PHPUnit Manual

Release latest

Sebastian Bergmann

out 17, 2018

Sumário

1 Instalando PHPUnit **3**
- 1.1 Requisitos . . . 3
- 1.2 PHP Archive (PHAR) . . . 3
  - 1.2.1 Windows . . . 4
  - 1.2.2 Verificando lançamentos do PHAR PHPUnit . . . 4
- 1.3 Composer . . . 6
- 1.4 Pacotes opcionais . . . 6

2 Escrevendo Testes para o PHPUnit **9**
- 2.1 Dependências de Testes . . . 10
- 2.2 Provedores de Dados . . . 13
- 2.3 Testando Exceções . . . 18
- 2.4 Testando Erros PHP . . . 19
- 2.5 Testando Saídas . . . 21
- 2.6 Saída de Erro . . . 22
  - 2.6.1 Casos Extremos . . . 24

3 O executor de testes em linha-de-comando **27**
- 3.1 Opções de linha-de-comando . . . 28

4 Ambientes **35**
- 4.1 Mais setUp() que tearDown() . . . 38
- 4.2 Variações . . . 38
- 4.3 Compartilhando Ambientes . . . 38
- 4.4 Estado Global . . . 39

5 Organizando Testes **41**
- 5.1 Compondo uma Suíte de Testes usando o Sistema de Arquivos . . . 41
- 5.2 Compondo uma Suíte de Testes Usando Configuração XML . . . 42

6 Testes arriscados **45**
- 6.1 Testes Inúteis . . . 45
- 6.2 Cobertura de Código Involuntária . . . 45
- 6.3 Saída Durante a Execução de Teste . . . 45
- 6.4 Tempo de Espera de Execução de Teste . . . 46
- 6.5 Manipulação do Estado Global . . . 46

7 Testes Incompletos e Pulados **47**
7.1 Testes Incompletos . . . 47
7.2 Pulando Testes . . . 48
7.3 Pulando Testes usando @requires . . . 49

8 Testando Bancos de Dados **51**
8.1 Fornecedores Suportados para Testes de Banco de Dados . . . 51
8.2 Dificuldades em Testes de Bancos de Dados . . . 52
8.3 Os quatro estágios dos testes com banco de dados . . . 52
8.3.1 1. Limpar o Banco de Dados . . . 53
8.3.2 2. Configurar o ambiente . . . 53
8.3.3 3–5. Executar Teste, Verificar resultado e Desmontar (Teardown) . . . 53
8.4 Configuração de um Caso de Teste de Banco de Dados do PHPUnit . . . 53
8.4.1 Implementando getConnection() . . . 54
8.4.2 Implementando getDataSet() . . . 54
8.4.3 E quanto ao Esquema do Banco de Dados (DDL)? . . . 55
8.4.4 Dica: Use seu próprio Caso Abstrato de Teste de Banco de Dados . . . 55
8.5 Entendendo Conjunto de Dados e Tabelas de Dados . . . 57
8.5.1 Implementações disponíveis . . . 57
8.5.2 Cuidado com Chaves Estrangeiras . . . 67
8.5.3 Implementando seus próprios Conjuntos de Dados/ Tabelas de Dados . . . 67
8.6 A API de Conexão . . . 68
8.7 API de Asserções de Banco de Dados . . . 69
8.7.1 Asseverando a contagem de linhas de uma Tabela . . . 70
8.7.2 Asseverando o Estado de uma Tabela . . . 70
8.7.3 Asseverando o Resultado de uma Query . . . 71
8.7.4 Asseverando o Estado de Múltiplas Tabelas . . . 72
8.8 Perguntas Mais Frequentes . . . 73
8.8.1 O PHPUnit vai (re)criar o esquema do banco de dados para cada teste? . . . 73
8.8.2 Sou forçado a usar PDO em minha aplicação para que a Extensão para Banco de Dados funcione? . . . 73
8.8.3 O que posso fazer quando recebo um Erro “Too much Connections“? . . . 73
8.8.4 Como lidar com NULL usando Conjuntos de Dados XML Plano / CSV? . . . 73

9 Dublês de Testes **75**
9.1 Esboços (stubs) . . . 76
9.2 Objetos Falsos (Mock Objects) . . . 81
9.3 Profecia . . . 87
9.4 Falsificando Traits e Classes Abstratas . . . 88
9.5 Esboçando e Falsificando Serviços Web . . . 89
9.6 Esboçando o Sistema de Arquivos . . . 90

10 Práticas de Teste **93**
10.1 Durante o Desenvolvimento . . . 93
10.2 Durante a Depuração . . . 94

11 Análise de Cobertura de Código **95**
11.1 Métricas de Software para Cobertura de Código . . . 95
11.2 Lista-branca de arquivos . . . 96
11.3 Ignorando Blocos de Código . . . 97
11.4 Especificando métodos cobertos . . . 98
11.5 Casos Extremos . . . 100

12 Outros Usos para Testes **101**
12.1 Documentação Ágil . . . 101

12.2 Testes Inter-Equipes . . . 102

13 Registrando **103**
13.1 Resultados de Teste (XML) . . . 103
13.2 Cobertura de Código (XML) . . . 104
13.3 Cobertura de Código (TEXT) . . . 105

14 Estendendo o PHPUnit **107**
14.1 Subclasse PHPUnit\Framework\TestCase . . . 107
14.2 Escreva asserções personalizadas . . . 107
14.3 Implementando PHPUnit\Framework\TestListener . . . 109
14.4 Implementando PHPUnit_Framework_Test . . . 110

15 Asserções **113**
15.1 Uso Estático vs. Não-Estático de Métodos de Asserção . . . 113
15.2 assertArrayHasKey() . . . 113
15.3 assertClassHasAttribute() . . . 114
15.4 assertArraySubset() . . . 115
15.5 assertClassHasStaticAttribute() . . . 116
15.6 assertContains() . . . 116
15.7 assertContainsOnly() . . . 119
15.8 assertContainsOnlyInstancesOf() . . . 119
15.9 assertCount() . . . 120
15.10 assertDirectoryExists() . . . 121
15.11 assertDirectoryIsReadable() . . . 122
15.12 assertDirectoryIsWritable() . . . 122
15.13 assertEmpty() . . . 123
15.14 assertEqualXMLStructure() . . . 124
15.15 assertEquals() . . . 126
15.16 assertFalse() . . . 131
15.17 assertFileEquals() . . . 131
15.18 assertFileExists() . . . 132
15.19 assertFileIsReadable() . . . 133
15.20 assertFileIsWritable() . . . 134
15.21 assertGreaterThan() . . . 134
15.22 assertGreaterThanOrEqual() . . . 135
15.23 assertInfinite() . . . 136
15.24 assertInstanceOf() . . . 137
15.25 assertInternalType() . . . 137
15.26 assertIsReadable() . . . 138
15.27 assertIsWritable() . . . 139
15.28 assertJsonFileEqualsJsonFile() . . . 140
15.29 assertJsonStringEqualsJsonFile() . . . 140
15.30 assertJsonStringEqualsJsonString() . . . 141
15.31 assertLessThan() . . . 142
15.32 assertLessThanOrEqual() . . . 143
15.33 assertNan() . . . 144
15.34 assertNull() . . . 144
15.35 assertObjectHasAttribute() . . . 145
15.36 assertRegExp() . . . 146
15.37 assertStringMatchesFormat() . . . 147
15.38 assertStringMatchesFormatFile() . . . 148
15.39 assertSame() . . . 148
15.40 assertStringEndsWith() . . . 150

15.41 assertStringEqualsFile() . . . 150
15.42 assertStringStartsWith() . . . 151
15.43 assertThat() . . . 152
15.44 assertTrue() . . . 153
15.45 assertXmlFileEqualsXmlFile() . . . 154
15.46 assertXmlStringEqualsXmlFile() . . . 155
15.47 assertXmlStringEqualsXmlString() . . . 156

16 Anotações **159**
16.1 @author . . . 159
16.2 @after . . . 159
16.3 @afterClass . . . 160
16.4 @backupGlobals . . . 160
16.5 @backupStaticAttributes . . . 161
16.6 @before . . . 162
16.7 @beforeClass . . . 162
16.8 @codeCoverageIgnore* . . . 163
16.9 @covers . . . 163
16.10 @coversDefaultClass . . . 164
16.11 @coversNothing . . . 165
16.12 @dataProvider . . . 165
16.13 @depends . . . 165
16.14 @expectedException . . . 165
16.15 @expectedExceptionCode . . . 165
16.16 @expectedExceptionMessage . . . 166
16.17 @expectedExceptionMessageRegExp . . . 167
16.18 @group . . . 167
16.19 @large . . . 168
16.20 @medium . . . 168
16.21 @preserveGlobalState . . . 168
16.22 @requires . . . 169
16.23 @runTestsInSeparateProcesses . . . 169
16.24 @runInSeparateProcess . . . 169
16.25 @small . . . 170
16.26 @test . . . 170
16.27 @testdox . . . 170
16.28 @ticket . . . 170
16.29 @uses . . . 170

17 O Arquivo de Configuração XML **173**
17.1 PHPUnit . . . 173
17.2 Suítes de Teste . . . 175
17.3 Grupos . . . 175
17.4 Lista-branca de Arquivos para Cobertura de Código . . . 175
17.5 Registrando . . . 176
17.6 Ouvintes de Teste . . . 177
17.7 Definindo configurações PHP INI, Constantes e Variáveis Globais . . . 177

18 Bibliografia **179**

19 Direitos autorais **181**

Edition for PHPUnit latest. Updated on out 17, 2018.

Sebastian Bergmann

Contents:

CAPÍTULO 1

Instalando PHPUnit

1.1 Requisitos

PHPUnit 6.4 requer PHP 7; Usar a última versão do PHP é altamente recomendável.

PHPUnit requer as extensões dom e json, que são normalmente habilitadas por padrão.

PHPUnit também requer as extensões pcre, reflection e spl. Elas são requeridas pelo núcleo do PHP desde 5.3.0 e normalmente não podem ser desabilitadas.

A funcionalidade de relatório de cobertura de código requer as extensões Xdebug (2.5.0 or later) e tokenizer. Geração de relatórios XML requer a extensão xmlwriter.

1.2 PHP Archive (PHAR)

A maneira mais fácil de obter o PHPUnit é baixar um PHP Archive (PHAR) que tem todas dependências requeridas (assim como algumas opcionais) do PHPUnit empacotados em um único arquivo.

A extensão phar é requerida para usar PHP Archives (PHAR).

Se a extensão Suhosin está habilitada, você precisa permitir a execução dos PHARs em seu `php.ini`:

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

Para instalar globalmente o PHAR:

```
$  wget https://phar.phpunit.de/phpunit-6.4.phar
$  chmod +x phpunit-6.4.phar
$  sudo mv phpunit-6.4.phar /usr/local/bin/phpunit
$  phpunit --version
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

Você pode também usar o arquivo PHAR baixado, diretamente:

```
$  wget https://phar.phpunit.de/phpunit-6.4.phar
$  php phpunit-6.4.phar --version
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

1.2.1 Windows

A instalção global do PHAR envolve o mesmo procedimento que a instalação manual do Composer no Windows:

1. Crie um diretório para os binários PHP; e.g., `C:\bin`
2. Acrescente ;C:bin à sua variável de ambiente `PATH` (ajuda relacionada)
3. Baixe https://phar.phpunit.de/phpunit.phar e salve o arquivo como `C:\bin\phpunit.phar`
4. Abra uma linha de comando (e.g., pressione `WindowsR` » digite cmd » `ENTER`)
5. Crie um script batch envoltório (resulta em `C:\bin\phpunit.cmd`):

```
C:\Users\username>  cd C:\bin
C:\bin>  echo @php "%~dp0phpunit.phar" %* > phpunit.cmd
C:\bin>  exit
```

6. Abra uma nova linha de comando e confirme que você pode executar PHPUnit de qualquer caminho:

```
C:\Users\username>  phpunit --version
PHPUnit x.y.z by Sebastian Bergmann and contributors.
```

Para ambientes shell Cygwin e/ou MingW32 (e.g., TortoiseGit), você pode pular o passo 5 acima, simplesmente salve o arquivo como `phpunit` (sem a extensão `.phar`), e torne-o executável via chmod 775 phpunit.

1.2.2 Verificando lançamentos do PHAR PHPUnit

Todos lançamentos oficiais do código distribuído pelo Projeto PHPUnit são assinados pelo gerenciador de lançamentos para o lançamento. Assinaturas PGP e hashes SHA1 estão disponíveis para verificação no phar.phpunit.de.

O exemplo a seguir detalha como a verificação de lançamento funciona. Começamos baixando `phpunit.phar` bem como sua assinatura PGP independente `phpunit.phar.asc`:

```
wget https://phar.phpunit.de/phpunit.phar
wget https://phar.phpunit.de/phpunit.phar.asc
```

Queremos verificar o PHP Archive do PHPUnit (`phpunit.phar`) contra sua assinatura independente (`phpunit.phar.asc`):

```
gpg phpunit.phar.asc
gpg: Signature made Sat 19 Jul 2014 01:28:02 PM CEST using RSA key ID 6372C20A
gpg: Can't check signature: public key not found
```

Nós não temos a chave pública do gerenciador de lançamento (`6372C20A`) no próprio sistema local. A fim de prosseguir com a verificação nós precisamos recuperar a chave pública do gerenciador de lançamentos a partir de um servidor de chaves. Um tal servidor é `pgp.uni-mainz.de`. Os servidores de chaves públicas são conectados entre si, então você deve ser capaz de se conectar a qualquer servidor de chaves.

```
gpg --keyserver pgp.uni-mainz.de --recv-keys 0x4AA394086372C20A
gpg: requesting key 6372C20A from hkp server pgp.uni-mainz.de
gpg: key 6372C20A: public key "Sebastian Bergmann <sb@sebastian-bergmann.de>" imported
```

(continues on next page)

(continuação da página anterior)

```
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Agora recebemos uma chave pública para uma entidade conhecida como "Sebastian Bergmann <sb@sebastian-bergmann.de>". Porém, nós não temos nenhuma maneira de verificar que essa foi criada pela pessoa conhecida como Sebastian Bergmann. Mas, vamos tentar verificar a assinatura de lançamento novamente.

```
gpg phpunit.phar.asc
gpg: Signature made Sat 19 Jul 2014 01:28:02 PM CEST using RSA key ID 6372C20A
gpg: Good signature from "Sebastian Bergmann <sb@sebastian-bergmann.de>"
gpg:                 aka "Sebastian Bergmann <sebastian@php.net>"
gpg:                 aka "Sebastian Bergmann <sebastian@thephp.cc>"
gpg:                 aka "Sebastian Bergmann <sebastian@phpunit.de>"
gpg:                 aka "Sebastian Bergmann <sebastian.bergmann@thephp.cc>"
gpg:                 aka "[jpeg image of size 40635]"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: D840 6D0D 8294 7747 2937  7831 4AA3 9408 6372 C20A
```

Neste ponto, a assinatura é boa, mas não confiamos nesta chave. Uma boa assinatura significa que o arquivo não foi adulterado. Porém, devido a natureza da criptografia de chave pública, você precisa adicionalmente verificar que a chave `6372C20A` foi criada pelo verdadeiro Sebastian Bergmann.

Qualquer invasor pode criar uma chave pública e enviá-la para os servidores de chave pública. Eles podem, então, criar um lançamento malicioso assinado pela chave fake. Tal que, se você tentar verificar a assinatura desse lançamento corrompido, terá sucesso porque a chave não é a chave "verdadeira". Portanto, você precisa validar a autenticidade dessa chave. Validar a autenticidade de uma chave pública, no entanto, está fora do escopo desta documentação.

Pode ser prudente criar um script shell para gerenciar a instalação do PHPUnit que verifica a assinatura GnuPG antes de rodar sua suíte de teste. Por exemplo:

```
#!/usr/bin/env bash
clean=1 # Delete phpunit.phar after the tests are complete?
aftercmd="php phpunit.phar --bootstrap bootstrap.php src/tests"
gpg --fingerprint D8406D0D82947747293778314AA394086372C20A
if [ $? -ne 0 ]; then
    echo -e "\033[33mDownloading PGP Public Key...\033[0m"
    gpg --recv-keys D8406D0D82947747293778314AA394086372C20A
    # Sebastian Bergmann <sb@sebastian-bergmann.de>
    gpg --fingerprint D8406D0D82947747293778314AA394086372C20A
    if [ $? -ne 0 ]; then
        echo -e "\033[31mCould not download PGP public key for verification\033[0m"
        exit
    fi
fi

if [ "$clean" -eq 1 ]; then
    # Let's clean them up, if they exist
    if [ -f phpunit.phar ]; then
        rm -f phpunit.phar
    fi
    if [ -f phpunit.phar.asc ]; then
        rm -f phpunit.phar.asc
    fi
fi

# Let's grab the latest release and its signature
```

(continues on next page)

(continuação da página anterior)

```
if [ ! -f phpunit.phar ]; then
    wget https://phar.phpunit.de/phpunit.phar
fi
if [ ! -f phpunit.phar.asc ]; then
    wget https://phar.phpunit.de/phpunit.phar.asc
fi

# Verify before running
gpg --verify phpunit.phar.asc phpunit.phar
if [ $? -eq 0 ]; then
    echo
    echo -e "\033[33mBegin Unit Testing\033[0m"
    # Run the testing suite
    `$after_cmd`
    # Cleanup
    if [ "$clean" -eq 1 ]; then
        echo -e "\033[32mCleaning Up!\033[0m"
        rm -f phpunit.phar
        rm -f phpunit.phar.asc
    fi
else
    echo
    chmod -x phpunit.phar
    mv phpunit.phar /tmp/bad-phpunit.phar
    mv phpunit.phar.asc /tmp/bad-phpunit.phar.asc
    echo -e "\033[31mSignature did not match! PHPUnit has been moved to /tmp/bad-
↪phpunit.phar\033[0m"
    exit 1
fi
```

1.3 Composer

Simplesmente adicione uma dependência (em desenvolvimento) `phpunit/phpunit` ao arquivo `composer.json` do projeto se você usa Composer para gerenciar as dependências do seu projeto:

```
composer require --dev phpunit/phpunit ^6.4
```

1.4 Pacotes opcionais

Os seguintes pacotes opcionais estão disponíveis:

`PHP_Invoker`

A classe utilitária para invocar callables com um tempo limite. Este pacote é requerido para impor limites de tempo de teste no modo estrito.

Este pacote está incluso na distribuição PHAR do PHPUnit. Ele pode ser instalado via Composer usando o seguinte comando:

```
composer require --dev phpunit/php-invoker
```

`DbUnit`

Porta do DbUnit para PHP/PHPUnit suportar teste de interação de banco de dados.

Este pacote está incluso na distribuição PHAR do PHPUnit. Ele pode ser instalado via Composer usando o seguinte comando:

```
composer require --dev phpunit/dbunit
```

CAPÍTULO 2

Escrevendo Testes para o PHPUnit

Example 2.1 mostra como podemos escrever testes usando o PHPUnit que exercita operações de vetor do PHP. O exemplo introduz as convenções básicas e passos para escrever testes com o PHPUnit:

1. Os testes para uma classe `Class` vão dentro de uma classe `ClassTest`.
2. `ClassTest` herda (na maioria das vezes) de `PHPUnit\Framework\TestCase`.
3. Os testes são métodos públicos que são nomeados `test*`.

   Alternativamente, você pode usar a anotação `@test` em um bloco de documentação de um método para marcá-lo como um método de teste.
4. Dentro dos métodos de teste, métodos de asserção tal como `assertEquals()` (veja *Asserções*) são usados para asseverar que um valor real equivale a um valor esperado.

Example 2.1: Testando operações de vetores com o PHPUnit

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

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

Martin Fowler:

> Sempre que você estiver tentado a escrever algo em uma declaração `print` ou uma expressão depuradora, escreva-a como um teste em vez disso.

2.1 Dependências de Testes

> *Adrian Kuhn et. al.*:
>
> Testes Unitários são primeiramente escritos como uma boa prática para ajudar desenvolvedores a identificar e corrigir defeitos, refatorar o código e servir como documentação para uma unidade de programa sob teste. Para conseguir esses benefícios, testes unitários idealmente deveriam cobrir todos os caminhos possíveis em um programa. Um teste unitário geralmente cobre um caminho específico em uma função ou método. Porém um método de teste não é necessariamente uma entidade encapsulada e independente. Frequentemente existem dependências implícitas entre métodos de teste, escondidas no cenário de implementação de um teste.

O PHPUnit suporta a declaração de dependências explícitas entre métodos de teste. Tais dependências não definem a ordem em que os métodos de teste devem ser executados, mas permitem o retorno de uma instância do ambiente do teste por um produtor e a passagem dele para os consumidores dependentes.

- Um produtor é um método de teste que produz a sua unidade sob teste como valor de retorno.
- Um consumidor é um método de teste que depende de um ou mais produtores e seus valores retornados.

Example 2.2 mostra como usar a anotação `@depends` para expressar dependências entre métodos de teste.

Example 2.2: Usando a anotação `@depends` para expressar dependências

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

class StackTest extends TestCase
{
    public function testEmpty()
    {
        $stack = [];
        $this->assertEmpty($stack);

        return $stack;
    }

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

        return $stack;
    }

    /**
```

(continues on next page)

(continuação da página anterior)

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

No exemplo acima, o primeiro teste, `testEmpty()`, cria um novo vetor e assegura que o mesmo é vazio. O teste então retorna o ambiente como resultado. O segundo teste, `testPush()`, depende de `testEmpty()` e lhe é passado o resultado do qual ele depende como um argumento. Finalmente, `testPop()` depende de `testPush()`.

Note

O valor de retorno produzido por um produtor é passado "como está" para seus consumidores por padrão. Isso significa que quando um produtor retorna um objeto, uma referência para esse objeto é passada para os consumidores. Quando uma cópia deve ser usada ao invés de uma referência, então @depends clone deve ser usado ao invés de @depends.

Para localizar defeitos rapidamente, queremos nossa atenção focada nas falhas relevantes dos testes. É por isso que o PHPUnit pula a execução de um teste quando um teste do qual ele depende falha. Isso melhora a localização de defeitos por explorar as dependências entre os testes como mostrado em Example 2.3.

Example 2.3: Explorando as dependências entre os testes

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

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

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

```
$ phpunit --verbose DependencyFailureTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

FS

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) DependencyFailureTest::testOne
```

(continues on next page)

(continuação da página anterior)

```
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.
```

Um teste pode ter mais de uma anotação `@depends`. O PHPUnit não muda a ordem em que os testes são executados, portanto você deve se certificar de que as dependências de um teste podem realmente ser encontradas antes de executar o teste.

Um teste que tem mais de uma anotação `@depends` vai obter um ambiente a partir do primeiro produtor como o primeiro argumento, um ambiente a partir do segundo produtor como o segundo argumento, e assim por diante. Veja Example 2.4

Example 2.4: Teste com múltiplas dependências

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

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

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

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

```
$ phpunit --verbose MultipleDependenciesTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

```

(continues on next page)

(continuação da página anterior)

```
...

Time: 0 seconds, Memory: 3.25Mb

OK (3 tests, 3 assertions)
```

2.2 Provedores de Dados

Um método de teste pode aceitar argumentos arbitrários. Esses argumentos devem ser fornecidos por um método provedor de dados (`additionProvider()` em Example 2.5). O método provedor de dados a ser usado é especificado usando a anotação `@dataProvider`.

Um método provedor de dados deve ser `public` e retornar um vetor de vetores ou um objeto que implemente a interface `Iterator` e produza um vetor para cada passo da iteração. Para cada vetor que é parte da coleção o método de teste será chamado com os conteúdos do vetor como seus argumentos.

Example 2.5: Usando um provedor de dados que retorna um vetor de vetores

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

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

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

```
$ phpunit DataTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) DataTest::testAdd with data set #3 (1, 1, 3)
```

(continues on next page)

(continuação da página anterior)

```
Failed asserting that 2 matches expected 3.

/home/sb/DataTest.php:9

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

Quando usar um número grande de conjuntos de dados é útil nomear cada um com uma chave string ao invés do padrão numérico. Output will be more verbose as it'll contain that name of a dataset that breaks a test.

Example 2.6: Usando um provedor de dados com conjuntos de dados nomeados

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

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

    public function additionProvider()
    {
        return [
            'adding zeros'  => [0, 0, 0],
            'zero plus one' => [0, 1, 1],
            'one plus zero' => [1, 0, 1],
            'one plus one'  => [1, 1, 3]
        ];
    }
}
?>
```

```
$ phpunit DataTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:9

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

Example 2.7: Usando um provedor de dados que retorna um objeto Iterador

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

require 'CsvFileIterator.php';

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

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

```
$ phpunit DataTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

/home/sb/DataTest.php:11

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

Example 2.8: A classe CsvFileIterator

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

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

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

(continues on next page)

(continuação da página anterior)

```
    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++;
    }
}
?>
```

Quando um teste recebe uma entrada tanto de um método `@dataProvider` quanto de um ou mais testes dos quais ele `@depends`, os argumentos do provedor de dados virão antes daqueles dos quais ele é dependente. Os argumentos dos quais o teste depende serão os mesmos para cada conjunto de dados. Veja Example 2.9

Example 2.9: Combinação de @depends e @dataProvider no mesmo teste

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

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

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

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

(continues on next page)

(continuação da página anterior)

```
    }

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

```
$ phpunit --verbose DependencyAndDataProviderComboTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

...F

Time: 0 seconds, Memory: 3.50Mb

There was 1 failure:

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

/home/sb/DependencyAndDataProviderComboTest.php:31

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

Note

Quando um teste depende de um teste que usa provedores de dados, o teste dependente será executado quando o teste do qual ele depende for bem sucedido em pelo menos um conjunto de dados. O resultado de um teste que usa provedores de dados não pode ser injetado dentro de um teste dependente.

Note

Todos provedores de dados são executados antes da chamada ao método estático `setUpBeforeClass` e a primeira chamada ao método `setUp`. Por isso você não pode acessar quaisquer variáveis que criar ali de dentro de um provedor

de dados. Isto é necessário para que o PHPUnit seja capaz de calcular o número total de testes.

2.3 Testando Exceções

Example 2.10 mostra como usar a anotação `expectException()` para testar se uma exceção é lançada dentro do código de teste.

