PHPUnit マニュアル

Bergmann Sebastian [FAMILY Given]

PHPUnit マニュアル

Bergmann Sebastian [FAMILY Given]

発行日 PHPUnit 4.1 対応版 Updated on 2014-06-16.

目次

図の一覧

表の一覧

例の一覧

第1章 PHPUnit のインストール

注記

以前のバージョンの PHPUnit からのアップグレードをする場合は、「Upgrading from PHPUnit 4.0 to PHPUnit 4.1」を参照ください。

要件

PHPUnit 4.1 は PHP 5.3.3 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 [http://php.net/manual/ja/dom.setup.php]、json [http://php.net/manual/ja/json.installation.php]、pcre [http://php.net/manual/ja/pcre.installation.php]、reflection [http://php.net/manual/ja/reflection.installation.php]、そして spl [http://php.net/manual/ja/spl.installation.php] が必要です。これらは、通常はデフォルトでコンパイルされて有効になっています。中には無効にすることすらできず、常に有効になっているものもあります。

コードカバレッジをサポートするには Xdebug [http://xdebug.org/] 2.1.3 以降が必要です。しかし、最新版の Xdebug を使うことを強く推奨します。また、コードカバレッジ機能を利用するには、tokenizer [http://php.net/manual/ja/tokenizer.installation.php] 拡張モジュールも必要です。コードカバレッジ情報を XML 形式で記録するには、xmlwriter [http://php.net/manual/ja/xmlwriter.installation.php] 拡張モジュールも必要です。

PHPUnit を PHP Archive (PHAR) から使うには、phar [http://php.net/manual/ja/phar.installation.php] 拡張モジュールが必要です。Suhosin [http://suhosin.org/] 拡張モジュールを使っている環境で PHPUnit を PHP Archive (PHAR) から使いたい場合は、`suhosin.executor.include.whitelist = phar` の設定が必要です。

PHPUnit の PHAR で `--self-update` 機能を使うには、openssl [http://php.net/manual/ja/openssl.installation.php] 拡張モジュールが必要です。

PHP Archive (PHAR)

PHPUnit を入手する一番簡単な方法は、PHP Archive (PHAR) [http://php.net/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。

```
chmod +x phpunit.phar
mv phpunit.phar /usr/local/bin/phpunitwget https://phar.phpunit.de/phpunit.phar
chmod +x phpunit.phar
mv phpunit.phar /usr/local/bin/phpunit
```

もちろん、ダウンロードした PHAR をそのまますぐに使ってもかまいません。

```
php phpunit.pharwget https://phar.phpunit.de/phpunit.phar
php phpunit.phar
```

Composer

Composer [http://getcomposer.org/] を使ってプロジェクトの依存関係を管理するには、`phpunit/phpunit` への依存情報をプロジェクトの `composer.json` ファイルに追加しま

す。 次に示すのは最小限の `composer.json` ファイルの例で、 開発時の PHPUnit 4.1 への依存を定義しています。

```
{
    "require-dev": {
        "phpunit/phpunit": "4.1.*"
    }
}
```

システム全体で使えるように Composer でインストールするには、次のようにします。

```
composer global require "phpunit/phpunit=4.1.*"
```

`~/.composer/vendor/bin/` にパスを通すことを忘れないようにしましょう。

オプションのパッケージ

オプションのパッケージとして、これらが使えます。

`PHP_Invoker`

callable をタイムアウトつきで実行するユーティリティクラス。 テストのタイムアウトを厳格に指定するために必要なパッケージ。

このパッケージは、PHPUnit の PHAR 版の中に含まれています。 Composer でインストールするには、`"require-dev"` に次の行を追加します。

```
"phpunit/php-invoker": "*"
```

`DbUnit`

DbUnit の PHP/PHPUnit 向けの移植。データベースとのやりとりをテスト可能にする。

このパッケージは、PHPUnit の PHAR 版の中に含まれています。 Composer でインストールするには、`"require-dev"` に次の行を追加します。

```
"phpunit/dbunit": ">=1.2"
```

`PHPUnit_Selenium`

PHPUnit 用の Selenium RC インテグレーション。

このパッケージは、PHPUnit の PHAR 版の中に含まれています。 Composer でインストールするには、`"require-dev"` に次の行を追加します。

```
"phpunit/phpunit-selenium": ">=1.2"
```

第2章 PHPUnit 用のテストの書き方

例2.1「PHPUnit での配列操作のテスト」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.
4. テストメソッドの中で `assertEquals()` のようなアサーションメソッド (付録A *Assertions* 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。

例2.1 PHPUnit での配列操作のテスト

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

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

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

> 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。
>
> —Martin Fowler

テストの依存性

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 necessary 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。
>
> —Adrian Kuhn et. al.

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。

例2.2「@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。

例2.2 @depends アノテーションを使った依存性の表現

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

        return $stack;
    }

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

        return $stack;
    }

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

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。

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例2.3「テストの依存性の活用」に示します。

例2.3 テストの依存性の活用

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

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit --verbose DependencyFailureTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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。 例2.4「複数の依存性を持つテスト」を参照ください。

例2.4 複数の依存性を持つテスト

```php
<?php
class MultipleDependenciesTest extends PHPUnit_Framework_TestCase
{
    public function testProducerFirst()
    {
        $this->assertTrue(true);
        return 'first';
    }

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

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

...

Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 3 assertions)phpunit --verbose MultipleDependenciesTest
PHPUnit 4.1.0 by Sebastian Bergmann.

...

Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 3 assertions)
```

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 (例2.5「配列の配列を返すデータプロバイダの使用」 の provider()) 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。

例2.5 配列の配列を返すデータプロバイダの使用

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

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:9

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.phpunit DataTest
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:9

FAILURES!
```

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

例2.6 Iterator オブジェクトを返すデータプロバイダの使用

```php
<?php
require 'CsvFileIterator.php';

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

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:11

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.phpunit DataTest
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:11

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

例2.7 CsvFileIterator クラス

```php
<?php
class CsvFileIterator implements Iterator {
    protected $file;
    protected $key = 0;
```

```
    protected $current;

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

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

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

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

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

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

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

@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。 例2.8「同じテストでの @depends と @dataProvider の組み合わせ」を参照ください。

例2.8 同じテストでの @depends と @dataProvider の組み合わせ

```
<?php
class DependencyAndDataProviderComboTest extends PHPUnit_Framework_TestCase
{
    public function provider()
    {
        return array(array('provider1'), array('provider2'));
    }

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

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

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 3.50Mb

There was 1 failure:

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

/home/sb/DependencyAndDataProviderComboTest.php:31

FAILURES!
Tests: 4, Assertions: 4, Failures: 1.
phpunit --verbose DependencyAndDataProviderComboTest
PHPUnit 4.1.0 by Sebastian Bergmann.

...F

Time: 0 seconds, Memory: 3.50Mb

There was 1 failure:

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

/home/sb/DependencyAndDataProviderComboTest.php:31

FAILURES!
```

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

注記

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。

注記

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。

例外のテスト

例2.9「@expectedException アノテーションの使用法」 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。

例2.9 @expectedException アノテーションの使用法

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ExceptionTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
```

```
Expected exception InvalidArgumentException

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

さらに、@expectedExceptionMessage や @expectedExceptionCode を @expectedException と組み合わせて使うと、 例外メッセージや例外コードを 例2.10「@expectedExceptionMessage および @expectedExceptionCode アノテーションの使用法」のようにテストできます。

例2.10 @expectedExceptionMessage および @expectedExceptionCode アノテーションの使用法

```php
<?php
class ExceptionTest extends PHPUnit_Framework_TestCase
{
    /**
     * @expectedException        InvalidArgumentException
     * @expectedExceptionMessage Right Message
     */
    public function testExceptionHasRightMessage()
    {
        throw new InvalidArgumentException('Some Message', 10);
    }

    /**
     * @expectedException     InvalidArgumentException
     * @expectedExceptionCode 20
     */
    public function testExceptionHasRightCode()
    {
        throw new InvalidArgumentException('Some Message', 10);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

FF

Time: 0 seconds, Memory: 3.00Mb

There were 2 failures:

1) ExceptionTest::testExceptionHasRightMessage
Failed asserting that exception message 'Some Message' contains 'Right Message'.


2) ExceptionTest::testExceptionHasRightCode
Failed asserting that expected exception code 20 is equal to 10.


FAILURES!
Tests: 2, Assertions: 4, Failures: 2.phpunit ExceptionTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FF

Time: 0 seconds, Memory: 3.00Mb

There were 2 failures:
```

```
1) ExceptionTest::testExceptionHasRightMessage
Failed asserting that exception message 'Some Message' contains 'Right Message'.

2) ExceptionTest::testExceptionHasRightCode
Failed asserting that expected exception code 20 is equal to 10.

FAILURES!
Tests: 2, Assertions: 4, Failures: 2.
```

@expectedExceptionMessage や @expectedExceptionCode を使ったその他の例が、それぞれ「@expectedExceptionMessage」と「@expectedExceptionCode」にあります。

一方、setExpectedException() メソッドを使用して、発生するであろう例外を指定することもできます。この方法を 例2.11「テスト対象のコードで発生するであろう例外の指定」に示します。

例2.11 テスト対象のコードで発生するであろう例外の指定

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

    public function testExceptionHasRightMessage()
    {
        $this->setExpectedException(
          'InvalidArgumentException', 'Right Message'
        );
        throw new InvalidArgumentException('Some Message', 10);
    }

    public function testExceptionHasRightCode()
    {
        $this->setExpectedException(
          'InvalidArgumentException', 'Right Message', 20
        );
        throw new InvalidArgumentException('The Right Message', 10);
    }
}?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 3.00Mb

There were 3 failures:

1) ExceptionTest::testException
Expected exception InvalidArgumentException


2) ExceptionTest::testExceptionHasRightMessage
Failed asserting that exception message 'Some Message' contains 'Right Message'.


3) ExceptionTest::testExceptionHasRightCode
Failed asserting that expected exception code 20 is equal to 10.
```

```
FAILURES!
Tests: 3, Assertions: 6, Failures: 3.phpunit ExceptionTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FFF

Time: 0 seconds, Memory: 3.00Mb

There were 3 failures:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

2) ExceptionTest::testExceptionHasRightMessage
Failed asserting that exception message 'Some Message' contains 'Right Message'.

3) ExceptionTest::testExceptionHasRightCode
Failed asserting that expected exception code 20 is equal to 10.

FAILURES!
Tests: 3, Assertions: 6, Failures: 3.
```

表2.1「例外のテスト用のメソッド」 は、例外をテストするために用意されているメソッドをまとめたものです。

表2.1 例外のテスト用のメソッド

メソッド	意味
void setExpectedException(string $exceptionName[, string $exceptionMessage = '', integer $exceptionCode = NULL])	期待する $exceptionName、$exceptionMessage および $exceptionCode を設定します。
String getExpectedException()	発生することを期待する例外の名前を返します。

一方、 例2.12「例外をテストするための、別の方法」のような方法で例外をテストすることもできます。

例2.12 例外をテストするための、別の方法

```php
<?php
class ExceptionTest extends PHPUnit_Framework_TestCase {
    public function testException() {
        try {
            // ... 例外が発生するであろうコード ...
        }

        catch (InvalidArgumentException $expected) {
            return;
        }

        $this->fail('期待通りの例外が発生しませんでした。');
    }
}
?>
```

例外が発生するはずの 例2.12「例外をテストするための、別の方法」のコードで例外が発生しなかった場合、それに続く fail() によってテストが終了し、問題を報告します。期待通りに例外が発生すると、 catch ブロックが実行されてテストは正常終了します。

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 例 2.13「@expectedException を用いた、PHP エラーが発生することのテスト」 のように PHP のエラーが発生することをテストできます。

注記

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。

例2.13 @expectedException を用いた、PHP エラーが発生することのテスト

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 1 assertion)phpunit -d error_reporting=2 ExpectedErrorTest
PHPUnit 4.1.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 1 assertion)
```

`PHPUnit_Framework_Error_Notice` および `PHPUnit_Framework_Error_Warning` は、 それぞれ PHP の notice と警告に対応します。

注記

例外をテストするときには可能な限り限定的にしなければいけません。 あまりに一般化されすぎたクラスをテストすると、予期せぬ副作用を引き起こしかねません。 というわけで、 `@expectedException` や `setExpectedException()` を使った `Exception` クラスのテストはできないようにしました。

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。

例2.14 PHP のエラーが発生するコードの返り値のテスト

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

?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)phpunit ErrorSuppressionTest
PHPUnit 4.1.0 by Sebastian Bergmann.

.

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。

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 [http://www.php.net/manual/ja/ref.outcontrol.php] 機能を使用してこの仕組みを提供します。

例2.15「関数やメソッドの出力内容のテスト」 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。

例2.15 関数やメソッドの出力内容のテスト

```
<?php
class OutputTest extends PHPUnit_Framework_TestCase
{
    public function testExpectFooActualFoo()
    {
```

```
        $this->expectOutputString('foo');
        print 'foo';
    }

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

.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.phpunit OutputTest
PHPUnit 4.1.0 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

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

表2.2「テストの出力用のメソッド」 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。

表2.2 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。

メソッド	意味
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。

注記

strict モードでは、出力を発生させるテストは失敗します。

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。

例2.16 配列の比較に失敗したときのエラー出力

```
<?php
class ArrayDiffTest extends PHPUnit_Framework_TestCase
{
    public function testEquality() {
        $this->assertEquals(
            array(1,2,3 ,4,5,6),
            array(1,2,33,4,5,6)
        );
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/ArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit ArrayDiffTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

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

```
There was 1 failure:

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

/home/sb/ArrayDiffTest.php:7

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

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。

例2.17 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
class LongArrayDiffTest extends PHPUnit_Framework_TestCase
{
    public function testEquality() {
        $this->assertEquals(
            array(0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3 ,4,5,6),
            array(0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,33,4,5,6)
        );
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

```
 )


/home/sb/LongArrayDiffTest.php:7

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit LongArrayDiffTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/LongArrayDiffTest.php:7

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

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。

例2.18 緩い比較を使った場合の diff の生成のエッジケース

```php
<?php
class ArrayWeakComparisonTest extends PHPUnit_Framework_TestCase
{
    public function testEquality() {
        $this->assertEquals(
            array(1  ,2,3 ,4,5,6),
            array('1',2,33,4,5,6)
        );
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ArrayWeakComparisonTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 4.1.0 by Sebastian Bergmann.

..

Time: 0 seconds


OK (2 tests, 2 assertions)phpunit ArrayTest
PHPUnit 4.1.0 by Sebastian Bergmann.

..

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 (6章*Strict* モード 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 (7章不完全なテスト・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 (7章不完全なテスト・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 assertEquals() 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。

Command-Line switches

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 4.1.0 by Sebastian Bergmann.

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

```
Code Coverage Options:

  --coverage-clover <file>  Generate code coverage report in Clover XML format.
  --coverage-crap4j <file>  Generate code coverage report in Crap4J XML format.
  --coverage-html <dir>     Generate code coverage report in HTML format.
  --coverage-php <file>     Export PHP_CodeCoverage object to file.
  --coverage-text=<file>    Generate code coverage report in text format.
                            Default: Standard output.
  --coverage-xml <dir>      Generate code coverage report in PHPUnit XML format.

Logging Options:

  --log-junit <file>        Log test execution in JUnit XML format to file.
  --log-tap <file>          Log test execution in TAP format to file.
  --log-json <file>         Log test execution in JSON format.
  --testdox-html <file>     Write agile documentation in HTML format to file.
  --testdox-text <file>     Write agile documentation in Text format to file.

Test Selection Options:

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

Test Execution Options:

  --report-useless-tests    Be strict about tests that do not test anything.
  --strict-coverage         Be strict about unintentionally covered code.
  --disallow-test-output    Be strict about output during tests.
  --enforce-time-limit      Enforce time limit based on test size.
  --strict                  Run tests in strict mode (enables all of the above).

  --process-isolation       Run each test in a separate PHP process.
  --no-globals-backup       Do not backup and restore $GLOBALS for each test.
  --static-backup           Backup and restore static attributes for each test.

  --colors                  Use colors in output.
  --stderr                  Write to STDERR instead of STDOUT.
  --stop-on-error           Stop execution upon first error.
  --stop-on-failure         Stop execution upon first error or failure.
  --stop-on-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.
  -v|--verbose              Output more verbose information.
  --debug                   Display debugging information during test execution.

  --loader <loader>         TestSuiteLoader implementation to use.
  --repeat <times>          Runs the test(s) repeatedly.
  --tap                     Report test execution progress in TAP format.
  --testdox                 Report test execution progress in TestDox format.
  --printer <printer>       TestSuiteListener implementation to use.

Configuration Options:

  --bootstrap <file>        A "bootstrap" PHP file that is run before the tests.
  -c|--configuration <file> Read configuration from XML file.
  --no-configuration        Ignore default configuration file (phpunit.xml).
  --include-path <path(s)>  Prepend PHP's include_path with given path(s).
  -d key[=value]            Sets a php.ini value.
```

```
Miscellaneous Options:

  -h|--help                 Prints this usage information.
  --version                 Prints the version and exits.phpunit --help
PHPUnit 4.1.0 by Sebastian Bergmann.

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

Code Coverage Options:

  --coverage-clover <file>  Generate code coverage report in Clover XML format.
  --coverage-crap4j <file>  Generate code coverage report in Crap4J XML format.
  --coverage-html <dir>     Generate code coverage report in HTML format.
  --coverage-php <file>     Export PHP_CodeCoverage object to file.
  --coverage-text=<file>    Generate code coverage report in text format.
                            Default: Standard output.
  --coverage-xml <dir>      Generate code coverage report in PHPUnit XML format.

Logging Options:

  --log-junit <file>        Log test execution in JUnit XML format to file.
  --log-tap <file>          Log test execution in TAP format to file.
  --log-json <file>         Log test execution in JSON format.
  --testdox-html <file>     Write agile documentation in HTML format to file.
  --testdox-text <file>     Write agile documentation in Text format to file.

Test Selection Options:

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

Test Execution Options:

  --report-useless-tests    Be strict about tests that do not test anything.
  --strict-coverage         Be strict about unintentionally covered code.
  --disallow-test-output    Be strict about output during tests.
  --enforce-time-limit      Enforce time limit based on test size.
  --strict                  Run tests in strict mode (enables all of the above).

  --process-isolation       Run each test in a separate PHP process.
  --no-globals-backup       Do not backup and restore $GLOBALS for each test.
  --static-backup           Backup and restore static attributes for each test.

  --colors                  Use colors in output.
  --stderr                  Write to STDERR instead of STDOUT.
  --stop-on-error           Stop execution upon first error.
  --stop-on-failure         Stop execution upon first error or failure.
  --stop-on-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.
  -v|--verbose              Output more verbose information.
  --debug                   Display debugging information during test execution.

  --loader <loader>         TestSuiteLoader implementation to use.
  --repeat <times>          Runs the test(s) repeatedly.
  --tap                     Report test execution progress in TAP format.
  --testdox                 Report test execution progress in TestDox format.
```

```
  --printer <printer>       TestSuiteListener implementation to use.

Configuration Options:

  --bootstrap <file>        A "bootstrap" PHP file that is run before the tests.
  -c|--configuration <file> Read configuration from XML file.
  --no-configuration        Ignore default configuration file (phpunit.xml).
  --include-path <path(s)>  Prepend PHP's include_path with given path(s).
  -d key[=value]            Sets a php.ini value.

Miscellaneous Options:

  -h|--help                 Prints this usage information.
  --version                 Prints the version and exits.
```

`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 14 章ログ出力 を参照ください。 この機能は、tokenizer 拡張モジュールおよび Xdebug 拡張モジュールがインストールされている場合にのみ使用可能となることに注意しましょう。
`--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 11章コードカバレッジ解析 を参照ください。 この機能は、tokenizer 拡張モジュールおよび Xdebug 拡張モジュールがインストールされている場合にのみ使用可能となることに注意しましょう。
`--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 11章コードカバレッジ解析 を参照ください。 この機能は、tokenizer 拡張モジュールおよび Xdebug 拡張モジュールがインストールされている場合にのみ使用可能となることに注意しましょう。
`--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。 この機能は、tokenizer 拡張モジュールおよび Xdebug 拡張モジュールがインストールされている場合にのみ使用可能となることに注意しましょう。

`--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。 詳細は 14章ログ出力 を参照ください。

この機能は、tokenizer 拡張モジュールおよび Xdebug 拡張モジュールがインストールされている場合にのみ使用可能となることに注意しましょう。

`--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。 詳細は 14章ログ出力 を参照ください。

`--log-tap`

Test Anything Protocol (TAP) [http://testanything.org/] 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。 詳細は 14章ログ出力 を参照ください。

`--log-json`

JSON [http://www.json.org/] フォーマットを使用して、ログファイルを作成します。 詳細は 14章ログ出力 を参照ください。

`--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 詳細は 12章テストのその他の使用法 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　例3.1「名前つきデータセッ

