PHPUnit Manual

Выпуск latest

Sebastian Bergmann

09.09.21 10:39:13

Оглавление

Документация на русском языке для PHPUnit версии latest. Обновлено: 09.09.21 10:39:05.

Себастьян Бергман (Sebastian Bergmann)

> Перевод документации на русский язык вышел 10 июля 2018 года, поэтому возможны опечатки и неточности. Пожалуйста, если вы найдете их, создайте ишью или пулреквест с исправлением.
>
> Если вы новичок в тестировании, либо желаете поддержать перевод документации, то вы можете купить книгу рецептов PHPUnit.
>
> Также перед чтением документации ознакомьтесь с соглашением по переводу.
>
> (c) Алексей Пыльцын.

Содержание:

Глава 1

Установка PHPUnit

1.1 Требования

PHPUnit latest требует PHP 7.2; настоятельно рекомендуется использовать последнюю версию PHP.

PHPUnit требует расширений dom и json, которые обычно включены по умолчанию.

PHPUnit также требует расширений pcre, reflection и spl. Эти стандартные расширения включены по умолчанию и не могут быть отключены без внесения изменений в систему сборки PHP и/или в исходный код C.

Для функциональности отчёта по покрытию кода тестами требуются расширения Xdebug (2.7.0 или новее) и tokenizer. Генерация XML-отчётов требует расширения xmlwriter.

1.2 PHP Archive (PHAR)

Самый простой способ получить PHPUnit — загрузить PHP Archive (PHAR), в котором есть все необходимые (а также некоторые необязательные) зависимости PHPUnit, собранные в одном-единственном файле.

Расширение phar обязательно для использования PHP Archives (PHAR).

Если расширение Suhosin включено, вам необходимо разрешить выполнение PHAR в вашем `php.ini`:

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

PHAR с PHPUnit можно использовать сразу после загрузки:

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

Как правило, далее необходимо сделать исполняемым PHAR-файл:

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

1.2.1 Проверка релизов PHPUnit PHAR

Все официальные релизы кода, распространяемые проектом PHPUnit, подписываются релиз-менеджером. Подписи PGP и хеши SHA256 доступны для проверки на phar.phpunit.de.

В следующем примере показано, как работает проверка релиза. Мы начинаем с загрузки `phpunit.phar`, а также его отделённой подписи PGP `phpunit.phar.asc`:

```
$ wget https://phar.phpunit.de/phpunit-latest.phar
$ wget https://phar.phpunit.de/phpunit-latest.phar.asc
```

Мы хотим проверить PHP Archive (`phpunit-x.y.phar`) PHPUnit с его отделённой подписью (`phpunit-x.y.phar.asc`):

```
$ gpg phpunit-latest.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
```

У нас нет открытого ключа релиз-менеджера (`6372C20A`) в нашей локальной системе. Для продолжения проверки нам нужно получить открытый ключ релиз-менеджера с сервера ключей. Один из таких серверов — это `pgp.uni-mainz.de`. Серверы открытых ключей связаны между собой, поэтому вы можете подключиться к любому из них.

```
$ 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)
```

Теперь мы получили открытый ключ для сущности, известной как «Sebastian Bergmann <sb@sebastian-bergmann.de>». Однако, у нас нет способа проверить, что этот ключ был создан человеком под именем Себастьян Бергман (Sebastian Bergmann). Но давайте снова попробуем проверить подпись релиза.

```
$ gpg phpunit-latest.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
```

В данный момент подпись хорошая, но мы не доверяем этому ключу. Хорошая подпись означает, что файл не был изменён. Однако ввиду характера криптографии открытого ключа вам необходимо дополнительно проверить, что ключ `6372C20A` был создан настоящим Себастьяном Бергманом.

Любой злоумышленник может создать открытый ключ и загрузить его на серверы открытых серверов. Затем они могут создать вредоносный релиз, подписанный этим поддельным ключом. После чего, если вы попытаетесь проверить подпись этого испорченного релиза, проверка будет успешной, потому что

ключ не является «реальным» ключом. Поэтому вам нужно проверить подлинность этого ключа. Однако проверка подлинности открытого ключа выходит за рамки данной документации.

Проверка подлинности и целостности PHAR с PHPUnit вручную через GPG утомительна. Вот зачем нужен PHIVE (PHAR Installation and Verification Environment), среда установки и проверки PHAR. Вы можете узнать про PHIVE на сайте.

1.3 Composer

Просто добавьте (для разработки) зависимость `phpunit/phpunit` в файл `composer.json` вашего проекта, если вы используете Composer для управления зависимостями в вашем проекте:

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

1.4 Глобальная установка

Обратите внимание, что не рекомендуется устанавливать PHPUnit глобально, например, в `/usr/bin/phpunit` или `/usr/local/bin/phpunit`.

Вместо этого PHPUnit должен использоваться в виде локальной зависимости проекта.

Поэтому либо поместите PHAR определённой версии PHPUnit, которая вам нужна, в директорию `tools` вашего проекта (который должен управляться с помощью PHIVE), либо укажите конкретную версию PHPUnit в файле `composer.json` вашего проекта, если вы используете Composer.

Глава 2

Написание тестов на PHPUnit

Пример 2.1 показывает, как мы можем писать тесты, используя PHPUnit, которые выполняют операции с массивом PHP. В этом примере представлены основные соглашения и шаги для написания тестов с помощью PHPUnit:

1. Тесты для класса `Class` содержатся в классе `ClassTest`.
2. `ClassTest` наследуется (чаще всего) от `PHPUnit\Framework\TestCase`.
3. Тесты — общедоступные методы с именами `test*`.

   Кроме того, вы можете использовать аннотацию `@test` в докблоке метода, чтобы пометить его как метод тестирования.
4. Внутри тестовых методов для проверки того, соответствует ли фактическое значение ожидаемому используются методы-утверждения, такие как `assertSame()` (см. *Утверждения*).

Пример 2.1: Тестирование операций с массивами с использованием PHPUnit

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

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

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

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

Мартин Фаулер (Martin Fowler):

> Всякий раз, когда возникает соблазн что-то распечатать, используя `print`, или написать отладочное выражение, напишите тест вместо этого.

2.1 Зависимости тестов

> *Адриан Кун (Adrian Kuhn) и другие*:
>
> Модульные тесты главным образом пишутся в качестве хорошей практики, помогающей разработчикам выявлять и исправлять баги, проводить рефакторинг кода и служить в качестве документации для тестируемого программного модуля (программы). Для достижения этих преимуществ модульные тесты в идеале должны охватывать все возможные пути исполнения программы. Один модульный тест обычно покрывает один конкретный путь в одной функции или методе. Однако тестовые методы необязательно должны быть инкапсулированными и независимыми. Часто существуют неявные зависимости между тестовыми методами, скрытые в сценарии реализации теста.

PHPUnit поддерживает объявление явных зависимостей между тестовыми методами. Эти зависимости не определяют порядок, в котором должны выполняться тестовые методы, но они позволяют возвращать экземпляр (данные) фикстуры теста, созданные поставщиком (producer) для передачи его зависимым потребителям (consumers).

- Поставщик — тестовый метод, который предоставляет свой тестируемый модуль в качестве возвращаемого значения.
- Потребитель — тестовый метод, который зависит от одного или более поставщиков и их возвращаемых значений.

Пример 2.2 показывает, как использовать аннотацию `@depends` для представления зависимостей между тестовыми методами.

Пример 2.2: Использование аннотации `@depends` для описания зависимостей

```php
<?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->assertSame('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);
```

```
        return $stack;
    }

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

В вышеприведённом примере первый тест, `testEmpty()`, создаёт новый массив и утверждает, что он пуст. Затем тест возвращает фикстуру в качестве результата. Второй тест, `testPush()`, зависит от `testEmpty()` и ему передаётся результат этого зависимого теста в качестве аргумента. Наконец, `testPop()` зависит от `testPush()`.

Примечание

Возвращаемое значение, предоставленное поставщиком, по умолчанию передаётся потребителям «как есть». Это означает, что когда поставщик возвращает объект, ссылка на этот объект передаётся потребителям. Вместо ссылки возможна, либо (а) (глубокая) копия через `@depends clone` или (б) (поверхностная) копия (на основе ключевого слова PHP `clone`) через `@depends shallowClone`.

Чтобы быстро находить дефекты, нам нужно сосредоточить внимание на соответствующих неудачных тестах. Вот почему PHPUnit пропускает выполнение теста, когда зависимый тест (тест с зависимостью) провалился (не прошёл). Это помогает локализовать дефекты за счёт использования зависимостей между тестами, как это показано в Пример 2.3.

Пример 2.3: Использование зависимостей между тестами

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

class DependencyFailureTest extends TestCase
{
    public function testOne()
    {
        $this->assertTrue(false);
    }

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

```
$ phpunit --verbose DependencyFailureTest
PHPUnit latest.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.
```

У теста может быть несколько аннотаций `@depends`. PHPUnit не изменяет порядок выполнения тестов, поэтому вы должны убедиться, что все зависимости действительно могут быть выполнены до запуска теста.

Тест, содержащий более одной аннотации `@depends`, получит фикстуру от первого поставщика в качестве первого аргумента, фикстуру от второго поставщика вторым аргументом и т.д. См. Пример 2.4

Пример 2.4: Тест с несколькими зависимостями

```php
<?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($a, $b)
    {
        $this->assertSame('first', $a);
        $this->assertSame('second', $b);
    }
}
```

```
$ phpunit --verbose MultipleDependenciesTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

...
```

```
Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 4 assertions)
```

2.2 Провайдеры данных

Тестовый метод может принимать произвольное количество аргументов. Эти аргументы могут быть предоставлены одним или несколькими методами провайдеров данных (data provider) (см. `additionProvider()` в Пример 2.5). Метод, который будет использован в качестве провайдера данных, обозначается с помощью аннотации `@dataProvider`.

Метод провайдера данных должен быть объявлен как `public` и возвращать либо массив массивов, либо объект, реализующий интерфейс `Iterator` и возвращать массив при каждой итерации. Для каждого массива, являющегося частью коллекции, будет вызываться тестовый метод с элементами массива в качестве его аргументов.

Пример 2.5: Использование провайдера данных, который возвращает массив массивов

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

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

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

```
$ phpunit DataTest
PHPUnit latest.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 is identical to 3.
```

```
/home/sb/DataTest.php:9

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

При использовании большого количества наборов данных полезно указывать для каждого из них строковый ключ, вместо использования числового ключа по умолчанию. Вывод станет более подробным, так как он будет содержать имя набора данных, не прошедший тест.

Пример 2.6: Использование провайдера данных с наборами данных

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

class DataTest extends TestCase
{
    /**
     * @dataProvider additionProvider
     */
    public function testAdd($a, $b, $expected)
    {
        $this->assertSame($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 latest.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 is identical to 3.

/home/sb/DataTest.php:9

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

Пример 2.7: Использование провайдера данных, который возвращает объект Iterator

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

```php
require 'CsvFileIterator.php';

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

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

```
$ phpunit DataTest
PHPUnit latest.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 is identical to 3.

/home/sb/DataTest.php:11

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

Пример 2.8: Класс CsvFileIterator

```php
<?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++;
    }
}
```

Когда тест получает входные данные как из метода с `@dataProvider`, так и от одного или более методов с аннотацией `@depends`, первыми будут приходить аргументы от провайдера данных, а после от зависимых тестов. Аргументы от зависимых тестов будут одинаковыми для каждого набора данных. См. Пример 2.9

Пример 2.9: Комбинация @depends и @dataProvider в одном тесте

```
<?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->assertSame(
            ['provider1', 'first', 'second'],
            func_get_args()
        );
    }
}
```

```
$ phpunit --verbose DependencyAndDataProviderComboTest
PHPUnit latest.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 identical.
--- Expected
+++ Actual
@@ @@
Array &0 (
-    0 => 'provider1'
+    0 => 'provider2'
     1 => 'first'
     2 => 'second'
)
/home/sb/DependencyAndDataProviderComboTest.php:32

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

Пример 2.10: Использование нескольких провайдеров данных для одного теста :name: writing-tests-for-phpunit.data-providers.examples.DataTest.php

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

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

    public function additionWithNonNegativeNumbersProvider()
```

```
    {
        return [
            [0, 1, 1],
            [1, 0, 1],
            [1, 1, 3]
        ];
    }

    public function additionWithNegativeNumbersProvider()
    {
        return [
            [-1, 1, 0],
            [-1, -1, -2],
            [1, -1, 0]
        ];
    }
}
```

```
$ phpunit DataTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

..F...                                                              6 / 6 (100%)

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 (1, 1, 3)
Failed asserting that 2 is identical to 3.

/home/sb/DataTest.php:12

FAILURES!
Tests: 6, Assertions: 6, Failures: 1.
```

Примечание

Когда тест зависит от теста, который использует провайдеры данных, зависимый тест начнёт выполняться, когда тест, от которого тот зависит, успешно выполнится хотя бы для одного набора данных. Результат теста, который использует провайдеры данных, не может быть внедрён в зависимый тест.

Примечание

Все провайдеры данных выполняются как перед вызовом статического метода `setUpBeforeClass(): void`, так и перед первым вызовом метода `setUp(): void`. Из-за этого вы не сможете получить доступ к переменным, определённым внутри провайдера данных. Это требуется для того, чтобы PHPUnit смог вычислить общее количество тестов.

2.3 Тестирование исключений

Пример 2.11 показывает, как использовать метод `expectException()` для проверки того, было ли выброшено исключение в тестируемом коде.

Пример 2.11: Использование метода expectException()

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

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

```
$ phpunit ExceptionTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Failed asserting that exception of type "InvalidArgumentException" is thrown.

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

В дополнение к методу `expectException()` существуют методы `expectExceptionCode()`, `expectExceptionMessage()` и `expectExceptionMessageRegExp()`, чтобы установить ожидания для исключений, вызванных тестируемым кодом.

Примечание

Обратите внимание, что метод expectExceptionMessage, утверждает, что фактическое сообщение в `$actual` содержит ожидаемое сообщение `$expected` без выполнения точного сравнения строк.

Кроме того, вы можете использовать аннотации `@expectedException`, `@expectedExceptionCode`, `@expectedExceptionMessage` и `@expectedExceptionMessageRegExp`, чтобы установить ожидания для исключений, вызванных тестируемым кодом. Пример 2.12 демонстрирует пример использования.

Пример 2.12: Использование аннотации @expectedException

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

class ExceptionTest extends TestCase
{
    /**
     * @expectedException InvalidArgumentException
```

```
     */
    public function testException()
    {
    }
}
```

```
$ phpunit ExceptionTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Failed asserting that exception of type "InvalidArgumentException" is thrown.

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

2.4 Тестирование ошибок PHP

По умолчанию PHPUnit преобразует ошибки, предупреждения и уведомления, вызываемые PHP во время выполнения теста, в исключения. Используя эти исключения, вы можете, например, ожидать, что тест вызовет ошибку PHP, как показано в Пример 2.13.

Примечание

Конфигурация времени выполнения PHP `error_reporting` может ограничивать, какие ошибки PHPUnit будет конвертировать в исключения. Если у вас возникли проблемы с этой настройкой, убедитесь, что PHP не настроен на подавление типов ошибок, которые вы тестируете.

Пример 2.13: Ожидание ошибки PHP в тесте, используя @expectedException

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

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

```
$ phpunit -d error_reporting=2 ExpectedErrorTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

.
```

```
Time: 0 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

Классы `PHPUnit\Framework\Error\Notice` `PHPUnit\Framework\Error\Warning` представляют уведомления и предупреждения PHP соответственно.

Примечание

Вы должны как можно более конкретно указывать исключения при тестировании. Тестирование слишком общих классов исключений может привести к нежелательным побочным эффектам. Поэтому проверка исключения на соответствие классу `Exception` с помощью `@expectedException` или `expectException()` больше не разрешена.

При тестировании кода, использующего функции PHP, которые вызывают ошибки, например, `fopen`, иногда бывает полезно использовать подавление ошибок во время тестирования. Таким образом, это позволит вам проверять возвращаемые значения, подавляя уведомления, которые преобразуются в объекты PHPUnit `PHPUnit\Framework\Error\Notice`.

Пример 2.14: Тестирование возвращаемых значений в коде, в котором возникают ошибки PHP

```php
<?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 latest.0 by Sebastian Bergmann and contributors.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

Без подавления ошибки тест завершится неудачей с сообщением `fopen(/is-not-writeable/file):`

```
failed to open stream: No such file or directory.
```

2.5 Тестирование вывода

Иногда вам нужно проверить, что выполнение метода, например, генерирует ожидаемый вывод (к примеру, через `echo` или `print`). Класс `PHPUnit\Framework\TestCase` использует возможности буферизации вывода PHP для предоставления такой функциональности.

Пример 2.15 показывает, как использовать метод `expectOutputString()` для установки ожидаемого вывода. Если этот ожидаемый вывод не будет сгенерирован, тест будет считаться проваленным.

Пример 2.15: Тестирование вывода функции или метода

```
<?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 latest.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.
```

Таблица 2.1 показывает доступные методы для тестирования вывода

Таблица 2.1: Методы для тестирования вывода

Метод	Описание
`void expectOutputRegex(string $regularExpression)`	Проверить, что вывод соответствует регулярному выражению `$regularExpression`.
`void expectOutputString(string $expectedString)`	Проверить, что вывод соответствует строке `$expectedString`.
`bool setOutputCallback(callable $callback)`	Устанавливает функцию обратного вызова, используемую, например, для нормализации фактического вывода.
`string getActualOutput()`	Получить фактический вывод.

Примечание

Тест, который генерирует вывод, не будет работать в строгом режиме.

2.6 Вывод ошибки

Всякий раз, когда тест терпит неудачу, PHPUnit изо всех сил пытается предоставить вам максимально возможный контекст, который может помочь выявить проблему.

Пример 2.16: Вывод ошибки, сгенерированный при неудачном сравнении массива

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

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

```
$ phpunit ArrayDiffTest
PHPUnit latest.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 identical.
--- 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.
```

В этом примере только одно из значений массива отличается, а остальные значения показаны, для обеспечения контекста, где произошла ошибка.

Когда сгенерированный вывод будет длинным для чтения, PHPUnit разделит его и отобразит несколько строк контекста вокруг каждого несоответствия (разницы).

Пример 2.17: Вывод ошибки при неудачном сравнении длинного массива

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

class LongArrayDiffTest extends TestCase
{
    public function testEquality()
    {
        $this->assertSame(
            [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 latest.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 identical.
--- Expected
+++ Actual
@@ @@
     11 => 0
     12 => 1
     13 => 2
-    14 => 3
```

```
+    14 => 33
     15 => 4
     16 => 5
     17 => 6
 )

/home/sb/LongArrayDiffTest.php:7

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

2.6.1 Крайние случаи

Когда сравнение терпит неудачу, PHPUnit создаёт текстовые представления входных значений и сравнивает их. Благодаря этой реализации результат сравнения изменений (формат diff) может показать больше проблем, чем существуют на самом деле.

Это происходит только при использовании `assertEquals()` или „слабых“ („weak“) функций сравнения массивов или объектов.

Пример 2.18: Крайний случай в генерации сравнения при использовании слабого сравнения

```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 latest.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.
```

В этом примере сообщается о различии в первом индексе между `1` и `'1'`, хотя метод `assertEquals()` считает, что эти значения совпадают.

Глава 3

Исполнитель тестов командной строки

Исполнитель тестов командной строки PHPUnit можно запустить с помощью команды `phpunit`. Следующий пример показывает, как запускать тесты с помощью этого инструмента командной строки PHPUnit:

```
$ phpunit ArrayTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

..

Time: 0 seconds

OK (2 tests, 2 assertions)
```

При вводе команды, как показано выше, исполнитель тестов командной строки PHPUnit будет искать исходный файл `ArrayTest.php` в текущей рабочей директории, загрузит его с целью найти в нём класс теста `ArrayTest`. Затем он выполнит тесты этого класса.

Для каждого тестового запуска инструмент командной строки PHPUnit выведет один символ для обозначения прогресса:

`.`

Выводится, если тест успешно пройден.

`F`

Выводится, когда утверждение не проходит во время выполнения тестового метода.

`E`

Выводится, когда возникает ошибка во время запуска тестового метода.

`R`

Выводится, когда тест был отмечен как рискованный (см. *Рискованные тесты*).

`S`

Выводится, когда тест был пропущен (см. *Неполные и пропущенные тесты*).

`I`

Выводится, когда тест отмечен как незавершённый или ещё не реализован (см. *Неполные и пропущенные тесты*).

PHPUnit различает *неудачные выполнения (failures)* и *ошибки (errors)*. Неудачное выполнение - это непройденное утверждение PHPUnit, например вызов `assertSame()`. Ошибка — необработанное исключение или ошибка PHP. Иногда это различие оказывается полезным, поскольку ошибки гораздо легче исправить, чем неудачные выполнения. В случае большого списка проблем, лучше всего сначала устранить ошибки и посмотреть, остались ли ещё какие-либо неудачные выполнения, когда ошибки исправлены.

3.1 Опции командной строки

Давайте посмотрим на опции командной строки исполнителя тестов в следующем коде:

```
$ phpunit --help
PHPUnit latest.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
  --no-coverage               Ignore code coverage configuration
  --dump-xdebug-filter <file> Generate script to set Xdebug code coverage filter

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
  --list-tests                List available tests
```

```
  --list-tests-xml <file>     List available tests in XML format
  --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
  --default-time-limit=<sec>  Timeout in seconds for tests without @small, @medium or␣
↪@large
  --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-defect            Stop execution upon first not-passed test
  --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

  --resolve-dependencies      Resolve dependencies between tests
  --order-by=<order>          Run tests in order: default|defects|duration|no-
↪depends|random|reverse
  --random-order-seed=<N>     Use a specific random seed <N> for random order
  --cache-result              Write run result to cache to enable ordering tests defects-
↪first

Configuration Options:

  --prepend <file>            A PHP script that is included as early as possible
```

```
  --bootstrap <file>          A PHP script that is included before the tests run
  -c|--configuration <file>   Read configuration from XML file
  --no-configuration          Ignore default configuration file (phpunit.xml)
  --no-logging                Ignore logging 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
  --cache-result-file==<FILE> Specify result cache path and filename

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
  --check-version             Check whether PHPUnit is the latest version
```

`phpunit UnitTest`

Запускает тесты, представленные в классе `UnitTest`. Ожидается, что этот класс будет объявлен в исходном файле `UnitTest.php`.

`UnitTest` должен быть либо классом, который наследуется от `PHPUnit\Framework\TestCase`, либо классом с методом `public static suite()`, возвращающим объект типа `PHPUnit\Framework\Test`, например, экземпляр класса `PHPUnit\Framework\TestSuite`.

`phpunit UnitTest UnitTest.php`

Выполняет тесты в классе `UnitTest`. Ожидается, что этот класс будет объявлен в указанном исходном файле.

`--coverage-clover`

Генерирует файл логов в формате XML с информацией о покрытии кода тестами для выполненных тестов. См. *Логирование* для получения более подробной информации.

Обратите внимание, что данная функциональность доступна только в случае установленных расширений tokenizer и Xdebug.

`--coverage-crap4j`

Генерирует отчёт о покрытии кода тестами в формате Crap4j. См. *Анализ покрытия кода* для получения более подробной информации.

Обратите внимание, что данная функциональность доступна только в случае установленных расширений tokenizer и Xdebug.

`--coverage-html`

Генерирует отчёт о покрытии кода тестами в формате HTML. См. *Анализ покрытия кода* для получения более подробной информации.

Обратите внимание, что данная функциональность доступна только в случае установленных расширений tokenizer и Xdebug.

`--coverage-php`

Генерирует сериализованный объект класса `PHP_CodeCoverage` с информацией о покрытии кода тестами.

Обратите внимание, что данная функциональность доступна только в случае установленных расширений tokenizer и Xdebug.

`--coverage-text`

Генерирует файл логов или вывод командной строки в человекочитаемом формате с информацией о покрытии кода тестами для запуска тестов. См. *Логирование* для получения более подробной информации.

Обратите внимание, что данная функциональность доступна только в случае установленных расширений tokenizer и Xdebug.

`--log-junit`

Генерирует файл журнала (logfile) в формате JUnit XML для запуска тестов. См. *Логирование* для получения более подробной информации.

`--testdox-html` и `--testdox-text`

Генерирует agile-документацию в HTML или текстовом формате для запущенных тестов (см. *TestDox*).

`--filter`

Выполняются только те тесты, названия которых совпадают с регулярным выражением. Если регулярное выражение не заключёно в разделители, PHPUnit будет автоматически заключать его в разделители `/`.

Имена тестов для совпадения может быть в одном из следующих форматов:

`TestNamespace\TestCaseClass::testMethod`

Формат имени теста по умолчанию эквивалентен использованию магической константы `__METHOD__` внутри тестового метода.

`TestNamespace\TestCaseClass::testMethod with data set #0`

Когда в тесте есть провайдер данных, каждая итерация данных получает текущий индекс, добавленный в концу имени теста по умолчанию.

`TestNamespace\TestCaseClass::testMethod with data set "my named data"`

Когда в тесте есть провайдер данных, использующий именованные наборы, каждая итерация данных получает текущее название, добавленное к концу имени теста по умолчанию. См. Пример 3.1 для просмотра примера именованных наборов данных.

Пример 3.1: Именованные наборы данных

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

namespace TestNamespace;

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

    public function provider()
    {
```

```
        return [
            'my named data' => [true],
            'my data'       => [true]
        ];
    }
}
```

`/path/to/my/test.phpt`

Путь в файловой системе к имени теста типа PHPT.

