PHPUnit Manual

リリース *latest*

Sebastian Bergmann

2021 年 08 月 22 日

目次

対応バージョン：PHPUnit latest

更新日：2021 年 08 月 22 日

Sebastian Bergmann

目次

第 1 章

PHPUnit のインストール

1.1 要件

PHPUnit latest は PHP 7.3 以降のバージョンで動作しますが、最新版の PHP を使うことを強く推奨します。

PHPUnit を使うには、拡張モジュール dom、json、が必要です。これらは通常、デフォルトで有効になっています。

PHPUnit また、拡張モジュール pcre、reflection、そして spl も必要です。これらは標準の拡張モジュールとしてデフォルトで有効になっており、PHP のビルドシステムやソースファイルに手を加えない限り、無効にすることはできません。

コードカバレッジをサポートするには Xdebug 2.7.0 以降と tokenizer 拡張モジュールが必要です。XML 形式で情報を出力するには、xmlwriter 拡張モジュールも必要です。

1.2 PHP Archive (PHAR)

PHPUnit を入手する一番簡単な方法は、PHP Archive (PHAR) をダウンロードすることです。必要な依存コンポーネントがすべて (オプションのコンポーネントの一部も含めて) ひとつのファイルにまとめられています。

PHP Archives (PHAR) を利用するには、phar 拡張モジュールが必要です。

PHAR の `--self-update` 機能を使うには、openssl 拡張モジュールが必要です。

Suhosin 拡張モジュールが有効になっている場合は、`php.ini` で PHAR の実行を許可する必要があります。

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

PHPUnit の PHAR は、ダウンロードするだけですぐに使えます。

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

PHAR ファイルに実行可能属性をつけておくのが一般的です。

```
$ 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 PHPUnit の PHAR リリースの検証

PHPUnit プロジェクトが配布する公式リリースにはすべて、リリースマネージャーによる署名がついています。検証用の PGP 署名と SHA256 ハッシュは、phar.phpunit.de から取得できます。

リリースの検証をどのように行うのかについて、説明しましょう。まず、`phpunit.phar` をダウンロードし、さらにその PGP 署名 `phpunit.phar.asc` もダウンロードします。

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

ダウンロードした PHPUnit の PHP Archive (`phpunit-x.y.phar`) を、署名 (`phpunit-x.y.phar.asc`) で検証します。

```
$ 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
```

リリースマネージャーの公開鍵 (`6372C20A`) が、ローカルシステム上に存在しないようです。検証を進めるには、リリースマネージャーの公開鍵を、鍵サーバーから取得する必要があります。鍵サーバーには、たとえば `pgp.uni-mainz.de` などがあります。公開鍵サーバーはお互いリンクしあっているので、どの鍵サーバーを使ってもかまいません。

```
$ curl --silent https://sebastian-bergmann.de/gpg.asc | gpg --import
gpg: key 4AA394086372C20A: 452 signatures not checked due to missing keys
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 4AA394086372C20A: public key "Sebastian Bergmann <sb@sebastian-bergmann.de>"␣
↪imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: no ultimately trusted keys found
```

これで、" Sebastian Bergmann <sb@sebastian-bergmann.de> " さんの公開鍵を取得できました。ただ、この鍵を作ったのが本当に Sebastian Bergmann という人なのかは、確かめようがありません。ともあれ、もう一度リリースの署名を検証してみましょう。

```
$ 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
```

とりあえず、署名が正しいことはわかりました。ただ、この署名が信頼できるものであるかどうかは、まだわかりません。ここで言う「署名が正しい」とは、リリースのファイルが改ざんされていないということです。しかし、公開鍵暗号方式の性質上、これだけでは不十分です。`6372C20A` を作ったのが Sebastian Bergmann 本人であることを、確かめる必要があります。

公開鍵を作って公開鍵サーバーにアップロードするのは、誰にだってできることです。当然、悪意のある攻撃者にも可能なことです。攻撃者は、このニセの鍵を使って署名した、悪意のあるリリースを作ることもできます。このリリース (そして署名) をダウンロードして検証すると、成功するでしょう。なぜならその公開鍵は、悪意のある攻撃者が作ったニセの鍵だからです。こういったことを防ぐために、鍵の作者も検証しなければいけないのです。公開鍵の作者を検証する方法については、このマニュアルの範囲を超えるので、割愛します。

GPG を使っていちいち署名を検証したり PHPUnit の PHAR を検証したりするのはつまらない作業です。そこで作られたのが PHIVE で、これは PHAR のインストールや検証状況を管理するためのツールです。詳しくは ウェブサイト をご覧ください。

1.3 Composer

Composer を使ってプロジェクトの依存関係を管理するには、`phpunit/phpunit` への (開発時の) 依存情報をプロジェクトの `composer.json` ファイルに追加します。

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

1.4 グローバルなインストール

PHPUnit をグローバルに（たとえば `/usr/bin/phpunit` や`/usr/local/bin/phpunit` などとして）インストールすることはおすすめできません。

PHPUnit はプロジェクト単位でローカルな依存として管理すべきです。

PHPUnit の特定のバージョンの PHAR をプロジェクトの `tools` ディレクトリに置く（そして PHIVE で管理する）か、Composer を使っているならそのプロジェクトで必要とする PHPUnit のバージョンを `composer.json` に書きましょう。

1.5 Webserver

PHPUnit はテストを実行するためのフレームワークであり、また、コマンドラインツールです。テストのコーディングおよび実行は開発時に行うものであるため、ウェブサーバに PHPUnit をインストールする理由は全くありません。

ウェブサーバに PHPUnit をインストールすると、そのデプロイは壊れた状態となります。より一般的な話をすると、 `vendor` ディレクトリをウェブサーバ上で公開したときにも同様にそのデプロイは壊れた状態となります。

PHPUnit をウェブサーバにアップロードすることは良くないことであると認識してください。警告はしましたからね。

第 2 章

PHPUnit 用のテストの書き方

Example 2.1 で、PHP の配列操作のテストを PHPUnit 用に書く方法を示します。この例では、PHPUnit を使ったテストを書く際の基本的な決まり事や手順を紹介します。

1. `Class` という名前のクラスのテストは、`ClassTest` という名前のクラスに記述します。
2. `ClassTest` は、(ほとんどの場合) `PHPUnit\Framework\TestCase` を継承します。
3. テストは、`test*` という名前のパブリックメソッドとなります。

   あるいは、`@test` アノテーションをメソッドのコメント部で使用することで、それがテストメソッドであることを示すこともできます。
4. テストメソッドの中で `assertSame()` のようなアサーションメソッド (**アサーション** を参照ください) を使用して、期待される値と実際の値が等しいことを確かめます。

Example 2.1　PHPUnit での配列操作のテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StackTest extends TestCase
{
    public function testPushAndPop(): void
    {
        $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));
    }
}
```

Martin Fowler:

> Whenever you are tempted to type something into a `print` statement or a debugger expression, write it as a test instead.
>
> 何かを `print` 文やデバッガの式に書きたくなったときは、代わりにその内容をテストに書くようにするんだ。

2.1 テストの依存性

> *Adrian Kuhn et. al.*:
>
> Unit Tests are primarily written as a good practice to help developers identify and fix bugs, to refactor code and to serve as documentation for a unit of software under test. To achieve these benefits, unit tests ideally should cover all the possible paths in a program. One unit test usually covers one specific path in one function or method. However a test method is not necessarily an encapsulated, independent entity. Often there are implicit dependencies between test methods, hidden in the implementation scenario of a test.
>
> ユニットテストを書くそもそもの目的は、バグの発見と修正やコードのリファクタリングを開発者がやりやすくすること。そしてテスト対象のソフトウェアのドキュメントとしての役割を果たすことだ。これらの目的を達成するためには、ユニットテストがプログラム内のすべてのルートをカバーしていることが理想である。ひとつのユニットテストがカバーするのは、通常はひとつの関数やメソッド内の特定のルートだけとなる。しかし、テストメソッドは必ずしもカプセル化して独立させる必要はない。複数のテストメソッドの間に暗黙の依存性があって、隠された実装シナリオがテストの中にあるのもよくあることだ。

PHPUnit は、テストメソッド間の依存性の明示的な宣言をサポートしています。この依存性とは、テストメソッドが実行される順序を定義するものではありません。プロデューサーがテストフィクスチャを作ってそのインスタンスを返し、依存するコンシューマーがそれを受け取って利用するというものです。

- プロデューサーとは、返り値としてテスト対象のユニットを生成するテストメソッドのこと。
- コンシューマーとは、プロデューサーの返り値に依存するテストメソッドのこと。

Example 2.2 は、`@depends` アノテーションを使ってテストメソッドの依存性をあらわす例です。

Example 2.2 `@depends` アノテーションを使った依存性の表現

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StackTest extends TestCase
{
    public function testEmpty(): array
    {
        $stack = [];
```

```
        $this->assertEmpty($stack);

        return $stack;
    }

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

        return $stack;
    }

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

上の例では、まず最初のテスト `testEmpty()` で新しい配列を作り、それが空であることを確かめます。このテストは、フィクスチャを返します。二番目のテスト `testPush()` は `testEmpty()` に依存しており、依存するテストの結果を引数として受け取ります。最後の `testPop()` は `testPush()` に依存しています。

Note

プロデューサーの生成する戻り値は、デフォルトでは「そのままの形式」でコンシューマーに渡されます。つまり、プロデューサーがオブジェクトを戻した場合は、そのオブジェクトへの参照がコンシューマーに渡されるということです。参照を使う代わりに、 `@depends clone` を用いてディープコピーをしたり`@depends shallowClone` を用いてシャローコピー（PHP のキーワード `clone` によるコピー）をしたりすることもできます。

問題の局所化を手早く行うには、失敗したテストに目を向けやすくしたいものです。そのため PHPUnit では、あるテストが失敗したときにはそのテストに依存する他のテストの実行をスキップします。テスト間の依存性を活用して問題点を見つけやすくしている例を Example 2.3 に示します。

Example 2.3　テストの依存性の活用

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

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

```
$ 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.
```

ひとつのテストに複数の `@depends` アノテーションをつけることもできます。PHPUnit はテストが実行される順序を変更しないので、テストが実行されるときに確実に依存性が満たされているようにしておく必要があります。

複数の @depends アノテーションを持つテストは、最初のプロデューサーからのフィクスチャを最初の引数、二番目のプロデューサーからのフィクスチャを二番目の引数、……として受け取ります。Example 2.4 を参照ください。

Example 2.4　複数の依存性を持つテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class MultipleDependenciesTest extends TestCase
{
    public function testProducerFirst(): string
    {
        $this->assertTrue(true);

        return 'first';
    }

    public function testProducerSecond(): string
    {
        $this->assertTrue(true);

        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     */
    public function testConsumer(string $a, string $b): void
    {
        $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 データプロバイダ

テストメソッドには任意の引数を渡すことができます。この引数は、データプロバイダメソッド (Example 2.5 の `additionProvider()`) で指定します。使用するデータプロバイダメソッドを指定するには`@dataProvider` アノテーションを使用します。

データプロバイダメソッドは、`public` でなければなりません。また、メソッドの返り値の型は、配列の配列あるいはオブジェクト (`Iterator` インターフェイスを実装しており、反復処理の際に配列を返すもの) である必要があります。この返り値の各要素に対して、その配列の中身を引数としてテストメソッドがコールされます。

Example 2.5　配列の配列を返すデータプロバイダの使用

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public function additionProvider(): array
    {
        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.
```

大量のデータセットを使う場合は、デフォルトの数字を使うのではなく、各データセットに文字列の名前をつけておくと便利です。出力もよりわかりやすくなり、テストを失敗させたデータセットの名前もわかるようになります。

Example 2.6　データプロバイダでの名前つきデータセットの使用

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public function additionProvider(): array
    {
        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 4.6.0 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.
```

Example 2.7　Iterator オブジェクトを返すデータプロバイダの使用

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public function additionProvider(): CsvFileIterator
    {
        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.
```

Example 2.8　CsvFileIterator クラス

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

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

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

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

    public function rewind(): void
    {
        rewind($this->file);

        $this->current = fgetcsv($this->file);
        $this->key = 0;
    }

    public function valid(): bool
    {
        return !feof($this->file);
    }

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

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

    public function next(): void
    {
        $this->current = fgetcsv($this->file);

        $this->key++;
```

```
    }
}
```

@dataProvider で指定したメソッドと@depends で指定したテストの両方からの入力を受け取るテストの場合、データプロバイダからの引数のほうが依存するテストからの引数より先にきます。依存するテストからの引数は、どちらのデータセットに対しても同じになります。Example 2.9 を参照ください。

Example 2.9　同じテストでの @depends と @dataProvider の組み合わせ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public function testProducerFirst(): string
    {
        $this->assertTrue(true);

        return 'first';
    }

    public function testProducerSecond(): string
    {
        $this->assertTrue(true);

        return 'second';
    }

    /**
     * @depends testProducerFirst
     * @depends testProducerSecond
     * @dataProvider provider
     */
    public function testConsumer(): void
    {
        $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:31

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

Example 2.10　ひとつのテストでの複数のデータプロバイダの使用

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public function additionWithNonNegativeNumbersProvider(): array
    {
```

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

    public function additionWithNegativeNumbersProvider(): array
    {
        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.
```

Note

あるテストがデータプロバイダを使う別のテストに依存している場合、別のテストで少なくともひとつのデータセットに対するテストが成功すればそのテストも実行されます。データプロバイダを使ったテストの結果をそのテストに注入することはできません。

Note

すべてのデータプロバイダを実行してから、静的メソッド `setUpBeforeClass()` や `setUp()` メソッドの最初の呼び出しが発生します。そのため、これらのメソッドで作った変数にデータプロバイダ内からアクセスすることはできません。そうなっている理由は、PHPUnit がテストの総数を算出できるようにするためです。

2.3 例外のテスト

Example 2.11 は、テストするコード内で例外がスローされたかどうかを `expectException()` メソッドを使用して調べる方法を示すものです。

Example 2.11　expectException() メソッドの使用法

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class ExceptionTest extends TestCase
{
    public function testException(): void
    {
        $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.
```

`expectException()` メソッドのほかにも `expectExceptionCode()`、`expectExceptionMessage()`、`expectExceptionMessageMatches()` といったメソッドで、テスト対象のコードで発生するであろう例外をテストできます。

Note

`expectExceptionMessage()` は `$actual` の中に`$expected` のメッセージが含まれるかどうかを確かめるだけのものであり、完全一致するかどうかを確かめるわけではないことに注意しましょう。

2.4 PHP のエラーのテスト

デフォルトでは、PHPUnit はテストの実行中に発生した PHP のエラーや警告そして notice を例外に変換します。これらの例外を用いて、たとえば Example 2.12 のように PHP のエラーが発生することをテストできます。

Note

PHP の実行時設定 `error_reporting` を使うと、PHPUnit がどのエラーを例外に変換するのかを制限できます。この機能に関して何か問題がでた場合は、PHP の設定を見直し、調べたいと思っているエラーを抑制するようになっていないかどうか確認しましょう。

Example 2.12　PHP のエラーが発生することのテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;
final class ErrorTest extends TestCase
{
    public function testDeprecationCanBeExpected(): void
    {
        $this->expectDeprecation();
        // Optionally test that the message is equal to a string
        $this->expectDeprecationMessage('foo');
        // Or optionally test that the message matches a regular expression
        $this->expectDeprecationMessageMatches('/foo/');
        \trigger_error('foo', \E_USER_DEPRECATED);
    }
    public function testNoticeCanBeExpected(): void
    {
        $this->expectNotice();
        // Optionally test that the message is equal to a string
        $this->expectNoticeMessage('foo');
        // Or optionally test that the message matches a regular expression
        $this->expectNoticeMessageMatches('/foo/');
        \trigger_error('foo', \E_USER_NOTICE);
    }
    public function testWarningCanBeExpected(): void
    {
        $this->expectWarning();
```

```
        // Optionally test that the message is equal to a string
        $this->expectWarningMessage('foo');
        // Or optionally test that the message matches a regular expression
        $this->expectWarningMessageMatches('/foo/');
        \trigger_error('foo', \E_USER_WARNING);
    }
    public function testErrorCanBeExpected(): void
    {
        $this->expectError();
        // Optionally test that the message is equal to a string
        $this->expectErrorMessage('foo');
        // Or optionally test that the message matches a regular expression
        $this->expectErrorMessageMatches('/foo/');
        \trigger_error('foo', \E_USER_ERROR);
    }
}
```

エラーを引き起こすような PHP の関数、たとえば `fopen` などに依存するテストを行うときには、テスト中にエラーを抑制できれば便利なことがあります。そうすれば、notice のせいで `PHPUnit\Framework\Error\Notice` が出てしまうことなく、返り値だけをチェックできるようになります。

Example 2.13　PHP のエラーが発生するコードの返り値のテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class ErrorSuppressionTest extends TestCase
{
    public function testFileWriting(): void
    {
        $writer = new FileWriter;

        $this->assertFalse(@$writer->write('/is-not-writeable/file', 'stuff'));
    }
}

final 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)
```

もしエラーを抑制しなければ、このテストは失敗して `fopen(/is-not-writeable/file): failed to open stream: No such file or directory` となります。

2.5 出力内容のテスト

メソッドの実行結果を確かめる方法として、(`echo` や `print` などによる) 出力が期待通りのものかを調べたいこともあるでしょう。`PHPUnit\Framework\TestCase` クラスは、PHP の出力バッファリング 機能を使用してこの仕組みを提供します。

Example 2.14 では、期待する出力内容を `expectOutputString()` メソッドで設定する方法を示します。期待通りの出力が得られなかった場合は、そのテストは失敗という扱いになります。

Example 2.14　関数やメソッドの出力内容のテスト

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class OutputTest extends TestCase
{
    public function testExpectFooActualFoo(): void
    {
        $this->expectOutputString('foo');

        print 'foo';
    }

    public function testExpectBarActualBaz(): void
    {
        $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.
```

Table 2.1 は、テストの出力用に提供するメソッドをまとめたものです。

Table2.1　テストの出力用のメソッド

メソッド	意味
`void expectOutputRegex(string $regularExpression)`	出力が正規表現 `$regularExpression` にマッチするであろうという予測を設定します。
`void expectOutputString(string $expectedString)`	出力が文字列 `$expectedString` と等しくなるであろうという予測を設定します。
`bool setOutputCallback(callable $callback)`	たとえば出力時の正規化などに使用するコールバック関数を設定します。
`string getActualOutput()`	実際の出力を取得します。

Note

strict モードでは、出力を発生させるテストは失敗します。

2.6 エラー出力

テストが失敗した場合、PHPUnit は、状況を可能な限り詳細に報告します。これが、何が問題だったのかを調べるのに役立つでしょう。

Example 2.15　配列の比較に失敗したときのエラー出力

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class ArrayDiffTest extends TestCase
{
    public function testEquality(): void
    {
        $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.
```

この例では配列の要素のうちひとつだけが異なっています。それ以外の値も表示することで、どこが悪かったのかをわかりやすくしています。

出力が長すぎる場合は PHPUnit が出力を分割し、違っている部分の前後数行だけを出力します。

Example 2.16　要素数の多い配列の比較に失敗したときのエラー出力

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class LongArrayDiffTest extends TestCase
{
    public function testEquality(): void
    {
        $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
@@ @@
      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 エッジケース

比較に失敗したときに、PHPUnit は入力値をテキスト形式にしてこれを比較します。この実装が原因で、実際の違う箇所よりも多くの問題を報告してしまうことがあります。

この問題が発生するのは、`assertEquals()` などの「緩い」比較の関数を、配列やオブジェクトに対して使った場合だけです。

Example 2.17　緩い比較を使った場合の diff の生成のエッジケース

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class ArrayWeakComparisonTest extends TestCase
{
    public function testEquality(): void
    {
        $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.
```

この例では、最初のインデックスの `1` と `'1'` がエラー報告されていますが、 `assertEquals()` ではこれらを等しいとみなしているはずです。

第 3 章

コマンドラインのテストランナー

phpunit コマンドを実行すると、PHPUnit のコマンドライン版テストランナーが起動します。コマンドラインのテストランナーを使用したテストの様子を以下に示します。

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

..