ト」 を参照ください。

例3.1 名前つきデータセット

```
<?php
namespace TestNamespace;

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

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

`/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、例3.2「フィルターパターンの例」を参照ください。

例3.2 フィルターパターンの例

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

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、 例3.3「フィルターのショートカット」を参照ください。

例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` アノテーションは `@group` のエイリアスで、 テストの作者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。
`--report-useless-tests`	何もテストをしないテストについて厳格にチェックします。 詳細は 6章*Strict* モード 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 6章*Strict* モード 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 6章*Strict* モード を参照ください。
`--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 6章*Strict* モード を参照ください。
`--strict`	テストを strict モードで実行します (`--report-useless-tests`、 `--strict-coverage`、`--disallow-test-output`、 そして `--enforce-time-limit` を指定するのと同じ動きです)。詳細は 6章*Strict* モード 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。
--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。
--tap	Test Anything Protocol (TAP) [http://testanything.org/] を使用して、テストの進行状況を報告します。 詳細は 14 章ログ出力 を参照ください。
--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。 詳細は 12章テストのその他の使用法 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 付録C *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。
`--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。

注記

これらのオプションは、引数の後に指定してはいけないことに注意しましょう。

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。

例2.1「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。

例2.2「@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。 例4.1「setUp() を使用して stack フィクスチャを作成する」 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 assertEquals() 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。

例4.1 setUp() を使用して stack フィクスチャを作成する

```
<?php
class StackTest extends PHPUnit_Framework_TestCase
{
    protected $stack;

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

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

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

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

```
    }
}
?>
```

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。

例4.2 利用可能なすべてのテンプレートメソッド

```
<?php
class TemplateMethodsTest extends PHPUnit_Framework_TestCase
{
    public static function setUpBeforeClass()
    {
        fwrite(STDOUT, __METHOD__ . "\n");
    }

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

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

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

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

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

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

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

    protected function onNotSuccessfulTest(Exception $e)
```

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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


Time: 0 seconds, 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.phpunit TemplateMethodsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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

Time: 0 seconds, 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.
```

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。

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。

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。

例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」 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。

例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
class DatabaseTest extends PHPUnit_Framework_TestCase
{
    protected static $dbh;

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

    public static function tearDownAfterClass()
    {
        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 (9章テストダブル 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。

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 [http://googletesting.blogspot.com/2008/05/tott-using-dependancy-injection-to.html]。 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。

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す。 オプションで、この分離をクラスの静的属性まで拡張することもできます。

注記

グローバル変数やクラスの静的属性のバックアップ・リストアの実装には `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、グローバル変数のブラックリストを指定して、 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。

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

    // ...
}
```

注記

`$backupGlobalsBlacklist` 属性をたとえば `setUp()` メソッド内で設定しても効果が及ばないことに注意しましょう。

「`@backupStaticAttributes`」 で説明する `@backupStaticAttributes` アノテーションを使って、 静的属性の保存と復元を制御することができます。また、静的属性のブラック

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。 ブラックリストは、このように指定します。

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

    // ...
}
```

注記

$backupStaticAttributesBlacklist 属性をたとえば setUp() メソッド内で設定しても効果が及ばないことに注意しましょう。

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。

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。

Object_Freezer [http://github.com/sebastianbergmann/php-object-freezer/] 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 `Object` ディレクトリ以下の構造と同じになっています。

```
Object                                Tests
|-- Freezer                           |-- Freezer
|   |-- HashGenerator                 |   |-- HashGenerator
|   |   `-- NonRecursiveSHA1.php      |   |   `-- NonRecursiveSHA1Test.php
|   |-- HashGenerator.php             |   |
|   |-- IdGenerator                   |   |-- IdGenerator
|   |   `-- UUID.php                  |   |   `-- UUIDTest.php
|   |-- IdGenerator.php               |   |
|   |-- LazyProxy.php                 |   |
|   |-- Storage                       |   |-- Storage
|   |   `-- CouchDB.php               |   |   `-- CouchDB
|   |                                 |   |       |-- WithLazyLoadTest.php
|   |                                 |   |       `-- WithoutLazyLoadTest.php
|   |-- Storage.php                   |   |-- StorageTest.php
|   `-- Util.php                      |   `-- UtilTest.php
`-- Freezer.php                       `-- FreezerTest.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してやるだけで、 このライブラリのすべてのテストを実行できます。

```

PHPUnit 4.1.0 by Sebastian Bergmann.

............................................................ 60 / 75
...............

Time: 0 seconds

OK (75 tests, 164 assertions)phpunit Tests
PHPUnit 4.1.0 by Sebastian Bergmann.

............................................................ 60 / 75
...............
```

```
Time: 0 seconds

OK (75 tests, 164 assertions)
```

注記

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/FreezerTest.php` にあるテストケースクラス `Object_FreezerTest` 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 4.1.0 by Sebastian Bergmann.

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

Time: 0 seconds

OK (28 tests, 60 assertions)phpunit Tests/FreezerTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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

Time: 0 seconds

OK (28 tests, 60 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 4.1.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 2 assertions)phpunit --filter testFreezingAnObjectWorks Tests
PHPUnit 4.1.0 by Sebastian Bergmann.

.

Time: 0 seconds

OK (1 test, 2 assertions)
```

注記

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。

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 (付録C *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。 例5.1「XML 設定ファイルを用いたテストスイートの構成」 に、最

小限の例を示します。これは、 `Tests` 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。

例5.1 XML 設定ファイルを用いたテストスイートの構成

```
<phpunit>
  <testsuites>
    <testsuite name="Object_Freezer">
      <directory>Tests</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

どのテストを実行するのかを明示的に指定することができます。

例5.2 XML 設定ファイルを用いたテストスイートの構成

```
<phpunit>
  <testsuites>
    <testsuite name="Object_Freezer">
      <file>Tests/Freezer/HashGenerator/NonRecursiveSHA1Test.php</file>
      <file>Tests/Freezer/IdGenerator/UUIDTest.php</file>
      <file>Tests/Freezer/UtilTest.php</file>
      <file>Tests/FreezerTest.php</file>
      <file>Tests/Freezer/StorageTest.php</file>
      <file>Tests/Freezer/Storage/CouchDB/WithLazyLoadTest.php</file>
      <file>Tests/Freezer/Storage/CouchDB/WithoutLazyLoadTest.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

第6章 Strict モード

PHPUnit can perform additional checks while it executes the tests. In addition to the fine-grained control over the various strict mode checks (see below) you may use the `--strict` commandline option or set `strict="true"` in PHPUnit's XML configuration file to enable all of them.

Useless Tests

PHPUnit can be strict about tests that do not test anything. This check can be enabled by using the `--report-useless-tests` option on the commandline or by setting `beStrictAboutTestsThatDoNotTestAnything="true"` in PHPUnit's XML configuration file.

A test that that does not perform an assertion will be marked as risky when this check is enabled. Expectations on mock objects or annotations such as `@expectedException` count as an assertion.

Unintentionally Covered Code

PHPUnit can be strict about unintentionally covered code. This check can be enabled by using the `--strict-coverage` option on the commandline or by setting `checkForUnintentionallyCoveredCode="true"` in PHPUnit's XML configuration file.

A test that is annotated with `@covers` and executes code that is not listed using a `@covers` or `@uses` annotation will be marked as risky when this check is enabled.

Output During Test Execution

PHPUnit can be strict about output during tests. This check can be enabled by using the `--disallow-test-output` option on the commandline or by setting `beStrictAboutOutputDuringTests="true"` in PHPUnit's XML configuration file.

A test that emits output, for instance by invoking `print` in either the test code or the tested code, will be marked as risky when this check is enabled.

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。 The enforcing of this time limit can be enabled by using the `--enforce-time-limit` option on the commandline or by setting `beStrictAboutTestSize="true"` in PHPUnit's XML configuration file.

`@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。

第7章 不完全なテスト・テストの省略

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。

例7.1「テストに未完成の印をつける」 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。

例7.1 テストに未完成の印をつける

```
<?php
class SampleTest extends PHPUnit_Framework_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 4.1.0 by Sebastian Bergmann.

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.phpunit --verbose SampleTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

表7.1「未完成のテスト用の API」に、テストを未完成扱いにするための API を示します。

表7.1 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。

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。

例7.2「テストを省略する」 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。

例7.2 テストを省略する

```php
<?php
class DatabaseTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        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 4.1.0 by Sebastian Bergmann.

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.phpunit --verbose DatabaseTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

表7.2「テストを省略するための API」に、テストを省略するための API を示します。

表7.2 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。

@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。

表7.3 @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 5.4-dev

型	取り得る値	例	別の例
PHPUnit	PHPUnit のバージョン	@requires PHPUnit 3.6.3	@requires PHPUnit 4.1
OS	PHP_OS [http://php.net/reserved.constants.php#constant.php-os] にマッチする正規表現	@requires OS Linux	@requires OS WIN32|WINNT
function	function_exists [http://php.net/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	@requires extension mysqli	@requires extension curl

例7.3 @requires を使ったテストケースのスキップ

```
<?php
/**
 * @requires extension mysqli
 */
class DatabaseTest extends PHPUnit_Framework_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 データベースのテスト

初級者・中級者向けのユニットテストのサンプルは、 どんな言語を対象としたものであっても、 テストしやすいようなロジックに対してシンプルなテストをしているものばかりです。 データベースを扱う一般的なアプリケーションを考えると、これはまったく現実離れしています。 たとえば Wordpress や TYPO3、あるいは Symfony で Doctrine や Propel などを使い始めるとすぐに、 PHPUnit でのテストがやりづらいことを実感するはずです。 データベースとこれらのライブラリが密結合になっているからです。

きっと日々の業務やプロジェクトでも身に覚えがあることでしょう。 自分の持つ PHPUnit に関する知識を駆使して作業を進めようとしたのに、 こんな問題のせいで行き詰ってしまうことが。

1. テストしたいメソッドがかなり大きめの JOIN 操作を実行し、 データを使って重要な結果を算出している。
2. ひとつのビジネスロジックの中で SELECT、INSERT、UPDATE そして DELETE を組み合わせて実行している。
3. ふたつ以上の (おそらくもっと多い) テーブルから初期データを準備しないと そのメソッドのテストができない。

DBUnit　拡張を使うと、テスト用のデータベースのセットアップを単純化でき、　データベース操作後の内容の検証もすることができます。

データベースのテストに対応しているベンダー

DBUnit が現在サポートしているのは、MySQL および PostgreSQL、Oracle、SQLite です。 Zend Framework [http://framework.zend.com] や Doctrine 2 [http://www.doctrine-project.org] を使うと、IBM DB2 や Microsoft SQL Server のような他のデータベースにもアクセスできます。

データベースのテストの難しさ

ウェブ上にあるユニットテストのサンプルの中にデータベースを扱うものが全く見当たらない理由はなぜか。 それは、データベースを扱うテストは準備するのも保守するのもたいへんだからです。 データベースを使うテストをするには、このようなことに気をつける必要があります。

- データベースのスキーマやテーブル
- テーブルへの、テストで必要となるレコードの追加
- テスト実行後のデータベースの状態の検証
- テスト実行ごとのデータベースの後始末

PDO や MySQLi あるいは OCI8 といったデータベース API はどれも使いにくい上に、 こういった処理を自分で書こうとすると長ったらしくなってしまって面倒です。

テストコードはできる限り簡潔に、そして明確に書かねばなりません。その理由は次のとおりです。

- 製品コードにちょっと手を加えるたびに大量のテストコードを変更する羽目になるのは困る。
- 数ヵ月後に改めて読み直したときにも 読みやすく理解しやすいテストコードであってほしい。

さらに知っておく必要があることは、 データベースは基本的に、自分のコードへのグローバルな入力変数であるということです。 テストスイート内にあるふたつのテストを同じデータベースに対して実行すると、 おそらくデータを複数回再利用することになります。 あるテストが失敗すると それ以降のテストの結果にも影響を及ぼしやすく、テストを進めるのが非常に難しくなります。 先ほど箇条書きでまとめた中の「後始末」こそが、この「データベースがグローバルな入力になる」問題を解決するために重要です。

DbUnit を使うと、 データベースのテストにおけるこれらの問題をシンプルにする助けになります。

PHPUnit では助けようにもどうにもならないことが、 データベースのテストはデータベースを使わないものに比べてとても遅くなるという事実です。 テストの実行時間がどれくらいになるかはデータベースとのやりとりの量に依存しますが、 各テストで使うデータの量を少なめにしておいて 可能な限りはデータベースを使わないテストで済ませるようにすれば、 巨大なテストスイートであっても 1 分未満で実行させるのは容易です。

Doctrine 2 プロジェクト [http://www.doctrine-project.org] がよい例です。 このプロジェクトのテストスイートには現時点で約 1000 件のテストが含まれています。 そのほぼ半数がデータベースを扱うテストですが、 標準的なデスクトップコンピューター上の MySQL を使ってテストスイートを実行しても 15 秒程度でテストが完了します。

データベーステストの四段階

Gerard Meszaros は、著書 xUnit Test Patterns でユニットテストを次の四段階に分類しています。

1. フィクスチャのセットアップ (Setup)
2. テストしたいシステムの実行 (Exercise)
3. 結果の検証 (Verify)
4. 後始末 (Teardown)

> フィクスチャとは?
>
> フィクスチャとは、アプリケーションやデータベースの初期状態のことです。 テストを実行する前に用意します。

データベースをテストするには、少なくとも setup と teardown のときにはテーブルに接続してフィクスチャのクリーンアップや書き込みをしなければなりません。 しかし、データベース拡張には、 データベーステストの四段階を次のようなワークフローに振り向ける十分な理由があります。 このフローは、個々のテストに対して実行します。

1. データベースのクリーンアップ

データベースを扱う最初のテストというのはいつでも存在します。 実際のところ、そのときテーブルにデータが存在するのかどうかはわかりません。 PHPUnit は指定した全テーブルに対して TRUNCATE を実行し、 テーブルの中身を空にします。

2. フィクスチャの準備

その後、PHPUnit はフィクスチャの各行を順次処理し、対応するテーブルに書き込みます。

3–5. テストの実行、結果の検証、そして後始末

データベースをリセットして初期状態を読み込んだら、 実際のテストを PHPUnit が実行します。 テストコードのこの部分は Database Extension の存在を知っている必要はなく、コードに対してなんでもお好みのテストをすることができます。

テストの中で assertDataSetsEqual() という特殊なアサーションを使って検証しているかもしれません。 しかし、この機能は完全なオプションです。 この機能は 「データベースアサーション」 で説明します。

PHPUnit のデータベーステストケースの設定

通常、PHPUnit を使うテストケースでは PHPUnit_Framework_TestCase クラスを継承してこのようにします。

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

テストコードで Database Extension を使う場合は少しだけ複雑になり、 別の抽象テストケースを継承しなければなりません。そして、二つの抽象メソッド getConnection() と getDataSet() を実装します。

```
<?php
class MyGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $pdo = new PDO('sqlite::memory:');
        return $this->createDefaultDBConnection($pdo, ':memory:');
    }

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

getConnection() の実装

クリーンアップとフィクスチャの読み込みの機能を動かすには、 PHPUnit Database Extension からデータベース接続にアクセスできなければなりません。 データベース接続の抽象化には PDO ライブラリを使います。 重要なのは、PHPUnit のデータベース拡張を使うためだけに わざわざアプリケーションを PDO ベースにする必要はないということです。 この接続を使うのは、単にクリーンアップとフィクスチャの準備のためだけです。

先ほどの例では、インメモリの SQLite 接続を作って createDefaultDBConnection メソッドに渡しました。 このメソッドは PDO のインスタンスをラップしたもので、二番目のパラメータ (データベース名) に非常にシンプルなデータベース接続の抽象化レイヤーを渡します。このパラメータの型は PHPUnit_Extensions_Database_DB_IDatabaseConnection です。

「データベース接続の使い方」で、このインターフェイスの API と、その活用法について説明します。

getDataSet() の実装

getDataSet() メソッドで定義するのは、 個々のテストを実行する前のデータベースの初期状態がどうあるべきかということです。 データベースの状態の抽象化は DataSet と DataTable という概念を使って行い、これらをそれぞれ PHPUnit_Extensions_Database_DataSet_IDataSet および PHPUnit_Extensions_Database_DataSet_IDataTable というインターフェイスで表します。次の節でこれらの概念を詳しく説明し、 これをデータベースのテストに使うと何がうれしいのかについても示します。

実装するために最低限知っておくべきことは、 getDataSet() メソッドがコールされるのが setUp() の中で一度だけであり、 ここでフィクスチャのデータセットを取得してデータベースに挿入するということです。 先ほどの例では、ファクトリメソッド createFlatXMLDataSet($filename) を使って XML 形式のデータセットを表しました。

データベーススキーマ (DDL) とは?

PHPUnit は、テストの実行前にデータベーススキーマ (すべてのテーブル、トリガー、シーケンス、ビューを含むもの) ができあがっていることを想定しています。つまり開発者としては、 テストスイートを実行する前にデータベースを正しく準備しておかねばならないということです。

データベースのテストにおけるこの事前条件を満たす方法には、次のようなものがあります。

1. インメモリの SQLite ではなく永続化したデータベースを使うのなら、 最初に一度 phpMyAdmin (MySQL の場合) などのツールでデータベースを用意しておけば、 あとはテストを実行するたびにそれを再利用できます。

2. Doctrine 2 [http://www.doctrine-project.org] や Propel [http://www.propelorm.org/] といったライブラリを使っている場合は、その API を使えばテストの実行前に必要なデータベーススキーマを作ることができます。 PHPUnit のブートストラップ [http://www.phpunit.de/manual/current/en/textui.html] 機能を使うと、そのコードをテスト実行時に毎回実行させることもできます。

ヒント: 自前でのデータベーステストケースの抽象化

先の実装例を見ればすぐにわかるでしょうが、 getConnection() メソッドはきわめて静的なものであり、 さまざまなデータベーステストケースで再利用することができます。 さらに、テストのパフォーマンスを良好に保ちつつデータベースのオーバーヘッドを下げるために、 ちょっとしたリファクタリングを施して汎用的な抽象テストケースを用意しましょう。 このようにしても、テストケースごとに異なるデータフィクスチャを指定することができます。

```
<?php
abstract class MyApp_Tests_DatabaseTestCase extends PHPUnit_Extensions_Database_TestCase
{
    // PDO のインスタンス生成は、クリーンアップおよびフィクスチャ読み込みのときに一度だけ
    static private $pdo = null;

    // PHPUnit_Extensions_Database_DB_IDatabaseConnection のインスタンス生成は、テストご    p

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

```
            }
            $this->conn = $this->createDefaultDBConnection(self::$pdo, ':memory:');
        }

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

しかし、これはまだデータベースへの接続情報を PDO 接続の設定にハードコードしてしまっています。 PHPUnit にはさらにすばらしい機能があるので、それを使ってテストケースをより汎用的にしましょう。 XML 設定ファイル [http://www.phpunit.de/manual/current/en/appendixes.configuration.html#appendixes.configuration.php-ini-constants-variables] を使えば、テストの実行のたびにデータベース接続を設定できます。 まずは「phpunit.xml」というファイルをアプリケーションの tests/ ディレクトリに作り、 中身をこのようにします。

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

テストケースはこのように書き直せます。

```
<?php
abstract class Generic_Tests_DatabaseTestCase extends PHPUnit_Extensions_Database_TestCa
{
    // PDO のインスタンス生成は、クリーンアップおよびフィクスチャ読み込みのときに一度だけ
    static private $pdo = null;

    // PHPUnit_Extensions_Database_DB_IDatabaseConnection のインスタンス生成は、テストごとに一
    private $conn = null;

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

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

データベースの設定情報を切り替えてテストスイートを実行するには、 コマンドラインから次のようにします。

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

データベースのテストを実行するときにターゲットデータベースを切り替えられるようにしておくことは、 開発機で作業をしている場合などは特に重要です。 複数の開発者が同じデータベース接続を使ってデータベースのテストを実行したりすると、 レースコンディション (競合条件) によるテストの失敗が頻発するでしょう。

データセットとデータテーブルについて知る

PHPUnit Database Extension の中心となる概念が データセットとデータテーブルです。まずはこの考え方を理解することが、 PHPUnit でのデータベースのテストをマスターする近道です。 データセットとデータテーブルは、データベースのテーブルや行、 そしてカラムの抽象化レイヤーです。シンプルな API によってデータベースの内容をオブジェクト構造に隠蔽できるだけでなく、 データベース以外のソースによる実装もできるようになっています。

この抽象化を使って、データベースの実際の中身と我々が期待する内容を比較します。 期待する内容は XML や YAML そして CSV などのファイルでも表せますし、 PHP の配列として表すこともできます。 DataSet インターフェイスと DataTable インターフェイスのおかげで、 これらの全く異なる概念のソースをリレーショナルデータベースに見立てて同様に扱えるようになります。

データベースのアサーションをテストの中で行う流れは、 次のようにシンプルな三段階となります。

- ひとつあるいは複数のテーブルをデータベース内から指定する (実際のデータセット)。
- 期待するデータセットをお好みのフォーマット (YAML, XML など) で用意する。
- 両者がお互いに等しいことを確認する。

データセットやデータテーブルの PHPUnit Database Extension における使い道は、 何もアサーションだけだというわけではありません。先ほどの節で見たように、 これらを使ってデータベースの初期状態の内容を記述することもできます。 フィクスチャとなるデータセットを Database TestCase で定義すると、それをこのように使うことができます。

- データセットで指定したテーブルのすべての行を削除する。
- データテーブルのすべての行をデータベースに書き込む。

利用できる実装

これら三種類のデータセット/データテーブルが用意されています。

- ファイルベースのデータセットやデータテーブル
- クエリベースのデータセットやデータテーブル
- フィルタ用や合成用のデータセットやデータテーブル

ファイルベースのデータセットやデータテーブルは、 初期状態のフィクスチャを定義したり期待する状態を定義したりするときによく使います。

フラット XML データセット

最も一般的なデータセットは、フラット XML と呼ばれるものです。 これは非常にシンプルな xml 形式で、ルートノード `<dataset>` の中のタグがデータベースのひとつの行を表します。 テーブルと同じ名前のタグが追加する行を表し、 その属性がカラムを表します。 単純な掲示板アプリケーションの例は、このようになります。

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

見るからに書きやすそうですね。この場合は <guestbook> がテーブル名で、 2 行が追加されます。そして、四つのカラム 「id」、 「content」、「user」 そして 「created」 に、それぞれ対応する値が設定されています。

しかし、この単純性による問題もあります。

たとえば、先ほどの例で空のテーブルはどうやって指定すればいいのかがよくわかりません。 実は、何も属性を指定せずにテーブルと同じ名前のタグを追加すれば、空のテーブルを表すことができます。 空の guestbook テーブルを表すフラット xml ファイルは、このようになります。

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

フラット xml データセットでの NULL 値の処理は、あまりおもしろいものではありません。 ほとんどのデータベースでは、NULL 値と空文字列は別のものとして扱います (例外のひとつは Oracle です) が、これをフラット xml 形式で表すのは困難です。NULL 値を表すには、 行の指定のときに属性を省略します。 この例の掲示板で、匿名の投稿を許可し、そのときには user カラムに NULL を指定することにしましょう。 guestbook テーブルの状態は、このようになります。

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

この例では、二番目のエントリが匿名の投稿を表します。 しかし、これはカラムの認識において深刻な問題につながります。 データセットが等しいことを確認するアサーションでは、各データセットでテーブルの持つカラムを指定しなければなりません。 ある属性がデータテーブルのすべての行で NULL だったなら、 Database Extension はそのカラムがテーブルに存在することをどうやって知るというのでしょう?

