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
-<foo>
-  <bar/>
-</foo>
+<bar>
+  <foo/>
+</bar>

/home/sb/EqualsTest.php:12

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

`assertEquals(object $expected, object $actual[, string $message = ''])`

当`$expected`和`$actual`这两个对象的属性值不相等时报告错误，错误讯息由`$message`指定。

例 A.19. assertEquals()应用于对象时的用法

```
<?php
use PHPUnit\Framework\TestCase;

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $expected = new stdClass;
        $expected->foo = 'foo';
        $expected->bar = 'bar';

        $actual = new stdClass;
        $actual->foo = 'bar';
        $actual->baz = 'bar';

        $this->assertEquals($expected, $actual);
    }
}
?>
```

```
phpunit EqualsTest
```

```
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
 stdClass Object (
-    'foo' => 'foo'
-    'bar' => 'bar'
+    'foo' => 'bar'
+    'baz' => 'bar'
 )

/home/sb/EqualsTest.php:14

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

`assertEquals(array $expected, array $actual[, string $message = ''])`

当 `$expected` 和 `$actual` 这两个数组不相等时报告错误，错误讯息由 `$message` 指定。

例 A.20. assertEquals() 应用于数组时的用法

```
<?php
use PHPUnit\Framework\TestCase;

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEquals(['a', 'b', 'c'], ['a', 'c', 'd']);
    }
}
?>
```

```
phpunit EqualsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
     0 => 'a'
-    1 => 'b'
-    2 => 'c'
+    1 => 'c'
+    2 => 'd'
```

```
 )

/home/sb/EqualsTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertFalse()

`assertFalse(bool $condition[, string $message = ''])`

当 `$condition` 为 `true` 时报告错误，错误讯息由 `$message` 指定。

`assertNotFalse()` 是与之相反的断言，接受相同的参数。

例 A.21. assertFalse() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class FalseTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFalse(true);
    }
}
?>
```

```
phpunit FalseTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalseTest::testFailure
Failed asserting that true is false.

/home/sb/FalseTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertFileEquals()

`assertFileEquals(string $expected, string $actual[, string $message = ''])`

当 `$expected` 所指定的文件与 `$actual` 所指定的文件内容不同时报告错误，错误讯息由 `$message` 指定。

`assertFileNotEquals()` 是与之相反的断言，接受相同的参数。

例 A.22. assertFileEquals() 的用法

```
<?php
```

```
use PHPUnit\Framework\TestCase;

class FileEqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileEquals('/home/sb/expected', '/home/sb/actual');
    }
}
?>
```

```
phpunit FileEqualsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) FileEqualsTest::testFailure
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'expected
+'actual
 '

/home/sb/FileEqualsTest.php:6

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertFileExists()

`assertFileExists(string $filename[, string $message = ''])`

当 `$filename` 所指定的文件不存在时报告错误，错误讯息由 `$message` 指定。

`assertFileNotExists()` 是与之相反的断言，接受相同的参数。

例 A.23. assertFileExists() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class FileExistsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileExists('/path/to/file');
    }
}
?>
```

```
phpunit FileExistsTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F
```

```
Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) FileExistsTest::testFailure
Failed asserting that file "/path/to/file" exists.

/home/sb/FileExistsTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertFileIsReadable()

`assertFileIsReadable(string $filename[, string $message = ''])`

当 `$filename` 所指定的文件不是个文件或不可读时报告错误，错误讯息由 `$message` 指定。

`assertFileNotIsReadable()` 是与之相反的断言，并接受相同的参数。

例 A.24. assertFileIsReadable() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class FileIsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileIsReadable('/path/to/file');
    }
}
?>
```

```
phpunit FileIsReadableTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) FileIsReadableTest::testFailure
Failed asserting that "/path/to/file" is readable.

/home/sb/FileIsReadableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertFileIsWritable()

`assertFileIsWritable(string $filename[, string $message = ''])`

当 `$filename` 所指定的文件不是个文件或不可写时报告错误，错误讯息由 `$message` 指定。

`assertFileNotIsWritable()` 是与之相反的断言，并接受相同的参数。

例 A.25. assertFileIsWritable() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class FileIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileIsWritable('/path/to/file');
    }
}
?>
```

```
phpunit FileIsWritableTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) FileIsWritableTest::testFailure
Failed asserting that "/path/to/file" is writable.

/home/sb/FileIsWritableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertGreaterThan()

`assertGreaterThan(mixed $expected, mixed $actual[, string $message = ''])`

当 `$actual` 的值不大于 `$expected` 的值时报告错误，错误讯息由 `$message` 指定。

`assertAttributeGreaterThan()` 是便捷包装(convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.26. assertGreaterThan() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class GreaterThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertGreaterThan(2, 1);
    }
}
?>
```

```
phpunit GreaterThanTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F
```

```
Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) GreaterThanTest::testFailure
Failed asserting that 1 is greater than 2.

/home/sb/GreaterThanTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertGreaterThanOrEqual()

`assertGreaterThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])`

当 `$actual` 的值不大于且不等于 `$expected` 的值时报告错误，错误讯息由 `$message` 指定。

`assertAttributeGreaterThanOrEqual()` 是便捷包装(convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.27. assertGreaterThanOrEqual() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class GreatThanOrEqualTest extends TestCase
{
    public function testFailure()
    {
        $this->assertGreaterThanOrEqual(2, 1);
    }
}
?>
```

```
phpunit GreaterThanOrEqualTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) GreatThanOrEqualTest::testFailure
Failed asserting that 1 is equal to 2 or is greater than 2.

/home/sb/GreaterThanOrEqualTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

assertInfinite()

`assertInfinite(mixed $variable[, string $message = ''])`

当 `$actual` 不是 `INF` 时报告错误，错误讯息由 `$message` 指定。

assertFinite() 是与之相反的断言，接受相同的参数。

例 A.28. assertInfinite() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class InfiniteTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInfinite(1);
    }
}
?>
```

```
phpunit InfiniteTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InfiniteTest::testFailure
Failed asserting that 1 is infinite.

/home/sb/InfiniteTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertInstanceOf()

assertInstanceOf($expected, $actual[, $message = ''])

当 $actual 不是 $expected 的实例时报告错误，错误讯息由 $message 指定。

assertNotInstanceOf() 是与之相反的断言，接受相同的参数。

assertAttributeInstanceOf() 和 assertAttributeNotInstanceOf() 是便捷包装(convenience wrapper)，可以应用于某个类或对象的某个 public、protected 或 private 属性。

例 A.29. assertInstanceOf() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class InstanceOfTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInstanceOf(RuntimeException::class, new Exception);
    }
}
?>
```

```
phpunit InstanceOfTest
```

```
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InstanceOfTest::testFailure
Failed asserting that Exception Object (...) is an instance of class "RuntimeException".

/home/sb/InstanceOfTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertInternalType()

`assertInternalType($expected, $actual[, $message = ''])`

当 `$actual` 不是 `$expected` 所指明的类型时报告错误，错误讯息由 `$message` 指定。

`assertNotInternalType()` 是与之相反的断言，接受相同的参数。

`assertAttributeInternalType()` 和 `assertAttributeNotInternalType()` 是便捷包装(convenience wrapper)，可以应用于某个类或对象的某个 `public`、`protected` 或 `private` 属性。

例 A.30. assertInternalType() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class InternalTypeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInternalType('string', 42);
    }
}
?>
```

```
phpunit InternalTypeTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InternalTypeTest::testFailure
Failed asserting that 42 is of type "string".

/home/sb/InternalTypeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertIsReadable()

assertIsReadable(string $filename[, string $message = ''])

当 $filename 所指定的文件或目录不可读时报告错误，错误讯息由 $message 指定。

assertNotIsReadable() 是与之相反的断言，并接受相同的参数。

例 A.31. assertIsReadable() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class IsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsReadable('/path/to/unreadable');
    }
}
?>
```

```
phpunit IsReadableTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IsReadableTest::testFailure
Failed asserting that "/path/to/unreadable" is readable.

/home/sb/IsReadableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertIsWritable()

assertIsWritable(string $filename[, string $message = ''])

当 $filename 所指定的文件或目录不可写时报告错误，错误讯息由 $message 指定。

assertNotIsWritable() 是与之相反的断言，并接受相同的参数。

例 A.32. assertIsWritable() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class IsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsWritable('/path/to/unwritable');
    }
}
?>
```

```
phpunit IsWritableTest
```

```
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IsWritableTest::testFailure
Failed asserting that "/path/to/unwritable" is writable.

/home/sb/IsWritableTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertJsonFileEqualsJsonFile()

```
assertJsonFileEqualsJsonFile(mixed $expectedFile, mixed
$actualFile[, string $message = ''])
```

当 $actualFile 对应的值与 $expectedFile 对应的值不匹配时报告错误，错误讯息由 $message 指定。

例 A.33. assertJsonFileEqualsJsonFile() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class JsonFileEqualsJsonFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonFileEqualsJsonFile(
          'path/to/fixture/file', 'path/to/actual/file');
    }
}
?>
```

```
phpunit JsonFileEqualsJsonFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonFileEqualsJsonFile::testFailure
Failed asserting that '{"Mascot":"Tux"}' matches JSON string "["Mascott", "Tux", "OS", "

/home/sb/JsonFileEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertJsonStringEqualsJsonFile()

```
assertJsonStringEqualsJsonFile(mixed $expectedFile, mixed
$actualJson[, string $message = ''])
```

当 $actualJson 对应的值与 $expectedFile 对应的值不匹配时报告错误，错误讯息由 $message 指定。

例 A.34. assertJsonStringEqualsJsonFile() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class JsonStringEqualsJsonFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonFile(
            'path/to/fixture/file', json_encode(['Mascot' => 'ux'])
        );
    }
}
?>
```

```
phpunit JsonStringEqualsJsonFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonFile::testFailure
Failed asserting that '{"Mascot":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/JsonStringEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertJsonStringEqualsJsonString()

assertJsonStringEqualsJsonString(mixed $expectedJson, mixed $actualJson[, string $message = ''])

当 $actualJson 对应的值与 $expectedJson 对应的值不匹配时报告错误，错误讯息由 $message 指定。

例 A.35. assertJsonStringEqualsJsonString() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class JsonStringEqualsJsonStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonString(
            json_encode(['Mascot' => 'Tux']),
            json_encode(['Mascot' => 'ux'])
        );
    }
}
?>
```

```
phpunit JsonStringEqualsJsonStringTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonStringTest::testFailure
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
 stdClass Object (
 -    'Mascot' => 'Tux'
 +    'Mascot' => 'ux'
)

/home/sb/JsonStringEqualsJsonStringTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertLessThan()

`assertLessThan(mixed $expected, mixed $actual[, string $message = ''])`

当 `$actual` 的值不小于 `$expected` 的值时报告错误，错误讯息由 `$message` 指定。

`assertAttributeLessThan()` 是便捷包装(convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.36. assertLessThan() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class LessThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThan(1, 2);
    }
}
?>
```

```
phpunit LessThanTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) LessThanTest::testFailure
Failed asserting that 2 is less than 1.

/home/sb/LessThanTest.php:6
```

```
FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertLessThanOrEqual()

`assertLessThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])`

当 `$actual` 的值不小于且不等于 `$expected` 的值时报告错误，错误讯息由 `$message` 指定。

`assertAttributeLessThanOrEqual()` 是便捷包装(convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.37. assertLessThanOrEqual() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class LessThanOrEqualTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThanOrEqual(1, 2);
    }
}
?>
```

```
phpunit LessThanOrEqualTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) LessThanOrEqualTest::testFailure
Failed asserting that 2 is equal to 1 or is less than 1.

/home/sb/LessThanOrEqualTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

assertNan()

`assertNan(mixed $variable[, string $message = ''])`

当 `$variable` 不是 NAN 时报告错误，错误讯息由 `$message` 指定。

例 A.38. assertNan() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class NanTest extends TestCase
{
```

```
    public function testFailure()
    {
        $this->assertNan(1);
    }
}
?>
```

```
phpunit NanTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NanTest::testFailure
Failed asserting that 1 is nan.

/home/sb/NanTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertNull()

`assertNull(mixed $variable[, string $message = ''])`

当 `$actual` 不是 `null` 时报告错误，错误讯息由 `$message` 指定。

`assertNotNull()` 是与之相反的断言，接受相同的参数。

例 A.39. assertNull() 的使用

```
<?php
use PHPUnit\Framework\TestCase;

class NullTest extends TestCase
{
    public function testFailure()
    {
        $this->assertNull('foo');
    }
}
?>
```

```
phpunit NotNullTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NullTest::testFailure
Failed asserting that 'foo' is null.