Example 2.10: Usando o método expectException()

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

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

```
$ phpunit ExceptionTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

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

Além do método `expectException()` os métodos `expectExceptionCode()`, `expectExceptionMessage()`, e `expectExceptionMessageRegExp()` existem para configurar expectativas de exceções lançadas pelo código sob teste.

Alternativamente, você pode usar as anotações `@expectedException`, `@expectedExceptionCode`, `@expectedExceptionMessage`, e `@expectedExceptionMessageRegExp` para configurar expectativas de exceções lançadas pelo código sob teste. Example 2.11 mostra um exemplo.

Example 2.11: Usando a anotação @expectedException

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

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

(continues on next page)

(continuação da página anterior)

```
    public function testException()
    {
    }
}
?>
```

```
$ phpunit ExceptionTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) ExceptionTest::testException
Expected exception InvalidArgumentException

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

2.4 Testando Erros PHP

Por padrão, o PHPUnit converte os erros, avisos e notificações do PHP que são disparados durante a execução de um teste para uma exceção. Usando essas exceções, você pode, por exemplo, esperar que um teste dispare um erro PHP como mostrado no Example 2.12.

Note

A configuração em tempo de execução `error_reporting` do PHP pode limitar quais erros o PHPUnit irá converter para exceções. Se você está tendo problemas com essa funcionalidade, certifique-se que o PHP não está configurado para suprimir os tipos de erros que você esta testando.

Example 2.12: Esperando um erro PHP usando @expectedException

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

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

```
$ phpunit -d error_reporting=2 ExpectedErrorTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

.

Time: 0 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

`PHPUnit\Framework\Error\Notice` e `PHPUnit\Framework\Error\Warning` representam notificações e avisos do PHP, respectivamente.

Note

Você deve ser o mais específico possível quando testar exceções. Testar por classes que são muito genéricas pode causar efeitos colaterais indesejáveis. Da mesma forma, testar para a classe `Exception` com `@expectedException` ou `setExpectedException()` não é mais permitido.

Ao testar algo que dependa de funções php que disparam erros como `fopen` pode ser útil algumas vezes usar a supressão de erros enquanto testa. Isso permite a você verificar os valores retornados por suprimir notificações que levariam a uma `PHPUnit\Framework\Error\Notice` phpunit.

Example 2.13: Testando valores de retorno de código que utiliza PHP Errors

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

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

?>
```

```
$ phpunit ErrorSuppressionTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

.

Time: 1 seconds, Memory: 5.25Mb

OK (1 test, 1 assertion)
```

Sem a supressão de erros o teste teria relatado uma falha ‘‘fopen(/is-not-writeable/file): failed to open stream:

No such file or directory‘‘.

2.5 Testando Saídas

Às vezes você quer assegurar que a execução de um método, por exemplo, gere uma saída esperada (via `echo` ou `print`, por exemplo). A classe `PHPUnit\Framework\TestCase` usa a funcionalidade Output Buffering do PHP para fornecer a funcionalidade que é necessária para isso.

Example 2.14 mostra como usar o método `expectOutputString()` para definir a saída esperada. Se essa saída esperada não for gerada, o teste será contado como uma falha.

Example 2.14: Testando a saída de uma função ou método

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

class OutputTest extends TestCase
{
    public function testExpectFooActualFoo()
    {
        $this->expectOutputString('foo');
        print 'foo';
    }

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

```
$ phpunit OutputTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

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

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

Table 2.1 mostra os métodos fornecidos para testar saídas.

Table 2.1: Métodos para testar a saída

Método	Significado
`void expectOutputRegex(string $regularExpression)`	Configura a expectativa de que a saída combine com uma `$regularExpression`.
`void expectOutputString(string $expectedString)`	Configura a expectativa de que a saída é igual a uma `$expectedString`.
`bool setOutputCallback(callable $callback)`	Define um callback que é usado, por exemplo, para normalizar a saída real.
`string getActualOutput()`	Recupera a saída real.

Note

Um teste que emite saída irá falhar no modo estrito.

2.6 Saída de Erro

Sempre que um teste falha o PHPUnit faz o melhor para fornecer a você o máximo possível de conteúdo que possa ajudar a identificar o problema.

Example 2.15: Saída de erro gerada quando uma comparação de vetores falha

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

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

```
$ phpunit ArrayDiffTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) ArrayDiffTest::testEquality
Failed asserting that two arrays are equal.
--- Expected
+++ Actual
@@ @@
 Array (
```

(continues on next page)

(continuação da página anterior)

```
     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.
```

Neste exemplo apenas um dos valores dos vetores diferem e os outros valores são exibidos para fornecer o contexto onde o erro ocorreu.

Quando a saída gerada for longa demais para ler o PHPUnit vai quebrá-la e fornecer algumas linhas de contexto ao redor de cada diferença.

Example 2.16: Saída de erro quando uma comparação de um vetor longo falha

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

class LongArrayDiffTest extends TestCase
{
    public function testEquality() {
        $this->assertEquals(
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2,  3, 4, 5, 6],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 33, 4, 5, 6]
        );
    }
}
?>
```

```
$ phpunit LongArrayDiffTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

(continues on next page)

(continuação da página anterior)

```
     17 => 6
 )

/home/sb/LongArrayDiffTest.php:7

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

2.6.1 Casos Extremos

Quando uma comparação falha o PHPUnit cria uma representação textual da entrada de valores e as compara. Devido a essa implementação uma diferenciação pode mostrar mais problemas do que realmente existem.

Isso só acontece quando se usa assertEquals ou outra função de comparação 'fraca' em vetores ou objetos.

Example 2.17: Caso extremo na geração de diferenciação quando se usa uma comparação fraca

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

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

```
$ phpunit ArrayWeakComparisonTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

(continues on next page)

(continuação da página anterior)

```
 )

/home/sb/ArrayWeakComparisonTest.php:7

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

Neste exemplo a diferença no primeiro índice entre `1` e `'1'` é relatada ainda que o assertEquals considere os valores como uma combinação.

CAPÍTULO 3

O executor de testes em linha-de-comando

O executor de testes em linha-de-comando do PHPUnit pode ser invocado através do comando `phpunit`. O código seguinte mostra como executar testes com o executor de testes em linha-de-comando do PHPUnit:

```
$ phpunit ArrayTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

..

Time: 0 seconds

OK (2 tests, 2 assertions)
```

Quando invocado como mostrado acima, o executor de testes em linha-de-comando do PHPUnit irá procurar por um arquivo-fonte `ArrayTest.php` no diretório de trabalho atual, carregá-lo, e espera encontrar uma classe de caso de teste `ArrayTest`. Irá então executar os testes desta classe.

Para cada teste executado, a ferramenta de linha-de-comando do PHPUnit imprime um caractere para indicar o progresso:

`.`
: Impresso quando um teste é bem sucedido.

`F`
: Impresso quando uma asserção falha enquanto o método de teste executa.

`E`
: Impresso quando um erro ocorre enquanto o método de teste executa.

`R`
: Impresso quando o teste foi marcado como arriscado (veja *Testes arriscados*).

`S`
: Impresso quando o teste é pulado (veja *Testes Incompletos e Pulados*).

`I`

Impresso quando o teste é marcado como incompleto ou ainda não implementado (veja *Testes Incompletos e Pulados*).

O PHPUnit distingue entre *falhas* e *erros*. Uma falha é uma asserção do PHPUnit violada assim como uma chamada falha ao `assertEquals()`. Um erro é uma exceção inesperada ou um erro do PHP. Às vezes essa distinção se mostra útil já que erros tendem a ser mais fáceis de consertar do que falhas. Se você tiver uma grande lista de problemas, é melhor enfrentar os erros primeiro e ver se quaisquer falhas continuam depois de todos consertados.

3.1 Opções de linha-de-comando

Vamos dar uma olhada nas opções do executor de testes em linha-de-comando no código seguinte:

```
$ phpunit --help
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

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

Code Coverage Options:

  --coverage-clover <file>    Generate code coverage report in Clover XML format.
  --coverage-crap4j <file>    Generate code coverage report in Crap4J XML format.
  --coverage-html <dir>       Generate code coverage report in HTML format.
  --coverage-php <file>       Export PHP_CodeCoverage object to file.
  --coverage-text=<file>      Generate code coverage report in text format.
                              Default: Standard output.
  --coverage-xml <dir>        Generate code coverage report in PHPUnit XML format.
  --whitelist <dir>           Whitelist <dir> for code coverage analysis.
  --disable-coverage-ignore   Disable annotations for ignoring code coverage.

Logging Options:

  --log-junit <file>          Log test execution in JUnit XML format to file.
  --log-teamcity <file>       Log test execution in TeamCity format to file.
  --testdox-html <file>       Write agile documentation in HTML format to file.
  --testdox-text <file>       Write agile documentation in Text format to file.
  --testdox-xml <file>        Write agile documentation in XML format to file.
  --reverse-list              Print defects in reverse order

Test Selection Options:

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

Test Execution Options:

  --dont-report-useless-tests Do not report tests that do not test anything.
  --strict-coverage           Be strict about @covers annotation usage.
```

(continues on next page)

(continuação da página anterior)

```
  --strict-global-state       Be strict about changes to global state
  --disallow-test-output      Be strict about output during tests.
  --disallow-resource-usage   Be strict about resource usage during small tests.
  --enforce-time-limit        Enforce time limit based on test size.
  --disallow-todo-tests       Disallow @todo-annotated tests.

  --process-isolation         Run each test in a separate PHP process.
  --globals-backup            Backup and restore $GLOBALS for each test.
  --static-backup             Backup and restore static attributes for each test.

  --colors=<flag>             Use colors in output ("never", "auto" or "always").
  --columns <n>               Number of columns to use for progress output.
  --columns max               Use maximum number of columns for progress output.
  --stderr                    Write to STDERR instead of STDOUT.
  --stop-on-error             Stop execution upon first error.
  --stop-on-failure           Stop execution upon first error or failure.
  --stop-on-warning           Stop execution upon first warning.
  --stop-on-risky             Stop execution upon first risky test.
  --stop-on-skipped           Stop execution upon first skipped test.
  --stop-on-incomplete        Stop execution upon first incomplete test.
  --fail-on-warning           Treat tests with warnings as failures.
  --fail-on-risky             Treat risky tests as failures.
  -v|--verbose                Output more verbose information.
  --debug                     Display debugging information.

  --loader <loader>           TestSuiteLoader implementation to use.
  --repeat <times>            Runs the test(s) repeatedly.
  --teamcity                  Report test execution progress in TeamCity format.
  --testdox                   Report test execution progress in TestDox format.
  --testdox-group             Only include tests from the specified group(s).
  --testdox-exclude-group     Exclude tests from the specified group(s).
  --printer <printer>         TestListener implementation to use.

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).
  --no-coverage               Ignore code coverage configuration.
  --no-extensions             Do not load PHPUnit extensions.
  --include-path <path(s)>    Prepend PHP's include_path with given path(s).
  -d key[=value]              Sets a php.ini value.
  --generate-configuration    Generate configuration file with suggested settings.

Miscellaneous Options:

  -h|--help                   Prints this usage information.
  --version                   Prints the version and exits.
  --atleast-version <min>     Checks that version is greater than min and exits.
```

`phpunit UnitTest`

> Executa os testes que são fornecidos pela classe `UnitTest`. Espera-se que essa classe seja declarada no arquivo-fonte `UnitTest.php`.
>
> `UnitTest` deve ser ou uma classe que herda de `PHPUnit\Framework\TestCase` ou uma classe que fornece um método `public static suite()` que retorna um objeto `PHPUnit_Framework_Test`, por exemplo uma instância da classe

`PHPUnit_Framework_TestSuite`.

`phpunit UnitTest UnitTest.php`

Executa os testes que são fornecidos pela classe `UnitTest`. Espera-se que esta classe seja declarada no arquivo-fonte especificado.

`--coverage-clover`

Gera um arquivo de registro no formato XML com as informações da cobertura de código para a execução dos testes. Veja *Registrando* para mais detlahes.

Por favor, note que essa funcionalidade está disponível somente quando as extensões tokenizer e Xdebug estão instaladas.

`--coverage-crap4j`

Gera um relatório de cobertura de código no formato Crap4j. Veja *Análise de Cobertura de Código* para mais detalhes.

Por favor, note que essa funcionalidade está disponível somente quando as extensões tokenizer e Xdebug estão instaladas.

`--coverage-html`

Gera um relatório no formato HTML. Veja *Análise de Cobertura de Código* para mais detalhes.

Por favor, note que essa funcionalidade está disponível somente quando as extensões tokenizer e Xdebug estão instaladas.

`--coverage-php`

Gera um objeto PHP_CodeCoverage serializado com as informações de cobertura de código.

Por favor, note que essa funcionalidade está disponível somente quando as extensões tokenizer e Xdebug estão instaladas.

`--coverage-text`

Gera um arquivo de registro ou saída de linha de comando no formato legível por humanos com a informação de cobertura de código para a execução dos testes. Veja *Registrando* para mais detalhes.

Por favor, note que essa funcionalidade está disponível somente quando as extensões tokenizer e Xdebug estão instaladas.

`--log-junit`

Gera um arquivo de registro no formato XML Junit para a execução dos testes. Veja *Registrando* para mais detalhes.

`--testdox-html` e `--testdox-text`

Gera documentação ágil no formato HTML ou texto plano para os testes que são executados. Veja *Outros Usos para Testes* para mais detalhes.

`--filter`

Apenas executa os testes cujos nomes combinam com o padrão fornecido. Se o padrão não for colocado entre delimitadores, o PHPUnit irá colocar o padrão no delimitador `/`.

Os nomes de teste para combinar estará em um dos seguintes formatos:

`TestNamespace\TestCaseClass::testMethod`

O formato do nome de teste padrão é o equivalente ao usar a constante mágica `__METHOD__` dentro do método de teste.

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

Quando um teste tem um provedor de dados, cada iteração dos dados obtém o índice atual acrescido ao final do nome do teste padrão.

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

Quando um teste tem um provedor de dados que usa conjuntos nomeados, cada iteração dos dados obtém o nome atual acrescido ao final do nome do teste padrão. Veja Example 3.1 para um exemplo de conjunto de dados nomeados.

Example 3.1: Conjunto de dados nomeados

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

namespace TestNamespace;

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

    public function provider()
    {
        return [
            'my named data' => [true],
            'my data'       => [true]
        ];
    }
}
?>
```

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

O nome do teste para um teste PHPT é o caminho do sistema de arquivos.

Veja Example 3.2 para exemplos de padrões de filtros válidos.

Example 3.2: Exmplos de padrão de filtro

Veja Example 3.3 para alguns atalhos adicionais que estão disponíveis para combinar provedores de dados

Example 3.3: Atalhos de filtro

`--testsuite`

Só roda o conjunto de teste cujo nome combina com o padrão dado.

`--group`

Apenas executa os testes do(s) grupo(s) especificado(s). Um teste pode ser marcado como pertencente a um grupo usando a anotação `@group`.

A anotação `@author` é um apelido para `@group`, permitindo filtrar os testes com base em seus autores.

`--exclude-group`

Exclui testes do(s) grupo(s) especificado(s). Um teste pode ser marcado como pertencente a um grupo usando a anotação `@group`.

`--list-groups`

Lista os grupos de teste disponíveis.

`--test-suffix`

Só procura por arquivos de teste com o sufixo(s) especificado..

`--report-useless-tests`

Seja estrito sobre testes que não testam nada. Veja *Testes arriscados* para detalhes.

`--strict-coverage`

Seja estrito sobre a cobertura de código involuntariamente. Veja *Testes arriscados* para detalhes.

`--strict-global-state`

Seja estrito sobre manipulação de estado global. Veja *Testes arriscados* para detalhes.

`--disallow-test-output`

Seja estrito sobre a saída durante os testes. Veja *Testes arriscados* para detalhes.

`--disallow-todo-tests`

Não executa testes que tem a anotação `@todo` em seu bloco de documentação.

`--enforce-time-limit`

Impõem limite de tempo baseado no tamanho do teste. Veja *Testes arriscados* para detalhes.

`--process-isolation`

Executa cada teste em um processo PHP separado.

`--no-globals-backup`

Não faz cópia de segurança e restauração de $GLOBALS. Veja *Estado Global* para mais detalhes.

`--static-backup`

Faz cópia de segurança e restauração atributos estáticos das classes definidas pelo usuário. Veja *Estado Global* para mais detalhes.

`--colors`

Usa cores na saída. No Windows, usar ANSICON ou ConEmu.

Existem três valores possíveis para esta opção:

- `never`: nunca exibe cores na saída. Esse é o valor padrão quando a opção `--colors` não é usada.
- `auto`: exibe cores na saída a menos que o terminal atual não suporte cores, ou se a saída é canalizada (piped) para um comando ou redirecionada para um arquivo.
- `always`: sempre exibe cores na saída mesmo quando o terminal atual não suporta cores, ou quando a saída é canalizada para um comando ou redirecionada para um arquivo.

Quando `--colors` é usada com nenhum valor, `auto` é o valor escolhido.

`--columns`

Define o número de colunas a serem usadas para a saída de progresso. Se `max` é definido como valor, o número de colunas será o máximo do terminal atual.

`--stderr`

Opcionalmente imprime para `STDERR` em vez de `STDOUT`.

`--stop-on-error`

Para a execução no primeiro erro.

`--stop-on-failure`

Para a execução no primeiro erro ou falha.

`--stop-on-risky`

Para a execução no primeiro teste arriscado.

`--stop-on-skipped`

Para a execução no primeiro teste pulado.

`--stop-on-incomplete`

Para a execução no primeiro teste incompleto.

`--verbose`

Saída mais verbosa de informações, por exemplo, os nomes dos testes que ficaram incompletos ou foram pulados.

`--debug`

Informação de depuração na saída, tal como o nome de um teste quando a execução começa.

`--loader`

Especifica a implementação `PHPUnit_Runner_TestSuiteLoader` a ser usada.

O carregador de suíte de teste padrão irá procurar pelo arquivo-fonte no diretório de trabalho atual e em cada diretório que está especificado na diretiva de configuração `include_path` do PHP. Um nome de classe como `Project_Package_Class` é mapeado para o nome de arquivo-fonte `Project/Package/Class.php`.

`--repeat`

Executa repetidamente o(s) teste(s) um determinado número de vezes.

`--testdox`

Relata o progresso do teste como uma documentação ágil. Veja *Outros Usos para Testes* para mais detalhes.

`--printer`

Especifica a impressora de resultados a ser usada. A classe impressora deve estender `PHPUnit_Util_Printer` e implementar a interface `PHPUnit\Framework\TestListener`.

`--bootstrap`

Um arquivo PHP "bootstrap" que é executado antes dos testes.

`--configuration`, `-c`

Lê a configuração de um arquivo XML. Veja *O Arquivo de Configuração XML* para mais detalhes.

Se `phpunit.xml` ou `phpunit.xml.dist` (nessa ordem) existir no diretório de trabalho atual e `--configuration` *não* for usado, a configuração será lida automaticamente desse arquivo.

`--no-configuration`

Ignora `phpunit.xml` e `phpunit.xml.dist` do diretório de trabalho atual.

`--include-path`

Precede o `include_path` do PHP com o(s) caminho(s) fornecido(s).

`-d`

Define o valor da opção de configuração do PHP fornecida.

Note

Por favor, note que a partir da 4.8, opções podem ser colocadas após o argumento(s).

CAPÍTULO 4

Ambientes

Uma das partes que mais consomem tempo ao se escrever testes é escrever o código para ajustar o ambiente para um estado conhecido e então retorná-lo ao seu estado original quando o teste está completo. Esse estado conhecido é chamado de *ambiente* do teste.

Em *Testando operações de vetores com o PHPUnit*, o ambiente era simplesmente o vetor que está guardado na variável `$stack`. Na maior parte do tempo, porém, o ambiente será mais complexo do que um simples vetor, e a quantidade de código necessária para defini-lo aumentará na mesma proporção. O conteúdo real do teste se perde na bagunça da configuração do ambiente. Esse problema piora ainda mais quando você escreve vários testes com ambientes similares. Sem alguma ajuda do framework de teste, teríamos que duplicar o código que define o ambiente para cada teste que escrevermos.

O PHPUnit suporta compartilhamento do código de configuração. Antes que um método seja executado, um método modelo chamado `setUp()` é invocado. `setUp()` é onde você cria os objetos que serão alvo dos testes. Uma vez que o método de teste tenha terminado sua execução, seja bem-sucedido ou falho, outro método modelo chamado `tearDown()` é invocado. `tearDown()` é onde você limpa os objetos que foram alvo dos testes.

Em *Usando a anotação @depends para expressar dependências* usamos o relacionamento produtor-consumidor entre testes para compartilhar ambientes. Isso nem sempre é desejável, ou mesmo possível. Example 4.1 mostra como podemos escrever os testes do `StackTest` de forma que o próprio ambiente não é reutilizado, mas o código que o cria. Primeiro declaramos a variável de instância `$stack`, que usaremos no lugar de uma variável do método local. Então colocamos a criação do ambiente `vetor` dentro do método `setUp()`. Finalmente, removemos o código redundante dos métodos de teste e usamos a nova variável de instância, `$this->stack`, em vez da variável do método local `$stack` com o método de asserção `assertEquals()`.

Example 4.1: Usando setUp() para criar o ambiente stack

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

class StackTest extends TestCase
{
    protected $stack;

    protected function setUp()
```

(continues on next page)

(continuação da página anterior)

```
    {
        $this->stack = [];
    }

    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));
    }
}
?>
```

Os métodos-modelo `setUp()` e `tearDown()` são executados uma vez para cada método de teste (e em novas instâncias) da classe do caso de teste.

Além disso, os métodos-modelo `setUpBeforeClass()` e `tearDownAfterClass()` são chamados antes do primeiro teste da classe do caso de teste ser executado e após o último teste da classe do caso de teste ser executado, respectivamente.

O exemplo abaixo mostra todos os métodos-modelo que estão disponíveis em uma classe de caso de teste.

Example 4.2: Exemplo mostrando todos os métodos-modelo disponíveis

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

class TemplateMethodsTest extends 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()
```

(continues on next page)

(continuação da página anterior)

```
    {
        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 TemplateMethodsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

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

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) TemplateMethodsTest::testTwo
Failed asserting that <boolean:false> is true.
```

(continues on next page)

(continuação da página anterior)

```
/home/sb/TemplateMethodsTest.php:30

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

4.1 Mais setUp() que tearDown()

`setUp()` e `tearDown()` são bastante simétricos em teoria, mas não na prática. Na prática, você só precisa implementar `tearDown()` se você tiver alocado recursos externos como arquivos ou sockets no `setUp()`. Se seu `setUp()` apenas cria objetos planos do PHP, você pode geralmente ignorar o `tearDown()`. Porém, se você criar muitos objetos em seu `setUp()`, você pode querer `unset()` as variáveis que apontam para aqueles objetos em seu `tearDown()` para que eles possam ser coletados como lixo. A coleta de lixo dos objetos dos casos de teste não é previsível.

4.2 Variações

O que acontece quando você tem dois testes com definições (setups) ligeiramente diferentes? Existem duas possibilidades:

- Se o código `setUp()` diferir só um pouco, mova o código que difere do código do `setUp()` para o método de teste.
- Se você tiver um `setUp()` realmente diferente, você precisará de uma classe de caso de teste diferente. Nomeie a classe após a diferença na configuração.

4.3 Compartilhando Ambientes

Existem algumas boas razões para compartilhar ambientes entre testes, mas na maioria dos casos a necessidade de compartilhar um ambiente entre testes deriva de um problema de design não resolvido.

Um bom exemplo de um ambiente que faz sentido compartilhar através de vários testes é a conexão ao banco de dados: você loga no banco de dados uma vez e reutiliza essa conexão em vez de criar uma nova conexão para cada teste. Isso faz seus testes serem executados mais rápido.

Example 4.3 usa os métodos-modelo `setUpBeforeClass()` e `tearDownAfterClass()` para conectar ao banco de dados antes do primeiro teste da classe de casos de teste e para desconectar do banco de dados após o último teste dos casos de teste, respectivamente.

Example 4.3: Compartilhando ambientes entre os testes de uma suíte de testes

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

class DatabaseTest extends TestCase
{
    protected static $dbh;

    public static function setUpBeforeClass()
    {
```

(continues on next page)

(continuação da página anterior)

```
        self::$dbh = new PDO('sqlite::memory:');
    }

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

Não dá para enfatizar o suficiente que o compartilhamento de ambientes entre testes reduz o custo dos testes. O problema de design subjacente é que objetos não são de baixo acoplamento. Você vai conseguir melhores resultados resolvendo o problema de design subjacente e então escrevendo testes usando pontas (veja *Dublês de Testes*), do que criando dependências entre os testes em tempo de execução e ignorando a oportunidade de melhorar seu design.

4.4 Estado Global

É difícil testar um código que usa singletons (instâncias únicas de objetos). Isso também vale para os códigos que usam variáveis globais. Tipicamente, o código que você quer testar é fortemente acoplado com uma variável global e você não pode controlar sua criação. Um problema adicional é o fato de que uma alteração em uma variável global para um teste pode quebrar um outro teste.

Em PHP, variáveis globais trabalham desta forma:

- Uma variável global `$foo = 'bar';` é guardada como `$GLOBALS['foo'] = 'bar';`.
- A variável `$GLOBALS` é chamada de variável *super-global*.
- Variáveis super-globais são variáveis embutidas que estão sempre disponíveis em todos os escopos.
- No escopo de uma função ou método, você pode acessar a variável global `$foo` tanto por acesso direto à `$GLOBALS['foo']` ou usando `global $foo;` para criar uma variável local com uma referência à variável global.

Além das variáveis globais, atributos estáticos de classes também são parte do estado global.

Antes da versão 6, por padrão, o PHPUnit executa seus testes de forma que mudanças às variáveis globais ou super-globais (`$GLOBALS`, `$_ENV`, `$_POST`, `$_GET`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_REQUEST`) não afetem outros testes.

A partir da versão 6, o PHPUnit não executa essa operação de backup e restauração para variáveis globais e super-globais por padrão. Isso pode ser ativado usando a opção `--globals-backup` ou definindo `backupGlobals="true"` no arquivo de configuração XML.

Ao usar a opção `--static-backup` ou ao definir `backupStaticAttributes="true"` no no arquivo de configuração XML, esse isolamento pode ser estendido para atributos estáticos de classes.