Time: 0 seconds

OK (2 tests, 2 assertions)
```

このように実行すると、PHPUnit のコマンドラインテストランナーは、まず現在の作業ディレクトリにあるソースファイル ArrayTest.php を探してそれを読み込み、テストケースクラス ArrayTest を探します。そして、そのクラス内のテストを実行します。

テストがひとつ実行されるたびに、PHPUnit コマンドラインツールはその経過を示す文字を出力します。

.
: テストが成功した際に表示されます。

F
: テストメソッドの実行中、アサーションに失敗した際に表示されます。

E
: テストメソッドの実行中、エラーが発生した際に表示されます。

R
: テストが危険だとマークされている場合に表示されます (**リスクを伴うテスト** を参照ください)。

`S`

テストが飛ばされた場合に表示されます (**不完全なテスト・テストの省略** を参照ください)。

`I`

テストが「不完全」あるいは「未実装」とマークされている場合に表示されます (**不完全なテスト・テストの省略** を参照ください)。

PHPUnit は、**失敗** *(failures)* と**エラー** *(errors)* を区別します。「失敗」は PHPUnit のアサーションに違反した場合、つまり例えば `assertSame()` のコールに失敗した場合などで、「エラー」は予期せぬ例外や PHP のエラーが発生した場合となります。この区別は、時に有用です。というのは「エラー」は一般的に「失敗」より修正しやすい傾向があるからです。もし大量の問題が発生した場合は、まず「エラー」を最初に片付け、その後で「失敗」を修正していくのが最良の方法です。

3.1 コマンドラインオプション

以下のコードで、コマンドライン版テストランナーのオプションの一覧を見てみましょう。

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

Usage:
  phpunit [options] 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
  --coverage-cache <dir>      Cache static analysis results
  --warm-coverage-cache       Warm static analysis cache
  --coverage-filter <dir>     Include <dir> in code coverage analysis
  --path-coverage             Perform path coverage analysis
  --disable-coverage-ignore   Disable annotations for ignoring code coverage
```

```
  --no-coverage               Ignore code coverage configuration

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
  --no-logging                Ignore logging configuration

Test Selection Options:
  --filter <pattern>          Filter which tests to run
  --testsuite <name>          Filter which testsuite to run
  --group <name>              Only runs tests from the specified group(s)
  --exclude-group <name>      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 <suffixes>    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-incomplete         Treat incomplete tests as failures
  --fail-on-risky              Treat risky tests as failures
  --fail-on-skipped            Treat skipped tests as failures
  --fail-on-warning            Treat tests with warnings as failures
  -v|--verbose                 Output more verbose information
  --debug                      Display debugging information

  --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)
  --no-interaction             Disable TestDox progress animation
  --printer <printer>          TestListener implementation to use

  --order-by <order>           Run tests in order: default|defects|duration|no-
↪depends|random|reverse|size
  --random-order-seed <N>      Use a specific random seed <N> for random order
  --cache-result               Write test results to cache file
  --do-not-cache-result        Do not write test results to cache file

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)
```

```
  --extensions <extensions>   A comma separated list of PHPUnit extensions to␣
↪load
  --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
  --cache-result-file <file>  Specify result cache path and filename
  --generate-configuration    Generate configuration file with suggested␣
↪settings
  --migrate-configuration     Migrate configuration file to current format

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`

`UnitTest` という名前のクラスで定義されているテストを実行します。このクラスは、`UnitTest.php` という名前のファイルの中に定義されているものとします。

`UnitTest` は、`PHPUnit\Framework\TestCase` を継承したクラスであるか、あるいは `PHPUnit\Framework\Test` オブジェクト、例えば `PHPUnit\Framework\TestSuite` のインスタンスを返す `public static suite()` というメソッドを保持するクラスでなければなりません。

`phpunit UnitTest UnitTest.php`

`UnitTest` という名前のクラスで定義されているテストを実行します。このクラスは、指定したファイルの中で定義されているものとします。

`--coverage-clover`

テスト結果から XML 形式のログファイルを作成し、コードカバレッジ情報もそこに含めます。詳細は**コードカバレッジ解析** を参照ください。

`--coverage-crap4j`

コードカバレッジレポートを Crap4j 形式で作成します。詳細は**コードカバレッジ解析** を参照ください。

`--coverage-html`

コードカバレッジレポートを HTML 形式で作成します。詳細は**コードカバレッジ解析** を参照ください。

`--coverage-php`

シリアライズした PHP_CodeCoverage オブジェクトを生成し、コードカバレッジ情報もそこに含めます。

`--coverage-text`

テストを実行したときに、ログファイルあるいはコマンドライン出力で可読形式のコードカバレッジ情報を生成します。

`--log-junit`

JUnit XML フォーマットを使用して、テストの実行結果のログを作成します。

`--testdox-html` および `--testdox-text`

実行したテストについて、HTML あるいはプレーンテキスト形式のドキュメントを生成します (*TestDox* を参照ください)。

`--filter`

指定した正規表現パターンにマッチする名前のテストのみを実行します。パターンがデリミタで囲まれていない場合は、PHPUnit はパターンをデリミタ / で囲みます。

マッチするテスト名は、次のいずれかのフォーマットになります。

`TestNamespace\TestCaseClass::testMethod`

デフォルトのテスト名のフォーマットは、テストメソッドの中でマジック定数__METHOD__ を使うのと同等です。

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

テストがデータプロバイダーを持つ場合、データを処理するたびに、現在のインデックスをデフォルトのテスト名の後に続けたものを取得します。

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

テストが持つデータプロバイダーが名前つきセットを使う場合、データを処理するたびに、現在の名前をデフォルトのテスト名の後に続けたものを取得します。名前つきデータセットの例は Example 3.1 を参照ください。

Example 3.1　名前つきデータセット

```
<?php
namespace TestNamespace;

use PHPUnit\Framework\TestCase;

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

PHPT のテストのテスト名は、ファイルシステムのパスになります。

有効なフィルターパターンの例は、Example 3.2 を参照ください。

Example 3.2　フィルターパターンの例

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

データプロバイダーのマッチングに使えるショートカットは、Example 3.3 を参照ください。

Example 3.3　フィルターのショートカット

```
--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

指定したパターンにマッチする名前のテストスイートのみを実行します。

--group

指定したグループのテストのみを実行します。あるテストを特定のグループに所属させるには、@group アノテーションを使用します。

`@author` アノテーションおよび `@ticket` アノテーションは`@group` のエイリアスで、それぞれテストの作者およびチケット ID に基づいてテストをフィルタリングします。

`--exclude-group`

指定したグループをテストの対象外とします。あるテストを特定のグループに所属させるには、`@group` アノテーションを使用します。

`--list-groups`

使用可能なテストグループの一覧を表示します。

`--test-suffix`

指定したサフィックスのテストファイルだけを探します。

`--dont-report-useless-tests`

何もテストをしないテストについて報告しません。詳細は **リスクを伴うテスト** を参照ください。

`--strict-coverage`

意図せずカバーしているコードについて厳格にチェックします。詳細は **リスクを伴うテスト** を参照ください。

`--strict-global-state`

グローバルな状態の変更について厳格にチェックします。詳細は **リスクを伴うテスト** を参照ください。

`--disallow-test-output`

実行中に何かを出力するテストについて厳格にチェックします。詳細は **リスクを伴うテスト** を参照ください。

`--disallow-todo-tests`

docblock に `@todo` アノテーションが指定されているテストを実行しません。

`--enforce-time-limit`

テストのサイズに応じて、制限時間を設定します。詳細は **リスクを伴うテスト** を参照ください。

`--process-isolation`

各テストを個別の PHP プロセスで実行します。

`--no-globals-backup`

$GLOBALS のバックアップ・リストアを行いません。詳細は **グローバルな状態** を参照ください。

`--static-backup`

ユーザ定義クラスの静的属性のバックアップ・リストアを行います。詳細は **グローバルな状態** を参照ください。

`--colors`

出力に色を使用します。Windows では、ANSICON あるいは ConEmu を利用します。

指定できる値は、以下の三つです。

- `never`: 出力を色分けしません。これは、`--colors` オプションを省略したときのデフォルトです。
- `auto`: ターミナルが色をサポートしていない場合や、出力をコマンドにパイプしたりファイルにリダイレクトしたりする場合を除き、出力に色を使います。
- `always`: ターミナルが色をサポートしていない場合や、出力をコマンドにパイプしたりファイルにリダイレクトしたりする場合も含めて、常に出力に色を使います。

`--colors` だけを指定して何も値を指定しなかった場合は、`auto` を選んだものとみなされます。

`--stderr`

オプションで、出力先を `STDOUT` ではなく `STDERR` にします。

`--stop-on-error`

最初にエラーが発生した時点で実行を停止します。

`--stop-on-failure`

最初にエラーあるいは失敗が発生した時点で実行を停止します。

`--stop-on-risky`

最初に危険なテストがあらわれた時点で実行を停止します。

`--stop-on-skipped`

最初にテストのスキップが発生した時点で実行を停止します。

`--stop-on-incomplete`

最初に不完全なテストがあらわれた時点で実行を停止します。

`--verbose`

より詳細な情報を出力します。例えば、未完成のテストや省略したテストの名前が表示されます。

`--debug`

テスト名などのデバッグ情報を、テストの実行開始時に出力します。

`--loader`

PHPUnit\Runner\TestSuiteLoader を実装したクラスのうち、実際に使用するものを指定します。

標準のテストスイートローダーは、現在の作業ディレクトリおよび PHP の設定項目 include_path で指定されているディレクトリからソースファイルを探します。Project_Package_Class クラスがソースファイル Project/Package/Class.php に対応します。

--repeat

指定された回数だけ、繰り返しテストを実行します。

--testdox

テストの進行状況を、アジャイルな文書として報告します。(*TestDox*) を参照ください。

--printer

結果を表示するために使うプリンタクラスを指定します。このプリンタクラスは PHPUnit\Util\Printer を継承し、かつ PHPUnit\Framework\TestListener インターフェイスを実装したものでなければなりません。

--bootstrap

テストの前に実行される“ ブートストラップ ”PHP ファイルを指定します。

--configuration, -c

設定を XML ファイルから読み込みます。詳細は *XML* **設定ファイル** を参照ください。

phpunit.xml あるいは phpunit.xml.dist (この順番で使用します) が現在の作業ディレクトリに存在しており、かつ --configuration が使われて **いない** 場合、設定が自動的にそのファイルから読み込まれます。

指定されているのがディレクトリで、phpunit.xml あるいは phpunit.xml.dist (この順番で使用します) がそのディレクトリ内に存在する場合、設定が自動的にそのファイルから読み込まれます。

--no-configuration

現在の作業ディレクトリにある phpunit.xml および phpunit.xml.dist を無視します。

--include-path

PHP の include_path の先頭に、指定したパスを追加します。

-d

指定した PHP 設定オプションの値を設定します。

Note

これらのオプションは、引数の後に指定することもできます。

3.2 TestDox

PHPUnit の TestDox 機能は、テストクラス内のすべてのテストメソッドの名前を抽出し、それを PHP 風のキャメルケースから通常の文に変換します。つまり `testBalanceIsInitiallyZero()` が “ Balance is initially zero ”のようになるわけです。最後のほうの数字のみが違うメソッド、例えば `testBalanceCannotBecomeNegative()` と `testBalanceCannotBecomeNegative2()` のようなものが存在した場合は、文 “ Balance cannot become negative ” は一度のみ表示され、全てのテストが成功したことを表します。

`BankAccount` クラスのアジャイルな文書を見てみましょう。

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

BankAccount
 ^^e2^^9c^^94 Balance is initially zero
 ^^e2^^9c^^94 Balance cannot become negative
```

また、アジャイルな文書を HTML あるいはプレーンテキスト形式で作成してファイルに書き出すこともできます。この場合は、引数 `--testdox-html` あるいは `--testdox-text` を使用します。

アジャイルな文書は、プロジェクト内であなたが作成しようとしている外部パッケージについて、このように動作するであるという期待をまとめた文書にもなります。外部のパッケージを使用するときには、そのパッケージが期待通りに動作しなくなるというリスクに常にさらされています。パッケージのバージョンアップにより知らないうちに挙動が変わってしまい、あなたのコードが動作しなくなる可能性もあります。そのようなことを避けるため、「このパッケージはこのように動作するはず」ということを常にテストケースで記述しておくようにします。テストが成功すれば、期待通りに動作していることがわかります。もし動作仕様をすべてテストで記述できているのなら、外部パッケージが将来バージョンアップされたとしても何の心配もいりません。テストをクリアしたということは、システムは期待通りに動作するということだからです。

第 4 章

フィクスチャ

テストを記述する際にいちばん時間を食うのは、テストを開始するための事前設定とテスト終了後の後始末の処理を書くことです。この事前設定は、テストの**フィクスチャ** と呼ばれます。

PHPUnit **での配列操作のテスト**では、フィクスチャは`$stack` という変数に格納された配列だけでした。しかし、たいていの場合はフィクスチャはこれより複雑なものとなり、それを準備するにはかなりの量のコードが必要です。本来のテストの内容が、フィクスチャを設定するためのコードの中に埋もれてしまうことになります。この問題は、複数のテストで同じようなフィクスチャを設定する場合により顕著になります。テストフレームワークの助けがなければ、個々のテストのなかで同じような準備コードを繰り返し書くはめになってしまいます。

PHPUnit は、準備用のコードの共有をサポートしています。各テストメソッドが実行される前に、`setUp()` という名前のテンプレートメソッドが実行されます。`setUp()` は、テスト対象のオブジェクトを生成するような処理に使用します。テストメソッドの実行が終了すると、それが成功したか否かにかかわらず、`tearDown()` という名前の別のテンプレートメソッドが実行されます。`tearDown()` では、テスト対象のオブジェクトの後始末などを行います。

@depends **アノテーションを使った依存性の表現** producer-consumer の関係を使って複数のテストでフィクスチャを共有しました。これは常に要求されるというものではありませんし、常に可能だとも限りません。Example 4.1 では、フィクスチャを再利用するのではなくコードで作成する方式で `StackTest` にテストを書く方法をごらんいただきましょう。まずインスタンス変数 `$stack` を宣言し、メソッドローカル変数のかわりにこれを使うことにします。そして、`array` の作成を `setUp()` メソッドで行います。最後に、冗長なコードをテストメソッドから削除し、アサーションメソッド `assertSame()` ではメソッド変数 `$stack` のかわりに新たに導入したインスタンス変数 `$this->stack` を使うようにします。