/home/sb/NotNullTest.php:6

FAILURES!
```

```
Tests: 1, Assertions: 1, Failures: 1.
```

assertObjectHasAttribute()

`assertObjectHasAttribute(string $attributeName, object $object[, string $message = ''])`

当 `$object->attributeName` 不存在时报告错误，错误讯息由 `$message` 指定。

`assertObjectNotHasAttribute()` 是与之相反的断言，接受相同的参数。

例 A.40. assertObjectHasAttribute() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class ObjectHasAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertObjectHasAttribute('foo', new stdClass);
    }
}
?>
```

```
phpunit ObjectHasAttributeTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ObjectHasAttributeTest::testFailure
Failed asserting that object of class "stdClass" has attribute "foo".

/home/sb/ObjectHasAttributeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertRegExp()

`assertRegExp(string $pattern, string $string[, string $message = ''])`

当 `$string` 不匹配于正则表达式 `$pattern` 时报告错误，错误讯息由 `$message` 指定。

`assertNotRegExp()` 是与之相反的断言，接受相同的参数。

例 A.41. assertRegExp() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class RegExpTest extends TestCase
{
    public function testFailure()
    {
```

```
        $this->assertRegExp('/foo/', 'bar');
    }
}
?>
```

```
phpunit RegExpTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) RegExpTest::testFailure
Failed asserting that 'bar' matches PCRE pattern "/foo/".

/home/sb/RegExpTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertStringMatchesFormat()

`assertStringMatchesFormat(string $format, string $string[, string $message = ''])`

当 `$string` 不匹配于 `$format` 定义的格式时报告错误，错误讯息由 `$message` 指定。

`assertStringNotMatchesFormat()` 是与之相反的断言，接受相同的参数。

例 A.42. assertStringMatchesFormat() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class StringMatchesFormatTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormat('%i', 'foo');
    }
}
?>
```

```
phpunit StringMatchesFormatTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringMatchesFormatTest::testFailure
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+$/s".

/home/sb/StringMatchesFormatTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

格式定义字符串中可以使用如下占位符：

- %e：表示目录分隔符，例如在 Linux 系统中是 /。
- %s：一个或多个除了换行符以外的任意字符（非空白字符或者空白字符）。
- %S：零个或多个除了换行符以外的任意字符（非空白字符或者空白字符）。
- %a：一个或多个包括换行符在内的任意字符（非空白字符或者空白字符）。
- %A：零个或多个包括换行符在内的任意字符（非空白字符或者空白字符）。
- %w：零个或多个空白字符。
- %i：带符号整数值，例如 +3142、-3142。
- %d：无符号整数值，例如 123456。
- %x：一个或多个十六进制字符。所谓十六进制字符，指的是在以下范围内的字符：0-9、a-f、A-F。
- %f：浮点数，例如 3.142、-3.142、3.142E-10、3.142e+10。
- %c：单个任意字符。

assertStringMatchesFormatFile()

`assertStringMatchesFormatFile(string $formatFile, string $string[, string $message = ''])`

当 `$string` 不匹配于 `$formatFile` 的内容所定义的格式时报告错误，错误讯息由 `$message` 指定。

`assertStringNotMatchesFormatFile()` 是与之相反的断言，接受相同的参数。

例 A.43. assertStringMatchesFormatFile() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class StringMatchesFormatFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormatFile('/path/to/expected.txt', 'foo');
    }
}
?>
```

```
phpunit StringMatchesFormatFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringMatchesFormatFileTest::testFailure
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+
$/s".
```

```
/home/sb/StringMatchesFormatFileTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

assertSame()

`assertSame(mixed $expected, mixed $actual[, string $message = ''])`

当两个变量 `$expected` 和 `$actual` 的值与类型不完全相同时报告错误，错误讯息由 `$message` 指定。

`assertNotSame()` 是与之相反的断言，接受相同的参数。

`assertAttributeSame()` 和 `assertAttributeNotSame()` 是便捷包装(convenience wrapper)，以某个类或对象的某个 `public`、`protected` 或 `private` 属性作为实际值来进行比较。

例 A.44. assertSame() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame('2204', 2204);
    }
}
?>
```

```
phpunit SameTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) SameTest::testFailure
Failed asserting that 2204 is identical to '2204'.

/home/sb/SameTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

`assertSame(object $expected, object $actual[, string $message = ''])`

当两个变量 `$expected` 和 `$actual` 不是指向同一个对象的引用时报告错误，错误讯息由 `$message` 指定。

例 A.45. assertSame() 应用于对象时的用法

```php
<?php
use PHPUnit\Framework\TestCase;
```

```
class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame(new stdClass, new stdClass);
    }
}
?>
```

```
phpunit SameTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) SameTest::testFailure
Failed asserting that two variables reference the same object.

/home/sb/SameTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertStringEndsWith()

`assertStringEndsWith(string $suffix, string $string[, string $message = ''])`

当 `$string` 不以 `$suffix` 结尾时报告错误，错误讯息由 `$message` 指定。

`assertStringEndsNotWith()` 是与之相反的断言，接受相同的参数。

例 A.46. assertStringEndsWith() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class StringEndsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEndsWith('suffix', 'foo');
    }
}
?>
```

```
phpunit StringEndsWithTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 1 second, Memory: 5.00Mb

There was 1 failure:

1) StringEndsWithTest::testFailure
Failed asserting that 'foo' ends with "suffix".
```

```
/home/sb/StringEndsWithTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertStringEqualsFile()

`assertStringEqualsFile(string $expectedFile, string $actualString[, string $message = ''])`

当 `$expectedFile` 所指定的文件其内容不是 `$actualString` 时报告错误，错误讯息由 `$message` 指定。

`assertStringNotEqualsFile()` 是与之相反的断言，接受相同的参数。

例 A.47. assertStringEqualsFile() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class StringEqualsFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEqualsFile('/home/sb/expected', 'actual');
    }
}
?>
```

```
phpunit StringEqualsFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringEqualsFileTest::testFailure
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'expected
-'
+'actual'

/home/sb/StringEqualsFileTest.php:6

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

assertStringStartsWith()

`assertStringStartsWith(string $prefix, string $string[, string $message = ''])`

当 `$string` 不以 `$prefix` 开头时报告错误，错误讯息由 `$message` 指定。

`assertStringStartsNotWith()` 是与之相反的断言，并接受相同的参数。

例 A.48. assertStringStartsWith() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class StringStartsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringStartsWith('prefix', 'foo');
    }
}
?>
```

```
phpunit StringStartsWithTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringStartsWithTest::testFailure
Failed asserting that 'foo' starts with "prefix".