フラット XML データセットはここで、重大な前提を使っています。 テーブルの最初の行で定義されている属性が、そのテーブルのカラムを定義しているものと見なすのです。 先ほどの例では、guestbook テーブルのカラムが 「id」 と 「content」、「user」 そして 「created」 であると見なすということです。二番目の行には 「user」 が定義されていないので、データベースには NULL を挿入します。

guestbook の最初のエントリをデータセットから削除すると、guestbook テーブルのカラムは 「id」、「content」 そして 「created」 だけになってしまいます。 「user」 が指定されていないからです。

フラット XML データセットを効率的に使うには、NULL 値がからむ場合は 各テーブルの最初の行には NULL を含まないようにします。 それ以降の行では、属性を省略して NULL を表すことができます。 これはあまりスマートなやり方ではありません。 というのも、データベースのアサーションで行の順番が影響してしまうからです。

一方、テーブルのカラムの一部だけをフラット XML データセットで指定すると、 それ以外のカラムにはデフォルト値が設定されます。 そのため、もし省略したカラムの定義が 「NOT NULL DEFAULT NULL」 などの場合はエラーになります。

結論として言えるのは、フラット XML データセットを使うなら NULL 値が不要な場合だけにしておいたほうがよい、ということだけです。

フラット XML データセットのインスタンスを Database TestCase から作るには、`createFlatXmlDataSet($filename)` メソッドを使います。

```
<?php
```

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

XML データセット

もうひとつ別の構造の XML データセットもあります。これは多少冗長な書き方ですが、フラット XML データセットにおける NULL の問題は発生しません。 ルートノード <dataset> の配下に指定できるタグは、 <table> や <column>、<row>、 <value> そして <null /> です。 先に定義した Guestbook のフラット XML と同様のデータセットは、このようになります。

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

<table> には name が必須で、 さらにすべてのカラムの名前を定義しなければなりません。 また、ゼロ個以上の <row> 要素を含めることができます。<row> 要素を定義しなければ、そのテーブルが空であることになります。 <value> タグや <null /> タグは、先に指定した column> 要素の順番で指定しなければなりません。 <null /> タグは、 見た目の通り、値が NULL であることを表します。

XML データセットのインスタンスを Database TestCase から作るには、 createXmlDataSet($filename) メソッドを使います。

```
<?php
class MyTestCase extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        return $this->createXMLDataSet('myXmlFixture.xml');
    }
}
?>
```

MySQL XML データセット

この新しい XML フォーマットは、 MySQL データベース [http://www.mysql.com] 専用です。 PHPUnit 3.5 以降で対応します。この形式のファイルを生成するには、 mysqldump

[http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html] を使います。mysqldump では CSV データセットも対応していますが、 それとは違ってこの XML 形式の場合はひとつのファイルに複数のテーブルを含めることができます。 この形式のファイルを作るには、mysqldump を次のように実行します。

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

このファイルを Database TestCase で使うには、 createMySQLXMLDataSet($filename) メソッドをコールします。

```
<?php
class MyTestCase extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        return $this->createMySQLXMLDataSet('/path/to/file.xml');
    }
}
?>
```

YAML データセット

あるいは、YAML データセットを使って、guestbook の例をこのように表すこともできます。

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

これは、シンプルで便利なうえに、さらにフラット XML データセットが持つ NULL の問題も解決しています。 NULL を YAML で表すには、単にカラム名の後に何も値を指定しなければよいのです。 空文字列を指定する場合は column1: "" のようにします。

YAML Dataset 用のファクトリーメソッドは今のところ Database TestCase に存在しないので、手動でインスタンスを生成しなければなりません。

```
<?php
class YamlGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        return new PHPUnit_Extensions_Database_DataSet_YamlDataSet(
            dirname(__FILE__)."/_files/guestbook.yml"
        );
    }
}
?>
```

CSV データセット

さらにもうひとつのファイルベースのデータセットとして、CSV ファイルを使ったものもあります。データセット内の各テーブルを、 それぞれ単一の CSV ファイルとして扱います。 guestbook の例では、このようなファイル guestbook-table.csv を定義します。

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

この形式は Excel や OpenOffice で編集できるという点で非常に便利ですが、 CSV データセットでは NULL 値を指定することができません。 空のカラムは、データベースのデフォルトに基づいた空の値として扱われます。

CSV データセットを作るには、このようにします。

```
<?php
class CsvGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        $dataSet = new PHPUnit_Extensions_Database_DataSet_CsvDataSet();
        $dataSet->addTable('guestbook', dirname(__FILE__)."/_files/guestbook.csv");
        return $dataSet;
    }
}
?>
```

Array データセット

PHPUnit の Database Extension には、(今のところ) 配列ベースのデータセットが存在しません。 しかし、自分で簡単に実装できます。 guestbook の例だと、このようになります。

```
<?php
class ArrayGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    protected function getDataSet()
    {
        return new MyApp_DbUnit_ArrayDataSet(array(
            'guestbook' => array(
                array('id' => 1, 'content' => 'Hello buddy!', 'user' => 'joe', 'created'
                array('id' => 2, 'content' => 'I like it!',   'user' => null,  'created'
            ),
        ));
    }
}
?>
```

PHP の DataSet には、これまでのファイルベースのデータセットに比べて明らかな利点があります。

- PHP の配列は NULL 値を扱える。
- アサーション用に新たなファイルを用意する必要がなく、 直接テストケース内で指定できる。

このデータセットでは、フラット XML や CSV そして YAML データセットと同様に、最初に指定した行のキーがテーブルのカラム名を表します。 つまり、先ほどの例だと「id」、「content」、「user」そして「created」です。

この Array データセットの実装は、シンプルで直感的なものです。

```
<?php
class MyApp_DbUnit_ArrayDataSet extends PHPUnit_Extensions_Database_DataSet_AbstractData
{
    /**
```

```
     * @var array
     */
    protected $tables = array();

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

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

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

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

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

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

Query (SQL) データセット

データベースのアサーションでは、ファイルベースのデータセットだけでなく Query/SQL ベースのデータセットでデータベースの実際の中身を含むものが必要になることもあります。 そんなときに使えるのが Query データセットです。

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

単にテーブル名だけを指定してテーブルを追加すると、 次のクエリを実行してデータテーブルを定義したのと同じ意味になります。

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

ここでテーブルに対して任意のクエリを実行して、 取得する行や列を絞り込んだり ORDER BY 句を追加したりすることができます。

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

データベースアサーションの節で、このデータセットを使う方法をより詳しく説明しています。

Database (DB) データセット

テスト用のデータベース接続にアクセスすると、 自動的にすべてのテーブルとその中身を含むデータセットを生成します。 接続先のデータベースは、接続用のファクトリーメソッドの二番目のパラメータで指定します。

データベース全体の完全なデータセットを作るには `testGuestbook()` のようにします。 ホワイトリスト形式で指定したテーブルだけに絞り込むには `testFilteredGuestbook()` メソッドのようにします。

```
<?php
class MySqlGuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $database = 'my_database';
        $pdo = new PDO('mysql:...', $user, $password);
        return $this->createDefaultDBConnection($pdo, $database);
    }

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

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

Replacement データセット

これまで、フラット XML や CSV のデータセットには NULL の問題があると説明してきました。 しかし、ちょっとわかりにくい回避策を使えばこれらのデータセットで NULL を扱うこともできます。

Replacement データセットは既存のデータセットに対するデコレータで、 データセットの任意のカラムの値を別の値で置換することができます。 guestbook の例で NULL 値を扱うには、このようなファイルを作ります。

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

そして、フラット XML データセットを Replacement データセットでラップします。

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

データセットフィルタ

巨大なフィクスチャファイルを扱うときには、 データセットフィルタをホワイトリストあるいはブラックリストとして使って テーブルやカラムを絞り込んだサブデータセットを作ることができます。 これは、DB データセットと組み合わせて データセットのカラムを絞り込むときに使うと非常に便利です。

```
<?php
class DataSetFilterTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testIncludeFilteredGuestbook()
    {
        $tableNames = array('guestbook');
        $dataSet = $this->getConnection()->createDataSet();

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

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

        $filterDataSet = new PHPUnit_Extensions_Database_DataSet_DataSetFilter($dataSet)
        $filterDataSet->addExcludeTables(array('foo', 'bar', 'baz')); // only keep the g
        $filterDataSet->setExcludeColumnsForTable('guestbook', array('user', 'created'))
        // ..
    }
}
?>
```

注意 ひとつのテーブルに対してカラムの exclude フィルタと include フィルタを同時に使うことはできません。 さらに、テーブルのホワイトリストとブラックリストはどちらか一方しか指定できません。

Composite データセット

Composite データセットは、既存の複数のデータセットをひとつにまとめるときに有用です。 複数のデータセットに同名のテーブルが含まれる場合は、 指定した順で行を連結します。 たとえば、このようなふたつのデータセットがあるものとしましょう。 まずは *fixture1.xml*。

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

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

そして *fixture2.xml*。

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

Composite データセットを使えば、両方のフィクスチャファイルをまとめることができます。

```
<?php
class CompositeTest extends PHPUnit_Extensions_Database_TestCase
{
    public function getDataSet()
    {
        $ds1 = $this->createFlatXmlDataSet('fixture1.xml');
        $ds2 = $this->createFlatXmlDataSet('fixture2.xml');

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

        return $compositeDs;
    }
}
?>
```

外部キーには注意

フィクスチャを準備するとき、PHPUnit の Database Extension はフィクスチャ内で定義された順に行を追加していきます。 データベースのスキーマ定義で外部キーを使っている場合は、 外部キー制約に違反しないような順番でテーブルを指定しなければなりません。

自作のデータセットやデータテーブルの実装

データセットやデータテーブルの内部構造を理解するために、 まずはデータセットのインターフェイスから見ていきましょう。 自分でデータセットやデータテーブルを作るつもりのない人は、 読み飛ばしてもかまいません。

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

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

公開インターフェイスは、データベーステストケースの `assertDataSetsEqual()` アサーションで内部的に使われており、これでデータセットの内容を検証します。 IDataSet は `IteratorAggregate` インターフェイスから `getIterator()` メソッドを継承しており、これを使ってデータセット内の全テーブルの反復処理を行います。 リバースイテレー

タを使うと、PHPUnit で作ったテーブルのデータの切り詰めを、 テーブルを作ったときと逆の順番で行えます。これで、外部キー制約に違反せずに済むようになります。

テーブルのインスタンスをデータセットに追加するには、 実装によってさまざまな手法があります。たとえば YamlDataSet や XmlDataSet そして FlatXmlDataSet のようなファイルベースのデータセットでは、 データセットの作成時にソースファイルを使って内部的に追加します。

テーブルは、このようなインターフェイスを使って表します。

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

getTableMetaData() メソッドは別として、 それ以外のメソッドはまさに文字通りの働きをするものです。 これらのメソッドはすべて、Database Extension のさまざまなアサーションで必須となります。 その詳細は次の章で説明します。 getTableMetaData() メソッドの返す値は、 PHPUnit_Extensions_Database_DataSet_ITableMetaData 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。 このインターフェイスはテーブルの構造を表し、このような情報を保持します。

- テーブル名。
- テーブルのカラム名の配列。並び順は、結果セットに登場する順と同じ。
- 主キーカラムの配列。

このインターフェイスには、ふたつの TableMetaData のインスタンスがお互いに等しいかを調べるアサーションも定義されています。 これは、データセットの同一性を調べるアサーションで利用するものです。

接続 API

Connection インターフェイスには、三種類のおもしろいメソッドが用意されています。 このインターフェイスは、データベーステストケースの getConnection() メソッドが返すものです。

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

    // ...
}
?>
```

1. createDataSet() メソッドは、Database (DB) データセットを作ります。これは、データセットの実装の節で説明したものです。

```
<?php
class ConnectionTest extends PHPUnit_Extensions_Database_TestCase
{
```

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

2. `createQueryTable()` メソッドを使うと、 QuryTable のインスタンスを作れます。引数には、結果の名前と SQL クエリを渡します。 これは、次の節 (データベースアサーション API) で説明する結果やテーブルのアサーションで有用なメソッドです。

```
<?php
class ConnectionTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testCreateQueryTable()
    {
        $tableNames = array('guestbook');
        $queryTable = $this->getConnection()->createQueryTable('guestbook', 'SELECT *
    }
}
?>
```

3. `getRowCount()` は、 テーブル内の行数を手軽に取得するためのメソッドです。 オプションで、where 句によるフィルタリングもできます。 これを使えば、シンプルな同一性のアサーションが可能です。

```
<?php
class ConnectionTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testGetRowCount()
    {
        $this->assertEquals(2, $this->getConnection()->getRowCount('guestbook'));
    }
}
?>
```

データベースアサーション API

テストツール用として、Database Extension ではいくつかのアサーションを提供しています。 これらを使えば、データベースやテーブルの現在の状態 そしてテーブルの行数を検証できます。この節では、 これらの機能の詳細を説明します。

テーブルの行数のアサーション

テーブルの行数が特定の値であるかどうかを調べられれば便利なことがよくあります。 これは、接続 API を使ってちょっとしたコードを書かなくとも簡単に実現できます。 guestbook に行を追加した後で、初期登録した 2 エントリ以外にもう一行増えて 3 行になっていることを調べるには、このようにします。

```
<?php
class GuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testAddEntry()
    {
        $this->assertEquals(2, $this->getConnection()->getRowCount('guestbook'), "Pre-Co

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

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

テーブルの状態のアサーション

先ほどのアサーションも有用ですが、本当にチェックしたいのは、 すべての値が正しいカラムにきちんと登録されたかどうかです。 これは、テーブルのアサーションで実現します。

そのために、QueryTable のインスタンスを定義しました。 テーブル名と SQL クエリからその内容を取得し、 それをファイルベースあるいは配列ベースのデータセットと比較します。

```
<?php
class GuestbookTest extends PHPUnit_Extensions_Database_TestCase
{
    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);
    }
}
?>
```

さて次に、このアサーションに使うフラット XML ファイル *expectedBook.xml* を用意しましょう。

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

残念ながら、このアサーションが成功するのは、ちょうど *2010–05–01 21:47:08* に実行したときだけになります。 日付はデータベースのテストでいつも問題になるものなので、それを回避する手段として「created」カラムをアサーションで無視させることができます。

調整後のフラット XML ファイル *expectedBook.xml* はこのようになり、これでアサーションを通過させることができます。

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

QueryTable の呼び出しも修正しなければなりません。

```
<?php
```

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

クエリの結果のアサーション

複雑なクエリの結果に対するアサーションも、 QueryTable 方式で可能です。単に結果の名前とクエリを指定して、 それをデータセットと比較すればよいのです。

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

複数のテーブルの状態のアサーション

もちろん、複数のテーブルの状態を一度に確かめたり クエリデータセットをファイルベースのデータセットと比較したりすることも可能です。 データセットのアサーションには二通りの方法があります。

1. 接続の Database (DB) データセットを使い、 それをファイルベースのデータセットと比較する。

```
<?php
class DataSetAssertionsTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testCreateDataSetAssertion()
    {
        $dataSet = $this->getConnection()->createDataSet(array('guestbook'));
        $expectedDataSet = $this->createFlatXmlDataSet('guestbook.xml');
        $this->assertDataSetsEqual($expectedDataSet, $dataSet);
    }
}
?>
```

2. データセットを自分で作ることもできます。

```
<?php
class DataSetAssertionsTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testManualDataSetAssertion()
    {
        $dataSet = new PHPUnit_Extensions_Database_DataSet_QueryDataSet();
        $dataSet->addTable('guestbook', 'SELECT id, content, user FROM guestbook'); //
        $expectedDataSet = $this->createFlatXmlDataSet('guestbook.xml');

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

よくある質問

PHPUnit は、テストごとにデータベーススキーマを作り直すの?

いいえ。PHPUnit は、テストスイートの開始時にすべてのデータベースオブジェクトが存在することを前提とします。 データベースやテーブル、シーケンス、トリガー、そしてビューなどは、 テストスイートを実行する前に作っておく必要があります。

Doctrine 2 [http://www.doctrine-project.org] や eZ Components [http://www.ezcomponents.org] の強力なツールを使えば、定義済みのデータ構造からデータベーススキーマを作成できます。 しかし、これらを使うには PHPUnit extension にフックで組み込まねばなりません。 そうしないと、テストスイートを実行する前にデータベースの自動再作成ができなくなります。

各テストの実行後はデータベースをクリアするので、 テストを実行するたびにデータベースを再作成する必要はありません。 事前に作ったデータベースをずっと使いまわすことができます。

PDO を使ったアプリケーションじゃないと Database Extension を使えないの?

いいえ。PDO が必要なのは、フィクスチャの準備や後始末とアサーションのときだけです。 テスト対象のコード内では、なんでもお好みの方法でデータベースにアクセスできます。

「Too much Connections」 というエラーが出たらどうすればいい?

テストケースの `getConnection()` メソッドで作った PDO インスタンスをキャッシュしていなければ、 データベースを使うテストを実行するたびにデータベースへの接続の数は増加し続けます。 デフォルトの設定では MySQL が受け付ける同時接続は 100 までであり、 他のデータベースにも同様の接続数制限があります。

「自前でのデータベーステストケースの抽象化」 に、このエラーを回避する方法を示しています。 ひとつの PDO インスタンスをキャッシュして、すべてのテストで使いまわす方法です。

フラット XML や CSV のデータセットで NULL を扱う方法は?

そんな方法はありません。NULL が使いたければ XML あるいは YAML データセットを使わないといけません。

第9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。

> 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。
>
> —Gerard Meszaros

PHPUnit の `getMock($className)` メソッドを使うと、 指定した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、元クラス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。

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` を返すだけの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。

警告

パラメータの管理方法が変わったことに気をつけましょう。 以前の実装ではオブジェクトのすべてのパラメータをクローンしており、 あるメソッドに渡されたオブジェクトが同じものであるかどうかを確かめることができませんでした。 例9.15「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」 に、新しい実装の活用例を示します。 例9.16「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」 に、以前の挙動に戻す方法を示します。

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。 」

例9.2「メソッドに固定値を返させるスタブ」 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 getMock() 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 (例9.1「スタブを作りたいクラス」)。 次に、PHPUnit が提供する、いわゆる Fluent Interface [http://martinfowler.com/bliki/FluentInterface.html] (流れるようなインターフェイス [http://capsctrl.que.jp/kdmsnr/wiki/bliki/?FluentInterface]) 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。

例9.1 スタブを作りたいクラス