См. Пример 3.2 для примеров корректных шаблонов фильтров.

Пример 3.2: Примеры шаблонов фильтров

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

См. Пример 3.3 для некоторых дополнительных сокращений, доступных для сопоставления с провайдерами данных.

Пример 3.3: Сокращения фильтра

```
--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`

Выполняется только тот тестовый набор, который совпадает с заданным шаблоном.

`--group`

Выполняются только тесты из указанных групп. Тест можно добавить в группу, используя аннотацию `@group`.

Аннотации `@author` и `@ticket` — это псевдонимы для `@group`, позволяющие фильтровать тесты по их авторам или по номерам связанных тикетов соответственно.

`--exclude-group`

Исключить тесты из указанных групп. Тест можно добавить в группу, используя аннотацию `@group`.

`--list-groups`

Список доступных групп тестов.

`--test-suffix`

Поиск тестовых файлов только с указанными суффиксами.

`--dont-report-useless-tests`

Не сообщать о тестах, которые ничего не тестируют. См. *Рискованные тесты* для получения подробной информации.

`--strict-coverage`

Строгая проверка непроизвольного охвата тестами кода. См. *Рискованные тесты* для получения подробной информации.

`--strict-global-state`

Строгая проверка относительно манипуляций с глобальным состоянием. См. *Рискованные тесты* для получения подробной информации.

`--disallow-test-output`

Строгая проверка относительно вывода во время выполнения тестов. См. *Рискованные тесты* для получения подробной информации.

`--disallow-todo-tests`

Не выполнять тесты, имеющие аннотацию `@todo` в своем докблоке.

`--enforce-time-limit`

Применить ограничение по времени, основываясь на размере теста. См. *Рискованные тесты* для получения более подробной информации.

`--process-isolation`

Запускать каждый тест в отдельном процессе PHP.

`--no-globals-backup`

Не создавать резервную копию и восстанавливать суперглобальный массив `$GLOBALS`. См. *Глобальное состояние* для получения более подробной информации..

`--static-backup`

Резервное копирование и восстановление статических атрибутов пользовательских классов. См. *Глобальное состояние* для получения более подробной информации.

`--colors`

Использовать цвета в выводе. В Windows используйте ANSICON или ConEmu.

Существует три возможных значения этой опции:

- `never`: никогда не отображать цвета в выводе. Это значение по умолчанию, когда не используется опция `--colors`.
- `auto`: отображает цвета в выводе, за исключением, если текущий терминал не поддерживает цвета, либо если вывод не был передан в другую команду или не перенаправлен в файл.
- `always`: всегда отображать цвета в выводе, даже если текущий терминал не поддерживает цвета, или когда вывод передаётся в команду или перенаправляется в файл.

Когда опция `--colors` используется без значения, используется `auto`.

`--columns`

Определяет количество столбцов для вывода прогресса выполнения тестов. Если задано значение `max`, количество столбцов будет максимальным для текущего терминала.

`--stderr`

Необязательно печатать в поток `STDERR` вместо `STDOUT`.

`--stop-on-error`

Прекратить выполнение при первой ошибке.

`--stop-on-failure`

Прекратить выполнение при первой ошибке или неудачном выполнении.

`--stop-on-risky`

Прекратить выполнение при первом рискованном тесте.

`--stop-on-skipped`

Прекратить выполнение при первом пропущенном тесте.

`--stop-on-incomplete`

Прекратить выполнение при первом незавершённом тесте.

`--verbose`

Выводить более подробную информацию, например, имена незавершённых или пропущенных тестов.

`--debug`

Выводить отладочную информацию, такую как название теста при его запуске.

`--loader`

Указывает используемую реализацию загрузчика `PHPUnit\Runner\TestSuiteLoader`

Стандартный загрузчик тестового набора будет искать исходный файл теста в текущей рабочей директории и в каждой директории, указанной в конфигурационной PHP-директиве `include_path`. Имя класса, такое как `Project_Package_Class`, сопоставляется с исходным файлом `Project/Package/Class.php`.

`--repeat`

Повторять выполнение тестов указанное количество раз.

`--testdox`

Сообщает о ходе тестирования в формате TestDox (см. *TestDox*).

`--printer`

Указывает используемую реализацию форматирования вывода. Этот класс должен наследоваться от `PHPUnit\Util\Printer` и реализовывать интерфейс `PHPUnit\Framework\TestListener`.

`--bootstrap`

«Загрузочный» («bootstrap») файл PHP, который будет запускаться перед выполнением тестов.

`--configuration`, `-c`

Прочитать конфигурацию из XML-файла. См. *Конфигурационный XML-файл* для получения более подробной информации.

Если файл `phpunit.xml` или `phpunit.xml.dist` (в таком порядке) существует в текущей рабочей директории, а опция `--configuration` *не* используется, конфигурация будет автоматически прочитана из этого файла.

Если директория указана и файл `phpunit.xml` или `phpunit.xml.dist` (в таком порядке) существует в этой директории, конфигурация будет автоматически загружена из этого файла.

`--no-configuration`

Игнорировать `phpunit.xml` и `phpunit.xml.dist` из текущей рабочей директории.

`--include-path`

Добавить в PHP-опцию `include_path` указанные пути.

`-d`

Устанавливает значение заданной опции конфигурации PHP.

Примечание

Обратите внимание, что с версии 4.8 параметры могут быть указаны после аргументов.

3.2 TestDox

Функциональность TestDox PHPUnit просматривает тестовый класс и все названия его тестовых методов, и преобразует их из имён PHP в стиле написания CamelCase (или snake_case) в предложения: `testBalanceIsInitiallyZero()` (или `test_balance_is_initially_zero()`) становится «Balance is initially zero». Если есть несколько тестовых методов, названия которых отличаются только одной или более цифрой на конце, например `testBalanceCannotBecomeNegative()` и `testBalanceCannotBecomeNegative2()`, предложение «Balance cannot become negative» появится только один раз, при условии, что все эти тесты прошли успешно.

Давайте посмотрим aglie-документацию, сгенерированную для класса `BankAccount`:

```
$ phpunit --testdox BankAccountTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

BankAccount
  Balance is initially zero
  Balance cannot become negative
```

В качестве альтернативы, aglie-документация может быть сгенерирована в HTML или текстовом формате и записана в файл, используя аргументы `--testdox-html` и `--testdox-text`.

Документация Agile может использоваться для документирования предположений, которые вы делаете относительно внешних пакетов, используемых в проекте. Когда вы используете внешний пакет, вы подвержены рискам, что пакет не будет работать так, как ожидалось, то есть он изменит своё поведение, а будущие версии пакета изменятся завуалированным способом, тем самым ломая ваш код, даже не подозревая об этом. Вы можете снизить эти риски, путём написания каждый раз теста, когда вы делаете предположение. Если тест проходит, значит ваше предположение верно. Если вы будете документировать все свои предположения с помощью тестов, новые версии внешнего пакета не будут вызывать беспокойства: если тесты проходят, то система должна продолжать работать.

Глава 4

Фикстуры

Одной из наиболее трудозатратных частей при написании тестов является написание кода для настройки тестового окружения в известное состояние, а затем возврат его в исходное состояние, когда тест будет завершён. Это известное состояние называется *фикстурой* теста.

В разделе *Тестирование операций с массивами с использованием PHPUnit* фикстурой был простой массив, который хранится в переменной `$stack`. Однако, в большинстве случаев, фикстура будет более сложной, чем простой массив, и количество кода, необходимое для её настройки, будет соответственно расти. Фактическое содержание теста потеряется в шуме настройки фикстуры. Эта проблема становится хуже, когда вы пишете несколько тестов с похожими фикстурами. Без помощи от фреймворка тестирования, нам пришлось бы дублировать код, который устанавливает фикстуру, для каждого теста, который мы пишем.

PHPUnit поддерживает общий код установки. Перед выполнением тестового метода будет вызван шаблонный метод `setUp(): void`. `setUp(): void` — это место, где вы создаёте тестируемые объекты. После того, как тестовый метод выполнится, вне зависимости успешно или нет, вызывается другой шаблонный метод с названием `tearDown(): void`. `tearDown(): void` - это место, где вы очищаете протестированные объекты.

В разделе *Использование аннотации @depends для описания зависимостей* мы использовали отношения продюсер-потребитель между тестами для совместного использования фикстур. Это не всегда желательно или даже возможно. Пример 4.1 показывает, как мы можем написать тесты для `StackTest` таким образом, чтобы повторно использовалась не сама фикстура , а код, который её создаёт. Сначала мы объявляем переменную экземпляра, `$stack`, которую мы будем использовать вместо обычной переменной в методе. Затем мы помещаем создание массива (`array`) фикстуры в метод `setUp(): void`. Наконец, мы удаляем избыточный код из тестовых методов и используем недавно созданную переменную экземпляра `$this->stack` вместо локальной переменной метода `$stack` в утверждениях `assertSame()`.

Пример 4.1: Использование setUp(): void для создания фикстуры

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

class StackTest extends TestCase
{
```

```
    protected $stack;

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

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

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

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

Шаблонные методы `setUp(): void` и `tearDown(): void` вызываются по одному разу при каждом выполнении тестового метода (и для нового экземпляра) тестового класса.

Кроме того, вызываются шаблонные методы `setUpBeforeClass(): void` и `tearDownAfterClass(): void` перед тем, как первый тест в тестовом классе будет выполнен, и после запуска последнего теста тестового класса, соответственно.

Приведённый ниже пример показывает все шаблонные методы, доступные в тестовом классе.

Пример 4.2: Пример, показывающий все доступные шаблонные методы

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

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

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

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

    public function testOne(): void
```

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

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

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

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

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

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

```
$ phpunit TemplateMethodsTest
PHPUnit latest.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.
```

4.1 Больше setUp(): void чем tearDown(): void

Методы `setUp(): void` и `tearDown(): void` довольно симметричны в теории, но не на практике. На практике вам нужно реализовывать `tearDown(): void`, если вы в `setUp(): void` создали внешние ресурсы, такие как файлы или сокеты. Если ваш метод `setUp(): void` просто создаёт обычные PHP-объекты, вы можете вообще игнорировать `tearDown(): void`. Однако, если вы создаёте много объектов в своём `setUp(): void`, вам, возможно, потребуется использовать `unset()` для удаления переменных, указывающих на эти объекты в своём методе `tearDown(): void`, чтобы они могли быть очищены сборщиком мусора. Сборщик мусора объектов тестового класса непредсказуем.

4.2 Разновидности

Что произойдёт, если у вас есть два теста с немного различающимися настройками? Есть два варианта:

- Если код `setUp(): void` отличается совсем немного, то необходимо перенести код, отличающийся от `setUp(): void`, в тестовый метод.
- Если у вас действительно разный `setUp(): void`, вам нужен другой тестовый класс. Задайте соответствующее название классу после внесения изменений.

4.3 Совместное использование фикстур

Есть несколько веских причин для совместного использования фикстур между тестами, но в большинстве случаев эта необходимость связана с неразрешённой проблемой проектирования.

Хорошим примером фикстуры для совместного использования между тестами может быть соединение с базой данных: вы подключаетесь к базе данных только один раз и затем повторно используете это соединение к базе данных вместо создания нового подключения для каждого теста. Это позволяет сделать ваши тесты быстрее.

Пример 4.3 использует шаблонные методы `setUpBeforeClass(): void` и `tearDownAfterClass(): void` для подключения к базе данных до выполнения первого теста в тестовом классе и закрытие соединения с базой данных после запуска последнего теста, соответственно.

Пример 4.3: Совместное использование фикстур тестами в тестовом наборе

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

class DatabaseTest extends TestCase
{
    protected static $dbh;

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

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

Следует вновь отметить, что совместное использование фикстур между тестами снижает ценность тестов. Основная проблема проектирования заключается в том, что объекты сильно связаны между собой. Вы достигнете лучших результатов, если решите эту основную проблему в проектировании, а затем напишете тесты, используя заглушки (см. *Тестовые двойники*), вместо создания зависимостей между тестами во время выполнения и игнорируя возможность улучшения архитектуры.

4.4 Глобальное состояние

Трудно тестировать код, который использует синглтоны. То же самое относится и к коду, использующему глобальные переменные. Обычно код, который вы хотите протестировать, сильно связан с глобальной переменной, и вы не можете управлять её созданием. Ещё одна проблема заключается в том, что одно изменение в тесте, использующем глобальную переменную, может сломать другой тест.

В PHP глобальные переменные работают следующим образом:

- Глобальная переменная `$foo = 'bar';` сохраняется как `$GLOBALS['foo'] = 'bar';`.
- Переменная `$GLOBALS` — это так называемая *суперглобальная* переменная.
- Суперглобальные переменные — это встроенные переменные, доступные во всех областях видимости.
- В области видимости функции или метода вы можете получить доступ к `$foo` либо напрямую через `$GLOBALS['foo']` или используя `global $foo;` для создания локальной переменной в текущей области видимости, ссылающейся на глобальную переменную.

Помимо глобальных переменных, статические атрибуты классов также являются частью глобального состояния.

До версии 6, PHPUnit по умолчанию запускал тесты таким образом, что изменения в глобальных и суперглобальных переменных (`$GLOBALS`, `$_ENV`, `$_POST`, `$_GET`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_REQUEST`) не влияли на другие тесты.

Начиная с версии 6, PHPUnit больше не делает операции резервного копирования и восстановления глобальных и суперглобальных переменных по умолчанию. Это можно включить, используя опцию `--globals-backup` или настройку `backupGlobals="true"` в конфигурационном XML-файле.

Используя опцию `--static-backup` или настройку `backupStaticAttributes="true"` в конфигурационном XML-файле, данная изоляция выше может быть расширена до статических атрибутов классов.

Примечание

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

Объекты некоторых классов (например, `PDO`) не могут быть сериализованы, и операция резервного копирования будет прервана, когда подобный объект будет сохраняться, например, в массив `$GLOBALS`.

Аннотация `@backupGlobals`, которая обсуждается в *@backupGlobals*, может использоваться для управления операциями резервного копирования и восстановления глобальных переменных. Кроме этого,

вы можете предоставить чёрный список глобальных переменных, которые должны быть исключены при выполнении операций резервного копирования и восстановления, как показано ниже:

```
class MyTest extends TestCase
{
    protected $backupGlobalsBlacklist = ['globalVariable'];

    // ...
}
```

Примечание

Установка свойства `$backupGlobalsBlacklist` внутри, например, метода `setUp(): void`, не даст никакого эффекта.

Аннотацию `@backupStaticAttributes`, обсуждаемую в *@backupStaticAttributes*, можно использовать для резервного копирования всех статических значений свойств во всех объявленных классах перед каждым тестом с последующим их восстановлением.

Она обрабатывает все классы, объявленные в момент запуска теста, а не только сам тестовый класс. Она применяется только к статическим свойствам класса, а не к статическим переменным внутри функций.

Примечание

Операция `@backupStaticAttributes` выполняется перед каждым тестовым методом, но только если она включена. Если статическое значение было изменено ранее выполненным тестом с отключенным `@backupStaticAttributes`, тогда это значение будет скопировано и восстановлено, но не к первоначальному значению по умолчанию. PHP не записывает первоначально объявленное значение по умолчанию любой статической переменной.

То же самое относительно и к статическим свойствам классов, которые недавно были загружены или объявлены внутри теста. Они не могут быть сброшены к первоначально объявленному значению по умолчанию после теста, так как это значение неизвестно. Независимо установленного значения, произойдёт утечка памяти в последующие тесты.

Для модульных тестов рекомендуется явно сбросить значения статических свойств в методе теста `setUp(): void` (и в идеале также в методе `tearDown(): void`, чтобы не повлиять на последующие выполняемые тесты).

Вы можете предоставить чёрный список статических атрибутов, которые должны быть исключены из операций резервного копирования и восстановления:

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

    // ...
}
```

Примечание

Установка свойства `$backupStaticAttributesBlacklist` внутри, например, метода `setUp(): void`, не даст никакого эффекта.

Глава 5

Организация тестов

Одна из целей PHPUnit заключается в том, что тесты должны быть составными: мы хотим запускать любое количество или комбинацию тестов вместе, например, все тесты для всего проекта, либо тесты всех классов компонента, который является частью проекта, либо просто тесты для одного класса.

PHPUnit поддерживает различные способы организации тестов и составления их в набор тестов. В этой главе показаны наиболее часто используемые подходы.

5.1 Составление набора тестов с помощью файловой системы

Возможно, самый простой способ составить набор тестов — это держать все исходные файлы тестов в тестовом каталоге. PHPUnit может автоматически обнаруживать и запускать тесты путём рекурсивного обхода тестового каталога.

Давайте посмотрим на набор тестов библиотеки sebastianbergmann/money. Просматривая структуру каталогов этого проекта, мы видим, что классы тестов в каталоге `tests` отражают структуру пакета и классов тестируемой системы в каталоге `src`:

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

Для запуска всех тестов библиотеки нам просто нужно указать исполнителю тестов командной строки PHPUnit каталог с тестами:

```
$ phpunit --bootstrap src/autoload.php tests
PHPUnit latest.0 by Sebastian Bergmann and contributors.

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

Time: 636 ms, Memory: 3.50Mb
```

```
OK (33 tests, 52 assertions)
```

Примечание

Если вы укажите исполнителю тестов командной строки PHPUnit каталог, он будет искать файлы с маской `*Test.php`

Для запуска только тестов, объявленных в классе `CurrencyTest`, находящегося в файле `tests/CurrencyTest.php`, мы можем использовать следующую команду:

```
$ phpunit --bootstrap src/autoload.php tests/CurrencyTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

........

Time: 280 ms, Memory: 2.75Mb

OK (8 tests, 8 assertions)
```

Для более точного контроля, какие тесты запускать, мы можем использовать опцию `--filter`:

```
$ phpunit --bootstrap src/autoload.php --filter testObjectCanBeConstructedForValidConstructorArgument␣
↪tests
PHPUnit latest.0 by Sebastian Bergmann and contributors.

..