Example 4.1　setUp() を使用して stack フィクスチャを作成する

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StackTest extends TestCase
{
    private $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));
    }
}
```

テンプレートメソッド `setUp()` および `tearDown()` は、テストケースクラスのテストメソッドごとに (そして最初にインスタンスを作成したときに) 一度ずつ実行されます。

さらに、テンプレートメソッド `setUpBeforeClass()` および `tearDownAfterClass()` が存在します。これらはそれぞれ、テストケースクラスの最初のテストメソッドの実行前とテストケースクラスの最後のテストの実行後にコールされます。

以下の例は、テストケースクラスで使用できるすべてのテンプレートメソッドを示すものです。

Example 4.2　利用可能なすべてのテンプレートメソッド

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final 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(Throwable $t): void
    {
        fwrite(STDOUT, __METHOD__ . "\n");
        throw $t;
    }
}
```

```
$ 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 tearDown() よりも setUp()

`setUp()` と `tearDown()` は理屈上では対称的になるはずですが、実際にはそうではありません。実際には、`tearDown()` を実装する必要があるのは `setUp()` で外部リソース (ファイルやソケットなど) を割り当てた場合のみです。もし `setUp()` で単に PHP オブジェクトを作成しただけの場合は、一般には `tearDown()` は必要ありません。しかし、もし `setUp()` で大量のオブジェクトを作成した場合には、それらの後始末をするために `tearDown()` で変数を `unset()` したくなることもあるでしょう。テストケースオブジェクト自体のガベージコレクションにはあまり意味がありません。

4.2 バリエーション

ふたつのテストがあって、それぞれの setup がほんの少しだけ違う場合にはどうなるでしょう? このような場合は、二種類の可能性が考えられます。

- もし `setUp()` の違いがごくわずかなものなら、その違う部分を `setUp()` からテストメソッドのほうに移動させます。
- `setUp()` の違いが大きければ、テストケースクラスを別に分ける必要があります。それぞれのクラスに

は、setup の違いを表す名前をつけます。

4.3 フィクスチャの共有

複数のテストの間でフィクスチャを共有する利点は、ほとんどありません。しかし、設計上の問題などでどうしてもフィクスチャを共有しなければならないこともあるでしょう。

複数のテスト間で共有する意味のあるフィクスチャの例として意味のあるものといえば、データベースとの接続でしょう。テストのたびに新しいデータベース接続を毎回作成するのではなく、最初にログインした状態を再利用するということです。こうすることで、テストの実行時間を短縮できます。

Example 4.3 では、テンプレートメソッド `setUpBeforeClass()` および `tearDownAfterClass()` を用いて、テストケースクラス内の最初のテストを実行する前にデータベースに接続し、最後のテストが終わってから接続を切断するようにしています。

Example 4.3　テストスイートの複数テスト間でのフィクスチャの共有

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class DatabaseTest extends TestCase
{
    private static $dbh;

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

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

このようにフィクスチャを共有することがテストの価値を下げてしまうということを、まだうまく伝え切れていないかもしれません。問題なのは、各オブジェクトが疎結合になっていないという設計なのです。複数が連携しているようなテストを作って設計上の問題から目をそらしてしまうのではなく、きちんと設計しなおした上で、スタブ (**テストダブル** を参照ください) を使用するテストを書くことをお勧めします。

4.4 グローバルな状態

singleton を使用するコードをテストするのはたいへんです。同様に、グローバル変数を使うコードのテストもまたたいへんです。一般に、テスト対象のコードはグローバル変数と密接に関連しており、グローバル変数の内容を

制御することはできません。さらに別の問題もあって、あるテストの中でグローバル変数を変更してしまうと別のテストがうまく動かなくなる可能性があります。

PHP では、グローバル変数は次のような動きをします。

- グローバル変数 `$foo = 'bar';` は、`$GLOBALS['foo'] = 'bar';` として格納される。
- `$GLOBALS` は*スーパーグローバル*変数と呼ばれる。
- スーパーグローバル変数は組み込みの変数で、すべてのスコープで常に利用できる。
- 関数やメソッドのスコープでグローバル変数 `$foo` にアクセスするには、直接 `$GLOBALS['foo']` にアクセスするか、あるいは `global $foo;` を用いて (グローバル変数を参照する) ローカル変数を作成する。

グローバル変数のほかに、クラスの静的属性もグローバル状態となります。

PHPUnit 6 より前のバージョンのデフォルトでは、PHPUnit がテストを実行する際には、グローバル変数やスーパーグローバル変数 (`$GLOBALS`, `$_ENV`, `$_POST`, `$_GET`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_REQUEST`) への変更が他のテストへの影響を及ぼさないようにしました。

PHPUnit 6 以降は、グローバル変数やスーパーグローバル変数のバックアップとリストアをデフォルトでは行わなくなりました。この機能を使いたい場合は、`--globals-backup` オプションを指定するか、XML 設定ファイルで `backupGlobals="true"` を指定します。

Note

グローバル変数やクラスの静的属性のバックアップ・リストアには `serialize()` および `unserialize()` を使用しています。

PHP 組み込みの一部のクラス、たとえば `PDO` のオブジェクトはシリアライズできないため、そのようなオブジェクトが`$GLOBALS` 配列に格納されている場合はバックアップ操作が失敗します。

@backupGlobals で説明している`@backupGlobals` アノテーションを使用すると、グローバル変数のバックアップ・リストア操作を制御することができます。あるいは、グローバル変数のリストを指定して、その変数だけはバックアップ・リストアの対象から除外することもできます。

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

    // ...
}
```

Note

`$backupGlobalsExcludeList` プロパティをたとえば `setUp()` メソッド内で設定しても効果が及びません。

@backupStaticAttributes で説明する `@backupStaticAttributes` アノテーションを使うと、宣言されたクラス内のすべての static プロパティの値をバックアップしてからテストを始め、テストが終わった後でそれらの値を復元することができます。

テストが始まる際に、宣言されたすべてのクラスについて処理を行います。テストクラス自身だけではありません。処理するのは、クラスの static プロパティだけです。関数の内部の static 変数は対象外です。

Note

`@backupStaticAttributes` の操作は、テストメソッドの前に実行されます。ただし、有効になっている場合だけです。先に実行されたテストメソッドの中で static プロパティの値が変更されており、かつそのメソッドでは`@backupStaticAttributes` が有効になっていなかった場合、バックアップ (そしてリストア) されるのは、先に実行されたメソッドで変更後の値になります。もともと宣言されていたデフォルト値ではありません。PHP は、static 変数が宣言された当時のデフォルト値を、どこにも記録していないのです。

これは、テストの内部で新しく読み込んだ (宣言した) クラスの static プロパティについても同様です。この場合も、もともと宣言されていたデフォルト値を、テストの後に復元することはできません。デフォルト値が残っていないからです。テスト内で設定された値が、それ以降のテストに持ち越されます。

ユニットテストでは、テスト対象の static プロパティの値は、`setUp()` で明示的にリセットしておくことを推奨します (そして、`tearDown()` でもリセットしておけば、それ以降のテストに影響を及ぼすこともなくなります)。

static 属性のリストを渡せば、保存と復元の対象からそれらを除外することもできます。

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

    // ...
}
```

Note

`$backupStaticAttributesBlacklist` プロパティをたとえば `setUp()` メソッド内で設定しても効果が及びません。

第 5 章

テストの構成

PHPUnit の目指すところのひとつに「自由に組み合わせられる」ということがあります。つまり、例えば「そのプロジェクトのすべてのテストを実行する」「プロジェクトの中のある部品を構成するすべてのクラスについて、すべてのテストを実行する」「特定のひとつのクラスのテストのみを実行する」など、数や組み合わせにとらわれずに好きなテストを一緒に実行できるということです。

PHPUnit では、さまざまな方法でテストを組み合わせてテストスイートにまとめることができます。本章では、その中でもよく使われる手法を説明します。

5.1 ファイルシステムを用いたテストスイートの構成

おそらく、テストスイートをとりまとめるもっとも簡単な方法はすべてのテストケースのソースファイルを一つのテストディレクトリにまとめることでしょう。PHPUnit はテストディレクトリを再帰的に探索し、テストを自動的に見つけて実行します。

sebastianbergmann/money ライブラリのテストスイートを見てみましょう。このプロジェクトのディレクトリ構成を見ると、テストケースクラスが `tests` ディレクトリにまとめられていることがわかります。その中のディレクトリの構造は、テスト対象のシステム (SUT) がある `src` ディレクトリ以下の構造と同じになっています。

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

PHPUnit のコマンドラインテストランナーにテストディレクトリの場所を指示することで、このライブラリのすべてのテストを実行できます。

```
$ 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)
```

Note

PHPUnit のコマンドラインテストランナーでディレクトリを指定すると、その中の `*Test.php` ファイルを見つけて実行します。

`tests/CurrencyTest.php` にあるテストケースクラス `CurrencyTest` で宣言されているテストだけを実行するには、次のコマンドを実行します。

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

........

Time: 280 ms, Memory: 2.75Mb

OK (8 tests, 8 assertions)
```

実行したいテストをより細かく指示するには`--filter` オプションを使います。

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

..

Time: 167 ms, Memory: 3.00Mb

OK (2 test, 2 assertions)
```

Note

この方式の欠点は、テストの実行順を制御できないことです。そのため、テストの依存性に関する問題を引き起こすことがあります。**テストの依存性**を参照ください。次の節では、テストの実行順序を XML 設定ファイルで明示的に指定する方法を説明します。

5.2 XML 設定ファイルを用いたテストスイートの構成

PHPUnit の XML 設定ファイル (*XML 設定ファイル*) を使ってテストスイートを構成することもできます。Example 5.1 に、最小限の phpunit.xml ファイルを示します。これは、tests ディレクトリを再帰的に探索して*Test.php というファイルにある*Test クラスをすべて追加する設定です。

Example 5.1　XML 設定ファイルを用いたテストスイートの構成

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

特定のテストスイートを実行したい場合は、--testsuite オプションを利用します。

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


..
Time: 167 ms, Memory: 3.00Mb


OK (2 test, 2 assertions)
```

--configuration が設定**されていない** 場合は、現在の作業ディレクトリから phpunit.xml あるいは phpunit.xml.dist を (この順に) 探し、見つかった場合はそれを自動的に読み込みます。

また以下のように、テストの実行順序を XML 設定ファイルで明示的に指定することもできます。

Example 5.2　XML 設定ファイルを用いたテストスイートの構成

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

第 6 章

リスクを伴うテスト

PHPUnit は、テストを実行する際に、以下のような追加のチェックをすることができます。

6.1 無意味なテスト

PHPUnit はデフォルトで、何も確かめていないテストを検出します。このチェックを無効にするには、コマンドラインオプション`--dont-report-useless-tests` を使うか、あるいは PHPUnit の XML 設定ファイルで`beStrictAboutTestsThatDoNotTestAnything="false"` を設定します。

何もアサーションを実行していないテストは、このチェックを有効にしておくと、危険であるとマークされます。モックオブジェクトでの例外や、@expectedException などのアノテーションは、アサーションとみなします。

6.2 意図せぬうちにカバーされているコード

PHPUnit は、意図せずカバーされているコードを検出することができます。このチェックを有効にするには、コマンドラインオプション`--strict-coverage` を使うか、あるいは PHPUnit の XML 設定ファイルで`beStrictAboutCoversAnnotation="true"` を設定します。

@covers アノテーションつきのテストが、@covers や @uses に記されていないコードを実行している場合に、このチェックを有効にしておくと、危険であるとマークされます。

6.3 テストの実行時の出力

PHPUnit は、テストの最中の出力を検出することができます。このチェックを有効にするには、コマンドラインオプション`--disallow-test-output` を使うか、あるいは PHPUnit の XML 設定ファイルで`beStrictAboutOutputDuringTests="true"` を設定します。

テストコードあるいは被テストコードの中で、print などで何かを出力している場合に、このチェックを有効にしておくと、危険であるとマークされます。

6.4 テストの実行時のタイムアウト

PHP_Invoker パッケージがインストールされており、かつ pcntl 拡張モジュールが利用可能な場合は、テストの実行時間に制限を設けることができます。この時間制限を有効にするには、コマンドラインオプション--enforce-time-limit を使うか、あるいは PHPUnit の XML 設定ファイルで beStrictAboutTestSize="true" を設定します。

@large とマークされたテストは、実行時間が 60 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの timeoutForLargeTests 属性で変更できます。

@medium とマークされたテストは、実行時間が 10 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの timeoutForMediumTests 属性で変更できます。

@medium とも @large ともマークされていないテストは、@small とマークされたものとみなします。このテストは、実行時間が 1 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの timeoutForSmallTests 属性で変更できます。

6.5 グローバルな状態の変更

PHPUnit は、グローバルな状態を変更するテストを厳格にチェックすることができます。このチェックを有効にするには、コマンドラインで --strict-global-state オプションを指定するか、PHPUnit の XML 設定ファイルで beStrictAboutChangesToGlobalState="true" を指定します。

第 7 章

不完全なテスト・テストの省略

7.1 不完全なテスト

新しいテストケースクラスを作成する際には、これから書くべきテストの内容をはっきりさせるために、まず最初は以下のような空のテストメソッドを書きたくなることでしょう。

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

しかし、PHPUnit フレームワークでは空のメソッドを「成功した」と判断してしまうという問題があります。このような解釈ミスがあると、テスト結果のレポートが無意味になってしまいます。そのテストがほんとうに成功したのか、それともまだテストが実装されていないのかが判断できないからです。実装していないテストメソッドの中で `$this->fail()` をコールするようにしたところで事態は何も変わりません。こうすると、テストが「失敗した」と判断されてしまいます。これは未実装のテストが「成功」と判断されてしまうのと同じくらいまずいことです (訳注: レポートを見ても、そのテストがほんとうに失敗したのか、まだ実装されていないだけなのかがわかりません)。

テストの成功を青信号、失敗を赤信号と考えるなら、テストが未完成あるいは未実装であることを表すための黄信号が必要です。そのような場合に使用するインターフェイスが `PHPUnit\Framework\IncompleteTest` で、これは未完成あるいは未実装のテストメソッドで発生する例外を表すものです。このインターフェイスの標準的な実装が `PHPUnit\Framework\IncompleteTestError` です。

Example 7.1 では `SampleTest` というテストケースクラスを定義しています。便利なメソッド `markTestIncomplete()` (これは、自動的に `PHPUnit\Framework\IncompleteTestError` を発生させます) をテストメソッド内でコールすることで、このメソッドがまだ完成していないことをはっきりさせます。

Example 7.1　テストに未完成の印をつける

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

```

```
class SampleTest extends TestCase
{
    public function testSomething()
    {
        // オプション: お望みなら、ここで何かのテストをしてください。
        $this->assertTrue(true, 'これは動いているはずです。');

        // ここで処理を止め、テストが未完成であるという印をつけます。
        $this->markTestIncomplete(
          'このテストは、まだ実装されていません。'
        );
    }
}
?>
```

未完成のテストは、PHPUnit のコマンドライン版テストランナーでは以下のように I で表されます。

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

I

Time: 0 seconds, Memory: 3.95Mb

There was 1 incomplete test:

1) SampleTest::testSomething
このテストは、まだ実装されていません。

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

Table 7.1 に、テストを未完成扱いにするための API を示します。

Table7.1　未完成のテスト用の API

メソッド	意味
`void markTestIncomplete()`	現在のテストを未完成扱いにします。
`void markTestIncomplete(string $message)`	現在のテストを未完成扱いにします。それを説明する文字列として `$message` を使用します。

7.2 テストの省略

すべてのテストがあらゆる環境で実行できるわけではありません。考えてみましょう。たとえば、データベースの抽象化レイヤーを使用しており、それがさまざまなドライバを使用してさまざまなデータベースシステムをサポートしているとします。MySQL ドライバのテストができるのは、当然 MySQL サーバが使用できる環境だけです。

Example 7.2 に示すテストケースクラス `DatabaseTest` には、テストメソッド `testConnection()` が含まれています。このクラスのテンプレートメソッド `setUp()` では、MySQLi 拡張モジュールが使用可能かを調べたうえで、もし使用できない場合は `markTestSkipped()` メソッドでテストを省略するようにしています。

Example 7.2　テストを省略する

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

class DatabaseTest extends TestCase
{
    protected function setUp(): void
    {
        if (!extension_loaded('mysqli')) {
            $this->markTestSkipped(
              'MySQLi 拡張モジュールが使用できません。'
            );
        }
    }

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

飛ばされたテストは、PHPUnit のコマンドライン版テストランナーでは以下のように `S` で表されます。

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

S

Time: 0 seconds, Memory: 3.95Mb

There was 1 skipped test:

1) DatabaseTest::testConnection
MySQLi 拡張モジュールが使用できません。

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

```
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Skipped: 1.
```

Table 7.2 に、テストを省略するための API を示します。

Table7.2　テストを省略するための API

メソッド	意味
`void markTestSkipped()`	現在のテストを省略扱いにします。
`void markTestSkipped(string $message)`	現在のテストを省略扱いにします。それを説明する文字列として `$message` を使用します。

7.3 @requires によるテストのスキップ

ここまでに示したメソッドに加えて、`@requires` アノテーションを使って共通の事前条件を記述することもできます。

Table7.3　@requires の例用例

型	取り得る値	例	別の例
`PHP`	PHP のバージョン	@requires PHP 5.3.3	@requires PHP 7.1-dev
`PHPUnit`	PHPUnit のバージョン	@requires PHPUnit 3.6.3	@requires PHPUnit 4.6
`OS`	PHP_OS にマッチする正規表現	@requires OS Linux	@requires OS WIN32|WINNT
`OSFAMILY`	任意の OS ファミリー	@requires OSFAMILY Solaris	@requires OSFAMILY Windows
`function`	function_exists に渡せるパラメータ	@requires function imap_open	@requires function ReflectionMethod::setAccessible
`extension`	拡張モジュール名 (オプションでバージョンも指定できる)	@requires extension mysqli	@requires extension redis 2.2.0

Example 7.3　@requires を使ったテストケースのスキップ

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

/**
 * @requires extension mysqli
 */
class DatabaseTest extends TestCase
{
```

```
    /**
     * @requires PHP 5.3
     */
    public function testConnection()
    {
        // このテストには mysqli 拡張モジュールと PHP 5.3 以降が必須です
    }

    // ... その他のすべてのテストには mysqli 拡張モジュールが必須です
}
?>
```

特定のバージョンの PHP でしか使えない構文を利用する場合は、**テストスイート**にあるように XML 設定ファイルでのバージョン依存のインクルードを検討しましょう。

第 8 章

テストダブル

Gerard Meszaros は、テストダブルの概念を *Meszaros2007* でこのように述べています。

> *Gerard Meszaros*:
>
> Sometimes it is just plain hard to test the system under test (SUT) because it depends on other components that cannot be used in the test environment. This could be because they aren ' t available, they will not return the results needed for the test or because executing them would have undesirable side effects. In other cases, our test strategy requires us to have more control or visibility of the internal behavior of the SUT.
>
> - テスト対象のシステム (SUT: system under test) をテストすることは、時に非常に困難なこととなります。というのも、システムが他のコンポーネントに依存しており、そのコンポーネントをテスト環境で利用できないことがあるからです。そもそも使用不可能であったりテストで必要な結果を返さなかったり、あるいは好ましくない副作用があったりといったことです。それ以外の場合も、テスト環境の内部的な振る舞いをきちんと制御して目に見えるようにしておくことが必要です。
>
> When we are writing a test in which we cannot (or chose not to) use a real depended-on component (DOC), we can replace it with a Test Double. The Test Double doesn ' t have to behave exactly like the real DOC; it merely has to provide the same API as the real one so that the SUT thinks it is the real one!
>
> - 実際に依存するコンポーネント (DOC: depended-on component) を使わないテストを書く場合は、それをテストダブルで置き換えることができます。テストダブルは、必ずしも実際の DOC とまったく同様に動作する必要はありません。単に実際のものと同じ API を提供し、SUT に「これは本物だ!」と思わせるだけでいいのです。

PHPUnit の `createStub($type)` メソッドや `getMockBuilder($type)` メソッドを使うと、指定した元インターフェイス (あるいは元クラス) のテストダブルとして振る舞うオブジェクトを自動的に生成することができます。このテストダブルオブジェクトは、元のオブジェクトを要するすべての場面で使うことができます。

`createStub($type)` 、および `createMock($type)` メソッドは、指定した型 (インターフェイスやクラス) のテストダブルオブジェクトをその場で返します。テストダブルの作成は、デフォルトではベストプラクティスに沿って行われます (元クラスの `__construct()` や `__clone()` は実行しません)。また、テストダブルのメソッ

ドに渡された引数はクローンされません。デフォルトと異なる挙動を求める場合は、`getMockBuilder($type)` メソッドを用いてテストダブルの生成処理をカスタマイズする必要があります。