/home/sb/StringStartsWithTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertThat()

可以用 PHPUnit_Framework_Constraint 类来订立更加复杂的断言。随后可以用 assertThat() 方法来评定这些断言。例 A.49 “assertThat() 的用法” 展示了如何用 logicalNot() 和 equalTo() 约束条件来表达与 assertNotEquals() 等价的断言。

```
assertThat(mixed $value, PHPUnit_Framework_Constraint $constraint[,
$message = ''])
```

当 $value 不符合约束条件 $constraint 时报告错误，错误讯息由 $message 指定。

例 A.49. assertThat() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class BiscuitTest extends TestCase
{
    public function testEquals()
    {
        $theBiscuit = new Biscuit('Ginger');
        $myBiscuit  = new Biscuit('Ginger');

        $this->assertThat(
          $theBiscuit,
          $this->logicalNot(
            $this->equalTo($myBiscuit)
          )
        );
    }
```

```
}
?>
```

表 A.1 “约束条件”列举了所有可用的 PHPUnit_Framework_Constraint 类。

表 A.1. 约束条件

约束条件	含义
PHPUnit_ Framework_ Constraint_ Attribute attribute(PHPUnit_ Framework_ Constraint $constraint, $attributeName)	此约束将另外一个约束应用于某个类或对象的某个属性。
PHPUnit_ Framework_ Constraint_ IsAnything anything()	此约束接受任意输入值。
PHPUnit_ Framework_ Constraint_ ArrayHasKey arrayHasKey(mixed $key)	此约束断言所评定的数组拥有指定键名。
PHPUnit_ Framework_ Constraint_ TraversableContains contains(mixed $value)	此约束断言所评定的 array 或实现了 Iterator 接口的对象包含有给定值。
PHPUnit_ Framework_ Constraint_ TraversableContainsOnly containsOnly(string $type)	此约束断言所评定的 array 或实现了 Iterator 接口的对象仅包含给定类型的值。
PHPUnit_ Framework_ Constraint_ TraversableContainsOnly containsOnlyInstancesOf(string $classname)	此约束断言所评定的 array 或实现了 Iterator 接口的对象仅包含给定类名的类的实例。
PHPUnit_ Framework_ Constraint_ IsEqual equalTo($value, $delta = 0, $maxDepth = 10)	此约束检验一个值是否等于另外一个。
PHPUnit_ Framework_ Constraint_ Attribute attributeEqualTo($attributeName, $value, $delta = 0, $maxDepth = 10)	此约束检验一个值是否等于某个类或对象的某个属性。
PHPUnit_ Framework_ Constraint_ DirectoryExists directoryExists()	此约束检验所评定的目录是否存在。
PHPUnit_ Framework_ Constraint_ FileExists fileExists()	此约束检验所评定的文件名对应的文件是否存在。
PHPUnit_ Framework_ Constraint_ IsReadable isReadable()	此约束检验所评定的文件名对应的文件是否可读。
PHPUnit_ Framework_ Constraint_ IsWritable isWritable()	此约束检验所评定的文件名对应的文件是否可写。
PHPUnit_ Framework_ Constraint_ GreaterThan greaterThan(mixed $value)	此约束断言所评定的值大于给定值。
PHPUnit_ Framework_ Constraint_ Or greaterThanOrEqual(mixed $value)	此约束断言所评定的值大于或等于给定值。
PHPUnit_ Framework_ Constraint_ ClassHasAttribute	此约束断言所评定的类具有给定属性。

约束条件	含义
`classHasAttribute(string $attributeName)`	
`PHPUnit_ Framework_ Constraint_ ClassHasStaticAttribute classHasStaticAttribute(string $attributeName)`	此约束断言所评定的类具有给定静态属性。
`PHPUnit_ Framework_ Constraint_ ObjectHasAttribute hasAttribute(string $attributeName)`	此约束断言所评定的对象具有给定属性。
`PHPUnit_ Framework_ Constraint_ IsIdentical identicalTo(mixed $value)`	此约束断言所评定的值与另外一个值全等。
`PHPUnit_ Framework_ Constraint_ IsFalse isFalse()`	此约束断言所评定的值为 `false`。
`PHPUnit_ Framework_ Constraint_ IsInstanceOf isInstanceOf(string $className)`	此约束断言所评定的对象是给定类的实例。
`PHPUnit_ Framework_ Constraint_ IsNull isNull()`	此约束断言所评定的值为 `null`。
`PHPUnit_ Framework_ Constraint_ IsTrue isTrue()`	此约束断言所评定的值为 `true`。
`PHPUnit_ Framework_ Constraint_ IsType isType(string $type)`	此约束断言所评定的值是指定类型的。
`PHPUnit_ Framework_ Constraint_ LessThan lessThan(mixed $value)`	此约束断言所评定的值小于给定值。
`PHPUnit_ Framework_ Constraint_ Or lessThanOrEqual(mixed $value)`	此约束断言所评定的值小于或等于给定值。
`logicalAnd()`	逻辑与(AND)。
`logicalNot(PHPUnit_ Framework_ Constraint $constraint)`	逻辑非(NOT)。
`logicalOr()`	逻辑或(OR)。
`logicalXor()`	逻辑异或(XOR)。
`PHPUnit_ Framework_ Constraint_ PCREMatch matchesRegularExpression(string $pattern)`	此约束断言所评定的字符串匹配于正则表达式。
`PHPUnit_ Framework_ Constraint_ StringContains stringContains(string $string, bool $case)`	此约束断言所评定的字符串包含指定字符串。
`PHPUnit_ Framework_ Constraint_ StringEndsWith stringEndsWith(string $suffix)`	此约束断言所评定的字符串以给定后缀结尾。
`PHPUnit_ Framework_ Constraint_ StringStartsWith stringStartsWith(string $prefix)`	此约束断言所评定的字符串以给定前缀开头。

assertTrue()

```
assertTrue(bool $condition[, string $message = ''])
```

当 `$condition` 为 `false` 时报告错误，错误讯息由 `$message` 指定。

`assertNotTrue()` 是与之相反的断言，接受相同的参数。

例 A.50. assertTrue() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class TrueTest extends TestCase
{
    public function testFailure()
    {
        $this->assertTrue(false);
    }
}
?>
```

```
phpunit TrueTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) TrueTest::testFailure
Failed asserting that false is true.

/home/sb/TrueTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

assertXmlFileEqualsXmlFile()