```
<?php
class SomeClass
{
    public function doSomething()
    {
        // なにかをします
    }
}
?>
```

例9.2 メソッドに固定値を返させるスタブ

```
<?php
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

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

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

舞台裏では、getMock() 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。 生成されるテストダブルクラスの設定は、 getMock() メソッドのオプションの引数を使って行います。

- デフォルトでは、指定したクラスのすべてのメソッドが単に NULL を返すだけのテストダブルとなります。返り値を変更するには、たとえば will($this->returnValue()) を使います。
- オプションの第二パラメータを指定すると、その配列の中に含まれる名前のメソッドだけがテストダブルに置き換えらて、その他のメソッドはそのままとなります。パラメータに NULL を渡すと、どのメソッドも置き換えません。

- オプションの第三パラメータには、元クラスのコンストラクタに渡すパラメータの配列を渡します (デフォルトでは、コンストラクタはダミー実装に置き換えられません)。
- オプションの第四パラメータを使うと、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。
- オプションの第五パラメータを使うと、元クラスのコンストラクタを呼び出さないようにすることができます。
- オプションの第六パラメータを使うと、元クラスの clone コンストラクタを呼び出さないようにすることができます。
- オプションの第七パラメータを使うと、テストダブルクラスの生成時に `__autoload()` を無効にすることができます。

もうひとつのやり方として、生成されたテストダブルクラスの設定を モックビルダー API で行うことができます。 例9.3「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」 に例を示します。 モックビルダーで使え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。
- `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。

例9.3 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
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMockBuilder('SomeClass')
                     ->disableOriginalConstructor()
                     ->getMock();

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

        // $stub->doSomething() をコールすると
```

```
        // 'foo' を返すようになります
        $this->assertEquals('foo', $stub->doSomething());
    }
}
?>
```

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。 例9.4「メソッドに引数のひとつを返させるスタブ」は、 `returnValue()` のかわりに `returnArgument()` を用いてこれを実現する例です。

例9.4 メソッドに引数のひとつを返させるスタブ

```
<?php
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnArgumentStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

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

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

        // $stub->doSomething('bar') は 'bar' を返します
        $this->assertEquals('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。 例9.5「スタブオブジェクトへの参照を返すメソッドのスタブ」は、 `returnSelf()` を使ってこれを実現する例です。

例9.5 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
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnSelf()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

        // スタブの設定を行います
        $stub->expects($this->any())
             ->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。 例9.6「メソッドにマップからの値を返させるスタブ」を参照ください。

例9.6 メソッドにマップからの値を返させるスタブ

```
<?php
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnValueMapStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

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

        // スタブの設定を行います
        $stub->expects($this->any())
             ->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->assertEquals('d', $stub->doSomething('a', 'b', 'c'));
        $this->assertEquals('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。 例9.7「メソッドにコールバックからの値を返させるスタブ」を参照ください。

例9.7 メソッドにコールバックからの値を返させるスタブ

```
<?php
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testReturnCallbackStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

        // スタブの設定を行います
        $stub->expects($this->any())
             ->method('doSomething')
             ->will($this->returnCallback('str_rot13'));

        // $stub->doSomething($argument) は str_rot13($argument) を返します
```

```
        $this->assertEquals('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。 例9.8「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」 の例を参照ください。

例9.8 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
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testOnConsecutiveCallsStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

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

        // $stub->doSomething() は毎回異なる値を返します
        $this->assertEquals(2, $stub->doSomething());
        $this->assertEquals(3, $stub->doSomething());
        $this->assertEquals(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。 例9.9「メソッドに例外をスローさせるスタブ」 に、throwException() でこれを行う方法を示します。

例9.9 メソッドに例外をスローさせるスタブ

```
<?php
require_once 'SomeClass.php';

class StubTest extends PHPUnit_Framework_TestCase
{
    public function testThrowExceptionStub()
    {
        // SomeClass クラスのスタブを作成します
        $stub = $this->getMock('SomeClass');

        // スタブの設定を行います
        $stub->expects($this->any())
             ->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 (façade : ファサード) 経由でアクセスするようにすることで、 それを簡単に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。

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。

制限

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。 例9.10「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」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。

例9.10 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
class Subject
{
    protected $observers = array();
    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
}
?>
```

例9.11「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」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 getMock() 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。 getMock() メソッドの二番目の (オプションの) パラメータに配列を指定しているので、Observer クラスの中の update() メソッドについてのみモック実装が作成されます。

例9.11 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
class SubjectTest extends PHPUnit_Framework_TestCase
{
    public function testObserversAreUpdated()
    {
        // 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->getMock('Observer', array('update'));

        // 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。

例9.12 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
class SubjectTest extends PHPUnit_Framework_TestCase
{
    public function testErrorReported()
    {
        // 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->getMock('Observer', array('reportError'));

        $observer->expects($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。

例9.13 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
```

```
class FooTest extends PHPUnit_Framework_TestCase
{
    public function testFunctionCalledTwoTimesWithSpecificArguments()
    {
        $mock = $this->getMock('stdClass', array('set'));
        $mock->expects($this->exactly(2))
             ->method('set')
             ->withConsecutive(
                 array($this->equalTo('foo'), $this->greaterThan(0)),
                 array($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。

例9.14 より複雑な引数の検証

```
<?php
class SubjectTest extends PHPUnit_Framework_TestCase
{
    public function testErrorReported()
    {
        // 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->getMock('Observer', array('reportError'));

        $observer->expects($this->once())
                 ->method('reportError')
                 ->with($this->greaterThan(0),
                        $this->stringContains('Something'),
                        $this->callback(function($subject){
                          return is_callable(array($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();
    }
}
?>
```

例9.15 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
class FooTest extends PHPUnit_Framework_TestCase
{
    public function testIdenticalObjectPassed()
    {
        $expectedObject = new stdClass;

        $mock = $this->getMock('stdClass', array('foo'));
```

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

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

例9.16 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
class FooTest extends PHPUnit_Framework_TestCase
{
    public function testIdenticalObjectPassed()
    {
        $cloneArguments = true;

        $mock = $this->getMock(
            'stdClass',
            array(),
            array(),
            '',
            FALSE,
            TRUE,
            TRUE,
            $cloneArguments
        );

        // あるいは、モックビルダーを使います
        $mock = $this->getMockBuilder('stdClass')
                     ->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
    }
}
?>
```

表A.1「Constraints」 はメソッドの引数に適用できる制約、そして 表9.1「Matchers」 は起動回数を指定するために使える matcher です。

表9.1 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。

Matcher	意味
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。

注記

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。

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。

例9.17 トレイトの具象メソッドのテスト

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

    public abstract function abstractMethod();
}

class TraitClassTest extends PHPUnit_Framework_TestCase
{
    public function testConcreteMethod()
    {
        $mock = $this->getMockForTrait('AbstractTrait');
        $mock->expects($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。

例9.18 抽象クラスの具象メソッドのテスト

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

    public abstract function abstractMethod();
```

```
}

class AbstractClassTest extends PHPUnit_Framework_TestCase
{
    public function testConcreteMethod()
    {
        $stub = $this->getMockForAbstractClass('AbstractClass');
        $stub->expects($this->any())
             ->method('abstractMethod')
             ->will($this->returnValue(TRUE));

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

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りやすくするために 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。

例9.19「ウェブサービスのスタブ」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。

例9.19 ウェブサービスのスタブ

```
<?php
class GoogleTest extends PHPUnit_Framework_TestCase
{
    public function testSearch()
    {
        $googleSearch = $this->getMockFromWsdl(
          'GoogleSearch.wsdl', 'GoogleSearch'
        );

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

        $element = new StdClass;
        $element->summary = '';
        $element->URL = 'http://www.phpunit.de/';
        $element->snippet = '...';
        $element->title = '<b>PHPUnit</b>';
        $element->cachedSize = '11k';
        $element->relatedInformationPresent = TRUE;
        $element->hostName = 'www.phpunit.de';
        $element->directoryCategory = $directoryCategory;
        $element->directoryTitle = '';

        $result = new StdClass;
        $result->documentFiltering = FALSE;
        $result->searchComments = '';
        $result->estimatedTotalResultsCount = 3.9000;
        $result->estimateIsExact = FALSE;
        $result->resultElements = array($element);
        $result->searchQuery = 'PHPUnit';
```

```
        $result->startIndex = 1;
        $result->endIndex = 1;
        $result->searchTips = '';
        $result->directoryCategories = array();
        $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,
            '',
            '',
            ''
          )
        );
    }
}
?>
```

ファイルシステムのモック

vfsStream [https://github.com/mikey179/vfsStream] は 仮想ファイルシステム [http://ja.wikipedia.org/wiki/仮想ファイルシステム] 用の ストリームラッパー [http://www.php.net/streams] で、 ユニットテストにおいて実際のファイルシステムのモックを作るときに有用です。

Composer [http://getcomposer.org/] を使ってプロジェクトの依存関係を管理するには、`mikey179/vfsStream` へ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。 次に示すのは最小限の `composer.json` ファイルの例で、 開発時の PHPUnit 4.1 と vfsStream への依存を定義しています。

```
{
    "require-dev": {
        "phpunit/phpunit": "4.1.*",
        "mikey179/vfsStream": "1.*"
    }
}
```

例9.20「ファイルシステムを操作するクラス」 は、ファイルシステムを操作するクラスの例です。

例9.20 ファイルシステムを操作するクラス

```
<?php
class Example
{
    protected $id;
```

```
    protected $directory;

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

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

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

vfsStream のような仮想ファイルシステムがなければ、外部への影響なしに `setDirectory()` メソッドを個別にテストすることができません (例9.21「ファイルシステムを操作するクラスのテスト」を参照ください)。

例9.21 ファイルシステムを操作するクラスのテスト

```
<?php
require_once 'Example.php';

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

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

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

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

この方式には、次のような問題があります。

- 外部のリソースを使うため、ファイルシステムのテストが断続的になる可能性があります。その結果、テストがあまり当てにならないものになります。
- `setUp()` と `tearDown()` で、テストの前後にそのディレクトリがないことを確認する必要があります。
- `tearDown()` メソッドを実行する前にテストが異常終了したときに、ファイルシステム上にディレクトリが残ったままとなります。

例9.22「ファイルシステムを操作するクラスのテストにおけるファイルシステムのモックの作成」は、vfsStream を使ってファイルシステムのモックを作成し、 ファイルシステムを操作するクラスのテストを行う例です。

例9.22　ファイルシステムを操作するクラスのテストにおけるファイルシステムのモックの作成

```
<?php
require_once 'vfsStream/vfsStream.php';
require_once 'Example.php';

class ExampleTest extends PHPUnit_Framework_TestCase
{
    public function setUp()
    {
        vfsStreamWrapper::register();
        vfsStreamWrapper::setRoot(new vfsStreamDirectory('exampleDir'));
    }

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

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

この方式には次のような利点があります。

- テストが簡潔になります。
- vfsStream が、テスト対象のコードから操作するファイルシステム環境を用意してくれるので、開発者はそれを自由に扱えるようになります。
- 実際のファイルシステムを操作することがなくなるので、`tearDown()` メソッドでの後始末が不要になります。

第10章 テストの進め方

> You can always write more tests. However, you will quickly find that only a fraction of the tests you can imagine are actually useful. What you want is to write tests that fail even though you think they should work, or tests that succeed even though you think they should fail. Another way to think of it is in cost/benefit terms. You want to write tests that will pay you back with information.
>
> テストはいくらでも書くことができる。でも、じきにわかるだろうが、きみが考えているテストの中で本当に有用なものはごくわずかだ。 本当に書かなきゃいけないのは、 これは動くだろうと考えているにもかかわらず失敗するテスト。それから、 これは失敗するだろうと考えているにもかかわらず実際は成功するテストだ。 あるいはコストと利益の観点から考えてみてもいいだろう。 きみに何らかの情報を返してくれるテストを書かないとね。
>
> —Erich Gamma

開発中のテスト

開発中のソフトウェアの内部構造を変更し、 わかりやすく変更が簡単なものにする必要が出てきたときのことを考えましょう。 それによってソフトウェアの外部的な振る舞いが変わってしまってはいけません。 この、いわゆる リファクタリング [http://martinfowler.com/bliki/DefinitionOfRefactoring.html] (日本語) [http://capsctrl.que.jp/kdmsnr/wiki/bliki/?DefinitionOfRefactoring] を安全に行うにあたり、テストスイートが非常に重要となります。 もしテストスイートがなければ、リファクタリングによってシステムを壊してしまっても あなたはそれに気づかないでしょう。

以下の条件が、あなたのプロジェクトのコードや設計を改善するための助けとなるでしょう。 また、ユニットテストを使用することで、リファクタリングによって振る舞いが変化していないこと・ エラーが発生していないことが確認できます。

1. すべてのユニットテストが正常に動作すること。
2. コードが設計指針を満たしていること。
3. コードに冗長性がないこと。
4. コードには最小限のクラスおよびメソッドのみが含まれていること。

システムに新しい機能を追加する際には、まず最初にテストを書きます。 そのテストがきちんと実行できるようになった時点で、開発は終了です。 この手法については、次の章で詳しく説明します。

デバッグ中のテスト

不具合の報告を受けたら、すぐにでもそれを修正したいと思われることでしょう。 しかし、あせって修正しようとしても、経験上なかなかうまくいきません。 不具合を修正したつもりが新たな不具合を引き起こしていたなんてこともありがちですね。

はやる気持ちを抑えて、以下のようにしてみましょう。

1. 不具合を再現できることを確認します。
2. 不具合が発生する最小限のコードを見つけます。例えば、 もしおかしな数値が出力されるのなら、 その数値を計算しているオブジェクトが何なのかを探します。
3. その不具合のせいで今は失敗する (そして、不具合が修正されたら成功する) テストを書きます。

4. 不具合を修正します。

不具合が再現する最小限のコードを見つける過程で、 不具合の原因がわかるかもしれません。テストを書くことによって、 不具合を真の意味で修正できる可能性が高まるでしょう。なぜなら、 テストを書くことで、将来同じ間違いをする可能性を減らせるからです。これまでに書いたすべてのテストが、 不注意によって別の問題を発生させる可能性を減らすために役立っているのです。

> Unit testing offers many advantages:
>
> - Testing gives code authors and reviewers confidence that patches produce the correct results.
> - Authoring testcases is a good impetus for developers to discover edge cases.
> - Testing provides a good way to catch regressions quickly, and to make sure that no regression will be repeated twice.
> - Unit tests provide working examples for how to use an API and can significantly aid documentation efforts.
>
> Overall, integrated unit testing makes the cost and risk of any individual change smaller. It will allow the project to make [...] major architectural improvements [...] quickly and confidently.
>
> ユニットテストには、こんなに多くの利点がある。
>
> - 実際にコードを書いた人とコードをレビューする人が意識を共有できるようになり、パッチの精度があがる。
> - テストケースを書くことは、いろいろな想定外の事態に備えるための原動力となる。
> - テストにより、リグレッションを早期に発見できるようになる。また、同じリグレッションを二度と起こさないようになる。
> - ユニットテストそのものが、その API の使用法についてのよいサンプルとなる。ドキュメント作成の際に助けとなるだろう。
>
> まとめよう。ユニットテストをうまく組み込めば、 プログラムを変更する際の手間やリスクをより減らすことになるのだ。 プロジェクトが【中略】のアーキテクチャに関する大きな改修【中略】を素早く、自信を持って行うことを可能にするだろう。
>
> —Benjamin Smedberg

第11章 コードカバレッジ解析

> The beauty of testing is found not in the effort but in the efficiency.
>
> - テストの美学は、どれだけ汗を流したかではなく どれだけ効率的であるかである。
>
> Knowing what should be tested is beautiful, and knowing what is being tested is beautiful.
>
> - 何をテストすべきなのかを知ること、何がテストされているのかを知ること。 これが大切だ。
>
> —Murali Nandigama

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。 次のような疑問に対する答えとなるでしょう。

- テストされていないコードを見つけるには? 言い換えれば、まだテストで カバーされていない部分を見つけるには?
- 完全にテストができたことをどうやって確認するの?

ステートメントカバレッジというのは、たとえば 100 行のコードで構成されるメソッドがあった場合に、 もしテストで実際に実行されたのがそのうちの 75 行だけだったなら、 そのメソッドのコードカバレッジは 75 パーセントだと考えるということです。

PHPUnit のコードカバレッジ解析では PHP_CodeCoverage [http://github.com/sebastianbergmann/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 [http://www.xdebug.org/] 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。

注記

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 Xdebug がロードできないという 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 [http://xdebug.org/docs/install] を読んでみましょう。

それでは、BankAccount クラスについての コードカバレッジレポートを作成してみましょう。

```
PHPUnit 4.1.0 by Sebastian Bergmann.

...

Time: 0 seconds

OK (3 tests, 3 assertions)

Generating report, this may take a moment.phpunit --coverage-html ./report BankAccountTe
PHPUnit 4.1.0 by Sebastian Bergmann.

...

Time: 0 seconds
```

```
OK (3 tests, 3 assertions)

Generating report, this may take a moment.
```

図11.1「setBalance() のコードカバレッジ」 は、コードカバレッジレポートの一部を抜粋したものです。 テスト時に実行された行は、緑色で強調表示されます。 実行可能なコードであるにもかかわらず実行されなかった行については赤色で強調表示されます。 また、"無意味なコード" についてはグレーで強調表示されます。 行の左にある数字は、その行をカバーするテストの数を表します。

図11.1 setBalance() のコードカバレッジ

```
 82          :     /**
 83          :      * Sets the bank account's balance.
 84          :      *
 85          :      * @param  float $balance
 86          :      * @throws BankAccountException
 87          :      * @access protected
 88          :      */
 89          :     protected function setBalance($balance)
 90          :     {
 91        2 :         if ($balance >= 0) {
 92        0 :             $this->balance = $balance;
 93        0 :         } else {
 94        2 :             throw new BankAccountException;
 95          :         }
 96        0 :     }
```

BankAccount のコードカバレッジレポートからわかることは、setBalance()、depositMoney() をコールするテストがまだ存在しないということ、そして withdrawMoney() に正しい値を指定した場合のテストも存在しないということです。 BankAccountTest クラスに追加するテストを 例11.1「完全なコードカバレッジを達成するために欠けているテスト」 に示します。これによって、BankAccount クラスのテストケースを完全に網羅できるようになります。

例11.1 完全なコードカバレッジを達成するために欠けているテスト

```php
<?php
require_once 'BankAccount.php';

class BankAccountTest extends PHPUnit_Framework_TestCase
{
    // ...

    public function testDepositWithdrawMoney()
    {
        $this->assertEquals(0, $this->ba->getBalance());
        $this->ba->depositMoney(1);
        $this->assertEquals(1, $this->ba->getBalance());
        $this->ba->withdrawMoney(1);
        $this->assertEquals(0, $this->ba->getBalance());
    }
}
?>
```

図11.2「setBalance() にテストを追加した後のコードカバレッジ」 は、 テストを追加した後の setBalance() のコードカバレッジです。

図11.2 **setBalance()** にテストを追加した後のコードカバレッジ

```
 82                 :     /**
 83                 :      * Sets the bank account's balance.
 84                 :      *
 85                 :      * @param  float $balance
 86                 :      * @throws BankAccountException
 87                 :      * @access protected
 88                 :      */
 89                 :     protected function setBalance($balance)
 90                 :     {
 91               3 :         if ($balance >= 0) {
 92               1 :             $this->balance = $balance;
 93               1 :         } else {
 94               2 :             throw new BankAccountException;
 95                 :         }
 96               1 :     }
```

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（表B.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」）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。 例11.2「どのメソッドを対象とするかを指定したテスト」 に例を示します。

例11.2 どのメソッドを対象とするかを指定したテスト

```php
<?php
require_once 'BankAccount.php';

class BankAccountTest extends PHPUnit_Framework_TestCase
{
    protected $ba;

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

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

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

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

```
            return;
        }

        $this->fail();
    }

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

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

            return;
        }

        $this->fail();
    }

    /**
     * @covers BankAccount::getBalance
     * @covers BankAccount::depositMoney
     * @covers BankAccount::withdrawMoney
     */

    public function testDepositWithdrawMoney()
    {
        $this->assertEquals(0, $this->ba->getBalance());
        $this->ba->depositMoney(1);
        $this->assertEquals(1, $this->ba->getBalance());
        $this->ba->withdrawMoney(1);
        $this->assertEquals(0, $this->ba->getBalance());
    }
}
?>
```

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。

例11.3 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
class GuestbookIntegrationTest extends PHPUnit_Extensions_Database_TestCase
{
    /**
     * @coversNothing
     */
    public function testAddEntry()
    {
        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Hello world!");

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

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

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 アノテーションを例 11.4「@codeCoverageIgnore、@codeCoverageIgnoreStart および @codeCoverageIgnoreEnd アノテーションの使用法」のように使用します。

例11.4 @codeCoverageIgnore、@codeCoverageIgnoreStart および @codeCoverageIgnoreEnd アノテーションの使用法

```
<?php
/**
 * @codeCoverageIgnore
 */
class Foo
{
    public function bar()
    {
    }
}

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

if (FALSE) {
    // @codeCoverageIgnoreStart
    print '*';
    // @codeCoverageIgnoreEnd
}
?>
```

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。

ファイルのインクルードや除外

デフォルトでは、1 行でもコードが実行されたソースコードファイルはすべて (そしてそのようなファイルのみが) レポートに含められます。レポートに含まれるソースコードファイルは、ホワイトリスト方式あるいはブラックリスト方式でフィルタリングすることができます。

ブラックリストには、PHPUnit 自身のソースコードファイルやテストファイルがデフォルトで登録されています。ホワイトリストが空 (デフォルト) の場合はブラックリストを使用し、ホワイトリストが空でない場合はホワイトリストを使用します。ホワイトリスト内

の各ファイルは、そのファイルが実行されるかどうかにかかわらず レポートに追加されます。追加されたファイルのすべての行は、 実行不能な行も含めて「実行されなかった行」とみなします。

PHPUnit の設定で `processUncoveredFilesFromWhitelist="true"` (「コードカバレッジ対象のファイルの追加や除外」 を参照ください) とすると、これらのファイルが PHP_CodeCoverage に渡され、実行可能な行数を適切に算出します。

注記

`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。

PHPUnit の XML 設定ファイル (「コードカバレッジ対象のファイルの追加や除外」 を参照ください) を使って、ブラックリストやホワイトリストを制御することができます。 おすすめの方法は、ホワイトリスト方式を使ってコードカバレッジレポートに含めるファイルを制御することです。

エッジケース

ほとんどの場面では、PHPUnit が「行単位の」コードカバレッジ情報を提供してくれると考えて間違いないでしょう。 しかし、その情報収集の方法が原因で、特筆すべきエッジケースもいくつか存在します。

例11.5

```
<?php
// 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_coveraged();

// 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();
}
?>
```

第12章 テストのその他の使用法

自動テストに慣れてくると、 ほかの目的のためにもテストを使いたくなってくることでしょう。 ここではそんな例を説明します。

アジャイルな文書作成

一般的に、エクストリームプログラミングのようなアジャイルプロセスを採用しているプロジェクトでは、 ドキュメントの内容が実際の設計やコードに追いついていないことが多いものです。 エクストリームプログラミングでは コードの共同所有 *(collective code ownership)* を要求しており、 すべての開発者がシステム全体の動作を知っておく必要があります。 作成するテストに対して、そのクラスが何を行うべきなのかを示すような 「わかりやすい」名前をつけられるようにさえしておけば、PHPUnit の TestDox 機能を使用して自動的にドキュメントを生成することができます。 このドキュメントにより、開発者たちはプロジェクト内の各クラスが どのようにふるまうべきなのかを知ることができます。

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 4.1.0 by Sebastian Bergmann.