デフォルトでは、元クラスのすべてのメソッドが置き換えられて、(元のメソッドは呼び出さずに) 単に `null` を返すダミー実装になります。たとえば `will($this->returnValue())` メソッドを使うと、ダミー実装がコールされたときに値を返すよう設定することができます。

制限：final、private および static メソッド

`final`, `private` および `static` メソッドのスタブやモックは作れないことに注意しましょう。PHPUnit のテストダブル機能ではこれらを無視し、元のメソッドの振る舞いをそのまま維持します。ただし `static` メソッドは例外で、これは`\PHPUnit\Framework\MockObject\BadMethodCallException` をスローするメソッドに置き換えられます。

8.1 スタブ

実際のオブジェクトを置き換えて、設定した何らかの値を (オプションで) 返すようなテストダブルのことを**スタブ** といいます。**スタブ** を使うと、「SUT が依存している実際のコンポーネントを置き換え、SUT の入力を間接的にコントロールできるようにすることができます。これにより、SUT が他の何者も実行しないことを強制させることができます。」

Example 8.2 に、スタブメソッドの作成と返り値の設定の方法を示します。まず、`PHPUnit\Framework\TestCase` クラスの `createStub()` メソッドを用いて `SomeClass` オブジェクトのスタブを作成します (Example 8.1)。次に、PHPUnit が提供する、いわゆる Fluent Interface (流れるようなインターフェイス) を用いてスタブの振る舞いを指定します。簡単に言うと、いくつもの一時オブジェクトを作成して、それらを連結するといった操作は必要ないということです。そのかわりに、例にあるようにメソッドの呼び出しを連結します。このほうが、より読みやすく“流れるような”コードとなります。

Example 8.1　スタブを作りたいクラス

```
<?php declare(strict_types=1);
class SomeClass
{
    public function doSomething()
    {
        // なにかをします
    }
}
```

Example 8.2　メソッドに固定値を返させるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // スタブの設定を行います
        $stub->method('doSomething')
             ->willReturn('foo');

        // $stub->doSomething() をコールすると
        // 'foo' を返すようになります
        $this->assertSame('foo', $stub->doSomething());
    }
}
```

制限：“ method ”という名前のメソッド

この例がきちんと動作するのは、元のクラスで“ method ”という名前のメソッドが宣言されていない場合だけです。

元のクラスで“ method ”という名前のメソッドが宣言されている場合は、`$stub->expects($this->any())->method('doSomething')->willReturn('foo');` としなければいけません。

舞台裏では、`createStub()` メソッドが使われたときに PHPUnit が自動的に、求める振る舞いを実装した新たな PHP のクラスを生成しています。

なお、`createStub()` は、メソッドの戻り値の型に基づいて自動的に、そして再帰的に戻り値をスタブ化します。以下のような例を考えてみましょう。

Example 8.3　返り値について型宣言されたメソッド

```
<?php declare(strict_types=1);
class C
{
    public function m(): D
    {
        // Do something.
    }
```

```
}
```

上の例のように、 C::m() は返り値として D のオブジェクトを返すことが宣言されています。 C のテストダブルを作成し、返り値を willReturn() メソッドなどで設定しなかった場合、 C のテストダブルで m() のメソッドを呼び出すと、 PHPUnit は自動的に D のテストダブルを作成して返します。同様に m の型定義がスカラー型であった場合は、 0 (int の場合)、 0.0 (float の場合)、 [] (**配列** の場合) が生成され返されます。

Example 8.4 に例を示します。これは、モックビルダーの流れるようなインターフェイスを使って、テストダブルの作成方法を設定するものです。このテストダブルで使っている設定は、createStub() がデフォルトで使用するベストプラクティスと同じです。

Example 8.4　モックビルダー API を使った、生成されるテストダブルクラスの変更

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMockBuilder(SomeClass::class)
                     ->disableOriginalConstructor()
                     ->disableOriginalClone()
                     ->disableArgumentCloning()
                     ->disallowMockingUnknownTypes()
                     ->getMock();

        // スタブの設定を行います
        $stub->method('doSomething')
             ->willReturn('foo');

        // $stub->doSomething() をコールすると
        // 'foo' を返すようになります
        $this->assertSame('foo', $stub->doSomething());
    }
}
```

ここまでの例では、willReturn($value) を使ってシンプルな値を返していました。この構文は、will($this->returnValue($value)) と同じ意味です。この長い構文での検証を使うと、より複雑な動きをするスタブも作れます。

時には、メソッドをコールした際の引数のひとつを (そのまま) スタブメソッドコールの返り値としたいこともあるでしょう。Example 8.5 は、returnValue() のかわりに returnArgument() を用いてこれを実現する例です。

Example 8.5　メソッドに引数のひとつを返させるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testReturnArgumentStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // スタブの設定を行います
        $stub->method('doSomething')
             ->will($this->returnArgument(0));

        // $stub->doSomething('foo') は 'foo' を返します
        $this->assertSame('foo', $stub->doSomething('foo'));

        // $stub->doSomething('bar') は 'bar' を返します
        $this->assertSame('bar', $stub->doSomething('bar'));
    }
}
```

流れるようなインターフェイスをテストするときには、スタブメソッドがオブジェクト自身への参照を返すようにできると便利です。Example 8.6 は、`returnSelf()` を使ってこれを実現する例です。

Example 8.6　スタブオブジェクトへの参照を返すメソッドのスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testReturnSelf(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // スタブの設定を行います
        $stub->method('doSomething')
             ->will($this->returnSelf());

        // $stub->doSomething() は $stub を返します
        $this->assertSame($stub, $stub->doSomething());
    }
}
```

スタブメソッドをコールした結果として、定義済みの引数リストにあわせて異なる値を返さなければならないこと

もあるでしょう。returnValueMap() を使えば、マップを作って引数と関連付け、それを返り値に対応させることができます。Example 8.7 を参照ください。

Example 8.7　メソッドにマップからの値を返させるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testReturnValueMapStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // 値を返すための、引数のマップを作製します
        $map = [
            ['a', 'b', 'c', 'd'],
            ['e', 'f', 'g', 'h']
        ];

        // スタブの設定を行います
        $stub->method('doSomething')
             ->will($this->returnValueMap($map));

        // $stub->doSomething() は、渡した引数に応じて異なる値を返します
        $this->assertSame('d', $stub->doSomething('a', 'b', 'c'));
        $this->assertSame('h', $stub->doSomething('e', 'f', 'g'));
    }
}
```

スタブメソッドをコールした結果として固定値 (returnValue() を参照ください) や (不変の) 引数 (returnArgument() を参照ください) ではなく計算した値を返したい場合は、returnCallback() を使用します。これは、スタブメソッドからコールバック関数やメソッドの結果を返させます。Example 8.8 を参照ください。

Example 8.8　メソッドにコールバックからの値を返させるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testReturnCallbackStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // スタブの設定を行います
```

```
        $stub->method('doSomething')
             ->will($this->returnCallback('str_rot13'));

        // $stub->doSomething($argument) は str_rot13($argument) を返します
        $this->assertSame('fbzrguvat', $stub->doSomething('something'));
    }
}
```

コールバックメソッドを設定するよりももう少しシンプルな方法として、希望する返り値のリストを指定することもできます。この場合に使うのは onConsecutiveCalls() メソッドです。Example 8.9 の例を参照ください。

Example 8.9　メソッドに、リストで指定した値をその順で返させるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testOnConsecutiveCallsStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);

        // スタブの設定を行います
        $stub->method('doSomething')
             ->will($this->onConsecutiveCalls(2, 3, 5, 7));

        // $stub->doSomething() は毎回異なる値を返します
        $this->assertSame(2, $stub->doSomething());
        $this->assertSame(3, $stub->doSomething());
        $this->assertSame(5, $stub->doSomething());
    }
}
```

値を返すのではなく、スタブメソッドで例外を発生させることもできます。Example 8.10 に、throwException() でこれを行う方法を示します。

Example 8.10　メソッドに例外をスローさせるスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class StubTest extends TestCase
{
    public function testThrowExceptionStub(): void
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->createStub(SomeClass::class);
```

```
        // スタブの設定を行います
        $stub->method('doSomething')
             ->will($this->throwException(new Exception));

        // $stub->doSomething() は例外をスローします
        $stub->doSomething();
    }
}
```

また、スタブを使用することで、よりよい設計を行うことができるようにもなります。あちこちで使用されているリソースを単一の窓口 (faÃğade : ファサード) 経由でアクセスするようにすることで、それをスタブに置き換えられるようになります。例えば、データベースへのアクセスのコードをそこらじゅうにちりばめるのではなく、その代わりに `IDatabase` インターフェイスを実装した単一の `Database` オブジェクトを使用するようにします。すると、`IDatabase` を実装したスタブを作成することで、それをテストに使用できるようになるのです。同時に、テストを行う際にスタブデータベースを使用するか本物のデータベースを使用するかを選択できるようになります。つまり開発時にはローカル環境でテストし、統合テスト時には実際のデータベースでテストするといったことができるようになるのです。

スタブ化しなければならない機能は、たいてい同一オブジェクト内で密結合しています。この機能ををひとつの結合したインターフェイスにまとめることで、システムのそれ以外の部分との結合を緩やかにすることができます。

8.2 モックオブジェクト

実際のオブジェクトを置き換えて、(メソッドがコールされたことなどの) 期待する内容を検証するテストダブルのことを**モック** といいます。

モックオブジェクト は " SUT の間接的な出力の内容を検証するために使用する観測地点です。一般的に、モックオブジェクトにはテスト用スタブの機能も含まれます。まだテストに失敗していない場合に、間接的な出力の検証用の値を SUT に返す機能です。したがって、モックオブジェクトとはテスト用スタブにアサーション機能を足しただけのものとは異なります。それ以外の用途にも使うことができます " (Gerard Meszaros)。

制限：期待値の自動検証

そのテストのスコープ内で生成されたモックオブジェクトだけが、PHPUnit による自動検証の対象となります。たとえば、データプロバイダなどで生成されたモックオブジェクトや`@depends` アノテーションで注入されたオブジェクトについては、PHPUnit では検証しません。

ひとつ例を示します。ここでは、別のオブジェクトを観察しているあるオブジェクトの特定のメソッド (この例では `update()`) が正しくコールされたかどうかを調べるものとします。Example 8.11 は、テスト対象のシステム (SUT) の一部である `Subject` クラスと `Observer` クラスのコードです。

Example 8.11　テスト対象のシステム (SUT) の一部である Subject クラスと Observer クラス

```
<?php declare(strict_types=1);
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()
    {
        // なにかをします
        // ...

        // なにかしたということをオブザーバに通知します
        $this->notify('something');
    }

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

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

    // その他のメソッド
```

```
}

class Observer
{
    public function update($argument)
    {
        // なにかをします
    }

    public function reportError($errorCode, $errorMessage, Subject $subject)
    {
        // なにかをします
    }

    // その他のメソッド
}
```

Example 8.12 では、モックオブジェクトを作成して Subject オブジェクトと Observer オブジェクトの対話をテストする方法を説明します。

まず PHPUnit\Framework\TestCase クラスの createMock() メソッドを使用して Observer のモックオブジェクトを作成します。

あるメソッドがコールされたのかどうか、そしてどんな引数を渡してコールされたのかを検証したいので、expects() メソッドと with() メソッドを利用します。これらを使って、このやりとりがどのように行われるのかを指定します。

Example 8.12　あるメソッドが、指定した引数で一度だけコールされることを確かめるテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class SubjectTest extends TestCase
{
    public function testObserversAreUpdated(): void
    {
        // Observer クラスのモックを作成します。
        // update() メソッドのみのモックです。
        $observer = $this->createMock(Observer::class);

        // update() メソッドが一度だけコールされ、その際の
        // パラメータは文字列 'something' となる、
        // ということを期待しています。
        $observer->expects($this->once())
                 ->method('update')
                 ->with($this->equalTo('something'));

        // Subject オブジェクトを作成し、Observer オブジェクトの
```

```
        // モックをアタッチします。
        $subject = new Subject('My subject');
        $subject->attach($observer);

        // $subject オブジェクトの doSomething() メソッドをコールします。
        // これは、Observer オブジェクトのモックの update() メソッドを、
        // 文字列 'something' を引数としてコールすることを期待されています。
        $subject->doSomething();
    }
}
```

with() メソッドには任意の数の引数を渡すことができます。これは、モック対象のメソッドの引数の数に対応します。メソッドの引数に対して、単なるマッチだけでなくより高度な制約を指定することもできます。

Example 8.13　メソッドが引数つきでコールされることを、さまざまな制約の下でテストする例

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class SubjectTest extends TestCase
{
    public function testErrorReported(): void
    {
        // Observer クラスのモックを作成します。
        // reportError() メソッドをモックします。
        $observer = $this->createMock(Observer::class);

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

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

        // doSomethingBad() メソッドは、
        // reportError() メソッドを通じてオブザーバにエラーを報告しなければなりません。
        $subject->doSomethingBad();
    }
}
```

withConsecutive() メソッドには、テスト対象の呼び出しにあわせて、引数の配列を好きなだけ渡せます。個々の配列は制約のリストです。with() と同様に、これがモック対象メソッドのそれぞれの引数に対応します。

Example 8.14　あるメソッドが、指定した引数つきで 2 回呼び出されることを確かめるテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

        $mock->expects($this->exactly(2))
             ->method('set')
             ->withConsecutive(
                 [$this->equalTo('foo'), $this->greaterThan(0)],
                 [$this->equalTo('bar'), $this->greaterThan(0)]
             );

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

`callback()` 制約を使えば、より複雑な引数の検証ができます。この制約は、PHP のコールバックを引数として受け取ります。このコールバックは、検証したい引数を受け取って、検証を通過した場合に `true`、それ以外の場合に `false` を返します。

Example 8.15　より複雑な引数の検証

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class SubjectTest extends TestCase
{
    public function testErrorReported(): void
    {
        // Observer クラスのモックを作成します。
        // reportError() メソッドをモックします。
        $observer = $this->createMock(Observer::class);
        $observer->expects($this->once())
                 ->method('reportError')
                 ->with(
                     $this->greaterThan(0),
                     $this->stringContains('Something'),
                     $this->callback(function($subject)
                     {
                         return is_callable([$subject, 'getName']) &&
```

```
                                $subject->getName() == 'My subject';
                    }
                ));

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

        // doSomethingBad() メソッドは、
        // reportError() メソッドを通じてオブザーバにエラーを報告しなければなりません。
        $subject->doSomethingBad();
    }
}
```

Example 8.16　メソッドが一度だけ呼ばれ、同じオブジェクトが渡されたことを確かめるテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

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

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

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

Example 8.17　パラメータのクローンの有効にしたモックオブジェクトの作成

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

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

```
                    ->getMock();

        // これでモックがパラメータをクローンするようになり、
        // identicalTo 制約は失敗します
    }
}
```

制約はメソッドの引数に適用できる制約、そして Table 8.1 は起動回数を指定するために使える matcher です。

Table8.1 Matchers

Matcher	意味
PHPUnit\Framework\MockObject\Matcher\AnyInvokedCount any()	評価対象のメソッドがゼロ回以上実行された際にマッチするオブジェクトを返します。
PHPUnit\Framework\MockObject\Matcher\InvokedCount never()	評価対象のメソッドが実行されなかった際にマッチするオブジェクトを返します。
PHPUnit\Framework\MockObject\Matcher\InvokedAtLeastOnce atLeastOnce()	評価対象のメソッドが最低一回以上実行された際にマッチするオブジェクトを返します。
PHPUnit\Framework\MockObject\Matcher\InvokedCount once()	評価対象のメソッドが一度だけ実行された際にマッチするオブジェクトを返します。
PHPUnit\Framework\MockObject\Matcher\InvokedCount exactly(int $count)	評価対象のメソッドが指定した回数だけ実行された際にマッチするオブジェクトを返します。
PHPUnit\Framework\MockObject\Matcher\InvokedAtIndex at(int $index)	評価対象のメソッドが $index 回目に実行された際にマッチするオブジェクトを返します。

Note

at() マッチャーのパラメータ $index は、指定したモックオブジェクトでの **すべてのメソッドの実行**の、ゼロからはじまるインデックスを参照します。このマッチャーを使うときには注意しましょう。テストが実装の詳細とあまりにも密結合になり、脆いテストになってしまう可能性があるからです。

最初に説明したとおり、 createStub() および createMock() メソッドが用いるデフォルトのテストダブル生成方法がニーズを満たさない場合は、getMockBuilder($type) メソッドを使えば生成方法をカスタマイズできます。モックビルダーが提供するメソッドの一覧は、次のとおりです。

- setMethods(array $methods) をモックビルダーオブジェクト上でコールすると、テストダブルで置き換えるメソッドを指定することができます。その他のメソッドの挙動は変更しません。setMethods(NULL) とすると、どのメソッドも置き換えません。
- setMethodsExcept(array $methods) をモックビルダーオブジェクト上でコールすると、テストダブルで置き換えないメソッドを指定することができます。その他のすべての pubilic メソッドは置き換えられます。このメソッドは setMethods() の逆の働きをします。

- `setConstructorArgs(array $args)` をコールしてパラメータの配列を渡すと、それを元クラスのコンストラクタに渡すことができます (デフォルトのダミー実装では、コンストラクタは置き換えません)。
- `setMockClassName($name)` を使うと、生成されるテストダブルクラスのクラス名を指定することができます。
- `disableOriginalConstructor()` を使うと、元クラスのコンストラクタを無効にすることができます。
- `disableOriginalClone()` を使うと、元クラスのクローンコンストラクタを無効にすることができます。
- `disableAutoload()` を使うと、テストダブルクラスを生成するときに `__autoload()` を無効にすることができます。

8.3 トレイトと抽象クラスのモック

`getMockForTrait()` メソッドは、指定したトレイトを使ったモックオブジェクトを返します。そのトレイトのすべての抽象メソッドがモックの対象となります。これを使えば、トレイトの具象メソッドをテストすることができます。

Example 8.18 トレイトの具象メソッドのテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public abstract function abstractMethod();
}

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

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

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