```
assertXmlFileEqualsXmlFile(string $expectedFile, string
$actualFile[, string $message = ''])
```

当 `$actualFile` 对应的 XML 文档与 `$expectedFile` 对应的 XML 文档不相同时报告错误，错误讯息由 `$message` 指定。

`assertXmlFileNotEqualsXmlFile()` 是与之相反的断言，接受相同的参数。

例 A.51. assertXmlFileEqualsXmlFile() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class XmlFileEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlFileEqualsXmlFile(
          '/home/sb/expected.xml', '/home/sb/actual.xml');
```

```
    }
}
?>
```

```
phpunit XmlFileEqualsXmlFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) XmlFileEqualsXmlFileTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>

/home/sb/XmlFileEqualsXmlFileTest.php:7

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])`

当 `$actualXml` 对应的 XML 文档与 `$expectedFile` 对应的 XML 文档不相同时报告错误，错误讯息由 `$message` 指定。

`assertXmlStringNotEqualsXmlFile()` 是与之相反的断言，并接受相同的参数。

例 A.52. assertXmlStringEqualsXmlFile() 的用法

```
<?php
use PHPUnit\Framework\TestCase;

class XmlStringEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlFile(
          '/home/sb/expected.xml', '<foo><baz/></foo>');
    }
}
?>
```

```
phpunit XmlStringEqualsXmlFileTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb
```

```
There was 1 failure:

1) XmlStringEqualsXmlFileTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>

/home/sb/XmlStringEqualsXmlFileTest.php:7

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])`

当 `$actualXml` 对应的 XML 文档与 `$expectedXml` 对应的 XML 文档不相同时报告错误，错误讯息由 `$message` 指定。

`assertXmlStringNotEqualsXmlString()` 是与之相反的断言，接受相同的参数。

例 A.53. assertXmlStringEqualsXmlString() 的用法

```php
<?php
use PHPUnit\Framework\TestCase;

class XmlStringEqualsXmlStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlString(
          '<foo><bar/></foo>', '<foo><baz/></foo>');
    }
}
?>
```

```
phpunit XmlStringEqualsXmlStringTest
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) XmlStringEqualsXmlStringTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>
```

```
/home/sb/XmlStringEqualsXmlStringTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

附录 B. 标注

所谓标注，是指某些编程语言中允许加在源代码中的一种特殊格式的语法元数据。PHP 并没有专门的语言特性来支持对源代码进行标注，然而 PHP 社区早已经形成惯例，通过在文档注释块中使用诸如 `@annotation arguments` 这样的标签来为源代码加上标注。在 PHP 中，文档注释块是可反射的：可以对函数、方法、类以及属性调用相应级别的反射 API `getDocComment()` 方法来获取相应的文档注释块。诸如 PHPUnit 这样的应用程序在运行时用这些信息来配置其行为。

> **注意**
>
> PHP中的文档注释块必须以 `/**` 开头，以 `*/` 结尾。任何其他形式的注释中出现的标注都将被忽略。

本附录列出了 PHPUnit 所支持的所有标注种类。

@author

`@author` 标注是 `@group` 标注（参见 “@group”一节）的别名，允许基于作者对测试进行过滤。

@after

`@after` 标注用于指明此方法应当在测试用例类中的每个测试方法运行完成之后调用。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @after
     */
    public function tearDownSomeFixtures()
    {
        // ...
    }

    /**
     * @after
     */
    public function tearDownSomeOtherFixtures()
    {
        // ...
    }
}
```

@afterClass

`@afterClass` 标注用于指明此静态方法应该于测试类中的所有测试方法都运行完成之后调用，用于清理共享基境。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
```

```
     * @afterClass
     */
    public static function tearDownSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @afterClass
     */
    public static function tearDownSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

@backupGlobals

全局变量的备份与还原操作可以对某个测试用例类中的所有测试彻底禁用，像这样：

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    // ...
}
```

@backupGlobals 标注也可以用在测试方法这一级别。这样可以对备份与还原操作进行更细粒度的配置：

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testThatInteractsWithGlobalVariables()
    {
        // ...
    }
}
```

@backupStaticAttributes

如果指定了 @backupStaticAttributes 标注，那么将在每个测试之前备份所有已声明的类的静态属性的值，并在测试完成之后全部恢复。它可以用在测试用例类或测试方法级别：

```
use PHPUnit\Framework\TestCase;

/**
 * @backupStaticAttributes enabled
 */
class MyTest extends TestCase
```

```
{
    /**
     * @backupStaticAttributes disabled
     */
    public function testThatInteractsWithStaticAttributes()
    {
        // ...
    }
}
```

注意

受限于 PHP 的内部实现，在某些情况下即使使用了 @backupStaticAttributes 也可能有个别静态值出现意料外的延续，并污染后继测试。

详细信息参见 “全局状态”一节。

@before

@before 标注用于指明此方法应当在测试用例类中的每个测试方法开始运行之前调用。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @before
     */
    public function setupSomeFixtures()
    {
        // ...
    }

    /**
     * @before
     */
    public function setupSomeOtherFixtures()
    {
        // ...
    }
}
```

@beforeClass

@beforeClass 标注用于指明此静态方法应该于测试类中的所有测试方法都运行完成之后调用，用于建立共享基境。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @beforeClass
     */
    public static function setUpSomeSharedFixtures()
    {
        // ...
    }

    /**
```

```
     * @beforeClass
     */
    public static function setUpSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

@codeCoverageIgnore*

@codeCoverageIgnore, @codeCoverageIgnoreStart and @codeCoverageIgnoreEnd 标注用于从覆盖率分析中排除掉某些代码行。

用法参见“略过代码块”一节。

@covers

在测试代码中用 @covers 标注来指明测试方法想要对哪些方法进行测试：

```
/**
 * @covers BankAccount::getBalance
 */
public function testBalanceIsInitiallyZero()
{
    $this->assertEquals(0, $this->ba->getBalance());
}
```

如果提供了此标注，则代码覆盖率信息中只考虑指定的这些方法。

表 B.1 “用于指明测试覆盖哪些方法的标注”列出了 @covers 标注的语法。

表 B.1. 用于指明测试覆盖哪些方法的标注

Annotation（标注）	描述
@covers ClassName::methodName	指明所标注的测试方法覆盖指定的方法。
@covers ClassName	指明所标注的测试方法覆盖给定类的全部方法。
@covers ClassName<extended>	指明所标注的测试方法覆盖给定类以及其所有父类与接口的全部方法。
@covers ClassName::<public>	指明所标注的测试方法覆盖给定类的所有 public 方法。
@covers ClassName::<protected>	指明所标注的测试方法覆盖给定类的所有 protected 方法。
@covers ClassName::<private>	指明所标注的测试方法覆盖给定类的所有 private 方法。
@covers ClassName::<!public>	指明所标注的测试方法覆盖给定类的所有非 public 方法。
@covers ClassName::<!protected>	指明所标注的测试方法覆盖给定类的所有非 protected 方法。
@covers ClassName::<!private>	指明所标注的测试方法覆盖给定类的所有非 private 方法。
@covers ::functionName	指明所标注的测试方法覆盖给定的全局函数。

@coversDefaultClass

`@coversDefaultClass` 标注用于指定一个默认的命名空间或类名，这样就不用在每个 `@covers` 标注中重复长名称。参见例 B.1 “用 @coversDefaultClass 缩短标注”。

例 B.1. 用 @coversDefaultClass 缩短标注

