Метки

.Net CSS Drupal Eclipse Fedora HTTP Java JavaDB Linux MySQL Oracle Oracle Linux PHP PHPUnit PL/SQL QNX Smarty SWT Twitter UAC UTF8 Web Windows7 XML xUnit Zend Framework Безопасность Книги ОСи разное Регулярные выражения СУБД Тестирование

PHPUnit. Часть 04 Тестовые окружения (Fixtures)

Статьи -> Программирование -> PHP

PHPUnit. Часть 04 Тестовые окружения (Fixtures)

v:1.0 30.03.2010

Перевод статьи Chapter 6. Fixtures.
Автор: Sebastian Bergmann
Перевод: Петрелевич Сергей

Предисловие переводчика
Эта статья продолжает серию переводов официальной документации по PHPUnit на русский язык.
Часть 1, Часть 2, Часть 3,

Установка параметров тестовой среды или, другими словами, создание тестируемого мира - это одна из самых трудоемких задач. А ведь после завершения теста всем переменным надо вернуть первоначальные значения - эта задача тоже не из простых. Тестируемый мир или параметры тестируемой среды называются - *тестовое окружение (fixture)*.

В Примере 4.1 тестовое окружение было простым массивом, который сохранялся в переменную `$stack`. Однако, чаще всего тестовое окружение оказывается значительно сложнее, и количество кода, для работы с ним растет соответственно. Содержание теста может потеряться в шуме кода, отвечающего за работу с тестовым окружением. Все станет совсем плохо в тот момент, когда Вы напишите несколько тестов, использующих похожие тестовые окружения. Нам явно необходима помощь среды тестирования (framework), если мы ходим избавиться от многочисленного дублирования кода.

PHPUnit поддерживает совместное использование кода установки тестового окружения.
До того как начнет выполняться метод тестирования, будет вызван шаблонный метод `setUp()`.
Как только метод тестирования завершит свою работу будет вызван другой шаблонный метод - `tearDown()`.
Причем его вызов не зависит от того успешно ли завершился тест или нет.

В Примере 4.2 мы применяем подход "источник-приемник" для совместного использования тестового окружения. Но это не всегда приемлемо, а часто и вовсе невозможно.
Пример 6.1 демонстрирует как можно написать тест `StackTest` таким образом, чтобы повторно использовать не тестовое окружение, а код, который его создает.
Прежде всего, мы объявляем переменную класса `$stack`, которую мы будем использовать вместо локальной переменной метода.
После этого мы переместим создание тестового окружения в метод `setUp()`. В заключении мы уберем ставший уже ненужным код из тестовых методов и начнем использовать новую переменную класса `$this->stack`, вместо локальной переменной метода `$stack`.

Пример 6.1: Использование setUp() для создания тестового окружения тестирования стека

```
<?php
class StackTest extends
```

```
PHPUnit_Framework_TestCase
{
    protected $stack;

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

    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));
    }
}
?>

* This source code was highlighted with Source Code Highlighter.
```

Шаблонные методы `setUp()` и `tearDown()` вызываются по одному разу для каждого тестового метода (и для нового экземпляра) тестового класса.

И дополнительно, шаблонные методы `setUpBeforeClass()` и `tearDownAfterClass()` вызываются прежде, чем будет выполнен первый метод тестового класса и после того как последний метод будет завершен.

В следующем примере демонстрируются все возможные шаблонные методы, которые доступны в тестовом классе.

Пример 6.2: Пример демонстрирует применение всех возможных шаблонным методов

```
<?php
require_once 'PHPUnit/Framework.php';

class TemplateMethodsTest extends
PHPUnit_Framework_TestCase
{
    public static function
setUpBeforeClass()
    {
        print __METHOD__ . "\n";
    }

    protected function setUp()
    {
        print __METHOD__ . "\n";
    }

    protected function
assertPreConditions()
    {
        print __METHOD__ . "\n";
    }
```

```
        public function testOne()
        {
            print __METHOD__ . "\n";
            $this->assertTrue(TRUE);
        }

        public function testTwo()
        {
            print __METHOD__ . "\n";
            $this->assertTrue(FALSE);
        }

        protected function
    assertPostConditions()
        {
            print __METHOD__ . "\n";
        }

        protected function tearDown()
        {
            print __METHOD__ . "\n";
        }

        public static function
    tearDownAfterClass()
        {
            print __METHOD__ . "\n";
        }

        protected function
    onNotSuccessfulTest(Exception $e)
        {
            print __METHOD__ . "\n";
            throw $e;
        }
    }
    ?>

    * This source code was highlighted with Source Code Highlighter.
```

```
phpunit TemplateMethodsTest PHPUnit 3.4.2 by
Sebastian Bergmann.
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 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.
```