```
    }
}
```

getMockForAbstractClass() メソッドは、抽象クラスのモックオブジェクトを返します。そのクラスのすべての抽象メソッドがモックの対象となります。これを使えば、抽象クラスにある具象メソッドをテストすることができます。

Example 8.19　抽象クラスの具象メソッドのテスト

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

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

    public abstract function abstractMethod();
}

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

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

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

8.4 ウェブサービスのスタブおよびモック

ウェブサービスとのやりとりを行うアプリケーションを、実際にウェブサービスとやりとりすることなくテストしたくなることもあるでしょう。ウェブサービスのスタブやモックを作成するために getMockFromWsdl() メソッドが用意されており、これは getMock() (上を参照ください) とほぼ同様に使うことができます。唯一の違いは、getMockFromWsdl() が返すスタブやモックが WSDL のウェブサービス記述にもとづくものであるのに対して getMock() が返すスタブやモックが PHP のクラスやインターフェイスにもとづくものであるという点です。

Example 8.20 は、getMockFromWsdl() を使って GoogleSearch.wsdl に記述されたウェブサービスのスタブを作る例です。

Example 8.20　ウェブサービスのスタブ

```
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class GoogleTest extends TestCase
{
    public function testSearch(): void
    {
        $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() はスタブが用意した結果を返し、
```

```
         * ウェブサービスの doGoogleSearch() が呼び出されることはありません
         */
        $this->assertEquals(
          $result,
          $googleSearch->doGoogleSearch(
            '00000000000000000000000000000000',
            'PHPUnit',
            0,
            1,
            false,
            '',
            false,
            '',
            '',
            ''
          )
        );
    }
}
```

第 9 章

コードカバレッジ解析

> *Wikipedia*:
>
> In computer science, code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. A program with high code coverage has been more thoroughly tested and has a lower chance of containing software bugs than a program with low code coverage.

この章では、PHPUnit のコードカバレッジ機能について学びます。これは、テストを実行したときに、実装コードのどの部分が実行されたかを調べるものです。PHPUnit のコードカバレッジ解析では php-code-coverage コンポーネントを使っています。このコンポーネントは、Xdebug 拡張モジュールが提供するステートメントカバレッジ機能を利用しています。

Note

Xdebug は PHPUnit 本体には組み込まれていません。テストを実行したときに no code coverage driver is available という notice が出る場合は、Xdebug がインストールされていないかあるいはうまく設定できていないのでしょう。PHPUnit のコードカバレッジ機能を使う前に、まずは Xdebug のインストールガイドを読んでみましょう。

php-code-coverage は phpdbg もコードカバレッジデータのソースとしてサポートしています。

PHPUnit は、HTML ベースのコードカバレッジレポートを生成するだけでなく、XML ベースのログファイルにコードカバレッジ情報を出力することもできます。Clover、Crap4J、PHPUnit など、さまざまな形式に対応しています。また、コードカバレッジ情報をテキスト形式で出力 (そして、標準出力に表示) したり、PHP のコードとして出力して後処理をしたりすることもできます。

コードカバレッジ機能を制御するためのコマンドラインスイッチの一覧は、**コマンドラインのテストランナー** を参照ください。また、設定項目については **ログ出力** を参照ください。

9.1 コードカバレッジの指標

コードカバレッジを計測するための指標には、さまざまなものがあります。

Line Coverage

ラインカバレッジ は、実行可能な行が実行されたかどうかを計測します。

Function and Method Coverage

関数・メソッドカバレッジ は、関数やメソッドが実行されたかどうかを計測します。php-code-coverage は、その関数やメソッド内の実行可能な行がすべて実行された場合にのみ、その関数やメソッドが実行されたとみなします。

クラス・トレイトカバレッジ

クラス・トレイトカバレッジ は、クラスやトレイトがカバーされたかどうかを計測します。php-code-coverage は、クラスやトレイト内のすべてのメソッドがカバーされている場合にのみ、そのクラスやトレイトがカバーされたとみなします。

Opcode Coverage

オペコードカバレッジ は、関数やメソッドのオペコードが、テストスイートの実行中に実行されたかどうかを計測します。通常は、1 行のコードをコンパイルすると、複数のオペコードになります。ラインカバレッジは、複数のオペコードのうち少なくともひとつが実行された時点で、その行が実行されたとみなします。

Branch Coverage

ブランチカバレッジ は、テストスイートの実行中に、制御構造内の boolean 式が `true` あるいは `false` のどちらかとして評価されたかどうかを計測します。

Path Coverage

パスカバレッジ は、テストスイートの実行中に、関数やメソッド内で取りうる実行パスが網羅されたかどうかを計測します。実行パスとは、関数やメソッドに入ってから出るまでの間のルート内での分岐のことです。

Change Risk Anti-Patterns (CRAP) Index

Change Risk Anti-Patterns (CRAP) **インデックス**とは、循環的複雑度と、あるコード単位のコードカバレッジに基づいて算出される指標です。複雑度が低く、適切なテストカバレッジが達成されているコードは、CRAP インデックスの値が低くなります。CRAP インデックスを下げるには、テストを書くか、あるいはリファクタリングでコードの複雑性を下げます。

Note

オペコードカバレッジ、**ブランチカバレッジ**、**パスカバレッジ** については、php-code-coverage ではまだサポートしていません。

9.2 ファイルのホワイトリスト

ホワイトリスト を設定して、PHPUnit に対してどのソースコードファイルをコードカバレッジレポートに含めるかを指定する必要があります。ホワイトリストの設定には、コマンドラインオプションの `--whitelist` を使うか、あるいは設定ファイルを使います (**コードカバレッジ対象のファイルのホワイトリスト** を参照ください)。

設定項目 `addUncoveredFilesFromWhitelist` および `processUncoveredFilesFromWhitelist` で、ホワイトリストの使いかたを設定できます。

- `addUncoveredFilesFromWhitelist="false"` と指定すると、ホワイトリストの追加したファイルのうち、実行される行が少なくとも一行以上あるファイルだけをコードカバレッジレポートに含めます。
- `addUncoveredFilesFromWhitelist="true"` (デフォルト) と指定すると、実行されるコードが一行もないファイルを含めてホワイトリスト上のすべてのファイルをコードカバレッジレポートに含めます。
- `processUncoveredFilesFromWhitelist="false"` (デフォルト) と指定すると、実行される行が一行もないホワイトリスト上のファイルも (`addUncoveredFilesFromWhitelist="true"` が指定されている場合) コードカバレッジレポートに追加されますが、PHPUnit には読み込まれず、実行可能な行数の解析は行われません。
- `processUncoveredFilesFromWhitelist="true"` と指定すると、実行される行が一行もないホワイトリスト上のファイルが PHPUnit に読み込まれて、実行可能な行数の解析も行われるようになります。

Note

`processUncoveredFilesFromWhitelist="true"` が設定されている場合のソースコードファイルの読み込みでは、もしクラスや関数のスコープから外れるコードが含まれていたときに問題が起こる可能性があります。

9.3 コードブロックの無視

どうしてもテストができないコードブロックなどを、コードカバレッジ解析時に無視させたいこともあるでしょう。PHPUnit でこれを実現するには、`@codeCoverageIgnore`、`@codeCoverageIgnoreStart` および`@codeCoverageIgnoreEnd` アノテーションを Example 9.1 のように使用します。

Example 9.1 @codeCoverageIgnore、@codeCoverageIgnoreStart および @codeCoverageIgnoreEnd アノテーションの使用法

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

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

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

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

exit; // @codeCoverageIgnore
?>
```

これらのアノテーションを使って無視するよう指定された行は、もし実行可能なら (たとえ実行されていなくても) 実行されたものとみなされ、強調表示されません。

9.4 カバーするメソッドの指定

テストコードで @covers アノテーション (**カバーするメソッドを指定するためのアノテーション**) を参照ください) を使用すると、そのテストメソッドがどのメソッドをテストしたいのかを指定することができます。これを指定すると、指定したメソッドのコードカバレッジ情報のみを考慮します。Example 9.2 に例を示します。

Example 9.2　どのメソッドを対象とするかを指定したテスト

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

あるテストが、**一切** メソッドをカバーしてはならないことも指定できます。そのために使うのが @coversNothing アノテーションです。(*@coversNothing* を参照ください)。これは、インテグレーションテストを書く際にユニットテストだけのコードカバレッジを生成させたい場合に便利です。

Example 9.3　どのメソッドもカバーすべきでないことを指定したテスト

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

9.5 エッジケース

この節では、コードカバレッジ情報がわかりにくくなってしまうような、エッジケースについて紹介します。

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

// カバレッジは「行単位」であって文単位ではないので、
// 一行にまとめられた行はひとつのカバレッジ状態しか持ちません
if (false) this_function_call_shows_up_as_covered();

// コードカバレッジの内部動作上、これら 2 行は特別です。
// 次の行は「実行されていない」となります
if (false)
    // 次の行は「実行されている」となります
    // 実際のところ、ひとつ上の if 文のカバレッジ情報がここに表示されることになるからです!
    will_also_show_up_as_covered();

// これを避けるには、必ず波括弧を使わなければなりません
if (false) {
    this_call_will_never_show_up_as_covered();
}
?>
```

9.6 Xdebug を使ったコードカバレッジ出力の高速化

Xdebug 2.6 以降では、ホワイトリストによるフィルタリングを Xdebug に任せることで、コードカバレッジ用データを収集する効率を劇的に向上させることができます。

そのためには、まず `--dump-xdebug-filter` オプションを使って Xdebug 用のフィルタースクリプトを生成します。

```
$ phpunit --dump-xdebug-filter build/xdebug-filter.php
PHPUnit 7.4.0 by Sebastian Bergmann and contributors.

Runtime:       PHP 7.2.11 with Xdebug 2.6.1
Configuration: /workspace/project/phpunit.xml
```

```
Wrote Xdebug filter script to build/xdebug-filter.php
```

これで、 コードカバレッジレポートの生成時に `--prepend` オプションを使って Xdebug フィルタースクリプトをロードできるようになりました。

```
$ phpunit --prepend build/xdebug-filter.php --coverage-html build/coverage-report
```

第 10 章

PHPUnit の拡張

テストを書きやすくする、あるいはテストの実行結果の表示方法を変更するなど、PHPUnit はさまざまな方法で拡張することができます。PHPUnit を拡張するための第一歩をここで説明します。

10.1 PHPUnit\Framework\TestCase のサブクラスの作成

`PHPUnit\Framework\TestCase` を継承した抽象サブクラスにカスタムアサーションやユーティリティメソッドを書き、そのクラスをさらに継承してテストクラスを作成します。これが、PHPUnit を拡張するための一番簡単な方法です。

10.2 カスタムアサーションの作成

カスタムアサーションを作成するときには、PHPUnit 自体のアサーションの実装方法を真似るのがおすすめです。Example 10.1 を見ればわかるとおり、`assertTrue()` メソッドは `isTrue()` および `assertThat()` メソッドの単なるラッパーに過ぎません。`isTrue()` が matcher オブジェクトを作り、それを `assertThat()` に渡して評価しています。

Example 10.1　PHPUnit\Framework\Assert クラスの assertTrue() および isTrue() メソッド

```
<?php
namespace PHPUnit\Framework;

use PHPUnit\Framework\TestCase;

abstract class Assert
{
    // ...

    /**
     * Asserts that a condition is true.
     *
```

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

    // ...

    /**
     * Returns a PHPUnit\Framework\Constraint\IsTrue matcher object.
     *
     * @return PHPUnit\Framework\Constraint\IsTrue
     * @since  Method available since Release 3.3.0
     */
    public static function isTrue()
    {
        return new PHPUnit\Framework\Constraint\IsTrue;
    }

    // ...
}
```

Example 10.2 は、PHPUnit\Framework\Constraint\IsTrue が matcher オブジェクト (あるいは制約) のために抽象クラス PHPUnit\Framework\Constraint を継承している部分です。

Example 10.2　PHPUnit\Framework\Constraint\IsTrue クラス

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

use PHPUnit\Framework\Constraint;

class IsTrue extends Constraint
{
    /**
     * Evaluates the constraint for parameter $other. Returns true if the
     * constraint is met, false otherwise.
     *
     * @param mixed $other Value or object to evaluate.
     * @return bool
     */
    public function matches($other)
    {
        return $other === true;
    }
```

```
    /**
     * Returns a string representation of the constraint.
     *
     * @return string
     */
    public function toString()
    {
        return 'is true';
    }
}?>
```

assertTrue() や isTrue() メソッドの実装を PHPUnit\Framework\Constraint\IsTrue クラスと同じようにしておけば、アサーションの評価やタスクの記録 (テストの統計情報に自動的に更新するなど) を assertThat() が自動的に行ってくれるようになります。さらに、モックオブジェクトを設定する際の matcher として isTrue() メソッドを使えるようにもなります。

10.3 PHPUnit\Framework\TestListener の実装

Example 10.3 は、PHPUnit\Framework\TestListener インターフェイスのシンプルな実装例です。

Example 10.3 シンプルなテストリスナー

```
<?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("テスト '%s' の実行中にエラーが発生\n", $test->getName());
    }

    public function addWarning(PHPUnit\Framework\Test $test, PHPUnit\Framework\Warning
↪$e, float $time): void
    {
        printf("テスト '%s' の実行中に警告が発生\n", $test->getName());
    }

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

```
    public function addIncompleteTest(PHPUnit\Framework\Test $test, \Throwable $e,␣
↪float $time): void
    {
        printf("テスト '%s' は未完成\n", $test->getName());
    }

    public function addRiskyTest(PHPUnit\Framework\Test $test, \Throwable $e, float
↪$time): void
    {
        printf("テスト '%s' は危険\n", $test->getName());
    }

    public function addSkippedTest(PHPUnit\Framework\Test $test, \Throwable $e, float
↪$time): void
    {
        printf("テスト '%s' をスキップ\n", $test->getName());
    }

    public function startTest(PHPUnit\Framework\Test $test): void
    {
        printf("テスト '%s' が開始\n", $test->getName());
    }

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

    public function startTestSuite(PHPUnit\Framework\TestSuite $suite): void
    {
        printf("テストスイート '%s' が開始\n", $suite->getName());
    }

    public function endTestSuite(PHPUnit\Framework\TestSuite $suite): void
    {
        printf("テストスイート '%s' が終了\n", $suite->getName());
    }
}
?>
```

Example 10.4 は、トレイト PHPUnit\Framework\TestListenerDefaultImplementation の使用例です。これは、インターフェイスのメソッドのうち実際に使うものだけを指定し、他のメソッドについては空の実装を提供します。

Example 10.4　テストリスナーのデフォルト実装のトレイトの利用法

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

class ShortTestListener implements TestListener
{
    use TestListenerDefaultImplementation;

    public function endTest(PHPUnit\Framework\Test $test, float $time): void
    {
        printf("テスト '%s' が終了\n", $test->getName());
    }
}
?>
```

テストリスナーに、自作のテストリスナーをテスト実行時にアタッチするための PHPUnit の設定方法についての説明があります。

10.4 PHPUnit\Framework\Test の実装

`PHPUnit\Framework\Test` インターフェイスの機能は限られており、実装するのは簡単です。`PHPUnit\Framework\Test` を実装するのは `PHPUnit\Framework\TestCase` の実装より単純で、これを用いて例えば **データ駆動のテスト** *(data-driven tests)* などを実行します。

カンマ区切り (CSV) ファイルの値と比較する、データ駆動のテストを Example 10.5 に示します。このファイルの各行は `foo;bar` のような形式になっており (訳注: CSV じゃない……)、最初の値が期待値で 2 番目の値が実際の値です。

Example 10.5　データ駆動のテスト

```
<?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 |version|.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.
```

10.5 TestRunner の拡張

PHPUnit latest は TestRunner エクステンションをサポートしており、テストの実行中のさまざまなイベントにフックを組み込めます。PHPUnit の XML 設定ファイルでエクステンションを組み込む方法については *TestRunner* **エクステンションの組み込み** を参照ください。

エクステンションでフックを組み込めるイベントはインターフェイスとして公開されており、これをエクステンションが実装する必要があります。PHPUnit latest で使えるイベントの一覧を**利用可能なフックインターフェイス**に示します。

10.5.1 利用可能なフックインターフェイス

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

Example 10.6 に、BeforeFirstTestHook と AfterLastTestHook を実装したエクステンションの例を示します。

Example 10.6　TestRunner エクステンションの例

```
<?php

namespace Vendor;

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

final class MyExtension implements BeforeFirstTestHook, AfterLastTestHook
{
    public function executeAfterLastTest(): void
    {
        // 最後のテストの実行後にコールされます
    }

    public function executeBeforeFirstTest(): void
    {
        // 最初のテストの実行前にコールされます
    }
}
```

第 11 章

アサーション

この節では、利用可能なアサーションメソッドの一覧を示します。

11.1 アサーションメソッドは static で使うべきか、それとも非 static で使うべきか

PHPUnit のアサーションの実装は、`PHPUnit\Framework\Assert` およびそれを継承した `PHPUnit\Framework\TestCase` にあります。

アサーションメソッドは static 宣言されていて、あらゆるコンテキストから `PHPUnit\Framework\Assert::assertTrue()` などのように使えます。また、`PHPUnit\Framework\TestCase` を継承したクラスの中では`$this->assertTrue()` や `self::assertTrue()` などとしても使えます。

さらに、PHPUnit に含まれるファイル `src/Framework/Assert/Functions.php` を (手動で) インクルードしてしまえば、グローバルなラッパー関数 `assertTrue()` などを使うことさえできてしまいます。これは、`PHPUnit\Framework\TestCase` を継承したクラスを含めてあらゆるコンテキストで使えます。

PHPUnit を使い始めたばかりの開発者の多くは、たとえばアサーションを実行するときに`$this->assertTrue()` と `self::assertTrue()` のどちらを使うのが「正しい方法」なのだろうかと思うことでしょう。端的に言って、どちらがよいなどということはありません。同様に、どちらが悪いというものでもありません。どちらを使うかは、個人的な好みの問題です。

どちらかといえば `$this->assertTrue()` を使うほうが自然に感じるという人が多いようです。テストメソッドは、テストオブジェクト上で実行されるからです。アサーションメソッドが static 宣言されていることによって、テストオブジェクトのスコープを超えた再利用が可能になります。また、グローバル関数のラッパーを使えば、(`$this->assertTrue()` や `self::assertTrue()` ではなく `assertTrue()` と書くことによって) タイプ量を抑えることができます。

11.2 assertArrayHasKey()

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

`$array` にキー `$key` が存在しない場合にエラー `$message` を報告します。

`assertArrayNotHasKey()` はこのアサーションの逆で、同じ引数をとります。

Example 11.1 assertArrayHasKey() の使用法

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

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

```
$ phpunit ArrayHasKeyTest
PHPUnit 7.0.0 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.
```

11.3 assertClassHasAttribute()

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

`$className::attributeName` が存在しない場合にエラー `$message` を報告します。

`assertClassNotHasAttribute()` はこのアサーションの逆で、同じ引数をとります。

Example 11.2　assertClassHasAttribute() の使用法

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

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

```
$ phpunit ClassHasAttributeTest
PHPUnit 7.0.0 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.
```

11.4 assertArraySubset()

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

`$array` が `$subset` を含まない場合にエラー `$message` を報告します。

`$strict` フラグを使うと、配列内のオブジェクトの比較にその識別子を利用します。

Example 11.3　assertArraySubset() の使用法

```
<?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 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) Epilog\EpilogTest::testNoFollowOption
Failed asserting that an array has the subset Array &0 (
    'config' => Array &1 (
        0 => 'key-a'
        1 => 'key-b'
    )
).