```
<?php
use PHPUnit\Framework\TestCase;

/**
 * @coversDefaultClass \Foo\CoveredClass
 */
class CoversDefaultClassTest extends TestCase
{
    /**
     * @covers ::publicMethod
     */
    public function testSomething()
    {
        $o = new Foo\CoveredClass;
        $o->publicMethod();
    }
}
?>
```

@coversNothing

在测试代码中用 `@coversNothing` 标注来指明所标注的测试用例不需要记录任何代码覆盖率信息。

这可以用于集成测试。例子可参见例 11.3 “指明测试不欲覆盖任何方法”。

这个标注可以用在类级别或者方法级别，并且会覆盖掉任何 `@covers` 标注。

@dataProvider

测试方法可以接受任意参数。这些参数可以由数据供给器方法（例 2.5 “使用返回数组的数组的数据供给器”中的 `provider()`）提供。所要使用的数据供给器方法用 `@dataProvider` 标注来指定。

更多细节参见“数据供给器”一节。

@depends

PHPUnit支持对测试方法之间的显式依赖关系进行声明。这种依赖关系并不是定义在测试方法的执行顺序中，而是允许生产者(producer)返回一个测试基境(fixture)的实例，并将此实例传递给依赖于它的消费者(consumer)们。例 2.2 “用 `@depends` 标注来表达依赖关系”展示了如何用 `@depends` 标注来表达测试方法之间的依赖关系。

更多细节参见“测试的依赖关系”一节。

@expectedException

例 2.10 “使用 expectException() 方法”展示了如何用 `@expectedException` 标注来测试被测代码中是否抛出了异常。

更多细节参见“对异常进行测试”一节。

@expectedExceptionCode

将 @expectedExceptionCode 标注与 @expectedException 联合使用，可以对抛出异常的代码作出断言，这样可以缩小具体异常的范围。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException     MyException
     * @expectedExceptionCode 20
     */
    public function testExceptionHasErrorcode20()
    {
        throw new MyException('Some Message', 20);
    }
}
```

为了方便测试并减少冗余，可以用"@expectedExceptionCode ClassName::CONST"这样的语法将指定类常量作为 @expectedExceptionCode

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
      * @expectedException     MyException
      * @expectedExceptionCode MyClass::ERRORCODE
      */
    public function testExceptionHasErrorcode20()
    {
      throw new MyException('Some Message', 20);
    }
}
class MyClass
{
    const ERRORCODE = 20;
}
```

@expectedExceptionMessage

@expectedExceptionMessage 标注的运作方式类似于 @expectedExceptionCode ，用它可以对异常的错误讯息作出断言。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException        MyException
     * @expectedExceptionMessage Some Message
     */
    public function testExceptionHasRightMessage()
    {
        throw new MyException('Some Message', 20);
    }
}
```

预期讯息可以是异常讯息的子串。在只需要断言传入的特定名称或参数确实出现于异常中时这个特性很有用，这样就无需在测试中关注完整的异常讯息。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException        MyException
      * @expectedExceptionMessage broken
      */
     public function testExceptionHasRightMessage()
     {
         $param = "broken";
         throw new MyException('Invalid parameter "'.$param.'".', 20);
     }
}
```

为了方便测试同时减少冗余，可以用"@expectedExceptionMessage ClassName::CONST"这样的语法将指定类常量作为 @expectedExceptionMessage。在“@expectedExceptionCode”一节中可以看到范例。

@expectedExceptionMessageRegExp

预期讯息也可以通过 @expectedExceptionMessageRegExp 标注以正则表达式来指定。当无法用子串来完成对给定讯息的匹配时，这种方式就非常有用了。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException              MyException
      * @expectedExceptionMessageRegExp /Argument \d+ can not be an? \w+/
      */
     public function testExceptionHasRightMessage()
     {
         throw new MyException('Argument 2 can not be an integer');
     }
}
```

@group

测试可以用 @group 标注来标记为属于一个或多个组，就像这样：

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
    }

    /**
     * @group regresssion
     * @group bug2204
     */
```

```
    public function testSomethingElse()
    {
    }
}
```

测试可以基于组来选择性的执行，使用命令行测试执行器的 `--group` and `--exclude-group` 选项，或者使用对应的 XML 配置文件指令。

@large

`@large` 标注是 `@group large` 的别名。

如果安装了 `PHP_Invoker` 组件包并启用了严格模式，一个执行时间超过60秒的大型(large)测试将视为失败。这个超时限制可以通过 XML 配置文件的 `timeoutForLargeTests` 属性进行配置。

@medium

`@medium` 标注是 `@group medium` 的别名。中型(medium)测试不能依赖于标记为 `@large` 的测试。

如果安装了 `PHP_Invoker` 组件包并启用了严格模式，一个执行时间超过10秒的中型(medium)测试将视为失败。这个超时限制可以通过 XML 配置文件的 `timeoutForMediumTests` 属性进行配置。

@preserveGlobalState

在单独的进程中运行测试时，PHPUnit 会尝试保持来自父进程的全局状态（通过在父进程序列化全局状态然后在子进程反序列化的方式）。这当父进程包含非可序列化的全局内容时可能会导致问题。为了修正这种问题，可以用 `@preserveGlobalState` 标注来禁止 PHPUnit 保持全局状态。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     * @preserveGlobalState disabled
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

@requires

`@requires` 标注用于在常规前提条件（例如 PHP 版本或所安装的扩展）不满足时跳过测试。

完整的可能用法以及例子见表 7.3 “可能的 @requires 用法”

@runTestsInSeparateProcesses

指明单个测试类内的所有测试要各自运行在独立的 PHP 进程中。

```
use PHPUnit\Framework\TestCase;