Note

As operações de backup e restauração para variáveis globais e atributos estáticos de classes usa `serialize()` e `unserialize()`.

Objetos de algumas classes (e.g., `PDO`) não podem ser serializados e a operação de backup vai quebrar quando esse tipo de objeto for guardado no vetor `$GLOBALS`, por exemplo.

A anotação `@backupGlobals` que é discutida na *@backupGlobals* pode ser usada para controlar as operações de backup e restauração para variáveis globais. Alternativamente, você pode fornecer uma lista-negra de variáveis globais que deverão ser excluídas das operações de backup e restauração como esta:

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

    // ...
}
```

Note

Definir a propriedade `$backupGlobalsBlacklist` dentro do método `setUp()`, por exemplo, não tem efeito.

A anotação `@backupStaticAttributes` que é discutida na *@backupStaticAttributes* pode ser usada para fazer backup de todos os valores de propriedades estáticas em todas as classes declaradas antes de cada teste e restaurá-los depois.

Ele processa todas as classes que são declaradas no momento que um teste começa, não só a classe de teste. Ele só se aplica a propriedades estáticas de classe, e não variáveis estáticas dentro de funções.

Note

A operação `@backupStaticAttributes` é executada antes de um método de teste, mas somente se ele está habilitado. Se um valor estático foi alterado por um teste executado anteriormente que não tinha ativado `@backupStaticAttributes`, então esse valor será copiado e restaurado - não o valor padrão originalmente declarado. O PHP não registra o valor padrão originalmente declarado de nenhuma variável estática.

O mesmo se aplica a propriedades estáticas de classes que foram recém-carregadas/declaradas dentro de um teste. Elas não podem ser redefinidas para o seu valor padrão originalmente declarado após o teste, uma vez que esse valor é desconhecido. Qualquer que seja o valor definido irá vazar para testes subsequentes.

Para teste unitários, recomenda-se redefinir explicitamente os valores das propriedades estáticas sob teste em seu código `setUp()` ao invés (e, idealmente, também `tearDown()`, de modo a não afetar os testes posteriormente executados).

Você pode fornecer uma lista-negra de atributos estáticos que serão excluídos das operações de backup e restauração:

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

    // ...
}
```

Note

Definir a propriedade `$backupStaticAttributesBlacklist` dentro do método `setUp()` , por exemplo, não tem efeito.

CAPÍTULO 5

Organizando Testes

Um dos objetivos do PHPUnit é que os testes devem ser combináveis: queremos ser capazes de executar qualquer quantidade ou combinação de testes juntos, por exemplo todos os testes para um projeto inteiro, ou os testes para todas as classes de um ambiente que é parte do projeto, ou apenas os testes para uma única classe.

O PHPUnit suporta diferentes formas de organizar testes e combiná-los em uma suíte de testes. Este capítulo mostra as abordagens mais comuns.

5.1 Compondo uma Suíte de Testes usando o Sistema de Arquivos

Provavelmente o jeito mais fácil de compor uma suíte de testes é manter todos os arquivos-fonte dos casos de teste em um diretório de testes. O PHPUnit pode descobrir automaticamente e executar os testes atravessando recursivamente o diretório de testes.

Vamos dar uma olhada na suíte de testes da biblioteca sebastianbergmann/money. Observando a estrutura de diretórios desse projeto, podemos ver que as classes dos casos de teste no diretório `tests` espelha o pacote e estrutura de classes do Sistema Sob Teste (SST – ou SUT: System Under Teste) no diretório `src`:

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

Para executar todos os testes para a biblioteca precisamos apenas apontar o executor de testes em linha-de-comando do PHPUnit para o diretório de teste:

```
$ phpunit --bootstrap src/autoload.php tests
PHPUnit 7.0.0 by Sebastian Bergmann.

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

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

(continues on next page)

(continuação da página anterior)

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

Note

Se você apontar o executor de testes em linha-de-comando do PHPUnit para um diretório, ele irá procurar por arquivos `*Test.php`.

Para executar apenas os testes declarados na classe de casos de teste `CurrencyTest` em `tests/CurrencyTest.php`, podemos usar o seguinte comando:

```
$ phpunit --bootstrap src/autoload.php tests/CurrencyTest
PHPUnit 7.0.0 by Sebastian Bergmann.

........

Time: 280 ms, Memory: 2.75Mb

OK (8 tests, 8 assertions)
```

Para um controle mais refinado sobre quais testes executar, podemos usar a opção `--filter`:

```
$ phpunit --bootstrap src/autoload.php --filter␣
↪testObjectCanBeConstructedForValidConstructorArgument tests
PHPUnit 7.0.0 by Sebastian Bergmann.

..

Time: 167 ms, Memory: 3.00Mb

OK (2 test, 2 assertions)
```

Note

Uma desvantagem dessa abordagem é que não temos controle sobre a ordem em que os testes são executados. Isso pode causar problemas com relação às dependências dos testes, veja *Dependências de Testes*. Na próxima seção você vai ver como pode tornar explícita a ordem de execução de testes usando o arquivo de configuração XML.

5.2 Compondo uma Suíte de Testes Usando Configuração XML

O arquivo de configuração XML do PHPUnit (*O Arquivo de Configuração XML*) também pode ser usado para compor uma suíte de testes. Example 5.1 mostra um arquivo mínimo `phpunit.xml` que adicionará todas as classes `*Test` que forem encontradas em arquivos `*Test.php` quando o diretório `tests` é atravessado recursivamente.

Example 5.1: Compondo uma Suíte de Testes Usando Configuração XML

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

(continues on next page)

(continuação da página anterior)

```
      <directory>tests</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

Se o arquivo `phpunit.xml` ou `phpunit.xml.dist` (nessa ordem) existir no diretório de trabalho atual e `--configuration` *não* é usada, a configuração será automaticamente lida desse arquivo.

A ordem em que os testes são executados pode ser explicitada:

Example 5.2: Compondo uma Suíte de Testes Usando Configuração XML

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

CAPÍTULO 6

Testes arriscados

PHPUnit pode realizar verificações adicionais documentadas abaixo enquanto executa os testes.

6.1 Testes Inúteis

PHPUnit pode ser estrito sobre testes que não testam nada. Esta verificação pode ser habilitada usando a opção `--report-useless-tests` na linha de comando ou pela definição `beStrictAboutTestsThatDoNotTestAnything="true"` no arquivo de configuração XML do PHPUnit.

Um teste que não realiza uma afirmação irá ser marcado como arriscado quando essa verificação está habilitada. Expectativas sobre objetos falsificados ou anotações tais como @expectedException contam como uma asserção.

6.2 Cobertura de Código Involuntária

PHPUnit pode ser estrito sobre cobertura de código involuntária. Esta verificação pode ser habilitada usando a opção `--strict-coverage` na linha de comando ou pela definição `beStrictAboutCoversAnnotation="true"` no arquivo de configuração XML do PHPUnit.

Um teste que é anotado com @covers e executa código que não está na lista utilizando uma anotação @covers ou @uses será marcado como arriscado quando essa verificação é habilitada.

6.3 Saída Durante a Execução de Teste

PHPUnit pode ser estrito sobre a saída durante os testes. Esta verificação pode ser habilitada usando a opção `--disallow-test-output` na linha de comando ou pela definição `beStrictAboutOutputDuringTests="true"` no arquivo de configuração XML do PHPUnit.

Um teste que emite saída, por exemplo pela invocação de print tanto no código de teste ou no código testado, será marcado como arriscado quando esta verificação está habilitada.

6.4 Tempo de Espera de Execução de Teste

Um limite de tempo pode ser forçado para a execução de um teste se o pacote `PHP_Invoker` está instalado e a extensão `pcntl` está disponível. A imposição deste tempo de espera pode ser habilitado pelo uso da opção `--enforce-time-limit` na linha de comando ou pela definição `beStrictAboutTestSize="true"` no arquivo de configuração XML do PHPUnit.

Um teste anotado com `@large` irá falhar se ele demorar mais que 60 segundos para executar. Esse tempo de espera é configurável através do atributo `timeoutForLargeTests` no arquivo de configuração XML.

Um teste anotado com `@medium` irá falhar se ele demorar mais que 10 segundos para executar. Esse tempo de espera é configurável através do atributo `timeoutForMediumTests` no arquivo de configuração XML.

Um teste que não é anotado com `@medium` ou `@large` será tratado como se fosse anotado com `@small`. Um teste pequeno irá falhar se demorar mais que 1 segundo para executar. Esse tempo de espera é configurável através do atributo `timeoutForSmallTests` no arquivo de configuração XML.

6.5 Manipulação do Estado Global

PHPUnit pode ser estrito sobre testes que manipulam o estado global. Esta verificação pode ser habilitada usando a opção `--strict-global-state` na linha de comando ou pela definição `beStrictAboutOutputDuringTests="true"` no arquivo de configuração XML do PHPUnit.

CAPÍTULO 7

Testes Incompletos e Pulados

7.1 Testes Incompletos

Quando você está trabalhando em uma nova classe de caso de teste, você pode querer começar a escrever métodos de teste vazios, como:

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

para manter o controle sobre os testes que você já escreveu. O problema com os métodos de teste vazios é que eles são interpretados como bem-sucedidos pelo framework do PHPUnit. Esse erro de interpretação leva à inutilização dos relatórios de testes – você não pode ver se um teste foi realmente bem-sucedido ou simplesmente ainda não foi implementado. Chamar `$this->fail()` no método de teste não implementado não ajuda em nada, já que o teste será interpretado como uma falha. Isso seria tão errado quanto interpretar um teste não implementado como bem-sucedido.

Se imaginarmos que um teste bem-sucedido é uma luz verde e um teste mal-sucedido (falho) é uma luz vermelha, precisaremos de uma luz amarela adicional para marcar um teste como incompleto ou ainda não implementado. O `PHPUnit_Framework_IncompleteTest` é uma interface marcadora para marcar uma exceção que surge de um método de teste como resultado do teste ser incompleto ou atualmente não implementado. O `PHPUnit_Framework_IncompleteTestError` é a implementação padrão dessa interface.

O Example 7.1 mostra uma classe de caso de teste, `SampleTest`, que contém um método de teste, `testSomething()`. Chamando o conveniente método `markTestIncomplete()` (que automaticamente lançará uma exceção `PHPUnit_Framework_IncompleteTestError`) no método de teste, marcamos o teste como sendo incompleto.

Example 7.1: Marcando um teste como incompleto

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

class SampleTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testSomething()
    {
        // Opcional: Teste alguma coisa aqui, se quiser.
        $this->assertTrue(true, 'This should already work.');

        // Pare aqui e marque este teste como incompleto.
        $this->markTestIncomplete(
          'This test has not been implemented yet.'
        );
    }
}
?>
```

Um teste incompleto é denotado por um `I` na saída do executor de testes em linha-de-comando do PHPUnit, como mostrado no exemplo abaixo:

```
$ phpunit --verbose SampleTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

I

Time: 0 seconds, Memory: 3.95Mb

There was 1 incomplete test:

1) SampleTest::testSomething
This test has not been implemented yet.

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

A Table 7.1 mostra a API para marcar testes como incompletos.

Table 7.1: API para Testes Incompletos

Método	Significado
`void markTestIncomplete()`	Marca o teste atual como incompleto.
`void markTestIncomplete(string $message)`	Marca o teste atual como incompleto usando `$message` como uma mensagem explanatória.

7.2 Pulando Testes

Nem todos os testes podem ser executados em qualquer ambiente. Considere, por exemplo, uma camada de abstração de banco de dados contendo vários drivers para os diversos sistemas de banco de dados que suporta. Os testes para o driver MySQL podem ser executados apenas, é claro, se um servidor MySQL estiver disponível.

O Example 7.2 mostra uma classe de caso de teste, `DatabaseTest`, que contém um método de teste, `testConnection()`. No método-modelo `setUp()` da classe de caso de teste, verificamos se a extensão MySQLi está disponível e usamos o método `markTestSkipped()` para pular o teste caso contrário.

Example 7.2: Pulando um teste

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

class DatabaseTest extends TestCase
{
    protected function setUp()
    {
        if (!extension_loaded('mysqli')) {
            $this->markTestSkipped(
              'The MySQLi extension is not available.'
            );
        }
    }

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

Um teste que tenha sido pulado é denotado por um `S` na saída do executor de testes em linha-de-comando do PHPUnit, como mostrado no seguinte exemplo:

```
$ phpunit --verbose DatabaseTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

S

Time: 0 seconds, Memory: 3.95Mb

There was 1 skipped test:

1) DatabaseTest::testConnection
The MySQLi extension is not available.

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

Table 7.2 mostra a API para pular testes.

Table 7.2: API para Pular Testes

Método	Significado
`void markTestSkipped()`	Marca o teste atual como pulado.
`void markTestSkipped(string $message)`	Marca o teste atual como pulado usando `$message` como uma mensagem explanatória.

7.3 Pulando Testes usando @requires

Além do método acima também é possível usar a anotação `@requires` para expressar pré-condições comuns para um caso de teste.

Table 7.3: Possíveis usos para @requires

Tipo	Valores Possíveis	Exemplos	Outro exemplo
PHP	Qualquer identificador de versão do PHP	@requires PHP 5.3.3	@requires PHP 7.1-dev
PHPUnit	Qualquer identificador de versão do PHPUnit	@requires PHPUnit 3.6.3	@requires PHPUnit 4.6
OS	Uma expressão regular que combine PHP_OS	@requires OS Linux	@requires OS WIN32|WINNT
função	Qualquer parâmetro válido para function_exists	@requires function imap_open	@requires function ReflectionMethod::setAccessible
extensão	Qualquer nome de extensão junto com um opcional identificador de versão	@requires extension mysqli	@requires extension redis 2.2.0

Example 7.3: Pulando casos de teste usando @requires

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

/**
 * @requires extension mysqli
 */
class DatabaseTest extends TestCase
{
    /**
     * @requires PHP 5.3
     */
    public function testConnection()
    {
        // O Teste requer as extensões mysqli e PHP >= 5.3
    }

    // ... Todos os outros testes requerem a extensão mysqli
}
?>
```

Se você está usando uma sintaxe que não compila com uma certa versão do PHP, procure dentro da configuração xml por inclusões dependentes de versão na *Suítes de Teste*.

CAPÍTULO 8

Testando Bancos de Dados

Muitos exemplos de testes unitários iniciantes e intermediários em qualquer linguagem de programação sugerem que é perfeitamente fácil testar a lógica de sua aplicação com testes simples. Para aplicações centradas em bancos de dados isso está longe da realidade. Comece a usar WordPress, TYPO3 ou Symfony com Doctrine ou Propel, por exemplo, e você vai experimentar facilmente problemas consideráveis com o PHPUnit: apenas porque o banco de dados é fortemente acoplado com essas bibliotecas.

Note

Tenha certeza que você possui a extensão PHP `pdo` e as extensões de banco de dados específicas tal como `pdo_mysql` instalada. Caso contrário os exemplos abaixo não irão funcionar.

Você provavelmente conhece esse cenário dos seus trabalhos e projetos diários, onde você quer colocar em prática suas habilidades (novas ou não) com PHPUnit e acaba ficando preso por um dos seguintes problemas:

1. O método que você quer testar executa uma operação JOIN muito grande e usa os dados para calcular alguns resultados importantes.
2. Sua lógica de negócios faz uma mistura de declarações SELECT, INSERT, UPDATE e DELETE.
3. Você precisa definir os dados de teste em (muito provavelmente) mais de duas tabelas para conseguir dados iniciais razoáveis para os métodos que deseja testar.

A extensão DbUnit simplifica consideravelmente a configuração de um banco de dados para fins de teste e permite a você verificar os conteúdos de um banco de dados após fazer uma série de operações.

8.1 Fornecedores Suportados para Testes de Banco de Dados

DbUnit atualmente suporta MySQL, PostgreSQL, Oracle e SQLite. Através das integrações Zend Framework ou Doctrine 2 ele tem acesso a outros sistemas como IBM DB2 ou Microsoft SQL Server.

8.2 Dificuldades em Testes de Bancos de Dados

Existe uma boa razão pela qual todos os exemplos de testes unitários não incluírem interações com bancos de dados: esses tipos de testes são complexos tanto em configuração quanto em manutenção. Enquanto testar contra seu banco de dados você precisará ter cuidado com as seguintes variáveis:

- O Esquema e tabelas do banco de dados
- Inserção das linhas exigidas para o teste nessas tabelas
- Verificação do estado do banco de dados depois de executar os testes
- Limpeza do banco de dados para cada novo teste

Por causa de muitas APIs de bancos de dados como PDO, MySQLi ou OCI8 serem incômodos de usar e verbosas para escrever, fazer esses passos manualmente é um completo pesadelo.

O código de teste deve ser o mais curto e preciso possível por várias razões:

- Você não quer modificar uma considerável quantidade de código de teste por pequenas mudanças em seu código de produção.
- Você quer ser capaz de ler e entender o código de teste facilmente, mesmo meses depois de tê-lo escrito.

Adicionalmente você tem que perceber que o banco de dados é essencialmente uma variável global de entrada para seu código. Dois testes em sua suíte de testes podem executar contra o mesmo banco de dados, possivelmente reutilizando dados múltiplas vezes. Falhas em um teste podem facilmente afetar o resultado dos testes seguintes, fazendo sua experiência com os testes muito difícil. O passo de limpeza mencionado anteriormente é de maior importância para resolver o problema do "banco de dados ser uma entrada global".

DbUnit ajuda a simplificar todos esses problemas com testes de bancos de dados de uma forma elegante.

O PHPUnit só não pode ajudá-lo no fato de que testes de banco de dados são muito lentos comparados aos testes que não usam bancos de dados. Dependendo do tamanho das interações com seu banco de dados, seus testes podem levar um tempo considerável para executar. Porém se você mantiver pequena a quantidade de dados usados para cada teste e tentar testar o máximo possível sem usar testes com bancos de dados, você facilmente conseguirá tempos abaixo de um minuto, mesmo para grandes suítes de teste.

A suíte de testes do projeto Doctrine 2 por exemplo, atualmente tem uma suíte com cerca de 1000 testes onde aproximadamente a metade deles tem acesso ao banco de dados e ainda executa em 15 segundos contra um banco de dados MySQL em um computador desktop comum.

8.3 Os quatro estágios dos testes com banco de dados

Em seu livro sobre Padrões de Teste xUnit, Gerard Meszaros lista os quatro estágios de um teste unitário:

1. Configurar o ambiente (fixture)
2. Exercitar o Sistema Sob Teste
3. Verificar o resultado
4. Desmontagem (Teardown)

    O que é um ambiente (fixture)?

    Um ambiente (fixture) descreve o estado inicial em que sua aplicação e seu banco de dados estão ao executar um teste.

Testar o banco de dados exige que você utilize pelo menos a configuração (setup) e desmontagem (teardown) para limpar e escrever em suas tabelas os dados de ambiente exigidos. Porém a extensão do banco de dados tem uma boa razão para reverter esses quatro estágios em um teste de banco de dados para assemelhar o seguinte fluxo de trabalho que é executado para cada teste:

8.3.1 1. Limpar o Banco de Dados

Já que sempre existe um primeiro teste que é executado contra o banco de dados, você não sabe exatamente se já existem dados nas tabelas. O PHPUnit vai executar um TRUNCATE contra todas as tabelas que você especificou para redefinir seus estados para vazio.

8.3.2 2. Configurar o ambiente

O PHPUnit então vai iterar sobre todas as linhas do ambiente especificado e inseri-las em suas respectivas tabelas.

8.3.3 3–5. Executar Teste, Verificar resultado e Desmontar (Teardown)

Depois de redefinir o banco de dados e carregá-lo com seu estado inicial, o verdadeiro teste é executado pelo PHPUnit. Esta parte do código de teste não exige conhecimento sobre a Extensão do Banco de Dados, então você pode prosseguir e testar o que quiser com seu código.

Em seu teste use uma asserção especial chamada `assertDataSetsEqual()` para fins de verificação, porém isso é totalmente opcional. Esta função será explicada na seção "Asserções em Bancos de Dados".

8.4 Configuração de um Caso de Teste de Banco de Dados do PHPUnit

Ao usar o PHPUnit seus casos de teste vão estender a classe `PHPUnit\Framework\TestCase` da seguinte forma:

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

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

Se você quer um código de teste que trabalha com a Extensão para Banco de Dados a configuração é um pouco mais complexa e você terá que estender um TestCase abstrato diferente, exigindo que você implemente dois métodos abstratos `getConnection()` e `getDataSet()`:

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

class MyGuestbookTest extends TestCase
{
```

(continues on next page)

(continuação da página anterior)

```
    use TestCaseTrait;

    /**
     * @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');
    }
}
?>
```

8.4.1 Implementando getConnection()

Para permitir que as funcionalidades limpeza e carregamento de ambiente funcionem, a extensão de banco de dados PHPUnit requer acesso a uma conexão de banco de dados abstraída entre fornecedores através da biblioteca PDO. É importante notar que sua aplicação não precisa ser baseada em PDO para usar a Extensão para Banco de Dados do PHPUnit, pois a conexão é meramente usada para limpeza e configuração de ambiente.

No exemplo anterior criamos uma conexão Sqlite na memória e a passamos ao método `createDefaultDBConnection` que embrulha a instância do PDO e o segundo parâmetro (o nome do banco de dados) em uma camada simples de abstração para conexões do banco de dados do tipo `PHPUnit_Extensions_Database_DB_IDatabaseConnection`.

A seção "Usando a Conexão de Banco de Dados" explica a API desta interface e como você pode usá-la da melhor forma possível.

8.4.2 Implementando getDataSet()

O método `getDataSet()` define como deve ser o estado inicial do banco de dados antes de cada teste ser executado. O estado do banco de dados é abstraído através de conceitos DataSet (Conjunto de Dados) e DataTable (Tabela de Dados), ambos sendo representados pelas interfaces `PHPUnit_Extensions_Database_DataSet_IDataSet` e `PHPUnit_Extensions_Database_DataSet_IDataTable`. A próxima seção vai descrever em detalhes como esses conceitos trabalham e quais os benefícios de usá-los nos testes com bancos de dados.

Para a implementação precisaremos apenas saber que o método `getDataSet()` é chamado uma vez durante o `setUp()` para recuperar o conjunto de dados do ambiente e inseri-lo no banco de dados. No exemplo estamos usando um método de fábrica `createFlatXMLDataSet($filename)` que representa um conjunto de dados através de uma representação XML.

8.4.3 E quanto ao Esquema do Banco de Dados (DDL)?

O PHPUnit assume que o esquema do banco de dados com todas as suas tabelas, gatilhos, sequências e visualizações é criado antes que um teste seja executado. Isso quer dizer que você como desenvolvedor deve se certificar que o banco de dados está corretamente configurado antes de executar a suíte.

Existem vários meios para realizar esta pré-condição para testar bancos de dados.

1. Se você está usando um banco de dados persistente (não Sqlite Memory) você pode facilmente configurar o banco de dados uma vez com ferramentas como phpMyAdmin para MySQL e reutilizar o banco de dados para cada execução de teste.
2. Se você estiver usando bibliotecas como Doctrine 2 ou Propel você pode usar suas APIs para criar o esquema de banco de dados que precisa antes de rodar os testes. Você pode utilizar as capacidades de Configuração e Bootstrap do PHPUnit para executar esse código sempre que seus testes forem executados.

8.4.4 Dica: Use seu próprio Caso Abstrato de Teste de Banco de Dados

Do exemplo prévio de implementação você pode facilmente perceber que o método `getConnection()` é bastante estático e pode ser reutilizado em diferentes casos de teste de banco de dados. Adicionalmente para manter uma boa performance dos seus testes e pouca carga sobre seu banco de dados, você pode refatorar o código um pouco para obter um caso de teste abstrato genérico para sua aplicação, o que ainda permite você especificar um ambiente de dados diferente para cada caso de teste:

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

abstract class MyApp_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // só instancia o pdo uma vez para limpeza de teste e carregamento de ambiente
    static private $pdo = null;

    // só instancia PHPUnit_Extensions_Database_DB_IDatabaseConnection uma vez por␣
↪teste
    private $conn = null;

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

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

Contudo, isso tem a conexão ao banco de dados codificada na conexão do PDO. O PHPUnit tem outra incrível característica que pode fazer este caso de teste ainda mais genérico. Se você usar a Configuração XML você pode tornar a conexão com o banco de dados configurável por execução de teste. Primeiro vamos criar um arquivo "phpunit.xml" em nosso diretório/de/teste da aplicação, de forma semelhante a isto:

```
<?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>
```

Agora podemos modificar seu caso de teste para parecer com isso:

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

abstract class Generic_Tests_DatabaseTestCase extends TestCase
{
    use TestCaseTrait;

    // só instancia o pdo uma vez para limpeza de teste e carregamento de ambiente
    static private $pdo = null;

    // só instancia PHPUnit_Extensions_Database_DB_IDatabaseConnection uma vez por␣
↪teste
    private $conn = null;

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

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

Agora podemos executar a suíte de testes de banco de dados usando diferentes configurações através da interface de linha-de-comando:

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

A possibilidade de executar facilmente os testes de banco de dados contra diferentes alvos é muito importante se você está desenvolvendo na máquina de desenvolvimento. Se vários desenvolvedores executarem os testes de banco de dados contra a mesma conexão de banco de dados você experimentará facilmente falhas de testes devido à condição de execução.

8.5 Entendendo Conjunto de Dados e Tabelas de Dados

Um conceito central da Extensão para Banco de Dados do PHPUnit são os Conjuntos de Dados e as Tabelas de Dados. Você deveria tentar entender este conceito simples para dominar os testes de banco de dados com PHPUnit. Conjunto de Dados e Tabela de Dados formam uma camada abstrata em torno das tabelas, linhas e colunas do seu banco de dados. Uma simples API esconde os conteúdos subjacentes do banco de dados em uma estrutura de objetos, que também podem ser implementada por outra fonte que não seja um banco de dados.

Essa abstração é necessária para comparar os conteúdos reais de um banco de dados contra os conteúdos esperados. Expectativas podem ser representadas como arquivos XML, YAML, CSV ou vetores PHP, por exemplo. As interfaces DataSet (Conjunto de Dados) e DataTable (Tabela de Dados) permitem a comparação dessas fontes conceitualmente diferentes, emulando o armazenamento de banco de dados relacional em uma abordagem semanticamente similar.

Um fluxo de trabalho para asserções em banco de dados em seus testes consiste em três etapas simples:

- Especificar uma ou mais tabelas em seu banco de dados por nome de tabela (conjunto de dados real)
- Especificar o Conjunto de Dados esperado no seu formato preferido (YAML, XML, . . .)
- Asseverar que ambas as representações de conjunto de dados se equivalem.

Asserções não são o único caso de uso para o Conjunto de Dados e a Tabela de Dados na Extensão para Banco de Dados do PHPUnit. Como mostrado na seção anterior, eles também descrevem os conteúdos iniciais de um banco de dados. Você é forçado a definir um conjunto de dados de ambiente pelo Caso de Teste de Banco de Dados, que então é usado para:

- Deletar todas as linhas das tabelas especificadas no conjunto de dados.
- Escrever todas as linhas nas tabelas de dados do banco de dados.

8.5.1 Implementações disponíveis

Existem três tipos diferentes de conjuntos de dados/tabelas de dados:

- Conjuntos de Dados e Tabelas de Dados baseados em arquivo
- Conjuntos de Dados e Tabelas de Dados baseados em query
- Filtro e Composição de Conjunto de Dados e Tabelas de Dados

Os Conjuntos de Dados e Tabelas de Dados baseadas em arquivo são geralmente usados para o ambiente inicial e para descrever os estados esperados do banco de dados.

Conjunto de Dados de XML Plano

O Conjunto de Dados mais comum é chamado XML Plano. É um formato de xml muito simples onde uma tag dentro do nó-raiz `<dataset>` representa exatamente uma linha no banco de dados. Os nomes das tags equivalem à tabela onde inserir a linha e um atributo representa a coluna. Um exemplo para uma simples aplicação de livro de visitas poderia se parecer com isto:

```
<?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>
```

Isso é, obviamente, fácil de escrever. Aqui `<guestbook>` é o nome da tabela onde duas linhas são inseridas onde cada qual com quatro colunas "id", "content", "user" e "created" com seus respectivos valores.

Porém essa simplicidade tem um preço.

O exemplo anterior não deixa tão óbvio como você pode fazer para especificar uma tabela vazia. Você pode inserir uma tag sem atributos com o nome da tabela vazia. Um arquivo xml plano para uma tabela vazia do livro de visitas ficaria parecido com isso:

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

A manipulação de valores NULL com o xml plano é tedioso. Um valor NULL é diferente de uma string com valor vazio em quase todos os bancos de dados (Oracle é uma exceção), algo difícil de descrever no formato xml plano. Você pode representar um valor NULL omitindo o atributo da especificação da linha. Se nosso livro de visitas vai permitir entradas anônimas representadas por um valor NULL na coluna user, um estado hipotético para a tabela do livro de visitas seria parecido com:

```
<?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>
```

Nesse caso a segunda entrada é postada anonimamente. Porém isso acarreta um problema sério no reconhecimento de colunas. Durante as asserções de igualdade do conjunto de dados, cada conjunto de dados tem que especificar quais colunas uma tabela possui. Se um atributo for NULL para todas as linhas de uma tabela de dados, como a Extensão para Banco de Dados vai saber que a coluna deve ser parte da tabela?

O conjunto de dados em xml plano faz uma presunção crucial agora, definindo que os atributos na primeira linha definida de uma tabela define as colunas dessa tabela. No exemplo anterior isso significaria que "id", "content", "user" e "created" são colunas da tabela guestbook. Para a segunda linha, onde "user" não está definido, um NULL seria inserido no banco de dados.

Quando a primeira entrada do guestbook for apagada do conjunto de dados, apenas "id", "content" e "created" seriam colunas da tabela guestbook, já que "user" não é especificado.

Para usar o Conjunto de Dados em XML Plano efetivamente, quando valores NULL forem relevantes, a primeira linha de cada tabela não deve conter qualquer valor NULL e apenas as linhas seguintes poderão omitir atributos. Isso pode parecer estranho, já que a ordem das linhas é um fator relevante para as asserções com bancos de dados.

Em troca, se você especificar apenas um subconjunto das colunas da tabela no Conjunto de Dados do XML Plano, todos os valores omitidos serão definidos para seus valores padrão. Isso vai induzir a erros se uma das colunas omitidas estiver definida como "NOT NULL DEFAULT NULL".

Para concluir eu posso dizer que os conjuntos de dados XML Plano só devem ser usadas se você não precisar de valores NULL.

Você pode criar uma instância de conjunto de dados xml plano de dentro de seu Caso de Teste de Banco de Dados chamando o método `createFlatXmlDataSet($filename)`:

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

class MyTestCase extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    use TestCaseTrait;

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

Conjunto de Dados XML

Existe um outro Conjunto de Dados em XML mais estruturado, que é um pouco mais verboso para escrever mas evita os problemas do NULL nos conjuntos de dados em XML Plano. Dentro do nó-raiz `<dataset>` você pode especificar as tags `<table>`, `<column>`, `<row>`, `<value>` e `<null />`. Um Conjunto de Dados equivalente ao definido anteriormente no Guestbook em XML Plano seria como:

```
<?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>
```

Qualquer `<table>` definida tem um nome e requer uma definição de todas as colunas com seus nomes. Pode conter zero ou qualquer número positivo de elementos `<row>` aninhados. Não definir nenhum elemento `<row>` significa que a tabela está vazia. As tags `<value>` e `<null />` têm que ser especificadas na ordem dos elementos fornecidos previamente em `<column>`. A tag `<null />` obviamente significa que o valor é NULL.

Você pode criar uma instância de conjunto de dados xml de dentro de seu Caso de Teste de Banco de Dados chamando o método `createXmlDataSet($filename)`:

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

class MyTestCase extends TestCase
{
    use TestCaseTrait;
```

(continues on next page)

(continuação da página anterior)

```
    public function getDataSet()
    {
        return $this->createXMLDataSet('myXmlFixture.xml');
    }
}
?>
```

Conjunto de Dados XML MySQL

Este novo formato XML é específico para o servidor de banco de dados MySQL. O suporte para ele foi adicionado no PHPUnit 3.5. Arquivos nesse formato podem ser gerados usando o utilitário `mysqldump <http://dev.mysql.com/doc/refman/5.0/en/mysqldump.html>‘_. Diferente dos conjuntos de dados CSV, que o `mysqldump` também suporta, um único arquivo neste formato XML pode conter dados para múltiplas tabelas. Você pode criar um arquivo nesse formato invocando o `mysqldump` desta forma:

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

Esse arquivo pode ser usado em seu Caso de Teste de Banco de Dados chamando o método `createMySQLXMLDataSet($filename)`:

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

class MyTestCase extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        return $this->createMySQLXMLDataSet('/path/to/file.xml');
    }
}
?>
```

Conjunto de Dados YAML

Alternativamente, você pode usar o Conjunto de dados YAML para o exemplo guestbook:

```
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
```

Isso é simples, conveniente E resolve o problema do NULL que o Conjunto de Dados similar do XML Plano tem. Um NULL em um YAML é apenas o nome da coluna sem nenhum valor especificado. Uma string vazia é especificada

como `column1: ""`.

O Conjunto de Dados YAML atualmente não possui método fábrica no Caso de Teste de Banco de Dados, então você tem que instanciar manualmente:

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

class YamlGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
        return new YamlDataSet(dirname(__FILE__)."/_files/guestbook.yml");
    }
}
?>
```

Conjunto de Dados CSV

Outro Conjunto de Dados baseado em arquivo é baseado em arquivos CSV. Cada tabela do conjunto de dados é representada como um único arquivo CSV. Para nosso exemplo do guestbook, vamos definir um arquivo guestbook-table.csv:

```
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"
```

Apesar disso ser muito conveniente para edição no Excel ou OpenOffice, você não pode especificar valores NULL em um Conjunto de Dados CSV. Uma coluna vazia levaria a um valor vazio padrão de banco de dados a ser inserido na coluna.

Você pode criar um Conjunto de Dados CSV chamando:

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

class CsvGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
        $dataSet = new CsvDataSet();
        $dataSet->addTable('guestbook', dirname(__FILE__)."/_files/guestbook.csv");
        return $dataSet;
    }
}
?>
```

Conjunto de Dados em Vetor

Não existe Conjunto de Dados baseado em vetor na Extensão de Banco de Dados do PHPUnit (ainda), mas podemos implementar a nossa própria facilmente. Nosso exemplo do guestbook deveria ficar parecido com:

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

class ArrayGuestbookTest extends TestCase
{
    use TestCaseTrait;

    protected function getDataSet()
    {
        return new MyApp_DbUnit_ArrayDataSet(
            [
                'guestbook' => [
                    [
                        'id' => 1,
                        'content' => 'Hello buddy!',
                        'user' => 'joe',
                        'created' => '2010-04-24 17:15:23'
                    ],
                    [
                        'id' => 2,
                        'content' => 'I like it!',
                        'user' => null,
                        'created' => '2010-04-26 12:14:20'
                    ],
                ],
            ]
        );
    }
}
?>
```

Um Conjunto de Dados do PHP tem algumas vantagens óbvias sobre todos os outros conjuntos de dados baseados em arquivos:

- Vetores PHP podem, obviamente, trabalhar com valores `NULL`.
- Você não precisa de arquivos adicionais para asserções e pode especificá-las diretamente no TestCase (na classe de Caso de Teste).

Para este Conjunto de Dados, como nos Conjuntos de Dados em XML Plano, CSV e YAML, as chaves da primeira linha especificada definem os nomes das colunas das tabelas, que no caso anterior seriam “id“, “content“, “user“ e “created“.

A implementação para esse Conjunto de Dados é simples e direta:

```
<?php
class MyApp_DbUnit_ArrayDataSet extends PHPUnit_Extensions_Database_DataSet_
↪AbstractDataSet
{
    /**
     * @var array
     */
    protected $tables = [];
```

(continues on next page)

(continuação da página anterior)

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

            $metaData = new PHPUnit_Extensions_Database_DataSet_DefaultTableMetaData(
↪$tableName, $columns);
            $table = new PHPUnit_Extensions_Database_DataSet_DefaultTable($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->
↪tables, $reverse);
    }

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

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

Conjunto de Dados Query (SQL)

Para asserções de banco de dados você não precisa somente de conjuntos de dados baseados em arquivos, mas também de conjuntos de dados baseados em Query/SQL que contenha os conteúdos reais do banco de dados. É aí que entra o Query DataSet:

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

Adicionar uma tabela apenas por nome é um modo implícito de definir a tabela de dados com a seguinte query:

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

Você pode fazer uso disso especificando queries arbitrárias para suas tabelas, por exemplo restringindo linhas, colunas, ou adicionando cláusulas `ORDER BY`:

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

A seção nas Asserções de Banco de Dados mostrará mais alguns detalhes sobre como fazer uso do Conjunto de Dados Query.

Conjunto de Dados de Banco de Dados (BD)

Acessando a Conexão de Teste você pode criar automaticamente um Conjunto de Dados que consiste de todas as tabelas com seus conteúdos no banco de dados especificado como segundo parâmetro ao método Fábrica de Conexões.

Você pode tanto criar um Conjunto de Dados para todo o banco de dados como mostrado em `testGuestbook()`, ou restringi-lo a um conjunto de nomes específicos de tabelas com uma lista branca, como mostrado no método `testFilteredGuestbook()`.

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

class MySqlGuestbookTest extends TestCase
{
    use TestCaseTrait;

    /**
     * @return PHPUnit_Extensions_Database_DB_IDatabaseConnection
     */
    public function getConnection()
    {
        $database = 'my_database';
        $user = 'my_user';
        $password = 'my_password';
        $pdo = new PDO('mysql:...', $user, $password);
        return $this->createDefaultDBConnection($pdo, $database);
    }

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

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

(continues on next page)

(continuação da página anterior)

```
    }
}
?>
```

Conjunto de Dados de Substituição

Eu tenho falado sobre problemas com NULL nos Conjunto de Dados XML Plano e CSV, mas existe uma alternativa um pouco complicada para fazer ambos funcionarem com NULLs.

O Conjunto de Dados de Substituição é um decorador para um Conjunto de Dados existente e permite que você substitua valores em qualquer coluna do conjunto de dados por outro valor de substituição. Para fazer nosso exemplo do guestbook funcionar com valores NULL devemos especificar o arquivo como:

```
<?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>
```

Então envolvemos o Conjunto de Dados em XML Plano dentro de um Conjunto de Dados de Substituição:

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

class ReplacementTest extends TestCase
{
    use TestCaseTrait;

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

Filtro de Conjunto de Dados

Se você tiver um arquivo grande de ambiente você pode usar o Filtro de Conjunto de Dados para as listas branca e negra das tabelas e colunas que deveriam estar contidas em um sub-conjunto de dados. Isso ajuda especialmente em combinação com o Conjunto de dados DB para filtrar as colunas dos conjuntos de dados.

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

class DataSetFilterTest extends TestCase
{
```

(continues on next page)

(continuação da página anterior)

```
    use TestCaseTrait;

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

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

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

        $filterDataSet = new PHPUnit_Extensions_Database_DataSet_DataSetFilter(
↪$dataSet);
        $filterDataSet->addExcludeTables(['foo', 'bar', 'baz']); // only keep the␣
↪guestbook table!
        $filterDataSet->setExcludeColumnsForTable('guestbook', ['user', 'created']);
        // ..
    }
}
?>

*NOTA* Você não pode usar ambos os
filtros de incluir e excluir coluna na mesma tabela, apenas em tabelas
diferentes. E mais: só é possível para a lista branca ou negra,
mas não para ambas.
```

Conjunto de Dados Composto

O Conjunto de Dados composto é muito útil para agregar vários conjuntos de dados já existentes em um único Conjunto de Dados. Quando vários conjuntos de dados contém as mesmas tabelas, as linhas são anexadas na ordem especificada. Por exemplo se tivermos dois conjuntos de dados *fixture1.xml*:

```
<?xml version="1.0" ?>
<dataset>
    <guestbook id="1" content="Hello buddy!" user="joe" created="2010-04-24 17:15:23"␣
↪/>
</dataset>
```

e *fixture2.xml*:

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

Usando o Conjunto de Dados Composto podemos agregar os dois arquivos de ambiente:

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

class CompositeTest extends TestCase
{
    use TestCaseTrait;

    public function getDataSet()
    {
        $ds1 = $this->createFlatXmlDataSet('fixture1.xml');
        $ds2 = $this->createFlatXmlDataSet('fixture2.xml');

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

        return $compositeDs;
    }
}
?>
```

8.5.2 Cuidado com Chaves Estrangeiras

Durante a Configuração do Ambiente a Extensão para Banco de Dados do PHPUnit insere as linhas no banco de dados na ordem que são especificadas em seu ambiente. Se seu esquema de banco de dados usa chaves estrangeiras isso significa que você tem que especificar as tabelas em uma ordem que não faça as restrições das chaves estrangeiras falharem.

8.5.3 Implementando seus próprios Conjuntos de Dados/ Tabelas de Dados

Para entender os interiores dos Conjuntos de Dados e Tabelas de Dados, vamos dar uma olhada na interface de um Conjunto de Dados. Você pode pular esta parte se você não planeja implementar seu próprio Conjunto de Dados ou Tabela de Dados.

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

A interface pública é usada internamente pela asserção `assertDataSetsEqual()` no Caso de Teste de Banco de Dados para verificar a qualidade do conjunto de dados. Da interface `IteratorAggregate` o IDataSet herda o método `getIterator()` para iterar sobre todas as tabelas do conjunto de dados. O iterador reverso permite o PHPUnit truncar corretamente as tabelas em ordem reversa à que foi criada para satisfazer as restrições de chaves estrangeiras.

Dependendo da implementação, diferentes abordagens são usadas para adicionar instâncias de tabela a um Conjunto de Dados. Por exemplo, tabelas são adicionadas internamente durante a construção a partir de um arquivo fonte em

todos os conjuntos de dados baseados em arquivo como `YamlDataSet`, `XmlDataSet` ou `FlatXmlDataSet`.

Uma tabela também é representada pela seguinte interface:

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

Com exceção do método `getTableMetaData()` isso é bastante auto-explicativo. Os métodos usados são todos requeridos para as diferentes asserções da Extensão para Banco de Dados que são explicados no próximo capítulo. O método `getTableMetaData()` deve retornar uma implementação da interface `PHPUnit_Extensions_Database_DataSet_ITableMetaData` que descreve a estrutura da tabela. Possui informações sobre:

- O nome da tabela
- Um vetor de nomes de colunas da tabela, ordenado por suas aparições nos conjuntos de resultados.
- Um vetor de colunas de chaves-primárias.

Essa interface também tem uma asserção que verifica se duas instâncias de Metadados de Tabela se equivalem, o que é usado pela asserção de igualdade do conjunto de dados.

8.6 A API de Conexão

Existem três métodos interessantes na interface Connection que devem ser retornados do método `getConnection()` no Caso de Teste de Banco de Dados:

```
<?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. O método `createDataSet()` cria um Conjunto de Dados de Banco de Dados (BD) como descrito na seção de implementações de Conjunto de Dados.

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

   class ConnectionTest extends TestCase
   {
       use TestCaseTrait;

   ```

(continues on next page)

(continuação da página anterior)

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

2. O método `createQueryTable()` pode ser usado para criar instâncias de uma QueryTable, dado um nome de resultado e uma query SQL. Este é um método conveniente quando se fala sobre asserções de resultado/tabela como será mostrado na próxima seção de API de Asserções de Banco de Dados.

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

class ConnectionTest extends TestCase
{
    use TestCaseTrait;

    public function testCreateQueryTable()
    {
        $tableNames = ['guestbook'];
        $queryTable = $this->getConnection()->createQueryTable('guestbook',
↪'SELECT * FROM guestbook');
    }
}
?>
```

3. O método `getRowCount()` é uma forma conveniente de acessar o número de linhas em uma tabela, opcionalmente filtradas por uma cláusula where adicional. Isso pode ser usado com uma simples asserção de igualdade:

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

class ConnectionTest extends TestCase
{
    use TestCaseTrait;

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

8.7 API de Asserções de Banco de Dados

Para uma ferramenta de testes, a Extensão para Banco de Dados certamente fornece algumas asserções que você pode usar para verificar o estado atual do banco de dados, tabelas e a contagem de linhas de tabelas. Esta seção descreve essa funcionalidade em detalhes:

8.7.1 Asseverando a contagem de linhas de uma Tabela

Às vezes ajuda verificar se uma tabela contém uma quantidade específica de linhas. Você pode conseguir isso facilmente sem colar códigos adicionais usando a API de Conexão. Suponha que queiramos verificar se após a inserção de uma linha em nosso guestbook não temos somente as duas entradas iniciais que nos acompanharam em todos os exemplos anteriores, mas uma terceira:

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

class GuestbookTest extends TestCase
{
    use TestCaseTrait;

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

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

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

8.7.2 Asseverando o Estado de uma Tabela

A asserção anterior ajuda, mas certamente queremos verificar os conteúdos reais da tabela para verificar se todos os valores foram escritos nas colunas corretas. Isso pode ser conseguido com uma asserção de tabela.

Para isso vamos definir uma instância de Tabela Query que deriva seu conteúdo de um nome de tabela e de uma query SQL e compara isso a um Conjunto de Dados baseado em Arquivo/Vetor:

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

class GuestbookTest extends TestCase
{
    use TestCaseTrait;

    public function testAddEntry()
    {
        $guestbook = new Guestbook();
        $guestbook->addEntry("suzy", "Hello world!");

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

(continues on next page)

(continuação da página anterior)

```
    }
}
?>
```

Agora temos que escrever o arquivo XML Plano *expectedBook.xml* para esta asserção:

```
<?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>
```

Apesar disso, esta asserção só vai passar em exatamente um segundo do universo, em *2010–05–01 21:47:08*. Datas possuem um problema especial nos testes de bancos de dados e podemos circundar a falha omitindo a coluna "created" da asserção.

O arquivo ajustado *expectedBook.xml* em XML Plano provavelmente vai ficar parecido com o seguinte para fazer a asserção passar:

```
<?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>
```

Nós temos que consertar a chamada da Tabela Query:

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

8.7.3 Asseverando o Resultado de uma Query

Você também pode asseverar o resultado de queries complexas com a abordagem da Tabela Query, apenas especificando um nome de resultado com uma query e comparando isso a um conjunto de dados:

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

class ComplexQueryTest extends TestCase
{
    use TestCaseTrait;

    public function testComplexQuery()
    {
        $queryTable = $this->getConnection()->createQueryTable(
            'myComplexQuery', 'SELECT complexQuery...'
```

(continues on next page)

(continuação da página anterior)

```
        );
        $expectedTable = $this->createFlatXmlDataSet("complexQueryAssertion.xml")
                              ->getTable("myComplexQuery");
        $this->assertTablesEqual($expectedTable, $queryTable);
    }
}
?>
```

8.7.4 Asseverando o Estado de Múltiplas Tabelas

Certamente você pode asseverar o estado de múltiplas tabelas de uma vez e comparar um conjunto de dados de query contra um conjunto de dados baseado em arquivo. Existem duas formas diferentes para asserções de Conjuntos de Dados.

1. Você pode usar o Database (Banco de Dados - DB) e o DataSet (Conjunto de Dados) da Connection (Conexão) e compará-lo com um Conjunto de Dados Baseado em Arquivo.

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

class DataSetAssertionsTest extends TestCase
{
    use TestCaseTrait;

    public function testCreateDataSetAssertion()
    {
        $dataSet = $this->getConnection()->createDataSet(['guestbook']);
        $expectedDataSet = $this->createFlatXmlDataSet('guestbook.xml');
        $this->assertDataSetsEqual($expectedDataSet, $dataSet);
    }
}
?>
```

2. Você pode construir o Conjunto de Dados por si próprio:

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

class DataSetAssertionsTest extends TestCase
{
    use TestCaseTrait;

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

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

8.8 Perguntas Mais Frequentes

8.8.1 O PHPUnit vai (re)criar o esquema do banco de dados para cada teste?

Não, o PHPUnit exige que todos os objetos do banco de dados estejam disponíveis quando a suíte começar os testes. O Banco de Dados, tabelas, sequências, gatilhos e visualizações devem ser criadas antes que você execute a suíte de testes.

Doctrine 2 ou eZ Components possuem ferramentas poderosas que permitem você criar o esquema de banco de dados através de estruturas de dados pré-definidas. Entretanto, devem ser ligados à extensão do PHPUnit para permitir a recriação automática de banco de dados antes que a suíte de testes completa seja executada.

Já que cada teste limpa completamente o banco de dados, você nem sequer é forçado a recriar o banco de dados para cada execução de teste. Um banco de dados permanentemente disponível funciona perfeitamente.

8.8.2 Sou forçado a usar PDO em minha aplicação para que a Extensão para Banco de Dados funcione?

Não, PDO só é exigido para limpeza e configuração do ambiente e para asserções. Você pode usar qualquer abstração de banco de dados que quiser dentro de seu próprio código.

8.8.3 O que posso fazer quando recebo um Erro "Too much Connections"?

Se você não armazena em cache a instância de PDO que é criada a partir do método do Caso de Teste `getConnection()` o número de conexões ao banco de dados é aumentado em um ou mais com cada teste do banco de dados. Com a configuração padrão o MySQL só permite 100 conexões concorrentes e outros fornecedores também têm um limite máximo de conexões.

A Sub-seção "Use seu próprio Caso Abstrato de Teste de Banco de Dados" mostra como você pode prevenir o acontecimento desse erro usando uma instância única armazenada em cache do PDO em todos os seus testes.

8.8.4 Como lidar com NULL usando Conjuntos de Dados XML Plano / CSV?

Não faça isso. Em vez disso, você deveria usar Conjuntos de Dados XML ou YAML.

CAPÍTULO 9

Dublês de Testes

Gerard Meszaros introduz o conceito de Dublês de Testes em Meszaros2007 desta forma:

> *Gerard Meszaros*:
>
> Às vezes é muito difícil testar o sistema sob teste (SST - em inglês: system under test - SUT) porque isso depende de outros ambientes que não podem ser usados no ambiente de testes. Isso pode ser porque não estão disponíveis, não retornarão os resultados necessários para o teste, ou porque executá-los causaria efeitos colaterais indesejáveis. Em outros casos, nossa estratégia de testes requer que tenhamos mais controle ou visibilidade do comportamento interno do SST.
>
> Quando estamos escrevendo um teste no qual não podemos (ou decidimos não) usar um componente dependente (depended-on component - DOC) real, podemos substituí-lo por um Dublê de Teste. O Dublê de Teste não precisa se comportar exatamente como o DOC real; apenas precisa fornecer a mesma API como o real, de forma que o SST pense que é o real!

Os métodos `createMock($type)` e `getMockBuilder($type)` fornecidos pelo PHPUnit podem ser usados em um teste para gerar automaticamente um objeto que possa atuar como um dublê de teste para a classe original especificada. Esse objeto de dublê de teste pode ser usado em cada contexto onde um objeto da classe original é esperado ou requerido.

O método `createMock($type)` imediatamente retorna um objeto de dublê de teste para o tipo especificado (interface ou classe). A criação desse dublê de teste é realizada usando os padrões de boas práticas (os métodos `__construct()` e `__clone()` da classe original não são executados) e os argumentos passados para um método do dublê de teste não serão clonados. Se esses padrões não são o que você precisa então você pode usar o método `getMockBuilder($type)` para customizar a geração do dublê de teste usando uma interface fluente.

Por padrão, todos os métodos da classe original são substituídos com uma implementação simulada que apenas retorna `null` (sem chamar o método original). Usando o método `will($this->returnValue())`, por exemplo, você pode configurar essas implementações simuladas para retornar um valor quando chamadas.

Limitações: métodos final, private e static

Por favor, note que os métodos `final`, `private` e `static` não podem ser esboçados (stubbed) ou falsificados (mocked). Eles são ignorados pela funcionalidade de dublê de teste do PHPUnit e mantêm seus comportamentos

originais.

9.1 Esboços (stubs)

A prática de substituir um objeto por um dublê de teste que (opcionalmente) retorna valores de retorno configurados é chamada de *stubbing*. Você pode usar um *esboço* para "substituir um componente real do qual o SST depende de modo que o teste tenha um ponto de controle para as entradas indiretas do SST. Isso permite ao teste forçar o SST através de caminhos que não seriam executáveis de outra forma".

Example 9.2 mostra como esboçar chamadas de método e configurar valores de retorno. Primeiro usamos o método `createMock()` que é fornecido pela classe `PHPUnit\Framework\TestCase` para configurar um esboço de objeto que parece com um objeto de `SomeClass` (Example 9.1). Então usamos a Interface Fluente que o PHPUnit fornece para especificar o comportamento para o esboço. Essencialmente, isso significa que você não precisa criar vários objetos temporários e uni-los depois. Em vez disso, você encadeia chamadas de método como mostrado no exemplo. Isso leva a códigos mais legíveis e "fluentes".

Example 9.1: A classe que queremos esboçar

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

class SomeClass
{
    public function doSomething()
    {
        // Faz algo.
    }
}
?>
```

Example 9.2: Esboçando uma chamada de método para retornar um valor fixo

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

class StubTest extends TestCase
{
    public function testStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Configura o esboço.
        $stub->method('doSomething')
             ->willReturn('foo');

        // Chamando $stub->doSomething() agora vai retornar
        // 'foo'.
        $this->assertEquals('foo', $stub->doSomething());
    }
}
?>
```

Limitações: Métodos nomeados como "method"

O exemplo acima só funciona quando a classe original não declara um método nomeado "method".

Se a classe original declara um método nomeado "method" então `$stub->expects($this->any())->method('doSomethi` deve ser usado.

"Atrás dos bastidores" o PHPUnit automaticamente gera uma nova classe PHP que implementa o comportamento desejado quando o método `createMock()` é usado.

Example 9.3 mostra um exemplo de como usar a interface fluente do Mock Builder para configurar a criação do dublê de teste. A configuração desse dublê de teste usa os mesmos padrões de boas práticas usados por `createMock()`.

Example 9.3: A API Mock Builder pode ser usada para configurar a classe de dublê de teste gerada

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

class StubTest extends TestCase
{
    public function testStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->getMockBuilder($originalClassName)
                     ->disableOriginalConstructor()
                     ->disableOriginalClone()
                     ->disableArgumentCloning()
                     ->disallowMockingUnknownTypes()
                     ->getMock();

        // Configura o esboço.
        $stub->method('doSomething')
             ->willReturn('foo');

        // Chamar $stub->doSomething() agora irá retornar
        // 'foo'.
        $this->assertEquals('foo', $stub->doSomething());
    }
}
?>
```

Nos exemplos até agora temos retornado valores simples usando `willReturn($value)`. Essa sintaxe curta é o mesmo que `will($this->returnValue($value))`. Podemos usar variações desta sintaxe longa para alcançar comportamento mais complexo de esboço.