Time: 167 ms, Memory: 3.00Mb

OK (2 test, 2 assertions)
```

Примечание

Недостатком этого подхода является то, что мы не можем контролировать порядок выполнения тестов. Это может привести к проблемам с зависимостями теста см. *Зависимости тестов*. В следующем разделе вы увидите, как можно явно задать порядок выполнения тестов, используя конфигурационный XML-файл.

5.2 Составление набора тестов с помощью конфигурации XML

XML-файл конфигурации PHPUnit (*Конфигурационный XML-файл*) также может использоваться для составления набора тестов. Пример 5.1 показывает файл `phpunit.xml` с минимальной настройкой, который добавит все классы `*Test`, находящиеся в файлах `*Test.php`, после рекурсивного обхода каталога `tests`.

Пример 5.1: Составление набора тестов, используя конфигурацию XML

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

```
  </testsuites>
</phpunit>
```

Если `phpunit.xml` или `phpunit.xml.dist` (в этом порядке) существует в текущем рабочем каталоге, а опция `--configuration` *не* используется, то конфигурация будет автоматически считана из этого файла.

Порядок выполнения тестов можно сделать явным:

Пример 5.2: Составление набора тестов, используя конфигурацию XML

```
<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>
```

Глава 6

Рискованные тесты

При выполнении тестов PHPUnit может проводить дополнительные проверки, описанные ниже.

6.1 Бесполезные тесты

PHPUnit по умолчанию строг по отношению к тестам, которые ничего не тестируют. Эта проверка может быть отключена с помощью опции командной строки `--dont-report-useless-tests` или через установку `beStrictAboutTestsThatDoNotTestAnything="false"` в конфигурационном XML-файле PHPUnit.

Тест, в котором нет утверждений, будет отмечен как рискованный, если эта проверка включена. Ожидания на поддельных объектах или аннотаций, таких как @expectedException, считаются за утверждение.

6.2 Непреднамеренно покрытый код

PHPUnit может быть строгим по отношению к непреднамеренно покрытому коду. Эта проверка может быть включена с помощью опции командной строки `--strict-coverage` или через установку `beStrictAboutCoversAnnotation="true"` в конфигурационном XML-файле PHPUnit.

Тест с аннотацией @covers, проверяющий код, который не указан при помощи @covers или @uses, будет отмечен как рискованный, если эта проверка включена.

6.3 Вывод во время выполнения теста

PHPUnit может быть строгим по отношению к выводу во время выполнения тестов. Эту проверку можно включить с помощью опции командной строки `--disallow-test-output` или через установку `beStrictAboutOutputDuringTests="true"` в конфигурационном XML-файле PHPUnit.

Тест, который производит вывод, например, через вызов функции `print` либо в тестовом коде, либо в тестируемом, будет отмечен как рискованный, если эта проверка включена.

6.4 Тайм-аут выполнения теста

Для теста может быть применено ограничение времени выполнения, если установлен пакет `PHP_Invoker` и доступно расширение `pcntl`. Обеспечение ограничения времени выполнения может включено с помощью опции командной строки `--enforce-time-limit``или через установку ``enforceTimeLimit="true"` в конфигурационном XML-файле PHPUnit.

Тест с аннотацией `@large` завершится неудачно, если время его выполнения превысит 60 секунд. Этот тайм-аут настраивается через атрибут `timeoutForLargeTests` в конфигурационном XML-файле.

Тест с аннотацией `@medium` завершится неудачно, если время его выполнения займёт больше 10 секунд. Этот тайм-аут настраивается через атрибут `timeoutForMediumTests` в конфигурационном XML-файле.

Тест с аннотацией `@small` завершится неудачно, если его выполнение займёт более 1 секунды. Этот тайм-аут настраивается через атрибут `timeoutForSmallTests` в конфигурационном XML-файле.

Примечание

Тесты должны явно иметь аннотацию либо `@small`, `@medium` или `@large`, чтобы сработало ограничение выполнения теста по времени.

6.5 Манипуляция глобальным состоянием

PHPUnit может быть строгим по отношению к тестам, которые манипулируют глобальным состоянием. Эта проверка может быть включена с помощью опции командной строки `--strict-global-state` или через настройку `beStrictAboutChangesToGlobalState="true"` в конфигурационном XML-файле PHPUnit.

Глава 7

Неполные и пропущенные тесты

7.1 Неполные тесты

Когда вы работаете над новым тестовым классом, вы можете начать с написания пустых тестовых методов для отслеживания тех тестов, которые нужно написать, например:

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

Проблема с пустыми тестовыми методами состоит в том, что фреймворком PHPUnit они интерпретируется как успешно пройденные. Это ошибочное толкование приводит к тому, что отчёты о покрытии становятся бесполезными — вы не сможете увидеть, действительно ли тест прошёл, либо он просто ещё не реализован. Вызов `$this->fail()` в нереализованном тестовом методе также не поможет, поскольку тогда тест будет интерпретироваться как не пройденный. Это было бы так же неверно, как и считать нереализованный тест как пройденный.

Если мы думаем об успешном тестировании как о зелёном свете, а о непройденном тесте как о красном свете, то нам нужен дополнительный жёлтый свет для обозначения теста как неполного или ещё не реализованного. `PHPUnit\Framework\IncompleteTest` — это интерфейс для обозначения исключения, выбрасываемого тестовым методом как результат того, что данный тестовый метод неполный или в данный момент ещё не реализован. `PHPUnit\Framework\IncompleteTestError` — стандартная реализация этого интерфейса.

Пример 7.1 показывает тестовый класс `SampleTest`, содержащий один тестовый метод `testSomething()`. Вызывая удобный метод `markTestIncomplete()` (который автоматически вызывает исключение `PHPUnit\Framework\IncompleteTestError`) в тестовом методе, мы отмечаем, что данный тест является неполным.

Пример 7.1: Маркировка теста как неполного

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

```

```php
class SampleTest extends TestCase
{
    public function testSomething()
    {
        // Необязательно: протестируйте здесь что-нибудь, если хотите.
        $this->assertTrue(true, 'This should already work.');

        // Остановиться тут и отметить, что тест неполный.
        $this->markTestIncomplete(
          'Этот тест ещё не реализован.'
        );
    }
}
```

Неполный тест обозначается `I` в выводе исполнителя тестов командной строки PHPUnit, как показано в следующем примере:

```
$ phpunit --verbose SampleTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

I

Time: 0 seconds, Memory: 3.95Mb

There was 1 incomplete test:

1) SampleTest::testSomething
Этот тест ещё не реализован.

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

Таблица 7.1 показывает API для маркировки тестов как неполных.

Таблица 7.1: API для неполных тестов

Метод	Описание
`void markTestIncomplete()`	Помечает текущий тест как неполный.
`void markTestIncomplete(string $message)`	Помечает текущий тест как неполный, используя `$message` в качестве пояснительного сообщения.

7.2 Пропущенные тесты

Не все тесты могут выполняться в любом окружении. Рассмотрим, например, уровень абстракции базы данных, содержащий несколько драйверов для различных систем баз данных, которые он поддерживает. Разумеется, тесты для драйвера MySQL могут выполняться только в том случае, если доступен сервер MySQL.

Пример 7.2 демонстрирует тестовый класс `DatabaseTest`, содержащий один тестовый метод `testConnection()`. В шаблонном методе `setUp(): void` тестового класса мы проверяем, доступно ли расширение MySQLi, и используем метод `markTestSkipped()` для пропуска этого теста в противном случае.

Пример 7.2: Пропуск теста

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

class DatabaseTest extends TestCase
{
    protected function setUp(): void
    {
        if (!extension_loaded('mysqli')) {
            $this->markTestSkipped(
              'Расширение MySQLi недоступно.'
            );
        }
    }

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

Пропущенный тест обозначается `S` в выводе исполнителя тестов командной строки PHPUnit, как показано в следующем примере:

```
$ phpunit --verbose DatabaseTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

S

Time: 0 seconds, Memory: 3.95Mb

There was 1 skipped test:

1) DatabaseTest::testConnection
Расширение MySQLi недоступно.

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

Таблица 7.2 показывает API пропущенных тестов.

Таблица 7.2: API для пропущенных тестов

Метод	Описание
`void markTestSkipped()`	Отмечает текущий тест как пропущенный.
`void markTestSkipped(string $message)`	Отмечает текущий тест как пропущенный, используя `$message` в качестве пояснительного сообщения.

7.3 Пропуск тестов с помощью @requires

В дополнение к вышеперечисленным методам можно также использовать аннотацию `@requires`, чтобы предоставить общие предварительные условия для тестового класса.

Таблица 7.3: Возможные примеры использования @requires

Тип	Возможные значения	Примеры	Дополнительный пример
PHP	Любой идентификатор версии PHP с необязательным оператором	@requires PHP 7.1.20	@requires PHP >= 7.2
PHPUnit	Любой идентификатор версии PHPUnit с необязательным оператором	@requires PHPUnit 7.3.1	@requires PHPUnit < 8
OS	Регулярное выражения для PHP_OS	@requires OS Linux	@requires OS WIN32|WINNT
OSFAMILY	Любое семейство ОС	@requires OSFAMILY Solaris	@requires OSFAMILY Windows
function	Любой корректный параметр для function_exists	@requires function imap_open	@requires function ReflectionMethod::setAccessible
extension	Имя расширения вместе с необязательным идентификатором версии и необязательным оператором	@requires extension mysqli	@requires extension redis >= 2.2.0

Типы PHP, PHPUnit и extension поддерживают следующие операторы: <, <=, >, >=, =, ==, !=, <>.

Пример 7.3: Пропуск тестового класса с использованием @requires

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

/**
 * @requires extension mysqli
 */
class DatabaseTest extends TestCase
{
    /**
     * @requires PHP >= 5.3
     */
    public function testConnection()
    {
        // Тест требует расширения mysqli и PHP >= 5.3
    }

    // ... Все остальные тесты требуют расширения mysqli
}
```

Если вы используете синтаксис, который не компилируется с определённой версией PHP, посмотрите на версии, от которых зависят тестовые классы в XML-конфигурации (см *Набор тестов*)

Глава 8

Тестирование базы данных [устарело]

Внимание: Не рекомендуется больше использовать, см. данную ишью.

Многие примеры модульного тестирования начального и среднего уровня на любом языке программирования предполагают, что с помощью простых тестов можно легко протестировать логику приложения. Для приложений, ориентированных на базы данных, это далеко от реальности. При начале использования, например, WordPress, TYPO3 или Symfony с Doctrine или Propel, вы легко столкнётесь с серьёзными проблемами с PHPUnit: просто потому, что база данных тесно связана с этими библиотеками.

Примечание

Убедитесь, что у вас PHP-расширение `pdo` и расширения для баз данных, например `pdo_mysql`, установлены. В противном приведённые ниже примеры не будут работать.

Вероятно, вам знакома такая ситуация из своей повседневной работы и проектов, когда вы хотите применить свои новые или профессиональные навыки работы с PHPUnit, но у вас возникла одна из следующих проблем:

1. Метод, который вы хотите протестировать довольно большую операцию JOIN и затем использует полученные данные для вычисления некоторых важных результатов.
2. В вашей бизнес-логике выполняются целый рад операторов SELECT, INSERT, UPDATE и DELETE.
3. Вам необходимо настроить тестовые данные (возможно, значительное количество) в более двух таблиц для получения подходящих первоначальных данных для тестируемых методов.

Расширение DbUnit значительно упрощает настройку базы данных для целей тестирования и позволяет проверять содержимое базы данных после выполнения ряда операций. Установка расширения DbUnit простая и описана в installation.optional-packages.

8.1 Поддерживаемые поставщики для тестирования баз данных

В настоящее время DbUnit поддерживает MySQL, PostgreSQL, Oracle и SQLite. За счёт интеграции в Zend Framework или Doctrine 2 это расширение имеет доступ к другим системам управления баз данных (СУБД), таким как IBM DB2 или Microsoft SQL Server.

8.2 Трудности при тестировании баз данных

Существует веская причина, почему все примеры модульного тестирования не включают взаимодействие с базой данных: такого рода тесты одновременно сложны в настройке и для поддержки. Во время тестирования с базой данных вам необходимо позаботиться о следующих факторов:

- Схема и таблицы базы данных
- Вставка строк, необходимых для теста, в эти таблицы
- Проверка состояния базы данных после того, как тест был пройден
- Очистка базы данных для каждого нового теста

Поскольку многие API баз данных, такие как PDO, MySQLi или OCI8, громоздкие в использовании и многословные при написании, выполнение этих шагов вручную может стать настоящим кошмаром.

Тестовый код должен быть как можно более коротким и точным по нескольким причинам:

- Вы не хотите изменять значительное количество тестового кода при небольших изменений в коде на продакшене.
- Вы хотите легко читать и понимать тестовый код, даже спустя несколько месяцев после его написания.

Кроме того, вы должны понимать, что база данных по существу является глобальной переменной, вставленной в ваш код. Два теста в вашем тестовом наборе могут работать с одной и той же базой данных, и, возможно, повторно использовать эти данные несколько раз. Неудачи в одном тесте могут легко повлиять на результат последующих тестов, тем самым затрудняя процесс тестирования. Ранее упомянутый этап очистки имеет большое значение для решения проблемы «база данных — глобально введённая переменная».

DbUnit помогает упростить все эти проблемы при тестировании с базой данных элегантным способом.

С чем PHPUnit вам точно не сможет помочь, так это то, что тесты, использующие базу данных, значительно медленнее по сравнению с тестами, которые её не используют. В зависимости от того, насколько велико взаимодействие с базой данных, выполнение ваших тестов может занять значительное количество времени. Однако, если вы храните небольшой объём данных, используемый для каждого теста и пытаетесь протестировать как можно больше кода, который не взаимодействует с базой данных, то на выполнение всех тестов займёт около одной минуты, даже на больших наборов тестов.

Например, набор тестов проекта Doctrine 2 в настоящее время содержит около 1000 тестов, где почти половина из которых использует базу данных и при этом всём выполнение тестов укладывается в 15 секунд, используя базу данных MySQL на стандартом настольном компьютере.

8.3 Четыре этапа теста базы данных

В своей книге «Шаблоны тестирования xUnit» (xUnit Test Patterns) Джерард Месарош (Gerard Meszaros) перечисляет четыре этапа (стадии) модульного тестирования:

1. Настройка фикстуры
2. Выполнение системы тестирования (System Under Test)
3. Проверка результата
4. Очистка (teardown)

   Что такое фикстура?

   Фикстура описывает первоначальное состояние вашего приложения и базы данных в момент выполнения теста.

Тестирование базы данных требует, по крайней мере, установки и очистки, чтобы очистить и записать необходимые данные фикстуры в ваши таблицы. Тем не менее, у расширения базы данных есть веские основания для возврата к четырём этапам при тестировании, использующем базу данных для формирования рабочего процесса, выполняемого для каждого из тестов:

8.3.1 1. Очистка базы данных

Поскольку всегда есть первый тест, который работает с базой данных, вы точно не знаете, есть ли в таблицах уже какие-нибудь данные. PHPUnit выполнит операцию TRUNCATE для всех таблиц, чтобы вернуть их в пустое состояние.

8.3.2 2. Настройка фикстуры

Затем PHPUnit выполнит итерацию по всем указанным строкам фикстуры и вставит их в соответствующие таблицы.

8.3.3 3–5. Запуск теста, проверка результата и очистка

После того, как база данных сбрасывается и загружается с её изначальным состоянием, текущий тест выполняется PHPUnit. Эта часть тестового кода не требует знание о расширении базы данных вообще, вы можете продолжать и тестировать всё, что вам нравится, с помощью вашего кода.

В вашем тесте используйте специальное утверждение `assertDataSetsEqual()` для целей проверки, однако, это совершенно необязательно. Эта возможность будет объяснена в разделе «Утверждения базы данных».

8.4 Конфигурация PHPUnit Database TestCase

Обычно при использовании PHPUnit ваши тесты наследуются от `PHPUnit\Framework\TestCase` следующим образом:

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

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

Если вы хотите протестировать код, который использует базу данных, установка такого теста будет немного посложнее, потому что вам нужно использовать дополнительный трейт `TestCaseTrait`, требующий реализации двух абстрактных методов `getConnection()` и `getDataSet()`:

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

class MyGuestbookTest extends TestCase
{
    use TestCaseTrait;

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

    /**
     * @return PHPUnit\DbUnit\DataSet\IDataSet
     */
    public function getDataSet()
    {
        return $this->createFlatXMLDataSet(dirname(__FILE__).'/_files/guestbook-seed.xml');
    }
}
```

8.4.1 Реализация getConnection()

Для работы функциональности очистки и загрузки фикстур, расширение базы данных PHPUnit требует доступа к соединению с базой данных, которое абстрагируется между поставщиками и библиотекой PDO. Важно отметить, что ваше приложение необязательно должно основываться на PDO для использования расширения базы данных PHPUnit, подключение просто используется для очистки и настройки фикстуры.

В предыдущем примере мы создаём подключение SQLite в памяти и передаём его в метод `createDefaultDBConnection`, который оборачивает экземпляр PDO и второй параметр (имя базы данных) в очень простой уровень абстракции с базой данных типа `PHPUnit\DbUnit\Database\Connection`.

Раздел «Использование API подключения к базе данных» объясняет API этого интерфейса и то, как вы можете наилучшим образом его использовать.

8.4.2 Реализация getDataSet()

Метод `getDataSet()` определяет, каким должно быть первоначальное состояние базы данных перед выполнением каждого теста. Состояние базы данных абстрагируется с помощью двух концепций — DataSet и DataTable, которые представлены интерфейсами `PHPUnit\DbUnit\DataSet\IDataSet` и `PHPUnit\DbUnit\DataSet\IDataTable` соответственно. В следующем разделе будет подробно описано, как эти концепции работают и в чём их преимущества при использовании их в тестировании базы данных.

Для реализации нам нужно только знать, что метод `getDataSet()` вызывается только один раз во время `setUp(): void` для извлечения набора данных фикстуры и вставки его в базу данных. В этом

примере мы используем фабричный метод `createFlatXMLDataSet($filename)`, который представляет собой набор данных на основе XML-представления.

8.4.3 Как насчёт схемы базы данных (Database Schema, DDL)?

PHPUnit предполагает, что схема база данных со всеми её таблицами, триггерами, последовательностями и представлениями создаётся до запуска теста. Это означает, что вы как разработчик должны убедиться, что ваша база данных правильно настроена перед выполнением набора тестов.

Существует несколько способов достижения этого предусловия для тестирования с базой данных.

1. Если вы используете базу данных с постоянным соединением (не SQLite в оперативной памяти), вы можете легко настроить базу данных один раз с помощью таких инструментов, как phpMyAdmin для MySQL, и повторно использовать базу данных при каждом запуске теста.
2. Если вы используете такие библиотеки как Doctrine 2 или Propel, вы можете использовать их API для создания схемы базы данных, который понадобиться всего один раз до запуска тестов. Вы можете использовать возможности первоначальной (bootstrap) загрузки PHPUnit и конфигурации для выполнения этого кода каждый раз при выполнении тестов.

8.4.4 Совет: Используйте собственную абстрактную реализацию PHPUnit Database TestCase

Из предыдущего примера реализации вы легко можете увидеть, что метод `getConnection()` довольно статичен и может повторно использован в различных тестовых классов с использованием базы данных. Кроме того, чтобы повысить производительность тестов и снизить накладные расходы, связанные с базой данных, вы можете немного провести рефакторинг кода для создания общего абстрактного класса для тестов вашего приложения, который по-прежнему всё ещё позволяет указать другую фикстуру с данными для каждого теста:

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

abstract class MyApp_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // инстанцировать только pdo один во время выполнения тестов для очистки/загрузки фикстуры
    static private $pdo = null;

    // инстанцировать только PHPUnit\DbUnit\Database\Connection один раз во время теста
    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;
    }
}
```

Однако это соединение с базой данных жёстко закодировано в соединении PDO. PHPUnit имеет одну удивительную возможность, которая поможет сделать этот тестовый класс ещё более универсальным. Если вы используете XML-конфигурацию, вы можете сделать подключение к базе данных настраиваемым для каждого запуска теста. Сначала давайте создадим файл «phpunit.xml» в тестовом каталоге tests/ приложения со следующим содержимым:

```
<?xml version="1.0" encoding="UTF-8" ?>
<dataset>
    <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>
</dataset>
```

Теперь мы можем изменить тестовый класс, чтобы он выглядел так:

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

abstract class Generic_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // инстанцировать только pdo один во время выполнения тестов для очистки/загрузки фикстуры
    static private $pdo = null;

    // инстанцировать только PHPUnit\DbUnit\Database\Connection один раз во время теста
    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['DB_PASSWD
↪'] );
            }
            $this->conn = $this->createDefaultDBConnection(self::$pdo, $GLOBALS['DB_DBNAME']);
        }

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

Теперь мы можем запустить набор тестов базы данных, используя различные конфигурации из интерфейса командной строки:

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

Возможность легко запускать тесты, использующие базу данных, с различными конфигурациями очень важно, если вы ведёте разработку на компьютере разработчика. Если несколько разработчиков выполняют тесты базы данных, используя одно и то же соединение с базой данных, то вы запросто можете столкнуться с неудачами выполнения тестов из-за состояния гонки (race-conditions).

8.5 Понимание DataSets и DataTables

Ключевой концепцией расширения базы данных PHPUnit являются DataSets и DataTables. Вы должны попытаться понять эту простую концепцию для освоения тестирования с использованием базы данных с помощью PHPUnit. DataSet и DataTable — это уровни абстракции вокруг строк и столбцов баз данных. Простой API скрывает основное содержимое базы данных в структуре объекта, который также может быть реализован другими источниками, отличными от базы данных.

Эта абстракция необходима для сравнения текущего содержимого базы данных с ожидаемым. Ожидаемое содержимое может быть представлено в виде файлов формата XML, YAML, CSV или массива PHP, например. Интерфейсы DataSet и DataTable позволяют сравнивать эти концептуально разные источники путём эмуляции хранилища реляционных баз данных в семантически подобном подходе.

Рабочий процесс для утверждений базы данных в ваших тестах, таким образом, состоит из трёх простых шагов:

- Указать одну или более таблиц в базе данных по имени таблицы (фактический набор данных)
- Указать ожидаемый набор данных в предпочтительном формате (YAML, XML, ..)
- Проверить утверждение, что оба представления набора данных равны друг другу (эквивалентны).

Утверждения это не единственный вариант использования для DataSet и DataTable в расширении базы данных PHPUnit. Как показано в предыдущем разделе, они также описывают первоначальное содержимое базы данных. Вы вынуждены определять набор данных фикстуры в Database TestCase, который затем используется для:

- Удаления всех строк из таблиц, указанных в наборе данных.
- Записи всех строк в таблицы данных в базе данных.

8.5.1 Доступные реализации

Существует три различных типов наборов данных/таблиц данных:

- DataSets и DataTables на основе файлов
- DataSet и DataTable на основе запросов
- Фильтр и объединение DataSets и DataTables

Файловые наборы данных и таблиц обычно используются для первоначальной фикстуры и описывают ожидаемое состояние базы данных.

Flat XML DataSet

Наиболее распространённый набор называется Flat XML. Это очень простой (flat) XML-формат, где тег внутри корневого узла `<dataset>` представляет ровно одну строку в базе данных. Имена тегов соответствуют таблице, куда будут добавляться строки (записи), а атрибуты тега представляют столбцы записи. Пример для приложения простой гостевой книги мог бы выглядеть подобным образом:

```
<?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>
```

Это, очевидно, легко писать. В этом примере `<guestbook>` — имя таблицы, в которую добавляются две строки с четырьмя столбцами «id», «content», «user» и «created» с соответствующими им значениями.

Однако за эту простоту приходиться платить.

Из предыдущего примера неочевидно, как указать пустую таблицу. Вы можете вставить тег без атрибутов с именем пустой таблицы. Тогда такой XML-файл для пустой таблицы гостевой книги будет выглядеть так:

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

Обработка значений NULL в простых наборах данных XML утомительна. Значение NULL отличается от пустого строкового значения почти в любой базе данных (Oracle — исключение), что трудно описать в обычном формате XML. Вы можете представить значение NULL, опуская атрибут из строки (записи). Если наша гостевая книга разрешает анонимные записи, представленные значением NULL в столбце «user», гипотетическое состояние таблицы гостевой книги может быть таким:

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Привет, дружище!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="Мне нравится это!" created="2010-04-26 12:14:20" />
</dataset>
```

В нашем случае вторая запись добавлена анонимна. Однако это приводит к серьёзной проблеме определения столбцов. Во время утверждений о равенстве данных каждый набор данных должен указывать, какие столбцы хранятся в таблице. Если атрибут указан NULL для всех строк таблицы данных, как расширение базы данных определит, что столбец должен быть частью таблицы?