BankAccount
 [x] Balance is initially zero
 [x] Balance cannot become negativephpunit --testdox BankAccountTest
PHPUnit 4.1.0 by Sebastian Bergmann.

BankAccount
 [x] Balance is initially zero
 [x] 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。

複数チームでのテスト

あるパッケージについての機能を文書化するためにテストを書いているとき、 そのテストの所有者はあなたです。今あなたがテストを作成しているパッケージの作者は、 そのテス

トのことについては何も知りません。パッケージの作者とよりつながりを深めるため、 作成したテストを使用してコミュニケートしたり、 そのテストを使用して共同作業をしたりすることができるでしょう。

あなたが作成したテストを使用してパッケージの作者と共同作業をすることになれば、 テストも共同で書くことになります。そうすることで、 より多くのテストケースを挙げられるようになるでしょう。 「暗黙の了解」などに頼っていては、共同作業はできません。 テストと同時に、あなたはそのパッケージに対して期待していることを正確に文書化することになります。 また、すべてのテストにクリアした時点で、 作者はパッケージが完成したことを知ることになります。

スタブ (本書の前のほうで説明した "モックオブジェクト" の章を参照ください) を使用することで、パッケージの作者と別れても作業できるようになります。 パッケージ作者の仕事は、パッケージの実際の実装でテストをクリアするようにすること。 そしてあなたの仕事はあなたが書いたコードでテストをクリアするようにすることです。 この段階になれば、あなたはスタブオブジェクトを使用すればよいのです。 このやり方により、2 つのチームが独立して開発できるようになります。

第13章 PHPUnit と Selenium

Selenium Server

Selenium Server [http://seleniumhq.org/] はテストツールのひとつです。これは、OS を通してブラウザのプロセスを動かし、 ブラウザのタスクを自動実行します。 あらゆるプログラミング言語で稼動しているウェブサイトに対応しており、 現在主流のあらゆるブラウザで使用することができます。Selenium RC は Selenium Core [http://seleniumhq.org/] を使用しています。これは、ブラウザ上でのタスクを自動的に実行する JavaScript のライブラリです。Selenium でのテストは、 一般のユーザが使用するのと同じようにブラウザ上で直接実行されます。 主な使用例としては、受け入れテスト (各システム単体のテストではなく、結合されたシステム全体に対するテスト) や ブラウザの互換性のテスト (ウェブアプリケーションを、さまざまなオペレーティングシステムやブラウザでテストする) などがあります。

PHPUnit_Selenium がサポートしている唯一のシナリオは、 Selenium 2.x サーバを使うものです。 Selenium 2.x サーバにアクセスするには、1.0 から存在する古い形式の Selenium RC API を使うか、 あるいは PHPUnit_Selenium 1.2 で一部実装済みの WebDriver API を使います。

なぜそうしたかというと、Selenium 2 には後方互換性があり、 Selenium RC がもうメンテナンスされていないからです。

インストール

まず、Selenium Server をインストールします。

1. Selenium Server [http://seleniumhq.org/download/] の配布アーカイブをダウンロードする。
2. アーカイブを展開し、`selenium-server-standalone-2.9.0.jar` (バージョンをチェックすること) を `/usr/local/bin` などにコピーする。
3. **`java -jar /usr/local/bin/selenium-server-standalone-2.9.0.jar`** などのようにして Selenium RC サーバを起動する。

PHPUnit_Selenium パッケージは、PHPUnit の PHAR 版の中に含まれています。 Composer でインストールするには、 `"require-dev"` に次の行を追加します。

```
"phpunit/phpunit-selenium": ">=1.2"
```

これで、クライアント/サーバ プロトコルを用いて Selenium Server にコマンドを送信できるようになりました。

PHPUnit_Extensions_Selenium2TestCase

`PHPUnit_Extensions_Selenium2TestCase` テストケースは、 WebDriver API を利用します (実装しているのはその一部だけです)。

例13.1「PHPUnit_Extensions_Selenium2TestCase の使用例」は、 ウェブサイト `http://www.example.com/` の `<title>` 要素の内容をテストする方法を示したものです。

例13.1 PHPUnit_Extensions_Selenium2TestCase の使用例

```
<?php
```

```
class WebTest extends PHPUnit_Extensions_Selenium2TestCase
{
    protected function setUp()
    {
        $this->setBrowser('firefox');
        $this->setBrowserUrl('http://www.example.com/');
    }

    public function testTitle()
    {
        $this->url('http://www.example.com/');
        $this->assertEquals('Example WWW Page', $this->title());
    }

}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 28 seconds, Memory: 3.00Mb

There was 1 failure:

1) WebTest::testTitle
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'Example WWW Page'
+'IANA — Example domains'

/home/giorgio/WebTest.php:13

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 28 seconds, Memory: 3.00Mb

There was 1 failure:

1) WebTest::testTitle
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'Example WWW Page'
+'IANA — Example domains'

/home/giorgio/WebTest.php:13

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

Selenium2TestCare のコマンドは __call() を使って実装しています。 サポートする機能の一覧は PHPUnit_Extensions_Selenium2TestCase のエンドツーエンドテスト [https://github.com/sebastianbergmann/phpunit-selenium/blob/master/Tests/Selenium2TestCaseTest.php] を参照ください。

PHPUnit_Extensions_SeleniumTestCase

PHPUnit_Extensions_SeleniumTestCase は、Selenium Server と通信するための クライアント/サーバ プロトコルを実装したものです。 また、ウェブのテスト用に特化したアサーションメソッドも提供します。

例13.2「PHPUnit_Extensions_SeleniumTestCase の使用例」 は、 ウェブサイト http://www.example.com/ の <title> 要素の内容をテストする方法を示したものです。

例13.2 PHPUnit_Extensions_SeleniumTestCase の使用例

```
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
    protected function setUp()
    {
        $this->setBrowser('*firefox');
        $this->setBrowserUrl('http://www.example.com/');
    }

    public function testTitle()
    {
        $this->open('http://www.example.com/');
        $this->assertTitle('Example WWW Page');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 9 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/domains/example/

Failed asserting that 'IANA — Example domains' matches PCRE pattern "/Example WWW Page/"


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 9 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/domains/example/

Failed asserting that 'IANA — Example domains' matches PCRE pattern "/Example WWW Page/"

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

PHPUnit_Framework_TestCase クラスとは異なり、PHPUnit_Extensions_SeleniumTestCase を継承したテストケースクラスは setUp() メソッドが必須となります。 このメソッド内で、Selenium Server セッションの設定を行います。 ここで使用できるメソッドの一覧は 表13.1「Selenium Server API: セットアップ」を参照ください。

表13.1 Selenium Server API: セットアップ

メソッド	意味
void setBrowser(string $browser)	Selenium Server が使用するブラウザを設定します。
void setBrowserUrl(string $browserUrl)	テストするベース URL を設定します。
void setHost(string $host)	Selenium Server に接続する際のホスト名を設定します。
void setPort(int $port)	Selenium Server に接続する際のポートを設定します。
void setTimeout(int $timeout)	Selenium Server に接続する際のタイムアウト値を設定します。
void setSleep(int $seconds)	Selenium Server クライアントが、Selenium Server のサーバにアクションコマンドを送信してから待機する秒数を設定します。

PHPUnit では、Selenium のテストが失敗したときのスクリーンショットを撮ることができます。 この機能を使うには、$captureScreenshotOnFailure、 $screenshotPath および $screenshotUrl をテストケースクラス内で 例13.3「テストに失敗したときのスクリーンショットの取得」のように指定します。

例13.3 テストに失敗したときのスクリーンショットの取得

```
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
    protected $captureScreenshotOnFailure = TRUE;
    protected $screenshotPath = '/var/www/localhost/htdocs/screenshots';
    protected $screenshotUrl = 'http://localhost/screenshots';

    protected function setUp()
    {
        $this->setBrowser('*firefox');
        $this->setBrowserUrl('http://www.example.com/');
    }

    public function testTitle()
    {
        $this->open('http://www.example.com/');
        $this->assertTitle('Example WWW Page');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F
```

```
Time: 7 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/domains/example/
Screenshot: http://localhost/screenshots/334b080f2364b5f11568ee1c7f6742c9.png

Failed asserting that 'IANA — Example domains' matches PCRE pattern "/Example WWW Page/"


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit WebTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 7 seconds, Memory: 6.00Mb

There was 1 failure:

1) WebTest::testTitle
Current URL: http://www.iana.org/domains/example/
Screenshot: http://localhost/screenshots/334b080f2364b5f11568ee1c7f6742c9.png

Failed asserting that 'IANA — Example domains' matches PCRE pattern "/Example WWW Page/"

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

複数のブラウザを使用してテストを行なうこともできます。この場合は、`setBrowser()` でブラウザの設定を行うかわりに、テストケースクラスの中で `$browsers` という名前の `public static` な配列を作成します。この配列の各項目が個々のブラウザの設定を表します。これらのブラウザは、それぞれ別の Selenium Server のサーバで管理することができます。例13.4「複数のブラウザの設定管理」に例を示します。

例13.4 複数のブラウザの設定管理

```php
<?php
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
    public static $browsers = array(
      array(
        'name'    => 'Firefox on Linux',
        'browser' => '*firefox',
        'host'    => 'my.linux.box',
        'port'    => 4444,
        'timeout' => 30000,
      ),
      array(
        'name'    => 'Safari on MacOS X',
        'browser' => '*safari',
        'host'    => 'my.macosx.box',
        'port'    => 4444,
        'timeout' => 30000,
      ),
      array(
        'name'    => 'Safari on Windows XP',
        'browser' => '*custom C:\Program Files\Safari\Safari.exe -url',
        'host'    => 'my.windowsxp.box',
```

```
        'port'    => 4444,
        'timeout' => 30000,
      ),
      array(
        'name'    => 'Internet Explorer on Windows XP',
        'browser' => '*iexplore',
        'host'    => 'my.windowsxp.box',
        'port'    => 4444,
        'timeout' => 30000,
      )
    );

    protected function setUp()
    {
        $this->setBrowserUrl('http://www.example.com/');
    }

    public function testTitle()
    {
        $this->open('http://www.example.com/');
        $this->assertTitle('Example Web Page');
    }
}
?>
```

PHPUnit_Extensions_SeleniumTestCase を使用すると、 Selenium で実行したテストのカバレッジ情報を収集することができます。

1. PHPUnit/Extensions/SeleniumCommon/phpunit_coverage.php をウェブサーバのドキュメントルートディレクトリにコピーします。

2. ウェブサーバ上の php.ini ファイルで、PHPUnit/Extensions/SeleniumCommon/prepend.php と PHPUnit/Extensions/SeleniumCommon/append.php をそれぞれ auto_prepend_file および auto_append_file に設定します。

3. PHPUnit_Extensions_SeleniumTestCase を継承したテストケースクラスで、

   ```
   protected $coverageScriptUrl = 'http://host/phpunit_coverage.php';
   ```

   のようにして phpunit_coverage.php スクリプトの URL を指定します。

表13.2「アサーション」は、PHPUnit_Extensions_SeleniumTestCase が提供するさまざまなアサーションメソッドの一覧です。

表13.2 アサーション

アサーション	意味
void assertElementValueEquals(string $locator, string $text)	$locator で表される要素の値が $text と異なる場合にエラーを報告します。
void assertElementValueNotEquals(string $locator, string $text)	$locator で表される要素の値が $text と等しい場合にエラーを報告します。
void assertElementValueContains(string $locator, string $text)	$locator で表される要素の値が $text を含まない場合にエラーを報告します。
void assertElementValueNotContains(string $locator, string $text)	$locator で表される要素の値が $text を含む場合にエラーを報告します。

アサーション	意味
`void assertElementContainsText(string $locator, string $text)`	`$locator` で表される要素が `$text` を含まない場合にエラーを報告します。
`void assertElementNotContainsText(string $locator, string $text)`	`$locator` で表される要素が `$text` を含む場合にエラーを報告します。
`void assertSelectHasOption(string $selectLocator, string $option)`	指定したオプションが使用できない場合にエラーを報告します。
`void assertSelectNotHasOption(string $selectLocator, string $option)`	指定したオプションが使用できる場合にエラーを報告します。
`void assertSelected($selectLocator, $option)`	指定したラベルが選択されていない場合にエラーを報告します。
`void assertNotSelected($selectLocator, $option)`	指定したラベルが選択されている場合にエラーを報告します。
`void assertIsSelected(string $selectLocator, string $value)`	指定した値が選択されていない場合にエラーを報告します。
`void assertIsNotSelected(string $selectLocator, string $value)`	指定した値が選択されている場合にエラーを報告します。

表13.3「テンプレートメソッド」 は、 `PHPUnit_Extensions_SeleniumTestCase` のテンプレートメソッドをまとめたものです。

表13.3 テンプレートメソッド

メソッド	意味
`void defaultAssertions()`	テストケース内のすべてのテストで共有するアサーションを上書きします。 このメソッドは、Selenium Server のサーバにコマンドが送信されるたびに (送信された後に) コールされます。

使用できるコマンドのリファレンスや実際の使用法については Selenium のドキュメント [http://release.seleniumhq.org/selenium-core/1.0.1/reference.html] を参照ください。

Selenium 1 のコマンドは、__call で動的に実装されています。 PHPUnit_Extensions_SeleniumTestCase_Driver::__call() の API ドキュメント [https://github.com/sebastianbergmann/phpunit-selenium/blob/master/PHPUnit/Extensions/SeleniumTestCase/Driver.php#L410] に、PHP 側で対応しているすべてのメソッドの一覧があります。 また、引数や返り値の型も確認できます。

`runSelenese($filename)` メソッドを使用すると、 Selenese/HTML の設定から Selenium のテストを実行することができます。 さらに、静的属性 `$seleneseDirectory` を使用すると、 Selenese/HTML ファイルを含むディレクトリから自動的にテストオブジェクトを作成することができます。 指定したディレクトリ配下を再帰的に走査し、 `.htm` ファイルを探します。このファイルには Selenese/HTML が含まれているものとします。例として 例13.5「Selenese/HTML ファイルのディレクトリをテストとして使用する」 を参照ください。

例13.5 Selenese/HTML ファイルのディレクトリをテストとして使用する

```
<?php
```

```
require_once 'PHPUnit/Extensions/SeleniumTestCase.php';

class SeleneseTests extends PHPUnit_Extensions_SeleniumTestCase
{
    public static $seleneseDirectory = '/path/to/files';
}
?>
```

Selenium 1.1.1 から取り込まれた実験的な機能として、ユーザーが複数のテストでセッションを共有できるようになりました。 現在サポートしているのは、ひとつのブラウザを使うときに全テストでセッションを共有するという場合だけです。 セッションの共有機能を使うには、ブートストラップファイルで `PHPUnit_Extensions_SeleniumTestCase::shareSession(true)` をコールします。 テストが成功しなかった (失敗、あるいは不完全) 場合は、共有セッションがリセットされます。 クッキーをリセットしたり、(tearDown() メソッドで) テスト対象のアプリケーションからログアウトしたりして テストがお互い干渉しあわないようにするのは、ユーザ側の責任となります。

第14章 ログ出力

PHPUnit は、いくつかの形式のログファイルを作成することができます。

テスト結果 (XML)

PHPUnit が作成するテスト結果の XML のログファイルは、 Apache Ant の JUnit タスク [http://ant.apache.org/manual/OptionalTasks/junit.html] が使用しているものを参考にしています。 以下の例は、`ArrayTest` のテストが生成した XML ログファイルです。

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

次の XML ログファイルは、テストクラス `FailureErrorTest` にある 2 つのテスト `testFailure` および `testError` が出力したものです。失敗やエラーがどのように表示されるのかを確認しましょう。

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

/home/sb/FailureErrorTest.php:8
</failure>
    </testcase>
```

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

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

テスト結果 (TAP)

Test Anything Protocol (TAP) [http://testanything.org/] は、Perl のモジュールをテストする際に使用する、 シンプルなテキストベースのインターフェイスです。 以下の例は、ArrayTest のテストが生成した TAP ログファイルです。

```
TAP version 13
ok 1 - testNewArrayIsEmpty(ArrayTest)
ok 2 - testArrayContainsAnElement(ArrayTest)
1..2
```

次の TAP ログファイルは、テストクラス FailureErrorTest にあるメソッド testFailure および testError が出力したものです。失敗やエラーがどのように表示されるのかを確認しましょう。

```
TAP version 13
not ok 1 - Failure: testFailure(FailureErrorTest)
  ---
  message: 'Failed asserting that <integer:2> matches expected value <integer:1>.'
  severity: fail
  data:
    got: 2
    expected: 1
  ...
not ok 2 - Error: testError(FailureErrorTest)
1..2
```

テスト結果 (JSON)

JavaScript Object Notation (JSON) [http://www.json.org/] は、軽量なデータ交換用フォーマットです。次の例は、 ArrayTest のテストが作成した JSON メッセージです。

```
{"event":"suiteStart","suite":"ArrayTest","tests":2}
{"event":"test","suite":"ArrayTest",
 "test":"testNewArrayIsEmpty(ArrayTest)","status":"pass",
 "time":0.000460147858,"trace":[],"message":""}
{"event":"test","suite":"ArrayTest",
 "test":"testArrayContainsAnElement(ArrayTest)","status":"pass",
 "time":0.000422954559,"trace":[],"message":""}
```

次の JSON メッセージは、 FailureErrorTest にある 2 つのテスト testFailure および testError が出力したものです。失敗やエラーがどのように表示されるのかを確認しましょう。

```
{"event":"suiteStart","suite":"FailureErrorTest","tests":2}
```

```
{"event":"test","suite":"FailureErrorTest",
 "test":"testFailure(FailureErrorTest)","status":"fail",
 "time":0.0082459449768066,"trace":[],
 "message":"Failed asserting that <integer:2> is equal to <integer:1>."}
{"event":"test","suite":"FailureErrorTest",
 "test":"testError(FailureErrorTest)","status":"error",
 "time":0.0083.90152893066,"trace":[],"message":""}
```

コードカバレッジ (XML)

PHPUnit がコードカバレッジ情報のログ出力の際に使用している XML のフォーマットは、 Clover [http://www.atlassian.com/software/clover/] のものを参考にしています。 以下の例は、`BankAccountTest` のテストが生成した XML ログファイルです。