Às vezes você quer retornar um dos argumentos de uma chamada de método (inalterada) como o resultado de uma chamada ao método esboçado. Example 9.4 mostra como você pode conseguir isso usando `returnArgument()` em vez de `returnValue()`.

Example 9.4: Esboçando uma chamada de método para retornar um dos argumentos

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

class StubTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testReturnArgumentStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Configura o esboço.
        $stub->method('doSomething')
             ->will($this->returnArgument(0));

        // $stub->doSomething('foo') retorna 'foo'.
        $this->assertEquals('foo', $stub->doSomething('foo'));

        // $stub->doSomething('bar') retorna 'bar'.
        $this->assertEquals('bar', $stub->doSomething('bar'));
    }
}
?>
```

Ao testar uma interface fluente, às vezes é útil fazer um método esboçado retornar uma referência ao objeto esboçado. Example 9.5 mostra como você pode usar `returnSelf()` para conseguir isso.

Example 9.5: Esboçando uma chamada de método para retornar uma referência ao objeto esboçado

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

class StubTest extends TestCase
{
    public function testReturnSelf()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Configura o esboço.
        $stub->method('doSomething')
             ->will($this->returnSelf());

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

Algumas vezes um método esboçado deveria retornar valores diferentes dependendo de uma lista predefinida de argumentos. Você pode usar `returnValueMap()` para criar um mapa que associa argumentos com valores de retorno correspondentes. Veja Example 9.6 para ter um exemplo.

Example 9.6: Esboçando uma chamada de método para retornar o valor de um mapa

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

class StubTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testReturnValueMapStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Cria um mapa de argumentos para valores retornados.
        $map = [
            ['a', 'b', 'c', 'd'],
            ['e', 'f', 'g', 'h']
        ];

        // Configura o esboço.
        $stub->method('doSomething')
             ->will($this->returnValueMap($map));

        // $stub->doSomething() retorna diferentes valores dependendo do
        // argumento fornecido.
        $this->assertEquals('d', $stub->doSomething('a', 'b', 'c'));
        $this->assertEquals('h', $stub->doSomething('e', 'f', 'g'));
    }
}
?>
```

Quando a chamada ao método esboçado deve retornar um valor calculado em vez de um fixo (veja `returnValue()`) ou um argumento (inalterado) (veja `returnArgument()`), você pode usar `returnCallback()` para que o método esboçado retorne o resultado da função ou método callback. Veja Example 9.7 para ter um exemplo.

Example 9.7: Esboçando uma chamada de método para retornar um valor de um callback

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

class StubTest extends TestCase
{
    public function testReturnCallbackStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Configura o esboço.
        $stub->method('doSomething')
             ->will($this->returnCallback('str_rot13'));

        // $stub->doSomething($argument) retorna str_rot13($argument)
        $this->assertEquals('fbzrguvat', $stub->doSomething('something'));
    }
}
?>
```

Uma alternativa mais simples para configurar um método callback pode ser especificar uma lista de valores de retorno desejados. Você pode fazer isso com o método `onConsecutiveCalls()`. Veja Example 9.8 para ter um exemplo.

Example 9.8: Esboçando uma chamada de método para retornar uma lista de valores na ordem especificada

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

class StubTest extends TestCase
{
    public function testOnConsecutiveCallsStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

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

        // $stub->doSomething() retorna um valor diferente a cada vez
        $this->assertEquals(2, $stub->doSomething());
        $this->assertEquals(3, $stub->doSomething());
        $this->assertEquals(5, $stub->doSomething());
    }
}
?>
```

Em vez de retornar um valor, um método esboçado também pode causar uma exceção. Example 9.9 mostra como usar `throwException()` para fazer isso.

Example 9.9: Esboçando uma chamada de método para lançar uma exceção

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

class StubTest extends TestCase
{
    public function testThrowExceptionStub()
    {
        // Cria um esboço para a classe SomeClass.
        $stub = $this->createMock(SomeClass::class);

        // Configura o esboço.
        $stub->method('doSomething')
             ->will($this->throwException(new Exception));

        // $stub->doSomething() lança Exceção
        $stub->doSomething();
    }
}
?>
```

Alternativamente, você mesmo pode escrever um esboço enquanto melhora o design. Recursos amplamente utilizados são acessados através de uma única fachada, então você pode substituir facilmente o recurso pelo esboço. Por exemplo, em vez de ter chamadas diretas ao banco de dados espalhadas pelo código, você tem um único objeto `Database` que implementa a interface `IDatabase`. Então, você pode criar um esboço de implementação da `IDatabase` e usá-la em seus testes. Você pode até criar uma opção para executar os testes com o esboço do banco de dados ou com o banco de dados real, então você pode usar seus testes tanto para testes locais durante o desenvolvimento quanto para

integração dos testes com o banco de dados real.

Funcionalidades que precisam ser esboçadas tendem a se agrupar no mesmo objeto, aumentando a coesão. Por apresentar a funcionalidade com uma interface única e coerente, você reduz o acoplamento com o resto do sistema.

9.2 Objetos Falsos (Mock Objects)

A prática de substituir um objeto por um dublê de teste que verifica expectativas, por exemplo asseverando que um método foi chamado, é conhecido como *falsificação (mocking)*.

Você pode usar um *objeto falso* "como um ponto de observação que é usado para verificar as saídas indiretas do SST durante seu exercício. Tipicamente, o objeto falso também inclui a funcionalidade de um esboço de teste que deve retornar valores para o SST se ainda não tiver falhado nos testes, mas a ênfase está na verificação das saídas indiretas. Portanto, um objeto falso é muito mais que apenas um esboço de testes mais asserções; é utilizado de uma forma fundamentalmente diferente".

Limitações: Verificação automática de expectativas

Somente objetos falsos gerados no escopo de um teste irá ser verificado automaticamente pelo PHPUnit. Objetos falsos gerados em provedores de dados, por exemplo, ou injetados dentro do teste usando a anotação `@depends` não serão verificados pelo PHPUnit.

Aqui está um exemplo: suponha que queiramos testar se o método correto, `update()` em nosso exemplo, é chamado em um objeto que observa outro objeto. Example 9.10 mostra o código para as classes `Subject` e `Observer` que são parte do Sistema Sob Teste (SST).

Example 9.10: As classes Subject e Observer que são parte do Sistema Sob Teste (SST)

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

class Subject
{
    protected $observers = [];
    protected $name;

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

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

    public function attach(Observer $observer)
    {
        $this->observers[] = $observer;
    }

    public function doSomething()
    {
```

(continues on next page)

(continuação da página anterior)

```
        // Faz algo.
        // ...

        // Notifica aos observadores que fizemos algo.
        $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);
        }
    }

    // Outros métodos.
}

class Observer
{
    public function update($argument)
    {
        // Faz algo.
    }

    public function reportError($errorCode, $errorMessage, Subject $subject)
    {
        // Faz algo.
    }

    // Outros métodos.
}
?>
```

Example 9.11 mostra como usar um objeto falso para testar a interação entre os objetos `Subject` e `Observer`.

Primeiro usamos o método `getMockBuilder()` que é fornecido pela classe `PHPUnit\Framework\TestCase` para configurar um objeto falso para ser o `Observer`. Já que fornecemos um vetor como segundo parâmetro (opcional) para o método `getMock()`, apenas o método `update()` da classe `Observer` é substituído por uma implementação falsificada.

Porque estamos interessados em verificar se um método foi chamado, e com quais argumentos ele foi chamado, introduzimos os métodos `expects()` e `with()` para especificar como essa interação deve se parecer.

Example 9.11: Testando se um método é chamado uma vez e com o argumento especificado

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

class SubjectTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testObserversAreUpdated()
    {
        // Cria uma falsificação para a classe Observer,
        // apenas falsificando o método update().

        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['update'])
                         ->getMock();

        // Configura a expectativa para o método update()
        // para ser chamado apenas uma vez e com a string 'something'
        // como seu parâmetro.
        $observer->expects($this->once())
                 ->method('update')
                 ->with($this->equalTo('something'));

        // Cria um objeto Subject e anexa a ele o objeto
        // Observer falsificado.
        $subject = new Subject('My subject');
        $subject->attach($observer);

        // Chama o método doSomething() no objeto $subject
        // no qual esperamos chamar o método update()
        // do objeto falsificado Observer, com a string 'something'.
        $subject->doSomething();
    }
}
?>
```

O método `with()` pode receber qualquer número de argumentos, correspondendo ao número de argumentos sendo falsos. Você pode especificar restrições mais avançadas do que uma simples igualdade no argumento do método.

Example 9.12: Testando se um método é chamado com um número de argumentos restringidos de formas diferentes

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

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // Cria um mock para a classe Observer, falsificando o
        // método reportError()
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['reportError'])
                         ->getMock();

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

(continues on next page)

(continuação da página anterior)

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

        // O método doSomethingBad() deve reportar um erro ao observer
        // através do método reportError()
        $subject->doSomethingBad();
    }
}
?>
```

O método `withConsecutive()` pode receber qualquer número de vetores de argumentos, dependendo das chamadas que você desejar testar. Cada vetor é uma lista de restrições correspondentes para os argumentos do método falsificado, como em `with()`.

Example 9.13: Testar que um método foi chamado duas vezes com argumentos especificados

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

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

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

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

A restrição `callback()` pode ser usada para verificação de argumento mais complexa. Essa restrição recebe um callback PHP como seu único argumento. O callback PHP receberá o argumento a ser verificado como seu único argumento e deverá retornar `true` se o argumento passou a verificação e `false` caso contrário.

Example 9.14: Verificação de argumento mais complexa

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

class SubjectTest extends TestCase
{
    public function testErrorReported()
    {
        // Cria um mock para a classe Observer, falsificando o
```

(continues on next page)

(continuação da página anterior)

```
        // método reportError()
        $observer = $this->getMockBuilder(Observer::class)
                         ->setMethods(['reportError'])
                         ->getMock();

        $observer->expects($this->once())
                 ->method('reportError')
                 ->with($this->greaterThan(0),
                        $this->stringContains('Something'),
                        $this->callback(function($subject){
                          return is_callable([$subject, 'getName']) &&
                                 $subject->getName() == 'My subject';
                        }));

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

        // O método doSomethingBad() deve reportar um erro ao observer
        // através do método reportError()
        $subject->doSomethingBad();
    }
}
?>
```

Example 9.15: Testando se um método foi chamado uma vez e com o objeto idêntico ao que foi passado

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

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

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

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

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

Example 9.16: Cria um objeto falsificado com clonagem de parâmetros habilitada

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

class FooTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testIdenticalObjectPassed()
    {
        $cloneArguments = true;

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

        // Agora seu mock clona parâmetros tal que a restrição identicalTo
        // irá falhar.
    }
}
?>
```

Restrições mostra as restrições que podem ser aplicadas aos argumentos do método e Table 9.1 mostra os comparados que estão disponíveis para especificar o número de invocações.

Table 9.1: Comparadores

Comparador	Significado
`PHPUnit_Framework_MockObject_Matcher_AnyInvokedCount any()`	Retorna um comparador que corresponde quando o método que é avaliado for executado zero ou mais vezes.
`PHPUnit_Framework_MockObject_Matcher_InvokedCount never()`	Retorna um comparador que corresponde quando o método que é avaliado nunca for executado.
`PHPUnit_Framework_MockObject_Matcher_InvokedAtLeastOnce atLeastOnce()`	Retorna um comparador que corresponde quando o método que é avaliado for executado pelo menos uma vez.
`PHPUnit_Framework_MockObject_Matcher_InvokedCount once()`	Retorna um comparador que corresponde quando o método que é avaliado for executado exatamente uma vez.
`PHPUnit_Framework_MockObject_Matcher_InvokedCount exactly(int $count)`	Retorna um comparador que corresponde quando o método que é avaliado for executado exatamente `$count` vezes.
`PHPUnit_Framework_MockObject_Matcher_InvokedAtIndex at(int $index)`	Retorna um comparador que corresponde quando o método que é avaliado for invocado no `$index` fornecido.

Note

O parâmetro `$index` para o comparador `at()` se refere ao índice, iniciando em zero, em *todas invocações de métodos* para um dado objeto falsificado. Tenha cuidado ao usar este comparador, pois pode levar a testes frágeis que são muito intimamente ligados a detalhes de implementação específicos.

As mentioned in the beginning, when the defaults used by the `createMock()` method to generate the test double do not match your needs then you can use the `getMockBuilder($type)` method to customize the test double generation using a fluent interface. Here is a list of methods provided by the Mock Builder:

- `setMethods(array $methods)` pode ser chamado no objeto Mock Builder para especificar os métodos que devem ser substituídos com um dublê de teste configurável. O comportamento dos outros métodos não muda. Se você chamar `setMethods(null)`, então nenhum dos métodos serão substituídos.
- `setConstructorArgs(array $args)` pode ser chamado para fornecer um vetor de parâmetros que é passado ao construtor da classe original (que por padrão não é substituído com uma implementação falsa).
- `setMockClassName($name)` pode ser usado para especificar um nome de classe para a classe de dublê de teste gerada.

- `disableOriginalConstructor()` pode ser usado para desabilitar a chamada ao construtor da classe original.
- `disableOriginalClone()` pode ser usado para desabilitar a chamada ao construtor clone da classe original.
- `disableAutoload()` pode ser usado para desabilitar o `__autoload()` durante a geração da classe de dublê de teste.

9.3 Profecia

Prophecy é um "framework PHP de falsificação de objetos muito poderoso e flexível, porém altamente opcional. Embora inicialmente criado para atender as necessidades do phpspec2, ele é flexível o suficiente para ser usado dentro de qualquer framework de teste por aí, com o mínimo de esforço".

O PHPUnit tem suporte nativo para uso do Prophecy para criar dublês de testes. Example 9.17 mostra como o mesmo teste mostrado no Example 9.11 pode ser expressado usando a filosofia do Prophecy de profecias e revelações:

Example 9.17: Testando que um método foi chamado uma vez e com um argumento específico

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

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

        // Cria uma profecia para a classe Observer.
        $observer = $this->prophesize(Observer::class);

        // Configura a expectativa para o método update()
        // para que seja chamado somente uma vez e com a string 'something'
        // como parâmetro.
        $observer->update('something')->shouldBeCalled();

        // Revela a profecia e anexa o objeto falsificado
        // ao Subject.
        $subject->attach($observer->reveal());

        // Chama o método doSomething() no objeto $subject
        // que esperamos que chame o método update() do objeto
        // Observer falsificado com a string 'something'.
        $subject->doSomething();
    }
}
?>
```

Por favor, referencie a documentação do Prophecy para mais detalhes sobre como criar, configurar, e usar esboços, espiões, e falsificações usando essa alternativa de framework de dublê de teste.

9.4 Falsificando Traits e Classes Abstratas

O método `getMockForTrait()` retorna um objeto falsificado que usa uma trait especificada. Todos métodos abstratos de uma dada trait são falsificados. Isto permite testar os métodos concretos de uma trait.

Example 9.18: Testando os métodos concretos de uma trait

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

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

    public abstract function abstractMethod();
}

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

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

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

O método `getMockForAbstractClass()` retorna um objeto falso para uma classe abstrata. Todos os métodos abstratos da classe abstrata fornecida são falsificados. Isto permite testar os métodos concretos de uma classe abstrata.

Example 9.19: Testando os métodos concretos de uma classe abstrata

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

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

    public abstract function abstractMethod();
}

class AbstractClassTest extends TestCase
{
    public function testConcreteMethod()
    {
```

(continues on next page)

(continuação da página anterior)

```
        $stub = $this->getMockForAbstractClass(AbstractClass::class);

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

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

9.5 Esboçando e Falsificando Serviços Web

Quando sua aplicação interage com um serviço web você quer testá-lo sem realmente interagir com o serviço web. Para tornar mais fácil o esboço e falsificação dos serviços web, o `getMockFromWsdl()` pode ser usado da mesma forma que o `getMock()` (veja acima). A única diferença é que `getMockFromWsdl()` retorna um esboço ou falsificação baseado em uma descrição de um serviço web em WSDL e `getMock()` retorna um esboço ou falsificação baseado em uma classe ou interface PHP.

Example 9.20 mostra como `getMockFromWsdl()` pode ser usado para esboçar, por exemplo, o serviço web descrito em `GoogleSearch.wsdl`.

Example 9.20: Esboçando um serviço web

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

class GoogleTest extends TestCase
{
    public function testSearch()
    {
        $googleSearch = $this->getMockFromWsdl(
          'GoogleSearch.wsdl', 'GoogleSearch'
        );

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

        $element = new stdClass;
        $element->summary = '';
        $element->URL = 'https://phpunit.de/';
        $element->snippet = '...';
        $element->title = '<b>PHPUnit</b>';
        $element->cachedSize = '11k';
        $element->relatedInformationPresent = true;
        $element->hostName = 'phpunit.de';
        $element->directoryCategory = $directoryCategory;
        $element->directoryTitle = '';

        $result = new stdClass;
        $result->documentFiltering = false;
        $result->searchComments = '';
        $result->estimatedTotalResultsCount = 3.9000;
```

(continues on next page)

(continuação da página anterior)

```
        $result->estimateIsExact = false;
        $result->resultElements = [$element];
        $result->searchQuery = 'PHPUnit';
        $result->startIndex = 1;
        $result->endIndex = 1;
        $result->searchTips = '';
        $result->directoryCategories = [];
        $result->searchTime = 0.248822;

        $googleSearch->expects($this->any())
                     ->method('doGoogleSearch')
                     ->will($this->returnValue($result));

        /**
         * $googleSearch->doGoogleSearch() agora irá retornar um resultado esboçado e
         * o método doGoogleSearch() do serviço web não será invocado.
         */
        $this->assertEquals(
          $result,
          $googleSearch->doGoogleSearch(
            '00000000000000000000000000000000',
            'PHPUnit',
            0,
            1,
            false,
            '',
            false,
            '',
            '',
            ''
          )
        );
    }
}
?>
```

9.6 Esboçando o Sistema de Arquivos

vfsStream é um stream wrapper para um sistema de arquivos virtual que pode ser útil em testes unitários para falsificar um sistema de arquivos real.

Simplesmente adicione a dependência `mikey179/vfsStream` ao seu arquivo `composer.json` do projeto se você usa o Composer para gerenciar as dependências do seu projeto. Aqui é um exemplo simplório de um arquivo `composer.json` que apenas define uma dependência em ambiente de desenvolvimento para o PHPUnit 4.6 e vfsStream:

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

Example 9.21 mostra a classe que interage com o sistema de arquivos.

Example 9.21: Uma classe que interage com um sistema de arquivos

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

class Example
{
    protected $id;
    protected $directory;

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

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

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

Sem um sistema de arquivos virtual tal como o vfsStream não poderíamos testar o método `setDirectory()` isolado de influências externas (veja Example 9.22).

Example 9.22: Testando uma classe que interage com o sistema de arquivos

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

class ExampleTest extends 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');
        }
```

(continues on next page)

(continuação da página anterior)

```
    }
}
?>
```

A abordagem acima tem várias desvantagens:

- Assim como um recurso externo, podem haver problemas intermitentes com o sistema de arquivos. Isso deixa os testes, com os quais interage, esquisitos.
- Nos métodos `setUp()` e `tearDown()` temos que assegurar que o diretório não existe antes e depois do teste.
- Quando a execução do teste termina antes do método `tearDown()` ser invocado, o diretório permanece no sistema de arquivos.

Example 9.23 mostra como o vfsStream pode ser usado para falsificar o sistema de arquivos em um teste para uma classe que interage com o sistema de arquivos.

Example 9.23: Falsificando o sistema de arquivos em um teste para uma classe que interage com o sistema de arquivos

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

class ExampleTest extends 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'));
    }
}
?>
```

Isso tem várias vantagens:

- O próprio teste fica mais conciso.
- O vfsStream concede ao desenvolvedor de testes controle total sobre a aparência do ambiente do sistema de arquivos para o código testado.
- Já que as operações do sistema de arquivos não operam mais no sistema de arquivos real, operações de limpeza em um método `tearDown()` não são mais exigidas.

CAPÍTULO 10

Práticas de Teste

> *Erich Gamma*:
>
> Você sempre pode escrever mais testes. Porém, você vai descobrir rapidamente que apenas uma fração dos testes que você pode imaginar são realmente úteis. O que você quer é escrever testes que falham mesmo quando você acha que eles deveriam funcionar, ou testes que passam mesmo quando você acha que eles deveria falhar. Outra forma de pensar sobre isso é com relação ao custo/benefício. Você quer escrever testes que lhe darão retorno com informação.

10.1 Durante o Desenvolvimento

Quando você precisa fazer uma mudança na estrutura interna do programa em que está trabalhando para torná-lo mais fácil de entender e mais barato de modificar sem alterar seu comportamento visível, uma suíte de testes é inestimável na aplicação das assim chamadas refatorações seguras. De outra forma, você poderia não notar o sistema quebrando enquanto você está cuidando da reestruturação.

As seguintes condições vão ajudá-lo a melhorar o código e design do seu projeto, enquanto usa testes unitários para verificar que os passos de transformação da refatoração são, de fato, preservadores de comportamento e não introduzem erros:

1. Todos os testes unitários são executados corretamente.
2. O código comunica seus princípios de design.
3. O código não contém redundâncias.
4. O código contém o mínimo número de classes e métodos.

Quando você precisar adicionar novas funcionalidades ao sistema, escreva os testes primeiro. Então, você terá terminado de desenvolver quando os testes executarem. Esta prática será discutida em detalhes no próximo capítulo.

10.2 Durante a Depuração

Quando você recebe um relatório de defeito, seu impulso pode ser consertar o defeito o mais rápido possível. A experiência mostra que esse impulso não vai lhe servir bem; é provável que o conserto do defeito acaba causando outro defeito.

Você pode conter esses seus impulsos fazendo o seguinte:

1. Verifique que você pode reproduzir o defeito.
2. Encontre a demonstração em menor escala do defeito no código. Por exemplo, se um número aparece incorretamente em uma saída, encontre o objeto que está calculando esse número.
3. Escreva um teste automatizado que falha agora, mas vai passar quando o defeito for consertado.
4. Conserte o defeito.

Encontrar a menor reprodução confiável do defeito vai te dar a oportunidade de examinar realmente a causa do defeito. O teste que você escreve vai melhorar as chances de que, quando você corrigir o defeito, você realmente terá corrigido-o, porque o novo teste reduz a probabilidade de desfazer a correção com futuras modificações no código. Todos os testes que você escreveu antes reduzem a probabilidade de causar diferentes problemas inadvertidamente.

> *Benjamin Smedberg*:
>
> Testes unitários oferecem muitas vantagens:
>
> - Testar dá aos autores e revisores de código confiança de que remendos produzem os resultados corretos.
> - Criar casos de testes é um bom ímpeto para desenvolvedores descobrirem casos extremos.
> - Testar fornece uma boa maneira de capturar regressões rapidamente, e ter certeza de que nenhuma regressão será repetida duas vezes.
> - Testes unitários fornecem exemplos funcionais de como usar uma API e podem auxiliar significativamente os trabalhos de documentação.
>
> No geral, testes unitários integrados reduzem o custo e o risco de qualquer mudança individual menor. Isso permitirá que o projeto faça [...] maiores mudanças arquitetônicas [...] rápida e confiavelmente.

CAPÍTULO 11

Análise de Cobertura de Código

> *Wikipedia*:
>
> Na ciência da computação, cobertura de código é uma medida usada para descrever o grau em que um código fonte de um programa é testado por uma particular suíte de testes. Um programa com cobertura de código alta foi mais exaustivamente testado e tem uma menor chance de conter erros de software do que um programa com baixa cobertura de código.

Neste capítulo você aprenderá tudo sobre a funcionalidade de cobertura de código do PHPUnit que lhe dará uma perspicácia sobre quais partes do código de produção são executadas quando os testes são executados. Ele faz uso do componente PHP_CodeCoverage, que por sua vez utiliza a funcionalidade de cobertura de código fornecida pela extensão Xdebug para PHP.