Обычный набор данных XML делает сейчас решающе важное предположение, объявляя, что атрибуты в первой определённой строке таблицы определяют столбцы этой таблицы. В предыдущем примере это означало бы, что «id», «content“, «user» и «created» будет столбцами таблицы гостевой книги. Для второй строки, где пользователь («user») не определён, в базу данных в столбец «user» будет вставлено значение NULL.

Когда первая запись гостевой книги удаляется из набора данных, только «id», «content» и «created» будут столбцами таблицы гостевой книги, поскольку столбец «user» не определён.

Чтобы эффективно использовать набор данных Flat XML, когда значения NULL имеют важное значение, первая строка каждой таблицы не должна содержать значения NULL, и только последующие строки могут пропускать атрибуты. Это может быть неудобно, поскольку порядок строк является значимым фактором для утверждений базы данных.

В свою очередь, если вы укажете только подмножество столбцов таблицы в наборе данных Flat XML, все пропущенные значения будут установлены в значения по умолчанию. Это приведёт к ошибкам, только если один из пропущенных столбцов определён как «NOT NULL DEFAULT NULL».

В заключение я могу только посоветовать использовать наборы данных Flat XML, только если вам не нужны значения NULL.

Вы можете создать экземпляр обычного набора данных XML внутри Database TestCase, вызвав метод `createFlatXmlDataSet($filename)`:

```
<?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

Есть ещё один структурированный набор данных XML, который немного более многословный при записи, но не имеет проблем с NULL-значениями из набора данных Flat XML. Внутри корневого узла `<dataset>` вы можете указать теги `<table>`, `<column>`, `<row>`, `<value>` и `<null />`. Эквивалентный набор данных для ранее определённой гостевой книги с использованием Flat XML, будет выглядеть так:

```
<?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>Привет, дружище!</value>
            <value>joe</value>
            <value>2010-04-24 17:15:23</value>
        </row>
        <row>
            <value>2</value>
            <value>Мне нравится это!</value>
            <null />
            <value>2010-04-26 12:14:20</value>
        </row>
    </table>
</dataset>
```

Любой определённый тег `<table>` имеет имя и требует определение всех столбцов с их именами. Он может содержать ноль или любое положительное число вложенных элементов `<row>`. Отсутствие элементов `<row>``означает, что таблица пуста. Теги ``<value>` и `<null />` должны быть указаны в порядке, ранее заданных элементов `<column>`. Тег `<null />`, очевидно, означает, что значение равно NULL.

Вы можете создать экземпляр набора данных XML внутри Database TestCase, вызвав метод `createXmlDataSet($filename)`:

```
<?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

Этот новый XML-формат специально предназначен для сервера баз данных MySQL. Его поддержка была добавлена в PHPUnit 3.5. Файлы в этом формате могут быть сгенерированы с помощью утилиты mysqldump. В отличие от наборов данных CSV, которые `mysqldump` также поддерживает, один файл в этом XML-формате может содержать данные для нескольких таблиц. Вы можете создать файл в этом формате, запустив `mysqldump` следующим образом:

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

Этот файл можно использовать в вашем Database TestCase, путём вызова метода `createMySQLXMLDataSet($filename)`:

```
<?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

Кроме того, вы можете использовать набор данных YAML для примера гостевой книги:

```
guestbook:
  -
    id: 1
    content: "Привет, дружище!"
    user: "joe"
    created: 2010-04-24 17:15:23
  -
    id: 2
    content: "Мне нравится это!"
    user:
    created: 2010-04-26 12:14:20
```

Этот формат прост и удобен, а главное он решает проблему с NULL в похожем наборе данных Flat XML. NULL в YAML — это просто имя столбца без указанного значения. Пустая строка указывается таким образом — `column1: ` .

В настоящее время набор данных YAML не имеет фабричного метода в Database TestCase, поэтому вам необходимо создать его самим:

```
<?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

Ещё один файловый набор данных на основе формата CSV. Каждая таблица набора данных представлена одним CSV-файлом. Для нашего примера с гостевой книгой мы определяем файл guestbook-table.csv:

```
id,content,user,created
1,"Привет, дружище!","joe","2010-04-24 17:15:23"
2,"Мне нравится это!","nancy","2010-04-26 12:14:20"
```

Хотя это очень удобно для редактирования через Excel или OpenOffice, вы не можете указать значения NULL в наборе данных CSV. Пустой столбец приведёт к тому, что в столбец в базе данных будет вставлено пустое значение.

Вы можете создать CSV DataSet следующим образом:

```
<?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 DataSet

В расширении базы данных PHPUnit не существует (пока) массива на основе DataSet, но мы может легко реализовать свой собственный. Пример гостевой книги должен выглядеть так:

```php
<?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' => 'Привет, дружище!',
                        'user' => 'joe',
                        'created' => '2010-04-24 17:15:23'
                    ],
                    [
                        'id' => 2,
                        'content' => 'Мне нравится это!',
                        'user' => null,
                        'created' => '2010-04-26 12:14:20'
                    ],
                ],
            ]
        );
    }
}
```

DataSet PHP имеет очевидные преимущества перед всеми другими наборами данных на основе файлов:

- Массивы PHP, очевидно, могут обрабатывать значения `NULL`.
- Вам не нужны дополнительные файлы для утверждений, и вы можете непосредственно использовать их в TestCase.

Чтобы этот набор выглядел как Flat XML, CSV или YAML, ключи первой указанной строки определяют имена столбцов таблицы, в предыдущем случае это были бы «id», «content», «user» и «created».

Реализация массива DataSet проста и понятна:

```php
<?php

use PHPUnit\DbUnit\DataSet\AbstractDataSet;
use PHPUnit\DbUnit\DataSet\DefaultTableMetaData;
use PHPUnit\DbUnit\DataSet\DefaultTable;
use PHPUnit\DbUnit\DataSet\DefaultTableIterator;

class MyApp_DbUnit_ArrayDataSet extends AbstractDataSet
{
    /**
     * @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]);
            }

            $metaData = new DefaultTableMetaData($tableName, $columns);
            $table = new DefaultTable($metaData);

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

    protected function createIterator($reverse = false)
    {
        return new DefaultTableIterator($this->tables, $reverse);
    }

    public function getTable($tableName)
    {
        if (!isset($this->tables[$tableName])) {
            throw new InvalidArgumentException("$tableName не является таблицей в текущей базе␣
↪данных.");
        }

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

Query (SQL) DataSet

Для утверждений базы данных вам нужен не только набор данный на основе файлов, но также набор данных на основе запросов (Query)/SQL, содержащий фактическое содержимое базы данных. Здесь показан Query DataSet:

```
<?php
$ds = new PHPUnit\DbUnit\DataSet\QueryDataSet($this->getConnection());
$ds->addTable('guestbook');
```

Добавление таблицы просто по имени — это неявный способ определения таблицы данных со следующим запросом:

```
<?php
$ds = new PHPUnit\DbUnit\DataSet\QueryDataSet($this->getConnection());
$ds->addTable('guestbook', 'SELECT * FROM guestbook');
```

Вы можете использовать его, указав произвольные запросы для своих таблиц, например, ограничивая количество строк, столбцов или добавление предложение `ORDER BY`:

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

В разделе «Утверждения базы данных» будет приведена подробная информация о том, как использовать Query DataSet.

Database (DB) Dataset

При доступе к тестовому подключению вы можете автоматически создать DataSet, который состоит из всех таблиц с их содержимым в базе данных, указанной в качестве второго параметра, для фабричного метода соединений.

Вы можете либо создать набор данных для полной базы данных, как показано в `testGuestbook()`, либо ограничится набором указанных имён таблиц с помощью белого списка, как показано в методе `testFilteredGuestbook()`.

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

class MySqlGuestbookTest extends TestCase
{
    use TestCaseTrait;

    /**
     * @return PHPUnit\DbUnit\Database\Connection
     */
    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);
        // ...
    }
}
```

Замена DataSet

Я говорил о проблемах с NULL в наборах данных Flat XML и CSV, но есть несколько сложное обходное решение для получения обоих наборов данных, работающих с NULL.

Замена DataSet — декоратор для существующего набора данных, позволяющий заменять значения в любом столбце набора данных другим заменяющим значением. Для получения примера нашей гостевой книги, работающим со значениями NULL, мы указываем файл следующим образом:

```
<?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>
```

Затем мы оборачиваем Flat XML DataSet в Replacement DataSet:

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

class ReplacementTest extends TestCase
{
    use TestCaseTrait;

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

DataSet Filter

Если у вас большой файл фикстуры, вы можете использовать фильтрацию набора данных для создания белого и чёрного списка таблиц и столбцов, которые должны содержаться поднаборе. Это особенно удобно в сочетании с DB DataSet для фильтрации столбцов набора данных.

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

class DataSetFilterTest extends TestCase
{
    use TestCaseTrait;

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

        $filterDataSet = new PHPUnit\DbUnit\DataSet\DataSetFilter($dataSet);
        $filterDataSet->addIncludeTables(['guestbook']);
        $filterDataSet->setIncludeColumnsForTable('guestbook', ['id', 'content']);
        // ..
    }

    public function testExcludeFilteredGuestbook()
    {
        $tableNames = ['guestbook'];
```

```
        $dataSet = $this->getConnection()->createDataSet();

        $filterDataSet = new PHPUnit\DbUnit\DataSet\DataSetFilter($dataSet);
        $filterDataSet->addExcludeTables(['foo', 'bar', 'baz']); // only keep the guestbook table!
        $filterDataSet->setExcludeColumnsForTable('guestbook', ['user', 'created']);
        // ..
    }
}
```

Примечание

Вы не можете одновременно использовать исключение и включение фильтрации столбцов на одной и той же таблице, только на разных. Кроме того, это возможно только для таблиц белого или чёрного списка, а не для обоих.

Составной DataSet

Составной DataSet очень полезен для объединения (агрегирования) нескольких уже существующих наборов данных в один набор данных. Когда несколько наборов данных содержат одну и ту же таблицу, строки добавляются в указанном порядке. Например, если у нас есть два набора данных — *fixture1.xml*:

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

и *fixture2.xml*:

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

Используя составной DataSet, мы можем объединить оба файла фикстуры:

```
<?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\DbUnit\DataSet\CompositeDataSet();
        $compositeDs->addDataSet($ds1);
        $compositeDs->addDataSet($ds2);

        return $compositeDs;
    }
}
```

8.5.2 Остерегайтесь внешних ключей

Во время установки фикстуры расширения базы данных, PHPUnit вставляет строки в базу данных в том порядке, в котором они указаны в вашей фикстуре. Если ваша схема базы данных использует внешние ключи, это означает, что вы должны указывать таблицы в порядке, не вызывающем нарушений ограничений внешних ключей.

8.5.3 Реализация собственного DataSets/DataTables

Для понимания внутренностей DataSets и DataTables, давайте взглянем на интерфейс DataSet. Вы можете пропустить эту часть, если не планируете реализовать собственный DataSet или DataTable.

```
<?php
namespace PHPUnit\DbUnit\DataSet;

interface IDataSet extends IteratorAggregate
{
    public function getTableNames();
    public function getTableMetaData($tableName);
    public function getTable($tableName);
    public function assertEquals(IDataSet $other);

    public function getReverseIterator();
}
```

Общедоступный интерфейс используется внутри утверждения `assertDataSetsEqual()` в Database TestCase для проверки качества набора данных. Из интерфейса `IteratorAggregate` IDataSet наследует метод `getIterator()` для итерации по всем таблицам набора данных. Обратный итератор позволяет PHPUnit очистить строки таблицы, противоположные порядку их создания для удовлетворения ограничений внешнего ключа.

В зависимости от реализации применяются различные подходы для добавления экземпляров таблиц в набор данных. Например, таблицы добавляются внутри структуры во время создания из исходного файла во все файловые наборы данных, таких как `YamlDataSet`, `XmlDataSet` или `FlatXmlDataSet`.

Таблица также представлена следующим интерфейсом:

```
<?php
namespace PHPUnit\DbUnit\DataSet;

interface ITable
{
    public function getTableMetaData();
    public function getRowCount();
    public function getValue($row, $column);
    public function getRow($row);
    public function assertEquals(ITable $other);
}
```

За исключением метода `getTableMetaData()`, который говорит сам за себя. Используемые методы необходимы для различных утверждений расширения базы данных, которые поясняются в следующей главе. Метод `getTableMetaData()` должен возвращать реализацию интерфейса `PHPUnit\DbUnit\DataSet\ITableMetaData`, который описывает структуру таблицы. В нём содержится следующая информация:

- Имя таблицы
- Массив имён столбцов таблицы, упорядоченных по их появлению в результирующем наборе.
- Массив столбцов первичных ключей.

Этот интерфейс также имеет утверждение, которое проверяет, равны ли два экземпляра табличных метаданных (Table Metadata) друг другу, которое используется утверждением равенства набора данных.

8.6 Использование API подключения к базе данных

В интерфейсе Connection есть три интересных метода, которые необходимо вернуть из метода `getConnection()` в Database TestCase:

```
<?php
namespace PHPUnit\DbUnit\Database;

interface Connection
{
    public function createDataSet(array $tableNames = null);
    public function createQueryTable($resultName, $sql);
    public function getRowCount($tableName, $whereClause = null);

    // ...
}
```

1. Метод `createDataSet()` создаёт набор данных базы данных (Database (DB) DataSet), как описано в разделе реализации DataSet.

   ```
   <?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. Метод `createQueryTable()` может использоваться для создания экземпляров QueryTable, передавая им имя результат и SQL-запроса. Это удобный метод, когда дело доходит до утверждений результата/таблицы, как будет показано в следующем разделе «API утверждений базы данных».

   ```
   <?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 * FROM␣
↪guestbook');
    }
}
```

3. Метод `getRowCount()` — это удобный способ получения доступа к количеству строк в таблице, необязательно отфильтрованное дополнительным предложением where. Это можно использовать с простым утверждением равенства:

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

class ConnectionTest extends TestCase
{
    use TestCaseTrait;

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

8.7 API утверждений базы данных

Инструмент тестирования расширения базы данных, безусловно, содержит утверждения, которые вы можете использовать для проверки текущего состояния базы данных, таблиц и подсчёта строк таблиц. В этом разделе подробно описывается эта функциональность:

8.7.1 Утверждение количество строк таблицы

Часто бывает полезно проверить, содержит ли таблица определённое количество строк. Вы можете легко достичь этого без дополнительного кода, используя API Connection. Предположим, мы хотим проверить, что после вставки строк в нашу гостевую книгу мы имеем не только две первоначальные записи, которые были во всех предыдущих примерах, но а также третью, только что добавленную:

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

class GuestbookTest extends TestCase
{
    use TestCaseTrait;

    public function testAddEntry()
    {
        $this->assertSame(2, $this->getConnection()->getRowCount('guestbook'), "Pre-Condition");

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

```
        $this->assertSame(3, $this->getConnection()->getRowCount('guestbook'), "Inserting failed");
    }
}
```

8.7.2 Утверждение состояния таблицы

Предыдущее утверждение полезно, но мы обязательно хотим проверить фактическое содержимое таблицы, чтобы убедиться, что все значения были записаны в соответствующие столбцы. Это может быть достигнуто с помощью утверждения таблицы.

Для этого нам нужно определить экземпляр таблицы запроса (Query Table), который выводит содержимое по имени таблицы и SQL-запроса и сравнивает его с набором данных на основе файлов/массивов:

```
<?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);
    }
}
```

Теперь для этого утверждения мы должны создать обычный XML-файл *expectedBook.xml*:

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Привет, дружище!" user="joe" created="2010-04-24 17:15:23" />
    <guestbook id="2" content="Мне нравится это!" user="nancy" created="2010-04-26 12:14:20" />
    <guestbook id="3" content="Привет, мир!" user="suzy" created="2010-05-01 21:47:08" />
</dataset>
```

Это утверждение будет успешным только в том случае, если оно будет запущено точно в *2010–05–01 21:47:08*. Даты представляют собой особую проблему при тестировании с использованием базы данных, и мы может обойти эту ошибку, опуская столбец «created» в утверждении.

Скорректированный файл Flat XML *expectedBook.xml*, вероятно, теперь должен выглядеть следующим образом для прохождения утверждения:

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Привет, дружище!" user="joe" />
    <guestbook id="2" content="Мне нравится это!" user="nancy" />
```

```
    <guestbook id="3" content="Привет, мир!" user="suzy" />
</dataset>
```

Мы должны исправить вызов таблицы запроса (Query Table):

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

8.7.3 Утверждение результата запроса

Вы также можете утверждать результат сложных запросов с помощью подхода Query Table, просто указав имя результата с запросом и сравнивая его с набором данным:

```
<?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);
    }
}
```

8.7.4 Утверждение состояния нескольких таблиц

Конечно, вы можете утверждать состояние одновременно нескольких таблиц и сравнивать запрос набора результата с файловым набором данных. Для утверждений DataSet существует два разных способа.

1. Вы можете использовать базу данных (Database, DB) DataSet из Connection и сравнить её с набором данных на основе файлов.

   ```
   <?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. Вы можете создать DataSet самостоятельно:

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

class DataSetAssertionsTest extends TestCase
{
    use TestCaseTrait;

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

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

8.8 Часто задаваемые вопросы

8.8.1 Будет ли PHPUnit (повторно) создавать схему базу данных для каждого теста?

Нет, PHPUnit требует, чтобы все объекты базы данных были доступны при запуске набора. Перед запуском набора тестов необходимо создать базу данных, таблицы, последовательности, триггеры и представления.

У Doctrine 2 или eZ Components есть мощные инструменты, которые позволяют вам создать схему базу данных из предопределённых структур данных. Однако они должны подключены к расширению PHPUnit, чтобы разрешить автоматическое восстановление базы данных до запуска всего полного набора тестов.

Поскольку каждый тест полностью очищает базу данных, вам даже не требуется пересоздавать базу данных для каждого запуска теста. Постоянно доступная база данных работает отлично.

8.8.2 Необходимо ли мне обязательно использовать PDO в моём приложении для расширения базы данных?

Нет, PDO требуется только для очистки и установки фикстуры, а также для утверждений. Вы можете использовать любую понравившуюся абстракцию базы данных внутри своего кода.

8.8.3 Что мне делать, когда я получаю ошибку «Too much Connections»?

Если вы не кешируете экземпляр PDO, созданный через метод TestCase `getConnection()`, количество подключений к базе данных увеличивается на один или несколько при каждом тесте базы данных.

По умолчанию конфигурация MySQL позволяет только 100 одновременных подключений, у других поставщиков также имеются свои ограничения на количество максимальных подключений.

Подраздел «Используйте собственную реализацию PHPUnit Abstract Database TestCase» показывает, как можно предотвратить эту ошибку, используя один закешированный экземпляр PDO во всех ваших тестов.

8.8.4 Как обрабатывать NULL в наборах данных Flat XML / CSV?

Не делайте этого. Вместо этого вы должны использовать наборы данных XML или YAML.

Глава 9

Тестовые двойники

Джерард Месарош (Gerard Meszaros) вводит концепцию тестовых двойников в *Meszaros2007* следующим образом:

> *Gerard Meszaros*:
>
> Иногда просто сложно проверить тестируемую систему, поскольку она зависит от других компонентов, которые невозможно использовать в тестовой среде. Это может быть связано из-за недоступности компонентов или отсутствия необходимых тесту возвращаемых значений или из-за нежелательных побочных эффектов при их выполнении. В других случаях стратегия тестирования может потребовать большей видимости или контроля над внутренним поведением SUT.
>
> Когда мы пишем тест, в котором мы не можем (или не хотим) использовать настоящий (реальный) зависимый компонент (depended-on component, DOC), мы можем заменить его тестовым двойником. Тестовый двойник необязательно должен вести себя, как настоящий DOC. От него требуется предоставить такой же API, чтобы тестируемая система не отличала его от настоящего, считала, что он и есть реальный компонент!

Методы `createMock($type)` и `getMockBuilder($type)`, предоставляемые PHPUnit могут использоваться в тесте для автоматической генерации объекта, который может действовать как тестовый двойник для указанного типа (интерфейса или имени класса). Этот объект тестового двойника может использоваться в каждом контексте, где ожидается или требуется объект исходного (оригинального) типа.

Метод `createMock($type)` немедленно возвращает объект тестового двойника для указанного типа (интерфейса или класса). Создание этого тестового двойника осуществляется с использованием настроек по умолчанию. Методы `__construct()` и `__clone()` не выполняются, а аргументы, переданные методу тестового двойника, не будут клонироваться. Если эти значения по умолчанию не нужны, вы можете использовать метод `getMockBuilder($type)` для настройки генерации тестового двойника, используя текучий (fluent) интерфейс.

По умолчанию все методы исходного класса заменяются фиктивной (dummy) реализацией, которая просто возвращает `null` (без вызова исходного метода). Например, используя метод `will($this->returnValue())`, вы можете настроить эти фиктивные реализации для возврата значения при вызове.

Ограничение: окончательные, закрытые и статические методы

Обратите внимание, что методы, объявленные как `final`, `private` и `static` не могут быть подменены (stubbed) или имитированы (mocked). Они игнорируются функциональностью тестовых двойников PHPUnit и сохраняют своё первоначальное поведение, за исключением методов, объявленных как `static`, которые будут заменены вызовом, выбрасывающим исключение `\PHPUnit\Framework\MockObject\BadMethodCallException`.

9.1 Заглушки

Практика замены объекта тестовым двойником, который (необязательно) возвращает сконфигурированные возвращаемые значения, называется *подмена* (*stubbing*). Вы можете использовать *заглушку* (*stub*) «для замены настоящего компонента, от которого зависит тестируемая система, чтобы обеспечить тест контрольной точкой для опосредованного ввода тестируемой системы. Это позволяет тесту переключить тестируемую систему на ветвь кода, не выполняемую в обычной ситуации.»

Пример 9.2 показывает, как вызывать методы заглушки и устанавливать возвращаемые значения. Сначала мы используем метод `createMock()`, предоставляемый классом `PHPUnit\Framework\TestCase` для установки объекта-заглушки, который будет похож на объект `SomeClass` (Пример 9.1). Затем мы используем текучий интерфейс, который предоставляет PHPUnit, чтобы указать поведение для заглушки. По сути, это означает, что вам не нужно создавать несколько временных объектов и связывать их вместе впоследствии. Вместо этого вы вызываете цепочку методов, как показано в примере. Это приводит к более читабельному и «текучему» коду.