/**
 * @runTestsInSeparateProcesses
 */
class MyTest extends TestCase
{
    // ...
}
```

注意：“@preserveGlobalState”一节 默认情况下，PHPUnit 会尝试通过在父进程序列化全局状态然后在子进程反序列化的方式在子进程中保持来自父进程的全局状态。这当父进程包含非可序列化的全局内容时可能会导致问题。关于如何修正此问题的信息参见“@preserveGlobalState”一节。

@runInSeparateProcess

明某个测试要运行在独立的 PHP 进程中。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

注意：“@preserveGlobalState”一节 默认情况下，PHPUnit 会尝试通过在父进程序列化全局状态然后在子进程反序列化的方式在子进程中保持来自父进程的全局状态。这当父进程包含非可序列化的全局内容时可能会导致问题。关于如何修正此问题的信息参见“@preserveGlobalState”一节。

@small

`@small` 标注是 `@group  small` 的别名。小型(small)测试不能依赖于标记为 `@medium` 或 `@large` 的测试。

如果安装了 `PHP_Invoker` 组件包并启用了严格模式，一个执行时间超过1秒的小型(small)测试将会视为失败。这个超时限制可以通过 XML 配置文件的 `timeoutForSmallTests` 属性进行配置。

注意

需要启用运行时间限制的测试必须显式地标注为 `@small`、`@medium` 或 `@large`。

@test

除了用 `test` 作为测试方法名称的前缀外，还可以在方法的文档注释块中用 `@test` 标注来将其标记为测试方法。

```
/**
 * @test
 */
```

```
public function initialBalanceShouldBe0()
{
    $this->assertEquals(0, $this->ba->getBalance());
}
```

@testdox

@ticket

@uses

@uses 标注用来指明那些将会在测试中执行到但同时又不打算让其被测试所覆盖的代码。在对代码单元进行测试时所必须的值对象就是个很好的例子。

```
/**
 * @covers BankAccount::deposit
 * @uses   Money
 */
public function testMoneyCanBeDepositedInAccount()
{
    // ...
}
```

在严格覆盖模式中，意外覆盖的代码将导致测试判定为失败，这个标注就显得特别有用。关于严格覆盖模式的更多信息，参见“意外的代码覆盖”一节。

附录 C. XML 配置文件

PHPUnit

<phpunit> 元素的属性用于配置 PHPUnit 的核心功能。

```
<phpunit
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/6.3/phpunit.xsd"
         backupGlobals="true"
         backupStaticAttributes="false"
         <!--bootstrap="/path/to/bootstrap.php"-->
         cacheTokens="false"
         colors="false"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         forceCoversAnnotation="false"
         mapTestClassNameToCoveredClassName="false"
         printerClass="PHPUnit_TextUI_ResultPrinter"
         <!--printerFile="/path/to/ResultPrinter.php"-->
         processIsolation="false"
         stopOnError="false"
         stopOnFailure="false"
         stopOnIncomplete="false"
         stopOnSkipped="false"
         stopOnRisky="false"
         testSuiteLoaderClass="PHPUnit_Runner_StandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
         timeoutForSmallTests="1"
         timeoutForMediumTests="10"
         timeoutForLargeTests="60"
         verbose="false">
  <!-- ... -->
</phpunit>
```

以上 XML 配置对应于在“命令行选项”一节描述过的 TextUI 测试执行器的默认行为。

其他那些不能用命令行选项来配置的选项有：

convertErrorsToExceptions 默认情况下，PHPUnit 将会安插一个错误处理函数来将以下错误转换为异常：

- E_WARNING
- E_NOTICE
- E_USER_ERROR
- E_USER_WARNING
- E_USER_NOTICE
- E_STRICT
- E_RECOVERABLE_ERROR
- E_DEPRECATED
- E_USER_DEPRECATED

	将 convertErrorsToExceptions 设为 false 可以禁用此功能。
convertNoticesToExceptions	此选项设置为 false 时，由 convertErrorsToExceptions 安插的错误处理函数不会将 E_NOTICE、E_USER_NOTICE、E_STRICT 错误转换为异常。
convertWarningsToExceptions	此选项设置为 false 时，由 convertErrorsToExceptions 安插的错误处理函数不会将 E_WARNING 或 E_USER_WARNING 错误转换为异常。
forceCoversAnnotation	只记录使用了 @covers 标注（文档参见“@covers”一节）的测试的代码覆盖率。
timeoutForLargeTests	如果实行了基于测试规模的时间限制，那么此属性为所有标记为 @large 的测试设定超时限制。在配置的时间限制内未执行完毕的测试将视为失败。
timeoutForMediumTests	如果实行了基于测试规模的时间限制，那么此属性为所有标记为 @medium 的测试设定超时限制。在配置的时间限制内未执行完毕的测试将视为失败。
timeoutForSmallTests	如果实行了基于测试规模的时间限制，那么此属性为所有未标记为 @medium 或 @large 的测试设定超时限制。在配置的时间限制内未执行完毕的测试将视为失败。

测试套件

带有一个或多个 <testsuite> 子元素的 <testsuites> 元素用于将测试套件及测试用例组合出新的测试套件。

```
<testsuites>
  <testsuite name="My Test Suite">
    <directory>/path/to/*Test.php files</directory>
    <file>/path/to/MyTest.php</file>
    <exclude>/path/to/exclude</exclude>
  </testsuite>
</testsuites>
```

可以用 phpVersion 和 phpVersionOperator 属性来指定 PHP 版本需求。在以下例子中，仅当 PHP 版本至少为 5.3.0 时才会将 /path/to/*Test.php 文件与 /path/to/MyTest.php 文件添加到测试套件中。

```
<testsuites>
    <testsuite name="My Test Suite">
      <directory suffix="Test.php" phpVersion="5.3.0" phpVersionOperator=">=">/path/to/f
      <file phpVersion="5.3.0" phpVersionOperator=">=">/path/to/MyTest.php</file>
    </testsuite>
  </testsuites>
```

phpVersionOperator 属性是可选的，其默认值为 >=。

分组

<groups> 元素及其 <include>、<exclude>、<group> 子元素用于从带有 @group 标注（相关文档参见 “@group”一节）的测试中选择需要运行（或不运行）的分组。

```
<groups>
  <include>
    <group>name</group>
  </include>
  <exclude>
    <group>name</group>
  </exclude>
</groups>
```

以上 XML 配置对应于以如下选项调用 TextUI 测试执行器：

- `--group name`
- `--exclude-group name`

Whitelisting Files for Code Coverage

`<filter>` 元素及其子元素用于配置代码覆盖率报告所使用的白名单。

```
<filter>
  <whitelist processUncoveredFilesFromWhitelist="true">
    <directory suffix=".php">/path/to/files</directory>
    <file>/path/to/file</file>
    <exclude>
      <directory suffix=".php">/path/to/files</directory>
      <file>/path/to/file</file>
    </exclude>
  </whitelist>
</filter>
```

Logging（日志记录）

`<logging>` 元素及其 `<log>` 子元素用于配置测试执行期间的日志记录。