Практика применения setUp() и tearDown()

`setUp()` и `tearDown()` теоретически должны быть полностью симметричны, однако на практике это ни так. На практике, Вы обязательно должны вызвать `tearDown()`, если в `setUp()` открыли какой-нибудь внешний ресурс, например, сокет или файл. Если `setUp()` создает только объекты PHP, `tearDown()` можно игнорировать. Однако, если в `setUp()` создается множество объектов, разумно в `tearDown()` поместить вызовы `unset()` для созданных объектов. Таким образом, `tearDown()` выполнит функцию сборщика мусора.
Сборка мусора для объектов тестовых методов

практически не предсказуема.

Вариации

Что получится, если у Вас будет два теста с немного разными тестовыми окружениями? Возможны два варианта:

- Если функции `setUp()` отличаются незначительно, перенесите специфический код из `setUp()` в тестовые методы.
- Если функции `setUp()` сильно отличаются, то надо создать другой тестовый класс. Назовите этот новый класс по аналогии с отличиями в тестовом окружении.

Совместное использование тестовых окружений

Есть несколько веских причин совместного использования тестового окружения несколькими тестами, однако в большинстве случает подобная необходимость является следствием проблем в архитектуре приложения.

Хороший пример совместного использования тестового окружения несколькими тестами - соединение с базой данных. Соединение с базой устанавливается только один раз, и все тесы используют это соединение, вместо того чтобы для каждого теста создавать новое.
Эта практика позволяет ускорить выполнение тестов.

В Примере 6.3 показано как использовать шаблонные методы `setUp()` и `tearDown()` класса `PHPUnit_Framework_TestSuite` (см. раздел Использование класса TestSuite) для подключения к базе данных до выполнения первого теста тестового набора и отключения от базы после выполнения последнего теста. Атрибут `$sharedFixture` объекта `PHPUnit_Framework_TestSuite` доступен во всех объектах классов, унаследованных от `PHPUnit_Framework_TestSuite` и `PHPUnit_Framework_TestCase`.

Пример 6.3: Совместное использование тестового окружения несколькими тестами тестового набора

```
<?php
require_once 'PHPUnit/Framework.php';

class DatabaseTestSuite extends
PHPUnit_Framework_TestSuite
{
    protected function setUp()
    {
        $this->sharedFixture = new PDO(
         'mysql:host=wopr;dbname=test',
         'root',
         ''
        );
    }

    protected function tearDown()
    {
        $this->sharedFixture = NULL;
    }
}
?>

* This source code was highlighted with Source Code Highlighter.
```

Сокращение количества тестов за счет применения совместного использования тестового окружения должно настораживать. Это может говорить о наличии скрытой пробемы в архитектуре - объекты излишне взаимосвязаны. Вы получите значительно лучший результат, если сначала решите эту скрытую проблему, а только потом напишите тесты, используя заглушки (stubs) (см. Глава 11). Это намного лучше, чем создание зависимостей между

тестами и игнорирование возможности улучшить архитектуру.

Глобальное состояние

Очень трудно тестировать код, который использует паттерн singleton *(В приложении создается только один экземпляр класса. Таким образом реализуется функционал глобальных переменных. Прим. переводчика)*. Это утверждение справедливо и для случаев применения глобальных переменных. Часто, код, который Вы хотите протестировать сильно связан с глобальными переменными, создание которых Вы не можете контролировать. Другая проблема заключается в том, что один тест может изменить значение глобальной переменной, из-за чего сломается другой тест.