/home/sb/ArraySubsetTest.php:6

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

11.5 assertClassHasStaticAttribute()

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

`$className::attributeName` が存在しない場合にエラー `$message` を報告します。

`assertClassNotHasStaticAttribute()` はこのアサーションの逆で、同じ引数をとります。

Example 11.4 assertClassHasStaticAttribute() の使用法

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

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

```
    }
}
```

```
$ phpunit ClassHasStaticAttributeTest
PHPUnit 7.0.0 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.
```

11.6 assertContains()

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

`$needle` が `$haystack` の要素でない場合にエラー `$message` を報告します。

`assertNotContains()` はこのアサーションの逆で、同じ引数をとります。

`assertAttributeContains()` と `assertAttributeNotContains()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性を haystack として使用することができます。

Example 11.5 assertContains() の使用法

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

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

```
$ phpunit ContainsTest
PHPUnit 7.0.0 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])`

`$needle` が `$haystack` の部分文字列でない場合にエラー `$message` を報告します。

`$ignoreCase` が `true` の場合、テストで大文字小文字を区別しなくなります。

Example 11.6　assertContains() の使用法

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

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains('baz', 'foobar');
    }
}
```

```
$ phpunit ContainsTest
PHPUnit 7.0.0 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.
```

Example 11.7　assertContains() で $ignoreCase を使う方法

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

class ContainsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContains('foo', 'FooBar');
    }

    public function testOK()
    {
        $this->assertContains('foo', 'FooBar', '', true);
    }
}
```

```
$ phpunit ContainsTest
PHPUnit 7.0.0 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.
```

11.7 assertContainsOnly()

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

`$haystack` の中身の型が `$type` だけではない場合にエラー `$message` を報告します。

`$isNativeType` はフラグで、`$type` がネイティブな PHP の型であるかどうかを表します。

assertNotContainsOnly() はこのアサーションの逆で、同じ引数をとります。

assertAttributeContainsOnly() と assertAttributeNotContainsOnly() は便利なラッパーで、クラスやオブジェクトの public、protected、private 属性を haystack として使用することができます。

Example 11.8 assertContainsOnly() の使用法

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

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

```
$ phpunit ContainsOnlyTest
PHPUnit 7.0.0 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.
```

11.8 assertContainsOnlyInstancesOf()

assertContainsOnlyInstancesOf(string $classname, Traversable|array $haystack[, string $message = ''])

$haystack が $classname クラスの唯一のインスタンスを含まない場合にエラー $message を報告します。

Example 11.9　assertContainsOnlyInstancesOf() の使用法

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

class ContainsOnlyInstancesOfTest extends TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnlyInstancesOf(
            Foo::class,
            [new Foo, new Bar, new Foo]
        );
    }
}
```

```
$ phpunit ContainsOnlyInstancesOfTest
PHPUnit 7.0.0 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.
```

11.9 assertCount()

`assertCount($expectedCount, $haystack[, string $message = ''])`

`$haystack` の要素数が `$expectedCount` でない場合にエラー `$message` を報告します。

`assertNotCount()` はこのアサーションの逆で、同じ引数をとります。

Example 11.10　assertCount() の使用法

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

class CountTest extends TestCase
```

```
{
    public function testFailure()
    {
        $this->assertCount(0, ['foo']);
    }
}
```

```
$ phpunit CountTest
PHPUnit 7.0.0 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.
```

11.10 assertDirectoryExists()

`assertDirectoryExists(string $directory[, string $message = ''])`

`$directory` で指定したディレクトリが存在しない場合にエラー `$message` を報告します。

`assertDirectoryNotExists()` はこのアサーションの逆で、同じ引数をとります。

Example 11.11　assertDirectoryExists() の使用法

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

class DirectoryExistsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryExists('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryExistsTest
PHPUnit 7.0.0 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.
```

11.11 assertDirectoryIsReadable()

`assertDirectoryIsReadable(string $directory[, string $message = ''])`

`$directory` で指定したディレクトリが読み込み可能でない場合にエラー `$message` を報告します。

`assertDirectoryNotIsReadable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.12　assertDirectoryIsReadable() の使用法

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

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

```
        $this->assertDirectoryIsReadable('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryIsReadableTest
PHPUnit 7.0.0 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.
```

11.12 assertDirectoryIsWritable()

`assertDirectoryIsWritable(string $directory[, string $message = ''])`

`$directory` で指定したディレクトリが書き込み可能でない場合にエラー `$message` を報告します。

`assertDirectoryNotIsWritable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.13 assertDirectoryIsWritable() の使用法

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

class DirectoryIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertDirectoryIsWritable('/path/to/directory');
    }
}
```

```
$ phpunit DirectoryIsWritableTest
PHPUnit 7.0.0 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.
```

11.13 assertEmpty()

`assertEmpty(mixed $actual[, string $message = ''])`

`$actual` が空でない場合にエラー `$message` を報告します。

`assertNotEmpty()` はこのアサーションの逆で、同じ引数をとります。

`assertAttributeEmpty()` および `assertAttributeNotEmpty()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性に対して使えます。

Example 11.14 assertEmpty() の使用法

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

class EmptyTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEmpty(['foo']);
    }
}
```

```
$ phpunit EmptyTest
PHPUnit 7.0.0 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.
```

11.14 assertEqualXMLStructure()

`assertEqualXMLStructure(DOMElement $expectedElement, DOMElement $actualElement[, boolean $checkAttributes = false, string $message = ''])`

`$actualElement` の DOMElement の XML 構造が `$expectedElement` の DOMElement の XML 構造と等しくない場合にエラー `$message` を報告します。

Example 11.15 assertEqualXMLStructure() の使用法

```
<?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 7.0.0 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.
```

11.15 assertEquals()

`assertEquals(mixed $expected, mixed $actual[, string $message = ''])`

2 つの変数 `$expected` と `$actual` が等しくない場合にエラー `$message` を報告します。

`assertNotEquals()` はこのアサーションの逆で、同じ引数をとります。

`assertAttributeEquals()` と `assertAttributeNotEquals()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性を実際の値として使用することができます。

Example 11.16　assertEquals() の使用法

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

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEquals(1, 0);
    }

    public function testFailure2()
    {
        $this->assertEquals('bar', 'baz');
    }

    public function testFailure3()
```

```
    {
        $this->assertEquals("foo\nbar\nbaz\n", "foo\nbah\nbaz\n");
    }
}
```

```
$ phpunit EqualsTest
PHPUnit 7.0.0 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.
```

引数 `$expected` と `$actual` の型により特化した比較については、以下を参照ください。

```
assertEquals(float $expected, float $actual[, string $message = '', float
```

```
$delta = 0])
```

2 つの float 値 $expected と $actual の絶対差が $delta より大きい場合にエラー $message を報告します。2 つの float 値 $expected と $actual の絶対差が $delta **以下** である場合はアサーションに成功します。

なぜ $delta が必要となるのかについては“ What Every Computer Scientist Should Know About Floating-Point Arithmetic ”を参照ください。

Example 11.17　float 値での assertEquals() の使用法

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

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 7.0.0 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 =
'']])
```

2 つの DOMDocument オブジェクト $expected と $actual で表される XML ドキュメントが (コメントを除去して正規化した状態で) 等しくない場合にエラー $message を報告します。

Example 11.18　DOMDocument オブジェクトでの assertEquals() の使用法

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

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo><bar/></foo>');

        $actual = new DOMDocument;
        $actual->loadXML('<bar><foo/></bar>');

        $this->assertEquals($expected, $actual);
    }
}
```

```
$ phpunit EqualsTest
PHPUnit 7.0.0 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 = ''])
```

2 つのオブジェクト `$expected` と `$actual` が同じ属性値を持たない場合にエラー `$message` を報告します。

Example 11.19　オブジェクトでの assertEquals() の使用法

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

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $expected = new stdClass;
        $expected->foo = 'foo';
        $expected->bar = 'bar';

        $actual = new stdClass;
        $actual->foo = 'bar';
        $actual->baz = 'bar';

        $this->assertEquals($expected, $actual);
    }
}
?>
```

```
$ phpunit EqualsTest
PHPUnit 7.0.0 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 = ''])
```

2 つの配列 $expected と $actual が等しくない場合にエラー $message を報告します。

Example 11.20 配列での assertEquals() の使用法

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

class EqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertEquals(['a', 'b', 'c'], ['a', 'c', 'd']);
    }
}
?>
```

```
$ phpunit EqualsTest
PHPUnit 7.0.0 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.
```

11.16 assertFalse()

```
assertFalse(bool $condition[, string $message = ''])
```

`$condition` が `true` の場合にエラー `$message` を報告します。

`assertNotFalse()` はこのアサーションの逆で、同じ引数をとります。

Example 11.21　assertFalse() の使用法

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

class FalseTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFalse(true);
    }
}
?>
```

```
$ phpunit FalseTest
PHPUnit 7.0.0 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.
```

11.17 assertFileEquals()

`assertFileEquals(string $expected, string $actual[, string $message = ''])`

`$expected` で指定したファイルと `$actual` で指定したファイルの内容が異なる場合にエラー `$message` を報告します。

`assertFileNotEquals()` はこのアサーションの逆で、同じ引数をとります。

Example 11.22　assertFileEquals() の使用法

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

class FileEqualsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileEquals('/home/sb/expected', '/home/sb/actual');
    }
}
?>
```

```
$ phpunit FileEqualsTest
PHPUnit 7.0.0 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.
```

11.18 assertFileExists()

`assertFileExists(string $filename[, string $message = ''])`

ファイル `$filename` が存在しない場合にエラー `$message` を報告します。

`assertFileNotExists()` はこのアサーションの逆で、同じ引数をとります。

Example 11.23　assertFileExists() の使用法

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

class FileExistsTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileExists('/path/to/file');
    }
}
?>
```

```
$ phpunit FileExistsTest
PHPUnit 7.0.0 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.
```

11.19 assertFileIsReadable()

`assertFileIsReadable(string $filename[, string $message = ''])`

`$filename` で指定したファイルが読み込み可能でない場合、あるいはファイルでない場合にエラー `$message` を報告します。

`assertFileNotIsReadable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.24　assertFileIsReadable() の使用法

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

class FileIsReadableTest extends TestCase
{
```

```
    public function testFailure()
    {
        $this->assertFileIsReadable('/path/to/file');
    }
}
?>
```

```
$ phpunit FileIsReadableTest
PHPUnit 7.0.0 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.
```

11.20 assertFileIsWritable()

`assertFileIsWritable(string $filename[, string $message = ''])`

`$filename` で指定したファイルが書き込み可能でない場合、あるいはファイルでない場合にエラー `$message` を報告します。

`assertFileNotIsWritable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.25 assertFileIsWritable() の使用法

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

class FileIsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertFileIsWritable('/path/to/file');
    }
}
?>
```

```
$ phpunit FileIsWritableTest
PHPUnit 7.0.0 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.
```

11.21 assertGreaterThan()

`assertGreaterThan(mixed $expected, mixed $actual[, string $message = ''])`

`$actual` の値が `$expected` の値より大きくない場合にエラー `$message` を報告します。

`assertAttributeGreaterThan()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性を実際の値として使用することができます。

Example 11.26 assertGreaterThan() の使用法

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

class GreaterThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertGreaterThan(2, 1);
    }
}
?>
```

```
$ phpunit GreaterThanTest
PHPUnit 7.0.0 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.
```

11.22 assertGreaterThanOrEqual()

`assertGreaterThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])`

`$actual` の値が `$expected` の値以上でない場合にエラー `$message` を報告します。

`assertAttributeGreaterThanOrEqual()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性を実際の値として使用することができます。

Example 11.27 assertGreaterThanOrEqual() の使用法

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

class GreatThanOrEqualTest extends TestCase
{
    public function testFailure()
    {
        $this->assertGreaterThanOrEqual(2, 1);
    }
}
?>
```

```
$ phpunit GreaterThanOrEqualTest
PHPUnit 7.0.0 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.
```

11.23 assertInfinite()

`assertInfinite(mixed $variable[, string $message = ''])`

`$variable` が INF でない場合にエラー `$message` を報告します。

`assertFinite()` はこのアサーションの逆で、同じ引数をとります。

Example 11.28 assertInfinite() の使用法

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

class InfiniteTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInfinite(1);
    }
}
?>
```

```
$ phpunit InfiniteTest
PHPUnit 7.0.0 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.
```

11.24 assertInstanceOf()

`assertInstanceOf($expected, $actual[, $message = ''])`

`$actual` が `$expected` のインスタンスでない場合にエラー `$message` を報告します。

`assertNotInstanceOf()` はこのアサーションの逆で、同じ引数をとります。

`assertAttributeInstanceOf()` および `assertAttributeNotInstanceOf()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性に対して使えます。

Example 11.29 assertInstanceOf() の使用法

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

class InstanceOfTest extends TestCase
{
    public function testFailure()
    {
        $this->assertInstanceOf(RuntimeException::class, new Exception);
    }
}
?>
```

```
$ phpunit InstanceOfTest
PHPUnit 7.0.0 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.
```

11.25 assertIsArray()

`assertIsArray($actual[, $message = ''])`

`$actual` の型が `array` でない場合にエラー `$message` を報告します。

`assertIsNotArray()` はこのアサーションの逆で、同じ引数をとります。

Example 11.30　assertIsArray() の使用法

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

class ArrayTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsArray(null);
    }
}
```

```
$ phpunit ArrayTest
PHPUnit |version|.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.
```

11.26 assertIsBool()

`assertIsBool($actual[, $message = ''])`

`$actual` の型が `bool` でない場合にエラー `$message` を報告します。

`assertIsNotBool()` はこのアサーションの逆で、同じ引数をとります。

Example 11.31　assertIsBool() の使用法

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

class BoolTest extends TestCase
```

```
{
    public function testFailure()
    {
        $this->assertIsBool(null);
    }
}
```

```
$ phpunit BoolTest
PHPUnit |version|.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.
```

11.27 assertIsCallable()

`assertIsCallable($actual[, $message = ''])`

`$actual` の型が `callable` でない場合にエラー `$message` を報告します。

`assertIsNotCallable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.32　assertIsCallable() の使用法

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

class CallableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsCallable(null);
    }
}
```

```
$ phpunit CallableTest
PHPUnit |version|.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.
```

11.28 assertIsFloat()

`assertIsFloat($actual[, $message = ''])`

`$actual` の型が `float` でない場合にエラー `$message` を報告します。

`assertIsNotFloat()` はこのアサーションの逆で、同じ引数をとります。

Example 11.33　assertIsFloat() の使用法

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

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

```
        $this->assertIsFloat(null);
    }
}
```

```
$ phpunit FloatTest
PHPUnit |version|.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.
```

11.29 assertIsInt()

`assertIsInt($actual[, $message = ''])`

`$actual` の型が `int` でない場合にエラー `$message` を報告します。

`assertIsNotInt()` はこのアサーションの逆で、同じ引数をとります。

Example 11.34 assertIsInt() の使用法

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

class IntTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsInt(null);
    }
}
```

```
$ phpunit IntTest
PHPUnit |version|.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.
```

11.30 assertIsIterable()

`assertIsIterable($actual[, $message = ''])`

`$actual` の型が `iterable` でない場合にエラー `$message` を報告します。

`assertIsNotIterable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.35 assertIsIterable() の使用法

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

class IterableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsIterable(null);
    }
}
```

```
$ phpunit IterableTest
PHPUnit |version|.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.
```

11.31 assertIsNumeric()

`assertIsNumeric($actual[, $message = ''])`

`$actual` の型が `numeric` でない場合にエラー `$message` を報告します。

`assertIsNotNumeric()` はこのアサーションの逆で、同じ引数をとります。

Example 11.36 assertIsNumeric() の使用法

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

class NumericTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsNumeric(null);
    }
}
```

```
$ phpunit NumericTest
PHPUnit |version|.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.
```

11.32 assertIsObject()

`assertIsObject($actual[, $message = ''])`

`$actual` の型が `object` でない場合にエラー `$message` を報告します。

`assertIsNotObject()` はこのアサーションの逆で、同じ引数をとります。

Example 11.37　assertIsObject() の使用法

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

class ObjectTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsObject(null);
    }
}
```

```
$ phpunit ObjectTest
PHPUnit |version|.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.
```

11.33 assertIsResource()

`assertIsResource($actual[, $message = ''])`

`$actual` の型が `resource` でない場合にエラー `$message` を報告します。

`assertIsNotResource()` はこのアサーションの逆で、同じ引数をとります。

Example 11.38　assertIsResource() の使用法

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

class ResourceTest extends TestCase
```

```
{
    public function testFailure()
    {
        $this->assertIsResource(null);
    }
}
```

```
$ phpunit ResourceTest
PHPUnit |version|.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.
```

11.34 assertIsScalar()

`assertIsScalar($actual[, $message = ''])`

`$actual` の型が `scalar` でない場合にエラー `$message` を報告します。

`assertIsNotScalar()` はこのアサーションの逆で、同じ引数をとります。

Example 11.39　assertIsScalar() の使用法

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

class ScalarTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsScalar(null);
    }
}
```

```
$ phpunit ScalarTest
PHPUnit |version|.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.
```

11.35 assertIsString()

`assertIsString($actual[, $message = ''])`

`$actual` の型が `string` でない場合にエラー `$message` を報告します。

`assertIsNotString()` はこのアサーションの逆で、同じ引数をとります。

Example 11.40　assertIsString() の使用法

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

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