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

コードカバレッジ (テキスト)

人間が読める形式のコードカバレッジ情報を、コマンドラインあるいはテキストファイルに出力します。 この出力フォーマットの狙いは、ちょっとしたクラス群のカバレッジの概要を手軽に把握することです。 大規模なプロジェクトでは、このフォーマットを使えばプロジェクト全体のカバレッジを大まかに把握しやすくなるでしょう。 `--filter` と組み合わせて使うこともできます。 コマンドラインから使う場合は `php://stdout` に書き込みます。 この出力は `--colors` の設定を反映したものになります。 コマンドラインから使った場合は、デフォルトの出力先は標準出力となります。 デフォルトでは、テストで少なくとも一行はカバーしているファイルしか表示しません。 この設定は、xml の

showUncoveredFiles オプションでしか変更できません。 「ログ出力」 を参照ください。 デフォルトでは、すべてのファイルとそのカバレッジ情報が、詳細形式で表示されます。 この設定は、xml のオプション showOnlySummary で変更できます。

図14.1 コマンドラインでの、色つきのコードカバレッジ出力

```
Code Coverage Report for "BankAccount"
  2011-10-21 13:12:17

 Summary:
  Classes: 87.50% (21/24)
  Methods: 78.95% (30/38)
  Lines:   90.86% (169/186)

@bankaccount.controller::BankAccountController
  Methods: 100.00% ( 2/ 2)   Lines: 100.00% ( 13/ 13)
@bankaccount.controller::BankAccountListController
  Methods: 100.00% ( 1/ 1)   Lines: 100.00% (  2/  2)
@bankaccount.framework::ControllerException
  Methods: 100.00% ( 0/ 0)   Lines: 100.00% (  0/  0)
@bankaccount.framework::ControllerFactory
  Methods: 100.00% ( 2/ 2)   Lines: 100.00% ( 13/ 13)
@bankaccount.framework::FrontController
  Methods: 100.00% ( 2/ 2)   Lines: 100.00% ( 13/ 13)
@bankaccount.framework::HashMap
  Methods: 100.00% ( 2/ 2)   Lines: 100.00% (  5/  5)
@bankaccount.framework::IdentityMap
  Methods:   0.00% ( 0/ 6)   Lines:   0.00% (  0/ 13)
```

第15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。

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。

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。 例15.1「PHPUnit_Framework_Assert クラスの assertTrue() および isTrue() メソッド」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。

例15.1 PHPUnit_Framework_Assert クラスの assertTrue() および isTrue() メソッド

```php
<?php
abstract class PHPUnit_Framework_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;
    }

    // ...
}?>
```

例15.2「PHPUnit_Framework_Constraint_IsTrue クラス」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。

例15.2 PHPUnit_Framework_Constraint_IsTrue クラス

```
<?php
class PHPUnit_Framework_Constraint_IsTrue extends PHPUnit_Framework_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。

PHPUnit_Framework_TestListener の実装

例15.3「シンプルなテストリスナー」は、PHPUnit_Framework_TestListener インターフェイスのシンプルな実装例です。

例15.3 シンプルなテストリスナー

```
<?php
class SimpleTestListener implements PHPUnit_Framework_TestListener
{
    public function addError(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("テスト '%s' の実行中にエラーが発生\n", $test->getName());
    }

    public function addFailure(PHPUnit_Framework_Test $test, PHPUnit_Framework_Assertion
    {
        printf("テスト '%s' に失敗\n", $test->getName());
    }

    public function addIncompleteTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("テスト '%s' は未完成\n", $test->getName());
```

```
    }

    public function addRiskyTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("テスト '%s' は危険\n", $test->getName());
    }

    public function addSkippedTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("テスト '%s' をスキップ\n", $test->getName());
    }

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

    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        printf("テスト '%s' が終了\n", $test->getName());
    }

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

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

例15.4「ベーステストリスナーの利用法」 は、抽象クラス PHPUnit_Framework_BaseTestListener のサブクラスを作る例です。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。

例15.4 ベーステストリスナーの利用法

```
<?php
class ShortTestListener extends PHPUnit_Framework_BaseTestListener
{
    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        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。

PHPUnit_Extensions_TestDecorator のサブクラスの作成

PHPUnit_Extensions_TestDecorator のサブクラスでテストケースあるいはテストスイートをラッピングし、 デコレータパターンを使用することで 各テストの実行前後に何らかの処理をさせることができます。

PHPUnit には、PHPUnit_Extensions_RepeatedTest および PHPUnit_Extensions_TestSetup という 2 つの具象テストデコレータが付属しています。 前者はテストを繰り返し実行し、それらが全て成功した場合にのみ成功とみなします。 後者については 4章フィクスチャ で説明しました。

例15.5「RepeatedTest デコレータ」 は、テストデコレータ PHPUnit_Extensions_RepeatedTest の一部を抜粋したものです。独自のデコレータを作成するための参考にしてください。

例15.5 RepeatedTest デコレータ

```
<?php
require_once 'PHPUnit/Extensions/TestDecorator.php';

class PHPUnit_Extensions_RepeatedTest extends PHPUnit_Extensions_TestDecorator
{
    private $timesRepeat = 1;

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

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

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

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

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

        return $result;
    }
}
?>
```

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 例15.6「データ駆動のテスト」 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。

例15.6 データ駆動のテスト

```
<?php
```

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

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

    public function count()
    {
        return 1;
    }

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

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

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

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

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

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

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

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

        return $result;
    }
}

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

.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.
```

付録A Assertions

This appendix lists the various assertion methods that are available.

assertArrayHasKey()

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

Reports an error identified by `$message` if `$array` does not have the `$key`.

`assertArrayNotHasKey()` is the inverse of this assertion and takes the same arguments.

例A.1 Usage of assertArrayHasKey()

```
<?php
class ArrayHasKeyTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertArrayHasKey('foo', array('bar' => 'baz'));
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ArrayHasKeyTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ArrayHasKeyTest.php:6

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

assertClassHasAttribute()

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

Reports an error identified by `$message` if `$className::attributeName` does not exist.

`assertClassNotHasAttribute()` is the inverse of this assertion and takes the same arguments.

例A.2 Usage of assertClassHasAttribute()

```
<?php
class ClassHasAttributeTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertClassHasAttribute('foo', 'stdClass');
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ClassHasAttributeTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertClassHasStaticAttribute()

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

Reports an error identified by `$message` if `$className::attributeName` does not exist.

`assertClassNotHasStaticAttribute()` is the inverse of this assertion and takes the same arguments.

例A.3 Usage of assertClassHasStaticAttribute()

```
<?php
```

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ClassHasStaticAttributeTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ClassHasStaticAttributeTest::testFailure
Failed asserting that class "stdClass" has static attribute "foo".

/home/sb/ClassHasStaticAttributeTest.php:6

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

assertContains()

`assertContains(mixed $needle, Iterator|array $haystack[, string $message = ''])`

Reports an error identified by `$message` if `$needle` is not an element of `$haystack`.

`assertNotContains()` is the inverse of this assertion and takes the same arguments.

`assertAttributeContains()` and `assertAttributeNotContains()` are convenience wrappers that use a `public`, `protected`, or `private` attribute of a class or object as the haystack.

例A.4 Usage of assertContains()

```
<?php
class ContainsTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertContains(4, array(1, 2, 3));
```

```
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ContainsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 = '']])`

Reports an error identified by `$message` if `$needle` is not a substring of `$haystack`.

例A.5 Usage of assertContains()

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ContainsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertContainsOnly()

`assertContainsOnly(string $type, Iterator|array $haystack[, boolean $isNativeType = NULL, string $message = ''])`

Reports an error identified by `$message` if `$haystack` does not contain only variables of type `$type`.

`$isNativeType` is a flag used to indicate whether `$type` is a native PHP type or not.

`assertNotContainsOnly()` is the inverse of this assertion and takes the same arguments.

`assertAttributeContainsOnly()` and `assertAttributeNotContainsOnly()` are convenience wrappers that use a `public`, `protected`, or `private` attribute of a class or object as the haystack.

例A.6 Usage of assertContainsOnly()

```php
<?php
class ContainsOnlyTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnly('string', array('1', '2', 3));
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ContainsOnlyTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsOnlyTest::testFailure
Failed asserting that Array (
    0 => '1'
    1 => '2'
    2 => 3
) contains only values of type "string".

/home/sb/ContainsOnlyTest.php:6

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

assertContainsOnlyInstancesOf()

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

Reports an error identified by `$message` if `$haystack` does not contain only instances of class `$classname`.

例A.7 Usage of assertContainsOnlyInstancesOf()

```php
<?php
class ContainsOnlyInstancesOfTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertContainsOnlyInstancesOf('Foo', array(new Foo(), new Bar(), new Foo(
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ContainsOnlyInstancesOfTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsOnlyInstancesOfTest::testFailure
Failed asserting that Array ([0]=> Bar Object(...)) is an instance of class "Foo".

/home/sb/ContainsOnlyInstancesOfTest.php:6

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

assertCount()

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

Reports an error identified by `$message` if the number of elements in `$haystack` is not `$expectedCount`.

`assertNotCount()` is the inverse of this assertion and takes the same arguments.

例A.8 Usage of assertCount()

```php
<?php
class CountTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertCount(0, array('foo'));
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit CountTest
PHPUnit 4.1.0 by Sebastian Bergmann.
```

```
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.
```

assertEmpty()

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

Reports an error identified by `$message` if `$actual` is not empty.

`assertNotEmpty()` is the inverse of this assertion and takes the same arguments.

`assertAttributeEmpty()` and `assertAttributeNotEmpty()` are convenience wrappers that can be applied to a `public`, `protected`, or `private` attribute of a class or object.

例A.9 Usage of assertEmpty()

```php
<?php
class EmptyTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertEmpty(array('foo'));
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit EmptyTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:
```

```
1) EmptyTest::testFailure
Failed asserting that an array is empty.

/home/sb/EmptyTest.php:6

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

assertEqualXMLStructure()

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

Reports an error identified by `$message` if the XML Structure of the DOMElement in `$actualElement` is not equal to the XML structure of the DOMElement in `$expectedElement`.

例A.10 Usage of assertEqualXMLStructure()

```php
<?php
class EqualXMLStructureTest extends PHPUnit_Framework_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 4.1.0 by Sebastian Bergmann.

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.phpunit EqualXMLStructureTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.75Mb

There were 4 failures:

1) EqualXMLStructureTest::testFailureWithDifferentNodeNames
Failed asserting that two strings are equal.
--- Expected
```

```
+++ Actual
@@ @@
-'foo'
+'bar'

/home/sb/EqualXMLStructureTest.php:9

2) EqualXMLStructureTest::testFailureWithDifferentNodeAttributes
Number of attributes on node "foo" does not match
Failed asserting that 0 matches expected 1.

/home/sb/EqualXMLStructureTest.php:22

3) EqualXMLStructureTest::testFailureWithDifferentChildrenCount
Number of child nodes of "foo" differs
Failed asserting that 1 matches expected 3.

/home/sb/EqualXMLStructureTest.php:35

4) EqualXMLStructureTest::testFailureWithDifferentChildren
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'bar'
+'baz'

/home/sb/EqualXMLStructureTest.php:48

FAILURES!
Tests: 4, Assertions: 8, Failures: 4.
```

assertEquals()

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

Reports an error identified by `$message` if the two variables `$expected` and `$actual` are not equal.

`assertNotEquals()` is the inverse of this assertion and takes the same arguments.

`assertAttributeEquals()` and `assertAttributeNotEquals()` are convenience wrappers that use a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.11 Usage of assertEquals()

```php
<?php
class EqualsTest extends PHPUnit_Framework_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 4.1.0 by Sebastian Bergmann.

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.phpunit EqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

More specialized comparisons are used for specific argument types for `$expected` and `$actual`, see below.

`assertEquals(float $expected, float $actual[, string $message = '', float $delta = 0])`

Reports an error identified by `$message` if the two floats `$expected` and `$actual` are not within `$delta` of each other.

Please read "What Every Computer Scientist Should Know About Floating-Point Arithmetic [http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html]" to understand why `$delta` is neccessary.

例A.12 Usage of assertEquals() with floats

```php
<?php
class EqualsTest extends PHPUnit_Framework_TestCase
{
    public function testSuccess()
    {
        $this->assertEquals(1.0, 1.1, '', 0.2);
    }

    public function testFailure()
    {
        $this->assertEquals(1.0, 1.1);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

.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.phpunit EqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

.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 = ''])`

Reports an error identified by `$message` if the uncommented canonical form of the XML documents represented by the two DOMDocument objects `$expected` and `$actual` are not equal.

例A.13 Usage of assertEquals() with DOMDocument objects

```php
<?php
class EqualsTest extends PHPUnit_Framework_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 4.1.0 by Sebastian Bergmann.

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.phpunit EqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 = ''])`

Reports an error identified by `$message` if the two objects `$expected` and `$actual` do not have equal attribute values.

例A.14 Usage of assertEquals() with objects

```php
<?php
class EqualsTest extends PHPUnit_Framework_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 4.1.0 by Sebastian Bergmann.

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.phpunit EqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 = ''])`

Reports an error identified by `$message` if the two arrays `$expected` and `$actual` are not equal.

例A.15 Usage of assertEquals() with arrays

```php
<?php
class EqualsTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertEquals(array('a', 'b', 'c'), array('a', 'c', 'd'));
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit EqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
     0 => 'a'
-    1 => 'b'
-    2 => 'c'
+    1 => 'c'
+    2 => 'd'
 )

/home/sb/EqualsTest.php:6

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

assertFalse()

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

Reports an error identified by `$message` if `$condition` is `TRUE`.

`assertNotFalse()` is the inverse of this assertion and takes the same arguments.

例A.16 Usage of assertFalse()

```
<?php
```

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit FalseTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) FalseTest::testFailure
Failed asserting that true is false.

/home/sb/FalseTest.php:6

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

assertFileEquals()

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

Reports an error identified by `$message` if the file specified by `$expected` does not have the same contents as the file specified by `$actual`.

`assertFileNotEquals()` is the inverse of this assertion and takes the same arguments.

例A.17 Usage of assertFileEquals()

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

```
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit FileEqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) FileEqualsTest::testFailure
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'expected
+'actual
 '

/home/sb/FileEqualsTest.php:6

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertFileExists()

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

Reports an error identified by `$message` if the file specified by `$filename` does not exist.

`assertFileNotExists()` is the inverse of this assertion and takes the same arguments.

例A.18 Usage of assertFileExists()

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

```
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit FileExistsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertGreaterThan()

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

Reports an error identified by `$message` if the value of `$actual` is not greater than the value of `$expected`.

`assertAttributeGreaterThan()` is a convenience wrapper that uses a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.19 Usage of assertGreaterThan()

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.
```

```
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.phpunit GreaterThanTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) GreaterThanTest::testFailure
Failed asserting that 1 is greater than 2.

/home/sb/GreaterThanTest.php:6

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

assertGreaterThanOrEqual()

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

Reports an error identified by `$message` if the value of `$actual` is not greater than or equal to the value of `$expected`.

`assertAttributeGreaterThanOrEqual()` is a convenience wrapper that uses a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.20 Usage of assertGreaterThanOrEqual()

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit GreaterThanOrEqualTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertInstanceOf()

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

Reports an error identified by `$message` if `$actual` is not an instance of `$expected`.

`assertNotInstanceOf()` is the inverse of this assertion and takes the same arguments.

`assertAttributeInstanceOf()` and `assertAttributeNotInstanceOf()` are convenience wrappers that can be applied to a `public`, `protected`, or `private` attribute of a class or object.

例A.21 Usage of assertInstanceOf()

```php
<?php
class InstanceOfTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertInstanceOf('RuntimeException', new Exception);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit InstanceOfTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InstanceOfTest::testFailure
Failed asserting that Exception Object (...) is an instance of class "RuntimeException".

/home/sb/InstanceOfTest.php:6

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

assertInternalType()

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

Reports an error identified by `$message` if `$actual` is not of the `$expected` type.

`assertNotInternalType()` is the inverse of this assertion and takes the same arguments.

`assertAttributeInternalType()` and `assertAttributeNotInternalType()` are convenience wrappers that can be applied to a `public`, `protected`, or `private` attribute of a class or object.

例A.22 Usage of assertInternalType()

```php
<?php
class InternalTypeTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertInternalType('string', 42);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InternalTypeTest::testFailure
Failed asserting that 42 is of type "string".

/home/sb/InternalTypeTest.php:6

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.phpunit InternalTypeTest
PHPUnit 4.1.0 by Sebastian Bergmann.
```

```
F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InternalTypeTest::testFailure
Failed asserting that 42 is of type "string".

/home/sb/InternalTypeTest.php:6

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

assertJsonFileEqualsJsonFile()

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

Reports an error identified by `$message` if the value of `$actualFile` does not match the value of `$expectedFile`.

例A.23 Usage of assertJsonFileEqualsJsonFile()

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonFileEqualsJsonFile::testFailure
Failed asserting that '{"Mascott":"Tux"}' matches JSON string "["Mascott", "Tux", "OS",

/home/sb/JsonFileEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit JsonFileEqualsJsonFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonFileEqualsJsonFile::testFailure
Failed asserting that '{"Mascott":"Tux"}' matches JSON string "["Mascott", "Tux", "OS",
```

```
/home/sb/JsonFileEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertJsonStringEqualsJsonFile()

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

Reports an error identified by `$message` if the value of `$actualJson` does not match the value of `$expectedFile`.

例A.24 Usage of assertJsonStringEqualsJsonFile()

```
<?php
class JsonStringEqualsJsonFileTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonFile(
          'path/to/fixture/file', json_encode(array("Mascott" => "ux"))
        );
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonFile::testFailure
Failed asserting that '{"Mascott":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/JsonStringEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit JsonStringEqualsJsonFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonFile::testFailure
Failed asserting that '{"Mascott":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/JsonStringEqualsJsonFileTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertJsonStringEqualsJsonString()

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

Reports an error identified by `$message` if the value of `$actualJson` does not match the value of `$expectedJson`.

例A.25 Usage of assertJsonStringEqualsJsonString()

```
<?php
class JsonStringEqualsJsonStringTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertJsonStringEqualsJsonString(
          json_encode(array("Mascott" => "Tux")), json_encode(array("Mascott" => "ux"))
        );
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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 (
 -    'Mascott' => 'Tux'
 +    'Mascott' => 'ux'
)

/home/sb/JsonStringEqualsJsonStringTest.php:5

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.phpunit JsonStringEqualsJsonStringTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 (
 -    'Mascott' => 'Tux'
 +    'Mascott' => 'ux'
)

/home/sb/JsonStringEqualsJsonStringTest.php:5
```

```
FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertLessThan()

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

Reports an error identified by `$message` if the value of `$actual` is not less than the value of `$expected`.

`assertAttributeLessThan()` is a convenience wrapper that uses a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.26 Usage of assertLessThan()

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

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit LessThanTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) LessThanTest::testFailure
Failed asserting that 2 is less than 1.

/home/sb/LessThanTest.php:6

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

assertLessThanOrEqual()

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

Reports an error identified by `$message` if the value of `$actual` is not less than or equal to the value of `$expected`.

`assertAttributeLessThanOrEqual()` is a convenience wrapper that uses a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.27 Usage of assertLessThanOrEqual()

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

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit LessThanOrEqualTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertNull()

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

Reports an error identified by `$message` if `$variable` is not `NULL`.

`assertNotNull()` is the inverse of this assertion and takes the same arguments.

例A.28 Usage of assertNull()

```
<?php
class NullTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertNull('foo');
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit NotNullTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NullTest::testFailure
Failed asserting that 'foo' is null.

/home/sb/NotNullTest.php:6

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

assertObjectHasAttribute()

`assertObjectHasAttribute(string $attributeName, object $object[, string $message = ''])`

Reports an error identified by `$message` if `$object->attributeName` does not exist.

`assertObjectNotHasAttribute()` is the inverse of this assertion and takes the same arguments.

例A.29 Usage of assertObjectHasAttribute()

```
<?php
class ObjectHasAttributeTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
```

```
        $this->assertObjectHasAttribute('foo', new stdClass);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit ObjectHasAttributeTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ObjectHasAttributeTest::testFailure
Failed asserting that object of class "stdClass" has attribute "foo".

/home/sb/ObjectHasAttributeTest.php:6

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

assertRegExp()

`assertRegExp(string $pattern, string $string[, string $message = ''])`

Reports an error identified by `$message` if `$string` does not match the regular expression `$pattern`.

`assertNotRegExp()` is the inverse of this assertion and takes the same arguments.

例A.30 Usage of assertRegExp()

```
<?php
class RegExpTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertRegExp('/foo/', 'bar');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.
```

```
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.phpunit RegExpTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) RegExpTest::testFailure
Failed asserting that 'bar' matches PCRE pattern "/foo/".