Note

Xdebug não é distribuído como parte do PHPUnit. Se você receber um aviso durante a execução de testes que a extensão Xdebug não está carregada, isso significa que Xdebug não está instalada ou não está configurada corretamente. Antes que você possa usar as funcionalidades de análise de cobertura de código no PHPUnit, você deve ler o manual de instalação Xdebug.

PHPUnit pode gerar um relatório de cobertura de código baseado em HTML, tal como arquivos de registros baseado em XML com informações de cobertura de código em vários formatos (Clover, Crap4J, PHPUnit). Informação de cobertura de código também pode ser relatada como texto (e impressa na STDOUT) e exportada como código PHP para futuro processamento.

Por favor, consulte o *O executor de testes em linha-de-comando* para uma lista de opções de linha de comando que controlam a funcionalidade de cobertura de código, bem como em *Registrando* para as definições de configurações relevantes.

11.1 Métricas de Software para Cobertura de Código

Várias métricas de software existem para mensurar a cobertura de código:

Cobertura de Linha

A métrica de software *Cobertura de Linha* mensura se cada linha executável foi executada.

Cobertura de Função e Método

A métrica de software *Cobertura de Função e Método* mensura se cada função ou método foi invocado. PHP_CodeCoverage só considera uma função ou método como coberto quando todas suas linhas executáveis são cobertas.

Cobertura de Classe e Trait

A métrica de software *Cobertura de Classe e Trait* mensura se cada método de uma classe ou trait é coberto. PHP_CodeCoverage só considera uma classe ou trait como coberta quando todos seus métodos são cobertos.

Cobertura de Código de Operação

A métrica de software *Cobertura de Código de Operação* mensura se cada código de operação de uma função ou método foi executado enquanto executa a suíte de teste. Uma linha de código geralmente compila em mais de um código de operação. Cobertura de Linha considera uma linha de código como coberta logo que um dos seus códigos de operações é executado.

Cobertura de Ramo

A métrica de software *Cobertura de Ramo* mensura se cada expressão booleana de cada estrutura de controle avaliou tanto para `true` quanto para `false` enquanto executa a suíte de teste.

Cobertura de Caminho

A métrica de software *Cobertura de Caminho* mensura se cada um dos caminhos de execução possíveis em uma função ou método foi seguido durante a execução da suíte de teste. Um caminho de execução é uma sequência única de ramos a partir da entrada de uma função ou método para a sua saída.

Índice de Anti-Patterns de Mudança de Risco (CRAP - Change Risk Anti-Patterns)

O *Índice de Anti-Patterns de Mudança de Risco (CRAP - Change Risk Anti-Patterns)* é calculado baseado na complexidade ciclomática e cobertura de código de uma unidade de código. Código que não é muito complexo e tem uma cobertura de teste adequada terá um baixo índice CRAP. O índice CRAP pode ser reduzido escrevendo testes e refatorando o código para reduzir sua complexidade.

Note

As métricas de software *Cobertura de Código de Operação*, *Cobertura de Ramo* e *Cobertura de Caminho* ainda não são suportadas pelo PHP_CodeCoverage.

11.2 Lista-branca de arquivos

É obrigatório configurar uma *whitelist* para dizer ao PHPUnit quais arquivos de código-fonte incluir no relatório de cobertura de código. Isso pode ser feito usando a opção de linha de comando `--whitelist` ou via arquivo de configuração (veja *Lista-branca de Arquivos para Cobertura de Código*).

Opcionalmente, todos arquivos da lista-branca podem ser adicionados ao relatório de cobertura de código ao definiri `addUncoveredFilesFromWhitelist="true"` em sua configuração do PHPUnit (veja *Lista-branca de Arquivos para Cobertura de Código*). Isso permite a inclusão de arquivos que não são testados ainda no todo. Se você quer ter informação sobre quais linhas de um tal arquivo não-coberto são executáveis, por exemplo, você também precisa definir `processUncoveredFilesFromWhitelist="true"` em sua configuração do PHPUnit (veja *Lista-branca de Arquivos para Cobertura de Código*).

Note

Por favor, note que o carregamento de arquivos de código-fonte que é realizado, quando `processUncoveredFilesFromWhitelist="true"` é definido, pode causar problemas quando um arquivo de código-fonte contém código fora do escopo de uma classe ou função, por exemplo.

11.3 Ignorando Blocos de Código

Às vezes você tem blocos de código que não pode testar e que pode querer ignorar durante a análise de cobertura de código. O PHPUnit deixa você fazer isso usando as anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd` como mostrado em Example 11.1.

Example 11.1: Usando as anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd`

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

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

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

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

exit; // @codeCoverageIgnore
?>
```

As linhas de código ignoradas (marcadas como ignoradas usando as anotações) são contadas como executadas (se forem executáveis) e não serão destacadas.

11.4 Especificando métodos cobertos

A anotação `@covers` (veja *Anotações para especificar quais métodos são cobertos por um teste*) pode ser usada em um código de teste para especificar qual(is) método(s) um método de teste quer testar. Se fornecido, apenas a informação de cobertura de código para o(s) método(s) especificado(s) será considerada. Example 11.2 mostra um exemplo.

Example 11.2: Testes que especificam quais métodos eles querem cobrir

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

class BankAccountTest extends 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) {
```

(continues on next page)

(continuação da página anterior)

```
            $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());
    }
}
?>
```

Também é possível especificar que um teste não deve cobrir *qualquer* método usando a anotação `@coversNothing` (veja *@coversNothing*). Isso pode ser útil quando escrever testes de integração para certificar-se de que você só gerará cobertura de código com testes unitários.

Example 11.3: Um teste que especifica que nenhum método deve ser coberto

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

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);
    }
}
?>
```

11.5 Casos Extremos

Esta seção mostra casos extremos notáveis que induz a confundir a informação de cobertura de código.

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

// Por ser "baseado em linha" e não em declaração,
// uma linha sempre terá um estado de cobertura
if (false) this_function_call_shows_up_as_covered();

// Devido ao modo como a cobertura de código funciona internamente, estas duas linhas␣
↪são especiais.
// Esta linha vai aparecer como não-executável
if (false)
    // Esta linha vai aparecer como coberta, pois de fato é a cobertura
    // da declaração if da linha anterior que é mostrada aqui!
    will_also_show_up_as_covered();

// Para evitar isso é necessário usar chaves
if (false) {
    this_call_will_never_show_up_as_covered();
}
?>
```

CAPÍTULO 12

Outros Usos para Testes

Uma vez que você se acostumar a escrever testes automatizados, você vai querer descobrir mais usos para testes. Aqui temos alguns exemplos.

12.1 Documentação Ágil

Tipicamente, em um projeto desenvolvido usando um processo ágil, como a Programação Extrema, a documentação não pode se manter com as mudanças frequentes do design e código do projeto. Programação Extrema exige *propriedade coletiva de código*, então todos os desenvolvedores precisam saber como o sistema todo funciona. Se você for disciplinado o suficiente para consequentemente usar "nomes falantes" para seus testes que descrevam o que cada classe deveria fazer, você pode usar a funcionalidade TestDox do PHPUnit para gerar documentação automatizada para seu projeto baseada nos testes. Essa documentação dá aos desenvolvedores uma visão geral sobre o que cada classe do projeto deveria fazer.

A funcionalidade TestDox do PHPUnit olha para uma classe de teste e todos os nomes dos métodos de teste e os converte de nomes camelCase do PHP para sentenças: `testBalanceIsInitiallyZero()` se torna "Balance is initially zero". Se houverem vários métodos de teste cujos nomes apenas diferem em um sufixo de um ou mais dígitos, como em `testBalanceCannotBecomeNegative()` e `testBalanceCannotBecomeNegative2()`, a sentença "Balance cannot become negative" aparecerá apenas uma vez, assumindo-se que todos os testes foram bem-sucedidos.

Vamos dar uma olhada na documentação ágil gerada para a classe `BankAccount`:

```
$ phpunit --testdox BankAccountTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

BankAccount
 [x] Balance is initially zero
 [x] Balance cannot become negative
```

Alternativamente, a documentação ágil pode ser gerada nos formatos HTML ou texto plano e escrita em um arquivo usando os argumentos `--testdox-html` e `--testdox-text`.

A Documentação Ágil pode ser usada para documentar as suposições que você faz sobre os pacotes externos que você usa em seu projeto. Quando você usa um pacote externo, você está exposto ao risco do pacote não se comportar como você espera, e de futuras versões do pacote mudarem de formas súbitas que quebrarão seu código, sem que você saiba. Você pode dar um jeito nesses problemas escrevendo um teste toda vez que fizer uma suposição. Se seu teste for bem sucedido, sua suposição é válida. Se você documentar todas as suas suposições com testes, futuras versões do pacote externo não serão motivo de preocupação: se o teste for bem-sucedido, seu sistema deverá continuar funcionando.

12.2 Testes Inter-Equipes

Quando você documenta suposições com testes, você possui os testes. O fornecedor do pacote – sobre o qual você faz suposições – não sabe nada sobre seus testes. Se você quer ter um relacionamento mais próximo com o fornecedor do pacote, você pode usar os testes para comunicar e coordenar suas atividades.

Quando você concorda em coordenar suas atividades com o fornecedor de um pacote, vocês podem escrever os testes juntos. Faça isso de modo que os testes revelem o máximo possível de suposições. Suposições escondidas são a morte da cooperação. Com os testes você documenta exatamente o que você espera de um pacote fornecido. O fornecedor vai saber que o pacote está completo quando todos os testes executarem.

Usando esboços (veja o capítulo em "Objetos Falsos", anteriormente neste livro), você pode chegar a se desassociar do fornecedor: O trabalho do fornecedor é fazer os testes executarem com a implementação real do pacote. O seu trabalho é fazer os testes executarem para seu próprio código. Até esse momento como você tem a implementação real do pacote fornecido, você usa objetos esboçados. Seguindo esta abordagem, os dois times podem desenvolver independentemente.

CAPÍTULO 13

Registrando

O PHPUnit pode produzir vários tipos de arquivos de registro (logfiles).

13.1 Resultados de Teste (XML)

O arquivo de registro XML para resultados de testes produzidos pelo PHPUnit é baseado naquele usado pela tarefa do JUnit para Apache Ant. O seguinte exemplo mostra o arquivo de registro XML gerado para os testes em `ArrayTest`:

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

O seguinte arquivo de registro XML foi gerado por dois testes, `testFailure` e `testError`, de uma classe de caso de teste chamada `FailureErrorTest` e mostra como falhas e erros são denotadas.

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

/home/sb/FailureErrorTest.php:8
</failure>
    </testcase>
    <testcase name="testError"
              class="FailureErrorTest"
              file="/home/sb/FailureErrorTest.php"
              line="11"
              assertions="0"
              time="0.008288">
      <error type="Exception">testError(FailureErrorTest)
Exception:

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

13.2 Cobertura de Código (XML)

O formato XML para registro de informação de cobertura de código produzido pelo PHPUnit é amplamente baseado naquele usado pelo Clover. O seguinte exemplo mostra o arquivo de registro XML gerado para os testes em `BankAccountTest`:

```
<?xml version="1.0" encoding="UTF-8"?>
<coverage generated="1184835473" phpunit="3.6.0">
  <project name="BankAccountTest" timestamp="1184835473">
    <file name="/home/sb/BankAccount.php">
      <class name="BankAccountException">
        <metrics methods="0" coveredmethods="0" statements="0"
                 coveredstatements="0" elements="0" coveredelements="0"/>
      </class>
      <class name="BankAccount">
        <metrics methods="4" coveredmethods="4" statements="13"
                 coveredstatements="5" elements="17" coveredelements="9"/>
      </class>
```

(continues on next page)

(continuação da página anterior)

```
      <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>
```

13.3 Cobertura de Código (TEXT)

Saída de cobertura de código legível para linha-de-comando ou arquivo de texto. O objetivo deste formato de saída é fornecer uma rápida visão geral de cobertura enquanto se trabalha em um pequeno grupo de classes. Para projetos maiores esta saída pode ser útil para conseguir uma rápida visão geral da cobertura do projeto ou quando usado com a funcionalidade `--filter`. Quando usada da linha-de-comando escrevendo para `php://stdout` isso vai honrar a configuração `--colors`. Escrever na saída padrão é a opção padrão quando usado a partir da linha-de-comando. Por padrão isso só vai mostrar arquivos que tenham pelo menos uma linha coberta. Isso só pode ser alterado através da opção de configuração xml `showUncoveredFiles`. Veja *Registrando*. Por padrão todos arquivos e seus status de cobertura são mostrados no relatório detalhado. Isso pode ser alterado através da opção de configuração xml `showOnlySummary`.

CAPÍTULO 14

Estendendo o PHPUnit

O PHPUnit pode ser estendido de várias formas para facilitar a escrita de testes e personalizar as respostas que você recebe ao executar os testes. Aqui estão pontos de partida comuns para estender o PHPUnit.

14.1 Subclasse PHPUnit\Framework\TestCase

Escreva asserções personalizadas e métodos utilitários em uma subclasse abstrata de `PHPUnit\Framework\TestCase` e derive suas classes de caso de teste dessa classe. Essa é uma das formas mais fáceis de estender o PHPUnit.

14.2 Escreva asserções personalizadas

Ao escrever asserções personalizadas, a melhor prática é seguir a mesma forma que as asserções do próprio PHPUnit são implementadas. Como você pode ver no Example 14.1, o método `assertTrue()` é apenas um invólucro em torno dos métodos `isTrue()` e `assertThat()`: `isTrue()` cria um objeto comparador que é passado para `assertThat()` para avaliação.

Example 14.1: Os métodos assertTrue() e isTrue() da classe PHPUnit_Framework_Assert

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

abstract class PHPUnit_Framework_Assert
{
    // ...

    /**
     * Assevera que uma condição é verdadeira (true).
     *
```

(continues on next page)

(continuação da página anterior)

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

    // ...

    /**
     * Retorna um objeto comparador PHPUnit_Framework_Constraint_IsTrue.
     *
     * @return PHPUnit_Framework_Constraint_IsTrue
     * @since  Método disponível desde a versão 3.3.0
     */
    public static function isTrue()
    {
        return new PHPUnit_Framework_Constraint_IsTrue;
    }

    // ...
}?>
```

O Example 14.2 mostra como `PHPUnit_Framework_Constraint_IsTrue` estende a classe base abstrata para objetos comparadores (ou objetos-restrições), `PHPUnit_Framework_Constraint`.

Example 14.2: A classe PHPUnit_Framework_Constraint_IsTrue

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

class PHPUnit_Framework_Constraint_IsTrue extends PHPUnit_Framework_Constraint
{
    /**
     * Avalia a restrição para o parâmetro $other. Retorna true se a
     * restrição é confirmada, false caso contrário.
     *
     * @param mixed $other Valor ou objeto a avaliar.
     * @return bool
     */
    public function matches($other)
    {
        return $other === true;
    }

    /**
     * Retorna uma representação string da restrição.
     *
     * @return string
     */
    public function toString()
    {
        return 'is true';
    }
}?>
```

O esforço de implementar os métodos `assertTrue()` e `isTrue()` assim como a classe `PHPUnit_Framework_Constraint_IsTrue` rende o benefício de que `assertThat()` automaticamente cuida de avaliar a asserção e escriturar tarefas tais como contá-las para estatísticas. Além disso, o método `isTrue()` pode ser usado como um comparador ao configurar objetos falsificados.

14.3 Implementando PHPUnit\Framework\TestListener

O Example 14.3 mostra uma implementação simples da interface `PHPUnit\Framework\TestListener`.

Example 14.3: Um simples ouvinte de teste

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

class SimpleTestListener implements TestListener
{
    public function addError(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Error while running test '%s'.\n", $test->getName());
    }

    public function addFailure(PHPUnit_Framework_Test $test, PHPUnit_Framework_
↪AssertionFailedError $e, $time)
    {
        printf("Test '%s' failed.\n", $test->getName());
    }

    public function addIncompleteTest(PHPUnit_Framework_Test $test, Exception $e,
↪$time)
    {
        printf("Test '%s' is incomplete.\n", $test->getName());
    }

    public function addRiskyTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Test '%s' is deemed risky.\n", $test->getName());
    }

    public function addSkippedTest(PHPUnit_Framework_Test $test, Exception $e, $time)
    {
        printf("Test '%s' has been skipped.\n", $test->getName());
    }

    public function startTest(PHPUnit_Framework_Test $test)
    {
        printf("Test '%s' started.\n", $test->getName());
    }

    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        printf("Test '%s' ended.\n", $test->getName());
    }

    public function startTestSuite(PHPUnit_Framework_TestSuite $suite)
    {
```

(continues on next page)

(continuação da página anterior)

```
        printf("TestSuite '%s' started.\n", $suite->getName());
    }

    public function endTestSuite(PHPUnit_Framework_TestSuite $suite)
    {
        printf("TestSuite '%s' ended.\n", $suite->getName());
    }
}
?>
```

Example 14.4 mostra como a subclasse da classe abstrata `PHPUnit_Framework_BaseTestListener`, que permite que você especifique apenas os métodos de interface que são interessantes para seu caso de uso, ao fornecer implementações vazias para todos os outros.

Example 14.4: Usando o ouvinte de teste base

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

class ShortTestListener extends PHPUnit_Framework_BaseTestListener
{
    public function endTest(PHPUnit_Framework_Test $test, $time)
    {
        printf("Test '%s' ended.\n", $test->getName());
    }
}
?>
```

Em *Ouvintes de Teste* você pode ver como configurar o PHPUnit para anexar seu ouvinte de teste para a execução do teste.

14.4 Implementando PHPUnit_Framework_Test

A interface `PHPUnit_Framework_Test` é limitada e fácil de implementar. Você pode escrever uma implementação do `PHPUnit_Framework_Test` que é mais simples que `PHPUnit\Framework\TestCase` e que executa *testes guiados por dados*, por exemplo.

O Example 14.5 mostra uma classe de caso de teste guiado por dados que compara valores de um arquivo com Valores Separados por Vírgula (CSV). Cada linha de tal arquivo parece com `foo;bar`, onde o primeiro valor é o qual esperamos e o segundo valor é o real.

Example 14.5: Um teste guiado por dados

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

class DataDrivenTest implements PHPUnit_Framework_Test
{
    private $lines;

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

(continues on next page)

(continuação da página anterior)

```
    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 7.0.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds
```

(continues on next page)

(continuação da página anterior)

```
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.
```

CAPÍTULO 15

Asserções

Esse apêndice lista os vários métodos de asserções que estão disponíveis.

15.1 Uso Estático vs. Não-Estático de Métodos de Asserção

Asserções do PHPUnit são implementadas in PHPUnit\Framework\Assert. PHPUnit\Framework\TestCase herda de PHPUnit\Framework\Assert.

Os métodos de asserção são declarados static e podem ser invocados de qualquer contexto usando PHPUnit\Framework\Assert::assertTrue(), por exemplo, ou usando $this->assertTrue() ou self::assertTrue(), por exemplo, em uma classe que extende PHPUnit\Framework\TestCase.

Na verdade, você pode usar invólucros de funções globais tal como assertTrue() em qualquer contexto (incluindo classes que extendem PHPUnit\Framework\TestCase) quando você (manualmente) inclui o arquivo de código-fonte `src/Framework/Assert/Functions.php` que vem com PHPUnit.

Uma questão comum, especialmente de desenvolvedores novos em PHPUnit, é se usar $this->assertTrue() ou self::assertTrue(), por exemplo, é "a forma certa" de invocar uma asserção. A resposta rápida é: não existe forma certa. E também não existe forma errada. Isso é mais uma questão de preferência.

Para a maioria das pessoas "se sente melhor" usar $this->assertTrue() porque o método de teste é invocado em um objeto de teste. O fato de que os métodos de asserção são declarados static permite (re)usá-los de fora do escopo de um objeto de teste. Por último, os invólucros de funções globais permitem desenvolvedores digitarem menos caracteres (assertTrue() ao invés de $this->assertTrue() ou self::assertTrue()).

15.2 assertArrayHasKey()

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

Reporta um erro identificado pela `$message` se o `$array` não contém a `$key`.

`assertArrayNotHasKey()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.1: Utilização de assertArrayHasKey()

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

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

```
$ phpunit ArrayHasKeyTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ArrayHasKeyTest.php:6

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

15.3 assertClassHasAttribute()

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

Reporta um erro identificado pela `$message` se `$className::attributeName` não existir.

`assertClassNotHasAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.2: Utilização de assertClassHasAttribute()

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

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

```
$ phpunit ClassHasAttributeTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

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

/home/sb/ClassHasAttributeTest.php:6

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

15.4 assertArraySubset()

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

Reporta um erro identificado pela `$message` se `$array` não contém o `$subset`.

`$strict` é uma flag usada para comparar a identidade de objetos dentro dos arrays.

Example 15.3: Utilização de assertArraySubset()

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

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

```
$ phpunit ArrayHasKeyTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

(continues on next page)

(continuação da página anterior)

```
    )
).

/home/sb/ArraySubsetTest.php:6

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

15.5 assertClassHasStaticAttribute()

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

Reporta um erro identificado pela `$message` se o `$className::attributeName` não existir.

`assertClassNotHasStaticAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.4: Utilização de assertClassHasStaticAttribute()

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

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

```
$ phpunit ClassHasStaticAttributeTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

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

/home/sb/ClassHasStaticAttributeTest.php:6

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

15.6 assertContains()

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

Reporta um erro identificado pela `$message` se o `$needle` não é um elemento de `$haystack`.

`assertNotContains()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeContains()` e `assertAttributeNotContains()` são invólucros convenientes que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como a haystack.

Example 15.5: Utilização de assertContains()

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

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

```
$ phpunit ContainsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsTest::testFailure
Failed asserting that an array contains 4.

/home/sb/ContainsTest.php:6

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

`assertContains(string $needle, string $haystack[, string $message = '', boolean $ignoreCase = FALSE])`

Reporta um erro identificado pela `$message` se `$needle` não é uma substring de `$haystack`.

Se `$ignoreCase` é `true`, o teste irá ser case insensitive.

Example 15.6: Utilização de assertContains()

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

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

```
$ phpunit ContainsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) ContainsTest::testFailure
Failed asserting that 'foobar' contains "baz".

/home/sb/ContainsTest.php:6

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

Example 15.7: Utilização de assertContains() com $ignoreCase

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

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

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

```
$ phpunit ContainsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F.

Time: 0 seconds, Memory: 2.75Mb

There was 1 failure:

1) ContainsTest::testFailure
Failed asserting that 'FooBar' contains "foo".

/home/sb/ContainsTest.php:6

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

15.7 assertContainsOnly()

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

Reporta um erro identificado pela `$message` se `$haystack` não contém somente variáveis do tipo `$type`.

`$isNativeType` é uma flag usada para indicar se `$type` é um tipo nativo do PHP ou não.

`assertNotContainsOnly()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeContainsOnly()` e `assertAttributeNotContainsOnly()` são invólucros convenientes que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como a haystack.

Example 15.8: Utilização de assertContainsOnly()

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

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

```
$ phpunit ContainsOnlyTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContainsOnlyTest.php:6

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

15.8 assertContainsOnlyInstancesOf()

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

Reporta um erro identificado pela `$message` se `$haystack` não contém somente instâncias da classe `$classname`.

Example 15.9: Utilização de assertContainsOnlyInstancesOf()

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

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

```
$ phpunit ContainsOnlyInstancesOfTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/ContainsOnlyInstancesOfTest.php:6

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

15.9 assertCount()

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

Reporta um erro identificado pela `$message` se o número de elementos no `$haystack` não for `$expectedCount`.

`assertNotCount()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.10: Utilização de assertCount()

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

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

(continues on next page)

(continuação da página anterior)

```
}
?>
```

```
$ phpunit CountTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) CountTest::testFailure
Failed asserting that actual size 1 matches expected size 0.

/home/sb/CountTest.php:6

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

15.10 assertDirectoryExists()

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

Reporta um erro identificado pela `$message` se o diretório especificado por `$directory` não existir.

`assertDirectoryNotExists()` é o inverso dessa asserção e recebe os mesmos argumentos.

Example 15.11: Utilização de assertDirectoryExists()

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

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

```
$ phpunit DirectoryExistsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryExistsTest::testFailure
Failed asserting that directory "/path/to/directory" exists.
```

(continues on next page)

(continuação da página anterior)

```
/home/sb/DirectoryExistsTest.php:6

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

15.11 assertDirectoryIsReadable()

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

Reporta um erro identificado pela `$message` se o diretório especificado por `$directory` não é um diretório ou não é legível.

`assertDirectoryNotIsReadable()` é o inverso dessa asserção e recebe os mesmos argumentos.

Example 15.12: Utilização de assertDirectoryIsReadable()

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

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

```
$ phpunit DirectoryIsReadableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryIsReadableTest::testFailure
Failed asserting that "/path/to/directory" is readable.

/home/sb/DirectoryIsReadableTest.php:6

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

15.12 assertDirectoryIsWritable()

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

Reporta um erro identificado pela `$message` se o diretório especificado por `$directory` não é um diretório ou não é gravável.

`assertDirectoryNotIsWritable()` é o inverso dessa asserção e recebe os mesmos argumentos.

Example 15.13: Utilização de assertDirectoryIsWritable()

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

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

```
$ phpunit DirectoryIsWritableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) DirectoryIsWritableTest::testFailure
Failed asserting that "/path/to/directory" is writable.

/home/sb/DirectoryIsWritableTest.php:6

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

15.13 assertEmpty()

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

Reporta um erro identificado pela `$message` se `$actual` não está vazio.

`assertNotEmpty()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeEmpty()` e `assertAttributeNotEmpty()` são invólucros convenientes que podem ser aplicados para um atributo `public`, `protected`, ou `private` de uma classe ou objeto.

Example 15.14: Utilização de assertEmpty()

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

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

```
$ phpunit EmptyTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

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

/home/sb/EmptyTest.php:6

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

15.14 assertEqualXMLStructure()

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

Reporta um erro identificado pela `$message` se a estrutura XML do DOMElement no `$actualElement` não é igual a estrutura XML do DOMElement no `$expectedElement`.