```
        $this->assertIsString(null);
    }
}
```

```
$ phpunit StringTest
PHPUnit |version|.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.
```

11.36 assertIsReadable()

`assertIsReadable(string $filename[, string $message = ''])`

`$filename` で指定したファイルあるいはディレクトリが読み込み可能でない場合にエラー `$message` を報告します。

`assertNotIsReadable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.41 assertIsReadable() の使用法

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

class IsReadableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsReadable('/path/to/unreadable');
    }
}
?>
```

```
$ phpunit IsReadableTest
PHPUnit 7.0.0 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.
```

11.37 assertIsWritable()

`assertIsWritable(string $filename[, string $message = ''])`

`$filename` で指定したファイルあるいはディレクトリが書き込み可能でない場合にエラー `$message` を報告します。

`assertNotIsWritable()` はこのアサーションの逆で、同じ引数をとります。

Example 11.42 assertIsWritable() の使用法

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

class IsWritableTest extends TestCase
{
    public function testFailure()
    {
        $this->assertIsWritable('/path/to/unwritable');
    }
}
?>
```

```
$ phpunit IsWritableTest
PHPUnit 7.0.0 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.
```

11.38 assertJsonFileEqualsJsonFile()

`assertJsonFileEqualsJsonFile(mixed $expectedFile, mixed $actualFile[, string $message = ''])`

`$actualFile` の値が`$expectedFile` の値にマッチしない場合にエラー `$message` を報告します。

Example 11.43 assertJsonFileEqualsJsonFile() の使用法

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

class JsonFileEqualsJsonFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonFileEqualsJsonFile(
          'path/to/fixture/file', 'path/to/actual/file');
    }
}
?>
```

```
$ phpunit JsonFileEqualsJsonFileTest
PHPUnit 7.0.0 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.
```

11.39 assertJsonStringEqualsJsonFile()

`assertJsonStringEqualsJsonFile(mixed $expectedFile, mixed $actualJson[, string $message = ''])`

`$actualJson` の値が`$expectedFile` の値にマッチしない場合にエラー `$message` を報告します。

Example 11.44 assertJsonStringEqualsJsonFile() の使用法

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

class JsonStringEqualsJsonFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonFile(
            'path/to/fixture/file', json_encode(['Mascot' => 'ux'])
        );
    }
}
?>
```

```
$ phpunit JsonStringEqualsJsonFileTest
PHPUnit 7.0.0 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.
```

11.40 assertJsonStringEqualsJsonString()

`assertJsonStringEqualsJsonString(mixed $expectedJson, mixed $actualJson[, string $message = ''])`

`$actualJson` の値が`$expectedJson` の値にマッチしない場合にエラー `$message` を報告します。

Example 11.45 assertJsonStringEqualsJsonString() の使用法

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

class JsonStringEqualsJsonStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonString(
            json_encode(['Mascot' => 'Tux']),
            json_encode(['Mascot' => 'ux'])
        );
    }
}
?>
```

```
$ phpunit JsonStringEqualsJsonStringTest
PHPUnit 7.0.0 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.
```

11.41 assertLessThan()

assertLessThan(mixed $expected, mixed $actual[, string $message = ''])

$actual の値が $expected の値より小さくない場合にエラー $message を報告します。

assertAttributeLessThan() は便利なラッパーで、クラスやオブジェクトの public、protected、private 属性を実際の値として使用することができます。

Example 11.46 assertLessThan() の使用法

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

class LessThanTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThan(1, 2);
    }
}
?>
```

```
$ phpunit LessThanTest
PHPUnit 7.0.0 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.
```

11.42 assertLessThanOrEqual()

assertLessThanOrEqual(mixed $expected, mixed $actual[, string $message = ''])

$actual の値が $expected の値以下でない場合にエラー $message を報告します。

assertAttributeLessThanOrEqual() は便利なラッパーで、クラスやオブジェクトの public、

protected、private 属性を実際の値として使用することができます。

Example 11.47　assertLessThanOrEqual() の使用法

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

class LessThanOrEqualTest extends TestCase
{
    public function testFailure()
    {
        $this->assertLessThanOrEqual(1, 2);
    }
}
?>
```

```
$ phpunit LessThanOrEqualTest
PHPUnit 7.0.0 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.
```

11.43 assertNan()

```
assertNan(mixed $variable[, string $message = ''])
```

$variable が NAN でない場合にエラー $message を報告します。

Example 11.48　assertNan() の使用法

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

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

```
        $this->assertNan(1);
    }
}
?>
```

```
$ phpunit NanTest
PHPUnit 7.0.0 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.
```

11.44 assertNull()

`assertNull(mixed $variable[, string $message = ''])`

`$variable` が NULL でないときにエラー `$message` を報告します。

`assertNotNull()` はこのアサーションの逆で、同じ引数をとります。

Example 11.49 assertNull() の使用法

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

class NullTest extends TestCase
{
    public function testFailure()
    {
        $this->assertNull('foo');
    }
}
?>
```

```
$ phpunit NotNullTest
PHPUnit 7.0.0 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.
```

11.45 assertObjectHasAttribute()

`assertObjectHasAttribute(string $attributeName, object $object[, string $message = ''])`

`$object->attributeName` が存在しない場合にエラー `$message` を報告します。

`assertObjectNotHasAttribute()` はこのアサーションの逆で、同じ引数をとります。

Example 11.50 assertObjectHasAttribute() の使用法

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

class ObjectHasAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertObjectHasAttribute('foo', new stdClass);
    }
}
?>
```

```
$ phpunit ObjectHasAttributeTest
PHPUnit 7.0.0 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.
```

11.46 assertRegExp()

`assertRegExp(string $pattern, string $string[, string $message = ''])`

`$string` が正規表現 `$pattern` にマッチしない場合にエラー `$message` を報告します。

`assertNotRegExp()` はこのアサーションの逆で、同じ引数をとります。

Example 11.51 assertRegExp() の使用法

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

class RegExpTest extends TestCase
{
    public function testFailure()
    {
        $this->assertRegExp('/foo/', 'bar');
    }
}
?>
```

```
$ phpunit RegExpTest
PHPUnit 7.0.0 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.
```

11.47 assertStringMatchesFormat()

`assertStringMatchesFormat(string $format, string $string[, string $message = ''])`

`$string` が書式文字列 `$format` にマッチしない場合にエラー `$message` を報告します。

`assertStringNotMatchesFormat()` はこのアサーションの逆で、同じ引数をとります。

Example 11.52 assertStringMatchesFormat() の使用法

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

class StringMatchesFormatTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormat('%i', 'foo');
    }
}
?>
```

```
$ phpunit StringMatchesFormatTest
PHPUnit 7.0.0 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.
```

書式文字列には次のプレースホルダを含めることができます。

- `%e`: ディレクトリ区切り文字、たとえば Linux なら / を表します。
- `%s`: 一文字以上の何か (文字あるいは空白)、ただし改行文字は含みません。
- `%S`: ゼロ文字以上の何か (文字あるいは空白)、ただし改行文字は含みません。
- `%a`: 一文字以上の何か (文字あるいは空白)、改行文字も含みます。

- %A: ゼロ文字以上の何か (文字あるいは空白)、改行文字も含みます。
- %w: ゼロ文字以上の空白。
- %i: 符号付き整数値。例: +3142, -3142
- %d: 符号なし整数値。例: 123.66
- %x: 一文字以上の十六進文字 (0-9, a-f, A-F)。
- %f: 浮動小数点数値。例: 3.142, -3.142, 3.142E-10, 3.142e+10
- %c: 任意の一文字。
- %%: パーセント文字 % そのもの。

11.48 assertStringMatchesFormatFile()

assertStringMatchesFormatFile(string $formatFile, string $string[, string $message = ''])

$string が $formatFile の内容にマッチしない場合にエラー $message を報告します。

assertStringNotMatchesFormatFile() はこのアサーションの逆で、同じ引数をとります。

Example 11.53 assertStringMatchesFormatFile() の使用法

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

class StringMatchesFormatFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormatFile('/path/to/expected.txt', 'foo');
    }
}
?>
```

```
$ phpunit StringMatchesFormatFileTest
PHPUnit 7.0.0 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.
```

11.49 assertSame()

`assertSame(mixed $expected, mixed $actual[, string $message = ''])`

2 つの変数 `$expected` と `$actual` が同じ型・同じ値でない場合にエラー `$message` を報告します。

`assertNotSame()` はこのアサーションの逆で、同じ引数をとります。

`assertAttributeSame()` と `assertAttributeNotSame()` は便利なラッパーで、クラスやオブジェクトの `public`、`protected`、`private` 属性を実際の値として使用することができます。

Example 11.54 assertSame() の使用法

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame('2204', 2204);
    }
}
?>
```

```
$ phpunit SameTest
PHPUnit 7.0.0 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 = ''])

2 つの変数 $expected と $actual が同じオブジェクトを参照していない場合にエラー $message を報告します。

Example 11.55　オブジェクトでの assertSame() の使用法

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame(new stdClass, new stdClass);
    }
}
?>
```

```
$ phpunit SameTest
PHPUnit 7.0.0 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.
```

11.50 assertStringEndsWith()

assertStringEndsWith(string $suffix, string $string[, string $message = ''])

$string が $suffix で終わっていない場合にエラー $message を報告します。

assertStringEndsNotWith() はこのアサーションの逆で、同じ引数をとります。

Example 11.56　assertStringEndsWith() の使用法

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

class StringEndsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEndsWith('suffix', 'foo');
    }
}
?>
```

```
$ phpunit StringEndsWithTest
PHPUnit 7.0.0 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.
```

11.51 assertStringEqualsFile()

`assertStringEqualsFile(string $expectedFile, string $actualString[, string $message = ''])`

`$expectedFile` で指定したファイルの内容に `$actualString` が含まれない場合にエラー `$message` を報告します。

`assertStringNotEqualsFile()` はこのアサーションの逆で、同じ引数をとります。

Example 11.57　assertStringEqualsFile() の使用法

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

class StringEqualsFileTest extends TestCase
```

```
{
    public function testFailure()
    {
        $this->assertStringEqualsFile('/home/sb/expected', 'actual');
    }
}
?>
```

```
$ phpunit StringEqualsFileTest
PHPUnit 7.0.0 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.
```

11.52 assertStringStartsWith()

`assertStringStartsWith(string $prefix, string $string[, string $message = ''])`

`$string` が `$prefix` で始まっていない場合にエラー `$message` を報告します。

`assertStringStartsNotWith()` はこのアサーションの逆で、同じ引数をとります。

Example 11.58 assertStringStartsWith() の使用法

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

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

```
    {
        $this->assertStringStartsWith('prefix', 'foo');
    }
}
?>
```

```
$ phpunit StringStartsWithTest
PHPUnit 7.0.0 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.
```

11.53 assertThat()

もっと複雑なアサーションを行う場合には、PHPUnit\Framework\Constraint クラスを使用します。これらは、assertThat() メソッドを使用して評価されます。Example 11.59 は、logicalNot() と equalTo() を用いて assertNotEquals() と同じアサーションを行う方法を示すものです。

assertThat(mixed $value, PHPUnit\Framework\Constraint $constraint[, $message = ''])

$value が $constraint にマッチしない場合にエラー $message を報告します。

Example 11.59 assertThat() の使用法

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

class BiscuitTest extends TestCase
{
    public function testEquals()
    {
        $theBiscuit = new Biscuit('Ginger');
        $myBiscuit  = new Biscuit('Ginger');
```

```
        $this->assertThat(
          $theBiscuit,
          $this->logicalNot(
            $this->equalTo($myBiscuit)
          )
        );
    }
}
?>
```

Table 11.1 に、使用できる PHPUnit\Framework\Constraint クラスをまとめます。

制約
PHPUnit\Framework\Constraint\Attribute attribute(PHPUnit\Framework\Constraint $constraint,
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 $class
PHPUnit\Framework\Constraint\IsEqual equalTo($value, $delta = 0, $maxDepth = 10)
PHPUnit\Framework\Constraint\Attribute attributeEqualTo($attributeName, $value, $delta = 0
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 $attri
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()

制約
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)

11.54 assertTrue()

assertTrue(bool $condition[, string $message = ''])

$condition が false の場合にエラー $message を報告します。

assertNotTrue() はこのアサーションの逆で、同じ引数をとります。

Example 11.60 assertTrue() の使用法

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

class TrueTest extends TestCase
{
    public function testFailure()
    {
        $this->assertTrue(false);
    }
}
?>
```

```
$ phpunit TrueTest
PHPUnit 7.0.0 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.
```

11.55 assertXmlFileEqualsXmlFile()

`assertXmlFileEqualsXmlFile(string $expectedFile, string $actualFile[, string $message = ''])`

`$actualFile` の XML ドキュメントが `$expectedFile` の XML ドキュメントと異なる場合にエラー `$message` を報告します。

`assertXmlFileNotEqualsXmlFile()` はこのアサーションの逆で、同じ引数をとります。

Example 11.61 assertXmlFileEqualsXmlFile() の使用法

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

class XmlFileEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlFileEqualsXmlFile(
          '/home/sb/expected.xml', '/home/sb/actual.xml');
    }
}
?>
```

```
$ phpunit XmlFileEqualsXmlFileTest
PHPUnit 7.0.0 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.
```

11.56 assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])`

`$actualXml` の XML ドキュメントが `$expectedFile` の XML ドキュメントと異なる場合にエラー `$message` を報告します。

`assertXmlStringNotEqualsXmlFile()` はこのアサーションの逆で、同じ引数をとります。

Example 11.62 assertXmlStringEqualsXmlFile() の使用法

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

class XmlStringEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlFile(
          '/home/sb/expected.xml', '<foo><baz/></foo>');
    }
}
?>
```

```
$ phpunit XmlStringEqualsXmlFileTest
PHPUnit 7.0.0 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.
```

11.57 assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])`

`$actualXml` の XML ドキュメントが `$expectedXml` の XML ドキュメントと異なる場合にエラー `$message` を報告します。

`assertXmlStringNotEqualsXmlString()` はこのアサーションの逆で、同じ引数をとります。

Example 11.63 assertXmlStringEqualsXmlString() の使用法

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

class XmlStringEqualsXmlStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlString(
          '<foo><bar/></foo>', '<foo><baz/></foo>');
    }
}
?>
```

```
$ phpunit XmlStringEqualsXmlStringTest
PHPUnit 7.0.0 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.
```

第 12 章

アノテーション

アノテーションとはメタデータを表す特別な構文のことで、プログラミング言語のソースコードに追加することができます。PHP そのものにはソースコードにアノテーションする専用の仕組みはありませんが、ドキュメンテーションブロックに `@`**アノテーション名 引数**のようなタグを書くことでアノテーションを表すという記法が PHP コミュニティ内で一般に使われています。PHP では、リフレクション API の `getDocComment()` メソッドを使えば関数、クラス、メソッド、属性それぞれのドキュメンテーションブロックにアクセスすることができます。PHPUnit などのアプリケーションでは、この情報をもとに実行時の振る舞いを設定するのです。

Note

PHP の doc コメントは、`/**` で始めて`*/` で終わる必要があります。その他の形式のコメントで書いたアノテーションは、無視されます。

本章では、PHPUnit がサポートするすべてのアノテーションについて解説します。

12.1 @author

`@author` アノテーションは`@group` アノテーション (*@group* を参照ください) のエイリアスで、テストの作者にもとづいたフィルタリングができるようになります。

12.2 @after

`@after` アノテーションを使うと、テストケースクラス内の各テストメソッドを実行した後に呼ぶメソッドを指定できます。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
```

```
{
    /**
     * @after
     */
    public function tearDownSomeFixtures()
    {
        // ...
    }

    /**
     * @after
     */
    public function tearDownSomeOtherFixtures()
    {
        // ...
    }
}
```

12.3 @afterClass

@afterClass アノテーションを使うと、テストケースクラス内の各テストメソッドを実行した後に呼ぶ静的メソッドを指定できます。ここで共有フィクスチャの後始末をします。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @afterClass
     */
    public static function tearDownSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @afterClass
     */
    public static function tearDownSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

12.4 @backupGlobals

グローバル変数の保存や復元を、テストケースクラスのすべてのテストで完全に無効にすることができます。このように使います。

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    // ...
}
```

@backupGlobals アノテーションは、テストメソッドレベルで使うこともできます。これによって、保存と復元の操作をより細やかに制御できるようになります。

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testThatInteractsWithGlobalVariables()
    {
        // ...
    }
}
```

12.5 @backupStaticAttributes

@backupStaticAttributes アノテーションを使うと、宣言されたクラス内のすべての static プロパティの値をバックアップしてからテストを始め、テストが終わった後でそれらの値を復元することができます。テストケースクラス単位、あるいはテストメソッド単位で使えます。

```
use PHPUnit\Framework\TestCase;

/**
 * @backupStaticAttributes enabled
 */
```

```
class MyTest extends TestCase
{
    /**
     * @backupStaticAttributes disabled
     */
    public function testThatInteractsWithStaticAttributes()
    {
        // ...
    }
}
```

Note

PHP の内部的な制約のため、@backupStaticAttributes が、意図していない static 値を保存してしまい、その後のテストに影響してしまうことがあります。

詳細は **グローバルな状態** を参照ください。

12.6 @before

@before アノテーションを使うと、テストケースクラス内の各テストメソッドを実行する前に呼ぶメソッドを指定できます。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @before
     */
    public function setupSomeFixtures()
    {
        // ...
    }

    /**
     * @before
     */
    public function setupSomeOtherFixtures()
    {
        // ...
    }
}
```

12.7 @beforeClass

@beforeClass アノテーションを使うと、テストケースクラス内の各テストメソッドを実行する前に呼ぶ静的メソッドを指定できます。ここで共有フィクスチャの準備をします。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @beforeClass
     */
    public static function setUpSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @beforeClass
     */
    public static function setUpSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

12.8 @codeCoverageIgnore*

@codeCoverageIgnore や@codeCoverageIgnoreStart、そして@codeCoverageIgnoreEnd アノテーションを使うと、コード内の特定の行をカバレッジ解析の対象外にできます。

利用法は **コードブロックの無視** を参照ください。

12.9 @covers

@covers アノテーションをテストコードで使うと、そのテストメソッドがどのメソッドをテストするのかを指定することができます。

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