/home/sb/RegExpTest.php:6

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

assertStringMatchesFormat()

`assertStringMatchesFormat(string $format, string $string[, string $message = ''])`

Reports an error identified by `$message` if the `$string` does not match the `$format` string.

`assertStringNotMatchesFormat()` is the inverse of this assertion and takes the same arguments.

例A.31 Usage of assertStringMatchesFormat()

```php
<?php
class StringMatchesFormatTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormat('%i', 'foo');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit StringMatchesFormatTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

The format string may contain the following placeholders:

- `%e`: Represents a directory separator, for example `/` on Linux.
- `%s`: One or more of anything (character or white space) except the end of line character.
- `%S`: Zero or more of anything (character or white space) except the end of line character.
- `%a`: One or more of anything (character or white space) including the end of line character.
- `%A`: Zero or more of anything (character or white space) including the end of line character.
- `%w`: Zero or more white space characters.
- `%i`: A signed integer value, for example `+3142`, `-3142`.
- `%d`: An unsigned integer value, for example `123456`.
- `%x`: One or more hexadecimal character. That is, characters in the range `0-9`, `a-f`, `A-F`.
- `%f`: A floating point number, for example: `3.142`, `-3.142`, `3.142E-10`, `3.142e+10`.
- `%c`: A single character of any sort.

assertStringMatchesFormatFile()

`assertStringMatchesFormatFile(string $formatFile, string $string[, string $message = ''])`

Reports an error identified by `$message` if the `$string` does not match the contents of the `$formatFile`.

`assertStringNotMatchesFormatFile()` is the inverse of this assertion and takes the same arguments.

例A.32 Usage of assertStringMatchesFormatFile()

```
<?php
class StringMatchesFormatFileTest extends PHPUnit_Framework_TestCase
{
```

```
    public function testFailure()
    {
        $this->assertStringMatchesFormatFile('/path/to/expected.txt', 'foo');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit StringMatchesFormatFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringMatchesFormatFileTest::testFailure
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+
$/s".

/home/sb/StringMatchesFormatFileTest.php:6

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

assertSame()

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

Reports an error identified by `$message` if the two variables `$expected` and `$actual` do not have the same type and value.

`assertNotSame()` is the inverse of this assertion and takes the same arguments.

`assertAttributeSame()` and `assertAttributeNotSame()` are convenience wrappers that use a `public`, `protected`, or `private` attribute of a class or object as the actual value.

例A.33 Usage of assertSame()

```
<?php
class SameTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertSame('2204', 2204);
```

```
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit SameTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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 = ''])`

Reports an error identified by `$message` if the two variables `$expected` and `$actual` do not reference the same object.

例A.34 Usage of assertSame() with objects

```
<?php
class SameTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertSame(new stdClass, new stdClass);
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit SameTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertSelectCount()

`assertSelectCount(array $selector, integer $count, mixed $actual[, string $message = '', boolean $isHtml = TRUE])`

Reports an error identified by `$message` if the CSS selector `$selector` does not match `$count` elements in the DOMNode `$actual`.

`$count` can be one of the following types:

- `boolean`: Asserts for presence of elements matching the selector (`TRUE`) or absence of elements (`FALSE`).
- `integer`: Asserts the count of elements.
- `array`: Asserts that the count is in a range specified by using `<`, `>`, `<=`, and `>=` as keys.

例A.35 Usage of assertSelectCount()

```php
<?php
class SelectCountTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        $this->xml = new DomDocument;
        $this->xml->loadXML('<foo><bar/><bar/><bar/></foo>');
    }

    public function testAbsenceFailure()
    {
        $this->assertSelectCount('foo bar', FALSE, $this->xml);
    }

    public function testPresenceFailure()
    {
        $this->assertSelectCount('foo baz', TRUE, $this->xml);
    }

    public function testExactCountFailure()
    {
```

```
        $this->assertSelectCount('foo bar', 5, $this->xml);
    }

    public function testRangeFailure()
    {
        $this->assertSelectCount('foo bar', array('>'=>6, '<'=>8), $this->xml);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectCountTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectCountTest.php:12

2) SelectCountTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectCountTest.php:17

3) SelectCountTest::testExactCountFailure
Failed asserting that 3 matches expected 5.

/home/sb/SelectCountTest.php:22

4) SelectCountTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectCountTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelectCountTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectCountTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectCountTest.php:12

2) SelectCountTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectCountTest.php:17

3) SelectCountTest::testExactCountFailure
Failed asserting that 3 matches expected 5.

/home/sb/SelectCountTest.php:22
```

```
4) SelectCountTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectCountTest.php:27

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

assertSelectEquals()

`assertSelectEquals(array $selector, string $content, integer $count, mixed $actual[, string $message = '', boolean $isHtml = TRUE])`

Reports an error identified by `$message` if the CSS selector `$selector` does not match `$count` elements in the DOMNode `$actual` with the value `$content`.

`$count` can be one of the following types:

- `boolean`: Asserts for presence of elements matching the selector (`TRUE`) or absence of elements (`FALSE`).
- `integer`: Asserts the count of elements.
- `array`: Asserts that the count is in a range specified by using `<`, `>`, `<=`, and `>=` as keys.

例A.36 Usage of assertSelectEquals()

```php
<?php
class SelectEqualsTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        $this->xml = new DomDocument;
        $this->xml->loadXML('<foo><bar>Baz</bar><bar>Baz</bar></foo>');
    }

    public function testAbsenceFailure()
    {
        $this->assertSelectEquals('foo bar', 'Baz', FALSE, $this->xml);
    }

    public function testPresenceFailure()
    {
        $this->assertSelectEquals('foo bar', 'Bat', TRUE, $this->xml);
    }

    public function testExactCountFailure()
    {
        $this->assertSelectEquals('foo bar', 'Baz', 5, $this->xml);
    }

    public function testRangeFailure()
    {
        $this->assertSelectEquals('foo bar', 'Baz', array('>'=>6, '<'=>8), $this->xml);
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.
```

```
FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectEqualsTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectEqualsTest.php:12

2) SelectEqualsTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectEqualsTest.php:17

3) SelectEqualsTest::testExactCountFailure
Failed asserting that 2 matches expected 5.

/home/sb/SelectEqualsTest.php:22

4) SelectEqualsTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectEqualsTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelectEqualsTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectEqualsTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectEqualsTest.php:12

2) SelectEqualsTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectEqualsTest.php:17

3) SelectEqualsTest::testExactCountFailure
Failed asserting that 2 matches expected 5.

/home/sb/SelectEqualsTest.php:22

4) SelectEqualsTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectEqualsTest.php:27

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

assertSelectRegExp()

```
assertSelectRegExp(array $selector, string $pattern, integer $count,
mixed $actual[, string $message = '', boolean $isHtml = TRUE])
```

Reports an error identified by `$message` if the CSS selector `$selector` does not match `$count` elements in the DOMNode `$actual` with a value that matches `$pattern`.

`$count` can be one of the following types:

- `boolean`: Asserts for presence of elements matching the selector (`TRUE`) or absence of elements (`FALSE`).
- `integer`: Asserts the count of elements.
- `array`: Asserts that the count is in a range specified by using <, >, <=, and >= as keys.

例A.37 Usage of assertSelectRegExp()

```
<?php
class SelectRegExpTest extends PHPUnit_Framework_TestCase
{
    protected function setUp()
    {
        $this->xml = new DomDocument;
        $this->xml->loadXML('<foo><bar>Baz</bar><bar>Baz</bar></foo>');
    }

    public function testAbsenceFailure()
    {
        $this->assertSelectRegExp('foo bar', '/Ba.*/', FALSE, $this->xml);
    }

    public function testPresenceFailure()
    {
        $this->assertSelectRegExp('foo bar', '/B[oe]z]/', TRUE, $this->xml);
    }

    public function testExactCountFailure()
    {
        $this->assertSelectRegExp('foo bar', '/Ba.*/', 5, $this->xml);
    }

    public function testRangeFailure()
    {
        $this->assertSelectRegExp('foo bar', '/Ba.*/', array('>'=>6, '<'=>8), $this->xml
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectRegExpTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectRegExpTest.php:12

2) SelectRegExpTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectRegExpTest.php:17
```

```
3) SelectRegExpTest::testExactCountFailure
Failed asserting that 2 matches expected 5.

/home/sb/SelectRegExpTest.php:22

4) SelectRegExpTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectRegExpTest.php:27

FAILURES!
Tests: 4, Assertions: 4, Failures: 4.phpunit SelectRegExpTest
PHPUnit 4.1.0 by Sebastian Bergmann.

FFFF

Time: 0 seconds, Memory: 5.50Mb

There were 4 failures:

1) SelectRegExpTest::testAbsenceFailure
Failed asserting that true is false.

/home/sb/SelectRegExpTest.php:12

2) SelectRegExpTest::testPresenceFailure
Failed asserting that false is true.

/home/sb/SelectRegExpTest.php:17

3) SelectRegExpTest::testExactCountFailure
Failed asserting that 2 matches expected 5.

/home/sb/SelectRegExpTest.php:22

4) SelectRegExpTest::testRangeFailure
Failed asserting that false is true.

/home/sb/SelectRegExpTest.php:27

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

assertStringEndsWith()

`assertStringEndsWith(string $suffix, string $string[, string $message = ''])`

Reports an error identified by `$message` if the `$string` does not end with `$suffix`.

`assertStringEndsNotWith()` is the inverse of this assertion and takes the same arguments.

例A.38 Usage of assertStringEndsWith()

```php
<?php
class StringEndsWithTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertStringEndsWith('suffix', 'foo');
    }
```

```
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit StringEndsWithTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 1 second, Memory: 5.00Mb

There was 1 failure:

1) StringEndsWithTest::testFailure
Failed asserting that 'foo' ends with "suffix".

/home/sb/StringEndsWithTest.php:6

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

assertStringEqualsFile()

`assertStringEqualsFile(string $expectedFile, string $actualString[, string $message = ''])`

Reports an error identified by `$message` if the file specified by `$expectedFile` does not have `$actualString` as its contents.

`assertStringNotEqualsFile()` is the inverse of this assertion and takes the same arguments.

例A.39 Usage of assertStringEqualsFile()

```
<?php
class StringEqualsFileTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertStringEqualsFile('/home/sb/expected', 'actual');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit StringEqualsFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) StringEqualsFileTest::testFailure
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'expected
-'
+'actual'

/home/sb/StringEqualsFileTest.php:6

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

assertStringStartsWith()

`assertStringStartsWith(string $prefix, string $string[, string $message = ''])`

Reports an error identified by `$message` if the `$string` does not start with `$prefix`.

`assertStringStartsNotWith()` is the inverse of this assertion and takes the same arguments.

例A.40 Usage of assertStringStartsWith()

```php
<?php
class StringStartsWithTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertStringStartsWith('prefix', 'foo');
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit StringStartsWithTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

assertTag()

`assertTag(array $matcher, string $actual[, string $message = '', boolean $isHtml = TRUE])`

Reports an error identified by `$message` if `$actual` is not matched by the `$matcher`.

`$matcher` is an associative array that specifies the match criteria for the assertion:

- `id`: The node with the given `id` attribute must match the corresponsing value.
- `tag`: The node type must match the corresponding value.
- `attributes`: The node's attributes must match the corresponsing values in the `$attributes` associative array.
- `content`: The text content must match the given value.
- `parent`: The node's parent must match the `$parent` associative array.
- `child`: At least one of the node's immediate children must meet the criteria described by the `$child` associative array.
- `ancestor`: At least one of the node's ancestors must meet the criteria described by the `$ancestor` associative array.
- `descendant`: At least one of the node's descendants must meet the criteria described by the `$descendant` associative array.
- `children`: Associative array for counting children of a node.
  - `count`: The number of matching children must be equal to this number.

- `less_than`: The number of matching children must be less than this number.
- `greater_than`: The number of matching children must be greater than this number.
- `only`: Another associative array consisting of the keys to use to match on the children, and only matching children will be counted.

`assertNotTag()` is the inverse of this assertion and takes the same arguments.

例A.41 Usage of assertTag()

```
<?php
// Matcher that asserts that there is an element with an id="my_id".
$matcher = array('id' => 'my_id');

// Matcher that asserts that there is a "span" tag.
$matcher = array('tag' => 'span');

// Matcher that asserts that there is a "span" tag with the content
// "Hello World".
$matcher = array('tag' => 'span', 'content' => 'Hello World');

// Matcher that asserts that there is a "span" tag with content matching the
// regular expression pattern.
$matcher = array('tag' => 'span', 'content' => 'regexp:/Try P(HP|ython)/');

// Matcher that asserts that there is a "span" with an "list" class attribute.
$matcher = array(
  'tag'        => 'span',
  'attributes' => array('class' => 'list')
);

// Matcher that asserts that there is a "span" inside of a "div".
$matcher = array(
  'tag'    => 'span',
  'parent' => array('tag' => 'div')
);

// Matcher that asserts that there is a "span" somewhere inside a "table".
$matcher = array(
  'tag'      => 'span',
  'ancestor' => array('tag' => 'table')
);

// Matcher that asserts that there is a "span" with at least one "em" child.
$matcher = array(
  'tag'   => 'span',
  'child' => array('tag' => 'em')
);

// Matcher that asserts that there is a "span" containing a (possibly nested)
// "strong" tag.
$matcher = array(
  'tag'        => 'span',
  'descendant' => array('tag' => 'strong')
);

// Matcher that asserts that there is a "span" containing 5-10 "em" tags as
// immediate children.
$matcher = array(
  'tag'      => 'span',
  'children' => array(
    'less_than'    => 11,
```

```
    'greater_than' => 4,
    'only'         => array('tag' => 'em')
  )
);

// Matcher that asserts that there is a "div", with an "ul" ancestor and a "li"
// parent (with class="enum"), and containing a "span" descendant that contains
// an element with id="my_test" and the text "Hello World".
$matcher = array(
  'tag'        => 'div',
  'ancestor'   => array('tag' => 'ul'),
  'parent'     => array(
    'tag'        => 'li',
    'attributes' => array('class' => 'enum')
  ),
  'descendant' => array(
    'tag'   => 'span',
    'child' => array(
      'id'      => 'my_test',
      'content' => 'Hello World'
    )
  )
);

// Use assertTag() to apply a $matcher to a piece of $html.
$this->assertTag($matcher, $html);

// Use assertTag() to apply a $matcher to a piece of $xml.
$this->assertTag($matcher, $xml, '', FALSE);
?>
```

assertThat()

More complex assertions can be formulated using the `PHPUnit_Framework_Constraint` classes. They can be evaluated using the `assertThat()` method. 例A.42「Usage of assertThat()」 shows how the `logicalNot()` and `equalTo()` constraints can be used to express the same assertion as `assertNotEquals()`.

`assertThat(mixed $value, PHPUnit_Framework_Constraint $constraint[, $message = ''])`

Reports an error identified by `$message` if the `$value` does not match the `$constraint`.

例A.42 Usage of assertThat()

```
<?php
class BiscuitTest extends PHPUnit_Framework_TestCase
{
    public function testEquals()
    {
        $theBiscuit = new Biscuit('Ginger');
        $myBiscuit  = new Biscuit('Ginger');

        $this->assertThat(
          $theBiscuit,
          $this->logicalNot(
            $this->equalTo($myBiscuit)
          )
        );
    }
}
?>
```

表A.1「Constraints」 shows the available `PHPUnit_Framework_Constraint` classes.

表A.1 Constraints

Constraint	Meaning
`PHPUnit_ Framework_ Constraint_ Attribute attribute(PHPUnit_ Framework_ Constraint $constraint, $attributeName)`	Constraint that applies another constraint to an attribute of a class or an object.
`PHPUnit_ Framework_ Constraint_ IsAnything anything()`	Constraint that accepts any input value.
`PHPUnit_ Framework_ Constraint_ ArrayHasKey arrayHasKey(mixed $key)`	Constraint that asserts that the array it is evaluated for has a given key.
`PHPUnit_ Framework_ Constraint_ TraversableContains contains(mixed $value)`	Constraint that asserts that the `array` or object that implements the `Iterator` interface it is evaluated for contains a given value.
`PHPUnit_ Framework_ Constraint_ TraversableContainsOnly containsOnly(string $type)`	Constraint that asserts that the `array` or object that implements the `Iterator` interface it is evaluated for contains only values of a given type.
`PHPUnit_ Framework_ Constraint_ TraversableContainsOnly containsOnlyInstancesOf(string $classname)`	Constraint that asserts that the `array` or object that implements the `Iterator` interface it is evaluated for contains only instances of a given classname.
`PHPUnit_ Framework_ Constraint_ IsEqual equalTo($value, $delta = 0, $maxDepth = 10)`	Constraint that checks if one value is equal to another.
`PHPUnit_ Framework_ Constraint_ Attribute attributeEqualTo($attributeName, $value, $delta = 0, $maxDepth = 10)`	Constraint that checks if a value is equal to an attribute of a class or of an object.
`PHPUnit_ Framework_ Constraint_ FileExists fileExists()`	Constraint that checks if the file(name) that it is evaluated for exists.
`PHPUnit_ Framework_ Constraint_ GreaterThan greaterThan(mixed $value)`	Constraint that asserts that the value it is evaluated for is greater than a given value.
`PHPUnit_ Framework_ Constraint_ Or greaterThanOrEqual(mixed $value)`	Constraint that asserts that the value it is evaluated for is greater than or equal to a given value.
`PHPUnit_ Framework_ Constraint_ ClassHasAttribute classHasAttribute(string $attributeName)`	Constraint that asserts that the class it is evaluated for has a given attribute.
`PHPUnit_ Framework_ Constraint_ ClassHasStaticAttribute classHasStaticAttribute(string $attributeName)`	Constraint that asserts that the class it is evaluated for has a given static attribute.
`PHPUnit_ Framework_ Constraint_ ObjectHasAttribute hasAttribute(string $attributeName)`	Constraint that asserts that the object it is evaluated for has a given attribute.

Constraint	Meaning
`PHPUnit_ Framework_ Constraint_ IsIdentical identicalTo(mixed $value)`	Constraint that asserts that one value is identical to another.
`PHPUnit_ Framework_ Constraint_ IsFalse isFalse()`	Constraint that asserts that the value it is evaluated is `FALSE`.
`PHPUnit_ Framework_ Constraint_ IsInstanceOf isInstanceOf(string $className)`	Constraint that asserts that the object it is evaluated for is an instance of a given class.
`PHPUnit_ Framework_ Constraint_ IsNull isNull()`	Constraint that asserts that the value it is evaluated is `NULL`.
`PHPUnit_ Framework_ Constraint_ IsTrue isTrue()`	Constraint that asserts that the value it is evaluated is `TRUE`.
`PHPUnit_ Framework_ Constraint_ IsType isType(string $type)`	Constraint that asserts that the value it is evaluated for is of a specified type.
`PHPUnit_ Framework_ Constraint_ LessThan lessThan(mixed $value)`	Constraint that asserts that the value it is evaluated for is smaller than a given value.
`PHPUnit_ Framework_ Constraint_ Or lessThanOrEqual(mixed $value)`	Constraint that asserts that the value it is evaluated for is smaller than or equal to a given value.
`logicalAnd()`	Logical AND.
`logicalNot(PHPUnit_ Framework_ Constraint $constraint)`	Logical NOT.
`logicalOr()`	Logical OR.
`logicalXor()`	Logical XOR.
`PHPUnit_ Framework_ Constraint_ PCREMatch matchesRegularExpression(string $pattern)`	Constraint that asserts that the string it is evaluated for matches a regular expression.
`PHPUnit_ Framework_ Constraint_ StringContains stringContains(string $string, bool $case)`	Constraint that asserts that the string it is evaluated for contains a given string.
`PHPUnit_ Framework_ Constraint_ StringEndsWith stringEndsWith(string $suffix)`	Constraint that asserts that the string it is evaluated for ends with a given suffix.
`PHPUnit_ Framework_ Constraint_ StringStartsWith stringStartsWith(string $prefix)`	Constraint that asserts that the string it is evaluated for starts with a given prefix.

assertTrue()

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

Reports an error identified by `$message` if `$condition` is `FALSE`.

`assertNotTrue()` is the inverse of this assertion and takes the same arguments.

例A.43 Usage of assertTrue()

```
<?php
```

```
class TrueTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertTrue(FALSE);
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit TrueTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) TrueTest::testFailure
Failed asserting that false is true.

/home/sb/TrueTest.php:6

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

assertXmlFileEqualsXmlFile()

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

Reports an error identified by `$message` if the XML document in `$actualFile` is not equal to the XML document in `$expectedFile`.

`assertXmlFileNotEqualsXmlFile()` is the inverse of this assertion and takes the same arguments.

例A.44 Usage of assertXmlFileEqualsXmlFile()

```
<?php
class XmlFileEqualsXmlFileTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertXmlFileEqualsXmlFile(
          '/home/sb/expected.xml', '/home/sb/actual.xml');
```

```
    }
}
?>
```

```
PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit XmlFileEqualsXmlFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) XmlFileEqualsXmlFileTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>

/home/sb/XmlFileEqualsXmlFileTest.php:7

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```

assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])`

Reports an error identified by `$message` if the XML document in `$actualXml` is not equal to the XML document in `$expectedFile`.