Example 15.15: Utilização de assertEqualXMLStructure()

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

class EqualXMLStructureTest extends TestCase
{
    public function testFailureWithDifferentNodeNames()
    {
        $expected = new DOMElement('foo');
        $actual = new DOMElement('bar');

        $this->assertEqualXMLStructure($expected, $actual);
    }

    public function testFailureWithDifferentNodeAttributes()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo bar="true" />');

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

        $this->assertEqualXMLStructure(
          $expected->firstChild, $actual->firstChild, true
        );
    }

    public function testFailureWithDifferentChildrenCount()
    {
```

(continues on next page)

(continuação da página anterior)

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

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

        $this->assertEqualXMLStructure(
          $expected->firstChild, $actual->firstChild
        );
    }

    public function testFailureWithDifferentChildren()
    {
        $expected = new DOMDocument;
        $expected->loadXML('<foo><bar/><bar/><bar/></foo>');

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

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

```
$ phpunit EqualXMLStructureTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

FFFF

Time: 0 seconds, Memory: 5.75Mb

There were 4 failures:

1) EqualXMLStructureTest::testFailureWithDifferentNodeNames
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'foo'
+'bar'

/home/sb/EqualXMLStructureTest.php:9

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

/home/sb/EqualXMLStructureTest.php:22

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

/home/sb/EqualXMLStructureTest.php:35
```

(continues on next page)

(continuação da página anterior)

```
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.
```

15.15 assertEquals()

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

Reporta um erro identificado pela `$message` se as variáveis `$expected` e `$actual` não são iguais.

`assertNotEquals()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeEquals()` e `assertAttributeNotEquals()` são invólucros convenientes que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.16: Utilização de assertEquals()

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

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

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

    public function testFailure3()
    {
        $this->assertEquals("foo\nbar\nbaz\n", "foo\nbah\nbaz\n");
    }
}
?>
```

```
$ phpunit EqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

FFF

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

(continues on next page)

(continuação da página anterior)

```
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.
```

Comparações mais especializadas são usadas para tipos de argumentos específicos para `$expected` e `$actual`, veja abaixo.

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

Reporta um erro identificado pela `$message` se os dois floats `$expected` e `$actual` não estão dentro do `$delta` um do outro.

Por favor, leia "O Que Cada Cientista da Computação Deve Saber Sobre Aritmética de Ponto Flutuante" para entender por que `$delta` é necessário.

Example 15.17: Utilização de assertEquals() with floats

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

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

(continues on next page)

(continuação da página anterior)

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

```
$ phpunit EqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

.F

Time: 0 seconds, Memory: 5.75Mb

There was 1 failure:

1) EqualsTest::testFailure
Failed asserting that 1.1 matches expected 1.0.

/home/sb/EqualsTest.php:11

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

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

Reporta um erro identificado pela `$message` se a forma canonical não-comentada dos documentos XML representada pelos objetos DOMDocument `$expected` e `$actual` não são iguais.

Example 15.18: Utilização de assertEquals() com objetos DOMDocument

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

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

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

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

```
$ phpunit EqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F
```

(continues on next page)

(continuação da página anterior)

```
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 = ''])`

Reporta um erro identificado pela `$message` se os objetos `$expected` e `$actual` não tem valores de atributos iguais.

Example 15.19: Utilização de assertEquals() com objetos

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

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

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

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

```
$ phpunit EqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

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

(continues on next page)

(continuação da página anterior)

```
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 = ''])`

Reporta um erro identificado pela `$message` se os arrays `$expected` e `$actual` não são iguais.

Example 15.20: Utilização de assertEquals() com arrays

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

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

```
$ phpunit EqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

(continues on next page)

(continuação da página anterior)

```
+    2 => 'd'
 )

/home/sb/EqualsTest.php:6

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

15.16 assertFalse()

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

Reporta um erro identificado pela `$message` se a `$condition` é `true`.

`assertNotFalse()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.21: Utilização de assertFalse()

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

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

```
$ phpunit FalseTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/FalseTest.php:6

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

15.17 assertFileEquals()

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

Reporta um erro identificado pela `$message` se o arquivo especificado pelo `$expected` não tem o mesmo conteúdo que o arquivo especificado pelo `$actual`.

`assertFileNotEquals()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.22: Utilização de assertFileEquals()

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

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

```
$ phpunit FileEqualsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/FileEqualsTest.php:6

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

15.18 assertFileExists()

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

Reporta um erro identificado pela `$message` se o arquivo especificado pelo `$filename` não existir.

`assertFileNotExists()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.23: Utilização de assertFileExists()

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

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

(continues on next page)

(continuação da página anterior)

```
        $this->assertFileExists('/path/to/file');
    }
}
?>
```

```
$ phpunit FileExistsTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) FileExistsTest::testFailure
Failed asserting that file "/path/to/file" exists.

/home/sb/FileExistsTest.php:6

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

15.19 assertFileIsReadable()

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

Reporta um erro identificado pela `$message` se o arquivo especificado por `$filename` não é um arquivo ou não é legível.

`assertFileNotIsReadable()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.24: Usage of assertFileIsReadable()

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

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

```
$ phpunit FileIsReadableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:
```

(continues on next page)

(continuação da página anterior)

```
1) FileIsReadableTest::testFailure
Failed asserting that "/path/to/file" is readable.

/home/sb/FileIsReadableTest.php:6

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

15.20 assertFileIsWritable()

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

Reporta um erro identificado pela `$message` se o arquivo especificado por `$filename` não é um arquivo ou não é gravável.

`assertFileNotIsWritable()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.25: Usage of assertFileIsWritable()

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

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

```
$ phpunit FileIsWritableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) FileIsWritableTest::testFailure
Failed asserting that "/path/to/file" is writable.

/home/sb/FileIsWritableTest.php:6

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

15.21 assertGreaterThan()

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

Reporta um erro identificado pela `$message` se o valor de `$actual` não é maior que o valor de `$expected`.

`assertAttributeGreaterThan()` é um invólucro conveniente que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.26: Utilização de assertGreaterThan()

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

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

```
$ phpunit GreaterThanTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/GreaterThanTest.php:6

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

15.22 assertGreaterThanOrEqual()

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

Reporta um erro identificado pela `$message` se o valor de `$actual` não é maior ou igual ao valor de `$expected`.

`assertAttributeGreaterThanOrEqual()` é um invólucro conveniente que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.27: Utilização de assertGreaterThanOrEqual()

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

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

(continues on next page)

(continuação da página anterior)

```
}
?>
```

```
$ phpunit GreaterThanOrEqualTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) GreatThanOrEqualTest::testFailure
Failed asserting that 1 is equal to 2 or is greater than 2.

/home/sb/GreaterThanOrEqualTest.php:6

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

15.23 assertInfinite()

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

Reporta um erro identificado pela `$message` se `$variable` não é `INF`.

`assertFinite()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.28: Utilização de assertInfinite()

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

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

```
$ phpunit InfiniteTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) InfiniteTest::testFailure
Failed asserting that 1 is infinite.
```

(continues on next page)

(continuação da página anterior)

```
/home/sb/InfiniteTest.php:6

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

15.24 assertInstanceOf()

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

Reporta um erro identificado pela `$message` se `$actual` não é uma instância de `$expected`.

`assertNotInstanceOf()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeInstanceOf()` e `assertAttributeNotInstanceOf()` são invólucros convenientes que podem ser aplicados a um atributo `public`, `protected`, ou `private` de uma classe ou objeto.

Example 15.29: Utilização de assertInstanceOf()

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

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

```
$ phpunit InstanceOfTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/InstanceOfTest.php:6

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

15.25 assertInternalType()

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

Reporta um erro identificado pela `$message` se `$actual` não é do tipo `$expected`.

`assertNotInternalType()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeInternalType()` e `assertAttributeNotInternalType()` são invólucros convenientes que podem aplicados a um atributo `public`, `protected`, ou `private` de uma classe ou objeto.

Example 15.30: Utilização de assertInternalType()

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

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

```
$ phpunit InternalTypeTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

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.
```

15.26 assertIsReadable()

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

Reporta um erro identificado pela `$message` se o arquivo ou diretório especificado por `$filename` não é legível.

`assertNotIsReadable()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.31: Utilização de assertIsReadable()

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

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

(continues on next page)

(continuação da página anterior)

```
}
?>
```

```
$ phpunit IsReadableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IsReadableTest::testFailure
Failed asserting that "/path/to/unreadable" is readable.

/home/sb/IsReadableTest.php:6

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

15.27 assertIsWritable()

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

Reporta um erro especificado pela `$message` se o arquivo ou diretório especificado por `$filename` não é gravável.

`assertNotIsWritable()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.32: Utilização de assertIsWritable()

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

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

```
$ phpunit IsWritableTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb

There was 1 failure:

1) IsWritableTest::testFailure
Failed asserting that "/path/to/unwritable" is writable.
```

(continues on next page)

(continuação da página anterior)

```
/home/sb/IsWritableTest.php:6

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

15.28 assertJsonFileEqualsJsonFile()

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

Reporta um erro identificado pela `$message` se o valor de `$actualFile` não combina com o valor de `$expectedFile`.

Example 15.33: Utilização de assertJsonFileEqualsJsonFile()

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

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

```
$ phpunit JsonFileEqualsJsonFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/JsonFileEqualsJsonFileTest.php:5

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

15.29 assertJsonStringEqualsJsonFile()

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

Reporta um erro identificado pela `$message` se o valor de `$actualJson` não combina com o valor de `$expectedFile`.

Example 15.34: Utilização de assertJsonStringEqualsJsonFile()

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

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

```
$ phpunit JsonStringEqualsJsonFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonFile::testFailure
Failed asserting that '{"Mascot":"ux"}' matches JSON string "{"Mascott":"Tux"}".

/home/sb/JsonStringEqualsJsonFileTest.php:5

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

15.30 assertJsonStringEqualsJsonString()

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

Reporta um erro identificado pela `$message` se o valor de `$actualJson` não combina com o valor de `$expectedJson`.

Example 15.35: Utilização de assertJsonStringEqualsJsonString()

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

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

(continues on next page)

(continuação da página anterior)

```
            json_encode(['Mascot' => 'ux'])
        );
    }
}
?>
```

```
$ phpunit JsonStringEqualsJsonStringTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) JsonStringEqualsJsonStringTest::testFailure
Failed asserting that two objects are equal.
--- Expected
+++ Actual
@@ @@
 stdClass Object (
 -    'Mascot' => 'Tux'
 +    'Mascot' => 'ux'
)

/home/sb/JsonStringEqualsJsonStringTest.php:5

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

15.31 assertLessThan()

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

Reporta um erro identificado pela `$message` se o valor de `$actual` não é menor que o valor de `$expected`.

`assertAttributeLessThan()` é um invólucro conveniente que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.36: Utilização de assertLessThan()

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

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

```
$ phpunit LessThanTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) LessThanTest::testFailure
Failed asserting that 2 is less than 1.

/home/sb/LessThanTest.php:6

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

15.32 assertLessThanOrEqual()

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

Reporta um erro identificado pela `$message` se o valor de `$actual` não é menor ou igual ao valor de `$expected`.

`assertAttributeLessThanOrEqual()` é um invólucro conveniente que usa um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.37: Utilização de assertLessThanOrEqual()

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

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

```
$ phpunit LessThanOrEqualTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) LessThanOrEqualTest::testFailure
Failed asserting that 2 is equal to 1 or is less than 1.

/home/sb/LessThanOrEqualTest.php:6
```

(continues on next page)

(continuação da página anterior)

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

15.33 assertNan()

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

Reporta um erro identificado pela `$message` se `$variable` não é `NAN`.

Example 15.38: Usage of assertNan()

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

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

```
$ phpunit NanTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NanTest::testFailure
Failed asserting that 1 is nan.

/home/sb/NanTest.php:6

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

15.34 assertNull()

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

Reporta um erro identificado pela `$message` se o `$variable` não é `null`.

`assertNotNull()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.39: Utilização de assertNull()

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

(continues on next page)

(continuação da página anterior)

```
class NullTest extends TestCase
{
    public function testFailure()
    {
        $this->assertNull('foo');
    }
}
?>
```

```
$ phpunit NotNullTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) NullTest::testFailure
Failed asserting that 'foo' is null.

/home/sb/NotNullTest.php:6

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

15.35 assertObjectHasAttribute()

`assertObjectHasAttribute(string $attributeName, object $object[, string $message = ''])`

Reporta um erro identificado pela `$message` se `$object->attributeName` não existir.

`assertObjectNotHasAttribute()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.40: Utilização de assertObjectHasAttribute()

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

class ObjectHasAttributeTest extends TestCase
{
    public function testFailure()
    {
        $this->assertObjectHasAttribute('foo', new stdClass);
    }
}
?>
```

```
$ phpunit ObjectHasAttributeTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F
```

(continues on next page)

(continuação da página anterior)

```
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.
```

15.36 assertRegExp()

`assertRegExp(string $pattern, string $string[, string $message = ''])`

Reporta um erro identificado pela `$message` se `$string` não combina com a expressão regular `$pattern`.

`assertNotRegExp()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.41: Utilização de assertRegExp()

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

class RegExpTest extends TestCase
{
    public function testFailure()
    {
        $this->assertRegExp('/foo/', 'bar');
    }
}
?>
```

```
$ phpunit RegExpTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) RegExpTest::testFailure
Failed asserting that 'bar' matches PCRE pattern "/foo/".

/home/sb/RegExpTest.php:6

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

15.37 assertStringMatchesFormat()

`assertStringMatchesFormat(string $format, string $string[, string $message = ''])`

Reporta um erro identificado pela `$message` se a `$string` não combina com a string `$format`.

`assertStringNotMatchesFormat()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.42: Utilização de assertStringMatchesFormat()

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

class StringMatchesFormatTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormat('%i', 'foo');
    }
}
?>
```

```
$ phpunit StringMatchesFormatTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringMatchesFormatTest::testFailure
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+$/s".

/home/sb/StringMatchesFormatTest.php:6

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

A string de formato pode conter os seguintes substitutos (placeholders):

- `%e`: Representa um separador de diretório, por exemplo `/` no Linux.
- `%s`: Um ou mais de qualquer coisa (caractere ou espaço em branco) exceto o caractere de fim de linha.
- `%S`: Zero ou mais de qualquer coisa (caractere ou espaço em branco) exceto o caractere de fim de linha.
- `%a`: Um ou mais de qualquer coisa (caractere ou espaço em branco) incluindo o caractere de fim de linha.
- `%A`: Zero ou mais de qualquer coisa (caractere ou espaço em branco) incluindo o caractere de fim de linha.
- `%w`: Zero ou mais caracteres de espaços em branco.
- `%i`: Um valor inteiro assinado, por exemplo `+3142`, `-3142`.
- `%d`: Um valor inteiro não-assinado, por exemplo `123456`.
- `%x`: Um ou mais caracteres hexadecimais. Isto é, caracteres na classe `0-9`, `a-f`, `A-F`.
- `%f`: Um número de ponto flutuante, por exemplo: `3.142`, `-3.142`, `3.142E-10`, `3.142e+10`.

- `%c`: Um único caractere de qualquer ordem.

15.38 assertStringMatchesFormatFile()

`assertStringMatchesFormatFile(string $formatFile, string $string[, string $message = ''])`

Reporta um erro identificado pela `$message` se a `$string` não combina com o conteúdo do `$formatFile`.

`assertStringNotMatchesFormatFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.43: Utilização de assertStringMatchesFormatFile()

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

class StringMatchesFormatFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringMatchesFormatFile('/path/to/expected.txt', 'foo');
    }
}
?>
```

```
$ phpunit StringMatchesFormatFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringMatchesFormatFileTest::testFailure
Failed asserting that 'foo' matches PCRE pattern "/^[+-]?\d+
$/s".

/home/sb/StringMatchesFormatFileTest.php:6

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

15.39 assertSame()

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

Reporta um erro identificado pela `$message` se as variáveis `$expected` e `$actual` não tem o mesmo tipo e valor.

`assertNotSame()` é o inverso desta asserção e recebe os mesmos argumentos.

`assertAttributeSame()` e `assertAttributeNotSame()` são invólucros convenientes que usam um atributo `public`, `protected`, ou `private` de uma classe ou objeto como o valor atual.

Example 15.44: Utilização de assertSame()

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame('2204', 2204);
    }
}
?>
```

```
$ phpunit SameTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) SameTest::testFailure
Failed asserting that 2204 is identical to '2204'.

/home/sb/SameTest.php:6

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

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

Reporta um erro identificado pela `$message` se as variáveis `$expected` e `$actual` não referenciam o mesmo objeto.

Example 15.45: Utilização de assertSame() com objetos

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

class SameTest extends TestCase
{
    public function testFailure()
    {
        $this->assertSame(new stdClass, new stdClass);
    }
}
?>
```

```
$ phpunit SameTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 4.75Mb
```

(continues on next page)

(continuação da página anterior)

```
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.
```

15.40 assertStringEndsWith()

`assertStringEndsWith(string $suffix, string $string[, string $message = ''])`

Reporta um erro identificado pela `$message` se a `$string` não termina com `$suffix`.

`assertStringEndsNotWith()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.46: Utilização de assertStringEndsWith()

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

class StringEndsWithTest extends TestCase
{
    public function testFailure()
    {
        $this->assertStringEndsWith('suffix', 'foo');
    }
}
?>
```

```
$ phpunit StringEndsWithTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 1 second, Memory: 5.00Mb

There was 1 failure:

1) StringEndsWithTest::testFailure
Failed asserting that 'foo' ends with "suffix".

/home/sb/StringEndsWithTest.php:6

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

15.41 assertStringEqualsFile()

`assertStringEqualsFile(string $expectedFile, string $actualString[, string $message = ''])`

Reporta um erro identificado pela `$message` se o arquivo especificado por `$expectedFile` não tem `$actualString` como seu conteúdo.

`assertStringNotEqualsFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.47: Utilização de assertStringEqualsFile()

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

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

```
$ phpunit StringEqualsFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

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

/home/sb/StringEqualsFileTest.php:6

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

15.42 assertStringStartsWith()

`assertStringStartsWith(string $prefix, string $string[, string $message = ''])`

Reporta um erro identificado pela `$message` se a `$string` não inicia com `$prefix`.

`assertStringStartsNotWith()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.48: Utilização de assertStringStartsWith()

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

class StringStartsWithTest extends TestCase
```

(continues on next page)

(continuação da página anterior)

```
{
    public function testFailure()
    {
        $this->assertStringStartsWith('prefix', 'foo');
    }
}
?>
```

```
$ phpunit StringStartsWithTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) StringStartsWithTest::testFailure
Failed asserting that 'foo' starts with "prefix".

/home/sb/StringStartsWithTest.php:6

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

15.43 assertThat()

Asserções mais complexas podem ser formuladas usando as classes `PHPUnit_Framework_Constraint`. Elas podem ser avaliadas usando o método `assertThat()`. Example 15.49 mostra como as restrições `logicalNot()` e `equalTo()` podem ser usadas para expressar a mesma asserção como `assertNotEquals()`.

`assertThat(mixed $value, PHPUnit_Framework_Constraint $constraint[, $message = ''])`

Reporta um erro identificado pela `$message` se o‘‘$value‘‘ não combina com a `$constraint`.

Example 15.49: Utilização de assertThat()

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

class BiscuitTest extends TestCase
{
    public function testEquals()
    {
        $theBiscuit = new Biscuit('Ginger');
        $myBiscuit  = new Biscuit('Ginger');

        $this->assertThat(
          $theBiscuit,
          $this->logicalNot(
            $this->equalTo($myBiscuit)
          )
        );
```

(continues on next page)

(continuação da página anterior)

```
    }
}
?>
```

Table 15.1 mostra as classes `PHPUnit_Framework_Constraint` disponíveis.

Restrição
`PHPUnit_Framework_Constraint_Attribute attribute(PHPUnit_Framework_Constraint $constraint`
`PHPUnit_Framework_Constraint_IsAnything anything()`
`PHPUnit_Framework_Constraint_ArrayHasKey arrayHasKey(mixed $key)`
`PHPUnit_Framework_Constraint_TraversableContains contains(mixed $value)`
`PHPUnit_Framework_Constraint_TraversableContainsOnly containsOnly(string $type)`
`PHPUnit_Framework_Constraint_TraversableContainsOnly containsOnlyInstancesOf(string $class`
`PHPUnit_Framework_Constraint_IsEqual equalTo($value, $delta = 0, $maxDepth = 10)`
`PHPUnit_Framework_Constraint_Attribute attributeEqualTo($attributeName, $value, $delta =`
`PHPUnit_Framework_Constraint_DirectoryExists directoryExists()`
`PHPUnit_Framework_Constraint_FileExists fileExists()`
`PHPUnit_Framework_Constraint_IsReadable isReadable()`
`PHPUnit_Framework_Constraint_IsWritable isWritable()`
`PHPUnit_Framework_Constraint_GreaterThan greaterThan(mixed $value)`
`PHPUnit_Framework_Constraint_Or greaterThanOrEqual(mixed $value)`
`PHPUnit_Framework_Constraint_ClassHasAttribute classHasAttribute(string $attributeName)`
`PHPUnit_Framework_Constraint_ClassHasStaticAttribute classHasStaticAttribute(string $attr`
`PHPUnit_Framework_Constraint_ObjectHasAttribute hasAttribute(string $attributeName)`
`PHPUnit_Framework_Constraint_IsIdentical identicalTo(mixed $value)`
`PHPUnit_Framework_Constraint_IsFalse isFalse()`
`PHPUnit_Framework_Constraint_IsInstanceOf isInstanceOf(string $className)`
`PHPUnit_Framework_Constraint_IsNull isNull()`
`PHPUnit_Framework_Constraint_IsTrue isTrue()`
`PHPUnit_Framework_Constraint_IsType isType(string $type)`
`PHPUnit_Framework_Constraint_LessThan lessThan(mixed $value)`
`PHPUnit_Framework_Constraint_Or lessThanOrEqual(mixed $value)`
`logicalAnd()`
`logicalNot(PHPUnit_Framework_Constraint $constraint)`
`logicalOr()`
`logicalXor()`
`PHPUnit_Framework_Constraint_PCREMatch matchesRegularExpression(string $pattern)`
`PHPUnit_Framework_Constraint_StringContains stringContains(string $string, bool $case)`
`PHPUnit_Framework_Constraint_StringEndsWith stringEndsWith(string $suffix)`
`PHPUnit_Framework_Constraint_StringStartsWith stringStartsWith(string $prefix)`

15.44 assertTrue()

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

Reporta um erro identificado pela `$message` se a `$condition` é `false`.

`assertNotTrue()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.50: Utilização de assertTrue()

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

class TrueTest extends TestCase
{
    public function testFailure()
    {
        $this->assertTrue(false);
    }
}
?>
```

```
$ phpunit TrueTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

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

/home/sb/TrueTest.php:6

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

15.45 assertXmlFileEqualsXmlFile()

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

Reporta um erro identificado pela `$message` se o documento XML em `$actualFile` não é igual ao documento XML em `$expectedFile`.

`assertXmlFileNotEqualsXmlFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.51: Utilização de assertXmlFileEqualsXmlFile()

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

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

```
$ phpunit XmlFileEqualsXmlFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.25Mb

There was 1 failure:

1) XmlFileEqualsXmlFileTest::testFailure
Failed asserting that two DOM documents are equal.
--- Expected
+++ Actual
@@ @@
 <?xml version="1.0"?>
 <foo>
-  <bar/>
+  <baz/>
 </foo>

/home/sb/XmlFileEqualsXmlFileTest.php:7

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

15.46 assertXmlStringEqualsXmlFile()

`assertXmlStringEqualsXmlFile(string $expectedFile, string $actualXml[, string $message = ''])`

Reporta um erro identificado pela `$message` se o documento XML em `$actualXml` não é igual ao documento XML em `$expectedFile`.

`assertXmlStringNotEqualsXmlFile()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.52: Utilização de assertXmlStringEqualsXmlFile()

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

class XmlStringEqualsXmlFileTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlFile(
          '/home/sb/expected.xml', '<foo><baz/></foo>');
    }
}
?>
```

```
$ phpunit XmlStringEqualsXmlFileTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F
```

(continues on next page)

(continuação da página anterior)

```
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.
```

15.47 assertXmlStringEqualsXmlString()

`assertXmlStringEqualsXmlString(string $expectedXml, string $actualXml[, string $message = ''])`

Reporta um erro identificado pela `$message` se o documento XML em `$actualXml` não é igual ao documento XML em `$expectedXml`.

`assertXmlStringNotEqualsXmlString()` é o inverso desta asserção e recebe os mesmos argumentos.

Example 15.53: Utilização de assertXmlStringEqualsXmlString()

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

class XmlStringEqualsXmlStringTest extends TestCase
{
    public function testFailure()
    {
        $this->assertXmlStringEqualsXmlString(
          '<foo><bar/></foo>', '<foo><baz/></foo>');
    }
}
?>
```

```
$ phpunit XmlStringEqualsXmlStringTest
PHPUnit 7.0.0 by Sebastian Bergmann and contributors.