Пример 9.1: Класс, который будет подменён (для него будет сделана заглушка)

```
<?php
class SomeClass
{
    public function doSomething()
    {
        // Сделать что-нибудь.
    }
}
```

Пример 9.2: Подмена вызова метода для возврата фиксированного значения

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

class StubTest extends TestCase
{
    public function testStub()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->willReturn('foo');

        // Вызов $stub->doSomething() теперь вернёт 'foo'.
        $this->assertSame('foo', $stub->doSomething());
```

```
    }
}
```

Ограничение: Методы с названием «method»

Пример, показанный выше, работает только тогда, когда в исходном классе нет метода с названием «method».

Если исходный класс объявляет метод, названный «method», тогда для проверки утверждения нужно использовать `$stub->expects($this->any())->method('doSomething')->willReturn('foo');`.

«За кулисами» PHPUnit автоматически генерирует новый PHP-класс, который реализует желаемое поведение при использовании метода `createMock()`.

Пример 9.3 показывает пример использования текучего интерфейса Mock Builder для настройки создания тестового двойника. Конфигурация этого тестового двойника использует те же самые настройки по умолчанию, которые используются при `createMock()`.

Пример 9.3: Используя API Mock Builder можно настроить генерируемый класс тестового двойника

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

class StubTest extends TestCase
{
    public function testStub()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->getMockBuilder(SomeClass::class)
                     ->disableOriginalConstructor()
                     ->disableOriginalClone()
                     ->disableArgumentCloning()
                     ->disallowMockingUnknownTypes()
                     ->getMock();

        // Настроить заглушку.
        $stub->method('doSomething')
             ->willReturn('foo');

        // Вызов $stub->doSomething() теперь вернёт 'foo'.
        $this->assertSame('foo', $stub->doSomething());
    }
}
```

В приведённых примерах мы до сих пор возвращали простые значения, используя `willReturn($value)`. Это короткий синтаксис делает то же, что и `will($this->returnValue($value))`. Мы можем использовать вариации этого более длинного синтаксиса для достижения более сложного поведения заглушки.

Иногда вы хотите вернуть один из аргументов вызванного метода (без изменений) в качестве результата вызова подмены метода. Пример 9.4 показывает, как вы можете сделать этого, используя `returnArgument()` вместо `returnValue()`.

Пример 9.4: Подмена вызова метода для возврата одного из аргументов

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

```
class StubTest extends TestCase
{
    public function testReturnArgumentStub()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->returnArgument(0));

        // $stub->doSomething('foo') вернёт 'foo'
        $this->assertSame('foo', $stub->doSomething('foo'));

        // $stub->doSomething('bar') вернёт 'bar'
        $this->assertSame('bar', $stub->doSomething('bar'));
    }
}
```

При тестировании текучего интерфейса иногда полезно, чтобы подменённый метод возвращал ссылку на самого себя (объект-заглушку). Пример 9.5 показывает, как вы можете использовать `returnSelf()` для достижения этого.

Пример 9.5: Подмена вызова метода для возврата ссылки на объект заглушки

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

class StubTest extends TestCase
{
    public function testReturnSelf()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->returnSelf());

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

Иногда подменённый метод должен возвращать разные значения в зависимости от предопределённого списка аргументов. Вы можете использовать `returnValueMap()` для создания сопоставления, которое привязывает аргументы к соответствующим возвращаемым значениям. См. Пример 9.6.

Пример 9.6: Подмена вызова метода для возврата значения из карты

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

class StubTest extends TestCase
{
    public function testReturnValueMapStub()
    {
        // Создать заглушку для класса SomeClass.
```

```
        $stub = $this->createMock(SomeClass::class);

        // Создать карту аргументов для возврата значений
        $map = [
            ['a', 'b', 'c', 'd'],
            ['e', 'f', 'g', 'h']
        ];

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->returnValueMap($map));

        // $stub->doSomething() возвращает разные значения в зависимости
        // от предоставленного списка.
        $this->assertSame('d', $stub->doSomething('a', 'b', 'c'));
        $this->assertSame('h', $stub->doSomething('e', 'f', 'g'));
    }
}
```

Когда вызов подменённого метода должен вернуть вычисленное значение вместо фиксированного (см. `returnValue()`) или (неизменённый) аргумент (см. `returnArgument()`), вы можете использовать `returnCallback()`, чтобы подменённый метод возвращал результат функции обратного вызова или метода. См. Пример 9.7.

Пример 9.7: Подмена вызова метода для возврата значения из функции обратного вызова

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

class StubTest extends TestCase
{
    public function testReturnCallbackStub()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->returnCallback('str_rot13'));

        // Вызов $stub->doSomething($argument) вернёт str_rot13($argument)
        $this->assertSame('fbzrguvat', $stub->doSomething('something'));
    }
}
```

Более простой альтернативой настройке метода обратного вызова может быть указание списка ожидаемых возвращаемых значений. Вы можете сделать это с помощью метода `onConsecutiveCalls()`. См. Пример 9.8.

Пример 9.8: Подмена вызова метода для возврата списка значений в указанном порядке

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

class StubTest extends TestCase
{
    public function testOnConsecutiveCallsStub()
    {
```

```
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->onConsecutiveCalls(2, 3, 5, 7));

        // Вызов $stub->doSomething() вернёт разное значение каждый раз
        $this->assertSame(2, $stub->doSomething());
        $this->assertSame(3, $stub->doSomething());
        $this->assertSame(5, $stub->doSomething());
    }
}
```

Вместо возврата значения, подменённый метод может вызывать исключение. Пример 9.9 показывает как использовать `throwException()` для этого.

Пример 9.9: Подмена вызова метода для выбрасывания исключения

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

class StubTest extends TestCase
{
    public function testThrowExceptionStub()
    {
        // Создать заглушку для класса SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Настроить заглушку.
        $stub->method('doSomething')
             ->will($this->throwException(new Exception));

        // Вызов $stub->doSomething() выбрасывает исключение класса Exception
        $stub->doSomething();
    }
}
```

Кроме того, вы можете написать заглушку самостоятельно и улучшить архитектуры в процессе. Доступ к широко используемым ресурсам осуществляется через один фасад, поэтому вы можете легко заменить ресурс заглушкой. Например, вместо непосредственных вызовов к базе данных, разбросанных по всему коду, у вас может быть единственный объект `Database`, реализующий интерфейс ``IDatabase`. Затем вы можете создать заглушку для реализации `IDatabase` и использовать её в своих тестах. Вы даже можете создать опцию для запуска тестов с этой заглушкой базы данных или реальной базы данных, чтобы вы могли использовать ваши тесты как во время разработки, так и при тестировании интеграции с реальной базой данных.

Функциональность, которая должна быть подменена, имеет тенденцию группироваться в один и тот же объект. Представляя функциональность одним, когерентным интерфейсом, вы уменьшаете связанность (coupling) с остальной частью системы.

9.2 Подставные объекты

Практика замены объекта тестовым двойником, который проверяет ожидания, например, утверждая, что метод был вызван, называется *подстановкой* или *имитацией* (*mocking*).

Вы можете использовать *подставной объект* «в качестве точки наблюдения для проверки опосредованного вывода тестируемой системы во время её работы. Обычно подставной объект также содержит функциональность тестовой заглушки, так как он должен возвращать значения в ответ на вызовы, но основное внимание при его реализации уделяется проверке опосредованного вывода. Таким образом, подставной объект — это значительно больше, чем просто тестовая заглушка с дополнительными утверждениями: он используется совершенно иначе.» (Джерард Месарош).

Ограничение: Автоматическая проверка ожиданий

Только подставные объекты, сгенерированные в рамках теста, будут автоматически проверяться PHPUnit. Например, подставные объекты, созданные в провайдерах данных или введённые в тест с использованием аннотации `@depends`, не проверяются автоматически PHPUnit.

Вот пример: предположим, что мы хотим проверить, что корректный метод `update()` в нашем примере вызывается на объекте, который наблюдает за другим объектом. Пример 9.10 показывает код для классов `Subject` и `Observer`, которые являются частью тестируемой системы.

Пример 9.10: Классы Subject и Observer, которые являются частью тестируемой системы

```
<?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()
    {
        // Сделать что-нибудь.
        // ...

        // Уведомить наблюдателей, что мы что-то сделали.
        $this->notify('something');
    }

    public function doSomethingBad()
    {
        foreach ($this->observers as $observer) {
            $observer->reportError(42, 'Произошло что-то плохое', $this);
        }
    }
```

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

    // Другие методы.
}

class Observer
{
    public function update($argument)
    {
        // Сделать что-нибудь.
    }

    public function reportError($errorCode, $errorMessage, Subject $subject)
    {
        // Сделать что-нибудь
    }

    // Другие методы.
}
```

Пример 9.11 показывает, как использовать подставной объект для тестирования взаимодействия между объектами `Subject` и `Observer`.

Сначала мы используем метод `getMockBuilder()`, предоставляемый классом `PHPUnit\Framework\TestCase` для установки подставного объекта для `Observer`. Поскольку мы передаём массив в качестве второго (необязательного) параметра для метода `getMock()`, только метод `update()` класса `Observer` заменяется реализацией подставного объекта.

Поскольку мы заинтересованы в проверке того, что метод вызывается и с какими аргументы он вызывался, мы вводим методы `expects()` и `with()`, чтобы указать, как должно выглядеть это взаимодействие.

Пример 9.11: Тестирование того, что метод вызывается один раз и с указанным аргументом

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

class SubjectTest extends TestCase
{
    public function testObserversAreUpdated()
    {
        // Создать подставной объект для Observer,
        // имитируя только метод update().
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['update'])
                         ->getMock();

        // Настроить ожидание для метода update(),
        // который должен вызваться только один раз со строкой 'something'
        // в качестве своего параметра.
        $observer->expects($this->once())
                 ->method('update')
```

```
                ->with($this->equalTo('something'));

        // Создать объект Subject и присоединить
        // подставной объект Observer к нему.
        $subject = new Subject('My subject');
        $subject->attach($observer);

        // Вызвать метод doSomething() на объекте $subject,
        // который, как мы ожидаем, вызовет метод update()
        // подставного объекта Observer со строкой 'something'.
        $subject->doSomething();
    }
}
```

Метод `with()` может принимать любое количество аргументов, соответствующее количеству аргументов подставного объекта. Вы можете указать более сложные ограничения аргументов метода, чем простое сравнение.

Пример 9.12: Тестирование того, что метод вызывается с несколькими аргументами со своими ограничениями

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

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // Создать подставной объект для класса Observer, имитируя
        // метод reportError()
        $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);

        // Метод doSomethingBad() должен сообщить об ошибке наблюдателю
        // через метод reportError()
        $subject->doSomethingBad();
    }
}
```

Метод `withConsecutive()` может принимать любое количество массивов аргументов, в зависимости от вызовов, которые вы хотите протестировать. Каждый массив — это список ограничений, соответствующих аргументам подставного метода, как в `with()`.

Пример 9.13: Тестирование того, что метод вызывается два раза с определёнными аргументами.

```
<?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);
    }
}
```

Ограничение `callback()` может использоваться для более сложной проверки аргументов. Это ограничение принимает функцию обратного вызова PHP в качестве единственного аргумента. Функция обратного вызова PHP получит аргумент, который будет проверяться как единственный аргумент, и должен возвращать `true`, если аргумент проходит проверку или `false` в противном случае.

Пример 9.14: Более сложная проверка аргументов

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

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // Создать подставной объект для Observer, имитируя
        // метод reportError()
        $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);

        // Метод doSomethingBad() должен сообщить об ошибке наблюдателю
```

```
        // через метод reportError()
        $subject->doSomethingBad();
    }
}
```

Пример 9.15: Проверка того, что метод вызывается один раз с идентичным переданным объектом

```
<?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);
    }
}
```

Пример 9.16: Создание подставного объекта с включённым клонированием параметров

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

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

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

        // теперь подставной объект клонирует параметры, поэтому ограничение на идентичность␣
↪(identicalTo)
        // терпит неудачу.
    }
}
```

Таблица *Ограничения* показывает ограничения, которые могут быть применены к аргументам метода, а в Таблица 9.1 показаны сопоставления, доступные для указания количества вызовов.

Таблица 9.1: Сопоставления (Matchers)

Сопоставление	Описание
PHPUnit\Framework\MockObject\Matcher\AnyInvokedCount any()	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, выполняется ноль или более раз.
PHPUnit\Framework\MockObject\Matcher\InvokedCount never()	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, никогда не выполняется.
PHPUnit\Framework\MockObject\Matcher\InvokedAtLeastOnce atLeastOnce()	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, выполняется хотя бы один раз.
PHPUnit\Framework\MockObject\Matcher\InvokedCount once()	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, выполняется ровно один раз.
PHPUnit\Framework\MockObject\Matcher\InvokedCount exactly(int $count)	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, выполняется указанное в $count раз.
PHPUnit\Framework\MockObject\Matcher\InvokedAtIndex at(int $index)	Возвращает соответствующее сопоставление, когда метод, для которого он вычисляется, выполняется при заданном $index.

Примечание

Параметр $index для сопоставления at() относится к индексу, начинающемуся с нуля, во *всех вызовах метода* для заданного подставного объекта. Соблюдайте осторожность при использовании этого сопоставления, поскольку это может привести к хрупким (brittle) тестам, которые слишком тесно связаны с конкретными деталями реализации.

Как уже упоминалось в начале, когда значения по умолчанию, используемые методом createMock() при генерации тестового двойника, не соответствуют ваши потребностям, то вы можете использовать метод getMockBuilder($type) для настройки генерации тестового двойника с использованием текучего интерфейса. Вот список методов, предоставляемых Mock Builder:

- setMethods(array $methods) может вызываться в объекте Mock Builder для указания методов, которые должны быть заменены настраиваемым тестовым двойником. Поведение других методов не изменится. Если вы вызываете setMethods(null), то никакие методы не будут заменены.
- setMethodsExcept(array $methods) может вызываться в объекте Mock Builder для указания методов, которые не будут заменены настраиваемым тестовым двойником при замене всех остальных общедоступных методов. Это работает обратным образом для setMethods().
- setConstructorArgs(array $args) может вызываться для предоставления массива параметров, которые передаются конструктору исходного класса (который по умолчанию не заменяется фиктивной реализацией).
- setMockClassName($name) может использоваться для указания имени класса для генерируемого класса тестового двойника.
- disableOriginalConstructor() может использоваться для отключения вызова конструктора исходного класса.
- disableOriginalClone() может использоваться для отключения вызова конструктора исходного класса при клонировании.
- disableAutoload() может использоваться для отключения __autoload() во время генерации класса тестового двойника.

9.3 Prophecy

Prophecy - «очень самоуверенный, но мощный и гибкий фрейморк для имитации PHP-объектов. Хотя первоначально он был создан для удовлетворения потребностей phpspec2, он достаточно гибкий, чтобы его можно было использовать внутри любого фреймворка тестирования с минимальными усилиями».

PHPUnit имеет встроенную поддержку использования Prophecy для создания тестовых двойников. Пример 9.17 показывает, как один и тот же тест в Пример 9.11, может быть переписан с использованием философии пророчеств (prophecies) и откровений (revelations) фреймворка Prophecy:

Пример 9.17: Тестирование того, что метод вызывается один раз с определённым аргументом

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

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

        // Создать prophecy для класса Observer.
        $observer = $this->prophesize(Observer::class);

        // Настроить ожидание для метода update(),
        // который должен вызваться только один раз со строкой 'something'
        // в качестве своего параметра.
        $observer->update('something')->shouldBeCalled();

        // Раскрыть (reveal) prophecy и привязать подставной объект
        // к Subject.
        $subject->attach($observer->reveal());

        // Вызвать метод doSomething() на объекте $subject,
        // который, как мы ожидаем, вызовет метод update()
        // подставного объекта Observer со строкой 'something'.
        $subject->doSomething();
    }
}
```

Обратитесь к документации по Prophecy для получения дополнительной информации о том, как создавать, настраивать и использовать заглушки, шпионы и подстановки, используя этот альтернативный фреймворк тестовых двойников.

9.4 Имитация трейтов и абстрактных классов

Метод `getMockForTrait()` возвращает подставной объект, который использует указанный трейт. Все абстрактные методы данного трейта будут имитированы. Это позволяет проверить конкретные методы трейта.

Пример 9.18: Тестирование конкретных методов трейта

```
<?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());
    }
}
```

Метод `getMockForAbstractClass()` возвращает подставной объект для абстрактного класса. Все абстрактные методы заданного абстрактного класса имитируются. Это позволяет проверить конкретные методы абстрактного класса.

Пример 9.19: Тестирование конкретных методов абстрактного класса

```
<?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());
    }
}
```

9.5 Создание заглушек и имитация веб-сервисов

Когда ваше приложение взаимодействует с веб-сервисами, вы хотите протестировать его без фактического взаимодействия с веб-сервисом. Для упрощения создания заглушек и имитации веб-сервисов, может использоваться метод `getMockFromWsdl()`, по аналогии с `getMock()` (см. выше). Единственное отличие заключается в том, что `getMockFromWsdl()` возвращает заглушку или подставной объект на основе описания веб-сервиса в формате WSDL, тогда как `getMock()` возвращает заглушку или подставной объект, основанный на классе или интерфейсе PHP.

Пример 9.20 показывает, как `getMockFromWsdl()` может использоваться для заглушки, например, веб-сервиса, описанного в `GoogleSearch.wsdl`.

Пример 9.20: Создание заглушки для веб-сервиса

```
<?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() теперь возвратит результат заглушки (stubbed result),
         * а метод doGoogleSearch() веб-сервиса не будет вызван.
         */
        $this->assertEquals(
          $result,
          $googleSearch->doGoogleSearch(
            '000000000000000000000000000000000',
            'PHPUnit',
            0,
            1,
            false,
            '',
            false,
            '',
            '',
            ''
          )
        );
    }
}
```

9.6 Имитация файловой системы (УСТАРЕЛО)

vfsStream — обёртка потока для виртуальной файловой системы, которая может быть полезной в модульных тестах для имитации реальной файловой системы.

Просто добавьте зависимость `mikey179/vfsstream` в файл `composer.json` вашего проекта, если вы используете Composer для управления зависимостями в своём проекте. Вот самый минимальный файл `composer.json`, который просто определяет зависимости для разработки PHPUnit 4.6 и vfsStream:

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

Пример 9.21 показывает класс, взаимодействующий с файловой системой.

Пример 9.21: Класс, взаимодействующий с файловой системой

```
<?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);
        }
    }
}
```

Без виртуальной файловой системы, такой как vfsStream, мы не можем протестировать метод `setDirectory()` в изоляции от внешнего воздействия (см. Пример 9.22).

Пример 9.22: Тестирование класса, взаимодействующего с файловой системой

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

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

Приведённый выше подход имеет несколько недостатков:

- Как и в случае с любым внешним ресурсом, могут возникать периодические проблемы с файловой системой. Это делает взаимодействие с тестами непредсказуемым.
- В методах `setUp(): void` и `tearDown(): void` мы должны убедиться, что каталог не существует до и после теста.
- Когда выполнение теста завершается до того, как метод `tearDown(): void` будет выполнен, каталог останется в файловой системе.

Пример 9.23 показывает, как vfsStream может использоваться для имитации файловой системы в тесте для класса, который взаимодействует с файловой системой.

Пример 9.23: Имитация файловой системы в тесте для класса, взаимодействующего с файловой системой

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

class ExampleTest extends TestCase
{
    public function setUp(): void
    {
        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'));
    }
}
```

Это имеет ряд преимуществ:

- Тест сам стал более кратким.
- vfsStream даёт разработчику теста полный контроль над тем, как выглядит окружение файловой системы для тестируемого кода.
- Поскольку операции файловой системы больше не выполняются на реальной файловой системе, операции очистки в методе `tearDown(): void` больше не требуются.

Глава 10

Анализ покрытия кода

> *Википедия*:
>
> В информатике покрытие кода — мера, используемая для описания степени, в которой исходной код программы протестирован определённым набором тестов. Программа с высоким покрытием кода была более тщательно протестирована и имеет меньше шансов содержать ошибки программного обеспечения, чем программа с низким покрытием кода тестами.

В этой главе вы узнаете всё о функциональности покрытия кода PHPUnit, которая даёт представление о том, какие части кода выполняются при выполнении тестов. Она использует компонент php-code-coverage, который, в свою очередь, использует функциональность покрытия кода, предоставляемую PHP-расширением Xdebug.

Примечание

Xdebug не распространяется как часть PHPUnit. Если во время тестирования вы получаете уведомление о том, что драйвер покрытия кода отсутствует, это означает, что Xdebug либо не установлен, либо неправильно настроен. Прежде чем вы сможете использовать возможности анализа покрытия кода, вам следует прочитать руководство по установке Xdebug.

php-code-coverage также поддерживает phpdbg в качестве альтернативного источника для данных покрытия кода.

PHPUnit может генерировать отчёт о покрытии кода на основе HTML, а также лог-файлы в представлении XML с информацией о покрытии кода в различных форматах (Clover, Crap4J, PHPUnit). Информация о покрытии кода также может быть представлена в виде текста (и напечатана в STDOUT) и экспортирована как код PHP для дальнейшей обработки.

Обратитесь к *Исполнитель тестов командной строки* для просмотра списка переключателей командной строки, которые управляют функциональностью покрытия кода, а также *Логирование* для получения соответствующих параметров конфигурации.

10.1 Показатели программного обеспечения покрытия кода

Существуют различные показатели (метрики) программного обеспечения (далее просто «показатель») для оценки покрытия кода:

Покрытие строки (Line Coverage)

Показатель *Line Coverage* определяет, была ли выполнена каждая исполняемая строка.

Покрытие функции и метода (Function and Method Coverage)

Показатель *Function and Method Coverage* определяет, была ли вызвана каждая функция или метод. php-code-coverage рассматривает функцию или метод как покрытую тестом, только когда все исполняемые строки покрыты.

Покрытие класса и трейта (Class and Trait Coverage)

Показатель *Class and Trait Coverage* определяет, был ли покрыт каждый метод класса или трейта. php-code-coverage рассматривает класс или трейт как покрытый, только когда все их методы покрыты.

Покрытие кода операции (Opcode Coverage)

Показатель *Opcode Coverage* определяет, выполнялся ли каждый опкод (код операции, opcode) функции или метода во время выполнения набора тестов. Строка кода обычно компилируется в несколько кодов опкодов. *Line Coverage* рассматривает строку как покрытую, как только один из опкодов будет выполнен.

Покрытие ветки (Branch Coverage)

Показатель *Branch Coverage* определяет, было ли логическое выражение каждой управляющей структуры оценено как `true`, так и `false` при выполнении набора тестов.

Покрытие пути (Path Coverage)

Показатель *Path Coverage* определяет, использовался ли каждый из возможных путей выполнения функции или метода во время выполнения набора тестов. Путь выполнения — это уникальная последовательность ветвей от входа функции или метода до его выхода.

Индекс Change Risk Anti-Patterns (CRAP)

Индекс *Change Risk Anti-Patterns (CRAP)* рассчитывается на основе цикломатической сложности и покрытия кода единицы кода. Код, который не слишком сложный и имеет адекватный процент покрытия кода, будет иметь низкий индекс CRAP. Индекс CRAP может быть снижен путём написания тестов и рефакторинга тестов для уменьшения его сложности.

Примечание

Показатели *Opcode Coverage*, *Branch Coverage* и *Path Coverage* ещё не поддерживаются php-code-coverage.

10.2 Белый список файлов

Необходимо настроить *белый список* (*whitelist*) для указания PHPUnit, какие файлы исходного кода следует включить в отчёт о покрытии кода. Это можно сделать либо используя опцию командной строки `--whitelist`, либо через файл конфигурации (см. *Файлы в белом списке для покрытия кода*).

Доступны параметры конфигурации `addUncoveredFilesFromWhitelist` и `processUncoveredFilesFromWhitelist` для настройки использования белого списка:

- `addUncoveredFilesFromWhitelist="false"` означает, что в отчёт о покрытии кода будут включены только файлы из белого списка, содержащие хотя бы одну строку выполненного кода
- `addUncoveredFilesFromWhitelist="true"` (по умолчанию) означает, что все файлы из белого списка будут включены в отчёт о покрытии кода, даже если ни одна строка кода такого файла не была выполнена
- `processUncoveredFilesFromWhitelist="false"` (по умолчанию) означает, что в отчёт о покрытии кода будет включены файлы из белого списка, у которых нет исполненных строк кода (если установлено `addUncoveredFilesFromWhitelist="true"`), но он не будет загружен PHPUnit и поэтому не будет анализироваться для корректной информации о исполненных строк кода
- `processUncoveredFilesFromWhitelist="true"` означает, что файл в белом списке, у которого нет исполненных строк кода, будет загружен PHPUnit, чтобы его можно было анализировать для корректной информации о исполненных строк

Примечание

Обратите внимание, что загрузка файлов исходного кода, выполняемая при установке `processUncoveredFilesFromWhitelist="true"`, может вызвать проблемы, например, когда файл исходного кода содержит код вне области класса или функции.

10.3 Игнорирование блоков кода

Иногда у вас есть блоки кода, которые вы не можете протестировать и поэтому вы можете игнорировать при анализе покрытия кода. PHPUnit позволяет сделать это с использованием аннотаций `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` и `@codeCoverageIgnoreEnd`, как показано в Пример 10.1.

Пример 10.1: Использование аннотаций `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` и `@codeCoverageIgnoreEnd`

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

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

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