```
}
```

これを指定した場合は、指定したメソッドのみのコードカバレッジ情報を考慮することになります。

Table 12.1 に `@covers` アノテーションの構文を示します。

Table12.1　カバーするメソッドを指定するためのアノテーション

アノテーション	説明
`@covers ClassName::methodName`	そのテストメソッドが指定したメソッドをカバーすることを表します。
`@covers ClassName`	そのテストメソッドが指定したクラスのすべてのメソッドをカバーすることを表します。
`@covers ClassName<extended>`	そのテストメソッドが、指定したクラスとその親クラスおよびインターフェイスのすべてのメソッドをカバーすることを表します。
`@covers ClassName::<public>`	そのテストメソッドが、指定したクラスのすべての public メソッドをカバーすることを表します。
`@covers ClassName::<protected>`	そのテストメソッドが、指定したクラスのすべての protected メソッドをカバーすることを表します。
`@covers ClassName::<private>`	そのテストメソッドが、指定したクラスのすべての private メソッドをカバーすることを表します。
`@covers ClassName::<!public>`	そのテストメソッドが、指定したクラスのすべての非 public メソッドをカバーすることを表します。
`@covers ClassName::<!protected>`	そのテストメソッドが、指定したクラスのすべての非 protected メソッドをカバーすることを表します。
`@covers ClassName::<!private>`	そのテストメソッドが、指定したクラスのすべての非 private メソッドをカバーすることを表します。
`@covers ::functionName`	そのテストメソッドが、指定したグローバル関数をカバーすることを表します。

12.10 @coversDefaultClass

`@coversDefaultClass` アノテーションを使うと、デフォルトの名前空間あるいはクラス名を指定できます。こうすることで、`@covers` アノテーションのたびに長い名前を繰り返す必要がなくなります。Example 12.1 を参照ください。

Example 12.1 @coversDefaultClass を使ったアノテーションの短縮

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

/**
 * @coversDefaultClass \Foo\CoveredClass
 */
class CoversDefaultClassTest extends TestCase
{
    /**
     * @covers ::publicMethod
     */
    public function testSomething()
    {
        $o = new Foo\CoveredClass;
        $o->publicMethod();
    }
}
```

12.11 @coversNothing

@coversNothing アノテーションをテストコードで使うと、そのテストケースについてはコードカバレッジ情報を記録しないように指定できます。

これはインテグレーションテストで使えます。例として**どのメソッドもカバーすべきでないことを指定したテスト**を参照ください。

このメソッドはクラスレベルおよびメソッドレベルで使え、あらゆる @covers タグを上書きします。

12.12 @dataProvider

テストメソッドには任意の引数を渡すことができます。引数は、データプロバイダメソッド (**配列の配列を返すデータプロバイダの使用** の provider()) から渡されます。使用するデータプロバイダメソッドを指定するには@dataProvider アノテーションを使います。

詳細は **データプロバイダ**を参照ください。

12.13 @depends

PHPUnit は、テストメソッド間の依存性の明示的な宣言をサポートしています。この依存性とは、テストメソッドが実行される順序を定義するものではありません。プロデューサーがテストフィクスチャを作ってそのインスタ

ンスを返し、依存するコンシューマーがそれを受け取って利用するというものです。*@depends* **アノテーションを使った依存性の表現**は、@depends アノテーションを使ってテストメソッドの依存性をあらわす例です。

詳細は **テストの依存性**を参照ください。

12.14 @doesNotPerformAssertions

アサーションがひとつもないテストを、リスキーであるとみなさないようにします。

12.15 @expectedException

expectException() **メソッドの使用法**は、テストするコード内で例外がスローされたかどうかを@expectedException アノテーションを使用して調べる方法を示すものです。

詳細は **例外のテスト**を参照ください。

12.16 @expectedExceptionCode

@expectedExceptionCode アノテーションを@expectedException と組み合わせて使うと、スローされた例外のエラーコードについてのアサーションが可能となり、例外をより狭い範囲に特定できるようになります。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException     MyException
     * @expectedExceptionCode 20
     */
    public function testExceptionHasErrorCode20()
    {
        throw new MyException('Some Message', 20);
    }
}
```

テストを実行しやすくし、重複を減らすために、ショートカットを使ってクラス定数を指定することができます。@expectedExceptionCode で“ @expectedExceptionCode ClassName::CONST ”のようにして使います。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
```

```
    /**
      * @expectedException      MyException
      * @expectedExceptionCode MyClass::ERRORCODE
      */
    public function testExceptionHasErrorCode20()
    {
      throw new MyException('Some Message', 20);
    }
}
class MyClass
{
    const ERRORCODE = 20;
}
```

12.17 @expectedExceptionMessage

@expectedExceptionMessage アノテーションは@expectedExceptionCode と似ており、例外のエラーメッセージに関するアサーションを行います。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException        MyException
     * @expectedExceptionMessage Some Message
     */
    public function testExceptionHasRightMessage()
    {
        throw new MyException('Some Message', 20);
    }
}
```

期待するメッセージを、例外メッセージの一部にすることもできます。これは、特定の名前や渡したパラメータが例外に表示されることを確かめたいけれども例外メッセージ全体は固定していない場合に便利です。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException        MyException
      * @expectedExceptionMessage broken
      */
     public function testExceptionHasRightMessage()
     {
```

```
        $param = "broken";
        throw new MyException('Invalid parameter "'.$param.'".', 20);
    }
}
```

テストを実行しやすくし、重複を減らすために、ショートカットを使ってクラス定数を指定することができます。@expectedExceptionMessage で“ @expectedExceptionMessage ClassName::CONST ”のようにして使います。サンプルコードは *@expectedExceptionCode* を参照ください。

12.18 @expectedExceptionMessageRegExp

期待するメッセージを、@expectedExceptionMessageRegExp アノテーションを使って正規表現で指定することもできます。これは、部分文字列だけでは指定したメッセージとのマッチングが不十分なときに便利です。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException               MyException
     * @expectedExceptionMessageRegExp /Argument \d+ can not be an? \w+/
     */
    public function testExceptionHasRightMessage()
    {
        throw new MyException('Argument 2 can not be an integer');
    }
}
```

12.19 @group

あるテストを、ひとつあるいは複数のグループに属するものとすることができます。@group アノテーションをこのように使用します。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
    }
```

```
    /**
     * @group regresssion
     * @group bug2204
     */
    public function testSomethingElse()
    {
    }
}
```

@group アノテーションはテストクラスにも指定できます。指定すると、そのテストクラスのすべてのメソッドにアノテーションが「継承」されます。

特定のグループに属するテストのみを選んで実行するには、コマンドラインのテストランナーの場合は--groupオプションあるいは --exclude-group オプションを指定します。XML 設定ファイルの場合は、それぞれ対応するディレクティブを指定します。

12.20 @large

@large アノテーションは、@group large のエイリアスです。

PHP_Invoker パッケージがインストールされていて strict モードが有効な場合に、large テストは実行時間が 60 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの timeoutForLargeTests 属性で変更できます。

12.21 @medium

@medium アノテーションは@group medium のエイリアスです。medium テストは、@large とマークしたテストに依存してはいけません。

PHP_Invoker パッケージがインストールされていて strict モードが有効な場合に、medium テストは実行時間が 10 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの timeoutForMediumTests 属性で変更できます。

12.22 @preserveGlobalState

テストを別プロセスで実行するときに、PHPUnit は親プロセスのグローバルな状態を保存しようと試みます。親プロセスのすべてのグローバル状態をシリアライズし、子プロセス内で最後にそれをアンシリアライズするのです。しかし、親プロセスのグローバル状態の中にもしシリアライズできないものがあれば、問題が発生します。この問題に対応するために、グローバル状態の保存を無効にすることができます。そのために使うのが @preserveGlobalState アノテーションです。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     * @preserveGlobalState disabled
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

12.23 @requires

@requires アノテーションを使うと、共通の事前条件 (たとえば PHP のバージョンや拡張モジュールのインストール状況) を満たさないときにテストをスキップできます。

条件に指定できる内容やその例については*@requires* **の例用例**を参照ください。

12.24 @runTestsInSeparateProcesses

テストクラス内のすべてのテストケースを、個別の PHP プロセスで実行するように指示します。

```
use PHPUnit\Framework\TestCase;

/**
 * @runTestsInSeparateProcesses
 */
class MyTest extends TestCase
{
    // ...
}
```

注意: デフォルトで、PHPUnit は親プロセスのグローバルな状態を保存しようと試みます。親プロセスのすべてのグローバル状態をシリアライズし、子プロセス内で最後にそれをアンシリアライズするのです。しかし、親プロセスのグローバル状態の中にもしシリアライズできないものがあれば、問題が発生します。この問題に対応するために、グローバル状態の保存を無効にすることができます。この問題の対処法については *@preserveGlobalState* を参照ください。

12.25 @runInSeparateProcess

そのテストを個別の PHP プロセスで実行するように指示します。

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

注意: デフォルトで、PHPUnit は親プロセスのグローバルな状態を保存しようと試みます。親プロセスのすべてのグローバル状態をシリアライズし、子プロセス内で最後にそれをアンシリアライズするのです。しかし、親プロセスのグローバル状態の中にもしシリアライズできないものがあれば、問題が発生します。この問題に対応するために、グローバル状態の保存を無効にすることができます。この問題の対処法については *@preserveGlobalState* を参照ください。

12.26 @small

`@small` アノテーションは`@group small` のエイリアスです。small テストは、`@medium` や `@large` とマークしたテストに依存してはいけません。

`PHP_Invoker` パッケージがインストールされていて strict モードが有効な場合に、small テストは実行時間が 1 秒を超えたら失敗します。このタイムアウト時間は、XML 設定ファイルの `timeoutForSmallTests` 属性で変更できます。

Note

テストの実行時間の制限を有効にするには、`@small`、`@medium`、`@large` のいずれかのアノテーションで明示的に指定する必要があります。

12.27 @test

テストメソッド名の先頭に `test` をつけるかわりに、メソッドのドキュメンテーションブロックで `@test` アノテーションを使ってそのメソッドがテストメソッドであることを指定することができます。

```
/**
 * @test
 */
public function initialBalanceShouldBe0()
{
    $this->assertSame(0, $this->ba->getBalance());
}
```

12.28 @testdox

アジャイルドキュメントを生成する際に使う別の説明を指定します。

@testdox アノテーションは、クラスにもテストメソッドにも指定できます。

```
/**
 * @testdox A bank account
 */
class BankAccountTest extends TestCase
{
    /**
     * @testdox has an initial balance of zero
     */
    public function balanceIsInitiallyZero()
    {
        $this->assertSame(0, $this->ba->getBalance());
    }
}
```

Note

PHPUnit 7.0 より前のバージョンでは、アノテーションのパースにバグがあるため、@testdox アノテーションを指定すると自動的に@test アノテーションも指定したものとみなされます。

12.29 @testWith

テストメソッド @dataProvider とともに使うメソッドを実装するかわりに、@testWith アノテーションを使ってデータセットを定義することができます。

データセットには複数の要素を含めることができます。複数の要素からなるデータセットを定義するには、要素ごとに別の行で定義します。データセットの要素は、JSON の配列形式でなければいけません。

データセットをテストに渡す方法について、詳しくは **データプロバイダ**を参照ください。

```
/**
 * @param string    $input
 * @param int       $expectedLength
 *
 * @testWith        ["test", 4]
 *                  ["longer-string", 13]
 */
public function testStringLength(string $input, int $expectedLength)
{
    $this->assertSame($expectedLength, strlen($input));
}
```

JSON のオブジェクト形式で書いた場合は、連想配列として扱われます。

```
/**
 * @param array     $array
 * @param array     $keys
 *
   * @testWith        [{"day": "monday", "conditions": "sunny"}, ["day", "conditions"]]
 */
public function testArrayKeys($array, $keys)
{
    $this->assertSame($keys, array_keys($array));
}
```

12.30 @ticket

@ticket アノテーションは@group アノテーション (*@group* を参照ください) のエイリアスで、チケット ID にもとづいたテストのフィルタリングができるようになります。

12.31 @uses

@uses アノテーションは、テストから実行されてはいるが、そのテストでカバーするつもりはないコードを指定します。たとえば、コード片をテストするために必要な値オブジェクトなどに使います。

```
/**
 * @covers BankAccount::deposit
 * @uses   Money
 */
public function testMoneyCanBeDepositedInAccount()
{
    // ...
}
```

このアノテーションは、厳密なカバレッジモードで使うと特に有用です。このモードの場合、意図せずカバーしてしまったコードがテストを失敗させてしまうことがあるからです。厳密なカバレッジモードに関する詳細な情報は**意図せぬうちにカバーされているコード**を参照ください。

第 13 章

XML 設定ファイル

13.1 PHPUnit

<phpunit> 要素の属性を使って PHPUnit のコア機能を設定します。

```
<phpunit
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/|version|/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="PHPUnit\TextUI\ResultPrinter"
         <!--printerFile="/path/to/ResultPrinter.php"-->
         processIsolation="false"
         stopOnError="false"
         stopOnFailure="false"
         stopOnIncomplete="false"
         stopOnSkipped="false"
         stopOnRisky="false"
         testSuiteLoaderClass="PHPUnit\Runner\StandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
         timeoutForSmallTests="1"
         timeoutForMediumTests="10"
         timeoutForLargeTests="60"
         verbose="false">
  <!-- ... -->
</phpunit>
```

上の XML 設定ファイルは、TextUI テストランナーをデフォルトの設定で起動します。その詳細は **コマンドラインオプション** で説明します。

その他、コマンドラインからは設定できないオプションもあります。

`convertErrorsToExceptions`

`false` にすると、すべての PHP のエラーを例外に変換するエラーハンドラをインストールしません。

`convertNoticesToExceptions`

`false` にすると、`convertErrorsToExceptions` でインストールしたエラーハンドラが `E_NOTICE` や `E_USER_NOTICE` そして `E_STRICT` を例外に変換しなくなります。

`convertWarningsToExceptions`

`false` にすると、`convertErrorsToExceptions` でインストールしたエラーハンドラが `E_WARNING` や `E_USER_WARNING` を例外に変換しなくなります。

`forceCoversAnnotation`

コードカバレッジの記録を、`@covers` アノテーションを使っている関数だけに限定します。このアノテーションについては*@covers* で説明します。

`timeoutForLargeTests`

テストのサイズによるタイムアウト値を制限する場合に、`@large` とマークされたすべてのテストのタイムアウトをこの属性で設定します。ここで指定した時間内にテストが完了しなかった場合、テストは失敗します。

`timeoutForMediumTests`

テストのサイズによるタイムアウト値を制限する場合に、`@medium` とマークされたすべてのテストのタイムアウトをこの属性で設定します。ここで指定した時間内にテストが完了しなかった場合、テストは失敗します。

`timeoutForSmallTests`

テストのサイズによるタイムアウト値を制限する場合に、`@medium` あるいは `@large` のいずれのマークもついていないすべてのテストのタイムアウトをこの属性で設定します。ここで指定した時間内にテストが完了しなかった場合、テストは失敗します。

13.2 テストスイート

`<testsuites>` 要素とその子要素である`<testsuite>` を使って、テストスイート群やテストケース群の中からテストスイートを構成します。

```
<testsuites>
  <testsuite name="My Test Suite">
    <directory>/path/to/*Test.php files</directory>
    <file>/path/to/MyTest.php</file>
    <exclude>/path/to/exclude</exclude>
  </testsuite>
</testsuites>
```

phpVersion および phpVersionOperator 属性を使うと、必要な PHP のバージョンを指定できます。次の例は、PHP のバージョンが 5.3.0 以降である場合にのみ/path/to/*Test.php と/path/to/MyTest.php を追加します。

```
<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>
```

phpVersionOperator 属性はオプションで、デフォルトは>= です。

13.3 グループ

<groups> 要素とその子要素である<include>、<exclude> および<group> を使って、@group アノテーション (*@group* を参照ください) でマークされたテストグループから実行する (しない) ものを選びます。

```
<groups>
  <include>
    <group>name</group>
  </include>
  <exclude>
    <group>name</group>
  </exclude>
</groups>
```

上の XML 設定ファイルは、 TextUI テストランナーを以下の引数で起動します。

- --group name
- --exclude-group name

13.4 コードカバレッジ対象のファイルのホワイトリスト

<filter> 要素とその子要素を使って、コードカバレッジレポートのホワイトリストを設定します。

```
<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>
```

13.5 ログ出力

<logging> 要素とその子要素である<log> を使って、テストの実行結果のログ出力を設定します。

```
<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>
```

上の XML 設定ファイルは、 TextUI テストランナーを以下の引数で起動します。

- --coverage-html /tmp/report
- --coverage-clover /tmp/coverage.xml
- --coverage-php /tmp/coverage.serialized
- --coverage-text
- > /tmp/logfile.txt
- --log-junit /tmp/logfile.xml
- --testdox-html /tmp/testdox.html
- --testdox-text /tmp/testdox.txt

lowUpperBound、highLowerBound および showUncoveredFiles 属性には、TextUI テストランナーで対応するオプションがありません。

- lowUpperBound: カバー率がこの値に満たないときに、カバー率が“ 低い ”とみなします。
- highLowerBound: カバー率がこの値を超えるときに、カバー率が“ 高い ”とみなします。
- showUncoveredFiles: --coverage-text の出力で、カバレッジ情報だけではなくホワイトリストの全ファイル一覧も表示します。
- showOnlySummary: Show only the summary in --coverage-text output.

13.6 テストリスナー

<listeners> 要素とその子要素である<listener> を使って、テスト実行時にテストリスナーをアタッチします。

```
<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>
```

上の XML 設定は、$listener オブジェクト (以下を参照ください) をテストの実行時にアタッチします。

```
$listener = new MyListener(
    ['Sebastian'],
    22,
    'April',
    19.78,
    null,
    new stdClass
);
```

13.7 TestRunner エクステンションの組み込み

<extensions> 要素とその子要素である <extension> を使ってカスタムの TestRunner エクステンションを組み込みます。

Example 13.1 にエクステンションを組み込む方法を示します。

Example 13.1 TestRunner エクステンションの組み込み

```
<?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>
```

13.8 PHP INI 項目や定数、グローバル変数の設定

<php> 要素とその子要素を使って、PHP の設定や定数、グローバル変数を設定します。また、include_path の先頭にパスを追加することもできます。

```
<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>
```

上の XML 設定は、次の PHP コードに対応します。

```
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';
```

デフォルトでは、既存の環境変数は上書きしません。設定済みの環境変数を上書きしたい場合は `force` 属性を使いましょう。

```
<php>
  <env name="foo" value="bar" force="true"/>
```

第 14 章

参考文献

[Astels2003] David Astels. *Test Driven Development*.

[Beck2002] Kent Beck. *Test Driven Development by Example*.

[Beck2002-ja] Kent Beck. **テスト駆動開発入門**.

[Meszaros2007] Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*.

第 15 章

著作権

```
   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/.

======================================================================
```