F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:
```

(continues on next page)

(continuação da página anterior)

```
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.
```

CAPÍTULO 16

Anotações

Uma anotação é uma forma especial de metadados sintáticos que podem ser adicionados ao código-fonte de algumas linguagens de programação. Enquanto PHP não tem um recurso de linguagem dedicado a anotação de código-fonte, o uso de tags como `@annotation arguments` em bloco de documentação tem sido estabelecido na comunidade PHP para anotar o código-fonte. Os blocos de documentação PHP são reflexivos: eles podem ser acessados através do método `getDocComment()` da API Reflection a nível de função, classe, método e atributo. Aplicações como o PHPUnit usam essa informação em tempo de execução para configurar seu comportamento.

Note

Um comentário de documentação em PHP deve começar com `/**` e terminar com `*/`. Anotações em algum outro estilo de comentário serão ignoradas.

Este apêndice mostra todas as variedades de anotações suportadas pelo PHPUnit.

16.1 @author

A anotação `@author` é um apelido para a anotação `@group` (veja *@group*) e permite filtrar os testes baseado em seus autores.

16.2 @after

A anotação `@after` pode ser usada para especificar métodos que devem ser chamados depois de cada método em uma classe de caso de teste.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
```

(continues on next page)

(continuação da página anterior)

```
    /**
     * @after
     */
    public function tearDownSomeFixtures()
    {
        // ...
    }

    /**
     * @after
     */
    public function tearDownSomeOtherFixtures()
    {
        // ...
    }
}
```

16.3 @afterClass

A anotação `@afterClass` pode ser usada para especificar métodos estáticos que devem ser chamados depois que todos os métodos de teste em uma classe de teste foram executados para limpar ambientes compartilhados.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @afterClass
     */
    public static function tearDownSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @afterClass
     */
    public static function tearDownSomeOtherSharedFixtures()
    {
        // ...
    }
}
```

16.4 @backupGlobals

As operações de cópia de segurança e restauração para variáveis globais podem ser completamente desabilitadas para todos os testes de uma classe de caso de teste como esta:

```
use PHPUnit\Framework\TestCase;

/**
```

(continues on next page)

(continuação da página anterior)

```
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    // ...
}
```

A anotação `@backupGlobals` também pode ser usada a nível de método de teste. Isso permite uma configuração refinada das operações de cópia de segurança e restauração:

```
use PHPUnit\Framework\TestCase;

/**
 * @backupGlobals disabled
 */
class MyTest extends TestCase
{
    /**
     * @backupGlobals enabled
     */
    public function testThatInteractsWithGlobalVariables()
    {
        // ...
    }
}
```

16.5 @backupStaticAttributes

A anotação `@backupStaticAttributes` pode ser usada para copiar todos valores de propriedades estáticas em todas classes declaradas antes de cada teste e restaurá-los depois. Pode ser usado em nível de classe de caso de teste ou método de teste:

```
use PHPUnit\Framework\TestCase;

/**
 * @backupStaticAttributes enabled
 */
class MyTest extends TestCase
{
    /**
     * @backupStaticAttributes disabled
     */
    public function testThatInteractsWithStaticAttributes()
    {
        // ...
    }
}
```

Note

`@backupStaticAttributes` é limitada pela parte interna do PHP e pode causar valores estáticos não intencionais ao persistir e vazar para testes subsequentes em algumas circunstâncias.

Veja *Estado Global* para detalhes.

16.6 @before

A anotação `@before` pode ser usada para especificar métodos que devem ser chamados antes de cada método de teste em uma classe de caso de teste.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @before
     */
    public function setupSomeFixtures()
    {
        // ...
    }

    /**
     * @before
     */
    public function setupSomeOtherFixtures()
    {
        // ...
    }
}
```

16.7 @beforeClass

A anotação `@beforeClass` pode ser usada para especificar métodos estáticos que devem ser chamados antes de quaisquer métodos de teste em uma classe de teste serem executados para criar ambientes compartilahdos.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @beforeClass
     */
    public static function setUpSomeSharedFixtures()
    {
        // ...
    }

    /**
     * @beforeClass
     */
    public static function setUpSomeOtherSharedFixtures()
    {
        // ...
```

(continues on next page)

(continuação da página anterior)

```
    }
}
```

16.8 @codeCoverageIgnore*

As anotações `@codeCoverageIgnore`, `@codeCoverageIgnoreStart` e `@codeCoverageIgnoreEnd` podem ser usadas para excluir linhas de código da análise de cobertura.

Para uso, veja *Ignorando Blocos de Código*.

16.9 @covers

A anotação `@covers` pode ser usada no código de teste para especificar quais métodos um método de teste quer testar:

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

Se fornecida, apenas a informação de cobertura de código para o(s) método(s) especificado(s) será considerada.

Table 16.1 mostra a sintaxe da anotação `@covers`.

Table 16.1: Anotações para especificar quais métodos são cobertos por um teste

Anotação	Descrição
`@covers ClassName::methodName`	Especifica que o método de teste anotado cobre o método especificado.
`@covers ClassName`	Especifica que o método de teste anotado cobre todos os métodos de uma dada classe.
`@covers ClassName<extended>`	Especifica que o método de teste anotado cobre todos os métodos de uma dada classe e sua(s) classe(s) pai(s) e interface(s).
`@covers ClassName::<public>`	Especifica que o método de teste anotado cobre todos os métodos públicos de uma dada classe.
`@covers ClassName::<protected>`	Especifica que o método de teste anotado cobre todos os métodos protegidos de uma dada classe.
`@covers ClassName::<private>`	Especifica que o método de teste anotado cobre todos os métodos privados de uma dada classe.
`@covers ClassName::<!public>`	Especifica que o método de teste anotado cobre todos os métodos que não sejam públicos de uma dada classe.
`@covers ClassName::<!protected>`	Especifica que o método de teste anotado cobre todos os métodos que não sejam protegidos de uma dada classe.
`@covers ClassName::<!private>`	Especifica que o método de teste anotado cobre todos os métodos que não sejam privados de uma dada classe.
`@covers ::functionName`	Especifica que método de teste anotado cobre a função global especificada.

16.10 @coversDefaultClass

A anotação `@coversDefaultClass` pode ser usada para especificar um namespace padrão ou nome de classe. Dessa forma nomes longos não precisam ser repetidos para toda anotação `@covers`. Veja o Example 16.1.

Example 16.1: Usando @coversDefaultClass para encurtar anotações

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

/**
 * @coversDefaultClass \Foo\CoveredClass
 */
class CoversDefaultClassTest extends TestCase
{
    /**
     * @covers ::publicMethod
     */
    public function testSomething()
    {
        $o = new Foo\CoveredClass;
        $o->publicMethod();
    }
}
```

(continues on next page)

(continuação da página anterior)

```
?>
```

16.11 @coversNothing

A anotação `@coversNothing` pode ser usada no código de teste para especificar que nenhuma informação de cobertura de código será registrada para o caso de teste anotado.

Isso pode ser usado para testes de integração. Veja *Um teste que especifica que nenhum método deve ser coberto* para um exemplo.

A anotação pode ser usada nos níveis de classe e de método e vão sobrescrever quaisquer tags `@covers`.

16.12 @dataProvider

Um método de teste pode aceitar argumentos arbitrários. Esses argumentos devem ser fornecidos por um método provedor (`provider()` em *Usando um provedor de dados que retorna um vetor de vetores*). O método provedor de dados a ser usado é especificado usando a anotação `@dataProvider`.

Veja *Provedores de Dados* para mais detalhes.

16.13 @depends

O PHPUnit suporta a declaração de dependências explícitas entre métodos de teste. Tais dependências não definem a ordem em que os métodos de teste devem ser executados, mas permitem o retorno de uma instância do ambiente de teste por um produtor e passá-la aos consumidores dependentes. O *Usando a anotação @depends para expressar dependências* mostra como usar a anotação `@depends` para expressar dependências entre métodos de teste.

Veja *Dependências de Testes* para mais detalhes.

16.14 @expectedException

O *Usando o método expectException()* mostra como usar a anotação `@expectedException` para testar se uma exceção é lançada dentro do código testado.

Veja *Testando Exceções* para mais detalhes.

16.15 @expectedExceptionCode

A anotação `@expectedExceptionCode` em conjunto com a `@expectedException` permite fazer asserções no código de erro de uma exceção lançada, permitindo diminuir uma exceção específica.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
```

(continues on next page)

(continuação da página anterior)

```
     * @expectedException     MyException
     * @expectedExceptionCode 20
     */
    public function testExceptionHasErrorcode20()
    {
        throw new MyException('Some Message', 20);
    }
}
```

Para facilitar o teste e reduzir a duplicação, um atalho pode ser usado para especificar uma constante de classe como um `@expectedExceptionCode` usando a sintaxe "`@expectedExceptionCode ClassName::CONST`".

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException     MyException
     * @expectedExceptionCode MyClass::ERRORCODE
     */
    public function testExceptionHasErrorcode20()
    {
      throw new MyException('Some Message', 20);
    }
}
class MyClass
{
    const ERRORCODE = 20;
}
```

16.16 @expectedExceptionMessage

A anotação `@expectedExceptionMessage` funciona de modo similar a `@expectedExceptionCode` já que lhe permite fazer uma asserção na mensagem de erro de uma exceção.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @expectedException        MyException
     * @expectedExceptionMessage Some Message
     */
    public function testExceptionHasRightMessage()
    {
        throw new MyException('Some Message', 20);
    }
}
```

A mensagem esperada pode ser uma substring de uma Mensagem de exceção. Isso pode ser útil para asseverar apenas que um certo nome ou parâmetro que foi passado é mostrado na exceção e não fixar toda mensagem de exceção no teste.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException        MyException
      * @expectedExceptionMessage broken
      */
     public function testExceptionHasRightMessage()
     {
         $param = "broken";
         throw new MyException('Invalid parameter "'.$param.'".', 20);
     }
}
```

Para facilitar o teste e reduzir duplicação um atalho pode ser usado para especificar uma constante de classe como uma `@expectedExceptionMessage` usando a sintaxe "`@expectedExceptionMessage ClassName::CONST`". Um exemplo pode ser encontrado na seção chamada *@expectedExceptionCode*.

16.17 @expectedExceptionMessageRegExp

A mensagem experada também pode ser especificada como uma expressão regular usando a anotação `@expectedExceptionMessageRegExp`. Isso é útil em situações onde uma substring não é adequada para combinar uma determinada mensagem.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
     /**
      * @expectedException              MyException
      * @expectedExceptionMessageRegExp /Argument \d+ can not be an? \w+/
      */
     public function testExceptionHasRightMessage()
     {
         throw new MyException('Argument 2 can not be an integer');
     }
}
```

16.18 @group

Um teste pode ser marcado como pertencente a um ou mais grupos usando a anotação `@group` desta forma

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @group specification
     */
    public function testSomething()
    {
```

(continues on next page)

(continuação da página anterior)

```
    }

    /**
     * @group regresssion
     * @group bug2204
     */
    public function testSomethingElse()
    {
    }
}
```

Testes podem ser selecionados para execução baseados em grupos usando as opções `--group` e `--exclude-group` do executor de teste em linha-de-comando ou usando as respectivas diretivas do arquivo de configuração XML.

16.19 @large

A anotação `@large` é um apelido para `@group large`.

Se o pacote `PHP_Invoker` estiver instalado e o modo estrito estiver habilitado, um teste grande irá falhar se ele demorar mais de 60 segundos para executar. Esse tempo de espera é configurável através do atributo `timeoutForLargeTests` no arquivo de configuração XML.

16.20 @medium

A anotação `@medium` é um apelido para `@group medium`. Um teste médio não deve depender de um teste marcado como `@large`.

Se o pacote `PHP_Invoker` estiver instalado e o modo estrito estiver habilitado, um teste médio irá falhar se ele demorar mais de 10 segundos para executar. Esse tempo de espera é configurável através do atributo `timeoutForMediumTests` no arquivo de configuração XML.

16.21 @preserveGlobalState

Quando um teste é executado em um processo separado, o PHPUnit tentará preservar o estado global do processo pai serializando todos globais no processo pai e deserializando-os no processo filho. Isso pode causar problemas se o processo pai contém globais que não são serializáveis. Para corrigir isto, você pode prevenir o PHPUnit de preservar o estado global com a anotação `@preserveGlobalState`.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     * @preserveGlobalState disabled
     */
    public function testInSeparateProcess()
    {
        // ...
```

(continues on next page)

(continuação da página anterior)

```
    }
}
```

16.22 @requires

A anotação `@requires` pode ser usada para pular testes quando pré-condições comuns, como a Versão do PHP ou extensões instaladas, não batem.

Uma lista completa de possibilidades e exemplos pode ser encontrada em *Possíveis usos para @requires*

16.23 @runTestsInSeparateProcesses

Indica que todos testes em uma classe de teste devem ser executados em um processo PHP separado.

```
use PHPUnit\Framework\TestCase;

/**
 * @runTestsInSeparateProcesses
 */
class MyTest extends TestCase
{
    // ...
}
```

Nota: Por padrão, o PHPUnit tentará preservar o estado global do processo pai serializando todos globais no processo pai e deserializando-os no processo filho. Isso pode causar problemas se o processo pai contém globais que não são serializáveis. Veja *@preserveGlobalState* para informações de como corrigir isso.

16.24 @runInSeparateProcess

Indica que um teste deve ser executado em um processo PHP separado.

```
use PHPUnit\Framework\TestCase;

class MyTest extends TestCase
{
    /**
     * @runInSeparateProcess
     */
    public function testInSeparateProcess()
    {
        // ...
    }
}
```

Nota: Por padrão, o PHPUnit tentará preservar o estado global do processo pai serializando todos globais no processo pai e deserializando-os no processo filho. Isso pode causar problemas se o processo pai contém globais que não são serializáveis. Veja *@preserveGlobalState* para informações de como corrigir isso.

16.25 @small

A anotação `@small` é um apelido para `@group small`. Um teste pequeno não deve depender de um teste marcado como `@medium` ou `@large`.

Se o pacote `PHP_Invoker` estiver instalado e o modo estrito estiver habilitado, um teste pequeno irá falhar se ele demorar mais de 1 segundo para executar. Esse tempo de espera é configurável através do atributo `timeoutForLargeTests` no arquivo de configuração XML.

Note

Testes precisam ser explicitamente anotados com `@small`, `@medium` ou `@large` para ativar limites de tempo de execução.

16.26 @test

Como uma alternativa para prefixar seus nomes de métodos de teste com `test`, você pode usar a anotação `@test` no Bloco de Documentação do método para marcá-lo como um método de teste.

```
/**
 * @test
 */
public function initialBalanceShouldBe0()
{
    $this->assertEquals(0, $this->ba->getBalance());
}
```

16.27 @testdox

16.28 @ticket

16.29 @uses

A anotação `@uses` especifica o código que irá ser executado por um teste, mas não se destina a ser coberto pelo teste. Um bom exemplo é um objeto valor que é necessário para testar uma unidade de código.

```
/**
 * @covers BankAccount::deposit
 * @uses   Money
 */
public function testMoneyCanBeDepositedInAccount()
```

(continues on next page)

(continuação da página anterior)

```
{
    // ...
}
```

Essa anotação é especialmente útil em modo de cobertura estrita onde código coberto involuntariamente fará um teste falhar. Veja *Cobertura de Código Involuntária* para mais informação sobre modo de cobertura estrita.

CAPÍTULO 17

O Arquivo de Configuração XML

17.1 PHPUnit

Os atributos do elemento `<phpunit>` podem ser usados para configurar a funcionalidade principal do PHPUnit.

```
<phpunit
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/6.3/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"
         stopOnRisky="false"
         testSuiteLoaderClass="PHPUnit_Runner_StandardTestSuiteLoader"
         <!--testSuiteLoaderFile="/path/to/StandardTestSuiteLoader.php"-->
         timeoutForSmallTests="1"
         timeoutForMediumTests="10"
         timeoutForLargeTests="60"
         verbose="false">
  <!-- ... -->
</phpunit>
```

A configuração XML acima corresponde ao comportamento padrão do executor de teste TextUI documentado na *Opções de linha-de-comando*.

Opções adicionais que não estão disponíveis como opções em linha-de-comando são:

`convertErrorsToExceptions`

Por padrão, PHPUnit irá instalar um manipulador de erro que converte os seguintes erros para exceções:

- `E_WARNING`
- `E_NOTICE`
- `E_USER_ERROR`
- `E_USER_WARNING`
- `E_USER_NOTICE`
- `E_STRICT`
- `E_RECOVERABLE_ERROR`
- `E_DEPRECATED`
- `E_USER_DEPRECATED`

Definir `convertErrorsToExceptions` para `false` desabilita esta funcionalidade.

`convertNoticesToExceptions`

Quando definido para `false`, o manipulador de erro instalado pelo `convertErrorsToExceptions` não irá converter os erros `E_NOTICE`, `E_USER_NOTICE`, ou `E_STRICT` para exceções.

`convertWarningsToExceptions`

Quando definido para `false`, o manipulador de erro instalado pelo `convertErrorsToExceptions` não irá converter os erros `E_WARNING` ou `E_USER_WARNING` para exceções.

`forceCoversAnnotation`

A Cobertura de Código só será registrada para testes que usem a anotação `@covers` documentada na *@covers*.

`timeoutForLargeTests`

Se o limite de tempo baseado no tamanho do teste são forçados então esse atributo define o tempo limite para todos os testes marcados como `@large`. Se um teste não completa dentro do tempo limite configurado, irá falhar.

`timeoutForMediumTests`

Se o limite de tempo baseado no tamanho do teste são forçados então esse atributo define o tempo limite para todos os testes marcados como `@medium`. Se um teste não completa dentro do tempo limite configurado, irá falhar.

`timeoutForSmallTests`

Se o limite de tempo baseado no tamanho do teste são forçados então esse atributo define o tempo limite para todos os testes marcados como `@medium` ou `@large`. Se um teste não completa dentro do tempo limite configurado, irá falhar.

17.2 Suítes de Teste

O elemento `<testsuites>` e seu(s) um ou mais filhos `<testsuite>` podem ser usados para compor uma suíte de teste fora das suítes e casos de teste.

```
<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>
```

Usando os atributos `phpVersion` e `phpVersionOperator`, uma versão exigida do PHP pode ser especificada. O exemplo abaixo só vai adicionar os arquivos `/path/to/*Test.php` e `/path/to/MyTest.php` se a versão do PHP for no mínimo 5.3.0.

```
<testsuites>
  <testsuite name="My Test Suite">
    <directory suffix="Test.php" phpVersion="5.3.0" phpVersionOperator=">=">/path/to/
↪files</directory>
    <file phpVersion="5.3.0" phpVersionOperator=">=">/path/to/MyTest.php</file>
  </testsuite>
</testsuites>
```

O atributo `phpVersionOperator` é opcional e padronizado para `>=`.

17.3 Grupos

O elemento `<groups>` e seus filhos `<include>`, `<exclude>`, e `<group>` podem ser usados para selecionar grupos de testes marcados com a anotação `@group` (documentada na *@group*) que (não) deveriam ser executados.

```
<groups>
  <include>
    <group>name</group>
  </include>
  <exclude>
    <group>name</group>
  </exclude>
</groups>
```

A configuração XML acima corresponde a invocar o executor de testes TextUI com as seguintes opções:

- `--group name`
- `--exclude-group name`

17.4 Lista-branca de Arquivos para Cobertura de Código

O elemento `<filter>` e seus filhos podem ser usados para configurar a lista-branca para o relatório de cobertura de código.

```
<filter>
  <whitelist processUncoveredFilesFromWhitelist="true">
    <directory suffix=".php">/path/to/files</directory>
    <file>/path/to/file</file>
    <exclude>
      <directory suffix=".php">/path/to/files</directory>
      <file>/path/to/file</file>
    </exclude>
  </whitelist>
</filter>
```

17.5 Registrando

O elemento `<logging>` e seus filhos `<log>` podem ser usados para configurar o registro da execução de teste.

```
<logging>
  <log type="coverage-html" target="/tmp/report" lowUpperBound="35"
       highLowerBound="70"/>
  <log type="coverage-clover" target="/tmp/coverage.xml"/>
  <log type="coverage-php" target="/tmp/coverage.serialized"/>
  <log type="coverage-text" target="php://stdout" showUncoveredFiles="false"/>
  <log type="junit" target="/tmp/logfile.xml" logIncompleteSkipped="false"/>
  <log type="testdox-html" target="/tmp/testdox.html"/>
  <log type="testdox-text" target="/tmp/testdox.txt"/>
</logging>
```

A configuração XML acima corresponde a invocar o executor de teste TextUI com as seguintes opções:

- `--coverage-html /tmp/report`
- `--coverage-clover /tmp/coverage.xml`
- `--coverage-php /tmp/coverage.serialized`
- `--coverage-text`
- `> /tmp/logfile.txt`
- `--log-junit /tmp/logfile.xml`
- `--testdox-html /tmp/testdox.html`
- `--testdox-text /tmp/testdox.txt`

Os atributos `lowUpperBound`, `highLowerBound`, `logIncompleteSkipped` e `showUncoveredFiles` não possuem opção de execução de teste TextUI equivalente.

- `lowUpperBound`: Porcentagem máxima de cobertura para ser considerado "moderadamente" coberto.
- `highLowerBound`: Porcentagem mínima de cobertura para ser considerado "altamente" coberto.
- `showUncoveredFiles`: Mostra todos arquivos da lista-branca na saída `--coverage-text` não apenas aqueles com informação de cobertura.
- `showOnlySummary`: Mostra somente o resumo na saída `--coverage-text`.

17.6 Ouvintes de Teste

O elemento `<listeners>` e seus filhos `<listener>` podem ser usados para anexar ouvintes adicionais de teste para a execução dos testes.

```
<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>
```

A configuração XML acima corresponde a anexar o objeto `$listener` (veja abaixo) à execução de teste:

```
$listener = new MyListener(
    ['Sebastian'],
    22,
    'April',
    19.78,
    null,
    new stdClass
);
```

17.7 Definindo configurações PHP INI, Constantes e Variáveis Globais

O elemento `<php>` e seus filhos podem ser usados para definir configurações do PHP, constantes e variáveis globais. Também pode ser usado para preceder o `include_path`.

```
<php>
  <includePath>.</includePath>
  <ini name="foo" value="bar"/>
  <const name="foo" value="bar"/>
  <var name="foo" value="bar"/>
  <env name="foo" value="bar"/>
  <post name="foo" value="bar"/>
  <get name="foo" value="bar"/>
  <cookie name="foo" value="bar"/>
  <server name="foo" value="bar"/>
  <files name="foo" value="bar"/>
  <request name="foo" value="bar"/>
</php>
```

A configuração XML acima corresponde ao seguinte código PHP:

```
ini_set('foo', 'bar');
define('foo', 'bar');
$GLOBALS['foo'] = 'bar';
$_ENV['foo'] = 'bar';
$_POST['foo'] = 'bar';
$_GET['foo'] = 'bar';
$_COOKIE['foo'] = 'bar';
$_SERVER['foo'] = 'bar';
$_FILES['foo'] = 'bar';
$_REQUEST['foo'] = 'bar';
```

CAPÍTULO 18

Bibliografia

[Astels2003] David Astels. *Test Driven Development*.

[Beck2002] Kent Beck. *Test Driven Development by Example*.

[Meszaros2007] Gerard Meszaros. *xUnit Test Patterns: Refactoring Test Code*.

CAPÍTULO 19

Direitos autorais

```
Direitos autorais (c) 2005-2018 Sebastian Bergmann.

Este trabalho é licenciado sob a Licença Não-adaptada Creative Commons
Attribution 3.0.

Um resumo da licença é dada abaixo, seguido pelo texto legal
completo.

---------------------------------------------------------------------

Você tem o direito de:

    * Compartilhar -- copiar, distribuir e transmitir o material
    * Adaptar -- adaptar o material

De acordo com os termos seguintes:

Atribuição. Você deve atribuir o material da maneira especificado pelo
autor ou licenciador (mas não de qualquer form que sugira que eles
endossem você ou seu uso ao material).

    * Para qualquer reuso ou distribuição, você deve deixar claro
      os termos da licença deste material. A melhor maneira de fazer isso e com
      um link para esta página web.

    * Qualquer das condições acima podem ser renunciadas se você
      obter permissão de detentor dos direitos autorais.

    * Nada nesta licença prejudica ou restringe os diretos autorais do
      autor.

Sua justa negação e outros direitos não são, de maneira alguma,
afetados pelo citado acima.
```

(continues on next page)

(continuação da página anterior)

```
Esse é um resumo legível do Código Legal (a licença
completa) abaixo.

=============================================================================

Creative Commons Legal Code
Attribution 3.0 Unported

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE
LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN
ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS
INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO
WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS
LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE
WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW
IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND
AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS
LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU
THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
SUCH TERMS AND CONDITIONS.

1. Definitions

   a. "Adaptation" means a work based upon the Work, or upon the
      Work and other pre-existing works, such as a translation,
      adaptation, derivative work, arrangement of music or other
      alterations of a literary or artistic work, or phonogram or
      performance and includes cinematographic adaptations or any
      other form in which the Work may be recast, transformed, or
      adapted including in any form recognizably derived from the
      original, except that a work that constitutes a Collection
      will not be considered an Adaptation for the purpose of this
      License. For the avoidance of doubt, where the Work is a
      musical work, performance or phonogram, the synchronization of
      the Work in timed-relation with a moving image ("synching")
      will be considered an Adaptation for the purpose of this
      License.

   b. "Collection" means a collection of literary or artistic works,
      such as encyclopedias and anthologies, or performances,
      phonograms or broadcasts, or other works or subject matter
      other than works listed in Section 1(f) below, which, by
      reason of the selection and arrangement of their contents,
      constitute intellectual creations, in which the Work is
      included in its entirety in unmodified form along with one or
      more other contributions, each constituting separate and
      independent works in themselves, which together are assembled
      into a collective whole. A work that constitutes a Collection
      will not be considered an Adaptation (as defined above) for
```

(continues on next page)

(continuação da página anterior)

```
   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
```

(continues on next page)

(continuação da página anterior)

```
      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
```

(continues on next page)

(continuação da página anterior)

```
            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
```

(continues on next page)

(continuação da página anterior)

```
      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
```

(continues on next page)

(continuação da página anterior)

```
   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
```

(continues on next page)

(continuação da página anterior)

```
      Performances and Phonograms Treaty of 1996 and the Universal
      Copyright Convention (as revised on July 24, 1971). These
      rights and subject matter take effect in the relevant
      jurisdiction in which the License terms are sought to be
      enforced according to the corresponding provisions of the
      implementation of those treaty provisions in the applicable
      national law. If the standard suite of rights granted under
      applicable copyright law includes additional rights not
      granted under this License, such additional rights are deemed
      to be included in the License; this License is not intended to
      restrict the license of any rights under applicable law.

Creative Commons is not a party to this License, and makes no
warranty whatsoever in connection with the Work. Creative Commons
will not be liable to You or any party on any legal theory for any
damages whatsoever, including without limitation any general,
special, incidental or consequential damages arising in connection
to this license. Notwithstanding the foregoing two (2) sentences,
if Creative Commons has expressly identified itself as the Licensor
hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the
Work is licensed under the CCPL, Creative Commons does not authorize
the use by either party of the trademark "Creative Commons" or any
related trademark or logo of Creative Commons without the prior
written consent of Creative Commons. Any permitted use will be in
compliance with Creative Commons' then-current trademark usage
guidelines, as may be published on its website or otherwise made
available upon request from time to time. For the avoidance of
doubt, this trademark restriction does not form part of this
License.

Creative Commons may be contacted at http://creativecommons.org/.

======================================================================
```