```
    }
}

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

exit; // @codeCoverageIgnore
```

Пропущенные строки кода (отмеченные как игнорируемые с помощью аннотаций) считаются выполненными (если они могут быть исполнены) и не будут подсвечиваться.

10.4 Определение покрытых методов

Аннотация `@covers` (см. *Аннотации для указания, какие методы покрываются тестом*) может использоваться в тестовом коде для указания, какие методы тестовый метод хочет протестировать. Если она указана, то в информации о покрытии кода будут будут только эти указанные методы. Пример 10.2 показывает это на примере.

Пример 10.2: Тесты, в которых указывается, какой метод они хотят покрыть

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

class BankAccountTest extends TestCase
{
    protected $ba;

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

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

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

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

            return;
```

```
        }

        $this->fail();
    }

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

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

            return;
        }

        $this->fail();
    }

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

Также можно указать, что тест не должен покрывать *какой-либо* метод, используя аннотацию `@coversNothing` (см. *@coversNothing*). Это может быть полезно при написании интеграционных тестов, чтобы убедиться, что вы только генерируете покрытие кода с помощью модульных тестов.

Пример 10.3: Тест, который указывает, что ни один метод не должен быть покрыт

```
<?php
use PHPUnit\DbUnit\TestCase

class GuestbookIntegrationTest extends 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);
    }
}
```

10.5 Крайние случаи

В этом разделе показаны заслуживающие внимания крайние случаи, которые приводят к путанице информации о покрытии кода.

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

// Потому этот код "находится на одной строке", а в не отдельном блоке инструкций,
// в одной строке всегда будет один статус покрытия
if (false) this_function_call_shows_up_as_covered();

// Из-за того, как покрытие кода работает внутри, эти две строки - особенные.
// Эта строка будет отображаться как не исполняемая
if (false)
    // Эта строка будет отображаться как покрытая, потому что на самом деле
    // покрытие оператора if в строке выше показано здесь!
    will_also_show_up_as_covered();

// Чтобы избежать этого, необходимо использовать фигурные скобки
if (false) {
    this_call_will_never_show_up_as_covered();
}
```

Глава 11

Логирование

PHPUnit может создавать несколько типов лог-файлов.

11.1 Результаты теста (XML)

Лог-файл XML результатов тестирования, созданный PHPUnit, основан на использовании задачи JUnit для Apache Ant. В следующем примере показан лог-файл XML, сгенерированный для тестов в `ArrayTest`:

```
<?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>
```

Следующий лог-файл XML был сгенерирован для двух тестов `testFailure` и `testError` тестового класса `FailureErrorTest` и показывает как обозначаются неудачи и ошибки.

```
<?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>
```

11.2 Покрытие кода (XML)

Формат XML для логирования информации о покрытии, созданный PHPUnit, отчасти основан на том, что используется в Clover.

В следующем примере показан лог-файл XML, сгенерированный для тестов `BankAccountTest`:

```
<?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>
```

11.3 Покрытие кода (TEXT)

Человекочитаемое покрытие кода можно выводить в командую строку или текстовый файл.

Цель этого формата вывода — обеспечить общий обзор покрытия кода тестами при работе с небольшим набором классов. Для больших проектов этот вывод может быть полезен для краткого обзора покрытия проектов или при использовании функциональности с флагом `--filter`. При использовании из командной строки, записывая в `php://stdout`, будет учитываться настройка `--colors`.

Запись в стандартный вывод — это опция по умолчанию при использовании из командной строки. По умолчанию будут отображаться только файлы, имеющие хотя бы одну покрытую строку. Это можно изменить через опцию конфигурации `showUncoveredFiles`. См. *Логирование*. По умолчанию все файлы и их статус покрытия отображаются в подробном отчёте. Это можно изменить с помощью конфигурационной опции `showOnlySummary`.

Глава 12

Расширение PHPUnit

PHPUnit можно расширить различными способами для облегчения процесса написания тестов и настройки обратной связи, получаемой от выполнения тестов. Вот общие отправные точки для расширения PHPUnit.

12.1 Подкласс PHPUnit\Framework\TestCase

Написать пользовательские утверждения и вспомогательные методы в абстрактных подклассах `PHPUnit\Framework\TestCase` и наследуйте ваши классы тестов от этого класса. Это один из самых простых способов расширения PHPUnit.

12.2 Написание пользовательских утверждений

При написании пользовательских утверждений лучше всего следовать принципам реализации собственных утверждений PHPUnit. Как вы можете видеть в Пример 12.1, метод `assertTrue()` — это просто обёртка над методами `isTrue()` и `assertThat()`: `isTrue()` создаёт объект сопоставления, который передаётся `assertThat()` для проведения вычисления.

Пример 12.1: Методы assertTrue() и isTrue() класса PHPUnit\Framework\Assert

```
<?php
namespace PHPUnit\Framework;

use PHPUnit\Framework\TestCase;

abstract class Assert
{
    // ...

    /**
     * Утверждает, что условие истинно
```

```
     *
     * @param  boolean $condition
     * @param  string  $message
     * @throws PHPUnit\Framework\AssertionFailedError
     */
    public static function assertTrue($condition, $message = '')
    {
        self::assertThat($condition, self::isTrue(), $message);
    }

    // ...

    /**
     * Возвращает объект сопоставления PHPUnit\Framework\Constraint\IsTrue.
     *
     * @return PHPUnit\Framework\Constraint\IsTrue
     * @since  Method available since Release 3.3.0
     */
    public static function isTrue()
    {
        return new PHPUnit\Framework\Constraint\IsTrue;
    }

    // ...
}
```

Пример 12.2 показывает, как `PHPUnit\Framework\Constraint\IsTrue` наследует абстрактный базовый класс для объектов сопоставления (или ограничений), `PHPUnit\Framework\Constraint`.

Пример 12.2: Класс PHPUnit\Framework\Constraint\IsTrue

```
<?php
namespace PHPUnit\Framework\Constraint;

use PHPUnit\Framework\Constraint;

class IsTrue extends Constraint
{
    /**
     * Вычисляет ограничение для параметра $other. Возвращает true, если
     * ограничение удовлетворяется, в противном случае - false.
     *
     * @param mixed $other Значение или объект для вычисления
     * @return bool
     */
    public function matches($other)
    {
        return $other === true;
    }

    /**
     * Возвращает ограничения в виде строки
     *
     * @return string
     */
    public function toString()
    {
        return 'это true';
```

```
    }
}
```

Усилия по реализации методов `assertTrue()` и `isTrue()`, а также класса `PHPUnit\Framework\Constraint\IsTrue` дают преимущество, состоящее в том, что `assertThat()` автоматически выполняет вычисление утверждения и задач отчётности, таких как подсчёт статистики. Кроме того, метод `isTrue()` может использоваться как сопоставление при настройке подставных объектов.

12.3 Реализация PHPUnit\Framework\TestListener

Пример 12.3 показывает простую реализацию интерфейса `PHPUnit\Framework\TestListener`.

Пример 12.3: Простой обработчик тестов

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

class SimpleTestListener implements TestListener
{
    public function addError(PHPUnit\Framework\Test $test, \Throwable $e, float $time): void
    {
        printf("Ошибка во время выполнения теста '%s'.\n", $test->getName());
    }

    public function addWarning(PHPUnit\Framework\Test $test, PHPUnit\Framework\Warning $e, float
↪$time): void
    {
        printf("Предупреждение во время выполнения теста '%s'.\n", $test->getName());
    }

    public function addFailure(PHPUnit\Framework\Test $test,␣
↪PHPUnit\Framework\AssertionFailedError $e, float $time): void
    {
        printf("Тест '%s' провалился.\n", $test->getName());
    }

    public function addIncompleteTest(PHPUnit\Framework\Test $test, Exception $e, float $time):␣
↪void
    {
        printf("Тест '%s' является неполным.\n", $test->getName());
    }

    public function addRiskyTest(PHPUnit\Framework\Test $test, Exception $e, float $time): void
    {
        printf("Тест '%s' считается рискованным.\n", $test->getName());
    }

    public function addSkippedTest(PHPUnit\Framework\Test $test, Exception $e, float $time): void
    {
        printf("Тест '%s' был пропущен.\n", $test->getName());
    }

    public function startTest(PHPUnit\Framework\Test $test): void
    {
```

```
        printf("Тест '%s' запустился.\n", $test->getName());
    }

    public function endTest(PHPUnit\Framework\Test $test, float $time): void
    {
        printf("Тест '%s' завершился.\n", $test->getName());
    }

    public function startTestSuite(PHPUnit\Framework\TestSuite $suite): void
    {
        printf("Набор тестов '%s' запустился.\n", $suite->getName());
    }

    public function endTestSuite(PHPUnit\Framework\TestSuite $suite): void
    {
        printf("Набор тестов '%s' завершился.\n", $suite->getName());
    }
}
```

Пример 12.4 показывает, как использовать трейт `PHPUnit\Framework\TestListenerDefaultImplementation`, который позволяет указать только интересующие методы интерфейса для вашего случая, но при этом предоставляет пустые реализации для всех остальных методов.

Пример 12.4: Использование трейта с реализацией по умолчанию для обработчика тестов

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

class ShortTestListener implements TestListener
{
    use TestListenerDefaultImplementation;

    public function endTest(PHPUnit\Framework\Test $test, $time): void
    {
        printf("Тест '%s' завершился.\n", $test->getName());
    }
}
```

В *Обработчики тестов* вы увидите, как настроить PHPUnit для добавления обработчика тестов к выполнению теста.

12.4 Реализация PHPUnit\Framework\Test

Интерфейс `PHPUnit\Framework\Test` — небольшой и простой для реализации. Вы можете написать реализацию `PHPUnit\Framework\Test`, которая проще, чем `PHPUnit\Framework\TestCase`, и которая, например, запускает *тесты, управляемые данными*.

Пример 12.5 показывает класс теста, управляемого данными, который сравнивает значения из CSV-файла, где значения разделены запятой. Каждая строка такого файла выглядит примерно как `foo;bar`, где первое значение — это то, что мы ожидаем, а второе значение — фактическое.

Пример 12.5: Тест, управляемый данными

```
<?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) {
                $result->addFailure($this, $e, $stopTime);
            }

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

            finally {
                $stopTime = PHP_Timer::stop();
            }

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

        return $result;
    }
}

$test = new DataDrivenTest('data_file.csv');
```

```
$result = PHPUnit\TextUI\TestRunner::run($test);
```

```
PHPUnit latest.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.
```

12.5 Расширение TestRunner

PHPUnit latest поддерживает расширения TestRunner, которые привязываются к различным событиям во время выполнения теста. См. *Регистрация расширений TestRunner* для получения дополнительной информации о регистрации расширений в конфигурационном XML-файле PHPUnit.

Каждое доступное событие, к которому может подключаться расширение, представлено интерфейсом, которое расширению необходимо реализовать. *Интерфейсы доступных событий* перечисляет доступные события в PHPUnit latest.

12.5.1 Интерфейсы доступных событий

- `AfterIncompleteTestHook`
- `AfterLastTestHook`
- `AfterRiskyTestHook`
- `AfterSkippedTestHook`
- `AfterSuccessfulTestHook`
- `AfterTestErrorHook`
- `AfterTestFailureHook`
- `AfterTestWarningHook`
- `BeforeFirstTestHook`
- `BeforeTestHook`

Пример 12.6 показывает пример расширения, реализующего `BeforeFirstTestHook` и `AfterLastTestHook`:

Пример 12.6: Пример расширения TestRunner

```php
<?php

namespace Vendor;

use PHPUnit\Runner\AfterLastTestHook;
use PHPUnit\Runner\BeforeFirstTestHook;

final class MyExtension implements BeforeFirstTestHook, AfterLastTestHook
{
    public function executeAfterLastTest(): void
    {
        // вызывается после последнего выполненного теста
    }

    public function executeBeforeFirstTest(): void
    {
        // вызывается до выполнения первого теста
    }
}
```

Глава 13

Утверждения

В этом приложении перечислены различные доступные методы утверждения.

13.1 Статическое в сравнении с нестатическим использованием методов утверждения

Утверждения PHPUnit реализованы в `PHPUnit\Framework\Assert`. `PHPUnit\Framework\TestCase` наследуется от `PHPUnit\Framework\Assert`.

Методы утверждения объявляются статическими и могут быть вызваны из любого контекста, например `PHPUnit\Framework\Assert::assertTrue()`, или используя, например, `$this->assertTrue()` или `self::assertTrue()`, в классе, наследующий `PHPUnit\Framework\TestCase`.

Фактически, вы даже можете использовать глобальные функции-обёртки, такие как `assertTrue()` в любом контексте (включая классы, наследующие `PHPUnit\Framework\TestCase`), когда вы (вручную) включаете файл исходного кода `src/Framework/Assert/Functions.php`, который поставляется с PHPUnit.

Часто задаваемый вопрос, особенно от разработчиков, впервые работающие с PHPUnit, является ли использование `$this->assertTrue()` или `self::assertTrue()`, например, «правильным путём» для вызова утверждения. Короткий ответ: нет правильного пути. И нет также неправильного пути. Это вопрос личных предпочтений.

Для большинства людей «кажется правильным» использовать `$this->assertTrue()` потому что тестовый метод вызывается в контексте тестового объекта. Тот факт, что методы утверждения объявлены статическими позволяет (повторно) использовать их вне области видимости тестового объекта. Наконец, глобальные функции-обёртки позволяют разработчикам вводить меньше символов (`assertTrue()` вместо `$this->assertTrue()` или `self::assertTrue()`).

13.2 assertArrayHasKey()

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

Сообщает об ошибке, определённой в `$message`, если `$array` не имеет `$key`.

`assertArrayNotHasKey()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.1: Использование assertArrayHasKey()

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

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

```
$ phpunit ArrayHasKeyTest
PHPUnit latest.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.
```

13.3 assertClassHasAttribute()

`assertClassHasAttribute(string $attributeName, string $className[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$className::attributeName` не существует.

`assertClassNotHasAttribute()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.2: Использование assertClassHasAttribute()

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

class ClassHasAttributeTest extends TestCase
{
    public function testFailure()
```

```
    {
        $this->assertClassHasAttribute('foo', stdClass::class);
    }
}
```

```
$ phpunit ClassHasAttributeTest
PHPUnit latest.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.
```

13.4 assertArraySubset()

`assertArraySubset(array $subset, array $array[, bool $strict = false, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$array` не содержит `$subset`.

`$strict` — флаг, используемый для сравнения идентичности объектов внутри массивов.

Пример 13.3: Использование assertArraySubset()

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

class ArraySubsetTest extends TestCase
{
    public function testFailure()
    {
        $this->assertArraySubset(['config' => ['key-a', 'key-b']], ['config' => ['key-a']]);
    }
}
```

```
$ phpunit ArraySubsetTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) EpilogEpilogTest::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.
```

13.5 assertClassHasStaticAttribute()

`assertClassHasStaticAttribute(string $attributeName, string $className[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$className::attributeName` не существует.

`assertClassNotHasStaticAttribute()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.4: Использование assertClassHasStaticAttribute()

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

class ClassHasStaticAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertClassHasStaticAttribute('foo', stdClass::class);
    }
}
```

```
$ phpunit ClassHasStaticAttributeTest
PHPUnit latest.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.
```

13.6 assertContains()

`assertContains(mixed $needle, iterable $haystack[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$needle` не является элементом в `$haystack`.

`assertNotContains()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeContains()` и `assertAttributeNotContains()` — удобные обёртки, которые используют общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве параметра haystack.

Пример 13.5: Использование assertContains()

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

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains(4, [1, 2, 3]);
    }
}
```

```
$ phpunit ContainsTest
PHPUnit latest.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])`

Сообщает об ошибке, определённой в `$message`, если `$needle` не является подстрокой `$haystack`.

Если `$ignoreCase` равен `true`, тест будет нечувствителен к регистру.

Пример 13.6: Использование assertContains()

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

class ContainsTest extends TestCase
{
    public function testFailure()
```

```
    {
        $this->assertContains('baz', 'foobar');
    }
}
```

```
$ phpunit ContainsTest
PHPUnit latest.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.
```

Пример 13.7: Использование assertContains() с $ignoreCase

```
<?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 latest.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.
```

13.7 assertContainsOnly()

`assertContainsOnly(string $type, iterable $haystack[, boolean $isNativeType = null, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$haystack` содержит не только переменные типа `$type`.

`$isNativeType` — флаг, используемый для указания, является ли `$type` встроенным в PHP или нет.

`assertNotContainsOnly()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeContainsOnly()` и `assertAttributeNotContainsOnly()` — удобные обёртки, которые используют общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве параметра haystack.

Пример 13.8: Использование assertContainsOnly()

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

class ContainsOnlyTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnly('string', ['1', '2', 3]);
    }
}
```

```
$ phpunit ContainsOnlyTest
PHPUnit latest.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.
```

13.8 assertContainsOnlyInstancesOf()

`assertContainsOnlyInstancesOf(string $classname, Traversable|array $haystack[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$haystack` содержит не только экземпляры класса `$classname`.

Пример 13.9: Использование assertContainsOnlyInstancesOf()

```
<?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 latest.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.
```

13.9 assertCount()

`assertCount($expectedCount, $haystack[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если количество элементов в `$haystack` не равно `$expectedCount`.

`assertNotCount()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.10: Использование assertCount()

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

```
class CountTest extends TestCase
{
    public function testFailure()
    {
        $this->assertCount(0, ['foo']);
    }
}
```

```
$ phpunit CountTest
PHPUnit latest.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.
```

13.10 assertDirectoryExists()

`assertDirectoryExists(string $directory[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если каталог, указанный `$directory`, не существует.

`assertDirectoryNotExists()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.11: Использование assertDirectoryExists()

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

class DirectoryExistsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryExists('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryExistsTest
PHPUnit latest.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.
```

13.11 assertDirectoryIsReadable()

`assertDirectoryIsReadable(string $directory[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если каталог, указанный `$directory`, не является каталогом или не доступен для чтения.

`assertDirectoryNotIsReadable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.12: Использование assertDirectoryIsReadable()

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

class DirectoryIsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryIsReadable('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryIsReadableTest
PHPUnit latest.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.
```

13.12 assertDirectoryIsWritable()

`assertDirectoryIsWritable(string $directory[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если каталог, указанный `$directory`, не является каталогом или не доступен для записи.

`assertDirectoryNotIsWritable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.13: Использование assertDirectoryIsWritable()

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

class DirectoryIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryIsWritable('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryIsWritableTest
PHPUnit latest.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.
```

13.13 assertEmpty()

`assertEmpty(mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является пустым.

`assertNotEmpty()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeEmpty()` и `assertAttributeNotEmpty()` — удобные обёртки, которые могут применяться к общедоступному (`public`), защищённому (`protected`) или закрытому (`private`) атрибуту класса или объекта.

Пример 13.14: Использование assertEmpty()

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

class EmptyTest extends TestCase
{
```

```
    public function testFailure()
    {
        $this->assertEmpty(['foo']);
    }
}
```

```
$ phpunit EmptyTest
PHPUnit latest.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.
```

13.14 assertEqualXMLStructure()

`assertEqualXMLStructure(DOMElement $expectedElement, DOMElement $actualElement[, boolean $checkAttributes = false, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если XML-структура объекта DOMElement в `$actualElement` не равна XML-структуре объекта DOMElement в `$expectedElement`.

Пример 13.15: Использование assertEqualXMLStructure()

```
<?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 latest.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.
```

13.15 assertEquals()

`assertEquals(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если две переменные `$expected` и `$actual` не равны.

`assertNotEquals()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeEquals()` and `assertAttributeNotEquals()` — удобные обёртки, которые используют общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.16: Использование assertEquals()

```php
<?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 latest.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.
```

См. ниже более конкретные сравнения, используемые для определённых типов `$expected` и `$actual`.

`assertEquals(float $expected, float $actual[, string $message = '', float $delta = 0])`

Сообщает об ошибке, определённой в `$message`, если абсолютная разница между двумя числами с плавающей точкой `$expected` и `$actual` больше, чем `$delta`. Если абсолютная разница между двумя числами с плавающей точкой `$expected` и `$actual` меньше *или равно* `$delta`,то утверждение пройдёт успешно.

Прочитайте «What Every Computer Scientist Should Know About Floating-Point Arithmetic» для понимания, зачем требуется `$delta`.

Пример 13.17: Использование assertEquals() с числом с плавающей точкой

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

```php
class EqualsTest extends TestCase
{
    public function testSuccess()
    {
        $this->assertEquals(1.0, 1.1, '', 0.1);
    }

    public function testFailure()
    {
        $this->assertEquals(1.0, 1.1);
    }
}
```

```
$ phpunit EqualsTest
PHPUnit latest.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 = ''])`

Сообщает об ошибке, определённой в `$message`, если каноническая форма XML-документов, представленная двумя объектами DOMDocument `$expected` и `$actual`, не равна.