В PHP, гобальные переменные работают так:

- Глобальная переменная `$foo = 'bar';` сохраняется как `$GLOBALS['foo'] = 'bar';`.
- Переменная `$GLOBALS` по другому называется *суперглобальная*.
- Суперглобальные переменные - это встроенные переменные, которые доступны во всех модулях приложения.
- В пределах функции или метода Вы можете получить доступ к глобальной переменной `$foo` или используя прямой доступ `$GLOBALS['foo']`, или применив ключевое слово `global $foo;` для создания локальной переменной, которая будет связана с глобальной.

Кроме глобальных переменных к глобальным состояниям относятся еще и атрибуты классов.

По умолчанию, PHPUnit выполняет Ваши тесты таким образом, что изменение глобальных и суперглобальных переменных (`$GLOBALS`, `$_ENV`, `$_POST`, `$_GET`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_REQUEST`) не влияет на другие тесты. Дополнительно, эта изоляция может быть расширена и на статические атрибуты классов.

Примечание

Применение операций резервного хранения и восстановления статических атрибутов классов требует PHP 5.3 (или выше).

Операции резервного хранения и восстановления глобальных переменных и статических атрибутов классов используют функции `serialize()` и `unserialize()`.

Объекты некоторых классов, которые входят в состав PHP, например `PDO`, не могут быть сохранены и восстановлены. Попытка сохранить подобный объект в массив `$GLOBALS` завершится с ошибкой.

Операциями сохранения и восстановления глобальных переменных можно управлять при помощи аннотации `@backupGlobals`, см. раздел `@backupGlobals`. В качестве альтернативы, Вы можете составить список переменных, с которыми операции сохранения и восстановления не должны работать, см. код ниже:

```
class MyTest extends PHPUnit_Framework_TestCase {
protected $backupGlobalsBlacklist =
array('globalVariable'); // ... }
```

Примечание

Пожалуйста, учтите, что установка атрибута `$backupGlobalsBlacklist` внутри метода, например, `setUp()` не будет иметь эффект.

Операциями сохранения и восстановления статических атрибутов можно управлять при помощи аннотации

`@backupStaticAttributes`, см. раздел `@backupStaticAttributes` В качестве альтернативы, Вы можете составить список статических атрибутов, с которыми операции сохранения и восстановления не должны работать, см. код ниже:

```
class MyTest extends PHPUnit_Framework_TestCase {
protected $backupStaticAttributesBlacklist = array(
'className' => array('attributeName') ); // ... }
```

Примечание

Пожалуйста, учтите, что установка атрибута `$backupStaticAttributesBlacklist` внутри метода, например, `setUp()` не будет иметь эффект.

Метки: PHP Web PHPUnit Тестирование

Комментарии.

Внимание.
Комментировать могут только зарегистрированные пользователи.
Возможно использование следующих HTML тегов: <a>, <b>, <i>,
.

djoa 28.08.2012 13:43:51

Здравствуйте! Спасибо за перевод, очень актуально. До сих пор сложно найти в интернете статьи про практическое использование PhpUnit, на русском. У меня к вам вопрос: почему-то при запуске примера 6.2 у меня выводится такая последовательность:
StepsTest::setUpBeforeClass
.StepsTest::setUp
StepsTest::assertPreConditions
StepsTest::testOne
StepsTest::assertPostConditions
StepsTest::tearDown
StepsTest::onNotSuccessfulTest
FStepsTest::setUp
StepsTest::assertPreConditions
StepsTest::testTwo
StepsTest::tearDown
StepsTest::tearDownAfterClass Т. е. onNotSuccessfulTest вообще непонятно как выводится. Можете подсказать, в чём может быть проблема?

djoa 28.08.2012 13:44:23

И ещё. У вас ссылки "авторизоваться" и "зарегистрироваться" внизу статьи ведут не на те урлы (видимо, девелоперская версия): http://smartyit_dev/users/login и http://smartyit_dev/users/registration, соответственно.

Sergey 28.08.2012 22:05:57

djoa
Спасибо, что заметили ошибку в ссылках и сообщили:)
А PHPUnit'ом я уже давно не пользуюсь, забыл что там и как, поэтому на Ваш вопрос ответить не могу, сорри...

Пожалуйста авторизуйтесь или зарегистрируйтесь.