`assertXmlStringNotEqualsXmlFile()` is the inverse of this assertion and takes the same arguments.

例A.45 Usage of assertXmlStringEqualsXmlFile()

```
<?php
class XmlStringEqualsXmlFileTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlFile(
          '/home/sb/expected.xml', '<foo><baz/></foo>');
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit XmlStringEqualsXmlFileTest
PHPUnit 4.1.0 by Sebastian Bergmann.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) XmlStringEqualsXmlFileTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>

/home/sb/XmlStringEqualsXmlFileTest.php:7

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

assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])`

Reports an error identified by `$message` if the XML document in `$actualXml` is not equal to the XML document in `$expectedXml`.

`assertXmlStringNotEqualsXmlString()` is the inverse of this assertion and takes the same arguments.

例A.46 Usage of assertXmlStringEqualsXmlString()

```
<?php
class XmlStringEqualsXmlStringTest extends PHPUnit_Framework_TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlString(
          '<foo><bar/></foo>', '<foo><baz/></foo>');
    }
}
?>
```

```

PHPUnit 4.1.0 by Sebastian Bergmann.

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.phpunit XmlStringEqualsXmlStringTest
PHPUnit 4.1.0 by Sebastian Bergmann.

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.
```

付録B 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。

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。

@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。

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @after
     */
    public function tearDownSomeFixtures()
    {
        // ...
    }

    /**
     * @after
     */
    public function tearDownSomeOtherFixtures()
    {
        // ...
    }
}
```

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @afterClass
     */
    public static function tearDownSomeSharedFixtures()
    {
        // ...
```

```
    }

    /**
     * @afterClass
     */
    public static function tearDownSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

@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。

```
/**
 * @backupGlobals disabled
 */
class MyTest extends PHPUnit_Framework_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。

```
/**
 * @backupGlobals disabled
 */
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testThatInteractsWithGlobalVariables()
    {
        // ...
    }
}
```

@backupStaticAttributes

クラスの静的属性の保存や復元を、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。

```
/**
 * @backupStaticAttributes disabled
 */
class MyTest extends PHPUnit_Framework_TestCase
{
    // ...
}
```

`@backupStaticAttributes` 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。

```
/**
 * @backupStaticAttributes disabled
```

```
 */
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @backupStaticAttributes enabled
     */
    public function testThatInteractsWithStaticAttributes()
    {
        // ...
    }
}
```

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @before
     */
    public function setupSomeFixtures()
    {
        // ...
    }

    /**
     * @before
     */
    public function setupSomeOtherFixtures()
    {
        // ...
    }
}
```

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @beforeClass
     */
    public static function setUpSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @beforeClass
     */
    public static function setUpSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

@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。

@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->assertEquals(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。

表B.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」に @covers アノテーションの構文を示します。

表B.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。

アノテーション	説明
@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。

@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。 例B.1「@coversDefaultClass を使ったアノテーションの短縮」 を参照ください。

例B.1 @coversDefaultClass を使ったアノテーションの短縮

```
<?php
/**
 * @coversDefaultClass \Foo\CoveredClass
 */
class CoversDefaultClassTest extends PHPUnit_Framework_TestCase
{
    /**
     * @covers ::publicMethod
     */
    public function testSomething()
    {
        $o = new Foo\CoveredClass;
        $o->publicMethod();
    }
}
?>
```

@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 例11.3「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」 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。

@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 (例2.5「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。

@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。 例2.2「@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。

@expectedException

例2.9「@expectedException アノテーションの使用法」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。

@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。

```
class MyTest extends PHPUnit_Framework_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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
      * @expectedException     MyException
      * @expectedExceptionCode MyClass::ERRORCODE
      */
    public function testExceptionHasErrorcode20()
    {
      throw new MyException('Some Message', 20);
    }
}
class MyClass
{
    const ERRORCODE = 20;
}
```

@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。

```
class MyTest extends PHPUnit_Framework_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。

```
class MyTest extends PHPUnit_Framework_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。

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
    }

    /**
     * @group regresssion
     * @group bug2204
     */
    public function testSomethingElse()
    {
    }
}
```

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。

@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。

@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。

@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。

```
class MyTest extends PHPUnit_Framework_TestCase
{
    /**
     * @runInSeparateProcess
     * @preserveGlobalState disabled
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

@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 表7.3「@requires の例用例」 を参照ください。

@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。

```
/**
 * @runTestsInSeparateProcesses
 */
class MyTest extends PHPUnit_Framework_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。

@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。

```
class MyTest extends PHPUnit_Framework_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。

@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。

注記

デフォルトでは、`@medium` あるいは `@large` のどちらかのアノテーションがないテストはすべて small と見なされます。 しかし、`--group` およびそれに関連するオプションは、 明示的なアノテーションで `small` とマークしているテストだけを small と見なすことに注意しましょう。

@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->assertEquals(0, $this->ba->getBalance());
}
```

@testdox

@ticket

@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 「Unintentionally Covered Code」 を参照ください。

付録C XML 設定ファイル

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="http://schema.phpunit.de/4.1/phpunit.xsd"
         backupGlobals="true"
         backupStaticAttributes="false"
         <!--bootstrap="/path/to/bootstrap.php"-->
         cacheTokens="false"
         colors="false"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         forceCoversAnnotation="false"
         mapTestClassNameToCoveredClassName="false"
         printerClass="PHPUnit_TextUI_ResultPrinter"
         <!--printerFile="/path/to/ResultPrinter.php"-->
         processIsolation="false"
         stopOnError="false"
         stopOnFailure="false"
         stopOnIncomplete="false"
         stopOnSkipped="false"
         testSuiteLoaderClass="PHPUnit_Runner_StandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
         timeoutForSmallTests="1"
         timeoutForMediumTests="10"
         timeoutForLargeTests="60"
         strict="false"
         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「Command-Line switches」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 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 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	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 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	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 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。

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/f
      <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。

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`

コードカバレッジ対象のファイルの追加や除外

`<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ブラックリストとホワイトリストを設定します。

```
<filter>
  <blacklist>
    <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>
  </blacklist>
  <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>
```

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" charset="UTF-8"
       highlight="false" 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="json" target="/tmp/logfile.json"/>
  <log type="tap" target="/tmp/logfile.tap"/>
  <log type="junit" target="/tmp/logfile.xml" logIncompleteSkipped="false"/>
  <log type="testdox-html" target="/tmp/testdox.html"/>
  <log type="testdox-text" target="/tmp/testdox.txt"/>
</logging>
```

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`
- `--log-json /tmp/logfile.json`
- `> /tmp/logfile.txt`

- `--log-tap /tmp/logfile.tap`
- `--log-junit /tmp/logfile.xml`
- `--testdox-html /tmp/testdox.html`
- `--testdox-text /tmp/testdox.txt`

`charset`、`highlight`、`lowUpperBound`、`highLowerBound`、`logIncompleteSkipped` 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。

- `charset`: 生成する html ページで使う文字セット。
- `highlight`: `true` にすると、カバレッジレポート内のコードにシンタックスハイライト処理を施します。
- `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.

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(
  array('Sebastian'),
  22,
  'April',
  19.78,
  NULL,
  new stdClass
);
```

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';
```

Selenium RC の設定ブラウザ

<selenium> 要素とその子要素である <browser> を使って、 Selenium RC サーバのリストを設定します。

```
<selenium>
  <browser name="Firefox on Linux"
           browser="*firefox /usr/lib/firefox/firefox-bin"
           host="my.linux.box"
           port="4444"
           timeout="30000"/>
</selenium>
```

上の XML 設定は、次の PHP コードに対応します。

```
class WebTest extends PHPUnit_Extensions_SeleniumTestCase
{
    public static $browsers = array(
      array(
        'name'    => 'Firefox on Linux',
        'browser' => '*firefox /usr/lib/firefox/firefox-bin',
        'host'    => 'my.linux.box',
        'port'    => 4444,
        'timeout' => 30000
      )
    );

    // ...
```

```
}
```

付録D アップグレード

Upgrading from PHPUnit 3.7 to PHPUnit 4.0

- The limited support for stubbing and mocking static methods [http://sebastian-bergmann.de/blog/883-Stubbing-and-Mocking-Static-Methods.html] that was introduced in PHPUnit 3.5 has been removed. This feature only worked when the stubbed or mocked static method was invoked from another method of the same class. We believe that the limited use of this feature did not justify the increased complexity in the test doubles code generator it incurred. We apologize for any inconvenience this removal may cause and encourage refactoring the code under test to not require this feature for testing.
- The `addRiskyTest()` was added to the `PHPUnit_Framework_TestListener` interface. Classes that implement this interface have to implement this new method. This is the reason why PHPStorm 7 is not compatible with PHPUnit 4, for instance.
- The fixes for #552 [https://github.com/sebastianbergmann/phpunit/issues/552], #573 [https://github.com/sebastianbergmann/phpunit/issues/573], and #582 [https://github.com/sebastianbergmann/phpunit/issues/582] required a change to how relative paths are resolved for PHPUnit's XML configuration file. All relative paths in a configuration file are now resolved relative to that configuration file. When upgrading, you may need to update relative paths for the following configurations `testSuiteLoaderFile`, `printerFile`, `testsuites/file`, and `testsuites/exclude`.
- The numeric comparator is no longer invoked when provided with two strings [https://github.com/sebastianbergmann/phpunit/commit/f5df97cda0b25f2b7368395344da095ac529de62].

Please note that starting with PHPUnit 4.0.0 the PEAR package of PHPUnit is merely a distribution mechanism for the PHP Archive (PHAR) and that many of PHPUnit's dependencies will no longer be released individually via PEAR. We will eventually stop making releases of PHPUnit available via PEAR altogether.

Please note that using the PEAR installer to update from PHPUnit 3.7 to PHPUnit 4.0 will leave stale source files from previous versions of PHPUnit's dependencies (PHP_CodeCoverage, PHPUnit_MockObject, ...) behind in your PHP environment's PEAR directory. It is advised to uninstall the respective PEAR packages.

Upgrading from PHPUnit 4.0 to PHPUnit 4.1

付録E 目次

索引

シンボル

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

W

X

付録F 参考文献

[Astels2003] *Test Driven Development*. Astels David [FAMILY Given]. 製作著作 © 2003. Prentice Hall. ISBN 0131016490.

[Beck2002] *Test Driven Development by Example*. Beck Kent [FAMILY Given]. 製作著作 © 2002. Addison-Wesley. ISBN 0-321-14653-0.

[Beck2002-ja] テスト駆動開発入門. Beck Kent [FAMILY Given]. 製作著作 © 2003. ピアソンエデュケーション. ISBN 4894717115.

[Meszaros2007] *xUnit Test Patterns: Refactoring Test Code*. Meszaros Gerard [FAMILY Given]. 製作著作 © 2007. Addison-Wesley. ISBN 978-0131495050.

付録G 著作権

```
a. "Adaptation" means a work based upon the Work, or upon the
   Work and other pre-existing works, such as a translation,
   adaptation, derivative work, arrangement of music or other
   alterations of a literary or artistic work, or phonogram or
   performance and includes cinematographic adaptations or any
   other form in which the Work may be recast, transformed, or
   adapted including in any form recognizably derived from the
   original, except that a work that constitutes a Collection
   will not be considered an Adaptation for the purpose of this
   License. For the avoidance of doubt, where the Work is a
   musical work, performance or phonogram, the synchronization of
   the Work in timed-relation with a moving image ("synching")
   will be considered an Adaptation for the purpose of this
   License.

b. "Collection" means a collection of literary or artistic works,
   such as encyclopedias and anthologies, or performances,
   phonograms or broadcasts, or other works or subject matter
   other than works listed in Section 1(f) below, which, by
   reason of the selection and arrangement of their contents,
   constitute intellectual creations, in which the Work is
   included in its entirety in unmodified form along with one or
   more other contributions, each constituting separate and
   independent works in themselves, which together are assembled
   into a collective whole. A work that constitutes a Collection
   will not be considered an Adaptation (as defined above) for
   the purposes of this License.

c. "Distribute" means to make available to the public the
   original and copies of the Work or Adaptation, as appropriate,
   through sale or other transfer of ownership.

d. "Licensor" means the individual, individuals, entity or
   entities that offer(s) the Work under the terms of this License.

e. "Original Author" means, in the case of a literary or artistic
   work, the individual, individuals, entity or entities who
   created the Work or if no individual or entity can be
   identified, the publisher; and in addition (i) in the case of
   a performance the actors, singers, musicians, dancers, and
   other persons who act, sing, deliver, declaim, play in,
   interpret or otherwise perform literary or artistic works or
   expressions of folklore; (ii) in the case of a phonogram the
   producer being the person or legal entity who first fixes the
   sounds of a performance or other sounds; and, (iii) in the
   case of broadcasts, the organization that transmits the
   broadcast.

f. "Work" means the literary and/or artistic work offered under
   the terms of this License including without limitation any
   production in the literary, scientific and artistic domain,
   whatever may be the mode or form of its expression including
   digital form, such as a book, pamphlet and other writing; a
   lecture, address, sermon or other work of the same nature; a
   dramatic or dramatico-musical work; a choreographic work or
   entertainment in dumb show; a musical composition with or
   without words; a cinematographic work to which are assimilated
   works expressed by a process analogous to cinematography; a
   work of drawing, painting, architecture, sculpture, engraving
   or lithography; a photographic work to which are assimilated
   works expressed by a process analogous to photography; a work
   of applied art; an illustration, map, plan, sketch or three-
   dimensional work relative to geography, topography,
```

```
      architecture or science; a performance; a broadcast; a
      phonogram; a compilation of data to the extent it is protected
      as a copyrightable work; or a work performed by a variety or
      circus performer to the extent it is not otherwise considered
      a literary or artistic work.

   g. "You" means an individual or entity exercising rights under
      this License who has not previously violated the terms of
      this License with respect to the Work, or who has received
      express permission from the Licensor to exercise rights under
      this License despite a previous violation.

   h. "Publicly Perform" means to perform public recitations of the
      Work and to communicate to the public those public
      recitations, by any means or process, including by wire or
      wireless means or public digital performances; to make
      available to the public Works in such a way that members of
      the public may access these Works from a place and at a place
      individually chosen by them; to perform the Work to the public
      by any means or process and the communication to the public of
      the performances of the Work, including by public digital
      performance; to broadcast and rebroadcast the Work by any
      means including signs, sounds or images.

   i. "Reproduce" means to make copies of the Work by any means
      including without limitation by sound or visual recordings and
      the right of fixation and reproducing fixations of the Work,
      including storage of a protected performance or phonogram in
      digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to
   reduce, limit, or restrict any uses free from copyright or rights
   arising from limitations or exceptions that are provided for in
   connection with the copyright protection under copyright law or
   other applicable laws.

3. License Grant. Subject to the terms and conditions of this
   License, Licensor hereby grants You a worldwide, royalty-free,
   non-exclusive, perpetual (for the duration of the applicable
   copyright) license to exercise the rights in the Work as stated
   below:

   a. to Reproduce the Work, to incorporate the Work into one or
      more Collections, and to Reproduce the Work as incorporated
      in the Collections;

   b. to create and Reproduce Adaptations provided that any such
      Adaptation, including any translation in any medium, takes
      reasonable steps to clearly label, demarcate or otherwise
      identify that changes were made to the original Work. For
      example, a translation could be marked "The original work was
      translated from English to Spanish," or a modification could
      indicate "The original work has been modified.";

   c. to Distribute and Publicly Perform the Work including as
      incorporated in Collections; and,

   d. to Distribute and Publicly Perform Adaptations.

   e. For the avoidance of doubt:

      i. Non-waivable Compulsory License Schemes. In those
         jurisdictions in which the right to collect royalties
         through any statutory or compulsory licensing scheme cannot
```

```
          be waived, the Licensor reserves the exclusive right to
          collect such royalties for any exercise by You of the
          rights granted under this License;

      ii. Waivable Compulsory License Schemes. In those
          jurisdictions in which the right to collect royalties
          through any statutory or compulsory licensing scheme can
          be waived, the Licensor waives the exclusive right to
          collect such royalties for any exercise by You of the
          rights granted under this License; and,

      iii. Voluntary License Schemes. The Licensor waives the right
           to collect royalties, whether individually or, in the
           event that the Licensor is a member of a collecting
           society that administers voluntary licensing schemes, via
           that society, from any exercise by You of the rights
           granted under this License.

The above rights may be exercised in all media and formats whether
now known or hereafter devised. The above rights include the right
to make such modifications as are technically necessary to exercise
the rights in other media and formats. Subject to Section 8(f), all
rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly
   made subject to and limited by the following restrictions:

   a. You may Distribute or Publicly Perform the Work only under the
      terms of this License. You must include a copy of, or the
      Uniform Resource Identifier (URI) for, this License with every
      copy of the Work You Distribute or Publicly Perform. You may
      not offer or impose any terms on the Work that restrict the
      terms of this License or the ability of the recipient of the
      Work to exercise the rights granted to that recipient under
      the terms of the License. You may not sublicense the Work. You
      must keep intact all notices that refer to this License and to
      the disclaimer of warranties with every copy of the Work You
      Distribute or Publicly Perform. When You Distribute or
      Publicly Perform the Work, You may not impose any effective
      technological measures on the Work that restrict the ability
      of a recipient of the Work from You to exercise the rights
      granted to that recipient under the terms of the License. This
      Section 4(a) applies to the Work as incorporated in a
      Collection, but this does not require the Collection apart
      from the Work itself to be made subject to the terms of this
      License. If You create a Collection, upon notice from any
      Licensor You must, to the extent practicable, remove from the
      Collection any credit as required by Section 4(b), as
      requested. If You create an Adaptation, upon notice from any
      Licensor You must, to the extent practicable, remove from the
      Adaptation any credit as required by Section 4(b), as requested.

   b. If You Distribute, or Publicly Perform the Work or any
      Adaptations or Collections, You must, unless a request has
      been made pursuant to Section 4(a), keep intact all copyright
      notices for the Work and provide, reasonable to the medium or
      means You are utilizing: (i) the name of the Original Author
      (or pseudonym, if applicable) if supplied, and/or if the
      Original Author and/or Licensor designate another party or
      parties (e.g., a sponsor institute, publishing entity,
      journal) for attribution ("Attribution Parties") in Licensor's
      copyright notice, terms of service or by other reasonable
      means, the name of such party or parties; (ii) the title of
      the Work if supplied; (iii) to the extent reasonably
```

practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv), consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4 (b) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

c. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate

```
      automatically upon any breach by You of the terms of this
      License. Individuals or entities who have received Adaptations
      or Collections from You under this License, however, will not
      have their licenses terminated provided such individuals or
      entities remain in full compliance with those licenses.
      Sections 1, 2, 5, 6, 7, and 8 will survive any termination of
      this License.

   b. Subject to the above terms and conditions, the license granted
      here is perpetual (for the duration of the applicable
      copyright in the Work). Notwithstanding the above, Licensor
      reserves the right to release the Work under different license
      terms or to stop distributing the Work at any time; provided,
      however that any such election will not serve to withdraw this
      License (or any other license that has been, or is required to
      be, granted under the terms of this License), and this License
      will continue in full force and effect unless terminated as
      stated above.

8. Miscellaneous

   a. Each time You Distribute or Publicly Perform the Work or a
      Collection, the Licensor offers to the recipient a license to
      the Work on the same terms and conditions as the license
      granted to You under this License.

   b. Each time You Distribute or Publicly Perform an Adaptation,
      Licensor offers to the recipient a license to the original
      Work on the same terms and conditions as the license granted
      to You under this License.

   c. If any provision of this License is invalid or unenforceable
      under applicable law, it shall not affect the validity or
      enforceability of the remainder of the terms of this License,
      and without further action by the parties to this agreement,
      such provision shall be reformed to the minimum extent
      necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived
      and no breach consented to unless such waiver or consent shall
      be in writing and signed by the party to be charged with such
      waiver or consent.

   e. This License constitutes the entire agreement between the
      parties with respect to the Work licensed here. There are no
      understandings, agreements or representations with respect to
      the Work not specified here. Licensor shall not be bound by
      any additional provisions that may appear in any communication
      from You. This License may not be modified without the mutual
      written agreement of the Licensor and You.

   f. The rights granted under, and the subject matter referenced,
      in this License were drafted utilizing the terminology of the
      Berne Convention for the Protection of Literary and Artistic
      Works (as amended on September 28, 1979), the Rome Convention
      of 1961, the WIPO Copyright Treaty of 1996, the WIPO
      Performances and Phonograms Treaty of 1996 and the Universal
      Copyright Convention (as revised on July 24, 1971). These
      rights and subject matter take effect in the relevant
      jurisdiction in which the License terms are sought to be
      enforced according to the corresponding provisions of the
      implementation of those treaty provisions in the applicable
      national law. If the standard suite of rights granted under
      applicable copyright law includes additional rights not
```