Пример 13.18: Использование assertEquals() с объектами DOMDocument

```php
<?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 latest.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 = ''])`

Сообщает об ошибке, определённой в `$message`, если два объекта `$expected` и `$actual` не имеют одинаковых значений атрибутов.

Пример 13.19: Использование assertEquals() с объектами

```php
<?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 latest.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 = ''])`

Сообщает об ошибке, определённой в `$message`, если два массива `$expected` и `$actual` не равны.

Пример 13.20: Использование assertEquals() с массивом

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

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEquals(['a', 'b', 'c'], ['a', 'c', 'd']);
    }
}
```

```
$ phpunit EqualsTest
PHPUnit latest.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.
```

13.16 assertFalse()

`assertFalse(bool $condition[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$condition` равняется `true`.

`assertNotFalse()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.21: Использование assertFalse()

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

class FalseTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFalse(true);
    }
}
```

```
$ phpunit FalseTest
PHPUnit latest.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.
```

13.17 assertFileEquals()

`assertFileEquals(string $expected, string $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл, указанный в `$expected`, не имеет того же содержимого, что и файл, переданный в `$actual`.

`assertFileNotEquals()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.22: Использование assertFileEquals()

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

class FileEqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileEquals('/home/sb/expected', '/home/sb/actual');
    }
}
```

```
$ phpunit FileEqualsTest
PHPUnit latest.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.
```

13.18 assertFileExists()

`assertFileExists(string $filename[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл, указанный в `$filename`, не существует.

`assertFileNotExists()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.23: Использование assertFileExists()

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

class FileExistsTest extends TestCase
{
```

```
    public function testFailure()
    {
        $this->assertFileExists('/path/to/file');
    }
}
```

```
$ phpunit FileExistsTest
PHPUnit latest.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.
```

13.19 assertFileIsReadable()

`assertFileIsReadable(string $filename[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл, указанный в `$filename`, не является файлом или не доступен для чтения.

`assertFileNotIsReadable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.24: Использование assertFileIsReadable()

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

class FileIsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileIsReadable('/path/to/file');
    }
}
```

```
$ phpunit FileIsReadableTest
PHPUnit latest.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.
```

13.20 assertFileIsWritable()

`assertFileIsWritable(string $filename[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл, указанный в `$filename`, не является файлом или не доступен для записи.

`assertFileNotIsWritable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.25: Использование assertFileIsWritable()

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

class FileIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileIsWritable('/path/to/file');
    }
}
```

```
$ phpunit FileIsWritableTest
PHPUnit latest.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.
```

13.21 assertGreaterThan()

`assertGreaterThan(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actual` не превышает значение `$expected`.

`assertAttributeGreaterThan()` - удобная обёртка, которая использует общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.26: Использование assertGreaterThan()

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

class GreaterThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertGreaterThan(2, 1);
    }
}
```

```
$ phpunit GreaterThanTest
PHPUnit latest.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.
```

13.22 assertGreaterThanOrEqual()

`assertGreaterThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actual` не больше или равно значению `$expected`.

`assertAttributeGreaterThanOrEqual()` - удобная обёртка, которая использует общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.27: Использование assertGreaterThanOrEqual()

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

class GreatThanOrEqualTest extends TestCase
{
    public function testFailure()
```

```
    {
        $this->assertGreaterThanOrEqual(2, 1);
    }
}
```

```
$ phpunit GreaterThanOrEqualTest
PHPUnit latest.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.
```

13.23 assertInfinite()

`assertInfinite(mixed $variable[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$variable` не является `INF`.

`assertFinite()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.28: Использование assertInfinite()

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

class InfiniteTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInfinite(1);
    }
}
```

```
$ phpunit InfiniteTest
PHPUnit latest.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.
```

13.24 assertInstanceOf()

`assertInstanceOf($expected, $actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является экземпляром `$expected`.

`assertNotInstanceOf()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeInstanceOf()` и `assertAttributeNotInstanceOf()` - удобные обёртки, которые могут применяться к общедоступному (`public`), защищённому (`protected`) или закрытому (`private`) атрибуту класса или объекта.

Пример 13.29: Использование assertInstanceOf()

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

class InstanceOfTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInstanceOf(RuntimeException::class, new Exception);
    }
}
```

```
$ phpunit InstanceOfTest
PHPUnit latest.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.
```

13.25 assertIsArray()

`assertIsArray($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `array`.

`assertIsNotArray()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.30: Использование assertIsArray()

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

class ArrayTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsArray(null);
    }
}
```

```
$ phpunit ArrayTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ArrayTest::testFailure
Failed asserting that null is of type "array".

/home/sb/ArrayTest.php:8

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

13.26 assertIsBool()

`assertIsBool($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `bool`.

`assertIsNotBool()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.31: Использование assertIsBool()

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

class BoolTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsBool(null);
    }
}
```

```
$ phpunit BoolTest
```

```
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) BoolTest::testFailure
Failed asserting that null is of type "bool".

/home/sb/BoolTest.php:8

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

13.27 assertIsCallable()

`assertIsCallable($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `callable`.

`assertIsNotCallable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.32: Использование assertIsCallable()

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

class CallableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsCallable(null);
    }
}
```

```
$ phpunit CallableTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) CallableTest::testFailure
Failed asserting that null is of type "callable".

/home/sb/CallableTest.php:8

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

13.28 assertIsFloat()

`assertIsFloat($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `float`.

`assertIsNotFloat()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.33: Использование assertIsFloat()

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

class FloatTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsFloat(null);
    }
}
```

```
$ phpunit FloatTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FloatTest::testFailure
Failed asserting that null is of type "float".

/home/sb/FloatTest.php:8

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

13.29 assertIsInt()

`assertIsInt($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `int`.

`assertIsNotInt()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.34: Использование assertIsInt()

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

class IntTest extends TestCase
{
    public function testFailure()
```

```
    {
        $this->assertIsInt(null);
    }
}
```

```
$ phpunit IntTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IntTest::testFailure
Failed asserting that null is of type "int".

/home/sb/IntTest.php:8

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

13.30 assertIsIterable()

`assertIsIterable($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `iterable`.

`assertIsNotIterable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.35: Использование assertIsIterable()

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

class IterableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsIterable(null);
    }
}
```

```
$ phpunit IterableTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) IterableTest::testFailure
```

```
Failed asserting that null is of type "iterable".

/home/sb/IterableTest.php:8

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

13.31 assertIsNumeric()

`assertIsNumeric($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `numeric`.

`assertIsNotNumeric()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.36: Использование assertIsNumeric()

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

class NumericTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsNumeric(null);
    }
}
```

```
$ phpunit NumericTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NumericTest::testFailure
Failed asserting that null is of type "numeric".

/home/sb/NumericTest.php:8

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

13.32 assertIsObject()

`assertIsObject($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `object`.

`assertIsNotObject()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.37: Использование assertIsObject()

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

class ObjectTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsObject(null);
    }
}
```

```
$ phpunit ObjectTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ObjectTest::testFailure
Failed asserting that null is of type "object".

/home/sb/ObjectTest.php:8

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

13.33 assertIsResource()

`assertIsResource($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `resource`.

`assertIsNotResource()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.38: Использование assertIsResource()

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

class ResourceTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsResource(null);
    }
}
```

```
$ phpunit ResourceTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.
```

```
F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ResourceTest::testFailure
Failed asserting that null is of type "resource".

/home/sb/ResourceTest.php:8

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

13.34 assertIsScalar()

`assertIsScalar($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `scalar`.

`assertIsNotScalar()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.39: Использование assertIsScalar()

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

class ScalarTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsScalar(null);
    }
}
```

```
$ phpunit ScalarTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ScalarTest::testFailure
Failed asserting that null is of type "scalar".

/home/sb/ScalarTest.php:8

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

13.35 assertIsString()

`assertIsString($actual[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$actual` не является типом `string`.

`assertIsNotString()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.40: Использование assertIsString()

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

class StringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsString(null);
    }
}
```

```
$ phpunit StringTest
PHPUnit latest.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringTest::testFailure
Failed asserting that null is of type "string".

/home/sb/StringTest.php:8

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

13.36 assertIsReadable()

`assertIsReadable(string $filename[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл или каталог, указанный в `$filename`, не доступен для чтения.

`assertNotIsReadable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.41: Использование assertIsReadable()

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

class IsReadableTest extends TestCase
```

```
{
    public function testFailure()
    {
        $this->assertIsReadable('/path/to/unreadable');
    }
}
```

```
$ phpunit IsReadableTest
PHPUnit latest.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.
```

13.37 assertIsWritable()

`assertIsWritable(string $filename[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл или каталог, указанный в `$filename`, не доступен для записи.

`assertNotIsWritable()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.42: Использование assertIsWritable()

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

class IsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsWritable('/path/to/unwritable');
    }
}
```

```
$ phpunit IsWritableTest
PHPUnit latest.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.
```

13.38 assertJsonFileEqualsJsonFile()

`assertJsonFileEqualsJsonFile(mixed $expectedFile, mixed $actualFile[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actualFile` не соответствует значению `$expectedFile`.

Пример 13.43: Использование assertJsonFileEqualsJsonFile()

```php
<?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 latest.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",␣
↪"Linux"]".

/home/sb/JsonFileEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

13.39 assertJsonStringEqualsJsonFile()

`assertJsonStringEqualsJsonFile(mixed $expectedFile, mixed $actualJson[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actualJson` не соответствует значению `$expectedFile`.

Пример 13.44: Использование assertJsonStringEqualsJsonFile()

```
<?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 latest.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.
```

13.40 assertJsonStringEqualsJsonString()

`assertJsonStringEqualsJsonString(mixed $expectedJson, mixed $actualJson[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actualJson` не соответствует значению `$expectedJson`.

Пример 13.45: Использование assertJsonStringEqualsJsonString()

```
<?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 latest.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.
```

13.41 assertLessThan()

`assertLessThan(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actual` больше значения `$expected`.

`assertAttributeLessThan()` — удобная обёртка, которая использует общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.46: Использование assertLessThan()

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

class LessThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThan(1, 2);
    }
```

```
}
```

```
$ phpunit LessThanTest
PHPUnit latest.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.
```

13.42 assertLessThanOrEqual()

`assertLessThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если значение `$actual` больше или не равно значению `$expected`.

`assertAttributeLessThanOrEqual()` — удобная обёртка, которая использует общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.47: Использование assertLessThanOrEqual()

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

class LessThanOrEqualTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThanOrEqual(1, 2);
    }
}
```

```
$ phpunit LessThanOrEqualTest
PHPUnit latest.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.
```

13.43 assertNan()

`assertNan(mixed $variable[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$variable` не является `NAN`.

Пример 13.48: Использование assertNan()

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

class NanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertNan(1);
    }
}
```

```
$ phpunit NanTest
PHPUnit latest.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.
```

13.44 assertNull()

`assertNull(mixed $variable[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$variable` не является `null`.

`assertNotNull()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.49: Использование assertNull()

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

class NullTest extends TestCase
{
    public function testFailure()
    {
        $this->assertNull('foo');
    }
}
```

```
$ phpunit NotNullTest
PHPUnit latest.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.
```

13.45 assertObjectHasAttribute()

`assertObjectHasAttribute(string $attributeName, object $object[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$object->attributeName` не существует.

`assertObjectNotHasAttribute()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.50: Использование assertObjectHasAttribute()

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

class ObjectHasAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertObjectHasAttribute('foo', new stdClass);
    }
}
```

```
$ phpunit ObjectHasAttributeTest
PHPUnit latest.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.
```

13.46 assertRegExp()

`assertRegExp(string $pattern, string $string[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$string` не соответствует регулярному выражению `$pattern`.

`assertNotRegExp()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.51: Использование assertRegExp()

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

class RegExpTest extends TestCase
{
    public function testFailure()
    {
        $this->assertRegExp('/foo/', 'bar');
    }
}
```

```
$ phpunit RegExpTest
PHPUnit latest.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.
```

13.47 assertStringMatchesFormat()

`assertStringMatchesFormat(string $format, string $string[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$string` не соответствует строке формата в `$format`.

`assertStringNotMatchesFormat()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.52: Использование assertStringMatchesFormat()

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

class StringMatchesFormatTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormat('%i', 'foo');
    }
}
```

```
$ phpunit StringMatchesFormatTest
PHPUnit latest.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.
```

Строка формата может содержать следующие заполнители:

- `%e`: Представляет разделитель каталогов, например `/` в Linux.
- `%s`: Один или несколько чего-либо (символ или пробел), кроме символа конца строки.
- `%S`: Ноль или более чего-либо (символ или пробел), кроме символа конца строки.
- `%a`: Один или несколько чего-либо (символ или пробел), включая символ конца строки.
- `%A`: Ноль или более чего-либо (символ или пробел), включая символ конца строки.
- `%w`: Ноль или более символов пробела.
- `%i`: Целое число со знаком, например `+3142`, `-3142`.
- `%d`: Целое число без знака, например `123456`.
- `%x`: Один или более шестнадцатеричного символа. То есть, символы в диапазоне `0-9`, `a-f`, `A-F`.
- `%f`: Число с плавающей точкой, например: `3.142`, `-3.142`, `3.142E-10`, `3.142e+10`.

- %c: Один символ любого типа.
- %%: Буквальный символ процента: %.

13.48 assertStringMatchesFormatFile()

`assertStringMatchesFormatFile(string $formatFile, string $string[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$string` не соответствует содержимому `$formatFile`.

`assertStringNotMatchesFormatFile()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.53: Использование assertStringMatchesFormatFile()

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

class StringMatchesFormatFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormatFile('/path/to/expected.txt', 'foo');
    }
}
```

```
$ phpunit StringMatchesFormatFileTest
PHPUnit latest.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.
```

13.49 assertSame()

`assertSame(mixed $expected, mixed $actual[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если две переменные `$expected` и `$actual` не имеют одного и того же типа и значения.

`assertNotSame()` — это противоположный этому утверждению, принимающий те же самые аргументы.

`assertAttributeSame()` и `assertAttributeNotSame()` — удобные обёртки, которые используют общедоступный (`public`), защищённый (`protected`) или закрытый (`private`) атрибут класса или объекта в качестве фактического значения.

Пример 13.54: Использование assertSame()

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame('2204', 2204);
    }
}
```

```
$ phpunit SameTest
PHPUnit latest.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 = ''])`

Сообщает об ошибке, определённой в `$message`, если две переменные `$expected` и `$actual` ссылаются не на один и тот же объект.

Пример 13.55: Использование assertSame() с объектами

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame(new stdClass, new stdClass);
    }
}
```

```
$ phpunit SameTest
PHPUnit latest.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.
```

13.50 assertStringEndsWith()

`assertStringEndsWith(string $suffix, string $string[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$string` не заканчивается на `$suffix`.

`assertStringEndsNotWith()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.56: Использование assertStringEndsWith()

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

class StringEndsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEndsWith('suffix', 'foo');
    }
}
```

```
$ phpunit StringEndsWithTest
PHPUnit latest.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.
```

13.51 assertStringEqualsFile()

`assertStringEqualsFile(string $expectedFile, string $actualString[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если файл, указанный в `$expectedFile`, не имеет `$actualString` в качестве его содержимого.

`assertStringNotEqualsFile()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.57: Использование assertStringEqualsFile()

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

class StringEqualsFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEqualsFile('/home/sb/expected', 'actual');
    }
}
```

```
$ phpunit StringEqualsFileTest
PHPUnit latest.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.
```

13.52 assertStringStartsWith()

`assertStringStartsWith(string $prefix, string $string[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$string` не начинается с `$prefix`.

`assertStringStartsNotWith()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.58: Использование assertStringStartsWith()

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

class StringStartsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringStartsWith('prefix', 'foo');
    }
}
```

```
$ phpunit StringStartsWithTest
PHPUnit latest.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.
```

13.53 assertThat()

Более сложные утверждения могут быть сформулированы с использованием классов `PHPUnit\Framework\Constraint`. Их можно вычислить с помощью метода `assertThat()`. Пример 13.59 показывает, как ограничения `logicalNot()` и `equalTo()` могут использоваться для выражения того же утверждения, что и `assertNotEquals()`.

`assertThat(mixed $value, PHPUnit\Framework\Constraint $constraint[, $message = ''])`

Сообщает об ошибке, определённой в `$message`, если `$value` не соответствует `$constraint`.

Пример 13.59: Использование assertThat()

```
<?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)
            )
          );
    }
}
```

Таблица 13.1 показывает доступные классы PHPUnit\Framework\Constraint.

Ограничение
PHPUnit\Framework\Constraint\Attribute attribute(PHPUnit\Framework\Constraint $constraint, $attributeNa
PHPUnit\Framework\Constraint\IsAnything anything()
PHPUnit\Framework\Constraint\ArrayHasKey arrayHasKey(mixed $key)
PHPUnit\Framework\Constraint\TraversableContains contains(mixed $value)
PHPUnit\Framework\Constraint\TraversableContainsOnly containsOnly(string $type)
PHPUnit\Framework\Constraint\TraversableContainsOnly containsOnlyInstancesOf(string $classname)
PHPUnit\Framework\Constraint\IsEqual equalTo($value, $delta = 0, $maxDepth = 10)
PHPUnit\Framework\Constraint\Attribute attributeEqualTo($attributeName, $value, $delta = 0, $maxDepth =
PHPUnit\Framework\Constraint\DirectoryExists directoryExists()
PHPUnit\Framework\Constraint\FileExists fileExists()
PHPUnit\Framework\Constraint\IsReadable isReadable()
PHPUnit\Framework\Constraint\IsWritable isWritable()
PHPUnit\Framework\Constraint\GreaterThan greaterThan(mixed $value)
PHPUnit\Framework\Constraint\Or greaterThanOrEqual(mixed $value)
PHPUnit\Framework\Constraint\ClassHasAttribute classHasAttribute(string $attributeName)
PHPUnit\Framework\Constraint\ClassHasStaticAttribute classHasStaticAttribute(string $attributeName)
PHPUnit\Framework\Constraint\ObjectHasAttribute objectHasAttribute(string $attributeName)
PHPUnit\Framework\Constraint\IsIdentical identicalTo(mixed $value)
PHPUnit\Framework\Constraint\IsFalse isFalse()
PHPUnit\Framework\Constraint\IsInstanceOf isInstanceOf(string $className)
PHPUnit\Framework\Constraint\IsNull isNull()
PHPUnit\Framework\Constraint\IsTrue isTrue()
PHPUnit\Framework\Constraint\IsType isType(string $type)
PHPUnit\Framework\Constraint\LessThan lessThan(mixed $value)
PHPUnit\Framework\Constraint\Or lessThanOrEqual(mixed $value)
logicalAnd()
logicalNot(PHPUnit\Framework\Constraint $constraint)
logicalOr()
logicalXor()
PHPUnit\Framework\Constraint\PCREMatch matchesRegularExpression(string $pattern)
PHPUnit\Framework\Constraint\StringContains stringContains(string $string, bool $case)
PHPUnit\Framework\Constraint\StringEndsWith stringEndsWith(string $suffix)
PHPUnit\Framework\Constraint\StringStartsWith stringStartsWith(string $prefix)

13.54 assertTrue()

assertTrue(bool $condition[, string $message = ''])

Сообщает об ошибке, определённой в $message, если $condition равно false.

`assertNotTrue()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.60: Использование assertTrue()

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

class TrueTest extends TestCase
{
    public function testFailure()
    {
        $this->assertTrue(false);
    }
}
```

```
$ phpunit TrueTest
PHPUnit latest.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.
```

13.55 assertXmlFileEqualsXmlFile()

`assertXmlFileEqualsXmlFile(string $expectedFile, string $actualFile[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если XML-документ в `$actualFile` не равен XML-документу в `$expectedFile`.

`assertXmlFileNotEqualsXmlFile()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.61: Использование assertXmlFileEqualsXmlFile()

```
<?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 latest.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.
```

13.56 assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если XML-документ в `$actualXml` не равен XML-документу в `$expectedFile`.

`assertXmlStringNotEqualsXmlFile()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.62: Использование assertXmlStringEqualsXmlFile()

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

class XmlStringEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlFile(
          '/home/sb/expected.xml', '<foo><baz/></foo>');
    }
}
```

```
$ phpunit XmlStringEqualsXmlFileTest
PHPUnit latest.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.
```

13.57 assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])`

Сообщает об ошибке, определённой в `$message`, если XML-документ в `$actualXml` не равен XML-документу в `$expectedXml`.

`assertXmlStringNotEqualsXmlString()` — это противоположный этому утверждению, принимающий те же самые аргументы.

Пример 13.63: Использование assertXmlStringEqualsXmlString()

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

class XmlStringEqualsXmlStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlString(
          '<foo><bar/></foo>', '<foo><baz/></foo>');
    }
}
```

```
$ phpunit XmlStringEqualsXmlStringTest
PHPUnit latest.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.
```

Глава 14

Аннотации

Аннотация — специальная форма синтаксических метаданных, которые могут добавлены в исходный код некоторых языков программирования. Хотя у PHP нет собственной языковой возможности для аннотирования исходного кода, использование тегов, таких как `@annotation arguments` в блоке документации (альтернативное название — докблок), было принято в сообществе PHP для аннотации кода. В PHP блоки документации «рефлексивны»: к ним можно получить доступ с помощью метода API Reflection `getDocComment()` на уровне функции, класса, методе и атрибуте. Такие приложения, как PHPUnit, используют информацию во время выполнения для настройки их поведения.

Примечание

Комментарий документации в PHP должен начинаться с `/**` и заканчиваться `*/`. Аннотации в любом другом стиле комментария будут проигнорированы.

В этом приложении представлены все разновидности аннотаций, поддерживаемые PHPUnit.

14.1 @author

Аннотация `@author` — это псевдоним для аннотации `@group` (см. *@group*), позволяющая фильтровать тесты на основе их авторов.

