

## Содержание

- Новости
- Идея сайта
- ▼ Статьи
  - ▼ Операционные системы
    - Linux
    - QNX
  - ▼ Программирование
    - ▼ Java
      - Примеры
    - PHP
    - Web
    - Zend Framework
    - Тестирование
  - ▼ СУБД
    - MySQL
    - Oracle
- Наши работы
- Ссылки

## 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 03 Написание тестов для PHPUnit

Статьи -> Программирование -> PHP

### PHPUnit. Часть 03 Написание тестов для PHPUnit

v:1.0 25.03.2010

Перевод статьи [Chapter 4. Writing Tests for PHPUnit](#).

Автор: [Sebastian Bergmann](#)

Перевод: [Петрелевич Сергей](#)

*Предисловие переводчика*

*Эта статья продолжает серию переводов официальной документации по PHPUnit на русский язык.*

*Часть 1, Часть 2*

**Пример 4.1** демонстрирует как с помощью PHPUnit можно выполнить тестирование операций с массивами PHP. В этом примере показаны базовые соглашения и шаги, свойственные тестам PHPUnit:

- Наименование тестирующего класса образуется путем добавления постфикса `Test` к наименованию тестируемого класса. Например тестируемый класс называется `Class`, тестирующий - `ClassTest`.
- `ClassTest` наследуется (в большинстве случаев) от `PHPUnit_Framework_TestCase`.
- Наименования тестирующих методов образуются путем добавления приставки `test` к наименованиям тестируемых методов.
- Внутри тестовых методов утверждения задаются специальными функциями `assertEquals()` (смотрите [раздел PHPUnit\\_Framework\\_Assert](#)). `assertEquals()` задает соответствие реально полученного значения и ожидаемого.

### Пример 4.1: Тестирование операций с массивами PHP при с использованием PHPUnit

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

class StackTest extends
PHPUnit_Framework_TestCase
{
    public function testPushAndPop()
    {
        $stack = array();
        $this->assertEquals(0,
count($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));
    }
}
```

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

Мой блог в Живом Журнале  
Мой блог на Хабрахабр

Рассылки <b>Subscribe.Ru</b>
Килограмм килобайтов
ваш e-mail <input type="text"/> <input type="button" value="OK"/>
<a href="#">Подписаться письмом</a>



"Пишите тест в независимости от того хотите ли Вы вывести значение через функцию `print` или выражение отладки." Martin Fowler

## Зависимости тестов

*"Unit-тесты обычно пишутся, чтобы помочь разработчику найти и исправить ошибки, выполнить рефакторинг кода и облегчить документирование модулей. Чтобы достичь эти цели, тесты в идеале должны покрывать все возможные пути в программе. Обычно один тест покрывает один специфический путь в одной функции или методе. Однако тестовый метод не обязательно может быть инкапсулированной, независимой сущностью. Часто существуют не очевидные зависимости между тестами, скрытые в реализации тестовых сценариев". Adrian Kuhn et. al.*

PHPUnit поддерживает декларирование явных зависимостей между тестами. Такие зависимости не определяют в какой последовательности тесты должны выполняться, однако они позволяют возвращать экземпляр тестового окружения (fixture) и передавать его в другой тест; источник (producer) передает приемнику (consumer).

- Источник - это тестовый метод, который возвращает значения, от которых зависят другие методы модуля.
- Приемник - это тестовый метод, который зависит от одного или более источников и их возвращаемых значений.

Пример 4.2 показывает как использовать аннотацию `@depends` для описания зависимостей между тестами.

### Пример 4.2: Использование аннотации `@depends` для описания зависимостей

```
<?php
class StackTest extends
PHPUnit_Framework_TestCase
{
    public function testEmpty()
    {
        $stack = array();
        $this->assertTrue(empty($stack));

        return $stack;
    }

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

        return $stack;
    }

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

```
}  
?>
```

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

В этом примере первый тест, `testEmpty()`, создает пустой массив и задает утверждение, что массив пустой. После этого тест возвращает окружение (fixture) в качестве результата. Второй тест, `testPush()`, зависит от `testEmpty()` и получает результат работы `testEmpty()` в качестве аргумента. И наконец, `testPop()` зависит от `testPush()`.

Мы хотим, чтобы наше внимание фокусировалось на важных сообщениях о провале тестов, это позволит максимально быстро находить дефекты. По этой причине PHPUnit пропускает выполнение зависимых тестов, если Источник (основной тест) закончил работу с ошибкой. Улучшение обнаружения дефектов достигается за счет использования зависимостей между тестами, как показано в [Примере 4.3](#).

#### Пример 4.3: Использование зависимостей между тестами

```
<?php  
class DependencyFailureTest extends  
    PHPUnit_Framework_TestCase  
{  
    public function testOne()  
    {  
        $this->assertTrue(FALSE);  
    }  
  
    /**  
     * @depends testOne  
     */  
    public function testTwo()  
    {  
    }  
}  
?>
```

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

```
phpunit --verbose DependencyFailureTest PHPUnit  
3.4.2 by Sebastian Bergmann. DependencyFailureTest  
FS Time: 0 seconds There was 1 failure: 1)  
testOne(DependencyFailureTest) Failed asserting  
that is true. /home/sb/DependencyFailureTest.php:6  
There was 1 skipped test: 1)  
testTwo(DependencyFailureTest) This test depends on  
"DependencyFailureTest::testOne" to pass. FAILURES!  
Tests: 2, Assertions: 1, Failures: 1, Skipped: 1.
```

У теста может быть несколько аннотаций `@depends`. PHPUnit не изменяет последовательность, в которой запускаются тесты, Вы должны быть уверены, что зависимости будут выполнены до того как тест запустится.