```
<logging>
  <log type="coverage-html" target="/tmp/report" lowUpperBound="35"
       highLowerBound="70"/>
  <log type="coverage-clover" target="/tmp/coverage.xml"/>
  <log type="coverage-php" target="/tmp/coverage.serialized"/>
  <log type="coverage-text" target="php://stdout" showUncoveredFiles="false"/>
  <log type="junit" target="/tmp/logfile.xml" logIncompleteSkipped="false"/>
  <log type="testdox-html" target="/tmp/testdox.html"/>
  <log type="testdox-text" target="/tmp/testdox.txt"/>
</logging>
```

以上 XML 配置对应于以如下选项调用 TextUI 测试执行器：

- `--coverage-html /tmp/report`
- `--coverage-clover /tmp/coverage.xml`
- `--coverage-php /tmp/coverage.serialized`
- `--coverage-text`
- `> /tmp/logfile.txt`
- `--log-junit /tmp/logfile.xml`
- `--testdox-html /tmp/testdox.html`

- `--testdox-text /tmp/testdox.txt`

`lowUpperBound`、`highLowerBound`、`logIncompleteSkipped` 及 `showUncoveredFiles` 属性没有等价的 TextUI 测试执行器选项。

- `lowUpperBound`：视为“低”覆盖率的最大覆盖率百分比。
- `highLowerBound`：视为“高”覆盖率的最小覆盖率百分比。
- `showUncoveredFiles`：在 `--coverage-text` 输出中显示所有符合白名单的文件，不仅限于有覆盖率信息的那些。
- `showOnlySummary`：在 `--coverage-text` 输出中只显示摘要。

测试监听器

`<listeners>` 元素及其 `<listener>` 子元素用于在测试执行期间附加额外的测试监听器。

```
<listeners>
  <listener class="MyListener" file="/optional/path/to/MyListener.php">
    <arguments>
      <array>
        <element key="0">
          <string>Sebastian</string>
        </element>
      </array>
      <integer>22</integer>
      <string>April</string>
      <double>19.78</double>
      <null/>
      <object class="stdClass"/>
    </arguments>
  </listener>
</listeners>
```

以上 XML 配置对应于将 `$listener` 对象（见下文）附到测试执行过程上。

```
$listener = new MyListener(
    ['Sebastian'],
    22,
    'April',
    19.78,
    null,
    new stdClass
);
```

设定 PHP INI 设置、常量、全局变量

`<php>` 元素及其子元素用于配置 PHP 设置、常量以及全局变量。同时也可用于向 `include_path` 前部置入内容。

```
<php>
  <includePath>.</includePath>
  <ini name="foo" value="bar"/>
  <const name="foo" value="bar"/>
  <var name="foo" value="bar"/>
  <env name="foo" value="bar"/>
  <post name="foo" value="bar"/>
  <get name="foo" value="bar"/>
  <cookie name="foo" value="bar"/>
```

```
  <server name="foo" value="bar"/>
  <files name="foo" value="bar"/>
  <request name="foo" value="bar"/>
</php>
```

以上 XML 配置对应于如下 PHP 代码：

```
ini_set('foo', 'bar');
define('foo', 'bar');
$GLOBALS['foo'] = 'bar';
$_ENV['foo'] = 'bar';
$_POST['foo'] = 'bar';
$_GET['foo'] = 'bar';
$_COOKIE['foo'] = 'bar';
$_SERVER['foo'] = 'bar';
$_FILES['foo'] = 'bar';
$_REQUEST['foo'] = 'bar';
```

附录 D. 索引

索引

符号

A

C

D

E

F

G

H

I

L

M

O

P

R

S

T

W

X

附录 E. 参考书目

[Astels2003] *Test Driven Development*. David Astels. 版权 © 2003. Prentice Hall. ISBN 0131016490.

[Beck2002] *Test Driven Development by Example*. Kent Beck. 版权 © 2002. Addison-Wesley. ISBN 0-321-14653-0.

[Meszaros2007] *xUnit Test Patterns: Refactoring Test Code*. Gerard Meszaros. 版权 © 2007. Addison-Wesley. ISBN 978-0131495050.

附录 F. 版权

```
Copyright (c) 2005-2017 Sebastian Bergmann.

此作品依照 Creative Commons Attribution 3.0
Unported License 授权。

以下是此授权许可协议的摘要信息，完整的法律文件附在其后。

----------------------------------------------------------------------

您可以自由地:

    * 分享 - 复制、分发、传播此作品
    * 重组 - 创作演绎此作品

惟须遵守下列条件:

姓名标示。弄必须按照作者或者版权人指定的方式表彰其姓名（但不得以任何方式暗示他们认可你或你使用本作品的方式

    * 在再使用或者发行本作品时，您必须向他人明示本作品使用的许可协议条款。明示的最佳方法是附上本网页的链

    * 若您获得著作权人准许，则上述所有条件都可予以免除。

    * 此协议对作者的人身权不构成任何损害与限制。

合理使用及其他权利不受许可协议影响。

以上是易于常人了解的法律条文（完整的授权许可协议）摘要。

======================================================================

Creative Commons Legal Code
Attribution 3.0 Unported

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN
ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO
WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS
LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE
WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW
IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND
AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS
LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU
THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
SUCH TERMS AND CONDITIONS.

1. Definitions

   a. "Adaptation" means a work based upon the Work, or upon the
      Work and other pre-existing works, such as a translation,
      adaptation, derivative work, arrangement of music or other
      alterations of a literary or artistic work, or phonogram or
```

performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.

d. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;

b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";

c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,

d. to Distribute and Publicly Perform Adaptations.

e. For the avoidance of doubt:

i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

ii. Waivable Compulsory License Schemes. In those

jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,

iii. Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(b), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(b), as requested.

   b. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv), consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the

Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses.

Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons is not a party to this License, and makes no

```
warranty whatsoever in connection with the Work. Creative Commons
will not be liable to You or any party on any legal theory for any
damages whatsoever, including without limitation any general,
special, incidental or consequential damages arising in connection
to this license. Notwithstanding the foregoing two (2) sentences,
if Creative Commons has expressly identified itself as the Licensor
hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the
Work is licensed under the CCPL, Creative Commons does not authorize
the use by either party of the trademark "Creative Commons" or any
related trademark or logo of Creative Commons without the prior
written consent of Creative Commons. Any permitted use will be in
compliance with Creative Commons' then-current trademark usage
guidelines, as may be published on its website or otherwise made
available upon request from time to time. For the avoidance of
doubt, this trademark restriction does not form part of this
License.

Creative Commons may be contacted at http://creativecommons.org/.

======================================================================
```