14.2 @after

Аннотацию `@after` можно использовать для указания методов, которые должны вызываться после каждого тестового метода в тестовом классе.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
```

```
    /**
     * @after
     */
    public function tearDownSomeFixtures()
    {
        // ...
    }

    /**
     * @after
     */
    public function tearDownSomeOtherFixtures()
    {
        // ...
    }
}
```

14.3 @afterClass

Аннотацию `@afterClass` можно использовать для указания статических методов, которые должны вызываться после того, как все тестовые методы в тестовом классе были запущены для очистки общих фикстур.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @afterClass
     */
    public static function tearDownSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @afterClass
     */
    public static function tearDownSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

14.4 @backupGlobals

Операции резервного копирования и восстановления глобальных переменных могут быть полностью отключены для всех тестов в тестовом классе следующим образом:

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
```

```
 */
class MyTest extends TestCase
{
    // ...
}
```

Аннотацию `@backupGlobals` также можно использовать на уровне тестового метода. Это позволяет выполнять тонкую настройку операций резервного копирования и восстановления:

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testThatInteractsWithGlobalVariables()
    {
        // ...
    }
}
```

14.5 @backupStaticAttributes

Аннотацию `@backupStaticAttributes` можно использовать для резервного копирования всех значений статических свойств во всех объявленных классах перед каждым тестом с последующим их восстановлением. Она может использоваться на уровне тестового класса или тестового метода:

```
use PHPUnit\Framework\TestCase;

/**
 * @backupStaticAttributes enabled
 */
class MyTest extends TestCase
{
    /**
     * @backupStaticAttributes disabled
     */
    public function testThatInteractsWithStaticAttributes()
    {
        // ...
    }
}
```

Примечание

Аннотация `@backupStaticAttributes` ограничивается внутренне PHP и при определённых условиях может привести к непреднамеренному сохранению статических значений и утечке памяти в последующих тестах.

См. *Глобальное состояние* дополнительной информации.

14.6 @before

Аннотацию `@before` можно использовать для указания методов, которые должны вызываться перед каждым тестовым методом в тестовом классе.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @before
     */
    public function setupSomeFixtures()
    {
        // ...
    }

    /**
     * @before
     */
    public function setupSomeOtherFixtures()
    {
        // ...
    }
}
```

14.7 @beforeClass

Аннотацию `@beforeClass` можно использовать для указания статических методов, которые должны вызываться до выполнения любых тестов в тестовом классе для настройки общих фикстур.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @beforeClass
     */
    public static function setUpSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @beforeClass
     */
    public static function setUpSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

14.8 @codeCoverageIgnore*

Аннотации `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` и `@codeCoverageIgnoreEnd` могут использоваться для исключения строк кода из анализа покрытия.

Для использования см. *Игнорирование блоков кода*.

14.9 @covers

Аннотация `@covers` может использовать в тестовом коде для указания, какие методы собирается тестировать данный тестовый метод:

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

Если эта аннотация задана, будет учитываться информация о покрытии кода только для указанных методов.

Таблица 14.1 показывает синтаксис аннотации `@covers`.

Таблица 14.1: Аннотации для указания, какие методы покрываются тестом

Аннотация	Описание
`@covers ClassName::methodName`	Указывает, что аннотированный тестовый метод покрывает указанный метод.
`@covers ClassName`	Указывает, что аннотированный тестовый метод покрывает все методы данного класса.
`@covers ClassName<extended>`	Указывает, что аннотированный тестовый метод покрывает все методы заданного класса и его родительских классов или интерфейсов.
`@covers ClassName::<public>`	Указывает, что аннотированный тестовый метод покрывает все общедоступные методы заданного класса.
`@covers ClassName::<protected>`	Указывает, что аннотированный тестовый метод покрывает все защищённые методы заданного класса.
`@covers ClassName::<private>`	Указывает, что аннотированный тестовый метод покрывает все закрытые методы заданного класса.
`@covers ClassName::<!public>`	Указывает, что аннотированный тестовый метод покрывает все не общедоступные методы заданного класса.
`@covers ClassName::<!protected>`	Указывает, что аннотированный тестовый метод покрывает все не защищённые методы заданного класса.
`@covers ClassName::<!private>`	Указывает, что аннотированный тестовый метод покрывает все не закрытые методы заданного класса.
`@covers ::functionName`	Указывает, что аннотированный тестовый метод покрывает указанную глобальную функцию.

14.10 @coversDefaultClass

Аннотацию `@coversDefaultClass` можно использовать для указания пространства имени по умолчанию или класса. Таким образом, длинные имена не нужно повторно указывать для каждой аннотации `@covers`. См. Пример 14.1.

Пример 14.1: Использование @coversDefaultClass для сокращений аннотаций

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

/**
 * @coversDefaultClass \Foo\CoveredClass
 */
class CoversDefaultClassTest extends TestCase
{
    /**
     * @covers ::publicMethod
     */
    public function testSomething()
    {
        $o = new Foo\CoveredClass;
        $o->publicMethod();
    }
}
```

14.11 @coversNothing

Аннотацию `@coversNothing` можно использовать в тестовом коде для указания, что информация о покрытии кода не должна учитываться для данного тестового класса.

Это можно использовать для интеграционного тестирования. См. *Тест, который указывает, что ни один метод не должен быть покрыт* для примера.

Данную аннотацию можно использовать на уровне классе или метода и переопределить любые теги `@covers`.

14.12 @dataProvider

Тестовый метод может принимать произвольное количество аргументов. Эти аргументы должны быть предоставлены одним или несколькими методами провайдера данных (`provider()` в *Использование провайдера данных, который возвращает массив массивов*). Используемый метод провайдера данных задаётся с помощью аннотации `@dataProvider`.

См. *Провайдеры данных* для получения подробной информации.

14.13 @depends

PHPUnit поддерживает объявление явных зависимостей между тестовыми методами. Такие зависимости не определяют порядок, в котором должны выполняться тестовые методы, но они позволяют

возвращать экземпляр фикстуры теста продюсером и передавать его зависимым потребителям. *Использование аннотации @depends для описания зависимостей* показывает, как использовать аннотацию `@depends` для выражения зависимостей между тестовыми методами.

См. *Зависимости тестов* для подробной информации.

14.14 @doesNotPerformAssertions

Предотвращает выполнение теста, не выполняющего никаких утверждений, для того чтобы не считать его рискованным.

14.15 @expectedException

Использование метода expectException() показывает, как использовать аннотацию `@expectedException` для проверки того, было ли выброшено исключение внутри тестируемого кода.

См. *Тестирование исключений* для получения подробной информации.

14.16 @expectedExceptionCode

Аннотация `@expectedExceptionCode` в сочетании с `@expectedException` позволяет делать утверждения по коду ошибке выбрасываемого исключения, таким образом, сужая конкретное исключение.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException     MyException
     * @expectedExceptionCode 20
     */
    public function testExceptionHasErrorCode20()
    {
        throw new MyException('Сообщение исключения', 20);
    }
}
```

Для облегчения тестирования и уменьшения дублирования можно указать константу класса в `@expectedExceptionCode`, используя синтаксис «`@expectedExceptionCode ClassName::CONST`».

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
      * @expectedException     MyException
      * @expectedExceptionCode MyClass::ERRORCODE
      */
    public function testExceptionHasErrorCode20()
    {
      throw new MyException('Сообщение исключения', 20);
    }
```

```
}
class MyClass
{
    const ERRORCODE = 20;
}
```

14.17 @expectedExceptionMessage

Аннотация `@expectedExceptionMessage` работает аналогично `@expectedExceptionCode`, поскольку она может сделать утверждение на сообщении исключения.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException        MyException
     * @expectedExceptionMessage Сообщение исключения
     */
    public function testExceptionHasRightMessage()
    {
        throw new MyException('Сообщение исключения', 20);
    }
}
```

Ожидаемое сообщение может быть подстрокой сообщения исключения. Это может быть полезно, для того чтобы только утверждать, что переданное определённое имя или параметр встречается в исключении, не фокусируясь на полном совпадении сообщения исключения в тесте.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException        MyException
      * @expectedExceptionMessage broken
      */
     public function testExceptionHasRightMessage()
     {
         $param = 'broken';
         throw new MyException('Некорректный параметр "' . $param . '".', 20);
     }
}
```

Для облегчения тестирования и уменьшения дублирования можно указать константу класса в `@expectedExceptionMessage`, используя синтаксис «`@expectedExceptionMessage ClassName::CONST`». Для примера можно посмотреть на *@expectedExceptionCode*.

14.18 @expectedExceptionMessageRegExp

Ожидаемое сообщение также можно указать в виде регулярного выражения, используя аннотацию `@expectedExceptionMessageRegExp`. Это полезно в ситуациях, когда подстрока не подходит для соответствия заданному сообщению.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException              MyException
      * @expectedExceptionMessageRegExp /Аргумент \d+ не может быть целым ? \w+/u
      */
     public function testExceptionHasRightMessage()
     {
         throw new MyException('Аргумент 2 не может быть целым числом');
     }
}
```

14.19 @group

Тест может быть отмечен как принадлежащий одной или нескольким группам, используя аннотацию `@group` следующим образом:

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
    }

    /**
     * @group regresssion
     * @group bug2204
     */
    public function testSomethingElse()
    {
    }
}
```

Аннотацию `@group` можно задать для тестового класса. Затем она будет «унаследована» всеми методами этого тестового класса.

Тесты могут быть выбраны для выполнения на основе групп с использованием опций командной строки исполнителя тестов `--group` и `--exclude-group` или используя соответствующие директивы конфигурационного XML-файла.

14.20 @large

Аннотация `@large` — псевдоним для `@group large`.

Если пакет `PHP_Invoker` установлен и включён строгий режим, большой тест завершится неудачно, если для его выполнения потребуется более 60 секунд. Этот тайм-аут настраивается через атрибут `timeoutForLargeTests` в конфигурационном XML-файле.

14.21 @medium

Аннотация `@medium` — псевдоним для `@group medium`. Средний тест не должен зависеть от теста, отмеченного как `@large`.

Если пакет `PHP_Invoker` установлен и включён строгий режим, средний тест завершится неудачно, если для его выполнения потребуется более 10 секунд. Этот тайм-аут настраивается через атрибут `timeoutForMediumTests` в конфигурационном XML-файле.

14.22 @preserveGlobalState

Когда тест запускается в отдельном процессе, PHPUnit попытается сохранить глобальное состояние из родительского процесса, сериализуя все глобальные переменные в родительском процессе и десериализуя их в дочернем процессе. Это может вызвать проблемы, если родительский процесс содержит глобальные переменные, которые невозможно сериализовать. Для исправления этого, вы можете запретить PHPUnit сохранять глобальное состояние с помощью аннотации `@preserveGlobalState`.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     * @preserveGlobalState disabled
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

14.23 @requires

Аннотация `@requires` можно использовать для пропуска тестов, когда общие предварительные условия, такие как версия PHP или установленные расширения, не выполняются.

Полный список возможностей и примеров можно найти в *Возможные примеры использования @requires*

14.24 @runTestsInSeparateProcesses

Указывает, что все тесты в тестовом классе должны выполняться в отдельном процессе PHP.

```
use PHPUnit\Framework\TestCase;

/**
 * @runTestsInSeparateProcesses
 */
class MyTest extends TestCase
{
```

```
    // ...
}
```

Примечание: По умолчанию PHPUnit пытается сохранить глобальное состояние из родительского процесса, сериализуя все глобальные переменные в родительском процессе и десериализуя их в дочернем процессе. Это может вызвать проблемы, если родительский процесс содержит глобальные переменные, которые невозможно сериализовать. См. *@preserveGlobalState* для получения информации по изменению этого поведения.

14.25 @runInSeparateProcess

Указывает, что тест должен выполняться в отдельном процессе PHP.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

Примечание: По умолчанию PHPUnit пытается сохранить глобальное состояние из родительского процесса, сериализуя все глобальные переменные в родительском процессе и десериализуя их в дочернем процессе. Это может вызвать проблемы, если родительский процесс содержит глобальные переменные, которые невозможно сериализовать. См. *@preserveGlobalState* для получения информации по изменению этого поведения.

14.26 @small

Аннотация `@small` — это псевдоним для `@group small`. Небольшой тест не должен зависеть от теста, отмеченного как `@medium` или `@large`.

Если пакет `PHP_Invoker` установлен и включён строгий режим, небольшой тест завершится неудачно, если для его выполнения потребуется более 1 секунды. Этот тайм-аут настраивается через атрибут `timeoutForSmallTests` в конфигурационном XML-файле.

Примечание

Тесты должны быть явно аннотированы либо `@small`, `@medium` или `@large` для включения ограничения времени выполнения.

14.27 @test

В качестве альтернативы добавления префиксов именам тестовым методам `test`, вы можете использовать аннотацию `@test` в блоке документации метода, чтобы отметить его как тестовый метод.

```
/**
 * @test
 */
public function initialBalanceShouldBe0()
{
    $this->assertSame(0, $this->ba->getBalance());
}
```

14.28 @testdox

Указывает альтернативное описание, используемое при создании предложений для agile-документации.

Аннотацию `@testdox` можно применять как к тестовым классам, так и к тестовым методам.

```
/**
 * @testdox A bank account
 */
class BankAccountTest extends TestCase
{
    /**
     * @testdox has an initial balance of zero
     */
    public function balanceIsInitiallyZero()
    {
        $this->assertSame(0, $this->ba->getBalance());
    }
}
```

Примечание

До PHPUnit 7.0 (из-за бага в разборе аннотации) использование аннотации `@testdox` также активировало поведение аннотацию `@test`.

```

```

14.29 @testWith

Вместо реализации метода для использования с `@dataProvider`, вы можете определить набор данных, используя аннотацию `@testWith`.

Набор данных состоит из одного или нескольких элементов. Для определения набора данных с несколькими элементами, определите каждый элемент на отдельной строке. Каждый элемент набора данных должен быть массив, определённым в JSON.

См. *Провайдеры данных* для получения дополнительной информации о передачи набора данных в тест.

```
/**
 * @param string    $input
 * @param int       $expectedLength
 *
 * @testWith        ["test", 4]
 *                  ["longer-string", 13]
 */
public function testStringLength(string $input, int $expectedLength)
{
    $this->assertSame($expectedLength, strlen($input));
}
```

Представление объекта в JSON будет преобразовано в ассоциативный массив.

```
/**
 * @param array     $array
 * @param array     $keys
 *
 * @testWith        [{"day": "monday", "conditions": "sunny"}, ["day", "conditions"]]
 */
public function testArrayKeys($array, $keys)
{
    $this->assertSame($keys, array_keys($array));
}
```

14.30 @ticket

Аннотация `@ticket` — это псевдоним для аннотации `@group` (см. *@group*) и позволяет фильтровать тесты на основе их идентификатора тикета.

14.31 @uses

Аннотация `@uses` указывает код, который будет выполняться тестом, но не предназначен для покрытия тестом. Хорошим примером может быть объект значения (value object), который необходим для тестирования единицы (модуля) кода.

```
/**
 * @covers BankAccount::deposit
 * @uses   Money
 */
public function testMoneyCanBeDepositedInAccount()
{
    // ...
}
```

Эта аннотация особенно полезна в режиме строгого режима, когда непреднамеренно покрытый код приводит тесте к неудаче. См. *Непреднамеренно покрытый код* для получения дополнительной информации о строгом режиме покрытия.

Глава 15

Конфигурационный XML-файл

15.1 PHPUnit

Атрибуты элемента <phpunit> можно использовать для настройки основной функциональности PHPUnit.

```
<phpunit
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/latest/phpunit.xsd"
         backupGlobals="true"
         backupStaticAttributes="false"
         <!--bootstrap="/path/to/bootstrap.php"-->
         cacheResult="false"
         cacheTokens="false"
         colors="false"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         forceCoversAnnotation="false"
         printerClass="PHPUnitTextUIResultPrinter"
         <!--printerFile="/path/to/ResultPrinter.php"-->
         processIsolation="false"
         stopOnError="false"
         stopOnFailure="false"
         stopOnIncomplete="false"
         stopOnSkipped="false"
         stopOnRisky="false"
         testSuiteLoaderClass="PHPUnitRunnerStandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
         timeoutForSmallTests="1"
         timeoutForMediumTests="10"
         timeoutForLargeTests="60"
         verbose="false">
```

```
  <!-- ... -->
</phpunit>
```

Конфигурация XML выше соответствует поведению по умолчанию исполнителя тестов TextUI, описанному в *Опции командной строки*.

Дополнительные опции, недоступные в качестве опций командной строки:

`convertErrorsToExceptions`

По умолчанию PHPUnit установит обработчик ошибок, которые преобразует следующие ошибки в исключения:

- `E_WARNING`
- `E_NOTICE`
- `E_USER_ERROR`
- `E_USER_WARNING`
- `E_USER_NOTICE`
- `E_STRICT`
- `E_RECOVERABLE_ERROR`
- `E_DEPRECATED`
- `E_USER_DEPRECATED`

Установите `convertErrorsToExceptions` в `false` для отключения этой возможности.

`convertNoticesToExceptions`

Когда установлено значение `false`, обработчик ошибок, установленный `convertErrorsToExceptions`, не будет преобразовывать ошибки `E_NOTICE`, `E_USER_NOTICE` или `E_STRICT` в исключения.

`convertWarningsToExceptions`

Когда установлено значение `false`, обработчик ошибок, установленный `convertErrorsToExceptions`, не будет преобразовывать ошибки `E_WARNING` или `E_USER_WARNING` в исключения.

`forceCoversAnnotation`

Покрытие кода будет записываться только для тестов, в которых используется аннотация `@covers`, задокументированная в *@covers*.

`timeoutForLargeTests`

Если применяется ограничение по времени, основанное на размере теста, тогда этот атрибут устанавливает тайм-аут для всех тестов, отмеченных как `@large`. Если тест не завершится в течение установленного тайм-аута, он завершится неудачей.

`timeoutForMediumTests`

Если применяется ограничение по времени, основанное на размере теста, тогда этот атрибут устанавливает тайм-аут для всех тестов, отмеченных как `@medium`. Если тест не завершится в течение установленного тайм-аута, он завершится неудачей.

`timeoutForSmallTests`

Если применяется ограничение по времени, основанное на размере теста, тогда этот атрибут установит тайм-аут для всех тестов, не отмеченных как `@medium` или `@large`. Если тест не завершится в течение установленного тайм-аута, он завершится неудачей.

15.2 Набор тестов

Элемент `<testsuites>` и его один или несколько дочерних элементов `<testsuite>` можно использовать для составления набора тестов из наборов тестов и тестовых классов.

```
<testsuites>
  <testsuite name="Мой набор тестов">
    <directory>/path/to/*Test.php files</directory>
    <file>/path/to/MyTest.php</file>
    <exclude>/path/to/exclude</exclude>
  </testsuite>
</testsuites>
```

Используя атрибуты `phpVersion` и `phpVersionOperator` можно указать требуемую версию PHP. В приведённом ниже примере будут добавлены только файлы `/path/to/*Test.php` и файл `/path/to/MyTest.php`, если версия PHP не менее 5.3.0.

```
<testsuites>
  <testsuite name="My Test Suite">
    <directory suffix="Test.php" phpVersion="5.3.0" phpVersionOperator=">=">/path/to/files</
↪directory>
    <file phpVersion="5.3.0" phpVersionOperator=">=">/path/to/MyTest.php</file>
  </testsuite>
</testsuites>
```

Атрибут `phpVersionOperator` не является обязательным и по умолчанию `>=`.

15.3 Группы

Элемент `<groups>` и его дочерние элементы `<include>`, `<exclude>` и `<group>` можно использовать для выбора групп тестов, отмеченных аннотацией `@group` (описанных в *@group*), которые должны (или не должны) выполняться.

```
<groups>
  <include>
    <group>name</group>
  </include>
  <exclude>
    <group>name</group>
  </exclude>
</groups>
```

Вышеприведённая конфигурация XML соответствует вызову исполнителя тестов TextUI со следующими опциями:

- `--group name`
- `--exclude-group name`

15.4 Файлы в белом списке для покрытия кода

Элемент `<filter>` и его дочерние элементы можно использовать для настройки белого списка при создании отчёта о покрытии кода.

```
<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>
```

15.5 Логирование

Элемент `<logging>` и его дочерние элементы `<log>` можно использовать для настройки логирования выполнения тестов.

```
<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"/>
  <log type="testdox-html" target="/tmp/testdox.html"/>
  <log type="testdox-text" target="/tmp/testdox.txt"/>
</logging>
```

Вышеприведённая конфигурация XML соответствует вызову исполнителя тестов TextUI со следующими опциями:

- `--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`

Атрибуты `lowUpperBound`, `highLowerBound`, `showUncoveredFiles` не имеет эквивалента опции исполнителя тестов TextUI.

- `lowUpperBound`: Максимальный процент покрытия, который считается «низко» покрытым.
- `highLowerBound`: Минимальный процент покрытия, который считается «высоко» покрытым.
- `showUncoveredFiles`: Показать все файлы в белом списке при выводе с опцией `--coverage-text`, а не только те, для которых есть информация о покрытии.
- `showOnlySummary`: Показать только краткую сводку в выводе при использовании `--coverage-text`.

15.6 Обработчики тестов

Элемент `<listeners>` и его дочерние элементы `<listener>` можно использовать для присоединения дополнительных обработчиков теста к выполнению теста.

```
<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>
```

Вышеприведённая конфигурация XML соответствует прикреплению объекта `$listener` (см. ниже) к выполнению теста:

```
$listener = new MyListener(
    ['Sebastian'],
    22,
    'April',
    19.78,
    null,
    new stdClass
);
```

15.7 Регистрация расширений TestRunner

Элемент `<extensions>` и его дочерние элементы `<extension>` можно использовать для регистрации пользовательских расширений TestRunner.

Пример 15.1 показывает, как зарегистрировать такое расширение.

Пример 15.1: Регистрация расширения TestRunner

```
  <?xml version="1.0" encoding="UTF-8"?>
  <phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
↪"https://schema.phpunit.de/7.1/phpunit.xsd">
      <extensions>
          <extension class="Vendor\MyExtension"/>
      </extensions>
  </phpunit>
```

15.8 Установка INI-настроек, констант и глобальных переменных PHP

Элемент <php> и его дочерние элементы можно использовать для настройки параметров, констант и глобальных переменных PHP. Он может также использоваться для добавления новых путей в опцию include_path.

```
<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>
```

Вышеприведённая конфигурация XML соответствует следующем коду PHP:

```
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';
```

Глава 16

Библиография

[Astels2003] David Astels. *Test Driven Development.*

[Beck2002] Kent Beck. *Test Driven Development by Example.*[1]

[Meszaros2007] Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code.*[2]

[1] Кент Бек. «Экстремальное программирование. Разработка через тестирование». Питер, 2017.
[2] Джерард Месарош. «Шаблоны тестирования xUnit: рефакторинг кода тестов». Вильямс, 2009.

Глава 17

Авторские права

```
Это доступное для человеческого понимания резюме Юридического Кодекса
(полная лицензия) ниже.

=====================================================================

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/.

======================================================================
```