### Источники данных (Data Providers)

Тестовый метод может работать с произвольными аргументами. Аргументы передаются с помощью метода источника данных (`provider()`, см. [Пример 4.4](#)). Метод источника данных должен быть определен с помощью аннотации `@dataProvider`.

Метод источника данных должен быть `public` и должен возвращать массив массивов или объект, поддерживающий интерфейс `Iterator`, который на каждой итерации возвращает массив. Для каждого массива, который является частью коллекции, будет вызван тестовый метод. В качестве аргумента методу будет передан массив значений.

#### Пример 4.4: Использование источника данных

```
<?php
class DataTest extends
PHPUnit_Framework_TestCase
{
    /**
     * @dataProvider provider
     */
    public function testAdd($a, $b, $c)
    {
        $this->assertEquals($c, $a + $b);
    }

    public function provider()
    {
        return array(
            array(0, 0, 0),
            array(0, 1, 1),
            array(1, 0, 1),
            array(1, 1, 3)
        );
    }
}
?>
```

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

```
phpunit DataTest PHPUnit 3.4.2 by Sebastian
Bergmann. ...F Time: 0 seconds There was 1 failure:
1) testAdd(DataTest) with data (1, 1, 3) Failed
asserting that matches expected value .
/home/sb/DataTest.php:21 FAILURES! Tests: 4,
Assertions: 4, Failures: 1.
```

#### Примечание

Если тестовому методу одновременно передаются параметры от источника данных (@dataProvider) и одного или нескольких тестов, которые определены как зависимые (@depends), то в первую очередь используется источник данных и только потом другие тесты.

#### Тестирование исключений

Пример 4.5 демонстрирует как использовать аннотацию @expectedException для тестирования исключений, которые выбрасываются внутри тестового кода.

#### Пример 4.5: Использование аннотации @expectedException

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

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

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

```
phpunit ExceptionTest PHPUnit 3.4.2 by Sebastian
Bergmann. F Time: 0 seconds There was 1 failure: 1)
testException(ExceptionTest) Expected exception
InvalidArgumentException FAILURES! Tests: 1,
Assertions: 1, Failures: 1.
```

Метод `setExpectedException()` - это еще один способ указать, что в тестовом методе выбрасывается исключение, см. [Пример 4.6](#).

**Пример 4.6: Ожидается, что тестовый метод выбросит исключение**

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

class ExceptionTest extends
PHPUnit_Framework_TestCase
{
    public function testException()
    {
        $this->
setExpectedException('InvalidArgumentException');
    }
}

?>
```

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

**phpunit ExceptionTest** PHPUnit 3.4.2 by Sebastian Bergmann. F Time: 0 seconds There was 1 failure: 1) testException(ExceptionTest) Expected exception InvalidArgumentException FAILURES! Tests: 1, Assertions: 1, Failures: 1.

В [таблице 4.1](#) приведены методы для тестирования исключений.

**Таблица 4.1. Методы для тестирования исключений**

Метод	Назначение
void setExpectedException(string \$exceptionName)	Установка имени ожидаемого исключения - \$exceptionName.
String getExpectedException()	Получение имени ожидаемого исключения.

Для тестирования исключений Вы можете использовать подход, показанный в [Примере 4.7](#).

**Пример 4.7: Альтернативный подход к тестированию исключений**

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

class ExceptionTest extends
PHPUnit_Framework_TestCase {
    public function testException() {
        try {
            // ... Code that is expected
            to raise an exception ...
        }

        catch (InvalidArgumentException
$expected) {
            return;
        }

        $this->fail('An expected
exception has not been raised.');
```

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

Ожидается, что код, показанный в [Примере 4.7](#) выбросит исключение. Если это не произойдет, то будет вызван

метод `fail()`, (см. (см. [Таблицу 22.2](#)), который прервет выполнение теста и просигнализирует об ошибке. Если исключение будет сгенерировано, то сработает блок `catch` и тест завершится успешно.

## Тестирование ошибок PHP

По умолчанию, ошибки, предупреждения и уведомления PHP, которые появляются во время тестирования, PHPUnit преобразует в исключения. Используя эту особенность, Вы можете, например, настроить механизм ожидания появления подобного исключения в тесте, см. [Пример 4.8](#).

### Пример 4.8: Применение `@expectedException` для ожидания ошибки PHP

```
<?php
class ExpectedErrorTest extends
PHPUnit_Framework_TestCase
{
    /**
     * @expectedException
     PHPUnit_Framework_Error
     */
    public function testFailingInclude()
    {
        include 'not_existing_file.php';
    }
}
?>
```

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

```
phpunit ExpectedErrorTest PHPUnit 3.4.2 by
Sebastian Bergmann. . Time: 0 seconds OK (1 test, 1
assertion)
```

PHPUnit\_Framework\_Error\_Notice и  
PHPUnit\_Framework\_Error\_Warning представляют PHP  
уведомления и предупреждения.

**Метки:** [PHP](#) [Web](#) [PHPUnit](#) [Тестирование](#)

---

Комментарии.

#### Внимание.

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

Пожалуйста [авторизуйтесь](#) или [зарегистрируйтесь](#).

